Proceedings of the 1984 MACSYMA USERS' CONFERENCE

GENERAL ELECTRIC

Schenectady, New York
July 23 - 25, 1984

Proceedings of the 1984 MACSYMA USERS' CONFERENCE

GENERAL ELECTRIC

Schenectady, New York
July 23 - 25, 1984

Table of Contents

PHOTOGRAPH THIS SHEET

AD A146847

DTIC ACCESSION NUMBER

II

LEVEL

1

INVENTORY

Proceedings of the 1984 MACSYMA User's Conference

DOCUMENT IDENTIFICATION 23-25 July '84

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR	
NTIS GRA&I	X
DTIC TAB	☐
UNANNOUNCED	☐
JUSTIFICATION	
BY	
DISTRIBUTION /	
AVAILABILITY CODES	
DIST	AVAIL AND/OR SPECIAL
A/1	

DISTRIBUTION STAMP

DTIC
ELECTE
S OCT 22 1984 D
D

DATE ACCESSIONED

DATE RETURNED

84 09 27 006

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NO.

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDAC

DTIC FORM 70A

DOCUMENT PROCESSING SHEET

PREVIOUS EDITION MAY BE USED UNTIL STOCK IS EXHAUSTED.

(20) Abstract:

The purpose of this Third MACSYMA Users' Conference is to provide a forum for the discussion of all areas related to the development and application of MACSYMA and of similar systems. Five years have passed since the last MACSYMA Users' Conference. A much larger and more dispersed community is now using computers for performing symbolic, as well as numeric, mathematical computations. A meeting for the purpose of exchanging information between individuals applying such systems to the solution of scientific and engineering problems is especially timely.

At the time of the previous MACSYMA User's Conferences, MACSYMA was on the MACSYMA Consortium Computer at MIT under the direction of Professor Joel Moses. The ARPANET made access to MACSYMA convenient for a user community geographically distributed from Hawaii to England. The MACSYMA Consortium no longer exists. Most users have their own copies of MACSYMA on VAX's (VAXIMA) or on the Symbolics 3600 (LISP) machine. A few are still using MACSYMA via ARPANET on the MIT-MULTICS.

The First MACSYMA Users' Conference was held at the University of California at Berkeley in 1977 and the Second in Washington, DC in 1979. The Proceedings of these conferences were published by NASA (as NASA Report No. CP-2012) and by MIT Laboratory for Computer Science, respectively.

The early development of MACSYMA at MIT was funded by the Defense Advanced Research Projects Agency. Later sponsors included the Department of Energy, the National Aeronautics and Space Administration, the U.S. Navy, the U.S. Army and the U.S. Air Force.

The Third MACSYMA Users' Conference sponsors include the General Electric Corporation Research and Development Center; Symbolics, Inc.; the U.S. Army Research Office; the Office of Naval Research and the Air Force Office of Scientific Research.

We are grateful to all of those who have made this Conference and the publication of these Proceedings possible including our sponsors, the Organizing Committee, the invited speakers, those who contributed papers and the conference attendees. Very special thanks are due to Dr. Hussain of GE who initiated the conference and contributed in so many ways to its success, to Ellen Golden of Symbolics whose efforts have made the publication of these proceedings possible and to Mickey McGinn and others at GE who attended to the many conference details.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
Proceedings of the 1984 MACSYMA Users Conference

5. TYPE OF REPORT & PERIOD COVERED
Conference Proceedings

6. PERFORMING ORG. REPORT NUMBER
None

7. AUTHOR(s)
Editor: V. Ellen Golden
Co-editor: M. A. Hussain

8. CONTRACT OR GRANT NUMBER(s)
61153N

9. PERFORMING ORGANIZATION NAME AND ADDRESS
General Electric Researcy & Development Center
Schenectady, NY 12301

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
61153N (Program Element)
RR011-07-02 (Project)
NR 396-063 (Work Unit)

11. CONTROLLING OFFICE NAME AND ADDRESS
Code 412
Dept. of Navy, Office of Naval Research

12. REPORT DATE
25 July 1984

13. NUMBER OF PAGES

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)
Same as above

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)
Unclassified - Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Papers presented at the 3rd MACSYMA Users' Conference, on applications of symbolic manipulation in research and engineering.

18. SUPPLEMENTARY NOTES
The purpose of the 3rd MACSYMA Users' Conference was to provide a forum for all areas of the development and applications of MACSYMA and similar symbolic manipulation systems.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Symbolic-manipulation, mathematics, automatic-code-generation, computer algebra

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Please see reverse.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Forward

The Proceedings of the 1977 MACSYMA Users' Conference contained 46 papers, of which 19 were written by members of the MIT Mathlab Group. The Proceedings of the 1979 MACSYMA Users' Conference contained 44 papers, of which 10 were written by members of the MIT Mathlab Group. This proceedings contains 43 papers, of which only 4 are written by members of the current MACSYMA Group.

In addition, the first two conferences dealt only with MACSYMA, while this conference includes participants and papers from most of the other symbolic manipulation systems now in use.

Clearly the MACSYMA User Community and the User Community of symbolic manipulation systems in general is growing and thriving.

I wish to thank all the authors for their efforts to get their manuscripts to me in time. As another indication of the advance of the computer revolution, all but two of the papers or abstracts included in this volume were produced by word processors or computer text formatters.

V. Ellen Golden
Editor

Officers of the 1984 MACSYMA Users' Conference

General Chairman:
Elizabeth Cuthill, David W. Taylor Naval Ship Research
Development Center

Co-Chairman:
M. A. Hussain, General Electric Corporate Reseach and Development

Local Arrangements:
Mickey McGinn, General Electric Corporate Research and Development

Proceedings Editor:
V. Ellen Golden, Symbolics, Inc.
Co-Editor
M. A. Hussain, General Electric Corporate Research and Development

Organizing Committee:
Dr. Carl Andersen, College of William and Mary
Dr. Jagdish Chandra, U. S. Army Research Office
Dr. Richard Pavelle, Symbolics, Inc.
Dr. Paul Wang, Kent State University
Dr. Richard Zippel, Massachusetts Institute of Technology

Sponsors:
General Electric Corporate Research and Development
Office of Naval Research
Symbolics, Inc.
U. S. Air Force Office of Scientific Research
U. S. Army Research Office

Invited Speakers

Monday:

Dr. Richard Pavelle, Symbolics, Inc.
Prof. Stanly Steinberg, University of New Mexico
Prof. Paul S. Want, Kent State University
Prof. Carl Andersen, College of William and Mary

Tuesday:

Prof. Richard J. Fateman, University of California at Berkeley
Prof. Keith O. Geddes, University of Waterloo
Dr. Stephen Wolfram, Institute for Advanced Studay, Princeton
Prof. David R. Stoutemyer, University of Hawaii
Prof. John P. Fitch, University of Bath

Wednesday:

Prof. Donald F. Stanat, University of North Carolina at Chapel Hill
Prof. George E. Andrews, Penn State University
Dr. Richard D. Jenks, IBM Thomas J. Watson Research Center
Dr. Ben Noble, University of Wisconsin Mathematics Research Center

Banquet Speaker:

Prof. Joel Moses, Massachusetts Institute of Technology

INTRODUCTION

The purpose of this Third MACSYMA Users' Conference is to provide a forum for the discussion of all areas related to the development and application of MACSYMA and of similar systems. Five years have passed since the last MACSYMA Users' Conference. A much larger and more dispersed community is now using computers for performing symbolic, as well as numeric, mathematical computations. A meeting for the purpose of exchanging information between individuals applying such systems to the solution of scientific and engineering problems is especially timely.

At the time of the previous MACSYMA Users' Conferences, MACSYMA was on the MACSYMA Consortium Computer at MIT under the direction of Professor Joel Moses. The ARPANET made access to MACSYMA convenient for a user community geographically distributed from Hawaii to England. The MACSYMA Consortium no longer exists. Most users have their own copies of MACSYMA on VAX's ("VAXIMA") or on the Symbolics 3600 ("LISP") machine. A few are still using MACSYMA via ARPANET on the MIT-MULTICS.

The First MACSYMA Users' Conference was held at the University of California at Berkeley in 1977 and the Second in Washington, D.C. in 1979. The Proceedings of these conferences were published by NASA (as NASA Report No. CP-2012) and by MIT Laboratory for Computer Science, respectively.

The early development of MACSYMA at MIT was funded by the Defense Advanced Research Projects Agency. Later sponsors included the Department of Energy, the National Aeronautics and Space Administration, the U.S. Navy, the U.S. Army and the U.S. Air Force.

The Third MACSYMA Users' Conference sponsors include the General Electric Corporate Research and Development Center, Symbolics, Inc., the U.S. Army Research Office, the Office of Naval Research and the Air Force Office of Scientific Reserach.

We are grateful to all of those who have made this Conference and the publication of these Proceedings possible including our sponsors, the Organizing Committee, the invited speakers, those who contributed papers and the conferences attendees. Very special thanks are due to Dr. Hussain of GE who initiated the conference and contributed in so many ways to its success, to Ellen Golden of Symbolics whose efforts have made the publication of these proceedings possible and to Mickey McGinn and others at GE who attended to the many conference details.

Elizabth Cuthill
Conference Chairman

William A. Beyer
Los Alamos National Laboratory
Solution of Simultaneous Polynomial Equations by Elimination in MACSYMA

15:45 - 16:00 Break (CRD Lobby)

16:00 - 17:45 Dr. David Wood
Naval Underwater Systems Center
An Overdetermined System of Partial Differential Equations

Leo Harten
Paradigm Associates, Inc.
Applications of MACSYMA in Solving Linear Systems of Differential Equations

Ralph Wilcox and Leo Harten
Hughes Aircraft and Paradigm Associates, Inc.
Analytical Solutions to Some Matrix Ricatti Equations

Dr. Kenneth A. Bannister
Naval Surface Weapons Center
MACSYMA-Aided Large Deformation Analysis of a Cylindrical Shell Under Pure Bending

P. Hollis and Prof. D. L. Taylor
Cornell University
Hopf Bifurcation in Multi-Degree-of-Freedom Systems Using MACSYMA

R. Drew Drinkard, Jr.
Naval Underwater Systems Center
Tutorial on Particular Uses of MACSYMA

18:15 Social Hour (Atrium)

19:00 Banquet (Cafeteria)

20:30 Auditorium
Prof. Joel Moses
Massachusetts Institute of Technology

21:30 Bus Departs from Lobby for Hotels

Tuesday Morning, July 24

8:15 Bus Departs from Ramada Inn and Holiday Inn

Chairpersons: Dr. Elizabeth Cuthill, David W. Taylor, Naval Ship R&D Center

9:00 - 9:45 Prof. Richard Fateman et al.
University of California at Berkeley
Research in Algebraic Manupilation at the University of California at Berkeley

9:45 - 10:30 Prof. Keith Geddes et al.
University of Waterloo
On the Design of Performance of the Maple Systems

10:30 - 10:45 Break (CRD Lobby)

10:45 - 11:30 Dr. Stephen Wolfram
Institute for Advanced Study, Princeton
Five Years of SMP

Technical Program
Third MACSYMA Users' Conference
July 23-25 1984
All sessions will be held in the audirotium of the
General Electric Corporate Research and Development Center

Monday Morning, July 23

7:45 Bus Departs from Ramada Inn and Holiday Inn

8:15 - 9:00 Registration (CRD Lobby)

9:00 - 9:15 Opening Remarks and Welcome (Auditorium)

Dr. Elizabeth Cuthill, David W. Taylor Naval Ship R&D Center
Dr. Pete Meenan, Information System Operation, GE CRD

Technical Session

Chairperson: Dr. Jagdish Chandra, U.S. Army Research Office

9:15 - 10:00 Dr. Richard Pavelle
Symbolics, Inc.
Status and Future of MACSYMA

10:00 - 10:45 Prof. Stanly Steinberg and Patrick Roache
University of New Mexico
Using VAXIMA to Write FORTRAN Code

10:45 - 11:00 Break (CRD Lobby)

11:00 - 11:45 Prof. Paul Wang
Kent University
MACSYMA Aided Finite Element Analysis

11:45 - 12:30 Prof. Carl M. Andersen
College of William and Mary
Noor Reduction Technique for Differential Equations

12:45 - 14:00 Lunch (Cafeteria)

Monday Aftenoon

Chairperson: Dr. Sandra DeLoatch, NASA Langley Research Center

14:00 - 15:45 Dr. Raymond Mejia
National Institute of Health
Some Applications of Symbolic Manipulation in Biomathematics

J. F. Schenck, MD, and M. A. Hussain
GE Corporate Research and Development
Application of MACSYMA to a Boundary Value Problem in Nuclear Magnetic Resonance Imaging

Prof. Johnnie W. Baker and Oberta Slotterberg
Kent State University, Hiram College and University of Texas at Austin
Providing a Complex Number Environment for MACSYMA and VAXIMA

Dr. Richard L. Brenner
Symbolics, Inc.
Simplifying Large Algebraic Expressions by Computer

9:00 - 9:45	Prof. Donald Stanat University of North Carolina at Chapel Hill *A Functional Language Machine and Its Programming*
9:45 - 10:30	Prof. George Andrews Penn State University *Ramanujan and SCRATCHPAD*
10:30 - 10:45	Break (CRD Lobby)
10:45 - 11:30	Dr. R. D. Jenks IBM *The New SCRATCHPAD Language and System for Computer Algebra*
11:30 -12:15	Prof. Ben Noble and M. A. Hussain University of Wisconsin Mathematics Research Center and GE CRD *Application of MACSYMA to Kinematics and Mechanical Systems*
12:30 - 13:30	Lunch (Cafeteria)

Wednesday Afternoon

Chairperson: Dr. Robert B. Grafton, Office of Naval Research

13:30 - 15:45	Prof. Dennis Arnon and Thomas W. Sederberg Purdue University *Implicit Equation for a Parametric Surface by Groebner Basis*
	Abdelilah Kandri-Rody and Deepak Kapur Rensselaer Polytechnic Institute and GE CRD *Computing the Groebner Basis of an Ideal in Polynormial Rings over the Integers*
	S. Agnarsson, A. Kandri-Rody, D. Kapur, P. Narendran, and B. D. Saunders Rensselaer Polytechnic Institute and GE CRD *Complexity of Testing Whether a Polynomial Ideal Is Nontrivial*
	A. Kandri-Rody and B. D. Saunders Rensselaer Polytechnic Institute *Primality of Ideals in Polynomial Rings*
	Erich Kaltofen and Nori Yui Rensselaer Polytechnic Institute *On the Modular Equation of Order 11*
	W. Dubuque Symbolics, Inc. *New Foundations for Computer Algebra*
15:45 - 16:00	Break (CRD Lobby)
16:00 - 17:45	Dr. James O'Dell Symbolics, Inc. *Role of Maintenance in Knowledge Programming*
	Dr. J.T. Bendler GE Corporate Research and Development *Levy (Stable) Probability Densities and Relaxation in Solid Polymers*
	M.A. Hussain and J.F. Schenck GE Corporate Research and Development *Approximate Solution of an Integral Equation That Arises in the Design of Magnetic Field Coils*

11:30 - 12:15	Prof. David R. Stoutemyer University of Hawaii *Which Polynomial Representation is Best?*
12:15 - 12:45	Poster Session (Conference Room 5)
12:45 - 14:00	Lunch (Cafeteria)
	Tuesday Afternoon
	Chairperson: Dr. Phil M. Lewis, Computer Science Branch, GE CRD
14:00 - 15:45	Prof. John Fitch University of Bath, England *A Survey of Reduce*
	Dr. R. Berman Massachusetts Institute of Technology *Measuring the Performance of a Computational Physics Environment*
	Dr. Elizabeth Cuthill David W. Taylor Naval Ship R&D Center *Evaluating Infinite Integrals Using MACSYMA*
	George J. Carrette Massachusetts Institute of Technology and Paradigm Associates *Results in Unexpected MACSYMA Implementation Environments*
	Dr. Jon J. Yagla Naval Surface Weapons Center *Stability Criteria for Finite Difference Equations*
15:45 - 16:00	Break (CRD Lobby)
16:00 - 17:45	Paul D. Engleman The Pentagon, Washington, DC *Two to Three Dimensional Mapping*
	Barbara L. Gates and Paul S. Wang Kent State University *A LISP-Based RATFOR Code Generator*
	Michael Wester and Prof. Stanly Steinberg University of New Mexico *A Survey of Symbolic Differentiation Implementations*
	Gene Cooperman GTE Labratory *Alternate Top-Level for MACSYMA*
	V. Ellen Golden Symbolics, Inc. *Computational Geography - The Habitats of the Migratory MACSYMA*
18:00	Bus Departs for Hotels
	Wednesday Morning, July 25
8:15	Bus Departs from Ramada Inn and Holiday Inn
	Chairperson: Dr. Steve Wolff, Ballistic Research Lab

Carl R. Powell
Kent State University
An Automatic Testing Facility for VAXIMA

Leo Harten and G. Carrette
Paradigm Associates
Using MACSYMA to Generate (Somewhat) Optimized FORTRAN Code and Macros, Translation and Compilation in MACSYMA

17:45 Adjourn

USING VAXIMA TO WRITE FORTRAN CODE†

Stanly Steinberg
Department of Mathematics and Statistics
University of New Mexico
Albuquerque, NM 87131

Patrick Roache
Ecodynamics Research Associates, Inc.
P.O. Box 8172
Albuquerque, NM 87198

Abstract

This paper describes the symbol manipulation aspects of a project that produced a large FORTRAN program that is now used to model lasers and other physical devices. VAXIMA (MACSYMA) was used to write subroutines that were combined with standard software to produce the full program.

1. INTRODUCTION AND PROJECT OVERVIEW

The purpose of this paper is to describe the symbol manipulation aspects of a project that used VAXIMA (MACSYMA) [8] to write FORTRAN subroutines that are part of a finite difference code that is now used to model lasers and other physical devices. Some of the results of this project were reported in [10, 11, 12, 13, 15, 16] and some substantial improvements will be reported in [18]. Without the help of a symbol manipulator, portions of the project would have been impossible. With the help of VAXIMA, useful code was produced in a few weeks. Thus, as users of a symbol manipulator, we had a reduction in code development time that was infinite, a fact of importance for anyone considering using a symbol manipulator in a code development project. Unfortunately, not all of our report is accolades.

The physical devices that are being modeled are assumed to be in a steady state and consequently it is assumed that the physics of interest can be modeled by a

† This work was partially supported by the U.S. Army Research Office, by the U.S. Air Force Office of Scientific Research, by the National Science Foundation Grant #MCS-8102683, and by System Development Foundation.

partial differential equation (called the hosted equation) that involves the Laplace operator or a generalization of this operator. Mathematically, the equations must be elliptic. Such differential equations have been well studied both analytically and numerically. The difficulties come not from the differential equations, but from the fact that the physical devices have an irregular shape. This means that the full model will involve a boundary value problem in an irregular three dimensional region, or if the device has sufficient symmetry, then a boundary value problem in an irregular two dimensional region.

There are several different finite difference or finite element methods available for handling such problems. We are interested in a technique called *Boundary-Fitted Coordinates* that involves finite difference techniques. This subject has become a field of study in its own right as evidenced by the proceedings [2, 6, 19]. The basic idea is to find a transformation (or change of coordinates) that maps the given region (called *physical* space) into a rectangular region (called *logical* space), see Figure 1. In the rectangular region it is easy to produce finite difference schemes. Although the idea is simple, there are several complications and it is these complications that made using a symbol manipulator so helpful.

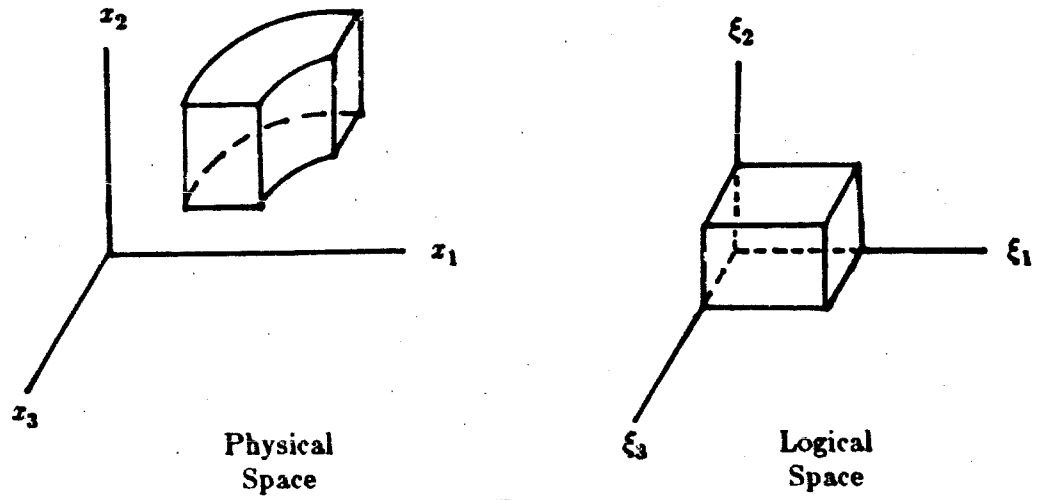


Figure 1

One complication is that the given regions are usually so irregular that it is impossible to find analytic transformations; the transformations must be determined numerically. The geometric idea that underlies the numerical methods is that the transformation should be smooth. Historically this is translated into requiring the transformation be harmonic: that each component of the transformation satisfy Laplace's equation. If the transformations are required to be harmonic as mapping from logical space to physical space, then converting the differential equations to finite difference schemes is easy. However, it was discovered that such a formulation leads to poorly behaved numerical methods while if the transformations are required to be harmonic as mappings from physical space to logical space, then the numerical methods

are better behaved. In this formulation, the Laplace equations are called the smoothness equations. The fact that the smoothness equations must now be transformed to equations on logical space is an important complication in this approach. This is done using the yet to be determined transformation from logical to physical space which results in a coupled system of quasi-linear elliptic differential equations (also called the smoothness equations) for determining the the transformation from logical to physical space. The prescription of the transformation on the boundary of the region provides the boundary conditions for the smoothness equations. We used VAXIMA to do the algebra and calculus in this step. The condition that the transformation be harmonic has been generalized, and we included these generalizations in our work. More recently, the harmonic condition has been replaced by a variational principle which we will describe in Section 5.

Another complication is that the hosted equations (the equations describing the physical process) must be transformed to logical space. Because the transformation is not known analytically, a general transformation must be used. To give the reader some idea of the size of these problems, we note that the Laplacian in three dimensions in general coordinates and in fully expanded form contains 1611 terms. Again, we used VAXIMA to do this algebra and calculus although we did not always use the fully expanded form. Once the smoothness and the hosted equations are known in logical space, they must be converted to finite difference form. This was also done with VAXIMA. Finally VAXIMA was used to write all of the formulas into a file in a form appropriate for the FORTRAN compiler. In fact, we had VAXIMA write two complete subroutines, one for the hosted equation and one for the smoothness equations. A few minutes work with the editor and the subroutines were ready to compile.

After the subroutines were written, they were combined with other standard numerical software to produce a program that could model physical devices. Although the level of confidence in the resulting code was high there was no guarantee that it was correct. Consequently, the program was checked using convergence rate testing on a set of examples that would exercise all parts of the program. The program has now been distributed to several universities and laboratories for production modeling. At the end of this paper we have included three coordinate systems generated by our programs. Each figure represents a coordinate system in the interior of a laser cavity. Figures 6 and 7 represent regions that have one axis of symmetry while Figure 8 represents the surface of a three dimensional region. We did not label the axes in these figures because such labeling is arbitrary and should be chosen for the convenience of the user. The details of the devices being modeled are described in [11, 13].

We now describe some parts of this project in more detail. In Section 2 we will describe the mathematical formulation of coordinate changes, in Section 3 we will describe how to introduce the finite difference schemes, and in Section 4 we will describe how to write the FORTRAN code. Section 5 is devoted to describing a variational formulation of the grid problem while Section 6 is devoted to a summary of what was accomplished. This project provided us with experiences we believe are

relevant to general applied mathematics and symbol manipulations problems so some of our opinions are presented in Section 7. Finally, some of the basic ideas used in this project we used in a project to develop a program that performs analytic changes of coordinate for partial differential equation, so a brief introduction to this material is given is Section 8.

2. MATHEMATICAL FORMULATION

Here we will give a brief introduction to the mathematics involved in our problem. The mathematical formulation is done in n-dimensions, not because we need the formulation for dimensions other than 2 and 3, but because this allows us to write VAXIMA code that works for all dimensions including 2 and 3. Thus a point in space will be denoted

$$\vec{x} = (x_i) = (x_1, x_2, \ldots, x_n) \tag{2.1}$$

where n is a positive integer parameter and we think of $\vec{x}$ as a column vector. The simplest hosted equation is the Laplacian,

$$\Delta f = \sum_i \frac{\partial^2 f}{\partial x_i^2} , \tag{2.2}$$

where $f = f(\vec{x})$ and we assume that all sums run from 1 to n. In general, the hosted equation will have the form

$$Lf = \sum_{ij} a_{ij} \frac{\partial^2}{\partial x_i \partial x_j} f + \sum_i b_i \frac{\partial}{\partial x_i} f + cf + d \tag{2.3}$$

where a_{ij} , b_i , c , d and f depend on $\vec{x}$. The smoothness equations have a similar form where the coefficients depend on the unknown transformation, that is, the equations are quasi-linear rather than linear.

In our applications we will be doing numerical calculations in logical space so we will want to write all of our formulas in terms of the $\vec{\xi}$ variables. Thus we write

$$\vec{x} = \vec{x}(\vec{\xi}), \tag{2.4}$$

and choose the Jacobian matrix J to be

$$J = (J_{ij}) = \left(\frac{\partial x_j}{\partial \xi_i}\right) = \begin{Vmatrix} \frac{\partial x_1}{\partial \xi_1} & \frac{\partial x_2}{\partial \xi_1} & \cdots \\ \frac{\partial x_1}{\partial \xi_2} & \cdots & \\ \cdots & & \cdots \end{Vmatrix} \tag{2.5}$$

and K to be the cofactor matrix of J. Thus if $\Delta = \textit{determinant}(J)$, then

$$J^{-1} = \frac{K}{\Delta} . \tag{2.6}$$

The chain rule gives

$$\frac{\partial}{\partial \xi_i} = \sum_j J_{ij} \frac{\partial}{\partial x_j} \tag{2.7}$$

and multiplying by K/Δ gives

$$\frac{\partial}{\partial x_i} = \frac{1}{\Delta} \sum_j K_{ij} \frac{\partial}{\partial \xi_j} . \tag{2.8}$$

The formula for $\partial/\partial x_i$ can be used to compute the second derivatives,

$$\begin{aligned} \frac{\partial^2}{\partial x_i \partial x_j} &= \sum_{rs} \frac{1}{\Delta} K_{ir} \frac{\partial}{\partial \xi_r}\left(\frac{1}{\Delta} K_{js} \frac{\partial}{\partial \xi_s}\right) \\ &= \sum_{rs} \frac{1}{\Delta^2} K_{ir} K_{js} \frac{\partial^2}{\partial \xi_r \partial \xi_s} + \sum_{rs} \frac{1}{\Delta} K_{ir} \frac{\partial}{\partial \xi_r}\left(\frac{1}{\Delta} K_{js}\right) \frac{\partial}{\partial \xi_s} . \end{aligned} \tag{2.9}$$

The next step is to use the above formulas to transform the hosted and smoothness equations to logical space. In our first approach, this is exactly what we did. We believe that this is a very natural approach to the problem and the fact that this leads one to a trap points out that symbol manipulation is not as easy as it may seem. Using this approach, VAXIMA required about 60 cpu hours to write the subroutine for the hosted equation in three dimensions and produced about 1800 lines of very dense FORTRAN code. This work was done on a VAX 11/780 computer with 4 megabytes of RAM memory. After some minor adjustments to the f77 compiler, approximately one cpu hour was required to compile the subroutine. As it turned out, the subroutine was correct the first time it was written, which is not the same as saying that no errors were made in the symbol code.

If a less obvious approach is taken, then VAXIMA can write an equivalent subroutine in about 8 cpu minutes (60 cpu hours over 8 cpu minutes equals 450). This subroutine contains only 180 lines of code (1800 lines over 180 lines equals 10). As we proceed with our discussion we will point out the differences between the first and the second approaches that make such a great difference in the VAXIMA run time and the size of the subroutine.

Before we proceed we need to point out that we would have liked to derive the above formulas using VAXIMA. This is not practical because VAXIMA does not know about vectors of length n, where n is an integer parameter and does not know about functions of n variables where n is an integer parameter. We believe that none of the existing symbol manipulators can do this type of computation.

A major improvement in our programs was achieved by noting that there is a classical formula for differentiating the inverse of a matrix A whose entries are functions of the ξ variables,

$$\frac{\partial}{\partial \xi_i}\frac{1}{A} = -\frac{1}{A}\frac{\partial A}{\partial \xi_i}\frac{1}{A}. \tag{2.10}$$

Combining this with a previous formula gives

$$\frac{\partial}{\partial \xi_r}\left(\frac{1}{\Delta}K_{jt}\right) = -\frac{1}{\Delta^2}\sum_{uv} K_{ju}\frac{\partial}{\partial \xi_r}J_{uv}\,K_{vt}. \tag{2.11}$$

and thus

$$\frac{\partial^2}{\partial x_i \partial x_j} = \frac{1}{\Delta^2}\sum_{rs} K_{ir}K_{js}\frac{\partial^2}{\partial \xi_r \partial \xi_s} - \frac{1}{\Delta^3}\sum_{rsuv} K_{ir}K_{ju}K_{vs}\left(\frac{\partial}{\partial \xi_r}J_{uv}\right)\frac{\partial}{\partial \xi_s}. \tag{2.12}$$

It is this last formula that was used in the second approach. Moreover, in the first approach we plugged everything into the differential equation that was to be transformed and then expanded out the formula. This produced a rather large expression which, in turn, accounts for part of the excessive time used in the first approach. In the second approach, we introduced some intermediate quantities that are equal to various coefficients in the previous expression. This has the effect of reducing the size of the expressions to be manipulated but makes it very difficult to understand what is the simplest form for expressions defined in terms of the intermediate expressions. In either case, this produces the formulas needed to generate the finite difference equations. For more details see [16].

3. FINITE DIFFERENCES

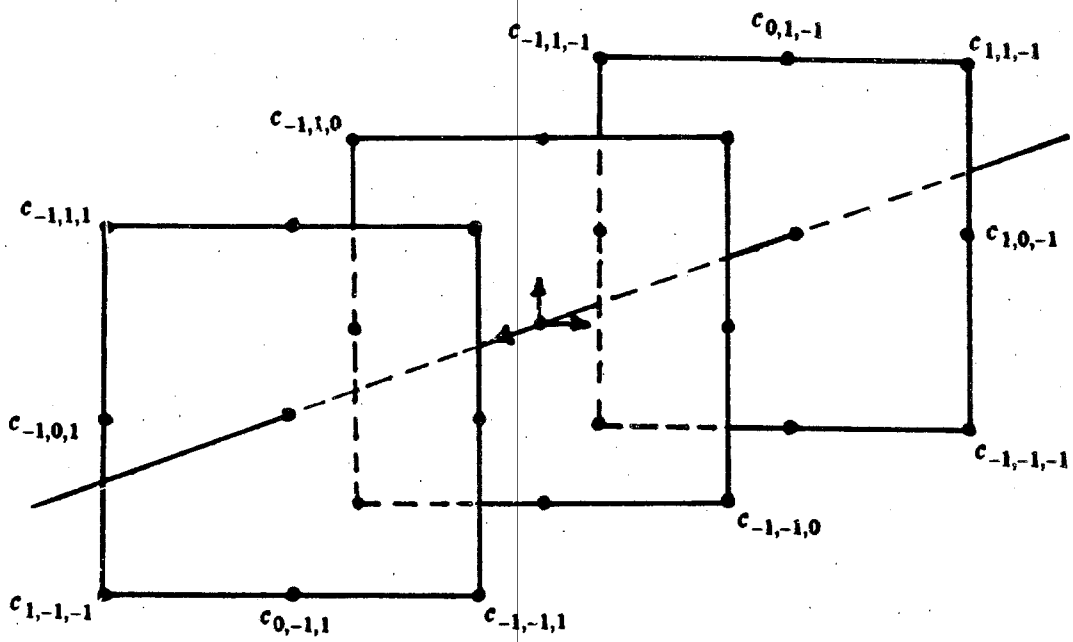
The next step is to replace all of the ξ derivatives in the previous formulas by standard centered finite differences (which we do not write here). The portions of the VAXIMA code that deal with finite differences were programmed separately for two

and three dimensions because we found it impossible to do this with the dimension n as a formal parameter, see also [22]. The fact that we could not manage a general formulation was disappointing, but VAXIMA was still an indispensable tool for completing this part of our work.

In our first approach we simply substituted these formulas into the expressions for the differential equations. Now our formulas were getting really large but were, in reality, not very complicated. In three dimensions, the resulting difference equations have the form

$$\sum_{|i|+|j|+|k|\leq 3} c_{i,j,k}(\xi_1,\xi_2,\xi_3)\; g(\xi_1+i\Delta\xi_1,\xi_2+j\Delta\xi_2,\xi_3+k\Delta\xi_3) = R(\xi_1,\xi_2,\xi_3)\,. \qquad (3.1)$$

Here i , j , k each run over the set of values {-1,0,1} . The coefficients $c_{i,j,k}$ are called a stencil. For convenience, we view the elements of the stencil as points on a cube in three space and then label the elements as in Figure 2. Note that not all of the stencil elements have been labeled so as to avoid cluttering the figure. In addition, the ξ_1 axis points to the right, the ξ_2 axis points up, and the ξ_3 axis points forward.



The Stencil
Figure 2

It is easy to check that, because we used centered differences, there are many relationships among various stencil elements, for example,

$$c_{1,1,1} = 0 \ , \ c_{-1,0,1} = c_{1,0,1} \ .$$

These relationships reduce the 27 stencil elements to 10 independent stencil elements.

Because of these relationships, the formulas generated by the first method contain substantial redundant information. In the second method the difference formulas were never substituted into the differential equations. Instead, various coefficients of the differential equation were collected and combined to give the formulas for the stencil elements. This can be thought of as reversing the order of the substitution and the collecting of the coefficients. We did not find this convenient in VAXIMA. It is our impression that decisions concerning the order in which the steps of a computation are to be done are very important. Shoulda't symbol manipulators provide facilities to help the user with such decisions?

At this point the stencil elements are defined in terms of intermediate expressions which are, in turn, defined in terms of the parameters and derivatives of parameters appearing in the original differential equations. These expressions also contain derivatives of the coordinate change. The situation for the smoothness equations is a bit different from that for the hosted equation but the computations are similar. For more details, see [16]. In either case, these parameter functions and derivatives must be replaced by a notation that the FORTRAN compiler understands. In the three dimensional case, we created 31 atomic variables (variable names) and substituted these into the formulas needed to write the FORTRAN subroutine.

In the first formulation this was a disaster. At this point we were dealing with a very large formula and every time a substitution was made the simplifier rearranged the terms in the formula in an attempt to find simplifications where we knew that none existed. We estimate that roughly 50 of the 60 cpu hours used in the first method went into this step, that is, by stopping this irrelevant work we reduced the computation to about 10 cpu hours. The remaining savings required some mathematics to be done.

At this point our VAXIMA program had produced all of the formulas needed to write the FORTRAN subroutines. These formulas were recorded in several lists in the form of VAXIMA equations that corresponded to FORTRAN assignment statements. We had also collected lists of all of the variables used in the formulas that were to be used in the subroutines. So let us write the subroutines.

4. WRITING FORTRAN

At this point we had all of the formulas needed to write the FORTRAN code. We had also saved some lists that contained all of the variable used in these formulas. The next thing that needed to be done was to create the subroutine header, some comments, some variable declarations, and some loops over the grid in logical space. We could have written these things into a file and then used the VAXIMA *fortran* function to convert the formulas to FORTRAN syntax and then written these expressions into

a file. A few minutes work with an editor would have produced the subroutine. Instead, we decided to write a VAXIMA function to produce a complete subroutine. This is not a particularly efficient way to do things given the state of the facilities in VAXIMA or, for than matter, any other symbol manipulator. However, we now have some opinions on what is needed to make writing floating point programs more efficient and less irritating.

When we are writing FORTRAN code we would like our symbol manipulator to be a good programmer's assistant. We are not interested in converting symbol manipulation programs or functions to floating point programs (although some may be interested in this). One step in writing a program is creating the assignment formulas which is the forte of symbol manipulators. These formulas need to be in or converted to FORTRAN syntax. The arithmetic operations seem to cause no difficulty whereas the logical operators do cause problems. First note that the VAXIMA *fortran* function converts the VAXIMA assignment operator ":" to the FORTRAN assignment operator "=" while converting the VAXIMA logical "=" to the FORTRAN ".eq.". Here is a short sample of a VAXIMA run that illustrates the point.

```
(c_1) x:a^2+b^2;

                                    2    2
(d_1)                              a  + b

(c_2) y:x+c*d;

                                  2    2
(d_2)                            a  + b  + c d

(c_3) fortran(y);

(d_3)       a**2+b**2+c*d

(c_4) fortran(y:x+c*d);

(d_4)       y=a**2+b**2+c*d

(c_5) fortran(z=a^2+b^2);

(d_5)       z.eq.a**2+b**2
```

VAXIMA Output
Figure 3

Now imagine that the the values being assigned are very complicated and we don't want VAXIMA to do the substitutions, we want FORTRAN to do floating point evaluations and substitutions. Now things become a bit convoluted. We opted to use the VAXIMA "=" in our assignment formulas and then use the editor to change the

".eq." to "=". We believe a good resolution to this problem is to have a data type that is a FORTRAN assignment statement so that the meaning of "=" is clear.

A good assistant could help with many other chores. It would be nice to have the manipulator check that we had declared all of the variables used in the subroutine, that the formulas were in a consistent order, that all of the values used in the assignment formulas were computed locally or passed into the subroutine through the calling sequence or a common block, and that all of the values passed out of the subroutine were computed; it also would have been nice to have the manipulator read into the program a file that contained some comments. No doubt, there are many other programming tasks that a good assistant could do for the user. Again, we believe that it would be helpful if the symbol manipulator had a data type called FORTRAN program and could manipulate such an object. One thing that we could do with VAXIMA was very useful: we wrote all of the assignment formulas used in the subroutine into a file in the VAXIMA two dimensional format. This made reading the FORTRAN code much easier. In fact, much of our FORTRAN code is not very readable by humans. However, we hope the the VAXIMA code and the two dimensional format formulas are readable.

Of course, we have saved the real bad news to the last. All of the subroutines that we have written using VAXIMA contain an outrageous amount of redundant arithmetic. This can be corrected with a large amount of *at the terminal* work with either VAXIMA or a text editor, but there are many chances for mistakes. Because the subroutines that we wrote account for a small percentage of the total run time of our programs, we have not yet optimized the formulas. Anyone planning on using a symbol manipulator to write code should be aware of this problem. There is a function in VAXIMA called "optimize" that will correct this problem for small formulas, although "optimize" does not change any of our formulas and was not designed to work on a list of formulas where the formulas contain common expressions. It does not seem that it will be difficult to improve this situation. However, it is not clear to us how to optimize both the arithmetic count and the stability of formulas. We are currently working on these problems. To give the reader some idea of what we are talking about we have included, see Figure 4, one assignment statement from the three dimensional code.

```
 u1 = s33*vk31*z33+2*s23*vk31*z23+s22*vk31*z22+2*s13*vk31*z13+2*s1
1    2*vk31*z12+s11*vk31*z11+s33*vk21*y33+2*s23*vk21*y23+s22*vk21*y2
2    2+2*s13*vk21*y13+2*s12*vk21*y12+s11*vk21*y11+s33*vk11*x33+2*s23
3    *vk11*x23+s22*vk11*x22+2*s13*vk11*x13+2*s12*vk11*x12+s11*vk11*x
4    11
```

Sample of VAXIMA Written FORTRAN Code
Figure 4

Note that almost any rewriting of this expression in a form analogous to that given by

Horner's rule will improve the operation count and presumably the stability of evaluating the expression. The problem is that there are many ways of rewriting the formula and so it is not obvious how to automate such a procedure.

5. THE VARIATIONAL FORMULATION

Recently, there has been an interest in formulating variational problems for determining coordinate systems in physical space [3, 14, 18]. A problem with the previous methods is that they are ad hoc. There is some geometric intuition but the parameters in the method have no direct geometric interpretation. With the variational methods the parameters do have a geometric interpretation and consequently this geometric intuition can be used to help determine the parameters. Previously, numerical experimentation and experience were the best guides to choosing the parameters. In the simplest cases the variational methods yield the previous methods. However, the variational method provides direct control over many aspects of the grid including the smoothness, the angles between the grid lines, and the area or volume of the grid cells. For more details see [3, 14, 18].

In this section we will describe how to convert a variational problem into a FORTRAN subroutine. We will see that this type of problem is very appropriate for a symbol manipulator. On the other hand, the derivation of a variational problem from geometric intuition seems the proper domain for human thinking so we leave describing how this is done to another paper [18]. The simplest variational problem is the one that is related to smoothness, so we now describe that problem and its conversion to a FORTRAN subroutine for generating a grid. The more general variational problems involve the same mathematics; they are just more complicated and complicated enough to warrant using a symbol manipulator.

We will use the notation of the Section 2 and formulate an n-dimensional version of the variational problem. The variational problem is to find a transformation $\vec{x} = \vec{x}(\vec{\xi})$ mapping a rectangle S in logical space to the given region in physical space which minimizes the integral

$$I(\vec{x}) = \int_S \sum_{i,j=1}^{n} \left(\frac{\partial x_i}{\partial \xi_j} \right)^2 d\xi_1 \cdots d\xi_n \,. \tag{5.1}$$

The transformation $\vec{x}(\vec{\xi})$ is to be specified on the boundary of S. It is known that if $\vec{x}(\vec{\xi})$ minimizes $I(\vec{x})$ then the components $x_i(\vec{\xi})$ must satisfy the Euler equations which are a system of partial differential equations, that are well known. Instead of writing the Euler equations we will derive them. This is because the method of deriving these equations is far more interesting than the equations themselves.

Let $\vec{c}(\vec{\xi})$ be defined on the region S and be zero on the boundary of S. Then for every ϵ, the transformation $\vec{x} = \vec{x}(\vec{\xi}) + \epsilon\, \vec{c}(\vec{\xi})$ maps the region S in logical space to the given region in physical space. If $\vec{x} = \vec{x}(\vec{\xi})$ minimizes $I(\vec{x})$ then $\epsilon = 0$ must be a minimum of

$$F(\epsilon) = I(\vec{x}+\epsilon\vec{c}) \ , \tag{5.2}$$

that is, it must be the case that

$$\frac{dF}{d\epsilon}(0)=0 \ . \tag{5.3}$$

A common way of thinking of the above is to consider $I(\vec{x})$ to be a functional (function) on an infinite dimensional space of smooth functions and then the previous derivative is thought of as being a directional derivative of the functional $I(\vec{x})$ in the direction $\vec{c}$. This type of derivative is a direct generalization of the notion of directional derivative in finite dimensional spaces and is frequently called a Fréchet or Gâteaux derivative. This derivative has many diverse applications in applied mathematics.

A bit of calculus gives

$$\frac{dF}{d\epsilon}(0) = \int_S \sum_{i,j=1}^{n} \frac{\partial x_i}{\partial \xi_j} \frac{\partial c_i}{\partial \xi_j} d\xi_1 \cdots d\xi_n = 0 \ . \tag{5.4}$$

An integration by parts gives

$$\int_S \sum_{i,j=1}^{n} \frac{\partial^2 x_i}{\partial \xi_j^2} c_i \, d\xi_1 \cdots d\xi_n = 0 \ . \tag{5.5}$$

The integration by parts is easy to do symbolically because what is required is to remove all of the derivatives from the c_i.

The previous integral must be zero for all choices of $\vec{c}$ which are zero on the boundary of S, which implies that the coefficient of each c_i must be zero. Thus

$$\sum_{j=1}^{n} \frac{\partial^2 x_i}{\partial \xi_j^2} = 0 \ , \ 1 \leq i \leq n \ , \tag{5.6}$$

which are the usual smoothness equations!

Now we are in a position to apply all that we have learned previously. We would like to note that we are very fond of this approach to grid generation problems: it has geometric insight, straightforward computations, and considerable versatility. In addition, VAXIMA handles the computation easily if we don't try to completely implement the n-dimensional formulation.

6. SUMMARY

Let us look at what was accomplished. Using our first mathematical formulation it is certainly possible to generate the two dimensional FORTRAN subroutine by hand but probably impossible to generate the three dimensional subroutine by hand. The new mathematical formulation of the problem has probably brought the writing by hand of the three dimensional subroutine within the realm of possibility. For two dimensional codes there probably is no time saved in producing the subroutines using VAXIMA. Even though no time is saved in the production, it is still advantageous to use VAXIMA. The reason is that there is a very small probability of *typo* type errors in the VAXIMA written code. In fact, we did not have to spend any time debugging the FORTRAN subroutines although there was a small problem in combining the subroutines with the elliptic equation solver. This comment is a bit unfair because a reasonable amount of time was spent debugging the symbol codes. However, because the symbol codes are written at a higher mathematical level than the subroutines, they are usually correct or produce garbage, and consequently are considerably easier to debug than FORTRAN code. Thus the VAXIMA project had the advantage of requiring less debugging time and producing a product that we were confident was correct.

As stated before, the mathematical formulation used to write the VAXIMA code was derived for the general n-dimensional case. About half of this was programmed in VAXIMA using n as a formal parameter. The parts of the formulation that could not be programmed for the general case were programmed for the two dimensional case and then the two dimensional subroutines were written and tested. A modest amount of programming produced three dimensional versions of those parts that were not general and then it was possible to write the three dimensional subroutines. The fact that much of the VAXIMA code was used in the two dimensional case or was a direct analog of the code used in the three dimensional case gave us a very high level of confidence in the three dimensional subroutines, and now we had realized a tremendous saving of time!

Certainly VAXIMA is a useful tool. However, this project and other projects have shown that there are problems. Here we do not mean *bug* type problems; in fact this type of problem is rather rare; we mean problems in the fundamental design of VAXIMA. Clearly part of the problem could be that VAXIMA evolved over a number of years and involved a large number of programmers. However, we have looked at several of the new general purpose manipulators, which were certainly designed, and find that the problems are still there. Thus we are led to believe that there is some disparity between what the symbol manipulation community is designing and the needs of the applied mathematicians who use symbol manipulators.

7. COMMENTS

In this section we will make some general comments about the use of symbol manipulators in applied mathematics.

One of the most important phenomenon in symbol manipulation is that of *intermediate expression swell.* Certainly, our 60 cpu hour run times were a result of this phenomena. It seems reasonably clear that faster hardware is not going to be all that helpful in tackling many problems that have large intermediate expressions. Clearly some problems will have a *best* formulation that is very large and for such problems fast machines are crucial. We believe that improved design of symbol manipulators along with the user community developing more skill in using these programs will have more impact.

The problem of large intermediate expressions is not unique to computer symbol manipulation; the same problem occurs in hand computation. When students do computations in elementary courses we often refer to their approach as *plug and chug* and are clearly aware that more experience may improve their computational abilities: they will start to have an overview of their computations and will start to choose among several computational strategy. We believe [7] that more experienced users of mathematics use abstraction to overcome the intermediate expression swell problem. They tend to introduce symbols to represent large expressions. Such symbols need to be well chosen; they must have nice manipulation properties and represent important parts of the underlying problem. The more abstract symbols may be manipulated in an attempt to find an approach to a problem that has tractable intermediate expressions.

The fact that we could discover a reformulation of our problem that reduced our run time by nearly three orders of magnitude can be interpreted in many ways; perhaps we should have thought more before programming. From an applied mathematics point of view, our original programs were very natural and this is what allowed our rapid progress. We believe that the use of the identity that reduced the run time could have been found by a symbol manipulator. The identity is well known. The use of the identity is indicated because it allows some of the calculations to be done before the messy details are put into the formulas; clearly this a good thing to try.

Another problem is the notion of functional dependencies that is used by the VAXIMA differentiation routines. This notion is not adequate for our need and is probably not adequate to carry out many applied mathematics projects that involve multivariate calculus. This problem caused us to carry out hand derivations of the coordinate transformation formulas rather than doing this work in VAXIMA. As far as we can tell all of the manipulators that are commonly available have problems in this area. This problem is so important that M. Wester and one of the authors have published a paper [20] on this subject and are presenting a separate paper [21] on this subject at this conference.

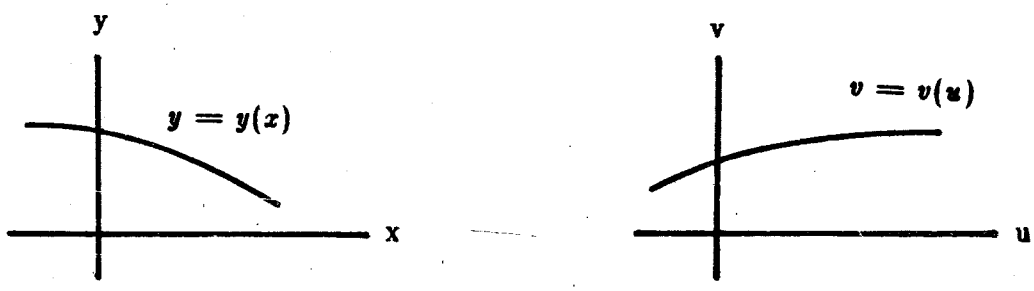
As we noted above, some of the VAXIMA code could be written using the dimension, n, of physical space as a parameter. The fact that about half of the VAXIMA code could not be written this way was more than a nuisance. One of the problems here is that it is not possible to define vectors of length n where n is an integer parameter and then have VAXIMA know how to manipulate such objects. Stated

more mathematically, it is not feasible to teach VAXIMA about abstract vector spaces. We believe it is impossible to over-estimate the importance of abstract vector spaces in applied mathematics. A simpler version of this problem can be found in the fact that it is not easy to define lists of length n where n is a formal integer parameter and then have VAXIMA (or Lisp) know how to manipulate these objects. As far as we know, no other manipulator has this type of facility.

8. ANALYTIC CHANGES OF COORDINATES

Some of the previous ideas we used in a project [17] that developed a program to perform analytic changes of coordinates for partial differential equations. There are some facilities available [8] for changing the independent variables in partial differential equations: what we wanted was a program that would change both the dependent and independent variables. The program developed will transform up to second order partial differential equations in any number of dependent and independent variables. As the algebra here is quite complicated, let us briefly describe the case of one dependent and one independent variable. In fact, what is needed is a program that will change any partial derivative in one coordinate frame to partial derivatives in a second coordinate frame. These formulas are then substituted into the partial differential equation.

Let x be the independent variable, while y is the dependent variable in the given coordinate frame, and let u be the independent variable and v be the dependent variable in the new coordinate frame, as is shown in Figure 5.



Change of Coordinates
Figure 5

The curves in Figure 5 are given by $y = y(x)$ and $v = v(u)$. We are interested in transforming the derivative $y' = dy/dx$ into an expression involving the derivative $v' = dv/du$.

We assume that the transformations are given implicitly,

$$F(x,y,u,v) = 0 \ , \ G(x,y,u,v) = 0 \ , \tag{8.1}$$

because this was the case that occurred in our applications. Here we assume that it is possible to solve these equations numerically for x and y in terms of u and v, that is, a certain Jacobian described below is not zero and that the Jacobian of the resulting transformation is nonzero. This will imply that the inverse transformation exists, that is, the equations can be numerically solved for u and v in terms of x and y. We are not assuming that the equations can be solve algebraicly, although if this can be done then the results we obtain can be improved.

Because this case is so simple it can be done in many ways. We found that the ideas from elementary calculus were not powerful enough to allow us to do the more general problem so we opted to use differential forms to solve the problem. It should be noted that differential forms are nice because they convert analytic problems to linear algebraic problems as we will see below. First, calculate the differentials of the transformation:

$$\begin{aligned} F_x\,dx + F_y\,dy + F_u\,du + F_v\,dv &= 0\ , \\ G_x\,dx + G_y\,dy + G_u\,du + G_v\,dv &= 0 \end{aligned} \tag{8.2}$$

where dx, dy, du, and dv are the differential of the dependent and independent variables and $F_x = \partial F/\partial x$ and so forth. Introduce the matrices

$$M_1 = \begin{bmatrix} F_y & F_x \\ G_y & G_x \end{bmatrix}\ , M_2 = \begin{bmatrix} F_v & F_u \\ G_v & G_u \end{bmatrix}\ . \tag{8.3}$$

Our assumptions on the Jacobians means that the determinants of the matrices M_1 and M_2 must not be zero. Now the system of equations (8.2) can be solved for dy and dx yielding

$$dy = A\,dv + B\,du\ ,\ dx = C\,dv + D\,du \tag{8.4}$$

where the matrix

$$M = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \tag{8.5}$$

is given by

$$M = M_1^{-1}\,M_2\ . \tag{8.6}$$

Note that M depends on x, y, u, and v.

To transform the first derivatives note that

$$\frac{dy}{dx} = \frac{A\,dv + B\,du}{C\,dv + D\,du} = \frac{A\frac{dv}{du} + B}{C\frac{dv}{du} + D} \tag{8.7}$$

which implies that

$$y' = \frac{A + B\,v'}{C + D\,v'} \tag{8.8}$$

As noted above A, B, C, and D depend on x, y, u, and v. If the equations (8.1) can be solved algebraically for x and y in terms of u and v, then this information can be used in the previous equation. This should not be done before computing the second derivative.

The computation of the transformation of higher derivatives is simple: for the second derivative compute, as we did for the first derivative, divide dy' by dx and then use the formulas for dy and dx to eliminate these terms from the resulting expression. This computation can be done using the matrix form of the equations, in which case it is important to use the identity that was discussed in Section 2 for differentiating the inverse of a matrix. The computations in the multivariate case are similar but considerably more complicated than what we just did. Since we carried out the derivation of these formulas by hand in [17], we do not believe the computation to be practical in VAXIMA. As can be seen from this note and the paper [17], we believe that it is important for symbol manipulators to know about differential forms.

References

1. Anderson, J.D., Lau, E.L., and Hellings, R.L. Use of MACSYMA as an automatic FORTRAN coder. In *1979 MACSYMA Users' Conference*, (V.E. Lewis, Ed.), (Washington, D.C., 1979), pp. 583-595.

2. Babuska I., Chandra, J., and Flaherty J.E., Eds. *Adaptive Computational Methods for Partial Differential Equations*, *SIAM*, (Philadelphia, 1983).

3. Brackbill, J.U. and Saltzman, J.S. Adaptive Zoning for Singular Problems in Two Dimensions. *J. Computational Physics, 46* (1982), pp. 342-368.

4. Engquist B. and Smedsaas T. Automatic computer code generation for hyperbolic and parabolic differential equations. *SIAM J. Sci. Stat. Comput., 1*, (1980), pp. 249-259.

5. Fateman R. MACSYMA's general simplifier: philosophy and operation. In *1979*

MACSYMA Users' Conference, (V.E. Lewis, Ed.), (Washington, D.C., 1979), pp. 563-582.

6. Ghia K.N. and Ghia U., Eds., *Advances in Grid Generation, ASME FED, 5*, (1983).

7. Hermann, R. This point was discussed while R. Hermann was visiting the Department of Mathematics and Statistics at the University of New Mexico in April of 1984.

8. *MACSYMA Reference Manual*, The MATHLAB Group, Laboratory for Computer Science, MIT, Cambridge, MA, 1977.

9. Ng E. and Char B. Gradient and Jacobian Computation for numerical applications. In *1979 MACSYMA Users' Conference*, (V.E. Lewis, Ed.), (Washington, D.C., 1979), pp. 604-621.

10. Roache, P. and Steinberg, S. Numerical aspects and potential of symbolic manipulations for partial differential equations. Talk presented at the Army Research Office - General Electric Corporation Workshop on Symbolic Computations, (Dec. 1982, Schenectady, NY).

11. Roache P.J., Steinberg S., Happ H.J., and Moeny W.M., 3D electric field solutions in boundary fitted coordinates. In *Proceedings of the 4th IEEE Pulsed Power Conference*, (Albuquerque, NM, June 1983).

12. Roache P.J. and Steinberg S., Symbolic manipulation and computational fluid dynamics (General background and demonstrations). In *Proceedings of the AIAA 6th Computational Fluid Dynamics Conference*, (Danvers, Mass., July 1983), pp. 443-462.

13. Roache R.J., Moeny W.M., and Steinberg S. Interactive Electric Field Calculations for Lasers. In *AIAA 17th Fluid Dynamics, Plasma Dynamics and Lasers Conference*, (June 1984, Snowmass, Colo.), pp. 25-27.

14. Saltzman, J.S. and Brackbill, J.U. Applications and generation of variational methods for generating adaptive meshes. *Numerical Grid Generation, Proceedings of the Symposium on the Numerical Generation of Curvilinear Coordinate Systems and use in the Numerical Solution of Partial Differential Equations*, (Thompson J.F., Ed.), (April 1982, Nashville, Tennessee), North-Holland, New York, 1982.

15. Steinberg, S. and Roache, P. Symbolic manipulation for generation of FORTRAN codes for partial differential equations. Talk presented at the Army Research

Office - General Electric Corporation Workshop on Symbolic Computations, (Dec. 1982, Schenectady, NY).

16. Steinberg S. and Roache, P.J. Symbolic manipulation and computational fluid dynamics. To appear in the *Journal of Computational Physics*, (1984).

17. Steinberg S. Change of Variables in partial differential equations. Technical report, Dept. of Math. and Stat., Univ. of New Mexico, Albuquerque, March 1983.

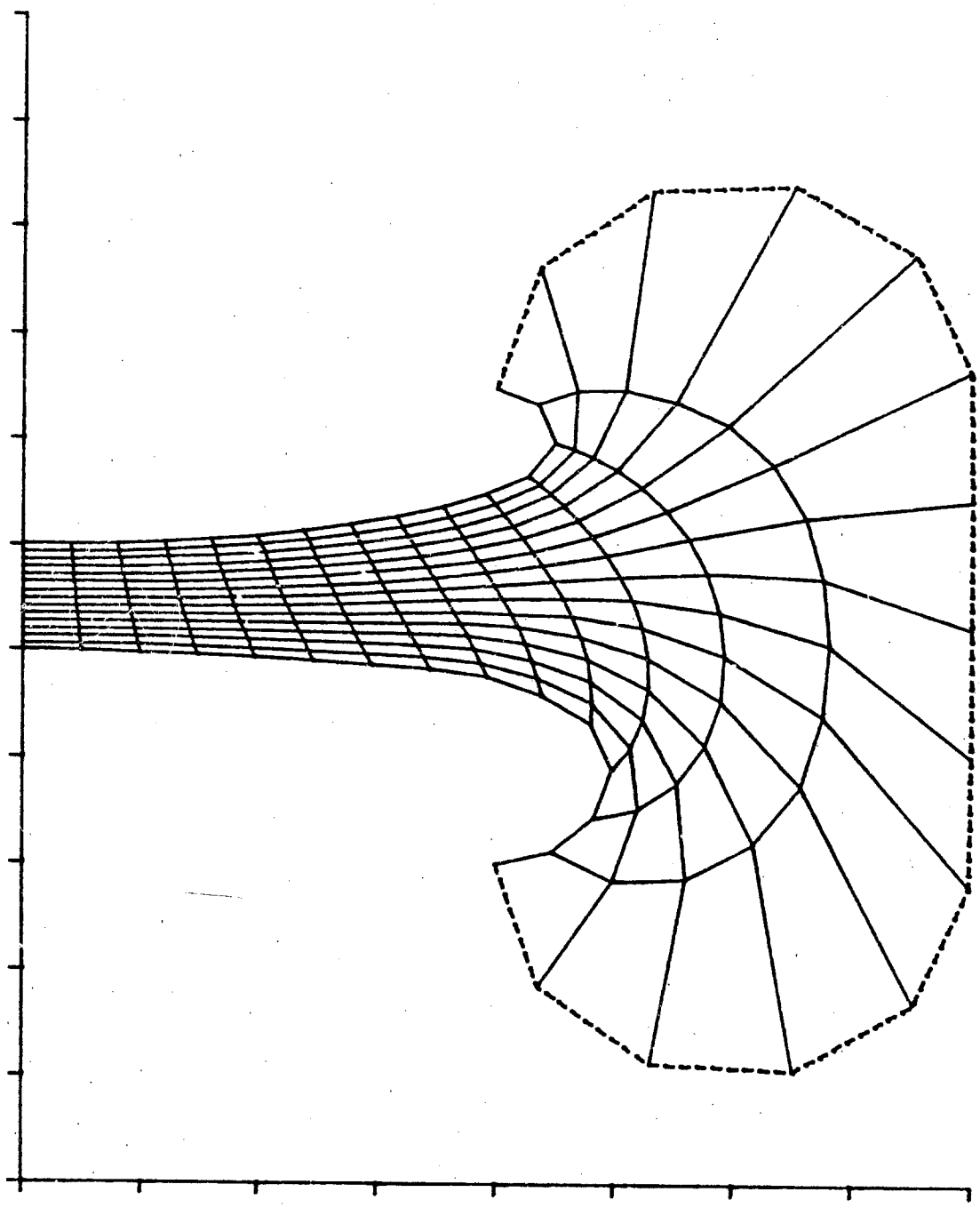
18. Steinberg S. and Roache P.J. Variational formulation for numerical coordinate changes, In preparation.

19. Thompson J.F., Ed. Numerical Grid Generation. *Proceedings of the Symposium on the Numerical Generation of Curvilinear Coordinate Systems and use in the Numerical Solution of Partial Differential Equations*, (April 1982, Nashville, Tennessee), North-Holland, New York, 1982.

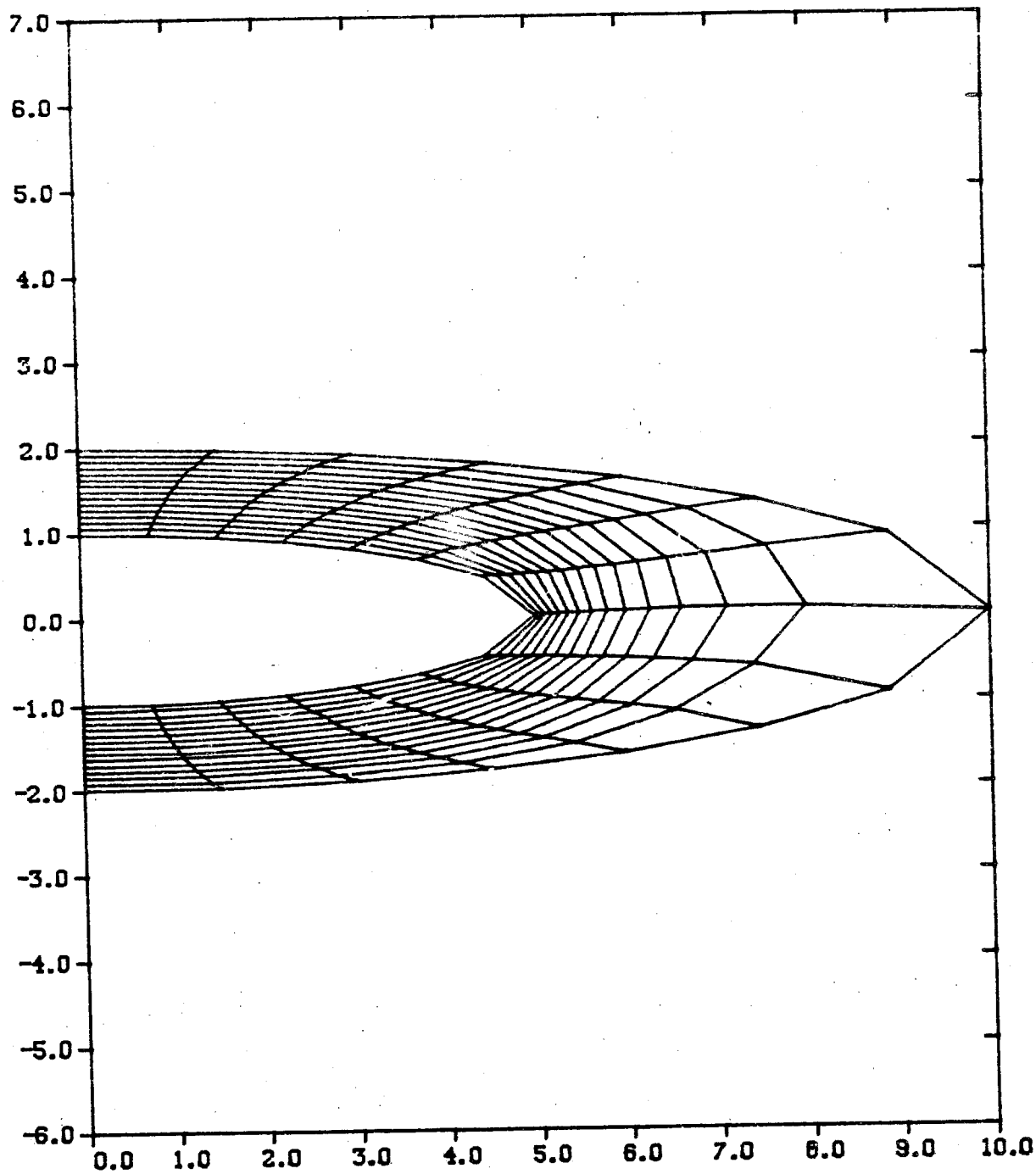
20. Wester M. and Steinberg S. An extension to MACSYMA's concept of functional differentiation. *ACM SIGSAM Bulletin 17*, (1983), pp. 25-30.

21. Wester M. and Steinberg S. A survey of symbolic differentiation implementations. In *Third MACSYMA Users' Conference*, (V.E. Golden, Ed.), (July 1984, Schenectady, N.Y.).

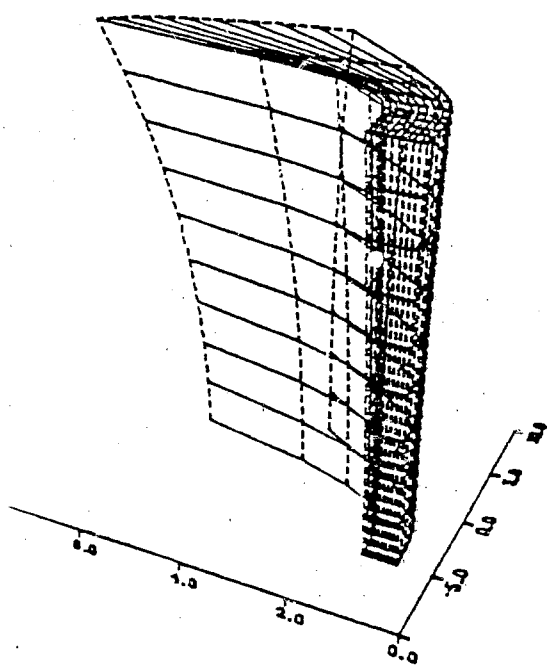
22. Wirth M.C. Automatic Generation of Finite Difference Equations and Fourier Stability Analysis. In *Proc. 1981 ACM Symposium on Symbolic and Algebraic Computation*, (Aug., 1981, Snowbird, Utah), pp. 73-78.



Grid for a 2D Laser Cavity
Figure 6



Grid for a 2D Laser Cavity
Figure 7



Grid for a 3D Laser Cavity
Figure 8

MACSYMA-AIDED FINITE ELEMENT ANALYSIS: TECHNIQUES FOR THE AUTOMATIC GENERATION OF NUMERICAL PROGRAMS

*Paul S. Wang**

Department of Mathematical Sciences
Kent State University
Kent, Ohio 44242

ABSTRACT

MACSYMA is used to derive formulas needed in finite element analysis. The automatically derived formulas are then used to generate programs which can be used with existing, FORTRAN-based finite element analysis packages. A system for this purpose is described. Symbolic computational techniques for efficient derivation and generation of code are discussed. These techniques are useful not only for finite element analysis but in general.

1. INTRODUCTION

One important application of a symbolic manipulation system is to facilitate the tedious pre-processing involved in large numerical computations. A MACSYMA subsystem is under construction for finite element analysis computations. The system deals with the symbolic derivation of quantities such as the strain-displacement matrix, the material properties matrix and the stiffness matrix. It contains a package for the generation of complete FORTRAN programs from symbolic formulas. The reader is referred to [1], [5], [6] and [7] for some previous work in this area. It is found that the degree of success of such an approach depends on how well several problems are dealt with:

* Work reported herein has been supported in part by the US National Aeronautics and Space Administration under Grant NAG 3-298 and by the US National Science Foundation under Grant MCS-82-01239 and by the Department of Energy under Grant DE-AC02-ER7602075-A010.

i) the efficiency of the symbolic processor and its ability to handle the large expressions associated with practical problems,

(ii) the interface between a symbolic system and a finite element system on the same computer, and

(iii) the inefficiencies that are usually associated with automatically generated code.

Techniques used and experiences gained in this research effort will benefit other automatic code derivation and generation applications.

We describe our on-going research on the design and implementation of this finite element generator. Ways to simplify derivation and to generate efficient code will be presented. Techniques used include the use of symmetry in the given problem, automatically generating functions and subroutines and the systematic identification of common subexpressions. The computer system used is a VAX-11/780 under Berkeley UNIX [12] which runs VAXIMA.

2. FUNCTIONAL SPECIFICATIONS AND DESIGN

The finite element generator (Generator) as a software system should provide the following functionalities.

(1) to assist the user in the symbolic derivation of finite elements;

(2) to provide routines for a variety of symbolic computations in finite element analysis, including linear and non-linear applications especially for shells [2];

(3) to provide easy to use interactive commands for most common operations;

(4) to allow the mode of operation to range from interactive manual control to fully automatic;

(5) to generate, based on symbolic computations, efficient FORTRAN code in a form specified by the user;

(6) to automatically arrange for generated FORTRAN code to compile, link and run with FORTRAN-based finite element analysis packages such as the NFAP package [3];

(7) to provide for easy verification of computational results and testing of the code generated.

In providing the above functions attention must be paid to making the system easy to use, modify and extend. Our initial effort is focused on the isoparametric element family. Later the system can be extended to a wider range of finite elements.

3. CODE DERIVATION AND GENERATION

As an example, we shall describe the automatic processing leading to the derivation of the strain-displacement matrix [B], and the elment stiffness matrix [K] in the isoparametric formulation. From user supplied input information such as the element type, the number of nodes, the nodal degrees of freedom, the displacement field interpolation polynomial and the material properties matrix [D], the Generator will derive the shape functions, the matrix [B] and the matrix [K].

The computation is divided into five logical phases each is implemented as a LISP program module running under the VAXIMA system. Aside from certain interface considerations, these modules are largely independent and can therefore be implemented and tested separately. Detailed descriptions of these phases foilow.

3.1. Phase I : define input parameters

The task of this phase is to interact with the user to define all the names, variables, and values that will be needed later. The basic input mode is interactive with the system prompting the user at the terminal for needed information. While the basic mode provides flexibility, the input phase can be tedious. Thus we also provide a menu-driven mode where well-known element types together with their usual parameter values are pre-defined for user selection. A flexible and user-friendly input phase is a goal of the system.

The input handling features include :

(1) free format for all input with interactive prompting showing the correct input form;

(2) editing capabilities for correcting typing errors;

(3) the capability of saving all or part of the input for use later;

(4) the flexibility of receiving input either interactively or from a text file.

3.2. Phase II : Jacobian and [B] matrix computation

The strain-displacement matrix [B] is derived from symbolically defined shape functions in this phase. Let n be the number of nodes then

$$\mathbf{H} = (h_1, h_2, \ldots, h_n)$$

is the shape function vector whose components are the n shape functions h_1 through h_n. The value for the shape functions will be derived in a later phase. Here we simply compute with the symbolic names. Let r,s and t be the natural coordinates in the isoparametric formulation

and **HM** be a matrix

$$\mathbf{HM} = \begin{bmatrix} \mathbf{H},r \\ \mathbf{H},s \\ \mathbf{H},t \end{bmatrix}$$

where H,r stands for the partial derivative of H with respect to r. The Jacobian J is then

$$\mathbf{J} = \mathbf{HM} \; . \; [\mathbf{x},\mathbf{y},\mathbf{z}]$$

where x stands for the column vector $[x_1,..., x_n]$ etc. Now the inverse, in full symbolic form, of J can be computed as

$$\mathbf{J}^{-1} = \frac{\mathbf{invj}}{det(\mathbf{J})}$$

By forming the matrix **DH = (invj . HM)** we can then form the [B] matrix.

3.3. Phase III : shape function calculation

Based on the interpolation polynomials and nodal coordinates the shape function vector H is derived and expressed in terms of the natural coordinates r,s and t in the isoparametric formulation. Thus the explicit values for all h_i and all their partial derivatives with respect to r,s and t, needed in **HM** are computed here.

The input handling module, the derivation of the shape functions, the [B] matrix and its corresponding FORTRAN code, were done by Mr. P. Young [11] and later improved by Mr. H. Tan [8] as their master's thesis projects.

3.4. Phase IV: FORTRAN code generation for [B]

A set of FORTRAN subroutines for the numerical evaluation of the strain-displacement matrix [B] is generated for use with the NFAP package. The NFAP package is a large FORTRAN based system for linear and nonlinear finite element analysis. It is developed and made available to us by P. Chang of the University of Akron. It has been modified and made to run in FORTRAN 77 under UNIX.

FORTRAN assignment statements will be generated to define the components of the shape function vector H and the various partial derivatives. The [B] matrix can be generated with little difficulty. The resultant code for the plane 4-node element is shown in Figure 3. This code can be improved and many repeated computations avoided by adopting another

strategy discussed in section 5.

3.5. Phase V : compute and generate FORTRAN code for [K]

The inverse of the Jacobian, J, appears in [B]. By keeping the inverse of J as invj/det(J), the quantity det(J) can be factored from [B] and, denoting by [BJ] the matrix [B] thus reduced, we have

$$[K] = \int_{-1}^{1}\int\int \frac{[BJ]^T \cdot [D] \cdot [BJ]}{det(J)}\, dr\, ds\, dt$$

The determinant of the Jacobian involves the natural coordinates r,s and t. This makes the exact integration in the above formula difficult. We elect to evaluate det(J) at r=s=t=0 and factor it out of the integral. The resulting integrand involves only polynomials in r,s and t which are readily integrated. This approximation may not be reasonable for certain applications. In other applications the symbolic form of the stiffness matrix may be too large. In studying this problem, we have developed several techniques of general interest for reducing the size of automatically derived code and to increase its efficiency.

4. THE FORTRAN CODE GENERATOR

A separate module is developed for producing FORTRAN code from results derived symbolically. This package, called **GENTRAN** [9], has been developed to satisfy the needs of producing finite element code. However, it is also of independent interest as a general purpose code generator/translator. Rather than generating FORTRAN directly, **GENTRAN** generates RATFOR code. It can generate control-flow constructs, functions, subroutines and complete programs. **GENTRAN** can generate a program with or without a template" file.

5. CODE GENERATION TECHNIQUES

Let us discuss here some techniques we have applied to generate better FORTRAN code. Although these were used in finite element code generation, they are general techniques which should be helpful for other symbolic code derivation and generation applications.

5.1. Automatic expression labeling

Figure 1 contains straight forward FORTRAN code for two stiffness coefficient in the plane 4-node case. Figure 2 contains a different version of the code for the same coefficients. One can see that the latter is much more efficient. The key is to automatically generate and use the labeled expressions t0, t1 and t2 that appear repeatedly in the sk(1,1) and sk(1,2) computations. This means in the mathematical derivation of these coefficients, certain intermediate results should be generated with machine created labels. These results can be saved on an association list to prevent the re-computation and re-generation of the same expressions in subsequent computations. The following LISP function is used for this purpose.

```
(defun intermediate(operand alist fn labelname labelcnt file)
 (prog(exp label ans)
     (setq ans (assoc operand (cdr alist)))
     (cond (ans (return (cdr ans)))  ;; label previously defined
           (fn  (setq exp (apply fn (list operand))))
            (t   (setq exp operand)))

     ;; makelabel creates a new label and increments labelcnt
     (setq label (makelabel labelname labelcnt))

     ;; now generate assignment code
     (cond ((null file) (ratfor (list '(msetq) label exp)))
           (t (ratfor (list '(msetq) label exp) file)))

     ;; record operand-label pair in alist
     (setq alist (rplacd alist (cons (cons operand label) (cdr alist))))
     (return label)))
```

This function is called when automatic labeling is needed. Input parameters to "intermediate" are:

(1) operand: the expression on which an operation specified by the parameter "fn" is to be performed;

(2) alist: an association list of dotted pairs each in the form (operand . label), (initially nil);

(3) fn: the intended operation on the parameter "operand" (no operation if fn is nil);

(4) labelname: an atom which serves as a prefix for the automatically generated label;

(5) lab·lcnt: an integer count, associated with a given labelname, which is incremented after each new label is formed;

(6) file: a file to which any new code generated by "intermediate" will be appended.

5.2. Using subroutines in template files to eliminate repeated computations

As an example of this technique let us look at Figure 3 where the [B] code is shown. The code results from deriving the [B] directly in the LISP environment. But instead of computing [B] in LISP, we can generate the FORTRAN array, "gb", corresponding to **DH** as shown in Fig. 4. A FORTRAN subroutine (contained in the template file) is then used to fill the array [B] by simply taking an appropriate entry of the array "gb" or zero. This requires only 1/3 of the total computation as in Fig. 3.

In forming "gb", note also that another subroutine "inner" is used to form inner products of linear arrays.

5.3. Using symmetry by generating functions and calls

Upon re-examination of the mathematical derivations leading to the expressions for t0, t1, t2 etc. as contained in Fig. 2, we soon realize that the given problem has many symmetries that can be exploited for more efficient code. Since symmetries do arrise in practical problems, techniques for taking advantage of symmetry are of great interest.

For example, the expression x+y-z is related to x-y+z by symmetry. If we have a function F(x,y,z)=x+y-z then the latter expression is F(x,z,y). If F(x,y,z) is a large expression then we can simplify the resulting code generated by first generating the function definition for F(x,y,z) then generate calls to F with the appropriate arguments wherever F or its symmetric equivalent occurs. Here we are not talking about finding symmetric patterns in large expressions. Rather we want to make sure that the symbolic derivation phase preserve and use the symmetry in the given problem.

Let us take a detailed look at the symmetry involved in the [K] computation. To keep expressions simple, we again use the two dimensional element as our example. If we let P(x) = H,s.x and Q(x) = H,r.x then **invj** can be written as

$$\mathbf{invj} = \begin{bmatrix} P(y) & -P(x) \\ -Q(y) & Q(x) \end{bmatrix}$$

If **JROW(y,z)** denotes the 1st row of **invj**, **HMT** the transpose of **HM**, and **HMT[j]** the jth row of **HMT**, then

$$\mathbf{DH = invj.HM} = \begin{bmatrix} \mathbf{JROW(y).HMT[j]} \\ \mathbf{- JROW(x).HMT[j]} \end{bmatrix}$$

The matrix **DH** contains all the non-zero entries in [B]. Let us define a function FF.

$$\mathbf{FF(a,b,i,j) = JROW(a).HMT[i]*JROW(b).HMT[j].}$$

Note that FF(a,b,i,j)=FF(b,a,j,i). Then the matrix [K] can be constructed in terms of

$$G_{ij} = \int_{-1}^{1}\int FF(\mathbf{a,b},i,j)\ dr\ ds$$

Each G_{ij}(a,b) is a sum of terms in the form

$$p_i q_j(\mathbf{a,b}) = \int_{-1}^{1}\int P(\mathbf{a}).\mathbf{HMT}[j]\ Q(\mathbf{b}).\mathbf{HMT}[i]\ \ dr\ ds.$$

Thus we see in Fig. 5 that functions g11, p1p1, q1q1 and q1p1 are automatically generated with appropriate declarations in RATFOR. Then calls to these functions are generated to compute t0, t1 and t2.

5.4. Optimizing the final expressions before code generation

Our experiments with the REDUCE code optimizer has shown that, in addition to the above techniques, a systematic common subexpression search can help reduce code size and increase code efficiency. For more details see [10].

```
sk(1,1) = 4*((4*m1*y4**2+(-8*m1*y3-8*m3*x4+8*m3*x3)*y4+4*m1*y3**2+
1   (8*m3*x4-8*m3*x3)*y3+4*m6*x4**2-8*m6*x3*x4+4*m6*x3**2)/3.0+4*m1
2   *y4**2+(-8*m1*y2-8*m3*x4+8*m3*x2)*y4+(4*m1*y3**2+(-8*m1*y2-8*m3
3    *x3+8*m3*x2)*y3+4*m1*y2**2+(8*m3*x3-8*m3*x2)*y2+4*m6*x3**2-8*m6
4   *x2*x3+4*m6*x2**2)/3.0+4*m1*y2**2+(8*m3*x4-8*m3*x2)*y2+4*m6*x4*
5   *2-8*m6*x2*x4+4*m6*x2**2)/detk
 sk(1,2) = 4*((4*m3*y4**2+(-8*m3*y3+(-4*m6-4*m2)*x4+(4*m6+4*m2)*x3)
1   *y4+4*m3*y3**2+((4*m6+4*m2)*x4+(-4*m6-4*m2)*x3)*y3+4*m5*x4**2-8
2   *m5*x3*x4+4*m5*x3**2)/3.0+4*m3*y4**2+(-8*m3*y2+(-4*m6-4*m2)*x4+
3   (4*m6+4*m2)*x2)*y4+(4*m3*y3**2+(-8*m3*y2+(-4*m6-4*m2)*x3+(4*m6+
4   4*m2)*x2)*y3+4*m3*y2**2+((4*m6+4*m2)*x3+(-4*m6-4*m2)*x2)*y2+4*m
5   5*x3**2-8*m5*x2*x3+4*m5*x2**2)/3.0+4*m3*y2**2+((4*m6+4*m2)*x4+(
6   -4*m6-4*m2)*x2)*y2+4*m5*x4**2-8*m5*x2*x4+4*m5*x2**2)/detk
```

Fig. 1 Code for two stiffness coefficients

```
 t0 = (16*y4**2+(-8*y3-24*y2)*y4+8*y3**2-8*y2*y3+16*y2**2)/3.0
 t1 = -((16*x4-4*x3-12*x2)*y4+(-4*x4+8*x3-4*x2)*y3+(-12*x4-4*x3+16*
1   x2)*y2)/3.0
 t2 = (16*x4**2+(-8*x3-24*x2)*x4+8*x3**2-8*x2*x3+16*x2**2)/3.0
 sk(1,1) = 4*(m6*t2+2*m3*t1+m1*t0)/detk
 sk(1,2) = 4*(m5*t2+m6*t2+m2*t1+m3*t0)/detk
```

Fig. 2 Improved code for the two stiffness coefficients

```
b(1,1) = (-2*y4+r*(2*y3-2*y4)+s*(2*y2-2*y3)+2*y2)/det
b(1,2) = 0
b(1,3) = (r*(2*y4-2*y3)+s*(2*y4-2*y1)+2*y3-2*y1)/det
b(1,4) = 0
b(1,5) = (2*y4+s*(2*y1-2*y4)+r*(2*y2-2*y1)-2*y2)/det
b(1,6) = 0
b(1,7) = (s*(2*y3-2*y2)-2*y3+r*(2*y1-2*y2)+2*y1)/det
b(1,8) = 0
b(2,1) = 0
b(2,2) = (r*(2*x4-2*x3)+2*x4+s*(2*x3-2*x2)-2*x2)/det
b(2,3) = 0
b(2,4) = (r*(2*x3-2*x4)+s*(2*x1-2*x4)-2*x3+2*x1)/det
b(2,5) = 0
b(2,6) = (s*(2*x4-2*x1)-2*x4+2*x2+r*(2*x1-2*x2))/det
b(2,7) = 0
b(2,8) = (2*x3+s*(2*x2-2*x3)+r*(2*x2-2*x1)-2*x1)/det
b(3,1) = (r*(2*x4-2*x3)+2*x4+s*(2*x3-2*x2)-2*x2)/det
b(3,2) = (-2*y4+r*(2*y3-2*y4)+s*(2*y2-2*y3)+2*y2)/det
b(3,3) = (r*(2*x3-2*x4)+s*(2*x1-2*x4)-2*x3+2*x1)/det
b(3,4) = (r*(2*y4-2*y3)+s*(2*y4-2*y1)+2*y3-2*y1)/det
b(3,5) = (s*(2*x4-2*x1)-2*x4+2*x2+r*(2*x1-2*x2))/det
b(3,6) = (2*y4+s*(2*y1-2*y4)+r*(2*y2-2*y1)-2*y2)/det
b(3,7) = (2*x3+s*(2*x2-2*x3)+r*(2*x2-2*x1)-2*x1)/det
b(3,8) = (s*(2*y3-2*y2)-2*y3+r*(2*y1-2*y2)+2*y1)/det
```

Fig. 3 FORTRAN code for [B]

```
gb(1,1) = inner(jinv1,hm1)/det
gb(1,2) = inner(jinv1,hm2)/det
gb(1,3) = inner(jinv1,hm3)/det
gb(1,4) = inner(jinv1,hm4)/det
gb(2,1) = inner(jinv2,hm1)/det
gb(2,2) = inner(jinv2,hm2)/det
gb(2,3) = inner(jinv2,hm3)/det
gb(2,4) = inner(jinv2,hm4)/det
```

Fig. 4 the array "gb"

```
t0=g11(y,y)
t1=-g11(x,y)
t2=g11(x,x)
sk(1,1)=(m1*t0+2*m3*t1+m6*t2)/detk
sk(1,2)=(m3*t0+m2*t1+m6*t1+m5*t2)/detk

function p1p1(aa,bb)
implicit real*8 (a-h,o-z)
dimension aa(4),bb(4)
v0=vi(12,aa); v1=vi(12,bb); v2=vi(10,bb); v3=vi(10,aa)
return (16.0/3.0*v0*v1+16.0/9.0*v2*v3)
end

function q1p1(aa,bb)
implicit real*8 (a-h,o-z)
dimension aa(4),bb(4)
v0=vi(9,aa); v1=vi(12,bb); v2=vi(10,aa); v3=vi(10,bb)
return (-4*v0*v1+-4.0/3.0*v1*v2+-4.0/3.0*v0*v3+-4.0/9.0*v2*v3)
end

function q1q1(aa,bb)
implicit real*8 (a-h,o-z)
dimension aa(4),bb(4)
v0=vi(9,aa); v1=vi(9,bb); v2=vi(10,aa); v3=vi(10,bb)
return (16.0/3.0*v0*v1+16.0/9.0*v2*v3)
end

function g11(aa,bb)
dimension aa(4),bb(4)
return (p1p1(aa,bb)+q1p1(aa,bb)+q1q1(aa,bb)+q1p1(bb,aa))
end
```

Fig. 5 functions and calls in generated RATFOR code

REFERENCES

[1] M. M. Cecchi and C. Lami, "Automatic generation of stiffness matrices for finite element analysis", Int. J. Num. Meth. Engng 11, pp. 396-400, 1977.

[2] T. Y. Chang and K. Sawamiphakdi "Large Deformation Analysis of Laminated Shells by Finite Element Method", Comput. Structures, Vol. 13, 1981.

[3] T. Y. Chang, "NFAP - A Nonlinear Finite Element Analysis Program Vol. 2 - User's Manual", Technical Report, College of Engineering, University of Akron, Akron Ohio, 1980.

[4] J. K. Foderaro and R. J. Fateman, "Characterization of x Macsyma", Proceedings, ACM SYMSAC'81 Conference, Aug. 5-8, Snowbird, Utah, pp. 14-19, 1981.

[5] A. R. Korncoff and S. J. Fenves, "Symbolic generation of finite element stiffness matrices", Comput. Structures, 10, pp. 119-124, 1979.

[6] A. K. Noor and C. M Andersen, "Computerized Symbolic Manipulation in Nonlinear Finite Element Analysis", Comput. Structures 13, pp. 379-403, 1981.

[7] A. K. Noor and C. M. Andersen, "Computerized symbolic Manipulation in structural mechanics-progress and potential", Comput. Structures 10, pp. 95-118, 1977.

[8] H. Q. Tan, "Design and Implementation of an Improved Finite Element Code Generator", master's thesis, Dept. Mathematical Sciences, Kent State University, Kent Ohio, 1984.

[9] P. S. Wang and B. Gates, "A LISP-based RATFOR Code Generator", Proceedings, the Third Users Conference, August, 1984.

[10] P. S. Wang, T. Y. P. Chang and J. A. van Hulzen, "Code Generation and Optimization for Finite Element Analysis", Proceedings, EUROSAM'84, London, England, July 9-11, 1984.

[11] P. Y. Young, "Automatic Finite Element Generator", master's thesis, Dept. Mathematical Sciences, Kent State University, Kent Ohio, 1983.

[12] UNIX programmer's manual, Vol. I and II, Seventh Edition, Bell Telephone Laboratories, Inc., Murray Hill, New Jersey, 1979 .

SOME APPLICATIONS OF SYMBOLIC MANIPULATION IN BIOMATHEMATICS

Raymond Mejia
Intramural Research, NHLBI
and Mathematical Research Branch, NIAMDDK
National Institutes of Health
Bethesda, MD 20205

Abstract

Symbolic manipulation, and MACSYMA in particular, has been used to solve or simplify a number of models of biological phenomena. Some uses described previously include: (1) matrix operations and FORTRAN code generation preliminary to the numerical solution of a model of the mammalian kidney [1,2], (2) parameter sensitivity analysis of a central core model for urine formation [3]. Other applications include the manipulation of statistical models that do not have a closed form solution and the manipulation of differential forms that describe certain probability distributions. A recent use of symbolic manipulation has been in the development of a preliminary model of the formation of the septum in the morphogenesis of the yeast, Saccharomyces cerevisiae.

The formation of the septum between mother and daughter cells of the yeast has been correlated with the deposition of chitin in the cell wall [4,5]. A model for the growth of the septum, developed in collaboration with Drs. J. González-Fernández, L. Lara-Carrera and E. Cabib, describes this growth as a function of curvature. Thus, given a space curve at time t with the equation

$$x^t = x^t(s) \qquad (1)$$

with $t \geq 0$ and the parameter s being arc length, we define the curvature of x^t at the point s_o to be

$$\frac{1}{R} = \lim_{\Delta s \to 0} \left| \frac{\Delta \theta}{\Delta s} \right| \qquad (2)$$

where $\Delta\theta$ is the angle between the tangents at the points s_o and $s_o + \Delta s$.

For symmetric growth and $x^t \in C^2(E^2)$, the growth may be written as

$$G(x^t) = \int \frac{B}{R}\, ds / \int ds \qquad (3)$$

where B is constant, and where we choose a positive orientation to be away from the origin.

The position of the curve at time t_1 may then be computed using the equation

$$x^{t_1} = x^{t_0} + \int_{t_0}^{t_1} G(x^t)\, dt. \qquad (4)$$

Another use of symbolic manipulation has been in the calculation of solute concentration profiles in permeable flow tubes. An example is the computation, in collaboration with Drs. M. Knepper and R. Star, of radial gradients due to diffusion in axially symmetric flows, where there is production or consumption of species due to chemical reactions [6]. The model is described by Laplace's equation in cylindrical coordinates as follows:

$$-D_i \nabla^2 C_i + S_i = 0 \qquad (5)$$

where D_i and C_i are the diffusion coefficient and the concentration of species i, respectively, and

$$S_i = \sum_j S_i^j = \sum_j (k_{-j} C_{\cdot} - k_j C_i) \qquad (6)$$

are the reaction equations for each species with rates k_j and k_{-j} for the jth reaction.

Solutions of (5) with the radius r, $0 \leq r \leq a$, symmetry at $r = 0$, and a radiation condition with Michaelis-Menten kinetics at $r = a$ are considered. Henderson-Hasselbach algebraic equations for titrating buffers complete the model.

References

1. R. Mejia and J.L. Stephenson (1979). "Symbolics and numerics of a multinephron kidney model," Proc. 1979 MACSYMA Users' Conference, Washington, DC.

2. R. Mejia and J.L. Stephenson (1979). "Numerical solution of multinephron kidney equations," J. Computational Phys. 32:235.

3. R. Mejia (1982). "Sensitivity analysis of a central core model of renal concentration," Navy MACSYMA Users Mini-Symposium, Carderock, MD.

4. E. Cabib and B. Bowers (1975). "Timing and function of chitin synthesis in yeast," J. Bacteriol. 124:1586.

5. M. Vrsánska, Z. Krátky, P. Biely and S. Machala (1979). "Chitin structures of the cell walls of synchronously grown virgin cells of Saccharomyces cerevisiae," Zeitschrift f. Allg. Mikrobiologie, 19:357.

6. K.W. Wang and W.M. Deen (1980). "Chemical kinetic diffusional limitations on bicarbonate reabsorption by the proximal tubule," Biophys. J. 31:161.

APPLICATION OF MACSYMA TO A BOUNDARY VALUE PROBLEM ARISING IN NUCLEAR MAGNETIC RESONANCE IMAGING

J.F. Schenck
M.A. Hussain
General Electric Company
Corporate Research and Development
Schenectady, NY

A major issue of NMR imaging is the effect of induced eddy currents within the patient's body on the imaging process. Such currents alter the distribution of the radio frequency field, thereby changing the pattern of excitation. Furthermore, they limit the penetration of high-frequency energy into and out of the body, thereby providing a potential limitation to the imaging at high field strength.

For this reason, it is desirable to have a soluble, closed-form model that can provide a reasonable guide to the distribution of induce eddy currents and to characterize the power deposition associated with them. Such a model was provided by Bottomley and Andrew in 1978 and extended somewhat by Mansfield in 1981.

These previous calculations, however, model the human body as an infinitely long cylinder in a uniform external field. Additional pertinent phenomena can be exhibited by a spherical model which overcomes the limitation of two-dimensional models.

The problem solved is that of a uniform sphere with given homogeneous values for conductivity and permitivity. The problem is set up using the standard methods of Mie scattering theory. The resultant boundary value can be readily solved using MACSYMA. The results can be manipulated readily to compute resonant frequencies, damping factors and the impedance properties of the transmitting coils that are coupled to the spherical model of the human tissue.

The analysis suggest methods to extend the calculations to more refined models.

The results of calculations have a direct application to NMR imaging system design.

PROVIDING A COMPLEX NUMBER ENVIRONMENT FOR MACSYMA AND VAXIMA*

Johnnie W. Baker
Department of Mathematical Sciences
Kent State University
Kent, OH 44242

Oberta A. Slotterbeck
Department of Mathematical Sciences
Hiram College
Hiram, OH 44234

Department of Computer Sciences**
University of Texas at Austin

Abstract

While looking at the problem of handling contour integration symbolically, the authors discovered that the underlying complex number environment of MACSYMA was awkward at best and sometimes incorrect. In this paper, the authors discuss some of the problems found and describe their revision of the MACSYMA simplifier to provide a complex number environment.

1. THE PROBLEMS WITH THE CURRENT SIMPLIFIER.

MACSYMA was designed originally to support a real number environment. Complex number support, such as it is, is awkward at best and sometimes incorrect.

* This work was supported for both authors, in part, by The Department of Energy under Grant DE-AS02-76ER02075.
** The authors wish to thank members of the department for their gracious hospitality during their 1983-84 sabbaticals.

Basically, MACSYMA treats the number i as an indeterminate whose square is negative one. Consequently, the expression

$$(2 + 3i) + (1 + 2i)$$

is simplified by the system to $3 + 5i$, just as the expression

$$(2 + 3x) + (1 + 2x)$$

is system-simplified to $3 + 5x$. However, the expression

$$(5 + 2i)\,(3 - 4i)$$

is not automatically changed to the more natural $23 - 14i$ because the underlying MACSYMA philosophy does not support automatic expansion of expressions.

In the real number environment provided by MACSYMA, real constants in expressions are combined automatically into simpler forms while full accuracy is maintained for rational numbers and the specified precision is carried for floatnums or bigfloats. As typical examples,

(1) $2 * 3$ becomes 6

(2) $2 / 3$ becomes $\frac{2}{3}$

(3) 2^3 becomes 8

(4) 2.0/3.0 becomes 0.666...67 (up to specified precision)

(5) $e^{2.0}$ becomes 7.389... (up to specified precision)

By way of contrast, each of the following is returned unsimplified by the existing MACSYMA simplifier:

(1) $(2 + 3i)\,(1 + 2i)$

(2) $(2 + 3i) / (1 + 2i)$

(3) $(2 + 3i)^3$

(4) $(2.0 + 3.0i) / (1.0 + 2.0i)$

(5) $e^{(2.0 + 3.0i)}$

A more natural approach, in line with the MACSYMA philosophy for dealing with real number simplifications, would be to reduce the above automatically to

(1) -4 + 7i

(2) $\frac{8}{5} - \frac{i}{5}$

(3) - 46 + 9i

(4) 1.6 - 0.2 i

(5) 1.0427436562359O4i - 7.315110094901103 (assuming 16 digit precision)

A few user functions, such as RECTFORM, POLARFORM, REALPART,IMAGPART, CARG, AND CABS in the MACRAK package, exist in MACSYMA to manipulate complex expressions. In addition, some packages, such as Paul Wang's LIMIT package [9], allow some complex expressions as input. However, the existing environment does not provide a good working environment for the user interested in working with complex-valued functions. Although (5 + 2i) (3 - 4i) can be user-simplified to 23 - 14i with RECTFORM (or EXPAND or RATSIMP), even simple expressions such as

$$(2 + 2i)^3 (1 + 2i)^2 (a + bi)^4$$

cannot be displayed in the more natural form

$$(102 - 211i) (a + bi)^4$$

without effort from the user.

The standard exponential, logarithmic, power, trigonometric, hyperbolic, and inverse functions are already implemented in MACSYMA. Since the present MACSYMA simplifier assumes that these functions have the same properties as their real counterparts, some of the results it produces when simplifying expressions involving these functions are either not correct or not adequately simplified. Some typical examples that illustrate these problems are listed below:

(1) $\log(e^{3+4i})$ is simplified by the system to 4i + 3 instead of 3 + (4 - 2 pi)i.

(2) $\log((3 + 4i)^{25})$ is simplified to 25 log(4i + 3) instead of 25 log(3 + 4i) - 8 pi i.

(3) $((-2 + 3i)(-4 + 5i))^{2/3}$ is simplified incorrectly to $(3i - 2)^{2/3} (5i - 4)^{2/3}$.

The proper value is $(-2 + 3i)^{2/3} (-4 + 5i)^{2/3} e^{(-4/3 \text{ pi i})}$.

(4) $(e^{5-4i})^{4i+3}$ is simplified incorrectly to $e^{(4i-5)(-4i-3)}$, i.e. e^{31+8i},

instead of the correct value of $e^{(31-8 \text{ pi}) + (8-2 \text{ pi})i}$.

Consequently, the absolute value of the original expression in (4) is returned as e^31

instead of e^(31-8pi).

The difficulties enumerated above lie with the design of the MACSYMA simplifier, which was written to support a real number environment, not a complex number environment. By looking at some of the changes in the code over the years and the packages, such as the MACRAK package, which have been added, it appears that some knowledge about complex numbers has been added to the simplifier on an ad-hoc basis as the need arose. Although many users of MACSYMA feel that complex number operations are supported by the system, the preceding calculations and comments illustrate that this is not the case.

The absence of a complex number environment also leads to some problems in the real number environment. While each of these can be dismissed with "let the user beware", the addition of a complex environment in MACSYMA can help prevent these problems. Three examples follow:

(1) $\log(x^2)$ simplifies to 2 log(x). Evaluating this expression at $x = -1$ yields 2 log(-1), which is 2 pi i. However, $\log((-1)^2) = 0$.

(2) $\log((\mathrm{sqrt}(x))^2)$ simplifies automatically to log(x). If we are in a real environment, x must be nonnegative. However, MACSYMA accepts negative values for x without complaint. Evaluation of this expression at $x = -1$ produces log(-1), which has pi i as its value.

(3) Sqrt(x) evaluated at $x = -1$ produces the non-real value of i.

In the last two examples, the answers are correct. However, we have entered only real data into a simplifier that basically supports only real number calculations and produced, without warning, a non-real answer. This illustrates a basic problem with the real number environment: It is not closed under normal mathematical operations. It is easy to produce examples where a user could enter only real data, obtain non-real intermediate results which are never displayed, and obtain an incorrect real number for an answer. It is also easy to imagine situations where a user has sqrt(x) occur in some expression and then evaluates x at a negative value many steps later when no square root function is present to warn that this evaluation in a real number environment is illegal.

We have talked to many MACSYMA users that believe the present simplifier fully supports complex number operations. Although they may realize that such calculations are awkward, they do not doubt the accuracy of complex number calculations in MACSYMA using the current simplifier. Since it is difficult to hand-check complicated complex number operations, there is no easy way to observe that answers obtained may be incorrect. This is obviously an undesirable and dangerous situation. Moreover, since some physicists, engineers, and mathematicians need to do difficult complex number environment computations, it is imperative that a complex number environment be supported in MACSYMA and other general purpose computer algebra systems.

2. AN OVERVIEW OF THE REVISED SIMPLIFIER: ITS DESIGN AND IMPLEMENTATION

Our original goal was to provide additional user-callable functions to MACSYMA that could be utilized whenever a user desired a complex number environment. This goal was soon abandoned when, as discussed earlier, it was discovered that the present simplifier "messed up" some complex number expressions before they were returned to the top level. Consequently, it was necessary to redesign and rewrite major sections of the simplifier code.

One could design a high level algorithm for a complex number simplifier independent of an existing system, but a more challenging problem is to design and implement one within an existing computer algebra system while attempting to maintain the original simplification philosophy of that system. By embedding the new simplifier directly into the existing code we can test our design for accuracy, speed, completeness, acceptability of returned forms, and interaction with existing packages and demo files. We chose to embed in the VAXIMA version of MACSYMA, specifically UNIX MAXSYMA release 304 running under UNIX 4.2 BSD #19.

The user can enter the complex number domain for simplification by setting a switch COMPDOMAIN to true. This has the effect of routing the expression during the simplification process to the new code when necessary. This approach was chosen to minimize the timing impact of the complex domain simplifier upon the simplification process in general. (As the entire complex domain simplifier code is not complete, we have not yet been able to determine what the final impact will be. However, preliminary timing results do not indicate a significant degradation of response time.)

In the complex domain simplifier, complex number constants are simplified in line with the MACSYMA philosophy for dealing with real number constants. Complex number constants can occur in either rectangular form (i.e., a + ib with a and b real), polar form (i.e., r e^(i arg) with r and arg real) or in mixed forms such as (a + ib) e^(c+id). Moreover, expressions can be constructed from any of these general forms, such as

$$(2 + i)\, e^{i \sin(x)} \text{ and } \log(z)\, e^{w}$$

In general, no automatic conversion from either rectangular form or polar form to the other form is provided. Users can manually use functions such as the present RECTFORM and POLARFORM to convert from one form to another. Some preference is given to rectangular forms in that a few special numbers in polar form with a simple rectangular form (e.g., 3 e^(pi i/2) = 3 i) are automatically converted to rectangular form. Likewise, "corrupted" numerical expressions such as 2 e^(3.5i) which contain floatnums or bigfloats are converted automatically to an approximately equivalent rectangular form. In fact, all corrupted quantities are numerically simplified as much as possible. This is consistent with the current simplifier's handling of real expressions involving floatnums or bigfloats. Some examples using rectangular coordinate representation for constants are shown in Section 3.

Some complex-valued functions, such as the logarithm function, are multi-valued. A critical issue is which branch(es) should the simplifier provide? We default to the principal branch. However, some users may wish to choose a specific branch or to maintain information about all possible branches. This service is provided by the GENARG switch. When GENARG is set to TRUE, integer variables %n1, %n2, ..., are generated, as needed. For example, when simplifying a logarithmic expression such as

```
log(5 * sqrt(3) * %i - 5)
```

the default simplification is

```
2 %i %pi
--------  +  log(10)
   3
```

while the GENARG simplification is

```
  2
( --- + 2 %n1 ) %pi %i + log(10).
  3
```

Automatic integer variable reductions such as replacing %n1 +%n2 with %n3 are included.

When simplifying complex expressions, it is often necessary to know if certain subexpressions are integer or real and if they are negative, nonnegative, or positive. In order to answer questions like this, variables can be tagged as real, integer, positive, etc. Untagged variables are handled as complex variables and require the use of the more general simplifying formulas. Moreover, if a variable is tagged as a nonnegative real, then an evaluation or substitution setting x to a value such as -1 produces a warning message to the user. However, the requested evaluation or substitution will still be produced, and the warning message can be ignored, if desired.

Germane to the above question is the need to determine the sign of real number expressions when evaluating, for example, a logarithm. Our simplifier tries to determine the sign of real numbers by utilizing numerical evaluation. If, during the numerical evaluation of any part of an expression, an overflow or underflow occurs, the simplification process automatically switches to more simplistic techniques (i.e., are all the terms positive or are all the terms negative in a sum?). If these more simplistic techniques fail also, a "don't know" answer is returned. In this case, further simplification of the original expression may be impossible or restricted and the expression would be returned unsimplified. Some examples showing simplifications with the complex logarithm functions are shown in Section 3.

We are implementing the usual complex mathematical functions for this complex environment. In particular, the complex logarithm, exponent, power, absolute value, real part, imaginary part, argument, rectangular form, polar form, conjugate, and square root will be available. Most of these are available in a restricted form currently, but the code related to many of these functions has to be either modified or rewritten so that these functions will perform correctly, have their expected properties, and be

easy to use in this new environment.

A complete integration of a complex environment into MACSYMA requires that some additional functions in MACSYMA be either modified or replaced. For example, the EV function provides a numerical evaluation for real number expression (using NUMER), but does not provide numerical evaluation for many complex number expressions. Some of the packages that presently run correctly with the present environment may not run correctly with a complex environment simplifier. While some of the required modifications or revisions may be natural to include as part of the development of a complex simplifier (e.g., the extension of the EV function to handle numerical evaluation of complex expressions), other projects such as modifying existing packages to take advantage of a complex number environment will be left to researchers interested in working in those particular areas. Initially, packages that do not run correctly in the new environment will be blocked until the user switches back to the present real environment simplifier.

3. SOME EXAMPLES.

The following MACSYMA-like worksheet illustrates some of the features of the current complex number simplifier. The symbol (ci) denotes the expression entered by the user, the symbol (di) denotes the expression returned by the current MACSYMA simplifier, and the symbol (ei) denotes the expression returned by the complex number simplifier.

```
(c1) /* Complex constants should be simplified automatically
just as real constants are. */

z*(-2+4*%i)^(-2)/(-3*(1-6*%i)^3*(4/5)*%i*(-%i+2))*w*(4+2*%i)*
x^3*(-3-%i)^(-3)*%pi*f(2*(%i-4)*%i*(t+s*%i)*(-3-%i)*(2+4*%i)^(-2));

                        2 (- %i - 3) (%i - 4) %i (t + %i s)      3
     5 %i (2 %i + 4) %pi f(-----------------------------------) w x  z
                                          2
                                 (4 %i + 2)
(d1) ----------------------------------------------------------------
                            3            3            2
            12 (1 - 6 %i)  (- %i - 3)  (2 - %i) (4 %i - 2)
```

```
          391         1179 %i                   11     7 %i                        3
(e1)   (----------- - -------------) %pi f((----- - ------) (t + %i s)) w x  z
        607836000   202612000                 10     10
```

(c2) /* Floatnums and bigfloats should propagate through an expression */

```
(3.2*%i+6.4)^7+(-2*%i-6)*(3*%i+2)^3/(2-3*%i)^2;
```

```
                                                        3
                                   7   (- 2 %i - 6) (3 %i + 2)
(d2)             (3.2 %i + 6.4)  + --------------------------
                                                  2
                                          (2 - 3 %i)

(e2)             - 99623.48978791007 %i - 955212.1230801042
```

```
(c3) (1.6b-3+2.4*%i)/5*%i*(a+b*%i)*6/7*(a+b*%i);
```

```
                                                              2
                      6 %i (2.4 %i + 1.6b-3) (%i b + a)
(d3)                  ---------------------------------
                                      35

                                                                            2
(e3)    (2.742857142857143b-4 %i - 4.114285714285714b-1) (%i b + a)
```

(c4) /* The principal branch of the multi-valued logarithm function can be returned */

```
log(-2);

(d4)                        log(- 2)

(e4)                     %pi %i + log(2)

(c5) log(%i);

(d5)                        log(%i)

                            %pi %i
(e5)                        ------
                               2
```

```
(c6) log(-2/3*%e^-3*%pi);

                                  - 3
                            2 %e     %pi
(d6)                   log(- -------------)
                                  3

                         2 %pi
(e6)                 log(-------) + %i %pi - 3
                           3

(c7) log(5*(3*%i-2)*(5+%i)/13);

                     5 (%i + 5) (3 %i - 2)
(d7)             log(----------------------)
                              13

                                      3 %i %pi
(e7)                 log(5 sqrt(2)) + --------
                                         4

(c8) log(-%e*%pi*12-sqrt(3)*4*%e*%i*%pi);

(d8)          log(- 4 sqrt(3) %e %i %pi - 12 %e %pi)

                                          5 %i %pi
(e8)               log(8 sqrt(3) %pi) - ---------- + 1
                                             6

(c9) /* The logexpand switch should inhibit expansion on
expressions for which the result would be incorrect,
but expand properly on other expressions. */

logexpand:super $

(c10) log(2*%e^(-9*%i));

(d10)                    log(2) - 9 %i

(e10)                log(2) + %i (2 %pi - 9)
```

[5] R.J. Fateman, MACSYMA's general simplifier: philosophy and operation MACSYMA User's Conference (V.E. Lewis, editor), MIT Lab Publication, 1979, 563-582.

[6] J.P. Fitch, On algebraic simplification ,*Comput. J. 16/1*, 23-27, (1973) 23-27.

[7] J. Korpela, Some problems connected with ambiguous expressions, *SIGSAM Bull. 11*,(August 1977) 7-9.

[8] J. Moses, Algebraic simplification, a guide for the perplexed, *Comm. ACM 14*, (1971) 527-538.

[9] P.S. Wang, Automatic computation of limits, Second Symposium on Symbolic and Algebraic Manipulation (S.R. Petrick, Editor), New York:ACM, 1971, 458-464.

Simplifying Large Algebraic Expressions by Computer

Richard L. Brenner

Symbolics, Inc.
Cambridge Research Center

ABSTRACT

Computer simplification of very large algebraic expressions by direct methods is often impractical because of the exhaustion of available resources. Some of the causes of this difficulty are discussed, and a method of circumventing it for certain types of problems is presented. The method has been implemented for the Computer Algebra system MACSYMA and is being used in the calculation of one-loop corrections to the hadronic decay rate of quarkonium in Quantum Chromodynamics.

1. Introduction

Anyone who tries to simplify by computer a large algebraic expression learns quickly how easy it is to exhaust the available resources of computer memory, disk memory or computer time. Computer Algebra systems are capable of simplifying small problems much more rapidly and accurately than people can, but often their margin of superiority declines as the size of the problem increases. Although the cause of this difficulty varies from problem to problem and from program to program, their is a procedure, described in this paper, that enables the computer to reduce a large class of expressions that might otherwise prove difficult. This method enhances the capabilities of Computer Algebra systems by exploiting the ability of the machine to recall the results of intermediate calculations, and the ability of the user to invent techniques specialized to particular simplification problems. Although it is general, the method is flexible enough to accomodate specific features of the problem at hand because it is merely a

The reduction of a porphyrytic expression typically requires extensive processing of the quantities in the second class, followed by rational simplification of the quantities from the first class. Since one usually focusses upon the second class of quantities first, they are called the *foreground* kernels[3]. The rest of the expression is called *background*.

A simple example may clarify this terminology, and is useful in the discussions in the remainder of this section. Consider the set of functions defined by [4]:

$$F(0,x) = -\log(\frac{x-1}{x}) \tag{2.1a}$$

$$F(n,x) = -x^n \log(\frac{x-1}{x}) - \sum_{k=1}^{n} \frac{x^{n-k}}{k} \qquad n = 1,2,3 \cdots \tag{2.1b}$$

These functions arise when performing loop momentum integrals in perturbative treatments of certain quantum field theories in the context of dimensional regularization of ultraviolet divergences. They have a number of interesting properties, including:

$$F(n,x) = x\,F(n-1,x) - \frac{1}{n} \tag{2.2}$$

In the discussion below, we shall use this recursive definition of $F(n,x)$ in preference to (2.2) because the recursive form better illustrates the important concepts. For the purposes of illlustration, we shall also need a slightly more complicated structure:

$$Y(m,n,x,y) = \frac{y^m F(n,x) - x^n F(m,y)}{x\,y} \tag{2.3}$$

In our terminology, the right hand side of the equation

$$Y(3,5,x,y) = \frac{y^3 F(5,x) - x^5 F(3,y)}{x\,y} \tag{2.4}$$

is porphyritic. Its foreground kernels are $F(5,x)$ and $F(3,y)$, and the background variables are x and y. We shall return to this example later to illustrate several possible reduction strategies.

A diagrammatic representation of these algebraic structures is also useful for discussing reduction strategies. Let us represent an algebraic expression as an n-ary tree. We can represent the rational operators *plus*, *times*, and *exponentiation* as nodes of this tree, and kernels as the leaves of the tree. We label a node p for *plus*, t for *times*, and e for *exponentiation*. We also assume that a complete set of rules determine the order of the branches attached to these nodes, so that we are assured that the mapping from the set of rational expressions to the set of trees is one-to-one. Thus we can represent the expression

$$x + y + Y(3,5,x,y) + \frac{1}{x} \tag{2.5}$$

as shown in Figure 1. If we apply (2.3) to the above expression, we obtain

$$x + y + \frac{y^3 F(5,x) - x^5 F(3,y)}{xy} + \frac{1}{x} \tag{2.6}$$

which is represented as shown in Figure 2.

Although these diagrams are useful for modeling the expressions themselves, they are not intended to represent the actual contents of any part of a computer memory. The contents of memory cells that are used to represent these expressions depend strongly on the computer algebra system being used. Nevertheless, this diagrammatic representation of the algebraic structure does provide a valuable conceptual framework, and can be useful for judging the complexity of the structures in question.

Now that we have established a suitable language, we turn to a discussion of strategies for reducing porphyritic algebraic expressions. The most significant characteristic of such reduction efforts is the practical difficulty of actually performing such a computation. When confronted

with a porphyrytic expression to simplify, one's first impulse is to apply the entire reduction procedure to the entire expression. If the reduction procedure is relatively simple, and the expression is relatively compact, this direct approach is perhaps the best. But when a large expression is being simplified, and the simplification procedure is relatively complicated, it may be wise to segment the calculation. This segmentation may be advisable for a variety of reasons:

[1] Non-fatal errors of conception or execution of a calculation may be discovered while the calculation is in progress. If the calculation has been segmented, then there is the possibility that only certain portions of it require correction. If the calculation has not been segmented, recovery of correct partial results may be difficult.

[2] The procedure may take so long to execute that much effort could be lost if the operation of the computer were interrupted due to either a programming error, hardware malfunction, or scheduled maintenance of equipment. This is possible, since it is not unusual for the time scale of a simplification effort to approach the scale of the mean time between failures for the equipment being used for that effort.

[3] Space may be at a premium, because of the size of the expression itself or the size of the programs that are required to operate on the expression, or both. For machines with limited address space, the size of the calculation may force segmentation.

[4] Most important, it may be possible to use intermediate results for parts of the calculation in subsequent parts of the calculation. If intermediate results have been saved, they can easily be extracted for later use. This is difficult unless the segmentation has been done systematically. This capability can greatly increase the efficiency of any procedure.

For these reasons, most large reduction efforts are eventually segmented, often by necessity if not by choice. In the remainder of this section, we shall describe an approach to this segmentation that systematically addresses the problems indicated abovew, while providing significant increases in efficiency. We shall carefully characterize the types of problems to which this approach is best suited, and compare the efficiency of this method to that of alternative methods. A family of MACSYMA programs called LTAB provides one example of a possible implementation of this segmentation, and it is the subject of Section 3.

To understand the need for, and the advantages of segmenting a reduction problem, it is necessary to examine the sourcess of the difficulties of reducing porphyritic expressions. Problems associated with reducing porphyritic expressions are duc in large part to the well-known difficulty of simplifying rationally any large expression that contains many distinct kernels. The difficulty usually appears as a choice between rational simplification of a very large expression in only a few variables, and rational simplification of several smaller expressions involving large numbers of variables. In a typical situation, one begins with a relatively compact algebraic expression that is to be reduced according to some well-defined procedure to another relatively compact form. Unfortunately, the intermediate forms that are generated during this reduction procedure can be quitre large. If in addition, the intermediate results are rational functions of many variables, then intermediate rational simplification may be very costly, or even impossible in practical terms. For this reason, one might postpone rational simplification until the number of distinct kernels has been reduced. But this usually occurs late in the reduction, when the expression is larger still. Thus the size of the set of kernels has been traded for bulk

of the rational expression, and one difficulty has been replaced by another. This is the unfortunate dilemma that one often faces when attempting to reduce a large porphyritic expression.

As a practical example of such a reduction problem, consider a typical Feynman diagram evaluation, a procedure that requires the application of several operations in sequence to reduce the original expression to its final form. For example, one might have to perform tensor contractions, then carry out traces of Dirac matrix prodects, then perform a Taylor expansion , and finally an integration. The end result might be a function of only a few kinematic invariants, but the intermediate results, viewed as rational expressions, might depend on several tens or even hundreds of kernels. Thus, any attempt to rationally simplify the intermediate results could lead to disaster.

This effect is easily demonstrated in terms of our simple example (2.3). Let:

$$Z(n) = \sum_{k=1}^{n} Y(n-k, n, x, y) \tag{2.7}$$

Consider the problem of reducing the expression:

$$\sum_{k=1}^{10} Y(10-k, 10, x, y) \tag{2.8}$$

The final result is a rational function of only four distinct kernels: x, y, $\log(\frac{x-1}{x})$ and $\log(\frac{y-1}{y})$. On the other hand, the number of distinct kernels present at intermediate stages of the expansion can be much larger. After applying (2.3), the expression contains the kernels x, y, $F(n,y)$ and $F(10,x)$, for values of n from 1 through 10, so that there are 13 kernels in all, or roughly three times as many kernels as occur in the final result. If we try to avoid the difficulty of rationally simplifying such intermediate forms and instead choose to apply (2.2) to the result we have obtained so far, we reduce the number of distinct

kernels to 4, but the resulting expression is very large. In this case, there are 10 occurrences of $F(10, y)$ alone, each with 11 terms. Although the scale of this particular example is well within the reach of modern Computer Algebra systems, one can easily imagine reduction problems that display the same expansion characteristics and present real difficulties to any existing system.

Explicit examples of the ideas described above should provide the reader with additional insight into these difficulties. The fundamental question is the timing of rational simplification. We shall discuss this issue in terms of two possible strategies, which we call Postponement and Interspersion.

As shown above, algebraic expressions can be represented diagramatically as tree structures. In this language, rational transformations can be represented as mappings from one diagram to another, and we can now construct a diagrammatic representation of the two most straightforward reduction streategies. For example, postponement of rational simplification until after all foreground kernels have been expanded is equivalent to scanning the tree for those leaves that represent foreground kernels, and then replacing them by subtrees that represent the expanded forms. Finally the whole structure is rationally simplified. We shall call this approach the Postponement Method. The other method that we shall consider is equivalent to first replacing some of the leaves that represent foreground kernels by partially expanded subtrees, simplifying the entire expression rationally, then alternately repeating replacement and simplification of the entire expression until the desired form is obtained. We shall call this approach the Interspersion Method.

We begin our discussion of these two approaches by comparing their advantages. Each can be useful for specific problems. In particular, Interspersion offers the possibility of avoiding duplication of effort in reduction problems that produce multiple copies of foreground kernels. In the Interspersion Method, it is possible to collect together many terms that involve a

particular foreground kernel, and then to evaluate that kernel once, or perhaps only a few times. By comparison, in the Postponement Method, this evaluation may occur many more times, but of course, the actual amount of duplicated effort depends on the reduction procedure and on the Computer Algebra system itself. For some problems, duplication may not arise at all, in which case the Postponement Method becomes somewhat more attractive. For example, consider the expression $Y(3,5,x,y)+Y(2,5,x,y)$. Applying (2.4):

$$Y(3,5,x,y)+Y(2,5,x,y)$$

$$= \frac{y^3F(5,x)-x^5F(3,y)}{xy} + \frac{y^2F(5,x)-x^5F(2,y)}{xy} \tag{2.9}$$

Proceeding by Postponement, we see that the evaluation of $F(5,x)$ is duplicated, whereas in Interspersion, judicious rational simplification of (2.9) before applying (2.2) can eliminate duplication:

$$= \frac{(y^3+y^2)F(5,x)-x^5(F(3,y)+F(2,y))}{xy} \tag{2.10}$$

Such savings are conveniently obtained by the Interspersion method. Interspersion can also recognize cancellations, as shown below:

$$Y(3,5,x,y)-yY(2,5,x,y)$$

$$= \frac{y^3F(5,x)-x^5F(3,y)}{xy} - \frac{y^2F(5,x)-x^5F(2,y)}{x}$$

$$= \frac{x^4}{y}(yF(2,y)-F(3,y)) \tag{2.11}$$

However, Interspersion cannot recognize all such duplication. Duplications that occur within the same rational expression can be recognized, but there are many examples of duplications

that appear in other ways. In terms of our example (2.4), the expression below illustrates this effect:

$$Y(3,5,x,y)+Y(2,4,x,y)$$

$$=\frac{y^3F(5,x)-x^5F(3,y)}{xy}+\frac{y^2F(4,x)-x^5F(2,y)}{xy} \qquad (2.12)$$

Applying (2.2) to (2.12), we see that $F(4,x)$ again appears in the result, duplicating its appearance in (2.12). Thus unless the reduction is designed so that previously evaluated occurrences of $F(n,x)$ are stored, duplicate evaluations are necessary. Although many Computer Algebra systems do provide facilities for such storage, it is important to note that Interspersion alpne cannt eliminate all duplicated effort in all reduction problems.

We now consider examples of the failure of these two approaches. To see most clearly how the Postponement method can fail, let us expand the right hand side of (2.4). In Figure 3a, we illustrate the Postponement method, which in this case might be the sequential expansion of (2.4) by applying (2.2) throughout the expression repeatedly until all occurrences of F kernels have been eliminated. It is clear that the Postponement method leads to a large result similar to that shown in Figure 3b. Since rational simplification is to some extent a matter of taste, the form shown in Figure 3b is offered only as an example of what might be desirable for certain applications. In this example we see that rational simplification achieves considerable reduction in the complexity of the result. Because the expansion by (2.2) never increases the number of unique kernels in an expression, the Interspersion method does a bit better in this case. Since the complexity of the intermediate expressions is always comparable to the complexity of the final result, Interspersion is effective for this example, and superior to Postponement.

Figures 4a-4g illustrate a problem for which Interspersion is ineffective. We have chosen a problem similar to (2.8), but in the interest of brevity we have set $n = 5$. Referring to Figure 4, we see that although the intermediate expressions are only a bit more complex than the final result, they involve as many as twice the number of distinct kernels found in the final result. In this case, Interspersion requires the rational simplification of 6 intermediate expressions, each one more complex than the final result. Although duplicate evaluations of many F kernels are required, Interspersion itself does not prevent any duplication. The display of the tree structures of the intermediate expressions also clearly demonstrates the magnitude of the intermediate calculations, which casts some doubt on the wisdom of employing the Interspersion method. Moreover, Postponement offers no improvement, since this problem is actually composed of parts similar to the problem illustrated in Figure 3.

now summarize the advantages and disadvantages of Postponement and Interspersion. Postponement avoids rational simplification of large expressions that contain many distinct kernels, but may lead to rational simplification of enormous expressions. Postponement does not allow for the possibility of duplicate foreground kernels unless intermediate evaluations of those kernels are stored. Interspersion can avoid rational simplification of very large expressions, but may require rational simplification of expressions that contain many distinct kernels. In addition, Interspersion may avoid duplicate evaluation of foreground kernels, but unless those kernels are stored, duplicate kernels are recognized only when they are present in the same rational expression. Thus we have shown that the two most straightforward reduction procedures are insufficient.

The method used by LTAB provides a desirable alternative to these two approaches. In this method, rational simplification is performed only on expressions that contain a small number of kernels, and since simplification is performed on intermediate subexpressions, the size of each expression that is subjected to rational simplification is limited. In this way, we

combine the virtues of both methods and reduce the difficulties associated with each. The procedure that is implemented in LTAB accomplishes this by postponing rational simplification, and maintaining tables of intermediate results. To avoid the problem of simplifying expressions that contain many distinct kernels, rational simplification is in fact postponed until expansion is complete, but instead of simplifying the expreession as a whole, the simplification is applied to subtrees that represent the expanded forms of the foreground kernels. These expressions are then combined into progressively more inclusive subtrees and rationally simplified together. To avoid the duplication of effort that results from expanding multiple copies of the same foreground kernel, tables of intermediate results are maintained, so that previously obtained results can be used again whenever possible. Because rational simplification is performed on parts of the intermediate results, we shall call this method Dissection.

More specifically, Dissection begins by extracting all of the foreground kernels from the original expression. These kernels are compared for duplication, and then they are formed into a list. The first step of the reduction procedure is then applied to the elements of this list, which results in a new list, each entry of which is a porphyrytic expression. If a particular entry in this evaluated list is not a porphyritic expression, no further reduction is necessary, but some other entries may require more work. This part of the procedure is now complete for this level of the reduction, but it has resulted in a similar reduction problem, one step further along in the reduction procedure. We continue in this way, creating more lists of foreground kernels, one for each level, and their associated lists of further porphyrytic expressions until finally the reduction of a list of foreground kernels for some level leads to a list of expressions that consists entirely of background. Now we work back through the levels of lists it has generated, rationally simplifying at each level. Specifically, beginning at the penultimate level, the foreground kernels that were evaluated in the last level are replaced by their equivalent background expressions wherever they occur in the porhyrytic expressions of the penultimate level. The

results are then rationally simplified. Next, this procedure is repeated, with the penultimate level now in the role of the last level. We continue in this way until the foreground kernels of the original expression have been replaced by their equivalent background expressions and the entire expression has been rationally simplified.

The advantages of this procedure result from three important features. First, rational simplification is never applied to expressions large expressions that contain more than a few distinct kernels. The problems inherent in the Interspersion Method, namely the rational simplification of expressions that involve many unique kernels, are avoided completely. Second, the extraction of the foreground kernels is done a way that avoids duplications, so some of the advantages of the Interspersion Method are recovered. Finally, the availability of intermediate results makes it possible to reclaim them for all parts of the calculation for which they are valid, even if those separate parts of the reduction have been carried out at different times. Thus, if the intermediate results are saved on disk or tape, it is unnecessary to duplicate any portions of the reduction that may later require those results, which can greatly improve the efficiency of the procedure.

One shortcoming of this method is that cancellations cannot be recognized automatically, because the intermediate rational expressions that involve the intermediate foreground kernels are never assembled. So if one expects numerous such cancellations, the Dissection method is less convenient than a segmented reduction interspersed with rational simplification, although in LTAB one can always reassemble the expression at any point where such cancellations are expected. Unless one expects frequent cancellations or small numbers of foreground kernels during the intermediate rational simplification, either Postponement or Interspersion will probably be considerably less efficient than Dissection.

The method implemented in LTAB is therefore best suited to problems that involve complicated reduction of large porphyritic intermediate expressions with porphyritic intermediate

expressions involving many intermediate foreground kernels that cannot be expected to cancel at intermediate stages.

To demonstrate the behavior of this alternative reduction procedure, consider the example used previously to examine the efficiency of Interspersion:

$$\sum_{k=1}^{5} Y(5-k,5,x,y) \tag{2.13}$$

LTAB is flexible. This flexibility can be exploited to implement the particular reduction procedure that is best suited to this problem. In this case one might rdeuce this expression as follows. It is usually wise to reorganize the expression while it is still compact, so that multiple copies of foreground kernels can be eliminated. Thus we obtain

$$\frac{F(5,x)(y^4+y^3+y^2+y+1)-(F(4,y)+F(3,y)+F(2,y)+F(1,y)+F(0,y))x^5}{xy}. \tag{2.14}$$

Next we extract from (2.14) all F kernels and enter them in separate lists. For this problem, it is probably best to organize them according to their first argument, so we obtain in this way six lists, as shown in Figure 5a. We have numbered these lists according to their level in this hierarchy. This organization of the problem is suggested by the recursion relation (2.2), which, in the course of expanding foreground kernels of a given level, generates foreground kernels that belong to the levels beneath it. The Dissection method as currently implemented in LTAB cannot deduce the reduction strategy that is best for a given problem. The strategy to be employed must be supplied by the user.

We now apply (2.2) to the highest level in the hierarchy, Level 1, to obtain list of partially evaluated forms, as shown in Figure 5b. In this case, there is only one entry in the list of foreground kernels of Level 1, but in general there may be any number. These forms contain new foreground kernels, whose values are as yet unknown. They are then entered into the lists

lower down in the hierarchy, for later evaluation. The result of this step is shown in Figure 5c. The first step of the reduction procedure is now complete.

For the next step of this procedure, we direct our effort at the second list, labeled Level 2, and procede as above. The result is shown in Figure 5d. Repeating this process for each level, we finally reach the state depicted in Figure 5e, in which all the foreground kernels that have been assigned to Level 6 have been expressed in terms of the background variables.

Next we substitute these values from Level 6 into the expressions entered in Level 5, as shown in Figure 5f. This upward substitution and simplification is then repeated for each level until we reach the state shown in Figure 5g. Finally all these results are inserted into the original expression, and the entire result is rationally simplified to obtain:

$$((60x^5(\log(\frac{y-1}{y})-\log(\frac{x-1}{x}))$$

$$-(60x^4+30x^3+20x^2+15x+12))(y^4+y^3+y^2+y+1)$$

$$+x^5(60y^3+90y^2+110y+125))/60xy \qquad (2.15)$$

The effect of this procedure is exactly equivalent to applying (2.1b) to (2.13). Consequently, one might wonder whether anything has been gained by such efforts. For example, in light of the existence of (2.1b), one might object that the above rather intricate procedure is wasteful. However, in more practical problems than this one, closed form results analogous to (2.1b) do not necessarily exist. Furthermore, the steps of the reduction procedure of a practical problem are generally far more complex than those illustrated here, which unfortunately leads to the difficulties discussed above. Briefly, the reply to such objections is that the purpose of this illustration has been to compare the Dissection method to other methods that might be employed if (2.1b) were unavailable.

What then has been achieved by this method? First, this segmented reduction procedure has provided a derivation of (2.1b) for those kernels that appeared in our problem. In practical problems for which closed forms are not known in advance, Dissection provides a method of obtaining these necessary forms. Second, this result has been achieved by a method that involved manipulating only those particular intermediate expressions that were directly related to the goal of deriving the rquired closed forms. By ignoring the original expression (2.13) throughout most of the procedure, expensive rational simplification of largely irrelevant structures was completely avoided. This principle was actually applied at all 6 levels of the segmented reduction procedure. Finally storage of intermediate results allowed us to avoid wasteful recalculation of the forms $F(n,y)$ for $n=0$, 1, 2, and 3, as would have been required in either the Postponement or Interspersion methods described above.

These ideas are illustrated in Figure 6. The tree structure of (2.14), which is the starting point of the reduction, is shown in Figure 6a. Since the entries in the lists of Figure 5 are all relatively small, the tree structures of the intermediate structures that are stored there are not shown. The largest structure, which is obtained when the results shown in Figure 5g are substituted into (2.14), is illustrated in Figure 6b. Since (2.14) was simplified by Interspersion as shown in Figure 4, the result of simplifying the expression of Figure 6b is shown in Figure 4g.

Although the expression of Figure 6b is a bit larger than the expressions obtained when the Interspersion method is applied to this problem, note that it contains only 4 distinct kernels. By comparison, the intermediate forms that were simplified in the Interspersion treatment of this problem contained as many as eight distinct kernels. The expression of Figure 6b is also much smaller than the intermediate expression that would have been simplified if Postponement had been applied to this problem. Moreover, in the Dissection method, only one such large expression was rationally simplified, whereas in the Interspersion method, five such intermediate forms were simplified. This shows how Dissection can reduce the number of kernels

functions inadvisable. A BATCH mode approach is preferable for several reasons. First, the batch file provides an accurate record of the specific operations that were performed. This record is very useful for locating and understanding errors. Second, if an error is discovered, then it is often possible to recover from that error by simply running some of the batch files again, after making minor corrections. Finally, several attempts may be necessary before a particular step in the reduction can be achieved to satisfaction, and the availability of the batch file eliminates duplication ofd the simpler portions of that effort.

Given this *caveat*, this section is divided into several parts. We begin with a general description (3.1) of the preparations that are necessary before one can begin to reduce an algebraic expression by means of the LTAB programs. Next we describe in detail (3.2) the structure of the data tables that are used by these programs. Finally, we describe (3.3) the functions that one actually uses to carry out the reduction.

3.1. Preparation for Segmented Reduction

To use LTAB in MACSYMA, one must first decide where to segment the reduction procedure. With experience, one will probably develop a feeling for choosing the segmentation points, but we offer the following suggestions. The choice of segmentation depends somewhat upon the computer facilities that are available. For example, if the reduction is expected to procede quickly, one might divide the reduction into only a few parts. On the other hand, a shortage of workspace may necessitate a more segmented approach. Often it is impossible to determine in advance the precise division points for a segmented reduction, and in the end, experimentation usually provides this information. The choice of segmentation is also directly influenced by the nature of the reduction problem. For example, the reduction may lead through some point where many identical foreground kernels are produced. Although it may not always be possible to anticipate that this will happen, it is usually advantageous to segment

contained in expressions that must be rationally simplified, while at the same time reducing the amount of effort expended during the intermediate rational transformation. In this way, Dissection provides a useful compromise between Postponement and Interspersion.

3. Using LTAB in MACSYMA

We now turn to a detailed discussion of a computer program that makes use of the ideas preesented in Section 2. This discussion has a twofold purpose. First, it can serve as a guide to MACSYMA users who may wish to undertake a very large reduction problem, and second, it may suggest a useful reduction scheme to users of other Computer Algebra programs.

The reader who is unfamiliar with MACSYMA will probably be most interested in determining just what is required to achieve the efficiency improvements that are possible with segmented reduction. With this goal in mind, we have tried to organize this section to clearly answer two important questions.

[1] What facilities does LTAB provide for implementing a segmented reduction procedure?

[2] What steps of the reduction procedure must be provided by the user?

To discuss these questions, it is not necessary to understand the precise technical meaning of all of the terminology used below. However, readers who have little or no experience with Computer Algebra systems or who may find portions of this section somewhat obscure may wish to consult the MACSYMA Reference Manual[1].

We begin with a recommendation of caution. Although much of MACSYMA is oriented toward interactive use, calculations to which LTAB might be applied are best carried out in BATCH mode. Some advance planning is desirable, and may even be necessary as a glance at the following LTAB function descriptions may indicate. The complexity of this evaluation strategy, combined with the complexity of a large calculation, makes interactive use of these

the reduction procedure may fail to take account of certain special characteristics of the problem at hand. This failure can be expressed as a conceptual error, but more likely it appears as clumsiness or inefficiency in the reduction procedure. Finally, there is always the possibility of programming errors either in the user's programs or in system programs. For these reasons one almost inevitably needs to retrace some step or sequence of steps in reduction procedure. In such cases it is necessary to have a thorough understanding of the internal structure of LTAB and its data tables, so that recovery from errors can be as painless as possible. Therefor, detailed descriptions of the structure and content of the LTAB data tables ar provided below. The description of each component includes suggestions for its use whenever appropriate.

For the purpose of these descriptions, we think of a reduction procedure as if it consisted of a simple vertical chain, with the original porphyritic expression at the top and the final LTAB table at the bottom. Thus *down*, *below*, and *subsequent* refer to the direction of further reduction, while *up*, *above* and *previous* refer to the other direction. For convenience, we shall describe the contents of one such table named A. The table just above A is called PARENT, and A's successor is called CHILD.

In actual use however, there is no such restriction on the structure of the interrelations between the various LTAB tables. It is possible for any table to have several parents or children, and there are many instances when such structures are desirable. As an example of multiple children, suppose that at some stage in the reduction procedure two different types of foreground kernels are generated, and that further reduction of these kernels requires distinctly different methods. In such a case one might wish to process the two types of foreground kernels in separate tables, which would require that table A have two children. Multiple PARENT tables are also useful. Suppose that a given problem has been divided into two or more parts according to the validity of a certain approximation procedure or other special method particular to the problem. Algebraic expressions are then set up in each domain and must be reduced

the reduction at such a point to avoid duplication of that part of the reduction procedure. Of course, the efficiency gains that result from the segmented depend upon the cost of reducing each kernel, but in general it is advantageous to segment a reduction procedure at those points where one expects multiple copies of identical foreground kernels to be produced.

Once one has determined a segmentation point, the next required preparatory step is to write a function that performs the step of the reduction procedure that carries the reduction from that segmentation point to the next one. In the discussion below, this function is called the PROCESSOR function. This function is specific to the problem at hand, so it cannot be provided by LTAB.

In the final preparatory step, one creates a blank table for each step in the segmented reduction. As the evaluation procedes, LTAB stores all necessary information in these tables. Although the structure of these tables is uniform independent of their level in the procedure, it is not possible for LTAB to generate these tables automatically, since the content of the tables is dependent on the procedure itself. The function LTAB_INITIALIZE has been provided for creating such blank tables.

3.2. Description of the Data Tables

In principle, preparation for a segmentation using LTAB consists of three steps: choosing the segmentation points, writing PROCESSOR functions for the corresponding reduction steps, and creating the requisite blank tables. In practice, however, things are more complicated for three reasons. First, choosing the segmentation points can be a difficult task. In many instances, the optimum choice of segmentation is not at all evident at the outset, and in the extreme case, it may not be possible to determine the next segmentation point until one has proceded part way into the reduction effort. Second, even the most careful advance planning of

A[_PROCESSOR] This array element holds the name of the function that is used during the A step of the reduction procedure to process the foreground kernels that are being held in A[_VALLIST]. These kernels are the ones that were discovered during the step in the evaluation procedure that is just above A. If one wishes to change the name of the PROCESSOR function after the A table has been created, one merely sets A[_PROCESSOR] to that name. The processor function itself must be a function of one argument.

A[_NEXLIST] This array element refers to a list of expanded forms corresponding to the quantities held in A[_VALLIST]. The elements in this list are the results of applying the function A[_PROCESSOR] to the elements of A[_VALLIST]. The form of the elements in the list held in A[_NEXLIST] is as follows. Each entry consists of background kernels supporting the foreground kernels generated in the A step of the evaluation procedure. However, the newly generated foreground kernels have been replaced by new labels, using the alphabetic label prefix provided by CHILD, and stored in CHILD[_PREFIX]. If A is the last step in the procedure, then there are no such labels and the elements consists purely of background kernels.

This component of the A table is very useful as a safety mechanism. If one discovers an error in the reduction procedure somewhere below A, and if one has already back-substituted into the A table or perhaps even above it, one need correct only the results beginning at the point of the error, but not above it, if one uses the _NEXLIST of the tables above the error. Since back-substitution has been carried out above the error, one might think at first that all downward calculations would also be lost, but they can be recovered from the _NEXLIST, since it holds the forms calculated during the downward phase of the reduction. One simply executes A[_FULL_EV_LIST]:A[_NEXLIST] or equivalently, LTAB_FULL_EV_ERASE(A) to restore the A table to the state it was in prior to the back substitution of the incorrect forms. If only certain portions of A[_FULL_EV_LIST] have been affected, a correct structure can always be composed of pieces of A[_FULL_EV_LIST] and A[_NEXLIST], using the MACSYMA functions PART and SUBSTPART. In this way, any incorrect results can be carefully removed, without the necessity of duplicating results that are known to be correct.

A[_FULL_EV_LIST] This array element refers to a list of expanded forms corresponding to the quantities held in A[_VALLIST]. The form of the elements in this list, unlike the list held in A[_NEXLIST], can vary. Immediately after the processing of the elements in A[_VALLIST] with the function A[_PROCESSOR], the elements are in the form of the elements in A[_NEXLIST]. However, there are several circumstances that can result in changes in A[_FULL_EV_LIST]. Suppose for example, that one has evaluated all the tables beneath A, and then back-evaluated up to A, or possibly above A. The elements in A[_FULL_EV_LIST] then consist entirely of background. That is, the elements are "fully evaluated" (hence the name _FULL_EV_LIST). In this situation, they differ from the elements in A[_NEXLIST], which are always in the form of structures composed of background kernels and labels from the table beneath A. Yet another possible form of the elements in A[_FULL_EV_LIST] can result if one subsequently adds more elements to

according to distinct procedures appropriate to each domain. However, suppose further that as the reduction procedes, it becomes possible to merge the domains and process the expressions according to a single procedure. This could happen if, for example, one has proceded past the point at which the special technique was applied. One might then save much effort by designating a particular table as CHILD of as many parents as possible, because of the sharing of effort that would then become automatic. Another example of a multiple parent structure is given in Section 4.

Each LTAB table is an array with 11 elements. Four of these elements refer to lists of raw or processed data: _VALLIST, _LABLIST, _NEXLIST, and _FULL_EV_LIST. Of the remaining seven elements, four are used by various LTAB functions when these data are being processed or stored. These elements are _PROCESSOR, _FULL_EV_FCN, _NEXT_TABLE and _MISCLIST. The remaining elements are used by the PARENT table when it is adding data to A. These elements are _COUNTR, _PREFIX and _OPERATORS. The formation and applications of each of these quantities are described below.

A[_VALLIST] This array element refers to a list of foreground kernels that were generated in the PARENT step of the evaluation procedure. During the PARENT step, whenever LTAB encountered one of these kernels, A[_VALLIST] was checked to see if it had already been entered there. If not, then an entry was made and a label generated for that kernel. All of these kernels were then replaced by the corresponding labels that were being held in A[_LABLIST]. These kernels are the input for the A step in the reduction procedure. After executing the PARENT step of the reduction procedure, one may examine the foreground kernels extraced in that step by asking for the value of A[_VALLIST]. One may also test the PROCESSOR of A by applying it to one of the elements of this list.

A[_LABLIST] This array element refers to a list of atomic symbols that were generated during the PARENT step in the reduction procedure. These are the symbols that were used in place of the foreground kernels that were discovered in that PARENT step, and which are being held in A[_VALLIST]. They were generated as labels for their corresponding foreground kernels by concatenating A[_PREFIX] with successive values of A[_COUNTER].

PARENT, it is essential that the table A exist, at least in blank form, prior to the execution of the PARENT step of the reduction procedure.

A[_PREFIX] This array element holds the alphabetic string that PARENT uses when it generates labels for the foreground kernels of the A step. Bote that since this list is used by PARENT, it is essential that the table A exist, at least in blank form, prior to the execution of the PARENT satep of the reduction procedure.

