3-D PRINTING EXTRUSION RATES THROUGH A TAPERED NOZZLE

by

Jacob W. Sitison

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Bachelor of Science in Applied Mathematics with Distinction

Spring 2020

3-D PRINTING EXTRUSION RATES

THROUGH A TAPERED NOZZLE

by

Jacob W. Sitison

I certify that I have read this thesis and that in my opinion it meets the academic and professional standard required by the University as a thesis for the degree of Bachelor of Science.

Signed: _____

David A. Edwards, Ph.D.
Professor in charge of thesis

Approved: _____

Michael Mackay, Ph.D.
Committee member from the Department of Materials Science and Engineering and Department of Chemical and Biomolecular Engineering

Approved: _____

Sebastian Cioabă, Ph.D.
Committee member from the Board of Senior Thesis Readers

Approved: _____

Michael Chajes, Ph.D.
Chair of the University Committee on Student and Faculty Honors

ACKNOWLEDGMENTS

First and foremost, I would like to thank my thesis advisor David Edwards for the opportunity to work on this project and his continued guidance throughout. I would also like to thank Michael Mackay for being a member of my thesis committee and his research group for providing the experimental data used in this study. Furthermore, I would like to thank the last member of my thesis committee Sebastian Cioabă for his feedback on my work.

I would like to thank the Undergraduate Research Program at the University of Delaware for giving me the opportunity to write this thesis and for their financial support during the 2019 Summer Scholars Program where this project began.

I would like to thank all of the University of Delaware professors from the Departments of Mathematical Sciences, Chemical and Biomolecular Engineering, and others who have contributed to my undergraduate education and prepared me for this project.

Lastly, I would like to thank my parents Margaret and Andy Sitison and the rest of my family and friends for their support throughout my academic career.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

In applications of 3-D printing, production rates and product quality are enhanced by increased printing speeds. A polymer feedstock is fed through the hot end of the 3-D printer, which operates at a set temperature. Since some amount of heating time is necessary for the polymer to become pliant, there is an upper bound on the flow velocity before it remains too rigid to be extruded. The hot end is comprised of a cylinder that feeds directly into a tapered nozzle immediately prior to extrusion. In this study, we model the effects of this geometry in both amorphous and crystalline polymers. We consider the former case, a heat transfer problem, in an idealized tapered hot end (without cylindrical portion) using separation of variables to provide an analytical temperature profile. We consider the latter case, a Stefan (moving boundary) problem, in three geometries (a cylinder, a taper, and a combined system) using the heat balance integral method to provide an analytical approximation for the temperature profile. We develop several different conditions based on these temperature profiles to predict maximum velocity. In amorphous polymers, the model fails to predict the experimental data due to limitations from the considered geometry. In crystalline polymers, using the exit temperature of the hot end yields a model that adheres well to the experimental data regardless of the geometry considered.

NOMENCLATURE

When appropriate, units are listed in terms of length (L), mass (M), time (τ), and temperature (T). If a symbol appears both with and without tildes, the symbol with tildes has units whereas the one without is dimensionless. The location where a notation is first introduced is also listed.

Variables and Parameters

A: area of heat transfer through the melt front, units L^2, (4.3).

a: dimensionless constant used in crystalline model, (4.14).

$B(z)$: dimensionless piecewise constant function used to describe the surface radius in the combined hot end case, (4.66).

C: coefficient for general solution of an ODE in the amorphous case, (3.9).

c: a negative proportionality constant, (4.58).

c_L: specific latent heat of melting, units L^2/τ^2, (4.3).

c_P: specific heat capacity of polymer, units $L^2/(\tau^2 T)$, (2.1).

D: coefficient for the series solution to the heat equation used in the amorphous model, (3.15).

$f(z)$: function used to discuss small-Pe asymptotics in the crystalline case, (4.34).

$g(z)$: function used to discuss small-α asymptotics in the crystalline case, (4.38).

$h(z)$: function used to discuss large-σ asymptotics in the crystalline case, (4.43).

H: vertical length of portion of hot end, units L, Assumption 2.

k: thermal conductivity, units $ML/(\tau^3 T)$, (2.1).

ℓ: integer used to discuss numerics, §4.5.3.

$M(a,b,x)$: confluent hypergeometric functions of the first kind, (3.9).

m: indexing variable, (3.17);
scaling exponent, (4.34).

N: index of largest necessary eigenvalue of the Sturm-Liouville problem in the amorphous case, (3.25).

n: indexing variable, (3.12);
scaling exponent, (4.38).

P: dummy variable, (4.27a).

Pe: Péclet number, (2.6).

Q: volumetric flow rate, units L^3/τ, (2.2).

$q(\tilde{r})$: heat flux, units M/τ^3, (4.3).

$\tilde{R}(\tilde{z})$: radius of outer surface of hot end, units L, Assumption 2.

$\tilde{r}$: radial coordinate, units L, (2.1).

St: Stefan number, (4.7).

$\tilde{s}(\tilde{z})$: melt front radius, units L, §2.

$\tilde{T}(\tilde{r},\tilde{z})$: temperature, units T, (2.1).

$\tilde{t}$: temporal coordinate, units τ, (2.1).

$U(a,b,x)$: confluent hypergeometric functions of the second kind, (3.9).

$V(\tilde{z})$: vertical (flow) velocity, units L/τ, Assumption 4.

$w(z)$: logarithm of the normalized melt front radius, (4.42a).

$X(x)$: function of radial coordinate used in the amorphous case, (3.7).

x: scaled square of normalized radial coordinate, (3.5).

$Y(y)$: function of radial coordinate used in the amorphous case, (3.3).

y: normalized radial coordinate, (2.10).

$Z(z)$: function of vertical coordinate used in the amorphous case, (3.3).

$\tilde{z}$: vertical coordinate, units L, (2.1).

α: dimensionless temperature at heater, (2.8b).

β: normalized nozzle exit radius, (2.5).

γ: decay constant for the approximate surface radius function used in the amorphous case, (3.4).

ΔT: differential between transition and room temperature, units T, (2.4).

Δz: step in vertical coordinate used to discuss numerics, §4.5.

$\Delta\sigma$: step in normalized melt front radius used to discuss numerics, §4.5.

ζ: dummy variable, (3.36).

$\eta(z)$: function containing information on the dependence of the melt front on the surface radius, (4.25b).

$\Theta(y, z)$: heat function used in amorphous case, (3.1).

λ: eigenvalue of the Sturm-Liouville problem in the amorphous case, (3.7a).

μ: some real number, (3.5d).

ν: some real number, (3.7c).

ϵ: normalized radial position used in the exit temperature condition for the crystalline model, (4.54).

ε: aspect ratio of cylindrical portion of the hot end, Assumption 2.

ξ: dummy variable, (3.41).

ρ: polymer density, units M/L^2, (2.1).

$\sigma(z)$: normalized melt front radius, (4.9).

ς: dummy variable, (4.27a).

ϕ: angular coordinate, (2.1).

φ: dummy function, (4.2).

dummy variable, (4.35a).

χ: variable used in the exit temperature condition for the crystalline model, (4.55).

$\psi(z)$: function defined to save computational expense in the crystalline model, (4.47).

ω: dummy variable, (4.46a).

Other Notation

0:	as subscript on w, z, σ, and ψ, denotes an initial condition for an asymptotic solution for the melt front developed for the crystalline model, (4.27).
cyl:	as subscript on H, denotes vertical length of the cylindrical portion of the hot end, Assumption 2.
end:	as subscript on z, denotes the vertical position of the exit of the hot end, (2.5).
g:	as subscript on T, denotes the the glass-rubber transition temperature, §3.1.
i:	as subscript on T, denotes the initial temperature, (2.4).
m:	as subscript on T, denotes the melting temperature, §4.1.
max:	as subscript on R, denotes the maximum of R over the vertical length of the hot end, Assumption 2; as subscript on T, denotes (dimensional) temperature of the heater, (2.8b).
min:	as subscript on R and V, denotes the minimum of a function over the vertical length of the hot end (2.5).
noz:	as subscript on H, denotes vertical length of the tapered portion of the hot end (the nozzle), (2.5).
p:	as subscript on q, T, and φ, denotes property in pliant region, (2.8b).
r:	as subscript on q, T, and φ, denotes property in rigid region, (2.8a).
$[\cdot]_{\tilde{s}}$:	denotes the magnitude in the discontinuity of $\cdot$ at the melt front, (4.2).
t:	as subscript on T, denotes the extrusion threshold temperature, (3.27).
$*$:	as subscript on T, denotes the pliancy transition temperature, (2.4).
$\langle\cdot\rangle$:	denotes the cross-sectional average of $\cdot$, (3.27).
$\bar{\cdot}$:	denotes the cumulative cross-sectional average of $\cdot$, (3.34).

$+$: as subscript on s, denotes the limit as r approaches s from above, (4.2).

$-$: as subscript on s, denotes the limit as r approaches s from below, (4.2).

Chapter 1

INTRODUCTION

1.1 Motivations

Additive manufacturing (AM) is an area of increasing academic interest in recent decades. The development of these technologies has been motivated by the increasing scope of their practical applications in fields such as aerospace, automotive design, biomedical engineering, energy technologies, and rapid prototyping [1]. There are various types of AM such as powder bed fusion, where fine particles are fused together layer-by-layer, and polymerization, where monomers are bound forming polymer chains of a desired structure. The type of AM of interest in this study is fused deposition modeling (FDM) (also known as 3-D printing) where a nozzle deposits layers of molten material onto a substrate.

In applications of 3-D printing, the extrusion rate is a critical factor affecting the quality of manufactured products. Typically, higher printing speeds are desired because faster printing means that the temperature of deposited polymer layers will have been reduced less by ambient conditions when it comes time to deposit the next layer. This increased sublayer temperature helps facilitate binding to extruded polymer and thus improves the quality and durability of products [2, 3, 4]. Quicker printing also reduces processing time [5, 6]. There are several factors that limit the maximum extrusion velocity. The extruded polymer is heated as it travels through the hot end of the 3-D printer extruder (Figure 1.1). Therefore, if the velocity of the polymer is increased, then the heating time, and thus the total heat load delivered to the polymer, is decreased. This can result in the polymer being too rigid to be extruded through the tapered nozzle of the hot end by the insertion pump. There are other problems

associated with exceeding a maximum extrusion rate, such as the development of what is known as "shark skin" where the surfaces of deposited layers are distorted by elastic surface instability [7]. This study will primarily focus on insertion pump failure due to excess rigidity in the flowing polymer.

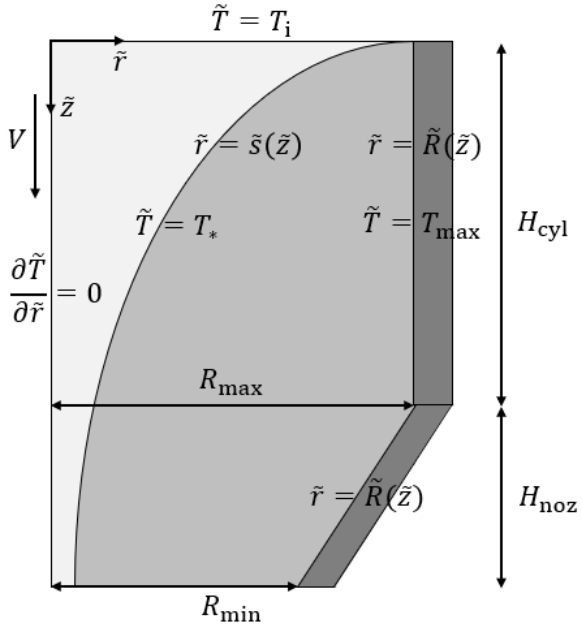


Figure 1.1: Cross-section of half of hot end in dimensional coordinates (not to scale). Light grey is rigid (glass or crystalline) polymer, medium grey is pliant (rubber or melted) polymer, and dark grey is the heater.

A recent article [8] developed a relationship between the failure (maximum) velocity and hot end temperature for acrylonitrile butadiene styrene (ABS), poly(lactic acid) (PLA), and poly(lactic acid)-polyhydroxybutyrate copolymer (PLAPHA). In this study, a master curve was developed that can be used to find the failure velocity for a particular operating temperature, or vice versa, given a particular set of polymer properties and hot end specifications. Another article [5] gave a more rigorous description of the mathematical development of this master curve. The model developed was shown to agree with experimental data; however, some aspects of the model require further investigation. First, the model bases failure velocity on temperature averages in the hot

end. The motivation for these criteria is that below a certain temperature the polymer viscosity will be too low for extrusion; however, attempts to model failure velocity on a viscosity-based model showed less agreement with experimental data [5]. Second, the model ignores the narrowing of the hot end at the nozzle and merely bases failure conditions on the temperature before the inlet of the nozzle. Third, when considering the effects of phase changes in crystalline polymers (a Stefan problem), agreement with experimental data was diminished compared to applying the amorphous model to these polymers [5]. The amorphous model is designed as a heat transfer problem that neglects any potential effects of phase changes in the polymer during heating and thus should be applicable to ABS (an amorphous polymer) but not to PLA or PLAPHA (crystalline polymers). The focus of this study will be to develop a model that addresses the latter two concerns.

1.2 Objectives and Overview

We seek to develop models that are not only useful in engineering applications but also give us a better theoretical understanding of the relevant physical phenomena than current models. This goal entails the development of approximate analytical solutions to simplified problems rather than sole reliance on numerical solutions. After characterizing the temperature profile in each case, we compared different fitting conditions to experimental data to assess the validity of each model. The conditions considered are all based on the polymer temperature in part or all of the hot end. Temperature was chosen as a more readily calculable alternative to viscosity, the material property that is expected to more directly affect the polymer pressure (the state function that causes insertion pump failure). Since the models considered are correlative, the physical dependence of viscosity on temperature is sufficient to justify this choice. All computation was conducted in MATLAB®.

In §2, we describe the physical system in mathematical terms and apply a series of reasonable assumptions that give way to approximate analytical solutions to the principal governing equations.

In §3, we consider the flow of amorphous polymers through a tapered hot end. This is a heat transfer problem as amorphous polymers do not exhibit a phase transition during extrusion. We began by simplifying the system using a series of geometric transformations, which allows for the characterization of the temperature profile using a standard separation of variables method.

In §4, we consider the flow of crystalline polymers through a cylindrical, tapered, and combined hot end. This is a type of free boundary problem known as a Stefan problem since crystalline polymers do exhibit a phase transition during extrusion. After again utilizing geometric transformations to simplify the problem, we characterize the temperature profile using the heat balance integral (HBI) method.

This study shows that sole consideration of the tapered portion of the hot end is not sufficient to estimate a maximum extrusion velocity from the developed threshold conditions in amorphous polymers. For crystalline polymers, consideration of either the cylindrical portion of the hot end, the tapered portion of the hot end, or both produces an empirically accurate model for the maximum extrusion velocity based on the exit temperature. These results should help engineers to understand the mechanisms of 3-D printing and optimize printing processes to manufacture better products more quickly.

Chapter 2

GOVERNING EQUATIONS

The physical model of interest is similar to those of [5, 6] except with a surface radius that is a function of the vertical space coordinate (Figure 1.1). We consider the heat transfer through an initially rigid polymer as it flows through an axisymmetric hot end as governed by the heat equation:

$$\rho c_P \frac{\partial \tilde{T}}{\partial \tilde{t}} = k \left[\frac{1}{\tilde{r}} \frac{\partial}{\partial \tilde{r}} \left(\tilde{r} \frac{\partial \tilde{T}}{\partial \tilde{r}} \right) + \frac{1}{\tilde{r}^2} \frac{\partial^2 \tilde{T}}{\partial \phi^2} + \frac{\partial^2 \tilde{T}}{\partial \tilde{z}^2} \right], \tag{2.1}$$

where ρ is density, c_P is specific heat capacity, k is thermal conductivity, T is temperature, t is time, r is the radial coordinate, ϕ is the angular coordinate, z is the vertical coordinate, and tildes denote dimensional quantities.

A full mathematical description of this system would be quite complex and necessitate numerical solutions [9, 10, 11, 12]. This moves against our goal of a physically illuminating analytical model. Fortunately, we are not interested in a complete model but merely a model that predicts the maximum possible extrusion velocity. This simpler goal allows us to apply a number of simplifying assumptions, namely the following:

1. Due to the axisymmetry of the problem, we neglect thermal diffusion in the angular direction.

2. Due to the small aspect ratio $\varepsilon = R_{\max}/H_{\text{cyl}}$ of the cylindrical portion of the hot end (and thus the entire hot end), we neglect thermal diffusion in the vertical direction [5, 6].

3. We are interested in the stationary problem and thus consider the steady-state flow developed after all transients have decayed away.

4. We assume that the flow velocity V varies only in the $\tilde{z}$-direction (See Figure 1.1). This is a realistic assumption near the inlet of the hot end because the inserted polymer fiber is slightly smaller than the tube, and hence will slide easily inside it. Once the polymer becomes pliant and liquid-like, it will develop into a Poiseuille or non-Newtonian flow profile. We ignore the details of the flow profile in the $\tilde{r}$-direction and assume $V(\tilde{r},\tilde{z})$ is the average velocity at $\tilde{z}$ over $\tilde{r}\in\left[0,\tilde{R}(\tilde{z})\right]$.

The former two assumptions are easily justified by the experimental setup of interest, whereas the latter two are made for mathematical simplicity. Assumptions 1 and 2 reduce the number of independent variables in our problem as $\tilde{T}\left(\tilde{r},\phi,\tilde{z},\tilde{t}\right)=\tilde{T}\left(\tilde{r},\tilde{t}\right)$. By Assumption 3, the overall temperature profile in the extruder is constant and thus $\tilde{t}$ corresponds to the time that a particular polymer element has been in the extruder since its insertion. To simplify calculation, we use a reference frame that moves with the flow of polymer. This is equivalent to the change of variables $\left(\tilde{r},\tilde{t}\right)\to(\tilde{r},\tilde{z})$, where $\tilde{t}=\tilde{z}/V(\tilde{z})=\pi\tilde{R}^2(\tilde{z})\,\tilde{z}/Q$, where Q is the volumetric flow rate (a constant by mass conservation). It follows that

$$\frac{\partial}{\partial\tilde{t}}=\frac{Q}{\pi\tilde{R}^2(\tilde{z})}\frac{\partial}{\partial\tilde{z}}, \tag{2.2}$$

which relies on Assumption 4. Thus, (2.1) becomes

$$\rho c_P\frac{Q}{\pi\tilde{R}^2(\tilde{z})}\frac{\partial\tilde{T}}{\partial\tilde{z}}=\frac{k}{\tilde{r}}\frac{\partial}{\partial\tilde{r}}\left(\tilde{r}\frac{\partial\tilde{T}}{\partial\tilde{r}}\right),\quad 0<\tilde{r}<R_{\max}. \tag{2.3}$$

The cylindrical portion of the hot end is of length H_{cyl} with surface radius $\tilde{R}(\tilde{z})=R_{\max}$ and the tapered portion of the hot end (the nozzle) is of length H_{noz} with surface radius tapering (linearly) from $\tilde{R}(H_{\text{cyl}})=R_{\max}$ to $\tilde{R}(H_{\text{cyl}}+H_{\text{noz}})=R_{\min}$ (see Figure 1.1). The heater maintains the surface $\tilde{r}=\tilde{R}(\tilde{z})$ at a fixed temperature $\tilde{T}=T_{\max}$ for $\tilde{z}\in[0,H_{\text{cyl}}]$; there is no heating in the nozzle, but $\tilde{T}\approx T_{\max}$ at $\tilde{r}=\tilde{R}(\tilde{z})$ for $\tilde{z}\in(H_{\text{cyl}},H_{\text{cyl}}+H_{\text{noz}}]$ due to the high thermal conductivity of materials used to construct the hot end. Note that $T_{\max}>T_*$, where T_* is the pliancy temperature defined to be the glass-rubber transition temperature for amorphous polymers or the melting point for crystalline polymers. The polymer is inserted at $\tilde{z}=0$ with $\tilde{T}=T_{\text{i}}<T_*$.

Motivated by these conditions, we scale the problem as follows:

$$r = \frac{\tilde{r}}{R_{\max}}, \quad z = \frac{\tilde{z}}{H_{\text{cyl}}}, \quad T(r,z) = \frac{\tilde{T}\left(\tilde{r},\tilde{z}\right) - T_*}{\Delta T}, \quad \Delta T = T_* - T_{\text{i}}. \tag{2.4}$$

For later convenience, we also introduce the following dimensionless quantities:

$$\beta = \frac{R_{\min}}{R_{\max}}, \quad z_{\text{end}} = \frac{H_{\text{cyl}} + H_{\text{noz}}}{H_{\text{cyl}}}. \tag{2.5}$$

With (2.4), (2.3) becomes

$$\frac{1}{R^2(z)}\frac{\partial T}{\partial z} = \frac{\text{Pe}^{-1}}{r}\frac{\partial}{\partial r}\left(r\frac{\partial T}{\partial r}\right), \quad \text{Pe} = \frac{\rho c_P Q}{\pi k H_{\text{cyl}}}, \tag{2.6}$$

where Pe is the *Péclet number*, the ratio of thermal advection to thermal diffusion. Note that since Q is constant, $Q = \pi\tilde{R}^2(0)V(0) = \pi R_{\max}^2 V_{\min}$, where $V = V(\tilde{z})$ is the average flow velocity in the $\tilde{z}$-direction, and thus Pe can be written as

$$\text{Pe} = \frac{\rho c_P R_{\max}^2 V_{\min}}{k H_{\text{cyl}}}. \tag{2.7}$$

Because experimental measurements were made of $R_{\max}$ and $V_{\min}$ rather than Q, the formulation of Pe given in (2.7) is used for computation rather than that of (2.6). The dimensionless system is shown in Figure 2.1.

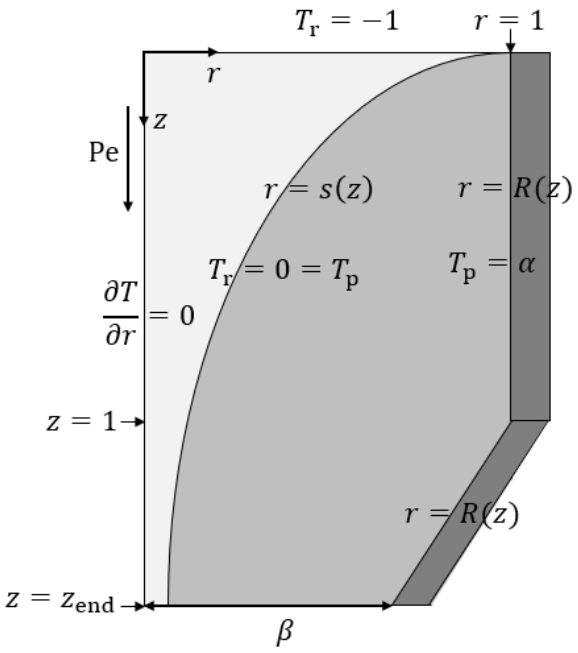


Figure 2.1: Cross-section of half of hot end in dimensionless coordinates (not to scale). Light grey is rigid (glass or crystalline) polymer, medium grey is pliant (rubber or melted) polymer, and dark grey is the heater.

The initial and boundary conditions discussed previously become

$$T_{\mathrm{r}}(r, 0) = -1, \tag{2.8a}$$

$$T_{\mathrm{p}}(R(z), z) = \alpha, \quad \alpha = \frac{T_{\max} - T_*}{\Delta T} > 0, \tag{2.8b}$$

where subscripts r and p denote rigid and pliant regions, respectively. Furthermore, due to the axisymmetry of the problem, there is a no-flux boundary condition at the centerline in accordance with the first law of thermodynamics:

$$\left.\frac{\partial T}{\partial r}\right|_{r=0} = 0. \tag{2.9}$$

Lastly, the interface between the rigid and pliant regions gives way to a free boundary problem for the front $r = s(z)$. This proves to be unimportant in the amorphous case but will be discussed more thoroughly in §4, as it is significant to the solution for crystalline polymers when a phase transition occurs.

In order to simplify calculations, we introduce the change of variables $(r, z) \to (y, z)$ where $y = r/R$, which transforms (2.6) to

$$\frac{\partial T}{\partial z} - \frac{yR'}{R}\frac{\partial T}{\partial y} = \frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T}{\partial y}\right). \tag{2.10}$$

The transformed initial and boundary conditions for (2.10) are

$$T_{\mathrm{r}}(y, 0) = -1, \tag{2.11a}$$

$$T_{\mathrm{p}}(1, z) = \alpha, \tag{2.11b}$$

$$\left.\frac{\partial T}{\partial y}\right|_{y=0} = 0. \tag{2.11c}$$

Chapter 3

THE AMORPHOUS CASE

3.1 Solution in a Taper

First, we consider the amorphous case in a tapered hot end, i.e., without a cylindrical portion (Figure 3.1). We consider the tapered problem to better understand the behavior in this portion of the hot end before moving onto the real geometry of a cylinder feeding into a taper. In this model problem, we still consider a system of length H_{cyl} rather than the length of the nozzle H_{noz}; this choice is equivalent to modeling the cylindrical portion of the hot end as a taper rather than trying to capture information about the nozzle itself and is made to allow for comparison to the cylindrical case considered in [5]. Thus, we still use the scaling $z = \tilde{z}/H_{\text{cyl}}$. In amorphous polymers, the pliancy temperature T_* is the glass-rubber transition temperature T_{g}. Furthermore, the material properties are not expected to change significantly across the pliancy front $r = s(z)$ and thus we need not track this moving boundary. For this reason, this section will not use the subscripts r and p on the variable T.

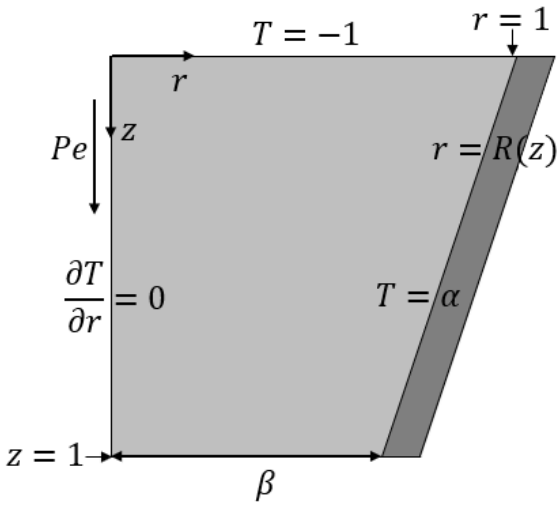


Figure 3.1: Cross-section of half of tapered hot end in dimensionless coordinates (not to scale). Medium grey is amorphous polymer and dark grey is the heater.

In order to characterize the temperature profile, we wish to solve (2.10) subject to (2.11). We do so by solving an equivalent problem with homogeneous boundary conditions by defining $\Theta(y, z)$ as follows:

$$\Theta(y, z) = \frac{\alpha - T(y, z)}{\alpha + 1}, \tag{3.1}$$

which has the following initial and boundary conditions:

$$\Theta(y, 0) = 1, \tag{3.2a}$$

$$\Theta(1, z) = 0, \tag{3.2b}$$

$$\left.\frac{\partial \Theta}{\partial y}\right|_{y=0} = 0. \tag{3.2c}$$

In hopes of using a standard separation of variables approach, we assume $\Theta(y, z) = Y(y)Z(z)$. With this assumption, (2.10) is equivalent to

$$\frac{Z'}{Z} = \frac{\mathrm{Pe}^{-1}}{yY}\frac{\partial}{\partial y}(yY') + y\frac{R'}{R}\frac{Y'}{Y}. \tag{3.3}$$

In order for separation of variables to work, both sides of (3.3) must be constant. An important consequence of this is that R'/R must be a constant (since R is dependent

on z). With the additional conditions that $R(0) = 1$ and $R(1) = \beta$, we are forced to approximate the linear taper as

$$R(z) = e^{-\gamma z}, \quad \gamma = \log \beta^{-1}. \tag{3.4}$$

(3.4) implies $R'/R = -\gamma$.

For later convenience, we introduce another change of variables $(y, z) \to (x, z)$ where $x = y^2 \gamma \operatorname{Pe}/2$. The new system for $\Theta(x, y)$ is

$$\frac{\partial \Theta}{\partial z} + 2\gamma x \frac{\partial \Theta}{\partial x} = 2\gamma \frac{\partial}{\partial x}\left(x \frac{\partial \Theta}{\partial x}\right), \tag{3.5a}$$

$$\Theta(x, 0) = 1, \tag{3.5b}$$

$$\Theta\left(\frac{\gamma \operatorname{Pe}}{2}, z\right) = 0, \tag{3.5c}$$

$$\left|\frac{\partial \Theta}{\partial x}\right|_{x=0} \leq \mu, \text{ for some } \mu \in \mathbb{R}, \tag{3.5d}$$

where (3.5d) is obtained using the following:

$$0 = \lim_{y \to 0} \frac{\partial \Theta}{\partial y}(y, z) = \lim_{x \to 0} \sqrt{2\gamma \operatorname{Pe} x} \frac{\partial \Theta}{\partial x}(x, z), \tag{3.6}$$

which follows from (3.2c).

To use separation of variables in the new coordinate system, we assume $\Theta(x, z) = X(x)Z(z)$. With (3.5), this gives

$$\frac{Z'}{Z} = 2\gamma \frac{(1 - x)X' + xX''}{X} = -2\gamma\lambda, \tag{3.7a}$$

$$X\left(\frac{\gamma \operatorname{Pe}}{2}\right) = 0, \tag{3.7b}$$

$$|X'(0)| \leq \nu, \text{ for some } \nu \in \mathbb{R}, \tag{3.7c}$$

where λ is a real and positive constant to be determined. The initial condition (not described in (3.7)) will be discussed shortly. Observe that the ODE for X is of the form

$$xX'' + (b - x)X' - ax = 0, \tag{3.8}$$

which is known as Kummer's equation. In our case, $a = -\lambda$ and $b = 1$ and thus the general solution is

$$X(x) = C_1 M\left(-\lambda, 1, x\right) + C_2 U\left(-\lambda, 1, x\right), \tag{3.9}$$

where M and U are the confluent hypergeometric functions of the first and second kinds (also known as Kummer's M and U functions) and C_1 and C_2 are constants to be determined. Since $U\left(-\lambda, 1, 0\right)$ is singular, we deduce $C_2 = 0$ and thus

$$X(x) \propto M\left(-\lambda, 1, x\right), \tag{3.10}$$

in order to satisfy (3.7c). The ODE for X in (3.7a) can also be written as

$$\left(xe^{-x}X'\right)' + \lambda e^{-x}X = 0, \tag{3.11}$$

which is the canonical form of the ODE for a singular Sturm-Liouville problem. This tells us that there are infinitely many linearly dependent solutions for X given by the eigenfunctions

$$X_n(x) = M\left(-\lambda_n, 1, x\right), \tag{3.12}$$

where we have taken $C_1 = 1$. The eigenfunctions have associated eigenvalues λ_n determined by the eigenvalue condition:

$$M\left(-\lambda_n, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right) = 0, \tag{3.13}$$

as per (3.7b). After determining each λ_n, we can find the respective solutions to the ODE for Z as given by

$$Z_n(z) \propto e^{-2\gamma\lambda_n z}. \tag{3.14}$$

By the principle of superposition, (3.12) and (3.14) give the following series solution for Θ:

$$\Theta(x, z) = \sum_{n=1}^{\infty} D_n X_n(x) Z_n(z) = \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right) e^{-2\gamma\lambda_n z}, \tag{3.15}$$

where each D_n is a constant to be determined. Note that this means our initial condition is given by

$$1 = \Theta(x, 0) = \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right). \tag{3.16}$$

Sturm-Liouville theory also tells us that our eigenfunctions must satisfy the following orthogonality relation:

$$\int_0^{\gamma\,\mathrm{Pe}\,/2} X_m(x)X_n(x)e^{-x}\,\mathrm{d}x = 0, \quad m \neq n, \tag{3.17}$$

where the weight function e^{-x} can be deduced from (3.11). Thus, if we multiply (3.16) through by some eigenfunction $X_m(x)$ and the weight function e^{-x} and subsequently integrate with respect to x over $x \in [0, \gamma\,\mathrm{Pe}\,/2]$, we obtain

$$\begin{aligned}\int_0^{\gamma\,\mathrm{Pe}\,/2} X_m(x)e^{-x}\,\mathrm{d}x &= \int_0^{\gamma\,\mathrm{Pe}\,/2} X_m(x)e^{-x}\sum_{n=1}^{\infty} D_n X_n\,\mathrm{d}x \\ &= \int_0^{\gamma\,\mathrm{Pe}\,/2} D_m X_m^2(x)e^{-x}\,\mathrm{d}x. \end{aligned}\tag{3.18}$$

It follows that

$$D_m = \frac{\int_0^{\gamma\,\mathrm{Pe}\,/2} X_m(x)e^{-x}\,\mathrm{d}x}{\int_0^{\gamma\,\mathrm{Pe}\,/2} X_m^2(x)e^{-x}\,\mathrm{d}x} = \frac{\int_0^{\gamma\,\mathrm{Pe}\,/2} M\left(-\lambda_m, 1, x\right)e^{-x}\,\mathrm{d}x}{\int_0^{\gamma\,\mathrm{Pe}\,/2} M^2\left(-\lambda_m, 1, x\right)e^{-x}\,\mathrm{d}x}. \tag{3.19}$$

Using [13, Eq. 13.3.21], we obtain

$$\begin{aligned}\int_0^{\gamma\,\mathrm{Pe}\,/2} M\left(-\lambda_m, 1, x\right)e^{-x}\,\mathrm{d}x &= xM\left(1-\lambda_m, 2, x\right)e^{-x}\Big|_0^{\gamma\,\mathrm{Pe}\,/2} \\ &= \frac{\gamma\,\mathrm{Pe}}{2}M\left(1-\lambda_m, 2, \frac{\gamma\,\mathrm{Pe}}{2}\right)e^{-\gamma\,\mathrm{Pe}\,/2}. \end{aligned}\tag{3.20}$$

Using [13, Eq. 13.3.4], we obtain

$$\frac{\gamma\,\mathrm{Pe}}{2}M\left(1-\lambda_n, 2, \frac{\gamma\,\mathrm{Pe}}{2}\right) = M\left(1-\lambda_m, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right) - M\left(-\lambda_m, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right) \tag{3.21}$$

$$= M\left(1-\lambda_m, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right), \tag{3.22}$$

where we have used (3.13). Therefore, (3.23) can be written as

$$D_m = \frac{M\left(1-\lambda_m, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right)e^{-\gamma\,\mathrm{Pe}\,/2}}{\int_0^{\gamma\,\mathrm{Pe}\,/2} M^2\left(-\lambda_m, 1, x\right)e^{-x}\,\mathrm{d}x}. \tag{3.23}$$

In summary, the solution is

$$\Theta(x, z) = \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right)e^{-2\gamma\lambda_n z}, \tag{3.24}$$

where D_n is given by (3.23) and each respective λ_n is computed using (3.13).

