Analytical Study of Steady State Plasma Ablation from Soft X-Ray Laser Target

by

Susan Sujono

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering

and

Bachelor of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

August 1997

Author ....
Department of Electrical Engineering and Computer Science
September 2, 1997

Certified by ....
Peter L. Hagelstein
~~Associate~~ Professor of Electrical Engineering
Thesis Supervisor

Accepted by ....
Arthur C. Smith
Chairman, Departmental Committee on Graduate Theses

Analytical Study of Steady State Plasma Ablation from Soft X-Ray Laser Target

by

Susan Sujono

Submitted to the Department of Electrical Engineering and Computer Science
on September 2, 1997, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Electrical Engineering
and
Bachelor of Science in Electrical Engineering

Abstract

This thesis project is based on the experimental setup of the MIT table-top EUV laser system. We model the behavior of plasma ablated from soft x-ray laser target in the steady state. The simulation involves solving the relevant hydrodynamics equations for the density, temperature, and velocity of the plasma. First, we solve the problem in one dimension by assuming radial symmetry, using both numerical and analytical minimum-residue methods. In two dimension, we only use analytical method and obtain the desired solution in a closed form. From this solution, we learn more about the plasma profile. This will help us better understand the lack of gain observed in the experiment.

Thesis Supervisor: Peter L. Hagelstein
Title: Associate Professor of Electrical Engineering

Acknowledgments

I would like to thank my advisor, Prof. Peter Hagelstein, for his patience and guidance from the beginning to the completion of this thesis project. I also would like to thank both my parents for their continual support.

Contents

Chapter 1

Introduction

1.1 Background

This thesis project is based on the experiments done with the MIT table-top extreme ultraviolet (EUV) laser system. The goal of these experiments was to demonstrate a gain-length product greater than 4 at a wavelength below 40 nm. However, experiments on several lasant plasmas have detected gain-length products no larger than 1.6 [1].

In this thesis, we attempt to investigate the discrepancy between the observed and the predicted gain-length products. In order to do so, we first need to understand how those values were calculated. We will give an overview of the gain calculation in the next section. For reference, the details of the experimental set up can be found in [1, chapter 4].

1.2 Gain Calculation

The small signal gain can be expressed as [2]

$$\alpha = \sigma N_u \left(1 - \frac{g_u}{g_l}\frac{N_l}{N_u}\right) \tag{1.1}$$

where σ is the stimulated emission cross section, N_u and N_l are the upper and lower state populations, g_u and g_l are the degeneracies of the upper and lower states. Note that we are dealing with a three-level laser. We will refer the states as ground (o), lower (l), and upper (u) levels.

For the stimulated emission cross section, we can take the Doppler limit. This is the high temperature limit, where $\frac{v_{th}}{c}\omega >> \Gamma$. The variable Γ is the inverse lifetime of the upper state. Therefore, at resonance, the cross section is [3]

$$\sigma = \frac{\lambda^2}{2\pi}\frac{A_{ul}}{\Gamma_D}, \tag{1.2}$$

where A_{ul} is the upper level spontaneous decay rate. Also, Γ_D and v_{th} are defined as

$$\Gamma_D = \frac{v_{th}}{c}\omega_o, \tag{1.3}$$

$$v_{th} = \sqrt{\frac{8kT_i}{\pi M}}, \tag{1.4}$$

where ω_o is the resonant frequency, T_i is the ion temperature, and M is the atomic mass.

For lasing to occur, population inversion must be reached. It turns out that some optimum electron density, $N_{e,opt}$, has to be satisfied. The values of N_u and N_l greatly depend on this optimum value.

This optimum value describes the balance between the different mechanisms that populate the upper and lower states. This is the density when the collisional de-excitation rate from the upper state approaches the radiative decay rate of the lower state. Then, the collisional equilibrium between the two states is approached. The optimum density can be expressed in cm^{-3} as [4]

$$N_{e,opt} = c_1 \frac{5.7 \times 10^{26}}{\lambda^{3.5}} \frac{A_{lo}}{A_{ul}} \frac{\sqrt{T_e \Delta E_{ul}}}{< g_{ul} >}, \tag{1.5}$$

where c_1 is an adjustable parameter to match analytic values with experimental results, which is set to 0.1 for the lasants used in the experiment [1]. Meanwhile, ΔE_{ul} is the energy difference between the upper and lower states, $< g_{ul} >$ is Gaunt factor, and λ is the wavelength of the upper to lower level transition. The Gaunt factor depends on whether we consider collisional or recombination schemes.

From [1], for collisional amplifier in the steady state limit, we get

$$\frac{N_l}{N_u} = \frac{N_e C_{ul} + A_{ul} + \epsilon N_e C_{ol}}{N_e C_{lu} + \xi A_{lo}}, \tag{1.6}$$

where ϵ is $\frac{N_o}{N_u}$, $\xi \leq 1$ is escape factor for the lower level radiation, and C's are the collisional excitation rates with the subscripts denoting the 2 states involved.

For the recombination case [1],

$$N_u \approx \kappa \frac{N_e}{Z_i} \frac{P_{rec}}{A_{uo}}, \tag{1.7}$$

$$\frac{N_u}{N_l} \approx \frac{A_{ul}}{\xi A_{lo}}, \tag{1.8}$$

where P_{rec} is the pumping rate for the n-th level in a recombining ion [4].

With 8mm plasma in the experiment, the above equations predict a gain length product of at least 3 should be observed. However, the values detected were no larger than 1.6. This discrepancy is what we would like to investigate in this thesis project.

1.3 Overview of Thesis

There are several possible explanations for the lack of gain in the experiments:

- the electron temperature in the plasma is significantly lower than 150 eV for the collisional scheme at 10^{19} cm^{-3}
- the plasma has a steep density gradient

The two issues above can be better understood if we compute the plasma temperature, density, and velocity. We shall do so in this thesis by solving the hydrodynamical equations that govern the plasma behavior. We will consider the steady state problem only. We solve these equations using both numerical and analytical procedures. The analytical method gives us the solution in a closed form, although we do use computer codes to implement both methods. In fact, we obtain numerical values as results from both methods.

For a reason that will be explained later, we are interested in the two-dimensional (2D) description of the hydrodynamics model. However, before tackling the 2D case, we discuss and solve the one-dimensional (1D) problem, which is simpler. From the 1D solution, we hopefully will gain some intuition on how to solve the 2D case.

Chapter 2

Hydrodynamics Model

When a solid target is irradiated by a high-powered laser beam, plasma is formed on the solid surface. To accurately describe this ablated plasma, let us estimate the Debye length [10], which is $\lambda_D = 7.43 \text{ x } 10^2\sqrt{\frac{T}{N}}$ cm. Here, T is temperature in units of eV and N is number density in cm^{-3}. In the range of interest, we can take T=100 eV and $N=10^{21}/\text{cm}^3$. Thereby, the Debye length is of the order of nanometers.

Since the size of our system - which is in the millimeters - is much larger than the Debye length, the system as a whole will exhibit electrically neutral behavior. It is therefore natural to ignore the microscopic scale behavior of the plasma and only consider the collective dynamics. In other words, we treat the plasma as fluid instead of point-like electrons and ions. We can then use equations from kinetic theory and hydrodynamics to describe the plasma.

In the Eulerian description, the compressible hydrodynamic equations are

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0, \tag{2.1}$$

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} = -\frac{1}{\rho}\nabla P, \tag{2.2}$$

$$\frac{\partial}{\partial t}(\frac{1}{2}|\vec{u}|^2 + \epsilon) + \nabla \cdot (\frac{1}{2}|\vec{u}|^2 + \epsilon)\vec{u} = \nabla \cdot (\kappa \nabla T), \tag{2.3}$$

where ρ is mass density, $\vec{u}$ is velocity, P is pressure, ϵ is internal energy, and κ is thermal conductivity of the plasma. These equations respectively describe the conservations of mass, momentum, and energy of the plasma. Before going further, we need to elaborate on the internal energy of the plasma which we discuss in the next section.

2.1 Thermal Energy of Plasma

The plasma thermal energy is characterized by both the temperatures of the electrons, T_e, and that of the ions, T_i. This internal energy is primarily acquired through heating by the laser beam. Ions have much lower thermal speed than electrons - even at equal temperatures. The ion thermal conductivity is also much smaller than that of the electrons. Consequently, the beam will mostly heat the electrons. Ion temperature changes primarily during plasma expansion due to ion-electron coupling. Because of the weak coupling, typically $T_i \ll T_e$.

Also note that there are Z electrons per ion in the plasma. Thus, in general, the dynamics of our plasma will be governed by the electrons.

For convenience, let us express the energy equation in terms of the electron temperature. Substituting $\epsilon = \frac{3}{2}N_e kT_e$ and using the mass-momentum conservations, the energy equation can be written as

$$\frac{\partial T_e}{\partial t} + (\vec{u}_e \cdot \nabla)T_e = -\frac{2}{3}T_e(\nabla \cdot \vec{u}_e) + \frac{2}{3}\frac{1}{N_e}\nabla \cdot (\kappa_o T_e^{\frac{5}{2}} \nabla T_e) + (\frac{\partial T_e}{\partial t})_{ext}, \qquad (2.4)$$

where N_e is the number density. We have used the form given by Spitzer [7] for the thermal conductivity. We have also added $(\frac{\partial T_e}{\partial t})_{ext}$ to take into account any heating by external sources. Since we know that electrons dominate the plasma behavior, let us drop the subscript e in subsequent discussions.

2.2 Simplifying Assumptions

So far, we have obtained nonlinear and coupled differential equations. From Equations 2.1, 2.2, and 2.4, we would like to solve for five physical variables, which are mass density, velocity, temperature, pressure, and number density. We use the following assumptions to simplify those equations:

- steady state approximation
- constant average degree of ionization, Z_{eff}
- plasma obeys the equation of state of an ideal gas
- most of the laser energy is absorbed on the critical surface
- plasma ablates with local sound speed on the critical surface

We discuss each of the above assumptions separately in the next sections.

2.2.1 Steady State Approximation

From our experimental setup, the plasma is in fact not in the steady state. However, it is indeed a relevant limit to look at. Also, dropping the time derivatives in the hydrodynamics equations lead to a simpler problem.

On the other hand, if we only have space derivatives, the task generally translates to solving a boundary-value type problem. This boils down to satisfying all boundary conditions simultaneously. We will then obtain a large set of linear equations. Of course if the problem is nonlinear, as in our case, we need to linearize it first.

2.2.2 Constant Z_{eff} and Ideal Gas Model

If we assume that the degree of ionization, Z_{eff}, is constant, we have

$$\rho = \frac{1}{Z_{eff}} NM, \tag{2.5}$$

where M is the electron mass. Using the equation of state of an ideal gas,

$$P = NkT, \tag{2.6}$$

the momentum equation in steady state becomes

$$(\vec{u} \cdot \nabla)\vec{u} = -\frac{Z_{eff}k}{M}(\nabla T + \frac{T}{N}.\nabla N), \tag{2.7}$$

we can dot the above equation with $\vec{u}$ and use the following identity

$$\nabla(\vec{A} \cdot \vec{B}) = \vec{A} \times (\nabla \times \vec{B}) + \vec{B} \times (\nabla \times \vec{A}) + (\vec{A} \cdot \nabla)\vec{B} + (\vec{B} \cdot \nabla)\vec{A}, \tag{2.8}$$

to eliminate N in the momentum equation and obtain

$$\vec{u} \cdot \nabla(\frac{1}{2}|\vec{u}|^2) = -\frac{Z_{eff}k}{M}[\nabla T \cdot \vec{u} - T(\nabla \cdot \vec{u})]. \tag{2.9}$$

2.2.3 Absorption of Laser Energy

Let us consider how electrons in the plasma affect the propagation of the laser beam. The dispersion relation for the beam is [5]

$$(\frac{kc}{\omega})^2 = 1 - (\frac{\omega_p}{\omega})^2, \tag{2.10}$$

where k is the beam wavenumber, ω is the beam frequency, and ${\omega_p}^2 = 4\pi e^2 \frac{N}{M}$ is the plasma oscillation frequency. At the critical density, $N_c = \frac{1}{4\pi e^2} M\omega^2$, the plasma frequency matches that of the laser beam and the beam is reflected. This critical density defines the critical surface. In the region between the target and the critical surface, $N > N_c$. The beam cannot propagate in this region. Meanwhile, outside the critical surface, the beam absorption strongly depends on the density [5]. Therefore, we may assume that most of the energy absorption occurs on the critical surface.

We will incorporate this assumption through boundary conditions on some distance, r_o, where the critical surface is located. We will further assume that this surface is fixed, although in reality it moves as the plasma expands [6].

We can conveniently use the boundary conditions on the critical surface as scaling factors and normalize our variables as follows

$$n = \frac{N}{N_o}, \quad |\vec{v}| = \frac{|\vec{u}|}{|\vec{u}_o|}, \quad \tau = \frac{T}{T_o}, \tag{2.11}$$

where the naught subscript denotes the value at the critical surface. We will assume the following values at the surface:

$$\begin{aligned} N_o &= 10^{21}/cm^3, && (2.12)\\ T_o &= 100\ eV, && (2.13)\\ r_o &= 20\ \mu m. && (2.14)\end{aligned}$$

.

2.2.4 Speed of Ablation on the Critical Surface

From both numerical and theoretical analyses [9], it is known that plasma ablates with the local sound speed at the critical surface. This speed is related to the local temperature by [5]

$$|\vec{u}_o| = \sqrt{\frac{Z_{eff}k}{M_i}T_o} \simeq 3x10^7\sqrt{(\frac{Z_{eff}}{A})\ (\frac{T}{keV})}\ cm/s \qquad (2.15)$$

where $M_i = Am_p$ is the ionic mass. Take the case of Ni-like molybdenum for an example. Here, $Z_{eff} = 14$ and $A = 42$ [6], then

$$|\vec{u_o}| = 5.5x10^6\ cm/s \qquad (2.16)$$

2.3 Simplified Conservation Equations

Using all the assumptions above, our conservation equations can then be expressed as

$$\begin{aligned} \nabla\cdot(n\vec{v}) &= 0, && (2.17)\\ \vec{v}\cdot\nabla(\frac{1}{2}|\vec{v}|^2) &= [\tau(\nabla\cdot\vec{v}) - \nabla\tau\cdot\vec{v}], && (2.18)\\ \vec{v}\cdot\nabla\tau &= -\frac{2}{3}\tau(\nabla\cdot\vec{v}) + \frac{2}{3}(\frac{T_o^{\frac{5}{2}}}{N_o v_o})\ (\frac{1}{n})\ \nabla\cdot(\kappa_o\tau^{\frac{5}{2}}\nabla\tau). && (2.19)\end{aligned}$$

Note that the we have not defined what κ_o is. We discuss this in the next chapter.

The momentum conservation is a vector equation; it can be decomposed into as many scalar components as the problem dimensionality requires. As will be explained soon, we are interested in the two-dimensional description.

We have mentioned that the laser pulses are short; accordingly, the plasma expansion is as in Fig. 2-1. This figure is taken from reference [5]. Near the critical surface, the expansion is planar. Going farther, it becomes cylindrical as pictured. We can also imagine if the laser beam is much smaller than the target, we have the planar case. In the other limit, if the beam is much larger, we have the spherical case.

The target in the picture extends in the direction into or out of the paper. We have a symmetry in that direction. Hence, we concern ourselves with two dimensions

(a)

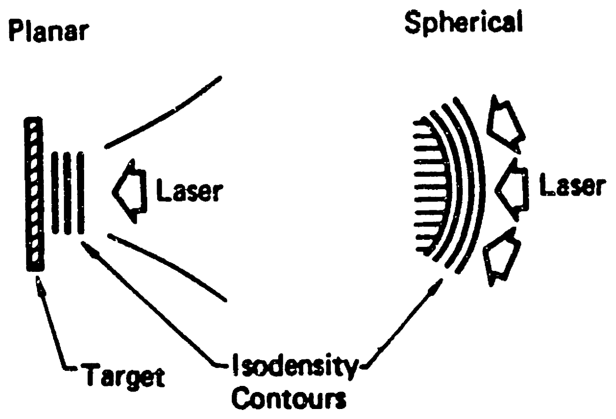

Figure 2-1: Two-dimensional plasma expansion - typical isodensity contours.

only.

We are interested in solving the 2D problem for all the planar, cylindrical, and spherical cases. As we have mentioned, we are interested to solve the simpler 1D problem first. We do this in the next chapter. The 1D case is obtained if we assume radial symmetry. From this easier problem, we learn what procedure(s) to use for solving the 2D problem.

Chapter 3

One-Dimensional Problem

Assuming radial symmetry, the differential operators are reduced to the following form:

$$\nabla f = \frac{df}{dr}\hat{r}, \tag{3.1}$$

$$\nabla \cdot \mathbf{F} = \frac{1}{r^\alpha}\frac{d(r^\alpha F_r)}{dr}, \tag{3.2}$$

for arbitrary scalar f and vector **F**. α=0,1, or 2 corresponds to the planar, cylindrical, or spherical cases respectively. The coordinate r is perpendicular to the critical surface.

This radial symmetry gives us 1D equations, which are simpler than 2D partial differential equations that otherwise describe the plasma. The computer subroutine is then less complex in one dimension, hence easier to debug.

First, rewrite the hydrodynamics equations as:

$$\frac{1}{r^\alpha}\frac{d(r^\alpha nv)}{dr} = 0, \tag{3.3}$$

$$\frac{1}{2}\frac{d(v^2)}{dr} = (\frac{\tau}{r^\alpha v})\frac{d(r^\alpha v)}{dr} - \frac{d\tau}{dr}, \tag{3.4}$$

$$\frac{d\tau}{dr} = -\frac{2}{3}(\frac{\tau}{r^\alpha v})\frac{d(r^\alpha v)}{dr} + \frac{2}{3}(\frac{\kappa_o T_o^{\frac{5}{2}}}{N_o v_o})(\frac{1}{r^\alpha nv})\frac{d}{dr}(\tau^{\frac{5}{2}}\frac{d\tau}{dr}). \tag{3.5}$$

We have assumed that κ_o is constant. Later, we will substitute a more proper description.

Note that a solution of Eq. 3.3 is

$$r^\alpha nv = r_o{}^\alpha n_o v_o. \tag{3.6}$$

Recall that r_o is the location of the critical surface and $n_o = v_o = 1$. We have just eliminated one equation and are left with only two equations.

3.1 Conservation Equations in Scaled Coordinate

For later convenience, let us define a new coordinate

$$\xi = ln\ (\frac{r}{r_o}). \tag{3.7}$$

We then obtain,

$$\frac{df}{dr} = \frac{df}{d\xi}\frac{e^{-\xi}}{r_o}, \tag{3.8}$$

This choice of coordinate intuitively makes sense. The new origin now coincides with the critical surface. Also, we expect the functions to vary most rapidly close to the critical surface.

In terms of the scaled coordinate, Equations 3.3-3.5 are

$$\begin{aligned} v\frac{dv}{d\xi} &= \frac{\tau}{v}\frac{dv}{d\xi} - \frac{d\tau}{d\xi} + \alpha\tau, & (3.9)\\ \frac{d\tau}{d\xi} &= -\frac{2}{3}\tau(\alpha + \frac{1}{v}\frac{dv}{d\xi}) + \frac{2}{3}A\frac{d}{d\xi}(\tau^{\frac{5}{2}}\frac{d\tau}{d\xi}e^{(\alpha-1)\xi}), & (3.10)\end{aligned}$$

where we have defined

$$A = \frac{\kappa_o {T_o}^{\frac{5}{2}}}{N_o r_o v_o}. \tag{3.11}$$

For simplicity, let us set the value of A to unity for now. As we will see later, this value is in fact quite reasonable.

Lastly, the boundary conditions become

$$v(\xi = 0) = \tau(\xi = 0) = 1. \tag{3.12}$$

We also know that

$$\lim_{\xi\to\infty} \tau \to 0 \tag{3.13}$$

3.2 Numerical and Analytical Approaches

As discussed earlier, we propose to solve the hydrodynamics equations both numerically and analytically. Analytical here means that the solution is in closed form. In both cases, we will compute numerical values as results.

In the numerical method, we first linearize the equations and then use Newton's method. As the solution, we obtain the velocity and temperature values at each grid point that best agree with our differential equations.

For the analytical approach, we guess analytical forms of the velocity and temperature functions. Unlike in the numerical method, where we specify the values at each grid, we describe the functions with a set of parameters. In principle, we can also use Newton's method at this point.

We elaborate more on these methods in the next two chapters.

Chapter 4

Numerical Solution in One Dimension by Newton's Method

We propose to use Newton's method to solve the conservation equations as expressed in Equations 3.9-3.10. First, we define the following functions:

$$
\begin{aligned}
D(v,\tau) &= (v - \frac{\tau}{v})\frac{dv}{d\xi} + \frac{d\tau}{d\xi} - \alpha\tau, && (4.1)\\
E(v,\tau) &= \frac{d\tau}{d\xi} + \frac{2}{3}\tau(\alpha + \frac{1}{v}\frac{dv}{d\xi}) - \frac{2}{3}A\frac{d}{d\xi}(\tau^{\frac{5}{2}}\frac{d\tau}{d\xi}e^{(\alpha-1)\xi}). && (4.2)
\end{aligned}
$$

Then, we discretize them [8]:

$$
\begin{aligned}
D_{i+\frac{1}{2}} &= \frac{1}{2h}\,[(v_{i+1} + v_i) - (\frac{\tau_{i+1}}{v_{i+1}} + \frac{\tau_i}{v_i})]\,(v_{i+1} - v_i) - \frac{\alpha}{2}(\tau_{i+1} + \tau_i),\\
E_{i+\frac{1}{2}} &= \frac{1}{h}(\tau_{i+1} - \tau_i) + \frac{1}{3}(\tau_{i+1} + \tau_i)\,[\alpha + \frac{1}{2h}(\frac{1}{v_{i+1}} + \frac{1}{v_i})(v_{i+1} - v_i)]\\
&\quad -\frac{A}{3}(e^{(\alpha-1)\xi_{i+1}} + e^{(\alpha-1)\xi_i})\,[(\alpha-1) + \frac{5}{4h^2}(\tau_{i+1}^{\frac{3}{2}} + \tau_i^{\frac{3}{2}})(\tau_{i+1} - \tau_i)^2\\
&\qquad \frac{1}{4h^2}(\tau_{i+1}^{\frac{5}{2}} + \tau_i^{\frac{5}{2}})(\tau_{i+2} - \tau_{i+1} - \tau_i + \tau_{i-1})]
\end{aligned}
$$

where h is the grid size, $\xi_i = i * h$, v_i=$v(\xi_i)$, and $\tau(i) = \tau(\xi_i)$. We have used centered differencing at half step to avoid spurious solution.

The solution of our differential equations is nothing but the simultaneous roots of D and E at each grid. Since the equations are nonlinear, we need to linearize them. Expanding D up to the first order, we have

$$
\begin{aligned}
D_{i+\frac{1}{2}} &= D_{i+\frac{1}{2}}^{(m)} + (\frac{\partial D_{i+\frac{1}{2}}}{\partial u_i})^{(m)}\delta u_i^{(m)} + (\frac{\partial D_{i+\frac{1}{2}}}{\partial u_{i+1}})^{(m)}\delta u_{i+1}^{(m)} + (\frac{\partial D_{i+\frac{1}{2}}}{\partial \tau_i})^{(m)}\delta\tau_i^{(m)}\\
&\quad + (\frac{\partial D_{i+\frac{1}{2}}}{\partial \tau_{i+1}})^{(m)}\delta\tau_{i+1}^{(m)} + \ldots
\end{aligned}
$$

where $\delta u_i^{(m)} = u_i - u_i^{(m)}$, similarly for $\delta\tau$. The superscript m denotes the value at the

m-th iteration.

