Reduced-Dimensional Coupled Electromagnetic, Thermal, and Mechanical Models of Microwave Sintering

Erin M. Kiley

A Dissertation

Submitted to the Faculty of
Worcester Polytechnic Institute

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematical Sciences

Department of Mathematical Sciences
Worcester Polytechnic Institute
April 2016

Reduced-Dimensional Coupled Electromagnetic, Thermal, and Mechanical Models of Microwave Sintering

by

Erin M. Kiley

A Dissertation
Submitted to the Faculty
of
Worcester Polytechnic Institute

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Mathematical Sciences

has been approved April 15, 2016 by:

Abstract

In recent years, sintering of powdered materials in microwaves has emerged as a manufacturing technique with many potential advantages over conventional sintering methods, including the possibility of faster processing and finer microstructure, along with the potential for vast energy savings. However, the technique remains on the level of laboratory studies and is underutilized in industry, mostly due to the difficulty of controlling the process: the intrinsically nonuniform temperature pattern that results from microwave heating routinely induces nonuniform mechanical deformation. Mathematical models and computer simulations can help to clarify the factors that influence this process and aid experimentalists in the design of efficient processing equipment. Although a number of modelling techniques have been reported to this end, they appear to inadequately represent the entire chain of related physical phenomena, which involves interaction of the electromagnetic field with the material, heat transfer, and mechanical deformation, each of which is coupled with both of the others, and all of which occur on different time scales. In this work, we present an original comprehensive mathematical formulation that accounts for the chain of physical processes comprising microwave sintering in one- and two-dimensional scenarios. We develop models for simulating the coupled electromagnetic, thermal, and mechanical phenomena at their appropriate time and spatial scales, and in addition, we account for the temperature and density dependence of the full set of thermal and dielectric properties of the material undergoing sintering. The electromagnetic and temperature fields are approximated using finite difference methods, and the mechanical problem is solved using the Master Sintering Curve representation of the density kinetics, which gives a way of accounting for the effect of microscale transport on the macroscopic property of relative density. For constant-rate sintering trials, we use the exponential integral to compute the work of sintering, which reduces computation time. The presented algorithms are all implemented and shown in `MATLAB` and `python`. Simulation of density and temperature evolution of the sintered sample shows processing times and shrinkage rates comparable to experimental results. This work lays a theoretical and computational foundation for modelling the general three-dimensional problem and computer-aided design of efficient sintering processes.

Acknowledgments

Many thanks, first, to my advisor at Worcester Polytechnic Institute, Prof. Vadim V. Yakovlev, who has worked hard to ensure that my time as a student has been well spent, who has introduced me to important people and interesting places, and whose guidance has been influential to my work and to my life for nearly ten years.

Merci to my co-advisor at the *Institut Polytechnique de Grenoble*, Prof. Didier Bouvard, who took time from his busy schedule as an administrator to supervise my work on this dissertation, and whose perspectives on engineering and materials science have notably enriched this work.

I am grateful for the assistance of Profs. Tilley, Walker, and Weekes, who served on my dissertation committee, and for that of Profs. Lurie, Tilley, and Walker, who served on my examination committee. They have helped to shape my future and that of so many other students throughout their careers.

Since first participating in WPI's REU program under her supervision in 2006, I have seen Prof. Suzanne Weekes as a role model whose tireless efforts have made real, positive change in the many and varied communities she contributes to, including, of course, the world of mathematics. I am particularly thankful for her encouragement and advice in the last two years.

To Sébastien Vaucher at the Swiss Federal Laboratories for Materials Science and Technology (EMPA), I express my gratitude for providing the resources I needed to be a productive member of the electromagnetic processing group during two summer visits, and for several inspired discussions of science, life, and all that comes between. His continuing support has meant a great deal.

Many thanks to the faculty and staff in the Department of Mathematical Sciences, and to my friends and classmates at Worcester Polytechnic Institute. I will miss the laughs shared with Bill, Grigor, Nguyenho, and Yiqing in Stratton Hall 205 (and am truly glad that so few of the pencils I've thrown at Grigor throughout the years actually hit their target).

The support of my family and friends, and of those I consider both, has also made the time spent on this work much easier. I am particularly grateful to my mother, who has always worked hard to provide a headstrong and sometimes haughty young girl with an environment in which she could learn and grow.

The material for this dissertation is based on research supported by the National Science Foundation Graduate Student Research Fellowship, the Arvid and Marietta Anderson Fellowship, and by the Chateaubriand Fellowship of the Office for Science & Technology of the Embassy of France in the United States.

Mr. Palomar's rule had gradually been changing: now he needed a great variety of models, whose elements could be combined in order to arrive at the one that would best fit reality, a reality that, for its own part, was always made up of many different realities, in time and in space.

—Italo Calvino[1]

[1]*Mr. Palomar*, Copyright 1983 by Giulio Einaudi *editore s.p.a.*, Torino. English translation copyright 1985 by Harcourt Brace Jovanovich, Inc.

Contents

List of Figures

List of Tables

Abbreviations

MSC	**M**aster **S**intering **C**urve
EM	**E**lectro**m**agnetic
MW	**M**icro**w**ave

Units of Interest

For an in-depth discussion of the various systems of units and dimensions used both historically and presently in electromagnetism, the reader is referred to [8]. In this dissertation, we use basic units from the *Système International d'Unités* (SI); that is, the four *fundamental dimensions* $\mathscr{M}$ (mass), $\mathscr{L}$ (length), $\mathscr{T}$ (time), and Θ (temperature) are expressed in meters (m), kilograms (kg), seconds (s), and Kelvin (K), respectively, and a fifth unit, $\mathscr{I}$, is the electric current—the amount of electric charge passing a point in an electric circuit per unit time—expressed in amperes (A). A sixth fundamental dimension, introduced to the SI in 1971, is the mole, which we denote with the corresponding basic unit mol, and which is defined as the amount of a chemical substance that contains as many elementary entities (*e.g.*, atoms, molecules, ions, electrons, or photons) as there are atoms in 12 grams of carbon-12 (this amount is expressed by Avogadro's constant N_A). The following table includes relevant units derived from these base units, described in terms of the base units and with their relationships to the other derived units given where this provides insight into their physical meaning. Also included are the natural and base-10 logarithmic units.

Symbol	Unit	Describes	Equivalent Units
C	coulomb	quantity of electricity carried in one second by a current of one ampere	C = A·s
dB	decibel	base-10 logarithmic unit: $L_{\mathrm{dB}} = 10\log_{10}\frac{P_1}{P_0}$	$1\ \mathrm{dB} = \frac{1\ \mathrm{Np}}{20\log_{10} e} \approx 0.115129\ \mathrm{Np}$
F	farad	1 F of capacitance produces a potential difference of 1 V when it has been charged by 1 C	$\mathrm{F} = \frac{\mathrm{A\cdot s}}{\mathrm{V}} = \frac{\mathrm{J}}{\mathrm{V}^2} = \frac{\mathrm{W\cdot s}}{\mathrm{V}^2} = \frac{\mathrm{C}}{\mathrm{V}} = \frac{\mathrm{C}^2}{\mathrm{J}} = \frac{\mathrm{C}^2}{\mathrm{N\cdot m}} = \frac{\mathrm{s}^2\cdot\mathrm{C}^2}{\mathrm{m}^2\cdot\mathrm{kg}} = \frac{\mathrm{s}}{\Omega} = \frac{\mathrm{s}^4\cdot\mathrm{A}^2}{\mathrm{m}^2\cdot\mathrm{kg}}$

Symbol	Unit	Describes	Equivalent Units
H	henry	inductance of a circuit is 1 H if the rate of change of current is 1 A/s and resulting EMF is 1 V	$H = \frac{J}{A^2} = \frac{Wb}{A} = \frac{V \cdot s}{A} = \frac{s^2}{F} = \Omega \cdot s = \frac{m^2 \cdot kg}{C^2} = \frac{m^2 \cdot kg}{s^2 \cdot A^2}$
J	joule	energy transferred (or work done) to an object when a force of one newton acts on that object in the direction of its motion through a distance of one meter	$J = \frac{kgm^2}{s^2}$
Np	neper	natural logarithmic unit: $L_{\mathrm{Np}} = \ln \frac{x_1}{x_2}$	$1\ \mathrm{Np} = \frac{20\ \mathrm{dB}}{\ln 10} \approx 8.6858897\ \mathrm{dB}$
N	newton	the force needed to accelerate one kilogram of mass at the rate of one meter per second squared in direction of the applied force	$N = \frac{kgm}{s^2}$
Ω	ohm	resistance between two points of a conductor when a constant potential difference of 1V, applied to these points, produces in the conductor a current of 1A, the conductor not being the seat of any EMF	$\Omega = \frac{1}{S} = \frac{s}{F} = \frac{V}{A} = \frac{J \cdot s}{C^2} = \frac{J}{s \cdot A^2} = \frac{m^2 \cdot kg}{s \cdot C^2} = \frac{kg \cdot m^2}{s^3 \cdot A^2}$
P	poise	viscosity; if a fluid with a viscosity of 1P is placed between two plates, and one plate is pushed sideways with a shear stress of 1Pa, it moves a distance equal to the thickness of the layer between the plates in 1s	$1P = 1\frac{g}{cms} = 0.1Pa \cdot s$
Pa	pascal	the pressure exerted by a force of magnitude one newton perpendicularly upon an area of one square meter	$Pa = \frac{N}{m^2} = \frac{kg}{m \cdot s^2}$
S	siemens	for a device with a conductance of 1S, the electric current through the device will increase by 1 A for every increase of 1V of electric potential across the device	$S = \frac{1}{\Omega} = \frac{A}{V} = \frac{A^2 s^3}{kgm^2}$
T	tesla	a particle carrying 1 C of charge passing through a magnetic field of 1 T at 1 m/s perpendicularly to the field experiences a force of 1 N.	$T = \frac{N \cdot s}{C \cdot m} = \frac{N}{A \cdot m} = \frac{V \cdot s}{m^2} = \frac{Wb}{m^2} = \frac{kg}{C \cdot s} = \frac{kg}{A \cdot s^2}$

Symbol	Unit	Describes	Equivalent Units
V	volt	1 V is the difference in electric potential across a wire when 1 W of power is dissipated by an electric current of 1 A	$V = \frac{W}{A} = A \cdot \Omega = \sqrt{W \cdot \Omega} = \frac{J}{C} = \frac{J}{A \cdot s} = \frac{N \cdot m}{A \cdot s} = \frac{N \cdot m}{C} = \frac{T \cdot m^2}{s} = \frac{kg \cdot m^2}{C \cdot s^2} = \frac{kg \cdot m^2}{A \cdot s^3}$
W	watt	measures power; 1 W is the rate at which work is done when 1 A of current flows through an electric potential difference of 1 V	$W = V \cdot A = \frac{V^2}{\Omega} = A^2 \cdot \Omega = \frac{J}{s} = \frac{kg \cdot m^2}{s^3}$
Wb	weber	a flux changing at 1 Wb/sec induces an EMF of 1 V across two open-circuited terminals	$Wb = V \cdot s = T \cdot m^2 = \frac{J}{A} = \frac{kg \cdot m^2}{A \cdot s^2}$

Physical Constants

Where final digits are given in parentheses, they represent the uncertainty in the last two digits of the value.

Constant Name	Symbol		Constant Value
Speed of light	c	$=$	$2.997\,924\,58 \times 10^{8}$ ms^{-2} (exact)
Boltzmann constant	k_B	$=$	$\frac{R}{N_A} = 1.38064852(79) \times 10^{-23}$ JK^{-1}
Avogadro's constant	N_A	$=$	$6.022140857(74) \times 10^{23}$ mol^{-1}
Gas constant[2]	R	$=$	$8.3144598(48)$ Jmol^{-1}K^{-1}
Electrical permittivity in a vacuum	ε_0	$=$	$\frac{10^7}{4\pi c^2} \approx 8.854 \times 10^{-12}$ Fm^{-1}
Magnetic permeability in a vacuum	μ_0	$=$	$4\pi \times 10^{-7} \approx 1.256 \times 10^{-6}$ Hm^{-1}
Ratio of circle circumference to diameter	π	$\approx$	$3.141592\ldots$

Symbols

Following the convention in [9], the dimensions which appear in this table in script-style capital letters denote the *fundamental dimensions* $\mathscr{M}$ (mass), $\mathscr{L}$ (length), $\mathscr{T}$ (time), Θ (temperature), and $\mathscr{I}$ (electrical current). The units used here and in the text to quantify these dimensions are written with Roman-style letters, and are included so that the reader might easily relate the fundamental dimensions with standard units often found in textbooks and handbooks. A discussion of units in general may be found in the preface to the Units of Interest section.

Symbol	Name	Unit	Dimensions
a	grain radius	m	$\mathscr{L}$
a	width of waveguide cross-section	m	$\mathscr{L}$
A	surface/interfacial area	m^2	$\mathscr{L}^2$
b	height of waveguide cross-section	m	$\mathscr{L}$
B	piecewise smooth surface bounding a volume		
$\vec{B}$	magnetic field	T	
c_p	specific heat capacity	JK^{-1}	$\mathscr{L}^2\mathscr{T}^{-2}\Theta^{-1}$
C	closed curve in space		
$\vec{D}$	electric displacement field	Cm^{-2}	$\mathscr{I}\mathscr{T}\mathscr{L}^{-2}$
$\mathrm{d}a$	surface element		
$\mathrm{d}\vec{\ell}$	line element		
$D_{\hat{s}}$	directional derivative in the direction of $\hat{s}$		
$\mathrm{d}V$	volume element		
D_b	coefficient for grain boundary diffusion in MSC model	(unitless)	
D_v	coefficient for volume diffusion in MSC model	(unitless)	
e	specific internal energy	J	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-2}$
$E_1(x)$	exponential integral function		

Symbol	Name	Unit	Dimensions
$Ei(x)$	exponential integral function		
E_{inc}	electric field incident at port-side wall	Vm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{I}^{-1}\mathscr{T}^{-3}$
$\vec{E}$	electric field	Vm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{I}^{-1}\mathscr{T}^{-3}$
$\vec{E}_\tau$	tangential component of electric field	Vm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{I}^{-1}\mathscr{T}^{-3}$
$\mathcal{E}$	phasor form of electric field	Vm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{I}^{-1}\mathscr{T}^{-3}$
$\vec{F}$	Lorentz force	N	$\mathscr{M}\mathscr{L}\mathscr{T}^{-2}$
f	frequency of electromagnetic wave	Hz	$\mathscr{T}^{-1}$
$f(\vec{x},t)$	source term in heat equation	Ks^{-1}	$\Theta\mathscr{T}^{-1}$
f_c	cutoff frequency of electromagnetic wave	Hz	$\mathscr{T}^{-1}$
$f := 1-\rho_{\text{rel}}$	porosity	(unitless)	
g_1	normalized shear viscosity for open porosity	P	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-1}$
g_2	normalized shear viscosity for closed porosity	P	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-1}$
g_m	amplification factor	(unitless)	
G	shear viscosity	P	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-1}$
G_{lin}	parameter in viscosity equations	(unitless)	
$G(\rho)$	mean grain diameter	m	$\mathscr{L}$
h_1	distance along x-axis from lower computational domain boundary to nearest air/insulation interface	m	$\mathscr{L}$
h_2	distance along x-axis from lower computational domain boundary to farthest insulation/air interface	m	$\mathscr{L}$
H	height of microwave cavity or computational domain (Cartesian coordinates)	m	$\mathscr{L}$
$\vec{H}$	magnetic field intensity	Am^{-1}	$\mathscr{I}\mathscr{L}^{-1}$
$\mathcal{H}$	phasor form of magnetic field	Am^{-1}	$\mathscr{I}\mathscr{L}^{-1}$
i	imaginary unit $\sqrt{-1}$		
I_{enc}	electric current	A	$\mathscr{I}$
$\vec{J}$	volume density of total current	Am^{-2}	$\mathscr{I}\mathscr{L}^{-2}$
$\vec{J}_{\text{bound}}$	volume density of bound current	Am^{-2}	$\mathscr{I}\mathscr{L}^{-2}$
$\vec{J}_{\text{free}}$	volume density of free current	Am^{-2}	$\mathscr{I}\mathscr{L}^{-2}$

Symbol	Name	Unit	Dimensions
$\vec{J}_{\text{ind}}$	volume density of induced current	Am^{-2}	$\mathscr{I}\mathscr{L}^{-2}$
$\vec{J}_{\text{pol}}$	volume density of polarization current	Am^{-2}	$\mathscr{I}\mathscr{L}^{-2}$
$\vec{\mathcal{J}}_s$	electric source current	Am^{-2}	$\mathscr{I}\mathscr{L}^{-2}$
k	thermal conductivity	$\text{Wm}^{-1}\text{K}^{-1}$	$\mathscr{M}\mathscr{L}\mathscr{T}^{-3}\Theta^{-1}$
k_1	distance along x-axis from lower computational domain boundary to nearest insulation/material interface	m	$\mathscr{L}$
k_2	distance along x-axis from lower computational domain boundary to farthest material/insulation interface	m	$\mathscr{L}$
k_1	normalized bulk viscosity for open porosity	P	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-1}$
k_2	normalized bulk viscosity for closed porosity	P	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-1}$
$k_c := \omega^2\varepsilon\mu$	cutoff constant of wave propagation	(unitless)	
K	bulk viscosity	P	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-1}$
K_{lin}	parameter in viscosity equations	(unitless)	
ℓ_1	distance along z-axis from left-hand computational domain boundary to nearest air/insulation interface	m	$\mathscr{L}$
ℓ_2	distance along z-axis from left-hand computational domain boundary to farthest insulation/air interface	m	$\mathscr{L}$
L	length of microwave cavity or computational domain (Cartesian coordinates)	m	$\mathscr{L}$
m	mass	g	$\mathscr{M}$
m_1	distance along z-axis from left-hand computational domain boundary to nearest insulation/material interface	m	$\mathscr{L}$
m_2	distance along z-axis from left-hand computational domain boundary to farthest material/insulation interface	m	$\mathscr{L}$
M	number of spatial nodes in x-direction		
$\vec{M}$	magnetization density	Am^{-1}	$\mathscr{I}\mathscr{L}^{-1}$

Symbol	Name	Unit	Dimensions
N	number of spatial nodes in z-direction		
$\mathbb{N}$	set of natural numbers, $\mathbb{N} = \{1, 2, 3, \cdots\}$		
$\vec{n}$	outward-pointing vector normal to a surface		
$\hat{n}$	unit outward-pointing vector normal to a surface		
q	point electrical charge	C	
$\vec{q}$	heat flux vector	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
Q	activation energy	J	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-2}$
P	power	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
P_{diss}	power dissipated inside object	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
P_J	Joule power	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
P_o	power flowing outward through surface	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
P_s	power delivered by sources inside surface	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
P_v	power delivered to charge density	W	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-3}$
p_{ex}	external pressure	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
Δp	change in hydrostatic pressure	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
$\vec{P}$	electric polarization density	Cm^{-2}	$\mathscr{I}\mathscr{T}\mathscr{L}^{-2}$
$\mathbb{R}$	set of real numbers, $\mathbb{R} = (-\infty, \infty)$		
$\hat{s}$	unit vector pointing in any direction		
S	surface in space		
$\mathcal{S} := \mathcal{E} \times \mathcal{H}^*$	Poynting vector		
T_0	initial temperature	K	Θ
T_{amb}	ambient temperature	K	Θ
u	temperature	K	Θ
U	factor describing effect of grain rearrangement	(unitless)	
v	volume fraction of mixture component	(unitless)	
$v_{\text{old}}, v_{\text{new}}$	volume	m^3	$\mathscr{L}^3$
$\vec{v}$	velocity	ms^{-1}	$\mathscr{L}\mathscr{T}^{-1}$
v_p	phase velocity of electromagnetic wave	ms^{-1}	$\mathscr{L}\mathscr{T}^{-1}$

Symbol	Name	Unit	Dimensions
V	volume in space		
W	width of microwave cavity (Cartesian coordinates)	m	$\mathscr{L}$
W_e	time-average energy stored in electric field	J	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-2}$
W_m	time-average energy stored in magnetic field	J	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-2}$
$\mathbb{Z}$	set of integers, $\mathbb{Z} = \{\ldots,-3,-2,-1,0,1,2,3,\ldots\}$		
α	constant parameter determining deviation of viscosities from linearity	(unitless)	
$\beta := \frac{\pi}{L}$	propagation constant of incident waves	m^{-1}	$\mathscr{L}^{-1}$
γ	surface/interfacial energy	J	$\mathscr{M}\mathscr{L}^2\mathscr{T}^{-2}$
γ_s	surface/interfacial tension	N	$\mathscr{M}\mathscr{L}\mathscr{T}^{-2}$
γ_b	grain boundary tension	N	$\mathscr{M}\mathscr{L}\mathscr{T}^{-2}$
Γ	curve bounding cross-section of waveguide		
$\Gamma(a,z)$	incomplete gamma function		
Γ_b	scaling parameter in MSC model	(unitless)	
Γ_{port}	port-side waveguide boundary		
Γ_v	scaling parameter in MSC model	(unitless)	
Γ_{wall}	waveguide wall boundary		
δ	width of grain boundary	m	$\mathscr{L}$
δ_{ij}	Kroenecker delta		
δD_b	product of grain boundary diffusion coefficient and the thickness of the grain boundary	m	$\mathscr{L}$
δD_{b0}	pre-exponential factor in Arrhenius-type expression for ΔD_b	m	$\mathscr{L}$
δD_s	product of grain surface diffusion coefficient and the thickness of the grain boundary	m	$\mathscr{L}$
δD_{s0}	pre-exponential factor in Arrhenius-type expression for ΔD_b	m	$\mathscr{L}$
$\underline{\delta}$	unit tensor		
$\tan\delta := \frac{\varepsilon''}{\varepsilon'}$	loss tangent	(unitless)	

Symbol	Name	Unit	Dimensions
ε	$\varepsilon := \varepsilon' - i\varepsilon''$ electrical permittivity	$\mathrm{Fm^{-1}}$	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
ε_{eff}	effective electrical permittivity of a mixture	$\mathrm{Fm^{-1}}$	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
ε_L	Wiener's lower limit on electrical permittivity	$\mathrm{Fm^{-1}}$	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
$\varepsilon_r := \frac{\varepsilon}{\varepsilon_0}$	relative electrical permittivity	(unitless)	1
ε_U	Wiener's upper limit on electrical permittivity	$\mathrm{Fm^{-1}}$	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
ε'	dielectric constant	$\mathrm{Fm^{-1}}$	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
ε''	loss factor	$\mathrm{Fm^{-1}}$	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
$\underline{\varepsilon}$	strain tensor	(unitless)	
$\underline{\dot{\varepsilon}}$	strain rate tensor	$\mathrm{m^{-1}}$	$\mathscr{L}^{-1}$
$\underline{\dot{\varepsilon}^e}$	elasticity strain rate tensor	$\mathrm{m^{-1}}$	$\mathscr{L}^{-1}$
$\underline{\dot{\varepsilon}^t}$	thermal expansion strain rate tensor	$\mathrm{m^{-1}}$	$\mathscr{L}^{-1}$
$\underline{\dot{\varepsilon}^s}$	free sintering strain rate tensor	$\mathrm{m^{-1}}$	$\mathscr{L}^{-1}$
$\underline{\dot{\varepsilon}^v}$	viscous deformation strain rate tensor	$\mathrm{m^{-1}}$	$\mathscr{L}^{-1}$
θ	parameter ranging from 0 to 1	(unitless)	
Θ	work of sintering parameter in MSC model	$\mathrm{sK^{-1}}$	$\mathscr{T}\Theta^{-1}$
κ	thermal diffusivity	$\mathrm{m^2s^{-1}}$	$\mathscr{L}^2\mathscr{T}^{-1}$
λ	wavelength in free space	m	$\mathscr{L}$
λ_c	cutoff wavelength in	m	$\mathscr{L}$
λ_g	wavelength in waveguide	m	$\mathscr{L}$
μ	magnetic permeability	$\mathrm{Hm^{-1}}$	$\mathscr{L}\mathscr{M}\mathscr{T}^{-2}\mathscr{I}^{-2}$
ρ	mass density	$\mathrm{gm^{-3}}$	$\mathscr{M}\mathscr{L}^{-3}$
$\dot{\rho}$	free sintering densification rate	$\mathrm{gm^{-3}s^{-1}}$	$\mathscr{M}\mathscr{L}^{-3}\mathscr{T}^{-1}$
ω	angular frequency of electromagnetic wave	Hz	$\mathscr{T}^{-1}$
ω_c	angular cutoff frequency of electromagnetic wave	Hz	$\mathscr{T}^{-1}$
Ω	cross-section of waveguide		
Ω	atomic (molecular, molar) volume	$\mathrm{m^3mol^{-1}}$	$\mathscr{L}^{3-1}$
ρ_{bulk}	mass density of bulk material	$\mathrm{gm^{-3}}$	$\mathscr{M}\mathscr{L}^{-3}$
ρ_{cl}	relative density at which transition from open to closed porosity occurs	(unitless)	
$\rho_{\text{rel}} := \frac{\rho}{\rho_{\text{bulk}}}$	relative mass density of powder material	(unitless)	

Symbol	Name	Unit	Dimensions
ρ	volume density of total charge	Cm^{-3}	$\mathscr{I}\mathscr{T}\mathscr{M}^{-3}$
ρ_{bound}	volume density of bound charge	Cm^{-3}	$\mathscr{I}\mathscr{T}\mathscr{M}^{-3}$
ρ_{free}	volume density of free charge	Cm^{-3}	$\mathscr{I}\mathscr{T}\mathscr{M}^{-3}$
σ	electrical conductivity	Sm^{-1}	
σ_e	von Mises stress equivalent	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
σ_m	mean (hydrostatic) stress	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
σ_s	sintering stress	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
$\underline{\sigma}$	stress tensor	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
$\underline{\sigma'}$	deviatoric stress tensor	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
$\overline{\sigma}$	effective stress	Pa	$\mathscr{M}\mathscr{L}^{-1}\mathscr{T}^{-2}$
ϕ	phase reference of wave	rad	
ψ	dihedral angle	rad	
Ψ	expression accounting for microstructural and material properties in MSC model		
ω	angular frequency	$\mathrm{sec}^{-1}\mathrm{rad}$	$\mathscr{T}^{-1}$

This dissertation is dedicated to Patrick Michael Kiley (1955–2014), whom I am lucky and proud to have had as a father, and whose absence I feel only more acutely with each passing day.

Chapter 1

Introduction

Sintering is a process by which several different transport mechanisms influence the microstructure of a granular material during the course of thermal processing, causing the welding of particles and the growth of interfaces or "necks" between particles. For solid particulate materials that do not undergo phase changes, we refer to the process as "solid-phase sintering"[1] . Potters have used this technique for millennia in kiln-firing of ceramic materials, and in the 1940s, tools were commonly forged via sintering of sponge iron [10]. Today, sintering is used in forming various ceramic and metal parts and materials from pre-compressed powders [10, 11], and is finding a keen use in manufacturing technologies for biomedical materials, including scaffolds for bone tissues [12] and dental implants [13], and is also used in the creation of metal foams [14]. The range of possible applications of sintering in manufacturing is broad, but this study is motivated by the fact that sintering has not fully realized its potential as an innovative manufacturing technology. Among the potential advantages sintering has over other forms of thermal processing is the possibility of creating materials whose thermal and mechanical properties do not occur in nature [15–17].

In recent years, strong interest has developed in sintering dielectric and metal powders using microwaves as the heat source [18–20]. This manufacturing technique may prove fundamentally different from sintering in conventional ovens, with key differences including faster processing, greater shrinkage of metal powder compacts, finer microstructure and otherwise differing properties of the resulting new materials, and—in well-designed systems—the potential for vast energy savings [20–25].

Despite these promising results and the strong growing interest in the use of microwaves, the technique remains underutilized in industry, due in part to the difficulty of controlling the process, as the intrinsically nonuniform temperature pattern that results from microwave heating also induces nonuniform mechanical deformation. As a result, the design of systems for carrying out

[1]Liquid-phase sintering, in which parts of solid materials may temporarily take on a liquid phase, can result in a cheaper and easier control over the microstructure but also frequently leads to unpredictable material properties [10]. Transient liquid-phase sintering (in which the liquid disappears as densification progresses), and viscous flow sintering (when the volume fraction of liquid is high, and densification is achieved without shape change of the grains) are other applications of conventional and microwave sintering, but this study is primarily concerned with solid-phase sintering.

microwave sintering currently involves extensive and repetitive experimentation, in which reproducibility of results may be difficult to achieve [26]. Mathematical models and computer simulations employing those models would offer the possibility to better study this process, which may lead to improvements of the theory of microwave sintering. A comprehensive model covering all relevant physical phenomena entailed by the process would help to test different geometrical configurations of microwave sintering applicators and, if judiciously used in the design of such systems, could provide a means of rectifying the challenges that have prevented microwave sintering from fulfilling its potential as a green and efficient industrial manufacturing technology.

A number of modelling techniques to treat microwave sintering have been reported [27–33], but these are not comprehensive models, as each gives insufficient treatment of at least one of the multiphysics processes involved with sintering, and each ignores the dependence of dielectric and thermal properties on density and temperature. As a result, various aspects of microwave sintering have not yet received the especially careful mathematical treatment they warrant, including the strong multiphysical coupling, the vastly different time and spatial scales on which the processes evolve, and the impact of material parameters on the course of sintering.

In this study, we demonstrate the treatment of some of these issues by providing a simplified, mathematically consistent, dimensionally reduced model of microwave sintering that is implemented numerically in a single iterative routine involving each of the key multiphysics processes. The physical configurations of the scenarios described by the one- and two-dimensional models of microwave cavities loaded with a sample of the material undergoing sintering, surrounded by insulation, can be seen in Figures 1.1 and 1.2, respectively. The model presented herein, together with its computational implementation, addresses the multiphysics nature of microwave sintering by coupling the electromagnetic, thermal, and mechanical deformation portions of the problem. This routine does not require the transfer of data between the different meshes that various solution methods relying on conceptually different solvers for each portion of the problem require. Our routine addresses the multi-scale nature of the mechanical deformation problem by relying on an auxiliary Master Sintering Curve (MSC) model whose output is a parameter on which the density of the material depends. For the first time, this model considers the dielectric and thermal properties of the material undergoing microwave sintering as functions not only of the material's temperature, but of the material's relative density. The multi-scale nature of the problem in time is resolved either by assuming a time-harmonic electric field and using the Helmholtz equation to represent electromagnetic phenomena, or by simulating the transient evolution of the electric field using the wave equation until the length of one thermal time step passes.

The remainder of the dissertation is structured as follows. In Chapter 2, we provide a review of the electromagnetic problem, including a description of Maxwell's equations, the electromagnetic wave equation, and the Helmholtz equation as related to the problem of propagation within a waveguide, and we formulate corresponding initial boundary value problems to describe the electromagnetic phenomenon as it relates to microwave sintering in practical equipment. In Chapter 3, we review the classical derivation of the heat equation and discuss how its source term, when describing the microwave heating problem, depends on the magnitude of the electric field; we then formulate the initial boundary value problem describing thermal diffusion in the insulation and

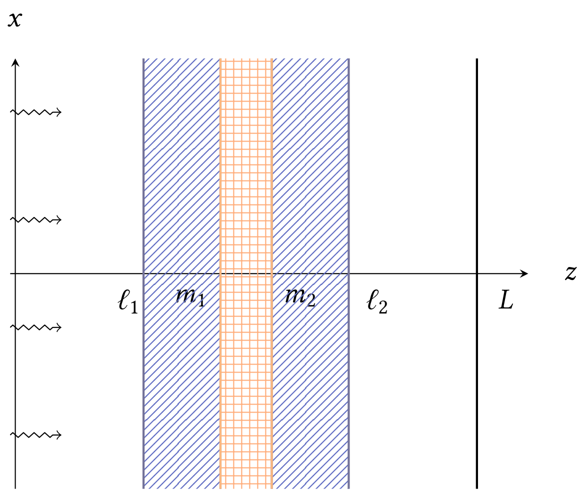


Figure 1.1: Physical scenario for the one-dimensional model of microwave sintering, where the space occupied by apricot-colored grid lines is assumed to be filled by the material undergoing sintering, the space occupied by diagonal blue lines is assumed to be filled by insulation, and the remainder of the cavity is assumed to be filled by air. The source of microwaves is on the left-hand boundary.

material in the one- and two-dimensional domains. In Chapter 4, we provide a review of the driving forces of sintering and their contribution to densification, and the MSC method is shown as a way of synthesizing the described energy considerations into a formulation of the density kinetics along a known temperature cycle.

In Chapter 5, we present two novel methods of accounting for the density dependence of thermal and dielectric material properties throughout the course of simulated sintering, one of which relies on the inversion of some classical and contemporary formulas for determining effective dielectric properties of mixtures, along with other classical approximation formulas for the thermal properties. In Chapter 6, we describe an original experiment in assessing the applicability of the classical and contemporary mixture models to the case where the materials involved are comprised of metal powders. In Chapter 7, we discuss the solution of the wave equation and Helmholtz equation using finite difference methods, and compare our results to those obtained using the finite element method, analytical methods, and to results that exist in literature. In Chapter 8, we discuss the finite difference method for solving the heat equation, providing details of our implementation. In Chapter 9, we discuss the way that our solver finds the Master Sintering Curve that describes the relation of density kinetics to the thermal cycle, and we propose a novel use of the exponential integral function for speeding up the determination of activation energy in certain heating scenarios. We also discuss the use of the Master Sintering Curve in determining density and volume change during processing, and how to account for changing geometry in our coupled model.

In Chapter 10, we describe the way in which our solvers are incorporated into one coupled

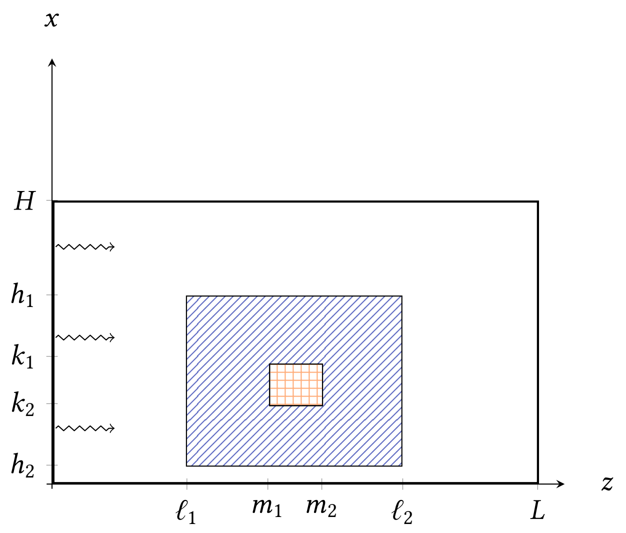


Figure 1.2: Physical scenario for the two-dimensional model of microwave sintering, where the space occupied by apricot-colored grid lines is assumed to be filled by the material undergoing sintering, the space occupied by diagonal blue lines is assumed to be filled by insulation, and the remainder of the cavity is assumed to be filled by air. The source of microwaves is on the left-hand boundary.

routine for transient simulation of microwave sintering. We provide several computational examples in Chapter 11 as an illustration of the model's operation, via one- and two-dimensional simulations of the microwave sintering of zirconia (ZiO_2) surrounded by alumina (Al_2O_3) insulation. Some concluding remarks addressing the theoretical and computational foundation that this work lays for modelling the general three-dimensional problem, and prospects on computer-aided design of efficient sintering processes, are given in Chapter 12, alongside some comments on possible future directions for this work.

Chapter 2

The Electromagnetic Problem

Within the microwave cavity in which sintering experiments are performed, the rapidly changing electric field results in a field of power dissipated into the sample, which in turn results in temperature change. In this chapter, we present the physical considerations leading to Maxwell's equations, and use constitutive equations to arrive at the wave equations and the Helmholtz equations describing the electric and magnetic fields.

2.1 Maxwell's Equations and Constitutive Relations

In 1873, James Clerk Maxwell first described the electromagnetic wave propagation phenomenon [34], synthesizing a basis for the foundations of modern electromagnetic theory from existing fragments of empirical and theoretical evidence developed by Carl Friedrich Gauss [35, 36], André-Marie Ampère [37], and Michael Faraday [38–40]. Oliver Heaviside [41–43] would later use vector calculus to combine and simplify Maxwell's contributions into four physical laws that correspond to the ones that have come to be known as "Maxwell's equations," which we present and discuss in the context of modelling electric and magnetic fields inside of a microwave heating cavity.

Gauss's Law for Electric Fields

Gauss's law for electric fields arises from consideration of a point electrical charge q, surrounded by an arbitrary closed surface S. Such a point charge will produce an electric field $\vec{E}$, and the flux of that field passing through S is proportional to the charge. This holds equally true when considering not only a single point charge, but the total sum of charges (both bound and free) enclosed by S, as characterized by the volume integral of the continuous charge density function $\rho(\vec{x})$. We thus write Gauss's law for electric fields in its integral form as in [8]:

$$\oint_S \vec{E} \circ \hat{n} \, \mathrm{d}a = \varepsilon^{-1} q = \varepsilon^{-1} \int_V \rho(\vec{x}) \, \mathrm{d}V, \tag{2.1}$$

where V is the volume enclosed by S, $\hat{n}$ is an outward-pointing unit vector normal to S originating at the center of the infinitesimal area element da, and where the proportionality factor ε is referred to as the *permittivity*. The permittivity is, most generally, a function $\varepsilon : \mathbb{R}^3 \to \mathbb{R}^3$ of the spatial variable, and depends on the medium in which the wave propagates. In case we consider the scenario in a vacuum, ε represents the constant vacuum permittivity ε_0, whose value is given in the List of Constants. In matter, the permittivity depends on temperature and density, and varies with the frequency of radiation as well. In isotropic media—that is, in case the polarization and magnetization of the material do not depend on the directions—the permittivity is simply a real-valued function, *i.e.*, $\varepsilon : \mathbb{R}^3 \to \mathbb{R}$, but in anisotropic media, such as crystal structures and ionized gases, in order to account for the more complicated relation between the various field components, ε is a rank-two tensor with nondiagonal matrix representation.

Applying the divergence theorem[1] to the electric flux on the left-hand side of Equation 2.1, we obtain

$$\int_V \vec{\nabla} \circ \vec{E} \, \mathrm{d}V = \varepsilon^{-1} \int_V \rho(\vec{x}) \, \mathrm{d}V,$$

and for equality to hold over all surfaces and the volumes they enclose, it must be true in general that

$$\nabla \circ \vec{E} = \varepsilon^{-1} \rho. \tag{2.2}$$

Equation 2.2 is a simple way of writing Gauss's law for electric fields.

However, if the charge q or charge density ρ and surrounding surface S exist within matter, as opposed to within free space, then the notion of *bound charge* exists, and contributes to q or ρ as well as the *free charge*[2] in any metals or through free space, so it may result—as we will see—in a more fundamental statement of Gauss's Law when we write

$$\rho = \rho_{\text{free}} + \rho_{\text{bound}}. \tag{2.3}$$

Bound charge occurs within matter when electrons are displaced inside their atoms by the presence of the electric field; these electrons cannot move freely through the matter, but still, the sum of the microscopic shifts of the electrons within each atom results in a macroscopic change in the total distribution of charge, and it is this quantity of charge that is referred to as the bound charge [46]. It is differences in the electric polarization $\vec{P}$ of a material that generate accumulation of charge within the material, with the resulting volume density of bound charge given by the quantity

$$\rho_{\text{bound}} = -\vec{\nabla} \circ \vec{P}. \tag{2.4}$$

[1]The Divergence Theorem, also known as Gauss's theorem or Ostrogradsky's theorem, states that for a continuously differentiable vector field $\vec{A}(\vec{x}, t)$ and for a compact volume V enclosed within a piecewise smooth surface S, $\int_V \vec{\nabla} \circ \vec{A} \, \mathrm{d}V = \oint_S \vec{A} \circ \hat{n} \, \mathrm{d}a$, where $\hat{n}$ is an outward-pointing unit vector normal to S originating at the center of the infinitesimal area element da [44].

[2]The free charge density ρ_{free} is sometimes referred to in the literature as the *conductive charge density* ρ_c [45].

The proof of this statement is given in [47]. Using Equation 2.4 and Equation 2.3 together, Equation 2.2 becomes

$$\vec{\nabla} \circ \vec{E} = \left(\rho_{\text{free}} + \rho_{\text{bound}}\right) \varepsilon^{-1} = \rho_{\text{free}}\varepsilon^{-1} - \left(\vec{\nabla} \circ \vec{P}\right) \varepsilon^{-1}.$$

Multiplying by ε, which we now assume to be invariant under the divergence, and collecting terms with the divergence operator, we obtain

$$\vec{\nabla} \circ (\varepsilon\vec{E} + \vec{P}) = \rho_{\text{free}}. \tag{2.5}$$

The argument of the divergence operator is often denoted $\vec{D}$, and is referred to as the *displacement of the electric field*:

$$\vec{D} = \varepsilon\vec{E} + \vec{P}. \tag{2.6}$$

In polarizable matter, $\vec{D}$ may differ significantly from $\vec{E}$ depending on the material's polarization $\vec{P}$; however, in free space, $\vec{P} = \vec{0}$, and so the electric field displacement has the same direction as $\vec{E}$ with magnitude scaled by the constant ε_0 (in this case, bound charges are also zero, and the equivalence of Equations 2.7 and 2.2 follows immediately). Typically, $\vec{D}$ is directly computed only in certain simple physical scenarios where the free charge is known and where symmetry may be exploited, and in these cases, is subsequently used in finding $\vec{E}$.

We may therefore write the differential form of Gauss's law for electric fields in matter as

$$\vec{\nabla} \circ \vec{D} = \rho_{\text{free}}. \tag{2.7}$$

Gauss's Law for Magnetic Fields

Gauss's law for electric fields has its analogue in magnetism, with the key difference arising from the fact that opposing (positive and negative) electric charges may occur separately from one another, whereas in nature there exist no free magnetic charges; *i.e.*, opposing (north and south) magnetic charges always occur in pairs [48], or as the basic entity that is referred to in magnetic studies as the magnetic dipole. This crucial difference in basic behavior between electricity and magnetism precluded the scientific community's connecting the two phenomena until after 1820, when the French physicists Jean-Baptiste Biot and Félix Savart began working, followed shortly thereafter by Ampère, to establish experimental laws relating magnetic induction to currents [8].

Gauss's law for magnetic fields is independent of the electric phenomena, and stems from this basic observation that magnetic monopoles cannot exist; in particular, if we consider a surface S enclosing a volume V in a vacuum as in the case of Gauss's law for electric fields, we may deduce from this basic observation that the net flux of the magnetic induction $\vec{B}$ through this surface must be zero:

$$\oint_S \vec{B} \circ \hat{n}\, \mathrm{d}a = 0.$$

Once again, applying the divergence theorem,

$$\vec{\nabla} \circ \vec{B} = 0. \tag{2.8}$$

These are the integral and differential forms, respectively, of Gauss's law for magnetic fields.

The Ampère-Maxwell Law

Inspired by an 1819 observation by Hans Christian Ørsted that wires carrying electric current were capable of deflecting magnetic compass needles in their vicinity [49], Ampère conducted elaborate and thorough experiments for the following five years, eventually synthesizing his results into a quantitative characterization of the relationship between electric current and magnetic fields [37]. Maxwell would further develop this characterization via his observation that it is not only the enclosed electric current I_{enc} that affects the circulating magnetic field, but also a changing electric flux, which crucially accounts for the time-dependence of such fields; these two phenomena correspond to the terms on the right-hand side of the following integral form of the Ampère-Maxwell equation [48]:

$$\oint_C \vec{B} \circ \mathrm{d}\vec{\ell} = \mu_0 \left(I_{\text{enc}} + \varepsilon_0 \frac{\mathrm{d}}{\mathrm{d}t} \int_S \vec{E} \circ \hat{n}\, \mathrm{d}a \right),$$

where $\vec{B}$, C, $\mathrm{d}\vec{\ell}$, $\vec{E}$, S, $\mathrm{d}a$, $\hat{n}$, and ε are as before, and where μ is referred to as the *permeability*. The permeability is an analogue concept to the permittivity, and as such shares with it some fundamental defining characteristics: it is typically a function $\mu : \mathbb{R}^3 \to \mathbb{R}^{3\times 3}$ of the spatial variable, and in the case of isotropic media, is also simply a real-valued function $\mu : \mathbb{R}^3 \to \mathbb{R}$. In case we consider the scenario in a vacuum, we also have an analogue constant vacuum permeability μ_0, whose value is given in the List of Constants. In matter, the permeability of a material depends on the material's temperature and density, and varies with the frequency of radiation as well.

Applying Stokes' theorem[3] to the magnetic field circulation on the left-hand side yields

$$\int_S \left(\vec{\nabla} \times \vec{B}\right) \circ \hat{n}\, \mathrm{d}a = \mu \left(I_{\text{enc}} + \varepsilon \frac{\mathrm{d}}{\mathrm{d}t} \int_S \vec{E} \circ \hat{n}\ \mathrm{d}a \right),$$

and the enclosed current may be written as the integral of the normal component of the total current density $\vec{J}$, *i.e.*,

$$I_{\text{inc}} = \int_S \vec{J} \circ \hat{n}\, \mathrm{d}a,$$

so that the Ampère-Maxwell law becomes

$$\int_S \left(\vec{\nabla} \times \vec{B}\right) \circ \hat{n}\, \mathrm{d}a = \mu \left(\int_S \vec{J} \circ \hat{n}\, \mathrm{d}a + \varepsilon \int_S \frac{\partial \vec{E}}{\partial t} \circ \hat{n}\, \mathrm{d}a \right),$$

under the assumption that both $\vec{E}$ and $\frac{\partial \vec{E}}{\partial t}$ are continuous with respect to time. For equality to hold over all surfaces, it must be true in general that

$$\vec{\nabla} \times \vec{B} = \mu \left(\vec{J} + \varepsilon \frac{\partial \vec{E}}{\partial t} \right). \tag{2.9}$$

[3]Stokes' theorem states that for an arbitrary vector field $\vec{A}(\vec{x}, t)$ and for any surface S bounded by a closed curve C, $\int_S \left(\vec{\nabla} \times \vec{A}\right) \circ \hat{n}\, \mathrm{d}a = \oint_C \vec{A} \circ \mathrm{d}\ell$, where $\hat{n}$ is an outward-pointing unit vector normal to S originating at the center of the infinitesimal area element $\mathrm{d}a$, and where $\mathrm{d}\vec{\ell}$ is an infinitesimal line element on C.

In an analogue scenario to the one considered in our discussion of Gauss's law for electric fields, in magnetic materials, bound currents may act as the source of additional magnetic fields. Therefore, following [48], we write the *bound current density*, which is caused by the motion of bound charges discussed in Section 2.1, as the curl of the magnetization:

$$\vec{J}_{\text{bound}} = \vec{\nabla} \times \vec{M}, \tag{2.10}$$

just as the bound charge density is the divergence of the polarization in dielectric materials. That is, the curl of the magnetization gives the equivalent volume electric current density resulting from alignment of microscopic magnetic dipoles in a magnetic medium [45]. However, in scenarios where the polarization changes with time, since this quantity, called the *polarization current density*, generates accumulation of (moving) charge, it will also contribute to the electric current density, which is expressed by

$$\vec{J}_{\text{pol}} = \frac{\partial \vec{P}}{\partial t}. \tag{2.11}$$

In conductive materials or in free space, a *free current density*, which is comprised by the motion of the free charges discussed in Section 2.1, may also exist in the presence of an electric field, and we refer to this portion of the current $\vec{J}$ as the conduction current $\vec{J}_{\text{free}}$.

Since $\vec{J}$ in Equation 2.9 refers to the total density of the bound, free, and polarization currents, we may therefore write

$$\vec{J} = \vec{J}_{\text{free}} + \vec{J}_{\text{bound}} + \vec{J}_{\text{pol}}, \tag{2.12}$$

and using Equations 2.10, 2.11, and 2.12 together, Equation 2.9 may be rewritten as

$$\vec{\nabla} \times \vec{B} = \mu \left(\vec{J}_{\text{free}} + \vec{J}_{\text{bound}} + \vec{J}_{\text{pol}} + \varepsilon \frac{\partial \vec{E}}{\partial t} \right) = \mu \left(\vec{J}_{\text{free}} + \vec{\nabla} \times \vec{M} + \frac{\partial \vec{P}}{\partial t} + \varepsilon \frac{\partial \vec{E}}{\partial t} \right).$$

Multiplying by the inverse of μ, collecting the arguments of the divergence operator on the right-hand side, and collecting those of the time derivative on the left-hand side, we obtain

$$\vec{\nabla} \times \left(\mu^{-1} \vec{B} - \vec{M} \right) = \vec{J}_{\text{free}} + \frac{\partial (\varepsilon \vec{E} + \vec{P})}{\partial t},$$

into which we substitute Equation 2.6 for the displacement to obtain

$$\vec{\nabla} \times \left(\mu^{-1} \vec{B} - \vec{M} \right) = \vec{J}_{\text{free}} + \frac{\partial \vec{D}}{\partial t}. \tag{2.13}$$

As in the case of the displacement, we may rewrite the argument of the curl operator as its own physical quantity $\vec{H}$, referred to as the magnetic field intensity, expressed as

$$\vec{H} = \mu^{-1} \vec{B} - \vec{M}. \tag{2.14}$$

As is the case with displacement in dielectric matter, $\vec{H}$ may differ from $\vec{B}$ significantly in magnetic matter; however, in free space, $\vec{H}$ has the same direction as $\vec{B}$, with magnitude scaled by the constant μ_0. The Ampère-Maxwell law may therefore be written in differential form in terms of $\vec{H}$ as

$$\vec{\nabla} \times \vec{H} = \vec{J}_{\text{free}} + \frac{\partial \vec{D}}{\partial t}. \tag{2.15}$$

Faraday's Law of Induction

Following a series of experiments on the behavior of currents in circuits exposed to time-varying magnetic fields, Faraday first demonstrated the relationship between time-dependent electric and magnetic fields [38–40]. The principal observation that Faraday contributed to this area was that changing magnetic flux through a surface induces an electromotive force around that surface, which in turn causes a current flow according to Ohm's law. Faraday's law is thus expressed in integral form as [48]

$$\oint_C \vec{E} \circ \mathrm{d}\vec{\ell} = -\frac{\mathrm{d}}{\mathrm{d}t} \int_S \vec{B} \circ \hat{n}\, \mathrm{d}a, \tag{2.16}$$

where S, $\vec{E}$, $\vec{B}$, $\hat{n}$, and $\mathrm{d}a$ are as defined in the preceding subsections on Gauss's laws, and where C is a closed curve bounding S and $\mathrm{d}\vec{\ell}$ an infinitesimal line element of C. This form is equivalent [50] to the form

$$\oint_C \left(\vec{E} + \vec{v} \times \vec{B}\right) \circ \mathrm{d}\vec{\ell} = -\frac{\mathrm{d}}{\mathrm{d}t} \int_S \vec{B} \circ \hat{n}\, \mathrm{d}a,$$

where on each element of C, in the presence of $\vec{E}$ and $\vec{B}$, the force $\vec{F}$ acting on a point charge q that moves with velocity $\vec{v}(t, \vec{x})$ is given by the *Lorentz force law* as

$$\vec{F} = q\left(\vec{E} + \vec{v} \times \vec{B}\right). \tag{2.17}$$

The quantity $\left(\vec{E} + \vec{v} \times \vec{B}\right)$ is the *Lorentz force* acting on a unit-charged carrier in the circuit and is sometimes referred to as the *effective electric field* [51].

Applying Stokes' theorem to the electric field circulation on the left-hand side of Equation 2.16 yields

$$\int_S \left(\vec{\nabla} \times \vec{E}\right) \circ \hat{n}\, \mathrm{d}a = -\frac{\mathrm{d}}{\mathrm{d}t} \int_S \vec{B} \circ \hat{n}\, \mathrm{d}a,$$

and for geometries that are stationary, the time derivative may be moved inside of the integral in the electromotive force term on the right-hand side, yielding

$$\int_S \left(\vec{\nabla} \times \vec{E}\right) \circ \hat{n}\, \mathrm{d}a = -\int_S \left(\frac{\partial \vec{B}}{\partial t} \circ \hat{n}\right) \mathrm{d}a.$$

For equality to hold over all surfaces, it must be true in general that

$$\vec{\nabla} \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}. \tag{2.18}$$

This is the differential form of Faraday's law.

Maxwell's Equations

Together, Equations 2.7, 2.8, 2.18, and 2.15 comprise what are known as Maxwell's equations, and they are reproduced here in differential form.

$$\begin{aligned}\vec{\nabla} \circ \vec{D} &= \rho_{\text{free}} \\ \vec{\nabla} \circ \vec{B} &= 0 \\ \vec{\nabla} \times \vec{E} &= -\frac{\partial \vec{B}}{\partial t} \\ \vec{\nabla} \times \vec{H} &= \vec{J}_{\text{free}} + \frac{\partial \vec{D}}{\partial t}.\end{aligned} \tag{2.19}$$

These four equations constitute the synthesis of electromagnetic theory, and as such are applicable to electromagnetic phenomena at all frequencies. Figure 2.1 shows the names and most common applications of electromagnetic waves at various frequencies along the electromagnetic spectrum. The microwave frequency range is from 0.3 to 300 GHz, which includes waves used for mobile telephone communication, radar, and television satellite communications [52]. The frequencies most commonly used for microwave sintering are 915 MHz and 2.45 GHz [53], but other frequencies in the microwave range have reportedly been used for sintering with varying results, including 5.8 GHz [54], 22.00 GHz [55], 24.12 GHz [56], 28 GHz [57, 58], 60 GHz [59], and 300 GHz [53].

In the statement of Maxwell's equations, it is sometimes useful to consider induced and externally impressed motion of microscopic charges as separate phenomena. In this section, we follow the approach of [45] in formulating a restatement of Equations 2.19 that will later be useful in describing energy phenomena.

As we observed in Section 2.1, in dielectric media, the presence of electric or magnetic fields can, by instantaneously altering the positions of electrons within their atoms, create bound charge within a material. Through the action of the Lorentz force, the electric and magnetic fields are said to *induce* a motion of these microscopic charges, which is called *induced current* and is denoted by $\vec{J}_{\text{ind}}$; on the other hand, sometimes the configuration of the medium itself or the action of non-electromagnetic forces can cause distributions of current in a material that are called *externally impressed*, or *impressed* [45]. The free current $\vec{J}_{\text{free}}$, the polarization $\vec{P}$, and the magnetization density $\vec{M}$ are the sources of the electric and magnetic fields, and in general, both induced and impressed current may result from each source; we may therefore write each of these as the sum of two terms, as follows:

$$\vec{J}_{\text{free}} = \vec{J}_{\text{free,ind}} + \vec{J}_{\text{free,ext}} \tag{2.20}$$

$$\vec{P} = \vec{P}_{\text{ind}} + \vec{P}_{\text{ext}} \tag{2.21}$$

$$\vec{M} = \vec{M}_{\text{ind}} + \vec{M}_{\text{ext}} \tag{2.22}$$

where the subscript $_{\text{ind}}$ denotes the induced sources while $_{\text{ext}}$ denotes the externally impressed sources [45]. We also split the free charge density ρ_{free}:

$$\rho_{\text{free}} = \rho_{\text{free,ind}} + \rho_{\text{free,ext}}, \tag{2.23}$$

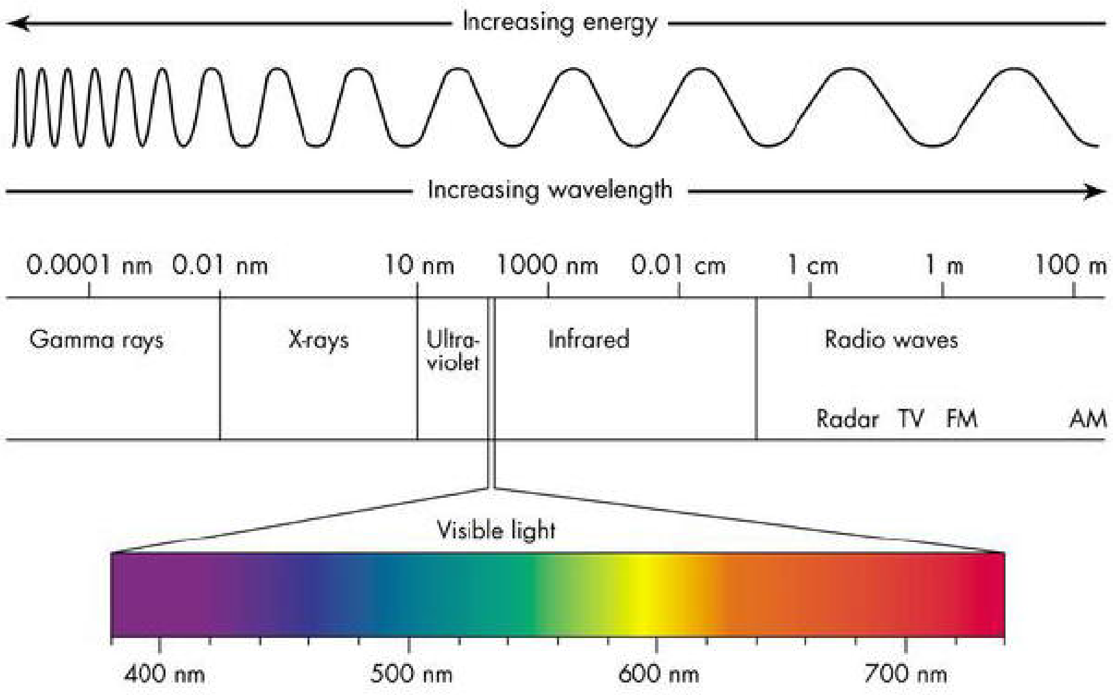


Figure 2.1: The electromagnetic spectrum.

and we assume that $\rho_{\text{free,ind}}$ and $\vec{J}_{\text{free,ind}}$ themselves satisfy the law of conservation of charge, which can be derived from Maxwell's Equations and is, for these quantities,

$$\nabla \circ \vec{J}_{\text{free,ind}} + \frac{\partial \rho_{\text{free,ind}}}{\partial t} = 0. \tag{2.24}$$

We also assume that $\vec{D}_{\text{ind}}$ satisfies the modified constitutive relation

$$\vec{D}_{\text{ind}} = \varepsilon \vec{E} + \vec{P}_{\text{ind}} + \int \vec{J}_{\text{free,ind}} \, \mathrm{d}t, \tag{2.25}$$

and that $\vec{B}_{\text{ext}} = \mu \vec{M}_{\text{ext}}$, so that $\vec{B}_{\text{ind}}$ satisfies

$$\vec{B}_{\text{ind}} = \vec{B} - \mu \vec{M}_{\text{ext}}. \tag{2.26}$$

Taking the divergence of Equation 2.25, we obtain

$$\nabla \circ \vec{D}_{\text{ind}} = \nabla \circ \left(\varepsilon \vec{E} + \vec{P}_{\text{ind}} + \int \vec{J}_{\text{free,ind}} \, \mathrm{d}t \right),$$

and using the linearity of the divergence together with Equations 2.2 and 2.21, we obtain

$$\nabla \circ \vec{D}_{\text{ind}} = \rho + \nabla \circ \left(\vec{P} - \vec{P}_{\text{ext}}\right) + \nabla \circ \left(\int \vec{J}_{\text{free,ind}}\, \mathrm{d}t\right).$$

To the first term on the right-hand side, we apply Equations 2.3 and 2.4 to obtain

$$\nabla \circ \vec{D}_{\text{ind}} = \rho_{\text{free}} - \nabla \circ \vec{P} + \nabla \circ \left(\vec{P} - \vec{P}_{\text{ext}}\right) + \nabla \circ \left(\int \vec{J}_{\text{free,ind}}\, \mathrm{d}t\right),$$

and once more exploiting the linearity of the divergence operator, we obtain

$$\nabla \circ \vec{D}_{\text{ind}} = \rho_{\text{free}} - \nabla \circ \vec{P}_{\text{ext}} + \nabla \circ \left(\int \vec{J}_{\text{free,ind}}\, \mathrm{d}t\right).$$

If we assume that all of the quantities $\vec{J}_{\text{free,ind}}$, $\frac{\partial \vec{J}_{\text{free,ind}}}{\partial x}$, $\frac{\partial \vec{J}_{\text{free,ind}}}{\partial y}$, and $\frac{\partial \vec{J}_{\text{free,ind}}}{\partial z}$ are continuous over the entire spatial and time domains, then the Leibniz rule applies [60], and we may rewrite the final term on the right-hand side to obtain

$$\nabla \circ \vec{D}_{\text{ind}} = \rho_{\text{free}} - \nabla \circ \vec{P}_{\text{ext}} + \int \nabla \circ \vec{J}_{\text{free,ind}}\, \mathrm{d}t,$$

and using Equation 2.24 together with the fundamental theorem of calculus gives rise to

$$\nabla \circ \vec{D}_{\text{ind}} = \rho_{\text{free}} - \nabla \circ \vec{P}_{\text{ext}} - \rho_{\text{free,ind}}.$$

Finally, after applying Equation 2.23, we obtain

$$\nabla \circ \vec{D}_{\text{ind}} = \rho_{\text{free,ext}} - \nabla \circ \vec{P}_{\text{ext}}, \tag{2.27}$$

which is considered an analogue expression of Gauss's law for electric fields. Now, when we take the divergence of Equation 2.26, we obtain

$$\nabla \circ \vec{B}_{\text{ind}} = \nabla \circ \left(\vec{B} - \mu \vec{M}_{\text{ext}}\right),$$

and applying the linearity property of the divergence, together with Equation 2.8, we obtain

$$\nabla \circ \vec{B}_{\text{ind}} = -\nabla \circ \left(\mu \vec{M}_{\text{ext}}\right), \tag{2.28}$$

which is considered an analogue expression of Gauss's law for magnetic fields. Applying Equations 2.20, 2.21, and 2.6 to Equation 2.15, we obtain

$$\nabla \times \vec{H} = \vec{J}_{\text{free,ind}} + \vec{J}_{\text{free,ext}} + \frac{\partial}{\partial t}\left(\varepsilon \vec{E} + \vec{P}_{\text{ind}} + \vec{P}_{\text{ext}}\right),$$

to which applying Equation 2.25 results in

$$\nabla \times \vec{H} = \vec{J}_{\text{free,ind}} + \vec{J}_{\text{free,ext}} + \frac{\partial}{\partial t}\left(\varepsilon\vec{E} + \vec{D}_{\text{ind}} - \varepsilon\vec{E} - \int \vec{J}_{\text{free,ind}}\,\mathrm{d}t + \vec{P}_{\text{ext}}\right).$$

We now apply the linearity of the partial derivative, together with the fundamental theorem of calculus, to obtain

$$\nabla \times \vec{H} = \vec{J}_{\text{free,ind}} + \vec{J}_{\text{free,ext}} + \frac{\partial\vec{D}_{\text{ind}}}{\partial t} - \vec{J}_{\text{free,ind}} + \frac{\partial\vec{P}_{\text{ext}}}{\partial t},$$

which is equivalent to

$$\nabla \times \vec{H} = \vec{J}_{\text{free,ext}} + \frac{\partial\vec{D}_{\text{ind}}}{\partial t} + \frac{\partial\vec{P}_{\text{ext}}}{\partial t}, \tag{2.29}$$

the analogue expression of the Ampère-Maxwell law. Finally, under the assumption given by Equation 2.26, Faraday's law, stated in Equation 2.18, becomes

$$\nabla \times \vec{E} = -\frac{\partial\vec{B}_{\text{ind}}}{\partial t} - \mu\frac{\partial\vec{M}_{\text{ext}}}{\partial t}. \tag{2.30}$$

We may synthesize these expression of Maxwell's equations by defining some quantities using an equivalence principle to be described at the end of this section. We define the *equivalent impressed electric current density* as

$$\vec{J}_{\text{eq,ext}} := \vec{J}_{\text{free,ext}} + \frac{\partial\vec{P}_{\text{ext}}}{\partial t}, \tag{2.31}$$

the *equivalent impressed magnetic current density* as

$$\vec{\mathcal{M}}_{\text{eq,ext}} := \mu\frac{\partial\vec{M}_{\text{ext}}}{\partial t}, \tag{2.32}$$

the corresponding *equivalent impressed magnetic charge density* as

$$\rho_{\text{m,eq,ext}} := -\nabla \circ \left(\mu\vec{M}_{\text{ext}}\right), \tag{2.33}$$

and the *impressed electric charge density* (not under the equivalence principle) as

$$\rho_{\text{exp}} := \rho_{\text{free,ext}} - \nabla \circ \vec{P}_{\text{ext}}. \tag{2.34}$$

Substituting Equation 2.34 into Equation 2.27, Equation 2.33 into Equation 2.28, Equation 2.31 into Equation 2.29, and Equation 2.32 into Equation 2.30, we obtain, finally,

$$\begin{aligned}
\nabla \circ \vec{D}_{\text{ind}} &= \rho_{\text{ext}} \\
\nabla \circ \vec{B}_{\text{ind}} &= \rho_{\text{m,eq,ext}} \\
\nabla \times \vec{H} &= \vec{J}_{\text{eq,ext}} + \frac{\partial\vec{D}_{\text{ind}}}{\partial t} \\
\nabla \times \vec{E} &= -\frac{\partial\vec{B}_{\text{ind}}}{\partial t} - \vec{\mathcal{M}}_{\text{eq,ext}},
\end{aligned}$$

from which subscripts are customarily dropped to yield

$$\begin{aligned}\nabla \circ \vec{D} &= \rho \\ \nabla \circ \vec{B} &= \rho_{\mathrm{m}} \\ \nabla \times \vec{H} &= \vec{J} + \frac{\partial \vec{D}}{\partial t} \\ \nabla \times \vec{E} &= -\frac{\partial \vec{B}}{\partial t} - \vec{\mathcal{M}}.\end{aligned} \tag{2.35}$$

The equivalence principle used in naming the subscripted quantities arises from the fact that the solution of Equations 2.35 for $\vec{B}$ and $\vec{E}$ using the impressed sources yields, in this modified case, not the true $\vec{B}$ and $\vec{E}$, but rather $\vec{B}_{\mathrm{ind}}$ and $\vec{E}$, from which the true $\vec{B}$ may be recovered using Equation 2.26 only when the magnetization density $\vec{M}_{\mathrm{ext}}$ is known [45].

2.2 Energy, Power, and Poynting's Theorem

The common explanation of the physical phenomenon of microwave heating depends on the fact that the energy required for dipolar molecules to remain in the presence of an electric field is minimized when the molecules are physically oriented so that their poles align with that of the field. In a microwave cavity, the electric field changes rapidly in time according to a standing wave pattern, which causes dipolar molecules within materials residing in the cavity to reverse their orientation so quickly that the friction from this action generates heat within materials. The rate at which this occurs depends on the frequency of radiation, and on the temperature and molecular composition of the material in the cavity, and will be discussed in the rest of this section in the frame of [61].

In general, a source of electromagnetic energy sets up fields that store electric and magnetic energy and carry power that may be transmitted or dissipated as loss [62]. The time-average energy stored in the sinusoidal, steady-state electric field that exists in a volume V is given by

$$W_e = \frac{1}{4}\,\mathrm{Re}\left\{\int_V \vec{D} \circ \vec{E}^* \; \mathrm{d}V\right\}, \tag{2.36}$$

and correspondingly, the time-average energy stored in the magnetic field is

$$W_m = \frac{1}{4}\,\mathrm{Re}\left\{\int_V \vec{B} \circ \vec{H}^* \; \mathrm{d}V\right\}, \tag{2.37}$$

where the asterisk (*) denotes the complex conjugate, and the factor $\frac{1}{2}$ on each of the fields, from which arises the coefficient $\frac{1}{4}$ in each of the above expressions, is due to the averaging over a single time interval. This formulation is valid for media without dissipation (which allows a relation between energy and work done on the system), and for media in which ε and μ do not depend on ω.

Poynting's theorem leads to energy conservation for electromagnetic fields and sources, and we derive the theorem here according to [45] and [62]. In the presence of the Lorentz force $\vec{F}$ in Equation 2.17, conversion of energy between electromagnetic and non-electromagnetic forms results in a motion of charges; therefore, the power being delivered to a single test charge q is equal to the rate of work being done *against* the Lorentz force to move the charge [45], which results in

$$P = \vec{F} \circ \vec{v} = q\left(\vec{E} + \vec{v} \times \vec{B}\right) \circ \vec{v} = q\vec{E} \circ \vec{v} + q\left(\vec{v} \times \vec{B}\right) \circ \vec{v} = q\vec{E} \circ \vec{v},$$

since $\vec{v} \times \vec{B} \perp \vec{v}$ (and therefore $\left(\vec{v} \times \vec{B}\right) \circ \vec{v} = 0$). In case the volumetric charge density function ρ is continuous, there will be a power $P_v \mathrm{d}V$ delivered to the charge density in the differential volume element $\mathrm{d}V$, where

$$P_v := \vec{F}_v \circ \vec{v} = \rho\vec{E} \circ \vec{v} = \vec{E} \circ \vec{J},$$

using the definition of the volumetric current density in terms of the charge density: $\vec{J} := \rho\vec{v}$. Therefore, in order to maintain a current density $\vec{J}$ within the volume V, the power that must be delivered is

$$P_J = \int_V P_v \,\mathrm{d}V = \int_V \vec{E} \circ \vec{J} \,\mathrm{d}V, \tag{2.38}$$

which is referred to as the *Joule power* [45]. If P_J is positive, then energy is being delivered to the system, and is dissipated—that is, it is converted to a non-electromagnetic form of energy, such as heat. Alternatively, if P_J is negative, then energy is coming from the system—that is, it is being generated *from* some non-electromagnetic form, such as chemical energy in the case of a battery [45]. It is the former case that we are interested in.

We assume a time-harmonic version of Maxwell's equations, where

$$\vec{E}(\vec{x},t) = \mathrm{Re}\left\{\vec{\mathcal{E}}(\vec{x})e^{j\omega t}\right\} \quad \text{and} \quad \vec{H}(\vec{x},t) = \mathrm{Re}\left\{\vec{\mathcal{H}}(\vec{x})e^{j\omega t}\right\},$$

and $\vec{\mathcal{E}} := \langle E_1, E_2, E_3 \rangle$ and $\vec{\mathcal{H}} := \langle H_1, H_2, H_3 \rangle$ are functions of space only. Under this cosine-based phasor representation, where $\vec{\phi} := \langle \phi_1, \phi_2, \phi_3 \rangle$ is the phase reference of the wave, the electric field is

$$\vec{E} = \langle E_1 \cos(\omega t + \phi_1), E_2 \cos(\omega t + \phi_2), E_3 \cos(\omega t + \phi_3) \rangle,$$

and the average of the square of the magnitude of the electric field over the time interval $[0, T]$ may be calculated as

$$\begin{aligned}
|\vec{E}|^2_{\text{avg}} &= \frac{1}{T}\int_0^T \vec{E}\circ\vec{E}\,\mathrm{d}t \\
&= \frac{1}{T}\int_0^T \Big[E_1^2\cos^2(\omega t+\phi_1) + E_2^2\cos^2(\omega t+\phi_2) + \\
&\qquad + E_3^2\cos^2(\omega t+\phi_3)\Big]\,\mathrm{d}t \\
&= \frac{1}{T}\int_0^T \frac{1}{2}\Big[E_1^2 + E_1^2\cos(2\omega t+2\phi_1) + E_2^2 + E_2^2\cos(2\omega t+2\phi_2) + \\
&\qquad + E_3^2 + E_3^2\cos(2\omega t+2\phi_3)\Big]\,\mathrm{d}t \\
&= \frac{1}{2T}\Bigg[E_1^2 t + \frac{E_1^2}{2\omega}\sin(2\omega t+2\phi_1) + E_2^2 t + \frac{E_2^2}{2\omega}\sin(2\omega t+2\phi_2) + \\
&\qquad + E_3^2 t + \frac{E_3^2}{2\omega}\sin(2\omega t+2\phi_3)\Bigg]_0^T \qquad (2.39)\\
&= \frac{1}{2T}\Bigg[E_1^2 T + \frac{E_1^2}{2\omega}\sin(2\omega T+2\phi_1) - \frac{E_1^2}{2\omega}\sin(2\phi_1) + E_2^2 T + \\
&\qquad + \frac{E_2^2}{2\omega}\sin(2\omega T+2\phi_2) - \frac{E_2^2}{2\omega}\sin(2\phi_2) + E_3^2 T + \\
&\qquad + \frac{E_3^2}{2\omega}\sin(2\omega T+2\phi_3) - \frac{E_1^3}{2\omega}\sin(2\phi_3)\Bigg] \\
&= \frac{1}{2T}\Big[E_1^2 T + E_2^2 T + E_3^2 T\Big] \\
&= E_1^2 + E_2^2 + E_3^2 \\
&= \frac{1}{2}|\vec{\mathcal{E}}|^2.
\end{aligned}$$

The $|\vec{E}|^2_{\text{avg}}$ representation of this quantity will be useful when $\vec{E}$ is calculated directly using the electromagnetic wave equation, as in Section 2.4, whereas the $\frac{1}{2}|\vec{\mathcal{E}}|^2$ representation will be useful when the phasor form of the electric field is computed using the Helmholtz equation, as in Section 2.5. Whichever way it is computed and represented, this quantity is used as a factor in the source term of the heat equation, which will be discussed in Section 3.2.

In the phasor representation, the partial derivative in time is replaced by $j\omega$, and, assuming linear media where ε and μ do not depend on $\mathcal{E}$ or $\mathcal{H}$, Faraday's law and the Maxwell-Ampère law become

$$\nabla\times\vec{\mathcal{E}} = -j\omega\mu\vec{\mathcal{H}} - \vec{\mathcal{M}}_s \qquad (2.40)$$

and

$$\nabla\times\vec{\mathcal{H}} = -j\omega\varepsilon\vec{\mathcal{E}} + \vec{\mathcal{J}}. \qquad (2.41)$$

Taking the dot product of Equation 2.40 with $\vec{\mathcal{H}}^*$, the complex conjugate of $\vec{\mathcal{H}}$, and of the complex conjugate of Equation Equation 2.41 with $\vec{\mathcal{E}}$, we obtain

$$\begin{aligned} \vec{\mathcal{H}}^* \circ \left(\nabla \times \vec{\mathcal{E}}\right) &= -j\omega\mu|\vec{\mathcal{H}}|^2 - \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s \\ \vec{\mathcal{E}} \circ (\nabla \times \vec{*}) &= -j\omega\varepsilon^*|\vec{\mathcal{E}}|^2 + \vec{\mathcal{E}} \circ \vec{\mathcal{J}}^* = -j\omega\varepsilon^*|\vec{\mathcal{E}}|^2 + \vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* + \sigma|\vec{\mathcal{E}}|^2, \end{aligned} \tag{2.42}$$

using a modified phasor form of Ohm's law in Equation 2.51 and considering the electric source current $\vec{\mathcal{J}}_s$ separately from the conduction current $\sigma\vec{\mathcal{E}}$. Using the vector identity

$$\vec{A} \circ \left(\vec{A} \times \vec{B}\right) = \vec{B} \circ \left(\nabla \times \vec{A}\right) - \vec{A} \circ \left(\nabla \times \vec{B}\right),$$

we see that Equations 2.42 lead to

$$\begin{aligned} \nabla \circ \left(\vec{\mathcal{E}} \times \vec{\mathcal{H}}^*\right) &= \vec{\mathcal{H}}^* \circ \left(\nabla \times \vec{\mathcal{E}}\right) - \vec{\mathcal{E}} \circ \left(\nabla \times \vec{\mathcal{H}}^*\right) \\ &= -j\omega\mu|\vec{\mathcal{H}}|^2 - \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s + j\omega\varepsilon|\vec{\mathcal{E}}|^2 - \vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* - \sigma|\vec{\mathcal{E}}|^2 \\ &= -\sigma|\vec{\mathcal{E}}|^2 + j\omega\left(\varepsilon^*|\vec{\mathcal{E}}|^2 - \mu|\vec{\mathcal{H}}|^2\right) - \left(\vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* + \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s\right). \end{aligned}$$

We integrate over the volume V, so that

$$\begin{aligned} \int_V \nabla \circ \left(\vec{\mathcal{E}} \times \vec{\mathcal{H}}^*\right)\, \mathrm{d}V = -\int_V \sigma|\vec{\mathcal{E}}|^2\, \mathrm{d}V + \int_V j\omega\left(\varepsilon^*|\vec{\mathcal{E}}|^2 - \mu|\vec{\mathcal{H}}|^2\right)\, \mathrm{d}V - \\ - \int_V \left(\vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* + \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s\right)\, \mathrm{d}V, \end{aligned}$$

and using the Divergence theorem to rewrite the left-hand side, we obtain

$$\begin{aligned} \oint_S \left(\vec{\mathcal{E}} \times \vec{\mathcal{H}}^*\right) \circ \hat{n}\, \mathrm{d}A = -\int_V \sigma|\vec{\mathcal{E}}|^2\, \mathrm{d}V + \int_V j\omega\left(\varepsilon^*|\vec{\mathcal{E}}|^2 - \mu|\vec{\mathcal{H}}|^2\right)\, \mathrm{d}V - \\ - \int_V \left(\vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* + \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s\right)\, \mathrm{d}V, \end{aligned}$$

where S is the closed surface enclosing the volume V, and $\hat{n}$ is an outward-pointing unit vector normal to S originating at the center of the infinitesimal area element $\mathrm{d}A$. Now, accounting for dielectric losses by writing the complex permittivity and permeability as

$$\varepsilon := \varepsilon' - j\varepsilon'' \qquad \text{and} \qquad \mu := \mu' - j\mu'',$$

we obtain

$$\begin{aligned} -\frac{1}{2}\int_V \left(\vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* + \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s\right)\, \mathrm{d}V = \frac{1}{2}\oint_S \left(\vec{\mathcal{E}} \times \vec{\mathcal{H}}^*\right) \circ \hat{n}\, \mathrm{d}A + \frac{\sigma}{2}\int_V |\vec{\mathcal{E}}|^2\, \mathrm{d}V + \\ + \frac{\omega}{2}\int_V \left(\varepsilon''|\vec{\mathcal{E}}|^2 + \mu''|\vec{\mathcal{H}}|^2\right)\, \mathrm{d}V + j\frac{\omega}{2}\int_V \left(\mu'|\vec{\mathcal{H}}|^2 - \varepsilon'|\vec{\mathcal{E}}|^2\right)\, \mathrm{d}V. \end{aligned} \tag{2.43}$$

This result is known as *Poynting's theorem*, and is a power-balance equation where the integral on the left-hand side represents the complex power, P_s, delivered by the sources $\vec{\mathcal{J}}_s$ and $\vec{\mathcal{M}}_s$ inside S:

$$P_s := -\frac{1}{2}\int_V \left(\vec{\mathcal{E}} \circ \vec{\mathcal{J}}_s^* + \vec{\mathcal{H}}^* \circ \vec{\mathcal{M}}_s\right)\, \mathrm{d}V.$$

The first integral on the right-hand side of Equation 2.43 represents complex power flow out of the closed surface S. The time-average complex power flow P_o out of the closed surface S is given by the integral of the complex *Poynting vector* $\vec{\mathcal{S}}$ over that surface, *i.e.*, the first term on the right-hand side of Equation 2.43:

$$P_o = \frac{1}{2}\oint_S \left(\vec{\mathcal{E}} \times \vec{\mathcal{H}}^*\right) \circ \hat{n}\, \mathrm{d}A := \frac{1}{2}\oint_S \vec{\mathcal{S}} \circ \hat{n}\, \mathrm{d}A.$$

The real parts of P_s and P_o represent time-average powers. The second and third integrals on the right-hand side of Equation 2.43 are real quantities representing the time-average power dissipated in the volume V due to conductivity, dielectric, and magnetic losses. If we define this *dissipated power* as P_{diss}, we have

$$P_{\mathrm{diss}} := \frac{\sigma}{2}\int_V |\vec{\mathcal{E}}|^2\, \mathrm{d}V + \frac{\omega}{2}\int_V \left(\varepsilon''|\vec{\mathcal{E}}|^2 + \mu''|\vec{\mathcal{H}}|^2\right)\, \mathrm{d}V, \tag{2.44}$$

which is sometimes referred to as *Joule's law*. The last integral in Equation 2.43 is related to the stored electric and magnetic energies, as defined in Equations 2.36 and 2.37, and so, using these together with the preceding definitions of P_{diss}, P_o, and P_s, Poynting's theorem may be rewritten as

$$P_s = P_o + P_{\mathrm{diss}} + 2j\omega(W_m - W_e). \tag{2.45}$$

This power balance equation states that the power delivered by the sources is equal to the sum of the power transmitted through the surface, the power lost to heat in the volume, and 2ω times the net reactive energy stored in the volume.

2.3 Propagation of Guided Microwaves

As the previous sections have discussed problems related to the propagation of waves in free space, we discuss briefly some aspects of propagation of guided microwaves. Typically, in systems for microwave heating, waves are guided from the generator (which may be of various types, including magnetrons, which are traditional in domestic microwave ovens, or solid state generators, which are found in systems requiring a high degree of precision) into the cavity, where they come into contact with the specimen to be heated. Before describing the problem of computing electric and magnetic fields within the cavity, it will be necessary to discuss some aspects of propagation of guided waves in particular.

Waveguides are, in their most basic description, tubes that consolidate and guide waves; waveguides of various types are used in practice, and the most common are metal waveguides with circular, elliptical, or rectangular cross-sections. The distribution of the field inside of the waveguide

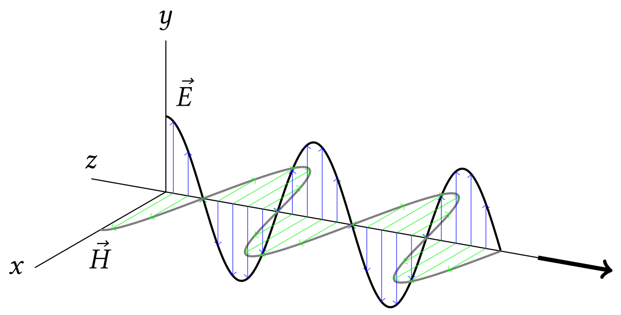


Figure 2.2: Propagation of an electromagnetic plane wave in free space. Here, $\vec{E}$ represents the electric field, and $\vec{H}$ represents the magnetic field.

is highly dependent on the geometry, which influences the evolution of multiple wave reflections from perfectly conducting walls. In this work, we are interested in metal waveguides with rectangular cross-section.

Typical waveguides have fixed dimensions, dependent on the frequency of radiation with which they are intended to work, and on the mode of propagation and polarization of the fields. The two fundamental polarizations of the electromagnetic field associated with a given direction of propagation are referred to as the *transverse electric*, or *TE-polarization*, and the *transverse magnetic*, or *TM-polarization*[4]. With the TE-polarization, the electric field is perpendicular to the direction of propagation, and the incident field in this direction is assumed to be zero; that is, if $\vec{E} := \langle E_x, E_y, E_z \rangle$, then $E_z \equiv 0$ in the TE-polarization; in the three-dimensional TE-polarization, the magnetic field $\vec{H}$ exists in the plane perpendicular to $\vec{E}$, as seen in Figure 2.2, and so $H_z \not\equiv 0$. With the TM-polarization, the magnetic field is perpendicular to the direction of propagation, and so if $\vec{H} := \langle H_x, H_y, H_z \rangle$, then $H_z \equiv 0$, and correspondingly, $E_z \neq 0$.

In this work, we study the TE mode of propagation. For a TE-polarized wave in a guide, the function that describes the displacement of a wave in the direction of propagation is H_z, which satisfies the Helmholtz equation

$$\Delta H_z + k_c^2 H_z = 0,$$

where $k_c^2 = \omega^2 \varepsilon \mu$ is referred to as the cutoff constant of wave propagation, subject to the boundary condition

$$\frac{\partial H_z}{\partial \vec{n}} = 0,$$

where $\vec{n}$ is an outward-pointing vector normal to the surface of the metal waveguide.

The problem described above has a solution only when k_c takes on the values

$$k_c = \sqrt{\left(\frac{m\pi}{a}\right)^2 + \left(\frac{n\pi}{b}\right)^2}, \quad m, n \in \mathbb{N}, \tag{2.46}$$

[4]The TM-polarization is sometimes also referred to as the *E-polarization*, and the TE-polarization is sometimes also referred to as the *H-polarization*.

where the waveguide has rectangular cross-section of width a and height b. These values of k_c are referred to as eigenvalues of the boundary value problem, and their corresponding eigenfunctions—that is, for each pair (m,n), the function H_z which satisfies the boundary value problem and the corresponding H_x, H_y, E_x, E_y, and E_z that satisfy Maxwell's equations in the cavity—are given by

$$\begin{aligned}
H_z &= H_0 \cos\left(\frac{m\pi x}{a}\right)\cos\left(\frac{n\pi y}{b}\right),\\
E_x(y,z) &= H_0 \frac{j\omega\mu}{k_c^2}\frac{n\pi}{b}\cos\left(\frac{m\pi x}{a}\right)\sin\left(\frac{n\pi y}{b}\right)\\
E_y(y,z) &= -H_0 \frac{j\omega\mu}{k_c^2}\frac{m\pi}{a}\sin\left(\frac{m\pi x}{a}\right)\cos\left(\frac{n\pi y}{b}\right)\\
H_x(y,z) &= H_0 \frac{\gamma}{k_c^2}\frac{m\pi}{a}\sin\left(\frac{m\pi x}{a}\right)\cos\left(\frac{n\pi y}{b}\right)\\
H_y(y,z) &= H_0 \frac{\gamma}{k_c^2}\frac{n\pi}{b}\cos\left(\frac{m\pi x}{a}\right)\sin\left(\frac{n\pi y}{b}\right).
\end{aligned}$$

Here, γ refers to the propagation constant of the wave. Each pair (m,n) of natural numbers is referred to as a "fundamental mode" of propagation, which is written as TE_{mn}, and corresponds to a single eigenvalue and solution of the electric and magnetic fields. In our computations, we use the TE_{10} mode, also called the *fundamental mode* because its corresponding wavelength is equal to the maximum. When the operating frequency is such that the TE_{10} mode is the dominant one, the maximum value of the electric field occurs in the center of the waveguide, and if samples are heated directly in the guide, then they may be placed here in order to most efficiently utilize the field's energy.

The length of the guided wave may be determined according to the relation

$$\left(\frac{1}{\lambda}\right)^2 = \left(\frac{1}{\lambda_c}\right)^2 + \left(\frac{1}{\lambda_g}\right)^2, \tag{2.47}$$

where λ is the length of the wave in free space, and λ_c is the quantity referred to as the *cutoff wavelength*, computed using

$$\lambda_c = \frac{c}{f_c} = \frac{2\pi c}{\omega_c} = \frac{2\pi}{k_c}. \tag{2.48}$$

The cutoff frequencies and wavelength are, therefore, characteristics of the mode of propagation, and when they are determined, the propagation conditions may be described according to Table 2.1. These conditions of propagation show that the waveguide acts as a high-pass frequency filter, in which only those frequencies greater than the cutoff frequency may propagate. Knowledge of λ and λ_c, or λ and (m,n), along with the dimensions of the cross-section of the waveguide, permits calculation of λ_g, and in case the sample is to be heated inside the waveguide itself, rather than in a separate cavity, the waveguide's optimal length may be calculated as an integer-plus-one-half multiple of λ_g, in order that the maximum value of the electric field be located in the center of the waveguide, where the sample would be placed.

- If $\lambda < \lambda_c$, then the mode is called *evanescent*, and the wave does not propagate;
- If $\lambda = \lambda_c$, then the wave propagates with speed equal to $\frac{1}{\sqrt{\mu\varepsilon'}}$;
- If $\lambda > \lambda_c$, then the wave may propagate with no attenuation, with a speed $v_g := f\lambda_g = \frac{\omega}{2\pi}\lambda_g$.

Table 2.1: Propagation conditions when the operating wavelength λ takes various values in relation to the cutoff frequency λ_c.

2.4 The Electromagnetic Wave Equations

Maxwell's equations may be combined to form a single wave equation through the use of the constitutive relations. This process gives us a way of evaluating fields within macroscopic media in space using the basic information that Maxwell's equations provide about electromagnetic fields in a vacuum, and deriving the wave equations also provides us a method of eliminating two field quantities from the system of Maxwell's equations.

In scenarios involving certain media on varying spatial scales (*e.g.*, situations involving ferroelectric or ferromagnetic media, or those considering microscopic material interactions), these constitutive relations can take different forms [63]. However, the ones we use correspond to the scenario of non-ferroelectric and non-ferromagnetic media on the macroscopic scale that may nevertheless be inhomogeneous or anisotropic. The two constitutive relations are, therefore, derived from Equations 2.6 and 2.14 under the assumptions that $\vec{P} \equiv 0$ and $\vec{M} \equiv 0$, and take the forms

$$\vec{D} = \varepsilon\vec{E} \tag{2.49}$$

and

$$\vec{B} = \mu\vec{H}, \tag{2.50}$$

together with Ohm's law, which describes how the electric field induces a current in conducting media (we consider the case where the external current density is zero):

$$\vec{J} = \sigma\vec{E}, \tag{2.51}$$

where σ is generally a bounded function of position, $\sigma : \mathbb{R}^3 \to \mathbb{R}$, and may be represented as a rank-two tensor.

We may rewrite Equations 2.5–2.13 using Equations 2.49, 2.50, and 2.51, in order to remove the dependency on $\vec{B}$, $\vec{E}$, and $\vec{J}$, as follows:

$$\nabla \circ \vec{E} = \frac{\rho}{\varepsilon} \tag{2.52}$$

$$\vec{\nabla} \circ \vec{H} = 0 \tag{2.53}$$

$$\vec{\nabla} \times \vec{E} = -\frac{\partial(\mu\vec{H})}{\partial t} \tag{2.54}$$

$$\vec{\nabla} \times (\mu\vec{H}) = \mu\left(\sigma\vec{E} + \varepsilon\frac{\partial\vec{E}}{\partial t}\right). \tag{2.55}$$

The system of Maxwell's equations and the three constitutive relations may be combined into a single wave equation via the following procedure. Equation 2.54 implies that

$$\vec{\nabla} \times \left(\vec{\nabla} \times \vec{E}\right) = \vec{\nabla} \times \left(-\frac{\partial(\mu\vec{H})}{\partial t}\right),$$

and when μ is assumed to be constant in space, and when $\vec{H}$ is assumed to be sufficiently smooth to permit interchanging the orders of the curl and differential operators, then we obtain

$$\vec{\nabla} \times \left(\vec{\nabla} \times \vec{E}\right) = -\frac{\partial\left(\vec{\nabla} \times (\mu\vec{H})\right)}{\partial t}. \tag{2.56}$$

We note here the useful vector operator identity

$$\vec{\nabla} \times \left(\vec{\nabla} \times \vec{A}\right) = \vec{\nabla}\left(\vec{\nabla} \circ \vec{A}\right) - \Delta\vec{A},$$

where $\vec{A}$ is an arbitrary vector field, and where Δ is the vector Laplace operator[5]. Applying this identity to Equation 2.56, we obtain

$$\vec{\nabla}\left(\vec{\nabla} \circ \vec{E}\right) - \Delta\vec{E} = -\frac{\partial\left(\vec{\nabla} \times (\mu\vec{H})\right)}{\partial t},$$

and into this, we substitute Equations 2.52 and 2.55, obtaining

$$\vec{\nabla}\left(\frac{\rho}{\varepsilon}\right) - \Delta\vec{E} = -\frac{\partial}{\partial t}\left[\mu\left(\sigma\vec{E} + \varepsilon\frac{\partial\vec{E}}{\partial t}\right)\right] = -\mu\frac{\partial(\sigma\vec{E})}{\partial t} - \mu\varepsilon\frac{\partial^2\vec{E}}{\partial t^2},$$

[5]In Cartesian coordinates, for $\vec{x} = \langle x, y, z\rangle$ and $\vec{A} = \langle A_x(\vec{x}), A_y(\vec{x}), A_z(\vec{x})\rangle$, the vector Laplace operator or the vector Laplacian, denoted $\Delta\vec{A}$ or $\nabla^2\vec{A}$, is expressed as $\Delta\vec{A} =: \nabla^2\vec{A} := \langle\Delta A_x, \Delta A_y, \Delta A_z\rangle$, where for a scalar field $B = B(\vec{x})$ in Cartesian coordinates, the scalar Laplace operator, alternatively called the scalar Laplacian, of $\vec{B}$ is denoted either ΔB or $\nabla \circ \nabla B$, and is defined as $\Delta B =: \nabla \circ \nabla B := \frac{\partial^2 B}{\partial x^2} + \frac{\partial^2 B}{\partial y^2} + \frac{\partial^2 B}{\partial z^2}$.

or, rearranging terms,

$$\Delta\vec{E} = \vec{\nabla}\left(\frac{\rho}{\varepsilon}\right) + \mu\frac{\partial(\sigma\vec{E})}{\partial t} + \mu\varepsilon\frac{\partial^2\vec{E}}{\partial t^2}.$$

Assuming that the charge density function ρ is identically zero, this reduces to the wave equation for the electric field:

$$\Delta\vec{E} = \mu\frac{\partial(\sigma\vec{E})}{\partial t} + \mu\varepsilon\frac{\partial^2\vec{E}}{\partial t^2}. \tag{2.57}$$

In general, a first-order derivative term in a second-order wave equation corresponds to damping or attenuation; the electromagnetic wave equation is no exception, as its first-order derivative term accounts for the attenuation of the electromagnetic waves in media, where the conductivity σ describes the transformation of electromagnetic energy of the incident field into internal electrical currents [64].

In three dimensions, where $\vec{x} = \langle x, y, z\rangle$ and $\vec{E} = \langle E_x(\vec{x},t), E_y(\vec{x},t), E_z(\vec{x},t)\rangle$, Equation 2.57 becomes

$$\begin{bmatrix} \frac{\partial^2 E_x}{\partial x^2} + \frac{\partial^2 E_x}{\partial y^2} + \frac{\partial^2 E_x}{\partial z^2} \\ \frac{\partial^2 E_y}{\partial x^2} + \frac{\partial^2 E_y}{\partial y^2} + \frac{\partial^2 E_y}{\partial z^2} \\ \frac{\partial^2 E_z}{\partial x^2} + \frac{\partial^2 E_z}{\partial y^2} + \frac{\partial^2 E_z}{\partial z^2} \end{bmatrix} = \mu\sigma \begin{bmatrix} \frac{\partial E_x}{\partial t} \\ \frac{\partial E_y}{\partial t} \\ \frac{\partial E_z}{\partial t} \end{bmatrix} + \mu\varepsilon \begin{bmatrix} \frac{\partial^2 E_x}{\partial t^2} \\ \frac{\partial^2 E_y}{\partial t^2} \\ \frac{\partial^2 E_z}{\partial t^2} \end{bmatrix}, \tag{2.58}$$

which corresponds to three equations—one for each of the x, y, and z coordinates.

The equation for the magnetic field can be derived in an analogous way from Equations 2.18 and 2.13, and is expressed as

$$\Delta\vec{H} = \mu\frac{\partial\vec{J}}{\partial t} + \mu\varepsilon\frac{\partial^2\vec{H}}{\partial t^2},$$

but as no one- or two-dimensional force vector arising from the electric field interacts with electric charges [65], we do not consider the magnetic field in our simulation.

Initial and Boundary Conditions for the Electromagnetic Wave Equations

Our boundary conditions model the situation where we have only one port of input for electromagnetic energy, located at the left-hand boundaries of the one- and two-dimensional domains shown in Figures 1.1 and 1.2, and denoted here as Γ_{port}. In particular, we assume that the value of the electromagnetic field on Γ_{port} is equal to the value of the incident field at that point; that is, on the left boundary, the field value is the constant

$$E_i(\vec{x},t) := \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}, \quad \vec{x}\in\Gamma_{\text{port}}, \quad t\in[0,\infty), \tag{2.59}$$

where L is the length of the waveguide in m; P is the power supplied by the magnetron in W; ω is the angular frequency of the incident waves in Hz (or $\frac{1}{\text{s}}$); $\mu_0 = 4\pi\times10^{-7}\frac{\text{N}}{\text{A}^2}$ is the permeability of free space; and $\beta = \frac{\pi}{L}$ is the propagation constant of the incident waves in $\frac{1}{\text{m}}$.

We assume that, initially, the amplitude of the wave is zero except at the port-side boundary; that is,

$$\vec{E}(\vec{x},0) = \begin{cases} 0, & x \notin \Gamma_{\text{port}}, \\ \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}, & x \in \Gamma_{\text{port}}. \end{cases} \tag{2.60}$$

At the right-hand boundary of the domain, which we denote Γ_{wall}, we may alternatively apply two different conditions. The assumption that this boundary represents a waveguide wall made from a material that perfectly conducts electricity leads to the Dirichlet condition

$$\vec{E}(\vec{x},t) \equiv 0, \quad \vec{x} \in \Gamma_{\text{wall}}, \quad t \in [0,\infty), \tag{2.61}$$

whereas, alternatively, we may wish to simulate a real-life domain longer than our computational scenario allows. In this case, we assume that the length of the actual domain is infinite, and that the right-hand boundary of our computational domain allows full propagation with no reflection. It is this scenario that is simulated by the *absorbing boundary condition* [2], and in one dimension, this condition is represented by

$$\left.\frac{\partial E_y}{\partial z}\right|_{z=L} = -\frac{1}{c}\left.\frac{\partial E_y}{\partial t}\right|_{z=L}, \qquad t \in [0,\infty), \tag{2.62}$$

where the use of variable z and electric field E_y is explained in Section 2.4 and shown in Figure 2.3.

For a general two-dimensional problem, no exact absorbing boundary condition exists, but we may write an approximation in the form of the *second-order absorbing boundary condition*, also referred to as the *Engquist-Majda absorbing boundary condition*, which takes the following form [2]:

$$\left.\frac{\partial^2 E_y}{\partial t \partial z}\right|_{z=L} \approx -\frac{1}{c}\left.\frac{\partial^2 E_y}{\partial t^2}\right|_{z=L} + \frac{c}{2}\left.\frac{\partial^2 E_y}{\partial x^2}\right|_{z=L}, \qquad t \in [0,\infty),\ x \in [0,H]. \tag{2.63}$$

One- and Two-Dimensional Initial Boundary Value Problems

In one dimension, we follow the scenario in Figure 2.3, and we assume that the electric field $\vec{E}$ consists of a wave propagating along the z-axis that has a nonzero component only in the y-direction. We therefore define $\vec{E}$ to be

$$\vec{E}(\vec{x},t) = \vec{E}(z,t) = \langle 0, E_y(z,t), 0\rangle,$$

and observe that in this case, the three equalities corresponding to the respective coordinates of Equation 2.58 all reduce to a single equation, and we restate the initial and boundary conditions given by Equations 2.60 and 2.62, respectively, to obtain the following initial boundary value problem.

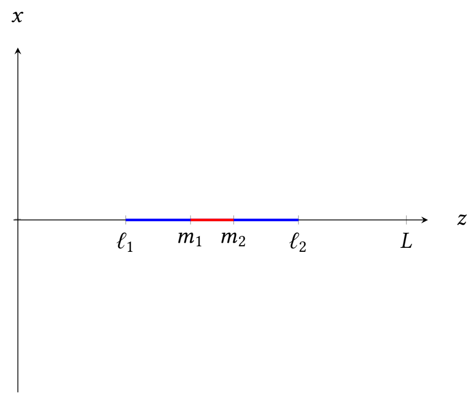


Figure 2.3: One-dimensional domain, where the interval $[m_1, m_2]$ (red) is assumed to be filled by the material undergoing sintering, $[\ell_1, m_1] \cup [m_2, \ell_2]$ (blue) is assumed to be filled by insulation, and $[0, \ell_1) \cup (\ell_2, L]$ (white) is assumed to be filled by air.

Problem 1. *Find $E_y(z,t)$ that satisfies*

$$\begin{cases} \dfrac{\partial^2 E_y}{\partial z^2} - \dfrac{\mu_r \varepsilon_r}{c^2}\dfrac{\partial^2 E_y}{\partial t^2} - \mu\sigma\dfrac{\partial E_y}{\partial t} = 0, \quad z \in (0,L), \quad t \in (0,\infty), \\ E_y(z,0) = 0, \quad z \in (0,L] \\ E_y(0,t) = \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}, \quad t \in [0,\infty), \\ \left.\dfrac{\partial E_y}{\partial z}\right|_{z=L} = -\frac{1}{c}\left.\dfrac{\partial E_y}{\partial t}\right|_{z=L}, \quad t \in [0,\infty), \end{cases} \tag{2.64}$$

where we have written $\mu = \mu_r\mu_0$ and $\varepsilon = \varepsilon_r\varepsilon_0$, and have used the relation between the speed of light and the permittivity and permeability of free space, i.e., $c^2 = 1/\sqrt{\mu_0\varepsilon_0}$.

In two dimensions, we assume the scenario in Figure 2.4, where plane waves propagate along the z-axis with transverse variation in the x-direction. We assume that the electric field has a nonzero component only in the y-direction, and so

$$\vec{E}(\vec{x},t) = \vec{E}(x,z,t) = \langle 0, E_y(x,z,t), 0\rangle.$$

In this scenario the three equalities corresponding to the respective coordinates of Equation 2.58 also all reduce to a single equation, which is shown with its initial and boundary conditions. Boundary conditions for the two transverse walls are derived from Equation 2.61.

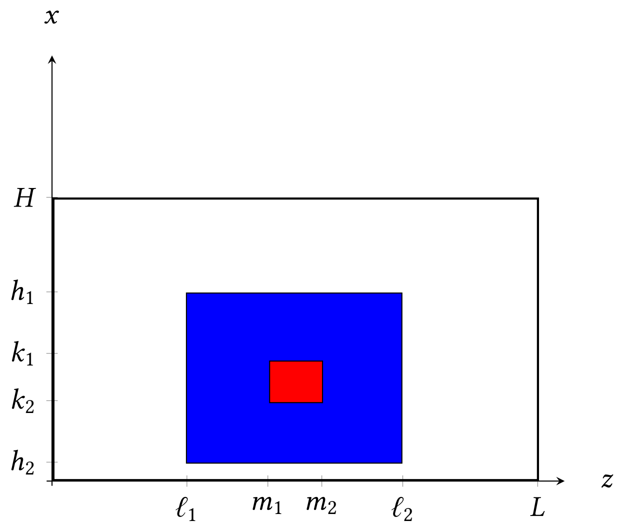


Figure 2.4: Two-dimensional domain, where the interval $[m_1, m_2] \times [k_2, k_1]$ (red) is assumed to be filled by the material undergoing sintering, $([\ell_1, \ell_2] \times [h_2, h_1]) \setminus ([m_1, m_2] \times [k_2, k_1])$ (blue) is assumed to be filled by insulation, and $([0, L] \times [0, H]) \setminus ([\ell_1, \ell_2] \times [h_2, h_1])$ (white) is assumed to be filled by air.

Problem 2. *Find $E_y(x, z, t)$ that satisfies*

$$\begin{cases} \frac{\partial^2 \vec{E_y}}{\partial x^2} + \frac{\partial^2 \vec{E_y}}{\partial z^2} - \frac{\mu_r \varepsilon_r}{c^2} \frac{\partial^2 \vec{E_y}}{\partial t^2} - \mu \frac{\partial(\sigma \vec{E_y})}{\partial t} = 0, \quad z \in (0, L), \quad x \in (0, H), \quad t \in (0, \infty) \\ E_y(x, z, 0) = 0, \quad (x, z) \in [0, H] \times (0, L], \\ E_y(x, 0, t) = \frac{2}{L} \sqrt{2P \left(\frac{\omega \cdot \mu_0}{\beta} \right)}, \quad x \in [0, H], \quad t \in [0, \infty), \\ \left. \frac{\partial^2 E_y}{\partial t \partial z} \right|_{z=L} \approx -\frac{1}{c} \left. \frac{\partial^2 E_y}{\partial t^2} \right|_{z=L} + \frac{c}{2} \left. \frac{\partial^2 E_y}{\partial x^2} \right|_{z=L}, \quad x \in [0, H], \quad t \in (0, \infty), \\ E_y(0, z, t) = E_y(H, z, t) = 0, \quad z \in (0, L), \quad t \in [0, \infty). \end{cases} \tag{2.65}$$

Nondimensionalized Initial Boundary Value Problems

We now proceed to nondimensionalize Problems 1 and 2 using the method proposed in [9]. Although we use the dimensional Problems 1 and 2 with the numerical solvers described in Chapter 7, we present the nondimensionalized versions here for completeness. In Table 2.2, we provide the dimensions of the quantities involved in Equation 2.64, along with typical units and names for reference.

In order to nondimensionalize the problem, we transform the variables and functions into di-

Symbol	Name	Units	Dimensions
$E_y(z,t)$	electric field	Vm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{T}^{-3}\mathscr{I}^{-1}$
z	space variable	m	$\mathscr{L}$
t	time variable	sec	$\mathscr{T}$
c	speed of light	$\mathrm{m{\cdot}sec}^{-1}$	$\mathscr{L}\mathscr{T}^{-1}$
$\mu_r := \frac{\mu}{\mu_0}$	relative permeability	1	1
μ_0	permeability of free space	Hm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{T}^{-2}\mathscr{I}^{-2}$
$\varepsilon_r := \frac{\varepsilon}{\varepsilon_0}$	relative permittivity	1	1
σ	electrical conductivity	Sm^{-1}	$\mathscr{M}^{-1}\mathscr{L}^{-3}\mathscr{T}^{3}\mathscr{I}^{2}$

Table 2.2: Dimensions of physical quantities in Equations Equation 2.64 and Equation 2.65.

mensionless quantities through the judicious choice of scales. We let

$$\hat{z} := \frac{z}{L_*}, \quad \hat{t} := \frac{t}{T_*}, \quad \text{and} \quad \hat{E_y}(\hat{z},\hat{t}) := \frac{E_y(z,t)}{E_*}, \tag{2.66}$$

where L_* is a constant with fundamental dimension $\mathscr{L}$, T_* is a constant with fundamental dimension $\mathscr{T}$, and E_* is a constant with fundamental dimension $\mathscr{M}\mathscr{L}\mathscr{T}^{-3}\mathscr{I}^{-1}$. Values for L_*, T_*, and E_* will be chosen after a cursory analysis of the result of applying these scales to Problem 1, as follows.

We consider the time and spatial derivatives involved in Equation 2.64, and rewrite them in terms of the dimensionless variables by use of the chain rule:

$$\begin{aligned}
\frac{\partial E_y}{\partial t} &= \frac{\partial}{\partial t}\left(E_*\hat{E}_y(\hat{z},\hat{t})\right) = E_*\frac{\partial \hat{u}}{\partial t} = E_*\frac{\partial \hat{E}_y}{\partial \hat{t}}\frac{\mathrm{d}\hat{t}}{\mathrm{d}t} = \frac{E_*}{T_*}\frac{\partial \hat{E}_y}{\partial \hat{t}},\\
\frac{\partial E_y}{\partial z} &= \frac{\partial}{\partial z}\left(E_*\hat{E}_y(\hat{z},\hat{t})\right) = E_*\frac{\partial \hat{E}_y}{\partial z} = E_*\frac{\partial \hat{E}_y}{\partial \hat{z}}\frac{\mathrm{d}\hat{z}}{\mathrm{d}z} = \frac{E_*}{L_*}\frac{\partial \hat{E}_y}{\partial \hat{z}},\\
\frac{\partial^2 E_y}{\partial t^2} &= \frac{\partial}{\partial t}\left(\frac{E_*}{T_*}\frac{\partial \hat{E}_y}{\partial \hat{t}}\right) = \frac{E_*}{T_*}\frac{\partial^2 \hat{E}_y}{\partial t\partial \hat{t}} = \frac{E_*}{T_*}\frac{\partial^2 \hat{E}_y}{\partial \hat{t}^2}\frac{\mathrm{d}\hat{t}}{\mathrm{d}t} = \frac{E_*}{T_*^2}\frac{\partial^2 \hat{E}_y}{\partial \hat{t}^2},\\
\frac{\partial^2 E_y}{\partial z^2} &= \frac{\partial}{\partial z}\left(\frac{E_*}{L_*}\frac{\partial \hat{E}_y}{\partial \hat{z}}\right) = \frac{E_*}{L_*}\frac{\partial^2 \hat{E}_y}{\partial z\partial \hat{z}} = \frac{E_*}{L_*}\frac{\partial^2 \hat{E}_y}{\partial \hat{z}^2}\frac{\mathrm{d}\hat{z}}{\mathrm{d}z} = \frac{E_*}{L_*^2}\frac{\partial^2 \hat{E}_y}{\partial \hat{z}^2}.
\end{aligned} \tag{2.67}$$

Substituting Equations 2.66 and 2.67 into Equation 2.64, we obtain the governing equation

$$\frac{E_*}{L_*^2}\frac{\partial^2 \hat{E}_y}{\partial \hat{z}^2} - \frac{\mu_r\varepsilon_r E_*}{c^2 T_*}\frac{\partial^2 \hat{E}_y}{\partial \hat{t}^2} - \frac{\mu_r\mu_0\sigma E_*}{T_*}\frac{\partial \hat{E}_y}{\partial \hat{t}} = 0, \quad \hat{z} \in \left(0, \frac{L}{L_*}\right), \quad \hat{t} \in (0,\infty),$$

which simplifies to

$$\frac{\partial^2 \hat{E}_y}{\partial \hat{t}^2} - \frac{T_*^2 c^2}{L_*^2\mu_r\varepsilon_r}\frac{\partial^2 \hat{E}_y}{\partial \hat{z}^2} + \frac{T_* c^2\mu_0\sigma}{\varepsilon_r}\frac{\partial \hat{E}_y}{\partial \hat{t}} = 0, \quad \hat{z} \in \left(0, \frac{L}{L_*}\right), \quad \hat{t} \in (0,\infty). \tag{2.68}$$

This simplified version suggests that an appropriate time scale may be the choice

$$T_* := \frac{L_*}{c}\sqrt{\mu_r \varepsilon_r} = L_*\sqrt{\mu_r \mu_0 \varepsilon_r \varepsilon_0} = L_*\sqrt{\mu\varepsilon} =: \frac{L_*}{v_p}, \tag{2.69}$$

where we have used $c = \frac{1}{\sqrt{\varepsilon_0 \mu_0}}$, together with the definitions of the relative permittivity and permeability, along with the definition of $\frac{1}{v_p} := \sqrt{\mu\varepsilon}$ as the phase velocity of the wave propagating through the media with permittivity ε and permeability μ [66]. This choice of scale has physical meaning: $\frac{L_*}{v_p}$ is the time it takes for a wave to propagate the entire length of the spatial domain.

Since it is sometimes convenient to choose a spatial scale over which the nondimensionalized space domain becomes the interval $[0, 1]$, Equation 2.68 also suggests a choice of space scaling constant. For the computational domain to be $[0, 1]$, we should take

$$L_* := L. \tag{2.70}$$

Equation 2.68 does not, however, suggest anything about an appropriate choice for the scale on the electric field. We may, instead, look to the initial and boundary conditions in Problem 1 for some guidance. Using Equation 2.66, the initial condition becomes

$$\hat{E_y}(\hat{z}, 0) = 0, \quad \hat{z} \in (0, 1], \tag{2.71}$$

and the boundary conditions become

$$\begin{aligned} \hat{E_y}(0, \hat{t}) &= \frac{2}{E_* L}\sqrt{2P\left(\frac{\omega\mu_0}{\beta}\right)}, \quad \hat{t} \in [0, \infty), \\ \left.\frac{\partial \hat{E_y}}{\partial \hat{z}}\right|_{\hat{z}=1} &= -\frac{v_p}{c}\left.\frac{\partial \hat{E_y}}{\partial \hat{t}}\right|_{\hat{z}=1}, \quad \hat{t} \in (0, \infty). \end{aligned} \tag{2.72}$$

It is also sometimes convenient to choose a scale for the electric field over which a nondimensional constant in the boundary and initial conditions may become identically 1. For this, we observe the left-hand boundary condition, and set

$$E_* := \vec{E}_{\text{inc}}\Big|_{z=0} = \frac{L}{2}\sqrt{\frac{\beta}{2P\omega\mu_0}}, \tag{2.73}$$

so that, using Equations 2.68–2.73, Problem 1 reduces to the following.

Problem 3. *Find $\hat{E_y}(\hat{z}, \hat{t})$ that satisfies*

$$\begin{cases} \frac{\partial^2 \hat{E_y}}{\partial \hat{t}^2} - \frac{\partial^2 \hat{E_y}}{\partial \hat{z}^2} + \frac{L\sigma}{\varepsilon v_p}\frac{\partial \hat{E_y}}{\partial \hat{t}} = 0, \quad \hat{z} \in (0, 1), \quad \hat{t} \in (0, \infty), \\ \hat{E_y}(\hat{z}, 0) = 0, \quad \hat{z} \in (0, 1], \\ \hat{E_y}(0, \hat{t}) = 1, \quad \hat{t} \in [0, \infty), \\ \left.\frac{\partial \hat{E_y}}{\partial \hat{z}}\right|_{\hat{z}=1} = -\frac{v_p}{c}\left.\frac{\partial \hat{E_y}}{\partial \hat{t}}\right|_{\hat{z}=1}, \quad \hat{t} \in (0, \infty). \end{cases} \tag{2.74}$$

For the two-dimensional problem, we use a similar process to nondimensionalize the governing equation, initial condition, and boundary conditions in Problem 2. Consider the scales on time, length, and temperature shown in Equations 2.69, 2.70, and 2.73, respectively; setting one more dimensionless variable $\hat{x} := \frac{x}{L_*}$ and applying these to Equation 2.65 results in the following problem.

Problem 4. *Find $\hat{E_y}(\hat{x},\hat{z},\hat{t})$ that satisfies*

$$\begin{cases} \frac{\partial^2 \hat{E_y}}{\partial \hat{t}^2} - \frac{\partial^2 \hat{E_y}}{\partial \hat{x}^2} - \frac{\partial^2 \hat{E_y}}{\partial \hat{z}^2} + \frac{L\sigma}{\varepsilon v_p}\frac{\partial \hat{E_y}}{\partial \hat{t}} = 0, \quad \hat{x} \in \left(0, \frac{H}{L}\right), \quad \hat{z} \in (0,1), \quad \hat{t} \in (0,\infty), \\ \hat{E_y}(\hat{x},\hat{z},0) = 0, \quad \hat{x} \in \left[0, \frac{H}{L}\right], \quad \hat{z} \in (0,1], \\ \hat{E_y}(\hat{x},0,\hat{t}) = 1, \quad \hat{x} \in \left[0, \frac{H}{L}\right], \quad \hat{t} \in [0,\infty), \\ \hat{E_y}(0,\hat{z},\hat{t}) = \hat{E_y}\left(\frac{H}{L},\hat{z},\hat{t}\right) = 0, \quad \hat{z} \in (0,1), \quad \hat{t} \in [0,\infty), \\ \frac{\partial^2 \hat{E_y}}{\partial \hat{t}\partial \hat{z}}\Big|_{\hat{z}=1} \approx -\frac{v_p}{c}\frac{\partial^2 \hat{E_y}}{\partial \hat{t}^2}\Big|_{\hat{z}=1} + \frac{c}{2v_p}\frac{\partial^2 \hat{E_y}}{\partial \hat{x}^2}\Big|_{\hat{z}=1}, \quad \hat{x} \in \left[0, \frac{H}{L}\right], \quad \hat{t} \in (0,\infty). \end{cases} \tag{2.75}$$

2.5 Time-Harmonic Electromagnetic Waves in the Waveguide

This section follows the procedure in [67] of deriving the system of equations to solve in a resonant cavity under the assumption of a time-harmonic electric and magnetic field. This assumption is valid because the electric and magnetic fields are induced by oscillating charges that produce incident waves of a fixed angular frequency. To this end, we introduce the dimensionless variables and parameters

$$\omega\sqrt{\mu_0\varepsilon_0}x \to x; \qquad \sqrt{\mu_0/\varepsilon_0}\vec{H} \to \vec{H}; \qquad \vec{E} \to \vec{E},$$

where ε_0 and μ_0 are the permittivity and permeability of free space, and their values are given in the List of Physical Constants.

We consider electromagnetic waves in a waveguide of cross section Ω, which is assumed to be a two-dimensional domain bounded by a smooth curve Γ, but which is otherwise arbitrary. The waveguide is assumed to be parallel to the z-axis in the cartesian coordinate system where $\vec{x} = \langle x,y,z\rangle$. In this case, the homogeneous system of time-harmonic Maxwell's equations written in the normalized form, with $\vec{\mathcal{E}}$ representing only the spatially dependent quantity in the expression $\vec{E}(\vec{x},t) = \vec{\mathcal{E}}(\vec{x})e^{-j\omega t}$ and with $\vec{\mathcal{H}}$ representing only the spatially dependent quantity in $\vec{H}(\vec{x},t) = \vec{\mathcal{H}}(\vec{x})e^{-j\omega t}$, is given by:

$$\begin{aligned} \vec{\nabla} \times \vec{\mathcal{E}} &= -i\vec{\mathcal{H}}, \quad \vec{x} \in \Sigma, \\ \vec{\nabla} \times \vec{\mathcal{H}} &= i\varepsilon\vec{\mathcal{E}}, \\ \vec{\mathcal{E}}(\vec{x}) &= \langle E_x(\vec{x}') \quad E_y(\vec{x}') \quad E_z(\vec{x}')\rangle e^{i\gamma z}, \\ \vec{\mathcal{H}}(\vec{x}) &= \langle H_x(\vec{x}') \quad H_y(\vec{x}') \quad H_z(\vec{x}')\rangle e^{i\gamma z}, \\ \vec{x}' &= (x,y), \end{aligned}$$

with the boundary conditions for the tangential electric field components on the perfectly conducting surfaces of the waveguide given by

$$\vec{E}_\tau\Big|_{\Gamma_{\text{wall}}} = 0. \tag{2.76}$$

Written componentwise, these are equivalent to the system

$$\frac{\partial H_3}{\partial x_2} - i\gamma H_2 = i\varepsilon E_1, \quad \frac{\partial E_3}{\partial x_2} - i\gamma E_2 = -i\varepsilon H_1, \quad i\gamma H_1 - \frac{\partial H_3}{\partial x_1} = i\varepsilon E_2,$$
$$i\gamma E_1 - \frac{\partial E_3}{\partial x_1} = -i\varepsilon H_2, \quad \frac{\partial H_2}{\partial x_2} - \frac{\partial H_1}{\partial x_2} = i\varepsilon E_3, \quad \frac{\partial E_2}{\partial x_1} - \frac{\partial E_1}{\partial x_2} = -i\varepsilon H_3.$$

Combining these and setting $\tilde{k}^2 := \varepsilon - \gamma^2$, we obtain

$$E_1 = \frac{i}{\tilde{k}^2}\left(\gamma\frac{\partial E_3}{\partial x_1} - \frac{\partial H_3}{\partial x_2}\right), \quad E_2 = \frac{i}{\tilde{k}^2}\left(\gamma\frac{\partial E_3}{\partial x_2} - \frac{\partial H_3}{\partial x_1}\right),$$
$$H_1 = \frac{i}{\tilde{k}^2}\left(\varepsilon\frac{\partial E_3}{\partial x_2} + \gamma\frac{\partial H_3}{\partial x_1}\right), \quad H_2 = \frac{i}{\tilde{k}^2}\left(-\varepsilon\frac{\partial E_3}{\partial x_1} + \gamma\frac{\partial H_3}{\partial x_2}\right),$$

which is valid when $\gamma^2 \neq \varepsilon_2$ and $\gamma^2 \neq \varepsilon_2$.

From Equation 2.76, we note that the field of a normal wave can be expressed via two scalar functions

$$\Pi(x_1,x_2) = E_3(x_1,x_2), \qquad \Psi(x_1,x_2) = H_3(x_1,x_2).$$

For the TE-polarization, the particular solutions $[\vec{\mathcal{E}},\vec{\mathcal{H}}]$ are found by setting $E_3 \equiv 0$, whereas for the TM-polarization, the solutions are found by setting $H_3 \equiv 0$.

When we set $\gamma = 0$ in Equation 2.5, this is equivalent to removing the dependence of the electric and magnetic fields on the longitudinal coordinate x_3, and in this case, the two fundamental polarizations provide two separate problems for the sets of component functions—in the case of TE-polarization, we solve for component functions $[E_1,E_2,H_3]$, and in the case of TM-polarization, we solve for component functions $[H_1,H_2,E_3]$. In this way, the problem on normal waves is reduced to a boundary eigenvalue problem for functions Π and Ψ:

Problem 5. *Find $\gamma \in \mathbb{C}$ such that there exist nontrivial solutions of the Helmholtz equations*

$$\Delta\Pi + \tilde{k}^2\Pi = 0, \qquad \vec{x}' = (x_1,x_2) \in \Omega,$$

$$\Delta\Psi + \tilde{k}^2\Psi = 0, \qquad \tilde{k}^2 = \varepsilon - \gamma^2,$$

satisfying the boundary conditions on Γ_0

$$\Pi\Big|_{\Gamma_0} = 0, \qquad \frac{\partial\Psi}{\partial n}\Big|_{\Gamma_0} = 0.$$

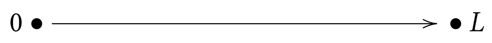


Figure 2.5: One-dimensional domain on which Helmholtz equation is described.

The γ for which there exist Π and Ψ satisfying the above conditions are referred to as eigenvalues of normal waves in a waveguide filled by a homogeneous substance. Again, here, ε is the permittivity of the matter filling the waveguide, and $\tilde{k}^2 = \varepsilon - \gamma^2$. Here, Ω is the cross section of the waveguide, and Γ is the smooth curve that bounds Ω. We assume $\Gamma_0 \subset \Gamma$. The waveguides that we consider are, nevertheless, not filled homogeneously.

For a one-dimensional domain where $z \in [0, L]$, as in Figure 2.5, where the electric field depends on the space variable z and the time variable t, and exists only in the x-direction, the assumption of a time-harmonic electric field leads to the phasor representation

$$\vec{\mathcal{E}}(z,t) = \left\langle \operatorname{Re}\left\{E(z)e^{i\omega t}\right\}, 0, 0\right\rangle, \tag{2.77}$$

where $E(z)$ satisfies the Helmholtz equation

$$\frac{\partial}{\partial z}\left(\mu^{-1}\frac{\partial \mathcal{E}(z)}{\partial z}\right) = \mu\omega^2\varepsilon\mathcal{E}(z), \tag{2.78}$$

derived from Problem 5.

In two dimensions, the Helmholtz equation describes the x-component of the electric field that varies in the yz-plane and with time:

$$\vec{\mathcal{E}}(x,z,t) = \left\langle \operatorname{Re}\left\{E(x,z)e^{i\omega t}\right\}, 0, 0\right\rangle, \tag{2.79}$$

where $E(y,z)$ satisfies

$$\left(\frac{\partial^2\mathcal{E}(x,z)}{\partial z^2} + \frac{\partial^2\mathcal{E}(x,z)}{\partial x^2}\right) = \mu^2\omega^2\varepsilon\mathcal{E}(x,z), \tag{2.80}$$

assuming that μ may be taken outside of the space derivatives, and where ε and μ refer to the absolute permittivity and permeability, respectively, rather than the relative permittivity and permeability ε_{rel} and μ_{rel}.

In the two-dimensional case, the boundaries representing transverse walls are assumed to satisfy the perfect electric conducting condition in Equation 2.61, which implies

$$\mathcal{E}(x,z)\Big|_{(x,z)\in\Gamma_{\text{wall}}} = 0. \tag{2.81}$$

For both the one- and two-dimensional problems, the right-hand boundary of the computational domain is assumed to represent the far-side wall in a physical scenario, and so either the absorbing boundary condition, or the perfect electric conductor condition represents a physically plausible scenario. For our simulations, we apply the perfect electric conductor condition from

Equation 2.81. In both one- and two-dimensional scenarios, the left-hand boundary is assumed to represent the port-side wall in a physical scenario, and so we apply the Dirichlet condition

$$\mathcal{E}(x,z)\Big|_{(x,z)\in\Gamma_{\text{wall}}} = E_{\text{inc}}, \tag{2.82}$$

where E_{inc} is the value of the incident field at the port.

Chapter 3

The Thermal Problem

The process of sintering is driven by the addition of thermal energy to a system. In this chapter, we derive a way of modelling temperature within the sample under certain assumptions on the domain and the nature of the solutions, and we discuss the limitations of our model in its physical context.

3.1 Physical Derivation of Heat Conduction

The problem of heat conduction within media begins with energy considerations. The law of conservation of energy, applied to the specific case of thermal energy in an arbitrary volume V is as follows.

Law 1 (Conservation of Thermal Energy [68]). *The rate of change of thermal energy in V with respect to time is equal to the net flow of energy across the surface of V, plus the rate at which heat is generated within V.*

This statement involves the consideration of three quantities, which we define in the following way. Let $\vec{x} := \langle x, y, z \rangle$ denote a point within V, let $\rho := \rho(\vec{x}, \Theta)$ denote the density of the matter within V, where Θ is the work of sintering parameter discussed in Section 4.3, and let $e(\vec{x}, t)$ denote the specific internal energy of the solid; *i.e.*, the energy per unit mass. Then the amount of thermal energy in V is given by the expression

$$(\text{Total thermal energy within } V) = \int_V \rho(\vec{x}, \Theta) e(\vec{x}, t)\ \mathrm{d}V.$$

For many materials, over fairly wide but not large temperature ranges, the function $e(\vec{x}, t)$ depends nearly linearly on the temperature [68]. If we denote temperature by the function $u(\vec{x}, t)$, then the amount of thermal energy in V becomes

$$(\text{Total thermal energy within } V) = \int_V \rho(\vec{x}, \Theta) c_p u(\vec{x}, t)\ \mathrm{d}V, \tag{3.1}$$

where c_p is the constant of proportionality referred to as the *specific heat capacity*. The specific heat capacity depends on the material filling V, and is defined as the amount of energy required to raise a unit mass of this material by one unit in temperature [69].

From this expression, we may derive the one defined in Law 1 by differentiating with respect to time:

$$\begin{aligned}(\text{Rate of change of thermal energy within } V) &= \frac{\mathrm{d}}{\mathrm{d}t}\left[\int_V \rho(\vec{x},\Theta)c_p u(\vec{x},t)\ \mathrm{d}V\right]\\ &= \int_V \frac{\partial}{\partial t}\left[\rho(\vec{x},\Theta)c_p u(\vec{x},t)\right]\ \mathrm{d}V\\ &= \int_V \rho(\vec{x},\Theta)c_p\frac{\partial u(\vec{x},t)}{\partial t} + \frac{\partial \rho(\vec{x},\Theta)}{\partial t}c_p u(\vec{x},t)\ \mathrm{d}V,\end{aligned} \tag{3.2}$$

assuming the hypotheses of the Leibniz theorem, which allows us to differentiate under the integral only when both $\vec{u}(\vec{x},t)$, $\frac{\partial u(\vec{x},t)}{\partial t}$, and $\frac{\partial \rho(\vec{x},\Theta)}{\partial t}$ are continuous with respect to t.

To obtain mathematical expressions for the other two terms defined in Law 1, we first assume that the arbitrary volume V is bounded by the piecewise smooth surface B, and let $\vec{q} := \vec{q}(\vec{x},t) := \langle q_1(\vec{x},t), q_2(\vec{x},t), q_3(\vec{x},t)\rangle$ denote the heat flux vector. Then the amount of heat per unit time flowing into V across B is given by the expression

$$(\text{Thermal energy flowing into } V) = -\oint_B \vec{q}\circ\hat{n}\ \mathrm{d}S,$$

where $\hat{n}$ denotes a unit vector pointing outward normally from B on the area element $\mathrm{d}S$ at the point $\vec{x}$. The negative sign in front of the surface integral accounts for the concept that if more heat flows out of V than into V, the energy in V decreases.

Now, under the assumption that B is piecewise smooth, V is compact, and $\vec{q}(\vec{x},t)$ is continuously differentiable with respect to each of its spatial variables, we may apply the Divergence Theorem[1] to obtain

$$(\text{Thermal energy flowing into } V) = -\int_V \nabla\circ\vec{q}(\vec{x},t)\ \mathrm{d}V. \tag{3.3}$$

We may use *Fourier's law of conduction* to write $\vec{q}$ in terms of the temperature. Fourier's law accounts for two facts that are easily observed in nature: first, that heat flows from areas of higher temperature to areas of lower temperature, and second, that the rate of heat flow from a hot area to a cold one depends on the difference in temperature between those two areas, with a greater temperature difference yielding a faster rate of heat transfer. In order to cast Fourier's law in mathematical notation, we let $\hat{s}$ represent a unit vector, pointing in any direction, with its tail at $\vec{x}$. Then the rate of heat flow at $\vec{x}$ in the direction of $\hat{s}$ is given by $\vec{q}\circ\hat{s}$, and the rate of change of temperature at $\vec{x}$ in the direction of $\hat{s}$ is given by the directional derivative $\mathrm{D}_{\hat{s}}(u) =: \nabla_{\hat{s}} =: \frac{\partial u}{\partial \hat{s}} = \nabla u\circ\hat{s}$. With these notions in place, we may recast Fourier's law as follows.

[1]See Footnote 1 on Page 37 for the statement of the Divergence Theorem.

Law 2 (Fourier's Law of Conduction [68]). *The rate of heat transfer through a point $\vec{x}$ in the direction of an arbitrary unit vector $\hat{s}$ is proportional to the rate of change in temperature at $\vec{x}$ in the direction of $\hat{s}$,* i.e.,

$$\vec{q} \circ \hat{s} = -k\nabla u \circ \hat{s},$$

and as the direction $\hat{s}$ is arbitrary, the temperature flux must be generally proportional to the temperature gradient, i.e.,

$$\vec{q}(\vec{x},t) = -k\nabla u(\vec{x},t).$$

The constant k of proportionality in Law 2 is referred to as the *thermal conductivity*, and is a property of the matter that occupies V. In general, the matter in V may be inhomogeneous and anisotropic, in which case $k = k(\vec{x})$ may be either a scalar or tensor function of position. However, we consider the case of homogeneous, isotropic media. A detailed discussion of the physical proof of Fourier's law, and how to measure k for a variety of materials, may be found in [70].

Using Law 2, Equation 3.3 may be rewritten as

$$(\text{Thermal energy flowing into } V) = \int_V k\nabla \circ \nabla u(\vec{x},t)\ \mathrm{d}V =: \int_V k\Delta u(\vec{x},t)\ \mathrm{d}V, \tag{3.4}$$

using the symbol Δ to denote the scalar Laplace operator[2].

Finally, heat may also be generated within V by various physical mechanisms, of which we assume *a priori* knowledge. We account for these by letting $f(\vec{x},t)$ denote the rate at which heat is produced by these sources of sinks per unit volume, and we refer to the following expression as the corresponding *source term*, which is the final quantity of interest from Law 1:

$$(\text{Rate at which heat is generated within } V) = \int_V f(\vec{x},t)\ \mathrm{d}V. \tag{3.5}$$

Using Equations 3.2, 3.4, and 3.5, Law 1 may be recast mathematically as

$$\int_V \rho(\vec{x},\Theta)c_p\frac{\partial u(\vec{x},t)}{\partial t} + \frac{\partial \rho(\vec{x},\Theta)}{\partial t}c_p u(\vec{x},t)\ \mathrm{d}V = \int_V k\Delta u(\vec{x},t)\ \mathrm{d}V + \int_V f(\vec{x})\ \mathrm{d}V,$$

and since all integrals are over the same volume, we obtain

$$\int_V \rho(\vec{x},\Theta)c_p\frac{\partial u(\vec{x},t)}{\partial t} + \frac{\partial \rho(\vec{x},\Theta)}{\partial t}c_p u(\vec{x},t) - k\Delta u(\vec{x},t) - f(\vec{x},t)\ \mathrm{d}V = 0.$$

Because the volume V was initially supposed to be arbitrary, we conclude that

$$\rho(\vec{x},\Theta)c_p\frac{\partial u(\vec{x},t)}{\partial t} + \frac{\partial \rho(\vec{x},\Theta)}{\partial t}c_p u(\vec{x},t) - k\Delta u(\vec{x},t) - f(\vec{x},t) = 0,$$

[2] The scalar Laplace operator, also known as the scalar Laplacian, is as defined in Footnote 5 on Page 54; that is, for $B = B(\vec{x})$, $\Delta B :=: \nabla \circ \nabla B := \frac{\partial^2 B}{\partial x^2} + \frac{\partial^2 B}{\partial y^2} + \frac{\partial^2 B}{\partial z^2}$.

