THREE-DIMENSIONAL NUMERICAL STUDY OF FLAMES SUPPORTED BY A ROTATING BURNER

BY

KISHWAR N. HOSSAIN

B.S., Lafayette College, 2000
M.S., University of Illinois at Urbana-Champaign, 2003

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Adjunct Professor Thomas Jackson, Chair
Professor John Buckmaster
Associate Professor Gregory Elliott
Assistant Professor Dimitrios Kyritsis

Abstract

In this study non-uniform methane diffusion flames, formed from a porous plug burner spinning in quiescent air are investigated numerically in a three-dimensional context. Flames are simulated for Damköhler numbers on the upper branch of the S-response curve close to the extinction point. Multi-dimensional instabilities appear at these near extinction Damköhler numbers, as observed in experimental studies of flames sustained by a rotating fuel disk [1] and by a rotating porous burner[2], [3].

Simulation results from the constant density and constant viscosity model suggest that the non-uniform flames are a result of thermodiffusional instabilities and are a function of the Damköhler number. Non-uniform flames simulated in this study include flame holes, single armed spirals and double armed spirals. The flame holes have stationary edges and the radius of these holes is found to increase as the Damköhler number is lowered. The single and double spirals have edges that rotate about the axis of the burner. It is found that the velocity of the single spiral relative to the flow is considerably higher than that of the double spiral. It is also found that the shapes of the spirals are affected by the velocity vectors and by interactions between distinct spiral arms. Analyses of the scalar dissipation and cross-scalar dissipation rates of these flames show that the flames are primarily diffusion flames with some premixing near the edges.

Factors, other than the Damköhler number, that are found to significantly affect the stability of the spinning porous plug burner include the mixture strength and the exit velocity at the burner surface. The range of Damköhler number within which the system exhibits non-uniform behaviour is larger at higher values of exit velocity and lower values of mixture strength.

Acknowledgments

I would like to express my gratitude to all those who have contributed to the completion of this thesis. Firstly, I would like to thank my husband, Peshala, for his unrelenting support, encouragement, love and affection. In addition, I would like to thank my parents and my brother for their love and patience. I would also like to thank my father-in-law and mother-in-law for their encouragement.

I wish to acknowledge the continued guidance of my advisors, Professor Thomas L. Jackson and Professor John D. Buckmaster. I am forever indebted to them for providing me with the opportunity to pursue a Ph.D. and I will treasure the knowledge that I have gained from working with them. I would also like to thank Professor Gregory Elliott of the Department of Aerospace Engineering and Professor Dimitrios Kyritsis of the Department of Mechanical Science and Engineering for serving on my thesis committee.

I would like to acknowledge the support of this work by the Center for Simulation of Advanced Rockets (CSAR) at the University of Illinois which is supported by the U.S. Department of Energy under contract number B523819, as a part of its Advanced Simulation and Computing Program (ASC). I would also like to acknowledge the support of Computer Sciences and Engineering (CSE) and the support of AFOSR contract number AF FA9550-06-1-0332, A. Nachman, program manager, for computer time on the Air Force cluster located

at Illinois.

I would like to acknowledge the support of the Champaign Simulation Center, CAT Inc., through Enterprise Works. I would particularly like to thank Mr. Walt Lohmann, Dr. Christopher Ha and Dr. One-Chul Lee from the Champaign Simulation Center.

I would like to give special thanks to Staci Tankersley of AE and Jodi Gritten-Dorsett of CSAR for their continued help. In addition, I would like to thank Diane Jeffers and Kendra Lindsey of AE and Sheryl Hembrey of CSAR. I am also indebted to the Turing support staff, especially Michael Campbell, for all their help with queuing issues on the cluster. I would also like to thank Dr. Luca Massa for his help.

I owe many thanks to Bill Mason, Bill and Kris Hartmann, Jennifer and Filip Rysanek, Jason and Cindy Kamphaus, Chet Hammill, Jim Wrzosek and many others for their friendship and support. Finally, I would like to add that I will always cherish the time I have spent at the University, and for that I thank all those who have contributed to this memorable experience.

Table of Contents

List of Tables

List of Figures

List of Symbols

English Symbols

$A_i, \quad B_i, \quad C_i$	Runge-Kutta coefficients
b	Angular velocity
B	Pre-exponential factor
C_p	Specific heat
D	Damköhler number
D^*	Highest Damköhler number at which non-uniform flames can be sustained
D_E	Extinction Damköhler number
$\overline{D}$	Scaling factor for the Damköhler number
D_i	Diffusion coefficient for species i
E	Internal energy; average error
$L2$	Average error
E_p	Parallel efficiency
F, G, H	Similarity variables for the Von Karman spinning disk solution
h	Grid spacing
k	Wave number
Le_i	Lewis number of species i
n	Problem size
N	Total number of grid points
P	Pressure
p	Pressure; number of processors; order of convergence
Pe	Peclet number
Pr	Prandtl number
Q	Heat release
$r, \quad \theta \quad z$	Cylindrical coordinates
R_u	Universal gas constant
S_p	Speed-up
Sc_i	Schmidt number for species i
T	Temperature
t	Time
T^*	Burke-Schumann flame temperature

u	Radial velocity
v	Tangential velocity
w	Axial velocity
$x, \quad y, \quad z$	Cartesian coordinates
Y_i	Species i
Z	Mixture fraction
z^*	Burke-Schumann flame location
Z_e	Zeldovich number

Greek Symbols

α_i	Ratio of stoichiometric mass fraction for species i
β	Non-dimensional heat release parameter
χ	Scalar dissipation rate
χ_c	Cross-scalar dissipation rate
χ_s	Scalar dissipation rate at the stoichiometric surface
κ	Curvature
λ	Thermal conductivity
Λ	Reaction rate
μ	Viscosity
ν	Dynamic viscosity
ν_i	Stoichiometric coefficient of species i
ω	Growth rate
ϕ	Mixture strength
ρ	Density
τ	Non-dimensional time
τ_χ	Diffusive time

Chapter 1

Introduction

The evolution of non-uniform flames is typically a manifestation of the intrinsic instabilities of the system. These instabilities lead to dynamics that are unique and offer insight into the conditions necessary for the sustainability of the flame. An understanding of these instabilities can also be of importance to turbulence modeling. The flame supported by a rotating porous plug burner offers a suitable platform for the study of such an instability. Here the non-uniformity appears in the form of flame holes and spirals. The spirals are particularly interesting because they are distinct flames that rotate about the axis of the burner, and thus simultaneously support an ignition front and a trailing extinction front. An instantaneous representation of such a flame in terms of the scalar dissipation rate is characterized by distinct regions separated by edges representing propagation and recession of the flame. This study aims to identify the conditions that render the diffusion flame formed on a rotating porous plug burner unstable, and to analyze the features of the resulting non-uniform flame such as the dynamic edges, the characteristics of which may be of some importance to the study of turbulent flames. We begin our discussion with a more detailed

look at the configuration we are considering and a review of other studies that provide information on the onset of instabilities in diffusion flames.

Rotation of the porous plug causes a flow of the ambient air toward the burner, while there is a steady flow of fuel from the burner exit. Consequently, the fuel and air mix at a finite distance above the burner surface and a diffusion flame is formed, as sketched in figure 1.1. Here, the injection velocity of the fuel is w_0 and the angular velocity of the burner is b_0. If $b_0 > 0$ then the burner rotates counterclockwise. In this study the fuel is taken to be methane and the oxidizer is air. The viscosity, thermal conductivity, and the specific heat at constant pressure are taken to be constants. The density is also taken to be constant, uncoupling the mass and momentum equations from the energy and species equations. This is a standard modeling approach in combustion and has proven to be robust in identifying important parameters and their effect on stability [6]. The flow is solved, independently of the combustion equations, using the similarity solution for the Von Karman swirling flow [7]. For the combustion equations, a one-step global irreversible reaction is considered and the reaction rate is taken to be of the Arrhenius type. The system of unsteady, three-dimensional equations is then solved numerically using the velocity profiles as inputs.

Experimental studies using the rotating burner configuration show the evolution of non-uniform flames with variations in the rotational speeds of the burner [2], [3]. The experiments commence at low rotational speeds that sustain a steady flat flame. As the rotational speed is increased a pulsating flame hole, characterized by a flame sheet with a circular extinguished region in the center, forms. The hole expands outward until it reaches a critical radius, and then propagates inward to reestablish the flat flame; the cycle continues with frequencies

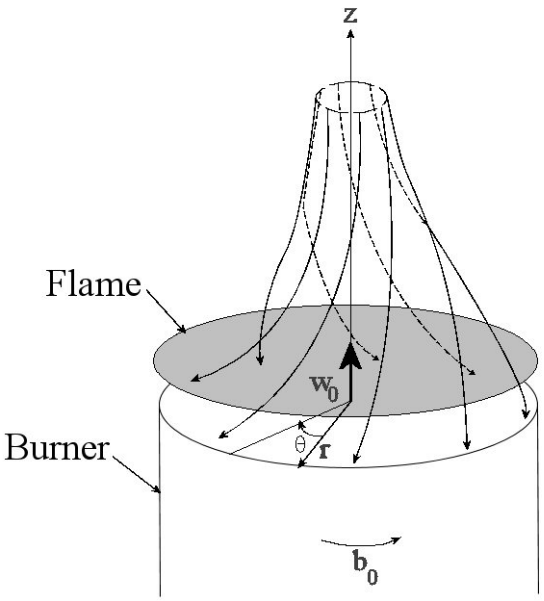


Figure 1.1: Sketch of a diffusion flame supported by a rotating burner

ranging from $1 - 4\ Hz$. As the rotational speed is increased further this pulsating hole transitions into single-armed, and then multi-armed, spiral flames.

Another set of experiments that use a similar configuration are those performed using a rotating PMMA disk, facing downward, burning in air. These show that steadily burning diffusion flames last for 15 to 20 s at low rotation speeds and high flow rates [1]. In these experiments two distinct flame configurations are observed. In one configuration a circular flame sheet of finite radius is formed and undergoes symmetric shrinking until it is extinguished. In the other, the initial circular flame disintegrates asymmetrically, forms spirals and is extinguished. In a related study Nayagam and Williams [8] investigate the inflammability limits of a stagnation point flow impinging on a spinning fuel disk. Here the authors consider a stagnation point flow with an external radial velocity gradient and a spinning

fuel disk with Arrhenius pyrolysis. This asymptotic study shows that the inflammability boundaries for the one-dimensional system are a function of the strain-rate parameter, a combination of the external radial velocity and the angular velocity of the disk. Thus, the strain-rate is an important parameter and may also have an effect on the stability of the system.

The influence of the strain-rate on the stability of diffusion flames is evidenced by a study of flames formed between opposed slot jet burners [9]. In these experiments a strain-rate gradient along the length of the burners is introduced by a slight misalignment of the burner exits. There is a threshold for the local strain-rate value below which the spatially uniform diffusion flame is replaced by a stable edge flame configuration [10], [11].

An experimental study by Pellet et al. [12] of diffusion flames formed between opposed jet burners shows that increases in the flow velocities causes a disk flame to "rupture" from the center outward. The resulting annular flame shifts axially until it reaches a stable location at the stagnation point of the flow. A reduction of the mass flow rate from the value where the ring flame is observed, leads to a shrinking and shifting of the ring until the disk flame is restored. The subsequent two-dimensional numerical study of Frouzakis et al. [13], reproduces the qualitative results of Pellet et al. [12], and shows that an increase in the Reynolds number of the flow leads to the local extinction of the flat diffusion flame and the formation of an edge-flame. The authors identify a range of Reynolds numbers within which either a diffusion flame or an edge-flame can form. Within this range of Reynolds numbers the type of flame that is obtained is dependent on the initial conditions; however, a mechanism for the transition of a diffusion flame to an edge flame is not established.

In another two-dimensional numerical study, Lu and Ghoshal [14] show that the dynamic behavior of a hole is dependent on the strain rate and the hole radius; for every strain rate there is a critical hole radius that is a bifurcation point and separates the expanding and shrinking behavior.

These destabilizing effects of the strain rate on diffusion flames are reminiscent of the transition from a stable to an unstable branch along the S-shaped response curve shown in figure 1.2. The S-shaped curve represents the typical response of a one-dimensional diffusion flame to variations in Damköhler number (D), the ratio of the chemical reaction rate to the rate of diffusion. Here, D_E represents the extinction point while D_I represents the ignition point. An investigation of the fast-time instability of a counterflow system shows that the top branch of the S-response curve is stable while most of the middle branch is unstable, and that the extinction point is where the flame sheet transitions from a stable to an unstable solution [15]. Sustainable non-uniform flame patterns are typically observed within a narrow range of Damköhler numbers on the upper branch of the S-response curve [16], [17], [18]. Buckmaster and Jackson [19], in their investigation of the propensity of flame holes to close in a zero velocity field, identify detachment Damköhler numbers, the maximum Damköhler number for which a hole will close. Thus, the S-response curve can provide a relatively clear visualization of parameters where the system is unstable.

Although the onset of instability in a system is typically a function of the Damköhler number, there are other parameters that affect the range of Damköhler numbers within which the system is unstable. A theoretical study by Kukuck and Matalon [18], on oscillations in a diffusion flame formed in a semi-infinite domain, shows that increases in the

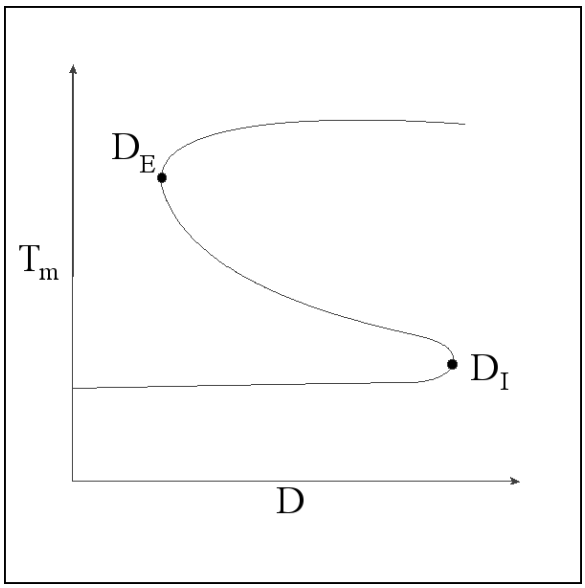


Figure 1.2: The S-shaped sesponse curve

mixture strength (defined as the ratio of the fuel and oxidizer mass fractions at the boundaries normalized by the stoichiometric coefficients), results in an increase in the range of Damköhler numbers where oscillations occur. This study also shows that an increase in the fuel supply temperature relative to the temperature of the oxidizer also promotes the onset of oscillations. The importance of these boundary conditions is also highlighted in a study of a counterflow in a rectangular channel closed at one end [20]. Here the authors show that stable flames formed in the channel consist of an edge near the closed end of the channel and a trailing diffusion flame. From a large Damköhler number analysis on the quasi one dimensional diffusion flame far from the edge they find that the wall boundary conditions have a significant effect on the strength of the diffusion flame. It is shown that there exists a range for the supply mixture strength beyond which a solution for the diffusion flame does not exist. The imposition of the diffusive fluxes at the boundaries leads to weaker flames, which in turn are prone to oscillations if the supply mixture strength has a low enough value.

Heat losses to the channel wall also have an effect on the onset of stability. The effect of heat loss is considered in detail in the context of a premixed flame anchored to a porous plug burner by Margolis [21] and by Buckmaster [22]. These studies show that heat loss to the burner has a stabilizing effect on the left stability boundary of the premixed flame. In terms of the right stability boundary, increases in the heat loss lead to a destabilizing shift till a threshold is reached, beyond which the burner is stabilizing. For high enough values of the activation energy, a Lewis number stability band does not exist.