If we set the left hand side of the above equation to zero and do the same operations to E, we obtain a system of linear equations which can be written as

$$\begin{pmatrix} \frac{\partial D_{\frac{1}{2}}}{\partial v_1} & \frac{\partial D_{\frac{1}{2}}}{\partial \tau_1} & & & & \\ \frac{\partial E_{\frac{1}{2}}}{\partial v_1} & \frac{\partial E_{\frac{1}{2}}}{\partial \tau_1} & 0 & \frac{\partial E_{\frac{1}{2}}}{\partial \tau_2} & & \\ \frac{\partial D_{\frac{3}{2}}}{\partial v_1} & \frac{\partial D_{\frac{3}{2}}}{\partial \tau_1} & \frac{\partial D_{\frac{3}{2}}}{\partial v_2} & \frac{\partial D_{\frac{3}{2}}}{\partial \tau_2} & & \\ \frac{\partial E_{\frac{3}{2}}}{\partial v_1} & \frac{\partial E_{\frac{3}{2}}}{\partial \tau_1} & \frac{\partial E_{\frac{3}{2}}}{\partial v_2} & \frac{\partial E_{\frac{3}{2}}}{\partial \tau_2} & 0 & \frac{\partial E_{\frac{3}{2}}}{\partial \tau_3} \\ & & \frac{\partial D_{\frac{5}{2}}}{\partial v_2} & \frac{\partial D_{\frac{5}{2}}}{\partial \tau_2} & \frac{\partial D_{\frac{5}{2}}}{\partial v_3} & \frac{\partial D_{\frac{5}{2}}}{\partial \tau_3} \\ & \frac{\partial E_{\frac{5}{2}}}{\partial \tau_1} & \frac{\partial E_{\frac{5}{2}}}{\partial v_2} & \frac{\partial E_{\frac{5}{2}}}{\partial \tau_2} & \frac{\partial E_{\frac{5}{2}}}{\partial v_3} & \frac{\partial E_{\frac{5}{2}}}{\partial \tau_3} \end{pmatrix} \begin{pmatrix} \delta v_1 \\ \delta \tau_1 \\ \delta v_2 \\ \delta \tau_2 \\ \delta v_3 \\ \delta \tau_3 \end{pmatrix} = - \begin{pmatrix} D_{\frac{1}{2}} \\ E_{\frac{1}{2}} \\ D_{\frac{3}{2}} \\ E_{\frac{3}{2}} \\ D_{\frac{5}{2}} \\ E_{\frac{5}{2}} \end{pmatrix}$$

The set of equations we just acquired is of order N, where N is the number of gridpoints. In our case, N is in the order of hundreds. Although the matrix - or the Jacobian - is large in size, it is fortunately banded. Hence, we can conveniently use Gaussian elimination. The computer subroutine for the procedure can be found in Appendix A.

To start the iteration, we need to specify the initial guessed values of v and τ at all grid points. To incorporate the specified boundary conditions at ξ=0 or i=0, we fix v_o and τ_o to be unity during all iterations. However, they will still affect the solution - this is because $D_{\frac{1}{2}}$, $E_{\frac{1}{2}}$, and $E_{\frac{3}{2}}$ depend on v_o and τ_o during all iterations.

Once we have found v and τ, we can express Eq. 3.6 in the scaled coordinate and obtain

$$n(\xi) = \frac{e^{-\alpha\xi}}{v(\xi)}. \tag{4.3}$$

The thin plasma column in the experiment is 100μm x 1-4 cm [1]. Taking 1 cm for r and using 20μm for r_o, then we have $0 < \xi < 6$. Setting $h = 0.01$, or using 600 gridpoints, we obtain results as presented in Figures 4-1 to 4-3. In all plots, the horizontal axes represent ξ. The vertical axes correspond to $\tau(\xi)$, $v(\xi)$, and $n(\xi)$, respectively. The computer routines used to implement this calculation can be found in Appendix B.

From Fig. 4-1, we observe that the plasma expectedly cools down as it expands. This cooling is due to the *PdV* work. From conservation of energy, the velocity should increase meanwhile. Indeed, this can be seen in Fig. 4-2. Also, we observe from the plots that the higher the values of α is, the faster the temperature drops.

In the next chapter, we solve the 1D problem using analytical method. We then will be able to compare the two solutions.

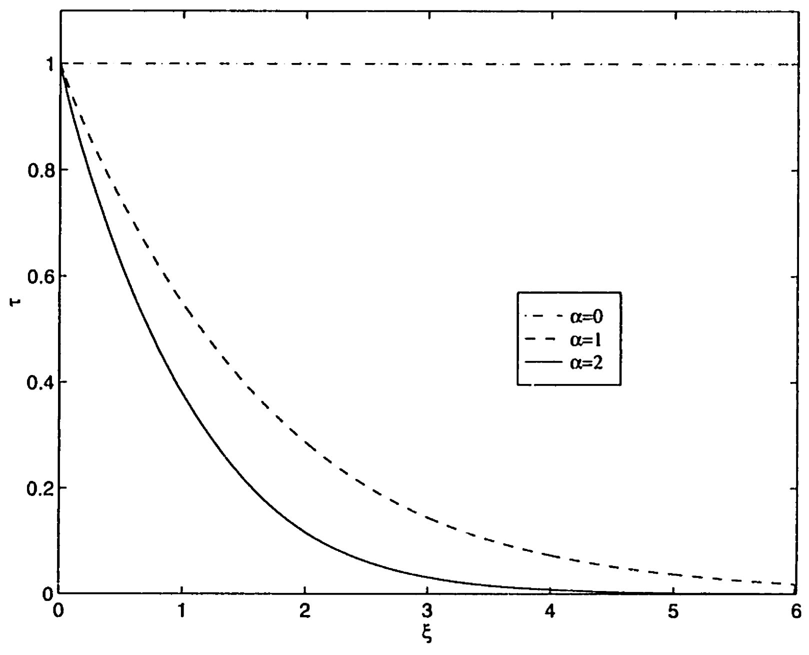

Figure 4-1: Numerical solution of $\tau(\xi)$ obtained from Newton's method

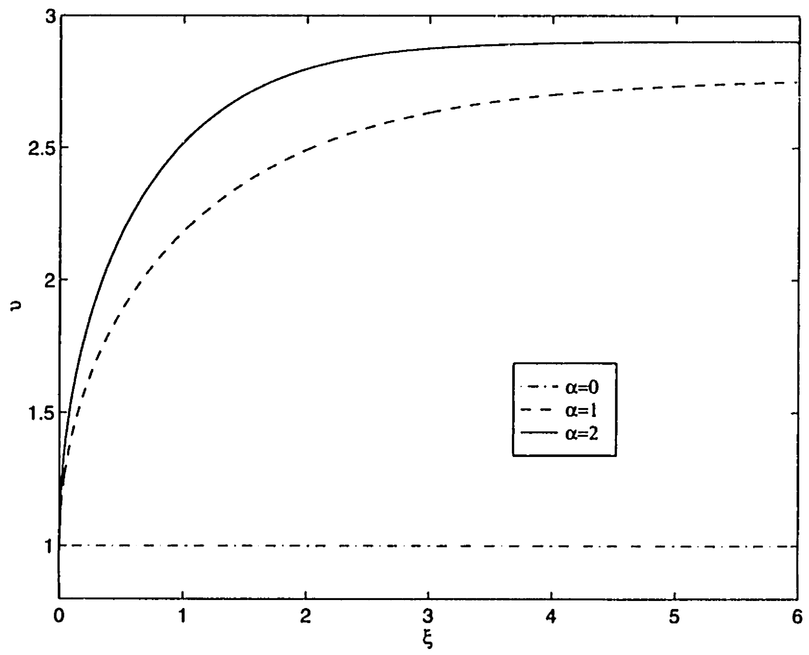

Figure 4-2: Numerical solution of $v(\xi)$ obtained from Newton's method

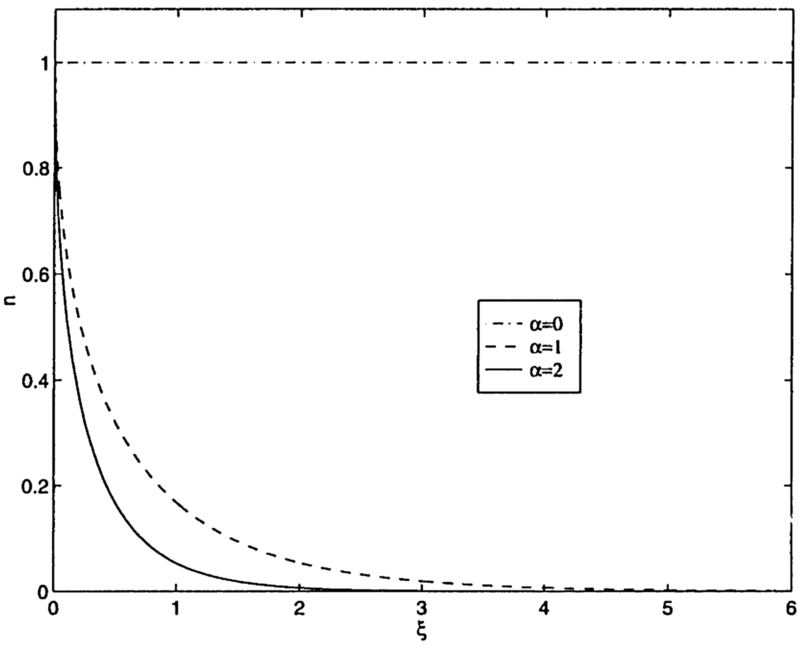

Figure 4-3: Numerical solution of $n(\xi)$ obtained from Newton's method

Chapter 5

Analytical Solution in One Dimension by Minimizing Residues

In this chapter, we strive to calculate temperature and velocity in closed forms without necessarily having any knowledge of the numerical solution. We can do so by minimizing the residues as defined:

$$R_\upsilon[a,b] = \sum_i [\ D_{i+\frac{1}{2}}(\tau(a), \upsilon(b))\ ]^2, \tag{5.1}$$

$$R_\tau[a,b] = \sum_i [\ E_{i+\frac{1}{2}}(\tau(a), \upsilon(b))\ ]^2, \tag{5.2}$$

where D and E were written in Equations 4.1-4.2 and i denotes the i-th gridpoint. The sets of parameters a and b respectively describe τ and υ.

We would like to minimize R_υ and R_τ with respect to their parameters. The minimum corresponds to the roots of D and E, which are the solution to our differential equations.

In other words, we would like to solve

$$\frac{\partial R_\tau}{\partial a_j} = 0, \tag{5.3}$$

$$\frac{\partial R_\upsilon}{\partial b_k} = 0, \tag{5.4}$$

for all j's and k's. This is nothing but solving for the roots of the first partial derivatives. In the previous chapter, we successfully use Newton's method to compute simultaneous roots of D and E. There are as many roots as there are gridpoints, which is of the order of hundreds. In the present case, the total number of parameters a_j's and b_k's will be certainly be much less than 100. We expect to be able to use Newton's method as well.

However, using the computer program as written in Appendix E, it turns out that Newton's method is not very useful. The success of this method depends very much on the initial point we choose. If the gradient at this location happens to point in the wrong direction, we may never converge to the desired minimum. We will instead use the conjugate gradient algorithm taken from [11]. Before using this algorithm, let us first make sure it works properly, which we discuss in the next section. Once, we are confident, we shall use it to minimize the residues.

5.1 Fitting to Numerical Solutions - Checking the Minimization Algorithm

In this section, we check whether the minimization routine we would like to use is working properly. We do so by fitting analytical functions to the numerical solution we previously obtained.

Define the following quantity:

$$\chi = \sum_i [\ f(p, \xi_i) - f_o(\xi_i)\ ]^2, \tag{5.5}$$

where f is some analytical trial function, p is the set of parameters describing f, $f_o(\xi_i)$ is the numerical data on the i-th gridpoint, and $\xi_i = i * h$, where h is the grid size. This function, χ, measures the deviation of our analytical function compared to the numerical data points.

Analogous to the reasoning in minimizing residues, χ is smallest when each contributing term is zero, which means that $f = f_o$ or that the analytical function 'fits' the numerical data. Thence, we need to solve

$$\frac{d\chi}{dp_j} = 0 \tag{5.6}$$

for all j's.

Similar to the Newton's method, the conjugate gradient algorithm also requires us to input an initial guess. Instead of specifying the values at all gridpoints, we pick some functional form and initialize the values of its parameters. The boundary conditions are included by choosing the appropriate form of trial function.

Before writing down our set of trial functions, let us look at Eq. 3.10 more closely. The last term in this equation vanishes as $\xi \to \infty$. Therefore, in this region, we are left with

$$\frac{d\tau}{d\xi} + \frac{2}{3}\tau(\alpha + \frac{1}{\upsilon}\frac{d\upsilon}{d\xi}) \approx 0, \tag{5.7}$$

which implies that

$$\tau^{\frac{3}{2}} \approx \frac{e^{-\alpha\xi}}{\upsilon}. \tag{5.8}$$

Since υ goes to constant at this regime, we can conclude that τ will decay exponentially at points far from the origin. Given that, let us include an exponential dependence in our trial function for τ.

For υ, we do not have any extra information beside the numerical data. Let us

α	0	1	2
a_1	0.000000	0.607442	0.997560
a_2	0.412896	1.046977	1.087227
χ	0.000000	0.006149	0.018985

Table 5.1: Parameters of $\tau = e^{-a_1\xi^{a_2}}$ from fitting the numerical results

α	0	1	2
b_1	0.843699	1.076936	1.598165
b_2	0.902746	0.667898	0.714015
b_3	0.000000	1.816120	1.922853
χ	0.000000	0.072901	0.137610

Table 5.2: Parameters of $v = 1 + b_3(1 - e^{-b_1\xi^{b_2}})$ from fitting the numerical results

choose the simplest functional forms possible, such as

$$\tau(\xi) = e^{-a_1\xi^{a_2}}, \tag{5.9}$$
$$v(\xi) = 1 + b_3(1 - e^{-b_1\xi^{b_2}}), \tag{5.10}$$

where all parameters are zero or greater. The parameters a_2 and b_2 allow the functions to have non-linear behavior for ξ around zero.

Meanwhile, n can be obtained directly from v via Eq. 4.3.

We obtain the parameters as in Tables 5.1-5.2 and as plotted in Figures 5-1 and 5-2. We have ignored the trivial case of $\alpha = 0$ in the plots- and we will continue to do so from now on.

From the plots, it seems that we have acquired a reasonably good fit. One would tend to think that the best fit will have χ to be very close to zero - it may be non-zero due to round-off error. However, our χ's are non-zero.

For any form of trial function, we are always able to obtain a best fit solution, given that our minimization routine works. In this set of best fit solutions, the values of χ will differ. The value is closer to zero as our trial function better approximates the true solution. Hence, we may want to try different trial functions and compare their χ values.

From Tables 5.1, the values of a_2 are around 1 for any α. If a_2 is exactly 1, then τ has a linear dependence at small ξ. For simplicity, let us assume that our new trial function for τ is indeed linear in this region.

Meanwhile, for v, we do not have any extra conditions that will limit our choice. We will simply guess and use fraction of polynomials in ξ, since such form will allow us to add extra parameters in a systematic way, if needed.

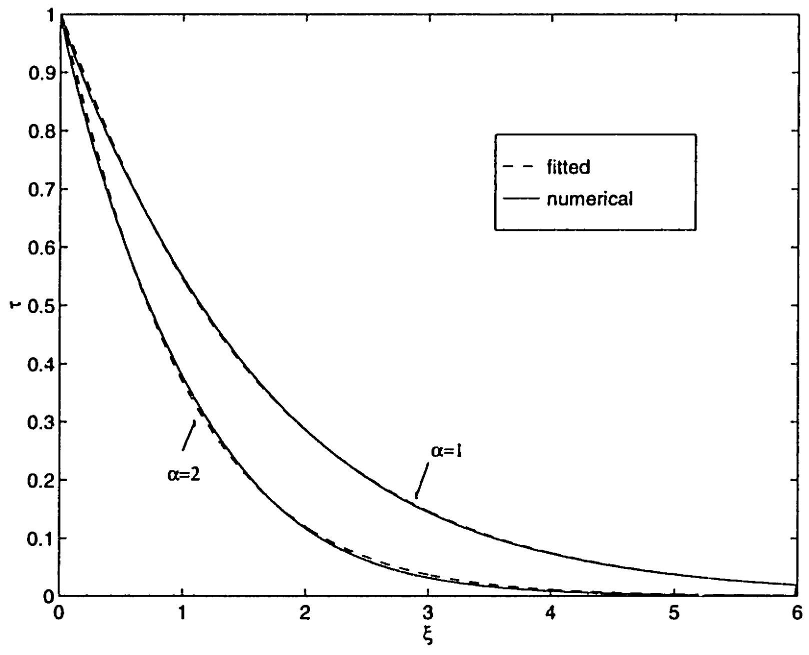

Figure 5-1: Comparison of fitted $\tau = e^{-a_1 \xi^{a_2}}$ and numerical values

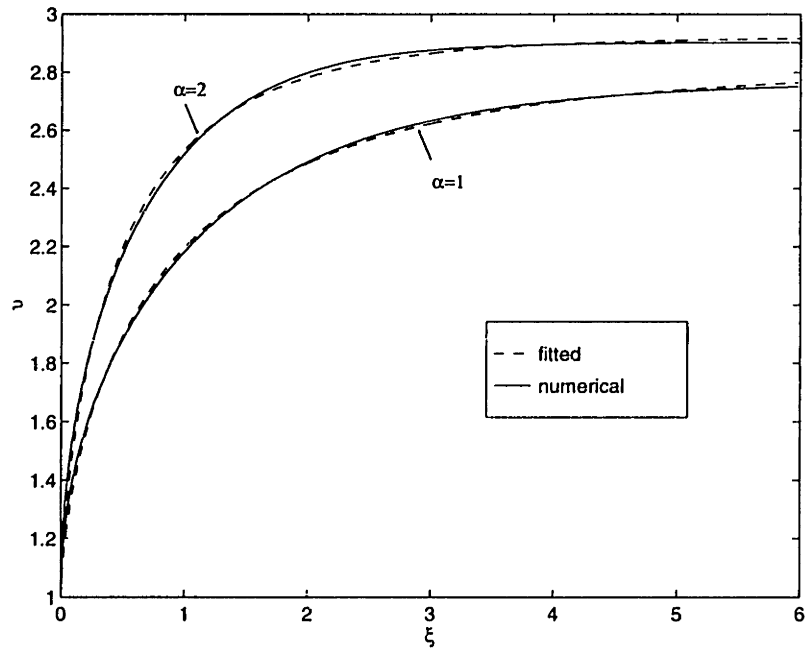

Figure 5-2: Comparison of fitted $\upsilon = 1 + b_3(1 - e^{-b_1 \xi^{b_2}})$ and numerical values

α	1	2
a_o	0.696557	1.221817
a_1	4.725678	2.414493
χ	0.002824	0.002454

Table 5.3: Parameters of $\tau = \frac{1+a_1}{1+a_1 e^{a_o \xi}}$ from fitting the numerical results

α	1	2
b_o	0.486862	0.472110
b_1	1.653894	2.952702
b_2	0.284879	1.142508
c_1	0.265511	0.344391
c_2	0.105610	0.335146
χ	0.000141	0.000235

Table 5.4: Parameters of $\upsilon = 1 + (\frac{b_1\xi + b_2\xi^2}{1+c_1\xi+c_2\xi^2})^{b_o}$ from fitting the numerical results

.

For now, let us try the following trial functions

$$\tau(\xi) = \frac{1+a_1}{1+a_1 e^{a_o \xi}}, \tag{5.11}$$

$$\upsilon(\xi) = 1 + (\frac{b_1\xi + b_2\xi^2}{1+c_1\xi+c_2\xi^2})^{b_o}. \tag{5.12}$$

The results are in Tables 5.3-5.4 and illustrated in Figures 5-3 and 5-4.

Comparing the χ values, the new set of trial functions are better. For both τ and υ, we have changed the functional forms. Although, for τ, we have the same number of parameters. While for υ, we have added two more parameters.

Note that for small ξ, τ behaves linearly, while υ has a square root dependence. Also, at large ξ, the exponent in τ seems to depend on α in the following manner: $\tau \approx e^{-c\alpha\xi}$, where c is some constant greater than zero. This makes sense in light of Eq. 5.8. For large ξ, υ is constant, then

$$\tau \approx e^{-\frac{2}{3}\alpha\xi}. \tag{5.13}$$

From Tables 5.3 and 5.4, the results are certainly in the right vicinity.

At this point, we can continue finding new sets of trial functions such that the values of χ are as close to zero as possible. However, let us not stray from the goal of this section, which is to check whether or not the minimization algorithm works. From the results, it certainly seems that we do have a working minimization routine.

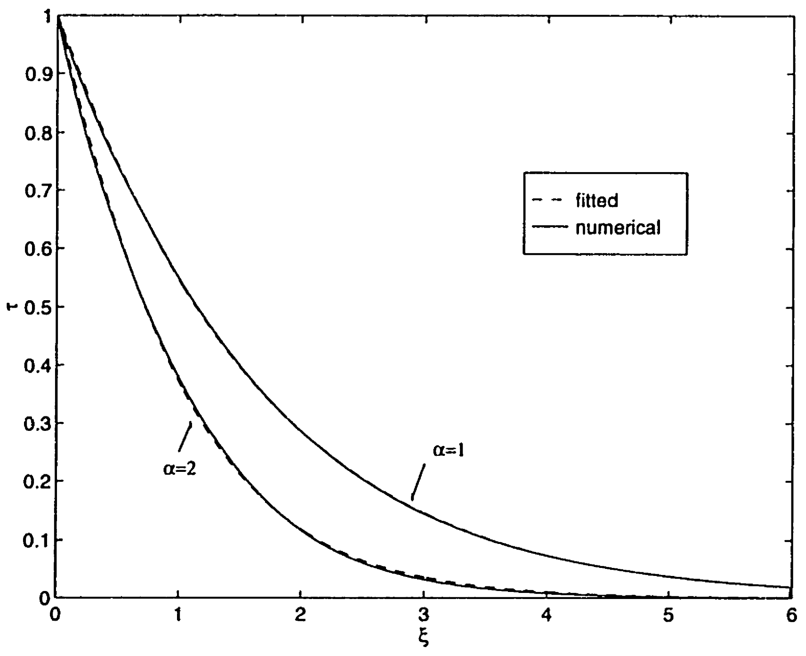

Figure 5-3: Comparison of fitted $\tau = \frac{1+a_1}{1+a_1 e^{a_o \xi}}$ and numerical values

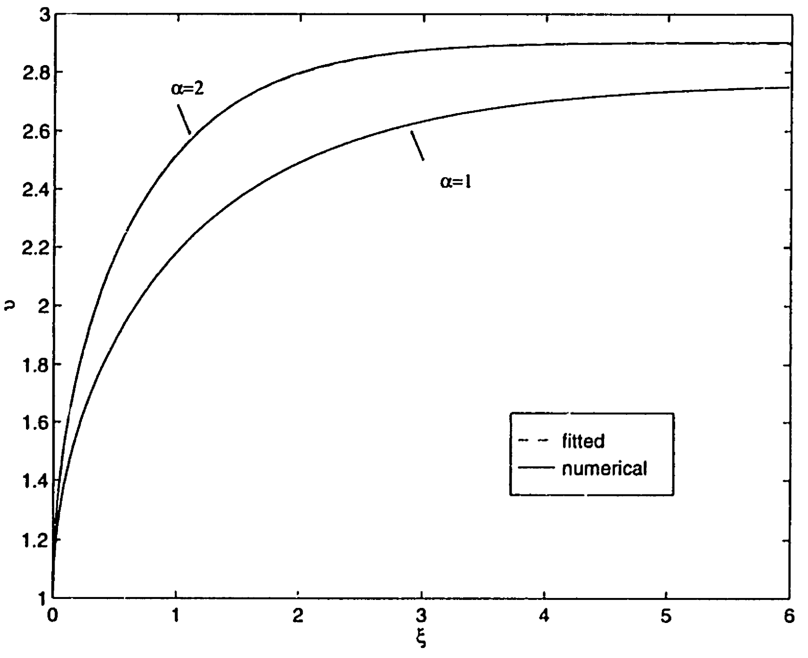

Figure 5-4: Comparison of fitted $\upsilon = 1 + (\frac{b_1\xi + b_2\xi^2}{1+c_1\xi+c_2\xi^2})^{b_o}$ and numerical values

α	1	2
a_o	0.716265	1.189054
a_1	4.787242	3.221097
b_o	0.486220	0.482529
b_1	1.658895	3.177194
b_2	0.283187	1.113252
c_1	0.259938	0.442663
c_2	0.109458	0.330154
R_τ	0.002545	0.000958
R_v	0.007238	0.011886
iteration	4	4

Table 5.5: Parameters of $\tau = \frac{1+a_1}{1+a_1 e^{a_o \xi}}$ and $v = 1 + (\frac{b_1\xi + b_2\xi^2}{1+c_1\xi+c_2\xi^2})^{b_o}$ from minimizing the residues

5.2 Minimum Residues

Now, we are finally ready to solve our differential equations by minimizing the residues. Notice that we are dealing with two coupled differential equations. Previously, we took care of this coupling by expressing the system of equations as written in Chapter 4. In analogue to that approach, we would need to simultaneously solve for all a's and b's. This may prove to be imprudent since the minimization routine grows as N^2 where N is the number of parameters [11].