Applying the chain rule to the partial derivative in the second term, we obtain

$$\begin{aligned} c_p u(\vec{x},t)\frac{\partial \rho(\vec{x},\Theta)}{\partial t} &= c_p u(\vec{x},t)\frac{\partial \rho(\vec{x},\Theta)}{\partial \Theta(t,u)}\frac{\partial \Theta(t,u)}{\partial t} \\ &= c_p u(\vec{x},t)\frac{\partial \rho(\vec{x},\Theta)}{\partial \Theta(t,u)}\frac{\partial}{\partial t}\left[\int_0^t \frac{1}{u(\vec{x},\tau)}\exp\left(\frac{-Q}{Ru(\vec{x},\tau)}\right)\,\mathrm{d}\tau\right] \\ &= c_p u(\vec{x},t)\frac{\partial \rho(\vec{x},\Theta)}{\partial \Theta(t,u)}\left[\frac{1}{u(\vec{x},\tau)}\exp\left(\frac{-Q}{Ru(\vec{x},\tau)}\right)\right] \\ &= c_p \frac{\partial \rho(\vec{x},\Theta)}{\partial \Theta(t,u)}\exp\left(\frac{-Q}{Ru(\vec{x},\tau)}\right). \end{aligned}$$

Because the activation energy Q is, for our materials, so large in comparison with the universal gas constant R, and because the densification of materials typically occurs relatively slowly, this term becomes small enough to neglect in our consideration of the thermal problem. Neglecting this term, the heat equation becomes

$$\rho(\vec{x},\Theta)c_p\frac{\partial u(\vec{x},t)}{\partial t} - k\Delta u(\vec{x},t) - f(\vec{x},t) = 0, \tag{3.6}$$

which is sometimes written

$$\frac{\partial u(\vec{x},t)}{\partial t} - \kappa\Delta u(\vec{x},t) = F(\vec{x},t),$$

where $\kappa := \frac{k}{\rho(\vec{x})c_p}$ is referred to as the *thermal diffusivity*, and $F(\vec{x},t) := \frac{1}{\rho(\vec{x})c_p}f(\vec{x},t)$.

Equation 3.6 is referred to as the heat equation, but also appears in a great variety of problems in mathematical physics, *e.g.*, the concentration of diffusing material, the motion of tidal waves in long channels, transmission in electrical cables, and unsteady boundary layers in viscous fluid flows [71].

3.2 Source Term

The physical scenario we consider in the microwave heating problem is one where the source of heat in the load is the power dissipated by the changing electric field. This stands in contrast to the heating scenario in a conventional oven, where the mechanism responsible for adding thermal energy to the load is convection; in the microwave scenario, heat does not enter the load through its boundaries due to surrounding hot air, but rather is generated directly within the interior of the object [72]. Therefore, in this scenario, the source term $f(\vec{x},t)$ must account for dissipated power. The time-averaged power dissipated into the domain is represented by the quantity

$$P_{\text{diss}} = -\frac{1}{2}\,\mathrm{Re}\left\{\oint_S \vec{E}\times\vec{H}^*\circ\hat{n}\,\mathrm{d}S\right\}, \tag{3.7}$$

and as discussed in Section 2.2, this integral may be rewritten under the assumptions that the magnetic field does not affect the course of heating, and that the electric field is harmonic in time. In this case, Equations 2.44 and 2.39 may be used to rewrite Equation 3.7 as

$$P_{\text{avg}} = \frac{1}{2}\omega\varepsilon''|\vec{\mathcal{E}}|^2 = \omega\varepsilon''|\vec{E}|^2_{\text{avg}}. \tag{3.8}$$

It is this quantity that is used as the source term $f(\vec{x},t)$ in Equation 3.6, so that for the case of microwave heating,

$$\rho(\vec{x})c\frac{\partial u(\vec{x},t)}{\partial t} - k\Delta u(\vec{x},t) = \omega\varepsilon''|\vec{E}|^2_{\text{avg}}. \tag{3.9}$$

3.3 Initial and Boundary Conditions

The initial condition applied to the situation of microwave sintering is that the entire domain starts at room temperature; that is,

$$u(\vec{x},0) = T_0, \tag{3.10}$$

where T_0 is the constant room temperature.

The heat equation can be used with various kinds of boundary conditions to model certain physical scenarios. Here, as in Section 2.5, we assume a three-dimensional domain that is bounded by a smooth, two-dimensional curve Γ, but which is otherwise arbitrary.

Common boundary conditions for simple scenarios include combinations of the Dirichlet condition, where the boundary is held at a fixed temperature:

$$u(\vec{x},t)\Big|_{\vec{x}\in\Gamma} = G(\vec{x},t) \tag{3.11}$$

where $G(\vec{x},t)$ specifies the temperature at which the boundary is fixed, and the Neumann condition, where the flux is held fixed on the boundary:

$$\nabla u(\vec{x},t)\Big|_{\vec{x}\in\Gamma} \circ \hat{n} = H(\vec{x},t), \tag{3.12}$$

where $\hat{n}$ represents the unit vector pointing outward from Ω, normal to Γ at $\vec{x}$, and where $H(\vec{x},t)$ specifies the heat flux at the boundary. For our case, we use *Newton's law of cooling*, stated below and confirmed by numerous physical experiments over centuries [73], to derive a mixed boundary condition.

Law 3 (Newton's Law of Cooling [73]). *The rate of change of temperature of an object is proportional to the difference between its own temperature and the ambient temperature.*

To state Newton's law of cooling quantitatively, we let $\vec{q}(\vec{x})$ represent the heat flux vector, and we define T_{amb} as the ambient temperature (typically, this is the same as T_0 in the initial condition). Then Newton's law of cooling states

$$\vec{q}(\vec{x},t) \circ \hat{n} = h\left(u(\vec{x},t) - T_{\text{amb}}\right),$$

where, again, $\hat{n}$ is the unit vector pointing outward from Ω, normal to Γ at $\vec{x}$, and where h is a constant of proportionality with dimensions $\mathscr{M}^{-1}$. Applying Law 2 to this formulation, we obtain

$$-k\nabla u(\vec{x},t) \circ \hat{n} = h\left(u(\vec{x},t) - T_{\text{amb}}\right),$$

from which we rewrite the proportionality constant $\tilde{h} := \frac{h}{k}$ to obtain the boundary condition

$$\nabla u(\vec{x},t) \circ \hat{n} = -\tilde{h}\left(u(\vec{x},t) - T_{\text{amb}}\right). \tag{3.13}$$

3.4 One- and Two-Dimensional Initial Boundary Value Problems

In one space dimension, we consider the spatial variable $\vec{x} = \langle 0,0,z\rangle$, where $z \in [\ell_1,\ell_2]$ as shown in Figure 2.3, and the time variable $t \in [0,\infty)$. Note that we are interested only in modelling temperature within the insulation and the load, and so we exclude $[0,\ell_1) \cup (\ell_2, L]$ from the spatial domain. In this case, Equation 3.6, with the source term in Equation 3.9 and the initial and boundary conditions in Equations 3.10 and 3.13, respectively, reduces to the following initial boundary value problem.

Problem 6. *Find $u(z,t)$ that satisfies*

$$\begin{cases} \rho c_p \frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial z^2} = \omega\varepsilon''|\vec{E}|^2_{avg}, & z \in (\ell_1,\ell_2), \quad t \in (0,\infty), \\ u(z,0) = T_0, \quad x \in [\ell_1,\ell_2], \\ \left.\frac{\partial u}{\partial z}\right|_{z=\ell_1} = \tilde{h}\left(u(\ell_1,t) - T_{amb}\right), \quad t \in (0,\infty), \\ \left.\frac{\partial u}{\partial z}\right|_{z=\ell_2} = -\tilde{h}\left(u(\ell_2,t) - T_{amb}\right), \quad t \in (0,\infty). \end{cases} \tag{3.14}$$

In two space dimensions, we consider the spatial variable $\vec{x} = \langle x,0,z\rangle$, where $z \in [\ell_1,\ell_2]$ and $x \in [h_1,h_2]$ as shown in Figure 2.4, and the time variable $t \in [0,\infty)$. Note that, again, we are interested in modelling temperature within only the insulation and load, so we exclude the other regions from the spatial domain. In this case, Equation 3.6, with the source term in Equation 3.9 and the initial and boundary conditions in Equations 3.10 and 3.13, respectively, reduces to the following initial boundary value problem.

Problem 7. *Find $u(x,z,t)$ that satisfies*

$$\begin{cases} \rho c_p \frac{\partial u}{\partial t} - k\frac{\partial^2 u}{\partial x^2} - k\frac{\partial^2 u}{\partial z^2} = \omega\varepsilon''|\vec{E}|^2_{avg}, & z \in (\ell_1,\ell_2), \quad x \in (h_1,h_2), \quad t \in (0,\infty), \\ u(x,z,0) = T_0, \quad z \in [\ell_1,\ell_2], \quad x \in [h_1,h_2], \\ \left.\frac{\partial u}{\partial z}\right|_{z=\ell_1} = \tilde{h}\left(u(x,\ell_1,t) - T_{amb}\right), \quad x \in [m_1,m_2], \quad t \in (0,\infty), \\ \left.\frac{\partial u}{\partial z}\right|_{z=\ell_2} = -\tilde{h}\left(u(x,\ell_2,t) - T_{amb}\right), \quad x \in [m_1,m_2], \quad t \in (0,\infty), \\ \left.\frac{\partial u}{\partial x}\right|_{x=m_1} = \tilde{h}\left(u(m_1,z,t) - T_{amb}\right), \quad z \in [\ell_1,\ell_2], \quad t \in (0,\infty), \\ \left.\frac{\partial u}{\partial x}\right|_{x=m_2} = -\tilde{h}\left(u(m_2,z,t) - T_{amb}\right), \quad z \in [\ell_1,\ell_2], \quad t \in (0,\infty). \end{cases} \tag{3.15}$$

Analytical and numerical techniques for solving and approximating the solutions of these problems will be discussed in Chapter 8.

3.5 Nondimensionalized Initial Boundary Value Problems

In this section, we nondimensionalize Problems 6 and 7 using the method proposed in [9]. Though the problems whose computer simulation routines are discussed in Chapter 8 are the dimensional ones in Problems 6 and 7, the nondimensionalization process is presented here for completeness.

One-Dimensional Problem

In Table 3.1, we provide the dimensions of the quantities involved in Equation (3.14), along with typical units and names for reference. In order to nondimensionalize the problem, we transform the variables and functions into dimensionless quantities through the judicious choice of scales. We let

$$\hat{z} := \frac{z-\lambda_*}{L_*}, \quad \hat{t} := \frac{t}{T_*}, \quad \hat{u}(\hat{z},\hat{t}) := \frac{u(z,t)}{U_*}, \quad \text{and} \quad \hat{q}(\hat{z},\hat{t}) := \frac{q(z,t)}{U_*}, \tag{3.16}$$

where $q(z,t) := \omega\varepsilon''|\vec{E}|^2_{\text{avg}}$ is the source term, L_* and λ_* are constants with fundamental dimension $\mathscr{L}$, T_* is a constant with fundamental dimension $\mathscr{T}$, and U_* is a constant with fundamental dimension Θ. Values for L_*, T_*, and U_* will be chosen after a cursory analysis of the result of applying these scales to Problem 6, as follows.

Symbol	Name	Units	Dimensions
$u(z,t)$	temperature	K	Θ
z	space variable	m	$\mathscr{L}$
t	time variable	sec	$\mathscr{T}$
ρ	mass density	gm^{-3}	$\mathscr{M}\mathscr{L}^{-3}$
c_p	specific heat capacity	JK^{-1}	$\mathscr{L}^2\mathscr{T}^{-2}\Theta^{-1}$
k	thermal conductivity	$\text{Wm}^{-1}\text{K}^{-1}$	$\mathscr{M}\mathscr{L}\mathscr{T}^{-3}\Theta^{-1}$
ω	angular frequency	$\text{sec}^{-1}\text{rad}$	$\mathscr{T}^{-1}$
ε	electrical permittivity	Fm^{-1}	$\mathscr{T}^4\mathscr{I}^2\mathscr{L}^{-2}\mathscr{M}^{-1}$
$\vec{E}$	electric field	Vm^{-1}	$\mathscr{M}\mathscr{L}\mathscr{T}^{-3}\mathscr{I}^{-1}$

Table 3.1: Dimensions of physical quantities in Equations 3.14 and 3.15.

We consider the time and spatial derivatives involved in Equation 3.14, and rewrite them in terms of the dimensionless variables by use of the chain rule:

$$\begin{aligned}
\frac{\partial u}{\partial t} &= \frac{\partial}{\partial t}\left(U_*\hat{u}(\hat{z},\hat{t}\right) = U_*\frac{\partial \hat{u}}{\partial t} = U_*\frac{\partial \hat{u}}{\partial \hat{t}}\frac{\mathrm{d}\hat{t}}{\mathrm{d}t} = \frac{U_*}{T_*}\frac{\partial \hat{u}}{\partial \hat{t}} \\
\frac{\partial u}{\partial z} &= \frac{\partial}{\partial z}\left(U_*\hat{u}(\hat{z},\hat{t}\right) = U_*\frac{\partial \hat{u}}{\partial z} = U_*\frac{\partial \hat{u}}{\partial \hat{z}}\frac{\mathrm{d}\hat{z}}{\mathrm{d}z} = \frac{U_*}{L_*}\frac{\partial \hat{u}}{\partial \hat{z}} \\
\frac{\partial^2 u}{\partial z^2} &= \frac{\partial}{\partial z}\left(\frac{U_*}{L_*}\frac{\partial \hat{u}}{\partial \hat{z}}\right) = \frac{U_*}{L_*}\frac{\partial^2 \hat{u}}{\partial z \partial \hat{z}} = \frac{U_*}{L_*}\frac{\partial^2 \hat{u}}{\partial \hat{z}^2}\frac{\mathrm{d}\hat{z}}{\mathrm{d}z} = \frac{U_*}{L_*^2}\frac{\partial^2 \hat{u}}{\partial \hat{z}^2}.
\end{aligned} \tag{3.17}$$

Substituting Equations 3.16 and 3.17 into Equation 3.14, we obtain the governing equation

$$\frac{U_* k}{L_*^2}\frac{\partial^2 \hat{u}}{\partial \hat{z}^2} - \frac{U_* \rho c_p}{T_*}\frac{\partial \hat{u}}{\partial \hat{t}} = U_* \hat{q}(\hat{z},\hat{t}), \quad \hat{z} \in \left[\frac{\ell_1 - \lambda_*}{L_*}, \frac{\ell_2 - \lambda_*}{L_*}\right], \quad t \in (0,\infty),$$

which simplifies to

$$\frac{\partial \hat{u}}{\partial \hat{t}} - \frac{T_* k}{L_*^2 \rho c_p}\frac{\partial^2 \hat{u}}{\partial \hat{z}^2} = \frac{T_*}{\rho c_p}\hat{q}(\hat{z},\hat{t}), \quad \hat{z} \in \left[\frac{\ell_1 - \lambda_*}{L_*}, \frac{\ell_2 - \lambda_*}{L_*}\right], \quad t \in (0,\infty) \tag{3.18}$$

This simplified version suggests that an appropriate time scale may be the choice

$$T_* := \frac{L_*^2 \rho c_p}{k}, \tag{3.19}$$

which indeed has the correct dimensions of $\mathscr{T}$. Since it is sometimes convenient to choose a spatial scale over which the nondimensionalized space domain becomes the interval $[0,1]$, Equation 3.18 also suggests a choice of space scaling constants. For the computational domain to be $[0,1]$, we should take

$$\lambda_* := \ell_1, \quad \text{and} \quad L_* := \ell_2 - \ell_1. \tag{3.20}$$

Equation 3.18 does not, however, suggest anything about an appropriate choice for the temperature scale. We may, instead, look to the initial and boundary conditions in Problem 6 for some guidance. Using Equation 3.16, the initial condition becomes

$$\hat{u}(\hat{z},0) = \frac{T_0}{U_*}, \quad \hat{z} \in \left[\frac{\ell_1 - \lambda_*}{L_*}, \frac{\ell_2 - \lambda_*}{L_*}\right] = [0,1], \tag{3.21}$$

and the boundary conditions become

$$\begin{aligned} \frac{\partial \hat{u}}{\partial \hat{z}}\Big|_{\hat{z}=0} &= \tilde{h} L_* \left(\hat{u}(0,\hat{t}) - \frac{T_{\text{amb}}}{U_*}\right), \quad \hat{t} \in (0,\infty), \\ \frac{\partial \hat{u}}{\partial \hat{z}}\Big|_{\hat{z}=1} &= -\tilde{h} L_* \left(\hat{u}(1,\hat{t}) - \frac{T_{\text{amb}}}{U_*}\right), \quad \hat{t} \in (0,\infty). \end{aligned} \tag{3.22}$$

It is, again, sometimes convenient to choose a temperature scale over which a nondimensional constant in the boundary and initial conditions may become identically 1. Since it is typically the case that $T_0 \equiv T_{\text{amb}}$, we may let

$$U_* := T_0, \tag{3.23}$$

so that, using Equations 3.18–3.23, Problem 6 reduces to the following.

Problem 8. *Find a function $\hat{u}(\hat{z},\hat{t})$ that satisfies the following:*

$$\begin{cases} \frac{\partial \hat{u}}{\partial \hat{t}} - \frac{\partial^2 \hat{u}}{\partial \hat{z}^2} = \frac{(\ell_2 - \ell_1)^2}{k}\hat{q}(\hat{z},\hat{t}), \quad \hat{z} \in (0,1), \quad t \in (0,\infty), \\ \hat{u}(\hat{z},0) = 1, \quad \hat{z} \in [0,1], \\ \frac{\partial \hat{u}}{\partial \hat{z}}\Big|_{\hat{z}=0} = \tilde{h}(\ell_2 - \ell_1)\left(\hat{u}(0,\hat{t}) - 1\right), \quad \hat{t} \in (0,\infty), \\ \frac{\partial \hat{u}}{\partial \hat{z}}\Big|_{\hat{z}=1} = -\tilde{h}(\ell_2 - \ell_1)\left(\hat{u}(1,\hat{t}) - 1\right), \quad \hat{t} \in (0,\infty). \end{cases} \tag{3.24}$$

Two-Dimensional Problem

For the two-dimensional problem, we use a similar process to nondimensionalize the governing equation, initial condition, and boundary conditions in Problem 7. Consider the scales on time, length, and temperature shown in Equations 3.19, 3.20, and 3.23, respectively; setting one more dimensionless variable $\hat{x} := \frac{x}{L_*}$ and applying these to Equation 3.15 results in the following problem.

Problem 9. *Find a function $\hat{u}(\hat{x},\hat{z},\hat{t})$ that satisfies the following:*

$$\begin{cases} \frac{\partial \hat{u}}{\partial \hat{t}} - \frac{\partial^2 \hat{u}}{\partial \hat{z}^2} - \frac{\partial^2 \hat{u}}{\partial \hat{x}^2} = \frac{(\ell_2-\ell_1)^2}{k}\hat{q}(\hat{z},\hat{t}), \quad \hat{z} \in (0,1), \quad \hat{x} \in \left(\frac{h_1-\ell_1}{\ell_2-\ell_1}, \frac{h_2-\ell_1}{\ell_2-\ell_1}\right) \quad t \in (0,\infty), \\ \hat{u}(\hat{z},\hat{x},0) = 1, \quad \hat{z} \in [0,1], \quad \hat{x} \in \left[\frac{h_1-\ell_1}{\ell_2-\ell_1}, \frac{h_2-\ell_1}{\ell_2-\ell_1}\right], \\ \left.\frac{\partial \hat{u}}{\partial \hat{z}}\right|_{\hat{z}=0} = \tilde{h}(\ell_2-\ell_1)\left(\hat{u}(\hat{x},0,\hat{t})-1\right), \quad \hat{x} \in \left[\frac{h_1-\ell_1}{\ell_2-\ell_1}, \frac{h_2-\ell_1}{\ell_2-\ell_1}\right], \quad \hat{t} \in (0,\infty), \\ \left.\frac{\partial \hat{u}}{\partial \hat{z}}\right|_{\hat{z}=1} = -\tilde{h}(\ell_2-\ell_1)\left(\hat{u}(\hat{x},1,\hat{t})-1\right), \quad \hat{x} \in \left[\frac{h_1-\ell_1}{\ell_2-\ell_1}, \frac{h_2-\ell_1}{\ell_2-\ell_1}\right], \quad \hat{t} \in (0,\infty), \\ \left.\frac{\partial \hat{u}}{\partial \hat{x}}\right|_{\hat{x}=\frac{h_1-\ell_1}{\ell_2-\ell_1}} = \tilde{h}(\ell_2-\ell_1)\left(\hat{u}\left(\frac{h_1-\ell_1}{\ell_2-\ell_1},\hat{z},\hat{t}\right)-1\right), \quad \hat{z} \in [0,1], \quad \hat{t} \in (0,\infty), \\ \left.\frac{\partial \hat{u}}{\partial \hat{x}}\right|_{\hat{x}=\frac{h_2-\ell_1}{\ell_2-\ell_1}} = -\tilde{h}(\ell_2-\ell_1)\left(\hat{u}\left(\frac{h_2-\ell_1}{\ell_2-\ell_1},\hat{z},\hat{t}\right)-1\right), \quad \hat{z} \in [0,1], \quad \hat{t} \in (0,\infty). \end{cases} \tag{3.25}$$

Chapter 4

Mechanical Deformation in the Course of Sintering

Sintering is a process through which particulate materials undergo thermal treatment and change their microstructure via granular growth, grain merging and neck formation, and growth of the necks between particles. This microstructural change leads to altered effective material properties, including density and strength, yet the medium undergoing sintering does not necessarily undergo any phase changes, and may or may not be pressurized during sintering. This thermal processing has traditionally been done in conventional ovens, but increasing interest has been shown in using microwaves as the heat source instigating this process [21, 53]. Mechanical deformation due to sintering can be observed on the *macroscale* (*i.e.*, by the naked eye), but is caused by *microscale* changes within the material undergoing processing (*i.e.*, changes on the spatial scale around the grain size, typically micro- or nanometers, and invisible without the use of imaging technology), via the action of several different transport mechanisms [74]. There may be up to six different paths of matter transport during solid-state sintering, and in practice, more than one of these mechanisms may operate simultaneously during processing [75].

In this chapter, we discuss some of these physical phenomena that influence the process of microwave sintering, and some of the ways of accounting for mechanical deformation in terms of the microstructural variables involved with the process, as accurate determination of the relationship between temperature, relative density and the microstructural variables remains one of the most challenging aspects of modelling sintering. Finally, we discuss the use of the Master Sintering Curve as a method of representing the density kinetics along a thermal cycle.

4.1 Physical Mechanisms Influencing the Progress of Sintering

The forces that give rise to the phenomenon of sintering, and its resulting reduction in the free energy of the system, are referred to as *driving forces of sintering* [75]. In this work, we characterize the curvature of particle forces as the principal driving force of sintering, because we assume the

Mechanism	Contribution
Surface diffusion	Coarsening, neck growth
Lattice diffusion from surface	Coarsening, neck growth
Vapor transport	Neck growth
Grain boundary diffusion	Densification, neck growth
Lattice diffusion from boundary	Densification, neck growth
Viscous flow	Densification, neck growth

Table 4.1: Physical mass transport mechanisms occurring during sintering.

absence of the two other most commonly observed driving forces: externally applied pressures and chemical reactions. Because the decrease in free surface energy is accompanied by an increase in the energy associated with the grain boundaries, these boundaries influence the magnitude of the driving force.

For polycrystalline materials, the driving force of sintering occurs when matter is transported along definite paths, from regions of higher chemical potential to those of lower chemical potential [75]. There are at least six known mechanisms for this transport, shown in Table 4.1 along with a depiction in Figure 4.1.

The mechanisms may be classified as those which contribute to densification, and those which contribute to grain coarsening, processes that are visualized in Figure 4.2. Coarsening reduces the driving force necessary for densification—but the coarsening mechanisms also reduce the curvature of the neck surface between the particles, and in this way they reduce the rate of the densifying mechanisms [75]. Coarsening and densification may be individually characterized by the contribution they make to the total reduction of energy in the system [10]:

$$\Delta(\gamma A) = \Delta(\gamma)A + \gamma\Delta(A),$$

where γ refers to the surface/interfacial energy, and A refers to the surface/interfacial area. On the right-hand side, $\Delta(\gamma)A$ refers to the contribution of densification via reduction in interfacial energy, and $\gamma\Delta(A)$ refers to the contribution of grain coarsening via reduction in the interfacial area. A seventh operating mechanism of sintering, atomic diffusion, has been proposed and studied, but the process may invoke chemical reactions and phase changes [24], which are beyond the scope of practical tools for modelling free solid-phase sintering.

4.2 The Constitutive Equation

One of the main challenges of modelling the sintering process lies in determining an appropriate phenomenological constitutive relation that preserves the integrity of microstructural changes while accurately characterizing the resulting effects on the material's macrostructural configuration. Such a constitutive relation obeys laws governing the process of grain coarsening and densification, characterizing all macroscale changes in terms of the effect of the microstructural variables on the strain

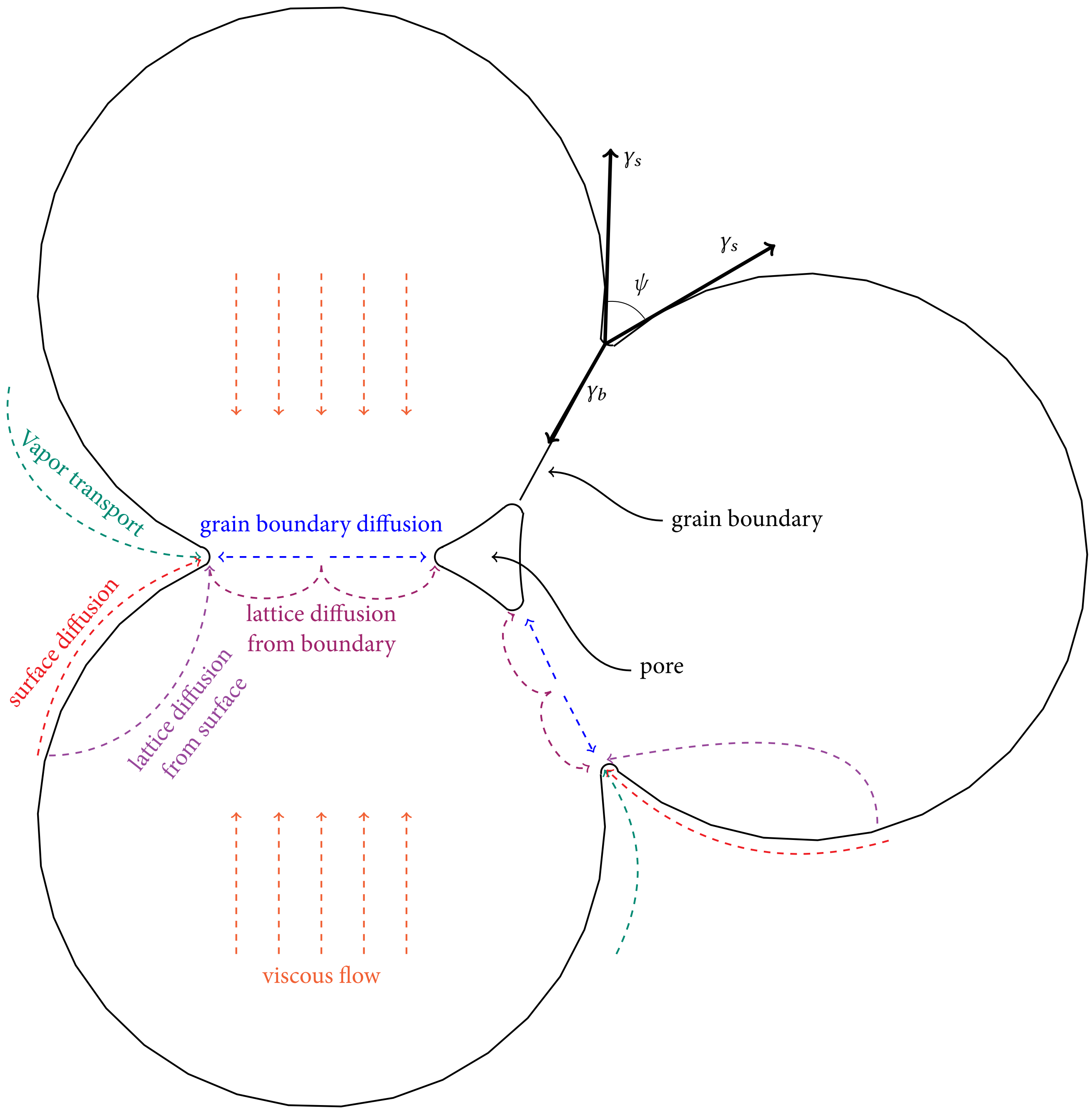


Figure 4.1: Particles undergoing sintering, with material transport paths indicated by colored, dashed arrows, and with the grain boundary γ_b tension, surface tensions γ_s, and dihedral angle ψ indicated.

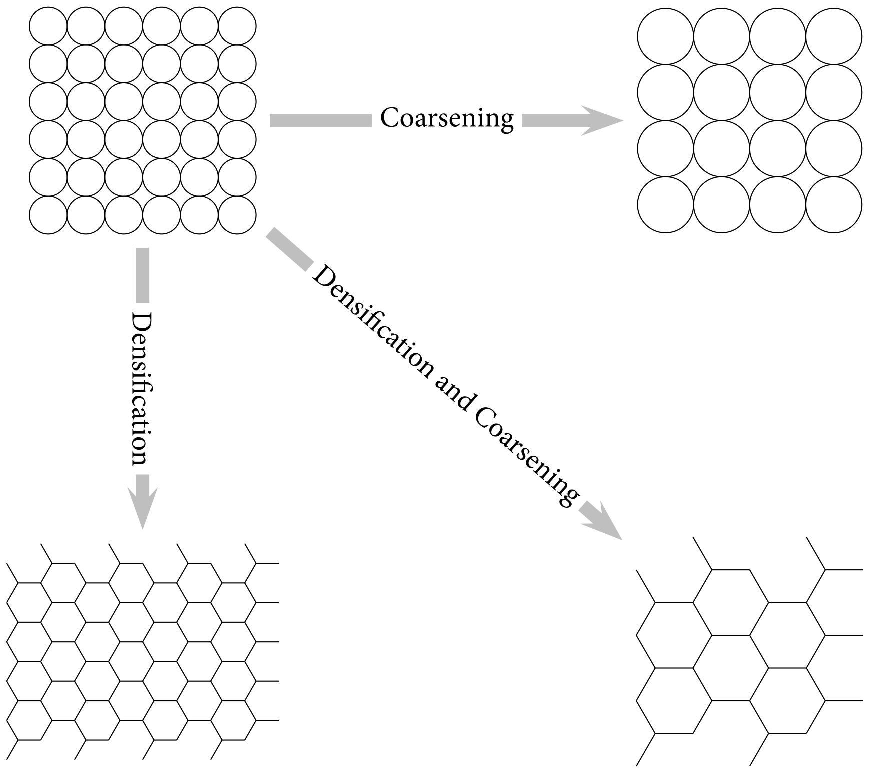


Figure 4.2: Densification and grain coarsening during sintering.

rate tensor.

Mechanical deformation in the course of sintering is modelled by examining the effect of thermal processing on the strain rate tensor inside the powder being sintered. The decomposition of the strain rate tensor is the following [24]:

$$\underline{\dot{\varepsilon}} = \underline{\dot{\varepsilon}}^e + \underline{\dot{\varepsilon}}^t + \underline{\dot{\varepsilon}}^s + \underline{\dot{\varepsilon}}^v, \tag{4.1}$$

where $\underline{\dot{\varepsilon}}^e$ is the elasticity strain rate tensor; $\underline{\dot{\varepsilon}}^t$ is the thermal expansion strain rate tensor; $\underline{\dot{\varepsilon}}^s$ is the free sintering strain rate tensor; and $\underline{\dot{\varepsilon}}^v$ is the viscous deformation strain rate tensor. We examine the latter two quantities individually, and determine

$$\underline{\dot{\varepsilon}}^s = -\dot{\rho}\frac{\underline{\delta}}{3} = \frac{-\sigma_s}{K}\frac{\underline{\delta}}{3}, \tag{4.2}$$

where $\underline{\delta}$ is the unit tensor, σ_s is the sintering stress ($\sigma_s = K\dot{\varepsilon}_s$), $\dot{\rho}$ is the free sintering densification rate, and K is the bulk viscosity.

The viscous strain rate is described by the Newtonian law

$$\underline{\dot{\varepsilon}}^v = -\frac{\Delta p}{3K}\underline{\delta} + \frac{\underline{\sigma}'}{2G}, \tag{4.3}$$

where Δp is the hydrostatic pressure, $\underline{\sigma}'$ is the deviatoric stress tensor, and G is the shear viscosity.

A constitutive equation describing sintering will express $\underline{\dot{\varepsilon}}^t$, $\dot{\rho}$, K, and G as functions of temperature, green density, relative density, grain size, and other parameters that should be either measured experimentally, or simulated by modelling [24]. The approach of [32] is an integrated one, combining the continuum theory of sintering with a kinetic Monte-Carlo model that simulates microstructural evolution in order to determine grain growth, pore migration, and densification; these parameters determine the sintering stress and normalized bulk viscosity that are subsequently used in the continuum-scale model.

Combining Equations 4.1, 4.2, and 4.3 gives the strain rate tensor in the course of sintering as

$$\underline{\dot{\varepsilon}} = \frac{\underline{\sigma}'}{2G} + \underline{\delta}\frac{-\sigma_s + \Delta p}{3K}, \tag{4.4}$$

which agrees with the model in [33] for simulating second (open porosity) and third (closed porosity) stage sintering and grain growth in porous solids. This model provides one way of utilizing the information that the MSC method provides about the density kinetics along temperature cycles, which we discuss in Section 4.3. In the subsections that follow, we discuss the key features of sintering that this model is capable of incorporating, with each subsection dedicated to a parameter that is an input to Equation 4.4.

Densification rate

The densification rate during sintering is described by

$$\dot{\rho} = \rho\frac{\sigma_s - \sigma_m - \Delta p}{K},$$

where ρ represents the density of the sample, and σ_m represents the mean (hydrostatic) stress. In the case of pressureless sintering, $\sigma_m \equiv 0$.

Bulk and Shear Viscosities

If we assume that grain boundaries are not perfect sources and sinks for vacancies, then there arises a nonlinear effect referred to as *source-controlled diffusion* or *interface reaction-controlled diffusion* [76, 77], which influences the description of the bulk and shear viscosities. The viscosities are expressed, respectively, as the quadratic terms

$$K = K_{\text{lin}}\left(1 + \frac{\alpha}{\overline{\sigma}a^2}\right) \quad \text{and} \quad G = G_{\text{lin}}\left(1 + \frac{\alpha}{\overline{\sigma}a^2}\right),$$

where a is the grain radius, α is a constant parameter determining the deviation from linearity, where

$$\overline{\sigma} = \frac{1}{2}|\sigma_m - \sigma_s + \Delta p| + \frac{1}{2}\sigma_e$$

is referred to as an *effective stress*, and where σ_e is the von Mises stress equivalent. The factors K_{lin} and G_{lin} are given by

$$K_{\text{lin}} = \frac{k_B u a^3}{\Omega(\delta D_b)}\left[(1-\theta)k_1 + \theta k_2\right]U, \quad \text{and} \quad G_{\text{lin}} = \frac{k_B u a^3}{\Omega(\delta D_b)}\left[(1-\theta)g_1 + \theta g_2\right]U,$$

where k_B is the Boltzmann constant, u is the absolute temperature, Ω is the atomic (or molecular) volume, δD_b is the grain boundary diffusion coefficient times the thickness of the grain boundary, θ is a parameter ranging from 0 to 1 that is introduced in order to represent a smooth transition from open to closed porosity, k_1 and k_2 are the normalized bulk viscosities for open and closed porosity, respectively, g_1 and g_2 are the normalized shear viscosities for open and closed porosities, respectively, and U is a factor describing the effect of grain rearrangement.

The parameter δD_b follows an Arrhenius-type dependence on temperature, with activation energy Q and pre-exponential factor δD_{b0}. The relative density at which transition from open to closed porosity occurs is $\rho_{\text{cl}} = 1.05 - 0.115\psi$, where $\cos\psi = \frac{\gamma_b}{2\gamma_s}$ defines the dihedral angle, as shown in Figure 4.1, and γ_b and γ_s are the interfacial tensions of the grain boundary and surface, respectively.

Grain Boundary Diffusion

Grain boundary diffusion is the primary mechanism by which densification occurs, and is similar to *Coble creep*, which is diffusional creep by grain boundary diffusion [78]. With this interpretation, the normalized bulk viscosities k_1 and k_2 assume the forms

$$k_i = \frac{k_{ib} + k_{is}}{k_{ib} + k_{is} + k_{iv}},$$

where the subscript v indicates volume (bulk) diffusion, b indicates grain boundary diffusion, s indicates surface boundary diffusion, and $i \in \{1,2\}$ indicates, as before, open or closed porosity.

The values at the open porosity stage are [79–81]

$$\begin{aligned} k_{1b} &= \begin{cases} A_0 + A_1 f + A_2 f^2, & \text{if } \rho_{\text{rel}} \geq 0.68, \\ A_9 \exp\left(-A_{10}(0.32 - f)\right), & \text{if } \rho_{\text{rel}} < 0.68, \end{cases} \\ k_{1s} &= \frac{\delta D_b}{\delta D_s}\left(\frac{-2\ln\Phi - (3-\Phi)(1-\Phi)}{2(1-\Phi)^2}\right), \\ k_{1v} &= \frac{\delta D_b}{0.6 a D_v} k_{1b}, \end{aligned} \tag{4.5}$$

where $f := 1 - \rho_{\text{rel}}$ is referred to as the *porosity*, δD_s is the surface diffusion coefficient, which also exhibits an Arrhenius-like dependence on temperature: $\delta D_s := \delta D_{s0}\exp\left(\frac{-Q}{Ru}\right)$, where D_v is the bulk diffusion coefficient, where $\Phi := 2(A_3 + A_4 f)^2$, and where the expressions A_0 through A_{10} are given in Appendix C.

At the closed porosity stage, the normalized bulk viscosities are as follows:

$$\begin{aligned} k_{2b} &= \frac{1}{18\rho_{\text{rel}}}\left(-2\ln\omega_b - \frac{33}{64} + \omega_b - \frac{\omega_b^2}{16}\right), \\ k_{2s} &= \frac{\delta D_b}{\delta D_s}\frac{1}{\rho_{\text{rel}}}(A_5 + A_6\omega_b + A_7\omega_b^2) \\ k_{2v} &= \frac{\delta D_b}{0.6aD_v}\frac{1}{18\rho_{\text{rel}}}\left(-2\ln\omega - \frac{33}{64} + \omega - \frac{\omega_b^2}{16}\right), \end{aligned}$$

where $\omega_b := A_8 f_b^{2/3}$ is the area fraction of grain boundaries covered by pores, $\omega := A_8 f^{2/3}$ is a factor. The distinction between ω and ω_b is made since during grain growth, pores may detach from migrating grain boundaries. The volume fraction of pores that remain on grain boundaries is given by

$$f_b = \begin{cases} f, & \text{if } \rho_{\text{rel}} < \rho_{\text{cl}} \text{ or } f > f_d, \\ \beta_0 f - (\beta_0 - 1)f_d, & \text{if } \rho_{\text{rel}} > \rho_{\text{cl}} \text{ and } f_d > f > \frac{\beta_0 - 1}{\beta_0} f_d, \\ 0, & \text{if } f < \frac{\beta_0-1}{\beta_0} f_d, \end{cases}$$

where β_0 describes the width of the range over which pore detachment occurs. The authors of [82] chose $\beta_0 = 1.3$, f_d is the porosity at which detachment occurs theoretically according to the condition [28, 83]

$$a^2 f_d \frac{1 - f_d^{1/3}}{1 - \omega_d} = 4.5\frac{\Omega\delta D_s}{k_B u M_b},$$

where $\omega_d := A_8 f_d^{2/3}$, and M_b is a factor accounting for the grain boundary mobility.

For open porosity [79, 84], $g_1 = \beta_1 k_1$, where β_1 is the ratio of shear (g_1) to bulk (k_1) viscosity and has the estimated upper bound $\beta_1 \leq 0.6^{68} \approx 8.21 \times 10^{-16}$ for freely sliding grain boundaries, and $\beta_1 = 0.27^{54} \approx 1.97 \times 10^{-31}$ as a self-consistent estimate.

For closed porosity,

$$g_2 = \beta_2 \frac{(g_{2\text{b}} + g_{2\text{s}})g_{2\text{v}}}{g_{2\text{b}} + g_{2\text{s}} + g_{2\text{v}}},$$

where

$$\begin{aligned} g_{2\text{b}} &= \frac{1}{\rho_{\text{rel}}}(0.029 - 0.022\sqrt{\omega_\text{b}}) \\ g_{2\text{s}} &= \frac{k_{2\text{s}}}{k_{2\text{b}}} g_{2\text{b}} \\ g_{2\text{v}} &= \frac{1}{\rho_{\text{rel}}}(0.029 - 0.022\sqrt{\omega})\frac{\delta D_\text{b}}{0.6aD_\text{v}}, \end{aligned}$$

and by the self-consistent estimate in [80], $\beta_2 = 1$.

Interpolation between open and closed porosity

The transition parameter θ varies from 0 to 1 in a density range from ρ_{lo} to ρ_{hi}:

$$\theta = \begin{cases} 0 & \text{for } \rho_{\mathrm{rel}} \le \rho_{\mathrm{lo}} \\ 1 - \cos\left(\frac{\pi}{2}\frac{\rho_{\mathrm{rel}}-\rho_{\mathrm{lo}}}{\rho_{\mathrm{hi}}-\rho_{\mathrm{lo}}}\right) & \text{for } \rho_{\mathrm{lo}} < \rho_{\mathrm{rel}} < \rho_{\mathrm{hi}} \\ 1 & \text{for } \rho_{\mathrm{rel}} \ge \rho_{\mathrm{hi}}. \end{cases}$$

In [82], the values of ρ_{lo} and ρ_{hi} are taken to be $\rho_{\mathrm{lo}} = \rho_{\mathrm{cl}} - 0.04$ and $\rho_{\mathrm{hi}} = \rho_{\mathrm{cl}} + 0.04$, where 0.04 is the arbitrarily chosen width of the transition range, and the relative density at pore closure obtained from $\rho_{\mathrm{cl}} = 1.05 - 0.115\psi$, with $\cos\psi = \frac{\gamma_{\mathrm{b}}}{2\gamma_{\mathrm{s}}}$.

Particle rearrangement

The phenomenological term for grain rearrangement takes the form

$$U := \begin{cases} \left(\frac{\rho_{\mathrm{rel}}-\rho_0+0.02}{0.63-\rho_0+0.02}\right)^{\zeta} & \text{for } \rho_{\mathrm{rel}} < 0.63 \\ 1 & \text{for } \rho_{\mathrm{rel}} > 0.63, \end{cases}$$

where ρ_0 is the initial ('green') relative density, and the numbers 0.63 and 0.02 are chosen arbitrarily, according to [82]. The above formulation accounts for the fact that rearrangement can contribute to densification and deformation only in the initial sintering stages. Above a certain density (here 63%, the relative density of a random dense sphere packing), rearrangement can make no further contribution to densification. If the parameter ζ is zero, the rearrangement term has no influence.

Sintering Stress

The sintering stress is given by

$$\sigma_{\mathrm{s}} := ((1-\theta)\sigma_{\mathrm{s1}} + \theta\sigma_{\mathrm{s2}})\frac{\gamma_{\mathrm{s}}}{a},$$

where

$$\sigma_{\mathrm{s1}} := \begin{cases} C_0 + C_1 f + C_2 f^2 & \text{for } \rho_{\mathrm{rel}} \ge 0.68 \\ C_5 \exp(-C_6(0.32 - f)) & \text{for } \rho_{\mathrm{rel}} < 0.68 \end{cases}$$

$$\sigma_{\mathrm{s2}} := 2\left(C_3\frac{\rho_{\mathrm{rel}}}{f}\right)^{1/3} + \frac{2\sqrt{3}+1}{2}\left(\frac{3\rho_{\mathrm{rel}}}{\pi}\right)^{1/3}\cos\psi,$$

where C_0–C_6 are given in Appendix C.

Gas Pressure

When gas is entrapped within the microstructure by a closed pore, the overpressure in the pore is

$$\Delta p = p_{\text{ex,cl}} = \frac{1-\rho_{\text{cl}}}{1-\rho_{\text{rel}}} \frac{\rho_{\text{rel}}}{\rho_{\text{cl}}} \frac{u}{u_{\text{cl}}} - p_{\text{ex}},$$

where the subscript cl denotes the values of density, temperature and external pressure, p_{ex}, at the time of pore closure. In [82], the sintering stress in the processing of SiC was found to be on the order of 3.5 MPa, whereas the gas overpressure is less than 0.3 MPa at relative densities up to 98%, so we conclude that the effect of gas pressure on the evolution of sintering is negligible.

Grain Coarsening

A model of grain coarsening may be derived from the Hillert law [85]

$$\dot{a} = \frac{\gamma_b M_b}{4a} \frac{F_d}{F_p},$$

where M_p is the grain mobility boundary and exhibits the Arrhenius-type temperature dependence $M_b = M_{b0} \exp\left(-\frac{Q}{R_g u}\right)$.

A modification to the Hillert law [82] introduces the factor F_d to account for the fact that the powder usually does not have the steady-state grain size distribution, which is implicit in the Hillert solution. Take

$$F_d := \frac{1}{1-\delta a_0/a},$$

where a_0 is the initial average grain radius and δ can lie between $-\infty$ and 1. $\delta = 0$ is sufficient for simulating the sintering of silicon carbide (SiC) [82], which corresponds to the Hillert law with no correction.

A second modification accounts for the fact that pores can exert drag on migrating grain boundaries, which suggests introduction of the factor F_p. For open porosity ($\rho_{\text{rel}} < \rho_{\text{cl}}$), we set

$$F_p := 1 - D_3\sqrt{f} + D_2 a^2 f^{3/2} \frac{k_B u M_b}{\Omega \delta D_s},$$

while for closed porosity ($\rho > \rho_{\text{cl}}$):

$$F_p := 1 - \omega_b + D_1 a^2 f_b^{4/3} \frac{k_B u M_b}{\Omega \delta D_s},$$

where the D terms are found in Appendix C.

4.3 Master Sintering Curve

Assuming an isotropic free sintering strain rate, the stress tensor will depend on the relative density. Because it gives a way of defining the relationship between temperature cycle and density evolution, the Master Sintering Curve (MSC) method [74] provides a way of describing $\underline{\dot{\varepsilon}_s}$ along any thermal cycle.

In order to characterize the relationship between relative density, temperature, and heating rate in our model, we use the MSC, in which the parameters comprising the sintering rate equation are separated, with those related to the microstructure isolated from those related to the temperature; the two sides of the equation are then related to each other experimentally.

The combined-stage sintering model gives the instantaneous rate of linear shrinkage of material as [74]

$$-\frac{\mathrm{d}L}{L\mathrm{d}t} = \frac{\gamma\Omega}{k_B u}\left(\frac{\Gamma_v D_v}{G^3} + \frac{\Gamma_b D_b}{16a^4}\right), \tag{4.6}$$

where, as before, γ is the surface energy, Ω the atomic volume, k_B the Boltzmann constant, u the absolute temperature, a the grain radius, D_b and D_v the coefficients for grain boundary and volume diffusion, respectively, and δ is the width of the grain boundary. $G(\rho)$ represents the mean grain diameter, and is assumed to be a function only of the density ρ. The scaling parameters Γ_v and Γ_b relate the instantaneous linear shrinkage rate to the diffusion coefficient and other material parameters, and to the mean grain radius [86]. These values may be determined experimentally or with the use of simplified sintering models, but, with few exceptions, are typically independent of the thermal cycle [86–88].

For isotropic shrinkage, the linear shrinkage rate can be converted to the densification rate by

$$-\frac{\mathrm{d}L}{L\mathrm{d}t} = \frac{\mathrm{d}\rho_{\mathrm{rel}}}{3\rho_{\mathrm{rel}}\mathrm{d}t},$$

where, again ρ_{rel} represents the relative density of the sample. If there exists only one dominant diffusion mechanism (either volume diffusion or grain boundary diffusion) in the sintering process, then Equation 4.6 can be simplified to

$$\frac{\mathrm{d}\rho}{3\rho\mathrm{d}t} = \frac{\gamma\Omega(\Gamma(\rho_{\mathrm{rel}}))D_0}{k_B u(G(p))^n}\exp\left(-\frac{Q}{Ru}\right),$$

where Q is the apparent activation energy, R is the gas constant, $D_0 := (D_v)_0$ and $n = 3$ for volume diffusion, $D_0 := (\delta D_b)_0$ and $n = 4$ for grain boundary diffusion, and where Γ represents a lumped scaling parameter incorporating the components Γ_v and Γ_b above. It is assumed here that G and Γ are functions of only density. Integrating, we obtain

$$\int_{\rho_0}^{\rho_{\mathrm{rel}}} \frac{G(r))^n}{3\rho_{\mathrm{rel}}\Gamma(r)}\,\mathrm{d}r = \int_0^t \frac{\gamma\Omega D_0}{k_B u}\exp\left(-\frac{Q}{Ru}\right)\,\mathrm{d}\tau, \tag{4.7}$$

where ρ_0 is the green density of the material.

All terms on the left-hand side of this equation are quantities related to microstructural evolution of the sample, and if $G(\rho_{\text{rel}})$ and $\Gamma(\rho_{\text{rel}})$ are functions only of density, then the individual terms on the left-hand side are independent of thermal history. With further rearrangement, the left-hand side of Equation 4.7 becomes

$$\Psi(\rho_{\text{rel}}) \equiv \frac{k_B}{\gamma \Omega D_0} \int_{\rho_0}^{\rho_{\text{rel}}} \frac{G(r))^n}{3\rho_{\text{rel}}\Gamma(r)} \, \mathrm{d}r, \tag{4.8}$$

an expression which includes all microstructural and material properties, except for Q. The right-hand side of Equation 4.7 contains terms related to the dominant atomic diffusion process, which are independent of the material properties except Q. With rearrangement, the right-hand side of Equation 4.7 becomes

$$\Theta(t, u(t)) \equiv \int_0^t \frac{1}{u} \exp\left(-\frac{Q}{Ru}\right) \, \mathrm{d}\tau, \tag{4.9}$$

so that Equation 4.7 may finally be written as

$$\Psi(\rho_{\text{rel}}) = \Theta(t, u(t)). \tag{4.10}$$

Because Ψ incorporates both the microstructure scale $G(\rho_{\text{rel}})$ and the scaling parameter $\Gamma(\rho_{\text{rel}})$, it may be considered a characteristic that quantifies the effects of microstructural evolution on the sintering kinetics as densification occurs. The effects of starting particle size distribution, pore size distribution, and green structure on the sintering behavior are included in the MSC [74].

Alternative, generalized formulations of the MSC corresponding to several different expressions of the constitutive equations given in the literature have also been formulated [89], though for our purposes, we find the classical MSC to be sufficient. Expressions similar to $\Theta(t, u(t))$ appear in Arrhenius-type equations and laws that model the influence of temperature on reaction rates, including such a model in [90]. The practical use of the classical MSC in the context of its use in our coupled model is discussed in Chapter 9.

Chapter 5

Density and Temperature Dependence of Dielectric and Thermal Properties

A feature that several of the existing models of microwave sintering [28, 30, 32, 33] lack is a way of modelling the dependence of thermal and dielectric properties of the material being sintered on temperature of the sample, and currently, there do not exist models that account for dependence of these properties on both temperature and relative density. In this chapter, we discuss two ways of characterizing the dependence of material properties on density, and we give mixture models for dielectric properties, along with linear models for the thermal properties.

5.1 Models for Determining Dielectric Properties of Mixtures

In this section, we discuss theory and practical tests of several models for determining the complex permittivity and permeability of mixtures of dielectric media.

Lichtenecker's Logarithmic Mixture Formula

Lichtenecker's logarithmic mixture formula for dielectrics [91, 92] has been used since the 1930s to calculate effective complex permittivity for various mixtures of dielectric substances such as silica and ferrite powders [93], pavement mixtures [94], polyelectric sensors [95], and biological cells [96].

Lichtenecker and Rother [92] presented the formula in its logarithmic form, which can be used to compute the permittivity most efficiently as follows:

$$\varepsilon_{\text{eff}} = \prod_{n=1}^{N} \varepsilon_n^{v_n}, \tag{5.1}$$

where the n^{th} component of the N substances comprising the mixture is said to have effective permittivity ε_n and volume fraction v_n.

VFrac	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Neel's	2.00	2.89	4.17	6.01	8.67	12.51	18.06	26.06	37.60	54.26	78.3
Ours	2	2.89	4.17	6.01	8.67	12.51	18.06	26.06	37.60	54.26	78.3
Diff	0	0	0	0	0	0	0	0	0	0	0

Table 5.1: Comparison of results from [6] to those obtained by our MATLAB code implementing Lichtenecker's logarithmic mixture formula.

Some recent efforts have been focused on developing a physical foundation for the original Lichtenecker mixture formula, including a 1998 work by Zakri, Laurent, and Vauclin [97] that assumes a beta function distribution of inclusion shapes and makes use of the effective medium theory to construct a physical foundation for the formula.

In 2000, Goncharenko, Lozovski, and Venger [98] analyzed the formula in the context of spectral representation, attempting to characterize more precisely the range of anisotropic systems to which it is applicable. In particular, they showed using graphic representation for ε in the complex plane, along with Bergman's analytical representation based on resonance spectral density formalism [99], that Lichtenecker's formula has a wider area of applicability than Bruggeman's formula (Section 5.1) or the Maxwell Garnett equations (Section 5.1).

Recently, Simpkin [100] showed that the Lichtenecker formula may be of a still "more fundamental nature than previously regarded", by deriving the formula directly from Maxwell's equations and the principle of charge conservation under the assumption of a random spatial distribution of shapes and orientations of inclusions in a dielectric mixture. This characterization indicates that the closer the spatial distribution of components is to being random, the more accurate will be Lichtenecker's approximation of effective permittivity, which may account for the errors noted in previous work. Simpkin also showed how the Maxwell Garnett equations and Bruggeman's formula are derived from Lichtenecker's equations under certain conditions.

Neelakantaswamy *et al.* [6] compared results of their implementation of Lichtenecker's formula to computations performed by Boned and Peyrelasse [7] in the table reproduced in Table 5.1. These computations are supposed to represent the permittivity of a dielectric mixture of two components with permittivities $\varepsilon_1 = 78.3 + 0i$ and $\varepsilon_2 = 2.0 + 0i$ respectively, where the ellipsoidal inclusions have a varying ratio of major to minor axis (a/b), as shown.

For the case where $a/b = 1$ (the only case listed where the classical Lichtenecker formula can be applied), we compare the results to those obtained by applying our own MATLAB code, which can be found in Appendix G.1, with results of the comparison shown in Table 5.1.

Reynolds and Hough [101] charged in 1957 that the theoretical justification of the Lichtenecker formulas [92] was based on the incorrect assumption that the final dielectric constant of a mixture is independent of the method of preparation of the mixture. In particular, they cited two assumptions made in derivation of the formula which they assert can hold simultaneously neither in the case when "particles" of two materials are formed during two-stage mixing, nor in the case when particles are not formed. This assertion was reinforced by Dukhin and Shilov in 1974 [102], where they reformulated this statement as the result of erroneous consideration of a disperse system as simultaneously both ordered and chaotic.

In 1985, Neelakantaswamy, Turkman, and Sarkar [6] proposed corrections to the Lichtenecker mixture formula to address these concerns by considering ε_{eff} not as a logarithmic mean, but as a different form of weighted geometric mean that accounts for Wiener's upper and lower limits [103]. The final formulation they present for a mixture of two substances is as follows:

$$\varepsilon_{\text{eff}} = \begin{cases} \begin{cases} X(v)/2, & \varepsilon_1 > \varepsilon_2 \\ Y(v)/2, & \varepsilon_1 < \varepsilon_2 \end{cases}, & 0 \leq v \leq v_1 \\ \begin{cases} \frac{1}{2}\left[\frac{A(v_1)}{2C(v_1)} + \frac{B(v_2)}{2C(v_2)}\right] C(v)Z(v), & \varepsilon_1 > \varepsilon_2 \\ \frac{1}{2}\left[\frac{B(v_1)}{2C(v_1)} + \frac{A(v_2)}{2C(2)}\right] C(v)Z(v), & \varepsilon_1 < \varepsilon_2 \end{cases}, & v_1 \leq v \leq 2 \\ \begin{cases} Y(v)/2, & \varepsilon_1 > \varepsilon_2 \\ X(v)/2, & \varepsilon_1 < \varepsilon_2 \end{cases}, & v_2 \leq v \leq 1, \end{cases}$$

where $X(v) = Z(v) + 1/\varepsilon_L$, $Y(v) = Z(v) + \varepsilon_U$, $Z(v) = \varepsilon_U^n/\varepsilon_L^{n-1}$, $A(v) = 1 + 1/\varepsilon_U^n\varepsilon_L^n$, $B(v) = 1 + 1/\varepsilon_U^{n-1}\varepsilon_L^{n-1}$, and $C(v) = \sqrt{\varepsilon_L/\varepsilon_U}\varepsilon_1^v\varepsilon_2^{1-v}$. Here, ε_L and ε_U denote Wiener's upper and lower limits, respectively. This formulation is based on results presented by Kisdanasamy and Neelakantaswamy in 1984 [104] with estimates related to eccentricity of ellipsoidal inclusions taken from Coelho [105], and is valid only when

$$\frac{\varepsilon_1\varepsilon_2}{(\varepsilon_1 - \varepsilon_2)^2} - \frac{\varepsilon_1 + \varepsilon_2}{2(\varepsilon_1 - \varepsilon_2)\ln(\varepsilon_1/\varepsilon_2)} \geq -\frac{1}{4},$$

in order for Wiener's upper and lower limits both to be positive.

The dependence of the formula on eccentricity takes into account not only the volume fraction of the inclusions, but also their shape. In particular, unlike the original Lichtenecker formula, it is intended to be used not for randomly-shaped inclusions, but for ellipsoidal ones where the dimensions of the ellipsoids are known. Neelekantaswamy's correction can thus be thought of as, rather, an extension of Lichtenecker's formula.

Rayleigh Mixture Formula for Complex Permittivity

In 1892, Lord Rayleigh introduced a mixture formula describing the effective transport coefficients of mixtures where cylindrical or spherical particles were embedded within a matrix in a rectangular lattice structure [106]. The formula arose from the explicit solution of electric potentials inside and outside of the particles, which was obtained using the spectral method and imposing a conservation of flux law at the boundaries of the particles, as well as continuity of the potential. The formulation takes into consideration the models of Lorentz [107] and Lorenz [108] for the refractive index as a function of material density, and its predicted values lie within the Wiener limit [103]. Rayleigh's formula for these mixtures is

$$\frac{\varepsilon_{\text{eff}} - \varepsilon_1}{\varepsilon_{\text{eff}} + 2\varepsilon_1} = v\frac{\varepsilon_2 - \varepsilon_1}{2\varepsilon_1 + \varepsilon_2}, \tag{5.2}$$

where ε_{eff} represents the effective permittivity of the mixture, ε_1 and ε_2 represent, respectively, the permittivity of the first and second components of the mixture, and v represents the percent of the mixture occupied by the second component of the mixture ($v \in [0, 1]$).

Maxwell Garnett Mixing Rule for Complex Permittivity

First presented by J. C. Maxwell Garnett[1] in 1904 [110, 111], the Maxwell Garnett formulas were developed to determine the optical properties of a substance called gold ruby glass, which contains minute spherical particles of gold. As such, this model is considered suitable to describe mixtures involving metal particles, provided that those mixtures satisfy validity conditions discussed below. The formulation of the Maxwell Garnett mixing rule for a mixture of two materials is as follows:

$$\varepsilon_{\text{eff}} = \varepsilon_1 \left(\frac{\varepsilon_2(1 + 2v) - \varepsilon_1(2v - 2)}{\varepsilon_1(2 + v) + \varepsilon_2(1 - v)} \right), \tag{5.3}$$

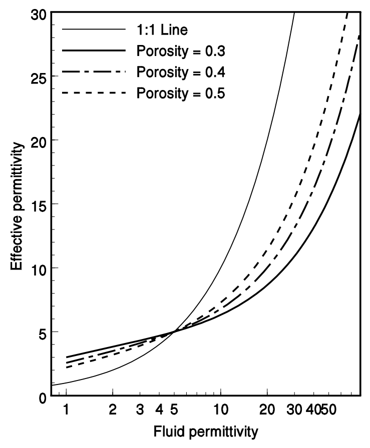
where v is the volume ratio of the embedded material, ε_2 is the permittivity of the embedded material, and ε_1 is the permittivity of the matrix material. Simpkin has shown that, under the condition that the value $2\alpha\varepsilon_2 - \varepsilon_1\varepsilon_2 + 2\varepsilon_1$ is small, the Maxwell Garnett equations may be derived as an approximation to Lichtenecker's formula [100].

In general, the model is valid for mixtures that have the following qualities [112]:

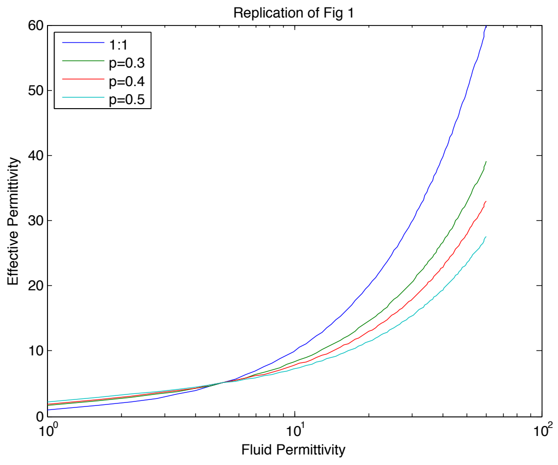
- the mixture is electrodynamically isotropic;
- the mixture is linear, that is, none of its constitutive parameters depends on the intensity of the electromagnetic field;
- the mixture is non-parametric, that is, its parameters do not change in time as a result of external forces;
- inclusions are separated by distances greater than their characteristic size;
- the characteristic size of inclusions is small compared to the wavelength in the effective medium;
- inclusions are arbitrary, randomly oriented ellipsoids;
- if there are conducting inclusions, their concentration should be lower than the percolation threshold.

Since its original formulation, the model has been used to calculate permittivity of various mixtures, including those of glass spheres, quartz sand grains and their mixtures [1], and snow [113]. The dependence of ε_{eff} of the mixture on the permittivity of its fluid matrix was computed in [1] using the Maxwell Garnett model, and is replicated in Figure 5.1a, alongside the version in Figure 5.1b created using our own implementation of the Maxwell Garnett formula in MATLAB (code can be found in Appendix G.3).

[1]James Clerk Maxwell Garnett was the son of William Garnett, James Clerk Maxwell's scientific demonstrator at the Cavendish Laboratory[109].



(a) Figure from [1], showing the effective permittivity of a mixture of quartz sand grains.



(b) Replication using in-house MATLAB code.

Figure 5.1: Comparison of our implementation of the Maxwell Garnett mixture formula, with the results obtained in [1].

In 1984, Thériault and Boivin [114] extended the Maxwell Garnett theory to include the shape factor and size of the metal particles suspended in a dielectric matrix, and observed good agreement with reality for volume fractions ranging from 0 to 0.12. Their formula for ε_{eff} is given implicitly as follows:

$$\frac{\varepsilon_{\text{eff}} - \varepsilon_m}{f\varepsilon_{\text{eff}} + (1-f)\varepsilon_m} = v\frac{\varepsilon_i - \varepsilon_m}{f\varepsilon_i + (1-f)\varepsilon_m},$$

where ε_m is the permittivity of the matrix; ε_i is the permittivity of the inclusion; v is the volume fraction of the inclusion; and f is the shape factor of the metal particles, given below as a function of ε_{eff}.

$$f = \varepsilon_m \frac{v\varepsilon_i(a) - \varepsilon_{\text{eff}} + (1-v)\varepsilon_m}{(1-v)(\varepsilon_{\text{eff}} - \varepsilon_m)(\varepsilon_i(a) - \varepsilon_m)},$$

where a is the radius of the inclusions.

In 2006, Koledintseva *et al.* [115] applied a Maxwell Garnett model to engineer microwave-absorbing materials containing an arbitrary number of different types of carbon particles, using the following formula:

$$\varepsilon_{\text{eff}} = \varepsilon_m + \frac{\frac{1}{3}\sum_{i=1}^{n} v_i(\varepsilon_i - \varepsilon_m)\sum_{k=1}^{3}\frac{\varepsilon_m}{\varepsilon_m + N_{ik}(\varepsilon_i - \varepsilon_m)}}{1 - \frac{1}{3}\sum_{i=1}^{n} v_i(\varepsilon_i - \varepsilon_m)\sum_{k=1}^{3}\frac{N_{ik}}{\varepsilon_m + N_{ik}(\varepsilon_i - \varepsilon_m)}},$$

where ε_m is the relative permittivity of the matrix dielectric; ε_i is the permittivity of the i^{th} type of inclusion; v_i are the volume fractions occupied by the i^{th} type of inclusion; N_{ik} are the depolarization

VFrac	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
MG	2	2.61	3.37	4.31	5.54	7.18	9.52	13.1	19.2	32.2	78.3
Corr	2	2.61	3.37	4.31	5.54	7.18	9.52	13.1	19.2	32.2	78.3
Diff	0	0	0	0	0	0	0	0	0	0	0

Table 5.2: Koledintseva's extension compared with the original Maxwell Garnett equation for the case of a mixture of only two components, where the ratio of the major to minor axis length of the ellipsoidal inclusions is 1 (which represents spherical inclusions).

factors of the i^{th} type of inclusion; and the index $k = 1, 2, 3$ corresponds to x, y, and z in Cartesian coordinates.

Validation of Koledintseva's extension against the original Maxwell Garnett equation can be carried out easily by assuming a mixture of only two components and setting the ratio of major to minor axis length of the ellipsoidal inclusions to be 1 (this represents spherical inclusions). In that case, the permittivity calculated by the extension should be the same as that calculated by the original Maxwell Garnett formula. Running the two corresponding MATLAB codes (found in Appendices G.3 and G.4 respectively) yields the results shown in Table 5.2.

Bruggeman's Models

Bruggeman's symmetric mixture formula for a two-part mixture, introduced in 1935 [116], was stated by Reynolds and Hough [101] to be a "fairly obvious" extension of Lichtenecker's formulas to the more complicated cases of mixtures that could be anisotropic or could have random spatial distributions. Simpkin [100] recently showed that Bruggeman's symmetric mixture formula results from the classical Lichtenecker formula for a mixture of two components, under the assumption that the Clausius-Mossotti factors F_1 and F_2,

$$F_1 = \left(\frac{\varepsilon_1 - \varepsilon_{\text{eff}}}{\varepsilon_1 + 2\varepsilon_{\text{eff}}}\right) \quad F_2 = \left(\frac{\varepsilon_2 - \varepsilon_{\text{eff}}}{\varepsilon_2 + 2\varepsilon_{\text{eff}}}\right),$$

are small enough in magnitude for the dependence on F_1 and F_2 to be taken as linear. This, as Simpkin states, is equivalent to considering only first-order interactions between each component of the mixture embedded in a homogeneous effective medium. Bruggeman's model and its theoretical basis are explained in English in [117].

Bruggeman's symmetric mixture formula for a two-part mixture is as follows.

$$v\left(\frac{\varepsilon_1 - \varepsilon_{\text{eff}}}{\varepsilon_1 + 2\varepsilon_{\text{eff}}}\right) + (1 - v)\left(\frac{\varepsilon_2 - \varepsilon_{\text{eff}}}{\varepsilon_2 + 2\varepsilon_{\text{eff}}}\right) = 0, \tag{5.4}$$

where v is the volume fraction of the second component, the components have permittivities ε_1 and ε_2 respectively, and the mixture has permittivity ε_{eff}.

5.2 Implementation of Mixture Models for Density- and Temperature-Dependent Material Properties

In order to determine the effective relative dielectric constant $\varepsilon'_{\text{eff,rel}}$, electrical conductivity σ_{eff}, or relative permeability $\mu_{\text{eff,rel}}$ as functions of temperature and density, we assume that experimental data obtained during a trial processing of the material consists of a set of measurements of temperature u, density ρ_{rel}, the effective relative dielectric constant $\varepsilon'_{\text{eff,rel}}$, electrical conductivity σ_{eff}, and relative permeability $\mu_{\text{eff,rel}}$.

To use the mixture models discussed above, we treat the porous dielectric medium as a two-component mixture of the bulk solid and air. In this case, the relative density ρ_{rel} of the granular sample is equal to the volume fraction of media. We first "invert" the mixture formula in order to determine a function relating the desired property of the bulk material to its temperature—that is, we find expressions for $\varepsilon'_{\text{rel,bulk}}(u)$, $\sigma_{\text{bulk}}(u)$, and $\mu_{\text{rel,bulk}}(u)$ by interpolating the measured data, and we then use the "forward" mixture formula to compute the desired property of the porous medium using the relative density and the desired bulk property value determined using the inversion.

This process is repeated for each of $\varepsilon'_{\text{rel}}$, μ_{rel}, and σ_{rel} individually, to construct each as a function of ρ_{rel} and u.

Implementation of Lichtenecker's Formula

When we treat the particulate sample as a mixture of bulk solid and air, assuming that air is the first component of the mixture, we obtain $\varepsilon_1 \approx 1$, and $v = \rho_{\text{rel}}$, and so Lichtenecker's formula in Equation 5.1 reveals the two equations

$$\varepsilon'_{\text{eff,bulk}}(u) = \varepsilon'_{\text{eff,rel}}(u)^{1/\rho_{\text{rel}}(u)} \tag{5.5}$$

and

$$\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u) = \varepsilon'_{\text{eff,bulk}}(u)^{\rho_{\text{rel}}}. \tag{5.6}$$

In order to determine $\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u)$ given the measured data on $\varepsilon'_{\text{eff}}$, temperature u, and density ρ_{rel}, we first compute the discrete $\varepsilon'_{\text{bulk}}$ values using Equation 5.5 for each of the measured data points, and we then perform interpolation to reveal the function $\varepsilon'_{\text{eff,bulk}}(u)$.

Once this function is known, we may find $\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u)$ using Equation 5.6. This procedure is also assumed to be an accurate way of determining the relative effective magnetic permeability $\mu_{\text{rel,eff}}$ of the porous medium during sintering.

Implementation of Rayleigh's Formula

As above, we treat the particulate sample as a mixture of bulk solid and air, assuming that air is the first component of the mixture, and obtain $\varepsilon_1 \approx 1$, and $v = \rho_{\text{rel}}$. Rayleigh's formula in Equation 5.2

reveals the two equations

$$\varepsilon'_{\text{eff,bulk}}(u) = \frac{1 + \frac{2}{\rho_{\text{rel}}(u)}\left(\frac{\varepsilon'_{\text{eff,rel}}(u)-1}{\varepsilon'_{\text{eff,rel}}(u)+1}\right)}{1 - \frac{1}{\rho_{\text{rel}}(u)}\left(\frac{\varepsilon'_{\text{eff,rel}}(u)-1}{\varepsilon'_{\text{eff,rel}}(u)+1}\right)} \tag{5.7}$$

and

$$\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u) = \frac{\varepsilon'_{\text{eff,bulk}}(u)(2\rho_{\text{rel}}+1) - (2\rho_{\text{rel}}-2)}{\varepsilon'_{\text{eff,bulk}}(u)(1-\rho_{\text{rel}}) + (\rho_{\text{rel}}-2)}. \tag{5.8}$$

In order to determine $\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u)$ given the measured data on $\varepsilon'_{\text{eff}}$, temperature u, and density ρ_{rel}, we first compute the discrete $\varepsilon'_{\text{bulk}}$ values using Equation 5.7 for each of the measured data points, and we then perform interpolation to reveal the function $\varepsilon'_{\text{eff,bulk}}(u)$.

Once this function is known, we may find $\varepsilon'_{\text{eff,rel}}(u, \rho_{\text{rel}})$ using Equation 5.8. This procedure is also assumed to be an accurate way of determining the relative effective magnetic permeability $\mu_{\text{rel,eff}}$ of the porous medium during sintering.

Implementation of the Maxwell Garnett Formula

As above, we treat the particulate sample as a mixture of bulk solid and air, assuming that air is the first component of the mixture, and obtain $\varepsilon_1 \approx 1$, and $v = \rho_{\text{rel}}$. Maxwell Garnett's formula in Equation 5.3 reveals the two equations

$$\varepsilon'_{\text{eff,bulk}}(u) = \frac{(1+\rho_{\text{rel}}(u))(\varepsilon'_{\text{eff,rel}}(u)-1)}{2\rho_{\text{rel}}(u) - (1-\rho_{\text{rel}}(u))(\varepsilon'_{\text{eff,rel}}(u)-1)} \tag{5.9}$$

and

$$\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u) = \frac{\varepsilon'_{\text{eff,bulk}}(u) - 1}{\varepsilon'_{\text{eff,bulk}} + 1 - \rho_{\text{rel}}\varepsilon'_{\text{eff,bulk}}(u) + \rho_{\text{rel}}}. \tag{5.10}$$

In order to determine $\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}}, u)$ given the measured data on $\varepsilon'_{\text{eff}}$, temperature u, and density ρ_{rel}, we first compute the discrete $\varepsilon'_{\text{bulk}}$ values using Equation 5.9 for each of the measured data points, and we then perform interpolation to reveal the function $\varepsilon'_{\text{eff,bulk}}(u)$.

Once this function is known, we may find $\varepsilon'_{\text{eff,rel}}(u, \rho_{\text{rel}})$ using Equation 5.10. This procedure is also assumed to be an accurate way of determining the relative effective magnetic permeability $\mu_{\text{rel,eff}}$ of the porous medium during sintering.

Implementation of Bruggeman's Formula

As above, we treat the particulate sample as a mixture of bulk solid and air, assuming that air is the first component of the mixture, and obtain $\varepsilon_1 \approx 1$, and $v = \rho_{\text{rel}}$. Bruggeman's formula in Equation 5.4 reveals the two equations

$$\varepsilon'_{\text{eff,bulk}}(u) = \frac{(1-3\rho_{\text{rel}}(u))\varepsilon'_{\text{eff,rel}}(u) + 2(\varepsilon'_{\text{rel,eff}}(u))^2}{1 + (2-3\rho_{\text{rel}}(u))\varepsilon'_{\text{rel,eff}}(u)}, \tag{5.11}$$

and

$$\rho_{\text{rel}}\left(\frac{1-\varepsilon'_{\text{eff,rel}}}{1+2\varepsilon'_{\text{eff,rel}}}\right)+(1-\rho_{\text{rel}})\left(\frac{\varepsilon'_{\text{eff,bulk}}(u)-\varepsilon'_{\text{eff,rel}}}{\varepsilon'_{\text{eff,bulk}}(u)+2\varepsilon'_{\text{eff,rel}}}\right)=0. \tag{5.12}$$

In order to determine $\varepsilon'_{\text{eff,rel}}(\rho_{\text{rel}},u)$ given the measured data on $\varepsilon'_{\text{eff}}$, temperature u, and density ρ_{rel}, we first compute the discrete $\varepsilon'_{\text{bulk}}$ values using Equation 5.11 for each of the measured data points, and we then perform interpolation to reveal the function $\varepsilon'_{\text{bulk}}(u)$.

Once this function is known, we may find $\varepsilon'_{\text{eff,rel}}(u,\rho_{\text{rel}})$ using Equation 5.12, together with either Newton's method, or directly using the quadratic formula and restricting consideration to the positive branch:

$$\varepsilon'_{\text{eff,rel}}(u,\rho_{\text{rel}})=\frac{1}{2}\left(1+3\rho_{\text{rel}}(1-3\varepsilon'_{\text{eff,bulk}})\right)\mp\frac{1}{2}\sqrt{4\varepsilon'_{\text{eff,bulk}}+\left(1+3\rho_{\text{rel}}(1-3\varepsilon'_{\text{eff,bulk}})\right)^2}.$$

This procedure is also assumed to be an accurate way of determining the relative effective magnetic permeability $\mu_{\text{rel,eff}}$ of the porous medium during sintering.

5.3 Porosity Models for Thermal Properties

We refer here to the density ρ, the specific heat capacity c_p, and the thermal conductivity k as "thermal properties" of matter, because these properties all appear in the heat equation (Equations 3.24 and 3.25). We discuss how to determine the density as a function of temperature and rate of heating using the Master Sintering Curve method in Section 4.3. In the remainder of this chapter, we discuss how to handle the density dependence of the other thermal properties of the material undergoing sintering, using the technique described in [118].

The *porosity* of matter, denoted here as p, is defined as the volume of pores, relative to the total volume of the porous material; in terms of the relative density ρ_{rel} expressed as a percentage between 0 and 1, we therefore have $p = 1-\rho_{\text{rel}}$.

At any given temperature u, for a powder material undergoing sintering, the specific heat capacity and the thermal conductivity change with porosity. If the specific heat capacity of air within the pores is neglected, then that of the porous material scales linearly as

$$c_p(\rho_{\text{rel}},u)=(1-p)c_{p,\text{bulk}}(u)=\rho_{\text{rel}}c_{p,\text{bulk}}(u), \tag{5.13}$$

where $c_{p,\text{bulk}}(u)$ denotes the specific heat capacity of the bulk material at the temperature u. We discuss in Section 10.2 how to use this model in practice.

The thermal conductivity depends on the material's microstructure, and in general, may be determined using the same mixture formulas that are used in approximating dielectric properties [118]. However, if we neglect the thermal conductivity of air, then that of the porous material may be approximated as

$$k(\rho_{\text{rel}},u)=\left(1-\frac{3}{2}p\right)k_{\text{bulk}}(u)=\left(\frac{3}{2}\rho_{\text{rel}}-\frac{1}{2}\right)k_{\text{bulk}}(u), \tag{5.14}$$

where, as before, $k_{\text{bulk}}(u)$ represents the thermal conductivity of the bulk material at temperature u. This expression is not valid for highly porous materials in which heat transfer through air is significant, but we consider materials that start at a "green" density above 50% of bulk density, and so this model suffices.

In order to use the model in Equations 5.13 and 5.14 with experimental data that consist of measurements of temperature u, relative density ρ_{rel}, and the specific heat capacity c_p and thermal conductivity k, we use alternative expressions of Equations 5.13 and 5.14 to compute the bulk properties as functions of temperature:

$$c_{p,\text{bulk}}(u) = \frac{c_p(u)}{\rho_{\text{rel}}(u)}, \qquad \text{and} \qquad k_{\text{bulk}}(u) = \frac{k(u)}{\left(\frac{3}{2}\rho_{\text{rel}}(u) - \frac{1}{2}\right)},$$

which are substituted into Equations 5.13 and 5.14 to obtain expressions for the properties that depend on both temperature and relative density. The interpolation functions for finding the bulk properties are computed using third-order b-spline interpolants of the values of ρ_{rel} with the variable u.

5.4 Θ-Based Models for Both Dielectric and Thermal Properties

An alternative characterization of the evolution of dielectric and thermal properties by assuming their dependence on the work of sintering Θ has only a phenomenological basis; however, it provides accurate results, as shown in Section 11.2. This method assumes that if the density may be characterized as a function of Θ, the *work of sintering* parameter defined in Section 4.3, and if Θ is, itself, a function of the temperature and its evolution, then the dielectric and thermal material properties, which are assumed to depend on density and temperature, may also be characterized as functions of Θ. That is, if discussing the effective dielectric constant of the mixture, for example,

$$\varepsilon'_{\text{eff}} = \varepsilon'_{\text{eff}}(\rho_{\text{rel}}, u) = \varepsilon'_{\text{eff}}(\rho_{\text{rel}}(\Theta), u) = \varepsilon'_{\text{eff}}(\rho_{\text{rel}}(\Theta(u), u)) = \varepsilon'_{\text{eff}}(\Theta).$$

The input data needed to produce functions for the dielectric and thermal properties in this case are measurements of those properties throughout the full temperature range of a processing experiment, with reference times and temperatures also recorded. Once these are known, and once the activation energy Q has been computed, the Θ-values corresponding to the experiment in which the property of interest was measured is computed, and these are used as the independent variable in a third-degree b-spline interpolation to determine the function that yields the dielectric or thermal property of interest.

Chapter 6

Mixture Formulas for Determining the Effective Complex Permittivity of Metal Powders

As long as the sintering progress may still be characterized by densification, and as long as the Master Sintering Curve discussed in Section 4.3 provides a valid characterization of densification, our multiphysics simulation routine remains applicable to various materials, including metal powders, and this necessitates the computation of effective dielectric properties of those powders. This chapter contains describes a series of tests of the various mixture models for effective dielectric properties metal powders, and has been previously published by the author [119].

While reports on measurements of the dielectric constant and loss factors of dielectric materials may be found in literature, data on the effective complex permittivity of metal powders can often not be found so readily, and even when they can be, conflicting values (sometimes up to orders of magnitude) often exist [120]. With this in mind, we focus on development of simple and practical computational routines based on classical and contemporary models for those determining complex permittivity of composites or mixtures whose applications may be extended to metal powders. We briefly review the most notable models, present them in closed form, examine the ranges of validity of their input parameters, demonstrate their computational implementations and discuss their applicability with reference to original measurements of effective complex permittivity of a mixture of titanium and stearic acid, in addition to previously reported [121] measurements of the loss factor of a mixture of tungsten and Teflon®. We discuss the reasons for discrepancies between the results obtained from the models and the experimental measurements. Under the identified limitations for their use, the classical Bruggeman mixture model [116] and the core-shell mixture model proposed by Buchelnikov *et al.* [122–124] are shown to produce the most accurate results for mixtures of metal powders in which the volume fraction of bulk metal is below the percolation threshold.

When Lichtenecker's or Maxwell Garnett's model is used for estimating $\varepsilon_{\text{eff}} := \varepsilon' - j\varepsilon''$ of a mixture in which the i^{th} component is a metal powder, then ε_i should represent the effective property of the metal powder in air, as bulk metals do not behave like dielectrics and thus do not have effective dielectric properties.

6.1 Bruggeman's Model with Compacted Metal Powders

Ignatenko *et al.* [125] give Bruggeman's formula differently for compacted metal powders of the type shown in Figure 6.1, asserting

$$v\left(\frac{\varepsilon_p - \varepsilon_{\text{eff}}}{\varepsilon_p + 2\varepsilon_{\text{eff}}}\right) + (1-v)\left(\frac{\varepsilon_g - \varepsilon_{\text{eff}}}{\varepsilon_g + 2\varepsilon_{\text{eff}}}\right) = 0,$$

where v is the volume fraction of the particles, ε_g is the permittivity of gas in pores, and $\varepsilon_p = \varepsilon_2 F_2$, with the coefficient F_2 defined as:

$$F_2 = 2\frac{1-(r_1/r_2)^3 F_1}{1+2(r_1/r_2)^3 F_1}, \quad F_1 = 2\frac{1-(\varepsilon_1/\varepsilon_2)^3 F_0}{1+2(\varepsilon_1/\varepsilon_2)^3 F_0}, \quad F_0 = 2\frac{-y\cos y + \sin y}{y\cos y - \sin y + y^2 \sin y},$$

where $r_{1,2}$ are the radii of the core and shell of the particle and $\varepsilon_{1,2}$ are the permittivities of the core and shell respectively. The argument of the factor F_0 is $y = k_1 r_1$, where $k_{1,2} = \omega(\varepsilon_{1,2}\mu_{1,2})^{1/2}$, $\omega = 2\pi f$, and f is the frequency of the electromagnetic wave irradiating the sample. The skin depth of highly conductive non-magnetic core material can be accounted for by setting $y = (1+i)R_1/\delta$, where δ is the skin depth.

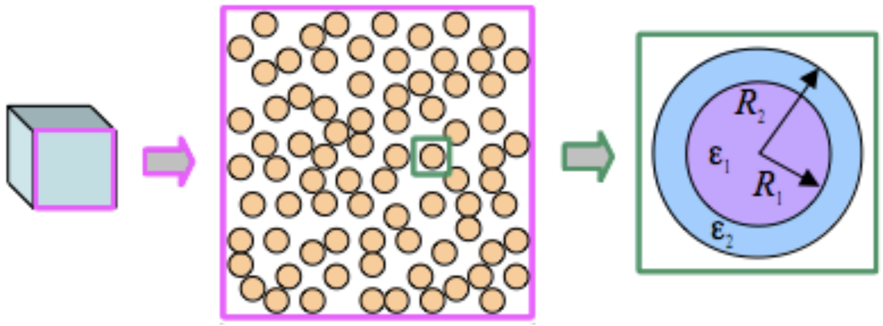


Figure 6.1: Core-shell concept of metal particles.

Numerical Verification of Bruggeman's Model

Kärkkäinen, Sihvola, and Nikoskinen carried out in 2000 a verification of Bruggeman's formula in random dielectric materials, with specific tests run for raisin pudding and swiss cheese [126]. Their verification was based on calculations of the effective permittivity of the mixtures using FDTD simulations of the sample in a TEM waveguide, and was carried out with an eye toward the validation of not only Bruggeman's formula, but also the Wiener [103] and Hashin-Shtrikman [127] limits.

VolFrac	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
Boned [7]	2	3.02	4.25	5.00	7.50	11.00	17.01	26.96	40.52	58.3	78.3
Licht	2	2.89	4.17	6.01	8.67	12.51	18.06	26.06	37.60	54.26	78.3
MG	2	7.32	13.02	19.12	25.68	32.75	40.39	48.67	57.68	67.52	78.3
Brugg	2	2.75	4.23	7.55	14.2	23.4	33.8	44.7	55.8	67	78.3

Table 6.1: Effective dielectric constants of a two-medium mixture, comparing results from [7] with those obtained using our implementations of the Lichtenecker, Maxwell Garnett, and Bruggeman formulas.

Comparison of Bruggeman's Model to Lichtenecker and Maxwell Garnett

Table 6.1 lists the effective permittivities of a mixture of two materials with, respectively, $\varepsilon_1 = 78.3$ and $\varepsilon_2 = 2.0$. The effective permittivity has been computed using three methods, and the first row contains the values computed by Boned and Peyrelasse [7].

6.2 Buchelnikov's Model

Like the model presented by Ignatenko [125], the model presented by Buchelnikov *et al.* [124], [123] considers spherical core-shell particles randomly distributed in the effective medium. They determine a relationship between the effective permittivity of the mixture and the radii of the spherical inclusions, the permittivities of the core and shell of the inclusions, and the value of the external electric field E_0:

$$v\zeta \frac{\varepsilon_2 \left[3\varepsilon_1 + (\zeta - 1)(\varepsilon_1 + 2\varepsilon_2)\right] - \varepsilon_{\text{eff}} \left[3\varepsilon_2 + (\zeta - 1)(\varepsilon_1 + 2\varepsilon_2)\right]}{2a\varepsilon_{\text{eff}} + b\varepsilon_2} + (1 - v\zeta)\frac{\varepsilon_g - \varepsilon_{\text{eff}}}{\varepsilon_g + \varepsilon_{\text{eff}}} = 0, \tag{6.1}$$

where v is the volume fraction of the metal inclusions, ε_g is the permittivity of the gas or vacuum, $\varepsilon_{1,2}$ are the permittivities of the metallic core and shell respectively, and the expressions for ζ, a, and b are:

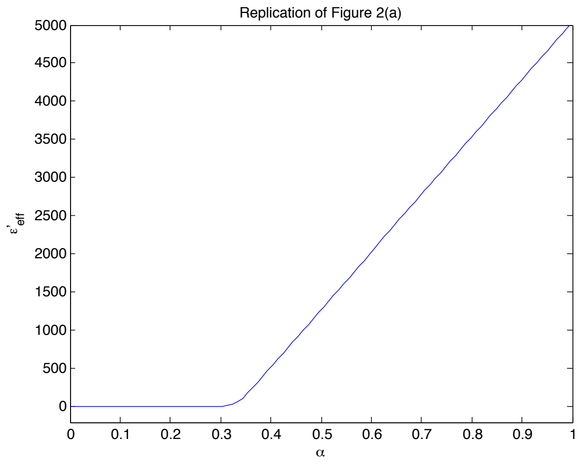
$$\zeta = (R_2/R_1)^3 = (1 + l)^3, \quad l = (R_2 - R_1)/R_1,$$
$$a = (\zeta - 1)\varepsilon_1 + (2\zeta + 1)\varepsilon_2, \quad b = (2 + \zeta)\varepsilon_1 + 2(\zeta - 1)\varepsilon_2,$$

where $R_{1,2}$ are the radii of the metallic core and shell respectively.

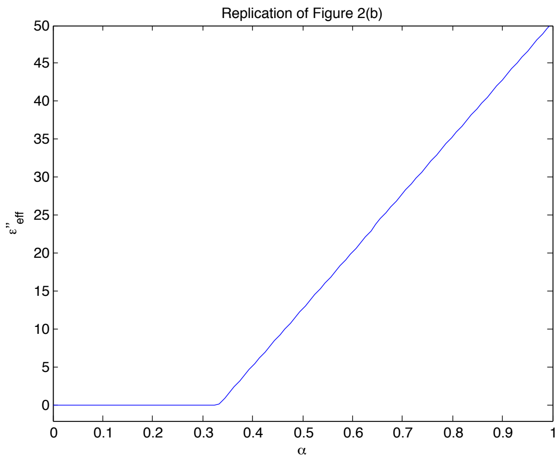
In [123] it is observed that in the limiting case $R_1 \to 0$, this reduces to the formula:

$$v_1 \frac{\varepsilon_2 - \varepsilon_{\text{eff}}}{\varepsilon_2 + 2\varepsilon_{\text{eff}}} + (1 - v_1)\frac{\varepsilon_g - \varepsilon_{\text{eff}}}{\varepsilon_g + 2\varepsilon_{\text{eff}}} = 0,$$

where v_1 is the volume fraction of the dielectric consisting of the shell material. They also discuss the parallel case where $R_2 \to R_1$, where they obtain the formula for determining the permittivity of



(a) Dielectric constant as a function of the volume fraction of iron oxide powder, produced using our MATLAB implementation of Buchelnikov's formula in Equation 6.1.



(b) Loss factor as a function of the volume fraction of iron oxide powder, produced using our MATLAB implementation of Buchelnikov's formula in Equation 6.1.

the mixture of pure metallic and dielectric spherical particles:

$$v\frac{\varepsilon_1 - \varepsilon_{\text{eff}}}{\varepsilon_1 + 2\varepsilon_{\text{eff}}} + (1-v)\frac{\varepsilon_g - \varepsilon_{\text{eff}}}{\varepsilon_g + 2\varepsilon_{\text{eff}}} = 0.$$

Note that these formulae are exactly the Bruggeman equations; in this respect, Buchelnikov's model can be considered an extension of the Bruggeman formula to the case where spherical inclusions are themselves comprised of a core and shell.

Verification of Our Implementation

Buchelnikov *et al.* presented in [124] graphs showing the dependence of ε_{eff} on the volume fraction of core-shell particles of iron powder in oxide shells. This figure is replicated in Figure **??**, which was produced using our implementation of Buchelnikov's formula in the MATLAB code shown in Appendix G.6.

Alternative Models

There are also other, different, mixing formulas to be considered. Sihvola [128] presented in 1989 a method of using one equation with a dimensionless parameter ν to characterize the results of several different mixing formulas, including the Rayleigh formula [129] (Section 5.1), the Maxwell Garnett Formula (Section 5.1), the Bruggeman formula (Section 5.1) the Böttcher mixing formula [130], the Polder-van Santen formula [131], and the QCA (quasi-crystalline approximation) formula [131].

The Looyenga model [113, 132] and other so-called "power-law models" can also be derived from this equation.

The equation is as follows:

$$\frac{\varepsilon_{\text{eff}} - \varepsilon_1}{\varepsilon_{\text{eff}} + \varepsilon_1 + \nu(\varepsilon_{\text{eff}} - \varepsilon_1)} = v \frac{\varepsilon_2 - \varepsilon_1}{\varepsilon_2 + \varepsilon_1 + \nu(\varepsilon_{\text{eff}} - \varepsilon_1)} = 0,$$

where v is the volume fraction of the second component, the components have permittivities ε_1 and ε_2 respectively, and the mixture has permittivity ε_{eff}. Note that $\nu = 0$ gives the Maxwell Garnett equation, and $\nu = 1$ gives Bruggeman's formula. We should investigate this equation and the various models it is capable of describing.

Sheen *et al.* [133] present six different "mixture rules", corresponding to ceramic powders with alumina inclusions that are spherical, cylindrical, or rod-, lamella-, or disk-shaped. Their six rules correspond to various combinations of known mixture formulas, and should be investigated as such.

Also useful to us will be a consideration of all the above models in the context of the well-known Wiener [103] and Hashin-Shtrikman bounds [127], the latter of which is used for statistically homogeneous and isotropic mixtures.

6.3 Experimental Results

We describe the results of two attempts to evaluate the effective complex permittivity of mixtures involving metal powders. The results of these experiments are used to test some of the models described in this chapter.

Tungsten-Teflon® mixture

In [121], Zimmerman *et al.*dealt with samples of powders made with varying volume fractions of tungsten in Teflon® powder. The mixtures were formed into cylindrical pellets of diameter 41 mm and height 64 mm, with varying particle sizes, and the authors determined the effective complex permittivity and permeability of each sample using cavity perturbation techniques.

Titanium-stearic acid mixture

In our experiment [119], we mechanically mixed gas atomized spherical titanium particles with radius 25 μm (obtained from Pyrogenesis Inc., Canada) with stearic acid (Sigma-Aldrich Co., 95%) in various volume fractions. These mixtures were compacted uniaxially into cylindrical pellets of diameter 10 mm and height 20 mm.

For this and the boron nitride/graphite mixture, the cavity perturbation approach [134–136] involves the transverse magnetic mode TM_{010} of a cylindrical resonator with two coaxial holes. The cylindrical samples, longer than the cavity height, are inserted coaxially and maintained in the center of the cavity. The variations of resonant frequency of the cavity as well as its quality factor due to the presence of the sample are determined with an HP 8720D vector network analyzer. As certain commonly used numerical methods can present large uncertainties for high permittivity, the complex permittivity of the sample is evaluated instead using a calibration method based on standard samples. These experimentally obtained results are used in testing the core-shell models described in this chapter.

Hexagonal boron nitride/graphite mixture

As also reported in [119], graphite particles (flakes, 4 μm, TIMCAL Ltd.) and hexagonal boron nitride powder (10 μm, Kennametal Sintec Keramic GmbH, Germany) were mixed in an agate mortar in various volume fractions and pressed uniaxially to form cylindrical samples of 10 mm diameter and 20 mm in height. We also used the cavity perturbation techniques to determine the effective complex permittivity of these samples.

6.4 Experimental results compared with modeling results

These above experimentally obtained results are used in testing the mixture and core-shell models described in this chapter.

Mixture Models

Taking the effective complex permittivity of tungsten to be constant at $30 + 8j$ [137], and the complex permittivity of Teflon® to be $2.29 + 0.03j$ [138], we use the Lichtenecker, Maxwell Garnett, and Bruggeman models to reconstruct the dielectric constant and the loss factor of the mixture for different values of the volume fraction v of tungsten in Teflon®. The corresponding curves are shown in Figure 6.3a. The mixture exhibits distinct percolation behaviors, characterized by a peak in $\tan\delta := \varepsilon'/\varepsilon''$ at volume fractions that depend on the average particle size. The location of this peak is shown in Figure 6.3b by the appropriate values of $\tan\delta$ for particles of diameter 2.3 μm, alongside the same $\tan\delta$ values predicted by the mixture models.

Assuming that ε' and ε'' are smooth functions of volume fraction, a peak in $\tan\delta$ may occur only at those critical volume fractions v_p for which the first derivative of $\tan\delta$ is zero; that is,

$$\varepsilon''(v_p)\frac{\mathrm{d}\varepsilon'(v_p)}{\mathrm{d}v} = \varepsilon'(v_p)\frac{\mathrm{d}\varepsilon''(v_p)}{\mathrm{d}v}.$$

Using the expression for complex permittivity predicted by Lichtenecker's model, the only time this situation occurs is independent of volume fraction, when $\varepsilon_1'\varepsilon_2' = \varepsilon_1''\varepsilon_2''$. In this case, all subsequent derivatives of $\tan\delta$ are also zero—so no peak occurs for any mixture whose permittivity is found using the Lichtenecker model. Using the expression predicted by the Maxwell Garnett model, no zeros of the first derivative of $\tan\delta$ exist for any mixture, and so neither the Maxwell Garnett nor the Lichtenecker model can accurately predict effective properties of mixtures of metal powders at volume fractions beyond the percolation threshold. The Bruggeman model, applied to the mixture of tungsten and Teflon®, also did not predict any peaks.

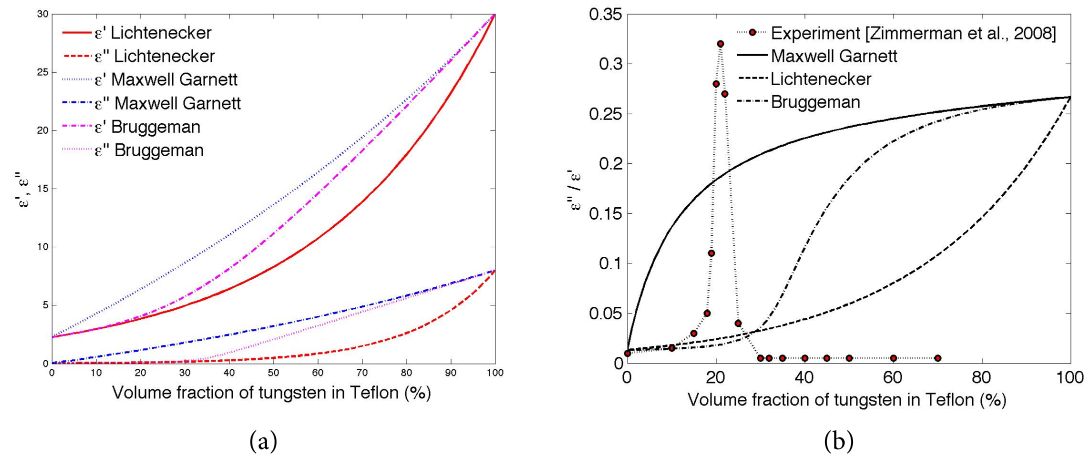


Figure 6.3: Complex permittivity (a) and tan δ (b) of tungsten/Teflon® mixture—models and experiment.

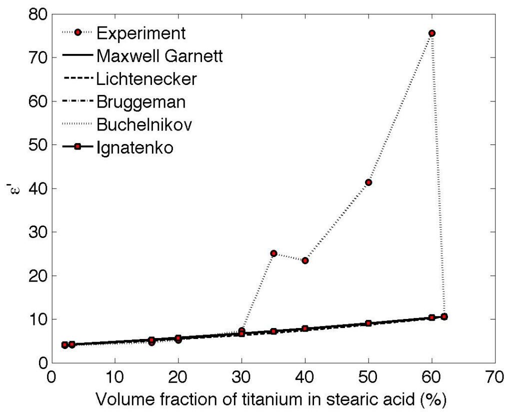
Core-Shell Models

The effective complex permittivity of core-shell titanium particles in a stearic acid matrix was computed using Buchelnikov's model, and the complex permittivity of the titanium and stearic acid mixture, ignoring the presence of the titanium oxide layer, was computed using various mixture models, with the results shown in Figure 6.4. Values used for the complex permittivity of titanium and the complex permittivity of stearic acid were taken directly from the experiment, and they were $10.55 + 1.0j$ and $3.95 + 0.006j$, respectively. The permittivity of the oxide layer was picked, in accordance with [139], as $114 + 0.003j$. The conductivity of titanium was taken to be 4.2×10^9 S/m, in accordance with [140].

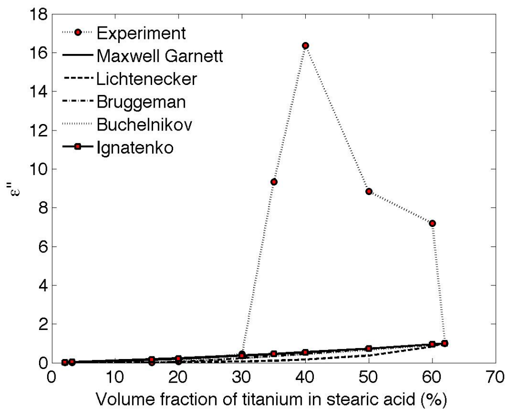
Since the radius of the oxide layer on the titanium particles is a necessary input to Buchelnikov's formula but is not known for the titanium particles we study, this parameter was chosen through golden selection search and parabolic interpolation to be the one which produced the permittivity curve closest to the experimentally obtained data.

In our experiments, the measured material properties also exhibit distinct percolation behaviors, where both ε' and ε'' of the mixture exceed the values of the corresponding parameters for pure stearic or for tapped titanium powder. However, it is seen that neither the core-shell nor any of the conventional mixture models is capable of accurately predicting permittivity at volume fractions beyond the percolation threshold of the mixture. Yet, it should be noted that before the percolation threshold, all of the curves obtained are a good fit to the experimental data obtained. The minimum error taken using only the first five data points (that is, those before the percolation threshold) is 0.441 (using Buchelnikov's model) for the ε' curve, and 0.101 (using Bruggeman's model) for the ε'' curve.

Our experiment was the source of the data on the effective complex permittivity of graphite



(a)



(b)

Figure 6.4: Real (a) and imaginary (b) parts of effective complex permittivity of titanium/stearic acid mixture—models and experimental measurements.

(which was found to be $25.5 + 0.15j$) and of boron nitride ($3.05 + 0.006j$). The conductivity of bulk boron nitride was estimated at 1×10^9 S/m, as shown in the graphs of [141].

Chapter 7

Numerical and Analytical Techniques for Solving the Electromagnetic Problem

As we saw in Chapter 2, the electromagnetic problem of microwaves within a waveguide may be framed as a wave equation problem, or as a Helmholtz equation problem. In this chapter, we describe several numerical and analytical techniques for solving each of those two problems in one and two dimensions.

7.1 Techniques for Solving the One-Dimensional Wave Equation

This section provides techniques for solving the one-dimensional wave equation in Problem 3.

We begin with two techniques based on finite difference methods, and proceed to a transient solution using finite element methods. We then explore some classical analytical techniques for a simplified scenario, and compare our numerical results to these.

Finite Difference Methods

For each of the following two methods, we discretize the domain $[0, L]$ shown in Figure 2.3 into $N - 1$ many intervals using N many nodes for the endpoints of those intervals, which need not be of uniform length. This discretization is shown in Figure 7.1. We are not concerned with whether the boundary of the insulation or the load is located at a node, or between two nodes (the latter is typically the case with our solver).

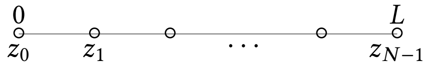


Figure 7.1: Discretization of the one-dimensional computational domain for the wave equation.

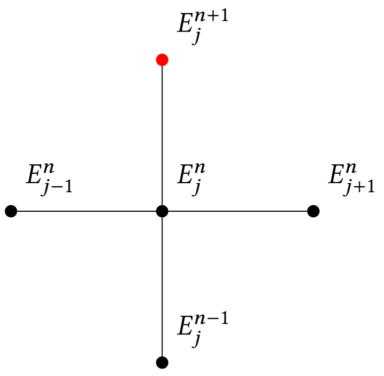


Figure 7.2: Computational stencil of the explicit finite difference scheme for solving the one-dimensional heat equation. Here, $j \in [1, N-2] \cap \mathbb{N}$ represents the position along the spatial domain, and $n \in \mathbb{N}$ represents the current time step. The nodes in black are ones at which the solution E is known, and the one in red may be solved for with knowledge of the ones in black.

We assume a time-marching scheme with uniform time steps of length $\Delta t_{\vec{E}}$, which, for simplicity of expression, we denote in this chapter as Δt.

Explicit, Second-Order Method

We use the second-order, explicit centered difference approximations

$$\left.\frac{\partial^2 E}{\partial z^2}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{E_{j-1}^n - 2E_j^n + E_{j+1}^n}{(z_{j+1}-z_j)(z_j - z_{j-1})},$$
$$\left.\frac{\partial^2 E}{\partial t^2}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{E_j^{n-1} - 2E_j^n + 2E_j^{n+1}}{(\Delta t)^2}, \quad \text{and}$$
$$\left.\frac{\partial E}{\partial t}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{E_j^{n+1} - E_j^{n-1}}{2\Delta t},$$

where the solution $E_y(z,t)$ to Equation 2.64 has been renamed as $E(z,t)$ for convenience, and where the subscripts denote field values in space and the superscripts those in time, so that, *e.g.*, $E_j^n := E(z_j, t_n)$. These approximations are valid for $j \in [1, N-2] \cap \mathbb{N}$ and $n \in \mathbb{N}$, and when the field values are plotted as a grid, the "stencil" for this scheme looks as in Figure 7.2. Applying the approximations in Equations 7.1 to the governing equation in Equation 2.64, we obtain the approximation

$$\frac{E_{j-1}^n - 2E_j^n + E_{j+1}^n}{(z_{j+1}-z_j)(z_j - z_{j-1})} - \frac{\mu\varepsilon'\left(E_j^{n-1} - 2E_j^n + 2E_j^{n+1}\right)}{(\Delta t)^2} - \frac{\mu\sigma\left(E_j^{n+1} - E_j^{n-1}\right)}{2\Delta t} = 0,$$

where we have used the dimensional representations ε' and μ of absolute permittivity and permeability, and the fact that $\frac{1}{c^2} = \varepsilon_0'\mu_0$. Keeping the term at time level $n+1$ on the left-hand side and moving all others to the right-hand side, we obtain

$$\begin{aligned}\left(\frac{\mu\sigma}{2\Delta t}+\frac{\mu\varepsilon'}{(\Delta t)^2}\right)E_j^{n+1} &= \left(\frac{\mu\sigma}{2\Delta t}-\frac{\mu\varepsilon'}{(\Delta t)^2}\right)E_j^{n-1}+\\ &+\left(\frac{1}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right)E_{j-1}^n+\left(\frac{2\mu\varepsilon'}{(\Delta t)^2}-\frac{2}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right)E_j^n+\\ &+\left(\frac{1}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right)E_{j+1}^n.\end{aligned} \tag{7.1}$$

Using the abbreviations

$$\begin{aligned} a &:= \frac{\mu\sigma}{2\Delta t}+\frac{\mu\varepsilon'}{(\Delta t)^2},\\ b &:= \frac{\mu\sigma}{a(2\Delta t)}-\frac{\mu\varepsilon'}{a(\Delta t)^2},\\ s_j &:= \frac{2\mu\varepsilon'}{a(\Delta t)^2}-\frac{2}{a(z_{j+1}-z_j)(z_j-z_{j-1})}, \quad \text{and}\\ t_j &:= \frac{1}{a(z_{j+1}-z_j)(z_j-z_{j-1})},\end{aligned}$$

and isolating the term E_j^{n+1} on the left-hand side, Equation 7.1 becomes

$$E_j^{n+1} = bE_j^{n-1}+t_jE_{j-1}^n+s_jE_j^n+t_jE_{j+1}^n, \tag{7.2}$$

for each $j \in [1, N-2] \cap \mathbb{N}$.

To examine the stability of this second-order scheme, we follow the energy considerations suggested in [142] by the von Neumann stability analysis; namely, by the law of conservation of energy, the energy contained in the field over the solution domain should not increase with time, and in fact, due to loss within the medium, the energy should decrease. To examine the energy of the field, we expand the field in terms of a Fourier series:

$$E_j^n := \sum_{m=-\infty}^{\infty} A_m^n e^{ik_m j\Delta z}, \tag{7.3}$$

where $k_m := \frac{m\pi}{L}$, and Δz is the minimum node spacing in the domain. The energy of the field is proportional to the sum of the squares of the amplitudes of the Fourier modes, and so to ensure that the energy does not increase with time, we examine the way these amplitudes behave within our time-stepping scheme. When Equation 7.3 is substituted into Equation 7.1 with the source term removed, the following relationship between A_m^{n-1}, A_m^n, and A_m^{n+1}, for $n \in \mathbb{N}$ and $m \in [0, N-1] \cap \mathbb{Z}$, arises:

$$A_m^{n+1} = 2\left(1-2r_j\sin^2\left(\frac{k_m\Delta z}{2}\right)\right)A_m^n - A_m^{n-1},$$

and so the *amplification factor* g_m, defined as $g_m := \frac{A_m^{n+1}}{A_m^n}$, will satisfy

$$g_m^2 - 2\left(1 - 2r_j \sin^2\left(\frac{k_m \Delta z}{2}\right)\right) g_m + 1 = 0,$$

where

$$r_j := \frac{\Delta t}{\mu\varepsilon'(z_{j+1} - z_j)(z_j - z_{j-1})}.$$

The quadratic formula gives a solution where $|g_m| < 1$ only in the case where

$$\left(1 - 2r_j \sin^2\left(\frac{k_m \Delta z}{2}\right)\right)^2 < 1,$$

where Δz represents the smallest spatial step in the domain. This yields the stability condition $|1 - 2r_j| < 1$, which yields

$$\Delta t \leq \Delta z \sqrt{\mu\varepsilon'} = \frac{\Delta z}{v_p}. \tag{7.4}$$

When simulating wave propagation with finite difference methods, an important consideration is that because of the numerical discretization, the simulated wave propagates at a velocity slightly different from the exact velocity; this kind of error is referred to as the *dispersion error*, and to quantify the error for our difference scheme, we test the simulation of a plane wave, computing its numerical wavenumber based on our scheme following the procedure in [142]. Such a plane wave may be expressed, as discussed in Chapter 2, as

$$E(z,t) := \operatorname{Re}\left\{E_0 e^{i(\omega t - kz)}\right\},$$

where ω is the angular frequency of the plane wave, and $k = \omega\sqrt{\mu\varepsilon}$ is referred to as the wavenumber. Assuming $(\Delta z)_j := z_j - z_{j-1}$, the simulated wave may be expressed as

$$E_j^n := \operatorname{Re}\left\{E_0 e^{i(\omega n \Delta t - k \sum_{J=1}^{j} (\Delta z)_J)}\right\},$$

which may be substituted into Equation 7.1 to approximate the numerical wavenumber. Stipulating that this numerical wavenumber equal the exact wavenumber, we obtain the condition

$$\Delta t = \Delta z \sqrt{\mu\varepsilon'} = \frac{\Delta z}{v_p}, \tag{7.5}$$

under which dispersion error is controlled.

To implement the boundary condition in Equation 2.74 at the left-hand endpoint $z = j = 0$, we set

$$E_0^n = \frac{2}{L}\sqrt{2P\left(\frac{\omega \cdot \mu_0}{\beta}\right)}, \tag{7.6}$$

for all $n \in \mathbb{N}$.

At the right-hand endpoint $z = L, j = N - 1$, Equation 2.74 gives us two possible conditions to impose. The perfect electric conductor condition is implemented via

$$E^n_{N-1} = 0, \tag{7.7}$$

for all $n \in \mathbb{N}$.

On the other hand, the absorbing boundary condition is implemented as the modified Neumann condition in Equation 2.62, whose incorporation into our scheme requires the temporary use of the "ghost node" z_N. The ghost node is placed to the right of the computational domain, so that $z_N - z_{N-1} = z_{N-1} - z_{N-2}$, as seen in Figure 7.3. Using the ghost node, we substitute the difference

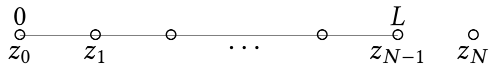


Figure 7.3: Discretization of the one-dimensional computational domain for the wave equation.

approximations

$$\left.\frac{\partial E}{\partial z}\right|_{j=N-1} \approx \frac{E^n_N - E^n_{N-2}}{z_N - z_{N-2}}, \qquad \left.\frac{\partial E}{\partial t}\right|_{j=N-1} \approx \frac{E^{n+1}_{N-1} - E^{n-1}_{N-1}}{2\Delta t}$$

into Equation 2.62 to obtain the approximation

$$E^n_N = E^n_{N-2} - \frac{(z_N - z_{N-2})}{2c\Delta t} E^{n+1}_{N-1} + \frac{(z_N - z_{N-2})}{2c\Delta t} E^{n-1}_{N-1}. \tag{7.8}$$

Meanwhile, using the ghost node, the approximation of the governing equation according to Equation 7.2 is

$$E^{n+1}_{N-1} = bE^{n-1}_{N-1} + t_{N-1}E^n_{N-2} + s_{N-1}E^n_{N-1} + t_{N-1}E^n_N,$$

into which we substitute Equation 7.8 to obtain

$$E^{n+1}_{N-1}\left(1 + \frac{s_{N-1}(z_N - z_{N-2})}{2c(\Delta t)}\right) = E^n_{N-2}\left(t_{N-1} + s_{N-1}\right) + s_{N-1}E^n_{N-1} + \left(b + \frac{s_{N-1}(z_N - z_{N-2})}{2c(\Delta t)}\right)E^n_{N-1},$$

which may be rewritten using the coefficients

$$\begin{aligned} d_{N-1} &:= 1 + \frac{s_{N-1}(z_N - z_{N-2})}{2c(\Delta t)}, \\ e_{N-1} &:= \frac{t_{N-1} + s_{N-1}}{d_{N-1}}, \\ f_{N-1} &:= \frac{s_{N-1}}{d_{N-1}}, \\ g_{N-1} &:= \frac{b}{d_{N-1}} + \frac{s_{N-1}(z_N - z_{N-2})}{2cd_{N-1}(\Delta t)}, \end{aligned}$$

as

$$E_{N-1}^{n+1} = e_{N-1}E_{N-2}^{n} + f_{N-1}E_{N-1}^{n} + g_{N-1}E_{N-1}^{n-1}. \tag{7.9}$$

Gathering Equations 7.6, 7.2, and 7.9, we obtain the linear system

$$\begin{aligned}
E_0^{n+1} &= E_0^n,\\
E_1^{n+1} &= t_1E_0^n + s_1E_1^n + t_1E_2^n + bE_1^{n-1},\\
E_2^{n+1} &= t_2E_1^n + s_2E_2^n + t_2E_3^n + bE_2^{n-1},\\
&\vdots\\
E_{N-2}^{n+1} &= s_{N-2}E_{N-3}^n + r_{N-2}E_{N-2}^n + s_{N-2}E_{N-1}^n + bE_{N-2}^{n-1},\\
E_{N-1}^{n+1} &= e_{N-1}E_{N-2}^n + f_{N-1}E_{N-1}^n + g_{N-1}E_{N-1}^{n-1},
\end{aligned}$$

which may be represented by the system

$$\begin{bmatrix} E_0^{n+1}\\ E_2^{n+1}\\ \vdots\\ E_j^{n+1}\\ \vdots\\ E_{N-1}^{n+1}\end{bmatrix} = b\begin{bmatrix} 0\\ E_1^{n-1}\\ E_2^{n-1}\\ \vdots\\ E_j^{n-1}\\ \vdots\\ E_{N-2}^{n-1}\\ \frac{g_{N-1}}{b}E_{N-1}^{n-1}\end{bmatrix} + \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\ 0 & t_1 & s_1 & t_1 & 0 & \cdots & \cdots & 0\\ \vdots & 0 & t_2 & s_2 & t_2 & 0 & \cdots & \vdots\\ \vdots & & & \ddots & \ddots & \ddots & & \vdots\\ \vdots & & & & t_{N-2} & s_{N-2} & t_{N-2} & 0\\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & e_{N-1} & f_{N-1}\end{bmatrix}\begin{bmatrix} E_0^n\\ E_1^n\\ E_2^n\\ \vdots\\ E_j^n\\ \vdots\\ E_{N-1}^n\end{bmatrix}. \tag{7.10}$$

Using the initial condition in Equation 2.71, which is interpreted according to our difference scheme as $E_j^0 = 0$ for $j \in [1, N-1] \cap \mathbb{N}$, and $E_0^0 = \frac{2}{L}\sqrt{2P\left(\frac{\omega \cdot \mu_0}{\beta}\right)}$, the electric field may be directly computed using the one matrix multiplication and the vector addition in Equation 7.10, for all subsequent time steps.

Implicit, Second-Order Method

As an alternative to the explicit method, our codes also implement an implicit finite difference method, which, as will be shown, involves solving a linear system at each time step, but on the other hand, allows for the use of a longer time step. This method employs the second-order, implicit

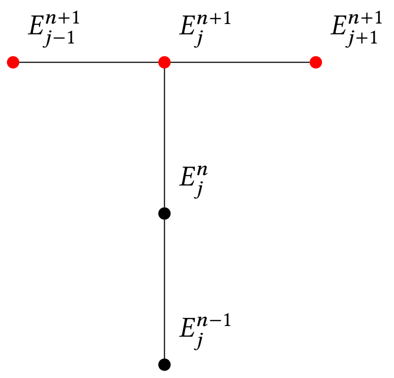


Figure 7.4: Computational stencil of the implicit finite difference scheme for solving the one-dimensional wave equation. Here, $j \in [1, N-2] \cap \mathbb{N}$ represents the position along the spatial domain, and $n \in \mathbb{N}$ represents the current time step. The nodes in black are ones at which the solution E is known, and the ones in red may be solved for with knowledge of the ones in black.

centered difference approximations

$$\left.\frac{\partial^2 E}{\partial z^2}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{E_{j-1}^{n+1} - 2E_j^{n+1} + E_{j+1}^{n+1}}{(z_{j+1} - z_j)(z_j - z_{j-1})},$$

$$\left.\frac{\partial^2 E}{\partial t^2}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{E_j^{n-1} - 2E_j^n + 2E_j^{n+1}}{(\Delta t)^2}, \quad \text{and}$$

$$\left.\frac{\partial E}{\partial t}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{E_j^{n+1} - E_j^{n-1}}{2\Delta t},$$

where, as before, the solution $E_y(z,t)$ to Equation 2.64 has been renamed as $E(z,t)$ for convenience, and where the subscripts denote field values in space and the superscripts those in time, so that, *e.g.*, $E_j^n := E(z_j, t_n)$. These approximations are valid for $j \in [1, N-2] \cap \mathbb{N}$ and $n \in \mathbb{N}$, and when the field values are plotted as a grid, the "stencil" for this scheme looks as in Figure 7.4.

Applying the approximations in Equations 7.1 to the governing equation in Equation 2.64, we obtain the approximation

$$\frac{E_{j-1}^{n+1} - 2E_j^{n+1} + E_{j+1}^{n+1}}{(z_{j+1} - z_j)(z_j - z_{j-1})} - \frac{\mu\varepsilon'\left(E_j^{n-1} - 2E_j^n + 2E_j^{n+1}\right)}{(\Delta t)^2} - \frac{\mu\sigma\left(E_j^{n+1} - E_j^{n-1}\right)}{2\Delta t} = 0,$$

where, as in the explicit case, we have used the dimensional representations ε' and μ of absolute permittivity and permeability, and the fact that $\frac{1}{c^2} = \varepsilon_0'\mu_0$. Keeping the terms at time level $n+1$ on the left-hand side and moving all others to the right-hand side, we obtain

$$\begin{aligned}\frac{E_{j-1}^{n+1}}{(z_{j+1}-z_j)(z_j-z_{j-1})} + E_j^{n+1}\left(-\frac{2}{(z_{j+1}-z_j)(z_j-z_{j-1})} - \frac{2\mu\varepsilon'}{(\Delta t)^2} - \frac{\mu\sigma}{2\Delta t}\right) + \\ + \frac{E_{j+1}^{n+1}}{(z_{j+1}-z_j)(z_j-z_{j-1})} = E_j^{n-1}\left(\frac{\mu\varepsilon'}{(\Delta t)^2} - \frac{\mu\sigma}{2\Delta t}\right) - \frac{2\mu\varepsilon' E_j^n}{(\Delta t)^2}.\end{aligned} \tag{7.11}$$

Using the abbreviations

$$\begin{aligned}(\Delta z)_j^2 &:= (z_{j+1}-z_j)(z_j-z_{j-1}),\\ s_j &:= -\frac{2}{(z_{j+1}-z_j)(z_j-z_{j-1})} - \frac{2\mu\varepsilon'}{(\Delta t)^2} - \frac{\mu\sigma}{2\Delta t},\\ a_j &:= \frac{\mu\varepsilon'}{(\Delta t)^2} - \frac{\mu\sigma}{2\Delta t},\\ b_j &:= \frac{2\mu\varepsilon'}{(\Delta t)^2},\end{aligned}$$

the difference approximation becomes

$$\frac{E_{j-1}^{n+1}}{(\Delta z)_j^2} + s_j E_j^{n+1} + \frac{E_{j+1}^{n+1}}{(\Delta z)_j^2} = a_j E_j^{n-1} - b_j E_j^n. \tag{7.12}$$

To implement the boundary condition at the left-hand endpoint $z = j = 0$, we use the expression in Equations 7.6. To implement the absorbing boundary condition at the right-hand side, the approximation will differ from Equation 7.8, because the approximation of the spatial derivative in Equation 2.62 should, in this case, be taken at the $(n+1)^{\text{st}}$ time step. Using the ghost node in Figure 7.3, we substitute the difference approximations

$$\left.\frac{\partial E}{\partial z}\right|_{j=N-1} \approx \frac{E_N^{n+1} - E_{N-2}^{n+1}}{2(\Delta z)_{N-1}}, \qquad \left.\frac{\partial E}{\partial t}\right|_{j=N-1} \approx \frac{E_{N-1}^{n+1} - E_{N-1}^{n-1}}{2\Delta t}$$

into Equation 2.62 to obtain the approximation

$$E_N^{n+1} = E_{N-2}^{n+1} + \frac{(\Delta z)_{N-1}}{c\Delta t} E_{N-1}^{n+1} - \frac{(\Delta z)_{N-1}}{c\Delta t} E_{N-1}^{n-1}. \tag{7.13}$$

Using the ghost node, the approximation of the governing equation at the right-hand endpoint, according to Equation 7.12, is

$$\frac{E_{N-2}^{n+1}}{(\Delta z)_{N-1}^2} + s_{N-1} E_{N-1}^{n+1} + \frac{E_N^{n+1}}{(\Delta z)_{N-1}^2} = a_{N-1} E_{N-1}^{n-1} - b_{N-1} \frac{2\mu\varepsilon' E_{N-1}^n}{(\Delta t)^2},$$

into which Equation 7.8 may be substituted to obtain the expression

$$\frac{E_{N-2}^{n+1}}{(\Delta z)_{N-1}^2} + s_{N-1}E_{N-1}^{n+1} + \frac{E_{N-2}^{n+1} + \frac{(\Delta z)_{N-1}}{c\Delta t}E_{N-1}^{n+1} - \frac{(\Delta z)_{N-1}}{c\Delta t}E_{N-1}^{n-1}}{(\Delta z)_{N-1}^2} = a_{N-1}E_{N-1}^{n-1} - b_{N-1}E_{N-1}^{n},$$

which simplifies to

$$\frac{2}{(\Delta z)_{N-1}^2}E_{N-2}^{n+1} + d_{N-1}E_{N-1}^{n+1} = e_{N-1}E_{N-1}^{n-1} - b_{N-1}E_{N-1}^{n},$$

where

$$d_{N-1} := s_{N-1} + \frac{1}{c(\Delta z)_{N-1}\Delta t}, \qquad e_{N-1} := a_{N-1} + \frac{1}{c(\Delta z)_{N-1}\Delta t}. \tag{7.14}$$

Taken together, Equations 7.6, 7.12, and 7.14 result in the linear system

$$\begin{aligned}
E_0^{n+1} &= \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}\\
\frac{E_0^{n+1}}{(\Delta z)_1^2} + s_1E_1^{n+1} + \frac{E_2^{n+1}}{(\Delta z)_1^2} &= a_1E_1^{n-1} - b_1E_1^n\\
\frac{E_1^{n+1}}{(\Delta z)_2^2} + s_2E_2^{n+1} + \frac{E_3^{n+1}}{(\Delta z)_2^2} &= a_2E_2^{n-1} - b_2E_2^n\\
&\vdots\\
\frac{E_{N-3}^{n+1}}{(\Delta z)_{N-2}^2} + s_{N-2}E_{N-2}^{n+1} + \frac{E_{N-1}^{n+1}}{(\Delta z)_{N-2}^2} &= a_{N-2}E_{N-2}^{n-1} - b_{N-2}E_{N-2}^n\\
\frac{2}{(\Delta z)_{N-1}^2}E_{N-2}^{n+1} + d_{N-1}E_{N-1}^{n+1} &= e_{N-1}E_{N-1}^{n-1} - b_{N-1}E_{N-1}^n,
\end{aligned}$$

which has N many equations in N many unknowns, and is equivalent to its matrix representation

$$\begin{bmatrix}
1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0\\
0 & t_1 & s_1 & t_1 & 0 & \cdots & \cdots & 0\\
\vdots & 0 & t_2 & s_2 & t_2 & 0 & \cdots & \vdots\\
\vdots & & & \ddots & \ddots & \ddots & & \vdots\\
\vdots & & & & t_{N-2} & s_{N-2} & t_{N-2} & 0\\
0 & \cdots & \cdots & \cdots & \cdots & 0 & \frac{2}{(\Delta z)_{N-1}^2} & d_{N-1}
\end{bmatrix}
\begin{bmatrix}
E_0^{n+1}\\ E_2^{n+1}\\ \vdots\\ E_j^{n+1}\\ \vdots\\ E_{N-1}^{n+1}
\end{bmatrix}
=
\begin{bmatrix}
0\\ a_1E_1^{n-1}\\ a_2E_2^{n-1}\\ \vdots\\ a_jE_j^{n-1}\\ \vdots\\ a_{N-2}E_{N-2}^{n-1}\\ e_{N-1}E_{N-1}^{n-1}
\end{bmatrix}
+
\begin{bmatrix}
E_0^n\\ b_1E_1^n\\ b_2E_2^n\\ \vdots\\ b_jE_j^n\\ \vdots\\ b_{N-1}E_{N-1}^n
\end{bmatrix}, \tag{7.15}$$

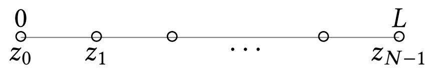


Figure 7.5: Discretization of the one-dimensional spatial domain for the finite element solution.

where $t_j := \frac{1}{(\Delta z)_j^2}$.

Using the initial condition in Equation 2.71, which is interpreted according to our difference scheme as $E_j^0 = 0$ for $j \in [1, N-1] \cap \mathbb{N}$, and $E_0^0 = \frac{2}{L}\sqrt{2P\left(\frac{\omega \cdot \mu_0}{\beta}\right)}$, the electric field may be computed by solving the matrix system in Equation 7.15, for all subsequent time steps.

Finite Element Methods

Weak Formulation of the Governing Equation

We formulate a single matrix equation that, given an initial electric field, can be solved at each time step for a transient solution.

Spatial Discretization

We use N many nodes, not necessarily evenly spaced, to discretize the domain into $N-1$ many elements, as shown in Figure 7.5

On this domain, we assume that our solutions $E(z,t)$ are separable, so that $E(z,t) := \sum_{j=0}^{N-1} T_j(t)E_j(z)$ for some functions $T_j(t)$ and $E_j(z)$, and we plug this into Equation 2.64, multiply by the test function $W(z) := \sum_{i=0}^{N-1} W_i(z)$, where for $i \in [0, N-1] \cap \mathbb{N}$, $W_i(z)$ is some known function, and integrate over the domain $[z_0, z_{N-1}]$:

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left\{T_j(t)\int_{z_0}^{z_n} W_i(z)\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{\mu(z,t)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\right)\mathrm{d}z + \right.$$
$$\left. + \int_{z_0}^{z_{N-1}}\left(\mu_0\sigma(z,t)\frac{\mathrm{d}T_j}{\mathrm{d}t} + \frac{\varepsilon'(z,t)}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)W_i(z)E_j(z)\mathrm{d}z\right\} = 0.$$

We integrate the second spatial derivative term by parts:

$$\int_{z_0}^{z_{N-1}} W_i(z)\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{\mu(z,t)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\right)\mathrm{d}z = \left.\frac{W_i(z)}{\mu(z,t)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\right|_{z_0}^{z_{N-1}} - \int_{z_1}^{z_n}\frac{1}{\mu(z,t)}\frac{\mathrm{d}W_i}{\mathrm{d}z}\frac{\mathrm{d}E_j}{\mathrm{d}z}\mathrm{d}z,$$

and we move the first term of the result (henceforth called the "boundary term") to the right-hand side of the governing equation to obtain the form:

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left\{-T_j(t)\int_{z_1}^{z_n}\frac{1}{\mu(z,t)}\frac{\mathrm{d}W_i}{\mathrm{d}z}\frac{\mathrm{d}E_j}{\mathrm{d}z}\mathrm{d}z+\int_{z_0}^{z_{N-1}}\left(\mu_0\sigma(z,t)\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'(z,t)}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)W_i(z)E_j(z)\mathrm{d}z\right\}=$$
$$=-\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}T_j(t)\frac{W_i(z)}{\mu(z,t)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\bigg|_{z_0}^{z_{N-1}}. \tag{7.16}$$

We now explicitly state the definition of our functions E_i, noting that since we will proceed with a Galerkin formulation, $W_i(z)\equiv E_i(z)$ for all $i\in[0,N-1]\cap\mathbb{N}$. For each $i\in[1,N-2]\cap\mathbb{N}$, we define $E_i(z)$ to be the hat function

$$E_i(z):=\begin{cases}\frac{z-z_{i-1}}{z_i-z_{i-1}}, & \text{if } z\in[z_{i-1},z_i]\\ \frac{z_{i+1}-z}{z_{i+1}-z_i}, & \text{if } z\in[z_i,z_{i+1}]\\ 0, & \text{if } z\notin[z_{i-1},z_{i+1}]\end{cases}:=\begin{cases}\frac{z-z_{i-1}}{h_{i-1}}, & \text{if } z\in[z_{i-1},z_i]\\ \frac{z_{i+1}-z}{h_i}, & \text{if } z\in[z_i,z_{i+1}]\\ 0, & \text{if } z\notin[z_{i-1},z_{i+1}]\end{cases}, \tag{7.17}$$

and

$$E_0(z)=\begin{cases}\frac{z_1-z}{z_1-z_0}, & \text{if } z\in[z_0,z_1],\\ 0, & \text{if } z\notin[z_0,z_1]\end{cases}:=\begin{cases}\frac{z_1-z}{h_0}, & \text{if } z\in[z_0,z_1],\\ 0, & \text{if } z\notin[z_0,z_1]\end{cases} \tag{7.18}$$

and

$$E_{N-1}(z)=\begin{cases}\frac{z-z_{N-2}}{z_{N-1}-z_{N-2}}, & \text{if } z\in[z_{N-2},z_{N-1}],\\ 0, & \text{if } z\notin[z_{N-2},z_{N-1}]\end{cases}:=\begin{cases}\frac{z-z_{N-2}}{h_{N-2}}, & \text{if } z\in[z_{N-2},z_{N-1}],\\ 0, & \text{if } z\notin[z_{N-2},z_{N-1}],\end{cases} \tag{7.19}$$

where we have defined h_i to be the length of the i^{th} element; that is, the length of the interval $[z_i,z_{i+1}]$.

Because of our choice of the E_j as hat functions, and because of our assumption that $E(z,t)$ was separable and continuous, we may conclude that the functions $T_j(t)$ represent the evolution of the electric field intensity at the nodes j. The hat functions look as shown in Figure 7.6.