Another parameter which affects the stability of diffusion flames is the Lewis number. The Lewis number has an affect, primarily, on the type of instability exhibited by a system. Chen et al. [23] observe that cellular flames can only be established close to extinction and for low effective Lewis numbers (Le_{eff}), defined as the Lewis number for the more completely consumed reactant. Cheatham and Matalon [17], in a theoretical study of cellular instabilities in a diffusion flame formed in a semi-infinite domain, show that for $Le_i \leq 1$ there exist unstable modes at D higher than the extinction value. For sufficiently large D the flame is stable. As D is lowered the flame becomes unstable before the extinction value, D_E, is reached. The growth rate for the instabilities increases as D is lowered further. The stability boundaries for different Lewis numbers indicate that, as the Lewis number for the oxidant is lowered, the boundaries shift toward higher values of Le_f making the cellular instability more accessible.

The effect of Lewis numbers on pulsating instabilities for the same configuration is given by Kukuck and Matalon [18]. They show that for a fixed mixture strength and fixed oxidant Lewis number there is a range of fuel Lewis numbers $1 < Le_f < Le_f*$ within which

oscillations can occur. An increase in the oxidant Lewis number leads to an expansion of this range and an increase in the range of Damköhler numbers within which the system can support oscillations.

From the discussion above it is evident that the key parameters for the onset of instabilities, and the consequent spatially non-uniform flame structures, in diffusion flames are Damköhler number, Lewis number, the boundary conditions at the injection surface, and the strain rate. In this study we analyze the affect of some of these parameters on the stability of the flame supported by a rotating porous plug burner. We begin with a detailed account of the formulation in section 2 and a discussion of the numerical method in section 3. A discussion of the results is given in section 4.

Chapter 2

Governing Equations and Boundary Conditions

In this study the density is taken to be constant. This facilitates the numerical solution by uncoupling the transport equations from the combustion equations. In doing so the effect of the temperature increase, due to reaction, on the flow is ignored. Consequently, the flow is modeled using the Von Karman spinning disk similarity solution [7]. The dimensional governing equations for a constant density and constant viscosity axisymmetric flow in cylindrical coordinates (r, θ, z) are given by

$$\frac{1}{r}\frac{\partial(ru)}{\partial r} + \frac{\partial w}{\partial z} = 0, \tag{2.1}$$

$$u\frac{\partial u}{\partial r} - \frac{v^2}{r} + w\frac{\partial u}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial r} + \nu\left[\frac{\partial}{\partial r}\left(\frac{1}{r}\frac{\partial ru}{\partial r}\right) + \frac{\partial^2 u}{\partial z^2}\right], \tag{2.2}$$

$$u\frac{\partial v}{\partial r} + w\frac{\partial v}{\partial z} + \frac{1}{r}uv = \nu\left[\frac{\partial^2}{\partial r}\left(\frac{1}{r}\frac{\partial rv}{\partial r}\right) + \frac{\partial^2 v}{\partial z^2}\right], \tag{2.3}$$

$$u\frac{\partial w}{\partial r} + w\frac{\partial w}{\partial z} = -\frac{1}{\rho}\frac{\partial p}{\partial z} + \nu\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial w}{\partial r}\right) + \frac{\partial^2 w}{\partial z^2}\right], \tag{2.4}$$

where $(u,\ v,\ w)$ are the radial, tangential and axial velocities, respectively. The appropriate boundary conditions are,

$$At \quad z = 0, \quad u_0 = 0, \quad v_0 = b_0 r, \quad w_0 = w_0, \quad P_0 = 0, \tag{2.5}$$

$$At \quad z = \infty, \quad u_\infty = 0, \quad v_\infty = 0 \quad and \quad w_\infty = C. \tag{2.6}$$

At $z = 0$ the burner rotates with an angular velocity of b_0 while fuel is injected from the porous surface. The vertical velocity at the surface is given by, w_0. The constant C is to be determined as a part of the solution.

A one-step irreversible reaction of the form,

$$\nu_f Fuel + \nu_o Oxidant \rightarrow \nu_p Products \tag{2.7}$$

is considered, where ν_f *and* ν_o are the stoichiometric coefficients of the fuel and oxidant, respectively. The chemical reaction rate is assumed to be Arrhenius. The Mach number is assumed to be low and so the effect of pressure is negligible in the energy and species equations. The unsteady, three-dimensional energy and species equations for constant density and constant viscosity are given by

$$\mathbf{L}[\rho C_p T, \rho Y_i] - \mathbf{M}[\lambda T, \rho D_i Y_i] = \Omega[Q, -\alpha_i], \tag{2.8}$$

where

$$\mathbf{L} = \frac{\partial}{\partial t} + u\frac{\partial}{\partial r} + \frac{v}{r}\frac{\partial}{\partial \theta} + w\frac{\partial}{\partial z},$$
$$\mathbf{M} = \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial r^2} + \frac{1}{r^2}\frac{\partial}{\partial \theta^2} + \frac{\partial^2}{\partial z^2}.$$

Here, T is the temperature, Y_i the species mass fraction, Q the heat release, α_i the ratio of

stoichiometric mass fractions, and D_i the diffusion coefficient for species i. Ω is the reaction rate and has the form, $\Omega = BY_oY_f exp\{-E/R_uT\}$, where B is the pre-exponential factor and E/R_u is the gas phase activation temperature. The boundary conditions are

$$\begin{aligned}
At\ z = 0: &\quad T = T_s, \quad \frac{\partial Y_f}{\partial z} = \frac{w_o}{D_f}(Y_f - Y_{f0}), \quad Y_o = \frac{w_o}{D_o}Y_o, \\
At\ z = \infty: &\quad T = T_\infty, \quad Y_o = Y_{o\infty}, \quad Y_f = 0, \\
At\ r = \delta: &\quad T = T_{avg}, \quad Y_i = Y_{i_{avg}}, \\
At\ r = \infty: &\quad \frac{\partial T}{\partial r} = 0, \quad \frac{\partial Y_i}{\partial r} = 0, \\
Periodic\ in\ \theta: &\quad T|_{\theta=0} = T|_{\theta=2\pi}, \quad Y_i|_{\theta=0} = Y_i|_{\theta=2\pi}, \\
&\quad \frac{\partial T}{\partial \theta}|_{\theta=0} = \frac{\partial T}{\partial \theta}|_{\theta=2\pi}, \quad \frac{\partial Y_i}{\partial \theta}|_{\theta=0} = \frac{\partial Y_i}{\partial \theta}|_{\theta=2\pi}.
\end{aligned} \tag{2.9}$$

At the burner surface the temperature is set to a constant, T_s. As discussed earlier, this is an important parameter because increases in the temperature differential between the burner exit and the ambient increases the range of Damköhler number within which the system is unstable [18]. Flux conditions are used for the species at the burner surface. At $z = \infty$ the temperature is set to the ambient temperature, T_∞, while the fuel mass fraction is set to zero and the oxidizer mass fraction is set to the appropriate value for air. Neumann boundary conditions are applied at $r = \infty$, while at $r = 0$ dirichlet conditions derived from averages of the field variables from adjacent θs are used, that is,

$$T_{avg} = \int_{\theta-a}^{\theta+a} T(\delta, \theta, z) d\theta \tag{2.10}$$

$$Y_{i_{avg}} = \int_{\theta-a}^{\theta+a} Y_i(\delta, \theta, z) d\theta \tag{2.11}$$

with $\delta << 1$ and $a = \pi/8$. The temperature and species are taken to be periodic in θ .

2.1 The Constant-Density Model

The constant-density approximation is crucial in rendering combustion problems manageable for analysis. Burke-Schumann in effect introduced this approximation in their pioneering analysis of diffusion flames [24]. The justification for it can be derived from the notion that variations in temperature are a result of combustion and the heat release is small compared to the energy of the mixture, hence, the density variations are small and the velocity field can be approximated to be that of a constant-density fluid to leading order [6]. In most combustion phenomena, however, there are large increases in temperature and its primary effect on the flow is in the form of thermal expansion. Thermal expansion plays a crucial role in premixed flames in terms of promoting an intrinsic hydrodynamic instability, but it does not have a similar effect on diffusion flames. Thus, the constant-density model has been used prudently in numerous studies to model diffusion flames. In a constant-density model, the density is taken to be a constant everywhere, thus neglecting any effects of thermal expansion. In recent years the validity of the constant-density model, in terms of its effect on diffusion flames, has been explored through numerical studies [25], [26], [27].

These studies suggest that although thermal expansion affects flame stability it does not play as crucial a role in diffusion flames as it does in premixed flames. It is found that oscillatory or cellular diffusion flames are typically not driven by hydrodynamic instabilities but are thermo-diffusional in nature [26]. A comparison of growth rates for a planar unstrained flame produced by one reactant introduced at one end of a tubular domain diffusing against a stream of the other reactant at the opposite end, shows that thermal expansion results in a widening of the range of Damköhler numbers, within which cellular instabilities occur

while shrinking the range within which oscillatory instabilities occur [25].

A study of edge flames formed by introducing two streams, one of fuel and the other oxidizer, through porous plugs shows that the effect of thermal expansion is primarily on the standoff of distance of the flame [27]. At low values of axial velocity this effect is small however at higher velocities it is significant.

These studies show that the onset of oscillations in diffusion flames is not a result of the constant-density model. Thermal expansion does have an effect on the quantitative nature of the instabilities in terms of ranges of parameters and stand-off distances, however, the constant-density model generally provides a good approximation to the qualitative results. In terms of the current study, it is expected that thermal expansion would result in the widening of the range of Damköhler numbers within which the instabilities manifest themselves, while having little effect on the qualitative nature of the instabilities.

2.2 The Porous Plug Burner

A porous plug burner is considered in this study primarily for ease of numerical representation. In a porous plug burner the reactant is injected through the the burner tube at a controlled rate and a constant temperature; the backflow of products is prevented by the porous plug. It is assumed that the heat conducted back to the burner is removed by cooling coils such that the surface temperature is constant. Thus, the temperature, mass flux and the composition of the reactants can be prescribed at the burner exit. These boundary conditions at the burner exit are expected to affect both the structure and the stability of the diffusion flame. As mentioned earlier, heat loss to the burner affects the stability boundaries

of a premixed flame anchored to a porous plug [21], [22]. Similar effects are expected in the current configuration as evidenced by the experimental studies of Nayagam [3]. Here, a 17.78 cm diameter burner is considered and the injection velocity is varied from 0 cm/s to 0.20 cm/s. In the corresponding map of flame patterns it is shown that for a rotation rate of around 3 rps the flame can transition from a single spiral to a semi-circular flame with a straight edge, to a pulsating flame hole as the fuel flow rate increases. Thus, the value of the mass flux is an important parameter of study.

The temperature at the burner exit is also expected to affect the nature of the flame supported by the rotating porous plug. Studies by Kukuck and Matalon [18] and Short and Liu [20] show that an increase in the temperature differential between the reactants has an adverse effect on the stability of the diffusion flame. In the current study the temperature at infinity is set to the ambient, thus the parameter of interest in terms of the temperature differential is T_s. Since the burner exit cannot be cooler than the ambient the minimum possible temperature at the burner exit is T_∞. This is also expected to be the most stable exit temperature as indicated by [18] and [20]. Previous studies using burners show that under experimental settings, the temperature at the burner exit is likely to be slightly higher than the ambient. Smooke et al [28] find that taking an exit temperature of $1.4T_\infty$ improves correlation of the flame structure and temperature between experiments and two-dimensional numerical simulations. In a more recent numerical study of an axisymmetric counterflow diffusion flame formed between opposing jets, the temperatures at the jet exits were set to 345 K at the oxidizer boundary, 315 K at the fuel boundary and 350 K at the cooling flange boundaries surrounding the burner exits [29]. The results from these numerical sim-

ulations correlated well with the experimental values as well. This suggests, however, that the temperature in the vicinity of the burner exit is likely to be higher than the ambient even when cooling mechanisms are utilized. Thus, in this study two exit temperatures will be considered to evaluate the affect of this boundary condition on the stability of the flame. The ideal case where the exit temperature is set to the value of the ambient, T_∞, and a case where the exit temperature is $1.4T_\infty$.

2.3 Non-dimensional Equations

For non-dimensionalization, the lengths are scaled by $\sqrt{\nu/b_0}$, time by the reciprocal of b_0, temperature by T_∞, and the species by Y_{f0}, the mass fraction of the fuel at the burner exit. The similarity variables for the velocities are

$$F = \frac{u}{b_0 r}, \quad G = \frac{v}{b_0 r}, \quad H = \frac{w}{\sqrt{b_0 \nu}}, \quad P = \frac{p}{\rho b_0 \nu}, \tag{2.12}$$

where F, G, H and P are functions of z. The resulting non-dimensional equations are

$$H' = -2F, \tag{2.13}$$

$$F'' = F^2 + F'H - G^2, \tag{2.14}$$

$$G'' = 2FG + HG', \tag{2.15}$$

$$P' = 2HF - 2F'. \tag{2.16}$$

Here, one boundary condition is needed for H, two for F, two for G, and one for P. Also, the equation for P is uncoupled from the rest and may be ignored in the simulation. The boundary conditions are given by

$$F(0) = 0, \quad G(0) = G_0 = 1, \quad H(0) = H_0, \quad P(0) = 0,$$

$$F(\infty) = 0, \quad G(\infty) = 0. \tag{2.17}$$

The non-dimensional unsteady energy and species equations are given by

$$L[T, Y_i] - M[T, Le_i Y_i] = \Omega[\beta, -\alpha_i], \tag{2.18}$$

where

$$\mathbf{L} = \partial/\partial t + rF\partial/\partial r + G\partial/\partial\theta + H\partial/\partial z,$$

$$\mathbf{M} = r^{-1}\ \partial/\partial r + \partial^2/\partial r^2 + r^{-2}\ \partial^2/\partial\theta^2 + \partial^2/\partial z^2.$$

Here, $Le_i = \rho C_p D_i/\lambda$, is the Lewis number for species i, $\beta = QY_{f0}/C_pT_0$, the heat release parameter, $\Omega = D\overline{D}Y_oY_f exp\{Ze(1 - T^*/T)\}$, the reaction rate, $D = Bb_0^{-1}\rho^{-1}Y_{f0}$, the Damköhler number, the ratio of the chemical reaction rate to the diffusion rate, $Z_e = E/(R_uT_0T^*)$ the Zeldovich number, $\overline{D} = Ze^3T^*/z^{*2}$, a scaling factor, and, T^* and z^* the Burke-Schumann flame temperature and flame location, respectively, for the baseline parameters listed in section 4.1. The Prandtl number, $Pr = \nu\rho C_p/\lambda$, is set to 1.0. The boundary conditions are

$$\begin{aligned}
z = 0: &\qquad T_0 = T_s,\quad Y_{f_z} = H_0 Le_f(Y_f - 1),\\
&\qquad Y_{o_z} = H_0 Le_o Y_o,\\
z = \infty: &\qquad T_\infty = 1,\quad Y_f = 0,\quad Y_o = \phi^{-1},\\
r = \delta: &\qquad T = T_{avg},\quad Y_i = Y_{i,avg}\\
r = \infty: &\qquad T_r = 0,\quad Y_{i_r} = 0,
\end{aligned}$$

and periodic in θ

$$T_\theta|_{\theta=0} = T_\theta|_{\theta=2\pi},\quad Y_{i_\theta}|_{\theta=0} = Y_{i_\theta}|_{\theta=2\pi},$$

$$T_\theta|_{\theta=0} = T_\theta|_{\theta=2\pi},\quad Y_{i_\theta}|_{\theta=0} = Y_{i_\theta}|_{\theta=2\pi}.$$

Here, ϕ is the mixture strength defined as the ratio of the fuel mass fraction at the burner surface to the oxidizer mass fraction at infinity. This definition of the mixture strength is not normalized by the stoichiometric coefficients and is thus different from the one used by Kukuck and Matalon [18]. At the burner surface the non-dimensional temperature is set to a constant, T_s, and at $z = \infty$ the temperature is set to the ambient temperature, unity, while

the fuel mass fraction is set to zero. Neumann boundary conditions are applied at the outer radial boundary and Dirichlet conditions at the inner boundary, while the temperature and species are taken to be periodic in θ .

For the current study the fuel is taken to be methane, and the oxidizer is air; thus, $Le_f = 1.0$, $Le_o = 1.0$ $\alpha_f = 0.25$ and $\alpha_o = 1.0$. The mass fraction of air at infinity, $Y_{o\infty}$ is taken to be 0.2. The parameters of study are D, T_s, H_0, and ϕ while β and Ze are kept constant.