3.3. Functions for the Reduction Process

Once the preparations are complete, one executes the reduction procedure by means of the functions provided. For example, to add the first foreground kernels to the first table in the reduction procedure, one uses the function LTAB_LABEL_UPDATE. To execute one step of the reduction procedure, one uses the function LTAB_VALUE_UPDATE. To substitute the final background kernels from one level into the table just above it, one uses the function BACKEV. To substitute the final values of foreground kernels back into the original expression, one uses the function LTAB_FINAL_SUBST. To store one or more LTAB tables in a disk file, one uses the function LTAB_SAVE. These and other useful functions are described below.

3.3.1. Creating a Table

A necessary first step for carrying out a segmented evaluation using LTAB is the creation of the desired evaluation tables. The following function is provided for this purpose.

LTAB_INITIALIZE(*name*, *processor*, *full_ev_fcn*, *next_table*, *string*, *ops*, *optional-args*) creates a table named *name* for a segmented evaluation. *processor* is the name of the function that is used to process the elements in the _VALLIST of the table *name*. *full_ev_fcn* is the name of the function that is applied by BACKEV to the elements of the _FULL_EV_LIST after substitutions have been made from lower level results. *next_table* is the name of table that is immediately below NAME in the segmented evaluation. If NEXT_TABLE is the atom END, then LTAB deduces that this table is the last of a chain.

A[_VALLIST] using the function LTAB_LABEL_UPDATE. This often happens, especially in a large calculation that has been segmented. The resulting form of the elements of A[_FULL_EV_LIST] is then mixed: some elements are full evaluated, and some are identical to their corresponding elements in A[_NEXLIST], awaiting further processing. Finally, the elements in A[_FULL_EV_LIST] can be pointers to other elements in that same list. This form results only when the function LTAB_OPT has been used to reduce the size of the A table.

A[_MISCLIST] This array element holds a list of the names of miscellaneous items that are to be saved along with A when the function LTAB_SAVE is used. For example, one might wish to store a predefined quantity for use with A's PROCESSOR function, or perhaps the names of auxiliary user-defined functions that PROCESSOR calls to carry out its task. Including the names of these quantities or functions in A[_MISCLIST] forces the function LTAB_SAVE to save these items whenever it save A. Elements can be added to this list at any time by the user, but they must evaluate to valid arguments to SAVE or FASSAVE.

A[_COUNTER] This array element holds an integer that is the number of the highest label generated so far by PARENT. Thus it is the number of foreground kernels stored in A.

A[_NEXT_TABLE] This array element holds the name of the table CHILD. If this array element holds a list, then the list is interpreted as a list of the CHILDREN of A. This feature allows multiple branching downward. If A has no children, A[_NEXT_TABLE] has the value END.

A[_FULL_EV_FCN] This array element holds the name of the function that is used by BACKEV when values from CHILD are inserted for CHILD's foreground kernels in A[_FULL_EV_LIST]. After BACKEV performs the substitutions of the values obtained from CHILD, A[_FULL_EV_FCN] is applied to the resulting expression. Examples of functions useful for this purpose are RATSIMP, FACTOR and SQFR, but the user may also provide the name of a more specialized function if desired. The only restriction is that the function must accept a single argument.

A[_OPERATORS] This array element holds a list of the leading operators of foreground kernels that are stored in A[_VALLIST], and is used by PARENT when it is searching for the foreground kernels to be inserted in A[_VALLIST]. Normally this is a list of one element, but it may take several other possible forms. For rg, there may be several different operators that can be the leading operators of foreground kernels stored in A. In this case, the list A[_OPERATORS] may contain several elements, including one for each operator. Also, one or more of the elements of the list A[_OPERATORS] can be of the form PREDICATE($predicate-name_1$, $predicate-name_2$, $\cdots$). In this case any kernel whose leading operator satisfies any of the predicates named in the argument list of the pseudo-function PREDICATE are also stored in A. Note that since this list is used by

then FASSAVE is used, otherwise SAVE is used. It is reccommended that periodic storage be employed whenever a long computation is anticipated. In this way, recovery from error is required only for the calculations done since the last writing.

LTAB_LABEL_UPDATE(*name*,*processor*,exp) is used to add new foreground kernels to the table *name* using the function *processor* on the expression exp. LTAB_LABEL_UPDATE is most useful for generating or adding to the first table in a chain of tables, or the root table in a tree of tables. The LINECHAR that is used for isolation of any new foreground kernels that might arise in the course of this evaluation is held in *name*[_PREFIX], and can be any string of alphabetic characters.

One switch controls the behavior of both LTAB_LABEL_UPDATE and LTAB_VALUE_UPDATE when they encounter foreground kernels that have already been fully or partially processed in a previous evaluation.

LABEL_UPDATE_FULL_EV determines precisely what quantities are substituted for the foreground kernels encountered by LTAB_LABEL_UPDATE and LTAB_VALUE_UPDATE. If a kernel has never been encountered in a previous evaluation of LTAB_LABEL_UPDATE or LTAB_VALUE_UPDATE then of course, there is no choice but to return a newly generated label. But if one is adding to an old table, and that table has had some further processing before the new additions are made, then its _FULL_EV_LIST may contain some fully- or partially-processed values of foreground kernels already in the table. Normally, one would prefer that these values be inserted whenever these kernels are reencountered. This is indeed the behavior when LABEL_UPDATE_FULL_EV is TRUE, the default. Setting this switch to FALSE forces substitution of the labels themselves, which may occasionally be preferable, especially when one wishes to make comparisons to expressions calculated earlier. Finally, setting this switch to ZERO_ONLY produces behavior similar to the FALSE setting except that if any of the fully evaluated kernels are known from previous calculation to be 0, that 0 is inserted instead of a label.

LTAB_LABEL_FIND(*name*) returns a list of the newly added and unevaluated foreground kernels that are held in *name*[_LABLIST]. This includes only those kernels that have not been processed in any way, and excludes those elements that have been expanded in terms of the next forground kernels of higher levels. It is unlikely that the user would ever require access to this function.

string is the string of alphabetic characters that is to be used for generating labels to stand for the foreground kernels discovered either by the PROCESSOR of the table above *name* or by means of the function LTAB_VALUE_UPDATE. *Ops* is a list of the possible leading operators that the foreground kernels stored in *name*[_VALLIST] may have. It is also possible for *ops* to include elements of the form PREDICATE(< $predicate\text{-}name_1$, $predicate\text{-}name_2$, ··· >). If such elements are included, then any kernel that satisfies any of the predicates is also added to *name*[_VALLIST] when encountered by the *processor* of the table above *name*. *Optional-args* refers to any number of additional optional arguments that are the names of any objects that one might wish to save along with *name* when the function LTAB_SAVE is used, or when the switch PERIODIC_SAVE_FILENAME is not FALSE. For example, one might wish to have the *processor* function or the *full_ev_fcn* function stored in the same file as the table. If so, one would include the names of these functions among the arguments of LTAB_INITIALIZE, in the position indicated by *optional-args*.

3.3.2. Downward Evaluation

There are two different types of downward evaluation. In the beginning of an evaluation, one wishes to make entries in a table by processing an expression that is not itself a table. For this case one uses the function LTAB_LABEL_UPDATE. To make entries in a table A from a parent table of A, one uses the function LTAB_VALUE_UPDATE.

LTAB_VALUE_UPDATE(*name*, *n*) updates the values of any recent additions to the table named *name*. If no new additions are found, no changes are made in the table. If new additions are found, then the function whose name is held in *name*[_PROCESSOR] is used to process these new elements. The new foreground expressions that are generated in the course of this evaluation are automatically added to the table whose name is held in *name*[_NEXT_TABLE]. They appear in the table *name* only in an ISOLATEd form in *name*[_FULL_EV_LIST]. That is, only labels that stand for these newly generated foreground kernels appear in *name*[_FULL_EV_LIST]. The LINECHAR that is used for the generation of these labels is held in *name*[_NEXT_LABEL].

The second argument, *n*, is optional. If it is given, it must be a positive integer. When such an argument is given, it represents the maximum number of elements that are evaluated before the partially updated table *name* is written out into a disk file. Specification of a file for this purpose is given by setting PERIODIC_SAVE_FILENAME to the name of the desired file. If PERIODIC_SAVE_FILENAME is FALSE, then no such intermediate storage occurs. The switch DGVALFASSAVE[TRUE] determines whether SAVE or FASSAVE is used for this purpose. If DGVALFASSAVE is TRUE

LTAB_LABEL_CLEAR(*exp*, *name*) examines the array *name* to determine what labels are stored in it as the result of any previous segmented evaluations. Then it substitutes the values that these labels stand for into the expression exp. This operation is the inverse of LTAB_LABEL_UPDATE, at least as far as exp is concerned. No alterations are made in *name*.

LTAB_FINAL_SUBST(*exp*, *name*) examines the array *name* to determine what labels are stored in it as the result of any previous segmented evaluations. Then it substitutes the entries of the _FULL_EV_LIST that correspond to these labels into the expression exp. No alterations are made in *name*.

3.3.5. Storing and Retrieving the Tables

LTAB tables can be stored with any of the standard storage functions, but if one has several things to store along with the array itself, then it may be more convenient to use the function LTAB_SAVE. For example, one might wish to store a few global variables with the table, or perhaps the _PROCESSOR and _FULL_EV_FCN definitions. If so, then inclusion of the names of these objects in the _MISCLIST entry of the table permits one to exploit the convenience of LTAB_SAVE.

LTAB_SAVE(*filename*, arg_1, arg_2, $\cdots$) saves the quantities named arg_1, arg_2, $\cdots$ in the file named filename. The form of the arguments of LTAB_SAVE is identical to the form of the arguments of SAVE or FASSAVE, except that LTAB_SAVE evaluates its arguments. Thus if any of the arguments has a value, it is necessary to present it to LTAB_SAVE in a "single-quoted" form. Failure to do so leads to an error when LTAB_SAVE passes such arguments to SAVE or FASSAVE. The use of SAVE or FASSAVE by LTAB_SAVE is determined by the value of the switch LTABFASSAVE[TRUE]. If LTABFASSAVE is TRUE then FASSAVE is used; if FALSE, SAVE is used. If any of the arg_i are hashed arrays, and the value of the array indexed by _MISCLIST is a list, then the elements of that list is also saved.

Retrieval of LTAB tables that have been stored on disk can be accomplished with the standard MACSYMA forms LOAD or LOADFILE.

3.3.3. Correcting Errors

As noted earlier, it is frequently necessary to alter the contents of an evaluation table to correct errors. Two functions are provided for this purpose.

LTAB_ERASE(*name*) erases all labels and values that are already entered in the array *name*. This function is useful for purging the array of previously calculated erroneous results without changing other parameters or functions that might be associated with the array. It is also useful for obtaining a copy of the initialized array from one calculation for use in another.

LTAB_FULL_EV_ERASE(*arrayname*) is similar to LTAB_ERASE, except that the labels already stored in the array are not removed. _NEXLIST and _FULL_EV_LIST are, however, reinitialized. This function is useful for purging the array of previously calculated erroneous results while retaining the foreground kernels that are stored in the array.

3.3.4. Upward Evaluation

There are two possible kinds of upward evaluation. The first type involves substitution of the fully-evaluated foreground kernels of one level into the tabel one level immediately above it. This is done by means of the function BACKEV. The second type involves substitution into some expression that is not a table. This capability is needed for those tables that do not have parents. If the substituted quantities are to be foreground kernels then LTAB_LABEL_CLEAR is used. If the evaluated, reduced forms of these kernels are required, then LTAB_FINAL_SUBST is used.

BACKEV(*higher_level*, *lower_level*) BACKEV lifts the results of a table lower in the chain up to a table higher in the chain. This is accomplished by modifying the entries in *higher_level*[_FULL_EV_LIST] to reflect the values contained in *lower_level*[_FULL_EV_LIST]. The function whose name is held in *higher_level*[_FULL_EV_FCN] is applied to the entries in *higher_level*[_FULL_EV_LIST] after the substitutions are made.

we try a segmented reduction.

We begin by choosing as segmentation points the points determined by the values of n. Thus we attempt to build tables of foreground kernels of the form:

$$f(n, x, p(x), s, \omega),$$

one table for each of the three possible values of n.

We are now prepared to write a program for performing this reduction by means of LTAB. First, we need functions that can detect f kernels, which appear in the expression Y as nouns. So:

```
F3P(EXP) := IS(PART(EXP,0) = NOUNIFY('F) AND PART(EXP,1) = 3)$
F2P(EXP) := IS(PART(EXP,0) = NOUNIFY('F) AND PART(EXP,1) = 2)$
F1P(EXP) := IS(PART(EXP,0) = NOUNIFY('F) AND PART(EXP,1) = 1)$
```

Each of these predicate functions returns TRUE for a given EXP if the value of n is as required and the leading operator of EXP is f. These functions will be used by LTAB to locate the kernels that must be expanded. Since we have decided to break the expansion at points that correspond to the different values of n, we now construct a PROCESSOR for each of the three levels. For this example, the PROCESSOR for $n = 3$ can also serve for $n = 2$:

4. An Example

It is difficult to construct a specific example of the application of this method that is general enough to be widely understood without special knowledge. Therefore we offer a generalized example in the hope that the important principles are clearly illustrated without introducting the obscurity that may result from a highly specialized example. Consider the following problem. We are given a rational expression Y that depends on variables ω, x, and s and on kernels of the form

$$f(n, x, p(x), s, \omega)$$

where p is a rational function of its argument. The precise form of p is not fixed, and may be different for each occurrence of f. The possible values of n are 1, 2, and 3. the function f is defined as:

$$f(n, x, p(x), s, \omega) = g(x, p(x), s) f(n-1, x, \frac{dp(x)}{dx}, s, \omega)$$

$$+ h(x, p(x), \log s, \log(1-\omega)) \qquad (n \neq 1)$$

and

$$f(n, x, p(x), s, \omega) = j(\frac{d\,p(x)}{dx}, \log s, \log(1-\omega), \omega) \qquad (n = 1)$$

where g, h, and j are known rational functions of their arguments. For all n, the expansions of f are cumbersome. We require a power series expansion for Y about $\omega = 0$. To this end, we have already applied the MACSYMA function TAYLOR (which produces a Taylor series expansion of its argument) to the result of expanding all of the f kernels in Y, and have found that the expanded form is much too large for our computer to work with efficiently. Therefore

PHOTOGRAPH THIS SHEET

LEVEL: II

INVENTORY: 1

DTIC ACCESSION NUMBER: AD A146847

Proceedings of the 1984 MACSYMA User's Conference

DOCUMENT IDENTIFICATION 23-25 July '84

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DISTRIBUTION STATEMENT

ACCESSION FOR
NTIS GRA&I [X]
DTIC TAB [ ]
UNANNOUNCED [ ]
JUSTIFICATION

BY
DISTRIBUTION /
AVAILABILITY CODES

DIST	AVAIL AND/OR SPECIAL	
A/1		

DISTRIBUTION STAMP

DTIC
ELECTE
S OCT 22 1984 D
D

DATE ACCESSIONED

DATE RETURNED

84 09 27 006

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NO.

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-DDAC

DTIC FORM 70A

DOCUMENT PROCESSING SHEET

PREVIOUS EDITION MAY BE USED UNTIL STOCK IS EXHAUSTED.

```
PROCESSOR32(EXP) :=
 SUBST(LAMBDA([NVALUE,XVAR,RATFUN_X,X,OMEGA],
         'F(NVALUE-1,XVAR,DIFF(RATFUN_X,XVAR),X,OMEGA)
           *G(XVAR,RATFUN_X,S)
         + H(XVAR,RATFUN_X,LOG(S),LOG(1-OMEGA))
     NOUNIFY('F),EXP)$


PROCESSOR1(EXP) :=
 RATSIMP(SUBST(LAMBDA([NVALUE,XVAR,RATFUN_X,X,OMEGA],
                J(DIFF(RATFUN_X,XVAR),LOG(S),
                  LOG(1-OMEGA),OMEGA)),
           NOUNIFY('F),EXP),
      LOG(1-OMEGA),OMEGA)$
```

The FULL_EV_FCN for each table should organize the results around terms involving ω, since we require a series expansion in ω. This is desirable for all levels, so we shall use the function RATSIMP_LOG_OMEGA, defined below, as the FULL_EV_FCN of every level. The FULL_EV_FCN of the lowest table is ignored, so this restructuring with respect to ω is done in PROCESSOR1. For LEVEL2 and LEVEL3 we need:

```
RATSIMP_LOG_OMEGA(EXP)  :=  RATSIMP(EXP,LOG(1-OMEGA),'OMEGA)$
```

We are now prepared to create the needed blank tables. We choose the names of the tables to correspond to the values of n, and choose the alphabetic strings I, II and III as the prefixes for generating the labels for the kernels of tables LEVEL1, LEVEL2 and LEVEL3, respectively. Thus LEVEL1 is the lowest, LEVEL3 the highest of the tables in this reduction. We have included all functions associated with a particular table in the _MISCLIST of that table so that they will be saved with the table itself when we use LTAB_SAVE.

```
LTAB_INITIALIZE('LEVEL3,'PROCESSOR32,'RATSIMP_LOG_OMEGA,'LEVEL2,'III,
        [PREDICATE('F3P)],
        ['F3P,'PROCESSOR32,'RATSIMP_LOG_OMEGA])$


LTAB_INITIALIZE('LEVEL2,'PROCESSOR32,'RATSIMP_LOG_OMEGA,'LEVEL1,'II,
        [PREDICATE('F2P)],
        ['F2P,'PROCESSOR32,'RATSIMP_LOG_OMEGA])$


LTAB_INITIALIZE('LEVEL1,'PROCESSOR1,'RATSIMP_LOG_OMEGA,'END,'I,
        [PREDICATE('F1P)],
        ['F1P,'PROCESSOR1,'RATSIMP_LOG_OMEGA])$
```

We begin processing the expression Y by extracting the f kernels from Y and inserting them in their designated tables.

```
Y_EXTRACT_3:LTAB_LABEL_UPDATE('LEVEL3,'PROCESSOR32,Y)$
```

Y_EXTRACT_23:LTAB_LABEL_UPDATE('LEVEL2,'PROCESSOR32,Y_EXTRACT_3)$
Y_EXTRACT_123:LTAB_LABEL_UPDATE('LEVEL1,'PROCESSOR32,Y_EXTRACT_23)$

In the above sequence, the quantity Y_EXTRACT_3 has only the kernels of Level 3 extracted, Y_EXTRACT_23 has kernels of of both Level 2 and Level 3 extracted, and Y_EXTRACT_123 has all f's extracted. Y_EXTRACT_123 is the quantity into which we will substitute the final values of the expanded kernels. Y_EXTRACT_3 and Y_EXTRACT_23 are no longer needed. Now thathave made the initial entries of the foreground kernels into the tables LEVEL1, LEVEL2 and LEVEL3, we procede to evaluate the kernels stored in those tables. This is done easily:

LTAB_VALUE_UPDATE(LEVEL3)$
LTAB_VALUE_UPDATE(LEVEL2)$
LTAB_VALUE_UPDATE(LEVEL1)$

We have now reached the lowest level, and all foreground kernels in LEVEL1 are reduced to background. The next step is to substitute these results into the tables above:

BACKEV(LEVEL2,LEVEL1)$
BACKEV(LEVEL3,LEVEL2)$

All tables now contain expanded forms of al kernels, each one reduced to background and restructured so that the quantities involving ω are in leading positions. The next step is to

substitute these results into Y. Since the original expression may have contained kernels from all three levels, iwe must make substitutions from all three tables.

```
Y_FINAL_3:LTAB_FINAL_SUBST(Y_EXTRACT_123,LEVEL3)$
Y_FINAL_23:LTAB_FINAL_SUBST(Y_FINAL_3,LEVEL2)$
Y_FINAL_123:LTAB_FINAL_SUBST(Y_FINAL_23,LEVEL1)$
```

Again, Y_FINAL_3 and Y_FINAL_23 are intermediate forms that are no longer useful. The result we have sought is Y_FINAL_123. All that remains is to apply TAYLOR to this form. The efficiency of this part of the procedure is improved if we first ISOLATE with respect to ω. In this case ISOLATE simply replaces expressions that do not contain ω with newly-generated atomic quantities.

```
ISOLATE_WRT_TIMES:TRUE$
Y_FINAL_ISOLATED:ISOLATE(Y_FINAL_123,'OMEGA)$
Y_FINAL_ISOLATED_TAYLOR:TAYLOR(Y_FINAL_ISOLATED,'OMEGA,0,3)$
Y_FINAL:EV(Y_FINAL_ISOLATED_TAYLOR)$
```

The final call to EV removes the isolation variables inserted by ISOLATE. The result we seek is Y_FINAL.

This problem has a feature that deserves special emphasis. The original expression Y contains kernels of each of the three types. Therefore it is possible that some of the kernels that are found in the original expression are also generated as a result of expanding higher

order kernels that are themselves found in the original expression. A straightforward expansion of all kernels can therefore result in duplication of effort. In the approach that we have taken here, each unique kernel is evaluated only once.

5. Outlook

In its current implementation, LTAB provides a systematic method for applying intricate reduction procedures to large expressions. However, it is possible to construct programs which automate this procedure even further. Currently, the user must generate blank tables before the reduction process can begin, and provide names for the prefixes used in those tables, but one can define functions or macros that will carry out these steps automatically if necessary. These evaluation functions would need as arguments both the downward and upward processor functions for each level, as well as the expression to be reduced. Alternatively, one might construct a macro equivalent to a function definition operator, except that instead of producing a simple function definition, it produces a function definition that uses Dissection to procede from one step to the next within the body of the definition. The user might be required to specify the downward processors for each level, as well as their corresponding upward processors, all as statements in the "function definition". The only evidence that a Dissection-oriented reduction scheme was actually being used would be this tripling of statements in the definition. Such a scheme could greatly improve performance of Computer Algebra systems on machines with large address spaces.

In this way one can automate much more of the Dissection method than has been done in LTAB. However, the general problem of choosing a particular dissection for a given problem is more difficult. Although it may now be possible to automate the entire dissection procedure, including choice of segmentation, for some classes of problems, it is likely that general dissector programs capable of treating many kinds of reduction problems will continue to require

human intervention.

Notes and References

[1] MACSYMA Reference Manual (Version 9), Mathlab Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1977.

[2] In Section 2 we use the term *list* to denote an ordered set of algebraic expressions. For many Computer Algebra systems, *list* has a specific technical meaning. This is not the sense in which the word is used here.

[3] A kernel of an expression is a subexpression that is not rationally simplified as a result of rational simplification of the expression itself. That is, its leading operator is not rational.

[4] G. Passarino and M. Veltmann, *Nucl. Phys.* B160, 151, (1979).

Acknowledgements

This work was carried out under Contract No. DE-AC-03-81ER40050 at the California Institute of Technology using MACSYMA, a symbol manipulation program developed at the MIT Laboratory for Computer Science and supported by the National Aeronautics and Space Administration under grant 1323, by the Office of Naval Research under grant ET-78-C-02-4687, and by the U.S. Air Force under grant F49620-79-C-020. The MACSYMA community, and especially J.P. Golden, has been very helpful. The author is also grateful to M.P. Shatz and to G.C. Fox for many useful discussions during the preparation of the manuscript.

Figures

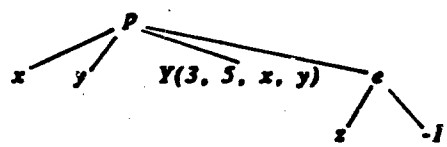


Figure 1.

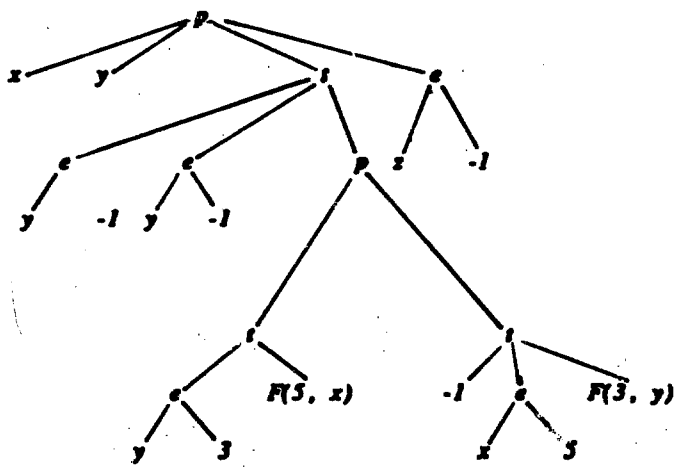


Figure 2.

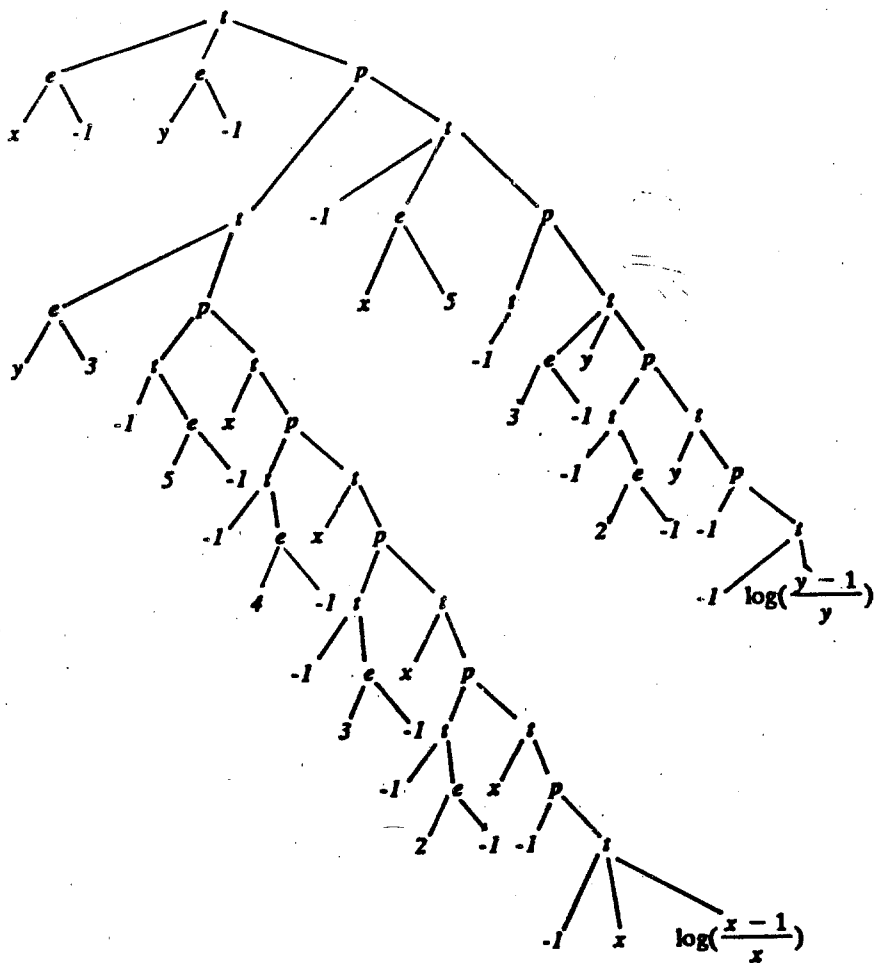

t
e
x
-1
e
y
-1
p
t
-1
e
x
5
p
t
-1
e
y
3
t
-1
e
x
p
5
-1
t
t
-1
e
x
p
4
-1
t
t
-1
e
x
p
3
-1
t
t
-1
e
x
p
2
-1
-1
t
-1
x
log(x − 1 / x)
t
-1
e
y
p
3
-1
t
t
-1
e
y
p
2
-1
-1
t
-1
log(y − 1 / y)

Figure 3a.

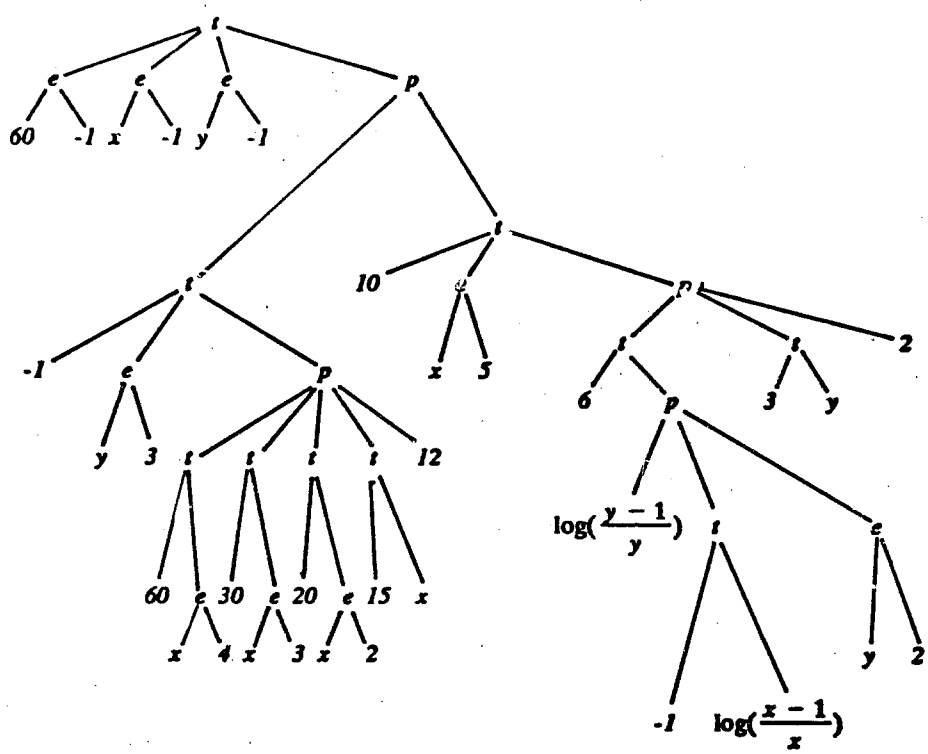

t
e
e
e
p
60
-1
x
-1
y
-1
t
t
10
p
-1
e
p
x
5
t
t
2
y
3
t
t
t
t
12
6
p
3
y
60
e
30
e
20
e
15
x
log(y − 1 / y)
t
e
x
4
x
3
x
2
y
2
-1
log(x − 1 / x)

Figure 3b.

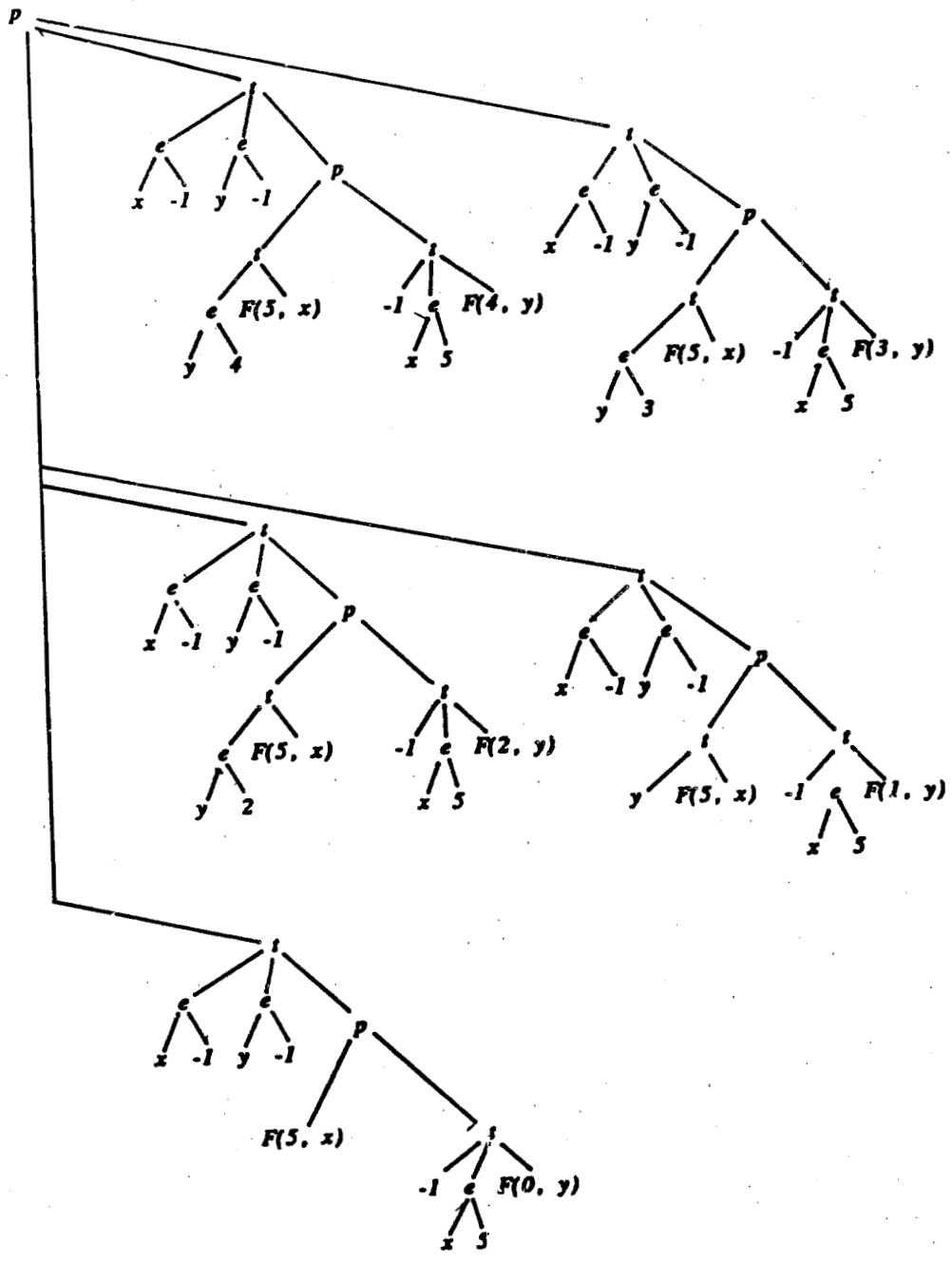

p
e
x
-1
y
-1
p
t
e
F(5, x)
y
4
t
-1
e
F(4, y)
x
5
t
e
x
-1
y
-1
p
t
e
F(5, x)
y
3
t
-1
e
F(3, y)
x
5
t
e
x
-1
y
-1
p
t
e
F(5, x)
y
2
t
-1
e
F(2, y)
x
5
t
e
x
-1
y
-1
p
t
y
F(5, x)
t
-1
e
F(1, y)
x
5
t
e
x
-1
y
-1
p
F(5, x)
t
-1
e
F(0, y)
x
5

Figure 4a.

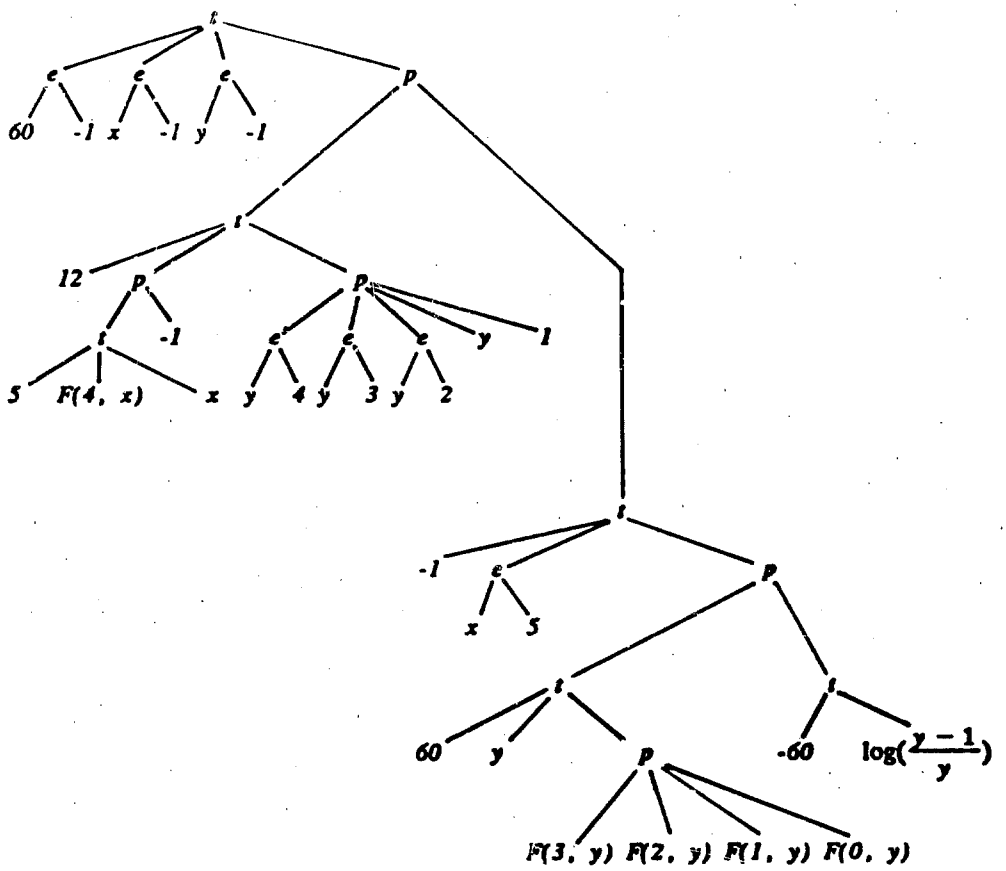

t
e
e
e
60
-1
x
-1
y
-1
p
t
12
p
p
t
-1
e
e
e
y
1
5
F(4, x)
x
y
4
y
3
y
2
t
-1
e
p
x
5
t
t
60
y
p
-60
log((y − 1)/y)
F(3, y)
F(2, y)
F(1, y)
F(0, y)

Figure 4b.

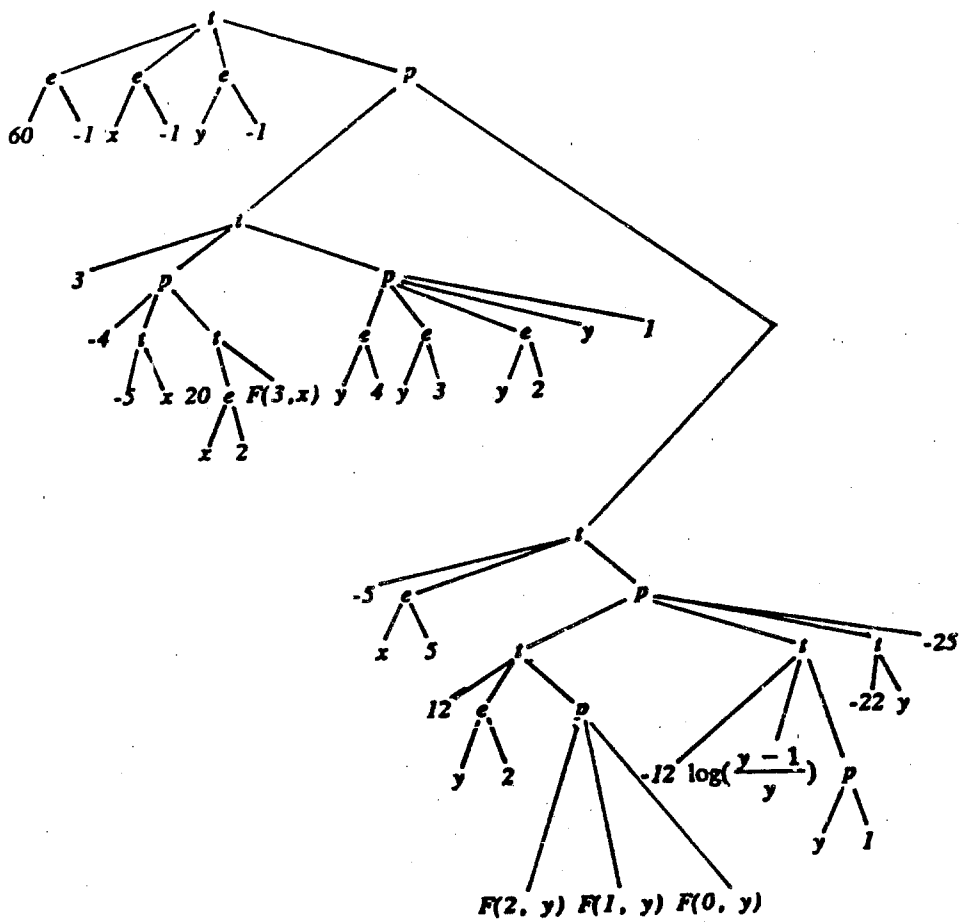

t
e
e
e
p
60
-1
x
-1
y
-1
t
3
p
p
-4
t
t
e
e
e
y
1
-5
x
20
e
F(3,x)
y
4
y
3
y
2
x
2
t
-5
e
p
x
5
t
t
t
-25
12
e
p
-22
y
y
2
-12
log(
y − 1
y
)
p
y
1
F(2, y)
F(1, y)
F(0, y)

Figure 4c.

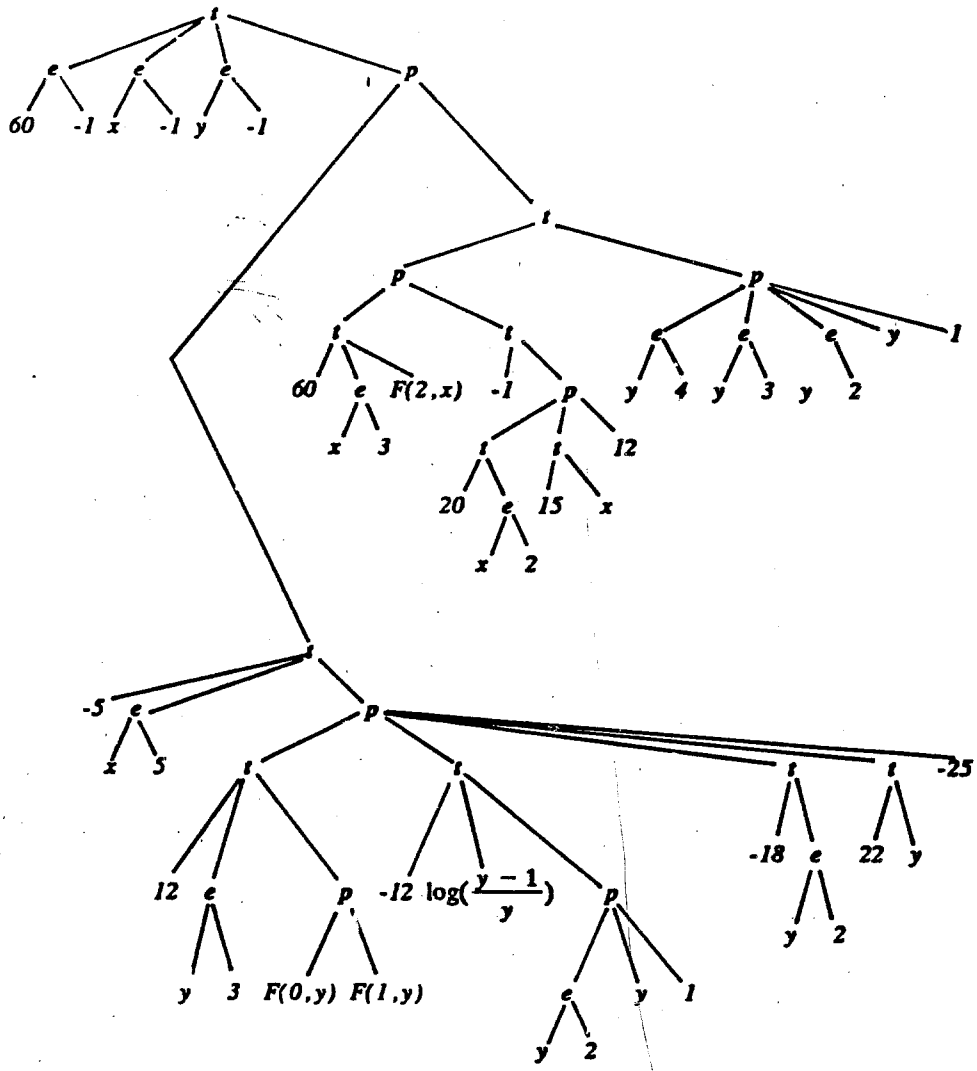

t
e
e
e
p
60
-1
x
-1
y
-1
t
p
p
t
t
e
e
e
y
1
60
e
F(2,x)
-1
p
y
4
y
3
y
2
x
3
t
t
12
20
e
15
x
x
2
t
-5
e
p
x
5
t
t
t
t
-25
12
e
p
-12
log(
y − 1
y
)
p
-18
e
22
y
y
3
F(0,y)
F(1,y)
y
2
e
y
1
y
2

Figure 4d.

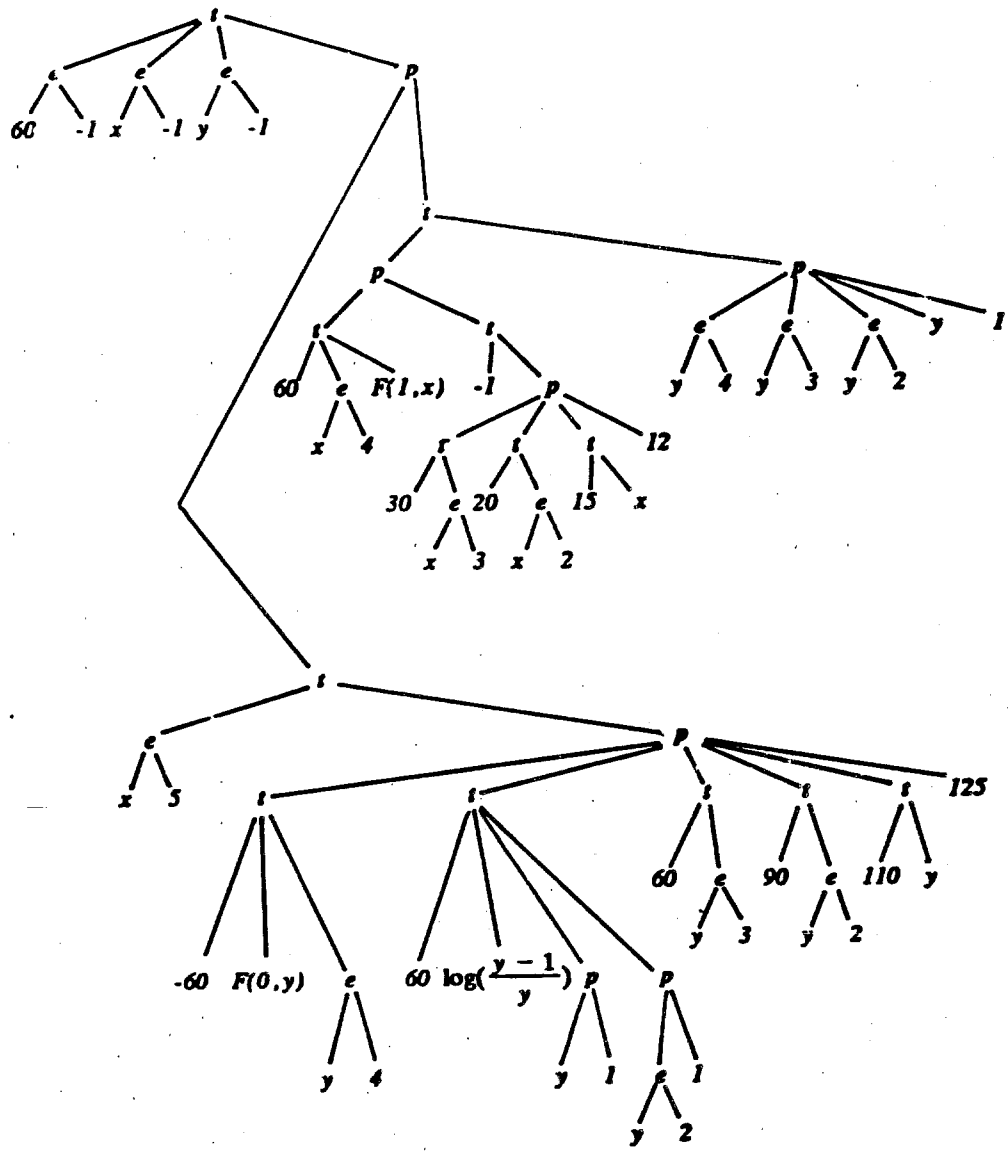

t
e
e
e
60 -1 x -1 y -1
p
t
p
p
t
t
e
e
e
y 1
60 e F(1,x) -1
p
y 4 y 3 y 2
x 4
t
t
t
12
30 e 20 e 15 x
x 3 x 2
t
e
x 5
p
t
t
t
t
t
125
60 e
90 e
110 y
y 3
y 2
-60 F(0,y) e
60 log((y − 1)/y) p
p
y 4
y 1
e 1
y 2

Figure 4e.

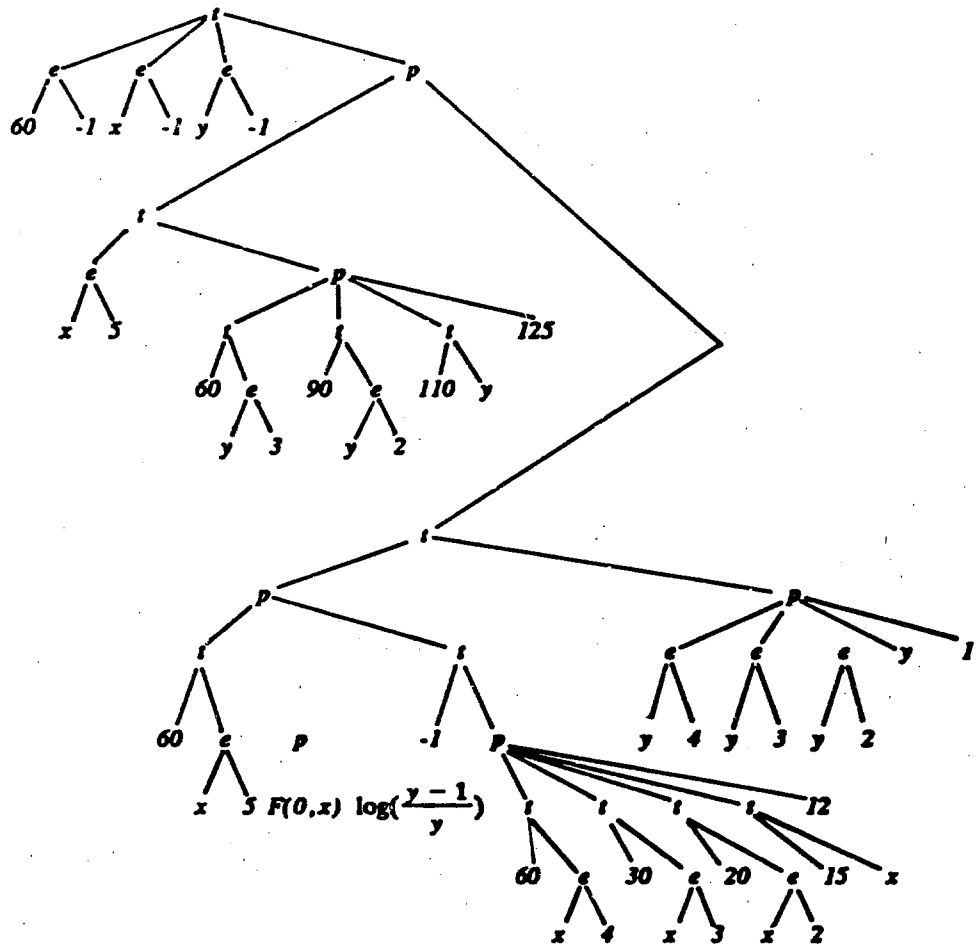

t
e
e
e
p
60
-1
x
-1
y
-1
t
e
p
x
5
t
t
t
125
60
e
90
e
110
y
y
3
y
2
t
p
p
t
t
e
e
e
y
1
60
e
p
-1
p
y
4
y
3
y
2
x
5
F(0,x)
log(y − 1 / y)
t
t
t
t
12
60
e
30
e
20
e
15
x
x
4
x
3
x
2

Figure 4f.

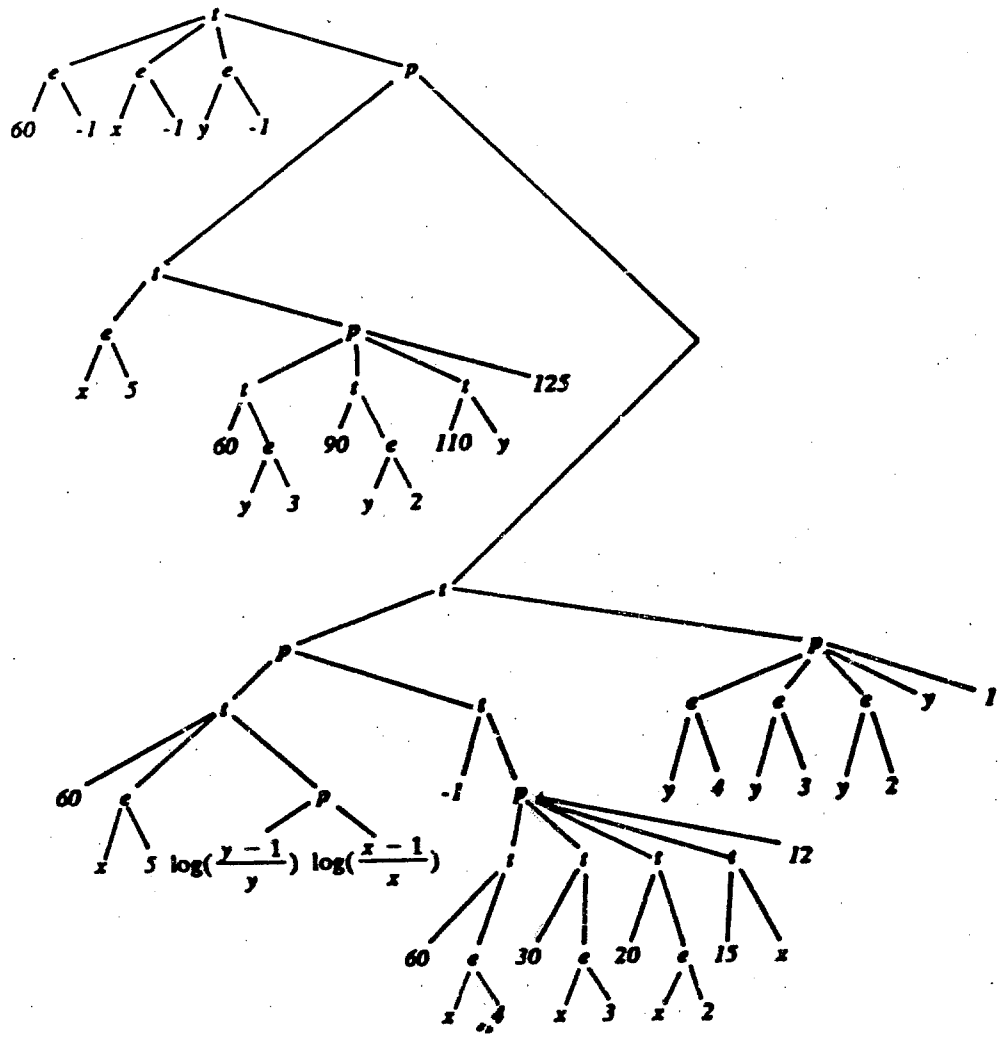

60 -1 x -1 y -1
x 5
60 90 110
y 3 y 2
125
60 x 5 log((y − 1)/y) log((x − 1)/x)
-1
y 4 y 3 y 2
y 1
60 30 20 15 x
x 4 x 3 x 2
12

Figure 4g.

Level 1	
Kernel	Value
$F(5, x)$	

Level 2	
Kernel	Value
$F(4, y)$	

Level 3	
Kernel	Value
$F(3, y)$	

Level 4	
Kernel	Value
$F(2, y)$	

Level 5	
Kernel	Value
$F(1, y)$	

Level 6	
Kernel	Value
$F(0, y)$	

Figure 5a.

Level 1	
Kernel	Value
$F(5, x)$	$x \quad F(4, x) - \frac{1}{3}$

Level 2	
Kernel	Value
$F(4, y)$	

Level 3	
Kernel	Value
$F(3, y)$	

Level 4	
Kernel	Value
$F(2, y)$	

Level 5	
Kernel	Value
$F(1, y)$	

Level 6	
Kernel	Value
$F(0, y)$	

Figure 5b.

Level 1	
Kernel	Value
$F(5, x)$	$x \quad F(4, x) - \frac{1}{5}$

Level 2	
Kernel	Value
$F(4, x)$	$x \quad F(3, x) - \frac{1}{4}$
$F(4, y)$	$y \quad F(3, y) - \frac{1}{4}$

Level 3	
Kernel	Value
$F(3, x)$	
$F(3, y)$	

Level 4	
Kernel	Value
$F(2, y)$	

Level 5	
Kernel	Value
$F(1, y)$	

Level 6	
Kernel	Value
$F(0, y)$	

Figure 5c.

Level 1	
Kernel	Value
$F(5, x)$	$x \quad F(4, x) - \frac{1}{5}$

Level 2	
Kernel	Value
$F(4, x)$	$x \quad F(3, x) - \frac{1}{4}$
$F(4, y)$	$y \quad F(3, y) - \frac{1}{4}$

Level 3	
Kernel	Value
$F(3, x)$	$x \quad F(2, x) - \frac{1}{3}$
$F(3, y)$	$y \quad F(2, y) - \frac{1}{3}$

Level 4	
Kernel	Value
$F(2, x)$	$x \quad F(1, x) - \frac{1}{2}$
$F(2, y)$	$y \quad F(1, y) - \frac{1}{2}$

Level 5	
Kernel	Value
$F(1, x)$	$x \quad F(0, x) - 1$
$F(1, y)$	$y \quad F(0, y) - 1$

Level 6	
Kernel	Value
$F(0, x)$	$-\log(\frac{x-1}{x})$
$F(0, y)$	$-\log(\frac{y-1}{y})$

Figure 5d.

Level 1	
Kernel	Value
$F(5, x)$	$x \quad F(4, x) - \frac{1}{5}$

Level 2	
Kernel	Value
$F(4, x)$	$x \quad F(3, x) - \frac{1}{4}$
$F(4, y)$	$y \quad F(3, y) - \frac{1}{4}$

Level 3	
Kernel	Value
$F(3, x)$	$x \quad F(2, x) - \frac{1}{3}$
$F(3, y)$	$y \quad F(2, y) - \frac{1}{3}$

Level 4	
Kernel	Value
$F(2, x)$	$x \quad F(1, x) - \frac{1}{2}$
$F(2, y)$	$y \quad F(1, y) - \frac{1}{2}$

Level 5	
Kernel	Value
$F(1, x)$	$-x \quad \log(\frac{x-1}{x}) - 1$
$F(1, y)$	$-y \quad \log(\frac{y-1}{y}) - 1$

Level 6	
Kernel	Value
$F(0, x)$	$-\log(\frac{x-1}{x})$
$F(0, y)$	$-\log(\frac{y-1}{y})$

Figure 5c.

Level 1	
Kernel	Value
$F(5, x)$	$-\log(\frac{x-1}{x})\, x^5 - x^4 - \frac{1}{2}x^3 - \frac{1}{3}x^2 - \frac{1}{4}x - \frac{1}{5}$

Level 2	
Kernel	Value
$F(4, x)$	$-\log(\frac{x-1}{x})\, x^4 - x^3 - \frac{1}{2}x^2 - \frac{1}{3}x - \frac{1}{4}$
$F(4, y)$	$-\log(\frac{y-1}{y})\, y^4 - y^3 - \frac{1}{2}y^2 - \frac{1}{3}y - \frac{1}{4}$

Level 3	
Kernel	Value
$F(3, x)$	$-\log(\frac{x-1}{x})\, x^3 - x^2 - \frac{1}{2}x - \frac{1}{3}$
$F(3, y)$	$-\log(\frac{y-1}{y})\, y^3 - y^2 - \frac{1}{2}y - \frac{1}{3}$

Level 4	
Kernel	Value
$F(2, x)$	$-\log(\frac{x-1}{x})\, x^2 - x - \frac{1}{2}$
$F(2, y)$	$-\log(\frac{y-1}{y})\, y^2 - y - \frac{1}{2}$

Level 5	
Kernel	Value
$F(1, x)$	$-\log(\frac{x-1}{x})\, x - 1$
$F(1, y)$	$-\log(\frac{y-1}{y})\, y - 1$

Level 6	
Kernel	Value
$F(0, x)$	$-\log(\frac{x-1}{x})$
$F(0, y)$	$-\log(\frac{y-1}{y})$

Figure 5f.

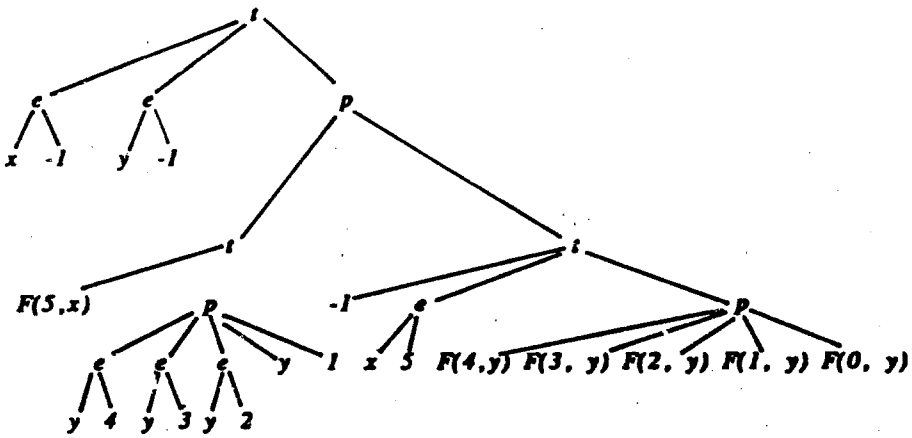


Figure 6a.

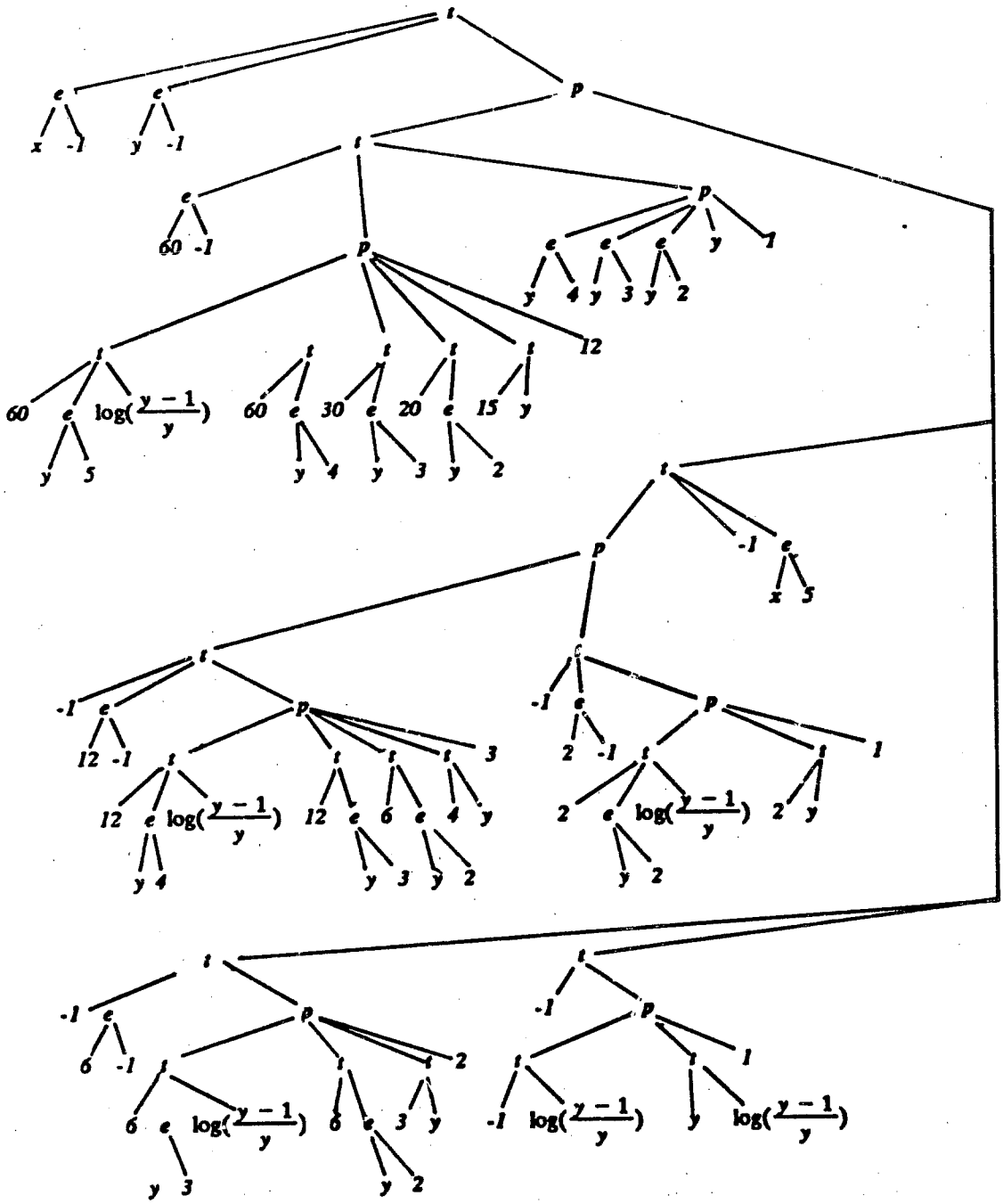

t
e
x -1
e
y -1
p
t
e
60 -1
p
e e e y 1
y 4 y 3 y 2
t t t t 12
60 e log((y − 1)/y)
y 5
60 e 30 e 20 e 15 y
y 4 y 3 y 2
t
p -1 e
x 5
t
-1 e p
12 -1
t t t t 3
12 e log((y − 1)/y)
y 4
12 e 6 e 4 y
y 3 y 2
t
-1 e p
2 -1 t t 1
2 e log((y − 1)/y) 2 y
y 2
t
-1 e p
6 -1
t t t 2
6 e log((y − 1)/y)
y 3
6 e 3 y
y 2
t
-1 p
t t 1
-1 log((y − 1)/y) y log((y − 1)/y)

Figure 6b.

Figure Captions

Figure 1.

The tree structure of the expression: $x + y + Y(3, 5, x, y) + \frac{1}{z}$.

Figure 2.

The tree structure of the expression:

$$x + y + \frac{y^3 F(5, x) - x^5 F(3, y)}{x\ y} + \frac{1}{z}.$$

Figure 3a.

The tree structure of the expression:

$$((x\ (x\ (x\ (-x \log(\frac{x-1}{x}) - 1) - \frac{1}{2}) - \frac{1}{3}) - \frac{1}{4}) - \frac{1}{5})y^3$$

$$- x^5(y\ (y\ (-\log(\frac{y-1}{y}) - 1) - \frac{1}{2}) - \frac{1}{3})/xy$$

Figure 3b.

The tree structure of the expression:

$$(10x^5(6(\log(\frac{y-1}{y}) - \log(\frac{x-1}{x}) + y^2) + 3y + 2)$$

$$- (60x^4 + 30x^3 + 20x^2 + 15x + 12)y^5)/60xy$$

Figure 4a.

The tree structure of the expression:

$$\frac{F(5,x)y^4 - F(4,y)x^5}{xy} + \frac{F(5,x)y^3 - F(3,y)x^5}{xy} + \frac{F(5,x)y^2 - F(2,y)x^5}{xy}$$

$$\frac{F(5,x)y\ 1 - F(1,y)x^5}{xy} + \frac{F(5,x) - F(0,y)x^5}{xy} +$$

Figure 4b.

After applying (2.2) to the expression in Figure 4a, and after some simplification we obtain:

$$(12(5xF(4,x) - 1)(y^4 + y^3 + y^2 + y + 1)$$

$$- x^5(60y\ (F(3,y) + F(2,y) + F(1,y) + F(0,y)) - 60\log(\frac{y-1}{y}) - 125)/60xy$$

This figure illustrates the tree structure of this expression.

Figure 4c.

After applying (2.2) to the expression in Figure 4b, and after some simplification we obtain:

$$(3(20x^2F(3,x) - 5x - 4)(y^4 + y^3 + y^2 + y + 1)$$

$$- 5x^5(12y^2(F(2,y) + F(1,y) + F(0,y))$$

$$- 12\log(\frac{y-1}{y})(y + 1) - 22y - 25)/60xy$$

This figure illustrates the tree structure of this expression.

Figure 4d.

After applying (2.2) to the expression in Figure 4c, and after some simplification we obtain:

$$(60x^3F(2, x) - (20x^2 + 15x + 12)(y^4 + y^3 + y^2 + y + 1)$$

$$- 5x^5(12y^3(F(1, y) + F(0, y)))$$

$$- 12\log(\frac{y-1}{y})(y^2 + y + 1) - 18y^2 - 22y - 25)/60xy$$

This figure illustrates the tree structure of this expression.

Figure 4e.

After applying (2.2) to the expression in Figure 4c, and after some simplification we obtain:

$$(60x^4F(1, x) - (30x^3 + 20x^2 + 15x + 12)(y^4 + y^3 + y^2 + y + 1)$$

$$x^5(-60F(0, y)y^4 + 60\log(\frac{y-1}{y})(y + 1)(y^2 + 1)$$

$$60y^3 + 90y^2 + 110y + 125))/60xy$$

This figure illustrates the tree structure of this expression.

Figure 4f.

After applying (2.2) to the expression in Figure 4c, and after some simplification we obtain:

$$((60x^{5}F(0, x) + \log(\frac{y-1}{y})))$$

$$-(60x^{4} + 30x^{3} + 20x^{2} + 15x + 12)(y^{4} + y^{3} + y^{2} + y + 1)$$

$$x^{5}(60y^{3} + 90y^{2} + 110y + 125))/60xy$$

This figure illustrates the tree structure of this expression.

Figure 4g.

After applying (2.2) to the expression in Figure 4f, and after some simplification we obtain:

$$((60x^{5}(\log(\frac{x-1}{x}) + \log(\frac{y-1}{y}))$$

$$-(60x^{4} + 30x^{3} + 20x^{2} + 15x + 12))$$

$$(y^{4} + y^{3} + y^{2} + y + 1)$$

$$x^{5}(60y^{3} + 90y^{2} + 110y + 125))/60xy$$

This figure illustrates the tree structure of this expression.

Figure 5a.

The six-level hierarchy obtained by extracting the foreground kernels from (2.11).

Figure 5b.

The result of applying (2.2) to the highest level in Figure 5a.

Figure 5c.

New foreground kernels generated in Figure 5b have been entered in the table for later evaluation.

Figure 5d.

All foreground kernels that were inserted in the table from the original expression have been expanded according to (2.2). This resulted in the discovery of several new foreground kernels which were in turn inserted into the table and expanded. The procedure that was used was an extension of the one used to generate Figures 5a-c, applied sequentially to each of the 6 levels.

Figure 5e.

New foreground kernels generated in Figure 5b have been entered in the table for later evaluation.

Figure 5f.

The expansions of the kernels of Level 6 resulted in no new foreground kernels. These expressions were then substituted into the expansions of the foreground kernels that were being held in Level 5. The result is that now the expansions of the Level 5 kernels are expressed entirely in terms of background variables.

Figure 5g.

The procedure that was used to obtain Figure 5f was repeated for each of the other 5 levels.

Figure 6a.

The tree structure of the expression of Eq. (2.12).

$$(F(5, x)(y^4 + y^3 + y^2 + y + 1)$$

$$- (F(4, y) + F(3, y) + F(2, y) + F(1, y) + F(0, y))x\ 5/xy.$$

Figure 6b.

The tree structure of the expression obtained by substituting the results shown in Figure 5g into (2.12).

SOLUTION OF SIMULTANEOUS POLYNOMIAL EQUATIONS BY ELIMINATION IN MACSYMA

William A. Beyer
Theoretical Division, MS B284
Los Alamos National Laboratory
Los Alamos, NM 87545 USA

Abstract

Discussion of the solution of simultaneous polynomial equations by the classical elimination algorithm is given. This algorithm is compared with more recent Newton and homotopy algorithms. The MACSYMA implementation of elimination theory is reviewed and its shortcomings are commented on. Directions for future work are discussed.

1. INTRODUCTION

At present the principal algorithms for solving simultaneous polynomial equations with real coefficients are Newton's algorithm in several variables [5] and homotopy algorithms due to Li, York, Garcia, Zangwill and Morgan [3]. The disadvantage of the Newton algorithm is that it is not easy to ensure that all solutions to the system are obtained. Homotopy algorithms have the disadvantage that they depend on solving initial value problems for nonlinear differential equations and the solution curves may involve singularities, at least as far as the numerical implementation is concerned.

In this paper we discuss the MACSYMA implementation of the oldest algorithm for solving simultaneous polynomial equations: elimination, which goes back to Babylonian times circa 1700 B.C. It was subsequently developed by Euler, Bézout, Sylvester, and others. Ultimately, elimination theory evolved into algebraic geometry, where it lost its algorithmic flavor. For a history see van der Waerden [9]. The best summary of the algorithmic aspects of elimination theory is given in the first 29 pages of Macaulay [2]. The treatment is old-fashioned. A modern treatment seems not to be available and we will not provide it. Future work on the topic is suggested.

Our experience is that elimination algorithms are more practical and more useful than the Newton or the homotopy algorithms for solving polynomial systems.

2 ELIMINATION AND THE RESULTANT

A following polynomial system is found in a cuneiform text from the first Babylonian dynasty circa 1700 B.C.:

$$x^2 + y^2 + z^2 = 1400 \qquad (2.1a)$$

$$x - y = 10 \qquad (2.1b)$$

$$y - z = 10 \quad . \qquad (2.1c)$$

The system is solved in the text by using (2.1b) and (2.1c) to express x and y in terms of z and substituting into (2.1a) to obtain a quadratic equation for z. The quadratic equation is solved for z and x and y are obtained by back substitution, yielding the two solutions

$$[30, 20, 10] \quad , \quad [-10, -20, -30] \quad . \qquad (2.1d)$$

The MACSYMA system carries out a similar procedure, using the subroutine ALGSYS, short for algebraic system:

```
(c1) ALGSYS  ([x ^ 2 + y ^ 2 + z ^ 2 = 1400  ,

                x - y - 10  , y - z - 10]  , [x, y, z])  ,
```

which yields (2.1d).

ALGSYS uses the method of the resultant, which we now review following the exposition in Muir and Metzler [4]. Let

$$a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = 0 \quad , \tag{2.3a}$$

$$b_0x^m + b_1x^{m-1} + \cdots + b_{m-1}x + b_m = 0 \quad , \tag{2.3b}$$

be a pair of equations in a single variable x. Multiplying (2.3a) by x^i $(0 \le i \le m-1)$ and (2.3b) by x^j $(0 \le j \le n-1)$, we obtain the n+m system

$$a_0x^{n+i} + a_1x^{n+i-1} + \cdots + a_nx^i = 0 \quad 0 \le i \le m-1 \tag{2.4a}$$

$$b_0x^{m+j} + b_1x^{m+j-1} + \cdots + b_mx^j = 0 \quad 0 \le j \le n-1 \quad . \tag{2.4b}$$

The system (2.4) is a nonhomogeneous system of n+m linear equations in the n+m variables x^{n+m}, x^{n+m-1}, $\cdots$, x. In order that this system have a solution for x, $\cdots$, x^{n+m}, it is

necessary that the resultant

$$R = \begin{vmatrix} a_0 a_1 \cdots a_n & & \\ & a_0 \cdots a_n & \\ & & \ddots \\ & & a_0 a_1 \cdots a_n \\ b_0 b_1 \cdots b_m & & \\ & \ddots & \\ & & b_0 b_1 \cdots b_m \end{vmatrix} \qquad (2.5)$$

vanish. The resultant can be applied to a pair of equations by treating one variable (usually of lowest order) as a variable and the rest of the variables as constants. The resultant of the pair of equations is then regarded as a single equation with the variable eliminated. For example, the resultant with respect to x of (2.1a) and (2.1b) is

$$z^2 + 2y + 20y - 1300 \qquad (2.6a)$$

and of (2.1a) and (2.1c) is

$$(z - y + 10)^2 \quad . \qquad (2.6b)$$

The resultant of (2.6) with respect to y is

$$9(z^2 + 20z - 300)^2 \quad . \tag{2.7}$$

The roots of (2.7) are z = -30, 10. Back substitution then yields (2.1d).

More generally, one can use the subroutine RESULTANT in Macsyma. The general procedure will be discussed in the next section.

3. ALGSYS

The subroutine ALGSYS solves a system of n polynomial equations with integer coefficients in $m \geq n$ variables as follows. The equations with right side zero are first factored into polynomial factors of degree > 0 over the domain of integers and the system is split into a system of systems of irreducible polynomial equations. For each system, a succession of resultants are calculated and variables are eliminated until one is left with a single equation which may be multivariate or univariate. If the single equation is univariate, the subroutine ALLROOTS is called to solve the equation. (The routine ALLROOTS solves the general univariate polynomial equation (with real coefficients) over the complex field. It obtains all the roots with their multiplicity by the method of Jenkins and Traub [1]). The roots of the univariate equation are then back substituted to obtain the general solution.

If the single equation is multivariate and of degree < 5 in some variable, the equation is solved as an equation of degree < 5 in terms of the remaining variables. If the degree of each of the variables is ≥ 5, the solution procedure is terminated.

The algorithm ALGSYS has at least several shortcomings. One is that if the final resultant is identically zero, the algorithm yields the empty set, which may not be the correct answer. Also, it is sometimes not clear what the complete algorithm is. These shortcomings can be overcome by resorting to the algorithms RESULTANT and BEZOUT, where the procedure is unambiguous.

We call attention to the "flags" ALGEXACT (affecting the method of solution of univariate polynomials) and ALGEPSILON which affects the accuracy of the solution of univariate polynomials.

4. BEZOUTIAN

A method of producing an n-rowed determinant as the eliminant of two polynomial equations of degree n was given by Bézout in 1779. See Turnbull [7]. Suppose we have

$$F(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \cdots + a_{n-1}x + a_n = 0 \qquad (4.1)$$

$$\phi(x) = b_0x^n + b_1x^{n-1} + b_2x^{n-2} + \cdots + b_{n-1}x + b_n = 0 \qquad (4.2)$$

where $a_0 \neq 0$. Multiply (4.1) by b_0 and (4.2) by a_0 and subtract to obtain

$$|a_0b_1|x^{n-1} + |a_0b_2|x^{n-2} + |a_0b_3|x^{n-3} + \cdots + |a_0b_n| = 0 \ , \qquad (4.3)$$

where

$$|a_0 b_i| = \begin{vmatrix} a_0 & a_i \\ b_0 & b_i \end{vmatrix} \quad .$$

Multiply (4.1) by $b_0 x + b_1$ and (4.2) by $a_0 x + a_1$ and subtract to obtain

$$|a_0 b_2| x^{n-1} + (|a_0 b_3| + |a_1 b_2|) x^{n-2} + (|a_0 b_4| + |a_1 b_3|) x^{n-3} + \cdots + (|a_0 b_n| + |a_1 b_{n-1}|) x + |a_1 b_n| = 0 \quad . \tag{4.4}$$

and so on, the final equation being

$$|a_0 b_n| x^{n-1} + |a_1 b_n| x^{n-2} + \cdots + |a_{n-2} b_n| x + |a_{n-1} b_n| = 0 \quad . \tag{4.5}$$

The quantities x^{n-1}, x^{n-2}, ..., x are then in succession eliminated from (4.3), (4.4) and (4.5) and finally a determinantal equation not involving the variable x is obtained. The determinant is called the Bezoutian of the systems (4.1) - (4.2). Evidently, the Bezoutian has the same value as the resultant, although we do not have an explicit statement of this.

5. EXAMPLES

The following are ten test examples taken from a report by Morgan [3] where the examples were used to test homotopy methods of solving polynomial systems due to Li, York, Garcia, and Zangwill. The examples were all solved correctly and easily using the MACSYMA routine ALGSYS. Some of the examples were selected because traditional methods of solutions had difficulty.

	Example	Number and Type of Solutions
1.	$x^2 + xy - 1 = 0$ $y^2 + x - 5 = 0$	4 real solutions
2.	$4x^3 - 3x - y = 0$ $x^2 - y = 0$	3 real solutions
3.	$4(x+y) = 0$ $4(x+y) + (x-y)((x-2)^2 + y^2 - 1) = 0$	1 real and two complex solutions.
4.	$x_1^2 + x_2^2 - 1 = 0$ $\det \begin{bmatrix} x_1-a_1 & x_2-a_2 & x_3-a_3 \\ x_1-b_1 & x_2-b_2 & x_3-b_3 \\ 2x_1 & 2x_2 & 0 \end{bmatrix} = 0$	4 real solutions 6 complex solutions
	$\lvert x-b\rvert^2 \, (x-a, N)^2$ $-\lvert x-a\rvert^2 \, (x-b, N)^2 = 0$	(problem from geometric optics)

where

$x = (x_1, x_2, x_3)$,

$a = (-1, -10, 0)$,

$b = (1, -10, 0)$.

5. $x^2 + 2y^2 - 4 = 0$
$x^2 + y^2 + z - 8 = 0$
$(x-1)^2 + (2y - \sqrt{2})^2 + (z-5)^2 - 4 = 0$ — 2 real solutions, 6 complex solutions

6. $x + 10y = 0$
$z + w = 0$
$(z-2z)^2 = 0$
$(x-w)^2 = 0$. — $(0, 0, 0)$

7. $x+y+z+w - 1 = 0$
$x+y-z+w-3 = 0$
$x^2 + y^2 + z^2 + w^2 - 4 = 0$
$(x-1)^2 + y^2 + z^2 + w^2 = 0$ — two real solutions

8. $x_1^2 + x_2^2 + x_3^2 + x_4 - 6 = 0$
$x_1 + x_2 + x_5 + x_6 - 1 = 0$
$x_1 + 3x_3 + x_6 - 3 = 0$
$x_1 + x_2 + x_4 - 2 = 0$
$2x_5 + x_6 = 0$
$x_5 + x_6 = 0$ — 2 real solutions

9. $x_k + \sum_{j=1}^{5} x_j - 6 = 0 \qquad 1 \leq k \leq 4$
$\prod_{j=1}^{5} x_j - 1 = 0$ — 3 real solutions

10. $x^2 + y^2 - 1 = 0$
$x^2 + y^2 + z^2 - 5 = 0$ — 4 real one-parameter surfaces

6. GENERAL SIMULTANEOUS QUADRATIC EQUATIONS

The general pair of simultaneous quadratic equations in two variables has the form

$$\begin{aligned} &m_{11}x_1^2 + 2m_{12}x_1x_2 + m_{22}x_2^2 + m_1x_1 + m_2x_2 + m_3 = 0 \quad , \\ &n_{11}x_1^2 + 2n_{12}x_1x_2 + n_{22}x_2^2 + n_1x_1 + n_2x_2 + n_3 = 0 \quad . \end{aligned} \tag{6.1}$$

The form of the resultant has been given by P. R. Stein [6] as

$$\begin{aligned} &x_1^4(P_1P_3 + P_7^2) + X_1^3(P_1P_4 + P_2P_3 + 2P_7P_8) \\ &\qquad + X_1^2(P_1P_5 + P_2P_4 + P_8^2 + 2P_7P_9) \\ &\qquad + X_1(P_1P_6 + P_2P_5 + 2P_8P_9) + P_2P_6 + P_9^2 = 0 \end{aligned} \tag{6.2}$$

where

$$\begin{aligned} P_1 &= 2(m_{12}n_{22} - n_{12}m_{22}) \quad , \\ P_2 &= n_{22}m_2 - m_{22}n_2 \quad , \\ P_3 &= 2(m_{12}n_{11} - n_{12}m_{11}) \quad , \\ P_4 &= n_{11}m_2 - m_{11}n_2 + 2m_{12}n_1 - 2n_{12}m_1 \quad , \\ P_5 &= 2m_{12}n_3 - 2n_{12}m_3 + m_2n_1 - n_2m_1 \quad , \\ P_6 &= m_2n_3 - n_2m_3 \quad , \\ P_7 &= m_{22}n_{11} - n_{22}m_{11} \quad , \\ P_8 &= m_{22}n_1 - n_{22}m_1 \quad , \end{aligned} \tag{6.3}$$

and

$$P_9 = m_{22}n_3 - n_{22}m_3 .$$

Formulae (6.2) and (6.3) have been verified by MACSYMA. However, these days the utility of such formulae may be questioned.

We applied MACSYMA to the general triple of quadratic equations in three variables:

$$Q^i_{11}x_1^2 + Q^i_{22}x_2^2 + Q^i_{33}x_3^2 + Q^i_{12}x_1x_2 + Q^i_{13}x_1x_3 + Q^i_{23}x_2x_3 + L^i_1x_1 + L^i_2x_2 + L^i_3x_3 + C^i = 0 \tag{4.4}$$

for i = 1, 2, 3. MACSYMA did yield a formula for the general resultant of the three equations; but the formula filled the VAX and a partial printout did not reveal anything interesting.

Acknowledgment. The author thanks Paul R. Stein for information on the classical elimination method and for introducing him to elimination theory.

REFERENCES

1. Jenkins, M. A. and Traub, J. F. A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. Numer. Math. 14 (1970), 252-263.
2. Macaulay, F. S. The Algbraic Theory of Modular Systems. Cambridge University Press, Cambridge, 1916.
3. Morgan, A. P. A method for computing all solutions to systems of polynomial equations. GMR-3651, General Motors Research Laboratory, Warren, MI, July 1981.
4. Muir, T. and Metzler, W. A. A Treatise on the Theory of Determinants, Dover, New York, 1960.
5. Ortega, J. M. and Rheinboldt, W.C. Iterative Solution of Nonlinear Equations, Academic Press, New York, 1970.
6. Stein, P. R. Elimination theory applied to general systems of quadratic equations. Unpublished manuscript, Los Alamos National Laboratory, Los Alamos, about 1959.
7. Turnbull, H. W. Theory of Equations. Oliver and Boyd, London, 1947.
8. van der Waerden, B. L. Science Awakening. P. Noordhoff, Groningen, 1954.
9. van der Waerden, B. L. The foundation of algebraic geometry from Severi to André Weil. Archive for History of Exact Sciences 7 (1971), 171-180.

AN OVERDETERMINED SYSTEM OF PARTIAL DIFFERENTIAL EQUATIONS

David H. Wood
Code 3332, New Londc Laboratory
Naval Underwater Systems Center
New London, Connecticut 06320 USA

Abstract

We use MACSYMA to address the question, "What equations must be satisfied by the ratio of two solutions of a given linear partial differential equation in two variables?" The answer for ordinary differential equations is already known: the ratio satisfies a third order nonlinear ordinary differential involving the "Schwarzian derivative," which is known to be invariant under bilinear mappings. For partial differential equations, we find the ratio must satisify two fifth order partial differential equations, and we struggle to express them in terms of generalizations of the Schwarzian derivative to functions of more than one variable. A description of the method will make the need for MACSYMA obvious. Let U and V be two solutions of the partial differential equation, and let W be their ratio. Substituting both U and U*W into the equation and taking their difference, we obtain a first order partial differential equation for U with coefficients determined by first and second order partial derivatives of W. We get a new equation by differentiating this equation with respect to X and another by differentiating with respect to Y. These two equations, along with the original equation satisfied by U, are solved for the three second order partial derivatives of U. For these to be analytically consistent, the mixed third order derivatives are computed and equated. This gives two identical linear constraints on U and its first derivatives having coefficients depending on partial derivatives of W. One of these equations is solved for the two first order partial derivatives of U by using the previously mentioned first order partial differential equation for U. As before, the mixed partial derivatives of second order must be obtainable from differentiating either of the first order derivatives. When this is done, it results finally in two consistency conditions involving derivatives of first to fifth order in W. These are the two equations that w must satisfy.

APPLICATIONS OF MACSYMA IN SOLVING LINEAR SYSTEMS OF DIFFERENTIAL EQUATIONS

Leo P. Harten
MIT
Cambridge, MA 02139
and
Paradigm Associates, Inc.
29 Putnam Avenue, Suite 6
Cambridge, MA 02139

Abstract

The exponential of a matrix plays an important role in the solution of coupled differential equations. Some programs that interface to the DESOLVE routine are presented along with results for linear systems. A linearized analysis of stability of autonomous non-linear systems can be carried out about the equilibria. Certain Matrix Riccati equations can be converted into linear systems and then solved.

1. INTRODUCTION

The exponential of a square matrix A is defined by the power series:

```
exp( A ) = sum( A↑↑i / i! , i , 0 , inf)                    (1)
```

(the 0 power is interpreted as the identity matrix I.)

There are a great many applications in pure and applied sciences of linear systems of coupled differential equations which can be written:

$$\mathrm{diff}(U , t) = A \cdot U \qquad (2)$$

where U is either a vector or a matrix, and A is a constant N x N matrix (the dot stands for non-commutative multiplication.) The case of vector U is typically included in an undergraduate course on differential equations, but the treatment is often limited to cases in which the matrix A has a complete set of eigenvectors.

The solution to Eq. (2) is given by:

$$U(t) = \exp(A * t) \cdot U(0) \qquad (3)$$

It is often not evidently simple to compute exp(A * t) from Eq. (1), unless A satisfies some auxilliary condition such as A↑↑3=0. A more powerful method is to do a similarity transformation on A and put it in diagonal form:

$$A = S\uparrow\uparrow(-1) \cdot LAM \cdot S \qquad (4)$$

where S is the matrix whose columns are the eigenvectors of A, and LAM is the diagonal matrix of eigenvalues. However, not all matrices are diagonalizable, and thus this method will not always be useful. When A possesses N linearly independent eigenvectors,

$$\exp(A) = \exp(S\uparrow\uparrow(-1) \cdot LAM \cdot S)$$

$$= S\uparrow\uparrow(-1) \cdot \exp(LAM) \cdot S \qquad (5)$$

and exp(LAM) is a diagonal matrix with [i,i] element = exp(LAM[i ,i]). The EIGEN package [1] will find the similarity transformation, if it exists.

When A does not possess a complete set of eigenvectors, it is not diagonalizable, and a different approach is needed. The use of Laplace Transforms turns out to be useful for this case, and they also work when A is diagonalizable. Thus, the use of the DESOLVE program [1] is an ideal method for routine solution of coupled linear systems. Some programs that interface with the DESOLVE routine are presented and discussed.

In Section 2, the case of vector U is handled. Section 3 gives the results for matrix U. The linearized stability analysis of autonomous systems is discussed in Section 4, and Section 5 gives the reduction of a Matrix Riccati equation to a linear system.

MACSYMA is thus found to be capable of handling a variety of important systems of coupled differential equations. Only the homogeneous case is here considered for the linear systems; but Laplace transforms can handle inhomogeneous cases as well.

2. SOLUTION FOR VECTOR U

When U is an N-dimensional vector in Eq. (2), the solution is obtained by first finding the N eigenvalues lam[n] and the M (M <= N) eigenvectors v[m] of A (see [2]).

For each distinct eigenvalue, denoted by j, lam[j], there exists an eigenvector, v[j], and a solution

$$u[\,j\,] = const[\,j\,] * v[\,j\,] * \exp(\,lam[\,j\,] * t\,). \qquad (6)$$

For K repeated eigenvalues, denoted by r and K, lam[r , K], there may exist L (L <= K) linearly independent eigenvectors, v[r , l]. A solution is then

$$u[\,r\,] = sum(\,const[s] * v[\,r\,,\,s\,] * t\uparrow s * \exp(\,lam[\,r\,,\,s\,] * t\,)\,, \qquad (7)$$

$$s\,,\,0\,,\,L - 1\,).$$

If L < K then K - L additional solutions, denoted by q, of the form

$$u[\,q\,] = sum(\,w[\,q\,,\,p\,] * t\uparrow p * \exp(\,lam[\,q\,,\,p] * t)\,, \qquad (8)$$

$$p\,,\,0\,,\,K - L - 1\,)$$

must be sought, where the vectors w (not eigenvectors) are to be determined.

These calculations are somewhat laborious in general. Using the DESOLVE routine is far superior to hand computation

when the characteristic equation for the matrix A has quadratic or linear factors over the integers.

The command VECODE(A,U0); will solve dU/dt = A . U with U(0) = U0. The user specifies the matrix A and the list U0 of initial values. The output is U(t), stored in the global variable UV. To solve the same equation with a different set of initial conditions, U(0) = NEW_U0, the user runs RESOLV(NEW_U0), which avoids the re-computation of U(t). Thus for each matrix A there is one call to VECODE, and for each set of initial conditions after the first there is a call to RESOLV. The code is presented in Appendix 1.

3. SOLUTION FOR MATRIX U

The matrix system is converted into a vector system by adding the N rows of length N together to form one vector of length N*N. This requires a little bit of manipulation, but then the method is the same as in Section 2.

The command MATODE(A,U0); will solve for U(t), stored in the global variable UM, when the constant matrix A and the initial-value matrix U(0) = U0 are the input. After UM has been found once, different initial conditions can be imposed without the re-computation of UM by calling RESOLM(NEW_U0) with the new initial condition U(0) = NEW_U0. Thus for each matrix A there is exactly one call to the function MATODE, and for each initial

condition on U after the first there is one call to RESOLM. The code is presented in Appendix 2.

4. LINEARIZED STABILITY ANALYSIS ABOUT EQUILIBRIA OF AUTONOMOUS NON-LINEAR SYSTEMS

Autonomous non-linear systems are of the form

$$\begin{aligned} d(x1)/dt &= f1(x1,x2,\ldots,xn) \\ d(x2)/dt &= f2(x1,x2,\ldots,xn) \\ &\vdots \\ d(xn)/dt &= fn(x1,x2,\ldots,xn) \end{aligned} \qquad (9)$$

where the n unknowns are (x1,x2,...,xn) and the n functions (f1,f2,...,fn) depend only on the x's and not on the independent variable t.

The equilibria of this system are at points (x1a,x2a,...,xna) at which

$$f1(x1a,x2a,\ldots,xna) = f2(x1a,x2a,\ldots,xna) = \ldots = fn(x1a,x2a,\ldots,xna) = 0 . \qquad (10)$$

The object of linearized stability analysis is to determine the effect upon the system of a small displacement from such an equilibrium. If the perturbation tends to die out in time, the system is stable, such as for a ball at the bottom of a well which has friction. If the perturbation does not tend to change, then neutral equilibrium has been found - a ball on a flat plane. Finally, the perturbation may increase, resulting in instability, such as a ball perched on top of a hill which slopes down on both sides.

The SOLVE command may be used to find equilibria insofar as it is able to solve coupled non-linear systems, as for low order polynomial f's which result in a factorizable univariate polynomial upon elimination. The Taylor series command will then be able to expand the f's to first order in the x's around the equilibrium point. This linearized system will be homogeneous when the coordinate system is centered on the equilibrium. The results of Section 2 can now be used to determine the stability property: the solutions are exponentials involving the eigenvalues of a matrix, and as long as the real parts of the eigenvalues are negative there will be a decay back toward the equilibrium, and hence stability; if there is exactly one eigenvalue with real part equal to 0 (the rest being negative), neutral stability prevails; if more than one real part is 0, then a polynomial growth can occur and be unstable; and if any eigenvalue has a positive real part then exponential instability occurs. The program STAB in Appendix 3

can be called on simple systems to determine the stability property around the equilibria.

5. MATRIX RICCATI EQUATIONS WITH CONSTANT COEFFICIENTS

The non-linear and inhomogeneous differential equation:

$$\text{diff}(V , t) = A . V + V . AT + M - V . N . V \qquad (11)$$

where AT = Transpose(A), and A, M, and N are constant nxn matrices, and V is nxn - is a matrix Riccati equation. There are many problems in control, estimation, and scattering theories in which this equation appears. Please see Reference [3] for a list of references.

A transformation exists which converts the matrix Riccati equation into a linear system in twice as many variables. See Appendix 4 for a Macsyma derivation showing the equivalence. The linear system can then be solved as in Section 2.

6. ACKNOWLEDGMENTS

The author wishes to thank Dr. Ralph M. Wilcox of Hughes Aircraft Co. EDSG for the introduction to the subject of Matrix Riccati equations.

The use of the DESOLVE program written by R. Bogen, and of the EIGEN package written by Y. Gursel, aided this work greatly.

7. REFERENCES

1. MACSYMA Reference Manual, Version 9, The MATHLAB Group, Laboratory for Computer Science, MIT (1977)

2. Kaplan, W. *Ordinary Differential Equations*, Addison-Wesley Publishing Co., Inc. (1958)

3. Wilcox, R. M. and Harten, L. P. MACSYMA-Generated Closed-Form Solutions to Some Matrix Riccati Equations, Journal of Applied Mathematics and Computation, Vol. 14, pp 149-166 (1984)

APPENDIX 1

CODE FOR VECTOR U

```
/* Copyright Leo P.  Harten 1982, 1984 All Rights Reserved */
/* Permission is granted to use the code for any
   non-commercial purpose */

/* Sample call:
VECODE(matrix([1,3],[2,4]),[1,0]) */

/* UV will be a vector (matrix) in general, so mode is ANY */
DEFINE_VARIABLE(UV,'UV,ANY);

/* Define the function VECODE which solves d(UV)/dt=A.UV
   with UV(t=0)=U0 */
VECODE(A,U0):=(MODE_DECLARE([A,U0],ANY),
BLOCK([DIM,UL,EQ,EQNS],
/* temporary variables */
MODE_DECLARE([DIM],FIXNUM,[UL,EQNS],LIST,[EQ],ANY),
/* their modes */
DIM:LENGTH(A), /* get dimension of the problem */
/* simple error check */
IF LENGTH(U0)#DIM THEN
```

```
   ERROR("A AND U0 HAVE INCONSISTENT DIMENSIONS"),
/* generate a list UL containing U1(t),U2(t),...,UDIM(t) */
UL:MAKELIST(FUNMAKE(CONCAT('UV,I),['T]),I,1,DIM),
/* bind EQ to the equation and show the user */
PRINT(EQ:TRANSPOSE(DIFF(UL,T))=A.UL),
/* get all terms on one side */
EQ:LHS(EQ)-RHS(EQ),
/* make a list of all the equations */
EQNS:APPLY('APPEND,ARGS(EQ)),
/* call DESOLVE to solve by Laplace transform method */
UV:DESOLVE(EQNS,UL),
/* call the re-solver for vectors with the initial value U0 */
RESOLV(U0)))$

/* sample call:
RESOLV([0,1]) */

/* define the vector re-solver */
RESOLV(NEW_U0):=(MODE_DECLARE(NEW_U0,ANY),
BLOCK([DIM,UL,UL0],MODE_DECLARE(DIM,FIXNUM,UL,LIST,UL0,ANY),
/* temporary variable and mode */
DIM:LENGTH(NEW_U0), /* dimension of problem */
/* simple error check */
IF DIM#LENGTH(U) THEN
   ERROR("WRONG NUMBER OF INITIAL CONDITIONS"),
/* generate a list UL containing U1(t),U2(t),...,UDIM(t) */
UL:MAKELIST(FUNMAKE(CONCAT('UV,I),['T]),I,1,DIM),
/* supply initial values for UL from NEW_U0 */
UL0:MAP(LAMBDA([X,Y],SUBST(0,T,X)=Y),UL,NEW_U0),
/* return a vector from using UL0 in U */
TRANSPOSE(MAP('RHS,SUBST(UL0,U)))))$
```

APPENDIX 2

CODE FOR MATRIX U

```
/* Copyright Leo P. Harten 1982, 1984 All Rights Reserved */
/* Permission is granted to use the code for any
   non-commercial purpose */

/* sample call
MATODE(matrix([3,2],[-1,4]),ident(2)) */

/* UM is of mode any since it is a matrix */
DEFINE_VARIABLE(UM,'UM,ANY)$

/* define MATODE which solves d(UM)/dt=A.UM with matrix
   UM(t=0)=U0 */
```

```
MATODE(A,U0):=(MODE_DECLARE([A,U0],ANY),
     BLOCK([DEG,EQNS,UNK,EQ,ANS,GM], /* temporary variables */
               LOCAL(GM), /* GM is local to the block */
               MODE_DECLARE(DEG,FIXNUM,[EQNS,UNK,ANS],LIST,
                            GM,ANY), /* modes */
                  DEG:LENGTH(A), /* dimension of system */
                  /* define a matrix from an array with elements
                  U11(t), U12(t), ..., U1DEG(t)
                  .
                  .
                  .
                  UDEG1(t), UDEG2(t), ..., UDEGDEG(t) */
                  GM[I,J]:=FUNMAKE(CONCAT('U,I,J),['T]),
                  UM:GENMATRIX(GM,DEG,DEG),
                  EQ:DIFF(UM,T)-A.UM, /* here is the equation */
                  UNK:APPLY('APPEND,ARGS(UM)),
                  /* make a list of unknowns */
                  EQNS:APPLY('APPEND,ARGS(EQ)),
                  /* list of equations */
                  ANS:DESOLVE(EQNS,UNK),
                /* solve by Laplace transforms */
                  UM:SUBST(ANS,UM),
                /* use the answer to bind UM to general soln */
                  RESOLM(U0)))$ /* use the initial value */

/* sample call
RESOLM(matrix([1,e],[f,1])) */

/* define re-solver for matrix case */
RESOLM(NEW_U0):=(MODE_DECLARE(NEW_U0,ANY),
(IF NOT MATRIXP(UM) THEN /* simple error check */
                  ERROR("UM WAS NOT A MATRIX") ELSE
              IF FREEOF(U11(0),UM) THEN
               /* require that UM contain U11(0) */
                  ERROR("U11(0) DID NOT APPEAR IN UM") ELSE
            BLOCK([LU0,DEG], /* temporary variables */
                  MODE_DECLARE(LU0,LIST,DEG,FIXNUM), LU0:[],
                  /* modes */
                  DEG:LENGTH(U), /* dimension of system */
                /* DO loops to make list of UMij(0)=NEW_U0[i,j] */
                  FOR I:1 THRU DEG DO
                      FOR J:1 THRU DEG DO
                          LU0:CONS(APPLY(CONCAT('U,I,J),[0])=
                                        NEW_U0[I,J],LU0),
               SUBST(LU0,UM))))$ /* use initial values in UM */
```

APPENDIX 3

STABILITY ANALYSIS OF COUPLED AUTONOMOUS SYSTEMS

```
/* copyright 1983,1984 Leo P. Harten All Rights Reserved */
/* Permission is granted to use the code for any
   non-commercial purpose */

/* The routine STAB determines the stability to small
perturbations of a system of differential equations. Only
linearization is performed for this analysis, so the program
will predict neutral stability for the system dx/dt=x↑2+y↑2,
dy/dt=x↑2+y↑4 at the only real equilibrium (x0=0,y0=0), while
this system is non-linearly unstable for real (x,y). For
systems which have a non-trivial expansion to first order around
the equilibria, the eigenvalues of the matrix of coefficients
determines linear stability: if any eigenvalues have a positive
real part, the system is unstable; if the largest real part is
0, the system has neutral stability; and if all real parts are
negative, then the system is stable. */

/* Here is a sample call to the program STAB. Note that the
dependencies of X,Y, and Z on T must be stated explicitly with
the DEPENDS command.

DEPENDS([X,Y,Z],T);
DECLARE(A,CONSTANT);
EQ1:DIFF(X,T)=-A+X-Y;
EQ2:DIFF(Y,T)=Y*A-Z;
EQ3:DIFF(Z,T)=-X+Z*Y-1;
STAB(EQ1,EQ2,EQ3),A:1;
STAB(EQ1,EQ2,EQ3),A:-3;
ERRCATCH(STAB(EQ1,EQ2,EQ3));
 */

STAB([SYSTEM]):=(MODE_DECLARE(SYSTEM,ANY),
 /* SYSTEM must be of the form
    dx/dt=f(x,...,z),...,dz/dt=g(x,...,z) */
        BLOCK([NUMER,RATPRINT,LEN,EQ,DE,LOV,EQUIL,LOE,SOLS,TL,
               LIN,NL,PROD,LN,CHAR,DET,TEMP], /* temp values */
MODE_DECLARE([NUMER,RATPRINT],BOOLEAN,  /* modes */
             [LEN],FIXNUM,
             [EQ,DE,LOV,EQUIL,LOE,SOLS TL,LINS,NL,LIN,PROD,
              LN],LIST,
             [CHAR,DET,TEMP],ANY),
          NUMER:TRUE,RATPRINT:FALSE,
          LEN:LENGTH(SYSTEM), /* number of unknowns */
          EQ:MAP(RHS,SYSTEM), /* the functions f,...,g */
          DE:MAP(LHS,SYSTEM), /* the derivatives */
          LOV:MAP(FIRST,DE),  /* the unknowns */
```

```
EQUIL:SOLVE(EQ,LOV), /* call SOLVE to get the
                        equilibria */
IF EQUIL=[] THEN
   ERROR("SOLVE FOUND NO ROOTS"),
     /* complain if SOLVE could not handle */
     /* make a list [x=x0,...,z=z0]
         LOE:[], FOR I IN LOV DO
             LOE:ENDCONS(I=CONCAT(I,0),LOE),
     /* make a list of lists for x0=,...,z0= */
         SOLS:MAP(RECTFORM,SUBST(LOE,EQUIL)),
     /* make a list for Taylor of the form
        [[x,x0,1],...,[z,z0,1]] */
         TL:[], FOR I IN LOV DO
             TL:CONS([I,CONCAT(I,0),1],TL),
     /* linearize the functions f,...,g in x about
        x0,..., and in z about z0 */
         LIN:APPLY('TAYLOR,
               APPLY('CONS,[EQ,APPEND(TL)])),
     /* make a list [x=xn+x0,...,z=zn+z0] */
         NL:[], FOR I IN LOV DO
         NL:ENDCONS(I=CONCAT(I,N)+CONCAT(I,0),NL),
     /* replace x by xn+x0, etc., in the linearized
        system */
         LIN:SUBST(NL,LIN),
     /* make a list of products xn*yn=0 */
         PROD:[],
         FOR I IN LOV DO
         FOR J IN LOV WHILE J#I DO
             PROD:CONS(CONCAT(I,N)*CONCAT(J,N)=0,
                       PROD),
     /* remove products from the linearized system */
         LIN:LRATSUBST(PROD,LIN),
     /* make a list [xn,...,zn] */
         LN:[],
         FOR I IN LOV DO LN:ENDCONS(CONCAT(I,N),LN),
     /* characteristic equation of the matrix of
        coefficients of [xn,...,zn] */
         CHAR:COEFMATRIX(LIN,LN)-
              IDENT(LEN)*EIGENVALUE,
         DET:EXPAND(DETERMINANT(CHAR)),
     /* for each equilibrium compute DET,
        solve (for list of EIGENVALUEs),
        get realpart of EIGENVALUEs,
        find largest realpart,
        print the equilibrium point and
        its stability property */
         FOR I IN SOLS DO (TEMP:EV(DET,I,NUMER),
         TEMP:EV(SOLVE(TEMP),NUMER:FALSE),
         TEMP:EV(TEMP,NUMER:TRUE),
         TEMP:MAP(LAMBDA([U],REALPART(RHS(U))),TEMP),
         TEMP:APPLY('MAX,TEMP),
         IF TEMP>0. THEN PRINT("THE EQUILIBRIUM ",I,
               "IS UNSTABLE, MAX RATE OF GROWTH = ",
               TEMP)
```

```
      ELSE IF (TEMP=0 OR TEMP=0.) THEN
        PRINT("THE EQUILIBRIUM ",I,
              "HAS NEUTRAL STABILITY OR",
              "POLYNOMIAL GROWTH")
    ELSE IF TEMP<0. THEN
        PRINT("THE EQUILIBRIUM ",I,
              "IS STABLE"))))$
```

APPENDIX 4

EQUALITY OF MATRIX RICCATI EQUATION AND LINEAR SYSTEM

Demonstration that the Matrix Riccati Equation

dV/dt = A . V + V . transpose(A) + M - V . N . V

where A, M, and N are constant KxK matrices, and
V, X, and Y are t-dependent KxK matrices,

is equivalent to a linear system in X and Y where

V = X . Y↑↑(-1) .

```
(c4) /* TIME DEPENDENT FUNCTIONS */

DEPENDS([V,X,Y],T);
(d4)                          [v(t), x(t), y(t)]

(c5) /* MATRIX QUANTITIES */

DECLARE([V,X,Y,A,M,N],NONSCALAR);
(d5)                                done

(c6) /* RELATION OF V TO X AND Y */

V:X.Y↑↑(-1);
                                        <- 1>
(d6)                              x . y
```

```
(c7) /* EQN FOR DX/DT */

EQ1:DIFF(X,T)=A.X+M.Y;
                                dx
(d7)                            -- = m . y + a . x
                                dt

(c8) /* EQN FOR DY/DT */

EQ2:DIFF(Y,T)=N.X-TRANSPOSE(A).Y;
                               dy
(d8)                           -- = n . x - transpose(a) . y
                               dt

(c9) /* EQN FOR DV/DT */

EQ:DIFF(V,T)=A.V+V.TRANSPOSE(A)+M-V.N.V;
     dx    <- 1>        d    <- 1>
(d9) -- . y      + x . -- (y     ) =
     dt               dt

       <- 1>             <- 1>        <- 1>
- x . y      . n . x . y      + x . y      . transpose(a) + m

            <- 1>
 + a . x . y

(c10) /* USE THE DX/DT FROM EQ1 IN DV/DT */

EQ:EQ,EQ1;
          d    <- 1>                          <- 1>
(d10) x . -- (y     ) + (m . y + a . x) . y      =
          dt

       <- 1>             <- 1>        <- 1>
- x . y      . n . x . y      + x . y      . transpose(a) + m

            <- 1>
 + a . x . y

(c11) /* REPLACE D(Y↑↑(-1))/DT */

EQ:SUBST(-Y↑↑(-1).DIFF(Y,T).Y↑↑(-1),DIFF(Y↑↑(-1),T),EQ);
                          <- 1>        <- 1>  dy    <- 1>
(d11) (m . y + a . x) . y      - x . y      . -- . y     =
                                             dt

       <- 1>             <- 1>        <- 1>
- x . y      . n . x . y      + x . y      . transpose(a) + m

            <- 1>
 + a . x . y
```

```
(c12) /* USE EQ2 FOR DY/DT IN EQ */

EQ:EQ,EQ2:

                                <- 1>          <- 1>
(d12) (m . y + a . x) . y       - x . y
                                                <- 1>
 . (n . x - transpose(a) . y) . y       =

       <- 1>                  <- 1>          <- 1>
- x . y        . n . x . y       + x . y     . transpose(a) + m
                <- 1>
 + a . x . y

(c13) /* SHOW THAT THEY ARE EQUAL */

RHS(EQ)-LHS(EQ),EXPAND;
(d13)                                0
```

Since the difference is 0, the two forms are equivalent.

ANALYTICAL SOLUTIONS TO SOME MATRIX RICCATI EQUATIONS

Ralph Wilcox
Hughes Aircraft Co. EDSG
El Segundo, CA 90245

and

Leo P. Harten
Paradigm Associates, Inc.
29 Putnam Ave. Suite 6
Cambridge, MA 02139
and
MIT
Cambridge, MA 02139

Abstract

Macsyma [1] was used to convert the inhomogeneous non-linear matrix Riccati equation:

diff(V , t) = A . V + V . AT + M - V . N . V

[A, M, and N are constant nxn matrices, V is a symmetric nxn matrix, AT = transpose(A)]

into a linear system of differential equations in 2 * n↑2 unknowns. The DESOLVE program [1] can solve such linear systems by Laplace transforms.

A target tracker in which the target is subject to stochastic forces was modeled by such an equation. The resulting analytical solutions for V when n=2 and n=3, and with V either initially 0 or singular, were obtained and verified. The Taylor-Laurent series as t-->0+ and the limiting behavior as t-->infinity were

obtained, and confirmed the plots of the solutions. See Reference [2] for full details.

ACKNOWLEDGMENTS

The authors wish to thank Larry Rubin, Carlton Nealy, and Richard Frey of Hughes Aircraft Co. EDSG, and Frank Shields and Stanley Rodak of Night Vision and Electro-Optics Laboratory for their encouragement and interest in the work on the matrix Riccati equations.

This work was supported, in part, by the Night Vision and Electro-Optics Laboratory through U.S. Army Research Office contract DAAG-29-76-D-0100, awarded to Hughes Aircraft Company on 26 June 1981.

REFERENCES

1. MACSYMA Reference Manual, Version 9, The MATHLAB Group, Laboratory for Computer Science, MIT (1977)

2. Wilcox, R. M. and Harten, L. P. MACSYMA-Generated Closed-Form Solutions to Some Matrix Riccati Equations, Journal of Applied Mathematics and Computation, Vol. 14, pp 149-166 (1984)

MACSYMA - AIDED LARGE DEFORMATION ANALYSIS OF A CYLINDRICAL SHELL UNDER PURE BENDING

Kenneth A. Bannister
Naval Surface Weapons Center
White Oak, Silver Spring, MD 20910

Abstract

This paper addresses the large deformation behavior of cylindrical shells under pure bending with simultaneously applied uniform pressure. This problem is important in the practical design of cylindrical shell structures to resist failure due to in-service bending loads combined with pressure, for example, submarine pressure hulls, aircraft fuselages, and industrial piping systems. A new methodology is described for dealing with such nonlinear shell analysis problems. MACSYMA is first applied alone and then is coupled with two appropriate minimization algorithms to solve for the local large deformation response of the cylinder. A previous potential energy-based analysis of the problem has been extended for the purpose, and the mathematical labor is greatly expedited with the aid of MACSYMA. It is shown that in cases where explicit algebraic solutions for, say, the moment-curvature relation of the shell are impractical to generate, only a potential energy expression and its first derivatives need to be constructed. From these expressions, nonlinear optimization algorithms can then be brought to bear to minimize directly the potential energy by methodically and efficiently adjusting the displacements (or other appropriate basic quantities). Having solved for the basic quantities, derived quantities, such as strains, stresses, and moments can then be computed.

1. INTRODUCTION

This paper is concerned with the development and demonstration of a new methodology to aid in the solution of nonlinear problems in mechanics. The particular application discussed here is a very specific case of nonlinear shell

response. The original motivation for this study was to achieve a better understanding of the mechanics of a submarine pressure hull undergoing "whipping," i.e., low frequency flexural vibrations, caused by a nearby underwater explosion. Pressure hulls are complicated structures, so instead of tackling the complete problem from the outset, two simplifications are introduced. First, a highly idealized representation of the actual pressure hull is adopted; secondly, dynamic effects are ignored. Because the purpose here is to demonstrate a new methodology for solving this kind of mechanics problem, it is felt that, although structural details and the effects of motion are very important, trying to account for them all at once would obscure rather than enhance the purpose. In any case, subsequent refinements can be easily carried out given a powerful analytical tool such as MACSYMA. The underlying methodology actually has a much broader range of application for both shell vibration problems and in structural mechanics at large. It is shown how the methodology can be used to extend a previous nonlinear shell analysis and then facilitate practical solutions, exact and approximate (numerical), of the resulting equations. The large amount of mathematical labor typically associated with obtaining nonlinear shell solutions is greatly reduced and the analyst has much greater freedom to modify or extend the analysis without the penalty of tedious, time-consuming algebraic manipulations.

2. BRIEF REVIEW OF WHIPPING

An earlier paper [1] gave a fairly detailed discussion of explosion bubble-induced whipping of surface ships and submarines. Basically, whipping is defined as the transient beam-like response of a ship to some form of strong hydrodynamic loading, in this case the accelerating fluid flow field surrounding a pulsating and migrating explosion gas products bubble. Whipping analyses can be performed with a simple lumped mass-elastic finite element structural model coupled to the fluid equations of motion. Typically, only the first few modes of vibration of the ship are needed to satisfactorily capture the whipping response, i.e., the heave and pitch rigid body modes and the first 2-3 distortion modes.

If the whipping motions achieve large amplitudes, then any attempt at predicting the local hull plating response (that is, shell response) must account for possible large out-of-plane displacements of the shell which cannot be handled within the context of elementary beam theory. Fortunately the motions are low frequency, at least at locations remote from where the fluid loads impinge, hence a quasistatic shell analysis is appropriate. In the regions of

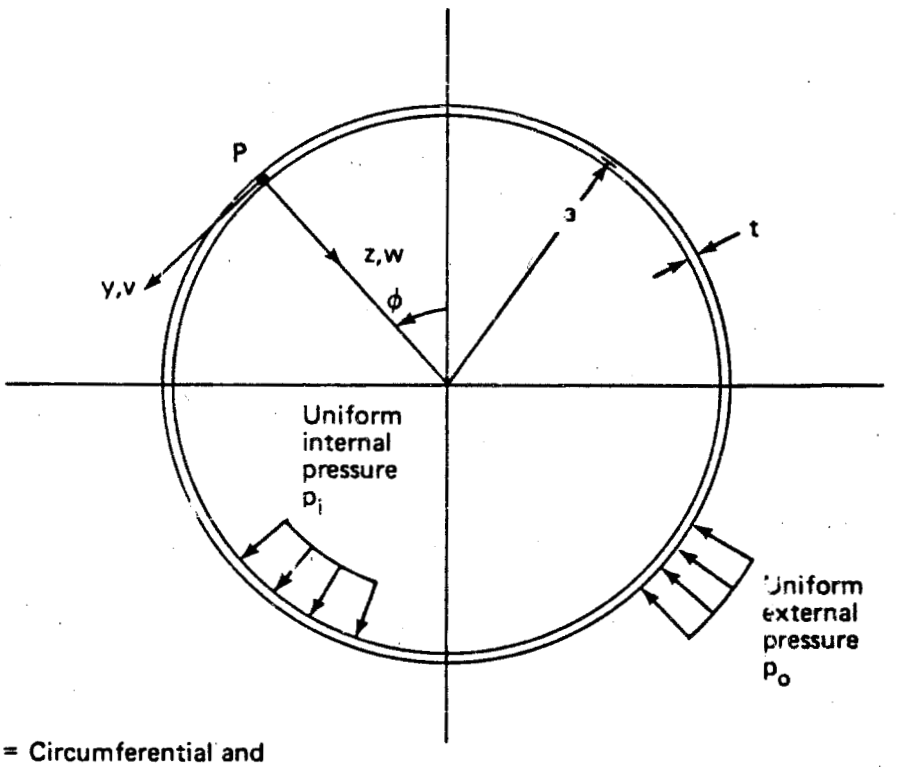
intense fluid loading, because of the high frequency shell motions, shell inertia terms become important and thus a more sophisticated transient analysis is required. In this paper we will focus on the problem of analyzing the quasistatic large deformation behavior of the overall pressure hull.

3. ENERGY ANALYSIS OF SHELL UNDER BENDING AND PRESSURIZATION

We consider the large deformation response of an elastic thin-walled circular cylindrical shell subjected to pure bending and either internal or external pressure. Figure 1 shows the cross section of the shell and defines pertinent geometric and pressure parameters used in the analysis. By "thin" we mean that $a/t>50$ and since most pressure hulls are of the order $a/t=100$ we can regard them as thin shell structures. Because the shell is thin, we say that the state of affairs in the shell wall can be adequately represented by conditions in the middle surface ($r=a$). For the sake of convenience, we further assume that moment and curvature do not vary over the shell length so that all cross sections deform in the same manner. A side view of the shell is shown in Figure 2. The shell's neutral axis is bent into an arc of circle with radius ρ due to the terminal moments. Figure 2 also defines d, distance from the neutral surface to a given point on the deformed shell middle surface. This parameter is used in computing stress and strain quantities in the shell wall. Linearly elastic material behavior is assumed, thus E (Young's modulus) and ν (Poisson's ratio) completely characterize the material response.

The thinness of the shell wall relative to other shell dimensions leads to large (when compared to the thickness t) displacements normal to the shell wall during bending, rendering the problem geometrically nonlinear. Therefore, any attempt to accurately determine the shell response must account for these geometric nonlinearities. There is a smooth transition from small to large displacements in this problem. Thus, nonlinear effects must be anticipated starting at fairly low load levels compared to the peak (maximum possible) moment that the shell can carry. Furthermore, pressure has a remarkable effect on the load-displacement response, as will be shown. Internal pressure tends to stiffen the shell in a way that increases the peak moment while external pressure weakens the shell's ability to withstand bending.

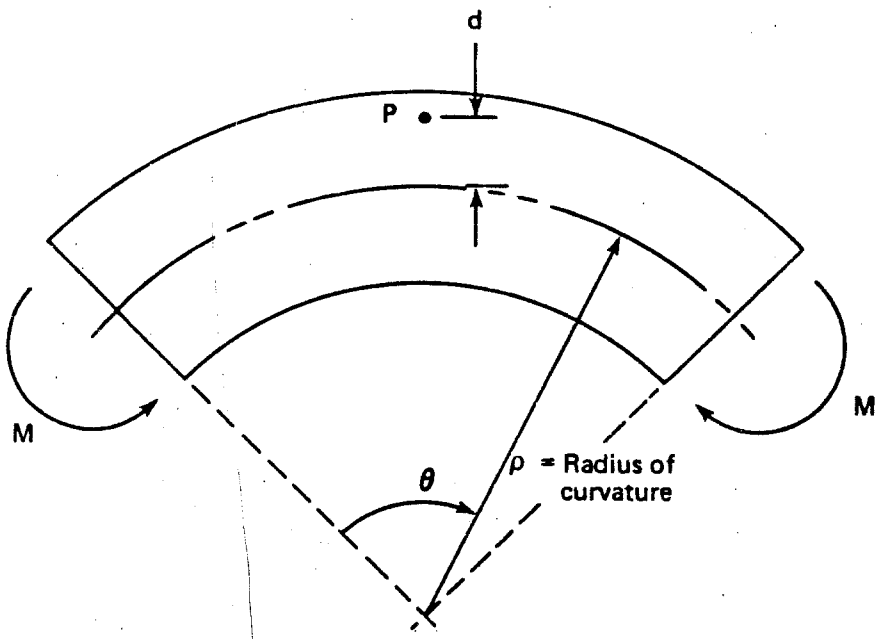
The problem will be analyzed by an energy minimization approach, facilitated by the use of MACSYMA. The total potential energy function is formulated for the loaded shell and is cast in terms of middle surface displacements at a cross section of the shell. The shell is assumed to be



y,z = Circumferential and normal coordinates

v, w = y and z-direction displacement components at the point P of the shell middle surface

Figure 1. Definition of Cross Section Parameters and Coordinates for Circular Cylindrical Shell



d = Distance from neutral axis to point P of shell middle surface

Figure 2. Cylindrical Shell in Pure Bending

infinitely long and moreover that no variation of conditions occurs along its length. Truncated Fourier series terms with a priori unknown coefficients are included in the displacement functions to account for nonlinear effects. The goal of the analysis is then to determine the coefficients of these additional terms which minimize the potential energy. The strategy followed is to generate an expression for the potential energy involving the unknown coefficients and the curvature parameter defined by $\varepsilon = a/\rho$. ε will also be unknown for given values of moment and pressure. Then, for each choice of bending moment and pressure, the minimizing set of unknown displacement coefficients and curvature parameter is found by two methods:

(1) Use of MACSYMA to obtain explicit solutions for the coefficients in terms of ε and subsequently $M(\varepsilon)$; this we will call the "exact" solution; and

(2) Generation of the nonlinear moment-curvature relation by direct application to the potential energy of two different computer algorithms designed to minimize nonlinear multivariate unconstrained functions; that is, the potential energy will be minimized directly with the aid of optimization algorithms.

The purpose for the second method is to show that in the case where an "exact" moment-curvature relation cannot easily be obtained by the first method, then accurate numerical results can nevertheless be produced through direct energy minimization. This notion has broader implications for situations where a large number of unknowns, say 200, are involved in a particular nonlinear mechanics problem.

As in most nonlinear shell analyses, the sheer amount of mathematical manipulation can be enormous. In the present case, MACSYMA is used to substantially reduce the mathematical labor. This is, to the author's knowledge, the first time such a tool has been applied in a comprehensive manner to a nonlinear shell analysis. MACSYMA is excellently suited to the task as it can handle all of the necessary mathematical operations involved such as functional evaluation, trigonometric expansion and reduction, differentiation, integration, and equation solving. Its convenient similarities to other standard programming languages, along with its file manipulation features make MACSYMA extremely useful for the problem at hand.

Using a powerful mathematical tool such as MACSYMA, it is now possible to carry out analyses of nonlinear structural mechanics problems and avoid many ad hoc simplifications authors in the past found necessary to make the effort tractable. In the present work, a MACSYMA code has been constructed which reproduces step-by-step the mathematical

analysis of the shell bending problem beginning with displacement function generation, derivation of stress and strain quantities, proceeding with construction of the potential energy function, and finally ending with solution for the unknown Fourier coefficients and generation of a moment-curvature parameter relation. It is possible, within this code, to include or exclude quite readily certain nonlinear displacement terms that previous authors felt compelled to drop due to mathematical complexity. Thus, the usual simplifications invoked in shell work suggested by the phrases "neglect quantities of small magnitude" and "neglect cross product and squared quantities" can be freely adopted, ignored, or modified as the analyst wishes. The extra calculational burden is carried by the computer, not the analyst. As an example of the savings in time and labor that can be realized with such a symbolic language tool, the author applied FORMAC-73, an older language, to a two-term Fourier series displacement function analysis based on a nonlinear shell theory in order to obtain the integrand of the strain energy (bending and stretching of the shell middle surface) expression. Manually, this effort required two weeks--with a considerable amount of time consumed in checking for and correcting errors. With FORMAC-73, the same exercise required less than one day with the algebraic operations done correctly throughout. For three, four, or more Fourier terms or, let us say, alternative displacement functions of greater complexity, it was clear that hand-computations would become very costly in time and the likelihood for errors very great. In addition, the coding could be stored if desired and modified and re-run for different cases and the results retained on files for listing or for later applications.

One of the earliest studies of the cylindrical shell in pure bending (without pressurization) was the classic paper by Brazier [2]. Brazier simplified the analysis greatly by the accurate assumption that the shell middle surface does not stretch, i.e., is inextensible. He also pointed out that the problem becomes fundamentally nonlinear due to the thinness of the shell wall relative to other dimensions of the shell. Thus the linear relationship between moment and curvature derived from the usual St. Venant's theory for bending of beams of solid cross section is invalidated since local wall deformations are large and thus strain cannot be a linear function of original position. Figure 3 shows the disparity between the linear (St. Venant) and large deformation (nonlinear) predictions of moment vs. curvature. Also, in Figure 4 we see the ovalization mode that the shell cross section takes on; this mode was assumed by Brazier in his analysis.

In the small displacement range, superposition holds; hence the moments and pressure can be applied in any order. Then, following the approach of Brazier [2] and Wood [4], the

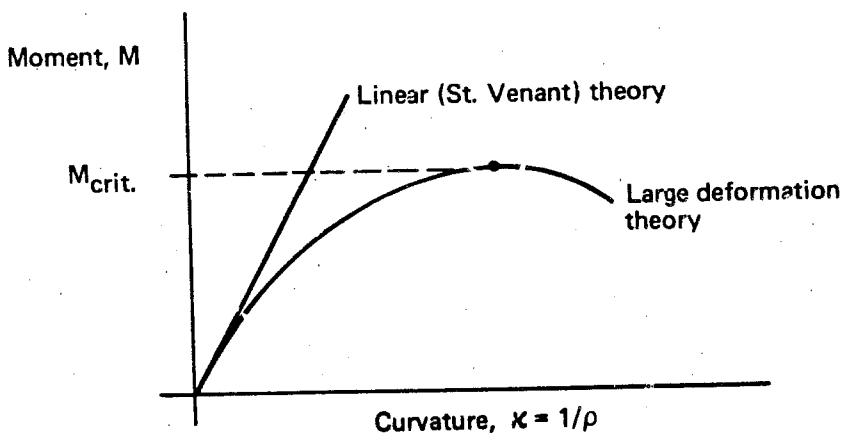


FIGURE 3. MOMENT-CURVATURE BEHAVIOR AS PREDICTED BY LINEAR (ST. VENANT) THEORY AND LARGE DEFORMATION (BRAZIER) THEORY

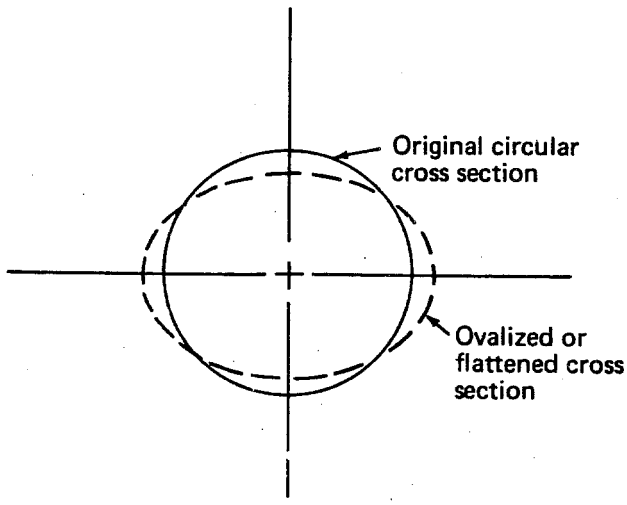


FIGURE 4. BRAZIER'S OVALIZATION MODE FOR THE CIRCULAR CYLINDRICAL SHELL IN PURE BENDING

possibility of nonlinear deformations is allowed by introducing truncated Fourier series terms in the displacement components v and w (defined in Figure 1). These terms have undetermined coefficients which are to be found by application of the Theorem of Minimum Potential Energy.

The potential energy V is given by

$$V = U + W \ . \tag{1}$$

U is the strain energy stored in the shell due to bending and W represents the work of the pressure and bending loads. In order to account for the bending strain energy, the change of curvature at any point on the shell must be computed. Brush and Almroth [3] show this to be given by (after accounting for a sign change due to use of a different convention for w)

$$\chi_{\phi\phi} = \frac{1}{a^2}\left(\frac{dv}{d\phi} + \frac{d^2w}{d\phi^2}\right). \tag{2}$$

Next, the strain energy U is given by

$$U = \frac{a}{2}\int_0^{2\pi}\Big[D\,\chi_{\phi\phi}^{\,2} + t(\sigma_{xx}\varepsilon_{xx} + \sigma_{\phi\phi}\varepsilon_{\phi\phi} + \bar{\sigma}_{\phi\phi}\bar{\varepsilon}_{\phi\phi} + \bar{\sigma}_{xx}\bar{\varepsilon}_{xx} + 2\,\bar{\sigma}_{xx}\varepsilon_{xx} + 2\,\bar{\sigma}_{\phi\phi}\varepsilon_{\phi\phi})\Big]d\phi \ . \tag{3}$$

An important geometric parameter is d, the distance from the shell neutral surface to a given fiber in the shell middle surface (as shown in Figure 2). The usual assumption of thin shell theory is applied here, namely, that stresses and strains do not vary through the wall thickness and that the normal stress through the thickness vanishes. Therefore, the middle surface stress-strain state adequately represents the response of the shell. The parameter d is given by

$$d = (a - w)\cos\phi - v\sin\phi \ . \tag{3a}$$

The circumferential and axial stresses and strains in the small displacement range are given by Wood [4] for bending and pressure as follows:

for bending

$$\varepsilon_{xx} = \frac{d}{\rho} ,$$

$$\sigma_{xx} = \frac{Ed}{\rho} ,$$

$$\varepsilon_{\phi\phi} = - \frac{\nu d}{\rho} , \tag{4}$$

$$\sigma_{\phi\phi} = 0;$$

and for pressure

$$\bar{\varepsilon}_{xx} = \frac{\alpha}{2} (1-2\nu),$$

$$\bar{\sigma}_{xx} = \frac{E\alpha}{2} , \tag{5}$$

$$\bar{\varepsilon}_{\phi\phi} = \frac{\alpha}{2} (2-\nu),$$

$$\bar{\sigma}_{\phi\phi} = E\alpha.$$

Although the deformations may become large, strains remain small; this is typical in thin shell structures. A large strain analysis would require accounting for plastic material response-this problem is not considered here.

In these equations, the pressure enters through the parameter α given by

$$\alpha = \frac{Pa}{Et} . \tag{6}$$

Sokolnikoff [5] reports the linear w and v displacement components for bending and pressure are given by:

for w

$$w_0 = \frac{\nu\varepsilon a}{2}\cos\phi \ , \tag{7}$$

$$\bar{w} = -\frac{(2-\nu)}{2}\, a\alpha \ ;$$

and for v

$$\bar{v} = 0 \text{ (symmetry)} \ , \tag{8}$$

$$v_0 = -\frac{\nu\varepsilon a}{2}\sin\phi \ .$$

We have introduced the curvature parameter ε in Equations (7) and (8). It is defined by

$$\varepsilon = \frac{a}{\rho} \ . \tag{9}$$

The total displacements v and w are then the sum of bending, pressurization, and additional terms v_1, w_1 needed to account for large deformations of the shell. Thus we have

$$v = v_0 + v_1 \ , \tag{10}$$

$$w = \bar{w} + w_0 + w_1 \ .$$

Following Brazier [2], we assume that the additional displacements v_1, w_1 are inextensional; thus we have

$$w_1 = \frac{dv_1}{d\phi} . \tag{11}$$

According to Wood [4], v_1 and w_1 can be expressed as the infinite series given by

$$v_1 = \sum_{n=2}^{\infty} A_n \sin n\phi , \tag{12}$$

and, following Equation (11), we must have

$$w_1 = \frac{dv_1}{d\phi} = \sum_{n=2}^{\infty} nA_n \cos n\phi . \tag{13}$$

Note that v and w do not vary with position along the shell axis; that is, all cross sections must deform in the same manner. Also note that the choice for v_1 satisfies displacement continuity and symmetry conditions on v. Hence, the total displacements become

$$\begin{aligned} v &= -\frac{\nu\epsilon a}{2} \sin\phi + \sum_{n=2}^{\infty} A_n \sin n\phi \\ w &= -\left(\frac{2-\nu}{2}\right) a\alpha + \frac{\nu\epsilon a}{2} \cos\phi \\ &\quad + \sum_{n=2}^{\infty} nA_n \cos n\phi . \end{aligned} \tag{14}$$

The work done by the pressure and applied moments is given by Wood [4] as

$$W = -\frac{P\pi}{2}(\nu^2 a^2 \varepsilon^2 - \sum_{n=2}^{\infty}(n^4-n^2)A_n^2) - \frac{M}{a}\varepsilon. \tag{15}$$

We see then that the potential energy V can ultimately be written as a nonlinear function of the coefficients A_n and the current curvature parameter ε in the general form

$$V = V(A_2, A_3, \ldots, \varepsilon). \tag{16}$$

The strategy followed from this point is to seek values for the coefficients $A_2, A_3, \ldots$ which minimize the total potential energy of the loaded shell.

Wood [4] proceeded with the solution by neglecting certain terms in V involving v_1^2, w_1^2, $v_1 w_1$, then expanding V and applying the theorem of Minimum Potential Energy: V is stationary when

$$\frac{\partial V}{\partial A_2} = \frac{\partial V}{\partial A_3} = \ldots = \frac{\partial V}{\partial A_n} = 0. \tag{17}$$

Wood [4], for example, has carried out the simplification (or linearization) of V just mentioned and has found that the coefficients A_2 and A_3 become in that case

$$A_2 = \frac{a\varepsilon^2(1-\nu^2)\left[1 + \frac{(1-\nu)}{2}\right]\alpha}{2(t/a)^2 + 8(1-\nu^2)\alpha}, \tag{18}$$

$$A_3 = \frac{-a\varepsilon^3(1-\nu^2)}{48(t/a)^2 + 72(1-\nu^2)\alpha}.$$

The remaining coefficients vanish, i.e.,

$$A_4 = A_5 = \ldots = A_n = 0. \tag{19}$$

However, these terms do not vanish if the linearization is not carried out, as is shown in [6]. Unfortunately, the amount of algebraic manipulation associated with including squared and cross product terms grows enormously as more terms in the Fourier expansions are taken. This difficulty is greatly lessened by use of MACSYMA.

It is useful to derive a general form for the moment-curvature relation of the shell. We have seen earlier that the coefficients A_n can be expressed as functions of the curvature parameter; hence, the potential energy V may be written

$$V = G(A_2(\varepsilon), A_3(\varepsilon), \ldots, \varepsilon). \tag{20}$$

For V to be stationary with respect to ε, we must have

$$\frac{\partial V}{\partial \varepsilon} = \frac{\partial G}{\partial A_2}\frac{dA_2}{d\varepsilon} + \ldots + \frac{dG}{d\varepsilon} = 0 . \tag{21}$$

But since

$$\frac{\partial V}{\partial A_2} = \frac{\partial V}{\partial A_3} = \ldots \frac{\partial V}{\partial A_n} = 0, \tag{22}$$

i.e.,

$$\frac{\partial G}{\partial A_2} = \frac{\partial G}{\partial A_3} = \ldots = \frac{\partial G}{\partial A_n} = 0 , \tag{23}$$

Equation (21) reduces to

$$\frac{dG}{d\varepsilon} = 0 . \tag{24}$$

This last equation may be rewritten quite easily in the following form

$$\frac{dG}{d\varepsilon} = f(A_2, A_3, \ldots, \varepsilon) - \frac{M}{a} = 0 \tag{25}$$

or, solving for M, we have

$$M = a\, f(A_2, A_3, \ldots, A_n, \varepsilon). \tag{26}$$

Wood has found in his linearized analysis [4] that Equation (26) becomes

$$M = \frac{EI_{xx}}{2a} \left\{ \varepsilon \left[2 + \frac{\nu^2 (t/a)^2}{6(1-\nu^2)} + 2 \frac{(2-\nu)}{2}^2 a^2 + 4 \frac{(2-\nu)}{2} \alpha - 2\nu^2 \alpha \right] \right.$$

$$+ \varepsilon^3 \left[\nu^2 - \frac{3(1-\nu^2)\ (1 + (\frac{2-\nu}{2})\alpha)^2}{(t/a)^2 + 4(1-\nu^2)\alpha} \right] \tag{27}$$

$$\left. + \varepsilon^5 \left[\frac{-\nu^2 (1-\nu^2)}{24(t/a)^2 + 36(1-\nu^2)\alpha} \right] \right\}$$

Equation (27) may be written, for convenience, in the simpler form

$$M = c_1\varepsilon + c_2\varepsilon^3 + c_3\varepsilon^5. \tag{28}$$

Note that peak moment values are readily found, corresponding to roots of the quartic given by

$$5c_3\varepsilon^4 + 3c_2\varepsilon^2 + c_1 = 0. \tag{29}$$

Since Equation (29) has four possible roots, the root desired must be real and positive. This value of ε can then be substituted in Equation (28) to get the corresponding peak moment. Other quantities of interest, such as the deformed cross section profile and stress and strain dependence on angle ϕ can also be generated at this peak moment value.

This analysis has been extended with the aid of MACSYMA to include the quadratic terms in v_1 and w_1 which Wood neglected in his energy function. It is shown in [6] also that Wood's entire analysis can be reproduced by use of MACSYMA. By including the quadratic terms previously neglected, we show that these terms have significant effects on the moment- cuvature behavior of the shell. Explicit solutions, in algebraic form, are given for two, three, and four term trigonometric expansions of v_1 and w_1 (see Equation (4)). It turns out that the coefficients A_4 and A_5 calculated in this case do not vanish as Wood found in his linearized analysis. As an alternative to straight-forward solution for the moment-curvature relationship through the use of MACSYMA, two different gradient method-based optimization algorithms are applied directly to the potential energy functional. These algorithms were designed for the minimization of nonlinear unconstrained multivariate functions. They require only explicit expressions for the function to be minimized (in this case the potential energy) and its first derivatives with respect to the independent variables $(A_2, \ldots, A_5, \varepsilon)$. We show that both algorithms give excellent agreement with the "exact" moment-curvature results calculated through MACSYMA- generated expressions. Thus in situations where it may not be practical to solve directly for a moment-curvature relation, useful and accurate numerical results can be obtained by direct minimization so long as expressions for the potential energy functional and its derivatives can be obtained.

4. RESULTS

By use of the MACSYMA symbolic manipulation system, quadratic and cross-product displacement terms can easily be retained in the energy expression (Equation (3)) and the necessary algebraic operations carried out. To do this by hand would prove to be a formidable task, even for a few terms in the Fourier expansions. Such routine mathematical operations as trigonometric reduction, expansion of products, differentiation, and integration can be done with MACSYMA. A further useful application of this tool, is illustrated in Appendix A, where the Euler equation for Brazier's analysis is solved.

A useful way to characterize the shell response is to display its moment-curvature behavior similar to the generic curve given in Figure 3. Two computational procedures are available: Direct generation of an explicit equation relating moment M and the curvature parameter ε; or direct minimization of the total potential energy expression by numerical minimization (i.e., optimization) methods. MACSYMA-generated solutions for the coefficients $A_1, \ldots, A_n$ have been obtained for the fully nonlinear energy expression. With these coefficients, we can then compare plots of $M(\varepsilon)$ for Wood's linearized analysis with results from the present nonlinear analysis. This will show clearly whether neglecting the quadratic terms (as Wood did) affects the moment-curvature behavior. Figures 5 and 6 show respectively comparisons of $M(\varepsilon)$ for Wood's analysis with the present analytical results for moderate internal and external pressure (300 psi). The shell is made of steel and has dimensions typical of a submarine pressure hull. The curves for the fully nonlinear analyses (two, three, and four-term Fourier expansions) are very nearly the same; hence, all three curves appear to coalesce to a single line (upper curves in Figures 5 and 6). Computations for the nonlinear cases were done with the MACSYMA code listed in Appendix B. Additional comparisons like those in Figures 5 and 6 can be found in [6] for other pressure values and shell geometries.

As an alternative to explicit generation of the moment-curvature equation, the energy expression for the system can be minimized directly through use of a numerical optimization technique. In this case, of course, we must forego seeing the explicit algebraic solution form and must instead settle for "approximate" answers, albeit with a controllable accuracy. The purpose here is to demonstrate that in those cases of nonlinear shell analysis where even the assistance of a tool such as MACSYMA is not completely effective in obtaining "exact" solutions, extremely accurate numerical results can be easily gotten if the energy functional can at least be generated. This situation could arise in the present work, for example, if a large number of Fourier expansion terms were desired or if the underlying shell theory formulation was made more complex. We will, without loss of generality, illustrate this approach for only the simplest of the Fourier expansions (i.e., two terms); more terms could easily be accommodated but with a greater expenditure of computer time in performing the MACSYMA manipulations.