Note that for each $i\in[1,N-2]\cap\mathbb{N}$, the derivatives of the hat functions are:

$$\frac{\mathrm{d}E_i}{\mathrm{d}z}=\begin{cases}\frac{1}{h_{i-1}}, & \text{if } z\in(z_{i-1},z_i)\\ -\frac{1}{h_i}, & \text{if } z\in(z_i,z_{i+1})\\ 0, & \text{if } z\notin[z_{i-1},z_{i+1}],\\ \text{undefined}, & \text{if } z\in\{z_{i-1},z_i,z_{i+1}\}\end{cases},$$

and

$$\frac{\mathrm{d}E_0}{\mathrm{d}z}=\begin{cases}\frac{-1}{h_0}, & \text{if } z\in(z_0,z_1),\\ 0, & \text{if } z\notin[z_0,z_1],\\ \text{undefined}, & \text{if } z\in\{z_0,z_1\}\end{cases}\quad\text{and}\quad\frac{\mathrm{d}E_n}{\mathrm{d}z}=\begin{cases}\frac{1}{h_{N-2}}, & \text{if } z\in(z_{N-2},z_{N-1}),\\ 0, & \text{if } z\notin[z_{N-2},z_{N-1}],\\ \text{undefined}, & \text{if } z\in\{z_{N-2},z_{N-1}\}.\end{cases}.$$

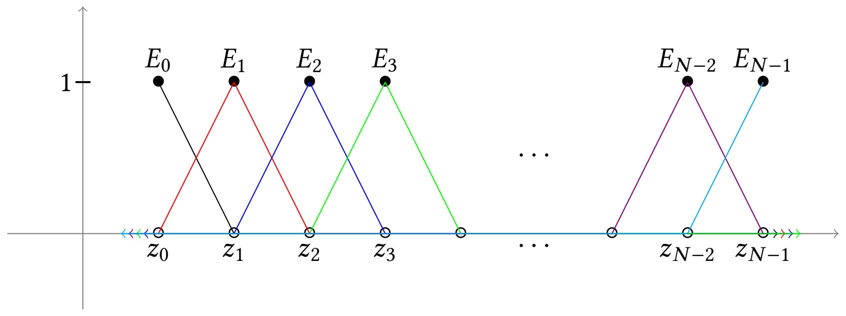


Figure 7.6: "Hat functions" E_j used in the finite element solution of the electromagnetic wave equation.

With this being the case, we note that if $|i-j| > 1$, then $W_i(z)E_j(z) = \frac{dW_i}{dz}\frac{dE_j}{dz} = 0$. In case $j = i+1$, $i \in [0, N-2] \cap \mathbb{N}$, we have that

$$\frac{dW_i}{dz}\frac{dE_{i+1}}{dz} = \begin{cases} -\frac{1}{h_i^2} & \text{if } z \in (z_i, z_{i+1}), \\ 0 & \text{if } z \notin [z_i, z_{i+1}], \end{cases}$$

and in case $i = j$,

$$\frac{dW_i}{dz}\frac{dE_i}{dz} = \begin{cases} \frac{1}{h_{i-1}^2}, & \text{if } z \in (z_{i-1}, z_i) \\ -\frac{1}{h_i^2}, & \text{if } z \in (z_i, z_{i+1}) \\ 0, & \text{if } z \notin [z_{i-1}, z_{i+1}], \end{cases}$$

and

$$\frac{dW_0}{dz}\frac{dE_0}{dz} = \begin{cases} \frac{1}{h_0^2}, & \text{if } z \in (z_0, z_1), \\ 0, & \text{if } z \notin [z_0, z_1] \end{cases} \quad \text{and} \quad \frac{dW_{N-1}}{dz}\frac{dE_{N-1}}{dz} = \begin{cases} \frac{1}{h_{N-2}^2}, & \text{if } z \in (z_{N-2}, z_{N-1}), \\ 0, & \text{if } z \notin [z_{N-2}, z_{N-1}], \end{cases}.$$

Moreover, assuming that on the i^{th} element (that is, for $z \in [z_i, z_{i+1}]$) at a fixed time t, $\mu(z,t)$ has the constant value μ_i, we may use the values of the derivatives calculated above to obtain:

$$\int_{z_{N-1}}^{z_0} \frac{1}{\mu(z,t)}\frac{dW_i}{dz}\frac{dE_j}{dz}dz = 0, \text{ if } |i-j| > 1,$$

$$\int_{z_0}^{z_{N-1}} \frac{1}{\mu(z,t)}\frac{dW_i}{dz}\frac{dE_{i-1}}{\partial z}dz = \int_{z_{i-1}}^{z_i} \frac{1}{\mu(z,t)}\left(\frac{-1}{h_{i-1}^2}\right)dz = -\frac{1}{\mu_{i-1}h_{i-1}} \text{ for } i \in [1, N-1],$$

$$\int_{z_0}^{z_{N-1}} \frac{1}{\mu(z,t)}\frac{dW_i}{dz}\frac{dE_i}{dz}dz = \int_{z_{i-1}}^{z_i} \frac{1}{\mu(z,t)}\frac{1}{h_{i-1}^2}dz + \int_{z_i}^{z_{i+1}} \frac{1}{\mu(z,t)}\frac{1}{h_i^2}dz = \frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i}, \text{ for } i \in [1, N-2],$$

$$\int_{z_0}^{z_{N-1}} \frac{1}{\mu(z,t)}\frac{dW_0}{dz}\frac{dE_0}{dz}dz = \frac{1}{\mu_0 h_0}, \text{ and } \int_{z_0}^{z_{N-1}} \frac{1}{\mu(z,t)}\frac{dW_{N-1}}{dz}\frac{dE_{N-1}}{dz}dz = \frac{1}{\mu_{N-2}h_{N-2}};$$

and

$$\int_{z_0}^{z_{N-1}} \frac{1}{\mu(z,t)} \frac{\mathrm{d}W_i}{\mathrm{d}z} \frac{\mathrm{d}E_{i+1}}{\mathrm{d}z} \mathrm{d}z = \int_{x_i}^{x_{i+1}} \frac{1}{\mu(z,t)} \left(\frac{-1}{h_i^2}\right) \mathrm{d}z = -\frac{1}{\mu_i h_i} \text{ for } i \in [0, N-2].$$

Also, note that in case $j = i + 1$, $i \in [0, N-2] \cap \mathbb{N}$, we have that

$$W_i(z)E_{i+1}(z) = \begin{cases} \frac{(z-z_i)^2}{h_i^2} & \text{if } z \in [z_i, z_{i+1}], \\ 0 & \text{if } z \notin [z_i, z_{i+1}], \end{cases}$$

and in case $i = j$,

$$W_i(z)E_i(z) = \begin{cases} \frac{(z-z_{i-1})^2}{h_{i-1}^2}, & \text{if } z \in [z_{i-1}, z_i] \\ \frac{(z-z_{i+1})^2}{h_i^2}, & \text{if } z \in [z_i, z_{i+1}] \\ 0, & \text{if } z \notin [z_{i-1}, z_{i+1}], \end{cases}$$

and

$$W_0(z)E_0(z) = \begin{cases} \frac{(z-z_1)}{h_0^2}, & \text{if } z \in [z_0, z_1], \\ 0, & \text{if } z \notin [z_0, z_1] \end{cases} \quad \text{and} \quad W_{N-1}(z)E_{N-1}(z) = \begin{cases} \frac{(z-z_{N-2})}{h_{N-2}^2}, & \text{if } z \in [z_{N-2}, z_{N-1}], \\ 0, & \text{if } z \notin [z_{N-2}, z_{N-1}], \end{cases}.$$

Moreover, assuming that on the ith element (that is, for $z \in [z_i, z_{i+1}]$) at a fixed time t, $\sigma(z,t)$ and $\varepsilon'(z,t)$ have the constant values σ_i and ε_i' respectively, we may use the values calculated above to obtain:

$$\int_{z_0}^{z_{N-1}} \left(\mu_0 \sigma(z,t) \frac{\mathrm{d}T_j}{\mathrm{d}t} + \frac{\varepsilon'(z,t)}{c^2} \frac{\mathrm{d}^2 T_j}{\mathrm{d}t^2}\right) W_i E_j \mathrm{d}z = 0, \text{ if } |i-j| > 1,$$

and

$$\begin{aligned} \int_{z_0}^{z_{N-1}} \left(\mu_0 \sigma(z,t) \frac{\mathrm{d}T_j}{\mathrm{d}t} + \frac{\varepsilon'(z,t)}{c^2} \frac{\mathrm{d}^2 T_j}{\mathrm{d}t^2}\right) W_i E_{i-1} \mathrm{d}z &= \left(\mu_0 \sigma_{i-1} \frac{\mathrm{d}T_j}{\mathrm{d}t} + \frac{\varepsilon_{i-1}'}{c^2} \frac{\mathrm{d}^2 T_j}{\mathrm{d}t^2}\right) \int_{z_{i-1}}^{z_i} \frac{(z_{i-1}-z)^2}{h_{i-1}^2} \mathrm{d}z \\ &= \left(\mu_0 \sigma_{i-1} \frac{\mathrm{d}T_j}{\mathrm{d}t} + \frac{\varepsilon_{i-1}'}{c^2} \frac{\mathrm{d}^2 T_j}{\mathrm{d}t^2}\right) \left.\frac{(z-z_{i-1})^3}{3h_{i-1}^2}\right|_{z_{i-1}}^{z_i} \\ &= \left(\mu_0 \sigma_{i-1} \frac{\mathrm{d}T_j}{\mathrm{d}t} + \frac{\varepsilon_{i-1}'}{c^2} \frac{\mathrm{d}^2 T_j}{\mathrm{d}t^2}\right) \frac{h_{i-1}}{3}, \text{ for } i \in [1, N-1], \end{aligned}$$

$$\begin{aligned}\int_{z_0}^{z_{N-1}}\left(\mu_0\sigma(z,t)\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'(z,t)}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)W_iE_i\mathrm{d}z &= \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\int_{z_{i-1}}^{z_i}\frac{(z-z_{i-1})^2}{h_{i-1}^2}\mathrm{d}z+\\
&\quad+\left(\mu_0\sigma_i\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\int_{z_i}^{z_{i+1}}\frac{(z_{i+1}-z)^2}{h_i^2}\mathrm{d}z\\
&= \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\left.\frac{(z-z_{i-1})^3}{3h_{i-1}^2}\right|_{z_{i-1}}^{z_i}+\\
&\quad+\left(\mu_0\sigma_i\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\left.\frac{(z-z_i)^3}{3h_i^2}\right|_{z_i}^{z_{i+1}}\\
&= \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\frac{h_{i-1}}{3}+\left(\mu_0\sigma_i\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\frac{h_i}{3},\end{aligned}$$

for $i\in[1,N-2]$. Also

$$\begin{aligned}&\int_{z_0}^{z_{N-1}}\left(\mu_0\sigma(z,t)\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'(z,t)}{c^2}\frac{\mathrm{d}^2T}{\mathrm{d}t^2}\right)W_0E_0\mathrm{d}z=\left(\mu_0\sigma_0\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_0}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\frac{h_0}{3}\text{ and}\\
&\int_{z_0}^{z_{N-1}}\left(\mu_0\sigma(z,t)\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'(z,t)}{c^2}\frac{\mathrm{d}^2T}{\mathrm{d}t^2}\right)W_{N-1}E_{N-1}\mathrm{d}z=\left(\mu_0\sigma_{n-1}\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_{N-2}}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\frac{h_{N-2}}{3},\end{aligned}$$

and

$$\begin{aligned}\int_{z_0}^{z_{N-1}}\left(\mu_0\sigma(z,t)\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'(z,t)}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)W_iE_{i+1}\mathrm{d}z &= \left(\mu_0\sigma_i\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\int_{z_i}^{z_{i-1}}\frac{(z-z_i)^2}{h_i^2}\mathrm{d}z\\
&= \left(\mu_0\sigma_i\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\left.\frac{(z-z_i)^3}{3h_i^2}\right|_{z_i}^{z_{i+1}}\\
&= \left(\mu_0\sigma_i\frac{\mathrm{d}T_j}{\mathrm{d}t}+\frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_j}{\mathrm{d}t^2}\right)\frac{h_i}{3},\text{ for }i\in[0,N-2].\end{aligned}$$

Using all of the integral values we have just computed in the weak form of the governing equation

7.16, we obtain

$$\begin{aligned}
&\sum_{i=1}^{N-2}\left\{\frac{T_{i-1}(t)}{\mu_{i-1}h_{i-1}} + \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_{i-1}}{\mathrm{d}t} + \frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_{i-1}}{\mathrm{d}t^2}\right)\frac{h_{i-1}}{3} -\right.\\
&\quad - T_i(t)\left(\frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i}\right) + \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_i}{\mathrm{d}t} + \frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_i}{\mathrm{d}t^2}\right)\frac{h_{i-1}}{3} + \left(\mu_0\sigma_i\frac{\mathrm{d}T_i}{\mathrm{d}t} + \frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_i}{\mathrm{d}t^2}\right)\frac{h_i}{3} +\\
&\quad\left. + \frac{T_{i+1}(t)}{\mu_i h_i} + \left(\mu_0\sigma_i\frac{\mathrm{d}T_{i+1}}{\mathrm{d}t} + \frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_{i+1}}{\mathrm{d}t^2}\right)\frac{h_i}{3}\right\} -\\
&\quad - \frac{T_0(t)}{\mu_0(t)h_0} + \left(\mu_0\sigma_0\frac{\mathrm{d}T_0}{\mathrm{d}t} + \frac{\varepsilon'_0}{c^2}\frac{\mathrm{d}^2T_0}{\mathrm{d}t^2}\right)\frac{h_0}{3} -\\
&\quad - \frac{T_{N-1}(t)}{\mu_{N-2}(t)h_{N-2}} + \left(\mu_0\sigma_{N-2}\frac{\mathrm{d}T_{N-1}}{\mathrm{d}t} + \frac{\varepsilon'_{N-2}}{c^2}\frac{\mathrm{d}^2T_{N-1}}{\mathrm{d}t^2}\right)\frac{h_{N-2}}{3} =\\
&= -\sum_{i=1}^{N-2}\left\{\frac{W_i(x)}{\mu(z,t)}\left(T_{i-1}(t)\frac{\mathrm{d}E_{i-1}}{\mathrm{d}z} + T_i(t)\frac{\mathrm{d}E_i}{\mathrm{d}z} + T_{i+1}(t)\frac{\mathrm{d}E_{i+1}}{\mathrm{d}z}\right)\right\}\Bigg|_{z_0}^{z_{N-1}} -\\
&\quad - \frac{W_0(z)}{\mu(z,t)}T_0(t)\frac{\mathrm{d}E_0}{\mathrm{d}z}\Bigg|_{z_0}^{z_{N-1}} - \frac{W_{N-1}(z)}{\mu(z,t)}T_{N-1}(t)\frac{\mathrm{d}E_{N-1}}{\mathrm{d}z}\Bigg|_{z_0}^{z_{N-1}}.
\end{aligned}$$

As for the boundary term, observe that $W_i(z)\big|_{z_0}^{z_{N-1}}$ is zero whenever $i \notin \{0, N-1\}$, and that $W_0(z_0) = W_{N-1}(z_{N-1}) = 1$ and $W_0(z_{N-1}) = W_{N-1}(z_0) = 0$, and so the right-hand side (henceforth referred to as "BT") becomes:

$$\mathrm{BT} = \frac{T_0(t)}{\mu_0(t)}\frac{\mathrm{d}E_0}{\mathrm{d}z}\bigg|_{z=z_0} - \frac{T_n(t)}{\mu_{N-2}(t)}\frac{\mathrm{d}E_{N-1}}{\mathrm{d}z}\bigg|_{z_{N-1}}.$$

However, the spatial derivatives $\frac{\mathrm{d}E_0}{\mathrm{d}z}\big|_{z=z_0}$ and $\frac{\mathrm{d}E_{N-1}}{\mathrm{d}z}\big|_{z=z_{N-1}}$ are not defined at their respective evaluation points. We will thus wait to evaluate this term until we explicitly apply the boundary conditions. Note for now, however, that $(\mathrm{BT})_i = 0$ unless $i \in \{0, N-1\}$.

We thus obtain the weak form of the governing equation

$$\begin{aligned}
&\sum_{i=1}^{N-2}\Bigg\{\frac{T_{i-1}(t)}{\mu_{i-1}(t)h_{i-1}} + \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_{i-1}}{\mathrm{d}t} + \frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_{i-1}}{\mathrm{d}t^2}\right)\frac{h_{i-1}}{3} - \\
&\quad - T_i(t)\left(\frac{1}{\mu_{i-1}(t)h_{i-1}} + \frac{1}{\mu_i(t)h_i}\right) + \left(\mu_0\sigma_{i-1}\frac{\mathrm{d}T_i}{\mathrm{d}t} + \frac{\varepsilon'_{i-1}}{c^2}\frac{\mathrm{d}^2T_i}{\mathrm{d}t^2}\right)\frac{h_{i-1}}{3} + \left(\mu_0\sigma_i\frac{\mathrm{d}T_i}{\mathrm{d}t} + \frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_i}{\mathrm{d}t^2}\right)\frac{h_i}{3} + \\
&\qquad + \frac{T_{i+1}(t)}{\mu_i(t)h_i} + \left(\mu_0\sigma_i\frac{\mathrm{d}T_{i+1}}{\mathrm{d}t} + \frac{\varepsilon'_i}{c^2}\frac{\mathrm{d}^2T_{i+1}}{\mathrm{d}t^2}\right)\frac{h_i}{3}\Bigg\} - \\
&\qquad - \frac{T_0(t)}{\mu_0(t)h_0} + \left(\mu_0\sigma_0\frac{\mathrm{d}T_0}{\mathrm{d}t} + \frac{\varepsilon'_0}{c^2}\frac{\mathrm{d}^2T_0}{\mathrm{d}t^2}\right)\frac{h_0}{3} - \\
&\qquad\quad - \frac{T_{N-1}(t)}{\mu_{N-2}(t)h_{N-2}} + \left(\mu_0\sigma_{N-2}\frac{\mathrm{d}T_{N-1}}{\mathrm{d}t} + \frac{\varepsilon'_{N-2}}{c^2}\frac{\mathrm{d}^2T_{N-1}}{\mathrm{d}t^2}\right)\frac{h_{N-2}}{3} = \mathrm{BT}.
\end{aligned} \tag{7.20}$$

Time Discretization

Note that the second-order central difference approximation to the first time derivative of $T_i(t)$ at the time step $t = t_k$ is, for $i \in [1,\infty) \cap \mathbb{N}$,

$$\left.\frac{\mathrm{d}T_i}{\mathrm{d}t}\right|_{t=t_k} \approx \frac{T_i(t_{k+1}) - T_i(t_{k-1})}{t_{k+1} - t_{k-1}} := \frac{T_i(t_{k+1}) - T_i(t_{k-1})}{2\Delta t},$$

where Δt is the (uniform) length of each time step. The central difference approximation to the second derivative is:

$$\left.\frac{\mathrm{d}^2T_i}{\mathrm{d}t^2}\right|_{t=t_k} \approx \frac{T_i(t_{k-1}) - 2T_i(t_k) + T_i(t_{k-1})}{(\Delta t)^2}.$$

For $t \in [t_k, t_{k+1}]$, we replace $T_i(t)$ in the weak form of the governing equation by

$$T_i(t) \approx \theta T_i(t_{k+1}) + (1-\theta)T_i(t_k),$$

for some weighting constant $\theta \in [0,1]$.

Using these approximations in our governing equation 7.20, keeping all t_{k+1} terms on the left-hand side and moving the t_k and t_{k-1} terms to the right-hand side, we obtain

$$
\begin{aligned}
&\sum_{i=1}^{N-2}\Bigg\{T_{i-1}(t_{k+1})\left[\frac{\theta}{\mu_{i-1}h_{i-1}}+\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}+\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}\right]+\\
&\quad+T_i(t_{k+1})\left[-\theta\left(\frac{1}{\mu_{i-1}h_{i-1}}+\frac{1}{\mu_i h_i}\right)+\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}+\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}+\left(\frac{\mu_0\sigma_i}{2\Delta t}+\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]+\\
&\qquad+T_{i+1}(t_{k+1})\left[\frac{\theta}{\mu_i h_i}+\left(\frac{\mu_0\sigma_i}{2\Delta t}+\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]\Bigg\}+\\
&\qquad\quad+T_0(t_{k+1})\left[-\frac{\theta}{\mu_0 h_0}+\left(\frac{\mu_0\sigma_0}{2\Delta t}+\frac{\varepsilon'_0}{c^2(\Delta t)^2}\right)\frac{h_0}{3}\right]+\\
&\qquad\qquad+T_{N-1}(t_{k+1})\left[-\frac{\theta}{\mu_{N-2}h_{N-2}}+\left(\frac{\mu_0\sigma_{N-2}}{2\Delta t}+\frac{\varepsilon'_{N-2}}{c^2(\Delta t)^2}\right)\frac{h_{N-2}}{3}\right]=\\
&=\sum_{i=1}^{N-2}\Bigg\{T_{i-1}(t_k)\left[\frac{\theta-1}{\mu_{i-1}h_{i-1}}+\frac{2\varepsilon'_{i-1}h_{i-1}}{3c^2(\Delta t)^2}\right]+\\
&\quad+T_i(t_k)\left[(1-\theta)\left(\frac{1}{\mu_{i-1}h_{i-1}}+\frac{1}{\mu_i h_i}\right)+\frac{2\varepsilon'_{i-1}h_{i-1}}{3c^2(\Delta t)^2}+\frac{2\varepsilon'_i h_i}{3c^2(\Delta t)^2}\right]+\\
&\qquad+T_{i+1}(t_k)\left[\frac{\theta-1}{\mu_i h_i}+\frac{2\varepsilon'_i h_i}{3c^2(\Delta t)^2}\right]\Bigg\}+\\
&\qquad\quad+T_0(t_k)\left[\frac{1-\theta}{\mu_0 h_0}+\frac{2\varepsilon'_0 h_0}{3c^2(\Delta t)^2}\right]+\\
&\qquad\qquad+T_{N-1}(t_k)\left[\frac{1-\theta}{\mu_{N-2}h_{N-2}}+\frac{2\varepsilon'_{N-2}h_{N-2}}{3c^2(\Delta t)^2}\right]+\\
&\qquad\qquad\quad+\sum_{i=1}^{N-2}\Bigg\{T_{i-1}(t_{k-1})\left[\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}-\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}\right]+\\
&\qquad\qquad\qquad+T_i(t_{k-1})\left[\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}-\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}+\left(\frac{\mu_0\sigma_i}{2\Delta t}+\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]+\\
&\qquad\qquad\qquad\quad+T_{i+1}(t_{k-1})\left[\left(\frac{\mu_0\sigma_i}{2\Delta t}-\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]\Bigg\}+\\
&\qquad\qquad\qquad\qquad+T_0(t_{k-1})\left[\left(\frac{\mu_0\sigma_0}{2\Delta t}-\frac{\varepsilon'_0}{c^2(\Delta t)^2}\right)\frac{h_0}{3}\right]+\\
&\qquad\qquad\qquad\qquad\quad+T_{N-1}(t_{k-1})\left[\left(\frac{\mu_0\sigma_{N-2}}{2\Delta t}-\frac{\varepsilon'_{N-2}}{c^2(\Delta t)^2}\right)\frac{h_{N-2}}{3}\right]+\mathrm{BT}.
\end{aligned}
$$

The single equation above will be satisfied when for $i = 1$:

$$
\begin{aligned}
T_0(t_{k+1}) \left[-\frac{\theta}{\mu_0 h_0} + \left(\frac{\mu_0 \sigma_0}{2\Delta t} + \frac{\varepsilon'_0}{c^2(\Delta t)^2} \right) \frac{h_0}{3} \right] + T_1(t_{k+1}) \left[\frac{\theta}{\mu_0 h_0} + \left(\frac{\mu_0 \sigma_0}{2\Delta t} + \frac{\varepsilon'_0}{c^2(\Delta t)^2} \right) \frac{h_0}{3} \right] = \\
= T_0(t_k) \left[-\frac{1-\theta}{\mu_0 h_0} + \left(\frac{\mu_0 \sigma_0}{2\Delta t} + \frac{\varepsilon'_0}{c^2(\Delta t)^2} \right) \frac{h_0}{3} \right] + T_1(t_k) \left[\frac{\theta}{\mu_0 h_0} + \left(\frac{\mu_0 \sigma_0}{2\Delta t} + \frac{\varepsilon'_0}{c^2(\Delta t)^2} \right) \frac{h_0}{3} \right] + \\
+ T_0(t_{k-1}) \left[\left(\frac{\mu_0 \sigma_0}{2\Delta t} - \frac{\varepsilon'_0}{c^2(\Delta t)^2} \right) \frac{h_0}{3} \right] + T_1(t_{k-1}) \left[\left(\frac{\mu_0 \sigma_0}{2\Delta t} - \frac{\varepsilon'_0}{c^2(\Delta t)^2} \right) \frac{h_0}{3} \right] + (\mathrm{BT})_0,
\end{aligned}
$$

for $i = N - 1$,

$$
\begin{aligned}
T_{N-2}(t_{k+1}) \left[\frac{\theta}{\mu_{N-2} h_{N-2}} + \left(\frac{\mu_0 \sigma_{N-2}}{2\Delta t} + \frac{\varepsilon'_{N-2}}{c^2(\Delta t)^2} \right) \frac{h_{N-2}}{3} \right] + T_{N-1}(t_{k+1}) \Bigg[- \frac{\theta}{\mu_{N-2} h_{N-2}} + \Bigg(\frac{\mu_0 \sigma_{N-2}}{2\Delta t} + \\
+ \frac{\varepsilon'_{N-2}}{c^2(\Delta t)^2} \Bigg) \frac{h_{N-2}}{3} \Bigg] = \\
= T_{N-2}(t_k) \left[\frac{\theta - 1}{\mu_{N-2} h_{N-2}} + \frac{2\varepsilon'_{N-2} h_{N-2}}{3c^2(\Delta t)^2} \right] + T_{N-1}(t_k) \left[\frac{1-\theta}{\mu_{N-2} h_{N-2}} + \frac{2\varepsilon'_{N-2} h_{N-2}}{3c^2(\Delta t)^2} \right] + \\
+ T_{N-2}(t_{k-1}) \left[\left(\frac{\mu_0 \sigma_{N-2}}{2\Delta t} - \frac{\varepsilon'_{N-2}}{c^2(\Delta t)^2} \right) \frac{h_{N-2}}{3} \right] + \\
+ T_{N-1}(t_{k-1}) \left[\left(\frac{\mu_0 \sigma_{N-2}}{2\Delta t} - \frac{\varepsilon'_{N-2}}{c^2(\Delta t)^2} \right) \frac{h_{N-2}}{3} \right] + (\mathrm{BT})_{N-1},
\end{aligned}
$$

and for $i \in [1, N-2]$,

$$
\begin{aligned}
&T_{i-1}(t_{k+1})\left[\frac{\theta}{\mu_{i-1}h_{i-1}}+\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}+\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}\right]+\\
&\quad+T_i(t_{k+1})\left[-\theta\left(\frac{1}{\mu_{i-1}h_{i-1}}+\frac{1}{\mu_i h_i}\right)+\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}+\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}+\left(\frac{\mu_0\sigma_i}{2\Delta t}+\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]+\\
&\quad\quad+T_{i+1}(t_{k+1})\left[\frac{\theta}{\mu_i h_i}+\left(\frac{\mu_0\sigma_i}{2\Delta t}+\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]=\\
&=T_{i-1}(t_k)\left[\frac{\theta-1}{\mu_{i-1}h_{i-1}}+\frac{2\varepsilon'_{i-1}h_{i-1}}{3c^2(\Delta t)^2}\right]+\\
&\quad+T_i(t_k)\left[(1-\theta)\left(\frac{1}{\mu_{i-1}h_{i-1}}+\frac{1}{\mu_i h_i}\right)+\frac{2\varepsilon'_{i-1}h_{i-1}}{3c^2(\Delta t)^2}+\frac{2\varepsilon'_i h_i}{3c^2(\Delta t)^2}\right]+\\
&\quad\quad+T_{i+1}(t_k)\left[\frac{\theta-1}{\mu_i h_i}+\frac{2\varepsilon'_i h_i}{3c^2(\Delta t)^2}\right]+\\
&\quad\quad\quad+T_{i-1}(t_{k-1})\left[\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}-\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}\right]+\\
&\quad\quad\quad\quad+T_i(t_{k-1})\left[\left(\frac{\mu_0\sigma_{i-1}}{2\Delta t}-\frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3}+\left(\frac{\mu_0\sigma_i}{2\Delta t}+\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right]+\\
&\quad\quad\quad\quad\quad+T_{i+1}(t_{k-1})\left[\left(\frac{\mu_0\sigma_i}{2\Delta t}-\frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3}\right].
\end{aligned}
$$

Matrix Formulation

The previous system of equations can be represented as a single matrix equation:

$$\begin{bmatrix} a_0^2 & a_0^3 & 0 & \dots & \dots & 0 \\ a_1^1 & a_1^2 & a_1^3 & 0 & \dots & \vdots \\ 0 & a_2^1 & a_2^2 & a_2^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & a_{N-2}^1 & a_{N-2}^2 & a_{N-2}^3 \\ 0 & \dots & \dots & 0 & a_{N-1}^1 & a_{N-1}^2 \end{bmatrix} \begin{bmatrix} T_0(t_{k+1}) \\ T_1(t_{k+1}) \\ \vdots \\ T_i(t_{k+1}) \\ \vdots \\ T_{N-1}(t_{k+1}) \end{bmatrix} = \begin{bmatrix} b_0^2 & b_0^3 & 0 & \dots & \dots & 0 \\ b_1^1 & b_1^2 & b_1^3 & 0 & \dots & \vdots \\ 0 & b_2^1 & b_2^2 & b_2^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & b_{N-2}^1 & b_{N-2}^2 & b_{N-2}^3 \\ 0 & \dots & \dots & 0 & b_{N-1}^1 & b_{N-1}^2 \end{bmatrix} \begin{bmatrix} T_0(t_k) \\ T_1(t_k) \\ \vdots \\ T_i(t_k) \\ \vdots \\ T_{N-1}(t_k) \end{bmatrix} +$$

$$+ \begin{bmatrix} c_0^2 & c_0^3 & 0 & \dots & \dots & 0 \\ c_1^1 & c_1^2 & c_1^3 & 0 & \dots & \vdots \\ 0 & c_2^1 & c_2^2 & c_2^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & c_{N-2}^1 & c_{N-2}^2 & c_{N-2}^3 \\ 0 & \dots & \dots & 0 & c_{N-1}^1 & c_{N-1}^2 \end{bmatrix} \begin{bmatrix} T_0(t_{k-1}) \\ T_1(t_{k-1}) \\ \vdots \\ T_i(t_{k-1}) \\ \vdots \\ T_{N-1}(t_{k-1}) \end{bmatrix} + \begin{bmatrix} (\mathrm{BT})_0 \\ 0 \\ \vdots \\ 0 \\ (\mathrm{BT})_{N-1} \end{bmatrix},$$

where for $i \in [1, N-2]$,

$$a_i^1 := -\frac{\theta}{\mu_{i-1}h_{i-1}} + \left(\frac{\mu_0\sigma_{i-1}}{2\Delta t} + \frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3},$$

$$a_i^2 := -\theta\left(\frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i}\right) + \left(\frac{\mu_0\sigma_{i-1}}{2\Delta t} + \frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3} + \left(\frac{\mu_0\sigma_i}{2\Delta t} + \frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3},$$

$$a_i^3 := \frac{\theta}{\mu_i h_i} + \left(\frac{\mu_0\sigma_i}{2\Delta t} + \frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3},$$

$$b_i^1 := \frac{\theta - 1}{\mu_{i-1}h_{i-1}} + \frac{2\varepsilon'_{i-1}h_{i-1}}{3c^2(\Delta t)^2},$$

$$b_i^2 := (1-\theta)\left(\frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i}\right) + \frac{2\varepsilon'_{i-1}h_{i-1}}{3c^2(\Delta t)^2} + \frac{2\varepsilon'_i h_i}{3c^2(\Delta t)^2},$$

$$b_i^3 := \frac{\theta - 1}{\mu_i h_i} + \frac{2\varepsilon'_i h_i}{3c^2(\Delta t)^2},$$

$$c_i^1 := \left(\frac{\mu_0\sigma_{i-1}}{2\Delta t} - \frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3},$$

$$c_i^2 := \left(\frac{\mu_0\sigma_{i-1}}{2\Delta t} - \frac{\varepsilon'_{i-1}}{c^2(\Delta t)^2}\right)\frac{h_{i-1}}{3} + \left(\frac{\mu_0\sigma_i}{2\Delta t} + \frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3},$$

$$c_i^3 := \left(\frac{\mu_0\sigma_i}{2\Delta t} - \frac{\varepsilon'_i}{c^2(\Delta t)^2}\right)\frac{h_i}{3},$$

in case $i = 0$,

$$a_0^2 := -\frac{\theta}{\mu_0 h_0} + \left(\frac{\mu_0\sigma_0}{2\Delta t} + \frac{\varepsilon_0'}{c^2(\Delta t)^2}\right)\frac{h_0}{3}, \qquad a_0^3 := \frac{\theta}{\mu_0 h_0} + \left(\frac{\mu_0\sigma_0}{2\Delta t} + \frac{\varepsilon_0'}{c^2(\Delta t)^2}\right)\frac{h_0}{3},$$

$$b_0^2 := -\frac{1-\theta}{\mu_0 h_0} + \left(\frac{\mu_0\sigma_0}{2\Delta t} + \frac{\varepsilon_0'}{c^2(\Delta t)^2}\right)\frac{h_0}{3}, \qquad b_0^3 := \frac{\theta}{\mu_0 h_0} + \left(\frac{\mu_0\sigma_0}{2\Delta t} + \frac{\varepsilon_0'}{c^2(\Delta t)^2}\right)\frac{h_0}{3},$$

$$c_0^2 := \left(\frac{\mu_0\sigma_0}{2\Delta t} - \frac{\varepsilon_0'}{c^2(\Delta t)^2}\right)\frac{h_0}{3}, \qquad c_0^3 := \left(\frac{\mu_0\sigma_0}{2\Delta t} - \frac{\varepsilon_0'}{c^2(\Delta t)^2}\right)\frac{h_0}{3},$$

and in case $i = N-1$,

$$a_{N-1}^1 := \frac{\theta}{\mu_{N-2}h_{N-2}} + \left(\frac{\mu_0\sigma_{N-2}}{2\Delta t} + \frac{\varepsilon_{N-2}'}{c^2(\Delta t)^2}\right)\frac{h_{N-2}}{3},$$

$$a_{N-1}^2 := -\frac{\theta}{\mu_{N-2}h_{N-2}} + \left(\frac{\mu_0\sigma_{N-2}}{2\Delta t} + \frac{\varepsilon_{N-2}'}{c^2(\Delta t)^2}\right)\frac{h_{N-2}}{3},$$

$$b_{N-1}^1 := \frac{\theta-1}{\mu_{N-2}h_{N-2}} + \frac{2\varepsilon_{N-2}'h_{N-2}}{3c^2(\Delta t)^2}, \qquad b_{N-1}^2 := \frac{1-\theta}{\mu_{N-2}h_{N-2}} + \frac{2\varepsilon_{N-2}'h_{N-2}}{3c^2(\Delta t)^2},$$

$$c_{N-1}^1 := \left(\frac{\mu_0\sigma_{N-2}}{2\Delta t} - \frac{\varepsilon_{N-2}'}{c^2(\Delta t)^2}\right)\frac{h_{N-2}}{3}, \qquad c_{N-1}^2 := \left(\frac{\mu_0\sigma_{N-2}}{2\Delta t} - \frac{\varepsilon_{N-2}'}{c^2(\Delta t)^2}\right)\frac{h_{N-2}}{3}.$$

Boundary Conditions

Now we consider the boundary conditions, which as discussed in Section 2.4 are the inhomogeneous Dirichlet condition at the left-hand endpoint (*i.e.*, at $z = 0$), and the absorbing boundary condition at the right-hand endpoint (*i.e.*, at $z = L$).

Note that $E_j(0) = 0$ for all j except $j = 0$, and that $E_0(0) = 1$; in this case, then, the boundary condition at $a = 0$ implies

$$\sum_{j=0}^{N-1} T_j(t)E_j(0) = T_0(t) = \frac{2}{a}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}.$$

We thus replace the first row of the left-hand side matrix by the first row of the $N-1\times N-1$ identity matrix, replace the first row of each right-hand side matrix by the $1\times N-1$ zero vector, and replace the first entry of the boundary term vector by the value $\frac{2}{a}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}$. That is, we let $a_0^2 = 1$, $a_0^3 = b_0^2 = b_0^3 = c_0^2 = c_0^3 = 0$, and $(\mathrm{BT})_0 = \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}$.

For the boundary condition at $z = L$, we have

$$\left.\frac{\partial E}{\partial z}\right|_{z=L} = -\frac{1}{c}\left.\frac{\partial E}{\partial t}\right|_{z=L},$$

and with our assumption of separability and our choice of hat functions so that $E_{N-1}(L) = 1$, this implies

$$\left.\frac{\mathrm{d}E_{N-1}}{\mathrm{d}z}\right|_{z=L} T_{N-1}(t) = -\frac{1}{c}\frac{\mathrm{d}T_{N-1}}{\mathrm{d}t},$$

and with our choice of the centered difference method to approximate the time derivative, we obtain

$$\left.\frac{\mathrm{d}E_{N-1}}{\mathrm{d}z}\right|_{z=L} T_{N-1}(t) = -\frac{1}{2c\Delta t}\left(T_{N-1}(t_{k+1}) - T_{N-1}(t_{k-1})\right),$$

so that our boundary term becomes

$$\mathrm{BT}_{N-1} = -\frac{T_{N-1}(t)}{\mu_{N-2}(t)}\left.\frac{\mathrm{d}E_{N-1}}{\mathrm{d}z}\right|_{z=L} = \frac{1}{2c\mu_{N-2}\Delta t}\left(T_{N-1}(t_{k+1}) - T_{N-1}(t_{k-1})\right).$$

We can implement this in our matrix equation by subtracting the quantity $1/(2c\mu_{N-2}\Delta t)$ from both a^2_{N-1} and c_2^{N-1}:

$$a^2_{N-1} \leftarrow a^2_{N-1} - \frac{1}{2c\mu_{N-2}\Delta t}, \qquad c^2_{N-1} \leftarrow c^2_{N-1} - \frac{1}{2c\mu_{N-2}\Delta t}.$$

The resulting system is, finally, the following:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ a_1^1 & a_1^2 & a_1^3 & 0 & \dots & \vdots \\ 0 & a_2^1 & a_2^2 & a_2^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & a_{N-2}^1 & a_{N-2}^2 & a_{N-2}^3 \\ 0 & \dots & \dots & 0 & a_{N-1}^1 & a_{N-1}^2 \end{bmatrix} \begin{bmatrix} T_0(t_{k+1}) \\ T_1(t_{k+1}) \\ \vdots \\ T_i(t_{k+1}) \\ \vdots \\ T_{N-1}(t_{k+1}) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & \dots & \dots & 0 \\ b_1^1 & b_1^2 & b_1^3 & 0 & \dots & \vdots \\ 0 & b_2^1 & b_2^2 & b_2^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & b_{N-2}^1 & b_{N-2}^2 & b_{N-2}^3 \\ 0 & \dots & \dots & 0 & b_{N-1}^1 & b_{N-1}^2 \end{bmatrix} \begin{bmatrix} T_0(t_k) \\ T_1(t_k) \\ \vdots \\ T_i(t_k) \\ \vdots \\ T_{N-1}(t_k) \end{bmatrix} +$$

$$+ \begin{bmatrix} 0 & 0 & 0 & \dots & \dots & 0 \\ c_1^1 & c_1^2 & c_1^3 & 0 & \dots & \vdots \\ 0 & c_2^1 & c_2^2 & c_2^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & c_{N-2}^1 & c_{N-2}^2 & c_{N-2}^3 \\ 0 & \dots & \dots & 0 & c_{N-1}^1 & c_{N-1}^2 \end{bmatrix} \begin{bmatrix} T_0(t_{k-1}) \\ T_1(t_{k-1}) \\ \vdots \\ T_i(t_{k-1}) \\ \vdots \\ T_{N-1}(t_{k-1}) \end{bmatrix} + \begin{bmatrix} \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)} \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix},$$

where the a_i^ℓ, b_i^ℓ, and c_1^ℓ, for $\ell \in \{1,2,3\}$ and $i \in [1, N-2]$, are the same ones given above, keeping in mind our recent modification to a^2_{N-1} and c^2_{N-1}.

This method may be used to solve for the electric field at each time step, and its `MATLAB` implementation can be found in Appendix D.3.

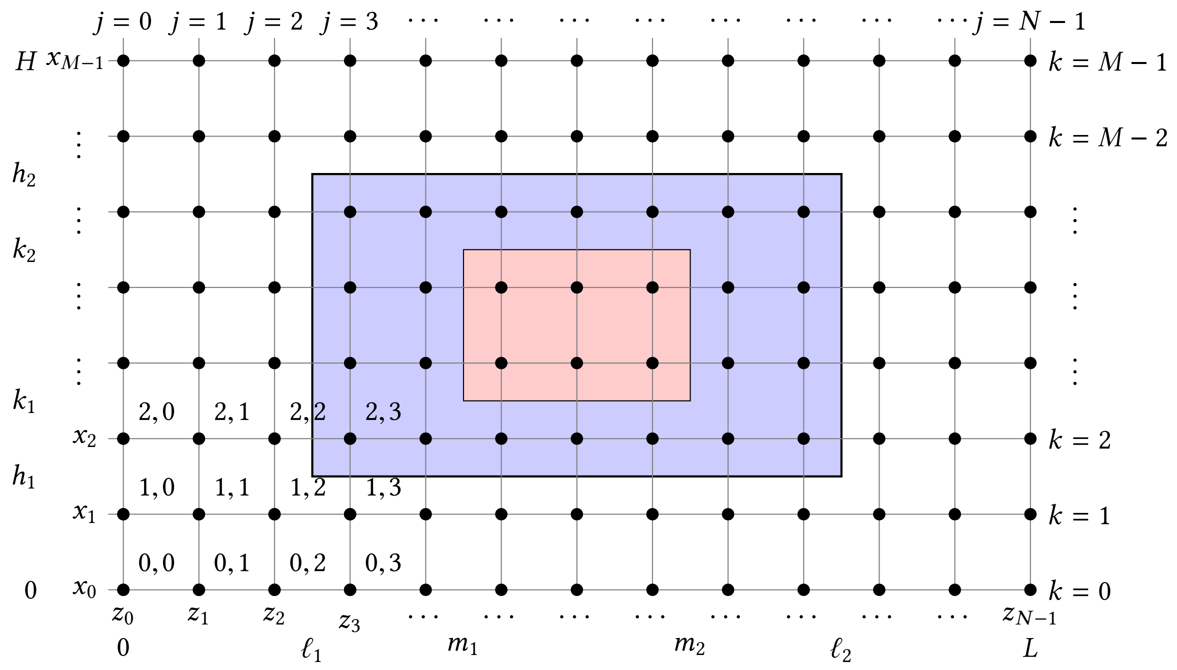


Figure 7.7: Discretization of the spatial domain for the finite difference solution of the two-dimensional wave equation. Area in blue is occupied by insulation, and area in red is occupied by material. Boundaries of insulation and material do not necessarily fall on the numbered nodes.

7.2 Techniques for Solving the Two-Dimensional Wave Equation

This section outlines a finite difference technique for solving the two-dimensional wave equation given in Problem 4. The technique used is a θ-method, where the spatial derivatives employ the parameters $\theta, \phi \in [0, 1]$, whose values dictate whether the scheme is explicit or explicit. This removes the need to write several different codes for these scenarios.

Finite Difference Methods

For the two-dimensional case, we use a time-marching scheme with constant time steps given by $\Delta t_{\vec{E}}$, denoted here Δt, as in the one-dimensional case. The domain is discretized into a spatial grid, seen in Figure 7.7, whose components are not necessarily of uniform length or height, where the z-dimension contains N many nodes and $N-1$ many intervals, and the x-dimension contains M many nodes and $M-1$ many intervals. We are not concerned with whether the boundary of the insulation or the load material is located at a node, or between nodes (the latter is typically the case with our solver).

Let $E^n_{j,k} := E(x_k, z_j, t_n)$, for $j \in [0, N-1] \cap \mathbb{N}$, $k \in [0, M-1] \cap \mathbb{N}$, and $n \in \{0\} \cup \mathbb{N}$. Consider the θ-scheme that uses difference approximations

$$\left.\frac{\partial E}{\partial t}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx \frac{E_{j,k}^{n+1}-E_{j,k}^{n-1}}{2\Delta t},$$

$$\left.\frac{\partial^2 E}{\partial t^2}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx \frac{E_{j,k}^{n-1}-2E_{j,k}^{n}+E_{j,k}^{n+1}}{(\Delta t)^2},$$

$$\left.\frac{\partial E}{\partial z^2}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx (1-\theta)\frac{E_{j-1,k}^{n}-2E_{j,k}^{n}+E_{j+1,k}^{n}}{(z_{j+1}-z_j)(z_j-z_{j-1})}+\theta\frac{E_{j-1,k}^{n+1}-2E_{j,k}^{n+1}+E_{j+1,k}^{n+1}}{(z_{j+1}-z_j)(z_j-z_{j-1})}, \quad \text{and}$$

$$\left.\frac{\partial E}{\partial x^2}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx (1-\phi)\frac{E_{j,k-1}^{n}-2E_{j,k}^{n}+E_{j,k+1}^{n}}{(x_{k+1}-x_k)(x_k-x_{k-1})}+\phi\frac{E_{j,k-1}^{n+1}-2E_{j,k}^{n+1}+E_{j,k+1}^{n+1}}{(x_{k+1}-x_k)(x_k-x_{k-1})},$$

where $\theta,\phi \in [0,1]$, $j \in [1,N-2]\cap\mathbb{N}$, $k \in [1,M-2]\cap\mathbb{N}$, and $n \in \mathbb{N}$. Here, the fully explicit method corresponds to taking $\theta=\phi=0$, the fully implicit method corresponds to taking $\theta=\phi=1$, and the Crank-Nicolson scheme corresponds to $\theta=\phi=0.5$.

Substituting the difference approximations into the governing equation in Equation 2.65, we obtain

$$\begin{aligned}&\left((1-\phi)\frac{E_{j,k-1}^{n}-2E_{j,k}^{n}+E_{j,k+1}^{n}}{(x_{k+1}-x_k)(x_k-x_{k-1})}+\phi\frac{E_{j,k-1}^{n+1}-2E_{j,k}^{n+1}+E_{j,k+1}^{n+1}}{(x_{k+1}-x_k)(x_k-x_{k-1})}\right)+\\ &\quad+\left((1-\theta)\frac{E_{j-1,k}^{n}-2E_{j,k}^{n}+E_{j+1,k}^{n}}{(z_{j+1}-z_j)(z_j-z_{j-1})}+\theta\frac{E_{j-1,k}^{n+1}-2E_{j,k}^{n+1}+E_{j+1,k}^{n+1}}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right)-\\ &\quad-\mu\varepsilon'\left(\frac{E_{j,k}^{n+1}-2E_{j,k}^{n}+E_{j,k}^{n}}{(\Delta t)^2}\right)-\mu\sigma\left(\frac{E_{j,k}^{n+1}-E_{j,k}^{n-1}}{2\Delta t}\right)=0\end{aligned}$$

where, as in the one-dimensional cases, we have used the dimensional representations ε' and μ of absolute permittivity and permeability, and the fact that $\frac{1}{c^2}=\varepsilon_0'\mu_0$. Keeping the terms at time level

$n+1$ on the left-hand side and moving all others to the right-hand side, we obtain

$$\begin{aligned}
&\left(\frac{2\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}+\frac{2\phi}{(x_{k+1}-x_k)(x_k-x_{k-1})}+\frac{\mu\sigma}{2\Delta t}+\frac{\mu\varepsilon'}{(\Delta t)^2}\right)E^{n+1}_{j,k}-\\
&-\frac{\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}E^{n+1}_{j-1,k}-\frac{\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}E^{n+1}_{j+1,k}-\frac{\phi}{(x_{k+1}-x_k)(x_k-x_{k-1})}E^{n+1}_{j,k-1}-\\
&-\frac{\phi}{(x_{k+1}-x_k)(x_k-x_{k-1})}E^{n+1}_{j,k+1}=\left(-\frac{\mu\varepsilon'}{(\Delta t)^2}+\frac{\mu\sigma}{2\Delta t}\right)E^{n-1}_{j,k}+\\
&+\left(-\frac{2(1-\phi)}{(x_{k+1}-x_k)(x_k-x_{k-1})}-\frac{2(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}+\frac{2\mu\varepsilon'}{(\Delta t)^2}\right)E^n_{j,k}+\\
&+\frac{(1-\phi)}{(x_{k+1}-x_k)(x_k-x_{k-1})}E^n_{j,k-1}+\frac{(1-\phi)}{(x_{k+1}-x_k)(x_k-x_{k-1})}E^n_{j,k+1}+\\
&+\frac{(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}E^n_{j-1,k}+\frac{(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}E^n_{j+1,k}
\end{aligned}$$

Using the abbreviations

$$(\Delta z)^2_j := (z_{j+1}-z_j)(z_j-z_{j-1}),\qquad (\Delta x)^2_k := (x_{k+1}-x_k)(x_k-x_{k-1}),$$
$$s_j := \frac{(\Delta t)^2}{(\Delta z)^2_j},\qquad r_k := \frac{(\Delta t)^2}{(\Delta x)^2_k},$$

the difference approximation becomes

$$\begin{aligned}
&\left(2\theta s_j+2\phi r_k+\mu\varepsilon'+\frac{\mu\sigma\Delta t}{2}\right)E^{n+1}_{j,k}-\theta s_j E^{n+1}_{j-1,k}-\theta s_j E^{n+1}_{j+1,k}-\phi r_k E^{n+1}_{j,k-1}-\phi r_k E^{n+1}_{j,k+1}=\\
&=\left(-\mu\varepsilon'+\frac{\mu\sigma\Delta t}{2}\right)E^{n-1}_{j,k}+\Big(-2(1-\phi)r_k-2(1-\theta)s_j+2\mu\varepsilon'\Big)E^n_{j,k}+\\
&+(1-\phi)r_k E^n_{j,k-1}+(1-\phi)r_k E^n_{j,k+1}+(1-\theta)s_j E^n_{j-1,k}+(1-\theta)s_j E^n_{j+1,k}
\end{aligned}\tag{7.21}$$

To implement the boundary condition in Equation 2.75 on the $z=0$ boundary, we set

$$E^n_{0,k}=\frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)},\tag{7.22}$$

for $k\in[0,M-1]\cap\mathbb{N}$ and for all $n\in\mathbb{N}$.

To implement the boundary conditions in Equation 2.75 on the $x=0$ and $x=H$ boundaries, we set

$$E^n_{j,0}=0,\qquad E^n_{j,H}=0\tag{7.23}$$

for $j\in[1,N-2]\cap\mathbb{N}$ and for all $n\in\mathbb{N}$.

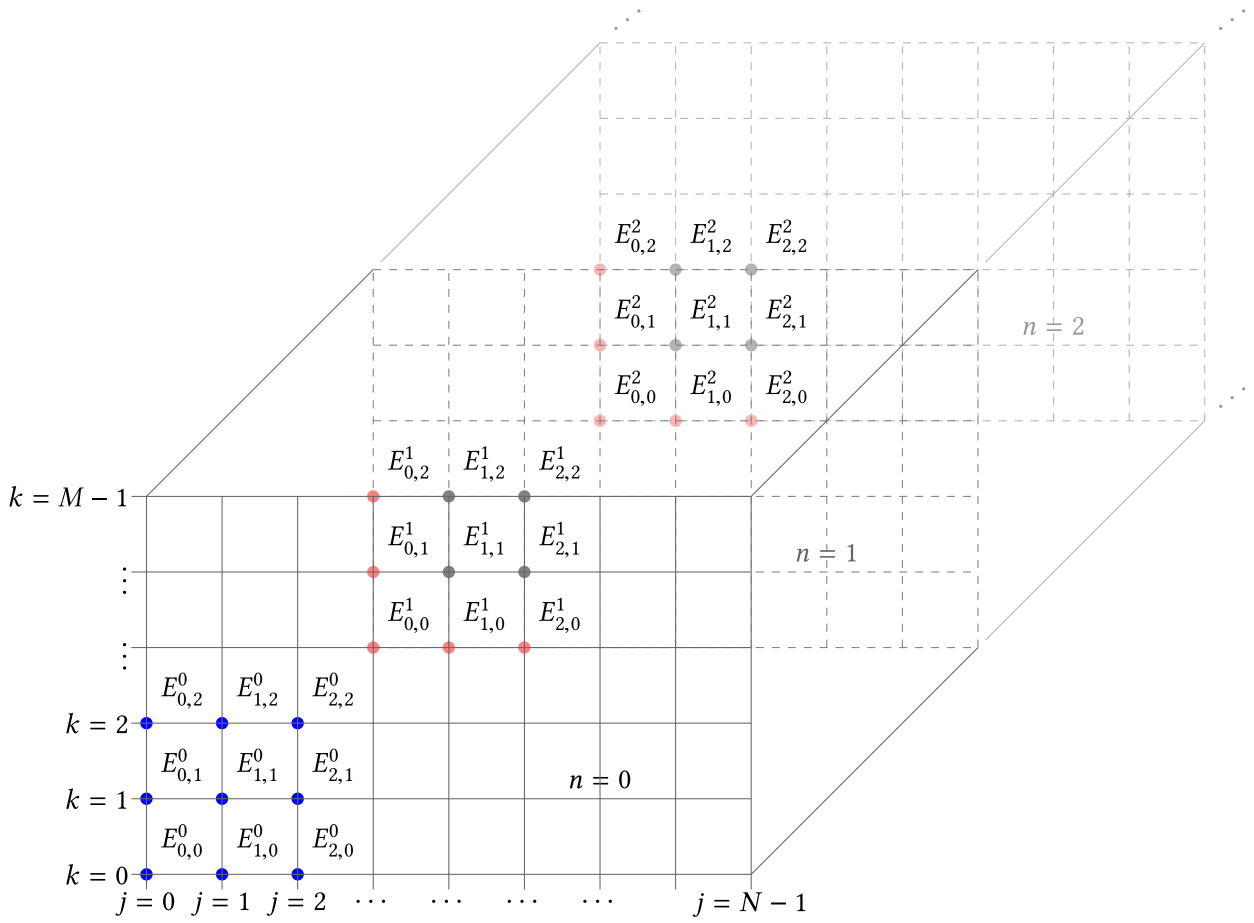

$E^2_{0,2}$ $E^2_{1,2}$ $E^2_{2,2}$
$E^2_{0,1}$ $E^2_{1,1}$ $E^2_{2,1}$
$E^2_{0,0}$ $E^2_{1,0}$ $E^2_{2,0}$
$n = 2$
$E^1_{0,2}$ $E^1_{1,2}$ $E^1_{2,2}$
$E^1_{0,1}$ $E^1_{1,1}$ $E^1_{2,1}$
$E^1_{0,0}$ $E^1_{1,0}$ $E^1_{2,0}$
$n = 1$
$k = M - 1$
$E^0_{0,2}$ $E^0_{1,2}$ $E^0_{2,2}$
$E^0_{0,1}$ $E^0_{1,1}$ $E^0_{2,1}$
$E^0_{0,0}$ $E^0_{1,0}$ $E^0_{2,0}$
$n = 0$
$k = 2$
$k = 1$
$k = 0$
$j = 0$ $j = 1$ $j = 2$ $\cdots$ $\cdots$ $\cdots$ $\cdots$ $j = N - 1$

Figure 7.8: Computational grids representing the solution space of the wave equation with two spatial dimensions and one time dimension. Here, $j \in [0, N-1] \cap \mathbb{N}$ and $k \in [0, M-1] \cap \mathbb{N}$ represent the position within the spatial domain, and $n \in \{0\} \cup \mathbb{N}$ represents the time step. The blue-colored nodes represent those where the solution is given by the initial condition in Equation 7.28, and the red-colored nodes represent those whose solution is given by the boundary conditions in Equations 7.22, 7.23, 7.25, and 7.24, and 7.26, while the solution at the black-colored nodes is given by Equation 7.21.

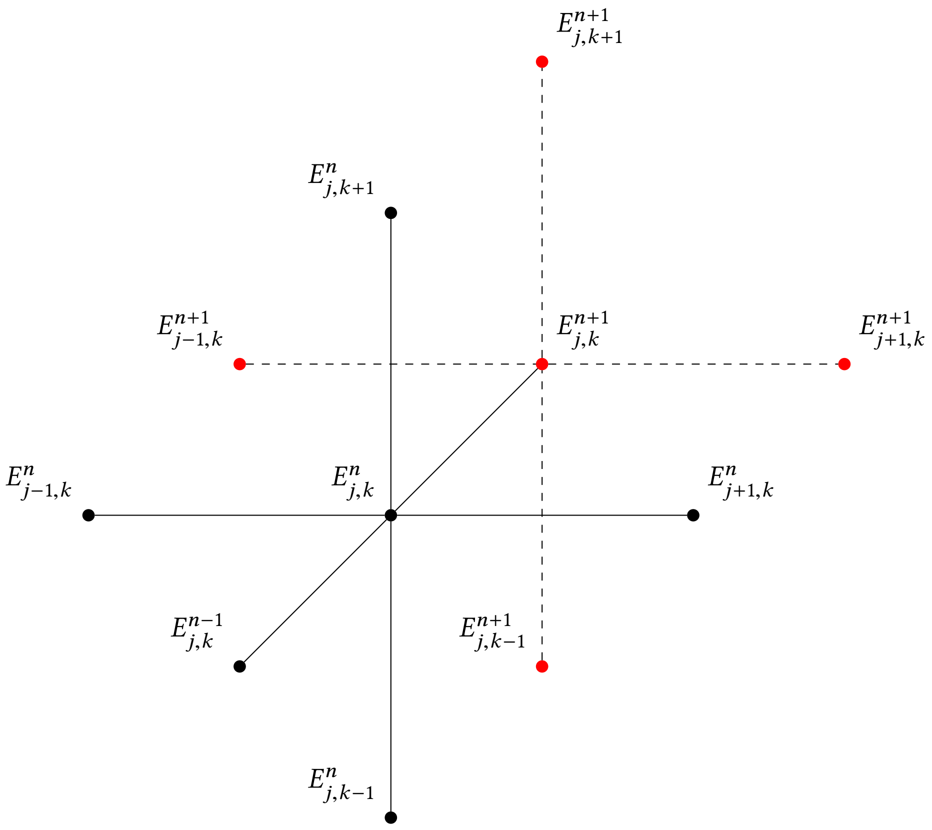


Figure 7.9: Computational stencil of θ-scheme for solving the two-dimensional wave equation. Here, $j \in [1, N-2] \cap \mathbb{N}$ and $k \in [1, M-2] \cap \mathbb{N}$ represent the position within the spatial domain, and $n \in \mathbb{N}$ represents the current time step. The nodes in black are ones at which the solution E is known, and the ones in red may be solved for with knowledge of the ones in black.

To implement the absorbing boundary condition in Equation 2.63 on the $z = L$ boundary, as done in [142] we set

$$\begin{aligned}\left(\frac{1}{\Delta z_{N-1}\Delta t} + \frac{1}{c(\Delta t)^2}\right) E^{n+1}_{N-1,k} - \frac{1}{\Delta z_{N-1}\Delta t} E^{n+1}_{N-2,k} = -\frac{1}{c(\Delta t)^2} E^{n-1}_{N-1,k} + \\ + \left(\frac{1}{\Delta z_{N-1}\Delta t} + \frac{2}{c(\Delta t)^2} - \frac{c}{(\Delta x_k)^2}\right) E^{n}_{N-1,k} - \frac{1}{\Delta z_{N-1}\Delta t} E^{n}_{N-2,k} + \frac{c}{2(\Delta x_k)^2} E^{n}_{N,k-1} + \frac{c}{2(\Delta x_k)^2} E^{n}_{N,k+1},\end{aligned} \tag{7.24}$$

j	k	Equation
0	$0,\ldots,M-1$	7.22
$1,\ldots,N-2$	0	7.23
$1,\ldots,N-2$	$1,\ldots,M-2$	7.21
$1,\ldots,N-2$	$M-1$	7.23
$N-1$	0	7.25
$N-1$	$1,\ldots,M-2$	7.24
$N-1$	$M-1$	7.26

Table 7.1: Description of the organization of the linear system for the finite difference approximation of the two-dimensional wave equation.

for $k \in [1, M-2] \cap \mathbb{N}$ and for all $n \in \mathbb{N}$, and at $k = 0$,

$$\begin{aligned}\left(\frac{1}{\Delta z_{N-1}\Delta t} + \frac{1}{c(\Delta t)^2}\right) E^{n+1}_{N-1,0} = -\frac{1}{c(\Delta t)^2} E^{n-1}_{N-1,0} + \\ + \left(\frac{1}{\Delta z_{N-1}\Delta t} + \frac{2}{c(\Delta t)^2} - \frac{c}{(\Delta x_0)^2}\right) E^n_{N-1,0} - \frac{1}{\Delta z_{N-1}\Delta t} E^n_{N-2,0} + \frac{c}{2(\Delta x_0)^2} E^n_{N,1},\end{aligned} \tag{7.25}$$

and, finally, at $k = M-1$,

$$\begin{aligned}\left(\frac{1}{\Delta z_{N-1}\Delta t} + \frac{1}{c(\Delta t)^2}\right) E^{n+1}_{N-1,M-1} = -\frac{1}{c(\Delta t)^2} E^{n-1}_{N-1,M-1} + \\ + \left(\frac{1}{\Delta z_{N-1}\Delta t} + \frac{2}{c(\Delta t)^2} - \frac{c}{(\Delta x_{M-2})^2}\right) E^n_{N-1,M-1} - \frac{1}{\Delta z_{N-1}\Delta t} E^n_{N-2,M-1} + \frac{c}{2(\Delta x_{M-2})^2} E^n_{N,M-2}.\end{aligned} \tag{7.26}$$

A linear system may be established using the equations described above, with Table 7.1 summarizing which equation corresponds to each coordinate pair (j, k). As is clear, the system is one of $N \times M$ many equations in $N \times M$ many unknowns, and so we establish the matrix equation for its solution at each time step:

$$A\vec{E}^{n+1} = B\vec{E}^n + \vec{c}\vec{E}^{n-1}, \tag{7.27}$$

where the vectors $\vec{E}^p$ and $\vec{c}$, and the matrices A and B, are defined in Appendix A, and where the initial vector $\vec{E}^0$ is given by

$$\vec{E}^0 := \left\langle \underbrace{E_{\text{inc}}, \ldots, E_{\text{inc}}}_{N}, 0, \ldots, 0 \right\rangle. \tag{7.28}$$

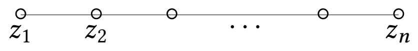


Figure 7.10: Discretization of the one-dimensional domain on which the finite element solution for the Helmholtz equation is described.

7.3 Techniques for Solving the One-Dimensional Helmholtz Equation

Weak Formulation of the Governing Equation

We formulate a single matrix equation that can be solved once to obtain the spatial function $\mathcal{E}(z)$, which can subsequently be multiplied by $e^{i\omega t}$ for each time step t to obtain a video of the transient solution.

Spatial Discretization

We use n many nodes, not necessarily evenly spaced, to discretize the domain into $n-1$ many elements, as shown in Figure 7.10.

On this domain, we assume that our solutions $\mathcal{E}(z)$ take the form $\mathcal{E}(z) := \sum_{j=1}^{n} T_j E_j(z)$ for some constants T_j and functions $E_j(z)$, and we plug this into Equation (7.32), multiply by the test function $W(z) := \sum_{i=1}^{n} W_i(z)$, where for $i \in [1,n] \cap \mathbb{N}$, $W_i(z)$ is some known function, and integrate over the domain $[z_1, z_n]$:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} T_j \left\{ \int_{z_1}^{z_n} W_i(z) \frac{\mathrm{d}}{\mathrm{d}x}\left(\frac{1}{\mu_{\text{rel}}(z)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\right)\mathrm{d}z - \left(\frac{\omega^2}{c^2}\right)\int_{z_1}^{z_n} \mu_{\text{rel}}(z)\varepsilon_{\text{rel}}'(z)W_i(z)E_j(z)\mathrm{d}z \right\} = 0.$$

We integrate the second spatial derivative term by parts:

$$\int_{z_1}^{z_n} W_i(z)\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{\mu_{\text{rel}}(z)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\right)\mathrm{d}x = \frac{W_i(x)}{\mu(x)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\bigg|_{z_1}^{z_n} - \int_{z_1}^{z_n}\frac{1}{\mu_{\text{rel}}(z)}\frac{\mathrm{d}W_i}{\mathrm{d}z}\frac{\mathrm{d}E_j}{\mathrm{d}z}\mathrm{d}z,$$

and we move the first term of the result (henceforth called the "boundary term") to the right-hand side of the governing equation to obtain the form:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} T_j\left\{\int_{z_1}^{z_n}\frac{1}{\mu_{\text{rel}}(z)}\frac{\mathrm{d}W_i}{\mathrm{d}z}\frac{\mathrm{d}E_j}{\mathrm{d}z}\mathrm{d}z + \left(\frac{\omega^2}{c^2}\right)\int_{z_1}^{z_n}\mu_{\text{rel}}(z)\varepsilon(z)W_i(x)E_j(z)\mathrm{d}z\right\} = \sum_{i=1}^{n}\sum_{j=1}^{n} T_j \frac{W_i(z)}{\mu_{\text{rel}}(z)}\frac{\mathrm{d}E_j}{\mathrm{d}z}\bigg|_{z_1}^{z_n}. \tag{7.29}$$

For the functions E_j and W_i, we will use the same definitions as were given in Section 7.1 in Equations (7.17)–(7.19), namely applying the Galerkin formulation that dictates $W_i(z) = E_i(z)$ for each $i \in [1,n] \cap \mathbb{N}$, where $E_i(z)$ is the hat function centered at node i.

The derivatives of these function were also calculated in Section 7.1, and we may use the relevant integral calculations in that section to obtain the following values of the integrals $\int_{z_1}^{z_n} \frac{1}{\mu_{\text{rel}}(x)}\frac{\mathrm{d}W_i}{\mathrm{d}z}\frac{\mathrm{d}E_j}{\mathrm{d}z}\mathrm{d}z$ and $\int_{z_1}^{z_n} \mu_{\text{rel}}(z)\varepsilon_{\text{rel}}(z)W_iE_j\mathrm{d}z$ for values of i and j in the interval $[1,n] \cap \mathbb{N}$:

$$\int_{z_n}^{z_1} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{\mathrm{d}W_i}{\mathrm{d}z} \frac{\mathrm{d}E_j}{\mathrm{d}z} \mathrm{d}x = 0, \text{ if } |i-j| > 1,$$

$$\int_{z_1}^{z_n} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{\mathrm{d}W_i}{\mathrm{d}z} \frac{\mathrm{d}E_{i-1}}{\partial z} \mathrm{d}z = \int_{z_{i-1}}^{z_i} \frac{1}{\mu_{\mathrm{rel}}(z)} \left(\frac{-1}{h_{i-1}^2}\right) \mathrm{d}z = -\frac{1}{\mu_{i-1}h_{i-1}} \text{ for } i \in [2,n],$$

$$\int_{z_1}^{z_n} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{\mathrm{d}W_i}{\mathrm{d}z} \frac{\mathrm{d}E_i}{\mathrm{d}z} \mathrm{d}z = \int_{z_{i-1}}^{z_i} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{1}{h_{i-1}^2} \mathrm{d}z + \int_{z_i}^{z_{i+1}} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{1}{h_i^2} \mathrm{d}z = \frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i}, \text{ for } i \in [2, n-1],$$

$$\int_{z_1}^{z_n} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{\mathrm{d}W_1}{\mathrm{d}z} \frac{\mathrm{d}E_1}{\mathrm{d}z} \mathrm{d}z = \frac{1}{\mu_1 h_1}, \text{ and } \int_{z_1}^{z_n} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{\mathrm{d}W_n}{\mathrm{d}z} \frac{\mathrm{d}E_n}{\mathrm{d}z} \mathrm{d}z = \frac{1}{\mu_{n-1}h_{n-1}};$$

and

$$\int_{z_0}^{z_n} \frac{1}{\mu_{\mathrm{rel}}(z)} \frac{\mathrm{d}W_i}{\mathrm{d}z} \frac{\mathrm{d}E_{i+1}}{\mathrm{d}z} \mathrm{d}z = \int_{z_i}^{z_{i+1}} \frac{1}{\mu_{\mathrm{rel}}(z)} \left(\frac{-1}{h_i^2}\right) \mathrm{d}z = -\frac{1}{\mu_i h_i} \text{ for } i \in [1, n-1],$$

together with

$$\int_{z_1}^{z_n} \mu_{\mathrm{rel}}(z)\varepsilon_{\mathrm{rel}}(z) W_i E_j \mathrm{d}z = 0, \text{ if } |i-j| > 1,$$

and

$$\int_{z_1}^{z_n} \mu_{\mathrm{rel}}(z)\varepsilon_{\mathrm{rel}}(z) W_i E_{i-1} \mathrm{d}z = \mu_{i-1}\varepsilon_{i-1} \int_{z_{i-1}}^{z_i} \frac{(z_{i-1}-z)^2}{h_{i-1}^2} \mathrm{d}z = \mu_{i-1}\varepsilon_{i-1} \frac{(z-z_{i-1})^3}{3h_{i-1}^2}\bigg|_{z_{i-1}}^{z_i}$$
$$= \mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}, \text{ for } i \in [2,n],$$

$$\int_{z_1}^{z_n} \mu_{\mathrm{rel}}(z)\varepsilon_{\mathrm{rel}}(z) W_i E_i \mathrm{d}z = \mu_{i-1}\varepsilon_{i-1} \int_{z_{i-1}}^{z_i} \frac{(z-z_{i-1})^2}{h_{i-1}^2} \mathrm{d}z + \mu_i\varepsilon_i \int_{z_i}^{z_{i+1}} \frac{(z_{i+1}-z)^2}{h_i^2} \mathrm{d}z$$
$$= \mu_{i-1}\varepsilon_{i-1} \frac{(z-z_{i-1})^3}{3h_{i-1}^2}\bigg|_{z_{i-1}}^{z_i} + \mu_i\varepsilon_i \frac{(z-z_i)^3}{3h_i^2}\bigg|_{z_i}^{z_{i+1}} = \mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3} + \mu_i\varepsilon_i\frac{h_i}{3},$$

for $i \in [2, n-1]$. Also

$$\int_{z_1}^{z_n} \mu(z)\varepsilon(z) W_1 E_1 \mathrm{d}z = \mu_1\varepsilon_1\frac{h_1}{3} \text{ and } \int_{z_1}^{z_n} \mu_{\mathrm{rel}}(z)\varepsilon_{\mathrm{rel}}(z) W_n E_n \mathrm{d}z = \mu_{n-1}\varepsilon_{n-1}\frac{h_{n-1}}{3},$$

and

$$\int_{z_1}^{z_n} \mu_{\mathrm{rel}}(z)\varepsilon_{\mathrm{rel}}(z) W_i E_{i+1} \mathrm{d}z = \mu_i\varepsilon_i \int_{z_i}^{z_{i-1}} \frac{(z-z_i)^2}{h_i^2} \mathrm{d}z = \mu_i\varepsilon_i \frac{(z-z_i)^3}{3h_i^2}\bigg|_{z_i}^{z_{i+1}} = \mu_i\varepsilon_i\frac{h_i}{3}, \text{ for } i \in [1, n-1].$$

Using all of the integral values we have just computed in the weak form of the governing equation (7.29), we obtain

$$\sum_{i=2}^{n-1}\left\{T_{i-1}\left[-\frac{1}{\mu_{i-1}h_{i-1}}+\mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}\right]+T_i\left[\frac{1}{\mu_{i-1}h_{i-1}}+\frac{1}{\mu_i h_i}+\mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}+\mu_i\varepsilon_i\frac{h_i}{3}\right]+\right.$$
$$\left.+T_{i+1}\left[-\frac{1}{\mu_i h_i}+\mu_i\varepsilon_i\frac{h_i}{3}\right]\right\}+T_1\left[\frac{1}{\mu_1 h_1}+\frac{\mu_1\varepsilon_1 h_1}{3}\right]+T_n\left[\frac{1}{\mu_{n-1}h_{n-1}}+\frac{\mu_{n-1}\varepsilon_{n-1}h_{n-1}}{3}\right]=$$
$$=\sum_{i=2}^{n-1}\left\{\frac{W_i(z)}{\mu_i}\left[T_{i-1}\frac{\mathrm{d}E_{i-1}}{\mathrm{d}z}+T_i\frac{\mathrm{d}E_i}{\mathrm{d}z}+T_{i+1}\frac{\mathrm{d}E_{i+1}}{\mathrm{d}z}\right]\right\}\Bigg|_{z_1}^{z_n}+T_1\frac{W_1(z)}{\mu_1}\frac{\mathrm{d}E_1}{\mathrm{d}z}\Bigg|_{z_1}^{z_n}+T_n\frac{W_n(z)}{\mu_{n-1}}\frac{\mathrm{d}E_n}{\mathrm{d}z}\Bigg|_{z_1}^{z_n}.$$

As for the boundary term, observe that $W_i(z)\big|_{z_1}^{z_n}$ is zero whenever $i\notin\{1,n\}$, and that $W_1(z_1)=W_n(z_n)=1$ and $W_1(z_n)=W_n(z_1)=0$, and so the right-hand side (henceforth referred to as "BT") becomes:

$$\mathrm{BT}=\frac{T_1}{\mu_1}\frac{\mathrm{d}E_1}{\mathrm{d}z}\bigg|_{z=z_1}-\frac{T_n}{\mu_{n-1}}\frac{\mathrm{d}E_n}{\mathrm{d}z}\bigg|_{z_n}.$$

However, the spatial derivatives $\frac{\mathrm{d}E_1}{\mathrm{d}z}\big|_{z=z_1}$ and $\frac{\mathrm{d}E_n}{\mathrm{d}z}\big|_{z=z_n}$ are not defined at their respective evaluation points. We will thus wait to evaluate this term until we explicitly apply the boundary conditions. Note for now, however, that $(\mathrm{BT})_i=0$ unless $i\in\{1,n\}$.

We thus obtain the weak form of the governing equation:

$$\sum_{i=2}^{n-1}\left\{T_{i-1}\left[-\frac{1}{\mu_{i-1}h_{i-1}}+\mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}\right]+\right.$$
$$\left.+T_i\left[\frac{1}{\mu_{i-1}h_{i-1}}+\frac{1}{\mu_i h_i}+\mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}+\mu_i\varepsilon_i\frac{h_i}{3}\right]+T_{i+1}\left[-\frac{1}{\mu_i h_i}+\mu_i\varepsilon_i\frac{h_i}{3}\right]\right\}+ \tag{7.30}$$
$$+T_1\left[\frac{1}{\mu_1 h_1}+\frac{\mu_1\varepsilon_1 h_1}{3}\right]+T_n\left[\frac{1}{\mu_{n-1}h_{n-1}}+\frac{\mu_{n-1}\varepsilon_{n-1}h_{n-1}}{3}\right]=\mathrm{BT}.$$

The single equation above will be satisfied when for $i=1$:

$$T_1\left[\frac{1}{\mu_1 h_1}+\mu_1\varepsilon_1\frac{h_1}{3}\right]+T_2\left[-\frac{1}{\mu_1 h_1}+\mu_1\varepsilon_1\frac{h_1}{3}\right]=(\mathrm{BT})_1,$$

for $i=n$,

$$T_{n-1}\left[-\frac{1}{\mu_{n-1}h_{n-1}}+\mu_{n-1}\varepsilon_{n-1}\frac{h_{n-1}}{3}\right]+T_n\left[\frac{1}{\mu_{n-1}h_{n-1}}+\mu_{n-1}\varepsilon_{n-1}\frac{h_{n-1}}{3}\right]=(\mathrm{BT})_n,$$

and for $i \in [2, n-1] \cap \mathbb{N}$,

$$T_{i-1}\left[-\frac{1}{\mu_{i-1}h_{i-1}} + \mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}\right] + T_i\left[\frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i} + \mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3} + \mu_i\varepsilon_i\frac{h_i}{3}\right] + \\ + T_{i+1}\left[-\frac{1}{\mu_i h_i} + \mu_i\varepsilon_i\frac{h_i}{3}\right] = 0.$$

Matrix Formulation

The previous system of equations can be represented as a single matrix equation:

$$\begin{bmatrix} a_1^2 & a_1^3 & 0 & \dots & \dots & 0 \\ a_2^1 & a_2^2 & a_2^3 & 0 & \dots & \vdots \\ 0 & a_3^1 & a_3^2 & a_3^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & a_{n-1}^1 & a_{n-1}^2 & a_{n-1}^3 \\ 0 & \dots & \dots & 0 & a_n^1 & a_n^2 \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ \vdots \\ T_i \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} (\mathrm{BT})_1 \\ 0 \\ \vdots \\ 0 \\ (\mathrm{BT})_n \end{bmatrix},$$

where for $i \in [2, n-1] \cap \mathbb{N}$,

$$\begin{aligned} a_i^1 &= -\frac{1}{\mu_{i-1}h_{i-1}} + \mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3}, \\ a_i^2 &= \frac{1}{\mu_{i-1}h_{i-1}} + \frac{1}{\mu_i h_i} + \mu_{i-1}\varepsilon_{i-1}\frac{h_{i-1}}{3} + \mu_i\varepsilon_i\frac{h_i}{3}, \quad \text{and} \\ a_i^3 &= -\frac{1}{\mu_i h_i} + \mu_i\varepsilon_i\frac{h_i}{3}, \end{aligned}$$

and for $i = 1$ and $i = n$,

$$\begin{aligned} a_n^1 &= -\frac{1}{\mu_{n-1}h_{n-1}} + \mu_{n-1}\varepsilon_{n-1}\frac{h_{n-1}}{3}, \\ a_1^2 &= \frac{1}{\mu_1 h_1} + \mu_1\varepsilon_1\frac{h_1}{3}, \\ a_n^2 &= \frac{1}{\mu_{n-1}h_{n-1}} + \mu_{n-1}\varepsilon_{n-1}\frac{h_{n-1}}{3}, \quad \text{and} \\ a_1^3 &= -\frac{1}{\mu_1 h_1} + \mu_1\varepsilon_1\frac{h_1}{3}. \end{aligned}$$

Boundary Conditions

Now we consider the boundary conditions, which are the inhomogeneous Dirichlet condition at the left-hand endpoint (*i.e.*, at $z = 0$), and the homogeneous Dirichlet condition at the right-hand endpoint (*i.e.*, at $z = L$), shown in Equations 2.81 and 2.82.

Note that $E_j(0) = 0$ for all j except $j = 1$, and that $E_1(0) = 1$; in this case, then, the boundary condition at $z = 0$ implies

$$\sum_{j=1}^{n} T_j(t)E_j(0) = T_1(t) = \frac{2}{a}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}.$$

We thus replace the first row of the left-hand side matrix by the first row of the $n \times n$ identity matrix, and replace the first entry of the boundary term vector by the value $\frac{2}{a}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}$. That is, we let $a_1^2 = 1$, $a_1^3 = 0$, and $(\mathrm{BT})_1 = \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)}$.

For the boundary condition at $x = L$, we have

$$\sum_{j=1}^{n} T_j(t)E_j(L) = T_n(t) = 0,$$

and to implement this, we replace the final row of the left-hand side matrix by the final row of the $n \times n$ identity matrix, and replace the final entry of the boundary term vector by 0; that is, we set $a_n^2 = 1$, $a_n^1 = 0$, and $(\mathrm{BT})_n = 0$.

The resulting system is, finally, the following:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & \dots & 0 \\ a_2^1 & a_2^2 & a_2^3 & 0 & \dots & \vdots \\ 0 & a_3^1 & a_3^2 & a_3^3 & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & a_{n-1}^1 & a_{n-1}^2 & a_{n-1}^3 \\ 0 & \dots & \dots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} T_1 \\ \vdots \\ T_i \\ \vdots \\ T_n \end{bmatrix} = \begin{bmatrix} \frac{2}{L}\sqrt{2P\left(\frac{\omega\cdot\mu_0}{\beta}\right)} \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where the a_i^ℓ, b_i^ℓ, and c_1^ℓ, for $\ell \in \{1,2,3\}$ and $i \in [2, n-1]$, are the same ones given above.

Simulation Results

We have written a MATLAB code (see the full code in Appendix D.7) to solve the above system. For this simulation, we have used a length $L = 1$ m, the number of (uniformly spaced) nodes $n = 50$, and have assumed that the parameters μ, σ, and ε' correspond to those of air in the first and final thirds of the domain (*i.e.*, for $z \in [0, L/3) \cup (2L/3, L]$) and to those of water in the second third (*i.e.*, for $z \in [L/3, 2L/3]$). Namely, we have used the following values:

$$\mu_{\text{air}} = 1, \qquad \sigma_{\text{air}} = 0, \qquad \varepsilon'_{\text{air}} = 1$$
$$\mu_{\text{water}} = 1, \qquad \sigma_{\text{water}} = 0.055, \qquad \varepsilon'_{\text{water}} = 75,$$

which were taken from [143–145]. The resulting envelope of the electric field intensity calculated by this routine is as shown in Figure 7.11.

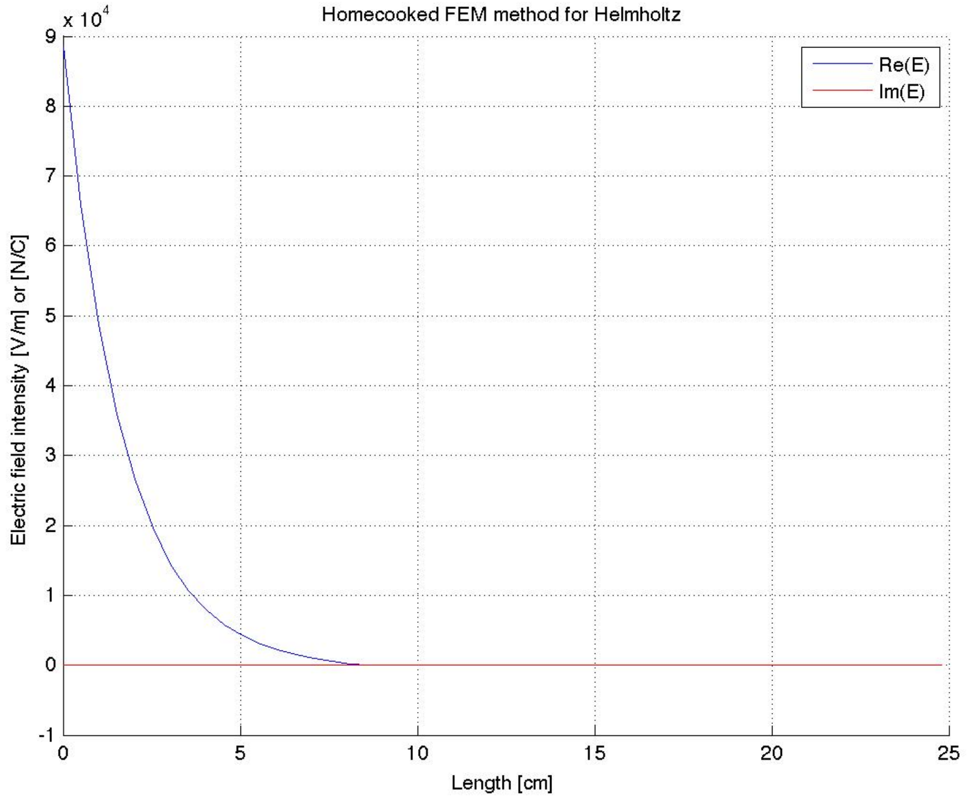


Figure 7.11: Envelope of electric field with one-dimensional Helmholtz equation solver.

Helmholtz, Laplace, and Poisson Solver Using FEM

A similar solver to that described in the beginning of this Chapter, but that is extensible to the Laplace and Poisson equations as well, is described in Chapter 3 of [2]. This solver finds a solution to the equation

$$-\frac{\mathrm{d}}{\mathrm{d}z}\left(a\frac{\mathrm{d}u}{\mathrm{d}z}\right) + bu = f,$$

where a and b are known parameters associated with the physical properties of the domain, and f is a known source or excitation function. The solver is capable of handling Dirichlet, Neumann, or third-kind boundary conditions at the left- and right-hand endpoints.

Taking $a = {\mu_{\mathrm{ref}}}^{-1}$, $b = \frac{\mu_{\mathrm{ref}}\omega^2\varepsilon_{\mathrm{ref}}}{c^2}$, and $f = 0$, it is not difficult to show that Jin's solver is mathematically identical to our own Helmholtz solver. Our code implementing Jin's solver can be found in Appendix D.7, and we test its accuracy by simulating a solution to Problem 10.

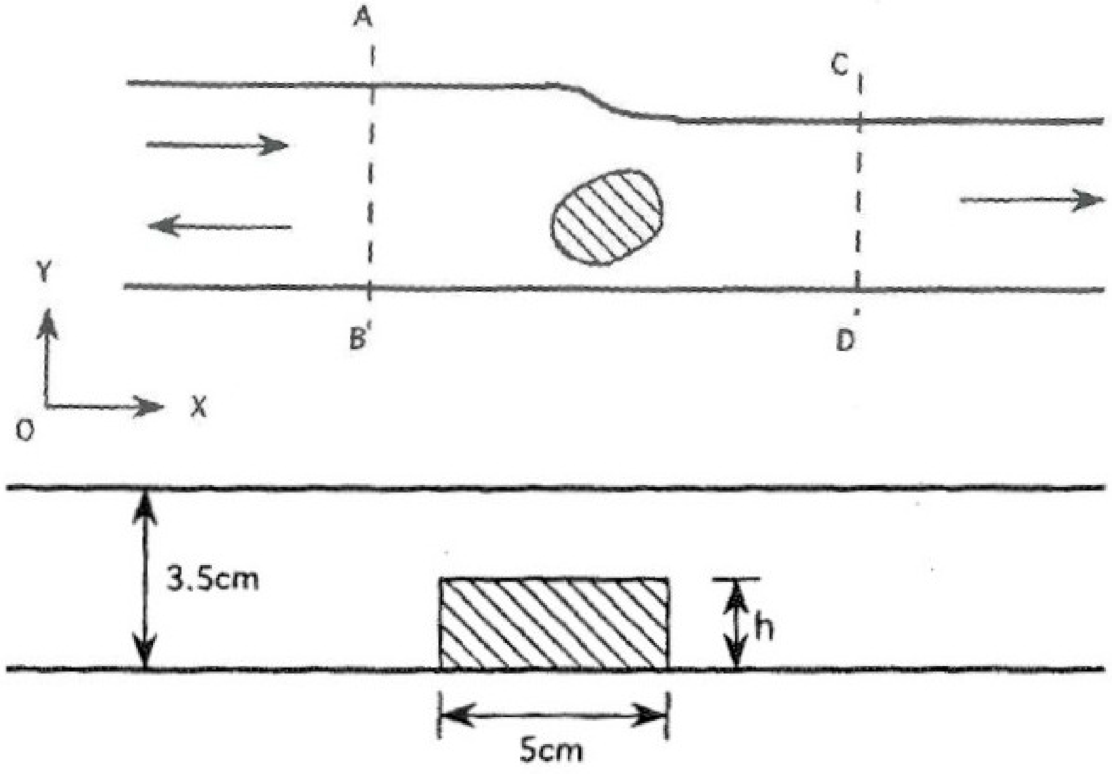


Figure 7.12: Discontinuity in a parallel plate waveguide [2].

Reflection for the Metal-Backed Dielectric Slab

This problem is fully described in Chapter 3 of [2], but we will reproduce here some basic information about its formulation.

We consider the situation shown in Figure 7.12 where a uniform plane wave is incident upon an inhomogeneous dielectric slab backed by a conducting plane. The dielectric slab has thickness L, relative permittivity ε_{rel}, and permeability μ_{rel}; the latter two can be functions of z. The surrounding medium is free space having $\varepsilon_{\text{rel}} = \mu_{\text{rel}} = 1$. We are interested in finding the power reflected by the slab.

It is well known that any plane wave can be decomposed into an E_y-polarized plane wave having only a y-component for the electric field, and an H_y-polarized plane wave having only a y-component for the magnetic field. Therefore, it is sufficient to consider only these two polarization cases.

For the E_y-polarization case, the Helmholtz equation governing the electric field E_y is

$$\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{\mu_{\text{rel}}}\frac{\mathrm{d}E_y}{\mathrm{d}z}\right) + k_0^2\left(\varepsilon_{\text{rel}} - \frac{1}{\mu_{\text{rel}}}\sin^2\theta\right)E_y = 0,$$

where $k_0 = \omega\sqrt{\varepsilon_0\mu_0}$, and θ is the angle of incidence shown in Figure 7.12. We will impose the homogeneous Dirichlet condition at the left-hand boundary ($E_y(0) = 0$), and at the right-hand boundary, we will impose the following boundary condition of the third kind:

$$\left[\frac{1}{\mu_{\text{rel}}}\frac{\mathrm{d}E_y}{\mathrm{d}z} + jk_0\cos\theta E_y\right]\Bigg|_{z=L} = 2jk_0\cos\theta E_0 e^{jk_0 L\cos\theta},$$

where E_0 is a constant denoting the magnitude of the incident field.

For the H_y-polarization case, the Helmholtz equation governing the magnetic field H_y is

$$\frac{\mathrm{d}}{\mathrm{d}z}\left(\frac{1}{\varepsilon_{\text{rel}}}\frac{\mathrm{d}E_y}{\mathrm{d}z}\right) + k_0^2\left(\mu_{\text{rel}} - \frac{1}{\varepsilon_{\text{rel}}}\sin^2\theta\right)E_y = 0,$$

and we will impose the homogeneous Neumann condition at the left-hand boundary: $\left(\frac{\mathrm{d}H_y}{\mathrm{d}z}\Big|_{z=0} = 0\right)$, and at the right-hand boundary, we will impose the following boundary condition of the third kind:

$$\left[\frac{1}{\varepsilon_{\text{rel}}}\frac{\mathrm{d}H_y}{\mathrm{d}z} + jk_0\cos\theta H_y\right]\Bigg|_{z=L} = 2jk_0\cos\theta H_0 e^{jk_0 L\cos\theta}.$$

Once we solve the E_y and H_y fields at $z = L$, the reflection coefficient may be found according to

$$R = \frac{E_y(L) - E_0 e^{jk_0 L\cos\theta}}{E_0 e^{-jk_0 L\cos\theta}}$$

for E_y-polarization, and similarly for H_y-polarization, according to

$$R = \frac{H_y(L) - H_0 e^{jk_0 L\cos\theta}}{H_0 e^{-jk_0 L\cos\theta}}.$$

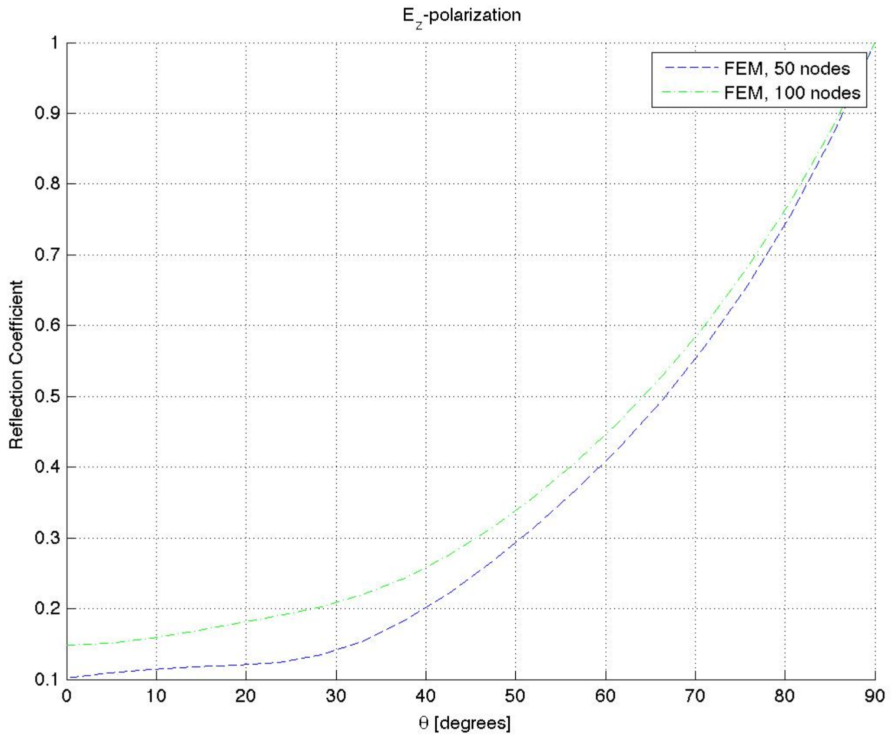
Problem 10. *Find the reflection coefficients for the scenario described above with the metal-backed dielectric slab.*

We carry out a simulation for this problem using out implementation of Jin's solver, running the simulation for both 50 nodes and 100 nodes. Code for this problem may be found in Appendix D.7. We have found that the results from this solver identically replicate those from Chapter 3 of [2], and our results look as shown in Figure 7.13.

Semi-Analytical Solution Using `bvp4c`

Starting with the representation in Equation (2.77), we let $S(z) = \mathrm{Re}(\mathcal{E}(z))$, and $W(z) = -\mathrm{Im}(\mathcal{E}(z))$, so that

$$\mathrm{Re}\left\{\mathcal{E}(z)e^{i\omega t}\right\} = S(z)\cos\omega t - W(z)\sin\omega t. \tag{7.31}$$



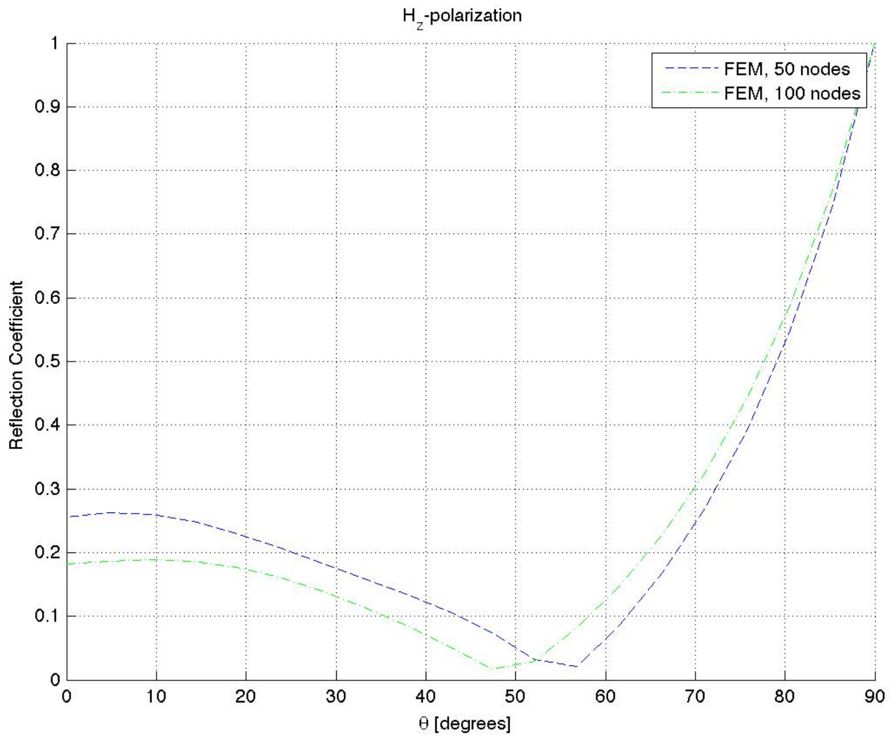


Figure 7.13: Simulated reflection coefficients for the metal-backed dielectric slab examples in the E_y and H_y polarizations.

Recall that the one-dimensional Helmholtz equation is

$$\frac{\partial}{\partial z}\left(\mu_{\text{rel}}^{-1}\frac{\partial \mathcal{E}(z)}{\partial z}\right) = \frac{\mu_{\text{rel}}\omega^2}{c^2}(\varepsilon' - i\varepsilon'')\mathcal{E}(z), \tag{7.32}$$

where $\varepsilon'' = \frac{\sigma}{\omega\varepsilon_0}$ is the material's (unitless) relative electromagnetic loss factor.

The first and second time derivatives of Equation 7.31 are as follows:

$$\begin{aligned}\frac{\partial\vec{\mathcal{E}}}{\partial t} &= \langle 0, 0, -\omega S(z)\sin\omega t - \omega W(z)\cos\omega t\rangle \\ \frac{\partial^2\vec{\mathcal{E}}}{\partial t^2} &= \langle 0, 0, -\omega^2 S(z)\cos\omega t + \omega^2 W(x)\sin\omega t\rangle.\end{aligned}$$

Assuming that μ_{rel} is isotropic (*i.e.*, constant in space), we may take its inverse outside of the partial derivatives on the left-hand side of Equation 2.68, and so

$$\frac{\partial}{\partial z}\left(\mu_{\text{rel}}^{-1}\frac{\partial\vec{\mathcal{E}}}{\partial z}\right) = \mu_{\text{rel}}^{-1}\frac{\partial^2\vec{\mathcal{E}}}{\partial z^2} = \mu_{\text{rel}}^{-1}\langle 0, 0, \frac{\mathrm{d}^2 S(z)}{\mathrm{d}z^2}\cos\omega t - \frac{\mathrm{d}^2 W(z)}{\mathrm{d}z^2}\sin\omega t\rangle.$$

We may plug this into Equation 2.68 to obtain

$$\begin{aligned}&\mu_{\text{rel}}^{-1}\left\langle 0, 0, \frac{d^2 S(z)}{dz^2}\cos\omega t - \frac{d^2 W(z)}{dz^2}\sin\omega t\right\rangle \\ &\quad = \left\langle 0, 0, \mu_0\sigma\omega\Big(S(z)\sin\omega t - \omega W(z)\cos\omega t\Big) + \frac{\varepsilon'}{c^2}\omega^2\Big(S(z)\cos\omega t + \omega^2 W(z)\sin\omega t\Big)\right\rangle,\end{aligned}$$

so that

$$\begin{aligned}\frac{\mathrm{d}^2 S(z)}{\mathrm{d}x^2}\cos\omega t - \frac{\mathrm{d}^2 W(z)}{\mathrm{d}z^2}\sin\omega t &= \mu_{\text{rel}}\mu_0\sigma\omega\left(S(x)\sin\omega t - \omega W(z)\cos\omega t\right) + \\ &\quad + \mu_{\text{rel}}\omega^2\frac{\varepsilon'}{c^2}\Big(S(z)\cos\omega t + \omega^2 W(z)\sin\omega t\Big) \\ &= \left(\mu\sigma\omega S(z) - \mu_{\text{rel}}\frac{\varepsilon'\omega^2}{c^2}W(z)\right)\sin\omega t + \\ &\quad + \left(\mu\sigma\omega W(z) + \mu_{\text{rel}}\frac{\varepsilon'\omega^2}{c^2}S(z)\right)\cos\omega t.\end{aligned}$$

We now employ an alternative definition of the permittivity of free space: $\varepsilon_0 = 1/\mu_0 c^2$. When we use this relationship and equate the coefficients of $\cos\omega t$ and $\sin\omega t$ in the above equation, we obtain

$$\begin{aligned}\frac{\mathrm{d}^2 S(z)}{\mathrm{d}z^2} &= \mu\sigma\omega S(z) - \mu\frac{\varepsilon'\omega^2}{c^2}W(z) \\ &= \frac{\mu\omega^2}{c^2}\left(\varepsilon' W(z) - \varepsilon'' S(z)\right)\end{aligned}$$

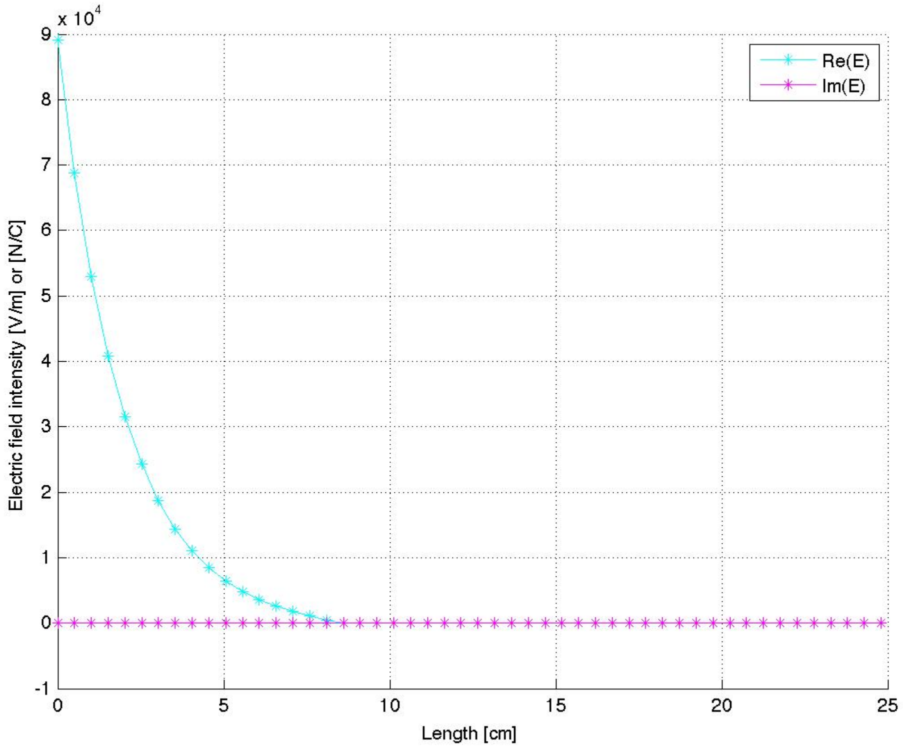


Figure 7.14: Real and imaginary parts of the electric field intensity calculated using `bvp4c`.

and

$$\begin{aligned}\frac{\mathrm{d}^2W(z)}{\mathrm{d}z^2} &= \mu\sigma\omega W(z) - \mu_{\text{rel}}\frac{\varepsilon'\omega^2}{c^2}S(z)\\ &= \frac{\mu\omega^2}{c^2}\Big(\varepsilon' S(z) + \varepsilon'' W(z)\Big),\end{aligned}$$

or, in matrix form,

$$\frac{\mathrm{d}^2}{\mathrm{d}z^2}\begin{bmatrix}S(z)\\W(z)\end{bmatrix} = \frac{\mu_{\text{rel}}\omega^2}{c^2}\begin{bmatrix}\varepsilon'' & \varepsilon'\\ \varepsilon' & \varepsilon''\end{bmatrix}\begin{bmatrix}S(z)\\W(z)\end{bmatrix} \tag{7.33}$$

We solve the above system using MATLAB's built-in solver `bvp4c`, with the full code shown in Appendix D.7. The resulting real and imaginary parts of the electric field intensity calculated with this code are as shown in Figure 7.14. These same curves, overlaid with those produced by our own finite element code from Section 7.3, look as shown in Figure 7.15

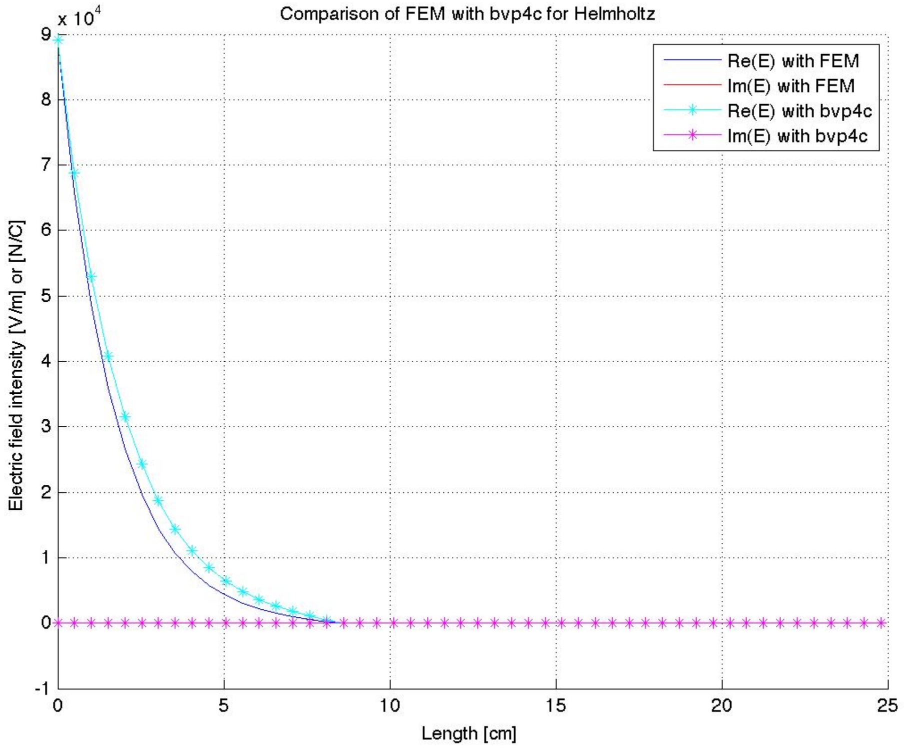


Figure 7.15: Comparison of the real and imaginary parts of the electric field intensity calculated using `bvp4c` and calculated using finite element method.

7.4 Techniques for Solving the Two-Dimensional Helmholtz Equation

Finite Difference Methods

This section outlines a finite difference technique for solving the two-dimensional Helmholtz equation given in Equation 2.80. The technique used is a second-order difference method that relies on the spatial discretization shown in Figure 7.7, and that employs the approximations

$$\frac{\partial \mathcal{E}}{\partial x^2}\Big|_{\substack{x=x_k\\ z=z_j}} \approx \frac{\mathcal{E}_j^{k-1} - 2\mathcal{E}_j^k + \mathcal{E}_j^{k+1}}{(x_{k+1}-x_k)(x_k - x_{k-1})}, \qquad \frac{\partial \mathcal{E}}{\partial z^2}\Big|_{\substack{x=x_k\\ z=z_j}} \approx \frac{\mathcal{E}_{j-1}^{k} - 2\mathcal{E}_j^k + \mathcal{E}_{j+1}^{k}}{(z_{j+1}-z_j)(z_j - z_{j-1})}.$$

Substituting these equations into Equation 2.80 yields

$$\frac{\mathcal{E}_j^{k-1} - 2\mathcal{E}_j^k + \mathcal{E}_j^{k+1}}{(x_{k+1}-x_k)(x_k - x_{k-1})} + \frac{\mathcal{E}_{j-1}^{k} - 2\mathcal{E}_j^k + \mathcal{E}_{j+1}^{k}}{(z_{j+1}-z_j)(z_j - z_{j-1})} + \mu^2\varepsilon\omega^2\mathcal{E}_j^k = 0,$$

j	k	Equation
0	$0,\dots,M-1$	7.35
$1,\dots,N-2$	0	7.35
$1,\dots,N-2$	$1,\dots,M-2$	7.34
$1,\dots,N-2$	$M-1$	7.35
$N-1$	$0,\dots,M-1$	7.35

Table 7.2: Description of the organization of the linear system for the finite difference approximation of the two-dimensional wave equation.

for $j \in [1, N-2] \cap \mathbb{N}$ and $k \in [1, M-2] \cap \mathbb{N}$, and where ε and μ refer to the absolute (not relative) permittivity and permeability.

This simplifies to the finite difference scheme

$$\mathcal{E}_j^k \left(\mu^2 \varepsilon \omega^2 - \frac{2}{(\Delta z)_j^2} - \frac{2}{(\Delta x)_k^2} \right) + \frac{\mathcal{E}_{j-1}^k + \mathcal{E}_{j+1}^k}{(\Delta z)_j^2} + \frac{\mathcal{E}_j^{k-1} + \mathcal{E}_j^{k+1}}{(\Delta x)_k^2}, \tag{7.34}$$

where $(\Delta x)_k^2 := (x_{k+1} - x_k)(x_k - x_{k-1})$ and $(\Delta z)_j^2 := (z_{j+1} - z_j)(z_j - z_{j-1})$.

The boundary conditions in Equations 2.82 and 2.81 are applied at the left-hand wall and on the remaining three walls, respectively, as

$$E_j^0 = E_j^{M-1} = 0, \text{ for } j \in [1, N-1] \cap \mathbb{N}, \quad E_0^k = E_{\text{inc}},\ E_{N-1}^k = 0, \text{ for } k \in [0, M-1]. \tag{7.35}$$

Together, Equations 7.34 and 7.35 constitute a system of $N \times M$ many equations in the same number of unknowns, and may therefore be taken as a linear system, where the equations are placed in the order shown in Table 7.2, which leads to the system

$$A\vec{\mathcal{E}} = \vec{c}, \tag{7.36}$$

where

$$\vec{\mathcal{E}} = \left[\mathcal{E}_0^0,\ \mathcal{E}_0^1,\ \cdots\ \mathcal{E}_0^{M-1},\ \mathcal{E}_1^0,\ \mathcal{E}_1^1,\ \cdots\ \mathcal{E}_1^{M-1},\ \cdots\ \mathcal{E}_{N-1}^0,\ \mathcal{E}_{N-1}^1,\ \cdots\ \mathcal{E}_{N-1}^{M-1}\right]^\top,$$

$$\vec{c} = \big[\underbrace{E_{\text{inc}},\ \cdots\ E_{\text{inc}},}_{M}\ 0,\ \cdots\ 0\big]^\top,$$

and A is given in Appendix A.2. The solution of this matrix equation yields the phasor representation of the electric field, which may be squared, according to Equation 2.39, to yield the average of the square of the magnitude of the electric field over a given time interval, for use as input to the source term of the heat equation, described in Problem 7.

Finite Element Methods

Governing Equation

Detailed description of a solver for the general differential equation

$$-\frac{\partial}{\partial x}\left(\alpha_x \frac{\partial \phi}{\partial x}\right) - \frac{\partial}{\partial z}\left(\alpha_z \frac{\partial \phi}{\partial z}\right) + \beta\phi = f,$$

where α_x, α_z, and β are known parameters associated with the physical properties of the domain, and f is the source or excitation function, is given in [2]. Again, special forms of this equation are the Helmholtz equation, Laplace equation, and Poisson equation. The boundary conditions on the two-dimensional domain may be given by

$$\phi = p \quad \text{on} \quad \Gamma_1,$$

and

$$\left(\alpha_x \frac{\partial \phi}{\partial x}\hat{x} + \alpha_z \frac{\partial \phi}{\partial z}\hat{z}\right) \cdot \hat{n} + \gamma\phi = q \quad \text{on} \quad \Gamma_2,$$

where $\Gamma := \Gamma_1 + \Gamma_2$ denotes the contour or boundary enclosing the entire domain Ω, $\hat{n}$ is the outward unit normal vector, and γ, p, and q are known parameters associated with the physical properties of the boundary.

This solver is described in detail in [2], but we give here an overview of its structure.

Weak Formulation of the Governing Equation

The variational problem equivalent to the boundary value problem described in the preceding section is given by

$$\begin{cases} \delta F(\phi) = 0 \\ \phi = p \quad \text{on } \Gamma_1, \end{cases}$$

where

$$F(\phi) = \frac{1}{2}\iint_\Omega \left[\alpha_x \left(\frac{\partial \phi}{\partial x}\right)^2 + \alpha_y \left(\frac{\partial \phi}{\partial z}\right)^2 + \beta\phi^2\right] \mathrm{d}\Omega + \int_{\Gamma_2} \left(\frac{\gamma}{2}\phi^2 - q\phi\right) \mathrm{d}\Gamma - \iint_\Omega f\phi \mathrm{d}\Omega.$$

The proof of equivalency of the variational problem and the boundary value problem is given in [2], along with a discussion of continuity conditions to be imposed in the event of a strongly inhomogeneous domain.

Spatial Discretization

The domain Ω is to be divided into a number of two-dimensional elements, and in this formulation, we choose triangular elements that satisfy the basic requirements of an admissible FEM mesh:

- There should be neither gaps nor overlaps between any elements;
- Elements should be connected via their vertices;

and additionally, the elements should be generated to satisfy the following constraints, which exist to ensure quick convergence to the correct solution:

- Narrow elements (those having a very small interior angle) should be avoided, and all elements should be made as close to equilateral as possible;
- The number of elements should be kept to the minimum for desired accuracy, which may entail the use of small elements where the solution is expected to have drastic variation, and larger elements elsewhere.

We label these elements and vertices with integers, and note that since each element is comprised of three nodes, a node has its own position in the associated element, in addition to its position in the entire system. This elemental position can also be labelled with an integer, referred to as the "local number", in addition to the "global number" indicating its position in the entire system. To relate these three numbers (the element number, the local node number, and the global node number), we use a $3 \times M$ array whose entries are denoted $n(i,e)$ (where $i \in \{1,2,3\}$ and $e \in [1,M] \times \mathbb{N}$), where M is the total number of elements. The entry $n(i,e)$ is the global number of the node that has local number i on element e. The local node numbers, it should be noted, must be ordered consistently for all elements in either a clockwise or counterclockwise way.

In addition to the connectivity matrix described above, some other inputs necessary for the finite element formulation include the x_i and z_i coordinates for each node of the domain; the α_x, α_z, β, and f values on each element; the global numbers of the nodes residing on Γ_1 and their p-values on those nodes; and the values of γ and q for each segment coincident with Γ_2.

The unknown function ϕ is approximated on element e as

$$\phi^e(x,z) = a^e + b^e x + c^e z,$$

where a^e, b^e, and c^e are constant coefficients to be determined. The imposition of this condition at each of the nodes, together with a condition enforcing continuity of ϕ, yields

$$\phi^e(x,z) = \sum_{i=1}^{3} N_j^e(x,z)\phi_j^e,$$

where for $j \in \{1,2,3\}$, $\phi_j^e(x,z) = a^e + b^e x_j + c^e z_j$, and N_j^e are the "hat"-shaped interpolation functions given by

$$N_j^e(x,z) = \frac{1}{2\Delta^e}(a_j^e + b_j^e x + c_j^e z),$$

in which

$$\begin{aligned}
a_1^e &= x_2^e z_3^e - z_2^e x_3^e; & b_1^e &= z_2^e - z_3^e; & c_1^e &= x_3^e - x_2^e;\\
a_2^e &= x_3^e z_1^e - z_3^e x_1^e; & b_1^e &= z_3^e - z_1^e; & c_1^e &= x_1^e - x_3^e;\\
a_3^e &= x_2^e z_2^e - z_1^e x_2^e; & b_1^e &= z_1^e - z_2^e; & c_1^e &= x_2^e - x_1^e,
\end{aligned}$$

and

$$\Delta^e = \frac{1}{2}(b_1^e c_2^e - b_2^e c_1^e)$$

is the area of element e.

Matrix Formulation

We consider the function $F(\phi)$ as the sum

$$F(\phi) = \sum_{e=1}^{M} F^e(\phi^e),$$

where again, M is the number of elements. Here, F^e is the subfunctional

$$F^e(\phi^e) = \frac{1}{2}\iint_{\Omega^e}\left[\alpha_x\left(\frac{\partial\phi^e}{\partial x}\right)^2 + \alpha_z\left(\frac{\partial\phi^e}{\partial z}\right)^2 + \beta(\phi^e)^2\right]\mathrm{d}\Omega - \iint_{\Omega^e} f\phi^e\mathrm{d}\Omega,$$

where Ω^e denotes the domain of element e.

Introducing the expression for ϕ^e derived previously, and differentiating F^e with respect to ϕ_i^e yields the matrix equation

$$\left\{\frac{\partial F^e}{\partial\phi^e}\right\} = [K^e]\{\phi^e\} - \{b^e\},$$

where

$$\left\{\frac{\partial F^e}{\partial\phi^e}\right\} = \left[\frac{\partial F^e}{\partial\phi_1^e}, \frac{\partial F^e}{\partial\phi_2^e}, \frac{\partial F^e}{\partial\phi_3^e}\right]^\top; \quad \{\phi^e\} = [\phi_1^e, \phi_2^e, \phi_3^e]^\top,$$

the elements of the 3×3 matrix K^e are given by

$$K_i^e = \frac{1}{4\Delta^e}\left(\alpha_x b_i^e b_j^e + \alpha_z c_i^e c_j^e\right) + \frac{\Delta^e}{12}\beta^e(1 + \delta_{ij}),$$

and the elements of the vector b^e are

$$b_i^e = \frac{\Delta^e}{3} f^e.$$

We arise at these expressions under the assumption that the coefficients α_x, α_z, β, and the source f are constant within each element.

To assemble the global system from M many local 3×3 systems as above, we follow the formulation

$$\left\{\frac{\partial F}{\partial \phi}\right\} = \sum_{e=1}^{M} \left\{\overline{\frac{\partial F^e}{\partial \phi_e}}\right\} = \sum_{e=1}^{M} \left(\left[\overline{K^e}\right]\{\overline{\phi^e}\} - \{\overline{b^e}\}\right) = \vec{0}, \tag{7.37}$$

where all vectors $\overline{b^e}$, $\overline{\phi^e}$, and the matrix $\overline{K^e}$ have been expanded so that the element $\overline{K^e}_{ij}$ is in the position $n(i,e)n(i,j)$ in the expanded form, and similarly with the vectors.

In particular, the matrix equation

$$K\phi = b$$

results, where

$$K = \sum_{e=1}^{M} \overline{K^e}, \quad \text{and} \quad b = \sum_{e=1}^{M} \overline{b^e}.$$

Boundary Conditions

To incorporate the boundary condition of the third kind, we include the functional

$$F_b(\phi) = \int_{\Gamma_2} \left(\frac{\gamma}{2}\phi^2 - q\phi\right) \mathrm{d}\Gamma$$

into the system by modifying Equation (7.37) as follows:

$$\left\{\frac{\partial F}{\partial \phi}\right\} = \sum_{e=1}^{M} \left\{\overline{\frac{\partial F^e}{\partial \phi_e}}\right\} + \sum_{s=1}^{M_s} \left\{\overline{\frac{\partial F_b^s}{\partial \phi_s}}\right\} = \sum_{e=1}^{M} \left(\left[\overline{K^e}\right]\{\overline{\phi^e}\} - \{\overline{b^e}\}\right) + \sum_{s=1}^{M_s} \left(\left[\overline{K^s}\right]\{\overline{\phi^s}\} - \{\overline{b^s}\}\right) = \vec{0},$$

where Γ_2 is comprised of M_s many sides or segments, F_b^s denotes the integral over segment s,

$$F_b(\phi) = \sum_{s=1}^{M_s} F_b^2(\phi),$$

and where the local matrices and vectors have been extended to their global versions.

Let $ns(i,s)$ be a $2 \times M_s$ connectivity matrix similar to $n(i,e)$, where the entry in position (i,s) is the global node number of the i^{th} local node on edge s comprising the Γ_2 boundary. In practice, then, the global K matrix may be modified so that for each $s \in [1, M_s] \cap \mathbb{N}$ and for $i,j \in \{1,2\}$, we add the quantity

$$\gamma^s \frac{l^s}{6}(1 + \delta_{ij}),$$

where l^s denotes the length of segment s, to the entry $K_{ns(i,s),ns(j,s)}$, and we add

$$q^s \frac{l^s}{2}$$

to the entry $b_{ns(i,s)}$ of vector b.

The Dirichlet condition may be imposed similarly to the way it was imposed in the one-dimensional case; namely, if global node number i is on the Dirichlet boundary, then we replace the i^{th} row of K by the i^{th} row of the identity matrix, and the i^{th} entry of b by the fixed value assumed at node i on the boundary. In practice, we may be able to reduce the size of the system by imposing the Dirichlet condition, as may be seen in the discussion of Chapter 4 in [2], and as was implemented in the code carrying out this simulation.

Simulation Results

Our general solver was based on the previous description, more details of which can be found in Chapter 4 of [2]. For meshing the domain, we made use of the MATLAB function `mesh2d`, which may be found in [146], and which incorporates MATLAB's built-in Delaunay triangulation capability. Full code of our solver can be found in Appendix D.9.

Discontinuity in a Parallel-Plate Waveguide

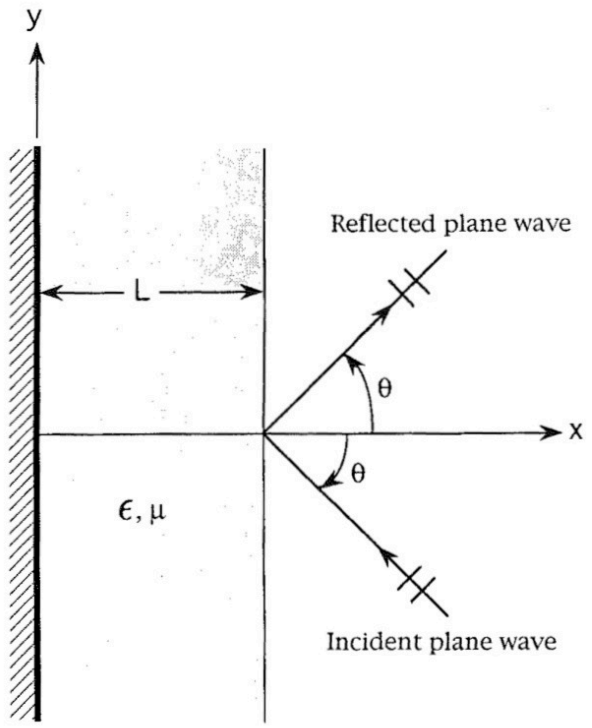


Figure 7.16: Discontinuity in a parallel-plate waveguide [2].

To test our solver, we consider the scenario of a discontinuity in a parallel-plate waveguide, depicted in Figure 7.16. This problem is discussed in detail in Section 4.6 of [2], but we will reproduce the basic formulation here.

We are interested in calculating the proportion of power that passes by the discontinuity and continues to propagate along the waveguide, and the related proportion of power that is reflected and propagates in the opposite direction. We assume that the waveguide is operating at such a frequency that only the dominant mode can propagate without attenuation. Thus, on the left-hand side, far enough away from the discontinuity, the wave can be expressed as the summation of the incident and reflected wave, namely

$$\mathcal{H}_y = \mathcal{H}_y^{\text{inc}} + \mathcal{H}_y^{\text{inc}} + \mathcal{H}_x^{\text{ref}} = \mathcal{H}_0 e^{-jk_0 x} + R\mathcal{H}_0 e^{jk_0 x},$$

where $\mathcal{H}_0$ is a constant, R denotes the reflection coefficient, and k_0 is the propagation constant. Similarly, on the right-hand side, far enough away from the discontinuity, the transmitted wave can be expressed as

$$\mathcal{H}_y = \mathcal{H}_y^{\text{trans}} = T\mathcal{H}_0 e^{-jk_0 x},$$

where T denotes the transmission coefficient. The problem is to determine R and T, and for this we should consider the field near and at the discontinuity; this field can be determined by solving the differential equation

$$\frac{\partial}{\partial x}\left(\frac{1}{\varepsilon_r}\frac{\partial \mathcal{H}_y}{\partial x}\right) + \frac{\partial}{\partial z}\left(\frac{1}{\varepsilon_r}\frac{\partial \mathcal{H}_y}{\partial z}\right) + k_0^2 \mu_r H_y = 0,$$

together with the homogeneous Neumann boundary condition at the waveguide walls $\left(i.e., \frac{\partial \mathcal{H}_y}{\partial n} = 0\right)$.

In order to apply our finite element code, we must truncate the domain, which would otherwise be infinite in the x-direction. We place artificial boundaries one wavelength from the discontinuity on the left- and right-hand sides, and at the left-hand boundary we impose the approximate boundary condition

$$\frac{\partial \mathcal{H}_y}{\partial x} \approx jk_0 \mathcal{H}_y - 2jk_0 \mathcal{H}_0 e^{-jk_0 x},$$

and at the right-hand boundary, we impose the condition

$$\frac{\partial \mathcal{H}_y}{\partial x} \approx -jk_0 \mathcal{H}_y.$$

Finally, from the expression of the $\mathcal{H}_y$ field obtained by solving the described equation together with its boundary conditions, we may calculate the reflection and transmission coefficients as

$$R = \frac{\mathcal{H}_y(x_1) - \mathcal{H}_0 e^{-jk_0 x_1}}{\mathcal{H}_0 e^{jk_0 x_1}},$$
$$T = \frac{\mathcal{H}_y(x_2)}{\mathcal{H}_0 e^{-jk_0 x_2}},$$

where x_1 and x_2 denote the positions of the left- and right-hand boundaries, respectively.

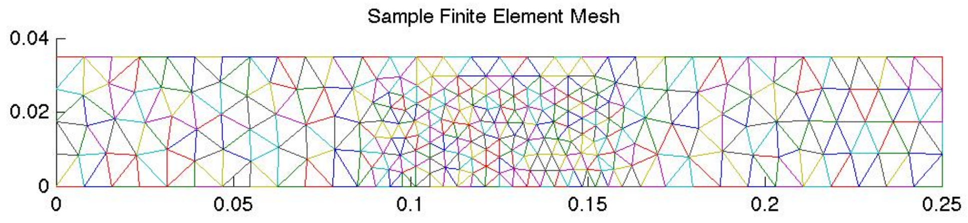


Figure 7.17: Finite element mesh for two-dimensional numerical approximation of the Helmholtz equation.

We generated the mesh of the domain as shown in Figure 7.17, where it can be seen that the mesh is kept finer inside the dielectric inclusion than it is outside of the inclusion.

We have used the code in Appendix D.9 to compute equi-H_y contours for the case with the following physical parameters: the height of the dielectric inclusion is $h = 1.75$ cm, the height of the waveguide is 3.5 cm, the wavelength is $\lambda = 10$ cm, $\mu_r = 2 - j0.1$, $H_0 = 1$, and we have varied the value of ε_r as can be seen in the equi-H_y contours in Figure 7.18. From these contours, it is evident that our evaluations $\mathcal{H}_y(x_1)$ and $\mathcal{H}_y(x_2)$ are well-defined, as at these boundaries, the field varies only insignificantly in the z-direction. The equi-H_y contours calculated by Jin are shown in Figure 7.19 for comparison, and they show agreement with those calculated in our routine.

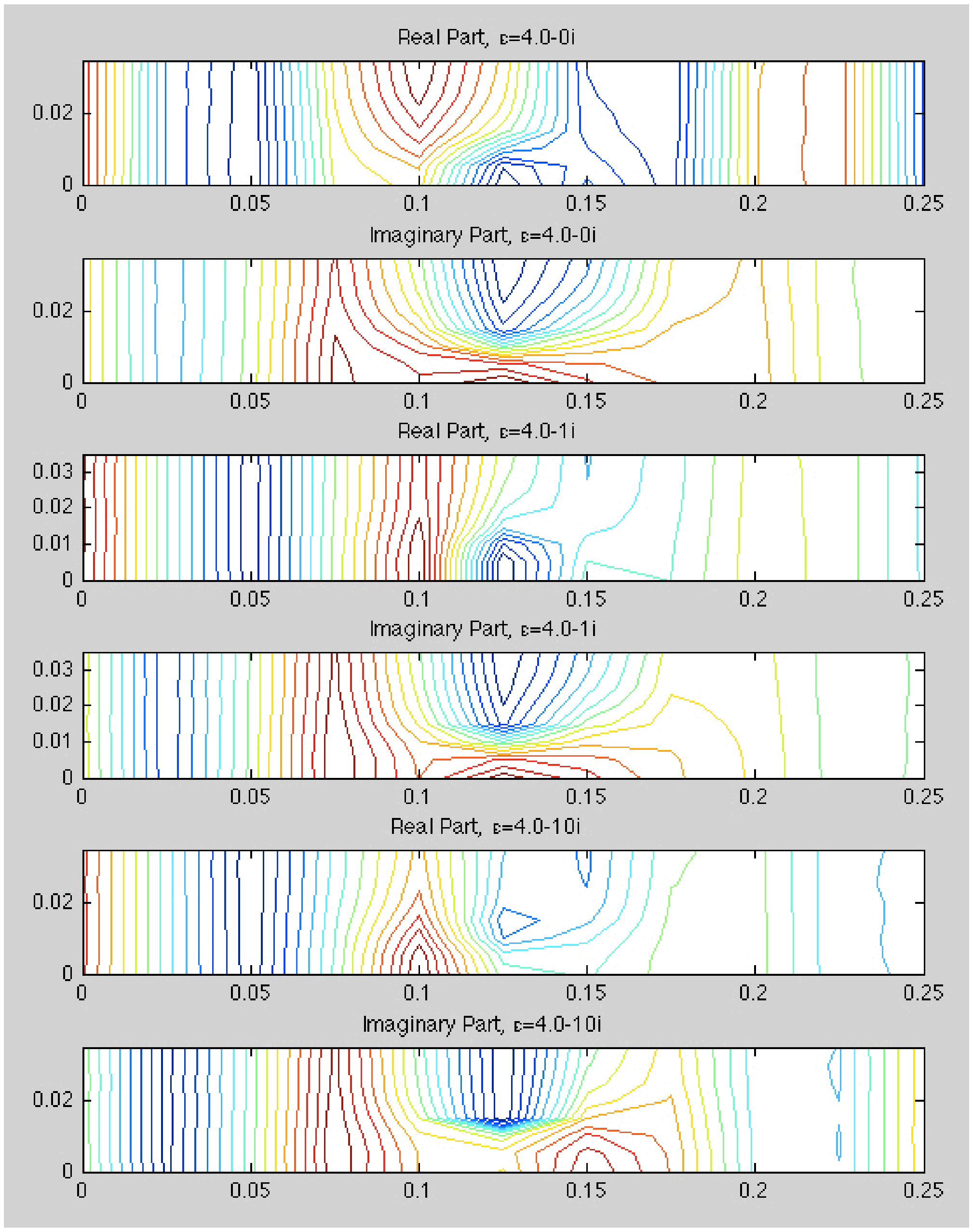


Figure 7.18: Equi-H_y contours generated by finite element method solution.

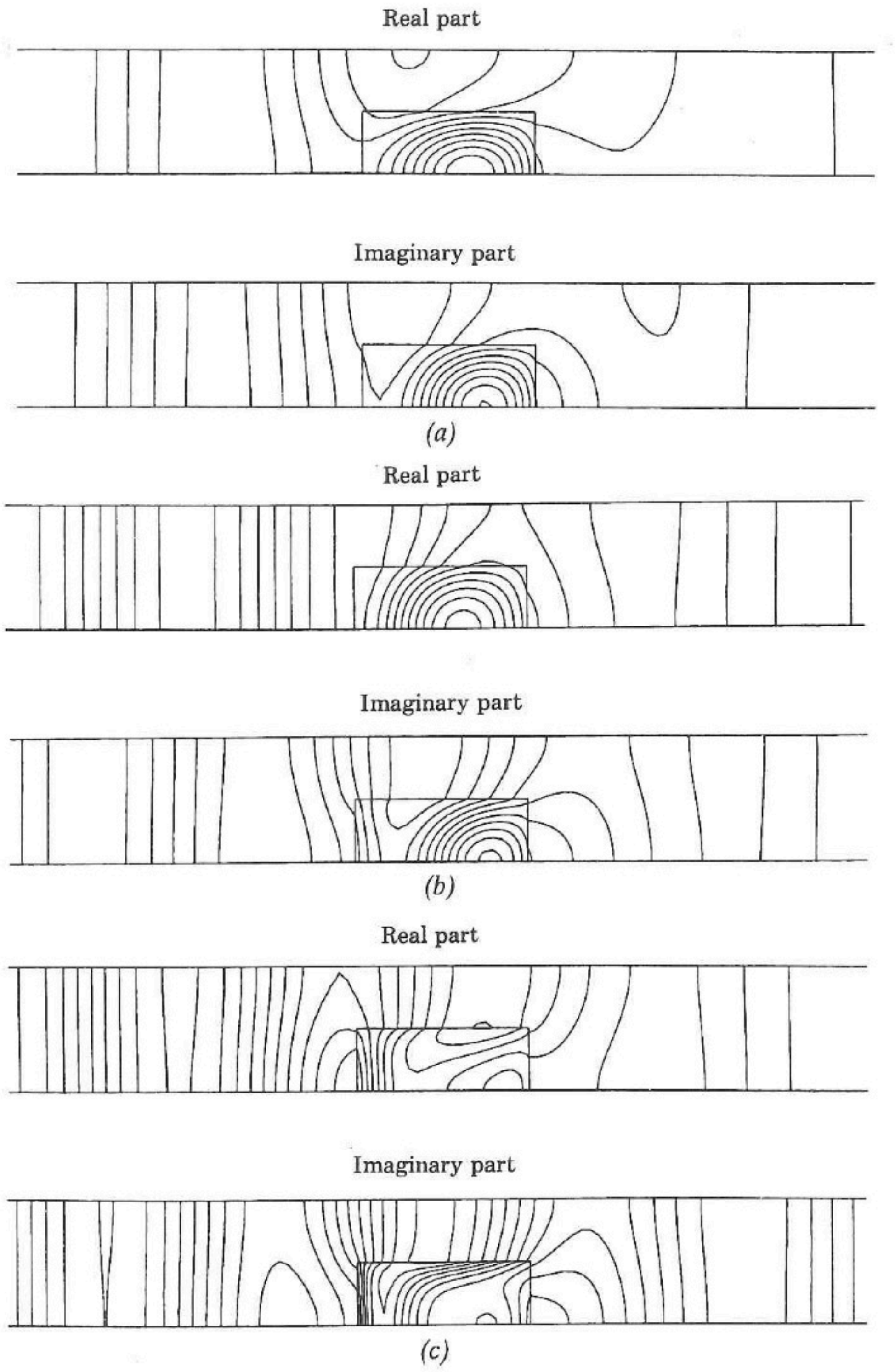


FIGURE 4.16 Equi-H_z contours for $h = 1.75$ cm and $\lambda = 10$ cm. (a) $\epsilon_r = 4.0$. (b) $\epsilon_r = 4.0 - j1.0$. (c) $\epsilon_r = 4.0 - j10.0$.