Chapter 3

Numerical Method

The one-dimensional similarity equations of the Von Karman spinning disk flow, given by equations (D.2-D.4), are solved using the COLSYS package [30]. Colsys is a robust collocation package used for solving mixed-order systems of multipoint boundary value problems. It employs a method of spline collocation at Gaussian points [30].

The energy and species equations, (D.7), are solved using a fourth-order centered finite difference scheme in space and a $2N$-storage, five-stage, fourth-order Runge-Kutta scheme, $(5,4)\ \ RK$, in time [4]. The fourth order centered difference schemes is used because it provides high accuracy and is relatively easy to implement on a parallel platform. Section B.0.1 gives a summary of the grid refinement study that shows that the centered difference scheme is fourth order accurate.

The $2N$-storage $(5,4)\ \ RK$ scheme is used due to its high accuracy and low storage requirements. The low-storage feature is especially relevant due to the relatively large problem sizes being considered in this study. The $2N$-storage method for an initial value problem

$$U_t = F[t, U(t)];\ \ U(t_0) = U_0, \tag{3.1}$$

is given by

$$t_j = t^N + C_j * h,$$

$$dU_j = A_j dU_{j-1} + hF(U_j),$$

$$U_j = U_{j-1} + B_j dU_j, \quad j = 1, ..., M, \tag{3.2}$$

where, h is the time step and j is a step in the RK scheme. Also, $U_1 \equiv U^N$, $U_M \equiv U^{N+1}$, $t_M \equiv t^{N+1}$, and the scheme is self-starting so that $A_1 = 0$. Thus, in a 2N storage scheme only the dU and U vectors need to be stored. The coefficients, A_j, B_j and C_j of the 2N storage (5,4) RK scheme used in this study are given in table 3. Carpenter and Kennedy [4] have shown that the non-linear accuracy for the $(5, 4)$ RK is higher than that for the $(4, 4)$ RK and that in terms of the work ratio the $(5, 4)$ RK is nearly as efficient. In comparison to the Williamson $(3, 3)$ RK, $2N$ storage[31] the $(5, 4)$ RK has higher accuracy for a given step size and greater stability boundaries [4]. Section B.0.2 also gives results from benchmark studies used to verify the capabilities of the code in solving simplified problems, the analytical solutions of which are known.

Coefficient	Value
A_1	0.0
A_2	-0.4178904745
A_3	-1.192151694643
A_4	-1.697784692471
A_5	-1.514183444257
B_1	0.1496590219993
B_2	0.3792103129999
B_3	0.8229550293869
B_4	0.6994504559488
B_5	0.1530572479681
C_1	0.0
C_2	0.1496590219993
C_3	0.3704009573644
C_4	0.6222557631345
C_5	0.9582821306748

Table 3.1: Coefficients for optimal $(5, 4)$, $2N$-storage RK scheme, solution 3 of Carpenter and Kennedy [4]

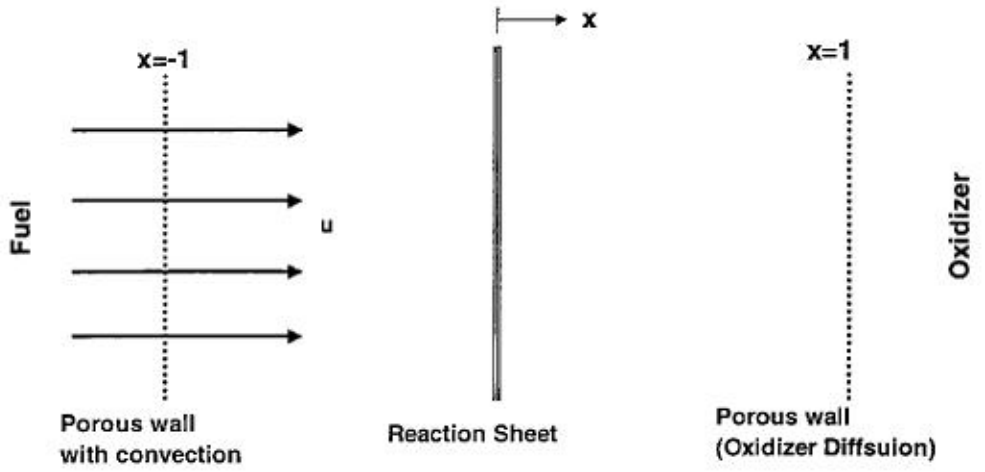


Figure 3.1: Sketch describing the one-dimensional diffusion problem used for code validation [5]

3.1 One-Dimensional Validation

A validation study is undertaken to compare the one-dimensional results obtained from the current solver with those obtained by Vance et al [5] for a diffusion flame between plane parallel porous walls. Vance et al [5], investigate the stability of a one-dimensional problem with flowing fuel and diffusing oxidant ($Le_f = Le_o = 1$) on opposite sides of the reaction sheet. Using perturbation analysis they find that for Lewis numbers less than unity the upper branch of the S response curve is stable and the middle branch is unstable. At Lewis numbers greater than unity there are damped oscillations for $1 < Le < Le_c$ but the entire upper branch is stable. For Lewis numbers greater than a critical value L_c the beginning of the upper branch of the S response curve is unstable.

Figure 3.1 shows a sketch of the problem considered. Here the fuel travels through a porous wall at a finite speed u toward another porous wall with diffusing oxidizer. A flame sheet forms from the interaction of the fuel and and the diffused oxidizer. The flow

is assumed to be uniform, radiative heat losses are ignored, and a single step irreversible Arrhenius reaction is assumed [5]. The non-dimensional equations are given by

$$\frac{\partial T}{\partial t} + Pe\frac{\partial T}{\partial x} = \frac{\partial^2 T}{\partial x^2} + \omega, \tag{3.3}$$

$$\frac{\partial y_i}{\partial t} + PeLe\frac{\partial y_i}{\partial x} = \frac{\partial^2 y_i}{\partial x^2} - \omega, \tag{3.4}$$

where $i = O,\ F$. The boundary conditions are

$$T = T_o \quad y_F = 1 \quad y_O = 0 \quad at\ x = -1,$$

$$T = T_o \quad y_F = 0 \quad y_O = 1 \quad at\ x = +1.$$

Figure 3.2 shows the two leading eigenvalues for this problem for a Lewis number of 2.0, where the two eigenvalues form a complex conjugate pair beyond point B'. Between points B' and C' the real parts of the eigenvalues are positive thus implying that the flame is unstable for this range of Damköhler numbers and so small perturbations would lead to oscillations which grow in time and eventually result in extinction. Beyond C', however, perturbations would cause damped oscillations resulting in a non-oscillatory steady state. This behavior is also illustrated by solving equations 3.3-3.4 using a second-order correct finite difference method and the results from this are shown in Figure 3.3, where the unstable solution is for a Damköhler number lower than C' and the stable solution is for a Damköhler number greater than that at C'.

For the purposes of code validation the above system is solved using the finite difference solver used in the current study. The steady state solution of the system given in equations

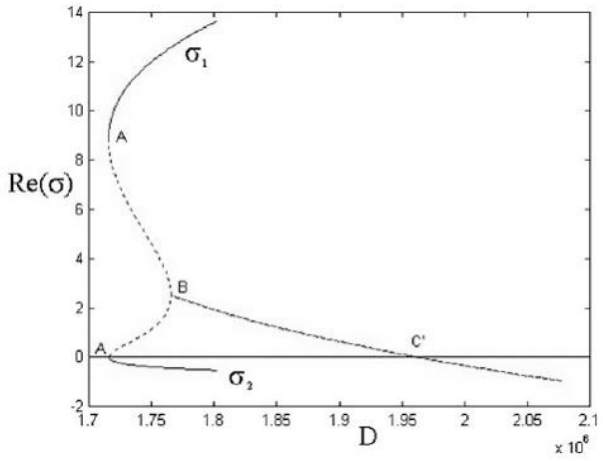


Figure 3.2: Leading two eigenvalues for one-dimensional code validation problem for $Le = 2.0,\ T_a = 4.0,\ Pe = 0.0,\ T_0 = 0.05$ [5]

3.3 and 3.4 are computed using COLSYS [30] and used as initial conditions for the solver. Figures 3.4 a and b show the steady state solutions at Damköhler numbers of 1.955e6 and 2.050e6 respectively.

Figures 3.5 and 3.6 shows transient results obtained for $Le = 2.0,\ Pe = 0.0,\ T_a = 4.0,\ T_0 = 0.05$. As seen from this plot, the results are similar to the ones obtained by Vance

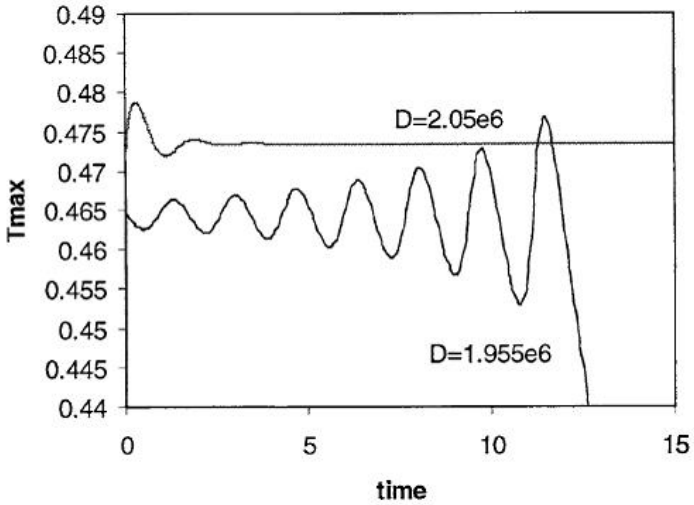
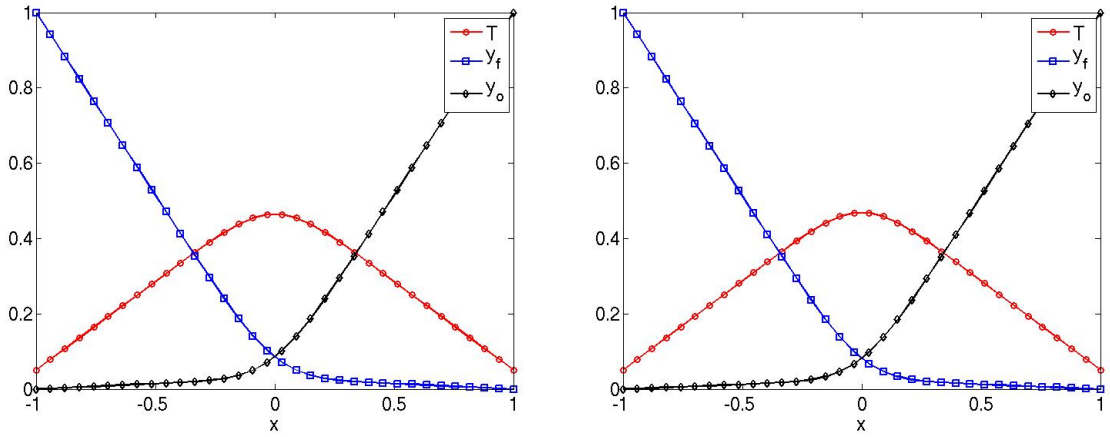


Figure 3.3: Transient solution for one-dimensional code validation problem $Le = 2.0,\ Pe = 0.0,\ T_a = 4.0,\ T_0 = 0.05$ [5]



a. b.

Figure 3.4: Steady state solutions for $Le = 2.0$, $Pe = 0.0$, $T_a = 4.0$, $T_0 = 0.05$, a. $D = 1.955e6$, b. $D = 2.050e6$

et al. Small perturbations at $D = 2.050e6$ cause damped oscillations which eventually die down and lead to a non-oscillatory steady state. At $D = 1.955e6$ the system is unstable and the amplitude of the oscillations increase with time and eventually the flame is extinguished. These results show that the finite difference solver is able to capture both the steady state and the transient characteristics of the simplified problem considered by Vance et. al. [5]. These results validate the finite different solver in one dimension, and the same finite difference scheme is used in all three spatial dimensions.

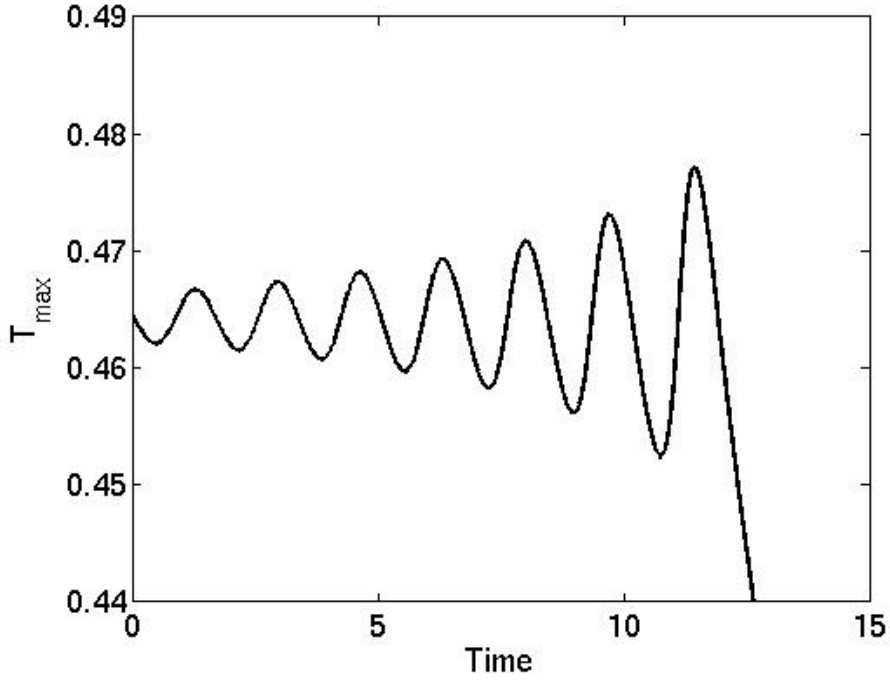


Figure 3.5: Transient solution for $Le = 2.0,\ Pe = 0.0,\ T_a = 4.0,\ T_0 = 0.05,\ D = 1.955e6$

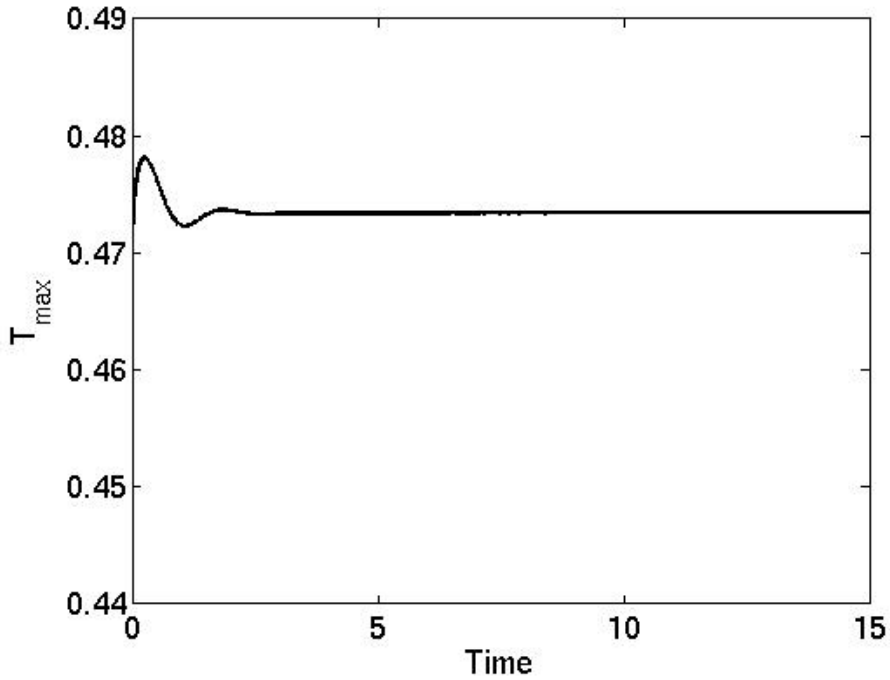


Figure 3.6: Transient solution for $Le = 2.0,\ Pe = 0.0,\ T_a = 4.0,\ T_0 = 0.05,\ D = 2.050e6$

3.2 Parallel Implementation

The solver is adapted for parallel processing using the Message Passing Interface (MPI). In the parallel implementation each processor is assigned a portion of the three-dimensional domain. A processor completes each step of the Runge-Kutta iterations for its domain and then communicates with its neighbor to exchange the necessary boundary values. Results of a performance study for the parallel implementation are given in figures 3.7 and 3.8. Here the problem size, n, is given by,

n = 16 : 16 X 16 X 16 in (r, θ, z),

n = 32 : 32 X 32 X 32,

n = 64 : 64 X 64 X 64,

n = 128: 128 X 128 X 128,

and p is the number of processors. The speedup (S_p) [32] is defined as,

$$S_p = \frac{\text{serial time}}{\text{parallel time}} = \frac{t_1}{t_p}, \tag{3.5}$$

and the efficiency (E_p) [32], is defined as,

$$E_p = \frac{\text{serial cost}}{\text{parallel cost}} = \frac{t_1}{pt_p}. \tag{3.6}$$

Here, the serial time, t_1, is the time taken for one processor to complete an assigned task, and the parallel time, t_p, is the time taken for p processors to complete the same taskin parallel.