Two gradient method-based algorithms have been used to minimize the energy functional. Both are designed to minimize quite general nonlinear unconstrained multivariate functions. At present V, the total potential energy functional, is the objective function for which a global minimum is sought for given moment and pressure. The minimum is found by

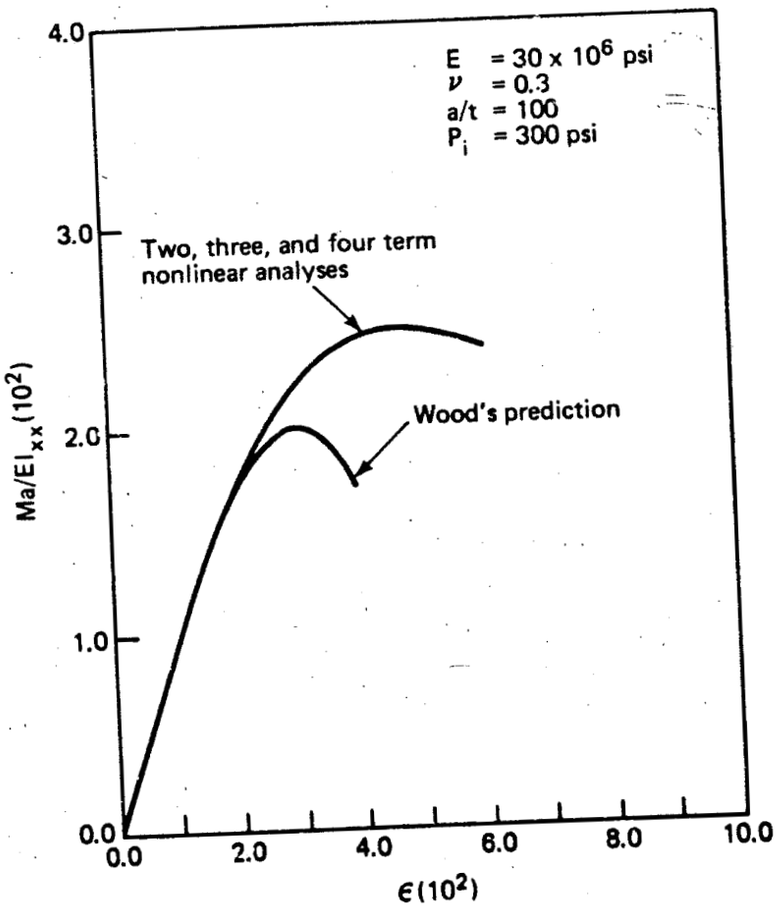


FIGURE 5. COMPARISON OF NONDIMENSIONAL MOMENT-CURVATURE PREDICTIONS (THIN SHELL, MODERATE INTERNAL PRESSURE)

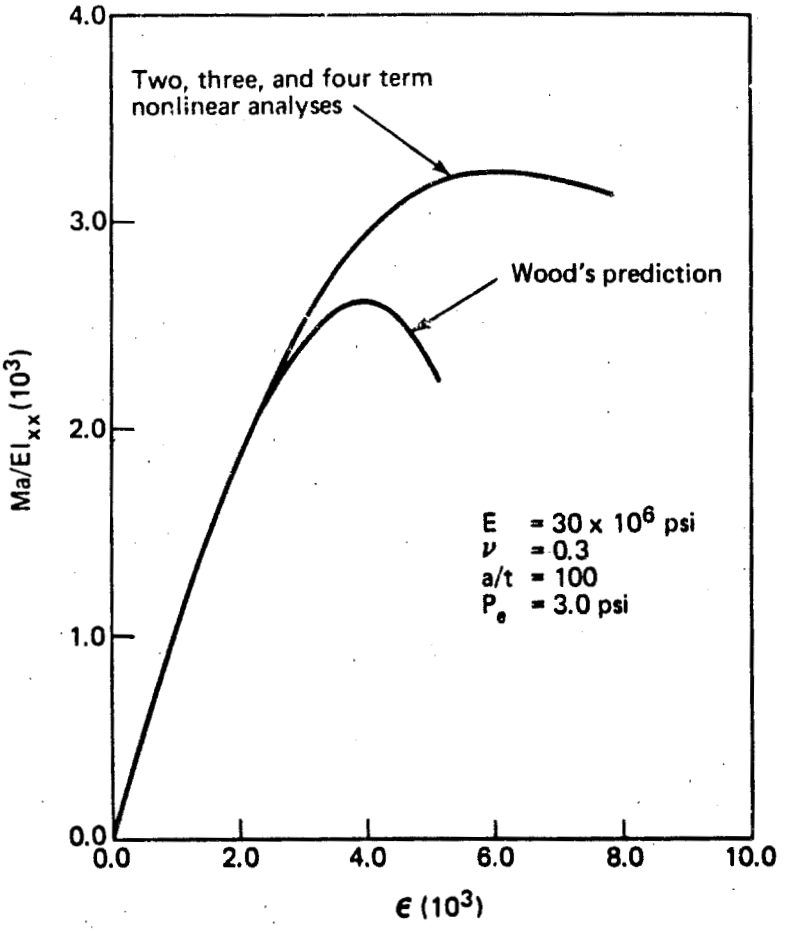


FIGURE 6. COMPARISON OF NONDIMENSIONAL MOMENT-CURVATURE PREDICTIONS (THIN SHELL, MODERATE EXTERNAL PRESSURE)

systematically adjusting ε and the coefficients A_n until V is minimized. The user must supply explicit expressions for V and its first partial derivatives with respect to $\varepsilon, A_1, \ldots, A_n$. Also, reasonable starting values must be supplied for these parameters from which the algorithms begin with iterative search for the minimum of V. Both algorithms are conveniently contained within the FORTRAN subroutine CONMIN developed by Shanno and Phua [7].

The first algorithm incorporates a variable metric technique, i.e., initially it resembles the Steepest Descent Method in performance while resembling the Newton-Raphson method as the minimum is approached [8]. The second algorithm is based on the conjugate gradient method. In this case, the procedure is to seek the minimum by successive linear searches along mutually conjugate directions. Of course a major task then becomes the generation of sets of such directions. The particular implementation used here is one due to Shanno [9]. As is shown in [6], both algorithms were first "put through their paces" by testing them against suitably difficult functions; both were found to be satisfactory. FORTRAN routines were then written, built around MACSYMA-generated FORTRAN coding for the energy expression, and coupled with the CONMIN subroutine to compute moment-curvature results similar to the previous "exact solution" procedure.

Moment-curvature plots, similar to Figures 5 and 6, were computed for the same shell geometry but for an internal pressure of 3000 psi by use of both algorithms. Results for the conjugate gradient and variable metric methods are shown in Figures 7 and 8, respectively. The upper line in both figures was generated from the exact solution for $M(\varepsilon)$ discussed earlier while the triangles represent the discrete points where the algorithms supplied approximate values. Both algorithms gave excellent agreement with the exact solution, typically to three or four digits' accuracy on moment values. Further calculations of this kind are given in [6]. It is expected that the excellent performance obtained for the simple two-term expansion carries over to the higher expansions as well. In general, in both the test problems and in the moment-curvature calculations, the conjugate gradient method required more function evaluations (of V and its partial derivatives), that is, more computer time, than the variable metric method to converge to a solution.

5. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE WORK

A new methodology has been developed for the solution of geometrically nonlinear shell problems and is illustrated by application to the specific case of a large deformation

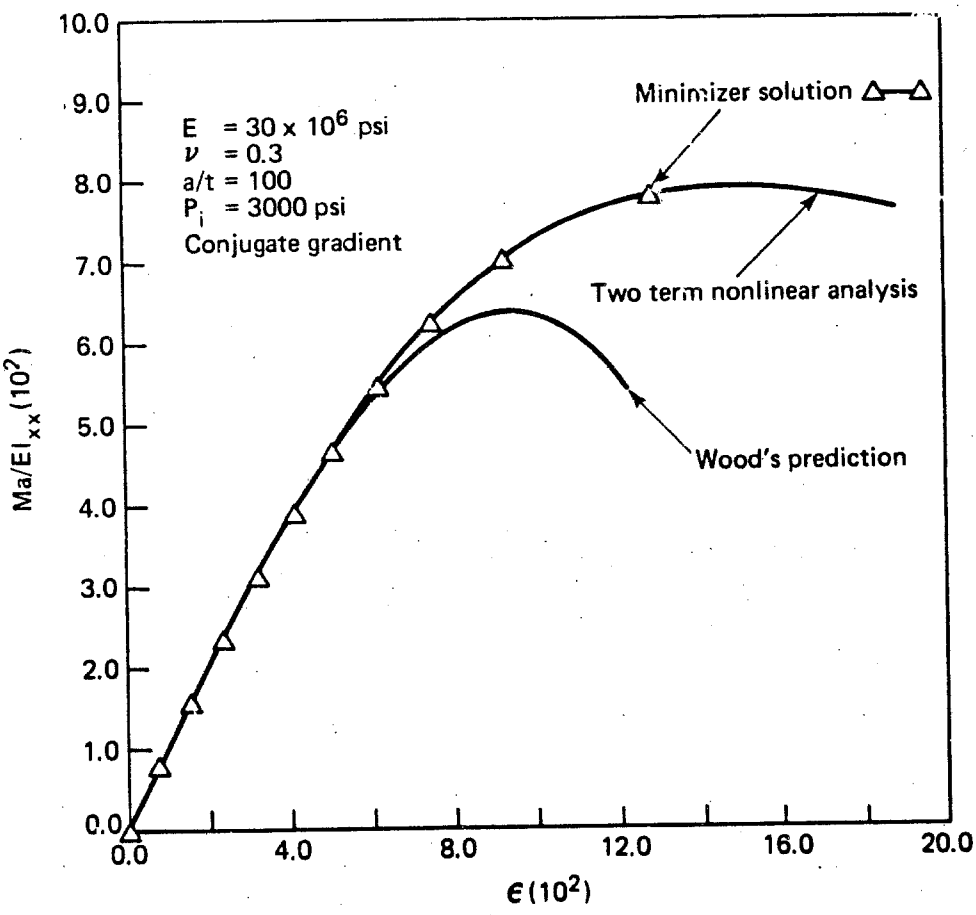


FIGURE 7. CONJUGATE GRADIENT, EXACT, AND WOOD'S RESULTS FOR 3000 psi INTERNAL PRESSURE

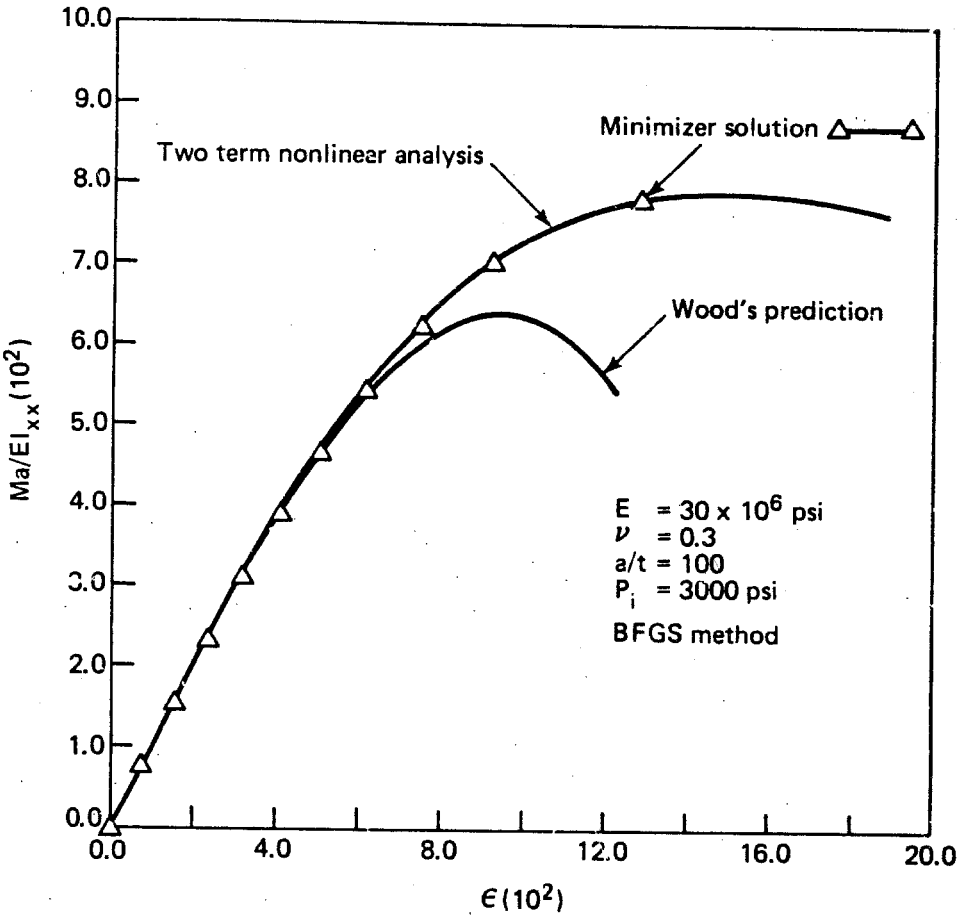


FIGURE 8. BFGS, EXACT, AND WOOD'S RESULTS FOR 3000 psi INTERNAL PRESSURE

analysis of a thin elastic circular cylindrical shell subjected to pure bending and pressurization. We have shown that a symbolic manipulation language, such as MACSYMA, is a powerful analytical tool. The massive algebraic work attendant with nonlinear shell work is greatly expedited with the aid of such tools. It is demonstrated that two optimization algorithms (one conjugate gradient, one variable metric) originally designed for minimization of nonlinear unconstrained multivariate functions can be used to compute extremely accurate results where MACSYMA-generated explicit solutions may be impractical. The function minimized in this case is the total potential energy of the shell and external loads.

Several assumptions and limitations are inherent in the shell analysis considered here. The main assumptions are that: (1) The possibility for bifurcation (buckling) from the nonlinear deformation states leading up to the limit moment is omitted; (2) Initial imperfections, or deviations from the true circular form, are not considered; (3) The shell is infinitely long and thus all cross sections deform the same way (also the influence of end boundary conditions are neglected); and (4) The additional displacements v_1, w_1 are assumed to be inextensible. These effects can all be considered by reformulating the analysis from the start. It was felt that the complexity associated with analyzing these effects would unnecessarily obscure the purpose here which is to illustrate clearly a new methodology. Of these assumptions, perhaps the most important to relax in extending the work, is the assumption concerning bifurcation. Other investigators have pointed out that bifurcation into a pattern of axial wrinkles on the compression side of the shell occurs just before the limit moment. In real shells, this wrinkling behavior is greatly affected by the presence of imperfections. Also, material nonlinearity (plasticity) may become important as the wrinkles increase in amplitude.

There are several avenues for further research. Clearly, the entire subject of shell buckling anaylsis can benefit from the introduction of tools such as MACSYMA--this applies to finite element-type work as well. An interesting alternative to starting with the displacement-based energy approach of Brazier and Wood is to begin with strains as discussed by Reissner. This would allow the use of series of polynominal functions such as Legendre or Chebyshev polynominals. In fact, Chebyshev polynominals appear very attractive for this application as they are known to possess certain optimal properties.

The analysis can also be extended to include stretching of the shell middle surface, influence of the axial dimension (i.e., three dimensional analysis), the presence of stiffeners, or plasticity effects. Again, the strain energy functional

may turn out to be very nonlinear, including non-smooth behavior if plasticity rules are invoked; but practical solutions may be computed by treating the problem as one in constrained minimization, or if need be, an equivalent succession of unconstrained problems.

6. REFERENCES

1. Bannister, K. A., "Whipping Analysis Techniques for Ships and Submarines," The Shock and Vibration Bulletin, Bulletin 50, Part 3 (September, 1980): 83-98.

2. Brazier, L. G., "On the Flexure of Thin Cylindrical Shells and Other Sections," Proceedings of the Royal Society of London, Series A CVXI (1927): 104-14.

3. Brush, D. O., and Almroth, B. O., Buckling of Bars, Plates, and Shells, New York: McGraw-Hill, 1975.

4. Wood, J. D. "The Flexure of a Uniformly Pressurized, Circular, Cylindrical Shell," Journal of Applied Mechanics 25 (1958): 453-58.

5. Sokolnikoff, I. S., Mathematical Theory of Elasticity, New York: McGraw-Hill Book Company, 1956.

6. Bannister, K. A. "Large Deformation Analysis of a Cylindrical Shell Under Pure Bending and Pressurization," Ph.D Dissertation, The Pennsylvania State University, 1983.

7. Shanno, D. F., and Phua, K. H. "Remark on Algorithm 500: Minimization of Unconstrained Multivariate Functions," Association for Computing Machinery Transactions on Mathematical Software 6, No. 4 (December 1980): 618-22.

8. Shanno, D. F., and Phua, K. H., "Matrix Conditioning and Nonlinear Optimization," Mathematical Programming 14 (1978): 149-60.

9. Shanno, D. F., "Conjugate Gradient Methods with Inexact Searches," Mathematics of Operations Research 3 (1978): 244-56.

10. Mathlab Group, MACSYMA Reference Manual Version 10, Massachusetts Institute of Technology, Laboratory for Computer Science, Cambridge, Massachusetts, 1982.

APPENDIX A

APPLICATION OF MACSYMA TO ROOT-FINDING

Brazier's analysis [2] leads to an Euler equation for the circumferential displacement component v in the following form

$$\frac{d^6v}{d\phi^6} + 2\frac{d^4v}{d\phi^4} + \frac{d^2v}{d\phi^2} = -\frac{18a^5}{\rho^2t^2}\sin 2\phi \ . \qquad \text{(A-1)}$$

The solution to Equation (A-1) can easily be obtained by the theory of ordinary differential equations, but it is instructive to employ MACSYMA in obtaining the homogeneous solution. Although MACSYMA can solve ODE's by the Laplace Transform method, it turns out that sixth-order differential equations exceed the present capability. However, by inspection we can write down the characteristic equation for (A-1) as

$$R^6 + 2R^4 + R^2 = 0 \qquad \text{(A-2)}$$

which obviously has the repeated roots

$$R = 0$$
$$R = \pm\sqrt{-1}. \qquad \text{(A-3)}$$

MACSYMA can be used to obtain the same roots. The input commands and replies to these commands are shown next,

```
This is a MACSYMA 303

FIX303 1 DSK MACSYM being loaded
Loading done

(C1) EQU:R**6+2*R**4+R**2;

(D1)      R^6 + 2R^4 + R^2

(C2) SOLVE(EQU,R);

SOLVE FASL DSK MACSYM being loaded
Loading done
(D2)      [R = - %I, R = %I, R = 0]
```

After a preliminary message, the righthand side of the characteristic equation is input to MACSYMA on the line (C1) and is stored in the variable named EQU. After this point, any reference to EQU is equivalent to referencing the polynominal expression in R. In line (D1), MACSYMA merely "plays back" or displays the contents of EQU in standard algebraic format. On line (C2), MACSYMA is instructed to solve for the roots of the R-expression; it is assumed EQU = 0 in this case. After printing a brief message concerning loading of files to perform the root search, the roots are computed and then are displayed on line (D2). Note that MACSYMA uses the convention

$$\%I = \sqrt{-1} = i. \tag{A-4}$$

Having found the roots, the homogeneous solution can be written

$$v(\phi) = (B_1 + B_2\phi) + (B_3+B_4\phi)\cos\phi + (B_5 + B_6\phi)\sin\phi. \tag{A-5}$$

APPENDIX B

LISTING OF MACSYMA CODE USED TO GENERATE NONLINEAR SOLUTION FOR THREE-TERM FOURIER EXPANSION

```
/* N IS THE NUMBER OF THE HIGHEST FOURIER TERM TO TAKE */
N:5$
/* DEFINE RULES WHICH WILL ELIMINATE SINE AND COSINE TERMS
   FROM TRIGREDUCED EXPANDED TRIGONOMETRIC EXPRESSIONS.  THIS
   SAVES AN INTEGRATION STEP SINCE THESE TERMS WILL VANISH
   UNDER INTEGRATION FROM ZERO TO 2*PI */
MATCHDECLARE(ZZ,TRUE)$
DEFRULE(R1,SIN(ZZ),0)$
DEFRULE(R2,COS(ZZ),0)$
/* THE FUNCTION EBOTH WILL TAKE CARE OF APPLYING RULES R1 AND
   R2 TO A GIVEN EXPRESSION */
EBOTH(XX,YY):=BLOCK([RESULT],
RESULT:TRIGREDUCE(EXPAND(XX),YY),
RESULT:APPLY1(RESULT,R1,R2),
RETURN(RESULT))$
/* AA ARRAY WILL CONTAIN THE TRUNCATED FOURIER SERIES
   COEFFICIENTS */
ARRAY(AA,10)$
/* V, W ARE THE TOTAL DISPLACEMENTS */
V:V0+V1$
W:W0+W1+WBAR$
/* DEFINE THE DISTANCE D AND ITS SQUARE */
D:(A-W)*COS(PHI)-V*SIN(PHI)$
DD:D*D$
/* DEFINE THE SMALL DISPLACEMENT TERMS CORRESPONDING TO LINEAR
   ELASTICITY */
V0:VV0*SIN(PHI)$
W0:WW0*COS(PHI)$
WBAR:WWBAR$
/* COMPUTE AND EVALUATE THE TOTAL DISPLACEMENTS */
V1:0$
W1:0$
V1:V1+SUM(AA[I]*SIN(I*PHI),I,2,N)$
W1:W1+SUM(I*AA[I]*COS(I*PHI),I,2,N)$
V:EV(V)$
W:EV(W)$
/* EVALUATE D AND ITS SQUARE */
D:EV(D)$
DD:EBOTH(EV(DD),PHI)$
/* COMPUTE THE LATERAL CURVATURE EXPRESSION */
CHI:(DIFF(W,PHI,2)+DIFF(V,PHI))/(A*A)$
```

```
/* COMPUTE THE VARIOUS STRESS AND STRAIN EXPRESSIONS AND THE
   TOTAL POTENTIAL ENERGY TERMS */
U1I:DC*EBOTH(CHI*CHI,PHI)*A$
ST:E*D*EPS/A$
SP:0$
STB:E*AL/2$
STB:E*AL$
ET:D*EPS/A$
EP:-NU*D*EPS/A$
ETB:AL*(1-2*NU)/2$
EPB:AL*(2-NU)/2$
U2I:DD*E*EPS**2/A**2*A*T$
U3I:0$
U4I:SPB*EPB*A*T$
U5I:STB*ETB*A*T$
U6I:EBOTH(STB*ET*A*T,PHI)$
U7I:A*T*EBOTH(SPB*EP,PHI)$
/* UI IS THE INTEGRAND FOR THE SHELL BENDING AND STRETCHING
   ENERGY.  U IS ITS INTEGRAL. */
UI:(U1I+U2I+U3I+U4I+U5I)/2+U6I+U7I$
U:2*%PI*U1$
/* TT IS THE WORK OF THE PRESSURE FORCES.  WW IS THE WORK OF
   THE EXTERNAL MOMENTS.  VV IS THE TOTAL POTENTIAL ENERGY */
TT:((AL*E*T/A)*%PI/2)*((A*NU*EPS)**2-SUM((I**4-I**2)*AA[I]**2,
I,2,N))$
WW:M*EPS/A$
VV:U-TT-WW$
/* SET UP A SET OF EQUATIONS TO SOLVE FOR THE UNKNOWN FOURIER
   SERIES COEFFICIENTS AA[2] THROUGH AA[N] */
ARRAY(DVV,1,10)$
FOR I:2 THRU N DO
(DVV[I]:DIFF(VV,AA[I])=0)$
SOLVE([DVV[2],DVV[3],DVV[4],DVV[5]],
[AA[2],AA[3],AA[4],AA[5]]),GLOBALSOLVE:TRUE;
/* EVALUATE AND DISPLAY THE TOTAL DISPLACEMENTS V, W */
V:EV(V);
W:EV(W);
/* DISPLAY FORTRAN VERSIONS OF AA[3] THRU AA[5] AND V AND W */
FORTRAN(AA[2]);
FORTRAN(AA[3]);
FORTRAN(AA[4]);
FORTRAN(AA[5]);
FORTRAN(V);
FORTRAN(W);
```

HOPF BIFURCATION IN MULTI-DEGREE-OF-FREEDOM SYSTEMS USING MACSYMA

P. Hollis and D. L. Taylor
Sibley School of Mechanical Engineering
Cornell University
Ithaca, NY 14853

Abstract

This paper deals with the use of MACSYMA to perform Hopf Bifurcation analysis of some multi-degree-of-freedom systems. Although it is possible to perform the same analysis numerically, MACSYMA allows the user to obtain a much better appreciation of the role of several parameters. The systems examined include the very familiar Van Der Pol equations, Langford's equations, the Lorenz equations and a follower force example. The system for which the program was initially written proved too difficult to examine.

1. INTRODUCTION

Although it is possible to perform numerical analysis of nonlinear systems, sometimes it is more desirable to perform some type of approximate analytical technique to examine the same systems. Such techniques can yield much more insight into global nonlinear behavior than can numerical calculations. Hopf

bifurcation analysis from a center manifold approach is just one such technique. It is mathematically equivalent to the more well known methods such as averaging, harmonic balance, describing functions etc. Hassard, Kazarinoff and Wan (1981) have provided a powerful numerical anlysis program to predict limit cycle behavior of nonlinear systems.

We take the same approach to develop a MACSYMA program to perform essentially the same analysis. It is greatly reduced in its capabilities for handling large systems, but for small tractable problems it does yield far more information about system behavior for several variables than does the equivalent numerical program. The technique allows for some simplification of the problem through the application of formulae rather than applying a general technique to the nonlinear problem. The program itself can be easily extended to calculate higher order terms which will allow a better approximation or, for some cases, information about degenerate points where the method cannot make any predictions.

2. HOPF BIFURCATION

Consider a system of differential equations

$$\dot{x} = f(x,\nu) \tag{1}$$

where $x \in R^n$, $\nu \in R$. Let $x_*(\nu)$ be an isolated stationary point i.e. $f(x_*(\nu),\nu) = 0$. Then $x_*(\nu)$ is linearly stable if all the eigenvalues $\lambda_i(\nu)$, $i = 1,2,\ldots,n$ of the Jacobian matrix

$$J(\nu) = \left.\frac{\partial f}{\partial x}\right|_{x_*(\nu)} \tag{2}$$

have negative real parts and is linearly unstable if some eigenvalues have positive real parts.

If periodic solutions arise out of $x_*(\nu)$ as the parameter ν is varied throgh some critical value ν_c, a Hopf bifurcation has taken place. It is associated with a change in stability type of the equilibrium point. In particular, it occurs when a conjugate pair of eigenvalues change from having negative or positive real parts to having positive or negative real parts respectively. At bifurcation, the rea. part is zero. For systems of interest, it is also required that the remainder of the eigenvalues have negative real parts for ν in the neighbourhood of ν_c. The conditions then are

(i) $\bar{\lambda}_1(\nu) = \bar{\lambda}_2(\nu) = \alpha(\nu) + i\,\omega(\nu)$

(ii) $\alpha(\nu_c) = 0, \qquad \omega(\nu_c) > 0$

(iii) $\alpha'(\nu_c) \neq 0$

(iv) $\mathrm{Re}\ \lambda_j(\nu_c) < 0, \qquad j = 3,\ldots,n$

Then Hopf bifurcation theory asserts that a parameter μ_2 exists and is interpreted as follows: if

$\mu_2 > 0$ periodic solutions exist for $\nu > \nu_c$

$\mu_2 < 0$ periodic solutions exist for $\nu < \nu_c$.

The periodic orbit is approximated by

$$x = x_*(\nu_c) \pm \left(\frac{\nu - \nu_c}{\mu_2}\right)^{1/2} \mathrm{Re}\,[e^{i\omega(\nu)} v_1] \qquad (3)$$

where v_1 is the eigenvector of the Jacobian at $\nu = \nu_c$ corresponding to $\lambda_1(\nu_c)$ and $\omega(\nu)$ is the frequency of the orbit.

Stability of the limit cycle can be determined from a second quantity β_2 defined as $\beta_2 = -2\alpha'(\nu_c)\mu_2$. If

$\beta_2 < 0$ the solution is orbitally asymptotically stable

$\beta_2 > 0$ the solution is orbitally asymptotically unstable.

That real systems exhibit such behavior is well known. For example, high speed instability of automobiles can be interpreted as Hopf bifurcation. Also the wind induced vibration of Venetian blinds can be considered Hopf bifurcation to limit cycles. For further information on Hopf bifurcation, the reader is referred to the books by Carr (1981) and Marsden and McCracken (1976).

3. PROCEDURE

The procedure to be outlined here is basically the same as that described in Hassard, Kazarinoff and Wan (1981) and it follows closely the numerical algorithm they have developed.

Given the equations in the form (1) we

a. Find $x_*(\nu)$ such that $f(x_*, \nu) = 0$.

b. Evaluate the Jacobian

$$J(\nu) = \left.\frac{\partial f}{\partial x}\right|_{x_*(\nu)}.$$

c. Evaluate the eigenvalues and eigenvectors of $J(\nu)$ and find the value of ν where there is a pair of purely imaginary eigenvalues (and all the rest have negative real parts). Let $v_1(\nu)$ be the eigenvector corresponding to the bifurcating pair of eigenvalues $\lambda_{1,2}(\nu) = \alpha(\nu) \pm i\omega(\nu)$.

d. Form the transformation matrix $P(\nu)$ such that

$$P(\nu)^{-1}J(\nu)P(\nu) = \begin{bmatrix} \alpha & \omega & 0 \\ \omega & \alpha & 0 \\ 0 & 0 & D \end{bmatrix}. \qquad (4)$$

$P(\nu)$ is of the form $[\text{Re } v_1, -\text{Im } v_1, v_3, \ldots, v_n]$ where $v_3, \ldots, v_n$ are vectors spanning the eigenspace of $\lambda_3, \ldots, \lambda_n$.

e. Change variables according to

$$x = x_* + Py,$$

$$\mu = \nu - \nu_c$$

and change equations according to

$$\dot{y} = P^{-1}f(x_* + Py, \mu) = F(y, \mu).$$

f. Perform another change of variables

$$z = y_1 + iy_2$$

and

$$w = [y_3, \ldots, y_n]^T.$$

Also form the new equations

$$G = F_1 + iF_2 - \lambda z$$

and

$$H_i = F_{i+2} - Dw$$

for $i = 1, \ldots, n-2$.

g. Evaluate the quantity

$$g_{ij} = G^0_{ij} = \frac{\partial^2}{\partial z^i \partial z^j}\Big|_{G(0,0,0,\mu)}$$

where $i + j = 2$ and the quantity

$$G^1_{ij} = \frac{\partial}{\partial z^i \partial z^j}\left[\frac{\partial G}{\partial w_1}, \ldots, \frac{\partial G}{\partial w_n}\right]\Big|_{(0,0,0,\mu)}$$

where $i + j = 1$.

h. Form

$$w_{ij} = [(\lambda i + \bar{\lambda} j)I - D]^{-1} \left.\frac{\partial^2 H}{\partial z^i \partial \bar{z}^j}\right|_{(0,0,0,\mu)}$$

where $(i,j) = (2,0), (1,1)$ and

$$g_{21} = \frac{\partial^3 G}{\partial z^2 \partial \bar{z}} + 2G^1_{10}w_{11} + G^1_{01}w_{20}.$$

i. Form

$$c_1(\ ) = \frac{g_{20}g_{11}(2\lambda+\bar{\lambda})}{2\lambda\bar{\lambda}} + \frac{g_{11}\overline{g_{11}}}{\lambda} + \frac{g_{02}\overline{g_{02}}}{2(2\lambda - \bar{\lambda})} + \frac{g_{21}}{2}$$

$$\mu_2 = -\mathrm{Re}\ \frac{c_1(0)}{\alpha(0)}$$

$$\beta_2 = 2\mathrm{Re}\ c_1(0)$$

$$\alpha'(0) = \mathrm{Re}\ \lambda'_1(\nu_c)$$

$$\omega'(0) = \mathrm{Im}\ \lambda'_1(\nu_c)$$

$$\tau_2 = -\frac{(\mathrm{Im}\ c_1(0) + \mu_2 \omega'(0))}{\omega(0)}$$

$$T = \frac{2\pi}{\omega(0)}(1 + \tau_2 \varepsilon^2)$$

$$\varepsilon^2 = \frac{\nu - \nu_c}{\mu_2}.$$

The main problems involved in doing these transformations are finding the equilibrium point for general nonlinear systems and finding the eigenvalues and eigenvectors for $n > 2$. Also the

size of the equations generated tends to be large, especially if the initial equations are already cumbersome. Out of these problems came some interesting side results: programmed breaks to allow the user to experiment with various solution methods or numeric substitutions; saving parts of large expressions in variably named files:

```
NAME : [CONCAT(D,'F,I,D,'X,J),MAC],
WHAT : CONCAT(TEMP_,I,J),
DERIV : DIFF(F[I],X[J]),                                    (5)
WHAT :: DERIV,
FILE  : 'SAVE(NAME,WHAT),
EV(FILE,NOUNS),
```

For the practicing engineer, there is the function for evaluating the linearized spring and damping coefficients about any nominated point.

4. EXAMPLES

Presented here are the results from several problems of increasing complexity.

4.1 Van der Pol Equations (Hassard, Kazarinoff and Wan (1981))

$$\dot{x}_1 = \mu x_1 - x_2 - x_1^3 \qquad (6)$$
$$\dot{x}_2 = x_1$$

yields

$$\text{equilibrium point} = [X_1 = 0,\ X_2 = 0]$$

bifurcation variable and value at bifurcation [NU, 0]

$$\text{jacobian} = \begin{bmatrix} NU & -1 \\ 1 & 0 \end{bmatrix}$$

$$\text{real part of eigenvector} = \left[1, \frac{NU}{2}\right]$$

$$\text{imaginary part of eigenvector} = \left[0, \frac{\%I\ SQRT(NU - 2)\ SQRT(NU + 2)}{2}\right]$$

$$p = \begin{bmatrix} 1 & 0 \\ NU & \%I\ SQRT(NU - 2)\ SQRT(NU + 2) \\ \frac{NU}{2} & -\frac{\%I\ SQRT(NU - 2)\ SQRT(NU + 2)}{2} \end{bmatrix}$$

$$F_1 = -\%I\ (-Y_2\ NU^2 + \%I\ Y_1\ SQRT(NU - 2)\ SQRT(NU + 2)\ (NU - 2\ Y_1^2) + 4\ Y_2)/(2\ SQRT(NU - 2)\ SQRT(NU + 2))$$

$$F_2 = -\%I\ (\%I\ Y_2\ SQRT(NU - 2)\ NU\ SQRT(NU + 2) + Y_1\ NU^2 - 2\ Y_1^3\ NU - 4\ Y_1)/(2\ SQRT(NU - 2)\ SQRT(NU + 2))$$

$$G = (\%I\ Y_2\ SQRT(NU - 2)\ NU\ SQRT(NU + 2) + Y_1\ NU^2 - 2\ Y_1^3\ NU - 4\ Y_1)/(2\ SQRT(NU - 2)\ SQRT(NU + 2))$$

$$- \%I\ (- Y_2\ NU^2 + \%I\ Y_1\ SQRT(NU - 2)\ SQRT(NU + 2)$$

$$(NU - 2\ Y_1^2) + 4\ Y_2)/(2\ SQRT(NU - 2)\ SQRT(NU + 2))$$

$$- (\%I\ Y_2 + Y_1)\ \left(\frac{NU^2 - 4}{2\ SQRT(NU - 2)\ SQRT(NU + 2)} + \frac{NU}{2}\right)$$

$$LAMBDA = \frac{NU^2 - 4}{2\ SQRT(NU - 2)\ SQRT(NU + 2)} + \frac{NU}{2}$$

$$C1MU = -\ \frac{3\ SQRT(NU - 2)\ NU\ SQRT(NU + 2) + 3\ NU^2 - 12}{2\ (4\ NU^2 - 16)}$$

$$C10 = -\ \frac{3}{8}$$

$$AMU2 = \frac{3}{4}$$

$$TAU2 = 0$$

$$BETA2 = -\ \frac{3}{4}$$

$$DALPHA0 = \frac{1}{2}$$

$$DOMEGA0 = 0$$

4.2 Langford's Equations (Hassard, Kazarinoff and Wan (1981))

$$\begin{aligned}\dot{x}_1 &= (\nu-1)x_1 - x_2 + x_1x_3\\ \dot{x}_2 &= x_1 + (\nu-1)x_2 + x_2x_3\\ \dot{x}_3 &= \nu x_3 - (x_1^2 + x_2^2 + x_3^2)\end{aligned} \tag{7}$$

yields

equilibrium point = $[X_1 = 0,\ X_2 = 0,\ X_3 = NU]$

bifurcation variable and value at bifurcation $[NU,\ \frac{1}{2}]$

$$\text{jacobian} = \begin{bmatrix} 2\,NU - 1 & -1 & 0 \\ 1 & 2\,NU - 1 & 0 \\ 0 & 0 & -NU \end{bmatrix}$$

real part of eigenvector = [1, 0, 0]

imaginary part of eigenvector = [0, - 1, 0]

$$p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_1 = 2\,Y_1\,NU + Y_1\,Y_3 - Y_2 - Y_1$$

$$F_2 = 2\,Y_2\,NU + Y_2\,Y_3 - Y_2 + Y_1$$

$$F_3 = -\,Y_3\,NU - Y_3^2 - Y_2^2 - Y_1^2$$

$$G = \%I\ (2\ Y_2\ (NU + \tfrac{1}{2}) + Y_2\ Y_3 - Y_2 + Y_1)$$

$$- (\%I\ Y_2 + Y_1)\ (2\ (NU + \tfrac{1}{2}) + \%I - 1) + 2\ Y_1\ (NU + \tfrac{1}{2})$$

$$+ Y_1\ Y_3 - Y_2 - Y_1$$

$$LAMBDA = 2\ (NU + \tfrac{1}{2}) + \%I - 1$$

$$H = MATRIX([- Y_3\ (NU + \tfrac{1}{2}) - Y_3\ (- NU - \tfrac{1}{2}) - Y_3^2 - Y_2^2$$

$$- Y_1^2])$$

$$D = \begin{bmatrix} & 1 \\ - NU - & \frac{}{2} \end{bmatrix}$$

$$C1MU = - \frac{2}{10\ NU + 1}$$

$$C10 = - 2$$

$$AMU2 = 1$$

$$TAU2 = 0$$

$$BETA2 = - 4$$

$$DALPHA0 = 2$$

$$DOMEGA0 = 0$$

4.3 Lorenz Equations (Hassard, Kazarinoff and Wan (1981))

$$\begin{aligned} \dot{x}_1 &= -\sigma x_1 + \sigma x_2 \\ \dot{x}_2 &= -x_1 x_3 + \gamma x_1 - x_2 \\ \dot{x}_3 &= x_1 x_2 - b x_3 \end{aligned} \tag{8}$$

yields

$$\text{equilibrium point} = [X_1 = \text{SQRT}(B\,(G-1)),\ X_2 = \text{SQRT}(B\,(G-1)),\ X_3 = G - 1]$$

bifurcation variable and value at bifurcation

$$[G,\ \frac{S^2 + (B+3)\,S}{S - B - 1}]$$

$$\text{jacobian} = \text{Col 1} = \begin{bmatrix} -S \\ 1 \\ \text{SQRT}(B\,(G-1)) \end{bmatrix}$$

$$\text{Col 2} = \begin{bmatrix} S \\ -1 \\ \text{SQRT}(B\,(G-1)) \end{bmatrix}$$

$$\text{Col 3} = \begin{bmatrix} 0 \\ -\text{SQRT}(B\,(G-1)) \\ -B \end{bmatrix}$$

```
_solve_main_mat;
```

$$\text{Col 1} = \begin{bmatrix} -S \\ 1 \\ \dfrac{\text{SQRT}(B)\ \text{SQRT}(S+1)\ \text{SQRT}(S+B+1)}{\text{SQRT}(S-B-1)} \end{bmatrix}$$

$$\text{Col 2} = \begin{bmatrix} S \\ -1 \\ \dfrac{\text{SQRT}(B)\ \text{SQRT}(S+1)\ \text{SQRT}(S+B+1)}{\text{SQRT}(S-B-1)} \end{bmatrix}$$

$$\text{Col 3} = \begin{bmatrix} 0 \\ -\dfrac{\text{SQRT}(B)\ \text{SQRT}(S+1)\ \text{SQRT}(S+B+1)}{\text{SQRT}(S-B-1)} \\ -B \end{bmatrix}$$

```
_x_eq;
```

$$[X_1 = \frac{\text{SQRT}(B)\ \text{SQRT}(S+1)\ \text{SQRT}(S+B+1)}{\text{SQRT}(S-B-1)},$$

$$X_2 = \frac{\text{SQRT}(B)\ \text{SQRT}(S+1)\ \text{SQRT}(S+B+1)}{\text{SQRT}(S-B-1)},$$

$$X_3 = \frac{S^2 + (B+2)\,S + B + 1}{S - B - 1}]$$

```
real part of eigenvector =
```

$$[1,\ 1,\ \frac{2\ \text{SQRT}(B)\ \text{SQRT}(S+1)}{\text{SQRT}(S-B-1)\ \text{SQRT}(S+B+1)}]$$

imaginary part of eigenvector =

$$[0, -\sqrt{2}\,i\,\sqrt{B}\,\sqrt{-S+B+1}\,\sqrt{S+1}\,/(\sqrt{S}\,(S-B-1)), -\frac{\sqrt{2}\,i\,\sqrt{-S}\,(S+1)}{S\,\sqrt{S+B+1}}]$$

$$p = \mathrm{MATRIX}([1, 0, 1], [1, \frac{\sqrt{2}\,i\,\sqrt{B}\,\sqrt{-S+B+1}\,\sqrt{S+1}}{\sqrt{S}\,(S-B-1)}, -\frac{B+1}{S}], [\frac{2\sqrt{B}\,\sqrt{S+1}}{\sqrt{S-B-1}\,\sqrt{S+B+1}}, \frac{\sqrt{2}\,i\,\sqrt{-S}\,(S+1)}{S\,\sqrt{S+B+1}}, -\frac{\sqrt{B}\,\sqrt{S-B-1}\,\sqrt{S+B+1}}{S\,\sqrt{S+1}}])$$

$$F_1 = -(2\,(S+1)\,\sqrt{S-B-1}\,(2\sqrt{2}\,Y_1\,B\,S^2\,\sqrt{S+B+1} + 2\sqrt{2}\,Y_1\,B\,S\,\sqrt{S+B+1} - Y_2\,Y_3\,S^{5/2} - Y_1\,Y_2\,S^{5/2} - Y_2\,Y_3\,B^2\,\sqrt{S} - Y_1\,Y_2\,B^2\,\sqrt{S} + Y_2\,Y_3\,\sqrt{S} + Y_1\,Y_2\,\sqrt{S}) - 2\,Y_2\,\sqrt{B}\,\sqrt{S}\,\sqrt{S+1}\,\sqrt{S+B+1}\,(S^3 - B\,S^2 + S^2 + 3\,B^2\,S - S - B^3 + B^2 + B - 1)$$

$$- \text{SQRT}(2)\ (Y_3 + Y_1)\ \text{SQRT}(B)\ S\ \text{SQRT}(S+1)\ (S - B - 1)$$

$$(Y_3\ S - 3\ Y_1\ S + Y_3\ B - Y_1\ B + Y_3 - 3\ Y_1))$$

$$/(\text{SQRT}(2)\ \text{SQRT}(S - B - 1)\ \text{SQRT}(S + B + 1)$$

$$(S^3 - 3\ B\ S^2 + S^2 - B^2\ S - 4\ B\ S - S + B^3 + B^2 - B$$

$$- 1))$$

The results here are incomplete and have been evaluated at the bifurcation point. The other equations are extremely long and complex, and it is not possible to find the general equations in this case. The main problem lies in forming the transformed equations although finding just the eigenvectors and eigenvalues yields large expressions for the general case.

4.4 Follower Force Equations (Sethna and Schapiro (1977))

$$\dot{x}_1 = x_3$$

$$\dot{x}_2 = x_4$$

$$\dot{x}_3 = \frac{(\sin(x_2-x_1)(\cos(x_2-x_1)x_3^2+x_4^2) - (x_1+bx_3)(\cos(x_2-x_1)+2) + (x_2+bx_4)(\cos(x_2-x_1)+1) - t\sin(x_2-x_1))}{(2 + \sin^2(x_2-x_1))} \tag{9}$$

$$\dot{x}_4 = \frac{(-\sin(x_2-x_1)(\cos(x_2-x_1)x_4^2+3x_3^2) + (x_1+bx_3)(2\cos(x_2-x_1)+3) - (x_2+bx_4)(\cos(x_2-x_1)+3) + t\sin(x_2-x_1)\cos(x_2-x_1))}{(2 + \sin^2(x_2-x_1))}$$

yields as far as MACSYMA is able to proceed with $b = \frac{2}{\text{sqrt}(7)}$

equilibrium point = $[X_1 = 0, X_2 = 0, X_3 = 0, X_4 = 0]$

bifurcation variable and value at bifurcation $[S, \frac{7}{4}]$

$$\text{jacobian} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{S-3}{2} & \frac{2-S}{2} & -\frac{3}{\text{SQRT}(7)} & \frac{2}{\text{SQRT}(7)} \\ \frac{5-S}{2} & \frac{S-4}{2} & \frac{5}{\text{SQRT}(7)} & -\frac{4}{\text{SQRT}(7)} \end{bmatrix}$$

5. CONCLUSIONS

Although MACSYMA can give useful results, it appears that the class of problems it can handle is very limited. It may be better to resort to numerical analysis of large problems, or for some cases, approximate the nonlinear problem by a simpler nonlinear one which still captures the basic nonlinear behavior. The fundamental problem lies not with the technique but rather with the amount of computer time and space available for dealing with such large expressions. These expressions are in general too big to be of any use unless they can be simplified greatly.

6. ACKNOWLEDGEMENTS

This research was supported in part by the United States Office of Naval Reseach under contract N0014-80-C-0618.

7. REFERENCES

1. Carr, J., Applications of Centre Manifold Theory, Applied Mathematical Sciences 35, Springer-Verlag, 1981.

2. Hassard, B.D., Kazarinoff, N.D., and Wan, Y-H., Theory and Applications of Hopf Bifurcation, London Mathematical Society Lecture Notes 41, Cambridge University Press, 1981.

3. Marsden, J.E., and McCracken, M.F., The Hopf Bifurcation and its Applications, Applied Mathematical Sciences 19, Springer-Verlag, 1976.

4. Sethna, P.R., and Schapiro, S.M., Nonlinear Behavior of Flutter Unstabe Dynamical Systems with Gyroscopic and Circulatoy Forces, Journal of Applied Mechanics, vol 44, Dec. 1977, pp 755-762.

A Tutorial On Particular Uses Of Macsyma

R. Drew Drinkard Jr.
Naval Underwater Systems Center
New London Laboratory
New London, Connecticut 06320

Abstract

The question is often asked by new users of macsyma: what can one use macsyma for? At least six major uses of macsyma can be idertified. They are

* simplification of large computational expressions
* mathematical reference to recall standard mathematical identities or formulas
* verification of the correctness of long, tedious and error prone mathematical manipulations performed by hand
* avoiding unnecessary numerical approximations by computer
* solution of mathematical problems symbolically
* symbolic manipulation and numerical method interaction

It is the purpose of this tutorial to provide specific examples of each of these six major uses of macsyma. Furthermore, a couple of new user turnoffs are pointed out, which perhaps have disillusioned some new users from exploring macsyma capabilities further.

COMPUTER ALGEBRA APPLIED TO KALMAN FILTERING

M. L SUAREZ
AGORA Systems, Inc.

Abstract

The Advent of Computer algebra brings with it new capabilities in the area of Kalman filtering. It is now possible to input symbolic versions of the filtering matrices and vectors and follow the effects of multiple updates on the state and/or the state covariance and to do so *symbolically* thus gaining greater insight into the effects of measurement noise, process noise, correlations, etc. The task of examining these by long-hand derivation is often prohibitive. This approach also allows for quick redefinition of the filter. A program, written in MACSYMA, is described which incorporates the equations of optimal linear estimation as well as smoothing. Both symbolic and numerical results are presented.

Research in Algebraic Manipulation at the University of California, Berkeley

Richard J. Fateman
John Foderaro
Gregg Foster
Rick McGeer
Neil Soiffer
Clifton J. Williamson

University of California
Berkeley, California

ABSTRACT

We describe recent work at UC Berkeley in the design and implementation of components of an algebraic manipulation system. Our intent has been to provide a sound framework for the next generation of workstation-based scientific software environments, and also explore the necessary technology for the development of mathematical expertise in a symbol processing system. We indicate the broad outline of this research project and its status.

1. Introduction and History

Over the last decade at UC Berkeley, we have been working on a number of applications of Macsyma and other algebraic manipulation systems. We have generated a number of specialized algebraic manipulation systems starting from the PDP-10 Macsyma system. For example, John Favaro's Symbolic Fortran execution system [], and a number of applications, were done via ARPAnet at MIT. In 1978, we began work on Franz Lisp, a host language sufficiently compatible with PDP-10 Maclisp to allow Macsyma code to run on the DEC VAX. This was motivated primarily by the acquisition of a VAX at UC Berkeley, and the need to use larger address spaces than available on the PDP-10. The kernel of Franz is written mostly in the portable language 'C'. The bulk of it is written in Lisp. Since 1979 we have used the PDP-10 Macsyma system rarely: principally for benchmarks and debugging. The 'Vaxima' variant of Macsyma, distributed under various mechanisms with the begrudging cooperation of MIT, has become fairly widespread, and is now under limited distribution by Symbolics, Inc. Although the current versions of the VAX computer are not as fast as the PDP-10 (KL), its address space is much larger than that of the PDP-10 simplifying programming matters considerably; furthermore, the hardware price is much lower.

In 1983, we once again ported Macsyma, this time to a workstation environment based on the Motorola 68000. This system has been demonstrated on computers produced by at least three different manufacturers. (Sun Microsystems, Pixel, and Masscomp). As this paper is being written, neither MIT nor Symbolics has any plans for distribution, or for allowing us to distribute this system. Since the US Department of Energy has funded most of this work, we hope to see a resolution which will make Macsyma more easily available.

2. Why a New System?

As is the case for many large software systems, Macsyma suffers from inertia: its size and lack of extensibility in certain crucial places. A paper in progress [Fateman, 84] describes the unforeseen consequences of Macsyma's design. In summary, though, we concluded that Macsyma could not serve as a testbed for many of the new ideas for the construction of an advanced symbolic algebra system. Given the changing computer environments, greater awareness of software engineering techniques, new mathematical abstraction languages, new devices for graphics, and the excitement of 'knowledge bases' it became clear that a new approach to the problem of constructing a computerized 'mathematical assistant' was warranted. It seemed that UC Berkeley would be the most likely location for an attempt to study the issues, and then design and build such a system. Interest by the System Development Foundation of Palo Alto, CA lead to the production of a discussion paper on open problems in in the area [Fateman, 1983] and then to some funding for startup of a major project. SDF support was vital in allowing us to look at a wide range of directions as alternatives to our own views.

Chosing a name for such a project is a problem: fortunately, this paper is being written on a word processing program so that late-breaking acronyms can be incorporated: the current choice is SCARAB (Symbolic/Scientific Computation and Representation/Reasoning at Berkeley).* The following sections outline topics of investigation at UC Berkeley. In some cases we have identified a need, but not made progress to date; in others, we have made substantial progress.

3. Newspeak

John Foderaro's PhD thesis work [Foderaro, 1983] on the design of a language for the construction of an algebraic manipulation system, provided the starting point for our recent efforts in experimentation in implementation strategies.

The direction taken with Foderaro's language, Newspeak is not entirely new, but grew out of the perceived problems in the use of Lisp or Lisp-level constructs

* We have been bombarded with suggestions including SNARK, CAL, Xi, Newmath, and ONION; (Onion's Not It's Original Name). I think SCARAB has some nice logo possibilities, though some people dislike beetles.)

implemented in 'C' as the standard language level for system implementation. By basing programs on manipulation of trees, linked lists, hash tables and similar data structures, the mathematical structure of the data is often obscured in Macsyma, SMP, Reduce, Maple, and many other systems. The Scratchpad '84/ Newspad system has taken a similar approach to ours. Approaches which have influenced the design of Newspeak include Barton's Andante, the IBM Research Modlisp [Jenks 1979], and various work by Hearn, Loos, and others.

The next few sections provide some indication of relevant features of Newspeak.

3.1. Language design and refinement

No computer language design is finished until the language is obsolete. We assumed that Newspeak would benefit from a rapid turn-around in implementation, and that substantial feedback from early users would insure that the language was actually suitable for the uses proposed. Pounding on the design and the implementation has been an important contributing factor in the refinement of Newspeak. Contributors have included David Barton, Clifton Williamson, Neil Soiffer, Scott Morrison, Andrew Lazarus, Carolyn Smith, Rick McGeer. The person responding to the pressure (threats?) and redirecting it has been John Foderaro.

The language is merely the raw material for construction of the mathematical components of the data and algorithm hierarchy which make up the "Berkeley Algebra and Analysis Development" described under 'BAAD' in the next major section.

3.2. Newspeak Philosophy

The pholosphy of the Newspeak language is to add strong typing in such a way that it enhances the user's productivity rather than degrades it. The major advantages of Newspeak's typing are 1) detection of simple typing bugs prior to execution; 2) its potential for compilation to efficient machine code; and 3) the compile-time resolution of generic function calls (thus making generic function calls in Newspeak as inexpensive as normal function calls in most other languages). The relationships between types in an algebra system are far more complex than can be described in the typical typed languages in common use (Pascal, Ada, C). As a consequence, to use of these languages for a symbolic algebra system one must first 1) disable the type system (simulating Lisp, in effect), or 2) use the inadequate type mechanism provided. Secondly, one must then build a type system on top of this first stage. The result is often a runtime type-checked system, and cannot take advantage of the type-checking inherent in the base language. Newspeak's type description facilities are sufficiently powerful that the relation between algebraic types can be expressed in Newspeak itself.

3.3. Language Implementation and Portability

Newspeak runs entirely interchangeably on the two UNIX implementations we use for our development. That is, it runs on Motorola 68000 and VAX architectures without alteration. This flexibility is available because the underlying system is based on Franz Lisp, which runs on these machines (and others). Newspeak code is compiled into byte codes which are then interpreted by an interpreter written in Lisp. This approach has the advantage of rapid initial implementation and ease of change. It also provides us a garbage collection system. The major disadvantage of our current implementation is that it runs slower than an equivalent compiled system. Furthermore, we are also restricted somewhat by Franz Lisp's view of data. Franz, in common with most modern Lisp dialects, has a number of data types, but must make concessions to the need for garbage collection. We generally garbage-collect over not only data space, but also over the (large) and relatively static program space. We intend to implement a generation scavenging garbage collector [Ungar, 1984] in the future. Needless to say, compilation of the byte codes is a high priority.

Newspeak is currently a residential system (ala Interlisp and Smalltalk), but separate compilation is planned.

4. Algebra and Analysis

The Berkeley Algebra and Analysis Development (BAAD) group, the major users of Newspeak, have designed a directed graph structure for describing the relationships of the principal components of algebraic manipulation systems in general use today. Certain extensions have been examined to see if it is, in fact, simple to provide for the future implementation of more esoteric algebraic objects and operations.

Principal advantages of this approach are:

- The developers are relatively free to redesign the mathematical hierarchy as needs change. It puts the algebraicist in control, rather than the data structure designer.

- The mathematical relationships between algebraic systems are sometimes revealed for what they are: matters of convenience of definition, subject to re-evaluation.

- Exceptions to the classification scheme are easily identified, can be discussed openly, and treated as special cases, rather than elevated to be the central objects of study.

4.1. General Structure

True to its name sake, Newspeak data structures provide both a diverse set of data structures and yet make the choice of a data structure unimportant. This is done through the use of generic functions: semantically similar function on similar data

structures have the same name. For example, the procedure for inserting a node into an AVL tree and to a B-tree both have the same name (*add!*). Thus the decision as to the final data structures can be deferred until after experimentation with alternatives and an empirical determination of the best choice.

4.1.1. Kernels

Kernels provide non-decomposable 'symbolic variables' used in polynomials, rational functions, power series, etc. Many systems only allow a single, system wide ordering of the kernels and give the user little control of how this is done. This affects efficiency of various global and local operations, usefulness of the display, and the effectiveness of pattern matching. SCARAB allows very flexible kernel ordering. Not only does SCARAB have a system ordering *and* allow the user to order kernels, it also allows the user to order the "internals" of a kernel (potentially, a separate ordering from the containing kernel). For example, SCARAB can represent $\sin(x+y)+\cos(y+x)$.

4.2. Implementation basis

A number of subsystems have been essentially implemented, documented, and mostly integrated into the system. Rather than list all current projects, we highlight a few:

4.2.1. Power series

At present power series in SCARAB are given a dense representation, that is, power series are represented by lists of their coefficients. The coefficients can come from an arbitrary ring, so that one can consider power series whose coefficients are matrices as well as power series with more typical coefficients, say, floating-point numbers. At this time only Taylor series (i.e. power series with only non-negative integral powers of the variable) have been implemented.

Programs have been written to use power series for solving explicit ordinary differential equations (i.e. ODE's of the form $y^{<n>}=p(x,y,y',\cdots y^{<n-1>})$). In addition to providing an application of power series arithmetic, it has proved to be a good way to test modifications of the power series code.

Future plans include a sparse representation of power series using a representation coefficient-exponent pairs, as well as allowing power series to have more general exponents, including negative or fractional powers of the variable. This will allow solution of more general ODE's. For example, we could solve an ODE about a singularity or we could obtain a solution involving a series with fractional powers of the variable. [Williamson, 1984]

4.2.2. BigFloats

Arbitrary precision floating point arithmetic has been implemented as an instance of a field. Included are basic arithmetic functions plus log, exponential, and trigonometric functions. The ideas for the algorithms are those used by Fateman's bigfloat package in Macsyma, although they are much cleaner as a result of restructuring. The pervasive use of global variables needed in Macsyma became unnecessary. Multiple-value returns are available in Newspeak, and the precision of float types is associated with the objects when they are created. The floats are also incorporated into complex types, and the same functions are available for them.

Extended bigfloats, which would include signed infinities and 'undefined' can also be used as though they constituted a field. A more rigorous implementation of a related type of extension is arithmetic on the Riemann sphere, which is also implemented. Projective coordinates and transformations on 'generalized circles' provides a neat application of our mechanism, and should be useful in conformal mapping.

5. Human Interface

The human interface must satisfy both beginners (to the system) and experts. Beginners need informative displays and help finding their way through an unfamiliar system. Experts want the system to be as unobtrusive as possible and provide a compact, sensibly orthogonal set of expressive commands. In each case, substantial development work will be needed to implement the features behind the scenes. Both experts and beginners want to be able to rely on the system to "do the right thing" in the presence of errors or elision. It has become extremely clear that a naive user does benefit from a naive system, but might well need sophisticated features as is present in the Apple Macintosh system and its precursors. Nevertheless, we feel it is important that this kind of cleverness be separate from the "core algebra" part of the system, which deals in fully specified (and strongly typed) mathematical manipulations. Attempts to second-guess the user which as a side effect impose rigid limits on what can be specified, are problematical.

Users of algebraic manipulation systems tend to be sophisticated mathematically but not familiar with the detailed program structure of the system. If we want to bring such systems to users who are both computationally and mathematically unsophisticated (e.g. a user with a technical problem but unaware of the tools required for the solution process) rather different problems are posed. The computer system has to have some meta-level information about the problem domain and the program capabilities. Incorporation of text-book or reference-book material into a computer system will undoubtedly be possible; the burgeoning 'expert system' area may have an impact on technology. At the moment, getting a computer program to cope with any but the simplest applied mathematical context would be a breakthrough.

5.1. The Interface technology: Windows/Menus/Mice

Recent technological advances make pointing devices and advanced graphics software more easily available. This has affected the approach to the user interface quite drastically. Using *windows*, several views of a session can be visible on a bit-mapped display at once. In addition to the multiple, overlaid structure of a 'desktop'-like view of information, it is possible to shrink windows down to 'icons' or pictorial reminder-symbols of the data available 'under' the icon. *Menus* are displays of currently possible commands from which an alternative can be selected. *Mice* are the most widely available pointing device on our systems, and have certain advantages to pointing with fingers or light pens. Among other features, they are not only indicators but provide additional input data via buttons. Working with menus and mice, a rapid display and selection mechanism can be implemented. SCARAB's interface design has been explored by Carl Andersen [Andersen, 1984] and Richard Anderson [Anderson, 1983].

5.2. Knowledge Bases for Programmers

Building tools to make the programmer's job easier is one of the favorite occupations of programmers. It increases productivity, at least in principle. Newspeak and SCARAB programmers are no exceptions.

Most developers of hierarchical languages have found that the proliferation of types (or their counterparts) necessitates the implementation of a browser so that users can find out about parts of the system that they have forgotten about, or never even knew about. This is the case with Smalltalk, and Interlisp's Masterscope. We have implemented a prototype knowledge base to help deal with the complexity of the system. The knowledge base can answer specific questions about types, procedures, and files, by giving documentation, argument lists, file dependencies, etc. It can also answer more complicated queries about global information (eg, "what files contain gcd procedures", "where is the *coef** parameter introduced, etc).

The above questions can only be asked by someone who has some familiarity with the system (e.g.,to jog one's memory). To aid novices, the knowledge base allows the use of pattern matching of regular expressions. Users of UNIX and other systems can readily attest to their usefulness in finding things. Another important feature of the knowledge base for novices is its ability to display information graphically on bit mapped displays. The ability to show hierarchies and quickly find out information about them using the point and click paradigm is particularly important for investigation. The display front end for the data base is not being pursued at this time.

We have specific plans for the improvement of the interface and functionality of the knowledge base, which we describe briefly in [Soiffer, 84].

5.3. Display: 'DREAMS'

Dreams is an elaborated expression syntax tree display system for mathematical expressions on bitmap computer displays. To support current display technology, we need to handle screen position, font information, and classes of displayable objects. Dreams, a Lisp-based system designed here, is the basis of an output and input feedback system for a user interface for algebraic manipulation.

Dreams expects expression input in the form of Lisp symbolic expressions. The input expression is processed to form a tree of successively refined expression "boxes" Once the box-tree is formed, DREAMS figures how to display the expression sensibly in its given window. A strategy for displaying the expression is chosen using the size and extent of the outermost enclosing box. Finally the tree of boxes is traversed printing the leaves (and doing a few other chores).

The envisioned interface using DREAMS would use a bitmap display (or two), windows, menus, a mouse, and (certainly) a keyboard. Windows would be used for text, mathematical expressions and subexpressions, and, eventually, graphs. Menus would provide help and inspection commands. A command language (from the keyboard) would be best for some kinds of interaction. The mouse would be used for focusing attention on expressions and for menu interaction. All of the above would be used in an expression editor for mathematical expressions.

5.4. Input/parser/editor and command processor

SCARAB's current parser is a variant of an operator precedence parser (similar to Macsyma's parser). It is user-extensible and table-driven, thus admitting different syntax for different problem domains. This property is important because syntax is a scarce resource and is used for different purposes in different problem domains (in fact, authors in the same area occasionally use different syntax for the same idea).

One annoyance with current parsers/displays is that the user types two dimensional input in one dimensional form and doesn't receive feedback from the system until after the expression is fully parsed. We are considering implementing a system that displays the expression in two dimensional form as it is being input (with "?" filling in for missing operands). This is a potentially useful feature, but we await implementation and user feedback before drawing any conclusions.

6. The Semantic Matching Facility in SCARAB

Pattern-matching has been a feature of many symbolic computation systems. A few systems such as Fenichel's FAMOUS was primarily a framework for expression of pattern matching and replacement. Numerous matching programs have been described in the literature. There has been much ado recently about the Prolog language, which is essentially a restricted matching engine. Rather than dwell on the well-known applications of pattern recognition to symbolic manipulation, or the various implementation

techniques, let us merely note that in SCARAB, we expect a very highly efficient pattern matcher for simple patterns, and a full backtracking unification matcher for the Prolog fans who may wish to be as general (or more general, considering the problems of commutativity and associativity in the usual commutative algebra semantics).

7. Applications

We mention briefly a few of our areas of current work.

7.1. Function representations

In a forthcoming paper, the need for representation of singularities of functions, for both numerical and symbolic needs. will be explored. We hope to see a more useful representation emerge; one that distinguished between a mathematical function and the representation of an expression which can sometimes be evaluated to obtain a value of the function at a point. The distinction is critical. [Kahan, Fateman, 1984]

7.2. Contours and Conformal Mapping

Data representation techniques and algorithms for dealing with maps on the complex plane or projective sphere provide a test-bed for ideas involving mathematical abstract and graphics, simultaneously.

7.3. 'NAGlink' and other numerical interface work

The Numerical Algorithms Group scientific software package has an enormous potential for use from symbolic systems. Access to this from lisp and our current Macsyma base will be extended to the BAAD work as the framework is put in place. Kevin Broughan of the Univ. of Waikato is working with us on this. Phil Colella of Lawrence Berkeley Labs has been working towards a system to aid in the generation of numerical PDE programs for parallel execution.

8. Comparison with other systems

In an earlier draft of this paper, we began a comparative study of Macsyma, Maple, Newspad (now called New Scratchpad), Reduce, SCARAB, and SMP. We have removed it from the current version because of length and our inability to get suitable consensus from highly opinionated people. Maybe another time...

Acknowledgments

This work was supported in part by the Systems Development Foundation, General Telephone and Electronics, the US Department of Energy.

References

Andersen, Carl M, "Specifications for a mouse-oriented mathematical manipulator," (draft, 1984)

Anderson, Richard, "EXED, an interactive algebraic expression editor." EECS Dept, U. Calif, Berkeley, 1983.

Fateman, R., "Open Problems in Algebraic Manipulation" 1983

Fateman, R., "The Legacy of Macsyma '82" 1984 (draft)

Favaro, J., "A Symbolic Executor for Fortran Code" MS project, UC Berkeley

Foderaro, J., "Design of a Language for Algebraic Computation Systems," Ph.D. Thesis,ECS Dept, U Calif, Berkeley. Dec. 1983.

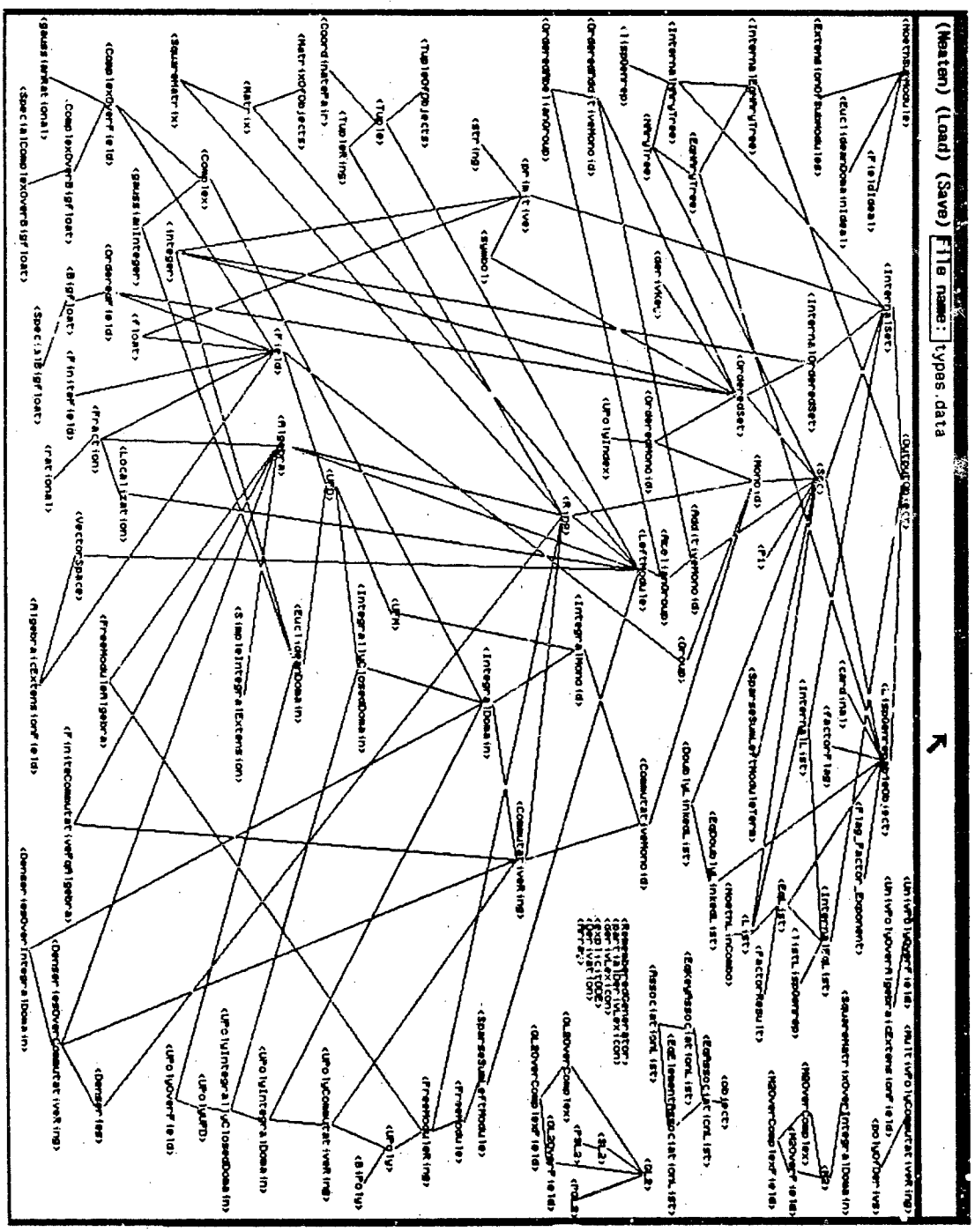
Foster, G., "DREAMS: Display REpresentation for Algebraic Manipulation Systems," M.S. Project Report, EECS Dept. UC Berkeley, April, 1984. (32 pages + program listings.)

Jenks, R. D., "MODLISP: An Introduction", Proc. Eurosam 79, Springer-Verlag Lecture Notes in Comp. Sci. 72, (466-480).

Kahan, W. and Fateman, R., "Symbolic Singularities and the IEEE Floating-Point Arithmetic Standard", in draft. 1984.

Soiffer, Neil, "Everything You Wanted to Know about SNARKDB but were Afraid to Ask," 1984.

Williamson, Clifton Jr., "Taylor Series Solutions of Explicit ODE's in a Strongly Typed Algebra System", ACM SIGSAM Bulletin *18, no 1,* (Feb. 1984) (25-29).



652 FATEMAN et al figure 1 SCARAB types (partial) 6/84

ON THE DESIGN AND PERFORMANCE OF THE MAPLE SYSTEM*

Bruce W. Char, Gregory J. Fee, Keith O. Geddes,
Gaston H. Gonnet, Michael B. Monagan, Stephen M. Watt

Symbolic Computation Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

Maple is a symbolic computation system under development at the University of Waterloo. A primary goal of the system is to be compact without sacrificing the functionality required for serious symbolic computation. The system has a modular design such that most of the mathematical functions exist as external library functions to be loaded only when they are invoked. The compiled kernel of the system is about 100K bytes in size. The library functions are interpreted. Efficiency is achieved through techniques including the identification of critical functions that are put into the compiled kernel, extensive use of hashing techniques, and careful design of the mathematical algorithms. Timing comparisons with other symbolic computation systems show that time efficiency is achieved as well as space efficiency.

1. Introduction

Maple is a language and system for symbolic mathematical computation which has been under development at the University of Waterloo since December, 1980. The Maple system can be used interactively as a mathematical calculator, and computational procedures can be written using the high-level Maple programming language.

The primary motivation for the design of Maple can be described as *user accessibility*. This concept has several aspects. The state of the art in 1980 was such that in order to have access to a powerful system such as Macsyma it was necessary to have a large, relatively costly mainframe computer and then to

* This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, and by the Academic Development Fund of the University of Waterloo.

dedicate it to a small number of simultaneous users. In the university setting, it was not feasible to offer symbolic computation to large classes for student computing. In a broader context, a large community of potential users of symbolic mathematical computation remained non-users. The development of the Mumath[1] and Picomath[2] systems showed that a significant symbolic computation capability could be provided on low-cost, small-address-space microcomputers. It seemed clear that it should be possible to design a symbolic system with a full range of capabilities for symbolic mathematical computation which was neither restricted by the small address space of the early microcomputers nor "inaccessible to the masses" because of unreasonable demands on computing resources. In particular, it seemed possible to design a modular system whose demands on memory would grow gracefully with the needs of the application program.

Portability was another of our earliest concerns, partly because we found ourselves users of a computing environment in transition, and partly because it was clear that a wide variety of computer systems would be coming onto the market in the decade of the 1980's.

Thus the primary design goals of the Maple system were: *compactness, modularity,* a *powerful set of facilities* for symbolic mathematical computation, *portability,* and a *good user interface.*

2. Design Philosophy

2.1. Space versus time

One of the fundamental conflicts facing systems designers is the tradeoff between space and time. In many circumstances, it is possible to improve speed by allowing space consumption to expand, and conversely it is often possible to conserve space consumption at the expense of speed. In the case of designing a symbolic computation system, the potential amount of system code is extremely large because such a system is inherently faced with the task of "mechanizing all of mathematics". An early design decision for the Maple system was that the system would have a relatively small kernel (say, on the order of a hundred kilobytes as opposed to a few megabytes). The vast bulk of system code for the various mathematical operations, such as gcd computation, factoring, integration, etc., exists as library codes to be loaded if and when they are needed. Furthermore, given the current state of the art of symbolic computation, we believe it is very important that the programs for these high-level mathematical operations should be readily accessible to, and modifiable by, the non-expert users of the system. Therefore, the library programs for the Maple system are coded in the high-level Maple programming language.

Since another design goal is to be portable across many different operating systems, the only practical implementation of the above model is that the library programs do not exist as compiled code but rather they are interpreted at run-time. Thus a fundamental design criterion for the Maple system is that

space is more crucial than time. In order to keep the compiled kernel small, we are willing to sacrifice some speed of execution. This can be viewed as a means to satisfy one of MacLennan's[3] design criteria, namely the principle of *localized cost:* users should only pay for what they use.

Given this model, there are several methods by which the time cost of the Maple system is kept to a minimum. One factor is the use of a *simple, efficient interpreter.* As one indication of the relative efficiency of Maple's interpreter, an experiment was performed using the "tak" function[4] and it showed Maple's interpreter to be about four times faster than Macsyma's interpreter on that particular benchmark. Consequently, the tradeoff between interpreted library code and compiled kernel code is not as great in Maple as in other systems.

Another factor in minimizing time cost is the identification of *critical functions* which are placed into the compiled kernel. This has been a dynamic process in the development of the Maple system. Some of the functions that were once in the external library but which have been identified to be critical and were moved to the kernel are: *indets* (to extract the indeterminates from an expression), *seq* (to construct a sequence), *subsop* (to substitute for a particular operand, or subexpression), *max, min, mod,* and *divide* (for polynomial division). On the other hand, some functions that were once in the kernel have been (or are being) moved to become external library functions (for example, *solve, sum,* and *int)* and for some internal functions an external library interface was developed to handle some of the higher-level cases (*diff, expand,* and *taylor* are examples of functions that have an external library interface).

Yet another very crucial factor in achieving minimal time cost is the use of *efficient algorithms.* This is perhaps a "motherhood" issue. However, particularly in symbolic computation, we have seen that some innocent-looking methods take exponential space and/or time while it is often possible to find better approaches. It has been our experience that most mathematical functions can be executed in the interpreted user language, instead of being included in the compiled kernel, without significantly affecting execution speed. Whereas the speed improvement that can be achieved by placing such a function into the compiled kernel is usually not more than 20-40%, we have in many instances achieved an order of magnitude improvement in speed by improving the algorithm. We note that the effort required to improve an algorithm once it is coded in the internal system implementation language is far greater than the effort required to modify an algorithm coded in the high-level language. Indeed, many of the contributors to the Maple system have never written code in the system implementation language, and would have been unlikely to make their contributions if coding in the low-level language was necessary. (We believe that this is a property of all system developments, not a special property of the Maple system and its particular system implementation language).

The conflict between space and time is, of course, not only a matter relating to the size of the compiled kernel. The run-time consumption of data space and processor time is of equal importance. When an algorithm is being designed for

a particular function, there are usually variations of the algorithm which trade off space consumption versus time consumption. We find it useful to consider a measure,

$$cost = (space)^2\ (time),$$

that arose originally in theoretical studies of time-space trade-offs in sorting [5]. It corresponds with our belief (which has also been expressed by others, such as Hearn [6]) that space is "scarcer" than time in typical algebraic manipulation.

2.2. Compact size as a design goal

The kernel of the Maple system (i.e., the only part of Maple that is written in the system implementation language and compiled) occupies a little more than 100K bytes on a VAX computer. The kernel system includes only the most basic facilities: the Maple programming language interpreter, numerical, polynomial, and series arithmetic, basic simplification, facilities for handling tables and arrays, print routines, and some fundamental functions such as *coeff, degree, subs* (substitute), *map, igcd* (integer gcd computation), *lcoeff* (leading coefficient of an expression), *op* (to extract operands from an expression), *divide, mod,* and a few others. Some of the fundamental functions have a small core coded in the kernel and an interface to the Maple library for extensions. The interface is general enough so that additional power, such as the ability to deal with new mathematical functions of interest to a particular user, can be obtained by user-defined Maple code. Some examples of functions which have such an internal core and an external user interface are *diff, expand, taylor, type,* and *evalf* (for evaluation to a floating-point number). Other functions supplied with the system are coded entirely in the user-level Maple programming language and exist in the Maple library, including *gcd, factor, normal* (for normalization of rational expressions), *limit, int, resultant, det,* and *solve.*

The compactness of a system is affected by many different design decisions. The following points outline some of the design decisions which have contributed to the compactness of the Maple system.

1. *The use of appropriate data structures.* We have designed into Maple a set of data structures appropriate to the mathematical objects being manipulated, with a direct mapping between these abstract structures and the machine-level "dynamic vectors".
2. *The use of a viable file system.* By having an efficient interpreter and by placing much of the code for system functions into the user-level library, Maple has the property that "you only pay for what you use". Writing functions in the user-level Maple language has the additional advantages of readability, maintainability, and portability.
3. *Avoiding a large run-time support system.* We view Maple as just one of many software tools that a user may employ to solve problems, regardless of which system it may be used on. We see no need to provide all of these tools within Maple itself, not only because they consume space and greatly

increase the problems of porting without providing any greater algebraic computation power, but also because many computing environments will allow their native software tools to be easily connected to Maple (say, as communicating processes).

4. *A policy of treating main memory as a scarce resource.* We believe that this point of view is important if we are to achieve the goal of providing a symbolic computation system to "the masses". Because we have adopted such a point of view, we are constantly concerned about which functions belong in the Maple kernel and which functions can be supplied as user-level code in the Maple library.
5. *The choice of the BCPL family of system implementation languages.* Implementing Maple in system implementation languages from the BCPL family has helped us to achieve the compactness goals outlined in the above points. The support of "dynamic vectors" in the implementation language allows the creation of compact data structures for the higher-level objects. Furthermore, an implementation language in the BCPL family typically has a run-time library that is small, selectively included, and yet provides the desired functionality.

2.3. Data structures

Maple has about 40 different internal data structures designed into it. Approximately one-quarter of these data structures correspond to programming language statements: assignment, *if*, *for*, *read*, etc. The remaining data structures correspond to the types of expressions including those formed using standard arithmetic and logical operators, numbers (integer, rational, and floating-point), lists, sets, tables, (unevaluated) functions, procedure definitions, equations, ranges, and series. All of these structures are represented internally as dynamic vectors.

This approach using dynamic vectors at the machine level and a rich set of data structures at the abstract level has significant advantages in improved compactness and efficiency of the resulting system code. First, in Maple there is only one level of abstraction above the system-level objects. We believe that the direct mapping between the abstract objects and the system-level objects simplifies our code and makes it more efficient than a scheme involving a less direct mapping. Secondly, we believe that the design of data structures should be related, if possible, to the language that describes the data objects. In our case we have a simple context-free language, and it is natural to relate the data structures to the productions in the grammar. This immediately suggests the need for many data structures since there are many productions in the language. Thirdly, dynamic vectors allow us, in many cases, to have direct access to each of the components of the structure at about the same cost. This is more desirable than the sequential access required when all objects are represented as lists. Fourthly, dynamic vectors are more compact than structures linked by pointers. In summary, an important part of the compactness and efficiency of Maple is

due to the use of appropriate data structures.

2.4. Computational power through libraries of functions

Another goal of the Maple system is to provide a powerful set of facilities for symbolic mathematical computation. In other words, we are not willing to achieve compactness by sacrificing the functionality of the system. Thus while the number of functions provided in the kernel system is kept to a minimum, many more functions for symbolic mathematics are provided in the system library, to be loaded as required. The functions in the system library are written in the high-level Maple programming language and are therefore readily accessible to all users of the Maple system. A load module for each library procedure is stored in "Maple internal format" which is a quick-loading expression-tree representation of the procedure definition. When a library function is invoked, its load module is read into the Maple environment (if not already loaded) and the expression tree is interpreted by the Maple interpreter.

3. The Use of Hashing in Maple

Maple's overall performance is in part achieved by the use of table based algorithms for critical functions. Tables are used within the Maple kernel in both evaluation and simplification, as well as less crucial functions. For simplification, Maple keeps a single copy of each expression or subexpression within an entire session. This is achieved by keeping all objects in a table. In user-level procedures, the *remember* option provides a hint to the interpreter that the values returned are likely to be needed again. These values are maintained in a table until a garbage collection is performed. Finally, tables are available at the user level as one of Maple's data types.

All of the table searching is done by hashing. The algorithm is an implementation of direct chaining in which the hash chains are dynamic vectors instead of linked lists. Each table element is stored as a pair of consecutive entries in the hash chain vector. The first entry of this pair is the hash key and the second is a pointer to the stored value. For efficiency, the hash chain vectors are grown a number of entries at a time and consequently some of the entries may not be filled.

3.1. Internal Use of Hash Tables

A computer algebra system spends most of its time evaluating and simplifying expressions. The Maple kernel manages two tables, the *partial computation table* and the *simplification table*, in an effort to make evaluation and simplification efficient. Other uses of hash tables in the kernel are the global symbol table and temporary tables used in performing input/output.

3.1.1. The Simplification Table

By far, the most important table maintained by the Maple kernel is the simplification table. All simplified expressions and subexpressions are stored in the simplification table. The main purpose of this table is to ensure that simplified expressions have a unique instance in memory. Every expression which is entered into maple or generated internally is checked against the simplification table, and if found, the new expression is discarded and the old one is used. This task is done by the simplifier which recursively simplifies (applies all the basic simplification rules) and checks against the table. Garbage collection deletes the entries in the simplification table which cannot be reached from a global name.

The task of checking for equivalent expressions within thousands of subexpressions would not be feasible if it was not done with the aid of hashing. Every expression is entered in the simplification table using its *signature* as a key. The signature of an expression is a hashing function itself, with one very important attribute: signatures of trivially equivalent expressions are equal [7]. For example, the signatures of the expressions a+b+c and c+a+b are identical; the signatures of a*b and b*a are also identical. If two expressions' signatures disagree then the expressions cannot be equal at the basic level of simplification.

Searching for an expression in the simplification table is done by:

— simplifying recursively all of its components;

— applying the basic simplification rules;

— computing its signature and searching for this signature in the table.

If the signature is found then we perform a full comparison (taking into account that additions and multiplications are commutative, etc.) to verify that it is the same expression. If the expression is found, the one in the table is used and the searched one is discarded. We have to do a full comparison of expressions only when we have a "collision" of signatures. How often this occurs is machine dependent. On a VAX, which has a 32-bit word, the signatures have 22 to 24 useful bits. An experiment we conducted measuring the collision rate during "typical" Maple computation indicated that signatures of inequivalent expressions coincide about once every 1500 comparisons for signatures of this size. Thus, the time spent searching the simplification table is typically negligible.

Since simplified expressions are guaranteed to have a unique occurrence, it is possible to test for equality of *simplified* expressions using a single pointer comparison.

3.1.2. The Partial Computation Table

Some functions tend to be called many times with the same arguments. Maple takes advantage of this fact by maintaining a table of function results for these functions. This is called the *partial computation table*. In it, function calls are used as the keys and their results as the values. Searching the hash table is extremely efficient so even for simple functions it is orders of magnitude faster than the actual evaluation of the function. Since both the function call and

function result are already existing as simplified data structures, the only storage consumed by an entry in the partial computation table is a pair of pointers. The partial computation table is cleared by garbage collection.

The original motivation for the partial computation table (which is still valid) was the observation that certain operations reproduce subexpressions multiple times in their results. As an example of this, consider the operation

taylor(exp(y/(1−x) + a), x=0)

where every term in the result contains the expression exp(y+a). Any further operation on this result (such as simplification, differentiation, etc.) will have to deal with this argument repeatedly.

There are four kernel functions that use the partial computation table: *diff*, *taylor*, *expand*, and *evalf*. (The *evalf* function is used for floating-point evaluation). External library functions and user-defined functions take advantage of the partial computation table by specifying the *remember* option in the procedure body. This is further discussed in a later section.

3.1.3. The Name Table

The simplest use of hashing in the Maple kernel is the *name table*. This is a symbol table for all global names. Each key is computed from the name's character string and the entry is a pointer to the data structure for the name. The name table is used to locate global names formed by the lexical scanner or by name concatenation. It is also used by functions that perform operations on all global names. These operations include: (i) marking for garbage collection, (ii) the saving of a Maple session environment in a file, and (iii) the Maple functions *anames* and *unames* which return all assigned global names and all unassigned global names, respectively.

3.1.4. Put Tables

It is possible to store Maple objects in a sequential file using a fast-loading internal format. The pointers in a collection of Maple objects form a general directed graph. The process of saving values in a file and later reading the values in from the file (usually in a different session) must preserve this graph, and in particular preserve shared subexpressions. A hash table is temporarily created for each **save** or **read** statement that uses internal format. These tables are known in Maple as *put tables*. The put tables are used to keep track of which subexpressions have already been output to (or input from) the file, and, in general, to perform the mapping from a directed graph into a linear (labelled) structure.

3.2. Option Remember

Functions written in the user-level Maple programming language, including the system-supplied external library functions, may use the partial computation table by specifying option remember in the options list of the procedure body. This is best viewed as a hint to the interpreter that the results of this function are likely to be used again. It may also be advantageous to use option remember in a function that is extremely expensive to compute, even if the result does not have a large probability of being re-used. It is important to note that remembered values disappear on garbage collection. For functions without side effects, this causes no problem because the act of remembering is an optimization; semantically it makes no difference whether the result is remembered or recomputed. For functions with side effects, this may cause erratic behaviour.

For many problems, remembering past results reduces the running time dramatically. For example, the Fibonacci numbers computed with

```
fib := proc(n)
    if n < 2 then n else fib(n-1) + fib(n-2) fi
end;
```

take exponential time to compute, while

```
fib := proc(n) option remember;
    if n < 2 then n else fib(n-1) + fib(n-2) fi
end;
```

takes only linear time. Although the effect is not as spectacular for most functions, it is not unusual for typical programs to be made roughly 30% faster by the judicious use of option remember. Of course this same factor could be obtained by recoding the crucial functions to use tables explicitly. The main advantage of option remember is that it achieves this performance factor without altering the function's code. The resulting code is very easy to read since the algorithmic intent is not obscured by code for saving intermediate results.

Sometimes the value of a function for some argument is known without actually computing it explicitly. An example would be an idempotent function such as *sqrfree*, which produces a square-free factorization of a polynomial. If the function uses option remember then this additional information may be entered in the partial computation table directly, using the *remember* function. An example would be:

```
p := sqrfree(q, x);
remember(sqrfree(p,x) = p);
```

Here the result of *sqrfree* is remembered for both p and q. The *remember* function evaluates its argument specially so that the function call is not executed.

Many library functions that use option remember have a front end that substitutes the indeterminates of the arguments for generic names. This is an attempt to remember a general result. This is done by the integrator, for example. All integrations are done with respect to the special variable name @X.

Once *int(x^20*exp(x),x)* has been computed, then the integral *int(y^20*exp(y),y)* is obtained from the partial computation table.

3.3. Arrays and Tables in the Maple Language

Arrays and tables are provided as data types in the Maple language. An array is a table for which the component indices must be integers lying within specified bounds. Arrays and tables are implemented using Maple's internal hash tables. Because of this, sparse arrays are equally as efficient as dense arrays. Contrary to the belief that arrays can be accessed quickly only by computing an element's address as an offset using the indices, our experience has shown that, in the Maple context, handling arrays as tables is at least as efficient while being more general.

A table object consists of (i) index bounds (for arrays only), (ii) a hash table of components, and (iii) an indexing function. The components of a table T are accessed using a subscript syntax, e.g., T[a,b*cos(x)]. Since a simplified expression is guaranteed to have a unique instance in memory, we use the address of the simplified index as the hash key for a component. If no component exists for a given index, then the indexed expression is returned.

The semantics of indexing into a table are described by its *indexing function*. Using an indexing function, it is possible to do such things as efficiently store a symmetric matrix or count how often each element of a table is referenced. Because each table defines its own indexing method, generic programs can be written that do not need to know about special data representations. Aside from the default, general indexing, some indexing functions are provided by the Maple kernel. Other indexing functions are loaded from the library or are supplied by the user.

Two typical system-supplied indexing functions are *symmetric* and *sparse*. The indexing function *symmetric* is used for tables in which the value of a component is independent of the order of the expressions in the index. This indexing function works by reordering the index expression sequence to produce a unique table reference. Thus, if the table T uses *symmetric*, the expression T[i,j] − T[j,i] evaluates to zero regardless of whether or not i, j or T[i,j] are assigned values. The indexing function *sparse* is used with tables for which a component is assumed to have the value 0 if it has not been assigned.

4. Hybrid Algorithms

It is well understood that many problems in algebraic computation do not have a single "best" algorithm. In fact, for some problems there may be many algorithms to choose from. Computing polynomial greatest common divisors is one such example. At least four major classes of gcd methods are in use in algebraic systems today. These are polynomial remainder sequence based algorithms[8,9], Hensel based algorithms[10,11], the sparse modular algorithm[12], and an integer-gcd based heuristic[13]. Comparison of their performance indicates that no one algorithm works best all the time. Some "win" on sparser problems, others on dense problems. Some work well on small problems and do poorly on problems of higher degree or numbers of variables. Others have such overhead that they should only be used on large problems where their asymptotic complexity begins to assert itself.

How then does a general purpose system organize the code to solve a problem where several algorithms should be considered? Consider applying a predetermined, fixed algorithm to all problems. Such a single algorithm must be robust. This rules out the application of algorithms that will succeed, or succeed quickly, only on certain classes of problems. The alternative to using a single algorithm is to automatically select from several: a "hybrid", or polyalgorithm. A polyalgorithm could also possibly use one method to partially solve the problem (for example, eliminating some of the unknowns from a system of equations), and then switch over to another more general and expensive algorithm when appropriate. This is not always possible but when it is, it often makes a substantial overall improvement in efficiency.

Thus, a hybrid procedure can be viewed as automating not only the algebraic computation, but also automating the expertise in selecting and combining algorithms for a particular problem. If this is done well, it can relieve the user from the unwanted burden of learning details of algorithms in areas that are not of direct interest to him or her. In order to justify a hybrid approach in contrast with using a single algorithm, it must be shown that the decisions about which algorithm to use, and when to start using it, can be automated without introducing undue overhead. It must also be shown that the hybrid algorithm often performs much better than any single algorithm, and rarely performs much worse.

We describe the Maple implementation of hybrid algorithms for several different problem areas. These include the determinant code, the gcd code, and the solve code (for solving systems of equations). All of the codes for these problems are implemented in the user-level Maple language and therefore they are interpreted rather than compiled. Timing comparisons are presented to show the relative performance of Maple, Macsyma, and Reduce on some sample problems. All timings (in seconds) were obtained on a Vax 11/780 running Berkeley Unix 4.2, by calling the user-level routine for solving the given problem.

4.1. Determinants

The two methods used are fraction-free Gaussian elimination and minor expansion. Comparisons of these two methods are given by Gentleman and Johnson, and Horowitz and Sahni [14,15]. Those authors' comments, their timing results, and our own experience, suggest the following general guideline for choosing between Gaussian elimination and minor expansion:

(1) for matrices with many numerical entries and/or larger dense matrices in only a few variables, use gaussian elimination;

(2) for small matrices (of dimension ≤ 5), sparse matrices, and matrices with many variables, use minor expansion.

We are also experimenting with the idea of running fraction-free elimination steps until a small pivot is no longer available, then switching to minor expansion. We note that the strengths and weaknesses of a particular computer algebra system must also be taken into consideration in algorithm selection. For example, Maple is particularly well suited to using minor expansion because of the facility provided by the partial computation table as described previously. By using *option remember*, we can implement the standard recursive definition of a determinant in terms of its minors (see Figure 1). Without the help of *option remember* (or some similar facility), this algorithm would be extremely inefficient, as minors would be recomputed an exponential number of times. In using *option remember*, the system avoids recomputation by automatically keeping track of the minors' determinants as it computes them. Gentleman and Johnson avoid recomputation by computing the determinants of the minors "bottom up". We believe that the use of *option remember* in Maple leads to a more natural and simpler coding, and furthermore avoids an exponential amount of work for the sparse cases.

The above discussion of determinant code organization is equally applicable to the problem of computing matrix inverses. For this problem, there is a choice between fraction-free Gaussian elimination and computing the inverse via the adjoint of the matrix.

The timing results in Table 1 show that Maple's determinant code performs quite well over a variety of different problems. For these (and subsequent) timing comparisons, note that Maple's code is executed by an interpreter while the Macsyma and Reduce codes have been compiled. For a detailed listing of the test problems used in Table 1, see[16]. We find that the overhead of algorithm selection is not unreasonable compared to the cost of computing the determinant.

```
minor := proc (A,r,c,n) local i, s, t; option remember;
# Compute the determinant of the n by n minor of the matrix A, whose row
# and column indices are given in the lists r and c, using minor expansion.

    if  n = 1 then A[r[1],c[1]]
    elif n = 2 then A[r[1],c[1]]*A[r[2],c[2]] - A[r[1],c[2]]*A[r[2],c[1]]
    elif n = 3 then
        A[r[1],c[1]] * (A[r[2],c[2]]*A[r[3],c[3]] - A[r[2],c[3]]*A[r[3],c[2]]) -
        A[r[2],c[1]] * (A[r[1],c[2]]*A[r[3],c[3]] - A[r[1],c[3]]*A[r[3],c[2]]) +
        A[r[3],c[1]] * (A[r[1],c[2]]*A[r[2],c[3]] - A[r[1],c[3]]*A[r[2],c[2]])
    else
        t := subsop(1=NULL,c);
        s := 0;
        for i to n do if A[r[i],c[1]] <> 0 then
            s := s + A[r[i],c[1]] * (-1)^(i+1) * minor(A,subsop(i=NULL,r),t,n-1)
        fi od
    fi;
    if type(", `+`) then expand(") else " fi
end
```

Figure 1: Maple library code for computation of a minor's determinant.

Matrix description	Maple	Macsyma (1)	Reduce (1)
5 by 5 Vandermonde	6.5	10.5	0.8
5 by 5 Dense univariate Bezout	19.9	19.8	17.5
6 by 6 Bezout (from Sigsam #7)	133.6	271.6	132.9
12 by 12 Eigenvalue problem (band matrix)	42.5	719.5	10.8
10 by 10 Hilbert	13.5	236.0	300.7
10 by 10 Univariate Sylvester	40.2	1414.0	264.9
11 by 11 Tridiagonal (univariate)	4.8	95.1	0.9
14 by 14 Eigenvalue problem (bivariate)	279.7	>1500	>1500

Table 1: Timings for determinant problems.

Notes: (1) The default algorithm for both Macsyma and Reduce on our system is minor expansion. Also, in collecting the Macsyma times, *ratexpand* was applied to the result from *determinant* where necessary.

4.2. Greatest Common Divisors of Polynomials

Maple's gcd code makes use of two algorithms. Initially, a heuristic, gcdheu,[13] is tried. Gcdheu computes polynomial gcds via polynomial evaluation, an integer gcd computation, and single-point polynomial interpolation. This method was motivated by the fact that the hardware provides support for integer arithmetic, and consequently even multiple-precision integer arithmetic is fast, whereas there is no hardware support for polynomial arithmetic. Therefore although the complexity of an integer gcd based computation is exponential in the number of variables, such a method performs very well on a significant class of practical problems. Roughly speaking, for most problems in three or fewer variables we find that gcdheu is the algorithm of choice. On the other hand, there are many problems that gcdheu would be extremely slow to solve. Fortunately, it is easy for gcdheu to detect its bad cases by estimating the size of the integer gcd problem before generating it. When the integer gcd problem about to be generated would be larger than a pre-specified size (currently set at 3000 digits), gcdheu gives up. Control is passed back to the main code, which then sets up the problem for the second algorithm. The second algorithm is a Hensel-based gcd algorithm (EEZGCD).*

Another important feature of gcdheu is that its code size is tiny, relative to Hensel-based codes or the sparse modular code. For most sessions we expect that the gcdheu algorithm will be sufficient and consequently the larger codes will not be loaded. This organization helps to maintain Maple's goal of compactness.

In Table 2 we present timings for some gcd problems. These problems were generated at random. All problems are non-trivial in either the number of variables, their degrees, the number of terms, or the size of the coefficients. Seven of the problems are sparse, three are dense; five of the problems have a non-trivial gcd, and in the other five the gcd is one. For a detailed listing of the test problems used in Table 2, see[16]. The timings illustrate both the power of gcdheu as an algorithm in its own right, and the robustness of the overall code organization since the timings for larger problems are also very reasonable.

* Code for the sparse modular algorithm has been written for Maple[17] but it is not yet determined how this will be incorporated into the gcd polyalgorithm.

Problem	Maple	Macsyma (1)	Reduce (2)
1	2.2	67.8	>1500
2	5.8	42.7	1472
3	6.3	17.5	>1500
4	10.7	31.3	>1500
5	5.1	4.8	>1500
6	29.5	69.4	>1500
7	9.6	2.4	>1500
8	25.7	24.9	11.6
9	110.6	34.8	>1500
10	34.5	24.6	>1500

Table 2: Timings for some gcd problems.

Notes: (1) Using the default Macsyma gcd algorithm, spmod. (2) Using a PRS algorithm with trial-division[18].

4.3. Solving Systems of Equations

The first method to be tried in *solve* on a system of equations is gensys. At each step, gensys selects the "easiest" equation to be solved for a particular unknown. That unknown is then eliminated from the other equations of the system via a substitution. Both under- and over-determined systems of both linear and nonlinear equations can be solved in this way. Gensys spends a considerable amount of time evaluating the complexities of each equation. Ideally, all unknowns will be found and eliminated from "simple" equations, preserving sparsity where possible. What is considered a simple equation in gensys is any equation containing an unknown that when eliminated, will most likely produce a simpler, smaller system. This elimination procedure is repeated until either the system has been reduced to a single equation, in which case back-substitution is employed to obtain the solution, or else further progress is blocked because proceeding would generate, for example, new quotients of polynomials.

At this point, control is passed to a second method, a modified fraction-free Gaussian elimination algorithm for solving rectangular linear systems. This algorithm solves the remaining linear problems for which gensys would be too expensive. If the system is found to be nonlinear then control is passed back to gensys, which continues the elimination. A resultant based algorithm is called for the general case when gensys cannot proceed.

This organization of the solve code has several advantages. Simple linear and nonlinear equations are eliminated quickly. Gensys preserves sparsity for as long as is practical. Since gensys is by nature a sparse algorithm, we are interested in how it performs on dense systems (its worse case) where much of the time will be spent in looking at the equations. The first problem in Table 3 shows that the cost of using gensys rather than immediately using Gaussian el-

imination is not unreasonable. (Our time for directly applying Gaussian elimination on the first problem is 23 seconds). For large sparse systems, the hybrid algorithm performs much better than Macsyma's default algorithm. The first four times reported in Table 3 are for linear systems and the last two are for nonlinear systems. For a detailed listing of the test problems used in Table 3, see[16].

Problem description	Maple	Macsyma	Reduce
10 equations, 10 unknowns dense with integer coefficients	50.8	22.5	21.5
30 equations, 29 unknowns integer coefficients	55.6	122.9	(1)
50 equations, 50 unknowns sparse band system	138.6	1180	1162
147 equations, 49 unknowns very sparse with trivariate coefficients	96.5	1078.3 (2)	(1)
19 equations, 17 unknowns sparse system with 4 solutions	68.5	>1500	(1)
22 equations, 17 unknowns sparse system with no solution	17.9	>1500	(1)

Table 3: Timings for solving systems of equations.

Notes: (1) Reduce's solver was not programmed to solve over-determined systems. (2) This time reported for Macsyma was obtained by Prof. Stanly Steinberg of the University of New Mexico, using special purpose code developed for the problem. Macsyma's default algorithm could not solve this problem in under 1500 seconds.

5. Further Comparisons of Space and Time

Table 4 presents some timing comparisons for a variety of symbolic computation problems which are summarized below. More details about these test problems can be found in [16]. All times are in seconds in the form *user time* + *system time* obtained from the Unix time command on a Vax 11/780 running Berkeley Unix version 4.2. The *Maple space* column indicates the total number of bytes of memory required by Maple (compiled kernel plus data space) for the problem. Note that automatic garbage collection is not yet operational in Maple and therefore the space consumption increases monotonically with execution time. Note also that the initial size of code plus data space for Reduce is over one megabyte and for Macsyma is over three megabytes, in contrast with Maple's initial size of 104K bytes.

Problem	Maple space	Maple time	Macsyma time	Reduce time
1	139K	10.4 + 0.6	23.3 + 8.4	134.0 + 29.7
2	145K	14.3 + 1.8	40.4 + 13.6	180.0 + 26.6
3	222K	4.8 + 1.0	46.1 + 21.0	43.5 + 10.0
4	777K	18.7 + 2.5	180.8 + 11.2	88.6 + 4.9
5	169K	1.5 + 0.4	26.2 + 9.7	4.7 + 1.4
6	432K	32.6 + 4.0	68.9 + 11.7	37.1 + 7.6
7	251K	23.6 + 2.4	88.5 + 18.3	>1000.0
8	169K	2.0 + 0.4	93.3 + 14.2	Not attempted
9	185K	2.2 + 0.5	183.3 + 22.1	Not attempted
10	603K	27.2 + 2.8	101.2 + 20.4	33.5 + 7.9
11	181K	2.6 + 0.5	3.3 + 5.4	Not attempted
12	247K	5.7 + 1.1	3.0 + 6.0	7.5 + 3.4
13	302K	12.4 + 1.5	36.7 + 14.8	11.5 + 3.0
14	152K	1.2 + 1.2	2.9 + 4.7	1.3 + 1.6
15	414K	16.8 + 2.4	46.9 + 13.5	Not attempted

Table 4: Space and time statistics for a variety of problems.

Description of Problems in Table 4

1 Compute and print 1000!.
2 Compute a "big" rational number: 13^1000 / 14^960
3 Compute arcsin(.7102633504 6985192786 3258652083 7914203194 9324761436) to 50 digits.
4 Read in a random polynomial but do not print it. It has 396 terms, 5 variables, each of degree 6, and 4-digit coefficients.
5 Do 1000 assignments in a *for* loop without printing:
for i to 1000 do a := i od.
6 Solve a sparse linear system of equations (20 by 20, 3 terms per equation, random 4-digit integer coefficients).
7 Compute and print −diff(u,z) from [19,p. 510]
8 Factor 16254399361 (= 89137 * 182353).
9 Taylor series of sin(x^5−3*x^8+7*x^29+13*x^59) up to the term in x^64.
10 Compute and print the f and g series to order 16.[20]
11 Compute and print the indefinite summation: sum(i^12, i = 0..n−1).
12 Find $\int x^{30} e^x \, dx$.
13 Expand (a+b+c+d+e+f+g+h)^4 and print it.
14 Recursion test: f := proc(n) if n=0 then 1 else f(n−1) fi end; f(100).
15 SIGSAM Problem #3: Reversion of a double series[21], solved to order 4 by Hall's 2nd method[22] (includes print time).

6. Future Development

The Maple project is an ongoing activity of the Symbolic Computation Group at the University of Waterloo. We mention here some of the developments that are anticipated for future versions.

6.1. Algorithm improvements

Some of the existing mathematical packages are being improved. For example, the gcd package is largely completed but its multivariate Hensel-based (EEZGCD) algorithm will have Wang's coefficient pre-determination added to it for improved performance on sparse problems. The factor package similarly needs to exploit coefficient pre-determination (this is currently implemented only for the leading coefficient) in the multivariate Hensel lifting stage. Maple's univariate factorizer is a heuristic algorithm based on single-point evaluation and integer factorization [23], which performs well on problems with reasonably small integer coefficients, but we have yet to complete implementation of the Berlekamp/Hensel algorithm for univariate factorization. Another package to be completed is the integration package, which currently includes only a "front end" of heuristics. Eventually the Risch procedure will be included as part of *int* (work is in progress). The method of resultants is being added to the solve package for solving systems of polynomial equations.

There are numerous mathematical packages yet to be introduced into the Maple library. For example, a differential equations package and a tensor package have yet to be implemented.

6.2. Language facilities

The following are some of the language facilities awaiting implementation.

(1) Automatic garbage collection (currently the user must issue a gc() function call).

(2) Pattern matching simplification.

(3) User-specified simplification rules.

(4) Operators, including an operator algebra facility.

(5) Foreign function interface (some work has been done on an interface to Fortran and an interface to Prolog).

(6) Language conversion (some work has been done on converting Maple output to Fortran syntax).

6.3. Porting Maple

The Maple system is designed to be portable to various operating systems, usually in the C language. The main restriction is that the host system must support a large address space (e.g., Maple is not designed to work with 16-bit addresses) and must have enough physical memory (we recommend a minimum of one megabyte) to be capable of handling typical symbolic computations. To date, Maple has been fully ported between C under Berkeley Unix on a VAX 11, B under GCOS-8 on a Honeywell DPS-8, C under Xenix on a Spectrix S-10 (M68000-based microcomputer) and C under TOPS-20 on a DEC20. The VAX/Unix and DEC20 versions are currently in distribution. Work is well underway to port Maple to the IBM VM/CMS operating system and to the WICAT operating system. Planned for the near future is a version for DEC's VAX/VMS operating system (see below).

6.4. Maple in undergraduate teaching

We are particularly excited about the introduction of Maple into the mainstream of the undergraduate mathematics curriculum. Current plans include experimenting with Maple as a laboratory tool to be used by first- and second-year calculus and linear algebra students at the University of Waterloo. A pilot project is scheduled for the term beginning in January 1985, probably using a VAX 11/785 running VMS, to service approximately 300 students. To increase the capacity beyond the size of a pilot project, we expect to move to a network of microprocessors connected to a file-server VAX, with the bulk of the symbolic computation being done on the microprocessors.

7. Availability of the Maple System

Maple version 3.2 is currently being distributed for VAX/4.2 BSD Unix, and for DEC20 systems running TOPS-20. During the latter part of 1984 we plan to begin distribution of the Maple system (version 3.3 and beyond) through the facilities of Watsoft, an institution within the University of Waterloo which is responsible for the distribution of several other software products (WATFOR, WATFIV, WPascal, etc.). We expect that the Watsoft distribution will initially include IBM mainframes (VM/CMS), and eventually VAX/VMS and M68000-based systems.

Licensing and distribution information, and copies of Maple documentation[24,25,26], are available by writing to:

Maple Lab
Symbolic Computation Group
Department of Computer Science
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

Acknowledgements

We wish to acknowledge the contributions our Maple co-workers Marta Gonnet and Benton Leong, and Prof. Stan Devitt of the University of Saskatchewan, have made to the design and development of Maple. We also wish to thank Prof. Stanly Steinberg of the University of New Mexico for supplying us with one of the test problems used in this paper, and our fellow Macsyma users for helping us with the Macsyma system at various times.

References

1. Art Rich and David Stoutemyer, "Capabilities of the muMATH-79 Computer Algebra System for the INTEL-8080 Microprocessor," *Proceedings of Eurosam '79*, pp. 241-248 Springer-Verlag, (1979).
2. David Stoutemyer, "PICOMATH-80, an Even Smaller Computer Algebra Package," *SIGSAM Bulletin* **14**(3) pp. 5-7 (1980).
3. Bruce J. MacLennan, *Principles of Programming Languages: Design, Evaluation, and Implementation*, Holt, Rinehart, & Winston,, Toronto (1983).
4. Martin Griss, Eric Benson, and Gerald Maguire, Jr, "PSL: A Portable LISP System," *Proceedings of the 1982 ACM Symposium on Lisp and Functional Programming*, pp. 88-97 (1982).
5. Allan Borodin, Michael Fischer, David Kirkpatrick, Nancy Lynch, and Martin Tompa, "A Time-Space Tradeoff for Sorting on Non-Oblivious Machines," pp. 319-327 in *Proceedings of 20th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society (1979).
6. A.C. Hearn, "Reduce - A Case Study in Algebra System Development," *Computer Algebra. Proceedings of Eurocam82*, Springer-Verlag, (1982). Lecture notes in Computer Science, v. 144.
7. Gaston H. Gonnet, "Determining Equivalence of Expressions in Random Polynomial Time," *Proceedings of the 16th ACM Symposium on the Theory of Computing*, pp. 334-341 (April 1984).
8. G.E. Collins, "Subresultants and Reduced Polynomial Remainder Sequences," *Journal of the ACM* **14** pp. 128-142 (1967).
9. W.S. Brown, "The Subresultant PRS Algorithm," *ACM Transactions on Mathematical Software* **4**(3) pp. 237-249 (1978).
10. Joel Moses and David Y.Y. Yun, "The EZ GCD Algorithm," pp. 159-166 in *Proceedings of the ACM Annual Conference*, (August 1973).
11. Paul Wang, "The EEZ-GCD Algorithm," *SIGSAM Bulletin* **14**(2) pp. 50-60 (May 1980).
12. Richard Zippel, "Probabilistic Algorithms for Sparse Polynomials," *Proceedings of Eurosam 79*, pp. 216-226 Springer-Verlag, (1979). Springer-Verlag Lecture Notes in Computer Science no. 72.

13. B.W. Char, K.O. Geddes, and G.H. Gonnet, *GCDHEU: Heuristic Polynomial GCD Algorithm Based On Integer GCD Computation*, To appear, Proceedings of the 1984 International Symposium on Symbolic and Algebraic Manipulation July, 1984.

14. W.M. Gentleman and S.C. Johnson, "Analysis of Algorithms, A Case Study: Determinants of Matrices with Polynomial Entries," *ACM Transactions on Mathematical Software* **2** pp. 232-241 (September 1976).

15. E. Horowitz and S. Sahni, "On Computing the Exact Determinant of Matrices with Polynomial Entires," *Journal of the Association for Computing Machinery* **22**(1) pp. 38-50 (January 1975).

16. B.W. Char, G.J. Fee, K.O. Geddes, G.H. Gonnet, M.B. Monagan, and S.M. Watt, *On the Design and Performance of the Maple System*, University of Waterloo Computer Science Department Research Report CS-84-13. June, 1984.

17. Mark E. Bryant, *The Sparse Modular GCD Algorithm in Maple*, University of Waterloo, Dept. of Computer Science (December, 1983). M.Math essay.

18. Anthony Hearn, "Non-modular Computation of Polynomial GCD's using Trial Division," *Proceedings of Eurosam 79*, pp. 227-239 Springer-Verlag, (1979). Springer-Verlag Lecture Notes in Computer Science no. 72.

19. J.A. Campbell and Simon, "Symbolic Computing with Compression of Data Structures: General Observations, and a Case Study," *EUROSAM 1979*, pp. 503-513 Springer-Verlag, (1979).

20. Richard J. Fateman, "An Open Letter from Fateman to Veltman," *SIGSAM Bulletin*, pp. 5-11 (Nov. 1978).

21. John S. Lew, "Problem #3 - Reversion of a Double Series," *SIGSAM Bulletin*, (23) pp. 6-7 (July 1972).

22. Andrew D. Hall Jr., "Solving a Problem in Eigenvalue Approximation with a Symbolic Algebra System," *SIGSAM Bulletin*, (26) pp. 15-23 (June 1973).

23. Leonard Adleman and Andrew Odlyzko, "Irreducibility Testing and Factorization of Polynomials," *Mathematics of Computation* **41**(164) pp. 699-709 (October 1983).

24. B.W. Char, K.O. Geddes, G.H. Gonnet, and S.M. Watt, *Maple User's Manual, 3rd edition*, University of Waterloo Computer Science Department Research Report CS-83-41. December, 1983.

25. B.W. Char, K.O. Geddes, W.M. Gentleman, and G.H. Gonnet, "The Design of Maple: A Compact, Portable, and Powerful Computer Algebra System," *Proceedings of Eurocal '83*, pp. 101-115 (1983). Springer-Verlag Lecture Notes in Computer Science no. 162

26. B.W. Char, K.O. Geddes, and G.H. Gonnet, *An Introduction to Maple: Sample Interactive Session*. University of Waterloo Computer Science Department Report CS-83-16 May, 1983.

FIVE YEARS OF SMP

by

STEPHEN WOLFRAM
The Institute for Advanced Study
Princeton, New Jersey 08540

ABSTRACT

SMP was born soon after the Second MACSYMA User's Conference, in the fall of 1979. It was started because I believed that systems like MACSYMA could be widely useful, but that none of the existing ones were really adequate. I built SMP to be a much more general system, with a coherent structure, based as closely as possible on standard mathematics. I included a core of mathematical knowledge, together with a language, largely based on pattern matching, to define new constructs and operations. The rather novel nature of the SMP language seems to have paid off handsomely: it is sufficiently natural that most users barely notice its structure. Within SMP, it is possible to create programs for all kinds of applications. The challenge now is to build up a library of programs that covers a very broad area. In this way, the contents of tables of formulae and handbooks of mathematical methods can be codified for computers.

WHICH POLYNOMIAL REPRESENTATION IS BEST?

Surprises Abound!

A Preliminary Report

by
David R. Stoutemyer,

The Soft Warehouse,
Honolulu

May, 1984

ABSTRACT

Computer algebra systems have been based on a bewildering variety of algorithms and representations for expressions. It is unclear how much of any performance differences between these systems is attributable to representations rather than differences in the hardware and software environment. Thus, there is clear need for a systematic comparison of the space and speed efficiencies for the major altematives all implemented in the same software and hardware environment. This paper provides such a comparison for unfactored polynomials. Their representation is crucial for any system because even factored rational or more general expressions typically include unfactored polynomials as subexpressions; and for many problems most of the space and time is consumed in adding, multiplying, dividing, factoring and determining greatest common divisors of expanded polynomials.

More specifically, comparisons are made between distributed vs recursive form, explicit vs implicit operators, binary vs n-ary operators, dense vs sparse representations, distributed vs extracted variables, and ascending vs descending order of degree, using several of the most practical algorithms. Tne major surprise is that a recursive dense representation is extremely efficient even for quite sparse multivariate polynomials. Another surprise is that sparse recursive Cambridge Prefix representation is fast enough to warrant selection on its other merits. Another surprise is that even for rather large problems, asymptotically attractive multiplication methods such as the tournament merge do not fare as well as certain variants of more mundane algorithms.

1. INTRODUCTION

Many of us who begin as users of computer algebra systems eventually fall prey to the temptation of implementing our own system. Besides masochism, reasons include a desire to focus on a different mix of capabilities, a different style of user interface, different computers or new foundations. There is certainly no evidence that we have witnessed the "last remake of computer algebra." Consequently for maximum progress, it wise to fully appreciate the lessons to be learned from earlier systems.

A particularly crucial decision is the choice of data structures for representing expressions, because the choice strongly influences the efficiency, the ease of implementation, and the class of representable expressions. Next in importance is the choice of algorithms for the most time-consuming elementary algebraic operations such as polynomial multiplication. Although it is desirable to make a wise decision in these regards, the literature and folklore provide little guidance:

a) Implementors generally do not publish information about which alternatives they implemented but rejected after comparative testing.

b) Each implementor probably tried only one or a few alternatives because of the substantial effort required to implement and test each alternative.

c) The sequence of problems published in the SIGSAM Bulletin [1972] and elsewhere compare different algebra systems each using a different data structure on a different computer. Moreover, the problems each test a mixture of various system aspects plus ingenuity in modeling the problem. Thus, the contributions of different data structures and fundamental algorithms is inextricably mixed with other factors.

d) Data structures and algorithms with the most attractive asymptotic complexity are often inferior for problems of typical interest or practical size.

For these reasons, I decided to compare the major general-purpose alternatives all in the same hardware and software environment. Of course the relative efficiencies are likely to be somewhat different in other environments, and perhaps even the relative rankings might be somewhat different. Consequently, the appendix presents timings for the major low-level operations that influence these relative efficiencies: the time for making function calls, list traversing, building new lists from a storage pool, reclaiming unused storage, and doing arithmetic. Analogous timings for another environment could help support prediction of the relative rankings there.

At the next level of consideration -- the one that I wish to address -- most of the time is often spent multiplying, dividing and adding or subtracting polynomials. For example, typical polynomial

factoring and greatest-common-divisor routines use these operations in their inner loops. Consequently, it is extremely important to determine the relative merits of alternative data structures and algorithms for these fundamental operations.

Although I have not yet tested all of the representations that I would like to test, I am publishing this preliminary report in order to solicit other opinions before deciding upon a set of data structures, algorithms and test cases.

2. REPRESENTATIONAL ALTERNATIVES & ISSUES

2.1 Recursive vs Distributed

In recursive form, a multivariate polynomial is represented as a polynomial in one variable, with coefficients that are polynomials in the remaining variables. Each of the coefficient polynomials is similarly represented, and so on until we reach the coefficient ground domain -- usually numbers: usually integers.

In distributed form, a multivariate polynomial is represented as a sum of terms, each of which is a product of a ground-domain coefficient and powers of variables. With any canonical ordering within and among terms, recursive and distributed form are both canonical. Moreover, it is easy to display either form as the opposite form without actually forming an internal structure representing that opposite form.

Recursive form can automatically achieve some sharing of common subexpressions because of the more encompassing definition of coefficients, hence of similar terms. During addition, entire groups of terms may be merged in one step, reducing list-building costs. The recursively-defined coefficients are also more convenient in many respects for division, greatest common divisors and factoring.

Distributed form wastes less space on list terminators, hence less time on function calls. Moreover, distributed form is more amenable for array implementation, which saves even more space. However, the longer lists or arrays entail greater sorting cost for multiplication of sparse representations.

Although recursive form yields a tree-like data structure that reduces sorting costs, the tree is not necessarily balanced. As described by Fateman [1974], structures such as 2-3 trees or AVL trees force balance, and therefore they can asymptotically furtner reduce sorting costs for very large problems at the expense of program complexity. As described by Gustavson and Yun [1976], bit-arrays provide another alternative for reducing sorting costs at the expense of space. As described by Goto and Kanada [1976], hash tables can reduce **average** sorting cost at the expense of **worst-case** sorting cost.

2.2 Explicit versus Implicit Operators

Any or all of the operators "+", "*", and "^" can be either implicit or explicitly included in the data structure. When included, they are usually placed at a standardized most accessible place to facilitate quick direct access. For example, x^3 can be represented by the Lisp list (^ x 3). This is an example of **Cambridge Prefix**, wherein all expressions are either numbers, variable names, or a list beginning with an operator and followed by its Cambridge-Prefix operands. Explicitly tagged data structures are increasingly popular in both hardware and software. In the context of Lisp, they have the appealing advantage that Cambridge Prefix is the legitimate form of argument for muSIMP's EVAL function, which thereby provides an efficient built-in main simplification function.

In order to save space, we can omit operators where they are known by context. For example in an expanded polynomial, we could represent x^3 by the list (x 3). In order to elide more than one kind of operator, we generally have to pad degenerate cases of polynomials so that context is properly deducible starting from a standardized outermost level. For example, if x + 3 and x * 3 are also represented by the list (x 3), then we cannot determine the implicit operator unless we keep track of the level of descent from a standardized form. Thus we might want to represent a polynomial as a list of terms even when there is only one term, similarly representing a term as a list of factors even when there is only one factor. Although implicit operators thus tend to require **more** space for trivial polynomials, the savings can be substantial for large polynomials where savings are more important. The structural regularity may also speed polynomial operations by obviating the need for some tests. For example, if we know that each term is a product, then we do not have to test whether it might alternatively be a name, number or power.

2.3 Binary versus N-ary operators

For Cambridge Prefix, we can can have "+" and/or "*" be either binary or n-ary operators. The latter may save space, particularly in the case of sequential array storage. However, the binary choice can facilitate writing the program almost entirely in a more natural and educational rule-driven style that uses direct recursion rather than secondary dispatching to list-traversing procedures.

For binary operators on systems such as muSIMP that do not use "CDR-coding", we can save space by making the two operands be a dotted pair rather than a list. For example, we can represent x^3 as (^ x . 3) or even as (x . 3) if we are also eliding "^". Implicit binary operations represented as dotted pairs generally entail some padding with zeros and ones in order to maintain the proper context. For example:

a) A variable might always be associated with an explicit degree even if it is 1 or perhaps even 0.

b) A term might always have an explicit coefficient even if it is 1.

c) A sum might always contain a constant term even if it is 0.

Although these structural paddings increase the size of trivial polynomials, the absence of list terminators and operators more than makes up for the space in large polynomials where space is more important -- especially in environments where 0 and 1 are stored directly or uniquely. Also, the structural regularity afforded by such padding obviates the need for some tests.

2.4 Sparse versus Dense Representations

In sparse representations, degrees are explicitly represented so that terms having a coefficient of zero can be omitted. Thus using sparse distributed representation, the structural space for a polynomial with t nonzero terms in v variables is O(vt). Sparse recursive polynomial representation can also require this much space if the terms all have distinct degrees in the the main variable.

In dense representations, the degree is implied by the position of the corresponding term in a list or array. Thus, terms having a coefficient of zero must be included to maintain the proper position count. One of the fundamental teachings of computer algebra has been the following argument that the traditional dense representation is hopeless for sparse multivariate problems: For a polynomial of degree d in each of v variables, a list or vector of all its numeric coefficients is of length $v^{(d+1)}$ even if most of them are zero. As a plausible example, all polynomials of ninth degree in six variables would require storing one million coefficients even if only a few were nonzero. Besides the storage inefficiency, processing times would asymptotically grow with the number of stored coefficients rather than the number of nonzero coefficients.

However, this argument presumes a **distributed** representation as a single list or vector of all numeric coefficients. What has apparently not been realized before is that recursive form is applicable to dense representations, with a resulting efficiency that is quite acceptable even for very sparse problems. The basic reason is that an explicit zero coefficient can represent a whole subtree of zeros that would each be explicit in a distributed dense representation:

Suppose that we have a t-term polynomial of degree d in each of v variables x_1, x_2, ..., x_v: Stored in dense recursive form with x_v as the main variable, the structural space for the top-level polynomial is $\Theta(d)$. In addition to this, any term of the form $c*(x_1x_2...x_{v-1})^d x_v^k$ with numeric coefficient $c\neq 0$ and $0<=k<=d$ requires structural space $\Theta[(v-1)d]$. No term could require more, and any terms that share leading powers share some of their structural space. Consequently, the total structural space is $O[d+(v-1)td]$. Although much better than the distributed dense bound, this typically pessimistic recursive dense bound is about d times as large as the sparse bounds.

2.5 Included versus Extracted variables

With either the distributed or recursive form, we can represent the variables with their corresponding degrees everywhere throughout the data structure, or we can extract the variables as a separate list. By referring to each of its variables only once per polynomial, extraction may save significant space for large polynomials where space is important. However, in order to deduce the implied variable associated with each degree, terms may have to be padded with zero degrees for missing variables. At the expense of some extra tests, we can terminate a term with a domain coefficient as soon as all of the remaining variables have degree zero, but padding cannot be avoided for earlier variables having degree zero. Either way, the storage inefficiency is most extreme for **"lean"** polynomials having many variables, but few per term.

Recursive form offers an intermediate alternative of extracting the main variable at each level of recursion. This avoids the necessity of padding with zero degrees for variables that are missing from a term. The number of references to variables tends to be intermediate between that of fully included or fully extracted variables.

For distributed form, variable extraction leaves contiguous exponents, which can therefore be packed several per word. At the expense of machine-dependence and program complexity, this technique permits parallel addition of all the exponents in a word. Using a 1-bit zone between exponents, exponent overflow can be detected by masking out all but the overflow zones then seeing if this gives a 0 word.

Full extraction significantly complicates the corresponding polynomial arithmetic algorithms -- particularly addition: Cancellations may cause some of the variables to occur in a result only to degree zero. Consequently, to achieve canonicality it is necessary to check for this possibility then make another pass to wring out the superfluous padding if it occurs. Whether the checking is done in separate passes or via posting notice of nonzero degrees during the first pass, the end result is a rather unsavory piece of code that can be several times as slow as a single pass. Although multiplication and the first pass of addition can be done without a preliminary padding so that the operands have identical sets of variables, an alternative is perform such preliminary paddings as necessary in order to simplify the logic within the multiplication and addition routines.

2.6 Descending versus Ascending Order

In traditional polynomial division with remainder, for list storage it is most convenient to order the terms primarily by variables, and secondarily in order of decreasing degree. Ascending degree may be equally suitable for arrays, where access is equally convenient from either end.

Surprisingly, in the case of exact division or a divide test, ascending or descending order is equally suitable for list storage too. For example:

$$
\begin{array}{r|l}
 & 2 - 3x + 2x^2 \\
\hline
3 + x & 6 - 7x + 3x^2 + 2x^3 \\
 & 6 + 2x \\
 & \hline
 & \quad -9x \\
 & \quad -9x - 3x^2 \\
 & \quad \hline
 & \qquad 6x^2 \\
 & \qquad 6x^2 + 2x^3 \\
 & \qquad \hline
 & \qquad\qquad 0
\end{array}
$$

Also surprisingly, ascending or descending order is equally suitable for polynomial-remainder-sequence greatest-common-divisor algorithms in the case of list storage: To compute the next polynomial in the sequence, we can choose a linear combination that annihilates the constant term rather than the leading term, then divide out the resulting trivial monomial factor.

Rather than descending lexical order, ascending **total** order is more appropriate for some p-adic methods, which iteratively develop results as truncated power series of increasing total degree.

Ascending order is more suitable for addition of dense list representations, because alignment of similar terms is automatic without need of storing or computing an explicit degree.

For recursive representations that collect together all of the terms that are of zero degree in the main variable, ascending order reduces the sorting costs when adding two polynomials having different main variables: The desired slot occurs at the beginning of the main list rather than the end, avoiding traversal of the main list followed by construction of an entire replacement. However, speed differences between ascending and descending order seemed insignificant for my multiplication test cases, so results are reported here for only one of the two orderings with each data structure. Perhaps large addition tests would reveal significant differences.

3. ALGORITHMIC ALTERNATIVES & ISSUES

For operands of a given nontrivial size, multiplication and division tend to be significantly slower than addition, negation and subtraction. Moreover, division is typically based on a sequence of multiplications and subtractions. Also, multiplication algorithms offer more varied and significant alternatives than those for addition, negation, subtraction and division. Thus, it is various multiplication algorithms that I have chosen to test in conjunction with the various data structures:

In the case of different leading variables, we merely distribute one of the polynomials over the coefficients of the other, which entails no merging.

For similar leading variables, the most straightforward approach is to repeatedly form a partial product of one term from the multiplier with the multiplicand, adding each whole partial product to a running total. To reduce merging costs in the case of imbalanced operands, it is definitely worth interchanging operands if necessary so that the multiplier is no longer than the multiplicand. Also, if we let T and R represent the leading term and reductum of the multiplicand while letting t and r represent those of the multiplier, then the following recursive expansion (or its iterative equivalent) saves some time by excluding t*T from the addition merge:

$$[t, r] * [T, R] \longrightarrow [t*T, t*R + r*[T, R]]$$

At the expense of a rather complicated and lengthy routine, after forming the first partial product, generation and merging of subsequent partial-product terms can be interleaved using pointer redirection (eg: Lisp RPLACA and RPLACD). This technique reduces the consumption of new cells (eg: Lisp CONS) and the peak intermediate storage requirements.

For long enough lists, other methods are asymptotically attractive, as described by Johnson [1974], Horowitz [1975] and Klip [1979]. Of these, the tournament merge has the advantages of being easy to program and having attractive asymptotic speed for sparsely represented polynomials that are either sparse or dense.

Pointer redirection was fastest for all of the recursive representations: With typical results that have even 1000 fully-distributed terms, for recursive form the number of nodes in any one list rarely exceeds even 10. Thus, the lists are much too short for the more complicated asymptotically fast methods such as the tournament merge to excell. Moreover, although the tournament merge was slightly faster for a few of the largest examples using a distributed representation, the tournament merge consumed significantly more intermediate storage and therefore precluded problems that could be done using pointer redirection. Thus, all of the reported sparse test results are for multiplication using pointer redirection.

For the dense recursive representation, it is worth checking for a zero multiplier term before multiplying it by each term of the multiplicand. It is also worth trimming leading zero coefficients from the multiplicand, with a compensatory padding of the result.

Dense representations encourage the alternative of entirely computing the convolution for each coefficient of the result in turn. This avoids the somewhat elaborate book keeping associated with pointer redirection without incurring additional new-cell consumption. Although this convolution method is about 10% faster than pointer redirection for densely represented dense polynomials,

pointer redirection is significantly better for sparse problems. Consequently, the reported dense representation test results are for the pointer redirection method.

For very large dense polynomials, special-purpose modular and Fourier techniques are advantageous -- at least in the context of array rather than list storage.

4. SOFTWARE & HARDWARE ENVIRONMENT

The software environment consisted of the muSIMP-83^{tm} programming system, version 4.12, developed by the Soft Warehouse [1984]. The hardware environment was an IBM-PC computer with 256 kilobytes of RAM storage available for muSIMP together with its programs and data. This microcomputer uses an Intel 8088 processor running at 4.77 mhz. A clock that can be checked from programs gives times in hundredth's of seconds, but its true resolution is about a tenth of a second. In contrast to typical mainframe profiles, the most notable disparity is that multiplication and division are much slower relative to other instructions.

muSIMP-83 is essentially a compact syntactically-sweetened Lisp interpreter, with features intended to speed the interpretation and facilitate direct implementation of computer algebra in the same environment that is offered to the user. The data types are uniquely-represented names, integers represented as arbitrary length signed-magnitude vectors of 16-bit binary words, and nodes consisting of two 16-bit pointers. Numbers of magnitude less than 2^{16} are stored more directly and uniquely via hashing. Garbage collection is of the compacting mark and sweep variety, with limited reallocation among the competing types of space when advantageous. With 256 kilobytes, collection and reallocation each average about 1.5 seconds. The appendix indicates speeds for basic operations.

In muSIMP, the value of a newly introduced name is automatically set to itself as is generally desired in algebra systems. Thus by making every final result be an integer or a name or a list beginning with a function name, the muSIMPs EVAL function automatically serves as an efficient machine-language main simplification function.

If given a noninteger argument, the arithmetic "+" function calls a trap that can be redefined to incorporate algebraic treatment rather than the default error break. The other arithmetic functions are treated similarly. This method avoids degradation of arithmetic speed by superposition of an algebra system on muSIMP. Moreover, this scheme permits even the polynomial routines themselves to use "+", "*", etc. on any operands that are names, integers or tagged lists of operands, without first checking to determine if the operands are integers. This technique avoids double checking, which shortens and speeds the program: Many if not most of the nodes of an expression tree have terminal leaves as descendants. Thus for many polynomial problems, both operands are often numeric.

5. SELECTED DATA STRUCTURES & ALGORITHMS

muSIMP-83 does not provide arrays, so the testing was limited to data structures using lists and dotted pairs. Arrays would most benefit distributed form and dense representations. At the expense of program complexity, arrays could almost half the structural data storage for large examples, where they might also reduce the time less dramatically.

Even without arrays, it was impractical to implement all possible combinations of the above data structures and algorithms. Nonetheless, the objective was to isolate the most decisive determinants of performance, then experiment with fine tuning the most promising major variants along avenues that were revealed as the implementation and testing proceeded. Here are the data structures that are compared, with domain ::= integer:

5.1 Distributed, Sparse, Variables Included:

```
expression  ::=  domain  |  variable  |  (POL term term ... term)
term  ::=  (coefficient power power ... power)
power  ::=  variable . degree
degree  ::=  {1, 2, ...}
```

5.2 Recursive, Sparse, Binary Cambridge Prefix:

```
expression  ::=  term  |  sum
monomial  ::=  variable  |  (^ variable degree)
term  ::=  domain  |  monomial  |  (* expression monomial)
sum  ::=  (+ term expression)
degree  ::=  {2, 3, ...}
```

5.3 Recursive, Sparse, N-ary+ Cambridge Prefix:

```
expression  ::=  term  |  sum
monomial  ::=  variable  |  (^ variable degree)
term  ::=  domain  |  monomial  |  (* expression monomial)
sum  ::=  (+ term term ... term)
degree  ::=  {2, 3, ...}
```

5.4 Recursive, Sparse, Variables Included:

```
expression  ::=  domain  |  variable  |  (POL poly)
poly  ::=  domain  |  term . poly
term  ::=  power . poly
power  ::=  variable . degree
degree  ::=  {1, 2, ...}
```

5.5 Recursive, Sparse, Variables Excluded:

```
expression  ::=  domain  |  variable
                 |  (POL (variable variable ... variable) . terms)
terms  ::=  (term term ... term)
term  ::=  degree . domain  |  degree . terms
degree  ::=  {0, 1, ...}
```

5.6 Recursive, Dense, Variables Part-way Out:

```
expression  ::=  domain  |  variable
             |  (POL variable expression expression ... expression)
```

Numerous secondary variations are of course possible. For example, the POL tags could be omitted from the above dense representation. In all cases, the ordering was lexical by variable. For the recursive dense and excluded-variable sparse representations, ordering was increasing degree. Otherwise, ordering was by decreasing degree.

6. TEST CASES & RESULTS

For testing it is desirable to have a set of problems that spans the commonly encountered types of problems while avoiding examples that are extremely uncharacteristic and therefore misleading. For the sake of manageable testing effort and succinct presentation, the number of test cases should be as few as possible consistent with the objective of coverage. The problems should be large enough to exercise the candidates heavily where efficiency matters, yet not so large as to try patience or enter a regime where site-dependent storage reclamation masks the time attributable to other aspects for any tested representation.

A major point of interest is the portion of polynomial multiplication time attributable to numeric arithmetic for coefficient multiplications and exponent additions: For operands that are fixed in all other respects, this portion will increase toward 100% as the numeric coefficients multiply in magnitude. Since the multiplications of nonzero coefficients are the same for all representations, the ratios of computing times will thus approach 1 as the numeric coefficients multiply in magnitude. Consequently, since an objective of this study is to determine extremes in these ratios, most of the test cases have nonzero coefficients that are all 1.

The portion of time attributable to numeric arithmetic is largest for dense univariate operands because they entail the most similar terms and less overhead for function calls, sorting, or list building. Thus, the effect of increasing coefficient magnitude is studied in the dense univariate case where it is most pronounced.

In accordance with these objectives, here first is the completely dense unit-coefficient polynomial of degree d in each of v variables:

$$D(x_1, x_2, \ldots, x_v; d) = \prod_{k=1}^{v} \sum_{j=0}^{d} x^j,$$

fully expanded and distributed. $x_1, x_2, \ldots, x_v$ represent distinct letters from the set {a, b, ..., z}.

Now, let

$$S\ (x_1,\ x_2,\ \ldots,\ x_v;\ d,\ t)$$

denote the set of all "**Sparse**" polynomials having $0 < t <= v^{d+1}$ terms taken from D $(x_1,\ x_2,\ \ldots,\ x_v;\ d)$, including the term $(x_1x_2\ldots x_v)^d$. Each element of this set has density t/v^{d+1}.

Polynomials having a large number of terms **and** variables can have only one or a few variables per term if the problem is to be manageable even by computer algebra. Thus, let

$$L\ (x_1,\ x_2,\ \ldots,\ x_v;\ d,\ t,\ m)$$

denote the set of all "**Lean**" polynomials having t terms with a maximum of m variables per term, taken from D $(x_1,\ x_2,\ \ldots,\ x_v;\ d)$, including the terms

$$x_1^{\,d},\ x_2^{\,d},\ \ldots.\ x_v^{\,d}.$$

In all cases, the variables are ordered alphabetically, with z being the most main variable, etc. In each case, the test was preceded by a forced garbage collection in order to reduce spurious variability.

Table 1 shows the computing times in seconds, including garbage collection and storage reallocation, for test cases that are either dense or drawn randomly with equal probability from a set of these sparse or lean polynomials. Where garbage collection or storage allocation occurred, their total **number** is displayed after the time. To prevent a significant influence of the 0.1 second timer resolution, times less than four seconds were inferred by timing ten repetitions of the problem in a loop, then dividing by ten. For the operands and results, Table 1 also indicates the number of terms and the number possible for completely dense polynomials of the same degree.

Table 2 shows the corresponding number of nodes and unique nodes in the result. The latter may depend in part in the amount of node sharing in the operands, which in turn depends upon how they were generated. Although the discussion below is based on the number of unique nodes, the operands were generally entered in recursively expanded form, so the number of unique nodes is often somewhat less than it would be if the operands were entered in factored form using intermediate variables to represent each distinct power of a variable. Names and small-magnitude numbers are stored uniquely in muSIMP, making their counts the same for all representations. Consequently, their counts are not reported.

It was not the objective to test algorithms for multinomial expansion, so problems with identical operands were entered as a product rather than a square. Moreover, problems having different operands were done in both orders, with the tables indicating the larger results. Although all but the Binary Cambridge Prefix implementation interchanged operands if the multiplier was "longer" than the multiplicand in an easily measured sense, remaining disparities associated with operand order were occasionally dramatic.

Table 1: Computing Times

Case	Description	OPERANDS Tms	OPERANDS PsbTms	RESULT Tms	RESULT PsbTms	Dist	SECONDS Bnry	SECONDS Nary	SECONDS VIn	SECONDS Vout	SECONDS Dense
1	1 var dense	49	49	97	97	14.1	62-3	58.8	32.4	18.5	3
2	with cfs 3 & 5	49	49	97	97	14.5	81-6	60.6	34	18.9	3.3
3	with 2-bigit cfs	49	49	97	97	19-2	83-6	65-2	38-2	23-2	8-2
4	with 5-bigit cfs	49	49	97	97	23-3	78	70-3	43-3	28-3	12-3
5	with imbalance	9,89	9,89	97	97	4.6	21-1	19.1	9.9	6	1.1
6	2 var dense	64	64	225	225	43.5	110-6	105-2	51.1	37.1	8.1
7	3 var dense	64	64	343	343	67.8	109-6	109-2	48.9	44.5	14.8
8	4 var dense	81	81	625	625	163-2	170-9	182-3	76.9	81.5	33.7
9	1 var sparse	8	49	45	97	0.72	1.6	2	0.97	0.57	0.55
10	2 var sparse	49	?56	703	?63	80	105-5	89.1	41.9	41.6	26.8
11	3 var sparse	25	7..	564	4913	29.2	32-2	28.5	13.1	16.2	22
12	4 var sparse	25	1296	602	14641	35.9	37-2	33.7	16.5	19.9	30
13	5 var sparse	25	7776	622	2E+05	36.1	40-2	42-2	23.2	26	47-2
14	5 var lean	49	7776	1849	2E+05	191	51-3	51	40.9	42.9	17.7
15	10 var lean	49	1E+06	2034	3E+08	225	38-2	44.4	44.9	49.8	15.8
16	1 dif var each	49	2401?	2401	2401	318	0.61	2.1	0.23	11.3	0.13
17	3 dif var each	25	3E5?	625	3E+05	41	9.7	16	4.6	11.8	9.5
18	3 var & 1	25,49	729&?	1045	41553	21.4	38-2	40.7	14.1	18	12
19	2 v mostly main	49	98	97	294	19.3	130-7	106-4	70-2	62.6	37.2
20	2 v mostly other	49	98	97	294	19.4	62-3	58.8	32.2	18.7	3.1

The objective of cases 1 through 4 is to study the portion of time consumed by arithmetic in the dense univariate case where this time is relatively most important: For case 1, both operands are

$$P_1(a) = D(a; 48) = a^{48} + a^{47} + \ldots + a + 1.$$

The muSIMP numeric multiply routines check for multiplication by 1 as a special case, so the arithmetic costs for this example are attributable to the addition of small integers.

Table 2: Structural Space Comparison

	Distrib.		Binary		N-ary+		Var-In		Var-Out		Dense	
Cas	Tot	Uniq	Tot	Uniq	Tot	Uniq	Tot	Uniq	Tot	Uniq	Tot	Uniq
1	387	292	858	858	668	527	290	243	196	196	99	99
2	387	292	861	861	671	530	290	242	196	196	99	99
3	387	292	861	861	671	530	290	242	196	196	99	99
4	387	292	861	861	671	530	290	242	196	196	99	99
5	387	371	858	738	668	644	290	202	196	196	99	99
6	1291	920	1962	1917	1544	1256	674	562	469	469	272	272
7	2451	1731	2895	2748	2309	1967	1028	857	754	754	513	513
8	5251	3707	5124	4749	4126	3658	1874	1562	1441	1441	1092	1092
9	181	167	327	311	226	209	137	130	93	93	99	99
10	4185	3873	5877	5742	4511	4248	2168	2056	1466	1466	989	986
11	4461	3915	4887	4506	3989	4506	2276	1869	1581	1581	2956	2751
12	5901	5018	6591	5778	5815	4978	3308	2774	2271	2271	5127	4155
13	7335	6289	9765	8475	9069	7748	5234	4531	3342	3342	9248	7502
14	14309	8316	12279	8010	9449	5693	5825	3430	4261	4261	5840	3112
15	16061	8373	13491	7923	10582	5782	6640	3063	5572	5572	7393	3998
16	14211	7107	14394	291	9692	290	7202	196	4855	4854	2550	102
17	7751	3815	9915	5145	9215	4544	4661	1810	3704	3703	9137	5290
18	7753	4956	8319	5889	6842	4381	3980	2126	2844	2843	4277	2729
19	581	533	1437	1437	1247	1106	581	533	392	392	584	578
20	581	485	864	861	674	533	293	245	199	199	104	104

For case 2, one operand is 3 $P_1(a)$ while the other is 5 $P_1(a)$ to test the influence of multiplying small-magnitude non-unit operand coefficients. The minimum increase over case 1 is 0.3 seconds, suggesting that the numeric arithmetic is consuming less than this for case 1. Thus, despite the relatively slow processor multiplication, non-unit small-number arithmetic consumes only slightly more than 10% of the time for the lowest-overhead dense recursive representation. The greater increase in time between cases 1 and 2 for the Cambridge Prefix representations can be explained by the fact

that non-unit coefficients significantly increase their operand complexity, hence the non-arithmetic overhead of dealing with them.

In case 3, the operands are $b = 2^{16}$ times as large as for case 2, making the 6-digit operand coefficients all require 2-bigit bignums. Subtracting an estimated 1.5 seconds for each indicated garbage collections and reallocations, the switch to bignum arithmetic appears to have changed the arithmetic cost from about 0.4 seconds to about 2.1 seconds. This is still a minority portion of the time even for the lowest overhead dense recursive representation, which requires about 2.9 seconds for non-arithmetic overhead.