Figure 7.19: Equi-H_y contours from [2].

Chapter 8

Numerical and Analytical Techniques for Solving the Thermal Problem

In this chapter, we develop a numerical technique for solving the one- and two-dimensional thermal problems formulated and discussed in Chapter 3.

8.1 Techniques for Solving the One-Dimensional Heat Equation

Finite Difference Method

As in Section 7.1, the domain is discretized into spatial intervals that are not necessarily of uniform length, and this discretization is the same one used in Figure 7.1. However, as discussed in Section 3.4, we exclude the portions of the computational domain that do not contain sample material or insulation, and so the size of the computational domain is decreased; we label the first node of our domain as z_0 and the last node z_{N-1}, so that the domain contains N many nodes and $N-1$ many intervals, as shown in Figure 8.1. This labelling scheme for the spatial nodes is different than the one introduced in Section 7.1, although the nodes themselves also appear in the computational domain of that larger scheme.

We assume a time-marching scheme with uniform time steps of length Δt_u, which, for simplicity of expression, we denote in this chapter as Δt.

Let $u_j^n := u(z_j, t_n)$, for $j \in [0, N-1] \cap \mathbb{N}$, and $n \in \{0\} \cup \mathbb{N}$. Consider the θ-scheme with difference

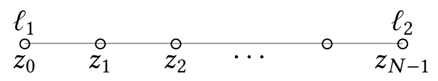


Figure 8.1: Discretization of the one-dimensional computational domain for the heat equation.

approximations

$$\left.\frac{\partial u}{\partial t}\right|_{\substack{z=z_j\\t=t_n}} \approx \frac{u_j^{n+1}-u_j^n}{\Delta t} \quad \text{and}$$

$$\left.\frac{\partial u}{\partial z^2}\right|_{\substack{z=z_j\\t=t_n}} \approx (1-\theta)\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{(z_{j+1}-z_j)(z_j-z_{j-1})} + \theta\frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{(z_{j+1}-z_j)(z_j-z_{j-1})},$$

where $\theta \in [0,1]$, $j \in [1, N-2] \cap \mathbb{N}$, and $n \in \mathbb{N}$. In this scheme, the fully explicit FTCS (forward in time, centered in space) method corresponds to taking $\theta = 0$, the fully implicit BTCS (backward in time, centered in space) method corresponds to taking $\theta = 1$, and the Crank-Nicolson scheme corresponds to $\theta = 0.5$. For $\theta = 0.5$, the scheme is second-order accurate, but for any other value of θ, the scheme is only first-order accurate [147]. For $\theta \geq 0.5$, the scheme is unconditionally stable, but for $\theta < 0.5$, the scheme is stable only when, for all spatial steps $j \in [1, N-2] \cap \mathbb{N}$,

$$\frac{\Delta t}{(z_{j+1}-z_j)(z_j-z_{j-1})} \leq \frac{1}{2-4\theta}.$$

Substituting the difference approximations into the governing equation in Equation 3.14, we obtain

$$\rho c_p\left(\frac{u_j^{n+1}-u_j^n}{\Delta t}\right) - k\left((1-\theta)\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{(z_{j+1}-z_j)(z_j-z_{j-1})} + \theta\frac{u_{j-1}^{n+1} - 2u_j^{n+1} + u_{j+1}^{n+1}}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right) = \omega\varepsilon''|\vec{E}|^2_{\text{avg}}.$$

Arranging the equation so that the "unknown" values at the $(n+1)^{\text{st}}$ time step are on the left-hand side, and all "known" values are on the right-hand side, we obtain

$$u_{j-1}^{n+1}\left(-\frac{k\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right) + u_j^{n+1}\left(\frac{\rho c_p}{\Delta t} + \frac{2k\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right) + u_{j+1}^{n+1}\left(-\frac{k\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right) =$$
$$= u_j^n\left(\frac{\rho c_p}{t_{n+1}-t_n} - \frac{2k(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right) + k(1-\theta)\left(\frac{u_{j-1}^n + u_{j+1}^n}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right) + \omega\varepsilon''|\vec{E}|^2_{\text{avg}},$$

which may be rewritten using the shorthand abbreviations

$$s_j := \frac{k(2\Delta t)}{\rho c_p(z_{j+1}-z_j)(z_j-z_{j-1})} \quad \text{and} \quad q_j := \omega\varepsilon''|\vec{E}|^2_{\text{avg}}\frac{(2\Delta t)}{\rho c_p}$$

as

$$\begin{aligned} -u_{j-1}^{n+1}s_j\theta + u_j^{n+1}(1+2s_j\theta) - u_{j+1}^{n+1}s_j\theta &= \\ = u_{j-1}^n s_j(1-\theta) + u_j^n(1-2s_j(1-\theta)) + u_{j+1}^n s_j(1-\theta) + q_j. \end{aligned} \tag{8.1}$$

The computational grid, representing the solution space with one spatial and one time dimension, is shown in Figure 8.2, and the computational stencil, representing the θ-scheme described by the

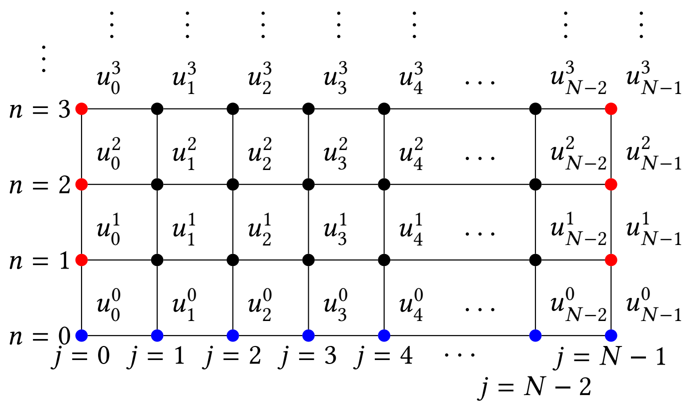

u_0^3 u_1^3 u_2^3 u_3^3 u_4^3 $\dots$ u_{N-2}^3 u_{N-1}^3
u_0^2 u_1^2 u_2^2 u_3^2 u_4^2 $\dots$ u_{N-2}^2 u_{N-1}^2
u_0^1 u_1^1 u_2^1 u_3^1 u_4^1 $\dots$ u_{N-2}^1 u_{N-1}^1
u_0^0 u_1^0 u_2^0 u_3^0 u_4^0 $\dots$ u_{N-2}^0 u_{N-1}^0
$n = 3$
$n = 2$
$n = 1$
$n = 0$
$j = 0$ $j = 1$ $j = 2$ $j = 3$ $j = 4$ $\cdots$ $j = N - 1$
$j = N - 2$

Figure 8.2: Computational grid representing the solution space of the one-dimensional heat equation. Here, $j \in [0, N-1] \cap \mathbb{N}$ represents the position along the spatial domain, and $n \in \{0\} \cup \mathbb{N}$ represents the time step. The blue-colored nodes represent those where the solution is given by the initial condition in Equation 8.6, and the red-colored nodes represent those whose solution is given by the boundary conditions in Equation 8.2, while the solution at the black-colored nodes is given by Equation 8.1.

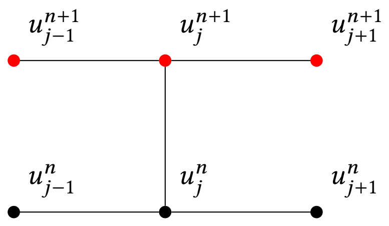


Figure 8.3: Computational stencil of θ-scheme for solving the one-dimensional heat equation. Here, $j \in [1, N-2] \cap \mathbb{N}$ represents the position along the spatial domain, and $n \in \mathbb{N}$ represents the current time step. The nodes in black are ones at which the solution u is known, and the ones in red may be solved for with knowledge of the ones in black.

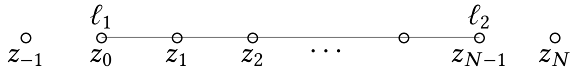


Figure 8.4: Location of the "ghost nodes" for approximating the solution of the heat equation at the boundaries.

above governing equation, may be seen in Figure 8.3. To solve for the unknown values u_K^{n+1} and u_{K+N+1}^{n+1}, we consider the general boundary conditions

$$\alpha_{1,1}\frac{\partial u}{\partial z}\bigg|_{z=\ell_1} + \alpha_{2,1}u(\ell_1,t) = g_1 \quad \text{and} \quad \alpha_{1,2}\frac{\partial u}{\partial z}\bigg|_{z=\ell_2} + \alpha_{2,2}u(\ell_2,t) = g_2, \tag{8.2}$$

where $\alpha_{1,i}$, $\alpha_{2,i}$ and g_i are assumed constant. Using this representation, the case when

$$\begin{cases} \alpha_{1,1} = 0 \\ \alpha_{2,1} = 1 \\ g_1 = T_0 \end{cases} \quad \text{and} \quad \begin{cases} \alpha_{1,2} = 0 \\ \alpha_{2,2} = 1 \\ g_2 = T_0 \end{cases}$$

represents the Dirichlet condition in Equation 3.11, where the temperature on the corresponding border is explicitly fixed; the case when

$$\begin{cases} \alpha_{1,1} = 1 \\ \alpha_{2,1} = 0 \\ g_1 = 0 \end{cases} \quad \text{and} \quad \begin{cases} \alpha_{1,2} = 1 \\ \alpha_{2,2} = 0 \\ g_2 = 0 \end{cases}$$

represents the Neumann condition in Equation 3.12, where the heat flux is fixed; and the case where

$$\begin{cases} \alpha_{1,1} = 1 \\ \alpha_{2,1} = -h \\ g_1 = -hT_{\text{amb}} \end{cases} \quad \text{and} \quad \begin{cases} \alpha_{1,2} = 1 \\ \alpha_{2,2} = h \\ g_2 = hT_{\text{amb}} \end{cases}$$

represents the radiative boundary condition discussed in Equation 3.13.

To implement these boundary conditions, we must temporarily assume the presence of the "ghost nodes" z_{-1} and z_N, so that the solution at z_0 and z_{N-1} may still be approximated using second-order difference formulas [148]. The placement of these ghost nodes is shown in Figure 8.4.

Using the ghost node, the second-order finite difference equation approximating the boundary condition from Equation 8.2 at time t_p and at the left-hand ($z = \ell_1$) endpoint is

$$\alpha_{1,1}\left(\frac{u_1^p - u_{-1}^p}{z_1 - z_{-1}}\right) + \alpha_{2,1}u_0^p = g_1,$$

which may be rearranged as

$$u_{-1}^p = u_1^p + \frac{(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} u_0^p - \frac{(z_1 - z_{-1})g_1}{\alpha_{1,1}}. \tag{8.3}$$

Meanwhile, using the ghost node, the governing equation may be approximated, as in Equation 8.1, at the left-hand endpoint by

$$\begin{aligned} &-u_{-1}^{n+1} s_0^n \theta + u_0^{n+1}(1 + 2s_0^n\theta) + - u_1^{n+1} s_0^n \theta = \\ &\qquad = u_0^{n-1} + u_{-1}^n s_0^n(1-\theta) + u_0^n(1 - 2s_0^n(1-\theta)) + u_1^n s_0^n(1-\theta) + q_0^n, \end{aligned}$$

where the value of s_0^n is computed assuming that the space between z_{-1} and z_0 is the same as the space between z_0 and z_1. Substituting Equation 8.3 into this equation, we obtain

$$\begin{aligned} &-\left(u_1^{n+1} + \frac{(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} u_0^{n+1} - \frac{(z_1 - z_{-1})g_1}{\alpha_{1,1}}\right) s_0^n\theta + u_0^{n+1}(1 + 2s_0^n\theta) - u_1^{n+1}s_0^n\theta = \\ &\quad = u_0^{n-1} + \left(u_1^n + \frac{(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} u_0^n - \right. \\ &\quad \left. - \frac{(z_1 - z_{-1})g_1}{\alpha_{1,1}}\right) s_0^n(1-\theta) + u_0^n(1 - 2s_0^n(1-\theta)) + u_1^n s_0^n(1-\theta) + q_0^n, \end{aligned}$$

which may be rearranged as

$$\begin{aligned} &u_0^{n+1}\left(1 + 2s_0^n\theta - \frac{s_0^n\theta(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}}\right) - u_1^{n+1}\,(2s_0^n\theta) = \\ &\quad = u_0^n\left(1 - 2s_0^n(1-\theta) + \frac{s_0^n(1-\theta)(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}}\right) + u_1^n\,(2s_0^n(1-\theta)) + q_0^n - \frac{s_0^n(z_1 - z_{-1})g_1}{\alpha_{1,1}}. \end{aligned}$$

The right-hand endpoint may be treated similarly; the second-order finite difference equation approximating the boundary condition from Equation 8.2 at time t_p and at the right-hand ($z = \ell_2$) endpoint is

$$\alpha_{1,2}\left(\frac{u_N^p - u_{N-2}^p}{z_N - z_{N-2}}\right) + \alpha_{2,2}u_{N-1}^p = g_2,$$

which may be rearranged as

$$u_N^p = u_{N-2}^p + \frac{(z_N - z_{N-2})g_2}{\alpha_{1,2}} - \frac{(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}} u_{N-1}^p. \tag{8.4}$$

Using the ghost node z_N, the governing equation may be approximated, as in Equation 8.1, at the right-hand endpoint by

$$\begin{aligned} &-u_{N-2}^{n+1}s_{N-1}^n\theta + u_{N-1}^{n+1}(1 + 2s_{N-1}^n\theta) - u_N^{n+1}s_{N-1}^n\theta = \\ &\quad = u_{N-2}^n s_{N-1}^n(1-\theta) + u_{N-1}^n(1 - 2s_{N-1}^n(1-\theta)) + u_N^n s_{N-1}^n(1-\theta) + q_{N-1}^n, \end{aligned}$$

where s_{N-1} is computed assuming the space between z_N and z_{N-1} to be the same as the space between z_{N-1} and z_{N-2}. Equation 8.4 may be substituted to obtain

$$\begin{aligned}
&-u_{N-2}^{n+1}s_{N-1}^{n}\theta + u_{N-1}^{n+1}(1+2s_{N-1}^{n}\theta) - s_{N-1}^{n}\theta\Bigg(u_{N-2}^{n+1} + \frac{(z_N - z_{N-2})g_2}{\alpha_{1,2}} - \\
&-\frac{(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}u_{N-1}^{n+1}\Bigg) = \\
&=u_{N-2}^{n}s_{N-1}^{n}(1-\theta) + u_{N-1}^{n}(1-2s_{N-1}^{n}(1-\theta)) + \\
&+s_{N-1}^{n}(1-\theta)\left(u_{N-2}^{n} + \frac{(z_N - z_{N-2})g_2}{\alpha_{1,2}} - \frac{(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}u_{N-1}^{n}\right) + q_{N-1}^{n},
\end{aligned}$$

which may be rearranged as

$$\begin{aligned}
&-2u_{N-2}^{n+1}s_{N-1}^{n}\theta + u_{N-1}^{n+1}\left(1 + 2s_{N-1}^{n}\theta + \frac{s_{N-1}^{n}\theta(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right) = 2u_{N-2}^{n}s_{N-1}^{n}(1-\theta) + \\
&+u_{N-1}^{n}\left(1 - 2s_{N-1}^{n}(1-\theta) - \frac{s_{N-1}^{n}(1-\theta)(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right) + \frac{s_{N-1}^{n}(z_N - z_{N-2})g_2}{\alpha_{1,2}} + q_{N-1}^{n}.
\end{aligned} \tag{8.5}$$

The initial condition in Equation 3.10 is equivalent to setting

$$u_j^0 \equiv T_0 \quad \text{for} \quad j \in [0, N-1] \cap \mathbb{N}. \tag{8.6}$$

Now, Equations 8.1,8.1, and 8.5 may be cast as a linear system with the unknown values being

u_j, $j \in [0, N-1] \cap \mathbb{N}$, and this linear system may be written as the matrix equation

$$\begin{bmatrix} a_{0,0} & -2s_0^n\theta & 0 & \dots & \dots & 0 \\ -s_1^n\theta & 1+2s_1^n\theta & -s_1^n\theta & 0 & \dots & \vdots \\ 0 & -s_2^n\theta & 1+2s_2^n\theta & -s_2^n\theta & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & -s_{N-2}^n\theta & 1+2s_{N-2}^n\theta & -s_{N-2}^n\theta \\ 0 & \dots & \dots & 0 & -2s_{N-1}^n\theta & a_{N-1,N-1} \end{bmatrix} \begin{bmatrix} u_0^{n+1} \\ u_1^{n+1} \\ \vdots \\ u_j^{n+1} \\ \vdots \\ u_{N-1}^{n+1} \end{bmatrix} =$$

$$= \begin{bmatrix} b_{0,0} & 2s_0^n(1-\theta) & 0 & \dots & \dots & 0 \\ s_1^n(1-\theta) & 1-2s_1^n(1-\theta) & s_1^n(1-\theta) & 0 & \dots & \vdots \\ 0 & s_2^n(1-\theta) & 1-2s_2^n(1-\theta) & s_2^n(1-\theta) & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & s_{N-2}^n(1-\theta) & 1-2s_{N-2}^n(1-\theta) & s_{N-2}^n(1-\theta) \\ 0 & \dots & \dots & 0 & 2s_{N-1}^n(1-\theta) & b_{N-1,N-1} \end{bmatrix} \begin{bmatrix} u_0^n \\ u_1^n \\ \vdots \\ u_j^n \\ \vdots \\ u_{N-1}^n \end{bmatrix} +$$

$$+ \begin{bmatrix} q_0^n - \frac{s_0^n(z_1-z_{-1})g_1}{\alpha_{1,1}} \\ q_1^n \\ q_2^n \\ \vdots \\ q_j \\ \vdots \\ q_{N-2}^n \\ q_{N-1}^n + \frac{s_{N-1}^n(z_N-z_{N-2})g_2}{\alpha_{1,2}} \end{bmatrix}, \tag{8.7}$$

where we have used the shorthand values

$$a_{0,0} = 1 + 2s_0^n\theta - \frac{s_0^n\theta(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}}, \qquad a_{N-1,N-1} = 1 + 2s_{N-1}^n\theta + \frac{s_{N-1}^n\theta(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}},$$

$$b_{0,0} = 1 - 2s_0^n(1-\theta) + \frac{s_0^n(1-\theta)(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}}, \quad b_{N-1,N-1} = 1 - 2s_{N-1}^n(1-\theta) - \frac{s_{N-1}^n(1-\theta)(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}.$$

Equation 8.7 is used, together with the initial condition

$$\begin{bmatrix} u_0^0 \\ u_1^0 \\ \vdots \\ u_{N-1}^0 \end{bmatrix} = \begin{bmatrix} T_0 \\ \vdots \\ T_0 \end{bmatrix},$$

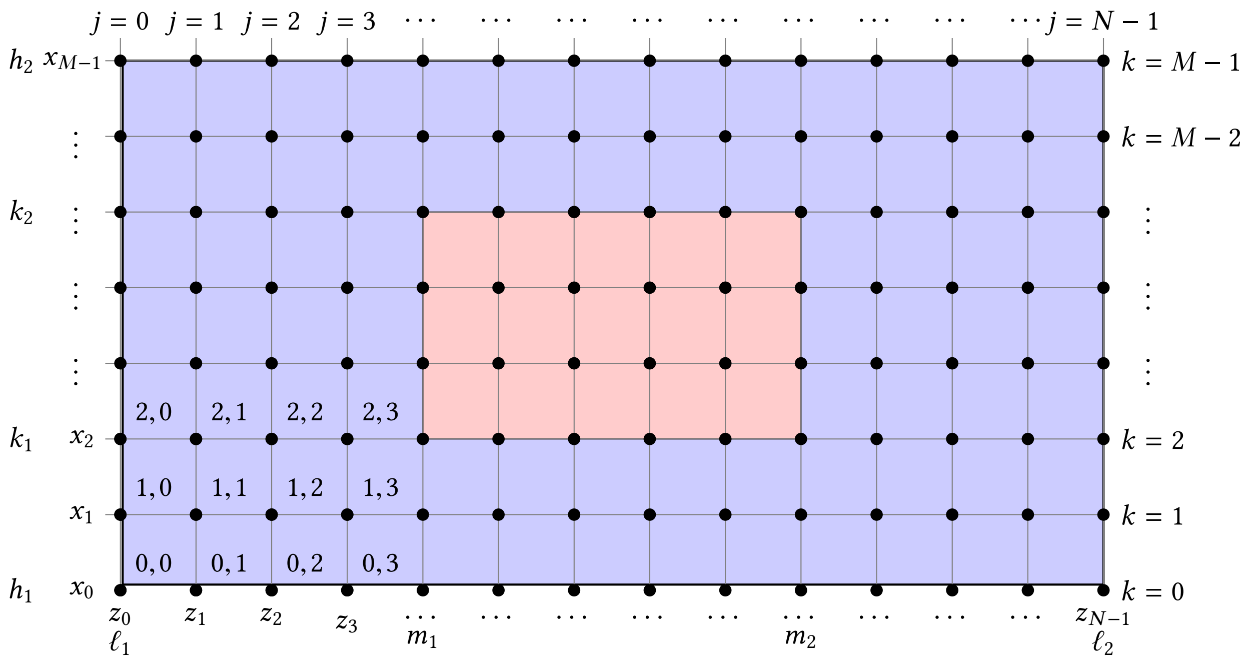


Figure 8.5: Discretization of the spatial domain for the finite difference solution of the two-dimensional heat equation. Area in blue is occupied by insulation, and area in red is occupied by material.

derived from Equation 8.6, to solve the heat equation at each time step.

8.2 Techniques for Solving the Two-Dimensional Heat Equation

Finite Difference Methods

For the two-dimensional case, we use a time-marching scheme with time steps given by Δt_u, denoted here Δt, as in the one-dimensional case. The domain is discretized into a spatial grid whose components are not necessarily of uniform length or height, and this discretization is the same one used in Figure 7.7. However, as discussed in Section 3.5, we exclude the portions of the computational domain that do not contain sample material or insulation, and so the size of the computational domain is decreased; we label the first z-node as z_0 and the last as z_{N-1}, so that the z-dimension contains N many nodes and $N-1$ many intervals, and similarly, we label the first x-node as x_0 and the last as x_{M-1}, so that the x-dimension contains M many nodes and $M-1$ many intervals. Note that this labelling scheme for the spatial nodes is different than the one introduced in Section 7.2 for the solution of the wave equation, although the nodes themselves also appear in the computational domain of that larger scheme. The nodes and their numbering scheme are shown in Figure 8.5.

Let $u^n_{j,k} := u(x_k, z_j, t_n)$, for $j \in [0, N-1] \cap \mathbb{N}$, $k \in [0, M-1] \cap \mathbb{N}$, and $n \in \{0\} \cup \mathbb{N}$. Consider the θ-scheme that uses difference approximations

$$\left.\frac{\partial u}{\partial t}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx \frac{u^{n+1}_{j,k} - u^n_{j,k}}{\Delta t},$$

$$\left.\frac{\partial u}{\partial z^2}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx (1-\theta)\frac{u^n_{j-1,k} - 2u^n_{j,k} + u^n_{j+1,k}}{(z_{j+1}-z_j)(z_j - z_{j-1})} + \theta\frac{u^{n+1}_{j-1,k} - 2u^{n+1}_{j,k} + u^{n+1}_{j+1,k}}{(z_{j+1}-z_j)(z_j - z_{j-1})}, \quad \text{and}$$

$$\left.\frac{\partial u}{\partial x^2}\right|_{\substack{x=x_k\\ z=z_j\\ t=t_n}} \approx (1-\phi)\frac{u^n_{j,k-1} - 2u^n_{j,k} + u^n_{j,k+1}}{(x_{k+1}-x_k)(x_k - x_{k-1})} + \phi\frac{u^{n+1}_{j,k-1} - 2u^{n+1}_{j,k} + u^{n+1}_{j,k+1}}{(x_{k+1}-x_k)(x_k - x_{k-1})},$$

where $\theta, \phi \in [0,1]$, $j \in [1, N-2] \cap \mathbb{N}$, $k \in [1, M-2] \cap \mathbb{N}$, and $n \in \mathbb{N}$. As in the one-dimensional case, the fully explicit FTCS (forward in time, centered in space) method corresponds to taking $\theta = \phi = 0$, the fully implicit BTCS (backward in time, centered in space) method corresponds to taking $\theta = \phi = 1$, and the Crank-Nicolson scheme corresponds to $\theta = \phi = 0.5$. For $\theta = \phi = 0.5$, the scheme is second-order accurate, but for any other value of θ, ϕ, the scheme is only first-order accurate [147]. For $\theta, \phi \geq 0.5$, the scheme is unconditionally stable, but for $\theta, \phi < 0.5$, the scheme is stable only when, for all time steps $n \in \mathbb{N}$ and for all spatial steps $j \in [1, N-2] \cap \mathbb{N}$, $k \in [1, M-2] \cap \mathbb{N}$

$$\frac{\Delta t}{(z_{j+1}-z_j)(z_j - z_{j-1})} \leq \frac{1}{2-4\theta} \quad \text{and} \quad \frac{\Delta t}{(x_{k+1}-x_k)(x_k - x_{k-1})} \leq \frac{1}{2-4\phi}.$$

Substituting the difference approximations into the governing equation of Equation 3.15, we obtain

$$\rho c_p \left(\frac{u^{n+1}_{j,k} - u^n_{j,k}}{\Delta t}\right) - k\left((1-\phi)\frac{u^n_{j,k-1} - 2u^n_{j,k} + u^n_{j,k+1}}{(x_{k+1}-x_k)(x_k - x_{k-1})} + \phi\frac{u^{n+1}_{j,k-1} - 2u^{n+1}_{j,k} + u^{n+1}_{j,k+1}}{(x_{k+1}-x_k)(x_k - x_{k-1})}\right) -$$
$$- k\left((1-\theta)\frac{u^n_{j-1,k} - 2u^n_{j,k} + u^n_{j+1,k}}{(z_{j+1}-z_j)(z_j - z_{j-1})} + \theta\frac{u^{n+1}_{j-1,k} - 2u^{n+1}_{j,k} + u^{n+1}_{j+1,k}}{(z_{j+1}-z_j)(z_j - z_{j-1})}\right) = \omega\varepsilon''|\vec{E}|^2_{\text{avg}}.$$

Arranging the equation so that the "unknown" values at the $(n+1)^{\text{st}}$ time step are on the left-hand side, and all "known" values are on the right-hand side, we obtain

$$\begin{aligned}
&u_{j,k}^{n+1}\left(\frac{\rho c_p}{\Delta t}+\frac{2k\phi}{(x_{k+1}-x_k)(x_k-x_{k-1})}+\frac{2k\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right)-\\
&\quad-u_{j,k-1}^{n+1}\frac{k\phi}{(x_{k+1}-x_k)(x_k-x_{k-1})}-u_{j,k+1}^{n+1}\frac{k\phi}{(x_{k+1}-x_k)(x_k-x_{k-1})}-\\
&\quad-u_{j-1,k}^{n+1}\frac{k\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}-u_{j+1,k}^{n+1}\frac{k\theta}{(z_{j+1}-z_j)(z_j-z_{j-1})}=\\
&\quad=u_{j,k}^{n}\left(\frac{\rho c_p}{\Delta t}+\frac{2k(1-\phi)}{(x_{k+1}-x_k)(x_k-x_{k-1})}+\frac{2k(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}\right)+\\
&\quad+u_{j,k-1}^{n}\frac{k(1-\phi)}{(x_{k+1}-x_k)(x_k-x_{k-1})}+u_{j,k+1}^{n}\frac{k(1-\phi)}{(x_{k+1}-x_k)(x_k-x_{k-1})}+\\
&\quad+u_{j-1,k}^{n}\frac{k(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}+u_{j+1,k}^{n}\frac{k(1-\theta)}{(z_{j+1}-z_j)(z_j-z_{j-1})}+\omega\varepsilon''|\vec{E}|_{\text{avg}}^2,
\end{aligned}$$

which may be rewritten using the shorthand abbreviations

$$\begin{aligned}
r_k &:= \frac{k(2\Delta t)}{\rho c_p(x_{k+1}-x_k)(x_k-x_{k-1})},\\
s_j &:= \frac{k(2\Delta t)}{\rho c_p(z_{j+1}-z_j)(z_j-z_{j-1})},\qquad \text{and}\\
q_{j,k}^n &:= \omega\varepsilon''|\vec{E}|_{\text{avg}}^2\frac{(2\Delta t)}{\rho c_p}
\end{aligned}$$

as

$$\begin{aligned}
&u_{j,k}^{n+1}\left(1+2\phi r_k+2\theta s_j\right)-u_{j,k-1}^{n+1}(\phi r_k)-u_{j,k+1}^{n+1}(\phi r_k)-u_{j-1,k}^{n+1}(\theta s_j)-u_{j+1,k}^{n+1}(\theta s_j)=\\
&\quad=u_{j,k}^{n}\left(1+2(1-\phi)r_k+2(1-\theta)s_j\right)+u_{j,k-1}^{n}((1-\phi)r_k)+u_{j,k+1}^{n}((1-\phi)r_k)+\\
&\quad+u_{j-1,k}^{n}((1-\theta)s_j)+u_{j+1,k}^{n}((1-\theta)s_j)+q_{j,k}^n,
\end{aligned}\tag{8.8}$$

To solve for the unknown values $u_{0,k}^{n+1}$, $u_{N-1,k}^{n+1}$, $u_{j,0}^{n+1}$, and $u_{j,M-1}^{n+1}$ we consider the general boundary conditions

$$\alpha_{1,1}\left.\frac{\partial u}{\partial z}\right|_{z=\ell_1}+\alpha_{2,1}u(x,\ell_1,t)=g_1,\qquad \alpha_{1,2}\left.\frac{\partial u}{\partial z}\right|_{z=\ell_2}+\alpha_{2,2}u(x,\ell_2,t)=g_2,\tag{8.9}$$

$$\alpha_{1,3}\left.\frac{\partial u}{\partial x}\right|_{x=h_1}+\alpha_{2,3}u(h_1,z,t)=g_3,\qquad \alpha_{1,4}\left.\frac{\partial u}{\partial x}\right|_{x=h_2}+\alpha_{2,4}u(h_2,z,t)=g_4,\tag{8.10}$$

where $\alpha_{1,i}$, $\alpha_{2,i}$ and g_i are assumed constant. As before, the case when

$$\begin{cases}\alpha_{1,i}=0\\ \alpha_{2,i}=1\\ g_i=T_0,\end{cases}$$

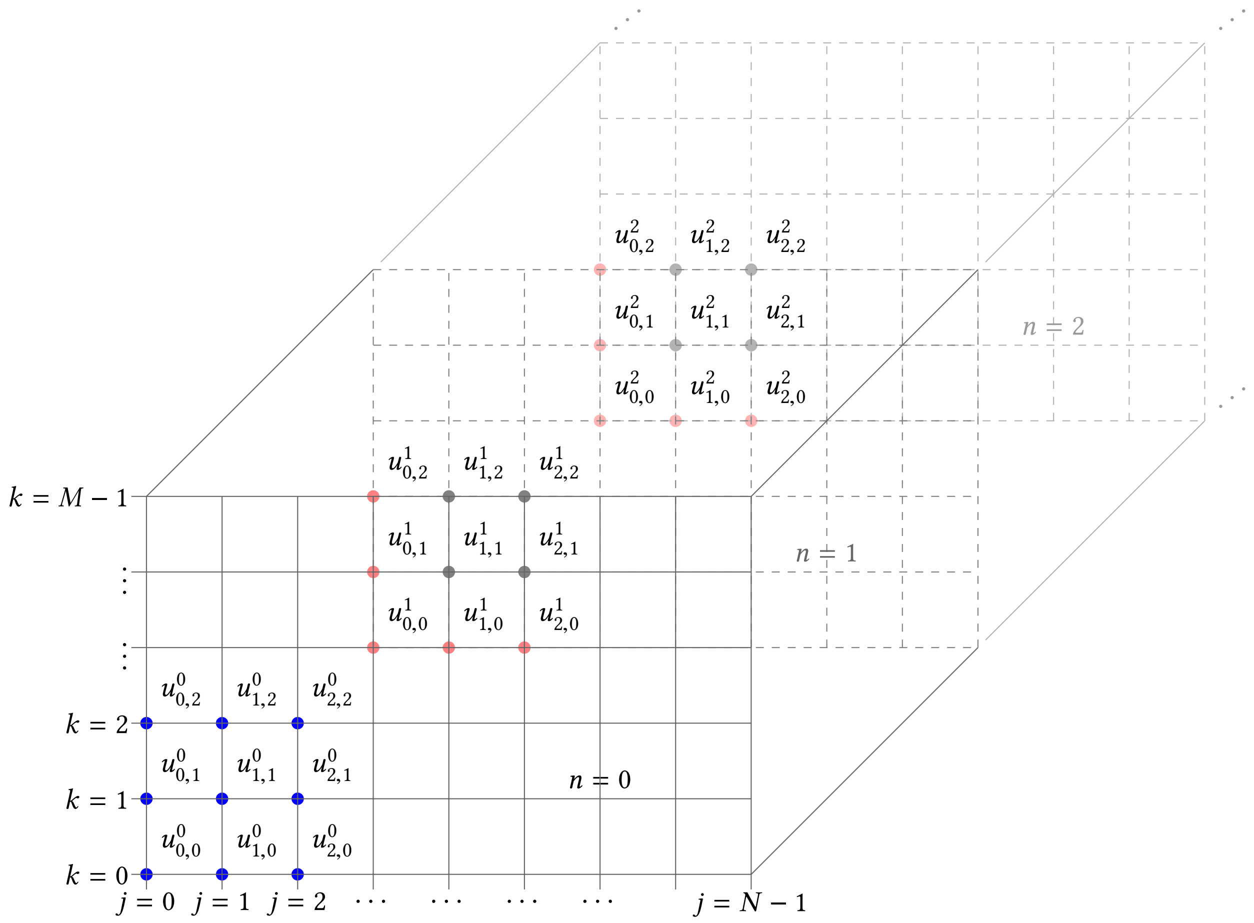

$u^2_{0,2}$ $u^2_{1,2}$ $u^2_{2,2}$
$u^2_{0,1}$ $u^2_{1,1}$ $u^2_{2,1}$
$u^2_{0,0}$ $u^2_{1,0}$ $u^2_{2,0}$
$n = 2$
$u^1_{0,2}$ $u^1_{1,2}$ $u^1_{2,2}$
$u^1_{0,1}$ $u^1_{1,1}$ $u^1_{2,1}$
$u^1_{0,0}$ $u^1_{1,0}$ $u^1_{2,0}$
$n = 1$
$u^0_{0,2}$ $u^0_{1,2}$ $u^0_{2,2}$
$u^0_{0,1}$ $u^0_{1,1}$ $u^0_{2,1}$
$u^0_{0,0}$ $u^0_{1,0}$ $u^0_{2,0}$
$n = 0$
$k = M - 1$
$k = 2$
$k = 1$
$k = 0$
$j = 0$ $j = 1$ $j = 2$ $\cdots$ $\cdots$ $\cdots$ $\cdots$ $j = N - 1$

Figure 8.6: Computational grids representing the solution space of the two-dimensional heat equation. Here, $j \in [0, N - 1] \cap \mathbb{N}$ and $k \in [0, M - 1] \cap \mathbb{N}$ represent the position within the spatial domain, and $n \in \{0\} \cup \mathbb{N}$ represents the time step. The blue-colored nodes represent those where the solution is given by the initial condition in Equation 8.25, and the red-colored nodes represent those whose solution is given by the boundary conditions in Equations 8.17, 8.16, 8.18, 8.22, 8.8, 8.23, 8.20, 8.19, and 8.21, while the solution at the black-colored nodes is given by Equation 8.8.

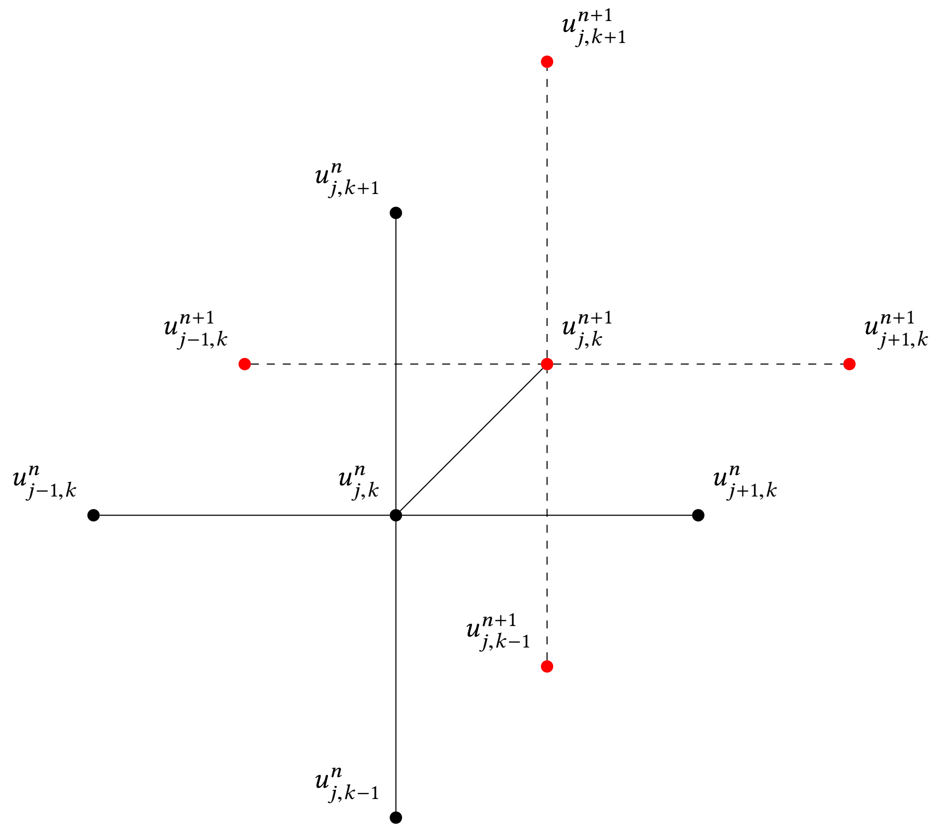


Figure 8.7: Computational stencil of θ-scheme for solving the two-dimensional heat equation. Here, Here, $j \in [1, N-2] \cap \mathbb{N}$ and $k \in [1, M-2] \cap \mathbb{N}$ represent the position within the spatial domain, and $n \in \mathbb{N}$ represents the current time step. The nodes in black are ones at which the solution u is known, and the ones in red may be solved for with knowledge of the ones in black.

where $i \in \{1, 2, 3, 4\}$, represents the Dirichlet condition in Equation 3.11, where the temperature on the corresponding border is explicitly fixed; the case when

$$\begin{cases} \alpha_{1,i} = 1 \\ \alpha_{2,i} = 0 \\ g_i = 0, \end{cases}$$

where $i \in \{1, 2, 3, 4\}$, represents the Neumann condition in Equation 3.12, where the heat flux is fixed; and the case where

$$\begin{cases} \alpha_{1,i} = 1 \\ \alpha_{2,i} = -h \\ g_i = -hT_{\text{amb}} \end{cases} \quad \text{and} \quad \begin{cases} \alpha_{1,j} = 1 \\ \alpha_{2,j} = h \\ g_j = hT_{\text{amb}}, \end{cases}$$

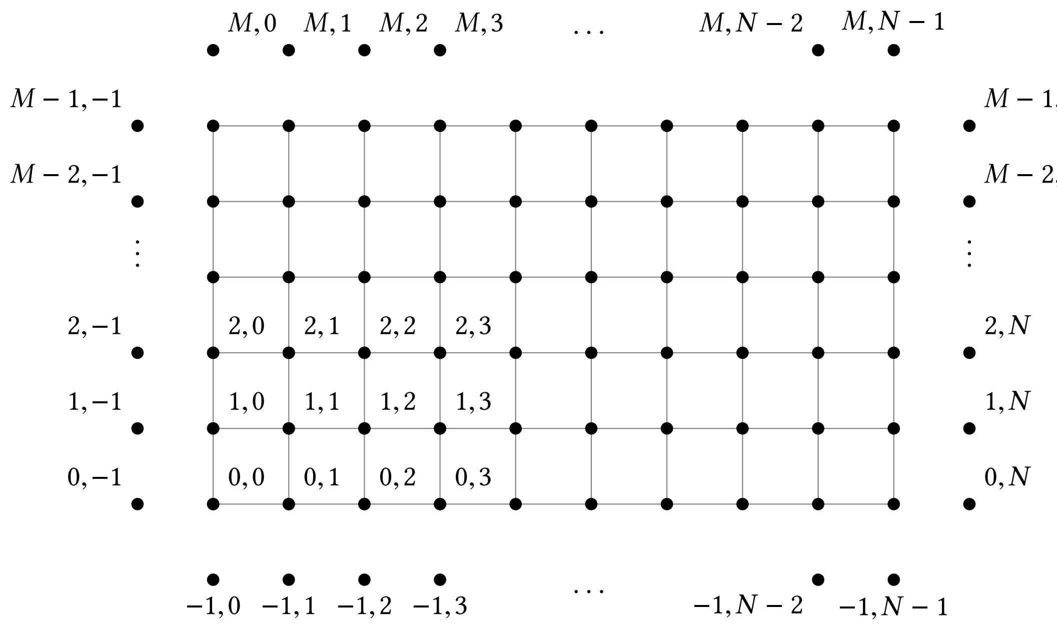


Figure 8.8: Discretization of the spatial domain for the finite difference solution of the two-dimensional heat equation. Area in blue is occupied by insulation, and area in red is occupied by material.

where $i \in \{1,3\}$ and $j \in \{2,4\}$, represents the radiative boundary condition discussed in Equation 3.13.

To implement these boundary conditions, similarly to the one-dimensional case in Section 8.1, we must temporarily assume the presence of the "ghost nodes" at $j=-1$, $j=N$, $k=-1$, and $k=M$, so that the solution on the boundaries may still be approximated using second-order difference formulas. The placement of these ghost nodes is shown in Figure 8.8.

Using the ghost nodes, the second-order finite difference equation approximating the boundary condition in Equation 8.9 for the $z=\ell_1$ boundary at time t_p, where $p>0$, is

$$\alpha_{1,1}\left(\frac{u^p_{1,k}-u^p_{-1,k}}{z_1-z_{-1}}\right)+\alpha_{2,1}u^p_{0,k}=g_1,$$

for $k \in [0,M-1]\cap\mathbb{N}$. This may be rearranged as

$$u^p_{-1,k}=u^p_{1,k}+\frac{(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}u^p_{0,k}-\frac{(z_1-z_{-1})g_1}{\alpha_{1,1}}. \tag{8.11}$$

Similarly, Equation 8.9 for the $z=\ell_2$ boundary yields

$$u^p_{N,k}=u^p_{N-2,k}-\frac{(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}u^p_{N-1,k}+\frac{(z_N-z_{N-2})g_2}{\alpha_{1,2}}, \tag{8.12}$$

for the $x = h_1$ boundary

$$u^p_{j,-1} = u^p_{j,1} + \frac{(x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}} u^p_{j,0} - \frac{(x_1 - x_{-1})g_3}{\alpha_{1,3}}, \tag{8.13}$$

and for the $x = h_2$ boundary,

$$u^p_{j,M} = u^p_{j,M-2} - \frac{(x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}} u^p_{j,M-1} + \frac{(x_M - x_{M-2})g_4}{\alpha_{1,4}}. \tag{8.14}$$

At the nodes where $j = 0$ (*i.e.*, where $z = \ell_1$), the governing equation may be approximated according to Equation 8.8 by

$$\begin{aligned} u^{n+1}_{0,k}\,(1 + 2\phi r_k + 2\theta s^n_0) - u^{n+1}_{0,k-1}(\phi r_k) - u^{n+1}_{0,k+1}(\phi r_k) - u^{n+1}_{-1,k}(\theta s^n_0) - u^{n+1}_{1,k}(\theta s^n_0) = \\ = u^n_{0,k}\,(1 + 2(1-\phi) r_k + 2(1-\theta) s^n_0) + u^n_{0,k-1}((1-\phi) r_k) + u^n_{0,k+1}(k(1-\phi) r_k) + \\ + u^n_{-1,k}((1-\theta) s^n_0) + u^n_{1,k}((1-\theta) s^n_0) + q^n_{0,k}, \end{aligned} \tag{8.15}$$

where the value of s^n_0 is computed assuming that the space between z_{-1} and z_0 is the same as the space between z_0 and z_1. For $k \in [1, M-2] \cap \mathbb{N}$, we substitute Equation 8.11 into Equation 8.15 to obtain

$$\begin{aligned} & u^{n+1}_{0,k}\,(1 + 2\phi r_k + 2\theta s^n_0) - u^{n+1}_{0,k-1}(\phi r_k) - u^{n+1}_{0,k+1}(\phi r_k) - \theta s^n_0 \left(u^{n+1}_{1,k} + \frac{(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} u^{n+1}_{0,k} - \right. \\ & \left. - \frac{(z_1 - z_{-1}) g_1}{\alpha_{1,1}} \right) - u^{n+1}_{1,k}(\theta s^n_0) = u^n_{0,k}\,(1 + 2(1-\phi) r_k + 2(1-\theta) s^n_0) + u^n_{0,k-1}((1-\phi) r_k) + u^n_{0,k+1}((1-\phi) r_k) + \\ & (1-\theta) s^n_0 \left(u^n_{1,k} + \frac{(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} u^n_{0,k} - \frac{(z_1 - z_{-1}) g_1}{\alpha_{1,1}} \right) + \\ & + u^n_{1,k}((1-\theta) s^n_0) + q^n_{0,k}, \end{aligned}$$

which may be rearranged as

$$\begin{aligned} u^{n+1}_{0,k} & \left(1 + 2\phi r_k + 2\theta s^n_0 - \frac{\theta s^n_0 (z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} \right) - u^{n+1}_{0,k-1}(\phi r_k) - u^{n+1}_{0,k+1}(\phi r_k) - 2\theta s^n_0 u^n_{1,k} \\ & = u^n_{0,k} \left(1 + 2(1-\phi) r_k + 2(1-\theta) s^n_0 + \frac{(1-\theta) s^n_0 (z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} \right) + u^n_{0,k-1}((1-\phi) r_k) + \\ & + u^n_{0,k+1}((1-\phi) r_k) + 2((1-\theta) s^n_0) u^n_{1,k} + q^n_{0,k} - \frac{s^n_0 (z_1 - z_{-1}) g_1}{\alpha_{1,1}}. \end{aligned} \tag{8.16}$$

For the case when $j = 0$ and $k = 0$, we substitute both Equations 8.11 and 8.13 into Equation 8.15 to obtain

$$\begin{aligned} &u_{0,0}^{n+1}\left(1+2\phi r_0^n+2\theta s_0^n\right)-\phi r_0^n\left(u_{0,1}^{n+1}+\frac{(x_1-x_{-1})\alpha_{2,3}}{\alpha_{1,3}}u_{0,0}^{n+1}-\frac{(x_1-x_{-1})g_3}{\alpha_{1,3}}\right)-u_{0,1}^{n+1}(\phi r_0^n)-\\ &\quad-\theta s_0^n\left(u_{1,0}^{n+1}+\frac{(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}u_{0,0}^{n+1}-\frac{(z_1-z_{-1})g_1}{\alpha_{1,1}}\right)-u_{1,0}^{n+1}(\theta s_0^n)=\\ &\qquad=u_{0,0}^{n}\left(1+2(1-\phi)r_0^n+2(1-\theta)s_0^n\right)+(1-\phi)r_0^n\left(u_{0,1}^{n}+\frac{(x_1-x_{-1})\alpha_{2,3}}{\alpha_{1,3}}u_{0,0}^{n}-\frac{(x_1-x_{-1})g_3}{\alpha_{1,3}}\right)+\\ &\qquad\quad+u_{0,1}^{n}((1-\phi)r_0^n)+(1-\theta)s_0^n\left(u_{1,0}^{n}+\frac{(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}u_{0,0}^{n}-\frac{(z_1-z_{-1})g_1}{\alpha_{1,1}}\right)+\\ &\qquad\qquad+u_{1,0}^{n}((1-\theta)s_0^n)+q_{0,0}^n, \end{aligned}$$

which may be rearranged as

$$\begin{aligned} &u_{0,0}^{n+1}\left(1+2\phi r_0^n+2\theta s_0^n-\frac{\phi r_0^n(x_1-x_{-1})\alpha_{2,3}}{\alpha_{1,3}}-\frac{\theta s_0^n(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}\right)-2\phi r_0^n u_{0,1}^{n+1}-2\theta s_0^n u_{1,0}^{n+1}=\\ &\quad=u_{0,0}^{n}\left(1+2(1-\phi)r_0^n+2(1-\theta)s_0^n+\frac{(1-\phi)r_0^n(x_1-x_{-1})\alpha_{2,3}}{\alpha_{1,3}}+\frac{(1-\theta)s_0^n(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}\right)+\\ &\qquad+2(1-\phi)r_0^n u_{0,1}^n+2(1-\theta)s_0^n u_{1,0}^n+q_{0,0}^n-\frac{r_0^n(x_1-x_{-1})g_3}{\alpha_{1,3}}-\frac{s_0^n(z_1-z_{-1})g_1}{\alpha_{1,1}}. \end{aligned} \tag{8.17}$$

Similarly, for the case when $j = 0$ and $k = M - 1$, we substitute both Equations 8.11 and 8.14 into Equation 8.15 and rearrange to obtain

$$\begin{aligned} &u_{0,M-1}^{n+1}\left(1+2\phi r_{M-1}^n+2\theta s_0^n+\frac{\phi r_{M-1}^n(x_M-x_{M-2})\alpha_{2,4}}{\alpha_{1,4}}-\frac{\theta s_0^n(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}\right)-\\ &-2\phi r_{M-1}^n u_{0,M-2}^{n+1}-2\theta s_0^n u_{1,M-1}^{n+1}=\\ &=u_{0,M-1}^{n}\left(1+2(1-\phi)r_{M-1}^n+2(1-\theta)s_0^n-\frac{(1-\phi)r_{M-1}^n(x_M-x_{M-2})\alpha_{2,4}}{\alpha_{1,4}}+\right.\\ &\left.+\frac{(1-\theta)s_0^n(z_1-z_{-1})\alpha_{2,1}}{\alpha_{1,1}}\right)+2(1-\phi)r_{M-1}^n u_{0,M-2}^n+2(1-\theta)s_0^n u_{1,M-1}^n+\\ &+q_{0,M-1}^n-\frac{r_{M-1}^n(x_M-x_{M-2})g_4}{\alpha_{1,4}}-\frac{s_0^n(z_1-z_{-1})g_1}{\alpha_{1,1}}. \end{aligned} \tag{8.18}$$

To model the right-hand boundary $z = \ell_2$, $j = N - 1$, we substitute Equation 8.12 into Equation 8.8 to obtain

$$\begin{aligned}
&u^{n+1}_{N-1,k}\left(1+2\phi r_k+2\theta s^n_{N-1}+\frac{\theta s^n_{N-1}(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right)-u^{n+1}_{N-1,k-1}(\phi r_k)-u^{n+1}_{N-1,k+1}(\phi r_k)-\\
&\quad-2\theta s^n_{N-1}u^n_{N-2,k}=u^n_{N-1,k}\left(1+2(1-\phi)r_k+2(1-\theta)s^n_{N-1}-\frac{(1-\theta)s^n_{N-1}(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right)+\\
&\quad+u^n_{N-1,k-1}((1-\phi)r_k)+u^n_{N-1,k+1}((1-\phi)r_k)+2((1-\theta)s^n_{N-1})u^n_{N-2,k}+\\
&\qquad+q^n_{N-1,k}+\frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}}.
\end{aligned}\tag{8.19}$$

For the case when $j=N-1$ and $k=0$, we substitute both Equations 8.12 and 8.13 into Equation 8.15 to obtain

$$\begin{aligned}
&u^{n+1}_{N-1,0}\left(1+2\phi r^n_0+2\theta s^n_{N-1}-\frac{\phi r^n_0(x_1-x_{-1})\alpha_{2,3}}{\alpha_{1,3}}+\frac{\theta s^n_{N-1}(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right)-\\
&\quad-2\phi r^n_0u^{n+1}_{N-1,1}-2\theta s^n_{N-1}u^{n+1}_{N-2,0}=\\
&\quad=u^n_{N-1,0}\left(1+2(1-\phi)r^n_0+2(1-\theta)s^n_{N-1}+\frac{(1-\phi)r^n_0(x_1-x_{-1})\alpha_{2,3}}{\alpha_{1,3}}-\frac{(1-\theta)s^n_{N-1}(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right)+\\
&\qquad+2(1-\phi)r^n_0u^n_{N-1,1}+2(1-\theta)s^n_{N-1}u^n_{N-2,0}+q^n_{N-1,0}-\frac{r^n_0(x_1-x_{-1})g_3}{\alpha_{1,3}}+\frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}}.
\end{aligned}\tag{8.20}$$

For the case when $j=N-1$ and $k=M-1$, we substitute both Equations 8.12 and 8.14 into Equation 8.15 and rearrange to obtain

$$\begin{aligned}
&u^{n+1}_{N-1,M-1}\left(1+2\phi r^n_{M-1}+2\theta s^n_{N-1}+\frac{\phi r^n_{M-1}(x_1-x_{-1})\alpha_{2,4}}{\alpha_{1,4}}+\frac{\theta s^n_{N-1}(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right)-\\
&\quad-2\phi r^n_{M-1}u^{n+1}_{N-1,M-2}-2\theta s^n_{N-1}u^{n+1}_{N-2,M-1}=\\
&\quad=u^n_{N-1,M-1}\left(1+2(1-\phi)r^n_{M-1}+2(1-\theta)s^n_{N-1}-\frac{(1-\phi)r^n_{M-1}(x_M-x_{M-2})\alpha_{2,4}}{\alpha_{1,4}}-\right.\\
&\quad\left.-\frac{(1-\theta)s^n_{N-1}(z_N-z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}\right)+2(1-\phi)r^n_{M-1}u^n_{N-1,M-2}+2(1-\theta)s^n_{N-1}u^n_{N-2,M-1}+\\
&\quad+q^n_{N-1,M-1}-\frac{r^n_{M-1}(x_M-x_{M-2})g_4}{\alpha_{1,4}}+\frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}}.
\end{aligned}\tag{8.21}$$

j	k	Equation
0	0	8.17
0	$1,\ldots,M-2$	8.16
0	$M-1$	8.18
$1,\ldots,N-2$	0	8.22
$1,\ldots,N-2$	$1,\ldots,M-2$	8.8
$1,\ldots,N-2$	$M-1$	8.23
$N-1$	0	8.20
$N-1$	$1,\ldots,M-2$	8.19
$N-1$	$M-1$	8.21

Table 8.1: Description of the organization of the linear system for the finite difference approximation of the two-dimensional heat equation.

At the lower boundary $x = h_1$, $k = 0$, we substitute Equation 8.13 into Equation 8.8 to obtain

$$\begin{aligned} u_{j,0}^{n+1}\left(1 + 2\phi r_0^n + 2\theta s_j - \frac{\phi r_0^n (x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}}\right) - 2u_{j,1}^{n+1}(\phi r_0^n) - u_{j-1,0}^{n+1}(\theta s_j) - u_{j+1,0}^{n+1}(\theta s_j) = \\ = u_{j,0}^n\left(1 + 2(1-\phi)r_0^n + 2(1-\theta)s_j + \frac{(1-\phi)r_0^n(x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}}\right) + 2u_{j,1}^n((1-\phi)r_0^n) + \\ + u_{j-1,0}^n((1-\theta)s_j) + u_{j+1,0}^n((1-\theta)s_j) + q_{j,0}^n - \frac{r_0^n(x_1 - x_{-1})g_3}{\alpha_{1,3}}, \end{aligned} \tag{8.22}$$

for $j \in [1, N-2] \cap \mathbb{N}$. Finally, at the upper boundary $x = h_2$, $k = M-1$, we substitute Equation 8.14 into Equation 8.8 to obtain

$$\begin{aligned} u_{j,M-1}^{n+1}\left(1 + 2\phi r_{M-1}^n + 2\theta s_j + \frac{\phi r_{M-1}^n (x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}}\right) - 2u_{j,M-2}^{n+1}(\phi r_{M-1}^n) - u_{j-1,M-1}^{n+1}(\theta s_j) - \\ - u_{j+1,M-1}^{n+1}(\theta s_j) = u_{j,M-1}^n\left(1 + 2(1-\phi)r_{M-1}^n + 2(1-\theta)s_j - \frac{(1-\phi)r_{M-1}^n(x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}}\right) + \\ + 2u_{j,M-2}^n((1-\phi)r_{M-1}^n) + u_{j-1,M-1}^n((1-\theta)s_j) + u_{j+1,M-1}^n((1-\theta)s_j) + \\ + q_{j,M-1}^n + \frac{r_{M-1}^n(x_M - x_{M-2})g_4}{\alpha_{1,4}}, \end{aligned} \tag{8.23}$$

for $j \in [1, N-2] \cap \mathbb{N}$.

A linear system may be established using the equations described above, with Table 8.1 summarizing which equation corresponds to each coordinate pair (j,k). As is clear, the system is one of $N \times M$ many equations in $N \times M$ many unknowns, and so we establish the matrix equation for its solution at each time step:

$$A\vec{u}^{n+1} = B\vec{u}^n + \vec{Q}^n, \tag{8.24}$$

where the vectors $\vec{u}^p$ and $\vec{Q}^p$, and the matrices A and B together with their entries, are shown in Appendix B. The starting value of temperature is assumed to be the room temperature, so that

$$\vec{u}^0_{j,k} \equiv T_{\text{amb}}, \tag{8.25}$$

where T_{amb} represents the ambient temperature.

The computer implementation of this code is shown in Appendix E.

Chapter 9

Numerical Techniques for Solving the Mechanical Deformation Problem

Sintering has been modelled on various spatial scales and by different methods depending on those scales [24, 30]. Microscale modelling of sintering, typically representing a heat source external to the object being sintered, has represented the mechanical changes of the sample using molecular dynamics [149] or various other analytical and numerical techniques including the discrete element method [150–153]. Such models can be loosely classified into those concerned with the early-to-intermediate stages of sintering, where interparticle behavior is dominant, and those concerned with late-stage sintering, which actively simulate not the particles, but the pores (*i.e.*, the spaces between particles) [154]. These models explicitly represent the current state of knowledge on the thermal dynamics of sintering, such as densification kinetics, influence of externally applied forces and structure heterogeneities [155]; however, many rely on simplifications, such as the assumption of uniform particle size and shape, that may not be valid in reality, making them of limited practical use for the powder metallurgy industry. Such industries are typically more concerned with accurately depicting macroscale evolution of components and parts, along with simulating their densities, rather than particle-pore interactions. On the macro scale, modelling has been done using the finite element method [24, 33]. Our focus is primarily limited to modelling sintering on a component scale, rather than on a molecular scale.

The component-scale models of sintering have traditionally represented an external heat source, assuming that the temperature on the boundary of the material is the only factor that dictates the interior temperature of the material. These solvers typically use the finite element method for numerical simulation, and may rely on software packages such as Abaqus [24, 156, 157], ANSYS [158–160], or COMSOL Multiphysics [161–164] to implement the continuum mechanical simulations together with a constitutive equation that determines evolution of the material properties and the geometrical configuration of the sample.

Component-scale models, including the one first described by Riedel and Blug [33], may employ microstructure variables and obey the law for grain coarsening, and their characterization of sintering progress relies on a constitutive equation such as the one in Equation 4.4; this equation may be used as input for routines driven by commercially available software packages such as Abaqus, in which simple formulations may rely on the CREEP subroutine, and more complex formulations rely on the UMAT subroutine [24, 79–82, 165]. Other models that explicitly employ microstructure variables can be found in [30, 149, 155].

In this work, we remain concerned with sintering on the component scale, but instead of employing the model in Equation 4.4, we make the simplifying assumption that sintering progress may be entirely characterized by evolution of density, which allows the simulation of mechanical deformation based on the principle of conservation of mass, and we therefore avoid the need to simulate the evolution of microstructural properties. This approach eliminates the need for numerical integration of the strain rate tensor, and therefore provides the possibility of more expedient simulation than the existing solvers [79–82, 165]. For typical material involved with industrial sinterforming processes, such as alumina and zirconia, density is, indeed, a good characterization of the process of sintering.

9.1 Master Sintering Curve Model

The basic formulation of the Master Sintering Curve theory was given in Section 4.3, and here, we describe its use in constructing an empirical relationship between a prescribed or simulated thermal cycle and the resulting relative density of a sample.

Equation 4.10 is a unique relationship for a given powder and sintering scenario, and it reveals that the evolution of relative density along the sintering path may be characterized as

$$\rho_{\mathrm{rel}} = \Psi^{-1}\left(\Theta(u, u(t))\right) =: \rho_{\mathrm{rel}}\left(\Theta(u, u(t))\right), \tag{9.1}$$

where the work of sintering parameter Θ is the one defined in Equation 4.9. The integral in Equation 4.9 may be easily computed for a given thermal cycle, provided that the activation energy Q is known or may be easily approximated [74].

One of the limitations of the MSC is its dependence on Q, because this parameter is highly variable not only among materials, but among different preparations of green-body samples of even the same material; indeed, different powders produced by green-body processing methods such as milling or pressing may result in different particle sizes and size distributions, different initial pore sizes, and different packing properties, all of which affect densification behavior [74]. The MSC can, however, be used to determine activation energy, as described in Subsection 9.1.

Form and Construction of the MSC

When values of relative density are computed[1] over the course of an experiment where the heating cycle is known and Θ-values are obtained from Equation 4.9, then the curve defined in Equation 9.1 is the one traced out when the pairs of Θ and ρ_{rel} are plotted as points on a coordinate plane.

[1]Relative density values are typically computed using recorded data on shrinkage obtained using in-situ dilatometry.

In [86], the idea of interpolating these points to construct a piecewise polynomial approximation of ρ_{rel} as a function of $\ln(\Theta)$ was presented, and later in [3], a sigmoid-type curve was proposed as a method of obtaining a more accurate fit to the data. The following model appears in [3] and has been successfully used to model both conventional and microwave sintering of alumina and zirconia composites [166–168].

$$\rho = \rho_0 + \frac{a}{\left[1 + \exp\left(-\frac{\ln\Theta - \ln\Theta_0}{b}\right)\right]^c}. \tag{9.2}$$

An alternative formulation for the sigmoid curve with adjustable parameters is the one given in [169–172] as

$$\rho = \rho_0 + \frac{1 - \rho_0}{1 + \exp\left[-\frac{\ln\Theta - a}{b}\right]}. \tag{9.3}$$

Our implementation, shown in Appendix F.2, is capable of constructing a best fit to either of these sigmoid curve models using Levenberg-Marquardt multiparameter optimization [173, 174]. The curves obtained using data from [3], together with an assumed activation energy of $Q = 660.1$ kJ/mol, are shown in Figure 9.1, and these show good agreement with the sigmoid curve computed in [3]. In the case where we use the sigmoid function representation from Equation 9.2, the optimal-fit sigmoid function was

$$\rho_{\text{rel}} = 0.525624 + \frac{0.464563}{\left[1 + \exp\left(-\frac{\ln\Theta - (-51.4957)}{1.68507}\right)\right]^{0.630385}}, \tag{9.4}$$

and in the case where we use the sigmoid function representation from Equation 9.3, the optimal-fit sigmoid function was

$$\rho_{\text{rel}} = 0.536209 + \frac{(1 - 0.536209)}{[1 + \exp(-\frac{\ln\Theta - (-52.5271)}{2.01868})]}. \tag{9.5}$$

The data for the three sintering trials described in [3] is plotted in Figure 9.2. The curves obtained using data from [4], together with an assumed activation energy of $Q = 660.1$ kJ/mol, are shown in Figure 9.3, and these show good agreement with the sigmoid curve computed in [4]. In the case where we use the sigmoid function representation from Equation 9.2, the optimal-fit sigmoid function was

$$\rho_{\text{rel}} = 0.461737 + \frac{0.428151}{\left[1 + \exp\left(-\frac{\ln\Theta - (-54.3133)}{1.17888}\right)\right]^{0.439561}}, \tag{9.6}$$

and in the case where we use the sigmoid function representation from Equation 9.3, the optimal-fit sigmoid function was

$$\rho_{\text{rel}} = 0.427313 + \frac{(1 - 0.427313)}{[1 + \exp(-\frac{\ln\Theta - (-55.0611)}{3.31359})]}. \tag{9.7}$$

The data for the three sintering trials described in [4] is plotted in Figure 9.4.

Computing the Activation Energy using the MSC

If the activation energy Q is not known for a particular preparation of material, then it may be approximated using the MSC [74].

Initially, an estimate of Q is made, and the Θ-values for several sintering experiments are computed using this value of Q; the process in Subsection 9.1 is carried out to construct the sigmoid function whose graph optimally approximates the curve defined by the $(\ln\Theta, \rho_{\text{rel}})$ data points, and the error in this optimal fit is computed as a sum of the squares of the differences between the ρ_{rel} values and the sigmoid function value at the corresponding $\ln\Theta$-values.

Another value of Q is chosen, and the process outlined above is repeated to find the corresponding optimal sigmoid function and the resulting sum of squared residuals. The best estimate of Q is the one that minimizes this sum of squared residuals, and in our implementation, we use Nelder-Mead optimization [173, 174] to find the best Q-value. Codes carrying out this optimization can be seen in Appendix F.2.

Quicker Computation of Θ for Constant Heating Rate

Many sintering experiments are conducted at constant rates of heating, which appears to diminish the inhibiting effect of surface diffusion on the process of sintering during its advanced stages [74]. In this case, there exists a closed-form elementary integral for computing $\Theta(t, T(t))$, which may help to reduce computation time during in-situ control routines, and during the optimization process used in finding Q.

In particular, assume that the heating rate is constant α—that is,

$$u(t) = \alpha t + u_0$$

Substituting this into Equation (4.9) yields

$$\Theta(t, u(t)) = \int_0^t \frac{1}{\alpha\tau + u_0} \exp\left(\frac{-Q}{R\alpha\tau + Ru_0}\right) \,\mathrm{d}\tau,$$

to which applying the substitution

$$\begin{cases} u := \frac{Q}{R\alpha\tau + Ru_0} \iff \tau := \frac{Q - uRu_0}{R\alpha u} \\ \mathrm{d}u := -\frac{Q\alpha}{R(\alpha\tau + u_0)^2}\mathrm{d}\tau \iff \mathrm{d}\tau := -\frac{Q}{R\alpha u^2}\mathrm{d}u \end{cases}$$

yields

$$\Theta(t, u(t)) = -\frac{1}{R\alpha} \int_{\frac{Q}{Ru_0}}^{\frac{Q}{R(\alpha t + u_0)}} \frac{e^{-u}}{u} \,\mathrm{d}u. \tag{9.8}$$

This is a special case of the exponential integral

$$\mathrm{Ei}(x) := -\int_{-x}^{+\infty} \frac{e^{-t}}{t} \,\mathrm{d}t,$$

a non-elementary function whose value for positive x is typically understood in terms of the Cauchy principal value by taking a branch cut along the negative real axis; the curve that results from evaluating this non-elementary integral is shown in [175].

The exponential integral is included as a function in the `scipy.special` module of `python`, as well as with many other scientific computing softwares, such as `MATLAB` (via the built-in function `expint`), Wolfram Mathematica (via `ExpIntegralEi`), and `fortran` (via the routines in [176]), which makes its computation in numerical routines especially simple. An example of the function values drawn using the `python` implementation is shown in Figure 9.5, and these function values are identical to the ones shown in [175].

For negative real values of x, which lie directly on the branch cut and for which $\mathrm{Ei}(x)$ cannot be computed in the traditional way, $\mathrm{Ei}(x)$ is computed using the relation

$$\lim_{\varepsilon \to 0^+} \mathrm{E}_1(-x \pm i\varepsilon) = -\mathrm{Ei}(x) \mp i\pi, \tag{9.9}$$

where $\mathrm{E}_1(x) := \Gamma(0, x) := \int_x^\infty \frac{e^{-u}}{u}\, \mathrm{d}u$ is also referred to as an exponential integral function, and $\Gamma(a, z)$ is an incomplete gamma function [175]. The `scipy.special` module computes the exponential integral $\mathrm{Ei}(x)$ for negative values of x using Equation 9.9. With this method, one may determine the Θ values for a given constant-rate heating experiment knowing only the rate constant, and the times at which the measurements of density were taken. Crucially, this method avoids the use of alternative techniques of numerical integration, such as the trapezoidal rule. A short `python` script testing the expediency of this method is given in Appendix F.1, and its results show that the exponential integral method runs in 60% of the time that the cumulative integral function in the `scipy.integrate` module for `python` requires.

The use of the exponential integral expression for constant heating rates in non-isothermal kinetics, such as in the theory of thermogravimetry, thermal desorption, and thermoluminescence, has been studied in [177, 178], and alternative methods for the approximation and solution of the exponential integral equation have been shown in [179–182]. However, this appears to be the first time such an expression has been proposed for speeding up the optimization routine for finding the activation energy of a material using experimental measurements of sintering data.

9.2 Conservation of Mass and Relative Density-Based Model

A simple model of mechanical deformation is appropriate for our situation, where the initial distribution of material parameters within the material being processed is assumed to be uniform, and that material is assumed to be anisotropic. In most scenarios of this type, we do not expect to have nonuniform stresses and strains on the material, and we may rely on conservation of mass and a computation of relative density change to simulate mechanical deformation, rather than on the constitutive relation given in Equation 4.4.

Namely, we use the curve found by method described in Section 9.1 to compute the change in relative density within each cell of the one- or two-dimensional computational domain. These density changes are averaged over the entire sample, and we assume that the law of conservation of mass holds, so that, where m refers to mass, v_{old} and v_{new} refer to the initial and updated volumes, and ρ_{old} and ρ_{new} refer to the initial and updated average absolute densities,

$$m = \rho_{\text{old}} \cdot v_{\text{old}} \quad \text{and} \quad m = \rho_{\text{new}} \cdot v_{\text{new}},$$

which leads to

$$v_{\text{new}} = \frac{\rho_{\text{old}}}{\rho_{\text{new}}} v_{\text{old}}.$$

In the one-dimensional case, the right-hand endpoints of both insulation and material are assumed to be fixed during processing, as is the amount of insulation to the left and right of the material; the material, however, changes length as its density changes, and so, as the material shrinks (or undergoes slight thermal expansion), and as the insulation shifts to the right (or slightly to the left) to accommodate this material change, the left-hand endpoints of the material and the insulation move.

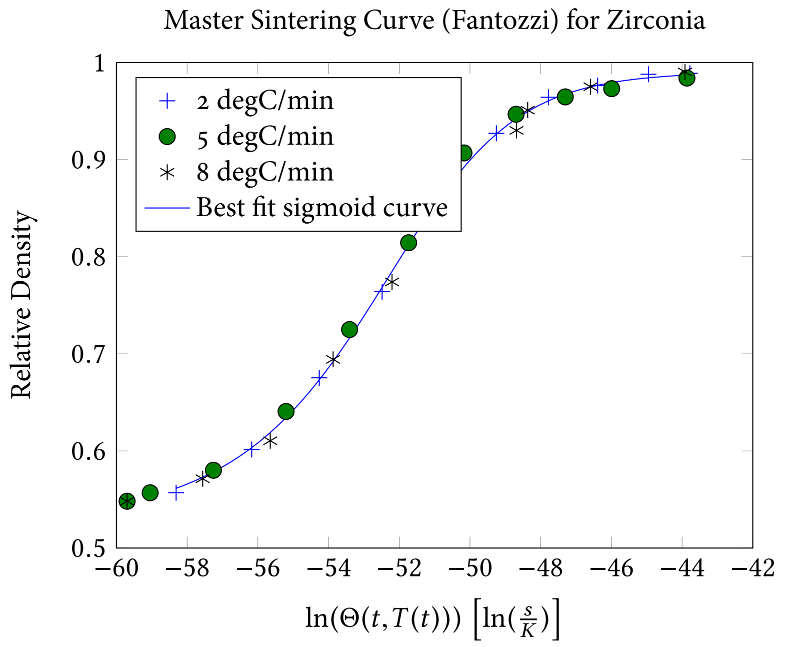
In the two-dimensional case, the upper boundaries of both insulation and material are the only ones that are not fixed during processing, and the amount of insulation surrounding the material is also fixed, except for the portions to the left and right of the material, residing above the material. As the material shrinks (or undergoes slight thermal expansion), the material's upper boundary falls (or slightly rises), and the insulation's upper boundary falls commensurately everywhere, including in the areas to the left and to the right of the material boundaries[2].

The computational grid on which the electromagnetic and thermal problems remains fixed throughout the simulation, and affects the capability of the model to account for material shrinkage, in the sense that each node in the grid is assigned a particular set of dielectric and thermal properties that are used in the solution of the governing equations. In case the amount of shrinkage predicted by the law of conservation of mass after a given thermal time step is greater than the length of the spatial cell immediately to the left of the location of the maximum density in the sample (in the one-dimensional case), or greater than the height of the row of spatial cells immediately above the location of the maximum density value in the sample (in the two-dimensional case), then a threshold is placed at the spatial node nearest to the location of maximum density minus the predicted amount of shrinkage (in the one-dimensional case), and at the row of spatial nodes nearest to the location of maximum density plus the predicted amount of shrinkage (in the two-dimensional case). Our model accounts for shrinkage by replacing the temperature, density, and material property values immediately to the left of the location of the maximum density by those to the left of the threshold, as shown in Figure 9.6.

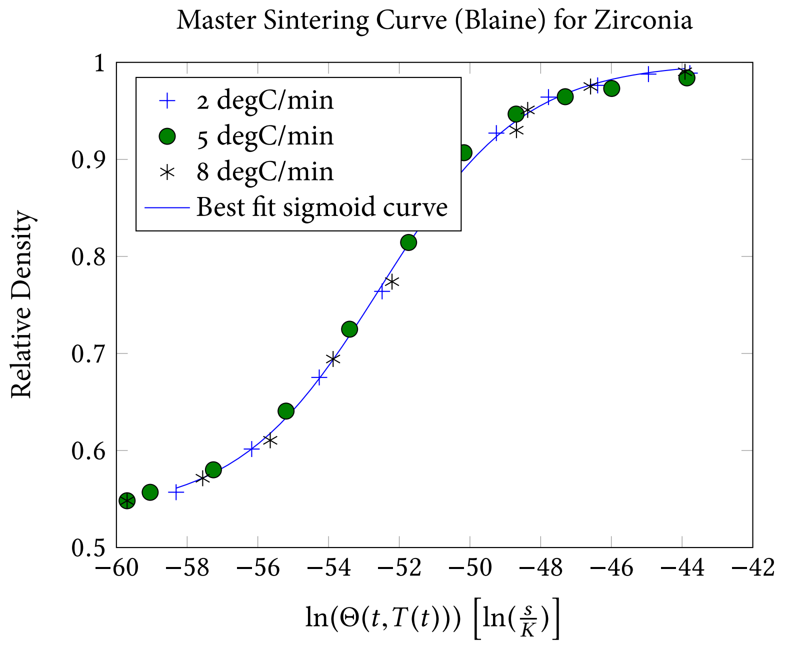
In case the predicted shrinkage of the material over a given thermal time step is less than the length of one spatial grid cell immediately adjacent to the position of maximum density, then the model has no way of accounting for the change in the material boundary position; in this case, the predicted shrinkage over this time step is set back to zero, and the relative density in the medium is set to that of the previous time step, so that in the event of further densification after future time

[2]This violates conservation of mass, but as we are not concerned with tracking the physical deformation of the insulation during processing, this does not affect the validity of the model.

steps, the accurate level of commensurate shrinkage may be executed, as described above and as shown in Figure 9.6.

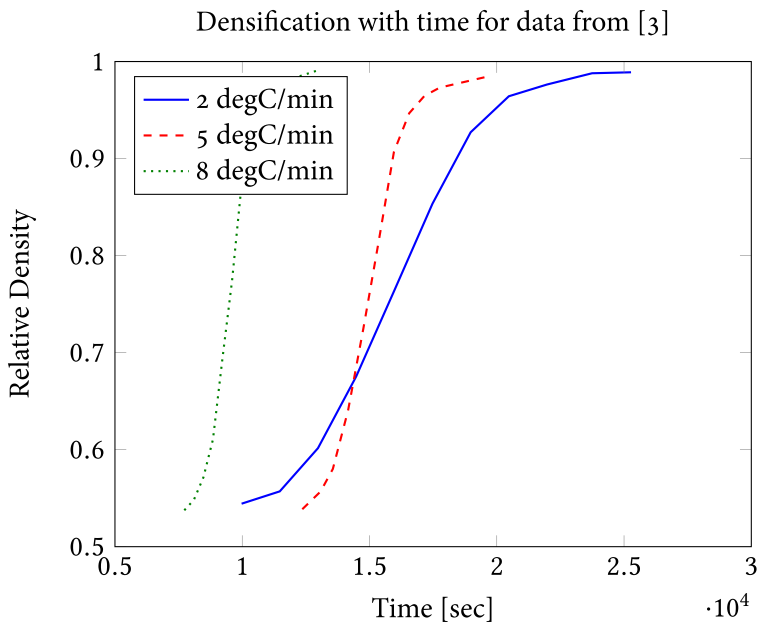


(a) Demonstration of sigmoid curve fitting using the model in Equation 9.3. Best fit was computed using the Levenberg-Marquardt optimization method, and has least-squares error 0.00144663.

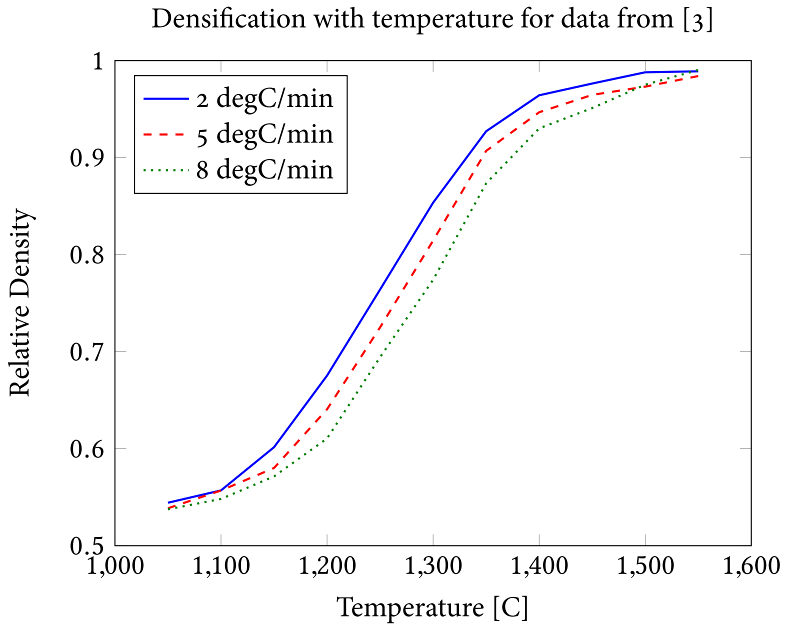


(b) Demonstration of sigmoid curve fitting using the model in Equation 9.2. Best fit was computed using the Levenberg-Marquardt optimization method, and has least-squares error 0.00116233.

Figure 9.1: Demonstration of the sigmoid curve-fitting methods carried out for zirconia data from [3] and an activation energy of $Q = 660.1$ kJ/mol. Optimal sigmoid functions are shown in Equations 9.4 and 9.5.

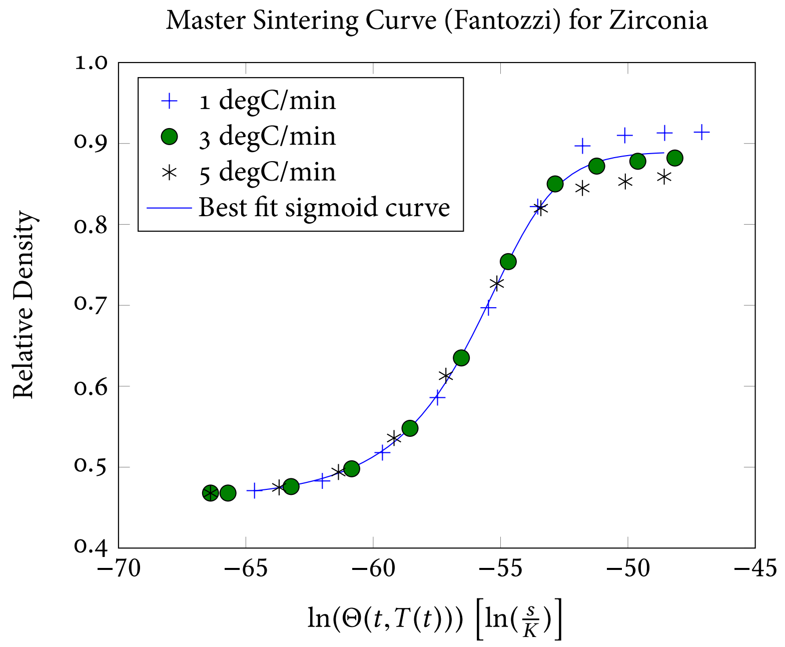


(a) Density with time for three constant-rate sintering trials.

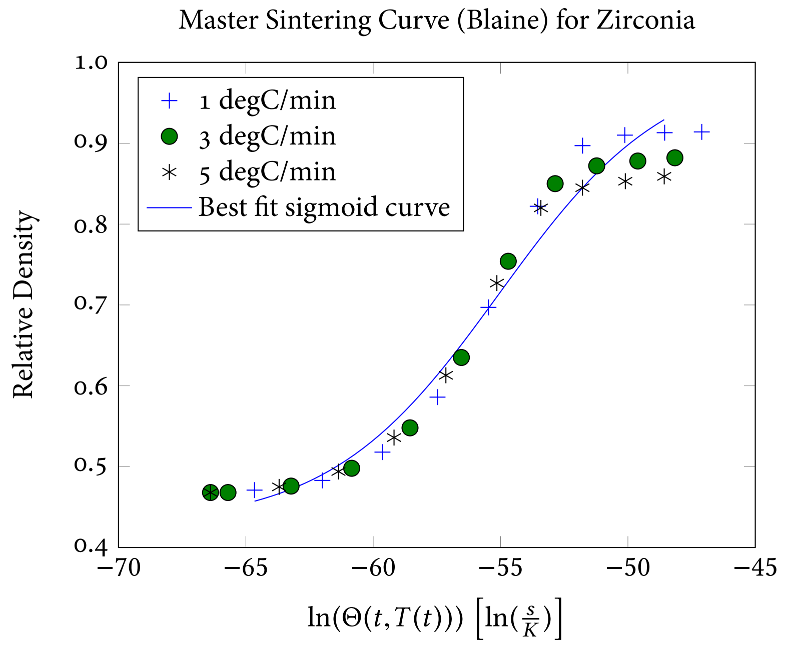


(b) Density with temperature for three constant-rate sintering trials.

Figure 9.2: Plots of the densification data for the three constant-rate sintering trials performed in [3].

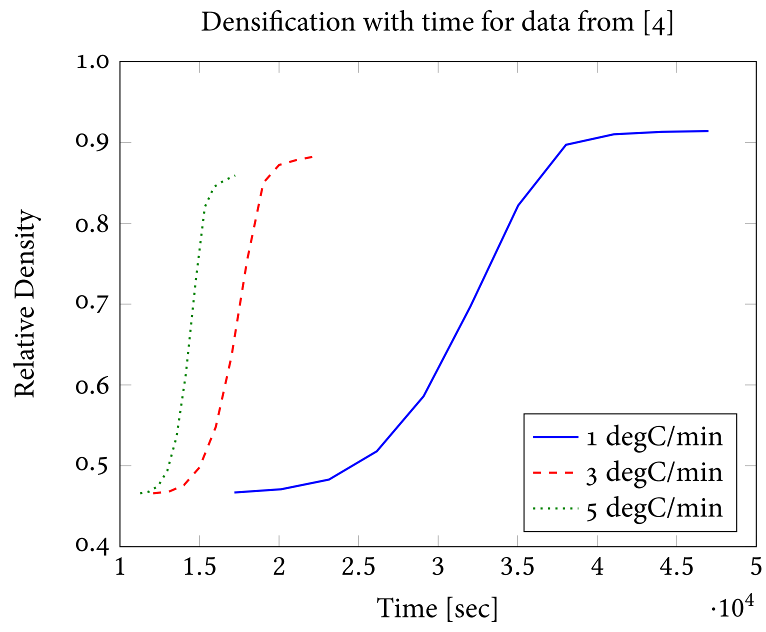


(a) Demonstration of sigmoid curve fitting using the model in Equation 9.3. Best fit was computed using the Levenberg-Marquardt optimization method, and has least-squares error 0.0373143.

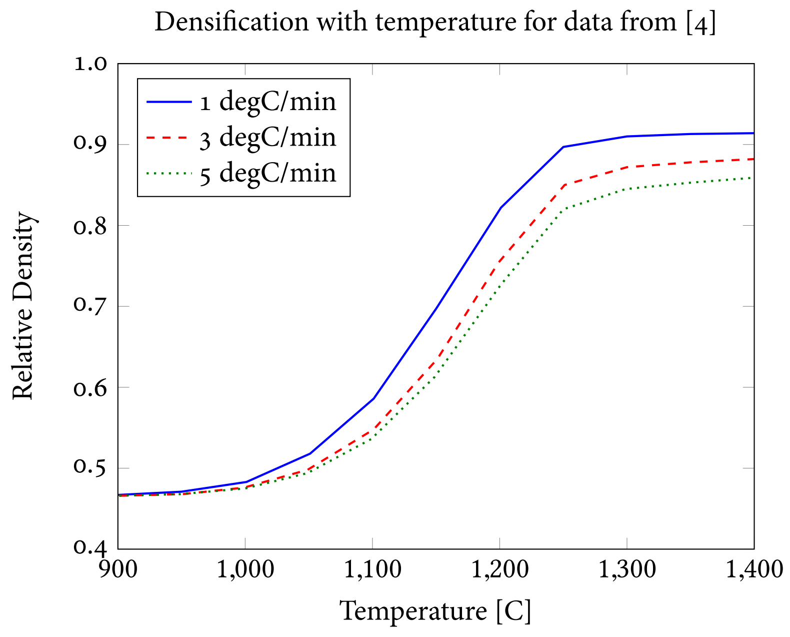


(b) Demonstration of sigmoid curve fitting using the model in Equation 9.2. Best fit was computed using the Levenberg-Marquardt optimization method, and has least-squares error 0.00786899.

Figure 9.3: Demonstration of the sigmoid curve-fitting methods carried out for zirconia data from [4] and an activation energy of $Q = 660.1$ kJ/mol. Optimal sigmoid functions are shown in Equations 9.6 and 9.7.



(a)



(b)

Figure 9.4: Plots of the densification data for the three constant-rate sintering trials performed in [4].

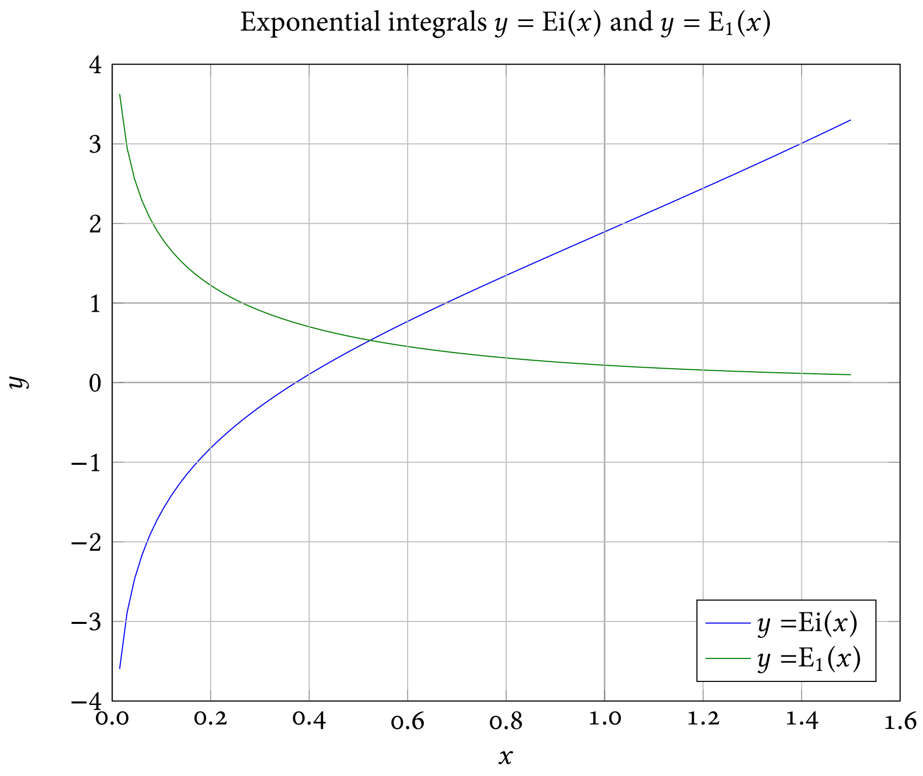


Figure 9.5: Exponential integral function $\mathrm{Ei}(x)$ for $x > 0$, found using `scipy.special` module.

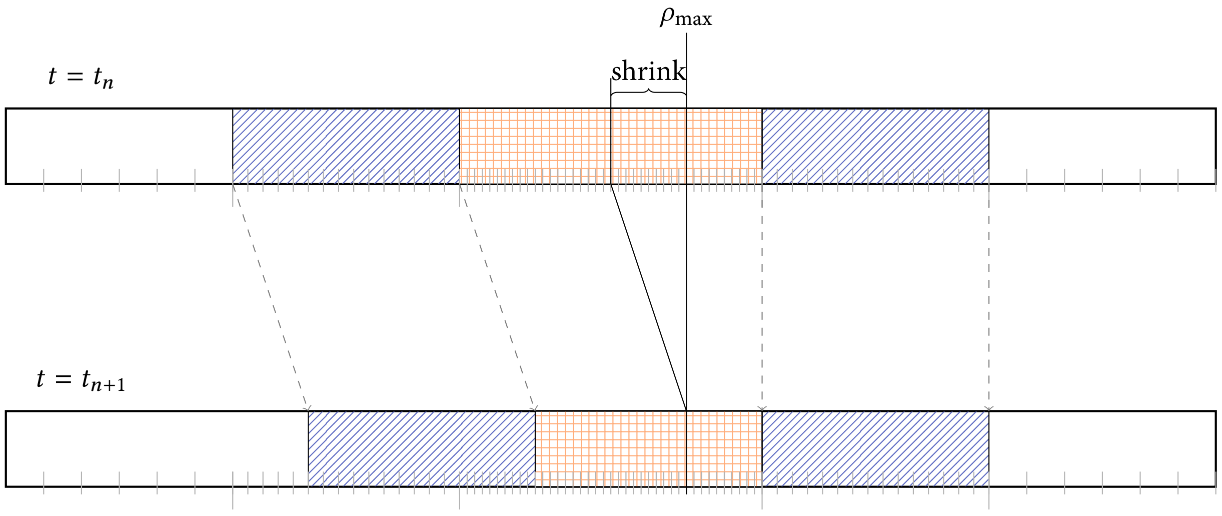


Figure 9.6: Example of execution of shrinkage by the mechanical model for the one-dimensional case. Area of domain occupied by air is shown in white, by insulation in blue (diagonal lines), and by material in apricot (squares). Physical scenario is depicted at time level $t = t_n$ above, and $t = t_{n+1}$ below. Sample numerical grid for the solution of the electromagnetic and thermal equations is fixed, and is shown at both time levels with gray tick marks. $\rho_{\max}$ indicates the location of maximum density within the sample, and the amount of shrinkage, computed using conservation of mass together with the average density change in the sample, is labelled "shrink", and is taken to the left of the maximum density.

Chapter 10

Coupled Multiscale Model of Microwave Sintering

Microwave sintering can be described by a model that couples representations of the electromagnetic, thermal, and mechanical phenomena, while considering the multiscale nature of the problem in both space and time. The operation of such a model is illustrated in Figure 10.1.

This chapter describes the operation of the model, whose corresponding Python implementation can be found in Appendix H.

10.1 Input Data

The user may input two basic kinds of data to the simulator: material data and process data. Material data refers to measurements of dielectric and thermal properties taken during experimental processing of samples of the material and insulation the user wishes to simulate, while process data refers to certain adjustable parameters of the simulated sintering experiment, such as microwave frequency, input power, total sintering time, the temperature or relative density at which the material is considered sintered, or data on the geometrical configuration of the experiment to be simulated. In this section, we discuss these input properties.

Experimentally Obtained Material Data

Before running the model, experiments should be run in order to generate input data for the model. Experiments should be done both for the material to undergo sintering, and for the material that comprises the surrounding insulation, and these experiments should consist of separate trials that involve heating at rates as close to constant as possible, during which the measurements of the necessary dielectric and thermal material properties should be recorded along with the temperature and elapsed processing time.

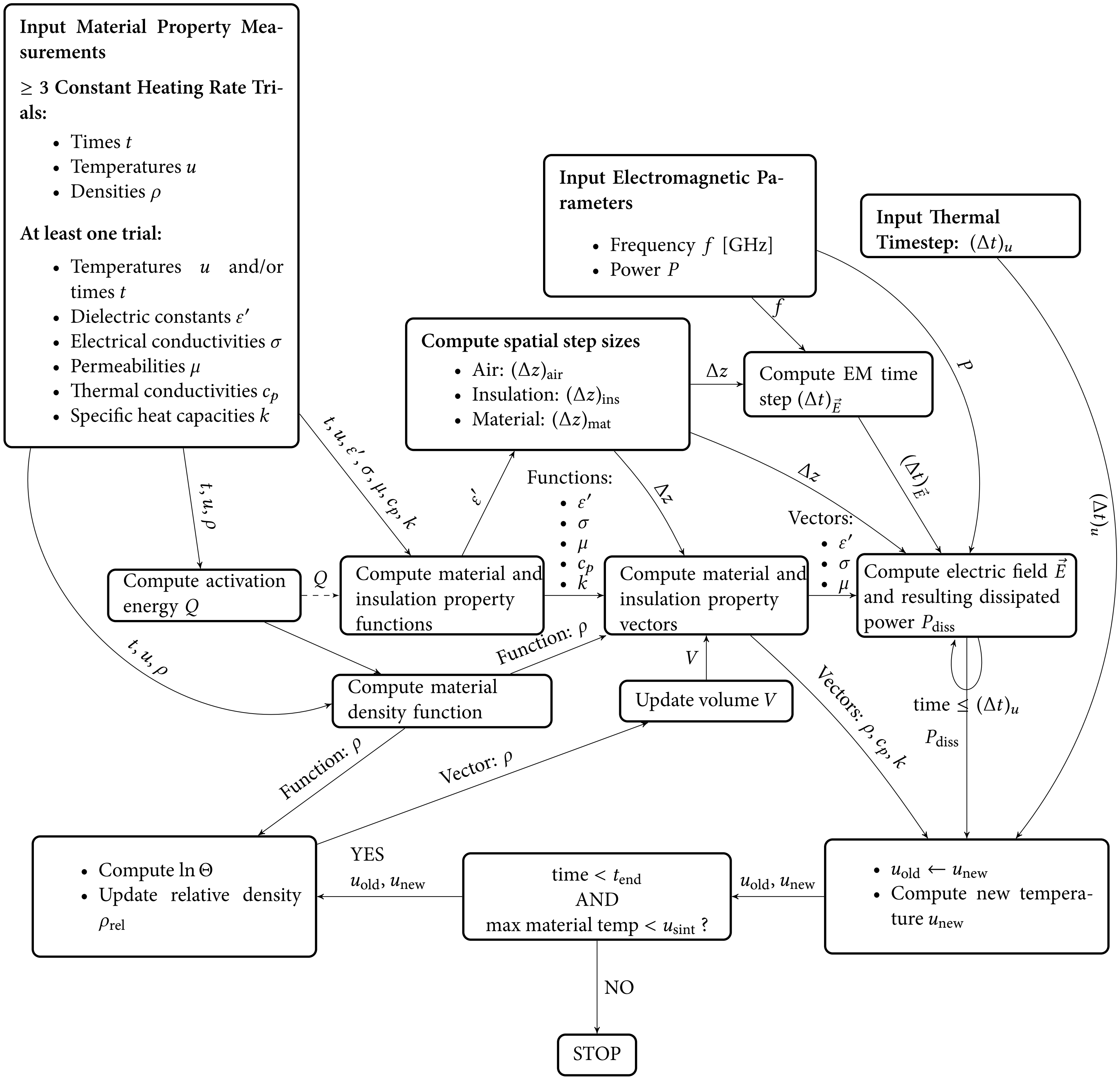

Input Material Property Measurements
≥ 3 Constant Heating Rate Trials:
Times t
Temperatures u
Densities ρ
At least one trial:
Temperatures u and/or times t
Dielectric constants ε'
Electrical conductivities σ
Permeabilities μ
Thermal conductivities c_p
Specific heat capacities k
Input Electromagnetic Parameters
Frequency f [GHz]
Power P
Input Thermal Timestep: $(\Delta t)_u$
f
Compute spatial step sizes
Air: $(\Delta z)_{\text{air}}$
Insulation: $(\Delta z)_{\text{ins}}$
Material: $(\Delta z)_{\text{mat}}$
Δz
Compute EM time step $(\Delta t)_{\vec{E}}$
P
$t, u, \varepsilon', \sigma, \mu, c_p, k$
ε'
Functions: ε' σ μ c_p k
Δz
Δz
$(\Delta t)_{\vec{E}}$
$(\Delta t)_u$
t, u, ρ
Vectors: ε' σ μ
Compute activation energy Q
Q
Compute material and insulation property functions
Compute material and insulation property vectors
Compute electric field $\vec{E}$ and resulting dissipated power P_{diss}
t, u, ρ
Function: ρ
V
time ≤ $(\Delta t)_u$
P_{diss}
Compute material density function
Update volume V
Vectors: ρ, c_p, k
Function: ρ
Vector: ρ
Compute $\ln \Theta$
Update relative density ρ_{rel}
YES
$u_{\text{old}}, u_{\text{new}}$
time < t_{end} AND max material temp < u_{sint} ?
$u_{\text{old}}, u_{\text{new}}$
$u_{\text{old}} \leftarrow u_{\text{new}}$
Compute new temperature u_{new}
NO
STOP

Figure 10.1: Flowchart showing operation of the coupled multiphysics, multiscale model of microwave sintering.

Since the insulation material does not undergo sintering, we do not expect its densification behavior to be predictable using the MSC method discussed in Sections 4.3 and 9.1. Moreover, the dielectric and thermal properties of the insulation material can be adequately characterized by only their temperature dependence, and so only one trial of microwave heating is necessary to produce the data necessary for constructing predictive functions for these properties. For the insulation material, the experimental measurements should consist of a table with six or seven columns: temperature u, dielectric constant ε', electrical conductivity σ, density ρ, thermal conductivity c_p, specific heat capacity k, and permeability μ in the case of magnetic material (for non-magnetic material, $\mu \approx \mu_0$ is a valid assumption). The i^{th} measured value of each of the latter five (or, in the case of magnetic materials, six) quantities should be taken at the i^{th} temperature value, and the temperature values should span the range of those that are expected to be encountered during processing. Such a table is given in [5] for high-temperature processing of alumina and zirconia[1], and is reproduced in Table 10.1.

For the material to undergo sintering, the dielectric and thermal properties are assumed to depend on both temperature and density of the sample during processing. As discussed in Chapter 4, the density of the sample is computed from the temperature evolution function using the Master Sintering Curve method, once the activation energy is known. We therefore need sufficient input data to compute the activation energy; following [74], we require measurements of time, temperature, and sample density from at least three trials of heating—preferably, microwave heating—at constant rates. The requirement that the heating rates be constant for the trials generating the input data is not a mathematical one, but rather, is intended to minimize the possible effects of surface diffusion, which may result in incorrect measurements, during the sintering process [74]. Such data is given in [3] and in [4] for zirconia and various other materials, with that of [4] reproduced in Table 10.2.

For the other dielectric and thermal material properties, which depend on temperature and density, this work incorporates, for the first time, two different methods of characterizing their evolution. The first such method involves the inversion of the mixture formulas, and is discussed in Chapter 5. This method necessitates measurements of dielectric and thermal properties throughout the full temperature range, with reference densities and temperatures also recorded, as shown in Table 10.1.

The second method of characterizing the evolution of dielectric and thermal properties has only a phenomenological basis, and assumes that if the density may be characterized as a function of Θ, the work of sintering defined in Section 4.3, and if Θ is, itself, a function of the temperature and its evolution, then the dielectric and thermal material properties may also be characterized as functions of Θ. This process of determining these functions is discussed in Section 10.2, but here, we simply state that the input data needed in this case are measurements of the dielectric and thermal properties throughout the full temperature range of a processing experiment, with reference times and temperatures also recorded.

[1]Measurements taken by Ron Hutcheon of Microwave Properties North, Inc.

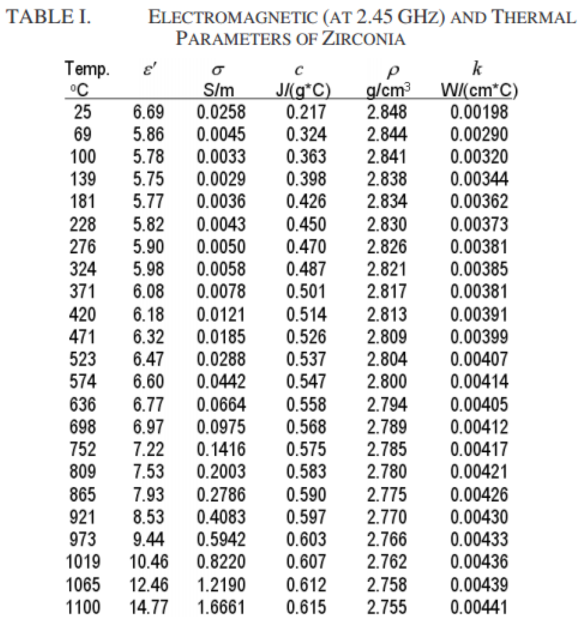


TABLE I. ELECTROMAGNETIC (AT 2.45 GHZ) AND THERMAL PARAMETERS OF ZIRCONIA

Temp.	ε'	σ	c	ρ	k
°C		S/m	J/(g*C)	g/cm^3	W/(cm*C)
25	6.69	0.0258	0.217	2.848	0.00198
69	5.86	0.0045	0.324	2.844	0.00290
100	5.78	0.0033	0.363	2.841	0.00320
139	5.75	0.0029	0.398	2.838	0.00344
181	5.77	0.0036	0.426	2.834	0.00362
228	5.82	0.0043	0.450	2.830	0.00373
276	5.90	0.0050	0.470	2.826	0.00381
324	5.98	0.0058	0.487	2.821	0.00385
371	6.08	0.0078	0.501	2.817	0.00381
420	6.18	0.0121	0.514	2.813	0.00391
471	6.32	0.0185	0.526	2.809	0.00399
523	6.47	0.0288	0.537	2.804	0.00407
574	6.60	0.0442	0.547	2.800	0.00414
636	6.77	0.0664	0.558	2.794	0.00405
698	6.97	0.0975	0.568	2.789	0.00412
752	7.22	0.1416	0.575	2.785	0.00417
809	7.53	0.2003	0.583	2.780	0.00421
865	7.93	0.2786	0.590	2.775	0.00426
921	8.53	0.4083	0.597	2.770	0.00430
973	9.44	0.5942	0.603	2.766	0.00433
1019	10.46	0.8220	0.607	2.762	0.00436
1065	12.46	1.2190	0.612	2.758	0.00439
1100	14.77	1.6661	0.615	2.755	0.00441

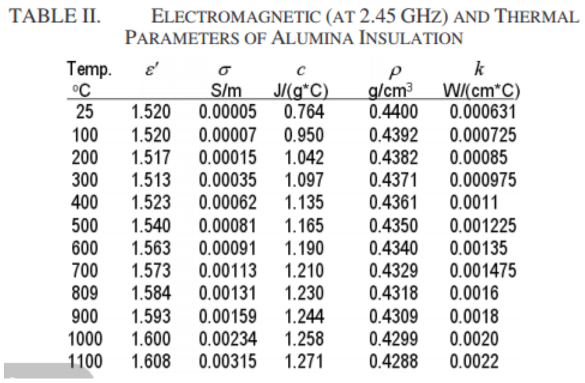
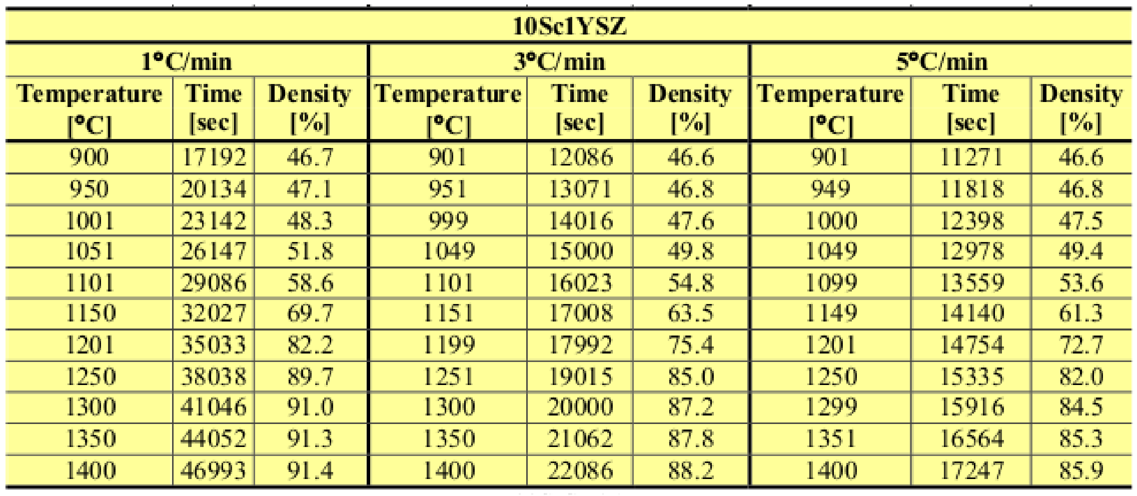


TABLE II. ELECTROMAGNETIC (AT 2.45 GHZ) AND THERMAL PARAMETERS OF ALUMINA INSULATION

Temp.	ε'	σ	c	ρ	k
°C		S/m	J/(g*C)	g/cm^3	W/(cm*C)
25	1.520	0.00005	0.764	0.4400	0.000631
100	1.520	0.00007	0.950	0.4392	0.000725
200	1.517	0.00015	1.042	0.4382	0.00085
300	1.513	0.00035	1.097	0.4371	0.000975
400	1.523	0.00062	1.135	0.4361	0.0011
500	1.540	0.00081	1.165	0.4350	0.001225
600	1.563	0.00091	1.190	0.4340	0.00135
700	1.573	0.00113	1.210	0.4329	0.001475
809	1.584	0.00131	1.230	0.4318	0.0016
900	1.593	0.00159	1.244	0.4309	0.0018
1000	1.600	0.00234	1.258	0.4299	0.0020
1100	1.608	0.00315	1.271	0.4288	0.0022

Table 10.1: Experimental measurements of temperature, dielectric constant, electrical conductivity, specific heat capacity, absolute density, and thermal conductivity for zirconia and alumina. Reproduced from [5].



10Sc1YSZ								
1°C/min			3°C/min			5°C/min		
Temperature [°C]	Time [sec]	Density [%]	Temperature [°C]	Time [sec]	Density [%]	Temperature [°C]	Time [sec]	Density [%]
900	17192	46.7	901	12086	46.6	901	11271	46.6
950	20134	47.1	951	13071	46.8	949	11818	46.8
1001	23142	48.3	999	14016	47.6	1000	12398	47.5
1051	26147	51.8	1049	15000	49.8	1049	12978	49.4
1101	29086	58.6	1101	16023	54.8	1099	13559	53.6
1150	32027	69.7	1151	17008	63.5	1149	14140	61.3
1201	35033	82.2	1199	17992	75.4	1201	14754	72.7
1250	38038	89.7	1251	19015	85.0	1250	15335	82.0
1300	41046	91.0	1300	20000	87.2	1299	15916	84.5
1350	44052	91.3	1350	21062	87.8	1351	16564	85.3
1400	46993	91.4	1400	22086	88.2	1400	17247	85.9

Table 10.2: Measurements of time, temperature, and relative density of Yttria-stabilized zirconia during sintering trials at three constant heating rates [4].

Simulated Process Data

As discussed, certain information about the process data is necessary to replicate the experiment the user wishes to simulate, beginning with geometrical properties. The user may, in addition to the size of the domain, also specify the placement of the insulation and material, in both the one- and two-dimensional scenarios. In the one-dimensional case, the user may specify the lengths and the right-hand endpoints of both the material and the insulation, while in the two-dimensional case, the user may, in addition to the lengths and right-hand boundaries, also specify the heights and lower boundaries of both the material and insulation.

The user should also specify the desired input power and frequency of microwave radiation, along with the ambient temperature of the room in which the simulated experiment is carried out.

As stopping conditions, our algorithm checks the elapsed time against the total time the user desires for the simulated experiment, and also checks the maximum temperature in the material against a prescribed maximum temperature that the user defines—and, so, the total time and maximum temperature are also necessary input parameters to the simulation.

10.2 Tasks Completed Before Iterative Loop

The tasks described in this section are completed once before the solver enters the iterative loop.

Computing the Activation Energy

After the user inputs the experimental data, the solver computes the activation energy Q of the material to be sintered using the method discussed in Section 4.3. The user has the option of representing the sigmoid curve by the model described in Equation 9.2, or by the model in Equation 9.3; once the sigmoid curve model is chosen, the activation energy is found using the Levenberg-Marquardt method to minimize the error in the sigmoid curve that best fits the data found experimentally; the best fit curve is fund using the Nelder-Mead method.

Finding the Material Density Function

The activation energy is found concurrently with the best-fit sigmoid curve, which we refer to as the function for computing material density. This function takes the parameter Θ as input, and yields the corresponding density of the sample, relative to the bulk density (the relative density ρ_{rel} is in the range $(0, 1)$).

Dielectric and Thermal Properties of Insulation

After the material density function is computed, the functions determining the dielectric and thermal properties of both the insulation and the sintering material are found. The functions for the insulation properties, including the density, are assumed to be dependent only on temperature, and so these are computed from the single set of experimental measurements using third-degree b-spline interpolation.

Dielectric and Thermal Properties of Sintered Material

For the sintered material, the properties (other than density) are assumed to be dependent on both density and temperature, and are found using either the method discussed in Chapter 5, which involves either the first method of inverting various mixture formulas, or the second method mentioned in Section 10.1 of this chapter, where properties are all assumed to depend on Θ. If the inversion of the mixture formulas is used, then the property functions take as input both the temperature and the relative density.

If the user indicates the first method, then the thermal properties c_p and k are computed according to Equations 5.13 and 5.14. The user then has the choice of whether to use the Lichtenecker, Rayleigh, Maxwell Garnett, or Bruggeman formulas for computing the dielectric properties; these formulas are found in Equations 5.6, 5.8, 5.12, and 5.10, respectively.

If the user indicates the second, Θ-based method, then the activation energy Q is used, along with the time and temperature measurements taken during the physical sintering experiment, to compute values of Θ for the experiment. These Θ values are then used, along with the measured property values, to construct interpolating functions using the method of third-order b-splines. The functions created using this method take Θ values as inputs.

Determining Spatial and Time Steps

Both the implicit and explicit finite difference methods for the electromagnetic wave equation, discussed in Chapter 7, require that the spatial step be proportional to the wavelength in media, with a typical simulation requiring approximately 10 to 20 spatial cells per wavelength [2, 183, 184]. However, the wavelength λ in media depends on the relative values of the dielectric constant ε' and the permittivity μ to those in free space, according to

$$\lambda = \frac{c}{f\sqrt{\varepsilon'_{\mathrm{rel}}\mu_{\mathrm{rel}}}}, \tag{10.1}$$

where c is the speed of light, and f is the frequency of radiation (in Hz). Because $\varepsilon'_{\mathrm{rel}}$ and μ_{rel} in the insulation depend on temperature, and in the material to be sintered depend on both temperature and relative density, the code first estimates the largest value of $\varepsilon'_{\mathrm{rel}}$ and μ_{rel} that are expected to be encountered during processing (in practice, it chooses the largest values from the input experimental data, which is assumed to cover the entire sintering range). Once the largest expected values of $\varepsilon'_{\mathrm{rel}}$ and μ_{rel} are determined for the insulation and for the material, each of these values is used in Equation 10.1 to find the smallest wavelength λ_{ins} and λ_{mat} expected in the insulation and in the material, respectively. Using these, together with λ_{air}, which is directly computed using the value $\varepsilon'_{\mathrm{rel,air}} \approx 1$, the required sizes of the spatial steps in air $(\Delta z)_{\mathrm{air}}$, insulation $(\Delta z)_{\mathrm{ins}}$ and the material to be sintered $(\Delta z)_{\mathrm{mat}}$ are determined. In two dimensions, we assume that $\Delta x = \Delta z$ in air, insulation, and material.

For an explicit difference method approximating the solution of the electromagnetic wave equation to accurately simulate the wave velocity, the time step should also be restricted, with dispersion analysis resulting in Equation 7.5. The physical interpretation of this condition is that over the course of one time step, the wave should not propagate more than the length of one spatial cell [2]. We therefore compute $(\Delta t)_{\vec{E}}$ according to

$$(\Delta t)_{\vec{E}} < \min\left\{\frac{(\Delta z)_{\mathrm{air}}}{c}, \frac{(\Delta z)_{\mathrm{ins}}}{v_{p,\mathrm{ins}}}, \frac{(\Delta z)_{\mathrm{mat}}}{v_{p,\mathrm{mat}}}\right\}, \tag{10.2}$$

where v_p represents the velocity of the wave, which, as noted in Section 2.4, is given by $v_p := \frac{1}{\sqrt{\varepsilon\mu}} = \frac{c}{\sqrt{\varepsilon_{\mathrm{rel}}\mu_{\mathrm{rel}}}}$. In practice, this condition is satisfied by setting $(\Delta t)_{\vec{E}} = \frac{1}{2f}$.

The time step for the approximate solution to the heat equation need not be as small as that required for the electromagnetic solver. Because code, as discussed in Chapter 8, implements the finite difference method as a θ-scheme, the user has the choice of what becomes, in practice, a fully implicit scheme ($\theta = 1$), a fully explicit scheme ($\theta = 0$), a Crank-Nicolson scheme ($\theta = \frac{1}{2}$), or something in between. For $\theta < \frac{1}{2}$, the scheme is conditionally stable, as discussed in Section 8.1. For $\theta \geq \frac{1}{2}$, the numerical scheme is unconditionally stable, but still may suffer from spurious oscillations if the time step is too large compared to the spatial step. We therefore determine the size of the time step by setting

$$(\Delta t)_u = \min\left\{\frac{(\Delta z)^2_{\mathrm{air}}}{2}, \frac{(\Delta z)^2_{\mathrm{ins}}}{2}, \frac{(\Delta z)^2_{\mathrm{mat}}}{2}\right\}.$$

10.3 Tasks Completed Within the Iterative Loop

With the pre-processing tasks completed, the iterative loop begins. We describe here one iteration of the process. To initialize the loop, the elapsed time is set to zero, the current temperature is assumed to be room temperature everywhere, the parameter value Θ is assumed to be one, and the electric field is assumed zero except on the left-hand boundary, where it takes the value of the incident field, as discussed in Section 2.4.

Setting the Material Property Vectors

First, the vector of density values is populated according to the current values of Θ in the material, the current values of temperature in the insulation, and according to the current location of the material, insulation, and air; that is, according to which nodes are currently occupied by which kind of matter. If a node—call it node number $[i,j]$, with coordinate value (z_i, y_j)—is occupied by air, then its density is assumed to be the constant value of the density of air:

$$\rho[i,j] = \rho_{\text{air}}.$$

If the node is occupied by insulation, then its density is computed using the temperature-dependent function found by the procedure described in Section 10.2 of this chapter:

$$\rho[i,j] = \rho_{\text{ins}}(u(z_i, y_j)).$$

If the node is occupied by material, then its density is computed using the Θ-dependent function found by the procedure described in Section 10.2 of this chapter, which will yield the value of relative density $\rho_{\text{rel,mat}}$; this value is then multiplied by the constant bulk density of the material to obtain the correct absolute density value:

$$\rho[i,j] = \rho_{\text{bulk,mat}} \cdot \rho_{\text{mat}}(\Theta(z_i, y_j)).$$

In summary, the density vector is populated according to:

$$\rho[i,j] = \begin{cases} \rho_{\text{air}}, & \text{if node } [i,j] \text{ is in air} \\ \rho_{\text{ins}}(u(z_i, y_j)), & \text{if node } [i,j] \text{ is in insulation} \\ \rho_{\text{bulk,mat}} \cdot \rho_{\text{mat}}(\Theta(z_i, y_j)), & \text{if node } [i,j] \text{ is in material.} \end{cases}$$

Once the density values are known, the other material property vectors may be populated; for example, the value of the dielectric constant vector at position $[i,j]$ is as follows, if the density-dependent properties of the material are accounted for by inverting mixture formulas:

$$\varepsilon'[i,j] = \begin{cases} \varepsilon'_{\text{air}}, & \text{if node } [i,j] \text{ is in air} \\ \varepsilon'_{\text{ins}}(u(z_i, y_j)), & \text{if node } [i,j] \text{ is in insulation} \\ \varepsilon'_{\text{mat}}(u(z_i, y_j), \rho[i,j]), & \text{if node } [i,j] \text{ is in material,} \end{cases}$$

and as follows, if the density-dependent properties of the material are accounted for under the assumption that they are Θ-dependent:

$$\varepsilon'[i,j] = \begin{cases} \varepsilon'_{\text{air}}, & \text{if node } [i,j] \text{ is in air} \\ \varepsilon'_{\text{ins}}(u(z_i,y_j)), & \text{if node } [i,j] \text{ is in insulation} \\ \varepsilon'_{\text{mat}}(\Theta(z_i,y_j)), & \text{if node } [i,j] \text{ is in material.} \end{cases}$$

Here, the function $\varepsilon'_{\text{ins}}(u)$ is the one discussed in Section 10.2 of this chapter, and $\varepsilon'_{\text{mat}}(u,\rho)$ is, in the first case, the function defined in Chapter 5, and in the second case, the function defined in Section 10.2 of this chapter.

The vectors σ, c_p, k, and, in the case of magnetic material or insulation, μ are also populated in a similar fashion.

Computing the Electric Field and Dissipated Power

Once the material property vectors are populated, the dissipated power is set to zero, and the electric field $\vec{E}$ is solved using the methods described in Chapter 7. The user may choose the finite difference or finite element method, though the examples in Chapter 11 were generated using the finite difference code.

The electric field solver takes as input the vectors μ, σ, and ε', along with the vectors h_z and h_y of spatial differences, and the time steps $(\Delta t)_{\vec{E}}$ and $(\Delta t)_u$. At each electromagnetic time step $(\Delta t)_{\vec{E}}$, the solver computes the electric field once, using the method discussed in Chapter 7, then uses the trapezoid rule to approximate the addition to the cumulative integral in Equation 2.44. This process continues until the elapsed time passes $(\Delta t)_u$, and the value of P_{diss} is passed to the thermal solver.

Computing the Temperature Field

The thermal solver, described in Chapter 8, takes as input the vectors c_p, ρ, and k, and a source term vector, which is computed according to Equation 3.9 and depends on P_{diss} and σ. The spatial difference vectors h_z and h_y are also input to the thermal solver, along with the size of the thermal time step $(\Delta t)_u$. The heat equation is solved only within the nodes that are occupied by either insulation or material, and the air in the cavity is assumed to remain at room temperature; therefore, the input vectors to the thermal solver are restricted to only those portions corresponding to nodes in insulation or air. Depending on the user's choice of boundary condition, the ambient temperature should also be input to the thermal solver.

The user has the choice of either the finite difference or the finite element method, though the examples in Chapter 11 were generated using the finite difference code. If the user chooses to use the finite difference method, then the θ parameter may also be chosen, which determines whether the method is implicit ($\theta = 1$), explicit ($\theta = 0$), Crank-Nicolson ($\theta = \frac{1}{2}$), or something in between. The user may also choose whether to use the Dirichlet boundary condition, in which the edges of the insulation are fixed at the ambient temperature, the Neumann boundary condition, in which the heat flux out of the insulation's boundary is fixed at zero, or the radiative condition, in which the heat flux through the insulation's boundary is proportional to the difference between ambient temperature and the temperature at the boundary; these boundary conditions are described in Section 3.3. Once the user specifies these settings, the solver is run, and the temperature distribution within insulation and material is output.

Computing Mechanical Deformation

Using the new temperature distribution, together with the temperature distribution from the previous thermal timestep, the cumulative integral in Equation 4.9 is approximated within the material using the trapezoidal rule, and is used in computing the value of the parameter Θ. Once Θ is known at each node within material, the average value is taken, and is input to the function $\rho_{\text{mat}}(\Theta)$ to yield the average relative density within the material.

The amount of shrinkage (or thermal expansion) of the material is computed using the conservation of mass law described in Section 9.2, and the labels on the nodes occupied by material, insulation, and air change commensurately with this deformation, before the iterative loop begins once more.

10.4 Tasks Completed after Iterative Loop Ends

The iterative loop ends once the elapsed time has exceeded the simulated processing time requested by the user, or once the maximum temperature or relative density with the material has exceeded thresholds requested by the user at the beginning of the simulation. When at least one of these conditions is met, the simulation halts, printing its final results to a log file and saving relevant plots and videos in the simulation directory.

Chapter 11

Computational Example: Sintering of Zirconia

In order to demonstrate the functionality of the model described in Chapter 10, we used it for simulating the microwave thermal processing of material that is dealt with in practical laboratory settings. In this chapter, we describe simulations corresponding to different types of boundary conditions, and different ways of handling density dependence of material properties, in both the one- and two-dimensional scenarios.

11.1 One-Dimensional Simulation with Θ-Dependent Dielectric Properties

Input Measurements and Parameters

Our trial simulated the microwave thermal processing of zirconia (ZrO_2) surrounded by alumina (Al_2O_3) insulation. This choice is motivated by the availability of experimentally measured input data to the model in the literature; this data includes measurements of ε, μ, σ, c_p, ρ, and k throughout the temperature range of sintering experiments described in [5] and performed by Ron Hutcheon at Microwave Properties North, which is reproduced in Table 10.1. The data required for the Master Sintering Curve method was obtained from [3] and [4], and is reproduced in Table 10.2.

The frequency of radiation was assumed to be 2.45 GHz, and the input power level was set to 1 kW. The domain length was set to be 43.35 cm, which was 2.5 times one wavelength in the waveguide, where length of the guided wave was computed according to Equations 2.46, 2.47, and 2.48, assuming that the cross-section of the waveguide was an 86.36×43.18 mm rectangle (this corresponds to the typical measurements of a D-band, WR-340 waveguide [62, 185]).

A sample of zirconia 4.82 cm long was centered in the waveguide's length, and the zirconia was assumed to be surrounded on either side by 4.82 cm of alumina insulation. The initial density of the zirconia was assumed to be 52.38 % of bulk density, in accordance with the value of bulk density of solid zirconia taken from [186]. Ambient temperature, which was assumed to be the same as the initial temperature of zirconia and alumina, was set to 20°C. The simulation was set to stop when the processing time reached 3600 seconds.

Insulation and Material Property Functions

Activation energy Q for zirconia was approximated using the Nelder-Mead algorithm, where the objective function was the error in the optimal sigmoid curve describing the relationship between density and $\Theta(t, T(t))$, where Θ was computed as a function of Q using the experimental data on time and temperature taken from [3]. At each evaluation of the objective function, the optimal-fit sigmoid curve was found using Levenberg-Marquardt optimization to minimize the error between the function [166]

$$\rho_{\text{rel}}(\Theta) = \rho_0 + \frac{A}{\left[1 + \exp\left(-\frac{\ln(\Theta)-\ln(\Theta_0)}{B}\right)\right]^C}$$

and the measured density values through the course of sintering, where ρ_0 and Θ_0 are the initial values of relative density and Θ at the start of the sintering experiments, and where A, B, and C are the parameters adjusted in the course of Nelder-Mead optimization. The entire routine for finding the optimal activation energy took a total of 65 function evaluations and required approximately 80.1 CPU-seconds to perform, including overhead for plotting and saving results. The optimal Q value was found to be approximately 674214 J/mol, which is within the range of values (615 ± 80 kJ/m) found in [187], and is also within the range of values found in [186, 188–192]. Using this value of Q, corresponding optimal sigmoid curve hit the data points with a mean relative residual of 0.001157. This curve, which was found using the data in [3], is shown in Figure 11.3a, and is given by

$$\rho_{\text{rel}} = 0.52536 + \frac{0.464799}{\left[1 + \exp\left(-\frac{\ln\Theta+52.5762}{1.71725}\right)\right]^{0.627238}}. \tag{11.1}$$

The dielectric and thermal properties of zirconia were determined as functions of temperature and density by assuming a dependence on the work of sintering, Θ. These functions of Θ were determined using third-degree b-splines, and the functions, along with the measured data points they interpolate, are shown in Figures 11.1a, 11.2a, 11.4a, and 11.5a. Because zirconia is not a magnetic material, the magnetic permeability μ was assumed to be the same as free space; that is, $\mu = \mu_0$, or $\mu_{\text{rel}} = 1$.

The dielectric and thermal properties of alumina insulation were determined as functions of temperature using interpolation by third-degree b-splines, and the functions, along with the measured data points they interpolate, are shown in Figures 11.1b, 11.2b, 11.3b, 11.4b, and 11.5b. Because alumina is not a magnetic material, the magnetic permeability μ was assumed to be the same as free space; that is, $\mu = \mu_0$, or $\mu_{\text{rel}} = 1$.

The evolution of the material properties in the entire domain through time can be seen in Figures 11.6, 11.7, 11.8, 11.9, and 11.10.

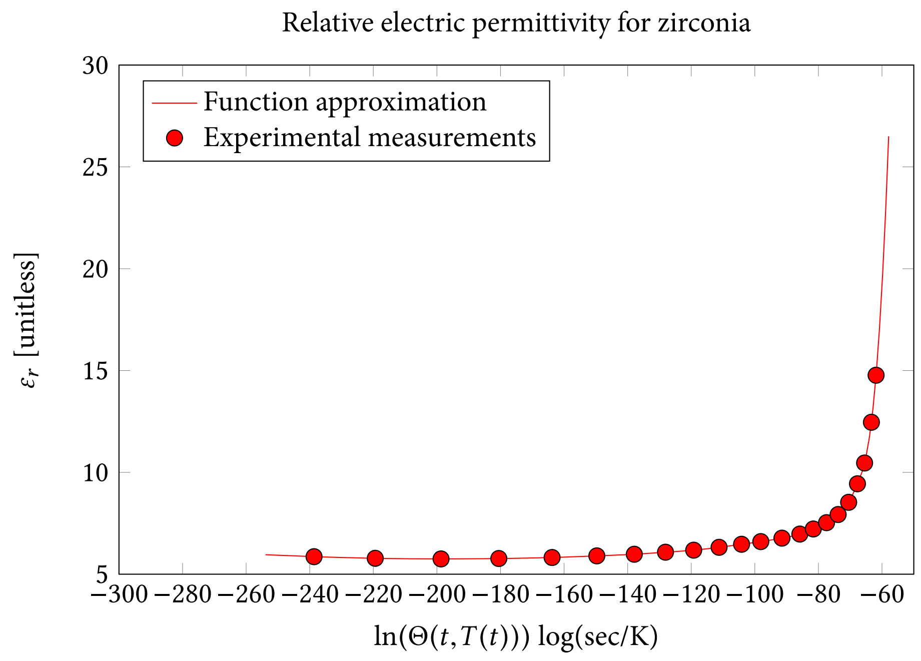
Electric and Temperature Fields

Inside the simulation loop, the electric field was solved using the θ-finite difference method with $\theta = 0.5$, and with a time step of 0.001 seconds, computed according to Equation 10.2. The absorbing boundary condition was used at the right-hand side. The envelope of the electric field at the end of processing is shown in Figure 11.11, and this did not change significantly during processing, despite the dielectric properties changing. The maximum, minimum, and mean values of the electric field at regular time intervals throughout the simulation can be found in the simulation output log file, partially reproduced in Appendix I.1.

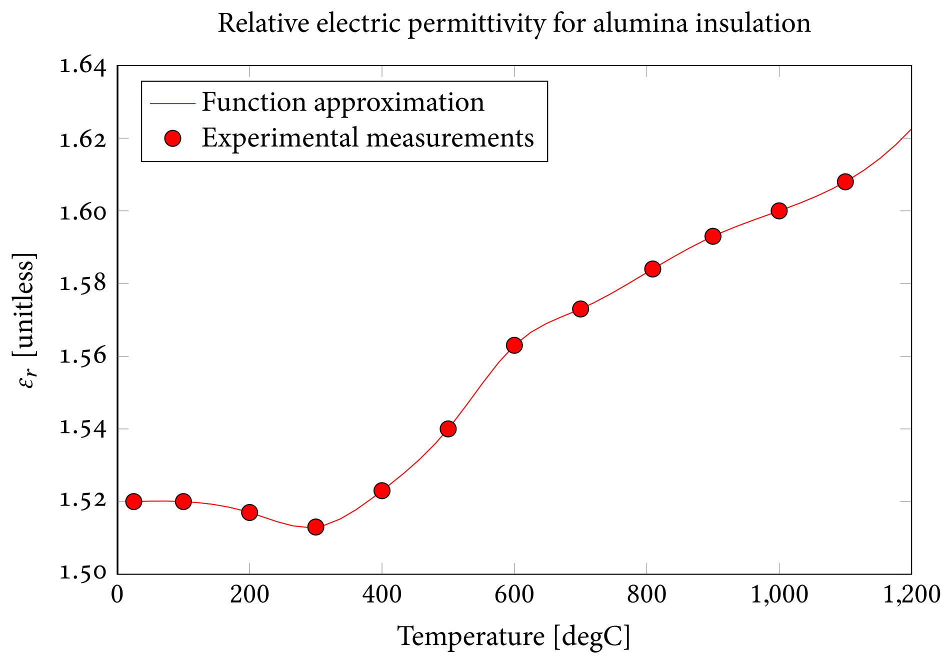
There were clear peaks in the temperature field during processing, which, by the end of processing, smoothed to the distribution seen in Figure 11.12. The evolution of the temperature distribution through time may be seen in Figure 11.13, and the evolution of the maximum and mean temperatures within the load may be seen in Figure 11.14.

After 650 seconds of simulated processing, the zirconia showed slight thermal expansion, and after 3600 seconds, showed shrinkage to 98% of its green length, with the relative density increasing to 52.54% of bulk density with a maximum zirconia temperature of 920.8° C. This appears to be consistent with the shrinkage results for three-dimensional samples of zirconia sintered by microwaves [4], which report densification to 53.6% of bulk density at 1099°C. The evolution of density with the work of sintering Θ can be seen in Figure 11.14.

The entire simulation took 25399.9 CPU seconds to complete, including overhead for logging and plotting results.