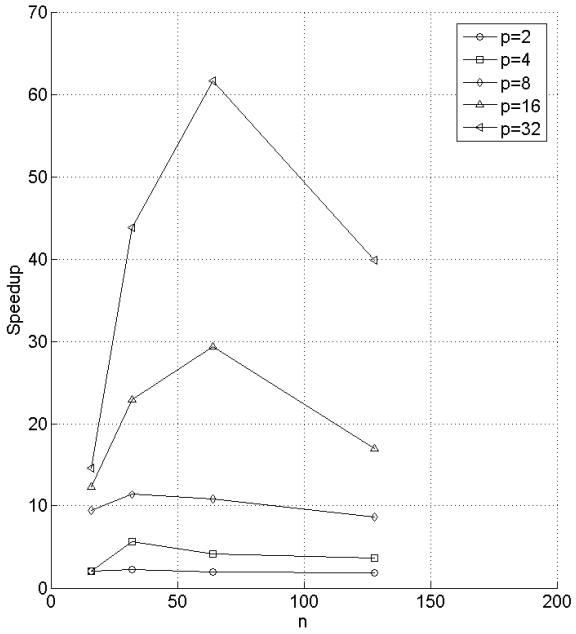


Figure 3.7: Speedup as a function of the problem size

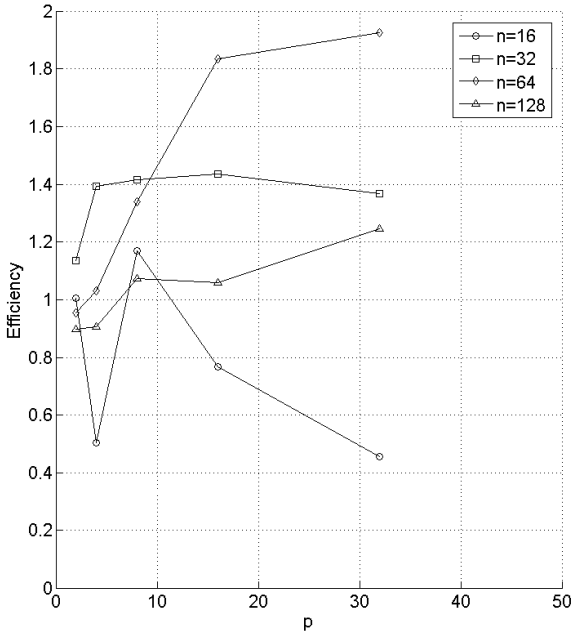


Figure 3.8: Efficiency as a function of the number of processors

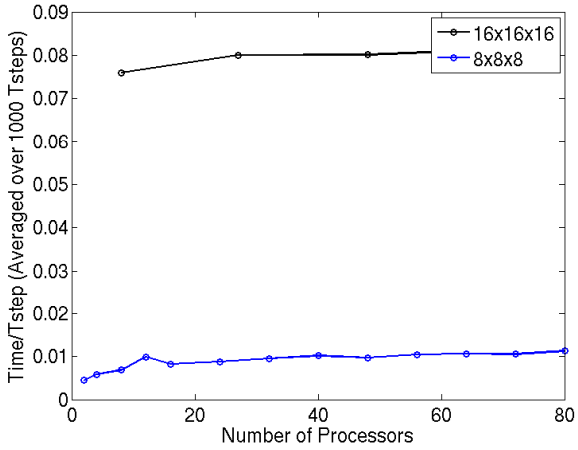


Figure 3.9: Scalability of the parallel code

Figure 3.7 shows the speedup as a function of the problem size for different number of processors. It is evident that an increase in the number of processors results in an increase in the speedup. As the problem size is increased, at first the speedup increases but it reaches a peak value and then decreases until it asymptotes to a particular value. Figure 3.8 shows that the efficiency is higher than 1 for most of the problem size/number of processor combinations. The solver is inefficient when the problem size is too small relative to the number of processors used. In these cases the communication time between the processors is higher than the time necessary for the computations and so additional processors actually result in longer run times.

Figure 3.9 demonstrates the scalability of the parallel code with increasing number of processors. Here, each processor is assigned either 8^3 or 16^3 grid points. As seen from the figure, the time taken remains fairly constant with increasing number of processors, and the consequent increase in the total problem size.

3.3 Solution Methodology

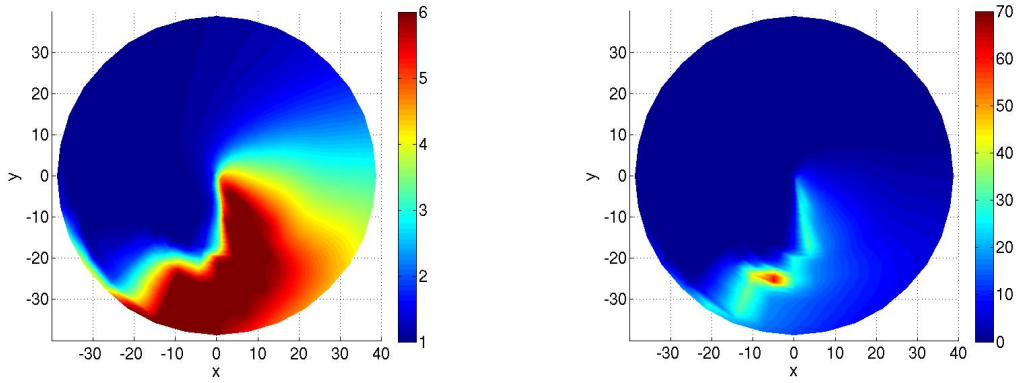
In order to solve the three-dimensional unsteady energy and species equations, the one-dimensional Burke-Schumann type flame sheet solution is used as the initial condition. This solution is computed using COLSYS [30]. This is a reasonable approach since the simulations commence at large Damköhler numbers where the fast chemistry limit is valid, and so the response is one-dimensional. Radial and angular perturbations are introduced into the system once the steady state solution is reached for a given Damköhler number. If the small perturbations do not grow and the flat flame persists, the simulations are restarted at a lower Damköhler number using the three-dimensional unperturbed solution from the previous Damköhler number as the initial condition. This process is repeated till the system becomes unstable to the perturbations and non-uniform flames appear.

3.4 Grid Convergence

Figures 3.10-3.13 show contour plots for the temperature and reaction rate for four different grids. The parameters used for these simulations are $D = 0.14$, $\phi = 2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 20$, and $Z_e = 15$. As seen from figure 3.10 the flame is not fully resolved when a 32^3 grid is used, however, it is evident from 3.11-3.13 that the contours vary little as the grid densities in r and θ are increased from 64^3-$128 \times 128 \times 64$. The flame rotates counter-clockwise and the oscillations at the trailing edge of the flame are seen in all three cases. These oscillations are difficult to resolve due to the dynamic nature of the flame. Although there are some differences in the size of the spiral between the grids, the general shape of the spiral is grid independent. Comparison of the integral value of the reaction rates over the entire spatial domain, between the 64^3, 96^3 and the $128 \times 128 \times 64$ grids, given in table 3.2, shows differences within 5% between the 64^3 and the two finer grids. Thus, for the results shown in the following sections a 64^3 grid is used.

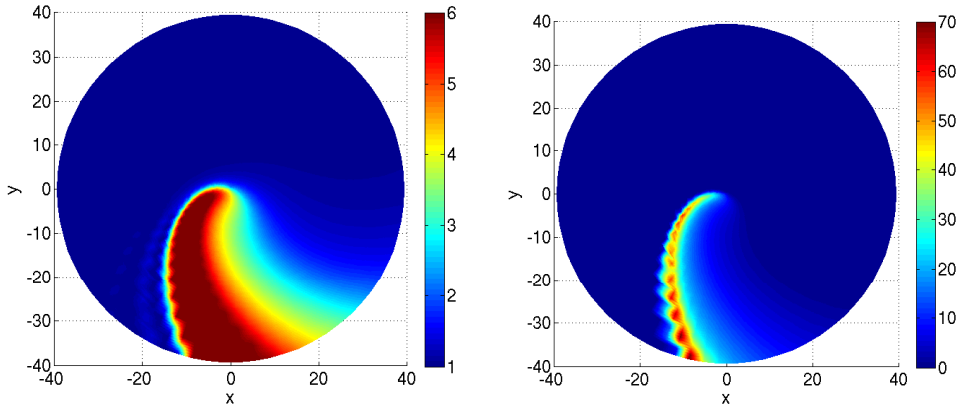
Grid	$\int$ Reaction Rate	Difference (%)
64^3	1.27×10^3	
96^3	1.22×10^3	3.94
$128 \times 128 \times 64$	1.21×10^3	4.72

Table 3.2: Comparison of the sum of reaction rates



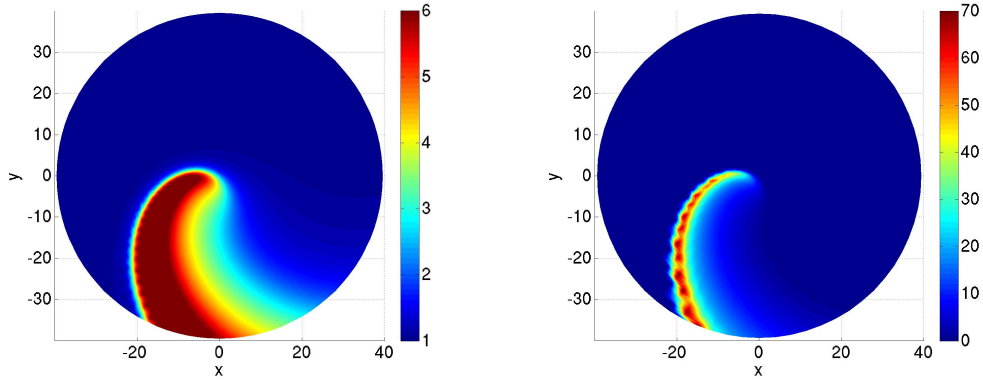
a. Temperature b. Reaction Rate

Figure 3.10: Temperature and reaction rate contours for a 32 x 32 x 32 grid



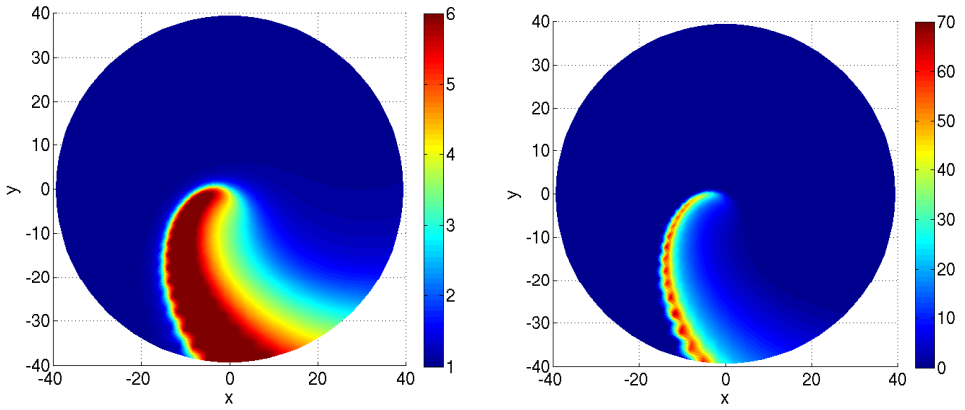
a. Temperature b. Reaction Rate

Figure 3.11: Temperature and reaction rate contours for a 64 x 64 x 64 grid



a. Temperature b. Reaction Rate

Figure 3.12: Temperature and reaction rate contours for a 96 x 96 x 96 grid



a. Temperature b. Reaction Rate

Figure 3.13: Temperature and reaction rate contours for a 128 x 128 x 64 grid

Chapter 4

Results

4.1 The Burke-Schumann Flame Sheet

The Burke-Schumann limit is important to consider in order to gain a fundamental understanding of the diffusion flame, and to establish initial conditions for numerical solutions. This limit arises as $D \to \infty$ resulting in an infinite reaction rate that can be balanced by taking the flame sheet to be very thin such that $Y_f Y_0 \to 0$. This thin flat flame sheet forms at a distance z^* from the burner surface and separates the fuel and the oxidizer. At this limit the combustion field is expected to be a function of z and t only. The energy and species equations then become

$$\mathbb{L}(T) = -\beta\alpha_i^{-1}\mathbb{L}(Y_i) = \Omega, \tag{4.1}$$

where

$$\mathbb{L} = \frac{\partial}{\partial t} - H\frac{\partial}{\partial z} - \frac{\partial^2}{\partial z^2} \tag{4.2}$$

however it should be noted that only the steady state is considered in order to solve for the Burke-Schumann flame sheet. The boundary conditions are

$$at \;\; z = 0, \;\; T = T_s, \;\; \frac{\partial Y_f}{\partial z} = H_0(Y_f - 1), \;\; Y_o = 0, \tag{4.3}$$

$$at \;\; z \to \infty, \;\; T = T_\infty, \;\; Y_f = 0, \;\; Y_o = \phi^{-1}. \tag{4.4}$$

The Schvab-Zeldovich variables are

$$Z_+ = T + \beta\alpha_f^{-1} Y_f \;\; \text{and} \;\; Z_- = T + \beta\alpha_o^{-1} Y_o. \tag{4.5}$$

As $D \to \infty$,

$$\mathbb{L}(Z_+) = \mathbb{L}(Z_-) = 0 \tag{4.6}$$

and the boundary conditions are

$$at \;\; z = 0, \;\; \frac{\partial Z_+}{\partial z} = \frac{\partial Z_-}{\partial z} + H_0(Z_+ - T_s - \beta\alpha_f^{-1}), \;\; , Z_- = T_s,$$

and

$$at \; z \to \infty, \;\; Z_+ = T_\infty, \;\; Z_- = T_\infty - \beta\phi^{-1}$$

Thus, the temperature and species profiles are given by,

$$0 < z < z^* : \;\; T = Z_+, \;\; Y_f = \beta^{-1}\alpha_f(Z_- - Z_+), \;\; Y_o = 0,$$

and

$$z^* < z < z_\infty : \;\; T = Z_-, \;\; Y_f = 0, \;\; Y_o = \beta^{-1}\alpha_o(Z_+ - Z_-).$$

The matching condition

$$Z_+(z^*) = Z_-(z^*) \tag{4.7}$$

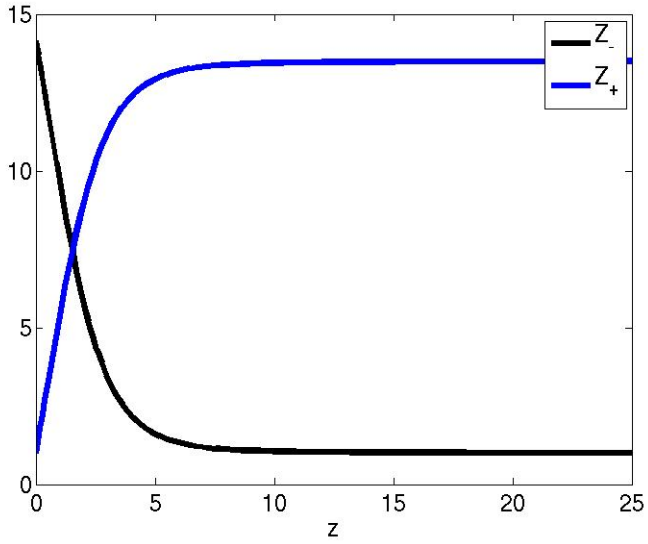


Figure 4.1: Burke-Schumann solution, Schvab-Zeldovich variables

is used to determine z^*. The steady state solutions for Z_+ and Z_- are obtained using COLSYS and are shown in figure 4.1 while the temperature and species profiles are shown in figure 4.2. As seen from this figure, for $H_0 = 0.1$, $T_s = 1.0$ and $\beta = 25.0$, the baseline parameters for this study, the Burke-Schumann flame temperature, $T^* = 7.43$, and the flame location, $z^* = 1.54$. These of the flame temperature and flame location are used in section 2.3 in the scaling factor for the Damköhler number.