Sturm-Liouville theory tells us that the series in (3.24) converges to the solution of (3.5a). However, there is no way to compute this infinite series numerically. Luckily, as λ_n increases with n, the terms in (3.24) decay to zero with increasing n as a result of the solution for $Z_n(z)$, an exponential decay function. This allows us to accurately approximate (3.24) with finitely many terms. To determine how many terms are necessary, we select a level of precision for Θ at a particular point (x, z). We have continued our analysis by computing $\Theta(0, 1)$ to within 10^{-6}. We chose $z = 1$ in order to control the error at the exit and $x = 0$ since this is the coldest point of the exit cross-section and thus should control whether the polymer flows. The value of 10^{-6} is used to balance accuracy with computational efficiency. Observe that

$$\Theta(0,1) = \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, 0\right) e^{-2\gamma\lambda_n} = \sum_{n=1}^{N} D_n e^{-2\gamma\lambda_n} + O\left(D_{N+1} e^{-2\gamma\lambda_{N+1}}\right). \tag{3.25}$$

Thus, we need to find N such that

$$D_{N+1} e^{-2\gamma\lambda_{N+1}} \leq 10^{-6}. \tag{3.26}$$

To do so, we solve for the minimum value of λ_{N+1} that satisfies (3.26) for any γ and Pe of interest. We then approximate (3.24) by as many terms as there are eigenvalues less than to λ_{N+1}. With the experimental setup of interest, we need only one or two terms in any case.

The temperature profile for the median datum is given in Figure 3.2.

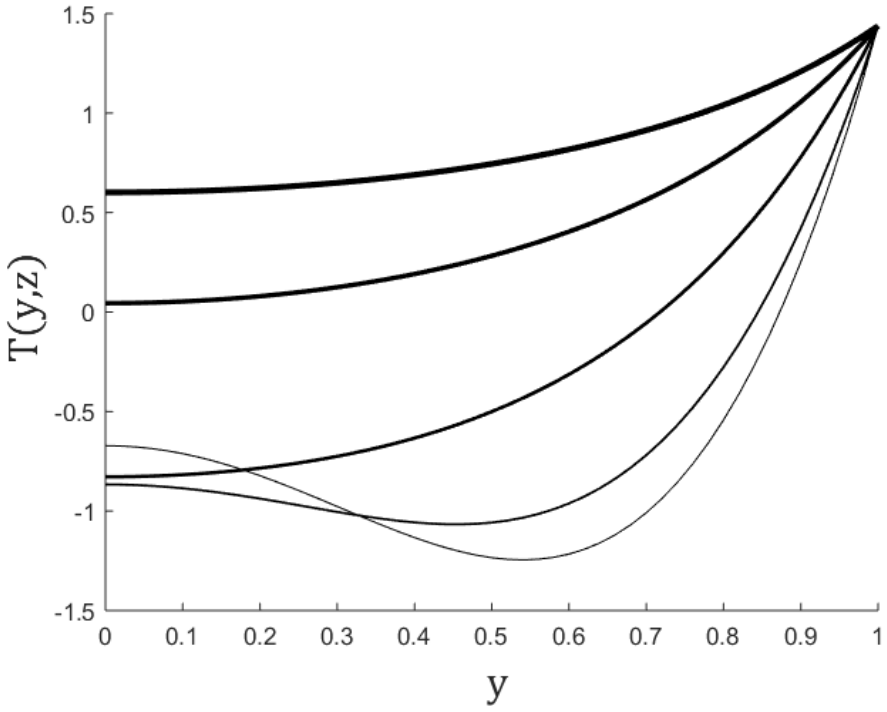


Figure 3.2: Plot of (3.1) with (3.24) for $z \in \{0.008, 0.04, 0.2, 0.6, 1\}$, where z increases with line thickness, for the median experimental datum $(\alpha, \mathrm{Pe}) = (1.44, 1.94)$. In this case, $N = 2$.

Figure 3.2 exhibits some spurious behavior at small z. This is because N is chosen to ensure convergence of $\Theta(0, 1)$ to within 10^{-6} at $z = 1$. From (3.24), we see that $\Theta(x, z)$ decays exponentially with z and thus convergence at small z requires more terms to ensure the same accuracy.

3.2 Average Temperature as Threshold Condition

Now that we have a solution for the temperature profile given by (3.1) with (3.24), we wish to find a temperature-based threshold condition that predicts when polymer extrusion is successful. In this section, we consider threshold conditions that depend on temperature averages. We start with the temperature average constraints due to their success in similar problems [5].

3.2.1 Cross-Sectional Average at Exit

The polymer is hottest at the exit of the extruder ($z = 1$). Thus, it is reasonable to presume that the threshold condition will have something to do with the temperature at the exit of the extruder. In particular, we require the following condition to be satisfied in order for extrusion to occur:

$$T_t \leq \langle T \rangle(1), \tag{3.27}$$

where T_t is the threshold temperature to be determined by fitting experimental data and $\langle T \rangle(z)$ is the cross-sectional average across the angular and radial coordinates ϕ and r, respectively, at a vertical coordinate z as given by

$$\begin{aligned}\langle T \rangle(z) &= \frac{\int_0^{2\pi}\int_0^{R(z)} T(r,z) r \,\mathrm{d}r\,\mathrm{d}\phi}{\int_0^{2\pi}\int_0^{R(z)} r\,\mathrm{d}r\,\mathrm{d}\phi} = \frac{2\pi\int_0^{R(z)} T(r,z) r\,\mathrm{d}r}{\pi R^2(z)} = 2\int_0^1 T(y,z) y\,\mathrm{d}y \\ &= \alpha - 2(\alpha+1)\int_0^1 \Theta(y,z)\,\mathrm{d}y = \alpha - (\alpha+1)\frac{2}{\gamma\,\mathrm{Pe}}\int_0^{\gamma\,\mathrm{Pe}/2} \Theta(x,z)\,\mathrm{d}x. \end{aligned} \tag{3.28}$$

Substituting (3.24) into (3.28) gives

$$\langle T \rangle(z) = \alpha - (\alpha+1)\frac{2}{\gamma\,\mathrm{Pe}}\sum_{n=1}^{\infty} D_n \int_0^{\gamma\,\mathrm{Pe}/2} M\left(-\lambda_n, 1, x\right)\mathrm{d}x e^{-2\gamma\lambda_n z}.$$

Using [13, Eq. 13.3.18], we obtain

$$\int_0^{\gamma\,\mathrm{Pe}/2} M\left(-\lambda_n, 1, x\right)\mathrm{d}x = xM\left(-\lambda_n, 2, x\right)\Big|_0^{\gamma\,\mathrm{Pe}/2} = \frac{\gamma\,\mathrm{Pe}}{2} M\left(-\lambda_n, 2, \frac{\gamma\,\mathrm{Pe}}{2}\right). \tag{3.29}$$

Thus, $\langle T \rangle(z)$ can be written as

$$\langle T \rangle(z) = \alpha - (\alpha+1)\sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 2, \frac{\gamma\,\mathrm{Pe}}{2}\right) e^{-2\gamma\lambda_n z}. \tag{3.30}$$

Experimental data of $V_{\min}$ versus $T_{\max}$ for ABS was taken from [8]. The experimental parameters used to scale the physical system (e.g., $(T_{\max}, V_{\min}) \to (\alpha, \mathrm{Pe})$) can be found in Appendix A. The results of using this condition are shown in Figure 3.3. Note that Figure 3.3 includes the solution to three different least squares problems (α-intercept, curve fit, and level set), which will be discussed in more detail later in this section. As

will be the case for all figures in this section, the areas under the curves correspond to successful extrusion.

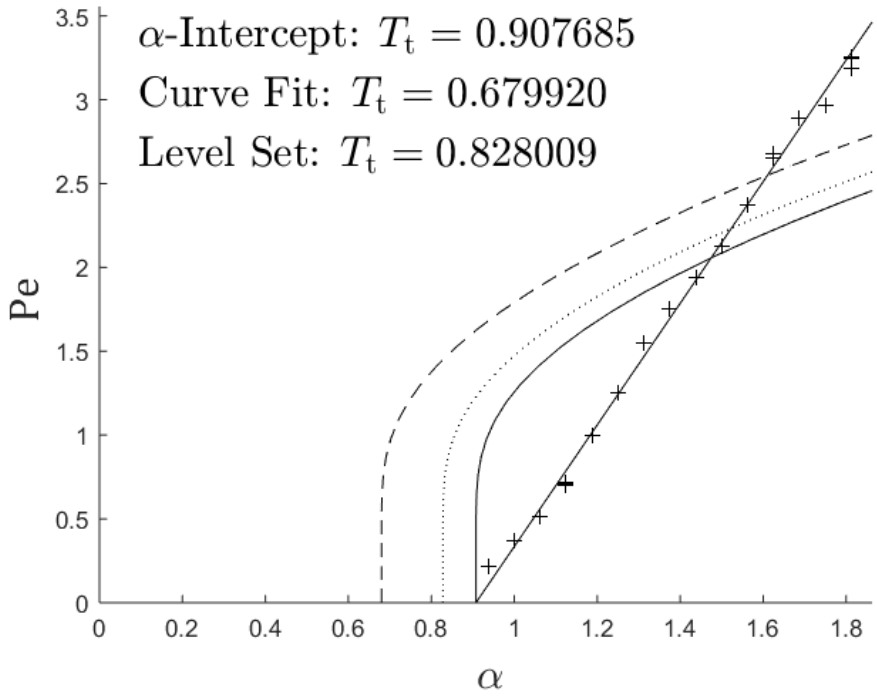


Figure 3.3: Plot of the experimental data (crosses) and (3.27) using the linear and α-intercept fit (solid curves), the curve fit (dashed curve), and the level set fit (dotted curve).

Contrary to the results of [5], Figure 3.3 demonstrates that this threshold condition is unable to predict the experimental data. However, Figure 3.3 does exhibit some important characteristics of the model. Consider the behavior of $\langle T \rangle$ at small Pe. First, expand (3.13) about $\mathrm{Pe} = 0$ (see [13, Eq. 13.3.2]):

$$0 = M\left(-\lambda_n, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right) \sim 1 - \lambda_n \frac{\gamma\,\mathrm{Pe}}{2} \implies \lambda_n \sim \frac{2}{\gamma\,\mathrm{Pe}}, \quad \mathrm{Pe} \to 0. \tag{3.31}$$

Thus, the eigenvalues $\lambda_n \to \infty$ as $\mathrm{Pe} \to 0$. Now consider D_n for small Pe:

$$D_n = \frac{M\left(1-\lambda_n, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right) e^{-\gamma\,\mathrm{Pe}/2}}{\int_0^{\gamma\,\mathrm{Pe}/2} M^2\left(-\lambda_n, 1, x\right) e^{-x}\,\mathrm{d}x} \sim \frac{\left[1+(1-\lambda_n)\,\frac{\gamma\,\mathrm{Pe}}{2}\right]\left(1-\frac{\gamma\,\mathrm{Pe}}{2}\right)}{\int_0^{\gamma\,\mathrm{Pe}/2}\left(1-\lambda_n x\right)^2(1-x)\,\mathrm{d}x}$$
$$\sim \frac{\frac{\gamma\,\mathrm{Pe}}{2}\left(1-\frac{\gamma\,\mathrm{Pe}}{2}\right)}{\int_0^{\gamma\,\mathrm{Pe}/2}\left(1-\frac{2}{\gamma\,\mathrm{Pe}}x\right)^2(1-x)\,\mathrm{d}x} = 12\frac{2-\gamma\,\mathrm{Pe}}{8-\gamma\,\mathrm{Pe}}, \quad \mathrm{Pe}\to 0, \tag{3.32}$$

where we have used the asymptotic solution for λ_n from (3.31) and the fact that the variable of integration x in the denominator is small because $x \in [0, \gamma\,\mathrm{Pe}/2]$. This shows that each D_n remains bounded as $\mathrm{Pe} \to 0$. Since every term in the series in (3.30) is proportional to $D_n e^{-2\gamma\lambda_n z}$ and $\lambda_n \to \infty$ as $\mathrm{Pe} \to 0$, those terms decay away as $\mathrm{Pe} \to 0$ and thus

$$\langle T\rangle(z) \sim \alpha, \quad \mathrm{Pe}\to 0. \tag{3.33}$$

This means that the fitted value for T_t corresponds to the α-intercept of the fitted curves, as demonstrated in Figure 3.3.

Since (3.27) is non-linear (as will be the rest of the developed conditions), we have multiple choices of least squares problems to solve that are not equivalent. We chose to solve three different least squares problems. First, as demonstrated by Figures 3.3, 3.4, and 3.5, we observe that the amorphous polymer data appears to be linear in the (α, Pe)-plane. As shown above, the threshold temperature T_t corresponds to the α-intercept of the fitted curves (this will be shown for the other amorphous conditions as well). Ergo, we expect the fitted value of T_t to be close to the α-intercept of a linear fit of the experimental data. Thus, the first fitting method we consider is imposing T_t to be the α-intercept of a linear fit and then generate a model-based "fitted" curve from this value of T_t. This fitting method will be referred to as the "α-intercept" fit.

The next fitting method we consider is the minimization of $\left|\boldsymbol{\alpha} - \alpha\left(\mathbf{Pe}; T_\mathrm{t}\right)\right|^2$ over $T_\mathrm{t} \in \mathbb{R}$, where $\boldsymbol{\alpha}$ is a vector of the experimental α data and $\alpha(\mathbf{Pe}; T_\mathrm{t})$ is some model-based function for α that depends on the variable Pe and the parameter T_t (obtained from solving $T_\mathrm{t} = \langle T\rangle(1)$ for α in this case), where $\mathbf{Pe}$ is a vector of the experimental Pe data. This is done using `lsqcurvefit`, a built-in MATLAB non-linear curve fitting

function. This fitting method will be referred to as the "curve fit." Note that we forgo the $|\mathbf{Pe} - \mathrm{Pe}(\boldsymbol{\alpha}; T_\mathrm{t})|^2$ minimization problem due to its computational complexity.

The last fitting method we consider is the minimization of $|T_\mathrm{t} \cdot \mathbf{1} - T(\boldsymbol{\alpha}, \mathbf{Pe})|^2$, where T_t is a fitting parameter to be determined, $\mathbf{1}$ is the all ones vector, and $T(\boldsymbol{\alpha}, \mathbf{Pe})$ is some temperature condition ($\langle T \rangle(1)$ in this case), where $\boldsymbol{\alpha}$ is a vector of the experimental α data and $\mathbf{Pe}$ is a vector of the experimental Pe data. Since this a one-parameter least squares problem, the minimization of $|T_\mathrm{t} \cdot \mathbf{1} - T(\boldsymbol{\alpha}, \mathbf{Pe})|^2$ is equivalent to taking the average of $T(\alpha, \mathrm{Pe})$ over each experimental datum (α, Pe), i.e., finding a level set at $T = T_\mathrm{t}$ in (T, α, Pe)-space. For this reason, this fitting method will be referred to as the "level set" fit. The runtimes to construct Figure 3.3 and the analogous figures for the remaining conditions (Figures 3.4 and 3.5) were all less than 40 seconds on a 2.60 GHz processor. These quick runtimes indicate that it is not too computationally expensive to do all three fitting methods.

As evident in Figure 3.3 (and as will be shown in Figures 3.4, and 3.5), each least squares problem gives slightly different quantitative results, but the qualitative behavior and ability to predict the experimental data remains the same across all three. For this reason, when fitting the crystalline polymer data, we will use only one fitting method for each condition considered. Specifically, we will use the level set fit when solving one-parameter least squares problems and the curve fit method when solving two-parameter least squares problems (see §4 for details).

3.2.2 Full Average

In [5], it was shown that extrusion success depends on the average viscosity in the entire cylinder, where average temperature is used as analog for average viscosity. Motivated by this work, we require the following condition to be satisfied in order for extrusion to occur:

$$T_t \leq \bar{T}(1), \tag{3.34}$$

where $\bar{T}(z)$ is the cumulative cross-sectional temperature up to length z as given by

$$\frac{\mathrm{d}\bar{T}}{\mathrm{d}z} = \langle T \rangle(z), \quad \bar{T}(0) = 0. \tag{3.35}$$

Therefore,

$$\bar{T}(z) = \int_0^z \langle T \rangle(\zeta)\,\mathrm{d}\zeta = \alpha - (\alpha+1)\frac{2}{\gamma\,\mathrm{Pe}}\int_0^z \int_0^{\gamma\,\mathrm{Pe}/2} \Theta(x,\zeta)\,\mathrm{d}x\,\mathrm{d}\zeta. \tag{3.36}$$

To simplify (3.36), we start by integrating both sides of (3.5a) with respect to $z \mapsto \zeta$ over $\zeta \in [0, z]$ for some fixed z:

$$\begin{aligned}
\int_0^z \left[\frac{\partial\Theta}{\partial\zeta} + 2\gamma x\frac{\partial\Theta}{\partial x}\right]\mathrm{d}\zeta &= \Theta(x,z) - \Theta(x,0) + 2\gamma x\frac{\mathrm{d}}{\mathrm{d}x}\int_0^z \Theta(x,\zeta)\,\mathrm{d}\zeta \\
&= \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right) e^{-2\gamma\lambda_n z} - 1 + 2\gamma x\frac{\mathrm{d}\theta_z}{\mathrm{d}x}, \qquad (3.37a) \\
\int_0^z 2\gamma\frac{\partial}{\partial x}\left(x\frac{\partial\Theta}{\partial x}\right)\mathrm{d}\zeta &= 2\gamma x\frac{\mathrm{d}^2}{\mathrm{d}x^2}\int_0^z \Theta(x,\zeta)\,\mathrm{d}\zeta + 2\gamma\frac{\mathrm{d}}{\mathrm{d}x}\int_0^z \Theta(x,\zeta)\,\mathrm{d}\zeta \\
&= 2\gamma x\frac{\mathrm{d}^2\theta_z}{\mathrm{d}x^2} + 2\gamma\frac{\mathrm{d}\theta_z}{\mathrm{d}x}, \qquad (3.37b) \\
\theta_z(x) &= \int_0^z \Theta(x,\zeta)\,\mathrm{d}\zeta, \qquad (3.37c)
\end{aligned}$$

where we have used (3.24). Note that $\bar{T}(z)$ in terms of θ_z is

$$\bar{T}(z) = \alpha - (\alpha+1)\frac{2}{\gamma\,\mathrm{Pe}}\int_0^{\gamma\,\mathrm{Pe}/2} \theta_z(x)\,\mathrm{d}x. \tag{3.38}$$

We have the following second order ODE for θ_z:

$$\begin{aligned}
\frac{1}{x}\left[\sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right) e^{-2\gamma\lambda_n z} - 1\right] &= 2\gamma\frac{\mathrm{d}^2\theta_z}{\mathrm{d}x^2} + 2\gamma\frac{1-x}{x}\frac{\mathrm{d}\theta_z}{\mathrm{d}x}, \qquad (3.39a) \\
\theta_z\left(\frac{\gamma\,\mathrm{Pe}}{2}\right) &= 0, \qquad (3.39b)
\end{aligned}$$

where the boundary condition on θ_z comes from (3.5c). Multiplying (3.39a) through by xe^{-x} gives

$$\begin{aligned}
e^{-x}\left[\sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right) e^{-2\gamma\lambda_n z} - 1\right] &= 2\gamma x e^{-x}\frac{\mathrm{d}^2\theta_z}{\mathrm{d}x^2} + 2\gamma(1-x)e^{-x}\frac{\mathrm{d}\theta_z}{\mathrm{d}x} + 1 \\
&= 2\gamma\frac{\mathrm{d}}{\mathrm{d}x}\left(xe^{-x}\frac{\mathrm{d}\theta_z}{\mathrm{d}x}\right). \qquad (3.40)
\end{aligned}$$

Next, we integrate (3.40) with respect to $x \mapsto \xi$ over $\xi \in [0, x]$:

$$\int_0^x \left[\sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, \xi\right) e^{-\xi} e^{-2\gamma\lambda_n z} - e^{-\xi}\right] \mathrm{d}\xi$$
$$= \sum_{n=1}^{\infty} D_n x e^{-x} M\left(1 - \lambda_n, 2, x\right) e^{-2\gamma\lambda_n z} - (1 - e^{-x}), \tag{3.41a}$$
$$\int_0^x 2\gamma \frac{\mathrm{d}}{\mathrm{d}\xi}\left(\xi e^{-\xi} \frac{\mathrm{d}\theta_z}{\mathrm{d}\xi}\right) \mathrm{d}\xi = 2\gamma x e^{-x} \frac{\mathrm{d}\theta_z}{\mathrm{d}x}, \tag{3.41b}$$

where we have again used [13, Eq. 13.3.21]. This gives the following first order ODE for θ_z:

$$2\gamma \frac{\mathrm{d}\theta_z}{\mathrm{d}x} = \sum_{n=1}^{\infty} D_n M\left(1 - \lambda_n, 2, x\right) e^{-2\gamma\lambda_n z} + \frac{1 - e^x}{x}. \tag{3.42}$$

We again integrate (3.42) with respect to $x \mapsto \xi$ except now over $\xi \in [\gamma\,\mathrm{Pe}\,/2, x]$:

$$\int_{\gamma\,\mathrm{Pe}\,/2}^{x} 2\gamma \frac{\mathrm{d}\theta_z}{\mathrm{d}\xi}\,\mathrm{d}\xi = 2\gamma\theta_z, \tag{3.43a}$$
$$\int_{\gamma\,\mathrm{Pe}\,/2}^{x} \left[\sum_{n=1}^{\infty} D_n M\left(1 - \lambda_n, 2, \xi\right) e^{-2\gamma\lambda_n z} + \frac{1 - e^{\xi}}{\xi}\right] \mathrm{d}\xi$$
$$= \log\left(\frac{2x}{\gamma\,\mathrm{Pe}}\right) - \mathrm{Ei}(x) + \mathrm{Ei}\left(\frac{\gamma\,\mathrm{Pe}}{2}\right) - \sum_{n=1}^{\infty} \frac{D_n}{\lambda_n} M\left(-\lambda_n, 2, x\right) e^{-2\gamma\lambda_n z}, \tag{3.43b}$$

where Ei is the exponential integral defined as

$$\mathrm{Ei}(x) = -\int_1^{\infty} \frac{e^{\xi x}}{\xi}\,\mathrm{d}\xi. \tag{3.44}$$

(3.43b) relies on [13, Eq. 13.3.15]:

$$\int_{\gamma\,\mathrm{Pe}\,/2}^{x} M\left(1 - \lambda_n, 2, \xi\right) \mathrm{d}\xi = -\frac{1}{\lambda_n} M\left(-\lambda_n, 1, \xi\right)\Big|_{\gamma\,\mathrm{Pe}\,/2}^{x}$$
$$= -\frac{1}{\lambda_n} M\left(-\lambda_n, 1, x\right), \tag{3.45}$$

where we have used (3.13). Thus, θ_z is given by

$$\theta_z(x) = \frac{1}{2\gamma}\left[\log\left(\frac{2x}{\gamma\,\mathrm{Pe}}\right) - \mathrm{Ei}(x) + \mathrm{Ei}\left(\frac{\gamma\,\mathrm{Pe}}{2}\right) - \sum_{n=1}^{\infty} \frac{D_n}{\lambda_n} M\left(-\lambda_n, 2, x\right) e^{-2\gamma\lambda_n z}\right]. \tag{3.46}$$

Substituting (3.46) into (3.38) gives

$$\bar{T}(z) = \alpha + (\alpha+1)\frac{1}{2\gamma}\left[1 + \frac{2}{\gamma\,\mathrm{Pe}}\left(1 - e^{\gamma\,\mathrm{Pe}/2}\right) + \sum_{n=1}^{\infty}\frac{D_n}{\lambda_n}M\left(-\lambda_n, 2, \frac{\gamma\,\mathrm{Pe}}{2}\right)e^{-2\gamma\lambda_n z}\right], \tag{3.47}$$

which again relies on [13, Eq. 13.3.18]. The results of this condition are shown in Figure 3.4.

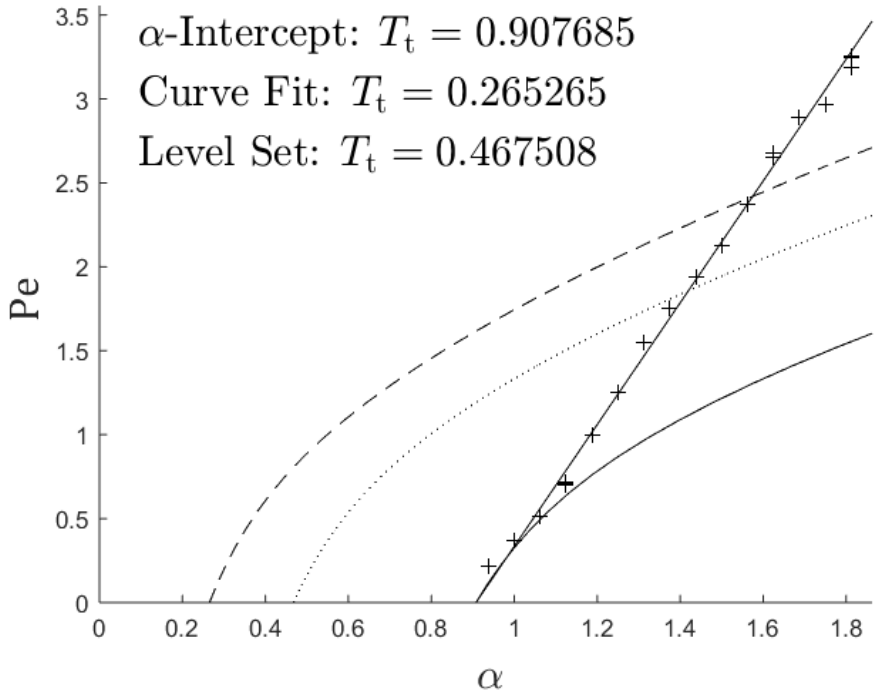


Figure 3.4: Plot of the experimental data (crosses) and (3.34) using the linear and α-intercept fit (solid curves), the curve fit (dashed curve), and the level set fit (dotted curve).

Observe that T_t is lower in the case of the full average than in the case of the cross-sectional average for each respective fitting method except for the α-intercept fit which is, of course, the same for both threshold conditions. This makes sense physically since the polymer is heated for $z \in [0,1]$ and thus the cross-sectional average at $z = 1$ will be the hottest in the cylinder, so $\bar{T} < \langle T \rangle(1)$. Therefore, to achieve extrusion given

the same data set, T_t must be less in the fully averaged case. Despite this physical result, Figure 3.4 demonstrates that this threshold condition is also unable to predict the experimental data, again contrary to the results of [5].

To analyze the behavior of $\bar{T}$ at small Pe, we use the asymptotic solution for $\langle T \rangle$ from (3.33) in the solution for $\bar{T}$ from (3.36):

$$\bar{T}(z) = \int_0^z \langle T \rangle(\zeta)\, \mathrm{d}\zeta \sim \int_0^z \alpha\, \mathrm{d}\zeta = \alpha z, \quad \mathrm{Pe} \to 0. \tag{3.48}$$

Since (3.34) uses $z = 1$, this means that the fitted value for T_t corresponds to the α-intercept of the fitted curves, as demonstrated in Figure 3.4.

3.3 Exit Temperature as Threshold Condition

We expect extrusion to fail if the polymer is too rigid at the exit of extruder. This motivates a third fitting condition based on the temperature on the centerline at the exit of the extruder:

$$T_t \leq T(0, 1), \tag{3.49}$$

where $T(y, z)$ is given by

$$T(y, z) = \alpha - (\alpha + 1)\Theta\left(\frac{\gamma\,\mathrm{Pe}}{2} y^2, z\right). \tag{3.50}$$

Substituting (3.24) into (3.50) gives

$$T(y, z) = \alpha - (\alpha + 1) \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, \frac{\gamma\,\mathrm{Pe}}{2} y^2\right) e^{-2\gamma\lambda_n z}. \tag{3.51}$$

The results of this condition are shown in Figure 3.5.

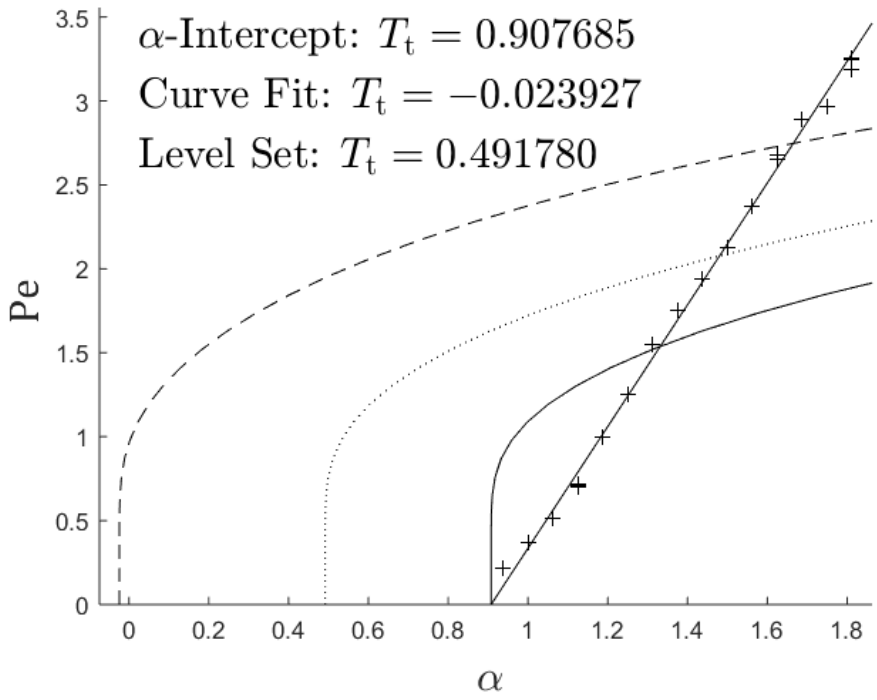


Figure 3.5: Plot of the experimental data (crosses) and (3.49) using the linear and α-intercept fit (solid curves), the curve fit (dashed curve), and the level set fit (dotted curve).

Observe that T_t is lower in the case of the centerline exit temperature than in the case of the cross-sectional average temperature for each respective fitting method (except for the α-intercept fit). This makes sense physically since the polymer is heated from $y = 1$ inward and thus the hot end will be coolest at the centerline for any particular value of z. So, $T(0,1) < \langle T \rangle(1)$. Therefore, to achieve extrusion given the same data set, T_t must be less in the centerline exit case.

To analyze the behavior of T at small Pe, we use a similar argument as in §3.2.1 to deduce that every term in the series in (3.51) decays away as $\mathrm{Pe} \to 0$ and thus

$$T(y,z) \sim \alpha, \quad \mathrm{Pe} \to 0. \tag{3.52}$$

Again, this means that the fitted value for T_t corresponds to the α-intercept of the fitted curves, as demonstrated in Figure 3.5.

Figure 3.5 again shows that this threshold condition is unable to predict the experimental data. Combined with the results of Figures 3.3 and 3.4, it is clear that we are unable to predict the amorphous polymer data by considering the tapered portion of the hot end alone, regardless of the (temperature-based) threshold condition or fitting method considered. Previous work [5] has shown similar methods to be effective when considering the cylindrical portion of the hot end alone, i.e., with the tapered portion ignored. The success of this model over the current model is likely due to the length of the cylindrical portion relative to the tapered portion, where the former is much longer. This means the cylindrical model uses a geometry that more closely aligns with the real system than what has been considered here. For this reason, we expect that a combined model (i.e., one that incorporates both cylinder and taper) would be more effective in predicting the experimental data than the tapered model. Furthermore, the combined model would have an even more realistic geometry than [5], suggesting that it could improve upon the cylindrical model as well; however, any discrepancy between the two should be small due to the short length of the tapered nozzle. This task is left for future work.

Chapter 4

THE CRYSTALLINE CASE

4.1 Governing Equations

Next, we consider the crystalline case in three geometries: a cylinder, a taper, and a taper at the end of a cylinder (i.e., the real case). In crystalline polymers, the pliancy temperature T_* is the melting temperature T_m. At T_m, the polymer goes from the crystalline state, where the long polymer chains are arranged in some ordered structure, to the melted state, where the thermal energy in the system is enough to disrupt the intermolecular forces holding the polymer structure together resulting in a liquid-like disordered state. Unlike the amorphous case, the material properties are expected to change across the melt front $r = s(z)$ separating rigid and pliant regions. Previous works [5, 6, 8] motivate us to neglect the changes in material properties across the melt front in this analysis; however, this effect is still realized by accounting for the latent heat of melting at the front via the Stefan condition (discussion to follow).