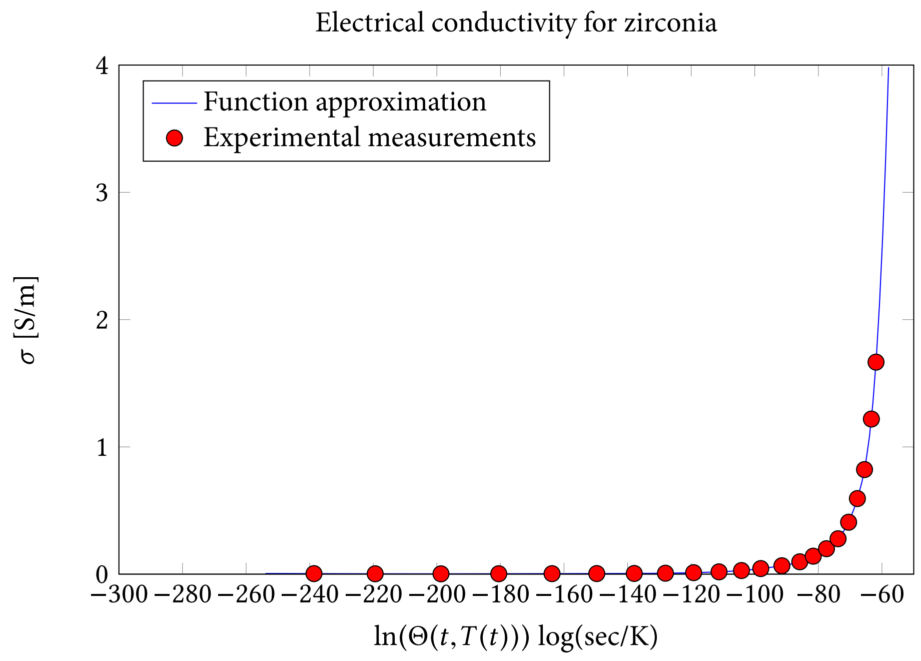


(a) Evolution of $\varepsilon'_{\mathrm{rel}}$ with the work of sintering Θ for zirconia.

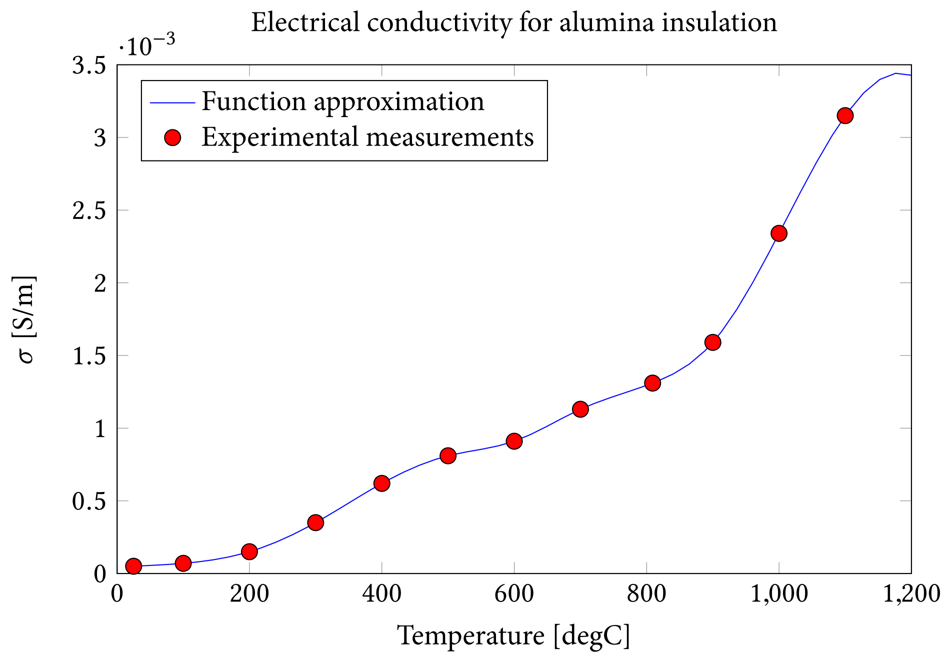


(b) Evolution of $\varepsilon'_{\mathrm{rel}}$ with temperature for alumina.

Figure 11.1: The curves, found using third-degree b-splines, describing the evolution of the dielectric constant $\varepsilon'_{\mathrm{rel}}$, relative to ε_0, of zirconia material and alumina insulation. Points represent measured input data from [5].

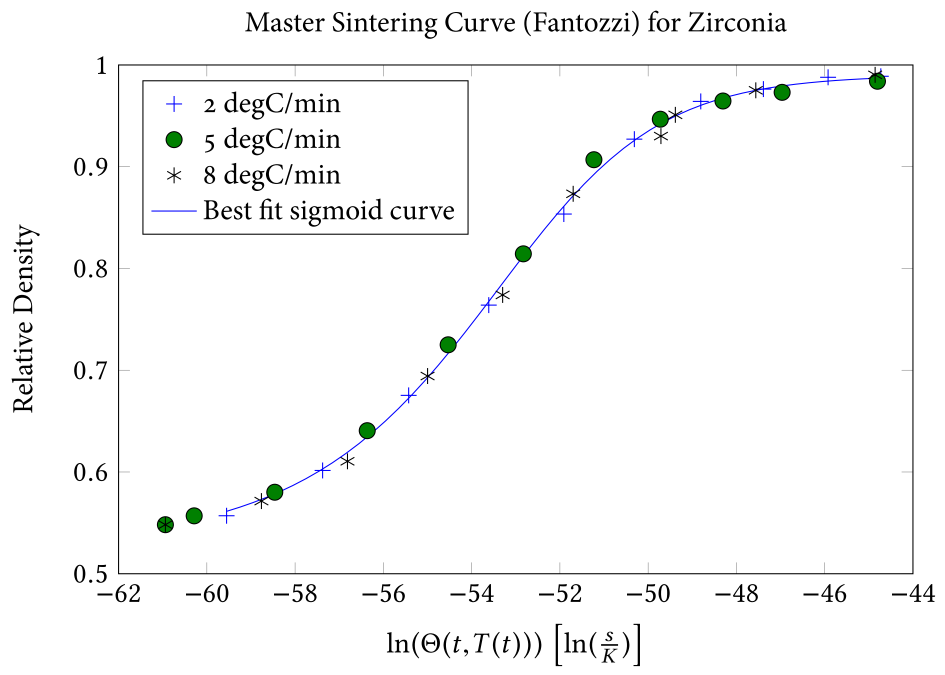


(a) Evolution of σ with the work of sintering Θ for zirconia.

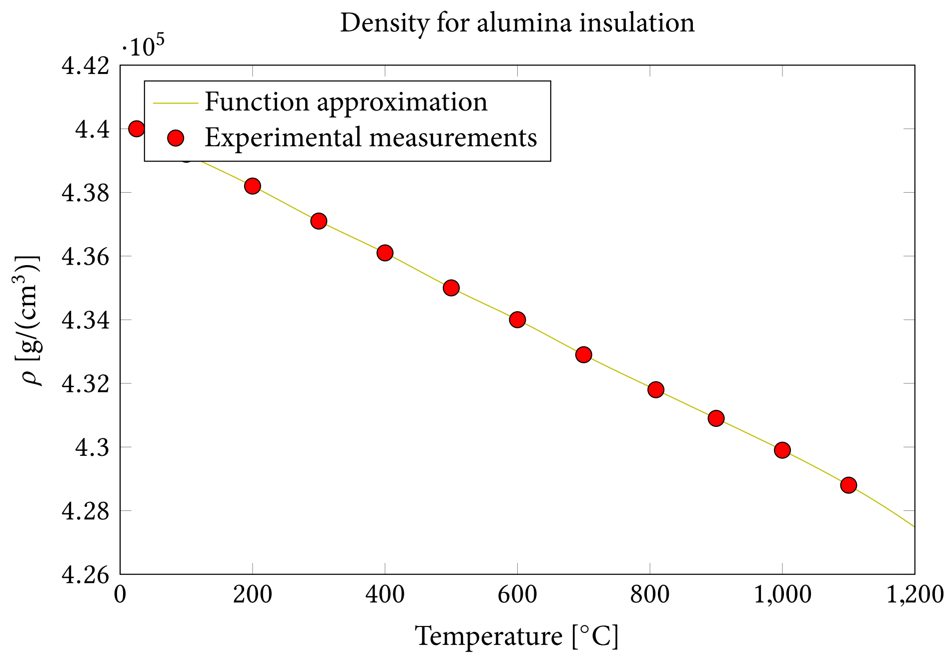


(b) Evolution of σ with temperature for alumina.

Figure 11.2: The curves, found using third-degree b-splines, describing the evolution of the electrical conductivity σ [S/m] of zirconia material and alumina insulation. Points represent measured input data from [5].

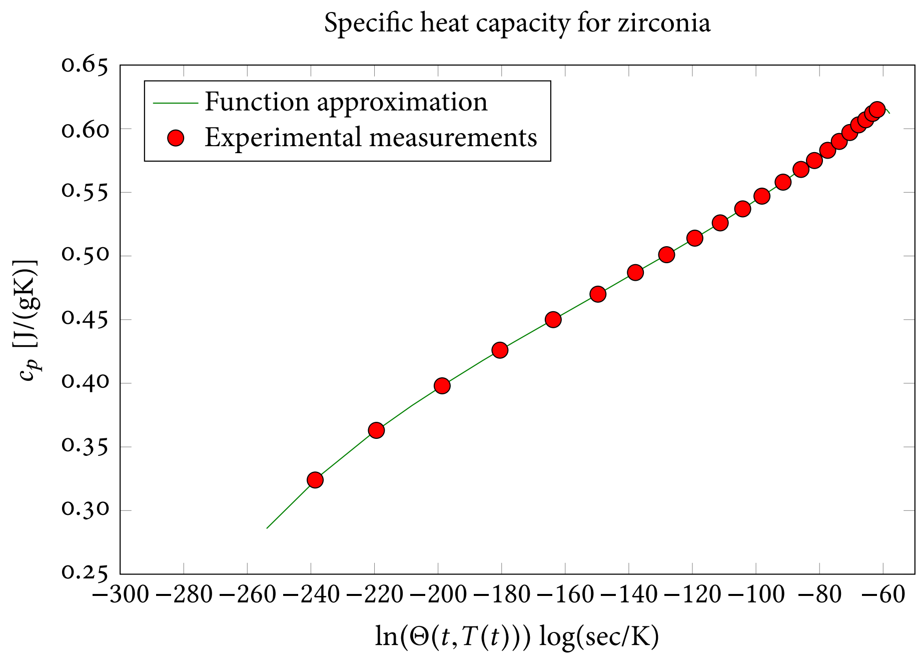


(a) The optimal-fit sigmoid curve describing the relationship between the work of sintering Θ and the [unitless] density ρ_{rel} of zirconia, relative to its bulk density, using data from [3] and the sigmoid curve described in Equation 11.1.

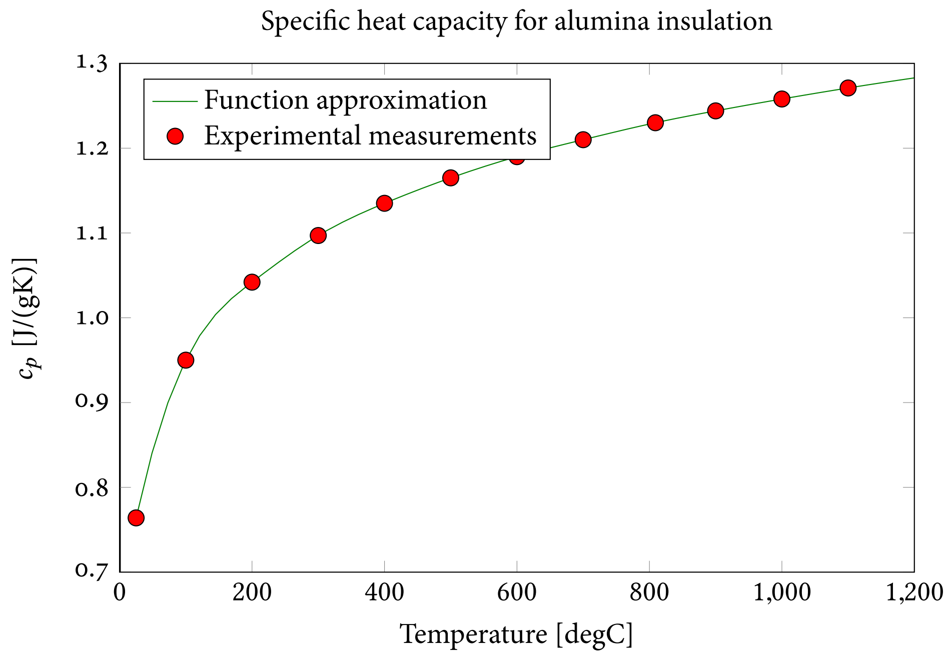


(b) The curves, found using third-degree b-splines, describing the evolution of absolute density ρ [g/m^3] of alumina insulation with temperature. Points represent measured input data from [5].

Figure 11.3: The functions describing the evolution of the density of zirconia material and alumina insulation.

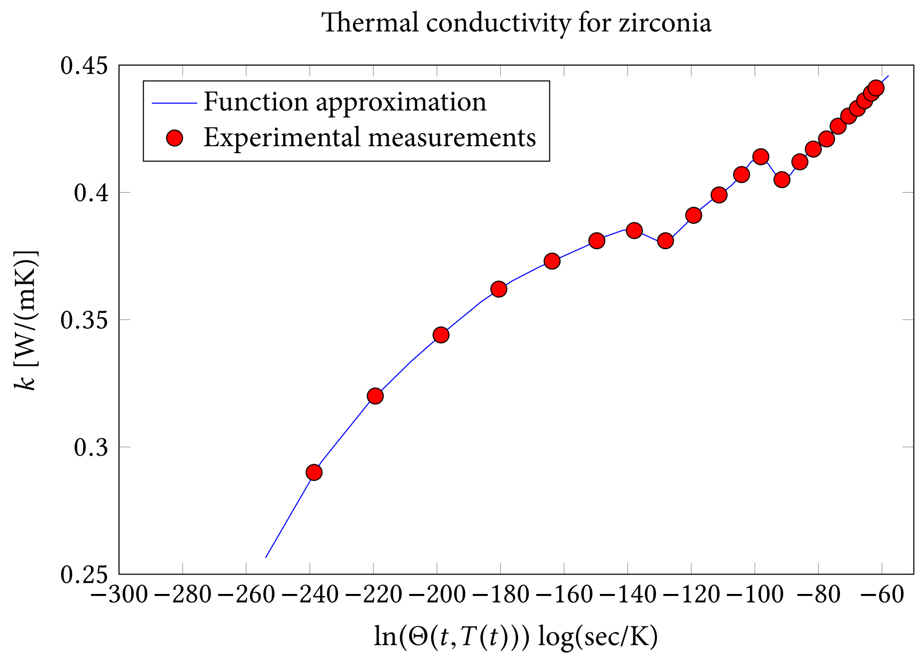


(a) Evolution of c_p with the work of sintering Θ for zirconia.

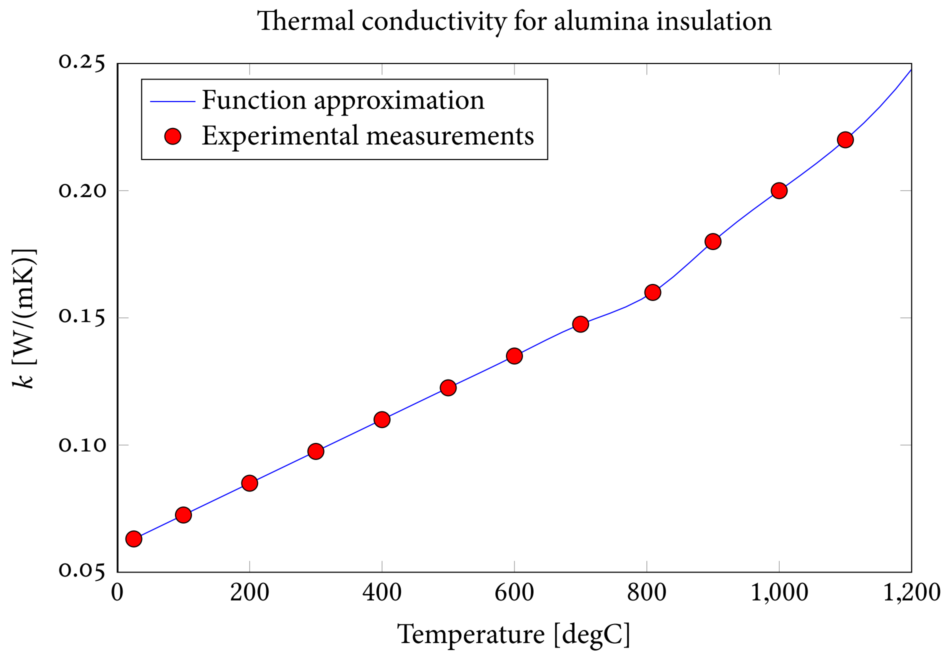


(b) Evolution of c_p with temperature for alumina insulation.

Figure 11.4: The curves, found using third-degree b-splines, describing the evolution of the specific heat capacity c_p [J/°C] of zirconia material and alumina insulation. Points represent measured input data from [5].



(a) Evolution of k with work of sintering Θ for zirconia.



(b) Evolution of k with temperature for alumina insulation.

Figure 11.5: The curves, found using third-degree b-splines, describing the evolution of the thermal conductivity k [W/(m·°C)] of zirconia material and alumina insulation. Points represent measured input data from [5].

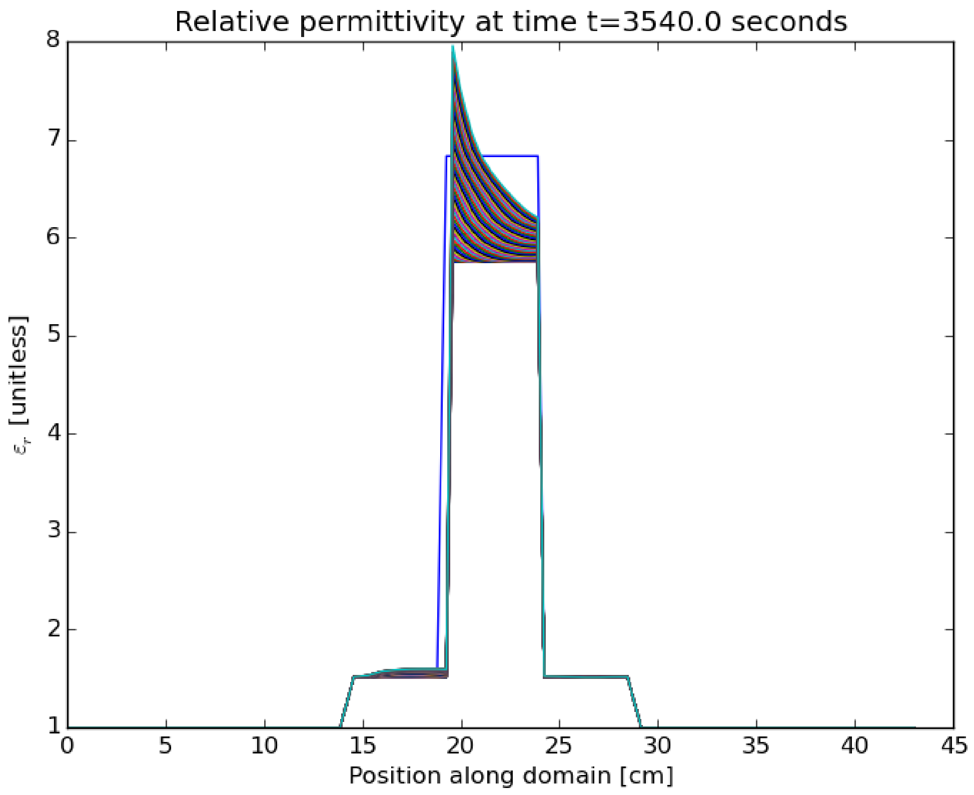


Figure 11.6: Simulated distribution of [unitless] relative permittivity $\varepsilon'_{\text{rel}}$ in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click 'play' button to view video of the evolution.

11.2 One-Dimensional Simulation with Lichtenecker Computation of Properties

This section describes the material property function results obtained for a one-dimensional simulation of sintering with process parameters and geometry identical to the one in Section 11.1, but with the zirconia parameters assumed to be calculable using an inversion of Lichtenecker's formula, rather than assumed to be functions of the work of sintering Θ.

Insulation and Material Property Functions

Activation energy Q for zirconia was approximated using the Nelder-Mead algorithm, where the objective function was the error in the optimal sigmoid curve describing the relationship between

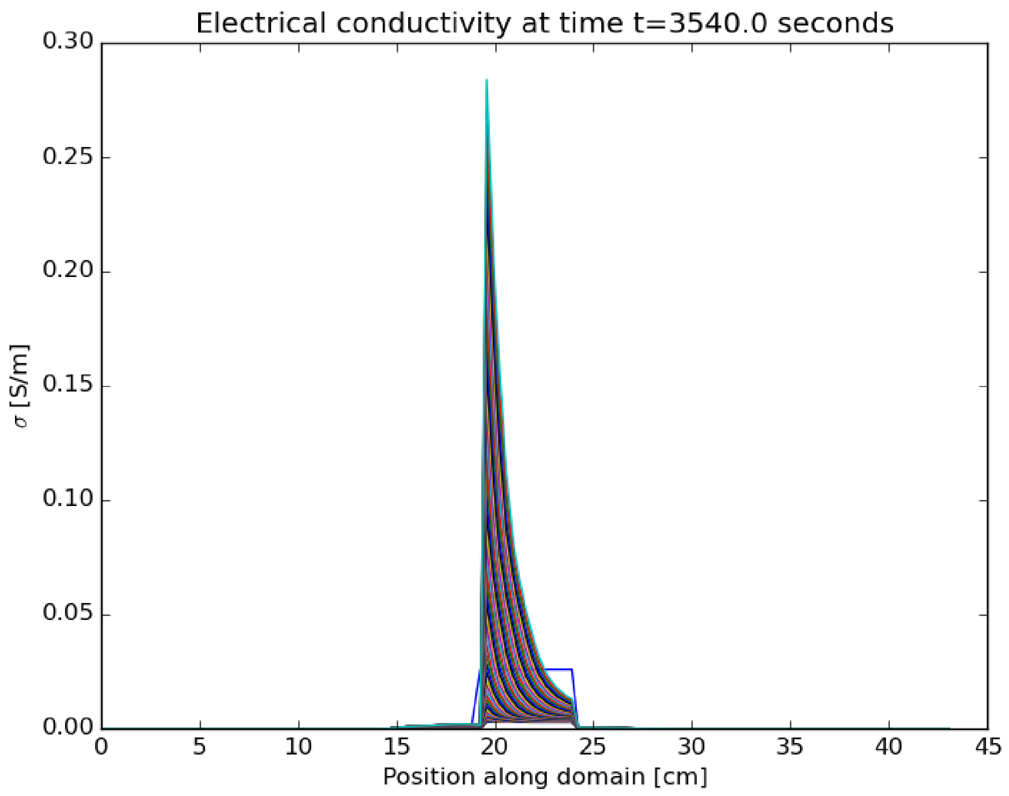


Figure 11.7: Simulated distribution of electrical conductivity σ [S/m] in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click ‘play’ button to view video of the evolution.

density and $\Theta(t, T(t))$, where Θ was computed as a function of Q using the experimental data on time and temperature taken, this time, from [4]. At each evaluation of the objective function, the optimal-fit sigmoid curve was found using Levenberg-Marquardt optimization to minimize the error between the function [166]

$$\rho_{\text{rel}}(\Theta) = \rho_0 + \frac{A}{\left[1 + \exp\left(-\frac{\ln(\Theta) - \ln(\Theta_0)}{B}\right)\right]^C}$$

and the measured density values through the course of sintering, where ρ_0 and Θ_0 are the initial values of relative density and Θ at the start of the sintering experiments, and where A, B, and C are the parameters adjusted in the course of Nelder-Mead optimization. The entire routine for finding the optimal activation energy took a total of 66 function evaluations and required approximately 77.8 CPU-seconds to perform, including overhead for plotting and saving results. The optimal Q

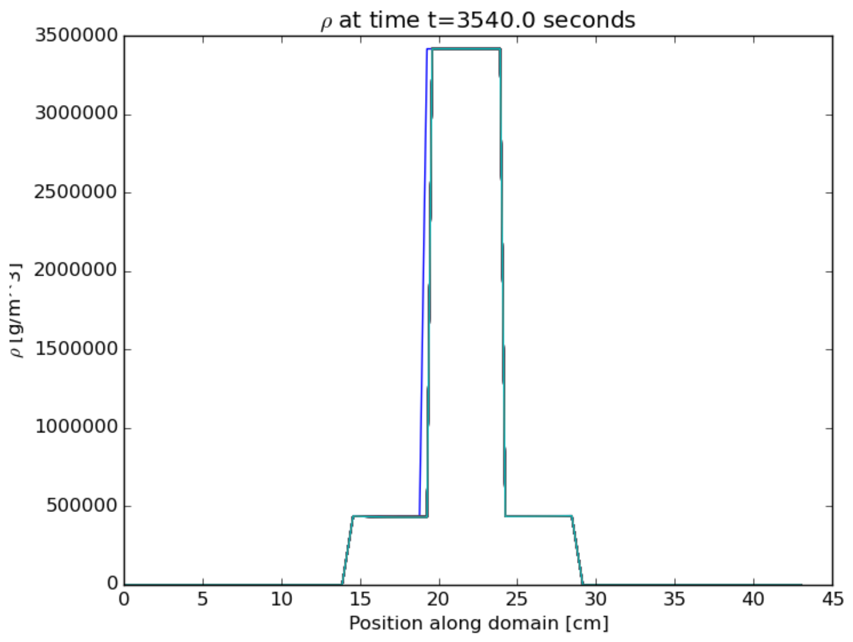


Figure 11.8: Simulated distribution of absolute density ρ [g/m^3] in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click 'play' button to view video of the evolution.

value was found to be approximately 653298 J/mol, which is within the range of values (615 ± 80 kJ/m) found in [187], and is also within the range of values found in [186, 188–192]. Using this value of Q, the corresponding optimal sigmoid curve hit the data points with a mean relative residual of 0.007867. This curve, which was found using the data in [3], is shown in Figure 11.18a, and is given by

$$\rho_{\text{rel}} = 0.461781 + \frac{0.428178}{\left[1 + \exp\left(-\frac{\ln \Theta + 53.758}{1.7004}\right)\right]^{0.441256}}. \tag{11.2}$$

The dielectric and thermal properties of zirconia were determined as functions of temperature and density by assuming explicit dependence on temperature and density and using the Lichtenecker formula as described in Section 5.2. The interpolated values of effective bulk properties were determined using third-degree b-splines, and the functions for the properties of the mixture, along with the measured data points they interpolate, are shown in Figures 11.16a, 11.17a, 11.19a, and 11.20a. Because zirconia is not a magnetic material, the magnetic permeability μ was assumed to be the same as free space; that is, $\mu = \mu_0$, or $\mu_{\text{rel}} = 1$.

The dielectric and thermal properties of alumina insulation were determined as functions of temperature using interpolation by third-degree b-splines, and the functions, along with the mea-

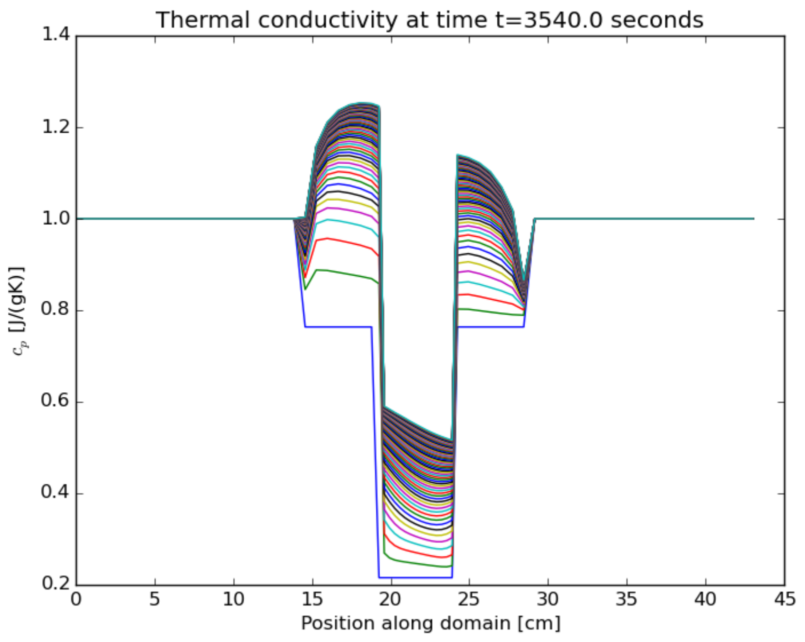


Figure 11.9: Simulated distribution of specific heat capacity c_p [J/°C] in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click 'play' button to view video of the evolution.

sured data points they interpolate, are shown in Figures 11.1b, 11.2b, 11.3b, 11.4b, and 11.5b. Because alumina is not a magnetic material, the magnetic permeability μ was assumed to be the same as free space; that is, $\mu = \mu_0$, or $\mu_{\text{rel}} = 1$.

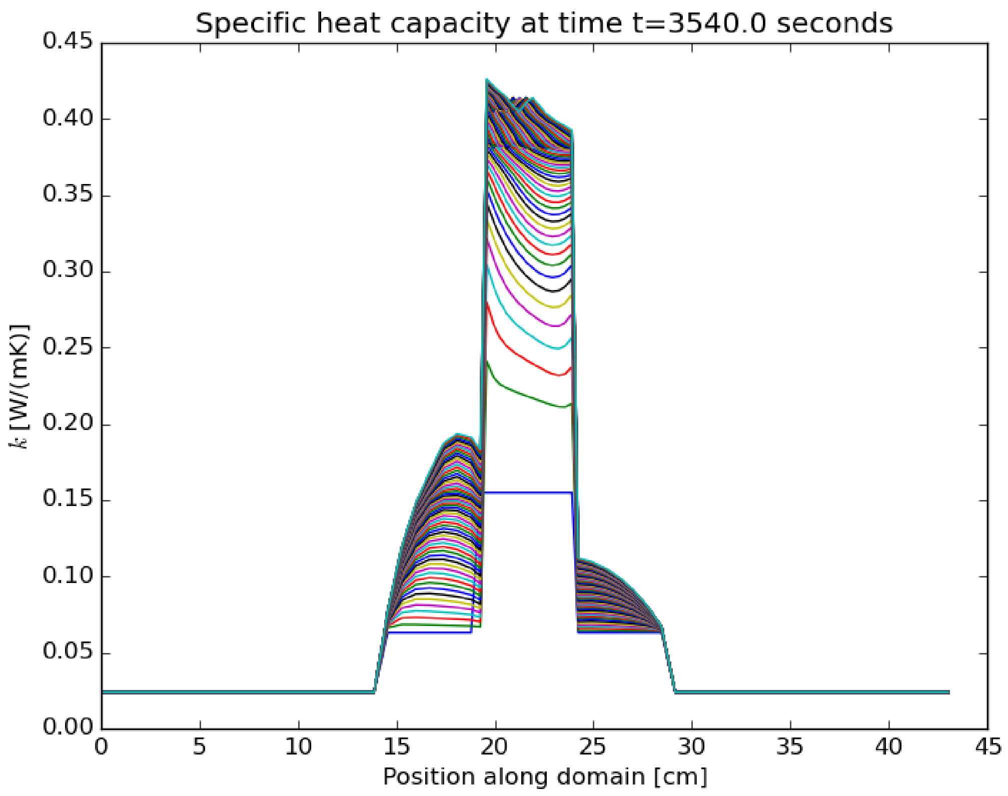


Figure 11.10: Simulated distribution of thermal conductivity k [W/(m·°C)] in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click 'play' button to view video of the evolution.

11.3 Two-Dimensional Simulation

Input Measurements and Parameters

Our trial again simulated the microwave thermal processing of zirconia (ZrO_2) surrounded by alumina (Al_2O_3) insulation, this time with a two-dimensional domain as shown in Figure 1.2. Experimentally measured input data to the model included measurements of ε, μ, σ, c_p, ρ, and k throughout the temperature range of sintering experiments described in [5] and performed by Ron Hutcheon at Microwave Properties North, which is reproduced in Table 10.1. The data required for the Master Sintering Curve method was obtained from [3] and [4], and is reproduced in Table 10.2.

The frequency of radiation was assumed to be 2.45 GHz, and the input power level was set to 1 kW. The domain length was set to be 43.35 cm, which was 2.5 times one wavelength in the waveg-

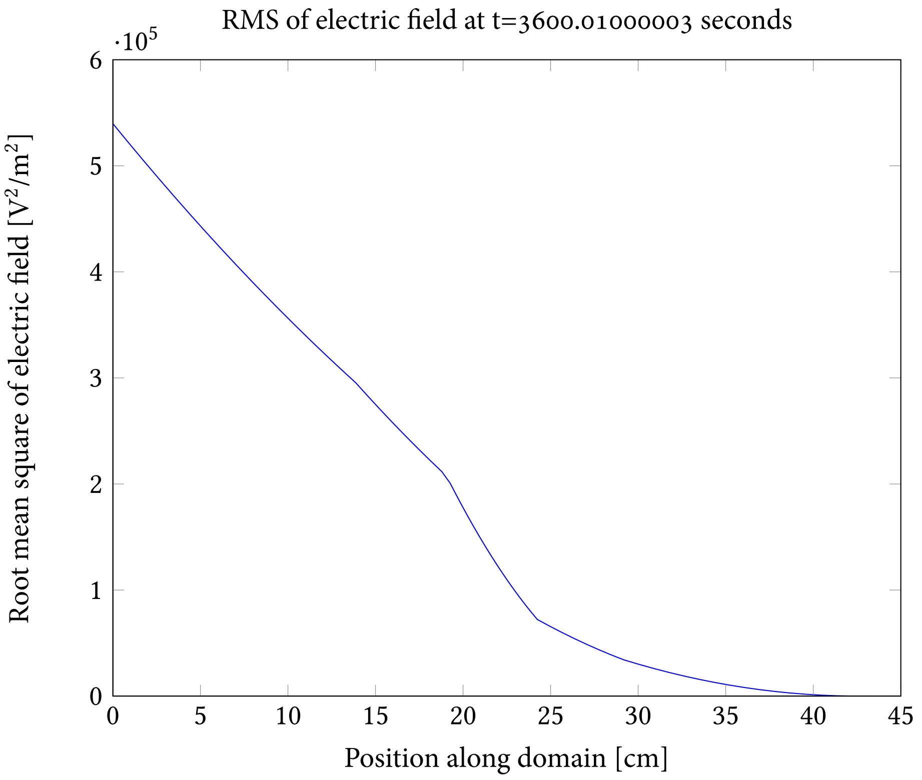


Figure 11.11: Simulated root mean square of electric field after processing.

uide, where length of the guided wave was computed according to Equations 2.46, 2.47, and 2.48, assuming that the cross-section of the waveguide was an 86.36×43.18 mm rectangle (this corresponds to the typical measurements of a D-band, WR-340 waveguide). The domain height was assumed to be 86.36 mm.

A sample of zirconia 4.82 cm long and 0.96 cm tall was centered in the waveguide's length, and the zirconia was assumed to be the center of an otherwise solid block of alumina insulation 14.45 cm long and 4.80 cm tall. The initial density of the zirconia was assumed to be 52.38 % of bulk density, in accordance with the value of bulk density of solid zirconia taken from [186]. Ambient temperature, which was assumed to be the same as the initial temperature of zirconia and alumina, was set to 20°C. The simulation was set to stop when the processing time reached 3600 seconds.

Insulation and Material Property Functions

As in the case of the one-dimensional simulation described in Section 11.1, the activation energy and MSC for zirconia were determined with the use of data from [3], and the resulting sigmoid function is identical to the one described in Equation 11.1. The properties of zirconia were again

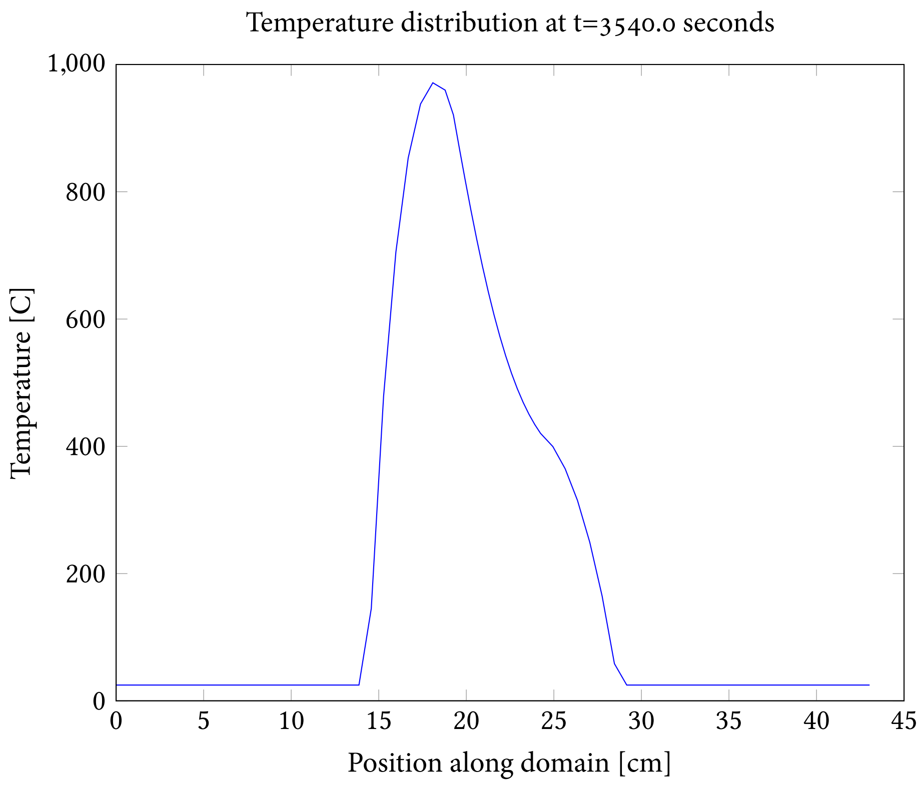


Figure 11.12: Simulated distribution of temperature in one-dimensional domain after processing.

assumed to depend on the work of sintering parameter Θ, and identical functions to those in Figures 11.1a, 11.2a, 11.4a, and 11.5a were found. Again here, the magnetic permeability μ was assumed to be the same as free space; that is, $\mu = \mu_0$, or $\mu_{\text{rel}} = 1$.

The dielectric and thermal properties of alumina insulation were also determined as functions of temperature using interpolation by third-degree b-splines, as in Section 11.1, and the functions, along with the measured data points they interpolate, are identical to those shown in Figures 11.1b, 11.2b, 11.3b, 11.4b, and 11.5b. For alumina, we also assume that $\mu = \mu_0$, or $\mu_{\text{rel}} = 1$.

The evolution of the material properties in the entire domain through time can be seen in Figures 11.21, 11.22, 11.23, 11.24, and 11.25.

Electric and Temperature Fields

Inside the simulation loop, the electric field was solved using the finite difference method for the Helmholtz equation, computed according to Equation 7.36. The perfect electric conducting boundary conditions were used on both the shorting and transverse walls, and the Dirichlet condition was used at the left-hand side. The evolution of the electric field during processing is shown in Fig-

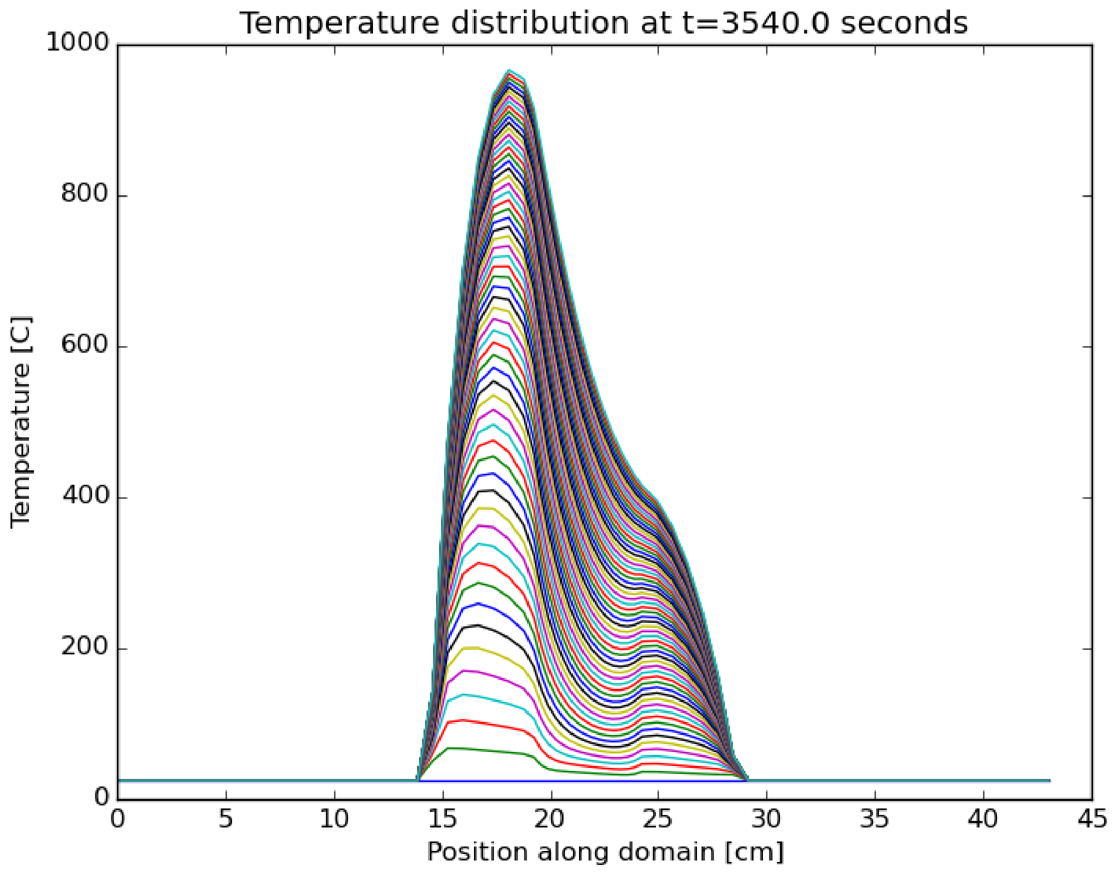


Figure 11.13: Simulated distribution of temperature in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click 'play' button to view video of the heating process.

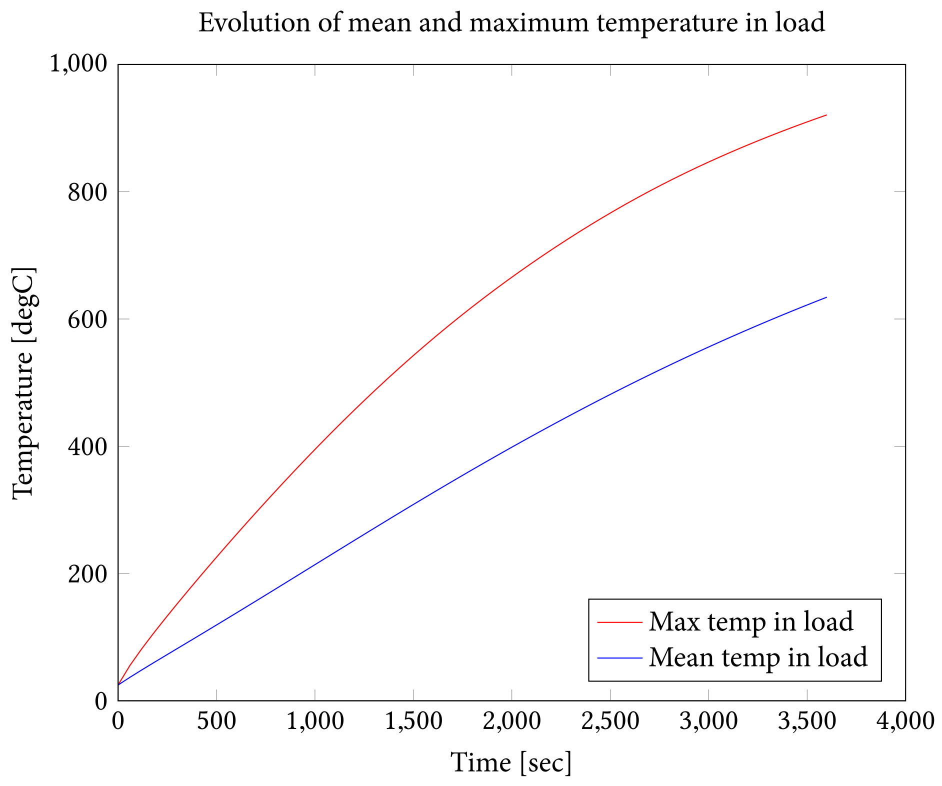


Figure 11.14: Simulated evolution of maximum and mean of temperature within zirconia sample during processing.

ure 11.26. The maximum, minimum, and mean values of the electric field at regular time intervals throughout the simulation can be found in the simulation output log file, partially reproduced in Appendix I.2.

There were clear peaks in the temperature field during processing, which, by the end of processing, smoothed to the distribution seen in Figure 11.27. The evolution of the temperature distribution through time may be seen in Figure 11.28, and the evolution of the maximum and mean temperatures within the load may be seen in Figure 11.29.

After 3600 seconds, the zirconia did not exhibit shrinkage, as the change in relative density to 52.54% of bulk density accounted for a physical shrinkage smaller than the height of one spatial grid cell. The maximum zirconia temperature reached during this processing was 454.7° C.

The entire simulation took 8902 CPU seconds to complete, including overhead for logging and plotting results.

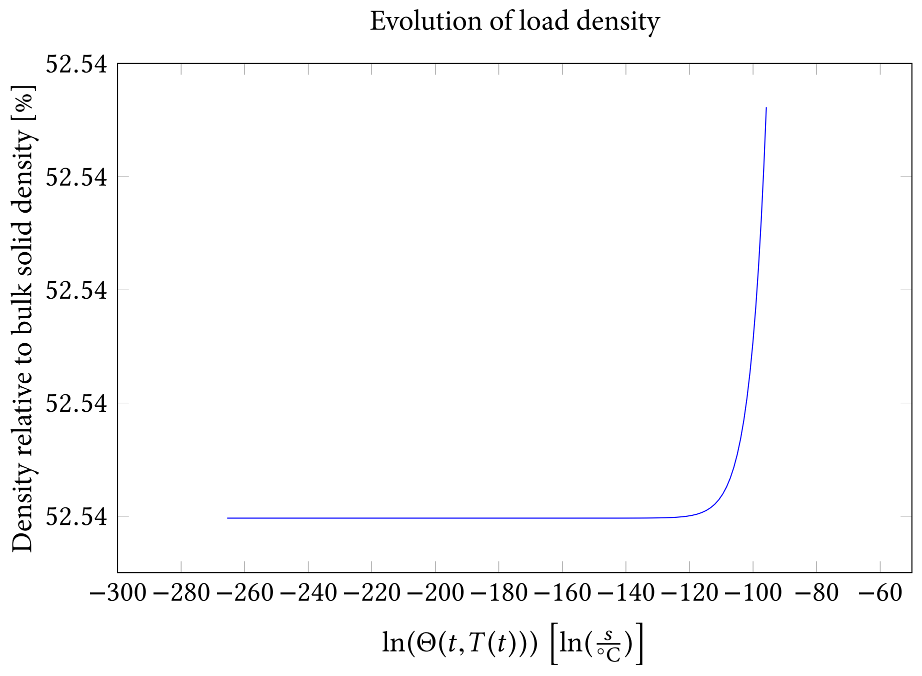
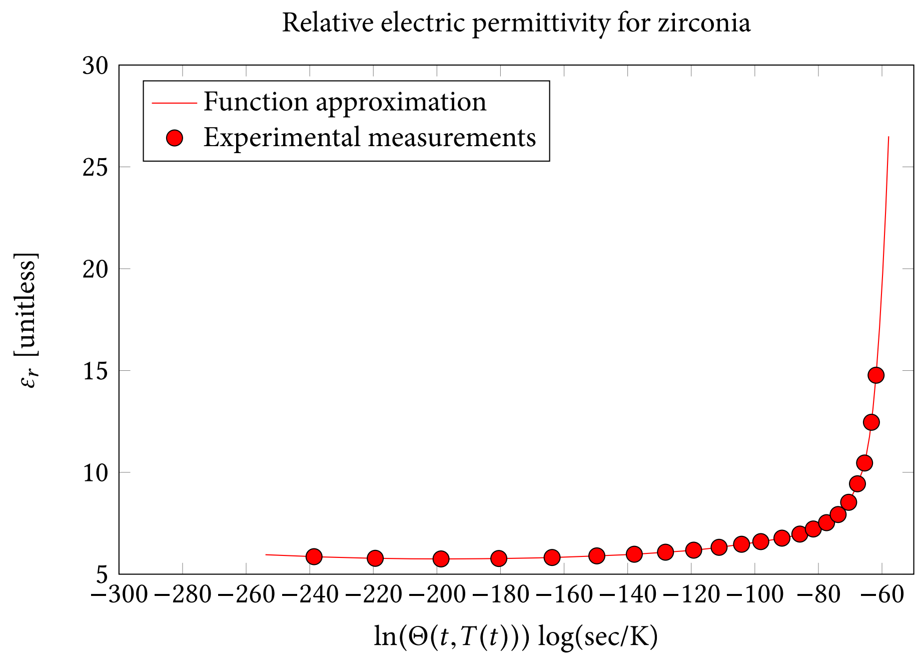
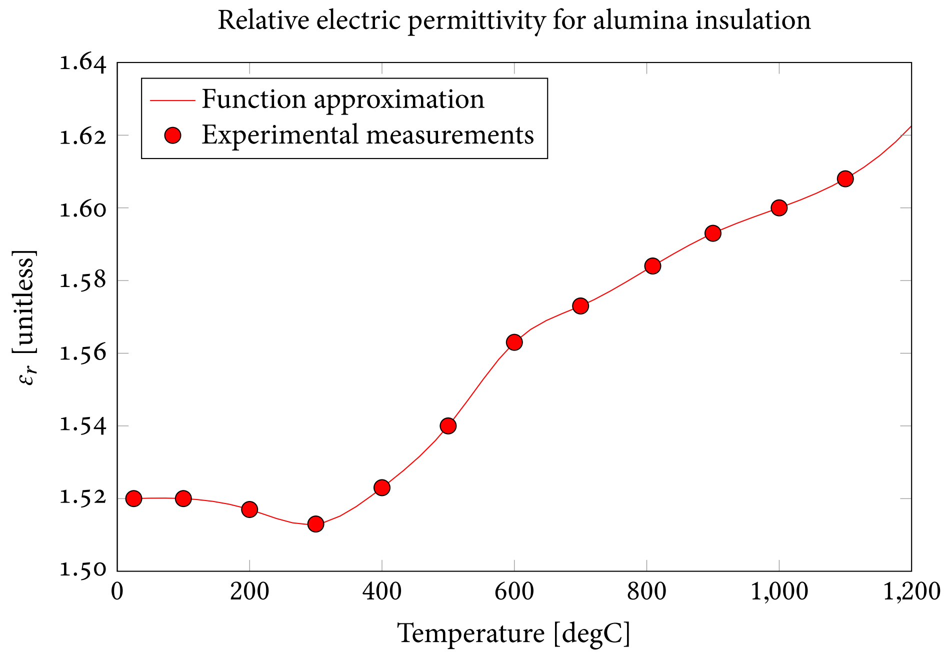


Figure 11.15: Evolution of ρ with the work of sintering Θ.

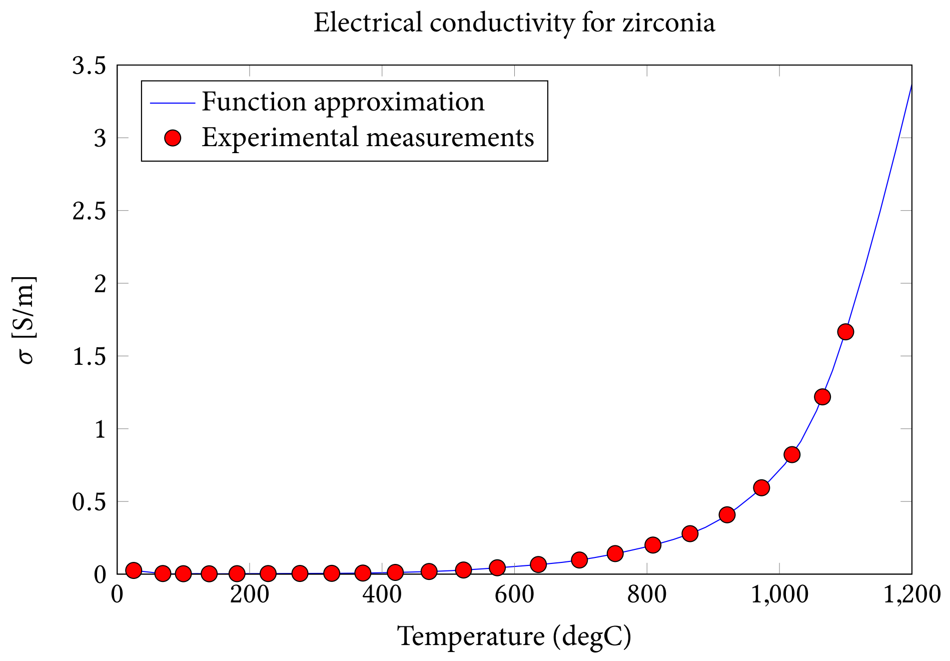


(a) Evolution of $\varepsilon'_{\mathrm{rel}}$ for zirconia with temperature for various values of densityzirconia.

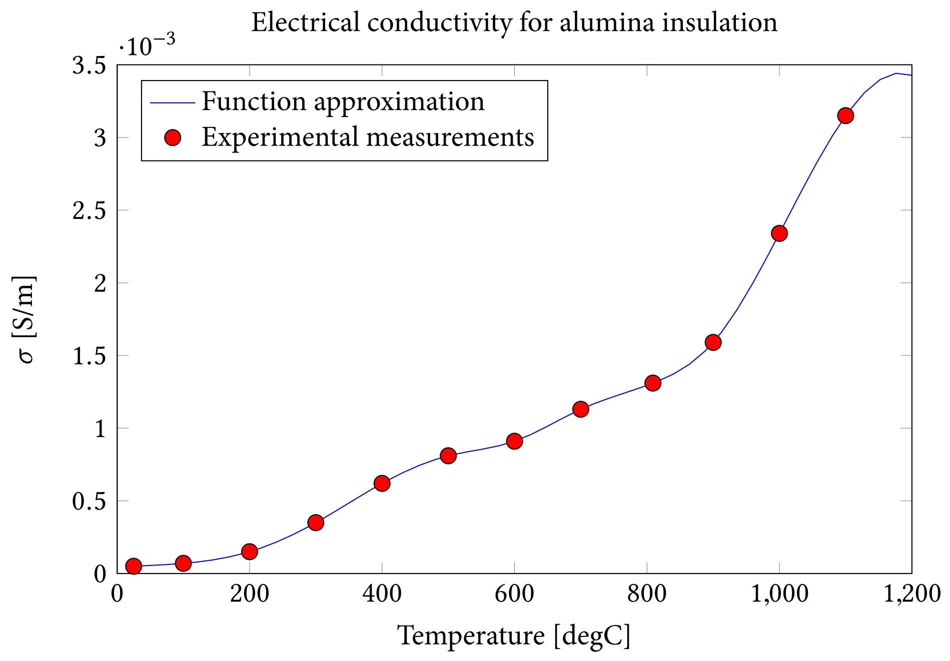


(b) Evolution of $\varepsilon'_{\mathrm{rel}}$ with temperature for alumina.

Figure 11.16: The curves, found using third-degree b-splines, describing the evolution of the dielectric constant $\varepsilon'_{\mathrm{rel}}$, relative to ε_0, of zirconia material and alumina insulation. Points represent measured input data from [5].

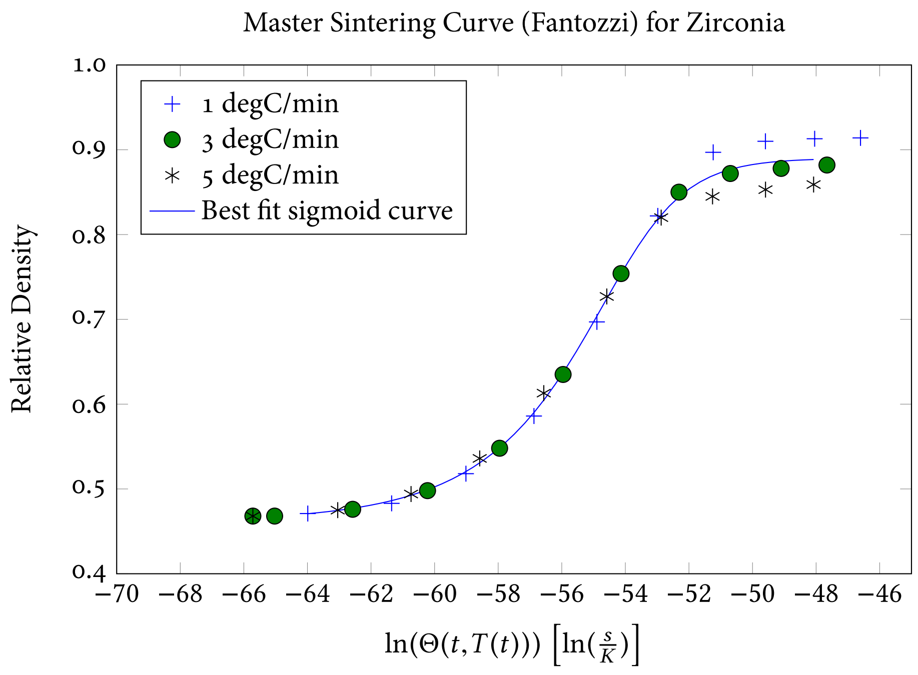


(a) Evolution of σ for zirconia with temperature for various values of density.

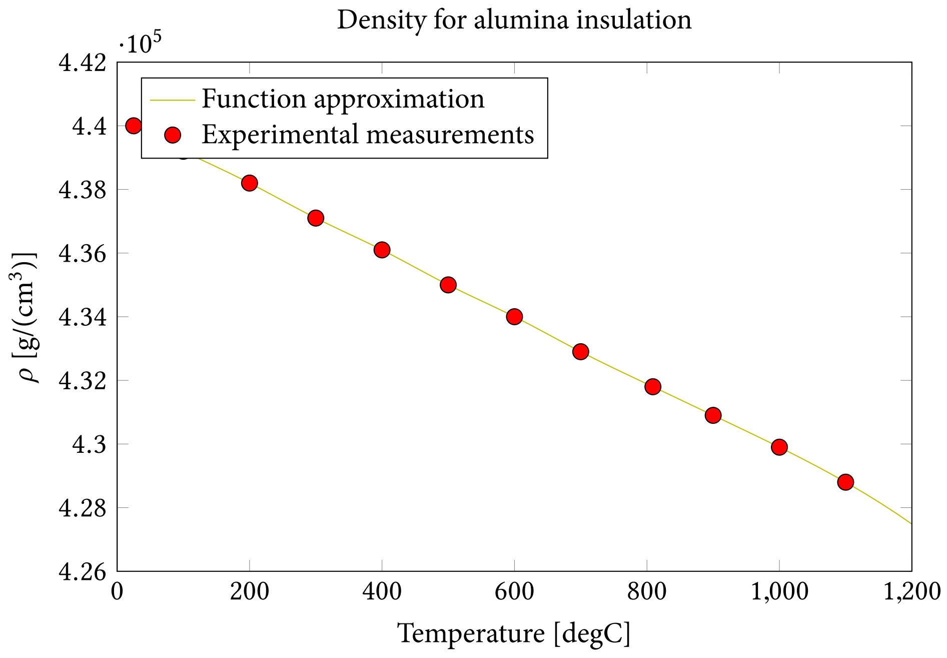


(b) Evolution of σ with temperature for alumina.

Figure 11.17: The curves, found using third-degree b-splines, describing the evolution of the electrical conductivity σ [S/m] of zirconia material and alumina insulation. Points represent measured input data from [5].

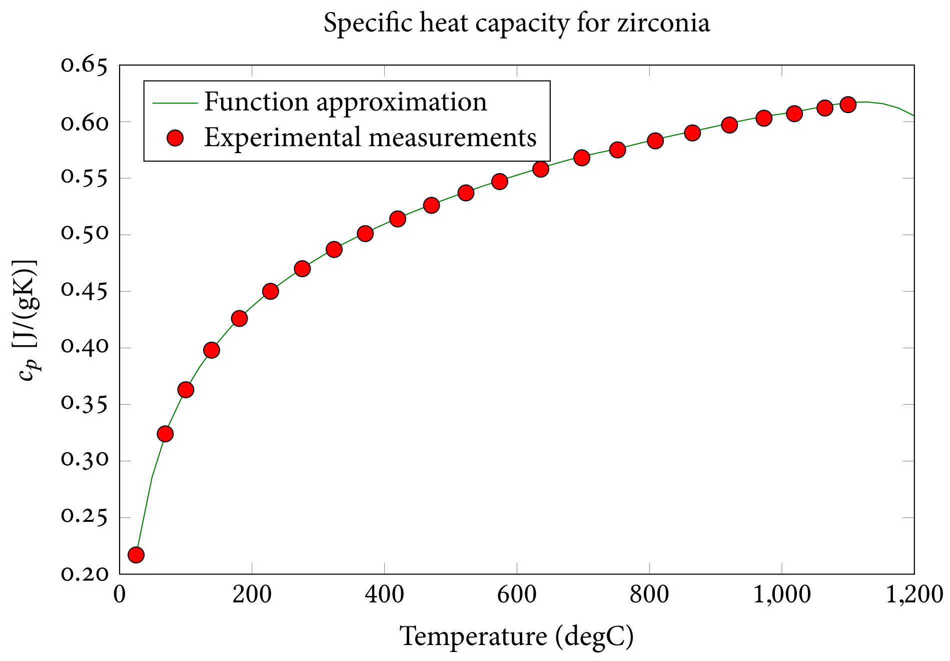


(a) The optimal-fit sigmoid curve describing the relationship between the work of sintering Θ and the [unitless] density ρ_{rel} of zirconia, relative to its bulk density, using data from [3] and the sigmoid curve described in Equation 11.2.

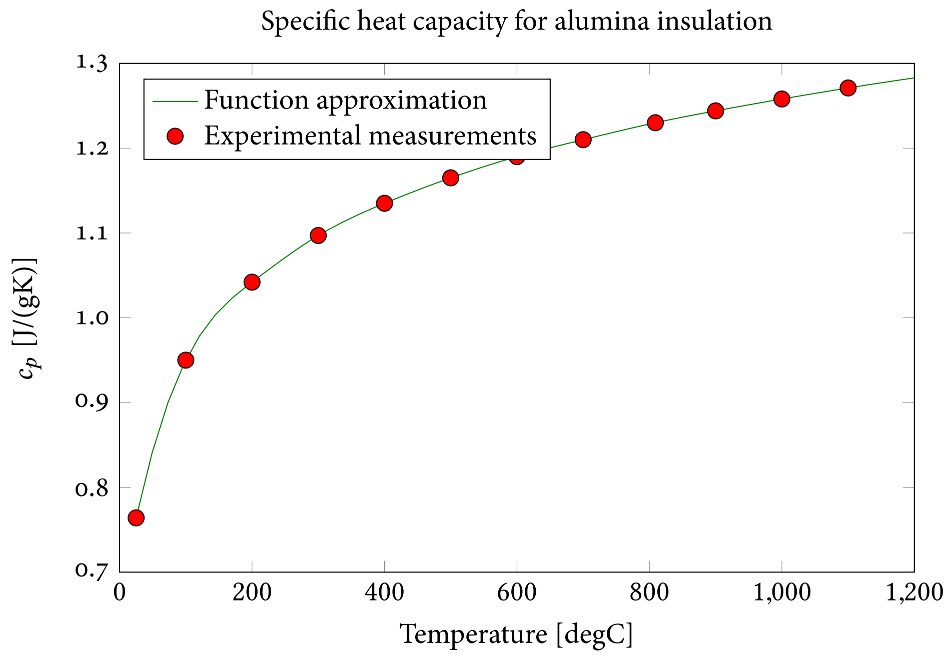


(b) The curves, found using third-degree b-splines, describing the evolution of absolute density ρ [g/m^3] of alumina insulation with temperature. Points represent measured input data from [5].

Figure 11.18: The functions describing the evolution of the density of zirconia material and alumina insulation.

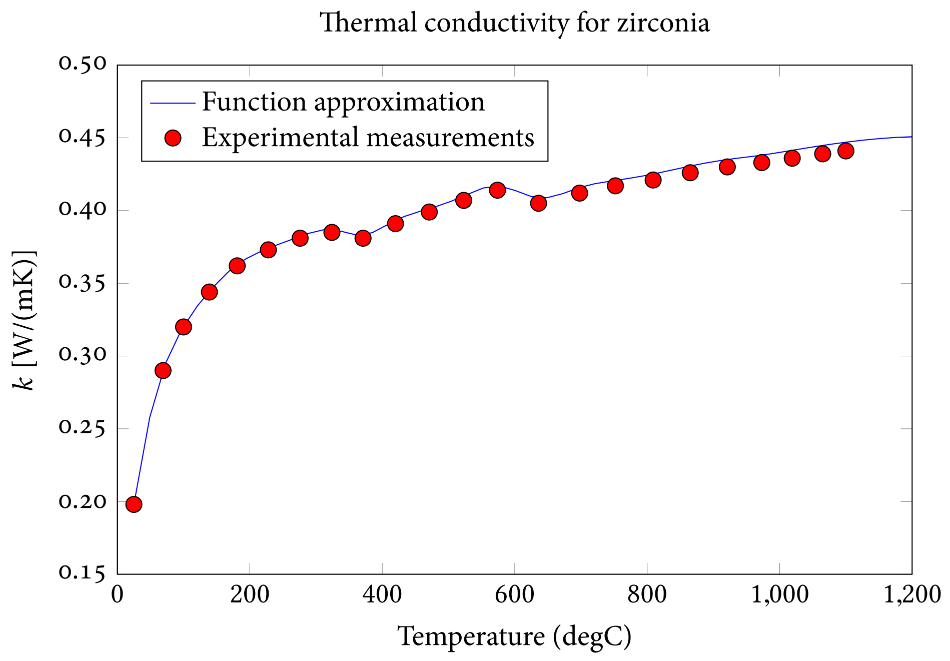


(a) Evolution of c_p for zirconia with temperature for various values of density.

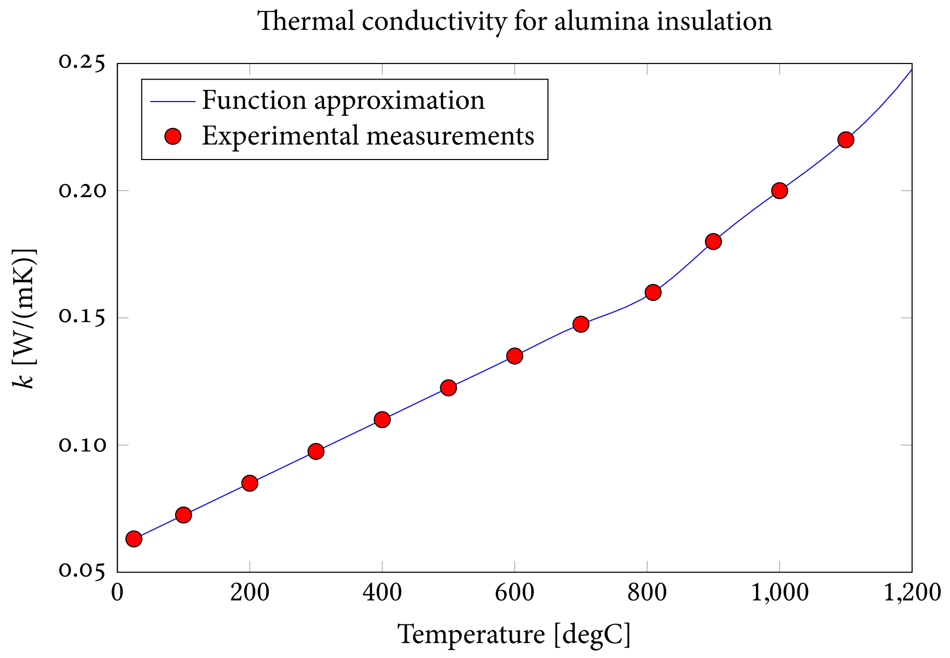


(b) Evolution of c_p with temperature for alumina insulation.

Figure 11.19: The curves, found using third-degree b-splines, describing the evolution of the specific heat capacity c_p [J/°C] of zirconia material and alumina insulation. Points represent measured input data from [5].



(a) Evolution of k for zirconia with temperature for various values of density.



(b) Evolution of k with temperature for alumina insulation.

Figure 11.20: The curves, found using third-degree b-splines, describing the evolution of the thermal conductivity k [W/(m·°C)] of zirconia material and alumina insulation. Points represent measured input data from [5].

If your PDF reader supports `mp4` playback, then click the 'play' button to view video.

Figure 11.21: Simulated distribution of [unitless] relative permittivity $\varepsilon'_{\mathrm{rel}}$ in one-dimensional domain during processing.

If your PDF reader supports `mp4` playback, then click the 'play' button to view video.

Figure 11.22: Simulated distribution of electrical conductivity σ [S/m] in one-dimensional domain during processing.

If your PDF reader supports `mp4` playback, then click the 'play' button to view video.

Figure 11.23: Simulated distribution of absolute density ρ [g/m^3] in one-dimensional domain during processing.

If your PDF reader supports `mp4` playback, then click the 'play' button to view video.

Figure 11.24: Simulated distribution of specific heat capacity c_p [J/°C] in one-dimensional domain during processing.

If your PDF reader supports mp4 playback, then click the 'play' button to view video.

Figure 11.25: Simulated distribution of thermal conductivity k [W/(m·°C)] in one-dimensional domain during processing.

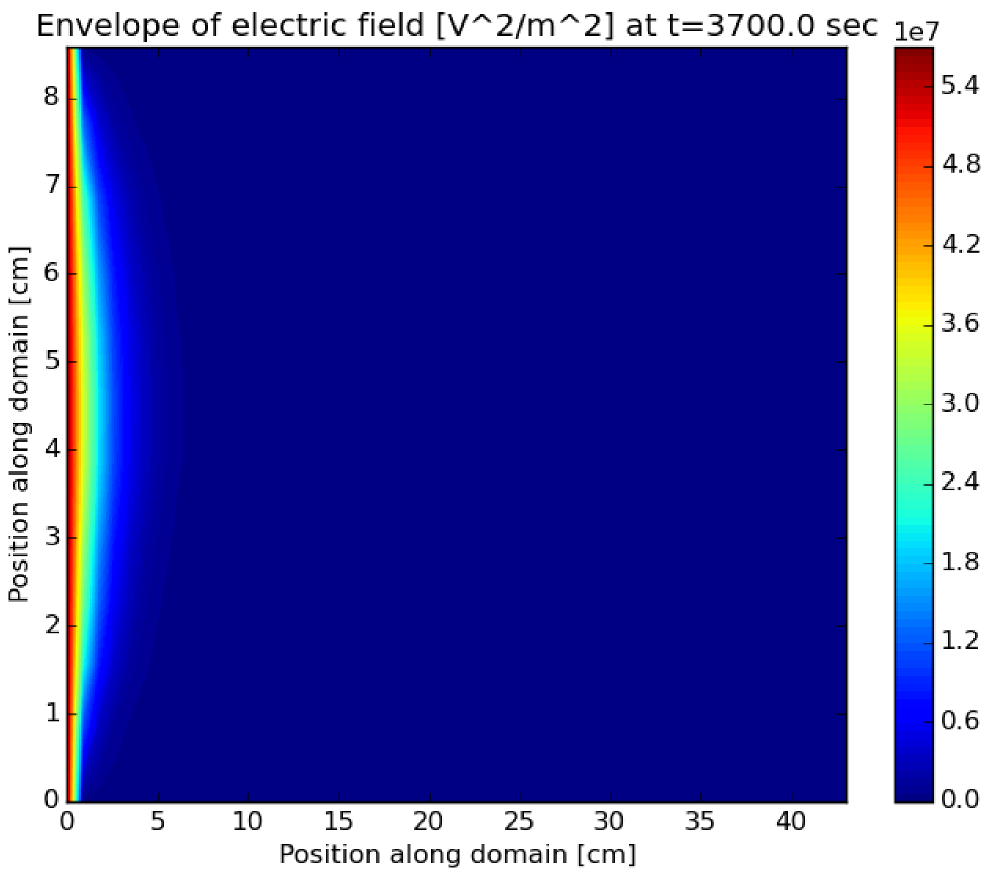


Figure 11.26: Simulated root mean square of electric field after processing.

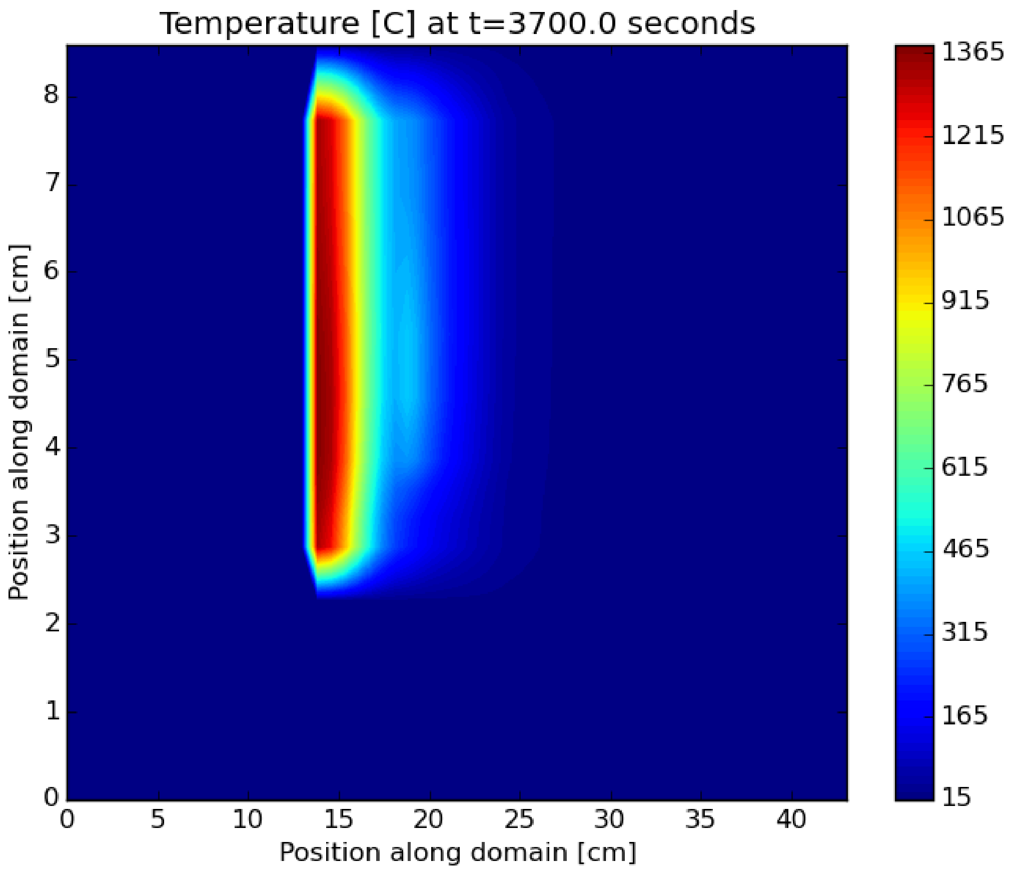


Figure 11.27: Simulated distribution of temperature in two-dimensional domain after processing.

If your PDF reader supports `mp4` playback, then click the 'play' button to view video.

Figure 11.28: Simulated distribution of temperature in one-dimensional domain during processing; if using Adobe Reader to view the current PDF file, click 'play' button to view video of the heating process.

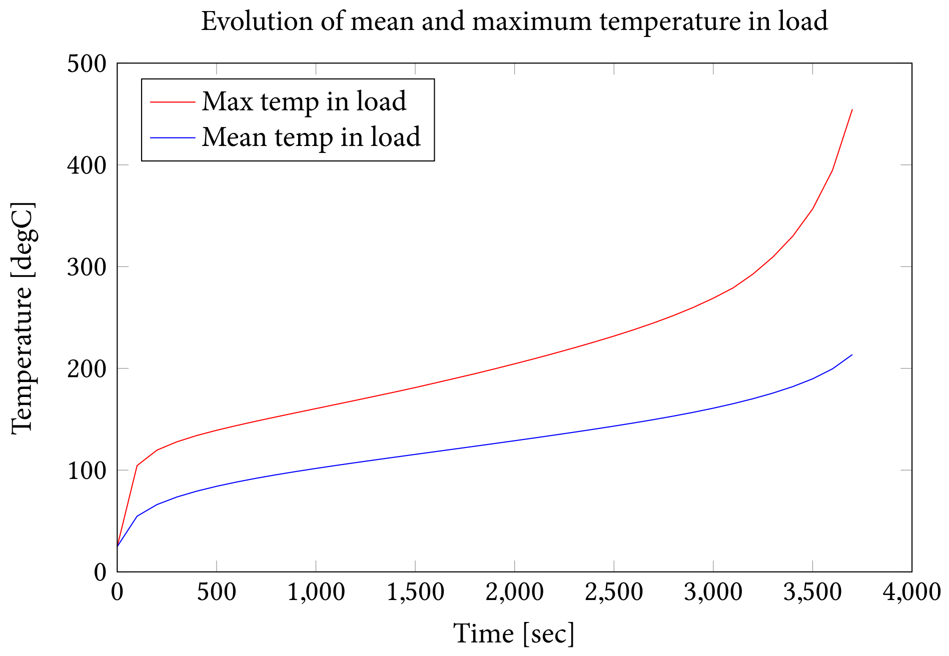


Figure 11.29: Simulated evolution of maximum and mean of temperature within zirconia sample during processing.

Chapter 12

Conclusions and Future Work

In Chapters 2, 3, and 4, we have described the electromagnetic, thermal, and mechanical phenomena that occur during microwave sintering. We presented algorithms for the solution of each of these problems, and synthesized these into a coupled, iterative routine described in Chapter 10. In Chapter 7, we showed finite difference solvers for the one- and two-dimensional wave equations, and finite element and analytical solutions for the one- and two-dimensional Helmholtz equation, and in Chapter 8, we show finite difference solvers for the one- and two-dimensional heat equation. In Chapter 9, we demonstrate the use of the Master Sintering Curve to simulate the density kinetics of matter undergoing sintering, and we provide a novel use of the exponential integral function in order to speed up computation of the work of sintering parameter Θ that is an input to the MSC.

In formulating the coupled multiphysics routine, we have discussed several physical and technical aspects of microwave sintering that warrant careful treatment, including strong multiphysics coupling via material parameter dependence on temperature and/or relative density, whose treatment via inversions of classical and contemporary mixture formulas is described in Chapter 5; the vastly different time scales on which the three key physical processes evolve, whose resolution is described in Section 10.2; and the wide spectrum of physically relevant spatial scales, which we synthesize using the MSC as described in Chapter 4.

The extremely important role played by the accurate and adequate determination of material parameters and their temperature and density dependence was also emphasized, as well as its particular importance in the context of microwave sintering.

We have described and presented computer implementations comprehensive models of microwave sintering in one and two dimensions that rely on a small set of simplifying assumptions, and these models, for the first time, accounts for density dependence of material properties as well as temperature dependence. Results of these simulations were presented in Chapter 11 for the simulated sintering of zirconia surrounded by alumina insulation, and the resulting temperature increase and percent of shrinkage appear to be consistent with experimental results reported in literature.

This work also lays a theoretical and computational foundation for modelling the general three-dimensional problem and computer-aided design of efficient sintering processes. Certain improvements, though, could still widen the range of materials to which our model applies, and other techniques may improve its computer implementation.

For certain materials, the expectation of anisotropy or nonuniformity of the stress tensor may be a valid one, and in these cases, the conservation of mass technique described in Section 9.2 is a less valid way of determining mechanical deformation. For these scenarios, the MSC should be used to determine density kinetics, and this should be used together with the constitutive relation in Equation 4.4 to directly simulate the mechanical deformation as a result of changes to the strain rate tensor. Preliminary work has already been done toward this end, as three `MATLAB` codes in Appendix F.3 are capable of converting the strain rate tensor to mechanical deformation in one, two, and three dimensions.

The explicit incorporation of the strain rate tensor into these routines would also enable the solver to account for deformation in the two-dimensional case that is not uniform; that is, the currently-used solution method, described in Section 9.2, based on an averaging of the relative density change throughout the entire sample, is incapable of accounting for the situation when certain portions of the material may shrink more than other portions.

Given the wide variety of techniques that are currently used for modelling conventional sintering on a spectrum of spatial scales, there are several future directions that this work could take. The work would benefit from mathematical homogenization techniques applied to the problem of mechanical deformation, because the explicit inclusion of certain micromechanical variables in the analysis could help to clarify the role of these variables on the outcome of the sintering process.

The nature of the solution method we present in Chapter 10 for the coupled routine also lends itself readily to alternative solution methods for either of the electromagnetic or thermal problems to be swapped in as substitutes for the finite difference methods described in Chapters 7 and 8. Integral equation solutions of the wave equation have been studied [193] and may be applicable in the scenarios we consider, and in simple cases, analytical solutions of the Helmholtz equation may also be employed for the one- and two-dimensional problems [67].

Indeed, finite element methods may prove more useful when the three-dimensional applications of this model are studied, as these methods may prove to more adequately handle certain irregularities in the geometrical configurations most likely to be encountered during actual sintering experiments and trials. The three-dimensional electromagnetic and thermal problems have been thoroughly studied in this context, and the problem of mechanical deformation in three dimensions would also be readily solved by integration of the strain rate tensor using, for example, the preliminary work in Appendix F.3.

Another avenue for expansion of this work could be in the incorporation of nonthermal effects of microwaves on the process of sintering. In crystalline solids such as those typically considered in studies of sintering, mass transport has been demonstrated to proceed preferentially along the electric field vector [194], which results in elongation of pores in comparison to traditional sintering experiments [195]. Indeed, experimental comparisons of microwave and conventional sintering reveal different patterns even under control of the heating rate [167]. The *ponderomotive effect* in microwave sintering has been studied in the context of modelling [196], but not extensively, and the present model would benefit from future investigation of this phenomenon.

Appendix A

Vector and Matrix Entries in the Finite Difference Approximation for the Solution of the Two-Dimensional Wave Equation

This appendix contains the vectors and matrices of Equation 7.27.

A.1 Coefficients for the Finite Difference Solver of the Wave Equation

$$\vec{E}^p := \left[\begin{array}{c} E^p_{0,0} \\ E^p_{0,1} \\ \vdots \\ E^p_{0,M-1} \\ \hdashline E^p_{1,0} \\ E^p_{1,1} \\ \vdots \\ E^p_{1,M-1} \\ \hdashline \vdots \\ \hdashline E^p_{N-1,0} \\ E^p_{N-1,1} \\ \vdots \\ E^p_{N-1,M-1} \end{array}\right],$$

where

$$\vec{c} := \left[\begin{array}{c} 0 \\ \vdots \\ 0 \\ \hdashline 0 \\ -\mu\varepsilon' + \frac{\mu\sigma\Delta t}{2} \\ \vdots \\ -\mu\varepsilon' + \frac{\mu\sigma\Delta t}{2} \\ 0 \\ \hdashline \vdots \\ \hdashline 0 \\ -\mu\varepsilon' + \frac{\mu\sigma\Delta t}{2} \\ \vdots \\ -\mu\varepsilon' + \frac{\mu\sigma\Delta t}{2} \\ 0 \\ \hdashline -\frac{1}{c(\Delta t)^2} \\ \vdots \\ -\frac{1}{c(\Delta t)^2} \end{array}\right],$$

and the multiplication $\vec{c}\vec{E}^{n-1}_{j,k}$ is component-wise, and where matrices A and B are given by

$$A := \left[\begin{array}{c:c:c:c:c}
\begin{matrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{matrix} & & & & \\
\hdashline
\begin{matrix} 0 & & & & \\ & a_1 & & & \\ & & \ddots & & \\ & & & a_1 & \\ & & & & 0 \end{matrix} &
\begin{matrix} 1 & & & & \\ c_1 & b_{1,1} & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{M-2} & b_{1,M-2} & c_{M-2} \\ & & & & 1 \end{matrix} &
\begin{matrix} 0 & & & & \\ & a_1 & & & \\ & & \ddots & & \\ & & & a_1 & \\ & & & & 0 \end{matrix} & & \\
\hdashline
 & \ddots & \ddots \quad \ddots \quad \ddots & \ddots & \\
\hdashline
 & & \begin{matrix} 0 & & & & \\ & a_{N-2} & & & \\ & & \ddots & & \\ & & & a_{N-2} & \\ & & & & 0 \end{matrix} &
\begin{matrix} 1 & & & & \\ c_1 & b_{N-2,1} & c_1 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{M-2} & b_{N-2,M-2} & c_{M-2} \\ & & & & 1 \end{matrix} &
\begin{matrix} 0 & & & & \\ & a_{N-2} & & & \\ & & \ddots & & \\ & & & a_{N-2} & \\ & & & & 0 \end{matrix} \\
\hdashline
 & & & & \begin{matrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{matrix}
\end{array}\right],$$

$$
B := \left[\begin{array}{c:c:c:c:c}
\begin{matrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 \end{matrix} & & & & \\
\hdashline
\begin{matrix} 0 & & & \\ & d_1 & & \\ & & \ddots & \\ & & & d_1 \\ & & & & 0 \end{matrix} &
\begin{matrix} 0 & & & & \\ f_1 & e_{1,1} & f_1 & & \\ & \ddots & \ddots & \ddots & \\ & & f_{M-2} & e_{1,M-2} & f_{M-2} \\ & & & & 0 \end{matrix} &
\begin{matrix} 0 & & & \\ & d_1 & & \\ & & \ddots & \\ & & & d_1 \\ & & & & 0 \end{matrix} & & \\
\hdashline
& \ddots & \ddots \quad \ddots \quad \ddots & \ddots & \\
\hdashline
& & \begin{matrix} 0 & & & \\ & d_{N-2} & & \\ & & \ddots & \\ & & & d_{N-2} \\ & & & & 0 \end{matrix} &
\begin{matrix} 0 & & & & \\ f_1 & d_{N-2,1} & f_1 & & \\ & \ddots & \ddots & \ddots & \\ & & f_{M-2} & e_{N-2,M-2} & f_{M-2} \\ & & & & 0 \end{matrix} &
\begin{matrix} 0 & & & \\ & d_{N-2} & & \\ & & \ddots & \\ & & & d_{N-2} \\ & & & & 0 \end{matrix} \\
\hdashline
& & & & \begin{matrix} 0 & & & & \\ & 0 & & & \\ & & \ddots & & \\ & & & 0 & \\ & & & & 0 \end{matrix}
\end{array}\right],
$$

where

$$
\begin{aligned}
a_j &:= -\theta s_j \\
b_{j,k} &:= 2\theta s_j + 2\phi r_k + \mu\varepsilon' + \frac{\mu\sigma\Delta t}{2} \\
c_k &:= -\phi r_k \\
d_j &:= (1-\theta)s_j \\
e_{j,k} &:= -2(1-\theta)s_j - 2(1-\phi)r_k + 2\mu\varepsilon' \\
f_k &:= (1-\phi)r_k
\end{aligned}
$$

A.2 Coefficients for the Finite Difference Solver of the Helmholtz Equation

The matrix A from Equation 7.36 is

$$
A = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
1&&&& & &&&& & &&&& & &&&& & &&&& \\
&1&&& & &&&& & &&&& & &&&& & &&&& \\
&&\ddots&& & &&&& & &&&& & &&&& & &&&& \\
&&&1& & &&&& & &&&& & &&&& & &&&& \\
&&&&1 & &&&& & &&&& & &&&& & &&&& \\
\hline
0&&&& & 1&&&& & 0&&&& & &&&& & &&&& \\
&a_1&&& & b_1&c_1^1&b_1&& & &a_1&&& & &&&& & &&&& \\
&&\ddots&& & &\ddots&\ddots&\ddots& & &&\ddots&& & &&&& & &&&& \\
&&&a_1& & &&b_{M-1}&c_1^{M-1}&b_{M-1} & &&&a_1& & &&&& & &&&& \\
&&&&0 & &&&0&1 & &&&&0 & &&&& & &&&& \\
\hline
&&&& & &&\ddots&& & &\ddots&\ddots&\ddots& & &&\ddots&& & &&&& \\
\hline
&&&& & &&&& & 0&&&& & 1&&&& & 0&&&& \\
&&&& & &&&& & &a_{N-1}&&& & b_1&c_{N-1}^1&b_1&& & &a_{N-1}&&& \\
&&&& & &&&& & &&\ddots&& & &\ddots&\ddots&\ddots& & &&\ddots&& \\
&&&& & &&&& & &&&a_{N-1}& & &&b_{M-1}&c_{N-1}^{M-1}&b_{M-1} & &&&a_{N-1}& \\
&&&& & &&&& & &&&&0 & &&&&1 & &&&&0 \\
\hline
&&&& & &&&& & &&&& & &&&& & 1&&&& \\
&&&& & &&&& & &&&& & &&&& & &1&&& \\
&&&& & &&&& & &&&& & &&&& & &&\ddots&& \\
&&&& & &&&& & &&&& & &&&& & &&&1& \\
&&&& & &&&& & &&&& & &&&& & &&&&1
\end{array}\right],
$$

where $a_j := \frac{1}{(\Delta z)_j^2}$, $b_k := \frac{1}{(\Delta x)_k^2}$, and $c_j^k := \mu^2 \varepsilon \omega^2 - \frac{2}{(\Delta z)_j^2} - \frac{2}{(\Delta x)_k^2}$.

Appendix B

Vector and Matrix Entries in the Finite Difference Approximation for the Solution of the Two-Dimensional Heat Equation

This appendix contains the vectors and matrices of Equation 8.24.

$$\vec{u}^p := \begin{bmatrix} u^p_{0,0} \\ u^p_{0,1} \\ \vdots \\ u^p_{0,M-1} \\ \hdashline u^p_{1,0} \\ u^p_{1,1} \\ \vdots \\ u^p_{1,M-1} \\ \hdashline \vdots \\ \hdashline u^p_{N-1,0} \\ u^p_{N-1,1} \\ \vdots \\ u^p_{N-1,M-1} \end{bmatrix},$$

$$\vec{Q}^p := \left[\begin{array}{c}
q^p_{0,0} - \frac{s^n_0(z_1-z_{-1})g_1}{\alpha_{1,1}} - \frac{r^n_0(x_1-x_{-1})g_3}{\alpha_{1,3}} \\
q^p_{0,1} - \frac{s^n_0(z_1-z_{-1})g_1}{\alpha_{1,1}} \\
\vdots \\
q^p_{0,M-2} - \frac{s^n_0(z_1-z_{-1})g_1}{\alpha_{1,1}} \\
q^p_{0,M-1} - \frac{s^n_0(z_1-z_{-1})g_1}{\alpha_{1,1}} + \frac{r^n_{M-1}(x_M-x_{M-2})g_4}{\alpha_{1,4}} \\
\hdashline
q^p_{1,0} - \frac{r^n_0(x_1-x_{-1})g_3}{\alpha_{1,3}} \\
q^p_{1,1} \\
\vdots \\
q^p_{1,M-2} \\
q^p_{1,M-1} + \frac{r^n_{M-1}(x_M-x_{M-2})g_4}{\alpha_{1,4}} \\
\hdashline
\vdots \\
\hdashline
q^p_{N-2,0} - \frac{r^n_0(x_1-x_{-1})g_3}{\alpha_{1,3}} \\
q^p_{N-2,1} \\
\vdots \\
q^p_{N-2,M-2} \\
q^p_{N-2,M-1} + \frac{r^n_{M-1}(x_M-x_{M-2})g_4}{\alpha_{1,4}} \\
\hdashline
q^p_{N-1,0} + \frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}} - \frac{r^n_0(x_1-x_{-1})g_3}{\alpha_{1,3}} \\
q^p_{N-1,1} + \frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}} \\
\vdots \\
q^p_{N-1,M-2} + \frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}} \\
q^p_{N-1,M-1} + \frac{s^n_{N-1}(z_N-z_{N-2})g_2}{\alpha_{1,2}} + \frac{r^n_{M-1}(x_M-x_{M-2})g_4}{\alpha_{1,4}}
\end{array}\right],$$

$$A = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
c_1 & d_1 & & & & e_1 & & & & & & & & & & & & & & & & & & & \\
b_1 & c_2 & d_2 & & & & e_2 & & & & & & & & & & & & & & & & & & \\
& \ddots & \ddots & \ddots & & & & \ddots & & & & & & & & & & & & & & & & & \\
& & b_1 & c_2 & d_2 & & & & e_2 & & & & & & & & & & & & & & & & \\
& & & b_2 & c_3 & & & & & e_3 & & & & & & & & & & & & & & & \\
\hline
a_1 & & & & & c_4 & d_3 & & & & e_4 & & & & & & & & & & & & & & \\
& a_2 & & & & b_3 & c_5 & d_4 & & & & e_5 & & & & & & & & & & & & & \\
& & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & & & & & & & & & \\
& & & a_2 & & & & b_3 & c_5 & d_4 & & & & e_5 & & & & & & & & & & & \\
& & & & a_3 & & & & b_4 & c_6 & & & & & e_6 & & & & & & & & & & \\
\hline
& & & & & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & & & & \\
\hline
& & & & & & & & & & a_1 & & & & & c_4 & d_3 & & & & e_4 & & & & \\
& & & & & & & & & & & a_2 & & & & b_3 & c_5 & d_4 & & & & e_5 & & & \\
& & & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & \\
& & & & & & & & & & & & & a_2 & & & & b_3 & c_5 & d_4 & & & & e_5 & \\
& & & & & & & & & & & & & & a_3 & & & & b_4 & c_6 & & & & & e_6 \\
\hline
& & & & & & & & & & & & & & & a_4 & & & & & c_7 & d_5 & & & \\
& & & & & & & & & & & & & & & & a_5 & & & & b_5 & c_8 & d_6 & & \\
& & & & & & & & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots & \\
& & & & & & & & & & & & & & & & & & a_5 & & & & b_5 & c_8 & d_6 \\
& & & & & & & & & & & & & & & & & & & a_6 & & & & b_6 & c_9
\end{array}\right]$$

$$B = \left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
h_1 & i_1 & & & & j_1 & & & & & & & & & & & & & & & & & & & \\
g_1 & h_2 & i_2 & & & & j_2 & & & & & & & & & & & & & & & & & & \\
& \ddots & \ddots & \ddots & & & & \ddots & & & & & & & & & & & & & & & & & \\
& & g_1 & h_2 & i_2 & & & & j_2 & & & & & & & & & & & & & & & & \\
& & & g_2 & h_3 & & & & & j_3 & & & & & & & & & & & & & & & \\
\hline
f_1 & & & & & h_4 & i_3 & & & & j_4 & & & & & & & & & & & & & & \\
& f_2 & & & & g_3 & h_5 & i_4 & & & & j_5 & & & & & & & & & & & & & \\
& & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & & & & & & & & & \\
& & & f_2 & & & & g_3 & h_5 & i_4 & & & & j_5 & & & & & & & & & & & \\
& & & & f_3 & & & & g_4 & h_6 & & & & & j_6 & & & & & & & & & & \\
\hline
& & & & & & & & & & & & & & & & & & & & & & & & \\
& & & & & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & & & & & & \\
& & & & & & & & & & & & & & & & & & & & & & & & \\
\hline
& & & & & & & & & & f_1 & & & & & h_4 & i_3 & & & & j_4 & & & & \\
& & & & & & & & & & & f_2 & & & & g_3 & h_5 & i_4 & & & & j_5 & & & \\
& & & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots & & & & \ddots & & \\
& & & & & & & & & & & & & f_2 & & & & g_3 & h_5 & i_4 & & & & j_5 & \\
& & & & & & & & & & & & & & f_3 & & & & g_4 & h_6 & & & & & j_6 \\
\hline
& & & & & & & & & & & & & & & f_4 & & & & & h_7 & i_5 & & & \\
& & & & & & & & & & & & & & & & f_5 & & & & g_5 & h_8 & i_6 & & \\
& & & & & & & & & & & & & & & & & \ddots & & & & \ddots & \ddots & \ddots & \\
& & & & & & & & & & & & & & & & & & f_5 & & & & g_5 & h_8 & i_6 \\
& & & & & & & & & & & & & & & & & & & f_6 & & & & g_6 & h_9
\end{array}\right]$$

where

$$a_1 = a_2 = a_3 = -\theta s_j^n, \qquad f_1 = f_2 = f_3 = (1-\theta)s_j^n$$
$$a_4 = a_5 = a_6 = -2\theta s_{N-1}^n, \qquad f_4 = f_5 = f_6 = 2(1-\theta)s_{N-1}^n,$$

$$b_1 = b_3 = b_5 = -\phi r_k^n, \qquad g_1 = g_3 = g_5 = (1-\phi)r_k^n$$
$$b_2 = b_4 = b_6 = -2\phi r_{M-1}^n, \qquad g_2 = g_4 = g_6 = 2(1-\theta)r_{M-1}^n,$$

$$d_1 = d_3 = d_5 = -2\phi r_0^n, \qquad i_1 = i_3 = i_5 = 2(1-\theta) r_0^n,$$
$$d_2 = d_4 = d_6 = -\phi r_k^n, \qquad i_2 = i_4 = i_6 = (1-\phi) r_k^n,$$

$$e_1 = e_2 = e_3 = -2\theta s_0^n, \qquad j_1 = j_2 = j_3 = 2(1-\theta) s_0^n,$$
$$e_4 = e_5 = e_6 = -\theta s_j^n, \qquad j_4 = j_5 = j_6 = (1-\theta) s_j^n,$$

$$\begin{aligned}
c_1 &= 1 + 2\phi r_0^n + 2\theta s_0^n - \frac{\phi r_0^n (x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}} - \frac{\theta s_0^n (z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}}, \\
c_2 &= 1 + 2\phi r_k^n + 2\theta s_0^n - \frac{\theta s_0^n (z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} \\
c_3 &= 1 + 2\phi r_{M-1}^n + 2\theta s_0^n + \frac{\phi r_{M-1}^n (x_1 - x_{-1})\alpha_{2,4}}{\alpha_{1,4}} - \frac{\theta s_0^n (z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}} \\
c_4 &= 1 + 2\phi r_0^n + 2\theta s_j^n - \frac{\phi r_0^n (x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}}, \\
c_5 &= 1 + 2\phi r_k^n + 2\theta s_j^n, \\
c_6 &= 1 + 2\phi r_{M-1}^n + 2\theta s_j^n + \frac{\phi r_{M-1}^n (x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}}, \\
c_7 &= 1 + 2\phi r_0^n + 2\theta s_{N-1}^n - \frac{\phi r_0^n (x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}} + \frac{\theta s_{N-1}^n (z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}, \\
c_8 &= 1 + 2\phi r_k^n + 2\theta s_{N-1}^n + \frac{\theta s_{N-1}^n (z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}, \\
c_9 &= 1 + 2\phi r_{M-1}^n + 2\theta s_{N-1}^n + \frac{\phi r_{M-1}^n (x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}} + \frac{\theta s_{N-1}^n (z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}},
\end{aligned}$$

$$
\begin{aligned}
h_1 &= 1 + 2(1-\phi)r_0^n + 2(1-\theta)s_0^n + \frac{(1-\phi)r_0^n(x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}} + \frac{(1-\theta)s_0^n(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}},\\
h_2 &= 1 + 2(1-\phi)r_k^n + 2(1-\theta)s_0^n + \frac{(1-\theta)s_0^n(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}},\\
h_3 &= 1 + 2(1-\phi)r_{M-1}^n + 2(1-\theta)s_0^n - \frac{(1-\phi)r_{M-1}^n(x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}} + \frac{(1-\theta)s_0^n(z_1 - z_{-1})\alpha_{2,1}}{\alpha_{1,1}},\\
h_4 &= 1 + 2(1-\phi)r_0^n + 2(1-\theta)s_j^n + \frac{(1-\phi)r_0^n(x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}},\\
h_5 &= 1 + 2(1-\phi)r_k^n + 2(1-\theta)s_j^n,\\
h_6 &= 1 + 2(1-\phi)r_{M-1}^n + 2(1-\theta)s_j^n - \frac{(1-\phi)r_{M-1}^n(x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}},\\
h_7 &= 1 + 2(1-\phi)r_0^n + 2(1-\theta)s_{N-1}^n + \frac{(1-\phi)r_0^n(x_1 - x_{-1})\alpha_{2,3}}{\alpha_{1,3}} - \frac{(1-\theta)s_{N-1}^n(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}},\\
h_8 &= 1 + 2(1-\phi)r_k^n + 2(1-\theta)s_{N-1}^n - \frac{(1-\theta)s_{N-1}^n(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}},\\
h_9 &= 1 + 2(1-\phi)r_{M-1}^n + 2(1-\theta)s_{N-1}^n - \frac{(1-\phi)r_{M-1}^n(x_M - x_{M-2})\alpha_{2,4}}{\alpha_{1,4}} - \frac{(1-\theta)s_{N-1}^n(z_N - z_{N-2})\alpha_{2,2}}{\alpha_{1,2}}.
\end{aligned}
$$

Appendix C

Coefficients of the Phenomenological Sintering Law

Here, we give values for the coefficients of Equation 4.5, reproduced from [82]. These are functions of the dihedral angle ϕ, which is shown in Figure 4.1, measured in radians.

$$
\begin{aligned}
A_0 &= 0.014573 + 0.0063822\phi + 0.0009983\phi^2 \\
A_1 &= -0.092348 - 0.028098\phi + 0.016495\phi^2 \\
A_2 &= 0.16242 - 0.0062352\phi - 0.022826\phi^2 \\
A_3 &= 0.5998 + 0.00533\phi \\
A_4 &= -1.271 + 0.4144\phi \\
A_5 &= \frac{-48\ln\left(\cos\frac{\phi}{2}\right) - 12 + 6\cos\phi + 14\cos^2\phi - 9\cos^3\phi + \cos^5\phi}{9(2+\cos\phi)^2(1-\cos\phi)^4} \\
A_6 &= \frac{3+\cos\phi}{18(2+\cos\phi)(1+\cos\phi)} \\
A_7 &= \frac{2+\cos\phi}{144(1+\cos\phi)^2} \\
A_8 &= \frac{3\sin^2\phi}{2C_3^{2/3}} \\
A_9 &= A_0 + 0.32A_1 + 0.1024A_2 \\
A_{10} &= \frac{A_1 + 0.64A_2}{A_9}
\end{aligned}
$$

$$C_0 = -4.069 + 6.557\phi + 0.0253\phi^2$$
$$C_1 = 26.75 - 42.58\phi + 5.986\phi^2$$
$$C_2 = -51.01 + 82.12\phi - 18.56\phi^2$$
$$C_3 = \frac{3}{2}\left(2 - 3\cos\phi + \cos^3\phi\right)$$
$$C_4 = 3\left(\phi - \frac{\pi}{6}\right) - 2\sqrt{3}\cos\phi\sin\left(\phi - \frac{\pi}{6}\right)$$
$$C_5 = C_0 + 0.32C_1 + 0.1024C_2$$
$$C_6 = \frac{C_1 + 0.64C_2}{C_5}$$
$$D_1 = \frac{1}{C_3^{1/3}}$$
$$D_2 = 0.7\frac{2C_4 + \left(\phi - \frac{\pi}{6}\right)\left(4\cos^2\phi - 3\right)}{C_4^{3/2}}$$
$$D_3 = 2\sqrt{2}\frac{\sin\left(\phi - \frac{\pi}{6}\right)}{C_4^{1/2}}$$

Appendix D

Computer Implementations in python and MATLAB of the Solvers for the One- and Two-Dimensional Wave and Helmholtz Equations

D.1 python Implementation of the Transient Finite Difference Method for the One-Dimensional Wave Equation

```
from pylab import * # so we know what pi is, etc
import scipy.sparse as sp # for using sparse matrix tools

def finite_diff_implicit(E_old,E_older,x,mu,sig,eps1,h,dt,tsim,starttime,bc):
       """finite_diff_implicit(E_old,E_older,x,mu,sig,eps1,h_sq,dt,tsim,starttime,bc):
           Implicit finite difference solver for iteratively solving the electromagnetic
           wave equation over a period of time. Requires solving linear system at each
           timestep--use with caution! This solution method allows the user to choose a
           longer timestep than the alternative explicit method below, but may take longer
            time.

       Inputs:
              E_old  The electric field at (n-1)st time step. A vector (array) of length N
                  .
              E_older  The electric field at (n-2)nd time step. A vector (array) of length
                   N.
              x  The x-coordinate values [m]. A vector (array) of length N.
              mu  The magnetic permeability at the x-values. A vector (array) of length N.
              sig  The electric conductivity at the x-values. A vector (array) of length N
                  .
              eps1  The electric permittivity at the x-values. A vector (array) of length
                  N.
              h  The differences between x-values [m]. A vector (array) of length N-1.
              dt  The length of the electromagnetic timestep [sec]. A scalar.
              tsim  The length of the simulation [sec]. A scalar.
              starttime The start time of the simulation [sec], used only for printing the
                  title of the graph. A scalar.
              bc  The type of boundary condition to use at right-hand endpoint. A string
                  that takes either of the two values 'pec' (perfect electric conductor) or
                   'abs' (absorbing).

       Outputs:
              E_new  The new electric field---that is, the field at the nth timestep
              E_old  The old electric field---that is, the field at the (n-s)st timestep
              eavg  The total power dissipated at each point of the system over the course
                  of processing. A vector (array) of length N.

       """

       mu0=pi*4e-7 # permeability of free space [N/A^2]
       c=299792458 # speed of light [m/s]

       h_sq = r_[h[0]**2 , h[1:]*h[:-1] , h[-1]**2] # represents h_left * h_right for each
            gridpoint (except that the left- and right-hand endpoints are just h_right
           and h_left, respectively, squared)

       if bc == 'abs': # implement absorbing (Neumann) boundary condition at right-hand
           endpoint by changing last row of A and last entry of RHS scaling vector of
           first older solution
              q_n = sqrt(h_sq[-1])/(c*dt)
              A_nn = 1+q_n
              A_nm = -1
       elif bc == 'pec': # implement perfect electric conductor (homogeneous Dirichlet)
           condition at right-hand endpoint by changing last row of A
              A_nn = 1
              A_nm = 0
              q_n = 0
       else: # throw an error if bc is neither of those strings
```

```
                print "The input variable 'bc' must be either the string 'abs' or the string 'pec'"
                input()
                import sys
                sys.exit(1)

        r = 2/(h_sq) + mu*eps1/((c*dt)**2) + mu*mu0*sig*0.5/dt # entries on the main diagonal of A
        s = mu*mu0*sig*0.5/dt - mu*eps1/((c*dt)**2) # multiplier for second older solution
        s = r_[1,s[1:-1],0] # accounting for boundary conditions
        q = 2*mu*eps1/((c*dt)**2) # multiplier for first older solution
        q = r_[0,q[1:-1],q_n] # accounting for boundary conditions

# A = diag(r_[1,r,1]) + diag(r_[1/(x[1]-x[0])**2 , 1/(h[0:-1]*h[0:-1]) , 0],-1) + diag(r_[0 , 1/(h*h)],1) # the non-sparse version (for testing speed-up)
        diagonals = [r_[1,r[1:-1],A_nn] , r_[1/h_sq[1:-1] , A_nm] , r_[0 , 1/h_sq[1:-1]] ]
        A = sp.diags(diagonals,[0,-1,1]).toarray() # make A directly as a sparse matrix

        time=0
        eavg = 0
        eavg_old = 0
        while time < tsim:
                E_new = linalg.solve(A,np.multiply(s,E_older)+np.multiply(q,E_old))
                time = time + dt
                E_older = E_old
                E_old = E_new

                eavg_new = 0.5*abs(E_new*E_new)
                eavg = eavg + 0.5*dt*(eavg_new+eavg_old)
                eavg_old = eavg_new

                # uncomment these lines if we want a plot at each timestep of EM solve (this could get expensive!)
                # Plot e-field average in whole cavity
                #plt.ion()
                #plt.figure(4)
                #plt.clf()
                #plt.plot(100*x,E_new)
                #plt.xlabel('Position along domain [cm]')
                #plt.title('E-field Average [V/m]')

```

```
    return  E_new,E_old,eavg

def finite_diff_explicit(E_old,E_older,x,mu,sig,eps1,h_sq,dt,tsim,starttime,bc):
    """finite_diff_explicit(E_old,E_older,x,mu,sig,eps1,h_sq,dt,tsim,starttime,bc):

    Explicit finite difference solver for the one-dimensional electromagnetic wave
        equation.

    Inputs:
        E_old  The electric field at (n-1)st time step. A vector (array) of length N
            .
        E_older  The electric field at (n-2)nd time step. A vector (array) of length
             N.
        x  The x-coordinate values [m]. A vector (array) of length N.
        mu  The magnetic permeability at the x-values. A vector (array) of length N.
        sig  The electric conductivity at the x-values. A vector (array) of length N
            .
        eps1  The electric permittivity at the x-values. A vector (array) of length
            N.
        h_sq  The squares of the differences between x-values [m]. A vector (array)
            of length N-1.
        dt  The length of the electromagnetic timestep [sec]. A scalar.
        tsim  The length of the simulation [sec]. A scalar. In the coupled routine,
            this should be equal to one timestep of the HEAT equation.
        starttime The start time of the simulation [sec], used exclusively for
            printing the title of the graph. A scalar.
        bc  The type of boundary condition to use at right-hand endpoint. A string
            that takes either of the two values 'pec' (perfect electric conductor) or
             'abs' (absorbing).

    Outputs:
        eavg  The total power dissipated into each point of the domain over the
            course of processing. A vector (array) of length N.



    """

    mu0=pi*4e-7 # permeability of free space [N/A^2]
    c=299792458 # speed of light [m/s]

    a = mu*mu0*sig/(2*dt) + mu*eps1/((c*dt)**2) # for quickly setting r, s, and A
```

```
        r = -2/(h_sq) + 2*mu*eps1/((c*dt)**2)
        s = mu*mu0*sig*0.5/dt - mu*eps1/((c*dt)**2)
        s = r_[0,s[1:-1],0] # multiplier for older e-field vector

        # Implement right-hand boundary condition by setting values of A[n,n] A[n,n-1], and
             possibly s[n]
        if bc == 'abs': # absorbing (inhomogeneous Neumann) boundary condition
              A_nn = a[-1]*(1-c*dt/sqrt(h_sq[-1]))
              A_nm = a[-2]*c*dt/sqrt(h_sq[-1])
        elif bc == 'pec': # perfect electric conductor (homogeneous Dirichlet) condition
              A_nn = 0
              A_nm = 0
        else: # throw an error if bc is neither of those strings
              print "The input variable 'bc' must be either the string 'abs' or the string
                  'pec'"
              input()
              import sys
              sys.exit(1)

        # Create A-matrix
        maindiag = r_[a[0],r[1:-1],A_nn]
        lowerdiag = r_[1/h_sq[1:-1],A_nm]
        upperdiag = r_[0,1/h_sq[1:-1]]

        diagonals = [ maindiag , lowerdiag , upperdiag ]
        A = sp.diags(diagonals, [0,-1,1]).toarray() # build A as a sparse matrix

        # Initialize and begin solution routine
        time=0
        eavg = 0
        eavg_old = 0
        while time < tsim:
              E_new = (1/a)*(multiply(s,E_older)+dot(A,E_old)) # compute the new field
              time = time + dt # update time
              E_older = E_old # update older
              E_old = E_new # update old

              eavg_new = 0.5*abs(E_new*E_new) # compute new dissipated power as root mean
                  square of electric field
              eavg = eavg + 0.5*dt*(eavg_new+eavg_old) # numerical integration (trapezoid
                  rule) of dissipated power
```

```
            eavg_old = eavg_new # update eavg_old term

        return  E_new,E_old,eavg
```

D.2 MATLAB Implementation of the Transient Finite Difference Method for the One-Dimensional Wave Equation

```
function [E_old,E_older]=emsolve1_fd(E_old,E_older,x,mu,sig,eps1,h,dt,tsim,starttime)
mu0=pi*4e-7; %[N/A^2] permeability of free space
c=299792458; % [m/s] speed of light

h=h(2:end);
eps1=eps1(2:end-1); mu=mu(2:end-1); sig=sig(2:end-1);

r=(-4*c*c*dt*dt-2*mu.*eps1.*h.*h-c*c*dt*mu0.*mu.*sig.*h.*h)./(2*c*c*dt*dt*h.*h);
s=(2*mu.*eps1-c*c*dt*mu0*mu.*sig)./(2*c*c*dt*dt); s=[1;s';1];
q=2*mu.*eps1./(c*c*dt*dt); q=[0;q';0];

A=diag([1,r,1])+...
   diag([1/(x(2)-x(1))^2 , 1./(h(1:end-1).*h(1:end-1)) , 0],-1)+...
   diag([0 , 1./(h.*h)],1);
A=sparse(A);
time=0;
while time < tsim
% Plot current field distribution
% figure(1);
% plot(100*x,E_old);
% title(strcat('Field distribution at t=',num2str(starttime+time,'%11.3g'),' seconds'));
% xlabel('Length [cm]'); ylabel('Electric field intensity [V/m]');

  E_new=A\(s.*E_older-q.*E_old);
  time=time+dt;
  E_older = E_old;
  E_old = E_new;
end

end
```

D.3 MATLAB Implementation of the Transient Finite Element Method for the One-Dimensional Wave Equation

```
%function emag1D()
% function emag1D()
%
% Performs transient FEM analysis of the electric field for a
% one-dimensional domain with a constant power source at the left-hand
% side. See problem description in PDF file of same directory.
% Uses a constant time step and uniform node spacing.
%
% Outputs: Saves figure at the end of process for embedding in PDF writeup.

% Physical setup
L=0.248; %length of domain [m]
P=1; % [W] power supplied by magnetron at left-hand endpoint
omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves
beta=pi/L; % [1/m] propagation constant

% Nodes and spacing
n=50; % number of (uniformly spaced) spatial nodes
x=linspace(0,L,n); %vector of x-values
h=x(2:end)-x(1:n-1); %h-values (as spacing is uniform, h is a multiple of ones vector)

% Time scenario
dt=0.1; % length of time step [sec]
time=0; % starting time [sec]
tsim=60; % time for which to perform the simulation [sec]

% Physical constants
mu0=pi*4e-7; %[N/A^2] permeability of free space
c=299792458; % [m/s] speed of light

% Load materials
mu_wat=1; % (unitless) relative permeability of water
sigma_wat=0.055; % [S/m] electrical conductivity of water
sigma_wat=345.66; %turn water into beef!
eps1_wat=75; % (unitless) relative permittivity of water
eps1_wat=33.6; % turn water into beef!
mu_air=1; % (unitless) relative permeability of air
sigma_air=0; % [S/m] electrical conductivity of air
eps1_air=1; % (unitless) relative permittivity of air

% Simulation values
```

```
theta=0.5; % weighting value--for the current time step, let T(t) be the
        %weighted combination T(t) = theta*T(t_{k+1}) + (1-theta)*T(t_k)

% Elemental values of physical properties
lim1=floor((n-1)/3); lim2=ceil(2*(n-1)/3); % limits for L/3 and 2L/3
mu=[mu_air*ones(lim1,1); mu_wat*ones(lim2-lim1,1); mu_air*ones((n-1)-lim2,1)]';
sigma=[sigma_air*ones(lim1,1); sigma_wat*ones(lim2-lim1,1); sigma_air*ones((n-1)-lim2,1)
    ]';
eps1=[eps1_air*ones(lim1,1); eps1_wat*ones(lim2-lim1,1); eps1_air*ones((n-1)-lim2,1)]';

% Set up left-hand side and right-hand side matrices, boundary condition, and forcing
    vector
parterm=((mu0.*sigma)./(2*dt)+eps1./((c*dt)^2)); % a helpful term
a2n=-theta/(mu(n-1)*h(n-1))-1/(2*c*mu(n-1)*dt); %a^2_n
A1=-theta.*(1./(h(1:n-1).*mu(1:n-1)))+parterm(1:n-1).*(h(1:n-1)./3); % subdiagonal
A2=[1, -theta.*(1./(mu(1:n-2).*h(1:n-2))+1./(mu(2:n-1).*h(2:n-1))) + ...
   parterm(1:n-2).*(h(1:n-2)./3)+parterm(2:n-1).*(h(2:n-1)./3), a2n]; % diagonal
A3=[0, theta.*(1./(h(2:n-1).*mu(2:n-1)))+parterm(2:n-1).*(h(2:n-1)./3)]; %superdiagonal
A=diag(A1,-1)+diag(A2,0)+diag(A3,1); %LHS

term=(2.*eps1.*h)./(3*(c*dt)^2); % a helpful term
b2n=-(theta-1)/(mu(n-1)*h(n-1))+term(n-1); %b^2_n
B1=(theta-1)./(h(1:n-1).*mu(1:n-1))+term(1:n-1); %subdiagonal
B2=[0, (1-theta).*(1./(mu(1:n-2).*h(1:n-2))+1./(mu(2:n-1).*h(2:n-1)))+ ...
   term(1:n-2)+term(2:n-1), b2n]; %diagonal
B3=[0, (theta-1)./(mu(2:n-1).*h(2:n-1))+term(2:n-1)]; % superdiagonal
B=diag(B1,-1)+diag(B2,0)+diag(B3,1); %RHS1

c2n=parterm(n-1)*h(n-1)/3-1/(2*c*mu(n-1)*dt);
C1=parterm(1:n-1).*h(1:n-1)./3; % subdiagonal
C2=[0, parterm(1:n-2).*h(1:n-2).*3+parterm(2:n-1).*h(2:n-1).*3 , c2n]; % diagonal
C3=[0, parterm(2:n-1).*h(2:n-1).*3]; % superdiagonal
C=diag(C1,-1)+diag(C2,0)+diag(C3,1); %RHS2

bc=[(2/L)*sqrt(2*P*(omega*mu0/beta)); zeros(n-1,1)]; %boundary condition vector

% Set up and perform the transient analysis
Tkm=zeros(n,1); %initial field distribution T(t_1)--will represent T(t_{k-1}) in the loop
Tk=Tkm; % field distribution at first time step T(t_2)--will represent T(t_k) in the loop
time=time+dt; % we start by plotting T(t_2), so time is already at second increment
while time<tsim
```

```
    %Plot current field distribution
    figure(1); clf; hold on
    plot(100*x,Tk);
    title(strcat('Field distribution at t=',num2str(time,'%11.3g'),' seconds'));
    xlabel('Length [cm]'); ylabel('Electric field intensity [V/m]');

    %Update time = t_{k+1}
    time=time+dt;

    %Solve for next field distribution
    rhs=B*Tk+C*Tkm+bc; % sets up RHS
    Tkp=A\rhs; % calcaultes T(t_{k+1})

    %Update T(t_k), T(t_{k-1})
    Tkm=Tk; %T(t_{k-1}) <-- T(t_k)
    Tk=Tkp; % T(t_k) <-- T(t_{k+1})
end %while time<tsim

% Plot final field distribution
figure(1); clf; hold on
plot(x*100,Tk);
title(strcat('Field distribution at t=',num2str(time,'%11.3g'),' seconds'));
xlabel('Length [cm]'); ylabel('Electric field intensity [V/m ]');
saveas(1,'End_fig.jpg','jpg');

%end %function
```

D.4 python Implementation of the Transient Finite Difference Method for the Two-Dimensional Wave Equation

```
import collections
from pylab import *
import matplotlib.pyplot as plt
import scipy.sparse as sp

def finite_diff(E_old,E_older,hx,hz,dt,tsim,theta,phi,L,Z,X,sig,eps,mu):
      """finite_diff(E_old,E_older,hx,hz,dt,tsim,p,vp,theta,phi,E_inc,L): Finite
          difference solver for iteratively solving the nondimensionalized
          electromagnetic wave equation over a period of time. Requires solving linear
          system at each timestep. Method is a theta-method where if theta=phi=1, method
          is fully implicit, if theta=phi=0 then method is fully explicit, and if theta=
        phi=0.5, then method is C-N-like. Uses trapezoidal rule to approximate
        integral for eavg.

    Inputs:
            E_old  The electric field at nth time step. A vector (array) of length (N+1)
                *(M+1).
            E_older  The electric field at (n-1)st time step. A vector (array) of length
                 (N+1)*(M+1).
            hx  The differences between unitless x-values. A vector (array) of length M
                +1.
            hz  The differences between unitless z-values. A vector (array) of length N
                +1.
            dt  The length of the electromagnetic timestep. A unitless scalar.
            tsim  The length of the simulation. A unitless scalar.
            theta  The parameter for the theta-method in the z-direction. A scalar that
                varies from 0 to 1. Take theta = 0 for a fully explicit method, theta = 1
                 for a fully implicit method, and theta = 0.5 for a Crank-Nicolson
                method.
            phi  The parameter for the theta-method in the x-direction. A scalar that
                varies from 0 to 1. Take phi = 0 for a fully explicit method, phi = 1 for
                 a fully implicit method, and phi = 0.5 for a Crank-Nicolson method.
            E_inc  The value of the incident field at the port [V/m]. Used for computing
                eavg. A scalar.
            L  The length of the domain in the z-direction [m]. Used for computing eavg.
                 A scalar.

    Outputs:
            E_new  The nondimensional e-field values at (n+1)st timestep. A vector (
                array) of length (N+1)*(M+1).
            E_old  The nondimensional e-field values at (n+1)st timestep. A vector (
                array) of length (N+1)*(M+1).
            eavg  The average dissipated power [V/m] over the time interval of the
                simulation. A vector (array) of length (N+1)*(M+1).

    """

    # Useful parameters
    c = 299792458.0 # speed of light [m/s]
    hx_sq = hx[1:]*hx[:-1]
    hz_sq = hz[1:]*hz[:-1]
    s = dt**2/(hz_sq)
    r = dt**2/(hx_sq)
    N = np.size(s)+1
    M = np.size(r)+1

    A = np.zeros(((N+1)*(M+1),(N+1)*(M+1)))
    B = np.zeros(((N+1)*(M+1),(N+1)*(M+1)))
    v = np.array([0]*(N+1)*(M+1))
    v[-1-M:]=-1.0/(c*dt*dt)

    for k in range(0,M+1):
        A[k,k]=1
        A[(N)*(M+1)+k,(N)*(M+1)+k]=1
        B[k,k]=1

    for j in range(1,N):
        A[j*(M+1),j*(M+1)]=1
        for k in range(1,M):
            #print "j="+str(j)+", k="+str(k)
            A[j*(M+1)+k,j*(M+1)+k]=2*theta*s[j-1]+2*phi*r[k-1]+mu[j,k]*eps[j,k
                ]+0.5*mu[j,k]*sig[j,k]*dt
            B[j*(M+1)+k,j*(M+1)+k]=-2*(1-theta)*s[j-1]-2*(1-phi)*r[k-1]+2*mu[j,
                k]*eps[j,k]

            A[j*(M+1)+k,(j-1)*(M+1)+k]=-theta*s[j-1]
            B[j*(M+1)+k,(j-1)*(M+1)+k]=(1-theta)*s[j-1]

            A[j*(M+1)+k,(j+1)*(M+1)+k]=-theta*s[j-1]
            B[j*(M+1)+k,(j+1)*(M+1)+k]=(1-theta)*s[j-1]

            A[j*(M+1)+k,j*(M+1)+k-1]=-phi*r[k-1]
            B[j*(M+1)+k,j*(M+1)+k-1]=(1-phi)*r[k-1]

            A[j*(M+1)+k,j*(M+1)+k+1]=-phi*r[k-1]
            B[j*(M+1)+k,j*(M+1)+k+1]=(1-phi)*r[k-1]

            v[j*(M+1)+k] = -1.0*mu[j,k]*eps[j,k]+0.5*mu[j,k]*sig[j,k]*dt

        A[j*(M+1)+M,j*(M+1)+M]=1

    time=0
    eavg = 0
```

```python
    eavg_old = 0
    while time < tsim:
        # Solve equation A*E_new = B*E_old + v*E_older

        E_new = linalg.solve(A,np.dot(B,E_old)+v*E_older)
        time = time + dt
        E_older = E_old
        E_old = E_new

        # Numerical integration (using trapezoidal rule) to compute eavg over solution interval
        eavg_new = 0.5*abs(E_new*E_new)
        eavg = eavg + 0.5*dt*(eavg_new+eavg_old)
        eavg_old = eavg_new

        # uncomment these lines if we want a plot at each timestep of EM solve (this could get expensive!)
        # Plot e-field average in whole cavity
        #plt.ion()
        #plt.figure(4)
        #plt.clf()
        #plt.contourf(100*Z,100*X,np.flipud(np.transpose(np.reshape(E_new,(N+1,M+1)))),100)
        #plt.colorbar()
        #plt.xlabel('Position along domain [cm]')
        #plt.ylabel('Position along domain [cm]')
        #plt.title('E-field Average [V/m], '+str(time)+' sec')
        #plt.draw()
    return  E_new,E_old,eavg


def helmsolve(hx,hz,K,E_inc):
    mu0=pi*4e-7 # permeability of free space [N/A^2]

    hx_sq = 1/(hx[1:]*hx[:-1])
    hz_sq = 1/(hz[1:]*hz[:-1])

    M = np.size(hx_sq)+2
    N = np.size(hz_sq)+2

    A = np.zeros((N*M,N*M))
```

```

        #print "number of xnodes is "+str(M)
        #print "number of znodes is "+str(N)

        #print "size of A is"
        #print np.shape(A)

        for k in range (0,M):
              A[k,k] = 1.0
              A[(N-1)*M+k,(N-1)*M+k] = 1.0
              #print "Changed the values of A["+str(k)+","+str(k)+"] and A["+str((N-1)*M+k
                  )+","+str((N-1)*M+k)+"]"

        for j in range (1,N-1):
              #print "j="+str(j)
              A[j*M,j*M]=1.0
              A[(j+1)*M-1,(j+1)*M-1]=1.0
              #print "Changed the values of A["+str(j*M)+","+str(j*M)+"] and A["+str((j+1)*
                  M-1)+","+str((j+1)*M-1)+"]"
              for k in range (1,M-1):
                    #print "k="+str(k)
                    A[j*M+k,j*M+k]=K[k-1,j-1]-2*hx_sq[k-1]-2*hz_sq[j-1]
                    A[j*M+k,(j-1)*M+k] = hz_sq[j-1]
                    A[j*M+k,(j+1)*M+k] = hz_sq[j-1]
                    A[j*M+k,j*M+k+1] = hx_sq[k-1]
                    A[j*M+k,j*M+k-1] = hx_sq[k-1]
                    #print "Changed the values of A on row "+str(j*M+k)

        b = np.array([0.0]*N*M)
        b[:M]=E_inc

        E = linalg.solve(A,b)

        return 0.5*E*E
```

D.5 MATLAB Implementation of the Transient Finite Difference Method for the Two-Dimensional Wave Equation

```
function [E_old,E_older]=emsolve2_fd(E_old,E_older,X,Y,Nx,Ny,mu,sigma,eps1,hx,hy,dt,tsim,
    starttime)
mu0=pi*4e-7; %[N/A^2] permeability of free space
```

```
c=299792458; % [m/s] speed of light

a_up=[zeros(1,Nx), repmat([0,(1./(hx(2:end).^2)),0],[1 Ny-2]), zeros(1,Nx-1)];
a_lo=[zeros(1,Nx-1), repmat([0,(1./(hx(1:end-1).^2)),0],[1 Ny-2]), zeros(1,Nx)];

b_up=[zeros(1,Nx), reshape([zeros(size(hy(2:end)));repmat((1./(hy(2:end)).^2),[Nx-2,1]);
    zeros(size(hy(2:end)))],1,[]) ];
b_lo=[reshape([zeros(size(hy(1:end-1)));repmat((1./(hy(1:end-1)).^2),[Nx-2,1]);zeros(size
    (hy(1:end-1)))],1,[]), zeros(1,Nx) ];

q_int=repmat([0,(-2./(hx(2:end)).^2),0],[1 Ny-2])...
   +reshape([zeros(size(hy(2:end)));repmat(-2./(hy(2:end)).^2,[Nx-2,1]);zeros(size(hy(2:
       end)))],1,[])...
   -(reshape(mu(2:end-1,:)',1,[])).*(reshape(eps1(2:end-1,:)',1,[]))/(c*c*dt*dt)...
   -(reshape(mu(2:end-1,:)',1,[])).*(reshape(sigma(2:end-1,:)',1,[]))*mu0/(2*dt);

q=[ones(1,Nx),q_int,ones(1,Nx)];
q((1:Ny-1)*Nx+1)=1; % electric field is fixed at pow on input port side
q((1:Ny)*Nx)=1; % electric field is fixed at 0 on output port side

A=diag(q,0) + diag(a_up,1) + diag(a_lo,-1) + diag(b_up,Nx) + diag(b_lo,-Nx);
A=sparse(A);

t=-2*(reshape(mu',1,[])).*(reshape(eps1',1,[]))./(c*c*dt*dt);
s=-0.5*t-(reshape(mu',1,[])).*(reshape(sigma',1,[]))*mu0/(2*dt);

s(1:Nx)=0; s(end-Nx:end)=0; % electric field fixed at 0 on top and bottom
s((1:Ny-1)*Nx+1)=0; % electric field is fixed at pow on input port side
s((1:Ny)*Nx)=0; % electric field is fixed at 0 on output port side
t(1:Nx)=1; t(end-Nx:end)=1; % electric field fixed at 0 on top and bottom
t((1:Ny-1)*Nx+1)=1; % electric field is fixed at pow on input port side
t((1:Ny)*Nx)=1; % electric field is fixed at 0 on output port side

time=0;
while time <= tsim
% Plot current field distribution
% figure(1); hold off;
% surf(100*X,flipud(100*Y),reshape(E_old,Nx,Ny)'); view(0,90); colorbar;
% title(strcat('Field distribution at t=',num2str(starttime+time,'%11.3g'),' seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Electric field
     intensity [V/m]');
```

```

  E_new=A\(s'.*E_older+t'.*E_old);

  time=time+dt;
  E_older = E_old;
  E_old = E_new;
end

end
```