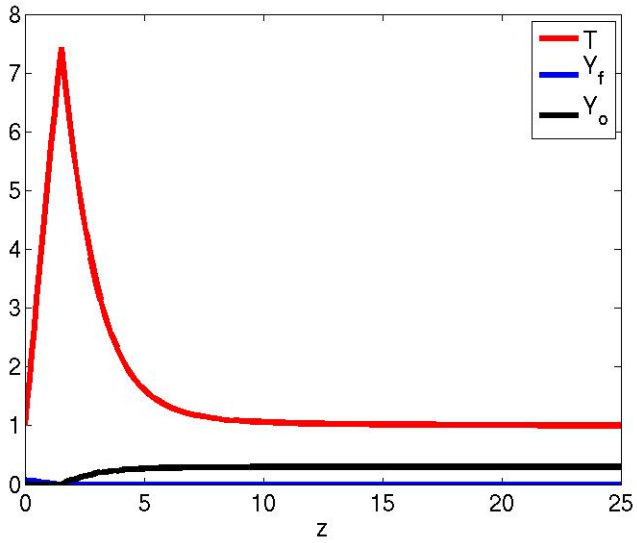


Figure 4.2: Burke-Schumann solution, temperature and species profiles

4.2 Non-uniform Flames at a Mixture Strength Value of 2.0

The stability of the three-dimensional spinning burner configuration is investigated using small pure mode perturbations. It is found that at appropriate Damköhler numbers these perturbations result in non-uniform flame patterns such as holes and spirals. This phenomenon is first investigated for parameter values of $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, and $Z_e = 40$. Figures 4.3 and 4.4 show the velocity profiles and the S-shaped response curve for this set of parameters. The extinction point, D_E, is approximately 1.48 as seen from figure 4.4. $D^* = 1.96$ represents the highest Damköhler number at which sustainable non-uniform flames appear. The non-uniform flames observed in this study include flame holes, single spirals and double spirals as shown in figures 4.6-4.10.

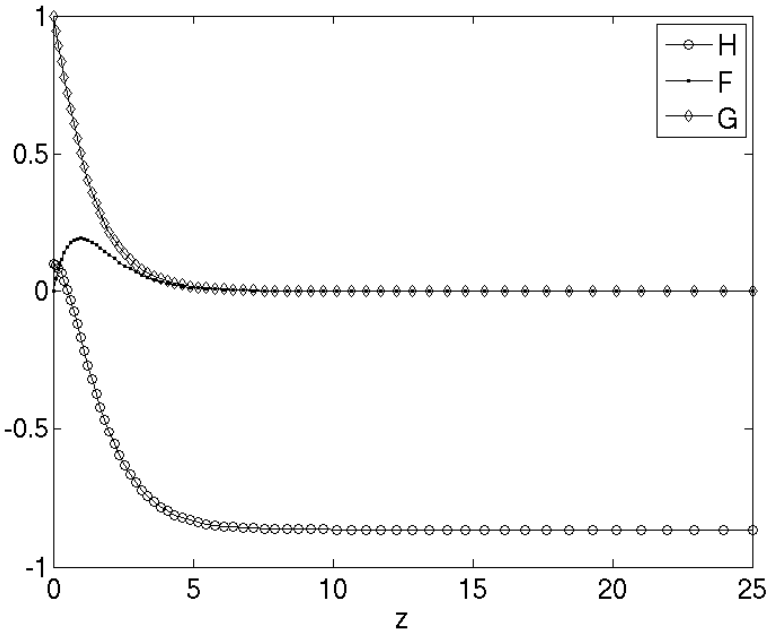


Figure 4.3: Non-dimensional velocities

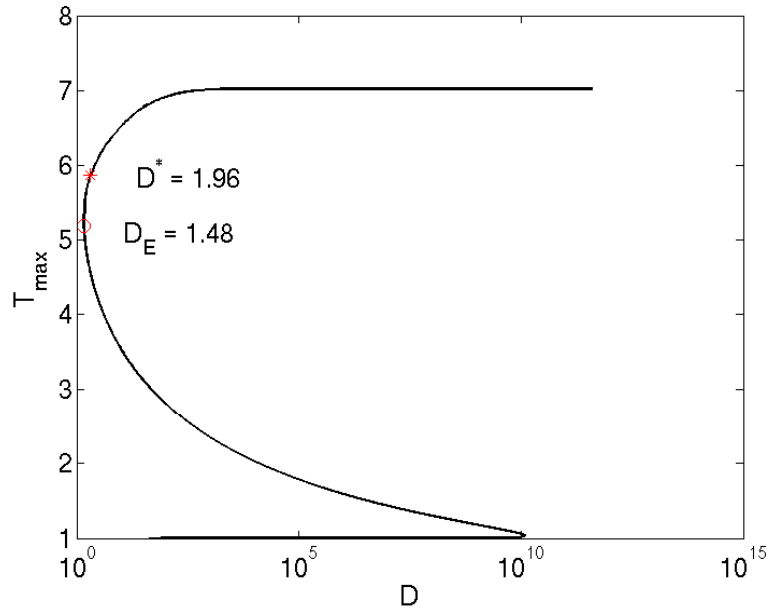


Figure 4.4: S-shaped response

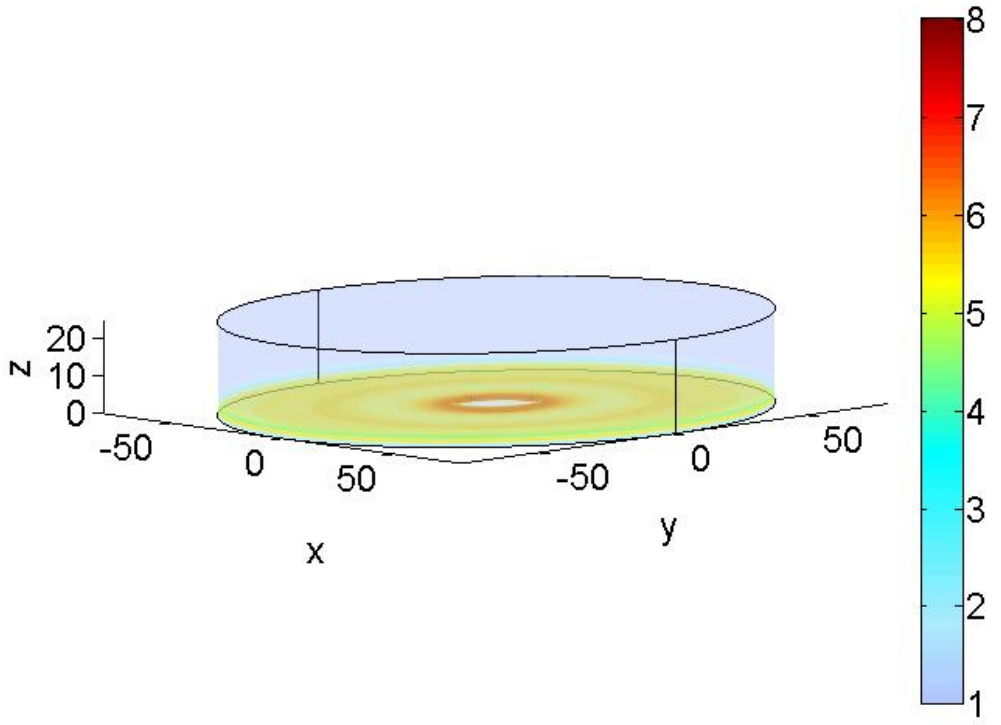


Figure 4.5: Temperature contour of a flame hole for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 40$ and $D = 1.96$

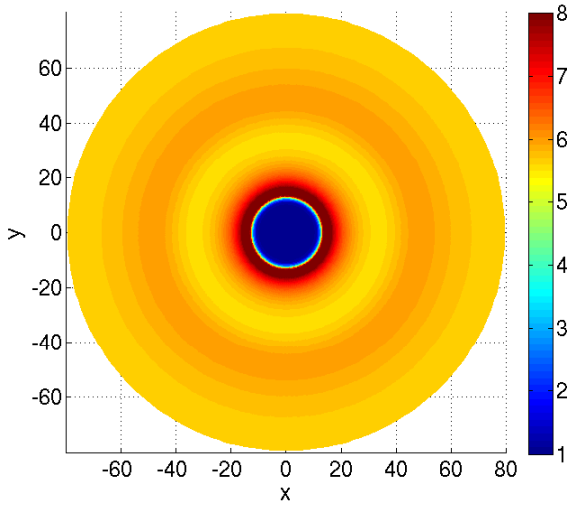


Figure 4.6: Temperature contour of a flame hole for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 40$ and $D = 1.96$

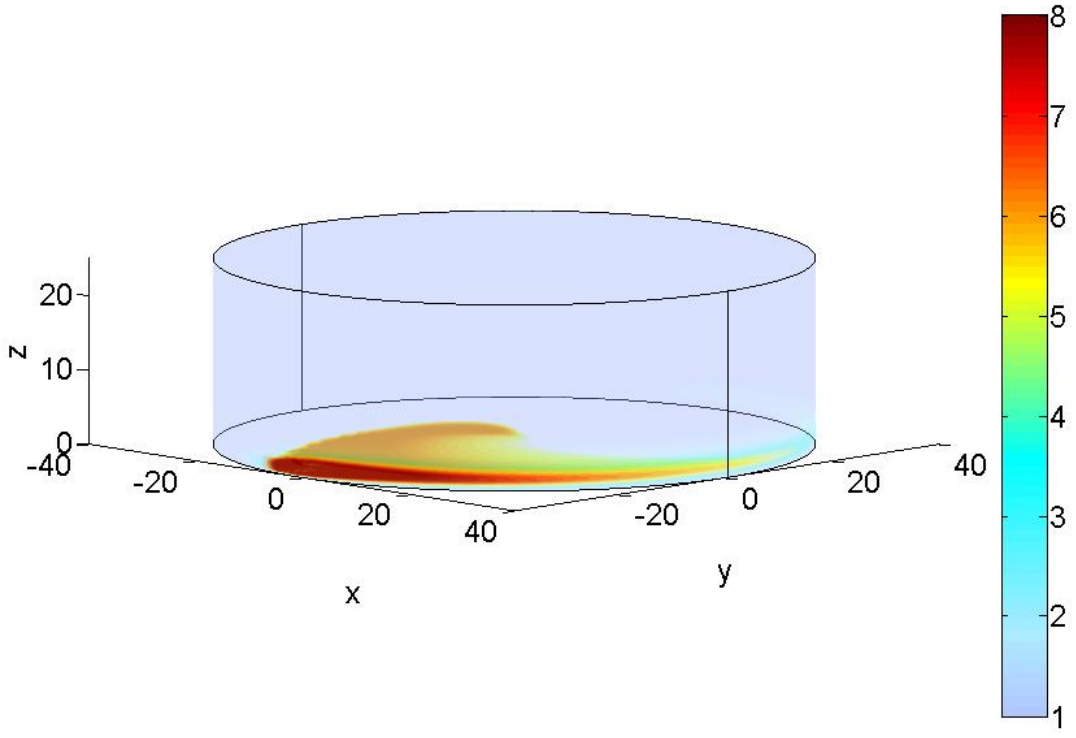


Figure 4.7: Temperature contour of a single spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 40$ and $D = 1.91$

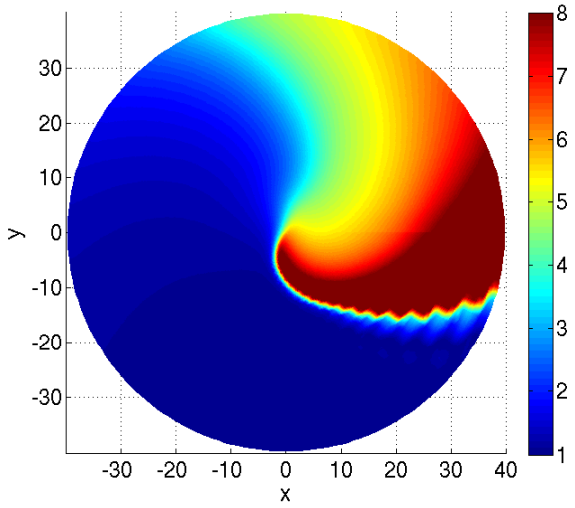


Figure 4.8: Temperature contour of a single spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.91$

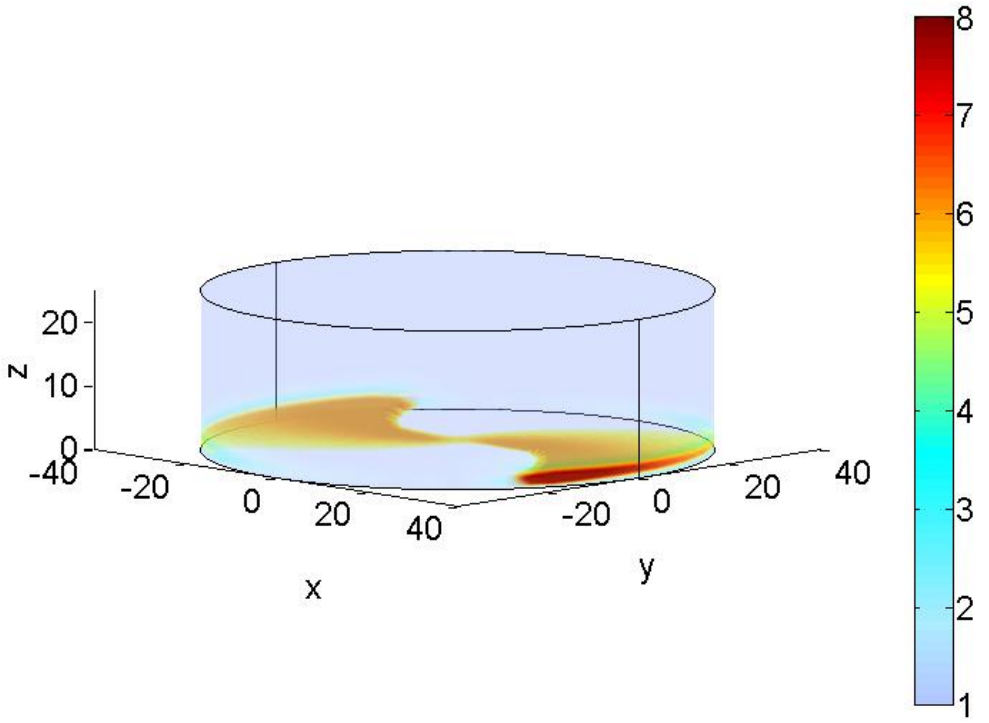


Figure 4.9: Temperature contour of a double spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 40$ and $D = 1.85$