What we will do instead is to iterate the minimization procedure between R_v and R_τ. For example, given a set of initial guesses, we minimize R_v with respect to a's while fixing all b's. Then, we fix this new set of a's, minimize R_τ, and obtain an updated set of b's. We then go back and minimize R_v, etc. This approach is more efficient than simultaneously minimizing for a and b, especially when there are plenty of parameters.

Let us use the same forms of trial functions, as written in Equations 5.11-5.12. We obtain the parameters as listed in Table 5.5. From the plots, it seems that fitting and minimizing residues give comparably good results. Direct comparison of the previous χ's to R_v and R_τ is inappropriate since they measure different types of 'error' or deviation: χ with respect to the numerical solution, while R's with respect to the residual functions.

However, for χ and R's - the better the trial function approximates the true solution, the smaller their values are. We have observed this in case of χ values, as evident from Tables 5.1-5.4.

At any rate, the minimum residue method has succeeded in producing a reasonable 1D solution in closed form. However, we have so far judged this by comparing our results to the numerical solution. Recall that the goal of the analytical method is to get good results - without - any knowledge of the numerical solution. To be more precise, at this point we can only conclude that the minimum residue method works,

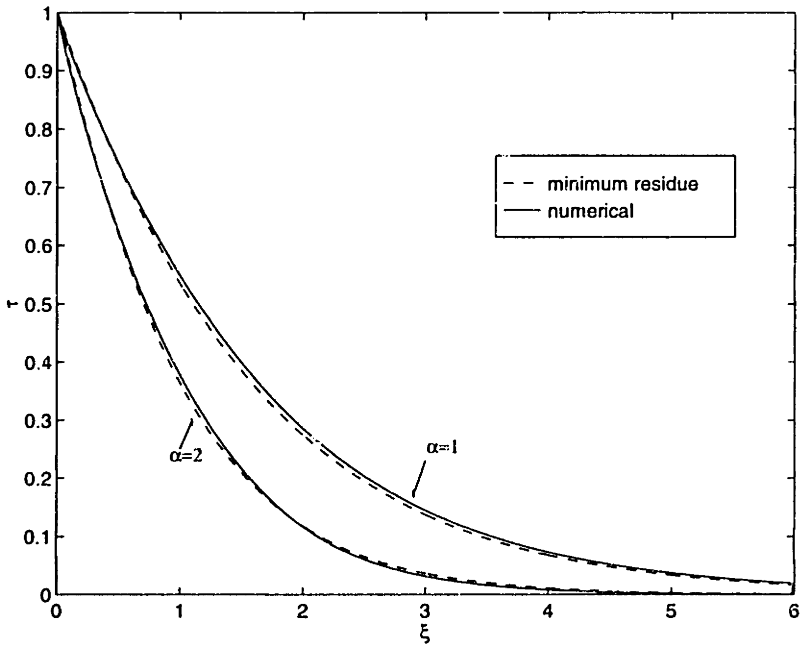

Figure 5-5: Comparison of minimum residue solution of $\tau = \frac{1+a_1}{1+a_1 e^{a_o \xi}}$ and numerical results

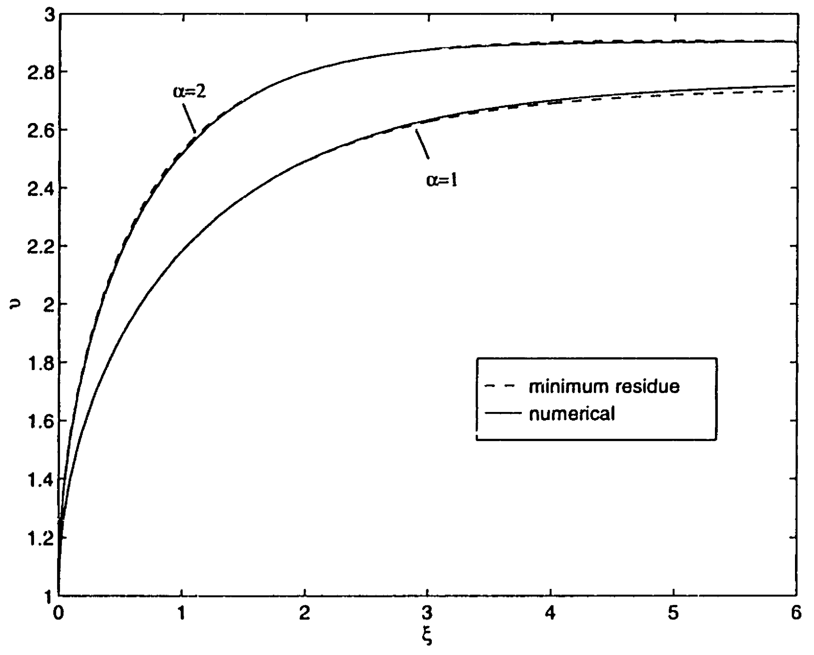

Figure 5-6: Comparison of minimum residue solution of $\upsilon = 1 + (\frac{b_1\xi + b_2\xi^2}{1+c_1\xi+c_2\xi^2})^{b_o}$ and numerical results

as long as we have some intuition on what the solution looks like.

Physically, based on the 1D solution, we have a very good idea how the 2D solution ought to behave. However, since we will have no 2D numerical data to compare our analytical solution to, how will we judge how good that solution is? Recall that we expect the 2D numerical data will be difficult to obtain, hence the motivation of this chapter.

In the next chapter, we explain a procedure that helps us decide how good our analytical solution is.

One last point before we close this chapter: one might wonder why we do not simply compute the roots of the residuals instead of minimizing them. After all, minimization gives us results that are the roots anyway. This is because multidimensional minimization is generally easier to solve than multidimensional root finding. Intuitively, if we pick a surface in our N-dimensional space, we can find the direction where the gradient changes the most. Following this direction, we will eventually hit a minimum. This procedure of 'sliding downhill' is effectively one dimensional. No analogous procedure is known in root finding [11].

Chapter 6

Adding More Parameters

In this chapter, we discuss a procedure that lets us judge the merit of our minimum residue solution independent of the numerical solution. This will be particularly useful in two dimensions, where the numerical results may be difficult to compute.

We have mentioned that the better a trial function approximates the true solution, the less its residue is. We propose to construct better trial functions by systematically adding more parameters to the existing function. Intuitively, these parameters describe the ability of our trial function to accommodate the behavior of the true solution. Thus, adding more degrees of freedom can only decrease the residuals. If adding more parameters changes the residuals insignificantly, then our trial function is a good description of the true solution. This function constitutes the best answer we have.

Then, how big a change should be considered as significant? One approach is to compare this change to the machine precision - but this may be too fine a measurement than we really need. Another approach is to compare this change to the error inherent in the computational algorithm being used. Yet another possibility, perhaps the simplest one, is to take advantage of the results in one dimension.

Based on the comparison with the 1D numerical results, we indeed have acquired a very good 1D analytical solution. Given this, let us see how much the residues vary if we add more parameters. Then, we will have some idea how big a change to expect when we have a good solution.

Let us now discuss how we can add the parameters in a systematic manner. For the temperature, we would like to keep the exponential dependence - see the discussion around Eq. 5.8. Therefore, in the most general form,

$$\tau(\xi) = \frac{1 + \sum_{i=1}^{n_\tau} a_i}{1 + \sum_{i=1}^{n_\tau} a_i e^{a_o i \xi}}. \tag{6.1}$$

This way, we can add one parameter at a time.

For the velocity, we know that it approaches some constant value at large ξ. Let us take this form

$$v(\xi) = 1 + \left(\frac{\sum_{i=1}^{n_v} b_i \xi^i}{1 + \sum_{i=1}^{n_v} c_i \xi^i}\right)^{b_o}. \tag{6.2}$$

(n_τ, n_υ)		(2,5)	(3,5)	(4,5)		(2,7)	(2,9)
a_o		0.716265	0.722262	0.722584		0.721398	0.721542
a_1		4.787242	4.572960	4.572998		4.576465	4.576841
a_2		0.000000	-0.010038	-0.010267		0.000000	0.000000
a_3		0.000000	0.000000	0.000040		0.000000	0.000000
b_o		0.486220	0.482759	0.482710		0.488448	0.488448
b_1		1.658895	1.655705	1.657530		1.656681	1.656681
b_2		0.283187	0.282396	0.282701		0.253153	0.253153
b_3		0.000000	0.000000	0.000000		-0.029030	-0.029030
b_4		0.000000	0.000000	0.000000		0.000000	-0.000000
c_1		0.259938	0.252012	0.253187		0.213911	0.213911
c_2		0.109458	0.108455	0.108298		0.113928	0.113928
c_3		0.000000	0.000000	0.000000		-0.010439	-0.010439
c_4		0.000000	0.000000	0.000000		0.000000	-0.000000
R_τ		0.002545	0.001824	0.001745		0.001219	0.000723
R_υ		0.007238	0.005581	0.005589		0.006740	0.005739

Table 6.1: Parameters of τ and υ from minimizing the residues with α=1

Due to the form we picked above, we will have to add two parameters every time.

First, let us modify τ while keeping υ fixed. For the initial guesses, we use the minimum-residue values and set the extra parameter, a_2, to be zero. Then, we use this set of newly obtained parameters, a_o, a_1, a_2, as the initial guesses when we add the next parameter, a_3; which will be initialized to zero. We continue doing so until extra parameter affects our solution negligibly.

From the tables, indeed the sum of R's decreases as more parameters are added. The variations are in the order of 1%-10%. Later, we shall gauge our success based on these values.

Before solving the 2D problem, there is one thing left to do in one dimension. So far, we have assumed the thermal conductivity, κ, is constant, as mentioned in Chapter 3. We would like to use a more physical description in the next chapter.

(n_τ, n_υ)		(2,5)	(3,5)	(4,5)		(2,7)	(2,9)
a_o		1.189054	1.189639	1.189762		1.191425	1.151856
a_1		3.221097	3.221086	3.221041		3.221053	3.295502
a_2		0.000000	0.000164	0.000084		0.000000	0.000000
a_3		0.000000	0.000000	0.000133		0.000000	0.000000
b_o		0.482529	0.482023	0.483060		0.481802	0.481804
b_1		3.177194	3.178394	3.180845		3.177142	3.177143
b_2		1.113252	1.113486	1.115108		1.113047	1.113049
b_3		0.000000	0.000000	0.000000		-0.000226	-0.000221
b_4		0.000000	0.000000	0.000000		0.000000	0.000025
c_1		0.442663	0.442757	0.443556		0.442350	0.442351
c_2		0.330154	0.329153	0.332451		0.330064	0.330063
c_3		0.000000	0.000000	0.000000		-0.000402	-0.000416
c_4		0.000000	0.000000	-0.000090		0.000000	0.000000
R_τ		0.000958	0.000935	0.000615		0.000840	0.000614
R_υ		0.011886	0.011845	0.011863		0.011852	0.009802

Table 6.2: Parameters of τ and υ from minimizing the residues with α=2

Chapter 7

Physical Results in One Dimension

So far, we have assumed κ to be constant and set A as in Eq. 3.11 to be unity. In this section, we shall consider the fact that κ is r-dependent. In its most general form [5],

$$\begin{aligned}\kappa &= 20\,(\frac{2}{\pi})^{\frac{3}{2}}\,\delta\,[\frac{kT_e^{\frac{5}{2}}}{e^4 Z(ln\ \Lambda)\sqrt{m_e}}], &(7.1)\\ \delta &\approx 0.095\,\frac{Z+0.24}{1+0.24Z}, &(7.2)\end{aligned}$$

where k is the Boltzmann constant, T_e is electron temperature, e is electron charge, Z is the effective ionic charge state, $(ln\ \Lambda)$ is Coulomb logarithm, and m_e is electron mass. The Coulomb logarithm can be computed using [10]

$$\Lambda = 24 - ln(\frac{\sqrt{N_e}}{T_e}), \tag{7.3}$$

where N_e is the electron density in cm^{-3} and T_e is in eV.

Because of the r-dependence, Eq. 3.10 becomes

$$\frac{d\tau}{d\xi} = -\frac{2}{3}\tau(\alpha + \frac{1}{v}\frac{dv}{d\xi}) + \frac{2}{3}(\frac{T_o^{\frac{5}{2}}}{N_o r_o v_o})\frac{d}{d\xi}[\ \kappa_o(\tau^{\frac{5}{2}}\frac{d\tau}{d\xi}e^{(\alpha-1)\xi})\ ]. \tag{7.4}$$

We will assume that r_o=20 μm and v_o can be obtained from Eq. 2.15.

Incorporating the above changes, and taking Ni-like Mo, we obtain results written in Table 7.1. We compare these values to the case when A was set to unity as in Figures 7-1 to 7-3.

Recall that the dimensionless variables are scaled as follows:

$$n = \frac{N}{N_o},\quad |\vec{v}| = \frac{|\vec{u}|}{|\vec{u}_o|},\quad \tau = \frac{T}{T_o}, \tag{7.5}$$

where N_o=10^{21}/cm^3, T_o=100 eV, and u_o is computed from Eq. 2.15. For Ni-like Mo,

α	1	2
a_o	0.738448	1.245027
a_1	4.796987	2.937139
b_o	0.485049	0.479091
b_1	1.668513	3.176505
b_2	0.285861	1.113113
c_1	0.262218	0.434441
c_2	0.113900	0.340701
R_τ	0.005463	0.013142
R_υ	0.001988	0.001069
iteration	3	6

Table 7.1: Parameters of $\tau = \frac{1+a_1}{1+a_1 e^{a_o \xi}}$ and $\upsilon = 1 + (\frac{b_1\xi + b_2\xi^2}{1+c_1\xi+c_2\xi^2})^{b_o}$ from minimizing residues for Ni-like molybdenum - using the physical form of thermal conductivity

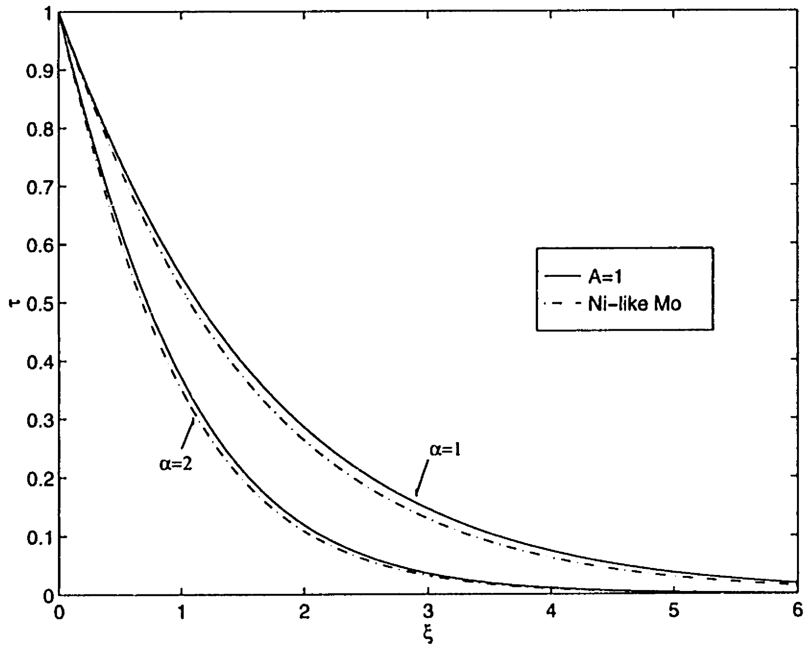

Figure 7-1: Comparison of setting A=1 and physical result in temperature for Ni-like Mo

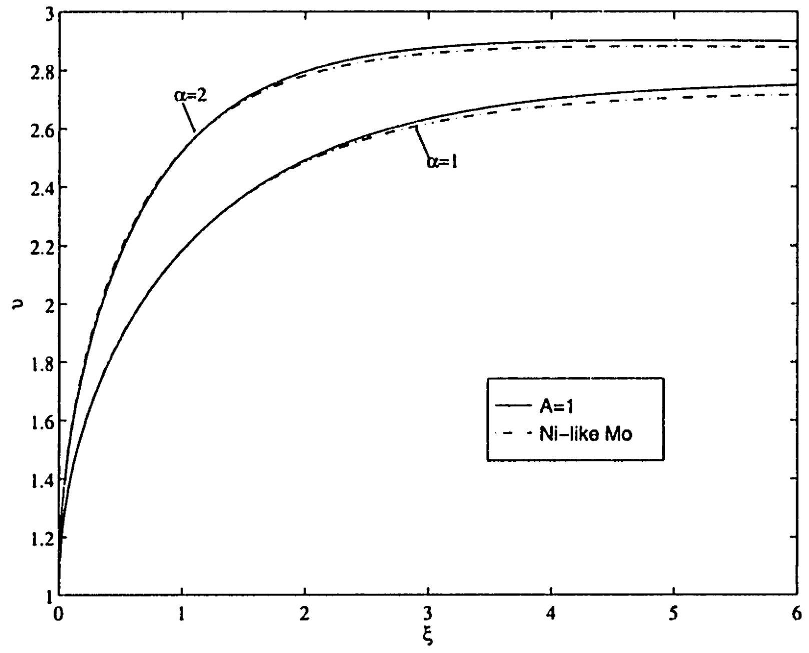

Figure 7-2: Comparison of setting A=1 and physical result in velocity for Ni-like Mo

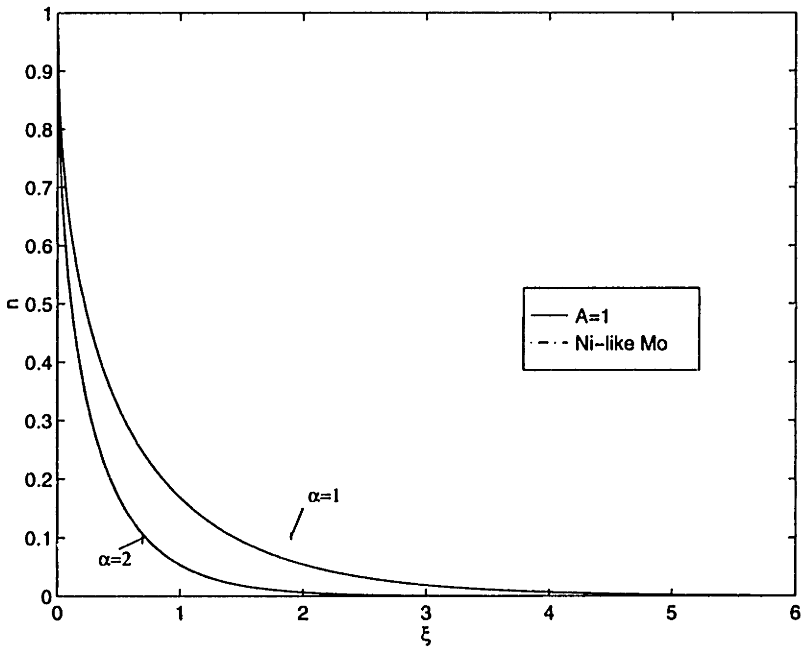

Figure 7-3: Comparison of setting A=1 and physical result in density for Ni-like Mo

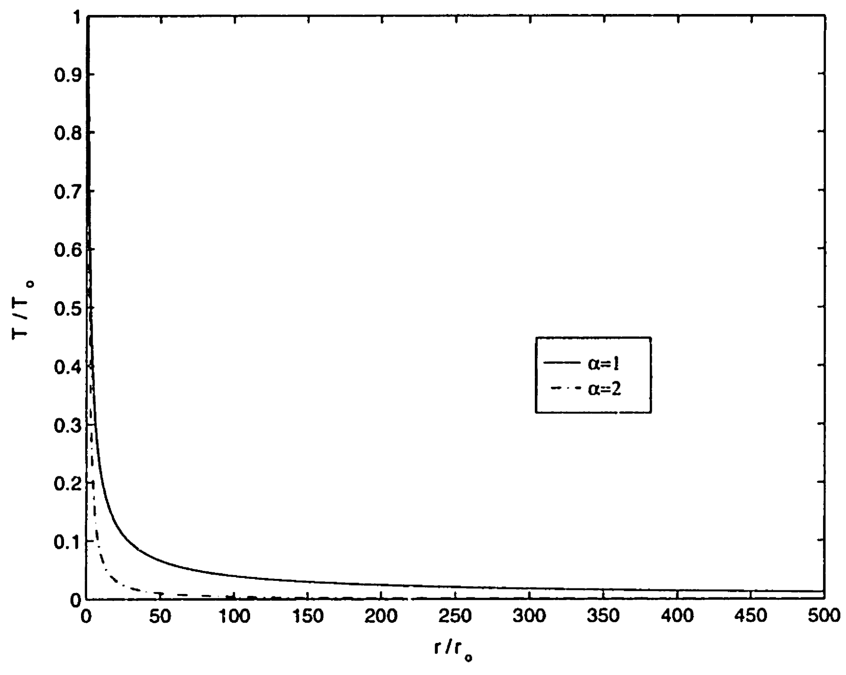

Figure 7-4: Temperature in one dimension for Ni-like Mo

Z=14, then A=.6. The value of A is in fact of the order of unity.

Now, let us plot the result in terms of the unscaled physical variables. Hence, the horizontal axis now denotes r, instead of ξ. The results are in Figures 7-4 to 7-6.

Finally, we are ready to face the 2D challenge. Based on the success of minimum-residue method in one dimension, we will also use it in two dimensions. We discuss the 2D problem in the next chapter.

In two dimensions, the hydrodynamics equations become a set of partial differential equations. We expect the solution to be more complicated, hence greater number of parameters will be needed to describe temperature, velocity, and density functions. The velocity is a vector function now. However, we can still use the same minimization routine. As in one dimension, we will first set κ to be constant. Once we obtain reasonable result, we shall modify it.

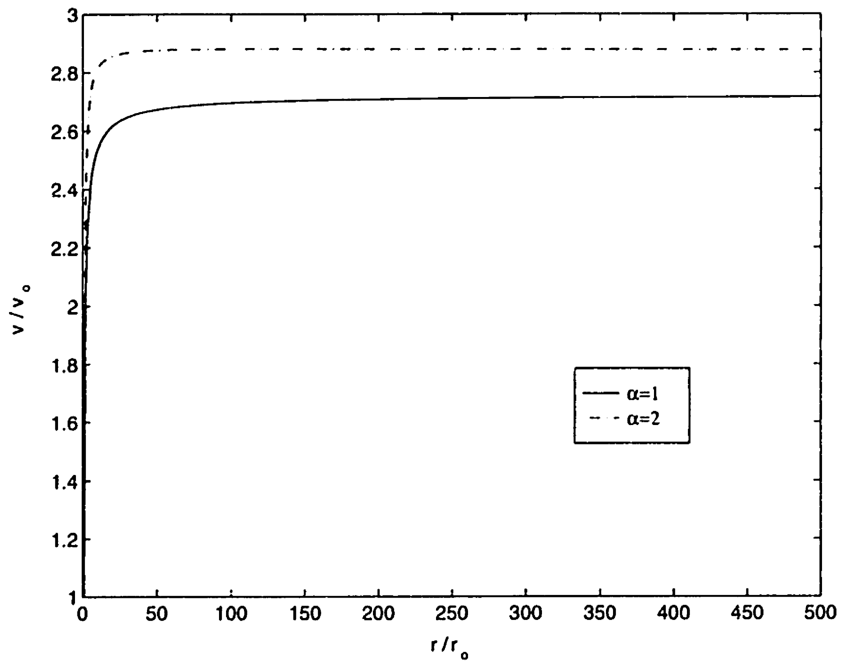

Figure 7-5: Velocity in one dimension for Ni-like Mo

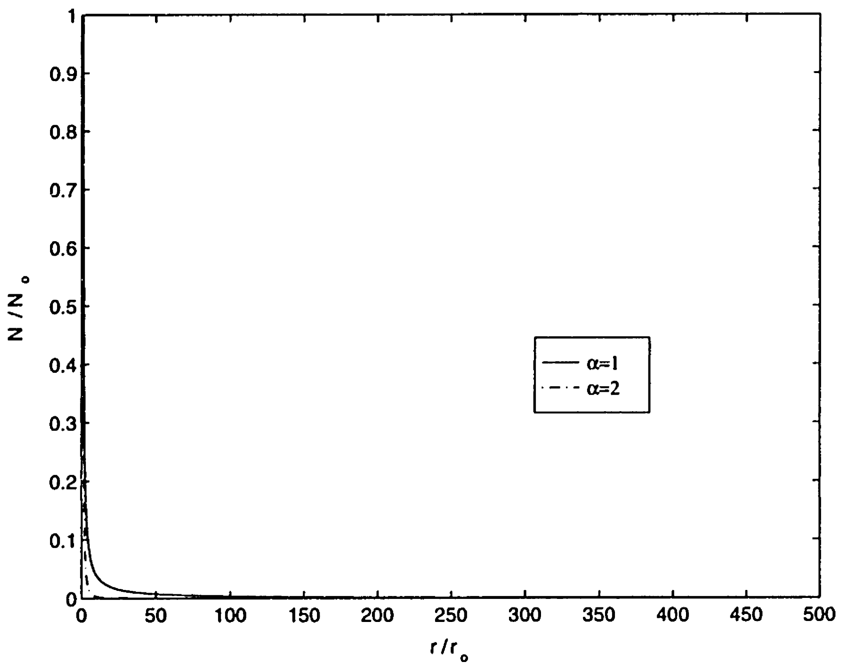

Figure 7-6: Density in one dimension for Ni-like Mo

Chapter 8

Two-Dimensional Problem

In this chapter, we extend the 1D problem to two dimensions by lifting the radial symmetry assumption. First, we write down the hydrodynamics equations using the proper 2D operators. Then, we explain how the boundary conditions in one dimension are modified. Afterwards, we discuss what symmetries are present in two dimensions.