Consideration of the melt front gives way to an additional boundary condition that was unused in the amorphous case:

$$T_\mathrm{r}\left(s(z), z\right) = T_\mathrm{p}\left(s(z), z\right) = 0, \quad s(0) = 1. \tag{4.1}$$

Recall that T_r denotes the rigid region $0 < r < s(z)$ (light grey region in Figure 2.1) and T_p denotes the pliant region $s(z) < r < R(z)$ (medium grey region in Figure 2.1). Let us also introduce the following notation:

$$\left[\varphi(r)\right]_s = \varphi_\mathrm{p}(s_+) - \varphi_\mathrm{r}(s_-), \tag{4.2}$$

where φ denotes some quantity that depends on r and $\left[\varphi(r)\right]_s$ denotes the magnitude in the discontinuity of φ at the melt front $r = s(z)$, $s_+ = \lim_{r\to s^+} r$, and $s_- = \lim_{r\to s^-} r$. For example, observe that $[T]_s = T_\mathrm{p}\left(s_+(z), z\right) - T_\mathrm{r}\left(s_-(z), z\right) = 0$.

The crystalline case is a Stefan problem, i.e., a heat-transfer-type problem in which two (or more) regions, each independently governed by the heat equation, are separated by a free boundary corresponding to a phase transition. In our case, the free boundary is the melt front $r = s(z)$. In addition to (4.1), the crystalline case has another boundary condition at $r = s(z)$ known as the Stefan condition. To understand this condition, consider heat flowing through a melt front of infinitesimal thickness $\mathrm{d}\tilde{s}$ for an infinitesimal duration $\mathrm{d}\tilde{t}$. The change in heat flow across the melt front is balanced by the energy required to melt the crystalline polymer:

$$[q]_{\tilde{s}} A \,\mathrm{d}\tilde{t} = \rho c_L A \,\mathrm{d}\tilde{s}, \tag{4.3}$$

where q_p and q_r are the heat fluxes into the melt front from the pliant region and out of the melt front into the rigid region, respectively, A is the (dimensional) area of heat transfer through the front, and c_L is the specific latent heat of melting. By Fourier's law, we know that

$$[q]_{\tilde{s}} = \left[-k\frac{\partial \tilde{T}}{\partial \tilde{r}}\right]_{\tilde{s}}. \tag{4.4}$$

Using (2.2), we deduce that

$$\frac{\mathrm{d}\tilde{s}}{\mathrm{d}\tilde{t}} = \frac{Q}{\pi \tilde{R}^2\left(\tilde{z}\right)}\frac{\mathrm{d}\tilde{s}}{\mathrm{d}\tilde{z}} = \frac{R_{\max}^2 V_{\min}}{\tilde{R}^2\left(\tilde{z}\right)}\frac{\mathrm{d}\tilde{s}}{\mathrm{d}\tilde{z}}. \tag{4.5}$$

Thus, (4.3) is equivalent to

$$\left[-k\frac{\partial \tilde{T}}{\partial \tilde{r}}\right]_{\tilde{s}} = \rho c_L \frac{R_{\max}^2 V_{\min}}{\tilde{R}^2\left(\tilde{z}\right)}\frac{\mathrm{d}\tilde{s}}{\mathrm{d}\tilde{z}}, \tag{4.6}$$

which scales to

$$\left[\frac{\mathrm{St}}{\mathrm{Pe}}\frac{\partial T}{\partial r}\right]_s = -\frac{1}{R^2(z)}\frac{\mathrm{d}s}{\mathrm{d}z}, \quad \mathrm{St} = \frac{\Delta T c_P}{c_L}, \tag{4.7}$$

where St is the *Stefan number*, the ratio of sensible heat to latent heat. Motivated by [5, 6, 8], we assume $[k]_s, [\rho]_s, [c_P]_s \approx 0$ and thus

$$\frac{\mathrm{St}}{\mathrm{Pe}}\left[\frac{\partial T}{\partial r}\right]_s = -\frac{1}{R^2(z)}\frac{\mathrm{d}s}{\mathrm{d}z}. \tag{4.8}$$

An important result to note from (4.8) is that without a discontinuity in the radial derivative of the temperature at $r = s$, then $\mathrm{d}s/\,\mathrm{d}z$ would be identically zero and there

would be no evolution of the melt front. Also note that the amorphous system can be thought of as a special case of this Stefan problem where $\mathrm{St} \to \infty$ as $c_L \to 0$, which implies $[q]_{\tilde{s}} \to 0$.

As in §2, we introduce the change of variables $(r, z) \to (y, z)$ where $y = r/R$. Let $\sigma = s/R$. Therefore, (4.8) is equivalent to

$$\frac{\mathrm{St}}{\mathrm{Pe}} \left[\frac{\partial T}{\partial y} \right]_{\sigma} = -\frac{\mathrm{d}\sigma}{\mathrm{d}z} - \frac{R'}{R}\sigma. \tag{4.9}$$

To summarize the crystalline case, we have the following systems in the rigid region, in the pliant region, and at the melt front, respectively:

$$0 < y < \sigma: \qquad \frac{\partial T_{\mathrm{r}}}{\partial z} - \frac{yR'}{R}\frac{\partial T_{\mathrm{r}}}{\partial y} = \frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_{\mathrm{r}}}{\partial y} \right), \tag{4.10a}$$

$$\left. \frac{\partial T_{\mathrm{r}}}{\partial y} \right|_{y=0} = 0, \tag{4.10b}$$

$$T_{\mathrm{r}}(y, 0) = -1; \tag{4.10c}$$

$$\sigma < y < 1: \qquad \frac{\partial T_{\mathrm{p}}}{\partial z} - \frac{yR'}{R}\frac{\partial T_{\mathrm{p}}}{\partial y} = \frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_{\mathrm{p}}}{\partial y} \right), \tag{4.10d}$$

$$T_{\mathrm{p}}(1, z) = \alpha; \tag{4.10e}$$

$$y = \sigma: \qquad \frac{\mathrm{St}}{\mathrm{Pe}} \left[\frac{\partial T}{\partial y} \right]_{\sigma} = -\frac{\mathrm{d}\sigma}{\mathrm{d}z} - \frac{R'}{R}\sigma, \tag{4.10f}$$

$$T_{\mathrm{r}}\left(\sigma(z), z\right) = T_{\mathrm{p}}\left(\sigma(z), z\right) = 0, \tag{4.10g}$$

$$\sigma(0) = 1, \tag{4.10h}$$

where the initial condition on the melt front (4.10h) comes from the fact the polymer that enters the hot end is completely rigid (i.e., $s(0) = 1$) and the initial radius is $R(0) = 1$.

4.2 Additional Approximations

A full solution to (4.10) would require a fully numerical approach. Since we wish to understand the system's dependence on parameters, we introduce some additional approximations that will allow for the development of partially analytical solutions.

4.2.1 The Quasistationary Approximation

In the rigid region, we apply what is known as the *quasistationary approximation*. The quasistationary approximation involves assuming one or more derivative terms (typically those with respect to time) are negligible compared to the remaining terms. This is equivalent to assuming the system (or part of the system) is at steady state, i.e., $\partial/\partial t \to 0$ in a subset of the domain. This approximation is often made to simplify systems with two or more significantly different time scales.

In our case, we take $\partial/\partial z \to 0$ (and thus $\mathrm{d}/\,\mathrm{d}z \to 0$), which means the left-hand side of (4.10a) vanishes. This is equivalent to taking Pe (and thus $V_{\min}$) to zero. After making this simplification, we are left with a second order ODE for T_{r} with respect to y:

$$0 = \frac{1}{y}\frac{\mathrm{d}}{\mathrm{d}y}\left(y\frac{\mathrm{d}T_{\mathrm{r}}}{\mathrm{d}y}\right), \quad 0 < y < \sigma. \tag{4.11}$$

However, there are three boundary conditions in the rigid region: (4.10b), (4.10c), and (4.10g). To rectify the overspecification, we ignore the initial condition. This decision reflects our interest in the solution throughout the interval $z \in [0, z_{\mathrm{end}}]$. The solution to (4.11) subject to (4.10b) and (4.10g) is

$$T_{\mathrm{r}} \equiv 0, \quad 0 < y < \sigma. \tag{4.12}$$

From (4.12), we see that the quasistationary approximation is equivalent to assuming that the time that the crystalline region takes to reach the melting temperature is negligible compared to the time it takes for melting to occur. These are the disparate time scales that correspond to this application of the quasistationary approximation.

It may seem concerning that (4.12) appears to violate our initial condition (4.10c). However, the two are consistent if a thin layer of thickness $O(\mathrm{Pe})$ is inserted near $z = 0$ in which rapid diffusion brings the rigid polymer from the initial condition to the melting temperature (4.12). Therefore, we expect the rapid diffusion layer explanation to hold when Pe is small. Another consequence of neglecting the evolution term in (4.10a) is the overestimation of the speed of $\sigma(z)$ through the rigid region [14].

This results from the omission of the time needed to raise the polymer to the melting temperature.

Another simplifying result of the quasistationary approximation is reduction of the two-phase Stefan problem to a one-phase Stefan problem. More formally, (4.10f) is replaced by

$$\frac{\mathrm{St}}{\mathrm{Pe}} \left.\frac{\partial T_\mathrm{p}}{\partial y}\right|_{y=\sigma} = -\frac{\mathrm{d}\sigma}{\mathrm{d}z} - \frac{R'}{R}\sigma. \tag{4.13}$$

4.2.2 The HBI Method

In the pliant region, we proceed using the heat balance integral (HBI) method. In essence, the HBI method simplifies transport problems with moving boundaries by assuming the governing transport equation is satisfied on average as opposed to at every point in the domain. More precisely, this method transforms the governing PDE into an ODE via the following steps:

1. Assume a general field profile.
2. Impose the relevant boundary conditions to fully specify the general profile up to the moving boundary position.
3. Integrate the governing PDE with respect to the space variable over the relevant interval. (Note that this integral is referred to as the HBI).
4. Solve the resultant ODE for the moving boundary.

This method was first developed by [15] to approximate a one-phase Stefan problem in Cartesian coordinates by a quadratic temperature profile. It has since been extended to other geometries [16], functional forms [17, 18, 19, 20], and solution methods [21, 22]. The HBI method can be understood in analogy to integral methods used to study other transport phenomena, e.g., the momentum integral in boundary-layer theory [23].

As shown in [5], the quasistationary solution to (4.10d) subject to (4.10e) and (4.10g) is linear in $\log y$. Due to the number of boundary conditions in these types of problems, a temperature profile with three degrees of freedom is typically used. These considerations motivate a temperature profile that is quadratic in $\log y$. For

later convenience, we assume a profile that is quadratic in $(1 - \log y/\log\sigma)$ as well. Therefore, the assumed form for T_p is

$$T_\mathrm{p}(y,z) = \alpha\left[a\left(1-\frac{\log y}{\log\sigma}\right)+(1-a)\left(1-\frac{\log y}{\log\sigma}\right)^2\right], \quad \sigma < y < 1, \tag{4.14}$$

where a is a constant to be determined. Note that (4.14) satisfies (4.10e) and (4.10g) automatically.

The final step is to determine a value of a such that (4.14) satisfies the Stefan condition (4.13). However, previous studies suggest that there is an alternate form of the Stefan condition that is more computationally convenient [19, 22, 24]. We start our derivation of this form with the observation that the temperature along the melt front is constant for all z. More formally, the total derivative of T_p along $y = \sigma$ is zero for any z, i.e.,

$$0 = \frac{\mathrm{d}T_\mathrm{p}}{\mathrm{d}z} = \frac{\partial T_\mathrm{p}}{\partial z} + \frac{\partial T_\mathrm{p}}{\partial y}\frac{\mathrm{d}\sigma}{\mathrm{d}z}, \quad y = \sigma(z). \tag{4.15}$$

It follows that

$$\frac{\mathrm{d}\sigma}{\mathrm{d}z} = -\frac{\partial T_\mathrm{p}}{\partial z}\bigg/\frac{\partial T_\mathrm{p}}{\partial y} = -\left[\frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_\mathrm{p}}{\partial y}\right)+\frac{yR'}{R}\frac{\partial T_\mathrm{p}}{\partial y}\right]\bigg/\frac{\partial T_\mathrm{p}}{\partial y}, \quad y = \sigma(z), \tag{4.16}$$

where (4.10d) has been used to evaluate $\partial T_\mathrm{p}/\partial z$ (see §4.2.3 for discussion). We also know from (4.13) that

$$\frac{\mathrm{d}\sigma}{\mathrm{d}z} = -\frac{R'}{R}\sigma - \frac{\mathrm{St}}{\mathrm{Pe}}\frac{\partial T_p}{\partial y}, \quad y = \sigma(z). \tag{4.17}$$

Equating (4.16) and (4.17) and multiplying through by $-\partial T_\mathrm{p}/\partial y$ gives

$$\mathrm{St}\,\mathrm{Pe}^{-1}\left(\frac{\partial T_p}{\partial y}\right)^2 + \frac{R'}{R}\sigma\frac{\partial T_\mathrm{p}}{\partial y} = \frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_\mathrm{p}}{\partial y}\right)+\frac{R'}{R}y\frac{\partial T_\mathrm{p}}{\partial y}, \quad y = \sigma(z). \tag{4.18}$$

Since we are interested in $y = \sigma(z)$, the following holds for any R of interest:

$$\left(\frac{\partial T_p}{\partial y}\right)^2 = \frac{\mathrm{St}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_\mathrm{p}}{\partial y}\right), \quad y = \sigma(z). \tag{4.19}$$

(4.19) gives the desired alternate Stefan condition. The flux condition at the melt front (4.13) is thus replaced by (4.19). By imposing (4.19) on (4.14), we obtain

$$a = \frac{-1+\sqrt{1+2\,\mathrm{St}\,\alpha}}{\mathrm{St}\,\alpha}, \tag{4.20}$$

where the positive root has been taken to satisfy $\partial T_\mathrm{p}/\partial y|_{y=\sigma} > 0$. Physically, we know that $\mathrm{St}, \alpha > 0$. From this, we deduce the following:

$$a = \frac{-1+\sqrt{1+2\,\mathrm{St}\,\alpha}}{\mathrm{St}\,\alpha} \geq \frac{-1+\sqrt{1}}{\mathrm{St}\,\alpha} = 0, \tag{4.21a}$$

$$0 < \mathrm{St}\,\alpha = 2\frac{1-a}{a^2} \implies a < 1, \tag{4.21b}$$

where we have used (4.20) to solve for $\mathrm{St}\,\alpha$ in (4.21b). $0 \leq a < 1$ implies both terms in (4.14) are non-negative.

Now that we have found the coefficients of (4.14), we are brought to Step 3 in the HBI method. In our case, we integrate the heat equation with respect to y over $y \in [0,1]$. However, since the quasistationary approximation in the rigid region results in $T_\mathrm{r} \equiv 0$ for $0 < y < \sigma$, we need only integrate (4.10d) over $y \in [\sigma(z), 1]$ (note that a factor of y is multiplied to both sides of (4.10d) since y is a radial coordinate):

$$\int_{\sigma(z)}^{1} \frac{\partial T_\mathrm{p}}{\partial z} y \,\mathrm{d}y = \int_{\sigma(z)}^{1} \left[\frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_\mathrm{p}}{\partial y}\right) + \frac{yR'}{R}\frac{\partial T}{\partial y}\right] y \,\mathrm{d}y. \tag{4.22}$$

As is a common approach [22], the left-hand side of (4.22) is more easily evaluated after applying Leibniz's integral rule, from which we deduce

$$\int_{\sigma(z)}^{1} \frac{\partial T_\mathrm{p}}{\partial z} y \,\mathrm{d}y = \frac{\mathrm{d}}{\mathrm{d}z}\int_{\sigma(z)}^{1} T_\mathrm{p}(y,z) y \,\mathrm{d}y, \tag{4.23}$$

where we have also applied (4.10g). Therefore, (4.22) is equivalent to

$$\frac{\mathrm{d}}{\mathrm{d}z}\int_{\sigma(z)}^{1} T_\mathrm{p}(y,z) y \,\mathrm{d}y = \int_{\sigma(z)}^{1} \left[\frac{\mathrm{Pe}^{-1}}{y}\frac{\partial}{\partial y}\left(y\frac{\partial T_\mathrm{p}}{\partial y}\right) + \frac{yR'}{R}\frac{\partial T_\mathrm{p}}{\partial y}\right] y \,\mathrm{d}y. \tag{4.24}$$

Substituting (4.14) into (4.24) gives the following ODE for σ:

$$\frac{\mathrm{d}\sigma}{\mathrm{d}z} = \frac{8\,\mathrm{Pe}^{-1}\,\eta(z)\sigma\log\sigma}{2(1-a)+(2-a)\log\sigma+\sigma^2\left[2a(\log\sigma)^2+(2-3a)\log\sigma-2(1-a)\right]}, \tag{4.25a}$$

$$\eta(z) = (1-a)\log\sigma + \frac{\mathrm{Pe}}{4}\frac{R'}{R}\left\{(1-a)\left(1-\sigma^2\right)+\left[2-a\left(1+\sigma^2\right)\right]\log\sigma\right\}, \tag{4.25b}$$

where a is given by (4.20). Though some limiting behavior will be discussed in analytical terms, the complexity of (4.25) precludes a general analytical solution. Thus, we use numerical solvers (in MATLAB) to compute $\sigma(z)$ when fitting experimental data.

4.2.3 Model Shortcomings from Approximations

The solution using the HBI method is valid only over the range of z where $\sigma > 0$. Once $\sigma = 0$, the problem reduces to a separation of variables problem similar to that of §3 with an initial condition determined by the temperature profile at the first value of z where $\sigma = 0$. However, it follows from (4.25) that σ cannot go to zero in finite z. This can be understood by considering an asymptotic solution to (4.25) in the limit of small σ. Expanding (4.25) for small σ gives

$$\frac{\mathrm{d}\sigma}{\mathrm{d}z} \sim \left[\frac{8\,\mathrm{Pe}^{-1}(1-a)}{2-a} + 2\frac{R'}{R}\right]\sigma\log\sigma. \tag{4.26}$$

(4.26) can be solved by separation and integration:

$$\begin{aligned}\int_{\sigma_0}^{\sigma}\frac{\mathrm{d}\varsigma}{\varsigma\log\varsigma} &\sim \int_{z_0}^{z}\left[\frac{8\,\mathrm{Pe}^{-1}(1-a)}{2-a} + 2\frac{R'}{R}\right]\mathrm{d}z \\ &= \int_{z_0}^{z}\frac{8\,\mathrm{Pe}^{-1}(1-a)}{2-a}\,\mathrm{d}z + 2\int_{R(z_0)}^{R(z)}\frac{\mathrm{d}P}{P}, \qquad (4.27a)\\ \log\left(\frac{\log\sigma}{\log\sigma_0}\right) &\sim \frac{8\,\mathrm{Pe}^{-1}(1-a)(z-z_0)}{2-a} + 2\log\left[\frac{R(z)}{R\,(z_0)}\right], \qquad (4.27b)\\ \sigma(z) &\sim \exp\left\{\log\sigma_0\left[\frac{R(z)}{R\,(z_0)}\right]^2\exp\left[\frac{8\,\mathrm{Pe}^{-1}(1-a)\,(z-z_0)}{2-a}\right]\right\}, \qquad (4.27c)\end{aligned}$$

where $\sigma\,(z_0) = \sigma_0$ corresponds to the point at which σ becomes "small." It is clear from (4.27c) that $\sigma > 0$ for any finite z. In terms of the physics of the system, this is a shortcoming of the model since it should be possible for the entire polymer to melt in finite time (and thus finite z).

There is one additional assumption in our application of the HBI method that was not discussed in the previous section. In (4.16), (4.10d) is used to give an expression for $\partial T_\mathrm{p}/\partial z$ at the melt front in terms of $\partial T_\mathrm{p}/\partial y$. This substitution requires (4.14) to satisfy (4.10d) at $y = \sigma(z)$. Unfortunately, this is not the case. This means that going from the original form of the Stefan condition in (4.13) to the alternate form in (4.19) with the assumed temperature profile in (4.14) alters the physical system being studied. This suggests that an assumed temperature profile other than that of (4.14) would be more suitable for the HBI method; however, finding such a profile is non-trivial and

is left for future work. Since the application of the HBI method with (4.14) gives an accurate description of the experimental data (see §4.6.3,4.7.3,4.8.3), it is reasonable to assume that the physical transformation that occurs when going from (4.13) to (4.19) is negligible.

4.3 Temperature Averages

For later convenience, we move forward by re-introducing the temperature averages discussed in the amorphous case. The cross-sectional average at a vertical coordinate z is given by

$$\langle T\rangle(z) = 2\int_0^1 T(y,z)y\,\mathrm{d}y = 2\int_{\sigma(z)}^1 T_\mathrm{p}(y,z)y\,\mathrm{d}y, \tag{4.28}$$

which relies on the fact that $T \equiv T_\mathrm{r} \equiv 0$ when $y \in [0,\sigma(z)]$. Using (4.14), we deduce

$$\langle T\rangle(z) = \alpha\left\{1 + \frac{2-a\left[1+\sigma^2(z)\right]}{2\log\sigma(z)} + \frac{(1-a)\left[1-\sigma^2(z)\right]}{2\left[\log\sigma(z)\right]^2}\right\}. \tag{4.29}$$

The cumulative cross-sectional temperature up to length z is given by

$$\frac{\mathrm{d}\bar{T}}{\mathrm{d}z} = \langle T\rangle(z), \quad \bar{T}(0) = 0. \tag{4.30}$$

In this case, we can compute $\bar{T}(z)$ with

$$\bar{T}(z) = 2\int_0^z\int_0^1 T(y,\zeta)y\,\mathrm{d}y\,\mathrm{d}\zeta = 2\int_0^z\int_{\sigma(\zeta)}^1 T_\mathrm{p}(y,\zeta)y\,\mathrm{d}y\,\mathrm{d}\zeta, \tag{4.31}$$

which again relies on the fact that $T \equiv T_\mathrm{r} \equiv 0$ when $r \in [0,\sigma(z)]$. Using (4.14), we deduce

$$\bar{T}(z) = \alpha + \alpha\int_0^z\left\{\frac{2-a\left[1+\sigma^2(\zeta)\right]}{2\log\sigma(\zeta)} + \frac{(1-a)\left[1-\sigma^2(\zeta)\right]}{2\left[\log\sigma(\zeta)\right]^2}\right\}\mathrm{d}\zeta. \tag{4.32}$$

Note that (4.32) is useful for comparison to (4.29) whereas (4.30) is more useful for computational purposes (see §4.5.3). The latter allows σ and $\bar{T}$ to be integrated simultaneously, thus saving computational expense compared to the former.

4.4 Asymptotics

Here we consider the asymptotic behavior of σ, T, $\langle T\rangle$, and $\bar{T}$ in the limits of small and large Pe to simplify later discussion. We will also consider the asymptotic behavior of σ in the limit of small α for the same end.

The limit of $\mathrm{Pe} \to 0$ corresponds to infinitesimally slow flow. This means the entire polymer will melt for any positive α. In this limit, $\sigma(z) \to 0$. This can be understood either by observing that, in limit of small Pe, the right-hand side of (4.25a) becomes large (and negative) corresponding to rapid decay of σ to zero or by taking the limit as Pe goes to zero of the right-hand side of (4.27c) directly. Furthermore, this behavior can be physically understood by realizing that slow flow gives the polymer time to heat to α throughout, which of course results in full melting. Substitution of this result into (4.14) gives

$$T_{\mathrm{p}}(y,z) \sim \alpha, \quad \mathrm{Pe} \to 0. \tag{4.33}$$

Since (4.29) and (4.32) both describe temperature averages, it follows that $\langle T\rangle(z) \sim \alpha$ and $\bar{T} \sim \alpha$ in this limit as well.

The limit of $\mathrm{Pe} \to \infty$ corresponds to infinitely fast flow. This means the polymer will be exposed to heat for only a short time and thus σ will remain close to one. Motivated by this, we assume $\sigma(z) \sim 1 - \mathrm{Pe}^{-m} f(z)$, where m is a positive constant to be determined and f is assumed to be small and independent of Pe. By substituting this melt profile into (4.25) and expanding about $f = 0$, we obtain, to leading order,

$$-\mathrm{Pe}^{-m}\frac{\mathrm{d}f}{\mathrm{d}z} = \frac{8(1-a)\,\mathrm{Pe}^{-(1+2m)}\,Rf^2}{-2(2+a)\,\mathrm{Pe}^{-3m}\,Rf^3/3}, \quad f(0) = 0. \tag{4.34}$$

To maintain the independence of f from Pe, we deduce that $m = 1/2$ in order for the Pe terms to cancel from both sides of (4.34). We solve (4.34) by separation and

integration:

$$\int_0^f \varphi \,\mathrm{d}\varphi = \int_0^z 8\frac{1-a}{2+a}\,\mathrm{d}\zeta, \tag{4.35a}$$

$$\frac{f^2}{2} = 12\frac{1-a}{2+a}z, \tag{4.35b}$$

$$\sigma(z) \sim 1 - \mathrm{Pe}^{-1/2} f(z), \quad f(z) = 2\sqrt{6\frac{1-a}{2+a}z}, \quad \mathrm{Pe} \to \infty, \tag{4.35c}$$

which relies on $f(z) \geq 0$ to maintain $\sigma(z) \leq 1$. To analyze the behavior of (4.29) in this limit, we can expand $\langle T \rangle$ about $\sigma = 1$ (since $\langle T \rangle$ is can be thought of as a function of z or σ) to obtain

$$\langle T \rangle(z) \sim \frac{\alpha(2+a)}{3}\,\mathrm{Pe}^{-1/2} f(z), \quad \mathrm{Pe} \to \infty, \tag{4.36}$$

where $\mathrm{d}\langle T \rangle / \mathrm{d}\sigma|_{\sigma=1} = -\alpha(2+a)/3$ explains the proportionality constant. This result can be understood by observing that $T_\mathrm{p} = O(1)$ and that the region of integration in (4.28) is $O(\mathrm{Pe}^{-1/2})$ by (4.35c). Also, it follows from (4.31) and (4.35c) that

$$\bar{T}(z) \sim \frac{2\alpha(2+a)}{3}\,\mathrm{Pe}^{-1/2}\,\bar{f}(z), \quad \mathrm{Pe} \to \infty. \tag{4.37}$$

The limit of small α corresponds to a heater temperature just above the melting point. As a result, we expect σ to remain close to one. Therefore, we assume $\sigma(z) \sim 1 - \alpha^n g(z)$, where n is a positive constant to be determined and g is assumed to be small and independent of α. By substituting this melt profile into (4.25) and expanding about $g = 0$, we obtain, to leading order,

$$-\alpha^n \frac{\mathrm{d}g}{\mathrm{d}z} = \frac{8(1-a)\alpha^{2n} R g^2}{-2(2+a)\alpha^{3n}\,\mathrm{Pe}\,Rg^3/3}, \quad g(0) = 0. \tag{4.38}$$

We must also use

$$a \sim 1 - \frac{\mathrm{St}\,\alpha}{2}, \quad \alpha \to 0, \tag{4.39}$$

which follows from (4.20). This reduces (4.38) to

$$-\alpha^n \frac{\mathrm{d}g}{\mathrm{d}z} = \frac{8\,\mathrm{St}\,\alpha^{2n+1} R g^2/2}{-6\alpha^{3n}\,\mathrm{Pe}\,Rg^3/3}, \quad g(0) = 0. \tag{4.40}$$

To maintain the independence of g from α, we deduce that $n = 1/2$ in order for the α terms to cancel from both sides of (4.40). We solve (4.40) by separation and integration:

$$\int_0^g \varphi \, \mathrm{d}\varphi = \int_0^z 2\frac{\mathrm{St}}{\mathrm{Pe}} \, \mathrm{d}\zeta, \tag{4.41a}$$

$$\frac{g^2}{2} = 2\frac{\mathrm{St}}{\mathrm{Pe}} z, \tag{4.41b}$$

$$\sigma(z) \sim 1 - \alpha^{1/2} g(z), \quad g(z) = 2\sqrt{\frac{\mathrm{St}}{\mathrm{Pe}} z}, \quad \alpha \to 0, \tag{4.41c}$$

which relies on $g(z) \geq 0$ to maintain $\sigma(z) \leq 1$. Observe that (4.41c) is the small-α limit of (4.35c).

4.5 Computational Considerations

Since we use numerical methods to solve (4.25) for $\sigma(z)$, numerical issues were sure to ensue. In particular, issues arise from the (lower and upper) bounds on the set of possible σ values given our initial condition. In numerical differential equation solvers, a value of σ is known at a particular value of z, which are used to find subsequent values of σ by finding a change $\Delta\sigma$ in σ by evaluating the right-hand side of (4.25a) at z and multiplying by Δz. When σ is close to a bound, a Δz value that is too large can cause $\sigma + \Delta\sigma$ to exceed said bound resulting in non-physical results. (Note the numerical integration techniques used in our calculations, namely MATLAB's `ode45` in the cylinder and tapered cases and `ode15s` in the combined case where the ODE is more stiff, are more sophisticated than the schema described here; however, the same principles are responsible for the numerical issues in these more advanced solvers.) In this section, we discuss the remedies of this issue at both small and large σ. We also discuss one technique used to improve the runtime of computing $\bar{T}$.

4.5.1 The Limit of Small σ

If σ is small, an overstep by the solver will give $\sigma < 0$. Due to the logarithms in (4.25), this results in complex σ values for subsequent z-steps. One solution to this issue is to introduce a trap that terminates the numerical integration once a complex

value of σ occurs. Storing the last real-valued (σ, z) as (σ_0, z_0), the asymptotic solution given by (4.27c) can be used to compute $\sigma(z)$ over the remaining interval ($z \in [z_0, 1]$ or $z \in [z_0, z_{\text{end}}]$ depending on the geometry).

However, a simpler solution exists. If we define $w(z) = \log \sigma(z)$, the resultant ODE for w depends on w and e^w, which cannot output complex values for real inputs, rather than $\log \sigma$ and σ. With this motivation, the ODE used in numerical calculations is

$$\frac{\mathrm{d}w}{\mathrm{d}z} = \frac{1}{\sigma}\frac{\mathrm{d}\sigma}{\mathrm{d}z} = \frac{8\,\mathrm{Pe}^{-1}\,\eta(z)e^w w}{2(1-a) + (2-a)w + e^{2w}\left[2aw^2 + (2-3a)w - 2(1-a)\right]}, \tag{4.42a}$$

$$\eta(z) = (1-a)w + \frac{\mathrm{Pe}}{4}\frac{R'}{R}\left\{(1-a)\left(1-e^{2w}\right) + \left[2 - a\left(1+e^{2w}\right)\right]w\right\}. \tag{4.42b}$$

4.5.2 The Limit of Large σ

If σ is large, an overstep by the solver will give $\sigma > 1$. This switches the signs of several terms in (4.25) culminating in an erroneous positive value for $\mathrm{d}\sigma/\,\mathrm{d}z$, which subsequently amplifies the issue. To understand this phenomenon, assume $\sigma(z) = 1 - h(z)$, where $0 < h(z) \ll 1$. Thus, expansion of (4.25) about $h = 0$ implies the leading order ODE for h is

$$\frac{\mathrm{d}h}{\mathrm{d}z} \sim 12\,\mathrm{Pe}^{-1}\,\frac{1-a}{2+a}h^{-1}. \tag{4.43}$$

This implies that

$$\frac{\mathrm{d}\sigma}{\mathrm{d}z} \sim -12\,\mathrm{Pe}^{-1}\,\frac{1-a}{2+a}(1-\sigma)^{-1}. \tag{4.44}$$

Recall that that $a \leq 1$. Therefore, $\mathrm{d}\sigma/\,\mathrm{d}z = -\,\mathrm{d}h/\,\mathrm{d}z \leq 0$, which explains why $\mathrm{d}\sigma/\,\mathrm{d}z$ should not be positive when σ is near one. In other words, even though σ may get close to one, it must start to decrease before $\sigma = 1$.

To rectify this issue, we return to the use of w, as defined in §4.5.1, since this is the variable implemented into code. We are interested in $w \sim 0$, as is the case when $\sigma \sim 1$. Thus, we expand (4.42) about $w = 0$. The resultant ODE is

$$\frac{\mathrm{d}w}{\mathrm{d}z} \sim 12\,\mathrm{Pe}^{-1}\,\frac{1-a}{2+a}w^{-1}. \tag{4.45}$$

We solve (4.45) by separation and integration:

$$\int_{w_0}^{w} \omega \, \mathrm{d}\omega = \int_{z_0}^{z} 12 \, \mathrm{Pe}^{-1} \frac{1-a}{2+a} \, \mathrm{d}\zeta, \tag{4.46a}$$

$$\frac{w^2 - w_0^2}{2} = 12 \, \mathrm{Pe}^{-1} \frac{1-a}{2+a} (z - z_0), \tag{4.46b}$$

$$w(z) \sim -\sqrt{w_0^2 + 24 \, \mathrm{Pe}^{-1} \frac{1-a}{2+a} (z - z_0)}, \tag{4.46c}$$

where the negative root has been taken to satisfy $w \leq 0$ and $w(z_0) = w_0$ corresponds to the point at which w becomes "small." To implement this solution, we introduce a trap that terminates the numerical integration once a positive value of w (i.e., $\sigma > 1$) occurs. Storing the last negative value of w as w_0 and its corresponding value of z as z_0, the asymptotic solution given by (4.46c) can be used to compute $w(z)$ until its absolute value is again large enough to resume numerical integration. This process is repeated until $z = 1$ or $z = z_{\mathrm{end}}$ depending on which geometry is being considered.