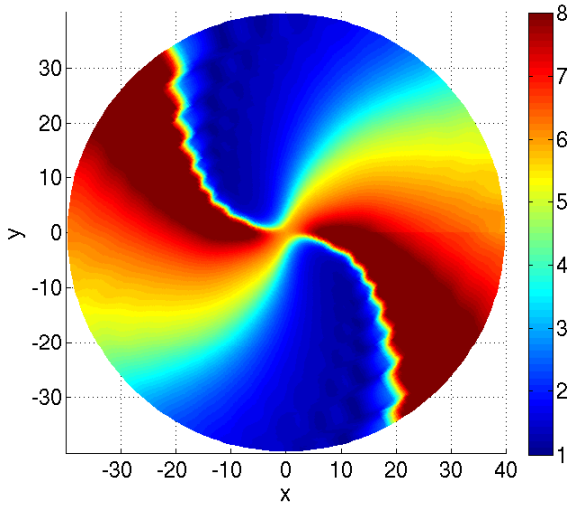
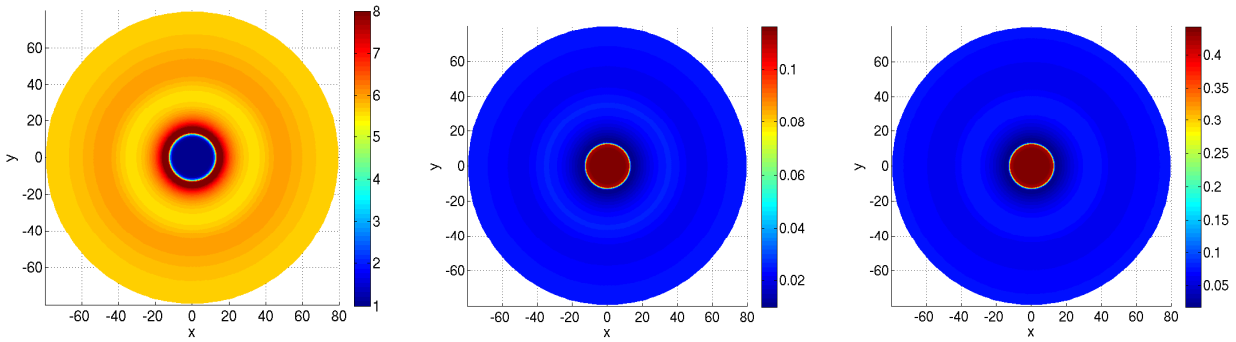


Figure 4.10: Temperature contour of a double spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.85$

As seen from figure 4.4 the range of Damköhler numbers within which these non-uniform flames appear is small. However, this range can be further divided into smaller groups where different modes are viable. For example, for the set of parameters listed above, the flame hole appears for $D = 1.96$ to $D = 1.92$, while single spirals appear for $D = 1.91$ to $D = 1.85$ and so on. It is expected that as the Damköhler number is lowered further the ranges within which higher modes are sustained can also be determined. However, for this section the discussion is limited up to double spirals, primarily due to the fact that for the higher modes the ranges become smaller and so it is difficult to identify distinct regions corresponding to each mode. The progression of the flames from the flame hole to the single armed and then the multiarmed spirals with decreasing Damköhler number is reminiscent of the observations of Nayagam and Williams [3] where the flames transition from flame holes to spirals with increasing angular velocity. It should be noted here that in the current formulation the Damköhler number is inversely proportional to the angular velocity. Thus the trends observed in the simulations are similar to those found in the experimental studies. The flame patterns are analyzed further in the following sections.



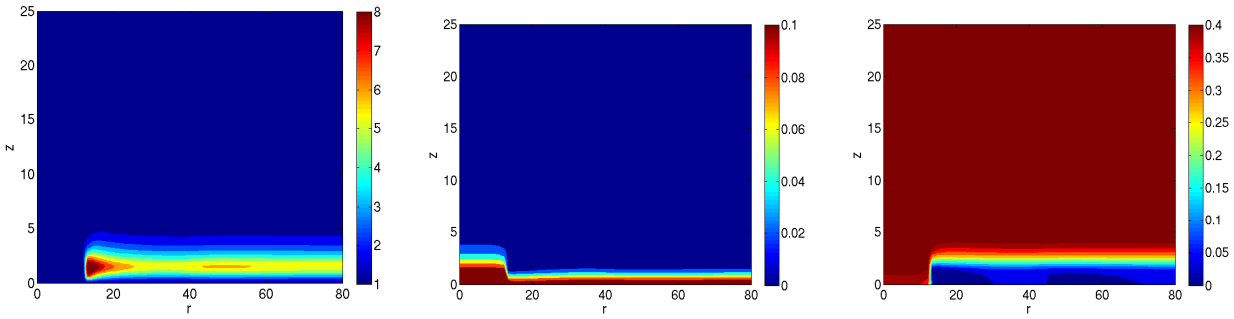
a. T b. Y_f c.Y_o

Figure 4.11: Temperature and species contours of a flame hole for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.96$

4.3 Flame Hole

Figures 4.11a-c show the temperature and species contours in the xy-plane of a flame hole at $D = 1.96$. Here, a central cold region is surrounded by an axisymmetric, annular flame sheet. A region of high temperature appears immediately adjacent to the boundary of the flame hole and then the temperature gradually decreases and oscillates along the radius. The geometry of the flame hole is better illustrated in the temperature and species contours in the rz plane given in figures 4.12a-c. Here, the edge of the flame is at a radial location of approximately $r = 13.4$. The temperature is high in the triangular region near the edge of the flame as compared to the tail of the flame. The fuel and oxidizer contours suggest that there is some mixing near the edge of the hole. However, beyond the edge, the fuel and oxidizer seem separated suggesting that beyond the boundary region of the hole, where some premixing may occur, there exists purely a diffusion flame. The characteristics of the edge of the flame hole is further analyzed in terms of the cross scalar dissipation parameter in a following section.

The radial oscillations in the trailing diffusion flame beyond the hole region are visible in



a. T b. Y_f c.Y_o

Figure 4.12: Temperature and species contours of a flame hole for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.96$

the rz contours as well. These oscillations can be explained by considering small perturbations in the large z field. As $z \to \infty$ the angular and radial velocities become negligible while the axial velocity asymptotes to H_∞. Here, the solution of the homogeneous heat equation is given by

$$T = exp(-1/2[H_\infty + \sqrt{H_\infty^2 + 4\mu^2}]z)J_0(\mu r). \tag{4.8}$$

It is found that the wavelength of the radial oscillations shown in figure 4.13 correspond to $\mu \approx 0.14$ which implies

$$T = exp(H_\infty z)J_0(\mu r) \tag{4.9}$$

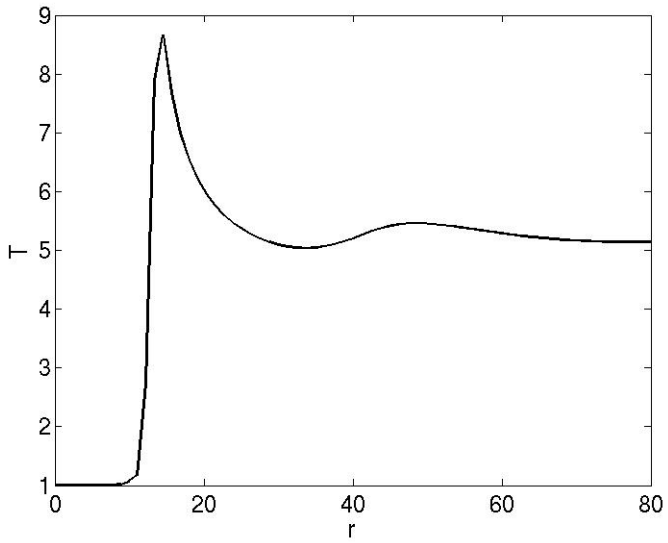


Figure 4.13: Radial temperature profile at $\theta = 0rad$.

4.3.1 Effect of Damköhler Number on Flame Hole Radius

As discussed in section 4.3 the flame hole is the first non-uniform flame observed as the value of the Damköhler number nears D_E. The flame hole forms at a Damköhler number of 1.96 while a single spiral forms at $D = 1.9$. It is found that at Damköhler numbers between these two values the flame hole expands with decreasing Damköhler number. Flame holes are simulated for $1.940 \leq D \leq 1.960$ and the flame hole radius, defined as the distance from the axis to the point where the temperature rises above 5 in the stoichiometric plane, are tabulated in table 4.1. Figure 4.14 shows the Damköhler number as a function of the flame hole radius while figures 4.15-4.19 show temperature contours of flame holes at different Damköhler numbers in the xy and rz planes.

D	Flame Hole Radius	Temp (r=80)	1-D Flame Temp
1.960	13.37	5.54	5.857
1.955	20.35	5.61	5.857
1.950	22.73	5.40	5.857
1.945	31.24	5.87	5.857
1.940	34.97	5.76	5.848

Table 4.1: Damköhler number and flame hole radii

It is evident from 4.1 that there is a significant increase in the flame hole radius within the range of Damköhler numbers considered. The expansion of the flame holes suggests that either there is an increase in the radial dissipation away from the axis and as the Damköhler

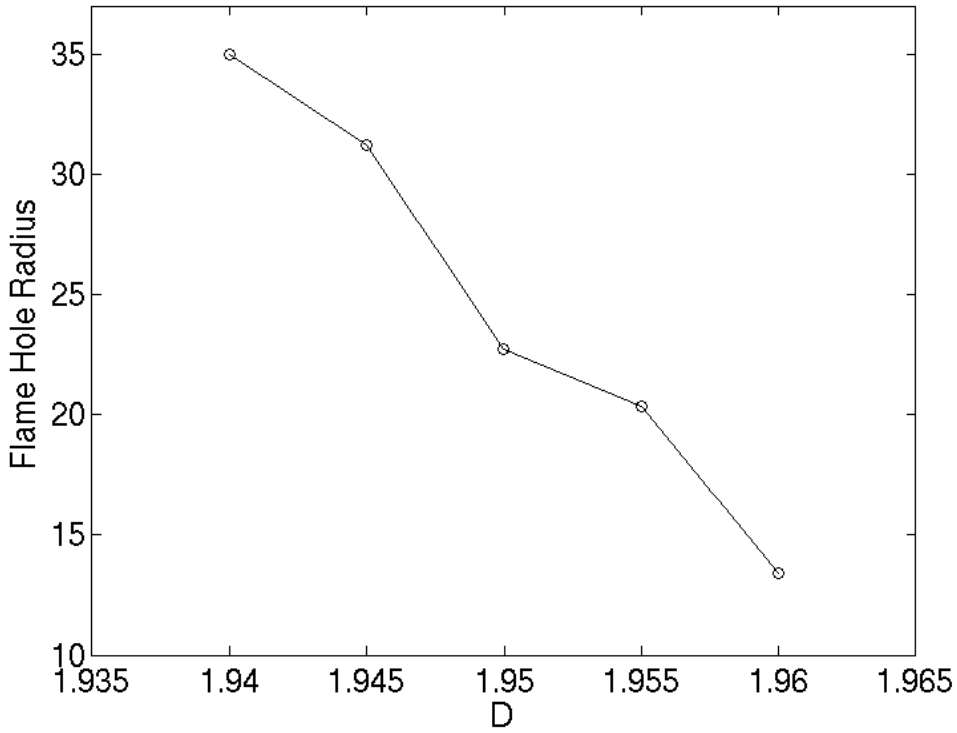


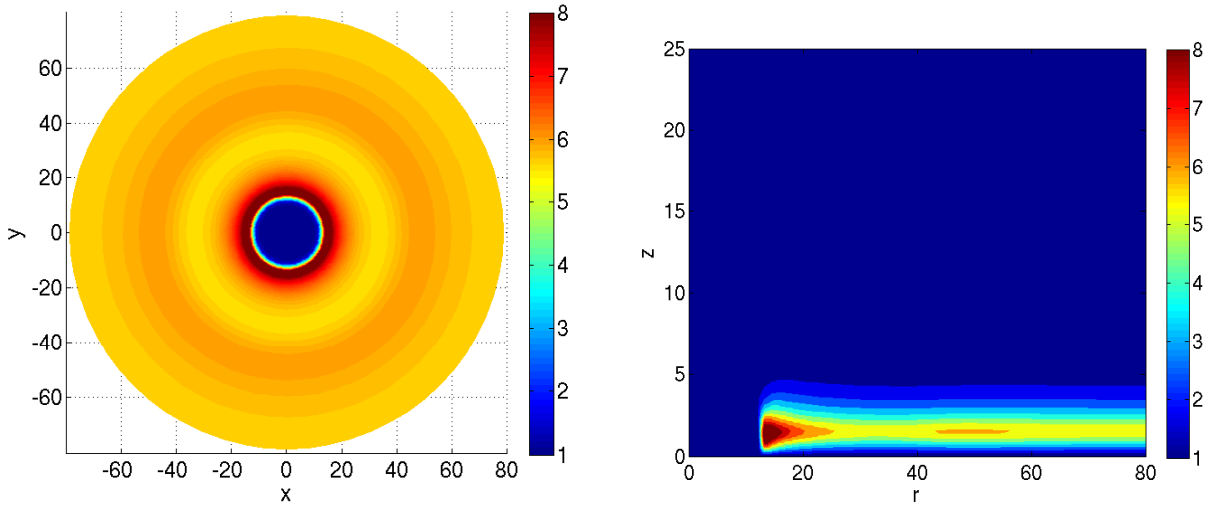
Figure 4.14: Flame hole radius as a function of Damköhler number

number decreases the reaction is not fast enough to sustain a flame so the hole expands, or that there is increased dissipation out of the plane in the quenched region as compared to the region where the flame prevails. The temperature contours in the rz plane suggest that the tail of the flame is purely a diffusion flame and from table 4.1 it is seen that the temperatures at $r = 80$ are generally close to the flame sheet temperatures. These contours also demonstrate that as the hole expands the hot triangular region adjacent to the flame hole also expands. Figures 4.20-4.24 show the reaction rate contours of the flame holes corresponding to Damköhler numbers from 1.96 to 1.94. These contours clearly demonstrate that the flame is tribrachial. As the Damköhler number is decreased and the hole radius increases the reaction rate in the rich premixed arm increases. Also, the height of the premixed arms increase as the Damköhler number decreases. Figure 4.25 demonstrates the

relationship between the curvature (κ) in the rz plane at the stoichiometric height, with the radial velocity. Here, κ is defined as,

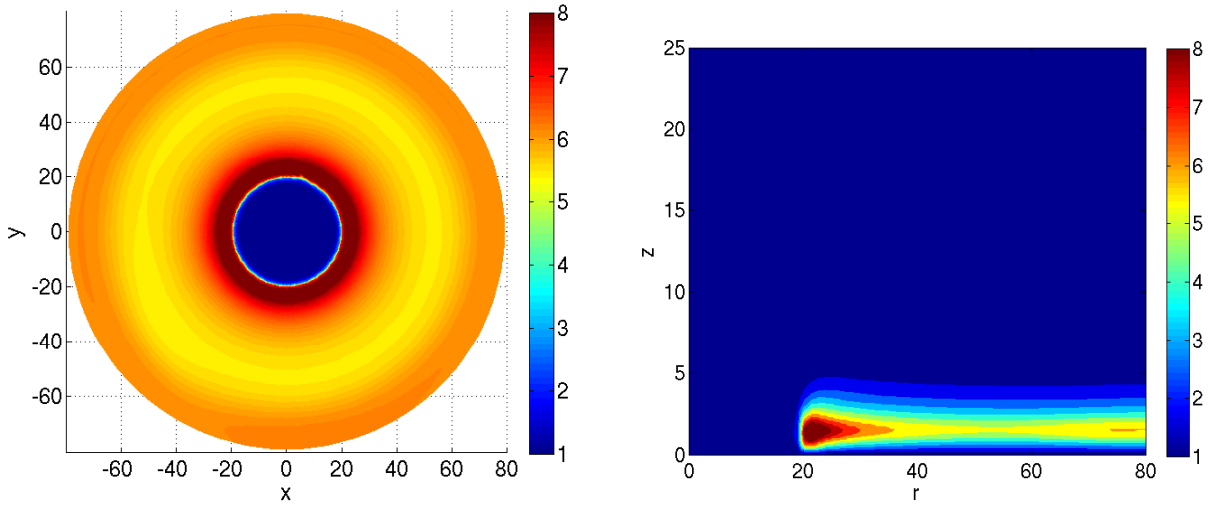
$$\kappa = \frac{d^2y}{dr^2} / [1 + \left(\frac{dy}{dr}\right)^2]^{3/2}, \tag{4.10}$$

[33], where $y = f(r)$ is the curve representing the reaction rate contour. As seen here, the curvature in the rz plane changes considerably with the hole radius.