In case 4, the operands are b^4 times as large as for case 2, making the 20-digit operand coefficients all require 5-bigit bignums. Subtracting for the garbage collections and reallocations, the arithmetic uses about 4.6 seconds, which is a minority portion for all but the most efficient dense recursive representation.

The results for cases 1 through 4 indicate that that arithmetic portions are surprisingly small considering the relatively slow processor multiply speed. As expected, dense recursive representation is significantly better than any of the others for these dense univariate examples. The relative advantage is greatest for unit operand coefficients where the arithmetic costs are least, but even for operands having 20-digit operands, the dense recursive representation is about twice as fast as its nearest competitor.

Case 5 tests the imbalanced dense univariate operands that give a result of the same degree as for case 1:

$$P_5 = D\ (a;\ 8) = a^8 + a^7 + \ldots + a + 1,$$

$$Q_5 = D\ (a;\ 88) = a^{88} + a^{87} + \ldots + a + 1,$$

Cases 6 through 8 continue the study of balanced dense operands having unit coefficients, with both operands being one of the following, expanded:

$$P_6 = D\ (a,\ b;\ 7) = (a^7 + a^6 + \ldots + a + 1)\ (b^7 + b^6 + \ldots + b + 1),$$

$$P_7 = D(a,b,c;3) = (a^3 + a^2 + a + 1)(b^3 + b^2 + b + 1)(c^3 + c^2 + c + 1),$$

$$P_8 = D(a,b,c,d;2) = (a^2 + a + 1)\ (b^2 + b + 1)\ (c^2 + c + 1)\ (d^2 + d + 1),$$

As expected, dense recursive representation is consistently fastest and most compact by a good margin, though the performance ratios are less dramatic with increasing numbers of variables.

Cases 9 through 13 have increasing sparsity number of variables:

$P_9, Q_9 \in S\ (a;\ 48,\ 8)$:

$$P_9 = a^2 + a^6 + a^{10} + a^{11} + a^{21} + a^{33} + a^{40} + a^{48},$$

$$Q_9 = 1 + a + a^6 + a^{10} + a^{21} + a^{28} + a^{44} + a^{48}.$$

$P_{10}, Q_{10} \in S(a, b; 15, 49)$:

$$P_{10} = (a^{15} + a^{13} + a^{10} + a^4)\, b^{15} + (a^9 + a^6 + a^4 + a)\, b^{14} + (a^{14} + a^{12} + a^2)\, b^{13} + (a^{11} + a^9 + a^4)\, b^{12} + (a^{10} + a^5 + a^4 + a^3 + a^2 + 1)\, b^{11} + (a^{15} + a^8)\, b^{10} + a^{12}\, b^9 + (a^{12} + a^{10} + a^7 + a^4)\, b^8 + (a^9 + a^6 + a^3 + a\hat{}2 + a)\, b^7 + (a^{13} + a^3)\, b^6 + (a^9 + 1)\, b^5 + (a^8 + a^5 + a^3)\, b^4 + (a^8 + a^7 + a^4 + a^2)\, b^3 + a^{15}\, b^2 + (a^8 + a^4)\, b + a^{12} + a^2 + a,$$

$$Q_{10} = (a^{15} + a^{14} + a^5)\, b^{15} + (a^{14} + a^{11} + a^7 + 1)\, b^{14} + (a^{12} + a^7 + a^3 + a^2)\, b^{13} + a^3\, b^{12} + (a^{15} + a^7 + a)\, b^{11} + (a^{12} + a^{10} + a^5 + a^4 + a^3)\, b^{10} + (a^{13} + a^5 + a^3)\, b^9 + (a^{13} + a^9 + a^7 + a^3)\, b^8 + b^7 + (a^9 + a^7 + a^2)\, b^6 + (a^{15} + a^{14} + a^9 + a^7 + a)\, b^5 + (a^8 + a^7 + a^4 + a)\, b^4 + (a^7 + 1)\, b^3 + (a^5 + a^3)\, b^2 + a^{13}\, b + a^{12} + a^{10} + a^8 + a^7.$$

$P_{11}, Q_{11} \in S(a, b, c; 8, 25)$:

$$P_{11}(a,b,c) = (a^8\, b^8 + a^5\, b^6)\, c^8 + (a^3\, b^5 + a^6\, b^2 + a^5\, b)\, c^7 + (a^6\, b^7 + a^2\, b^4 + a^6\, b^3 + a\, b)\, c^6 + a^3\, b^7\, c^5 + [a\, b^6 + (a^8 + a^5)\, b^5 + b^2]\, c^4 + a^5\, b^6\, c^3 + (a^7\, b^8 + a^3\, b^4 + a\, b^2 + a^7\, b)\, c + b^7 + (a^4 + 1)\, b^6 + a^7\, b^4 + (a^5 + 1)\, b^2,$$

$$Q_{11}(a,b,c) = [(a^8 + a^3)\, b^8 + a^5\, b^6 + a^2\, b^5 + a^3\, b^4]\, c^8 + (a^3\, b^8 + a^8\, b^7)\, c^7 + (a\, b^8 + b^6 + a^4)\, c^6 + (a\, b^3 + a^3\, b)\, c^5 + a\, b^5\, c^4 + (a^2\, b^4 + a^3\, b^2 + a^3 + 1)\, c^3 + (a^2\, b^4 + a^4\, b^3 + a^6\, b)\, c^2 + (a\, b^5 + a^5)\, c + a^6\, b^6 + a\, b^3 + a^4.$$

$P_{12}, Q_{12} \in S(a, b, c, d; 5; 25)$:

$$P_{12} = (a^5\, b^5\, c^5 + b^2\, c^4 + (a^3\, b^5 + a^3\, b^2 + a^5)\, c^2)\, d^5 + (a^2\, c^3 + b\, c^2)\, d^4 + [a^5\, b^3\, c^2 + (a\, b^3 + a^5\, b)\, c + a^5\, b^4 + a^4\, b^2]\, d^3 + [(a^3\, b^4 + a^2)\, c^5 + a^5\, b^5\, c^3 + a^2\, b^3\, c^2 + a^5\, b\, c]\, d^2 + (a^3\, b^4\, c^4 + b^4\, c^2 + a\, c)\, d + a^4\, c^5 + a\, b\, c^4 + (a^5\, b^4 + a^5)\, c^2 + a^5\, b\, c,$$

$$Q_{12} = [(a^5\, b^5 + 1)\, c^5 + a^2\, b^3\, c^4 + (a\, b^5 + b)\, c]\, d^5 + (a^4\, b\, c^3 + a^2\, b\, c + a^3\, b^3)\, d^4 + (a^4\, b^3\, c^4 + a^5\, b^5\, c^3 + a^5\, b^4\, c^2 + a^2\, b^5 + a)\, d^3 + [b^5\, c^4 + a^3\, b^5\, c^3 + (a\, b^3 + a^4)\, c^2 + a^4\, b^2\, c + b^4]\, d^2 + [(a^3\, b^3 + a^3\, b^2)\, c^4 + b^2\, c^2]\, d + a\, b^5\, c^5 + b\, c^3 + a^4\, b.$$

$P_{13}, Q_{13} \in S(a, b, c, d, e; 5, 25)$:

$$P_{13} = [(a^5\, b^5\, c^5 + a^5\, c^3)\, d^5 + a^2\, b\, c^5\, d^4 + a^3\, b^4\, c\, d]\, e^5 + \{[a^5\, b\, c^4 + b^2\, c^2]\, d^5 + a\, c\, d^4 + [(a^4 + a^2)\, c^4 + a\, b^5\, c]\, d^2\}\, e^4 + [(b^3\, c^3 + a^4\, b^5)\, d^5 + a\, b^2\, c^3\, d^4 + b^5\, d^3]\, e^3 + (a^3\, b^2\, c^5\, d^5 + a^2\, b^4\, c^3\, d^3 + a^5\, b^5\, c\, d^2)\, e^2 + [b^2\, c^5\, d^5 + a^2\, b^2\, c^5\, d^3 + (c^5 + a^4\, c^3)\, d^2 + b^5]\, e + a^4\, b^3\, c^4\, d^5 + (a^5\, b^4\, c^3 + a\, b\, c^2)\, d^4,$$

Q_{13} = [$(a^5\ b^5\ c^5 + a^2\ b^3\ c^4)\ d^5 + (a^2\ b^3\ c^2 + a^3\ b^2)\ d^4 + a^5\ b^5\ c\ d^3$] e^5 +
$((a^2\ b^2\ c^4 + a\ b\ c^3)d^5 + b^4\ d^4 + b^3\ c^2\ d^3 + (a^4\ b^5\ c^3 + a\ b^2c)d^2 + b^5\ c)e^4$
$+ (a^2\ b^5\ c^4\ d + a\ b^2\ c)\ e^3 + a^4\ b^5\ c^5\ d^2\ e^2$ +
$((b^2\ c^3 + a^4\ b^2)d^4 + a^2\ b^4\ c^3\ d^2 + (a\ b^5\ c^5 + a\ b^4\ c)d + a^5\ b^2\ c^4 + a^4b^4c)e$
$+ (a^2\ b^5\ c^5 + c^4)\ d^3 + a^5\ c^5$.

The 8/48 density of the case 9 univariate operands is at the crossover point below which the fastest sparse representation is faster than dense recursive. The 45/97 result density is the approximate crossover point for the resulting number of nodes too. In contrast, dense recursive is nearly twice as fast and significantly more compact than its nearest competitor for the case 10 bivariate example ith operand density 49/256 and result density is 703/963.

Implicit sparse recursive representations are most efficient for the less dense cases 11 through 13, with extracted variables slightly more compact, but included variables was slightly faster. The surprise is that dense recursive representation used only about twice as much space and time even though the operand density was as low as 25/7776 and the result density was as low as $622/(2\ 10^5)$.

Cases 14 and 15 study the effect of random lean operands:

P_{14}, Q_{14} ϵ L (a, b, c, d, e; 5, 49, 2):

$P_{14} = (b^3 + b^2 + a^2 + 1)\ e^5 + (c^5 + b^3 + b + a^4 + a^3 + 1)\ e^4$
$+ (a^3 + 1)\ e^3 + (d^4 + b + a^5)\ e^2$
$+ (d^5 + d^4 + d^2 + c^4 + b^5 + b^3 + a^5 + 1)\ e$
$+ (b^3 + 1)\ d^5 + b^5\ d^4 + (c^3 + b^5 + b^3)\ d^3 + (c^2 + b^4 + 1)\ d^2$
$+ (c^5 + c)\ d + c^5 + (b + a^5)\ c^3 + (b^4 + b^2 + 1)\ c^2 + (b^4 + b^3)\ c$
$+ (a^2 + 1)\ b^5 + b + a^5 + a^2 + a + 1,$

$Q_{14} = (d^2 + c^5 + c^2 + b^5 + 1)\ e^5 + d^5\ e^4 + c^3\ e^3 + e^2 + (b^4 + 1)\ e$
$+(b^5 + b^4 + b + 1)\ d^5 + (a + 1)\ d^4 + (a + 1)\ d^3$
$+ (c^3 + b^4 + b + 1)d^2 + (c^4 + c^2)\ d + (b^5 + a^5 + a + 1)\ c^5$
$+ (b^4 + a^5 + a + 1)\ c^4 + b^4\ c^3 + (a^3 + 1)\ c^2 + (b^5 + a^3 + a + 1)\ c$
$+ b^5 + b^4 + b^3 + (a^4 + a)\ b^2 + b + a^5 + a^2 + a + 1$ \$

P_{15}, Q_{15} ϵ L (a, b, c, d, e, f, g, h, i, j; 3, 49, 2):

$P_{15} = (f^2 + c + 1)\ j^3 + (i + f + b^3 + 1)\ j^2 + (f^2 + 1)\ i^3 + c^3\ i^2$
$+ (e^3 + c^2)\ i + (d^2 + 1)\ h^3 + (c + 1)\ h^2 + (b^2 + 1)\ h + (b^3 + 1)\ g^3$
$+ (f + b + a)\ g^2 + f^3 + (b + 1)\ f^2 + (b^2 + b + a + 1)\ f$
$+ (d^3 + a^2 + 1)\ e^3 + (c + 1)\ e^2 + (d^3 + a^2 + 1)\ e + d^3 + a\ d^2 + c^3$
$+ c^2 + c + b^3 + b^2 + a^3 + a^2 + a + 1,$

$Q_{15} = (i^3 + e^3 + e + 1)\ j^3 + a^2\ j^2 + (g^3 + e^3 + 1)\ j + (a^3 + 1)\ i^3$
$+ (d^2 + a^2)\ i^2 + (g^3 + f^2)\ i + (f^3 + c^2 + 1)\ h^3 + (a^3 + 1)\ h^2$
$+ (f + c)\ h + g^3 + g^2 + b\ g + (e + 1)\ f^3 + (e + d^2)\ f^2 + d\ f$
$+ (c + 1)\ e^3 + (c + a^3 + 1)\ e^2 + d^3 + (b^3 + a^3 + 1)\ d^2$
$+ (b^3 + 1)\ c^3 + (b^3 + a^3 + 1)\ c^2 + c + b^3 + a^3 + a^2 + a + 1.$

For both examples, dense recursive representation is about twice as fast and approximately the same compactness as the nearest competitor despite the low operand densities of 49/7776 and $49/10^6$ and the corresponding low result densities of $1849/(2\ 10^5)$ and $2034/(3\ 10^8)$. The reason that dense recursive representation fares so well for lean operands is that they tend to behave more nearly as a sequence of higher-density problems. An extreme case would be "piecewise dense univariate" operands such as

$$(a^d + a^{d-1} + \ldots + a) + (b^d + b^{d-1} + \ldots + b) + \ldots + 1.$$

Note how distributed representation performs very badly for lean examples because of the extensive distribution where so much recursive sharing is possible.

The purpose of cases 16 through 18 is to study the effects of operands having different variable sets. The question marks in the operand "possible terms" column warn that the entries are computed as if both operands had all variables. For case 16, both operands are dense univariate with different variables:

$$P_{16} = P_1(a) = a^{48} + a^{47} + \ldots + a + 1,$$

$$Q_{16} = P_1(b) = b^{48} + b^{47} + \ldots + b + 1.$$

The dense recursive representation is fastest and most compact. The sparse recursive representation with extracted variables performs poorly because of the different variables in the operands. Note how distributed representation also perform very badly for differing-variable examples such as this because of the extensive distribution where so much recursive sharing is possible. For this example, distributed representation was particularly sensitive to operand order despite the equal number of terms in both operands: $Q_{16}\ P_{16}$ required 17 seconds compared to 318 seconds for $P_{16}\ Q_{16}$.

Case 17 involves sparse trivariate polynomials with totally different variables that interleave in ordering:

$$P_{17} = P_{11}(a, c, e),$$

$$Q_{17} = Q_{11}(b, d, f).$$

Sparse recursive representation is fastest with included variables but most compact with excluded variables. Dense recursive representation requires somewhat more than twice as much time and space.

Case 18 involves a dense univariate polynomial and a sparse polynomial in this variable together with one that orders earlier and one that orders later.

$$P_{18} = Q_{16} = b^{48} + b^{47} + \ldots + b + 1,$$

$$Q_{18} = P_{11}(a, b, c).$$

The relative ordering of variables is known to have a significant effect on the efficiency of recursive representations, so cases 19 and 20 are a study of this effect: For case 19 both operands are the following polynomial having many terms in its main variable, with coefficients that are a monomial in another variable:

$$P_{19}(a, b) = a\ P_1(b) = a * (b^{48} + b^{47} + \ldots + b + 1),$$

expanded. For case 20, both operands are the same polynomial with the opposite ordering of variables, giving a trivial monomial in the main variable with a coefficient that has many terms in the secondary variable:

$$P_{20}(a, b) = P_{19}(b, a),$$

expanded. The different orderings have negligible effect on the distributed representation, but the recursive representations are more efficient when the order gives fewer terms in the main variable. The space and time savings are about a factor of two for the sparse recursive representations, but the time savings are a factor of ten for the dense recursive representation. However, even the less favorable ordering gives a dense time that is only about twice its nearest competitor.

Taken as a whole, the tables reveal several other tendencies:

a) Cambridge Prefix is usually slowest, with no decisive difference in the binary and N-ary speeds. The binary variant is usually least compact while tending to require more garbage collections due to the lack of a pointer redirection multiply.

b) Among the four sparse recursive representations, implicit operators typically about halve the space relative to Cambridge Prefix. The ratio of computing times is more variable, but implicit operators most often about halve the time too.

7. CONCLUSIONS

As indicated earlier, this study is a preliminary report. Before publication in its final form, I would like to receive reactions to the test case selection, and perhaps include additional data representations. For example, I could include Cambridge Prefix or extracted-variable representations of distributed form. It would also be helpful to learn the performance of other algebra systems on these examples and those of the appendix in order to determine their applicability to other environments.

Subject to these caveats, the tests suggest that with a general purpose list-oriented implementation in a LISP-like environment:

a) For examples that are lean or involve different variable sets, distributed form can be substantially less efficient than recursive form.

b) For sparse recursive forms, implicit operators typically halve the data space and computing time. In my value judgement, this is not enough efficiency advantage to preclude the more flexible Cambridge Prefix without further study in the context of specific design goals.

c) Binary Cambridge is nearly as efficient as N-ary; so if one selects Cambridge Prefix, the binary variant may be preferable on the basis of programming ease.

d) For sparse recursive representations with implicit operators, the performance of extracted versus included variables are close enough so that I prefer the included variables on the basis of programming ease.

e) Although dense recursive representation is not uniformly better than the best sparse recursive representation, it is efficient enough on even quite sparse problems to warrant strong consideration because of its simplicity and its efficiency for dense problems.

8. REFERENCES

ACM SIGSAM Bulletin [1972], Number 24, October, entire issue.

Fateman, R.J. [1974]: "On the Multiplication of Poisson Series", Celestial Mechanics 10, pp. 243-247.

Goto, E. & Kanada, Y. [1976]: "Hashing Lemmas on Time Complexities with Applications to Formula Manipulation", Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, pp. 154-158.

Gustavson, F. & Yun, D.Y.Y. [1976]: "Arithmetic Complexity of Unordered Sparse Polynomials", Proceedings of the 1976 ACM Symposium on Symbolic and Algebraic Computation, pp 149-153.

Horowitz, E. [1975]: "A Sorting Algorithm for Polynomial Multiplication", Journal of the ACM, Volume 22, Number 4, pp. 450-462.

Johnson, S.C. [1974]: "Sparse Polynomial arithmetic", Proceedings of Eurosam '74, pp. 63-71 (ACM SIGSAM Bulletin, Volume 8, Number 3, August, Issue Number 31).

Klip, D.A. [1979], "New Algorithms for Polynomial Multiplication", SIAM J. Computing, Vol. 8, Number 3, August, pp. 326-343.

The Soft Warehouse, [1984]: "muSIMP/muMATH Reference Manual", P.O. Box 11174, Honolulu, Hawaii 96828

9. APPENDIX: FUNDAMENTAL muSIMP TIMING TESTS

9.1 Dialog listing

```
? SUBROUTINE SecondsTimes10 (Expression,
    % Local: % RECLAIM),
    RECLAIM: TRUE,  % prevent display of garbage collection statistics %
    RECLAIM (),     % force garbage collection for quasi-repeatability %
    RECLAIM: FALSE, % force screen display of garbage collection statistics %
    TIME (TRUE),    % reset timer %
    EVAL (Expression),
    QUOTIENT (TIME () + 5, 10)    % return time in tenths of seconds %
  ENDSUB $

? FUNCTION CountDown (n),  % loop with subtract, assign & recognizer %
    LOOP
      WHEN NEGATIVE (n: n - 1), EXIT,
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (CountDown (10000));
@: 122          % <— includes 1 reallocation & 2 garbage collections %

? FUNCTION MakeList (n,     % make a list of n FALSEs %
    % Local: % ANS),  % unused parameters are initialized to FALSE %
    LOOP
      WHEN NEGATIVE (n: n - 1), ANS EXIT,
      PUSH (FALSE, ANS),
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (List10000: MakeList (10000));
@: 156          % <— includes 1 reallocation & 1 garbage collection %

? FUNCTION LoopWalk (List),  % walk List with loop %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (LoopWalk (List10000)) % interpreted list traversal %;
@: 31

? SecondsTimes10 (LASTNODE (List10000)) % vs built-in list traversal %;
@: 1

? SecondsTimes10 (REVERSE (List10000))  % built-in trav plus building %;
@: 8

? FUNCTION FunCall (List), % list trav & call trivial built-in function %
    LOOP WHEN ATOM (List), EXIT,
         POP (List),
         IDENTITY (), ENDLOOP,
  ENDFUN $
```

```
? SecondsTimes10 (FunCall (List10000));
@: 41

? FUNCTION Triv0 (), % Most trivial possible interpreted function %
  ENDFUN $

? FUNCTION Triv0Test (List), % list walking plus Triv0 %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
      Triv0 (),
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (Triv0Test (List10000));
@: 59

? FUNCTION Triv4 (Arg1, Arg2, Arg3, Arg4), % Trivial except 4 arguments %
  ENDFUN $

? FUNCTION Triv4Test (List), % list walking plus Triv4 %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
      Triv4 (1, 2, 3, 4),
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (Triv4Test (List10000));
@: 107

? ThreeDigits: 123 $

? FUNCTION SmallTimesSmallGivingSmall (List), % SmallNum multiplication %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
      321 ThreeDigits,
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (SmallTimesSmallGivingSmall (List10000));
@: 78              % <— smallnum hash avoids garbage collection %

? FiveDigits: 12345 $

? FUNCTION SmallTimesSmallGivingBig (List), % SmallNum mult, 2-Bigit ans %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
      54321 FiveDigits,
    ENDLOOP,
  ENDFUN $

? List10000: FALSE $
```

```
? List1000: MakeList (1000) $

? SecondsTimes10 (SmallTimesSmallGivingBig (List1000));
@: 13

? SeventyNineDigits:
12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
$

? FUNCTION SmallTimesBig (List), % SmallNum times 79 digit bignum %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
      12345 SeventyNineDigits,
    ENDLOOP,
  ENDFUN $

? List100: MakeList (100) $

? SecondsTimes10 (SmallTimesBig (List100));
@: 1

? FUNCTION BigTimesBig (List), % Product of two 79-digit bignums %
    LOOP
      WHEN ATOM (List), EXIT,
      POP (List),
98765432109876543210987654321098765432109876543210987654321098765432109876543210987654321
      SeventyNineDigits,
    ENDLOOP,
  ENDFUN $

? SecondsTimes10 (BigTimesBig (List100));
@: 18
```

9.2 Implications:

a) Entering and leaving the most trivial machine-language function requires 0.1 ms., versus 0.3 ms. + 0.1 ms. per argument for a user-defined function.

b) Interpreting multiplication of two smallnums requires 0.5 ms. for a smallnum result, versus 1.0 ms. for a bignum result.

c) Interpreting multiplication of a smallnum by a 79-digit number requires 10 ms., versus 180 ms. for two 79-digit numbers.

d) For smallnum n, interpreting "WHEN NEGATIVE (n: n-1), EXIT" requires 0.9 ms.

e) Interpreting "PUSH (FALSE, ANS)" requires 0.3 ms., versus perhaps 0.06 ms. if hand compiled as in REVERSE.

f) Interpreting "WHEN ATOM (List), EXIT, POP (List)" requires 0.3 ms., versus perhaps 0.01 ms. if hand compiled as in LASTNODE.

MEASURING THE PERFORMANCE OF A COMPUTATIONAL PHYSICS ENVIRONMENT

Robert H. Berman
Massachusetts Institute of Technology
Cambridge, MA

Abstract

Now that MACSYMA and other symbolic computing programs (e.g., REDUCE, SMP) are becoming commonly available on many different machines, it is of special interest to applied users to compare their performance. In this paper, I describe the results of three benchmarks that characterize symbolic computing and numeric performance on several machines including supercomputers (CRAY-1), Lisp machines (Symbolics 3600), and DEC machines (VAX, KL-10, 2060). The viewpoint I adopt for these tests is that of an applied user, not a hardware designer or software specialist. Thus, the benchmarks measure an average or aggregate performance on a complex set of tasks, not a finely tuned, simplistic test. This is the more realistic situation that a user, rather than a system implementor, experiences when performance is measured.

The first two benchmarks consider the raw numeric performance of the machines on a mixed set of memory addressing needs and floating point arithmetic calculations. Special attention to supercomputer hardware performance is discussed. The third test measures the symbolic computing performance on a problem designed to include many of the unique and desirable features of symbolic computing – exact, big number arithmetic, differentiation, recursion, summation, and large expression manipulation. Part of this final test is totally subjective and explores capabilities for error checking by numerical and graphical methods, and manipulating large expressions into FORTRAN for efficient or stable numerical evaluation.

1. INTRODUCTION

It is the purpose of this paper to investigate several issues regarding the performance of symbolic computing environments. One such environment, consisting of a symbolic computing machine integrated with an efficient numerical machine (array processor), was described at the 2nd MACSYMA Users conference (Berman and Kulp 1979). Several attempts at implementing this configuration, consisting of a Lisp machine (CADR), VAX 780 or KL-10 at M.I.T., network technology, and a supercomputer (CRAY) at the National Magnetic Fusion Energy Computing Center, have been in existence for some time. With the wide availability of VAX class machines, and the efforts to provide supercomputer services to university clients, such computational environments will become increasingly common. Indeed, other configurations involving primarily VAX or Lisp machines and supercomputers are now (middle of 1984) more generally available at numerous sites.

While it is conventional to measure the performance of numerical hardware by single numbers (e.g., cycle time), it is clear that a single statistic can be inadequate because of differences in hardware (pipelining, storage interleaving). Furthermore, parallelism, dynamic address translation, register-to-register instructions, memory, and instruction lookahead all affect performance. In addition to the speed of the central processing unit, the execution efficiency of the symbolic computation, the execution efficiency of the supporting system, and the algorithm all are factors (Jenks and Griesmer 1972). In spite of the danger of having too many variables to control, the average run time in solving a problem is useful to applied users to meter their costs. In several cases, the result is surprising in that the speed of the computer as measured in solving realistic problems is different than the theoretical hardware speed or the speed based on operation counts (See also Hockney 1978).

This paper examines three problems relevant to symbolic calculations now commonly available on superminicomputers like the VAX 780 (REDUCE, SMP, MACSYMA), specialized machines like Symbolics 3600, and supercomputers like the CRAY 1. The first two tests make an effort to calibrate the raw numerical performance of each machine. The first test is a table lookup, a calculation which involves capabilities of indirect addressing and memory

speeds. It is described in Section 3.1. The second test is a fast Fourier transform, which emphasizes floating point numerical performance, and is described in Section 3.2. The third test is a symbolic calculation of certain polynomials involving exact arithmetic, differentiation and recursion and is described in Section 4.

In describing the results of the symbolic calculation, several interesting and desirable features of a symbolic system are illustrated. These include: (1), error checking with word processor or by graphical methods; and (2), some optimization for stable numerical evaluation and generation of run-time efficient FORTRAN code for intensive numerical evaluation at later times or for other machines. In particular, a distinction arises between optimization for stable numerical evaluation and for run-time speed. This is especially true in two cases. First, for machines like the KL-10, VAX or Symbolics 3600, numerical evaluation of a symbolic calculation can be very time consuming even with translated or compiled code. Second, when the result of the symbolic calculation is a FORTRAN production that can run for hours of CRAY time (*e.g.*, Berman and Leboeuf 1984 or Brunel *et. al.* 1980), efficient code can be as important as correct code.

The principal results of the these tests in Sections 3.3 and 4.5 can be summarized as follows:

A. Supercomputers, such as the CRAY, seem to limited by memory access speeds in their performance for symbolic computing and do not perform significantly faster than super-minicomputers, such as the VAX 780. On the other hand, supercomputer performance for floating point computing can be as much as 1000 times faster for primarily numerical performance. Thus, symbolic manipulation programs may need substantial software improvements to take advanatge of the supercomputer hardware.

B. Specialized symbolic computers, such as the Symbolics 3600 Lisp Machine, emphasize single user interactivity, large address space, and an integrated environment for graphics, symbolic computing and networking. By linking such symbolic computing machines to large numerical supercomputers with existing network technology, a very impressive assault on a wide variety scientific problems with computers is now feasible. Lack of interactivity is a severe, if not fatal, handicap to this end.

C. Low cost virtual memory symbolic machines functioning as front-end workstations to large numerical machines provide an effective expenditure of resources to improve scientific productivity. This is feasible because large numerical (e.g., FORTRAN) codes can be more automatically generated and conveniently manipulated in an interactive symbolic machine while network technology makes intermachine communication practical. Cost-performance issues are addressed in Section 5.

D. In considering the automatic generation of code, there is an important distinction in optimizing expressions for stable numerical evaluation and for run time efficiency. This is particularly important when, as illustrated in Section 4.4, there are nonintuitive, nonlinear transformations of code that can result in more FORTRAN statements and operations, but with substantial improvements in performance on a CRAY-class machine. Operation count is not necessarily an adequate figure of merit for comparing algorithms in numerical analysis, especially when nonlinear improvemnts in execution speed are possible in the hardware (See Section 4.3).

This paper is organized as follows. Section 2 describes a number of issues concerning bias in comparing software and hardware performance. Section 3 describes the first two numerical tests for which results are presented in Section 3.3. The symbolic benchmark is formulated in Sections 4.1 and 4.2. Section 4.3 discusses several issues associated with automatic optimizing of code for stable numerical evaluation and for efficient execution speed, especially on supercomputers. Section 4.4 describes some novel results concerning error checking of symbolic calculations. Section 4.5 presents the results of the symbolic benchmark. Section 5 describes certain cost and performance conclusions based on the result of these studies. Appendices A, B, and C contain a number of technical results dealing with the calculation of results and the automatic generation of code.

2. EXPLANATION OF BIAS

This paper is concerned with the results of certain benchmarks concerning the capabilities of several machines and symbolic computing packages. The benchmarks were performed during the period December, 1981, to May, 1984.

Several remarks about the subjective nature and methodology of software evaluation are in order. There are generically at least three important difficulties: (1), the rapid obsolescence of today's software or choice of fashionable problem; (2), the elusiveness of objective criteria and reviewers, and (3), the cost (opportunity) for evaluation.

Perhaps the most difficult is the problem of objectivity. To integrate information for problems running over a number of different machines requires skill, expertise, knowledge, persistence, dedication, and, perhaps, a small bit of wisdom. In particular, fairness, requires judgment, and no set of standards can completely replace that.

Even if software/hardware developers can accurately forecast the capabilities of their products, there remains the question of whether software products should be evaluated (and therefore compared) on the basis of where they are at present or where they expect to be at some future date. Each can be misleading in its own way.

If the developer can not be trusted to be objective, what about the experienced user? From such users, we might anticipate insight into the strength and weaknesses of the product. Yet, there is also a problem of bias not immediately apparent. The experienced users of X are likely to be one who use X because it is very well suited to their needs or who find X well suited to personal taste and convenience or have the opportunity to use X. Those who find Y easier to use or who need capabilities not found in X will not likely be experienced users of X. Consequently, they will not find their views (needs) reflected in a user-based review of X. This comment is particularly true if the seasoned users are nominated by develcpers of the packages themselves.

I attempt to reveal my prejudices here by explaining that I am an applied MACSYMA user. I am more interested in the aggregate or average performance of a non-trivial scientific

computing problem rather than a highly specialized exercise. My viewpoint is that of a computational physicist in the Magnetic Fusion Energy program, and an applied user, not a hardware specialist, or system implementor or a computer scientist. I have relatively easy access to VAX 780, KL-10 (MC), Lisp machines (CADR and Symbolics 3600) at M.I.T. and CRAY, CDC 7600 and KL-10 machines at the MFE Computing Center in Livermore, CA. Typically, I will ignore most language-dependent issues or operating system issues. Indeed, some differences in timing results could possibly be attributed to differences in object code that a high level language compiler produces. Hockney (1978) discusses certain compiler and execution efficiencies. One reason these execution efficiencies occur is the ability to "chain" intermediate results between vector registers (See Sectir 4.3). However, this paper will not discuss these differences in detail. In timing code, measurements were made by using the installation supplied CPU timer (*e.g.*, SHOWTIME:TRUE in MACSYMA).

To illustrate this viewpoint, I offer an example. The CRAY 1 supercomputer has a clock time of 12.5 ns and is theoretically capable of performing floating point calculations at the rate of 160 Mflops (millions of floating point operations per second) on certain calculations like scalar inner products. However, problems that involve computations at this rate are rarely met in practice. Most of the computing time in solving a problem is more likely to spent doing specialized calculations, *e.g.*, solving Poisson's equation, performing fast Fourier transforms, input/output, graphics, or special function evaluation. An applied user becomes more interested in the time it takes to complete *all* the phases of his computation, not only in the fastest part. Thus, a fairer measure of performance is an average (wall clock) measure. Indeed, this type of average or aggregate rate is used by experts to compare supercomputer performance over a wide class of problems. The fourteen "Livermore FORTRAN kernels" perform 20 Mflops over a wide class of problems (Fuss 1984) for CRAY-class machines rather than the theoretical rate.

In this spirit, the raw numeric computing performance of a machine is best described by numerical tests that represent problems that an "average" user might encounter and that exercise a variety of capabilities of the machine. In particular, supercomputers are characterized by their raw performance rates, their ability to do "number crunching" on scientific problems and their requirements for large memories, bandwidths and resources. The fast Fourier

transform (FFT) is a representative, nontrivial scientific calculation and has an algorithm that has been carefully studied (Bingham 1974). It can serve as a measure of raw floating point performance. Hockney (1978) has studied the FFT as a specific test to compare compiler and hardware performance over a wide class of machines. It is noted that the timing measurements in this paper may not be generally valid, even though the calculations are representative of typical scientific calculations. It would require a wider class of tests to reach completely general conclusions.

Measuring the symbolic computing performance of a machine is much more difficult because the nature of symbolic computing can be more complex than numeric computing. For example, in a numerical problem, the answer is clearly recognized, say, by a number or graph. In a symbolic calculation, the answer often does not exist. Indeed, the usefulness of the calculation may be the process of investigation, rather than a derived graph. The calculation may involve, for example, exact arbitrary precision arithmetic, differentiation, integration, algebraic or trigonometric simplification, special function expansion, pattern matching and simplification, Taylor series, or generation of FORTRAN. Or it may be only part of a longer, iterative process of finding insight. However, it is possible to say that a symbolic calculation involves manipulating possibly arbitrary objects in possibly arbitrary ways. The LISP feature of list processing and its implementation with indirect addressing capabilities are well suited for this arbitrary manipulation. Furthermore, large physical memory is preferable to virtual memory for execution efficiency. A table lookup calculation is a representative calculation and can serve as a measure of raw symbolic computing performance. Of course, such a test omits the importance of these other capabilities (*e.g.*, pattern matching) and the usefulness of a highly interactive, high bandwidth computing environment for symbolic computing.

There are important differences in hardware design when optimizing a machine for these two calculations. For example, on the CRAY supercomputer there are 64 buffer registers for (24-bit) addresses and 64 scalar registers for (64-bit) floating point numbers. These buffers lie between the main memory and the registers. There are also 8 address registers, 8 scalar registers and 8 vector registers. For scalar arithmetic, the times for selected operations are register load (125 ns), floating point add, (75 ns) floating point multiply (87 ns). For vector

(pipelined) arithmetic, the minimum time per element is 12.5 ns and the maximum theoretical rate is 160 Mflops. The calculation for table lookup cannot be vectorized and is limited by the speed of random access into memory (register loading) to perhaps some ten times slower than the maximum floating point vector rate. Other parts of a symbolic calculation, such as garbage collection, may not vectorize easily either. Finding ways of improving software to take advantage of supercomputer hardware is a special challenge to designers.

3. NUMERICAL BENCHMARKS

In this section, two benchmarks for numerical performance are described that do a table lookup test and a fast Fourier transform.

3.1 Table Lookup Test

The first test assumes that the bulk of a large symbolic computation performs operations for which a table lookup is representative. No explicit arithmetic is done in the inner loop. Thus, the timing should reflect the speed of memory references.

Comparison of two different symbolic computing systems that employ different algorithms and internal computing efficiencies (*e.g.*, garbage collection, implementation of indirect addressing features) will be affected differently. Here we will use the same algorithm on KL-10 and Lisp Machine (CADR), VAX 780, and CRAY. Code fragments in FORTRAN and MACSYMA follow.

FORTRAN Fragment # 1: Source for Table Lookup

```
      parameter (n=1000)
      integer h(n),j(n),k(n)
c
      do 10 i=1,n
         h(i)=0
         k(i)=i
10       j(i)=i+1
      h(n)=1
c
c...set running timer
c
      call second(t0)
c
c...  begin search
c
      i=1
20    if(h(k(i)).eq.1)goto 30
      h(k(i))=h(1)
      i=j(i)
      go to 20
30    continue
      call second(t1)
c
c...compute elapsed time
c
      t1=t1-t0
      write(6,910) n,t1
```

MACSYMA Fragment # 1: Source for Table Lookup

```
/* This experiment is designed to test
   list-tracing table lookup facilities. Note
   that the inner loop does not do arithmetic --
   only indirect addressing. The conjecture is
   that most "symbol-crunching" problems stress
   this behavior */

/* create arrays */
setup(n):=block([kk:n-1],modedeclare([kk,n],fixnum),
                    kill(h,j,k),
                    array([h,j,k],fixnum,kk),
                    for i:0 thru kk do
                        (k[i]:i,j[i]:i+1),
     h[kk]:1)$

/* the inner loop */
loop():=block([i],modedeclare([array(h,i,j,k)],fixnum),
        i:0,
        while(h[k[i]]#1) do
           (h[k[i]]:h[0],i:j[i]),
        print(" whew!! all done "))$

n:1000;
setup(n);
```

3.2 Fast Fourier Transform Test

As described earlier, a reasonable test of average performance is the time taken to perform a "meaningful" computation like a fast Fourier transform (see Hockney 1978). The FFT is a particularly good test because it exercises arithmetic speed as well as addressing and memory performance. Differences in the addressing unit design as well as memory show up in these comparisons.

This FFT test considers the arithmetic power of the machine. This is a complex function of the design of the machine architecture, memory cycles, arithmetic units, etc. The bulk of the calculation is explicit arithmetic in calculating fast Fourier transforms. We use three algorithms to compute FFTs. First, on KL-10, we use a radix-2 scalar FFT. On CRAY-1, we use a scalar radix-4 + 2 FFT in FORTRAN and a vectorized CRAY Assembler Language (CAL) radix-2 FFT that computes 64 FFTs simultaneously. In all cases, we calculate the complex transform of complex input for lengths of $N = 256$. On the VAX 780, we use the same scalar radix 4 + 2 algorithm. Even though this test uses different algorithms on different machines, it is representative of the experiences an average user will encounter.

3.3 Results for Table Lookup and FFT Tests

The results for the table lookup test are described in Table 1.

Table 1. Times in msec for Table Lookup

7600	CRAY	VAX 780	MC KL-10	Trans.	Comp.	Lisp Machine CADR	Trans.	Comp.
1.578	0.631	8	16310	919	53	54567	6233	217

The additional columns labeled Trans. (Translated) and Comp (Compiled) refer to using code translated or compiled into LISP from MACSYMA in the KL-10 or Lisp machine environment. There is an substantial gain in run-time efficiency by using the compiler or

translator facilities. Or rather, there is a substantial penalty for not using these facilities. The loop time for the CRAY-1 is consistent with memory speed of 125 ns rather than a clock speed of 12.5 ns, indicating the calculation is limited by random access into memory rather than by the speed of the arithmetic units. Of course, this comparison does not take into account any implementation specific issues regarding specialized address registers or FORTRAN compiler issues regarding indirect addressing.

On the basis of this indirect address test, the following estimates for relative performance can be inferred: 1 CRAY = 1×3 VAX 780 and 1 CRAY = 344× CADR.

The results of the FFT test are described in Table 2.

Table 2.Times for $N = 256$ FFT Calculations

Machine	CDC 7600	CRAY 1	VAX 780	DEC KL-10
Test	4+2 FFT		4+2 FFT	radix-2 FFT
Language	FORTRAN	FORTRAN	FORTRAN	ASSEMBLER
Scalar FFT	3	0.75	125	62
Language		CAL		
Vector FFT 64 FFTs	X	0.1132	X	X

Here, X indicates that the test is inapplicable.

The FFT times represent a test of combining memory and arithmetic performance. Note the vectorized FFT achieves a rate of 45.22 Mflops per FFT in (CAL) for 64 simultaneous FFTs and the scalar FFT has 6.8 Mflops on the CRAY-1. The scalar FFT has 82.5 kflops on the KL-10. Comparing the vectorized FFT times, the following estimates for performance are consistent: 1 CRAY = 1000× VAX 780 and 1 CRAY = 548× KL-10, while comparing the scalar performance, 1 CRAY = 166× VAX 780 and 1 CRAY = 82× KL-10.

A different way of stating this result is that programs that emphasize indirect addressing make poor use of number crunchers. A more cost effective expenditure of money than attempting software conversion may be in a low-cost large address space virtual memory symbol cruncher like VAX, or Lisp machines. Fateman's (1978) conclusion for the 7600 is reaffirmed:

LISP programs such as MACSYMA make poor use of scientific floating point machines such as the CRAY. A more effective expenditure of money than attempting software conversion may be in low-cost large address space virtual memory symbol crunchers. Hybrid problem solving where modern network technology makes possible links between number crunchers and symbol crunchers may be an attractive alternative.

4. SYMBOLIC BENCHMARK

Symbolic computing can differ significantly from numerical computing in the way problems are formulated, solved and interpreted. In this Section, a symbolic calculation is examined and its solution is compared on several different machines and installations. This calculation is interesting because it illustrates many useful features of a symbolic computing system – exact arithmetic, differentiation, integration, recursion, large expression manipulation, and optimization for both stable numerical evaluation and for runtime efficiency. This test does not include other useful features like pattern matching capabilities. This tests also omits any measurement of the usefulness of interactivity (bandwidth) in displaying or manipulating very large expressions.

4.1 Is a Symbol Crunch Different than A Number Crunch?

In 1972, SIGSAM (Special Interest Group in Symbolic and Algebraic Manipulation) received a problem posed by Campbell (1972) which follows. SIGSAM problem #2 involves the computation of a class of functions useful in phase-integral approximations to a time-dependent oscillator developed by Campbell (1972).

The equation

$$\frac{d^2\psi}{dt^2} + \frac{\Omega^2(t)}{\lambda^2}\psi = 0, \tag{1}$$

where λ is small, has a solution in the form

$$\psi(t) = \exp\left(\frac{i}{\lambda}\int \sum_{m=0}^{M} \lambda^m y_m(t)\,dt\right) \tag{2}$$

which is the JWKB solution of order M. The functions $y_m(t)$ are determined by the requirement that each power of λ is zero. In particular, $y_0(t) = \pm\Omega(t)$. The remaining functions follow from the recurrence

$$\frac{dy_{m-1}(t)}{dt} = -i\sum_{\mu=0}^{m} y_\mu(t) y_{m-\mu}(t). \tag{3}$$

If all the odd-order functions $y_{2m+1}(t)$ are eliminated in favor of even-order functions, an alternate form of ψ results

$$\psi(t) = \frac{\exp\left(\frac{i}{\lambda}\int \sum_{n=0}^{N} \lambda^{2n} y_{2n}(t)\,dt\right)}{\left(\lambda^{-\frac{1}{2}}\sum_{n=0}^{N} \lambda^{2n} y_{2n}(t)\,dt\right)}, \tag{4}$$

which is the phase-integral approximation of order $2N+1$.

Two independent solutions $\Psi_N^{\pm}$ contained in Eq. (4) can be written as

$$\Psi_N^{\pm} = q_N^{-\frac{1}{2}}(t)\exp\left(\pm i\int q_N(t)\,dt\right), \tag{5}$$

where

$$q_N(t) = \frac{\Omega(t)}{\lambda}\sum_{n=0}^{N} Y_{2n}(t), \tag{6}$$

and Y_{2n} is an abbreviation for the combinations of the y functions which are computed from (3). Explicitly, they are given by the defining relations

$$Y_{2n} = \epsilon_0 Y_{2n-2} - \tfrac{1}{4}Y''_{2n-2} + \tfrac{1}{2}\sum_{\alpha+\beta=n}{}' Y_{2\alpha}Y_{2\beta} - \tfrac{1}{2}\sum_{\alpha+\beta+\gamma+\delta=n}{}' Y_{2\alpha}Y_{2\beta}Y_{2\gamma}Y_{2\delta}$$
$$+ \tfrac{1}{2}\sum_{\alpha+\beta=n-1}{}' [\epsilon_0 Y_{2\alpha}Y_{2\beta} + \tfrac{3}{4}Y'_{2\alpha}Y'_{2\beta} - \tfrac{1}{4}(Y_{2\alpha}Y''_{2\beta} + Y''_{2\alpha}Y_{2\beta})] \tag{7}$$

for $n > 1$. It is easy to see that $Y_0(t) = 1$. One also finds that $Y_2(t) = \frac{1}{2}\epsilon_0$, where

$$\epsilon_0 = \left(\frac{\lambda}{\Omega}\right)^{\frac{3}{2}}\frac{d^2}{dt^2}\left(\frac{\Omega(t)}{\lambda}\right)^{-\frac{1}{2}} \tag{8}$$

In Eq. (7), each prime on a Y function denotes one differentiation with the operator $\lambda\Omega^{-1}(t)(\frac{d}{dt})$. The primes on the summations have a differing meaning: it is required that at least two Y functions appearing in any term of each sum have subscript indices greater than zero.

In a general determination of $Y_{2n}(t)$, there is no need to differentiate directly with respect to t. It is possible to write the Y functions purely in terms of ϵ_0 and the quantities

$$\epsilon_m = \left(\frac{\lambda}{\Omega(t)}\frac{d}{dt}\right)^m \epsilon_0, \tag{9}$$

for $1 \leq m \leq 2n-2$.

For reference, several algorithms for calculating the functions Y_{2N} are described in Appendix C . The explicit determination for $N = 0$ thru $N = 10$ are listed below in Appendix A.

An interesting question arises for the computational physicist here if he wishes to examine the expressions in detail. Namely, can the symbolic computing system display the output in a useful way? Or be coupled with a word processor? When the expressions are as large as Y_{20}, say, it is important to have some facilities for grouping the expressions in systematic and readable fashion. The following output was produced in three stages. First, a simple MACSYMA subprogram was devised to take a MACSYMA expression and transform it into a form compatible with the word processing system TEX. Next, MACSYMA's GRIND and display facilities were used to write the output to a disk file. Then, each subgroup of terms on a line from Y_{2N} was examined and regrouped for a convenient display of multiple-line equations. Finally, the file was incorporated into the input file for the word processor. The capability to systematically generate output in form compatible with a word processor is particularly important and convenient for error checking the calculation by examining a user friendly and readable form of the answer. The results of generating these Y_{2N} for N upto 10 are listed in Appendix A.

4.2 Calculation of the Adiabatic Invariants

When the parameters of a physical system vary slowly under the effect of an external perturbation, some quantities are constant to any order of the variable describing the slow rate of change. Such a quantity is called an adiabatic invariant. This does not mean that these quantities are exactly constant, but rather that their variation goes to zero faster than

any power of the small parameter.

To find a series in the small parameter for a harmonic oscillator, a procedure has been devised by Kulsrud (1957) and later by Lewis (1968) who determined a class of exact invariants. It is well known that the ratio of energy to frequency for the oscillator is the zero-order adiabatic invariant. The produce adopted here for calculating the 2N-th order invariant I_N is to compute the ratio of the N-th order energy to N-th order frequency as determined by the phase integral approximation described above.

In terms of real variables, let the solution for the phase derived from Eq. (5) be $S_N = \int q_N(t_m)\,dt$ and the phase integral solution is

$$\Psi_N = \frac{1}{q_N(t)}[C_1 \sin(S_N) + C_2 \cos(S_N)]. \tag{10}$$

where C_1, C_2 are constants of integration. The adiabatic invariant, to order $2N$ is

$$I_N(t) = \frac{\left[q_n(t)^2 \Psi_N^2 + \left(\frac{d\Psi_N}{dt}\right)^2\right]}{2q_N(t)}. \tag{11}$$

This calculation for the invariants may be tested by a specific calculation involving the choice

$$\Omega(t) = (\epsilon t)^{-3/2} \tag{12}$$

where the frequency varies by two orders of magnitude over the range $\epsilon t = 1$ to $\epsilon t = 75$. MACSYMA expressions for invariants I_0 through I_4 are listed in Appendix B. A simple MACSYMA program that generates FORTRAN for these invariants and the FORTRAN is also included. Issues associated with the checking the accuracy of this calculation are described in Section 4.4.

4.3 Optimizing the Calculation of FORTRAN code

There are a number of very important issues associated with the generation of optimized FORTRAN, or some other language, as the output of a symbolic calculation. For those who develop large physics codes, using a symbolic computing system to formulate the physical

model with partial differential equations, to apply space and time discretization techniques, and to generate FORTRAN, or some other high level language, that is well suited for the hardware of the numerical machine are important objectives (See Cook, 1982, Wirth, 1980).

Typically, however, there are some serious reservations for very large codes. These include time and memory limitations for the processing of intermediate expressions, and the degree of optimization achievable for hardware performance, (*e.g.*, vectorizing for a CRAY), beyond optimization for numerically stable evaluation. Indeed, in Appendix B, the HORNER command provided in MACSYMA is very useful in generating expressions for stable numerical evaluation. However, the code may not be the most efficient for speed. On a supercomputer like the CRAY, certain nonlinear effects in performance for vectorizable problems occur because of "chaining." Chaining denotes the movement of intermediate results between vector registers when more than one vector register is in use. For example, chaining can be used between the multiply and add in the calculation of A*x(i)+y(i). Cook (1983) has also developed certain MACSYMA primative functions that help in generating FORTRAN for the CRAY.

There is a tradeoff to be made in writing efficient code between deciding to use FORTRAN (and thereby trust the compiler to generate efficient code) or to use assembler language (and thereby learn all the efficient techniques yourself) . The tricks that produce efficient assembler code are often not transportable to another site (with a different machine!), while a FORTRAN code usually is. On the other hand, writing highly optimized FORTRAN for run-time performance may be an arduous task in itself, perhaps as difficult as writing assembler language. There is also a danger that highly efficient FORTRAN code developed on one machine will not be efficient if it is transported to a new site. One suggestion is that a reasonable compromise can be achieved by automatically generating FORTRAN for the most part with a sensitivity to using the good pipelining techniques presented below, while isolated pieces of code can be written in assembler.

A final observation about coding styles and practices is appropriate here. Production codes would undoubtedly want methods in assembler or microcode. The observation is that ideas of data abstraction (message-passing semantics, generalized data structures, data flow analysis) are consistent with developing efficient vector coding. But, unlike other models of vectorization

(DO loops, expansions, temporary variables), properly chosen data abstractions can be used throughout the formulation of a problem (program) and can be applied to such activities as disk buffering, multi-processing environments, or parallel machine processing (data movement over a network, another machine doing pre- or post- processing). The formulation of an algorithm can be just as important as its efficient implementation.

The following example in FORTRAN illustrates several speed improvements as well as stable evaluation optimization techniques techniques so that they can be applied when new codes are developed for the CRAY and other pipeline machines. These techniques are different from the techniques for extracting common subexpressions (*e.g.*, Cook 1983). One can expect that these programs will be run at installations where there are handcoded routines already available for I/O buffering, random number generating, packing data and sorting while FORTRAN is more appropriate to express numerical algorithms.

Writing a particle pusher, in the style of FORTRAN, illustrates four techniques for vectorizing code for execution efficiency. A particle pusher is part of a kinetic model that calcualtions the motion of a number of of finite-size particles using time centered discretization of Newton's second law. The force at on each particle is calcualted by linear interpolation from a grid of forces. New charges (masses) from the update positions are also calculated using linear weights. Accuracy checks for the constancy of linear momentum and kinetic energy are also performed. A naive code fragment that illustrates this process follows (See also Hockney and Eastwood 1981). This discussion deliberately avoids any execution efficiency gained by the CFT compiler. Hockney (1978) addresses certain issues related to the quality of compiler code on different machines.

FORTRAN Fragment #2: Source for Naive 1d Particle Push

```
c
c...calculate acceleration  a with shape and
c...also the linear momentum p and kinetic energy ke
c...of particles il to iu. Particle positions are in array x
c...while particle velocities are in vx.
c
        v1s=0.
        v2s=0.
        p=0.
        do 100 i=il,iu
           j=x(i)
           vo=vx(i)
           vn=vo+a(j+1)+( x(i)-j )*( a(j+2)-a(j+1) )
           v1s=v1s+vn
           v2s=v2s+vo*vn
100        vx(i)=vn
        p=p+m*v1s*dxdt
        ke=ke+0.5*m*v2s*dxdt*dxdt
c
c now update the positions. Note that the particles have
c periodic boundary conditions. They are confined to the interval
c (0,xn). also accumulate the new charge density as well.
c
        do 200 i=il,iu
           x(i)=x(i)+vx(i)
           if( x(i).lt.0. ) x(i)=x(i)+xn
           if( x(i).ge.xn ) x(i)=x(i)-xn
           j=x(i)
           drho=qdx*( x(i)-j )
           rho(j+1)=rho(j+1)-drho+qdx
200        rho(j+2)=rho(j+2)+drho
```

This fragment of code costs 3.26 microseconds per particle on the CRAY-1. This was determined by using the system routine SECOND around these two loops. The simple structure of these loops suggests that vectorization is possible. There are five techniques to improve the speed (Berman 1979):

1. Remove the IF statements from the loop using the routines CVMGP, etc.

2. Minimize the indirect addressing that occurs through the variable J.

3. Use the memory fetches more effectively by explicitly writing out the contents of the loops. Two is better then one; four is better then two. The CRAY FORTRAN Compiler (CFT) at this writing does not object to expanding the loops in backward order. It does not vectorize the loops when expanded in forward order because believes that loop is then recursive.

4. Keep the loops separate. The summation of V1S and V2S is changed to a separate loop.

5. Introduce temporary vectors in the loops as well as temporary scalars.

The resultant code is

FORTRAN Fragment #3: Source for Optimized 1d Particle Push

```
c
c...introduce some temporary vectors
c
      common /comvec/idex(1024),fra(1024),dacn(1024),vnew(1024),
       ap(1024)
       dimension hp(64), hp2(64)
       v1s=0.
       v2s=0.
c
c...expect even units of iblksz particles
c...iblksz here is 1024 --best to have multiple of 64
c... 1024 works  better than 512
c
       do 2110 kb=il,iu,iblksz
c
c ..loop over blocks
c
          do 2010 i=1,iblksz,4
c
c...loop within a block
c...note explicit writing out of loop contents
c...use fra to hold the integer part of x
c...and do all the indirect  calculations here
c...note that the compiler recognizes repeated subscript
c...expressions if they are not too complicated
c
             fra(i+3)=int(x(kb+i+2))
             ap(i+3)=a(int(x(i+kb+2))+1)
             dacn(i+3)=a(int(x(i+kb+2))+2)-a(int(x(i+kb+2))+1)
             fra(i+2)=int(x(kb+i+1))
             ap(i+2)=a(int(x(i+kb+1))+1)
             dacn(i+2)=a(int(x(i+kb+1))+2)-a(int(x(i+kb+1))+1)
             fra(i+1)=int(x(kb+i))
             ap(i+1)=a(int(x(i+kb))+1)
```

```
            dacn(i+1)=a(int(x(i+kb))+2)-a(int(x(i+kb))+1)
            fra(i)=int(x(kb-1+i))
            ap(i)=a(int(x(i+kb-1))+1)
2010        dacn(i)=a(int(x(i+kb-1))+2)-a(int(x(i+kb-1))+1)
        do 300 i=1,iblksz,4
            vnew(i+3)=vx(i+kb+2)+ap(i+3)+(x(i+kb+2)-fra(i+3))*dacn(i+3)
            vnew(i+2)=vx(i+kb+1)+ap(i+2)+(x(i+kb+1)-fra(i+2))*dacn(i+2)
            vnew(i+1)=vx(i+kb)+ap(i+1)+(x(i+kb)-fra(i+1))*dacn(i+1)
            vnew(i)=vx(i+kb-1)+ap(i)+(x(i+kb-1)-fra(i))*dacn(i)
300   continue
c
c...now sum
c
        do 500 i=1,64
            hp2(i)=vx(i+kb -1)*vnew(i)
500         hp(i)=vnew(i)
        js=64
        kount=(iblksz-js)/64
        last=(iblksz-js)-64*kount
        do 520 j=1,kount
           do 515 i=1,64
              hp2(i)=hp2(i)+vx(i+js+kb-1 )*vnew(i+js)
515           hp(i)=hp(i)+vnew(i+js)
520        js=js+64
        do 530 i=1,last
           hp2(i)=hp2(i)+vx(kb-1+iblksz+1-i)*vnew(iblksz+1-i)
530        hp(i)=hp(i)+vnew(iblksz+1-i)
        do 540 i=1,min0(64,iblksz),1
           v1s=hp(i)+v1s
540        v2s=hp2(i)+v2s
        do 560 i=1,iblksz,4
           vx(i+kb+2)=vnew(i+3)
           vx(i+kb+1)=vnew(i+2)
           vx(i+kb)=vnew(i+1)
560        vx(i+kb-1)=vnew(i)
2110  continue
```

```
c
      p=p+m*v1s*dxdt
      ke=ke+0.5*m*v2s*dxdt*dxdt
c
c...now update the positions
c
       do 201 i=il,iu,4
          x(i+3)=x(i+3)+vx(i+3)
          x(i+2)=x(i+2)+vx(i+2)
          x(i+1)=x(i+1)+vx(i+1)
201       x(i)=x(i)+vx(i)
c
c...use the cvmgp routine to adjust the boundary conditions
c...no advantage to writing out the loops here
c
       do 2011 i=il,iu
2011      x(i)=cvmgp(x(i),x(i)+xn,x(i))

       do 2012 i=il,iu
2012      x(i)=cvmgp(x(i)-xn,x(i),x(i)-xn)
c
c...use the temporary vectors again for the indirect addressing
c
       do 2110 kb=il,iu,iblksz
          do 2014 i=1,iblksz,4
c
c...this time keep idex for indirect address and fra for fraction
c
             idex(i+3)= int(x(i+kb+2))
             fra(i+3)=x(i+kb+2)-idex(i+3)
             idex(i+2)= int(x(i+kb+1))
             fra(i+2)=x(i+kb+1)-idex(i+2)
             idex(1+i)= int(x(i+kb))
             fra(i+1)=x(i+kb)-idex(i+1)
             idex(i)= int(x(i+kb-1))
2014         fra(i)=x(i+kb-1)-idex(i)
```

```
      do 2015 i=1,iblksz,2
         rho(idex(i+1)+1)=rho(idex(i+1)+1)+qdx-qdx*fra(i+1)
         rho(idex(i+1)+2)=rho(idex(i+1)+2)+qdx*fra(i+1)
         rho(idex(i)+1) = rho(idex(i)+1) + qdx -qdx*fra(i)
         rho(idex(i)+2) = rho(idex(i)+2) +qdx*fra(i)
2015  continue
2110  continue
```

These rewritten code costs 1.53 microseconds per particle, or slightly more than *twice* as fast. We find it amusing to note that the original FORTRAN has 19 executable statements and many eyes, is tersely, but simply written. The version optimized for execution efficiency has 77 executable statements and, although longer, its structure is as straightforward once these optimizing techniques are understood. In particular, these techniques have been successfully applied to a hybrid $1\frac{1}{2}$-dimensional guiding center electrostatic code in a constant magnetic field and to a 3-dimensional magnetohydrodynamic code in toroidal geometry (Berman and Leboeuf 1984, Brunel *et. al.* 1980) where factors to two to three improvments in execution speed have been obtained on the CRAY.

The moral of this tale seems to be that these four techniques mentioned above *can* make a significant difference to FORTRAN programmers on the CRAY. Note that the vectorized code, although improved for the CRAY, would suffer on a machine like the IBM 370 because of the address calculations. The most significant improvement came from removing the IF statements. The next two improvements came from expanding the loops to use jumps of two (less writing), three, and then four before writer's cramp set in. There was no advantage in expanding the loops involving the system routine CVMGP, but in most cases, jumps of two in loops seem to be worthwhile. The indirect address calculation and the use of temporary vectors of length 1024 made about a 12% final improvement. The length 1024 was a compromise between speed at lengths 512 or 256 and storage for lengths of larger multiples of 64. Finally, the transfer of the summations for the momentum and kinetic energy to loops outside the push helped. There was a slightly slower time difference in explicitly using loops of length 64 here as compared to longer lengths.

4.4 Checking the Answer for the Adiabatic Invariants

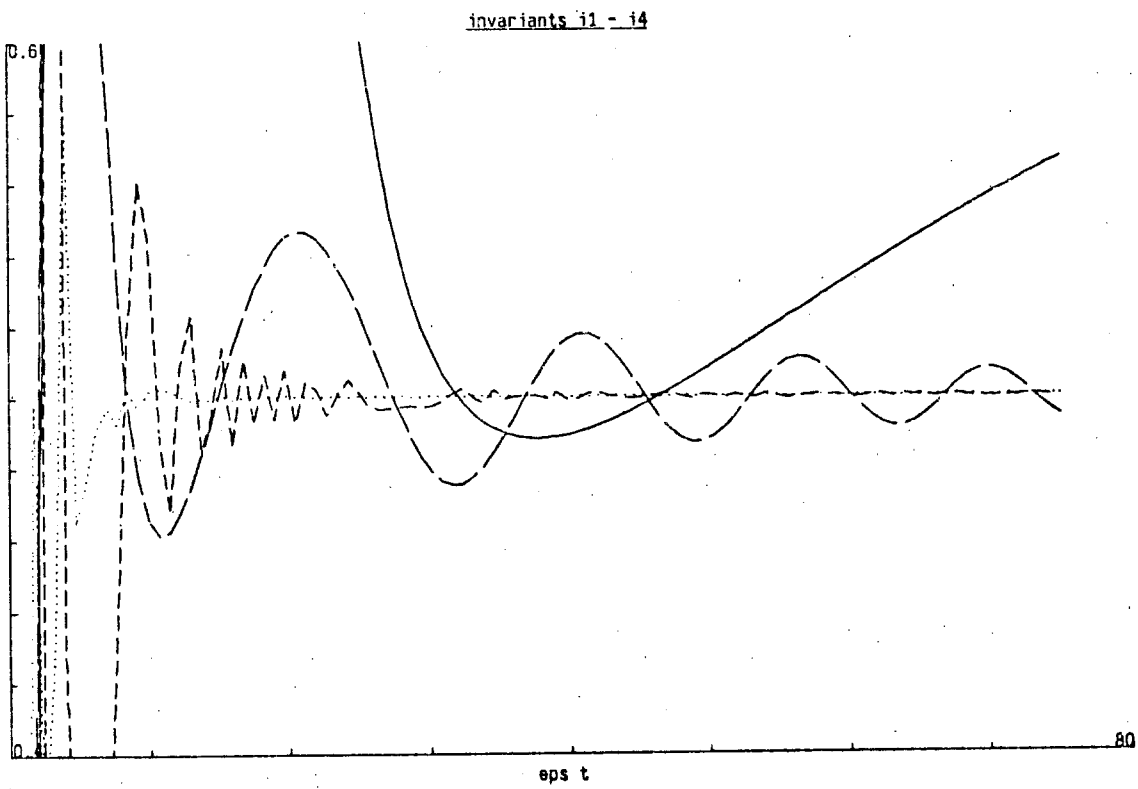
An important part of the the above calculations is checking the answer. When expressions are this big and unwieldy to manipulate or write down, it is important to develop some heuristics to to verify the answer. There are probably three checks that are worth performing.

First, for the Y_{2N}, certain simple recursion relations exists so that the leading coefficients

(powers of ϵ) can be evaluated. Next, exact numerical evaluation of the functions can be perform for very special cases, *e.g.*, $\Omega(t)$ a power of t, and checked. Thirdly, pictures showing the behavior of the invariants can detect systematic trends of errors. In the abovementioned example, $\Omega(t) = t^{3/2}$ and the Figure 1 shows the quality of the the invariants.

The adiabatic invariants $I_1(\epsilon t)$ through $I_4(\epsilon t)$ for $\Omega(t) = (\epsilon t)^{-3/2}$ are shown in Figure 1 for $\epsilon t = 1$ to $\epsilon t = 75$, This represents a change of almost two orders of magnitude in the frequency. In the diagram, I_1 is the solid line, I_2 is the broken line, I_3 is the short dashed line, and I_4 is the dotted lined. Note the successive improvments in constancy of I_N with N.

Figure 1. The Adiabatic Invariants I_1 through I_4



4.5 Timing Results

The calculation of the Y_{2N} has been performed in MACSYMA on DEC KL-10 and 2060, VAX 780 (VAXIMA and NIL MACSYMA), Lisp Machine Cadr and Symbolics 3600 with (5/84) and without (4/83) Symbolics Software Release 5.0. Using REDUCE, the calculation has been performed on DEC KL-10 and 2060, and CRAY 1s using using a bootstrap version of Lisp in 1982 and Portable Standard Lisp in 1984. It has also been performed in SMP on VAX 780 in 1982.

For the Symbolics 3600, the timings did not include performance improvements in software Release 5.0 or certain hardware improvements involving an instruction fetch unit or a floating point accelerator. It is also likely that there are a number of primitives which MACSYMA depends on which could and may be microcoded (*e.g.*, EQUAL). Finally, the new garbage collector is omitted. These additional hardware, microcode, and software improvements could easily result in another factor of 2 in overall performance. Thus, the Symbolics 3600 and its version of MACSYMA is still undergoing improvements and has not stabilized, unlike VAXIMA. The same can probably be said about NIL MACSYMA, although it is harder to estimate resources supporting improvements in NIL MACSYMA.

In performing the timings for the table lookup test, there were important differences between interpreted code and code translated or compiled into Lisp. For the KL-10, two different algorithms for calculating the Y_{2N} were used: a very naive algorithm making inefficient use of automatic storage facilities and a second one using array functions to store intermediate results (See Appendix C). Also, a translated version of the algorithm was run. No compiled versions were attempted. It is more likely the first time the problem is being formulated, the naive, untranslated algorithm will be used and represents the average users' experience.

The results of measuring the times for calculating Y_{2N} with the internal timers in MACSYMA and SMP are in Table 3.

Some entries that were unavailable because of memory limitations are denoted by M while those unavailable because of time limitations are denoted T.

Table 3. Times for Y_N in MACSYMA and SMP

	MIT-MC KL-10 naive	MIT-MC KL-10	MIT-MC KL-10 (Trans.)	VAXIMA VAX-780	NIL VAX-780	Dec 2060 Tops 20	Lisp Machine CADR/LM2	Lisp Machine 3600	Lisp Machine 3600	SMP VAX-780
Date	2/82	2/82	2/82	4/82	4/83	4/83	4/82	4/83	5/84	4/82
N										
4.	0.987	1.310	0.322	2.133	7.5	0.719	3.716	4.47	7.580	0.6
6.	2.900	2.071	1.320	4.833	15.95	1.472	4.916	3.12	1.530	1.05
8	5.090	4.453	2.035	10.433	29.59	2.850	9.150	5.93	3.870	2.17
10.	8.334	7.004	3.948	16.150	51.88	4.531	17.533	10.3	12.600	4.35
12.	15.236	11.643	7.132	30.416	92.23	8.809	27.983	18.8	15.600	9.9
14.	26.232	18.658	13.875	49.333	168.23	15.528	48.800	47.4	27.600	17.9
16.	45.299	31.065	25.087	87.583	287.22	25.886	84.566	56.7	30.200	38.5
18.	80.781	54.344	43.485	146.233	T	42.439	154.117	103.9	48.700	74.6
20.	140.026	91.924	74.248	269.666	T	67.110	283.333	175.1	96.100	158.7
22.	233.921	146.229	M	632.350	T	M	T	337.5	219.000	M

It is important to note that there are important differences in the timings for running the problem in interpreted or translated code. On the basis of running interpreted MACSYMA, the following estimates of relative performance can be made: 1 KL-10 = 3× VAX 780 and 1 KL-10 = 1 Symbolics 3600. Also, 1 VAXIMA (780) = 3× NIL MACSYMA. (780) Finally, 1 SMP (780) = 2× VAXIMA (780).

The results of times for Y_{2N} in REDUCE are in Table 4.

Table 4. Times for Y_N in REDUCE

N	KL-10	DEC2060	CRAY-1 1983	CRAY-1s (PSL) 1984
4	0.332	-	0.212	0.032
6	1.179	-	0.512	0.070
8	2.038	1.747	1.100	0.102
10	3.387	2.479	2.248	0.281
12	8.573	5.615	4.580	0.556
14	15.300	10.856	10.509	1.954
16	20.995	19.959	18.312	3.858
18	62.268	38.590	39.210	7.765
20	M	69.798	76.161	15.307
22	M	T	149.073	
24	M	T	285.218	

One serious problem occurred with the timings for CRAY REDUCE when this test was first performed in 1982: the timer did not accurately reflect the time used or charged. In initial attempts to time the calculation, the total running time for this problem was some 16,390s or some 25 min. The total time for running this problem on KL-10 MACSYMA was also some 25 min. Since the raw computing speed of the CRAY and KL-10 differ by a factor of ≈ 300 in performing arithmetic (FFTs), this discrepancy shows that the implementation of REDUCE has a long way to go to be efficient in utilizing CRAY hardware speeds based on 1982 timings. Substantial efforts have going into producing the PSL version of REDUCE for the CRAY. The PSL version shows a factor of 4 improvement and can be expected to continue to improve. On this basis, the following estimates of relative performance obtain: 1 CRAY = 10× KL-10 and 1 CRAY = 5× DEC 2060.

Taken together, these two MACSYMA/SMP and REDUCE tests are consistent with estimating 1 CRAY = 6× Symbolics 3600, 1 CRAY = 10× SMP (780) and 1 CRAY = 16× VAXIMA (780).

A final, subjective comment about the interactivity of running this problem on these different machines should be made. There are important differences in the quality of solving the problem on a Symbolics 3600 than a CRAY primiarily because of the degree of interactivity and the quality of the display. It is much more conveient to check the calulcation with the word processor or graphical techniques described above in a Lisp Machine than a CRAY. The lack of interactivity on the CRAY for these types of problems is a substantial handicap.

5. CONCLUSION – COST AND PERFORMANCE

A supercomputer can service the needs of a few thousand scientists and engineers, while a symbolic computer typically services a few users. This difference arises from the resources and needs of the users. In a supercomputer environment, the total number of floating point operations is most important, while symbolic computing tends to emphasize rapid turnaround and interactivity. Thus, a supercomputer environment puts a premium on delivering CPU cycles

to a large number of users running large or long calculations, while a symbolic computing environment emphasizes bandwidth.

It is also interesting to note that the capital costs of a CRAY installation are approximately $20M while a VAX 780 is $0.5M and a Symbolics 3600 is $0.1M. Their speeds on floating point problems (FFT) are 1 CRAY = 1000× VAX 780 and 1 CRAY = 1000× Symbolics 3600 while their speeds for symbolic problems are 1 CRAY = 20× VAX 780 or 6× Symbolics 3600. Thus, the cost of the specialized vector hardware and fast memory design on the CRAY is justified by this type of average arithmetic performance improvement of some 10 to 1000 over the VAX or Symbolics 3600. Further the cost of Lisp machines will be incrementally declining rapidly during the next year or two, probably by at least a factor of two in the next year alone (in several steps), which cannot be said of VAXs with the same performance of a 780 (although some new VAXs, like the 785, are more powerful, they cost more). Then, of course, this estimate does not take into account all the support features (*e.g.*, bit-map displays, laser printers) with the Symbolics 3600.

A more effective expenditure of money to improve scientific productivity may be in low cost large address space virtual memory symbolic machines functioning as front-end workstations to numerical machines. Modern network technology makes this link feasible.

A second reason why a symbolic computing – supercomputer link is important is because it provides the capability of automatically generating numerically intensive codes. Noting that this capability has existed for some 10 years with MACSYMA, it has recently been applied in plasma physics to the development of of codes of some thousands of lines for magnetohydrodynamic problems (See Wirth, 1980 or Cook 1982).

It is perhaps the flexibility and integrated environment that a symbolic computing facility like MACSYMA provides that makes it an important avenue for overcoming the FORTRAN barrier (Wilson 1984) to the computerization of science. Earlier efforts at symbolic generation of large production codes (symbolic Algol as described by Petravic, Kuo-Petravic and Roberts, 1972; OLYMPUS from the Computer Physics Communications Library, Belfast. See also the example of OLYMPUS code cited in Hockney and Eastwood, 1981) have not achieved wide popularity in part because of the rigidity of their structural stylistic conventions, and because

of the difficulties in controlling their execution time efficiencies. By using a more personally convenient symbolic computing environment, these objections can be easily overcome.

ACKNOWLEDGMENTS

It is a pleasure to thank John Aspinall, Francois Brunel, George Carrette, Richard Fateman, Ellen Golden, Jeff Greif, Charles Karney, Tony Hearn, John Kulp, Jean-Noel Leboeuf, Ralph Lewis, Jim O'Dell for assistance in running these benchmarks or for useful discussions. Some of the benchmarks reported in this paper were initially described at a Workshop on Symbolic Computing held at Los Alamos in 1982.

This work is supported in part by the National Science Foundation, the Department of Energy and the Office of Naval Research. Certain discussions at the Insitute for Fusion Studies were supported by N.S.F. grant ATN-82-14730.

REFERENCES

1 R.H. Berman, *Buffer*, **3**, 12 (1979).

2 R.H. Berman, "Symbolic Computing in Plasma Physics," 10th Conference on the Numerical Simulation of Plasma, San Diego (1983).

3 R.H. Berman and J.L. Kulp, "A Computational Physics Environment," 2nd Macsyma Users' Conference, Washington (1979).

4 R.H. Berman and J.N. Leboeuf, *Annual Sherwood Theory Meeting*, Lake Tahoe (1984).

5 E.O. Bingham, 1974, *The Fast Fourier Transform*, (Prentice-Hall: New York).

6 J. Bitoun, A. Nadeau, J. Guyard and M.R. Feix, J. Comp. Phys, **12**, 315 (1973).

7 F. Brunel, J.N. Leboeuf, T. Tajima, J.M. Dawson, M. Makino, and T. Kamimura, *J. Comp. Phys.*, **43**, 268 (1980).

8 J.A. Campbell, *J. Comp. Phys.*, 10, 308 (1972).

9 J.A. Campbell, *SIGSAM Bull.*, 22, 8 (1972).

10 G. Cook, Ph. D. thesis, UCRL-53324, U. California (1982).

11 G. Cook, private communication (1983).

12 R.J. Fateman, *SIGSAM Bull.*, 12, 8 (1978).

13 D. Fuss, *Buffer*, 8, 1 (1984).

14 A.C. Hearn, *SIGSAM Bull.*, 24, 14 (1972).

15 R.W. Hockney, 1978, in *Fast Poisson Solvers and Applications*, ed. U. Schumann, (Advance Publications: London), 78 - 97.

16 R.W. Hockney and J.W. Eastwood, 1981, *Computer Simulation Using Particles*, (McGraw Hill: New York).

17 R.D. Jenks and J.H. Griesmer, *SIGSAM Bull.*, 24, 3 (1972).

18 R.M. Kulsrud, *Phys. Rev.*, 106, 205 (1957).

19 H. R. Lewis, *J. Math. Phys*, 9, 1976 (1968)

20 K.G. Wilson, *Proc. I.E.E.E.*, 72, 6 (1984).

21 M.C. Wirth, Ph. D. thesis, UCRL-52996, U. California (1980).

APPENDIX A. TABULATION OF Y_N UPTO $N = 20$

This Section lists the functions Y_0 to Y_{20} using automatically generated output from MACSYMA for TEX.

$$Y_0 = 1$$

$$Y_2 = \frac{\epsilon_0}{2}$$

$$Y_4 = -\frac{\epsilon_0^2 + \epsilon_2}{8}$$

$$Y_6 = \frac{2\epsilon_0^3 + 5\epsilon_1^2 + 6\epsilon_0\epsilon_2 + \epsilon_4}{32}$$

$$Y_8 = -\frac{5\epsilon_0^4 + 50\epsilon_0\epsilon_1^2 + 30\epsilon_0^2\epsilon_2 + 19\epsilon_2^2 + 28\epsilon_1\epsilon_3 + 10\epsilon_0\epsilon_4 + \epsilon_6}{128}$$

$$\begin{aligned} Y_{10} = \frac{1}{512}[&14\epsilon_0^5 + 350\epsilon_0^2\epsilon_1^2 \\ &+ (140\epsilon_0^3 + 442\epsilon_1^2)\epsilon_2 + 266\epsilon_0\epsilon_2^2 + 392\epsilon_0\epsilon_1\epsilon_3 + 69\epsilon_3^2 \\ &+ (70\epsilon_0^2 + 110\epsilon_2)\epsilon_4 + 54\epsilon_1\epsilon_5 + 14\epsilon_0\epsilon_6 + \epsilon_8] \end{aligned}$$

$$\begin{aligned} Y_{12} = -\frac{1}{2048}[&42\epsilon_0^6 + 2100\epsilon_0^3\epsilon_1^2 + 1105\epsilon_1^4 \\ &+ (630\epsilon_0^4 + 7956\epsilon_0\epsilon_1^2)\epsilon_2 + 2394\epsilon_0^2\epsilon_2^2 + 1262\epsilon_2^3 \\ &+ (3528\epsilon_0^2\epsilon_1 + 5564\epsilon_1\epsilon_2)\epsilon_3 + 1242\epsilon_0\epsilon_3^2 \\ &+ (420\epsilon_0^3 + 1630\epsilon_1^2 + 1980\epsilon_0\epsilon_2)\epsilon_4 + 251\epsilon_4^2 \\ &+ (972\epsilon_0\epsilon_1 + 418\epsilon_3)\epsilon_5 + (126\epsilon_0^2 + 238\epsilon_2)\epsilon_6 \\ &+ 88\epsilon_1\epsilon_7 + 18\epsilon_0\epsilon_8 + \epsilon_{10}] \end{aligned}$$

$$
\begin{aligned}
Y_{18} = \frac{1}{131072}\{&1430\epsilon_0^9 + 300300\epsilon_0^6\epsilon_1^2 + 3160300\epsilon_0^3\epsilon_1^4 + 828250\epsilon_1^6 + 5771864\epsilon_1\epsilon_3^3 \\
&+ (51480\epsilon_0^7 + 5688510\epsilon_0^4\epsilon_1^2 + 14908500\epsilon_0\epsilon_1^4)\epsilon_2 \\
&+ (681684\epsilon_0^5 + 26912340\epsilon_0^2\epsilon_1^2)\epsilon_2^2 \\
&+ (3609320\epsilon_0^3 + 17287012\epsilon_1^2)\epsilon_2^3 + 5229510\epsilon_0\epsilon_2^4 \\
&+ [1009008\epsilon_0^5\epsilon_1 + 13166100\epsilon_0^2\epsilon_1^3 \\
&\quad + (15913010\epsilon_0^3\epsilon_1 + 25313832\epsilon_1^3)\epsilon_2 + 46002210\epsilon_0\epsilon_1\epsilon_2^2]\epsilon_3 \\
&+ (888030\epsilon_0^4 + 16871220\epsilon_0\epsilon_1^2 + 10265580\epsilon_0^2\epsilon_2 + 11803634\epsilon_2^2)\epsilon_3^2 \\
&+ [60060\epsilon_0^6 + 1661800\epsilon_0^3\epsilon_1^2 + 3682110\epsilon_1^4 \\
&\quad + (1415700\epsilon_0^4 + 26811720\epsilon_0\epsilon_1^2)\epsilon_2 + 8159580\epsilon_0^2\epsilon_2^2 + 6210260\epsilon_2^3 \\
&\quad + (12002640\epsilon_0^2\epsilon_1 + 27473720\epsilon_1\epsilon_2)\epsilon_3 + 6184110\epsilon_0\epsilon_3^2]\epsilon_4 \\
&+ (717860\epsilon_0^3 + 1003970\epsilon_1^2 + 1911660\epsilon_0\epsilon_2)\epsilon_4^2 + 577502\epsilon_4^3 \\
&+ [691980\epsilon_0^4\epsilon_1 + 4313100\epsilon_0\epsilon_1^3 + 7916640\epsilon_0^2\epsilon_1\epsilon_2 + 9055188\epsilon_1\epsilon_2^2 \\
&\quad + (1195480\epsilon_0^3 + 6653604\epsilon_1^2 + 8163240\epsilon_0\epsilon_2)\epsilon_3 \\
&\quad + (1778010\epsilon_0\epsilon_1 + 2876836\epsilon_3)\epsilon_4]\epsilon_5 \\
&+ (359970\epsilon_0^2 + 948614\epsilon_2)\epsilon_5^2 + 102930\epsilon_0\epsilon_6^2 \\
&+ [36036\epsilon_0^5 + 1665300\epsilon_0^2\epsilon_1^2 + (680680\epsilon_0^3 + 3762444\epsilon_1^2)\epsilon_2 \\
&\quad + 2309580\epsilon_0\epsilon_2^2 + 3403680\epsilon_0\epsilon_1\epsilon_3 + 1022138\epsilon_3^2 \\
&\quad + (616980\epsilon_0^2 + 1623780\epsilon_2)\epsilon_4 + 790828\epsilon_1\epsilon_5]\epsilon_6 \\
&+ [251680\epsilon_0^3\epsilon_1 + 457920\epsilon_1^3 + 1689840\epsilon_0\epsilon_1\epsilon_2 \\
&\quad + (385420\epsilon_0^2 + 1009928\epsilon_2)\epsilon_3 + 592336\epsilon_1\epsilon_4 + 180120\epsilon_0\epsilon_5]\epsilon_7 \\
&+ (12870\epsilon_0^4 + 276300\epsilon_0\epsilon_1^2 + 170820\epsilon_0^2\epsilon_2 \\
&\quad + 222186\epsilon_2^2 + 328056\epsilon_1\epsilon_3 + 120060\epsilon_0\epsilon_4 + 22878\epsilon_6)\epsilon_8 \\
&+ (50700\epsilon_0^3\epsilon_1 + 130380\epsilon_1\epsilon_2 + 60000\epsilon_0\epsilon_3 + 16014\epsilon_5)\epsilon_9 \\
&+ (2860\epsilon_0^3 + 17490\epsilon_1^2 + 21780\epsilon_0\epsilon_2 + 8734\epsilon_4)\epsilon_{10} \\
&+ (5400\epsilon_0\epsilon_1 + 3638\epsilon_3)\epsilon_{11} + (390\epsilon_0^2 + 1118\epsilon_2)\epsilon_{12} \\
&+ 238\epsilon_1\epsilon_{13} + 30\epsilon_0\epsilon_{14} + \epsilon_{16} + 12869\epsilon_7^2\}
\end{aligned}
$$

$$
\begin{aligned}
Y_{14} = \frac{1}{8192}\{&132\epsilon_0^7 + 11550\epsilon_0^4\epsilon_1^2 + 24310\epsilon_0\epsilon_1^4 + (13662\epsilon_0^2 + 26322\epsilon_2)\epsilon_3^2 \\
&+ (2772\epsilon_0^5 + 87516\epsilon_0^2\epsilon_1^2)\epsilon_2 + (17556\epsilon_0^3 + 69006\epsilon_1^2)\epsilon_2^2 \\
&+ 27761\epsilon_0\epsilon_2^3 + (25872\epsilon_0^3\epsilon_1 + 33760\epsilon_1^3 + 122408\epsilon_0\epsilon_1\epsilon_2)\epsilon_3 \\
&+ (2310\epsilon_0^4 + 35860\epsilon_0\epsilon_1^2 + 21780\epsilon_0^2\epsilon_2 + 20922\epsilon_2^2 + 30776\epsilon_1\epsilon_3)\epsilon_4 + 5522\epsilon_0\epsilon_1^2 \\
&+ (10692\epsilon_0^2\epsilon_1 + 20376\epsilon_1\epsilon_2 + 9196\epsilon_0\epsilon_4)\epsilon_5 \\
&+ 923\epsilon_5^2 + (924\epsilon_0^3 + 4270\epsilon_1^2 + 5236\epsilon_0\epsilon_2 + 1582\epsilon_4)\epsilon_6 \\
&+ (1936\epsilon_0\epsilon_1 + 988\epsilon_3)\epsilon_7 + (198\epsilon_0^2 + 438\epsilon_2)\epsilon_8 \\
&+ 130\epsilon_1\epsilon_9 + 22\epsilon_0\epsilon_{10} + \epsilon_{12}\}
\end{aligned}
$$

$$
\begin{aligned}
Y_{16} = -\frac{1}{32768}\{&429\epsilon_0^8 + 60060\epsilon_0^5\epsilon_1^2 + 316030\epsilon_0^2\epsilon_1^4 + 360932\epsilon_0^2\epsilon_2^3 + 174317\epsilon_2^4 \\
&+ (12012\epsilon_0^6 + 758472\epsilon_0^3\epsilon_1^2 + 496950\epsilon_1^4)\epsilon_2 + (114114\epsilon_0^4 + 1794156\epsilon_0\epsilon_1^2)\epsilon_2^2 \\
&+ (168168\epsilon_0^4\epsilon_1 + 877760\epsilon_0\epsilon_1^3 + 1591304\epsilon_0^2\epsilon_1\epsilon_2 + 1533408\epsilon_1\epsilon_2^2)\epsilon_3 \\
&+ (118404\epsilon_0^3 + 562474\epsilon_1^2 + 684372\epsilon_0\epsilon_2)\epsilon_3^2 \\
&+ [12012\epsilon_0^5 + 466180\epsilon_0^2\epsilon_1^2 + (188760\epsilon_0^3 + 893724\epsilon_1^2)\epsilon_2 \\
&\quad + 543972\epsilon_0\epsilon_2^2 + 800176\epsilon_0\epsilon_1\epsilon_3 + 206138\epsilon_3^2]\epsilon_4 \\
&+ (71786\epsilon_0^2 + 163722\epsilon_2)\epsilon_4^2 + 23998\epsilon_0\epsilon_5^2 \\
&+ [92664\epsilon_0^3\epsilon_1 + 144780\epsilon_1^3 + 529776\epsilon_0\epsilon_1\epsilon_2 \\
&\quad + (119548\epsilon_0^2 + 272108\epsilon_2)\epsilon_3 + 159268\epsilon_1\epsilon_4]\epsilon_5 \\
&+ (6906\epsilon_0^4 + 111020\epsilon_0\epsilon_1^2 + 68068\epsilon_0^2\epsilon_2 \\
&\quad + 76986\epsilon_2^2 + 113456\epsilon_1\epsilon_3 + 41132\epsilon_0\epsilon_4)\epsilon_6 \\
&+ 3431\epsilon_6^2 + (25168\epsilon_0^2\epsilon_1 + 56328\epsilon_1\epsilon_2 + 25688\epsilon_0\epsilon_3 + 6004\epsilon_5)\epsilon_7 \\
&+ (1716\epsilon_0^3 + 9210\epsilon_1^2 + 11388\epsilon_0\epsilon_2 + 4002\epsilon_4)\epsilon_8 \\
&+ (3380\epsilon_0\epsilon_1 + 2000\epsilon_3)\epsilon_9 + (286\epsilon_0^2 + 726\epsilon_2)\epsilon_{10} \\
&+ 180\epsilon_1\epsilon_{11} + 26\epsilon_0\epsilon_{12} + \epsilon_{14}\}
\end{aligned}
$$

$$
\begin{aligned}
Y_{18} = \frac{1}{131072}\{&1430\epsilon_0^9 + 300300\epsilon_0^6\epsilon_1^2 + 3160300\epsilon_0^3\epsilon_1^4 + 828250\epsilon_1^6 + 5771864\epsilon_1\epsilon_3^3 \\
&+ (51480\epsilon_0^7 + 5688540\epsilon_0^4\epsilon_1^2 + 14908500\epsilon_0\epsilon_1^4)\epsilon_2 \\
&+ (684684\epsilon_0^5 + 26912340\epsilon_0^2\epsilon_1^2)\epsilon_2^2 \\
&+ (3609320\epsilon_0^3 + 17287012\epsilon_1^2)\epsilon_2^3 + 5229510\epsilon_0\epsilon_2^4 \\
&+ [1009008\epsilon_0^5\epsilon_1 + 13166100\epsilon_0^2\epsilon_1^3 \\
&\quad + (15913040\epsilon_0^3\epsilon_1 + 25313832\epsilon_1^3)\epsilon_2 + 4600210\epsilon_0\epsilon_1\epsilon_2^2]\epsilon_3 \\
&+ (888030\epsilon_0^4 + 16871220\epsilon_0\epsilon_1^2 + 10265580\epsilon_0^2\epsilon_2 + 11803634\epsilon_2^2)\epsilon_3^2 \\
&+ [60060\epsilon_0^6 + 4661800\epsilon_0^3\epsilon_1^2 + 3682110\epsilon_1^4 \\
&\quad + (1415700\epsilon_0^4 + 26811720\epsilon_0\epsilon_1^2)\epsilon_2 + 8159580\epsilon_0^2\epsilon_2^2 + 6240260\epsilon_2^3 \\
&\quad + (12002640\epsilon_0^2\epsilon_1 + 27473720\epsilon_1\epsilon_2)\epsilon_3 + 6184140\epsilon_0\epsilon_3^2]\epsilon_4 \\
&+ (717860\epsilon_0^3 + 4003970\epsilon_1^2 + 4911660\epsilon_0\epsilon_2)\epsilon_4^2 + 577502\epsilon_4^3 \\
&+ [694980\epsilon_0^4\epsilon_1 + 4343400\epsilon_0\epsilon_1^3 + 7946640\epsilon_0^2\epsilon_1\epsilon_2 + 9055188\epsilon_1\epsilon_2^2 \\
&\quad + (1195480\epsilon_0^3 + 6653604\epsilon_1^2 + 8163240\epsilon_0\epsilon_2)\epsilon_3 \\
&\quad + (4778040\epsilon_0\epsilon_1 + 2876836\epsilon_3)\epsilon_4]\epsilon_5 \\
&+ (359970\epsilon_0^2 + 948614\epsilon_2)\epsilon_5^2 + 102930\epsilon_0\epsilon_6^2 \\
&+ [36036\epsilon_0^3 + 1665300\epsilon_0^2\epsilon_1^2 + (680680\epsilon_0^3 + 3762444\epsilon_1^2)\epsilon_2 \\
&\quad + 2309580\epsilon_0\epsilon_2^2 + 3403680\epsilon_0\epsilon_1\epsilon_3 + 1022138\epsilon_3^2 \\
&\quad + (616980\epsilon_0^2 + 1623780\epsilon_2)\epsilon_4 + 790828\epsilon_1\epsilon_5]\epsilon_6 \\
&+ [251680\epsilon_0^3\epsilon_1 + 457920\epsilon_1^3 + 1689840\epsilon_0\epsilon_1\epsilon_2 \\
&\quad + (385320\epsilon_0^2 + 1009928\epsilon_2)\epsilon_3 + 592336\epsilon_1\epsilon_4 + 180120\epsilon_0\epsilon_5]\epsilon_7 \\
&+ (12870\epsilon_0^4 + 276300\epsilon_0\epsilon_1^2 + 170820\epsilon_0^2\epsilon_2 \\
&\quad + 222186\epsilon_2^2 + 328056\epsilon_1\epsilon_3 + 120060\epsilon_0\epsilon_4 + 22878\epsilon_6)\epsilon_8 \\
&+ (50700\epsilon_0^2\epsilon_1 + 130380\epsilon_1\epsilon_2 + 60000\epsilon_0\epsilon_3 + 16014\epsilon_5)\epsilon_9 \\
&+ (2860\epsilon_0^3 + 17490\epsilon_1^2 + 21780\epsilon_0\epsilon_2 + 8734\epsilon_4)\epsilon_{10} \\
&+ (5400\epsilon_0\epsilon_1 + 3638\epsilon_3)\epsilon_{11} + (390\epsilon_0^2 + 1118\epsilon_2)\epsilon_{12} \\
&+ 238\epsilon_1\epsilon_{13} + 30\epsilon_0\epsilon_{14} + \epsilon_{16} + 12869\epsilon_7^2\}
\end{aligned}
$$

$$
\begin{aligned}
V_{20} \quad & \frac{1}{521288}\{1862c_0^{10} + 1158600c_0^7c_1^2 + 26862550c_0^4c_1^4 + 28160500c_0c_1^6 \\
& + (218790c_0^8 + 38682072c_0^5c_1^2 + 253141500c_0^2c_1^4)c_2 \\
& + (3879876c_0^6 + 305006520c_0^3c_1^2 + 243564834c_1^4)c_2^2 \\
& + (30679220c_0^4 + 587758408c_0c_1^2)c_2^3 \\
& + 88901670c_0^2c_2^4 + 109833566c_2^5 + 196243376c_0c_1c_3^3 + 29551761c_3^4 \\
& + [5717712c_0^6c_1 + 119219200c_0^3c_1^3 + 71217480c_1^5 \\
& \quad + (135260810c_0^4c_1 + 866670288c_0c_1^3)c_2 + 782038080c_0^2c_1c_2^2 + 599658104c_1c_2^3]c_3 \\
& + [6038601c_0^8 + 286861710c_0^5c_1^2 + (116343210c_0^3 + 658508748c_1^2)c_2 + 401323556c_0c_2^2]c_3^2 \\
& + [291720c_0^7 + 39625300c_0^4c_1^2 + 125191710c_0c_1^4 + (9626760c_0^5 + 45579924c_0^2c_1^2)c_2 \\
& \quad + (92475240c_0^3 + 521971110c_1^2)c_2^2 + 212168840c_0c_2^3 \\
& \quad + (136029920c_0^3c_1 + 255034560c_1^3 + 934106480c_0c_1c_2)c_3 \\
& \quad + (105130380c_0^2 + 281008100c_2)c_3^2]c_4 \\
& + [6101810c_0^4 + 136134980c_0c_1^2 + 83498220c_0^2c_2 + 111364926c_2^2 + 163583608c_1c_3]c_4^2 \\
& + 19635068c_0c_4^3 + (1749810c_0^2 + 5227602c_2)c_4^2 + [4725864c_0^5c_1 + 73837800c_0^2c_1^3 \\
& \quad + (90061920c_0^3c_1 + 168013536c_1^3)c_2 + 307876392c_0c_1c_2^2 + 135746860c_1c_2^3 \\
& \quad + (10161580c_0^4 + 226222536c_0c_1^2 + 138775080c_0^2c_2 + 184792292c_2^2)c_3 \\
& \quad + (81226680c_0^3c_1 + 215356288c_1c_2 + 97812424c_0c_3)c_4]c_5 \\
& + (4079660c_0^3 + 26095426c_1^2 + 32252876c_0c_2 + 13055198c_4)c_5^2 \\
& + [204204c_0^6 + 18873400c_0^3c_1^2 + 17419710c_1^4 \\
& \quad + (5785780c_0^4 + 127923096c_0c_1^2)c_2 + 39262860c_0^2c_2^2 + 34680068c_2^3 \\
& \quad + (57862560c_0^2c_1 + 152957768c_1c_2)c_3 + 34752692c_0c_3^2 + 11165226c_4^2 \\
& \quad + (6992440c_0^3 + 44667220c_1^2 + 55208520c_0c_2)c_4 \\
& \quad + (26888152c_0c_1 + 18534652c_3)c_5]c_6 \\
& + [2139280c_0^4c_1 + 15569280c_0c_1^3 + 28727280c_0^2c_1c_2 + 37731072c_1c_2^2 \\
& \quad + (20139424c_0c_1 + 13857916c_3)c_4 + (3062040c_0^2 + 9140140c_2)c_5 \\
& \quad + (4366960c_0^3 + 27776808c_1^2 + 34337552c_0c_2)c_3 + 3814924c_1c_6]c_7 \\
& + 437546c_0c_7^2 + 48619c_8^2 + 3823398c_3^2 + [87516c_0^5 + 4697100c_0^2c_1^2 + (1935960c_0^3 + 12214884c_1^2)c_2 \\
& \quad + 7554324c_0c_2^2 + 11153904c_0c_1c_3 + (2041020c_0^2 + 6076140c_2)c_4 \\
& \quad + 2964924c_1c_5 + 777852c_0c_6]c_8 \\
& + [574600c_0^3c_1 + 1192500c_1^3 + 4432920c_0c_1c_2 \\
& \quad + (1020000c_0^2 + 3021740c_2)c_3 + 1775980c_1c_4 + 544476c_0c_5 + 87514c_7]c_9 \\
& + [24310c_0^4 + 594660c_0c_1^2 + 370260c_0^2c_2 \\
& \quad + 544170c_2^2 + 804804c_1c_3 + 296956c_0c_4 + 63646c_6]c_{10} \\
& + [91800c_0^2c_1 + 266724c_1c_2 + 123692c_0c_3 + 37126c_5]c_{11} \\
& + (4420c_0^3 + 30342c_1^2 + 38012c_0c_2 + 17134c_4)c_{12} \\
& + (8092c_0c_1 + 6118c_3)c_{13} + (510c_0^2 + 1630c_2)c_{14} + 304c_1c_{15} + 34c_0c_{16} + c_{18}\}
\end{aligned}
$$

APPENDIX B. EXPLICIT MACSYMA AND FORTRAN EXPRESSIONS FOR $\Omega(t) = (t)^{3/2}$

The explicit invariants have been calculated in MACSYMA by using the above procedure for specific choice of $\Omega(t) = (t)^{-3/2}$. These expressions have been partially optimized for stable numerical evaluation with Horner's rule using the FLOATDEFUNK command. They have been transformed for direct inclusion in this report by using the MACSYMA GRIND and WRITEFILE commands. Of course, for this specific choice, the expressions are not of general utility. What is important is the ability to maniuplate such large expressions into usable FORTRAN code and check the accuracy of the calculation by grouping terms with word processors or by graphical means.

MACSYMA Fragment #2: Source for Invariants

```
I0(T):=BLOCK([T1,T2,T3,T4,T5,T6,T7],MODEDECLARE([T,EPS],FLONUM),T1:SQRT(EPS)**3,T2:EPS**(3/4),
   T3:1/T1,T4:SQRT(T),T5:2*T3/T4,T6:COS(T5),T7:T4**3,
   T1*(T2*T6**2/T7+(3*T2*T6/(4*T**(1/4))+SIN(T5)/(T2*T**(3/4)))**2)*T7/2);

I1(T):=BLOCK([T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14],MODEDECLARE([T,EPS],FLONUM),
   T1:SQRT(EPS)**3,T2:SQRT(T),T3:T2**3,T4:EPS**3,T5:1-3*T4*T/32,T6:1/T1,T7:1/T2,
   T8:-T6*(-3*T4*T2/16-2*T7),T9:COS(T8),T10:1/T3,T11:T**(3/4),T12:SQRT(T5),T13:EPS**(3/4),
   T14:1/T12,
   T1*T3
     *((SIN(T8)*(T10-3*T4*T7/32)*T11*T14/T13+3*T13*T9*T14/(4*T**(1/4))
     +3*EPS**(15/4)*T9*T11/(64*T12**3))
      **2
      +T6*T9**2*T10*T5)
    /(2*T5));
I2(T):=BLOCK([T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15],
   MODEDECLARE([EPS,T],FLONUM),T1:SQRT(EPS)**3,T2:SQRT(T),T3:T2**3,T4:EPS**3,T5:EPS**6,
   T6:63*T5*T**2/2048-3*T4*T/32+1,T7:1/T1,T8:1/T3,T9:1/T2,
   T10:2*T7*((63*T5*T3-576*T4*T2)/6144-T9),T11:COS(T10),T12:EPS**(3/4),T13:T**(3/4),
   T14:SQRT(T6),T15:1/T14,
   T1*T3
     *((-2*((189*T5*T2/2-288*T4*T9)/6144+T8/2)*T13*T15*SIN(T10)/T12
      +3*T12*T15*T11/(4*T**(1/4))-T12*T13*(63*T5*T/1024-3*T4/32)*T11/(2*T14**3))
      **2
      +T7*T8*T6*T11**2)
    /(2*T6));
I3(T):=BLOCK([T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17],
   MODEDECLARE([EPS,T],FLONUM),T1:SQRT(EPS)**3,T2:SQRT(T),T3:T2**3,T4:EPS**3,T5:EPS**6,T6:T**2,
   T7:EPS**9,T8:-1899*T7*T**3/65536+63*T5*T6/2048-3*T4*T/32+1,T9:1/T1,T10:1/T3,T11:1/T2,
   T12:2*T9*(-(5697*T7*T2**5-10080*T5*T3+92160*T4*T2)/983040-T11),T13:COS(T12),T14:EPS**(3/4),
```

```
   T15:T**(3/4),T16:SQRT(T8),T17:1/T16,
   T1*T3
     *((-2*T15*(T10/2-(28485*T7*T3/2-15120*T5*T2+46080*T4*T11)/983040)*T17*SIN(T12)/T14
      +3*T14*T17*T13/(4*T**(1/4))
      -T14*T15*(-5697*T7*T6/65536+63*T5*T/1024-3*T4/32)*T13/(2*T16**3))
      **2
      +T9*T10*T8*T13**2)
    /(2*T8));
I4(T):=BLOCK([T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20],
   MODEDECLARE([EPS,T],FLONUM),T1:SQRT(EPS)**3,T2:SQRT(T),T3:T2**3,T4:EPS**3,T5:EPS**6,T6:T**2,
   T7:EPS**9,T8:T**3,T9:EPS**12,
   T10:543483*T9*T**4/8388608-1899*T7*T8/65536+63*T5*T6/2048-3*T4*T/32+1,T11:1/T1,T12:1/T3,
   T13:1/T2,T14:T2**5,
   T15:2*T11*((8152245*T9*T2**7-5104512*T7*T14+9031680*T5*T3-82575360*T4*T2)/880803840-T13),
   T16:COS(T15),T17:EPS**(3/4),T18:T**(3/4),T19:SQRT(T10),T20:1/T19,
   T1*T3
     *((-2*T18
   *((57065715*T9*T14/2-12761280*T7*T3+13547520*T5*T2-41287680*T4*T13)/880803840+T12/2)
   *T20*SIN(T15)
      /T17
      +3*T17*T20*T16/(4*T**(1/4))
      -T17*T18*(543483*T9*T8/2097152-5697*T7*T6/65536+63*T5*T/1024-3*T4/32)*T16/(2*T19**3))
      **2
      +T11*T12*T10*T16**2)
    /(2*T10));
I5(T):=BLOCK(
   [T1,T2,T3,T4,T5,T6,T7,T8,T9,T10,T11,T12,T13,T14,T15,T16,T17,T18,T19,T20,T21,T22,T23],
   MODEDECLARE([T,EPS],FLONUM),T1:SQRT(EPS)**3,T2:SQRT(T),T3:T2**3,T4:EPS**3,T5:EPS**6,T6:T**2,
   T7:EPS**9,T8:T**3,T9:EPS**12,T10:T**4,T11:EPS**15,
   T12:-72251109*T11*T**5/268435456+543483*T9*T10/8388608-1899*T7*T8/65536+63*T5*T6/2048
   -3*T4*T/32+1,T13:1/T1,T14:1/T3,T15:1/T2,T16:T2**5,T17:T2**7,
   T18:2*T13
 *(-(842929605*T11*T2**9-260871840*T9*T17+163344384*T7*T16-289013760*T5*T3
       +2642411520*T4*T2)
  /28185722880
  -T15),T19:COS(T18),T20:EPS**(3/4),T21:T**(3/4),T22:SQRT(T12),T23:1/T22,
   T1*T3
     *((-2*T21
   *(T14/2-(7586366445*T11*T17/2-913051440*T9*T16+408360960*T7*T3-433520640*T5*T2
       +1321205760*T4*T15)
   /28185722880)*T23*SIN(T18)
      /T20
      +3*T20*T23*T19/(4*T**(1/4))
      -T20*T21
   *(-361255545*T11*T10/268435456+543483*T9*T8/2097152-5697*T7*T6/65536+63*T5*T/1024
 -3*T4/32)*T19
       /(2*T22**3))
```

```
  **2
  +I13*I14*I12*I19**2)
 /(2*I12));
```

The following MACSYMA code fragment takes the expressions for the invariants and generates FORTRAN code.

MACSYMA Fragment #3: Simple FORTRAN Generator for Invariants

```
/* the block generates FORTRAN for invariants I0 thru
        IN . First, OPTIMIZE identifies certain
        common subexpressions and tries to use
        Horner's rule to assist numerical evaluation.
          */

block([optimprefix:'t],
      result:[inv(1)=i0(t),inv(2)=i1(t),inv(3)=i2(t),
              inv(4)=i3(t),inv(5)=i4(t)],
      resopt:optimize(result),
      r0:rest(resopt),
      r1:last(r0),
      r2:reverse(rest(reverse(r0))),
      r3:subst("(","[",r1),
      r4:endcons(part(r3,1),r2),
      r4:endcons(part(r3,2),r4),
      r4:endcons(part(r3,3),r4),
      r4:endcons(part(r3,4),r4),
      r4:endcons(part(r3,5),r4),
      map(fortran,r4));
```

The resulting FORTRAN code fragment for the specifc choice $\Omega(t) = (\alpha t)^{-3/2}$ follows. As pointed out earlier, perhaps the most important feature of this fragment is the cability of manipulating it within a symbolic computing enviroment and deliver it to a large, efficient numerical machine.

FORTRAN Fragment #4: Source for Invariants

```
      T1 = EPS**1.5
      T2 = EPS**0.75
      T3 = 1/EPS**0.75
      T4 = 1/EPS**1.5
      T5 = T**0.5
      T6 = 1/T**0.5
      T7 = 2/(EPS**1.5*T**0.5)
      T8 = T**0.75
      T9 = COS(2/(EPS**1.5*T**0.5))
      T10 = 1/T**0.25
      T11 = T**1.5
      T12 = 1/T**1.5
      T13 = EPS**3
      T14 = -0.09375*EPS**3*T
      T15 = 1-0.09375*EPS**3*T
      T16 = (-0.1875*EPS**3*T**0.5-2/T**0.5)/EPS**1.5
      T17 = COS((-0.1875*EPS**3*T**0.5-2/T**0.5)/EPS**1.5)
      T18 = 1/(1-0.09375*EPS**3*T)**0.5
      T19 = EPS**6
      T20 = T**2
      T21 = 0.0307617188*EPS**6*T**2
      T22 = 0.0307617188*EPS**6*T**2-0.09375*EPS**3*T+1
      T23 = -1/T**0.5
      T24 = 2*(1.62760416E-4*(63*EPS**6*T**1.5-576*EPS**3*T**0.5)-1/T**0
     1    .5)/EPS**1.5
      T25 = COS(2*(1.62760416E-4*(63*EPS**6*T**1.5-576*EPS**3*T**0.5)-1/
     1    T**0.5)/EPS**1.5)
      T26 = -0.09375*EPS**3
      T27 = 0.0615234375*EPS**6*T
      T28 = 1/(0.0307617188*EPS**6*T**2-0.09375*EPS**3*T+1)**0.5
      T29 = 0.5/T**1.5
      T30 = EPS**9
      T31 = T**3
      T32 = -0.0289764404*EPS**9*T**3
      T33 = -0.0289764404*EPS**9*T**3+0.0307617188*EPS**6*T**2-0.09375*E
     1    PS**3*T+1
      T34 = T**2.5
      T35 = 2*(-1.0172526E-6*(5697*EPS**9*T**2.5-10080*EPS**6*T**1.5+921
     1    60*EPS**3*T**0.5)-1/T**0.5)/EPS**1.5
      T36 = COS(2*(-1.0172526E-6*(5697*EPS**9*T**2.5-10080*EPS**6*T**1.5
     1    +92160*EPS**3*T**0.5)-1/T**0.5)/EPS**1.5)
      T37 = -0.086929321*EPS**9*T**2
      T38 = 1/(-0.0289764404*EPS**9*T**3+0.0307617188*EPS**6*T**2-0.0937
     1    5*EPS**3*T+1)**0.5
      T39 = EPS**12
```

```
      I40 = -0.064788222*EPS**12*T**4-0.0289764404*EPS**9*T**3+0.03076171
     1   88*EPS**6*T**2-0.09375*EPS**3*T+1
      I41 = 2*(1.135326561-9*(8152245*EPS**12*T**3.5-5104512*EPS**9*T**2
     1    5+9031680*EPS**6*T**1.5-82575360*EPS**3*T**0.5)-1/T**0.5)/EPS*
     2   *1.5
      I42 = COS(2*(1.135326561-9*(8152245*EPS**12*T**3.5-5104512*EPS**9*
     1   T**2.5+9031680*EPS**6*T**1.5-82575360*EPS**3*T**0.5)-1/T**0.5)/
     2   EPS**1.5)
      I43 = 1/(0.064788222*EPS**12*T**4-0.0289764404*EPS**9*T**3+0.03076
     1   17188*EPS**6*T**2-0.09375*EPS**3*T+1)**0.5
      INV(1) = 0.5*EPS**1.5*(COS(2/(EPS**1.5*T**0.5))**2/(EPS**1.5*T**1.
     1   5)+(0.75*EPS**0.75*COS(2/(EPS**1.5*T**0.5))/T**0.25+SIN(2/(EPS*
     2   *1.5*T**0.5))/(EPS**0.75*T**0.75))**2)*T**1.5
      INV(2) = 0.5*EPS**1.5*T**1.5*((-SIN((-0.1875*EPS**3*T**0.5-2/T**0.
     1   5)/EPS**1.5)*(1/T**1.5-0.09375*EPS**3/T**0.5)*T**0.75/(EPS**0.7
     2   5*(1-0.09375*EPS**3*T)**0.5)+0.75*EPS**0.75*COS((-0.1875*EPS**3
     3   *T**0.5-2/T**0.5)/EPS**1.5)/(T**0.25*(1-0.09375*EPS**3*T)**0.5)
     4   +0.046875*EPS**3.75*COS((-0.1875*EPS**3*T**0.5-2/T**0.5)/EPS**1
     5   .5)*T**0.75/(1-0.09375*EPS**3*T)**1.5)**2+COS((-0.1875*EPS**3*T
     6   **0.5-2/T**0.5)/EPS**1.5)**2*(1-0.09375*EPS**3*T)/(EPS**1.5*T**
     7   1.5))/(1-0.09375*EPS**3*T)
      INV(3) = 0.5*EPS**1.5*T**1.5*((-2*(1.62760416E-4*(94.5*EPS**6*T**0
     1   .5-288*EPS**3/T**0.5)+0.5/T**1.5)*T**0.75*SIN(2*(1.62760416E-4*
     2   (63*EPS**6*T**1.5-576*EPS**3*T**0.5)-1/T**0.5)/EPS**1.5)/(EPS**
     3   0.75*(0.0307617188*EPS**6*T**2-0.09375*EPS**3*T+1)**0.5)+0.75*E
     4   PS**0.75*COS(2*(1.62760416E-4*(63*EPS**6*T**1.5-576*EPS**3*T**0
     5   .5)-1/T**0.5)/EPS**1.5)/(T**0.25*(0.0307617188*EPS**6*T**2-0.09
     6   375*EPS**3*T+1)**0.5)-0.5*EPS**0.75*T**0.75*(0.0615234375*EPS**
     7   6*T-0.09375*EPS**3)*COS(2*(1.62760416E-4*(63*EPS**6*T**1.5-576*
     8   EPS**3*T**0.5)-1/T**0.5)/EPS**1.5)/(0.0307617188*EPS**6*T**2-0.
     9   09375*EPS**3*T+1)**1.5)**2+(0.0307617188*EPS**6*T**2-0.09375*EP
     A   S**3*T+1)*COS(2*(1.62760416E-4*(63*EPS**6*T**1.5-576*EPS**3*T**
     B   0.5)-1/T**0.5)/EPS**1.5)**2/(EPS**1.5*T**1.5))/(0.0307617188*EP
     C   S**6*T**2-0.09375*EPS**3*T+1)
      INV(4) = 0.5*EPS**1.5*T**1.5*((-2*T**0.75*(0.5/T**1.5-1.0172526E-6
     1   *(14242.5*EPS**9*T**1.5-15120*EPS**6*T**0.5+46080*EPS**3/T**0.5
     2   ))*SIN(2*(-1.0172526E-6*(5697*EPS**9*T**2.5-10080*EPS**6*T**1.5
     3   +92160*EPS**3*T**0.5)-1/T**0.5)/EPS**1.5)/(EPS**0.75*(-0.028976
     4   4404*EPS**9*T**3+0.0307617188*EPS**6*T**2-0.09375*EPS**3*T+1)**
     5   0.5)+0.75*EPS**0.75*COS(2*(-1.0172526E-6*(5697*EPS**9*T**2.5-10
     6   080*EPS**6*T**1.5+92160*EPS**3*T**0.5)-1/T**0.5)/EPS**1.5)/(T**
     7   0.25*(-0.0289764404*EPS**9*T**3+0.0307617188*EPS**6*T**2-0.0937
     8   5*EPS**3*T+1)**0.5)-0.5*EPS**0.75*T**0.75*(-0.086929321*EPS**9*
     9   T**2+0.0615234375*EPS**6*T-0.09375*EPS**3)*COS(2*(-1.0172526E-6
     A   *(5697*EPS**9*T**2.5-10080*EPS**6*T**1.5+92160*EPS**3*T**0.5)-1
     B   /T**0.5)/EPS**1.5)/(-0.0289764404*EPS**9*T**3+0.0307617188*EPS*
     C   *6*T**2-0.09375*EPS**3*T+1)**1.5)**2+(-0.0289764404*EPS**9*T**3
```

```
D    +0.0307617188*EPS**6*I**2-0.09375*EPS**3*I+1)*COS(2*(-1.0172526
I    1.6*(5697*EPS**9*I**2.5-13080*EPS**6*I**1.5+9216*EPS**3*I**0.5
I    )-1/I**0.5)/EPS**1.5)**2/(EPS**1.5*I**1.5))/(-0.0289764404*EPS*
G    *9*I**3+0.0307617188*EPS**6*I**2-0.09375*EPS**3*I+1)
 INV(5) = 0.5*EPS**1.5*I**1.5*((-2*I**0.75*(1.13532656E-9*(28532857
1    .5*EPS**12*I**2.5-12761280*EPS**9*I**1.5+13547520*EPS**6*I**0.5
2    -4128768O*EPS**3/I**0.5)+0.5/I**1.5)*SIN(2*(1.13532656E-9*(8152
3    245*EPS**12*I**3.5-5104512*EPS**9*I**2.5+9031680*EPS**6*I**1.5-
4    82575360*EPS**3*I**0.5)-1/I**0.5)/EPS**1.5)/(EPS**0.75*(0.06478
5    8222*EPS**12*I**4-0.0289764404*EPS**9*I**3+0.0307617188*EPS**6*
6    I**2-0.09375*EPS**3*I+1)**0.5)+0.75*EPS**0.75*COS(2*(1.13532656
7    E-9*(8152245*EPS**12*I**3.5-51045.2*EPS**9*I**2.5+9031680*EPS**
8    6*I**1.5-82575360*EPS**3*I**0.5)-1/I**0.5)/EPS**1.5)/(I**0.25*(
9    0.064788222*EPS**12*I**4-0.0289764404*EPS**9*I**3+0.0307617188*
A    EPS**6*I**2-0.09375*EPS**3*I+1)**0.5)-0.5*EPS**0.75*I**0.75*(0.
B    25915289*EPS**12*I**3-0.086929321*EPS**9*I**2+0.0615234375*EPS*
C    *6*I-0.09375*EPS**3)*COS(2*(1.13532656E-9*(8152245*EPS**12*I**3
D    .5-5104512*EPS**9*I**2.5+9031680*EPS**6*I**1.5-82575360*EPS**3*
F    I**0.5)-1/I**0.5)/EPS**1.5)/(0.064788222*EPS**12*I**4-0.0289764
F    404*EPS**9*I**3+0.0307617188*EPS**6*I**2-0.09375*EPS**3*I+1)**1
G    .5)**2+(0.064788222*EPS**12*I**4-0.0289764404*EPS**9*I**3+0.030
H    7617188*EPS**6*I**2-0.09375*EPS**3*I+1)*COS(2*(1.13532656E-9*(8
I    152245*EPS**12*I**3.5-5104512*EPS**9*I**2.5+9031680*EPS**6*I**1
J    .5-82575360*EPS**3*I**0.5)-1/I**0.5)/EPS**1.5)**2/(EPS**1.5*I**
K    1.5))/(0.064788222*EPS**12*I**4-0.0289764404*EPS**9*I**3+0.0307
L    617188*EPS**6*I**2-0.09375*EPS**3*I+1)
```

APPENDIX C. ALGORITHM FOR CALCULATING Y_{2N}

It is worthwhile to compare times for Y_{2N} using several different symbolic computing implementations. Three different algorithms were used. The first was a naive attempt at at finding the Y_{2n} emphasizing recursion and is not displayed. The second version was sophisticated and used MACSYMA array functions to hold intermediate results. A REDUCE version of this algorithm is also included. The third was a solution that Hearn published to this problem in 1972 in REDUCE (Hearn, 1972).

The algorithms are:

MACSYMA Fragment #4: Source for Y_{2N} Calculation

```
gradef(e(n,z),z,e(n+1,z));

sum1[k]:=block([n:k/2,s:0],modedeclare([n,k,be],fixnum),
       for al:1 thru n-1 do (
           be:n-al,
           s:s+y[2*al]*y[2*be]),s);

sum2[k]:=block([n:k/2,s:0], modedeclare([n,k,del],fixnum),
       for al:0 thru n do (
         for be:0 thru n-al do (
           for gam:0 thru n-al-be do (
               del:n-al-be-gam,(
   if
    (al+be+del)*(al+be+gam)*(be+gam+del)*(al+gam+del)=0 then
                    done
   else  (
         if al+be+gam+del#n then
                     done
         else (
              s:s+y[2*al]*y[2*be]*y[2*gam]*y[2*del])))))),s);

/* check in sum2 that at least 2 indices are nonzero...must fail
      when at least three are 0. there are 4 cases when three
      indices are 0...and they are test for explicitly...
      the test that all four indices = 0 is redundant. */

sum3[k]:=block([n:k/2,s:0],modedeclare([k,n,be],fixnum),
      for al:1 thru n-2 do (
          be:n-1-al,
          (s:s+e(0,z)*y[2*al]*y[2*be] +3/4*dr[2*al]*dr[2*be]-
            1/4*(y[2*al]*dr2[2*be]+dr2[2*al]*y[2*be]))),s);

dr[n]:=diff(y[n],z);
dr2[n]:=diff(y[n],z,2);

y[n]:=ratsimp(e(0,z)*y[n-2]-1/4*dr2[n-2]
  +1/2*sum1[n]-1/2*sum2[n] +1/2*sum3[n]);

y[0]:1;
y[2]:1/2*e(0,z);
```

REDUCE Fragment #1: Calculation of Y_{2N}

```
comment store derivatives and y;
    array y(30),dr(30),dr2(30);

comment set up the differentiation rule;
    for all n,x let df(e(n,x),x)=e(n+1,x);

comment calculate the first sum;
    algebraic procedure sum1(k);
    begin  integer n,k,al,be; scalar s;
        n:=k/2;  s:=0;
        for al:=1:n-1 do  begin
             be:=n-al;  s:=s+ y(2*al)*y(2*be);
        end;   return s;
     end;

 comment the second sum (quadrilinear one);
   algebraic procedure sum2(k);
      begin integer n,k,al,be,gam,del;  scalar s;
      n:=k/2;    s:=0;
      for al:=0:n do
          begin
          for be:=0:n-al do begin
              for gam:=0:n-al-be do begin
                 del:=n-al-be-gam;
                 if
       (al+be+del)*(al+be+gam)*(be+gam+del)*(al+gam+del)=0
              then go to 11;
                 s:=s+ y(2*al)*y(2*be)*y(2*gam)*y(2*del);
          11:    end;     end;  end;
      return s;
   end;

 comment the third sum;
   algebraic procedure sum3(k);
      begin integer n,k,al,be;  scalar s;
        n:=k/2;   s:=0;
           for al:=1:n-2 do  begin
               be:=n-1-al;
```

```
                s:=s +e(0,z)*y(2*al)*y(2*be) + 3/4 *dr(2*al)*dr(2*be)
                - 1/4*(y(2*al)*dr2(2*be)+dr2(2*al)*y(2*be));
        end;   return s;
    end;

comment initialize the first 2 y's;
    y(0):=1;
    dr(0):=0;
    dr2(0):=0;
    y(2):=e(0,z)/2;
    dr(2):=df(y(2),z);
    dr2(2):=df(dr(2),z);


 algebraic procedure ycal(n);
    begin integer n;
    y(n):=e(0,z)*y(n-2)-1/4*dr2(n-2)
          +1/2*sum1(n)-1/2*sum2(n)+1/2*sum3(n);
    dr(n):=df(y(n),z);
    dr2(n):=df(dr(n),z);
    return y(n); end;

procedure doit(j);
   begin integer n,j;
   showtime;
   for n:=4 step 2 until j do begin
       ycal(n);
       showtime;
       write "n =",n," y(",n,")=",ycal(n);
       showtime;
   end;   end;
```

EVALUATING INFINITE INTEGRALS USING MACSYMA

Elizabeth Cuthill
David W. Taylor Naval Ship Research and Development Center
Bethesda, Maryland 20084

Abstract

MACSYMA has been of great value to our work in developing special approximation formulas and improved approximations for some infinite integrals representing spectral densities. MACSYMA readily provided plots of our integrands permitting a view of the behavior of the functions occuring in the integrals. Integration formulas especially adapted to the various forms of our integrals were generated using MACSYMA. These were used to obtain estimates for our integrals. Folding and extrapolation techniques were incorporated to obtain improved results.

Numerical integration algorithms normally see values of the integrands only at discrete values of the independent variable within the interval of integration and estimate the integral by calculating a weighted average of the corresponding function values. If the starting sample of values of the independent variable is an unfortunate one, the behavior of the integrand inferred from this sample may be very different from that of the integrand itself. This may happen for integrands characterized by rapid changes in function values and their derivatives within the interval of integration. Our integrands frequently have this behavior and use of adaptive routines without sufficient care can require excessive running times and in some cases produce wrong results.

Several examples of integrals illustrating some of this bothersome behavior are considered. The application of MACSYMA in generating more reliable approximation formulas and numerical approximations for these examples will be presented.

RESULTS IN UNEXPECTED MACSYMA IMPLEMENTATION ENVIRONMENTS

George J. Carrette
MIT
Cambridge, MA 02139
and
Paradigm Associates, Inc.
29 Putnam Ave. Suite 6
Cambridge, MA 02139

Abstract

Changes to the Macsyma evaluator and pattern matcher required to transport Macsyma to the lexically scoped environment of MIT's Common-Lisp implementation VAX-NIL are described.

Small changes to Macsyma static data representation and program dispatch methods are described which take advantage of efficient array structures and dispatch operations implemented on the VAX.

Special compilation schemes for parts of Macsyma are considered, including a "macsyma machine" with instructions generated by the use of lisp microcompilation.

Runtime and code-size results of the work are given in all areas.

Acknowledgments

The author wishes to thank Glenn Burke of MIT and Dr. Grant Cook of LLNL for their interest and comments on this work. Portions of this work were supported by DOE Contract W-7405-ENG-48.

STABILITY CRITERIA FOR FINITE DIFFERENCE EQUATIONS

Dr. J. J. Yagla
Engineering Department
Naval Surface Weapons Center
Dahlgren, Virginia 22448

ABSTRACT

Macsyma was used to develop stability criteria for a finite difference method for nonsteady compressible flow in two dimensions. The finite difference method is an adaptation of the two-step Lax-Wendroff method to curvilinear coordinate systems. Instabilities were encountered for large turning angles of the flow. Reduction of the time step of the integration had no apparent effect on the instability. Therefore a formal analysis of stability was indicated. The stability analysis required determining the spectral radius of a rather cumbersome 4 x 4 matrix whose elements were algebraic combinations of the variables (density, pressure, velocity) of the flow field. Through defining new variables and a rotation of coordinates, a form of the characteristic polynomial of the matrix was found that could be simplified with built-in Macsyma routines. The desired stability criteria were then readily determined. This work has resulted in a direct calculation of necessary conditions for stability of an important class of difference equations, where before the stability criteria were inferred from analysis of an equivalent system of equations.

1. INTRODUCTION

This paper describes a formal calculation of stability criteria for the Lax-Wendroff two-step finite difference method for the equations of gas dynamics. The approach would apply also to other areas of mathematical physics that require numerical solutions to hyperbolic or parabolic systems of partial differential equations. The Lax-Wendroff method has been widely used for compressible flow. An adaptation of the method to curvilinear coordinate systems correctly solved a variety of test problems for both steady and nonsteady flows in two space dimensions. However, the method became unstable for flows that turned through angles greater than twenty three degrees. Reduction of the time step of the integration, refinement of the grid, an alternate difference technique (MacCormack, 1969) and various boundary treatments failed to correct the instability. Therefore a formal analysis to obtain stability criteria was undertaken.

Macsyma made the analysis of stability tractable. First, Macsyma differentiation of matrices was used advantageously in constructing the amplification matrix for the finite difference scheme. The stability analysis then required computing eigenvalues of the amplification matrix. The characteristic polynomial has hundreds of terms and factors, many of which are trigonometric functions. Hand calculations with 3 x 3 systems, and the 4 x 4 system that results for Cartesian coordinates, showed that a formal calculation of the 4 x 4 case in curvilinear coordinates would take months or years, and errors would be likely. The problem was solved by Macsyma in less than 500 seconds of computer time.

This paper describes the differential equations, approximate finite difference equations, and the stability analysis. The application of Macsyma and the results obtained are discussed.

2. CONSERVATION LAW FORM

The differential equations of gas dynamics for nonsteady flow of an ideal gas can be written as

$$\frac{\partial U}{\partial t} + \frac{\partial}{\partial x} F(U) + \frac{\partial}{\partial y} G(U) + \frac{\partial}{\partial z} H(U) = 0. \tag{1}$$

For two-dimensional flow one has $\partial/\partial z = 0$. Here, U, F, and G are column vectors defined as follows:

$$U = \begin{pmatrix} \rho^* \\ \rho^* \left(\frac{\partial \xi}{\partial x} u + \frac{\partial \xi}{\partial y} v \right) \\ \rho^* \left(\frac{\partial \eta}{\partial x} u + \frac{\partial \eta}{\partial y} v \right) \\ E^* \end{pmatrix} \tag{2}$$

$$F(U) = \begin{pmatrix} \rho^* u \\ \rho^*\left(\frac{\partial \xi}{\partial x} u^2 + \frac{\partial \xi}{\partial y} uv\right) + \frac{\partial x}{\partial \xi} P^* \\ \rho^*\left(\frac{\partial \eta}{\partial x} u^2 + \frac{\partial \eta}{\partial y} uv\right) + \frac{\partial x}{\partial \eta} P^* \\ u\,(E^* + P^*) \end{pmatrix} \tag{3}$$

$$G(U) = \begin{pmatrix} \rho^* v \\ \rho^*\left(\frac{\partial \xi}{\partial y} v^2 + \frac{\partial \xi}{\partial x} uv\right) + \frac{\partial y}{\partial \xi} P^* \\ \rho^*\left(\frac{\partial \eta}{\partial y} v^2 + \frac{\partial \eta}{\partial x} uv\right) + \frac{\partial y}{\partial \eta} P^* \\ v\,(E^* + P^*) \end{pmatrix} \tag{4}$$

$$P^* = (\gamma - 1)\,\rho^*\left[\frac{E^*}{\rho^*} - \frac{1}{2}\sqrt{g}\,(u^2 + v^2)\right]. \tag{5}$$

In the above equations u and v are contravariant velocity components in the (x,y) curvilinear coordinate system. The curvilinear coordinates are related to the (ξ,η) Cartesian coordinates through a conformal transformation. The starred quantities are the usual physical quantities multiplied by $\sqrt{g}$, where $\sqrt{g}$ is the Jacobian of the coordinate transformation. The physical quantities ρ, E, and P are the density, total energy per unit volume, and pressure, respectively. I have called these scalar physical quantities multiplied by $\sqrt{g}$ "tensor densities" of the corresponding physical quantities. Special properties of conformal mapping, e.g. the Cauchy-Riemann conditions and the harmonic property, have been used to simplify the above equations.

Figure 1 shows three representations of a curved duct in the Cartesian and curvilinear coordinate systems. Figure 2 shows the representation of a fluid velocity vector in terms of physical and contravariant components along the curvilinear coordinate lines. The independent variables in the technique are the curvilinear coordinates (x,y). The dependent variables of the technique

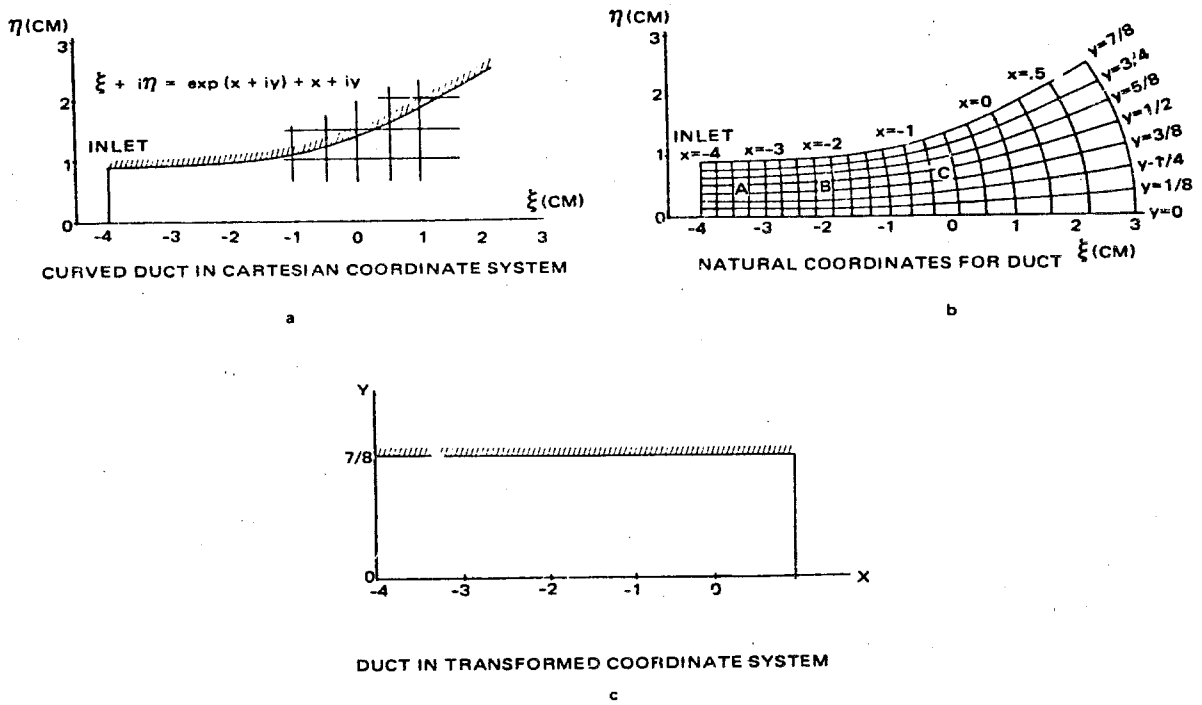


Fig. 1. Representation of duct in different coordinate systems

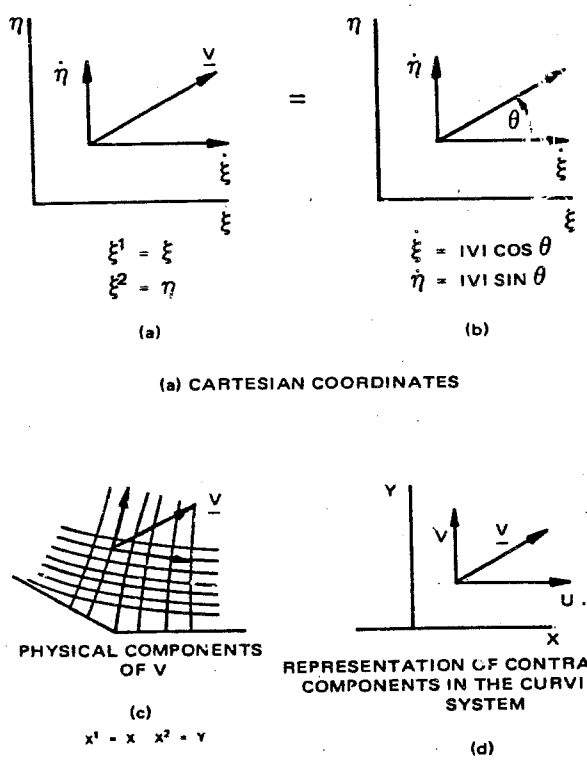


Fig. 2. Representations of the velocity vector

are the contravariant velocities u and v, and the tensor densities of the fluid density, total energy per unit volume, and pressure. Finite difference equations for solving these equations are presented in the following section.

3. FINITE DIFFERENCE EQUATIONS

The first attempt at approximating the above differential equations with finite differences failed. The two-step Lax-Wendroff finite difference scheme (Richtmyer and Morton, 1967), originally developed for Cartesian coordinates was used as a prototype method for the conservation law form equations in curvilinear coordinates. The approach worked under the trivial coordinate transformation, i.e. $w = z$, where $w = \xi + i\eta$ and $z = x + iy$. However, under the nontrivial coordinate transformation $w = e^z + z$, for a duct with a curved wall, Figure 1, the method failed. Flows spontaneously arose at interior points of regions of the flow field having an initially uniform quiescent state. The source of error was found, and could be alleviated through a slight modification of the Cartesian form of the finite difference equations (Yagla, 1972). The modified two-step equations are

$$U^{n+1}_{j,k} = \frac{1}{4} J_{j,k} \left[\frac{U^n_{j+1,k}}{J_{j+1,k}} + \frac{U^n_{j-1,k}}{J_{j-1,k}} + \frac{U^n_{j,k+1}}{J_{j,k+1}} + \frac{U^n_{n,k-1}}{J_{j,k-1}} \right]$$

$$- \frac{\Delta t}{2\Delta x} (F^n_{j+1,k} - F^n_{j-1,k}) - \frac{\Delta t}{2\Delta y} (G^n_{j,k+1} - G^n_{j,k-1}), \tag{6}$$

and

$$U^{n+2}_{j,k} = U^n_{j,k} - \frac{\Delta t}{\Delta x} (F^{n+1}_{j+1,k} - F^{n+1}_{j-1,k}) - \frac{\Delta t}{\Delta y} (G^{n+1}_{j,k+1} - G^{n+1}_{j,k-1}) \tag{7}$$

where J is the Jacobian of the transformation. For the trivial coordinate transformation one has J=1, and the usual Cartesian two-step equations are recovered. Several test problems were solved with good results using this "generalized" two-step finite difference technique.

However, when calculating supersonic flow in a corner as shown in Figure 2(c), a numerical instability occurred whenever the turning angle exceeded twenty-three degrees. Numerous attempts to correct the instability failed. Therefore the formal analysis of stability described below was undertaken.

4. STABILITY ANALYSIS

4.1 Operator Form of Equations

The curvilinear conservation law is written in the form

$$\frac{\partial U}{\partial t} + \frac{\partial F(U)}{\partial x} + \frac{\partial G(U)}{\partial y} = 0 \quad . \tag{8}$$

Jacobian matrices $A = \partial F/\partial U$ and $B = \partial G/\partial U$ are then defined so that the conservation law can be written

$$\frac{\partial U}{\partial t} + A\frac{\partial U}{\partial x} + B\frac{\partial U}{\partial y} = 0 \quad . \tag{9}$$

This has the form

$$\frac{\partial U}{\partial t} - LU = 0 \quad , \tag{10}$$

where the operator L is defined by

$$L = -\left[A\frac{\partial}{\partial x} + B\frac{\partial}{\partial y}\right] \quad . \tag{11}$$

The Lax-Wendroff two-step finite difference equations are of the form

$$U^{n+1} = D\,(U^n, \Delta x, \Delta y, \Delta t)\,U^n \quad , \tag{12}$$

where D is a finite difference operator. When considering consistency and stability one assumes an infinite sequence of calculations with increasingly refined meshes. Stability limits, however, dictate that the time step be reduced whenever the spacial increment is reduced. Therefore relationships $\Delta x = f_1(\Delta t)$ and $\Delta y = f_2(\Delta t)$ must be satisfied for stability in a sequence of calculations with finer and finer meshes. Therefore one can write

$$U^{n+1} = D\,(U^n, \Delta t)\,U^n \quad . \tag{13}$$

The time derivative of U in terms of the finite difference operator is

$$\frac{\partial U}{\partial t} = \lim_{\Delta t \to 0}\left[\frac{U^{n+1} - U^n}{\Delta t}\right] = \lim_{\Delta t \to 0}\left[\frac{D - I}{\Delta t}\right] U^n \quad , \tag{14}$$

where I is the identity operator.

Thus the quantity (D-I)/Δt must be an approximation to spacial operator L of the original differential equation. The finite difference equations are said to be consistent (Richtmyer, 1957) with the differential equations if

$$\lim_{\Delta t \to 0} \left\| \left(\frac{D - I}{\Delta t} - L \right) U \right\| = 0 \quad . \tag{15}$$

The double bars indicate a norm of the solution space. The quantity between the bars is the truncation error for Δt. The differential operator D is stable if there exists a quantity τ such that the infinite set of operators $D(\Delta t)^n$ is uniformly bounded for $0 < \Delta t < \tau$ and $0 \le n\Delta t \le T$.

The generalized Lax-Wendroff two-step difference equations (6) and (7) can be written as a single equation for the special case $\mathbf{F} = \mathbf{AU}$ and $\mathbf{G} = \mathbf{BU}$ as

$$\begin{aligned}
U^{n+2}_{j,k} = U^n_{j,k} &- \frac{1}{4}\frac{\Delta t}{\Delta x} A \left[J_{j+1,k} \left(\frac{U^n_{j+2,k}}{J_{j+2,k}} + \frac{U^n_{j,k}}{J_{j,k}} + \frac{U^n_{j+1,k+1}}{J_{j+1,k+1}} + \frac{U^n_{j+1,k-1}}{J_{j+1,k-1}} \right) \right. \\
&\left. - J_{j-1,k} \left(\frac{U^n_{j,k}}{J_{j,k}} + \frac{U^n_{j-2,k}}{J_{j-2,k}} + \frac{U^n_{j-1,k+1}}{J_{j-1,k+1}} + \frac{U^n_{j-1,k-1}}{J_{j-1,k-1}} \right) \right] \\
&- \frac{1}{4}\frac{\Delta t}{\Delta y} B \left[J_{j,k+1} \left(\frac{U^n_{j+1,k+1}}{J_{j+1,k+1}} + \frac{U^n_{j-1,k+1}}{J_{j-1,k+1}} + \frac{U^n_{j,k+2}}{J_{j,k+2}} + \frac{U^n_{j,k}}{J_{j,k}} \right) \right. \\
&\left. - J_{j,k-1} \left(\frac{U^n_{j+1,k-1}}{J_{j+1,k-1}} + \frac{U^n_{j-1,k-1}}{J_{j-1,k-1}} + \frac{U^n_{j,k}}{J_{j,k}} + \frac{U^n_{j,k-2}}{J_{j,k-2}} \right) \right] \\
&+ \frac{\Delta t}{\Delta x} A \left[\frac{\Delta t}{2\Delta x} A \left(U^n_{j+2,k} - 2U^n_{j,k} - U^n_{j-2,k} \right) \right. \\
&\left. + \frac{\Delta t}{2\Delta y} B \left(U^n_{j+1,k+1} - U^n_{j+1,k-1} - U^n_{j-1,k+1} + U^n_{j-1,k-1} \right) \right] \\
&+ \frac{\Delta t}{\Delta y} B \left[\frac{\Delta t}{2\Delta x} A \left(U^n_{j+1,k+1} - U^n_{j-1,k+1} - U^n_{j+1,k-1} + U^n_{j-1,k-1} \right) \right. \\
&\left. + \frac{\Delta t B}{2\Delta y} \left(U^n_{j,k+2} - 2U^n_{j,k} + U^n_{j,k-2} \right) \right] .
\end{aligned} \tag{16}$$

As will be seen later, the Jacobian matrices $A = \partial F/\partial U$ and $B = \partial G/\partial U$ are especially complicated. The special case $\mathbf{F} = A\mathbf{U}$ and $\mathbf{G} = B\mathbf{U}$ occurs when **F** and **G** are homogeneous functions of the elements of **U**, that is $\mathbf{F}(k\mathbf{U}) = k\mathbf{F}(\mathbf{U})$ and $\mathbf{G}(k\mathbf{U}) = k\mathbf{G}(\mathbf{U})$ for any k (Courant 1947, Warming

and Beam 1978). Remarkably, the functions **F** and **G** in the conservation law form of the equations of gas dynamics are homogeneous, provided that the equation for P (the equation of state) is linear in the density.

4.2 General Approach to Stability Analysis

The stability analysis for the generalized system of finite difference equations follows the method of J. von Neumann. Conceptually, one assumes that the solution in the time domain can be Fourier transformed to a frequency domain. Under Fourier transformation, the solution vector **U** at some fixed time has the form

$$\mathbf{U}(x,y) = \frac{1}{L_x L_y} \sum\sum \hat{\mathbf{U}}(k_x, k_y)\, e^{ik_x x}\, e^{ik_y y} \quad , \tag{17}$$

where the $\hat{\mathbf{U}}(k_x, k_y)$ are Fourier coefficients that correspond to each of the wave numbers k_x and k_y. The wave numbers are integral multiples of $2\pi/L_x$ and $2\pi/L_y$, where L_x and L_y are the periods of **U** in the x and y directions. The power of the method lies in the fact that difference formulae for the spacial derivatives can be factored, e.g., $\mathbf{U}(x+\Delta x,y) = \mathbf{U}(x,y)e^{ik_x\Delta x}$, and the resulting expressions can be simplified by using complex polar forms of the trigonometric and hyperbolic functions.

For a spacial operator L

$$\mathrm{L}\,\mathbf{U}(x,y,t) = \frac{1}{L_x L_y} \sum\sum \hat{\mathbf{U}}(k_x, k_y, t)\, \mathrm{L} e^{ik_x x}\, e^{ik_y y}$$

and a conservation law of the form $\partial \mathbf{U}/\partial t = \mathbf{L}\mathbf{U}$, the Fourier transformed solution must satisfy

$$\frac{1}{L_x L_y} \sum\sum \frac{\partial}{\partial t}\, \hat{\mathbf{U}}(k_x, k_y, t)\, e^{ik_x x}\, e^{ik_y y} = \frac{1}{L_x L_y} \sum\sum \hat{\mathbf{U}}(k_x, k_y, t)\, \mathrm{L} e^{ik_x x}\, e^{ik_y y} \quad .$$

Therefore the Fourier components satisfy

$$\frac{\partial}{\partial t}\, \hat{\mathbf{U}}(k_x, k_y, t)\, e^{ik_x x}\, e^{ik_y y} = \hat{\mathbf{U}}\, \mathrm{L} e^{ik_x x}\, e^{ik_y y} \quad .$$

The behavior of the finite difference solution to the original field equations can be anticipated through a study of the behavior of the Fourier coefficients under the same finite difference operators. Due to the factoring property in the frequency domain, the right-hand side of the above equation takes the form

$$\hat{U}\, L e^{ik_x x} e^{ik_y y} = \hat{U} e^{ik_x x} e^{ik_y y}\, H(k_x \Delta x, k_y \Delta y)$$

This defines a new matrix H that is used in the stability analysis.