In two dimensions, we intend to use the minimum residue method. This is because we do not presently have a 2D numerical solution.

To use the analytical method, we are required to define trial functions for the desired physical variables. This is more difficult in two dimensions since we do not have any numerical solution. However, even without these numerical data, we do have some intuition on the 2D solution. We can gain some understandings on the behavior of 2D plasma by making observations on the 1D solution. We do this in Section 8.4. Once we have an intuitive picture of the solution, only then can we write down appropriate trial functions.

8.1 Hydrodynamics Description in Two Dimensions

In two dimensions, the differential operators can be written as

$$\nabla f = \frac{\partial f}{\partial z}\,\hat{z} + \frac{\partial f}{\partial r}\,\hat{r}, \tag{8.1}$$

$$\nabla \cdot \mathbf{F} = \frac{\partial F_z}{\partial z} + \frac{1}{r^\alpha}\frac{\partial (r^\alpha F_r)}{\partial r}, \tag{8.2}$$

for any scalar function f and vector function **F**. Meanwhile, α=0,1,2 corresponds to the planar, cylindrical, or spherical cases, respectively. As before, $\hat{r}$ is the direction perpendicular to the critical surface. If we refer to Fig. 2-1, $\hat{z}$ is the vertical direction.

For the 1D problem, we have used the form in Eq. 2.9 for the momentum equation, which is

$$\vec{v}\cdot\nabla(\frac{1}{2}|\vec{v}|^2) - \tau(\nabla\cdot\vec{v}) + \nabla\tau\cdot\vec{v} = 0. \tag{8.3}$$

There is another form which is more useful in two dimensions. Let us rewrite Eq. 2.7

where the density has not been eliminated yet. This equation can also be expressed as

$$(v \cdot \nabla)v + \nabla\tau + \frac{\tau}{n}\nabla n = 0. \tag{8.4}$$

Then, the governing equations in two dimensions are

$$\begin{aligned}
\frac{\partial(nv_z)}{\partial z} + \frac{1}{r^\alpha}\frac{\partial(r^\alpha n v_r)}{\partial r} &= 0, && (8.5)\\
v_z\frac{\partial v_z}{\partial z} + v_r\frac{\partial v_z}{\partial r} + \frac{\partial \tau}{\partial z} + \frac{\tau}{n}\frac{\partial n}{\partial z} &= 0, && (8.6)\\
v_z\frac{\partial v_r}{\partial z} + v_r\frac{\partial v_r}{\partial r} + \frac{\partial \tau}{\partial r} + \frac{\tau}{n}\frac{\partial n}{\partial r} &= 0, && (8.7)\\
v_z\frac{\partial \tau}{\partial z} + v_r\frac{\partial \tau}{\partial r} + \frac{2}{3}\tau\left[\frac{\partial v_z}{\partial z} + \frac{1}{r^\alpha}\frac{\partial(r^\alpha v_r)}{\partial r}\right] & \\
-\frac{2}{3}\left(\frac{\kappa_o T_o^{\frac{5}{2}}}{N_o v_o}\right)\left[\frac{\partial}{\partial z}\left(\tau^{\frac{5}{2}}\frac{\partial \tau}{\partial z}\right) + \frac{1}{r^\alpha}\frac{\partial}{\partial r}\left(r^\alpha \tau^{\frac{5}{2}}\frac{\partial \tau}{\partial r}\right)\right] &= 0. && (8.8)
\end{aligned}$$

Equations 8.6 and 8.7 are the z and r components of Eq. 8.3, respectively. As in one dimension, we have again assumed κ_o to be constant.

Now, for convenience, let us define a new coordinate system:

$$\begin{aligned}
x &= ln\left(\frac{r}{r_o}\right), && (8.9)\\
y &= \left(\frac{z}{r_o}\right). && (8.10)
\end{aligned}$$

Similar to the 1D case, the line $x = 0$ coincides with the critical surface.

The partial derivatives become

$$\begin{aligned}
\frac{\partial f}{\partial z} &= \frac{\partial f}{\partial y}\frac{1}{r_o}, && (8.11)\\
\frac{\partial f}{\partial r} &= \frac{\partial f}{\partial x}\frac{e^{-x}}{r_o}. && (8.12)
\end{aligned}$$

In the new coordinate, Equations 8.5-8.8 are transformed to

$$\begin{aligned}
\frac{\partial(nv_y)}{\partial y} + e^{-x}\left[\alpha n v_x + \frac{\partial(nv_x)}{\partial x}\right] &= 0, && (8.13)\\
v_y\frac{\partial v_y}{\partial y} + \frac{\partial \tau}{\partial y} + \frac{\tau}{n}\frac{\partial n}{\partial y} + e^{-x}\left[v_x\frac{\partial v_y}{\partial x}\right] &= 0, && (8.14)\\
v_y\frac{\partial v_x}{\partial y} + e^{-x}\left[v_x\frac{\partial v_x}{\partial x} + \frac{\partial \tau}{\partial x} + \frac{\tau}{n}\frac{\partial n}{\partial x}\right] &= 0, && (8.15)\\
v_y\frac{\partial \tau}{\partial y} + \frac{2}{3}\tau\frac{\partial v_y}{\partial y} - \frac{2}{3}\frac{A}{n}\frac{\partial}{\partial y}\left(\tau^{\frac{5}{2}}\frac{\partial \tau}{\partial y}\right) & \\
+ e^{-x}\left\{v_x\frac{\partial \tau}{\partial x} + \frac{2}{3}\tau\left(\alpha v_x + \frac{\partial v_x}{\partial x}\right)\right. &
\end{aligned}$$

$$-\frac{2}{3}\frac{A}{n}\ [\ \alpha e^{-x}\tau^{\frac{5}{2}}\frac{\partial\tau}{\partial x}+\frac{\partial}{\partial x}\ (\ e^{-x}\tau^{\frac{5}{2}}\frac{\partial\tau}{\partial x})\ ]\ \}\ =\ 0, \tag{8.16}$$

where A was defined in Eq. 3.11. Unlike in the 1D case, we are unable to solve Eq. 8.13 here - compare this to Eq. 3.3 and its solution, Eq. 3.6. Instead of having two coupled equations as in one dimension, we now have four of them. Also, these equations are now partial instead of ordinary differential equations.

The solution of the equations above have to satisfy certain boundary conditions. These conditions are slightly more complex than in one dimension. Let us discuss them in the next section.

8.2 Boundary Conditions

At the origin, we have the following condition:

$$n(x=0,y=0)=\tau(x=0,y=0)=v(x=0,y=0)=1. \tag{8.17}$$

From the 1D solution, it is reasonable to expect that

$$\lim_{x\to\infty,y\to\infty}\{n,\tau\}\to 0. \tag{8.18}$$

The next boundary condition has been consistently shown by both numerical and theoretical analyses [9]. Both analyses agree that plasma ablates with the local sound speed at the critical surface, as expressed in Eq. 2.15. At the critical surface, we also assume the plasma ablates perpendicularly to the surface. Thence, we acquire these conditions:

$$v_x(x=0,y)\ =\ \sqrt{\tau(x=0,y)}, \tag{8.19}$$
$$v_y(x=0,y)\ =\ 0. \tag{8.20}$$

Except the last one, the boundary conditions in two dimensions are direct extension of the 1D case. Next, we explain the symmetries present in two dimension.

8.3 Symmetries

Referring to Fig. 2-1, we can see that y and $-y$ axes are indistinguishable. It is therefore sufficient to solve the 2D problem in the upper half, or $y\geq 0$, region only. This implies that the solution functions are even with respect to y.

Meanwhile, with respect to x, no symmetry exists. We have a problem where the energy source is on the positive x side, by definition. We obtain plasma ablation because of the laser energy absorption. This fact breaks any symmetry, since no source exists in the $-x$ axis. More precisely, we cannot allow any source to exist, since we will have an entirely different problem otherwise. In other words, there is plasma on positive x side and none on the negative side - hence, symmetry between the two sides would be unphysical.

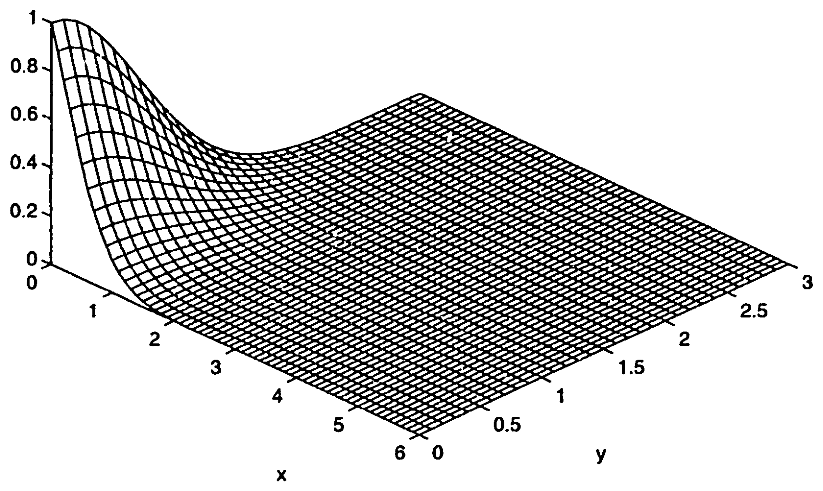

Figure 8-1: Qualitative behavior of the 2D density and temperature functions

8.4 Expected Qualitative Behavior of the Solution in Two Dimensions

Beside the $y \to -y$ symmetry, which has no direct analog in one dimension, everything in two dimensions so far has been a simple generalization of its 1D counterpart. We expect the 2D solution to be similar to the 1D result as well. Recall that we intend to solve the 2D problem by the minimum-residue method. Knowing the qualitative behavior of the 2D solution will aid us in defining trial functions for the density, temperature, and velocity functions.

For the density and temperature functions, we expect them to to decay away from the origin, just as in one dimension. We imagine these functions to have ellipsoidal contours. Qualitatively, the solution for temperature and density functions will probably look like Figures 8-1 and 8-2.

The velocity function is quite different in two dimensions; it is now a vector instead of a scalar function. Certainly, we would think the magnitude will have some semblance to the 1D velocity.

To have a better intuition, let us imagine connecting the velocity vectors to form velocity lines - similar to the electric field lines obtained from an arbitrary electric field. Along these lines, it is reasonable that the velocity behaves similarly to the 1D case - the magnitude increases and approaches some constant value for points far away frcm the origin. Of course, this constant may differ for each velocity line.

In the same time, the direction of the field will appear to fan out of the critical surface. We also expect the field to be radial for points far from the origin. Qualitatively, the velocity field will behave as in Fig. 8-3

Now, let us see how these pictures wil help us to define trial functions for the

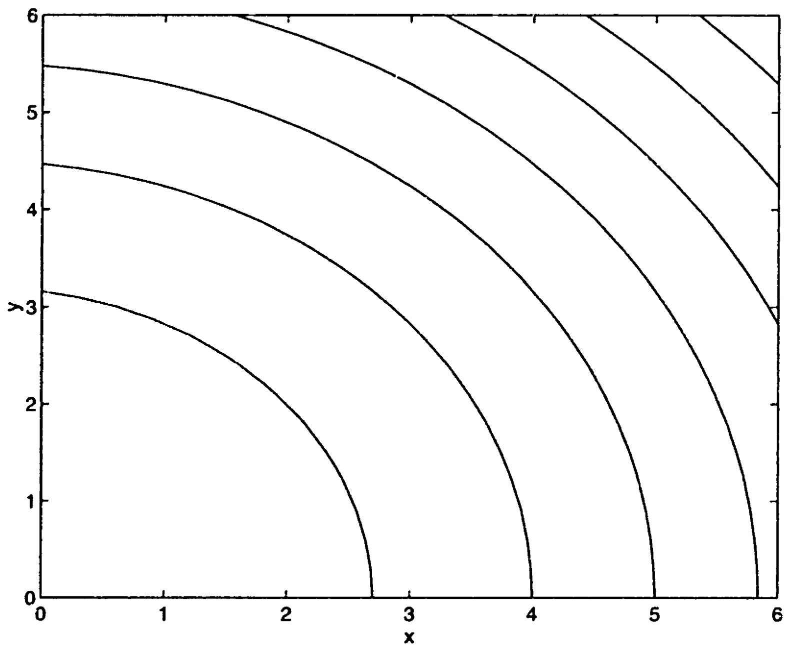

Figure 8-2: Expected contour lines of 2D density and temperature

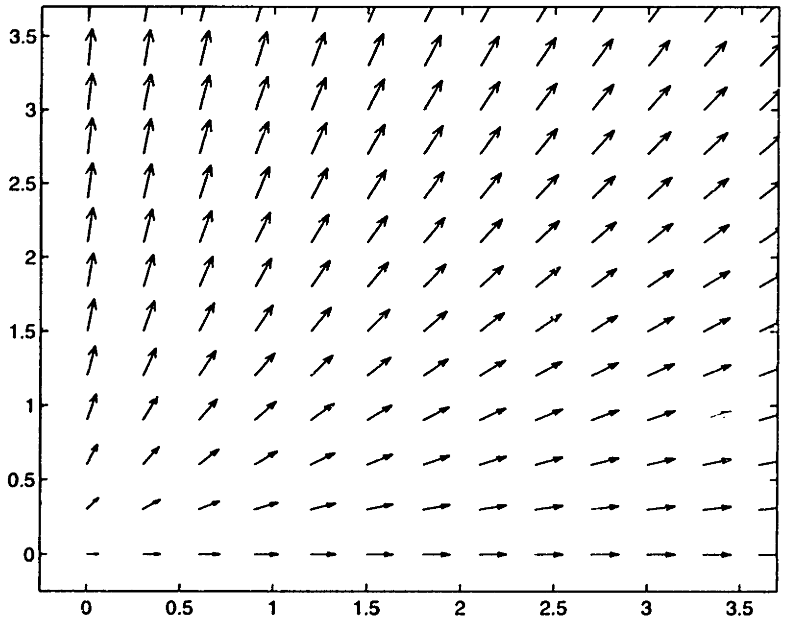

Figure 8-3: Qualitative behavior of the velocity field

density, temperature, and velocity. As mentioned, the density and temperature will probably have ellipsoidal contours. It is then natural to write them as a function of these contour lines.

For the velocity field, there are different ways to describe it. For each description, the hydrodynamics equations in 8.15-8.14 may be expressed differently for convenience.

From the differential equations, an obvious choice is to describe the field by its x and y components. However, we do not have any intuition how each component behaves separately.

A better approach is to derive the field from some potential function - analogous to the potential used to describe an electromagnetic field. We will elaborate more on this in the next chapter, Chapter 9.

Yet another possibility is to parameterize the velocity field by its magnitude and direction. We shall do so in Chapter 10. This approach is probably the most convenient since we have a good intuition on how the magnitude and direction of the field behave.

For all cases, we will parameterize the density and temperature using the contour lines as aforementioned.

Chapter 9

Potential Function

The velocity field can be obtained in the following manner:

$$v = \nabla\phi + \nabla \times \mathbf{A}, \tag{9.1}$$

where ϕ is scalar and $\mathbf{A}$ is vector functions.

Ideally, we would like to solve for both ϕ and $\mathbf{A}$ separately. From both, we obtain a different sets of solution for n, τ, and v. We will then sum the two solutions and use that as our trial functions. It is not obvious that we can simply take the sum of the two solutions, since the governing differential equations are not linear.

We can pick out the scalar potential contribution if we assume irrotational fluid flow. This means

$$\nabla \times v \;=\; 0. \tag{9.2}$$

Hence,

$$v = \nabla\phi. \tag{9.3}$$

This potential function is unique up to a constant.

From the plot of the desired velocity field in Figure 8-3, the potential function will probably have ellipsoidal contours. Let us parameterize these lines as ζ, where ζ is some form of ellipsoid, such as

$$\zeta = a_o + a_1 x^2 + a_2 y^2 + a_3 x. \tag{9.4}$$

A simple trial function to try out is

$$\phi = \sqrt{\zeta} - \frac{1}{c}\, e^{-cx}\sqrt{\tau(0,y)}. \tag{9.5}$$

The first term ensures the boundary condition infinity is satisfied. As we know, the velocity magnitude approaches constant along all velocity lines at points far away from the origin. Thus, we expect that the potential to be roughly linear in this region A caveat is due here. Depending on the form of the potential function, imposing this linearity may cause the constant to be the same regardless of θ. We do not have any prior knowledge whether or not this is the case.

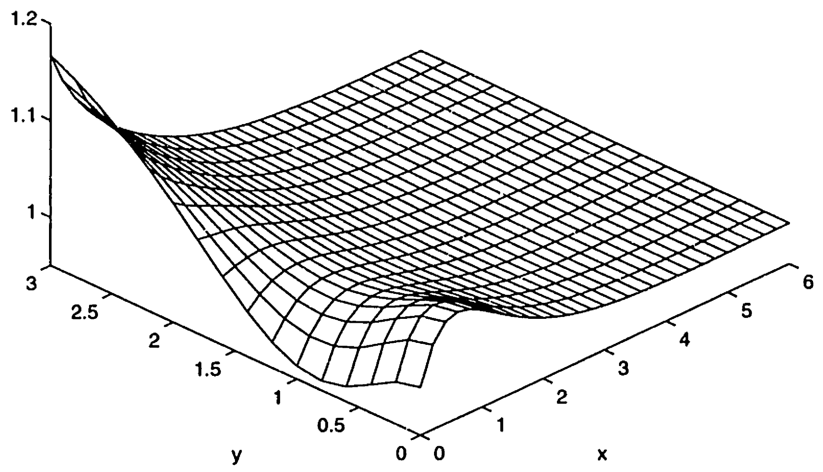

Figure 9-1: The velocity magnitude obtained from ϕ

The second term guarantess the boundary conditions at x=0 is satisfied. Refer to Eq. 8.19 for this condition. This term will decay such that the linearity condition at infinity is still satisfied. In regards to the boundary conditions, the potential function is incipiently well-behaved.

However, the potential function above has an artificial flavor to it. We construct two independent functions that satisfy the the boundary conditions close to the origin and another one for infinity separately, then 'glue' them together. For certain set of parameters, the function may not be as smooth as we expect in some region, as illustrated in Figure 9-1.

In the figure, we plot the velocity magnitude. Notice the local maximum. This is unphysical since there is no source or sink in our system. Let us plot the two velocity components separately to understand what may have produced this maximum.

From Fig. 9-3, the y-component is certainly well-behaved. The problem comes from the x-part as plotted in Fig. 9-2. This becomes more transparent if we separately plot the contributions from the first and second terms of Eq. 9.5. Each contributes a smooth function as shown in Figures 9-4 and 9-5. It is the difference in the amount of their contributions that causes the local maximum.

Nevertheless, it is plausible that there are sets of parameters where this problem may not occur. Before we explore and use this trial function any further, let us attempt to form another trial function - one that will hopefully include the boundary conditions more naturally.

Now, we shall attempt to include the $\sqrt{\tau}$ term inside ζ instead. For example, we

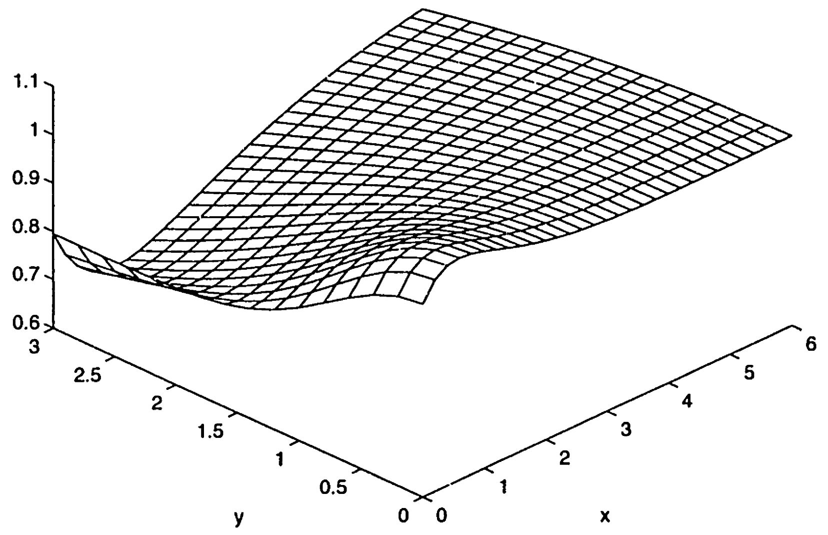

Figure 9-2: The x-component of the trial velocity

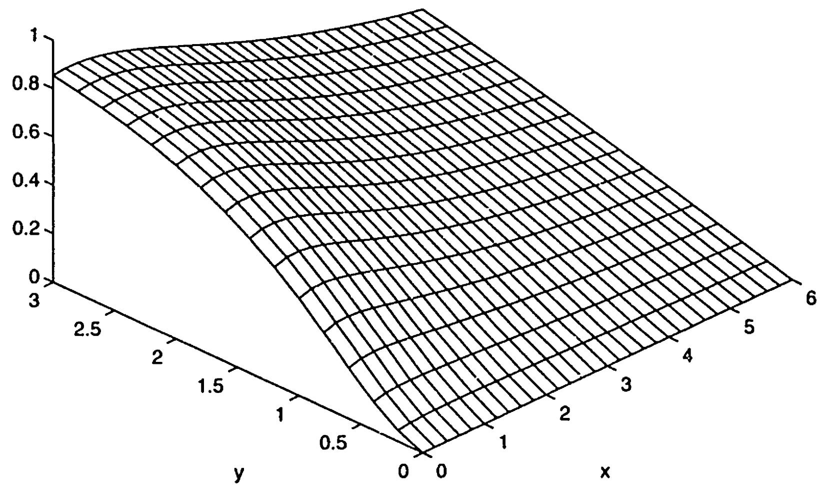

Figure 9-3: The y-component of the trial velocity

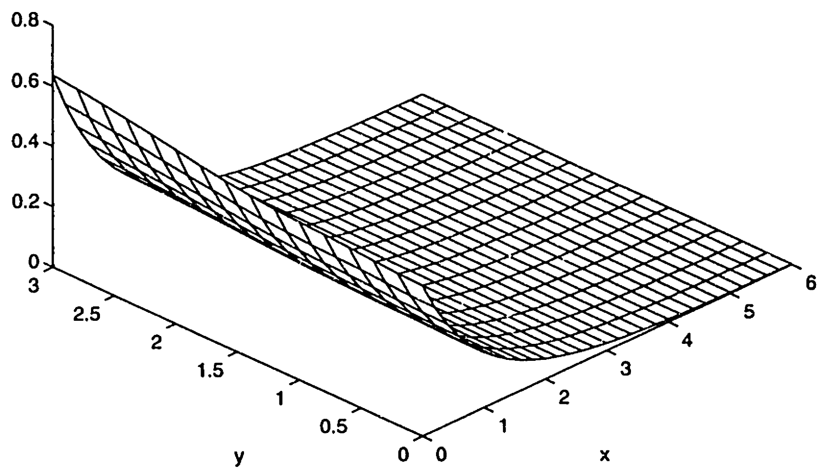

Figure 9-4: Contribution of the exponential term in v_x

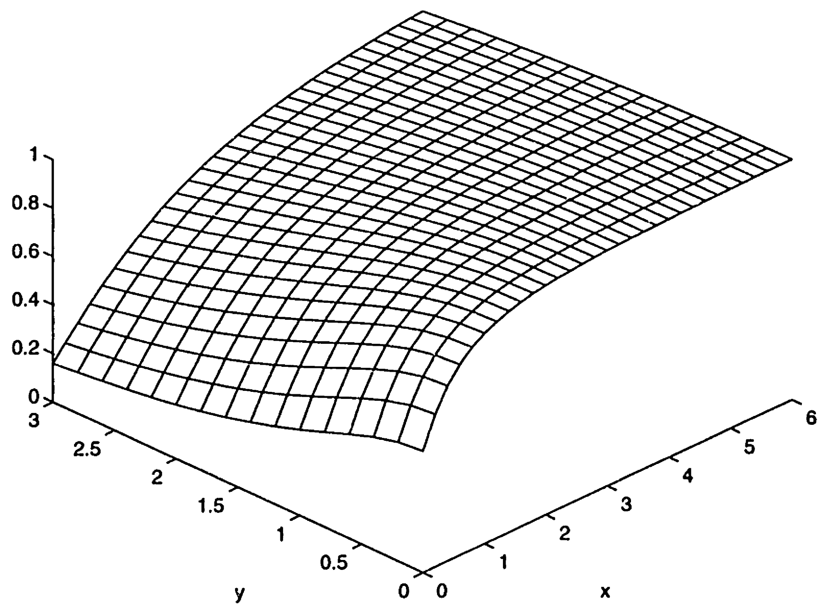

Figure 9-5: Contribution of the ζ term in v_x

can define the following:

$$\zeta = \sqrt{a_1 x^2 + a_2 y^2 + 2\sqrt{\frac{\tau(x=0,y)}{a_2 y^2 + a_3}}x + a_2}. \tag{9.6}$$

Then, if we set $\phi = \zeta$, all the boundary conditions are still satisfied indeed.