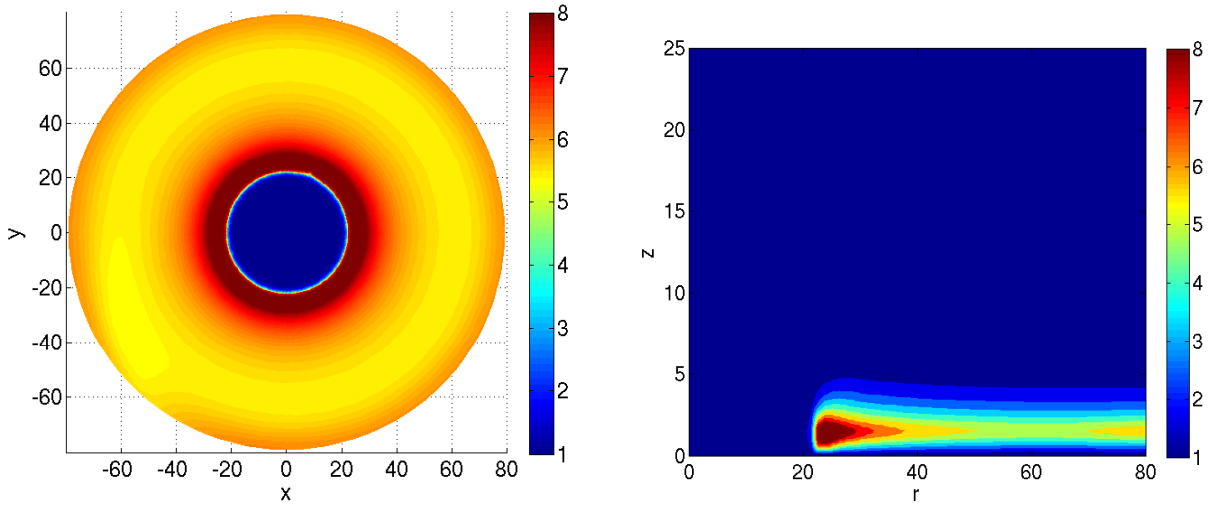
(a) (b)

Figure 4.15: Temperature contours of the flame hole in the xy and the rz planes at $D = 1.96$



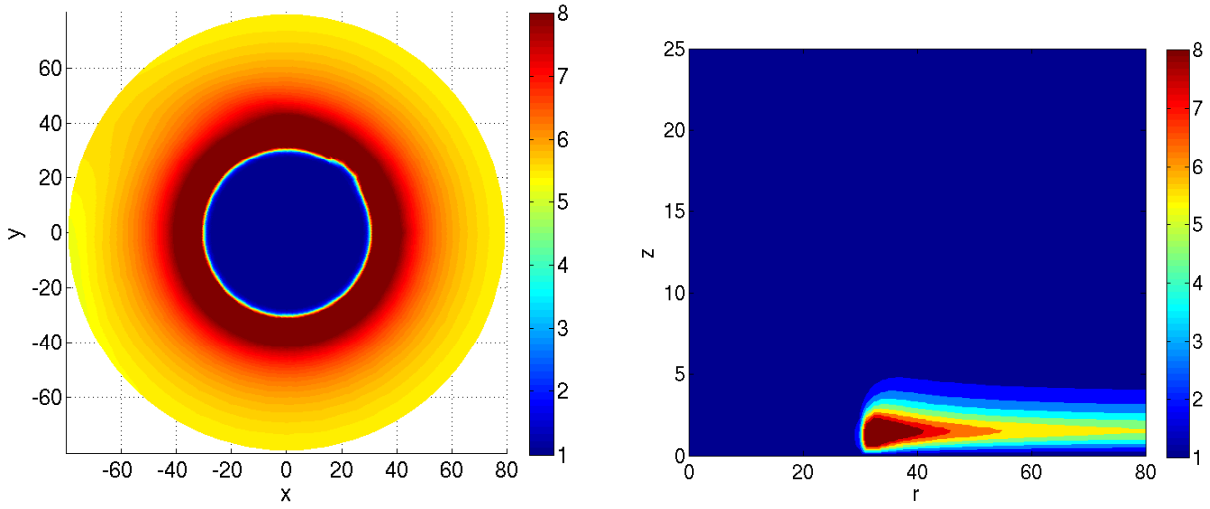
(a) (b)

Figure 4.16: Temperature contours of the flame hole in the xy and the rz planes at $D = 1.955$



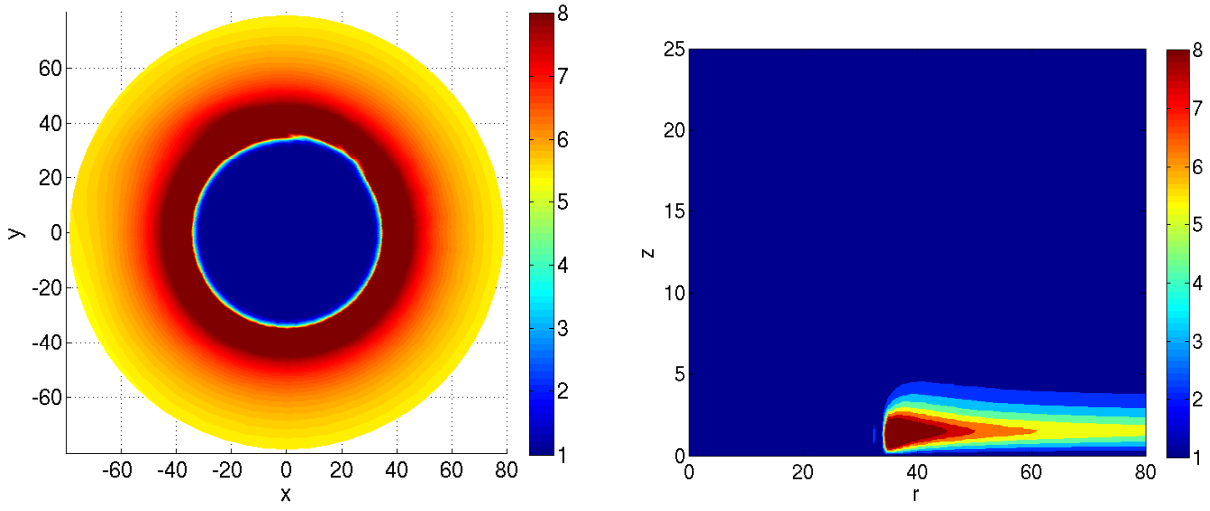
(a) (b)

Figure 4.17: Temperature contours of the flame hole in the xy and the rz planes at $D = 1.95$



(a) (b)

Figure 4.18: Temperature contours of the flame hole in the xy and the rz planes at $D = 1.945$



(a) (b)

Figure 4.19: Temperature contours of the flame hole in the xy and the rz planes at $D = 1.94$