4.5.3 Expensive Computation

Unrelated to the issues with numerical solvers discussed previously, there are two concerns regarding computational expense that should be noted. First, consider the limit of small Pe. Observe from (4.25) and (4.42) that $\lim_{\mathrm{Pe}\to 0} \sigma'(z)$ and $\lim_{\mathrm{Pe}\to 0} w'(z)$ do not exist, respectively. Because of this behavior, the ODEs become stiff at exceedingly small Pe. Thus, the computation in this region becomes too expensive to visualize the small-Pe limit in figures. For this reason, the curves in the figures in this section that enter the limit of small Pe will stop before actually reaching $\mathrm{Pe} = 0$.

Second, the use of $\bar{T}(z)$ has the potential to be very computationally expensive. A naive approach would be to compute $\bar{T}(z)$ directly using (4.32). However, this would require the ODE for σ (or w rather) to be solved over $z \in [0, \ell \Delta z]$ for every $\ell \in \{1, 2, \ldots, z_{\mathrm{end}}/\Delta z\}$ for every set of parameters. (Note that we have assumed uniformly spaced quadrature nodes here to simplify discussion. The same principles

apply to the adaptive quadrature used in actual computation.) A better approach is to define a new variable ψ as follows:

$$\frac{\mathrm{d}\psi}{\mathrm{d}z} = \frac{2 - a\left(1 + e^{2w}\right)}{2w} + \frac{(1-a)\left(1 - e^{2w}\right)}{2w^2}, \quad \psi(0) = 0, \tag{4.47}$$

and solve the ODE for ψ simultaneously with (4.42) for any particular set of parameters. It follows that $\bar{T}(z)$ can be written as

$$\bar{T}(z) = \alpha + \alpha \int_0^z \frac{\mathrm{d}\psi}{\mathrm{d}\zeta}\,\mathrm{d}\zeta = \alpha + \alpha\psi(z). \tag{4.48}$$

We must also develop an expression to be used for ψ if the issue described in §4.5.2 arises. This simply becomes

$$\psi(z) = \psi_0 - \int_{z_0}^z \frac{\mathrm{d}\psi}{\mathrm{d}\zeta}(w(\zeta))\,\mathrm{d}\zeta, \tag{4.49}$$

where $\psi_0 = \psi\left(z_0\right)$ and z_0 and $w(\zeta)$ are as defined in (4.46c). As no analytical solution exists, (4.49) can be evaluated by numerical integration.

While w is useful numerically, it is not particularly useful for discussion of the physics of the system. Therefore, we will return to our use of σ, which has a more tangible physical interpretation, in subsequent sections.

4.6 The Cylindrical Case

4.6.1 Temperature Profile

We start our analysis of the crystalline case by considering flow through a right circular cylinder. In this case, the surface radius is given by a constant: $R(z) = 1$. It follows that $y = r$ and $\sigma = s$. Thus, the temperature profile in this case can be computed using (4.12) and (4.14) with $(y, \sigma) \equiv (r, s)$, where σ can be computed using (4.25). In this case, (4.25b) reduces to

$$\eta(z) = (1 - a)\log\sigma, \tag{4.50}$$

where a is given by (4.20). Note that we maintain the use of y and σ as opposed to r and s, respectively, in this section for ease of comparison to the other geometries.

Furthermore, this notation allows for the general asymptotic arguments made in this section to hold unchanged for the other geometries of interest. The temperature profile for the median datum is given in Figure 4.1. When considering the temperature profiles of the other geometries (see Figures 4.8 and 4.12), it will be useful to track to $R(z)$ as well; thus, we plot the temperature versus r instead of y in Figure 4.1 for ease of comparison.

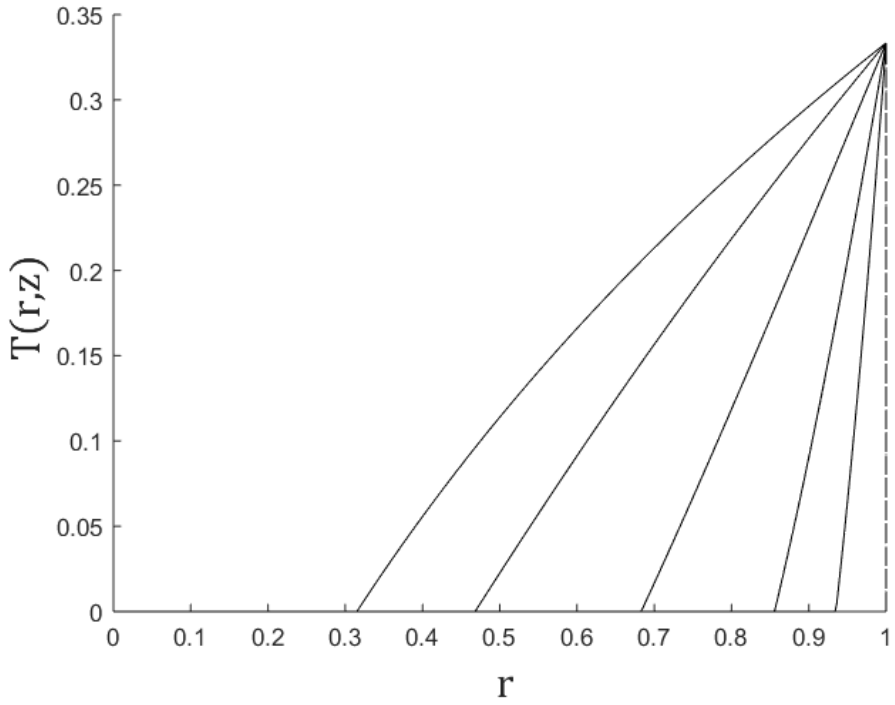


Figure 4.1: Plot of (4.12) and (4.14) with (4.20), (4.25a), and (4.50) (solid curves) for $z \in \{0.008, 0.04, 0.2, 0.6, 1\}$, where z increases from right to left, for the median experimental datum $(\alpha, \mathrm{Pe}) = (0.333, 3.94)$ and $R(z) = 1$ (dashed line). For each z, the temperature to the left of the r-intercept is zero.

4.6.2 Average Temperature as Threshold Condition

In finding threshold conditions, we proceed using the same average temperature conditions as in the amorphous case.

4.6.2.1 Cross-Sectional Average at Exit

First, we consider a threshold temperature defined by the cross-sectional average temperature at the exit of the extruder as defined in (3.27):

$$T_t \leq \langle T \rangle(1), \tag{4.51}$$

where $\langle T \rangle(z)$ is given by (4.29).

We will fit this condition to experimental data of $V_{\min}$ versus $T_{\max}$ for PLA taken from [8]. (Note that this data corresponds to relatively small values of α.) But before discussing the results of this fit, it should be noted that PLA is a semi-crystalline polymer. This means over certain temperature ranges, the polymer behaves amorphously. In particular, experimenters do not see the melt phase expected for crystalline polymers after extrusion. We expect that this is because the polymer is extruded with only a thin layer of melt on the outer surface, which rapidly cools to the amorphous-like, semi-crystalline state. This explanation is supported by (4.41c), which tells us that the melt front only achieves penetration depths of order $O(\alpha^{1/2})$ when alpha is small. This phenomena will be discussed further after Figures 4.8 and 4.12 and is sufficient justification for using the crystalline model for a semi-crystalline polymer such as PLA.

The experimental parameters used to scale the physical system (e.g., $(T_{\max}, V_{\min}) \to (\alpha, \mathrm{Pe})$) can be found in Appendix A. Given that $T_* = T_{\mathrm{m}} = 155°\mathrm{C}$, we use only that data for which $T_{\max} \geq 170°\mathrm{C}$ to ensure that melting has taken place; this decision is motivated by previous works [5, 6]. The results of this condition are shown in Figure 4.2. As will be the case for all figures in this section, the area under the curve corresponds to successful extrusion.

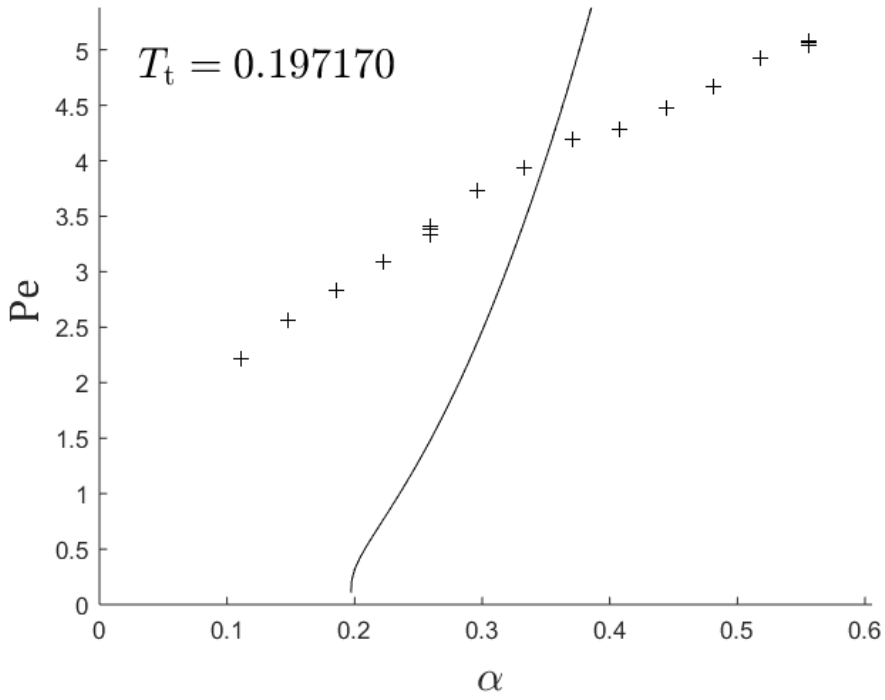


Figure 4.2: Plot of experimental data (crosses) and (4.51) (solid curve).

Figure 4.2 makes it clear that this threshold condition is unable to predict the experimental data. However, the figure does exhibit some important facets of the solution. First, observe that as $\mathrm{Pe} \to 0$, the solid curve approaches $\alpha = T_t$ as predicted in §4.4.

Second, as $\mathrm{Pe} \to \infty$, we replace the right-hand side (4.51) with (4.36) evaluated at $z = 1$, which gives,

$$T_t = \frac{\alpha(2+a)}{3}\,\mathrm{Pe}^{-1/2}\,f(1) \quad \Longrightarrow \quad \mathrm{Pe} = \left[\frac{(a+2)f(1)}{3T_t}\right]^2 \alpha^2, \quad \mathrm{Pe} \to \infty. \tag{4.52}$$

Physically, infinite Pe corresponds to infinitely fast feed speed. Thus, to maintain a temperature above the threshold, the heating temperature α must become infinite as well. Ergo, $\mathrm{Pe} \to \infty$ corresponds to $\alpha \to \infty$, which implies $a \to 0$ from (4.20). Thus, Pe versus α approaches a concave-up quadratic as Pe becomes large, as demonstrated in Figure 4.2.

Lastly, the fit in Figure 4.2 looks to defy the data to an unreasonable degree despite the nature of least squares regression. This is an amplification of the poor fit that results from the variables chosen to plot the data. Recall that we are fitting the data by minimizing $|T_\mathrm{t} \cdot \mathbf{1} - \langle T \rangle(1; \boldsymbol{\alpha}, \mathbf{Pe})|^2$, which is equivalent to finding a contour line of $\langle T \rangle(1; \alpha, \mathrm{Pe})$ that most closely matches the data in $(\langle T \rangle(1), \alpha, \mathrm{Pe})$-space. This is why the fit looks visually worse than if we had used a more traditional least squares approach, say minimizing $|\mathbf{Pe} - \mathrm{Pe}\,(\boldsymbol{\alpha}; T_\mathrm{t})|^2$, which is precluded by the complexity of the problem. This argument holds for all other visually poor fits.

4.6.2.2 Full Average

Next, we consider a threshold temperature defined by the average temperature across the entire cylinder as defined in (3.34) and (3.35):

$$T_t \leq \bar{T}(1), \tag{4.53a}$$

$$\frac{\mathrm{d}\bar{T}}{\mathrm{d}z} = \langle T \rangle(z), \quad \bar{T}(0) = 0. \tag{4.53b}$$

The results of this condition are shown in Figure 4.3.

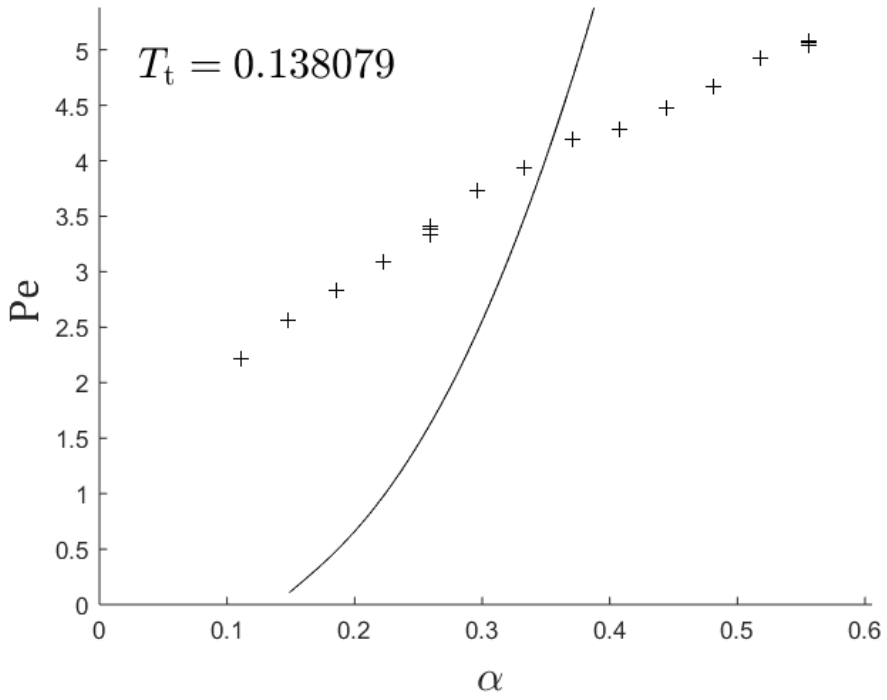


Figure 4.3: Plot of experimental data (crosses) and (4.53a) (solid curve).

Again, Figure 4.3 makes it clear that this threshold condition is unable to predict the experimental data. However, the figure does exhibit the features predicted in §4.4. First, observe that as $\mathrm{Pe} \to 0$, the solid curve goes to $\alpha = T_t$. Second, as $\mathrm{Pe} \to \infty$, the right-hand side (4.53a) goes to (4.37) evaluated at $z = 1$, which gives the same result as (4.52) except with $f(1)$ replaced $\bar{f}(1)$, which is still positive. Thus, a similar physical argument results in Pe versus α approaching a concave-up quadratic as Pe becomes large, as demonstrated in Figure 4.3.

Last, we again see that T_t is lower in the case of the full average than in the cross-sectional average. This is for the same reasons as discussed in §3.2.2.

4.6.3 Exit Temperature as Threshold Condition

Failure of the average temperature conditions in the crystalline case motivates the use of an exit temperature condition as in §3.3. However, we are unable to use the condition given in (3.49) since $T(0,1) = 0$ for any set of parameters, a result of

the quasistationary approximation and asymptotic behavior of (4.25). In analogy to (3.49), we use the following condition:

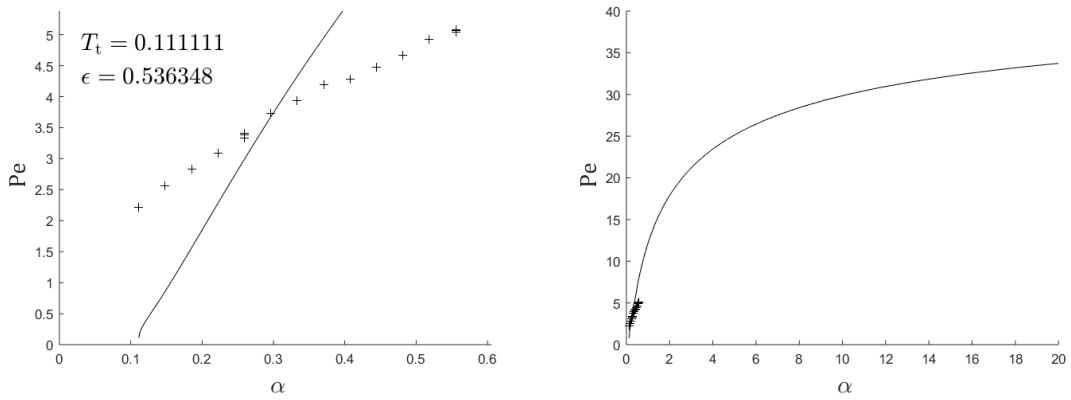
$$T_t \leq T(\epsilon, 1), \tag{4.54}$$

where $\epsilon > 0$ is a second fitting parameter to be determined. For emphasis, note that ϵ is a value of y with corresponding value of r given by $\epsilon R(1)$; of course, this distinction is moot in the case of the cylinder, but will be important in subsequent geometries. Using (4.14), we obtain

$$T_t \leq \alpha \left[a\chi + (1-a)\chi^2 \right], \quad \chi = 1 - \frac{\log \epsilon}{\log \sigma(1)}, \tag{4.55}$$

where χ has been defined for later convenience.

As a result of the quasistationary approximation, the crystalline solution is identically zero and the temperature is non-negative throughout the domain. In keeping with this theoretical consideration, it seems appropriate to restrict $T_\mathrm{t} \geq 0$, which forces $\epsilon \geq \sigma(1)$ for any experimental datum. The results of this condition and its asymptotic behavior are shown in Figure 4.4.



(a) (b)

Figure 4.4: (a) Plot of experimental data (crosses) and (4.54) with non-negative T_t (solid curve). (b) Asymptotic behavior of (4.54) with non-negative T_t.

As discussed in §4.4, the α-intercept in Figure 4.4a agrees with the fitted value of T_t. Unfortunately, we again see poor agreement between the model and the data;

however, unlike the average temperature conditions, this threshold condition appears to have negative concavity even at large Pe. This can be understood by considering the asymptotic behavior at large α. Expanding (4.20) about large α gives

$$a \sim \sqrt{\frac{2}{\mathrm{St}}}\alpha^{-1/2}, \quad \alpha \to \infty. \tag{4.56}$$

Substituting this result into (4.55) for large α gives

$$T_t \sim \alpha\left(\sqrt{\frac{2}{\mathrm{St}}}\alpha^{-1/2}\chi + \chi^2\right) = \sqrt{\frac{2}{\mathrm{St}}}\alpha^{1/2}\chi + \alpha\chi^2, \quad \alpha \to \infty, \tag{4.57}$$

which is a quadratic in $\alpha^{1/2}\chi$. Since $T_\mathrm{t} = O(1)$, we deduce that $\chi = O\left(\alpha^{-1/2}\right)$. It follows that

$$\sigma(1) \sim \epsilon\left(1 + c\alpha^{-1/2}\right), \quad \alpha \to \infty, \tag{4.58}$$

where c is a negative constant. This shows that $\sigma(1) \to \epsilon$ as $\alpha \to \infty$. If $\mathrm{Pe} \to \infty$ as $\alpha \to \infty$, then (4.35c) implies $\sigma(1) \to 1$ and thus $\epsilon \to 1$. This forces $T_\mathrm{t} \to 0$ for all parameter values. This means $\sigma(1)$ must remain bounded away from one and thus Pe must be bounded from above. Since Pe increases with increasing α, this means Pe must asymptote to a constant as $\alpha \to \infty$. It follows that the fit curve must be concave down as demonstrated in Figure 4.4b. Another argument for this behavior follows from the fact that $a \to 0$ as $\alpha \to \infty$. This means the leading order of (4.25) is independent of α in this limit. Therefore, $\sigma(1)$ asymptotes to a constant as $\alpha \to \infty$. Physically, this means that the melt front position becomes independent of the temperature at high enough temperatures, as is evident in Figure 4.4b.

Since the concavity does not force a poor fit in Figure 4.4, it forces us to seek other explanations. In coding this model, there were some difficulties with the sensitivity of the computed fitting parameters to their respective initial guesses. This indicates that several local minima could exist. This issue was remedied with the use of `MulitStart`, a built-in MATLAB object that is used to solve optimization problems over a range of initial guesses to find the global minimum. However, one local minimum of note was at $T_\mathrm{t} = 0$, which suggests that the true global minimum may occur for

negative T_t. This motivates the relaxation of the $T_\mathrm{t} \geq 0$ restriction. This could allow for the fitted value of ϵ to be less than $\sigma(1)$. Note that we still restrict $T_\mathrm{t} \geq -\mathrm{St}^{-1}/2$ to maintain $\chi \in \mathbb{R}$; as shown below, this restriction does not hinder the model to the same degree as $T_\mathrm{t} \geq 0$. The results of this condition, with the restriction removed, are shown in Figure 4.5.

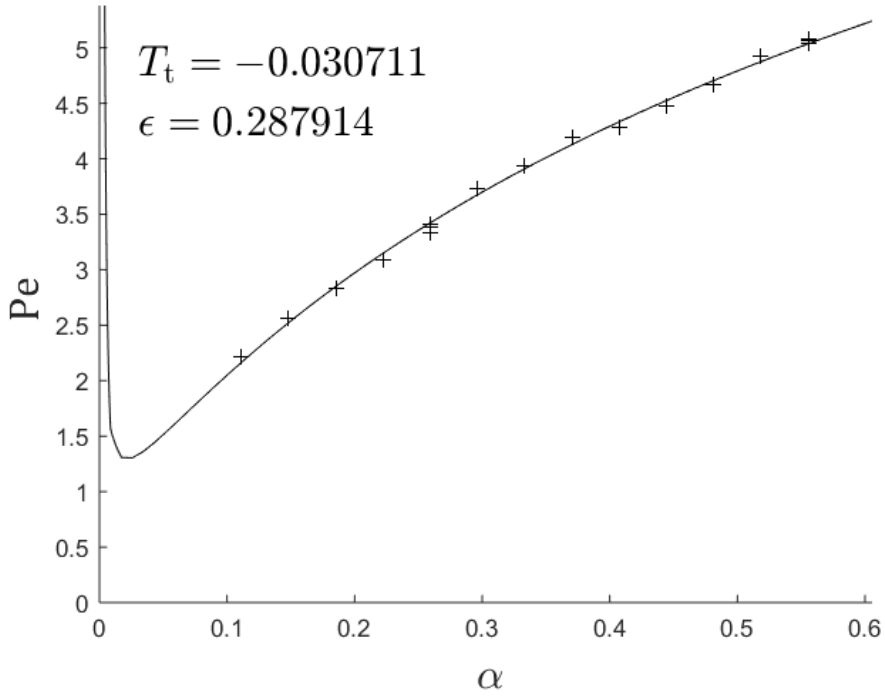


Figure 4.5: Plot of experimental data (crosses) and (4.54) with unrestricted T_t (solid curve).

With the restriction removed, there is significant improvement to the fit of the experimental data. Furthermore, the code consistently returned the same (negative) value of T_t, suggesting that this is indeed the global minimum. We are now tasked with justifying the use of a negative value of T_t. To understand what is happening in this computation, a temperature profile for an example set of parameters is given by Figure 4.6. This figure shows both the modeled temperature profile, i.e., with $T_\mathrm{r} \equiv 0$, and a temperature profile with the solution for T_p extended into the crystalline region.

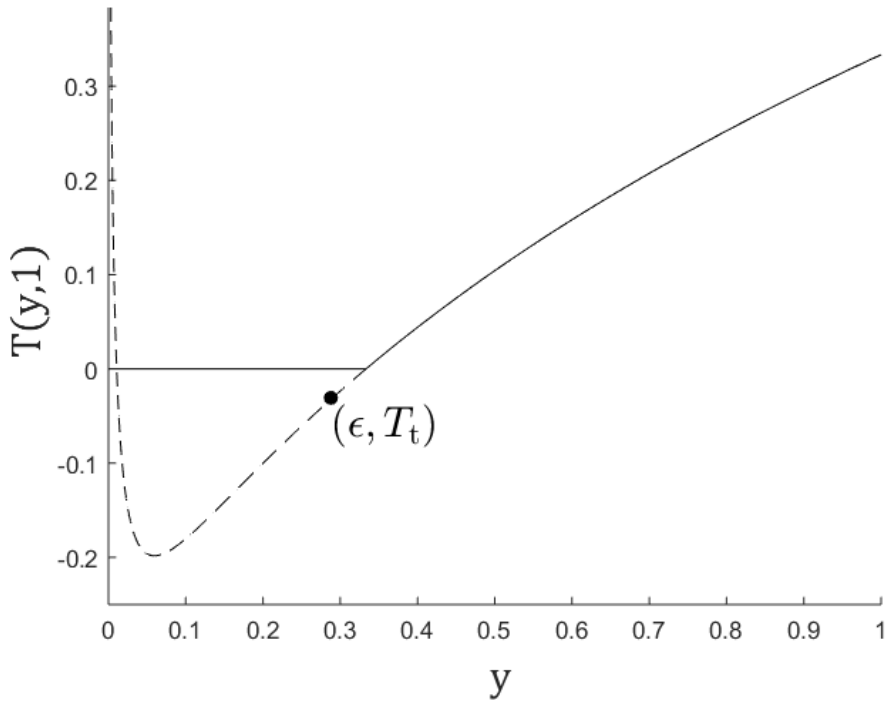


Figure 4.6: Plot of modelled temperature profile from (4.12) and (4.14) (solid curves) and extension of melt solution into the crystalline region (dashed curve) using the median experimental datum $(\alpha, \mathrm{Pe}) = (0.333, 3.94)$.

Figure 4.6 shows that, from a computational perspective, the code is merely imposing the threshold condition in the crystalline region by approximating the temperature profile with T_p for a small distance into the crystalline region. Given the success of the data fit, this suggests that the temperature profile in the crystalline region can be approximated well by extending the temperature profile from the melt region. Note that this approximation cannot be exact since there must be a discontinuity in the derivative of the temperature at the melt front to drive its evolution (see §4.1).

Another feature of note from Figure 4.6 is the minimum in the extended melt solution. This contradicts the intuition that the temperature should increase monotonically from the centerline to the outer surface. This would not normally concern

us since we truncate the the melt solution (4.14) with (4.12) at the melt front, but here we consider extending the melt solution into the crystalline region. Fortunately, ϵ does not extend past the monotonically increasing part of (4.14), so the existence of the minimum does not affect our analysis.

As discussed in §4.4, the right-hand side of (4.55) trends to a value of $\alpha > 0$; however, the left-hand side is now negative. Ergo, this inequality is strict in this limit. This results in a minimum in the fitted curve in Figure 4.5 since the curve corresponds to the set of (α, Pe) values where equality holds for (4.55). Physically, this means that for a slow enough flow velocity, extrusion will occur for any heating temperature. This results from the fact that the threshold temperature that the polymer must surpass is negative whereas $\alpha > 0$.

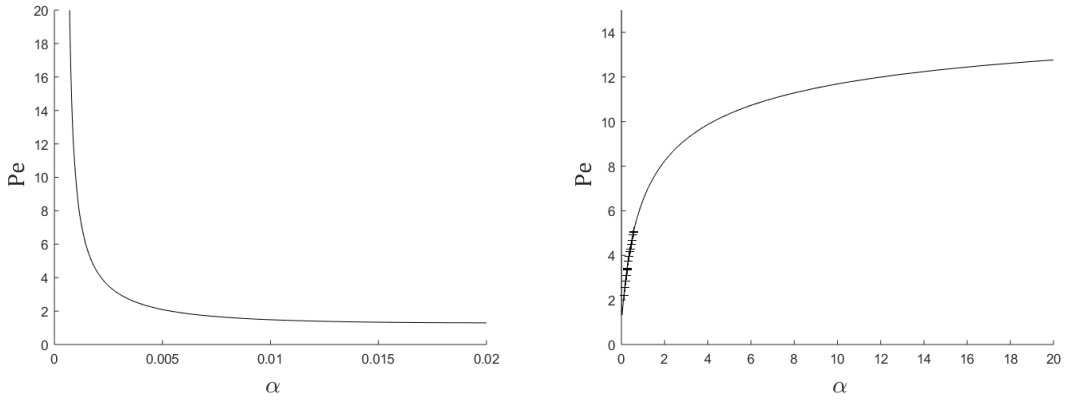
Using (4.39) in (4.55) gives

$$T_t \sim \alpha\chi + \frac{\mathrm{St}}{2}\alpha^2\chi^2, \quad \alpha \to \infty, \tag{4.59}$$

which is a quadratic in $\alpha\chi$. Again since $T_\mathrm{t} = O(1)$, we deduce that $\chi = O\left(\alpha^{-1}\right)$. This implies that $\sigma(1)$ must be very close to one. Using (4.41c) in (4.55) gives

$$\chi \sim \frac{\log\epsilon}{g(1)}\alpha^{-1/2} \propto -\mathrm{Pe}^{1/2}\,\alpha^{-1/2}, \quad \alpha \to \infty. \tag{4.60}$$

So, for $\chi = O\left(\alpha^{-1}\right)$ to be true, Pe must be proportional to α^{-1}. In other words, for small heating temperatures, $\sigma(1)$ must be very close to one for (4.55) to fail. This only occurs when both $\alpha \to 0$ and $\mathrm{Pe} \to \infty$. The asymptotic behavior for large α is similar to the case where T_t is restricted to be non-negative. The theory for small- and large-α asymptotics discussed here are corroborated by Figure 4.7.



(a) (b)

Figure 4.7: (a) Plot of asymptotic behavior of (4.54) at small α with unrestricted T_{t} (solid curve). (b) Plot of experimental data (crosses) and asymptotic behavior of (4.54) at large α with unrestricted T_{t} (solid curve).

Returning to our discussion of the small-α asymptotics, the model tells us that if the heating temperature is very close to the melting point, then almost any flow velocity will result in successful extrusion. This, of course, does not make sense from a physical perspective and thus the model cannot be used to extrapolate to arbitrarily small α. However, this failing is not cause to dismiss the model as it has been shown to be very successful when α is in the region where experimental data is collected and manufacturing processes operate.

4.7 The Tapered Case

4.7.1 Temperature Profile

We continue our analysis of the crystalline case by considering flow through a linear taper. As in the amorphous case, we consider the tapered problem to better understand the behavior in this portion of the hot end before moving onto the more realistic combined case. As in §3, we still consider a system of length H_{cyl} for comparison to §4.6 and still use the scaling $z = \tilde{z}/H_{\mathrm{cyl}}$. In this case, the surface radius is given

by a linear function:

$$R(z) = 1 - (1 - \beta)z. \tag{4.61}$$

The temperature profile in this case can be computed using (4.12) and (4.14), where σ can be computed using (4.25). In this case, (4.25b) reduces to

$$\eta(z) = (1 - a)\log\sigma - \frac{\mathrm{Pe}}{4}\frac{1-\beta}{1-(1-\beta)z}\left\{(1-a)\left(1-\sigma^2\right) + \left[2 - a\left(1+\sigma^2\right)\right]\log\sigma\right\}, \tag{4.62}$$

where a is given by (4.20). The temperature profile for the median datum is given in Figure 4.8.

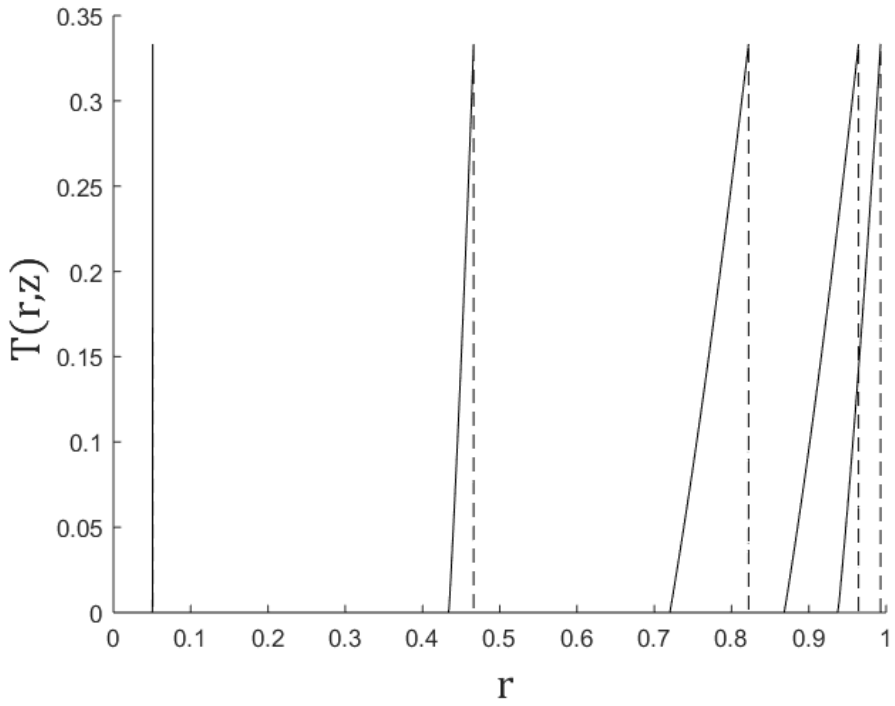


Figure 4.8: Plot of (4.12) and (4.14) with (4.20), (4.25a), and (4.62) (solid curves) for $z \in \{0.008, 0.04, 0.2, 0.6, 1\}$, where z increases from right to left, for the median experimental datum $(\alpha, \mathrm{Pe}) = (0.333, 3.94)$. For each z, the temperature to the left of the r-intercept is zero. The value of $R(z)$ given by (4.61) is indicated in each case by a dashed line to show the narrowing of the nozzle. (Note for $z = z_{\mathrm{end}}$, $R(z_{\mathrm{end}})$ is indistinguishable from $T(r, z_{\mathrm{end}})$.)