The transformed differential equation for a Fourier component, becomes

$$\frac{\partial \hat{U}}{\partial t} = H(k_x \Delta x, k_y \Delta y)\, \hat{U} \quad . \tag{18}$$

The general solution to this first order differential equation is

$$\hat{U}(k_x, k_y, t) = \hat{U}(k_x, k_y, 0)\ \exp(Ht) \quad . \tag{19}$$

The function exp M, where M is a matrix, is defined by the series: $I + M + M^2/2! + M^3/3! + \ldots +$. This is an explicit formula for the solution at time t in the frequency domain. It could, in principle, be constructed and transformed back to the physical domain. This is not done in practice because there is no efficient means of evaluating exp(Ht). The solution at $t = \Delta t$ can be approximated by

$$\hat{U}(k_x, k_y, \Delta t) = U(k_x, k_y, 0)(I + H\Delta t) \quad ,$$

similarly

$$\hat{U}(k_x, k_y, 2\Delta t) = \hat{U}(k_x, k_y, \Delta t)(I + H\Delta t) = \hat{U}(k_x, k_y, 0)\ (I + H\Delta t)^2 \quad ,$$

and by induction

$$\hat{U}(k_x, k_y, n\Delta t) = \hat{U}(k_x, k_y, 0)(I + H\Delta t)^n \quad .$$

Therefore the Fourier coefficients of the solution remain bounded if $(I + H\Delta t)^n$ remains bounded. The quantity $[I + H(k_x \Delta x, k_y \Delta y)]$ is called the amplification matrix, and is the frequency domain image of the finite difference operator $D(\Delta x, \Delta y, \Delta t)$ in equation (12).

Following equation (15) it was stated that the difference operator D (the finite difference scheme) is stable if there exists some τ for which the infinite sequence of operators D^n is uniformly bounded for $0 < \Delta t < \tau$, $0 \leq n\Delta t \leq T$. The stability condition in the frequency domain is therefore that $(I + H\Delta t)^n$ must be uniformly bounded. The von Neumann necessary condition for stability is that all of the eigenvalues of the amplification matrix satisfy

$$|\lambda_i| \leq 1 + O(\Delta t) \tag{20}$$

for all wave numbers k_x and k_y and $0 < \Delta t < \tau$. Sufficient conditions for stability are not so easy to ob-

tain; however, Richtmyer (1957) states that in all cases investigated so far, the von Nuemann condition has always turned out to be sufficient as well as necessary.

Thus, the stability of the generalized Lax-Wendroff two-step method can be determined by first constructing the amplification matrix, then determining its eigenvalues. A comparison of the expression for the eigenvalues with equation (20) then gives the stability criteria for the numerical method. The amplification matrix for the generalized two-step method is presented in the next section. Computations of the eigenvalues and determination of the stability criteria are presented in the final section.

4.3 Computation of the Amplification Matrix

The amplification matrix is computed by substituting a vector Fourier coefficient into the finite difference equations. A Fourier element of the solution at time $t = n\Delta t$ can be defined by

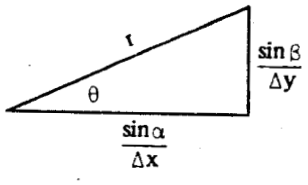
$$U^n_{j,k} = \hat{U}(k_x, k_y, t)\, e^{i(k_x x + k_y y)}, \tag{21}$$

where k_x and k_y are wave numbers as defined before. When the Fourier element is substituted into equation (16), the result can be factored to obtain:

$$U^{n+1}_{j,k} = U^n_{j,k}\left[I - \Delta t\left(\Delta x \frac{\partial \log J}{\partial x} \sin\alpha + \Delta y \frac{\partial \log J}{\partial y} \sin\beta \right)\left(A\frac{\sin\alpha}{\Delta x} + B\frac{\sin\beta}{\Delta x} \right) - i\Delta t\left(\cos\alpha + \cos\beta \right)\left(A\frac{\sin\alpha}{\Delta x} + B\frac{\sin\beta}{\Delta x} \right) - 2\Delta t^2\left(A\frac{\sin\alpha}{\Delta x} + B\frac{\sin\beta}{\Delta y} \right)^2 \right]. \tag{22}$$

The Jacobian matrices A and B are as defined before.

The substitutions $\alpha = k_x\Delta x$ and $\beta = k_y\Delta y$ have been used. The terms with derivatives of the Jacobian are the result of substituting leading terms of the Taylor expansions for the Jacobians of the coordinate transformations, resulting in terms of the form $\Delta x(\partial J/\partial x)/J = \Delta x\partial(\log J)/\partial x$. The quantity in square brackets is the amplification matrix. The von Neumann stability criteria is that the eigenvalues of the amplification matrix satisfy equation (20) above. The computation of the eigenvalues can be facilitated by defining a direction for $\sin\alpha/\Delta X$ and $\sin\beta/\Delta X$ according to the figure below.



$$\cos\theta = \frac{\sin\alpha}{r\Delta x}$$

$$\sin\theta = \frac{\sin\beta}{r\Delta y}$$

$$r^2 = \frac{\sin^2\alpha}{(\Delta x)^2} + \frac{\sin^2\beta}{(\Delta y)^2} \tag{23}$$

A new matrix C can then be defined by $C = A\cos\theta + B\sin\theta$. Then the amplification matrix can be written as

$$1 - \left(\Delta x \frac{\partial \log J}{\partial x} \sin\alpha + \Delta y \frac{\partial \log J}{\partial y} \sin\beta + i\cos\alpha + i\cos\beta\right) r\Delta t\, C - r^2\Delta t^2 C^2 \quad . \tag{24}$$

According to a theorem on matrices, the eigenvalues of the amplification matrix can be obtained from the eigenvalues of C by the following polynomial:

$$p = 1 - \left(\Delta x \frac{\partial \log J}{\partial x} \sin\alpha + \Delta y \frac{\partial \log J}{\partial y} \sin\beta + i\cos\alpha + i\cos\beta\right) r\Delta t\lambda - 2r^2\Delta t^2\lambda^2 \quad ,$$

where λ is an eigenvalue of C. The quantities $r^2\Delta t^2$ and $r\Delta t$ are of order of the trigonometric quantities in the limit of $\Delta x, \Delta y$, and Δt approaching zero for fixed ratios of $\Delta t/\Delta x$ and $\Delta t/\Delta y$. For example,

$$r^2\Delta t^2 = \left(\frac{\Delta t}{\Delta x}\right)^2 \sin^2\alpha + \left(\frac{\Delta t}{\Delta y}\right)^2 \sin^2\beta \quad . \tag{25}$$

However, the real part of the coefficient of λ in the equation giving the eigenvalues of the amplification matrix is of smaller order, being of Δx or Δy times the trigonometric quantities. Furthermore, the Jacobian of the coordinate transformation is a very smooth function. Thus with good approx-

imation, the eigenvalues of the amplification matrix can be computed from the eigenvalues of C by the polynomial

$$p = 1 - i(\cos\alpha + \cos\beta) r\Delta t\lambda - 2r^2\Delta t^2\lambda^2 \quad . \tag{26}$$

The modulus of an eigenvector p is then

$$|p| = (1 - 2\mu^2)^2 + (\cos\alpha + \cos\beta)^2\mu^2 \quad , \tag{27}$$

where $\mu = r\Delta t\lambda$. The next section deals with calculating the eigenvalues of C.

4.4 Eigenvalues of the Characteristic Equation

This section is devoted to the calculation of the eigenvalues of the matrix

$$C = A\cos\theta + B\sin\theta \quad ,$$

where A and B are the Jacobian matrices whose elements are

$$A_{ij} = \partial F_i / \partial U_j \quad \text{and} \quad B_{ij} = \partial G_i / \partial U_j \quad ,$$

where F and G are the fluxes in terms of the curvilinear coordinate system. Viviand (1974) has shown that the conservation law form in curvilinear coordinates can be written as

$$\frac{\partial}{\partial t}(JU) + \frac{\partial}{\partial x}\left(\frac{\partial x}{\partial \xi} JF + \frac{\partial x}{\partial \eta} JG\right) + \frac{\partial}{\partial y}\left(\frac{\partial y}{\partial \xi} JF + \frac{\partial y}{\partial \eta} JG\right) = 0 \quad . \tag{28}$$

The coordinate transformation equations are $x = x(\xi,\eta)$ and $y = y(\xi,\eta)$. F and G are the fluxes in the Cartesian directions ξ and η respectively. The derivatives $\partial F/\partial x$, $\partial G/\partial x$, $\partial F/\partial y$, and $\partial G/\partial y$ can be expanded by means of the chain rule as follows:

$$\frac{\partial F}{\partial x} = \frac{\partial F}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial F}{\partial \eta}\frac{\partial \eta}{\partial x} \quad . \tag{29}$$

When equation (28) is expanded using the chain rule, and the results simplified using the Cauchy-Rieman conditions, the result is

$$\frac{\partial}{\partial t}(JU) + JA\frac{\partial U}{\partial \xi} + JB\frac{\partial U}{\partial \eta} = 0 \quad , \tag{30}$$

where A and B are matrices with components given by $\partial F_i/\partial U_i$ and $\partial G_i/\partial U_j$, respectively. Therefore the study of the stability of the curvilinear system (28) is formally identical to the study of the stability of (30). Since the Jacobian is independent of time,

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{A}\,\frac{\partial \mathbf{U}}{\partial \xi} + \mathbf{B}\,\frac{\partial \mathbf{U}}{\partial \eta} = 0 \quad . \tag{31}$$

Warming and Beam (1978) obtained the same result for a general coordinate transformation after a linearization. With analytic transformations no linearization is required.

The calculation of the elements of the Jacobian matrices $\mathbf{A} = \partial \mathbf{F}/\partial \mathbf{U}$ and $\mathbf{B} = \partial \mathbf{G}/\partial \mathbf{U}$ is especially laborious. The matrices were recently presented by Warming and Beam (1978). However, they found direct calculation of the eigenvalues to be too difficult and instead used other arguments, as in Richtmyer and Morton (1967), to infer the eigenvalues from the original system of differential equations in primitive variables form. This paper shows a convenient means of calculating the Jacobian matrices of the conservation law form (rather than the primitive variable form) using computer calculus and algebra, and provides a direct algebraic calculation of eigenvalues for the amplification matrix of the von Nuemann stability analysis. With the method of this paper, one could expect to be able to determine stability criteria for even more complicated systems, such as ones that result when the number of spacial dimensions is increased, or other difference methods resulting in more complicated formulae.

The technique employs the computer program "Macsyma" which is in operation at the Massachusetts Institute of Technology. The Macsyma calculations described below were performed on a remote terminal of the Advanced Research Projects Agency Network (ARPA Net) using a terminal located at the David Taylor Naval Ship Research and Development Center, Washington, D.C. Mr. Kent Meals assisted in the calculation and operated the terminal.

5. MACSYMA CACULATIONS

The components of F and G were loaded into Macsyma. Macsyma was then instructed to form the matrices A and B by carrying out the partial differentiations $A_{ij} = \partial F_i/\partial U_j$ and $B_{ij} = \partial G_i/$

∂U_j. The ideal gas caloric equation of state was given to Macsyma as a means for computing the pressure from the independent variables ρ, m, n, and E. Macsyma was then instructed to construct the matrix

$$C = A \cos \theta + B \sin \theta \tag{32}$$

as appears in the amplification matrix of the von Nuemann stability analysis. Macsyma was then instructed to form the characteristic polynomial of the matrix C. The resulting expression for the characteristic polynomial contains 614 terms and factors involving the eigenvalues, the independent variables ρ, m, n, and E, the constants γ and θ, and trigonometric functions. The enormous labor of calculating the characteristic polynomial was done in just a few seconds by Macsyma.

The number of independent variables in the polynomial was reduced by commanding the substitutions

$$m = \rho u \quad , \tag{33}$$

$$n = \rho v \quad , \tag{34}$$

$$E = \left[\frac{1}{\gamma(\gamma - 1)} a^2 + \frac{1}{2}(u^2 + v^2)\right]\rho \quad . \tag{35}$$

Here the letter a denotes the speed of sound using an ideal gas equation of state. A preliminary study of the nonsteady, one-dimensional case motivated these substitutions. After this change the density could be factored out of the characteristic polynomial, and only velocity-like terms remained as independent variables. These substitutions reduced the number of terms in the characteristic polynomial by about thirty percent. Macsyma was then instructed to solve the characteristic polynomial for the roots $\lambda_1, \lambda_2, \lambda_3$, and λ_4. The instruction failed because the computer ran out of memory space.

After several attempts at algebraic reduction using built-in Macsyma routines, the following reduced form of the polynomial was obtained:

$$
\begin{aligned}
(D28)\ & (6\cos^4(TH) - 6\cos^6(TH) + 2\cos^4(TH)\sin^2(TH) - 6\cos^4(TH)\sin^4(TH) + \sin^4(TH))\,VP^4 \\
& + ((-12\cos^3(TH)\sin^5(TH) + 12\cos^3(TH)\sin^3(TH) \\
& + (12\cos^7(TH) - 12\cos^5(TH))\sin(TH))\,UP - 12L\cos^3(TH)\sin^3(TH) \\
& + (12L\cos^3(TH) - 12L\cos^5(TH))\sin(TH))\,VP^3 \\
& + ((-6\cos^2(TH)\sin^6(TH) + (18\cos^4(TH) + 6\cos^2(TH))\sin^4(TH) \\
& + (18\cos^6(TH) - 24\cos^4(TH))\sin^2(TH) - 6\cos^8(TH) + 6\cos^6(TH))\,UP^2 \\
& + (-24L\cos^2(TH)\sin^4(TH) + (24L\cos^2(TH) - 12L)\cos^4(TH))\sin^2(TH) \\
& + 12L\cos^6(TH) - 12L\cos^4(TH))\,UP + R^2\cos^2(TH)\sin^4(TH) \\
& + (2R^2\cos^4(TH) + (-6L^2 - R^2)\cos^2(TH))\sin^2(TH) + R^2\cos^6(TH) \\
& + (-6L^2 - R^2)\cos^4(TH) + 6L^2\cos^2(TH))\,VP^2 \\
& + ((12\cos^3(TH)\sin^5(TH) - 12\cos^3(TH)\sin^3(TH) \\
& + (12\cos^5(TH) - 12\cos^7(TH))\sin(TH))\,UP^3 \\
& + (-12L\cos(TH)\sin^5(TH) + (12L\cos^3(TH) + 12L\cos(TH))\sin^3(TH) \\
& + (24L\cos^5(TH) - 24L\cos^3(TH))\sin(TH))\,UP^2 \\
& + ((12L^2\cos(TH) - 12L^2\cos^3(TH))\sin(TH) - 12L^2\cos(TH)\sin^3(TH))\,UP \\
& + 2R^2L\cos(TH)\sin^3(TH) + (2R^2L\cos^3(TH) - 2R^2L\cos(TH))\sin(TH))\,VP \\
& + (\sin^8(TH) + 4\cos^2(TH)\sin^6(TH) + (6\cos^4(TH) - 2\cos^6(TH))\sin^2(TH) \\
& + \cos^8(TH))\,UP^4 + (-4L\sin^6(TH) - 12L\cos^2(TH)\sin^2(TH) - 4L\cos^6(TH))\,UP^3 \\
& + (-R^2\sin^6(TH) - 2R^2\cos^2(TH)\sin^4(TH) \\
& + (-R^2\cos^4(TH) + (6L^2 - R^2)\cos^2(TH) + 6L^2)\sin^2(TH) \\
& + (6L^2 - R^2)\cos^4(TH))\,UP^2 + (2R^2L\sin^4(TH) \\
& + (2R^2L\cos^2(TH) - 4L^3)\sin^2(TH) + (2R^2L - 4L^3)\cos^2(TH))\,UP + L^4 \\
& - R^2L^2
\end{aligned}
\tag{36}
$$

Attempts to solve this polynomial still failed due to lack of memory space. Finally, a solution was obtained through an artifice consisting of a rotation of coordinates. New independent velocity variables were defined by

$$u' = u \cos\theta + v \sin\theta \quad , \tag{37}$$

$$v' = u \sin\theta - v \cos\theta \quad . \tag{38}$$

These equations were solved for u and v and the resulting expressions were loaded into Macsyma with instructions to use them to replace u and v in the characteristic polynomial. The resulting characteristic polynomial was rather lengthy but could be solved after a few manipulations. Macsyma was instructed to make the trigonometric substitution $\sin^2 = 1\text{-}\cos^2\theta$. Then the Macsyma built-in trigonometric reduction routine was applied to the characteristic polynomial. The resulting characteristic polynomial was finally displayed by the computer,

$$\lambda^4 - 4u'\lambda^3 + (6u'^2 - a^2)\lambda^2 + (2a^2u' - 4u'^3)\lambda + u'^4 \quad , \tag{39}$$

which is far simpler than the previous equation with 614 terms.

Macsyma was then instructed to solve the polynomial and returned

$$\lambda = u',\ u'+a,\ u'-a \tag{40}$$

as roots (u' is a repeated root), and hence eigenvalues of the matrix C. The following section uses the eigenvalues to determine the stability criteria for the generalized Lax-Wendroff difference operators.

6.0 STABILITY CRITERIA FOR FINITE DIFFERENCE SCHEME

The von Neumann necessary conditions for stability can now be determined using equation (27), the definition of μ, and the eigenvalues just computed.

A rearrangement of equation (27) provides

$$|p| \leq 1 - 4\mu^2 + 4\mu^4 + (2\cos^2\alpha + 2\cos^2\beta)\mu^2$$

$$= 1 - 2\mu^2 (2 + 2u^2 \cos^2\alpha \cos^2\beta)\mu^2$$

$$= 1 - 2\mu^2 (\sin^2\alpha + \sin^2\beta - 2\mu^2) \quad . \tag{41}$$

From the results of computing the eigenvalues of C,

$$\mu^2 = r^2\Delta t^2\lambda^2 = r^2\Delta t^2 \begin{Bmatrix} u' \\ u' \\ u' + a \\ u' - a \end{Bmatrix}^2 , \tag{42}$$

where the eigenvalues have been written as a column vector and

$$r^2 = \frac{\sin^2\alpha}{\Delta x^2} + \frac{\sin^2\beta}{\Delta y^2} \quad . \tag{43}$$

Therefore,

$$\mu^2 = \left[\frac{\sin^2\alpha}{\Delta x^2} + \frac{\sin^2\beta}{\Delta y^2}\right]\Delta t^2 \begin{Bmatrix} u' \\ u' \\ u' + a \\ u' - a \end{Bmatrix}^2 \quad . \tag{44}$$

And the von Neumann necessary condition for stability becomes:

$$|p| \leq 1 - 2\mu^2 \left[\sin^2\alpha + \sin^2\beta - 2\left(\frac{\sin^2\alpha}{\Delta x^2} + \frac{\sin^2\beta}{\Delta y^2}\right)\Delta t^2 \begin{Bmatrix} u' \\ u' \\ u' + a \\ u' - a \end{Bmatrix}^2\right] \tag{45}$$

$$\leq 1 - 2\mu^2 \left[\left[1 - 2\left(\frac{\Delta t}{\Delta x}\right)^2 \begin{Bmatrix} u' \\ u' \\ u' + a \\ u' - a \end{Bmatrix}^2\right]\sin^2\alpha \right.$$

$$\left. + \left[1 - 2\left(\frac{\Delta t}{\Delta y}\right)^2 \begin{Bmatrix} u' \\ u' \\ u' + a \\ u' - a \end{Bmatrix}^2\right] \sin^2\beta\right] \quad . \tag{46}$$

Since the sound speed, a, is always positive for any real fluid, the moduli of the eigenvalues are therefore less than one when both of the following conditions are met:

$$\left(\frac{\Delta t}{\Delta x}\right)(|u'| + a) < \frac{1}{\sqrt{2}} \quad , \left(\frac{\Delta t}{\Delta y}\right)(|u'| + a) < \frac{1}{\sqrt{2}} \quad . \tag{47}$$

Here u′ is the contravariant velocity component in the direction θ as defined in the rotation of coordinates in equation (37). The direction θ depends on the choice of wave numbers through α, β and the definition of θ in equation (23). Since all possible combinations of wave numbers must be considered, the worst case (largest) value of u′ occurs when θ is in local alignment with the fluid velocity vector in the curvilinear coordinate system. The contravariant velocity scales with the physical velocity according to the square root of the Jacobian of the coordinate transformation. In terms of physical components of velocity at a point, the stability criteria are

$$\frac{\Delta t}{\Delta x}\left(\frac{V}{\sqrt{J}} + a\right) < \frac{1}{\sqrt{2}} \quad , \frac{\Delta t}{\Delta y}\left(\frac{V}{\sqrt{J}} + a\right) < \frac{1}{\sqrt{2}} \quad , \tag{48}$$

where V is the physical speed at the point.

These results agree with intuition based on experience in Cartesian coordinates. Unfortunately, from the standpoint of correcting the instability, there are no new types of criteria such as critical directions or angles. Strict compliance with the above criteria did not guarantee stability in practice. The finite difference method remains unstable for arbitrarily small Δt when turning angle exceeds twenty-three degrees. Therefore, the criteria in equation (48) are necessary, but not sufficient, to assure stability of the finite difference method in curvilinear coordinates.

ACKNOWLEDGEMENTS

The research presented above was a portion of the author's Ph.D. dissertation (Yagla, 1981) research performed under the supervision of Dr. D. L. Evans, Professor of Mechanical Engineering, Arizona State University. Most of the material also appears as an appendix to the dissertation. The Macsyma calculations were supported by the Combat Systems Department of NSWC. The calculations would not have been undertaken without the assistance of Mr. Kenton Meals of the David Taylor Naval Ship Research and Development Center (DTNSRDC). Dr. Elizabeth Cuthill, also of DTNSRDC, provided helpful discussions and a high speed terminal for some of the calculations.

REFERENCES

Courant, R. J., Differential and Integral Calculus, translated by E. J. McShane, Vol. 2 New York: Interscience Publishers, 1947.

MacCormack, R. W., "The Effect of Viscosity in Hypervelocity Impact Cratering," AIAA Paper No. 69-354, 1969.

Richtmyer, R. D., Difference Methods for Initial Value Problems. New York: Interscience Publishers, 1957.

Richtmyer, R. D. and K. W. Morton, Difference Methods for Initial Value Problems, Second Edition. New York: Interscience Publishers, 1967.

Viviand, H., "Formes Conservatives des Equations de la Dynamique des Gas." La Recherche Aerospatiale, Annee 1974, N. 1, p 65.

Warming, R. F. and Richard M. Beam, "On the Construction and Application of Implicit Factored Schemes for Conservation Laws, " SIAM-AMS Proceedings Volume II, 1978, p-85.

Yagla, J. J., "Conservation Law Form of the Equations of Gas Dynamics in Coordinate Systems Obtained by Conformal Mapping," NWL Technical Report TR-2724, Naval Weapons Laboratory, Dahlgren, Virginia, July 1972.

Yagla, J. J., "A Finite Difference Method for Solving the Equations of Gas Dynamics in Curvilinear Coordinate Systems Obtained by Conformal Mapping," Ph.D. Dissertation, Arizona State University, Tempe, Arizona, 1981.

TWO- TO THREE DIMENSIONAL MAPPING

Paul D. Engelman

During the feasibility and comparison studies for the "B1-era" bomber, a graphic display of the coverage area for different aircraft was required. Because of the aircraft's range and tactics, it was launched from, and recovered at different airfields. The trace of all the points reached by an aircraft flying its maximum range is an ellipse. The launch and recovery airfields correspond to an ellipse's two foci. The aircraft's maximum range is equivalent to the sum of the two line segments from the foci to a point on the periphery. The very large ranges being considered made the direct plotting of an ellipse on a flat-plane map projection (i.e., Mercator, orthographic) unacceptable, so the ellipse or "footprint" had to be projected on a globe and then plotted along with other earth features in some standard map projection.

The initial approach to the problem was to distort a two-dimensional ellipse to match the distortions of the map projection being displayed. In addition to being intuitively unsatisfying, this approach was fraught with problems, most notably those that arise with the discontinuities at the poles and at the 180-degree meridian (the International Date Line). Rather than derive a set of equations that was dependent on a particular map projection, I derived a set of equations that mapped the ellipse over the three-dimensional surface of the earth.

The final approach was to consider each point on the "footprint's" outline individually, rather than treating it as a geometric figure. The problem could then be stated: Plot the points (Target) reached by an aircraft launching from an airfield (A1) and recovering at a different airfield (A2). The aircraft has a maximum range K, and will fly great circle routes. The problem was then solved in two independent steps: the first to map the ellipse onto the globe, its points being computed as x, y, and z, and the second to use the existing cartographic system to plot these points. Only the first step will be discussed here.

Consider a plane (P) defined by three points (A1, A2, T). P intersects a sphere such that A1, A2, and T lie on the sphere's surface. Assume the following notation:

P(to)/(from): the distance A1 to T/A2 from T on P
S(to)/(from): the distance A1 to T/A2 from T along the sphere's surface

Drawing vectors from the origin to A1, A2, and T (V1, V2, VT), then subtracting to find the vectors on P gives:

$$VT - V2 = P(\text{from})$$
$$VT - V1 = P(\text{to})$$

and the magnitudes of these vectors:

1) $\text{SQRT}((VTX - V2X)^2 + (VTY - V2Y)^2 + (VTZ - V2Z)^2) = |P(\text{from})|$
2) $\text{SQRT}((VTX - V1X)^2 + (VTY - V1Y)^2 + (VTZ - V1Z)^2) = |P(\text{to})|$

Expanding (1) and (2), combining terms and substituting

3) $|V1| = |V2| = |VT| = r$

gives

4) $2(r^2 - VTX\ V1X - VTY\ V1Y - VTZ\ V1Z) = |P(to)|^2$

5) $2(r^2 - VTX\ V2X - VTY\ V2Y - VTZ\ V2Z) = |P(from)|^2$

To express $|P(to)|$ and $|P(from)|$ in terms of their lengths along the surface

$$\text{chord} = 2r\sin(\tfrac{1}{2}\theta)$$
$$\theta = \frac{\text{arc length}}{r}$$

and

6) $$|P(to)| = 2r \sin\left(\frac{S(to)}{2r}\right)$$

7) $$|P(from)| = 2r \sin\left(\frac{S(from)}{2r}\right)$$

Relating (4) and (5) to (6) and (7) gives

8) $$VTX\ V2X + VTY\ V2Y + VTZ\ V2Z = r^2\left(1 - \sin^2\frac{S(from)}{2r}\right) = a$$

9) $$VTX\ V1X + VTY\ V1Y + VTZ\ V1Z = r^2\left(1 - \sin^2\frac{S(to)}{2r}\right) = b$$

Equations (8), (9), and the radius vector (3) to T are input to MACSYMA. The resultant subsystem contains equations (3, 8, 9, 10, 11, 12) with the solution set [[3, 8, 9], [10, 11], [12]]. The constant parameters in the system are: r, a, b, V1X, V1Y, V1Z, V2X, V2Y, V2Z, and the unknowns are: VTX, VTY, and VTZ.

(10) $$-V1X\ a + (V1X\ V2Z - V1Z\ V2X)\ VTZ + (V1X\ V2Y - V1Y\ V2X)\ VTY + b\ V2X = 0$$

(11) $$a^2 + (-2\ V2Z\ VTZ - 2\ V2Y\ VTY)\ a + (V2Z^2 + V2X^2)\ VTZ^2 + 2\ V2Y\ V2Z\ VTY\ VTZ + (V2Y^2 + V2X^2)\ VTY^2 - r\ V2X^2 = 0$$

(12) $$(V1Y^2 + V1X^2)\ V2X\ a^2 + (((-2\ V1Y^2 - 2\ V1X^2)\ V2X\ V2Z + 2\ V1Y\ V1Z\ V2X\ V2Y + 2\ V1X\ V1Z\ V2X^2)\ VTZ - 2\ V1Y\ b\ V2X\ V2Y - 2\ V1X\ b\ V2X^2)\ a + ((V1Y^2 + V1X^2)\ V2X\ V2Z^2 + (-2\ V1Y\ V1Z\ V2X\ V2Y - 2\ V1X\ V1Z\ V2X^2)\ V2Z + (V1Z^2 + V1X^2)\ V2X\ V2Y^2 - 2\ V1X\ V1Y\ V2X^2\ V2Y + (V1Z^2 + V1Y^2)\ V2X^3)\ VTZ^2 + ((2\ V1Y\ b\ V2X\ V2Y + 2\ V1X\ b\ V2X^2)\ V2Z - 2\ V1Z\ b\ V2X\ V2Y^2 - 2\ V1Z\ b\ V2X^3)\ VTZ + (b^2 - r\ V1X^2)\ V2X\ V2Y^2 + 2\ r\ V1X\ V1Y\ V2X^2\ V2Y + (b^2 - r\ V1Y^2)\ V2X^3 = 0$$

A computer solution is generated by starting at the limiting case, when T is on the line connecting A1 and A2. This allows S(to) to be computed in terms of the constant K, and the distance between the two airfields. See the figure below.

13) S(to) + S(from) = K
14) S(from) - (dist A1 to A2) = S(to)

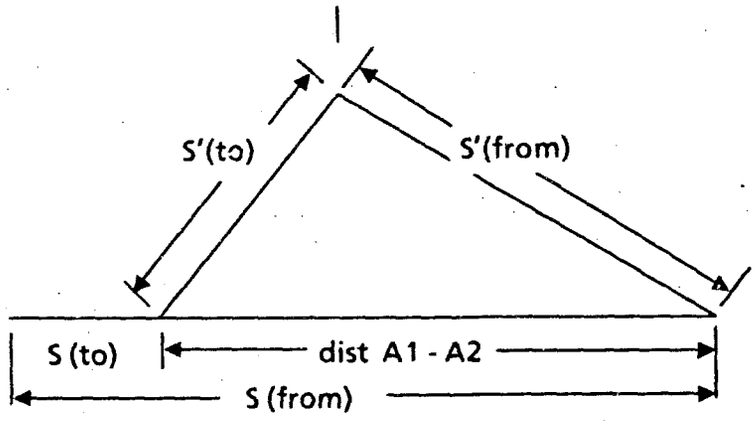
From (13);

15) S(from) = K - S(to)

Substituting (15) into (14) gives

16) $S(to) = \frac{K - (\text{dist A1 to A2})}{2}$

and (15) gives S(from).



From this starting point, a new T is computed by incrementing S(to) by some arbitrary amount, and then recomputing S(from) using this new value of S(to) and K. See the figure above. This is continued until S(from) is equal to the original value of S(to). Each set of values for S(to) and S(from) are used to determine the parameters a and b from which VTX, VTY, and VTZ are calculated. The solution for the Z component is generated by applying the quadratic formula to equation 12, with the positive and negative solutions corresponding to the "upper" and "lower" halves of the ellipse. Equation 10 yields the positive and negative values of VTY using the positive and negative values of VTZ, and equation 8 yields VTX in a similar manner.

Figure 1 shows identical ellipses mapped onto a globe, and then projected in Mercator and orthographic projections. At this point, this algorithm is limited to large-distance global applications, possibly search patterns for air-sea rescue. However, a change in the way S(to) and S(from) are generated allows other useful mappings.

If, instead of using some relationship between S(to) and S(from) (i.e., that their sums or differences are constant), the distance from two re-

ference points to an arbitrary third point is measured, it is possible to map this point on a sphere while maintaining its proper relation with the two reference points. Figure 2 shows the author's name "Paul" mapped onto the Atlantic Ocean in both Mercator and orthographic projections. Consider a satellite photograph; if two points in a photograph are known accurately, the two-dimensional photograph can be digitized and then "wrapped" around a sphere by measuring the distance from the two reference points to each digitized point and using these distances as S(to) and S(from). This will give more accurate mappings then matching known earth features with the photograph, plotting them and "filling-in" the rest, and would be more valuable with satellite photographs where the surface is unknown. The problems associated with overlaying two different photographs may also be reduced.

As a final extension of these equations, we can work backwards by solving for S(to) and S(from), and map points from a sphere to a plane. This may have application in the area of manufacturing and graphics art; a design could be done on the desired spherical surface, transfered to a plane where it could be easily reproduced, and then remapped back through some type of "shrinking" process onto the spherical surface.

Implicit in the above derivation is the use of a perfect sphere, and this is not the case with the earth. To "work around" this assumption, equations 10, 11, and 12 could be solved for an initial approximation of a position. Then, in an iterative fashion, you may use this position in the earth model for a more accurate estimation of the radius. This radius is then used to recalculate S(to) and S(from), and these values are used in equations 10, 11, and 12 to obtain a more accurate position. This was not required for our system, but it is a reasonable approach to the problem of a non-spherical body because we are only plotting single points, maintaining their spacial relationship with the two reference points, rather then with other points along the outline of the figure. As I mentioned above, the original request was for the mapping of the ellipse for a specific study. The equations finally derived are (I think) unsolvable using pencil and paper. Unfortunately, some of the applications have not been implemented but explored only in a test program, or as paper and pencil exercises.

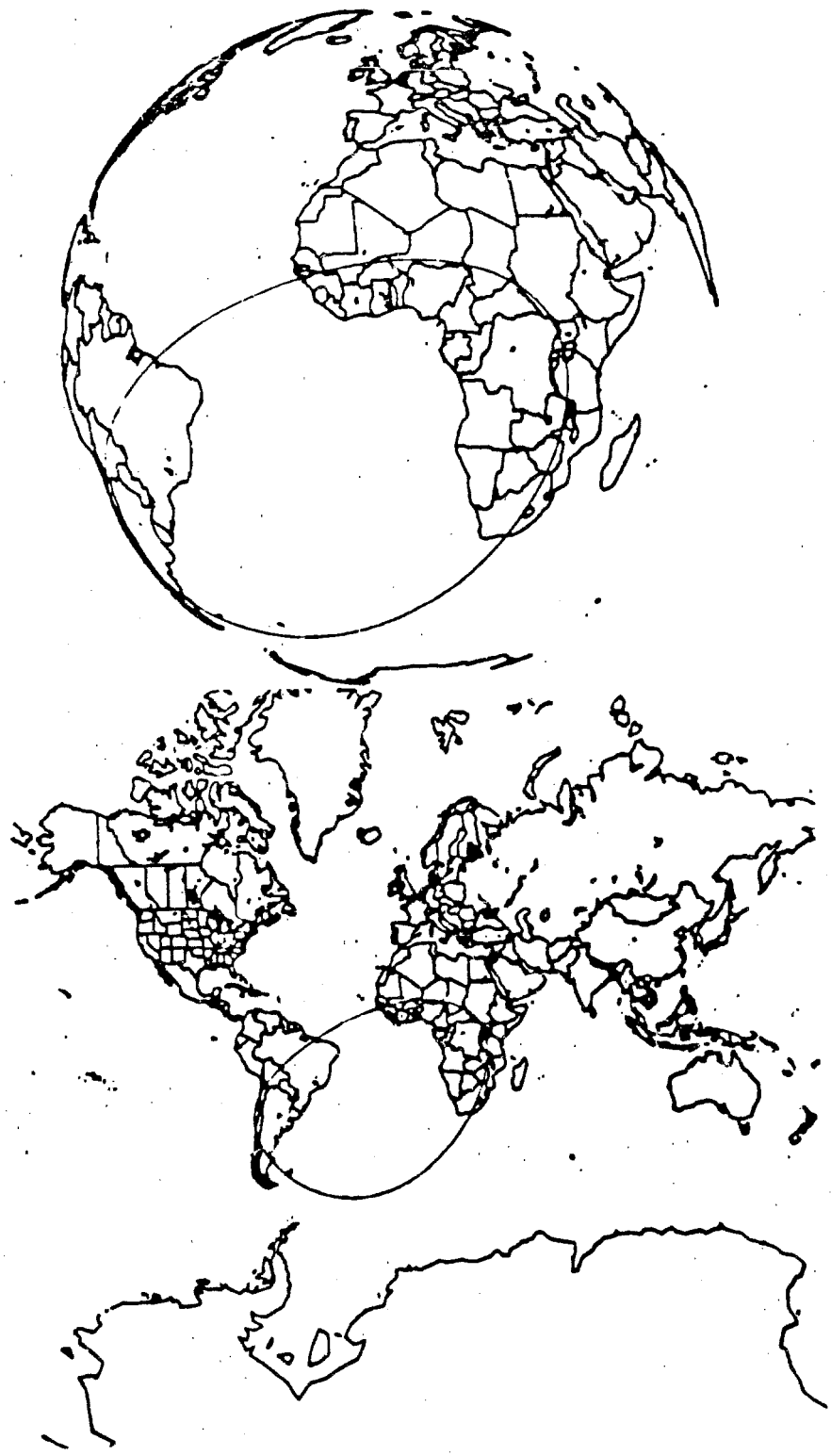


Fig. 1

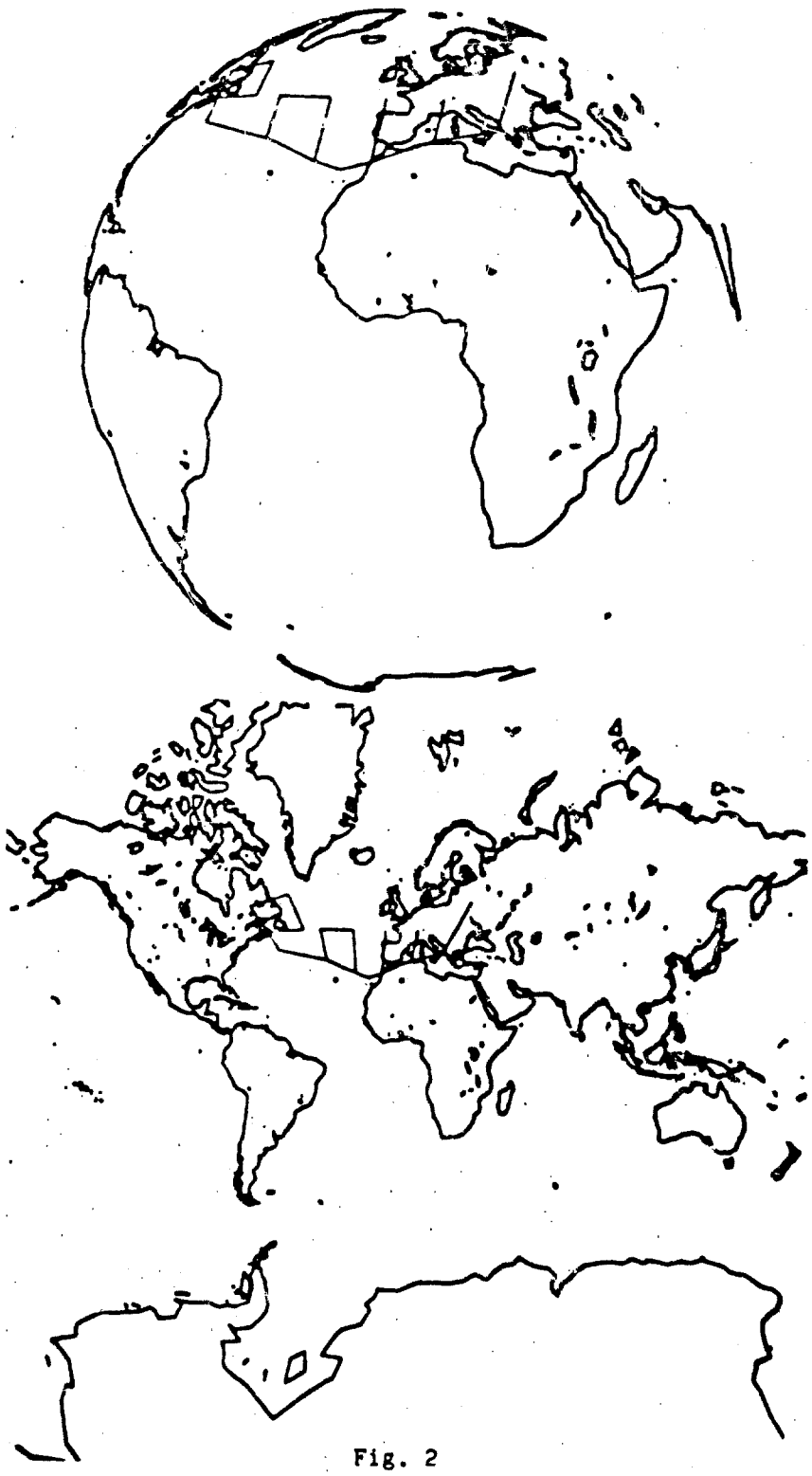


Fig. 2

A LISP-BASED RATFOR CODE GENERATOR

Barbara L. Gates* and Paul S. Wang†

Department of Mathematical Sciences
Kent State University
Kent, Ohio 44242

Abstract

We describe an automatic code generator which produces RATFOR or C code from symbolic VAXIMA expressions. The code generator is written in FRANZ LISP and is implemented as a set of functions callable from VAXIMA. Features of this code generator not previously available include generation of declarations, I/O statements, control flow structures, functions, and subroutines; segmentation of large expressions; interleaving of symbolic computations and code generation calls; and redirection of output code to multiple files. Code derivation routines have control over how, where, and when code is generated. Development of the code generator is motivated by the need for automatic derivation and generation of FORTRAN code for finite element analysis.

1. INTRODUCTION

The importance of automatic code generation in the interfacing of symbolic manipulation systems with numerical computing techniques has been widely recognized. For many scientific computations, formulas and expressions are derived on a symbolic system before they are used in repeated numerical calculations. Therefore, a code generator which combines the symbolic and numerical approach by producing executable RATFOR or C code can be most effective in reducing the effort typically required for a given problem. Our research is motivated in particular by the need for efficient generation of code for finite element analysis [12]. To fill this need, we have built a general purpose code generator which meets many requirements for automatic code generation under program

* Current address: Picker International, Inc., 595 Miner Road, Highland Heights, Ohio 44143

† Work reported herein has been supported in part by the US National Aeronautics and Space Administration under Grant NAG 3-298 and in part by the Department of Energy under Grant DE-AC02-ER76020?5-A013

control at the LISP level. This code generator runs under VAXIMA and is accessible from the user level as well as the LISP level.

Previous systems which have attempted to address the need for automatic code generation lack one or more vital functionalities which prohibit their general use. The MACSYMA system [6] provides a simple command which generates FORTRAN assignment statements from expressions and matrices. A similar facility is provided in the REDUCE system [7]. The MACTRAN package [13], created in 1980 by M. Wirth for MACSYMA, converts MACSYMA equations and other expressions into FORTRAN code, and provides a text processor which allows the derived FORTRAN code segments to be interspersed with fixed code from program skeletons. Thus, to form complete FORTRAN programs, the MACTRAN package requires the user to supply *template files* containing FORTRAN code and commands for converting computational results into FORTRAN assignment statements.

The VAXTRAN package [4,5] was developed by D. Lanam in 1981 to run under VAXIMA [3]. VAXTRAN is similar to MACTRAN in that it generates assignment statements, but VAXTRAN also makes it possible to compile, load, and interface the generated code directly with VAXIMA, thereby making it possible to combine symbolic and numerical computations on the same system.

MACTRAN and VAXTRAN represent a first step towards providing an interface between symbolic and numerical computing techniques. However, they do not supply a convenient way to generate statements such as declarations, control-flow structures, I/O statements, functions and subroutines. These statements are necessary for generating a complete and efficient FORTRAN program. We describe a code generator, written in FRANZ LISP for VAXIMA, which generates these constructs from an expression or statement in VAXIMA internal representation. A substantial subset of all VAXIMA statements can be handled. This generator, GENTRAN (symbolic to numerical code GENerator/TRANslator), has the ability to generate complete RATFOR or C programs or subprograms with or without a template file. When RATFOR code is generated, an existing preprocessor then takes the RATFOR programs and produces FORTRAN 77 code. The generated code can optionally be optimized, and an automatic testing facility is being developed to verify correctness of the generated code.

2. SPECIFICATIONS OF THE CODE GENERATOR

The GENTRAN package has been developed to generate RATFOR code based on symbolic mathematical expressions derived in VAXIMA. An overview of the code generation process is shown in Figure 1. All code generation functions can be accessed from VAXIMA's top level as well as from the LISP level. Thus, GENTRAN code generation functions can easily be called from LISP programs.

RATFOR code can be generated from literal or value translations. Literal (direct) translations are a convenient way to generate RATFOR assignment statements, control structures, I/O statements, and statement groups. Value translations make it possible to generate RATFOR code from the result of a VAXIMA expression evaluation.

The GENTRAN package also provides special code generation functions which make it possible to generate RATFOR statements such as declarations which have no direct equivalent representation in VAXIMA. Other special code generation functions provide convenient ways to piece together function and subroutine definitions in separate function calls. If desired, pieces of several subprograms can be saved in memory at the same time and RATFOR code can be generated from those saved definitions at a later time.

It is also possible to direct the generated code to multiple output files. Code generation for the main program and for function and subroutine definitions may be intermixed if output is directed to different files. GENTRAN provides a flexible LISP interface for sending output to designated files, for separately generating parts of a program such as function and subroutine declarations, type declarations, and individual statements. Thus, a code derivation routine can have maximal control over how, where, and when code is generated.

A template-processing routine is provided for applications in which it is convenient to generate RATFOR code by following a user-supplied *template file*. A template file contains a skeleton RATFOR program consisting of RATFOR statements and *active parts*. Each active part contains VAXIMA statements and code generation function calls enclosed in the characters "<<" and ">>". The template-processing routine reads the given template file and passes all characters to the output file without modification, except for active parts. VAXIMA commands in the active parts are executed in the order given, and any resultant output code is sent to the output file as well. Thus, the output file consists of the template file with all active parts replaced by generated code. It is

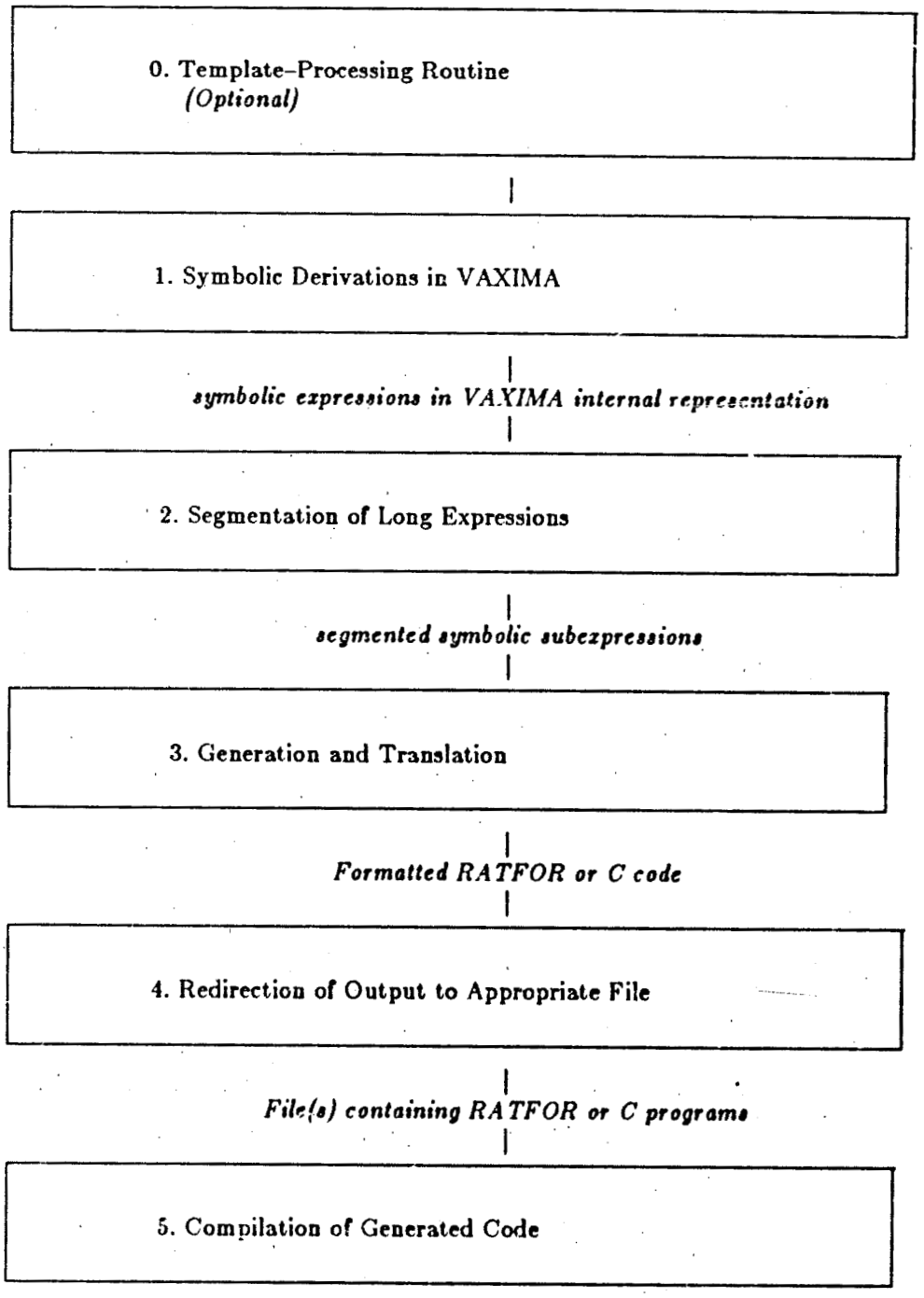


Figure 1: Overview of the code generation process.

therefore a syntactically correct RATFOR file ready to be compiled.

3. APPLICATION IN GENERATING FINITE ELEMENT CODE

An example of the application of GENTRAN to a finite element problem is shown in Figure 2. The code is for a four-node plane element in the isoparametric formulation and was produced by interleaving symbolic derivation computations and code generation calls to GENTRAN in LISP programs. The quantities m1 through m6 are entries of the material matrix which are declared real in an earlier part of the code. We show two stiffness coefficients, sk(1,1) and sk(1,2). The functions g11, p1p1, etc. are all formed by the symbolic derivation phase and are generated by calling appropriate functions of GENTRAN.

4. IMPLEMENTATION

The GENTRAN code generator is divided into five main modules: the parser extension, the code generator/translator, the formatter, the code optimization module, and the automatic testing facility.

The parser extension, which subdivides a given VAXIMA statement or expression to determine whether or not it can be translated literally into RATFOR, is a particularly useful GENTRAN feature. Since VAXIMA is a LISP-based system, the VAXIMA programming language is much more flexible than numerical programming languages and allows variables to be assigned symbolic or numerical values.

VAXIMA also provides many high-level operations for symbolic computations such as differentiation, integration, matrix multiplication, and matrix inversion. Therefore only a subset of all VAXIMA statements can be translated literally into RATFOR code. For this reason, error messages must be generated for VAXIMA statements and expressions which have no equivalent representation in RATFOR.

Similarly, in the GENTRAN parser extension, a context-free grammar is used to define the subset of all VAXIMA statements and expressions that can be translated literally into RATFOR. The implementation is based on a notation called a context-free grammar, or BNF (Backus-Naur Form) description used in compiler theory [1] to define the syntax of an entire programming language. Figure 3 contains the portion of the grammar that defines all valid GENTRAN assignment statements. The GENTRAN parser

```
.
.
.
t0 = g11(y,y)
t1 = -g11(x,y)
t2 = g11(x,x)
sk(1,1) = (m6*t2+2.0*m3*t1+m1*t0)/detk
sk(1,2) = (m5*t2+m6*t1+m2*t1+m3*t0)/detk
.
.
.

function p1p1(aa,bb)
implicit real*8 (a-h,o-z)
dimension aa(4),bb(4)
v0 = vi(12,aa); v1 = vi(12,bb); v2 = vi(10,bb); v3 = vi(10,aa)
return(16.0/3.0*v0*v1+16.0/9.0*v2*v3)
end

function q1p1(aa,bb)
implicit real*8(a-h,o-z)
dimension aa(4),bb(4)
v0 = vi(9,aa); v1 = vi(12,bb); v2 = vi(10,aa); v3 = vi(10,bb)
return(-4.0*v0*v1-4.0/3.0*v1*v2-4.0/3.0*v0*v3-4.0/9.0*v2*v3)
end

function q1q1(aa,bb)
implicit real*8 (a-h,o-z)
dimension aa(4),bb(4)
v0 = vi(9,aa); v1 = vi(9,bb); v2 = vi(10,aa); v3 = vi(10,bb)
return(16.0/3.0*v0*v1+16.0/9.0*v2*v3)
end

function g11(aa,bb)
dimension aa(4),bb(4)
return (p1p1(aa,bb)+q1p1(aa,bb)+q1q1(aa,bb)+q1p1(bb,aa))
end
```

Figure 2: Example of automatically generated finite element code.

is implemented as a recursive-descent algorithm.

The code generation/translation module consists of a set of LISP functions which translate internal LISP prefix data structures into RATFOR statements and expressions in infix form. When the translation module is called on more than one statement, it

```
assignment  ::= ((msetq) id exp) | ((msetq) id assignment)
            |   ((msetq) id matrix)
exp         ::= ((mminus) exp) | ((mquotient) exp exp) | ((rat) exp exp)
            |   ((mexpt) exp exp) | ((mplus) seqexp) | ((mtimes) seqexp)
            |   id | number
seqexp      ::= exp seqexp'
seqexp'     ::= exp seqexp' | ϵ
matrix      ::= (($matrix) seqlist)
seqlist     ::= ((mlist) seqexp) seqlist'
seqlist'    ::= ((mlist) seqexp) seqlist' | ϵ
```

Figure 3. Grammar for all acceptable assignment statements.

automatically groups the statements into three categories: nonexecutable statements, executable statements, and subprogram definitions. Subprogram definitions are then recursively subdivided until only executable and nonexecutable statements remain. Then all nonexecutable statements are generated before executable statements.

The GENTRAN formatter takes RATFOR code and generates it in a user-specified format. The user can control the number of statements generated on a line, maximum line length, indentation length, and placement of braces for statement grouping. Thus, the user has the option of tailoring the formatting process to match any desired coding style.

The code optimization module is currently based on an existing code optimizer in REDUCE [9,10]. It identifies common subexpressions to minimize the number of arithmetic operations. Experiments with the REDUCE code optimizer have shown that a systematic common subexpression search can help reduce code size and increase code efficiency [12].

An automatic testing facility is currently being developed. It will provide a convenient way for the user to verify the correctness of the generated code.

5. PRACTICAL CONSIDERATIONS

A table of variable attributes is created for each complete subprogram that is generated. Entries are placed in this table both explicitly — through user-supplied calls to type declaration functions — and implicitly — as the result of scanning all executable statements comprising the body of the subprogram. Declaration statements can be automatically generated from this table, and variable attributes can easily be cross-checked.

Expressions derived symbolically are sometimes much too large to deal with directly. They may be many lines or many pages long. These expressions cannot be translated directly into RATFOR code since there is a limit on the number of lines for any one RATFOR statement. Therefore, the code generator automatically segments, or breaks unreasonably large expressions down recursively, until several expressions of manageable size are found. Assignment statements are then generated for those smaller expressions. Figure 4a shows a large symbolic expressions that was derived in VAXIMA. The (unoptimized) result of segmenting this statement into smaller statements of manageable size is shown in Figure 4b. The temporary variables t0, t1,... are automatically generated to hold subexpression values. They are re-used as soon as possible so that only a minimal number of temporary variables is needed.

6. CONCLUSION

The GENTRAN package has been developed to satisfy the needs of producing FORTRAN subroutines based on symbolic computations for finite element analysis. However, it also functions as a general-purpose code generator. It contains many features for practical code generation not found in previous code generation facilities.

In addition to generating RATFOR code from LISP data structures, GENTRAN is capable of generating C code. RATFOR code is easier to read and much more compact than FORTRAN code. However, one disadvantage is the dependence on the RATFOR preprocessor which may not be available on every system where VAXIMA or LISP runs.

To provide flexibility, code generation can be guided by user-supplied template files. However, template files are not required for program generation.

```
p1p1 : 128.0/27.0*v0*v1*v12*v14+64.0/27.0*v0*v1*v13*v14
  +64.0/27.0*v0*v1*v12*v15+256.0/135.0*v0*v1*v13*v15
  +128.0/27.0*v0*v10*v14*v2+64.0/27.0*v0*v11*v14*v2
  +64.0/27.0*v0*v10*v15*v2+32.0/27.0*v0*v11*v15*v2
  +128.0/27.0*v0*v10*v12*v3+64.0/27.0*v0*v11*v12*v3
  +64.0/27.0*v0*v10*v13*v3+32.0/27.0*v0*v11*v13*v3
  +128.0/45.0*v0*v1*v2*v3+64.0/27.0*v0*v10*v12*v4
  +32.0/27.0*v0*v11*v12*v4+256.0/135.0*v0*v10*v13*v4
  +128.0/135.0*v0*v11*v13*v4+64.0/45.0*v0*v1*v2*v4
  +64.0/27.0*v0*v10*v14*v5+32.0/27.0*v0*v11*v14*v5
  +256.0/135.0*v0*v10*v15*v5

  .
  .
  .

  +512.0/675.0*v13*v4*v7*v9+128.0/135.0*v14*v5*v7*v9
  +512.0/675.0*v15*v5*v7*v9
```

Figure 4a: Long symbolic expression derived in VAXIMA.

Future work in this area includes a better interface to the code optimizer and developing techniques to help test the generated programs.

REFERENCES

1. Aho, A. V., and J. D. Ullman. *Principles of Compiler Design.* Addison-Wesley Publishing Company, Reading, Massachusetts, 1977.
2. Feldman, S. I., and J. Ho. A rational expression evaluation package. Computing Science Technical Report #34, Bell Laboratories, Murray Hill, New Jersey, 1975.
3. Foderaro, J. K., and Fateman, R. J. Characterization of VAX macsyma. *ACM SYMSAC '81 Conference Proceedings*, Snowbird, Utah, USA, Aug. 5-8, 1981.
4. Lanam, D. H. A package for generating and executing fortran programs with macsyma. Master's thesis, University of California, Berkeley, 1982.
5. Lanam, D. H. An algebraic front-end for the production and use of numeric programs. *ACM SYMSAC '81 Conference Proceedings*, Snowbird, Utah, USA, Aug. 5-8, 1981.

```
t0=128.0/27.0*v0*v1*v12*v14
t1=64.0/27.0*v0*v1*v13*v14
t2=64.0/27.0*v0*v1*v12*v15
t3=256.0/135.0*v0*v1*v13*v15
t4=128.0/27.0*v0*v10*v14*v2
t5=64.0/27.0*v0*v11*v14*v2
t6=64.0/27.0*v0*v10*v15*v2
t7=32.0/27.0*v0*v11*v15*v2
t8=128.0/27.0*v0*v10*v12*v3
t9=64.0/27.0*v0*v11*v12*v3
t10=64.0/27.0*v0*v10*v13*v3
t0=t0+t1+t2+t3+t4+t5+t6+t7+t8+t9+t10
t1=32.0/27.0*v0*v11*v13*v3
t2=128.0/45.0*v0*v1*v2*v3
t3=64.0/27.0*v0*v10*v12*v4
t4=32.0/27.0*v0*v11*v12*v4
t5=256.0/135.0*v0*v10*v13*v4
t6=128.0/135.0*v0*v11*v13*v4
t7=64.0/45.0*v0*v1*v2*v4
t8=64.0/27.0*v0*v10*v14*v5
t9=32.0/27.0*v0*v11*v14*v5
t10=256.0/135.0*v0*v10*v15*v5
t0=t0+t1+t2+t3+t4+t5+t6+t7+t8+t9+t10
   .
   .
   .
t5=512.0/675.0*v13*v4*v7*v9
t6=128.0/135.0*v14*v5*v7*v9
t7=512.0/675.0*v15*v5*v7*v9
p1p1=t0+t1+t2+t3+t4+t5+t6+t7
```

Figure 4b: Long expression after segmentation.

6. *MACSYMA Reference Manual*, Version Nine. The MATHLAB Group, Laboratory for Computer Science, M.I.T., Cambridge, Massachusetts, 1977.

7. *REDUCE User's Manual*, Version 3.0. A. C. Hearn (ed.) The Rand Corporation, Santa Monica, California, 1983.

8. *UNIX Programmer's Manual*, Vol. I and II, Seventh Edition. Bell Telephone Laboratories, Inc. Murray Hill, New Jersey, 1979.

9. van Hulzen, J. A. Code optimization by symbolic processing. *NGI-SION Symposium Proceedings*, Amsterdam, April 16–17, 1984. (To Appear).

10. van Hulzen, J. A. Code optimization of multivariate polynomial schemes: A pragmatic approach. In J. A. van Hulzen, (ed.), *EUROCAL '83 Proceedings*, Springer–Verlag LNCS Series 162, 1983.

11. Wang, P. S. Automatic finite element generators. Nonlinear Structural Analysis Workshop, NASA Lewis Research Center, Cleveland, Ohio. April 1983.

12. Wang, P. S., P. Chang, and J. A. van Hulzen. Code generation and optimization for finite element analysis. In J. P. Fitch, (ed.), *EUROSAM '84 Conference Proceedings*, Springer–Verlag, LNCS Series, July 9–11, 1984. (To appear.)

13. Wirth, M. C. On the automation of computational physics. Ph.D. thesis, University of California, Davis School of Applied Science, Lawrence Livermore Laboratory, 1980.

A SURVEY OF SYMBOLIC DIFFERENTIATION IMPLEMENTATIONS†

Michael Wester and Stanly Steinberg
Department of Mathematics and Statistics
University of New Mexico
Albuquerque, New Mexico 87131

Abstract

In this paper, we survey the implementation of differentiation in three major symbolic mathematics packages (MACSYMA, MAPLE and SMP). We use notation, correctness, flexibility and speed as our major criteria for evaluating these differentiators. In general, we found that each mathematics package had various failings and none of the differentiators would do everything that we thought it should be able to do. Based on our observations and experience, we propose some standard notations to be used for symbolic derivatives which will resolve many of the difficulties that we have encountered in current symbolic mathematics packages.

1. INTRODUCTION

It is important that a symbolic mathematics package should allow differentiation of arbitrary functions of arbitrary arguments. The differentiation process should be implemented in a "natural" manner as viewed by the user. This means that both input to the package and output from the package should be easy to understand and consistent with formal mathematical notations. This also means that normal properties of the derivative operation should be able to be applied either automatically or under user control in fairly simple ways. Finally and most importantly, it is vital that all products of derivative operations should be correct. At the very least, an incorrect result should never be produced.

One of the major faults of many symbolic mathematics programs is the lack of a clear cut distinction between the representations of total and partial derivatives. This lack of identity can lead to major difficulties implementing some methods for solving

† This work was partially supported by the U.S. Army Research Office, by the U.S. Air Force Office of Scientific Research, by the National Science Foundation Grant #MCS-8102683, and System Development Foundation.

partial differential equations. Another common problem is that many packages do not readily perform simplifications which use the property of the equivalence of various forms of mixed partial derivatives (e.g. $f_{xy}=f_{yx}$).

In this paper, we will survey the implementation of differentiation in various symbolic mathematics packages (MACSYMA [1], MAPLE [2] and SMP [3]). We will look at notation, correctness, flexibility and speed. We will show several examples from each of the packages, indicating some of the difficulties that we had and in some cases how we attempted to overcome some of the deficiencies that were present. Lastly, we will suggest some standard notations for symbolic derivatives that will resolve many of the difficulties that we have encountered in current symbolic mathematics packages.

2. DIFFERENTIATION IN MACSYMA

In MACSYMA, the DIFF function is used to generate derivatives. For example,

```
DIFF(f(x,y),x,1,y,2);
```

produces

$$\frac{d^3}{dx\,dy^2}(f(x,y)).$$

The arguments of f are listed explicitly; otherwise, DIFF will assume f is a simple variable. Explicit functional dependencies can be omitted if they are defined implicitly through the use of the DEPENDS function. If

```
DEPENDS(f,[x,y]);
```

has been executed, then DIFF will know that f is a function of x and y, so that

```
DIFF(f,x,1,y,2);
```

will produce

$$\frac{d^3 f}{dx\,dy^2}.$$

Note that f must now be specified as an atomic symbol for DIFF to recognize the dependencies which have been defined with the DEPENDS function. For both cases, DIFF will adopt a canonical ordering of the independent variables in the denominator

of the derivative. This allows simplification of mixed partial derivatives to occur automatically. Thus,

```
DIFF(f(x,y),x,1,y,1) - DIFF(f(x,y),y,1,x,1);
```

and

```
DIFF(f,x,1,y,1) - DIFF(f,y,1,x,1);
```

will simplify to zero immediately.

The "total differential" of a function can be obtained by omitting the second argument to DIFF. Thus,

```
DIFF(f(x,y));
```

produces

$$\left(\frac{d}{dy}\,(f(x,\,y))\right)\,del(y)\;+\;\left(\frac{d}{dx}\,(f(x,\,y))\right)\,del(x),$$

where del(x) is the differential of x.

MACSYMA does quite well with the easy cases. Suppose now that y is a function of x. Attempting

```
DIFF(f(x,y(x)),x);
```

yields

$$\frac{d}{dx}\,(f(x,\,y(x))),$$

the so called noun form of the derivative. In other words, MACSYMA was unable to perform the application of the chain rule for this example. In general, if MACSYMA "knows" about a function, then it will be able to produce a complete differentiation of that function when it is written with explicit arguments. For example, MACSYMA has no trouble taking the derivative of $\sin(x^2)$ with respect to x, and many much more complicated problems. However, if the function is "unknown" to MACSYMA, then all attempts to differentiate it will return the unsimplified noun form (with certain exceptions: MACSYMA will realize that f(x,y(x)) depends on x and y(x) [it will also, however, think that f depends on the simple variables y and f] so that differentiating with respect to anything else will yield zero).

Since MACSYMA will not apply the chain rule when differentiating an "unknown" function f with explicit arguments, the next logical alternative is to try using the DEPENDS function. Indeed, after performing

DEPENDS(f,[x,y],y,x);,

indicating that f depends on x and y and y depends on x, then

DIFF(f,x);

will result in

$$\frac{df}{dy}\frac{dy}{dx} + \frac{df}{dx}.$$

This looks good, but suppose the EV function is applied to this expression with the DIFF flag present (as one might do for a more complicated case in which there is a need to perform repeated substitutions and reevaluations [4]). The effect of this operation is to "cause all differentiations indicated [in the expression] to be performed" [1]. For the above expression,

EV(expression,DIFF);

produces

$$2\,\frac{df}{dy}\frac{dy}{dx} + \frac{df}{dx}\;!$$

This finding is not surprising if one realizes that MACSYMA does not distinguish between the total derivative df/dx and the partial derivative $\partial f/\partial x$. Reevaluating the df/dx term produces an expansion of it as a total derivative, contrary to its previous meaning as a partial derivative. Consequently, repeated evaluations of this expression will increment the integer coefficient.

One could avoid applying EV with the DIFF flag to derivative forms, although there are situations in which it would be useful to be able to do exactly this kind of reevaluation as was indicated above. The DEPENDS method, however, possesses certain fundamental difficulties. Suppose we wished to differentiate $g(x^2)$ with respect to x. Surprisingly, DEPENDS(g,x^2)$ does work.

DIFF(g,x);

then results in

$$2\,\frac{dg}{dx^2}\,x,$$

which is correct, although the notation here for the derivative of g with respect to its

argument is pretty bad (a pair of parentheses around the x^2 would be a major improvement, although, for more complicated arguments, this notation could become rather unwieldy). We note that DIFF will not allow the user to differentiate with respect to x^2 even though g depends on this quantity (examples of differentiating functions in this manner occur in the physics literature). Now, consider f(t,t), derived by letting x = x(t) = t and y(x) = x = t. There is no way to obtain the correct differentiation of this object (which is needed, for example, in the inverse scattering problem) using the DEPENDS concept. We consider DEPENDS to be useful, but the concept is too weak to cover the broad range of applications that a symbolic differentiator should be able to handle. DEPENDS does best for functions of simple, nonrepeated arguments.

Two other important functions are available in MACSYMA for dealing with differentiation properties. These are the ATVALUE function, which can be used to define the derivative of a function at a point (i.e. set up a boundary condition) and the GRADEF function. The GRADEF function is normally used to define the known derivatives of an unknown function (such as the known conservative force field due to an undetermined potential). Thus,

```
GRADEF(f(x,y),x*y,1);
```

defines the derivative of f with respect to its first argument as the product of its two arguments, and 1 to be the derivative of f with respect to its second argument. A somewhat perverse but useful application of GRADEF is to execute

```
GRADEF(f(x,y),'DIFF(f(x,y),x),'DIFF(f(x,y),y));
```

which seems redundant, but turns out to circumvent to some extent the problem of not being able to differentiate unknown functions with explicit arguments. Now,

```
DIFF(f(x,y(x)),x);
```

will yield

$$\left(\frac{d}{dx}\left(y(x)\right)\right)\left(\frac{d}{dy(x)}\left(f(x,\,y(x))\right)\right)+\frac{d}{dx}\left(f(x,\,y(x))\right)$$

as desired. The notation is still clumsy, however, and serious problems soon arise with this method. DIFF(f(x,y(x^2)),x); will not produce a complete differentiation unless GRADEF(y(x),'DIFF(y(x),x)); is first performed. One quickly sees that every time a new function is introduced, GRADEF has to be used to define the differentiation properties of the function before DIFF can be applied to it.

Applying EV with the DIFF flag to the above result, as with the DEPENDS example, starts a potentially infinite process of misinterpreting partial derivatives for total derivatives. If we try

DIFF(f(t,t),t);,

this will produce

$$2\ \left(\frac{d}{dt}\ (f(t,\ t))\right),$$

an outcome that is an obvious result of the notation employed by the differentiator. A much more serious failure is obtained by attempting to take the derivative of f(x,y(x)) with respect to x twice. In this case, the answer is simply incorrect; all terms that should be generated by differentiating

$$\frac{d}{dy(x)}\ (f(x,\ y(x)))$$

are missing as DIFF apparently does not recognize that this object is a function of x.

We conclude this survey of the MACSYMA differentiator by noting that for a majority of the standard applications one performs, the differentiation facilities are adequate. They are certainly not complete, however, and there are major areas of interest (such as solving differential equations through coordinate transformations) in which it is nontrivial to get MACSYMA to do what should be a simple procedure. MACSYMA has no trouble differentiating "known" standard functions and performs quite well for very complex problems (more on this later). For "unknown" functions of other than the simplest arguments, however, the differentiation facilities are not as good. MACSYMA does not distinguish clearly between the total derivative and partial derivatives of a function, nor does it really have a full sense of "position," in that if two arguments are the same then the derivatives with respect to those arguments are indistinguishable. In some cases, certain of these problems can be bypassed by devious means, but then care must be taken as the results may not always be correct.

3. DIFFERENTIATION IN MAPLE

The diff function is used to generate derivatives in MAPLE. For example,

diff(f(x,y),x,y,y);

produces

```
diff(diff(diff(f(x, y), x), y), y).
```

As in MACSYMA, the arguments of f are listed explicitly; otherwise, diff will assume f is a simple variable. Specifying diff with more than two arguments is simply a shorthand notation for making nested applications of this function. This is fine if functions

are being differentiated that have known derivatives, but if the functions are "unknown" to MAPLE, this will not be so nice. One immediate observation is that mixed partial derivatives will not be canonically ordered with this representation so that

```
diff(f(x,y),x,y) - diff(f(x,y),y,x);
```

will not further simplify. Nor can this problem be easily overcome as one cannot MAPLE give information about composition properties of operators.

Now, consider taking derivatives of "unknown" functions with non-atomic arguments. Suppose y is a function of x. Attempting

```
diff(f(x,y(x)),x);
```

will return the form unchanged, producing the noun form of the derivative as was the case with MACSYMA. Again, exactly like MACSYMA, MAPLE will do some simple lexical analysis when differentiation is attempted on f(x,y(x)), so that it will return zero unless an attempt is made to differentiate with respect to x (or the simple variables y or f). We note that in MAPLE, one cannot differentiate with respect to a non-atomic variable (like y(x)).

There is a mechanism in MAPLE for defining known derivatives of unknown functions (like the GRADEF function does in MACSYMA). This mechanism involves creating a procedure defining the derivative of an "unknown" function and then assigning this procedure to a special variable which the diff function will look for each time it attempts to take a derivative of this function. For example,

```
'diff/f':= proc(x,y,t) x*y*diff(x,t) + diff(y,t) end;
```

will define the derivative of f(x,y) with respect to t (this declaration is equivalent to the example presented for GRADEF). We tried to circumvent the lack of MAPLE's application of the chain rule to the unknown function y with the declaration

```
'diff/y':= proc(x,t) 'diff(y(x),x)'*diff(x,t) end;
```

which MAPLE translates into

```
diff/y := proc (x, t) ('diff(y(args[1]),args[1])')
                        *diff(args[1],args[2]) end.
```

Now, instead of returning the unsimplified noun form,

```
diff(y(x^2),x);
```

yields

$$2\ \mathrm{diff}(y(x^2),\ x^2)\ x$$

which is correct. However, attempting to perform a second derivative will produce the

message "Error, wrong number (or type) of parameters in function diff;". This is caused by MAPLE attempting to evaluate the derivative of $y(x^2)$ with respect to x^2. MAPLE will refuse to allow non-atomic variables of differentiation unless diff and its arguments are enclosed by quotes. One can attempt to get around this problem by declaring (in a fresh MAPLE - redefining the definition of a derivative a second time did not work in our version of the program)

```
'diff/y':= proc(x,t) "diff(y(x),x)"*diff(x,t) end;.
```

Thus, executing

```
diff(y(x^2),x);
```

will now produce

```
                                  2    2
2 ('diff(y(x ), x )') x
```

and taking a second derivative will yield

```
                                2    2                              2    2
2 diff(diff(y(x ), x ), x) x + 2 diff(y(x ), x ).
```

Of course, one cannot take a third derivative without redefining 'diff/y' once again. MAPLE will strip off one level of quotes every time diff is applied to an expression. This last result shows that diff really does not know how to differentiate the derivative of $y(x^2)$ with respect to x^2 and so our circumlocutions have not really accomplished more than surface changes in MAPLE's differentiator. We note that executing

```
'diff/f':= proc(x,y,t) "diff(f(x,y),x)"*diff(x,t)+
                       "diff(f(x,y),y)"*diff(y,t)  end;
```

followed by

```
diff(f(t,t),t);
```

produces the same kind of result that we saw with MACSYMA.

From our discussion, one can see that the MAPLE differentiator is a relatively unsophisticated processor when applied to functions that are "unknown" to the package. Like MACSYMA, MAPLE does not distinguish clearly between the total derivative and partial derivatives of a function. The only reason we did not get into trouble because of this is because MAPLE is not able to differentiate derivatives of "unknown" functions, producing only the unsimplified noun forms. MAPLE's representation of multiple derivatives as nested references to diff is a major trouble spot, precluding any easy solution to putting mixed partial derivatives into any kind of canonical form.

4. DIFFERENTIATION IN SMP

In SMP, the function D is used to generate derivatives. For example,

```
D[f[x,y],x,y,y]
```

produces

```
D[f[#1,#2],{#1,1,x},{#2,2,y}].
```

The arguments of f are listed explicitly; otherwise, as was the case with MACSYMA and MAPLE, D will assume f is a simple variable. The notation means the derivative of f with respect to its first argument once and with respect to its second argument twice, where the arguments are evaluated at x and y, respectively. This is mathematically correct and the results produced by D are true partial derivatives. Trying our standard examples, we find that

```
D[f[x,y[x]],x]
```

produces

```
D[f[#1,y[x]],{#1,1,x}] + D[f[x,#2],{#2,1,y[x]}] D[y[#1],{#1,1,x}]
```

and

```
D[f[t,t],t]
```

yields

```
D[f[t,#2],{#2,1,t}] + D[f[#1,t],{#1,1,t}].
```

There is no need to make SMP deceive itself (as we tried to do with MACSYMA and MAPLE) in order for it to produce fully differentiated results.

The notation employed by SMP (both on input and output), however, is rather unusual. Using brackets instead of parentheses to surround the arguments of functions (the SMP manual [3] calls f a "projector" and its arguments x and y "filters") is quite unnatural to most people working in science and engineering. Moreover, the output form of D is fairly lengthy and becomes rather unreadable for expressions containing many or complex partial derivatives. The SMP library does contain two specialized derivative formatters. Either of these formatters (contained in the SMP library files XDiffPR1 and XDiffPR2) can simply be loaded in at some point and is supposed to cause all subsequent derivatives to take on a new specific format. We found that for our first example, the notation did not change after we loaded in either file. However, the new notation did appear when we tried our second example. The notation produced by XDiffPR1 for

```
D[f[x,y[x]],x]
```

looked like

$$\left.\frac{d\ f[\#1,y[x]]}{d\ \#1}\right|_{\#1 = x} + \left.\frac{d\ f[x,\#2]}{d\ \#2}\right|_{\#2 = y[x]} \left.\frac{d\ y[\#1]}{d\ \#1}\right|_{\#1 = x}$$

while loading in XDiffPR2 produced notation that looked like

$$\frac{d\ f}{d\ x} + \frac{d\ f}{d\ y[x]}\frac{d\ y}{d\ x}.$$

The first notation closely resembles the output notation used by ATVALUE assignments in MACSYMA, while the second is very similar to MACSYMA's notation for derivatives of functions with implicit arguments. As in MACSYMA, this latter notation for derivatives of functions with non-atomic arguments does not look so well. For example,

```
D[g[x^2],x]
```

will produce

$$2x\ \frac{d\ g}{d\ x^2}.$$

Now, consider

```
D[f[x,y],y,x,y]
```

which yields

```
D[f[#1,#2],{#2,2,y},{#1,1,x}].
```

The derivatives with respect to y are collected together; however, the ordering of the arguments is not the same as for D[f[x,y],x,y,y] which was presented earlier. SMP does not put derivatives into a complete canonical form, so mixed partial derivatives will not necessarily automatically simplify. In this respect, SMP takes a position midway between MACSYMA and MAPLE. The former completely canonicalizes all derivatives while the latter does not (and probably cannot) do any sort of derivative canonicalization as it is presently written. There is a function (Dcan) in the SMP library file XDCanon which will canonicalize mixed partial derivatives. This seems a rather clumsy process, however, and we would prefer that all derivatives be completely

canonicalized automatically with perhaps a flag to completely inhibit this process for the rare cases when mixed partial derivatives do not commute. SMP does not completely implement either of these extremes.

One can take the "total derivative" of a function using the operator Dt. For example,

```
Dt[f[x,y]]
```

yields

```
D[f[x,#2],{#2,1,y}] Dt[y] + D[f[#1,y],{#1,1,x}] Dt[x].
```

This operator produces results exactly comparable to that produced by MACSYMA's DIFF function when applied directly to f(x,y) with no other arguments (forming the "total differential" of f). Note that the Dt operator is a more general tool than MACSYMA's DEL function. Dt is a true operator, while DEL is merely a place holder function which cannot simplify its arguments.

Interestingly, D can take derivatives with respect to non-atomic arguments. For example,

```
D[g[x^2],x^2]
```

produces

```
                    2
D[g[#1],{#1,1,x }].
```

In general,

```
D[f[x],g[x]]
```

will yield

```
D[f[#1],{#1,1,x}]
-----------------.
D[g[#1],{#1,1,x}]
```

Although this facility will probably not be used very often, it is nice to know that SMP has implemented such a capability.

SMP, like both MACSYMA and MAPLE, also has the ability to define the known derivatives of unknown functions. This process is performed by making an assignment directly to the relevant D or Dt reference. For example, the SMP equivalent of the MACSYMA and MAPLE examples that we have demonstrated for this operation is

```
D[f[$x,$y],$x]: $x*$y
```

D[f[$x,$y],$y]: 1

This will define the derivative of f with respect to its first argument as the product of its two arguments, and 1 to be the derivative of f with respect to its second argument. The $x and $y are generic symbols and must be used rather than just x and y for the assignments to be defined properly (D[f[x,y],x]: x*y will be only applied when the arguments of f are literally x and y). This mechanism can also be used to conveniently define the derivative of a function at a point (i.e. a boundary condition). Thus, SMP can create the same definitions as MACSYMA's GRADEF and ATVALUE functions by simply using its standard mechanism for assigning values to functions: in this case, by assigning values to the derivative operators. The only complaint is that one has to be careful in using the appropriate syntax (x versus $x versus $$x [$$x is a multi-generic symbol representing an arbitrary sequence of expressions as opposed to $x which just represents a single arbitrary expression]).

SMP seems to have avoided some of the problems of its predecessors and carefully distinguishes total derivatives from partial derivatives. This is very important. The ability to make assignments directly to the derivative operators is a nice facility. However, the notation used by SMP, although mathematically correct, is very non-standard. The output can become rather cumbersome and the current capability for displaying alternative notations is incomplete. SMP does not completely canonicalize its derivatives automatically, resulting in the annoying need to load in and apply a function in order to obtain complete simplifications of expressions containing mixed partial derivatives.

5. DIRECT TIMING COMPARISONS

To further compare the differentiation facilities in MACSYMA, MAPLE and SMP, we ran several timing comparisons on the three codes. We chose four large but simple calculations to run on the three packages. These calculations were designed to examine how quickly MACSYMA, MAPLE and SMP could differentiate and simplify a large general expression, a large polynomial expression and a large rational expression.

Tables 1 through 4 (at the end of this section) show the results of these calculations. For each problem, we ran four calculations with MACSYMA, one with MAPLE and five with SMP. Consider Table 1. We took the quantity $(x+1)^{50}$, expanded it, differentiated the resulting expression and then tried to factor the result (and hopefully get $50(x+1)^{49}$). The table shows how many seconds each operation took under various conditions (we ran all these examples under Unix on a VAX 11/780). The first column displays the time MACSYMA used for the least sophisticated applications of these operations. We simply did an EXPAND, followed by a DIFF, followed by a FACTOR, allowing normal display of the results at each step (indicated by the label "Disp" at the top of the column). We then repeated this sequence of operations, except that this time we suppressed all print out (indicated by the label "No D"). These results are shown in the second column. Notice that MACSYMA took a little less time since it did not need to format for display the rather lengthy expressions it produced. The

next two columns show the results when we used RATEXPAND and RATDIFF in place of EXPAND and DIFF. These two functions are part of MACSYMA's rational expression package. In this package, advantage is taken of a special compact form (called CRE - Canonical Rational Expression) that can be used to represent rational expressions. Functions that operate on expressions in CRE form are typically much faster than their counterparts that operate on general expressions. Expanded polynomials turn out to be particularly ideal for representing in CRE form.

The fifth column of Table 1 shows the corresponding times for MAPLE. We only ran one case (using the MAPLE functions expand, diff and factor) since we could not find a way from the MAPLE manual to suppress the output of a calculation. MAPLE is quite fast except for factoring. We note that the MAPLE manual [2] says that work on the factor function had not been completed at the time of publication of the manual.

The last five columns show results from SMP. The SMP functions for expanding, differentiating and factoring are Ex, D and Fac, respectively. SMP normally works with finite precision real numbers unlike MACSYMA and MAPLE which will normally use infinite precision integers. To be fair, we forced SMP to also work with infinite precision integers (by using the B function) for some of the calculations. The times used in applying Ex, D and Fac to B[(x+1)^50] are shown in columns 6 and 7, where we again displayed all results at each step in the first case and suppressed all the output in the second. The last three calculations allowed SMP to use its normal finite precision arithmetic. In the first of these real number calculations (column 8), we forced SMP to display its results as integers. We did this by applying B to the results of each intermediate step except the last (we did not use these integer expressions for any calculations, but only produced them for display purposes). The times shown thus include the time needed to do a B[...] of the real number expression (which was not displayed). For this example, SMP lost no accuracy by using real numbers as we found from a few quick checks of the displayed integers. In column 9 are the results for the same case, except that here we let SMP display the numbers in their normal format (scientific notation with 6 significant digits displayed). This made the calculations go much faster as SMP apparently spent a fair amount of time formatting the integer expressions. Fac did not work on the real number expressions but did work on integer expressions, so the time listed for the last calculation includes the time needed to convert the previous real number expression into integer format (using B once again). The last column in the table shows the times that were used by SMP when none of the results of the calculations were displayed.

Table 2 shows the MACSYMA, MAPLE and SMP results for performing a similar set of calculations with $(x+1)^{80}$. The dashes in the "Factor" row under SMP indicate that SMP did not complete the calculations, but instead terminated abnormally with a message to the effect that it had run out of memory. All three packages had comparable minimum differentiating times, while MAPLE was slower expanding than was MACSYMA or SMP at their fastest (of course, display time is included in the

MAPLE time, but the display was a simple linear output and so should not be too time consuming). SMP is consistently slower than MACSYMA or MAPLE when working with integer arithmetic. Our version of SMP also had a bug which caused the integer arithmetic in the expansion to be done only in finite precision. The coefficient of x^{40} was correct only through the first 16 digits. This problem would have caused the factoring to fail in any case, even if SMP had not first run out of memory.

The times required to expand and differentiate $(\cos^2 x+\sin^2 x)^{50}$ by the three symbolic mathematics programs are shown in Table 3. The dashes under the MACSYMA "Rat" columns indicate that RATDIFF will not operate on general functions of x (like $\cos x$ and $\sin x$), which is reasonable. RATDIFF was designed for taking derivatives of rational functions of x only. We had difficulties with the integer arithmetic in SMP on this problem. Unlike the previous two problems, B[(Cos[x]^2+Sin[x]^2)^50] (and other permutations) would not force SMP to work with integers. We finally tried B[2*(Cos[x]^2+Sin[x]^2)^50] and this did work. Apparently, taking B[...] of an expression will only force future integer arithmetic if the expression contains an integer coefficient (other than 0 or 1). Using this trick, we made SMP perform the expansion. We encountered a second problem when we differentiated the resulting expression. SMP successfully performed the differentiation, but then it refused to combine the resulting terms and produce zero. It was clear from the "Disp" case that all the terms canceled, but no matter what simplification function we tried, SMP would not collapse the expression to zero. It is possible that we missed something in the SMP reference manual, but we would normally call this failure a bug (as the simplification should have been done automatically), and so we have starred the times to indicate that the final answer was incomplete.

The final table (Table 4) shows the times that it took MACSYMA, MAPLE and SMP to differentiate the given rational expression. All of the differentiations produced very large expressions except for the RATDIFF cases. MACSYMA's RATDIFF produced the very nice form consisting of the ratio of two fully expanded polynomials. This form was nice since it was so compact. To reduce the expressions produced by the other differentiation operations into such a nice compact form would have required considerable simplification. Thus, the times shown for this problem do not give a complete picture of the situation.

The final results are mixed. For some problems, MACSYMA was the fastest code (when used optimally), while for others, MAPLE or SMP was faster. In general, it was faster not to display the results of a calculation, especially if the results were very lengthy. This difference in times was particularly dramatic with SMP, which seemed to spend a considerable time formatting the output of infinite precision integer calculations. Factoring was very slow in MAPLE, but MAPLE always succeeded in our sample problems while SMP ran out of memory on one occasion. SMP does not appear to be able to handle as large problems as MACSYMA and MAPLE (at least for the versions we were given). MACSYMA and MAPLE will normally work in infinite precision integer arithmetic, while SMP will normally work in finite precision real

arithmetic (about 16 digits of accuracy on a VAX). If the user is sure that any integers generated in a calculation will be within the accuracy of SMP's real arithmetic then this is a good deal, as SMP then will generally be as fast or faster than MACSYMA or MAPLE for operations like expansion and differentiation. However, if the user is not careful, he may lose accuracy unknowingly in SMP if his integers become too large. This can be a rather dangerous feature for an unsuspecting user who might be assuming that analytical solution techniques will normally be applied with infinite precision integer arithmetic. If the user attempts to use integer arithmetic in SMP, he will find that it is often slower than the integer arithmetic in MACSYMA and MAPLE as well as more likely to have bugs in its implementation (at least right now). Finally, the rational expression package in MACSYMA seems to be quite good. MAPLE and SMP have the nice feature that the user does not have to be as sophisticated as in MACSYMA in order to get fast differentiation, but then their differentiators do not differentiate rational expressions in quite so compact a way as MACSYMA's RATDIFF.

Table 1

$(x+1)^{50}$										
	MACSYMA				**MAPLE**	**SMP**				
	Disp	No D	Rat Disp	Rat No D	Disp	B Disp	B No D	(I) Disp	(R) Disp	No D
Expand	3	2	3	1	3	6	5	9	2	0.5
Diff	7	5	2	2	1	11	8	8	2	0.5
Factor	18	17	15	15	63	23	23	23	30	30
Total	28	24	20	18	67	40	36	40	34	31

Table 2

$(x+1)^{80}$										
	MACSYMA				MAPLE	SMP				
	Disp	No D	Rat Disp	Rat No D	Disp	B Disp	B No D	(I) Disp	(R) Disp	No D
Expand	7	4	4	1	6	10	7	11	4	1
Diff	14	11	5	3	3	17	15	13	2	2
Factor	42	39	39	37	206	-	-	-	-	-
Total	63	54	48	41	215	-	-	-	-	-

Table 3

$(\cos^2 x+\sin^2 x)^{50}$										
	MACSYMA				MAPLE	SMP				
	Disp	No D	Rat Disp	Rat No D	Disp	B Disp	B No D	(I) Disp	(R) Disp	No D
Expand	7	4	7	4	7	10	6	9	5	2
Diff	8	7	-	-	3	23*	17*	4	4	4
Total	15	11	-	-	10	33*	23*	13	9	6

Table 4

$$\prod_{n=1}^{13}(x-2n+1)/\prod_{n=1}^{12}(x-2n)$$

	MACSYMA				MAPLE	SMP				
	Disp	No D	Rat Disp	Rat No D	Disp	B Disp	B No D	(I) Disp	(R) Disp	No D
Diff	28	16	7	4	4	10	3	6	6	1

6. SIMILARITY SOLUTIONS OF THE HEAT EQUATION

The most important analytic method for solving partial differential equations is separation of variables. The next most important analytic method is no doubt the similarity procedure [5]. It is worth noting that, while separation of variables works only for linear equations, the similarity procedure will work for both linear and nonlinear equations. The first part of the similarity procedure derives a similarity form for the proposed solution of the given differential equation. We will not discuss this part of the technique here. After the similarity form is found, it is substituted into the differential equation. It is always possible to show (when the original equation has two independent variables) that the unknown function in the similarity solution satisfies an ordinary differential equation. The solution of this ordinary differential equation, followed by a transformation back to the original variables, results in a special solution of the original differential equation.

We will now present an example of applying the similarity technique to solving the two-dimensional heat equation. We ran this problem on both MACSYMA and SMP (MAPLE is not sophisticated enough right now to do these kinds of problems). MACSYMA took 28 seconds to complete the solution (4 seconds of this time was spent loading the ODE package), while SMP took 87 seconds performing this calculation (45 seconds was spent here loading the ODE package). The input and output from the two packages for this example are shown below.

6.1 MACSYMA Demonstration

To be able to do this problem in MACSYMA, it was necessary to be clever. We could have used our trick with GRADEF, but we found it more convenient to make use of a modification to the DIFF function that we have made [4]. This modification, invoked by loading in a file and setting the variable FDIFF to TRUE, allows DIFF to

freely apply the chain rule to "unknown" functions with explicit arguments. Derivatives with respect to arguments are indicated by subscripts. Thus,

```
DIFF(f(x,y(x)),x);
```

will now produce

$$y_1(x)\ f_2(x,\ y(x))\ +\ f_1(x,\ y(x))$$

where the subscripts indicate derivatives with respect to the corresponding arguments. A new function, REWRITE, has also been developed, which when applied to the above expression by

```
REWRITE(expression,f(x,y),y(x));
```

will convert the expression into MACSYMA's normal derivative notation (in this case, the result will be the same as that produced from DEPENDS[f,[x,y],y,x]; DIFF(f,x);).

```
(c1) LOAD(diff)$
/* Load in our modifications to DIFF */

diff.1 being loaded.
[load diff.1]

(c2) fdiff:true$
/* Make MACSYMA use our version of DIFF */

(c3) s: f(x/sqrt(t))/sqrt(t);
/* This is the similarity form of the proposed solution */
```

$$\text{(d3)}\qquad \frac{f\left(\frac{x}{\text{sqrt}(t)}\right)}{\text{sqrt}(t)}$$

```
(c4) DIFF(s,t) - DIFF(s,x,2) = 0;
/* This is the heat equation */
```

$$\text{(d4)}\qquad -\frac{f\left(\frac{x}{\text{sqrt}(t)}\right)}{2\,t^{3/2}} - \frac{f_{1,\,1}\left(\frac{x}{\text{sqrt}(t)}\right)}{t^{3/2}} - \frac{x\,f_1\left(\frac{x}{\text{sqrt}(t)}\right)}{2\,t^2} = 0$$

```
(c5) SUBST(z*sqrt(t),x,%);
/* Change to the similarity variable z = x/sqrt(t) */
```

$$(d5) \qquad -\frac{f(z)}{2\,t^{3/2}} - \frac{f_{1,1}(z)}{t^{3/2}} - \frac{z\,f_{1}(z)}{2\,t^{3/2}} = 0$$

```
(c6) RATSIMP(%);
```

$$(d6) \qquad -\frac{f(z) + 2\,f_{1,1}(z) + z\,f_{1}(z)}{2\,t^{3/2}} = 0$$

```
(c7) MULTTHRU(%,denom(lhs(%)));
```

$$(d7) \qquad -f(z) - 2\,f_{1,1}(z) - z\,f_{1}(z) = 0$$

```
(c8) eqn: REWRITE(%,f(z));
/* Rewrite the differential equation in MACSYMA's normal notation */
```

$$(d8) \qquad -f - 2\,\frac{d^2 f}{dz^2} - z\,\frac{df}{dz} = 0$$

```
(c9) LOAD(ode2)$
/* Load in the the ODE solver */

/usr/mac/share/ode2.l being loaded.
[load /usr/mac/share/ode2.l]

(c10) ODE2(eqn,f,z);
/* Now solve the ordinary differential equation */
```

$$(d10) \qquad f = \%k2\ \%e^{-\frac{z^2}{4}} - \frac{\sqrt{\%pi}\ \%i\ \%k1\ \%e^{-\frac{z^2}{4}}\ \mathrm{erf}\left(\frac{\%i\ z}{2}\right)}{2}$$

```
(c11) SUBST(x/sqrt(t),z,rhs(%))/sqrt(t);
/* Finally, transform back to the original variables */
```

$$
\text{(d11)}\qquad \frac{\%k2\ \%e^{-\frac{x^2}{4t}} - \frac{\sqrt{\%pi}\ \%i\ \%k1\ \%e^{\frac{x^2}{4t}}\ \mathrm{erf}\left(\frac{\%i\ x}{2\sqrt{t}}\right)}{2}}{\sqrt{t}}
$$

If we set $k_1 = 0$ in the previous expression and $k_2 = \frac{1}{2\sqrt{\pi}}$, we will obtain the usual fundamental solution of the heat equation.

6.2 SMP Demonstration

We did not need to modify SMP in order to do this calculation. There are a few points to note here, though. We did not work with an equation from the start because some of the SMP functions seemed only to work correctly on expressions and not on equations. The SMP ODE package required a very special form for the differential equation, so it was necessary to convert our equation into this form before applying the solver. Finally, the integral that produces the error function had to be defined, but it was reasonably easy for us to set up the definition as can be seen.

```
SMP 1.3.7
Mon May  7 22:32:11 1984


#I[1]::  s: f[x/Sqrt[t]]/Sqrt[t]
/* This is the similarity form of the proposed solution */

                 x
              f[----]
                1/2
               t
#O[1]:        -------
                1/2
              t
```

```
#I[2]::  exp1: D[s,t] - D[s,x,x]
/* This is the heat equation */

                                              x                     x
                                         -f[----]   x D[f[#1],{#1,1,----}]
                                             1/2                    1/2
                             x              t                      t
          -D[f[#1],{#1,2,----}]      -------- - ----------------------
                            1/2         2t                  3/2
                           t                              2 t
#O[2]:    ---------------------- + ----------------------------------
                  3/2                            1/2
                 t                              t

#I[3]::  exp2: S[exp1,x -> z*Sqrt[t]]
/* Change to the similarity variable z = x/Sqrt[t] */

                                     -f[z]   z D[f[#1],{#1,1,z}]
                                     ----- - -------------------
          -D[f[#1],{#1,2,z}]          2t             2t
#O[3]:    ------------------ + ----------------------------
                  3/2                        1/2
                 t                          t

#I[4]::  exp3: Ex[Num[Rat[2*exp2]]]

#O[4]:   -f[z] - 2D[f[#1],{#1,2,z}] - z D[f[#1],{#1,1,z}]

#I[5]::  exp4: S[exp3,f[z] -> y,           \
                      D[f[z],z] -> Dt[y,x],    \
                      D[f[z],z,z] -> Dt[y,x,x],\
                      z -> x]
/* Convert into a form acceptable to the ODE solver */

#O[5]:   -y - 2Dt[y,x,{x,1,x}] - x Dt[y,x]

#I[6]::  eqn: exp4 = 0
/* Make the expression into an equation for the ODE solver */

#O[6]:   0 = y + 2Dt[y,x,{x,1,x}] + x Dt[y,x]

#I[7]::  < XOde
/* Load in the ODE solver */
```

```
#I[8]::  exp5: Odesol[eqn]
/* Now solve the ordinary differential equation */

 O.D.E. Solver

 equation:       0 = y + 2Dt[y,x,{x,1,x}] + x Dt[y,x]

 1) order  =    2
 2) type   = linear: a[x] y''+b[x] y'+c[x] y=f[x]
 3) homogeneous: f[x] = 0

                                        2
                                       x
                  #k1 + #k2 Int[Exp[--],x]
                                       4
#O[8]:   {y -> ------------------------}
                          2 1/4
                     Exp[x ]

#I[9]::  exp6: exp5[1,2]

                                    2
                                   x
            #k1 + #k2 Int[Exp[--],x]
                                   4
#O[9]:      ----------------------
                       2 1/4
                  Exp[x ]

#I[10]::  Int[Exp[$$a*$x^2],$x]: -I/2*Sqrt[Pi/$$a]*Erf[I*Sqrt[$$a]*$x]
/* Define the integral of exp(a*x^2) as the error function */

                1/2                  1/2
           - Pi    Erf[($x $$a    ) I]
#O[10]:*   ------------------------------ I
                        1/2
                   2 $$a

#I[11]::  exp6
/* Apply the above definition to the solution of the ODE */

                                  1/2
              #k1          - Pi     #k2 Erf[x/2 I]
#O[11]:*   ---------- + ------------------------- I
                2 1/4                    2 1/4
           Exp[x ]                  Exp[x ]
```

```
#I[12]::  exp7: S[exp6,x -> x/Sqrt[t]]/Sqrt[t]
/* Finally, transform back to the original variables */
```

$$\#O[12]:* \quad \frac{\#k1}{t^{1/2}\,\mathrm{Exp}\left[\frac{x^2}{t}\right]^{1/4}} + \frac{-\sqrt{\pi}\;\#k2\;\mathrm{Erf}\left[\frac{x}{2\,t^{1/2}}\right] I}{t^{1/2}\,\mathrm{Exp}\left[\frac{x^2}{t}\right]^{1/4}}$$

The SMP arbitrary constants #k1 and #k2 correspond to the MACSYMA arbitrary constants %k2 and %k1/2, respectively.

7. A PROPOSED NOTATION

We propose that the best form of notation for differentiation that can be used by a symbol manipulator is the form used by many of the modern books on partial differential equations. In these books, differentiation is a linear mapping on a space of smooth functions to a space of smooth functions. This is meant to include vector valued multivariate functions. Thus, differentiation should operate on a space of smooth functions which are mappings of n dimensional space into m dimensional space, where n and m are positive integer parameters. The paper of Steinberg and Roache presented at this meeting [6] discusses the extension of these ideas to infinite dimensional spaces. We will briefly describe the scalar valued case $m = 1$. The other cases are an obvious extension of this case.

In the scalar case, one will often need to take a partial derivative of a function $y = f(\vec{x})$ with respect to one of the variables x_i, where $\vec{x} = (x_1, \cdots, x_n)$ and $1 \leq i \leq n$. Such a derivative is written

$$\partial_i f$$

and is defined mathematically as a limit. Note that the variables $\vec{x}$ in the expression $y = f(\vec{x})$ are universally quantified; that is, any point in n-dimensional Euclidean space may be substituted for $\vec{x}$. Thus, as a computation proceeds, the labels for these variables may change, a point that gives rise to much confusion. However, we may think of x_i as being a standard default label for the i-th argument of f. Then, we can use the notation

$$\frac{\partial f}{\partial x_i} = \partial_i f$$

with less chance of confusion.

This confusion is a result of the abuses of notation that commonly occur. It is often not clear in an expression like

$$\frac{\partial g}{\partial y}(x,y(z))$$

what the notation means exactly, either to the user or to the symbol manipulator! Is g evaluated at its arguments and then differentiated, or is g differentiated first and then evaluated at the point given by its arguments? Also, y is treated as an independent variable (with the assumed meaning of the second coordinate in x-y space) in one place and as the label of a dependent function in another place (as also happens with the commonly written definition $y = y(x)$). The ambiguities in this expression may be resolved in different ways by different people (and different symbol manipulators). We prefer to let the user resolve ambiguities as they occur rather than the symbol manipulator.

To evaluate a derivative at some point, one would now write

$$\frac{\partial f}{\partial x_i}(\vec{a}) = (\partial_i f)(\vec{a})$$

or in the slightly more confusing form

$$\frac{\partial f}{\partial x_i}(\vec{x}) = (\partial_i f)(\vec{x}) \,.$$

The fact that the arguments are centered at the fraction bar means that the differentiation is performed before the evaluation at $\vec{a}$ or $\vec{x}$.

One more thing is needed. Assume that an expression *expr* is given that depends on the variables $\vec{x}$ in some more or less complicated way. Then

$$TotalDiff(expr,x_i)$$

means the total derivative of *expr* with respect to x_i (i.e. the chain rule is applied repeatedly in *expr* until all explicit dependencies of x_i have been taken care of). For functions with known derivatives there is no problem, and for general functions the above notation is used.

Our favorite example would appear as follows:

$$TotalDiff(f(x,y(x)),x) \Longrightarrow$$

$$(\partial_1 f)(x,y(x)) + [(\partial_2 f)(x,y(x))]\,[(\partial_1 y)(x)] \,.$$

If it is assumed that the default notation for the arguments of f is given by $f(x,y)$ and for y is given by $y(x)$, then the above can be presented as

$$\frac{\partial f}{\partial x}(x,y(x)) + \frac{\partial f}{\partial y}(x,y(x))\frac{\partial y}{\partial x}(x)$$

Note that in the previous expression, y is being used as both a function name and a variable name. This type of confusion can only lead to disaster if all the program knows about is this last expression. However, this expression is only a displayed form of an expression where the confusion does not occur!

We have not tried to present a complete solution to all differentiation problems. We have merely indicated what we think is the best way to proceed. We do believe that this approach will solve all of the problems that we know about. Note that SMP is the only manipulator that we have studied that comes close to what we recommend.

8. SUMMARY

We have examined most of the major general symbolic mathematics packages that are currently available (we were unable to get a version of REDUCE running on our machine in time to include in this paper). None of these packages has implemented derivative operations in a completely ideal manner. All of the packages that we have examined (MACSYMA, MAPLE and SMP) will do an excellent job performing operations involving derivatives up through the equivalent of about university sophomore level. The speeds of these packages are comparable for the problems that are typical at this level. The only major drawback is the functional notation used by SMP, which is non-standard and would probably be confusing to many university sophomore students. For more advanced applications, the packages start to diverge. MAPLE and MACSYMA do not really understand the difference between total and partial derivatives. This is a serious defect that it makes it quite difficult to do certain kinds of problems. MAPLE and SMP do not readily recognize the equivalence of various forms of mixed partial derivatives.

We conclude with some general remarks about each of the symbolic mathematics packages discussed in this paper and some considerations about what kind of derivative notation is best for these programs. MACSYMA is a "mature" code. Its integer arithmetic is fast and it has many sophisticated packages. It cannot solve the heat equation, though, without the user modifying or tricking the code. Our modifications to the DIFF function were really relatively minor, so there is no reason why MACSYMA cannot be made generally more intelligent about differentiation. This has just not been done, however (at least, not through the time of the last publication of the MACSYMA manual). MAPLE is not really very sophisticated yet. It is hard to do problems beyond the sophomore level with this package. SMP is not a "mature" code. It has a number of bugs right now. Some of its calculations involving integer arithmetic can be slow compared with MACSYMA and MAPLE. SMP does, however, do

its differential calculus fairly correctly (it could solve the heat equation, for example, without resorting to new coding or non-obvious circumlocutions). Finally, we propose that the best form of notation to be used by a symbol manipulator for derivatives is one in which the notation is unambiguous as represented internally in the program, and which can be used to produce various forms of "natural" looking output for the user to read. In this notation, the dependent and independent variables must be carefully distinguished. Also, the order of operations (differentiation versus function argument evaluation) should always be made very clear.

References

1. *MACSYMA Reference Manual (Version Ten).* The Mathlab Group, Laboratory for Computer Science, MIT, January 1983.

2. Geddes, Keith O., Gonnet, Gaston H. and Char, Bruce W. *MAPLE User's Manual (Second Edition).* Research Report CS-82-40, Computer Science Department, University of Waterloo, December 1982.

3. *SMP Reference Manual (Version One).* Inference Corporation, 1983.

4. Wester, Michael and Steinberg, Stanly. An Extension to MACSYMA's Concept of Functional Differentiation. *SIGSAM Bulletin*, Volume 17, Number 3 & 4 (August & November 1983), p. 25-30.

5. Bluman, G. W. and Cole, J. D. *Similarity Methods for Differential Equations.* Springer-Verlag, New York, 1974.

6. Steinberg, Stanly and Roache, Pat. Using VAXIMA to Write FORTRAN Code. To be in *The Proceedings of the Third MACSYMA User's Conference*, Schenectady, New York, July 23-25, 1984.

AN ALTERNATE TOP-LEVEL FOR MACSYMA

Gene Cooperman
GTE Laboratories, Inc.
40 Sylvan Road
Waltham, MA 02254

Abstract

An alternate top-level for MACSYMA has been created that is expected to be more natural for use in manipulating large expressions, which include derivatives. The features include a different default semantics using mathematical variables (i.e., variables that are not evaluated, except by explicit commands), instead of programming variables. Dependencies of variables are declared automatically when the user writes an assignment statement, thereby making the chain rule easier to use. Finally, a new infinite evaluation function has been written to do "bottom-up" infinite evaluation, which is generally more efficient.

It is expected that these features, when combined together, will provide a more natural environment for a naive user with no previous computer algebra experience. It may also have special advantages for the user who has a large expression, possibly involving derivatives, and wishes to get quick numerical answers without extensive programming.

1. DISCUSSION

The work was motivated by the following equations encountered by the author in a physics problem concerning superlattices. [1]

```
F:   COSH(2*K1*A)*COS(2*K2*B)+(EPS/2)*SINH(2*K1*A)*SIN(2*K2*B) ;
EPS: K1/K2-K2/K1 ;
K1:  SQRT(2*M0*(V0-E))/HBAR ;
K2:  SQRT((2*M2/EG)*(E**2+E*EG))/HBAR ;
DEPENDS('F,'[K1,K2,EPS],'EPS,'[K1,K2],'[K1,K2],E) ;
D2:  DIFF(F,E,2),INFEVAL,DIFF ; /* assumes parameters
                        such as m0 were previously defined */
E: 1.0 ;
D2, INFEVAL,DIFF,NUMER ;
```

It was required to differentiate F with respect to E, twice, and then plug in values for all constants. Parameters such as 2*M2*EG could not be pre-computed, since the values of the parameters changed from one case to the next. The second derivative might be evaluated with pencil, paper, and a calculator in about fifteen minutes. Surprisingly, MACSYMA also takes about fifteen minutes of CPU time on a VAX 780, with the above brute-force evaluation using a top-down evaluation scheme. Alternatively, one could spend a half hour or more programming MACSYMA to do a bottom-up evaluation for this equation, whereupon it would numerically evaluate the second derivative in under a minute.

As part of the alternate toplevel, it was hoped to be able to automatically handle such problems in the following simple manner.

```
F:   COSH(2*K1*A)*COS(2*K2*B)+(EPS/2)*SINH(2*K1*A)*SIN(2*K2*B) ;
K1:  SQRT(2*M0*(V0-E))/HBAR ;
K2:  SQRT((2*M2/EG)*(E**2+E*EG))/HBAR ;
EPS: K1/K2-K2/K1 ;
DF: DIFF(F,E,2) ;
E: 1.0 ;
```

```
BOTTOM_UP_INFEVAL(DF); /* assumes parameters such as m0 were
                          previously defined */
```

Instead of writing a separate package with special commands, it was hoped to define a different semantics in which the above problem could be solved quickly, and yet in a mathematically natural way, such as one of the above. The first set of commands, above, was already fairly natural, but took too much CPU time. Changing syntax, or providing a special-purpose package was felt to be more likely to make such an evaluation less natural. So, it was decided to change the semantics of MACSYMA.

One difference in the two examples above, is that the first one will behave differently according to the order in which are listed the equations for F, EPS, K1, and K2. This was because MACSYMA's current semantics specify that if a variable is bound, its value is used, but if the variable is unbound, its name is used as the value. (i.e.: Unbound variables are not evaluated.) This evaluation scheme is reminiscent of LISP, the language in which MACSYMA was written. However, with this scheme, we lose the non-procedural statement of problems used by many mathematicians. The order in which the equations are stated affects whether values are substituted for variables, and hence affects the steps necessary for numerical evaluation. In order to prevent just this problem, people often liberally use quotes to avoid premature evaluation of variables.

It was felt that this behavior of the variables was not natural to the working mathematician. We shall refer to the type of variables currently used in MACSYMA as programming variables. [2] The use of such variables were presumably motivated by the MACSYMA designers in analogy with LISP, and their implicit desire to use MACSYMA more as a math programming language than a "natural language" math system.

In line with the second goal of a "natural" system, we chose to consider all variables as mathematical variables, by default. A mathematical variable would never be evaluated (replaced by its value), unless explicitly evaluated with the evaluation function "VAL()". The user has the freedom to declare variables to be programming variables. In the future, certain syntactic constructs may automatically determine such a declaration. (For example, "I" in FOR I:1 THRU 10 ...;).

To implement the alternate top level for MACSYMA, we wrote a routine called immediately after the built-in Macsyma parser. This routine puts a BLOCK command around the resulting LISP expression, which declares all variables to be local. This has the effect of pushing their values onto a stack. Hence, in the local context, the variables appear to be unbound, and are not evaluated, in keeping with our ideas on mathematical variables.

Naturally, this required care that in certain constructs, such as X:Y;, the variable X should not be declared local. Otherwise, the value of X would be lost in returning to the global context. In handling such special cases, it was also found convenient for each MACSYMA assignment statement of the form X:EXPRESSION; typed by the user, to have the alternate toplevel automatically declare to Macsyma the following information.

```
REMOVE(X,DEPENDS) ;
DEPENDS(X,LISTOFVARS(EXPRESSION)) ;
```

The effect of this addition is to automatically record the dependencies of the variables as they are stated, thereby allowing the chain rule to work automatically without explicit declarations by the user. Since only the value of a variable declared local to a BLOCK statement is pushed onto a stack, and not its properties such as DEPENDS, this idea nicely complements our implementation of mathematical variables. The automatic dependencies seem to do "the right thing" in almost all cases of practical interest. Most likely, this had not been done for the original MACSYMA because of problems of a limited-address machine. On most of the current machines running MACSYMA, this is no longer a problem.

In our top level, a variable, Y, can have evaluation forced by typing ",Y" at the end of a command. Such a variable will not be included as local to the BLOCK command. However, there is a danger in such commands as:

```
DIFF(X,Y)*DIFF(Y,Z),Y;
```

Hence we were forced to redefine the DIFF command to never evaluate its second-position argument, unless that argument was a programming variable.

Finally, a function that does infinite evaluations from the bottom-up was written. As can be seen in the first set of equations, since K2 appears implicitly in F four times, 2*M2/EG would be calculated four times in a brute force approach. This overhead would become intolerable in calculating DIFF(F,E,2), where 2*M2/EG would be evaluated about 40 times by a brute-force technique.

Tests have been run, infinitely evaluating f, with the parameters set to bigfloat numbers. Neglecting garbage collection, the command F,INFEVAL,BFLOAT; required 30 CPU seconds on a VAX 780. In comparison, our own function BFLOATEVAL(F); required 10 CPU seconds, neglecting garbage collection. Garbage collection times were roughly proportional, but not repeatable. The ability of the our BFLOATEVAL function to correctly take account of arrays and derivatives has only recently been added. Tests are in progress to compare DF,INFEVAL,DIFF,BFLOAT; with BFLOATEVAL(DF);, where DF:DIFF(F,E,2).

A comment should be made about the usefulness of the above methodology. Clearly, many physical models come ready-made with their own hierarchical levels of equations. For such a problem, our alternate infinite evaluation is eminently useful. Where an expression does not explicitly contain these hierarchical expressions, but still contains redundant operations, such as "2*M2/EG+COS(2*M2/EG)", our infinite evaluation routine is also useful. In this case, a form of the MACSYMA command OPTIMIZE can be used to create such a set of hierarchical equations. Further, the alternate top level should be compatible with share packages previously translated from MACSYMA-level into lisp code, since our technique can be viewed an alternate parsing of the MACSYMA-level code.

2. SUMMARY

It should be emphasized that with the possible exception of the infinite evaluation scheme, this top level does not bring any new capabilities to

MACSYMA. Instead, it is expected to be easier to use for a naive user who does not intend to use the full capability of MACSYMA as a programming language, and wishes to interact in a manner close to his own "natural language" mathematics. Experiments with such users are planned for the near future.

I would like to thank Jeffrey Golden for providing invaluable insight into MACSYMA and the issues surrounding MACSYMA.

References

1. Cooperman, G., Friedman, L., and Bloss, W.L., Corrections to Enhanced Optical Nonlinearity of Superlattices. **Applied Physics Letters**, Vol. 44(10), (May 15, 1984), pp. 977-979.

2. Moses, J., The Variety of Variables in Mathematical Expressions. **Proceedings of the MACSYMA Users' Conference, NASA**, Berkeley, Ca., (July 1977), pp. 123-129.

Computational Geography - The Habitats of the Migratory MACSYMA

V. Ellen Golden
MACSYMA Group
Symbolics, Inc.
257 Vassar Street
Cambridge, Massachusetts 02139

1. Introduction

MACSYMA was written at the M.I.T. Laboratory for Computer Science (then called Project MAC) to run on the Incompatible Time-Sharing System (ITS). ITS is the operating system written at MIT's Artificial Intelligence Laboratory for the PDP-10 computer. MACSYMA was written in MACLisp, a dialect of Lisp also written at the A.I. Lab. Many of the facilities and features of MACLisp were created in response to MACSYMA's needs. The hardware on which MACSYMA was developed is a PDP-10, a machine with an addressable memory of a limited size. In addition, since MACSYMA was developed and used at only one site (a MACSYMA was brought up on the MIT-Multics system in 1975 but never maintained), many features were designed with the PDP-10/ITS/MACLisp environment in mind.

MACSYMA remained an ITS-only system until 1979 when Multics MACLisp was updated and MACSYMA was re-ported to Multics by the MIT Mathlab Group [1]. It was also ported to the VAX UNIX system by Professor Richard Fateman of U.C. Berkeley, and to the Lisp Machine and TOPS-20 by the MIT Mathlab group. More recently it has been brought up on a 68000-based workstation by Professor Fateman at Berkeley.

This paper will discuss briefly the systems to which MACSYMA has been ported. The various sections have been or will be expanded into "primers" for the various operating systems, following the pattern of "An Introduction to ITS for the MACSYMA User" [3]

---

2. Operating Systems and Their Differences

Each operating system has its own particular features. ITS, TOPS-20, and the Symbolics 3600 for instance, are character-at-a-time systems, meaning that the operating system reads each character as it is typed. UNIX and Multics are line-at-a-time systems, meaning nothing happens until you type a carriage return. This means that MACSYMA commands may be terminated by a single character (; or $) on ITS, TOPS-20, and a 3600, but must be terminated by a ; or $ followed by a carriage return on VAX systems and on Multics.

File systems are also different on different operating systems. The ITS file system is quite primitive. It has a single level directory scheme (no sub-directories) and only two file names. Directory names and file names are limited to no more than six characters. MACLisp originally (in so called "Old I/O") referenced files as a list, (*name1 name2* DSK *directory*). MACSYMA file referencing was designed similarly, [*name1*,*name2*,DSK,*directory*]. When the Input/Output system of MACLisp was redesigned ("New I/O"), a more flexible format for file referencing was installed, a string format. (As well as a new list format which we won't go into here.) On ITS both formats are used in MACSYMA, you may reference a file as [*name1*,*name2*,DSK,*directory*] or as ["DSK:*directory*\;*name1 name2*"] (note the "\" which precedes the ";" so that MACSYMA does not think the ";" is a command line terminator). UNIX and Multics have hierarchically structured file systems with pathnames of varying length. These variable length pathnames do not translate easily into the familiar ITS list format, but do work very well as strings, inside quotation marks (double quotes), so this is the format used on non-ITS systems (with the exception of TOPS-20, which like ITS may use either format). MACSYMA commands which on ITS look like:

```
DEMO(BEGIN,DEMO,DSK,DEMO);
```

become on a UNIX system:

```
demo("/usr/macsyma/demo/begin.dem");
```

A number of MACSYMA commands were written to reference, list, and print files on the ITS operating system. These commands, DISKFREE, DISKUSE, FILELENGTH, FULLDISKUSE, LISTFILES, PRINTDISKUSE, PRINTFILE, QLISTFILES, and RENAMEFILE, will not work on other operating systems without further work.

Similarly, the commands BUG, MAIL, and SEND utilize the ITS electronic mail facilities, and work would be required to make them work under electronic mail programs on other systems.

An operating system will impose some constraints on the Lisp system in which MACSYMA is embedded, also. All of the Lisps in which MACSYMA currently runs are designed to be as close to MACLisp as possible or to be upward compatible with MACLisp, and to provide the facilities of MACLisp upon which MACSYMA depends. However, operating system considerations connected with

the file system and the structure of interrupts and job control sometimes result in slight differences, which may impact on what the user sees in MACSYMA.

3. TOPS-20

MACSYMA on the TOPS-20 is the closest to the original ITS MACSYMA. This is true for two reasons. First, the hardware upon which TOPS-20 runs is from the same family of machines. Second, the Lisp system is a MACLisp. The hardware being similar means that a TOPS-20 MACSYMA, like the ITS version, can run out of address space: the frustrating "No Core Available - You will have to start a new MACSYMA" message.

The file system is different, of course. It can have a hierarchical structure, and filenames have three elements as well as the device and directory, e.g. PS:*<directory>name1.name2.generation number*. Since each pathname will have five elements, it is possible to translate this format to the ITS list format, and as a result files on TOPS-20 may be referenced by either list or string format, as on ITS.

The following commands do not work in TOPS-20 MACSYMA at this time: BUG, COMPILE, DISKFREE, DISKUSE, FILELIST, FULLDISKUSE, LISTFILES, MAIL, PRIME, PRINTDISKUSE, QLISTFILES, SEND, TRANSLATE, WHO, and WORLDPLOT.

4. Multics

The Multics operating system was written at MIT's Project MAC for the Honeywell 6180 processor. Unlike the ITS and TOPS-20 system, Multics has "Virtual Addressing", which means that you are not constrained in how much memory you can use. The problem of running out of core does not present itself, although it may be necessary to request a larger "process directory". The file system is very elaborate, with great flexibility in organization and security. (The U. S. Air Force has called Multics the most secure time-sharing system available.)[5] MACLisp was ported to Multics in 1974-75 by David Moon, David Reed, and Alex Sunguroff specifically to bring up MACSYMA.[4]

The Multics system distinguishes between upper and lower case, which means that `macsyma` and `MACSYMA` are two different things to the Multics monitor, and both of them are different from `Macsyma`. This means that care must be taken with file names (or "pathnames" as they are called) to type them precisely as they appear. However, MACSYMA commands may be typed in upper or lower case and MACSYMA will translate them to lower case.

Because Multics is a line-at-a-time system, a carriage return is necessary after the ; or $ which terminates a MACSYMA command, and control characters do not work the way they do on ITS. To type the common MACSYMA control characters on Multics, hit the BREAK key on your terminal. (If you are logged in over a network, send the appropriate ATTENTION signal.) Then type the character whose control function you want and a carriage return. For example, to type a control-G to stop a computation, you would hit the BREAK key followed by G<*carriage return*>.

The control structure of the Multics monitor is somewhat different from the ITS system. You can exit from MACSYMA to "a monitor" and do whatever you wish, but this will not be the same monitor "level" at which you started. It is more like a chain, where you start at the Multics monitor (link 1), start up MACSYMA (link 2), exit to "monitor" (link 3). Now, how do you get back to your MACSYMA? Actually, this is quite simple, but perhaps counter-intuitive to users who have been accustomed to ITS: you type `start`.

Another way to describe it is that the structure of Multics is modeled on Dante's Inferno: there are 7 rings. You can exit via the DDT(); function in MACSYMA or by hitting the BREAK key twice. This gives you a "new level". If you don't `start` (or `release`) this level, but hit BREAK again, you will end up in another level, and confusion can appear to reign. `start` will return you to your previous level, and restart any processes. `release` will get you back to the previous level, but will not necessarily cause any job to continue. `release -all` will get rid of everything you were doing and leave you at the point at which you logged in. The Multics system warns you in its "ready" message about the level on which it has currently put you.

There are some commands in MACSYMA which are specially designed to allow you to access the Multics monitor from inside MACSYMA:

`cwd("pathname");` change working directory. This takes a pathname as an argument and sets your working directory to that pathname.

`pwd();` print working directory. This can help you if you want to use default directories or pathnames as much as possible but are not sure "where" you are.

`cline("`*command*`");`
Executes the command (which should be a Multics monitor command line) from inside MACSYMA. E.g. `cline("ls");` would list your current working directory.

There are also some differences in file handling commands, as a result of the differences in the Multics file system. On the ITS system you could use a command like SAVE and it would choose a file name for you if you did not specify one. Since the Multics file system is more complex, it is necessary for you to supply a pathname for the SAVE command. This is typed as a string, i.e. inside quotation marks, and enclosed in square brackets, thus:

```
save(["pathname"],arg1, arg2,...,argi);
```

Similarly, the LOAD command requires a full pathname, inside quotation marks:

```
load("pathname");
```

The PLOT2 package, PRIME command, and the on-line PRIMER do not work in Multics MACSYMA at this time.

5. VAX UNIX

UNIX is an operating system developed at Bell Laboratories and enhanced by the University of California at Berkeley. It runs on a number of machines. Two machines running UNIX which also run MACSYMA are DEC VAXes and The SUN Workstation. The version of MACSYMA which runs under UNIX runs in Franz Lisp, which was developed at Berkeley. Franz Lisp was designed to provide the features which MACSYMA needs, in other words it was designed to be very similar to MACLisp.

The UNIX monitor is called the Shell. Like the Multics monitor, the UNIX shell distinguishes between upper and lower case, so care must be taken to preserve case. Commands to MACSYMA may, however, be typed in upper or lower case and MACSYMA will itself translate them into lower case. UNIX is also a line-at-a-time system, so ; and $ require a terminating carriage return. Some control characters in MACSYMA similarly require a carriage return before they take effect, e.g. control-K.

Many aspects of the system were patterned after Multics ("UNIX" is one "Multics"). There is a MACSYMA command `shell();` which allows you to exit from MACSYMA to a UNIX shell on a different level, as with Multics. To return to your MACSYMA from this new shell, you type `logout` or control-D. There is also a `control-Z` which exits from MACSYMA to your original shell, in a manner similar to ITS or TOPS-20. "*%*" (or "*%n*" where *n* is the job number) will put you back in your MACSYMA. `jobs` will give you a list of the programs you have running, similar to the :LISTJ command on ITS. Each job will have its number next to its name so that you may refer to it with the *%* command. On ITS each job is assigned a unique number by the system, but these numbers are of little importance for most MACSYMA users. On UNIX on the other hand, job numbers are used to refer to jobs for purposes of other commands. The numbers are assigned on a per-user basis, and yours will be 1, 2, 3... in the order which you started up your jobs. Let us assume you went into an editor first, exited and started a MACSYMA, and then exited from it with `control-Z` and started to send some mail. If you now interrupt your mail job and type `jobs`, you will find something like this:

```
[1]  - Stopped              emacs
[2]  - Stopped              macsyma
[3]  + Stopped              mail
```

The "+" indicates that the mail job is the most recently used job (the one from which you just exited) and the one which the *%* command will re-enter if you do

not follow it with a number. To re-enter your MACSYMA at this point you would type *%2<carriage return>*.

There is a MACSYMA command `exec("command");` which takes a UNIX monitor command and sends it up to the shell for evaluation. For example, `exec("ls");` would list your directory.

The control-characters available in UNIX MACSYMA are slightly different from ITS MACSYMA. Control-L, Control-K and ?? work (requiring a carriage return, as noted above). Control-G, the ITS "Quit" character, does not work. The "Quit" character on UNIX is Control-C. Typing Control-C to MACSYMA will enter an interrupt loop. You then select by means of single letter commands whether you wish to kill your MACSYMA, enter a MACSYMA break, enter a Lisp break, or return to top-level MACSYMA. Control-A which enters a MACSYMA break on ITS does nothing in UNIX MACSYMA. Control-Y does not work as it does on ITS, where it retrieves your last command line. In UNIX MACSYMA control-Y has a somewhat surprising result to a former ITS user: it stops the job and returns you to top level. Fortunately, it does not kill the MACSYMA, you may re-enter it with %.

The "escape" key (followed by a carriage return) enters the editor. In UNIX MACSYMA 304 the default editor in UNIX MACSYMA is the UNIX VI editor. The line editor available on ITS may be used but you need to load it explicitly:

```
loadfile("/usr/macsyma/jpg/medit.o");
```

You exit from the VI editor by typing :wq. You exit from the line editor by typing two escapes, as on ITS. Starting with MACSYMA 306 the line editor will be the default. You will be able to use VI by loading it explicitly:

```
loadfile("/usr/macsyma.306/ucb/medit");
```

UNIX has a display processor, EQN, which produces phototypset output for mathematical expressions. You can use the MACSYMA `writefile` command in combination with several switches in MACSYMA to produce files which can be run through EQN to produce nice output.

The On-line PRIMER works on the UNIX system, but it has a special introductory script called CONSOLEPRIMER which is intended for users who are new to computers. When you have run this script, your username is added to a file which the PRIMER maintains. When the PRIMER is started up, the file is checked to see if you have seen the CONSOLEPRIMER script. If you have, the PRIMER starts with a menu of the available scripts, instead of making you plod through the introductory material again.

When MACSYMA was ported to UNIX, it was necessary to rewrite this feature of the PRIMER since Franz Lisp's manner of updating files was somewhat different from MACLisp's. Sites which have versions 303 or 304 of UNIX MACSYMA (or 305 of EUNICE or REX MACSYMA), will find that the PRIMER *does not work if you start it up with*

```
PRIMER();
```

because when it tries to update the file of who has run the PRIMER, it fails to

find the file. You can get the PRIMER to work in these versions if you start it by typing:

```
primer(help);
```

which will start it up with the menu of scripts from which to choose. This bypasses the CONSOLEPRIMER script, which in any case is written with the ITS environment in mind. Scripts are selected by typing the script number followed by a semi-colon and a carriage return. In MACSYMA 306, the problem with the PRIMER maintaining its file has been fixed and a replacement for the CONSOLEPRIMER script, called `franzprimer`, has been included.

In addition to the file referencing commands, which do not work anywhere except on ITS, the following commands are not available in UNIX MACSYMA at this time: COMPILE, MAKE_ARRAY, PRIME, TRANSLATE, and WHO.

Computing determinants by the SPARSE:TRUE$ method does not work at this time.

6. VAX VMS

There are three different VMS versions of MACSYMA: EUNICE MACSYMA, REX MACSYMA and NIL MACSYMA. NIL MACSYMA was developed at MIT and in the "New Implementation of Lisp" (NIL). NIL MACSYMA is not widely distributed at this time. EUNICE and REX MACSYMA are distributed by Symbolics. Both of them make use of the UNIX emulator EUNICE in order to provide the Franz Lisp environment for MACSYMA to run. Those machines which are running a full EUNICE have the EUNICE version of MACSYMA. EUNICE MACSYMA works in the same way as UNIX MACSYMA, since the EUNICE emulator makes the operating system look and behave like UNIX. For those machines which are not running EUNICE, a portion of EUNICE is compiled in with the MACSYMA. This is the Runtime EXecutable portion, called REX. REX MACSYMA is slightly different from EUNICE and UNIX MACSYMA, most notably in those areas where interaction with the operating system is required. Plotting is one notable area where REX MACSYMA falls short. There are plans to have the PLOT2 package working in REX MACSYMA in the future, but at present only the simple character plotting package works. Naturally the type-setting facilities do not exist.

VMS does not distinguish between upper and lower case, but due to some interaction between Franz Lisp, REX and VMS, filenames must be typed in lower case in REX MACSYMA. `Control-Y` exits from MACSYMA, as it does on UNIX. Here you get back to the VMS monitor, and must type CONTINUE*<carriage return>*, much as on TOPS-20, to re-enter your MACSYMA.

7. 3600

The Symbolics 3600 is a large single user machine developed at Symbolics for research computation. It uses a dialect of Lisp called ZetaLisp. It has 36 bit words and virtual addressing. With its integrated editor, window system, and mouse, the 3600 is a complete working environment. It offers many possibilities for MACSYMA to improve and extend its user interface in exciting ways.

Work is being done to take advantage of the facilities of the 3600. For instance, it should be possible to select a part of an expression by pointing at it with the mouse and then operate on it, recombine it, and continue the computation. A display editor for MACSYMA expressions, menus for OPTIONS and DESCRIBE, and expanded help facilities are all possibilities.

The 3600 has excellent graphics capabilities which the plotting routines take advantage of.

8. Plotting

There are two plotting packages in ITS MACSYMA, the simple character plotting package which works on any terminal, and the display oriented PLOT2 package. The simple character plotting package is not dependent on specific kinds of terminals or terminal drivers and thus can be easily ported to any operating system. It works in all versions of MACSYMA except for the UNIX version, where it has been intentionally replaced with commands for the UNIX plotter. DESCRIBE(PLOT); in UNIX gives information about how to use the UNIX plotting package.

The PLOT2 package is available in 3600, UNIX and EUNICE versions of MACSYMA, and in TOPS-20 versions. It is not currently available in REX versions or on Multics. Multics does have its own plotting package, however, and some facilities have been provided in Multics MACSYMA to allow you to use the Multics plotting package You must first give the command `setup_graphics` *terminal_type* to the Multics monitor, and then set the MACSYMA variable MULTIGRAPH to TRUE. Then the functions PLOT and PLOT3D will use the Multics graphics system to produce screen plots for you.

The `worldplot` command, which was written as a display hack anyway, does not work on any system except ITS.

9. Share Programs

The Share Library is extremely large and growing all the time. Some of the programs on the Share directories on the MC machine have not been used in years, and including them in porting efforts has not seemed worth it. However, in most cases there is no reason that a particular program cannot be made to work on any of the operating systems.

Work is proceeding on cleaning up, checking out, and making any needed modifications to files on the Share directories and in future releases of MACSYMA they should all be included.

10. References

1. Bawden, A., Burke, G., and Hoffman, C.W., "MACLisp Extensions", MIT-LCS/TM 203, July, 1981.

2. Carney, Charles, Implementation notes on the plot package.

3. Foderaro, John K., "Typesetting MACSYMA Equations", Proceedings of 1979 MACSYMA Users Conference, Washington, DC, June 20-22, 1979.

4. Golden, V. E., "An Introduction to ITS for the MACSYMA User", Mathlab Memo #3, revised Sept. 1982.

5. Moon, David, "MACLisp Reference Manual", MIT Laboratory for Computer Science, 1975.

6. "IPS Users Guide - An Introduction to Academic and Research Computing at Information Processing Services", Massachusetts Institute of Technology, 1981.

A Functional Language Machine and Its Programming

Donald F. Stanat
Department of Computer Science
University of North Carolina
Chapel Hill, North Carolina

Abstract

The FFP machine being developed at the University of North Carolina is a small-grain multiprocessor that directly executes Backus's FFP languages. The language provides a number of attractive features, including semantic simplicity, the ability to express many forms of parallelism, and a powerful algebra of programs. The machine supports a programmer's intuitive notions of program execution, is capable of executing powerful operations as machine primitives, and provides a rich basis for program optimization. This paper describes some of the important features of the language and the machine and how they may affect the programming process.

Introduction

Modern computing can be viewed as beginning with the introduction of the 'von Neumann' computer architecture, which originated when technology dictated an implementation using vacuum tubes and relays. Although the technology used to implement the machines has undergone a number of revolutionary changes, our contemporary machines remain unchanged in their basic design. Similarly, our most visible languages, such as Pascal and Ada, evolved from FORTRAN, the first high-level language for these machines. The improvements in both languages and machines have been substantial but far from satisfactory; the machines are still basically sequential, and the expressive power of the languages has not substantially increased. We are near the end of the evolutionary development of von Neumann machines and their languages. Satisfying the demand for greater computing speed as well as increased ease of programming will require approaches to language and machine design that are different from those in widespread use today.

What might we ask of a programming language? I will address only the matter of general purpose programming languages, those that can handle a wide variety of problems and do not sacrifice ease of programming in one area to accommodate the problems in another. First, we wish these languages to be suitably expressive, and hence capable of supporting arbitrary levels of abstraction. Second, we want them to make programming a mathematical activity in the sense that, although mathematical rigor is not *de rigueur*, it is nevertheless feasible. These two requirements specify that the languages will allow us to think in a natural way and will provide mathematical support for that thought.

A final requirement is quite different in nature, and stems from the fact that programs represent computations, and the same result can be computed in different ways. If we are to be able to exploit our machines well, the language we use must allow us to control computations in ways that will affect the use of resources, including time, space and the degree of parallelism.

What can we ask of a machine architecture? Operationally, we simply want it to execute our programs correctly and quickly, but this glib answer implies a host of specifics. First, the architecture should be related to the programming language in a way that allows (but preferably does not require) the programmer to exercise some control over allocation of resources. In order to satisfy the need for massive computation, the architecture should be capable of parallelism at a variety of levels, at least the highest of which can be controlled by the programmer. And finally, we would like the machine to be cheap. Cost is clearly technology-related, and assessing cost is complex. But we may safely assume that a low-cost design will make good use of the best technology available. The best technology today is VLSI, and using VLSI well implies the use of many copies of a small number of chip designs.

The desires to increase both computational speed and ease of programming are usually considered to be conflicting. Perhaps the most common approach to this pair of problems has been to design a machine for high performance in some limited context, hoping that programmability can be added on or incorporated into the design at a later stage. While one can always hope, the failure of so many architectural experiments because of a lack of programmability rather than a lack of speed should lead us to expect that a design that does not incorporate programmability *ab initio* will fail.

A promising approach to these problems is a language-based computer architecture that supports a language capable of expressing a variety of forms of large-scale parallelism. Work is presently in progress at the University of North Carolina at Chapel Hill on a machine design that we feel holds great promise, both because it is capable of a high degree of parallelism and because the machine language has many desirable properties. The language is the FFP (Formal Functional Programming) language proposed by Backus [BA78]. The machine is a fine-grain multiprocessor proposed by Mago [MA79]. In Section 1 of this paper we will describe informally the FP (Functional Programming) languages, which are essentially 'user-friendly' versions of the FFP language. In Section 2 we will describe the FFP machine. In Section 3 we will describe the outlook for programming this machine.

1. FP LANGUAGES

There is a growing belief that programmer productivity can be improved by utilizing languages that support careful thought and design better than von Neumann languages do. Until recently, the most vigorous challengers to von Neumann languages were functional programming languages; recent developments have made logic programming languages contenders as well. These classes require quite different programming styles and impose different computational demands. The FFP machine project is actively investigating the

possibility of executing logic programs on an FFP machine [SM84], but that part of the project is in its infancy and will not be covered here.

Functional languages have been around a long time; LISP is almost as old as FORTRAN [MC60]. Many functional languages have been described and implemented. Flavors of LISP abound. Newer and less established functional languages include ISWIM [BU78, LA79], KRC [TU81], VAL [AD79] LCF [GMW79] and FP [BA78]. Each has its own set of virtues and advocates. Because the FFP languages of Backus are the machine language of the multiprocessor we wish to describe, we will concern ourselves only with these languages and the closely-related but more easily understood FP languages.

1.1 Description

We begin by describing informally the FP languages that Backus introduced in his Turing Award Lecture [BA78]. These are purely functional languages in which a program and its input form an expression; the result, or output of the program, is the value of the expression. Execution of the program consists of evaluating the expression. The value of a program is defined by a fixed-point denotational semantics, but this value corresponds to an operational semantics based on the rewriting (reduction) of innermost expressions. Thus, a program is an expression, and evaluation of the program is done by iteratively replacing certain innermost expressions called ***reducible expressions*** or ***reducible applications*** (RAs). Each RA consists of a function applied to an argument, and each RA is replaced with another expression of the same value. FP languages have the Church-Rosser property, so RAs can be re-written in any order, or in parallel. Program execution terminates when the program expression does not contain any RAs.

An FP language is specified by a set of ***atoms*** A, a set of ***primitive functions*** F, and a set of ***functional forms*** G. The set A typically includes various representations of numbers as well as characters and boolean values. A ***well-formed expression*** (wfe) is (a) an atom, or (b) a sequence $< x_1, x_2, \ldots x_n >$, where each x_i is a wfe, or (c) an application $(f: x)$, where f is a function expression and x is a wfe. A function expression is an element of F or a representation of a function built from members of F, G and E, the set of well-formed expressions. Note that FP languages are 'variable-free.'

An FP *object* is an expression that contains no applications; that is, an object contains no unevaluated functions. An object is either an atom or a sequence of objects. Because the reduction semantics evaluates innermost expressions, an RA consists of an application of a function expression to an object.

The power of FP languages comes largely from the rich set of functional forms (program-forming operations), a parsimonious notation that expresses parallelism well, and an algebra over the set of programs.

1.2 Features

1.2.1 Semantics

FP languages share with other functional languages a semantics that should facilitate

creation of systems to prove such properties as program correctness and equivalence. Although FP languages are basically typeless, types can be added [FR81, GHW81]. Thus, although little work has been done in this area, FP languages appear to satisfy the criterion that programming environments can be built that will support program development in a mathematically substantial way.

1.2.2 Parallelism

A considerable problem in parallel computation is finding a suitable means of expressing parallelism. Language constructs that explicitly initiate and terminate execution paths of disparate subcomputations are at best clumsy and may add substantial overhead. Far more desirable is a system where parallelism is naturally expressed, especially if the mode of expression is one that allows the programmer to ignore efficiency considerations if he chooses to do so. FP languages are particularly attractive in this respect. Parallelism is expressed by functional forms, and the forms can be chosen to reflect those kinds of parallelism that are available on a particular machine. Examples of these functional forms are:

apply-to-all: $(\alpha f: \langle x_1, x_2, \ldots x_n\rangle) \rightarrow \langle (f{:}x_1), (f{:}x_2), \ldots (f{:}x_n)\rangle$

construction: $([f_1, f_2, \ldots f_n]: x) \rightarrow \langle (f_1{:}x), (f_2{:}x) \ldots (f_n{:}x)\rangle$

parallel-apply: $(PA\langle f_1, f_2, \ldots f_n\rangle: \langle x_1, x_2, \ldots x_n\rangle) \rightarrow \langle (f_1{:}x_1),(f_2{:}x_2), \ldots (f_n{:}x_n)\rangle$

Note that the availability of 'flat lists' rather than requiring that lists be deeply nested increases the ease of dealing with these functions and their arguments.

A programmer should be able to use parallelism as a convenient tool in algorithm specification just as one uses recursion or iteration. Ideally, the resulting program could be executed unchanged, or, if appropriate, could be transformed into another with different degrees or forms of parallelism. This would require a facility that could both devise transformations and compare the costs of execution of the programs.

1.2.3 Reasoning About Programs and Program Transformation

As a class, functional languages appear to provide a better basis for reasoning about programs and richer opportunities for program transformation than do von Neumann languages. There is a growing consensus that the Floyd-Hoare approach to reasoning about von Neumann programs, while once considered quite promising, is too unwieldy to deal with programs of a substantial size and complexity written in a sufficiently rich language. Experiences with functional languages such as LCF indicate that these languages hold greater promise for increasing programmer productivity and for providing a facility for proving properties of importance about programs.

FP languages provide a particularly attractive basis for these activities. There is a rich algebra of programs in which the algebraic objects are functions (either variables, denoting arbitrary functions, or constants, denoting specific functions) and the algebraic operations are the functional forms of the language. Functional forms such as those chosen by Backus

[BA78] allow the programmer to think consistently about both function constructs and data objects. Thus, the primitive function 'append-to-the-left' is defined by the re-write rule:

$$(apndl:<x, < y_1, y_2, \ldots y_n>>) \rightarrow < x, y_1, y_2, \ldots y_n>$$

but it also obeys a similar algebraic law over function variables

$$apndl \circ [f, [g_1, g_2, \ldots g_n]] = [f, g_1, g_2, \ldots g_n],$$

where $\circ$ denotes function composition and [ ] denotes the functional form 'construction' as defined earlier in this section.

1.2.4 Problems

But problems remain for functional programming languages, including FP languages. An overriding concern is that of efficient execution; we believe the machine described in the next section will provide the answer to this problem for FP languages.

Following execution efficiency, the problem of greatest concern is that of handling 'history-sensitivity', or computations that necessarily involve a state. In his Turing Award Lecture [BA78], Backus proposed a tentative solution for FP languages which he called **applicative state transition** (AST) systems. An alternative approach is to extend FP languages to a domain that includes streams [IT83]. No satisfactory solution has been found so far, however, and the topic remains one of active research.

Another problem is that of programming style. FP is capable of expressing overwhelming one-liners in the manner of APL, but its definitional facility also makes it both easy and natural to avoid such problems by writing programs in a top-down manner. Nevertheless, programs written in FP are, like those of LISP, easier to write than to understand, and, with the exception of [BA81], the problem has not been addressed.

1.3 The FFP Language

The FFP language is similar to FP languages except that

a. its syntax is more regular, and

b. it incorporates higher-level functions (that is, functions are first-class objects).

For ease of presentation, we have described FP rather than FFP languages. An extension of FP languages that incorporates higher-level functions has been proposed [ABP82]; this language, called FP1.5, is as 'user-friendly' as FP languages. The FFP language uses a more regular syntax and is more difficult for a programmer to work with, but a user's FP or FP1.5 program can be translated into an FFP program by a straightforward process, and the resulting program executed directly on the FFP machine, which is described in the following section.