However, recall that for the one-dimensional case, the velocity increases in a square-root manner for points close to the origin. We expect that the speed may not increase in a linear manner for the two-dimensional case as well. To allow for such possibility, we would need a term proportional to ζ^α in the potential function, where $\alpha < 1$. We will do so as follows

$$\phi = \zeta^\alpha e^{-\zeta} + \zeta. \tag{9.7}$$

We put the exponential term such that at far away, ϕ still behave linearly.

This form of ϕ also allows us to add more parameters and accomodate more complex behaviors in a systematic manner. We can attempt to express ϕ in terms of power series in ζ. In its most general form, our potential function is of the form

$$\phi = \zeta^\alpha e^{-\zeta} + \frac{\sum_i b_i \zeta_i}{\sum_i b_i}, \quad i = 1, 2, ... \tag{9.8}$$

where b_i's are constant parameters and ζ_i is defined as

$$\zeta_i = \left(a_1 x^2 + a_2 y^2 + \frac{2x}{i}\sqrt{\tau(x=0,y)}(a_2 y^2 + a_3)^{1-\frac{i}{2}} + a_3 \right)^{\frac{i}{2}}. \tag{9.9}$$

The ζ in Eq. 9.6 is the special case when $i = 1$. Note that we cannot simply use the powers of ζ due to the boundary condition at $x = 0$. Of course, for each new form of ζ, we may have to redefine ζ_i.

As we have mentioned earlier, adding two forms of independent functions like above easily gives rise to local maximum in ϕ. There probably exist sets of parameters with do not cause such problem. Nevertheless, finding them is not all that obvious. This is problematic especially that we need to have a reasonable initial guess to start off the minimization procedure.

As we have discussed previously, the success of any minimization procedure relies heavily on how close the initial guess is to the solution. Currently, we have a case where good initial guesses are difficult to come up with. We cannot expect to have a good solution when we have pathological values to start with.

Unless we can construct a better scalar potential, it is futile to continue and elaborate on the vector potential contribution. Instead of delving further into this problem, let us instead explore the next possible description of the velocity field, parameterization by its magnitude and direction. We do so in the next chapter.

Chapter 10

Parameterize Velocity Field by Its Magnitude and Direction

In this chapter, we specify the velocity field by its magnitude and direction. The density and temperature functions are assumed to have ellipsoidal contours as explained. Hence, we will parameterize them as functions of the contour lines.

The direction of the field is measured using the angle between the velocity vector and the y-axis. The field is then specified as

$$\begin{aligned} v_x(x,y) &= v(x,y)\cos\theta(x,y), && (10.1)\\ v_y(x,y) &= v(x,y)\sin\theta(x,y), && (10.2)\end{aligned}$$

where v is the magnitude and θ the angle of the velocity vector, respectively.

At this point, it is more convenient to rewrite the hydrodynamics equations in 8.13-8.16 and define the following residue functions:

$$\begin{aligned} R_n &= \sin\theta\,(\,\frac{v}{n}\frac{\partial n}{\partial y} + \frac{\partial v}{\partial y}\,) + v\cos\theta\,\frac{\partial\theta}{\partial y} \\ &\quad + e^{-x}\,[\,\cos\theta(\alpha v + \frac{v}{n}\frac{\partial n}{\partial x} + \frac{\partial v}{\partial x}) - v\sin\theta\frac{\partial\theta}{\partial x}\,], && (10.3)\\ R_u &= \sin\theta\,(v\frac{\partial v}{\partial y} + \frac{\partial\tau}{\partial y} + \frac{\tau}{n}\frac{\partial n}{\partial y}\,) \\ &\quad + e^{-x}\cos\theta\,(\,v\frac{\partial v}{\partial x} + \frac{\partial\tau}{\partial x} + \frac{\tau}{n}\frac{\partial n}{\partial x}\,), && (10.4)\\ R_\theta &= v^2\sin\theta\,\frac{\partial\theta}{\partial y} + \cos\theta\,(\,\frac{\partial\tau}{\partial y} + \frac{\tau}{n}\frac{\partial n}{\partial y}) \\ &\quad + e^{-x}\,[\,v^2\cos\theta\,\frac{\partial\theta}{\partial x} - \sin\theta\,(\,\frac{\partial\tau}{\partial x} + \frac{\tau}{n}\frac{\partial n}{\partial x}\,)], && (10.5)\\ R_\tau &= \sin\theta(\,v\frac{\partial\tau}{\partial y} + \frac{2}{3}\tau\frac{\partial v}{\partial y}\,) + \frac{2}{3}\tau v\cos\theta\,\frac{\partial\theta}{\partial y} - \frac{A}{3n}[5\tau^{\frac{3}{2}}(\frac{\partial\tau}{\partial y})^2 + 2\tau^{\frac{5}{2}}\frac{\partial^2\tau}{\partial y^2}] \\ &\quad + e^{-x}\{\,[\,\cos\theta\,(\,v\frac{\partial\tau}{\partial x} + \frac{2}{3}\tau\frac{\partial v}{\partial x} + \frac{2}{3}\alpha\tau v\,)\, - \frac{2}{3n}\tau v\sin\theta\,\frac{\partial\theta}{\partial x}\,] \end{aligned}$$

$$-\frac{A}{3n}e^{-x}\left[\,5\tau^{\frac{3}{2}}\left(\frac{\partial\tau}{\partial x}\right)^2+2\tau^{\frac{5}{2}}\frac{\partial^2\tau}{\partial x^2}+2(\alpha-1)\tau^{\frac{5}{2}}\frac{\partial\tau}{\partial x}\,\right]\}. \tag{10.6}$$

Equations 10.3 and 10.6 are obtained by direct substitution. Equations 10.4 and 10.5 can be computed from

$$\begin{aligned}(Eq.\ 10.4) &= \sin\theta(Eq.\ 8.14)+\cos\theta(Eq.\ 8.15), &(10.7)\\ (Eq.\ 10.5) &= \cos\theta(Eq.\ 8.14)-\sin\theta(Eq.\ 8.15). &(10.8)\end{aligned}$$

In effect, we have separated the differential equations for the magnitude and direction of the field.

10.1 Trial Functions in Two Dimensions

Let us first define the following for convenience:

$$\eta(a)=x^2+a_1y^2+a_2, \tag{10.9}$$

where η describes an ellipsoid.

Using the above, the density trial function can be written as

$$n(x,y)=e^{-b_1(\eta(a)+b_2\eta(a)^2+\ldots)^{b_o}}. \tag{10.10}$$

The ellipsis in Eq. 10.10 stands for higher order term. The next term would be $b_3\eta(a)^3$, and after that $b_4\eta(a)^4$, etc.

We use the same functional form for temperature

$$\tau(x,y)=e^{-c_1(\eta(d)+c_2\eta(d)^2+\ldots)^{c_o}}. \tag{10.11}$$

For the velocity magnitude, we use the following:

$$v=\sqrt{\tau(0,y)}+1-e^{-f_1((x\eta(g))+f_2(x\eta(g))^2+\ldots)^{f_o}}. \tag{10.12}$$

Meanwhile, for the angle, we use

$$\theta=\tan^{-1}k_1(\gamma+k_2\gamma^2+\ldots)^{k_o}\tanh q_1(\eta(p)+q_2\eta(p)^2+\ldots)^{q_o} \tag{10.13}$$

where γ is

$$\gamma(x,y)=\frac{xy}{m+x} \tag{10.14}$$

The $\tan^{-1}$ term ensures the field to appear radial at points far from the origin. The second term roughly describes how the angle varies along the velocity lines.

So far, it may seem these choices of trial functions are rather arbitrary. Indeed, to a certain extent, they are. All we desire is to have functions that qualitatively describe the behaviors in Figures 8-1 to 8-3. As we have learned in one dimension, there are plenty of trial functions that can describe an arbitrary behavior as such.

However, we have also learned that for a given set, our minimization algorithm is able to give us the best approximation to the true solution.

Also recall that the success of the minimization routine depends heavily on how close the initial guess is to the true minimum. We were fortunate in the 1D case to have the numerical data. We first fitted some trial function to the numerical solution to check whether the minimization program works or not. Then, we used these fitted parameters for our initial guesses in using the minimum-residue method. This way, we were able to 'lock in' on very good initial guesses. The success of the residue method relies heavily on these astute choices of initial values.

In two dimensions, the most difficult step in fact is to find good initial guess. There is no way to circumvent this but to try out different starting points. The only guideline we have, again, is our intuition, mostly as depicted in those qualitative figures.

There is another observation that proves useful as well. Based on our experience in one dimension, the higher the value of α is, the faster the solution reaches its asymptotic value. Given this, we try to solve the 2D problem with α=2 first. We do this in Section 10.2. We will add parameters to our trial function until the residue does not change significantly, just like what we did in Chapter 6. This way, we are reasonably confident that our trial function is good enough to approximate the behavior of the true solution. In Section 10.3, we use the solution of α=2 for the initial starting point for α=1.

For $\alpha = 0$, just as in one dimension, the solution is quite trivial. Any constant value of n, τ, v, and θ would do. Incorporating the boundary conditions, the solution is therefore

$$n(x,y) = 1 = \tau(x,y) = v(x,y) = 1, \tag{10.15}$$

$$\theta(x,y) = 0. \tag{10.16}$$

10.2 Spherical Case, $\alpha = 2$

As previously explained, finding a good initial value is difficult. For each starting value, our minimization routine produces a result. By plotting this result, we can immediately see whether it agrees with the expected behavior or not. If it does, we then check whether it is indeed a true minimum by using the routine in Appendix D. If so, we then add more parameters until the residue does not change significantly. What we mean by residue here is the sum of $R_n + R_\tau + R_v + R_\theta$.

As in the 1D case, we iterate the minimization procedure among the differential equations. For example, we minimize for the parameters describing τ, while keeping the other parameters fixed. Then, with this new set of τ parameters, we minimize for v parameters. Afterwards, we minimize for n, etc.

Solving the equations simultaneously is more natural, since these equations are coupled. However, the previous approach is more efficient. Iterating lets us keep the number of parameters minimized at any time to be as few as possible. This is more an issue in the 2D case, since we are dealing with more complicated functions; more

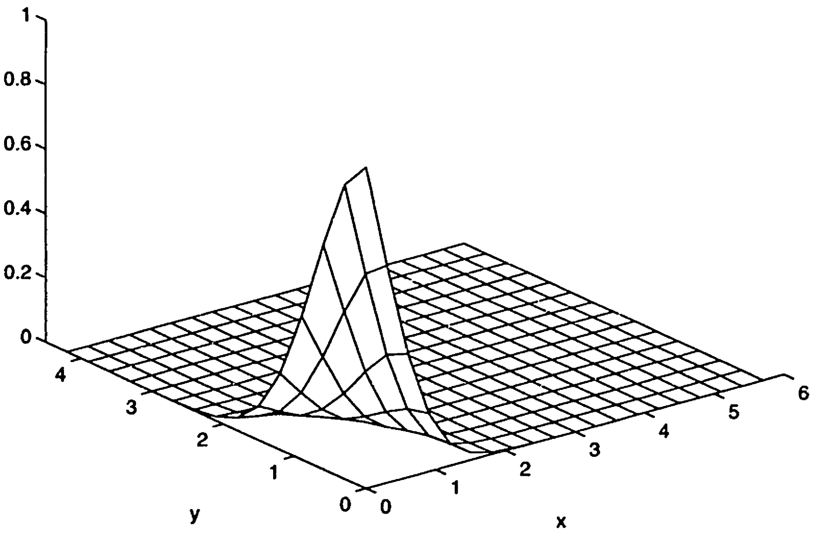

Figure 10-1: Initial trial function for density and temperature for α=2 with $n(x,y) = \tau(x,y) = e^{-(x^2+y^2+x)}$

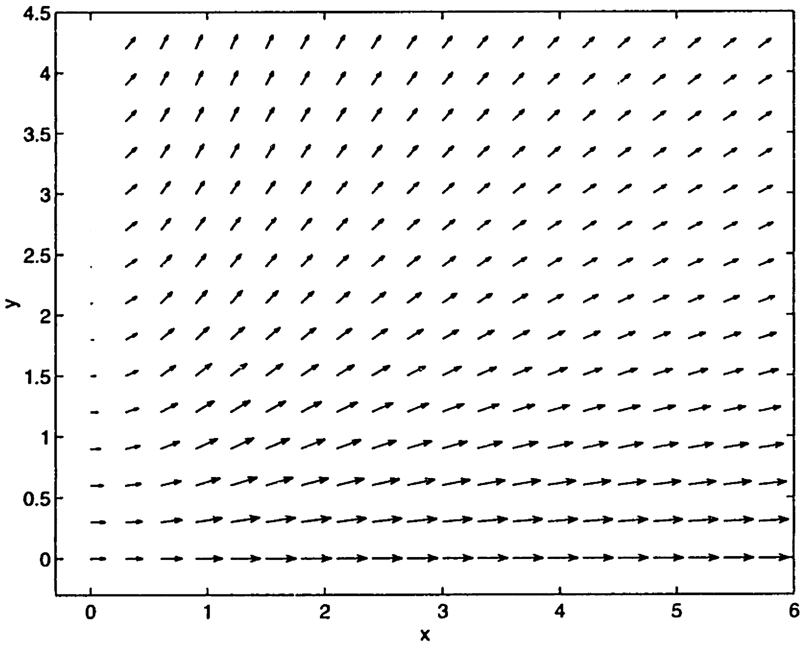

Figure 10-2: Initial trial function for the velocity field for α=2 where $v(x,y) = \sqrt{\tau(0,y)} + 1 - e^{-(x^2+y^2+x)x}$ and $\theta(x,y) = \tan^{-1}(\frac{yx}{1+x})\tanh(x^2+y^2+x)$

N_n	N_τ	N_υ	N_θ	R
4	4	4	6	0.036935
4	4	5	6	0.021333
4	4	6	6	0.021996
4	4	7	6	0.016205
4	4	8	6	0.015750
5	4	8	6	0.015634
6	4	8	6	0.015240
6	5	8	6	0.013446
6	6	8	6	0.013445

Table 10.1: Change in residual, $R = R_n + R_\tau + R_\upsilon + R_\theta$, as more parameters are added for α=2. N_n is the number of parameters describing n, similarly for N_τ, N_υ, and N_θ.

parameters are needed to describe them. For illustration: If we use the simplest set of trial functions, we have 4 parameters each for n, τ, and υ. For θ, we have 6. The total number is 18 parameters. Compare this to the simplest set in one dimension: we had 2 for τ and 3 for υ, for a total of only 5!

For the initial guess, let us use the simplest set of trial functions and assign all the parameters to be 1. We use the same form for both n and τ as plotted in Fig. 10-1. Explicitly, the trial function we use for both density and temperature is

$$n(x,y) = \tau(x,y) = e^{-(x^2+y^2+x)}. \tag{10.17}$$

For the velocity field, we use the followings for our initial guess:

$$\begin{aligned} \upsilon(x,y) &= \upsilon = \sqrt{\tau(0,y)} + 1 - e^{-(x^2+y^2+x)x}, & (10.18)\\ \theta(x,y) &= \tan^{-1}\left(\frac{yx}{1+x}\right)\tanh(x^2+y^2+x). & (10.19) \end{aligned}$$

The field is plotted in Fig. 10-2, where the horizontal and vertical directions are x and y axes, respectively.

Next, we add one extra parameter at a time to the trial functions until the residual does not change by much. Based on our observation in Chapter 6, anything less than 10% change can be deemed insignificant. We only list the resulting residue values as more parameters are added. These residuals are written presented in Table 10.1. N_n denotes the number of parameters describing n, similarly for N_τ, N_υ, and N_θ.

From the table, notice that we add parameters to one particular function at a time until the residual hardly changes. Only then do we add parameter to a different function. In our specific case, we started by making υ more complicated. When the residual changes from 0.158 to 0.157, we stop modifying υ and add new parameter to n instead. Then, we apply the same procedure for modifying n.

The final result is plotted in Figures 10-3 and 10-4. We will use these parameters

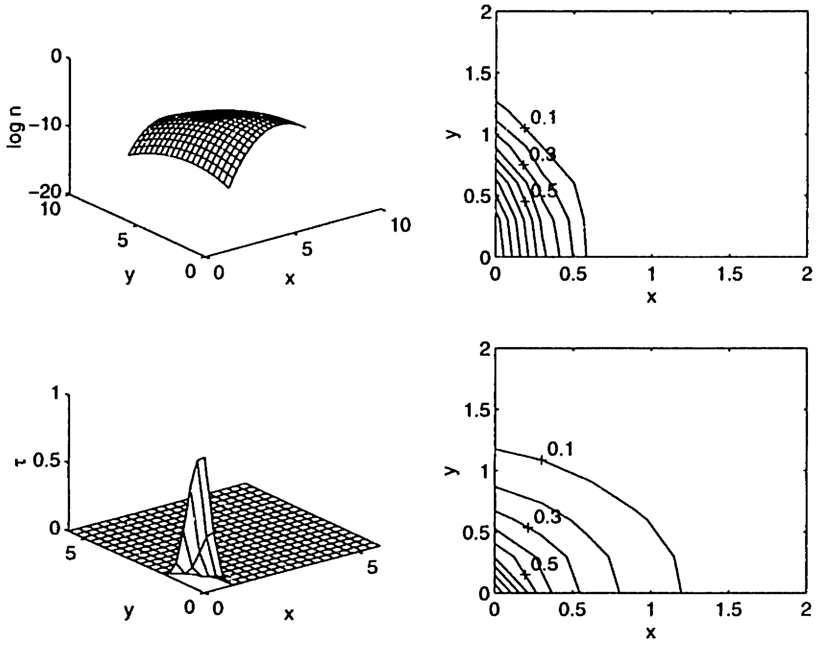

Figure 10-3: Density and temperature functions and their contours for α=2

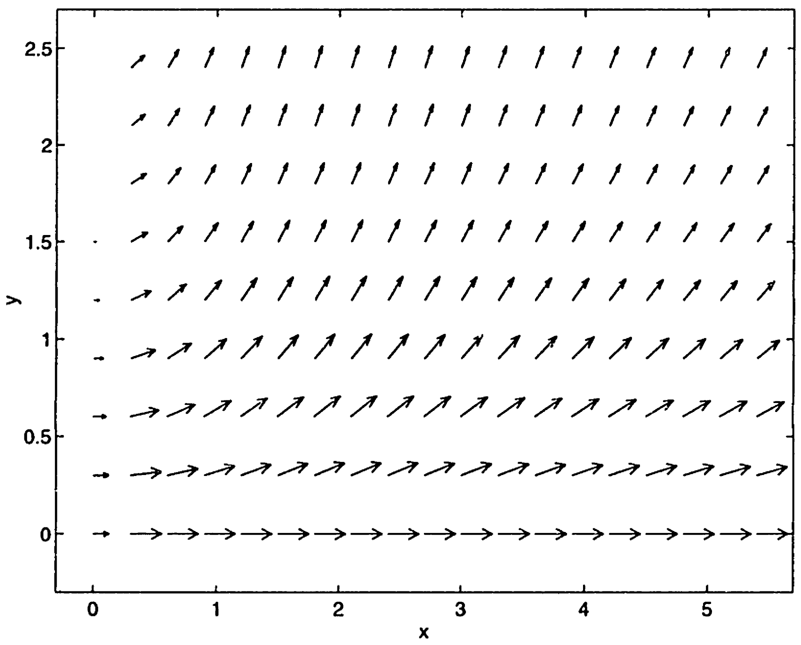

Figure 10-4: Velocity field for α=2

a_1	2.198380	d_1	0.387033	g_1	1.967589	p_1	0.921061
a_2	1.230637	d_2	0.381912	g_2	2.593603	p_2	1.024161
b_o	1.165142	c_o	1.493538	f_o	1.646485	q_o	0.838017
b_1	2.293723	c_1	0.026898	f_1	2.338281	q_1	1.055256
b_2	0.038231	c_2	-0.000486	f_2	0.513358	m	7.290804
b_3	0.254458	c_3	0.000000	f_3	0.140093	k_1	1.159926
b_4	-	c_4	-	f_4	0.140093	q_2	-
b_5	-	c_5	-	f_5	0.056809	k_2	-

Table 10.2: Parameter values for α=2

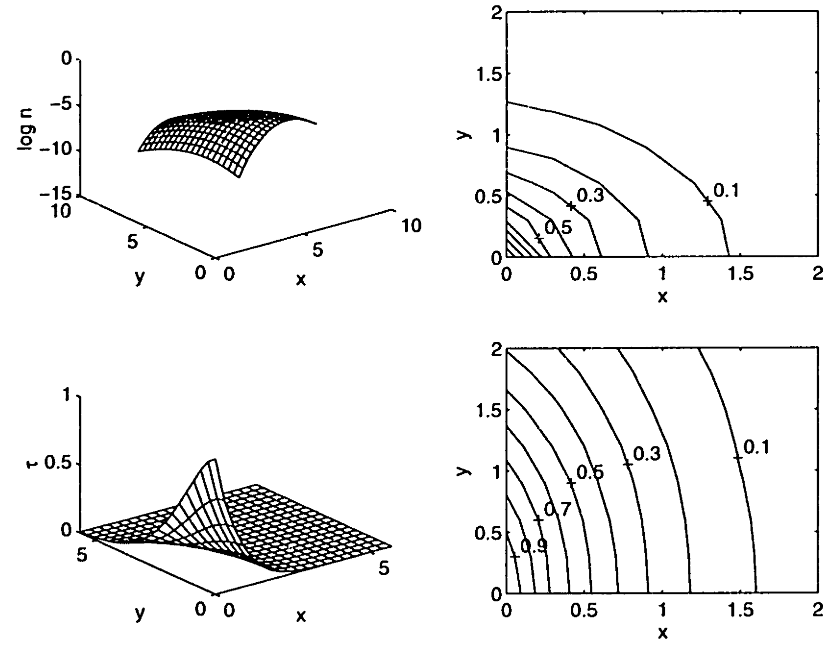

Figure 10-5: Density and temperature and their contours for α=1

for the initial guess for the α=1 case. The parameter values are listed in Table 10.2.

10.3 Cylindrical Case, $\alpha = 1$

We use the result in α=2 as the initial starting point and add more parameters as needed. The change in the residual as the functions are modified is shown in Table 10.3.

The final result is plotted in Figure 10-5 and 10-6. The parameters are tabulated in Table 10.3.

Compared to the α=2 case, the results for α=1 do reach their asymptotic values more slowly. The density and temperature do not decay as fast, the angle of the field does not increase as rapidly.