D.6 MATLAB Implementation of the Transient Finite Element Method for the Two-Dimensional Wave Equation

```
function T=jin2D(nodefile,elfile,bc1file,bc2file)
% function T=jin2D(nodefile,elfile,bc1file,bc2file)
%
% Performs FEM analysis of the 2D electromagnetic scenario decribed in Jin,
% Chapter 4. Solves the differential equation:
%
% -d/dx(alpha_x * du/dx) - d/dy(alpha_y * du/dy) + beta*u = f,
%
% Where a_x, a_y, and b are known parameters associated with the physical
% properties of the domain, and f is a known source or excitation function.
% The standard two-dimensional Laplace, Poisson, and Helmholtz equations
% are special forms of this equation.
%
% This solver takes a list of elements and nodes for a given domain as
% inputs, and provides the solution on that domain.
%
% Inputs: nodefile - name of file containing node coordinates. Should be
% formatted as follows (without the header):
%
% (x-coord) (y-coord)
% 0 0
% . .
% . .
% . .
%
% elfile - name of file containing element definitions.
% Contains information about alpha_x, alpha_y, beta,
% and f at each element as well. Should be formatted
% as follows (without the header):
%
% (node1) (node2) (node3) (ax) (ay) (beta) (f)
% 1 2 5 1 3 1 0
% . . . . . . .
% . . . . . . .
% . . . . . . .
%
% bcfile1 — name of file containing the nodes on the
% Dirichlet boundary and their values. Should be
% formatted as follows:
%
% (node number) (value)
% 1 118
% . .
% . .
% . .
%
% bcfile2 — name of file containing the edges on the
% third—kind boundary. Should be formatted as follows:
%
% (node1) (node2) (g) (q)
% 1 2 5 1
% . . . .
% . . . .
% . . . .

% Read in elements, nodes, and boundary conditions to matrices

N=dlmread(nodefile); % nodes
   nn=length(N(:,1)); %number of nodes
E=dlmread(elfile); % elements
   el=length(E(:,1)); %number of elements
% E(:,5)=ones(size(E(:,5))); % alpha_x
% E(:,6)=E(:,5); % alpha_y
% E(:,7)=E(:,5); % beta
% E(:,8)=E(:,5); % f

% Assemble the LHS matrix and RHS vector

s=zeros(1,3); x=s; y=s; b=s; c=s; % Initialize the vectors that store local
```

```
% node numbers and coordinate values
K=zeros(nn,nn); % Initialize the LHS matrix
rhs=zeros(size(N(:,1)));
for k=1:el % for each element
  % Get the alpha_x, alpha_y, beta, and f values for the current element
  alpha_x=E(k,4); alpha_y=E(k,5); beta=E(k,6); f=E(k,7);

  % Get the node numbers and coordinates for the current element
  for m=1:3
    s(m)=E(k,m); % numbers of the nodes in the order they appear (should be ccw in the list
         )
    x(m)=N(s(m),1); % x−coordinates of nodes in the order they appear
    y(m)=N(s(m),2); % y−coordinates of nodes in the order they appear
  end % for m=1:3

  % Calculate the area of the element and all b and c coefficients (a not necessary)
  Ar=polyarea(x,y); % area of the current element
  b(1)=y(2)−y(3); b(2)=y(3)−y(1); b(3)=y(1)−y(2); % b−coefficients
  c(1)=−(x(2)−x(3)); c(2)=−(x(3)−x(1)); c(3)=−(x(1)−x(2)); % c−coefficients

  % Populate the LHS matrix
  for i=1:3
    for j=1:3
       if i==j, delta=1; else delta=0; end

       K(s(i),s(j))=K(s(i),s(j))+(1/(4*Ar))*(alpha_x*b(i)*b(j)+alpha_y*c(i)*c(j)) + ...
           (Ar/12)*beta*(1+delta);
    end % for j=1:3

    rhs(s(i))=rhs(s(i))+Ar*f/3;

  end % for i=1:3
end % for k=1:el

% Boundary conditions and solving

if ~isempty(bc2file)
    % Impose the Third−kind condition
    BC2=dlmread(bc2file); % Third−kind boundary
    s=zeros(1,2); % Initialize the vector that stores local node numbers
    for k=1:length(BC2(:,1)) % for each edge on the boundary
```

```
        s(1)=BC2(k,1); s(2)=BC2(k,2); % get universal node numbers
        g=BC2(k,3); q=BC2(k,4); % get gamma and q values
        l=sqrt((N(s(1),1)-N(s(2),1))^2+(N(s(1),2)-N(s(2),2))^2); % get length of segment
        for i=1:2
           for j=1:2
              if i==j, delta=1; else delta=0; end
              K(s(i),s(j))=K(s(i),s(j))+g*l*(1+delta)/6; % modify K at the two nodes
           end % for j=1:2
           rhs(s(i))=rhs(s(i))+q*l; % modify rhs
        end % for i=1:2
    end % for k=1:length(BC2(:,1))
end % if ~isempty(bcfile2)

if ~isempty(bc1file)
    % Impose the Dirichlet boundary condition
    BC1=dlmread(bc1file);
    for i=1:length(BC1(:,1))
        nn=BC1(i,2);
        rhs=rhs-K(:,nn)*BC1(i,3);
        rhs(j)=[];
        K(:,j)=[]; K(j,:)=[];
    end % for i=1:length(BC1(:,1))

    % Solve the system
    T_small=K\rhs;

    % Clean up the solution (re-insert the values on the Dirichlet boundary)
    flag=max(BC1(:,3))+10; % a flag that we know is not one of the boundary values
    T=flag*ones(size(N(:,1))); % temporarily set all T entries to the flag

    T(BC1(i,2))=BC1(i,3); % replace the flags with the values on the Dirichlet
    % boundary in those positions
    T(T==flag)=T_small; % use the remaining flags put the values calculated
    % in the matrix where they belong
else
    % No Dirichlet condition to impose, just solve the system
    T=K\rhs;
end % if ~isempty(bc1file)

end % function
```

MATLAB Test: Equi-H_z Fields for the Parallel Plate Waveguide with Dielectric Inclusion

```
function []=parallelplate()

lambda=0.10; % wavelgnth [m]
objx=0.05; % length of the domain (x-direction) [m]
hml=1; % how many wavelengths the artificial boundary is from the inclusion
H=(0:0.05:0.35)*lambda; % h-values to use in generating the graph on p. 110 of Jin

mgx=0:0.025:2*hml*lambda+0.05; % domain in x-direction
mgy=0:0.005:0.035; % domain in y-direction

%% eps2=4

eps2=4;

% For R,T plots
Re1=zeros(size(H));
Te1=zeros(size(H));
for j=1:length(H)
 h=H(j);
 [R,T,Hz,p,t]=findRT(h,eps2,lambda,objx,hml);
 Re1(j)=abs(R);
 Te1(j)=abs(T);
end

% For equi-Hz contour
h=0.0175;
[RHz1,THz1,Hz1,p1,t1]=findRT(h,eps2,lambda,objx,hml);

%% eps2=4-i

eps2=4-1j;

% For R,T plots
Re2=zeros(size(H));
Te2=zeros(size(H));
for j=1:length(H)
 h=H(j);
 [R,T,Hz,p,t]=findRT(h,eps2,lambda,objx,hml);
 Re2(j)=abs(R);
 Te2(j)=abs(T);
end

% For equi-Hz contour
h=0.0175;
[RHz2,THz2,Hz2,p2,t2]=findRT(h,eps2,lambda,objx,hml);

%% eps2=4-10i

eps2=4-1j*10;

% For R,T plots
Re3=zeros(size(H));
Te3=zeros(size(H));
for j=1:length(H)
 h=H(j);
 [R,T,Hz,p,t]=findRT(h,eps2,lambda,objx,hml);
 Re3(j)=abs(R);
 Te3(j)=abs(T);
end

% For equi-Hz contour
h=0.0175;
[RHz3,THz3,Hz3,p3,t3]=findRT(h,eps2,lambda,objx,hml);


%% Plotting equi-Hz contours

figure(1); clf; hold on;
subplot(6,1,1),
    F=TriScatteredInterp(p1(:,1),p1(:,2),real(Hz1));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
% mesh(xq,yq,vq); hold on; plot3(p1(:,1),p1(:,2),real(Hz1),'o'); hold off;
% trisurf(t1(:,1:3),p1(:,1),p1(:,2),real(Hz1),'facecolor','interp');
    view(2);
    set(gca,'plotboxaspectratio',[16 2 1]);
    title('Real Part, \epsilon=4.0-0i');
subplot(6,1,2),
% trisurf(t1(:,1:3),p1(:,1),p1(:,2),imag(Hz1),'facecolor','interp');
```

```
    F=TriScatteredInterp(p1(:,1),p1(:,2),imag(Hz1));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
    title('Imaginary Part, \epsilon=4.0-0i');
    set(gca,'plotboxaspectratio',[16 2 1]);
    view(2);
subplot(6,1,3),
    F=TriScatteredInterp(p2(:,1),p2(:,2),real(Hz2));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
% trisurf(t2(:,1:3),p2(:,1),p2(:,2),real(Hz2),'facecolor','interp');
    set(gca,'plotboxaspectratio',[16 2 1]);
    title('Real Part, \epsilon=4.0-1i');
    view(2);
subplot(6,1,4),
    F=TriScatteredInterp(p2(:,1),p2(:,2),imag(Hz2));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
% trisurf(t2(:,1:3),p2(:,1),p2(:,2),imag(Hz2),'facecolor','interp');
    set(gca,'plotboxaspectratio',[16 2 1]);
    title('Imaginary Part, \epsilon=4.0-1i');
    view(2);
subplot(6,1,5),
    F=TriScatteredInterp(p3(:,1),p3(:,2),real(Hz3));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
% trisurf(t3(:,1:3),p3(:,1),p3(:,2),real(Hz3),'facecolor','interp');
    set(gca,'plotboxaspectratio',[16 2 1]);
    title('Real Part, \epsilon=4.0-10i');
    view(2);
subplot(6,1,6),
    F=TriScatteredInterp(p3(:,1),p3(:,2),imag(Hz3));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
% trisurf(t3(:,1:3),p3(:,1),p3(:,2),imag(Hz3),'facecolor','interp');
    set(gca,'plotboxaspectratio',[16 2 1]);
    title('Imaginary Part, \epsilon=4.0-10i');
    view(2);

%% Plots for SIAM Cover Photo

figure(4); clf; hold on;
    F=TriScatteredInterp(p1(:,1),p1(:,2),real(Hz1));
    [xq,yq]=meshgrid(mgx,mgy);
    vq = F(xq,yq);
    contour(xq,yq,vq,15);
    mesh(xq,yq,vq); hold on; plot3(p1(:,1),p1(:,2),real(Hz1),'o'); hold off;
    trisurf(t1(:,1:3),p1(:,1),p1(:,2),real(Hz1),'facecolor','interp');
    set(gca,'plotboxaspectratio',[16 2 1]);
    title('Real Part, \epsilon=4.0-0i');


%% Plotting reflection and transmission
H=H/lambda;

figure(2); clf; hold on; grid on;
plot(H,Re1,'k-',H,Re2,'b--',H,Re3,'r-.');
xlabel('h/\lambda'); ylabel('|R|');
legend('\epsilon = 4 - 0i','\epsilon = 4 - 1i','\epsilon = 4 - 10i','Location','NorthWest
    ');

figure(3); clf; hold on; grid on;
plot(H,Te1,'k-',H,Te2,'b--',H,Te3,'r-.');
xlabel('h/\lambda'); ylabel('|T|');
legend('\epsilon = 4 - 0i','\epsilon = 4 - 1i','\epsilon = 4 - 10i','Location','SouthWest
    ');

end

function [R,T,Hz,p,t]=findRT(objy,eps2,lambda,objx,hml)
% Solves the problem described from Page 105 of Jin: solves for the
% electric and magnetic fields near a discontinuity in a parallel-plate
% waveguide.

%% Initializing -- problem setup

% Delete old data files
```

```

delete '*.dat'

% Physical constants

mu0=pi*4e-7; %[N/A^2] permeability of free space
eps0=8.854e-12; %[F/m] permittivity of free space
c=299792458; % [m/s] speed of light in a vacuum

%lambda=0.1; % [cm] wavelength
omega=2*pi*c/lambda; % [Hz] angular frequency of microwaves
k0=omega*sqrt(eps0*mu0);
H0=10; % magnitude of incidence field

% Dielectric properties and coefficient values
mu1=1; % (unitless) relative permeability of air
sigma1=0; % [S/m] electrical conductivity of air
eps11=1; % (unitless) relative dielectric constant of air. See Wikipedia
% article: http://en.wikipedia.org/wiki/Relative_permittivity
tan1=0; % (unitless) loss tangent of air. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps12=(tan1*(omega*eps11)-sigma1)/omega; % (unitless) relative loss factor of air
eps1=eps11+1j*eps12;

mu2=2-0.1*1j;

ax1=1/eps1; ay1=1/eps1; beta1=-k0^2*mu1; f=0;
ax2=1/eps2; ay2=1/eps2; beta2=-k0^2*mu2;

% Geometrical setup

%objx=0.05; % [cm] object length (x-dir)
domx=2*hml*lambda+objx;
domy=0.035;

% Filenames for node and element lists

nodefile='n.dat';
elfile='e.dat';
bc2file='bc2.dat';
```

```

%% Meshing and saving lists

% Meshing

node = [0,0;domx,0;domx,domy;0,domy]; % coordinates of corners of domain (to use in mesh2d)
hdata.fun = @hfun; % function specifying size of mesh (to be used in mesh2d)
hdata.args={lambda,objx,objy,hml}; % arguments for hfun in addition to x,y (to use in
    mesh2d)
%options.mlim=0.02; % The convergence tolerance. The maximum percentage change in edge
% length per iteration must be less than mlim
%options.maxit=20; % The maximum allowable number of iterations.
%options.dhmax=0.30; % The maximum allowable (relative) gradient in the size function
options.output=false; % suppresses output for mesh generation

[p,t] = mesh2d(node,[],hdata,options); % generates mesh of domain

% Element−varying properties

p(:,3)=0; % flag indicating node is outside dielectric rod
chgind=intersect(intersect(find(p(:,1)>=hml*lambda),...
     find(p(:,1)<=hml*lambda+objx)),find(p(:,2)<=objy)); % indices of nodes in the rod
p(chgind,3)=1; % flag indicating node is inside dielectric rod

eltsin=intersect(intersect(find(p(t(:,1),3)==1),find(p(t(:,2),3)==1)),...
     find(p(t(:,3),3)==1)); % indices of elements in the rod

t(:,4)=ax1; t(:,5)=ay1; t(:,6)=beta1; t(:,7)=f; % putting material properties of air in
    place
t(eltsin,4)=ax2; t(eltsin,5)=ay2; t(eltsin,6)=beta2; % put properties of dielectric in
    place

% Writing node and element lists
%p=[(1:1:length(p(:,1)))',p]; % putting node numbers in leftmost column
p=p(:,1:2);
dlmwrite(nodefile,p,'delimiter','\t','precision','%.6f'); % save nodes
dlmwrite(elfile,t,'delimiter','\t','precision','%.6f'); % save elements

% Third kind condition at left−hand boundary

ABg=1j*k0*ax1;
```

```
ABq=2*1j*k0*H0*ax1;

ABnodes=find(p(:,1)==0); % find nodes on the left-hand aboundary
for i=1:length(t(:,1))
   A=t(i,1:3);
   a1=any(ABnodes==A(1));
   a2=any(ABnodes==A(2));
   a3=any(ABnodes==A(3));

   anysum=a1+a2+a3;

   if anysum==2
       % Then an edge is on the boundary
      if a1==1
         if a2==1
            % Then nodes 1 and 2 are on the boundary
            writebc=[A(1),A(2),ABg,ABq];
         elseif a3==1
            % Then nodes 1 and 3 are on the boundary
            writebc=[A(1),A(3),ABg,ABq];
         end % if a2==1
      elseif a2==1
         % Then nodes 2 and 3 are on the boundary
         writebc=[A(2),A(3),ABg,ABq];
      end % if a1==1
      dlmwrite(bc2file,writebc,'-append','delimiter','\t','precision','%.6f')
   end % if anysum==2
end % for i=1:length(t(:,1))

% Third kind condition at right-hand boundary

CDg=1j*k0*ax1;
CDq=0;

CDnodes=find(p(:,1)==domx); % find nodes on the right-hand aboundary
for i=1:length(t(:,1))
   A=t(i,1:3);
   a1=any(CDnodes==A(1));
   a2=any(CDnodes==A(2));
   a3=any(CDnodes==A(3));

```

```
    anysum=a1+a2+a3;

    if anysum==2
        % Then an edge is on the boundary
       if a1==1
           if a2==1
               % Then nodes 1 and 2 are on the boundary
               writebc=[A(1),A(2),CDg,CDq];
           elseif a3==1
               % Then nodes 1 and 3 are on the boundary
               writebc=[A(1),A(3),CDg,CDq];
           end % if a2==1
       elseif a2==1
           % Then nodes 2 and 3 are on the boundary
           writebc=[A(2),A(3),CDg,CDq];
       end % if a1==1
       dlmwrite(bc2file,writebc,'-append','delimiter','\t','precision','%.6f')
    end % if anysum==2
end % for i=1:length(t(:,1))

%% FEM Solve

Hz = jin2D(nodefile,elfile,[],bc2file);


%% Plotting mesh and equi-Hz curves and calculating R, T

figure(4); clf; hold on;
trimesh(t(:,1:3),p(:,1),p(:,2));
title('Sample Finite Element Mesh');
set(gca,'plotboxaspectratio',[6 1 1]);

x1=intersect(find(p(:,1)==0),find(p(:,2)==0)); % Get number of the node at bottom left
x2=intersect(find(p(:,1)==domx),find(p(:,2)==0)); % Get number of the node at bottom right

R=(Hz(x1)-H0*exp(-1j*k0*0.1))/(H0*exp(1j*k0*0.1));
T=(Hz(x2))/(H0*exp(-1j*k0*domx));

fprintf('|R|^2+|T|^2=%g,\n',abs(R)^2+abs(T)^2);
```

```
end % parallelplate


%% Mesh refinement function
function h=hfun(x,y,lambda,objx,objy,hml)
% User defined size function

h=0.01*ones(size(x,1),1); % size 0.001 outside the dielectric rod

in=(x>=hml*lambda)&(x<=hml*lambda+objx)&(y<=objy); % size 0.0001 inside
h(in)=0.005;

end % hfun
```

D.7 MATLAB Implementation of Three Solvers for the One-Dimensional Helmholtz Equation

Finite Element Method for the One-Dimensional Helmholtz Equation

```
function []=helm1D()
% function helm1D()
%
% Performs FEM analysis of the electric field for a
% one−dimensional domain with a constant power source at the left−hand
% side. See problem description in PDF file of same directory.
% Uses uniform node spacing.
%

% Physical setup
L=0.248; %length of domain [m]
P=40e3; % [W] power supplied by magnetron at left−hand endpoint
omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves
beta=pi/L; % [1/m] propagation constant

% Nodes and spacing
n=50; % number of (uniformly spaced) spatial nodes
x=linspace(0,L,n); %vector of x−values
h=x(2:end)−x(1:n−1); %h−values (as spacing is uniform, h is a multiple of ones vector)

% Time scenario
dt=1; % length of time step [sec]
```

```
time=0; % starting time [sec]
tsim=60; % time for which to perform the simulation [sec]

% Physical constants
mu0=pi*4e-7; %[N/A^2] permeability of free space
c=299792458; % [m/s] speed of light in a vacuum

magnetron=(2/L)*sqrt(2*P*(omega*mu0/beta)); % value of field at RHS

% Load materials
mu_wat=1; % (unitless) relative permeability of water
sigma_wat=0.055; % [S/m] electrical conductivity of water
eps1_wat=78.54; % (unitless) relative dielectric constant of water. See:
% http://www.kayelaby.npl.co.uk/general_physics/2_6/2_6_5.html , 25C
tan_wat=0.16; %(unitless) loss tangent of water. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps2_wat=(tan_wat*(omega*eps1_wat)-sigma_wat)/omega; %(unitless) relative loss factor of
    water
mu_air=1; % (unitless) relative permeability of air
sigma_air=0; % [S/m] electrical conductivity of air
eps1_air=1; % (unitless) relative dielectric constant of air. See Wikipedia
% article: http://en.wikipedia.org/wiki/Relative_permittivity
tan_air=0; % (unitless) loss tangent of air. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps2_air=(tan_air*(omega*eps1_air)-sigma_air)/omega; % (unitless) relative loss factor of
    air

% Elemental values of physical properties
lim1=floor((n-1)/3); lim2=ceil(2*(n-1)/3); % limits for L/3 and 2L/3
mu=[mu_air*ones(lim1,1); mu_wat*ones(lim2-lim1,1); mu_air*ones((n-1)-lim2,1)]';
%sigma=[sigma_air*ones(lim1,1); sigma_wat*ones(lim2-lim1,1); sigma_air*ones((n-1)-lim2,1)
    ]';
eps1=[eps1_air*ones(lim1,1); eps1_wat*ones(lim2-lim1,1); eps1_air*ones((n-1)-lim2,1)]';
eps2=[eps2_air*ones(lim1,1); eps2_wat*ones(lim2-lim1,1); eps2_air*ones((n-1)-lim2,1)]';
eps=eps1+1i*eps2;

% Set up left-hand side and right-hand side matrices, boundary condition, and forcing
    vector
% subdiagonal
```

```
A1=[-1./(h(1:n-2).*mu(1:n-2))+((omega^2.*mu(1:n-2).*eps(1:n-2).*h(1:n-2))./(3*c^2)),0];
% diagonal
A2=[1, 1./(mu(1:n-2).*h(1:n-2))+1./(mu(2:n-1).*h(2:n-1)) + ...
   (omega^2.*h(1:n-2).*mu(1:n-2).*eps(1:n-2))./(3*c^2) + ...
   (omega^2.*h(2:n-1).*mu(2:n-1).*eps(2:n-1))./(3*c^2), 1];
% superdiagonal
A3=[0, -1./(h(2:n-1).*mu(2:n-1))+(omega^2.*mu(2:n-1).*eps(2:n-1).*h(2:n-1))./(3*c^2)];
A=diag(A1,-1)+diag(A2,0)+diag(A3,1); %LHS

bc=[magnetron; zeros(n-1,1)]; %boundary condition vector

T=A\bc;
S=real(T); W=imag(T);

% Display the field through time
%while time<tsim
   %Plot current field distribution
   figure(1); clf; hold on
   plot(100*x,S,'b',100*x,W,'r');%S*cos(omega*time));
   legend('Re(E)','Im(E)');
% plot(100*x, S*cos(omega*time)+W*sin(omega*time));
% title(strcat('Field distribution at t=',num2str(time,'%11.3g'),' seconds'));
   xlabel('Length [cm]'); ylabel('Electric field intensity [V/m] or [N/C]');
   title('Homecooked FEM method for Helmholtz');
   grid on

   %Update time = t_{k+1}
   time=time+dt;
%end %while time<tsim
saveas(1,'helm1D_fig.jpg','jpg');
end
```

bvp4c Solver for the One-Dimensional Helmholtz Equation

```
function [yvals]=onedembvp()
%ONEDEMBVP uses bvp4c to solve for the electromagnetic field over a
%one-dimensional object, and returns the square of the modulus of that
%field.
%
%The physical situation is a one-dimensional, nonhomogeneous rod of length
%L, with a source of electromagnetic energy at the left-hand boundary of
%the rod. The rod is comprised of two different materials, with the middle
%third of the domain comprised of water, and the outer two thirds comprised
%of air. When we assume the electromagnetic field
%(E(x,t)) is harmonic in time, we say that E(x,t) = S(x)cos(omega*t) +
%W(x)sin(omega*t), where omega is the angular frequency of the incident
%microwaves.
%
%We apply Maxwell's Equations to obtain a coupled system of second-order
%ordinary differential equations for S and W,
% S'' = a*S + b*W
% W'' = -b*S + a*W
%where a = eps'*mu*omega^2/c^2 and b=eps''*mu*omega^2/c^2,
%subject to the boundary conditions,
% S(0) = s0, W(0) = w0, S(L) = 0, W(L) = 0
%where at the right-hand boundary (L), the tangential component of the EM
%field, in our case the field itself, is zero because we assume that the
%boundary is conducting, and at the left-hand boundary (0), the magnitude
%of the field is equal to the magnitude of the field induced by the
%magnetron at that end.
%
% In order to solve this system, we convert to a first order system:
% y(1) = S
% y(2) = K where S' = K
% y(3) = W
% y(4) = M where W' = M
%
% The converted system is as follows:
% y(1)' = y(2)
% y(2)' = a*y(1) + b*y(3)
% y(3)' = y(4)
% y(4)' = -b*y(1) + a*y(3)
%
```

```
% And due to the nature of boundary value problems, where often more than
% one solution exists, bvp4c requires that we input a guess for the
% solution; in this case, we guess that the solutions for both S and W are
% constant values of s0 and w0 respectively.

% Physical setup
L=0.248; % length of domain [m]
P=40e3; % [W] power supplied by magnetron at left-hand endpoint
omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves
beta = pi/L; % [1/m] propagation constant

% Nodes and spacing
n=50;
x=linspace(0,L,n); % x-values for which bvp4c will solve the bvp

% Physical constants
c=299792458; %speed of light in a vacuum [m/s]
mu0=pi*4e-7; %permeability of free space [N/A^2]

Magnetron = (2/L)*sqrt(2*P*omega*mu0/beta); %initial E-field at left-hand
% boundary (magnetron)

gamma = 1;
s0 = gamma*Magnetron; %Left-hand boundary condition
w0 = (1-gamma)*Magnetron; %Right-hand boundary condition
guess = [s0; 0; w0; 0]; % guess constant solutions for S, W
solinit = bvpinit(linspace(0,L,n),guess); %use bvpinit for the initial guess
omegac = omega^2/c^2;

sol = bvp4c(@onedemode,@onedembc,solinit,[],[s0, w0], omega,omegac,L);

%xvals = linspace(0,L,n); %same as xvals = [0:dx:L] where dx = L/(Nx-1)
                       %these are the x-values for which we'll solve the
                       %equations for y1, y2, y3, and y4
yvals = deval(sol, x); %returns an Nx-by-4 matrix containing values of
                        %y1, y2, y3, and y4 (columns) at each of the
                        %points specified in xvals (rows)
%Plotting solution
 figure(1); hold on;
 plot(x*100, yvals(1,:),'c*-', x*100,yvals(3,:),'m*-')
 legend('Re(E)', 'Im(E)')
 xlabel('Length [cm]')
 ylabel('Electric field intensity [V/m] or [N/C]')
% %axis([0 1.001*L 0 0.001])
 grid on
%
 eavg = (yvals(1,:).^2 + yvals(3,:).^2); %the square of the modulus of the e-field
%
% figure(2);
% plot(xvals, eavg)
% ylabel('S^2 + W^2')
% xlabel(['Distance from Left-Hand Boundary of Object (L =', num2str(L), ')'])
% ylabel('|E| _{avg} ^2')
% title([ num2str(Material1Name), ' | ', num2str(Material2Name)])
% %axis([0 1.001*L 0 0.001])
% grid on
return

%
    ------------------------------------------------------------------------------

function dydx = onedemode(x,y,S0,omega,omegac,L) %#ok<*INUSL>
% ONEDEMODE Evaluate the function f(x,y)

% Load materials
mu_wat=1; % (unitless) relative permeability of water
sigma_wat=0.055; % [S/m] electrical conductivity of water
eps1_wat=78.54; % (unitless) relative dielectric constant of water. See:
% http://www.kayelaby.npl.co.uk/general_physics/2_6/2_6_5.html ( 25 C )
tan_wat=0.16; %(unitless) loss tangent of water. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm )
eps2_wat=(tan_wat*(omega*eps1_wat)-sigma_wat)/omega; % (unitless) relative
% loss factor of water
mu_air=1; % (unitless) relative permeability of air
sigma_air=0; % [S/m] electrical conductivity of air
eps1_air=1; % (unitless) relative dielectric constant of air. See Wikipedia
% article: http://en.wikipedia.org/wiki/Relative_permittivity
tan_air=0; % (unitless) loss tangent of air. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps2_air=(tan_air*(omega*eps1_air)-sigma_air)/omega; % (unitless) relative
```

```
% loss factor of air


if x<=L/3 || x>=2*L/3 % x is in air part
   a=omegac*mu_air*eps1_air;
   b=omegac*mu_air*eps2_air;
else % x is in water part
   a=omegac*mu_wat*eps1_wat;
   b=omegac*mu_wat*eps2_wat;
end

%y(1)=S, y(2)=S', y(3)=W, y(4)=W'
dydx = [ y(2);
       a*y(1) + b*y(3);
        y(4);
        -b*y(1) + a*y(3)];
return

%
    ----------------------------------------------------------------------------------

function res = onedembc(ya, yb, S0, omega,omegac,L) %#ok<*INUSD>
% ONEDEMBC Evaluate the residual in the boundary conditions
s0 = S0(1); w0 = S0(2);
res = [ ya(1) - s0; % S(0) = s0
       ya(3) - w0; % W(0) = w0
       yb(1) - 0; % S(L) = sL
       yb(3) - 0]; % W(L) = wL
return
```

MATLAB Finite Element Method for the One-Dimensional Helmholtz, Laplace, and Poisson Equations

```
function T=jinsolve(x,l,a,b,f,lcond,rcond)
% function T=jinsolve(x,l,a,b,f,lcond,rcond)
%
% Performs FEM analysis of the 1D electromagnetic scenario decribed in Jin,
% Chapter 3. Solves the differential equation:
%
% -d/dx(a * du/dx) + b*u = f,
%
% Where a, b are known parameters associated with the physical properties
% of the domain, and f is a known source or excitation function. The
% standard one-dimensional Laplace, Poisson, and Helmholtz equations are
% special forms of this equation.
%
% Inputs: x - Domain on which solution is found (should be a vector)
% l - Node spacing (should be of length length(x)-1)
% a - See equation above (for each element--length should match that of h)
% b - See equation above (for each element--length should match that of h)
% f - See equation above (for each element--length should match that of h)
% lcond - Left-hand boundary condition.
% If Dirichlet, give a the value;
% If Third kind, give the row vector [g q], where
% [a*(du/dx) + g*u](x=0) = q.
% rcond - RIght-hand boundary condition.
% If Dirichlet, give a the value;
% If Third kind, give the row vector [g q], where
% [a*(du/dx) + g*u](x=L) = q.

n=length(x);
m=n-1;

% Construct the K matrix
K=zeros(n,n);
K(1,1)=a(1)/l(1)+b(1)*l(1)/3;
K(n,n)=a(m)/l(m)+b(m)*l(m)/3;
for i=2:n-1
   K(i,i)=a(i-1)/l(i-1)+b(i-1)*l(i-1)/3+a(i)/l(i)+b(i)*l(i)/3;
   K(i+1,i)=-a(i)/l(i)+b(i)*l(i)/6;
   K(i,i+1)=-a(i)/l(i)+b(i)*l(i)/6;
```

```
end
for i=[1,n-1]
  K(i+1,i)=-a(i)/l(i)+b(i)*l(i)/6;
  K(i,i+1)=-a(i)/l(i)+b(i)*l(i)/6;
end

% Construct the right-hand side
rhs=zeros(n,1);
rhs(1)=f(1)*l(1)/2;
rhs(n)=f(m)*l(m)/2;
for i=2:n-1
   rhs(i)=f(i-1)*l(i-1)/2+f(i)*l(i)/2;
end

% Modify the matrix and rhs to account for boundary conditions
if length(lcond)==1
   % Incorporate the Dirichlet condition at the left-hand endpoint
   rhs=rhs-K(:,1).*lcond; rhs=rhs(2:end);
   K=K(2:n,2:n);
elseif length(lcond)==2
   % Incorporate the third-kind BC at the left-hand endpoint
   K(1,1)=K(1,1)+lcond(1);
   rhs(1)=rhs(1)+lcond(2);
else
   error('Check left-hand boundary condition. Type \n>> help jinsolve\n for details');
end

if length(rcond)==1
   % Incorporate the Dirichlet condition at the right-hand endpoint
   rhs=rhs-K(:,end).*rcond; rhs=rhs(1:end-1);
   K=K(1:end-1,1:end-1);
elseif length(rcond)==2
   % Incorporate the third-kind BC at the right-hand endpoint
   K(end,end)=K(end,end)+rcond(1);
   rhs(end)=rhs(end)+rcond(2);
else
   error('Check right-hand boundary condition. Type \n>> help jinsolve\n for details');
end

% Solve the system
T=K\rhs;
```

```matlab

% Correct vector lengths in the event of Dirichlet conditions
if length(lcond)==1, T=[lcond;T]; end
if length(rcond)==1, T=[T;rcond]; end
```

MATLAB Test: Reflection for the Metal-Backed Dielectric Slab

```
function [ER,HR]=jin1Dslab(n,thetavec)

% Physical constants
mu0=pi*4e-7; %[N/A^2] permeability of free space
eps0=8.854e-12; %[F/m] permittivity of free space
c=299792458; % [m/s] speed of light in a vacuum

% Physical setup
nu=2.45e9; % [Hz] frequency of microwaves
omega=2*pi*nu; % [Hz] angular frequency of microwaves
lambda=c/nu; % [m] wavelength of microwaves
k0=omega*sqrt(eps0*mu0);
L=5*lambda; %length of domain [m]
P=40e3; % [W] power supplied by magnetron at left-hand endpoint
beta=pi/L; % [1/m] propagation constant
E0=1;%(2/L)*sqrt(2*P*(omega*mu0/beta)); % magnitude of incidence field
H0=1; % magnitude of incidence field

% Nodes and spacing
%n=50; % number of (uniformly spaced) spatial nodes
x=linspace(0,L,n); %vector of x-values
l=x(2:end)-x(1:n-1); %h-values (as spacing is uniform, h is a multiple of ones vector)

% Material properties
mu=2-0.1*1i*ones(size(x));
eps=4+(2-0.1*1i).*(1-x/L).^2;

% Solutions
ER=zeros(size(thetavec)); HR=zeros(size(thetavec));
for ii=1:length(thetavec);
   theta=thetavec(ii);
   % Ez-polarization FEM first
   a=1./mu; b=-k0^2*(eps-(sin(theta))./mu); f=zeros(size(x));
   lcond=0; % Homogeneous Dirichlet at left-hand boundary
   g=1j*k0*cos(theta); q=2*g*E0*exp(L*g);
   rcond=[g q]; % Third kind at right-hand boundary
   E=jinsolve(x,l,a,b,f,lcond,rcond);
   R=(E(n)-E0*exp(L*g))/(E0*exp(-L*g));
   ER(ii)=R;
```

```

    % Hz-polarization FEM next
    a=1./eps; b=-k0^2*(mu-(sin(theta))./eps); f=zeros(size(x));
    lcond=[0 0]; % Homogeneous Neumann at left-hand boundary
    g=1j*k0*cos(theta); q=2*g*H0*exp(L*g);
    rcond=[g q]; % Third kind at right-hand boundary
    H=jinsolve(x,l,a,b,f,lcond,rcond);
    R=(H(n)-H0*exp(L*g))/(H0*exp(-L*g));
    HR(ii)=R;
end
```

```
function []=jin1D()
% function []=jin1D()
%
% Performs FEM analysis of the 1D electromagnetic scenario decribed in Jin,
% Chapter 3. Solves the differential equation:
%
% -d/dx(a * du/dx) + b*u = f,
%
% Where a, b are known parameters associated with the physical properties
% of the domain, and f is a known source or excitation function. The
% standard one-dimensional Laplace, Poisson, and Helmholtz equations are
% special forms of this equation; we take the parameters a and b here to
% be:
%
% a= 1/mu and b=mu*(omega/c)^2*eps
%
% Which correspond to the Helmholtz equation in one dimension. The boundary
% conditions are taken to be the inhomogeneous Dirichlet condition at the
% right-hand endpoint (where the magnetron is located), and the homogeneous
% Dirichlet condition at the right-hand endpoint (where there is a
% perfectly electrically conducting wall).
%


% Physical setup
L=0.248; %length of domain [m]
P=40e3; % [W] power supplied by magnetron at left-hand endpoint
omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves
beta=pi/L; % [1/m] propagation constant

% Nodes and spacing
n=50; % number of (uniformly spaced) spatial nodes
x=linspace(0,L,n); %vector of x-values
l=x(2:end)-x(1:n-1); %h-values (as spacing is uniform, h is a multiple of ones vector)

% Time scenario
dt=1; % length of time step [sec]
time=0; % starting time [sec]
tsim=60; % time for which to perform the simulation [sec]

% Physical constants
```

```
mu0=pi*4e-7; %[N/A^2] permeability of free space
c=299792458; % [m/s] speed of light in a vacuum

magnetron=(2/L)*sqrt(2*P*(omega*mu0/beta)); % value of field at RHS

% Load materials
mu_wat=1; % (unitless) relative permeability of water
sigma_wat=0.055; % [S/m] electrical conductivity of water
eps1_wat=78.54; % (unitless) relative dielectric constant of water. See:
% http://www.kayelaby.npl.co.uk/general_physics/2_6/2_6_5.html ( 25 C )
tan_wat=0.16; %(unitless) loss tangent of water. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps2_wat=(tan_wat*(omega*eps1_wat)-sigma_wat)/omega; % (unitless) relative
% loss factor of water
mu_air=1; % (unitless) relative permeability of air
sigma_air=0; % [S/m] electrical conductivity of air
eps1_air=1; % (unitless) relative dielectric constant of air. See Wikipedia
% article: http://en.wikipedia.org/wiki/Relative_permittivity
tan_air=0; % (unitless) loss tangent of air. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps2_air=(tan_air*(omega*eps1_air)-sigma_air)/omega; % (unitless) relative
% loss factor of air

% Elemental values of physical properties
lim1=floor((n-1)/3); lim2=ceil(2*(n-1)/3); % limits for L/3 and 2L/3
mu=[mu_air*ones(lim1,1); mu_wat*ones(lim2-lim1,1); mu_air*ones((n-1)-lim2,1)]';
%sigma=[sigma_air*ones(lim1,1); sigma_wat*ones(lim2-lim1,1); sigma_air*ones((n-1)-lim2,1)
    ]';
eps1=[eps1_air*ones(lim1,1); eps1_wat*ones(lim2-lim1,1); eps1_air*ones((n-1)-lim2,1)]';
eps2=[eps2_air*ones(lim1,1); eps2_wat*ones(lim2-lim1,1); eps2_air*ones((n-1)-lim2,1)]';
eps=eps1-1i*eps2;


a=1./mu;
b=(omega/c)^2*mu.*eps;
f=zeros(n-1,1);
lcond=magnetron;
rcond=0;

```

```
T=jinsolve(x,l,a,b,f,lcond,rcond);
S=real(T); W=imag(T);

% Display the field through time
%while time<tsim
   %Plot current field distribution
   figure(2); clf; hold on
   plot(100*x,S,'b',100*x,W,'r');%S*cos(omega*time));
   legend('Re(E)','Im(E)');
% plot(100*x, S*cos(omega*time)+W*sin(omega*time));
% title(strcat('Field distribution at t=',num2str(time,'%11.3g'),' seconds'));
   xlabel('Length [cm]'); ylabel('Electric field intensity [V/m] or [N/C]');
   title('Jin''s method');
   grid on

   %Update time = t_{k+1}
   time=time+dt;
%end %while time<tsim
end
```

D.8 python Implementation of the Finite Difference Method for the Two-Dimensional Helmholtz Equation

```
def helmsolve(hx,hz,K,E_inc):
      mu0=pi*4e-7 # permeability of free space [N/A^2]

      hx_sq = 1/(hx[1:]*hx[:-1])
      hz_sq = 1/(hz[1:]*hz[:-1])

      M = np.size(hx_sq)+2
      N = np.size(hz_sq)+2

      A = np.zeros((N*M,N*M))

      #print "number of xnodes is "+str(M)
      #print "number of znodes is "+str(N)

      #print "size of A is"
      #print np.shape(A)

      for k in range (0,M):
```

```
                A[k,k] = 1.0
                A[(N-1)*M+k,(N-1)*M+k] = 1.0
                #print "Changed the values of A["+str(k)+","+str(k)+"] and A["+str((N-1)*M+k)+","+str((N-1)*M+k)+"]"

        for j in range (1,N-1):
                #print "j="+str(j)
                A[j*M,j*M]=1.0
                A[(j+1)*M-1,(j+1)*M-1]=1.0
                #print "Changed the values of A["+str(j*M)+","+str(j*M)+"] and A["+str((j+1)*M-1)+","+str((j+1)*M-1)+"]"
                for k in range (1,M-1):
                        #print "k="+str(k)
                        A[j*M+k,j*M+k]=K[k-1,j-1]-2*hx_sq[k-1]-2*hz_sq[j-1]
                        A[j*M+k,(j-1)*M+k] = hz_sq[j-1]
                        A[j*M+k,(j+1)*M+k] = hz_sq[j-1]
                        A[j*M+k,j*M+k+1] = hx_sq[k-1]
                        A[j*M+k,j*M+k-1] = hx_sq[k-1]
                        #print "Changed the values of A on row "+str(j*M+k)

        b = np.array([0.0]*N*M)
        b[:M]=E_inc

        E = linalg.solve(A,b)

        return 0.5*E*E
```

D.9 MATLAB Implementation of the Finite Element Method for the Two-Dimensional Helmholtz Equation

```
function T=jin2D(nodefile,elfile,bc1file,bc2file)
% function T=jin2D(nodefile,elfile,bc1file,bc2file)
%
% Performs FEM analysis of the 2D electromagnetic scenario decribed in Jin,
% Chapter 4. Solves the differential equation:
%
% -d/dx(alpha_x * du/dx) - d/dy(alpha_y * du/dy) + beta*u = f,
%
% Where a_x, a_y, and b are known parameters associated with the physical
% properties of the domain, and f is a known source or excitation function.
% The standard two-dimensional Laplace, Poisson, and Helmholtz equations
% are special forms of this equation.
%
% This solver takes a list of elements and nodes for a given domain as
% inputs, and provides the solution on that domain.
%
% Inputs: nodefile — name of file containing node coordinates. Should be
% formatted as follows (without the header):
%
% (x—coord) (y—coord)
% 0 0
% . .
% . .
% . .
%
% elfile — name of file containing element definitions.
% Contains information about alpha_x, alpha_y, beta,
% and f at each element as well. Should be formatted
% as follows (without the header):
%
% (node1) (node2) (node3) (ax) (ay) (beta) (f)
% 1 2 5 1 3 1 0
% . . . . . . .
% . . . . . . .
% . . . . . . .
%
% bcfile1 — name of file containing the nodes on the
% Dirichlet boundary and their values. Should be
% formatted as follows:
%
% (node number) (value)
% 1 118
% . .
% . .
% . .
%
% bcfile2 — name of file containing the edges on the
% third—kind boundary. Should be formatted as follows:
%
% (node1) (node2) (g) (q)
% 1 2 5 1
% . . . .
% . . . .
% . . . .

% Read in elements, nodes, and boundary conditions to matrices

N=dlmread(nodefile); % nodes
   nn=length(N(:,1)); %number of nodes
E=dlmread(elfile); % elements
   el=length(E(:,1)); %number of elements
% E(:,5)=ones(size(E(:,5))); % alpha_x
% E(:,6)=E(:,5); % alpha_y
% E(:,7)=E(:,5); % beta
% E(:,8)=E(:,5); % f

% Assemble the LHS matrix and RHS vector

s=zeros(1,3); x=s; y=s; b=s; c=s; % Initialize the vectors that store local
% node numbers and coordinate values
K=zeros(nn,nn); % Initialize the LHS matrix
rhs=zeros(size(N(:,1)));
for k=1:el % for each element
  % Get the alpha_x, alpha_y, beta, and f values for the current element
  alpha_x=E(k,4); alpha_y=E(k,5); beta=E(k,6); f=E(k,7);

  % Get the node numbers and coordinates for the current element
  for m=1:3
    s(m)=E(k,m); % numbers of the nodes in the order they appear (should be ccw in the list
        )
    x(m)=N(s(m),1); % x-coordinates of nodes in the order they appear
    y(m)=N(s(m),2); % y-coordinates of nodes in the order they appear
  end % for m=1:3

  % Calculate the area of the element and all b and c coefficients (a not necessary)
  Ar=polyarea(x,y); % area of the current element
  b(1)=y(2)-y(3); b(2)=y(3)-y(1); b(3)=y(1)-y(2); % b-coefficients
  c(1)=-(x(2)-x(3)); c(2)=-(x(3)-x(1)); c(3)=-(x(1)-x(2)); % c-coefficients

  % Populate the LHS matrix
  for i=1:3
    for j=1:3
      if i==j, delta=1; else delta=0; end

      K(s(i),s(j))=K(s(i),s(j))+(1/(4*Ar))*(alpha_x*b(i)*b(j)+alpha_y*c(i)*c(j)) + ...
         (Ar/12)*beta*(1+delta);
    end % for j=1:3

    rhs(s(i))=rhs(s(i))+Ar*f/3;

  end % for i=1:3
end % for k=1:el

% Boundary conditions and solving

if ~isempty(bc2file)
    % Impose the Third−kind condition
    BC2=dlmread(bc2file); % Third−kind boundary
    s=zeros(1,2); % Initialize the vector that stores local node numbers
    for k=1:length(BC2(:,1)) % for each edge on the boundary
        s(1)=BC2(k,1); s(2)=BC2(k,2); % get universal node numbers
        g=BC2(k,3); q=BC2(k,4); % get gamma and q values
        l=sqrt((N(s(1),1)−N(s(2),1))^2+(N(s(1),2)−N(s(2),2))^2); % get length of segment
        for i=1:2
            for j=1:2
                if i==j, delta=1; else delta=0; end
                K(s(i),s(j))=K(s(i),s(j))+g*l*(1+delta)/6; % modify K at the two nodes
            end % for j=1:2
            rhs(s(i))=rhs(s(i))+q*l; % modify rhs
        end % for i=1:2
    end % for k=1:length(BC2(:,1))
end % if ~isempty(bcfile2)

if ~isempty(bc1file)
    % Impose the Dirichlet boundary condition
    BC1=dlmread(bc1file);
    for i=1:length(BC1(:,1))
        nn=BC1(i,2);
        rhs=rhs−K(:,nn)*BC1(i,3);
        rhs(j)=[];
        K(:,j)=[]; K(j,:)=[];
    end % for i=1:length(BC1(:,1))

    % Solve the system
    T_small=K\rhs;

    % Clean up the solution (re-insert the values on the Dirichlet boundary)
    flag=max(BC1(:,3))+10; % a flag that we know is not one of the boundary values
    T=flag*ones(size(N(:,1))); % temporarily set all T entries to the flag

    T(BC1(i,2))=BC1(i,3); % replace the flags with the values on the Dirichlet
    % boundary in those positions
    T(T==flag)=T_small; % use the remaining flags put the values calculated
    % in the matrix where they belong
else
    % No Dirichlet condition to impose, just solve the system
    T=K\rhs;
end % if ~isempty(bc1file)

end % function
```

MATLAB Test: Discontinuity in a Parallel-Plate Waveguide

```
function []=parallelplate()

lambda=0.10; % wavelgnth [m]
objx=0.05; % length of the domain (x-direction) [m]
hml=1; % how many wavelengths the artificial boundary is from the inclusion
H=(0:0.05:0.35)*lambda; % h-values to use in generating the graph on p. 110 of Jin

mgx=0:0.025:2*hml*lambda+0.05; % domain in x-direction
mgy=0:0.005:0.035; % domain in y-direction

%% eps2=4

eps2=4;

% For R,T plots
Re1=zeros(size(H));
Te1=zeros(size(H));
for j=1:length(H)
 h=H(j);
 [R,T,Hz,p,t]=findRT(h,eps2,lambda,objx,hml);
 Re1(j)=abs(R);
 Te1(j)=abs(T);
```

```
end

% For equi-Hz contour
h=0.0175;
[RHz1,THz1,Hz1,p1,t1]=findRT(h,eps2,lambda,objx,hml);

%% eps2=4-i

eps2=4-1j;

% For R,T plots
Re2=zeros(size(H));
Te2=zeros(size(H));
for j=1:length(H)
 h=H(j);
 [R,T,Hz,p,t]=findRT(h,eps2,lambda,objx,hml);
 Re2(j)=abs(R);
 Te2(j)=abs(T);
end

% For equi-Hz contour
h=0.0175;
[RHz2,THz2,Hz2,p2,t2]=findRT(h,eps2,lambda,objx,hml);

%% eps2=4-10i

eps2=4-1j*10;

% For R,T plots
Re3=zeros(size(H));
Te3=zeros(size(H));
for j=1:length(H)
 h=H(j);
 [R,T,Hz,p,t]=findRT(h,eps2,lambda,objx,hml);
 Re3(j)=abs(R);
 Te3(j)=abs(T);
end

% For equi-Hz contour
h=0.0175;
[RHz3,THz3,Hz3,p3,t3]=findRT(h,eps2,lambda,objx,hml);
```

```


%% Plotting equi−Hz contours

figure(1); clf; hold on;
subplot(6,1,1),
   F=TriScatteredInterp(p1(:,1),p1(:,2),real(Hz1));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
% mesh(xq,yq,vq); hold on; plot3(p1(:,1),p1(:,2),real(Hz1),'o'); hold off;
% trisurf(t1(:,1:3),p1(:,1),p1(:,2),real(Hz1),'facecolor','interp');
   view(2);
   set(gca,'plotboxaspectratio',[16 2 1]);
   title('Real Part, \epsilon=4.0−0i');
subplot(6,1,2),
% trisurf(t1(:,1:3),p1(:,1),p1(:,2),imag(Hz1),'facecolor','interp');
   F=TriScatteredInterp(p1(:,1),p1(:,2),imag(Hz1));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
   title('Imaginary Part, \epsilon=4.0−0i');
   set(gca,'plotboxaspectratio',[16 2 1]);
   view(2);
subplot(6,1,3),
   F=TriScatteredInterp(p2(:,1),p2(:,2),real(Hz2));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
% trisurf(t2(:,1:3),p2(:,1),p2(:,2),real(Hz2),'facecolor','interp');
   set(gca,'plotboxaspectratio',[16 2 1]);
   title('Real Part, \epsilon=4.0−1i');
   view(2);
subplot(6,1,4),
   F=TriScatteredInterp(p2(:,1),p2(:,2),imag(Hz2));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
% trisurf(t2(:,1:3),p2(:,1),p2(:,2),imag(Hz2),'facecolor','interp');
   set(gca,'plotboxaspectratio',[16 2 1]);
   title('Imaginary Part, \epsilon=4.0−1i');
   view(2);
subplot(6,1,5),
   F=TriScatteredInterp(p3(:,1),p3(:,2),real(Hz3));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
% trisurf(t3(:,1:3),p3(:,1),p3(:,2),real(Hz3),'facecolor','interp');
   set(gca,'plotboxaspectratio',[16 2 1]);
   title('Real Part, \epsilon=4.0-10i');
   view(2);
subplot(6,1,6),
   F=TriScatteredInterp(p3(:,1),p3(:,2),imag(Hz3));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
% trisurf(t3(:,1:3),p3(:,1),p3(:,2),imag(Hz3),'facecolor','interp');
   set(gca,'plotboxaspectratio',[16 2 1]);
   title('Imaginary Part, \epsilon=4.0-10i');
   view(2);

%% Plots for SIAM Cover Photo

figure(4); clf; hold on;
   F=TriScatteredInterp(p1(:,1),p1(:,2),real(Hz1));
   [xq,yq]=meshgrid(mgx,mgy);
   vq = F(xq,yq);
   contour(xq,yq,vq,15);
   mesh(xq,yq,vq); hold on; plot3(p1(:,1),p1(:,2),real(Hz1),'o'); hold off;
   trisurf(t1(:,1:3),p1(:,1),p1(:,2),real(Hz1),'facecolor','interp');
   set(gca,'plotboxaspectratio',[16 2 1]);
   title('Real Part, \epsilon=4.0-0i');


%% Plotting reflection and transmission
H=H/lambda;

figure(2); clf; hold on; grid on;
plot(H,Re1,'k-',H,Re2,'b--',H,Re3,'r-.');
xlabel('h/\lambda'); ylabel('|R|');
legend('\epsilon = 4 - 0i','\epsilon = 4 - 1i','\epsilon = 4 - 10i','Location','NorthWest
    ');
```

```

figure(3); clf; hold on; grid on;
plot(H,Te1,'k-',H,Te2,'b--',H,Te3,'r-.');
xlabel('h/\lambda'); ylabel('|T|');
legend('\epsilon = 4 - 0i','\epsilon = 4 - 1i','\epsilon = 4 - 10i','Location','SouthWest
    ');

end

function [R,T,Hz,p,t]=findRT(objy,eps2,lambda,objx,hml)
% Solves the problem described from Page 105 of Jin: solves for the
% electric and magnetic fields near a discontinuity in a parallel-plate
% waveguide.

%% Initializing -- problem setup

% Delete old data files

delete '*.dat'

% Physical constants

mu0=pi*4e-7; %[N/A^2] permeability of free space
eps0=8.854e-12; %[F/m] permittivity of free space
c=299792458; % [m/s] speed of light in a vacuum

%lambda=0.1; % [cm] wavelength
omega=2*pi*c/lambda; % [Hz] angular frequency of microwaves
k0=omega*sqrt(eps0*mu0);
H0=10; % magnitude of incidence field

% Dielectric properties and coefficient values
mu1=1; % (unitless) relative permeability of air
sigma1=0; % [S/m] electrical conductivity of air
eps11=1; % (unitless) relative dielectric constant of air. See Wikipedia
% article: http://en.wikipedia.org/wiki/Relative_permittivity
tan1=0; % (unitless) loss tangent of air. See:
% http://cp.literature.agilent.com/litweb/pdf/genesys200801/elements/
% substrate_tables/tablelosstan.htm
eps12=(tan1*(omega*eps11)-sigma1)/omega; % (unitless) relative loss factor of air
eps1=eps11+1j*eps12;
```

```

mu2=2-0.1*1j;

ax1=1/eps1; ay1=1/eps1; beta1=-k0^2*mu1; f=0;
ax2=1/eps2; ay2=1/eps2; beta2=-k0^2*mu2;

% Geometrical setup

%objx=0.05; % [cm] object length (x-dir)
domx=2*hml*lambda+objx;
domy=0.035;

% Filenames for node and element lists

nodefile='n.dat';
elfile='e.dat';
bc2file='bc2.dat';

%% Meshing and saving lists

% Meshing

node = [0,0;domx,0;domx,domy;0,domy]; % coordinates of corners of domain (to use in mesh2d)
hdata.fun = @hfun; % function specifying size of mesh (to be used in mesh2d)
hdata.args={lambda,objx,objy,hml}; % arguments for hfun in addition to x,y (to use in
    mesh2d)
%options.mlim=0.02; % The convergence tolerance. The maximum percentage change in edge
% length per iteration must be less than mlim
%options.maxit=20; % The maximum allowable number of iterations.
%options.dhmax=0.30; % The maximum allowable (relative) gradient in the size function
options.output=false; % suppresses output for mesh generation

[p,t] = mesh2d(node,[],hdata,options); % generates mesh of domain

% Element-varying properties

p(:,3)=0; % flag indicating node is outside dielectric rod
chgind=intersect(intersect(find(p(:,1)>=hml*lambda),...
     find(p(:,1)<=hml*lambda+objx)),find(p(:,2)<=objy)); % indices of nodes in the rod
p(chgind,3)=1; % flag indicating node is inside dielectric rod

```

```
eltsin=intersect(intersect(find(p(t(:,1),3)==1),find(p(t(:,2),3)==1)),...
     find(p(t(:,3),3)==1)); % indices of elements in the rod

t(:,4)=ax1; t(:,5)=ay1; t(:,6)=beta1; t(:,7)=f; % putting material properties of air in
    place
t(eltsin,4)=ax2; t(eltsin,5)=ay2; t(eltsin,6)=beta2; % put properties of dielectric in
    place

% Writing node and element lists
%p=[(1:1:length(p(:,1)))',p]; % putting node numbers in leftmost column
p=p(:,1:2);
dlmwrite(nodefile,p,'delimiter','\t','precision','%.6f'); % save nodes
dlmwrite(elfile,t,'delimiter','\t','precision','%.6f'); % save elements

% Third kind condition at left-hand boundary

ABg=1j*k0*ax1;
ABq=2*1j*k0*H0*ax1;

ABnodes=find(p(:,1)==0); % find nodes on the left-hand aboundary
for i=1:length(t(:,1))
   A=t(i,1:3);
   a1=any(ABnodes==A(1));
   a2=any(ABnodes==A(2));
   a3=any(ABnodes==A(3));

   anysum=a1+a2+a3;

   if anysum==2
       % Then an edge is on the boundary
      if a1==1
         if a2==1
            % Then nodes 1 and 2 are on the boundary
            writebc=[A(1),A(2),ABg,ABq];
         elseif a3==1
            % Then nodes 1 and 3 are on the boundary
            writebc=[A(1),A(3),ABg,ABq];
         end % if a2==1
      elseif a2==1
         % Then nodes 2 and 3 are on the boundary
         writebc=[A(2),A(3),ABg,ABq];
      end % if a1==1
      dlmwrite(bc2file,writebc,'-append','delimiter','\t','precision','%.6f')
   end % if anysum==2
end % for i=1:length(t(:,1))

% Third kind condition at right-hand boundary

CDg=1j*k0*ax1;
CDq=0;

CDnodes=find(p(:,1)==domx); % find nodes on the right-hand aboundary
for i=1:length(t(:,1))
   A=t(i,1:3);
   a1=any(CDnodes==A(1));
   a2=any(CDnodes==A(2));
   a3=any(CDnodes==A(3));

   anysum=a1+a2+a3;

   if anysum==2
       % Then an edge is on the boundary
      if a1==1
         if a2==1
            % Then nodes 1 and 2 are on the boundary
            writebc=[A(1),A(2),CDg,CDq];
         elseif a3==1
            % Then nodes 1 and 3 are on the boundary
            writebc=[A(1),A(3),CDg,CDq];
         end % if a2==1
      elseif a2==1
         % Then nodes 2 and 3 are on the boundary
         writebc=[A(2),A(3),CDg,CDq];
      end % if a1==1
      dlmwrite(bc2file,writebc,'-append','delimiter','\t','precision','%.6f')
   end % if anysum==2
end % for i=1:length(t(:,1))

%% FEM Solve

Hz = jin2D(nodefile,elfile,[],bc2file);
```

```

%% Plotting mesh and equi-Hz curves and calculating R, T

figure(4); clf; hold on;
trimesh(t(:,1:3),p(:,1),p(:,2));
title('Sample Finite Element Mesh');
set(gca,'plotboxaspectratio',[6 1 1]);

x1=intersect(find(p(:,1)==0),find(p(:,2)==0)); % Get number of the node at bottom left
x2=intersect(find(p(:,1)==domx),find(p(:,2)==0)); % Get number of the node at bottom right

R=(Hz(x1)-H0*exp(-1j*k0*0.1))/(H0*exp(1j*k0*0.1));
T=(Hz(x2))/(H0*exp(-1j*k0*domx));

fprintf('|R|^2+|T|^2=%g,\n',abs(R)^2+abs(T)^2);


end % parallelplate


%% Mesh refinement function
function h=hfun(x,y,lambda,objx,objy,hml)
% User defined size function

h=0.01*ones(size(x,1),1); % size 0.001 outside the dielectric rod

in=(x>=hml*lambda)&(x<=hml*lambda+objx)&(y<=objy); % size 0.0001 inside
h(in)=0.005;

end % hfun
```

Appendix E

Computer Implementation in MATLAB and python of the Solvers for the 1D and 2D Heat Equations

E.1 python Implementation of the Finite Difference Method for the One-Dimensional Heat Equation

```
from pylab import * # so we know what sqrt is
import scipy.sparse as sp # for using sparse matrix tools

def finite_diff_theta(T_old,h,c_p,rho,kappa,eavg,h_dt,sig,theta,bc,htil,T0):
	"""finite_diff_theta(T_old,h,c_p,rho,kappa,eavg,h_dt,sig,theta,bc,htil,T0):
	    Implements a single timestep of the theta-scheme for solving the heat equation
	    using the finite difference method.

	Inputs:
		T_old  The initial temperature field. A vector (array) of length n.
		h  The differences between x-values [m]. A vector (array) of length n-1.
		c_p  The specific heat capacity [J/(gC)]. A vector (array) of length n.
		rho  The density [g/m^3]. A vector (array) of length n.
		kappa  The thermal conductivity [W/(mC)]. A vector (array) of length n.
		eavg  The power dissipated into each point of the simulated domain. A vector
		    (array) of length n.
		h_dt  The length of the thermal timestep [sec]. A scalar.
		sig  The electrical conductivity [S/m]. A vector (array) of length n.
```

```
        theta  The parameter for the theta—method. A scalar that varies from 0 to 1. Take theta = 0 for a fully explicit method, theta = 1 for a fully implicit method, and theta = 0.5 for a Crank—Nicolson method.
        bc  The type of boundary condition to use at the left— and right—hand endpoints. A string that takes either the value 'ins' (for the insulated boundary conditions; i.e., homogeneous Neumann), 'fix' (for the fixed temperature boundary conditions; i.e., inhomogeneous Dirichlet), or 'rad' (for the radiative boundary conditions; i.e., third—kind or mixed conditions).
        htil  The heat transfer coefficient for insulation material (if using radiative BC). A scalar.
        T0  The initial and ambient temperature of air surrounding sample and insulation. Scalar, degC.

    Outputs:
        T_new  The temperature field after a single timestep. A vector (array) of length n.


    """

    # Important constants
    eps0 = 8.8541878176e—12 # permittivity of free space

    h_sq = h[1:]*h[:—1]
    # Useful parameter
    s = h_dt*kappa/(rho*c_p*h_sq)

    # Implement the boundary conditions——choice of Dirichlet, Neumann, or radiative. Each BC can be written in the mixed formulation a1*T_x(0,t) + a2*T(0,t) = g1(t) and a3*T_x(L,t) + a4*T(L,t) = g2(t), with some or other parameters being zero, strategically
    if bc == 'ins': # insulated boundary at both left and right—hand endpoints: homogeneous Neumann condition
        a1 = 1
        a2 = 0
        g1 = 0
        a3 = 1
        a4 = 0
        g2 = 0
        A00 = 1+2*s[0]*theta*(1—sqrt(h_sq[0])*a2/a1)
```

```
            A01 = -2*s[0]*theta
            Ann = 1+2*s[-1]*theta*(1+sqrt(h_sq[-1])*a4/a3)
            Anm = -2*s[-1]*theta
            B00 = 1+2*s[0]*(1-theta)*(sqrt(h_sq[0])*a2/a1-1)
            B01 = 2*(1-theta)*s[0]
            Bnn = 1-2*s[0]*(1-theta)*(sqrt(h_sq[-1])*a4/a3+1)
            Bnm = 2*(1-theta)*s[-1]
            q0 = -2*g1*s[0]*sqrt(h_sq[0])/a1 + eavg[0]*h_dt/(rho[0]*c_p[0])
            qn = 2*g2*s[-1]*sqrt(h_sq[-1])/a3 + eavg[-1]*h_dt/(rho[-1]*c_p[-1])
        elif bc == 'rad': # radiative BC at both left and right-hand endpoints
            a1 = 1
            a2 = -htil # represents a1*T_x = -h(T-T_inf)
            g1 = -htil*T0
            a3 = 1
            a4 = htil
            g2 = htil*T0
            A00 = 1+2*s[0]*theta*(1-sqrt(h_sq[0])*a2/a1)
            A01 = -2*s[0]*theta
            Ann = 1+2*s[-1]*theta*(1+sqrt(h_sq[-1])*a4/a3)
            Anm = -2*s[-1]*theta
            B00 = 1+2*s[0]*(1-theta)*(sqrt(h_sq[0])*a2/a1-1)
            B01 = 2*(1-theta)*s[0]
            Bnn = 1-2*s[0]*(1-theta)*(sqrt(h_sq[-1])*a4/a3+1)
            Bnm = 2*(1-theta)*s[-1]
            q0 = -2*g1*s[0]*sqrt(h_sq[0])/a1 + eavg[0]*h_dt/(rho[0]*c_p[0])
            qn = 2*g2*s[-1]*sqrt(h_sq[-1])/a3 + eavg[-1]*h_dt/(rho[-1]*c_p[-1])
        elif bc == 'fix': # fixed temperature at both endpoints (Dirichlet condition)
            a1 = 0
            a2 = 1
            g1 = T0
            a3 = 0
            a4 = 1
            g2 = T0
            A00 = 1
            A01 = 0
            Ann = 1
            Anm = 0
            B00 = 0
            B01 = 0
            Bnn = 0
            Bnm = 0
            q0 = g1
            qn = g2
    else: # throw an error if bc is none of those strings
            print "The input variable 'bc' must be either the string 'ins' or the string
                'rad' or the string 'fix'"
            input()
            import sys
            sys.exit(1)

    # Create A-matrix (for solving A*T_new = B*T_old + q)
    maindiag = r_[A00,1+2*theta*s[1:-1],Ann]
    lowerdiag = r_[-theta*s[1:-1],Anm]
    upperdiag = r_[A01,-theta*s[1:-1]]
    diagonals = [ maindiag , lowerdiag , upperdiag ]
    A = sp.diags(diagonals,[0,-1,1]).toarray() # make A directly as a sparse matrix

    # Create B-matrix (for solving A*T_new = B*T_old + q)
    maindiag = r_[B00,1-2*(1-theta)*s[1:-1],Bnn]
    lowerdiag = r_[(1-theta)*s[1:-1],Bnm]
    upperdiag = r_[B01,(1-theta)*s[1:-1]]
    diagonals = [ maindiag , lowerdiag , upperdiag ]
    B = sp.diags(diagonals,[0,-1,1]).toarray() # make B directly as a sparse matrix

    # Create q (for solving A*T_new = B*T_old + q)
    q = (h_dt/(rho*c_p))*eavg # source term
    q = r_[q0,q[1:-1],qn] # replace first and last entries depending on boundary
        conditions

    # Solve equation A*T_new = B*T_old + q
    T_new = linalg.solve(A,dot(B,T_old)+q)
    return T_new
```

E.2 MATLAB Implementation of the Finite Difference Method for the One-Dimensional Heat Equation

```
function Tk=thermsolve1_fd(temp,h,c_p,rho,kappa,eavg,h_dt,sig)

eps_0 = 8.8541878176e-12; %permittivity of free space

n=length(temp);
kappa1=kappa(1:n-1); rho1=rho(1:n-1); c_p1=c_p(1:n-1);
```

```

dd=h_dt*kappa1./(rho1.*c_p1.*h.*h);

% BC: left and right-hand endpoints insulated and fixed at room temperature (the initial
    temperature)
% so row 1 should be [1 0...0] and row n should be [0...0 1]

a0 = [1,1+2*dd(2:n-1),1]' ; %main diagonal
a1 = [-dd(2:n-1),0]' ; %lower/left diagonal
a2 = [0,-dd(2:n-1)]' ; %upper/right diagonal
% aaa0 = [ ones(IN,1); 1 + dd*2*K_R(Temp(IN+1:Nx-1)); 1 + dd*K_R(Temp(Nx))];
% aaa1 = [ zeros(IN-1,1); -dd*K_R(Temp(IN+1:Nx))];
% aaa2 = [ zeros(IN-1,1); -1; -dd*K_R(Temp(IN+1:Nx-1))];
A = diag(a0) + diag(a1,-1)+ diag(a2,1);
A=sparse(A);
%%Source Term %need neg sign?
 q = h_dt*0.5*sig'.*eavg'./(rho'.*c_p'); % includes multiplying by dt/(rho*c_p)
 %eavg(1:IN-1)'*omega*eps_0.*eps_2L(Temp(1:IN-1))./a_L(Temp(1:IN-1));

 %mean(q)


%%Solve equation
Tk = A\(temp + q); % Solve A*T^(n+1) = T^n + q

end
```

E.3 python Implementation of the Finite Difference Method for the Two-Dimensional Heat Equation

```
from pylab import *
import matplotlib.pyplot as plt
import scipy.sparse as sp

def finite_diff_theta(T_old,hx,hz,eavg,h_dt,theta,phi,bc,h):
      """finite_diff_theta(T_old,h,c_p,rho,kappa,eavg,h_dt,sig,theta,bc): Implements a
          single timestep of the theta-scheme for solving the heat equation using the
          finite difference method.

      Inputs:
            T_old  The initial temperature field. A vector (array) of length (N+1)*(M+1).
            hx  The differences between x-values [m]. A vector (array) of length N.
```

```
        hz  The differences between z-values [m]. A vector (array) of length M.
        eavg  The power dissipated into each point of the simulated domain. A vector (array) of length (N+1)(M+1).
        h_dt  The length of the thermal timestep [sec]. A scalar.
        theta  The parameter for the theta-method in the z-direction. A scalar that varies from 0 to 1. Take theta = 0 for a fully explicit method, theta = 1 for a fully implicit method, and theta = 0.5 for a Crank-Nicolson method.
        phi  The parameter for the theta-method in the x-direction. A scalar that varies from 0 to 1. Take phi = 0 for a fully explicit method, phi = 1 for a fully implicit method, and phi = 0.5 for a Crank-Nicolson method.
        bc  The type of boundary condition to use at the left- and right-hand endpoints. A string that takes either the value 'ins' (for the insulated boundary conditions; i.e., homogeneous Neumann), 'fix' (for the fixed temperature boundary conditions; i.e., inhomogeneous Dirichlet), or 'rad' (for the radiative boundary conditions; i.e., third-kind or mixed conditions).
        h  Coefficient for radiative BC.
    Outputs:
        T_new  The temperature field after a single timestep. A vector (array) of length n.


    """

    # Useful parameters
    hx_sq = hx[1:]*hx[:-1]
    hz_sq = hz[1:]*hz[:-1]
    s = h_dt/(hz_sq)
    r = h_dt/(hx_sq)
    N = np.size(hz)
    M = np.size(hx)

    # Implement the boundary conditions--choice of Dirichlet, Neumann, or radiative. Each BC can be written in the mixed formulation a1*T_x(0,t) + a2*T(0,t) = g1(t) and a3*T_x(L,t) + a4*T(L,t) = g2(t), with some or other parameters being zero, strategically
    if bc == 'ins': # insulated boundary at both left and right-hand endpoints: homogeneous Neumann condition
        a11 = 1.0
        a12 = 0
```

```
            g1 = 0
            a21 = 1.0
            a22 = 0
            g2 = 0
            a31 = 1.0
            a32 = 0
            g3 = 0
            a41 = 1.0
            a42 = 0
            g4 = 0
    elif bc == 'rad': # radiative BC at both left and right−hand endpoints
            a11 = 1.0
            a12 = h # represents a1*T_x = −h(T−1)
            g1 = h
            a21 = 1.0
            a22 = −h
            g2 = −h
            a31 = 1.0
            a32 = h # represents a1*T_x = −h(T−1)
            g3 = h
            a41 = 1.0
            a42 = −h
            g4 = −h
    else: # throw an error if bc is neither of those strings
            print "The input variable 'bc' must be either the string 'ins' or the string
                'rad'"
            input()
            import sys
            sys.exit(1)
    # Coefficients

    # j=1,...N−1, k=1,...M−1
    # c1 is not inputted here, because it needs to be done ITERATIVELY
    Aa1 = Ae1 = −theta*s # vector, length N−1, ITERATIVELY
    Ab1 = Ad1 = −phi*r # vector, length M−1
    Ba1 = Be1 = (1−theta)*s # vector, length N−1, ITERATIVELY
    Bb1 = Bd1 = (1−phi)*r # vector, length M−1

    # j=0, k=1...M−1
    Ac2 = 1+2*theta*s[0]*(1−(2*hz[0]*a12/a11))+2*phi*r # vector, length M−1
    Ab2 = Ad2 = −phi*r # vector, length M−1
```

```
        Ae2 = -2*theta*s[0] # scalar
        Bc2 = 1+2*(theta-1)*s[0]*(1-(2*hz[0]*a12/a11))+2*(phi-1)*r # vector, length M-1
        Bb2 = Bd2 = (1-phi)*r # vector, length M-1
        Be2 = 2*(1-theta)*s[0] # scalar

        # j=N, k=1...M-1
        Ac3 = 1+2*theta*s[-1]*(1+(2*hz[-1]*a22/a21))+2*phi*r # vector, length M-1
        Ab3 = Ad3 = -phi*r # vector, length M-1
        Aa3 = -2*theta*s[-1] # scalar
        Bc3 = 1+2*(theta-1)*s[-1]*(1+(2*hz[-1]*a22/a21))+2*(phi-1)*r # vector, length M-1
        Bb3 = Bd3 = (1-phi)*r # vector, length M-1
        Ba3 = 2*(1-theta)*s[-1] # scalar

        # j=1...N-1, k=0
        Aa4 = Ae4 = -theta*s # vector, length N-1 , to be taken from ITERATIVELY
        Ac4 = 1+2*theta*s+2*phi*r[0]*(1-2*hx[0]*a32/a31) # vector, length N-1, ITERATIVE
        Ad4 = -2*phi*r[0] # scalar
        Ba4 = Be4 = (1-theta)*s # vector, length N-1 , to be taken from ITERATIVELY
        Bc4 = 1+2*(theta-1)*s+2*(phi-1)*r[0]*(1-2*hx[0]*a32/a31) # vector, length N-1,
            ITERATIVE
        Bd4 = 2*(1-phi)*r[0] # scalar

        # j=1...N-1, k=M
        Aa5 = Ae5 = -theta*s # vector, length N-1 , to be taken from ITERATIVELY
        Ac5 = 1+2*theta*s+2*phi*r[-1]*(1+2*hx[-1]*a42/a41) # vector, length N-1, ITERATIVE
        Ab5 = -2*phi*r[-1] # scalar
        Ba5 = Be5 = (1-theta)*s # vector, length N-1 , to be taken from ITERATIVELY
        Bc5 = 1+2*(theta-1)*s+2*(phi-1)*r[-1]*(1+2*hx[-1]*a42/a41) # vector, length N-1,
            ITERATIVE
        Bb5 = 2*(1-phi)*r[-1] # scalar

        # j=k=0
        Ac6 = 1+2*theta*s[0]*(1-(2*hz[0]*a12/a11))+2*phi*r[0]*(1-2*hx[0]*a32/a31) # scalar
        Ad6 = -2*phi*r[0] # scalar
        Ae6 = -2*theta*s[0] # scalar
        Bc6 = 1+2*(theta-1)*s[0]*(1-(2*hz[0]*a12/a11))+2*(phi-1)*r[0]*(1-2*hx[0]*a32/a31)
            # scalar
        Bd6 = 2*(1-phi)*r[0] # scalar
        Be6 = 2*(1-theta)*s[0] # scalar

        # j=N, k=0
        Ac7 = 1+2*theta*s[-1]*(1+(2*hz[-1]*a22/a21))+2*phi*r[0]*(1-2*hx[0]*a32/a31) #
            scalar
        Aa7 = -2*theta*s[-1] # scalar
        Ad7 = -2*phi*r[0] # scalar
        Bc7 = 1+2*(theta-1)*s[-1]*(1+(2*hz[-1]*a22/a21))+2*(phi-1)*r[0]*(1-2*hx[0]*a32/
            a31) # scalar
        Ba7 = 2*(1-theta)*s[-1] # scalar
        Bd7 = 2*(1-phi)*r[0] # scalar

        # j=0, k=M
        Ac8 = 1+2*theta*s[0]*(1-(2*hz[0]*a12/a11))+2*phi*r[-1]*(1-2*hx[-1]*a42/a41) #
            scalar
        Ab8 = -2*phi*r[-1] # scalar
        Ae8 = -2*theta*s[0] # scalar
        Bc8 = 1+2*(theta-1)*s[0]*(1-(2*hz[0]*a12/a11))+2*(phi-1)*r[-1]*(1-2*hx[-1]*a42/
            a41) # scalar
        Bb8 = 2*(1-phi)*r[-1] # scalar
        Be8 = 2*(1-theta)*s[0] # scalar

        # j=N, k=M
        Ac9 = 1+2*theta*s[-1]*(1+(2*hz[-1]*a22/a21))+2*phi*r[-1]*(1-2*hx[-1]*a42/a41) #
            scalar
        Aa9 = -2*theta*s[-1] # scalar
        Ab9 = -2*phi*r[-1] # scalar
        Bc9 = 1+2*(theta-1)*s[-1]*(1+(2*hz[-1]*a22/a21))+2*(phi-1)*r[-1]*(1-2*hx[-1]*a42
            /a41) # scalar
        Ba9 = 2*(1-theta)*s[-1] # scalar
        Bb9 = 2*(1-phi)*r[-1] # scalar


        # Create diagonals of A-matrix (for solving A*T_new = B*T_old + q)
        zer = np.zeros(np.shape(Ac3)) # M-1 long array of zeros

        a_diag = np.r_[np.reshape(np.c_[Aa4,np.transpose([Aa1]*(M-1)),Aa5],(N-1)*(M+1)),Aa7
            ,[Aa3]*(M-1),Aa9]
        b_diag = np.r_[Ab2,Ab8,np.tile(np.r_[0,Ab1,Ab5],N-1),0,Ab3,Ab9]
        c_diag = np.r_[Ac6,Ac2,Ac8]
        for j in range(1,N):
              Ac1 = 1+2*theta*s[j-1]+2*phi*r
              c_diag = np.r_[c_diag,Ac4[j-1],Ac1,Ac5[j-1]]
        c_diag = np.r_[c_diag,Ac7,Ac3,Ac9]
        d_diag = np.r_[Ad6,Ad2,np.tile(np.r_[0,Ad4,Ad1],N-1),0,Ad7,Ad3]
        e_diag = np.r_[Ae6,[Ae2]*(M-1),Ae8,np.reshape(np.c_[Ae4,np.transpose([Ae1]*(M-1)),Ae5],(M+1)*(N-1))]

        diagonals = [ a_diag , b_diag , c_diag, d_diag, e_diag ]

        # Create A-matrix (for solving A*T_new = B*T_old + q)
        A = sp.diags(diagonals,[-(M+1),-1,0,1,M+1]).toarray() # make A directly as a sparse matrix

        # Create diagonals of B-matrix (for solving A*T_new = B*T_old + q)
        zer = np.zeros(np.shape(Bc3)) # M-1 long array of zeros
        a_diag = np.r_[np.reshape(np.c_[Ba4,np.transpose([Ba1]*(M-1)),Ba5],(N-1)*(M+1)),Ba7,[Ba3]*(M-1),Ba9]
        b_diag = np.r_[Bb2,Bb8,np.tile(np.r_[0,Bb1,Bb5],N-1),0,Bb3,Bb9]
        c_diag = np.r_[Bc6,Bc2,Bc8]
        for j in range(1,N):
              Bc1 = 1+2*theta*s[j-1]+2*phi*r
              c_diag = np.r_[c_diag,Bc4[j-1],Bc1,Bc5[j-1]]
        c_diag = np.r_[c_diag,Bc7,Bc3,Bc9]
        d_diag = np.r_[Bd6,Bd2,np.tile(np.r_[0,Bd4,Bd1],N-1),0,Bd7,Bd3]
        e_diag = np.r_[Be6,[Be2]*(M-1),Be8,np.reshape(np.c_[Be4,np.transpose([Be1]*(M-1)),Be5],(M+1)*(N-1))]

        diagonals = [ a_diag , b_diag , c_diag, d_diag, e_diag ]

        # Create B-matrix (for solving A*T_new = B*T_old + q)
        B = sp.diags(diagonals,[-(M+1),-1,0,1,M+1]).toarray() # make B directly as a sparse matrix

        # Create q (for solving A*T_new = B*T_old + q)
        q = h_dt*eavg # source term
        q[:M+1]=q[:M+1] - 2*s[0]*hz[0]*g1/a11 # accounts for z=0 boundary
        q[-M:]=q[-M:] + 2*s[-1]*hz[-1]*g2/a21 # accounts for z=L boundary
        q[np.arange(0,(M+1)*N+1,(M+1))] = q[np.arange(0,(M+1)*N+1,(M+1))] - 2*r[0]*hx[0]*g3/a31 # accounts for x=0 boundary
        q[np.arange(M+1,(M+1)*(N+1)+1,(M+1))-1] = q[np.arange(M+1,(M+1)*(N+1)+1,(M+1))-1] + 2*r[-1]*hx[-1]*g4/a41 # accounts for x=H boundary

        # Solve equation A*T_new = B*T_old + q
        T_new = linalg.solve(A,dot(B,T_old)+q)
        return T_new


 def second_try_nondiml(T_old,hx,hz,eavg,h_dt,theta,phi,bc,h,T_init,k,rho,c_p):
        """second_try_diml(T_old,h,c_p,rho,kappa,eavg,h_dt,sig,theta,bc): Implements a
            single timestep of the theta-scheme for solving the heat equation using the
            finite difference method.

        Inputs:
               T_old  The initial temperature field. A vector (array) of length (N+1)*(M+1).
               hx  The differences between x-values [m]. A vector (array) of length N.
               hz  The differences between z-values [m]. A vector (array) of length M.
               eavg  The power dissipated into each point of the simulated domain. A vector
                   (array) of length (N+1)(M+1).
               h_dt  The length of the thermal timestep [sec]. A scalar.
               theta  The parameter for the theta-method in the z-direction. A scalar that
                   varies from 0 to 1. Take theta = 0 for a fully explicit method, theta = 1
                    for a fully implicit method, and theta = 0.5 for a Crank-Nicolson
                   method.
               phi  The parameter for the theta-method in the x-direction. A scalar that
                   varies from 0 to 1. Take phi = 0 for a fully explicit method, phi = 1 for
                    a fully implicit method, and phi = 0.5 for a Crank-Nicolson method.
               bc  The type of boundary condition to use at the left- and right-hand
                   endpoints. A string that takes either the value 'ins' (for the insulated
                   boundary conditions; i.e., homogeneous Neumann), 'fix' (for the fixed
                   temperature boundary conditions; i.e., inhomogeneous Dirichlet), or 'rad'
                    (for the radiative boundary conditions; i.e., third-kind or mixed
                   conditions).
               h  Coefficient for radiative BC.
        Outputs:
               T_new  The temperature field after a single timestep. A vector (array) of
                   length n.
        """

        # Useful parameters
        mu0=pi*4e-7 # permeability of free space [N/A^2]
        c = 299792458.0 # speed of light [m/s]
        hx_sq = hx[1:]*hx[:-1]
        hz_sq = hz[1:]*hz[:-1]
        s = h_dt/(hz_sq)
        r = h_dt/(hx_sq)
```

```
    N = np.size(hz)
    M = np.size(hx)

    # Implement the boundary conditions--choice of Dirichlet, Neumann, or radiative. Each BC can be written in the mixed formulation a1*T_x(0,t) + a2*T(0,t) = g1(t) and a3*T_x(L,t) + a4*T(L,t) = g2(t), with some or other parameters being zero, strategically
    if bc == 'ins': # insulated boundary on all four walls: homogeneous Neumann condition
        a11 = 1.0
        a12 = 0
        g1 = 0
        a21 = 1.0
        a22 = 0
        g2 = 0
        a31 = 1.0
        a32 = 0
        g3 = 0
        a41 = 1.0
        a42 = 0
        g4 = 0
    elif bc == 'rad': # radiative BC on all four walls
        a11 = 1.0
        a12 = -h
        g1 = -h*T_init
        a21 = 1.0
        a22 = -h
        g2 = -h*T_init
        a31 = 1.0
        a32 = -h
        g3 = -h*T_init
        a41 = 1.0
        a42 = -h
        g4 = -h*T_init
    elif bc == 'fix': # fixed temperature on all four walls
        #these are dummy values for now, and boundary rows will be changed after matrices created
        a11 = 1.0
        a12 = 0
        g1 = 0
        a21 = 1.0
```

```
            a22 = 0
            g2 = 0
            a31 = 1.0
            a32 = 0
            g3 = 0
            a41 = 1.0
            a42 = 0
            g4 = 0
    else: # throw an error if bc is neither of those strings
            print "The input variable 'bc' must be either the string 'fix', 'ins', or '
                rad'"
            input()
            import sys
            sys.exit(1)

    # Create diagonals of A-matrix and B-matrix (for solving A*T_new = B*T_old + v*q)
    Aa_diag = Ba_diag = np.array([])

    Ab_diag = Ad_diag = -theta*s
    Ab_diag = np.r_[Ab_diag, -2*theta*s[-1]]
    Ad_diag = np.r_[-2*theta*s[0], Ad_diag]

    Bb_diag = Bd_diag = (1-theta)*s
    Bb_diag = np.r_[Bb_diag, 2*(1-theta)*s[-1]]
    Bd_diag = np.r_[2*(1-theta)*s[0], Bd_diag]

    Ac_diag = np.r_[1+2*theta*s[0]*(1-(2*hz[0]*a12/a11))+2*phi*r[0]*(1-(2*hx[0]*a32/a31
        )) , 1+2*theta*s+[2*phi*r[0]*(1-(2*hx[0]*a32/a31))]*(N-1) , 1+2*theta*s
        [-1]*(1+(2*hz[-1]*a22/a21))+2*phi*r[0]*(1-(2*hx[0]*a32/a31))]
    Bc_diag = np.r_[1+2*(theta-1)*s[0]*(1-(2*hz[0]*a12/a11))+2*(phi-1)*r[0]*(1-(2*hx
        [0]*a32/a31)) , 1+2*(theta-1)*s+[2*(phi-1)*r[0]*(1-(2*hx[0]*a32/a31))]*(N-1) ,
         1+2*(theta-1)*s[-1]*(1+(2*hz[-1]*a22/a21))+2*(phi-1)*r[0]*(1-(2*hx[0]*a32/
        a31))]

    Ae_diag = np.r_[ [-2*phi*r[0]]*(N+1)]
    Be_diag = np.r_[ [2*(1-phi)*r[0]]*(N+1)]

    for k in range(1,M):
            Aa_diag = np.r_[Aa_diag, [-phi*r[k-1]]*(N+1)]
            Ba_diag = np.r_[Ba_diag, [(1.0-phi)*r[k-1]]*(N+1)]

```

```
                Ab_diag = np.r_[Ab_diag, 0.0, -theta*s , -2.0*theta*s[-1] ]
                Bb_diag = np.r_[Bb_diag, 0.0, (1.0-theta)*s , 2.0*(1.0-theta)*s[-1] ]

                Ac_diag = np.r_[Ac_diag, 1+2.0*phi*r[k-1]+2*theta*s[0]*(1-(2*hz[0]*a12/a11)) , 1+2.0*phi*r[k-1]+2*theta*s , 1+2.0*phi*r[k-1]+2*theta*s[-1]*(1+(2*hz[-1]*a22/a21)) ]
                Bc_diag = np.r_[Bc_diag, 1+2.0*(phi-1)*r[k-1]+2*(theta-1)*s[0]*(1-(2*hz[0]*a12/a11)) , 1+2.0*(phi-1)*r[k-1]+2*(theta-1)*s , 1+2.0*(phi-1)*r[k-1]+2*(theta-1)*s[-1]*(1+(2*hz[-1]*a22/a21)) ]

                Ad_diag = np.r_[Ad_diag, 0.0, -2.0*theta*s[0] , -theta*s ]
                Bd_diag = np.r_[Bd_diag, 0.0, 2.0*(1.0-theta)*s[0] , (1.0-theta)*s]

                Ae_diag = np.r_[Ae_diag, [-phi*r[k-1]]*(N+1)]
                Be_diag = np.r_[Be_diag, [(1.0-phi)*r[k-1]]*(N+1) ]

        Aa_diag = np.r_[Aa_diag, [-2*phi*r[-1]]*(N+1) ]
        Ba_diag = np.r_[Ba_diag, [2*(1-phi)*r[-1]]*(N+1) ]

        Ab_diag = np.r_[Ab_diag, 0.0, -theta*s , -2*theta*s[-1] ]
        Bb_diag = np.r_[Bb_diag, 0.0, (1.0-theta)*s, 2*(1-theta)*s[-1] ]

        Ac_diag = np.r_[Ac_diag, 1.0+2.0*theta*s[0]*(1.0-(2*hz[0]*a12/a11))+2.0*phi*r[-1]*(1.0+(2.0*hx[-1]*a42/a41)) , 1.0+2.0*theta*s+2*phi*r[-1]*(1+(2*hx[-1]*a42/a41)) , 1.0+2*theta*s[-1]*(1.0+(2*hz[-1]*a22/a21))+2.0*phi*r[-1]*(1.0+(2.0*hx[-1]*a42/a41)) ]
        Bc_diag = np.r_[Bc_diag, 1.0+2.0*(theta-1)*s[0]*(1.0-(2*hz[0]*a12/a11))+2.0*(phi-1)*r[-1]*(1.0+(2.0*hx[-1]*a42/a41)) , 1.0+2.0*(theta-1)*s+2*(phi-1)*r[-1]*(1+(2*hx[-1]*a42/a41)) , 1.0+2.0*(theta-1.0)*s[-1]*(1.0+(2.0*hz[-1]*a22/a21))+2.0*(phi-1)*r[-1]*(1.0+(2.0*hx[-1]*a42/a41)) ]

        Ad_diag = np.r_[Ad_diag, 0.0, -2.0*theta*s[0] , -theta*s ]
        Bd_diag = np.r_[Bd_diag, 0.0, 2.0*(1.0-theta)*s[0] , s*(1-theta) ]

        Adiagonals = [ Aa_diag , Ab_diag , Ac_diag, Ad_diag, Ae_diag ]
        Bdiagonals = [ Ba_diag , Bb_diag , Bc_diag, Bd_diag, Be_diag ]

        # Make A and B directly as sparse matrices
        A = sp.diags(Adiagonals,[-(N+1),-1,0,1,N+1]).toarray() # make A directly as a sparse matrix
        B = sp.diags(Bdiagonals,[-(N+1),-1,0,1,N+1]).toarray() # make B directly as a
            sparse matrix

        # Create v (for solving A*E_new = B*E_old + v*E_older)
        v = eavg*h_dt

        v[0] = v[0] -2*r[0]*hx[0]*g3/a31- 2*s[0]*hz[0]*g1/a11
        v[1:N] = v[1:N]-2*r[0]*hx[0]*g3/a31
        v[N] = v[N] - 2*r[0]*hx[0]*g3/a31+ 2*s[-1]*hz[-1]*g2/a21
        for k in range(1,M):
              v[k*(N+1)] = v[k*(N+1)] - 2*s[0]*hz[0]*g1/a11
              v[(k+1)*(N+1)-1] = v[(k+1)*(N+1)-1] + 2*s[-1]*hz[-1]*g2/a21
        v[-(N+1)] = v[-(N+1)] +2*r[-1]*hx[-1]*g4/a41 - 2*s[-1]*hz[0]*g1/a11
        v[-N:-1] = v[-N:-1] + 2*r[-1]*hx[-1]*g4/a41
        v[-1] = v[-1] + 2*r[-1]*hx[-1]*g4/a41+ 2*s[-1]*hz[-1]*g2/a21

        if bc == 'fix':
              for j in range(0,N):
                    A[j,:] = B[j,:] = 0
                    A[j,j] = 1
                    v[j] = T_init

                    A[-(j+1),:] = B[-(j+1),:] = 0
                    A[-(j+1),-(j+1)] = 1
                    v[-(j+1)] = T_init
              for k in range(1,M):
                    A[k*(N+1),:] = B[k*(N+1),:] = 0
                    A[k*(N+1),k*(N+1)] = 1
                    v[k*(N+1)] = T_init

                    A[(k+1)*(N+1)-1,:] = B[(k+1)*(N+1)-1,:] = 0
                    A[(k+1)*(N+1)-1,(k+1)*(N+1)-1] = 1
                    v[(k+1)*(N+1)-1] = T_init

        # Solve equation A*E_new = B*E_old + v*E_older
        T_new = linalg.solve(A,np.dot(B,T_old)+v)

        return  T_new

def second_try_diml(T_old,hx,hz,eavg,h_dt,theta,phi,bc,h,T_init,k,rho,c_p):
        """second_try_diml(T_old,h,c_p,rho,kappa,eavg,h_dt,sig,theta,bc): Implements a
            single timestep of the theta-scheme for solving the heat equation using the
            finite difference method.

        Inputs:
              T_old  The initial temperature field. A vector (array) of length (N+1)*(M+1).
              hx  The differences between x-values [m]. A vector (array) of length N.
              hz  The differences between z-values [m]. A vector (array) of length M.
              eavg  The power dissipated into each point of the simulated domain. A vector
                  (array) of length (N+1)(M+1).
              h_dt  The length of the thermal timestep [sec]. A scalar.
              theta  The parameter for the theta-method in the z-direction. A scalar that
                  varies from 0 to 1. Take theta = 0 for a fully explicit method, theta = 1
                   for a fully implicit method, and theta = 0.5 for a Crank-Nicolson
                  method.
              phi  The parameter for the theta-method in the x-direction. A scalar that
                  varies from 0 to 1. Take phi = 0 for a fully explicit method, phi = 1 for
                   a fully implicit method, and phi = 0.5 for a Crank-Nicolson method.
              bc  The type of boundary condition to use at the left- and right-hand
                  endpoints. A string that takes either the value 'ins' (for the insulated
                  boundary conditions; i.e., homogeneous Neumann), 'fix' (for the fixed
                  temperature boundary conditions; i.e., inhomogeneous Dirichlet), or 'rad'
                   (for the radiative boundary conditions; i.e., third-kind or mixed
                  conditions).
              h  Coefficient for radiative BC.
        Outputs:
              T_new  The temperature field after a single timestep. A vector (array) of
                  length n.
        """

        # Useful parameters
        mu0=pi*4e-7 # permeability of free space [N/A^2]
        c = 299792458.0 # speed of light [m/s]
        hx_sq = hx[1:]*hx[:-1]
        hz_sq = hz[1:]*hz[:-1]
        s = h_dt/(hz_sq)
        r = h_dt/(hx_sq)
        N = np.size(hz)
        M = np.size(hx)
        s,r = np.meshgrid(s,r)
        kappa = k/rho*c_p
        kappa = np.reshape(kappa,(M+1,N+1))

        s = s*kappa[1:-1,1:-1]
        r = r*kappa[1:-1,1:-1]

        # Implement the boundary conditions--choice of Dirichlet, Neumann, or radiative.
            Each BC can be written in the mixed formulation a1*T_x(0,t) + a2*T(0,t) = g1(t)
             and a3*T_x(L,t) + a4*T(L,t) = g2(t), with some or other parameters being zero,
             strategically
        if bc == 'ins': # insulated boundary on all four walls: homogeneous Neumann
            condition
               a11 = 1.0
               a12 = 0
               g1 = 0
               a21 = 1.0
               a22 = 0
               g2 = 0
               a31 = 1.0
               a32 = 0
               g3 = 0
               a41 = 1.0
               a42 = 0
               g4 = 0
        elif bc == 'rad': # radiative BC on all four walls
               a11 = 1.0
               a12 = h
               g1 = h*T_init
               a21 = 1.0
               a22 = -h
               g2 = -h*T_init
               a31 = 1.0
               a32 = -h
               g3 = -h*T_init
               a41 = 1.0
               a42 = h
               g4 = h*T_init
        elif bc == 'fix': # fixed temperature on all four walls
               #these are dummy values for now, and boundary rows will be changed after
                   matrices created
               a11 = 1.0
               a12 = 0
```

```
            g1 = 0
            a21 = 1.0
            a22 = 0
            g2 = 0
            a31 = 1.0
            a32 = 0
            g3 = 0
            a41 = 1.0
            a42 = 0
            g4 = 0
    else: # throw an error if bc is neither of those strings
            print "The input variable 'bc' must be either the string 'fix', 'ins', or '
                rad'"
            input()
            import sys
            sys.exit(1)

    # Create diagonals of A-matrix and B-matrix (for solving A*T_new = B*T_old + v*q)
    Aa_diag = Ba_diag = np.array([])

    Ab_diag = Ad_diag = -theta*s[0,:]
    Ab_diag = np.r_[Ab_diag, -2*theta*s[0,-1]]
    Ad_diag = np.r_[-2*theta*s[0,0], Ad_diag]

    Bb_diag = Bd_diag = (1-theta)*s[0,:]
    Bb_diag = np.r_[Bb_diag, 2*(1-theta)*s[0,-1]]
    Bd_diag = np.r_[2*(1-theta)*s[0,0], Bd_diag]

    Ac_diag = np.r_[1+2*theta*s[0,0]*(1-(2*hz[0]*a12/a11))+2*phi*r[0,0]*(1-(2*hx[0]*a32
        /a31)) , 1+2*theta*s[0,:]+2*phi*r[0,:]*(1-(2*hx[0]*a32/a31)) , 1+2*theta*s
        [0,-1]*(1+(2*hz[-1]*a22/a21))+2*phi*r[0,-1]*(1-(2*hx[0]*a32/a31))]
    Bc_diag = np.r_[1+2*(theta-1)*s[0,0]*(1-(2*hz[0]*a12/a11))+2*(phi-1)*r[0,0]*(1-(2*
        hx[0]*a32/a31)) , 1+2*(theta-1)*s[0,:]+2*(phi-1)*r[0,:]*(1-(2*hx[0]*a32/a31))
        , 1+2*(theta-1)*s[0,-1]*(1+(2*hz[-1]*a22/a21))+2*(phi-1)*r[0,-1]*(1-(2*hx[0]*
        a32/a31))]

    Ae_diag = np.r_[ -2*phi*r[0,0], -2*phi*r[0,:], -2*phi*r[0,-1] ]
    Be_diag = np.r_[ 2*(1-phi)*r[0,0], 2*(1-phi)*r[0,:] , 2*(1-phi)*r[0,-1] ]

    for k in range(1,M):
            Aa_diag = np.r_[Aa_diag, -phi*r[k-1,0], -phi*r[k-1,:], -phi*r[k-1,-1] ]
           Ba_diag = np.r_[Ba_diag, (1.0-phi)*r[k-1,0], (1.0-phi)*r[k-1,:], (1.0-phi)*r[k-1,-1] ]

           Ab_diag = np.r_[Ab_diag, 0.0, -theta*s[k-1,:] , -2.0*theta*s[k-1,-1] ]
           Bb_diag = np.r_[Bb_diag, 0.0, (1.0-theta)*s[k-1,:] , 2.0*(1.0-theta)*s[k-1,-1] ]

           Ac_diag = np.r_[Ac_diag, 1+2.0*phi*r[k-1,0]+2*theta*s[k-1,0]*(1-(2*hz[0]*a12/a11)) , 1+2.0*phi*r[k-1,:]+2*theta*s[k-1,:] , 1+2.0*phi*r[k-1,-1]+2*theta*s[k-1,-1]*(1+(2*hz[-1]*a22/a21)) ]
           Bc_diag = np.r_[Bc_diag, 1+2.0*(phi-1)*r[k-1,0]+2*(theta-1)*s[k-1,0]*(1-(2*hz[0]*a12/a11)) , 1+2.0*(phi-1)*r[k-1,:]+2*(theta-1)*s[k-1,:] , 1+2.0*(phi-1)*r[k-1,-1]+2*(theta-1)*s[k-1,-1]*(1+(2*hz[-1]*a22/a21)) ]

           Ad_diag = np.r_[Ad_diag, 0.0, -2.0*theta*s[k-1,0] , -theta*s[k-1,:] ]
           Bd_diag = np.r_[Bd_diag, 0.0, 2.0*(1.0-theta)*s[k-1,0] , (1.0-theta)*s[k-1,:]]

           Ae_diag = np.r_[Ae_diag, -phi*r[k-1,0], -phi*r[k-1,:], -phi*r[k-1,-1] ]
           Be_diag = np.r_[Be_diag, (1.0-phi)*r[k-1,0], (1.0-phi)*r[k-1,:], (1.0-phi)*r[k-1,-1] ]

    Aa_diag = np.r_[Aa_diag, -2.0*phi*r[-1,0], -2.0*phi*r[-1,:], -2.0*phi*r[-1,-1] ]
    Ba_diag = np.r_[Ba_diag, 2.0*(1.0-phi)*r[-1,0], 2.0*(1.0-phi)*r[-1,:] , 2.0*(1.0-phi)*r[-1,-1] ]

    Ab_diag = np.r_[Ab_diag, 0.0, -theta*s[-1,:] , -2.0*theta*s[-1,-1] ]
    Bb_diag = np.r_[Bb_diag, 0.0, (1.0-theta)*s[-1,:], 2.0*(1.0-theta)*s[-1,-1] ]

    Ac_diag = np.r_[Ac_diag, 1.0+2.0*theta*s[-1,0]*(1.0-(2*hz[0]*a12/a11))+2.0*phi*r[-1,0]*(1.0+(2.0*hx[-1]*a42/a41)) , 1.0+2.0*theta*s[-1,:]+2*phi*r[-1,:]*(1+(2*hx[-1]*a42/a41)) , 1.0+2*theta*s[-1,-1]*(1.0+(2*hz[-1]*a22/a21))+2.0*phi*r[-1,-1]*(1.0+(2.0*hx[-1]*a42/a41)) ]
    Bc_diag = np.r_[Bc_diag, 1.0+2.0*(theta-1)*s[-1,0]*(1.0-(2*hz[0]*a12/a11))+2.0*(phi-1)*r[-1,0]*(1.0+(2.0*hx[-1]*a42/a41)) , 1.0+2.0*(theta-1)*s[-1,:]+2*(phi-1)*r[-1,:]*(1+(2*hx[-1]*a42/a41)) , 1.0+2.0*(theta-1.0)*s[-1,-1]*(1.0+(2.0*hz[-1]*a22/a21))+2.0*(phi-1)*r[-1,-1]*(1.0+(2.0*hx[-1]*a42/a41)) ]

    Ad_diag = np.r_[Ad_diag, 0.0, -2.0*theta*s[-1,0], -theta*s[-1,:] ]
    Bd_diag = np.r_[Bd_diag, 0.0, 2.0*(1.0-theta)*s[-1,0] , s[-1,:]*(1-theta) ]

    Adiagonals = [ Aa_diag , Ab_diag , Ac_diag, Ad_diag, Ae_diag ]
    Bdiagonals = [ Ba_diag , Bb_diag , Bc_diag, Bd_diag, Be_diag ]

    # Make A and B directly as sparse matrices
    A = sp.diags(Adiagonals,[-(N+1),-1,0,1,N+1]).toarray() # make A directly as a
           sparse matrix
    B = sp.diags(Bdiagonals,[-(N+1),-1,0,1,N+1]).toarray() # make B directly as a
           sparse matrix

    # Create v (for solving A*E_new = B*E_old + v*E_older)
    v = eavg*h_dt/(c_p*rho)

# print M
# print N

# print (M+1)*(N+1)-1
# print M*(N+1)-1

    v[0] = v[0] -2*r[0,0]*hx[0]*g3/a31- 2*s[0,0]*hz[0]*g1/a11
# print "v[0] -g3 -g1"
    v[1:N] = v[1:N]-2*r[0,:]*hx[0]*g3/a31
# print "v[1:"+str(N)+"] -g3"
    v[N] = v[N] - 2*r[0,-1]*hx[0]*g3/a31+ 2*s[0,-1]*hz[-1]*g2/a21
# print "v["+str(N)+"] -g3 + g2"
    for k in range(1,M):
            v[k*(N+1)] = v[k*(N+1)] - 2*s[k-1,0]*hz[0]*g1/a11
#  print "v["+str(k*(N+1))+"] - g1"
            v[(k+1)*(N+1)-1] = v[(k+1)*(N+1)-1] + 2*s[k-1,-1]*hz[-1]*g2/a21
#  print "v["+str((k+1)*(N+1)-1)+"] + g2"
    v[-(N+1)] = v[-(N+1)] +2*r[-1,0]*hx[-1]*g4/a41 - 2*s[-1,0]*hz[0]*g1/a11
# print "v["+str(np.size(v)-N)+"] +g4 -g1"
    v[-N:-1] = v[-N:-1]+2*r[-1,:]*hx[-1]*g4/a41
# print "v["+str(np.size(v)-N+1)+":"+str(np.size(v)-1)+"] +g4"
    v[-1] = v[-1] + 2*r[-1,-1]*hx[-1]*g4/a41+ 2*s[-1,-1]*hz[-1]*g2/a21
# print "v["+str(np.size(v)-1)+"] +g4 +g2"

    if bc == 'fix':
            for j in range(0,N):
                    A[j,:] = B[j,:] = 0
                    A[j,j] = 1
```

```
                    v[j] = T_init

                    A[-(j+1),:] = B[-(j+1),:] = 0
                    A[-(j+1),-(j+1)] = 1
                    v[-(j+1)] = T_init
            for k in range(1,M):
                    A[k*(N+1),:] = B[k*(N+1),:] = 0
                    A[k*(N+1),k*(N+1)] = 1
                    v[k*(N+1)] = T_init

                    A[(k+1)*(N+1)-1,:] = B[(k+1)*(N+1)-1,:] = 0
                    A[(k+1)*(N+1)-1,(k+1)*(N+1)-1] = 1
                    v[(k+1)*(N+1)-1] = T_init

        # Solve equation A*E_new = B*E_old + v*E_older
        T_new = linalg.solve(A,np.dot(B,T_old)+v)

        return  T_new
```

E.4 MATLAB Implementation of the Finite Difference Method for the Two-Dimensional Heat Equation

```
function temp_new=thermsolve2_fd(temp,hx,hy,Nx,Ny,X,Y,cp,rho,k,eavg,dt,sigma,time)

omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves at 2.45GHz
eps0 = 8.8541878176e-12; %permittivity of free space

k=(reshape(k',1,[]));

% BC: all walls insulated and fixed at room temperature (the initial temperature)

a_up=-k(1:end-1).*[zeros(1,Nx), repmat([0,(1./(hx(2:end).^2)),0],[1 Ny-2]), zeros(1,Nx-1)
    ];
a_lo=-k(2:end).*[zeros(1,Nx-1), repmat([0,(1./(hx(2:end).^2)),0],[1 Ny-2]), zeros(1,Nx)];

b_up=-k(1:end-Nx).*[zeros(1,Nx), reshape([zeros(size(hy(2:end)));repmat((1./(hy(2:end))
    .^2),[Nx-2,1]);zeros(size(hy(2:end)))],1,[]) ];
b_lo=-k(Nx+1:end).*[reshape([zeros(size(hy(1:end-1)));repmat((1./(hy(2:end)).^2),[Nx
    -2,1]);zeros(size(hy(1:end-1)))],1,[]), zeros(1,Nx) ];

q_int=k(Nx+1:end-Nx).*repmat([0,(2./(hx(2:end)).^2),0],[1 Ny-2])...
```

```
    +k(Nx+1:end−Nx).*reshape([zeros(size(hy(2:end)));repmat(2./(hy(2:end)).^2,[Nx−2,1]);
        zeros(size(hy(2:end)))],1,[])...
    +(reshape(rho(2:end−1,:)',1,[])).*(reshape(cp(2:end−1,:)',1,[]))/dt;

q=[ones(1,Nx),q_int,ones(1,Nx)];
q((1:Ny−1)*Nx+1)=1; % temperature is fixed at room temp on left−hand wall
q((1:Ny)*Nx)=1; % temperature is fixed at room temp on right−hand wall

A=diag(q,0) + diag(a_up,1) + diag(a_lo,−1) + diag(b_up,Nx) + diag(b_lo,−Nx);
A=sparse(A);
%A=spdiags([b_lo a_lo q a_up b_up],[−Nx,−1,0,1,Nx],length(q),length(q));

s=(reshape(rho',1,[])).*(reshape(cp',1,[]))/dt;

%%Source Term

Q = 0.5*omega*eps0*(reshape(sigma',1,[]))'.*eavg;

Q(1:Nx)=0; % temp is fixed on top wall
Q((1:Ny−1)*Nx+1)=0; % temp is fixed on left−hand wall
Q((1:Ny)*Nx)=0; % temp is fixed on right−hand wall
Q(end−Nx:end)=0; % temp is fixed on bottom wall
s(1:Nx)=1; % temp is fixed on top wall
s((1:Ny−1)*Nx+1)=1; % temp is fixed on left−hand wall
s((1:Ny)*Nx)=1; % temp is fixed on right−hand wall
s(end−Nx:end)=1; % temp is fixed on bottom wall

%%Solve equation
temp_new = A\(s'.*temp + Q); % Solve A*T^(n+1) = s*T^n + Q

% figure(2); hold off; surf(X*100,flipud(Y*100),(reshape(temp_new,Nx,Ny))'−273); view
    (0,90); colorbar;
% title(strcat('Temperature distribution at t=',num2str(time+dt,'%11.3g'),' seconds'));
% xlabel('Length L (x−dir) [cm]'); ylabel('Height H (y−dir) [cm]'); zlabel('Temperature [C
    ]');



end
```

Appendix F

Computer Implementation in python and MATLAB of the Mechanical Solvers for the 1D and 2D Sintering Problems

F.1 Computer Implementation in python of the Exponential Integrals Method for Finding Θ Values

```python
#!/usr/bin/python
#
# Reproduces Figure 5.1 from Abramowitz & Stegun for the thesis;
# Computes Theta values for a constant heating rate trial using exponential integrals; this

from scipy import special # exponential integral functions
from scipy import integrate # cumulative trapezoidal integration
#from numpy import linalg as la # for computing vector norm
import numpy as np # for linspace and regular exponential function
import matplotlib.pyplot as plt # for plotting theta-values (and for reproducing
    AbramowitzStegun figure for thesis)
from matplotlib2tikz import save as tikz_save # for getting a file with tikz data to plot
    directly in thesis
import timeit # for computing speeds of cumtrapz vs expint

# Recreate Figure from AbramowitzStegun
x = np.linspace(0,1.5,100)
x = x[1:] # *positive* x-vals, not zero
yi = special.expi(x) # Ei(x)
```

```
y1 = special.exp1(x) # E_1(x)

plt.figure(1)
plt.clf()
plt.plot(x,yi,label=r'$y=$Ei$(x)$')
plt.plot(x,y1,label=r'$y=$E$_1(x)$')
plt.plot(x,np.zeros(np.shape(x)),'k-') # x-axis (this is the ugliest way of putting the x
    -axis here, but 'spines' isn't a recognized module in my version of matplotlib--why?)
plt.grid()
plt.legend(loc='lower right')
plt.xlabel('$x$')
plt.ylabel('$y$')
plt.title(r'Exponential integrals $y =$ Ei$(x)$ and $y =$ E$_1(x)$ computed with \texttt{
    python}')
plt.savefig('python_ei.png') # saves the plot as .png without displaying it
tikz_save('python_ei.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',
    show_info = False ) # save data as .tex so plot can be recreated directly in LaTeX

# Test speed and accuracy of computing Theta values
times = np.linspace(500,1000) # time from 500 to 1000 seconds
alpha = 5 # 5 degC/min
temps = 800+alpha*times # start at 800 degC
Q = 650000 # J/mol (for example--this is approximately the correct value for zirconia)
R = 8.314459848 # ideal gas constant [J/(mol*K)]
ntimes = 100000

# Compute using exponential integral function
def compute_expi():
      return (1/alpha)*special.expi(-Q/(R*alpha*times))
expitime=timeit.timeit(stmt="compute_expi()",number=ntimes,setup="from __main__ import
    compute_expi")
eithetas = compute_expi()

# Compute using cumulative trapezoidal integration
def compute_cumtrapz():
      integrands = np.exp(-Q/(R*temps))/temps # the integrand values
      return integrate.cumtrapz(integrands,times) # built-in cumulative trapezoidal
          integration
cumtrapztime = timeit.timeit(stmt="compute_cumtrapz()",number=ntimes,setup="from __main__
    import compute_cumtrapz")
ctthetas = compute_cumtrapz()
```

```

print "\nThe Frobenius norm of the difference between Theta−values computed using
    exponential integral function and those computed using cumulative trapezoidal
    integration is "+str(np.linalg.norm(ctthetas−eithetas[1:]))+" ."

print "\nExponential integral function method finds theta values in "+str(100*expitime/
    cumtrapztime)+" percent of the time that trapezoidal integration does (average over "+
    str(ntimes)+" simulations)."

#plt.figure(2)
#plt.plot(times[1:],eithetas[1:],'bs−',label='Exponential integral method')
#plt.plot(times[1:],ctthetas,'ro−',label='Trapezoidal integral method')
#plt.legend(loc="upper left")
#plt.xlabel('Times [sec]')
#plt.ylabel('$\Theta$−values')
#plt.title("Values of $\Theta$ computed for constant heating rate")
#plt.show()
```

F.2 Computer Implementation in `python` of the Master Sintering Curve Method

```
#!/usr/bin/python

# Computes coefficients for various representations of the sigmoid curve in MSC method

# This code requires python 2.7, along with the scipy and numpy packages.

# Import necessary packages
import numpy as np # numpy: 'np' prefix (because we use sizes of arrays and exp and log)
from scipy.optimize import curve_fit # because we use Levenberg−Marquardt
from scipy.optimize import minimize # because we use Nelder−Mead
import matplotlib.pyplot as plt # for plotting final sigmoid curve and input data
from matplotlib2tikz import save as tikz_save # for getting a file with tikz data to plot
    directly in thesis
from scipy import integrate # so we can use cumulative trapezoidal integration
import sys # for exiting after errors
import itertools # for plotting with different colors for markers of different experiments

def find_lnthetas(times,temps,Q):
       '''lnthetas = find_lnthetas(times,temps,Q):

        Integrates using [Su & Johnson] formula to find ln(Theta) values.
        Uses trapezoidal approximation.

        Inputs: times  the times at which temperature and density measurements
                           were taken experimentally. A vector (array) of length N.
                           Units of seconds.
               temps  the temperatures corresponding to the times in 'times'
                           input. A vector (array) of length N. Units of degC.
               Q  activation energy of substance (can be determined from
                           separate optimization routine). A scalar. Units of J/mol
                               .

        Outputs: lnthetas ln(Theta(t,T(t)) values. A vector (array) of length N-1.
                           Units of ln(s/K).
    '''
    N = np.size(times)
    R = 8.314459848 # ideal gas constant [J/(mol*K)]

    # Perform integration to find theta values using trapezoidal method

    dts = times[1:]-times[:-1] # lengths of time intervals (has length N-1)
    integrands = np.exp(-Q/(R*temps))/temps # the integrand values at each of the nodes
         (length N)
    areas = 0.5*dts*(integrands[1:]+integrands[:-1]) # areas of individual trapezoids
        under the curve
    thetas = np.r_[areas[0],[0]*(N-2)] # values of the integral from 0 to t_i, to be
        filled below
    for i in xrange(1,N-1):
        thetas[i] = thetas[i-1] + areas[i] # sums the areas of all trapezoids from
            first to current

    #thetas = integrate.cumtrapz(integrands,times) # built-in cumulative trapezoidal
        integration

    return np.log(thetas)

def plot_sigmoid(lnthetas,rhos,expnames,rhofun,savestring,funstring,titlestring,show):
    '''plot_sigmoid(lnthetas,rhos,rhofun,savestring,funstring,titlestring,show):

        Plots the sigmoid curve and raw data points.

        Inputs: lnthetas Generated by find_lnthetas. A vector (array) of length N.
                           Units of ln(s/K).
               rhos  the relative density values corresponding to the times in 'times'
                           input. A vector (array) of length N. Units of 1.
               expnames the titles of each experiment corresponding to a column of
                           times/temps/rhos matrices. A length-M list of strings.
               rhofun  function handle outputting rho for a given lntheta value.
               savestring filepaths to use when saving plots and data. A string.
               funstring function with optimal parameters. A string.
               titlestring title of graph. A string.
               show  whether to show the graph, or just save it

        Outputs: lnthetas ln(Theta(t,T(t)) values. A vector (array) of length N-1.
                           Units of ln(s/K).
   '''
    xs = np.linspace(lnthetas[0],lnthetas[-1]) # input lnthetas for plotting optimal fit sigmoid
    ys = rhofun(xs) # output rhos for plotting optimal fit sigmoid

    marker = itertools.cycle(('+', 'o', '*')) # cycle between these markers for the experimental data--one marker for one experiment

    M=len(expnames) # the number of experiments that gave the data in lnthetas and rho vectors
    N=np.size(lnthetas)/M # the number of data points in each experiment

    plt.figure(1)
    plt.clf()
    for i in range(0,M):
        plt.plot(lnthetas[(i)*N:(i+1)*N+1],rhos[(i)*N:(i+1)*N+1],linestyle='',marker=marker.next(),label=expnames[i])
# plt.plot(lnthetas,rhos,'ro',label='Experimental data points') # use if just one experiment
    plt.plot(xs,ys,'b-',label='Best fit sigmoid curve')
    plt.legend(loc='upper left')
    plt.xlabel(r'$\ln(\Theta(t,T(t)))$ $\left[\ln(\frac{s}{K})\right]$')
    plt.ylabel('Relative Density')
    plt.title(titlestring)
    plt.text(-58,0.54,funstring,fontsize=19) # figure out a better way to place this text than trial-error
    plt.savefig(savestring+'sigmoidplot.png') # saves the plot as .png without displaying it
    tikz_save(savestring+'sigmoidplot.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False ) # save data as .tex so plot can be recreated directly in LaTeX
    if show:
        plt.show() # displays the plot
    plt.close(1)
    return()

def fantozzi(lnthetas,rhos,expnames,Q,savestring,showinfo):
    ''' rhofun,err = fantozzi(lnthetas,rhos,Q,savestring):
        Finds a, b, c, rho_0, and ln(theta_0) as parameters of the
        sigmoid curve defined in [Fantozzi et al.] to be
        rho(theta) = rho_0 + a/[1+exp(-[ln(theta)-ln(theta_0)]/b)]^c.

        Uses Levenberg-Marquardt to solve the nonlinear optimization problem.

        Inputs: lnthetas the ln(thetas) for a given sintering experiment. A vector
                    (array) of length N. Units of ln(s/K).
            rhos  the relative density values corresponding to the values in
                    'lnthetas' input. A vector (array) of length N. Units of 1.
            expnames the titles of each experiment corresponding to a column of
                    times/temps/rhos matrices. A length-M list of strings.
            Q  activation energy of substance (can be determined from
                    separate optimization routine). A scalar. Units of J/mol.
            savestring filepaths to use for plots and data when saving automatically.
                    A string.
            showinfo Tells whether or not to print information about optimal
                    parameters and error. A boolean.

        Outprints: a  parameter in [Fantozzi et al.] model (see above description).
                    A scalar.
              b  parameter in [Fantozzi et al.] model (see above description).
                    A scalar.
              c  parameter in [Fantozzi et al.] model (see above description).
                    A scalar.
              rho_0 parameter in [Fantozzi et al.] model (see above description).
```

```
                                A scalar.
                    ln(theta_0) parameter in [Fantozzi et al.] model (see above
                        description).
                                A scalar.
        Outputs: rhofun  A function handle allowing user to input lntheta and get out
             rho.
                     err  The least squares error in particular sigmoid curve
    '''

    if showinfo:
        print "Using [Fantozzi et al.] representation of sigmoid curve"
    logfile = open(savestring+'msc.log','w+')
    logprint = "Using [Fantozzi et al.] representation of sigmoid curve: \n rho(theta) =
         rho_0 + a/[1+exp(-[ln(theta)-ln(theta_0)]/b)]^c \n"
    logfile.write(logprint)

    # Define sigmoid curve as a function of lnthetas and parameters a, b, c, rho_0, ln(
        theta_0)

    def fantozzisigmoid(lnth,a,b,c,rho0,lnth0):
        '''fantozzisigmoid(lnt,a,b,c,rho0,lnth0): the sigmoid curve defined in
            Fantozzi et al. as
                                rho(lnth) = rho_0 + a/[1+exp(-[lnth-ln(th0)]/b)]^c.'''
        return rho0 + a / ((1 + np.exp(-(lnth-lnth0)/b) )**(c))

    # Levenberg-Marquardt to find optimal a, b, c, rho_0, ln(theta_0) as parameters of
        the sigmoid curve

    # THE PROBLEM IS HOW STUPIDLY CLOSE THE INITIAL GUESS HAS TO BE !
    initguess = np.r_[0.4937,1.0615,0.3995,0.4975,-35.206] # initial guess for a,b,c,
        rho_0,lnth0
    popt, pcov = curve_fit(fantozzisigmoid,lnthetas,rhos,initguess)

    if showinfo:
        print "Optimal values of [a, b, c, rho0, lntheta0] found using Levenberg-
            Marquardt optimization:"
        print popt
    logprint = "Optimal parameter values found using Levenberg-Marquardt optimization:\
        n \t a = %g\n\t b = %g\n\t c = %g\n\t rho0 = %g\n\t lntheta0 = %g\n"%(popt[0],
        popt[1],popt[2],popt[3],popt[4])
    logfile.write(logprint)
```

```

        # Build a function with one input for ln(theta) using optimal parameters found

        def rhofun(lnth):
                return fantozzisigmoid(lnth,popt[0],popt[1],popt[2],popt[3],popt[4])

        # Determine least−squares error of optimal sigmoid curve

        err = np.sum((rhos−rhofun(lnthetas))**2)/np.mean(rhos)
        if showinfo:
                print "With least−squares error: %g"%(err)
        logprint = "With least−squares error: %g"%(err)
        logfile.write(logprint)

        # Plot sigmoid curve along with the measured data points
        titlestring = "Master Sintering Curve (Fantozzi) for Zirconia"
        funstring = r'$\rho = %g + \frac{%g}{\left[1+\mathrm{exp}\left(−\frac{\ln\theta−(%g
            )}{%g}\right)\right]^{%g}}$'%(popt[3],popt[0],popt[4],popt[1],popt[2])
        plot_sigmoid(lnthetas,rhos,expnames,rhofun,savestring,funstring,titlestring,showinfo
            )

        return (rhofun,err)

def blaine(lnthetas,rhos,expnames,Q,savestring,showinfo):
        ''' rhofun,err = blaine(lnthetas,rhos,Q,savestring):
                Finds a, b, c, rho_0, and ln(theta_0) as parameters of the
                sigmoid curve defined in [Blaine et al.] to be
                rho(theta) = rho_0 + (1−rho_0)/[1+exp(−[ln(theta)−a]/b)].

                Uses Levenberg−Marquardt to solve the nonlinear optimization problem.

                Inputs: lnthetas the ln(thetas) for a given sintering experiment. A vector
                                (array) of length N. Units of ln(s/K).
                        rhos  the relative density values corresponding to the values in
                                'lnthetas' input. A vector (array) of length N. Units of
                                    1.
                        expnames the titles of each experiment corresponding to a column of
                                times/temps/rhos matrices. A length−M list of strings.
                        Q  activation energy of substance (can be determined from
                                separate optimization routine). A scalar. Units of kJ/
                                    mol.
```

```
                    savestring filepaths to use for plots and data when saving
                        automatically.
                                A string.
                    showinfo Tells whether or not to print information about optimal
                                parameters and error. A boolean.

              Prints: a parameter in [Blaine et al.] model (see above description).
                                          A scalar.
                       b parameter in [Blaine et al.] model (see above description).
                                          A scalar.
                       rho0   parameter in [Blaine et al.] model (see above description).
                                          A scalar.
                       savestring_Q_path.png .png image file with the chosen Q-values and
                                          the optimization path taken to arrive at soln
                       savestring_Q_path.tex same as above, except a .tex file for
                           plotting
                                          in latex instead of just importing graphic

             Outputs: rhofun  A function handle allowing user to input lntheta and get out
                  rho.
                      err  The least squares error in particular sigmoid curve
    '''
    if showinfo:
        print "Using [Blaine et al.] representation of sigmoid curve"
    logfile = open(savestring+'msc.log','w+')
    logprint = "Using [Blaine et al.] representation of sigmoid curve: \n rho(theta) =
        rho_0 + (1−rho_0)/[1+exp(−[ln(theta)−a]/b)] \n"
    logfile.write(logprint)

    # Define sigmoid curve as a function of lnthetas and parameters a, b, rho_0

    def blainesigmoid(lnth,a,b,rho0):
        '''blainesigmoid(lnt,a,b,rho0): the sigmoid curve defined in [Blaine et al.]
            as
                    rho(lnth) = rho0 + (1−rho0)/[1+exp(−[lnth−a]/b)].'''
        return rho0 + (1−rho0)/(1+np.exp((−lnth+a)/b))

    # Levenberg−Marquardt to find optimal a, b, rho_0 as parameters of the sigmoid
        curve

    initguess = np.r_[−25,2,0.8] # initial guess for a,b,c,rho_0,lnth0
```

```
# initguess = np.r_[0.5,0.4,0.5] # initial guess for a,b,rho0
       popt, pcov = curve_fit(blainesigmoid,lnthetas,rhos,initguess)

       if showinfo:
              print "Optimal values of [a, b, rho0] found:"
              print popt
       logprint = "Optimal parameter values found using Levenberg−Marquardt optimization:\
           n \t a = %g\n\t b = %g\n\t rho0 = %g\n"%(popt[0],popt[1],popt[2])
       logfile.write(logprint)

       # Build a function with one input for ln(theta) using optimal parameters found

       def rhofun(lnth):
              return blainesigmoid(lnth,popt[0],popt[1],popt[2])

       # Determine least−squares error of optimal sigmoid curve

       err = np.sum((rhos−rhofun(lnthetas))**2)/np.mean(rhos)

       if showinfo:
              print "With least−squares error: %g"%(err)
       else:
              logprint = "With least−squares error: %g"%(err)
              logfile.write(logprint)

       # Plot sigmoid curve along with the measured data points

       titlestring = "Master Sintering Curve (Blaine) for Zirconia"
       funstring = r'$\rho = %g+\frac{(1−(%g))}{[1+\mathrm{exp}(−\frac{\ln\theta−(%g)}{%g
           })]}$'%(popt[2],popt[2],popt[0],popt[1])
       plot_sigmoid(lnthetas,rhos,expnames,rhofun,savestring,funstring,titlestring,showinfo
           )

       return (rhofun,err)

def find_sigmoid(times,temps,rhos,expnames,Q,method,savestring,showinfo):
       ''' rhofun,err = find_sigmoid(times,temps,rhos,Q,method,savestring,showinfo):
              Finds a, b, c, rho_0, and ln(theta_0) as parameters of the
              sigmoid curve defined in method string.

              Inputs: times  the times at which temperature and density measurements
```

```
                                    were taken experimentally. An N*M array, M = #
                                        experiments
                                    Units of seconds.
                       temps  the temperatures corresponding to the times in 'times'
                                    input. An N*M array. Units of degC.
                       rhos  the relative density values corresponding to the times in
                                    'times' input. An N*M array. Units of 1.
                       expnames the titles of each experiment corresponding to a column of
                                    times/temps/rhos matrices. A length-M list of strings.
                       Q  activation energy of substance (can be determined from
                                    separate optimization routine). A scalar. Units of J/mol
                                        .
                       method  which representation of sigmoid curve. A string,
                                    either 'blaine' or 'fantozzi'.
                       savestring filepaths to use for plots and data when saving
                           automatically.
                                    A string.
                       showinfo Should we display the plot of final msc?

              Outputs: rhofun  A function handle allowing user to input lntheta and get out
                   rho.
       '''

       # Check the input values -- do they make sense?

       if np.shape(temps) != np.shape(times) : # if temps isn't same size as times, throw
           an error
              print "Shape of temps matrix must be the same as shape of times matrix"
              input()
              sys.exit(1)
       elif np.shape(rhos) != np.shape(times) : # if rhos isn't same size as times, throw
           an error
              print "Shape of rhos matrix must be the same as shape of times matrix"
              input()
              sys.exit(1)

       if np.size(np.shape(temps)) == 2: # data comes from several experiments

              # lntheta-ize the times and temps
              lnthetas = np.zeros(np.shape(times[1:,:]))
              N = np.shape(lnthetas)[1]
```

```
            for i in xrange(0,N):
                  lnthetas[:,i] = find_lnthetas(times[:,i],temps[:,i],Q)

            # reshape the lnthetas and rhos into vectors, column-by-column
            lnthetas = np.reshape(np.transpose(lnthetas),np.size(lnthetas))
            rhos = np.reshape(np.transpose(rhos[1:,:]),np.size(rhos[1:,:])) #and get rid
                of first rho-val in each column

      elif np.size(np.shape(temps)) == 1: # data comes from only one experiment

            lnthetas=find_lnthetas(times,temps,Q) # lntheta-ize the times and temps
            rhos = rhos[1:] # get rid of first rho-val

      else: # temps is either a scalar, or an array with >= 3 dimensions, so throw an
          error
            print "Temps, rhos, and times must be stored as arrays of two dimensions!"
            input()
            sys.exit(1)

      # call sigmoid fitting function
      rhofun,err = eval(method)(lnthetas,rhos,expnames,Q,savestring,showinfo)

      # plot results

      return rhofun

def find_Q(times,temps,rhos,expnames,method,savestring,showinfo):
      ''' Q,rhofun = find_Q(times,temps,rhos,method,savestring,showinfo):
            Finds activation energy (Q) that minimizes the least square error of the
            sigmoid curve (with parameters defined by 'method' string) fitted to the
            experimental data in times, temps, and rhos.

            Uses Nelder-Mead method to solve the optimization problem (because we do not
                ,
            unfortunately, have any information about the gradient of the objective
                function).

            Inputs: times  the times at which temperature and density measurements
                                 were taken experimentally. An NxM array, with each col
                                 corresponding to a single experiment. Units of seconds.
                  temps  the temperatures corresponding to the times in 'times'
```

```
                              input. An NxM array, with each col corresponding to a
                              single experiment. Units of degC.
                rhos  the relative density values corresponding to the times in
                              'times' input. An NxM array, with each col corresponding
                              to a single experiment. Units of 1.
                expnames the titles of each experiment corresponding to a column of
                              times/temps/rhos matrices. A length-M list of strings.
                method  which representation of sigmoid curve. A string,
                              either 'blaine' or 'fantozzi'.
                savestring filepaths to use for plots and data when saving
                    automatically.
                              A string.
                showinfo Should we display the plot of final msc?

        Prints: savestring_Q_path.png .png image file with the chosen Q-values and
                                      the optimization path taken to arrive at soln
                 savestring_Q_path.tex same as above, except a .tex file for plotting
                                      in latex instead of just importing graphic

        Outputs: Q the activation energy that minimizes least square
                          error of sigmoid fit
                  rhofun the sigmoid curve that is the fit corresponding to Q
    '''
    print "Determining optimal activation energy (Q) and corresponding sigmoid curve..."

    # Check the input data
    if np.shape(times) != np.shape(temps):
        print "Size of temps must be the same as size of times"
        input()
        sys.exit(1)
    elif np.shape(times) != np.shape(rhos):
        print "Size of rhos must be the same as size of times"
        input()
        sys.exit(1)

    # Define the objective function for optimizing
    def objectivefun(Q,times,temps,rhos,method):

        # lntheta-ize the times and temps

        lnthetas = np.zeros(np.shape(times[1:,:]))
```

```
            N = np.shape(lnthetas)[1]
            for i in xrange(0,N):
                  lnthetas[:,i] = find_lnthetas(times[:,i],temps[:,i],Q)

            # reshape the lnthetas and rhos into vectors, column-by-column
            lnthetas = np.reshape(np.transpose(lnthetas),np.size(lnthetas))
            rhos = np.reshape(np.transpose(rhos[1:,:]),np.size(rhos[1:,:]))

            # find the error and the optimal sigmoid curve
            rhofun,err = eval(method)(lnthetas,rhos,expnames,Q,savestring,showinfo=False)

            return err

      # Callback function for printing/showing optimization results
# def callbackF(params):
#  global itno
#  global Qs
#  global errs
#  Qs = np.r_[Qs, params[0]]
#  errs = np.r_[errs, ]
#  itno += 1

      # Perform the optimization using Nelder-Mead

      OptimResult = minimize(objectivefun,660*1000,(times,temps,rhos,method),method='
          Nelder-Mead',options={'disp': True})

      if OptimResult.success == True:
            Q = OptimResult.x
            print "Optimal Q found: %g"%(Q)

            # Get back the corresponding optimal sigmoid curve

            lnthetas = np.zeros(np.shape(times[1:,:]))
            N = np.shape(lnthetas)[1]

            for i in xrange(0,N):
                  lnthetas[:,i] = find_lnthetas(times[:,i],temps[:,i],Q)
            lnthetas = np.reshape(np.transpose(lnthetas),np.size(lnthetas))
            rhos = np.reshape(np.transpose(rhos[1:,:]),np.size(rhos[1:,:]))
            rhofun,err = eval(method)(lnthetas,rhos,expnames,Q,savestring,showinfo)

            return (Q,rhofun)
        else:
            print "Optimization with Nelder-Mead failed!!"
            input()
            sys.exit(1)
```

F.3 Computer Implementation in MATLAB of the Solver for the Stress-Strain Problems

One-Dimensional Deformation

```
function mat_1d_example

L=10; % length of domain
h=0.1; % step size
x=(0:h:L)';

eps_dot=lintens(x); % create strain rate tensor corresponding to linear thermal
    deformations (see Salenon p.78)

if compat(eps_dot,x) % we pass the compatibility conditions
  v=resolve(eps_dot,x); % calculate the velocity field

  t=1; %timestep in seconds (?--units depend on the constants in lintens)

  xd=x+t*v(:,1); % calculate displacement in x-dir

  x=x(2:end); xd=xd(2:end);

  figure(2); clf; hold on;
    p=plot(x,0.5,'-b',xd,1,'-r');
    set(gca,'YTick',[0.5 1],'YTickLabel',{'Original','Deformed'});
    set(p,'LineWidth',2);
    axis([min([x;xd])-0.05*max([x;xd]) 1.05*max([x;xd]) 0 1.5]);

else error('Failed compatibility conditions'); % WHAT TO DO IN THIS CASE?? CAN WE GET '
    CLOSE ENOUGH' SOME OTHER WAY??
end

end % function mat_1d_example
```

```

function eps_dot=lintens(x)

N=length(x); % number of gridpoints
eps_dot=zeros(N,3,3);

% Constants are arbitrarily chosen
a1=1;
b=0;

for i=1:N % There must be a way to vectorize this . . .
  eps_dot(i,:,:)=(a1*x(i)+b)*eye(3);
end % for i=1:N

end % function eps_dot=lintens(x)

function pass=compat(e,x)
% sees whether eps_dot (an Nx3x3 tensor) satisfies compatibility conditions
%
% 1D only!

% Condition 2: e33_11 = 0
e33_11 = (e(1:end−2,3,3)−2*e(2:end−1,3,3)+e(3:end,3,3))./((x(2:end−1)−x(1:end−2)).*(x(3:
    end)−x(2:end−1)));
% Condition 3: e22_11 = 0
e22_11 = (e(1:end−2,2,2)−2*e(2:end−1,2,2)+e(3:end,2,2))./((x(2:end−1)−x(1:end−2)).*(x(3:
    end)−x(2:end−1)));
% Condition 5: e23_11 = 0
e23_11 = (e(1:end−2,2,3)−2*e(2:end−1,2,3)+e(3:end,2,3))./((x(2:end−1)−x(1:end−2)).*(x(3:
    end)−x(2:end−1)));

if max([e33_11 e22_11 e23_11]) < 1e−5, pass=1;
else pass=0;
end % if max([e33_11 e22_11 e23_11]) < 1e−5

end % function pass=compat(e,x)

function v=resolve(e,x)
% solves integral equations getting eps_dot into a velocity field
% assumes left−hand endpoint stays put

```

```
v=zeros(length(x),3);

v(2:end,1)=e(1:end−1,1,1)+0.5*(x(2:end)−x(1:end−1)).*(e(2:end,1,1)+e(1:end−1,1,1));
v(2:end,2)=2*e(1:end−1,1,2)+(x(2:end)−x(1:end−1)).*(e(2:end,1,2)+e(1:end−1,1,2));
v(2:end,3)=2*e(1:end−1,1,2)+(x(2:end)−x(1:end−1)).*(e(2:end,1,3)+e(1:end−1,1,3));

end % function v=resolve(e,x)
```

Two-Dimensional Deformation

```
function mat_2d_example

L1=20; % length in x−dir
L2=10; % length in y−dir

h1=1; % step size in x−dir
h2=1; % step size in y−dir

x=(0:h1:L1)';
y=(0:h2:L2)';

eps_dot=lintens(x,y);

if compat(eps_dot,x,y)

  v=resolve(eps_dot,x,y);
  t=1; % timestep is 1 second
  [X,Y]=meshgrid(x,y);
  XD=X+t*v(:,:,1); % calculate displacement in x−dir
  YD=Y+t*v(:,:,2); % calculate displacement in y−dir

  figure(1); clf; hold on; surf(X,Y,zeros(size(X)));
  figure(2); clf; hold on; surf(XD,YD,ones(size(XD)));

  D1=zeros(length(y),length(x));
  D2=zeros(length(y),length(x));
  a1=1; a2=1; b=1;
  r=0; l1=0; l2=0;
  for i=1:length(x)
    for j=1:length(y)
      x1=x(i); x2=y(j);
      D1(j,i)=(a1/2)*(x1*x1−x2*x2)+a2*x1*x2+b*x1−r*x2+l1;
      D2(j,i)=(a2/2)*(x2*x2−x1*x1)+a1*x2*x1+b*x2+r*x1+l2;
    end
  end

  figure(3); clf; hold on; surf(X+t*D1,Y+t*D2,zeros(size(X)));


else error('Failed compatibility conditions');
end

end

function eps_dot=lintens(x,y)

N1=length(x); N2=length(y);

eps_dot=zeros(N2,N1,3,3);

% Constants are arbitrarily chosen
a1=1;
a2=1;
b=1;

for i=1:N1
  for j=1:N2
    eps_dot(j,i,:,:)=(a1*x(i)+a2*y(j)+b)*eye(3);
  end
end

end

function pass=compat(e,x,y)
% sees whether eps_dot (an N1xN2x3x3 tensor) satisfies compatibility conditions
%
% 2D only!

N2=length(y);
N1=length(x);

```

```
y_cent_diffs=repmat((y(2:end−1)−y(1:end−2)).*(y(3:end)−y(2:end−1)),1,N1); % size is N2−2
     , N1
x_cent_diffs=repmat((x(2:end−1)−x(1:end−2))'.*(x(3:end)−x(2:end−1))',N2,1); % size is N2
     , N1−2
[X,Y] = meshgrid((x(2:end)−x(1:end−1)),(y(2:end)−y(1:end−1)));
xy_diffs = X.*Y; % size should be N1−1 , N2−1

% Condition 1: e33_22 = 0
e33_22 = zeros(N2,N1);
e33_22(2:end−1,:) = (e(1:end−2,:,3,3)−2*e(2:end−1,:,3,3)+e(3:end,:,3,3))./(y_cent_diffs)
    ;
%take care of the endpoints! What do e33_22(1,:) and e33_22(end,:) look like?
cond1 = max(max(e33_22));

% Condition 2: e33_11 = 0
e33_11 = zeros(N2,N1);
e33_11(:,2:end−1) = (e(:,1:end−2,3,3)−2*e(:,2:end−1,3,3)+e(:,3:end,3,3))./(x_cent_diffs);

cond2 = max(max(e33_11));

% Condition 3: 2e12_12 = e22_11 + e11_22
e12_12=zeros(N2,N1); e22_11=zeros(N2,N1); e11_22=zeros(N2,N1);

e12_12(1:end−1,1:end−1) = (e(2:end,2:end,1,2)−e(1:end−1,1:end−1,1,2)−e(2:end,1:end
    −1,1,2)+e(1:end−1,1:end−1,1,2))./(xy_diffs);
e22_11(:,2:end−1) = (e(:,1:end−2,2,2)−2*e(:,2:end−1,2,2)+e(:,3:end,2,2))./(x_cent_diffs);
e11_22(2:end−1,:) = (e(1:end−2,:,1,1)−2*e(2:end−1,:,1,1)+e(3:end,:,1,1))./(y_cent_diffs);

cond3 = max(max(2*e12_12 − e22_11 − e11_22));

% Condition 4: e33_21 = 0
e33_21=zeros(N2,N1);
e33_21(1:end−1,1:end−1) = (e(2:end,2:end,3,3)−e(1:end−1,2:end,3,3)−e(2:end,1:end−1,3,3)+
    e(1:end−1,1:end−1,3,3))./(xy_diffs);

cond4 = max(max(e33_21));
% Condition 5: e23_11 = e13_12
e23_11=zeros(N2,N1); e13_12=zeros(N2,N1);
e23_11(:,2:end−1) = (e(:,1:end−2,2,3)−2*e(:,2:end−1,2,3)+e(:,3:end,2,3))./(x_cent_diffs);
e13_12(1:end−1,1:end−1) = (e(2:end,2:end,1,3)−e(2:end,1:end−1,1,3)−e(1:end−1,2:end,1,3)+
    e(1:end−1,1:end−1,1,3))./(xy_diffs);
```

```

cond5 = max(max(e23_11 − e13_12));

% Condition 6: e31_22 = e32_12
e31_22=zeros(N2,N1); e32_12=zeros(N2,N1);
e31_22(2:end−1,:) = (e(1:end−2,:,3,1)−2*e(2:end−1,:,3,1)+e(3:end,:,3,1))./(y_cent_diffs);
e32_12(1:end−1,1:end−1) = (e(2:end,2:end,3,2)−e(2:end,1:end−1,3,2)−e(1:end−1,2:end,3,2)+
    e(1:end−1,1:end−1,3,2))./(xy_diffs);

cond6 = max(max(e31_22 − e32_12));

d=max([cond1,cond2,cond3,cond4,cond5,cond6]);

if d < 1e−2, pass=1; else pass=0; end

end

function v=resolve(e,x,y)
% takes eps_dot and outputs velocity field

v=zeros(length(y),length(x),3);

%v(:,:,3) = intgrad2(2*e(:,:,1,3),2*e(:,:,2,3),x,y);

intfun = zeros(size(y));

for i=2:length(x)
  for k=2:length(y)
    g_x=diff([0;e(2:k,1,2,2)])./(y(2:k)−y(1:k−1));
    intfun(k) = trapz(y(2:k),g_x,1); % this is needed for the second integration
  end
  for j=2:length(y)
    v(j,i,3) = 2*trapz(x(2:i),e(j,2:i,1,3),2) + 2*trapz(y(2:j),e(2:j,1,2,3),1);
    v(j,i,2) = trapz(x(2:i),e(j,2:i,1,2),2) + trapz(y(2:j),e(2:j,i,2,2),1);
    v(j,i,1) = trapz(x(2:i),e(1,2:i,1,1),2) + trapz(y(2:j),e(2:j,i,1,2),1) − trapz(y(2:j),
        intfun(2:j),1);
  end
end

end
```

Three-Dimensional Deformation

```
function mat_3d_example

L1=20; % length in x—dir
L2=10; % length in y—dir
L3=5; % length in z—die

h1=1; % step size in x—dir
h2=1; % step size in y—dir
h3=1;

x=(1:h1:L1)';
y=(1:h2:L2)';
z=(1:h3:L3)';

eps_dot=lintens(x,y,z);

if compat(eps_dot,x,y,z), v_field=resolve(eps_dot,x,y,z);
else error('Failed compatibility conditions');
end

end

function eps_dot=lintens(x,y,z)

N1=length(x); N2=length(y); N3=length(z);

eps_dot=zeros(N3,N2,N1,3,3);

% Constants are arbitrarily chosen
a1=2;
a2=pi;
a3=1;
b=1;

for i=1:N1
  for j=1:N2
    for k=1:N3
      eps_dot(k,j,i,:,:)=(a1*x(i)+a2*y(j)+a3*z(k)+b)*eye(3);
    end
  end
end

end

function pass=compat(e,x,y,z)
% sees whether eps_dot (an N1xN2xN3x3x3 tensor) satisfies compatibility conditions
%
% 3D
%
% MAKE THIS LESS MEMORY−INTENSIVE BY COMING UP WITH A FUNCTION TO DIFFERENTIATE ACROSS A
    GIVEN DIMENSION

N3=length(z);
N2=length(y);
N1=length(x);

z_cent_diffs=repmat((z(2:end−1)−z(1:end−2)).*(z(3:end)−z(2:end−1)),[1,N2,N1]); % size is
     N3−2, N2, N1,
y_cent_diffs=repmat((y(2:end−1)−y(1:end−2))'.*(y(3:end)−y(2:end−1))',[N3,1,N1]); % size
    is N3, N2−2 , N1
x_cent_diffs=repmat(reshape((x(2:end−1)−x(1:end−2)).*(x(3:end)−x(2:end−1)),[1 1 N1−2]),[
    N3,N2,1]); % size is N3, N2 , N1−2
[X,Y,Z] = meshgrid((y(2:end)−y(1:end−1)),z,(x(2:end)−x(1:end−1)));
xy_diffs = X.*Y; % size should be N3 , N2−1 , N1−1
[X,Y,Z] = meshgrid((y(2:end)−y(1:end−1)),(z(2:end)−z(1:end−1)),x);
yz_diffs = Y.*Z; % size should be N3−1 , N2−1 , N1
[X,Y,Z] = meshgrid(y,(z(2:end)−z(1:end−1)),(x(2:end)−x(1:end−1)));
xz_diffs = X.*Z; % size should be N3−1 , N2 , N1−1

% Condition 1: 2e23_23 = e33_22 + e22_33
e23_23=zeros(N3,N2,N1); e33_22=zeros(N3,N2,N1); e22_33 = zeros(N3,N2,N1);

e33_22(:,2:end−1,:) = (e(:,1:end−2,:,3,3)−2*e(:,2:end−1,:,3,3)+e(:,3:end,:,3,3))./(
    y_cent_diffs) ;
e22_33(2:end−1,:,:) = (e(1:end−2,:,:,2,2)−2*e(2:end−1,:,:,2,2)+e(3:end,:,:,2,2))./(
    z_cent_diffs) ;
e23_23(1:end−1,1:end−1,:) = (e(2:end,2:end,:,2,3)−e(1:end−1,1:end−1,:,2,3)−e(2:end,1:end
    −1,:,2,3)+e(1:end−1,1:end−1,:,2,3))./(yz_diffs);

cond1 = max(max(e33_22 + e22_33 − 2*e23_23));
```

```

% Condition 2: 2e31_31 = e11_33 + e33_11
e31_31=zeros(N3,N2,N1); e11_33=zeros(N3,N2,N1); e33_11 = zeros(N3,N2,N1);

e33_11(:,:,2:end−1) = (e(:,:,1:end−2,3,3)−2*e(:,:,2:end−1,3,3)+e(:,:,3:end,3,3))./(
    x_cent_diffs);
e11_33(2:end−1,:,:) = (e(1:end−2,:,:,1,1)−2*e(2:end−1,:,:,1,1)+e(3:end,:,:,1,1))./(
    z_cent_diffs);
e31_31(1:end−1,:,1:end−1) = (e(2:end,:,2:end,3,1)−e(1:end−1,:,1:end−1,3,1)−e(2:end,:,1:
    end−1,3,1)+e(1:end−1,:,1:end−1,3,1))./(xz_diffs);

cond2 = max(max(2*e31_31 − e11_33 − e33_11));

% Condition 3: 2e12_12 = e22_11 + e11_22
e12_12=zeros(N3,N2,N1); e22_11=zeros(N3,N2,N1); e11_22=zeros(N3,N2,N1);

e12_12(:,1:end−1,1:end−1) = (e(:,2:end,2:end,1,2)−e(:,1:end−1,1:end−1,1,2)−e(:,2:end,1:
    end−1,1,2)+e(:,1:end−1,1:end−1,1,2))./(xy_diffs);
e22_11(:,:,2:end−1) = (e(:,:,1:end−2,2,2)−2*e(:,:,2:end−1,2,2)+e(:,:,3:end,2,2))./(
    x_cent_diffs);
e11_22(:,2:end−1,:) = (e(:,1:end−2,:,1,1)−2*e(:,2:end−1,:,1,1)+e(:,3:end,:,1,1))./(
    y_cent_diffs);

cond3 = max(max(2*e12_12 − e22_11 − e11_22));

% Condition 4: e13_23 − e12_33 − e33_21 + e32_31 = 0
e13_23=zeros(N3,N2,N1); e12_33=zeros(N3,N2,N1); e33_21=zeros(N3,N2,N1); e32_31=zeros(N3,N2,
    N1);

e12_33(2:end−1,:,:) = (e(1:end−2,:,:,1,2)−2*e(2:end−1,:,:,1,2)+e(3:end,:,:,1,2))./(
    z_cent_diffs);
e33_21(:,1:end−1,1:end−1) = (e(:,2:end,2:end,3,3)−e(:,1:end−1,2:end,3,3)−e(:,2:end,1:end
    −1,3,3)+e(:,1:end−1,1:end−1,3,3))./(xy_diffs);
e32_31(1:end−1,:,1:end−1) = (e(2:end,:,2:end,3,2)−e(1:end−1,:,1:end−1,3,2)−e(2:end,:,1:
    end−1,3,2)+e(1:end−1,:,1:end−1,3,2))./(xz_diffs);
e13_23(1:end−1,1:end−1,:) = (e(2:end,2:end,:,1,3)−e(1:end−1,1:end−1,:,1,3)−e(2:end,1:end
    −1,:,1,3)+e(1:end−1,1:end−1,:,1,3))./(yz_diffs);

cond4 = max(max(e13_23 − e12_33 − e33_21 + e32_31));

% Condition 5: e21_31 − e23_11 − e11_32 + e13_12 = 0
```

```
e21_31=zeros(N3,N2,N1); e23_11=zeros(N3,N2,N1); e11_32=zeros(N3,N2,N1); e13_12=zeros(N3,N2,
    N1);

e23_11(:,:,2:end−1) = (e(:,:,1:end−2,2,3)−2*e(:,:,2:end−1,2,3)+e(:,:,3:end,2,3))./(
    x_cent_diffs);
e13_12(:,1:end−1,1:end−1) = (e(:,2:end,2:end,1,3)−e(:,2:end,1:end−1,1,3)−e(:,1:end−1,2:
    end,1,3)+e(:,1:end−1,1:end−1,1,3))./(xy_diffs);
e21_31(1:end−1,:,1:end−1) = (e(2:end,:,2:end,2,1)−e(1:end−1,:,1:end−1,2,1)−e(2:end,:,1:
    end−1,2,1)+e(1:end−1,:,1:end−1,2,1))./(xz_diffs);
e11_32(1:end−1,1:end−1,:) = (e(2:end,2:end,:,1,1)−e(1:end−1,1:end−1,:,1,1)−e(2:end,1:end
    −1,:,1,1)+e(1:end−1,1:end−1,:,1,1))./(yz_diffs);

cond5 = max(max(e21_31 − e23_11 − e11_32 + e13_12));

% Condition 6: e32_12 − e31_22 − e22_13 + e21_23 = 0
e32_12=zeros(N3,N2,N1); e31_22=zeros(N3,N2,N1); e22_13=zeros(N3,N2,N1); e21_23=zeros(N3,N2,
    N1);

e31_22(:,2:end−1,:) = (e(:,1:end−2,:,3,1)−2*e(:,2:end−1,:,3,1)+e(:,3:end,:,3,1))./(
    y_cent_diffs);
e32_12(:,1:end−1,1:end−1) = (e(:,2:end,2:end,3,2)−e(:,2:end,1:end−1,3,2)−e(:,1:end−1,2:
    end,3,2)+e(:,1:end−1,1:end−1,3,2))./(xy_diffs);
e22_13(1:end−1,:,1:end−1) = (e(2:end,:,2:end,2,2)−e(1:end−1,:,1:end−1,2,2)−e(2:end,:,1:
    end−1,2,2)+e(1:end−1,:,1:end−1,2,2))./(xz_diffs);
e21_23(1:end−1,1:end−1,:) = (e(2:end,2:end,:,2,1)−e(1:end−1,1:end−1,:,2,1)−e(2:end,1:end
    −1,:,2,1)+e(1:end−1,1:end−1,:,2,1))./(yz_diffs);

cond6 = max(max(e32_12 − e31_22 − e22_13 + e21_23));

d=max([cond1,cond2,cond3,cond4,cond5,cond6]);

if d < 1e−2, pass=1; else pass=0; end

end

function v=resolve(e,x,y,z)


v=zeros(length(z),length(y),length(x),3);

%v(:,:,1)=
%v(:,:,2)=
%v(:,:,3)=




























end
```

Appendix G

Computer Implementation in MATLAB of Models for Determining Effective Complex Permittivity of Metal Powders

Contains original MATLAB functions to realize each of the models described.