2. THE FFP MACHINE

One way of classifying multiprocessors is by the granularity of the individual processors. Each processor of a large-grain multiprocessor is capable of executing a substantial task independently, that is, each processor is a full-fledged computer, with a powerful CPU and a sizeable local memory. In contrast, each processing element of a small-grain multiprocessor is quite simple, consisting of a small (possibly bit-serial) ALU and a few dozen registers. In a small-grain multiprocessor, the performance of all but the most trivial task requires the cooperation of a number of separate processing elements

Although few small-grain multiprocessor designs have been proposed, they appear to have several potential advantages, as illustrated by the FFP machine:

a. They can exploit VLSI well. This requires that the multiprocessor be constructed of many copies of a few processor types, and that the processors be arranged in a regular topology. The FFP machine comprises many copies of processors of only two types, the L-cells and the T-cells, arranged as a full binary tree.

b. They can exploit parallelism in a wide variety of forms. The FFP can execute as many programs in parallel as will fit in its memory. Within each program, it can execute an arbitrary number of reducible applications (RAs) simultaneously. And within a single RA, at a sub-language level, it can simultaneously perform a number of operations within distinct cells, such as summing pairs of numbers or copying several expressions.

c. They can dynamically fit the hardware to the requirements of the program during program execution. The FFP machine automatically reallocates its resources during program execution by dividing the machine into submachines of appropriate sizes. When a subtask becomes executable, a submachine of the proper size is formed and allocated to the subtask.

The last point bears some elaboration. A large-grain system must break a problem up so that each subcomputation fits within a processor; thus, the problem must somehow be fit to the rigid boundaries imposed by the hardware. This *program decomposition problem* can be difficult even for fairly regular and straightforward computations. Even within a given class of problems, a desirable decomposition may depend on the size of the specific problem instance. For highly dynamic problems it will be impossible to perform the decomposition prior to the computation, and the overhead of doing it dynamically will be prohibitive.

2.1 Construction

The FFP machine is a small-grain multiprocessor consisting of a full binary tree of processors (cells) with connections between adjacent leaf cells. The cells are of two basic types. The leaf cells (or L-cells) contain the program text, which is re-written dynamically among the leaves during execution as dictated by the reduction semantics. The non-leaf tree cells (or T-cells) serve as communication and computation nodes. All the L-cells are identical, and all the T-cells are identical except those at one particular level that serve as I/O ports.

This network is a single computer that can be divided into a number of submachines, where each submachine consists of a contiguous segment of L-cells and a binary tree (not

necessarily full) of T-cells above it. Because of the regularity of the interconnections, such a network is arbitrarily expansible. Moreover, processing power is evenly distributed over the tree, and the processing power of a network is proportional to its size. Depending on the way the machine is divided, a particular T-cell may participate in up to three distinct computations, but never more. It's role in each computation is limited, so the computational requirements of the T-cells are quite modest. Both L- and T-cells are small; it is expected they will contain about 10 thousand transistors.

The *execution cycle* of the machine consists of three steps:

a. In the *partitioning* phase, the machine is divided into submachines and appropriate information is brought into the L-cells. The division into submachines is done by locating (on the basis of syntax) every reducible application (RA) that resides in the array of L-cells. The L-cells that contain an RA and a tree of T-cells above them are allocated to the subcomputation of re-writing the RA. The RA consists of a function and an argument; the information that must be brought into the machine to re-write the RA consists of the microprograms for the functions being executed. Each L-cell of the subtree that contains an RA receives a segment of a microprogram. The particular microprogram segment received by an L-cell is determined by (i) the function being applied in the RA, and (ii) the syntactic position in the RA of the symbol contained in that L-cell.

b. In the *execution* phase, the L-cells execute their microprograms. This may include sending information into and receiving information from the T-network. Information packets flowing through the T-network are processed according to the fixed program of the T-cells. Execution within an RA may continue until the computation is finished, or until it cannot proceed further because more resources (in particular, L-cells) are needed, or until the computation is suspended by the initiation of storage management.

c. In the *storage management* phase, additional space is provided where needed to re-write the current RAs. This is done by shifting symbols in the L-array to the left and right to provide empty L-cells in the required positions. Storage management is the only resource allocation done by the machine.

Note that when execution of an RA is suspended, the state of the partially-completed computation is recorded in the L-cells. These L-cells then participate in storage management, partitioning of the machine takes place, and the computation resumes. Thus a computation that spans several machine cycles may be moved from one place to another in the machine, each time making use of a different submachine.

Each FFP function can be put in one of three classes according to the kind of support that is necessary for the evaluation of an RA in which the function appears as operator:

a. In the simplest case, the various L-cells of the RA need not communicate with each other, and the RA can be re-written without additional L-cells. Examples include the *selector* functions that select one element from a sequence operand, the *tail* function, and such functions as *apndl* and *apndr*. All RAs with function operators from this class can be re-written in a single machine cycle.

b. Other functions require that information be communicated among the L-cells of an RA, but require no additional space. (Except for storage management, all communication among L-cells takes place through the T-network.) Examples of this class include summing the entries of a sequence of numbers, computing the dot product of two vectors, sorting the elements of an array of atoms, and computing the transpose of a matrix of atoms. In some cases, such as the addition of the entries of a sequence of numbers, information is combined as it is sent up into the T-network and these operations can be done in a single machine cycle regardless of the operand size. Other operations of this class, such as matrix transpose, require the information to remain distinct; the number of machine cycles required for these operations is a function of the operand size.

c. Finally, some functions require additional space, either to hold intermediate computations or the final result of the re-writing. Examples include the function ***double***, which changes an operand **x** into the pair **<x,x>**, the ***distribute-from-the-left*** function:

$$(distl : \langle x \langle y_1, y_2, \ldots y_n \rangle\rangle) \rightarrow \langle\langle x,y_1\rangle, \langle x,y_2\rangle, \ldots \langle x,y_n\rangle\rangle$$

and many others. For these functions, the first machine cycle is devoted to determining how much storage is needed, and where. Succeeding machine cycles accomplish the actual re-writing of the RA.

3. PROGRAMMING THE FFP MACHINE

Programming well implies both a good programming style and a good use of computing resources. I will not address the matter of a good style of functional programming, but I would like to say a few words about using resources well.

It goes without saying that we need a machine model if we are to judge that a program uses resources well, otherwise there is precious little we can say. Furthermore, if a programmer is to use resources well, he must have a conceptual model of the machine that coincides fairly closely to the actual machine, and the language constructs should be easily interpreted in terms of the conceptual model.

The programmer contemplating FFP program execution thinks quite naturally of a process that iteratively finds all the RAs in a current program expression and simultaneously re-writes them, thus producing a new program expression. Execution of a program on the FFP machine follows this model to some extent; the program expression resides in the L-array and RAs in the program expression are re-written as they are created. The model is accurate if one takes into account that the time taken to re-write an RA can vary greatly. The actual time to re-write an RA can be affected both by the operator and the operand.

The FFP machine is unusual among parallel processors in that sufficiently regular programs can be traced and their time and space requirements predicted. Analysis is not possible for all programs; just as with von Neumann machines, unpredictable programs must be run or simulated. But matrix multiplication [MSK], associative search [SW81] and partial differential equations [MP82] are some of the applications areas for which algorithms have been proposed and detailed predictions made for machine performance.

3.1 Efficiency

The work we have done in algorithm design and analysis indicate that this machine can be programmed at a variety of levels of sophistication. If one chooses, one can write straightforward programs, using parallelism if it is convenient. In this category I would place the first matrix multiplication algorithm given in Backus's Turing Award lecture; the algorithm is brief, easily understood and highly parallel. But careful analysis shows that less highly parallel algorithms perform better on the FFP machine. Backus's (first) algorithm does all the pairwise multiplication of matrix entries at the same time. In order to do so, it requires $\Theta(n^3)$ space for two $n \times n$ matrices, and hence, because of storage management, $\Theta(n^3)$ time. A more modest algorithm (but somewhat more complex) can be written that performs the matrix multiplication in n steps, each step using a row of the first matrix and the entire second matrix to obtain a row of the result matrix. This algorithm, quite accessible to a careful programmer and requiring no extraordinary functions in the FP language, gives an $\Theta(n^2)$ algorithm in both time and space.

But the potential rewards of careful programming of the FFP machine don't stop there. Because the microcode for each operation is not part of the machine design, the ambitious programmer can devise his own FP operations by writing the microcode that will allow the machine to execute the operations as machine primitives. A number of specialized primitive operations can be devised that aid in matrix multiplication. In fact, the multiplication of square matrices of atoms can be made a primitive operation, although one that still requires $\Theta(n^2)$ time and space. Using a primitive operation for matrix multiplication may not in fact be the best thing to do, since complex primitives usually have large segments of microcode that must be sent to the L-cells and then moved during storage management, slowing down machine operation. My intention here is not to give any final answers ... we don't know them yet ... but to point out that the FFP machine can be programmed at a variety of levels, with increasing rewards (of lowered execution cost) at each level.

3.2 Program Optimization

Much work has been done on optimization of functional programs [DA82] and some on FP programs in particular [WA82], but there has been no work so far on this problem for the FFP machine. I think the area is quite promising, and I'll give two simple illustrative examples.

Automatic program optimization refers to changing code to increase the execution speed or lower the space required. Contemporary optimizers use techniques, largely *ad hoc*, that affect the program only locally, e.g., by detecting common subexpressions or by removing unnecessary assignments from loops. Automatic optimization for FP programs using the algebra of programs has much greater potential.

Algebraic laws could simply be used to replace code in programs and subprograms with more efficient descriptions of the same functions. But, the same technique allows optimization over subroutine boundaries. For example, if a program definition has a subexpression of the form

$$f \circ g$$

where f and g are defined as

$$f = h \circ [1,3]$$
$$g = [3,4,6] \circ r$$

then we can replace $f \circ g$ by $h \circ [3,6] \circ r$, thus reducing copying costs.

A more sophisticated optimization could be done by describing how costs are affected by operand size and type so that the optimizer could choose the least expensive program expression for a particular situation. Thus, some matrix primitive machine operations require the operand matrix have only atoms as entries; these are the operations of choice if they can be used. Similarly, applying the law

$$[f_1, f_2, \ \ldots \ , f_n] \circ g = [f_1 \circ g, f_2 \circ g, \ \ldots \ , f_n \circ g]$$

gives two descriptions of the same function; the left description is efficient if the size of the value of g is small compared to the its argument, but if the reverse is true, this may not be the case. In particular, if the storage required can be reduced by optimizing each of the $f_i \circ g$ independently, then the right side may be less costly. A sufficiently sophisticated optimizer could choose correctly between these forms at compile time if adequate information was available. Even when compile-time optimization is not possible, the information gained about an RA operand during partitioning may make execution time optimization feasible for some programs.

Summary

The FFP machine is a small-grain highly parallel multiprocessor that directly executes a functional language. A suitably designed programming environment for the language is feasible, and could provide the basis for developing programs quickly as well as for transforming them into more efficient programs. Both manual optimization techniques, such as devising new machine primitives, and automatic techniques based on the algebra of programs, are available. This paper described some of the important features of the language and the machine and how they may influence programming.

Bibliography

[AD79] ACKERMAN, W.B. and DENNIS, J.B. VAL—A value-oriented algorithmic language: preliminary reference manual. MIT Laboratory of Computer Science Technical Report TR-218, Cambridge, Massachusetts, June 1979.

[ABP82] ARVIND, BROCK, J.D., and PINGALI, K.K. FP1.5: Backus' FP with higher order functions. Massachusetts Institute of Technology Laboratory for Computer Science, Computation Structures Group Note 46, March 1982.

[BA78] BACKUS, J. Can programming be liberated from the von Neuman style? A functional style and its algebra of programs. *Communications of the ACM* **21**, 8 (1978), 613–641.

[BA81] BACKUS, J. Function level programs as mathematical objects. *Proceedings of the 1981 Confernece on Functional Programming Languages and Computer Architecture* (Oct. 18–22, 1981, Portsmouth, New Hampshire), pp. 1–10.

[BU78] BURGE, W.H. ISWIM—A mixture of APL and LISP. IBM Research report RC 6967, Yorktown Heights, New York (January 1978).

[DA83] DANFORTH, S. DOT, a distributed operating system model of a tree-structured multiprocessor. *Proceedings of the 1983 International Conference on Parallel Processing*, pp. 194–201.

[DA82] DARLINGTON, J. Program Transformation. *Functional Programming and Its Applications*. Eds. J. Darlington, P. Henderson, and D.A. Turner, Cambridge: Cambridge University Press, 1982.

[FSS84] FRANK, G.A., SIDDALL, W.E., and STANAT, D.F. Virtual Memory Schemes for an FFP Machine. *International Workshop on High-Level Computer Architecture 84* (Los Angeles, California, May 23–25, 1984).

[FR81] FRANK, G.A. Specification of data structures for FP programs. *Proceedings of the 1981 Conference on Functional Programming Languages and Computer Architecture* (Oct. 18–22, 1981, Portsmouth, New Hampshire), pp. 221–228.

[GMW79] GORDON, M.G., MILNER, A.J. and WADSWORTH, C.P. Edinburgh LCF. Lecture Notes in Computer Science 78, Springer Verlag, Berlin (1979).

[GHW81] GUTTAG, J., HORNING, J., and WILLIAMS, J. FP with data abstraction and strong typing. *Proceedings of the 1981 Conference on Functional Programming Languages and Computer Architecture* (Oct. 18–22, 1981, Portsmouth, New Hampshire), pp. 11–24.

[IT83] IDA, T., TANAKA, J. Functional programming with streams. *Information Processing*. North-Holland, 1983. pp. 265–270.

[LA79] LANDIN, P.J. The next 700 programming languages. *Communications of the ACM 22*, 7 (1979), 424–436.

[MA79] MAGÓ, G.A. A network of microprocessors to execute reduction languages. Two parts. *International Journal of Computer and Information Sciences* **8**, 5, (1979), 349–385, **8**, 6 (1979), 435–471.

[MA81] MAGÓ, G.A. Copying operands versus copying results: a solution to the problem of large operands in FFP's. *Proceedings of the 1981 ACM Conference on Functional Programming Languages and Computer Architecture* (Oct. 18–22, 1981, Portsmouth, New Hampshire), pp. 93–97.

[MP82] MAGÓ, G.A. and PARGAS, R.P. Solving partial differential equations on a cellular tree machine. *Proceedings of the 10th IMACS World Congress* (Montreal, Aug. 8–13, 1982), vol. 1, pp. 368–373.

[MA82] MAGÓ, G.A. Data sharing in an FFP machine. *Conference Record of the 1982 ACM Symposium on LISP and Functional Programming* (Pittsburgh, Aug. 15–18, 1982), pp. 201–207.

[MM84] MAGÓ, G.A. and MIDDLETON, D. The FFP Machine—A Progress Report. *International Workshop on High-Level Computer Architecture 84* (Los Angeles, California, May 23–25, 1984).

[MSK] MAGÓ, G.A., STANAT, D.F. and KOSTER, A. Program execution in a cellular computer: some matrix algorithms (in preparation—draft available from authors).

[MC60] McCARTHY, J. Recursive functions of symbolic expressions and their computation by machine—I. *Comm. ACM* 3,4 (April 1960), 184–195.

[SM84] SMITH, B. Logic programming on an FFP machine. *Proceedings of the 1984 International Symposium on Logic Programming* (Febr. 6–9, 1984, Atlantic City, New Jersey), pp. 177–186.

[SW81] STANAT, D.F. and WILLIAMS, E.H. Jr. Optimal associative searching on a cellular computer. *Proceedings of the 1981 ACM Conference Functional Programming Languages and Computer Architecture* (Oct. 18–22, 1981, Portsmouth, New Hampshire), pp. 99–106.

[TS81] TOLLE, D.M. and SIDDALL, W.E. On the complexity of vector computations in binary tree machines. *Information Processing Letters* **13**, 3 (1981), 120–124.

[TU83] TURNER, D.A. Functional programming and proofs of program correctness. *Tools and Notions for Program Construction: An Advanced Course*. Eds. J. Darlington, P. Henderson, and D.A. Turner. Cambridge: Cambridge University Press, 1983.

[TU81] TURNER, D.A. The semantic elegance of applicative languages. *Proceedings of the 1981 ACM Conference on Functional Programming Languages and Computer Architecture*, pp. 85–92.

[WA82] WADLER, P. Applicative style programming, program transformation, and list operators. *Proceedings of the 1981 Conference on Functional Programming Languages and Computer Architecture* (Oct. 18–22, 1981, Portsmouth, New Hampshire), pp. 25–32.

Ramanujan and SCRATCHPAD

by

George E. Andrews

ABSTRACT

Recent mathematical work using SCRATCHPAD is discussed. This work is contrasted with the research procedures of S. Ramanujan (1887-1920).

Ramanujan and SCRATCHPAD

by

George E. Andrews[1]

1. Introduction. During the past two years SCRATCHPAD[14], [15] has been fully implemented at the Pennsylvania State University on a field test for IBM. Also during this time I have continued work on Ramanujan's "Lost" Notebook [5], [6]. These two projects are intertwined in very important ways that I hope to elucidate here.

In Section 2 we provide a sample of results for which SCRATCHPAD was instrumental in the discovery. Some of these discoveries are closely allied with Ramanujan's "Lost" Notebook, and this naturally leads us to our discussion of Ramanujan's type of research and discovery which we discuss in Section 3. In Section 4 we provide a short discussion of some of the many important results that have stemmed from Ramanujan's incredible empirical methods. In Section 5 we choose another of Ramanujan's problems and illustrate further aspects of SCRATCHPAD in handling such problems.

2. Achievements with SCRATCHPAD at Penn State. Let us begin with a solution of the Lusztig-Macdonald-Wall conjectures. This is a prototype of SCRATCHPAD at its best in doing problems of the type Ramanujan would have considered.

---

(1) Partially supported by NSF grant MCS8201733 and a fellowship from the Guggenheim Foundation.

In considering the conjugacy classes of the orthogonal and symplectic groups over finite fields of characteristic 2, each of Lusztig, Macdonald and Wall was led to consider the following conjecture [4]: Let $X_{-1} = X_{-1}(a,b,q) = a, X_0 = X_0(a,b,q) = b$, and for positive subscripts

$$X_{2n+1} = X_{2n} + q^{2n+1} X_{2n-1}, \tag{2.1}$$

$$X_{2n+2} = X_{2n+1} + q^{n+1}(1+q^{n+1})(X_{2n} + X_{2n-1}). \tag{2.2}$$

Then numerical evidence and significant possible theoretical consequences made the following likely

$$\lim_{n\to\infty} X_n(1,1,q) = \frac{\sum_{n=-\infty}^{\infty} q^{n^2}}{\prod_{n=1}^{\infty}(1-q^n)}, \tag{2.3}$$

and

$$\lim_{n\to\infty} X_n(0,1,q) = \frac{\sum_{n=0}^{\infty} q^{n^2+n}}{\prod_{n=1}^{\infty}(1-q^n)}. \tag{2.4}$$

My first proof of these conjectures [4] was a very "heavy" exercise that made extensive studies of the limiting functions of (2.3) and (2.4). However a "nice and natural" treatment is afforded by the following exercise in SCRATCHPAD:

First it is a simple matter to define these polynomials in SCRATCHPAD:

```
.X<-1>=1
```

$$X_{-1} = 1$$

```
.X<0>=1
```

$$X_0 = 1$$

```
.X<2*N+1>=X<2*N>+Q**(2*N+1)*X<2*N-1>, N IN (0,1,...)
```

$$X_a = X_{2x} + Q^{2x + 1} X_{2x - 1} \quad \text{WHEN } a=2x + 1 \ \& \ x \text{ IN } (0,1,\ldots)$$

```
.X<2*N+2>=X<2*N+1>+Q**(N+1)*(1+Q**(N+1))*(X<2*N>+X<2*N-1>), N IN (0,1,...)
```

$$X_a = X_{2x + 1} + Q^{x + 1} (1 + Q^{x + 1})(X_{2x} + X_{2x - 1}) \quad \text{WHEN } a=2x + 2 \ \& \ x \text{ IN } (0,1,\ldots)$$

Next we would like to stack these up against the well-known Gaussian polynomials or q-binomial coefficients [3; Ch. 3]. Our motivation lies in the fact that the Gaussian polynomials form the building blocks for all of the results in this area of mathematics [3; Ch. 3]. If we can represent the X_n in terms of the Gaussian polynomials GP(n,m) we suspect will be well on the road to knocking off the L-M-W conjectures.

```
.H<0>(X)=1
```

$$H_0(x) = 1$$

```
.H<N>(X)=H<N-1>(X*Q)+X*H<N-1>(X), N IN (1,2...,)
```

$$H_n(x) = H_{n-1}(x*Q) + xH_{n-1}(x) \text{ WHEN } n \text{ IN } (1,2...)$$

```
.GP(N,M)=COEFF(X,M,H<N>(X)), N IN (0,1,...)
```

$$GP(n,m) = COEFF(X,m,H_n(X)) \text{ WHEN } n \text{ IN } (0,1,...)$$

The following SCRATCHPAD output clearly exhibits certain important facts about $X_n(1,1,q)$:

```
.X<N> FOR N IN (0,1,...,10)
```

$$X_0 : 1$$

$$X_1 : Q + 1$$

$$X_2 : Q^2 + 3Q + 1$$

$$X_3 : Q^4 + Q^3 + 2Q^2 + 3Q + 1$$

$$X_4 : 2Q^6 + 4Q^5 + 5Q^4 + 5Q^3 + 4Q^2 + 3Q + 1$$

$$X_5 : Q^9 + Q^8 + 2Q^7 + 5Q^6 + 5Q^5 + 5Q^4 + 5Q^3 + 4Q^2 + 3Q + 1$$

$$X_6 : 2Q^{12} + 4Q^{11} + 6Q^{10} + 9Q^{9} + 11Q^{8} + 14Q^{7} + 13Q^{6} + 11Q^{5} + 11Q^{4} + 7Q^{3} + 4Q^{2} + 3Q + 1$$

$$X_7 : Q^{16} + Q^{15} + 2Q^{14} + 5Q^{13} + 7Q^{12} + 9Q^{11} + 11Q^{10} + 13Q^{9} + 14Q^{8} + 15Q^{7} + 13Q^{6} + 11Q^{5} + 11Q^{4} + 7Q^{3} + 4Q^{2} + 3Q + 1$$

$$X_8 : 2Q^{20} + 4Q^{19} + 8Q^{18} + 10Q^{17} + 15Q^{16} + 21Q^{15} + 26Q^{14} + 31Q^{13} + 35Q^{12} + 37Q^{11} + 37Q^{10} + 35Q^{9} + 32Q^{8} + 27Q^{7} + 21Q^{6} + 17Q^{5} + 13Q^{4} + 7Q^{3} + 4Q^{2} + 3Q + 1$$

$$X_9 : Q^{25} + Q^{24} + 2Q^{23} + 5Q^{22} + 7Q^{21} + 11Q^{20} + 15Q^{19} + 19Q^{18} + 24Q^{17} + 30Q^{16} + 34Q^{15} + 37Q^{14} + 42Q^{13} + 42Q^{12} + 41Q^{11} + 40Q^{10} + 36Q^{9} + 32Q^{8} + 27Q^{7} + 21Q^{6} + 17Q^{5} + 13Q^{4} + 7Q^{3} + 4Q^{2} + 3Q + 1$$

```
X
 10
:
     30      29      28       27       26       25       24       23
  2Q   + 4Q   + 6Q   + 10Q   + 16Q   + 25Q   + 33Q   + 44Q
  +
     22       21       20       19       18       17       16
  57Q   + 69Q   + 81Q   + 91Q   + 101Q   + 108Q   + 110Q
  +
     15       14       13       12       11       10       9
  110Q   + 109Q   + 102Q   + 92Q   + 81Q   + 70Q   + 60Q
  +
     8       7       6       5       4      3      2
  46Q   + 35Q   + 27Q   + 19Q   + 13Q   + 7Q   + 4Q   + 3Q + 1
```

In particular, we note that the degree of $X_{2n-1}(1,1,q)$ is n^2. Now GP(2n,n) is also known to be of degree n^2. Thus we might look at:

```
.X<2*N-1>-GP(2*N,N) FOR N IN (0,1,2,3,4,5)

        0

        0

        2Q

          6      5      4      3      2
        2Q   + 2Q   + 2Q   + 2Q   + 2Q   + 2Q


          13      12      11      10      9      8      7      6      5      4
        2Q    + 2Q    + 4Q    + 4Q    + 6Q   + 6Q   + 8Q   + 6Q   + 6Q   + 6Q
      +
          3      2
        4Q   + 2Q   + 2Q
```

```
    22      21      20      19      18      17      16      15
  2Q    + 2Q    + 4Q    + 6Q    + 8Q    + 10Q   + 14Q   + 16Q
+
    14      13      12      11      10      9       8       7
  18Q   + 22Q   + 22Q   + 22Q   + 22Q   + 20Q   + 18Q   + 16Q
+
    6       5       4       3       2
  12Q   + 10Q   + 8Q    + 4Q    + 2Q    + 2Q
```

This last sequence of polynomials, after we factor out $2q$, is of degree $n^2 - 4$ as is GP(2n,n-2). Thus we are naturally led to examine

```
.X<2*N-1>-GP(2*N,N)-2*Q*GP(2*N,N-2) FOR N IN (2,3,4,5)

     0

     0

       4
     2Q

       13      12      11      10      9       8       7       6       5       4
     2Q    + 2Q    + 2Q    + 2Q    + 2Q    + 2Q    + 2Q    + 2Q    + 2Q    + 2Q
```

This last sequence of polynomials, after we factor out $2q^4$, is of degree $n^2 - 16$ as is GP(2n,n-4). Thus we are naturally led to examine

```
.GP<2*N-1>-GP(2*N,N)-2*Q*GP(2*N,N-2)-2*Q**4*GP(2*N,N-4) FOR N IN (4,5)

     0

     0
```

It requires no great leap of a Ramanujan's imagination to guess that

$$X_{2n}(1,1,q) = GP(2n,n,q) + 2 \sum_{1\le j\le n/2} q^{j^2} GP(2n,n-2j,q). \tag{2.5}$$

Similar formulas for $X_{2n}(1,1,q)$, $X_{2n}(0,1,q)$ and $X_{2n+1}(0,1,q)$ are to be found in exactly the same way [7]. From here the battle with the L-M-W conjectures is 90% won. Standard techniques allow one to establish (2.5), and a simple limit argument leads from (2.5) to (2.3). All the details of this and many other results are given in [7].

Again let me emphasize: the above example is SCRATCHPAD operating at its best. Ramanujan might have approached the L-M-W conjectures in this way; however, even his calculating skill would not have allowed him the ease, accuracy and speed of the above procedure.

Let us now turn to a problem of great interest to Ramanujan himself. It is, in fact, a problem which either he did not solve or he did not record in such a way that anyone is aware of his solution. A few months before he died in 1920, Ramanujan wrote to G. H. Hardy that he had extended the classical theta functions to a larger class which he called "mock theta functions" [27]. Among them he listed a set of "third order mock theta functions", e.g.

$$f(q) = 1 + \sum_{n=1}^{\infty} \frac{q^{n^2}}{(1+q)^2(1+q^2)^2\cdots(1+q^n)^2}. \tag{2.6}$$

G. N. Watson [27] subsequently provided the key to the study of $f(q)$; namely, he proved

$$(2.7) \qquad f(q) \prod_{n=1}^{\infty} (1-q^n) = 1 + 4 \sum_{n=1}^{\infty} q^{n^2} \alpha_n,$$

where

$$(2.8) \qquad \alpha_n = \frac{(-1)^n q^{(n^2+n)/2}}{1 + q^n} .$$

Watson was unaware that Ramanujan had recorded (2.7) in the "Lost" Notebook [5]. Ramanujan also described "fifth order mock theta functions," e.g.

$$(2.9) \qquad f_o(q) = 1 + \sum_{n=1}^{\infty} \frac{q^{n^2}}{(1+q)(1+q^2)\ldots(1+q^n)} .$$

However an analog of (2.7) for $f_o(q)$ appears nowhere in Ramanujan's "Lost" Notebook, and Watson readily admits his inability to find such a formula [27]. Now the classical theory of q-hypergeometric series shows us that if

$$(2.10) \quad A_n = (-1)^n(1+q^n)q^{n(n-1)/2}B_0 + (1-q^{2n}) \sum_{j=1}^{n} (1-q^{n-j+1})(1-q^{n-j+2})\ldots(1-q^{n+j-1})(-1)^{n-j}q^{\binom{n-j}{2}} B_j, \quad (n>0),$$

$$(2.11) \quad A_0 = B_0,$$

then

$$(2.12) \qquad \sum_{n=0}^{\infty} q^{n^2} B_n = \frac{1}{\prod_{n=1}^{\infty} (1-q^n)} \sum_{n=0}^{\infty} q^{n^2} A_n .$$

If we define A_n in terms of B_n by (2.10) and (2.11) using SCRATCHPAD, then for

(2.13) $$B_n = \frac{1}{(1+q)^2(1+q^2)^2\ldots(1+q^n)^2} ,$$

```
.A<N> FOR N IN (0,1,2,3,4,5)
```

(4) $A_0 : 1$

(5) $A_1 : \dfrac{-4Q}{Q+1}$

(6) $A_2 : \dfrac{4Q^3}{Q^2+1}$

(7) $A_3 : \dfrac{-4Q^6}{Q^3+1}$

(8) $A_4 : \dfrac{-4Q^{10}}{-Q^4-1}$

(9) $A_5 : \dfrac{-4Q^{15}}{Q^5+1}$

We can easily guess from the first few examples that (2.8) must be true. Let us now move to (2.9).

```
.B<N>=PROD<J=1;N>(1/(1+Q**J)), N IN (1,2,...)
```

$$B_n = \prod_{J=1}^{n} \frac{1}{1+Q^J} \quad \text{WHEN } n \text{ IN } (1,2,\ldots)$$

```
.A<N> FOR N IN (0,1,2,3,4)
```

$$A_0 : 1$$

$$A_1 : Q^2 - 3Q$$

$$A_2 : Q^7 - 2Q^6 - Q^5 + 2Q^4 + 2Q^3$$

$$A_3 : Q^{15} - 2Q^{14} - Q^{12} + 4Q^{11} - 2Q^8 - 2Q^6$$

$$A_4 : Q^{26} - 2Q^{25} + Q^{22} + 2Q^{21} - 2Q^{18} - 2Q^{17} + 2Q^{13} + 2Q^{10}$$

```
.A<N> FOR N IN (5,6,7,8,9,10)
```

$$A_5$$

$$Q^{40} - 2Q^{39} + 2Q^{36} - Q^{35} + 2Q^{34} - 4Q^{31} + 2Q^{26} + 2Q^{24}$$

$$+ \; - 2Q^{19} - 2Q^{15}$$

A
6
:

$$Q^{57} - 2Q^{58} + 2Q^{53} - Q^{51} + 2Q^{50} - 2Q^{48} - 2Q^{47} + 2Q^{42}$$
$$+ 2Q^{41} - 2Q^{35} - 2Q^{32} + 2Q^{26} + 2Q^{21}$$

A
7
:

$$Q^{77} - 2Q^{76} + 2Q^{73} - Q^{70} + 2Q^{69} - 2Q^{68} - 2Q^{66} + 4Q^{61}$$
$$+ - 2Q^{54} - 2Q^{52} + 2Q^{45} + 2Q^{41} - 2Q^{34} - 2Q^{28}$$

A
8
:

$$Q^{100} - 2Q^{99} + 2Q^{96} - Q^{92} - 2Q^{88} + 2Q^{84} + 2Q^{83} - 2Q^{76}$$
$$+ - 2Q^{75} + 2Q^{67} + 2Q^{64} - 2Q^{56} - 2Q^{51} + 2Q^{43} + 2Q^{36}$$

A
9
:

$$Q^{126} - 2Q^{125} + 2Q^{122} - 3Q^{117} + 2Q^{116} - 2Q^{113} + 2Q^{110}$$
$$+ 2Q^{108} - 4Q^{101} + 2Q^{92} + 2Q^{90} - 2Q^{81} - 2Q^{77} + 2Q^{68} + 2Q^{62}$$
$$+ - 2Q^{53} - 2Q^{45}$$

Surely an examination of the first few values of A_n reveals nearly nothing. However the sequence $A_0, A_1, \ldots, A_9$ reveals first that probably A_n is of degree $n(3n+1)/2$.

Furthermore a look at the first few terms of A_8 and A_9 suggests

$$A_n = q^{n(3n+1)/2}\,(1-2q^{-1}+2q^{-4}-2q^{-9}+\ldots)$$
$$-q^{n(3n-1)/2}\,(1-2q^{-1}+2q^{-4}-2q^{-9}+\ldots).$$

Once this is observed only a little tidying up leads us to

$$(2.14)\quad A_n = q^{n(3n+1)/2}\sum_{j=-n}^{n}(-1)^j q^{-j^2}$$
$$-q^{n(3n-1)/2}\sum_{j=-n+1}^{n-1}(-1)^j q^{-j^2}\quad .$$

results renimiscent of D. Shanks' results on tancated theta series [24] [25]. Once we see that (2.14) is highly probable, it only requires the application of the techniques presented in [8] and some classical results to prove (2.14). This then establishes that

$$(2.15)\quad f_o(q) = \frac{1}{\prod_{n=1}^{\infty}(1-q^n)}\sum_{\substack{j=-\infty\\ m\geq|j|}}^{\infty}(-1)^j\, q^{\frac{1}{2}m(5m+1)-j^2}\,(1-q^{4m+2}),$$

a result definitely missed by Watson [28] and quite probably missed by Ramanujan.

4. Ramanujan's Methods. The work described in Section 3 has led us to Ramanujan, the self-taught Indian genius. The romantic tale of his rise to fame and tragically short life has been chronicled by Hardy [16], Ranganathan [22], Newman [19] and others. We are most interested here in descriptions of his methods of discovery. We begin with a paragraph by G. H. Hardy [21; p. xxxv] on Ramanujan's abilities:

> "It was his insight into algebraical formulae, transformations of infinite series, and so forth, that was most amazing. On this side most certainly I have never met his equal, and I can compare him only with Euler or Jacobi. He worked, far more than the majority of modern mathematicians, by induction from numerical examples; all of his congruence properties of partitions, for example, were discovered in this way. But with his memory, his patience, and his power of calculation, he combined a power of generalisation, a feeling for form, and a capacity for rapid modification of his hypotheses, that were often really startling, and made him, in his own peculiar field, without a rival in his day."

Subsequently in discussing his previous comments, Hardy [16; p. 14] goes on to say:

> "I do not think now that this extremely strong language is extravagant. It is possible that the great days of formulae are finished, and that Ramanujan ought to have been born 100 years ago; but he was by far the greatest formalist of his time. There have been a good many more important, and I suppose one must say greater, mathematicians than Ramanujan during the last fifty years, but not one who could stand up to him on his own ground. Playing the game of which he knew the rules, he could give any mathematician in the world fifteen."

Fortunately for us, the scrappy fragments of Ramanujan's last tortured year in India (when he was dying) provide us some true images of the genius at work. In my article [5] on the "Lost" Notebook, I included two photostats from this remarkable document. The second page (which resembles perhaps 60%

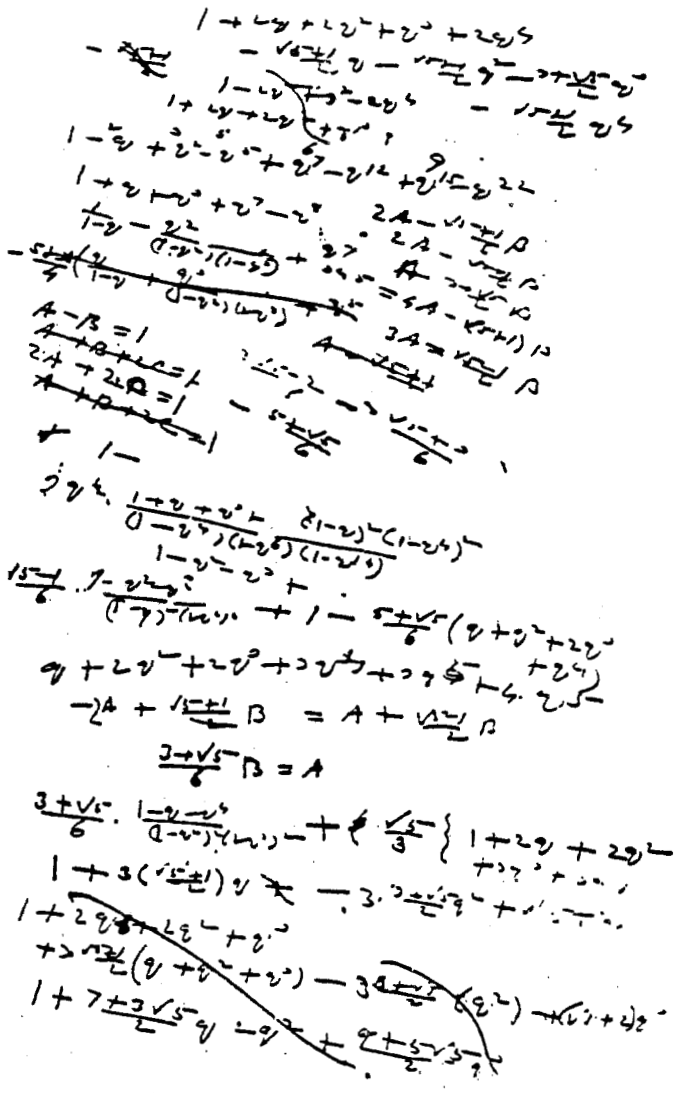


Fig. 1. Power series computations and comparisons from Ramanujan's "Lost" Notebook.

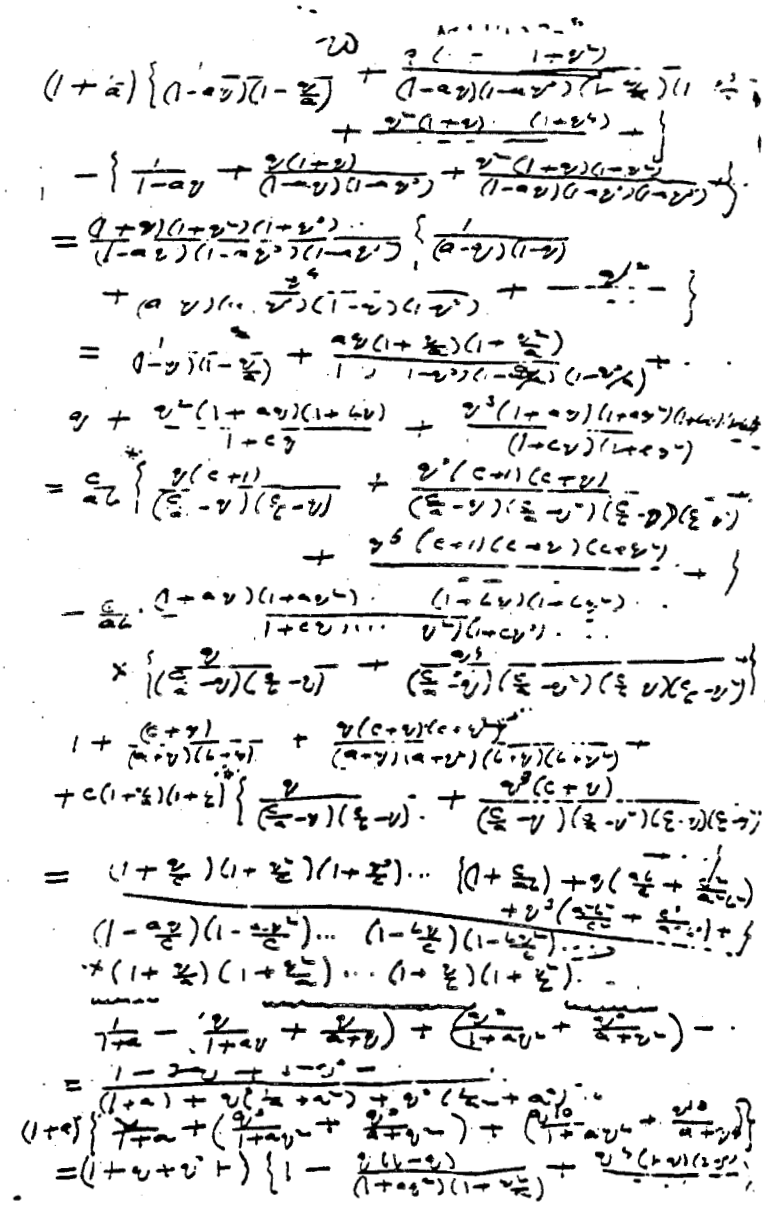


Fig. 2. A page of formulas from Ramanujan's "Lost" Notebook.

of the "Lost" Notebook presents finished formulas without proof. The first page shows numerous fragments of infinite series; surely this is Ramanujan as SCRATCHPAD. By the comparison, modification, and recomparison of these fragments, Ramanujan produced the numerous formulas making up this notebook. Occasionally he was led astray. For example, in one place he asserts

$$1 - x + x^3 - x^6 + \ldots = \cfrac{1}{1 + \cfrac{x + x^2}{1 + \cfrac{x^3 + x^4}{1 + \cfrac{x^5 + x^6}{1 + \ldots}}}} .$$

This formula (at least the one implied) is false [6; §4] although both sides agree as power series up through x^6.

4. The Significance of Ramanujan's Work. Before we examine further how SCRATCHPAD aids us in other problems considered by Ramanujan, we present a short account of the importance of some of Ramanujan's achievements.

While Ramanujan's life, mathematical education and mathematical methods are eccentric by modern day standards (indeed he forms the subject of a book-length psychoanalytic study [18]), nonetheless, his research has formed the background for many recent projects. We cite a few.

4.1. Ramanujan's introduction of the function $\tau(n)$ defined by

$$(4.1) \qquad \sum_{n=1}^{\infty} \tau(n) q^n = q \prod_{n=1}^{\infty} (1-q^n)^{24} ,$$

and his numerous conjectures about $\tau(n)$ (all discovered from the empirical

approach touched upon in Section 2) led to many important studies by eminent mathematicians such as E. Hecke and L. J. Mordell. His final conjecture on this topic:

$$|\tau(p)| \leq 2p^{11/2} \quad \text{for all primes } p \tag{4.2}$$

was proved by P. Deligne [12] in his work that won him a Fields Medal.

4.2. Ramanujan's empirical studies of divisibility properties of $p(n)$ defined by

$$1 + \sum_{n=1}^{\infty} p(n)q^n = \prod_{n=1}^{\infty} \frac{1}{1 - q^n}, \tag{4.3}$$

led to extensive work on modular forms. Finally in 1967 A. O. L. Atkin [9] settled the last of Ramanujan's conjectures in this area.

4.3. In the 1960's when huge numerical computations were becoming feasible on machines, D. Shanks and J. W. Wrench [26] utilized the most unlikely formula

$$\frac{1}{\pi} = \frac{1}{4}\left(\frac{1123}{882} - \frac{2253}{882^3}\cdot\frac{1}{2}\cdot\frac{1\cdot3}{4^2} + \frac{44043}{882^5}\cdot\frac{1\cdot3}{2\cdot4}\cdot\frac{1\cdot3\cdot5\cdot7}{4^2 8^2} \cdots\right. \tag{4.4}$$

for actually computing π to 100000 digits. This formula is one of those found by Ramanujan.

4.4. R. J. Baxter won the Boltzmann Medal in statistical mechanics in 1981 for his solution of the hard hexagon model [10; Ch. 14]. His solution relies in an absolutely critical way on the Rogers-Ramanujan identities [3; Ch. VII]:

$$(4.5)\quad 1 + \sum_{n=1}^{\infty} \frac{q^{n^2}}{(1-q)(1-q^2)\dots(1-q^n)} = \prod_{n=0}^{\infty} \frac{1}{(1-q^{5n+1})(1-q^{5n+4})},$$

$$(4.6)\quad 1 + \sum_{n=1}^{\infty} \frac{q^{n^2+n}}{(1-q)(1-q^2)\dots(1-q^n)} = \prod_{n=0}^{\infty} \frac{1}{(1-q^{5n+2})(1-q^{5n+3})}.$$

Again we have formulas arising from Ramanujan's incredible empirical work.

4.5. The Rogers-Ramanujan identities and their extensions have formed an extensive field of mathematical research [1], [2], [3; Ch. VII], [11]. This research has not only been of interest to classical analysts. Besides its surprising appearance in atomic physics alluded to above, it has also provided new questions (and some answers) in problems in algebraic-transcendental number theory [23] and bijective combinatorics [13].

There are a number of other areas of mathematics that have been heavily affected by Ramanujan's work. I chose the above examples because they highlight the discoveries Ramanujan made when acting as though he possessed a variation of SCRATCHPAD.

5. SCRATCHPAD and the Rogers-Ramanujan identities. As we demonstrated in Section 2, SCRATCHPAD can be used by humble followers of Ramanujan to solve problems that left even Ramanujan puzzled. However Section 2 showed SCRATCHPAD at its best in competition with Ramanujan. Some applications require more adroitness.

Let us take as a prototypical example the Rogers-Ramanujan identities themselves. For a time around 1915 they were viewed as a major unsolved problem posed by Ramanujan (actually the then little-known L. J. Rogers had already proved them). Here are the first two paragraphs of P. A. MacMahon's, Combinatory Analysis, Vol. II, Section VII, Chapter III, [17]:

"Mr. Ramanujan of Trinity College, Cambridge, has suggested a large number of formulae which have applications to the partition of numbers. Two of the most interesting of these concern partitions whose parts have a definite relation to the modulus five. Theorem 1 gives the relation

$$1 + \frac{x}{1-x} + \frac{x^4}{(1-x)(1-x^2)} + \frac{x^9}{(1-x)(1-x^2)(1-x^3)} + \ldots$$

$$+ \frac{x^{i^2}}{(1-x)(1-x^2)\ldots(1-x^i)} + \ldots$$

$$= \frac{1}{(1-x)(1-x^6)(1-x^{11})\ldots(1-x^{5m+1})\ldots}$$

$$\times \frac{1}{(1-x^4)(1-x^9)(1-x^{14})\ldots(1-x^{5m+4})\ldots} ,$$

where on the right-hand side the exponents of x are the numbers given by the congruences $\equiv 1 \bmod 5$, $\equiv 4 \bmod 5$.

This most remarkable theorem has been verified as far as the coefficient of x^{89} by actual expansion so that there is practically no reason to doubt its truth; but it has not yet been established."

Clearly MacMahon has checked this formula as far as x^{89} by hand.

Let us do this in SCRATCHPAD.

We must first get everything into polynomials. In particular, since

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \ldots, \tag{5.1}$$

we may truncate the geometric series and approximate

$$\frac{1}{(1-z)(1-zq)\ldots(1-zq^{n-1})}$$

by

```
.P<N>(Z,Q,L)=PROD<J=0;N-1>(SUM<K=0;FLOOR(L/J)>(Z**K*Q**(K*J)))
```

$$P_n(z,x,\ell) = \prod_{J=0}^{n-1} \sum_{K=0}^{\mathrm{FLOOR}(\ell/J)} z^K x^{K*J}$$

Thus the first 100 terms of the left side of (4.5) are given by

```
.ASYMP(Q) = 100
.SUM<N=0;9>(Q**(N**2)*P<N>(Q,Q,100-N**2))
```

$$\begin{aligned}
&68747Q^{99} + 63843Q^{98} + 59239Q^{97} + 54979Q^{96} + 50974Q^{95} \\
&+ 47276Q^{94} + 43802Q^{93} + 40594Q^{92} + 37582Q^{91} + 34806Q^{90} \\
&+ 32196Q^{89} + 29796Q^{88} + 27540Q^{87} + 25466Q^{86} + 23519Q^{85} \\
&+ 21732Q^{84} + 20050Q^{83} + 18512Q^{82} + 17066Q^{81} + 15742Q^{80} \\
&+ 14498Q^{79} + 13363Q^{78} + 12294Q^{77} + 11322Q^{76} + 10406Q^{75} + 9573Q^{74} \\
&+ 8790Q^{73} + 8080Q^{72} + 7409Q^{71} + 6804Q^{70} + 6233Q^{69} + 5717Q^{68} \\
&+ 5231Q^{67} + 4794Q^{66} + 4380Q^{65} + 4010Q^{64} + 3659Q^{63} + 3345Q^{62} \\
&+ 3049Q^{61} + 2785Q^{60} + 2533Q^{59} + 2311Q^{58} + 2100Q^{57} + 1913Q^{56} \\
&+ 1735Q^{55} + 1579Q^{54} + 1429Q^{53} + 1299Q^{52} + 1174Q^{51} + 1065Q^{50} \\
&+ 961Q^{49} + 871Q^{48} + 783Q^{47} + 709Q^{46} + 637Q^{45} + 575Q^{45} + 515Q^{43} \\
&+ 465Q^{42} + 415Q^{41} + 374Q^{40} + 333Q^{39} + 299Q^{38} + 266Q^{37} + 239Q^{36} \\
&+ 211Q^{35} + 189Q^{34} + 167Q^{33} + 149Q^{32} + 131Q^{31} + 117Q^{30} + 102Q^{29} \\
&+ 91Q^{28} + 79Q^{27} + 70Q^{26} + 61Q^{25} + 54Q^{24} + 46Q^{23} + 41Q^{22} + 35Q^{21} \\
&+
\end{aligned}$$

$$+ \; 31Q^{20} + 26Q^{19} + 23Q^{18} + 19Q^{17} + 17Q^{16} + 14Q^{15} + 12Q^{14} + 10Q^{13}$$
$$+ \; 9Q^{12} + 7Q^{11} + 6Q^{10} + 5Q^{9} + 4Q^{8} + 3Q^{7} + 3Q^{6} + 2Q^{5} + 2Q^{4} + Q^{3}$$
$$+ \; Q^{2} + Q + 1$$

The first 100 terms of the right side of (4.5) are then given by

```
.ASYMP(Q) = 100
.P<20>(Q,Q**5,100)*P<20>(Q**4,Q**5,100)
```

$$+ \; 68747Q^{99} + 63843Q^{98} + 59239Q^{97} + 54979Q^{96} + 50974Q^{95}$$
$$+ \; 47276Q^{94} + 43802Q^{93} + 40594Q^{92} + 37582Q^{91} + 34806Q^{90}$$
$$+ \; 32196Q^{89} + 29796Q^{88} + 27540Q^{87} + 25466Q^{86} + 23519Q^{85}$$
$$+ \; 21732Q^{84} + 20050Q^{83} + 18512Q^{82} + 17066Q^{81} + 15742Q^{80}$$
$$+ \; 14498Q^{79} + 13363Q^{78} + 122940^{77} + 11322Q^{76} + 10406Q^{75} + 9573Q^{74}$$
$$+ \; 8790Q^{73} + 8080Q^{72} + 7409Q^{71} + 6804Q^{70} + 6233Q^{69} + 5717Q^{68}$$
$$+ \; 5231Q^{67} + 4794Q^{66} + 4380Q^{65} + 4010Q^{64} + 3659Q^{63} + 3345Q^{62}$$
$$+ \; 3049Q^{61} + 2785Q^{60} + 2533Q^{59} + 2311Q^{58} + 2100Q^{57} + 1913Q^{56}$$
$$+ \; 1735Q^{55} + 1579Q^{54} + 1429Q^{53} + 1299Q^{52} + 11740^{51} + 1065Q^{50}$$
$$+ \; 961Q^{49} + 871Q^{49} + 783Q^{47} + 709Q^{46} + 637Q^{45} + 575Q^{44} + 515Q^{43}$$
$$+ \; 465Q^{42} + 415Q^{41} + 374Q^{40} + 333Q^{39} + 299Q^{38} + 266Q^{37} + 239Q^{36}$$
$$+ \; 211Q^{35} + 189Q^{34} + 167Q^{33} + 149Q^{32} + 131Q^{31} + 117Q^{30} + 102Q^{29}$$
$$+ \; 91Q^{28} + 79Q^{27} + 70Q^{26} + 61Q^{25} + 54Q^{24} + 46Q^{23} + 41Q^{22} + 35Q^{21}$$
$$+ \; 31Q^{20} + 26Q^{19} + 23Q^{18} + 19Q^{17} + 17Q^{16} + 14Q^{15} + 12Q^{14} + 10Q^{13}$$
$$+ \; 9Q^{12} + 7Q^{11} + 6Q^{10} + 5Q^{9} + 4Q^{8} + 3Q^{7} + 3Q^{6} + 2Q^{5} + 2Q^{4} + Q^{3}$$
$$+ \; Q^{2} + Q + 1$$

And as we expect the two results coincide.

6. Conclusion. The power of computer algebra in handling many mathematics problems has been amply demonstrated in many important articles, e.g. [20], [29]. We have tried here to show that much of the work of Ramanujan is facilitated through the use of a computer-algebra language like SCRATCHPAD. Indeed one feels at times in studying Ramanujan's work that he was a human version of such a language.

I close with the anecdotal recounting of a conversation between Ramanujan and an Indian acquaintance given in Ramanujan: The Man and the Mathematician [22; pp. 25-26]:

"Sandow: Ramanju, they all call you a genius.

Ramanujan: What! Me a genius! Look at my elbow, it will tell you a story.

Sandow: What is all this Ramanju? Why is it so rough and black?

Ramanujan: My elbow has become rough and black in making a genius of me! Night and day I do calculation on slate. It is too slow too look for a rag to wipe it out with. I wipe out the slate almost every few minutes with my elbow."

References

1. H. L. Alder, Partition identities - from Euler to the present, Amer. Math. Monthly, 76 (1969), 733-746.

2. G. E. Andrews, A general theory of identities of the Rogers-Ramanujan type. Bull. Amer. Math. Soc., 80 (1974), 1033-1052.

3. G. E. Andrews, The Theory of Partitions, Encyclopedia of Mathematics and Its Applications, Vol. II, Addison-Wesley, Reading, 1976.

4. G. E. Andrews, Partitions, q-series and the Lusztig-Macdonald-Wall conjectures, Invent. Math., 4 (1977), 91-102.

5. G. E. Andrews, An introduction to Ramanujan's "lost" notebook, Amer. Math. Monthly, 86 (1979), 89-108.

6. G. E. Andrews, Ramanujan's "lost" notebook: III. The Rogers-Ramanujan continued fraction, Advances in Math., 41 (1981), 186-208.

7. G. E. Andrews, On the Wall polynomials and the L-M-W conjectures, Australian J. Math., (to appear).

8. G. E. Andrews, Multiple series Rogers-Ramanujan type identities, Pacific J. Math., (to appear).

9. A. O. L. Atkin, Proof of a conjecture of Ramanujan, Glasgow Math. J., 8 (1967), 14-32.

10. R. J. Baxter, Exactly Solved Models in Statistical Mechanics, Academic Press, New York, 1982.

11. D. M. Bressoud, Analytic and combinatorial generalizations of the Rogers-Ramanujan identities, Memoirs of the Amer. Math. Soc., 24 (1980), No. 227.

12. P. Deligne, La Conjecture de Weil. I, I.H.E.S., 43 (1974), 273-307.

13. A. Garsia and S. Milne, A Rogers-Ramanujan bijection, J. Combinatorial Theory (A), 31 (1981). 289-339.

14. J. H. Griesmer, R. D. Jenks, D. Y. Y. Yun, SCRATCHPAD User's Manual, I.B.M. Thomas J. Watson Research Center, Yorktown Heights, N. Y., June, 1975.

15. J. H. Griesmer, R. D. Jenks, D. Y. Y. Yun, A set of SCRATCHPAD examples, IBM Thomas J. Watson Research Center, Yorktown Heights, N. Y., April, 1976.

16. G. H. Hardy, Ramanujan, Cambridge University Press, London and New York, 1940 (reprinted: Chelsea, New York).

17. P. A. MacMahon, Combinatory Analysis, Vol. II, Cambridge University Press, London and New York, 1916 (reprinted: Chelsea, New York, 1960).

18. A. Nandy, Alternative Sciences, Creativity and Authenticity in Two Indian Scientists, Allied Publishers, New Delhi, 1980.

19. J. R. Newman, The World of Mathematics, Vol. I., Simon and Schuster, New York, 1956.

20. R. Pavelle, M. Rothstein and J. Fitch, Computer algebra, Scientific American, 245 (1981), No. 6, 136-152.

21. S. Ramanujan, Collected Papers, Cambridge University Press, London, 1927 (reprinted: Chelsea, New York, 1962).

22. S. R. Ranganathan, Ramanujan, The Man and the Mathematician, Asia Publishing House, Bombay, 1967.

23. L. B. Richmond and G. Szekeres, Some formulas related to dilogarithms, the zeta function and the Andrews-Gordon identities, J. Australian Math. Soc. (4), 31 (1981), 362-373.

24. D. Shanks, A short proof of an identity of Euler, Proc. Amer. Math. Soc., 2 (1951), 747-749.

25. D. Shanks, Two theorems of Gauss, Proc. J. Math., 8 (1958) 609-612.

26. D. Shanks and J. W. Wrench Jr., Calculation of π to 100000 decimals, Math. of Comp., 16 (1962), 76-79.

27. G. N. Watson, The final problem: an account of the mock theta functions, J. London Math. Soc., 11 (1936), 55-80.

28. G. N. Watson, The mock theta functions, II, Proc. London Math. Soc. (2), 42 (1937), 272-304.

29. D. Y. Y. Yun and D. R. Stoutemyer, Symbolic Mathematical Computation Encyclopedia of Computer Science and Technology, Vol. 15 Supplement, Marcel Dekker, New York, 1980, 235-310.

THE PENNSYLVANIA STATE UNIVERSITY
UNIVERSITY PARK, PENNSYLVANIA 16802

The New SCRATCHPAD Language and System for Computer Algebra

Richard D. Jenks

Computer Algebra Group
Mathematical Sciences Department
IBM Research Center
Yorktown Heights, New York 10598

During the past seven years, IBM Research has been engaged in the design of a new implementation of the computer algebra system SCRATCHPAD. This system represents a new generation of computer algebra systems with a general-purpose programming language having generic operations and extensible, parameterized, and dynamically constructible abstract datatypes.

The system provides a single high-level language with an interpreter and compiler. This language can be used both by the naive user for convenient interactive mathematical calculations and by the advanced user for the efficient implementation of algorithms. Although especially designed for computer algebra, the language provides data abstraction and hiding mechanisms which generalize those found in such languages as CLU and Ada.

The system is semantically built on three concepts: categories, domains, and packages. Categories (e.g. Group, Ring, Set) name a set of operations together with corresponding attributes (such as "commutative") that the operations are asserted to have. Domains (e.g. Integer, Matrix, BalancedBinaryTree) provide a set of functions which implement categorical operations so as to satisfy the required attributes. Packages are clusters of functions, generally parameterized by categories and/or domains (e.g., a package for solving any system of multivariate polynomial equations over a field by computing a Grobner Basis). As a consequence, the new SCRATCHPAD system will provide facilities similar to old SCRATCHPAD except that those for this system will be very much more general (for example, it will have a package for integrating rational functions over any field of characteristic zero). In addition, users of the new system may construct and compute with any algebraically meaningful domain (such as "matrices of polynomials in x,y with integer coefficients extended by the square root of 5", or, in general, matrices over any ring).

Another advantage of the new system is its extensibility. In previous SCRATCHPAD, top-level user code was interpreted and therefore ran significantly slower than built-in system code. Extensions could otherwise be made only by system experts descending into LISP; unfortunately, these extensions were exceedingly susceptible to error due to a preponderance of system-wide global variables. In new SCRATCHPAD, all system-defined categories, domains, and packages are accessible for user enhancement, modification, and re-compilation. Since the same language is used both by users and system designers alike, no performance penalty is paid for user extension. Algebra code is written in modular components with no global-variable interdependencies. Components are linked together at

run-time with suitable checking for algebraic consistency as specified by the language (for example, it is not possible to create a matrix over a coefficient domain which is not a ring; also, an attempt to compute the determinant of a matrix using an algorithm for which multiplication had been declared to be commutative will be signaled as a semantic error if multiplication in the coefficient ring of the matrix is non-commutative).

Currently implemented domains are: number domains: short integers and bignums, floats and bigfloats, rational numbers, gaussians (integers, complex numbers), algebraic numbers, integers mod p, factored integers, and integer subdomains (positive-integers, non-negative-integers, even-integers, etc.); other atomic domains: strings, symbols, and general expressions; data structures: lists, vectors, records, unions, and balanced-binary-trees; polynomial domains: univariate (sparse/dense in exponents/coefficients, named/unnamed variables), multivariate (sparse/dense in coefficients, named/unnamed variables); matrices: general, rectangular, and square; gaussians and quaternions over any ring; algebraic functions; integral basis; direct products of any domain; and, fractions (quotient-fields and general localizations).

Currently implemented algebraic packages include those for: factorization of integers; factorization of polynomials: univariate over finite field, univariate/multivariate over integers (using a generalized Hensel lifting package); rational function integration; real and complex root finding; and, solution of linear and polynomial equations. A full implementation of integration (transcendental and algebraic cases) and multivariate power series is expected to be completed this year.

The interactive interface for new SCRATCHPAD bears strong resemblance to its predecessor. Users can use the system as a symbolic desk calculator, write rewrite-rules, and compose functions interactively, generally without type declarations. Essentially,

interactive language = programming language - restrictions

of mandatory type-declarations. The notion of "map" resembles a similar concept in SMP and is used to represent rewrite-rules, finite/infinite sequences, and function definitions at top level. System commands provide for various iteractive utilities such as reading/writing of input/output files, editing, tracing, querying interactive databases, and on-line documentation. User input/output is stored on a user's history file for later retrieval. An "undo" command enables interactive backtracking in history to a previous point in the interactive conversation.

SCRATCHPAD is an experimental program in a research stage of development. It is expected to become available to a limited number of users for test and evaluation by agreement with IBM Research over CSNET in Fall 1984. The system was initially demonstrated at the April 1984 conference at NYU entitled "Computer Algebra as a Tool for Research in Mathematics and Physics". A booklet of examples and a language primer handed out at this meeting can be obtained by writing to the author.

The design of the new SCRATCHPAD system is the product of many people, notably the author, James H. Davenport (Univ. of Bath), Barry M. Trager (IBM Research), David Y. Y. Yun (SMU), more recently, Victor S. Miller (IBM Research), and involved consultations early-on with David Barton (U. of Cal., Berkeley) and James W. Thatcher (IBM Research). Those responsible for its implementation include many of the above, Scott C. Morrison (U. of Cal., Berkeley), Christine J. Sundaresan (IBM Research), Robert S. Sutor (IBM Research), Josh Cohen (Yale University), Patrizia Gianni (Univ. of Pisa), and Michael Rothstein (Kent State University).

APPLICATION OF MACSYMA TO KINEMATICS AND MECHANICAL SYSTEMS

M.A. Hussain
General Electric Company
Corporate Research and Development

B. Noble
Mathematics Research Center
University of Wisconsin

ABSTRACT

The objective of this paper is to ilustrate that symbol manipulation systems can readily handle many of the typical symbolic calculations arising in the formulation of problems in kinematics and mechanical systems.

The paper consists of two parts. First, we discuss the use of MACSYMA in connection with the algebraic manipulations involved in transferring a body from one position to another in space, with particular reference to Rodrigues and Euler parameters and successive rotations, and an example involving quaternions. Second, we indicate how MACSYMA can be used to set up dynamical equations for the Stanford manipulator arm, and a spacecraft problem.

INTRODUCTION

Kinematics is a basic tool for the analysis of mechanisms and mechanical systems. Until recently, the most common approach has been to use vectors and Euler angles. More recently, other approaches have been gaining in popularity because of computers. We illustrate by several examples that these approaches are particularly amenable to symbolic manipulation. The immediate objective is limited, namely to indicate that several methods of representing rotations including Rodrigues and Euler parameters, and quaternions can be handled by MACSYMA by a unified approach that would seem to have some elements of novelty. But also it should be clear that our examples suggest a different approach to dynamical problems such as those considered by Branets and Shmyglevskiy [3] using quaternions and Dimentberg [4] using the screw calculus. The nearest connected account of the type of approach we have in mind is the mss. [12] by Nikravesh et al., but a systematic use of computer symbolic manipulation would certainly affect the detailed treatment. This is the first part of the paper.

It is clear that the complexity of mechanical systems is increasing to the point where symbol manipulation must play an important part in their formulation and solution. We illustrate by two dynamical examples, one involving a robot arm, the other a spacecraft problem. The main reason for choosing these particular examples is that the equations have been formulated and published in quite a detailed form already. By comparing our treatment with those already published, the reader will be able to make a judgment for himself concerning the usefulness of MACSYMA, and also how thinking in terms of symbol manipulation does change one's approach to the formula-

tion of the equations. We give the MACSYMA programs in detail in Appendices in order to encourage users of other systems to do the same.

KINEMATICS EXAMPLES

1. The Representation of Rotation by Orthogonal Matrices

We remind the reader of some standard results. We work in terms of matrices (this can be converted into vector interpretations as appropriate) using lower case for column matrices and upper case for rectangular matrices with more than one column.

A rotation of a body with a fixed point by an angle ϕ around an axis defined by the unit column matrix $\mathbf{n} = [n_1,n_2,n_3]^T$ transfers a point $\mathbf{r} = [x,y,z]^T$ into a point $\mathbf{r}' = [x',y',z']^T$ by (cf. Bottema and Roth [1] p. 59) $\mathbf{r}' = \mathbf{A}\mathbf{r}$ where (see Figure 1)

$$\mathbf{A} = [\cos\phi\mathbf{I} + (1 - \cos\phi)\mathbf{n}\mathbf{n}^T + \sin\phi\mathbf{N}] \ , \tag{1}$$

$$\mathbf{N} = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix} \tag{2}$$

(Note that N corresponds to the vector $\mathbf{n} \times \mathbf{r}$, and I is the identity matrix).

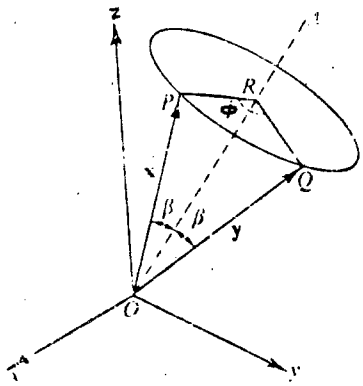


Figure 1. Rotation of a body with a fixed point.

The matrix A is orthogonal. We discuss three different ways of proving this using MACSYMA:

a) The simplest and most direct way is to express (1) in component form and simply check by brute force that $\mathbf{A}^T\mathbf{A} = \mathbf{I}$.

b) Alternatively we could use MACSYMA interactively as follows. It is easily checked that

$$\mathbf{n}^T\mathbf{n} = 1, \ \mathbf{N}^T = -\mathbf{N}, \ \mathbf{n}^T\mathbf{N} = 0, \ \mathbf{N}\mathbf{n} = 0, \ \mathbf{N}^2 = \mathbf{n}\mathbf{n}^T - \mathbf{I} \tag{3}$$

We use MACSYMA to form $\mathbf{A}^T\mathbf{A}$, will give nine terms involving $\mathbf{n}^T\mathbf{N}$, $\mathbf{n}\mathbf{n}^T\mathbf{n}\mathbf{n}^T$, $\mathbf{N}^2$ etc., and we use SUBST to simplify and finally derive $\mathbf{A}^T\mathbf{A} = \mathbf{I}$.

c) We can use TELLSIMP to build the rules (3) into MACSYMA. Then MACSYMA program can be written to produce the result I for $\mathbf{A}^T\mathbf{A}$.

Method a) is clearly simplest. Method c) is surprisingly tricky in MACSYMA because in addition to (3) we have to distinguish between scalars and matrices, and set proper switches. For verifying that $AA^T = I$, the simplest method is to use a) not c), but for more complicated problems, method a) soon produces algebraic expressions of horrendous complexity. As problem size increases, method c) will become preferable. In this paper, we have used the component form but; further developments may require the more abstract approach.

2. Rodrigues Parameters

We introduce these by stating the result that any 3×3 orthogonal matrix A can be expressed in the following product form by the Cayley-Klein decomposition which says that these exists a skew-symmetric 3×3 matrix B such (cf. Bottema and Roth [1], p. 10):

$$A = (I - B)^{-1}(I + B) \tag{4}$$

This tells us immediately that $B = (A - I)(A + I)^{-1}$. MACSYMA gives us directly (Appendix I):

$$b_i = n_i \tan \frac{1}{2}\phi \qquad i = 1,2,3 \tag{5}$$

The b_i $(i = 1,2,3)$ are the Rodrigues parameters.

We first express A in terms of the Rodrigues parameters. We find (Appendix II cf.. (Bottema and Roth [1] p. 148):

$$A = \frac{1}{\Delta}\begin{bmatrix} 1 + b_1^2 - b_2^2 - b_3^2 & 2(b_1b_2 - b_3) & 2(b_1b_3 + b_2) \\ 2(b_2b_1 + b_3) & 1 - b_1^2 + b_2^2 - b_3^2 & 2(b_2b_3 - b_1) \\ 2(b_3b_1 - b_2) & 2(b_3b_2 + b_1) & 1 - b_1^2 - b_2^2 + b_3^2 \end{bmatrix} \tag{6}$$

where $\Delta = 1 + b_1^2 + b_2^2 + b_3^2$. Using the notation $A = [a_{ij}]$, it is clear from this result that:

$$\begin{aligned} b_1 &= (a_{32} - a_{23})/d \\ b_2 &= (a_{13} - a_{31})/d \\ b_3 &= (a_{21} - a_{12})/d \end{aligned} \tag{7}$$

With $d = 1 + a_{11} + a_{22} + a_{33}$. Having established the necessary background, we derive typical basic results by means of MACSYMA. The reader should compare our derivation with those of, for example, Bottema and Roth [1], Gibbs [5], and Dimentberg [4].

Consider the result of first rotating a body round an axis n by angle ϕ, then around a second axis n' by an angle ϕ'. Euler's theorem tells us that the result is equivalent to a rotation by some angle ϕ'' round some axis n''. In matrices, if the matrices corresponding to these three rotations are A, A', A'' and we start with a point r, this is first transformed into $r' = Ar$, and then r' is transformed into $r'' = A'r'$. We also have $r'' = A''r$ so that

$$A'' = A'A$$

The Rodrigues parameters corresponding to n'', ϕ'' are given by (7) where a_{ij} are the

elements of A''. But these are given in terms of the first two rotations by the the corresponding elemnts of A'A .These matrix relations are carried out by MACSYMA in Appendix III, giving the result:

$$\mathbf{b}'' = \frac{\mathbf{b} + \mathbf{b}' - \mathbf{B}'\mathbf{b}'}{1 - \mathbf{b}^T\mathbf{b}'} \tag{8}$$

where B is related to b as N was to n in (2).

Note that this is a straightforward derivation that would be laborious to carry out by hand, as compared with derivations carried out in the literature that depend on special methods.

3. Euler Parameters

Instead of using Rodrigues parameter b_i, it is often convenient to use Euler parameters c_i related to b_i by (Bottema and Roth [1] p. 150)

$$b_i = c_i/c_0 \ , \ c_0^2 + c_1^2 + c_2^2 + c_3^2 = 1 \tag{9}$$

The relation (5) then gives

$$\mathbf{A} = \begin{bmatrix} c_0^2 + c_1^2 - c_2^2 - c_3^2 & 2(-c_0c_3 + c_1c_2) & 2(c_0c_2 + c_1c_3) \\ 2(c_0c_3 + c_2c_1) & c_0^2 - c_1^2 + c_2^2 - c_3^2 & 2(-c_0c_1 + c_2c_3) \\ 2(-c_0c_2 + c_3c_1) & 2(c_0c_1 + c_3c_2) & c_0^2 - c_1^2 - c_2^2 + c_3^2 \end{bmatrix} \tag{10}$$

Although it would seem that the Euler parameters are straightforward homogeneous forms of the Rodrigues parameters, it turns out that some relations are expressed much more simply in terms of the Euler parameters.

One example is the Euler parameter analog of (8) for two successive rotations. To derive this, substitute $\mathbf{b} = \mathbf{c}/c_0$, $\mathbf{b}' = \mathbf{c}'/c_0$ in (8) which gives:

$$\mathbf{b}'' = \frac{c_0'\mathbf{c} + c_0\mathbf{c}' - \mathbf{C}\mathbf{c}}{c_0c_0' - \mathbf{c}^T\mathbf{c}'} \tag{11}$$

When this is written out in detail we find that by introducing

$$\begin{aligned} c_0'' &= c_0'c_0 - c_1'c_1 - c_2'c_2 - c_3'c_3 \\ c_1'' &= c_1'c_0 + c_0'c_1 - c_3'c_2 + c_2'c_3 \\ c_2'' &= c_2'c_0 + c_3'c_1 + c_0'c_2 - c_1'c_3 \\ c_3'' &= c_3'c_0 - c_2'c_1 + c_1'c_2 + c_0'c_3 \end{aligned} \tag{12}$$

equation (11) can be written in the simple form

$$\mathbf{b}'' = \mathbf{c}''/c_0'' \tag{13}$$

In Appendix IV we check by MACSYMA that if $c_0^2 + \mathbf{c}^T\mathbf{c} = 1$, $(c_0')^2 + (\mathbf{c}')^T\mathbf{c} = 1$, then $(c_0'')^2 + (\mathbf{c}'')^T\mathbf{c} = 1$, which is a well-known result due to Eulers. This result and (12)

mean that c_0'',c_1'',c_2'',c_3'' are the Eulers parameters corresponding to the total rotation.

In the literature, the result (12) is often derived via quarternions (e.g. Bottema and Roth [1], p. 150). It is of some interest to express this approach in the present context of Euler parameters and matrices which can be done without mentioning quaternions explicitly. Introduce γ and Γ defined as follows:

$$\gamma = \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}, \quad \Gamma = \begin{bmatrix} c_0 & -c_1 & -c_2 & -c_3 \\ c_1 & c_0 & -c_3 & c_2 \\ c_2 & c_3 & c_0 & -c_1 \\ c_3 & -c_2 & c_1 & c_0 \end{bmatrix}$$

If γ',Γ' are the corresponding matrices with c' in place of c, and similarly for γ'',Γ'', we define the product $\gamma'\gamma$ by (compare the remark following (2)):

$$\gamma'' = \gamma'\gamma = \Gamma'\gamma \tag{14}$$

which says exactly the same as (12). We first note that if we define $\gamma^{-1} = [c_0,-c_1,-c_2,-c_3]$ then $\gamma\gamma^{-1} = \gamma^{-1}\gamma = [1,0,0,0]^T$. It can be verified (e.g. by the MACSYMA program in Appendix V) that introducing $\rho = [r_0,r_1,r_2,r_3]^T$, $\mathbf{r} = [r_1,r_2,r_3]^T$ and $\rho',\mathbf{r}'$ correspondingly, then if we form $\gamma\rho\gamma^{-1}$, and denote the result by ρ^1, item

$$\begin{bmatrix} r_0' \\ \mathbf{r}' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \mathbf{A} \end{bmatrix} \begin{bmatrix} r_0 \\ \mathbf{r} \end{bmatrix}$$

Where A is precisely the matrix that appeared in (10), i.e., $\gamma\rho\gamma^{-1}$ represents a rotation. This is our version of the standard quaternion theorem on rotation, derived of course from a completely different point of view (cf. Brand [2], p. 417). A second rotation would give $\rho'' = \gamma'\rho'(\gamma')^{-1}$, and combining the rotations leads to $\rho'' = \gamma'\gamma\rho(\gamma')^{-1}\gamma^{-1}$, i.e., if γ'' represents the combined rotation then $\gamma'' = \gamma'\gamma$, which is identical with (12).

Still another way of obtaining (12) is suggested by the discussion of Cayley-Klein parameters in Bottema and Roth ([1] p. 529), namely that a result corresponding to equation (9.8) in that reference should hold for Euler parameters. We introduce the notations:

$$\mathbf{V} = c_0\mathbf{I} + \mathbf{S}$$

$$\mathbf{V}^{-1} = c_0\mathbf{I} - \mathbf{S}$$

$$\mathbf{S} = \begin{bmatrix} 0 & -c_1 & -c_2 & -c_3 \\ c_1 & 0 & -c_3 & c_2 \\ c_2 & c_3 & 0 & -c_1 \\ c_3 & -c_2 & c_1 & 0 \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} y & z & 0 & -x \\ z & -y & x & 0 \\ 0 & x & y & z \\ -x & 0 & z & -y \end{bmatrix}$$

$$\mathbf{Q} = \begin{bmatrix} Y & Z & 0 & -X \\ Z & -Y & X & 0 \\ 0 & X & Y & Z \\ -X & 0 & Z & -Y \end{bmatrix}$$

The MACSYMA program in Appendix VI does the following. We form $\mathbf{VqV}^{-1}$ and

equate this to Q. This gives 16 equations. However, it is easily checked by MACSYMA that, in fact, there are only *three* independent relations involving x,y,z and X,Y,Z which can be written in the form

$$\mathbf{Aq} = \mathbf{Q}$$

where A is exactly the A given in (10). The implication of this, in connection with repeated rotations, is that if q corresponds to r and Q to r' defined in the second paragraph of Section 2 and the corresponding V is denoted by V, then

$$\mathbf{VrV}^{-1} = \mathbf{r}'$$

Similarly, the second rotation gives $V'r'V'^{-1} = r''$ and the rotation from the initial position to the final position gives $V''rV''^{-1} = r''$. Eliminating r' we have $V''r(V'')^{-1} = V'VrV^{-1}V'^{-1}$ so that finally

$$V'' = V'V \tag{14}$$

and this is precisely equation (12).

4. An Example Involving Dual Quaternions

The discussion in the last two sections was concerned with the rotation of a body with a fixed point and involved only three independent parameters. The general motion of a body involves displacement, as well as rotation, and requires six independent parameters. Rather than extending the methods of the last two sections, we illustrate how MACSYMA deals with a rather different approach to kinematics, namely via quaternions, by considering a calculation in a classic paper by Yang and Freudenstein ([14], 1964) dealing with a spatial four-bar mechanism.

In Figure 2, MA and NB are two nonparallel and nonintersecting lines. MN is the common perpendicular. Let a,b denote unit vectors in the direction of MA, NB respectively, and let r_a, r_b denote the vectors $\overrightarrow{OM}$, $\overrightarrow{ON}$. We introduce the quaternions

$$\hat{a} = a + \epsilon(r_a \times a) \quad , \quad \hat{b} = b + \epsilon(r_b \times b)$$

where ϵ is a symbol with the property that $\epsilon^2 = 0$. Note that this implies, for example, that if $\hat{\theta} = \theta + \epsilon s$ then

$$\sin\hat{\theta} = \sin\theta + \epsilon s\cos\theta \quad , \quad \cos\hat{\theta} = \cos\theta - \epsilon s\sin\theta \tag{15}$$

As discussed by Yang et al. [14], the relative shift between $\hat{a}$ and $\hat{b}$ can be expressed as

$$\hat{b} = Q\hat{a} \quad , \quad \hat{a} = \hat{b}Q$$

where Q is a dual quaternion (see [14], (22, 23)). Successive application of formulae of this type gives rise to a loop closure equation for the mechanism of the form:

$$A(\hat{\theta}_1)\sin\hat{\theta}_4 + B(\hat{\theta}_1)\cos\hat{\theta}_4 = C(\hat{\theta}_1) \tag{16}$$

where

$$A(\hat{\theta}_1) = \sin\hat{\alpha}_{12}\sin\hat{\alpha}_{34}\sin\hat{\theta}_1$$

$$B(\hat{\theta}_1) = -\sin\hat{\alpha}_{34}\,(\sin\hat{\alpha}_{41}\cos\hat{\alpha}_{12} + \cos\hat{\alpha}_{41}\sin\hat{\alpha}_{12}\cos\hat{\theta}_1)$$

$$C(\hat{\theta}_1) = \cos\hat{\alpha}_{23} - \cos\alpha_{34}\,(\cos\hat{\alpha}_{41}\cos\hat{\alpha}_{12} - \sin\hat{\alpha}_{41}\sin\hat{\alpha}_{12}\cos\hat{\theta}_1)$$

Here

$$\hat{\alpha}_{12} = \alpha_{12} + \epsilon\alpha_{12}\ ,\quad \hat{\theta}_1 = \theta_1 + \epsilon s_{11}$$

$$\hat{\alpha}_{23} = \alpha_{23} + \epsilon\alpha_{23}\ ,\quad \hat{\theta}_2 = \theta_2 + \epsilon s_2$$

$$\hat{\alpha}_{34} = \alpha_{34} + \epsilon\alpha_{34}\ ,\quad \hat{\theta}_3 = \theta_3 + \epsilon s_3$$

$$\hat{\alpha}_{41} = \alpha_{41} + \epsilon\alpha_{41}\ ,\quad \hat{\theta}_4 = \theta_4 + \epsilon s_4$$

It is then clear that (15) can be reduced to the form

$$P + \epsilon Q = R + \epsilon S$$

where P, Q, R, and S are independent of ϵ. It is required to find the explicit form of P, Q, R, and S. To calculate this by hand is extremely laborious, but straightforward in MACSYMA. The program is given in Appendix VII.

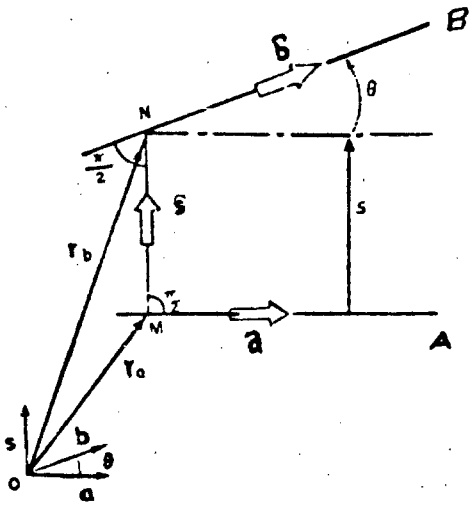


Figure 2. Relative position of two line vectors.

TWO EXAMPLES IN DYNAMICS

5. Equations of Motion for the Stanford Manipulator Arm

There are a number of ways to set up dynamical equations for robot manipulator arms (see Paul [13]). Kane-Levinson [9] have given an example of setting up dynamical equations for the Stanford manipulator. Our objective is to reproduce these equations from an algorithmic point of view, without having to do by hand the kind of extensive manipulation given in that paper. The method can help us to set up similar sets of equations for any manipulator automatically, thereby reducing the labor. We also show that MACSYMA can simplify the Kane-Levinson end-result, reducing the numbers of arithmetic operations required to complete numerical results.

We consider the Stanford manipulator arm (Paul [13]), a six-element, six-degree-

of-freedom manipulator. A schematic representation of this arm is given in Figure 3, from Kane-Levinson [9], where more details can be found. The six bodies are designated A, ..., F. Body A can be rotated about a vertical axis fixed in space. A supports B which can be rotated about a horizontal axis fixed relative to A. The figure should now be self-explanatory, the joint connecting B and C being translational, and the remaining joints rotational.

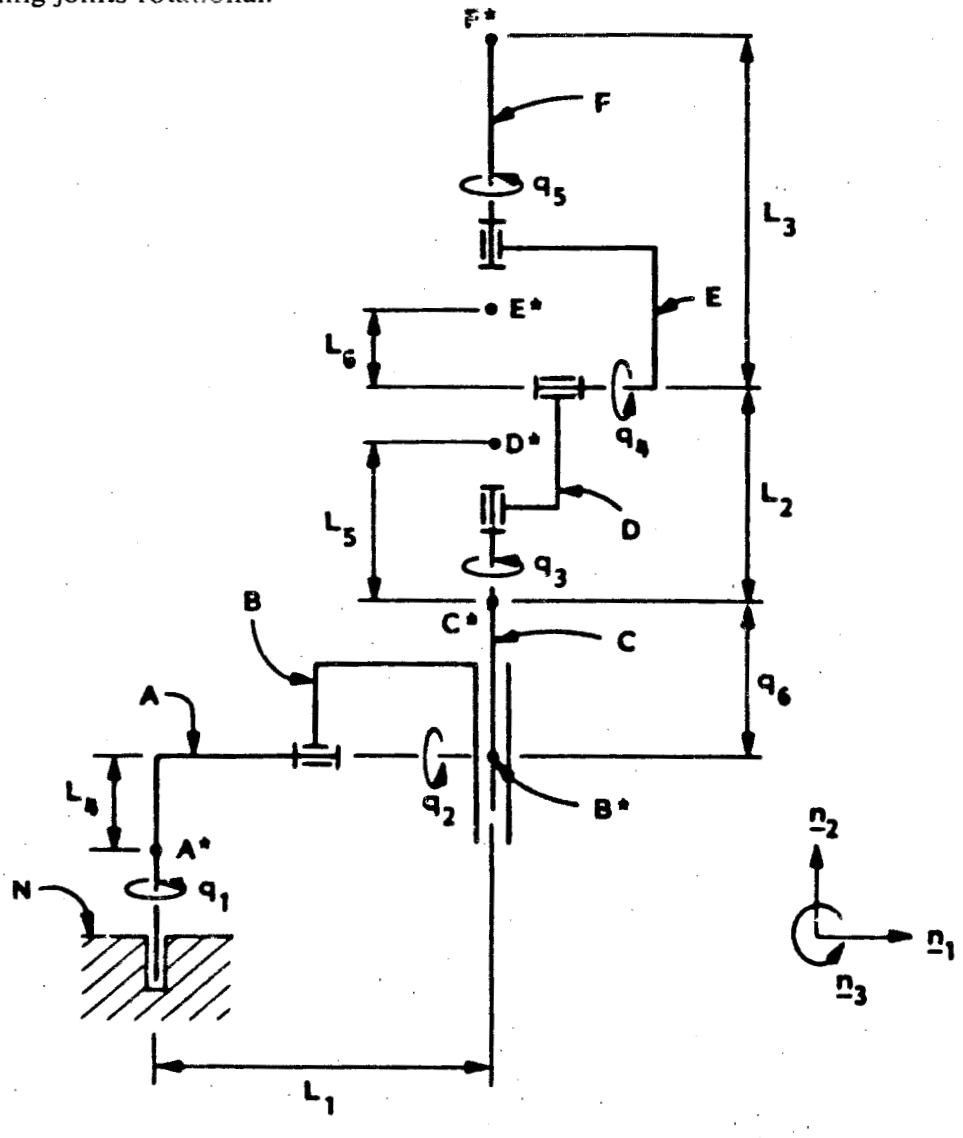


Figure 3. A schematic representation of Stanford manipulator arm.

$q_1, \ldots, q_6$ are generalized coordinates characterizing the instantaneous configuration of the arms, the first five being rotational and q_6 translational. For the plane configuration of the arms as drawn in Figure 3, it is assumed that $q_1, \cdots q_5$ are zero.

We choose coordinate axes as follows. n_1, n_2, n_3 are unit vectors fixed in space as

indicated in Figure 3, n_1, n_2, lying in the plane of the paper. a_1, a_2, a_3 are unit vectors fixed in the arm A which coincide with n_1, n_2, n_3 when the arm is in the configuration of Figure 3. Similarly, b_1, b_2, b_3 are unit vectors attached to the arm B and similarly for C, D, E, and F.

We give a mathematical description of an algorithm for setting up the dynamical equations. This is essentially the algorithm described by Kane-Levinson [9], but organized in a somewhat different way in order to facilitate implementation on MACSYMA. The stages and details of the MACSYMA program which are in Appendix VIII, parallel the mathematical description that follows:

Stage 1: .Set up angular velocities:

Rotations about x,y,z axes can be described by orthogonal matrices of simple form as discussed in detail by Paul [13], Chapter 1. For instance, rotation by an angle θ about the x-axis involves ([13], p. 15)

$$\mathrm{Rot}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

Let R_1, ... R_5 denote matrices corresponding to rotations θ_1, ..., θ_5 about axes, y,x,y,x,y respectively in the local coordinates fixed relative to arms A, B, C, D, E, F. Let $\dot{q}_1$, ..., $\dot{q}_5$ denote angular velocities around y,x,y,x,y axes respectively. These are vector quantities represented by matrices that we denote by ω_1, ..., ω_5. For instance, $\omega_1 = [0,\dot{q}_1,0]$ etc. Similarly, for the linear velocity $\dot{q}_6$.

Next introduce ω^A, ..., ω^F, the angular velocities of A, ..., F in our Newtonian frame of reference, but with components expressed in the local coordinate frame of reference. For example:

$$\omega^D = [u_1,u_2,u_3] \text{ means: } \omega^D = u_1\underline{d}_1 + u_2\underline{d}_2 + u_3\underline{d}_3 \tag{17}$$

The algorithm for computing ω^A, ..., ω^F is given by:

$$\omega^A = \omega_1 R_1$$

$$\omega^B = (\omega^A + \omega_2)R_2$$

$$\omega^C = \omega^B$$

$$\omega^D = (\omega^C + \omega_3)R_3$$

$$\omega^E = (\omega^D + \omega_4)R_4$$

$$\omega^F = (\omega^E + \omega_5)R_5$$

If these formulae are used as they stand, the expression for ω^F in terms of $\dot{q}_i$ will be complicated. The complexity can be reduced using a method due to Kane-Levinson [9]. The u_i that occur in (16) can be expressed in terms of $\dot{q}_1$, $\dot{q}_2$, $\dot{q}_3$ as follows

$$u_1 = \dot{q}_1 \sin q_2 \sin q_3 + \dot{q}_2 \cos q_3$$

$$u_2 = \dot{q}_1 \cos q_2 + \dot{q}_3$$

$$u_3 = -\dot{q}_1 \sin q_2 \cos q_3 + q_2 \sin q_3$$

$$u_i = \dot{q}_i \quad i = 5,6,7$$

Stage 2: Set up linear velocities:

In stage 1, the angular velocities were always expressed in local coordinates corresponding to the arm being considered. This is not necessarily the case for the way in which Kane-Levinson [9] formulate the linear velocities (see paragraph preceeding (28) in the paper). Because we wish our results to be comparable to those in [9], we state the formulae we use, which will lead to results that are the same as those in equations (28-43) in [9]. (Note that the stars in the following refer to the velocities of the centers of mass of the corresponding arms.)

$$v^{A^*} = 0$$

$$v^{B^*} = \omega^A \times R^B$$

$$v^{C^*} = \omega^C \times R^C + \bar{q}_0$$

$$v^{D^*} = \omega^B \times R^D + \bar{q}_0$$

The expressions for v^{E^*}, v^{F^*} correspond to those in equation (40) and (42) in the Kane-Levinson paper [9]. The exact form we use can be found from the expressions for VE and VF in the MACSYMA program given in Appendix VIII.

The remaining stages are relatively straightforward.

Stage 3: Find the partial angular velocities.

Stage 4: Find the partial linear velocities.

These are explained in the Kane-Levinson paper [9] and the MACSYMA implementation in Appendix VIII is self-explanatory.

Stage 5: Find the angular accelerations.

Stage 6: Find the linear accelerations.

These are obtained by simple differentiation of the corresponding angular and linear velocities as given in the MACSYMA program in Appendix VIII.

Stage 7: Define moments of inertia.

We next have to consider forces.

Stage 8: Define torques.

Stage 9: Set up generalized forces.

Stage 10: Set up active forces.

Stage 11: Set up Kane's equations.

These steps are straightforward; the MACSYMA program is given in appendix.

Finally, Figure 4 gives a comparison of some numerical results obtained from MACSYMA and Kane-Levinson [9].

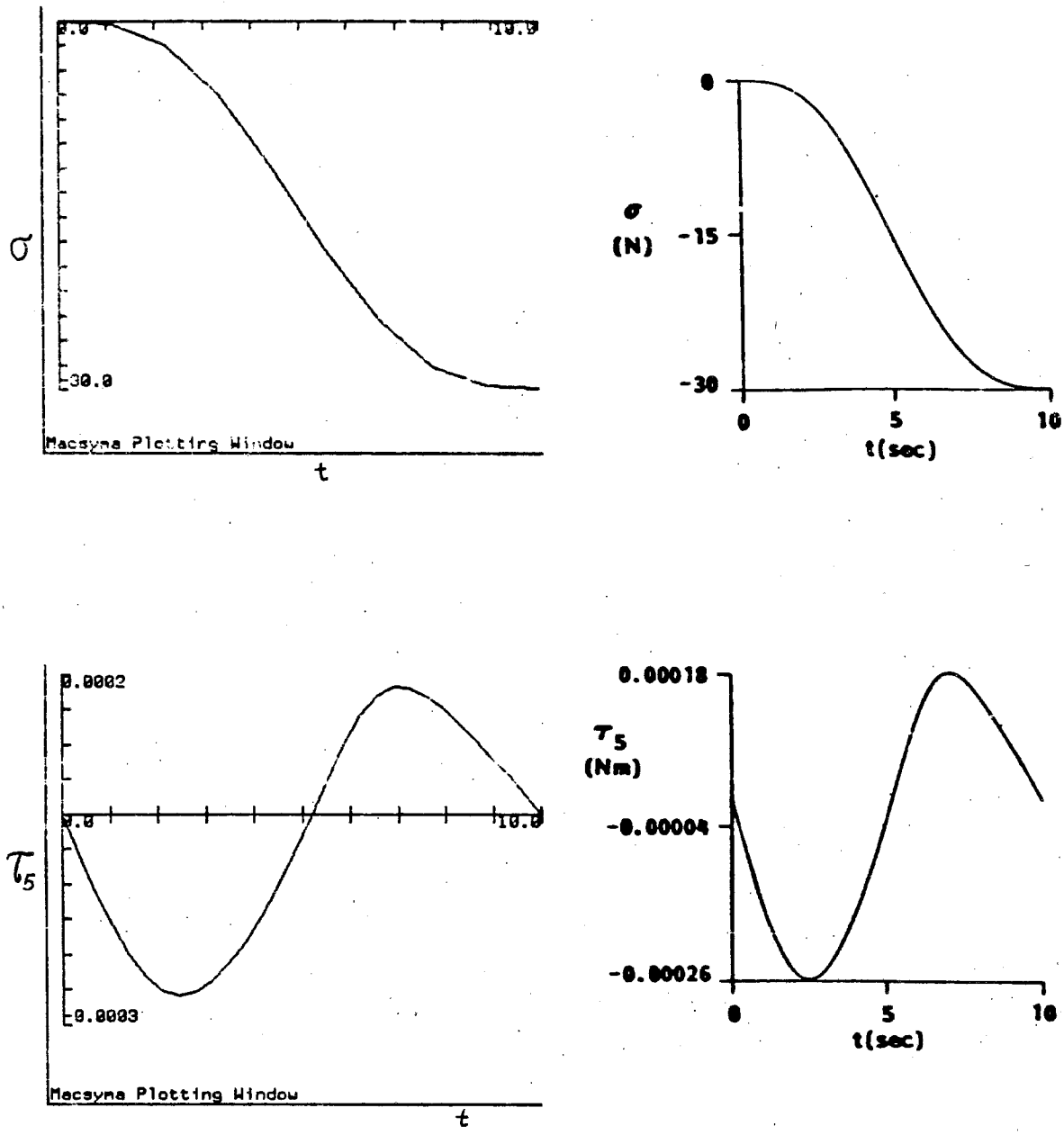


Figure 4a. Comparison of numerical results for σ. τ_5 obtained by MACSYMA and Reference 9.

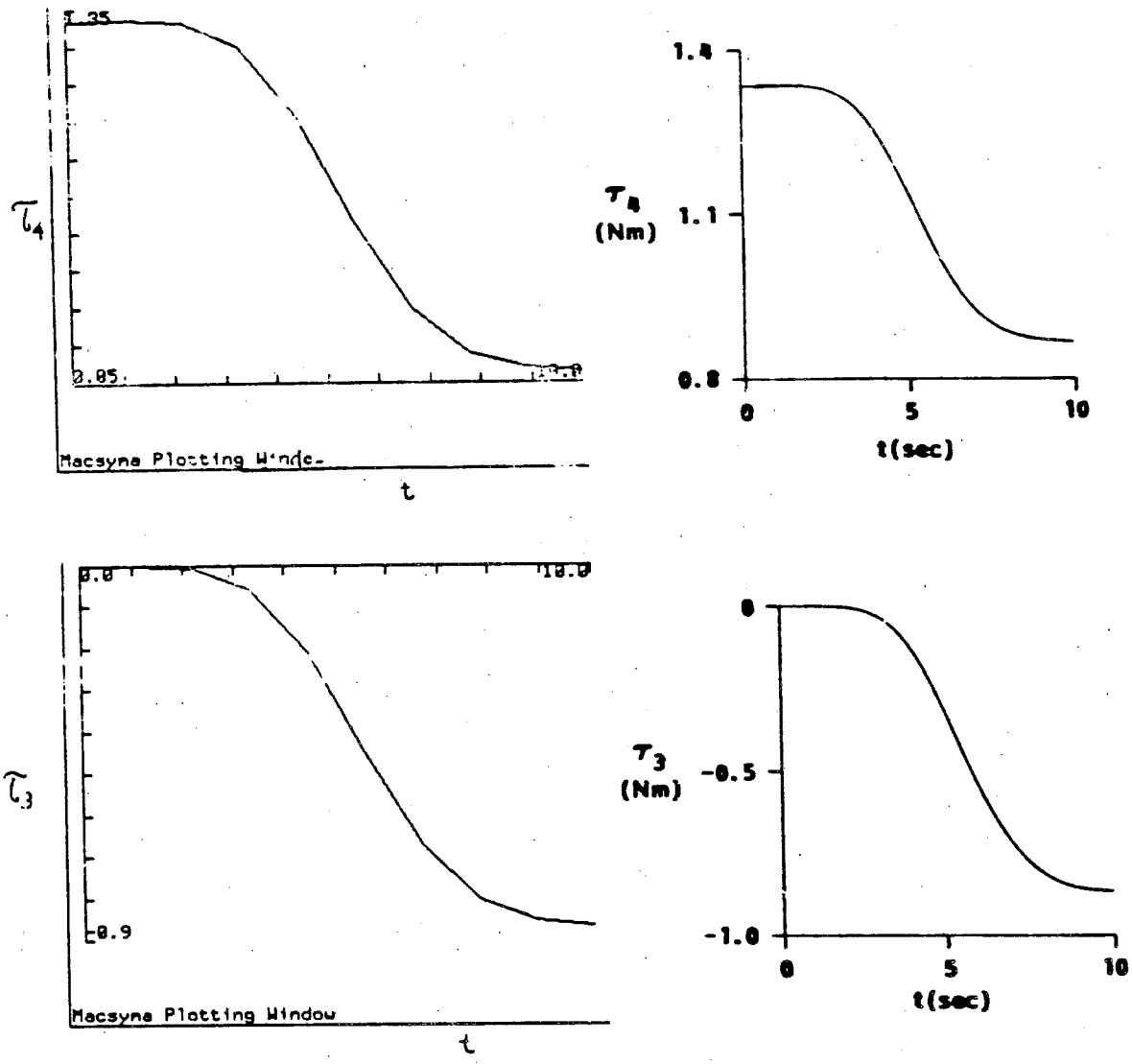


Figure 4b. Comparison of numerical results for τ_4 and τ_3 obtained by MACSYMA and Reference 9.

It is of some interest to compare the mathematical equations in the Kane-Levinson paper with the corresponding MACSYMA expressions. For example, consider:

Kane-Levinson [9] (underlined quantities are vectors)		MACSYMA
$\underline{\omega}^A = \dot{q}_1 \underline{a}_2$	(13)	WA: EXPAND(W1.R1) $\underline{\omega}^A \equiv$ WA , $\dot{q}_1 \underline{a}_2 \equiv$ W1.R1

But

$$\dot{q}_1 = \frac{u_1 s_3 - u_3 c_3}{s_2} \qquad (8)$$

Here ω^A, $\dot{q}_1$, $\underline{a}_2$ are vectors; WA, W1.R1 are matrices

Introduce

$$Z_4 = \frac{s_3}{s_2},\ Z_5 = -\frac{c_3}{c_2}$$

Then (13) becomes:

$$\underline{\omega}^A = (Z_4 u_1 + Z_5 u_3)\underline{a}_2 \qquad (15)$$

Similarly,

$$\underline{\omega}^B = Z_2\underline{b}_1 + Z_{10}\underline{b}_2 + Z_{11}\underline{b}_3 \qquad (16)$$

WB: EXPAND ((WA+W2).R2)

$$Z_{10} = Z_6 u_1 + Z_7 u_3\ Z_{11} = Z_8 u_1 + Z_9 u_3$$

One point here is that because Kane-Levinson [9] are carrying out the algebra by hand, it is convenient for them to introduce intermediate symbols $Z_1, Z_2 \cdots$ going up to Z_{196}, and similarly, 36 X's and 31 W's. MACSYMA has no difficulty in generating the end result in explicit form. These end results are no more complex than the complexity of the equation given in [9]. At the time of writing this paper a preliminary number count on additions and multiplication, for X_{ij}, the coefficients of equations of motion, obtained by MACSYMA, as compared to those in [9], shows a reduction by approximately a factor of two.

In conclusion, we note that Paul [13] sets up the dynamical equation of the Stanford manipulator arm using the Lagrangian equation approach. See also [6]. Some applications of the Lagrange method using MACSYMA are discussed in [9].

Various methods of setting up dynamical equations that could be carried out by MACSYMA are illustrated in [8].

6. A Spacecraft Problem

Levinson [11] has described in detail an application of the symbolic language FORMAC to formulate the spacecraft problem shown in Figure 5, consisting of two rigid bodies with a common axis of rotation *b*. (See also [10], pp.279-285).

The equations are given in complete detail in Ref. [11], and translated into MACSYMA in Appendix IX. In the example in the last section, we wrote the MACSYMA program in terms of matrices. In Appendix IX, the present example is written in terms of vectors, by writing BLOCK functions to perform the dot and cross products. To illustrate the comparison of the vector equation with the corresponding MACSYMA expressions:

Equations from Ref. [11]				MACSYMA
$\underline{r}_2$	=	cosq $\underline{b}_2$ + sinq $\underline{b}_3$	(1)	R[2]:COS(Q)*B[2]+SIN(Q)*B[3];

$\omega^B = u_1 \underline{b}_1 + u_2 \underline{b}_2 + u_3 \underline{b}_3$ (3) WB:U[1]*B[1]+U[2]*B[2]+U[3]*B[3];

$\mu_4 = \dot{q}$ U[4]:DIFF(Q,T);

$\alpha^R = \frac{d}{dt}(\omega^R) + \omega^B \times \omega^R$ (7) ALPR:DIFF(WR,T)+CROSS(WB,WR);

We discuss only one other correspondence. Equation (27) in Ref. [11] is

$$F_r = \frac{\partial v^{B^*}}{\partial \mu_r} \cdot (F)_B + \frac{\partial \omega^B}{\partial \mu_r} \cdot (T)_B \quad (r=1, \cdots, 7)$$

which becomes in MACSYMA

F(R):=DOT(DIFF(VBS,U[R]),FB) + DOT(DIFF(WB,U[R]),TB);

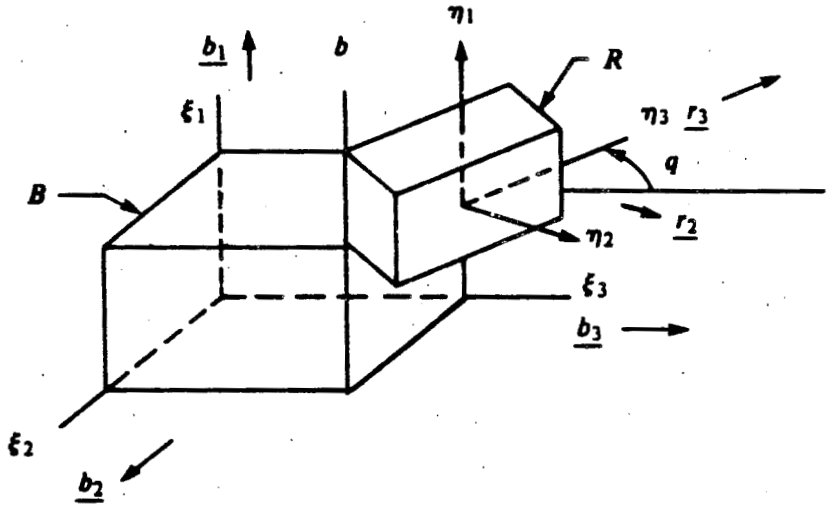


Figure 5. Two rigid bodies with a common axis of rotation.

The complete set of equations given in Ref. [11] is generated by Appendix IX. The reader should compare the corresponding FORMAC program given in Levinson [11].

CONCLUDING REMARKS

It should be clear from the examples given that symbolic manipulation by computer can carry out many of the laborious and routine calculations involved in the analysis of mechanical systems. But, potentially even more important, is the influence that symbolic manipulation is likely to have on the methods used to formulate problems. The reader should compare, for example, the algorithmic approach we have adopted to the Stanford manipulator arm problem with the approach in [9]. As another example, if symbolic manipulation methods are used, this will influence whether we formulate problems in terms of Euler angles, Euler parameters, Rodrigues parameters, or quaternions, etc. In addition, one can visualize the production of standard utilize MACSYMA software – e.g., a standard package written in MACSYMA to produce equations corresponding to those of Kane-Levinson [9] for *any* given combination of rotating and sliding joints.

REFERENCES

1. O. Bottema and B. Roth, *Theoretical Kinematics*, North-Holland, 1979.
2. L. Brand, *Vector and Tensor Analysis*, Wiley, 1957.
3. V. N. Branets and I. P. Shmyglevskiy, "Application of Quarternions to Rigid Body Rotation Problems," NASA Tech. Transl. TTF-15, 414, 1974 (1973 Russian original).
4. F. M. Dimentberg, "The Screw Calcules and its Applications in Mechanics," NTIS Transl. FTD-HT-23-1632-67, 1968 (1965 Russian original).
5. J. W. Gibbs, *Vector Analysis*, Dover reprint, 1960 (original published in 1909).
6. J. M. Hollerbach, "A Recursive Lagrangian Formulation of Manipulator Dynamics and a Comparative Study of Dynamics Formulation Complexity," *IEEE Trans. on Syst., Man and Cyb. SMC-10*, 1980, 730-736.
7. M. A. Hussian and B. Noble, "Application of Symbolic Computation to the Analysis of Mechanical Systems, Including Robot Arms," General Electric Technical Report 84CRD062, 1984. Also to be published in the Proceedings of the NATO Conference on Mechanisms, E. Haug, ed., University of Iowa, 1984.
8. T. R. Kane and D. A. Levinson, "Formulation of Equations of Motion for Complex Spacecraft," *J. Guidance and Control*, 1980, 99-112.
9. T. R. Kane and D. A. Levinson, "The Use of Kane's Dynamical Equations in Robotics", *Int. J. Robotics Research 2*, 1983, 3-21.
10. T. R. Kane, P. W. Likins, and D. A. Levinson, *Spacecraft Dynamics*, McGraw-Hill, 1983.
11. D. A. Levinson, "Equations of Motion for Multiple-Rigid-Body Systems via Symbol Manipulation," *J. Spacecraft and Rockets 14*, 1977, 479-487.
12. P. E. Nikravesh, R. A. Wehage, and E. J. Haug, *Computer-Aided Analysis of Mechanical Systems*, to be published.
13. R. P. Paul, *Robot Manipulators - Mathematics, Programming, and Control*, M. I. Press, 1981.
14. A. T. Yang and F. Freudenstein, "Application of Dual-Number Quarternion Algebra in the Analysis of Spatial Mechanisms," *Trans ASME J. Appl. Mech.*, 1966, 300-308.

APPENDIX I

```
/*PROVE THE IDENTITY OF EQUATION 5 */
/* TRIGNOMETRIC SIMPLIFICATION */
MATCHDECLARE(A,TRUE);
TELLSIMP(SIN(A)^2,1-COS(A)^2);
/* DEFINE CROSS PRODUCT MATRIX OR ALTERNATING TENSOR */
ALT(N):=MATRIX([0,-N[3,1],N[2,1]],[N[3,1],0,-N[1,1]],
[-N[2,1],N[1,1],0]);
N:MATRIX([N1],[N2],[N3]);
NN:ALT(N);
I:IDENT(3);
AA:COS(ALPHA)*I+(1-COS(ALPHA))*(N . TRANSPOSE(N))+NN*SIN(ALPHA);
AAP:I+AA;
/* WORK WITH HALF ANGLES */
ALPHA:BTA*2;
EV(AA);
TRIGEXPAND(%);
AA:%$
/* ADD IDENTITY MATRIX AND INVERT */
AAP:AA+I$
IAAP:AAP^^(-1)$
/*SUBTRACT IDENTITY MATRIX AND FORM MATRIX PRODUCT AS ANSWER*/
AAM:AA-I$
ANSWER:AAM . IAAP$
/*USE IDENTITY THAT N1^2+N2^2+N3^2=1 */
NN3:1-N1**2-N2**2;
ANSWER:RATSUBST(NN3,N3^2,ANSWER);
ANSWER:RATSIMP(%);
```

APPENDIX II

```
/* CAYLEY'S DECOMPOSITION OF ORTHOGONAL MATRIX
A =(I-B)^-1(I+B),WHERE B1,B2,B3 ARE RODRIGUES PARAMETERS*/
/* DEFINE CROSS PRODUCT OR ALTERNATING TENSOR MATRIX*/
ALT(N):=MATRIX([0,-N[3,1],N[2,1]],[N[3,1],0,-N[1,1]],
[-N[2,1],N[1,1],0]);
B:MATRIX([B1],[B2],[B3]);
BB:ALT(B);
I:IDENT(3);
INBB:(I-BB)^^-1;
A:INBB.(I+BB);
ANSWER:RATSIMP(%);
/*SOLVE ABOVE FOR B1 B2 B3,FOLLOWIN IS A CROSS CHECK */
DEL:RATSIMP(1+A[1,1]+A[2,2]+A[3,3]);
BB1:RATSIMP(1/DEL*(A[3,2]-A[2,3]));
BB2:RATSIMP(1/DEL*(A[1,3]-A[3,1]));
BB3:RATSIMP(1/DEL*(A[2,1]-A[1,2]));
```

APPENDIX III

```
/*TWO SUCCESIVE ROTATIONS IN TERMS OF RODRIGUES PARAMETER*/
ALT(N):=MATRIX([0,-N[3,1],N[2,1]],[N[3,1],0,-N[1,1]],[-N[2,1],N[1,1],0]);
B:MATRIX([B1],[B2],[B3]);
BB:ALT(B);
I:IDENT(3);
INBB:(I-BB)^^-1;
A:INBB.(I+BB)$
A:RATSIMP(%);
BP:MATRIX([BP1],[BP2],[BP3]);
BBP:ALT(BP);
INBBP:(I-BBP)^^-1;
AP:INBBP.(I+BBP)$
AP:RATSIMP(%);
APP:AP.A;
/*SOLVE ABOVE FOR BPP1 BPP2 BPP3 */
DEL:RATSIMP(1+APP[1,1]+APP[2,2]+APP[3,3]);
BPP1:RATSIMP(1/DEL*(APP[3,2]-APP[2,3]));
BPP2:RATSIMP(1/DEL*(APP[1,3]-APP[3,1]));
BPP3:RATSIMP(1/DEL*(APP[2,1]-APP[1,2]));
/*THE ABOVE RESULTS ARE SAME AS EQUATION (11) */
```

APPENDIX IV

```
/* DERIVE EULER IDENTITY SEE ALSO  BRAND REF. [2] P.408*/
S:MATRIX(
[0,-CC1,-CC2,-CC3],
[CC1,0,-CC3,CC2],
[CC2,CC3,0,-CC1],
[CC3,-CC2,CC1,0]);
SP:MATRIX(
[0,-CP1,-CP2,-CP3],
[CP1,0,-CP3,CP2],
[CP2,CP3,0,-CP1],
[CP3,-CP2,CP1,0]);
I:IDENT(4);
V:CC0*I+S;
VP:CP0*I+SP;
MAT1:V.VP;
/* NOW TAKE THE FIRST COLUMN OF THE ABOVE MATRIX AND SQUARE IT*/
MAT2:SUBMATRIX(%,2,3,4);
ANSWER:%-%;
ANSWER:FACTOR(ANSWER);
/* NOTE ABOVE IS A COMPLETE SQUARE */
```

APPENDIX V

```
/*QUATERNION MULTIPLICATION EXAMPLE */
/*ANALOG OF CAYLEY-KLEIN RESULT */
I:IDENT(4);
/* NOW WE DEFINE AN OPERATION SS ON A COLUMN MATIX BASED ON ANALOG
OF CAYLEY KLEIN DECOMPOSITION */
SS(CC):=MATRIX([CC[1,1],-CC[2,1],-CC[3,1],-CC[4,1]],
[CC[2,1],CC[1,1],-CC[4,1],CC[3,1]],
[CC[3,1],CC[4,1],CC[1,1],-CC[2,1]],
[CC[4,1],-CC[3,1],CC[2,1],CC[1,1]]);
/* DEFINE AN INVERSE OPERATION */
INV(CC):=1/(CC.CC)*MATRIX([CC[1,1]],[-CC[2,1]],[-CC[3,1]],[-CC[4,1]]);
/* NOW THE BRANDS'S THEOREM ON QUATERNION FORMULATED IN MATRIX FORM */
RHO:MATRIX([R0],[R1],[R2],[R3]);
GAM:MATRIX([Q0],[Q1],[Q2],[Q3]);
/*NOW DEFINE  QUATERNION PRODUCT */
APROD(R,Q):=SS(R).Q;
A:MATRIX([A0],[A1],[A2],[A3]);
RATSIMP(APROD(INV(A),A));
ANSWER:RATSIMP(APROD(GAM,APROD(RHO,INV(GAM))));
EQ1:ANSWER[1,1];
EQ2:ANSWER[2,1];
EQ3:ANSWER[3,1];
EQ4:ANSWER[4,1];
/* NOW GENERATE COEFFICIENT MATRIX FOR RHO */
COEFMATRIX([EQ1,EQ2,EQ3,EQ4],[R0,R1,R2,R3]);
/* THE ABOVE IS SAME AS EXTENDED EULER PARAMETER MATRIX */
```

APPENDIX VI

```
/* THE BASIC DECOMPOSITION FOR EULER PARAMETER */
/*TEST OUT (C0*I+S)X(C0*I-S)       */
II:IDENT(4);
SS:MATRIX([0,-CC1,-CC2,-CC3],
[CC1,0,-CC3,CC2],
[CC2,CC3,0,-CC1],
[CC3,-CC2,CC1,0]);
Q:MATRIX([Y,Z,0,-X],
[Z,-Y,X,0],
[0,X,Y,Z],
[-X,0,Z,-Y]);
EQ1:EXPAND((CC0*II+SS).Q.(CC0*II-SS));
T1:EQ1[2,3];
T2:EQ1[1,1];
T3:EQ1[4,3];
ANSWER:COEFMATRIX([T1,T2,T3],[X,Y,Z]);
/*ABOVE IS SAME A S EQUATION 11 */
```

APPENDIX VII

```
/*.....ALGEBRA FOR QUATERNIONS FROM YANG'S PAPER... */
NNPRED(N):=IS(N>=2);
MATCHDECLARE(NN,NNPRED);
TELLSIMPAFTER(EP^NN,0);
/*ABOVE WILL ELIMINATE EP**2 TERMS */
AL12H:AL12+EP*A12;
AL23H:AL23+EP*A23;
AL34H:AL34+EP*A34;
AL41H:AL41+EP*A41;
TH1H:TH1+EP*S1;
TH2H:TH2+EP*S2;
TH3H:TH3+EP*S3;
TH4H:TH4+EP*S4;
SAL12H: EXPAND(TAYLOR(SIN(AL12H),EP,0,1));
SAL23H: EXPAND(TAYLOR(SIN(AL23H),EP,0,1));
SAL34H: EXPAND(TAYLOR(SIN(AL34H),EP,0,1));
SAL41H: EXPAND(TAYLOR(SIN(AL4*H),EP,0,1));
STH1H:  EXPAND(TAYLOR(SIN(TH1H),EP,0,1));
STH2H:  EXPAND(TAYLOR(SIN(TH2H),EP,0,1));
STH3H:  EXPAND(TAYLOR(SIN(TH3H),EP,0,1));
STH4H:  EXPAND(TAYLOR(SIN(TH4H),EP,0,1));
CAL12H:EXPAND(TAYLOR(COS(AL12H),EP,0,1));
CAL23H:EXPAND(TAYL-OR(COS(AL23H),EP,0,1));
CAL34H:EXPAND(TAYLOR(COS(AL34H),EP,0,1));
CAL41H:EXPAND(TAYLOR(COS(AL41H),EP,0,1));
CTH1H: EXPAND(TAYLOR(COS(TH1H),EP,0,1));
CTH2H: EXPAND(TAYLOR(COS(TH2H),EP,0,1));
CTH3H: EXPAND(TAYLOR(COS(TH3H),EP,0,1));
CTH4H: EXPAND(TAYLOR(COS(TH4H),EP,0,1));
AATHIH:SAL12H*SAL34H*STH1H;
BBTH1H:-SAL34H*(SAL41H*CAL12H+CAL41H*SAL12H*CTH1H);
CCTH1H:CAL23H-CAL34H*(CAL41H*CAL12H-SAL41H*SAL12H*CTH1H);
EQ1:AATH1H*STH4H+BBTH1H*CTH4H-CCTH1H;
PRIMARY:EV(EQ1,EP=0);
DUAL:RATCOEFF(EQ1,EP);
A:RATCOEFF(PRIMARY,SIN(TH4));
B:RATCOEFF(PRIMARY,COS(TH4));
C:EXPAND(PRIMARY-A*SIN(TH4)-B*COS(TH4));
DUAL1:DUAL-S4*(A*COS(TH4)-B*SIN(TH4));
A0:RATCOEFF(DUAL1,SIN(TH4));
B0:RATCOEFF(DUAL1,COS(TH4));
CC0:EXPAND(DUAL1-A0*SIN(TH4)-B0*COS(TH4));
CC0:RATSIMP(CC0);
```

APPENDIX VIII

```
/*DYNAMICAL EQUATIONS FOR STANFORD MANIPULATOR*/
MATCHDECLARE(A,TRUE);
DEPENDS([Q1,Q2,Q3,Q4,Q5,Q6],T);
DEPENDS(U,T);
/*TRIGNOMETRIC SIMPLIFICATIONS */
TELLSIMP(SIN(A)^2,1-COS(A)^2);
S1:SIN(Q1);
CC1:COS(Q1);
S2:SIN(Q2);
CC2:COS(Q2);
S3:SIN(Q3);
CC3:COS(Q3);
/* EXPRESS LOCAL ANGULAR VELOCITIES IN TERMS OF GENERALIZED ONES*/
QD1:1/S2*(U[1]*S3-U[3]*CC3);
QD2:U[1]*CC3+U[3]*S3;
QD3:U[2]+(U[3]*CC3-U[1]*S3)*CC2/S2;
QD4:U[4];
QD5:U[5];
QD6:U[6];
GRADEF(Q1,T,QD1);
GRADEF(Q2,T,QD2);
GRADEF(Q3,T,QD3);
GRADEF(Q4,T,QD4);
GRADEF(Q5,T,QD5);
GRADEF(Q6,T,QD6);
/*DEFINE ROTATIONS */
ROTX(Q):=MATRIX([1,0,0],[0,COS(Q),-SIN(Q)],[0,SIN(Q),COS(Q)]);
ROTY(Q):=MATRIX([COS(Q),0,SIN(Q)],[0,1,0],[-SIN(Q),0,COS(Q)]);
ROTZ(Q):=MATRIX([COS(Q),-SIN(Q),0],[SIN(Q),COS(Q),0],[0,0,1]);
W1:MATRIX([0,QD1,0]);
W2:MATRIX([QD2,0,0]);
W3:MATRIX([0,QD3,0]);
W4:MATRIX([QD4,0,0]);
W5:MATRIX([0,QD5,0]);
W6:MATRIX([0,QD6,0]);
/*SET UP ROTATION MATRICES */
R1:ROTY(Q1);
R2:ROTX(Q2);
R3:ROTY(Q3);
R4:ROTX(Q4);
R5:ROTY(Q5);
/*STAGE 1. SET UP AGNGULAR VELOCITIES */
WA:EXPAND(W1 . R1);
WB:EXPAND(W1 . R1 . R2+W2 . R2);
WC:WB;
WD:EXPAND(W1 . R1 . R2 . R3+W2 . R2 . R3+W3 . R3);
WE:EXPAND(W1 . R1 . R2 . R3 . R4+W2 . R2 . R3 . R4+W3 . R3 . R4+W4 . R4);
WF:EXPAND(W1 R1 R2 R3 R4 R5+W2 R2 R3 R4 R5+W3 R3 R4 R5+W4 R4 R5+W5 R5);
/* SET UP BASE VECTORS AND CROSS PRODUCT */
AA:MATRIX([AA1,AA2,AA3]);
BB:MATRIX([BB1,BB2,BB3]);
CC:MATRIX([CC1,CC2,CC3]);
DD:MATRIX([DD1,DD2,DD3]);
EE:MATRIX([EE1,EE2,EE3]);
FF:MATRIX([FF1,FF2,FF3]);
CROSS(A,B,BASE):=BLOCK([],MATRIX([A[1,2]*B[1,3]-A[1,3]*B[1,2],
-(A[1,1]*B[1,3]-A[1,3]*B[1,1]),A[1,1]*B[1,2]-A[1,2]*B[1,1]]),
/* LENTHE VECTORS FOR VELOCITIES */
VECL1:MATRIX([L1,0,0]);
VECQ6:MATRIX([0,Q6,0]);
VECL5:MATRIX([0,L5,0]);
VECL2:MATRIX([0,L2,0]);
VECL6:MATRIX([0,L6,0]);
VECL3:MATRIX([0,L3,0]);
VECL4:MATRIX([0,L4,0]);
/*STAGE 2. SET UP LINEAR VELOCITIES */
VA:MATRIX([0,0,0]);
RB:MATRIX([L1,L4,0]);
VB:CROSS(WA,RB,AA);
RC:VECQ6+VECL1 . R2;
VC:CROSS(WB,RC,CC);
/*ADD LINEAR COMPONENT */
VC:VC+W6;
RD:VECL1 . R2+VECQ6+VECL5;
VD:CROSS(WB,RD,CC);
/*ADD LINEAR COMPONENT */
VD:VD+W6;
/*FOR VE START WITH VELOCITY OF C */
VE:EXPAND(VC . R3 . R4)+CROSS(WE-W4,VECL2 . R3 . R4,FF)+CROSS(WE,VECL6,FF);
/*REPLACE L6 BY L3 IN ABOVE FOR VELOCITY OF E*/
VE:RATSUBST(L3,L6,%);
/*STAGE 3. SET UP PARTIAL ANGULAR VELOCITIES */
FOR I TERU 6 DO LDISPLAY(WAR[I]:RATCOEFF(WA,U[I]));
FOR I THRU 6 DO WBR[I]:RATCOEFF(WB,U[I]);
FOR I THRU 6 DO WCR[I]:RATCOEFF(WC,U[I]);
FOR I THRU 6 DO WDR[I]:RATCOEFF(WD,U[I]);
FOR I THRU 6 DO WER[I]:RATCOEFF(WE,U[I]);
FOR I THRU 6 DO WFR[I]:RATCOEFF(WF,U[I]);
/*STAGE 4. SET UP PARTIAL LINEAR VELOCITIES */
FOR I THRU 6 DO VAR[I]:RATCOEFF(VA,U[I]);
FOR I THRU 6 DO VBR[I]:RATCOEFF(VB,U[I]);
FOR I THRU 6 DO VCR[I]:RATCOEFF(VC,U[I]);
FOR I THRU 6 DO VDR[I]:RATCOEFF(VD,U[I]);
FOR I THRU 6 DO VER[I]:RATCOEFF(VE,U[I]);
FOR I THRU 6 DO VFR[I]:RATCOEFF(VF,U[I]);
/*STAGE 5. FIND THE ANGULAR ACCELERATIONS */
ALPHAA:DIFF(WA,T)$
ALPHAB:DIFF(WB,T)$
ALPHAC:DIFF(WC,T)$
ALPHAD:DIFF(WD,T)$
ALPHAE:DIFF(WE,T)$
ALPHAF:DIFF(WF,T)$
/*STAGE 6. FIND THE LINEAR ACCELERATION */
ACCA:DIFF(VA,T);
ACCB:DIFF(VB,T)+CROSS(WA,VB,AA)$
ACCC:DIFF(VC,T)+CROSS(WB,VC,BB)$
ACCD:DIFF(VD,T)+CROSS(WC,VD,CC)$
ACCE:DIFF(VE,T)+CROSS(WF,VE,FF)$
ACCF:DIFF(VF,T)+CROSS(WE,VF,FF)$
/*STAGE 7. MOMENTS OF INERTIA */
IA:MATRIX([IA1,IA2,IA3]);
IB:MATRIX([IB1,IB2,IB3]);
```

```
IC:MATRIX([IC1,IC2,IC3]);
ID:MATRIX([ID1,ID2,ID3]);
IE:MATRIX([IE1,IE2,IE3]);
IIF:MATRIX([IIF1,IIF2,IIF3]);
/*STAGE 8,9,10. DEFINE TORQUES,REACTION,AND GENERALIZED FORCES FOR A,B,C,D,E,F*/
TAS:-ALPHAA*IA-CROSS(WA,IA*WA,AA)$
RAS:-MA*ACCA$
FOR I THRU 6 DO LDISPLAY(KAS[I] WAR[I] . TAS+VAR[I] RAS)$
TBS:-ALPHAB*IB-CROSS(WB,IB*WB,BB)$
RBS:-MB*ACCB$
FOR I THRU 6 DO KBS[I] WBR[I] . TBS+VBR[I] RBS$
TCS:-ALPHAC*IC-CROSS(WC,IC*WC,CC)$
RCS:-MC*ACCC$
FOR I THRU 6 DO KCS[I].WCR[I] TCS+VCR[I] . RCS$
TDS:-ALPHAD*ID-CROSS(WD,ID*WD,DD)$
RDS:-MD*ACCD$
FOR I THRU 6 DO KDS[I] WDR[I] . TDS+VDR[I] . RDS$
TES:-ALPHAE*IE-CROSS(WE,IE*WE,EE)$
RES:-ME*ACCE$
FOR I THRU 6 DO KES[I] WER[I] . TES+VER[I] RES$
TFS:-ALPHAF*IF-CROSS(WF,IF*WF,FF)$
RFS:-MF*ACCF$
FOR I THRU 6 DO KFS[I] WFR[I] TFS+VFR[I] RFS$
/*SUM ALL CORRESPONDING GENERALIZED FORCES */
KK1:KAS[1]+KBS[1]+KCS[1]+KDS[1]+KES[1]+KFS[1]$
KK2:KAS[2]+KBS[2]+KCS[2]+KDS[2]+KES[2]+KFS[2]$
KK3:KAS[3]+KBS[3]+KCS[3]+KDS[3]+KFS[3]+KFS[3]$
KK4:KAS[4]+KBS[4]+KCS[4]+KDS[4]+KES[4]+KFS[4]$
KK5:KAS[5]+KBS[5]+KCS[5]+KDS[5]+KES[5]+KFS[5]$
KK6:KAS[6]+KBS[6]+KCS[6]+KDS[6]+KES[6]+KFS[6]$
/*STAGE 10. SET UP ACTIVE FORCES */
GA:MATRIX([0,-G*MA,0]),
GB:MATRIX([0,-G*MB,0]),
GC:-G*MC*MATRIX([0,CC2,-S2]),
GD:-G*MD*MATRIX([0,CC2,-S2]),
GE:-G*ME*MATRIX([0,1,0]) . R1 R2 . R3 R4;
GF:-G*MF*MATRIX([0,1,0]) . R1 R2 . R3 . R4,
TNA:MATRIX([0,TAU1,0]),
TBA:MATRIX([TAU2,0,0]),
TCB:MATRIX([0,-SIGMA,0]),
TDC:MATRIX([0,TAU3,0]),
TED:MATRIX([TAU4,0,0]),
TFE:MATRIX([0,TAU5,0]),
RNA:MATRIX([0,0,0]),
/*SET UP GENERALIZED ACTIVE FORCES */
SPECIAL2[R]:=BLOCK(IF R = 6 THEN -SIGMA ELSE 0),
KTOTALR[R]:=SPECIAL2[R]+WAR[R] . TNA+(WAR[R] . R2-WBR[R]) TBA
+(WCR[R] . R3-WDR[R]) . TDC R3+(WDR[R] R4-WER[R]) . TED . R4
+(WER[R] R5-WFR[R]) . TFE . R5+VBR[R] . GB+VCR[k] . GC+VDR[R] . GD+VER[R]
KEEPFLOAT TRUE,
/*NUMERICAL EXAMPLE WITH VALUES GIVEN IN REF [9]*/
G:9.8, L1:1, L2:.6, L3:.2, L4:.1, L5:0.7, L6:0.06; MA:9, MB:6, MC:4; MD:1,
ME:0.6, MF:0.5, IA1:0.01, IA2:0.02, IA3:0.01, IB1:0.06, IB2:0.01, IB3:0.05,
IC1:0.4, IC2:0.01, IC3:0.4, ID1:0.0005, ID2:0.001, ID3:0.001; IE1:0.0005;
IE2:0.0002; IE3:0.0005, IIF1:0.001, IIF2:0.002, IIF3:0.003,
FT:(IFT (T-10/(2*%PI)*SIN(2*%PI*T/10))*(%PI/180),NUMER),
TQ1:60/10*FT,
TQ2:%PI/2+(60-90)/10*FT,
TQ3:TQ1,
TQ4:TQ1,
TQ5:TQ1,
TQ6:1/10,
U[1]:DIFF(TQ1,T)*SIN(TQ2)*SIN(TQ3)+DIFF(TQ2,T)*COS(TQ3),
U[2]:DIFF(TQ1,T)*COS(TQ2)+DIFF(TQ3,T),
U[3]:-DIFF(TQ1,T)*SIN(TQ2)*COS(TQ3)+DIFF(TQ2,T)*SIN(TQ3);
U[4]:DIFF(TQ4,T),
U[5]:DIFF(TQ5,T),
U[6]:DIFF(TQ6,T),
Q1:TQ1,
Q2:TQ2;
Q3:TQ3;
Q4:TQ4;
Q5:TQ5;
Q6:TQ6,
/* NOW WE PLOT RESULTS AND COMPARE WITH REF [9] */
FINAL6:KTOTALR[6]+SIGMA+KK6,
FINAL6:EV(FINAL6,DIFF)$
EQUALSCALE:FALSE,
PLOTNUM:10;
PLOT(FINAL6,T,0,10,"PLOT OF SIGMA"),
FINAL5:EV(KK5,DIFF)$
PLOT(FINAL5,T,0,10,"PLOT OF TAU 5"),
FINAL4:EV(KK4+KTOTALR[4]+TAU4,DIFF)$
PLOT(FINAL4,T,0,10,"PLOT OF TAU 4"),
FINAL3:EV(KK2+KTOTALR[2]+TAU3,DIFF)$
PLOT(FINAL3,T,0,10,"PLOT OF TAU3"),
/*TRY TO SIMPLIFY AND COLLECT TERMS X I J IN EQUATION OF MOTION*/
/*FIRST DELETE NUMERICAL VALUES */
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXA[I,J]:RATCOEFF(KAS[I],DIFF(U[J],T))),
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXB[I,J]:RATCOEFF(KBS[I],DIFF(U[J],T))),
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXC[I,J]:RATCOEFF(KCS[I],DIFF(U[J],T))),
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXD[I,J]:RATCOEFF(KDS[I],DIFF(U[J],T))),
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXE[I,J]:RATCOEFF(KES[I],DIFF(U[J],T))),
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXF[I,J]:RATCOEFF(KFS[I],DIFF(U[J],T))),
FOR I:1 QRU 6 DO (FOR J:1 QRU 6 DO XXX[I,J]:RATSIMP(XXA[I,J]+XXB[I,J]+
XXC[I,J]+XXD[I,J]+XXE[I,J]+XXF[I,J])),
FOR I:1 THRU 6 DO (FOR J:1 THRU 6 DO LDISPLAY(XXX[I,J])),
/*COMPARE ABOVE X(I,J) WITH THOSE OF REF [9]*/
```

APPENDIX IX

```
                    */
/*     CARTESIAN DIV AND CURL DEFINITION
       UNIT VECTORS ARE B1 B2 B3    SEE LEVINSON     */
/*          DEFINE DOT AND CROSS PRODUCTS   */
DOT(V1,V2) = BLOCK([P,PP],
FOR I:1 THRU 3 DO P[I] RATCOEFF(V1,B[I]),
FOR I:1 THRU 3 DO PP[I] RATCOEFF(V2,B[I]),
P[4] SUM(P[I]*PP[I],I,1,3),
RETURN(P[4]))$
CROSS(V1,V2) = BLOCK([P,PP,PPP],
FOR I:1 THRU 3 DO P[I] RATCOEFF(V1,B[I]),
FOR I:1 THRU 3 DO PP[I] RATCOEFF(V2,B[I]),
PPP[1] (P[2]*PP[3]-P[3]*PP[2]),
PPP[2] (-P[1]*PP[3] + P[3]*PP[1]),
PPP[3] (P[1]*PP[2]-P[2]*PP[1])
PPP[4] B[1]*PPP[1] + B[2]*PPP[2] + B[3]*PPP[3],
RETURN(PPP[4]))$
/*        NOW WE INPUT EQUATIONS FROM LEVINSON'S PAPER  */
DEPENDS(U,T),
DEPENDS(Q,T),
R[2] COS(Q)*B[2] + SIN(Q)*B[3],
R[3] -SIN(Q)*B[2] + COS(Q)*B[3],
WB U[1]*B[1] + U[2]*B[2] + U[3]*B[3],
DERIVABBREV TRUE,
U[4] DIFF(Q,T),
WR (U[1] + U[4])*B[1] + U[2]*B[2] + U[3]*B[3],
ALPB DIFF(U[1],T)*B[1] + DIFF(U[2],T)*B[2] + DIFF(U[3],T)*B[3],
ALPR DIFF(WR,T) + CROSS(WB,WR)
PPBS B1*B[1] + B2*B[2] + B3*B[3],
PPRS R1*B[1] + R2*R[2] + R3*R[3],
PRSBS PPBS-PPRS,
VBS U[5]*B[1] + U[6]*B[2] + U[7]*B[3],
VRS VBS + DIFF(PRSBS,T) + CROSS(WB,PRSBS),
ABS DIFF(VBS,T) + CROSS(WB,VBS),
ARS DIFF(VRS,T) + CROSS(WB,VRS),
IBBSWB BET1*B[1]*DOT(B[1],WB) + BET2*B[2]*DOT(B[2],WB) + BET3*B[3]*DOT(B[3],WB),
IRRSWR RHO1*B[1]*DOT(B[1],WR) + RHO2*B[2]*DOT(B[2],WR) + RHO3*B[3]*DOT(B[3],WR),
IBBSALPB BET1*B[1]*DOT(B[1],ALPB) + BET2*B[2]*DOT(B[2],ALPB) + BET3*B[3]*DOT(B[3],ALPB),
IRRSALPR RHO1*B[1]*DOT(B[1],ALPR) + RHO2*B[2]*DOT(B[2],ALPR) + RHO3*B[3]*DOT(B[3],ALPR),
FSB -MB*ABS,
FSR -MR*ARS,
TSB CROSS(IBBSWB,WB)-IBBSALPB,
TSR CROSS(IRRSWR,WR)-IRRSALPR,
FB F1*B[1] + F2*B[2] + F3*B[3],
TB T1*B[1] + T2*B[2] + T3*B[3],
F[R] = DOT(DIFF(VBS,U[R]),FB) + DOT(DIFF(WB,U[R]),TB),
FS[R] = DOT(DIFF(VBS,U[R]),FSB) + DOT(DIFF(VRS,U[R]),FSR)
 + DOT(DIFF(WB,U[R]),TSB) + DOT(DIFF(WR,U[R]),TSR),
EQ[R] = F[R] + FS[R],
EQ[I],
FOR I:1 THRU 7 DO LDISPLAY(X[I,J] RATCOEFF(EQ[I],DIFF(U[J],T)))
```

IMPLICIT EQUATION FOR A PARAMETRIC SURFACE BY GROEBNER BASIS

Dennis S. Arnon
Computer Science Department
Purdue University
West Lafayette, Indiana 47907

Thomas W. Sederberg
Civil Engineering Department
Brigham Young University
Provo, Utah 84602

Extended Abstract

Polynomial parametric curves and surfaces are widely used in computer graphics and computer-aided design. A parametric curve (in the plane) is the set of points $<x, y>$ defined by a system of equations

$$x = f(t) \qquad y = g(t),$$

f and g polynomials. A parametric surface is the set of $<x, y, z>$ defined by a system

$$x = u(s, t) \qquad y = v(s, t) \qquad z = w(s, t),$$

u, v, and w polynomials. In each case we have a *(parametric) hypersurface:* the number of parameters is one less than the number of variables. We

assume that all polynomials have real coefficients, and that we are only interested in real values of parameters and variables.

One frequently needs to intersect two parametric hypersurfaces. But whereas two plane curves intersect in some finite number of isolated points, two surfaces meet in a space curve comprised of finitely many components. Constructing useful descriptions of such intersection sets, suitable for subsequent manipulation, is a nontrivial task [Req83]. A closed-form expression is desirable. We describe how Gröbner bases can be used to achieve this goal.

We begin by introducing a second class of hypersurfaces. We call the set of points in $\mathbb{R}^n$ satisfying an equation

$$F(x_1, \ldots, x_n) = 0,$$

F a polynomial, an *(implicit) hypersurface*. The reader will see that this is merely a different term for what is usually called an "algebraic" hypersurface. As before, we assume that polynomials have real coefficients, and we are only interested in real values of the variables.

We will make use of the following straightforward observation. Suppose we have two hypersurfaces in $\mathbb{R}^n$, one parametric and the other implicit. Then if we substitute the parametric equations of the one, into the implicit equation of the other, we get a closed-form expression in parameter space for the intersection set. For example, if $x = u(s,t)$,

$y = v(s,t)$, $z = w(s,t)$ is a parametric surface, and $U(x,y,z) = 0$ an implicit surface, then $U(u(s,t), v(s,t), w(s,t)) = \bar{U}(s,t)$ is a closed-form expression for the intersection curve.

Suppose now we have two parametric hypersurfaces that we want to intersect. If we could construct an implicit equation for one of them, then we could intersect them as per the observation. Methods of performing this "implicitization", for particular classes of parametric curves and surfaces, have recently been developed by one of us [Sed]. Here we use Gröbner bases.

Classical resultant theory (see e.g. [Col71]) suggests that in general, an implicit equation exists. Consider, for example, a parametric surface in $\mathbb{R}^3$:

$$A(x,y,z,s,t) = x - u(s,t) = 0$$

$$B(x,y,z,s,t) = y - v(s,t) = 0$$

$$C(x,y,z,s,t) = z - w(s,t) = 0.$$

Let $S(x,y,z,s)$ be the resultant of A and B, and $T(x,y,z,s)$ the resultant of B and C. Let $R(x,y,z)$ be the resultant of S and T. Let $< \bar{x}, \bar{y}, \bar{z} >$ be a point on the surface. Then there exist $\bar{s}$ and $\bar{t}$ such that $< \bar{x}, \bar{y}, \bar{z}, \bar{s}, \bar{t} >$ is a common root of A, B, and C, and hence $R(\bar{x}, \bar{y}, \bar{z}) = 0$. It does not follow, however, that $R = 0$ is an implicit equation for the surface. There may exist $< \bar{x}, \bar{y}, \bar{z} >$ which are not on the surface, but for which

$R(\bar{x}, \bar{y}, \bar{z}) = 0$. Typically, however, either R or one of its factors gives an implicit equation. This line of argument can be generalized to arbitrary n.

Gröbner bases enable us to actually construct implicit equations. We review some relevant facts. Let K be a field, let x denote a k-tuple of variables $x_1, \ldots, x_k$, and let y denote an m-tuple of variables $y_1, \ldots, y_m$. For any ideal I in $K[x, y]$, the *contraction* of I with $K[x]$ is $I \cap K[x]$. It has been shown (see e.g. [Zac84]) that if G is a Gröbner basis for I, then $G \cap K[x]$ is a Gröbner basis for the contraction of I with $K[x]$. Suppose $P = \{P_1(x), \ldots, P_m(x)\}$ is a collection of polynomials in $K[x]$. The set J of all $Q(y_1, \ldots, y_m)$ in $K[y]$ such that $Q(P_1(x), \ldots, P_m(x)) = 0$, is an ideal in $K[y]$ which we call the ideal of *polynomial relations* of P. Clearly J is the contraction of the ideal $L = (y_1 - P_1, \ldots, y_m - P_m) K[x, y]$ with $K[y]$. By the results we have cited, if we compute a Gröbner basis for L and retain exactly those elements which involve only the variables $y_1, \ldots, y_m$, then we have a basis for J.

Suppose we are given a parametric hypersurface in R_n. We may write it as a set of polynomials:

$$y_1 - P_1(x_1, \ldots, x_{n-1})$$

$$y_2 - P_2(x_1, \ldots, x_{n-1})$$

$$y_n - P_n(x_1, \ldots, x_{n-1}) .$$

We construct a Gröbner basis G for the ideal generated by these polynomials; the subset G_y of G, consisting of the elements involving only $y_1, \ldots, y_n$, is a basis for the ideal of polynomial relations of $\{P_1, \ldots, P_n\}$. Clearly, each element of G_y vanishes at every point of the hypersurface. Typically, one of the elements of G_y is actually an implicit equation for the hypersurface.

References

Col71.
GE Collins, "The calculation of multivariate polynomial resultants," *J. Assoc. Comp. Mach.* **18**, pp. 515-532 (1971).

Req83.
AAG Requicha and H Voelcker, "Solid Modeling: Current Status and Research Directions," *IEEE Computer Graphics and Applications* **3**, pp. 25-38 (October 1983).

Sed. T Sederberg, "Ray tracing of Steiner patches," in *Proceedings of SIGGRAPH '84, July 23-27, 1984, Minneapolis, Michigan,*, Assoc. Comp. Mach.

Zac84.
G Zacharias, *Quick explanation of definitions and applications of Gröbner bases*, File MC; GZ; GROB BASIS on MIT-MC(1984).

COMPUTING THE GRÖBNER BASIS OF AN IDEAL IN POLYNOMIAL RINGS OVER THE INTEGERS

Abdelilah Kandri-Rody
Dept. of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, New York 12181
and University Mohammed – V
Rabat, Morocco

Deepak Kapur*
Computer Science Branch
General Electric Company
Corporate Research and Development
Schenectady, New York 12345

ABSTRACT

An algorithm is developed for computing a Gröbner basis of an ideal in polynomial rings over integers. The algorithm is a natural extension of Buchberger's algorithm for computing a Gröbner basis of an ideal in polynomial rings over a field. The algorithm is implemented in ALDES and LISP and the implementation is discussed with a number of examples given. The uniqueness of the Gröbner basis of a polynomial ideal over the integers is shown.