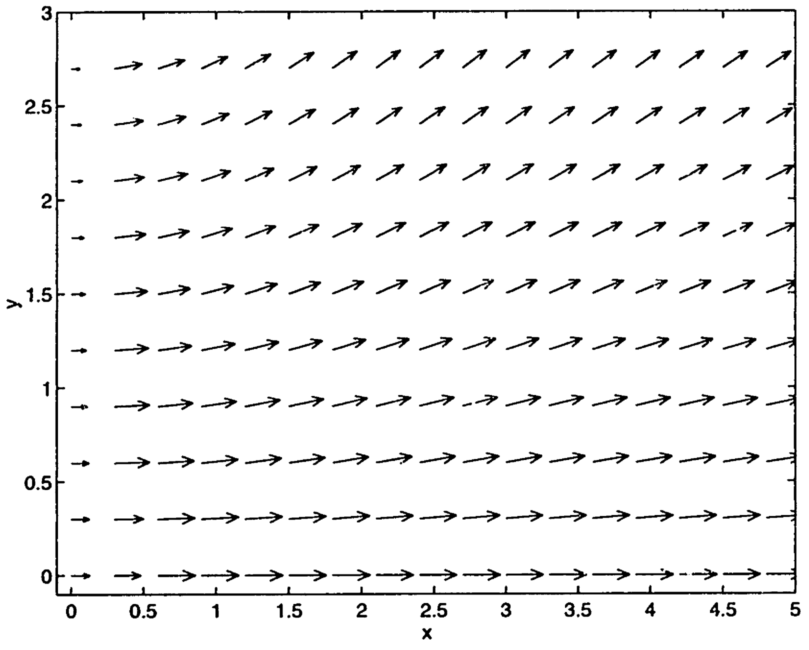

Figure 10-6: Velocity field for α=1

N_n	N_τ	N_v	N_θ	R
6	6	8	6	0.015395
6	6	9	6	0.015204
7	6	9	6	0.015098
7	6	9	7	0.014138
7	6	9	8	0.014075
7	6	10	8	0.014044

Table 10.3: Change in residual as more parameters are added for α=1

a_1	2.478317		d_1	0.384245		g_1	2.160431		p_1	1.014051
a_2	0.541381		d_2	0.381133		g_2	2.901048		p_2	1.165943
b_o	1.192008		c_o	1.491894		f_o	3.306387		q_o	0.318337
b_1	2.866678		c_1	-0.002313		f_1	2.727347		q_1	1.430346
b_2	0.115909		c_2	-0.000500		f_2	0.329353		m	7.198555
b_3	0.657744		c_3	-0.000564		f_3	0.018273		k_1	1.646210
b_4	0.657744		c_4	-		f_4	0.018273		q_2	0.011623
b_5	-		c_5	-		f_5	-0.016139		k_2	0.000995
b_6	-		c_6	-		f_6	0.008345		q_3	-
b_7	-		c_7	-		f_7	0.051116		k_3	-

Table 10.4: Parameter values for α=1

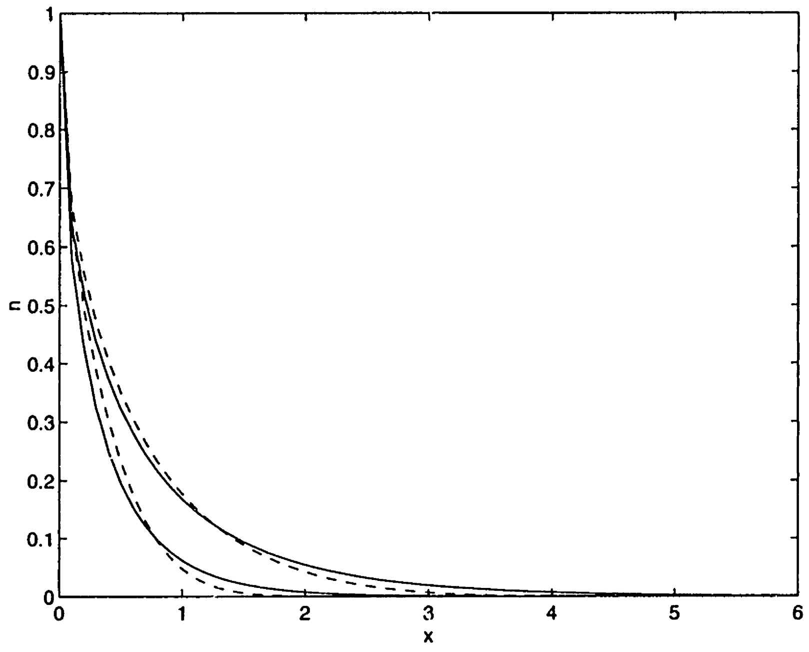

Figure 10-7: Comparison of density 1D solution to 2D solution on y=0 line

10.4 Comparison of Result to 1D Case

It seems that we have succeeded in obtaining the expected results in two dimensions. The solution agrees with the physical intuitions we discussed in Section 8.4. The residue is also of the same order of the one in 1D case, which is around one percent.

It is interesting to directly compare the two solutions. The 2D solution on the y=0 line is expected to behave very similarly to the 1D solution. We present this in the Figures 10-7 to 10-9.

As our intuition dictates, the density in two dimensions decay to lower value, similarly for temperature. This makes sense, since in two dimensions, the plasma has more room to expand to. More work is done for the same distance, hence more energy is lost as well.

As we did in one dimension, we will now modify the value of the thermal conductivity. We present the result in the next chapter.

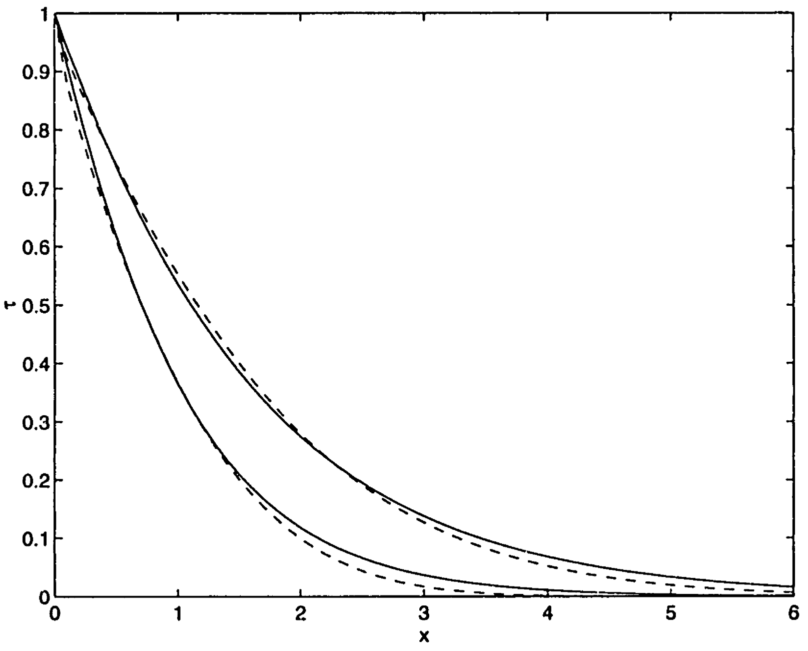

Figure 10-8: Comparison of temperature 1D solution to 2D solution on y=0 line

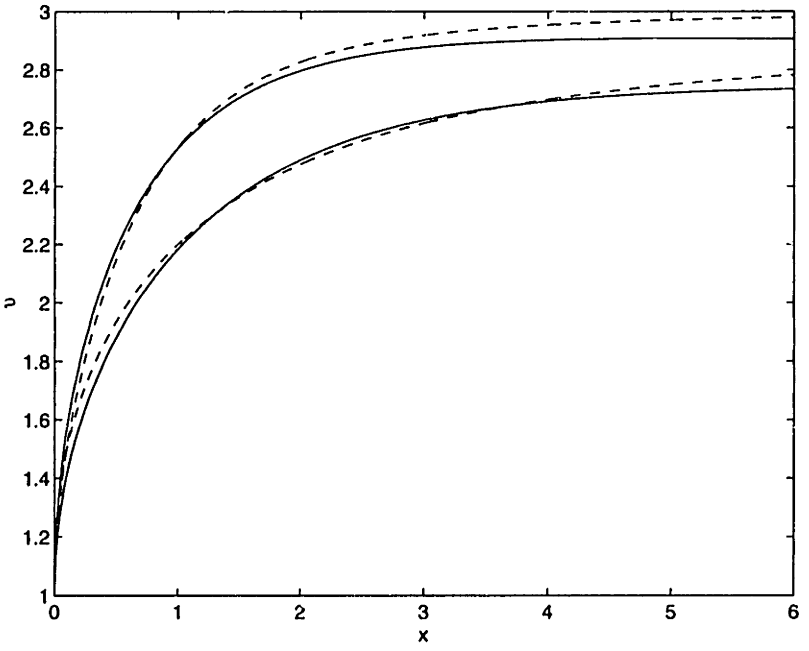

Figure 10-9: Comparison of velocity 1D solution to 2D solution on y=0 line

Chapter 11

Physical Result in Two Dimensions

As we did in one dimension, we now include a more accurate description of κ. Refer to Chapter 7 for more details. As before, we take the case of Ni-like Mo, where Z=14.

With the new form of κ, the only modification will be in Eq. 10.6. Now, A will be a function of x and y. Hence, we have

$$\begin{aligned} R_\tau \;=\;& n\,[\sin\theta(\,v\frac{\partial\tau}{\partial y}+\frac{2}{3}\tau\frac{\partial v}{\partial y}\,)+\frac{2}{3}\tau v\cos\theta\,\frac{\partial\theta}{\partial y}] - \frac{A}{3}[5\tau^{\frac{3}{2}}(\frac{\partial\tau}{\partial y})^2+2\tau^{\frac{5}{2}}\frac{\partial^2\tau}{\partial y^2}] \\ & -\frac{2}{3}\tau^{\frac{5}{2}}\frac{\partial\tau}{\partial y}\frac{\partial A}{\partial y} \\ & + e^{-x}\{n\,[\,\cos\theta\,(\,v\frac{\partial\tau}{\partial x}+\frac{2}{3}\tau\frac{\partial v}{\partial x}+\frac{2}{3}\alpha\tau v\,)\,-\frac{2}{3}\tau v\sin\theta\,\frac{\partial\theta}{\partial x}\,] \\ & -\frac{A}{3}e^{-x}\,[\,5\tau^{\frac{3}{2}}(\frac{\partial\tau}{\partial x})^2+2\tau^{\frac{5}{2}}\frac{\partial^2\tau}{\partial x^2}+2(\alpha-1)\tau^{\frac{5}{2}}\frac{\partial\tau}{\partial x}\,] \\ & -\frac{2}{3}e^{-x}\tau^{\frac{5}{2}}\frac{\partial\tau}{\partial x}\frac{\partial A}{\partial x}\,\}. \end{aligned} \tag{11.1}$$

We use the results from the previous chapter for our initial guesses. Again, we discuss the case for α=2 and α=1 separately.

11.1 Spherical Case, $\alpha = 2$

We present the change of the residual value as we modify the functions in Table 11.1. Refer to Equations 10.9-10.14 for notations. The resulting functions are plotted in Figures 11-1 and 11-2.

11.2 Cylindrical Case, $\alpha = 1$

In Table 11.2, we write down the change of the residual value as more parameters are added. We present the solution functions in Figures 11-3 and 11-4. These figures

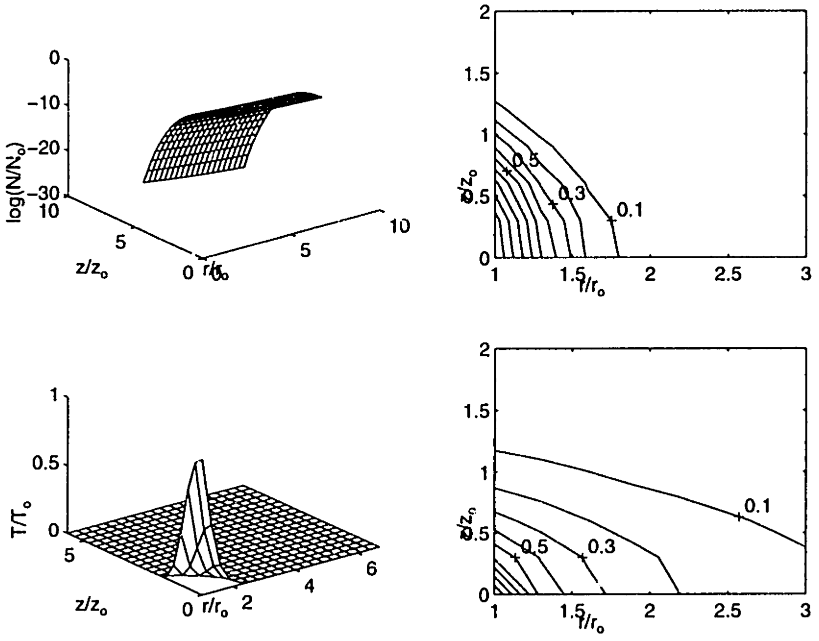

Figure 11-1: 2D density and temperature and their contours for α=2, using physical κ. We use r_o=z_o=20μm, N_o=10^{21}/cm^3, T_o=100eV.

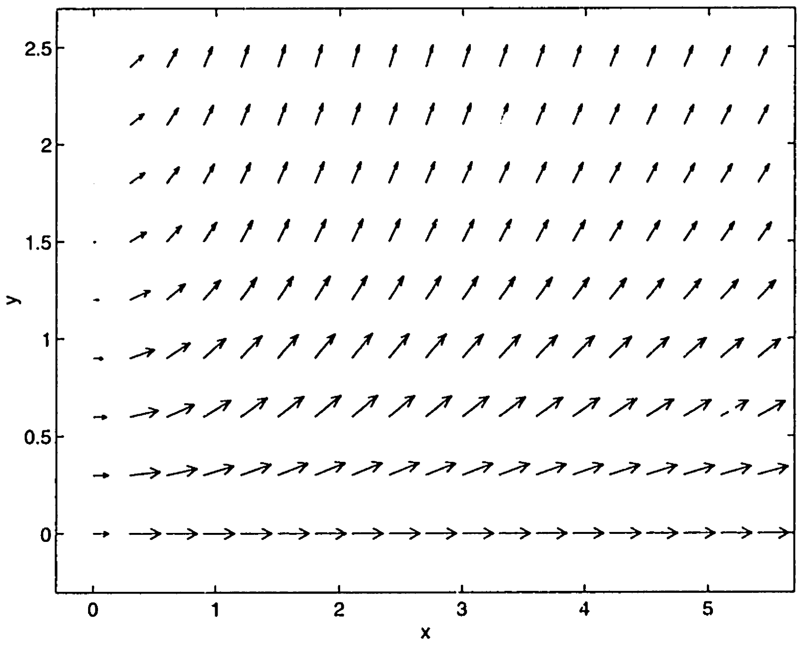

Figure 11-2: Velocity field for α=2, using physical κ

N_n	N_τ	N_v	N_θ	R
6	6	8	6	0.118087
6	7	8	6	0.116611
6	7	9	6	0.115241
6	7	9	7	0.113752
6	7	9	8	0.113305
6	7	9	9	0.033118
6	7	10	9	0.029254
6	7	11	9	0.027106
6	7	11	9	0.027038
6	7	12	9	0.026651
7	7	12	9	0.026266

Table 11.1: Change in residual as more parameters are added for α=2, using physical κ

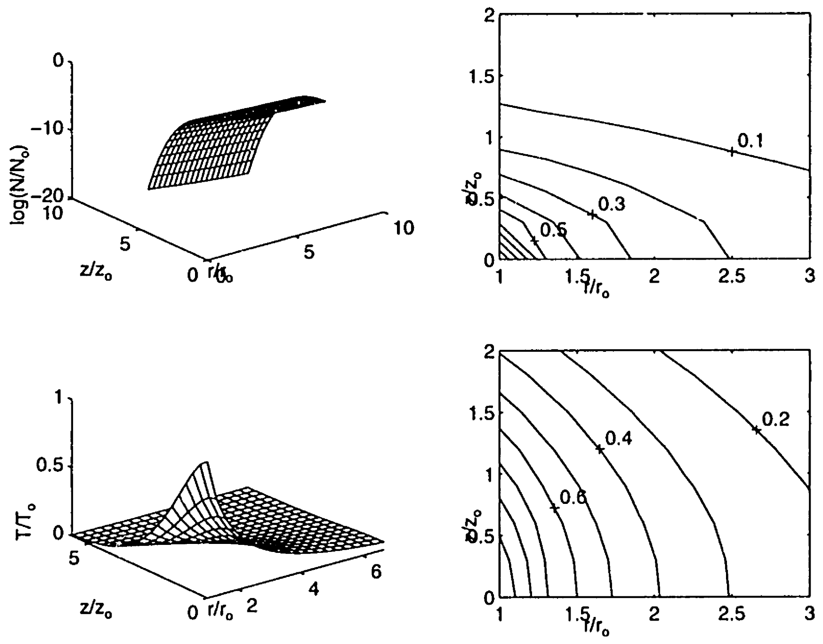

Figure 11-3: 2D density and temperature and their contours for α=1, using physical κ. We use r_o=z_o=20μm, N_o=10^{21}/cm^3, T_o=100eV.

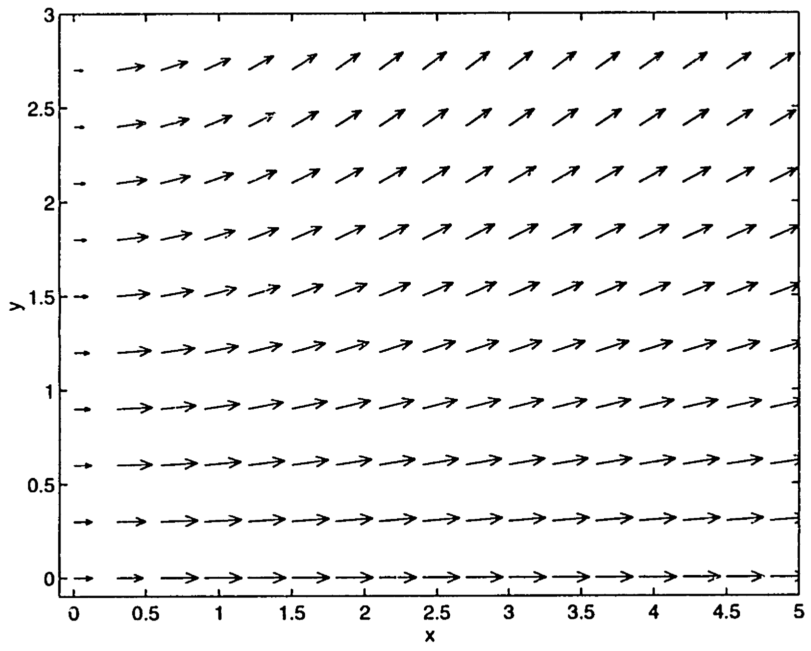

Figure 11-4: Velocity field for α=1, using physical κ

N_n	N_τ	N_v	N_θ	R
7	6	10	8	0.083442
7	6	10	9	0.075441
7	7	10	9	0.070002
7	7	11	9	0.025024
7	7	12	9	0.022024

Table 11.2: Change in residual as more parameters are added for α=1, using physical κ

are in terms of the physical coordinates, r and z.

Similar to the case in one dimension, modifying the value of κ does not significantly change the result obtained previously with A=1; as we can see from comparing the figures.

Chapter 12

Conclusion

12.1 Summary of Results

In this thesis, we computed the density, temperature, and velocity for the plasma ablated from the target in a soft X-ray laser system. We consider the planar, cylindrical, and spherical cases.

We first calculated the functions in one dimension, using both numerical and analytical methods. Both simulations produce results that agree with each other. From the numerical methods, we obtain values at each gridpoint. Meanwhile, the analytical minimum-residue method produces solution in a closed form.

The comparison between the two sets of results can be seen in Figures 5-5 and 5-6. The final results in one dimension are presented in Figures 7-4 and 7-6.

The success of the minimum-residue method in one dimension is especially important. It gave us enough confidence that the same method might work in two dimensions. This is crucial because it is challenging to solve the 2D problem by any numerical procedures.

In two dimensions, we obtained results as depicted in Figures 11-1 to 11-4. From the plots, the results behave as expected. The density function decays faster than the temperature function, similar to the 1D case. Also, the velocity magnitude increases as the temperature decreases, as energy conservation dictates.

As expected, we also observe the solution for the spherical case approaches its asymptotic value sooner than the cylindrical one.

We also need to check whether our results agree with the requirement due to heat flux limiter. The amount of heat that can flow in the system is constrained to be [5]

$$Max\{Q_e\} = N_e v_{th} k T_e, \tag{12.1}$$

where $\frac{1}{2} m {v_{th}}^2 = kT_e$. We need to compare whether the heat flow

$$Q_e = -\kappa \nabla T_e \tag{12.2}$$

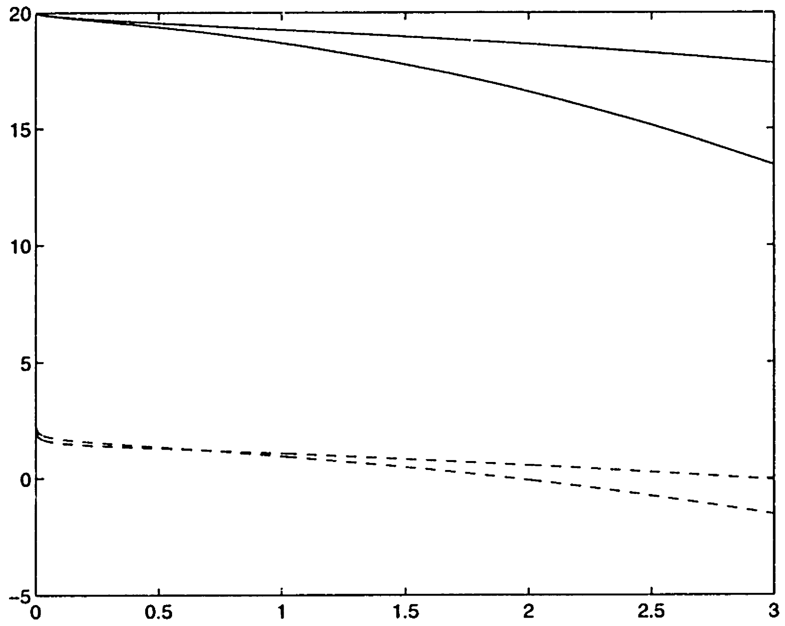

Figure 12-1: comparison of Q_e and $Max\{Q_e\}$. The dotted lines represent Q_e while the solid ones correspond to $Max\{Q_e\}$.

indeed is less than the maximum allowed. In fact, the condition is

$$Q_e < 0.1Max\{Q_e\}, \tag{12.3}$$

where 0.1 is a phenomenological factor. If our solution does not satisfy Eq. 12.3, then we have to modify the thermal conductivity to be

$$\kappa' = \frac{\kappa}{1 + \frac{Q_e}{0.1Max\{Q_e\}}} \tag{12.4}$$

From our solution, we compare the quantities in Eq. 12.3 as presented in Fig. 12-1. The dotted lines represent Q_e while the solid ones correspond to $Max\{Q_e\}$.

12.2 Discussion

The motivation of this thesis project is to understand better the lack of gain observed in the experiment using the MIT table top EUV X-ray laser system. We do so by studying the behavior of plasma ablated from the laser target.

As we have discussed, there are two possible reasons for this lack of gain: a steep density gradient and electron temperature much lower than the estimated 150eV.

From our simulation, the density varies rapidly indeed. Taking $N_o = 10^{21}/\text{cm}^3$, the density gradient is computed to be between 2.0 - 5 4 x $10^{24}/\text{cm}^3$. This is higher than the estimated value of 6.1 x $10^{22}/\text{cm}^3$ in Ref. [1]. The steeper the density gradient is, the more significant the beam refraction is. This refraction reduces the effective

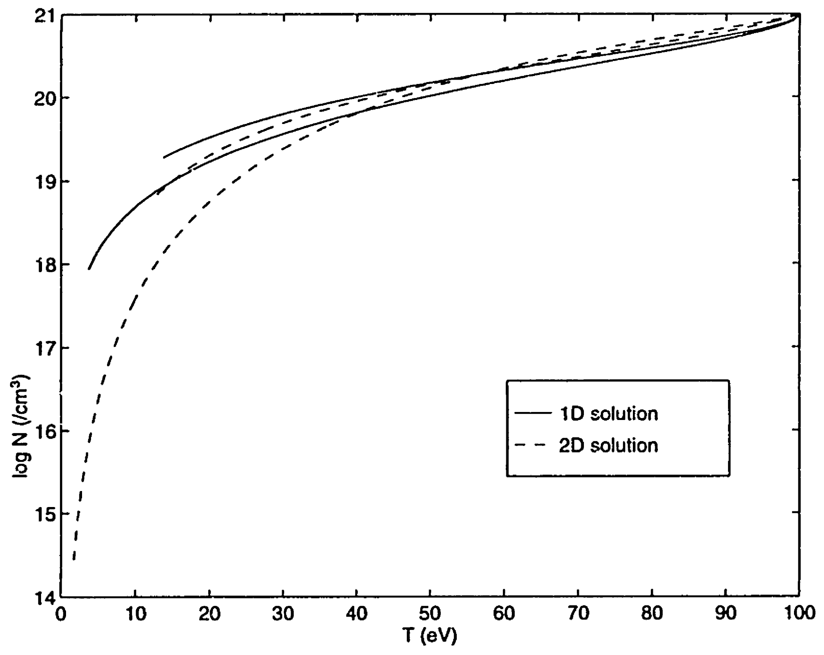

Figure 12-2: Comparison of temperature to density for the 1D and 2D solutions. The shorter curves represent the cylindrical cases, while the long ones are for the spherical cases. We take the solution on y=0 for the 2D case.

length of the plasma column, which in turn decreases the gain-length product.