G.1 Lichtenecker's Mixture Formula

```
function epseff=lichtenecker(epses,vols)
%function epseff=lichtenecker(epses,vols)
%
%Uses Lichtenecker's mixture formula to compute the effective permittivity
%of a mixture comprised of a number of components.
%
%Inputs: epses − vector of complex permittivity values of materials
% comprising the mixture
% vols − vector of volume ratios of materials comprising the
% mixture. Or for two materals: volume fraction of first
%
%Outputs: epseff − effective complex permittivity of the mixture.

if length(vols)==1
   vols=[vols,1−vols];
end

epseff=prod(epses.^vols);
```

```
end
```

G.2 Correction to Lichtenecker's Formula by Neelakantaswamy *et al.*

```
function epseff=lichteneckercorr(epses,vol,ecr)
%function epseff=lichteneckercorr(epses,vol,ecr)
%
%Implements the corrected Lichtenecker mixture formulas as presented in
%Neelakantaswamy, Turkman, and Sakar (1985) with parts from Kisdnasamy and
%Neelakantaswamy (1984).
%
%Inputs: epses - vector of permittivity values of materials comprising
% the mixture. epses(1) is the permittivity of the
% inclusions, and epses(2) is the permittivity of the
% dielectric matrix
% vol - volume ratio of inclusions
% ecr - ratio of major/minor axes of ellipsoidal inclusions.
% ecr is 1 for spherical inclusions.
%
%Outputs: epseff - effective permittivity of the mixture.

e1=epses(1); e2=epses(2);

%Calculating the value for M found in Kisdnasamy and Neelakantaswamy
%f=1-ecr;
%ec=sqrt(f*(2-f));
ec=1;

m=ec*ec/(1-sqrt(1-ec*ec)*(asin(ec)/ec));

if e1>e2, M=2/(m-1); n=(5-M)/4;
else M=(m-1)/2; n=(M-1)/4;
end

%Everything that follows can be found in Neelakantaswamy, Turkman, Sakar
ed=e1-e2;
t=(e1+e2)/(2*ed*log(e1/e2))-(e1*e2)/(ed*ed);

a1=0.5-0.5*sqrt(1-4*t);
a2=1-a1;

```

```
eu=@(a) a*e1+(1-a)*e2;
el=@(a) 1/(a/e1+(1-a)/e2);

C=@(a) sqrt(el(a)/eu(a))*e1^(a)*e2^(1-a);
B=@(a) 1+1/(eu(a)^(n-1)*el(a)^(n-1));
A=@(a) 1+1/(eu(a)^(n)*el(a)^(n));
Z=@(a) (eu(a)^n)/(el(a)^(n-1));
Y=@(a) Z(a)+eu(a);
X=@(a) Z(a)+1/el(a);

av=vol;
if av<=a1
   if e1>e2, epseff=X(av)/2;
   else epseff=Y(av)/2;
   end
elseif av<=a2
   if e1>e2, epseff=0.5*(A(a1)/(2*C(a1))+B(a2)/(2*C(a2)))*C(av)*Z(av);
   else epseff=0.5*(B(a1)/(2*C(a1))+A(a2)/(2*C(a2)))*C(av)*Z(av);
   end
else
   if e1>e2, epseff=Y(av)/2;
   else epseff=X(av)/2;
   end
end
end
```

G.3 Maxwell-Garnett Model

```
function epseff=mg(epses,vr)
%function epseff=mg(epses,vr)
%
%Uses the Maxwell-Garnett mixture formula to compute the effective
%permittivity of a mixture comprised of two components.
%
%Inputs: epses - vector of complex permittivity values of materials
% comprising the mixture; matrix permittivity comes first
% then inclusion permittivity
% vr - volume ratio of the inclusion
%
%Outputs: epseff - effective complex permittivity of the mixture.

```

```
vr=1-vr;
e1=epses(1); e2=epses(2);
epseff=e1*(e2+2*e1+2*vr*(e2-e1))/(e2+2*e1-vr*(e2-e1));

end
```

G.4 Extension of Maxwell-Garnett Model

```
function epseff=mgcorr(epses,vr,N)
%function epseff=mgcorr(epses,vr,N)
%
%Uses the corrected Maxwell-Garnett mixture formula described by
%Koledintseva et al. formula to compute the effective permittivity of a
%mixture comprised of arbitrarily many components.
%
%Inputs: epses - vector of complex permittivity values of materials
% comprising the mixture; has length n+1, where n is the
% number of types of inclusion, and the permittivity of
% the matrix should be the first component.
% vr - vector of volume ratios of the inclusions; length n.
% N - matrix of the depolarization factors; size 3 x n, with
% the rows corresponding to Cartesian dimensions x, y,
% and z respectively, and the columns corresponding to
% the inclusions. For all spherical particles,
% N=1/3*ones(3,n). For rod-shaped particles, see
% Koledintseva in references.
%
%Outputs: epseff - effective complex permittivity of the mixture.

eb=epses(1);

s1=@(i) eb*(1/(eb+N(1,i)*(epses(i+1)-eb))+1/(eb+N(2,i)*(epses(i+1)-eb)) ...
   + 1/(eb+N(3,i)*(epses(i+1)-eb)));
s2=@(i) N(1,i)/(eb+N(1,i)*(epses(i+1)-eb))+N(2,i)/(eb+N(2,i)*...
   (epses(i+1)-eb)) + N(3,i)/(eb+N(3,i)*(epses(i+1)-eb));

S1=0; S2=0;
for i=1:length(vr)
  S1=S1+vr(i)*(epses(i+1)-eb)*s1(i);
  S2=S2+vr(i)*(epses(i+1)-eb)*s2(i);
end
```

```

epseff=eb+S1/(3-S2);
end
```

G.5 Bruggeman's Model

```
function epseff=brugg(epses,alpha)
%function epseff=brugg(epses,alpha)
%
%Uses Bruggeman's mixture formula to compute the effective permittivity
%of a mixture comprised of two components.
%
%Inputs: epses - vector of complex permittivity values of components
% vols - volume ratio of second component
%
%Outputs: epseff - effective complex permittivity of the mixture.

e1=epses(1); e2=epses(2);

alpha=1-alpha;
% bruggformula=@(ep) alpha*(e2-ep)/(e2+2*ep)+(1-alpha)*(e1-ep)/(e1+2*ep);
% dbrugg=@(ep) -3*alpha*e2/(e2+2*ep)^2+(-3)*(1-alpha)*e1/(e1+2*ep)^2;
% epseff=0.5*(e1+e2);
% epseff=newtonit(bruggformula,dbrugg,epseff,1e-5);
% fprintf('Newton Method gives epseff=%g\n',epseff);

a=-2; b=e1*(2-3*alpha)+e2*(3*alpha-1); c=e1*e2;
epseff1=(-b+sqrt(b^2-4*a*c))/(2*a);
epseff2=(-b-sqrt(b^2-4*a*c))/(2*a);

%fprintf('Quadratic Formula gives epseff1=%g and epseff2=%g\n',...
% epseff1,epseff2);

%Choose the positive branch
epseff=max(epseff1,epseff2);
```

G.6 Buchelnikov's Model

```
function [epseff]=buch(e1,e2,eg,p,r1,r2)
%function [epseff]=buch(e1,e2,eg,p,r1,r2)
%
```

```
%Uses Buchelnikov's model to compute the effective permittivity
%of a mixture with inclusions made of core-shell spheres.
%
%Inputs: e1 - permittivity of metallic core
% e2 - permittivity of dielectric shell
% eg - permittivity of gas (vacuum)
% p - volume fraction of metal in effective medium
% r1 - radius of metallic core
% r2 - radius of dielectric shell
% l - optional; if only five inputs are given, l is the fifth
% argument and represents (r2-r1)/r1
%
%Output: epseff - effective complex permittivity of the mixture.

if nargin==5
   l=r1;
   zeta=(1+l)*(1+l)*(1+l);
elseif nargin==6
   l=(r2-r1)/r1;
   zeta=(1+l)*(1+l)*(1+l);
else
   error('Check the number of input arguments!');
end

alpha=(zeta-1)*e1+(2*zeta+1)*e2;
beta=(2+zeta)*e1+2*(zeta-1)*e2;

A=e2*(3*e1+(zeta-1)*(e1+2*e2));
B=3*e2+(zeta-1)*(e1+2*e2);

C0=eg*(beta*e2+p*zeta*(A-beta*e2));
C1=2*alpha*eg-beta*e2+p*zeta*(2*A+beta*e2-eg*(B+2*alpha));
C2=2*(p*zeta*(alpha-B)-alpha);

d=C1*C1-4*C2*C0;
if d<0
   error('Discriminant <0, no real solutions');
end

r1=(-C1+sqrt(C1*C1-4*C2*C0))/(2*C2);
r2=(-C1-sqrt(C1*C1-4*C2*C0))/(2*C2);
```

```

%Trial by roots method — just to check
%r=roots([C2 C1 C0]);
%root1=r(1);
%root2=r(2);
%d1=r1−root1;
%d2=r2−root2;

if r1>0
   if r2<=0
      epseff=r1;
   else %r2>0
      fprintf('For e1=%7.4g, e2=%7.4g, eg=%7.4g, p=%7.4g, l=%7.4g,\n',e1,e2,eg,p,l);
      fprintf('two possible permittivities found: %g and %g\n',r1,r2);
      epseff=input('Please enter the permittivity value to use in this case.');
   end
elseif r1==0
   if r2>=0
      epseff=r2;
   elseif r2<=0
      epseff=0;
   end
else %r1<0
   if r2>0
      epseff=r2;
   elseif r2==0
      epseff=0;
   else %r2<0
      fprintf('For e1=%7.4g, e2=%7.4g, eg=%7.4g, p=%7.4g, l=%7.4g,\n',e1,e2,eg,p,l);
      fprintf('no positive values found; negative values: %9.6g and %9.6g\n',r1,r2);
      epseff=input('Please enter the permittivity value to use in this case.');
   end
end

% Trial by Newton's method — just to check
% buchel=@(ep) p*zeta*(e2*(3*e1+(zeta−1)*(e1+2*e2))−ep*(3*e2+...
% (zeta−1)*(e1+2*e2)))/(2*alpha*ep+beta*e2)+(1−p*zeta)*(eg−ep)/(eg+2*ep);
%
% dbuchel=@(ep) p*zeta*(−(2*alpha*ep+beta*e2)*(3*e2+(zeta−1)*(e1+2*e2))−...
% (e2*(3*e1+(zeta−1)*(e1+2*e2))−ep*(3*e2+(zeta−1)*(e1+2*e2)))*...
% (2*alpha))/((2*alpha*ep+beta*e2)*(2*alpha*ep+beta*e2))+(1−p*zeta)...
% *(-(eg+2*ep)-2*(eg-ep))/((eg+2*ep)*(eg+2*ep));
%
% epseff=0;
%
% fprintf('e1=%g\n',e1);
% fprintf('buchel(epseff)=%g\n',buchel(epseff));
% fprintf('dbuchel(epseff)=%g\n',dbuchel(epseff));
% fprintf('-----------------------------------\n');
%
% [epseff,conv]=newtonit(buchel,dbuchel,epseff,1e-5,500);
% fprintf('Newton Method gives epseff=%g\n',epseff);
```

Appendix H

Computer Implementation in `python` and `MATLAB` of the Coupled Solver for the 1D and 2D MW Sintering Problems

H.1 `python` Implementation of the Coupled Solver for the One-Dimensional Microwave Sintering Problem

```
#!/usr/bin/python

# outputs: graphs of temperature and root mean square of electric field, full sets of dielectric and thermal properties, and file fullsolve1.log with detailed output at each timestep
#
# Performs transient solution for the electric field in a one-dimensional domain with a constant power source at the left-hand side. See problem description in file (Thesis.pdf). Simulation domain has middle third of cavity filled with insulation, and middle third of insulation filled with material for processing.
#
# This code requires python 2.7 , and requires ffmpeg or avconv (may need to modify movie-making parts, depending on your system). ffmpeg can be installed by typing
#
# > sudo apt-get install ffmpeg
#
# and avconv may be installed by typing
#
# > sudo apt-get install avconv
```

```
#
# ----------------------------------------------------------------------------------------------------------
# DEPENDENCY TREE:
#
# - fullsolve1.py
# - matplotlib ( available from http://matplotlib.org/ )
#  - pyplot ( packaged with matplotlib; documentation available at http://matplotlib.org/api/pyplot_summary.html )
# - matplotlib2tikz ( available from https://github.com/nschloe/matplotlib2tikz )
# - numpy ( available from http://www.numpy.org/ )
#  - matlib ( packaged with numpy; documentation available at http://docs.scipy.org/doc/numpy/reference/routines.matlib.html )
# - scipy ( available at http://www.scipy.org/ )
#  - interpolate ( packaged with scipy; documentation available at http://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html )
# - time ( module packaged with python 2.7 ; documentation available at https://docs.python.org/2/library/time.html )
# - os ( module packaged with python 2.7 ; documentation available at https://docs.python.org/2/library/os.html )
# - msc.py ( available from Erin Kiley , emkiley@wpi.edu )
#  - scipy ( available at http://www.scipy.org/ )
#   - optimize ( packaged with scipy; documentation available at http://docs.scipy.org/doc/scipy/reference/tutorial/optimize.html )
#    - minimize ( packaged with optimize; documentation available at http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html )
#    - curve-fit ( packaged with optimize; documentation available at http://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html )
#   - integrate ( packaged with scipy; documentation available at http://docs.scipy.org/doc/scipy/reference/tutorial/integrate.html )
#  - numpy ( available from http://www.numpy.org/ )
#  - matplotlib ( available from http://matplotlib.org/ )
#   -pyplot ( packaged with matplotlib; documentation available at http://matplotlib.org/api/pyplot_summary.html )
#  - matplotlib2tikz ( available from https://github.com/nschloe/matplotlib2tikz )
#  - itertools ( module packaged with python 2.7 ; documentation available at https://docs.python.org/2/library/itertools.html )
#  - sys ( module packaged with python 2.7 ; documentation available at https://docs.python.org/2/library/sys.html )
# - emsolve1.py ( available from Erin Kiley , emkiley@wpi.edu )
#  - scipy ( available at http://www.scipy.org/ )
#    - sparse ( packaged with scipy; documentation available at http://docs.scipy.org/doc/
    scipy/reference/sparse.html )
# - thermsolve1.py ( available from Erin Kiley , emkiley@wpi.edu )
#  -scipy ( available at http://www.scipy.org/ )
#    - sparse ( packaged with scipy; documentation available at http://docs.scipy.org/doc/
    scipy/reference/sparse.html )
#
#
    -------------------------------------------------------------------


total_time = 10000 # total processing time [sec]
capture_every = 10 # capture a plot and print to logfile every (this many) seconds
theta_dep_params = False
mat_params = "Licht" # model to use for dielectric props in case theta_dep_params == False.
     either "Licht", "Rayleigh", "MG", or "Bruggeman"
magmat = False # True if sintering material is magnetic
method = "fantozzi" # model we use for fitting sigmoid to MSC. either 'blaine' or 'fantozzi
    ', see file msc.py for details
embc='abs' # either 'abs' for absorbing (inhomogeneous Neumann) boundary condition at right
    -hand endpoint, or 'pec' for perfect electric conductor (homogeneous Dirichlet)
    condition
tempbc='rad' # either 'rad' for radiative (third-kind), 'ins' for insulating (homogeneous
    Neumann), or 'fix' for fixed (inhomogeneous Dirichlet)
th = 0.5 # 0 = explicit, 0.5 = C-N, 1 = implicit
savedir = "./1d_demo_apr4_Theta_"+mat_params+"_"+str(total_time)+"sec/" # directory where
    we save plots and logfile
saveprefix = "1d_demo_" # prefix for plots and logfile names
savestring = savedir+saveprefix
hiddensavestring = savedir+'.'+saveprefix # for hiding the individual movie frames we save

# Import necessary packages
import matplotlib as mpl # access matplotlib via shorter 'mpl' prefix
import matplotlib.pyplot as plt # plotting library: 'plt' prefix
from matplotlib2tikz import save as tikz_save # for getting a file with tikz data to plot
    directly in thesis
import numpy as np # numpy: 'np' prefix
from numpy import * # we use a number of functions and want to make available at toplevel
from numpy.matlib import rand,zeros,ones,empty,eye # make these functions accessible
    directly at top level, because we use them
```

```
import scipy.interpolate as intp # interpolators: 'intp' prefix. We use b-splines in this code.
import time # for printing times to logfile
import os # for issuing commands related to movie-making and auxfile-deleting
import pickle # for saving state of simulation
mpl.rcParams['axes.formatter.useoffset']=False # tell matplotlib not to convert axis tick labels to scientific notation (was getting weird results)

# Open log file for writing
if not os.path.exists(savedir): # if savedir doesn't already exist
    os.makedirs(savedir) # then create it
logfile = open(savestring+'fullsolve1.log','w+')

# Log file header
printstring = ("Simulation started "+time.strftime("%A, %B %d, %Y")+" at "+time.strftime("%H:%M:%S %Z")+".\n\n")
logfile.write(printstring)
initialstarttime = time.clock()

# Important constants
mu0=pi*4e-7 # permeability of free space [N/A^2]
c = 299792458 # speed of light [m/s]
R = 8.314459848 # ideal gas constant [J/(mol*K)]

# Microwave scenario
P=1000.0 # power [W] supplied by magnetron at left-hand endpoint
a = 86.36e-3 # length of long side of cross-section of 3D waveguide [m] --this value corresponds to D-band, WR-340 waveguide
b = 43.18e-3 # length of short side of cross-section of 3D waveguide [m]
n_mod = 1 # corresponds to TE_nm excitation mode
m_mod = 0 # corresponds to TE_nm excitation mode

f_fs = 2.45e9 # frequency [Hz] of waves in free space
omega_fs = 2*pi*f_fs # angular frequency [Hz] or [rad/sec] of waves in free space
l_fs = c/f_fs # wavelength [m] in free space
omega_c = c*sqrt( (n_mod*pi/a)**2 + (m_mod*pi/b)**2 ) # angular cutoff frequency [Hz] or [rad/sec]
f_c = omega_c/(2*pi) # cutoff frequency [Hz]
# TO DO: Throw a warning if freespace frequency is less than cutoff: then we have evanescent TE_10 mode (wave doesn't propagate)
l_c = c/f_c # cutoff wavelength [m]
```

```
l_g = sqrt(1/((1/l_fs)**2 - (1/l_c)**2)) # wavelength [m] in waveguide
f_g = c/l_g # frequency [Hz] in waveguide
omega_g = 2*pi*f_g # angular frequency [Hz] or [rad/sec] in waveguide

L = 2.5*l_g # length of waveguide [m], set here equal to 2.5* wavelength in guide, so in the unloaded wg, using effective frequency of loaded, we have 5 peaks with one in the center (where the sample will be)

# Physical setup

L_mat=L/9 # length of material [m], to be centered within domain
L_ins=L/9 # length of insulation [m], to be placed on either side of material (occurs on both sides, so this is half of the total length of insulation)
# TO DO: Throw an error if cavity is too small to contain material + insulation (we do not really need this)

# Initial temperature
temp_init = 298.0 # room temperature (in kelvin)

printstring=("Waveguide length is "+str(L*1e2)+" cm\nLength of material is "+str(L_mat*1e2)+" cm\nLength of insulation on either side of material is "+str(L_ins*1e2)+" cm\nInput power is "+str(P/1000)+" kW\nFrequency of radiation is "+str(f_fs*1e-9)+" GHz\nInitial temperature is "+str(temp_init-273.15)+" K\n")
logfile.write(printstring)

# Load material: zirconia, data taken from {}
bulkdens_mat = 6.52e6 # density of solid load material [g/m^3]

# Load material: zirconia, experimental resutls taken from {McCoyThesis}. These are the ones used in determining activation energy and MSC.
# First trial: 1 degC/min
times_1 = 1.00*np.array([17192,20134,23142,26147,29086,32027,35033,38038,41046,44052,46993])
temps_1 = 273.15+np.array([900,950,1001,1051,1101,1150,1201,1250,1300,1350,1400])
rhos_1 = 0.01*np.array([46.7,47.1,48.3,51.8,58.6,69.7,82.2,89.7,91.0,91.3,91.4])

# Second trial: 3 degC/min
times_3 = 1.00*np.array([12086,13071,14016,15000,16023,17008,17992,19015,20000,21062,22086])
temps_3 = 273.15+np.array([901,951,999,1049,1101,1151,1199,1251,1300,1350,1400])
rhos_3 = 0.01*np.array([46.6,46.8,47.6,49.8,54.8,63.5,75.4,85.0,87.2,87.8,88.2])
```

```

# Third trial: 5 degC/min
times_5 = 1.00*np.array
    ([11271,11818,12398,12978,13559,14140,14754,15335,15916,16564,17247])
temps_5 = 273.15+np.array([901,949,1000,1049,1099,1149,1201,1250,1299,1351,1400])
rhos_5 = 0.01*np.array([46.6,46.8,47.5,49.4,53.6,61.3,72.7,82.0,84.5,85.3,85.9])

# Load material: zirconia, results taken from {Teng et al}. These are the ones used in
    determining activation energy and the MSC.
# First trial: 2 degC/min
times_2 = 1.0*np.array([9975,11475,12975,14475,15975,17475,18975,20475,21975,23745,25275])
    # times at which the temperatures were measured for the first experiment [s]
temps_2 = 273.15+np.array([1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550]) #
    temperatures for the first experiment[C]
rhos_2 = 0.01*np.array([54.43,55.7,60.15,67.53,76.40,85.35,92.71,96.42,97.63,98.79,98.89])
    # relative densities for the first experiment[%]

# Second trial: 5 degC/min
#times_5 = 1.0*np.array
    ([12360,13080,13560,14160,14760,15360,15960,16560,17160,17760,19560]) # times at which
     the temperatures were measured for the second experiment
#temps_5 = 273.15+np.array([1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550]) #
    temperatures for the second experiment
#rhos_5 = 0.01*np.array
    ([53.86,55.69,58.01,64.06,72.50,81.44,90.69,94.67,96.46,97.31,98.40]) # relative
    densities for the second experiment

# Third trial: 8 degC/min
times_8 = 1.0*np.array([7725,8100,8475,8850,9225,9600,9975,10650,10725,11100,12900]) #
    times at which the temperatures were measured for the first experiment
temps_8 = 273.15+np.array([1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550]) #
    temperatures for the third experiment
rhos_8 = 0.01*np.array([53.75,54.82,57.14,61.05,69.43,77.40,87.34,93.01,95.10,97.51,99.03])
     # relative densities for the third experiment

# Load material: zirconia, experimental results taken from {Yakovlev & Ceralink}. These are
     the ones used in creating property-update functions for everything *except* density,
    in case we rely on the mixture formulas. (In case we rely on the function-of-theta
    approximation, then we actually construct another sigmoid approximation for density
    and we use only the activation energy from the above.
```

```
t_mat=273.15+np.array([25, 69, 100, 139, 181, 228, 276, 324, 371, 420, 471, 523, 574, 636,
    698, 752, 809, 865, 921, 973, 1019, 1065, 1100]) # temperatures [C−>K] at which each
    of (eps,sig,c,rho,k) was measured for load material
times_mat=np.zeros(np.shape(t_mat))
times_mat[0]=(t_mat[0]−273.15)*(60/20)
for i in range(1,np.size(t_mat)):
      times_mat[i]=times_mat[i−1]+(60/20)*(t_mat[i]−t_mat[i−1]) # simulate constant
          heating rate of 20 degC/min, in the absence of better information
epses_mat=np.array([6.69, 5.86, 5.78, 5.75, 5.77, 5.82, 5.90, 5.98, 6.08, 6.18, 6.32, 6.47,
     6.60, 6.77, 6.97, 7.22, 7.53, 7.93, 8.53, 9.44, 10.46, 12.46, 14.77]) # [unitless]
sigmas_mat=np.array([0.0258, 0.0045, 0.0033, 0.0029, 0.0036, 0.0043, 0.0050, 0.0058,
    0.0078, 0.0121, 0.0185, 0.0288, 0.0442, 0.0664, 0.0975, 0.1416, 0.2003, 0.2786,
    0.4083, 0.5942, 0.8220, 1.2190, 1.6661]) # [S/m]
cs_mat=np.array([0.217, 0.324, 0.363, 0.398, 0.426, 0.450, 0.470, 0.487, 0.501, 0.514,
    0.526, 0.537, 0.547, 0.558, 0.568, 0.575, 0.583, 0.590, 0.597, 0.603, 0.607, 0.612,
    0.615]) # [J/(g C)]
rhos_mat=1.0e6*np.array([2.848, 2.844, 2.841, 2.838, 2.834, 2.830, 2.826, 2.821, 2.817,
    2.813, 2.809, 2.804, 2.800, 2.794, 2.789, 2.785, 2.780, 2.775, 2.770, 2.766, 2.762,
    2.758, 2.755])/bulkdens_mat # RELATIVE
#rhos_mat = rhos_mat[::−1] # ZIRCONIA ACTUALLY SHOWS NO DENSIFICATION AT ALL DURING THIS
    TRIAL... IT SHOWS THERMAL EXPANSION. We flip the vector here only in order to account
    for 'densification' in the other material property functions, in the event that we don
    't use theta−dependent functions, in the end
ks_mat=100.0*np.array([0.00198, 0.00290, 0.00320, 0.00344, 0.00362, 0.00373, 0.00381,
    0.00385, 0.00381, 0.00391, 0.00399, 0.00407, 0.00414, 0.00405, 0.00412, 0.00417,
    0.00421, 0.00426, 0.00430, 0.00433, 0.00436, 0.00439, 0.00441]) # [W/(m C)]
mus_mat=np.ones(shape(t_mat))

# Insulation material: alumina
trans_ins = 500.0 # heat transfer coefficient of insulation material

# Insulation material: alumina, parameters taken from {Yakovlev & Ceralink}. These will be
    used to determine polynomial functions for updating temperature−dependent values and
    density−dependent values
t_ins=273.15+np.array([25,100,200,300,400,500,600,700,809,900,1000,1100]) # temperatures [C
    −>K] at which each of (eps,sig,c,rho,k) was measured for insulation
epses_ins=np.array([1.520, 1.520, 1.517, 1.513, 1.523, 1.540, 1.563, 1.573, 1.584, 1.593,
    1.600, 1.608])
sigmas_ins=np.array([0.00005, 0.00007, 0.00015, 0.00035, 0.00062, 0.00081, 0.00091,
    0.00113, 0.00131, 0.00159, 0.00234, 0.00315])
```

```
cs_ins=np.array([0.764, 0.950, 1.042, 1.097, 1.135, 1.165, 1.190, 1.210, 1.230, 1.244,
    1.258, 1.271])
rhos_ins=1.0e6*np.array([0.4400, 0.4392, 0.4382, 0.4371, 0.4361, 0.4350, 0.4340, 0.4329,
    0.4318, 0.4309, 0.4299, 0.4288])
ks_ins=100.0*np.array([0.000631, 0.000725, 0.00085, 0.000975, 0.0011, 0.001225, 0.00135,
    0.001475, 0.0016, 0.0018, 0.0020, 0.0022])
mus_ins=np.ones(shape(t_ins))

# Air material
eps_air=1.0 #[unitless] relative permittivity of air
sig_air=8.0e-15 # [S/m] electrical conductivity of air
c_air=1.0 # [J/g*C] specific heat capacity of air
rho_air=2.0 # [g/m^3] density of air
k_air=0.024 # [W/g*C] thermal conductivity of air
mu_air=1.0 # [unitless] relative permeability of air

# Determine activation energy and sigmoid function rho = rho(theta(t,T))
#msc_times = np.c_[times_2,times_5,times_8]
#msc_temps = np.c_[temps_2,temps_5,temps_8]
#msc_rhos = np.c_[rhos_2,rhos_5,rhos_8]
#msc_expnames = ['2 degC/min','5 degC/min','8 degC/min']

msc_times = np.c_[times_1,times_3,times_5]
msc_temps = np.c_[temps_1,temps_3,temps_5]
msc_rhos = np.c_[rhos_1,rhos_3,rhos_5]
msc_expnames = ['1 degC/min','3 degC/min','5 degC/min']

printstring=("\nDetermining optimal activation energy and density function...\n\tUsing
    densification data from {McCoy Thesis}...\n\tAttempting data fit to "+method+" sigmoid
     curve...\n")
logfile.write(printstring)
starttime=time.clock()

import msc
#Q,rhofun = msc.find_Q(msc_times,msc_temps,msc_rhos,msc_expnames,method,savestring,showinfo
    =False)
#Q = 674214 # this is from result of previous optimization with {Teng} data and Fantozzi
    curve
Q=653298 # this is from result of previous optimization with {McCoy} data and Fantozzi
    curve
```

```python
rhofun=msc.find_sigmoid(msc_times,msc_temps,msc_rhos,msc_expnames,Q,method,savestring,showinfo=False)
#rhofun=msc.find_sigmoid(times_mat,t_mat,rhos_mat,['20 degC/min'],Q,method,savestring,showinfo=False)

printstring=("\tDone; took "+str(time.clock()-starttime)+" seconds to find optimal activation energy and MSC.\n\tOptimal activation energy is "+str(Q/1000)+" kJ/mol.\n\nInterpolating measured data to find dielectric and thermal properties as functions of temperature and relative density...")
logfile.write(printstring)
starttime=time.clock()

sampleplottemp = np.linspace(np.min(np.r_[t_mat,t_ins]),273.15+1200) # for plotting the material properties
tempmin = 24+273.15 # minimum temp we expect to encounter (tells spline interpolator that its values will eventually need to be extrapolated down to this value)
tempmax = 1400+273.15 # maximum temp we expect to encounter (tells spline interpolator that its values will eventually need to be extrapolated up to this value)

spldeg = 3 # degree of splines to use for interpolating (cubic recommended)


# Functions for load parameters
if theta_dep_params: # Method 1: theta-dependent parameters
    printstring = "\n\tAssuming parameters are functions of ln(theta)..."
    logfile.write(printstring)

    sampleplottimes=np.zeros(np.shape(sampleplottemp)) # assume a constant heating rate of 20 degC/min for sample data
    sampleplottimes[0]=(sampleplottemp[0]-273.15)*(60/20) # the time it took to get to the first temperature we have property measurements for
    for i in range(1,np.size(sampleplottemp)):
        sampleplottimes[i]=sampleplottimes[i-1]+(60/20)*(sampleplottemp[i]-sampleplottemp[i-1]) # simulate constant heating rate of 20 degC/min, in the absence of better information
    sampleplotlnthetas = msc.find_lnthetas(sampleplottimes,sampleplottemp,Q) # get the ln(theta) values for plotting functions
    lnthetas = msc.find_lnthetas(times_mat,t_mat,Q) # get the ln(theta) values for actually doing the interpolation

    lntmin = -400 # minimum lntheta we expect to encounter (tells spline interpolator
        that its values will eventually need to be extrapolated down to this value)
    lntmax = 30 # maximum lntheta we expect to encounter (tells spline interpolator that
         its values will eventually need to be extrapolated up to this value)

    # Functions for load parameters (these take lntheta as input)
    epstck = intp.splrep(lnthetas,epses_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline
        interpolation
    def epsfun_mat(lntheta):
        if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-
            vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),epstck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,epstck) # spline evaluation

    sigtck = intp.splrep(lnthetas,sigmas_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline
        interpolation
    def sigfun_mat(lntheta):
        if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-
            vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),sigtck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,sigtck) # spline evaluation

    ctck = intp.splrep(lnthetas,cs_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline
        interpolation
    def cfun_mat(lntheta):
        if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-
            vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),ctck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,ctck) # spline evaluation

    ktck = intp.splrep(lnthetas,ks_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline
        interpolation
    def kfun_mat(lntheta):
        if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),ktck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,ktck) # spline evaluation

    if magmat:
        mutck = intp.splrep(lnthetas,mus_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline interpolation
        def mufun_mat(lntheta):
            if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-vector
                n,m=np.shape(lntheta)
                splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),mutck)
                return np.reshape(splevals,(n,m))
            else: # lntheta was either a scalar or an array-that's-a-vector
                return intp.splev(lntheta,mutck) # spline evaluation
    else:
        def mufun_mat(lntheta):
            return np.ones(np.shape(lntheta))

# Uncomment for simple barycentric interpolation; we don't like this, though, because of values extrapolated beyond range of initial data--diverges quickly to +/- infty
# epsfun_mat = intp.BarycentricInterpolator(lnthetas,epses_mat[1:])
# sigfun_mat = intp.BarycentricInterpolator(lnthetas,sigmas_mat[1:])
# cfun_mat = intp.BarycentricInterpolator(lnthetas,cs_mat[1:])
# kfun_mat = intp.BarycentricInterpolator(lnthetas,ks_mat[1:])
# mufun_mat = intp.BarycentricInterpolator(lnthetas,mus_mat[1:])

    plt.figure(10) # Plot eps(temp) for material
    plt.clf()
    plt.plot(sampleplotlnthetas,epsfun_mat(sampleplotlnthetas),'r-',label='Function approximation')
    plt.plot(lnthetas,epses_mat[1:],'ro',label='Experimental measurements')
    plt.legend(loc='upper left')
    plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
    plt.ylabel(r'$\varepsilon_r$ [unitless]')
    plt.title('Relative electric permittivity for zirconia')
```

```
        plt.savefig(savestring+'mat_epsfun.png')
        tikz_save(savestring+'mat_epsfun.tex', figureheight = '\\figureheight', figurewidth
            = '\\figurewidth',show_info = False )
        plt.close(10)

        plt.figure(11) # Plot sigma(temp) for material
        plt.clf()
        plt.plot(sampleplotlnthetas,sigfun_mat(sampleplotlnthetas),'b-',label='Function
            approximation')
        plt.plot(lnthetas,sigmas_mat[1:],'ro',label='Experimental measurements')
        plt.legend(loc='upper left')
        plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
        plt.ylabel('$\sigma$ [S/m]')
        plt.title('Electrical conductivity for zirconia')
        plt.savefig(savestring+'mat_sigfun.png')
        tikz_save(savestring+'mat_sigfun.tex', figureheight = '\\figureheight', figurewidth
            = '\\figurewidth',show_info = False )
        plt.close(11)

        plt.figure(12) # Plot c_p(temp) for material
        plt.clf()
        plt.plot(sampleplotlnthetas,cfun_mat(sampleplotlnthetas),'g-',label='Function
            approximation')
        plt.plot(lnthetas,cs_mat[1:],'ro',label='Experimental measurements')
        plt.legend(loc='upper left')
        plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
        plt.ylabel('$c_p$ [J/(gK)]')
        plt.title('Specific heat capacity for zirconia')
        plt.savefig(savestring+'mat_cfun.png')
        tikz_save(savestring+'mat_cfun.tex', figureheight = '\\figureheight', figurewidth =
            '\\figurewidth',show_info = False )
        plt.close(12)

        plt.figure(13) # Plot k(temp) for material
        plt.clf()
        plt.plot(sampleplotlnthetas,kfun_mat(sampleplotlnthetas),'b-',label='Function
            approximation')
        plt.plot(lnthetas,ks_mat[1:],'ro',label='Experimental measurements')
        plt.legend(loc='upper left')
        plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
        plt.ylabel('k [W/(mK)]')
```

```
    plt.title('Thermal conductivity for zirconia')
    plt.savefig(savestring+'mat_kfun.png')
    tikz_save(savestring+'mat_kfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
    plt.close(13)

    plt.figure(14) # Plot mu(temp) for material
    plt.clf()
    plt.plot(sampleplotlnthetas,mufun_mat(sampleplotlnthetas),'g-',label='Function approximation')
    plt.plot(lnthetas,mus_mat[1:],'ro',label='Experimental measurements')
    plt.legend(loc='upper left')
    plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
    plt.ylabel('$\mu_r$ [unitless]')
    plt.title('Relative magnetic permeability for zirconia')
    plt.savefig(savestring+'mat_mufun.png')
    tikz_save(savestring+'mat_mufun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
    plt.close(14)

else: # Method 2: mixture formula-based load parameter functions (these take temp, rho as inputs)
    #rho_rel_init=(intp.BarycentricInterpolator(t_mat,rhos_mat).__call__(temp_init)+0.1) # relative density; the +0.1 is to make it closer to the density in the data used for MSC

    printstring = "\n\tUsing inversions of mixture formulas plus interpolation of parameter along rho-axis to determine functions for dielectric and thermal properties of load and insulation..."
    logfile.write(printstring)

    # Estimate values of bulk parameters for interpolating
    if mat_params == "Licht":
        e2fn = epses_mat**(1/rhos_mat)
        s2fn = sigmas_mat**(1/rhos_mat)
        m2fn = mus_mat**(1/rhos_mat)
    elif mat_params == "Rayleigh":
        e2fn = (1+(2/rhos_mat)*((epses_mat-1)/(epses_mat+1)))/(1-(1/rhos_mat)*((epses_mat-1)/(epses_mat+1)))
        s2fn = (1+(2/rhos_mat)*((sigmas_mat-1)/(sigmas_mat+1)))/(1-(1/rhos_mat)*((sigmas_mat-1)/(sigmas_mat+1)))
```

```
            m2fn = (1+(2/rhos_mat)*((mus_mat-1)/(mus_mat+1)))/(1-(1/rhos_mat)*((mus_mat
                -1)/(mus_mat+1)))
    elif mat_params == "MG":
            e2fn = (eps_air*(1+rhos_mat)*(epses_mat-eps_air))/(2*rhos_mat*eps_air-(1-
                rhos_mat)*(epses_mat-eps_air))
            s2fn = (sig_air*(1+rhos_mat)*(sigmas_mat-sig_air))/(2*rhos_mat*sig_air-(1-
                rhos_mat)*(sigmas_mat-sig_air))
            m2fn = (mu_air*(1+rhos_mat)*(mus_mat-mu_air))/(2*rhos_mat*mu_air-(1-
                rhos_mat)*(mus_mat-mu_air))
    elif mat_params == "Bruggeman":
            e2fn = (epses_mat*(1-3*rhos_mat)+2*epses_mat*epses_mat)/(1+epses_mat*(2-3*
                rhos_mat))
            s2fn = (sigmas_mat*(1-3*rhos_mat)+2*sigmas_mat*sigmas_mat)/(1+sigmas_mat
                *(2-3*rhos_mat))
            m2fn = (mus_mat*(1-3*rhos_mat)+2*mus_mat*mus_mat)/(1+mus_mat*(2-3*rhos_mat))

    # Interpolate bulk parameters with temperature
    epstck = intp.splrep(t_mat,e2fn,xb=tempmin,xe=tempmax,k=spldeg)
    sigtck = intp.splrep(t_mat,s2fn,xb=tempmin,xe=tempmax,k=spldeg)
    mutck = intp.splrep(t_mat,m2fn,xb=tempmin,xe=tempmax,k=spldeg)

    # Construct functions
    if mat_params == "Licht":
            def epsfun_mat(temp,rho):
                    eps2=intp.splev(temp,epstck)
                    return eps2**rho
            def sigfun_mat(temp,rho):
                    sig2=intp.splev(temp,sigtck)
                    return sig2**rho
            def mufun_mat(temp,rho):
                    mu2=intp.splev(temp,mutck)
                    return mu2**rho
    elif mat_params == "Rayleigh":
            def epsfun_mat(temp,rho):
                    eps2=intp.splev(temp,epstck)
                    return (eps2*(2*rho+1)-(2*rho-2))/(eps2*(1-rho)+(rho-2))
            def sigfun_mat(temp,rho):
                    sig2=intp.splev(temp,sigtck)
                    return (sig2*(2*rho+1)-(2*rho-2))/(sig2*(1-rho)+(rho-2))
            def mufun_mat(temp,rho):
                    mu2=intp.splev(temp,mutck)
```

```python
            return (mu2*(2*rho+1)-(2*rho-2))/(mu2*(1-rho)+(rho-2))
    elif mat_params == "MG":
        def epsfun_mat(temp,rho):
            eps2=intp.splev(temp,epstck)
            return eps_air+2*rho*eps_air*(eps2-eps_air)/(eps2+eps_air-rho*(eps2-eps_air))-2
        def sigfun_mat(temp,rho):
            sig2=intp.splev(temp,sigtck)
            return sig_air+2*rho*sig_air*(sig2-sig_air)/(sig2+sig_air-rho*(sig2-sig_air))
        def mufun_mat(temp,rho):
            mu2=intp.splev(temp,mutck)
            return mu_air+2*rho*mu_air*(mu2-mu_air)/(mu2+mu_air-rho*(mu2-mu_air))
    elif mat_params == "Bruggeman":
        def epsfun_mat(temp,rho):
            eps2=intp.splev(temp,epstck)
            return 0.5*(1+3*rho*(1-eps2))+0.5*sqrt((1+3*rho*(1-eps2))**(2)+4*eps2)
        def sigfun_mat(temp,rho):
            sig2=intp.splev(temp,sigtck)
            return 0.5*(1+3*rho*(1-sig2))+0.5*sqrt((1+3*rho*(1-sig2))**(2)+4*sig2)
        def mufun_mat(temp,rho):
            mu2=intp.splev(temp,mutck)
            return 0.5*(1+3*rho*(1-mu2))+0.5*sqrt((1+3*rho*(1-mu2))**(2)+4*mu2)

    if not magmat:
        def mufun_mat(temp,rho):
            return np.ones(np.shape(temp))

    # Specfic heat capacity
    ctck = intp.splrep(t_mat,cs_mat/rhos_mat,xb=tempmin,xe=tempmax,k=spldeg)
    def cfun_mat(temp,rho): # takes RELATIVE density as input
        return intp.splev(temp,ctck)*rho

    # Thermal conductivity
    ktck = intp.splrep(t_mat,ks_mat/(1.5*rhos_mat-0.5),xb=tempmin,xe=tempmax,k=spldeg)
    def kfun_mat(temp,rho):
        return intp.splev(temp,ktck)*(1.5*rho-0.5)

    # For plotting
```

```
        sampleplotrhovals = np.linspace(rhos_mat[0],rhos_mat[-1])

        sampleploteps=np.zeros(np.shape(sampleplottemp))
        sampleplotsig=np.zeros(np.shape(sampleplottemp))
        sampleplotmu=np.zeros(np.shape(sampleplottemp))
        sampleplotc=np.zeros(np.shape(sampleplottemp))
        sampleplotk=np.zeros(np.shape(sampleplottemp))

        for ind in range(0,np.size(sampleplottemp)):
                sampleploteps[ind]=epsfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
                sampleplotsig[ind]=sigfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
                sampleplotmu[ind]=mufun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
                sampleplotc[ind]=cfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
                sampleplotk[ind]=kfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])

        plt.figure(10) # Plot eps(temp) for material
        plt.clf()
# plt.plot(sampleplottemp-273.15,epsfun_mat(sampleplottemp,rhoval),'r-',label='Function
    approximation')
        plt.plot(sampleplottemp-273.15,sampleploteps,'r-',label='Function approximation')
        plt.plot(t_mat-273.15,epses_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel(r'$\varepsilon_r$ [unitless]')
        plt.title(r'Relative electric permittivity for zirconia')
        plt.savefig(savestring+'mat_epsfun.png')
        tikz_save(savestring+'mat_epsfun.tex', figureheight = '\\figureheight', figurewidth
            = '\\figurewidth',show_info = False )
        plt.close(10)

        plt.figure(11) # Plot sigma(temp) for material
        plt.clf()
# plt.plot(sampleplottemp-273.15,sigfun_mat(sampleplottemp,rhoval),'b-',label='Function
    approximation')
        plt.plot(sampleplottemp-273.15,sampleplotsig,'b-',label='Function approximation')
        plt.plot(t_mat-273.15,sigmas_mat,'ro',label='Experimental measurements (temp only)'
            )
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$\sigma$ [S/m]')
        plt.title(r'Electrical conductivity for zirconia')
        plt.savefig(savestring+'mat_sigfun.png')
        tikz_save(savestring+'mat_sigfun.tex', figureheight = '\\figureheight', figurewidth
            = '\\figurewidth',show_info = False )
        plt.close(11)

        plt.figure(12) # Plot c_p(temp) for material
        plt.clf()
        #plt.plot(sampleplottemp-273.15,cfun_mat(sampleplottemp,rhoval),'g-',label='
            Function approximation')
        plt.plot(sampleplottemp-273.15,sampleplotc,'g-',label='Function approximation')
        plt.plot(t_mat-273.15,cs_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$c_p$ [J/(gK)]')
        plt.title(r'Specific heat capacity for zirconia')
        plt.savefig(savestring+'mat_cfun.png')
        tikz_save(savestring+'mat_cfun.tex', figureheight = '\\figureheight', figurewidth =
            '\\figurewidth',show_info = False )
        plt.close(12)

        plt.figure(13) # Plot k(temp) for material
        plt.clf()
        #plt.plot(sampleplottemp-273.15,kfun_mat(sampleplottemp,rhoval),'b-',label='
            Function approximation')
        plt.plot(sampleplottemp-273.15,sampleplotk,'b-',label='Function approximation')
        plt.plot(t_mat-273.15,ks_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$k$ [W/(mK)]')
        plt.title(r'Thermal conductivity for zirconia')
        plt.savefig(savestring+'mat_kfun.png')
        tikz_save(savestring+'mat_kfun.tex', figureheight = '\\figureheight', figurewidth =
            '\\figurewidth',show_info = False )
        plt.close(13)

        plt.figure(14) # Plot mu(temp) for material
        plt.clf()
        #plt.plot(sampleplottemp-273.15,mufun_mat(sampleplottemp,rhoval),'g-',label='
            Function approximation')
        plt.plot(sampleplottemp-273.15,sampleplotmu,'g-',label='Function approximation')
        plt.plot(t_mat-273.15,mus_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$\mu_r$ [unitless]')
        plt.title(r'Relative magnetic permeability for zirconia')
        plt.savefig(savestring+'mat_mufun.png')
        tikz_save(savestring+'mat_mufun.tex', figureheight = '\\figureheight', figurewidth =
             '\\figurewidth',show_info = False )
        plt.close(14)

        plt.figure(15) # Plot all(temp) for material
        plt.clf()
        plt.plot(sampleplottemp-273.15,sampleploteps,'r-',label='Eps Function')
#       plt.plot(t_mat-273.15,epses_mat,'ro',label='Eps meas')
        plt.plot(sampleplottemp-273.15,sampleplotsig,'b-',label='Sig fn')
#       plt.plot(t_mat-273.15,sigmas_mat,'bo',label='Sig meas')
        plt.plot(sampleplottemp-273.15,sampleplotmu,'g-',label='Mu func')
#       plt.plot(t_mat-273.15,mus_mat,'go',label='Mu meas')
        plt.plot(sampleplottemp-273.15,sampleplotc,'k-',label='cp func')
#       plt.plot(t_mat-273.15,cs_mat,'ko',label='cp meas')
        plt.plot(sampleplottemp-273.15,sampleplotk,'y-',label='k func')
        plt.plot(t_mat-273.15,ks_mat,'yo',label='k meas')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('Fun val')
        plt.title('All functions')
        plt.savefig(savestring+'mat_allfuns.png')
        tikz_save(savestring+'mat_allfuns.tex', figureheight = '\\figureheight', figurewidth
             = '\\figurewidth',show_info = False )
        plt.close(15)


# Functions for insulation parameters (these take temps as inputs)
epsinstck = intp.splrep(t_ins,epses_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline
    interpolation
def epsfun_ins(temp):
        return intp.splev(temp,epsinstck) # spline evaluation

siginstck = intp.splrep(t_ins,sigmas_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline
    interpolation
def sigfun_ins(temp):
        return intp.splev(temp,siginstck) # spline evaluation
```

```

cinstck = intp.splrep(t_ins,cs_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline interpolation
def cfun_ins(temp):
       return intp.splev(temp,cinstck) # spline evaluation

rhoinstck = intp.splrep(t_ins,rhos_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline
    interpolation
def rhofun_ins(temp):
       return intp.splev(temp,rhoinstck) # spline evaluation

kinstck = intp.splrep(t_ins,ks_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline interpolation
def kfun_ins(temp):
       return intp.splev(temp,kinstck) # spline evaluation

muinstck = intp.splrep(t_ins,mus_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline interpolation
def mufun_ins(temp):
       return intp.splev(temp,muinstck) # spline evaluation

# Uncomment to use barycentric interpolation instead of b-splines
#epsfun_ins = intp.BarycentricInterpolator(t_ins,epses_ins)
#sigfun_ins = intp.BarycentricInterpolator(t_ins,sigmas_ins)
#cfun_ins = intp.BarycentricInterpolator(t_ins,cs_ins)
#rhofun_ins = intp.BarycentricInterpolator(t_ins,rhos_ins)
#kfun_ins = intp.BarycentricInterpolator(t_ins,ks_ins)
#mufun_ins = intp.BarycentricInterpolator(t_ins,mus_ins)

plt.figure(20) # Plot eps(temp) for insulation
plt.plot(sampleplottemp-273.15,epsfun_ins(sampleplottemp),'r-',label='Function
    approximation')
plt.plot(t_ins-273.15,epses_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature [degC]')
plt.ylabel(r'$\varepsilon_r$ [unitless]')
plt.title('Relative electric permittivity for alumina insulation')
plt.savefig(savestring+'ins_epsfun.png')
tikz_save(savestring+'ins_epsfun.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )
plt.close(20)

plt.figure(21) # Plot sigma(temp) for insulation
plt.plot(sampleplottemp-273.15,sigfun_ins(sampleplottemp),'b-',label='Function
    approximation')
plt.plot(t_ins-273.15,sigmas_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature [degC]')
plt.ylabel('$\sigma$ [S/m]')
plt.title('Electrical conductivity for alumina insulation')
plt.savefig(savestring+'ins_sigfun.png')
tikz_save(savestring+'ins_sigfun.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )
plt.close(21)

plt.figure(22) # Plot c_p(temp) for insulation
plt.plot(sampleplottemp-273.15,cfun_ins(sampleplottemp),'g-',label='Function approximation
    ')
plt.plot(t_ins-273.15,cs_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature [degC]')
plt.ylabel('$c_p$ [J/(gK)]')
plt.title('Specific heat capacity for alumina insulation')
plt.savefig(savestring+'ins_cfun.png')
tikz_save(savestring+'ins_cfun.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )
plt.close(22)

plt.figure(23) # Plot rho(temp) for insulation
plt.plot(sampleplottemp-273.15,rhofun_ins(sampleplottemp),'y-',label='Function
    approximation')
plt.plot(t_ins-273.15,rhos_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature [degC]')
plt.ylabel(r'$\rho$ [g/(cm^3)]')
plt.title('Density for alumina insulation')
plt.savefig(savestring+'ins_rhofun.png')
tikz_save(savestring+'ins_rhofun.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )
plt.close(23)

plt.figure(24) # Plot k(temp) for insulation
plt.plot(sampleplottemp-273.15,kfun_ins(sampleplottemp),'b-',label='Function approximation
    ')
```

```
plt.plot(t_ins-273.15,ks_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature [degC]')
plt.ylabel('$k$ [W/(mK)]')
plt.title('Thermal conductivity for alumina insulation')
plt.savefig(savestring+'ins_kfun.png')
tikz_save(savestring+'ins_kfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(24)

plt.figure(25) # Plot mu(temp) for insulation
plt.plot(sampleplottemp-273.15,mufun_ins(sampleplottemp),'g-',label='Function approximation')
plt.plot(t_ins-273.15,mus_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature [degC]')
plt.ylabel('$\mu_r$ [unitless]')
plt.title('Relative magnetic permeability for alumina insulation')
plt.savefig(savestring+'ins_mufun.png')
tikz_save(savestring+'ins_mufun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(25)

printstring=("\n\tDone; took "+str(time.clock()-starttime)+" seconds to find functions for all dielectric and thermal material and insulation properties.\n\nSetting up simulation...\n")
logfile.write(printstring)

# Initialize elemental values of load properties (these vectors are updated in the course of mechanical solution)
# Since there is no concept of heating rate at time t=0, must start with only temp-dependent interpolated parameters
eps_mat=intp.BarycentricInterpolator(t_mat,epses_mat).__call__(temp_init) #[unitless] relative permittivity
sig_mat=intp.BarycentricInterpolator(t_mat,sigmas_mat).__call__(temp_init) # [S/m] electrical conductivity
c_mat=intp.BarycentricInterpolator(t_mat,cs_mat).__call__(temp_init) # [J/g*C] specific heat capacity
rho_mat=(intp.BarycentricInterpolator(t_mat,rhos_mat).__call__(temp_init)+0.1)*bulkdens_mat # [g/m^3] density; the +0.1 is to make it closer to the density in the data used for MSC
```

```
k_mat=intp.BarycentricInterpolator(t_mat,ks_mat).__call__(temp_init) # [W/g*C] thermal
    conductivity
mu_mat=intp.BarycentricInterpolator(t_mat,mus_mat).__call__(temp_init) # [unitless]
    relative permeability

# Initialize elemental values of insulation properties
eps_ins=epsfun_ins(temp_init) #[unitless] relative permittivity of insulation at initial
    temperature
sig_ins=sigfun_ins(temp_init) # [S/m] electrical conductivity of insulation at initial
    temperature
c_ins=cfun_ins(temp_init) # [J/g*C] specific heat capacity of insulation at initial
    temperature
rho_ins=rhofun_ins(temp_init) # [g/m^3] density of insulation at initial temperature
k_ins=kfun_ins(temp_init) # [W/g*C] thermal conductivity of insulation at initial
    temperature
mu_ins=mufun_ins(temp_init) # [unitless] relative permeability of insulation at initial
    temperature

# Nodes and spacing
delta_x_air = 0.05*c/(f_g*sqrt(eps_air)) # length of spatial step in air [m]
delta_x_mat = 0.05*c/(f_g*sqrt(eps_mat)) # length of spatial step in material [m]
delta_x_ins = 0.05*c/(f_g*sqrt(eps_ins)) # length of spatial step in insulation [m]

ins_left_bdry = 0.5*(L-L_mat)-L_ins # left-hand boundary of insulation [m]
mat_left_bdry = ins_left_bdry + L_ins # left-hand boundary of material [m]
mat_right_bdry = mat_left_bdry + L_mat # right-hand boundary of material [m]
ins_right_bdry = mat_right_bdry + L_ins # right-hand boundary of insulation [m]

left_air_vec = r_[0:ins_left_bdry:delta_x_air]
ins_left_bdry = max(ins_left_bdry,left_air_vec[-1]+delta_x_ins) # makes sure step at
    interface is not too small
left_ins_vec = r_[ins_left_bdry:mat_left_bdry:delta_x_ins]
mat_left_bdry = max(mat_left_bdry,left_ins_vec[-1]+delta_x_mat) # makes sure step at
    interface is not too small
mat_vec = r_[mat_left_bdry:mat_right_bdry:delta_x_mat]
mat_right_bdry = max(mat_right_bdry,mat_vec[-1]+delta_x_mat) # makes sure step at
    interface is not too small
right_ins_vec = r_[mat_right_bdry:ins_right_bdry:delta_x_ins]
ins_right_bdry = max(ins_right_bdry,right_ins_vec[-1]+delta_x_ins) # makes sure step at
    interface is not too small
right_air_vec = r_[ins_right_bdry:L:delta_x_air]
```

```

# Finally, create the vector of x-values
x = r_[ left_air_vec , left_ins_vec , mat_vec , right_ins_vec , right_air_vec ]

n=np.size(x) # number of spatial gridpoints
h=x[1:]-x[:-1] # h-values

n_air_left = np.size(left_air_vec) # the number of nodes containing air in left half of the
     cavity, initially
n_ins_left = np.size(left_ins_vec) # the number of nodes containing insulation in left half
     of the cavity, initially
n_mat = np.size(mat_vec) # the number of nodes containing material, initially
n_ins_right = np.size(right_ins_vec) # the number of nodes containing insulation in right
    half of the cavity, initially
n_air_right = np.size(right_air_vec) # the number of nodes containing air in right half of
    the cavity, initially

n_ins=n_ins_left+n_mat+n_ins_right

ins_startind = n_air_left # first index within left-hand insulation
mat_startind = n_air_left + n_ins_left # first index within material
ins_endind = n - n_air_right # first index within right-hand air
mat_endind = ins_endind-n_ins_right # first index within right-hand insulation

eps = r_[ [eps_air]*n_air_left, [eps_ins]*n_ins_left, [eps_mat]*n_mat, [eps_ins]*
    n_ins_right, [eps_air]*n_air_right ]
sig = r_[ [sig_air]*n_air_left, [sig_ins]*n_ins_left, [sig_mat]*n_mat, [sig_ins]*
    n_ins_right, [sig_air]*n_air_right ]
cp = r_[ [c_air]*n_air_left, [c_ins]*n_ins_left, [c_mat]*n_mat, [c_ins]*n_ins_right, [c_air
    ]*n_air_right ] # called cp to differentiate it from c, the speed of light
rho = r_[ [rho_air]*n_air_left, [rho_ins]*n_ins_left, [rho_mat]*n_mat, [rho_ins]*
    n_ins_right, [rho_air]*n_air_right ]
ks = r_[ [k_air]*n_air_left, [k_ins]*n_ins_left, [k_mat]*n_mat, [k_ins]*n_ins_right, [k_air
    ]*n_air_right ]
mu = r_[ [mu_air]*n_air_left, [mu_ins]*n_ins_left, [mu_mat]*n_mat, [mu_ins]*n_ins_right, [
    mu_air]*n_air_right ]

# UNCOMMENT THESE TO SIMULATE AN EMPTY CAVITY (useful for testing EM solver against known
    TE, TM, and TEM patterns)
#eps = np.array([eps_air]*n)
#sig = np.array([sig_air]*n)
```

```
#cp = np.array([c_air]*n)
#rho = np.array([rho_air]*n)
#ks = np.array([k_air]*n)
#mu = np.array([mu_air]*n)

# Time scenario
#em_dt = min(delta_x_air,delta_x_mat,delta_x_ins)/c # length of time step of em solve [sec]
em_dt = 1.0e-3 #(use this for speeding up the simulation--this is permissible when the
    dielectric properties are not changing too quickly with temperature)
h_dt = 1.0e-2 # length of time step of heat solve (i.e., how long to nuke before solving
    heat transfer) [sec]

# Initialize electric field
E_old = np.array([0]*n) # initialize electric field [V/m]
beta = pi/L # propagation constant [1/m]
pow = (2/L)*sqrt(2*P*omega_fs*mu0/beta) # initialize power at magnetron (left-hand
    boundary)
E_old[0] = pow # replace first E-field value with power at magnetron (left-hand boundary)
E_older = E_old # initialize second oldest electric field
eavg = [0]*n
import emsolve1

# Initialize temperature field
import thermsolve1
temp_old=np.array([temp_init]*n) # K

# Initialize theta
theta_integrand_old = np.array([(np.exp(-Q/(R*temp_init)))/(temp_init)]*(n_mat+1)) # theta
     is a cumulative integral; this is the initial value of the integrand
theta = np.zeros(np.shape(theta_integrand_old)) # initial value of cumulative integral is
    zero

# Initialize material 'volume' (in 1D, volume=length)
v_old = L_mat

# Initialize average density in material
rho_avg_old = rho_mat/bulkdens_mat

# Initialize Plots

#plt.figure(14) # Domain plot
```

```
#plt.plot(100*x,eps,'ro',100*x,sig,'bo',100*x,cp,'go',100*x,rho,'yo',100*x,rho,'rs',100*x,k
    ,'bs',100*x,mu,'gs')
#plt.xlabel('Position along domain [cm]')
#plt.ylabel('param vals')
#plt.title('Geometrical configuration')
#plt.show()

# Initialize time and iterations
loopstarttime=time.clock() # to track computing time
elapsed_time = 0.0 # initialize elapsed simulated processing time
itno=0 # iteration number (for saving frames of movies)
delfiles=[] # for storing names of files containing frames for movies--want to delete
    these files at the end of simulation

if embc=='abs':
       embcprintstring = "absorbing"
elif embc=='pec':
       embcprintstring = "perfect electric conductor (zero Dirichlet)"

if tempbc=='rad':
       tempbcprintstring="radiative"
elif tempbc=='ins':
       tempbcprintstring="insulating (zero Neumann)"
elif tempbc=="fix":
       tempbcprintstring="fixed (Dirichlet)"

printstring = ("\tSpatial cell size in air is "+str(delta_x_air*1e2)+" cm\n\tSpatial cell
    size in insulation "+str(delta_x_ins*1e2)+" cm\n\tSpatial cell size in material is "+
    str(delta_x_mat*1e2)+" cm\n\tTotal number of cells in entire domain is "+str(n)+"\n\
    tTotal number of cells in insulation+material is "+str(n_ins)+"\n\tTotal number of
    cells in material is "+str(n_mat)+"\n\tTime step for electromagnetic solve is "+str(
    em_dt)+" sec\n\tTime step for thermal solve is "+str(h_dt)+" sec\n\tTotal simulated
    processing time will be "+str(total_time)+" sec\n\nStarting simulation loop...\n\
    tUsing "+embcprintstring+" boundary condition for electromagnetic solver\n\tUsing "+
    tempbcprintstring+" boundary condition for thermal solver\n")
logfile.write(printstring)

instemps = r_[temp_old[ins_startind:mat_startind+1],temp_old[mat_endind:ins_endind
    +1]]-273.15
printstring = ('\nAt start of simulation...\n\tMax value of electric field is ' + str(max(
    eavg)) + ' V/m\n\tMin value of electric field is ' + str(min(eavg)) + ' V/m\n\tMean
```

```python
value of electric field is ' + str(mean(eavg))+' V/m\n\tMax temp in insulation is ' + str(max(instemps)) + ' degC\n\tMin temp in insulation is ' + str(min(instemps)) + ' degC\n\tMean temp in insulation is ' + str(mean(instemps)) + ' degC\n\tMax temp in load is ' + str(max(temp_old[mat_startind:mat_endind+1])-273.15) + ' degC\n\tMin temp in load is ' + str(min(temp_old[mat_startind:mat_endind+1])-273.15) + ' degC\n\tMean temp in load is ' + str(mean(temp_old[mat_startind:mat_endind+1])-273.15) + ' degC\n\tMean density in material is '+str(100*rho_mat/bulkdens_mat)+" percent of bulk density\n")
logfile.write(printstring)

T_maxes = np.array([temp_init-273.15])
T_means = np.array([temp_init-273.15])
load_rhos = np.array([100*rho_mat/bulkdens_mat])
plottingtimes = lntavgs = np.array([0])

# Simulation loop
while elapsed_time<total_time:
        print_time = str(round(elapsed_time,2))

        # Plot electric field
        plt.figure(30) # static image of field
        plt.plot(100*x,eavg)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Root mean square of electric field [V^2/m^2]')
        plt.title('RMS of electric field at t=' + print_time + ' seconds')
        plt.draw()
        plt.savefig(savestring+'efield.png')
# tikz_save(savestring+'efield.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )

        plt.figure(31) # save frame for movie
        plt.cla()
        plt.plot(100*x,eavg)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Root mean square of electric field [V^2/m^2]')
        plt.title('RMS of electric field at t=' + print_time + ' seconds')
        fname = hiddensavestring+'efield'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        # Plot temperature field
        plt.figure(32) # update static image of field
        plt.plot(100*x,temp_old-273.15)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Temperature [C]')
        plt.title('Temperature distribution at t=' + print_time + ' seconds')
        plt.draw()
        plt.savefig(savestring+'temp_wholecav.png')
# tikz_save(savestring+'temp_wholecav.tex', figureheight = '\\figureheight', figurewidth =
    '\\figurewidth',show_info = False )

        plt.figure(33) # save frame for movie
        plt.cla()
        plt.axis([100*x[0],100*x[-1],25,1400])
        plt.plot(100*x,temp_old-273.15)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Temperature [C]')
        plt.title('Temperature distribution at t=' + print_time + ' seconds')
        fname = hiddensavestring+'temp_wholecav'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        # Plot shrinkage results
        #plt.figure(34)
        #plt.plot(100*x,) # what to actually plot?
        #plt.xlabel('Position along domain [cm]')
        #plt.ylabel('') # figure out how to remove number labels and ticks from y-axis
        #plt.title('Shrinkage at t=' + print_time + ' seconds')
        #plt.draw()

        #plt.figure(35) # save frame for movie
        #plt.clf()
        #plt.plot(100*x,)
        #plt.xlabel('Position along domain [cm]')
        #plt.ylabel('')
        #plt.title('Shrinkage at t=' + print_time + ' seconds')
        #fname = hiddensavestring+'mechdef'+str(itno)+'.png'
        #plt.savefig(fname)
        #delfiles.append(fname)

        # Plot material properties
        plt.figure(40)
        plt.plot(100*x,eps)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel(r'$\varepsilon_r$ [unitless]')
        plt.title('Relative permittivity at time t=' + print_time + ' seconds')
        plt.savefig(savestring+'eps_evol.png')
# tikz_save(savestring+'eps_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

        plt.figure(41) # save frame for movie
        plt.cla()
        plt.plot(100*x,eps)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel(r'$\varepsilon_r$ [unitless]')
        plt.title('Relative permittivity at time t=' + print_time + ' seconds')
        fname = hiddensavestring+'eps_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(42)
        plt.plot(100*x,sig)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel(r'$\sigma$ [S/m]')
        plt.title('Electrical conductivity at time t='+print_time + ' seconds')
        plt.savefig(savestring+'sig_evol.png')
# tikz_save(savestring+'sig_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

        plt.figure(43) # save frame for movie
        plt.cla()
        plt.plot(100*x,sig)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel(r'$\sigma$ [S/m]')
        plt.title('Electrical conductivity at time t='+print_time + ' seconds')
        fname = hiddensavestring+'sig_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(44)
        plt.plot(100*x,cp)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('$c_p$ [J/(gK)]')
        plt.title('Thermal conductivity at time t='+print_time + ' seconds')
        plt.savefig(savestring+'c_evol.png')
# tikz_save(savestring+'c_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

        plt.figure(45) # save frame for movie
        plt.cla()
        plt.plot(100*x,cp)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('$c_p$ [J/(gK)]')
        plt.title('Thermal conductivity at time t='+print_time + ' seconds')
        fname = hiddensavestring+'c_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(46)
        plt.plot(100*x,rho)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel(r'$\rho$ [g/m^3]')
        plt.title(r'$\rho$ at time t='+print_time + ' seconds')
        plt.savefig(savestring+'rho_evol.png')
# tikz_save(savestring+'rho_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

        plt.figure(47) # save frame for movie
        plt.cla()
        plt.plot(100*x,rho)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel(r'$\rho$ [g/m^3]')
        plt.title(r'$\rho$ at time t='+print_time + ' seconds')
        plt.savefig(hiddensavestring+'rho_evol.png')
        fname = hiddensavestring+'rho_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(48)
        plt.plot(100*x,ks)
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('$k$ [W/(mK)]')
        plt.title('Specific heat capacity at time t='+print_time + ' seconds')
        plt.savefig(savestring+'k_evol.png')
```

```
# tikz_save(savestring+'k_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

      plt.figure(49) # save frame for movie
      plt.cla()
      plt.plot(100*x,ks)
      plt.xlabel('Position along domain [cm]')
      plt.ylabel('$k$ [W/(mK)]')
      plt.title('Specific heat capacity at time t='+print_time + ' seconds')
      fname = hiddensavestring+'k_evol'+str(itno)+'.png'
      plt.savefig(fname)
      delfiles.append(fname)

      plt.figure(50)
      plt.plot(100*x,mu)
      plt.xlabel('Position along domain [cm]')
      plt.ylabel(r'Relative permeability $\mu$ [unitless]')
      plt.title(r'$\mu$ at time t='+print_time + ' seconds')
      plt.savefig(savestring+'mu_evol.png')
# tikz_save(savestring+'mu_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

      plt.figure(51) # save frame for movie
      plt.cla()
      plt.plot(100*x,mu)
      plt.xlabel('Position along domain [cm]')
      plt.ylabel(r'Relative permeability $\mu$ [unitless]')
      plt.title(r'$\mu$ at time t='+print_time + ' seconds')
      fname = hiddensavestring+'mu_evol'+str(itno)+'.png'
      plt.savefig(fname)
      delfiles.append(fname)

      loopits = 0
      staycount = 0
      leftcount = 0
      rightcount = 0
      # Run coupled solver
      while loopits<capture_every/h_dt: # print shrinkage every (this many) timesteps,
          instead of every single timestep (avoid creating huuuuuuge logfiles)

            # Iterate electromagnetic solver for the duration of one thermal timestep
            E_old,E_older,eavg = emsolve1.finite_diff_implicit(E_old,E_older,x,mu,sig,eps,h,em_dt,h_dt,elapsed_time,embc) # returns modulus of electric field

            # Run thermal solver once, and *only* within the insulation and load
            mat_start = mat_startind-ins_startind
            mat_end = mat_endind-ins_startind

            temp_new = thermsolve1.finite_diff_theta(temp_old[ins_startind:ins_endind],h[ins_startind-1:ins_endind],cp[ins_startind:ins_endind],rho[ins_startind:ins_endind],ks[ins_startind:ins_endind],eavg[ins_startind:ins_endind],h_dt,sig[ins_startind:ins_endind],th,tempbc,trans_ins,temp_init) # returns new temperature field

            # Find theta values corresponding to new temperatures and heating rates *at each point in load*

            theta_integrand_new = (np.exp(-Q/(R*temp_new[mat_start:mat_end+1])))/(temp_new[mat_start:mat_end+1])
            theta = theta + 0.5*(theta_integrand_old + theta_integrand_new)*h_dt
            theta_integrand_old = theta_integrand_new
            lntheta = np.log(theta)
            lnt_avg = np.mean(lntheta)

            # Update density in load using MSC and computed theta

            rho_mat = rhofun(lntheta)

            rho_avg_new = np.mean(rho_mat)
            rho_mat = rho_mat * bulkdens_mat

            # Find index of the maximum density value in load (if there is more than one max, this is the left-most one)

            rho_max = mat_startind+np.argmax(rho_mat) # index (IN THE X-VECTOR NUMBERING) of max dens

            # Compute total shrinkage within material based on density change and conservation of mass

            v_new = v_old*rho_avg_old/rho_avg_new # new volume of material
            x_thr = x[rho_max]-v_old+v_new # material between x_thr and rho_max disappears

            # Update temperature in insulation + material
            temp_old[ins_startind:ins_endind] = temp_new # update temperature in insulation and material

            if x_thr > x[rho_max-1]: # x_thr is less than one spatial step from rho_max
                   # Then don't change the volume this time around; wait for more possible shrinkage in next time step
                   v_new = v_old
                   printstring = "\tShrinkage is less than the length of a single spatial step in material; not simulating shrinkage\n\tPercent of original length remains "+str(100*v_new/L_mat)+"\n\tNumber of nodes within material remains "+str(mat_endind-mat_startind)+"\n"
                   #logfile.write(printstring)
                   staycount = staycount + 1

            elif x_thr > 5*L/9-v_old: # x_thr is more than one spatial step from rho_max, but still within material
                   the_ind = np.max(np.where(x<x_thr)) # index just to left of x_thr
                   v_new = v_old-(x[rho_max]-x[the_ind]) # the actual new volume (as x_thr likely landed between nodes) # THIS VIOLATES CONSERVATION OF MASS, BUT IF SPATIAL GRID SIZE IS SMALL ENOUGH, IT SHOULDN'T BE "TOO" WRONG
                   shrink = rho_max-the_ind # number of nodes to shrink by
                   printstring = "\tShrinkage by deleting material to the left of max density\n\tMaterial shrinks by "+str(shrink)+" nodes ("+str((v_old-v_new)*100)+" cm)\n\tNew length is "+str(100*v_new/L_mat)+" percent of original length\n\tNumber of nodes remaining in material is "+str(mat_endind-mat_startind-shrink)+"\n"
                   #logfile.write(printstring)
                   leftcount = leftcount + 1
                   temp_old[mat_startind+shrink:rho_max+1]=temp_old[mat_startind:rho_max+1-shrink] # remove material between x_thr and rho_max, and shift remaining load material to right
                   temp_old[ins_startind+shrink:mat_startind+shrink]=temp_old[ins_startind:mat_startind] # shift insulation to right
                   temp_old[ins_startind:ins_startind+shrink]=temp_init # add air before insulation
                   rho[mat_startind:mat_endind+1] = rho_mat # update load densities
                   rho[mat_startind+shrink:rho_max+1]=rho[mat_startind:rho_max+1-shrink] # remove section from rho
                   theta = np.r_[theta[:rho_max-mat_startind-shrink+1],theta[rho_max+1-mat_startind:]] # remove section from theta
                   lntheta = np.r_[lntheta[:rho_max+1-mat_startind-shrink],lntheta[rho_max+1-mat_startind:]] # remove section from lntheta
                   theta_integrand_old = np.r_[theta_integrand_old[:rho_max+1-mat_startind-shrink],theta_integrand_old[rho_max-mat_startind:]] # remove section from old theta integrand
                   ins_startind = ins_startind+shrink # update insulation start index
                   mat_startind = mat_startind + shrink # update insulation end index
                   v_old = v_new

            elif x_thr < 5*L/9-v_old: # x_thr is *outside* the material! (*Lots* of shrinkage, or rho_max v close to bdry)
                   # Just get rid of enough material to the right of mat_startind
                   x_thr = 5*L/9-v_old+v_new
                   the_ind = np.min(np.where(x>x_thr)) # index just to the right of x_thr
                   v_new = v_old - (x[the_ind]-x[mat_startind]) # the actual new volume (x_thr likely btwn nodes) # VIOLATES CONSERVATION OF MASS, MAKE SURE SPATIAL GRID SIZE IS SMALL
                   shrink = the_ind+1-mat_startind # number of nodes to shrink by
                   printstring = "\tShrinkage by deleting material to the right of left-hand boundary\n\tMaterial shrinks by "+str(shrink)+" nodes ("+str((v_old-v_new)*100)+" cm)\n\tNew length is "+str(100*v_new/L_mat)+" percent of original length\n\tNumber of nodes remaining in material is "+str(mat_endind-mat_startind-shrink)+"\n"
                   #logfile.write(printstring)
                   rightcount = rightcount + 1
                   temp_old[ins_startind+shrink:mat_startind+shrink+1] = temp_old[ins_startind:mat_startind+1]
                   temp_old[ins_startind:ins_startind+shrink] = temp_init
                   rho[mat_startind+shrink:mat_endind+1] = rho_mat[shrink:]
                   theta = theta[shrink:]
                   lntheta = lntheta[shrink:]
                   theta_integrand_old = theta_integrand_old[shrink:]
                   mat_startind = mat_startind+shrink
                   ins_startind = ins_startind+shrink
                   v_old = v_new

            # Update material parameters, including dielectric properties, according to
                temperature change, density change, and shrinkage
            # Load parameters
            if theta_dep_params: # then load parameters depend on theta
                   eps[mat_startind:mat_endind+1] = epsfun_mat(lntheta)
                   sig[mat_startind:mat_endind+1] = sigfun_mat(lntheta)
                   cp[mat_startind:mat_endind+1] = cfun_mat(lntheta)
                   ks[mat_startind:mat_endind+1] = kfun_mat(lntheta)
                   mu[mat_startind:mat_endind+1] = mufun_mat(lntheta)
            else: # then load parameters depend on temp and rho
                   eps[mat_startind:mat_endind+1] = epsfun_mat(temp_old[mat_startind:
                       mat_endind+1],rho[mat_startind:mat_endind+1]/bulkdens_mat)
                   sig[mat_startind:mat_endind+1] = sigfun_mat(temp_old[mat_startind:
                       mat_endind+1],rho[mat_startind:mat_endind+1]/bulkdens_mat)
                   cp[mat_startind:mat_endind+1] = cfun_mat(temp_old[mat_startind:
                       mat_endind+1],rho[mat_startind:mat_endind+1]/bulkdens_mat)
                   ks[mat_startind:mat_endind+1] = kfun_mat(temp_old[mat_startind:
                       mat_endind+1],rho[mat_startind:mat_endind+1]/bulkdens_mat)
                   mu[mat_startind:mat_endind+1] = mufun_mat(temp_old[mat_startind:
                       mat_endind+1],rho[mat_startind:mat_endind+1]/bulkdens_mat)

            # Insulation parameters
            eps[ins_startind:mat_startind+1] = epsfun_ins(temp_old[ins_startind:
                   mat_startind+1])
            eps[mat_endind:ins_endind] = epsfun_ins(temp_old[mat_endind:ins_endind])
            sig[ins_startind:mat_startind+1] = sigfun_ins(temp_old[ins_startind:
                   mat_startind+1])
            sig[mat_endind:ins_endind] = sigfun_ins(temp_old[mat_endind:ins_endind])
            rho[ins_startind:mat_startind+1] = rhofun_ins(temp_old[ins_startind:
                   mat_startind+1])
            rho[mat_endind:ins_endind] = rhofun_ins(temp_old[mat_endind:ins_endind])
            cp[ins_startind:mat_startind+1] = cfun_ins(temp_old[ins_startind:mat_startind
                   +1])
            cp[mat_endind:ins_endind] = cfun_ins(temp_old[mat_endind:ins_endind])
            ks[ins_startind:mat_startind+1] = kfun_ins(temp_old[ins_startind:mat_startind
                   +1])
            ks[mat_endind:ins_endind] = kfun_ins(temp_old[mat_endind:ins_endind])
            mu[ins_startind:mat_startind+1] = mufun_ins(temp_old[ins_startind:
                   mat_startind+1])
            mu[mat_endind:ins_endind] = mufun_ins(temp_old[mat_endind:ins_endind])

```

```
                # Air parameters
                eps[:ins_startind] = eps_air
                sig[:ins_startind] = sig_air
                rho[:ins_startind] = rho_air
                cp[:ins_startind] = c_air
                ks[:ins_startind] = k_air
                mu[:ins_startind] = mu_air

                # Eventually: update spatial grid size here, too, if necessary, depending on whether the dielectric properties make the wavelength in material significantly shorter...

                elapsed_time = elapsed_time + h_dt
                loopits = loopits + 1

        # Print text version of results to logfile
        printstring = ('\nAt time ' + str(elapsed_time) + ' sec...\n\tMax value of electric field is ' + str(max(eavg)) + ' V/m\n\tMin value of electric field is ' + str(min(eavg)) + ' V/m\n\tMean value of electric field is ' + str(mean(eavg))+' V/m\n\tMax temp in insulation is ' + str(max(r_[temp_old[ins_startind:mat_startind],temp_old[mat_endind:ins_endind]])-273.15) + ' degC\n\tMin temp in insulation is ' + str(min(r_[temp_old[ins_startind:mat_startind],temp_old[mat_endind:ins_endind]])-273.15) + ' degC\n\tMean temp in insulation is ' + str(mean(r_[temp_old[ins_startind:mat_startind],temp_old[mat_endind:ins_endind]])-273.15) + ' degC\n\tMax temp in load is ' + str(max(temp_old[mat_startind:mat_endind])-273.15) + ' degC\n\tMin temp in load is ' + str(min(temp_old[mat_startind:mat_endind])-273.15) + ' degC\n\tMean temp in load is ' + str(mean(temp_old[mat_startind:mat_endind])-273.15) + ' degC\n\tMean density in material is '+str(100*rho_avg_new)+" percent of bulk density\n\tSince last printed results, material boundary did not change "+str(staycount)+" times\n\tSince last printed results, material immediately to the right of boundary was removed "+str(rightcount)+" times\n\tSince last printed results, material immediately to the left of maximum density was removed "+str(leftcount)+" times\n\tNew material length is "+str(100*v_new/L_mat)+" percent of original length\n\tNumber of nodes remaining in material is "+str(mat_endind-mat_startind)+"\n")
        logfile.write(printstring)
        T_maxes = np.r_[T_maxes, np.max(temp_old[mat_startind:mat_endind])-273.15 ]
        T_means = np.r_[T_means, np.mean(temp_old[mat_startind:mat_endind])-273.15 ]
        load_rhos = np.r_[load_rhos,100*rho_avg_new]
        plottingtimes = np.r_[plottingtimes, elapsed_time]
        lntavgs = np.r_[lntavgs, lnt_avg]

        # Save state of simulation in case we need to pick up later where we left off
        statefile = open(savestring+"state_of_sim.pckl","w")
        pickle.dump([E_old, E_older, eavg, temp_old, eps, sig, rho, cp, ks, mu, mat_startind
            , mat_endind, ins_startind, ins_endind, h, theta_integrand_old, bulkdens_mat,
            rho_avg_old, L_mat, v_old, T_maxes, T_means, load_rhos, lntavgs, plottingtimes,
             itno, elapsed_time, h_dt, em_dt, tempbc, embc, th, trans_ins, temp_init, Q ],
            statefile)
        statefile.close()
        # Pick up later with:
        # statefile = open(savestring+"state_of_sim.pckl","r")
        # E_old, E_older, eavg, temp_old, eps, sig, rho, cp, ks, mu, mat_startind,
            mat_endind, ins_startind, ins_endind, h, theta_integrand_old, bulkdens_mat,
            rho_avg_old, L_mat, v_old, T_maxes, T_means, load_rhos, lntavgs, plottingtimes,
             itno, elapsed_time, h_dt, em_dt, tempbc, embc, th, trans_ins, temp_init, Q =
            pickle.load(statefile)
        # statefile.close()

        itno = itno+1

completetime = time.clock()
printstring = "\n\nSimulation complete. Took "+str(completetime-loopstarttime)+" seconds
    to complete simulation loop\n\nSaving animations...\n"
logfile.write(printstring)

# Plot evolution of maximum temperature in load
plt.figure(60)
plt.plot(plottingtimes,T_maxes,'r-',label='Max temp in load')
plt.plot(plottingtimes,T_means,'b-',label='Mean temp in load')
plt.legend(loc='lower right')
plt.xlabel('Time [sec]')
plt.ylabel('Temperature [degC]')
plt.title('Evolution of mean and maximum temperature in load')
plt.savefig(savestring+'temp_evol.png')
tikz_save(savestring+'temp_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

# Plot evolution of density wrt time
plt.figure(61)
plt.plot(plottingtimes,load_rhos)
plt.xlabel('Time [sec]')
plt.ylabel('Density relative to bulk solid density [%]')
plt.title('Evolution of load density')
plt.savefig(savestring+'dens_time_evol.png')
tikz_save(savestring+'dens_time_evol.tex', figureheight = '\\figureheight', figurewidth = '
    \\figurewidth',show_info = False )

# Plot evolution of density wrt lnTheta
plt.figure(62)
plt.plot(lntavgs[1:],load_rhos[1:])
plt.xlabel(r'$\ln(\Theta(t,T(t)))$ $\left[\ln(\frac{s}{K})\right]$')
plt.ylabel('Density relative to bulk solid density [%]')
plt.title('Evolution of load density')
plt.savefig(savestring+'dens_lnt_evol.png')
tikz_save(savestring+'dens_lnt_evol.tex', figureheight = '\\figureheight', figurewidth = '
    \\figurewidth',show_info = False )

# Plot final electric field
plt.figure(63) # static image of field
plt.plot(100*x,eavg)
plt.xlabel('Position along domain [cm]')
plt.ylabel('Root mean square of electric field [V$^2$/m$^2$]')
plt.title('RMS of electric field at t=' + str(elapsed_time+h_dt) + ' seconds')
plt.draw()
plt.savefig(savestring+'efieldfin.png')
tikz_save(savestring+'efieldfin.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

# Plot final temperature field
plt.figure(64) # update static image of field
plt.plot(100*x,temp_old-273.15)
plt.xlabel('Position along domain [cm]')
plt.ylabel('Temperature [C]')
plt.title('Temperature distribution at t=' + print_time + ' seconds')
plt.draw()
plt.savefig(savestring+'tempfin_wholecav.png')
tikz_save(savestring+'tempfin_wholecav.tex', figureheight = '\\figureheight', figurewidth =
     '\\figurewidth',show_info = False )

startmovieclock = time.clock()
# Make movies
#-framerate : number of frames (images) per second
```

```
#-c:v libx264 - the video codec is libx264 (H.264).
#-profile:v high - use H.264 High Profile (advanced features, better quality).
#-crf 20 - constant quality mode, very high quality (lower numbers are higher quality, 18
    is the smallest you would want to use).
#-pix_fmt yuv420p - use YUV pixel format and 4:2:0 Chroma subsampling

# Electric field
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'efield%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'efield.mp4')
logfile.write("\tSaved electric field animation\n")

# Temperature field
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'temp_wholecav%d.png -y -loglevel
    quiet -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'
    temp_wholecav.mp4')
logfile.write("\tSaved temperature field animation\n")

# Mechanical deformation
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'efield%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'efield.mp4')
logfile.write("\tSaved mechanical deformation animation\n")

# Permittivity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'eps_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'eps_evol.mp4')
logfile.write("\tSaved permittivity animation\n")

# Electrical conductivity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'sig_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'sig_evol.mp4')
logfile.write("\tSaved electrical conductivity animation\n")

# Density
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'rho_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'rho_evol.mp4')
logfile.write("\tSaved density animation\n")

# Thermal conductivity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'c_evol%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'c_evol.mp4')
logfile.write("\tSaved thermal conductivity animation\n")
```

```

# Specific heat capacity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'k_evol%d.png -y -loglevel quiet -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'k_evol.mp4')
logfile.write("\tSaved specific heat capacity animation\n")

# Permeability
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'mu_evol%d.png -y -loglevel quiet -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'mu_evol.mp4')
logfile.write("\tSaved magnetic permeability animation\n")

for fname in delfiles:
      os.remove(fname)
logfile.write('\tDeleted individual frame files.\n\tDone; took '+str(time.clock()-startmovieclock)+' seconds to complete movie processing')

finaltime=time.clock()
printstring = "\n\nSimulation completed on "+time.strftime("%A, %B %d, %Y")+" at "+time.strftime("%H:%M:%S %Z")+".\nTook "+str(finaltime-initialstarttime)+" seconds to complete entire simulation.\n"
logfile.write(printstring)

logfile.close()
print('Simulation complete; see directory '+savedir+' for plots and text outputs of results.')
```

H.2 python Implementation of the Coupled Solver for the Two-Dimensional Microwave Sintering Problem

```
#!/usr/bin/python

# Performs transient solution for the electric field in a two-dimensional domain with a constant power source at the left-hand wall. See problem description in file (Thesis.pdf). Simulation domain has chunk of cavity filled with insulation, and chunk of insulation filled with material for processing.
#
# outputs: graphs of temperature and root mean square of electric field, full sets of dielectric and thermal properties, and file fullsolve2.log with detailed output at each timestep
#
```

```
# This code requires python 2.7 , and requires ffmpeg or avconv (may need to modify movie-
    making parts, depending on your system). ffmpeg can be installed by typing
#
# > sudo apt-get install ffmpeg
#
# and avconv may be installed by typing
#
# > sudo apt-get install avconv
#
#
    ------------------------------------------------------------------------------
# DEPENDENCY TREE:
#
# - fullsolve2.py
# - matplotlib ( available from http://matplotlib.org/ )
#  - pyplot ( packaged with matplotlib; documentation available at http://matplotlib.org/
    api/pyplot_summary.html )
# - matplotlib2tikz ( available from https://github.com/nschloe/matplotlib2tikz )
# - numpy ( available from http://www.numpy.org/ )
#  - matlib ( packaged with numpy; documentation available at http://docs.scipy.org/doc/
    numpy/reference/routines.matlib.html )
# - scipy ( available at http://www.scipy.org/ )
#  - interpolate ( packaged with scipy; documentation available at http://docs.scipy.org/
    doc/scipy/reference/tutorial/interpolate.html )
# - time ( module packaged with python 2.7 ; documentation available at https://docs.
    python.org/2/library/time.html )
# - os ( module packaged with python 2.7 ; documentation available at https://docs.python.
    org/2/library/os.html )
# - msc.py ( available from Erin Kiley , emkiley@wpi.edu )
#  - scipy ( available at http://www.scipy.org/ )
#   - optimize ( packaged with scipy; documentation available at http://docs.scipy.org/doc
    /scipy/reference/tutorial/optimize.html )
#    - minimize ( packaged with optimize; documentation available at http://docs.scipy.org
    /doc/scipy/reference/generated/scipy.optimize.minimize.html )
#    - curve-fit ( packaged with optimize; documentation available at http://docs.scipy.
    org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html )
#   - integrate ( packaged with scipy; documentation available at http://docs.scipy.org/
    doc/scipy/reference/tutorial/integrate.html )
#  - numpy ( available from http://www.numpy.org/ )
#  - matplotlib ( available from http://matplotlib.org/ )
#   −pyplot ( packaged with matplotlib; documentation available at http://matplotlib.org/
    api/pyplot_summary.html )
#  − matplotlib2tikz ( available from https://github.com/nschloe/matplotlib2tikz )
#  − itertools ( module packaged with python 2.7 ; documentation available at https://docs
    .python.org/2/library/itertools.html )
#  − sys ( module packaged with python 2.7 ; documentation available at https://docs.
    python.org/2/library/sys.html )
# − emsolve2.py ( available from Erin Kiley , emkiley@wpi.edu )
#  − scipy ( available at http://www.scipy.org/ )
#   − sparse ( packaged with scipy; documentation available at http://docs.scipy.org/doc/
    scipy/reference/sparse.html )
# − thermsolve2.py ( available from Erin Kiley , emkiley@wpi.edu )
#  −scipy ( available at http://www.scipy.org/ )
#   − sparse ( packaged with scipy; documentation available at http://docs.scipy.org/doc/
    scipy/reference/sparse.html )
#
#
    −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−


total_time = 3600 # total processing time [sec]
print_every = 100
theta_dep_params = True # If false, then invert Lichtenecker's formula0
mat_params = "Licht"
magmat = False
method = "fantozzi" # Sigmoid function to use in curve fitting 'blaine' or 'fantozzi'
embc='abs' # EM bc at RH wall, either 'abs' for absorbing (inhomogeneous Neumann) boundary
    condition at right−hand endpoint, or 'pec' for perfect electric conductor (homogeneous
     Dirichlet) condition.
tempbc='ins' # Heat bc, either 'rad' for radiative or 'ins' for insulated
th = phi = 0.5 # th and phi for finite difference method (th,phi = 1 for fully implicit, th
    ,phi = 0 for explicit, th,phi = 0.5 for Crank−Nicolson) for heat *and* EM eqns
savedir = "./2d_demo_"+tempbc+"_helm_"+str(total_time)+"_apr12_75/" # directory where we
    save plots and logfiles
saveprefix="2d_demo_" # prefix for plots and logfile names
savestring = savedir+saveprefix
hiddensavestring = savedir+'.'+saveprefix # for hiding the individual movie frames we save

# Import necessary packages
#from pylab import * # of these, we use numpy, scipy, and matplotlib
import matplotlib as mpl # access matplotlib via shorter 'mpl' prefix
```

```python
import matplotlib.pyplot as plt # plotting library: 'plt' prefix
from matplotlib2tikz import save as tikz_save # for getting a file with tikz data to plot
    directly in thesis
import numpy as np # numpy: 'np' prefix
from numpy import * # we use a number of functions and want to make available at toplevel
#import numpy.matlib as M # make matlib function available at the top level via M.func()
from numpy.matlib import rand,zeros,ones,empty,eye # make these functions accessible
    directly at top level, because we use them
import scipy.interpolate as intp # interpolators: 'intp' prefix. We use b-splines in this
    code.
import time # for printing times to logfile
import os # for issuing commands related to movie-making and auxfile-deleting
mpl.rcParams['axes.formatter.useoffset']=False # tell matplotlib not to convert axis tick
    labels to scientific notation (was getting weird results)
import emsolve2 # electromagnetic solvers (I wrote these; keep in same directory or add to
    path)
import thermsolve2 # thermal solvers (I wrote these; keep in same directory or add to path)
import msc # master sintering curve solvers (I wrote these; keep in same directory or add
    to path)


# Open log file for writing
if not os.path.exists(savedir): # if savedir doesn't already exist
   os.makedirs(savedir) # then create it
logfile = open(savestring+'fullsolve2.log','w+')

# Log file header
printstring = ("Simulation started "+time.strftime("%A, %B %d, %Y")+" at "+time.strftime("%
    H:%M:%S %Z")+".\n\n")
logfile.write(printstring)
initialstarttime = time.clock()

def mat2vec(mat): # reshapes spatial domain matrix into vector
       return np.reshape(np.flipud(mat),np.size(mat))

def vec2mat(vec,N,M): # reshapes spatial domain vector into matrix
       # should have N*M = np.size(vec)
       return np.flipud(np.reshape(vec,(N,M)))

# Important constants
mu0=pi*4e-7 # permeability of free space [N/A^2]
```

```
c = 299792458.0 # speed of light [m/s]
c_zero = c
eps0=1/(mu0*c**2) # permittivity of free space [F/m]
R = 8.314459848 # ideal gas constant [J/(mol*K)]

# Microwave scenario
P=1000.0 # power [W] supplied by magnetron at left-hand endpoint
a = 86.36e-3 # length of long side of cross-section of 3D waveguide [m] --this value
    corresponds to D-band, WR-340 waveguide
b = 43.18e-3 # length of short side of cross-section of 3D waveguide [m]
n_mod = 1 # corresponds to TE_nm excitation mode
m_mod = 0 # corresponds to TE_nm excitation mode

f_fs = 2.45e9 # frequency [Hz] of waves in free space
omega_fs = 2*pi*f_fs # angular frequency [Hz] or [rad/sec] of waves in free space
l_fs = c/f_fs # wavelength [m] in free space
omega_c = c*sqrt( (n_mod*pi/a)**2 + (m_mod*pi/b)**2 ) # angular cutoff frequency [Hz] or [
    rad/sec]
f_c = omega_c/(2*pi) # cutoff frequency [Hz]
# TO DO: Throw a warning if freespace frequency is less than cutoff: then we have
    evanescent TE_10 mode (wave doesn't propagate)
l_c = c/f_c # cutoff wavelength [m]
l_g = sqrt(1/((1/l_fs)**2 - (1/l_c)**2)) # wavelength [m] in waveguide
f_g = c/l_g # frequency [Hz] in waveguide
omega_g = 2*pi*f_g # angular frequency [Hz] or [rad/sec] in waveguide

# Initial temperature
temp_init = 298.0 # room temperature (in kelvin)

# Air properties
eps_air=1.0 #[unitless] relative permittivity of air
sig_air=0.0 # [S/m] electrical conductivity of air
c_air=1.0 # [J/g*C] specific heat capacity of air
rho_air=2.0 # [g/m^3] density of air
k_air=0.024 # [W/g*C] thermal conductivity of air
mu_air=1.0 # [unitless] relative permeability of air

# Physical setup
L = 2.5*l_g # length of waveguide [m], set here equal to 2.5* wavelength in guide, so in
    the unloaded wg, using effective frequency of loaded, we have 5 peaks with one in the
    center (where the sample will be)
```

```
H = 86.36/1000.0 # height of waveguide [m]

ell_1 = L/3 # start of insulation
ell_2 = 4*L/9 # start of material
ell_3 = 5*L/9 # end of material
ell_4 = 2*L/3 # end of insulation
h_1 = H/9 # start of insulation
h_2 = H/3 # start of material
h_3 = 4*H/9 # end of material
h_4 = 2*H/3 # end of insulation

printstring=("Waveguide length is "+str(L*1e2)+" cm\nWaveguide height is "+str(H*1e2)+" cm\nLength of insulation + material is "+str((ell_4-ell_1)*1e2)+" cm\nHeight of insulation + material is "+str((h_4-h_1)*1e2)+" cm\nLength of material is "+str((ell_3-ell_2)*1e2)+" cm\nHeight of material is "+str((h_3-h_2)*1e2)+" cm\nInput power is "+str(P/1000)+" kW\nFrequency of radiation is "+str(f_fs*1e-9)+" GHz\nInitial temperature is "+str(temp_init-273.15)+" degC\n")
logfile.write(printstring)


#############################################

# Load material: zirconia, bulk density of solid material [g/m^3], taken from {}.
bulkdens_mat = 6.52e6

# Load material: zirconia, experimental results taken from {McCoyThesis}. These are the ones used in determining activation energy and MSC.
# First trial: 1 degC/min
times_1 = 1.00*np.array([17192,20134,23142,26147,29086,32027,35033,38038,41046,44052,46993])
temps_1 = 273.15+np.array([900,950,1001,1051,1101,1150,1201,1250,1300,1350,1400])
rhos_1 = 0.01*np.array([46.7,47.1,48.3,51.8,58.6,69.7,82.2,89.7,91.0,91.3,91.4])

# Second trial: 3 degC/min
times_3 = 1.00*np.array([12086,13071,14016,15000,16023,17008,17992,19015,20000,21062,22086])
temps_3 = 273.15+np.array([901,951,999,1049,1101,1151,1199,1251,1300,1350,1400])
rhos_3 = 0.01*np.array([46.6,46.8,47.6,49.8,54.8,63.5,75.4,85.0,87.2,87.8,88.2])

# Third trial: 5 degC/min
```

```
#times_5 = 1.00*np.array
    ([11271,11818,12398,12978,13559,14140,14754,15335,15916,16564,17247])
#temps_5 = 273.15+np.array([901,949,1000,1049,1099,1149,1201,1250,1299,1351,1400])
#rhos_5 = 0.01*np.array([46.6,46.8,47.5,49.4,53.6,61.3,72.7,82.0,84.5,85.3,85.9])

# Load material: zirconia, results taken from {Teng et al}. These are the ones used in
    determining activation energy and the MSC.
# First trial: 2 degC/min
times_2 = 1.0*np.array([9975,11475,12975,14475,15975,17475,18975,20475,21975,23745,25275])
    # times at which the temperatures were measured for the first experiment [s]
temps_2 = 273.15+np.array([1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550]) #
    temperatures for the first experiment[C]
rhos_2 = 0.01*np.array([54.43,55.7,60.15,67.53,76.40,85.35,92.71,96.42,97.63,98.79,98.89])
    # relative densities for the first experiment[%]

# Second trial: 5 degC/min
times_5 = 1.0*np.array([12360,13080,13560,14160,14760,15360,15960,16560,17160,17760,19560])
     # times at which the temperatures were measured for the second experiment
temps_5 = 273.15+np.array([1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550]) #
    temperatures for the second experiment
rhos_5 = 0.01*np.array([53.86,55.69,58.01,64.06,72.50,81.44,90.69,94.67,96.46,97.31,98.40])
     # relative densities for the second experiment

# Third trial: 8 degC/min
times_8 = 1.0*np.array([7725,8100,8475,8850,9225,9600,9975,10650,10725,11100,12900]) #
    times at which the temperatures were measured for the first experiment
temps_8 = 273.15+np.array([1050,1100,1150,1200,1250,1300,1350,1400,1450,1500,1550]) #
    temperatures for the third experiment
rhos_8 = 0.01*np.array([53.75,54.82,57.14,61.05,69.43,77.40,87.34,93.01,95.10,97.51,99.03])
     # relative densities for the third experiment

# Load material: zirconia, experimental results taken from {Yakovlev & Ceralink}. These are
     the ones used in creating property-update functions for everything *except* density,
    in case we rely on the mixture formulas. (In case we rely on the function-of-theta
    approximation, then we construct a separate sigmoid approximation for all parameters
    from this data here, and we use only the activation energy calculated using the data
    from {McCoyThesis}.
times_mat=0.5*np.array([0, 69, 100, 139, 181, 228, 276, 324, 371, 420, 471, 523, 574, 636,
    698, 752, 809, 865, 921, 973, 1019, 1065, 1100]) # times [sec] at which each of (temp,
    eps,sig,c,rho,k) was measured for load material # Currently assumes constant heating
    rate of 2 degC/sec
```

```
t_mat=273.15+np.array([25, 69, 100, 139, 181, 228, 276, 324, 371, 420, 471, 523, 574, 636,
    698, 752, 809, 865, 921, 973, 1019, 1065, 1100]) # temperatures [C->K] at which each
    of (eps,sig,c,rho,k) was measured for load material
epses_mat=np.array([6.69, 5.86, 5.78, 5.75, 5.77, 5.82, 5.90, 5.98, 6.08, 6.18, 6.32, 6.47,
     6.60, 6.77, 6.97, 7.22, 7.53, 7.93, 8.53, 9.44, 10.46, 12.46, 14.77]) # [unitless]
sigmas_mat=np.array([0.0258, 0.0045, 0.0033, 0.0029, 0.0036, 0.0043, 0.0050, 0.0058,
    0.0078, 0.0121, 0.0185, 0.0288, 0.0442, 0.0664, 0.0975, 0.1416, 0.2003, 0.2786,
    0.4083, 0.5942, 0.8220, 1.2190, 1.6661]) # [S/m]
cs_mat=np.array([0.217, 0.324, 0.363, 0.398, 0.426, 0.450, 0.470, 0.487, 0.501, 0.514,
    0.526, 0.537, 0.547, 0.558, 0.568, 0.575, 0.583, 0.590, 0.597, 0.603, 0.607, 0.612,
    0.615]) # [J/(g C)]
rhos_mat=1.0e6*np.array([2.848, 2.844, 2.841, 2.838, 2.834, 2.830, 2.826, 2.821, 2.817,
    2.813, 2.809, 2.804, 2.800, 2.794, 2.789, 2.785, 2.780, 2.775, 2.770, 2.766, 2.762,
    2.758, 2.755])/bulkdens_mat # [g/m^3]
#rhos_mat = rhos_mat[::-1] # ZIRCONIA ACTUALLY SHOWS NO DENSIFICATION AT ALL DURING THIS
    TRIAL... IT SHOWS THERMAL EXPANSION... SO WE FLIP DENSITY VECTOR TO PRETEND THE DAMNED
     THING IS DENSIFYING, EVEN IF JUST A LITTLE BIT
ks_mat=100.0*np.array([0.00198, 0.00290, 0.00320, 0.00344, 0.00362, 0.00373, 0.00381,
    0.00385, 0.00381, 0.00391, 0.00399, 0.00407, 0.00414, 0.00405, 0.00412, 0.00417,
    0.00421, 0.00426, 0.00430, 0.00433, 0.00436, 0.00439, 0.00441]) # [W/(m C)]
mus_mat=np.ones(shape(t_mat))


# Insulation material: alumina, heat transfer coefficient of insulation material, taken
    from {}
# used only in the case of radiative BC for heat eq'n
trans_ins = 500.0

# Insulation material: alumina, parameters taken from {Yakovlev & Ceralink}. These will be
    used to determine polynomial functions for updating temperature-dependent values (we
    do not assume insulation properties are density-dependent)
t_ins=273.0+np.array([25,100,200,300,400,500,600,700,809,900,1000,1100]) # temperatures [C
    ->K] at which each of (eps,sig,c,rho,k) was measured for insulation
epses_ins=np.array([1.520, 1.520, 1.517, 1.513, 1.523, 1.540, 1.563, 1.573, 1.584, 1.593,
    1.600, 1.608])
sigmas_ins=np.array([0.00005, 0.00007, 0.00015, 0.00035, 0.00062, 0.00081, 0.00091,
    0.00113, 0.00131, 0.00159, 0.00234, 0.00315])
cs_ins=np.array([0.764, 0.950, 1.042, 1.097, 1.135, 1.165, 1.190, 1.210, 1.230, 1.244,
    1.258, 1.271])
rhos_ins=1.0e6*np.array([0.4400, 0.4392, 0.4382, 0.4371, 0.4361, 0.4350, 0.4340, 0.4329,
    0.4318, 0.4309, 0.4299, 0.4288])
```

```
ks_ins=100.0*np.array([0.000631, 0.000725, 0.00085, 0.000975, 0.0011, 0.001225, 0.00135,
    0.001475, 0.0016, 0.0018, 0.0020, 0.0022])
mus_ins=np.ones(shape(t_ins))

# Determine activation energy and sigmoid function rho = rho(theta(t,T))
msc_times = np.c_[times_2,times_5,times_8] # put data into matrices for feeding to find_Q
    function
msc_temps = np.c_[temps_2,temps_5,temps_8]
msc_rhos = np.c_[rhos_2,rhos_5,rhos_8]
msc_expnames = ["2 degC/min","5 degC/min","8 degC/min"]

printstring=("\nDetermining optimal activation energy and density function...\n\tUsing
    densification data from {Teng et al}...\n\tAttempting data fit to "+method+" sigmoid
    curve...\n")
logfile.write(printstring)
starttime=time.clock()

import msc
#Q,rhofun = msc.find_Q(msc_times,msc_temps,msc_rhos,msc_expnames,method,savestring,showinfo
    =False)
Q = 674214 # this is from result of previous optimization with {Teng} data and Fantozzi
    curve
rhofun=msc.find_sigmoid(msc_times,msc_temps,msc_rhos,msc_expnames,Q,method,savestring,
    showinfo=False)
#rhofun=msc.find_sigmoid(times_mat,t_mat,rhos_mat,['20 degC/min'],Q,method,savestring,
    showinfo=False)

printstring=("\tDone; took "+str(time.clock()-starttime)+" seconds to find optimal
    activation energy and MSC.\n\tOptimal activation energy is "+str(Q/1000)+" kJ/mol.\n\
    nInterpolating measured data to find dielectric and thermal properties as functions of
     temperature and relative density...")
logfile.write(printstring)
starttime=time.clock()

sampleplottemp = np.linspace(np.min(np.r_[t_mat,t_ins]),273.15+1200) # for plotting the
    material properties
tempmin = 24+273.15 # minimum temp we expect to encounter (tells spline interpolator that
    its values will eventually need to be extrapolated down to this value)
tempmax = 1400+273.15 # maximum temp we expect to encounter (tells spline interpolator that
     its values will eventually need to be extrapolated up to this value)
spldeg = 3 # degree of splines to use for interpolating (cubic recommended)
```

```

# Functions for load parameters
if theta_dep_params: # Method 1: theta-dependent parameters
       printstring = "\n\tAssuming parameters are functions of ln(theta)..."
       logfile.write(printstring)

       sampleplottimes=np.zeros(np.shape(sampleplottemp)) # assume a constant heating rate
           of 20 degC/min for sample data
       sampleplottimes[0]=(sampleplottemp[0]-273.15)*(60/20) # the time it took to get to
           the first temperature we have property measurements for
       for i in range(1,np.size(sampleplottemp)):
              sampleplottimes[i]=sampleplottimes[i-1]+(60/20)*(sampleplottemp[i]-
                  sampleplottemp[i-1]) # simulate constant heating rate of 20 degC/min, in
                   the absence of better information
       sampleplotlnthetas = msc.find_lnthetas(sampleplottimes,sampleplottemp,Q) # get the
           ln(theta) values for plotting functions
       lnthetas = msc.find_lnthetas(times_mat,t_mat,Q) # get the ln(theta) values for
           actually doing the interpolation

       lntmin = -400 # minimum lntheta we expect to encounter (tells spline interpolator
           that its values will eventually need to be extrapolated down to this value)
       lntmax = 30 # maximum lntheta we expect to encounter (tells spline interpolator that
            its values will eventually need to be extrapolated up to this value)

       # Functions for load parameters (these take lntheta as input)
       epstck = intp.splrep(lnthetas,epses_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline
           interpolation
       def epsfun_mat(lntheta):
              if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-
                  vector
                     n,m=np.shape(lntheta)
                     splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),epstck)
                     return np.reshape(splevals,(n,m))
              else: # lntheta was either a scalar or an array-that's-a-vector
                     return intp.splev(lntheta,epstck) # spline evaluation

       sigtck = intp.splrep(lnthetas,sigmas_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline
           interpolation
       def sigfun_mat(lntheta):
              if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-
                  vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),sigtck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,sigtck) # spline evaluation

    ctck = intp.splrep(lnthetas,cs_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline interpolation
    def cfun_mat(lntheta):
        if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),ctck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,ctck) # spline evaluation

    ktck = intp.splrep(lnthetas,ks_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline interpolation
    def kfun_mat(lntheta):
        if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-vector
            n,m=np.shape(lntheta)
            splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),ktck)
            return np.reshape(splevals,(n,m))
        else: # lntheta was either a scalar or an array-that's-a-vector
            return intp.splev(lntheta,ktck) # spline evaluation

    if magmat:
        mutck = intp.splrep(lnthetas,mus_mat[1:],xb=lntmin,xe=lntmax,k=spldeg) # spline interpolation
        def mufun_mat(lntheta):
            if np.size(np.shape(lntheta)) == 2: # lntheta is an array-that's-not-a-vector
                n,m=np.shape(lntheta)
                splevals=intp.splev(np.reshape(lntheta,np.size(lntheta)),mutck)
                return np.reshape(splevals,(n,m))
            else: # lntheta was either a scalar or an array-that's-a-vector
                return intp.splev(lntheta,mutck) # spline evaluation
    else:
        def mufun_mat(lntheta):
```

```
                    return 1.0*np.ones(np.shape(lntheta))

# Uncomment to use barycentric interpolation instead of b-splines; we don't like this, though, because values extrapolated beyond range of initial data may diverge quickly to +/- infty
# epsfun_mat = intp.BarycentricInterpolator(lnthetas,epses_mat[1:])
# sigfun_mat = intp.BarycentricInterpolator(lnthetas,sigmas_mat[1:])
# cfun_mat = intp.BarycentricInterpolator(lnthetas,cs_mat[1:])
# kfun_mat = intp.BarycentricInterpolator(lnthetas,ks_mat[1:])
# mufun_mat = intp.BarycentricInterpolator(lnthetas,mus_mat[1:])

        plt.figure(10) # Plot eps(temp) for material
        plt.clf()
        plt.plot(sampleplotlnthetas,epsfun_mat(sampleplotlnthetas),'r-',label='Function approximation')
        plt.plot(lnthetas,epses_mat[1:],'ro',label='Experimental measurements')
        plt.legend(loc='upper left')
        plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
        plt.ylabel(r'$\varepsilon_r$ [unitless]')
        plt.title('Relative electric permittivity for zirconia')
        plt.savefig(savestring+'mat_epsfun.png')
        tikz_save(savestring+'mat_epsfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
        plt.close(10)

        plt.figure(11) # Plot sigma(temp) for material
        plt.clf()
        plt.plot(sampleplotlnthetas,sigfun_mat(sampleplotlnthetas),'b-',label='Function approximation')
        plt.plot(lnthetas,sigmas_mat[1:],'ro',label='Experimental measurements')
        plt.legend(loc='upper left')
        plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
        plt.ylabel('$\sigma$ [S/m]')
        plt.title('Electrical conductivity for zirconia')
        plt.savefig(savestring+'mat_sigfun.png')
        tikz_save(savestring+'mat_sigfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
        plt.close(11)

        plt.figure(12) # Plot c_p(temp) for material
        plt.clf()
```

```
    plt.plot(sampleplotlnthetas,cfun_mat(sampleplotlnthetas),'g—',label='Function
        approximation')
    plt.plot(lnthetas,cs_mat[1:],'ro',label='Experimental measurements')
    plt.legend(loc='upper left')
    plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
    plt.ylabel('$c_p$ [J/(gK)]')
    plt.title('Thermal conductivity for zirconia')
    plt.savefig(savestring+'mat_cfun.png')
    tikz_save(savestring+'mat_cfun.tex', figureheight = '\\figureheight', figurewidth =
        '\\figurewidth',show_info = False )
    plt.close(12)

    plt.figure(13) # Plot k(temp) for material
    plt.clf()
    plt.plot(sampleplotlnthetas,kfun_mat(sampleplotlnthetas),'b—',label='Function
        approximation')
    plt.plot(lnthetas,ks_mat[1:],'ro',label='Experimental measurements')
    plt.legend(loc='upper left')
    plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
    plt.ylabel('k [W/(mK)]')
    plt.title('Specific heat capacity for zirconia')
    plt.savefig(savestring+'mat_kfun.png')
    tikz_save(savestring+'mat_kfun.tex', figureheight = '\\figureheight', figurewidth =
        '\\figurewidth',show_info = False )
    plt.close(13)

    plt.figure(14) # Plot mu(temp) for material
    plt.clf()
    plt.plot(sampleplotlnthetas,mufun_mat(sampleplotlnthetas),'g—',label='Function
        approximation')
    plt.plot(lnthetas,mus_mat[1:],'ro',label='Experimental measurements')
    plt.legend(loc='upper left')
    plt.xlabel('$\ln(\Theta(t,T(t)))$ log(sec/K)')
    plt.ylabel('$\mu_r$ [unitless]')
    plt.title('Relative magnetic permeability for zirconia')
    plt.savefig(savestring+'mat_mufun.png')
    tikz_save(savestring+'mat_mufun.tex', figureheight = '\\figureheight', figurewidth =
        '\\figurewidth',show_info = False )
    plt.close(14)