1. INTRODUCTION

Buchberger [1,3,4] developed an algorithm for computing the Gröbner Basis of an ideal in polynomial rings over a field. This algorithm takes an ideal specified by a finite set of polynomials as its input and produces another finite basis of the ideal which can be used to simplify polynomials such that every polynomial in the ideal simplifies to 0 and every polynomial in the polynomial ring simplifies to a unique normal form. The algorithm has been found useful in algebraic simplification [5].

In this paper, we develop an algorithm to compute the Gröbner basis of an ideal in polynomial rings over the integers. The algorithm is a natural extension of Buchberger's algorithm. New polynomials to complete the basis are computed between pairs of polynomials in the basis, the reduction process is simple and, the minimal Gröbner basis thus obtained is unique. An implementation of an efficient version of this algorithm in ALDES and LISP, patterned after Huet's version [7] of the Knuth-Bendix completion procedure, is also discussed.

* Partially supported by NSF grant MCS-82-11621.

1.1 Related Work

According to Lauer [19], Szekeres [22] showed the existence of a canonical basis for ideals over a Euclidean ring and Shtokhamer developed a generalization of the construction suggested by Szekeres to define a canonical basis over a principal ideal domain. Schaller [21] proposed an algorithm to compute a Gröbner basis of an ideal over polynomials over a principal ideal domain; at the same time, Zacharias [24] developed a similar algorithm for an ideal in a polynomial ring in which the ideal membership and basis problems for homogeneous linear equations are solvable. In both Schaller's and Zacharias's approaches, new polynomials needed for a complete basis must be computed for every finite set of polynomials in the input basis; this computation needs solving homogeneous linear equations. The reduction process in their approach needs a computation of extended greatest common divisor over many head-coefficients in the basis; their algorithm gives a Gröbner basis which is not necessarily unique. In contrast, our approach is simpler; the rewriting relation induced by a polynomial is defined in a natural way.

2. WELL-FOUNDED ORDERING ON POLYNOMIALS

Let $Z[X_1, ..., X_n]$ be the ring of polynomials with indeterminates $X_1, ..., X_n$ over the ring of integers Z; it is assumed that $X_1 < X_2 < \cdots < X_n$. A *term* is any product $\prod_{i=1}^{n} X_i^{k_i}$, where $k_i \geqslant 0$; the degree of a term is $\sum_{i=1}^{n} k_i$. A *monomial* is a term multiplied by a nonzero coefficient from Z. A *polynomial* is a sum of monomials; such a polynomial is said to be in *sum of products form*, abbreviated as SPF (this form of polynomials has also been called distributive normal form in the literature). If no term appears more than once in a polynomial in SPF, it is said to be in *simplified sum of products form*, abbreviated as SSPF. An arbitrary polynomial which is not in SSPF can be transformed into an equivalent polynomial in SSPF using the rules of the polynomial ring. Henceforth, we will assume polynomials to be in SSPF.

The ordering on terms is defined using the degree of a term and terms of the same degree are ordered lexicographically (this ordering is the same as the one used by Buchberger in [1,3]). Terms $t_1 = \prod_{1}^{n} x_i^{k_i} < t_2 = \prod_{1}^{n} x_i^{j_i}$ if and only if (1) the degree of t_1 < the degree of t_2, or (2) the degree of t_1 = degree of t_2 and there exists an $i \geqslant 1$, such that $k_i < j_i$ and for for each $1 \leqslant i' < i$, $k_{i'} = j_{i'}$. This ordering is a total ordering and is well-founded. Another total well-founded ordering, for example, is the pure lexicographic ordering on terms based on a total ordering on indeterminates in which the degree of terms is not considered. The results of the paper hold for this total ordering also.

Let c and c' be two integers. We say that c is less than c', written as $c << c'$, if and only if $|c| < |c'|$ or ($|c| = |c'|$, c is positive and c' is negative). For example, $2 << -2$, $2 << 3$, $2 << -3$, as well as $-2 << -3$. The ordering $<<$ on Z is total and well-founded.

Monomials are ordered using their terms and coefficients: Given two monomials $m_1 = c_1 t_1$ and $m_2 = c_2 t_2$, $m_1 << m_2$ if and only if $t_1 < t_2$, or ($t_1 = t_2$ and $c_1 << c_2$). It is easy to see that the ordering $<<$ on monomials is total and well-founded.

Let $p = m + r$ be a polynomial in SSPF such that the term of the monomial m is greater than those within r; then m is called the *head-monomial* of p, the term of m is called the *head-term* of p and the coefficient of m is called the *head-coefficient* of p. We will call r the *reductum* of p. The ordering $<<$ on monomials can be used to define a ordering $<<$ on polynomials in

the following way: polynomials $p_1 << p_2$ if and only if either (1) $m_1 << m_2$, or (2) $m_1 = m_2$ and $r_1 << r_2$, where m_i and r_i are, respectively, the head-monomial and reductum of p_i, $i = 1, 2$. It is easy to see that the ordering $<<$ on polynomials in $Z[X_1, ..., X_n]$ is total and well-founded.

3. GRÖBNER BASIS OF AN IDEAL

Informally, a finite set B of polynomials, say $\{p_1, \cdots, p_k\}$, in $Z[X_1, ..., X_n]$ is called a *Gröbner basis* for an ideal I generated by B if for any polynomial q, no matter how q is rewritten using the rules corresponding to polynomials in B, the result is always the same, i.e., it is unique [1,3]. An equivalent definition is that for any polynomial p in the ideal I generated by B, $p \rightarrow^* 0$. The Gröbner basis of an ideal generated by a finite set of polynomials is thus like a canonical rewriting system for an equational theory generated by a finite set of axioms. For examples, consider the ideal I generated by $B = \{X\,Y + 1,\ Y^2 + X\}$ in $Z[X, Y]$; $Y - X^2$ is in I but does not reduce to 0, so B is not a Gröbner basis. However, $B' = \{X\,Y + 1,\ Y^2 + X,\ X^2 - Y\}$ is a Gröbner basis.

In order to precisely define a Gröbner basis of an ideal I, it is necessary to define the rewriting relation induced by a polynomial.

3.1 Polynomials as Reduction Rules

Consider a polynomial $P = m_1 + m_2 + \cdots + m_k$ in $Z[X_1, ..., X_n]$ in SSPF such that m_1 is the head-monomial of P. Further, assume that its head term t_1 has a positive coefficient c_1 (i.e., $m_1 = c_1\,t_1$). Then the rewrite rule corresponding to P is as follows:

$$c_1\,t_1 \rightarrow -\,m_2 + \cdots + -\,m_k$$

In case the coefficient c_1 of the head term t_1 in P is negative, P is multiplied by -1 and the result is used as a rewrite rule. In contrast to the rewrite rule for a polynomial over a field, where both the sides of the rule are divided by the head coefficient c_1 as division is defined on coefficients, the whole monomial is the left-hand-side (lhs). For example, the rewrite rule corresponding to $2\,X^2\,Y - Y$ is $2\,X^2\,Y \rightarrow Y$.

A rule $L \rightarrow R$, where $L = c_1\,t_1$ and $c_1 > 0$ rewrites a monomial $c\,t$ to $(c - \epsilon\,c_1)\,t + \epsilon\,\sigma\,R$ where $\epsilon = 1$ if $c > 0$, $\epsilon = -1$ if $c < 0$, if and only if (1) there exists a term σ such that $t = \sigma\,t_1$ and (2) either $c > (c_1 / 2)$ or $c < -(c_1 - 1) / 2$. If $-(c_1 - 1)/2 \leqslant c \leqslant (c_1 / 2)$ or there does not exist any σ such that $t = \sigma\,t_1$, then the monomial $c\,t$ cannot rewritten.

A polynomial Q is rewritten to Q' using the rule $L \rightarrow R$ if and only if (1) $Q = Q_1 + c\,t$ and $c\,t$ is the largest monomial in Q which can be rewritten using the rule, and (2) $Q' = Q_1 + (c - \epsilon\,c_1)\,t + \epsilon\,\sigma\,R$, where $\epsilon = 1$ if $c > 0$, $\epsilon = -1$ otherwise. If there is no monomial in Q which can be rewritten using the rule, then Q is *irreducible* or in *normal form* with respect to the rule. For example, using the rule $2\,X^2\,Y \rightarrow Y$, the polynomial

$$4\,X^3\,Y + 5\,X\,Y^2 - 3\,X^2\,Y \rightarrow 2\,X^3\,Y + X\,Y + 5\,X\,Y^2 - 3\,X^2\,Y \rightarrow 2\,X\,Y + 5\,X\,Y^2 - 3\,X^2\,Y.$$

The result can be further reduced as the monomial $-3\,X^2\,Y$ is reducible:

$$\rightarrow 2\,X\,Y + 5\,X\,Y^2 - X^2\,Y - Y \rightarrow 2\,X\,Y + 5\,X\,Y^2 + X^2\,Y - 2\,Y.$$

We assume that after rewriting by a polynomial, polynomials are always brought back to SSPF, i.e., indeterminates in terms are ordered using the prespecified ordering on indeterminates, equal terms are combined, and terms with zero coefficients are omitted (see also [3]).

Let $T = \{L_1 \rightarrow R_1, ..., L_k \rightarrow R_k\}$ be the rule set corresponding to a basis $B = \{p_1, \cdots, p_k\}$ of an ideal I such that $\{L_i \rightarrow R_i\}$ be the rule corresponding to p_i. Let $\rightarrow$ denote the rewriting relation defined by T.

3.2 Properties of Reduction Relations

We define properties of $\rightarrow$ which are needed for defining a Gröbner basis (an interested reader may want to refer to [5,6] for more details). Let $\rightarrow^*$ be the reflexive and transitive closure of $\rightarrow$ and $\rightarrow^+$ be the transitive closure of $\rightarrow$.

Definition: A relation $\rightarrow$ is *Noetherian* if and only if there does not exist any infinite sequence $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots$.

Definition: Two elements x and y are said to be *joinable* if and only if there exists u such that $x \rightarrow^* u$ and $y \rightarrow^* u$,

Definition: A relation $\rightarrow$ is *confluent* if and only if for all x, y, z, such that $x \rightarrow^* y$ and $x \rightarrow^* z$, y and z are joinable.

Definition: A relation $\rightarrow$ is *canonical* if and only if $\rightarrow$ is Noetherian and confluent.

If the relation $\rightarrow$ is Noetherian, then the test for confluence reduces to a simple local test, called local confluence.

Definition: A relation $\rightarrow$ is *locally confluent* if and only if for each x, y, z, such that $x \rightarrow y$ and $x \rightarrow z$, y and z are joinable.

Theorem 3.1 [Newman]: A Noetherian relation $\rightarrow$ is confluent if and only if $\rightarrow$ is locally confluent.

See [6] for a proof.

3.3 Definition of Gröbner Basis

The Gröbner basis can be defined by requiring that the rewriting relation defined by a basis satisfies certain conditions. It is sufficient to require of a Gröbner basis B that the relation $\rightarrow$ induced by B is confluent. Since we are interested in developing algorithms, we put an additional requirement that $\rightarrow$ be Noetherian.

Definition: A basis B is a *Gröbner basis* if the rewriting relation $\rightarrow$ induced by B is canonical.

In order to develop a Gröbner basis test for polynomial ideals over Z, we first show that $\rightarrow$ is Noetherian using the total well-founded ordering defined on polynomials in Section 2. Then, we develop a test for local confluence and use the above theorem to check whether a basis is a Gröbner basis.

Lemma 3.2: The rewriting relation $\rightarrow$ induced by any finite basis over $Z[X_1, \cdots, X_n]$ is Noetherian.

Proof: Follows from the fact that for any polynomials Q, Q', such that $Q \rightarrow Q'$, $Q' \ll Q$. □

The test for local confluence is developed in a way similar to the approach developed by Buchberger for polynomial ideals over a field [1,3,4,5]. We define *critical pairs* for a pair of polynomials in a basis. Then it is shown that if these critical pairs are trivial, $\rightarrow$ is locally-confluent.

3.4 Critical Pairs

Given two rules $L_1 \rightarrow R_1$ and $L_2 \rightarrow R_2$, where $L_1 = c_1 t_1$ and $L_2 = c_2 t_2$, such that $c_1 \geq c_2 > 0$. Its *critical pair* $<p, q>$ is defined as: $p = (c_1 - c_2)\ lcm(t_1, t_2) + f_2 * R_2$, and $q = f_1 * R_1$, where $f_1 * t_1 = f_2 * t_2 = lcm(t_1, t_2)$. Polynomials p and q are obtained from the superposition $c_1\ lcm(t_1, t_2)$ by applying $L_2 \rightarrow R_2$ and $L_1 \rightarrow R_1$, respectively.

The above definition of critical pairs is a generalization of the definition used by Buchberger [1,3,4] for a field. In that case, since c_1 and c_2 are 1, the above definition reduces to taking the lcm of the left-hand sides. As in the case of polynomials over rationals, for any pair of polynomials, there is exactly one critical pair.

Example: in $Z[X, Y]$, consider the basis $B_1 = \{ 3X^2 Y \rightarrow Y,\ 10 X Y^2 \rightarrow X\}$. The superposition of the two polynomials is $10 X^2 Y^2$, and the critical pair is $<7 X^2 Y^2 + Y^2,\ X^2>$.

It is easy to see that for the critical pair $<p, q>$ of two polynomials in an ideal, the polynomial $p - q$ is also in the ideal. So, adding the polynomial $p - q$ to the ideal does not change the ideal.

The *S-Polynomial* corresponding to a critical pair $<p, q>$ is the polynomial $p - q$.

Definition: A critical pair $<p, q>$ is *trivial* if and only if its S-polynomial $p - q$ can be reduced to 0 by applying at every step, among all applicable rules, a rule whose left-hand-side has the least coefficient.

The above restriction is necessary because of the way the rewriting relation is defined above. If we do not have this restriction, then there are bases for which all critical pairs are trivial but the bases are not Gröbner bases. For example, consider the basis
$B_2 = \{ 1.\ 6X^2 Y \rightarrow Y,\ 2.\ 2X Y^2 \rightarrow X\}$. Its critical pair is $<4X^2 Y^2 + X^2,\ Y^2>$,
and the two polynomials are joinable if we apply rule 1 first and then rule 2 on the first polynomial.

3.5 Gröbner Basis Test

To test whether a given basis is a Gröbner basis, (1) get the rule set corresponding to the basis, and (2) check whether for each pair of distinct rules, the critical pair $<p, q>$ is trivial. For example, the basis B_1 in the above example is not a Gröbner basis because the two polynomials in the critical pair $< 7 X^2 Y^2 + Y^2,\ X^2>$ do not reduce to the same polynomial. The following theorem serves as the basis of this test.

Theorem 3.3: A basis B of polynomials in $Z[X_1, \cdots, X_n]$ is a Gröbner basis if and only if for every pair of polynomials in B, the critical pair $<p, q>$ is trivial.

Proof: By Lemma 3.2 above, $\rightarrow$ is Noetherian, so it is sufficient to show that the relation $\rightarrow$ induced by B is locally confluent if and only if the critical pairs are trivial.

Explanation of "only if": Since $\rightarrow$ is both locally confluent and Noetherian, $\rightarrow$ is confluent. For every pair $L_1 \rightarrow R_1$ and $L_2 \rightarrow R_2$, where $L_i = c_i t_i$, $i = 1, 2$, and $c_1 \geq c_2$, consider a polynomial $p = c_1 t - f_1 R_1$, where $t = lcm(t_1, t_2) = f_1 R_1 = f_2 R_2$. Using $L_1 \rightarrow R_1$, p reduces to 0, whereas using $L_2 \rightarrow R_2$, p reduces to $(c_1 - c_2)\ t + f_2 R_2 - f_1 R_1$. By confluence of $\rightarrow$, we have $(c_1 - c_2)\ t + f_2 R_2 - f_1 R_1$ reduce to 0 no matter how it is rewritten. Hence, the critical pair of these rules is trivial.

Explanation of "if": Consider a polynomial p which is rewritten in two different ways to q_1 and q_2. Let t' be the term of the largest monomial being rewritten and c be its coefficient in p (this is so, since the rewriting relation is defined to be rewriting the largest monomial, so the case when two different terms are being rewritten does not arise); let $p = p' + c\ t'$. Let

$L_1 \rightarrow R_1$ and $L_2 \rightarrow R_2$ be the two rules being used to rewrite the monomial $c\ t'$; these rules must be distinct as otherwise we have $q_1 = q_2$. Without any loss of generality, we can assume that $c_1 \geqslant c_2$; let $t = lcm(t_1, t_2) = f_1 R_1 = f_2 R_2$, and $t' = \sigma\ t$. Then
$q_1 = p' + (c - \epsilon\ c_1)\ t' + \epsilon\ \sigma\ f_1 R_1$, and
$q_2 = p' + (c - \epsilon\ c_2)\ t' + \epsilon\ \sigma\ f_2 R_2$,
where $\epsilon = 1$ if $c > 0$, and -1 otherwise.

Since the S-polynomial $SP = (c_1 - c_2)\ t + f_2 R_2 - f_1 R_1$ corresponding to the critical pair of the two rules is trivial, q_1 and q_2 are joinable using Lemma 3.4 proved below. This is so because if $\epsilon = 1$, then $q_2 - q_1 = \sigma * SP$, whereas if $\epsilon = -1$, then $q_1 - q_2 = \sigma * SP$. □

Lemma 3.4: For any two polynomials p and q, if $p - q \rightarrow'^{*} 0$, then p and q are joinable.

The relation $\rightarrow'$ is a subset of the relation $\rightarrow$ and is defined as: A monomial $c\ t \rightarrow' q'$ if and only if $c\ t \rightarrow q'$ using a rule $c_1\ t_1 \rightarrow R_1$ in B such that there does not exist any other rule $c_2\ t_2 \rightarrow R_2$ in B which can be applied on $c\ t$ and $c_2 < c_1$. A polynomial $P \rightarrow' Q$ if and only if Q is obtained from P by rewriting the largest monomial under $\rightarrow'$. The definition of a critical pair being trivial uses the rewriting relation $\rightarrow'$.

Before we give a proof of the above lemma, we show the following property of $\rightarrow'$, which is used in the proof of the above lemma.

Lemma 3.5: For any two polynomials p, q such that $p - q \rightarrow' h$ and $h \rightarrow'^{*} 0$, there exist p', q', such that $h = p' - q'$ and $p \rightarrow^{*} p'$ and $q \rightarrow^{*} q'$.

Proof: Suppose that $p - q$ is reduced to h by a rule $c\ t \rightarrow R$. Let $p = R_p + d_p\ t'$, $q = R_q + d_q\ t'$, $d = d_p - d_q$. Then $h = (R_p - R_q) + (d_p - d_q - \epsilon\ c)\ t' + \epsilon\ \sigma\ R$, where $t' = \sigma t$. There are two cases: (1) $d > c/2$ and (2) $d < -(c - 1)/2$.

Case 1: $d > c/2$: This implies $d_p > d_q + c/2$ and $h = (R_p - R_q) + (d_p - d_q - c)\ t' + \sigma\ R$. There are two subcases:

Subcase 1: $d_q \geqslant 0$, which implies $d_p > c/2$, hence d_p is not a remainder of c. So, we reduce p to $p' = R_p + (d_p - c)\ t' + \sigma\ R$. We take $q' = q$.

Subcase 2: $d_q < 0$: If $d_q < -(c - 1)/2$ then we reduce q to $q' = R_q + (d_q + c)\ t' - \sigma\ R$ and we take $p' = p$.

If $0 > d_q \geqslant -(c - 1)/2$, then $d_p > 0$. If $d_p > c/2$ then we take $p' = R_p + (d_p - c)\ t' + \sigma\ R$ and $q' = q$. If $d_p \leqslant c/2$ then $c/2 < (d_p - d_q) \leqslant (c/2 + (c - 1)/2)$ and $d_p - d_q - c$ is a remainder of c. This implies that h cannot be reduced to 0 since in $\rightarrow'$, we require that the rewriting be done using a rule with the smallest head-coefficient. This is a contradiction.

Case 2: $d < -(c - 1)/2$: This implies $d_q > d_p + (c - 1)/2$ and $h = (R_p - R_q) + (d_p - d_q + c)\ t' - \sigma\ R$. There are two subcases:

Subcase 1: $d_p > 0$, which implies $d_q > c/2$, hence d_q is not a remainder of c. So, we reduce q to $q' = R_q + (d_q - c)\ t' + \sigma\ R$. We take $p' = p$.

Subcase 2: $d_p \leqslant 0$: If $d_p < -(c - 1)/2$ then we reduce p to $p' = R_p + (d_p + c)\ t' - \sigma\ R$ and we take $q' = q$.

If $0 \geqslant d_p \geqslant -(c - 1)/2$, then $d_q \geqslant 0$. If $d_q > c/2$ then we take $q' = R_q + (d_q - c)\ t' + \sigma\ R$ and $p' = p$. If $d_q \leqslant c/2$ then $(c - 1)/2 < (d_p - d_q) \leqslant (c/2 + (c - 1)/2)$ and $d_p - d_q + c$ is a remainder of c. This implies that h cannot be reduced to 0 since in $\rightarrow'$, we require that the rewriting be done using a rule with the smallest head-coefficient. This is a contradiction. □

We now give the proof of Lemma 3.4.

Proof: Let $p - q \rightarrow'^{n} 0$. The proof is by induction on n. The basis step of $n = 0$ is trivial, as in that case, $p = q$.

Inductive Step: Assume for $n' < n$, to show for n.

Let $p - q \rightarrow' h \rightarrow'^{*} 0$. By the above lemma, there exists p' and q' such that $h = p' - q'$, $p \rightarrow^{*} p'$ and $q \rightarrow^{*} q'$. By inductive hypothesis on h, p' and q' are joinable. So, p and q are joinable. □

Another way of showing the correctness of the Gröbner basis test is to use the approach developed in [10] to show the relationship between Buchberger's Gröbner basis algorithm for polynomial ideals over a field and the Knuth-Bendix completion procedure. The polynomial simplification process is decomposed into two parts: reduction relation and simplification relation. The rules corresponding to the polynomials in the basis are in the reduction relation, whereas $t + -t = 0$ and $t + 0 = t$ are the only axioms in the simplification relation. In [10], it is shown that the canonicalization of a polynomial obtained after combining reduction and simplification such that simplification is performed before each step of reduction (as is done in Buchberger's Gröbner basis algorithm as well as in the implementation of our Gröbner basis algorithm over Z) is the same as the canonicalization obtained if reduction is completely performed first, followed by simplification at the end. Using this approach, the correctness of the Gröbner basis test is shown by proving a theorem similar to Theorem 4.12 in [10]; an interested reader may look at [9] for details.

4. GRÖBNER BASIS ALGORITHM

If a basis is not a Gröbner basis, it can be completed to get a Gröbner basis. The completion procedure is very much like the Knuth-Bendix completion procedure for term rewriting systems. For every non-trivial critical pair $<p, q>$, add a new rule corresponding to a normal form of the polynomial $p - q$ using the relation $\rightarrow'$, thus generating a new basis for the same ideal. This step is repeated until for each pair of polynomials in the basis, the critical pair is trivial.

Example: In $Z[X, Y]$, consider the basis $B = \{1.\ 2\ X^2\ Y \rightarrow Y$, and $2.\ 3\ X\ Y^2 \rightarrow X\}$, we first add the rule obtained by critical pair of rules 1 and 2: i.e., 3. $X^2\ Y^2 \rightarrow -Y^2 + X^2$. From rules 1 and 3, we get the critical pair $<X^2\ Y^2 - Y^2 + X^2,\ Y^2>$ which gives an additional rule: 4. $3\ Y^2 \rightarrow 2\ X^2$.

Using rule 4, rule 2 can be reduced to $2'.\ 2\ X^3 \rightarrow X$.

The above 4 rules constitute a Gröbner basis because every critical pair is trivial. There is no need to reduce rule 2 using rule 4, however, doing so turns out to be more efficient and also results in a unique Gröbner basis subject to an ordering on indeterminates. This will be discussed later in the paper.

One may think that the Gröbner basis of an ideal I in $Z[X_1, ..., X_n]$ could be obtained by first (1) generating the Gröbner basis B of I using Buchberger's algorithm over rationals and then (2) clearing the denominators of each polynomial in B to get the corresponding polynomials in $Z[X_1, ..., X_n]$.

This construction does not work. As illustrated by the above example, using this construction, we get

$$B' = \{1.\ 2\ X^2\ Y - Y,\ 2'.\ 2X^3 - X,\ 4.\ 3\ Y^2 - 2\ X^2\}$$

which is not a Gröbner basis.

The Gröbner basis algorithm for polynomial ideals over Z is given below. It is patterned after Huet's version [7] of the Knuth-Bendix completion procedure. Note that the basis being used for reduction is always kept in reduced form.

ALGORITHM:
Given F, a finite set of polynomials in $Z[X_1, \ldots, X_n]$,
find G such that ideal(F) = ideal(G) and G is a Gröbner basis.

Initialization: $T_0 := F$; $G_0 := \{\}$; i:=0; m:=0;
LOOP
WHILE $T_i \neq \{\}$ DO
{reduce polynomial: select polynomial P in T_i
(hm, red) := normalize(G_i, P);
;;;; hm and red are head monomial and reductum of normalized P, respectively.}
IF hm = 0 THEN { $T_{i+1} := T_i - \{P\}$; $G_{i+1} := G_i$; i := i+1; }
ELSE { Add new polynomial: let K be the set of labels k of polynomials of G_i
whose head term hm_k is reducible by (hm, red);
$T_{i+1} := (T_i - \{P\}) \cup \{(hm_k, red_k)$, k belongs to K$\}$;
m := m + 1;
$G_{i+1} := \{j: (hm_j, red'_j) \mid j: (hm_j, red_j)$ in G_i and j $\notin$ K$\} \cup \{n: (hm, red)\}$;
;;;; red'_j = normalize($G_i \cup \{m: (hm, red)\}$, red_j)
the new polynomial m:(hm,red) is unmarked;
i := i+1 }
ENDWHILE;
compute critical pairs: IF all polynomials in G_i are marked
THEN EXITLOOP (G_i canonical);
ELSE {select an unmarked polynomial in G_i, say with label k;
T_{i+1} := the set of all critical pairs computed between polynomial k and
any polynomial of G_i of label not greater than k;
$G_{i+1} := G_i$, except that polynomial k is now marked;
i := i+1 }
ENDLOOP.

G := G_i.

Since $Z[X_1, \cdots, X_n]$ is a Noetherian ring, the termination of the process of generating critical pairs and augmenting the basis is guaranteed because of the finite ascending chain condition of properly contained ideals over a Noetherian ring as shown below. The following theorem establishes that a version of the algorithm in which G_i and T_i are not separated and T_i is reduced immediately using G_i, will terminate (in the proof below, $G_i \cup T_i = TG_i$). The proof that the above algorithm terminates is a special case of the following proof because the loop for T_i is always guaranteed to terminate.

Theorem 4.1: The Gröbner Basis Completion Algorithm always terminates.

Proof: Let $TG_i = (B_1^i, \ldots, B_{k_i}^i)$ be the basis at the i-th iteration of the Gröbner basis algorithm. Let $M_j^i = C_j^i H_j^i$ be the head-monomial of the polynomial B_j^i in the basis TG_i, where C_j^i and H_j^i are the head-coefficient and head-term of B_j^i, respectively. Let $B_{k_{i+1}}^{i+1}$ be the polynomial corresponding to the nontrivial critical pair, if any, generated in the i-th iteration to get TG_{i+1} from TG_i.

From TG_i, we construct another basis S_i made only of the head-terms of the polynomials in TG_i, i.e., $S_i = (H_1^i, \ldots, H_{k_i}^i)$.

We first show that the ideal of S_i is a subset of the ideal of S_{i+1}, written as $S_i \subseteq S_{i+1}$

If in the i-th iteration, no nontrivial critical pair is generated, then $TG_{i+1} = TG_i$, so $S_{i+1} = S_i$. Consider the case when a nontrivial critical pair is generated in the i-th iteration. There are two cases:

(1) $B_{k_{i+1}}^{i+1}$ does not reduce the head-monomial of any polynomial in TG_i: then $S_{i+1} = S_i \cup \{H_{k_{i+1}}^{i+1}\}$, implying that $S_i \subseteq S_{i+1}$.

(2) $B_{k_{i+1}}^{i+1}$ reduces the head-monomial of some polynomials in TG_i. So there exist $1 \leqslant j \leqslant k_i$ (could be more than one j) and a term σ, $H_j^i = \sigma\, H_{k_{i+1}}^{i+1}$. In that case, S_i is a subset of S_{i+1} even though H_j^i may not be in the basis of S_{i+1}.

Further, for any i, since TG_i is finite, it is only possible to add finitely many polynomials (bound by the largest head-coefficient in TG_i) to TG_i so that the corresponding S_i remains invariant. This is so because in order to have $S_i = S_{i+1}$, we must have $H_{k_{i+1}}^{i+1}$ so that there exist j, σ, $H_{k_{i+1}}^{i+1} = \sigma\, H_j^i$ and for $B_{k_{i+1}}^{i+1}$ to be irreducible with respect to TG_i,
$-(C_j^i - 1)/2 \leqslant C_{k_{i+1}}^{i+1} \leqslant C_j^i/2$ for all such j.
(As if, there does not exist j, σ, such that $H_{k_i+1}^{i+1} = \sigma\, H_j^i$, then $H_{k_i+1}^{i+1}$ is not in S_i, so S_i is a proper subset of S_{i+1}.)

However, if the process of generating new rules does not terminate, then there is an infinite sequence of ideals $S_1 \subseteq S_2 \subseteq \cdots \subseteq S_i \ldots$, such that there does not exist any s for which $S_s = S_{s+1} = S_{s+2} = \cdots$. This leads to a contradiction because for a Noetherian ring of polynomials, such an infinite ascending chain of ideals does not exist (van der Waerden, Vol. II, p. 117, second formulation). □

Note that if we had assumed that when a new rule is added to TG_i, it is not used to simplify TG_i (as in another version of the Gröbner basis algorithm given in the Appendix, which is patterned after Buchberger's algorithm for polynomial ideals over a field), then $TG_{i+1} = TG_i \cup \{B_{k_{i+1}}^{i+1}\}$, then the above proof simplifies because in that case, there is no need to consider case (2) above.

The above algorithm has been implemented in ALDES (Algorithm Description Language developed by Collins and Loos) as well as in LISP with various strategies for normalizing a polynomial and choosing a polynomial from T_i. We found it is better to choose the smallest polynomial P in T_i. The normalization of P with respect to a basis G_i is done step by step starting with the head-monomial of P, reducing it with respect to G_i, taking the second highest monomial, reducing it and so on.

We also found that for many examples, a reduction check starting with the largest rule in G_i first was more efficient than that starting with the smallest rule. To generate the critical pairs we can choose an unmarked polynomial whose superposition with all the marked one's is the smallest one. If we compute all critical pairs and we reduce the basis at the end, we may run out of all available storage before finding the Gröbner basis. However, if we reduce the basis each time we add a new critical pair, the algorithm works much better. The above implementation takes care of these two problems, i.e., space and time. Different strategies for generating critical pairs discussed in [13] can also be implemented. The next section contains some examples which were run on this implementation.

5. EXAMPLES

We have run many examples (some of which were provided by Lankford) in the ALDES implementation. Some of these examples and their Gröbner bases are reproduced below.

Example 1: In $Z[X, Y, W]$, $X < Y < W$, consider the ideal generated by

(1) $-W + X\ Y^2 + 4\ X^2 + 1$

(2) $Y^2\ W + 2X + 1$

(3) $-X^2\ W + Y^2 + X$

The canonical Gröbner basis is

(1) W^2 - W - 4 Y^2 + 2 X^2 - 3 X

(2) $-W + X\ Y^2 + 4\ X^2 + 1$

(3) $X^2\ W - Y^2 - X$

(4) $Y^2\ W + 2\ X + 1$

(5) $-3\ X\ W - Y^2 + 2\ X^4 + 13\ X^3 + X - 1$

(6) $W + Y^4 + 2\ X^3 - 3\ X^2 - 1$

Example 2: In $Z[X, Y, W]$, $X < Y < W$, consider the ideal generated by

(1) $2\ X^2\ Y^3\ W^5 + 5\ X\ Y^2 + X\ W - 6\ Y$

(2) $X^2 + 2\ X + 1$

(3) $X^2\ Y^2 - 1$

(4) $8\ X\ Y\ W - 8$

(5) $6\ X + 3\ Y + 2\ W$

The canonical Gröbner basis is 1.

Example 3: In $Z[X, Y]$, $X < Y$, consider an ideal I generated by:

(1) $Y^6 + X^4\ Y^4 - X^2\ Y^4 - Y^4 - X^4\ Y^2 + 2\ X^2\ Y^2 + X^6 - X^4$

(2) $2\ X^3\ Y^4 - X\ Y^4 - 2\ X^3\ Y^2 + 2X\ Y^2 + 3\ X^5 - 2\ X^3$

(3) $3\ Y^5 + 2\ X^4\ Y^3 - 2\ X^2\ Y^3 - 2\ Y^3 - X^4\ Y + 2\ X^2\ Y$

The canonical Gröbner basis is:

(1) $4\ Y^4 + 4\ X^4\ Y^2 - 8\ X^2\ Y^2 - 4\ X^6 + 4\ X^4$

(2) $X^2\ Y^4 + 2\ Y^4 + 2\ X^4\ Y^2 - 6\ X^2\ Y^2 - 3\ X^6 + 4\ X^4$

(3) $4\ X\ Y^5 - 8\ X\ Y^3 - 4\ X^5\ Y + 8\ X^3\ Y$

(4) $Y^6 + X^4\ Y^2 - 2\ X^2\ Y^2 - 2\ X^6 + 2\ X^4$

(5) $-X\ Y^4 + 2\ X^3\ Y^2 + 2\ X\ Y^2 + 2\ X^7 - X^5 - 2\ X^3$

(6) $2\ Y^5 + 4\ X^2\ Y^3 - 4\ Y^3 + 4\ X^6\ Y - 6\ X^4\ Y + 4\ X^2\ Y$

(7) $3\ Y^5 + 2\ X^4\ Y^3 - 2\ X^2\ Y^3 - 2\ Y^3 - X^4\ Y + 2\ X^2\ Y$

(8) $3\ Y^4 + 2\ X^6\ Y^2 + 2\ X^4\ Y^2 - 2\ X^2\ Y^2 + X^8 - 2\ X^6 - X^4$

(9) $2\ X^5\ Y^2 + 2\ X\ Y^2 + X^9 - 2\ X^3$

(10) $4\ X^2\ Y^3 - 2\ Y^3 + X^8\ Y + 2\ X^6\ Y - 4\ X^4\ Y + 2\ X^2\ Y$

6. OPTIMIZATION

Using the definition of critical pairs given in Section 3 (henceforth called definition CP1), many intermediate rules are generated which later get simplified and thus do not appear in the Gröbner basis as illustrated by the following example:

The basis $B = \{1.\ 13\ X^2Y \rightarrow Y,\ 2.\ 8\ X\ Y^2 \rightarrow X\}$

Using definition CP1, we get from the superposition of rules 1 and 2, $<5\ X^2Y^2 + X^2,\ Y^2>$ as the critical pair, which gives a rule:

$$3.\ 3\ X^2Y^2 \rightarrow -\ Y^2 + 2\ X^2\ .$$

Rules 2 and 3 give a critical pair $<5\ X^2Y^2 - Y^2 + 2X^2,\ X^2>$, which gives the rule:

$$4.\ X^2Y^2 \rightarrow -3\ Y^2 + 5\ X^2\ ,$$

Rule 4 simplifies 3 to:

$$3'.\ 8\ Y^2 \rightarrow 13\ X^2\ .$$

Rule 3′ simplifies rule 2 above to:

$$2'.\ 13\ X^3 \rightarrow X\ .$$

Rules 1, 2′, 3′ and 4 constitute a Gröbner basis.

The above computation can be optimized using the following definition of critical pairs:

Definition CP2: The critical pair for two rules $c_1 t_1 \rightarrow R_1$ and $c_2 t_2 \rightarrow R_2$, where $c_1 \geqslant c_2$ is: let $t = \text{lcm}\ (t_1, t_2) = f_1 t_1 = f_2 t_2$ and $c_1 = a\ c_2 + b$, $-\left\lfloor \frac{c_2 - 1}{2} \right\rfloor \leqslant b \leqslant \frac{c_2}{2}$, then the superposition of the two left-hand-sides is the monomial $c_1 t$ from which by applying the two rules, we obtain the critical pair $<p, q>$, $p = b\ t + a\ f_2 * R_2$ and $q = f_1 * R_1$.

It is again easy to see that the polynomial $p - q$ obtained from the critical pair $<p, q>$ as defined above is still in the ideal.

As should be evident from the above discussion, rule 4 can be obtained directly from rules 1 and 2 by the greatest common divisor computation on the coefficients. This suggests a further optimization of definition CP2 using the gcd computation on the coefficients of the left-hand-sides of rules.

Definition CP3: The critical pair for two rules $c_1 t_1 \rightarrow R_1$ and $c_2 t_2 \rightarrow R_2$, where $c_1 > c_2$ or $c_1 = c_2$ is defined as follows:

(1) if c_2 divides c_1, we generate the critical pair using the lcm of $c_1 t_1$ and $c_2 t_2$ as the superposition and we obtain p and q by applying the given rules respectively. This case is the same as Definition CP2. Since lcm of c_1 and c_2 is c_1, suppose $c_1 = k\, c_2$, then, $p = f_1 R_1$ and $q = k\, f_2 R_2$; otherwise,

(2) if c_2 does not divide c_1, we generate the critical pair using the gcd. Let c be the extended gcd of c_1 and c_2; there exist a and b such that $c = a\, c_1 + b\, c_2$. Then the superposition of the two rules is $a\ c_1 f_1 t_1 + b\, c_2 f_2 t_2$, and the critical pair $<p, q>$ is: $p = c\ lcm(t_1, t_2)$ and $q = a\, f_1 R_1 + b\, f_2 R_2$.

Note that $<p,q>$ can be derived using the definition CP2 of critical pairs from the rules $c_1 t_1 \rightarrow R_1$ and $c_2 t_2 \rightarrow R_2$ by mimicking the gcd computation of c_1 and c_2.

We will now illustrate definition CP3 on the above example. The extended gcd of 13 and 8, the coefficients of the left-hand-sides of rules 1 and 2, respectively, is 1 such that $1 = (-3) * 13 + (5) * 8$. So, the critical pair of rules 1 and 2 using definition CP3 is $<X^2 Y^2, -3\ Y^2 + 5\ X^2>$, which directly gives rule 4. From rules 1 and 4, we get another rule:

$$6.\ 40\ Y^2 \rightarrow 65\ X^2\ ,$$

and from rules 2 and 4, we get yet another rule:

$$7.\ 24\ Y^2 \rightarrow 39\ X^2\ .$$

Definition CP3 on rules 6 and 7 produces rule 3' using which we get the Gröbner basis. As should be evident from this example, although definition CP3 using the extended gcd computation helps in directly generating certain rules (e.g., 4) without having to go through intermediate rules (e.g., 3), yet in order to generate other rules (e.g., 3'), it ends up generating other intermediate rules (e.g., 6 and 7). So, for some cases, definition CP3 works better than definition CP2 while in other cases, definition CP2 works better than definition CP3.

7. UNIQUENESS OF MINIMAL GRÖBNER BASIS

Definition: A Gröbner basis $B = \{b_1, ..., b_m\}$ is *minimal* (or *reduced*) if and only if for each i, $1 \leqslant i \leqslant m$, the head-coefficient of b_i is positive and p_i cannot be rewritten by any other polynomial in B when viewed as a rewrite rule.

Theorem 7.1: Let $B = (b_1, \cdots, b_m)$ be a basis of an ideal I in $Z[X_1, \cdots, X_n]$. Then, a minimal Gröbner basis of I is unique subject to a total ordering on indeterminates $X_1, ..., X_n$.

Proof: By contradiction. Assume that I has two minimal Gröbner bases $B = (b_1, \cdots, b_u)$ and $B' = (b_1', \cdots, b_v')$. Let $L_i \rightarrow R_i$ be the rule for b_i and $L_i' \rightarrow R_i'$ be the rule for b_i'. Assume also that the rules are ordered corresponding to their polynomials in both bases, i.e., $L_1 \rightarrow R_1 < \cdots < L_u \rightarrow R_u$ and $L_1' \rightarrow R_1' < \cdots < L_v' \rightarrow L_v'$.

Let i, $1 \leqslant i \leqslant u$, be the smallest rule number where the two bases differ. That is, b_i and b_i' are not identical, and for all $j \geqslant 1$ and $j < i$, if any, $b_j = b_j'$. There are two possibilities because of the same ordering being used on indeterminates $\{X_1, ..., X_n\}$ for both bases: (1) $L_i \neq L_i'$, or (2) $L_i = L_i'$, but $R_i \neq R_i'$.

Case (1): $L_i \neq L_i'$. Without any loss of generality we can assume that $L_i < L_i'$ since monomials are totally ordered.

The polynomial b_i is in I, and since B' is a Gröbner basis, b_i must reduce to 0 using polynomials in B'. But only rules corresponding to b_j', for $j \geqslant 1$ and $j < i$, if any, can be applied on b_i, which means that $i > 1$, as otherwise b_1 is not in I, which is a contradiction. The above also implies that b_i can also be reduced by polynomials in B, since for all $1 \leqslant j < i$, $b_j = b_j'$, which is a contradiction as B is a reduced basis.

Case (2): $L_i = L_i'$ but $R_i \neq R_i'$. This implies that $p = R_i - R_i'$ must reduce to 0. Let t_k be the head-term of p with a non-zero coefficient d (such a t_k must exist as otherwise $R_i = R_i'$). Let d_i and d_i' be the coefficients of t_k in R_i and R_i', respectively, so $d = d_i - d_i'$. For $R_i - R_i'$ to reduce to 0 there must be a rule $L_j \rightarrow R_j$, $1 \leqslant j < i$ (because $t_k <$ the head-term of L_i), in both B and B' which reduces $d\,t_k$. Consider a rule whose left-hand-side has the least coefficient; say, that rule is $L_j \rightarrow R_j$, where $L_j = c\,t_j$. Since B and B' are reduced, d_i and d_i' such that $-\frac{(c-1)}{2} \leqslant d_i \neq d_i' \leqslant \frac{c}{2}$. There does not exist any k such that $d = k\,c$, which implies that $R_i - R_i'$ cannot be reduced to 0 either using B or using B', since c is the smallest coefficient of the left-hand-sides of all the rules that can be applied to t_k. Hence, $R_i = R_i'$. (In a reduced basis, there cannot be two polynomials with the same head-term.)

So, there is no i such that b_i is different from b_i'. To show that $u = v$ implying that $B = B'$, assume $v > u$, in which case b'_{u+1} is in I such that b'_{u+1} reduces to 0 using B. But then, b'_{u+1} reduces using $\{b_1', \cdots, b_u'\}$ in B', which is a contradiction since B' is reduced. Hence the proof. □

Similar results about the uniqueness of a reduced canonical system have been reported in [12] for Thue systems and [15] for term rewriting systems; see also [16].

8. CONCLUSION

We have developed a Gröbner basis algorithm for polynomial ideals over Z. Similar argument can be used to get a Gröbner basis algorithm for finitely presented abelian groups. Implementations of these algorithms have been done in ALDES and LISP. Lauer [19] showed that the Gröbner basis can be used to construct canonical representatives for ideal residue classes. Since our algorithm computes a unique Gröbner basis of an ideal, in presence of such a basis, every polynomial in the polynomial ring has a canonical form. Moreover, the unique Gröbner basis of an ideal gives us insight into the structure of the ideal under consideration, such as its dimension, maximality, primality, etc., especially when the pure lexicographic ordering on monomials is used to compute the Gröbner basis, see [9] for more details.

Lankford [personal communication, Sept. 1983] suggested that there might be a relationship between the Gröbner basis computation and word problems over finitely presented algebras. It turns out that computing the Gröbner basis of a polynomial ideal over Z solves the uniform word problem (for elementary terms) over a finitely presented commutative ring with unity. The generators of the finitely presented commutative ring play the same role as the indeterminates of the polynomial ring over Z. The Gröbner basis is the canonical system for the finitely presented commutative ring. Further, a set of polynomials of the form $t_1 - t_2$, where t_i, $i = 1, 2$, is a term, can be treated as a presentation of a finitely presented commutative semi-group, so computing the Gröbner basis of the ideal generated by this set of polynomials also solves the word problem (for elementary terms) of the corresponding commutative semi-group (see [2] for an alternative but related approach). In a similar way, if the definition of critical pairs given in Section 3.4 is changed slightly so as not to consider the * operation of the polynomial ring, the Gröbner basis computation solves the uniform word

problem (for elementary terms) for finitely presented abelian groups. See [18] where an approach using a commutative-associative completion procedure is discussed for solving the uniform word problem for finitely presented abelian groups. The Gröbner basis approach for solving the uniform word problem for finitely presented abelian groups can also be used to solve the abelian group unification problem for elementary terms (see [17] for a different approach for solving this problem).

By adding additional polynomials (namely, $2 = 0$ and for each indeterminate X_i, $X_i * X_i = X_i$) into a basis over $Z[X_1, ..., X_n]$, we can simulate polynomial ideals over a boolean ring. Thus, the Gröbner basis algorithm over Z can be used as a way to prove theorems in propositional calculus by showing the unsatisfiability of a formula in conjunctive normal form; this method is closely related to Hsiang's approach [8]. The Gröbner basis approach also solves the uniform word problem (for elementary terms) for finitely presented boolean rings.

ACKNOWLEDGMENTS:

We thank Paliath Narendran for his helpful comments and suggestions on various drafts of this paper. Richard Harris implemented the algorithm in LISP on Symbolics 3600.

9. REFERENCES

[1] Bachmair, L., and Buchberger, B., "A Simplified Proof of the Characterization Theorem for Gröbner-Bases," *ACM-SIGSAM Bulletin*, 14/4, 1980, pp. 29-34.

[2] Ballantyne, A.M., and Lankford, D.S., "New Decision Algorithms for Finitely Presented Commutative Semigroups," *Computer and Mathematics with Applications* Vol. 7 pp. 159-165, 1981.

[3] Buchberger, B., "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms," *ACM-SIGSAM Bulletin*, 39, August 1976, pp. 19-29.

[4] Buchberger, B., "A Criterion for Detecting Unnecessary Reductions in the Construction of Gröbner-Bases," *Proceedings of EUROSAM 79*, Marseille, Springer Verlag Lecture Notes in Computer Science, Vol. 72, 1979, pp. 3-21.

[5] Buchberger, B. and Loos, R., "Algebraic Simplification," *Computer Algebra: Symbolic and Algebraic Computation* (B. Buchberger, G.E. Collins, and R. Loos, eds.), Computing Suppl. 4, Springer Verlag, New York, 1982, pp. 11-43.

[6] Huet, G., "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems," *JACM*, Vol. 27, No. 4, October 1980, pp. 797-821.

[7] Huet, G., "A Complete Proof of Correctness of the Knuth-Bendix Completion Procedure," *JCSS*, Vol. 23, No. 1, August 1981, pp. 11-21.

[8] Hsiang, J., *Topics in Theorem Proving and Program Synthesis*, Ph.D. Thesis, University of Illinois, Urbana-Champagne, July 1983.

[9] Kandri-Rody, A., *Effective Methods in the Theory of Polynomial Ideals*, Forthcoming Ph.D. Thesis, RPI, Troy, NY, May 1984.

[10] Kandri-Rody, A. and Kapur, D., "On Relationship between Buchberger's Gröbner Basis Algorithm and the Knuth-Bendix Completion Procedure," TIS Report No. 83CRD286, General Electric Research and Development Center, Schenectady, NY, December 1983.

[11] Kandri-Rody, A. and Saunders, B.D., "Primality of Ideals in Polynomial Rings," to appear in *Third MACSYMA User's Conference,* Schenectady, NY, July 1984.

[12] Kapur, D. and Narendran, P., "The Knuth-Bendix Completion Procedure and Thue Systems," *Third Conference on Foundation of Computer Science and Software Engg.,* Bangalore, India, December 1983, pp. 363-385.

[13] Kapur, D. and Sivakumar, G., "Architecture of and Experiments with RRL, a Rewrite Rule Laboratory," *Proceedings of a NSF Workshop on the Rewrite Rule Laboratory,* September 6-9, 1983, General Electric Report, April, 1984.

[14] Knuth, D.E. and Bendix, P.B., "Simple Word Problems in Universal Algebras," *Computational Problems in Abstract Algebras"* (J. Leech, ed.), Pergamon Press, 1970, pp. 263-297.

[15] Lankford, D.S. and Ballantyne, A.M., "On the Uniqueness of Term Rewriting Systems," Unpublished Manuscript, Louisiana Tech University, Math. Dept., December 1983.

[16] Lankford, D.S. and Butler, G., "Experiments with Computer Implementations of Procedures which often Derive Decision Algorithms for the Word Problem in Abstract Algebra," Technical Report, MTP-7, Louisiana Tech. University, August 1980.

[17] Lankford, D.S., Butler, G., and Brady, B., "Abelian Group Unification Algorithms for Elementary Terms," *Proceedings of a NSF Workshop on the Rewrite Rule Laboratory,* September 6-9, 1983, General Electric Report, April, 1984.

[18] Lankford, D.S., Butler, G., and Ballantyne, A.M., "A Progress Report on New Decision Algorithms for Finitely Presented Abelian Groups," *Proceedings of a NSF Workshop on the Rewrite Rule Laboratory,* September 6-9, 1983, General Electric Report, April, 1984. Also to appear in the *7th Conference on Automated Deduction,* NAPA Valley, Calif., May, 1984.

[19] Lauer, M., "Canonical Representatives for Residue Classes of a Polynomial Ideal," *SYMSAC,* 1976, pp. 339-345.

[20] Lausch, H., and Nobaurer, W., *Algebra of Polynomials,* North-Holland, Amsterdam, 1973.

[21] Schaller, S., *Algorithmic Aspects of Polynomial Residue Class Rings,* Ph.D. Thesis, Computer Science Tech., University of Wisconsin, Madison, Rep. 370, 1979.

[22] Szekeres, G., "A Canonical Basis for the Ideals of a Polynomial Domain," *American Mathematical Monthly,* Vol. 59, No. 6, 1952, pp. 379-386.

[23] van der Waerden, B.L., *Modern Algebra,* Vols. I and II, Fredrick Ungar Publishing Co., New York, 1966.

[24] Zacharias, G., *Generalized Gröbner Bases in Commutative Polynomial Rings,* Bachelor Thesis, Lab. for Computer Science, MIT, 1978.

APPENDIX

```
G := F
k := size(G)
i := 1
while i ⩽ size(G) do
 j := 1
 while 1 ⩽ j < i do
    <p, q> := critical_pair(G[j], G[i]);
    <hm, red> := normalize (G, S-polynomial (p,q)):
    if hm ≠ 0 then { G[k+1] := hm + red;
                                  k := k + 1 };
    j := j + 1;
  endwhile
  i := i + 1 ;
endwhile
```

COMPLEXITY OF TESTING WHETHER A POLYNOMIAL IDEAL IS NONTRIVIAL

S. Agnarsson*, A. Kandri-Rody*
Mathematical Sciences Department
Rensselaer Polytechnic Institute
Troy, New York

D. Kapur, P. Narendran
Computer Science Branch
Corporate Research and Development
General Electric
Schenectady, New York

B. D. Saunders*
Mathematical Sciences Department
Rensselaer Polytechnic Institute
Troy, New York

ABSTRACT

This paper considers the problem of deciding if a polynomial ideal generated by a finite set of polynomials is equal to the whole underlying polynomial ring. The problem is shown to be co-NP-hard for polynomials over any ring R which has Z_2 as a homomorphic image, under a mapping whose kernel is finitely generated. Such rings include the integers and the Gaussian integers. These results also show that for polynomial rings over integers, determining whether an ideal is prime and if the radical of an ideal I is I itself, are also co-NP-hard.

1. INTRODUCTION

The problem of determining if two polynomials represent the same element in a polynomial ring modulo an ideal is significant in symbolic computation. The method generally used is to compute a Grobner basis for the given basis of an ideal and to use the Grobner basis to reduce each polynomial to a canonical form [2]. This method is based on the result that equivalent polynomials modulo an ideal reduce by the Grobner basis of the ideal to the same canonical form. The Grobner basis can also be used to test the membership of a polynomial in an ideal. See the survey paper by Buchberger and Loos for many applications of polynomial simplification.

Using Hermann's results [6] Mayr and Meyer showed that the uniform word problem for commutative semi-groups is complete in exponential-space under log-space transformability. Their result can be used to show that computing the Grobner basis, as well as polynomial membership and polynomial equivalence with respect to an ideal, is exponential-space-hard [8]. For the case when the number of variables is 2, Buchberger has derived an explicit complexity bound. Apart from that case, not much is known about the complexity of computing the Grobner basis of an ideal.

This paper focuses on a simpler problem: deciding if an ideal of polynomials over a ring with certain structure is trivial (i.e., the given ideal is the whole polynomial ring). For rings containing a unit, this is equivalent to deciding whether 1 is a member of the ideal. We show that this problem for polynomial rings over integers is co-NP-hard. This result is also used to show that for polynomial rings over integers (i) the primality test of an ideal and (ii) if the radical of an ideal I is I, are also co-NP-hard. These results are based on relating the satisfiability problem in propositional calculus to checking if 1 does not belong to the ideal

* The authors were partly supported by NSF grant MCS-8314600.

over a boolean ring generated by a finite set of variables. Later the result about whether a given ideal in a polynomial ring on integers is trivial, is generalized to ideals of polynomials over a ring R such that R has Z_2 as a homomorphic image, under a mapping whose kernel is finitely generated. The Gaussian integers, for example, have this property. Similar results for computing ideals in polynomial rings over nontrivial fields have been reported in [1].

2. SATISFIABILITY PROBLEM AND CHECKING FOR NONTRIVIAL IDEAL

This section shows the relationship between the unsatisfiability problem for propositional calculus and the problem of testing if an ideal is trivial.

Stone showed the relationship between the Boolean ring

$$BR = (B, +, *, 0, 1)^1$$

and the Boolean algebra

$$BA = (B, \vee, \wedge, \sim, 0, 1),$$

where + is 'exclusive or' and * is $\wedge$. Here, 1 stands for 'true' and 0 stands for 'false.' Stone also proved that every element in a boolean ring has a unique normal form which can be obtained using the axioms of the boolean ring. The correspondence between the boolean ring and the boolean algebra can be shown using the following transformation:

from BA to BR:

$$x \vee y = x + y + x * y$$

$$x \wedge y = x * y$$

$$\sim x = x + 1$$

from BR to BA:

$$x + y = (x \wedge \sim y) \vee (\sim x \wedge y)$$

$$x * y = x \wedge y.$$

Hsiang showed how to use Stone's result to prove theorems in propositional calculus. In particular, he developed a clausal approach in which given a propositional formula f expressed using propositional variables $x_1, \ldots, x_n$, in conjunctive normal form, the unsatisfiability of f can be established. Each of the clauses in f are transformed into polynomials expressed using + and * in the boolean ring representation and equated to 1 (to mean that the clause is true). The formula f is unsatisfiable if, and only if, $1 = 0$ can be derived from the equations obtained from the clauses.

Given a clause $C = L_1 \vee \cdots \vee L_k$, where each L_i is a literal, $C = 1$ is transformed to a polynomial in a boolean ring over $x_1, \ldots, x_n$, as follows:

$$r(C) = \begin{cases} x_i + 1 & \text{if } C \text{ is a variable } x_i \\ x_i & \text{if } C \text{ is a literal } \sim x_i \\ r(L_1) * r(L_2 \vee \cdots \vee L_k), & \text{otherwise.} \end{cases}$$

[1] A boolean ring is a commutative ring with 1 such that for every element a in the ring, $a * a = a$ as well as $a + a = 0$.

Theorem 1 [7]: A formula f, which is a set of clauses, is unsatisfiable if, and only if, $1 = 0$ can be equationally derived from $\{r(C_i) = 0 \mid C_i \text{ in} f\}$ using axioms of boolear. ring.

Consider the free boolean ring over $[x_1, ..., x_n]$ (note that every monomial in a polynomial here has coefficient 1 and each indeterminate in the monomial has at most degree 1). The representation of a formula f as a set of elements in the boolean ring $BR[x_1, ..., x_n]$ is the ideal generated by $(r(C_1), ..., r(C_m))$. Thus the check whether $1 = 0$ can be equationally derived from $\{r(C_i) = 0\}$ is the same as checking whether 1 belongs to $(r(C_i), ..., r(C_m))$. Thus the above theorem can be restated as:

Theorem 2: A formula f is unsatisfiable if and only if 1 belongs to the ideal $(r(C_1), ..., r(C_m))$ in $BR[x_1, ..., x_n]$.

Since the unsatisfiability problem is known to be co-NP-complete (and we have a polynomial-time transformation), we see that the membership of 1 in an ideal in $BR[x_1, ..., x_n]$ is co-NP-complete. (See also Gary and Johnson (p. 251) where they credit a similar result to Fraenkel and Yesha.)

Henceforth, we will refer to the problem of testing whether an ideal is the whole ring as the *triviality* problem. For rings with identity, this is the same as the problem of testing whether 1 belongs to an ideal I. Since the free Boolean ring generated by $x_1, ..., x_n$ is isomorphic to the polynomial quotient ring $Z_2[x_1, ..., x_n]/(x_1^2 + x_1, ..., x_n^2 + x_n)$, it can be easily shown that

Theorem 3: The triviality problem of an ideal I in $Z_2[x_1, ..., x_n]$ is at least co-NP-hard.

Further, because Z_2 is a homomorphic image of Z (under the homomorphism that maps every even integer to 0 and every odd integer to 1), we have the following result about ideals in $Z[x_1, ..., x_n]$:

Theorem 4: Let $I = (f_1, ..., f_m)$ be an ideal in $Z[x_1, ..., x_n]$. Then, the problem to test whether 1 belongs to I is at least co-NP-hard.

Proof: An algorithm for testing whether 1 belongs to an ideal I in $Z[x_1, ..., x_n]$ can also be used to test whether 1 belongs to an ideal J of polynomials over $Z_2[x_1, ..., x_n]$ by testing whether $J' = (J, 2)$ in $Z[x_1, ..., x_n]$ has 1. So, checking for 1 belonging to I over $Z[x_1, ..., x_n]$ is at least as hard as checking whether 1 belongs to $Z_2[x_1, ..., x_n]$. Hence the theorem. □

In a later section, we generalize this result to show that the triviality problem of an ideal I over a polynomial ring $R[x_1, ..., x_n]$, where $R/(r_1, ..., r_s)$, $r_i \in R$, is isomorphic to Z_2 is also co-NP-hard. These results are shown by defining a homomorphism from R to Z_2 and identifying polynomials in $R[x_1, ..., x_n]$ which play the role of 'true' and 'false.'

2.1 Primality, Radical, and Power Problems

The above results can be also used to analyze the complexity of testing (i) primality of an ideal, (ii) whether the radical of an ideal I is I itself, and (iii) whether some power of a given polynomial P is in an ideal I.

Theorem 5: Let $I = (f_1, ..., f_m)$ be an ideal in $Z[x_1, ..., x_n]$. Let x be an indeterminate different from x_i, $i = 1, ..., n$, and let $I' = (I, x^2)$ be the ideal generated by the f_i's and x^2 in $Z[x_1, ..., x_n, x]$. Then,

(1) I' is a prime ideal if and only if $I = (1)$.

(2) Radical$(I') = I'$ if and only if $I = (1)$.

Proof: (a) I' is prime implies x belongs to I' but x is not in I' unless 1 belongs to I.

Conversely, 1 belongs to I implies 1 belongs to I' which implies I' is prime.

(b) Radical(I') = I' implies x belongs to I' but x is not in I' unless 1 belongs to I.

Conversely, 1 belongs to I implies (1) = I' which implies that Radical(I') = I'. □

Corollary 5.1: Let $I = (f_1, ..., f_m)$ be an ideal in $Z[x_1, ..., x_n]$. The problems of testing whether I is prime and Radical(I) = I are at least co-NP-hard.

Theorem 6: The problem of testing whether some power of a polynomial p is in an ideal I over $Z[x_1, ..., x_n]$ is as hard as the triviality problem for ideals.

Proof: The triviality problem can easily be reduced to the power problem, by choosing p=1. For the converse case, let $I = (f_1, ..., f_m)$ be an ideal in $Z[x_1, ..., x_n]$ and let p be a non-zero polynomial in $Z[x_1, ..., x_n]$. Consider $I' = (I, p\,x - 1)$ where x is a new indeterminate. If 1 belongs to I', then it can be seen that a power of p belongs to I by employing the substitution $x = 1/p$ as in the proof of Hilbert's Nullstellensatz. Conversely, if p^m is in I, then $p^m x^m$ is in I' which implies 1 is in I', using $px - 1$. □

3. BOOLEAN FORMULAS AND POLYNOMIAL IDEALS

Let R be a ring containing a finite set of elements $\{r_1, ..., r_s\}$ such that $R/(r_1, ..., r_s) = Z_2$. We will denote the homomorphism from R to Z_2 by ϕ. Then $R[x_1, ..., x_n]/(r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n)$ also behaves like a free Boolean ring. This will be used in the following to prove our main theorem.

Definition: For $r \in R$ we will call r *odd* if $\phi(r)$=1. We will call r *even* iff $\phi(r) = 0$.

Let ω be some odd element of R which commutes with all elements of R.

Definition: For $u, v \in R$ define $u \wedge v = u * v$, define $u \vee v = u + v + u * v$, define $\sim u = u + \omega$.

Definition: For $p \in R[x_1, ..., x_n]$ we define p to be *odd* iff $p(c_1, ..., c_n)$ is odd for all evaluation points $c_1, ..., c_n \in R$. Similarly, p is *even* iff the values of all possible evaluations are even.

It can easily be shown that 'odd' and 'even' behave like truth values; i.e. 'odd' behaves like 'true' and 'even' behaves like 'false' with respect to the operators '$\sim$', '$\wedge$' and '$\vee$'. This is true both for R and $R[x_1, ..., x_n]$, the difference being that in R, all elements are either even or odd, but in $R[x_1, ..., x_n]$ there exist elements, which are neither odd nor even. This relationship between polynomials and truth values is used to prove in a more general setting the result, which was proved for Z in the previous section.

Theorem 7: Let R be a ring, $\{r_1, ..., r_s\}$ be elements of R such that $R/(r_1, ..., r_s) = Z_2$. Let ω be some odd element of R which commutes with all elements of R. The problem of determining whether an ideal generated by a finite set of polynomials in $R[x_1, ..., x_n]$ contains ω is co-NP-hard.

Proof: This theorem can be derived from Theorem 8 below which relates 3-unsatisfiability and the ideal membership problem. □

From the above theorem, we immediately have:

Corollary 7.1: Determining whether an ideal in $R[x_1, ..., x_n]$ is trivial is co-NP-hard, where R fulfills the requirements of Theorem 7.

Proof: This follows directly from Theorem 8, since the ideal $(r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n, p_1 + \omega, ..., p_m + \omega)$ is trivial iff it contains ω (since it then contains all of R). □

Theorem 8: There exists a polynomial time transformation from a 3-unsatisfiability problem $C_1 \wedge, ..., \wedge C_m$ into the question whether ω is a member of the ideal generated by the set $\{r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n, p_1 + \omega, ..., p_m + \omega\}$, where $R/(r_1, ..., r_s) = Z_2$, ω is an arbitrary commutative odd element of R, and each p_i is constructed from C_i.

Proof: As in the translation from boolean algebras to boolean rings discussed above, interpret each clause $C_i = a \vee b \vee c$ as a polynomial $p_i = a \vee b \vee c$. As stated above, the operators '$\sim$,' '$\wedge$' and '$\vee$' can be defined for polynomials so that odd and even polynomials behave respectively as true and false. Therefore, the unsatisfiability problem can also be posed as the question of whether $p_1 \wedge \cdots \wedge p_m$ is even in all evaluations. This is the same as asking if $\prod_{i=1}^{m} p_i$ is even in all evaluations, which by definition, is the same as checking if $\prod_{i=1}^{m} p_i$ is an even polynomial.

By Theorem 9 below, this is the same as asking if ω is a member of the ideal generated by the set $\{r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n, p_1 + \omega, ..., p_m + \omega\}$.

As an example, the clause $x \vee \sim y \vee z$ gets transformed into the polynomial

$$x \vee \sim y \vee z = x + (\sim y \vee z) + x * (\sim y \vee z) =$$
$$x + \sim y + z + \sim y * z + x * (\sim y + z + \sim y * z) =$$
$$x + y + \omega + z + (y + \omega) * z + x * (y + \omega + z + (y + \omega) * z) =$$
$$x + y + \omega + z + y * z + \omega * z + x * y + x * \omega + x * z + x * y * z + x * \omega * z.$$

Since each clause contains at most three literals, the number of terms that can arise from an expansion such as the previous one, is bounded by a constant. □

Theorem 9: In $R[x_1, ..., x_n]$, $\prod_{i=1}^{m} p_i$ is even iff $\omega \in (r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n, p_1 + \omega, ..., p_m + \omega)$ The proof of this theorem follows from the following two lemmas.

Lemma 1: In $R[x_1, ..., x_n]$, $\prod_{j=1}^{m} p_j$ is even iff there exist $a_1, \ldots, a_m$ such that $\sum_{j=1}^{m} a_j(p_j+\omega)$ is odd.

Lemma 2: For $p \in R[x_1, ..., x_n]$, p is odd (respectively even) iff $p = \omega$ (respectively 0) mod $\{r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n\}$

Proof of Lemma 1:

Case a) Let it be given that $\prod_{j=1}^{m} p_j$ is even. Let $a_i = \omega \prod_{j=1}^{i-1} p_j$, where a_1 is ω. Then

$$\sum_{j=1}^{m} a_j(p_j + \omega) = \sum_{j=1}^{m} (\omega \prod_{i=1}^{j} p_i + \omega^2 \prod_{i=1}^{j-1} p_i) = \omega \sum_{j=1}^{m} \prod_{i=1}^{j} p_i + \omega^2 \sum_{j=1}^{m} \prod_{i=1}^{j-1} p_i =$$

$$\omega [\omega + \prod_{j=1}^{m} p_j + \sum_{j=1}^{m-1} (\prod_{i=1}^{j} p_i + \omega \prod_{i=1}^{j} p_i)].$$

Now $\prod_{j=1}^{m} p_j$ is even, and the last sum is even because each term is of the form $x + \omega x$. Therefore the whole result is odd.

Case b) Let it be given that $\sum_{j=1}^{m} a_j(p_j + \omega)$ is odd. Then by lemma 2, there exist b's and c's such that $\sum_{i=1}^{m} a_i(p_i + \omega) + \sum_{i=1}^{k} b_i(x_i^2 + x_i) + \sum_{i=1}^{l} c_i r_i = \omega$. Multiplying on both sides by $\prod_{i=1}^{m} p_i$, we get: $\sum_{i=1}^{m} a_i(p_i + \omega) \prod_{j=1}^{m} p_j + \sum_{i=1}^{m} b'_2(x_i^2 + x_i) \prod_{j=1}^{m} p_j + \sum_{i=1}^{l} c'_2 r_i \prod_{j=1}^{m} p_j = \omega \prod_{i=1}^{m} p_i$. Now each term in the first sum is even, because it contains $p_i + \omega$ and p_i. Each term in the second sum is even, because it contains $x_i^2 + x_i$. Each term in the third sum is even because it contains Er_2. Therefore each side of the equation is even. But for $\omega \prod_{i=1}^{m} p_i$ to be even, $\prod_{i=1}^{m} p_i$ must be even. □

Proof of Lemma 2: Let $l(p)$ be the highest index of a variable occurring in p. If no variable occurs in p, define $l(p)=0$. Let $t(p)$ be the highest power of $x_{l(p)}$ in p. If no variable occurs in p, define $t(p)=0$. The proof is by induction on $l(p)$ and $t(p)$.

Basis of outer induction, $l(p)=0$: Obvious — this case reduces the lemma to R.

Basis of inner induction, $t(p)=0$: Obvious as above.

Inner inductive step: Given that it holds for all p, such that $l(p) < c$, and that it holds for all p with $l(p) = c$ and $t(p) < d$, we wish to prove that it holds for p', where $l(p') = c$ and $t(p') = d$. Find p_0, p_1 and p_2 such that $p' = x_c^2 p_2 + x_c p_1 + p_0$ where p_0 contains no powers of x_c, and the polynomials p_2 and p_1 contain no powers of x_c higher than $d - 2$. Then $p' = x_c^2 p_2 + x_c p_2 - x_c p_2 + x_c p_1 + p_0 = (x_c^2 + x_c) p_2 + (p_1 - p_2) x_c + p_0$. By evaluating at $x_c = 0$, we can certify that p_0 is even (respectively even). Therefore, by induction, $p_0 = \omega$ (respectively 0) mod $\{r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n\}$ Now $x_c^2 + x_c$ is obviously even, so by lemma 2, $(x_c^2 + x_c)p_2$ is also even. Therefore $(p_1 - p_2)x_c$ must be even, which implies that $p_1 - p_2$ is even, and therefore, by inner inductive hypothesis, $p_1 - p_2 = 0$ mod $\{r_1, ..., r_s, x_1^2 + x_1, ..., x_n^2 + x_n\}$.

Outer inductive step: Given that the theorem holds for all p with $l(p) < c$, we wish to prove that it holds for p' with $l(p') = c$ and $t(p') = 0$. This condition is trivially satisfied, since there are no such p's. □

4. REFERENCES

1. Bayer, D.A., *The Division Algorithm and the Hilbert Scheme.* Ph.D. Thesis, Harvard University, June 1982.
2. Buchberger, B. "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms," *ACM-SIGSAM Bulletin, 39,* August, 1976, pp. 19-29.
3. Buchberger, B., "A Criterion for Detecting Unnecessary Reductions in the Construction of Grobner-Bases," *Proceedings of EUROSAM 79, Marseille,* Springer Verlag Lecture Notes in Computer Science, Vol. *72,* 1979, pp. 3-21.
4. Buchberger, B., and Loos, R., "Algebraic Simplification," in *Computer Algebra: Symbolic and Algebraic Computation.* (Eds. B. Buchberger, G.E. Collins, R. Loos), Computing Suppl. 4, Springer Verlag, New York, 1982, pp. 11-43.

5. Gary, M.R., and Johnson. D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, San Francisco, CA, 1979.

6. Hermann, G., "Die Frage der endlich vielen Schritte in der Theorie der Polynomideale," *Math. Ann., 95,* pp. 736-788, 1926.

7. Hsiang, J., *Topics in Theorem Proving and Program Synthesis.* Ph.D. Thesis, University of Illinois-Champagne, Sept., 1983.

8. Mayr, E., and Meyer, A.R., "The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals," MIT, Lab. for Computer Science Report LCS/TM-199, 1981.

9. Stone, M., "The Theory of Representations for Boolean Algebra," *Trans. AMS* Vol. *40,* pp. 37-111, 1936.

10. Van der Waerden, B.L., *Modern Algebra,* Vol. I, Fredrick Ungar Publishing Co, New York, 1966.

PRIMALITY OF IDEALS IN POLYNOMIAL RINGS[1]

Abdelilah Kandri Rody[2]
Rensselaer Polytechnic Institute
Troy, NY
and
University Mohammed V
Rabat, Morocco

B. David Saunders
Rensselaer Polytechnic Institute
Troy, NY

1. Introduction

We construct algorithms for testing the primality and maximality of an ideal in a multivariate polynomial ring over a field. Effective methods for these and related questions in commutative algebra have been studied in the past century. If one can determine the associated primes for the primary decomposition of an ideal then one can answer our main question, primality. In 1925, G. Hermann [Her] constructed an algorithm for the primary decomposition of an ideal. Seidenberg [Sei] expanded upon her methods and gave more precise conditions under which they apply. In 1982, Lazard [Laz 3] has given an algorithm for primality testing. Lazard's paper [Laz 3] is also a good reference to other work in this area. All of the algorithms, including ours, spend considerable time in the general case. We observe, however, that our method, which avoids adding new variables, is suitable for pencil and paper application to many non-trivial simple examples. The other methods are based on the construction and factorization of a polynomial, the "ground-form", in many new indeterminates. The approach taken here exploite the interplay between two reduction processes which serve, among other things, to determine membership of a polynomial in a

---

[1]This work partially supported by NSF grant MCS-8314600.
[2]Some of the the results will appear in A. Kandri-Rody's doctoral thesis at RPI.

given ideal. The first of these we call B-reduction. It is Buchberger's reduction process based on the the construction of a Gröbner basis for the given ideal. The second we call R-reduction. It is Ritt's reduction process based on the concept of characteristic set. Characteristic sets are used by Ritt primarily for work with differential ideals, but are useful also in application to polynomial ideals as this paper demonstrates. We are thankful to M. Singer for bringing characteristic sets to our attention.

In the next section, we will define these two reductions and give basic definitions. In section 3 we give an algorithm to test whether an ideal is maximal. In section 4 we give an algorithm to test whether an ideal is prime. Henceforth, let $R = K[X_1,\ldots,X_n]$ be the ring of polynomials where $X_1,\ldots,X_n$ are indeterminates and K is a field.

2. Definitions

In this section, we will characterize Buchberger's completion algorithm and the notion of characteristic set. For more details see Buchberger [Buc 1,2], Buchberger & Loos [B.L.] and Ritt [Rit]. A treatment of Buchberger's algorithm which unifies it with the Knuth-Bendix completion procedure is given by Kandri Rody and Kapur [K.K.].

2.1 B-complete basis

Following Buchberger, we reserve the word term to refer to any product $\prod_{i=1}^{n} X_i^{e_i}$, $e_i \geq 0$, of indeterminates with possible repetition. A monomial is a term multiplied by a non-zero coefficient. We represent a polynomial as a sum of monomials in which nc term appears more than once, such a polynomial is said to be in distributive normal form. The B-reduction process we consider will depend on an ordering which we apply both to monomials within a polynomial and to compare polynomials. Though, for most purposes, a variety of ordering can be used, we will use only the pure lexicographic ordering defined as follows:
Given two terms $t_1 = \prod_{i=1}^{n} X_i^{e_i}$ and $t_2 = \prod_{i=1}^{n} X_i^{f_i}$, we say that t_1 is of lower order than t_2 and write $t_1 < t_2$ if either $e_n < f_n$ or there exist $i < n$ such

that for every $j > i$, $e_j = f_j$ and $e_i < f_i$.
For example

$$X_1 < X_2 < \ldots < X_n , \qquad X_1X_2 < X_1X_3 , \qquad X_1^3 < X_1X_2.$$

Suppose we are given two polynomials $P_1 = c_1t_1 + P_1'$ and $P_2 = c_2t_2 + P_2'$ where the terms of P'_i $(i = 1, 2)$ are all less than t_i, then we say P_2 is lower than P_1 and we write $P_2 < P_1$ if $t_2 < t_1$ or $(t_2 = t_1$ and $P'_2 < P_1')$. We say the zero polynomial is of lower order than any nonzero polynomial.

Note that if C is a product of two polynomials A and B then $A < C$ and $B < C$.

The B-reduction process is characterized by the fact that if a polynomial P_1 B-reduces to P_2 with respect to $B_1,\ldots,B_k$ then $P_2 < P_1$ and $P_1-P_2 = \sum_{i=1}^{k} D_iB_i$. We say that P_1 is B-irreducible with respect to $A_1,\ldots,A_k$ if there is no polynomial P_2 such that P_1 B-reduces to P_2.

A finite set $B = \{B_1,\ldots,B_r\}$ of polynomials is called a Gröbner basis for an ideal $I = (A_1,\ldots,A_k)$ if for any polynomial P, the B-reduced form of P is unique. An equivalent definition is that any polynomial in the ideal B-reduces to 0.

If each B_i is B-irreducible with respect to B_j, $i \neq j$, we say that $B = (B_1,\ldots B_r)$ is B-complete. (B is the minimal Gröbner basis for I in Buchberger's termonology). One knows that such B-complete basis for a given ideal I can be effectively computed and is unique. In this case, each polynomial P in R B-reduces to a unique B-irreducible form. In the rest of the paper, for a given B-complete basis $(B_1,\ldots,B_r)$, we will assume $B_i < B_{i+1}$.

2.2 Characteristic sets

For the second reduction process, R-reduction, we view a polynomial P in $R = K[X_1,\ldots,X_n]$ as a polynomial in X_m the highest variable occuring in P. The coefficient of the highest power of X_m, a polynomial in the variables $X_1,\ldots,X_{m-1}$, is called the initial of P. Let $\{C_1,\ldots,C_s\}$ be polynomials in I, and let N_i, $i=1,\ldots,s$, be the initial of C_i. Let P_1 and P_2

be in $K[X_1,\ldots,X_n]$. We say that P_2 is R-reducible with respect to P_1 if either (i) $P_2 \geq P_1$ and P_2 does not have any variable greater than those occuring in P_1 or (ii) P_2 is B-reducible with respect to P_1.

For example, in $Q[X,Y,Z]$, $X < Y < Z$, let $P_1 = Z + Y^2 + 1$, let $P_2 = XY + 1$ and let $P_3 = XZ + X^2Y^2 + 2$. Then, P_2 is not R-reducible with respect to P_1 and P_1 is not R-reducible with respect to P_2. However, P_3 is R-reducible with respect to P_1 and P_2.

A set $\{C_1,\ldots,C_s\}$ is a characteristic set for an ideal I if
(1) C_i, $i=1,\ldots,s$, belongs to I and
(2) either (*) $s = 1$, $C_1 \neq 0$ and I contains no polynomial lower than C_1 or (**) $s > 1$, $\{C_1,\ldots,C_{s-1}\}$ is a characteristic set for some ideal, C_s introduces at least one variable higher than those occurring in $C_1,\ldots,C_{s-1}$ and (3) C_s is B-reduced with respect to $C_1,\ldots,C_{s-1}$ and
(4) every polynomial in I can be R-reduced to 0 with respect to $C_1,\ldots,C_s$. (This definition is equivalent to the one given in [Rit])

The set of polynomials which can be R-reduced to 0 with respect to a characteristic set $C = \{C_1,\ldots,C_s\}$ form an ideal J which we call the ideal generated characteristically by C. Note that J contains I.

A more general version of the following theorem, applicable to differential polynomials, is given in [Rit], page 6 (see also [Z.S], theorem 9, page 30).

Theorem 2.1 Let I be an ideal in $R = K[X_1,\ldots,X_n]$, let $C_1,\ldots,C_s$ be a characteristic set for I and, N_i be the initial of C_i. Then, given any G in R, we can construct a unique R-irreducible H in R such that

$$\left(\prod_{i=1}^{s} N_i^{s_i}\right)G - H = \sum_{i=1}^{s} D_i C_i$$

i.e. G R-reduces to H.

Theorem 2.2 Let I be an ideal in $K[X_1,\ldots,X_n]$ with B-complete basis $(B_1,\ldots,B_r)$. Let $C = \{C_1,\ldots,C_s\}$ be the set of B_i which first introduce a new variable. Then, I is contained in the ideal characteristically generated by C.

Proof: It suffices to show that each B_j can be R-reduced to 0 with respect to C. Suppose B_j R-reduces to R_j with respect to C. If $R_j \neq 0$ then R_j is of lower degree than any C_i in Y_i, hence of lower degree than any B_j in its leading variable, so that R_j does not B-reduce to 0; but

$$R_j = \left(\prod_{i=1}^{s} N_i^{s_i}\right)B_j - \sum_{i=1}^{s} D_i C_i \text{ is in I.}$$

Let $B = (B_1,\ldots,B_r)$ be the B-complete basis for an ideal I in $K[X_1,\ldots,X_n]$. Let $C = \{C_1,\ldots,C_s\}$ be the set of B_i which first introduce a new variable. Then, C is a characteristic set for I which we call the extracted characteristic set for I.

3. Maximal ideals

Let $I = (A_1,\ldots,A_s)$ be an ideal in $R = K[X_1,\ldots,X_n]$, where K is a field. The algorithm MaximalIdeal below tests whether I is a maximal ideal in R. The algorithm is expressed in the NEWSPAD language being developed by Jenks and Trager [J.T.]. Newspad is convenient for us because it permits suitably abstract specification of the algorithm (arbitrary ground field and variable list for example). But also we have executable code (well, nearly so). The code consists of a variety of declarations followed by the definition of the predicate MaximalIdeal per se.

```
IdealPackage (K, V, IdealBasis): ExportedFunctions == Definitions where
 K: Field
 V: List(Expression) { variable list }
 IdealBasis: List(MultivariatePolynomial(K, V))
 ExportedFunctions == with
      MaximalIdeal: IdealBasis -> Boolean
      PrimeIdeal: IdealBasis -> Boolean
 Definitions == add
      B-CompleteBasis := IdealBasis
      Poly := MultivariatePolynomial(K, V)
      { We assume this domain of polynomials in the variables V with
      coefficients in K has among other things the function
          irreducible: (Poly, Field) -> Boolean
      which returns true if the Poly is irreducible when viewed as a
      polynomial over the given field. The field may contain several of the
      variables in V as algebraic or transcendental elements.
      Also we use
          lastVariable: Poly -> Expression
      which returns the highest variable from V that occurs in its
      argument.}
```

```
{ declarations }
p,q: Poly
B: B-CompleteBasis
F: Field
MaximalIdeal(A: IdealBasis) ==
    B := GrobnerPackage$BuchbergerCompletion(A)
    if #B ≠ #V then return false
    {i-th poly in B must introduce i-th variable }
    for p in B , X in variables repeat
        if lastVariable(p) ≠ X then return false
    F := K
    for P in B, X in variables repeat
        if irreducible(p,F)
        then F := QuotientRing(polyRing(F,X),p)
        else return false
    Return true
```

{IdealPackage continued in section 4}

Theorem 3.1 The algorithm MaximalIdeal is correct, i.e. if $\{B_1,\ldots,B_s\}$ is a B-complete basis for I, then I is a maximal ideal in $K[X_1,\ldots,X_n]$ if and only if $s = n$, B_i introduces X_i, and B_i is an irreducible polynomial over $K_{i-1}=K[X_1,\ldots,X_{i-1}]/(B_1,\ldots,B_{i-1})$.

Proof If the ideal I is maximal then the algorithm returns "true". This follows from [Z.S.], theorem 24, page 197. Note that the B-complete basis is "the canonical basis" described there; this is because the canonical basis is a Gröbner basis and it is minimal. The unique basis with these properties is the B-complete basis. If the algorithm returns "true" we show I is maximal by an induction over n. If n=1, $I=(B_1)$ is a maximal ideal in $K[X_1]$. Assume $n>1$, by induction hypothesis $K_{n-1} = K[X_1,\ldots,X_{n-1}]/(B_1,\ldots,B_{n-1})$ is a field and, since B_n is irreducible in $K_{n-1}[X_n]$, $K_n = K[X_1,\ldots,X_n]/(B_1,\ldots,B_n)$is a field. Thus I is a maximal ideal.

Theorem 3.2 Let I be a maximal ideal in $R = K[X_1,\ldots,X_n]$ where K is a field and let $(B_1,\ldots,B_n)$ be the complete basis for I. Then,

(a) Each B_i introduces X_i.

(b) The intersection of I and $K[X_1,\ldots,X_i]$ is $(B_1,\ldots,B_i)$.

(c) If K is a finite field of cardinality q, then the cardinality of R/I is $q^{p_1\cdots p_n}$ where p_i = degree of B_i with respect to X_i.

(d) If K is algebraically closed then $B_i = X_i - a_i$ for some a_i in K.

The proof follows readily from theorem 3.1.

4. Prime ideals

In this section, we will show how to compute a characteristic set for a prime ideal with a given basis, and, given a characteristic set of a prime ideal, how to compute a basis for the ideal. With these facilities, we will prove the following algorithm to test whether an ideal is prime.

```
{IdealPackage -- continued from section 3}
    CharacteristicSet := List(MultivariatePolynomial(K, V))
    { declarations }
    p, g,: Poly
    A, G, H: IdealBasis
    B: B-CompleteBasis
    C: CharacteristicSet
    CharacteristicSetExtract: B-CompleteBasis -> CharacteristicSet
    CharacteristicSetOfPrimeIdeal: CharacteristicSet -> Boolean
    GenerateCompleteBasis: CharacteristicSet -> B-CompleteBasis
    primitive: (Poly, Expression) -> Boolean

    PrimeIdeal(A) ==
       { Get B-complete basis }
       B := GröbnerPackage$BuchbergerCompletion(A)
       C := CharacteristicSetExtract(B)
       if CharacteristicSetOfPrimeIdeal(C) then
            if B = GenerateCompleteBasis(C)
            then return true else return false

    CharacteristicSetExtract(B) ==
      {we assume B ordered following the pure lexicographical ordering }
       C := [ first B ]
       for p in rest B if lastVariable P > lastVariable(first C)
            then C := cons(p,C)
       return C

    CharacteristicSetOfPrimeIdeal(C) ==
       {True if C is a characteristic set of a prime ideal }
       F = QuotientField (MultivariatePolynomial(K, {X in V | for i in
       1..s, X ≠ lastVariable(C_i)}))
       for i in 1..s
            if not primitive(C_i, lastVariable (C_i)) then return false
            if not irreducible(C_i, F) then return false
            F := QuotientRing(PolyRing(F,lastVariable(C_i)), C_i)
            { Then F is a field because C_i is irreducible. }
       return true

    primitive(p, X) ==
       if 1 = greatest common divisor of the coefficients of p (as a
       polynomial in X) then return true else return false
```

```
GeneratedCompleteBasis(C) ==
    G := GröbnerPackage$BuchbergerCompletion(C)
    H := []
    while H ≠ G repeat
        H := G
        for p in C
            L := BasisOf({g | g*Initial(p) = ΣP_iG_i, G_i is in G})
            { For L we can use Herman's module basis algorithm,
            c.f. [Sei] page 276.}
            { Get B-complete basis for ideal generated by L }
            G := GrobnerPackage$B-CompleteBasisExtension(L)
    return G
```

Theorem 4.1 Let $B = (B_1,\ldots,B_r)$ be a B-complete basis for an ideal I in $K[X_1,\ldots,X_n]$. Let $C = \{C_1,\ldots,C_s\}$ be the extracted characteristic set for I, and, let J be the ideal generated characteristically from C. The X_i can be divided into two sets, $U_1,\ldots U_t$ and $Y_1,\ldots,Y_s$, $t+s=n$, where Y_i is the highest variable occuring in C_i and the U_i represent the other variables. Then (a) implies (b) and (b) is equivalent to (c), for the assertions:

(a) I is a prime ideal.

(b) For $i = 1,\ldots,s$, C_i is irreducible over K (hence C_i is primitive as a polynomial in the last variable) and is also irreducible when viewed as a polynomial over F_{i-1}, the field of quotients of $K(U_1,\ldots,U_t)[Y_1,\ldots,Y_{i-1}]/(C_1,\ldots,C_{i-1})$.

(c) J is a prime ideal and $C = \{C_1,\ldots,C_s\}$ be the extracted characteristic set for J.

Proof: Let us show (a) implies (b). Consider C_i as a polynomial over K, suppose C_i has a nontrivial factorisation $C_s = A.B$, where A or B do not belong to K. Then, A or B belong to I which implies A or B can be B-reduced with respect to B_j, $B_j < C_i$, but then C_i will be also B-reduced which is not possible since $(B_1,\ldots,B_r)$ is a B-complete basis. Thus, C_i is an irreducible polynomial over K. Consider C_i viewed as a polynomial over F_{i-1}. The case $s = 1$ is obvious from the irreducibility and $I = (C_1)$. Assume $s > 1$. Note that the intersection of I and $K(U_1,\ldots,U_t)[Y_1,\ldots,Y_{s-1}]$ is a maximal ideal for which $\{C_1,\ldots,C_{s-1}\}$ is a basis, hence that F_{s-1} is a field. Let us show that C_s is irreducible over F_{s-1}. Let $C_s = AB$ (over F_{s-1}). Clearing denominators of coefficients in A and B, we obtain $DC_s = GH$, where D is in $K[U_1,\ldots,U_t)]$ and G, H are

polynomials in $K[U_1,\ldots,U_t,Y_1,\ldots,Y_s]$. Without loss of generality, assume that G and H are both of positive degree in Y_s, but then each is of degree lower than C_s in Y_s and hence cannot be B-reduced to 0, i.e it is not in I. Note that the initial of H (or of G) does not belong to I because D and the initial of C_s do not belong to I. Since I is prime, C_s must be irreducible.

Now since (a) implies (b) it is immediate that (c) implies (b). Let us show that (b) implies (c). We remark that Ritt [Rit], page 89, has shown that the C_i are irreducible over F_{i-1} if and only if J is prime. C_i will be considered as a polynomial over F_{i-1}. The case s = 1 is obvious from the irreducibility over K and $J = (C_1)$. Suppose s > 1, by induction, J_{s-1}, the ideal generated characteristically by $C_1,\ldots,C_{s-1}$, is a prime ideal. Note that J_{s-1} is also the set of polynomials which vanish for $u_1,\ldots,u_t,y_1,\ldots,y_{p-1}$ where y_i is a root of Ci considered over F_{i-1}. Let y_s be a zero of C_s. Let I_s be the totality of those polynomials in $K[X_1,\ldots,X_n]$ which vanish for $u_1,\ldots,u_t,y_1,\ldots,y_s$. Then, I_s is a prime ideal. We shall prove that $J = I_s$. Let G be in J. $(\prod_{i=1}^{s} N_i^{s_i})G = \sum_{i=1}^{s} D_iC_i$. N_i does not vanish for $u_1,\ldots,u_t,y_1,\ldots,y_s$. Indeed if it does, N_i, considered as a polynomial over F(i-1), is multiple of C_{i-1}; clearing denominators and replacing each of x_i by the indeterminate X_i, we get $DN_i = HC_{i-1}$ where D belongs to $K[U_1,\ldots,U_tY_1,\ldots,Y_{i-2}]$, this implies that C_{i-1} divides DN_i. The irreducibility of C_{i-1} implies C_{i-1} divides D which is impossible since D does not involve X_{i-1}. Hence $G(u_1,\ldots,u_t,y_1,\ldots,y_s) = 0$ i.e G belongs to I_s. Let show the other inclusion. Suppose A is in I_s, i.e. $A(u_1,\ldots,u_t,y_1,\ldots,y_s) = 0$. There exists an integer ss such that $N_s^{ss}A = D.C_s + R(X_1,\ldots,X_n)$ where $\deg_{Y_s}R < \deg_{Y_s}C_s$. If $\deg_{Y_s}R = 0$ then R is in J_{s-1}, this implies A is in J. If $\deg_{Y_s}R > 0$ then $R(u_1,\ldots,u_t,y_1,\ldots,y_s) = 0$, hence there exists a polynomial $R(u_1,\ldots,u_t,y_1,\ldots,y_{s-1},Y_s) \neq 0$ which is annuled by y_s and $\deg_{Y_s}R < \deg_{Y_s}C_s$ this is contrary to the assumption C_s is the minimal polynomial of y_s. Suppose $C'=(C_1',\ldots,C_s')$ is the extracted characteristic set and let J' be the ideal characteristically generated by C'. Let us show that C = C'. If $C_1 < C_1'$ then C_1 cannot be R-reduced to 0 using C'. If $C_1' < C_1$ then C_1' cannot be R-reduced to 0 using C. This implies $C_1 = C_1'$. Suppose i > 1 and $C_j = C_j'$ for j = 1,..,i-1. If $C_i < C_i'$ then C_i cannot be R-reduced to 0 using C' otherwise

C_i will R-reduce to 0 using C which is impossible. The same argument applies if $C_i' < C_i$. Thus, t = s and C = C'.

Theorem 4.2 Let $B=(B_1,\ldots,B_r)$ be a B-complete basis for an ideal I in $K[X_1,\ldots,X_n]$. Let $C=(C_1,\ldots,C_s)$ be the extracted characteristic set for I, and, let J be the ideal generated characteristically from C. Then,

1. I is included in J.
2. I is a prime ideal if and only if J is a prime ideal and I=J.
3. If J is a prime ideal then algorithm PrimeIdeal computes a B-complete basis for J.

Proof: 1. This is theorem 1.2 .

2. The only if part is evident. Let us show the if part, from 1.,we need to show J is included in I. If G belongs to J, then $(\prod_{i=1}^{s} N_i^{s_i})G \equiv \sum_{i=1}^{s} D_i C_i$. this implies $(\prod_{i=1}^{s} N_i^{s_i})G$ is in I. Since I is prime and $\prod_{i=1}^{s} N_i^{s_i}$ does not belong to I, G belongs to I.

3. The proof follows from the lemma 1 and lemma 2 below.

Lemma 1 Given J, the prime ideal characteristically generated by $\{C_1,\ldots,C_s\}$, let $L_0=(C_1,\ldots,C_s)$ and $L_i=\{G \mid N_{1+i} \bmod {}_sG \text{ is in } L_{i-1}\}$. Then,

(1) L_i , i = 1,2,..., form an ascending chain of ideals.

(2) there exists r such that $L_r = L_{r+s-1}$ and $J = L_r$.

Proof: (1) Obvious.

(2)Since $K[X_1,\ldots,X_n]$ is Noetherian, the ascending chain is finite, i.e. there exists r such that $L_r=L_{r+k}$ for all $k \geq 0$. To detect the end of the chain, it is sufficient to find $L_r = L_{r+1} = \ldots = L_{r+s-1}$ for some r. This is because if $L_r = L_{r+1} = \ldots = L_{r+i-1} \neq L_{r+i}$ then there exists a G in L_{r+i} such that G is not in L_{r+i-1}. But that G is in L_{r+i} implies N.G is in L_{r+i-1}, and $L_{r+i-2}=L_{r+i-1}$ implies G is in L_{r+i-1}, contrary to the assumption.

Let us now show that $J = L_r$. Let G be an element of J, there exists s_i such that $(\prod_{i=1}^{s} N_i^{s_i})G = \sum_{i=1}^{s} D_i C_i$, then G belongs to L_t where

$t = \max(s_1, \ldots, s_s)$. Hence G belongs to L_r. This implies J is included in L_r. To show the other inclusion, let G be in L_r, then $N_i G = \Sigma D_i H_i$ where H_i are the generators of L_{r-1}, by induction over r, H_i are in J, this implies $N_i G$ is in J, i.e. G in J.

Lemma 2 Given the ideal $L = (H_1, \ldots, H_m)$ and N a polynomial in $K[X_1, \ldots, X_n]$, we can compute a basis for $M = (L : N)$ i.e.
$M = \{G : NG = 0 \bmod L\}$

Proof: We need to solve the equation
$NG = P_1H_1 + \ldots + P_mH_m$, the generators of such equation can be computed explicitly (see [Sei]), an improvement for the solutions of such equation has been done in [Laz 1] and [Laz 2]. Thus the given equation has a finite set of generators which can be computed explicitly. However, for our case, if the degree of the intial is not small, we may change the ordering to compute the B-complete basis and we may get an easier computation to do.

Theorem 4.3 Let I be an ideal in $K[X_1, \ldots, X_n]$. Then, there exists an effective algorithm to test whether I is a prime ideal.

The proof follows from theorem 4.2 and theorem 4.3.

Examples
1. In Q[X,Y,Z], $I = (Y^3 - X^4, ZX - Y^2, ZY - X^3, Z^2 - X^2Y)$ is a prime ideal where $(Y^3 - X^4, ZX - Y)$ is a characteristic set for I. Note that $N_1=1$ and $N_2=X$.
$((Y^3 - X^4, ZX - Y) : X) = (Y^3 - X^4, ZX - Y, ZY - X^3)$.
$((Y^3 - X^4, ZX - Y^2, ZY - X^3) : X) = (Y^3 - X^4, ZX - Y^2, ZY - X^3, Z^2 - X^2Y)$.
This example is taken from [VdW], page 154, exercice 16.1. I is represented here by its B-complete basis following the pure lexicographical ordering $X < Y < Z$.

2. In Q[X,Y,Z], $X < Y < Z$, $I = (Y^2 + X^2, ZX - Y, ZY + X, Z^2 + 1)$ is a prime ideal where $(Y^2 + X^2, ZX - Y)$ is characteristic set for I.

3. In Q[X,Y], X < Y, I = $(XY^2 - 1, X^2Y - 1)$ is not a prime ideal because the B-complete basis of I is B = $(X^3 - 1, Y - X)$ and $X^3 - 1$ is reducible over Q.

Corollary If $(C_1,\ldots,C_s)$ is a characteristic set of a prime ideal I and $N_i = 1$, for i = 1,..,s , then I = $(C_1,\ldots,C_s)$. If $N_i = 1$, for i = 1,...s, and s = n then I is a maximal ideal.

5. Further Results

We have given an algorithm to test the primality and the maximality of a given ideal. This algorithm leads also to a way to compute the radical of an ideal as an intersection of prime ideals [Kan], chapter 6. Since we know how to compute the intersection of two ideals, we can compute the radical of an ideal. Taking advantage of the pure lexicographic ordering and the fact that if $\{C_1,\ldots,C_s\}$ is a characteristic set of an ideal I and if Y_i represent the highest variable in C_i, then the variables which are different from Y_i, are algebraically independent over K, we can conclude that if I is a prime ideal, then n-s is the dimension of I. However, if I is not prime, this need not hold. The dimension may be computed by considering permutations of the variable ordering during the computation of B-complete bases to ensure the minimality of s.

REFERENCE

[Buc 1] Buchberger, B.: "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms," ACM-SIGSAM Bulletin, 39, August, 1976, pp. 19-29.

[Buc 2] Buchberger, B.: "A Criterion for Detecting Unnecessary Reductions in the construction of Gröbner Basis," Proceedings of EUROSAM 79, Marseille, Springer Verlag Lecture Notes in Computer Science, Vol. 72, 1979, pp. 3-21.

[B.L.] Buchberger, B., and Loos, R.,"Algebraic Simplification," in Computer Algebra: Symbolic and Algebraic Computation. (Eds. B.Buchberger, G.E. Collins, R. Loos), Computing Suppl. 4, Springer Verlag, New York, 1982, pp. 11-43.

[Her] Hermann, G.: "Die Frage der endlich vielen schritte in der Theorie der Polynomideale," Math. Ann. 95, 1926, pp. 736-788.

[J.T.] Jenks R.D and Trager B.M.: "Newspad System Programming Language Manual", Mathematical Science Departement, IBM Thomas J. Watson Research Centre, Yorktown Heights, New York 10598.

[Kan] Kandri-Rody, A. : "Effective Methods in the Theory of Polynomial Ideals", PhD Thesis, Rensselaer Polytechnic Institute, Troy, NY, May 1984.

[K.K.] Kandri-Rody, A. and Kapur, D.: "On Relationship between Buchbeger's Gröbner Basis Algorithm and the Knuth-Bendix Completion Procedure," Unpublished GE CRD Technical Report, Nov. 1983.Schenactudy NY.

[Laz 1] Lazard, D.: "Resolution des systemes d'equations algebriques," Theoritical Computer Science, 15, 1981, pp. 77-110.

[Laz 2] Lazard, D.: "Algebre Lineaire sur $K[X_1,\ldots,X_n]$ et elimination," Bull.Soc.Math.France, 105, 1977, 165-190.

[Laz 3] Lazard, D.: "Commutative Algebra and Computer Algebra," EUROCAM'82, Lect. Notes in Comp. Sc. No 144(1982), pp. 40-48.

[Rit] Ritt,J.: "Differential Algebra," Colloquium Publications of the American Mathematical Society, Vol.XXXIII, 1950.

[Sei] Seidenberg, A.: "Construction in algebra," Trans. Amer. Math. Soc. 197 (1974), pp. 273-314.

[VdW] Van der Waerden, B.L., Modern Algebra, Vol. II, Fredrick Ungar Publishiing Co, New York, 1966.

[Z.S.] Zariski, O. and Samuel, P.:Commutative Algebra," Vol.1; , 1958, D.Van Nostrand Company, Inc.

ON THE MODULAR EQUATION OF ORDER 11

Erich Kaltofen
Department of Mathematical Sciences
Rensselaer Polytechnic Institute
Troy, New York 12181

and

Noriko Yui*
Department of Mathematics
University of Toronto
Toronto, Ontario M5S 1A1

Extended Abstract

1. INTRODUCTION

In this paper we give the explicit form of the modular equation of order 11: $\Phi_{11}(x,y) = 0$, computed using the computer algebra system MACSYMA [10].

The modular equation $\Phi_n(x,y) = 0$ $(n \geq 2)$ was introduced by Kronecker and used by Kronecker and Weber in the theory of complex multiplication to prove the (algebraic) integrality of the "class invariants". The equation $\Phi_n(x,y) = 0$ defines a (singular) affine curve over $\mathbf{Z}$. We hope that our result will be of some use for the study of its geometrical as well as arithmetical properties (e.g. irreducibility, singularities and desingularization).

In 1878, Smith [11] computed Φ_3 (see also Fricke [3, II.4]). Φ_5 was first computed by Berwick [1] in 1916. In 1974, Herrmann [5] determined Φ_7 explicitly. Yui [13] described an algorithm which we used in [7] to compute Φ_5 and Φ_7, being unaware of previous work. The equation we next aimed to determine was Φ_{11}. However, our algorithm when applied to Φ_{11} became inefficient, and in fact, we ran out of storage after 7 hours of VAX-780 CPU-time. Herrmann, using a slightly different algorithm, stated that his program would consume unjustifiable much of computing time to produce Φ_{11}. In spite of this pessimistic forecast, owing to a

* This research was partially supported by NSERC grant 3-661-114-30.

very lucky, so far unnoticed, mathematical property of the coefficients of the modular equation (see section 3) we were able to modify our algorithm in such a way that it requires much less space. The renewed attack, running in the background of UNIX on a VAX-780, finally produced Φ_{11}. Because of several system failures, which, though partial information was retained, destroyed our time keeping records, we cannot tell how much CPU-time was consumed. However, we are not too far off to say that the time was 20 ± 5 hours.

We present a hard copy of $\Phi_{11}(x, y)$ in the appendix. We factored out primes ≤ 1000 in the coefficients, but the remaining factors are still of substantial size (e.g. 60 digits). Readers who are interested in using Φ_{11} can obtain either a FORTRAN-style source file or a MACSYMA save module from the authors.

Since the coefficients of Φ_{11} are rather large, we felt that it was paramount to provide an independent test to check its correctness. In section 4, we describe such a test, based on a theorem of Kronecker (the Kronecker relation) (cf. Weber [12]) and our previous work [7] on the determination of class equations. This test verified our computation. We recommend that readers who are interested in using our result apply this test to avoid typographical or transmission errors when defining the polynomial.

2. MATHEMATICAL PREREQUISITES

We first introduce the elliptic modular j-invariant. For each complex number z with non-negative imaginary part, let $q = e^{2\pi i z}$ and let

$$E_4(z) = 1 + 240 \sum_{n=1}^{\infty} \sigma_3(n) q^n, \quad \sigma_3(n) = \sum_{\substack{t \mid n \\ t>0}} t^3.$$

Furthermore, let

$$\eta(z) = q^{\frac{1}{24}} \prod_{n=1}^{\infty} (1-q^n) = q^{\frac{1}{24}} \left[1 + \sum_{n=1}^{\infty} (-1)^n \left(q^{\frac{n(3n-1)}{2}} + q^{\frac{n(3n+1)}{2}} \right) \right].$$

The elliptic modular j-invariant $j(z)$ is defined as

$$j(z) = \left[\frac{E_4(z)}{\eta(z)^8} \right]^3.$$

We see that $j(z)$ has the q-expansion with integer coefficients

$$j(q) = \frac{1}{q} + 744 + 196884q + 21493760q^2 + 864299970q^3 + \cdots.$$

Now let $GL_2^+(\mathbf{Z})$ denote the set of 2×2 matrices with entries in $\mathbf{Z}$ and positive determinant. If $\alpha = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in GL_2^+(\mathbf{Z})$, we say that α is primitive if $\mathrm{GCD}(a, b, c, d) = 1$. For a prime p, let Δ_p^* denote the subset of $GL_2^+(\mathbf{Z})$ consisting of primitive matrices with determinant p. Then $SL_2(\mathbf{Z})$ acts on Δ_p^* (indeed, the multiplication on the left or right by elements of $SL_2(\mathbf{Z})$ maps Δ_p^* into itself). The left coset representatives of Δ_p^* modulo $SL_2(\mathbf{Z})$ are given by the set A of the $p+1$ matrices:

$$A = \{ \left(\begin{smallmatrix} p & 0 \\ 0 & 1 \end{smallmatrix}\right), \left(\begin{smallmatrix} 1 & i \\ 0 & p \end{smallmatrix}\right) \text{ with } 0 \le i < p \}.$$

For $\alpha = \left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right) \in A$ and for $z = x + \sqrt{-1}\,y$ with $y > 0$, we write $j \bullet \alpha$ for

$$(j \bullet \alpha)(z) = j(\alpha(z)) = j\left(\frac{az+b}{cz+d}\right),$$

and form the polynomial

$$\Phi_p(x) = \prod_{\alpha \in A} (x - j \bullet \alpha) = (x - j(pz)) \prod_{i=0}^{p-1} (x - j\left(\frac{z+i}{p}\right)) = x^{p+1} + \sum_{i=0}^{p} S_i(x)$$

with an indeterminate x, where $S_i(x)$ are the elementary symmetric functions in the $j \bullet \alpha$. Then the coefficients of $\Phi_p(x)$ are in $\mathbf{Z}[j]$, i.e., polynomials in $j(z)$ with integral coefficients. Thus, we may view $\Phi_p(x)$ as a polynomial in two variables x and j, and we write it as

$$\Phi_p(x) = \Phi_p(x, j) \in \mathbf{Z}[x, j].$$

We call this the modular polynomial of order p. The equation $\Phi_p(x, j) = 0$ is called the modular equation of order p.

The properties of $\Phi_p(x, j)$, which are relevant in our discussion, are collected in the following theorem.

Theorem (see, e.g. Weber [12, §69], Fricke [3, II.4] and Lang [9]).

(a) *$\Phi_p(x, j)$ is symmetric with respect to x and j, i.e., $\Phi_p(x, j) = \Phi_p(j, x)$.*

(b) $\Phi_p(x, x) \in \mathbb{Z}[x]$ *and the leading term is* $-x^{2p}$.

(c) $\Phi_p(x, j)$ *satisfies the congruence* $\Phi_p(x, j) \equiv (x^p - j)(x - j^p) \pmod{p}$.

By virtue of the properties of $\Phi_p(x, j)$ stated in the theorem above, we can write

$$\Phi_p(x, j) = (x^p - j)(x - j^p) - \sum_{m,n=0}^{p} c_{m,n} x^m j^n$$

where $c_{m,n}$ are integers such that

$$\begin{aligned} c_{m,n} &= c_{n,m} \\ c_{m,n} &\equiv 0 \pmod{p} \end{aligned} \quad \text{for all } m, n = 0, 1, \ldots, p$$

and $c_{p,p} = 0$. Putting all the above together we get the following result.

Theorem (Yui [13]). *Let* $x = j(pz)$. *Then*

$$0 = \Phi_p(x, j) = (x^p - j)(x - j^p) - p \sum_{m=1}^{p} \sum_{n=0}^{m-1} d_{m,n}(x^m j^n + x^n j^m) - p \sum_{m=0}^{p-1} d_{m,m} x^m j^m$$

where $d_{m,n}$ *and* $d_{m,m}$ *are integers.*

3. THE ALGORITHM

The above theorem is the basis for our algorithm. In order to determine $d_{m,n}$ and $d_{m,m}$, we substitute for j and x their q-expansions $j(q)$ and $j(q^p)$, and then equate the coefficients of the power of q in

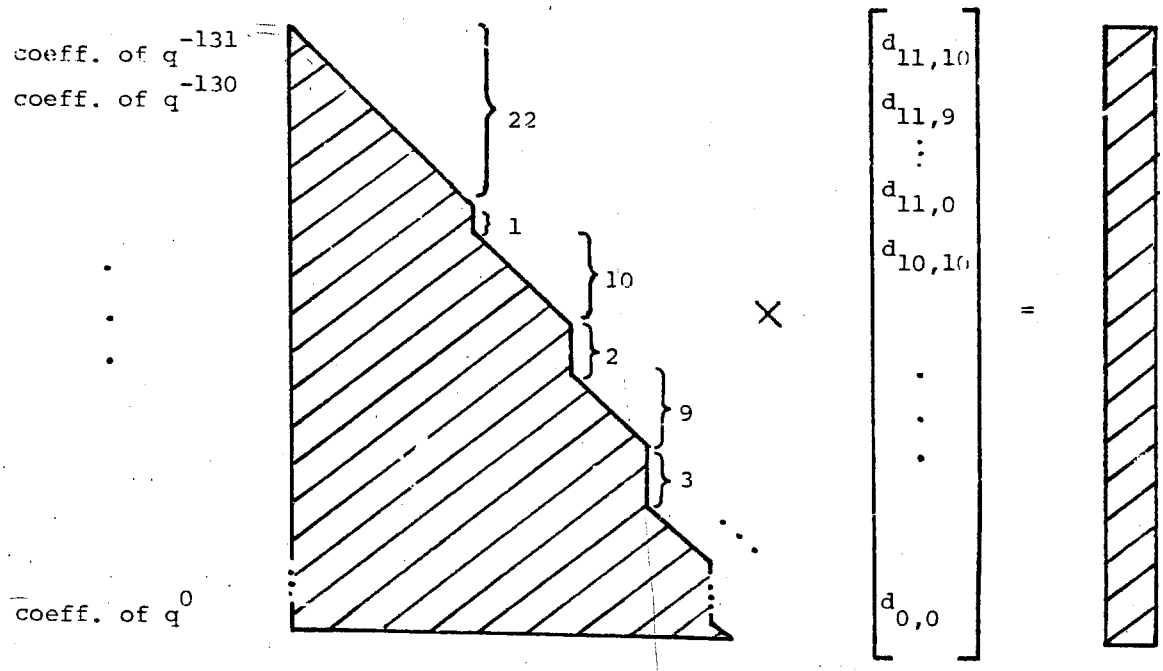
$$(j(q^p)^p - j(q))(j(q^p) - j(q)^p) =$$

$$p \sum_{m=1}^{p} \sum_{n=0}^{m-1} d_{m,n}(j(q^p)^m j(q)^n + j(q^p)^n j(q)^m) + p \sum_{m=0}^{p-1} d_{m,m} j(q^p)^m j(q)^m.$$

Note that in this expression the term of lowest order is q^{-p^2-p}, and that $d_{0,0}$ occurs in the term of order zero. Therefore, one needs the q-expansion of j to order $p^2 + p - 1$. This leaves us with a linear system of $(p^2 + 3p)/2$ variables and $p^2 + p$ equations (the lowest term coefficient is 0). Smith [11], Berwick [1], Herrmann [5] and Yui [13] all suggest setting up this linear system and solving it for $d_{m,n}$ and $d_{m,m}$.

For $p = 11$ we get an expansion with 132 terms in 77 variables whose

integer coefficients are typically 80 digits long. This expression is much too large, even before one attempts to solve the resulting system. However, on inspecting this system for Φ_5 and Φ_7, one quickly discovers the following computationally important fact: The resulting linear system is subdiagonal from lowest to highest coefficient, i.e. has the shape.



Though it might appear that this observation is important for the linear system solver, we make use of it long before that step. The idea is to set up the system for, say, the first 11 unknowns, $d_{11,10}, \ldots, d_{11,0}$. To do this we only need the q-expansion of j to order 10. After having found the correct values, we repeat this procedure for the next 11 unknowns, $d_{10,10}, \ldots, d_{10,0}$, now already using the values for the known coefficients. The q-expansion of j is needed to order 21, but the number of unknowns does not grow. In fact, one could introduce one variable at a time, instead of 11 new unknowns, thus reducing the storage requirement approximately 77-fold. Actually, we broke up the system into only two parts, since our available computing resources are abundant. Our observation also resolves an old question, namely, whether the linear system obtained from the q-expansion sufficiently determines the unknowns. It could have been that the system (of even infinitely many equations) was underdetermined, but this is not the case.

4. THE VERIFICATION

We use another property of the modular polynomial, known as the Kronecker relation.

Theorem (Kronecker, see e.g. Weber [12, 115]). *We have*

$$\Phi_p(x, x) = -\prod_D H_D(x)^{r'(D)}$$

where the quantities in the right-hand side are defined as follows. The product ranges over all $D \in \mathbb{Z}$, $D < 0$, such that $y^2 - Dx^2 = 4p$ has a solution $(x, y) \in \mathbb{Z} \times \mathbb{Z}$ with $x > 0$. Denoting by $r(D)$ the number of such solutions, the multiplicity $r'(D)$ is equal to $r(D)$ if $D < -4$, $r(D)/2$ if $D = -4$ and $r(D)/3$ if $D = -6$. $H_D(x)$ denotes the class equation for the imaginary quadratic order of discriminant D; it is an integral polynomial of degree h_D (the class number of order).

In case $-D$ is a prime (so necessarily $\equiv 3 \bmod 4$, since it must be a discriminant), we can determine the class equation $H_D(x)$ using the algorithm developed in Kaltofen and Yui [7]. For composite D the theory is more complicated and readers are referred to our full paper [8] for the explanation (see, also Weber [12] and Lang [9, §10]).

For $p = 11$, we list the discriminants D, class numbers h_D, and the corresponding class equations $H_D(x)$ with their multiplicities $r'(D)$ in the table below.

Our results satisfies the relation of Kronecker:

$$-\Phi_{11}(x,x) = H_{-7}(x)^2\, H_{-8}(x)^2\, H_{-11}(x)\, H_{-19}(x)^2\, H_{-28}(x)^2 \times H_{-35}(x)^2\, H_{-40}(x)^2\, H_{-43}(x)^2\, H_{-44}(x).$$

This verifies that the equation $\Phi_{11}(x, y) = 0$ is indeed correct.

D	$y^2-Dx^2=44$	h_D	z	$H_D(x)=\Pi(x-j(z))$	$r'(D)$
-7	$y=\pm4,\ x=2$	1	$\frac{1+\sqrt{-7}}{2}$	$x+3^3 5^3$	2
-8	$y=\pm6,\ x=1$	1	$3+\sqrt{-2}$	$x-2^6 5^3$	2
-11	$y=0,\ x=2$	1	$\frac{1+\sqrt{-11}}{2}$	$x+2^{15}$	1
-19	$y=\pm5,\ x=1$	1	$\frac{1+\sqrt{-19}}{2}$	$x+2^{15}3^3$	2
-28	$y=\pm4,\ x=1$	1	$2+\sqrt{-7}$	$x-3^3 5^3 17^3$	2
-35	$y=\pm3,\ x=1$	2	$\frac{1+\sqrt{-35}}{6}$ $\frac{3+\sqrt{-35}}{2}$	$x^2+2^{19}3^2 5^2 x$ $-2^{30}5^3$	2
-40	$y=\pm2,\ x=1$	2	$1+\sqrt{-40}$ $\frac{\sqrt{-10}}{2}$	$x^2-2^7 3^2 5^2 13\cdot 379 x$ $+2^{12}3^6 5^3 29^3$	2
-43	$y=\pm1,\ x=1$	1	$\frac{1+\sqrt{-43}}{2}$	$x+2^{18}3^3 5^3$	2
-44	$y=0,\ x=1$	3	$\sqrt{-11}$ $\frac{1+\sqrt{-11}}{4}$ $\frac{3+\sqrt{-11}}{4}$	$x^3-2^4 1709\cdot 41057 x^2$ $+2^8 3\cdot 11^4 24049 x$ $-2^{12}11^3 17^3 29^3$	1

5. ADDITIONAL PROPERTIES

It is known that $\Phi_p(x,y)$ is absolutely irreducible. This fact can be easily proved for $\Phi_{11}(x,y)$ with the help of a criterion developed by Kaltofen [6] stating that if $\Phi_{11}(x,y)$ is irreducible over $\mathbf{Q}$ and $\Phi_{11}(x,r)$ has a linear factor for some $r \in \mathbf{Q}$, then $\Phi_{11}(x,y)$ is absolutely irreducible. Choosing $r = j((1+\sqrt{-11})/2) = -2^{15}$ we get a linear factor $H_{-11}(x) = x + 2^{15}$ dividing $\Phi_{11}(x,-2^{15})$. The irreducibility of $\Phi_{11}(x,y)$ over $\mathbf{Q}$ may be verified directly on MACSYMA.

David Masser communicated to us that Paula Cohen [2] had recently established the following bound for the absolutely largest coefficient of Φ_n, $\|\Phi_n\|$:

$$\log \|\Phi_n\| = 6\psi(n)(\log n - 2\kappa(n) + O(1)) \quad \text{as } n \to \infty$$

where

$$\psi(n) = n \prod_{\substack{p \mid n \\ p \text{ prime}}} (1 + \frac{1}{p}), \quad \kappa(n) = \sum_{\substack{p \mid n \\ p \text{ prime}}} \frac{\log p}{p}.$$

Her estimate (ignoring $O(1)$ term) leads to $\log\|\Phi_{11}\| = 141.25$, whereas the true $\log\|\Phi_{11}\| = 289.09$. The difference by a factor of 2 can, perhaps, be explained by the fact that our n is rather small.

6. CONCLUSION

The modular equation $\Phi_{11}(x,y) = 0$ represents the (modular) algebraic correspondence

$$\{(j(z), j(\alpha(z))) \mid \alpha \in A,\ z = x + \sqrt{-1}\,y \text{ with } y > 0\} \subset \mathbf{P}^1 \times \mathbf{P}^1$$

and it defines an affine curve over $\mathbf{Z}$. After desingularization, this yields a (modular) elliptic curve with conductor 11. (For $p < 11$, $\Phi_p(x,y) = 0$, after desingularization, gives rise to a rational curve). It, therefore, seemed important to us to compute this equation explicitly to be used in future investigations.

Finally, we remark that the methods recently developed by Gross and Zagier [4] for computing values of class equations also seem to yield a very efficient algorithm for determining the explicit form of Φ_p with $p \leq 13$.

Acknowledgements

We would like to thank Professor David Masser for communicating the result of Paula Cohen on the estimate of the absolute coefficient of the modular equation.

REFERENCES

[1] W. E. H. Berwick, "An Invariant Modular Equation of the Fifth Order", Quart. J. Math. 47, 1916, pp. 94-103.

[2] P. Cohen, "On the Coefficients of the Transformation Polynomials for the Elliptic Modular Function", Proc. Cambridge Phil. Soc., to appear.

[3] R. Fricke, Lehrbuch der Algebra Bd. 3, Braunschweig, 1928.

[4] B. Gross and D. Zagier, Private communication.

[5] O. Herrmann, "Über die Berechnung der Fourier Koeffizienten der Funktion $j(\tau)$", J. Reine Angew. Math. 274/275, 1974, pp.187-195.

[6] E. Kaltofen, "Fast Parallel Absolute Irreducibility Testing", manuscript, 1983.

[7] E. Kaltofen and N. Yui, "Explicit Construction of Hilbert Class Fields of Imaginary Quadratic Fields with Class Numbers 7 and 11", EUROSAM '84, Springer Lecture Notes in Comp. Sci., to appear.

[8] E. Kaltofen and N. Yui, "The Modular Equation of Order 11 and its Arithmetical Properties", manuscript 1984.

[9] S. Lang, Elliptic Functions, Addison-Wesley Publishing Co. Inc., 1973.

[10] MACSYMA, Reference Manual v.1 and 2, The Mathlab Group, Laboratory for Computer Science, MIT, 1983.

[11] H.J.S. Smith, "Note on a Modular Equation for the Transformation of the Third Order", Proc. London Math. Soc., v.10, 1878, pp.87-91.

[12] H. Weber, Lehrbuch der Algebra, Bd. 3, Braunschweig, 1908.

[13] N. Yui, "Explicit Form of the Modular Equation", J. Reine Angew. Math. 299/300, 1978, pp.185-200.

APPENDIX

$$\Phi_{11}(x,y) = 0 =$$

$$x^{12} + y^{12} + 2^3 \cdot 3 \cdot 11 \cdot 31\,(y^{10}x^{11} + y^{11}x^{10})$$

$$- 71411\ \ 2^2 \cdot 3^2 \cdot 11\,(y^9x^{11} + y^{11}x^9)$$

$$+ 152519383 \cdot 2^5 \cdot 11\,(y^8x^{11} + y^{11}x^8)$$

$$- 185027238353 \cdot 2 \cdot 3 \cdot 5\ \ 11\,(y^7x^{11} + y^{11}x^7)$$

$$+ 2443204381063 \cdot 2^4 \cdot 3^2 \cdot 11^2\,(y^6x^{11} + y^{11}x^6)$$

$$- 803967223898807 \cdot 2^3 \cdot 11^2 \cdot 23\,(y^5x^{11} + y^{11}x^5)$$

$$+ 24009920521667 \cdot 2^6 \cdot 3 \cdot 5 \cdot 11^2 \cdot 23 \cdot 67\,(y^4x^{11} + y^{11}x^4)$$

$$- 2491107819565653 1 \cdot 3^2 \cdot 5 \cdot 11^2 \cdot 47 \cdot 83\,(y^3x^{11} + y^{11}x^3)$$

$$+ 1302864869715323531 \cdot 2^3 \cdot 5^2 \cdot 11^2 \cdot 863\,(y^2x^{11} + y^{11}x^2)$$

$$- 2835561656197600834891\ \ 2^2 \cdot 3 \cdot 7\ \ 11^2\ \ 13\,(yx^{11} + y^{11}x)$$

$$+ 204842039071 \cdot 2^{15} \cdot 3^4 \cdot 5^5 \cdot 11 \cdot 29 \cdot 547\,(x^{11} + y^{11}) - y^{11} \cdot x^{11}$$

$$+ 304071601918951 \cdot 2^6 \cdot 3^2 \cdot 7 \cdot 11^3 \cdot 59 \cdot 313\,(y^9x^{10} + y^{10}x^9)$$

$$+ 213632857915153125253728 1237 \cdot 2^4 \cdot 3 \cdot 11^2\,(y^8x^{10} + y^{10}x^8)$$

$$+ 139002462396449980652371073 3 \cdot 2^4 \cdot 5 \cdot 7^2 \cdot 11^3 \cdot 89\,(y^7x^{10} + y^{10}x^7)$$

$$+ 216211652871283314750552744726722052 09 \cdot 3 \cdot 11^2\,(y^6x^{10} + y^{10}x^6)$$

$$+ 1198618682080385582294052403766384436 3 \cdot 2^4 \cdot 3 \cdot 5 \cdot 11^2 \cdot 83\,(y^5x^{10} + y^{10}x^5)$$

$$+ 213507160242946938854998919923028533300 1$$

$$\cdot 2^3 \cdot 3 \cdot 5 \cdot 7 \cdot 11^2 \cdot 163\,(y^4x^{10} + y^{10}x^4)$$

$$+ 400943691425850890698895726587814069 7$$

$$\cdot 2^3 \cdot 3^2 \cdot 5^2 \cdot 11^2 \cdot 13 \cdot 41 \cdot 97 \cdot 313\,(y^3x^{10} + y^{10}x^3)$$

$$+ 126413487710766863183099189805278133505334699 67$$

$$\cdot 2 \cdot 3 \cdot 11^2\ \ 173\,(y^2x^{10} + y^{10}x^2)$$

$$+ 570691334149011771523086094755653851$$

$$\cdot 2^{16} \cdot 3^4 \cdot 5^3 \cdot 7 \cdot 11^2 \cdot 53 \cdot 787\,(yx^{10} + y^{10}x)$$

$$+ 1770507108874088630732300810321 9 \cdot 2^{32} \cdot 3^7 \cdot 5^8 \cdot 11 \cdot 41\,(x^{10} + y^{10})$$

$$+2310043787617\cdot 2\cdot 3\cdot 7\ \ 11^2\ \ 137y^{10}x^{10}$$

$$+9589861526688745956466782959500279774 9\cdot 2^3\cdot 3^2\cdot 11^2\cdot 29(y^8x^9+y^9x^8)$$

$$-22702385234737829463400035283393402548184 7$$

$$\cdot 2^2\cdot 3\cdot 5^3\cdot 11^2\cdot 17\cdot 73(y^7x^9+y^9x^7)$$

$$+3982182104234155991126030618219997181291052972 53$$

$$\cdot 2^4\cdot 5\cdot 11^2\cdot 37\cdot 103(y^6x^9+y^9x^6)$$

$$-1563634895691677537445996250662343904462533653604356 1$$

$$\cdot 2^2\cdot 3^2\cdot 5\cdot 11^2\cdot 23\cdot 127(y^5x^9+y^9x^5)$$

$$+5084592561048113954497357458608235518570914753005936792557$$

$$\cdot 2^5\cdot 3^2\cdot 5^3\cdot 11^2(y^4x^9+y^9x^4)$$

$$-2631641938847849826248466004202500462392871407080228827723$$

$$\cdot 2^2\cdot 5\cdot 7\cdot 11^2\cdot 73\cdot 97\cdot 631(y^3x^9+y^9x^3)$$

$$+3528184458872697450519006997940936790448213493341999219 1$$

$$\cdot 2^{16}\cdot 3^4\cdot 5^5\cdot 11^2\cdot 37\cdot 307(y^2x^9+y^9x^2)$$

$$-9982990784249350826214138937607607606311783622981742 9$$

$$\cdot 2^{31}\cdot 3^7\cdot 5^7\cdot 7\cdot 11^2\cdot 47(yx^9+y^9x)$$

$$+6201360168079554794154776324781254624005839317983$$

$$\cdot 2^{47}\cdot 3^9\cdot 5^{10}\cdot 11\cdot 523(x^9+y^9)$$

$$-5549102003290133646182846491\cdot 11^2\cdot 23\cdot 107\cdot 347y^9x^9$$

$$+6538603459601786748399998328460836913035658866376243$$

$$\cdot 2^5\cdot 3^2\cdot 5\cdot 11^2\cdot 13\cdot 43(y^7x^8+y^8x^7)$$

$$+3957582333464824304569977175726251437433645305141024410183 7$$

$$\cdot 2^3\cdot 3^3\cdot 5\cdot 11^2\cdot 191(y^8x^8+y^8x^8)$$

$$+5518494669494371174108555957222990496474636079801097960793403969 1$$

$$\cdot 2^5\cdot 3\cdot 5^2\cdot 11^2\cdot 13(y^5x^8+y^8x^5)$$

$$+4766789376342740059073368224664076252030503825753340470244227342819 7$$

$$\cdot 3^2\cdot 5\cdot 11^2\cdot 137\cdot 239(y^4x^8+y^8x^4)$$

$$+ 797488113325686051207144427556019807967175286756951523267265691$$
$$\cdot 2^{15} \cdot 3^5 \cdot 5^6 \cdot 11^2 \cdot 37 \cdot 179 (y^3x^8 + y^8x^3)$$
$$+ 8009960374040182967007753370486902998209765597116211138783 9$$
$$\cdot 2^{30} \cdot 3^9 \cdot 5^8 \cdot 11^2 \cdot 19 \cdot 113 (y^2x^8 + y^8x^2)$$
$$+ 697758403620157678136473723132640683814946939452754144989$$
$$\cdot 2^{45} \cdot 3^{11} \cdot 5^{10} \cdot 11^2 \cdot 13 (yx^8 + y^8x)$$
$$+ 68373043210852121539422934230893108139260834914441$$
$$\cdot 2^{61} \cdot 3^{14} \cdot 5^{12} \cdot 11 \cdot 661 \cdot (x^8 + y^8)$$
$$+ 55117514896231449147068983186871964597631244517 1 \cdot 2 \cdot 3 \cdot 11^2 \cdot 73 y^8x^8$$
$$+ 19624159586613730913255818360523449571328983547789405044812271453111$$
$$\cdot 2^4 \cdot 3^2 \cdot 5^2 \cdot 11^2 \cdot 29 (y^8x^7 + y^7x^6)$$
$$- 37903034895031327805539352832244877175850724280506390275987988311256352811 7$$
$$\cdot 2^3 \cdot 3^2 \cdot 11^2 \cdot 23 (y^5x^7 + y^7x^5)$$
$$+ 16082457740673086027378938716502400173773257619860927216857998804368 81$$
$$\cdot 2^{16} \cdot 3^5 \cdot 5^5 \cdot 11^2 \cdot 307 (y^4x^7 + y^7x^4)$$
$$- 15995784352741545183058918094143654855207137198627849332244101520 7$$
$$\cdot 2^{31} \cdot 3^8 \cdot 5^7 \cdot 11^2 \cdot 17 \cdot 81 (y^3x^7 + y^7x^3)$$
$$+ 18496189672180702475002689829123548285937055486002772199048899$$
$$\cdot 2^{48} \cdot 3^{11} \cdot 5^{10} \cdot 11^2 \cdot 41 (y^2x^7 + y^7x^2)$$
$$- 32320503289753251520227540443699131147932410508589007797$$
$$\cdot 2^{62} \cdot 3^{14} \cdot 5^{12} \cdot 11^2 \cdot 31 \cdot 37 (yx^7 + y^7x)$$
$$+ 68700902171492018107018221126937879758851427760149 1$$
$$\cdot 2^{76} \cdot 3^{17} \cdot 5^{17} \cdot 11^2 (x^7 + y^7)$$
$$- 19677003744712708547039589241259134968401021358155471217901735 7$$
$$\cdot 2 \cdot 3 \cdot 5 \cdot 7 \cdot 11^2 \cdot 13 y^7x^7$$
$$+ 52322745774174889647871266229346822948860240346290413406550886598300 69$$
$$\cdot 2^{19} \cdot 3^5 \cdot 5^5 \cdot 7 \cdot 11^2 \cdot 127 (y^5x^8 + y^8x^5)$$

$$+ 263959782232166076062398655111513718610325469385388072240826580074 9$$

$$\cdot 2^{30} \cdot 3^{8} \cdot 5^{7} \cdot 11^{2} \cdot 41 \cdot 269 \left(y^{4}x^{6} + y^{6}x^{4}\right)$$

$$- 339000645059197744157498717388588189582645439822787277712664 9$$

$$\cdot 2^{45} \cdot 3^{10} \cdot 5^{10} \cdot 11^{2} \cdot 13 \cdot 191 \cdot 349 \left(y^{3}x^{6} + y^{6}x^{3}\right)$$

$$+ 464501640037049186270780295143604166740365559210853212304531$$

$$\cdot 2^{60} \cdot 3^{14} \cdot 5^{12} \cdot 11^{2} \cdot 13 \cdot 31 \left(y^{2}x^{6} + y^{6}x^{2}\right)$$

$$- 211669775305514206517804008879242929069974069187753077301$$

$$\cdot 2^{75} \cdot 3^{17} \cdot 5^{17} \cdot 11^{2} \left(yx^{6} + y^{6}x\right)$$

$$+ 270909647855313899315632002810352263119290522273 03$$

$$\cdot 2^{92} \cdot 3^{19} \cdot 5^{20} \cdot 11^{2} \cdot 53 \left(x^{6} + y^{6}\right)$$

$$+ 17929822479611682569015747211559561628347489460983284512397278954317612125 1$$

$$\cdot 2^{2} \cdot 3 \cdot 7 \cdot 11^{2} \cdot 641\, y^{6}x^{6}$$

$$- 604196099441808431074554236446836961540340911056070363759033226213 3$$

$$\cdot 2^{47} \cdot 3^{11} \cdot 5^{10} \cdot 11^{2} \left(y^{4}x^{5} + y^{5}x^{4}\right)$$

$$- 509894443206950279118253108211746288670902754575995513741954271$$

$$\cdot 2^{64} \cdot 3^{14} \cdot 5^{12} \cdot 11^{2} \left(y^{3}x^{5} + y^{5}x^{3}\right)$$

$$+ 288273875757574108718257118868547016275500534534111371$$

$$\cdot 2^{78} \cdot 3^{18} \cdot 5^{19} \cdot 11^{2} \cdot 499 \left(y^{2}x^{5} + y^{5}x^{2}\right)$$

$$- 554253659534181630845812048633091751608705133761316 1$$

$$\cdot 2^{91} \cdot 3^{20} \cdot 5^{17} \cdot 11^{2} \cdot 71 \left(yx^{5} + y^{5}x\right)$$

$$- 165347689550314533263639657466185294828598961 9$$

$$\cdot 2^{107} \cdot 3^{23} \cdot 5^{22} \cdot 7 \cdot 11^{2} \cdot 61 \left(x^{5} + y^{5}\right)$$

$$- 192262416122548321953137134772767570206376697307986458387807452815953$$

$$\cdot 2^{33} \cdot 3^{9} \cdot 5^{7} \cdot 7^{2} \cdot 11^{2} y^{5}x^{5}$$

$$- 231269172271974353664230209820036871044329015363345898573 1$$

$$\cdot 2^{75} \cdot 3^{17} \cdot 5^{17} \cdot 7^{2} \cdot 11^{2} \left(y^{3}x^{4} + y^{4}x^{3}\right)$$

$$+ 56877893268414915073480651678485215370764693117258811$$

$$\cdot 2^{92} \cdot 3^{20} \cdot 5^{19} \cdot 11^2 \cdot 167 (y^2x^4 + y^4x^2)$$

$$+ 5415225397677834475422807358887936494076700872475 9$$

$$2^{105} \cdot 3^{23} \cdot 5^{22} \cdot 11^2 (yx^4 + y^4x)$$

$$+ 17939475983520239084276804767677227923260621 37$$

$$\cdot 2^{120} \cdot 3^{26} \cdot 5^{24} \cdot 11^2 (x^4 + y^4)$$

$$+ 7440188171658385376358337002121255117746294641223361 39999$$

$$\cdot 2^{61} \cdot 3^{14} \cdot 5^{12} \cdot 11^2 \ 13 \cdot 71^2 \cdot 947 y^4x^4$$

$$+ 4985689196260039104574994880749571567063178837 79$$

$$\cdot 2^{105} \cdot 3^{23} \cdot 5^{22} \cdot 11^2 \cdot 31 \cdot 61 (y^2x^3 + y^3x^2)$$

$$+ 4584255170679832459479690138586689073359163$$

$$\cdot 2^{122} \cdot 3^{26} \cdot 5^{25} \cdot 11^2 \cdot 13 \cdot 17 (yx^3 + y^3x)$$

$$- 109883762119073189635270552234428422 17 \cdot 2^{135} \cdot 3^{27} \cdot 5^{29} \cdot 11^3 \cdot 373 (x^3 + y^3)$$

$$- 17569166457345972065937392831868460372022052147353$$

$$\cdot 2^{92} \cdot 3^{19} \cdot 5^{19} \cdot 11^2 \cdot 17 \cdot 263 \cdot 887 y^3x^3$$

$$- 37183159968727376980451651056501135078603$$

$$\cdot 2^{135} \cdot 3^{28} \cdot 5^{30} \cdot 11^2 (yx^2 + y^2x)$$

$$+ 16465369559553482216627 39 \cdot 2^{153} \cdot 3^{31} \cdot 5^{33} \cdot 7 \cdot 11^3 \cdot 17^3 \cdot 29^3 (x^2 + y^2)$$

$$- 26133502139812394794832987638425967293174813$$

$$\cdot 2^{121} \cdot 3^{27} \cdot 5^{24} \cdot 11^2 \cdot 79 y^2x^2$$

$$- 162899624593 \cdot 2^{171} \cdot 3^{34} \cdot 5^{34} \cdot 11^3 \cdot 17^6 \cdot 29^6 \cdot 41 (x + y)$$

$$+ 26094174253158533018911091 \cdot 2^{153} \cdot 3^{31} \cdot 5^{31} \cdot 17^3 \cdot 29^3 \cdot 139 \cdot 487 yx$$

$$+ 2^{189} \cdot 3^{36} \cdot 5^{36} \cdot 11^3 \cdot 17^9 \cdot 29^9$$

New Foundations for Computer Algebra

The Ubiquity of Universal Techniques in Computer Algebra

William G. Dubuque
Symbolics, Inc.
Cambridge Research Center

Abstract

Has 1984 arrived for the field of computer algebra? Newspeak (Berkeley) and Newspad (IBM) are two recently developed state of the art computer algebra systems whose languages are largely founded upon universal ideas borrowed from category theory. Category theory is a relatively new framework for discussing and analyzing many ideas common to diverse mathematical theories. The startling success of category theory as a language for unifying a large core of mathematics has prompted many mathematicians to learn and reason in it. Indeed, there are even proposals for categorical foundations of mathematics to replace the Oldspeak of set theory. However, category theory is not universally accepted by the mathematics community. There are many mathematicians who are very uncomfortable working at such a high level of abstraction as is common in many categorical endeavors. Some are even quick to dismiss the field as a collection of "abstract nonsense". Nonetheless one would be foolish to totally dismiss a theory which has produced so many new insightful results towards the solution of problems resistant to attack by classical methods. A large amount of such criticism stems from the fact that such critics claim to lose a considerable amount of intuition in such an abstract setting. Results that might be considered as "natural" in a category theoretic setting may seem quite unintuitive to those classically trained. An interesting issue thus arises regarding what influence ones training has on his mathematical reasoning capabilities and intuition. It is fairly well known in the mathematical community that differently trained mathematicians will often reason in (seemingly) quite distinct ways. Hence it may be the case that those trained in certain ways will be so biased that they will never feel comfortable working in a categorical setting. However, this valid problem for a classically trained mathematician may actually turn out to be a blessing for computer algebra in view of the fact that no existing computer algebra system has biased intuition. Indeed, due to the lack of suitably powerful formalisms for expressing mathematical intuition, most computer algebra systems capture none of the intuitive knowledge of mathematics. In this talk we will argue that by adapting and enhancing methods from category theory, universal algebra and model theory it is possible to capture much more mathematical intuition in a computer algebra system than is possible in a system based on classical methods. Moreover, one need not fear an Orwellian system since it is quite possible (indeed, desirable) to construct such a system without imposing a totalitarian view of universality on the user. Put more

simply, the system can be designed so that the user can comfortably work at whatever level of abstraction he is at home with. To utilize such universal techniques in an effective manner it is necessary to incorporate basic universal concepts more cohesively at the foundational level. All existing computer algebra sytems which employ universal methods do so only minimally and then so only in the area of language design. As such, they capture only syntactic information and ignore the important semantic components which are crucial to modelling intuitive ideas. We will describe an approach which has as one of its primary goals the aim to formally capture as much intuitive mathematical information as possible. To illustrate the ubiquity of this approach as a powerful tool for computer-assisted mathematical problem solving we will discuss a number of diverse problems solvable by our techniques but beyond the scope of classical computer algebra systems.

The Role of Maintenance in Knowledge Programming

Dr. James E. O'Dell
Symbolics, Inc.
Cambridge, MA 02139

Abstract

In this paper we describe several of the differences between MACSYMA and BASIC, emphasizing the evolutionary nature of a Knowledge Based program such as MACSYMA and contrasting it to the fixed nature of a traditional application program. We conjecture that Knowledge Based programs in general have a different life cycle and that the traditional notion of maintenance programming may not be applicable. We make the case that the term Knowledge Engineering may be a more appropriate term to describe the ongoing involvement of a programmer with a Knowledge Based program.

1. Comparisons with the BASIC programming language

What is a Knowledge Based program? What expectations do the people who use one have? How are the tasks performed by programmers of Knowledge based programs differ from those performed by the maintainers of traditional programs? Some important light can be shed on these questions by taking a look at a program like MACSYMA and comparing it with a program like a BASIC interpreter. Out of this comparison will come a plausible explanation of the term Knowledge Based, which directly implies that the traditional concept of maintenance programming may not apply in the domain of Knowledge Based programs.

To most people who use a system like MACSYMA, the difference between it and a programming language like BASIC is immediate and obvious, MACSYMA acts intelligently. Take a look at a simple problem in the common programming language BASIC:

```
100 PRINT "ENTER A NUMBER" , READ A
200 B = A + A
300 PRINT A , "PLUS" , A , " IS " , B
400 END
```

Everyone can understand this program, it reads a number, adds it to itself and then prints out the results. For a computer to be able to execute this program, another program called an interpreter will have to be written which takes the characters as written and transforms them into instructions executable by a given piece of hardware. Hardware must be constructed to execute the instructions, a display must be created to show the results and some sort of keyboard or input device to enter the program. Even from this admittedly sketchy description it is pretty obvious that constructing a computer to execute a BASIC program would take quite a bit more knowledge and time than most of us have.

Even if we neglect the hardware expertise required and concentrate only on the software necessary to run a simple BASIC program the amount of knowledge that has to be distilled into a program is still quite large.

```
1) The syntax of BASIC
2) How to READ and PRINT in a manner acceptable to a human.
3) Understanding of the properties of numbers.
```

If we subject MACSYMA to the same close scrutiny we will find that, not only does MACSYMA require all of the same sorts of information that BASIC requires, it also relies on many mathematical facts. In fact by simply rearranging two lines of the above BASIC program and translating it to the MACSYMA language, we can create a program whose output is exactly the same as before but will require algebraic knowledge to run. Look at this simple MACSYMA program:

```
B : A + A;
A : READ( "ENTER A NUMBER");
PRINT( A , "PLUS" , A , " IS " , EV(B));
```

The program causes the variable B to be set to the simplification of the symbolic expression A+A, that is 2*A. It then reads a value for A and prints out the result of evaluating 2*A given the value of A previously prompted for. By rearranging 2 lines of a BASIC program and translating it to MACSYMA we have caused its execution to depend on algebraic simplification knowledge available only in a system such as MACSYMA. Although the example given is trivial, it makes the point that a user of both systems might find it hard to distinguish which was the Knowledge Based program.

2. What is a Knowledge Based Program

If we accept the naive meaning of the term Knowledge Based one might be led the think that it means any program which embodies a given set of facts and attempts to use them in its functioning. As we have seen though, even running a Basic program entails the understanding of a tremendous number of facts. It is simply not true, contrary to what some would have you believe, that MACSYMA is constructed based on knowledge and BASIC is constructed out of ignorance. It is simply not true that programs based on knowledge are something new. It has been at the heart of programming since its inception. What may indeed be true is that the term Knowledge Based is taking on a meaning different than one might think. We know that there is a difference between MACSYMA and BASIC, but that difference is not due to the fact that MACSYMA is based on knowledge and BASIC while BASIC is not.

One difference between the examples we have been studying is that BASIC only has knowledge about BASIC programs and programming while MACSYMA includes in its knowledge base facts about mathematics. It is one of the few computer systems that attempts to integrate knowledge about programs and programming and expert knowledge of an area of knowledge that is not computer oriented. By allowing programs to possess knowledge about domains such as "The Field of Mathematics" we have created an almost unending task for the programmers of such a system. The programs can take on mammoth proportions and have hundreds of person years invested in them without exhaustively covering the field of

Mathematics. As a direct result of this it often results that a large Knowledge Based system cannot be understood by a single person. To reiterate, here are the some of the characteristics of a Knowledge Based program we consider important:

1) Use knowledge outside the field of Computer Science.
2) Huge programs: not understood by any one person.
3) Incomplete: leaving large "Holes" in the Knowledge Base.