From Figure 4.8, we see that at first the heat penetration depth into the polymer increases, reaches a maximum, and subsequently decreases. This is likely the result of a crystalline core of some fixed radius less than one. Initially, there is further heat penetration because of the increased heating outside of the core. At some point, the edge of the nozzle gets closer to the core causing the range of heating to narrow and eventually go to zero. In other words, the melt front radius initially decreases faster than the surface radius, but after some z the surface radius decreases faster than the melt front radius causing the two to eventually intersect, leaving the unmelted crystalline core.

Note that Figure 4.8 tells us that heat is only able to penetrate a small distance into the polymer before extrusion. This supports the observation made in §4.6.2.1 that there should only be a thin layer of melt after extrusion.

4.7.2 Average Temperature as Threshold Condition

In finding threshold conditions, we proceed using the same average temperature conditions as in the previous cases.

4.7.2.1 Cross-Sectional Average at Exit

First, we consider a threshold temperature defined by the cross-sectional average temperature at the exit of the extruder as defined in (3.27):

$$T_t \leq \langle T \rangle(1), \tag{4.63}$$

where we compute $\langle T \rangle(z)$ with (4.29). The results of this condition are shown in Figure 4.9.

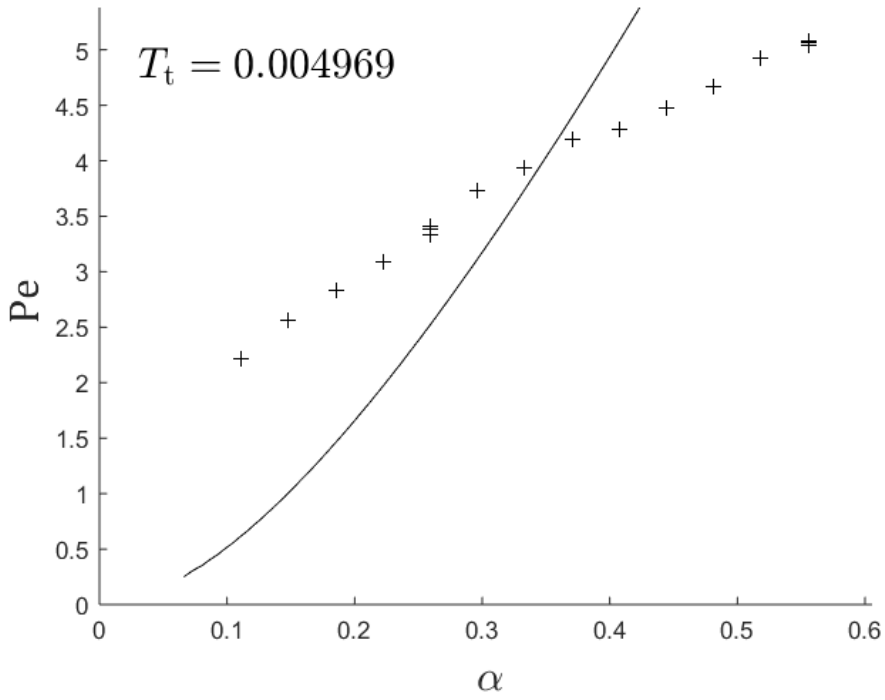


Figure 4.9: Plot of experimental data (crosses) and (4.63) (solid curve).

As in §4.6.2.1, the cross-sectional average as threshold temperature is unable to predict the experimental data. This can be explained using the same asymptotic arguments as before.

4.7.2.2 Full Average

Next, we consider a threshold temperature defined by the average temperature across the entire cylinder as defined in (3.34) and (3.35):

$$T_t \leq \bar{T}(1), \tag{4.64a}$$

$$\frac{\mathrm{d}\bar{T}}{\mathrm{d}z} = \langle T \rangle(z), \quad \bar{T}(0) = 0. \tag{4.64b}$$

The results of this condition are shown in Figure 4.10.

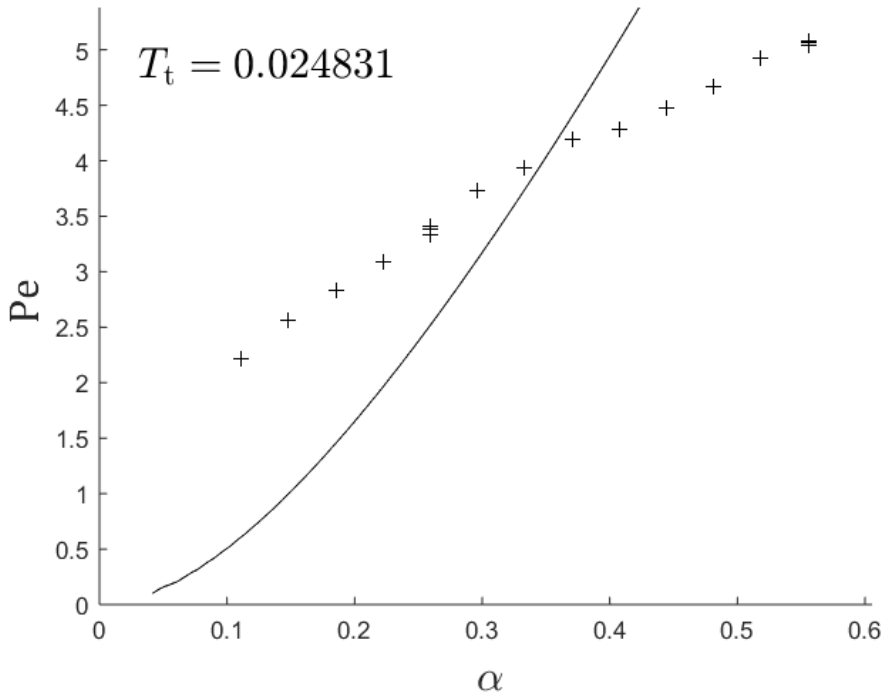


Figure 4.10: Plot of experimental data (crosses) and (4.64a) (solid curve).

As in §4.6.2.2, the full average as threshold temperature is unable to predict the experimental data. This can be explained using the same asymptotic arguments as before.

4.7.3 Exit Temperature as Threshold Condition

Failure of the average temperature conditions in the tapered crystalline case motivates the use of the exit temperature condition given by (4.54):

$$T_t \leq T(\epsilon, 1). \tag{4.65}$$

The results of this condition are shown in Figure 4.11.

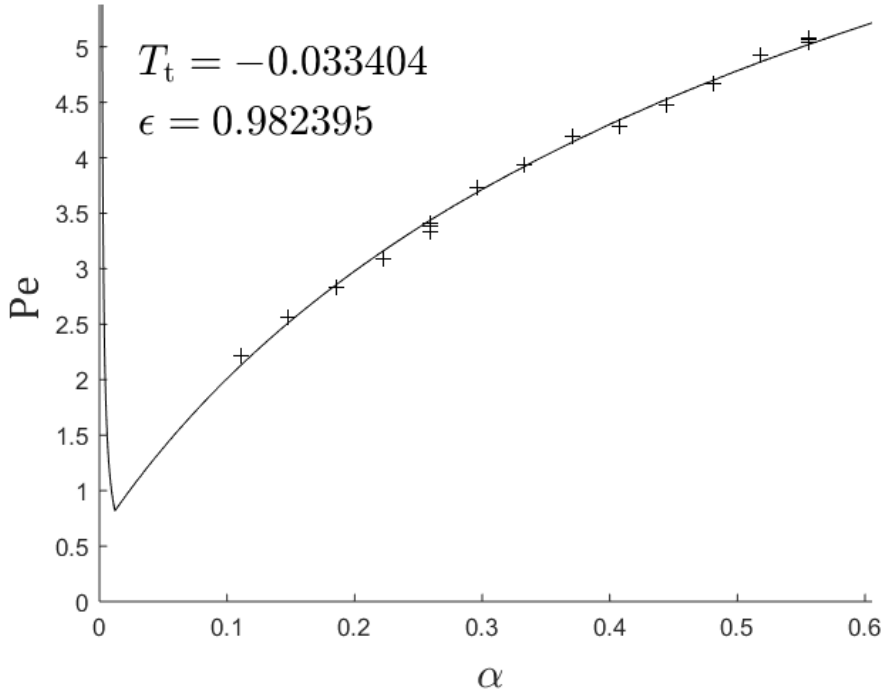


Figure 4.11: Plot of experimental data (crosses) and (4.65) (solid curve).

As in §4.6.3, the exit as threshold temperature is able to predict the experimental data. We again rely on the same justification for $T_t < 0$ as before. Note that the curve in Figure 4.11 appears to have a cusp at the minimum; however, the curve is indeed smooth as can be shown with mesh refinement.

4.8 The Combined Case

4.8.1 Temperature Profile

Lastly, we analyze the crystalline case by considering flow through the real geometry: a right cylinder feeding into a linear taper as in Figure 2.1. In this case, the surface radius is given by a piecewise linear function:

$$R(z) = 1 - B(z)(z-1), \tag{4.66a}$$

$$B(z) = \begin{cases} 0, & 0 \le z \le 1, \\ \frac{1-\beta}{z_{\text{end}}-1}, & 1 < z \le z_{\text{end}}. \end{cases} \tag{4.66b}$$

The temperature profile in this case can be computed using (4.12) and (4.14), where σ can be computed using (4.25). In this case, (4.25b) reduces to

$$\eta(z) = (1-a)\log\sigma - \frac{\mathrm{Pe}}{4}\frac{B(z)}{1-B(z)(z-1)}\left\{(1-a)\left(1-\sigma^2\right) + \left[2-a\left(1+\sigma^2\right)\right]\log\sigma\right\}, \tag{4.67}$$

where a is given by (4.20). The temperature profile for the median datum is given in Figure 4.8.

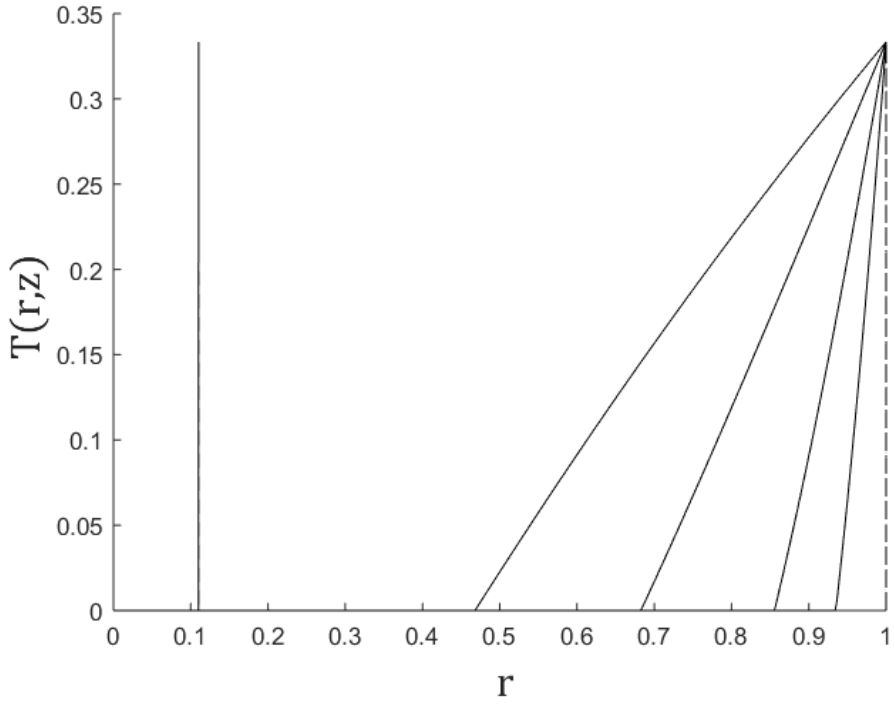


Figure 4.12: Plot of (4.12) and (4.14) with (4.20), (4.25a), and (4.67) (solid curves) for $z \in \{0.008, 0.04, 0.2, 0.6, z_{\mathrm{end}}\}$, where z increases from right to left, for the median experimental datum $(\alpha, \mathrm{Pe}) = (0.333, 3.94)$. For each z, the temperature to the left of the r-intercept is zero. The dashed line shows the width of the cylinder for the first four curves; at $z = z_{\mathrm{end}}$, the width of the nozzle $R(z_{\mathrm{end}})$ is indistinguishable from the curve $T(r, z_{\mathrm{end}})$.

Observe that Figure 4.12 looks much like Figure 4.1 for $z < 1$. For $z > 1$, we see the same phenomena as in Figure 4.8. In particular, note that Figure 4.12 shows

that the heat penetration depth and thus melt front depth will be small at extrusion, further corroborating the observation made in §4.6.2.1 that there should only be a thin layer of melt after extrusion.

4.8.2 Average Temperature as Threshold Condition

In finding threshold conditions, we proceed using the same average temperature conditions as in the previous cases.

4.8.2.1 Cross-Sectional Average at Exit

First, we consider a threshold temperature defined by the cross-sectional average temperature at the exit of the extruder as defined in (3.27):

$$T_t \leq \langle T \rangle (z_{\text{end}}), \tag{4.68}$$

where we compute $\langle T \rangle(z)$ with (4.29). The results of this condition are shown in Figure 4.13.

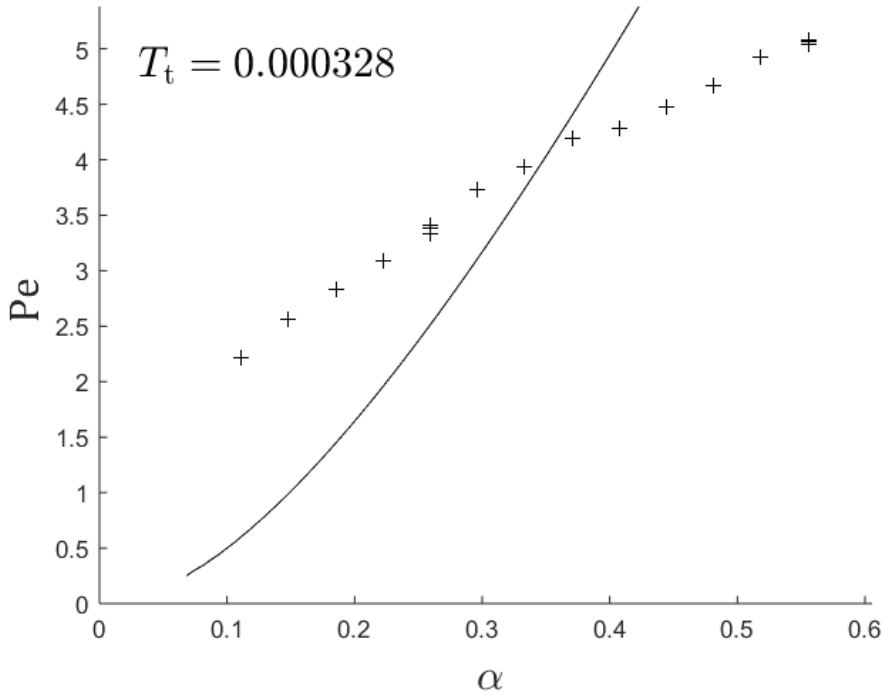


Figure 4.13: Plot of experimental data (crosses) and (4.68) (solid curve).

Due to the failure of the cross-sectional average as threshold temperature in §4.6.2.1 and §4.7.2.1, it is no surprise that the same asymptotic limitations exist in the combined case.

4.8.2.2 Full Average

Next, we consider a threshold temperature defined by the average temperature across the entire cylinder as defined in (3.34) and (3.35):

$$T_t \leq \bar{T}\left(z_{\text{end}}\right), \tag{4.69a}$$

$$\frac{\mathrm{d}\bar{T}}{\mathrm{d}z} = \langle T \rangle(z), \quad \bar{T}(0) = 0. \tag{4.69b}$$

The results of this condition are shown in Figure 4.14.

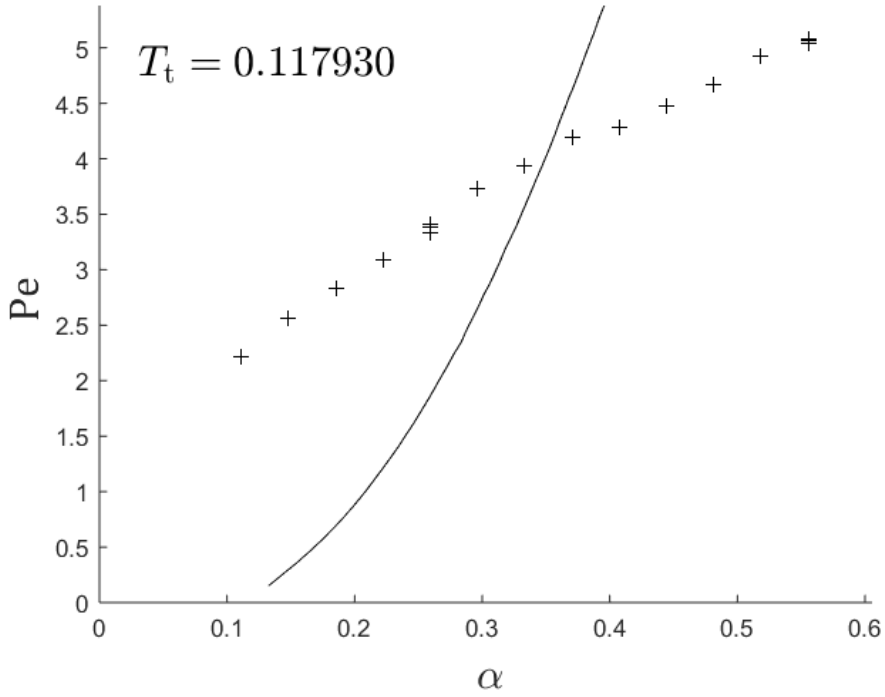


Figure 4.14: Plot of experimental data (crosses) and (4.69a) (solid curves).

Similarly, the failure of the full average as threshold temperature in §4.6.2.2 and §4.7.2.2 and the asymptotic limitations therein explain the poor fit here as well.

4.8.3 Exit Temperature as Threshold Condition

Failure of the average temperature conditions in the tapered crystalline case motivates the use of the exit temperature condition given by (4.54):

$$T_t \leq T\left(\epsilon, z_{\text{end}}\right). \tag{4.70}$$

The results of this condition are shown in Figure 4.15.

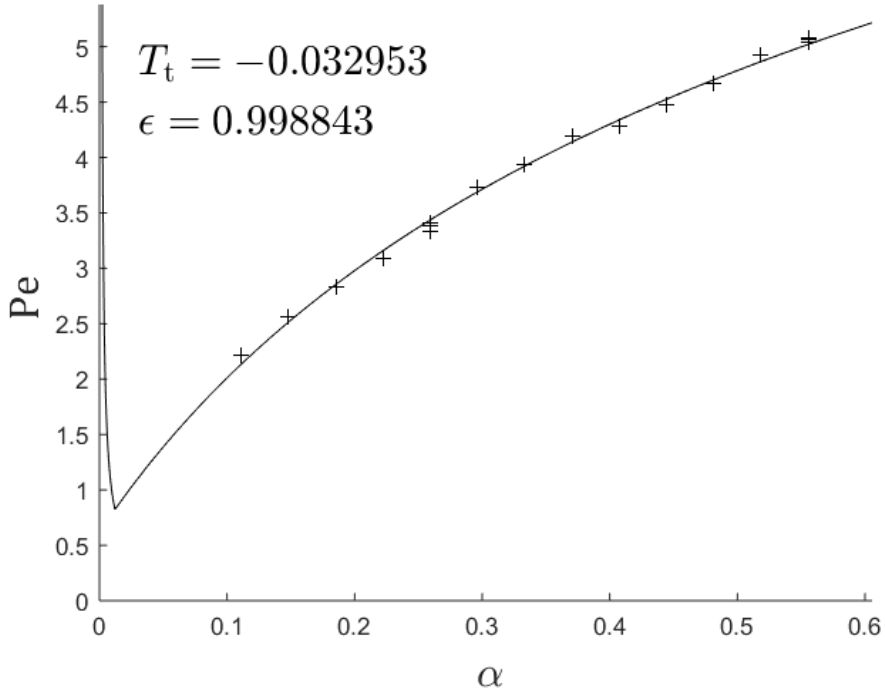


Figure 4.15: Plot of experimental data (crosses) and (4.70) (solid curve).

As expected from the analogous results of §4.6.3 and §4.7.3, the exit as threshold temperature is able to predict the experimental data. We again rely on the same justification for $T_t < 0$ as before. Also note that the curve in Figure 4.15 is indeed smooth at the minimum, as in Figure 4.11. Similar asymptotic arguments also apply except with $\chi = [1 - \log\epsilon/\log\sigma(1)]$ replaced with $[1 - \log\epsilon/\log\sigma\left(z_{\text{end}}\right)]$.

Given the three successful models, this one is the most faithful to the system of interest; however, this accuracy comes with computational expense. The figures in

sections §4.6 and §4.7 all took between 15 and 25 seconds to construct on a 2.60 GHz processor, with the tapered models taking slightly more time than the cylindrical ones for any condition. The figures in this section all took over 80 seconds to construct. This time difference is a result of the ODE stiffness in the combined case. While the tapered and combined problems are academically interesting, their lack of added accuracy means the cylindrical model is likely the most useful in engineering applications.

Chapter 5

CONCLUSIONS AND AREAS OF FUTURE RESEARCH

In order to optimize 3-D printing production processes, engineers must first understand the maximum rate at which the polymer construction material can be extruded. This quantity is useful in improving product quality and product processing time. The polymer feedstock is heated as it moves through the hot end of the printer and increases in pliancy throughout. The inverse relationship between flow velocity and heating time naturally gives way to an upper bound on the flow velocity. Above this velocity, the polymer does not become sufficiently pliant, thus increasing the required pressure beyond what the pump can feasibly produce. This causes the printer to jam and extrusion to fail.

After applying several simplifying assumptions justified by the physical systems of interest, we modeled the system using the heat equation in a moving cylindrical coordinate system. We first considered the flow of amorphous polymers in a taper, which is a relatively simple boundary value problem. We then considered crystalline polymers in three geometries: a cylinder, a taper, and a cylinder feeding into a taper. Since crystalline polymers exhibit a crystal-melt transition, this is a type of moving boundary problem known as a Stefan problem, where the moving boundary in this case is the melt front $s(z)$.

After several geometric transformations, the amorphous problem is solvable using a standard separation of variables approach. This results in two ODEs, one for each space variable, given by (3.7). The separable solution is a series solution given by (3.1) with (3.24).

Again relying on geometric transformations, the crystalline problem was solved using a quasistationary approach in the crystalline region and the HBI method in the

melt region. These approximations gave an expression for the temperature field, as in (4.12), (4.14), and (4.20), that is coupled to the ODE for the melt front (4.25). This ODE must be solved numerically, but (4.27c) and (4.46c) give an analytical approximations that can be used in the limits of small and large normalized melt front radius, respectively.

Given a solution for the temperature field, we were tasked with translating the solution into a velocity upper bound. We examined three temperature-based conditions. Temperature was chosen as a more readily calculable alternative to viscosity, the material property that is expected to more directly affect extrusion success/failure. Success in the case of amorphous polymers in a cylinder in [5] motivated imposing a constraint on the cross-sectional average of the exit temperature. As shown in Figures 3.3, 4.2, 4.9, and 4.13, this condition was unable to predict the data. In amorphous polymers, this is likely due to the large cylindrical portion of the hot end that is neglected combined with the use of a curved taper instead of a linear one. In crystalline polymers, this is due to the wrong concavity for large values of Pe. Since the overall shape of the curve did not match the data, the introduction of additional degrees of freedom is unlikely to improve the fit.

Moving forward, a similar constraint using the average temperature throughout the hot end was tested. This condition was again successful for amorphous polymers in a cylinder in [5]. As shown in Figures 3.4, 4.3, 4.10, and 4.14, this condition was also unable to predict the data. This is due to similar reasons as the failure of the cross-sectional average condition.

The last condition considered for amorphous polymers was based on the centerline temperature at the exit, which again failed to predict the data as shown in Figure 3.5. We are unable to impose this condition on crystalline polymers because in this case the centerline temperature at the exit will allays be zero for any set of parameters. This is a a result of the quasistationary approximation, which tells us that the temperature in the crystalline region is zero, and the asymptotic behavior of the melt front near the centerline, which tells us that the melt front will not go to zero in finite z.

An analogous constraint is considered with an additional degree of freedom: a distance ϵ from the centerline. From a mathematical perspective, we expect the fitted threshold temperature T_t to be non-negative since the crystalline temperature is identically zero (again an artifact of the quasistationary approximation). However, this proved unsuccessful in the cylindrical case as shown in Figure 4.4a.

Combined with some numerical observations, the failure of the fit for non-negative T_t motivated the relaxation of this restriction to allow $T_\mathrm{t} < 0$. The results of these fits, as given in Figures 4.5, 4.11, and 4.15, were quite good. Thus, the model works in the region where the experimental data was collected. However, the negative threshold temperature resulted in some non-physical asymptotic behavior in the limit of small α. In particular, we expect an α-intercept to exist, as shown in the other figures. The use of negative T_t is justified in Figure 4.6. Negative T_t arises from an extension of the melt solution for a small distance into the crystalline region. For moderate values of (α, Pe), the fitted extension aligns well with the solution profile. Outside those regimes, the extension does not match the profile, explaining the spurious results. That being said, this model remains useful as demonstrated by its correct behavior in the range where experimental data is collected and manufacturing processes operate.

One clear solution to improve the results for small α would be to eliminate the quasistationary approximation and use a more refined HBI method to consider the full two-phase Stefan problem. This, of course, would increase the complexity of the problem significantly. Authors tend avoid the use of cylindrical coordinates [16] or use a semi-discretized approach [25] due to the potential for singularities to occur along the centerline. A more robust solution for the crystalline region would necessitate a Taylor series in z, among other complications, due to the form of its boundary conditions. A solution of the form in [26], an exponential approach, may be able to resolve the issues regarding the initial condition that arise from a Taylor series or related functional forms. Another approach that may retain some simplicity and provide some accuracy is a Megerlin method in which we assume the heat equation is satisfied along the melt front [14]. Alternatively, the two-phase problem could be solved using the HBI method

via a fully numerical approach. These considerations are left as subjects for future research.

The crystalline model could also be improved by considering different functional forms for the temperature profile in Step 1 of the HBI method. As discussed in §4.2.3, imposing the Stefan condition on the assumed temperature profile in Step 2 of the HBI method required the heat equation to be satisfied along the melt front. While the assumed temperature profile was able to predict the experimental data, it does not satisfy the heat equation along the melt front. This means we have modeled a slightly different physical system. Thus, finding a functional form that does satisfy the heat equation along the melt front could provide additional physical insights.

For the amorphous case, the model could be improved by considering the more physically realistic geometry of a cylinder feeding into a taper. In the hot end, the cylindrical portion is much longer than the tapered portion. The success of [5] in modeling amorphous polymers using just the cylindrical portion suggests that a combined model would be much more successful than a purely tapered model.

Additional refinements for both the amorphous and crystalline polymers could come from the relaxation of the assumptions listed in §2. Furthermore, additional threshold conditions could be considered, such as the viscosity-based conditions used in [5].

In conclusion, our results show the behavior of amorphous polymers cannot be predicted with a temperature-based constraint without consideration of the cylindrical portion of the hot end whereas crystalline polymers are predicted well when using an exit temperature constraint in several relevant geometries. In particular, given a desired velocity of crystalline polymer flow, our solution is able to provide the heating temperature necessary for successful extrusion, even if beyond the range of commercially available equipment. As shown in Figures 4.5, 4.11, and 4.15, there are diminishing returns on the velocity bound from temperature increases. This agrees with physical intuition since heat takes time to diffuse through the polymer making the bound unreachable at higher speeds. This insight will help engineers to design better and more

productive 3-D printers.

REFERENCES

[1] I. Gibson, D. W. Rosen, and B. Stucker, *Additive Manufacturing Technologies: Rapid Prototyping to Direct Digital Manufacturing*. New York, NY: Springer Publishing Company Incorporated, 1 ed., 2009.

[2] K. G. Jaya Christiyan, U. Chandrasekhar, and K. Venkateswarlu, "A study on the influence of process parameters on the mechanical properties of 3D printed ABS composite," *IOP Conference Series: Materials Science and Engineering*, vol. 114, p. 012109, 2016.

[3] W. T.-M., J.-T. Xi, and Y. Jin, "A model research for prototype warp deformation in the FDM process," *The International Journal of Advanced Manufacturing Technology*, vol. 33, no. 11-12, pp. 1087–1096, 2007.

[4] Q. Sun, G. M. Rizvi, C. T. Bellehumeur, and P. Gu, "Effect of processing conditions on the bonding quality of FDM polymer filaments," *Rapid Prototyping Journal*, vol. 14, no. 2, pp. 72–80, 2008.

[5] D. A. Edwards, M. E. Mackay, Z. R. Swain, C. R. Banbury, and D. D. Phan, "Maximal 3D printing extrusion rates," *IMA Journal of Applied Mathematics*, vol. 84, no. 5, pp. 1022–1043, 2019.

[6] J. W. Sitison and D. A. Edwards, "The heat balance integral method for cylindrical extruders." *Journal of Engineering Mathematics* (in press), 2020.

[7] J. F. Agassant, D. R. Arda, C. Combeaud, A. Merten, H. Munstedt, M. R. Mackley, L. Robert, and B. Vergnes, "Polymer processing extrusion instabilities and methods for their elimination or minimisation," *International Polymer Processing*, vol. 21, no. 3, pp. 239–255, 2006.

[8] M. E. Mackay, Z. R. Swain, C. R. Banbury, D. D. Phan, and D. A. Edwards, "The performance of the hot end in plasticating 3D printing," *Journal of Rheology*, vol. 61, no. 2, pp. 229–236, 2017.

[9] F. Lotero, F. Couenne, B. Maschke, and D. Sbarbaro, "Distributed parameter bi-zone model with moving interface of an extrusion process and experimental validation," *Mathematical and Computer Modelling of Dynamical Systems*, vol. 22, no. 5, pp. 504–522, 2017.

[10] Y. Mu, G. Zhao, X. Wu, L. Hang, and H. Chu, "Continuous modeling and simulation of flow-swell-crystallization behaviors of viscoelastic polymer melts in the hollow profile extrusion process," *Applied Mathematical Modelling*, vol. 39, no. 3-4, pp. 1352–1368, 2015.

[11] J. L. Sandoval Murillo and G. C. Ganzenmueller, "A convergence analysis of the affine particle-in-cell method and its application in the simulation of extrusion processes," in *V International Conference on Particle-Based Methods - Fundamentals and Applications (Particles 2017)* (P. Wriggers, M. Bischoff, E. Onate, D. R. J. Owen, and T. Zohdi, eds.), p. 397–408, European Community on Computational Methods in Applied Sciences; International Association for Computational Mechanics, 2017.

[12] B. Schoinochoritis, D. Chantzis, and K. Salonitis, "Simulation of metallic powder bed additive manufacturing processes with the finite element method: a critical review," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 231, no. 1, pp. 96–117, 2017.

[13] *NIST Digital Library of Mathematical Functions*. http://dlmf.nist.gov/, Release 1.0.26 of 2020-03-15. F. W. J. Olver, A. B. Olde Daalhuis, D. W. Lozier, B. I. Schneider, R. F. Boisvert, C. W. Clark, B. R. Miller, B. V. Saunders, H. S. Cohl, and M. A. McClain, eds.

[14] V. Alexiades and A. D. Solomon, *Mathematical Modeling of Melting and Freezing Processes*, ch. 3.4. Washington, DC: Taylor & Francis, 1992.

[15] T. R. Goodman, "The heat balance integral and its application to problems involving a change of phase," *Transactions of ASME Journal of Heat Transfer*, vol. 80, pp. 335–342, 1958.

[16] H. S. Ren, "Application of the heat-balance integral to an inverse Stefan problem," *International Journal of Thermal Sciences*, vol. 46, no. 2, pp. 118–127, 2007.

[17] T. R. Goodman, "The heat-balance integral—further considerations and refinements," *Journal of Heat Transfer*, vol. 83, no. 1, pp. 83–85, 1961.

[18] S. L. Mitchell and T. G. Myers, "Heat balance integral method for one-dimensional finite ablation," *Journal of Thermophysics and Heat Transfer*, vol. 22, no. 3, pp. 508–514, 2008.

[19] S. L. Mitchell and T. G. Myers, "Applications of standard and refined heat balance integral methods to one-dimensional Stefan problems," *SIAM Review*, vol. 52, no. 1, pp. 57–86, 2010.

[20] T. G. Myers, S. L. Mitchell, G. Muchatibaya, and M. Y. Myers, "A cubic heat balance integral method for one-dimensional melting of a finite thickness layer,"

International Journal of Heat and Mass Transfer, vol. 50, no. 25-26, pp. 5305–5317, 2007.

[21] T. R. Goodman and J. J. Shea, "The melting of finite slabs," *Journal of Applied Mechanics*, vol. 27, pp. 16–24, 1960.

[22] A. S. Wood, "A new look at the heat balance integral method," *Applied Mathematical Modelling*, vol. 25, no. 10, pp. 815–824, 2001.