The large density gradient also decreases the width of the gain region. If this region is too narrow compared to the region observed in the measurement, small gain may be undetected.

In the experiment, we have assumed that the electron temperature does not change much as the density decreases; due to low thermal conductivity. Particularly for density as small as 10^{19}/cm^3. We assumed that the temperature is close to the initial 100eV. However, as the Fig. 12-2 shows, the temperature is considerably lower. Hence, the lack of gain observed.

From the simulation results, we confirm both the steep density gradient and overestimation of electron temperature in the plasma. Thus, these factors contribute to the lack of gain observed in the experiment.

Appendix A

Gaussian Elimination for a Banded Matrix

```
/* gaussBand.c
  solve ax=b
  where a is banded matrix with 'width' non-zero rows/column below/above diagonal
  and b is vector of 'num' size
  gauss-eliminate the matrix and then back substitute */

void gaussBand(double **a, double b[], double x[], int width, int num)
{
 int i, j, k, mid;
 double tmp;
 int n,m;

 mid=width+1;
 for(i=1; i<=num-1; ++i){
  j=1;
  while(j<mid && (i+j)<=num){
   tmp=a[i][mid+j]/a[i][mid];
   b[i+j]-=tmp*b[i];
   k=1;
   while(k<mid && (i+k)<=num){
      a[i+k][mid+j-k]-=tmp*a[i+k][mid-k];
      ++k;
   }
   a[i][mid+j]=0.0;
   ++j;
  }
 }

 /* back substitution */
 x[num]=b[num]/a[num][mid];
 for(i=num-1; i>=1; --i){
  tmp=0.0;
  j=1;
  while(j<mid && i+j<=num){
   tmp+=a[i+j][mid-j]*x[i+j];
```

```
    ++j;
  }
  x[i]=(b[i]-tmp)/a[i][mid];
 }
}
```

Appendix B

Codes for 1D Newton's Method

```
/* hydro1.c
   simultaneously solve both velocity and temperature in 1D by Newton's method
   solve for cylindrical, slab, and spherical configurations

   needs: gaussBand.c
*/

#include <stdio.h>
#include <math.h>
#include "gaussBand.c"

#define num 600
#define tolerance 1.0e-7
#define h .01
#define width 4
#define du .000001
#define dt .000001

#define pi 3.14159
#define t_o 100.0  /* eV */
#define n_o 1.0e21  /* cm-3 */
#define r_o 2.0e-3  /* cm */
#define z 6.0
#define ap 12.0

#define k 1.3807e-16  /* erg per deg K */
#define mp 1.6726e-24 /* g */
#define me 9.1094e-28 /* g */
#define eCharge 4.8032e-10  /* statcoul */

#define evToK 8.6174e-5

/* FUNCTIONS */
int check(double*);
double e(double,double,double,double,double,double,double);
double d(double,double,double,double);
double partialE(int,double,double,double,double,double,double,double);
double partialD(int, double, double, double, double);
```

```
void readFile(double*,double*);
double a(double,double,double);

static double alpha=1.0;
static double v_o, a_o;

int main(void)
{
  int i,j, even, odd, count, numRow, mid;
  double tau[num],u[num], minF[2*num], delta[2*num];
  double **A;
  double xi, tmp,tmp2;
  FILE *look;

  A=(double**)malloc(2*num*sizeof(double*));
  for(i=0; i<2*num; ++i)
    *(A+i)=(double*)malloc(2*(width+1)*sizeof(double));

  /* USEFUL CONSTANTS */
  v_o = sqrt(z*k*t_o/(ap*mp*evToK));
  tmp=2.0/pi;
  tmp2=k*t_o/evToK;
  a_o = 20.0*tmp*sqrt(tmp)*0.095*(z+.24)*tmp2*tmp2*
    sqrt(tmp2)/(z*eCharge*eCharge*eCharge*eCharge*sqrt(me)*(1+.24*z));
  a_o/=(n_o*r_o*v_o);

  /* INITIALIZE MATRIX OF GRAD F TO ZERO */
  for(i=0; i<2*num; ++i)
    for(j=0; j<2*(width+1); ++j)
      A[i][j]=0.0;

  /* DEFINE INITIAL GUESSES OR READ FROM FILE FOR PHYSICAL VALUES */
  /*readFile(u,tau); */

  for(i=0; i<num; ++i){
    xi=i*h;
    tau[i]=pow((1+xi)/(1+xi+.07*tmp+.03*tmp2),14.0);
    u[i]=pow((1+47*xi + 81*tmp + 51*tmp2 )/(1 + 45*xi + 71*tmp + 31*tmp2),9.0);
  }

  for(i=0; i<2*num; ++i)
    delta[i]=1.0;

  numRow=2*num-3;
  mid=width+1;

  /* BEGIN ITERATION */
  count=0;
  while(check(delta)){
    ++count;

    for(i=1; i<num; ++i){
      if(i==1) tmp=exp(h);
      else tmp=tau[i-2];
    even=2*i;
    odd=2*i-1;
    xi=(i-1)*h;

    A[odd][mid]=partialD(2, u[i-1], u[i], tau[i-1], tau[i]);
    A[odd+1][mid-1]=partialD(4, u[i-1], u[i], tau[i-1], tau[i]);
    if(i>1){
        A[odd-1][mid+1]=partialD(3, u[i-1], u[i], tau[i-1], tau[i]);
        A[odd-2][mid+2]=partialD(1, u[i-1], u[i], tau[i-1], tau[i]);
    }
    minF[odd]=-d(u[i-1], u[i],tau[i-1], tau[i]);

    if(odd<numRow){
        A[even][mid]=partialE(5,u[i-1], u[i],tmp,tau[i-1],tau[i],
                                tau[i+1],xi);
        A[even+2][mid-2]=partialE(6,u[i-1], u[i],tmp,tau[i-1],tau[i],
                                tau[i+1],xi);
        A[even-1][mid+1]=partialE(2,u[i-1], u[i],tmp,tau[i-1],tau[i],
                                tau[i+1],xi);
        if(i>1){
          A[even-2][mid+2]=partialE(4,u[i-1], u[i],tmp,tau[i-1],tau[i],
                                tau[i+1],xi);
          A[even-3][mid+3]=partialE(1,u[i-1], u[i],tmp,tau[i-1],tau[i],
                                tau[i+1],xi);
          if(i>2)
            A[even-4][mid+4]=partialE(3,u[i-1], u[i],tmp,tau[i-1],tau[i],
                                tau[i+1],xi);
        }
        minF[even]=-e(u[i-1], u[i],tmp,tau[i-1],tau[i],tau[i+1],xi);
    }
  }

  gaussBand(A,minF,delta,width,numRow);

  for(i=1; i<num; ++i){
    u[i]+=delta[2*i-1];
    if(i<num-1) tau[i]+=delta[2*i];
  }
}


/* PRINT OUT RESULTS */
look=fopen("cylTry", "w+");
for(i=0; i<num: ++i){
  if(i==1) tmp=exp(h);
  else tmp=tau[i-2];
  if(i==0) tmp2=-100;
  else tmp2=d(u[i-1], u[i], tau[i-1], tau[i]);
  fprintf(look, "%f\t%f\t%f\t%f\t%f\n",
          i*h, u[i], tmp2, tau[i],
          e(u[i-1], u[i], tmp, tau[i-1], tau[i], tau[i+1],(i-1)*h));
}
fclose(look);
```

```
  printf("iteration=%d\n", count);

}

/* CHECK CONVERGENCE */
int check(double delta[])
{
  int i;

  i=1;
  while(i<=2*num-3){
    if(fabs(delta[i])>tolerance) return 1;
    ++i;
  }
  return 0;
}

double d(double u1, double u2, double t1, double t2)
{
  return ( (u2*u2-u1*u1)/h - alpha*(t2+t1) + 2*(t2-t1)/h -
              (t2/u2 + t1/u1)*(u2-u1)/h );
}

double e(double u1, double u2, double t0, double t1, double t2,
            double t3,double xi)
{
  double tmp, tmp2, tmp3, tmp4, tmp5,tmp6,tmp7;

  tmp=t1*sqrt(fabs(t1));
  tmp2=t2*sqrt(fabs(t2));
  tmp3=t2-t1;
  tmp4=exp((alpha-1)*xi);
  tmp5=tmp*t1+tmp2*t2;
  tmp6=tmp4*(1.0 + exp(-(alpha-1)*h));
  tmp7=a(u1,t1,xi);

  return ( tmp3/h
              + (t2/u2+t1/u1)*(u2-u1)/(3*h)
              + alpha*(t2+t1)/3
              - tmp7*tmp6*(alpha-1)*tmp5*tmp3/(6*h)
              - 5*tmp7*tmp6*(tmp+tmp2)*tmp3*tmp3/(12*h*h)
              - tmp7*tmp6* tmp5*(t3-t2-t1+t0)/(12*h*h)
              - (a(u1,t1,xi+h)-tmp7)*tmp6*tmp5*tmp3/(6*h*h));
}

double partialD(int j, double u1, double u2, double t1, double t2)
{
  switch(j){
  case 1: return ( d(u1+du, u2, t1, t2)-d(u1-du, u2, t1, t2) )/(2*du);
  case 2: return ( d(u1, u2+du, t1, t2)-d(u1, u2-du, t1, t2))/(2*du);
  case 3: return ( d(u1, u2, t1+dt, t2)-d(u1, u2, t1-dt, t2))/(2*dt);
  default: return ( d(u1, u2, t1, t2+dt)-d(u1, u2, t1, t2-dt))/(2*dt);
  }
}
```

```
double partialE(int i, double u1, double u2, double t0, double t1,
                double t2, double t3,double xi)
{
 switch(i){
 case 1: return(e(u1+du,u2,t0,t1,t2,t3,xi)-e(u1-du,u2,t0,t1,t2,t3,xi))/(2*du);
 case 2: return(e(u1,u2+du,t0,t1,t2,t3,xi)-e(u1,u2-du,t0,t1,t2,t3,xi))/(2*du);
 case 3: return(e(u1,u2,t0+dt,t1,t2,t3,xi)-e(u1,u2,t0-dt,t1,t2,t3,xi))/(2*dt);
 case 4: return(e(u1,u2,t0,t1+dt,t2,t3,xi)-e(u1,u2,t0,t1-dt,t2,t3,xi))/(2*dt);
 case 5: return(e(u1,u2,t0,t1,t2+dt,t3,xi)-e(u1,u2,t0,t1,t2-dt,t3,xi))/(2*dt);
 default:return(e(u1,u2,t0,t1,t2,t3+dt,xi)-e(u1,u2,t0,t1,t2,t3-dt,xi))/(2*dt);
 }
}

void readFile(double u[],double tau[])
{
 FILE *look;
 int i;
 double tmp1,tmp2,tmp3;

 look=fopen("try.dat", "r");
 for(i=0; i<num; ++i)
   fscanf(look,"%lf%lf%lf%lf%lf",&tmp1,&u[i],&tmp2,&tau[i],&tmp3);
 fclose(look);
}


double a(double u, double t, double xi)
{
 double n;

 /* return 1.0 */
 n=exp(-alpha*xi)/u;
 return a_o/(24.0 - log(sqrt(n*n_o)/(t*t_o))/log(exp(1.0)));
}
```

Appendix C

Code for Fitting Numerical Results to Analytical Functions

```
/* minNum.c
   compute chi given the results from hydro1.c */

#include<stdio.h>
#include<math.h>
#include "nr.h"
#include "nrutil.h"

#define NRANSI
#define FTOL 1.0e-6
#define da .00001
#define tol 1.0e-6
#define dxi 0.01
#define num 599
#define h 1.0e-4
#define a 1.0

#define NDIMTau 2
#define NDIM 5

static float resTau(float*);
static void diffResTau(float*,float*);
static float res(float*);
static void diffRes(float*,float*);

static float uTrial(float,float*);
static float tauTrial(float,float*);
static float e(float,float*);
static float d(float,float*);

static float n=1.0;
static float *fix,*fixTau;
/* n=0 for 1D, n=1 for cylindrical, n=2 for spherical */

int main()
```

```
{
 int i,iter,iterTau,count,j;
 float *p,*pTau;
 float new,newTau,old,oldTau;
 float min;

 float *loc,*locTau;
 float *vary,*varyTau;
 float *lo,*loTau;
 float *hi,*hiTau;
 float (*ptr)(float*);

 p=vector(1,NDIM);
 pTau=vector(1,NDIMTau);
 fix=vector(1,NDIM);
 fixTau=vector(1,NDIMTau);

 for(i=1; i<=NDIMTau; ++i)
   pTau[i]=fixTau[i]=1.0;
 for(i=1; i<=NDIM; ++i)
   p[i]=fix[i]=1.0;

 /* PRINT OUT INITIAL VALUES */
 printf("ndim=%d ndimTau=%d n=%f\n", NDIM, NDIMTau, n);
 printf("initial guesses\np=[");
 for(i=1; i<=NDIM; ++i)
   printf("%f ", p[i]);
 printf("]\npTau=[");
 for(i=1; i<=NDIMTau; ++i)
   printf("%f ", pTau[i]);
 printf("]\nres=%f resTau=%f\n",res(p),resTau(pTau));

 /* START ITERATION */
 old=oldTau=10.0;
 new=newTau=0.0;
 count=0;
 printf("\n");
 while(fabs(new-old)>tol || fabs(newTau-oldTau)>tol){
   old=new;
   oldTau=newTau;

   for(i=1; i<=NDIM; ++i)
     fix[i]=p[i];
   frprmn(pTau,NDIMTau,FTOL,&iterTau,&newTau,resTau,diffResTau);
   printf("%d   ",count);

   for(i=1; i<=NDIMTau; ++i)
     fixTau[i]=pTau[i];
   frprmn(p,NDIM,FTOL,&iter,&new,res,diffRes);
   ++count;
 }

 /* PRINT OUT RESULTS */
 printf("\nn=%f\n", n);
```

```
  printf("count=%d newTau=%f new=%f\n",count,newTau,new);
  printf("   pTau=[");
  for(i=1; i<=NDIMTau; ++i)
    printf(" %f", pTau[i]);
  printf("];\n   p=[");
  for(i=1; i<=NDIM; ++i)
    printf(" %f", p[i]);
  printf("];\n");
}
#undef NRANSI

/* CALCULATE R_TAU */
float resTau(float pTau[])
{
  int i;
  float I,sum,t[num],tmp1;
  FILE *tmp;

  tmp=fopen("cylindrical.dat", "r");
  for(i=0; i<num; ++i)
    fscanf(tmp,"%f%f%f%f%f", &tmp1,&tmp1,&tmp1,&t[i],&tmp1);
  fclose(tmp);

  sum=0.0;
  for(i=0; i<num; ++i){
    t[i]=1.0;
    I=(tauTrial(i*dxi,pTau)-t[i]);
    sum+=I*I;
  }
  return sum;
}

/* COMPUTE FIRST DERIVATIVE OF R_TAU */
void diffResTau(float pTau[],float df[])
{
  int j;
  float IJ,IJMod;

  IJ=resTau(pTau);
  for(j=1; j<=NDIMTau; ++j){
    pTau[j]+=da;
    IJMod=resTau(pTau);
    pTau[j]-=da;
    df[j]=(IJMod-IJ)/da;
  }
}

/* COMPUTE R_U */
float res(float p[])
{
  int i;
  float I,sum,u[num],tmp1;
  FILE *tmp;
  tmp=fopen("cylindrical.dat", "r");
  for(i=0; i<num; ++i)
    fscanf(tmp,"%f%f%f%f%f%f", &tmp1,&u[i],&tmp1,&tmp1,&tmp1,&tmp1);
  fclose(tmp);

  sum=0.0;
  for(i=0; i<num; ++i){
    u[i]=1.0;
    I=(uTrial(i*dxi,p)-u[i]);
    sum+=I*I;
  }
  return sum;
}

/* CALCULATE FIRST DERIVATIVE OF R_U */
void diffRes(float p[],float df[])
{
  int j;
  float IJ,IJMod;

  IJ=res(p);
  for(j=1; j<=NDIM; ++j){
    p[j]+=da;
    IJMod=res(p);
    p[j]-=da;
    df[j]=(IJMod-IJ)/da;
  }
}

/* TRIAL FUNCTION FOR VELOCITY */
float uTrial(float x, float p[])
{
  /* x=pow(x,fabs(p[2]));
  return 1.0 + p[3]*(1.0 - exp(-fabs(p[1])*x));*/

  float numer,denom;

  numer=p[2]*x + p[4]*x*x;
  denom=1.0 + p[3]*x + p[5]*x*x;

  return 1.0 + pow(fabs(numer/denom),fabs(p[1]));
}

/* TRIAL FUNCTION FOR TEMPERATURE */
float tauTrial(float x, float pTau[])
{
  /* x=pow(x,fabs(pTau[2]));
  return exp(-fabs(pTau[1])*x);*/

  return (1.0+pTau[2])/(1.0 + pTau[2]*exp(fabs(pTau[1])*x));
}
```

Appendix D

Code to Map the Parameter Space

```
/* map.c
  - recursively compute the value of f(vary) in the 2D region
  of 'lo<=vary<=hi' for each element in the array
  - 'vary' increases each time by the amount of 'del'
  - return the minimum value of f(vary)
  */

float map(int dim, int which, float vect[], float (*f)(float*),
          float min, float tmp[], float vary[], float lo[], float hi[])
{
 int j;
 float residue,del,answer;

 vary[which]=lo[which];
 del=(hi[which]-lo[which])/pt;  /* pt is #defined */

 if(which==dim){
   while(lo[which]<=vary[which] && vary[which]<=hi[which]){
     residue=f(vary);
     if(residue<min){
          min=residue;
          for(j=1; j<=dim; ++j)
            tmp[j]=vary[j];
     }
     vary[which]+=del;
   }
   return min;
 }
 else if(which==1){
   while(lo[which]<=vary[which] && vary[which]<=hi[which]){
          min=map(dim,which+1,vect,f,min,tmp,vary,lo,hi);
     vary[which]+=del;
   }
 }
 else{
   while(lo[which]<=vary[which] && vary[which]<=hi[which]){
     min=map(dim,which+1,vect,f,min,tmp,vary,lo,hi);
```

```
      vary[which]+=del;
    }
  }

  return min;
}
```

Appendix E

Minimizing Residue by Newton's Method

```
/* min.c */

#include<stdio.h>
#include<math.h>
#include "nr.h"
#include "nrutil.h"

#define NRANSI
#define FTOL 1.0e−6
#define da .001
#define tol 1.0e−6
#define dxi 0.03
#define num 100
#define h .01

#define NDIMTau 2
#define NDIM 5

#define pi 3.14159
#define t_o 100.0  /* eV */
#define n_o 1.0e21  /* cm−3 */
#define r_o 2.0e−3  /* cm */
#define z 6.0
#define ap 12.0

#define k 1.3807e−16  /* erg per deg K */
#define mp 1.6726e−24 /* g */
#define me 9.1094e−28 /* g */
#define eCharge 4.8032e−10  /* statcoul */

#define evToK 8.6174e−5

static float resTau(float*);
static void diffResTau(int,float*,float*);
static float res(float*);
static void diffRes(int,float*,float*);

static float uTrial(float,float*);
static float tauTrial(float,float*);
static float e(float,float*);
static float d(float,float*);

static float a(float,float,float);


static void show(float*,float*);
static void guess(float*,float*);

static float *fix,*fixTau;
static float n;
/* n=0 for 1D, n=0 for cylindrical, n=2 for spherical, etc */
static float v_o, a_o;


int main()
{
  int i,count,j,check;
  float *p,*pTau, *df;
  float new,newTau,old,oldTau;
  float min,tmp,tmp2,xi;

  p=vector(1,NDIM);
  pTau=vector(1,NDIMTau);
  fix=vector(1,NDIM);
  fixTau=vector(1,NDIMTau);


  /* USEFUL CONSTANTS */
  v_o = sqrt(z*k*t_o/(ap*mp*evToK));
  tmp=2.0/pi;
  tmp2=k*t_o/evToK;
  a_o = 20.0*tmp*sqrt(tmp)*0.095*(z+.24)*tmp2*tmp2*sqrt(tmp2)/(z*eCharge*eCharge*eCharge*eCharge*sqrt(me
  a_o/=(n_o*r_o*v_o);

  for(i=1; i<=NDIMTau; ++i)
    pTau[i]=fixTau[i]=1.0;
  for(i=1; i<=NDIM; ++i)
    p[i]=fix[i]=1.0;

  n=2.0;

  guess(pTau,p);

 for(i=1; i<=NDIMTau; ++i)
    fixTau[i]=pTau[i];
  for(i=1; i<=NDIM; ++i)
    fix[i]=p[i];

  printf("initial guesses");
 show(pTau,p);

 old=oldTau=10.0;
 new=newTau=0.0;
 count=0;
 while(fabs(new-old)>tol || fabs(newTau-oldTau)>tol){
  old=new;
  oldTau=newTau;

  newt(pTau,NDIMTau,&check,diffResTau);
  for(i=1; i<=NDIMTau; ++i)
   fixTau[i]=pTau[i];
  newTau=resTau(pTau);

  newt(p,NDIM,&check,diffRes);
  for(i=1; i<=NDIM; ++i)
   fix[i]=p[i];
  new=res(p);

  ++count;

  printf("\ncount=%d newTau=%f new=%f",count,newTau,new);
  show(pTau,p);
 }

 printf("\nfinal result\n");
 show(pTau,p);

}
#undef NRANSI


/* ============================================== */
void guess(float pTau[], float p[])
{
 int i;
 FILE *init_p;

 init_p=fopen("init_p.dat", "r");
 for(i=1; i<=NDIMTau; ++i)
  fscanf(init_p, "%f", &pTau[i]);
 for(i=1; i<=NDIM; ++i)
  fscanf(init_p, "%f", &p[i]);
 fclose(init_p);
}

void show(float pTau[], float p[])
{
 int i;
 float *df;

 printf("\npTau=[");
 for(i=1; i<=NDIMTau; ++i)
  printf("%f ", pTau[i]);
  printf("]\np=[");
  for(i=1; i<=NDIM; ++i)
    printf("%f ", p[i]);
  printf("]\n");

  df=vector(1,NDIMTau);
  diffResTau(NDIMTau,pTau,df);
  printf("dfTau=[");
  for(i=1; i<=NDIMTau; ++i)
    printf("%f ", df[i]);
  printf("]\n");
  free_vector(df,1,NDIMTau);

  df=vector(1,NDIM);
  diffRes(NDIM,p,df);
  printf("df=[");
  for(i=1; i<=NDIM; ++i)
    printf("%f ", df[i]);
  printf("]\n");
  free_vector(df,1,NDIM);

  printf("restau=%f res=%f\n", resTau(pTau), res(p));
}



float resTau(float pTau[])
{
  int i;
  float I,sum;

  sum=0.0;
  for(i=1; i<num; ++i){
    I = e((i+1)*dxi,pTau);
    sum+=I*I;
  }
  return dxi*sum;
}

float res(float p[])
{
  int i;
  float I,sum;

  sum=0.0;
  for(i=1; i<num; ++i){
    I=d((i+1)*dxi,p);
    sum+=I*I;
  }
  return dxi*sum;
}

void diffResTau(int dim,float pTau[],float df[])
{
  int j;
  float IJ,IJMod;

  IJ=resTau(pTau);
  for(j=1; j<=dim; ++j){
    pTau[j]+=da;
    IJMod=resTau(pTau);
    pTau[j]-=da;
    df[j]=(IJMod-IJ)/da;
    /*if(df[j]>=2.0) df[j]*=.2;*/
  }
}

void diffRes(int dim, float p[],float df[])
{
  int j;
  float IJ,IJMod;

  IJ=res(p);
  for(j=1; j<=dim; ++j){
    p[j]+=da;
    IJMod=res(p);
    p[j]-=da;
    df[j]=(IJMod-IJ)/da;
    /*if(df[j]>=2.0) df[j]*=.2;*/
  }
}

float e(float x, float pTau[])
{
  float t0,t1,t2,t3,u1,u2;
  float tmp,tmp2,tmp3,tmp4,tmp5,tmp6,tmp7;

  t0=tauTrial(x-2*h,pTau);
  t1=tauTrial(x-h,pTau);
  t2=tauTrial(x,pTau);
  t3=tauTrial(x+h,pTau);

  u1=uTrial(x-h,fix);
  u2=uTrial(x,fix);

  tmp=t1*sqrt(fabs(t1));
  tmp2=t2*sqrt(fabs(t2));
  tmp3=t2-t1;
  tmp4=exp((n-1)*(x-.5*h));
  tmp5=tmp*t1+tmp2*t2;
  tmp6=tmp4*(1.0 + exp(-(n-1)*h));
  tmp7=a(u1,t1,x);

  return ( tmp3/h
             + (t2/u2+t1/u1)*(u2-u1)/(3.0*h)
             + n*(t2+t1)/3
             - tmp7*tmp6*(n-1)*tmp5*tmp3/(6.0*h)
             - 5.0*tmp7*tmp6*(tmp+tmp2)*tmp3*tmp3/(12.0*h*h)
```

```
        - tmp7*tmp6* tmp5*(t3-t2-t1+t0)/(12.0*h*h)
        - (a(u1,t1,x+h)-tmp7)*tmp6*tmp5*tmp3/(6*h*h));
}

float d(float x, float p[])
{
 float u1,u2,t1,t2;

 u1=uTrial(x-h,p);
 u2=uTrial(x,p);
 t1=tauTrial(x-h,fixTau);
 t2=tauTrial(x,fixTau);

 return ( (u2*u2-u1*u1)/h - n*(t2+t1) + 2*(t2-t1)/h - (t2/u2 + t1/u1)*(u2-u1)/h );
}

float a(float u, float t, float xi)
{
 float den;

 /* den=exp(-n*xi)/u;
 return a_o/(24.0 - log(sqrt(den*n_o)/(t*t_o))/log(exp(1.0))); */

 return 1.0;
}

float uTrial(float x, float p[])
{
 float numer,denom;
 numer=p[2]*x + p[4]*x*x;
 denom=1.0 + p[3]*x + p[5]*x*x;

 return 1.0 + pow(fabs(numer/denom),fabs(p[1]));
}

float tauTrial(float x, float pTau[])
{
 float tmp;
 tmp=exp(fabs(pTau[1])*x);

 return (1.0+pTau[2])/(1.0 + pTau[2]*tmp);
}
```