3. Maintenance and Dealing with incompleteness

Probably the most crucial feature of MACSYMA in relation to maintenance is that by some metric it can be judged incomplete and unfinished. By no means is this meant to be a critical remark about the MACSYMA system but a simple observation of the nature of all current Knowledge Based programs. Given the development time of a program of this type it is essential that they be available for general use during the development stage of their life. In fact many of the deficiencies in a program such as MACSYMA may not even become apparent until a user starts to explore a new area of mathematics for which the program has never been used before. There often exists a kind of Creative Tension between the users exploring new areas of the programs capabilities and the programmers trying to provide new areas of expertise.

To understand this better look at the simple example of the evolution of knowledge of complex numbers that has covered MACSYMA's first decade. Several years ago, an exploration of much of the code in MACSYMA that was trying to handle imaginary numbers would have revealed something very startling to even a novice mathematician. In most places the code was simply checking explicitly for the square root of -1 to determine if an expression was real or imaginary. The was done not because the early workers on MACSYMA were ignorant of mathematics but because they felt it necessary to get the program running. To do this they were willing to sacrifice mathematical accuracy. As the problems presented to MACSYMA grew in complexity this incorrect mathematics began to be seen as an area which needed work. While today a perusal of the MACSYMA code dealing with imaginary numbers would certainly reveal more accurate mathematics the job is by no means done. Users are now asking for correct handling of logarithms of complex numbers for example. This example of interplay between the users of a system and the programmers is the exciting part of the work. Of course there is always the more mundane problem of correcting simple programming bugs which get introduced during the process of writing the new code.

While modifications such as the above are proceeding, new versions of MACSYMA must strive to provide no less functionality than previous versions. In many respects the users of MACSYMA are using an unfinished and incomplete piece of software because of the constant growth and change. Most people would not think about using a BASIC interpreter that was judged incomplete.

MACSYMA is evolutionary in nature. It takes a long time to program, even to figure out how to program, some of the mathematics that MACSYMA is expected to know about. This means that the users and programmers of such a system must both work in a constantly changing environment. The goal of a maintenance programmer in such a situation is to continually add knowledge to the body of the program while at the same time maintaining a

sense of continuity between old versions of the program and new ones. This leads us to the conjecture that Knowledge Based programs in general have a different life cycle from "Traditional" programs. Because part of the job of programming a system such as MACSYMA involves adding new knowledge to the program and not simply correcting programming errors, the traditional notion of maintenance programming is clearly inappropriate. Traditional maintenance programming is only one part of the work in Engineering a large knowledge based program.

In this sense we feel the term Knowledge Engineering may be a more appropriate term to describe the ongoing involvement of a programmer with a Knowledge Based program.

volume and magnetic decay is described by the characteristic function $e^{-(\frac{t}{\tau})^{\alpha}}$. In view of this considerable practical potential and the present dearth of convenient computational methods, it seemed worthwhile to examine algorithms suitable for use in fitting data. Two series expansions for $Q_{\alpha}(z)$ are known. For small z ;

$$Q_{\alpha}(z) = \frac{1}{\pi\alpha}\sum_{n=0}^{\infty}\frac{(-)^n z^{2n}}{2n!}\Gamma(\frac{2n+1}{\alpha}) \tag{7}$$

(A result known to Cauchy.) This series diverges but may be considered asymptotic for small z. It is useful for high temperature solids where α is still small but the relaxation rate is large. A convergent large z series for $Q_{\alpha}(z)$ was found by Wintner; [2]

$$Q_{\alpha}(z) = \frac{\alpha}{\pi}\sum_{n=0}^{\infty}\frac{(-)^n}{(n!z^{\alpha(n+1)+1})}\Gamma(\alpha[n+1])\sin[\frac{\pi\alpha(n+1)}{2}] \tag{8}$$

Equation 8 is convergent for $0 < \alpha < 1$ and therefore in the range of interest for glassy solids. Unfortunately, several hundred terms of the series may be needed for α as large as 0.2, necessitating a search for better methods.

Using Macsyma, we have found (2) new algorithms for the sine, cosine and Laplace tranforms of the characteristic function $e^{-(\frac{t}{\tau})^{\alpha}}$. The inverse Laplace transform gives the so-called distribution of relaxation times corresponding to $e^{-(\frac{t}{\tau})^{\alpha}}$, which is itself a stable density. For very small α, $Q_{\alpha}(z)$ mimicks a lognormal density.

The stable densities are closely related to the theory of fractals. (3) Their non-analytic momentless properties reveal the lack of a normalizing time scale, and in fact they represent hierarchically-clustered time scales for glass defect migrations. The set of hopping times for bond motion is, on average, a Cantor set in the case when the charcateristic exponent , α falls below 1. α is a fractal or Hausdorff dimension of the set of hopping times. CTRW models leading to such behavior have recently been found. (4)

REFERENCES

1. A summary with references to early literature may be found in E.W. Montroll and B.J. West, Studies in Statistical Mechanics, Vol. VII (Eds. E.W. Montroll and J.L. Lebowitz) (North Holland, New York, 1979) chapter 2.

2. E.W. Montroll and J.T. Bendler, J. Statistical Physics, 34, 129 (1984).

3. B.B. Mandelbrot, The Fractal Geometry of Nature (Freeman, New York, 1982).

4. J.T. Bendler, J. Statistical Physics (In Press).

AN APPROXIMATE SOLUTION OF AN INTEGRAL EQUATION THAT ARISES IN THE DESIGN OF MAGNETIC FIELD COILS

M.A. Hussain
Information System Operation

and

J.F. Schenck
Electronic Systems Programs Operation

General Electric Company

Abstract

In this paper we reconsider the integral equation that arises in coil design. The form of the dominant kernel allows two distinct transformations corresponding to short and long coil approximations. Fourier expansions and Schwinger's transformation lend to closed-form solutions for the approximated kernel. Symbolic computation is used to carry out tedious algebra, and an explicit form of singularities is recovered. The method can be applied to any order of coil design. Results compare well with numerical solutions and lead to some practical coil designs.

DERIVATION OF THE INTEGRAL EQUATION

The complete representation of the magnetostatic field as well as the derivation of the integral equation are given in Ref. [1]. For completeness, a brief outline is given below (see Fig. 1).

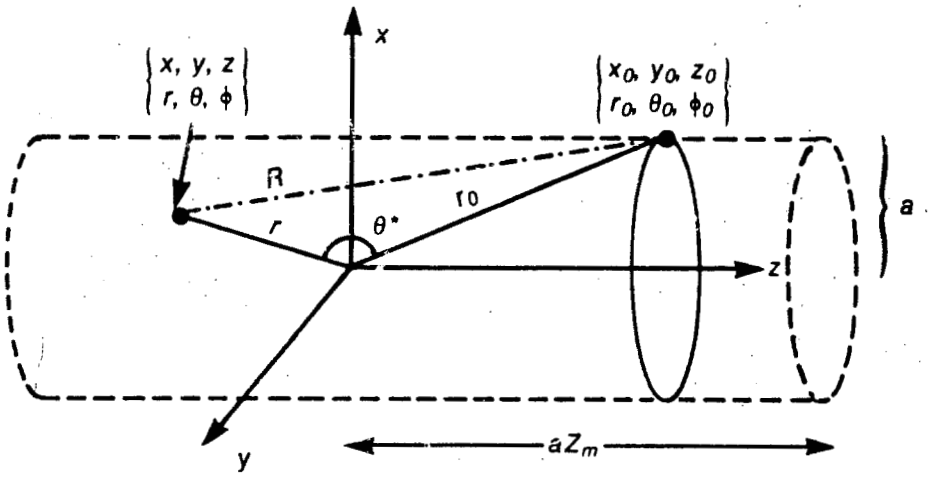


Figure 1. Representation of field and source points for cylindrical winding.

Using the conventional notations, the magnetic field $\bar{B}$ is given by

$$\bar{B} = \frac{\mu_o}{4\pi} \int \bar{\lambda} \times \nabla \left(\frac{1}{R} \right) dA \tag{1}$$

where $\bar{\lambda}$ is the surface current density vector and R is the distance between the field and the source point

$$R^2 = (x - x_o)^2 + (y - y_o)^2 + (z - z_o)^2$$

and

$$dA = a^2 \, d\phi_o \, dZ_o \, .$$

The z-component of Eq. 1 is

$$dB_z = \frac{\mu_o \, dA}{4\pi} \left\{ \lambda_x \frac{\partial}{\partial y_o} \left(\frac{1}{R} \right) - \lambda_y \frac{\partial}{\partial x_o} \left(\frac{1}{R} \right) \right\} \tag{2}$$

R can be expanded in Legendre polynomials (Ref. [2], p. 173)

$$\frac{1}{R} = \frac{1}{(r^2 + r_o^2 - 2 r r_o \cos\theta^*)^{1/2}} = \begin{cases} \dfrac{1}{r_o} \displaystyle\sum_{n=0}^{\infty} \left(\frac{r}{r_o} \right)^n P_n(\cos\theta^*), & r < r_o \\ \dfrac{1}{r} \displaystyle\sum_{n=0}^{\infty} \left(\frac{r_o}{r} \right)^n P_n(\cos\theta^*), & r > r_o \end{cases} \tag{3}$$

Using an expansion formula we have (Ref. [2])

$$P_n(\cos\theta^*) = \sum_{m=0}^{m=n} (2 - \delta_m^o) \, \frac{(n-m)!}{(n+m)!} \, P_n^m(\cos\theta) \, P_n^m(\cos\theta_o) \cos m(\phi - \phi_o) \tag{4}$$

Let

$$C_{nm} = P_n^m(\cos\theta)\cos m\phi$$

$$S_{nm} = P_n^m(\cos\theta)\sin m\phi$$

then it can be shown (Ref. [ ])

$$\frac{\partial}{\partial x} (r^{-n-1} C_{nm}) = (1 + \delta_m^o) \left\{ -\frac{1}{2} r^{-n-2} \, C_{n+1,m+1} + \frac{1}{2} (n-m+1)(n-m+2) r^{-n-2} \, C_{n+1,m-1} \right\}$$

$$\frac{\partial}{\partial y} (r^{-n-1} C_{nm}) = (1 + \delta_m^o) \left\{ -\frac{1}{2} r^{-n-2} \, S_{n+1,m+1} - \frac{1}{2} (n-m+2)(n-m+1) r^{-n-2} \, S_{n+1,m-1} \right\} \tag{5}$$

Using Eqs. 3, 4, and 5 we get the complete representation of dB_z from Eq. 2. The results are given in Ref. [1]

For the simpler case of cylindrical symmetry considered in the present paper, we represent the current distribution as $\lambda_\phi = c \, \sigma_\phi(Z_o)$ which gives

$$\lambda_x = -c\,\sigma_\phi(Z_o)\sin\phi_o$$

$$\lambda_y = c\,\sigma_\phi(Z_o)\cos\phi_o$$

and hence

$$B_z = \sum_{n=0}^{\infty} A_n r^n P_n(\cos\theta) \tag{5}$$

with

$$A_n = \frac{\mu_o c}{2a^n}\int_{-Z_m}^{Z_m} \sigma_\phi(Z_o) fn(Z_o)\,dZ_o \tag{6}$$

$$fn(Z_o) = \frac{P_{n+1}^1(\cos\theta_o)}{(1+Z_o^2)^{\frac{n+2}{2}}} \tag{7}$$

As can be seen from Eq. 5, A_n provides the constraints on the quality of the magnetic field for a given current distribution. In our case we take

$$A_n = \delta_p^o \quad p = 0,1,\cdots,N \tag{8}$$

Our objective is to find the current distribution $\sigma_\phi(Z_o)$ for a given order of homogeneity of magnetic field under the condition that the energy is minimized. As shown in Ref. [1], the energy W can be represented as

$$W = c^2\mu_o a^3\int_{-Z_m}^{Z_m}\int_{-Z_m}^{Z_m} \frac{\sigma_\phi(Z_o)\sigma_\phi(Z)}{\kappa}\left\{\left[1-\frac{\kappa^2}{2}\right]K(\kappa) - E(\kappa)\right\}dZ_o\,dZ \tag{9}$$

where

$\kappa^2 = \dfrac{4}{4+(Z-Z_o)^2}$ and $K(\kappa)$, $E(\kappa)$ complete elliptic integrals of first and second kind.

Using Lagrange multipliers, we set up a functional from Eqs. 8 and 9 as

$$I(\sigma_\phi) = W - \sum_{n=1}^{N}\lambda_n\int_{-Z_m}^{Z_m}\sigma_\phi(Z_o) f_n(Z_o)\,dZ_o \tag{9}$$

The above functional will be stationary around the exact solution of $\sigma_\phi(Z_o)$ provided the following integral equation is satisfied

$$\int_{-Z_m}^{Z_m}\sigma_\phi(Z_o)\,Q(Z_o-Z)\,dZ_o = \sum_{n=0}^{N}\lambda_n f_n(Z), \quad -Z_m < Z < Z_m$$

where the kernel

$$Q(Z_o-Z) = \frac{1}{\kappa}\left\{\left[1-\frac{\kappa^2}{2}\right]K(\kappa) - E(\kappa)\right\} \tag{10}$$

Hence, we need to solve a set of integral equations

$$\int_{-Z_m}^{Z_m} \sigma_\phi^i(Z_o) Q(Z_o - Z)\, dZ_o = f_i(Z), \quad -Z_m < Z < Z_m \tag{11}$$

and λ's are determined from the constraint conditions (Eq. 8)

$$\int_{-Z_m}^{Z_m} \sum_{i=0}^{N} \lambda_i \sigma_\phi^i(Z_o) f_p(Z_o)\, dZ_o = \delta_p^o, \quad p=1, \cdots, N \tag{12}$$

and the final solution is given by superposition:

$$\sigma_\phi(Z_o) - \sum_{i=1}^{N} \lambda_i \sigma_\phi^i(Z_o) \tag{13}$$

SHORT COIL APPROXIMATION

The kernel given by Eq. 10 has a logarithmic singularity around $Z=Z_o$. Expanding around this singularity

$$Q(Z) = \left(-1 + \frac{1}{2}\log 8\right) - \frac{1}{2}\log(|Z|) + O(Z^2 \log|Z|), \quad Z<1 \tag{14}$$

The main contribution to the solution comes from the singular part of the kernel, and the remaining expansion can be treated as a perturbation and transformed to the right-hand side of the integral equation, keeping as many terms as desired for improving the accuracy of the solution (Ref. [4]).

Hence, Eq. 11 becomes $\left(\text{with } \alpha = -1 + \frac{1}{2}\log 8,\ B = -\frac{1}{2}\right)$

$$\int_{-Z_m}^{Z_m} \sigma_\phi^i(Z_o) \left\{\alpha + \beta \log|Z - Z_o|\right\} dZ_o = f_i(Z), \quad -Z_m < Z < Z_m;\ i=0, \cdots, N \tag{15}$$

It can further be shown that

$$f_o(Z) = \frac{1}{(1+Z^2)^{3/2}}, \quad f_i = -\frac{1}{i}\frac{d}{dZ} f_{i-1} \tag{16}$$

and, in view of the expansion (Eq. 14), f_i, for the short coil approximation, can be expanded as a Taylor series of order T

$$f_o(Z) = 1 - \frac{3}{2}Z^2 + \frac{15}{8}Z^4 + \cdots + O(Z^T) \quad \text{etc.} \tag{17}$$

Let

$$\begin{aligned} Z &= Z_m \cos\omega \\ Z_o &= Z_m \cos\theta \end{aligned} \tag{18}$$

After trigonometric reduction, Eq. 17 can be represented as a Fourier series

$$f_o(Z) = \frac{1}{2} B_o^o + \sum_{m=1}^{T} B_m^o \cos m\omega \tag{19}$$

and Eq. 15 becomes (for $i = 0$)

$$\int_0^{\pi} \left\{ (\alpha + \log Z_m) + \beta \log|\cos\omega - \cos\theta| \right\} \left(-\sigma_\phi^o(Z_o) \frac{dZ_o}{d\theta} \right) d\theta = \frac{1}{2} B_o^o + \sum_{m=1}^{\infty} B_m^c \cos m\omega, \quad 0 < \theta < \pi \tag{20}$$

In view of identity, for the dominant kernel (Ref. [4])

$$\log|\cos\omega - \cos\theta| = -2 \sum_{n=1}^{\infty} \frac{\cos n\omega \cos n\theta}{n} - \log 2 \tag{21}$$

we expand the unknown solution as

$$-\sigma_\phi^o(Z_o) \frac{dZ_o}{d\theta} = \frac{1}{2} A_o^o + \sum_{m=1}^{\infty} A_m^o \cos m\theta \tag{22}$$

Substituting from Eqs. 22, 21, and 19 into Eq. 20 and using orthogonality conditions, we can solve for A_m^o

$$A_o^o = \frac{B_o^o}{\alpha + \log Z_m - \beta \log 2}$$

$$A_m^o = -\frac{mB_m^o}{\beta\pi}, \quad m = 1, 2, \cdots, T \tag{23}$$

Carrying out a similar procedure for each constraint and assembling the solution we have

$$\sigma_\phi(Z_o) = \sum_{n=1}^{N} \lambda_n \sigma_\phi^n(Z_o) \tag{24}$$

To obtain λ's we use the above in the constraint equations (Eq. 12); transforming variables by Eq. 18 and using the orthogonality condition, we have the following linear equations for λ_n

$$\frac{\pi}{2} \sum_{n=1}^{N} \lambda_n \left(\frac{1}{2} A_o^p B_o^p + \sum_{m=1}^{T} A_m^n B_m^p \right) = \delta_p^o \tag{25}$$

The analysis given above involves tedious algebra. This was accomplished using the symbolic manipulation program MACSYMA (Ref. [6]). The complete program is given in Appendix A1. Using this program, we plot the result in Figure 2 together with numerical results obtained by the method given in Ref. [1]. As can be seen, the results agree well except possibly at singularities near the end points. In the present case, the singularity is given by (see Eq. 22)

$$-\frac{1}{\frac{dZ_o}{d\theta}} = \frac{1}{(Z_m^2 - Z_o^2)^{1/2}} \tag{26}$$

The coefficient of the singularity can be easily computed. This phenomenon makes a short coil less desirable. In the next section we carry out the approximation for a large coil.

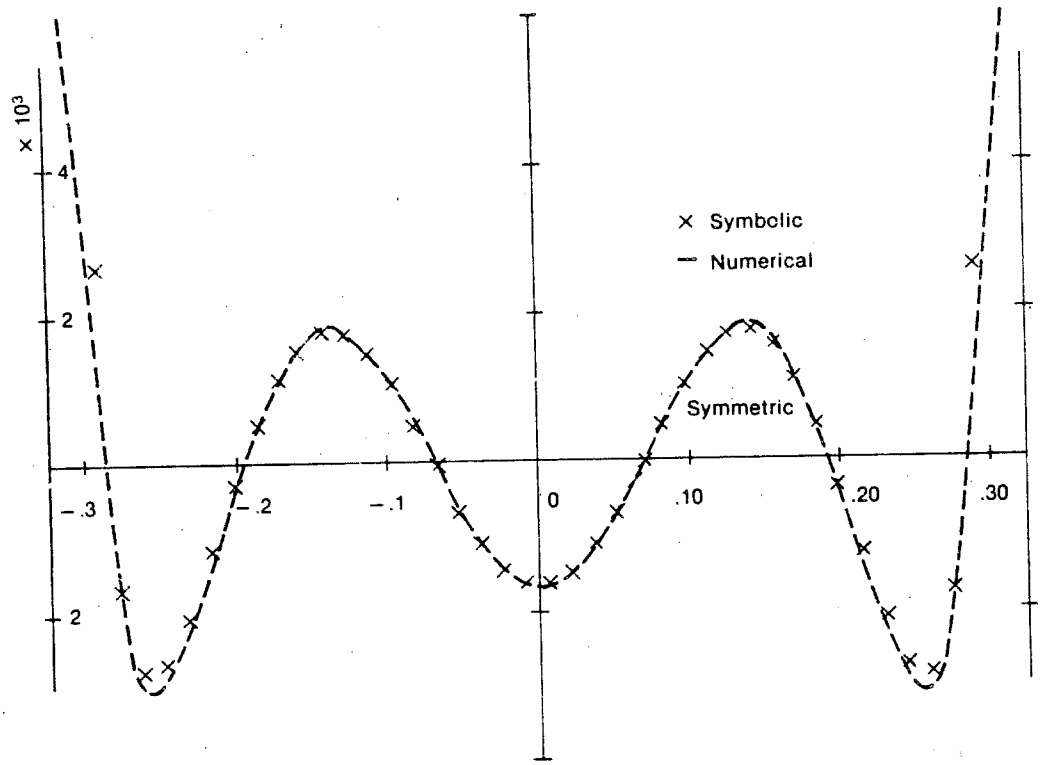


Figure 2. Short coil approximation for $\mathcal{L}_m = .3$, comparison of symbolic solution to numerical solution.

LARGE COIL APPROXIMATION

Again, for simplicity, consider the symmetric case [i.e., $\sigma_\phi(-Z_o) = \sigma_\phi(Z_o)$]; then the dominant integral of Eq. 15 becomes

$$\int_0^{Z_m} \sigma_\phi^i \left\{ 2\alpha + \beta \log|Z^2 - Z_o^2| \right\} dZ_o = f_i(Z), \quad 0 < Z < Z_m \tag{27}$$

For the large coil approximation we let

$$Z = \tan\theta, \quad Z_o = \tan\theta_o, \quad Z_m = \tan h$$

However, the logarithmic kernel has to be approximated further as

$$\log|\tan^2\theta - \tan^2\theta_o| \simeq \log 2 + \log|\cos\theta - \cos\theta_o|, \quad \text{for } \theta \neq \frac{\pi}{2} \tag{28}$$

In the present case

$$f_o(Z) = \frac{1}{(1+Z^2)^{3/2}} = \cos^3\theta = \frac{1}{4}(\cos 3\theta + 3\cos\theta) \tag{29}$$

Similarly, all $f_i(Z)$ will have the Fourier expansion and, hence, we need not take the Taylor expansion as was done for the short coil approximation. The integral equation then becomes

$$\int_0^h \left[\frac{\sigma_\phi^i}{\cos^2\theta_o}\right]\left\{(2\alpha+\beta\log 2)+\beta\log|\cos\theta-\cos\theta_o|\right\}d\theta_o = f_i(Z), \quad 0<\theta<h \tag{30}$$

In the above, the limits of integration do not allow us to use the orthogonality property, and we make a further transformation due to Schwinger (Ref. [5])

$$\begin{aligned}\cos\theta_o &= r + s\cos\xi, \quad r = \cos^2\left[\frac{h}{2}\right] \\ \cos\theta &= r + s\cos x, \quad s = \sin^2\left[\frac{h}{2}\right]\end{aligned} \tag{31}$$

Using Eq. 31, the integral equation now becomes

$$\int_0^\pi \left[\frac{\sigma_\phi^i}{\cos^2\theta_o}\frac{d\theta_o}{d\xi}\right]\left\{(2\alpha+\beta\log 2s+\beta\log|\cos\xi-\cos x|\right\}d\xi = f_i(x), \quad 0<x<\pi \tag{32}$$

Equation 32 has the same form as before (see Eq. 20); however, the algebra is quite complex and again we use MACSYMA (complete program is given in Appendix A2) to carry out the calculations shown in Figure 3. The agreement with the numerical analysis is quite good. It can be seen that singularity is not prominant. The reason for this phenomenon is that the coefficient of the singularity asymptotically goes to zero as $1/Z_m$. This can be seen by studying

$$\cos^2\theta_o / \frac{d\theta_o}{d\xi} \tag{33}$$

i.e.

$$\frac{\left[1+(1+Z_o^2)^{1/2}\right]^{1/2}(1+Z_m^2)^{1/4}}{(1+Z_o^2)\left[(1+Z_m^2)^{1/2}-(1+Z_o^2)^{1/2}\right]^{1/2}}$$

CONCLUSION

In this paper we have given a simple algorithm for the solution of a singular integral equation. The results can be easily used for discretizing coils which may lead to a practical design requiring close to minimum energy.

The algorithm given is straightforward, but involves a number of tedious calculations. MACSYMA has been used extensively to carry out manipulations. The method can be easily adapted to multi-coil design, the mixed boundary value problem in potential theory, the theory of elasticity, and also for dual-series and dual-integral equations.

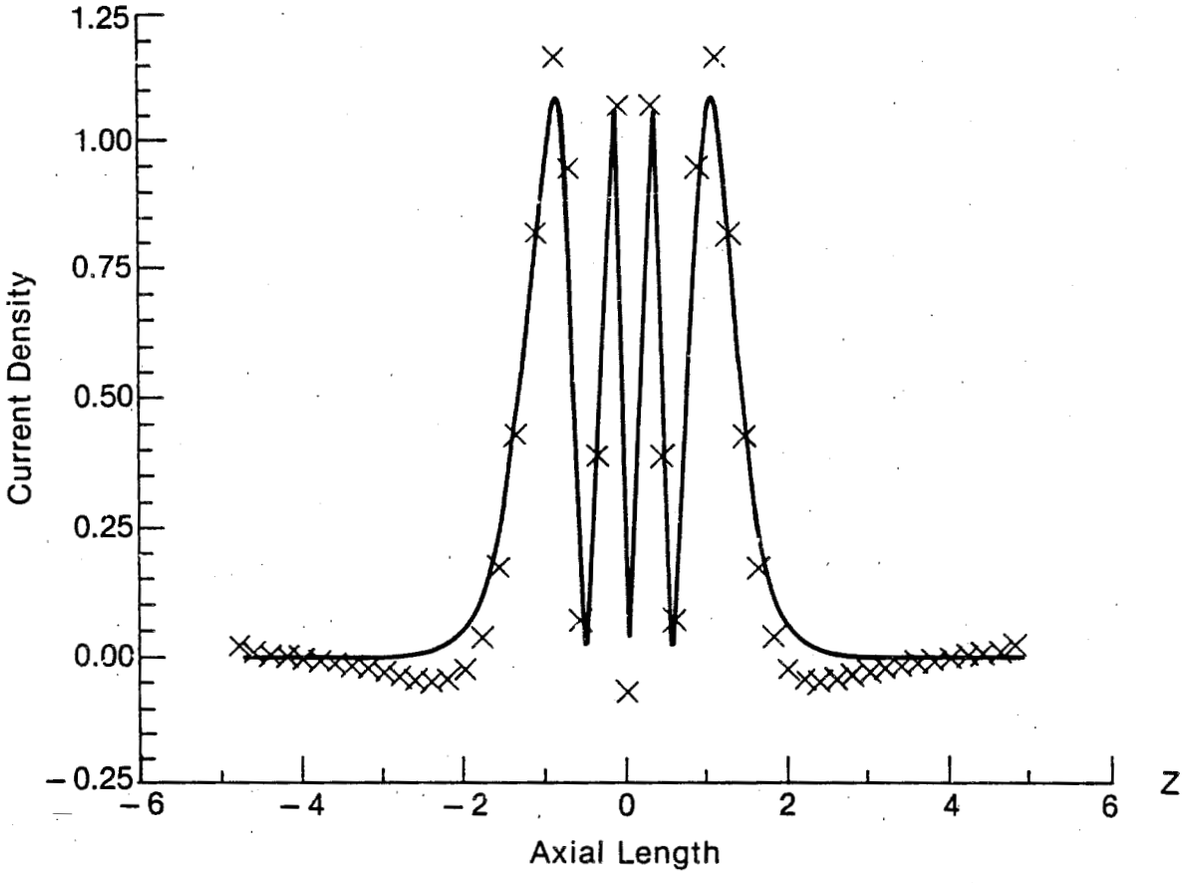


Figure 3. Large coil approximation for $Z_m = 4.9$, comparison of symbolic solution to numerical solution.

REFERENCES

1. J.F. Schenck, M.A. Hussain, W.A. Edelstein and B. Noble, "An Integral Equation for the Design of Magnetic Field Coils," General Electric Technical Report 82CRD150, May 1982; also *Proceedings of the 1982 Numerical Analysis and Computer Conference,"* ARO Report No. 82-3.
2. W.R. Smythe, *Static and Dynamic Electricity, Third Edition,* McGraw-Hill Book Company, New York, NY, 1953.
3. P.M. Morse and H. Feshbach, *Methods of Theoretical Physics,* McGraw-Hill Book Company, New York, NY, 1953.
4. B. Noble and M. Hussain, "Angle of Contact for Smooth Elastic Inclusions," *Developments in Mechanics: Proceedings of the Tenth Midwestern Mechanics Conference,* edited by J.E. Cerrmak and J.R. Goodman, pp. 459-473.
5. L. Levin, *Advanced Theory of Waveguides,* Iliffe and Sons, London 1951.
6. *MACSYMA: The Reference Manual, Version 10, 1983,* Math Lab Group, Laboratory for Computer Science, MIT.

APPENDIX A1: SHORT COIL APPROXIMATION

```
TITLE(EXP):=BLOCK(DISP(DPART(EXP)))$
TITLE("INTEGRAL EQUATION FOR SHORT COIL");
NRS:3;
OTE:4;
NTS:OTE/2+1;
TITLE("NRS=NO. OF R.H.S");
TITLE("OTE=ORDER OF TAYLOR EXPANSION");
TITLE("NTS=NO. OF TERMS IN SERIES");
TITLE("ZM=LENGTH OF THE COIL E.G.= .3");
ALPHA:-1+1/2*LOG(8)-1/2*LOG(ZM)$
BETA:-1/2$
G[0]:1/(1+Z0**2)**(3/2);
G[M]:=RATSIMP(-1/M*DIFF(G[M-1],Z0));
X:COS(TH)$
FOR I THRU NRS DO LDISPLAY ( F[I]:FACTOR(G[2*(I-1)]))$
FOR I THRU NRS DO TYLR[I]:TAYLOR(F[I],Z0,0,OTE)$
FOR I THRU NRS DO LDISPLAY ( RHS[I]:1/ZM*SUM(PART(TYLR[I],M),M,1,NTS))$
FOR I THRU NRS DO LDISPLAY(RHS[I]:TRIGREDUCE(SUBST(X*ZM,Z0,RHS[I])))$
FOR I THRU NRS DO B0(I):=BLOCK([F1,F2,F3],F1:INTEGRATE(RHS[I],TH),
F2:EV(F1,TH = %PI),F3:EV(F1,TH = 0),2/%PI*RATSIMP(F2-F3))$
FOR I THRU NRS DO LDISPLAY(B0(I))$
FOR I THRU NRS DO (FOR J THRU OTE DO LDISPLAY(B[I,J]:RATCOEF(RHS[I],COS(TH*J))))$
FOR I THRU NRS DO (FOR J FROM 0 THRU 0 DO B[I,J]:B0(I))$
FOR I THRU NRS DO (FOR J FROM 0 THRU OTE DO (IF J = 0 THEN
 A[I,J]:B[I,J]/(%PI*(ALPHA-BETA*LOG(2))) ELSE A[I,J]:-J*B[I,J]/(BETA*%PI)))$
FOR I THRU NRS DO (FOR J THRU NRS DO BB[I,J]:%PI/4*A[J,0]*B[I,0]
+SUM(%PI/2*B[I,N]*A[J,N],N,1,OTE))$
FOR I THRU NRS DO (FOR J THRU NRS DO LDISPLAY(BB[I,J]))$
FOR I THRU NRS DO EQ[I]:SUM(BB[I,J]*LAM[J],J,1,NRS)$
EQ[1]:EQ[1]-1/ZM**2$
EQQ:[];
FOR I:1 THRU NRS DO ( EQQ:CONS(EQ[I],EQQ));
LAMM:[];
FOR I:1 THRU NRS DO (LAMM:CONS(LAM[I],LAMM));
LINSOLVE(EQQ,LAMM),GLOBALSOLVE:TRUE;
FOR I:1 THRU NRS DO DISPLAY (VALU[I]=EV(LAM[I],NUMER))$
FOR I THRU NRS DO RSOL[I]:1/2*A[I,0]+SUM(A[I,N]*COS(N*W),N,1,OTE)$
FOR I THRU NRS DO RSOL[I]:TRIGEXPAND(RSOL[I])$
FOR I THRU NRS DO LDISPLAY(RSOL[I])$
FOR I THRU NRS DO LDISPLAY(RSOL[I]:RATSUBST(1-COS(W)^2,SIN(W)^2,RSOL[I]))$
FOR I THRU NRS DO LDISPLAY(RSOL[I]:RATSUBST(Y,COS(W),RSOL[I]))$
REGULARSOLN:SUM(RSOL[I]*LAM[I],I,1,NRS)$
RATSIMP(%)$
REGULARSOLN:%$
REGULARSOLN:RATSUBST(Z0/ZM,Y,%);
ZM:.3;
F1:EV(REGULARSOLN);
TERM:1/SQRT(1-(Z0/ZM)**2);
F2:EV(TERM*F1/ZM);
PLOTNUM:50;
EQUALSCALE:FALSE;
PLOT(F2,Z0,-.29,.29,"SYMBOLIC SOLUTION WITH THREE CONSTRAINTS AND ZM=.3");
```

APPENDIX A2: LARGE COIL APPROXIMATION

```
TITLE(EXP):=BLOCK(DISP(DPART(EXP)))$
TITLE("FIRST SELECT NUMBER OF CONSTRAINTS");
NLAM:4;
SHOWTIME:TRUE;
S:.4;
R:1-S;
CC0:1/2*LOG(8)-1,NUMER;
CC1:-1/2,NUMER;
KEEPFLOAT:TRUE;
ALPHA:2*CC0+CC1*LOG(S),NUMER;
BTA:-2*CC1;
/*.....GENERATION AND SIMPLIFICATION...*/
TITLE("GENERATION AND SIMPLIFICATION OF RIGHT HAND SIDE");
G[0]:1/(1+Z**2)**(3/2);
G[M]:=(-1/M*DIFF(G[M-1],Z));
FOR I:0 THRU NLAM-1 DO LDISPLAY ( RHS[I]:FACTOR((1+Z**2)**((4*I+3)/2)*G[2*I]));
FOR I:0 THRU NLAM-1 DO LDISPLAY (RHS[I]:(RATSUBST((1-COS(TH)**2)/COS(TH)**2
,Z**2,RHS[I])));
FOR I:0 THRU NLAM-1 DO LDISPLAY ( RHS[I]:RHS[I]*COS(TH)**(4*I+3));
FOR I:0 THRU NLAM-1 DO LDISPLAY ( RHS[I]:RATSUBST((R+S*COS(X)),COS(TH),RHS[I]));
FOR I:0 THRU NLAM-1 DO LDISPLAY ( RHS[I]:EXPONENTIALIZE(RHS[I]));
FOR I:0 THRU NLAM-1 DO DISPLAY ( RHS[I]:EXPAND(RHS[I]));
FOR I:0 THRU NLAM-1 DO LDISPLAY ( RHS[I]:DEMOIVRE(RHS[I]));
EP(N):=IF N=0 THEN 2 ELSE 1;
TITLE("NOW COLLECT COEFFECIENT BY SCHWINGER METHOD AND ASSEMBLE SOLUTION");
AB[I,J]:=EP(J)*RATCOEFF(RHS[I],COS(X*J));
FOR I:0 THRU NLAM-1 DO (FOR J:0 THRU 4*I+3 DO (AB[I,J]));
B[I,J]:=BLOCK( IF J=0 THEN B[I,J]:RATSIMP(2*(RHS[I]-SUM(COS(X*T)*AB[I,T],T,1,4*I+3)))
ELSE AB[I,J]);
FOR I:0 THRU NLAM-1 DO (FOR J:0 THRU 4*I+3 DO (B[I,J]));
FOR I:0 THRU NLAM-1 DO (FOR J:0 THRU 4*I+3 DO (
IF J=0 THEN A[I,J]:B[I,J]/(%PI*(ALPHA))
ELSE A[I,J]:2*J*B[I,J]/(BTA*%PI) ));
FOR I:0 THRU NLAM-1 DO (FOR J:0 THRU 4*I+3 DO (A[I,J]));
BB[I,J]:=%PI*(1/2*A[J,0]*B[I,0]
+SUM(A[J,N]*B[I,N],N,1,MIN(4*I+3,4*J+3)));
TITLE("NOW SET UP EQUATIONS FOR LAMBDA FROM CONSTRAINTS");
FOR I:0 THRU NLAM-1 DO (EQ[I]:SUM(BB[I,J]*LAM[J],J,0,NLAM-1));
EQ[0]:EQ[0]-1;
LIST1:[];
LIST2:[];
FOR L:0 THRU NLAM-1 DO ( LIST1:CONS(EQ[L],LIST1));
FOR L:0 THRU NLAM-1 DO (LIST2:CONS(LAM[L],LIST2));
TITLE ("NOW SOLVE AND ASSEMBLE FINAL SOLUTION FOR PLOT , AVOIDING SINGULARITY");
LINSOLVE(LIST1,LIST2),GLOBALSOLVE:TRUE;
FOR L:0 THRU NLAM-1 DO
(SOL1[L]:1/2*A[L,0]+SUM(A[L,N]*COS(N*W),N,1,4*L+3));
FOR L:0 THRU NLAM-1 DO (SOL1[L]:TRIGEXPAND(SOL1[L]));
FOR L:0 THRU NLAM-1 DO
(SOL1[L]:RATSUBST(1-COS(W)**2,SIN(W)**2,SOL1[L]));
FOR L:0 THRU NLAM-1 DO (SOL1[L]:RATSUBST(Y,COS(W),SOL1[L]));
FINALSOLUTION:RATSIMP(SUM(SOL1[L]*LAM[L],L,0,NLAM-1));
EV(FINALSOLUTION,NUMER);
EV(%,NUMER);
EXPAND(%);
EV(%,NUMER);
FINAL:%;
```

```
ZM:TAN(2*ASIN(SQRT(S)));
SQZ:SQRT(1+Z**2);
SQM:SQRT(1+ZM**2);
Y:1/S*(1/SQZ-R);
FACT:(1+SQZ)**(1/2)/(SQZ**(5/2)*(1/SQZ-1/SQM)**(1/2));
FINAL:EV(FACT*FINAL);
EQUALSCALE:FALSE;
PLOTNUM:100;
ZMM:ZM-ZM/PLOTNUM,NUMER;
PLOT(FINAL,Z,-ZMM,ZMM,"SYMBOLIC SOLUTION WITH FOUR CONSTRAINTS");
```

AN AUTOMATIC TESTING FACILITY FOR VAXIMA

Carl R. Powell
Dept. of Mathematical Sciences
Kent State University
Kent, Ohio 44242

IBM Corporation
1300 East Ninth Street
Cleveland, Ohio 44114

Abstract

An automatic testing facility for newly-created versions of VAXIMA, called VaxTest, is described. The facility, written primarily in FRANZ LISP on a DEC Vax 11/780 under the UNIX operating system, is not restricted to any particular machine or operating system. The VaxTest facility provided the tools for the interactive testing of the various prepared test files or functional subgroups by the individual user and the system manager, or automatic testing by the operating system. Each test file is executed in the new VAXIMA environment. Each computed result is compared to the "correct" answer stored in the corresponding standard file. Definitive, descriptive messages are printed to aid in the location and correction of comparison mismatches and execution errors.

1. INTRODUCTION

This paper describes the design and implementation of an automatic testing facility, termed VaxTest, used in the testing of newly-created versions of VAXIMA. VAXIMA is a dialect of MACSYMA designed at the University of California at Berkeley for use on VAX/UNIX systems. MACSYMA (Project MAC's Symbolic Manipulation System) is a large computer programming system, written in MACLISP (a dialect of the LISP programming language), used for performing symbolic and numerical mathematical manipulations. MACSYMA has been developed by the Mathlab Group at the Massachusetts Institute of Technology laboratory for Computer Science (formerly Project MAC). The testing facility described herein is written primarily in FRANZ LISP (another dialect of the LISP programming language), version Opus 38, on a DEC Vax 11/780 operating under Berkeley 4.1 UNIX operating system. Testing and execution of the VaxTest facility were done in version 2.04 of VAXIMA (a dialect of version 10 MACSYMA). The VaxTest facility is designed for use with any

MACSYMA facility, and is not restricted to any particular machine or operating system.

VaxTest is a structural testing package for the dynamic analysis of the VAXIMA facility following maintenance-phase modifications. The three phases of the facility (checking the status of standard files, creating standard files, and comparing test and standard files) can be invoked interactively at the user and system manager levels or automatically at the operating system level. Definitive, descriptive messages are printed to aid in the location and correction of comparison mismatches and execution errors.

VaxTest is designed to be used in the construction of modifications to the VAXIMA system as well as in the regression testing of these modifications once they have been installed. These modifications usually involve the upgrading or expanding of the capabilities of the VAXIMA system, by either adding new functions or enhancing the current ones. In retesting the VAXIMA system, the capabilities of the modified version are dynamically analyzed, using user-defined and VaxTest-defined test files as test cases. These test files are based on the various demo files used currently to test the VAXIMA system. These prepared demo files contain user-level VAXIMA commands designed to test a desired class of functions. Ideally, the contents of these demo files would be derived from a path analysis of the VAXIMA source code. This would provide for a very thorough structural test of the VAXIMA system. Currently, this is not the case. The testing diagnostics generated by the VaxTest facility are produced by automated output comparators used to compare the test case results, from the modified VAXIMA environment, against a set of standards. These diagnostics aid in the location and correction of any detected errors, ensuring that the modified VAXIMA system will perform at least to the same level as its previous version.

Work reported herein has been supported in part by the National Science Foundation under grant MCS 82-01239 and by the Department of Energy under grant DE-AC02-ER7602075-A010.

2. DESIGN OF NEW TESTING FACILITY (VAXTEST)

2.1 Command Summary

The new testing facility, VaxTest, consists of three VAXIMA-executable commands : *creatstd*, *checkstd*, and *runtest*. These commands are invoked from a VAXIMA environment containing the VaxTest source code (using the VAXIMA *loadfile* command). A brief summary of each of the commands is given below, with details on their use and implementation given later in this document.

The *creatstd* command is used to explicitly create the desired standard files. A standard file contains the correct representation of the evaluated VAXIMA commands stored in the corresponding test file. If given a legal test file name, the command creates the corresponding standard file. If given a legal test tag, the command creates the standard files corresponding to each test file in the tag. Each standard file contains the internal LISP representation of each VAXIMA command contained in the corresponding test file. Diagnostics are produced when illegal test file or tag names are used, or when file access is denied (read for test file, write for

standard file).

The *checkstd* command is used to check the status of the desired standard files and create those that are out of date. If given a legal test file name, the command checks to see if the corresponding standard file is up to date. If given a legal test tag, the command checks to see it each standard file corresponding to each test file in the tag is up to date. When a standard file needs to be created or updated, the *checkstd* command incorporates those routines used by the *creatstd* command. Diagnostics are produced when illegal test file or tag names are used, or when file access is denied (read for test file, write for standard file), or to report standard file status (up to date or requiring creation).

The *runtest* command is used to compare the desired test files against their corresponding standard files. If given a legal test file name, the command compares the internal LISP representation of each evaluated VAXIMA command in the test file to the "correct" representation stored in the corresponding standard file. If given a legal test tag, the command sequentially tests all test files in the tag against their corresponding standard files. An optional results file may also be specified. Diagnostics are produced when illegal test file or tag names are used, file access is denied (read for test file, read for standard file, write for optional results file), or to report comparison results (evaluation errors, code inequality, or file equality).

2.2 Advantages

VaxTest is designed as a testing facility to be used in the debugging of new additions to the VAXIMA system as well as in the regression testing of modified versions of VAXIMA. It avoids all of the drawbacks of its predecessor, and includes several advantages of its own.

First, the VaxTest facility is designed to be system and machine independent. All but one of its functions are designed in FRANZ LISP and can be used on any system supporting this LISP dialect. The only function in VaxTest which is dependent on the target system is the function *compstd*. It is used to check the modification status of test and standard files (used by the *checkstd* command), and should be relatively easy to design on any system. FRANZ LISP provides a facility (a version of the *fasl* command) for the incorporation of foreign subroutines. Since VaxTest was written on a machine supporting the UNIX operating system, the function *compstd* was written in the C programming language using the UNIX *stat* command.

Secondly, the three VAXIMA-executable commands allow for the interactive testing of any combination of test files or tags. These file or tag names can be user-defined or VaxTest-defined, allowing each user to personalize their test file data base. This allows easy debugging of proposed additions to the VAXIMA facility and assistance in the construction of new test files for the system. Also, by allowing each user to specify the exact test files that are to be tested (thus excluding all others), it also reduces the time and memory overhead needed to execute a test run.

Since any combination of test files or tags is possible with all three commands, there is no need to separate the coding for the separate tests. All test runs use the same functions, with only the test file data base changing from run to run. This reduces the amount of coding needed to design the facility, reduces the memory size needed to store the testing facility, and increases the ease of modifying the facility. Also, since all test runs use the same coding

segments, greater conformity among the results is achieved, allowing easier analysis of the output.

Next, since file comparison is done on the internal LISP representations of the evaluated results of the VAXIMA commands, no errors will be produced for mathematically equivalent answers. Whereas the old testing facility would flag the following examples as being nonequivalent, the new VaxTest facility recognizes them as being mathematically equal :

spacing : "x + 2" and " x + 2"
ordering : "x + 2" and "2 + x"
expansion : "$(x + 2)^2$ " and "$x^2 + 4x + 4$"

Finally, the VaxTest facility is designed to generate definitive and descriptive diagnostics to identify the test files and their line numbers where errors occur. This greatly increases the ease and rapidity with which problem areas can be isolated and corrected. These comparison diagnostics are printed when an evaluation error occurs (*evalerr*), when nonequivalent lines of code are found (*mismatch*), or when all lines of code are found to be equivalent (*allmatch*). The printing of these comparison diagnostics does not cause a premature termination of the test run. All lines remaining to be compared in the test file, as well as the remaining untested test files, will be processed. In addition, error diagnostics are also printed when illegal test file or tag names are used, or when file access is denied.

In all, the VaxTest facility allows for a greater flexibility in the number and types of tests that can be executed, and increases the range of applications in which it can be used. The facility is not tied down to any particular machine or system, and can be used jointly in the regression testing of VAXIMA modifications as well as the design of new VAXIMA functions and their corresponding test files.

2.3 Constraints

In designing the VaxTest facility, some restrictions had to be placed on the testing facility as well as its test files. First, since the VaxTest facility depends on the VAXIMA commands *alike*, *ratsimp*, and *meval*, it is assumed that these functions are working properly when the VaxTest commands are used. Unfortunately, this assumption may be incorrect if the modifications to the VAXIMA system have altered the operation of these commands. This places the VaxTest facility in a "catch-22" situation : it requires the flawless operation of the very item it is trying to find errors in. Fortunately, an error in any of these VAXIMA commands would cause erratic behavior in the operation of the VaxTest facility, indicating improper functioning of the testing facility.

Also, several restrictions have been placed on the design of the test files. First, although any of the VAXIMA-defined constants (*%E*, *%I*, *%PI*, and *%RHO*) can be used in the test file commands, the *%* variable (previous computation) is the only VAXIMA-defined variable allowed. The *%%* (last MACSYMA-BREAK computation) and the *%TH(i)* (i-th previous computation) variables were not implemented and will produce erroneous results if used. Second, even though any VAXIMA function can be used in the test file commands, the display functions (*display*, *dispfun*, and *letrules*) should be avoided. These functions produce results that are not

stored in the standard files and are, therefore, excess baggage in the test files.

Finally, each test file command is managed differently depending on its terminating character. In an attempt to conform to the VAXIMA facility, test file commands terminating with a semicolon (;) are evaluated and manipulated by the VaxTest facility. These commands will either be stored in the appropriate standard files (*createtd* and *checkstd* commands) or used in the comparison of the test and standard files (*runtest* command). Demo file commands terminating with a dollar sign ($) are evaluated but are not manipulated by the VaxTest facility. They are neither stored in the standard files nor used in the comparison of test and standard files. The % variable will always be set to the previous command line prior to evaluation, regardless of its terminating character.

3. IMPLEMENTATION OF VAXTEST

3.1 Diagnostics

Many diagnostic messages are printed at various stages in the VaxTest facility to assist in the location and correction of problem areas as well as to monitor the status of the various commands. There are three types of messages produced : those to report user or system errors (error messages), those to report the status of a function or command (information messages), and those to report comparison results (comparison messages). This section briefly describes each message, when it is printed, what is printed, and its effect on the execution of the current command.

If an illegal test file or tag name is supplied to any of the three VAXIMA-level VaxTest commands (*createtd*, *checkstd*, or *runtest*), the following error message will be printed to the standard error port :

Illegal file or tag : <*test file or tag name*>

The printing of this message suppresses execution of the rest of the command and causes an immediate return to the top-level VAXIMA environment.

If file access is denied to any of the test or standard files referenced by any of the VaxTest commands, one of the following error messages will be printed to the standard error port :

Cannot read from test file : <*test file name*>
Cannot write to standard file : <*standard file name*>

File access can be denied if the desired access permission (read or write) has been removed, or if the file does not exist. This error message is produced by the functions *makestd* (*createtd*) or *out-of-date* (*checkstd*) when read access for test files or write access for standard files is denied, or by the function *demo-vs-std* (*runtest*) when read access for test and standard files is denied. The printing of this message causes an immediate exit from the current VaxTest command if its argument is a test file name. If the argument is a test tag name, the command will exit for the

current test file name, but the remaining test files will be processed normally.

When the optional output file for the command *runtest* cannot be accessed (denial of write permission), the following error message is printed to the standard error port :

Cannot write to results file : <*output file name*>

The printing of this message suppresses execution of the rest of the command and causes an immediate return to the top-level VAXIMA environment.

When standard files are being created or having their status checked, one of the following information messages will be printed to the standard output :

Creating standard file : <*standard file name*>
Standard file up to date : <*standard file name*>

The first message is produced by the function *makestd* (*creatstd*), while both messages are produced by the function *out-of-date* (*checkstd*). The printing of either message does not cause an interruption in the execution of the desired command.

When the command *runtest* is invoked, the following information message will be printed to the desired output port (either the standard output or the optional results file) :

Run test for file or tag : <*test file or tag name*>

The printing of this message does not cause an interruption in the execution of the *runtest* command.

If an error is caused during the evaluation of a line of the internal LISP representation code (either from the test or standard file), the error will be trapped (using the LISP function *errset*) and the following error message will be printed to the desired output port :

ERROR : found in <*test file name*> at line # <*line no.*>

This error message is produced by the function *demo-vs-std* (*runtest*) and will cause a 'fail' condition (nonequivalent code) for the comparison of the test and standard files. The comparison function will resume with the following lines in each file.

If two lines of internal LISP representation code (one from the test file, the other from the standard file) are found to be nonequivalent, the following comparison message will be printed to the desired output port :

Mismatch : found in <*test file name*> at line # <*line no.*>

This comparison message is produced by the function *demo-vs-std* (*runtest*) and will cause a 'fail' condition for the comparison of the test and standard files. The comparison function will resume with the following lines in each file.

If all lines of both files (test and standard) are found to be equivalent, the following comparison message will be printed to the desired output port :

Test and standard files match for <*test file name*>

This comparison message is produced by the function *demo-vs-std* (*runtest*) and indicates a 'success' condition for the comparison of the test and standard files.

3.2 Command Usage

As stated earlier, the VaxTest facility consists of three user-level VAXIMA commands. This section contains a description of how these commands are used, when they are used, and examples of their use (using a DEC Vax 11/780 operating under the UNIX operating system).

To use the VaxTest facility, the file containing its source code must be loaded into a VAXIMA environment.

```
% vaxima
    :
(c1) loadfile ("vaxtest.l")$
    :
(c2)
```

User-defined test file or tag names should also be loaded in at this time, although user-defined modifications to the VAXIMA package should not be loaded until after the *creatstd* and *checkstd* commands have been executed. This is to ensure that any new or updated standard files are created in the current "correct" VAXIMA environment and not in an environment containing the proposed VAXIMA modifications (since this might result in the storage of erroneous LISP code in the standard files).

```
(c2) loadfile ("myfiles.l")$
    :
(c3)
```

The *creatstd* command, used to explicitly create the desired standard files, is a preparatory function to the actual file testing and should always be used after a new test file or tag has been created. It can also be used any time the checking overhead in the *checkstd* command is not needed. The *creatstd* command must be given one argument : a test file or tag name defined by the user or the VaxTest facility. Any diagnostic messages will be sent to the standard error port (usually the terminal screen).

```
(c3) creatstd ("my.demo");
Creating standard file : 'my.std'

(d3)          false

(c4) creatstd (mytag);
Creating standard file : 'my.std'
```

```
    :
Creating standard file : 'your.std'

(d4) (my.demo, ... , your.demo)

(c5)
```

The *checkstd* command, used to check the status of the desired standard files, is a preparatory function to the actual file testing and should always be used prior to the use of the *runtest* command or after a test file has been modified. The *checkstd* command ensures that all standard files to be tested are current, thus preventing any inappropriate comparison diagnostics if out of date standard files were used. The *checkstd* command must be given one argument : a test file or tag name defined by the user or the VaxTest facility. Any diagnostic messages will be sent to the standard error port.

```
(c5) checkstd ("my.demo");
Creating standard file : 'my.std'

(d5)          false

(c6) checkstd (mytag);
Standard file up to date : 'my.std'
    :
Creating standard file : 'your.std'

(d6) (my.demo, ... , your.demo)

(c7)
```

After the *creatstd* and *checkstd* commands have been used, the user-defined modifications to the VAXIMA facility should be loaded in. This ensures that the *runtest* command will evaluate the test file commands in the new version of VAXIMA while the standard files will contain the correct internal LISP representations.

```
(c7) loadfile ("new.vaxima")$
    :
(c8)
```

The *runtest* command, used to compare the desired test files against their corresponding standard files, is the heart of the VaxTest facility. The *runtest* command may be given one or two arguments. The first argument, which is mandatory, must be a test file or tag name defined by the user or the VaxTest facility. The second argument, which is optional, must be a file name to which any comparison messages will be appended to. If a second argument is not given, the standard output port is used. All error messages are sent to the standard error port.

```
(c8) runtest ("my.demo");
```

```
Run test for file or tag : 'my.demo'
Test and standard files match for 'my.demo'

(d8)          false

(c9) runtest (mytag);
Run test for file or tag : 'mytag'
Test and standard files match for 'my.demo'
    :
Mismatch : found in 'your.demo' at line #10

(d9)          false

(c10) runtest ("my.demo", "my.results");

(d10)         false

(c11) runtest (mytag, "my.results");

(d11)         false

(c12) exit();

% cat my.results
Run test for file or tag : 'my.demo'
Test and standard files match for 'my.demo'

Run test for file or tag : 'mytag'
Test and standard files match for 'my.demo'
    :
Mismatch : found in 'your.demo' at line #10
```

3.3 Levels of Implementation

There are three levels at which the VaxTest facility can be implemented : the user level, the manager level, and the system level. This section describes how the VaxTest facility is used at each level, along with examples of its use.

At the user level, the VaxTest facility provides an interactive tool to check the effect of user-defined modifications on the current VAXIMA environment, to aid in the debugging of these modifications, and to aid in the designing of the appropriate test files. The *creatstd* command is used on user-defined test files to create the appropriate user-defined standard files, with the command *runtest* used on all test files (user and VaxTest-defined) to aid in the debugging of the proposed VAXIMA modifications.

Once in a VAXIMA environment, the VaxTest source code should be loaded in. The user can then modify the VaxTest-defined test file and tag names by loading in their own test versions from a file containing the user-defined tag names and their associated test file names.

```
% vaxima
    :
(c1) loadfile ("vaxtest.l")$
    :
(c2) loadfile ("my.files")$
    :
(c3)
```

All user-defined test files must be appended to the VaxTest 'all' test tag, and all user-defined test tags must be appended to the VaxTest 'tags' list.

After modifying or creating the desired user-defined standards, the user-defined modifications to the VAXIMA environment should be loaded in.

```
(c3) creatstd ("my.demo");
    :
(c4) creatstd (mytag);
    :
(c5) loadfile ("new.vaxima")$
    :
(c6)
```

The user can then test the user-defined test files and tags, as well as the VaxTest-defined test files and tags, to see how they perform in the modified VAXIMA environment. The results of the initial tests should be sent to the standard output to aid in the debugging of the proposed modifications. After selective testing has been satisfactorily completed, all test files (user- and VaxTest-defined) should be tested using the 'all' test tag, with the results sent to a specific results file.

```
(c6) runtest ("my.demo");
    :
(c7) runtest (mytag);
    :
(c8) runtest (all, "my.results");
    :
(c9) exit();

%
```

At the manager level, the VaxTest facility provides an interactive tool to check the effect of proposed modifications to VAXIMA, or to the operating system, on the current VAXIMA environment. The *checkstd* command is used to create the standard files for any new or modified test files, with the *runtest* command used to check the entire VAXIMA facility for any

introduced perturbations.

Before any of the VaxTest commands are executed, all modifications to the test files or the test tags should be installed, although installation of the modifications to the VAXIMA package should wait until all standard files have been brought up to date.

modify VaxTest test file data base

```
% vaxima
   :
(c1) loadfile ("vaxtest.l")$
   :
(c2) checkstd (all);
   :
(c3) exit();

%
```

All test files should then be tested in the new VAXIMA environment to check for any perturbations not corrected at the user level.

modify VAXIMA package

```
% vaxima
   :
(c1) loadfile ("vaxtest.l")$
   :
(c2) runtest (all, "new.results");
   :
(c3) exit();

%
```

If any errors are detected, they should be isolated, corrected, and retested using the VaxTest commands. Once all test files have been tested successfully, all proposed modifications to the VAXIMA and VaxTest facilities should be permanently installed.

At the system level, the VaxTest facility provides an automatic batch facility to perform the manager's duties when new versions of VAXIMA are constructed (with the proposed modifications added) or to periodically check the current VAXIMA version. At the system level, only the standard files can be modified. There is no capability to modify the VAXIMA environment or the VaxTest test file data base. The *checkstd* command is used on all of the test files (using the old, "correct" version of VAXIMA) to ensure that all of the standard files are up to date. The *runtest* command is then used on all of the test files (using the new, proposed version of VAXIMA) to check for perturbations, if any, to the system. These commands should be placed in separate files, each one batch-executed in the appropriate VAXIMA environment using I/O redirection.

'vazima' is the current executable environment
% vaxima < check.batch > check.errors

'svazima' is the proposed environment
% svaxima < test.batch > test.errors

```
% cat check.batch
loadfile ("vaxtest.l")$
checkstd (all)$
exit()$

% cat test.batch
loadfile ("vaxtest.l")$
runtest (all, "vaxtest.out")$
exit()$
```

The results of the system test should be stored in an optional results file ("vaxtest.out" in this example), reviewed to check the status of the current VAXIMA system (error-free or error-containing).

4. INITIAL RESULTS OF VAXTEST

4.1 Creating Standard Files

Before preparing the standard files, some changes had to be made to the test files due to the restrictions placed on the VaxTest facility. First, since the *TH(i)* variable was not implemented, all commands containing this variable had to be modified. Fortunately, the modifications needed were minor and only affected two test files (*combin* and *simpl*). Secondly, all display commands (*display, dispfun,* and *letrules*) were removed from the test files. these commands are used as visual checkpoints and contribute nothing to the automatic testing of the test files.

In creating the standard files for these updated test files, several problems arose. Out of the forty-seven VaxTest-defined test files used, seven were flagged as containing erroneous code segments (*ball, begin, c2cyl, cyl2c, cyl2e, limit,* and *simpl*). Of these, six belonged to the *mit* functional group (out of eleven test files) and one belonged to the *int* functional group (out of five test files). Of the various functions used in these erroneous code segments, most were found to cause sporadic errors. While they worked fine in some test files, they performed poorly in others. On closer observation, it was found that these functions performed satisfactorily well in those test files used at the start of the test run, and caused error messages to be generated in those test files used near the end of the test run. The reason for this strange behavior was the carrying over of labels and variables defined and declared in previous test file evaluations. This caused a variable assumed to be unset in the current test file to be misinterpreted as being

bound to some value, thus resulting in the generation of unexpected error statements. This problem was easily alleviated by placing the VAXIMA command "KILL(ALL)$" at the beginning of all test files. This command has the effect of eliminating all previously defined variables and labels, giving each test file a new slate to work from.

With these changes made to the test files, the only errors that still occurred were a result of the poor design of the test files. It must be kept in mind that the test files currently in use were not specifically designed to put through such stringent tests as those performed by the VaxTest facility. For this reason, some errors will always be present when the current, albeit modified, test files are used. To "correct" these errors (at least temporarily), the command lines that still generated error diagnostics were removed from the test files.

One final error, attributable to the poor design of the test files, was the dependency of the *gen* test file on the previous evaluation of the *differ* test file. To remedy this, the *gen* test file was removed, with its contents appended to the *differ* test file. This resulted in a reduction of the number of test files to forty-six, and the number of *more* test files to fourteen.

Listed below are the changes made to the test files that resulted in the error-free creation of their standard files.

Changed in test file *array* :

```
MIDDLE&&&MAT:MATRIX([Q,V],[W,U]);
  to
MAT:MATRIX([Q,V],[W,U]);
```

Changed in test file *combin* :

```
FACTCOMB(%TH(3));
  to
FACTCOMB((N+2)*N!);
```

Removed from test file *legen* :

```
FOR I:0 THRU 4 DO DISPLAY(P[I](X),P[I](1));
... DISPLAY(Q[I])   in command #6
... DISPLAY(MOMENT[I,J])   in command #8
```

Changed in test file *limit* :

```
A*LOG(A+1)-A*LOG(A);
LIMIT(%,A,INF);
  to
X*LOG(X+1)-X*LOG(X);
LIMIT(%,X,INF);
```

Removed from test file *nisimp* :

```
LETRULES();   in commands #9, #49, and #51
LETRULES(ARULES);   in commands #36, #42, and #44
LETRULES(SUB);   in command #50
```

Changed in test file *simpl* :

```
RATIO:%TH(-3)/%;
```

to
RATIO:EXPAND((B+A)4)/%;

Removed from test file *simpl* :
FACTOR(%);
DPART(RATIO,2,4);
PART(RATIO,2,4);
(%I*V+U)/(F+%I*E)+%E(%I*ALPHA);
REALPART(%);
MAP(FACTOR,%);

Removed from test file *solve* :
ONEANS:EV("ONEANS,EVAL,NUMER);
EV(EQ,ONEANS,EXPAND);

4.2 Comparing Test Files

The test files were tested at two stages : before any changes to the test files were made, and after all errors had been corrected. In the first stage of testing, the errors that had been encountered in the creation of the standard files propagated through out the testing of the test files, resulting in numerous and unexpected mismatch errors. Some of this was due to the effect of the erroneous code segment on its neighboring commands. Most of the problems, though, were attributable to the frequent use of the *%* variable, used to record the previous command line. Unfortunately, once this variable was assigned an erroneous command segment, it adversely effected all successive command lines referencing the *%* variable. This caused an error in one command to be repeated in the next command, and on down the line, resulting in numerous mismatches (and their resultant comparison messages). These problems were cured by the removal of the initial problem commands (as mentioned earlier).

In the second stage of testing, all of the errors reported in the first stage were absent, although mismatches were still found in four of the test files. As was the case in the creation of the standard files, these mismatches can be directly attributable to the poor design of the test files. This is discussed further in the next chapter.

Listed below are the mismatch diagnostics, and their corresponding command lines in the test files, produced in the second stage of testing. All other test files were found to be equivalent, line for line, to their corresponding standard files.

Mismatch : found in 'algsys.demo' at line #15
ALGSYS([F1,F2,F3],[X,Y,Z]);
Mismatch : found in 'algsys.demo' at line #18
ALGSYS([F1,F2],[X,Y]);

Mismatch : found in 'eezgcd.demo' at line #20
EEZCONTENT(P,U);
Mismatch : found in 'eezgcd.demo' at line #21

EEZCONTENT(P,X);
Mismatch : found in 'eezgcd.demo' at line #23
EEZCONTENT(P,U);
Mismatch : found in 'eezgcd.demo' at line #24
EEZCONTENT(P,W);

Mismatch : found in 'matrix.demo' at line #9
MAT3(-1);
Mismatch : found in 'matrix.demo' at line #10
%.MAT3;

Mismatch : found in 'solve.demo' at line #22
POLYDECOMP(MOBY,S);

It should be noted that the VaxTest commands accomplished their goals : to definitively locate erroneous code segments. The diagnostics produced in the creation of the standard files and the resultant testing of the test files have helped in the location and correction of erroneous code segments in the test files. Although testing of the VaxTest facility was not intended to produce revised test files, some surface changes have been possible. This topic is also discussed in the next chapter.

5. FUTURE CONSIDERATIONS

5.1 VaxTest Facility

The VaxTest facility proposed herein is only the first step in automatic testing routines for VAXIMA. There are several features that could be added to the facility to enhance its capabilities and ensure its proper operation. First, since the VaxTest facility depends on the proper functioning of the VAXIMA commands ***alike***, ***ratsimp***, and ***meval***, some type of checking routine, front-ended to the VaxTest facility, is needed to ensure the flawless operation of these three commands. As it stands now, improper functioning of these commands will only cause erratic behavior in the VaxTest facility. Unless this behavior is quite severe, the testing facility may be incorrectly assumed to be working properly with the errors generated assumed to be the result of deficiencies elsewhere in the VAXIMA system.

Secondly, although the ***runtest*** comparison functions are able to indicate whether two lines of code are equivalent or not within a reasonable doubt, there is one more comparison function that should be implemented. This added function would go beyond the mere comparison of coding representations or evaluated results and would check to see if the difference between two evaluated VAXIMA commands is equivalent to zero. This is important if two functions, designed to do similar but different tasks, are compared. Although their answers, or the internal LISP representations of these answers, may be equivalent within a small tolerance (close to zero), they would be incorrectly flagged as being mismatches. Unfortunately, the VAXIMA

function *zeroequiv*, which would appear to satisfy these requirements, is restricted to equivalence on a single variable which must be explicitly specified in the function's parameter list. This is impractical, since it is not possible for the VaxTest facility to know beforehand which variable is being used in an equation. The problem is complicated even more when several variables are present in the equation. If these restrictions can be overcome, the ability of the *runtest* command to detect mathematically equivalent code would be more inclusive.

Also, even though messages for evaluation errors and comparison mismatches are printed, user-controlled messages for non-identical lines of code should also be possible. This would provide the user with the capability to specify whether the VaxTest command *runtest* is to flag those lines that are equivalent, but not identical. This added restriction may be necessary in certain applications of the VaxTest facility. This message would be printed after the *alike* comparison function had failed, but before any other comparison functions were tried.

Next, there are some VAXIMA functions (i.e. *solve* and *linsolve*) that return labels containing the results of the function, rather than the results themselves. The evaluated results of these labels need to be compared, not the labels themselves. This requires some means to detect these functions and treat them appropriately.

Finally, there are some VAXIMA functions which are not evaluated. Some functions have arguments that are not evaluated (i.e. *array, kill, loadfile, translate*), while other functions control the evaluation of their arguments (i.e. *ev, product, substpart, sum*). These functions may return a value (i.e. "done" or "false") that is not indicative of the results generated in their execution. These functions are difficult to test and require a means to check the results of the function other than by comparing the meaningless returned values. The operation of the VaxTest facility is very similar to the execution of these functions, making it difficult to test and difficult to validate the VaxTest facility.

This is not to say that any of the wind should be taken out of the sails of the VaxTest facility. In its current state, it supplies a variety of tools useful in the VAXIMA system. Also, its design structure can be applied easily to automated tools used for other software systems similar to VAXIMA and MACSYMA. And, although it is only the initial step in automated tools for VAXIMA and MACSYMA, no further advances would be possible without tools similar to those supplied by the VaxTest facility.

5.2 Test Files

Probably the biggest area requiring further work is the design and construction of the VaxTest test files. Through the testing and use of the VaxTest facility, it was found that the current test files have many flaws and require a critical reevaluation and revision of their design. Several of the test files contain erroneous code segments (as described in Chapter 6) that must be modified or removed. Also, there must be an effort made to ensure better testing coverage of the VAXIMA system by the VaxTest facility. Currently, several separate test files contain repetitious code segments that test the same VAXIMA functions. Although some overlap is needed, much of it is overly repetitious. Also, some of the VAXIMA functions escape testing due to their exclusion from the test files. A critical path analysis of the VAXIMA facility is needed, along with a complete revision of the current test files. It would also be advantageous to identify the critical VAXIMA functions (those most often used), since modifications in these

functions would have the greatest effect on the entire VAXIMA system. Finally, the test files need to be reclassified according to the class of VAXIMA functions they test. The current classification scheme (using test tags) is based more on the origin of the test files and not on their testing capabilities. Use of the VaxTest commands in the revision of its test files should prove invaluable in providing rapid debugging and testing aids. Therefore, use of the VaxTest facility creates a cycle of continuous improvement. As it is used, refinements are made in the capabilities of the VaxTest commands. These new abilities provide more improved tools to aid in further refinements of the VAXIMA system and the VaxTest facility.

REFERENCES

1. Adrion, W. R., Branstad, M. A., and Cherniavsky, J. C. "Validation, Verification, and Testing of Computer Software." *ACM Computing Surveys* 14 (Apr 1982), 159-192.
2. Beizer, B. *Software Testing Techniques.* New York : Van Nostrand Reinhold, 1983.
3. Deutsch, M. S. *Software Verification and Validation.* Englewood Cliffs, NJ : Prentice-Hall, 1982.
4. Fairley, R. E. "An Experimental Program-Testing Facility." *IEEE Transactions on Software Engineering* SE-1 (Dec 1975), 350-357.
5. Foderaro, J. K., and Sklower, K. L. *The FRANZ LISP Manual.* Berkeley : Univ. of California Press, 1981.
6. Gannon, C. "Error Detection Using Path Testing and Static Analysis." *Computer* 12 (Aug 1979), 26-32.
7. Goodenough, J. B. "A Survey of Program Testing Issues." in *Research Directions in Software Technology.* Ed. P. Wegner. Cambridge, MA : MIT Press, 1979.
8. Hetzel, W., ed. *Program Test Methods.* Englewood Cliffs, NJ : Prentice-Hall, 1973.
9. Howden, W. E. "Applicability of Software Validation Techniques to Scientific Programs." *ACM Transactions on Programming Languages and Systems* 2 (Jul 1980), 307-320.
10. Howden, W. E. "Functional Program Testing." *IEEE Transactions on Software Engineering* SE-6 (May 1980), 162-169.
11. Huang, J. C. "An Approach to Program Testing." *ACM Computing Surveys* 7 (Jul 1975), 113-128.
12. MIT Mathlab Group. *MACSYMA Reference Manual.* Cambridge, MA : MIT Press, 1983.
13. Myers, G. J. *The Art of Software Testing.* New York : John Wiley & Sons, 1979.
14. Ramamoorthy, C. V., and Ho, S. F. "Testing Large Software with Automated Evaluation Systems." *IEEE Transactions on Software Engineering* SE-1 (Jan 1975), 46-58.
15. Voges, U., Gmeiner, L., and von Mayrhauser, A. A. "SADAT - An Automated Test Tool." *IEEE Transactions on Software Engineering* SE-6 (May 1980), 286-290.
16. Winston, P. H., and Horn, B. K. P. *LISP.* Reading, MA : Addison-Wesley, 1981.

USING MACSYMA TO GENERATE (SOMEWHAT) OPTIMIZED FORTRAN CODE

Leo P. Harten
Paradigm Associates, Inc.
29 Putnam Ave., Suite 6
Cambridge, MA 02139
and
MIT
Cambridge, MA 02139

Abstract

MACSYMA can: manipulate expressions with high level commands; optimize expressions by replacing common sub-expressions with temporary variables; and generate FORTRAN code from matrices and simple expressions (e.g., arithmetic operators and standard FORTRAN numerical functions, or those without any control structures. Examples of control structures are BLOCK, IF-THEN-ELSE, DO loops, etc.)

This work describes a program, FORT, which outputs FORTRAN code from the more complicated format resulting from the optimization of matrices and simple expressions. The resulting code is not usually completely optimal from the standpoint of efficiency or numerical accuracy, but is much more nearly certain to be correct than a hand-coded effort. Various deficiencies are pointed out, and some possible extensions are considered.

1. INTRODUCTION

MACSYMA, as documented in Reference [1], has many commands to perform algebraic manipulations, calculus, etc. at a very high level, e.g., DIFF(F(X),X), SOLVE([EQ1,...,EQN],[X1,...,XN]), INTEGRATE(F(X),X,A,B), etc. The user can generate extremely complicated and lengthy expressions involving arithmetic operators and the standard transcendental functions using such commands. As long as Fortran compilers recognize the operators and functions, the expression will be considered "simple".

The FORTRAN command generates FORTRAN code from simple expressions of the form X:3*Y+SIN(Z-Y↑2), and the FORTMX command outputs FORTRAN code for matrices. Neither of these FORTRAN-producing commands has any ability to optimize the output, however, and much re-computation is performed in many cases of their use. Reference [2] makes such use of MACSYMA, while Reference [3] incorporates optimizations for Jacobians.

The OPTIMIZE command is capable of searching for common sub-expressions and replacing them with temporary variables. Passing the arguments to the FORTRAN command through an optimization procedure is useful in improving the efficiency of the output FORTRAN code. Limitations on OPTIMIZE prevent total optimality in the sense of efficiency, and a human being can make improvements as far as efficiency is concerned, though errors are apt to be introduced by such efforts. The work in

Reference [4] has been concerned with extreme efficiency. The results of Reference [5], based on Kalaba's Table Algorithm (itself an extension of Wengert's method of Reference [6]), are generally quite inefficient since they use FORTRAN subroutines to yield exact forms of derivatives, but have no simplifications such as 0*exp(x)=0, and thus compute the exponential many more times than is needed.

The HORNER command converts polynomial expressions into a nested form which generally requires fewer operations, and thus often increases efficiency and stability when applicable.

This paper presents some results for simple expressions and matrices as an illustration of the abilities which could be extended to more general constructs (DO loops, IF-THEN-ELSE clauses, ARRAYs, etc.). The on-line work by Carrette has handled DO loops which can be nested, but which do not contain other constructs. Some limitations and mis-features are discussed.

2. PROGRAM FOR GENERATION OF FORTRAN CODE

The MACSYMA code which generates FORTRAN from the optimized form of expressions and matrices is presented in the Appendix. The MACSYMA Reference Manual documents the various commands used in the program, and the code is commented (material enclosed within /* */).

The FORTRAN-generating functions can be put into the Multics MACSYMA environment (if you have been granted access) via the command BATCH(">udd>Paradigm>Harten>f>mfn.1"); , and into the MC MACSYMA environment via LOAD("LPH\;MFN FASL"); typed on an input line. Contact the author for versions to run on other hardware. The output can be viewed only on the terminal unless some terminal session listing is being made (e.g., via file_output/revert_output or attach_audit/ detach_audit on Multics, or writefile/closefile on MC, VAX, and Multics).

The user can transact with MACSYMA to generate an expression or matrix for which FORTRAN code is desired. Then the user calls the function FORT with two arguments: first, a name, to be used as the function name or the array name; and then either an expression to be used as the body of the function definition, or a matrix. The program will determine which mode of input has been given, and pass the second argument to an appropriate sub-program that produces the correct type of output. The user decides whether or not the HORNER function should be used on the expression and types true or false accordingly (try both ways to see the difference); and then enters a prefix for the temporary variables to be used, so that one can be selected which does not have a name conflict with the user's other FORTRAN code (a typical name is 'w, the single quote being used to cause the symbol w to be read, rather than the value of the symbol.)

Upon completion of this step, the user may then leave MACSYMA and view the FORTRAN code (if a saving mechanism was employed) in a text editor (such as EMACS), and insert it in the main program. By using simple commands, the text editor will cosmetically improve the expressions, so that line breaks do not come in the middle of a variable name, for example. The user may wish to leave the code in this state, confident that the expressions are correct (unless the user has introduced errors while moving the lines around), or he may desire to proceed with a hand-optimization procedure. The latter will improve efficiency of the code, but is the major source of errors in the final version, since it is unlikely that MACSYMA has a bug which caused a wrong result.

The basic techniques in hand-optimization are to search for products of temporary variables that occur more than once and to apply some special transformation rules. These requirements may arise in several ways, such as: having temporary variables VAR1=X**2 and VAR2=X**3, where one can do better by instead having VAR1=X**2 and VAR2=X*VAR1 (this leaves out the simpler question of whether X**2 is more/less/same efficient than X*X, which many FORTRAN compilers will take care of anyway); or the final expression may contain two (or more) terms of the form VAR1*VAR6 and VAR1*VAR3*VAR6, and the common product VAR1*VAR6 can be extracted as a new temporary variable; or there may be a factored form of an expression which is more efficient to compute; the ordering of the temporary variables may be

inefficient, with VAR1=X**4 followed by VAR2=X**3, and then VAR1=X**3, VAR2=X*VAR1 is better (then the occurrences of VAR1 and VAR2 in the later expressions must be interchanged); and there will always be the open question of using identities for simplification, such as 1-SIN(X)↑2-COS(X)↑2 being replaced by 0, which the user had an opportunity to examine while using MACSYMA before generating the FORTRAN code.

3. EXAMPLES OF FORTRAN CODE GENERATION

MACSYMA takes input on numbered C lines, and when a semi-colon (;) terminates the input the output is placed on the same-numbered D line; a dollar-sign ($) causes the printout of the output to be suppressed. The output is displayed in a text-book two dimensional format.

Here is a sample output from using the FORTRAN code generator on an expression. Comments have been placed prior to the C-lines in the /* comment */ form.

```
/* Let e be an expression */
(C21) e:x↑2*sin(x↑4)*exp(-x↑3);
Time= 51 msec.
                                      3
                              2    - X          4
(D21)                        X   %E      SIN(X )
/* To make it more interesting,
   differentiate e 3 times */
(C22) diff(e,x,3);
Time= 1098 msec.
                        3                           3
          10         - X           4        8    - X           4
(D22) 144 X     %E       SIN(X ) - 27 X   %E        SIN(X )

                    3                        3
          7      - X           4        5     - X          4
 - 240 X   %E        SIN(X ) + 108 X   %E        SIN(X )

                   3                        3
         2      - X          4        11     - X          4
 - 60 X   %E        SIN(X ) - 64 X    %E        COS(X )

                    3                        3
          9      - X           4        6     - X          4
 + 108 X   %E        COS(X ) - 324 X   %E        COS(X )

                    3
          3      - X           4
 + 120 X   %E        COS(X )
```

```
/* Use the default FORTRAN program as a basis for
   comparing the optimizing program FORT */
(C23) fortran(%);
      144*X**10*EXP(-X**3)*SIN(X**4)-27*X**8*EXP(-X**
     1   3)*SIN(X**4)-240*X**7*EXP(-X**3)*SIN(X**4)+1
     2   08*X**5*EXP(-X**3)*SIN(X**4)-60*X**2*EXP(-X*
     3   *3)*SIN(X**4)-64*X**11*EXP(-X**3)*COS(X**4)+
     4   108*X**9*EXP(-X**3)*COS(X**4)-324*X**6*EXP(-
     5   X**3)*COS(X**4)+120*X**3*EXP(-X**3)*COS(X**4
     6   )
Time= 298 msec.
(D23)                          DONE
/* Note that the EXP, SIN, and COS are re-computed
   quite often, and each time the argument is an
   exponentiated quantity. Now use the FORT command
   and request HORNER */
(C24) fort(e3rd_der,%th(2));
enter TRUE to use Horner's rule, FALSE to avoid it
true;
enter the prefix for temporary variables, e.g., 'var
'w;

      real function E3RD_DER(X)
      W1 = X**3
      W2 = X**2
      W3 = X**4
      E3RD_DER = EXP(-W1)*(W2*SIN(W3)*(W1*(W2*((144.0
     1   *W2-27.0)*X-240.0)+108.0)-60.0)+W1*(W1*(W1*(
     2   108.0-64.0*W2)-324.0)+120.0)*COS(W3))
      return
      end
Totaltime= 2301 msec.  GCtime= 586 msec.
(D24)                          DONE
/* This took a lot less arithmetic operations overall:
   saved the argument and the transcendental function
   re-computation; and HORNER reduced the number of
   exponentiations to just those in the three
   temporary variables.

   Now for a matrix of derivatives.
   Define an array with which to associate the matrix */
(C25) g[i,j]:=diff(x↑2*sin(x↑2*y)+exp(-x↑2*y↑2),x,i,y,j);
Time= 4 msec.
                        2      2              2  2
(D25) G      := DIFF(X  SIN(X  Y) + EXP(- X  Y ), X, I, Y, J)
       I, J
/* generate a 2x2 matrix from the array g */
(C26) genmatrix(g,2,2)$
Totaltime= 3685 msec.  GCtime= 1086 msec.
```

```
/* Generate the default FORTRAN code */
(C27) fortran('mess=%);
      MESS(1,1) = -2*X**5*Y*SIN(X**2*Y)+4*X**3*COS(X*
     1   *2*Y)+4*X**3*Y**3*EXP(-X**2*Y**2)-4*X*Y*EXP(
     2   -X**2*Y**2)
      MESS(1,2) = -6*X**5*SIN(X**2*Y)-2*X**7*Y*COS(X*
     1   *2*Y)-8*X**5*Y**4*EXP(-X**2*Y**2)+20*X**3*Y*
     2   *2*EXP(-X**2*Y**2)-4*X*EXP(-X**2*Y**2)
      MESS(2,1) = -18*X**4*Y*SIN(X**2*Y)-4*X**6*Y**2*
     1   COS(X**2*Y)+12*X**2*COS(X**2*Y)-8*X**4*Y**5*
     2   EXP(-X**2*Y**2)+20*X**2*Y**3*EXP(-X**2*Y**2)
     3   -4*Y*EXP(-X**2*Y**2)
      MESS(2,2) = 4*X**8*Y**2*SIN(X**2*Y)-30*X**4*SIN
     1   (X**2*Y)-26*X**6*Y*COS(X**2*Y)+16*X**6*Y**6*
     2   EXP(-X**2*Y**2)-80*X**4*Y**4*EXP(-X**2*Y**2)
     3   +68*X**2*Y**2*EXP(-X**2*Y**2)-4*EXP(-X**2*Y*
     4   *2)
Totaltime= 1265 msec.  GCtime= 579 msec.
(D27)                          DONE
/* Use FORT to avoid the re-computations
   that appear above */
(C28) fort(mess_mat,%th(2));
enter TRUE to use Horner's rule, FALSE to avoid it
true;
enter the prefix for temporary variables, e.g., 'var
'z;

      Z1 = X**2
      Z2 = Y**2
      Z3 = EXP(Z1*Z2)
      Z4 = 1/Z3
      Z5 = -4.0*X
      Z6 = X**3
      Z7 = Y*Z1
      Z8 = COS(Z7)
      Z9 = X**5
      Z10 = SIN(Z7)
      Z11 = X**4
      Z12 = X**6
      MESS_MAT(1,1) = Z4*(-2.0*Y*Z10*Z3*Z9+4.0*Z3*Z6*
     1   Z8+Y*(4.0*Z2*Z6+Z5))
      MESS_MAT(1,2) = Z4*(Z2*(20.0*Z6-8.0*Z2*Z9)-6.0*
     1   Z10*Z3*Z9-2.0*X**7*Y*Z3*Z8+Z5)
      MESS_MAT(2,1) = Z4*((12.0*Z1-4.0*Z12*Z2)*Z3*Z8-
     1   18.0*Y*Z10*Z11*Z3+Y*(Z2*(20.0*Z1-8.0*Z11*Z2)
     2   -4.0))
      MESS_MAT(2,2) = Z4*(-26.0*Y*Z12*Z3*Z8+Z10*(4.0*
     1   X**8*Z2-30.0*Z11)*Z3+Z2*(Z2*(16.0*Z12*Z2-80.
     2   0*Z11)+68.0*Z1)-4.0)
Totaltime= 5649 msec.  GCtime= 1769 msec.
(D28)                          DONE
```

While there still appear to be a fair number of lines of code here, it is important to recognize that the time-consuming re-computations are gone: the exp, sin, and cos are all computed exactly once; nearly all of the exponentiations are done in the temporary variables; and the consequent run-time saving is quite large as a result.

4. DISCUSSION OF EXAMPLES

The expression for the third derivative of e was quite substantially improved for FORTRAN calculation by the processing in FORT. Table 1 compares operations counts for FORTRAN and FORT with categories: addition/subtraction; multiplication/division; monomial exponentiation; and transcendental (exp, sin, and cos). The difference in run-times would be quite substantial: using cycle-times for the categories of 1, 4, 8, and 16, respectively, the projected number of cycles is shown in the last column.

Table 1. Comparison of operations counts: FORTRAN and FORT for third derivative of e

Operation:	+/-	*//	**	exp,sin,cos	cycles
FORTRAN	17	27	27	18	617
FORT	8	12	3	3	128

The optimization in FORT saves nearly 80% of the cycles. Of course, an even greater extreme can be reached by simplifying an expression to 0 and thus saving all but one cycle for the binding, and MACSYMA will enable the user to investigate a variety of simplifications.

The matrix was computed with y being the main variable by default. To see if there is any difference when x is made the main variable (by ratvars(x)), the following is done:

```
(c29) (ratvars(x),fort(mess_mat,d26));
enter TRUE to use Horner's rule, FALSE to avoid it
true;
enter the prefix for temporary variables, e.g., 'var
't;
      T1 = X**2
      T2 = Y**2
      T3 = EXP(T1*T2)
      T4 = 1/T3
      T5 = -4.0*Y
      T6 = Y**3
      T7 = T1*Y
      T8 = COS(T7)
      T9 = SIN(T7)
      T10 = Y**4
      MESS_X(1,1) = T4*X*(T1*(-2.0*T1*T3*T9*Y+4.0*T3*
     1   T8+4.0*T6)+T5)
      MESS_X(1,2) = T4*X*(T1*(T1*(-2.0*T1*T3*T8*Y-6.0
     1   *T3*T9-8.0*T10)+20.0*T2)-4.0)
      MESS_X(2,1) = T4*(T1*(T1*(-8.0*Y**5-18.0*T3*T9*
     1   Y-4.0*T1*T2*T3*T8)+12.0*T3*T8+20.0*T6)+T5)
      MESS_X(2,2) = T4*(T1*(T1*(T1*(16.0*Y**6-26.0*T3
     1   *T8*Y+4.0*T1*T2*T3*T9)-30.0*T3*T9-80.0*T10)+
     2   68.0*T2)-4.0)
```

There appears to be some improvement: only 10 t's instead of 12 z's; and slightly shorter lines. Both of the FORT codes are apparently much more efficient than the default FORTRAN. For multi-variate expressions it is often difficult to determine which ordering of the variables will produce the best code from

the combination numerical stability - run-time viewpoint. The trial method appears to be acceptable for modest size routines such as have been illustrated herein; though a combinatorially-explosive decision procedure could be implemented to count machine cycles for each of the permutations of multi-variate cases, and then pick the minimum run-time code(s).

5. CONCLUSIONS AND SUGGESTED FURTHER WORK

It has been demonstrated that a MACSYMA code can produce reasonable FORTRAN code in an (essentially) error-free computer algebra environment. The extra optimizations that a human can perform might introduce mistakes in the final FORTRAN; but the use of MACSYMA to perform manipulations (such as HORNER or FACTOR or RATVARS) can lead to alternative, possibly somewhat more efficient, FORTRAN codes than it would otherwise do by default, though not as efficient as those which are correctly hand-optimized.

Extensions of the MACSYMA code to other structures, such as DO loops, IF-THEN-ELSE, etc. could be written; doing the coding for all such MACSYMA-->FORTRAN statements in LISP would be much more efficient and avoid duplication of common code. Some progress has already been made in this regard.

The various MACSYMA commands which have no analogue in FORTRAN could be handled as calls to FORTRAN subroutines--e.g., if MACSYMA has a piece of code INTEGRATE(EXP(-X)/X,X,1,Z), then the MACSYMA program will realize that this cannot be performed in closed-form, and should cause the FORTRAN code to call a numerical quadrature routine (such as DCADRE) as part of the output. This gets into the AI aspects of the problem of a general MACSYMA-->FORTRAN translator.

The OPTIMIZE command could be improved to handle the problem with re-computation of expressions such as V1=X**4 and V2=X**2, in order to improve the output. A common mis-feature of Macsyma is to convert an input of x-y into an output of x+(-1.0*y) when numer:true is done to preserve floating point numbers that appear in expressions; and this could perhaps be fixed by using the internal representation of x-y prior to conversion to FORTRAN.

6. ACKNOWLEDGMENTS

The author would like to thank George Carrette, Richard Brenner, Jeff Golden (who generated promptly upon request an enhancement to the FORTRAN command so that the line length of the output code would meet the conference requirements), and Dr. Richard Pavelle for their interest in optimization and this program; and he is especially grateful to Prof. Abraham Bers and Dr. Abhay Ram of MIT for the plasma physics problems which

initially interested him in the idea of translation of MACSYMA expressions to optimized FORTRAN code.

Useful discussions with Dr. Ralph Wilcox, Prof. Robert Kalaba and Dr. V. K. Murthy of Hughes Aircraft Co. EDSG were held on the subject of the simultaneous computation of a function and its derivatives, and a paper on same, co-authored with Dr. Wilcox, is being submitted to the Journal of Applied Mathematics and Computation.

Dr. Grant Cook of LLNL also contributed some insights from his experience with LISP-coded optimization routines.

7. REFERENCES

1. MACSYMA Reference Manual, The Mathlab Group, Laboratory for Computer Science, MIT, Version 9, 1977

2. Anderson, J. D., Lau, E. L., and Hellings, R. W. Use of Macsyma as an Automatic Fortran Coder, pp. 583-595 Second Macsyma Users' Conference Proceedings, Edited by V. Ellen Lewis, MIT Lab for Computer Science, Cambridge, MA 02139 (1979)

3. Ng, E., and Char, B. Gradient and Jacobian Computation for Numerical Applications, pp. 604-621, op. cit.

4. Cook, Jr., G. O. Development of a Magnetohydrodynamic Code for Axisymmetric, High-Beta Plasmas with Complex Magnetic

Fields, Ph.D. thesis, published as UCRL-53324, Lawrence Livermore National Laboratory, Univ. of Calif., Livermore, CA 94550 (1982)

5. Kalaba, R., Rasakhoo, N., and Tishler, A. Nonlinear Least Squares via Automatic Derivative Evaluation, J. App. Math. and Comp., Vol. 12, pp. 119-137 (1983)

6. Wengert, R. A simple automatic derivative evaluation program, Comm. ACM Vol. 7, pp. 463-464 (1964)

7. On-line reference to programs and work by Grant Cook and George Carrette

APPENDIX

Here is the MACSYMA source code for the MACSYMA-->FORTRAN generator.

Code and comments are

Historical Note: The first version of the program to handle expressions and matrices was only 18 lines long (but rather densely packed; it was not very forgiving about "erroneous" input; and it lacked even the modest options presented herein.)

/* FORT(FUN_NAME,FUN_EXP) : macsyma-->fortran translation of FUN_EXP is done by testing to see if FUN_EXP is an expression or a matrix.

If FUN_EXP is an expression, then the function FUN_NAME is created with the n arguments (x1, x2, ... , xn) given by the list of variables in FUN_EXP, and the body of the function definition is the expression FUN_EXP. The body is passed through an optimization routine which generates m temporary variables (a1, a2, ... , am) to replace common sub-expressions (exp1, exp2, ... , expm), and a final expression composed of the x's and a's (exp_a1_am); and the output is legal fortran:

```
real function fun_exp(x1,x2,...,xn)
a1=exp1
a2=exp2
  .
  .
  .
am=expm
fun_exp=exp_a1_am
return
end
```

If FUN_EXP is a p-by-q matrix, then the matrix elements are passed through an optimization routine which generates m temporary variables (a1, a2, ... ,am) to replace common sub-expressions (exp1, exp2, ... , expm), and a final

expression which is a matrix whose elements are optimized combinations of the a's; and the output is legal fortran:

```
      a1=exp1
      a2=exp2
           .
           .
           .
      am=expm
      fun_name(1,1)=<optimized form of the 1,1 element>
      fun_name(1,2)=<optimized form of the 1,2 element>
           .
           .
           .
      fun_name(p,q)=<optimized form of the p,q element> */


FORT(FUN_NAME,FUN_EXP)::=
  /* DEFINE THE MACRO FORT */

        (MODE_DECLARE([FUN_NAME,FUN_EXP,FUNCTION(FORT)],ANY),
  /* DECLARE TYPE ANY */

         BUILDQ([FUN_NAME,FUN_EXP],
  /* USE THE BUILDQ CONSTRUCT TO PASS INPUT */

                BLOCK([OPTIMPREFIX,HORNERP,RATPRINT,NUMER],
  /* USE TEMPORARY VARIABLES IN THE BLOCK */

                      MODE_DECLARE([OPTIMPREFIX,
                                    FUNCTION(EXPFORT,GENFUN,
                                       OPTIMIZE,
                                       MATFORT,HORNER)],
                                   ANY,
                                   FUNCTION(LISTOFVARS),
                                   LIST,
                                   [RATPRINT,
                                    FUNCTION(MATRIXP),
                                    HORNERP,NUMER],
                                   BOOLEAN),
  /* DECLARE THE TYPES OF VARIOUS THINGS */
                      RATPRINT:FALSE,
  /* ASSIGN VALUE TO PREVENT INTERRUPTION OF OUTPUT WITH
     NOTIFICATIONS ABOUT APPROXIMATIONS OF REAL NUMBERS
     BY RATIONALS */

                      NUMER:TRUE,
  /* GET EVERYTHING DONE IN FLOATING POINT */

                      HORNERP:READ(
        "enter TRUE to use Horner's rule, FALSE to avoid it"),
```

```
                          OPTIMPREFIX:READ(
   "enter the prefix for temporary variables, e.g., 'var"),
 /* ALLOW THE USER TO SELECT THE PREFIX THAT WILL
    APPEAR IN OUTPUT */


                          IF MATRIXP(FUN_EXP)
 /* TEST INPUT TO SEE IF IT'S A MATRIX */

                               THEN MATFORT(FUN_NAME,
                                      OPTIMIZE(
                                        MATRIXMAP(
                                          LAMBDA([U],
                                                 IF HORNERP THEN
                                                   HORNER(U)
                                                   ELSE U),
                                                     FUN_EXP)))
 /* IF A MATRIX, RUN THE MATFORT FUNCTION ON AN OPTIMIZED AND
    OPTIONALLY HORNERED INPUT */

                               ELSE EXPFORT(
                                      APPLY('GENFUN,
                                            ['FUN_NAME,
                                             LISTOFVARS(FUN_EXP),
                                       OPTIMIZE(APPLY(LAMBDA([U],
                                            IF HORNERP THEN
                                               HORNER(U)
                                                 ELSE U),
                                            [FUN_EXP])))])),
 /* IF NOT A MATRIX, RUN THE EXPFORT FUNCTION ON A FUNCTION
    GENERATED FROM THE FUN_NAME, THE VARIABLES IN THE BODY,
    AND AN OPTIMIZED AND OPTIONALLY HORNERED FORM OF THE
    BODY */

                         RETURN('DONE))))$
/* RETURN THE ATOM "DONE" AND CLOSE THE DEFINITION */


EXPFORT(INP):=
  /* DEFINE A FUNCTION EXPFORT */

       (MODE_DECLARE([FUNCTION(EXPFORT),INP],ANY),
  /* DECLARE TYPE ANY */

        BLOCK([PART2,F_NAME,ARG_LIST],
  /* USE TEMPORARY VARIABLES IN BLOCK */
```

```
            MODE_DECLARE([PART2,F_NAME,FUNCTION(FORTRAN)],
                         ANY,
                         ARG_LIST,LIST,
                         FUNCTION(SYMBOLP),BOOLEAN),
/* DECLARE THE VARIOUS TYPES */

            IF SYMBOLP(F_NAME:PART(INP,1,0))
/* LET F_NAME BE THE OPERATOR OF THE FIRST
   PART OF INP, AND TEST TO SEE IF IT IS A SYMBOL */

              THEN (IF PART(PART2:PART(INP,2),0)='BLOCK
/* LET PART2 BE THE SECOND PART OF INP, AND TEST TO
   SEE IF THE OPERATOR IS THE ATOM "BLOCK" */

                    THEN (PRINT("      REAL FUNCTION",
                                PART(INP,1)),
/* PRINT THE INDENTED FORTRAN CODE FOR A FUNCTION HEADER */

                          MAP('FORTRAN,
                              ARG_LIST:
                                REST(
                                  REST(
                                    SUBST("=",
                                          ":",
                                          ARGS(PART2))),
                                  -1)),
/* REPLACE THE :'s WITH ='s IN THE ARGUMENTS OF THE BLOCK,
   AND LET ARG_LIST BE WHAT'S LEFT AFTER REMOVING THE
   FIRST AND LAST PIECES. THEN RUN MACSYMA'S FORTRAN
   COMMAND OVER THE LIST, GENERATING THE FORTRAN CODE
   FOR THE TEMPORARY VARIABLES. */

                          FORTRAN(F_NAME=LAST(PART2)),
/* GENERATE THE VALUE OF THE FUNCTION */

                          PRINT("      RETURN"),
/* PRINT THE RETURN */

                          PRINT("      END"),
/* PRINT THE END */

                          RETURN('DONE))
/* RETURN THE ATOM "DONE" TO MACSYMA */

                    ELSE ERROR("INPUT NOT OF RIGHT FORM",
                               INP))))$
/* COMPLAIN IF THE INPUT WAS NOT OF THE RIGHT FORM. THIS IS
   NOT A VERY EXTENSIVE ERROR HANDLER... */
```

```
MATFORT(FUN_NAME,OPT_MAT):=
  /* DEFINE FUNCTION MATFORT */

        (MODE_DECLARE([FUN_NAME,OPT_MAT,FUNCTION(MATFORT)],ANY),
  /* DECLARE TYPE ANY */

         BLOCK([PART2,LPART],
  /* USE BLOCK WITH TEMPORARY VARIABLES */

               MODE_DECLARE([PART2,LPART],LIST,
                            FUNCTION(FORTRAN,FORTMX),ANY),
  /* DECLARE TYPES */

               PART2:SUBST("=",":",ARGS(OPT_MAT)),
  /* REPLACE THE :'s WITH ='s IN THE ARGUMENTS OF THE BLOCK
     REPRESENTING THE OPTIMIZED MATRIX, AND LET PART2 BE THE
     RESULTING LIST. */

                LPART:LAST(PART2),
  /* LET LPART BE THE LAST PART OF PART2 */

               IF PART(OPT_MAT,0)='BLOCK
  /* TEST TO SEE THAT THE MAIN OPERATOR IS THE ATOM "BLOCK" */

                 THEN (IF PART(LPART,0)='MATRIX
  /* TEST THE OPERATOR OF THE LAST ARGUMENT TO THE BLOCK TO SEE
     IF IT IS THE ATOM "MATRIX" */

                           THEN (MAP('FORTRAN,
                                     REST(REST(PART2),-1)),
  /* RUN MACSYMA'S FORTRAN COMMAND OVER THE TEMPORARY
     VARIABLES */

                                 FORTMX(FUN_NAME,LPART),
  /* RUN MACSYMA'S FORTMX COMMAND OVER THE MATRIX ELEMENTS */

                                 RETURN('DONE))
  /* RETURN THE ATOM "DONE" TO MACSYMA */

                 ELSE ERROR("WAS EXPECTING MATRIX, NOT",
                                          PART(LPART,0)))
  /* COMPLAIN IF THE BLOCK DID NOT HAVE A MATRIX */

                 ELSE APPLY('FORTRAN,[FUN_NAME=OPT_MAT])))$

GENFUN(NAME,VARLIST,EXP)::=
  /* DEFINE THE MACRO GENFUN */

       BUILDQ([NAME,VARLIST,EXP],NAME(SPLICE(VARLIST)):=EXP)$
  /* GENERATE A FUNCTION NAME WITH ARGUMENTS VARLIST AND
     BODY EXP */
```

/* THE CODE HAS BEEN DEVELOPED FURTHER THAN WAS SHOWN HERE, BUT THE ESSENTIAL FEATURES ARE EVIDENT:

- DISCRIMINATE THE CASES AT A HIGH LEVEL (MATRIX OR EXPRESSION);
- USE THE PART FUNCTIONS AND PREDICATES TO SELECT OUT THE MOST IMPORTANT FEATURES;
- USE THE LISP-LIKE MAP AND LAMBDA TO HANDLE LISTS, WHICH ARE A PARTICULARLY VALUABLE DATA STRUCTURE;
- USE HIGH LEVEL UTILITY ROUTINES LIKE PRINT AND FORTRAN TO OUTPUT RESULTS IN A STANDARD FORMAT;
- USE TEMPORARY VARIABLES IN BLOCKS TO AVOID GLOBAL VARIABLES THAT CAN CONFLICT, AND PASS THE TEMPORARIES AS ARGUMENTS OF FUNCTIONS;
- USE MACROS TO OBTAIN MORE CONTROL OVER THE EVALUATION OF ARGUMENTS; AND
- DON'T DO EVERYTHING ALL AT ONCE (FILE HANDLING CAME LATER, FOR EXAMPLE, TO MAKE THE OUTPUT MORE USABLE.) */

MACROS, TRANSLATION, AND COMPILATION IN MACSYMA

George J. Carrette and Leo P. Harten

Paradigm Associates, Inc.
29 Putnam Avenue, Suite 6
Cambridge, MA 02139
and
MIT
Cambridge, MA 02139

Abstract

The use of Macros [1,2], and the MACSYMA --> LISP Translator/Compiler for MACSYMA [1] on the MIT-MC KL10 and MIT-MULTICS is described. For numerical computations, such use can increase speed by factors of from 5 to 245; and for symbolic manipulations there are moderate gains in speed of execution, loading of code, and size reduction of code representation resulting in lower garbage collection overhead.

Macros define transformations that are carried out on the arguments, and the resulting expression is then sent to the MACSYMA interpreter. The MACSYMA --> LISP Translator converts interpreted MACSYMA code into LISP and can generate compiler declarations. The Compiler takes the LISP code and produces a file containing machine code.

Translation/Compilation usage is described with examples in numerical applications, with indications

of "preferred methods" found to be easy to debug and understand.

1. INTRODUCTION

The LISP language uses macros to pass code constructed from the macro's un-evaluated arguments to the interpreter. Macros provide a variety of advantages: they allow syntactic substitutions to be made, and can thus improve readability of source code: and they can provide increased efficiency in open-compilation. Macros have been implemented in MACSYMA for the same purposes, and provide the opportunity to consider programs as data. Section 2 discusses the use of Macros in MACSYMA and gives some examples of general interest.

The MACSYMA --> LISP Translator and a variety of rules/guidelines are presented in Section 3. These show the types of benefits which result from the modest additional effort that the user must make to obtain proper translation.

The Compiler is discussed in Section 4, and some examples are presented and analyzed.

General discussion and conclusions constitute Section 5.

2. MACSYMA MACROS

The LISP language uses macros to allow more selective control in evaluation of arguments, or to provide special variable or control environments, or to otherwise extend the interpreter in ways which are not possible with the procedure defining mechanism alone. MACSYMA macros have been implemerted for the same reasons.

The action of a macro is to process its symbolic argument forms and perform some transformation upon them, returning a new form which is then passed to the MACSYMA interpreter for evaluation. Thus macros enable the user to generate code from a formal input specification which is itself code, an operation analogous to what could be done by hand by a programmer.

The BUILDQ construct is useful in defining macros. It takes two arguments, a list of parameters with optional bindings, and an expression. The parameters are evaluated left to right, and then their values are substituted in parallel into the expression. The resulting expression is not evaluated. When a macro is defined using the BUILDQ construct, the code resulting from the BUILDQ is passed to the MACSYMA interpreter for evaluation. The demo file MC:REH;BUILDQ DEMO includes the basics of usage.

Here is an example of using BUILDQ:

```
A:X$

BUILDQ([A,B:4],A:6*B);
```

which returns X:24, but does not evaluate this returned result.

The special keyword SPLICE allows the user to insert a list containing its argument in the expression, so that

```
BUILDQ([FUN,B:[X,Y,Z]],
       (MODE_DECLARE([SPLICE(B),FUNCTION(FUN)],FLOAT),
        FUN(SPLICE(B))));
```

yields (MODE_DECLARE([X,Y,Z,FUNCTION(FUN)],FLOAT),FUN(X,Y,Z)).

Macros are defined with "::=" in the same way that functions are defined with ":=" . To define a macro PUSH that adds VALUE to the front of the list STACK, do

```
PUSH(VALUE,STACK)::=BUILDQ([VALUE,STACK],
                           STACK:CONS(VALUE,STACK));
```

and when S:[]$, PUSH(A,S); is done, [A] is returned and S is now bound to [A]. Then PUSH(B,S); returns [B,A] and binds S to [B,A].

A STACK can be POP'ed by the macro

```
POP(STACK)::=BUILDQ([STACK],
                    BLOCK([F],
                          F:FIRST(STACK),
                          STACK:REST(STACK),
                          RETURN(F)));
```

so that now POP(S); will return B and bind S to [A], after which POP(S); will return A and bind S to [].

Without macros, one must use the DEFINE function to generate a function definition with a body and argument list specified inside the named function:

```
DEFINE(F(X,Y,Z),X*Y↑2+Z);
```

and this will yield

```
F(X,Y,Z):=X*Y↑2+Z .
```

Using macros, one can do more powerful definitions:

```
MAKEFUN(FUN,ARGS,BODY)::=
        BUILDQ([FUN,ARGS,BODY],
            FUN(SPLICE(ARGS)):=
              (MODE_DECLARE([SPLICE(ARGS)],FLOAT),
               BODY));
```

and then

```
MAKEFUN(G,[X,Y,Z],X*Y↑2+Z);
```

will produce

```
G(X,Y,Z):=(MODE_DECLARE([X,Y,Z],FLOAT),X*Y↑2+Z) .
```

Note that the function G now contains the declaration for X,Y, and Z because the macro allowed the form resulting from substitution of the un-evaluated arguments to be passed to the interpreter; but that typing

```
DEFINE(F(X,Y,Z),(MODE_DECLARE([X,Y,Z],FLOAT),X*Y↑2+Z));
```

results in the definition of F without the declaration because the second argument is evaluated. To cause the declaration to be present, one must type

```
DEFINE(F(X,Y,Z),'(MODE_DECLARE([X,Y,Z],FLOAT),X*Y↑2+Z));
```

where the single quote (') causes the quoted expression to evaluate to itself.

If one were going to define a great many functions with declarations, it would be more efficient to use the macro because the DEFINE method requires that the user type the declaration each time while macros require it but once.

The fundamental problem with DEFINE, however, is that the MACSYMA --> LISP Translator has a difficult time dealing with it, since the body of the function defined must be known at the time of translation for an effective analysis to be made. The exact body in this case is only known at the time the DEFINE is actually evaluated. Since macroexpansions are code to code transformations the translator is allowed to make them when it is analyzing code.

For interactive use, macros permit the user to construct a function definition from a variable list and a body. The following example macro will construct a function F with

variables VL from BODY by optimizing the BODY and optionally using a function S to map over the optimized form.

```
FLD(F,VL,BODY,[S]):::=BUILDQ([F,VL,BODY,S],
    BLOCK([OT,OT1,OT2],OT:OPTIMIZE(BODY),OT1:FIRST(OT),
          IF S = [] THEN OT2:FLOAT(REST(ARGS(OT)))
          ELSE IF LENGTH(S)=1 THEN
          OT2:FLOAT(MAP(FIRST(S),REST(ARGS(OT)))) ELSE
          ERROR("WRONG NUMBER OF ARGS TO
                FLD(F,VL,BODY,[S])"),
          EV(BUILDQ([OT,OT1,OT2],
          F(SPLICE(VL)):=(MODE_DECLARE([SPLICE(VL)],FLOAT),
          BLOCK([SPLICE(OT1)],
                MODE_DECLARE([SPLICE(OT1)],FLOAT),
                SPLICE(OT2)))),EVAL)));
```

Then `FLD(f,[x,y],diff(exp(x*y↑2)*y,x,2,y,1));` and

```
FLD(g,[x,y],diff(exp(x*y↑2)*y,x,2,y,1),
    lambda'[u],map(horner,factor(u))));
```

show how the functions are generated with the optional attempt to increase efficiency for g(x,y). This is similar to the FLOATDEFUNK function in MACSYMA, but has the optional argument S which can be used in efforts to reduce the computation time.

There is also a DEFM (for define macro) construct which uses its second argument as the macro definition of its first argument. An example is presented in Section 4 for a rectangle-rule numerical integrator.

3. THE MACSYMA --> LISP TRANSLATOR

The command TRANSLATE(f1, f2, ...) will cause the list of functions to be translated to LISP, and if the switch TRANSLATE (defaultly FALSE) is set to true then translation will occur automatically with each function definition. The result of translation is that the function becomes an EXPR and that future function calls will reference the LISP version. This is more efficient than interpreting a MACSYMA code for each function call, and generally results in a speed gain.

On MC, the TRANSLATE_FILE(fn1) command will cause the interpreted MACSYMA code in the file [fn1,>,dsk,direc] to be processed by a TRANSLATOR package that will generate a new file [fn1,trlisp,dsk,direc] containing LISP code. The trlisp file can then be processed for compilation to machine language as described in Section 4, or can be loaded into the MACSYMA environment via the LOAD command. The file [fn1,unlisp,dsk,direc] will have information and possibly warnings or error messages concerning the translation. On Multics, one must type at command level "asp translator >lib>macsyma>include" and in MACSYMA one must type "load(">udd>smi>a>e>troper");" in order to have the proper translation environment. Then TRANSLATE_FILE("fn.1") produces (on the working directory) the files "fn.trlisp" and "fn.unlisp".

The latter approach is part of the "preferred method" because it forces the user to prepare a file that is consistent and will allow the user to be warned about potential difficulties or inefficiencies of the code and have the warnings stored in a file.

Proper use of declarations is required to take advantage of speed gains in numerical work. Here is a list of rules to follow.

3.1 Use of MODE_DECLARE

```
MODE_DECLARE([...],FLOAT,
             [...],FIXNUM,
             [...],RATIONAL,
             [...],NUMBER,
             [...],LIST,
             [...],BOOLEAN,
             [...],ANY)
```

provides the declarations for each of the possible modes (unused modes can be omitted.)

FLOAT corresponds to REAL in FORTRAN, and is for single-precision flonums. FIXNUM is for integers, again as in FORTRAN. RATIONAL and NUMBER are ratios of integers and any number (except bigfloats), respectively. LIST is a MACSYMA list

[a, b, ...]. BOOLEAN is for TRUE/FALSE switches. ANY is for cases where the mode is general, or not known, the implied default.

The lists [...] contain all of the quantities to be of that mode. If only one argument, say x, is of a particular mode, then "[x]" can be replaced with "x". There may be arrays or functions appearing in these lists, as follows: ARRAY(X[M1,M2,...],Y[N1,N2,...],...) which specifies that the arrays X[M1,M2,...], Y[N1,N2,...], ... are of that mode; if the dimensions are not known then ARRAY(X,Y,...) will serve; and FUNCTION(F1,F2,...) which specifies that the functions F1, F2, ... return values of the particular mode.

The proper location for the MODE_DECLARE is at the beginning of each lexical contour, defined by: ":=" for the formal parameters and all free variables used in the function; "BLOCK" for the temporary variables in the BLOCK; "LAMBDA" for the LAMBDA variables; and "DO" for the "FOR" variable. The constructs "SUM" and "PRODUCT" determine the mode of the dummy variable by examining the modes of the upper and lower limits.

Here are some correct examples with all of the declarations in place:

```
F(X,Y):=(MODE_DECLARE(X,FLOAT,Y,FIXNUM),X↑Y);

G(X):=(MODE_DECLARE(X,FLOAT),
       BLOCK([T,W],
```

```
                MODE_DECLARE([T,W],FLOAT),
                T:X↑2. W:3.*X,
                EXP(T*W-SIN(W)))):

SUMAR(A):=(MODE_DECLARE(ARRAY(A),NUMBER),
           IF LENGTH(ARRAYDIMS(A))#1 THEN
             ERROR("NOT A 1-DIMENSIONAL ARRAY"),
           BLOCK([N,SUM],
                 MODE_DECLARE(N,FIXNUM,SUM,NUMBER),
                 N:ARRAYDIMS(A)[1], SUM:0,
                 FOR J:0 THRU N DO
                     (MODE_DECLARE(J,FIXNUM),
                      SUM:SUM+A[J]),
                 SUM));

ML(L):=(MODE_DECLARE(L,LIST),
        MAP(LAMBDA([U],MODE_DECLARE(U,BOOLEAN),
              IF U THEN FOO ELSE BAR),
            L));
```

Here is an example where the function parameter is not declared by the user. The Translator will print a warning that it has not been declared and that it will be taken as mode ANY:

```
F(X):=BLOCK([SUM],MODE_DECLARE(SUM,ANY),
            SUM:0,
            FOR I:1 THRU 5 DO (MODE_DECLARE(I,FIXNUM),
                               SUM:SUM+X↑I));
```

The quantity SUM has been properly declared inside the BLOCK, and I inside the DO.

3.2 Use of MODE_IDENTITY

MODE_IDENTITY(MODE,EXP) will return the evaluation of the expression EXP if it is of the type MODE, and otherwise will signal an error or warning.

Thus X:1.2; MODE_IDENTITY(FLOAT,X+1); returns 2.20000002, but MODE_IDENTITY(BOOLEAN,X); is an error.

The MODE_IDENTITY construct may be used to tell the Translator that a particular expression is of a certain mode in order that anything referring to that expression can have its mode determined. Thus the first element of a list L may be a FLOAT, and the user could type MODE_IDENTITY(FLOAT,FIRST(L)) whenever the first element of L is used as an argument to a function that expects a floating point argument. This is rather tedious if many such uses are required, and it would then be better to make an extra temporary variable or else use a macro; the latter is illustrated here.

```
MF(L)::=BUILDQ([L],MODE_IDENTITY(FLOAT,FIRST(L)));
```

and then MF(L); will return the floating point number which is the first element of L or will signal an error in the mode.

3.3 Use of DEFINE_VARIABLE

The use of DEFINE_VARIABLE is recommended for the introduction of global variables (those which are bound to a value outside of a function call). The syntax is

```
DEFINE_VARIABLE(VAR,VALUE,MODE,DOC_STRING);
```

where the variable VAR is initially bound to VALUE and is MODE_DECLAREd to be of type MODE, and there is an optional fourth argument, DOC_STRING, for documentation purposes. Here are some correct examples:

```
DEFINE_VARIABLE(X,1.,FLOAT);

DEFINE_VARIABLE(Y,2,NUMBER,"Y IS INITIALLY 2");

DEFINE_VARIABLE(SW,TRUE,BOOLEAN);

DEFINE_VARIABLE(R,'R,ANY,
                "R IS INITIALLY 'R BUT
                 MAY GET BOUND TO ANYTHING");
```

The purpose of using DEFINE_VARIABLE is that this will MODE_DECLARE VAR to be of type MODE, DECLARE VAR be a SPECIAL variable, bind it to VALUE if it is unbound, and put an assign property on it so that if an improper-mode binding is attempted an error will be signalled (e.g., SW:1; will cause an error after the third example).

3.4 Use of EVAL_WHEN

For MACSYMA files that are to TRANSLATE_FILEd, the EVAL_WHEN construct allows some control over cases of BATCH, DEMO, TRANSLATE, and LOAD. The user may need to make use of some special routines in a file during compilation, for example, and may thus do

```
EVAL_WHEN(TRANSLATE,LOAD(NEEDED FILE));
```

as the first line of his program. When TRANSLATE_FILE is run on the program, the needed file will be loaded into the environment. This is important for a routine such as the adaptive integrator, QUANC8, in SHARE1;QQ FASL:

```
EVAL_WHEN([TRANSLATE,BATCH,DEMO,LOAD],LOAD("QQ"));

F(X):=(MODE_DECLARE(X,FLOAT),
       QUANC8(LAMBDA([U],MODE_DECLARE(U,FLOAT),
                          EXP(U)/(U↑2+1.)),
              0.,X));
```

which calls the 3-argument version of QUANC8; it is important to have the QQ file loaded in order that the correct form of the macro is used during the translation/compilation phase, since a 4-argument version is also available and would leave an ambiguity if the file were not loaded.

Another important case is the declaration of arrays

```
EVAL_WHEN([BATCH,TRANSLATE,LOAD]),ARRAY([A,Y],FLOAT,99));
```

in order to have the compiler generate efficient code.

3.5 Pattern Matches

When translating pattern matches, use TRANSLATE_FILE on a file with first line

```
EVAL_WHEN([TRANSLATE,LOAD],TRANSCOMPILE:TRUE);
```

so that the Translator will generate the compiler declarations. MATCHDECLAREs must in effect during the translation analysis, i.e., they should be evaluated at TRANSLATE time.

3.6 Avoid EV

Do not have calls to EV implicitly or explicitly in code which is to be translated: use SUBST(1,X,EXP); instead of EV(EXP,X:1); in such cases. If you do need to use EV because there is no other way to get access to a certain feature then use the calling form APPLY('EV,[..arguments...]) which will at least be consistent in usage in compiled and interpreted code. This is also necessary for special functions which do not evaluate their arguments, such as GRAPH2, as in

```
APPLY('GRAPH2,[X,[Y1,Y2,Y3],[0,1,3]]);
```

which graphs the 3 y-lists vs. x with the indicated line-types.

4. COMPILATION

The Compiler can be invoked on MC by COMPILE(); which will prompt for further arguments, or by COMPILE_LISP_FILE("FN1 FN2"); which will compile the specified file of LISP code; the latter form can be used on the output of TRANSLATE_FILE. The LISP Compiler produces machine code and puts it in the file "FN1 FASL", so that LOAD("FN1"); will then load the compiled code into MACSYMA. The FASL file occupies a portion of memory which the LISP Garbage Collector will not disturb, and thus some time saving can be achieved by this fact alone. The MIT-MC KL10 does not permit FASL files to be unloaded from core, and thus the user must be careful to avoid running out of core on the limited-address space machine. The Compiler also produces an UNFASL file which is a record of the statistics associated with the functions compiled.

On Multics, it is necessary to add search paths and search rules in order to access the compiler as well: at command level one must type "asp exec_com >lib>macsyma>tools" and "asr >lib>macsyma>tools". To compile the file "foo.trlisp" produced by the translate_file("foo.1") command in MACSYMA, one must type at command level "macsyma_lcp -pn foo.trlisp", which will

produce the "foo.fasl" and "foo.unfasl" files. Then go back into MACSYMA and load the fasl file.

The use of the declarations in numerical functions causes an efficiency gain upon Translation/Compilation: when the mode of an argument is known to be FLOAT, the LISP code will use the flonum multiply command "*$" ; and upon compilation, the multiplicands will be assigned space in the FLONUM area, and the machine instructions for floating point multiply will be issued; if the mode were not known, then mode ANY would be used and then the "(MTIMES SIMP)" MACSYMA-generic-multiplication command would be generated by the translator, and the compiler would issue instructions for general symbolic multiplication, one of whose sub-cases is floating point multiplication, and place the variables in the general symbol area. The gain in proper declaration is that only the minimum amount of processing overhead is required in each function-call after compilation. For time-consuming compute-bound calculations, there can be very significant speed gains.

An example combining declarations and the use of macros is found in the file MC:SHARE;SIMPSN MACRO:

```
(LOAD_PACKAGE(SHAREM,"DSK:SHAREM\;AUTOLOAD FASL"),
 IF SHOWTIME=FALSE THEN SHOWTIME:'ALL)$

DEFM(RECTRULE('EXPRESSION,'VAR,'A,'B,'DVAR),
     BLOCK([%_SUM:0.0,%_A:FLOATCHECK(A),
            %_B:FLOATCHECK(B),%_DVAR:FLOATCHECK(DVAR)],
```

```
                MODE_DECLARE([%_SUM,%_A,%_B,%_DVAR],FLOAT),
                  FOR VAR:%_A THRU %_B STEP %_DVAR
                 DO %_SUM:%_SUM+EXPRESSION,
                %_SUM*%_DVAR));

MODE_DECLARE(FUNCTION(FLOAT_CHECK),FLOAT)$
FLOATCHECK(X):=
 (X:FLOAT(X),
  IF FLOATNUMP(X) THEN X ELSE ERROR("Not FLOATCHECK",X))$

F(P,N):=(MODE_DECLARE(P,FIXNUM),
         RECTRULE(X↑P,X,0,1,1/N))$

MACROEXPANSION:'DISPLACE$

/* If you want to experiment a little, use the
interpreted F, and try F(5,1000); It should take about
13 cpu seconds.  Then COMPILE(F); and try it.  It
should then take 0.053 cpu seconds, a speed up by a
factor of 245, i.e., the computation takes only 0.4%
as much time as it did before.  [I'm not sure I know
of any other compilers which give such a speed-up.] */
```

This combination of Macros and the MACSYMA --> LISP Translator / Compiler provides enormous versatility in the rapid computation of numerical results. The functions ROMBERG, PLOT2, INTERPOLATE, and QUANC8 have had special coding installed in them to handle the 3- and 4-argument cases by appropriate macros. Loading the source files for these during the

Translation/Compilation phase (by EVAL_WHEN([TRANSLATE,LOAD], LOAD("ROMBRG"), LOAD("APLOT2"), LOAD("INTPOL"), LOAD("QQ")); using only those which are needed) then enables the proper case to be compiled. The example F(X) defined in Section 3.4 takes some 580 msec to compute F(1.): interpreted; but after compilation, only 14 msec were required on the MIT-MC KL10, or 1/40 the time.

5. GENERAL DISCUSSION AND CONCLUSIONS

The use of Macros and the MACSYMA --> LISP Translator/Compiler are thus seen to be very valuable adjuncts to MACSYMA when used for numerical work. It is expected that the variety of data types present in MACSYMA will allow the user more freedom in the coding of algorithms than, say, FORTRAN, and at the same time there should not be any significant degradation of efficiency if the compiler is properly used.

Differential equations can take many special forms, with most of the coefficients of a general form dropped out (value zero) or unary in many examples. A problem with most simple numerical methods is that they do operations on these coefficient spaces rather blindly, needlessly doing extra work. This is why one sees many examples in the literature of programmers creating special case code for a given differential equation which becomes the object of extensive study. With the

MACSYMA symbolic environment, combined with macros and translation/compilation, one can write programs in the form of macros which automatically do these general to special case reductions, with increased confidence of reproducibility. Other uses of macros include the generation of code "to size", in which a macro is used to write a piece of code that can be written only after the user has supplied his input: a root-locus plotter can have the proper number of arrays to store the solutions generated at run-time when the number of solutions is determined from the equation, rather than being a parameter passed in by the user; and a Runge-Kutta solver for a system of differential equations which has arrays that are generated for each variable and its derivative after the input system has been analyzed by a macro, rather than passing in a parameter to describe the size of the system.

Perhaps the best part of this environment is that the user can compile "on the fly" and dynamically link the compiled code into the environment without having to go into an edit-compile-link-run loop, as in FORTRAN. This certainly allows very rapid access to plotting facilities even for somewhat computation-intensive calculations.

The "preferred method", though, is to write a BATCH'able file with EVAL_WHEN, DEFINE_VARIABLEs, functions with full and proper MODE_DECLAREs, and then use TRANSLATE_FILE. If the resulting UNLISP file indicates that there were no warnings or errors, then COMPILE_LISP_FILE or macsyma_lcp can be run on the

TRLISP file to produce a FASL file. The FASL file is then to be loaded in MACSYMA and the functions used. If errors are present, then some analysis and debugging should enable the user to correct the problems and get back on track.

Some large extensions of MACSYMA (e.g., DIAGEVAL by Richard Brenner and ODE by Ed Lafferty) have been TRANSLATED to LISP for the reduction in storage and for the gains in efficiency that can be achieved.

6. ACKNOWLEDGMENTS

The macro package was devised by Robert E. Handsaker and Kent M. Pitman. Early versions of the Translator were done by Michael R. Genesereth, John L. Kulp, Stavros Macrakis, Jeffrey P. Golden, Robert Handsaker, and George Carrette. One of the authors (LPH) would like to express his thanks to the many persons involved in teaching him LISP, and thus to appreciate macros, among them are: Kent M. Pitman; Robert W. Kerns; Robert E. Handsaker; Jeffrey P. Golden; William G. Dubuque; Jim O'Dell; and George Carrette.

Jim O'Dell is also to be thanked for his attention to the MIT-MULTICS implementation of these features of Macsyma.

7. REFERENCES

1. MACSYMA Reference Manual, The Mathlab Group, Laboratory for Computer Science, MIT, Version 9, 1977

2. Pitman, K. M. The Revised MACLISP Manual, MIT/LCS/TR-295, Laboratory for Computer Science, MIT, 1983

Third MACSYMA Users' Conference
July 23-24, 1984

LIST OF PARTICIPANTS

Agnarsson, Snorri, RPI, Troy, NY 12181
Akman, Varol, Elec., Computer & Systems Eng. Dept., RPI, Troy, NY 12181
Andersen, Carl, Department of Math, College of William & Mary, Williamsburg, VA 2318
Andrews, George E., Math Department, Penn State University, University Park, PA 1680
Anker, David S., US Army Night Vision & Electro-Optics Lab., Ft. Belvoir, VA 22309
Arnon, Dennis S., Computer Science Dept., Purdue University, West Lafayette, IN 47907

Babson, William, Naval Underwater Systems Center, Willimantic, CT 06226
Baker, Johnnie, Kent State University, 5525 Allyn Road, Mantua, OH 44255
Bannister, Kenneth, Naval Surface Weapons Center, Code R14, Silver Spring, MD 20910
Barber, William D., CRD, General Electric Company, Schenectady, NY 12301
Baruch, Marjory, Hamilton College, 106 Cammot Lane, Fayetteville, NY 13066
Baum, Alan, GM Research Labs., Computer Science Dept., Warren, MI 48090
Beauchamp, Philip, AEBG, GE Co., 1000 Western Avenue, Lynn, MA 01910
Bendler, John, CRD, General Electric Company, Schenectady, NY 12301
Berman, Robert, MIT, 38-191, 77 Massachusetts Avenue, Cambridge, MA 02139
Beyer, William, Los Alamos National Laboratory, Los Alamos, NM 87544
Brennan, Mary Ellen, The Aerospace Corporation, PO Box 92957, Los Angeles, CA 9000
Brenner, Richard, Symbolics Inc., 11 Cambridge Center, Cambridge, MA 02142

Carrette, George J., Paradigm Associates, Inc., 29 Putnam Ave., Suite 6, Cambridge,
Caviness, B. F., Dept. of Computer & Info Sci., U. of Delaware, Newark, DE 19716
Chandra, Jagdish, US Army Research Office, PO Box 12211, Research Triangle Park, N
27709
Char, Bruce W., Dept. of Computer Science, U. of Waterloo, Waterloo, Ont., Canada
N2L 3G1
Cheung, Maria H. Lam, Hampton Institute, PO Box 6622, Hampton, VA 23668
Chriss, Chuck, Inference Corp., 6 Landmark Square, 4th fl., Stamford, CT 06901
Clarkson, Michael, York University, CRESS, 4700 Keele St., Downsview, Ont., Canada
M3J 1P3
Coar, David, Floating Point Systems, PO Box 23489, Portland, OR 97233
Cohen, H. Ian, School of Mathematics, U. of Bath, Claverton Down, Bath BA2 7AY, Engl
Cook, Grant, Lawrence Livermore Labs., Livermore, CA
Cooper, A. Brinton, US Army Ballistic Research Lab., Attn: DRXBR-SECAD,
Aberdeen Proving Ground, MD 21005
Cooperman, Gene, GTE Laboratories, Inc., 40 Sylvan Road, Waltham, MA 02254
Cuthill, Barbara, Brown University, Providence, RI 02912
Cuthill, Elizabeth, David Taylor Naval Ship R&D Center, Bethesda, MD 20084
Cuthill, John, 12700 River Road, Potomac, MD 20854

Author Index

Daripa, Prabir, Brown University, Box F, Providence, RI 02912
DeLoath, Sandra J., Norfolk State University, 2401 Corprew Avenue, Norfolk, VA 235C
de Lorenzi, Horst, CRD, General Electric Company, Schenectady, NY 12301
Drinkard, R. Drew Jr., Naval Underwater Systems Center, 7 Gallup Lane, Waterford, CT 06385
Dubuque, William, Symbolics Inc., 11 Cambridge Center, Cambridge, MA 02142
Dyer, Charles C., Dept. of Astronomy, U. of Toronto, 1265 Military Trail, Scarborough, Ontario, Canada M1C 1A4

Engelman, Paul, US Air Force TPSC/TPJS, RM1D1039, The Pentagon, Washington, DC 20330
Eskin, Howard, CRD, General Electric Company, Schenectady, NY 12301

Fateman, Richard J., Computer Science Div., U. of California, EECS, 519 Evans Hall, Berkeley, CA 94720
Fee, Greg, U. of Waterloo, Maple Lab., Waterloo, Ontario, Canada N2L 3G1
Fitch, John, University of Bath, Claverton Down, Bath, England BA2 7AY
Fong, Jefferson, Brown University, Box F, Providence, RI 02912
Forth, Catherine, CRD, General Electric Company, Schenectady, NY 12301
Frawley, William, GTE Laboratories, Inc., 40 Sylvan Road, Waltham, MA 02254

Garcia, A., Emerson Electric Company, 8100 W. Florissant, MS 4333, St. Louis, MO 63136
Gardner, Russell, Symbolics, 11 Cambridge Center, Cambridge, MA 02142
Gates, Barbara, Dept. of Mathematical Sciences, Kent State University, Kent, OH 44242
Geddes, K. O., U. of Waterloo, Dept. of Comp. Sci., Waterloo, Ontario, Canada N2L 3G1
Giorgi, Vincent, Raytheon Company, Hartwell Rd., MS M8-12, Bedford, MA 01730
Gladd, N. Thomas, JAYCOR, PO Box 85154, San Diego, CA 92138
Golden, Jeffrey, GTE, 30 St. Paul Street - #4, Brookline, MA 02146
Golden, V. Ellen, Symbolics, Inc., 30 St. Paul St. - #4, Brookline, MA 02146
Gross, Dan, CRD, General Electric Company, Schenectady, NY 12301
Gupta, Dinesh, AID, MS-39, General Electric Co., 50 Fordham Rd., Wilmington, MA 0188
Gutowitz, Howard, Rockefeller University, 1230 York Ave., Box 179, New York, NY 10021

Hackert, Frank P., AEBG, GE Co., MD 2V310, 1000 Western Avenue, Lynn, MA 01910
Harper, John F., Dept. of Astronomy, U. of Toronto, 1265 Military Trail, Scarborough, Ontario, Canada M1C 1A4
Harten, Leo, Paradigm Associates, Inc., 20 Putnam Ave., Suite 6, Cambridge MA 02139
Hasegawa, J., Nichimen Corporation, Tokyo, Japan
Hellig, William B., CRD, General Electric Company, Schenectady, NY 12301
Hsu, Bowwei Su, CRD, General Electric Company, Schenectady, NY 12301
Hillis, Patrick, Cornell University, Box 32 Upson Hall, Ithaca, NY 16853
Horowitz, Henry, CRD, General Electric Company, Schenectady, NY 12301
Hossain, M. A., CRD, General Electric Company, Schenectady, NY 12301
Hotum, Kjeld, The Charles Stark Draper Laboratory, Inc., MS-96, 555 Technology Square, Cambridge, MA 02139

Inselmann, Edmund, Department of the Army, Combined Arms Operations Research Activity, Attn: ATOR-CAD, Ft. Leavenworth, KA 66027

Jenks, Richard D., Mathematical Sciences Dept., IBM Research Center, Yorktown Heights, NY 10598

Kaltofen, Erich, Dept. Computer Science, RPI, Troy, NY 12181
Kelly, Christine, CRD, General Electric Company, Schenectady, NY 12301
Klaus, John F., Electronic Associates, Inc., 185 Monmouth Parkway, West Long Branch, NJ 07764
Kolodziejczyk, Richard, Ordnance Systems, GE Co., 100 Plastics Ave., Pittsfield, MA
Konopasek Milos, Software Arts, 27 Mica Lane, Wellsley, MA 02181
Kwok, Y. K. Brown University, Box F, Providence, RI 02912

Lamson, Scott, CRD, General Electric Company, Schenectady, NY 12301
Lewis, John W., Martin Marietta Labs., 1450 S. Rolling Rd., Baltimore, MD 21227
Lewis, Philip, CRD, General Electric Company, Schenectady, NY 12301
Lewis, Timothy, Eastman Kodak Co., 901 Elmgrove Road, Rochester, NY 14650
Lim, Chjan, Brown University, Box F, Providence, RI 02912

Mackeonis, Peter C. Metacom Company, 26 Portland Square, Bristol, England
Makuch, William, CRD, General Electric Company, Schenectady, NY 12301
Manning, Ed, Pyramid Technology Corporation, 155 New Boston St., Suite 180, Woburn, MA 01801
Massell, Paul, Mathematics Department, US Naval Academy, Annapolis, MD 21402
Mawson, J. Bruce, Electronic Associates, Inc., 185 Monmouth Parkway, West Long Branch, NJ 07764
McCalley, Robert B., MAO, General Electric Co., PO Box 1021, Schenectady, NY 12301
McIssac, Kevin, U. of Western Australia, Crawly, W. Australia
Meenan, Peter, CRD, General Electric Company, Schenectady, NY 12301
Mejia, Raymond, National Institutes of Health, 9000 Rockville Pike, Bldg. 31, Rm. 4B44, Bethesda, MD 20205
Milligan, Vincent, Benet Weapons Lab., LCWSL, US Army Armament R&D Center, Watervliet Arsenal, Watervliet, NY 12189
Monogan, Michael B., U. of Waterloo, Dept. of Computer Science, Waterloo, Ontario, Canada N2L 3GL
Moses, Joel, MIT, Bldg. 38-403, Cambridge, MA 02139
Moskowitz, Jacob, CBS Labs, 227 High Ridge Road, Stamford, CT 06905
Mueller, Otward, CRD, General Electric Company, Schenectady, NY 12301

Narendran, Paliath, CRD, General Electric Company, Schenectady, NY 12301
Nielan, Paul, Sandia National Labs, Division 8121, Livermore, CA 94550
Noble, Ben, Math Research Center, U. of Wisconsin, 610 Walnut Street, Madison, WI 537
Noyes, Terri Alice, CRD, General Electric Company, Schenectady, NY 12301

Ocken, Stanley, Dept. of Math(NAC), City College of CUNY, 100-38th & Convent Ave., New York, NY 10031
Oliveira, Janet B. Jones, Draper Labs/MIT, 154 Magazine St., Apt. 32, Cambridge, MA 02139
Oliveira, Joseph, Rm. 33-407, Space Systems Lab., Dept. of A&A, MIT, Cambridge, M

Pavelle, Richard, Symbolics, 11 Cambridge Center, Cambridge, MA 02142
Piquette, Jean C., Naval Research Laboratory, Orlando, FL 32856
Pflegl, G. Albert, Benet Weapons Laboratory, LCWSL, US Army Armament R&D Center, Watervliet Arsenal, Watervliet, NY 12189
Plonski, Joseph, Symbolics, Inc., 4 Cambridge Center-7th fl., Cambridge, MA 02142
Pu, San Li, Benet Weapons Laboratory, LCWSL, US Army Armament R&D Center, Watervliet Arsenal, Watervliet, NY 12189
Putnick, Leonard J., Siena College, 1 Simon Lane, Latham, NY 12110
Rietsch, Eike, Texaco, Bellaire Research Labs., P.O. Box 425, Bellaire, TX 77401
Rocha, Henry, CRD, General Electric Company, Schenectady, NY 12301
Roeder, R. C., Southwestern University, Department of Physics, Georgetown, TX 78626
Rosencrans, Steven, Tulane U., New Orleans, LA 70118
Rothschild, Susan, KAPL, General Electric Company, PO Box 1072, Schenectady, NY 1230
Rothstein, Michael, Kent State University & IBM, Kent OH 44242
Saunders, B. David, RPI, Computer Science Department, Troy, NY 12181
Sayegh, Samir D., AEBG, General Electric Company, MD 153TL, 1000 Western Avenue, Lynn, MA 01910
Schenck, John F., CRD, General Electric Company, Schenectady, NY 12301
Schroeder, William, Gas Turbine Division, Bldg. 53, Rm. 332, GE Co., Schenectady, NY
Schumacher, Daniel, CRD, General Electric Company, Schenectady, NY 12301
Shah, Shantilal, Hampton Institute, Dept. of Math & Computer Sci., Hampton, VA 23668
Shen, C.N., Benet Weapons Laboratory, LCWSL, US Army Armament R&D Center, Watervliet Arsenal, Watervliet, NY 12189
Shuey, Ralph T., Gulf Research & Development Co., PO Box 37048, Houston, TX 77237
Shyy, Wei, CRD, General Electric Company, Schenectady, NY 12301
Simkins, Thomas, Benet Weapons Laboratory, LCWSL, US Army Armament R&D Center, Watervliet Arsenal, Watervliet, NY 12189
Singer, Michael, North Carolina State University, Raleigh, NC 27650
Strovich, Lawrence, Brown University, 37 Riverside Drive, New York, NY 10023
Slotterbeck, Oberta, Hiram College, 5525 Allyn Road, Mantua, OH 44255
Soanes, Royce, Jr., Benet Weapons Laboratory, LCWSL, US Army Armament R&D Center, Watervliet Arsenal, Watervliet, NY 12189
Stackelberg, Olaf, Kent State University, Dept. of Mathematical Sciences, Kent, OH 44242
Stanat, Donald, University of North Carolina @ Chapel Hill, Chapel Hill, NC
Steinberg, Stanly, U. of New Mexico, Department of Math. & Stat., Albuquerque, NM 8713
Stevens, Donald, New York University, 251 Mercer Street, New York, NY 10012
Storch, Joel, Draper Laboratory, 555 Technology Square, Cambridge, MA 02139
Stoutemyer, David R., University of Hawaii, Honolulu, Hawaii
Suarez, Mel, AGORA Systems, Inc., 156 Sixth Street, Cambridge, MA 02142
Tan, Hui-Qian, Kent State University, 6600 Alpha Drive, #116, Kent, OH 44240
Taylor, Dean, Cornell University, Upson 210, Ithaca, NY 14853
Tong, Siu Shing, CRD, General Electric Company, Schenectady, NY 12301
Topka, Terry M., CRD, General Electric Company, Schenectady, NY 12301
Turner, Mark, AEBG, General Electric Company, MD 2V310, 1000 Western Avenue, Lynn, MA 01910

Van Patten, Charles, PO Box 164, Schenectady, NY 12301
Vasilakis, John D., Benet Weapons Laboratory, LCWSL, US Army Armament R&D
Watervliet Arsenal, Watervliet, NY 12189

Wang, Hsin-Pang, CRD, General Electric Company, Schenectady, NY 12345
Wang, Paul S., Kent State University, Dept. Mathematical Sciences, Kent, OH 442
Watt, Steven, University of Waterloo, Dept. of Computer Science, Waterloo, Ontar
Canada N2L
Wester, Michael, U. of New Mexico, 1801 Quincy, S.E., Albuquerque, NM 87108
Wheeler, LaNae Estelle, Hampton Institute, PO Box 6622, Hampton, VA 23668
Whitaker, Rick, McDonnell Douglas Automation Company, PO Box 516 K211,
Bldg. 302/2 East, St. Louis, MO 63166
White, Gerald, CRD, General Electric Company, Schenectady, NY 12301
Wiggs, John F., Hampton Institute, 2707 Shell Road, Hampton, VA 23661
Wilcox, Ralph, Hughes Aircraft Co., PO Box 902, El Segundo, CA 90245
Williams, Vernon L., Westinghouse Electric Corporation, Space Div., MS 373,
Baltimore, M
Wolff, Stephen, US Army, Ballistic Research Laboratory, Aberdeen Proving Grour

Wolfram, Stephen, The Institute for Advanced Study, Princeton, NJ 08540
Wood, David, Code 3332, New London Laboratory, Naval Underwater Systems Cen
New London, CT 0632
Wu, Julian, US Army Research Office, PO Box 12211, Research Triangle Park, N

Yagla, Jon, Naval Surface Weapons Center, Dahlgren, VA 22448
Yeung, Daniel, CRD, General Electric Company, Schenectady, NY 12301
Yui, Noriko, Dept. of Mathematics, U. of Toronto, Toronto, Ontario, Canada, M5

Zwillinger, Daniel, Dept. of Mathematical Sciences, RPI, Troy, NY 12181

END

FILMED

11-84

DTIC

LEVY (STABLE) PROBABILITY DENSITIES AND RELAXATION IN SOLID POLYMERS

John T. Bendler
Polymer Physics and Engineering Branch
General Electric Corporate Research and Development
Schenectady, NY 12301

ABSTRACT

Non-integer power-laws now appear to be common in transport properties of disordered glassy polymers, and are interpreted as evidence for dynamical (fractal) scaling and modeled using momentless stable probability densities exhibiting hierarchical clustering on all time scales. Closed form expressions for stable densities and related quantities are not known in general, and some algorithms are found and studied using MACSYMA tools.

Viscoelastic responses in solid polymers are strongly characterized by hereditary "after-effects" and non-linearity, and so one does not expect that the theory of Markov processes will be useful in describing or understanding them. Nevertheless, a certain class of relaxation properties of polymers and glasses may now be modeled using a continuous- time random-walk (CTRW) formalism. A basic equation for the analysis of stationary stochastic processes is Chapman-Kolmogorovs';

$$P(y_1,y_2;t) = \int P(y_2,y;t_1)P(y,y_1;t-t_1)dy \qquad (1)$$

where $P(y_1,y_2;t)$ is the probability density that the variaole y suffers a transition from y_1 to y_2 in time t. If the process y has translational invariance and the unbounded range $(-\infty < y < \infty)$, equation (1) becomes

$$P(y_1-y_2;t) = \int_{-\infty}^{\infty} P(y_2-y;t_1)P(y-y_1;t-t_1)dy \qquad (2)$$

which is reduced by introduction of the Fourier transform (FT)

$$p(k,t) = \int_{-\infty}^{\infty} P(y;t)e^{iky}dy \qquad (3)$$

to the algebraic form

$$p(k,t) = p(k,t - t_1)\ p(k,t_1) \qquad (4)$$

The functions e^{-Dtk^2} and e^{-akt} satisfy equation 4 and are the FTs of the Gauss and Cauchy densities respectively. The more general case was investigated by Paul Levy (1);

$$p(k,t) = e^{-btk^{\alpha}} \quad 0 < \alpha < 2 \qquad (5)$$

If b is real and postive, then p(k,t) in equation 5 is related by Fourier inversion to the symmetric stable density of characteristic exponent α . An elementry change of variable allows the latter to be written;

$$Q_{\alpha}(z) = \frac{1}{\pi}\int_0^{\infty} e^{-u^{\alpha}} \cos zu\ du \qquad (6)$$

Closed-form expressions for $Q_{\alpha}(z)$ exist only for $\alpha = 2$, 1 and $\frac{1}{2}$. Empirically, it is found to successfully fit dielectric, mechanical and magnetic relaxation in polymers with $0.3 < \alpha < 0.8$ near the glass transition, with much smaller exponents in the glassy solid itself. Dielectric and mechanical dispersion may be fitted using the sine transform, and transient stress,