[23] H. Schlichting and K. Gersten, *Boundary-Layer Theory*. New York: Springer, 8 ed., 2000.

[24] T. R. Goodman, "Applications of integral methods in transient non-linear heat transfer," in *Advances in Heat Transfer* (T. F. Irvine and J. P. Hartnett, eds.), vol. 1, (New York), pp. 51–122, Academic Press, 1964.

[25] J. Caldwell and C. Chin, "Solution of two-phase Stefan problems by the heat balance integral method," in *Mathematics of Heat Transfer* (G. E. Tupholme and A. S. Wood, eds.), 66, pp. 131–138, Institute of Mathematics and its Applications Conference Series, 1998.

[26] F. Mosally, A. S. Wood, and A. Al-Fhaid, "An exponential heat balance integral method," *Applied Mathematics and Computation*, vol. 130, no. 1, pp. 87–100, 2002.

[27] M. Pyda, R. C. Bopp, and B. Wunderlich, "Heat capacity of poly(lactic acid)," *Journal of Chemical Thermodynamics*, vol. 36, no. 9, pp. 731–742, 2004.

Appendix A

EXPERIMENTAL PARAMETERS

In this appendix, we list the experimental parameter values used for computation. Some values come from literature sources, whereas others come directly from laboratory measurements. Other parameters calculated from these experimental values are also listed.

In Table A.1, we list the parameters for the hot end. As discussed in §2, $\varepsilon = O\left(10^{-2}\right)$ is small, thus validating Assumption 2.

Table A.1: Device parameters.

	[5]	[8]	Measured	Calculated
H_{cyl} (mm)	30	—	—	—
H_{noz} (mm)	—	—	2	—
R_{max} (mm)	—	1.5875	—	—
R_{min} (mm)	—	—	0.175	—
T_{i} (°C)	20	—	—	—
β	—	—	—	0.1102
γ	—	—	—	2.2051

In Table A.2, we list parameters for the ABS polymer, the polymer used in the amorphous case §3. As expected, the experimental data lies in the regime of relatively small Pe.

Table A.2: ABS parameters.

	[5]	[8]	Calculated
c_P [J/(kg·K)]	—	2100	—
k [W/(m·K)]	0.205	—	—
Pe	—	—	$[0.218, 3.26]$
$T_{\max}$ (°C)	—	$[175, 245]$	—
$\tilde{T}_{\mathrm{t}}$ (°C)	—	—	$[98, 173]$
T_{t}	—	—	$[-0.0239, 0.908]$
$T_* = T_{\mathrm{g}}$ (°C)	—	100	—
$V_{\min}$ (mm/s)	—	$[0.23, 3.44]$	—
α	—	—	$[0.938, 1.81]$
ΔT (K)	—	—	80
ρ (kg/m^3)	1100	—	—

In Table A.3, we list parameters for the PLA polymer, the polymer used in the crystalline case §4. As noted in §4, this model only uses those experiments with high enough $T_{\max}$ values that we think we may be in the melt regime, i.e., those with $T_{\max}$ sufficiently greater than T_{m}. Here the experimental data lies in the regime of relatively small Pe and α.

Table A.3: PLA parameters.

	[8]	[27]	Calculated
a	—	—	$[0.678, 0.889]$
c_L (kJ/kg)	—	91	—
c_P [J/(kg·K)]	1700	—	—
k [W/(m·K)]	0.13	—	—
Pe	—	—	$[2.21, 5.08]$
St	—	—	2.52
$T_{\max}$ (°C)	$[170, 230]$	—	—
$\tilde{T}_{\mathrm{t}}$ (°C)	—	—	$[150, 182]$
T_{t}	—	—	$[-0.0396, 0.197]$
$T_* = T_{\mathrm{m}}$ (°C)	155	—	—
$V_{\min}$ (mm/s)	$[1.61, 3.70]$	—	—
α	—	—	$[0.111, 0.556]$
ΔT (K)	—	—	135
ϵ	—	—	$[0.288, 0.999]$
ρ (kg/m^3)	1250	—	—

Appendix B

AMORPHOUS MODEL IMPLEMENTATION

In this appendix, we display the MATLAB code used to implement the amorphous model from §3. First, we will list the equations referenced in the code. Some equations are in a different form than those given in the body whereas others are merely repeated here for ease of comparison to the code.

B.1 Variables

The following are variable definitions used in the main script in order of appearance:

$$\Delta T = T_* - T_i, \tag{B.1}$$

$$\beta = \frac{R_{\min}}{R_{\max}}, \tag{B.2}$$

$$\gamma = \log \beta^{-1}, \tag{B.3}$$

$$\lambda_{N+1} = \min_{n\in\mathbb{N}} \left\{ \lambda_{n+1} : D_{n+1} e^{-2\gamma\lambda_{n+1}} \leq 10^{-6} \right\}, \tag{B.4}$$

$$\alpha = \frac{T_{\max} - T_*}{\Delta T}, \tag{B.5}$$

$$\mathrm{Pe} = \frac{\rho c_P R_{\max}^2 V_{\min}}{k H_{\mathrm{cyl}}}. \tag{B.6}$$

B.2 Functions

Here we describe all the functions defined in the code in alphabetical order.

B.2.1 AlphaFitCross

`AlphaFitCross` defines the function to be curve fitted using the cross-sectional average temperature condition in `FitData`. `AlphaFitCross` returns $T_\mathrm{t} = \langle T \rangle(z)$ solved for α using (B.12) to evaluate $\langle T \rangle(z)$:

$$\alpha = \frac{T_\mathrm{t} + \langle \Theta \rangle(z)}{1 - \langle \Theta \rangle(z)}. \tag{B.7}$$

`AlphaFitCross` is called by `FitData` and calls `ThetaCross`.

B.2.2 AlphaFitExit

`AlphaFitExit` defines the function to be curve fitted using the exit temperature condition in `FitData`. `AlphaFitExit` returns $T_\mathrm{t} = T(x, z)$ solved for α using (B.13) to evaluate $T(x, z)$:

$$\alpha = \frac{T_\mathrm{t} + \Theta(x, z)}{1 - \Theta(x, z)}. \tag{B.8}$$

`AlphaFitExit` is called by `FitData` and calls `ThetaExit`.

B.2.3 AlphaFitFull

`AlphaFitFull` defines the function to be curve fitted using the full average temperature condition in `FitData`. `AlphaFitFull` returns $T_\mathrm{t} = \bar{T}(z)$ solved for α using (B.14) to evaluate $\bar{T}(z)$:

$$\alpha = \frac{T_\mathrm{t} + \bar{\Theta}(z)}{1 - \bar{\Theta}(z)}. \tag{B.9}$$

`AlphaFitFull` is called by `FitData` and calls `ThetaFull`.

B.2.4 Dn

`Dn` defines the coefficient D_n from (3.23):

$$D_n = \frac{M\left(1 - \lambda_n, 1, \frac{\gamma\,\mathrm{Pe}}{2}\right) e^{-\gamma\,\mathrm{Pe}/2}}{\int_0^{\gamma\,\mathrm{Pe}/2} M^2\left(-\lambda_n, 1, x\right) e^{-x}\,\mathrm{d}x}. \tag{B.10}$$

`Dn` is called in `ThetaCross`, `ThetaExit`, and `ThetaFull` and calls `M`.

B.2.5 eVals

`eVals` computes a vector of eigenvalues using the eigenvalue condition (3.13):

$$\lambda_n : \ n\text{-th smallest } \lambda \text{ satisfying } M\left(-\lambda, 1, \frac{\gamma \,\mathrm{Pe}}{2}\right) = 0. \tag{B.11}$$

`eVals` returns a vector of eigenvalues less than or equal to λ_{N+1} from (B.4). `eVals` is called in `ThetaCross`, `ThetaExit`, and `ThetaFull` and calls `M`.

B.2.6 FitData

`FitData` determines the fitting parameter T_t for all three fitting methods (α-intercept, curve fit, and level set) for a particular fitting condition (cross-sectional average, full average, or exit temperature). `FitData` is called in the main script and calls `AlphaFitExit`, `AlphaFitExit`, `AlphaFitFull`, `ThetaCross`, `ThetaExit`, and `ThetaFull`.

B.2.7 M

`M` defines Kummer's M function (see discussion in §3.1). `M` is called in `Dn`, `eVals`, `ThetaCross`, `ThetaExit`, and `ThetaFull`.

B.2.8 MakeFigure

`MakeFigure` plots the experimental data and all three fitted curves (α-intercept, curve fit, and level set) for a particular fitting condition (cross-sectional average, full average, or exit temperature) and computes the runtime of this construction. `MakeFigure` is called in the main script and calls `ThetaCross`, `ThetaExit`, and `ThetaFull`.

B.2.9 ThetaCross

`ThetaCross` computes $\langle\Theta\rangle(z)$ defined as:

$$\langle\Theta\rangle(z) = \frac{\alpha - \langle T\rangle(z)}{\alpha + 1} = \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 2, \frac{\gamma \,\mathrm{Pe}}{2}\right) e^{-2\gamma\lambda_n z}, \tag{B.12}$$

where (3.30) is used to evaluate $\langle T\rangle(z)$. `ThetaCross` is called in `AlphaFitCross`, `FitData`, and `MakeFigure` and calls `Dn`, `eVals`, and `M`.

B.2.10 ThetaExit

`ThetaExit` computes $\Theta(x,z)$ defined as:

$$\Theta(x,z) = \frac{\alpha - T(x,z)}{\alpha + 1} = \sum_{n=1}^{\infty} D_n M\left(-\lambda_n, 1, x\right) e^{-2\gamma\lambda_n z}, \tag{B.13}$$

where (3.51) is used to evaluate $T(x,z)$. `ThetaExit` is called in `AlphaFitExit`, `FitData`, and `MakeFigure` and calls `Dn`, `eVals`, and `M`.

B.2.11 ThetaFull

`ThetaExit` computes $\bar{\Theta}(z)$ defined as:

$$\bar{\Theta}(z) = \frac{\alpha - \bar{T}(z)}{\alpha + 1} = -\frac{1}{2\gamma}\left[1 + \frac{2}{\gamma\,\mathrm{Pe}}\left(1 - e^{\gamma\,\mathrm{Pe}/2}\right) + \sum_{n=1}^{\infty} \frac{D_n}{\lambda_n} M\left(-\lambda_n, 2, \frac{\gamma\,\mathrm{Pe}}{2}\right) e^{-2\gamma\lambda_n z}\right], \tag{B.14}$$

where (3.47) is used to evaluate $\bar{T}(z)$. `ThetaFull` is called in `AlphaFitFull`, `FitData`, and `MakeFigure` and calls `Dn`, `eVals`, and `M`. Note that in computation we use the following expansion in (B.14) to avoid numerical error at small Pe:

$$\frac{2}{\gamma\,\mathrm{Pe}}\left(1 - e^{\gamma\,\mathrm{Pe}/2}\right) = \frac{2}{\gamma\,\mathrm{Pe}} - \frac{2e^{\gamma\,\mathrm{Pe}/2}}{\gamma\,\mathrm{Pe}}. \tag{B.15}$$

B.3 Code

```
%% Amorphous Polymer in a Taper
%
% |Amorph_Tap.m| fits experimental amorphous polymer data to the
%     tapered amorphous model
% |Amorpth_Tap.m| calls |FitData| and |MakeFigure|
%
% Functions:
%     |AlphaFitCross| sets up (B.7) to be curve fitted for the cross-
%         sectional average condition
%     |AlphaFitExit| sets up (B.8) to be curve fitted for the exit
```

```
%         temperature condition
%     |AlphaFitFull| sets up (B.9) to be curve fitted for the ful
%         average condition
%     |Dn| defines the coefficients $D_n$ from (B.10)
%     |eVals| computes a vector of eigenvalues using (B.11) less than
%         or equal to $\lambda_{N+1}$ from (B.4)
%     |FitData| determines the threshold temperature $T_t$ for a
%         particular theshold condition
%     |M| defines Kummer's $M$ function
%     |MakeFigure| plots the experimental data and fitted curves
%     |ThetaCross| computes $\langle\Theta\rangle(z)$ from (B.12)
%     |ThetaExit| computes $\Theta(x,z)$ from (B.13)
%     |ThetaFull| computes $\bar{\Theta}(z)$ from (B.14)
%
% Variables:
%     |alphaData| is the experimental data for $\alpha$ from (B.5)
%     |beta |is $\beta$ from (B.2)
%     |cond| is a string specifying which theshold condition is being
%         considered
%     |cP| is the specific heat capacity in J/g-K
%     |DeltaT| is the difference between the initial temperature $T_i$
%         and the glass-rubber transition temperature $T_*=T_g$
%         in $^o\text{C}$ from (B.1)
%     |gamma| is $\gamma$ from (B.3)
%     |Hcyl| is the height of the cylindrical portion of the hot end in
%         mm
%     |k| is the thermal conductivity in J/s-m-K
%     lambdaMin is the upper limit for an eigenvalue to contribute to
%         the sum in $\Theta(0,1)$, i.e., $\lambda_{N+1}$ from (B.4)
%     |PeData| is the experimental data for $\text{Pe}$ from (B.6)
%     |plotTimeCross| is the runtime needed to construct the figure
%         for the cross-sectional average condition
%     |plotTimeExit| is the runtime needed to construct the figure for
%         the exit temperature condition
%     |plotTimeFull| is the runtime needed to construct the figure for
```

```
46 %         the full average condition
47 %     |rho| is the density in g/cc
48 %     |Rmax| is the maximum nozzle radius in mm
49 %     |Rmin| is the minimum nozzle radius in mm
50 %     |TmaxData| is the experimental data $T_{\max}$ in $^o\text{C}$
51 %     |Tg| is the glass-rubber transition temperature $T_*=T_{\rm g}$
52 %         in $^o\text{C}$
53 %     |Ti| is the initial temperature in $^o\text{C}$
54 %     |TtCross| is a vector of the threshold temperature for the
55 %         cross-sectional averagecondition for each fitting method
56 %     |TtExit| is a vector of the threshold temperature for the exit
57 %         temperature condition for each fitting method
58 %     |TtFull| is a vector of the threshold temperature for the full
59 %         average condition for each fitting method
60 %     |VminData| is the experimental data for $V_{\min}$
61 %         in $^o\text{C}$
62 %%
63 % Experimental value from [5]
64 %%
65 Hcyl = 30; %mm
66 %%
67 % Experimental values from [8]
68
69 cP = 2.1; %J/g-K
70 k = 0.205; %J/s-m-K
71 rho = 1.1; %g/cc
72 Rmax = 3.175/2; %mm
73 Ti = 20; %^oC
74 Tg = 100; %^oC
75 DeltaT = Tg-Ti; %K
76 %%
77 % Measured value
78
79 Rmin = 0.35/2; %mm
80 %%
```

```
% Compute $\beta$ and $\gamma$

beta = Rmin/Rmax;
gamma = log(beta^(-1));
%%
% Set $\lambda_{N+1}$

lambdaMin = 3.84;
%%
% Experimental data for ABS through $0.35$ mm nozzle from [8]

TmaxData = [245;245;245;240;235;230;230;230;225;220;215;210;205;200;...
    195;190;190;190;185;180;175]; %^oC
VminData = [3.44;3.37;3.43;3.13;3.05;2.83;2.80;2.83;2.51;2.25;2.05;...
    1.85;1.64;1.32;1.05;0.76;0.74;0.75;0.54;0.39;0.23]; %mm/s
%%
% Scale experimental data

alphaData = (TmaxData-Tg)/DeltaT;
PeData = cP*rho*Rmax^2*VminData/(k*Hcyl);
%% Cross-Sectional Average Condition
%%
cond = 'Cross';
TtCross = FitData(cond,gamma,lambdaMin,alphaData,PeData);
plotTimeCross = MakeFigure(cond,TtCross,gamma,lambdaMin,alphaData,...
    PeData);
fprintf(['Plotting runtime is %f seconds for the cross-sectional '...
    'average condition.\n'],plotTimeCross);
%% Full Average Condition
%%
cond = 'Full';
TtFull = FitData(cond,gamma,lambdaMin,alphaData,PeData);
plotTimeFull = MakeFigure(cond,TtFull,gamma,lambdaMin,alphaData,...
    PeData);
fprintf(['Plotting runtime is %f seconds for the full average '...
```

```
116     'condition.\n'],plotTimeFull);
117 %% Exit Temperature Condition
118 %%
119 cond = 'Exit';
120 TtExit = FitData(cond,gamma,lambdaMin,alphaData,PeData);
121 plotTimeExit = MakeFigure(cond,TtExit,gamma,lambdaMin,alphaData,...
122     PeData);
123 fprintf(['Plotting runtime is %f seconds for the exit temperature '...
124     'condition.\n'],plotTimeExit);
125 %%
126 function alphaFitCross = AlphaFitCross(Tt,z,gamma,lambdaMin,Pe)
127 % |AlphaFitCross| sets up (B.7) to be curve fitted for the
128 %     cross-sectional average condition
129 % |AlphaFitCross| is called by |FitData|
130 % |AlphaFitCross| calls |ThetaCross|
131 %
132 % Input variables:
133 %   |gamma| is a scalar of $\gamma$ from (B.3)
134 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
135 %   |Pe| is an array of $\rm Pe$ from (B.6)
136 %   |Tt| is a scalar of $T_t$
137 %   |z| is a scalar of the $z$-coordinate
138 %
139 % Output variable:
140 %   |alphaFitCross| is an array of $\alpha$ computed from (B.7)
141 %
142 % Internal variables:
143 %   |thetaCross| is an array of $\langle\Theta\rangle(z)$ values
144 %     from (B.12)
145
146     thetaCross = ThetaCross(z,gamma,lambdaMin,Pe);
147     alphaFitCross = (Tt+thetaCross)./(1-thetaCross);
148 end
149
150 function alphaFitExit = AlphaFitExit(Tt,x,z,gamma,lambdaMin,Pe)
```

```
151 % |AlphaFitExit| sets up (B.8) to be curve fitted for the exit
152 %     temperature condition
153 % |AlphaFitExit| is called by |FitData|
154 % |AlphaFitExit| calls |ThetaExit|
155 %
156 % Input variables:
157 %   |gamma| is a scalar of $\gamma$ from (B.3)
158 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
159 %   |Pe| is an array of $\rm Pe$ from (B.6)
160 %   |Tt| is a scalar of $T_t$
161 %   |x| is a scalar of the $x$-coordinate
162 %   |z| is a scalar of the $z$-coordinate
163 %
164 % Output variable:
165 %   |alphaFitExit| is an array of $\alpha$ computed from (B.8)
166 %
167 % Internal variables:
168 %   |thetaExit| is an array of $\Theta(x,z)$ values from (B.13)
169
170     thetaExit = ThetaExit(x,z,gamma,lambdaMin,Pe);
171     alphaFitExit = (Tt+thetaExit)./(1-thetaExit);
172 end
173
174 function alphaFitFull = AlphaFitFull(Tt,z,gamma,lambdaMin,Pe)
175 % |AlphaFitFull| sets up (B.9) to be curve fitted for the full
176 %     average condition
177 % |AlphaFitFull| is called by |FitData|
178 % |AlphaFitFull| calls |ThetaFull|
179 %
180 % Input variables:
181 %   |gamma| is a scalar of $\gamma$ from (B.3)
182 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
183 %   |Pe| is an array of $\rm Pe$ from (B.6)
184 %   |Tt| is a scalar of $T_t$
185 %   |z| is a scalar of the $z$-coordinate
```

```
186 %
187 % Output variable:
188 %   |alphaFitFull| is an array of $\alpha$ computed from (B.9)
189 %
190 % Internal variables:
191 %   |thetaFull| is an array of $\bar{\Theta}(z)$ values from (B.14)
192
193     thetaFull = ThetaFull(z,gamma,lambdaMin,Pe);
194     alphaFitFull = (Tt+thetaFull)./(1-thetaFull);
195 end
196
197 function dn = Dn(gamma,Pe,lambdan)
198 % |Dn| defines the coefficient $D_n$ from (B.10)
199 % |Dn| is called by |ThetaCross|, |ThetaExit|, and |ThetaFull|
200 % |Dn| calls |M|
201 %
202 % Input variables:
203 %   |gamma| is a scalar of $\gamma$ from (B.3)
204 %   |lambdan| is a scalar of $\lambda_n$ from (B.11)
205 %   |Pe| is a scalar of $\rm Pe$ from (B.6)
206 %
207 % Output variable:
208 %   |dn| is an scalar of $D_n$ computed from (B.10)
209 %
210 % Internal variables:
211 %   |denominator| is an scalar of the denominator of (B.10)
212 %   |integrand| is a function handle of the integrand of the integral
213 %     in the denominator of (B.10)
214 %   |numerator| is an array of the numerator of (B.10)
215
216     numerator=M(1-lambdan,1,gamma*Pe/2)*exp(-gamma*Pe/2);
217     integrand=@(x) M(-lambdan,1,x).^2.*exp(-x);
218     denominator=integral(integrand,0,gamma*Pe/2);
219     dn=numerator./denominator;
220 end
```

```
221
222 function lambda = eVals(gamma,lambdaMin,Pe)
223 % |eVals| computes a vector of eigenvalues using (B.11) less than or
224 %     equal to $\lambda_{N+1}$ from (B.4)
225 % |eVals| is called by |ThetaCross|, |ThetaExit|, and |ThetaFull|
226 % |eVals| calls |M|
227 %
228 % Input variables:
229 %   |gamma| is a scalar of $\gamma$ from (B.3)
230 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
231 %   |Pe| is an scalar of $\rm Pe$ from (B.6)
232 %
233 % Output variable:
234 %   |lambda| is an array eigenvalues from (B.11)
235 %
236 % Internal variables:
237 %   |lambda1| is a scalar of $\lambda_1$ from (B.11)
238 %   |lambda2| is a scalar of $\lambda_2$ from (B.11)
239 %   |lambdaGuess| is a scalar of the initial guess for |lambda1|
240 %   |lambdaRange| is the interval $[\lambda_1,\lambda_{N+1}]$
241 %   |m| is a function handle of the eigenvalue condition (B.11)
242
243     m = @(lambda) M(-lambda,1,gamma*Pe/2);
244
245     % Find first eigenvalue
246     lambdaGuess = 2/(gamma*Pe);
247     lambda1 = fzero(m,lambdaGuess);
248
249     try
250     % Try and find second eigenvalue
251         lambdaRange = [lambda1+eps,lambdaMin];
252         lambda2 = fzero(m,lambdaRange);
253         lambda = [lambda1 lambda2];
254     catch
255     % Return 1st eigenvalue if there is only one less than |lambdaMin|
```

```
256         lambda = lambda1;
257     end
258 end
259
260 function Tt = FitData(cond,gamma,lambdaMin,alpha,Pe)
261 % |FitData| determines the threshold temperature $T_t$ for a particular
262 %   theshold condition
263 % |FitData| is called in the main script
264 % |FitData| calls |AlphaFitCross|, |AlphaFitExit|, |AlphaFitFull|,
265 %   |ThetaCross|, |ThetaExit|, and |ThetaFull|
266 %
267 % Input variables:
268 %   |alpha| is an array of the $\alpha$ from (B.5)
269 %   |cond| is a string specifying which theshold condition is being
270 %     considered
271 %   |gamma| is a scalar of $\gamma$ from (B.3)
272 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
273 %   |Pe| is an array of the $\rm Pe$ from (B.6)
274 %
275 % Output variable:
276 %   |Tt| is a vector storing the fitted $T_t$ using the $\alpha$-
277 %    intercept, curve fit, and level set fitting methods, respectively
278 %
279 % Internal variables:
280 %   |curveFitFun| is a function handle of (B.7), (B.8), or (B.9)
281 %     to be fitted in the curve fit method
282 %   |linFitParams| is a vector of the parameters from the linear fit of
283 %     the data
284 %   |options| is used to set the options for |lsqcurvefit|
285 %   |PeMat| is a matrix with a column of ones and a coulmn of the
286 %     $\rm Pe$ from (B.6)
287 %   |x| is a scalar of the $x$-coordinate where the theshold condtion
288 %     is imposed
289 %   |z| is a scalar of the $z$-coordinate where the theshold condtion
290 %     is imposed
```

```
291
292     % Pre-allocate size of |Tt|
293     Tt = zeros(3,1);
294
295     % $\alpha$-intercept fitting method
296     PeMat = [ones(size(Pe)),Pe];
297     linFitParams = (PeMat'*PeMat)\PeMat'*alpha;
298     Tt(1) = linFitParams(1);
299
300     options = optimoptions('lsqcurvefit','Display','off');
301     if strcmp(cond,'Cross')
302     % For cross-sectional average condtion
303         z = 1;
304
305         % curve fitting method
306         curveFitFun = @(Tt,Pe) AlphaFitCross(Tt,z,gamma,lambdaMin,Pe);
307         Tt(2) = lsqcurvefit(curveFitFun,Tt(1),Pe,alpha,...
308             -Inf,Inf,options);
309
310         % level set fitting method
311         Tt(3) = mean(alpha-(alpha+1).*ThetaCross(z,gamma,lambdaMin,...
312             Pe));
313
314     elseif strcmp(cond,'Full')
315     % For full average condtion
316         z = 1;
317
318         % curve fitting method
319         curveFitFun = @(Tt,Pe) AlphaFitFull(Tt,z,gamma,lambdaMin,Pe);
320         Tt(2) = lsqcurvefit(curveFitFun,Tt(1),Pe,alpha,...
321             -Inf,Inf,options);
322
323         % level set fitting method
324         Tt(3) = mean(alpha-(alpha+1).*ThetaFull(z,gamma,lambdaMin,...
325             Pe));
```

```
326     elseif strcmp(cond,'Exit')
327     % For exit temperature condtion
328         x = 0;
329         z = 1;
330 
331         % curve fitting method
332         curveFitFun = @(Tt,Pe) AlphaFitExit(Tt,x,z,gamma,lambdaMin,Pe);
333         Tt(2) = lsqcurvefit(curveFitFun,Tt(1),Pe,alpha,...
334             -Inf,Inf,options);
335 
336         % level set fitting method
337         Tt(3) = mean(alpha-(alpha+1).*ThetaExit(x,z,gamma,lambdaMin,...
338             Pe));
339     else
340     % Throw error for any other value of |cond|
341         error('Unknown theshold condition');
342     end
343 end
344 
345 function m = M(a,b,z)
346 % |M| defines Kummer's $M$ function
347 % |M| is called by |Dn|, |eVals|, |ThetaCross|, |ThetaExit|, and
348 %   |ThetaFull|
349 %
350 % Input variables:
351 %   |a| is an array of the first argument of $M$
352 %   |b| is an array of the second argument of $M$
353 %   |z| is an array of the third argument of $M$
354 %
355 % Output variable:
356 %   |m| is an array of $M$
357 
358     m = hypergeom(a,b,z);
359 end
360 
```

```
361 function plotTime = MakeFigure(cond,Tt,gamma,lambdaMin,alpha,Pe)
362 % |MakeFigure| plots the experimental data and fitted curves for a
363 %   particular fitting condition and computes the runtime of this
364 %   construction
365 % |MakeFigure| is called in the main script
366 % |MakeFigure| calls |ThetaCross|, |ThetaExit|, and |ThetaFull|
367 %
368 % Input variables:
369 %   |alpha| is an array of the $\alpha$ from (B.5)
370 %   |cond| is a string specifying which theshold condition is being
371 %     considered
372 %   |gamma| is a scalar of $\gamma$ from (B.3)
373 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
374 %   |Pe| is an array of the $\rm Pe$ from (B.6)
375 %   |Tt| is a vector storing the fitted $T_t$ using the $\alpha$-
376 %    intercept, curve fit, and level set fitting methods, respectively
377 %
378 % Output variable:
379 %   |plotTime| is a scalar of the runtime needed to construct the
380 %     figure
381 %
382 % Internal variables:
383 %   |alphaLim| is a vector of the $\alpha$-limits of the figure
384 %   |alphaPlot| is a matrix of $\alpha$ values to be plotted by contour
385 %   |i| is a scalar indexing the fitting methods
386 %   |linFitParams| is a vector of the parameters from the linear fit of
387 %     the data
388 %   |linSpec| is a cell array of line specifications for the fitted
389 %     curves
390 %   |PeMat| is a matrix with a column of ones and a coulmn of the
391 %     $\rm Pe$ from (B.6)
392 %   |PeLim| is a vector of the $\rm Pe$-limits of the figure
393 %   |PePlot| is an vector of $\rm Pe$ values from (B.6) to be used in
394 %     plotting
395 %   |plotRes| is a scalar specifying the size of |PePlot|
```

```
396 %   |ThetaPlot| is a vector of the size of |PePlot| of (B.12), (B.13),
397 %     or (B.14) depending on what condition is being considered
398 %   |x| is a scalar of the $x$-coordinate where the theshold condtion
399 %     is imposed
400 %   |z| is a scalar of the $z$-coordinate where the theshold condtion
401 %     is imposed
402
403     tic;
404
405     plotRes = 31;
406     PePlot = linspace(eps,max(Pe),plotRes);
407
408     if strcmp(cond,'Cross')
409     % For cross-sectional average condtion
410         z = 1;
411         ThetaPlot = ThetaCross(z,gamma,lambdaMin,PePlot);
412     elseif strcmp(cond,'Full')
413     % For full average condtion
414         z = 1;
415         ThetaPlot = ThetaFull(z,gamma,lambdaMin,PePlot);
416     elseif strcmp(cond,'Exit')
417     % For exit temperature condtion
418         x = 0;
419         z = 1;
420         ThetaPlot = ThetaExit(x,z,gamma,lambdaMin,PePlot);
421     else
422     % Throw error for any other value of |cond|
423         error('Unknown theshold condition');
424     end
425
426     figure;
427     hold on;
428
429     % Plot data
430     scatter(alpha,Pe,'+k');
```

```
431
432     % Plot linear fit of the data
433     PeMat = [ones(size(Pe)),Pe];
434     linFitParams = (PeMat'*PeMat)\PeMat'*alpha;
435     fplot(@(alpha) (alpha-linFitParams(1))/linFitParams(2),'k-',...
436         [0,max(alpha)+0.05]);
437
438     % Plot fitted curves
439     linSpec = {'k-','k--','k:'};
440     for i = 1:size(Tt)
441         alphaPlot = (Tt(i)+ThetaPlot)./(1-ThetaPlot);
442         plot(alphaPlot,PePlot,linSpec{i});
443     end
444
445     % Plot specifications
446     xlim([min([Tt;0.05])-0.05,max(alpha)+0.05]);
447     ylim([0,max(Pe)+0.3]);
448     xlabel('\fontname{cambria math} \alpha','fontsize',18);
449     ylabel('\fontname{cambria math} Pe','fontsize',18);
450
451     % Display $T_t$ values on figure
452     PeLim = get(gca,'ylim');
453     alphaLim = get(gca,'xlim');
454     text(alphaLim(1)+0.05*(alphaLim(2)-alphaLim(1)),PeLim(2)-0.05...
455         *(PeLim(2)-PeLim(1)),['$\alpha$-Intercept: '...
456         '$T_{\mathrm{t}} = ',num2str(Tt(1),'%.6f'),'$'],...
457         'FontSize',18,'interpreter','latex');
458     text(alphaLim(1)+0.05*(alphaLim(2)-alphaLim(1)),PeLim(2)-0.15*...
459         (PeLim(2)-PeLim(1)),['Curve Fit: '...
460         '$T_{\mathrm{t}} = ',num2str(Tt(2),'%.6f'),'$'],...
461         'FontSize',18,'interpreter','latex');
462     text(alphaLim(1)+0.05*(alphaLim(2)-alphaLim(1)),PeLim(2)-0.25*...
463         (PeLim(2)-PeLim(1)),['Level Set: '...
464         '$T_{\mathrm{t}} = ',num2str(Tt(3),'%.6f'),'$'],...
465         'FontSize',18,'interpreter','latex');
```

```
466
467     hold off
468
469     plotTime = toc;
470 end
471
472 function thetaCross=ThetaCross(z,gamma,lambdaMin,Pe)
473 % |ThetaCross| computes $\langle\Theta\rangle(z)$ from (B.12)
474 % |ThetaCross| is called by |AlphaFitCross|, |FitData|, and
475 %     |MakeFigure|
476 % |ThetaCross| calls |Dn|, |eVals|, and |M|
477 %
478 % Input variables:
479 %   |gamma| is a scalar of $\gamma$ from (B.3)
480 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
481 %   |Pe| is an array of $\rm Pe$ from (B.6)
482 %   |z| is a scalar of the $z$-coordinate where the theshold condtion
483 %     is imposed
484 %
485 % Output variable:
486 %   |thetaCross| is an array of $\langle\Theta\rangle(z)$ from (B.12)
487 %
488 % Internal variables:
489 %   |i| is a scalar indexing the rows of |Pe|
490 %   |j| is a scalar indexing the columns of |Pe|
491 %   |lambda| is a vector of the eigenvalues from (B.11)
492 %   |n| is a scalar indexing the terms in the sum in (B.12)
493
494     % Pre-allocate size of |thetaCross|
495     thetaCross = zeros(size(Pe));
496
497     for i = 1:size(Pe,1)
498     % Iterate over rows of |Pe|
499         for j = 1:size(Pe,2)
500         % Iterate over columns of |Pe|
```

```
501
502             % Determine eigenvalues
503             lambda = eVals(gamma,lambdaMin,Pe(i,j));
504
505             % Compute |thetaCross|
506             if ~isempty(lambda)
507             % Cases where some terms in sum are not small
508
509                 for n = 1:length(lambda)
510                 % Iterate over terms in sum
511                     thetaCross(i,j) = thetaCross(i,j)+...
512                         Dn(gamma,Pe(i,j),lambda(n)).*...
513                         M(-lambda(n),2,gamma*Pe(i,j)/2).*...
514                         exp(-2*gamma*lambda(n)*z);
515                 end
516             else
517             % Cases where all terms in sum are small
518
519                 % Compute smallest eigenvalue
520                 lambda = fsolve(@(lambda) M(-lambda,1,...
521                     gamma*Pe(i,j)/2),0,optimoptions('fsolve',...
522                     'Display','off'));
523
524                 % |thetaCross| is equal to the first term
525                 thetaCross(i,j) = Dn(gamma,Pe(i,j),lambda).*...
526                     M(-lambda,2,gamma*Pe(i,j)/2).*...
527                     exp(-2*gamma*lambda*z);
528             end
529         end
530     end
531 end
532
533 function thetaExit=ThetaExit(x,z,gamma,lambdaMin,Pe)
534 % |ThetaExit| computes $\Theta(x,z)$ from (B.13)
535 % |ThetaExit| is called by |AlphaFitExit|, |FitData|, and
```

```
536 %     |MakeFigure|
537 % |ThetaExit| calls |Dn|, |eVals|, and |M|
538 %
539 % Input variables:
540 %   |gamma| is a scalar of $\gamma$ from (B.3)
541 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
542 %   |Pe| is an array of $\rm Pe$ from (B.6)
543 %   |x| is a scalar of the $x$-coordinate where the theshold condtion
544 %     is imposed
545 %   |z| is a scalar of the $z$-coordinate where the theshold condtion
546 %     is imposed
547 %
548 % Output variable:
549 %   |thetaExit| is an array of $\Theta(x,z)$ from (B.13)
550 %
551 % Internal variables:
552 %   |i| is a scalar indexing the rows of |Pe|
553 %   |j| is a scalar indexing the columns of |Pe|
554 %   |lambda| is a vector of the eigenvalues from (B.11)
555 %   |n| is a scalar indexing the terms in the sum in (B.13)
556
557     % Pre-allocate size of |thetaExit|
558     thetaExit = zeros(size(Pe));
559
560     for i = 1:size(Pe,1)
561     % Iterate over rows of |Pe|
562         for j = 1:size(Pe,2)
563         % Iterate over columns of |Pe|
564
565             % Determine eigenvalues
566             lambda = eVals(gamma,lambdaMin,Pe(i,j));
567
568             % Compute |thetaExit|
569             if ~isempty(lambda)
570             % Cases where some terms in sum are not small
```

```
571
572                 for n = 1:length(lambda)
573                 % Iterate over terms in sum
574                     thetaExit(i,j) = thetaExit(i,j)+...
575                         Dn(gamma,Pe(i,j),lambda(n)).*...
576                         M(-lambda(n),1,x).*exp(-2*gamma*lambda(n)*z);
577                 end
578             else
579             % Cases where all terms in sum are small
580
581                 % Compute smallest eigenvalue
582                 lambda = fsolve(@(lambda) M(-lambda,1,...
583                     gamma*Pe(i,j)/2),0,optimoptions('fsolve',...
584                     'Display','off'));
585
586                 % |thetaExit| is equal to the first term
587                 thetaExit(i,j) = Dn(gamma,Pe(i,j),lambda).*...
588                     M(-lambda,1,x).*...
589                     exp(-2*gamma*lambda*z);
590             end
591         end
592     end
593 end
594
595 function thetaFull=ThetaFull(z,gamma,lambdaMin,Pe)
596 % |ThetaFull| computes $\bar{\Theta}(z)$ from (B.14)
597 % |ThetaFull| is called by |AlphaFitFull|, |FitData|, and
598 %     |MakeFigure|
599 % |ThetaFull| calls |Dn|, |eVals|, and |M|
600 %
601 % Input variables:
602 %   |gamma| is a scalar of $\gamma$ from (B.3)
603 %   |lambdaMin| is a scalar of $\lambda_{N+1}$ from (B.4)
604 %   |Pe| is an array of $\rm Pe$ from (B.6)
605 %   |z| is a scalar of the $z$-coordinate where the theshold condtion
```

```
606 %     is imposed
607 %
608 % Output variable:
609 %   |thetaFull| is an array of $\bar{\Theta}(z)$ from (B.14)
610 %
611 % Internal variables:
612 %   |i| is a scalar indexing the rows of |Pe|
613 %   |j| is a scalar indexing the columns of |Pe|
614 %   |lambda| is a vector of the eigenvalues from (B.11)
615 %   |n| is a scalar indexing the terms in the sum in (B.14)
616 %   |sumTerm| is the sum in (B.14)
617
618     % Pre-allocate size of |sumTerm|
619     sumTerm = zeros(size(Pe));
620
621     for i = 1:size(Pe,1)
622     % Iterate over rows of |Pe|
623         for j = 1:size(Pe,2)
624         % Iterate over columns of |Pe|
625
626             % Determine eigenvalues
627             lambda = eVals(gamma,lambdaMin,Pe(i,j));
628
629             % Compute |sumTerm|
630             if ~isempty(lambda)
631             % Cases where some terms in sum are not small
632
633                 for n = 1:length(lambda)
634                 % Iterate over terms in sum
635                     sumTerm(i,j) = sumTerm(i,j)+...
636                         Dn(gamma,Pe(i,j),lambda(n)).*...
637                         M(-lambda(n),2,gamma*Pe(i,j)/2)./lambda(n).*...
638                         exp(-2*gamma*lambda(n)*z);
639                 end
640             else
```

```
641             % Cases where all terms in sum are small
642
643                 % Compute smallest eigenvalue
644                 lambda = fsolve(@(lambda) M(-lambda,1,...
645                     gamma*Pe(i,j)/2),0,optimoptions('fsolve',...
646                     'Display','off'));
647
648                 % |sumTerm| is equal to the first term
649                 sumTerm(i,j) = Dn(gamma,Pe(i,j),lambda).*...
650                     M(-lambda,2,gamma*Pe(i,j)/2)./lambda...
651                     .*exp(-2*gamma*lambda*z);
652             end
653         end
654
655         % Compute |thetaFull|
656         thetaFull = -1./(2*gamma).*(1+2./(gamma.*Pe)...
657             -2*exp(gamma.*Pe/2)./(gamma.*Pe)+sumTerm);
658     end
659 end
```