Appendix F

Minimum Residue in Two Dimensions

```
/* theta.c */

/* #include'S */
#include <stdio.h>
#include <math.h>
#include "nr.h"
#include "nrutil.h"

/* #define'S */
#define NRANSI
#define pi 3.14159
#define ftol 1.0e-3
#define tol 0.2
#define dx .3
#define dy .3
#define h .1
#define da 1.0e-3
#define a 1.0

#define numx 15
#define numy 15

#define seenumx 20
#define seenumy 15

#define dim_n 4
#define dim_tau 5
#define dim_u 4
#define dim_theta 6

/* GLOBALS */
static float v_o, a_o;

static float alpha=2.0;
static float *p_n,*p_u,*p_tau,*p_theta;
```

```
/* alpha=0 -> slab, alpha=1 -> cylindrical */


/* PROTOTYPES */
static float sum_res2_n(float*);
static void diff_sum_res2_n(float*,float*);
static float n_trial(float,float,float*);

static float sum_res2_u(float*);
static void diff_sum_res2_u(float*,float*);
static float u1(float,float,float*);

static float sum_res2_theta(float*);
static void diff_sum_res2_theta(float*,float*);
static float theta_trial(float,float,float*);


static float sum_res2_tau(float*);
static void diff_sum_res2_tau(float*,float*);
static float tau_trial(float,float,float*);

static float res(float,float,int);


static void see(float*,float*,float*,float*);
static void printfile(float*,FILE*);
static void guess(float*,float*,float*,float*);
static void showParameters(float*, float*, float*,float*,float*,float*,float*,float*);

static float brute(float*,float(float*),int);

static int check(float*,float*,int);

static float minimize(float*,float(),int);

/* MAIN */
int main()
{
 int count,iter_tau,iter_n,iter_u,iter_theta;
 float new_n,new_u,new_tau,new_theta,old_n,old_u,old_tau,old_theta;

 float *min_p,min,tmp;
 float *p_old_n,*p_old_tau,*p_old_u,*p_old_theta;
 int i,j,k;
 min_p=vector(1,dim_tau);

 p_n=vector(1,dim_n);
 p_u=vector(1,dim_u);
 p_tau=vector(1,dim_tau);
 p_theta=vector(1,dim_theta);

 p_old_n=vector(1,dim_n);
 p_old_tau=vector(1,dim_tau);
 p_old_u=vector(1,dim_u);
```

```
p_old_theta=vector(1,dim_theta);

/* USEFUL CONSTANTS */
v_o = sqrt(z*k*t_o/(ap*mp*evToK));
tmp=2.0/pi;
tmp2=k*t_o/evToK;
a_o = 20.0*tmp*sqrt(tmp)*0.095*(z+.24)*tmp2*tmp2*sqrt(tmp2)/(z*eCharge*eCharge*eCharge*eCharge*sqrt(me
a_o/=(n_o*r_o*v_o);



/* read initial guesses from a file */
guess(p_n,p_tau,p_u,p_theta);

/* print out initial guesses and related residues to check */
printf("from initial guesses alpha=%f\n", alpha);
showParameters(p_n,p_tau,p_u,p_theta,&new_n,&new_tau,&new_u,&new_theta);

/* plot from initial guesses */
see(p_n,p_tau,p_u,p_theta);

for(i=1; i<=dim_n; ++i)
  p_old_n[i]=0.0;
for(i=1; i<=dim_tau; ++i)
  p_old_tau[i]=0.0;
for(i=1; i<=dim_u; ++i)
  p_old_u[i]=0.0;
for(i=1; i<=dim_theta; ++i)
  p_old_theta[i]=0.0;

count=1;
old_u=old_tau=old_n=old_theta=0.0;

while(check(p_n,p_old_n,dim_n) || check(p_tau,p_old_tau,dim_tau) || check(p_u,p_old_u,dim_u) ||
          check(p_theta,p_old_theta,dim_theta)){

  for(i=1; i<=dim_n; ++i)
    p_old_n[i]=p_n[i];
  for(i=1; i<=dim_tau; ++i)
    p_old_tau[i]=p_tau[i];
  for(i=1; i<=dim_u; ++i)
    p_old_u[i]=p_u[i];
  for(i=1; i<=dim_theta; ++i)
    p_old_theta[i]=p_theta[i];

  frprmn(p_tau,dim_tau,ftol,&iter_tau,&new_tau,sum_res2_tau,
           diff_sum_res2_tau);
  printf("-tau-", new_tau);
  frprmn(p_theta,dim_theta,ftol,&iter_theta,&new_theta,sum_res2_theta,
           diff_sum_res2_theta);
  printf("-theta-");
  frprmn(p_n,dim_n,ftol,&iter_n,&new_n,sum_res2_n,
           diff_sum_res2_n);
  printf("-n-", new_n);
```

```
    frprmn(p_u,dim_u,ftol,&iter_u,&new_u,sum_res2_u,
            diff_sum_res2_u);
    printf("-u-", new_u);

    printf("\n");
    showParameters(p_n,p_tau,p_u,p_theta,&new_n,&new_tau,&new_u,&new_theta);
    see(p_n,p_tau,p_u,p_theta);
    printf("%d done for matlab\n\n", count, old_u, new_u);

    ++count;
  }
}
#undef NRANSI



/* FUNCTION DEFINITIONS */

int check(float *p, float *old, int dim)
{
  int i;
  float tmp;

  i=1;
  while(i<=dim){
    if(fabs(old[i])<1.0e-6) old[i]=0.0;
    if(fabs(p[i])<1.0e-6) p[i]=0.0;
    if(old[i]==0.0 && p[i]==0.0) tmp=0.0;
    else tmp=fabs(p[i]-old[i])/fabs(old[i]);
    if(tmp > 0.2) return 1;
      ++i;
  }
  return 0;
}


float brute(float p[], float func(float*), int dim)
{
  int i,j,k;
  float *tmp,min,chk;

  tmp=vector(1,dim);

  for(i=1; i<=dim; ++i)
    tmp[i]=p[i];

  for(i=1; i<=dim; ++i){
    min=func(tmp);
    for(j=1; j<=10; ++j){
      tmp[i]+=0.5;
      chk=func(tmp);
      if(chk<min){
            min=chk;
            p[i]=tmp[i];
      }
    }
  }
  return min;
}



/* ==========================================================
   functions to be minimized
   ======================================================== */

float sum_res2_n(float p_n[])
{
  int i,j;
  float sum,I,norm,x,y;

  sum=0.0;
  for(i=0; i<numx; ++i){
    x=i*dx;
    for(j=0; j<numy; ++j){
      y=j*dy;
      I=res(x,y,1);
      sum+=I;
    }
  }
  return sum;
}

float sum_res2_u(float p_u[])
{
  int i,j;
  float sum,I,x,y;

  sum=0.0;
  for(i=0; i<numx; ++i){
    x=i*dx;
    for(j=0; j<numy; ++j){
      y=j*dy;
      I=res(x,y,3);
      sum+=I;
  }
    }
  return sum;
}

float sum_res2_tau(float p_tau[])
{
  int i,j;
  float sum,I,x,y;

  sum=0.0;
  for(i=0; i<numx; ++i){
    x=i*dx;
    for(j=0; j<numy; ++j){
      y=j*dy;
      I=res(x,y,2);
      sum+=I;
    }
  }
  return sum;
}

float sum_res2_theta(float p_theta[])
{
  int i,j;
  float sum,I,norm,x,y;

  sum=0.0;
  for(i=0; i<numx; ++i){
    x=i*dx;
    for(j=0; j<numy; ++j){
      y=j*dy;
      I=res(x,y,4);
      sum+=I;
    }
  }
  return sum;
}

/* ==============================================================
   first derivatives of functions to be minimized
   ============================================================== */

void diff_sum_res2_n(float p_n[],float df[])
{
  int i;
  float I2,I2mod;

  I2=sum_res2_n(p_n);
  for(i=1; i<=dim_n; ++i){
    p_n[i]+=da;
    I2mod=sum_res2_n(p_n);
    p_n[i]-=da;
    df[i]=(I2mod-I2)/da;
  }
}

void diff_sum_res2_u(float p_u[],float df[])
{
  int i;
  float I2,I2mod;

  I2=sum_res2_u(p_u);
  for(i=1; i<=dim_u; ++i){
    p_u[i]+=da;
    I2mod=sum_res2_u(p_u);
    p_u[i]-=da;
   df[i]=(I2mod−I2)/da;
  }
}

void diff_sum_res2_tau(float p_tau[],float df[])
{
  int i;
  float I2,I2mod;

  I2=sum_res2_tau(p_tau);
  for(i=1; i<=dim_tau; ++i){
    p_tau[i]+=da;
    I2mod=sum_res2_tau(p_tau);
    p_tau[i]−=da;
    df[i]=(I2mod−I2)/da;
  }
}

void diff_sum_res2_theta(float p_theta[],float df[])
{
  int i;
  float I2,I2mod;

  I2=sum_res2_theta(p_theta);
  for(i=1; i<=dim_theta; ++i){
    p_theta[i]+=da;
    I2mod=sum_res2_theta(p_theta);
    p_theta[i]−=da;
    df[i]=(I2mod−I2)/da;
  }
}



/* ==========================================================
   print data to files for plotting in matlab
   concatenate these files and plot.m before use
   ========================================================== */
void printfile(float f[], FILE *result)
{
  int i,j,k;
  float f2d[seenumx][seenumy];

  k=0;
  for(i=0; i<seenumx; ++i)
    for(j=0; j<seenumy; ++j){
      f2d[i][j]=f[k];
      ++k;
    }

  for(j=0; j<seenumy; ++j){
    for(i=0; i<seenumx; ++i)
      fprintf(result,"%f  ", f2d[i][j]);
    if(j<seenumy−1) fprintf(result, ";\n");
  }
  fprintf(result,"];\n\n");

}


void see(float p_n[], float p_tau[], float p_u[], float p_theta[])
{
  int i,j,k;
  float x,y;
  float n[seenumx*seenumy], tau[seenumx*seenumy], u[seenumx*seenumy], theta[seenumx*seenumy];
  FILE *result, *try;
  float rn[seenumx*seenumy], rt[seenumx*seenumy], ru[seenumx*seenumy], rth[seenumx*seenumy];
  k=0;

  for(i=0; i<seenumx; ++i)
    for(j=0; j<seenumy; ++j){
      x=i*dx;
      y=j*dy;
      n[k]=n_trial(x,y,p_n);
      tau[k]=tau_trial(x,y,p_tau);
      u[k]=u1(x,y,p_u);
      theta[k]=theta_trial(x,y,p_theta);
      rn[k]=res(x,y,1);
      ru[k]=res(x,y,3);
      rt[k]=res(x,y,2);
      rth[k]=res(x,y,4);

      ++k;
   }


 result=fopen("try2.m", "w+");

 fprintf(result, "n=[");
 printfile(n,result);

 fprintf(result, "tau=[");
 printfile(tau,result);

 fprintf(result, "u=[");
 printfile(u,result);

 fprintf(result, "theta=[");
 printfile(theta,result);


 fprintf(result, "rn=[");
 printfile(rn,result);

 fprintf(result, "rt=[");
 printfile(rt,result);

 fprintf(result, "ru=[");
```

```
  printfile(ru,result);

  fprintf(result, "rth=[");
  printfile(rth,result);


  /* matlab commands */
  fprintf(result, "[x,y]=meshgrid(0.0:%f:%f, 0:%f:%f);\n", dx, /*1.0+*/(seenumx-1.0)*dx, dy, (seenumy-1)*d
  fprintf(result, "figure(1)\n");
  fprintf(result, "subplot(2,2,1), mesh(x,y,n), xlabel 'x', ylabel 'y', zlabel 'n';\n");
  fprintf(result, "axis([0.0 %f 0.0 %f 0.0 1.0])\n;", (seenumx-1)*dx, (seenumy-1)*dy);
  fprintf(result, "subplot(2,2,2), contour(x,y,n), xlabel 'x', ylabel 'y';\n");
  fprintf(result, "axis([0.0 %f 0.0 %f])\n;", (seenumx-1)*dx, (seenumy-1)*dy);

  fprintf(result, "subplot(2,2,3), mesh(x,y,tau), xlabel 'x', ylabel 'y';\n");
  fprintf(result, "set (gca,'FontName','Symbol'),zlabel 't';\n");
  fprintf(result, "set (gca,'FontName', 'Helvetica');\n");
  fprintf(result, "axis([0.0 %f 0.0 %f 0.0 1.0])\n;", (seenumx-1)*dx, (seenumy-1)*dy);
  fprintf(result, "subplot(2,2,4), contour(x,y,tau), xlabel 'x', ylabel 'y';\n");
  fprintf(result, "axis([0.0 %f 0.0 %f])\n;", (seenumx-1)*dx, (seenumy-1)*dy);

  fprintf(result, "figure(2)\n quiver(x,y,u.*cos(theta), u.*sin(theta));\n");
  fprintf(result, "xlabel 'x', ylabel 'y';\n");
  fprintf(result, "axis([%f %f %f %f])\n;", -dx, (seenumx-1)*dx, -dy, (seenumy-1)*dy);
  fclose(result);
}


/* ================================================================
   read initial guesses from file
   ================================================================ */
void guess(float p_n[], float p_tau[], float p_u[], float p_theta[])
{
  int i;
  FILE *init_p;

  init_p=fopen("init_p.dat", "r");
  for(i=1; i<=dim_n; ++i)
    fscanf(init_p, "%f", &p_n[i]);
  for(i=1; i<=dim_tau; ++i)
    fscanf(init_p, "%f", &p_tau[i]);
  for(i=1; i<=dim_u; ++i)
    fscanf(init_p, "%f", &p_u[i]);
  for(i=1; i<=dim_theta; ++i)
    fscanf(init_p, "%f", &p_theta[i]);
  fclose(init_p);
}


/* ================================================================
   print out parameters values and related residues
   ================================================================ */
void showParameters(float p_n[], float p_tau[], float p_u[], float p_theta[], float *old_n,
                    float *old_tau, float *old_u, float *old_theta)
```

```
{
 int i;
 float *df;
 float rn,rtau,ru,rth;

 for(i=1; i<=dim_n; ++i)
   printf(" %f", p_n[i]);
 printf("\n");
 for(i=1; i<=dim_tau; ++i)
   printf(" %f", p_tau[i]);
 printf("\n");
 for(i=1; i<=dim_u; ++i)
   printf(" %f", p_u[i]);
 printf("\n");
 for(i=1; i<=dim_theta; ++i)
   printf(" %f", p_theta[i]);
 printf("\n"):

 *old_n=rn=sum_res2_n(p_n);
 *old_tau=rtau=sum_res2_tau(p_tau);
 *old_u=ru=sum_res2_u(p_u);
 *old_theta=rth=sum_res2_theta(p_theta);

 printf("total=%f sum_res2_n=%f  tau=%f u=%f theta=%f\n",rn+rtau+ru+rth,rn,rtau,ru,rth);
}


/* ============================================================
   residual functions
   ============================================================ */

float res(float x,float y, int which)
{
 float n,ndel_x,ndel_y,tau,taudel_x,taudel_y,u,udel_x,udel_y,theta,thetadel_x,thetadel_y;
 float ex,c,s,nx,ny,t32,t52,taux,tauy,d2t_x,d2t_y,u2,ux,uy,thetax,thetay;
 float from_x,from_y,from_n,from_tau,from_u,from_theta;
 float taudel_2x, taudel_2y;

 n = n_trial(x,y,p_n);
 ndel_x = n_trial(x+h,y,p_n);
 ndel_y = n_trial(x,y+h,p_n);

 tau = tau_trial(x,y,p_tau);
 taudel_x = tau_trial(x+h,y,p_tau);
 taudel_y = tau_trial(x,y+h,p_tau);
 taudel_2x = tau_trial(x+2*h,y,p_tau);
 taudel_2y = tau_trial(x,y+2*h,p_tau);

 u = u1(x,y,p_u);
 udel_x = u1(x+h,y,p_u);
 udel_y = u1(x,y+h,p_u);

 theta=theta_trial(x,y,p_theta);
 thetadel_x=theta_trial(x+h,y,p_theta);
```

```
thetadel_y=theta_trial(x,y+h,p_theta);

ex=exp(-x);
c=cos(theta);
s=sin(theta);

nx=(ndel_x-n)/h;
ny=(ndel_y-n)/h;

t32=tau*sqrt(fabs(tau));
t52=tau*t32;
taux=(taudel_x-tau)/h;
tauy=(taudel_y-tau)/h;
d2t_x=(taudel_2x-2.0*taudel_x+tau)/(h*h);
d2t_y=(taudel_2y-2.0*taudel_y+tau)/(h*h);

u2=u*u;
ux=(udel_x-u)/h;
uy=(udel_y-u)/h;

thetax=(thetadel_x-theta)/h;
thetay=(thetadel_y-theta)/h;

switch(which){
case 1:
  {
   from_x = ex*(c*(ux + u*nx/n + alpha*u) - s*u*thetax);
   from_y = s*(uy + u*ny/n) + c*u*thetay;
   from_n = (from_x + from_y);
   return (from_n*from_n);
  }
case 2:
  {
   from_x =ex*( c*u*taux + (2.0/3.0)*tau*(alpha*u*c + c*ux - s*u*thetax)
                      - (2.0/3.0)*a*ex*((alpha-1.0)*t52*taux + 2.5*t32*taux*taux + t52*d2t_x));
   from_y = u*s*tauy + (2.0/3.0)*tau*(s*uy + u*c*thetay)
        - (2.0/3)*a*( 2.5*t32*tauy*tauy + t52*d2t_y);

   from_tau = (from_x + from_y);
   return from_tau*from_tau;
  }
case 3:
  {
   from_x = ex*c*(u*ux + taux + tau*nx/n);
   from_y = s*(u*uy + tauy + tau*ny/n);
   from_u = (from_x + from_y);
   return from_u*from_u;
  }
default:
  {
   from_x = ex*( c*u2*thetax - s*(taux + tau*nx/n) );
   from_y = s*u2*thetay + c*(tauy + tau*ny/n);
   from_theta = (from_x + from_y);
   return from_theta*from_theta;
```

```
    }
  }
}

/* ========================================
  TRIAL FUNCTIONS DEFINITIONS
  ======================================== */

float n_trial(float x, float y, float p_n[])
{
  float z;

  z=x*x + fabs(p_n[2])*y*y + fabs(p_n[3])*x;
  return exp(-fabs(p_n[1])*pow(z, fabs(p_n[4])));


}

float tau_trial(float x, float y, float p_tau[])
{
  float z;

  z=x*x + fabs(p_tau[2])*y*y + fabs(p_tau[3])*x;
  return exp(-fabs(p_tau[1])*pow(z + fabs(p_tau[5])*z*z, fabs(p_tau[4])));


}

float u1(float x, float y, float p_u[])
{
  float z;

  z=(x*x + fabs(p_u[2])*y*y + fabs(p_u[3])*x)*x;
  return sqrt(fabs(tau_trial(0,y,p_tau))) +
    1.0 - exp(-fabs(p_u[1])*pow(z,fabs(p_u[4])));
}

float theta_trial(float x, float y, float p_theta[])
{
  float z1,z2,b,theta1;

  z1=fabs(y*x/(fabs(p_theta[5]) + x*x));
  b=atan(fabs(p_theta[6])*z1);
  z2 = x*x + fabs(p_theta[2])*y*y + fabs(p_theta[3])*x;
  theta1=tanh(fabs(p_theta[1])*pow(z2, fabs(p_theta[4])));
  return b*theta1;

}

float u_x_trial(float x, float y, float p_u[], float p_theta[])
{
  return u1(x,y,p_u)*cos(theta_trial(x,y,p_theta));
}
```

```
float u_y_trial(float x, float y, float p_u[], float p_theta[])
{
  return u1(x,y,p_u)*sin(theta_trial(x,y,p_theta));
}
```

Bibliography

[1] J.D. Goodberlet, *An Experimental Investigation of a Table-top, Laser-Driven Extreme Ultraviolet Laser*, Ph.D Thesis, MIT, Cambridge, Mass (1996).

[2] O. Svelto, *Principles of Laser*, Plenum Press, New York (1989).

[3] P.L. Hagelstein, *Development of the MIT Table-top Soft X-Ray Laser*, SPIE, Vol. 1551 Ultrashort-Wavelength Lasers (1991), p. 254-274.

[4] R.C. Elton, *X-Ray Lasers*, Academic Press, Inc., New York (1990).

[5] C.E. Max, *Physics of Laser Fusion, Volume I, Theory of the Coronal Plasma in Laser Fusion Targets*, National Technical Information Service, Springfield, VA (1982).

[6] M.H. Muendel, *Short Wavelength Laser Gain Studies in Plasma Produced by A Small ND:Glass Slab Laser*, MIT Physics Ph.D. Thesis (1994).

[7] L. Spitzer, *Physics of Fully Ionized Gases*, Interscience Publishers, New York (1962).

[8] P.L. Hagelstein, *6.673 Class Notes Chapter 11, Spring 1996*, MIT.

[9] R. Fabbro, C. Max, and E. Fabre, Physics of Fluid 28, 1463 (1985).

[10] J.D. Huba, *1994 Revised NRL Plasma Formulary.*

[11] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing, 2nd Edition*, Cambridge University Press (1995).

[12] F.B. Hildebrand, *Advanced Calculus for Applications, 2nd Edition*, Prentice-Hall (1976).

[13] L.D. Landau, *Fluid Mechanics*, Pergamon Press, London (1959).

[14] C. Yih, *Fluid Mechanics*, McGraw-Hill, New York (1969).

[15] F.S. Acton, *Numerical Methods That Work*, Harper & Row, New York (1970).

[16] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley (1986).

[17] D. Mihalas and B.W. Mihalas, *Foundations of Radiation Hydrodynamics*, Oxford University Press, New York (1984).