```

```
else: # Method 2: mixture formula-based load parameter functions (these take temp, rho as inputs)

        printstring = "\n\tUsing inversions of mixture formulas plus interpolation of parameter along rho-axis to determine functions for dielectric and thermal properties of load and insulation..."
        logfile.write(printstring)

        # Estimate values of bulk parameters for interpolating
        if mat_params == "Licht":
                e2fn = epses_mat**(1/rhos_mat)
                s2fn = sigmas_mat**(1/rhos_mat)
                m2fn = mus_mat**(1/rhos_mat)
        elif mat_params == "Rayleigh":
                e2fn = (1+(2/rhos_mat)*((epses_mat-1)/(epses_mat+1)))/(1-(1/rhos_mat)*((epses_mat-1)/(epses_mat+1)))
                s2fn = (1+(2/rhos_mat)*((sigmas_mat-1)/(sigmas_mat+1)))/(1-(1/rhos_mat)*((sigmas_mat-1)/(sigmas_mat+1)))
                m2fn = (1+(2/rhos_mat)*((mus_mat-1)/(mus_mat+1)))/(1-(1/rhos_mat)*((mus_mat-1)/(mus_mat+1)))
        elif mat_params == "MG":
                e2fn = (eps_air*(1+rhos_mat)*(epses_mat-eps_air))/(2*rhos_mat*eps_air-(1-rhos_mat)*(epses_mat-eps_air))
                s2fn = (sig_air*(1+rhos_mat)*(sigmas_mat-sig_air))/(2*rhos_mat*sig_air-(1-rhos_mat)*(sigmas_mat-sig_air))
                m2fn = (mu_air*(1+rhos_mat)*(mus_mat-mu_air))/(2*rhos_mat*mu_air-(1-rhos_mat)*(mus_mat-mu_air))
        elif mat_params == "Bruggeman":
                e2fn = (epses_mat*(1-3*rhos_mat)+2*epses_mat*epses_mat)/(1+epses_mat*(2-3*rhos_mat))
                s2fn = (sigmas_mat*(1-3*rhos_mat)+2*sigmas_mat*sigmas_mat)/(1+sigmas_mat*(2-3*rhos_mat))
                m2fn = (mus_mat*(1-3*rhos_mat)+2*mus_mat*mus_mat)/(1+mus_mat*(2-3*rhos_mat))

        # Interpolate bulk parameters with temperature
        epstck = intp.splrep(t_mat,e2fn,xb=tempmin,xe=tempmax,k=spldeg)
        sigtck = intp.splrep(t_mat,s2fn,xb=tempmin,xe=tempmax,k=spldeg)
        mutck = intp.splrep(t_mat,m2fn,xb=tempmin,xe=tempmax,k=spldeg)

        # Construct functions
        if mat_params == "Licht":
            def epsfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—
                    vector
                    n,m=np.shape(temp)
                    eps2=intp.splev(np.reshape(temp,np.size(temp)),epstck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = eps2**rho
                    return np.reshape(outvals,(n,m))
                else: # temp was either a scalar or an array—that's—a—vector
                    eps2=intp.splev(temp,epstck)
                    return eps2**rho
            def sigfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—
                    vector
                    n,m=np.shape(temp)
                    sig2=intp.splev(np.reshape(temp,np.size(temp)),sigtck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = sig2**rho
                    return np.reshape(outvals,(n,m))
                else:
                    sig2=intp.splev(temp,sigtck)
                    return sig2**rho
            def mufun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—
                    vector
                    n,m=np.shape(temp)
                    mu2=intp.splev(np.reshape(temp,np.size(temp)),mutck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = mu2**rho
                    return np.reshape(outvals,(n,m))
                else:
                    mu2=intp.splev(temp,mutck)
                    return mu2**rho
        elif mat_params == "Rayleigh":
            def epsfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—
                    vector
                    n,m=np.shape(temp)
                    eps2=intp.splev(np.reshape(temp,np.size(temp)),epstck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = (eps2*(2*rho+1)—(2*rho—2))/(eps2*(1—rho)+(rho—2))
                    return np.reshape(outvals,(n,m))
                else: # temp was either a scalar or an array-that's-a-vector
                    eps2=intp.splev(temp,epstck)
                    return (eps2*(2*rho+1)-(2*rho-2))/(eps2*(1-rho)+(rho-2))
            def sigfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    sig2=intp.splev(np.reshape(temp,np.size(temp)),sigtck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = (sig2*(2*rho+1)-(2*rho-2))/(sig2*(1-rho)+(rho-2))
                    return np.reshape(outvals,(n,m))
                else:
                    sig2=intp.splev(temp,sigtck)
                    return (sig2*(2*rho+1)-(2*rho-2))/(sig2*(1-rho)+(rho-2))
            def mufun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    mu2=intp.splev(np.reshape(temp,np.size(temp)),mutck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = (mu2*(2*rho+1)-(2*rho-2))/(mu2*(1-rho)+(rho-2))
                    return np.reshape(outvals,(n,m))
                else:
                    mu2=intp.splev(temp,mutck)
                    return (mu2*(2*rho+1)-(2*rho-2))/(mu2*(1-rho)+(rho-2))
        elif mat_params == "MG":
            def epsfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    eps2=intp.splev(np.reshape(temp,np.size(temp)),epstck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = eps_air+2*rho*eps_air*(eps2-eps_air)/(eps2+eps_air-rho*(eps2-eps_air))
                    return np.reshape(outvals,(n,m))
                else: # temp was either a scalar or an array-that's-a-vector
                    eps2=intp.splev(temp,epstck)
                    return eps_air+2*rho*eps_air*(eps2-eps_air)/(eps2+eps_air-rho*(eps2-eps_air))
            def sigfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    sig2=intp.splev(np.reshape(temp,np.size(temp)),sigtck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = sig_air+2*rho*sig_air*(sig2-sig_air)/(sig2+sig_air-rho*(sig2-sig_air))
                    return np.reshape(outvals,(n,m))
                else:
                    sig2=intp.splev(temp,sigtck)
                    return sig_air+2*rho*sig_air*(sig2-sig_air)/(sig2+sig_air-rho*(sig2-sig_air))
            def mufun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    mu2=intp.splev(np.reshape(temp,np.size(temp)),mutck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = mu_air+2*rho*mu_air*(mu2-mu_air)/(mu2+mu_air-rho*(mu2-mu_air))
                    return np.reshape(outvals,(n,m))
                else:
                    mu2=intp.splev(temp,mutck)
                    return mu_air+2*rho*mu_air*(mu2-mu_air)/(mu2+mu_air-rho*(mu2-mu_air))
        elif mat_params == "Bruggeman":
            def epsfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    eps2=intp.splev(np.reshape(temp,np.size(temp)),epstck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = 0.5*(1+3*rho*(1-eps2))+0.5*sqrt((1+3*rho*(1-eps2))**(2)+4*eps2)
                    return np.reshape(outvals,(n,m))
                else: # temp was either a scalar or an array-that's-a-vector
                    eps2=intp.splev(temp,epstck)
                    return 0.5*(1+3*rho*(1-eps2))+0.5*sqrt((1+3*rho*(1-eps2))**(2)+4*eps2)
            def sigfun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    sig2=intp.splev(np.reshape(temp,np.size(temp)),sigtck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = 0.5*(1+3*rho*(1-sig2))+0.5*sqrt((1+3*rho*(1-sig2))**(2)+4*sig2)
                    return np.reshape(outvals,(n,m))
                else:
                    sig2=intp.splev(temp,sigtck)
                    return 0.5*(1+3*rho*(1-sig2))+0.5*sqrt((1+3*rho*(1-sig2))**(2)+4*sig2)
            def mufun_mat(temp,rho):
                if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                    n,m=np.shape(temp)
                    mu2=intp.splev(np.reshape(temp,np.size(temp)),mutck)
                    rho = np.reshape(rho,np.size(rho))
                    outvals = 0.5*(1+3*rho*(1-mu2))+0.5*sqrt((1+3*rho*(1-mu2))**(2)+4*mu2)
                    return np.reshape(outvals,(n,m))
                else:
                    mu2=intp.splev(temp,mutck)
                    return 0.5*(1+3*rho*(1-mu2))+0.5*sqrt((1+3*rho*(1-mu2))**(2)+4*mu2)

        if not magmat:
            def mufun_mat(temp,rho):
                return 1.0*np.ones(np.shape(temp))

        # Specfic heat capacity
        ctck = intp.splrep(t_mat,cs_mat/rhos_mat,xb=tempmin,xe=tempmax,k=spldeg)
        def cfun_mat(temp,rho):
            if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                n,m=np.shape(temp)
                splevals=intp.splev(np.reshape(temp,np.size(temp)),ctck)
                rho = np.reshape(rho,np.size(rho))
                outvals=splevals*rho
                return np.reshape(outvals,(n,m))
            else: # temp was either a scalar or an array-that's-a-vector
                return (intp.splev(temp,ctck))*rho

        # Thermal conductivity
        ktck = intp.splrep(t_mat,ks_mat/(1.5*rhos_mat-0.5),xb=tempmin,xe=tempmax,k=spldeg)
        def kfun_mat(temp,rho):
            if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                n,m=np.shape(temp)
                splevals=intp.splev(np.reshape(temp,np.size(temp)),ktck)
                rho = np.reshape(rho,np.size(rho))
                outvals=(splevals)*(1.5*rho-0.5)
                return np.reshape(outvals,(n,m))
            else: # temp was either a scalar or an array-that's-a-vector
                return (intp.splev(temp,ktck))*(1.5*rho-0.5)

        # For plotting
        sampleplotrhovals = np.linspace(rhos_mat[0],rhos_mat[-1])

        sampleploteps=np.zeros(np.shape(sampleplottemp))
        sampleplotsig=np.zeros(np.shape(sampleplottemp))
        sampleplotmu=np.zeros(np.shape(sampleplottemp))
        sampleplotc=np.zeros(np.shape(sampleplottemp))
        sampleplotk=np.zeros(np.shape(sampleplottemp))

        for ind in range(0,np.size(sampleplottemp)):
            sampleploteps[ind]=epsfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
            sampleplotsig[ind]=sigfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
            sampleplotmu[ind]=mufun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
            sampleplotc[ind]=cfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])
            sampleplotk[ind]=kfun_mat(sampleplottemp[ind],sampleplotrhovals[ind])

        plt.figure(10) # Plot eps(temp) for material
        plt.clf()
    # plt.plot(sampleplottemp-273.15,epsfun_mat(sampleplottemp,rhoval),'r-',label='Function
        approximation')
        plt.plot(sampleplottemp-273.15,sampleploteps,'r-',label='Function approximation')
        plt.plot(t_mat-273.15,epses_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel(r'$\varepsilon_r$ [unitless]')
        plt.title(r'Relative electric permittivity for zirconia')
        plt.savefig(savestring+'mat_epsfun.png')
```

```
        tikz_save(savestring+'mat_epsfun.tex', figureheight = '\\figureheight', figurewidth
            = '\\figurewidth',show_info = False )
        plt.close(10)

        plt.figure(11) # Plot sigma(temp) for material
        plt.clf()
# plt.plot(sampleplottemp-273.15,sigfun_mat(sampleplottemp,rhoval),'b-',label='Function
    approximation')
        plt.plot(sampleplottemp-273.15,sampleplotsig,'b-',label='Function approximation')
        plt.plot(t_mat-273.15,sigmas_mat,'ro',label='Experimental measurements (temp only)'
            )
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$\sigma$ [S/m]')
        plt.title(r'Electrical conductivity for zirconia')
        plt.savefig(savestring+'mat_sigfun.png')
        tikz_save(savestring+'mat_sigfun.tex', figureheight = '\\figureheight', figurewidth
            = '\\figurewidth',show_info = False )
        plt.close(11)

        plt.figure(12) # Plot c_p(temp) for material
        plt.clf()
        #plt.plot(sampleplottemp-273.15,cfun_mat(sampleplottemp,rhoval),'g-',label='
            Function approximation')
        plt.plot(sampleplottemp-273.15,sampleplotc,'g-',label='Function approximation')
        plt.plot(t_mat-273.15,cs_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$c_p$ [J/(gK)]')
        plt.title(r'Specific heat capacity for zirconia')
        plt.savefig(savestring+'mat_cfun.png')
        tikz_save(savestring+'mat_cfun.tex', figureheight = '\\figureheight', figurewidth =
            '\\figurewidth',show_info = False )
        plt.close(12)

        plt.figure(13) # Plot k(temp) for material
        plt.clf()
        #plt.plot(sampleplottemp-273.15,kfun_mat(sampleplottemp,rhoval),'b-',label='
            Function approximation')
        plt.plot(sampleplottemp-273.15,sampleplotk,'b-',label='Function approximation')
        plt.plot(t_mat-273.15,ks_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$k$ [W/(mK)]')
        plt.title(r'Thermal conductivity for zirconia')
        plt.savefig(savestring+'mat_kfun.png')
        tikz_save(savestring+'mat_kfun.tex', figureheight = '\\figureheight', figurewidth =
            '\\figurewidth',show_info = False )
        plt.close(13)

        plt.figure(14) # Plot mu(temp) for material
        plt.clf()
        #plt.plot(sampleplottemp-273.15,mufun_mat(sampleplottemp,rhoval),'g-',label='
            Function approximation')
        plt.plot(sampleplottemp-273.15,sampleplotmu,'g-',label='Function approximation')
        plt.plot(t_mat-273.15,mus_mat,'ro',label='Experimental measurements (temp only)')
        plt.legend(loc='upper left')
        plt.xlabel('Temperature (degC)')
        plt.ylabel('$\mu_r$ [unitless]')
        plt.title(r'Relative magnetic permeability for zirconia')
        plt.savefig(savestring+'mat_mufun.png')
        tikz_save(savestring+'mat_mufun.tex', figureheight = '\\figureheight', figurewidth =
             '\\figurewidth',show_info = False )
        plt.close(14)

# Functions for insulation parameters (these take temps as inputs)
epsinstck = intp.splrep(t_ins,epses_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline
    interpolation
def epsfun_ins(temp):
        if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                #temp[temp>1200+273.15]=1200+273.15
                n,m=np.shape(temp)
                splevals=intp.splev(np.reshape(temp,np.size(temp)),epsinstck)
                return np.reshape(splevals,(n,m))
        else: # temp was either a scalar or an array-that's-a-vector
                return intp.splev(temp,epsinstck) # spline evaluation

siginstck = intp.splrep(t_ins,sigmas_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline
    interpolation
def sigfun_ins(temp):
        if np.size(np.shape(temp)) == 2: # temp is an array-that's-not-a-vector
                #temp[temp>1200+273.15]=1200+273.15
```

```
              n,m=np.shape(temp)
              splevals=intp.splev(np.reshape(temp,np.size(temp)),siginstck)
              return np.reshape(splevals,(n,m))
       else: # temp was either a scalar or an array—that's—a—vector
              return intp.splev(temp,siginstck) # spline evaluation

cinstck = intp.splrep(t_ins,cs_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline interpolation
def cfun_ins(temp):
       if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—vector
              #temp[temp>1200+273.15]=1200+273.15
              n,m=np.shape(temp)
              splevals=intp.splev(np.reshape(temp,np.size(temp)),cinstck)
              return np.reshape(splevals,(n,m))
       else: # temp was either a scalar or an array—that's—a—vector
              return intp.splev(temp,cinstck) # spline evaluation

rhoinstck = intp.splrep(t_ins,rhos_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline
    interpolation
def rhofun_ins(temp):
       if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—vector
              #temp[temp>1200+273.15]=1200+273.15
              n,m=np.shape(temp)
              splevals=intp.splev(np.reshape(temp,np.size(temp)),rhoinstck)
              return np.reshape(splevals,(n,m))
       else: # temp was either a scalar or an array—that's—a—vector
              return intp.splev(temp,rhoinstck) # spline evaluation

kinstck = intp.splrep(t_ins,ks_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline interpolation
def kfun_ins(temp):
       if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—vector
              #temp[temp>1200+273.15]=1200+273.15
              n,m=np.shape(temp)
              splevals=intp.splev(np.reshape(temp,np.size(temp)),kinstck)
              return np.reshape(splevals,(n,m))
       else: # temp was either a scalar or an array—that's—a—vector
              return intp.splev(temp,kinstck) # spline evaluation

muinstck = intp.splrep(t_ins,mus_ins,xb=tempmin,xe=tempmax,k=spldeg) # spline interpolation
def mufun_ins(temp):
       if np.size(np.shape(temp)) == 2: # temp is an array—that's—not—a—vector
              #temp[temp>1200+273.15]=1200+273.15
              n,m=np.shape(temp)
              splevals=intp.splev(np.reshape(temp,np.size(temp)),muinstck)
              return np.reshape(splevals,(n,m))
       else: # temp was either a scalar or an array-that's-a-vector
              return intp.splev(temp,muinstck) # spline evaluation

# Uncomment to use barycentric interpolation instead of b-splines
#epsfun_ins = intp.BarycentricInterpolator(t_ins,epses_ins)
#sigfun_ins = intp.BarycentricInterpolator(t_ins,sigmas_ins)
#cfun_ins = intp.BarycentricInterpolator(t_ins,cs_ins)
#rhofun_ins = intp.BarycentricInterpolator(t_ins,rhos_ins)
#kfun_ins = intp.BarycentricInterpolator(t_ins,ks_ins)
#mufun_ins = intp.BarycentricInterpolator(t_ins,mus_ins)

plt.figure(20) # Plot eps(temp) for insulation
plt.plot(sampleplottemp-273.15,epsfun_ins(sampleplottemp),'r-',label='Function approximation')
plt.plot(t_ins-273.15,epses_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature (degC)')
plt.ylabel(r'$\varepsilon_r$ [unitless]')
plt.title('Relative electric permittivity for alumina insulation')
plt.savefig(savestring+'ins_epsfun.png')
tikz_save(savestring+'ins_epsfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(20)

plt.figure(21) # Plot sigma(temp) for insulation
plt.plot(sampleplottemp-273.15,sigfun_ins(sampleplottemp),'b-',label='Function approximation')
plt.plot(t_ins-273.15,sigmas_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature (degC)')
plt.ylabel('$\sigma$ [S/m]')
plt.title('Electrical conductivity for alumina insulation')
plt.savefig(savestring+'ins_sigfun.png')
tikz_save(savestring+'ins_sigfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(21)

plt.figure(22) # Plot c_p(temp) for insulation
plt.plot(sampleplottemp-273.15,cfun_ins(sampleplottemp),'g-',label='Function approximation')
plt.plot(t_ins-273.15,cs_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature (degC)')
plt.ylabel('$c_p$ [J/(gK)]')
plt.title('Thermal conductivity for alumina insulation')
plt.savefig(savestring+'ins_cfun.png')
tikz_save(savestring+'ins_cfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(22)

plt.figure(23) # Plot rho(temp) for insulation
plt.plot(sampleplottemp-273.15,rhofun_ins(sampleplottemp),'y-',label='Function approximation')
plt.plot(t_ins-273.15,rhos_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature (degC)')
plt.ylabel(r'$\rho$ [g/(cm^3)]')
plt.title('Density for alumina insulation')
plt.savefig(savestring+'ins_rhofun.png')
tikz_save(savestring+'ins_rhofun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(23)

plt.figure(24) # Plot k(temp) for insulation
plt.plot(sampleplottemp-273.15,kfun_ins(sampleplottemp),'b-',label='Function approximation')
plt.plot(t_ins-273.15,ks_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature (degC)')
plt.ylabel('$k$ [W/(mK)]')
plt.title('Specific heat capacity for alumina insulation')
plt.savefig(savestring+'ins_kfun.png')
tikz_save(savestring+'ins_kfun.tex', figureheight = '\\figureheight', figurewidth = '\\figurewidth',show_info = False )
plt.close(24)

plt.figure(25) # Plot mu(temp) for insulation
plt.plot(sampleplottemp-273.15,mufun_ins(sampleplottemp),'g-',label='Function approximation')
plt.plot(t_ins−273.15,mus_ins,'ro',label='Experimental measurements')
plt.legend(loc='upper left')
plt.xlabel('Temperature (degC)')
plt.ylabel('$\mu_r$ [unitless]')
plt.title('Relative magnetic permeability for alumina insulation')
plt.savefig(savestring+'ins_mufun.png')
tikz_save(savestring+'ins_mufun.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )
plt.close(25)

printstring=("\n\tDone; took "+str(time.clock()−starttime)+" seconds to find functions for
     all dielectric and thermal material and insulation properties.\n\nSetting up
    simulation...\n")
logfile.write(printstring)

# Initialize elemental values of load properties (these are updated in the course of
    mechanical solution)
# Since we don't know heating rate, must start with only temp−dependent interpolated
    parameters
eps_mat=intp.BarycentricInterpolator(t_mat,epses_mat).__call__(temp_init) #[unitless]
    relative permittivity
sig_mat=intp.BarycentricInterpolator(t_mat,sigmas_mat).__call__(temp_init) # [S/m]
    electrical conductivity
c_mat=intp.BarycentricInterpolator(t_mat,cs_mat).__call__(temp_init) # [J/g*C] specific
    heat capacity
rho_mat=(intp.BarycentricInterpolator(t_mat,rhos_mat).__call__(temp_init)+0.1)*bulkdens_mat
     # [g/m^3] density
k_mat=intp.BarycentricInterpolator(t_mat,ks_mat).__call__(temp_init) # [W/g*C] thermal
    conductivity
mu_mat=intp.BarycentricInterpolator(t_mat,mus_mat).__call__(temp_init) # [unitless]
    relative permeability

# Initialize elemental values of insulation properties
eps_ins=epsfun_ins(temp_init) #[unitless] relative permittivity of insulation at initial
    temperature
sig_ins=sigfun_ins(temp_init) # [S/m] electrical conductivity of insulation at initial
    temperature
c_ins=cfun_ins(temp_init) # [J/g*C] specific heat capacity of insulation at initial
    temperature
rho_ins=rhofun_ins(temp_init) # [g/m^3] density of insulation at initial temperature
```

```
k_ins=kfun_ins(temp_init) # [W/g*C] thermal conductivity of insulation at initial temperature
mu_ins=mufun_ins(temp_init) # [unitless] relative permeability of insulation at initial temperature

#################################################################

# Nodes and spacing
delta_x_air = delta_z_air = 0.05*c/(f_g*sqrt(eps_air)) # length of spatial step in air [m]
delta_x_mat = delta_z_mat = 0.05*c/(f_g*sqrt(eps_mat)) # length of spatial step in material [m]
delta_x_ins = delta_z_ins = 0.05*c/(f_g*sqrt(eps_ins)) # length of spatial step in insulation [m]

# Create physical domain
left_air_vec = r_[0:ell_1:delta_z_air]
ell_1 = max(ell_1,left_air_vec[-1]+delta_z_ins) # makes sure step at interface is not too small
left_ins_vec = r_[ell_1:ell_2:delta_z_ins]
ell_2 = max(ell_2,left_ins_vec[-1]+delta_z_mat) # makes sure step at interface is not too small
mat_vec = r_[ell_2:ell_3:delta_z_mat]
ell_3 = max(ell_3,mat_vec[-1]+delta_z_mat) # makes sure step at interface is not too small
right_ins_vec = r_[ell_3:ell_4:delta_z_ins]
ell_4 = max(ell_4,right_ins_vec[-1]+delta_z_ins) # makes sure step at interface is not too small
right_air_vec = r_[ell_4:L:delta_z_air]

lower_air_vec = r_[0:h_1:delta_x_air]
h_1 = max(h_1,lower_air_vec[-1]+delta_x_ins) # makes sure step at interface is not too small
lower_ins_vec = r_[h_1:h_2:delta_x_ins]
h_2 = max(h_2,lower_ins_vec[-1]+delta_x_mat) # makes sure step at interface is not too small
x_mat_vec = r_[h_2:h_3:delta_x_mat]
h_3 = max(h_3,x_mat_vec[-1]+delta_x_mat) # makes sure step at interface is not too small
upper_ins_vec = r_[h_3:h_4:delta_x_ins]
h_4 = max(h_4,upper_ins_vec[-1]+delta_x_ins) # makes sure step at interface is not too small
upper_air_vec = r_[h_4:H:delta_x_air]

```

```
# Finally, create the vectors of z−values and x−values
z = r_[ left_air_vec , left_ins_vec , mat_vec , right_ins_vec , right_air_vec ]
x = r_[ lower_air_vec, lower_ins_vec, x_mat_vec, upper_ins_vec, upper_air_vec ]

Z,X = meshgrid(z,x) # for plotting

ins_start_z = np.size(left_air_vec) # index of first node within left−hand insulation
mat_start_z = ins_start_z+np.size(left_ins_vec) # index of first node within material
mat_end_z = mat_start_z+np.size(mat_vec) # index of first node within right−hand
    insulation
ins_end_z = mat_end_z+np.size(right_ins_vec) # index of first node within right−hand air

ins_start_x = np.size(upper_air_vec) # index of first node within lower insulation
mat_start_x = ins_start_x+np.size(upper_ins_vec) # index of first node within material
mat_end_x = mat_start_x+np.size(x_mat_vec) # index of first node within upper insulation
ins_end_x = mat_end_x+np.size(lower_ins_vec) # index of first node within upper air

ins_length = ell_4−ell_1 # length of insulation&material&insulation [m]
ins_height = h_4−h_1 # height of insulation&material&insulation [m]

mat_length = ell_3−ell_2 # length of material [m]
mat_height = h_3−h_2 # height of material [m]

n_mat_z = np.size(mat_vec)+1 # number of z−nodes in material
n_mat_x = np.size(x_mat_vec)+1 # number of x−nodes in material

# Initialize indices for material relative to insulation boundary
z_start = np.size(left_ins_vec)
x_start = np.size(upper_ins_vec)
z_end = −(np.size(right_ins_vec))
x_end = −(np.size(lower_ins_vec))

# Localized deformation model
#for i in range (mat_start_z,mat_end_z+1): # number of x−nodes in each "column" of
    material
# eval('n_mat_x_'+str(i)) = n_mat_x # is initially constant

n_ins_z = ins_end_z−ins_start_z+1 # number of nodes in insulation&material&insulation
n_ins_x = ins_end_x−ins_start_x+1 # number of nodes in insulation&material&insulation

nz=np.size(z) # number of z−gridpoints
```

```
N = nz−1 # highest index among z−gridpoints
nx=np.size(x) # number of x−gridpoints
M = nx−1 # highest index among x−gridpoints
hz=z[1:]−z[:−1] # delta−z−values
hx=x[1:]−x[:−1] # delta−x−values

##################################################################

eps = eps_air * np.ones((nx,nz))
eps[ins_start_x:ins_end_x,ins_start_z:ins_end_z] = eps_ins
eps[mat_start_x:mat_end_x,mat_start_z:mat_end_z] = eps_mat

sig = sig_air * np.ones((nx,nz))
sig[ins_start_x:ins_end_x,ins_start_z:ins_end_z] = sig_ins
sig[mat_start_x:mat_end_x,mat_start_z:mat_end_z] = sig_mat

cp = eps_air * np.ones((nx,nz))
cp[ins_start_x:ins_end_x,ins_start_z:ins_end_z] = c_ins
cp[mat_start_x:mat_end_x,mat_start_z:mat_end_z] = c_mat

rho = rho_air * np.ones((nx,nz))
rho[ins_start_x:ins_end_x,ins_start_z:ins_end_z] = rho_ins
rho[mat_start_x:mat_end_x,mat_start_z:mat_end_z] = rho_mat

k = k_air * np.ones((nx,nz))
k[ins_start_x:ins_end_x,ins_start_z:ins_end_z] = k_ins
k[mat_start_x:mat_end_x,mat_start_z:mat_end_z] = k_mat

mu = mu_air * np.ones((nx,nz))
mu[ins_start_x:ins_end_x,ins_start_z:ins_end_z] = mu_ins
mu[mat_start_x:mat_end_x,mat_start_z:mat_end_z] = mu_mat

#########################################################################

# Time scenario
#em_dt = min(delta_x_air,delta_x_mat,delta_x_ins)/c # length of time step of em solve [sec]
em_dt = 1.0e−2 # use when th,phi>=0.5
h_dt = 1.0e−1 # length of time step of heat solve (i.e., how long to nuke before solving
    heat transfer) [sec]

# Initialize electric field
```

```
E_old = np.zeros(np.shape(X)) # initialize electric field as matrix
beta = pi/L # propagation constant [1/m]
E_inc = (2/L)*sqrt(2*P*omega_fs*mu0/beta) # initialize power at magnetron (left-hand
    boundary)
E_old[:,0] = E_inc # replace E-field values on the left-hand boundary with value of
    incident field at magnetron (this is for nondimensional problem, so put ones there)
E_old = mat2vec(np.transpose(E_old)) # convert E_old to vector
E_older = E_old
E_old_init = E_old
E_older_init = E_older # initialize second oldest electric field as vector
eavg = np.zeros(np.shape(E_old))

# Initialize temperature field as matrix
temp = temp_init*np.ones((n_ins_x,n_ins_z))

# Initialize theta
theta_integrand_old = (np.exp(-Q/(R*temp_init))/temp_init)*np.ones((n_mat_x,n_mat_z)) #
    theta is a cumulative integral; this is the initial value of the integrand
theta = np.zeros(np.shape(theta_integrand_old)) # inital value of cumulative integral is
    zero

# Initialize material height
height_old = mat_height

# Initialize average density in material
rho_avg_old = rho_mat/bulkdens_mat

# Initialize time and iterations
loopstarttime = time.clock()
elapsed_time = 0.0 # initialize elapsed time
loopits = 0 # initialize number of iterations of coupled solver per 'loop' (one loop is how
     often we plot results, no matter how many timesteps we've taken between plots)
itno = 0
delfiles = [] # for storing names of files containing frames for movies--want to delete
    these files at the end of simulation

# Initialize phase velocity and dimensional factor for EM solver
#p = mu0*sig*em_dt/(2*eps*eps0) # factor for em solver
#vp = c*sqrt(1/(eps*mu)) # phase velocity in media

if embc=='abs':
```

```python
       embcprintstring = "absorbing"
elif embc=='pec':
       embcprintstring = "perfect electric conductor (zero Dirichlet)"

if tempbc=='rad':
       tempbcprintstring="radiative"
elif tempbc=='ins':
       tempbcprintstring="insulating (zero Neumann)"
elif tempbc=="fix":
       tempbcprintstring="fixed (Dirichlet)"

printstring = ("\tSpatial cell size in air (same in x-dir as in z-dir) is "+str(delta_x_air*1e2)+" cm\n\tSpatial cell size in insulation (same in x-dir as in z-dir) "+str(delta_x_ins*1e2)+" cm\n\tSpatial cell size in material (same in x-dir as in z-dir) is "+str(delta_x_mat*1e2)+" cm\n\tTotal number of nodes in entire domain is "+str(nx*nz)+"\n\tTotal number of nodes in insulation + material is "+str(n_ins_x*n_ins_z)+"\n\tTotal number of nodes in material is "+str(n_mat_x*n_mat_z)+"\n\tTime step for electromagnetic solve is "+str(em_dt)+" sec\n\tTime step for thermal solve is "+str(h_dt)+" sec\n\tTotal simulated processing time will be "+str(total_time)+" sec\n")
logfile.write(printstring)
printstring = ("\nStarting simulation loop...\n\tUsing "+embcprintstring+" boundary condition for electromagnetic solver\n\tUsing "+tempbcprintstring+" boundary condition for thermal solver\n")
logfile.write(printstring)

instemps = np.r_[mat2vec(temp[:x_start,:]),mat2vec(temp[x_end:,:]),mat2vec(temp[x_start:x_end,:z_start]),mat2vec(temp[x_start:x_end,z_end:])]-273.15
loadtemps = temp[x_start:x_end,z_start:z_end]-273.15
max_ins = np.max(instemps)
min_ins = np.min(instemps)
mean_ins = np.mean(instemps)
max_load = np.max(loadtemps)
min_load = np.min(loadtemps)
mean_load = np.mean(loadtemps)

printstring = ('\nAt start of simulation...\n\tMax value of electric field is ' + str(np.max(eavg)) + ' V/m\n\tMin value of electric field is ' + str(np.min(eavg)) + ' V/m\n\tMean value of electric field is ' + str(np.mean(eavg))+' V/m\n\tMax temp in insulation is ' + str(max_ins) + ' degC\n\tMin temp in insulation is ' + str(min_ins) + ' degC\n\tMean temp in insulation is ' + str(mean_ins) + ' degC\n\tMax temp in load is ' + str(max_load) + ' degC\n\tMin temp in load is ' + str(min_load) + ' degC\n\
    tMean temp in load is ' + str(mean_load) + ' degC\n\tMean density in material is '+str
    (100*rho_mat/bulkdens_mat)+" percent of bulk density\n")
logfile.write(printstring)

T_maxes = np.array([max_load])
T_means = np.array([mean_load])
load_rhos = np.array([100*rho_mat/bulkdens_mat])
plottingtimes = lnt_avgs = np.array([0])

# Simulation loop
while elapsed_time<total_time:

       T = (temp_init-273.15)*np.ones(np.shape(X)) # dimensional temperature in entire
           cavity (air is assumed constant temp)
       T[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = temp-273.15 # update plotting
           matrix for temperature

       # Plot electric field
       plt.figure(30)
       plt.clf()
       plt.contourf(100*Z,100*X,np.fliplr(np.transpose(vec2mat(eavg,nz,nx))),100)
       plt.colorbar()
       plt.xlabel('Position along domain [cm]')
       plt.ylabel('Position along domain [cm]')
       plt.title('Envelope of electric field [V^2/m^2] at t=' + str(elapsed_time) + ' sec')
       fname = hiddensavestring+'efield'+str(itno)+'.png'
       plt.savefig(fname)
       delfiles.append(fname)

       # Plot temperature field in only ins + mat
       plt.figure(31)
       plt.clf()
       plt.contourf(100*Z[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1],100*X[
           ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1],temp-273.15,100)
       plt.colorbar()
       plt.xlabel('Position along domain [cm]')
       plt.ylabel('Position along domain [cm]')
       plt.title('Temperature [C] at t=' + str(elapsed_time) + ' seconds')
       fname = hiddensavestring+'temp_insmat'+str(itno)+'.png'
       plt.savefig(fname)
       delfiles.append(fname)
```

```

        # Plot temperature field in whole cavity
        plt.figure(32)
        plt.clf()
        plt.contourf(100*Z,100*X,np.fliplr(T),100)
        plt.colorbar()
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title('Temperature [C] at t=' + str(elapsed_time) + ' seconds')
        fname = hiddensavestring+'temp_wholecav'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        # Plot material properties

        plt.figure(40) # relative permittivity
        plt.clf()
        plt.contourf(100*Z,100*X,eps,100)
        plt.colorbar()
        plt.plot(100*x,eps,'ro-')
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title(r'$\varepsilon_r$ [unitless] at time t=' + str(elapsed_time) + ' sec')
        fname = hiddensavestring+'eps_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(41) # electrical conductivity
        plt.clf()
        plt.contourf(100*Z,100*X,sig,100)
        plt.colorbar()
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title(r'$\sigma$ [S/m] at time t=' + str(elapsed_time) + ' sec')
        fname = hiddensavestring+'sig_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(42) # thermal conductivity
        plt.clf()
        plt.contourf(100*Z,100*X,cp,100)
        plt.colorbar()
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title('$c_p$ [J/(gK)] at time t=' + str(elapsed_time) + ' sec')
        fname = hiddensavestring+'c_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(43) # density
        plt.clf()
        plt.contourf(100*Z,100*X,rho,100)
        plt.colorbar()
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title(r'$\rho$ [g/m^3] at time t=' + str(elapsed_time) + ' sec')
        fname = hiddensavestring+'rho_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(44) # specific heat capacity
        plt.clf()
        plt.contourf(100*Z,100*X,k,100)
        plt.colorbar()
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title('$k$ [W/(mK)] at time t=' + str(elapsed_time) + ' sec')
        fname = hiddensavestring+'k_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

        plt.figure(45) # magnetic permeability
        plt.clf()
        plt.contourf(100*Z,100*X,mu,100)
        plt.colorbar()
        plt.xlabel('Position along domain [cm]')
        plt.ylabel('Position along domain [cm]')
        plt.title(r'$\mu$ [unitless] at time t='+str(elapsed_time) + ' seconds')
        fname = hiddensavestring+'mu_evol'+str(itno)+'.png'
        plt.savefig(fname)
        delfiles.append(fname)

```

```
    loopits = 0
    staycount = 0
    leftcount = 0
    rightcount = 0

    # Run coupled solver
    while loopits<floor(print_every/h_dt): # print shrinkage every (n many) timesteps instead of each timestep (avoid creating huuuuuge logfiles)

        # Special parameters for EM and T solvers
        cp_insmat = cp[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1]
        k_insmat = k[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1]
        rho_insmat = rho[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1]

        # Convert matrix-shaped field and property values to vector shapes
        temp = mat2vec(temp)
        cp_insmat = mat2vec(cp_insmat)
        k_insmat = mat2vec(k_insmat)
        rho_insmat = mat2vec(rho_insmat)

        k = mat2vec(k)
        cp = mat2vec(cp)

        # Run electric field solver

        # For wave equation, run as many times as it takes to achieve one timestep of heat equation (depending on em timestep, this could be many)

#  E_old,E_older,eavg = emsolve2.finite_diff(E_old,E_older,hx,hz,em_dt,h_dt,0.5,0.5,L,Z,X,np.transpose(sig),np.transpose(eps),np.transpose(mu))

        eavg = emsolve2.helmsolve(hx,hz,mu*mu*eps*mu0*eps0,E_inc)

        sig = mat2vec(np.transpose(sig))

        ndq = eavg*(sig) # nondimensionalized, scaled source term for heat equation
        ndq = np.transpose(vec2mat(ndq,nz,nx)) # for conveniently restricting to mat+ins only
        ndq = ndq[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] # restrict to mat+ins only
        ndq = mat2vec(ndq) # put back in vector form
```

```

            # Run thermal solver once, and *only* within the insulation and load
            temp = thermsolve2.second_try_diml(temp,hx[ins_start_x:ins_end_x],hz[
                ins_start_z:ins_end_z],750000*ndq,h_dt,th,phi,tempbc,trans_ins,temp_init,
                k_insmat,rho_insmat,cp_insmat) # returns new temperature field as vector

            # Make temperature a matrix, so it's easier to manipulate

            temp = vec2mat(temp,n_ins_x,n_ins_z)

            # Find theta values corresponding to new temperatures and heating rates at
                each point in load

            theta_integrand_new = (np.exp(-Q/(R*temp[x_start:x_end,z_start:z_end])))/(
                temp[x_start:x_end,z_start:z_end])
            theta = theta + 0.5*(theta_integrand_old + theta_integrand_new)*h_dt
            theta_integrand_old = theta_integrand_new
            lntheta = np.log(theta)

            # Update density in load using MSC and computed theta-values

            rho_mat = rhofun(lntheta) # gets density relative to that of bulk solid
            rho_avg_new = np.mean(rho_mat) # average density value in sample

            rho_mat = rho_mat * bulkdens_mat # dimensionalize the density

            # Find the x-index of maximum density value in load

            rmi,rmj = np.unravel_index(rho_mat.argmax(),rho_mat.shape) # rmi is index in
                material
            rho_max_ins = x_start + rmi # index in insulation + material
            rho_max = mat_start_x + rmi # index in entire domain

            # Compute total shrinkage within material based on density change and
                conservation of mass

            height_new = height_old*rho_avg_old/rho_avg_new # new volume of material
            x_thr = x[rho_max]-height_old+height_new # material between x_thr and
                rho_max disappears

            if x_thr > x[rho_max+1]: # x_thr is less than one spatial step from rho_max
```

```
                     # Then don't change the volume this time around; wait for more possible shrinkage in next time step
                     height_new = height_old
                     printstring = "\tShrinkage is less than the length of a single spatial step in material; not simulating shrinkage\n\tPercent of original length remains "+str(100*height_new/mat_height)+"\n\tNumber of nodes within material remains "+str(mat_end_x-mat_start_x)+"\n"
                     #logfile.write(printstring)
                     staycount = staycount + 1

              elif x_thr > h_2: # x_thr is more than one spatial step from rho_max, but still within material
                     the_ind = np.max(np.where(x<x_thr)) # index just above x_thr
                     height_new = height_old-(x[rho_max]-x[the_ind]) # the actual new height (as x_thr likely landed between nodes) # THIS VIOLATES CONSERVATION OF MASS, BUT IF SPATIAL GRID SIZE IS SMALL ENOUGH, IT SHOULDN'T BE "TOO" WRONG
                     shrink = rho_max-the_ind # number of nodes to shrink by
                     printstring = "\tShrinkage by deleting material to the left of max density\n\tMaterial shrinks by "+str(shrink)+" nodes ("+str((height_old-height_new)*100)+" cm)\n\tNew length is "+str(100*height_new/mat_height)+" percent of original length\n\tNumber of nodes remaining in material is "+str(mat_end_x-mat_start_x-shrink)+"\n"
                     #logfile.write(printstring)
                     leftcount = leftcount + 1
                     #temp[x_start+shrink:rho_max+1,:]=temp[x_start:rho_max+1-shrink,:] # remove material between x_thr and rho_max, and shift remaining load material to right
                     #temp[shrink:x_start+shrink,:]=temp[:x_start,:] # shift insulation to right
                     #temp = temp[shrink:,:]
                     temp = np.r_[temp[:rho_max_ins-shrink+1,:],temp[rho_max_ins+1:,:]] # remove section from temp_old
                     rho[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = rho_mat # update load densities
                     rho[mat_start_x+shrink:rho_max+1,:]=rho[mat_start_x:rho_max+1-shrink,:] # remove section from rho
                     theta = np.r_[theta[:rmi-shrink+1,:],theta[rmi+1:,:]] # remove section from theta
```

```
                    lntheta = np.r_[lntheta[:rmi-shrink+1,:],lntheta[rmi+1:,:]] # remove section from lntheta
                    theta_integrand_old = np.r_[theta_integrand_old[:rmi+1-shrink,:],theta_integrand_old[rmi+1:,:]] # remove section from old theta integrand
                    ins_start_x = ins_start_x + shrink # update insulation start index
                    mat_start_x = mat_start_x + shrink # update material start index
                    height_old = height_new

            elif x_thr < h_2: # x_thr is *outside* the material! (*Lots* of shrinkage, or rho_max v close to bdry)
                    # Just get rid of enough material to the right of mat_startind
                    x_thr = h_2+height_new
                    the_ind = np.min(np.where(x>x_thr)) # index just to the right of x_thr
                    height_new = height_old - (x[the_ind]-x[mat_start_x]) # the actual new volume (x_thr likely btwn nodes) # VIOLATES CONSERVATION OF MASS, MAKE SURE SPATIAL GRID SIZE IS SMALL
                    shrink = the_ind+1-mat_start_x # number of nodes to shrink by
                    printstring = "\tShrinkage by deleting material to the right of left-hand boundary\n\tMaterial shrinks by "+str(shrink)+" nodes ("+str((height_old-height_new)*100)+" cm)\n\tNew length is "+str(100*height_new/mat_height)+" percent of original length\n\tNumber of nodes remaining in material is "+str(mat_end_x-mat_start_x-shrink)+"\n"
                    #logfile.write(printstring)
                    rightcount = rightcount + 1
                    temp = np.r_[temp[:x_start,:],temp[x_start+shrink:,:]]
                    rho[mat_start_x+shrink:mat_end_x+1,mat_start_z:mat_end_z+1] = rho_mat[shrink:,:]
                    theta = theta[shrink:,:]
                    lntheta = lntheta[shrink:,:]
                    theta_integrand_old = theta_integrand_old[shrink:,:]
                    mat_start_x = mat_start_x+shrink
                    ins_start_x = ins_start_x+shrink
                    height_old = height_new

            # Update material parameters, including dielectric properties, according to temperature change, density change, and shrinkage

            k = vec2mat(k,nx,nz)
            cp = vec2mat(cp,nx,nz)
            #eps = vec2mat(eps,nx,nz)
            sig = np.transpose(vec2mat(sig,nz,nx))
            #mu = vec2mat(mu,nx,nz)

            # Air parameters
            eps[:ins_start_x,:] = eps_air
            sig[:ins_start_x,:] = sig_air
            rho[:ins_start_x,:] = rho_air
            cp[:ins_start_x,:] = c_air
            k[:ins_start_x,:] = k_air
            mu[:ins_start_x,:] = mu_air

            # Insulation parameters
            eps[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = epsfun_ins(temp)
            sig[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = sigfun_ins(temp)
            rho[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = rhofun_ins(temp)
            cp[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = cfun_ins(temp)
            k[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = kfun_ins(temp)
            mu[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = mufun_ins(temp)

            # Load parameters
            if theta_dep_params: # then load parameters depend on theta
                eps[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = epsfun_mat(
                    lntheta)
                sig[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = sigfun_mat(
                    lntheta)
                cp[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = cfun_mat(lntheta
                    )
                k[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = kfun_mat(lntheta)
                mu[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = mufun_mat(
                    lntheta)
            else: # then load parameters depend on temp and rho
                eps[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = epsfun_mat(temp
                    [x_start:x_end,z_start:z_end],rho[mat_start_x:mat_end_x+1,
                    mat_start_z:mat_end_z+1]/bulkdens_mat)
                sig[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = sigfun_mat(temp
                    [x_start:x_end,z_start:z_end],rho[mat_start_x:mat_end_x+1,
                    mat_start_z:mat_end_z+1]/bulkdens_mat)
                cp[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = cfun_mat(temp[
                    x_start:x_end,z_start:z_end],rho[mat_start_x:mat_end_x+1,
                    mat_start_z:mat_end_z+1]/bulkdens_mat)
                k[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = kfun_mat(temp[x_start:x_end,z_start:z_end],rho[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1]/bulkdens_mat)
                mu[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1] = mufun_mat(temp[x_start:x_end,z_start:z_end],rho[mat_start_x:mat_end_x+1,mat_start_z:mat_end_z+1]/bulkdens_mat)

         # Update spatial grid size here, too, if necessary, depending on whether the dielectric properties make the wavelength in material significantly shorter...

         elapsed_time = elapsed_time + h_dt
         loopits = loopits + 1

  instemps = np.r_[mat2vec(temp[:x_start,:]),mat2vec(temp[x_end:,:]),mat2vec(temp[x_start:x_end,:z_start]),mat2vec(temp[x_start:x_end,z_end:])]-273.15
  loadtemps = temp[x_start:x_end,z_start:z_end]-273.15
  max_ins = np.max(instemps)
  min_ins = np.min(instemps)
  mean_ins = np.mean(instemps)
  max_load = np.max(loadtemps)
  min_load = np.min(loadtemps)
  mean_load = np.mean(loadtemps)

  # Print text version of results to logfile
  printstring = ('\nAt time ' + str(elapsed_time) + ' sec...\n\tMax value of electric field is ' + str(np.max(eavg)) + ' V/m\n\tMin value of electric field is ' + str(np.min(eavg)) + ' V/m\n\tMean value of electric field is ' + str(np.mean(eavg))+' V/m\n\tMax temp in insulation is ' + str(max_ins) + ' degC\n\tMin temp in insulation is ' + str(min_ins) + ' degC\n\tMean temp in insulation is ' + str(mean_ins) + ' degC\n\tMax temp in load is ' + str(max_load) + ' degC\n\tMin temp in load is ' + str(min_load) + ' degC\n\tMean temp in load is ' + str(mean_load) + ' degC\n\tMean density in material is '+str(100*rho_avg_new)+" percent of bulk density\n\tSince last printed results, material boundary did not change "+str(staycount)+" times\n\tSince last printed results, material immediately to the right of boundary was removed "+str(rightcount)+" times\n\tSince last printed results, material immediately to the left of maximum density was removed "+str(leftcount)+" times\n\tNew material height is "+str(100*height_new/mat_height)+" percent of original height\n\tNumber of nodes remaining in material is "+str((mat_end_x-mat_start_x)*(n_mat_z))+"\n")
  logfile.write(printstring)
```

```

       T_maxes = np.r_[T_maxes, max_load ]
       T_means = np.r_[T_means, mean_load ]
       load_rhos = np.r_[load_rhos,100*rho_avg_new]
       lnt_avgs = np.r_[lnt_avgs,np.mean(lntheta)]
       plottingtimes = np.r_[plottingtimes, elapsed_time]

       itno = itno + 1

completetime = time.clock()
printstring = "\n\nSimulation complete. Took "+str(completetime-loopstarttime)+" seconds
    to complete simulation loop\n\nSaving animations...\n"
logfile.write(printstring)

# Plot evolution of maximum temperature in load
plt.figure(60)
plt.plot(plottingtimes,T_maxes,'r-',label='Max temp in load')
plt.plot(plottingtimes,T_means,'b-',label='Mean temp in load')
plt.legend(loc='upper left')
plt.xlabel('Time [sec]')
plt.ylabel('Temperature [degC]')
plt.title('Evolution of mean and maximum temperature in load')
plt.savefig(savestring+'temp_evol.png')
tikz_save(savestring+'temp_evol.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

# Plot evolution of density wrt time
plt.figure(61)
plt.plot(plottingtimes,load_rhos)
plt.xlabel('Time [sec]')
plt.ylabel('Density relative to bulk solid density [%]')
plt.title('Evolution of load density')
plt.savefig(savestring+'dens_time_evol.png')
tikz_save(savestring+'dens_time_evol.tex', figureheight = '\\figureheight', figurewidth = '
    \\figurewidth',show_info = False )

# Plot evolution of density wrt lnTheta
plt.figure(62)
plt.plot(lnt_avgs[1:],load_rhos[1:])
plt.xlabel(r'$\ln(\Theta(t,T(t)))$ $\left[\ln(\frac{s}{K})\right]$')
plt.ylabel('Density relative to bulk solid density [%]')
plt.title('Evolution of load density')
plt.savefig(savestring+'dens_lnt_evol.png')
tikz_save(savestring+'dens_lnt_evol.tex', figureheight = '\\figureheight', figurewidth = '
    \\figurewidth',show_info = False )

# Plot final electric field
plt.figure(63) # static image of field
plt.clf()
plt.contourf(100*Z,100*X,np.fliplr(np.transpose(vec2mat(eavg,nz,nx))),100)
plt.colorbar()
plt.xlabel('Position along domain [cm]')
plt.ylabel('Position along domain [cm]')
plt.title('Envelope of electric field [V^2/m^2] at t=' + str(elapsed_time) + ' sec')
plt.savefig(savestring+'efieldfin.png')
tikz_save(savestring+'efieldfin.tex', figureheight = '\\figureheight', figurewidth = '\\
    figurewidth',show_info = False )

T = (temp_init-273.15)*np.ones(np.shape(X)) # dimensional temperature in entire cavity (
    air is assumed constant temp)
T[ins_start_x:ins_end_x+1,ins_start_z:ins_end_z+1] = temp-273.15 # update plotting matrix
    for temperature

# Plot final temperature field
plt.figure(64) # update static image of field
plt.clf()
plt.contourf(100*Z,100*X,np.fliplr(T),100)
plt.colorbar()
plt.xlabel('Position along domain [cm]')
plt.ylabel('Position along domain [cm]')
plt.title('Temperature [C] at t=' + str(elapsed_time) + ' seconds')
plt.savefig(savestring+'tempfin_wholecav.png')
tikz_save(savestring+'tempfin_wholecav.tex', figureheight = '\\figureheight', figurewidth =
     '\\figurewidth',show_info = False )

startmovieclock = time.clock()
# Make movies
#-framerate : number of frames (images) per second
#-c:v libx264 - the video codec is libx264 (H.264).
#-profile:v high - use H.264 High Profile (advanced features, better quality).
#-crf 20 - constant quality mode, very high quality (lower numbers are higher quality, 18
    is the smallest you would want to use).
```

```
#-pix_fmt yuv420p - use YUV pixel format and 4:2:0 Chroma subsampling

# Electric field
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'efield%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'efield.mp4')
logfile.write("\tSaved electric field animation\n")

# Temperature field
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'temp_wholecav%d.png -y -loglevel
    quiet -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'
    temp_wholecav.mp4')
logfile.write("\tSaved temperature field animation\n")

# Mechanical deformation
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'efield%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'efield.mp4')
logfile.write("\tSaved mechanical deformation animation\n")

# Permittivity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'eps_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'eps_evol.mp4')
logfile.write("\tSaved permittivity animation\n")

# Electrical conductivity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'sig_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'sig_evol.mp4')
logfile.write("\tSaved electrical conductivity animation\n")

# Density
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'rho_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'rho_evol.mp4')
logfile.write("\tSaved density animation\n")

# Thermal conductivity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'c_evol%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'c_evol.mp4')
logfile.write("\tSaved thermal conductivity animation\n")

# Specific heat capacity
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'k_evol%d.png -y -loglevel quiet -
    c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'k_evol.mp4')
```

```
logfile.write("\tSaved specific heat capacity animation\n")

# Permeability
os.system('ffmpeg -framerate 24 -i '+hiddensavestring+'mu_evol%d.png -y -loglevel quiet
    -c:v libx264 -profile:v high -crf 20 -pix_fmt yuv420p '+savestring+'mu_evol.mp4')
logfile.write("\tSaved magnetic permeability animation\n")

for fname in delfiles:
      os.remove(fname)
logfile.write('\tDeleted individual frame files.\n\tDone; took '+str(time.clock()-
    startmovieclock)+' seconds to complete movie processing')

finaltime=time.clock()
printstring = "\n\nSimulation completed on "+time.strftime("%A, %B %d, %Y")+" at "+time.
    strftime("%H:%M:%S %Z")+".\nTook "+str(finaltime-initialstarttime)+" seconds to
    complete entire simulation.\n"
logfile.write(printstring)

logfile.close()
print('Simulation complete; see directory '+savedir+' for plots and text outputs of results
    .')
```

H.3 MATLAB Implementation of the Coupled Solver for the One-Dimensional Microwave Sintering Problem

```
function fullsolve1(total_time)
% function fullsolve1(total_time)
%
% Performs transient analysis of the electric field for a
% one-dimensional domain with a constant power source at the left-hand
% side. See problem description in PDF file of same directory.
% Uses a constant time step and uniform node spacing (for now).

%figure(1); clf; figure(2); clf; figure(3); clf;

% Physical setup
L=0.248; %length of domain [m]
P=1000; % [W] power supplied by magnetron at left-hand endpoint
omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves at 2.45GHz
beta=pi/L; % [1/m] propagation constant
mu0=pi*4e-7; %[N/A^2] permeability of free space
```

```

% Nodes and spacing
n=50; % number of (uniformly spaced) spatial nodes
x=linspace(0,L,n); %vector of x-values
h=x(2:end)-x(1:n-1); %h-values (as spacing is uniform, h is a multiple of ones vector)

% Initial temperature
temp=293*ones(size(x))'; % room temp in kelvin

% Time scenario
em_dt=1e-3; % length of time step of em solve [sec]
h_dt=1e-3; % length of time step of heat solve (i.e., how long to nuke before solving heat
     transfer) [sec]

if nargin<1,total_time=10*h_dt; end % total length of processing time [sec], if not
    specified then run for 10 cycles of thermal prob

% Load material: zirconia parameters taken from [Yakovlev & Ceralink]
t=273+[25 69 100 139 181 228 276 324 371 420 471 523 574 636 698 752 809 865 921 973 1019
    1065 1100];
epses=[6.69 5.86 5.78 5.75 5.77 5.82 5.90 5.98 6.08 6.18 6.32 6.47 6.60 6.77 6.97 7.22 7.53
     7.93 8.53 9.44 10.46 12.46 14.77];
sigmas=[0.0258 0.0045 0.0033 0.0029 0.0036 0.0043 0.0050 0.0058 0.0078 0.0121 0.0185 0.0288
     0.0442 0.0664 0.0975 ...
  0.1416 0.2003 0.2786 0.4083 0.5942 0.8220 1.2190 1.6661];
cs=[0.217 0.324 0.363 0.398 0.426 0.450 0.470 0.487 0.501 0.514 0.526 0.537 0.547 0.558
    0.568 0.575 0.583 0.590 ...
  0.597 0.603 0.607 0.612 0.615];
rhos=1e6*[2.848 2.844 2.841 2.838 2.834 2.830 2.826 2.821 2.817 2.813 2.809 2.804 2.800
    2.794 2.789 2.785 2.780 2.775 ...
  2.770 2.766 2.762 2.758 2.755];
ks=100*[0.00198 0.00290 0.00320 0.00344 0.00362 0.00373 0.00381 0.00385 0.00381 0.00391
    0.00399 0.00407 0.00414 0.00405 0.00412 ...
  0.00417 0.00421 0.00426 0.00430 0.00433 0.00436 0.00439 0.00441];
% FALSIFYING DATA TO SHOW OPERATION OF DENSIFICATION WITHIN SOLVER
  t = [t(1:5),0.99*t(6),t(6:end)];
  rhos=[rhos(1:5),2*rhos(6),2*rhos(6:end)];
  epses=[epses(1:5),epses(6),epses(6:end)];
  sigmas=[sigmas(1:5),sigmas(6),sigmas(6:end)];
  cs=[cs(1:5),cs(6),cs(6:end)];
  ks=[ks(1:5),ks(6),ks(6:end)];
```

```

epspoly=pchip(t,epses); % interpolate p/w polynomials
sigpoly=pchip(t,sigmas);
cpoly=pchip(t,cs);
rhopoly=pchip(t,rhos);
kpoly=pchip(t,ks);
mupoly=pchip(t,ones(size(t)));
   % epsfun=@(T) ppval(epspoly,T);%-.083*T+57.005; % function giving relationship of eps
       with temperature
   % mufun=@(T) 1; % function giving relationship of mu with temperature
   % sigfun=@(T) ppval(sigpoly,T); %0.00676*T+0.9939; % function giving relationship of
       sigma with temperature
   % cfun=@(T) ppval(cpoly,T);
   % kfun=@(T) ppval(kpoly,T);
   % rhofun=@(T) ppval(rhopoly,T);
   % Checking the interpolants
      %ptemp=linspace(270,300);
      %figure(3); clf; plot(ptemp,epsfun(ptemp)); title('\epsilon''');
      %figure(4); clf; plot(temps,mufun(temps)); title('\mu')
      %figure(5); clf; plot(ptemp,sigfun(ptemp)); title('\sigma');
      %figure(6); clf; plot(ptemp,cfun(ptemp)); title('c');
      %figure(7); clf; plot(ptemp,kfun(ptemp)); title('k');
      %figure(8); clf; plot(ptemp,rhofun(ptemp)); title('\rho');

R=30e-9; % grain radius of material [m]
Ea=27333; % activation energy of material

% Material parameters at initial temperature
rhoinit=ppval(rhopoly,293);
kinit=ppval(kpoly,293);
cinit=ppval(cpoly,293);
epsinit=ppval(epspoly,293);
muinit=ppval(mupoly,293);
siginit=ppval(sigpoly,293);

% Air parameters
mu_air=1; % (unitless) relative permeability of air
sigma_air=0; % [S/m] electrical conductivity of air
eps1_air=1; % (unitless) relative permittivity of air
rho0_air=2; % density of air
cp0_air=1; % specific heat capacity of air
```

```
k0_air=0.024; % thermal conductivity of air

% Elemental values of physical properties — sets up material vectors
lim1=floor((n−1)/3); lim2=ceil(2*(n−1)/3); % limits for L/3 and 2L/3 (material occupies
    middle third of cavity)
mu=[mu_air*ones(lim1,1); muinit*ones(lim2−lim1,1); mu_air*ones((n)−lim2,1)]';
sig=[sigma_air*ones(lim1,1); siginit*ones(lim2−lim1,1); sigma_air*ones((n)−lim2,1)]';
eps1=[eps1_air*ones(lim1,1); epsinit*ones(lim2−lim1,1); eps1_air*ones((n)−lim2,1)]';
rho=[rho0_air*ones(lim1,1); rhoinit*ones(lim2−lim1,1); rho0_air*ones((n)−lim2,1)]';
cp=[cp0_air*ones(lim1,1); cinit*ones(lim2−lim1,1); cp0_air*ones((n)−lim2,1)]'.*(1−temp');
k=[k0_air*ones(lim1,1); kinit*ones(lim2−lim1,1); k0_air*ones((n)−lim2,1)]'.*(1−1.5*temp')
    ;
por=zeros(lim2−lim1,1); % initial porosity

% Solution routine
time=0; % initialize elapsed time
numits=0; % initialize number of iterations

E_old = zeros(n,1); pow=(2/L)*sqrt(2*P*omega*mu0/beta); % initialize e−field
E_old(1)=pow;
E_older=E_old;

p1=linspace(100*x(lim1),100*x(lim2)); % for plotting

%% Movie: temperature profile in space over time
#moviecount=1;
#hft=figure(2);
#rect_t=get(hft,'Position');
#rect_t(1:2)=[0 0];

fprintf('At time 0 sec, object is at 100 percent of original length\n');
while time<total_time
while numits<1/h_dt %for printing shrinkage every second instead of every timestep

 [E_new,E_old] = emsolve1_fd(E_old,E_older,x,mu,sig,eps1,h,em_dt,h_dt,time);
 eavg = (E_new.^2)';

% figure(1); plot(100*x,eavg);
% title(strcat('Modulus of electric field at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length [cm]'); ylabel('Electric field modulus');

```

```
  temp_new=thermsolve1_fd(temp,h,cp,rho,k,eavg,h_dt,sig);

  [cp,rho,k,mu,eps1,sig,por,LL,lim1,rdens_avg]=update_params(x,temp,temp_new,rhoinit,...
     rho0_air,cp0_air,cinit,k0_air,kinit,eps1_air,epspoly,mu_air,mupoly,sigma_air,sigpoly,
         lim1,lim2,por,h_dt,Ea,R,cpoly,kpoly,rhopoly);

  E_older = E_old;
  E_old = E_new;

  temp=temp_new;
  time=time+h_dt;
  numits=numits+1;
end
  fprintf('At time %g sec, object length is %g percent of original length\n',time,100/
      rdens_avg);
  p2=linspace(100*x(lim1),100*x(lim2));
  figure(2); hold off; plot(x*100,temp-273,'b',p2,20,'r-');
  axis([0 25 0 300]);
  title(strcat('Temperature distribution at t=',num2str(time,'%11.3g'),' seconds'));
  xlabel('Length [cm]'); ylabel('Temperature [C]');
    #Mt(:,moviecount)=getframe(hft,rect_t); %#ok<AGROW>
    #moviecount=moviecount+1;

  numits=0;
end

fprintf('Max temperature in object at time t=%g is T=%g degC\n',time,max(temp)-273);

saveas(2,'temp_fin.fig','fig'); saveas(2,'fig_tempfin.jpg','jpg');
#save('mov_temp.mat','Mt');
#movie2avi(Mt,'mov_temp');

figure(3); hold off; plot(x*100,E_new,'b',x(lim1:lim2)*100,20,'r-');
axis([0 25 -3e4 3e4]);
title(strcat('Electric field at t=',num2str(time,'%11.3g'),' seconds'));
xlabel('Length [cm]'); ylabel('Electric field intensity [V/m]');
saveas(3,'efield.fig','fig'); saveas(3,'fig_efield.jpg','jpg');

%p2=linspace(100*x(lim1),100*x(lim2));
figure(10); plot(p1,zeros(size(p1)),'r',p2,0.1*ones(size(p2)),'b');
legend('Initial configuration','Final configuration');
```

```
axis([0 L*100 -1 10]); title('Geometrical configuration before and after processing');
saveas(10,'geomconfig.fig','fig'); saveas(10,'fig_geomconfig.jpg','jpg');

end

function [cp,rho,k,mu,eps,sig,por,LL,lim1,rdens_avg]=update_params(x,temp,temp_new,...
   rho0,rho0_air,cp0_air,cp0,k0_air,k0,eps_air,epspoly,mu_air,mupoly,sigma_air,sigpoly,
       lim1,lim2,por,h_dt,Ea,R,cpoly,kpoly,rhopoly)

%n=length(x);
L=x(end)-x(1);

% compute porosity according to [Su & Johnson, 1996]

%integrand_temp=(1./temp(lim1+1:lim2)).*exp((-Ea/R)./temp(lim1+1:lim2));
%integrand_temp_new=(1./temp_new(lim1+1:lim2)).*exp((-Ea/R)./temp_new(lim1+1:lim2));
%por=por(1:lim2-lim1)+0.5*(integrand_temp+integrand_temp_new)/h_dt;

%por=[zeros(lim1,1),por,zeros(n-1-lim2,1)];

% compute actual density -- this is instead of using the porosity measurement above

br=rhopoly.breaks.';
cf=rhopoly.coefs;
[throw,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
rho=sum([three two one zero].*cf(inds,:),2);
%rho=rhofun(temp_new');
rdens=rho./rho0;

% compute relative density

%rdens=1-por;
rdens_avg=mean(rdens); %this is the average in the whole sample!

% compute shrinkage in terms of lim1 change (lim2 stays the same)

```

```
l=L/(3*rdens_avg); % new length of material
LL = ( x > 2*L/3-l & x < 2*L/3 ); % LL(i)=1 if node at i is within material, 0 if in air
oldlim1=lim1;
lim1=find(LL~=0, 1, 'first');

% compute actual density

rho = rho0_air*(1-LL) + rho0*rdens'.*LL;

% compute c_p
%cp=(cp0_air*(1-LL) + cp0*LL).*(1-temp_new'); % this is Olevsky's suggestion
br=cpoly.breaks.';
cf=cpoly.coefs;
[throw,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
cnew=sum([three two one zero].*cf(inds,:),2);
%fprintf('c diff = %g\n',cnew-ppval(cpoly,temp_new));

cp=cp0_air*(1-LL) + cnew'.*LL;

% compute k
%k=(k0_air*(1-LL) + k0*LL).*(1-1.5*temp_new'); % Olevsky
br=kpoly.breaks.';
cf=kpoly.coefs;
[throw,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
knew=sum([three two one zero].*cf(inds,:),2);
%fprintf('k diff = %g\n',knew-ppval(kpoly,temp_new));
k=k0_air*(1-LL) + knew'.*LL;

% compute eps
br=epspoly.breaks.';
cf=epspoly.coefs;
```

```
[throw,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
epsnew=sum([three two one zero].*cf(inds,:),2);
%fprintf('Eps diff = %g\n',epsnew-ppval(epspoly,temp_new));
eps=eps_air*(1-LL) + epsnew'.*LL;

% compute mu
br=mupoly.breaks.';
cf=mupoly.coefs;
[throw,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
munew=sum([three two one zero].*cf(inds,:),2);
%fprintf('Mu diff = %g\n',munew-ppval(mupoly,temp_new));
mu=mu_air*(1-LL) + munew'.*LL;

% compute sigma
br=sigpoly.breaks.';
cf=sigpoly.coefs;
[throw,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
signew=sum([three two one zero].*cf(inds,:),2);
%fprintf('Sigma diff = %g\n',signew-ppval(sigpoly,temp_new));
sig=sigma_air*(1-LL) + signew'.*LL;

end
```

H.4 MATLAB Implementation of the Coupled Solver for the Two-Dimensional Microwave Sintering Problem

```
function fullsolve2(total_time)
% function fullsolve2(total_time)
%
% Performs transient analysis of the electric field for a
% two-dimensional domain with a constant power source at the left-hand
% side. See problem description in PDF file of same directory.
% Uses a constant time step and uniform node spacing (for now).

%figure(1); clf; figure(2); clf; figure(3); clf;

% Physical setup
L=0.248; %length of domain [m]
H=0.124; %width of domain [m]
h0=H/3;
P=1000; % [W] power supplied by magnetron at left-hand endpoint
omega=2*pi*2.45e9; % [Hz] angular frequency of microwaves at 2.45GHz
beta=pi/L; % [1/m] propagation constant
mu0=pi*4e-7; %[N/A^2] permeability of free space

% Nodes and spacing
Nx=50; % number of (uniformly spaced) spatial nodes in the x-direction
Ny=25; % number of (uniformly spaced) spatial nodes in the y-diretion
x=linspace(0,L,Nx); %vector of x-values
y=linspace(0,H,Ny); % vector of y-values
hx=x(2:end)-x(1:end-1); %hx-values (as spacing is uniform, hx is a multiple of ones
    vector)
hy=y(2:end)-y(1:end-1); %hy-values (as spacing is uniform, hy is a multiple of ones
    vector)

[X,Y]=meshgrid(x,y); % X has x-vectors as rows repeated Ny many times, Y has y'-vectors as
     columns repeated Nx many times

% Initial temperature
temp=293*ones(Nx*Ny,1); % room temp in kelvin is the initial constant temperature over
    whole domain

% Time scenario
em_dt=1e-1; % length of time step of em solve [sec]
h_dt=1e-1; % length of time step of heat solve (i.e., how long to nuke before solving heat
     transfer) [sec]

```

```
if nargin<1,total_time=h_dt; end % total length of processing time [sec], if not specified
    then run for 10 cycles of thermal prob

% Load material: zirconia parameters taken from [Yakovlev & Ceralink]
t=273+[25 69 100 139 181 228 276 324 371 420 471 523 574 636 698 752 809 865 921 973 1019
    1065 1100];
epses=[6.69 5.86 5.78 5.75 5.77 5.82 5.90 5.98 6.08 6.18 6.32 6.47 6.60 6.77 6.97 7.22 7.53
     7.93 8.53 9.44 10.46 12.46 14.77];
sigmas=[0.0258 0.0045 0.0033 0.0029 0.0036 0.0043 0.0050 0.0058 0.0078 0.0121 0.0185 0.0288
     0.0442 0.0664 0.0975 ...
  0.1416 0.2003 0.2786 0.4083 0.5942 0.8220 1.2190 1.6661];
cs=[0.217 0.324 0.363 0.398 0.426 0.450 0.470 0.487 0.501 0.514 0.526 0.537 0.547 0.558
    0.568 0.575 0.583 0.590 ...
  0.597 0.603 0.607 0.612 0.615];
rhos=1e6*[2.848 2.844 2.841 2.838 2.834 2.830 2.826 2.821 2.817 2.813 2.809 2.804 2.800
    2.794 2.789 2.785 2.780 2.775 ...
  2.770 2.766 2.762 2.758 2.755];
ks=100*[0.00198 0.00290 0.00320 0.00344 0.00362 0.00373 0.00381 0.00385 0.00381 0.00391
    0.00399 0.00407 0.00414 0.00405 0.00412 ...
  0.00417 0.00421 0.00426 0.00430 0.00433 0.00436 0.00439 0.00441];
% FALSIFYING DATA TO SHOW OPERATION OF DENSIFICATION WITHIN SOLVER
  t = [t(1:5),0.99*t(6),t(6:end)];
  rhos=[rhos(1:5),2*rhos(6),2*rhos(6:end)];
  epses=[epses(1:5),epses(6),epses(6:end)];
  sigmas=[sigmas(1:5),sigmas(6),sigmas(6:end)];
  cs=[cs(1:5),cs(6),cs(6:end)];
  ks=[ks(1:5),ks(6),ks(6:end)];

epspoly=pchip(t,epses); % interpolate p/w polynomials
sigpoly=pchip(t,sigmas);
cpoly=pchip(t,cs);
rhopoly=pchip(t,rhos);
kpoly=pchip(t,ks);
mupoly=pchip(t,ones(size(t)));
    % epsfun=@(T) ppval(epspoly,T);%-.083*T+57.005; % function giving relationship of eps
        with temperature
    % mufun=@(T) 1; % function giving relationship of mu with temperature
    % sigfun=@(T) ppval(sigpoly,T); %0.00676*T+0.9939; % function giving relationship of
        sigma with temperature
    % cfun=@(T) ppval(cpoly,T);
    % kfun=@(T) ppval(kpoly,T);
```

```
   % rhofun=@(T) ppval(rhopoly,T);
   % Checking the interpolants
      %ptemp=linspace(270,1e5);
      %figure(3); clf; plot(ptemp,epsfun(ptemp)); title('\epsilon''');
      %figure(4); clf; plot(temps,mufun(temps)); title('\mu')
      %figure(5); clf; plot(ptemp,sigfun(ptemp)); title('\sigma');
      %figure(6); clf; plot(ptemp,cfun(ptemp)); title('c');
      %figure(7); clf; plot(ptemp,kfun(ptemp)); title('k');
      %figure(8); clf; plot(ptemp,rhofun(ptemp)); title('\rho');

R=30e-9; % grain radius of material [m]
Ea=27333; % activation energy of material

% Material parameters at initial temperature
rhoinit=ppval(rhopoly,293);
kinit=ppval(kpoly,293);
cinit=ppval(cpoly,293);
epsinit=ppval(epspoly,293);
muinit=ppval(mupoly,293);
siginit=ppval(sigpoly,293);

% Air parameters
mu_air=1; % (unitless) relative permeability of air
sigma_air=0; % [S/m] electrical conductivity of air
eps1_air=1; % (unitless) relative permittivity of air
rho0_air=2; % density of air
cp0_air=1; % specific heat capacity of air
k0_air=0.024; % thermal conductivity of air

% Elemental values of physical properties - sets up material matrices
lim1=floor((Nx-1)/3); lim2=ceil(2*(Nx-1)/3); % limits for L/3 and 2L/3 (material occupies
    middle third of cavity)
lim3=ceil(2*(Ny-1)/3); % limit for 2H/3 (material occupies bottom third of cavity)
mu=[mu_air*ones(lim1,1); muinit*ones(lim2-lim1,1); mu_air*ones(Nx-lim2,1)]'; % forms the
    base x-vector for mu
[mu,~]=meshgrid(mu,y); mu(1:lim3,:)=mu_air; % makes mu a matrix and puts air in the top 2/3
sigma=[sigma_air*ones(lim1,1); siginit*ones(lim2-lim1,1); sigma_air*ones(Nx-lim2,1)]';
[sigma,~]=meshgrid(sigma,y); sigma(1:lim3,:)=sigma_air; % makes sigma a matrix and puts air
     in the top 2/3
eps1=[eps1_air*ones(lim1,1); epsinit*ones(lim2-lim1,1); eps1_air*ones(Nx-lim2,1)]';
```

```
[eps1,~]=meshgrid(eps1,y); eps1(1:lim3,:)=eps1_air; % makes eps1 a matrix and puts air in
    the top 2/3
rho=[rho0_air*ones(lim1,1); rhoinit*ones(lim2-lim1,1); rho0_air*ones(Nx-lim2,1)]';
[rho,~]=meshgrid(rho,y); rho(1:lim3,:)=rho0_air; % makes rho a matrix and puts air in the
    top 2/3
cp=[cp0_air*ones(lim1,1); cinit*ones(lim2-lim1,1); cp0_air*ones(Nx-lim2,1)]';
[cp,~]=meshgrid(cp,y); cp(1:lim3,:)=cp0_air; % makes cp a matrix and puts air in the top
    2/3
k=[k0_air*ones(lim1,1); kinit*ones(lim2-lim1,1); k0_air*ones(Nx-lim2,1)]';
[k,~]=meshgrid(k,y); k(1:lim3,:)=k0_air; % makes k a matrix and puts air in the top 2/3
por=zeros(lim2-lim1,1); % initial porosity

%figure(7); clf; hold off; surf(X*100,flipud(Y*100),rho); view(0,90); colorbar;
%title('Init rho');
%xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('1 where there
    is material, 0 where not');

LL=zeros(Ny,Nx); LL(lim3:end,lim1:lim2)=1;
LL=reshape(LL',1,[])';

% Solution routine
time=0; % initialize elapsed time
numits=0; % initialize number of iterations

E_old = zeros(Ny*Nx,1); pow=(2/L)*sqrt(2*P*omega*mu0/beta); % initialize e-field
E_old((1:(Ny-1))*Nx+1)=pow; % electric field is fixed at pow on input port side
E_older=E_old;

%figure(1); clf; hold on;
moviecount=1;
hft=figure(2);
rect_t=get(hft,'Position');
rect_t(1:2)=[0 0];

if total_time<1, numlim=total_time/h_dt; else numlim=1/h_dt; end %for printing shrinkage
    every second instead of every timestep
fprintf('At time 0 sec, object height is 100 percent of original height\n');
while time<total_time
while numits<numlim %for printing shrinkage every second instead of every timestep

 [E_new,E_old] = emsolve2_fd(E_old,E_older,X,Y,Nx,Ny,mu,sigma,eps1,hx,hy,em_dt,h_dt,time);
 eavg = (E_new.^2);

% figure(1); surf(100*X,100*Y,(reshape(eavg,Nx,Ny))');
% title(strcat('Modulus of electric field at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length [cm]'); ylabel('Electric field modulus');

   temp_new=thermsolve2_fd(temp,hx,hy,Nx,Ny,X,Y,cp,rho,k,eavg,h_dt,sigma,time);

  [cp,rho,k,mu,eps1,sigma,LL,rdens_avg]=update_params(h0,LL,L,X,Y,Nx,Ny,temp_new,rhoinit
      ,...
     rho0_air,cp0_air,cinit,k0_air,kinit,eps1_air,epspoly,mu_air,mupoly,sigma_air,sigpoly,
         cpoly,kpoly,rhopoly,time,h_dt);

  E_older = E_old;
  E_old = E_new;

  temp=temp_new;
  time=time+h_dt;
  numits=numits+1;
end
  fprintf('At time %g sec, object height is %g percent of original height\n',time,100/
      rdens_avg);
  figure(2); hold off; surf(X*100,flipud(Y*100),(reshape(temp,Nx,Ny))'-273); view(0,90);
      colorbar;
  title(strcat('Temperature distribution at t=',num2str(time,'%11.3g'),' seconds'));
  xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Temperature [C
      ]');
    Mt(:,moviecount)=getframe(hft,rect_t); %#ok<AGROW>
    moviecount=moviecount+1;

  numits=0;
end


saveas(2,'temp_fin2.fig','fig'); saveas(2,'fig_tempfin2.jpg','jpg');
save('mov_temp2.mat','Mt');
movie2avi(Mt,'mov_temp2');

figure(3); hold off; surf(100*X,100*Y,(reshape(eavg,Nx,Ny))');
title(strcat('Electric field at t=',num2str(time,'%11.3g'),' seconds'));
xlabel('Length [cm]'); ylabel('Electric field intensity [V/m]');
```

```
saveas(3,'efield2.fig','fig'); saveas(3,'fig_efield2.jpg','jpg');

end

function [cp,rho,k,mu,eps,sigma,LL,rdens_avg]=update_params(h0,LL,L,X,Y,Nx,Ny,temp_new,...
   rho0,rho0_air,cp0_air,cp0,k0_air,k0,eps_air,epspoly,mu_air,mupoly,sigma_air,sigpoly,
       cpoly,kpoly,rhopoly,time,h_dt)

% compute actual density -- this is instead of using the porosity measurement

br=rhopoly.breaks.';
cf=rhopoly.coefs;
[~,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
rho=sum([three two one zero].*cf(inds,:),2);
%rho=rhofun(temp_new');
rdens=rho./rho0;

% compute relative density
rdens_avg=mean(rdens(LL==1)); %this is the average in the sample

%figure(9); clf; hold off; surf(X*100,flipud(Y*100),reshape(LL,Nx,Ny)'); view(0,90);
    colorbar;
%title(strcat('Space occupied by material at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
%xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('1 where there
    is material, 0 where not');


% compute shrinkage in terms of height change (width stays the same)
h_new=h0/rdens_avg;
LL_ind = ( X > L/3 & X < 2*L/3 & Y > max(max(Y))-h_new ); % LL(i)=1 if node at i is within
     material, 0 if in air
LL=zeros(size(Y)); LL(LL_ind)=1;


LL=reshape(LL',1,[])';

```

```
% compute actual density
rho = rho0_air*(1-LL) + rho0*rdens_avg.*LL; rho=reshape(rho,Nx,Ny)';

%figure(3); clf; hold off; surf(X*100,Y*100,rho); view(0,90); colorbar;
% title(strcat('Density distribution at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Density [g/m
    ^2]');



% compute c_p
%cp=(cp0_air*(1-LL) + cp0*LL).*(1-temp_new'); % this is Olevsky's suggestion
br=cpoly.breaks.';
cf=cpoly.coefs;
[~,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
cnew=sum([three two one zero].*cf(inds,:),2);
%fprintf('c diff = %g\n',cnew-ppval(cpoly,temp_new));

cp=cp0_air*(1-LL) + cnew.*LL; cp=reshape(cp,Nx,Ny)';

%figure(4); clf; hold off; surf(X*100,Y*100,cp); view(0,90); colorbar;
% title(strcat('Specific Heat Capacity at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Specific heat
    capacity [ ]');

% compute k
%k=(k0_air*(1-LL) + k0*LL).*(1-1.5*temp_new'); % Olevsky
br=kpoly.breaks.';
cf=kpoly.coefs;
[~,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
knew=sum([three two one zero].*cf(inds,:),2);
%fprintf('k diff = %g\n',knew-ppval(kpoly,temp_new));
```

```
k=k0_air*(1-LL) + knew.*LL; k=reshape(k,Nx,Ny)';

%figure(5); clf; hold off; surf(X*100,Y*100,k); view(0,90); colorbar;
% title(strcat('Thermal conductivity at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Thermal
    conductivity [ ]');

% compute eps
br=epspoly.breaks.';
cf=epspoly.coefs;
[~,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
epsnew=sum([three two one zero].*cf(inds,:),2);
%fprintf('Eps diff = %g\n',epsnew-ppval(epspoly,temp_new));
eps=eps_air*(1-LL) + epsnew.*LL; eps=reshape(eps,Nx,Ny)';

%figure(6); clf; hold off; surf(X*100,Y*100,eps); view(0,90); colorbar;
% title(strcat('Real part of complex permittivity at t=',num2str(time+h_dt,'%11.3g'),'
    seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Relative \
    varepsilon''');

% compute mu
br=mupoly.breaks.';
cf=mupoly.coefs;
[~,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
munew=sum([three two one zero].*cf(inds,:),2);
%fprintf('Mu diff = %g\n',munew-ppval(mupoly,temp_new));
mu=mu_air*(1-LL) + munew.*LL; mu=reshape(mu,Nx,Ny)';

%figure(7); clf; hold off; surf(X*100,Y*100,mu); view(0,90); colorbar;
% title(strcat('Magnetic permeability at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('Relative \mu')
    ;

% compute sigma
br=sigpoly.breaks.';
cf=sigpoly.coefs;
[~,inds]=histc(temp_new,[-inf; br(2:end-1); +inf]);
t_shf=temp_new - br(inds);
zero=ones(size(t_shf));
one=t_shf;
two=one.*t_shf;
three=two.*t_shf;
signew=sum([three two one zero].*cf(inds,:),2);
%fprintf('Sigma diff = %g\n',signew-ppval(sigpoly,temp_new));
sigma=sigma_air*(1-LL) + signew.*LL; sigma=reshape(sigma,Nx,Ny)';

%figure(8); clf; hold off; surf(X*100,Y*100,sigma); view(0,90); colorbar;
% title(strcat('Electrical conductivity at t=',num2str(time+h_dt,'%11.3g'),' seconds'));
% xlabel('Length L (x-dir) [cm]'); ylabel('Height H (y-dir) [cm]'); zlabel('\sigma');

end
```

Appendix I

Output Log Files from Example Simulations

I.1 One-Dimensional Simulation with Radiative Thermal Boundary Conditions

```
Simulation started Saturday, March 19, 2016 at 14:10:15 EDT.

Waveguide length is 43.3450764915 cm
Length of material is 4.81611961017 cm
Length of insulation on either side of material is 4.81611961017 cm
Input power is 1.0 kW
Frequency of radiation is 2.45 GHz
Initial temperature is 24.85 K

Determining optimal activation energy and density function...
      Using densification data from {Teng et al}...
      Attempting data fit to fantozzi sigmoid curve...
      Done; took 1.395361 seconds to find optimal activation energy and MSC.
      Optimal activation energy is 674 kJ/mol.

Interpolating measured data to find dielectric and thermal properties as functions of
    temperature and relative density...
      Assuming parameters are functions of ln(theta)...
      Done; took 6.617361 seconds to find functions for all dielectric and thermal
          material and insulation properties.

Setting up simulation...
      Spatial cell size in air is 0.86690152983 cm
      Spatial cell size in insulation 0.703150201446 cm
```

```
       Spatial cell size in material is 0.33174621403 cm
       Total number of cells in entire domain is 63
       Total number of cells in insulation+material is 29
       Total number of cells in material is 15
       Time step for electromagnetic solve is 0.001 sec
       Time step for thermal solve is 0.01 sec
       Total simulated processing time will be 3600 sec

Starting simulation loop...
       Using absorbing boundary condition for electromagnetic solver
       Using radiative boundary condition for thermal solver

At start of simulation...
       Max value of electric field is 0 V/m
       Min value of electric field is 0 V/m
       Mean value of electric field is 0.0 V/m
       Max temp in insulation is 24.85 degC
       Min temp in insulation is 24.85 degC
       Mean temp in insulation is 24.85 degC
       Max temp in load is 24.85 degC
       Min temp in load is 24.85 degC
       Mean temp in load is 24.85 degC
       Mean density in material is 52.3886478968 percent of bulk density

At time 60.0 sec...
       Max value of electric field is 539769.09957 V/m
       Min value of electric field is 35.6777777493 V/m
       Mean value of electric field is 179946.818375 V/m
       Max temp in insulation is 67.6502376345 degC
       Min temp in insulation is 32.3157149903 degC
       Mean temp in insulation is 48.3401729231 degC
       Max temp in load is 55.4384366231 degC
       Min temp in load is 32.1729881136 degC
       Mean temp in load is 36.9114219769 degC
       Mean density in material is 52.5359627498 percent of bulk density
       Since last printed results, material boundary did not change 6000 times
       Since last printed results, material immediately to the right of boundary was
           removed 0 times
       Since last printed results, material immediately to the left of maximum density was
           removed 0 times
       New material length is 100.0 percent of original length
       Number of nodes remaining in material is 15

At time 120.0 sec...
       Max value of electric field is 539769.099139 V/m
       Min value of electric field is 35.6777777209 V/m
       Mean value of electric field is 179946.818232 V/m
       Max temp in insulation is 104.547973592 degC
       Min temp in insulation is 35.8011893681 degC
       Mean temp in insulation is 68.0507042218 degC
       Max temp in load is 81.6366892959 degC
       Min temp in load is 39.6183091388 degC
       Mean temp in load is 48.4564473407 degC
       Mean density in material is 52.5359627498 percent of bulk density
       Since last printed results, material boundary did not change 6000 times
       Since last printed results, material immediately to the right of boundary was
           removed 0 times
       Since last printed results, material immediately to the left of maximum density was
           removed 0 times
       New material length is 100.0 percent of original length
       Number of nodes remaining in material is 15

⋮                        ⋮

At time 3540.00000003 sec...
       Max value of electric field is 539769.074613 V/m
       Min value of electric field is 35.6777760998 V/m
       Mean value of electric field is 179946.810055 V/m
       Max temp in insulation is 966.079874108 degC
       Min temp in insulation is 58.2494084623 degC
       Mean temp in insulation is 498.668488423 degC
       Max temp in load is 914.20623514 degC
       Min temp in load is 428.94836084 degC
       Mean temp in load is 627.012040199 degC
       Mean density in material is 52.5421609079 percent of bulk density
       Since last printed results, material boundary did not change 6000 times
       Since last printed results, material immediately to the right of boundary was
           removed 0 times
       Since last printed results, material immediately to the left of maximum density was
           removed 0 times
       New material length is 100.0 percent of original length
       Number of nodes remaining in material is 15
```

```

At time 3600.00000003 sec...
        Max value of electric field is 539769.074191 V/m
        Min value of electric field is 35.6777760719 V/m
        Mean value of electric field is 179946.809914 V/m
        Max temp in insulation is 971.279395997 degC
        Min temp in insulation is 58.4142986487 degC
        Mean temp in insulation is 501.628035739 degC
        Max temp in load is 920.754099881 degC
        Min temp in load is 434.33280418 degC
        Mean temp in load is 634.292135322 degC
        Mean density in material is 52.5432262188 percent of bulk density
        Since last printed results, material boundary did not change 6000 times
        Since last printed results, material immediately to the right of boundary was removed 0 times
        Since last printed results, material immediately to the left of maximum density was removed 0 times
        New material length is 100.0 percent of original length
        Number of nodes remaining in material is 15


Simulation complete. Took 25388.204537 seconds to complete simulation loop

Saving animations...
        Saved electric field animation
        Saved temperature field animation
        Saved mechanical deformation animation
        Saved permittivity animation
        Saved electrical conductivity animation
        Saved density animation
        Saved thermal conductivity animation
        Saved specific heat capacity animation
        Saved magnetic permeability animation
        Deleted individual frame files.
        Done; took 0.196968 seconds to complete movie processing

Simulation completed on Saturday, March 19, 2016 at 15:14:44 EDT.
Took 25399.884567 seconds to complete entire simulation.
```

I.2 One-Dimensional Simulation with Radiative Thermal Boundary Conditions

```
Simulation started Tuesday, April 12, 2016 at 23:02:21 EDT.

Waveguide length is 43.3450764915 cm
Waveguide height is 8.636 cm
Length of insulation + material is 14.4483588305 cm
Height of insulation + material is 4.79777777778 cm
Length of material is 4.81611961017 cm
Height of material is 0.959555555556 cm
Input power is 1.0 kW
Frequency of radiation is 2.45 GHz
Initial temperature is 24.85 degC

Determining optimal activation energy and density function...
       Using densification data from {Teng et al}...
       Attempting data fit to fantozzi sigmoid curve...
       Done; took 1.420201 seconds to find optimal activation energy and MSC.
       Optimal activation energy is 674 kJ/mol.

Interpolating measured data to find dielectric and thermal properties as functions of
    temperature and relative density...
       Assuming parameters are functions of ln(theta)...
       Done; took 6.613446 seconds to find functions for all dielectric and thermal
           material and insulation properties.

Setting up simulation...
       Spatial cell size in air (same in x−dir as in z−dir) is 0.86690152983 cm
       Spatial cell size in insulation (same in x−dir as in z−dir) 0.703149990747 cm
       Spatial cell size in material (same in x−dir as in z−dir) is 0.33174621403 cm
       Total number of nodes in entire domain is 882
       Total number of nodes in insulation + material is 270
       Total number of nodes in material is 64
       Time step for electromagnetic solve is 0.01 sec
       Time step for thermal solve is 0.1 sec
       Total simulated processing time will be 3600 sec

Starting simulation loop...
       Using absorbing boundary condition for electromagnetic solver
```

```
        Using insulating (zero Neumann) boundary condition for thermal solver

At start of simulation...
        Max value of electric field is 0.0 V/m
        Min value of electric field is 0.0 V/m
        Mean value of electric field is 0.0 V/m
        Max temp in insulation is 24.85 degC
        Min temp in insulation is 24.85 degC
        Mean temp in insulation is 24.85 degC
        Max temp in load is 24.85 degC
        Min temp in load is 24.85 degC
        Mean temp in load is 24.85 degC
        Mean density in material is 53.8355714174 percent of bulk density

At time 100.0 sec...
        Max value of electric field is 56823262.0587 V/m
        Min value of electric field is 0.0 V/m
        Mean value of electric field is 1653011.99671 V/m
        Max temp in insulation is 45.89643003 degC
        Min temp in insulation is 24.8500344726 degC
        Mean temp in insulation is 27.6520848789 degC
        Max temp in load is 104.475620704 degC
        Min temp in load is 26.3257436379 degC
        Mean temp in load is 54.7335688463 degC
        Mean density in material is 52.5359627498 percent of bulk density
        Since last printed results, material boundary did not change 0 times
        Since last printed results, material immediately to the right of boundary was
            removed 0 times
        Since last printed results, material immediately to the left of maximum density was
            removed 1000 times
        New material height is 100.0 percent of original height
        Number of nodes remaining in material is 48

At time 200.0 sec...
        Max value of electric field is 56823262.0587 V/m
        Min value of electric field is 0.0 V/m
        Mean value of electric field is 1653011.99671 V/m
        Max temp in insulation is 64.4090262 degC
        Min temp in insulation is 24.8501053187 degC
        Mean temp in insulation is 31.2955456331 degC
        Max temp in load is 119.661734859 degC
        Min temp in load is 27.6577979837 degC
        Mean temp in load is 66.1324535106 degC
        Mean density in material is 52.5359627498 percent of bulk density
        Since last printed results, material boundary did not change 0 times
        Since last printed results, material immediately to the right of boundary was
            removed 0 times
        Since last printed results, material immediately to the left of maximum density was
            removed 1000 times
        New material height is 100.0 percent of original height
        Number of nodes remaining in material is 48

⋮                       ⋮

At time 3700.0 sec...
        Max value of electric field is 56823262.0587 V/m
        Min value of electric field is 0.0 V/m
        Mean value of electric field is 1653011.99671 V/m
        Max temp in insulation is 1375.61476114 degC
        Min temp in insulation is 26.3556702987 degC
        Mean temp in insulation is 336.715879236 degC
        Max temp in load is 454.713464172 degC
        Min temp in load is 59.1643221943 degC
        Mean temp in load is 213.563798802 degC
        Mean density in material is 52.53596275 percent of bulk density
        Since last printed results, material boundary did not change 0 times
        Since last printed results, material immediately to the right of boundary was
            removed 0 times
        Since last printed results, material immediately to the left of maximum density was
            removed 1000 times
        New material height is 100.0 percent of original height
        Number of nodes remaining in material is 48


Simulation complete. Took 8890.500719 seconds to complete simulation loop

Saving animations...
        Saved electric field animation
        Saved temperature field animation
        Saved mechanical deformation animation
        Saved permittivity animation
        Saved electrical conductivity animation
```

```
       Saved density animation
       Saved thermal conductivity animation
       Saved specific heat capacity animation
       Saved magnetic permeability animation
       Deleted individual frame files.
       Done; took 0.174344999999 seconds to complete movie processing

Simulation completed on Tuesday, April 12, 2016 at 23:23:09 EDT.
Took 8902.744219 seconds to complete entire simulation.
```

Bibliography

[1] D. Robinson and S. Friedman, "The effective permittivity of dense packings of glass beads, quartz sand and their mixtures immersed in different dielectric backgrounds," *Journal of Non-Crystalline Solids*, vol. 305, pp. 261–267, 2002.

[2] J. Jin, *The Finite Element Method in Electromagnetics*. New York, NY: John Wiley & Sons, 1993.

[3] M. Hua Teng, Y. Chun Lai, and Y. Tien Chen, "A computer program of Master Sintering Curve model to accurately predict sintering results," *Western Pacific Earth Sciences*, vol. 2, pp. 171–180, May 2002.

[4] T. M. McCoy, *Extension of the Master Sintering Curve for Constant Heating Rate Modeling*. PhD thesis, Georgia Institute of Technology, April 2008.

[5] V. V. Yakovlev, S. M. Allan, M. L. Fall, and H. S. Shulman, "Computational study of thermal runaway in microwave processing of zirconia," in *Microwave and RF Power Applications* (J. Tao, ed.), pp. 303–306, Cépaduès Éditions, 2011.

[6] P. Neelakantaswamy, R. Turkman, and T. Sarkar, "Complex permittivity of a dielectric mixture: Corrected version of Lichtenecker's logarithmic law of mixing," *Electronics Letters*, vol. 21, pp. 270–271, March 1985.

[7] C. Boned and J. Peyrelasse, "Some comments on the complex permittivity of ellipsoids dispersed in continuum media," *Journal of Physics D: Applied Physics*, vol. 16, pp. 1777–1784, 1983.

[8] J. D. Jackson, *Classical Electrodynamics*. New York, NY: John Wiley & Sons, third ed., 1999.

[9] C. C. Lin and L. A. Segel, *Mathematics Applied to Deterministic Problems in the Natural Sciences*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1988.

[10] S. L. Kang, *Sintering: Densification, Grain Growth, and Microstructure*. Oxford, U.K.: Elsevier Butterworth-Heinemann, 2005.

[11] P. Bertrand, F. Bayle, C. Combe, P. Goeuriot, and I. Smurov, "Cermic components manufacturing by selective laser sintering," *Applied Surface Science*, vol. 254, pp. 989–992, 2007.

[12] H. Seitz, W. Rieder, S. Irsen, B. Leukers, and C. Tille, "Three-dimensional printing of porous ceramic scaffolds for bone tissue engineering," *Journal of Biomedical Materials Research Part B: Applied Biomaterials*, vol. 74B, pp. 782–788, August 2005.

[13] T. Traini, C. Mangano, R. Sammons, F. Mangano, A. Macchi, and A. Piattelli, "Direct laser metal sintering as a new approach to fabrication of an isoelastic functionally graded material for manufacture of porous titanium dental implants," *Dental Materials*, vol. 24, pp. 1525–1533, March 2008.

[14] Y. Zhao, T. Fung, L. Zhang, and F. Zhang, "Lost carbonate sintering process for manufacturing metal foams," *Scripta Materialia*, vol. 52, pp. 295–298, 2005.

[15] M. Vasquez, *Analysis and Development of New Materials for Polymer Laser Sintering*. PhD thesis, Loughborough University, United Kingdom, July 2012.

[16] T. Tsutsui, "Recent technology of powder metallurgy and applications," Tech. Rep. 54, Hitachi Chemical Technical Report, Japan, March 2012. `http://www.hitachi-chem.co.jp/english/report/054/54_sou2.pdf`.

[17] S. I. Kim, K. H. Lee, H. A. Mun, H. S. Kim, S. W. Hwang, J. W. Roh, D. J. Yang, W. H. Shin, X. S. Li, Y. H. Lee, G. J. Snyder, and S. W. Kim, "Dense dislocation arrays embedded in grain boundaries for high-performance bulk thermoelectrics," *Science*, vol. 348, no. 6230, pp. 109–114, 2015.

[18] D. Agrawal, "Microwave sintering of ceramics, composites and metallic materials, and melting of glasses," *Transactions of the Indian Ceramic Society*, vol. 65, no. 3, pp. 129–144, 2006.

[19] J. Muñoz Hoyos, F. Zabotto, D. Garcia, and R. Kiminami, "Sinterização por micro-ondas de ferrita de níquel sintetizada pelo método Pechini," *Cerâmica*, pp. 360–365, 2013.

[20] A. Upadhyaya, S. Tiwari, and P. Mishra, "Microwave sintering of W–Ni–Fe alloy," *Scripta Materialia*, vol. 56, pp. 5–8, 2007.

[21] M. Oghbaei and O. Mirzaee, "Microwave versus conventional sintering: A review of fundamentals, advantages and applications," *Journal of Alloys and Compounds*, vol. 494, pp. 175–189, January 2010.

[22] M. Gupta and E. W. W. Leong, *Microwaves and Metals*. Singapore: John Wiley & Sons (Asia) Pte Ltd, 2007.

[23] K. Saitou, "Microwave sintering of iron, cobalt, nickel, copper and stainless steel powders," *Scripta Materialia*, vol. 54, pp. 875–879, March 2006.

[24] D. Bouvard, "Finite element simulation of sintering." Presented at the 12th Seminar "Computer Modeling in Microwave Engineering and Applications". Grenoble, France, March 8, 2010.

[25] D. Agrawal, "Microwave sintering developments spur emergence of new materials and technologies," *Industrial Heating*, vol. 72, pp. 37–39, June 2005.

[26] H. S. Shulman, D. W. J. Walker, T. A. Treado, M. Marks, M. Fall, S. J. Evans, and M. L. Tracy, "Sintering uniformity and reproducibility with 2.45 GHz microwaves in an industrial sized chamber," in *Innovative Processing and Synthesis of Ceramics, Glasses, and Composites VI*, pp. 13–25, John Wiley & Sons, Inc., 2006.

[27] A. Amri and A. Saidane, "TLM simulation of microwave sintering of ceramics using SiC stimulus," *Journal of Microwave Power and Electromagnetic Energy*, vol. 36, no. 2, pp. 89–100, 2001.

[28] H. Riedel and J. Svoboda, "Simulation of microwave sintering with advanced sintering models," in *Advances in Microwave and Radio Frequency Processing* (M. Willert-Porada, ed.), pp. 210–216, Berlin: Springer, 2006.

[29] D. Bouvard, S. Charmond, and C. P. Carry, "Finite element modeling of microwave sintering," in *Advances in Sintering Science and Technology: Ceramic Transactions* (R. Bordia and E. A. Olevsky, eds.), pp. 173–180, John Wiley & Sons, 2010.

[30] E. A. Olevsky, "Theory of sintering: from discrete to continuum," *Materials Science and Engineering*, vol. R23, pp. 41–100, 1998.

[31] R. Shen, L. Kemper, L. Zong, M. Hawley, and A. Bernard, "Coupled electromagnetic thermal and kinetic modeling for microwave processing of polymers with temperature- and cure-dependent permittivity using 3D FEM," *International Journal of Applied Electromagnetism and Mechanics*, vol. 30, pp. 9–28, 2009.

[32] E. A. Olevsky, A. L. Maximenko, and E. G. Grigoryev, "Ponderomotive effects during contact formation in microwave sintering," *Modelling and Simulation in Materials Science and Engineering*, vol. 21, no. 055022, 2013.

[33] H. Riedel and R. Blug, "A comprehensive model for solid state sintering and its application to silicon carbide," *Multiscale Deformation and Fracture in Materials and Structures*, pp. 49–70, 2000.

[34] J. C. Maxwell, *A Treatise on Electricity and Magnetism*. New York: Dover Publications, Inc., 1954.

[35] K. F. Gauß, "Zur mathematischen Theorie der elektrodynamischen Wirkung (1835)," *Werke*, vol. V.

[36] C. Jungnickel and R. McCormmach, *Intellectual Mastery of Nature: Theoretical Physics from Ohm to Einstein*. Chicago, IL: University of Chicago Press, 1990.

[37] A.-M. Ampère, *Mémoire sur la théorie mathématique des phénomènes électrodynamiques : uniquement déduite de l'expérience, dans lequel se trouvent réunis les mémoires que M. Ampère a communiqués à l'Académie royale des Sciences, dans les séances des 4 et 26 décembre 1820, 10 juni 1822, 22 décembre 1823, 12 septembre 1825, et 21 novembre 1825*. 1825. Available online: `http://dx.doi.org/10.3931/e-rara-3579`.

[38] M. Faraday, *Experimental Researches in Electricity*, vol. 1. Red Lion Court, Fleet Street, London: Richard and John Edward Taylor, Printers, 2 ed., 1849. Available online: `http://gallica.bnf.fr/ark:/12148/bpt6k94883h`.

[39] M. Faraday, *Experimental Researches in Electricity*, vol. 2. Red Lion Court, Fleet Street, London: Richard and John Edward Taylor, Printers and Publishers to the University of London, 1844. Available online: `http://docs.lib.noaa.gov/rescue/Rarebook_treasures/QC503F211839_PDF/QC503F211839v2.pdf`.

[40] M. Faraday, *Experimental Researches in Electricity*, vol. 3. Red Lion Court, Fleet Street, London: Richard and John Edward Taylor, Printers and Publishers to the University of London, 1855. Available online: `http://gallica.bnf.fr/ark:/12148/bpt6k948856/`.

[41] O. Heaviside, *Electromagnetic Theory*, vol. 1. 1–3 Salisbury Court, Fleet Street, London: "The Electrician" , Printing and Publishing Ltd., 1893. Available online: `https://archive.org/stream/electromagnetict01heavrich`.

[42] O. Heaviside, *Electromagnetic Theory*, vol. 2. 1–3 Salisbury Court, Fleet Street, London: "The Electrician" , Printing and Publishing Ltd., 1899. Available online: `https://archive.org/details/electromagnetict02heavrich`.

[43] O. Heaviside, *Electromagnetic Theory*, vol. 3. 1–3 Salisbury Court, Fleet Street, London: "The Electrician" , Printing and Publishing Ltd., 1912. Available online: `https://archive.org/stream/electromagnetict03heavuoft`.

[44] G. B. Arfken and H. J. Weber, *Mathematical Methods for Physicists*. London, U.K.: Elsevier Academic Press, sixth ed., 2005.

[45] E. F. Kuester and D. C. Chang, *Electromagnetic Boundary Problems*. London: CRC Press, Taylor & Francis Group, 2015.

[46] B. Banerjee, *An Introduction to Metamaterials and Waves in Composites*. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2011.

[47] E. M. Purcell and D. J. Morin, *Electricity and Magnetism*. New York, NY: Cambridge University Press, third ed., 2013.

[48] D. A. Fleisch, *A Student's Guide to Maxwell's Equations*. New York, NY: Cambridge University Press, 1 ed., 2008.

[49] H. C. Øersted, *Experimenta circa effectum conflictus electrici in acum magneticam*. 1920. Accessed Online: `https://ia800309.us.archive.org/14/items/Experimentacirc00Orst/Experimentacirc00Orst.pdf`.

[50] F. G. Rodrigues, "Formulações equivalentes de la lei de Faraday," *Revista Brasileira de Ensino de Física*, vol. 34, March 2012.

[51] N. Rosen and D. Schieber, "Some remarks on Faraday's law," *American Journal of Physics*, vol. 50, p. 974, 1982.

[52] E. Thostenson and T. Chou, "Microwave processing: fundamentals and applications," *Composites Part A: Applied Science and Manufacturing*, vol. 30, pp. 1055–1071, September 1999.

[53] R. R. Menezes, P. M. Souto, and R. H. G. A. Kiminami, "Microwave fast sintering of ceramic materials," in *Sintering of Ceramics – New Emerging Techniques* (A. Lakshmanan, ed.), ch. 1, pp. 3–26, InTech Open Science, 2012.

[54] M. Moeller, H. S. Shulman, and H. Giesche, "A novel approach to understanding microwave heating of zirconia," *Ceramic Transactions*, vol. 135, pp. 27–37, 2002.

[55] Y. Fang, L. Li, Q. Xiao, and X. M. Chen, "Preparation and microwave dielectric properties of cristobalite ceramics," *Ceramics International*, vol. 38, pp. 4511–4515, August 2012.

[56] I. Akin and G. Goller, *Handbook of Bioceramics and Biocomposites*, ch. Processing Technologies for Bioceramic Based Composites, pp. 1–22. Cham: Springer International Publishing, 2014.

[57] Z. J. Wang, H. Kokawa, H. Takizawa, M. Ichiki, and R. Maeda, "Low-temperature growth of high-quality lead zirconate titanate thin films by 28 GHz microwave irradiation," *Applied Physics Letters*, vol. 86, no. 212903, 2005.

[58] M. A. Janney, C. L. Calhoun, and H. D. Kimrey, "Microwave sintering of solid oxide fuel cell materials: I, Zirconia-8 mol% Yttria," *Journal of the American Ceramic Society*, vol. 75, pp. 341–346, February 1992.

[59] J. D. Katz, R. D. Blake, and J. J. Petrovic, "Microwave sintering of alumina-silicon carbide composites at 2.45 and 60 GHz," in *Proceedings of the Ceramics Engineering Society*, vol. 9, pp. 725–734, 1988.

[60] H. Flanders, "Differentiation under the integral sign," *The American Mathematical Monthly*, vol. 80, pp. 615–627, June–July 1973.

[61] R. E. Collin, *Field Theory of Guided Waves*. New York, NY: IEEE Press, 1991.

[62] D. M. Pozar, *Microwave Engineering*. New York, NY: John Wiley & Sons, 2005.

[63] A. Kirsch and F. Hettlich, "The Mathematical Theory of Maxwell's Equations." E-book, published online: `http://www.math.kit.edu/iag1/lehre/maxwellequ2012w/media/main.pdf`, January 14, 2013.

[64] M. Lanzagorta, *Underwater Communications*, vol. (print): 1932–1244, (electronic): 1932–1708 of *Synthesis Lectures on Communications*. San Rafael, CA, USA: Morgan & Claypool Publishers, 2013.

[65] K. T. McDonald, *Electrodynamics in 1 and 2 Spatial Dimensions*. Princeton University, Princeton, NJ, September 2014. Available online: `http://physics.princeton.edu/~mcdonald/examples/2dem.pdf`.

[66] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York, NY, USA: John Wiley & Sons, Inc., second ed., 2012.

[67] Y. V. Shestopalov, "Forward and inverse scattering in waveguides: Fundamentals of mathematical theory." Presentation at 14$^{\text{th}}$ Seminar "Computer Modeling in Microwave Power Engineering: Methods and Models for Microwave Procesing of Materials", Bled, Slovenia, March 2015.

[68] R. B. Guenther and J. W. Lee, *Partial Differential Equations of Mathematical Physics and Integral Equations*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1988.

[69] R. Haberman, *Applied Partial Differential Equations: with Fourier Series and Boundary Value Problems*. Prentice Hall, fourth ed., 2003.

[70] E. Sparrow, J. Dallman, and R. Ramazani, "Essay 3: Fourier's Law and the First Law of Thermodynamics." Available online: `http://www.me.umn.edu/courses/old_me_course_pages/me3333/essays/essay%203.pdf`, June 15, 2010.

[71] T. Myint-U and L. Debnath, *Linear Partial Differential Equations for Scientists and Engineers*. Boston, MA, USA: Birkhäuser, fourth ed., 2007.

[72] S. Perino-Issartier, J.-F. Maingonnat, and F. Chemat, "Microwave food processing," in *Alternatives to Conventional Food Processing* (A. Proctor, ed.), no. 10 in RSC Green Chemistry, ch. 11, London, UK: RSC Publishing, 2011.

[73] U. Besson, "The history of the cooling law: When the search for simplicity can be an obstacle," *Science & Education*, vol. 21, pp. 1085–1110, August 2012.

[74] H. Su and D. L. Johnson, "Master sintering curve: A practical approach to sintering," *Journal of the American Ceramic Society*, vol. 79, no. 12, pp. 3211–3217, 1996.

[75] M. N. Rahaman, *Sintering of Ceramics*. Boca Raton, FL: CRC Press, Taylor and Francis Group, 2008.

[76] B. Burton, "Interface reaction-controlled diffusional creep: a consideration of grain boundary dislocation climb sources," *Materials Science Engineering*, vol. 10, no. 9–14, 1972.

[77] A. C. F. Cocks, "Interface reaction controlled creep," *Mechanics of Materials*, vol. 13, pp. 165–174, 1992.

[78] S. D. Peteves and R. Abbaschian, "Growth kinetics of solid-liquid Ga interfaces: Part I, Experimental," *Metallurgical and Materials Transactions A*, vol. 27A, pp. 2809–2819, 1996.

[79] H. Riedel, H. Zipse, and J. Svoboda, "Equilibrium pore surfaces, sintering stresses and constitutive equations for the intermediate and late stages of sintering—Part II: diffusional densification and creep," *Acta Metallurgica et Materialia*, no. 42, 1994.

[80] H. Riedel, V. Kozák, and J. Svoboda, "Equilibrium pore surfaces, sintering stresses and constitutive equations for the intermediate and late stages of sintering," *Acta Metallurgica*, no. 42, pp. 3093–3103, 1994.

[81] H. Riedel, J. Svoboda, and H. Zipse, "Numerical simulation of die pressing and sintering—development of constitutive equations," pp. 663–671, Powder Metallurgy World Congress PM94, 1994.

[82] T. Kraft and H. Riedel, "Numerical simulation of solid state sintering: model and application," *Journal of the European Ceramic Society*, vol. 24, pp. 345–361, 2004.

[83] J. Svoboda and H. Riedel, "A theoretical study of grain coarsening in porous solids," *Acta Metallurgica et Materialia*, vol. 41, pp. 1929–1936, 1993.

[84] R. McMeeking and L. Kuhn, "A diffusional creep law for powder compacts," *Acta Metallurgica*, no. 40, pp. 961–969, 1992.

[85] M. Hillert, "On the theory of normal and abnormal grain growth," *Acta Metallurgica*, vol. 13, pp. 227–239, 1965.

[86] J. D. Hansen, R. P. Rusin, M.-H. Teng, and D. L. Johnson, "Combined stage sintering model," *Journal of the American Ceramic Society*, vol. 75, no. 5, pp. 1129–1135, 1992.

[87] D. L. Johnson, "Comment on 'Temperature-gradient-driven diffusion in rapid-rate sintering,'" *Journal of the American Ceramic Society*, vol. 73, no. 8, pp. 2576–2578, 1990.

[88] D. L. Johnson, R. P. Rusin, and J. D. Hansen, "Application of a new sintering model," *Sintering: Advances in Powder Metallurgy and Particulate Materials*, vol. 3, pp. 17–24, 1992. Metal Powder Industries Foundation, Princeton, NJ.

[89] S. J. Park, P. Suri, E. Olevsky, and R. M. German, "Master sintering curve formulated from constitutive models," *Journal of the American Ceramic Society*, vol. 92, no. 7, pp. 1410–1413, 2009.

[90] B. S. Tilley and G. A. Kriegsmann, "Microwave-enhanced chemical vapor infiltration: a sharp interface model," *Journal of Engineering Mathematics*, vol. 41, pp. 33–54, April 2001.

[91] K. Lichtenecker, "Die Dielektrizitätskonstante natürlicher und künstlicher Mischkörper," *Zeitschrift für Physik*, vol. XXVII, pp. 115–158, 1926.

[92] K. Lichtenecker and K. Rother, "Die Herleitung des logarithmischen Mischungs-gesetzes aus allegemeinen Prinzipien der stationaren Stromung," *Zeitschrift für Physik*, vol. XXXII, pp. 255–260, 1931.

[93] H. Ebara, T. Inoue, and O. Hashimoto, "Measurement method of complex permittivity and permeability for a powdered material using a waveguide in microwave band," *Science and Technology of Advanced Materials*, vol. 7, pp. 77–83, 2006.

[94] H. Z. M. Shafri, R. R. Abdullah, M. Roslee, and R. Muniandy, "Optimization of ground penetrating radar (GPR) mixture model in road pavement density analysis," pp. 1326–1329, IGARSS, 2008.

[95] M. Olszowy, "Dielectric and pyroelectric properties of the composites of ferroelectric ceramic and poly(vinyl chloride)," *Condensed Matter Physics*, vol. 6, no. 2(34), pp. 307–313, 2003.

[96] H. Saito, Y. Suzuki, and M. Taki, "Measurement of complex permittivity for biological cells at 1.7-2.6GHz by waveguide penetration method," URSI, August 7-16 2008.

[97] T. Zakri, J.-P. Laurent, and M. Vauclin, "Theoretical evidence for Lichtenecker's mixture formulae based on the effective medium theory," *Journal of Physics D: Applied Physics*, vol. 31, pp. 1589–1594, 1998.

[98] A. V. Goncharenko, V. Z. Lozovski, and E. F. Venger, "Lichtenecker's equation: applicability and limitations," *Optics Communications*, vol. 174, no. 1-4, pp. 19–32, 2000.

[99] D. J. Bergman, "The dielectric constant of a composite material: A problem in classical physics," *Physics Reports*, vol. 43, p. 377, July 1978.

[100] R. Simpkin, "Derivation of Lichtenecker's logarithmic mixture formula from Maxwell's equations," *IEEE Transactions on Microwave Theory and Techniques*, vol. 58, pp. 545–550, March 2010.

[101] J. Reynolds and J. Hough, "Formulae for dielectric constant of mixtures," *Proceedings of the Physical Society, London, Section B*, vol. 70, no. 8, 1957.

[102] S. S. Dukhin and V. N. Shilov, *Dielectric phenomena and the double layer in disperse systems and polyelectrolytes*. 1974.

[103] O. Wiener, "Zur Theorie der Refraktionskonstanten," *Berichte über die Verhandlunger der Königlich-Sächsischen Gesellschaft der Wisenschaften zu Leipzig*, vol. 62, pp. 256–277, 1910.

[104] S. Kisdnasamy and P. Neelakantaswamy, "Complex permittivity of a dielectric mixture: Modified Fricke's formula based on logarithmic law of mixing," *Electronic Letters*, vol. 20, pp. 291–293, 29 March 1984.

[105] R. Coelho, *Physics of Dielectrics for the Engineer*. Amsterdam: Elsevier Scientific, 1979.

[106] Lord Rayleigh (John William Strutt), "On the influence of obstacles arranged in rectangular order upon the properties of a medium," *Philosophical Magazine*, vol. 34, no. 211, pp. 481–502, 1892.

[107] H. A. Lorentz, "Ueber die Beziehung zwischen der Fortpflanzungsgeschwindigkeit des Lichtes und der Körperdichte," *Annalen der Physik und Chemie*, vol. 245, pp. 641–665, 1880.

[108] L. Lorenz, "Ueber die Refractionsconstante," *Annalen der Physik und Chemie*, vol. 247, pp. 70–103, 1880.

[109] L. Campbell and W. Garnett, *The Life of James Clerk Maxwell*. London: Macmillan and Co., 1882. Available on Google Books.

[110] J. C. Maxwell Garnett, "Colours in metal glasses and metallic films," *Philosophical Transactions of the Royal Society of London*, vol. 73, no. 203, pp. 385–420, 1904.

[111] J. C. Maxwell Garnett, "Colours in metal glasses and metallic films II," *Philosophical Transactions of the Royal Society of London*, vol. 76, pp. 370–373, August 4 1905.

[112] M. Koledintseva, R. DuBroff, and R. Schwartz, "A Maxwell-Garnett model for dielectric mixtures containing conducting particles at optical frequencies," *Progress in Electromagnetics Research*, vol. 63, pp. 223–242, 2006.

[113] A. H. Sihvola, E. Nyfors, and M. Tiuri, "Mixing formulae and experimental results for the dielectric constant of snow," *Journal of Glaciology*, vol. 31, no. 108, pp. 163–170, 1985.

[114] J.-M. Thériault and G. Boivin, "Maxwell-Garnett theory extended for Cu-PbI$_2$ cermets," *Applied Optics*, vol. 23, no. 24, pp. 4494–4498, 1984.

[115] M. Koledintseva, P. Ravva, R. DuBroff, J. Drewniak, K. Rozanov, and B. Archambault, "Engineering of composite media for shields at microwave frequencies," vol. 1, (Chicago, Illinois), pp. 169–174, IEEE EMC Symposium, August 2005.

[116] D. A. G. Bruggeman, "Calculation of various physical constants of heterogeneous substances," *Annals of Physics*, vol. 32, no. 12, pp. 636–664, 1935.

[117] B. Tjaden, S. J. Cooper, D. J. L. Brett, D. Kramer, and P. R. Shearing, "On the origin and application of the Bruggeman correlation for analysing transport phenomena in electrochemical systems," *Current Opinion in Chemical Engineering*, vol. 12, pp. 44–51, May 2016.

[118] K. I. Rybakov, E. A. Olevsky, and E. V. Krikun, "Microwave sintering: Fundamentals and modeling," *Journal of the American Ceramic Society*, vol. 96, no. 4, pp. 1003 – 1020, 2013.

[119] E. M. Kiley, V. V. Yakovlev, K. Ishizaki, and S. Vaucher, *Microwave and RF Power Applications*, ch. Applicability Study of Classical and Contemporary Models for Effective Complex Permittivity of Metal Powders, pp. 314–317. Toulouse, France: Cépaduès Éditions, 2012.

[120] D. Bouvard, S. Charmond, and C. P. Carry, "Multiphysics simulation of microwave sintering in a monomode cavity," pp. 21–26, 12th Seminar in Advances in Modeling of Microwave Sintering: Computer Modeling in Microwave Engineering and Applications, March 8–9 2010.

[121] D. Zimmerman, J. Cardellino, K. Cravener, K. Feather, N. Miskovsky, and G. Weisel, "Microwave absorption in percolating metal-insulator composites," *Applied Physics Letters*, vol. 93, no. 214103, pp. 1–3, 2008.

[122] V. D. Buchelnikov, D. V. Louzgine-Luzgin, N. Yoshikawa, M. Sato, A. P. Anzulevich, I. V. Bychkov, and A. Inoue, "Modeling of microwave heating of metallic powders," (Otsu, Japan), pp. 251–254, 1st Global Congress on Microwave Energy Applications, August 2008.

[123] V. D. Buchelnikov, D. V. Louzgine-Luzgin, G. Xie, S. Li, N. Yoshikawa, M. Sato, A. P. Anzulevich, I. V. Bychkov, and A. Inoue, "Heating of metallic powders by microwaves: Experiment and theory," *Journal of Applied Physics*, vol. 104, no. 113505, 2008.

[124] V. D. Buchelnikov, D. V. Louzgine-Luzgin, A. P. Anzulevich, I. V. Bychkov, and N. Yoshikawa, "Modeling of microwave heating of metallic powders," *Physica B: Condensed Matter*, vol. 403, pp. 4053–4058, 2008.

[125] M. Ignatenko, M. Tanaka, and M. Sato, "Numerical analysis of microwave heating of copper powders," pp. 206–209, AMPERE 12th International Conference on Microwave and High Frequency Heating, September 2009.

[126] K. K. Kärkkäinen, A. H. Sihvola, and K. I. Nikoskinen, "Effective permittivity of mixtures: Numerical validation by the FDTD method," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 38, pp. 1303–1308, May 2000.

[127] Z. Hashin and S. Shtrikman, "A variational approach to the theory of the effective magnetic permeability of multiphase materials," *Journal of Applied Physics*, vol. 33, no. 10, pp. 3125–3131, 1962.

[128] A. H. Sihvola, "Self-consistency aspects of dielectric mixing theories," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 27, no. 4, pp. 403–415, 1989.

[129] J. Behari, *Microwave Dielectric Behavior of Wet Soils*. Dordrecht, the Netherlands: Springer, 2005.

[130] C. J. F. Böttcher, *Theory of Electric Polarization: Dielectrics in Static Fields*. Amsterdam: Elsevier, 1952.

[131] L. Tsang, J. A. Kong, and R. T. Shin, *Theory of Microwave Remote Sensing (Wiley Series in Remote Sensing)*. Wiley-Interscience, 2 ed., 1985.

[132] W. Stiles and F. Ulaby, "Dielectric properties of snow," *NASA Technical Reports*, vol. NASA-CR-166764, December 1 1981.

[133] J. Sheena, Z.-W. Honga, W. Liua, W.-L. Maoa, and C.-A. Chenb, "Study of dielectric constants of binary composites at microwave frequency by mixture laws derived from three basic particle shapes," *European Polymer Journal*, vol. 45, pp. 1316–1321, April 2009.

[134] R. Waldron, *Perturbation Theory of Resonant Cavities*, pp. 272–274. No. 373 E, IEE Press, 1960.

[135] B. Meng, J. Bookse, and R. Cooper, "Extended cavity perturbation technique to determine the complex permittivity of dielectric materials," *IEEE Transactions on Microwave Theory and Techniques*, vol. 43, no. 11, pp. 2633–2636, 1995.

[136] S. Li, C. Aykel, and R. Boisio, "Precise calculation and measurements on the complex dielectric constant of lossy materials using TM_{010} cavity perturbation techniques," *IEEE Transactions on Microwave Theory and Techniques*, vol. 29, no. 10, pp. 1041–1047, 1981.

[137] P. Veronesi, C. Leonelli, G. Pellacani, and A. Boccaccini, "Unique microstructure of glass-metal composites obtained by microwave-assisted heat treatments," *Journal of Thermal Analysis and Calorimetry*, vol. 72, pp. 1141–1149, 2003.

[138] E. E. Eves, E. K. Murphy, and V. V. Yakovlev, "Practical aspects of complex permittivity reconstruction with neural-network-controlled FDTD modeling of a two-port fixture," *Journal of Microwave Power and Electromagnetic Energy*, vol. 41, no. 4, pp. 81–94, 2007.

[139] M. Tobar, J. Krupka, E. Ivanov, and R. Woode, "Anisotropic complex permittivity measurements of monocrystalline rutile between 10 and 300 K," *Journal of Materials Science: Material Electronics*, vol. 21, no. 8, pp. 817–821, 2010.

[140] S. Ruben, *Handbook of the Elements*, p. 95. La Salle, IL, USA: Open Court, 1998.

[141] K. Nose, H. Oba, and T. Yoshida, "Electric conductivity of boron nitride thin films ehanced by *in situ* doping of zinc," *Applied Physics Letters*, vol. 89, pp. 122–124, 2006.

[142] J.-M. Jin, *Theory and Computation of Eletromagnetic Fields*. Hoboken, NJ: IEEE Press, John Wiley and Sons, 2010.

[143] L. G. Hector and H. L. Schultz, "The dielectric constant of air at radiofrequencies," *Physics*, vol. 7, pp. 133—136, 1936.

[144] Agilent Technologies, "Genesys 2008 help guide: Substrate parameter tables," 1994–2008. Accessed Online: `http://cp.literature.agilent.com/litweb/pdf/genesys200801/reference.htm\#elements/substrate\_tables/tablelosstan.htm`.

[145] G. Kaye and T. Laby, "Tables of physical and chemical constants," 1911–2004. Accessed Online: `http://www.kayelaby.npl.co.uk/general_physics/2_6/2_6_5.html`.

[146] D. Engwirda, "MESH2D: Automatic mesh generation," 2009. Accessed Online: `http://www.mathworks.com/matlabcentral/fileexchange/25555-mesh2d-automatic-mesh-generation`.

[147] A. Quarteroni, *Numerical Models for Differential Problems*. Modeling, Simulations, & Applications, Milan: Springer-Verlag Italia, 2009.

[148] J. C. Strikwerda, *Finite Difference Schemes and Partial Differential Equations*. Philadelphia, PA, USA: SIAM: Society for Industrial and Applied Mathematics, second edition ed., 2004.

[149] F. Wakai, "Modeling and simulation of elementary processes in ideal sintering," *Journal of the American Ceramic Society*, vol. 89, no. 5, pp. 1471–1484, 2006.

[150] A. C. De Bellis, "Computer modeling of sintering in ceramics," Master's thesis, University of Pittsburgh, 2002.

[151] C. Martin, L. Schneider, L. Olmos, and D. Bouvard, "Discrete element modeling of metallic powder sintering," *Scripta Materialia*, vol. 55, pp. 425–428, 2006.

[152] D. Bouvard and R. McMeeking, "Deformation of interparticle necks by diffusion-controlled creep," *Journal of the American Ceramic Society*, vol. 79, pp. 666–672, March 1996.

[153] F. Parhami and R. McMeeking, "A network model for initial stage sintering," *Mechanics of Materials*, vol. 27, pp. 111–124, 1998.

[154] A. Jagota, P. Dawson, and J. Jenkins, "An anistotropic continuum model for the sintering and compaction of powder packings," *Mechanics of Materials*, vol. 7, pp. 255–269, 1988.

[155] R. Zhang, *Numerical Simulation of Solid State Sintering of Metal Powder Compact Dominated by Grain Boundary Diffusion*. PhD thesis, Pennsylvania State University, December 2005.

[156] H. Kim, O. Gillia, and D. Bouvard, "A phenomenological constitutive model for the sintering of alumina powder," *Journal of the European Ceramic Society*, vol. 23, pp. 1675–1685, 2003.

[157] H. Kim, O. Gillia, P. Dorémus, and D. Bouvard, "Near net shape processing of a sintered alumina component: adjustment of pressing parameters through finite element simulation," *International Journal of Mechanical Sciences*, vol. 44, pp. 2523–2539, 2002.

[158] A. Rosin and M. Willert-Porada, "Modeling of microwave heating of ceramic materials with respect to changes in sample size," in *Proceedings of the 16th Seminar "Computer Modeling in Microwave Power Engineering": Multiphysics Models and Material Properties* (V. V. Yakovlev and E. M. Kiley, eds.), (Karlsruhe, Germany), May 2014.

[159] T. Gupta, M. J. Akhtar, and A. Biswas, "A unit cell approach to the model and characterize the metal powders and metal-dielectric composistes at microwave frequencies," *Progress in Electromagnetics Research B*, vol. 49, pp. 363–387, 2013.

[160] D. Agrawal, "Microwave sintering of ceramics, composites and metallic materials, and melting of glasses," *Transactions of the Indian Ceramics Society: Topical Reviews*, vol. 65, no. 3, pp. 129–144, 2006.

[161] Y. Duan, D. C. Sorescu, and J. K. Johnson, "Finite element approach to microwave sintering of oxide materials," Proceedings of the COMSOL Users Conference, 2006.

[162] B. Zhang and M. M. Gasik, "Stress evolution in graded materials during densification by sintering process," *Computational Materials Science*, vol. 25, pp. 264–271, 2002.

[163] M. Gasik and B. Zhang, "A constitutive model and fe simulation for the sintering process of powder compacts," *Computational Materials Science*, vol. 18, pp. 93–101, 2000.

[164] A. P. Roday and P. J. Nicosia, "Simulation of PTFE billet sintering using COMSOL," in *Proceedings of the 2011 COMSOL Conference*, (Boston, MA, USA), 2011.

[165] M. Reiterer, T. Kraft, and H. Reidel, "Manufacturing of a gear wheel made from reaction bonded alumina–numerical simulation of the sinterforming process," *Journal of the European Ceramic Society*, vol. 4, pp. 239–246, 2004.

[166] G. Fantozzi, S. Le Gallet, and J. Claude Nièpce, *Science et Technologies Céramiques*. EDP, GFC, 2010.

[167] J. Croquesel, *Étude des spécificités du frittage par micro-ondes de poudres d'alumine α et γ*. PhD thesis, Université de Grenoble, 2015. Accessible Online: `https://tel.archives-ouvertes.fr/tel-01179586/file/CROQUESEL_2015_archivage.pdf`.

[168] D. Żymełka, *Suivi par methode optique du frittage micro-ondes d'oxydes ceramiques*. PhD thesis, École Nationale Supérieure des Mines de Saint-Étienne, 2012. Accessible Online: `https://tel.archives-ouvertes.fr/tel-00821161/document`.

[169] D. C. Blaine, S.-J. Park, and R. M. German, "Linearization of master sintering curve," *Journal of the American Ceramic Society*, vol. 92, no. 7, pp. 1403–1409, 2009.

[170] D. C. Blaine, S. Jin Park, and R. M. German, "Master sintering curve for a two-phase material," in *Presented at the Fourth International Conference on Science, Technology and Applications of Sintering*, (Grenoble, France), 2005.

[171] D. C. Blaine, J. D. Gurosik, S. Jin Park, D. F. Heaney, Jr., and R. M. German, "Master Sintering Curve concepts as applied to the sintering of Molybdenum," *Metallurgical and Materials Transactions A*, vol. 37A, pp. 715–720, March 2006.

[172] D. C. Blaine, S. Jin Park, P. Suri, and R. M. German, "Application of Work-of-Sintering concepts in powder metals," *Metallurgical Materials Transactions A*, vol. 37A, no. 9, pp. 2827–2835, 2006.

[173] J. E. Dennis, Jr. and R. B. Schnabelf, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, vol. 16 of *Classics in Applied Mathematics*. Philadelphia, PA, USA: SIAM, 1996.

[174] C. T. Kelley, *Iterative Methods for Optimization*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999.

[175] W. Gautschi and W. F. Cahill, "Exponential integral and related functions," in *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables* (M. Abramowitz and I. Stegun, eds.), no. 55 in National Bureau of Standards Applied Mathematics Series, ch. 5, pp. 227–252, Washington, DC: United States Department of Commerce, 1972. Tenth printing.

[176] D. M. Smith, "Algorithm 911: Multiple-precision exponential integral and related functions," *ACM Transactions on Mathematical Software*, vol. 37, pp. 46:1–46:16, February 2011.

[177] R. E. Lyon, "An integral method of non-isothermal kinetic analysis," Final Report DOT/FAA/AR-96/68, U.S. Department of Transportation, Federal Aviation Administration, Fire Research Section, AAR-423, William J. Hughes Technical Center, Atlantic City, NJ, July 1996.

[178] J. R. Biegen and A. W. Czanderna, "Analysis of thermal processes: The exponential integral," *Journal of Thermal Analysis*, vol. 4, pp. 39–45, June 1972.

[179] V. M. Gorbachev, "Algorism for the solution of the exponential integral in non-isothermal kinetics at linear heating," *Journal of Thermal Analysis*, vol. 10, pp. 447–449, March 1976.

[180] V. M. Gorbachev, "A solution of the exponential integral in the non-isothermal kinetics for linear heating," *Journal of Thermal Analysis and Calorimetry*, vol. 8, no. 2, p. 349, 1975.

[181] J. Zsakó and J. Zsakó Jr., "Kinetic analysis of thermogravimetric data," *Journal of Thermal Analysis*, vol. 19, pp. 333–345, January 1980.

[182] R. Chen, "The computation of the exponential integral as related to the analysis of thermal processes," *Journal of Thermal Analysis*, vol. 6, pp. 585–586, March 1974.

[183] W. Yu, X. Yang, Y. Liu, R. Mittra, and A. Muto, *Advanced FDTD Methods: Parallelization, Acceleration, and Engineering Applications*. Boston - London: Artech House, 2011.

[184] U. S. Inan and R. A. Marshall, *Numerical Electromagnetics: The FDTD Method*. Cambridge: Cambridge University Press, 1 ed., 2011.

[185] R. E. Collin, *Foundations for Microwave Engineering*. New York: IEEE Press, 2 ed., 2001.

[186] J. Park, *Bioceramics: Properties, Characterizations, and Applications*. New York, NY, USA: Springer Science+Business Media, 2008.

[187] J. Wang and R. Raj, "Estimate of the activation energies for boundary diffusion from rate-controlled sintering of pure alumina, and alumina doped with zirconia or titania," *Journal of the American Ceramic Society*, vol. 73, no. 5, pp. 1172–1175, 1990.

[188] A. J. Rayner, R. M. C. Clemmer, and S. F. Corbin, "Determination of the activation energy and master sintering curve for NiO/YSZ composite solid oxide fuel cell anodes," *Journal of the American Ceramic Society*, vol. 98, no. 4, pp. 1060–1065, 2015.

[189] G. Bernard-Granger and C. Guizard, "Apparent activation energy for the densification of a commercially available granulated zirconia powder," *Journal of the American Ceramic Society*, vol. 90, no. 4, pp. 1246–1250, 2007.

[190] G. Fadda, L. Trusnikovsky, and G. Zanzotto, "Unified Landau description of the tetragonal, orthorhombic, and monoclinic phases of zirconia," *Physical Review B*, vol. 66, no. 147107, 2002.

[191] T. D. Isfahani, J. Javadpour, A. Khavandi, and M. Goodarzi, "Nanocrystalline growth activation energy of Zirconia polymorphs synthesized by micromechanical technique," *Journal of Materials Science and Technology*, vol. 30, no. 4, pp. 387–393, 2014.

[192] S. Heiroth, R. Frison, J. L. M. Rupp, T. Lippert, E. J. B. Meier, E. M. Gubler, M. Döbeli, K. Conder, A. Wokaun, and L. J. Gauckler, "Cystallization and grain growth characteristics of yttria-stabilized zirconia thin films grown by pulsed laser deposition," *Solid State Ionics*, vol. 191, pp. 12–23, 2011.

[193] Y. G. Smirnov, "Application of grid-technology for the solution of the nonlinear volume singular integral equation for determining the effective dielectric permeability of nanomaterials," *Transactions of the universities of the Volga region: Physical-Mathematical Sciences*, vol. 3, pp. 39–54, 2008.

[194] A. G. Whittaker, “Diffusion in microwave-heated ceramics,” *Chemistry of Materials*, vol. 17, no. 13, pp. 3426–3432, 2005.

[195] K. I. Rybakov, V. E. Semenov, G. Link, and M. Thumm, “Preferred orientation of pores in ceramics under heating by a linearly polarized microwave field,” *Journal of Applied Physics*, vol. 101, no. 8, 2007.

[196] K. I. Rybakov, E. A. Olevsky, and V. E. Semenov, “The microwave ponderomotive effect on ceramic sintering,” *Scripta Materialia*, vol. 66, pp. 1049 – 1052, 2012.