Appendix C

CRYSTALLINE MODEL IMPLEMENTATION

In this appendix, we display the MATLAB code used to implement the crystalline models from §4. As in Appendix B, we start by listing the equations referenced in the code.

C.1 Variables

The following are variable definitions used in the main script in order of appearance:

$$\Delta T = T_* - T_i, \tag{C.1}$$

$$\beta = \frac{R_{\min}}{R_{\max}}, \tag{C.2}$$

$$\mathrm{St} = \frac{\Delta T c_P}{c_L}, \tag{C.3}$$

$$z_{\mathrm{end}} = \frac{H_{\mathrm{cyl}} + H_{\mathrm{noz}}}{H_{\mathrm{cyl}}}, \tag{C.4}$$

$$\alpha = \frac{T_{\max} - T_*}{\Delta T}, \tag{C.5}$$

$$\mathrm{Pe} = \frac{\rho c_P R_{\max}^2 V_{\min}}{k H_{\mathrm{cyl}}}, \tag{C.6}$$

$$a = \frac{-1 + \sqrt{1 + 2\,\mathrm{St}\,\alpha}}{\mathrm{St}\,\alpha}. \tag{C.7}$$

C.2 Functions

Here we describe all the functions defined in the code in alphabetical order.

C.2.1 AlphaFitExit

`AlphaFitExit` defines the function to be curve fitted using the exit temperature condition in `FitData`. `AlphaFitExit` returns $T_\mathrm{t} = T_\mathrm{p}(\epsilon, 1)$ solved for α using (C.11) to evaluate $T_\mathrm{p}(\epsilon, 1)$:

$$\alpha = w(z)\frac{\left(1 - \sqrt{1 + 2\,\mathrm{St}\,T_\mathrm{t}}\right)\log\epsilon + \mathrm{St}\,T_\mathrm{t} w(z)}{\mathrm{St}[\log\epsilon - w(z)]^2}. \tag{C.8}$$

`AlphaFitExit` is called by `FitData`.

C.2.2 B

`B` defined the function $B(z)$ from (4.66b):

$$B(z) = \begin{cases} 0, & 0 \le z \le 1, \\ \frac{1-\beta}{z_\mathrm{end}-1}, & z > 1. \end{cases} \tag{C.9}$$

`B` is called in the main script.

C.2.3 FitData

`FitData` determines the fitting parameters T_t and ϵ for all three fitting conditions (cross-sectional average, full average, and exit temperature) for a particular geometry. `FitData` is called in the main script and calls `AlphaFitExit`, `TCross`, `TExit`, `TFull`, and `wOfz`. When fitting the data to the average temperature conditions (cross-sectional and full), `FitData` minimizes $\left|T_\mathrm{t} \cdot \mathbf{1} - T\left(\boldsymbol{\alpha}, \mathbf{Pe}\right)\right|^2$ (i.e., the level set fitting method). When fitting the data to the exit temperature condition, `FitData` minimizes $\left|\boldsymbol{\alpha} - \alpha\left(\mathbf{Pe}; T_\mathrm{t}\right)\right|^2$ (i.e., the curve fit fitting method). (Note that more discussion on these fitting methods can be found in §3.2.1.) To ensure that the solution to the exit temperature optimization problem is indeed the global minimum, `FitData` uses a `MultiStart` object to solve the problem for several initial guesses of the fitting parameters.

C.2.4 MakeFigure

`MakeFigure` plots the experimental data and fitted curves for a particular fitting condition and geometry and computes the runtime of this construction. `MakeFigure` is called in the main script and calls `TCross`, `TExit`, and `TFull`.

C.2.5 TCross

`TCross` computes $\langle T \rangle(z)$ using (4.29) with the change of variables $\sigma \to w$ from §4.5:

$$\langle T \rangle(z) = \alpha \left\{ 1 + \frac{2 - a\left[1 + e^{2w(z)}\right]}{2 \log \sigma(z)} + \frac{(1-a)\left[1 - e^{2w(z)}\right]}{2w^2(z)} \right\}. \tag{C.10}$$

`TCross` is called in `FitData` and `MakeFigure` and calls `wOfz`.

C.2.6 TExit

`TExit` computes $T_{\mathrm{p}}(z)$ using (4.14) with the change of variables $\sigma \to w$ from §4.5:

$$T_{\mathrm{p}}(y, z) = \alpha \left\{ a \left[1 - \frac{\log y}{w(z)} \right] + (1-a) \left[1 - \frac{\log y}{w(z)} \right]^2 \right\}. \tag{C.11}$$

`TExit` is called in `FitData` and `MakeFigure` and calls `wOfz`.

C.2.7 TFull

`TFull` computes $\bar{T}(z)$ using (4.48) with the change of variables $\sigma \to w$ from §4.5:

$$\bar{T}(z) = \alpha[1 + \psi(z)]. \tag{C.12}$$

`TFull` is called in `FitData` and `MakeFigure` and calls `wpsiOfz`.

C.2.8 wpsiOfz

`wpsiOfz` computes $w(z)$ and $\psi(z)$ as defined in §4.5. `wOfz` is called by `TFull` and has several nested functions: `Derivatives`, `dpsidz`, `dwdz`, `psiSplice`, `wEvents`, and `wSplice`.

C.2.8.1 Derivatives

`Derivatives` creates a vector of the ODEs for $w(z)$ and $\psi(z)$. `Derivatives` is called in the main`wpsiOfz` and in `wSplice` and calls `dpsidz` and `dwdz`

C.2.8.2 dpsidz

`dpsidz` defines the ODE to be solved for $\psi(z)$. `dpsidz` is called in `Derivatives`. `dpsidz` returns $\psi'(z)$ from (4.47):

$$\frac{\mathrm{d}\psi}{\mathrm{d}z} = \frac{(1-a)\left(1-e^{2w}\right) + \left[2 - a\left(1+e^{2w}\right)\right] w}{2w^2}, \tag{C.13}$$

where $\psi(0) = 0$ is the initial condition.

C.2.8.3 dwdz

`dwdz` defines the ODE to be solved for $w(z)$. `dwdz` is called in `Derivatives`. `dwdz` returns $w'(z)$ from (4.42):

$$\frac{\mathrm{d}w}{\mathrm{d}z} = \frac{8\,\mathrm{Pe}^{-1}\,\eta(z) e^w w}{2(1-a) + (2-a)w + e^{2w}\left[2aw^2 + (2-3a)w - 2(1-a)\right]}, \tag{C.14a}$$

$$\eta(z) = (1-a)w + \frac{\mathrm{Pe}}{4}\frac{R'}{R}\left\{(1-a)\left(1-e^{2w}\right) + \left[2 - a\left(1+e^{2w}\right)\right] w\right\}, \tag{C.14b}$$

where R and R' are defined for each geometry. For the cylindrical case,

$$R(z) = 1, \tag{C.15a}$$

$$R'(z) = 0. \tag{C.15b}$$

For the tapered case,

$$R(z) = 1 - (1-\beta)z, \tag{C.16a}$$

$$R'(z) = -(1-\beta). \tag{C.16b}$$

For the combined case,

$$R(z) = 1 - B(z)(z-1), \tag{C.17a}$$

$$R'(z) = -B(z), \tag{C.17b}$$

where $B(z)$ is defined in (C.9). Also note that $w(0) = 0$ is the initial condition; however, since $w = 0$ is a solution to (C.14), we define $w(0)$ to be a small negative number. Increasing the magnitude of this number was able to resolve some numerical issues.

C.2.8.4 psiSplice

`psiSplice` is used as a patch to solve for ψ over ranges of z when w is too close to zero. It is called in the main `wpsiOfz` and in `wSplice`. The splicing function is given by (4.49):

$$\psi(z) = \psi_0 - \int_{z_0}^{z} \frac{\mathrm{d}\psi}{\mathrm{d}\zeta}(w(\zeta))\,\mathrm{d}\zeta, \tag{C.18}$$

where $\psi_0 = \psi(z_0)$ is where `wEvents` terminates integration and $w(\zeta)$ are as defined in (C.19) (see derivation in §4.5.3). (C.18) is used to compute $\psi(z)$ for a z where $w(z)$ is enough less than zero to resume integration. `psiSplice` is called in the main `wpsiOfz` and in`wSplice` and calls `dpsidz` and `wSplice`.

C.2.8.5 wEvents

`wEvents` terminates the integration of the ode solver if w becomes to close to zero to be computed accurately (see discussion in §4.5.2). `wEvents` is called in the main `wpsiOfz` and in `wSplice`. After `wEvents` terminates integration, `wSplice` and `psiSplice` are used as a patch $w(z)$ and $\psi(z)$ until w is again far enough from zero to resume integration.

C.2.8.6 wSplice

`wSplice` is used as a patch to solve for w over ranges of z when w is too close to zero. It is called in `psiSplice`, the main `wpsiOfz`, and `wSplice` and calls `Derivatives` and `wSplice`. The splicing function is given by (4.46c):

$$w(z) = -\sqrt{w_0^2 + 24\,\mathrm{Pe}^{-1}\,\frac{1-a}{2+a}\,(z - z_0)}, \tag{C.19}$$

where $w_0 = w(z_0)$ is where `wEvents` terminates integration (see derivation in §4.5.2). (C.19) is used to solve for a value of z where $w(z)$ is enough less than zero to resume integration. `wSplice` the resumes integration in a similar fashion to `wpsiOfz`.

C.2.9 wOfz

`wOfz` computes $w(z)$ as defined in §4.5. `wOfz` is called by `TCross` and `TExit` and has several nested functions: `dwdz`, `wEvents`, and `wSplice`.

C.2.9.1 dwdz

`dwdz` defines the ODE to be solved for $w(z)$. `dwdz` is called in the main `wOfz` and in `wSplice`. `dwdz` returns $w'(z)$ from (C.14), where R and R' are defined for each geometry (see definitions in §C.2.8.3). Also note that $w(0) = 0$ is the initial condition; however, since $w = 0$ is a solution to (C.14), we define $w(0)$ to be a small negative number. Increasing the magnitude of this number was able to resolve some numerical issues.

C.2.9.2 wEvents

`wEvents` terminates the integration of the ode solver if w becomes to close to zero to be computed accurately (see discussion in §4.5.2). `wEvents` is called in the main `wOfz` and in `wSplice`. After `wEvents` terminates integration, `wSplice` is used as a patch until w is again far enough from zero to resume integration.

C.2.9.3 wSplice

`wSplice` is used as a patch to solve for w over ranges of z when w is too close to zero. It is called in the main `wOfz` and in `wSplice` and calls `dwdz` and `wSplice`. The splicing function is given by (C.19), where $w_0 = w(z_0)$ is where `wEvents` terminates integration (see derivation in §4.5.2). (C.19) is used to solve for a value of z where $w(z)$ is enough less than zero to resume integration. `wSplice` the resumes integration in a similar fashion to `wOfz`.

C.3 Code

```
%% Crystalline Polymer
%
% |Cryst.m| fits experimental crystalline polymer data to the
%     cylindrical, tapered, and combined crystalline model
% |Cryst.m| calls |B|, |FitData|, and |MakeFigure|
%
% Functions:
%     |AlphaFitExit| sets-up (C.8) to be curve fitted for the exit
%         temperature condition
%     |B| defines $B(z)$ from (C.9)
%     |FitData| determines the threshold temperatures $T_t$
%         and $\epsilon$ for all three threshold condtions for a
%         particular geometry
%     |MakeFigure| plots the experimental data and fitted curves
%     |TCross| computes $\langle T\rangle(z)$ from (C.10)
%     |TExit| computes $T(x,z)$ from (C.11)
%     |TFull| computes $\bar{T}(z)$ from (C._12)
%     |wpsiOfz| computes $[w(z),\psi(z)]$ for the combined case
%     |wOfz| computes $w(z)$ for the cylindrical and tapered cases
%
% Variables:
%     |aData| is the experimental data for $a$ from (C.7)
%     |alphaData| is the experimental data for $\alpha$ from (C.5)
%     |beta |is $\beta$ from (C.2)
%     |cond| is a string specifying which theshold condition is being
%         considered
%     |cL| is the specific latent heat in J/g
%     |cP| is the specific heat capacity in J/g-K
%     |DeltaT| is the difference between the initial temperature $T_i$
%         and the glass-rubber transition temperature $T_*=T_g$
%         in $^o\text{C}$ from (C.1)
%     |dRdz| is function handle of $R^{\prime}(z)$ of the geometry of
```

```
33 %         interest
34 %     |epsilonComb| is $\epsilon$ for the exit temperature condition in
35 %         the combined case
36 %     |epsilonCyln| is $\epsilon$ for the exit temperature condition
37 %         in the cylindrical case
38 %     |epsilonTap| is $\epsilon$ for the exit temperature condition
39 %         in the tapered case
40 %     |Hcyl| is the height of the cylindrical portion of the hot end in
41 %         mm
42 %     |Hnoz| is the height of the nozzle of the hot end in mm
43 %     |k| is the thermal conductivity in J/s-m-K
44 %     |PeData| is the experimental data for $\rm Pe$ from (C.6)
45 %     |plotTimeComb| is a vector of the runtime needed to construct the
46 %         figures for the cross-sectional average, total average, and
47 %         exit temperature conditions, respectively, in the combined
48 %         case
49 %     |plotTimeCyln| is a vector of the runtime needed to construct the
50 %         figures for the cross-sectional average, total average, and
51 %         exit temperature conditions, respectively, in the cylindrical
52 %         case
53 %     |plotTimeTap| is a vector of the runtime needed to construct the
54 %         figures for the cross-sectional average, total average, and
55 %         exit temperature conditions, respectively, in the tapered
56 %         case
57 %     |R| is function handle of $R(z)$ of the geometry of interest
58 %     |rho| is the density in g/cc
59 %     |Rmax| is the maximum nozzle radius in mm
60 %     |Rmin| is the minimum nozzle radius in mm
61 %     |St| is the $\rm St$ (C.3)
62 %     |TmaxData| is the experimental data $T_{\max}$ in $^o\text{C}$
63 %     |Tm| is the melting temperature $T_*=T_{\rm m}$ in $^o\text{C}$
64 %     |Ti| is the initial temperature in $^o\text{C}$
65 %     TtComb is a vector of the threshold temperature for the cross-
66 %         sectional average, total average, and exit temperature
67 %         conditions, respectively, in the combined case
```

```
68 %     TtCyln is a vector of the threshold temperature for the cross-
69 %         sectional average, total average, and exit temperature
70 %         conditions, respectively, in the cylindrical case
71 %     TtTap is a vector of the threshold temperature for the cross-
72 %         sectional average, total average, and exit temperature
73 %         conditions, respectively, in the tapered case
74 %     |VminData| is the experimental data for $V_{\min}$
75 %         in $^o\text{C}$
76 %     |zEnd| is $z_{\rm end}$ from (C.4)
77 %     |zExit| is the $z$-coordinate at the exit of the geometry of
78 %         interest
79 %%
80 % Experimental value from [6]
81 %%
82 Hcyl = 30; %mm
83 %%
84 % Experimental values from [8]
85
86 cP = 1.7; %J/g-K
87 k = 0.13; %J/s-m-K
88 rho = 1.25; %g/cc
89 Rmax = 3.175/2; %mm
90 Ti = 20; %^oC
91 Tm = 155; %^oC
92 DeltaT = Tm-Ti; %K
93 %%
94 % Experimental value from [27]
95
96 cL = 91; %J/g
97 %%
98 % Measured values
99
100 Hnoz = 2; %mm
101 Rmin = 0.35/2; %mm
102 %%
```

```
103 % Compute $\beta$, $\rm St$, and $z_{\rm end}$
104
105 beta = Rmin/Rmax;
106 St = DeltaT*cP/cL;
107 zEnd = (Hcyl+Hnoz)/Hcyl;
108 %%
109 % Experimental data for PLA through $0.35$ mm nozzle from [8]
110
111 TmaxData = [230;230;230;225;220;215;210;205;200;195;190;190;190;185;...
112     180;175;170;165;160;155;150;150;150]; %^oC
113 VminData = [3.69;3.70;3.67;3.59;3.40;3.26;3.12;3.05;2.87;2.72;2.48;...
114     2.43;2.46;2.25;2.06;1.87;1.61;1.28;0.93;0.67;0.40;0.41;0.41]; %mm/s
115 %%
116 % Scale experimental data
117
118 alphaData = (TmaxData-Tm)/DeltaT;
119 PeData = cP*rho*Rmax^2*VminData/(k*Hcyl);
120 %%
121 % Remove data for temperatures below $170 ^o\text{C}$ to ensure melting
122 % has taken place (as
123 %
124 % the melting point of the polymer is $155^o\text{C}$)
125
126 PeData(alphaData<(170-Tm)/DeltaT)=[];
127 alphaData(alphaData<(170-Tm)/DeltaT)=[];
128 %%
129 % Compute $a$ for each datum
130
131 aData=(-1+sqrt(1+2*St.*alphaData))./(St.*alphaData);
132 %% Cylindrical Case
133 %%
134 zExit=1;
135 R=@(z) 1; % From (C.15a)
136 dRdz=@(z) 0; % From (C.15b)
137
```

```
138 [TtCyln,epsilonCyln]=FitData(zExit,R,dRdz,St,alphaData,aData,PeData,...
139     'ode45');
140
141 plotTimeCyln=zeros(3,1);
142 plotTimeCyln(1)=MakeFigure('Cross',TtCyln(1),zExit,R,dRdz,St,...
143     alphaData,PeData,eps,...
144     'ode45');
145 fprintf(['Cylinder: Plotting runtime is %f seconds for the cross-' ...
146     'sectional average condition.\n'],plotTimeCyln(1));
147 plotTimeCyln(2)=MakeFigure('Full',TtCyln(2),zExit,R,dRdz,St,...
148     alphaData,PeData,eps,...
149     'ode45');
150 fprintf(['Cylinder: Plotting runtime is %f seconds for the total ' ...
151     'average condition.\n'],plotTimeCyln(2));
152 plotTimeCyln(3)=MakeFigure('Exit',[TtCyln(3),log(epsilonCyln)],...
153     zExit,R,dRdz,St,alphaData,...
154     PeData,0.5,'ode45');
155 fprintf(['Cylinder: Plotting runtime is %f seconds for the exit ' ...
156     'temperature condition.\n'],plotTimeCyln(3));
157 %% Tapered Case
158 %%
159 zExit=1;
160 R=@(z) 1-(1-beta)*z; % From (C.16a)
161 dRdz=@(z) -(1-beta); % From (C.16b)
162
163 [TtTap,epsilonTap]=FitData(zExit,R,dRdz,St,alphaData,aData,PeData,...
164     'ode45');
165
166 plotTimeTap=zeros(3,1);
167 plotTimeTap(1)=MakeFigure('Cross',TtTap(1),zExit,R,dRdz,St,...
168     alphaData,PeData,0.25,...
169     'ode45');
170 fprintf(['Taper: Plotting runtime is %f seconds for the cross-' ...
171     'sectional average condition.\n'],plotTimeTap(1));
172 plotTimeTap(2)=MakeFigure('Full',TtTap(2),zExit,R,dRdz,St,...
```

```
173     alphaData,PeData,0.1,...
174     'ode45');
175 fprintf(['Taper: Plotting runtime is %f seconds for the total ' ...
176     'average condition.\n'],plotTimeTap(2));
177 plotTimeTap(3)=MakeFigure('Exit',[TtTap(3),log(epsilonTap)],zExit,R,...
178     dRdz,St,alphaData,PeData,0.5,...
179     'ode45');
180 fprintf(['Taper: Plotting runtime is %f seconds for the exit ' ...
181     'temperature condition.\n'],plotTimeTap(3));
182 %% Combined Case
183 %%
184 zExit=zEnd;
185 R=@(z) 1-B(z,beta,zEnd)*(z-1); % From (C.17a)
186 dRdz=@(z) -B(z,beta,zEnd); % From (C.17b)
187
188 [TtComb,epsilonComb]=FitData(zExit,R,dRdz,St,alphaData,aData,...
189     PeData,'ode15s');
190
191 plotTimeComb=zeros(3,1);
192 plotTimeComb(1)=MakeFigure('Cross',TtComb(1),zExit,R,dRdz,St,...
193     alphaData,PeData,0.25,'ode15s');
194 fprintf(['Combined: Plotting runtime is %f seconds for the cross-' ...
195     'sectional average condition.\n'],plotTimeComb(1));
196 plotTimeComb(2)=MakeFigure('Full',TtComb(2),zExit,R,dRdz,St,...
197     alphaData,PeData,0.15,'ode15s');
198 fprintf(['Combined: Plotting runtime is %f seconds for the total ' ...
199     'average condition.\n'],plotTimeComb(2));
200 plotTimeComb(3)=MakeFigure('Exit',[TtComb(3),log(epsilonComb)],...
201     zExit,R,dRdz,St,alphaData,PeData,0.5,'ode15s');
202 fprintf(['Combined: Plotting runtime is %f seconds for the exit ' ...
203     'temperature condition.\n'],plotTimeComb(3));
204 %%
205 function alphaFitExit=AlphaFitExit(param,wz,St)
206 % |AlphaFitExit| sets-up (C.8) to be curve fitted for the exit
207 %     temperature condition
```

```
208 % |AlphaFitExit| is called by |FitData|
209 %
210 % Input variables:
211 %   |param| is a vector of the fitting parameters $[T_t,\log\epsilon]$
212 %   |St| is a scalar of $\rm St$ from (C.3)
213 %   |wz| is an array of $w(z)$
214 %
215 % Output variable:
216 %   |alphaFitExit| is an array of $\alpha$ computed from (C.8)
217
218     alphaFitExit=wz.*(param(2)*(1-sqrt(1+2*St*param(1)))+...
219         St*param(1)*wz)./(St*(param(2)-wz).^2);
220 end
221
222 function b=B(z,beta,zEnd)
223 % |B| defines $B(z)$ from (C.9)
224 % |B| is called in the main script
225 %
226 % Input variables:
227 %   |beta| is a scalar of $\beta$ from (C.2)
228 %   |z| is an array of $z$-coordinates
229 %   |zEnd| is a scalar of the $z_{\rm end}$ from (C.4)
230 %
231 % Output variable:
232 %   |b| is an array of $B(z)$ computed from (C.9)
233
234     if 0<=z && z<=1
235         b=0;
236     elseif 1<z
237         b=(1-beta)/(zEnd-1);
238     else
239         error('B(z) undefined at z');
240     end
241 end
242
```

```
243 function [Tt,epsilon] = FitData(z,R,dRdz,St,alpha,a,Pe,odeSolver)
244 % |FitData| determines the threshold temperatures $T_t$ and $\epsilon$
245 %   for all three threshold condtions for a particular geometry
246 % |FitData| is called in the main script
247 % |FitData| calls |AlphaFitExit|, |TCross|, |TExit|, |TFull|, and
248 %     |wOfz|
249 %
250 % Input variables:
251 %   |a| is an array of the $a$ from (C.7)
252 %   |alpha| is an array of the $\alpha$ from (C.5)
253 %   |dRdz| is a function handle of $R^{\prime}(z)$
254 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
255 %     'ode15s')
256 %   |Pe| is an array of the $\rm Pe$ from (C.6)
257 %   |St| is a scalar of $\rm St$ from (C.3)
258 %   |R| is a function handle of $R(z)$
259 %   |z| is a scalar of the $z$-coordinate where the theshold condtions
260 %     are imposed
261 %
262 % Output variables:
263 %   |Tt| a vector of the threshold temperature for the cross-sectional
264 %     average, total average, and exit temperature conditions,
265 %     respectively
266 %   |epsilon| is a vector storing the fitted $\epsilon$ from the exit
267 %     temperature condition
268 %
269 % Internal variables:
270 %   |lowBnd| is the lower bound for fitting parameters for the exit
271 %     temperature condition
272 %   |numRuns| is the number of starting guess for the |optimProblem|
273 %     object
274 %   |modelFun| is the function handle to be fitted with the exit
275 %     temperature condition
276 %   |ms| is the |MultiStart| object for the exit temperature condition
277 %   |optimProbelm| is the optimization problem object for the exit
```

```
278 %     temperature condition
279 %   |param| is $[T_t,\epsilon]$ for the exit temperature condition
280 %   |paramGuess| is the inital guess for fitting parameters for the
281 %     exit temperature condition
282 %   |upBnd| is the upper bound for fitting parameters for the exit
283 %     temperature condition
284 %   |xData| is the independent variable for fitting the exit
285 %     temperature condition
286 %   |yData| is the dependent variable for fitting the exit temperature
287 %     condition
288
289     % Pre-allocate size of |Tt|
290     Tt = zeros(3,1);
291
292     % Cross-sectional average condition
293     Tt(1) = mean(TCross(z,R,dRdz,alpha,a,Pe,odeSolver));
294
295     % Full average condition
296     Tt(2)=mean(TFull(z,R,dRdz,alpha,a,Pe,odeSolver));
297
298     % Set up exit temperature optimization problem
299     modelFun=@(param,wz) AlphaFitExit(param,wz,St);
300     paramGuess=[-1/(2*St);log(eps)];
301     xData=wOfz(z,R,dRdz,a,Pe,odeSolver);
302     yData=alpha;
303     lowBnd=[-1/(2*St);log(eps)];
304     upBnd=[min(alpha);0]-eps;
305     optimProblem=createOptimProblem('lsqcurvefit','x0',paramGuess,...
306         'objective',modelFun,'lb',lowBnd,'ub',upBnd,'xdata',xData,...
307         'ydata',yData);
308     ms=MultiStart('Display','off');
309     numRuns=50;
310
311     % Exit temperature average condition
312     param=run(ms,optimProblem,numRuns);
```

```
313     Tt(3)=param(1);
314     epsilon=exp(param(2));
315 end
316
317 function plotTime=MakeFigure(cond,param,z,R,dRdz,St,alpha,Pe,minPe,...
318         odeSolver)
319 % |MakeFigure| plots the experimental data and fitted curves for a
320 %   particular fitting condition and computes the runtime of this
321 %   construction
322 % |MakeFigure| is called in the main script
323 % |MakeFigure| calls |TCross|, |TExit|, and |TFull|
324 %
325 % Input variables:
326 %   |a| is an array of the $a$ from (C.7)
327 %   |alpha| is an array of the $\alpha$ from (C.5)
328 %   |cond| is a string specifying which theshold condition is being
329 %     considered
330 %   |dRdz| is a function handle of $R^{\prime}(z)$
331 %   |minPe| is the minimum $\rm Pe$ plotted on the fitted curve
332 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
333 %     'ode15s')
334 %   |param| is a scalar of $T_t$ or vector of $[T_t,\epsilon]$
335 %     depending on the threshold condition
336 %   |Pe| is an array of the $\rm Pe$ from (C.6)
337 %   |St| is a scalar of $\rm St$ from (C.3)
338 %   |R| is a function handle of $R(z)$
339 %   |z| is a scalar of the $z$-coordinate where the theshold condtions
340 %     are imposed
341 %
342 % Output variable:
343 %   |plotTime| is a scalar of the runtime needed to construct the
344 %     figure
345 %
346 % Internal variables:
347 %   |aPlot| is a matrix of $a$ values to be plotted
```

```
348 %   |alphaLim| is a vector of the $\alpha$-limits of the figure
349 %   |alphaPlot| is a matrix of $\alpha$ values to be plotted
350 %   |PeLim| is a vector of the $\rm Pe$-limits of the figure
351 %   |PePlot| is a matrix of $\rm Pe$ values to be plotted
352 %   |plotRes| is a scalar to specify the resolution of the figure
353 %   |TPlot| is a matrix of $T$ values from (C.10), (C.11), or (C.12)
354 %     depending on what condition is being considered
355
356     tic;
357
358     plotRes=51;
359
360     % |contour| accepts matrices, |meshgrid| constructs these matrices
361     %   from vectors of $\alpha$ and $\rm Pe$ constructed with linspace
362     %   over the relevent range of these variables for |plotRes|
363     %   points
364     [alphaPlot,PePlot]=meshgrid(...
365         linspace(eps,max(alpha)+0.05,plotRes),...
366         linspace(minPe,max(Pe)+0.3,plotRes));
367
368     aPlot=(-1+sqrt(1+2*St.*alphaPlot))./(St*alphaPlot);
369
370     if strcmp(cond,'Cross')
371     % For cross-sectional average condtion
372         TPlot = TCross(z,R,dRdz,alphaPlot,aPlot,PePlot,odeSolver);
373     elseif strcmp(cond,'Full')
374     % For full average condtion
375         TPlot = TFull(z,R,dRdz,alphaPlot,aPlot,PePlot,odeSolver);
376     elseif strcmp(cond,'Exit')
377     % For exit temperature condtion
378         TPlot = TExit(param(2),z,R,dRdz,alphaPlot,aPlot,PePlot,...
379             odeSolver);
380     else
381     % Throw error for any other value of |cond|
382         error('Unknown theshold condition');
```

```
383     end
384
385     figure;
386     hold on;
387
388     % Plot data
389     scatter(alpha,Pe,'+k');
390
391     % Plot fitted curve
392     contour(alphaPlot,PePlot,TPlot,...
393         'LevelList',[param(1) param(1)],'LineColor','k');
394
395     % Plot specifications
396     xlim([0,max(alpha)+0.05]);
397     ylim([0,max(Pe)+0.3]);
398     xlabel('\fontname{cambria} \alpha','fontsize',18);
399     ylabel('\fontname{cambria} Pe','fontsize',18);
400
401     % Display fitting parameter values on figure
402     PeLim=get(gca,'ylim');
403     alphaLim=get(gca,'xlim');
404     text(alphaLim(1)+0.05*(alphaLim(2)-alphaLim(1)),...
405         PeLim(2)-0.1*(PeLim(2)-PeLim(1)),...
406         ['$T_{\mathrm{t}} = ',num2str(param(1),'%.6f'),'$'],...
407         'FontSize',18,'interpreter','latex');
408     if strcmp(cond,'Exit')
409         text(alphaLim(1)+0.05*(alphaLim(2)-alphaLim(1)),...
410             PeLim(2)-0.2*(PeLim(2)-PeLim(1)),...
411             ['$\epsilon = ',num2str(exp(param(2)),'%.6f'),'$'],...
412             'FontSize',18,'interpreter','latex');
413     end
414
415     hold off;
416
417     plotTime=toc;
```

```
418 end
419
420 function tCross=TCross(z,R,dRdz,alpha,a,Pe,odeSolver)
421 % |TCross| computes $\langle T\rangle(z)$ from (C.10)
422 % |TCross| is called by |FitData| and |MakeFigure|
423 % |TCross| calls |wOfz|
424 %
425 % Input variables:
426 %   |a| is an array of the $a$ from (C.7)
427 %   |alpha| is an array of the $\alpha$ from (C.5)
428 %   |dRdz| is a function handle of $R^{\prime}(z)$
429 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
430 %     'ode15s')
431 %   |Pe| is an array of the $\rm Pe$ from (C.6)
432 %   |R| is a function handle of $R(z)$
433 %   |z| is a scalar of the $z$-coordinate where the theshold condtions
434 %     are imposed
435 %
436 % Output variable:
437 %   |tCross| is an array of $\langle T\rangle(z)$ from (C.10)
438 %
439 % Internal variables:
440 %   |wz| is an array of $w(z)$
441
442     wz=wOfz(z,R,dRdz,a,Pe,odeSolver);
443     tCross=alpha.*(1+(2-a.*(1+exp(2*wz))./(2*wz)...
444         +((1-a).*(1-exp(2*wz)))./(2*wz.^2));
445 end
446
447 function tExit=TExit(logy,z,R,dRdz,alpha,a,Pe,odeSolver)
448 % |TExit| computes $T_{\rm p}(y,z)$ from (C.11)
449 % |TExit| is called by |FitData| and |MakeFigure|
450 % |TExit| calls |wOfz|
451 %
452 % Input variables:
```

```
453 %   |a| is an array of the $a$ from (C.7)
454 %   |alpha| is an array of the $\alpha$ from (C.5)
455 %   |dRdz| is a function handle of $R^{\prime}(z)$
456 %   |logy| is a scalar of $\log y$
457 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
458 %     'ode15s')
459 %   |Pe| is an array of the $\rm Pe$ from (C.6)
460 %   |R| is a function handle of $R(z)$
461 %   |z| is a scalar of the $z$-coordinate where the theshold condtions
462 %     are imposed
463 %
464 % Output variable:
465 %   |tExit| is an array of $T_{\rm p}(y,z)$ from (C.11)
466 %
467 % Internal variables:
468 %   |wz| is an array of $w(z)$
469
470     wz=wOfz(z,R,dRdz,a,Pe,odeSolver);
471     tExit=alpha.*(a.*(1-logy./wz)+(1-a).*(1-logy./wz).^2);
472 end
473
474 function tFull=TFull(z,R,dRdz,alpha,a,Pe,odeSolver)
475 % |TFull| computes $\bar{T}(z)$ from (C.12)
476 % |TFull| is called by |FitData| and |MakeFigure|
477 % |TFull| calls |wpsiOfz|
478 %
479 % Input variables:
480 %   |a| is an array of the $a$ from (C.7)
481 %   |alpha| is an array of the $\alpha$ from (C.5)
482 %   |dRdz| is a function handle of $R^{\prime}(z)$
483 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
484 %     'ode15s')
485 %   |Pe| is an array of the $\rm Pe$ from (C.6)
486 %   |R| is a function handle of $R(z)$
487 %   |z| is a scalar of the $z$-coordinate where the theshold condtions
```

```
488 %     are imposed
489 %
490 % Output variable:
491 %   |tFull| is an array of $\bar{T}(z)$ from (C.12)
492 %
493 % Internal variables:
494 %   |psiz| is an array of $\psi(z)$
495
496     [~,psiz]=wpsiOfz(z,R,dRdz,a,Pe,odeSolver);
497     tFull=alpha.*(1+psiz);
498 end
499
500 function [wz,psiz]=wpsiOfz(zEval,R,dRdz,a,Pe,odeSolver)
501 % |wpsiOfz| solves the differential equations for $w(z)$ and $\psi(z)$
502 % |wpsiOfz| is called by |TFull|
503 % |wpsiOfz| calls |Derivatives|, |dpsidz|, |dwdz|, |psiSplice|,
504 %   |wEvents|, and |wSplice|
505 %
506 % Input variables:
507 %   |a| is an array of the $a$ from (C.7)
508 %   |dRdz| is a function handle of $R^{\prime}(z)$
509 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
510 %     'ode15s')
511 %   |Pe| is an array of the $\rm Pe$ from (C.6)
512 %   |R| is a function handle of $R(z)$
513 %   |zEval| is a scalar of the $z$-coordinate
514 %
515 % Output variables
516 %   |psiz| is an array of $\psi(z)$
517 %   |wz| is an array of $w(z)$
518 %
519 % Internal variables:
520 %   |aExt| is a scalar $a$ from (C.7) to be used externally in
521 %     |dpsidz|, |dwdz|, and |wSplice|
522 %   |funs| is a vector of |[w;psi]|
```

```
523 %   |funs0| is a vector of the the inital condition |[w0;psi0]|
524 %   |funse| is a vector of |[we;psie]|, the value of |funs| when
525 %     |wEvents| occurs
526 %   |odeOptions| are the options for the ODE solver
527 %   |PeExt| is a scalar $\rm Pe$ from (C.6) to be used externally in
528 %     |dwdz| and |wSplice|
529 %   |psi0| is a scalar of the initial condition for the $\psi$ ODE:
530 %     $\psi(0)=0$
531 %   |w0| is a scalar of the initial condition for the $w$ ODE
532 %   |ze| is a vector of $z$ values when wEvents occurs
533 %   |zInterval| is the $z$-interval of the solution
534 %
535 % Nested functions:
536 %   |Derivatives| defines a vector $[w^{\prime}(z);\psi^{\prime}(z)]$
537 %     to be solved
538 %   |dpsidz| is defines $\psi^{\prime}(z)$ from (C.13)
539 %   |dwdz| is defines $w^{\prime}(z)$ from (C.14)
540 %   |psiSplice| deinfes the splice function for $\psi(z)$ given by
541 %     (C.18) if integration is stopped by |wEvents|
542 %   |wEvents| handles the event where $w\rightarrow0$; this is
543 %     problematic if a step in $z$ overshoots $w=0$ giving $w>0$, which
544 %     is not physically realizable
545 %   |wSplice| defines the splice function for $w(z)$ given by (C.19) if
546 %     integration is stopped by |wEvents| and then resumes integration
547
548     % Define initial conditions
549     % Since $w(z)=0$ is an (incorrect) solution to (C.14), |w0| is
550     %   taken to be a small negative number (determined by trial and
551     %   error)
552     w0=-1e-4;
553     psi0=0;
554     funs0=[w0;psi0];
555
556     % Pre-allocate size of |wz| and |psiz|
557     wz=zeros(size(Pe));
```

```
558     psiz=zeros(size(Pe));
559
560     % Create event to handle events where $w\rightarrow0$
561     odeOptions=odeset('Events',@wEvents);
562
563     % These loops cannot be vectorized due to the use of |contour| in
564     %   |MakeFigure|
565     for i=1:size(Pe,1)
566     % Iterate over rows of Pe
567         for j=1:size(Pe,2)
568         % Iterate over columns of Pe
569
570             aExt=a(i,j);
571             PeExt=Pe(i,j);
572             zInv=[0,zEval];
573
574             if strcmp(odeSolver,'ode45')
575                 [~,funs,ze,funse,~]=ode45(@Derivatives,zInv,funs0,...
576                     odeOptions);
577             elseif strcmp(odeSolver,'ode15s')
578                 [~,funs,ze,funse,~]=ode15s(@Derivatives,zInv,funs0,...
579                     odeOptions);
580             else
581                 errors('Unknown odeSolver');
582             end
583             % If w is too small to be accurately computed with the ODE
584             %   solver, integration is stoped and |wSplice| is used to
585             %   compute $w(z)%, this event is handled using
586             %   |odeOptions|
587
588             % Assign |wz(i,j)| and |psiz(i,j)| depending on if
589             %   |wEvents| occurred
590             if isempty(ze)
591             % If no event occurred, use ODE solution
592                 wz(i,j)=funs(end,1);
```

```
593                 psiz(i,j)=funs(end,2);
594             else
595             % If event occurred, use |wSplice| and |psiSplice|
596             %   solutions
597                 funse(abs(imag(funse))<1e-27)=...
598                     real(funse(abs(imag(funse))<1e-27));
599                 % Neglect imaginary part of the elements of |funse| if
600                 %   they are small
601                 if ~isreal(funse(1))
602                 % Throw error otherwise
603                     error('we is complex');
604                 end
605                 if ~isreal(funse(2))
606                 % Throw error otherwise
607                     error('psie is complex');
608                 end
609                 wz(i,j)=wSplice(zEval,ze,funse);
610                 psiz(i,j)=psiSplice(zEval,ze,funse);
611             end
612             if abs(imag(wz(i,j)))<1e-27
613             % Neglect imaginary part of |wz(i,j)| if it's small
614                 wz(i,j)=real(wz(i,j));
615             else
616             % Throw error otherwise
617                 error(['sz(',num2str(i),',',num2str(j),...
618                     ') is complex']);
619             end
620             if abs(imag(psiz(i,j)))<1e-27
621             % Neglect imaginary part of |psiz(i,j)| if it's small
622                 psiz(i,j)=real(psiz(i,j));
623             else
624             % Throw error otherwise
625                 error(['psiz(',num2str(i),',',num2str(j),...
626                     ') is complex']);
627             end
```

```
628         end
629     end
630
631     function derivatives=Derivatives(z,funs)
632     % |Derivatives| creates a vector of ODEs to be solved
633     % |Derivatives| is called in the main |wpsiOfz| and |wSplice|
634     % |Derivatives| calls |dpsidz| and |dwdz|
635     %
636     % Input variables:
637     %   |funs| is a vector of |[w;H]|
638     %   |zEval| is a scalar of the $z$-coordinate
639     %
640     % Output variable:
641     %   |derivatives| is a vector of ODEs to be solved
642
643         derivatives=zeros(2,1);
644         derivatives(1)=dwdz(z,funs(1));
645         derivatives(2)=dpsidz(funs(1));
646     end
647
648     function psiPrime=dpsidz(w)
649     % |dpsidz| sets up the ODE for $\psi$ to be solved
650     % |dpsidz| is called by |Derivatives|
651     %
652     % Input variable:
653     %   |w| is an array of $w(z)$
654     %
655     % Output variables:
656     %   |psiPrime| is an array of $\psi^{\prime}(z)$ from (C.13)
657     %
658     % Internal variables:
659     %   |denominator| is an array of the denominator from (C.13)
660     %   |numerator| is an array of the numerator from (C.13)
661     %
662     % Externally-scoped variables:
```

```
663     %   |aExt| is a scalar $a$ from (C.7) defined in the main |wpsiOfz|
664     %   |PeExt| is a scalar $\rm Pe$ from (C.6) defined in the main
665     %     |wpsiOfz|
666
667         numerator=(1-aExt).*(1-exp(2*w))+(2-aExt*(1+exp(2*w))).*w;
668         denominator=2*w.^2;
669         psiPrime=numerator./denominator;
670     end
671
672     function wPrime=dwdz(z,w)
673     % |dwdz| sets up the ODE for $\w$ to be solved
674     % |dwdz| is called by |Derivatives|
675     %
676     % Input variables:
677     %   |w| is an array of $w(z)$
678     %   |z| is an array of the $z$-coordinate
679     %
680     % Output variables:
681     %   |wPrime| is an array of $w^{\prime}(z)$ from (C.14)
682     %
683     % Internal variables:
684     %   |denominator| is an array of the denominator from (C.14a)
685     %   |eta| is an array of $\eta$ from (C.14b)
686     %   |numerator| is an array of the numerator from (C.14a)
687     %
688     % Externally-scoped variables:
689     %   |aExt| is a scalar $a$ from (C.7) defined in the main |wpsiOfz|
690     %   |PeExt| is a scalar $\rm Pe$ from (C.6) defined in the main
691     %     |wpsiOfz|
692
693         eta=(1-aExt)*w+PeExt*dRdz(z)/(4*R(z))*((1-aExt)*(1-exp(2*w))+...
694             (2-aExt*(1+exp(2*w))).*w);
695         numerator=8*PeExt.^(-1).*eta.*w;
696         denominator=2*(1-aExt)+(2-aExt).*w...
697             +exp(2*w).*(2*aExt*w.^2+(2-3*aExt)*w-2*(1-aExt));
```

```
698         wPrime=numerator./denominator;
699     end
700
701     function psi=psiSplice(zEval,z0,funs0)
702     % |psiSplice| computes $\psi(z)$ when the solver is stopped
703     %   |wEvents|
704     % |psiSplice| is called in the main |wpsiOfz| and |wSplice|
705     % |psiSplice| calls |dpsidz| and |wSplice|
706     %
707     % Input variables:
708     %   |funs0| is a vector of the the inital condition |[w0;psi0]|
709     %   |zEval| is an array of the $z$-coordinate at which $\psi(z)$
710     %     is evaluated
711     %   |z0| is a scalar the inital position for the $w(z)$ function
712     %     given by final value computed using the ode solver before the
713     %     solver was stopped by |wEevents|
714     %
715     % Output variables:
716     %   |psi| is an array of $\psi$ from (C.18)
717     %
718     % Internal variables:
719     %   |integrand| is a function handle of the integrand from (C.18)
720     %   |zDummy| is the dummy variable of integration
721
722         integrand=@(zDummy) dpsidz(wSplice(zDummy,z0,funs0(1)));
723         psi=funs0(2)+integral(integrand,z0,zEval,'ArrayValued',true);
724     end
725
726     function [value,isterminal,direction] = wEvents(z,funs)
727     % |wEvents| handles the event where computed value of $w$ goes to 0
728     %   by stopping the solver
729     % |wEvents| is called in the main |wpsiOfz| and |wSplice|
730     %
731     % Input variables:
732     %   |funs| is a vector of |[w;psi]|
```

```
733     %   |z| is a scalar of the $z$-coordinate
734     %
735     % Output variables:
736     %   |value| is the variable that goes to zero
737     %   |isterminal| stops the integration of the ode solver
738     %   |direction| specifies from which direction value goes to zero;
739     %     |direction = 0| corresponds to any direction
740
741         value = funs(1);
742         isterminal = 1;
743         direction = 0;
744     end
745
746     function [w,psi]=wSplice(zEval,z0,funs0)
747     % |wSplice| computes $w(z)$ when solver is stopped by |wEvents|
748     %   when $w$ goes to 0
749     % |wSplice| is called in |psiSplice|, the main |wpsiOfz|, and
750     %   |wSplice|
751     % |wSplice| calls |Derivatives| and |wSplice|
752     %
753     % Input variables:
754     %   |funs0| is a vector of the the inital condition |[w0;psi0]|
755     %   |zEval| is an array of the $z$-coordinate at which $\psi(z)$ is
756     %     evaluated
757     %   |z0| is a scalar the inital position for the $w(z)$ function
758     %     given by final value computed using the ode solver before the
759     %     solver was stopped by |wEevents|
760     %
761     % Output variables:
762     %   |psi| is an array of $\psi$ from (C.18)
763     %   |w| is an array of $w$ from (C.19)
764     %
765     % Internal variables:
766     %   |funseSplice| is a array of |[w,psi]| values when |wEvents|
767     %     occurs
```

```
768     %   |funsStart| is the final value of |[w;psi]| before restarting
769     %     the ODE solver after $w$ goes to 0
770     %   |funszSplice| is a array of |[w,psi]| values when |wEvents|
771     %     occurs
772     %   |psiStart| is the final value of $\psi$ before restarting the
773     %     ODE solver after $w$ goes to 0
774     %   |psizSplice| is a vector of $\psi$ values when |wEvents| occurs
775     %   |w0| is the inital condition for the $w(z)$ function given
776     %     by final value computed using the ODE solver before
777     %     the solver was stopped by |wEevents|
778     %   |wFun| is a function handle of $w(z)$ from (C.19)
779     %   |wRestart| is the target value of $w$ before resuming
780     %     integration after $w$ goes to 0
781     %   |wStart| is the final value of $w$ before resuing integration
782     %     after $w$ goes to 0
783     %   |wzSplice| is a vector of $w$ values when |wEvents| occurs
784     %   |zeSplice| is a vector of $z$ values when |wEvents| occurs
785     %   |zInvSplice| is the $z$-interval of the solution after
786     %     resuming integration after $w$ goes to 0
787     %   |zStart| is the final value of $z$ before resuming integration
788     %     after $w$ goes to 0
789     %
790     % Externally scoped variables:
791     %   |aExt| is a scalar $a$ from (C.7) defined in the main |wpsiOfz|
792     %   |odeOptions| are the options for the ODE solver defined in the
793     %     main |wpsiOfz|
794     %   |PeExt| is a scalar $\rm Pe$ from (C.6) defined in the main
795     %     |wpsiOfz|
796
797         % Define the value of $w$ which is far enough below 0 to
798         %   restart integration (determined by trial and error)
799         wRestart=log(1-1e-3);
800
801         w0=funs0(1);
802
```

```
803         % Given $z_0$ and $w_0$, this finds the next value of $z$ (and
804         %   corresponding value of $w$) where $w$ far enough below 0 to
805         %   restart integration;
806         %   |[w(zStart);H(zStart)]=[wStart;HStart]| is the new initial
807         %   condition for integration
808         wFun=@(z) -sqrt(w0^2+24*(1-aExt)/(2+aExt)*PeExt^(-1)*(z-z0));
809         [zStart,wStart]=fsolve(@(z) wRestart-wFun(z),z0,...
810             optimoptions('fsolve','Display','none'));
811         psiStart=psiSplice(zStart,z0,funs0);
812
813         % If |wStart>-1e-4|, then the ODE solver goes to the $w=0$
814         %   solution; in this case use the original initial condition
815         %   that was used as close enough to zero
816         if wStart>-1e-4
817             wStart=-1e-4;
818         end
819
820         funsStart=[wStart;psiStart];
821         zInvSplice=[zStart,zEval];
822
823         if strcmp(odeSolver,'ode45')
824             [~,funsSplice,zeSplice,funseSplice,~]=...
825                 ode45(@Derivatives,zInvSplice,funsStart,odeOptions);
826         elseif strcmp(odeSolver,'ode15s')
827             [~,funsSplice,zeSplice,funseSplice,~]=...
828                 ode15s(@Derivatives,zInvSplice,funsStart,odeOptions);
829         else
830             errors('Unknown odeSolver');
831         end
832         % If w is too small to be accurately computed with the ODE
833         %   solver, integration is stoped and |wSplice| is used to
834         %   compute $w(z)%, this event is handled using |odeOptions|
835
836         % Assign |wzSplice(i,j)| and |psizSplice(i,j)| depending on if
837         %   |wEvents| occurred
```

```
838         if isempty(zeSplice)
839         % If no event occurred, use ODE solution
840             [wzSplice,psizSplice]=funsSplice(end,:);
841         else
842         % If event occurred, use |wSplice| and |psiSplice| solutions
843             funseSplice(abs(imag(funseSplice))<1e-27)=...
844                 real(funseSplice(abs(imag(funseSplice))<1e-27));
845             % Neglect imaginary part of the elements of |funse| if they
846             %   are small
847             if ~isreal(funseSplice(1))
848             % Throw error otherwise
849                 error('weSplice is complex');
850             end
851             if ~isreal(funseSplice(2))
852             % Throw error otherwise
853                 error('psieSplice is complex');
854             end
855             [wzSplice,psizSplice]=wSplice(zEval,zeSplice,funseSplice);
856         end
857         if abs(imag(wzSplice))<1e-27
858         % Neglect imaginary part of |wz(i,j)| if it's small
859             wzSplice=real(wzSplice);
860         else
861         % Throw error otherwise
862             error(['wzSplice(',num2str(i),',',num2str(j),...
863                 ') is complex']);
864         end
865         if abs(imag(psizSplice))<1e-27
866         % Neglect imaginary part of |psiz(i,j)| if it's small
867             psizSplice=real(psizSplice);
868         else
869         % Throw error otherwise
870             error(['psizSplice(',num2str(i),',',num2str(j),...
871                 ') is complex']);
872         end
```

```
873         w=wzSplice;
874         psi=psizSplice;
875     end
876 end
877 
878 function wz=wOfz(zEval,R,dRdz,a,Pe,odeSolver)
879 % |wOfz| solves the differential equations for $w(z)$
880 % |wOfz| is called by |TCross| and |TExit|
881 % |wOfz| calls |dwdz|, |wEvents|, and |wSplice|
882 %
883 % Input variables:
884 %   |a| is an array of the $a$ from (C.7)
885 %   |dRdz| is a function handle of $R^{\prime}(z)$
886 %   |odeSolver| a string specifying which ode solver to use ('ode45' or
887 %     'ode15s')
888 %   |Pe| is an array of the $\rm Pe$ from (C.6)
889 %   |R| is a function handle of $R(z)$
890 %   |zEval| is a scalar of the $z$-coordinate
891 %
892 % Output variable:
893 %   |wz| is an array of $w(z)$
894 %
895 % Internal variables:
896 %   |aExt| is a scalar $a$ from (C.7) to be used externally in
897 %     |dpsidz|, |dwdz|, and |wSplice|
898 %   |odeOptions| are the options for the ODE solver
899 %   |PeExt| is a scalar $\rm Pe$ from (C.6) to be used externally in
900 %     |dwdz| and |wSplice|
901 %   |w| is a vector of $w$ values
902 %   |w0| is a scalar of the initial condition for the $w$ ODE
903 %   |we| is a vector of the value of $w$ when |wEvents| occurs
904 %   |ze| is a vector of $z$ values when wEvents occurs
905 %   |zInterval| is the $z$-interval of the solution
906 %
907 % Nested functions:
```

```
908 %   |dwdz| is defines $w^{\prime}(z)$ from (C.14)
909 %   |wEvents| handles the event where $w\rightarrow0$; this is
910 %     problematic if a step in $z$ overshoots $w=0$ giving $w>0$, which
911 %     is not physically realizable
912 %   |wSplice| defines the splice function for $w(z)$ given by (C.19) if
913 %     integration is stopped by |wEvents| and then resumes integration
914
915     % Define initial conditions
916     % Since $w(z)=0$ is an (incorrect) solution to (C.14), |w0| is
917     %   taken to be a small negative number (determined by trial and
918     %   error)
919     w0=-1e-4;
920
921     % Pre-allocate size of |wz|
922     wz=zeros(size(Pe));
923
924     % Create event to handle events where $w\rightarrow0$
925     odeOptions=odeset('Events',@wEvents);
926
927     % These loops cannot be vectorized due to the use of |contour| in
928     %   |MakeFigure|
929     for i=1:size(Pe,1)
930     % Iterate over rows of Pe
931         for j=1:size(Pe,2)
932         % Iterate over columns of Pe
933
934             aExt=a(i,j);
935             PeExt=Pe(i,j);
936             zInv=[0,zEval];
937
938             if strcmp(odeSolver,'ode45')
939                 [~,w,ze,we,~]=ode45(@dwdz,zInv,w0,odeOptions);
940             elseif strcmp(odeSolver,'ode15s')
941                 [~,w,ze,we,~]=ode15s(@dwdz,zInv,w0,odeOptions);
942             else
```

```
943                 error('Unknown odeSolver');
944             end
945             % If w is too small to be accurately computed with the ODE
946             %   solver, integration is stoped and |wSplice| is used to
947             %   compute $w(z)%, this event is handled using
948             %   |odeOptions|
949 
950             % Assign |wz(i,j)| and |psiz(i,j)| depending on if
951             %   |wEvents| occurred
952             if isempty(ze)
953             % If no event occurred, use ODE solution
954                 wz(i,j)=w(end);
955             else
956             % If event occurred, use |wSplice| solution
957                 if abs(imag(we))<1e-27
958                 % Neglect imaginary part of |we| if it's small
959                     we=real(we);
960                 else
961                 % Throw error otherwise
962                     error('we is complex');
963                 end
964                 wz(i,j)=wSplice(zEval,ze,we);
965             end
966 
967             if abs(imag(wz(i,j)))<1e-27
968             % Neglect imaginary part of |wz(i,j)| if it's small
969                 wz(i,j)=real(wz(i,j));
970             else
971             % Throw error otherwise
972                 error(['wz(',num2str(i),',',num2str(j),...
973                     ') is complex']);
974             end
975         end
976     end
977 
```

```
978     function wPrime=dwdz(z,w)
979     % |dwdz| sets up the ODE for $\w$ to be solved
980     % |dwdz| is called in the main |wOfz| and |wSplice|
981     %
982     % Input variables:
983     %   |w| is an array of $w(z)$
984     %   |z| is an array of the $z$-coordinate
985     %
986     % Output variables:
987     %   |wPrime| is an array of $w^{\prime}(z)$ from (C.14)
988     %
989     % Internal variables:
990     %   |denominator| is an array of the denominator from (C.14a)
991     %   |eta| is an array of $\eta$ from (C.14b)
992     %   |numerator| is an array of the numerator from (C.14a)
993     %
994     % Externally-scoped variables:
995     %   |aExt| is a scalar $a$ from (C.7) defined in the main |wOfz|
996     %   |PeExt| is a scalar $\rm Pe$ from (C.6) defined in the main
997     %     |wOfz|
998
999         eta=(1-aExt)*w+PeExt*dRdz(z)/(4*R(z))*...
1000            ((1-aExt)*(1-exp(2*w))+(2-aExt*(1+exp(2*w))).*w);
1001        numerator=8*PeExt.^(-1).*eta.*w;
1002        denominator=2*(1-aExt)+(2-aExt).*w...
1003            +exp(2*w).*(2*aExt*w.^2+(2-3*aExt)*w-2*(1-aExt));
1004        wPrime=numerator./denominator;
1005    end
1006
1007    function [value,isterminal,direction] = wEvents(z,w)
1008    % |wEvents| handles the event where computed value of $w$ goes to 0
1009    %   by stopping the solver
1010    % |wEvents| is called in the main |wpsiOfz|
1011    %
1012    % Input variables:
```

```
1013      %   |w| is a vector of $w$
1014      %   |z| is a scalar of the $z$-coordinate
1015      %
1016      % Output variables:
1017      %   |value| is the variable that goes to zero
1018      %   |isterminal| stops the integration of the ode solver
1019      %   |direction| specifies from which direction value goes to zero;
1020      %     |direction = 0| corresponds to any direction
1021
1022          value=w;
1023          isterminal=1;
1024          direction=0;
1025      end
1026
1027      function w=wSplice(zEval,z0,w0)
1028      % |wSplice| computes $w(z)$ when solver is stopped by |wEvents|
1029      %   when $w$ goes to 0
1030      % |wSplice| is called in the main |wpsiOfz| and |wSplice|
1031      % |wSplice| calls |dwdz| and |wSplice|
1032      %
1033      % Input variables:
1034      %   |w0| is a scalar of of the the inital condition $w_0$
1035      %   |zEval| is an array of the $z$-coordinate at which $\psi(z)$ is
1036      %     evaluated
1037      %   |z0| is a scalar the inital position for the $w(z)$ function
1038      %     given by final value computed using the ode solver before the
1039      %     solver was stopped by |wEevents|
1040      %
1041      % Output variable:
1042      %   |w| is an array of $w$ from (C.19)
1043      %
1044      % Internal variables:
1045      %   |w0| is the inital condition for the $w(z)$ function given
1046      %     by final value computed using the ODE solver before
1047      %     the solver was stopped by |wEevents|
```

```
1048      %   |weSplice| is a array of $w$ values when |wEvents| occurs
1049      %   |wFun| is a function handle of $w(z)$ from (C.19)
1050      %   |wRestart| is the target value of $w$ before resuming
1051      %     integration after $w$ goes to 0
1052      %   |wStart| is the final value of $w$ before resuing integration
1053      %     after $w$ goes to 0
1054      %   |wzSplice| is a vector of $w$ values when |wEvents| occurs
1055      %   |zeSplice| is a vector of $z$ values when |wEvents| occurs
1056      %   |zInvSplice| is the $z$-interval of the solution after
1057      %     resuming integration after $w$ goes to 0
1058      %   |zStart| is the final value of $z$ before resuming integration
1059      %     after $w$ goes to 0
1060      %
1061      % Externally scoped variables:
1062      %   |aExt| is a scalar $a$ from (C.7) defined in the main |wpsiOfz|
1063      %   |odeOptions| are the options for the ODE solver defined in the
1064      %     main |wpsiOfz|
1065      %   |PeExt| is a scalar $\rm Pe$ from (C.6) defined in the main
1066      %     |wpsiOfz|
1067
1068          % Define the value of $w$ which is far enough below 0 to
1069          %   restart integration (determined by trial and error)
1070          wRestart=log(1-1e-3);
1071
1072          % Given $z_0$ and $w_0$, this finds the next value of $z$ (and
1073          %   corresponding value of $w$) where $w$ far enough below 0 to
1074          %   restart integration; |w(zStart)=wStart| is the new initial
1075          %   condition for integration
1076          wFun=@(z) -sqrt(w0^2+24*PeExt^(-1)*(1-aExt)/(2+aExt)*(z-z0));
1077          [zStart,wStart]=fsolve(@(z) wRestart-wFun(z),z0,...
1078              optimoptions('fsolve','Display','none'));
1079
1080          % If |wStart>-1e-4|, then the ODE solver goes to the $w=0$
1081          %   solution; in this case use the original initial condition
1082          %   that was used as close enough to zero
```

```
1083        if wStart>-1e-4
1084            wStart=-1e-4;
1085        end
1086
1087        zInvSplice=[zStart,zEval];
1088
1089        if strcmp(odeSolver,'ode45')
1090            [~,wSpliceVec,zeSplice,weSplice,~]=ode45(@dwdz,...
1091                zInvSplice,wStart,odeOptions);
1092        elseif strcmp(odeSolver,'ode15s')
1093            [~,wSpliceVec,zeSplice,weSplice,~]=ode15s(@dwdz,...
1094                zInvSplice,wStart,odeOptions);
1095        else
1096            error('Unknown odeSolver');
1097        end
1098        % If w is too small to be accurately computed with the ODE
1099        %   solver, integration is stoped and |wSplice| is used to
1100        %   compute $w(z)%, this event is handled using |odeOptions|
1101
1102        % Assign |wzSplice| depending on if |wEvents| occurred
1103        if isempty(zeSplice)
1104        % If no event occurred, use ODE solution
1105            wzSplice=wSpliceVec(end);
1106        else
1107        % If event occurred, use |wSplice| solution
1108            if abs(imag(weSplice))<1e-27
1109            % Neglect imaginary part of |weSplice| if it's small
1110                weSplice=real(weSpliceVec);
1111            else
1112            % Throw error otherwise
1113                error('weSplice is complex');
1114            end
1115            wzSplice=wSplice(zEval,zeSplice,weSplice);
1116        end
1117        if abs(imag(wzSplice))<1e-27
```

```
1118          % Neglect imaginary part of |wz(i,j)| if it's small
1119              wzSplice=real(wzSplice);
1120          end
1121          w=wzSplice;
1122      end
1123  end
```