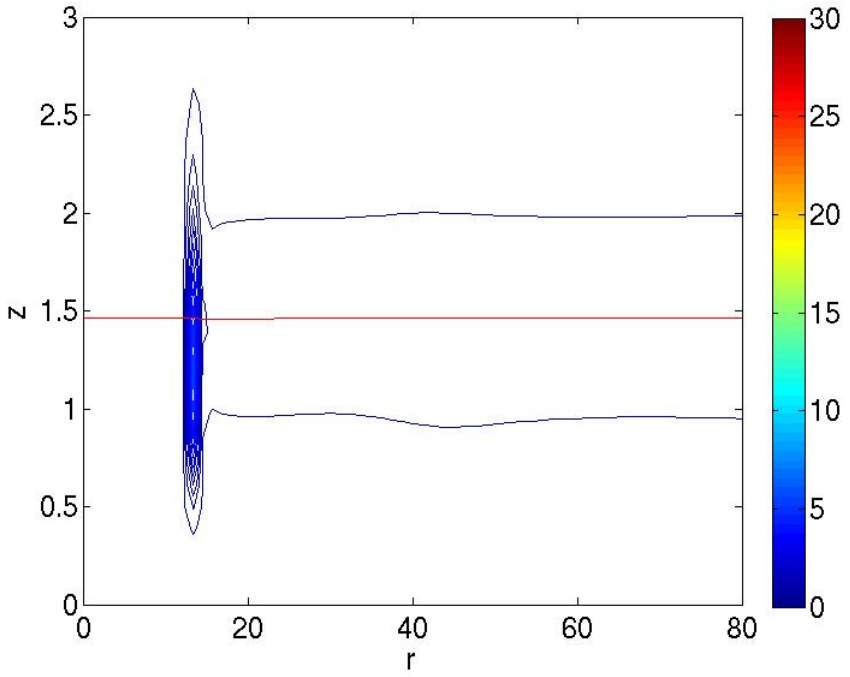


Figure 4.20: Reaction rate contours at D=1.96

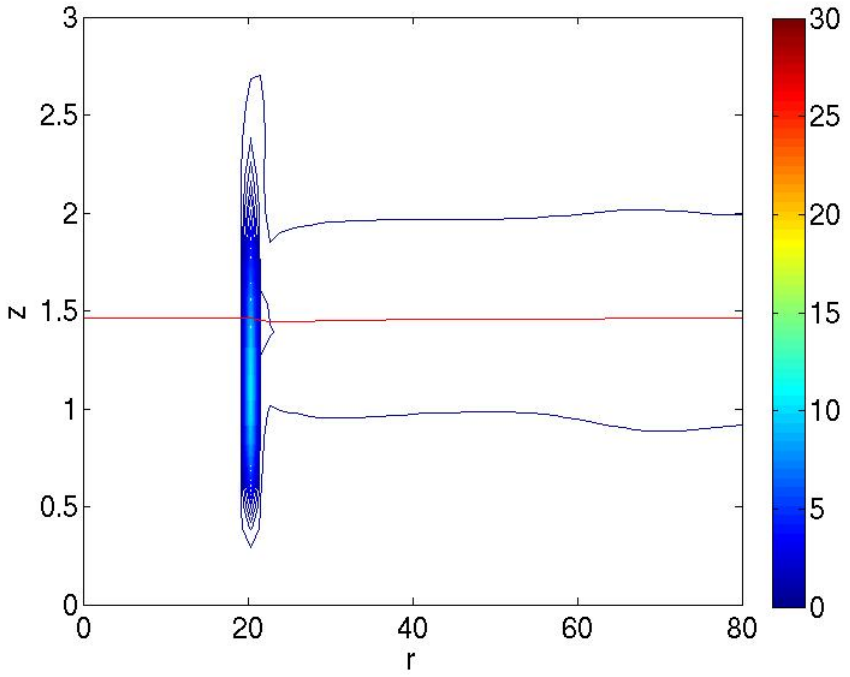


Figure 4.21: Reaction rate contours at D=1.955

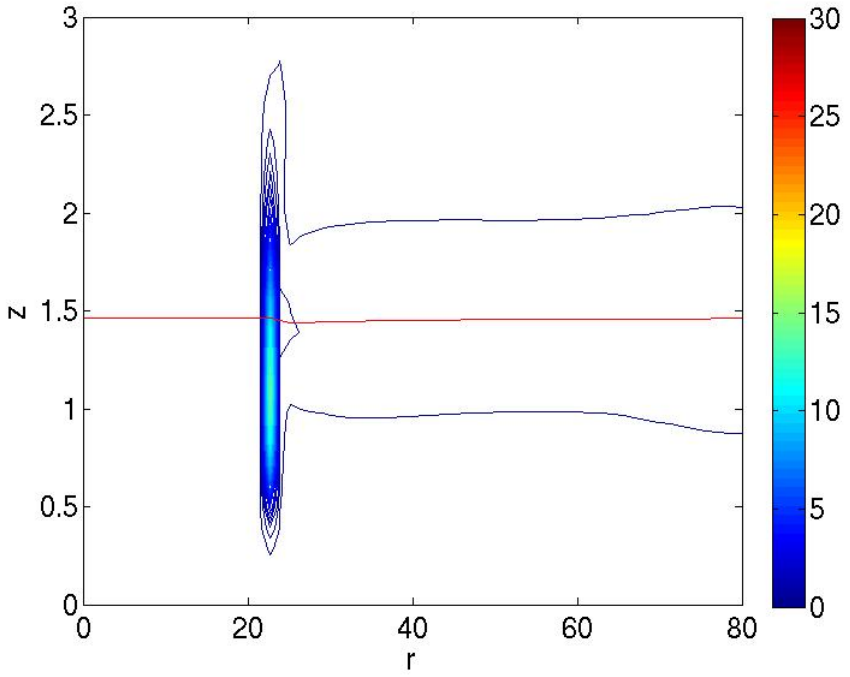


Figure 4.22: Reaction rate contours at D=1.95

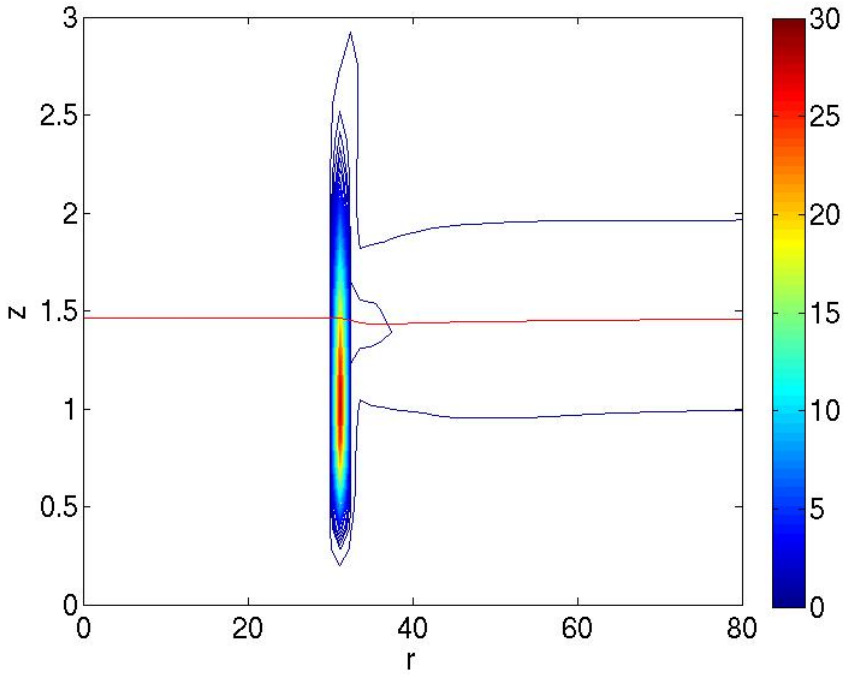


Figure 4.23: Reaction rate contours at D=1.945

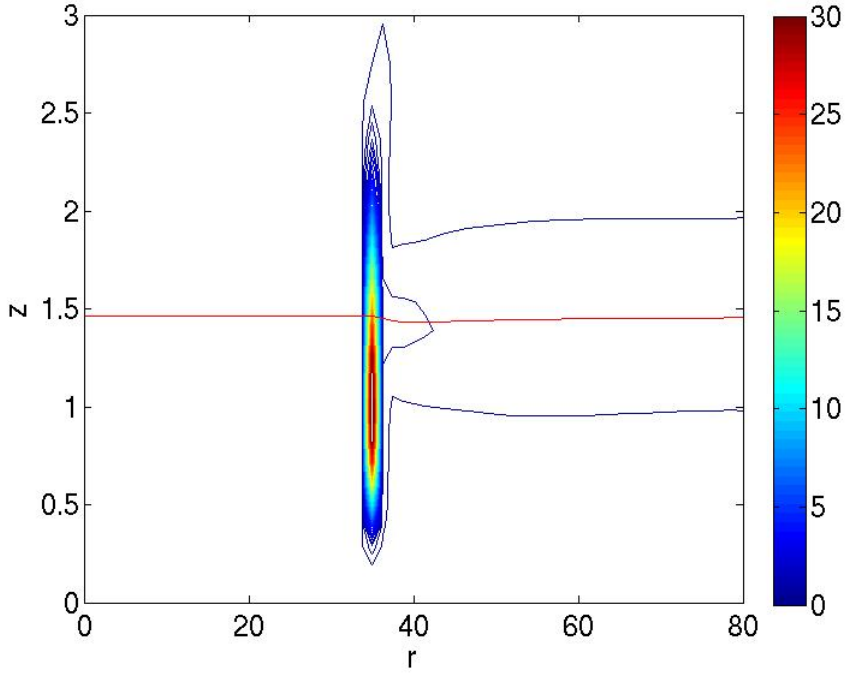


Figure 4.24: Reaction rate contours at D=1.94

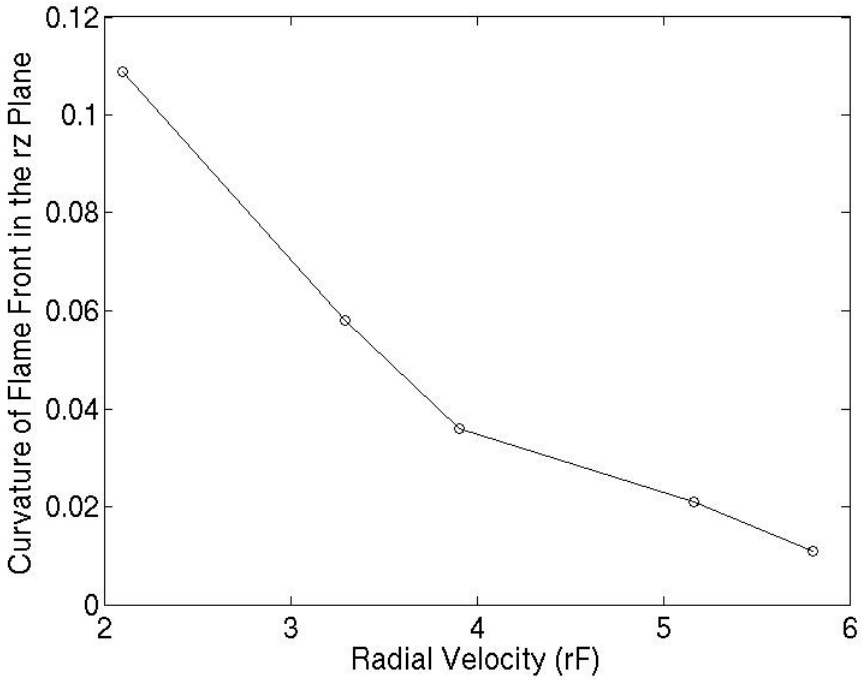


Figure 4.25: Curvature of flame front in the rz plane at stoichiometric vs the radial velocity

4.4 Single Spiral

Figure 4.26 shows the temperature and species contours for a single spiral for $D = 1.91$, at three different times. The spiral rotates as a rigid body at a rate of $-0.14\ rad/time$, while the burner rotates at $b_0 = 1\ rad/time$ and at the stoichiometric height the angular velocity is $b = 0.36\ rad/time$. This high relative angular velocity is believed to be one of the main factors contributing to the differences in the characteristics of the leading edge as compared to the the trailing one. As seen from figure 4.8 the temperature gradient is steep at the leading edge while there is a more gradual variation at the tail. This trend is similar to the one seen in the case of the flame hole where the stationary "leading edge" displays a high temperature gradient. Another point of note are the wiggles that are present at the leading edge of the spiral. These oscillations arise due to difficulties in fully resolving the dynamic edge of the flame. It is found that increasing the grid density reduces the prominence of these oscillations, however, due to time constraints, the finer grid is not used.

The shape of the spiral is affected by the velocity field as well. In the $r\theta$ plane the temperature profile can be described as a wave that is advected in θ. The line connecting the peak temperatures of these waves at each r location is at an angle that can be approximated by the resultant of the radial and angular velocities. This is demonstrated in figure 4.28 where the velocity field is superimposed on the temperature contours. As seen here the velocity vectors are almost tangent to the temperature contours. This is further demonstrated in figure 4.29 where the slope of the leading edge in the $r\theta$ plane is plotted against r. As seen here, the slope is linearly proportional to r, other than toward the radial boundary.

As in the case of the flame holes, both the fuel and oxidizer seem to be consumed only at

the regions of high temperature, i.e. the flame region. It is also apparent from the contours in the $z\theta$ plane given in figure 4.27 that there is very little mixing in the flame region suggesting that the flame is primarily a diffusion flame. In the simulations, the geometry of the flames do not change once the spiral shape is assumed. Simulations have been run for about four revolutions without any significant changes in the flame geometry.

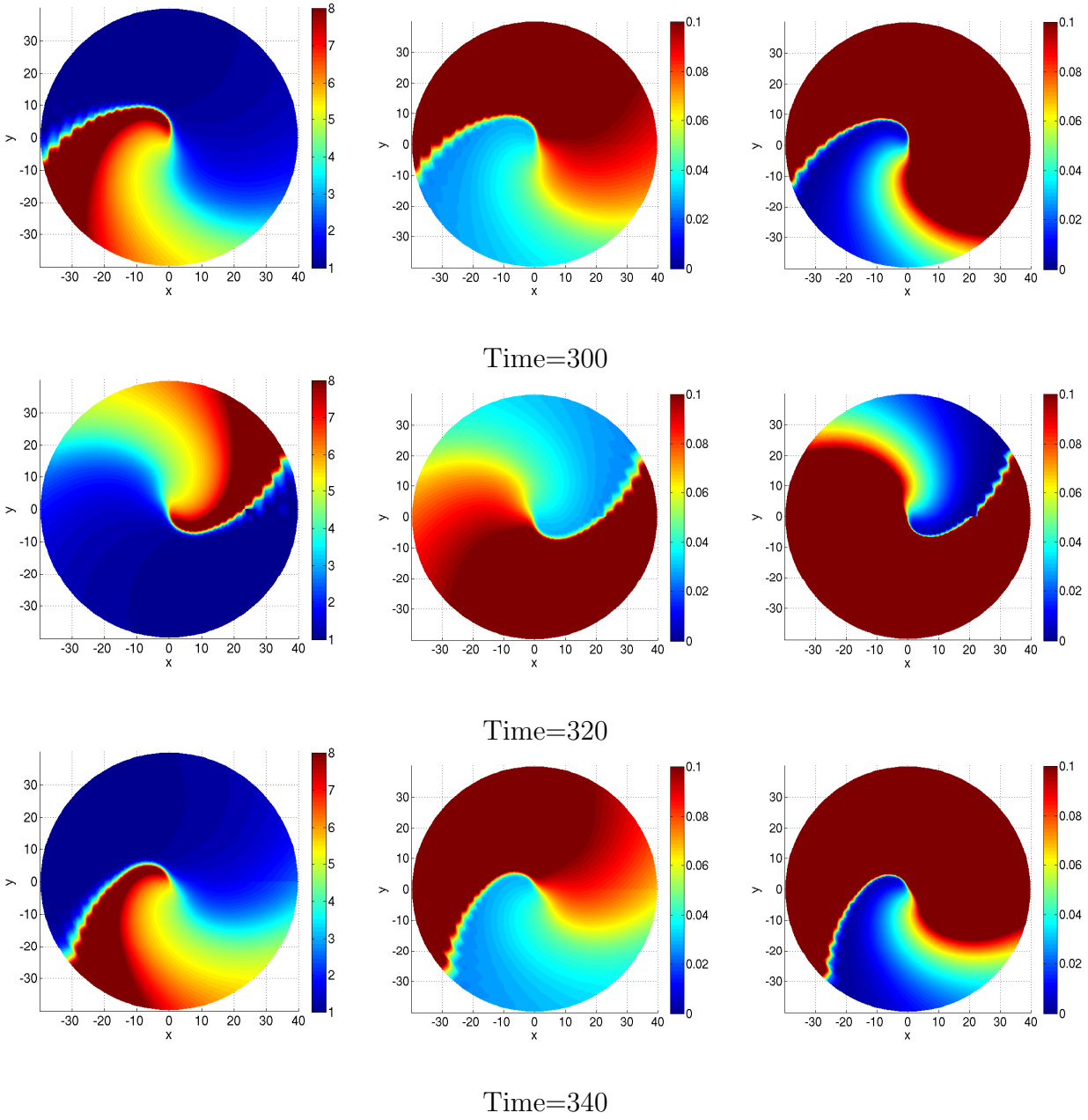
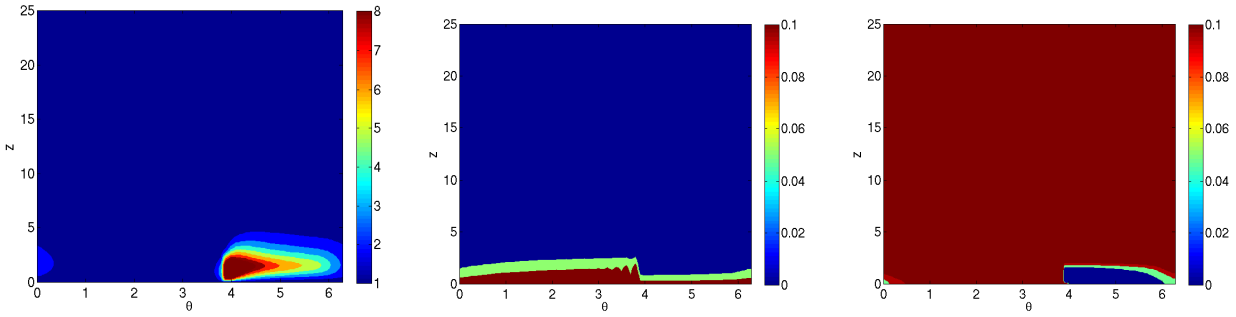


Figure 4.26: Temperature and species contours of a single spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.91$ at $time = 300,\ \ 320,\ \ 340$



T Y_f Y_o

Figure 4.27: Temperature and species contours in the $z\theta$ plane of a single spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.91$ at $time = 340$

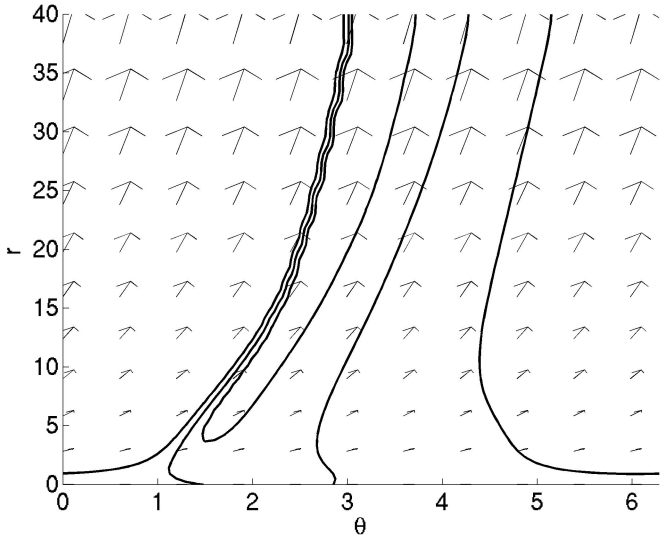


Figure 4.28: Single spiral contours and relative velocity vectors for case 1, $D = 1.91$. Contour Values:4, 6, 8.

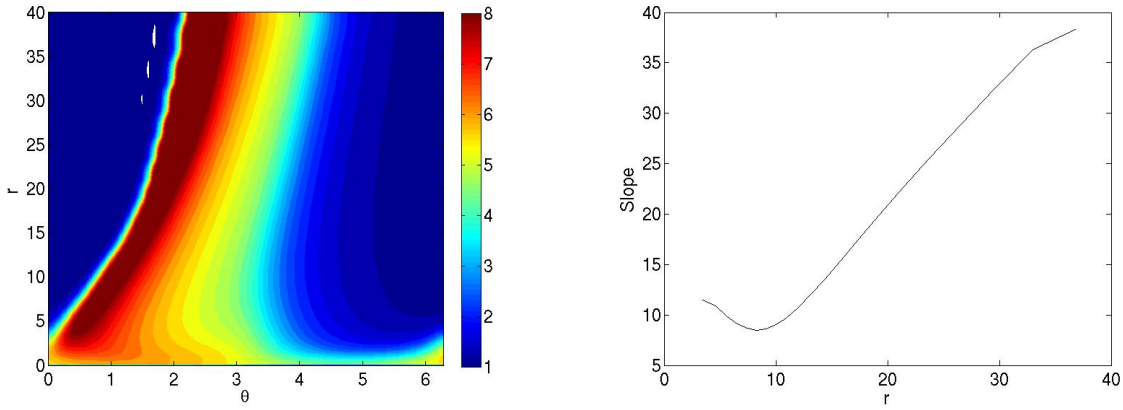


Figure 4.29: Slope of leading edge as function of r

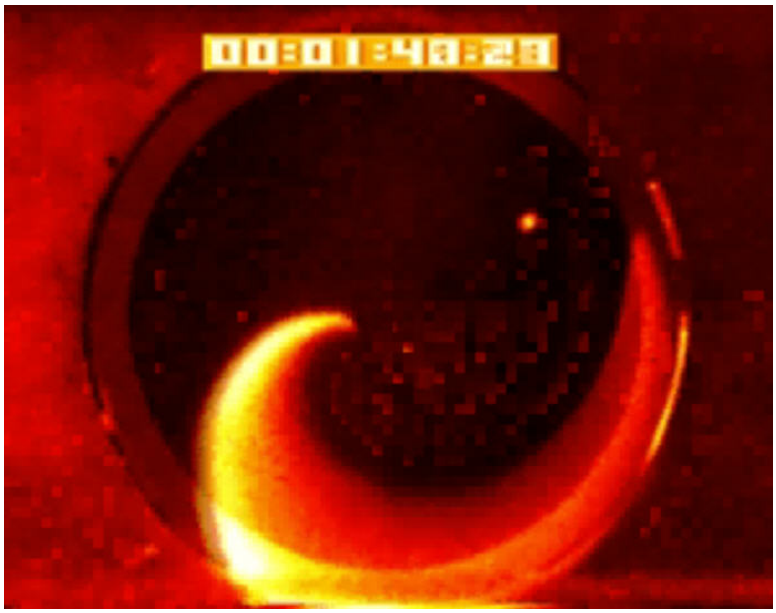


Figure 4.30: Single spiral observed in the experiments of Nayagam and Williams (Printed with permission).

4.4.1 Comparison with Experiment

The single spirals simulated in this study are qualitatively similar to the one observed in the experiments. A snapshot of one such spiral is shown in figure 4.30 (obtained in a private communication with Vedha Nayagam and printed with permission). This spiral rotates clockwise and the temperature gradients at the leading edge are steeper than those in the trailing one. One primary difference between the experimental results and the simulated ones is the absence of tip meandering in the simulated spirals. This difference could be attributed to hydrodynamics effects which are ignored in the current study. However, the qualitative similarities between the experimental and numerical spirals suggest that the inception of these non-uniform flames is a result of thermo-diffusional instabilities although their propagation may be affected by changes in the flow field.

4.4.2 Effect of Damköhler Number on the Single Spiral

Figure 4.31 shows single spirals simulated for the parameter values of section 4.2 at Damköhler numbers of 1.90, 1.88 and 1.86 while figure 4.32 shows an overlay plot. As seen here, the shape of the spirals are similar but the area between the leading and trailing edges of the spirals decreases slightly as the Damköhler number decreases. The angular velocity of all three spirals is approximately 0.14 $rad/time$. Therefore, the Damköhler number does not have a discernable effect on the angular velocity of the single spiral. It appears that if the Damköhler number is in the range within which single spirals can be sustained, the spirals may form due to the appropriate perturbations and, the dynamics and geometry of the single spiral are primarily functions of the velocity field as represented by the similarity variables.

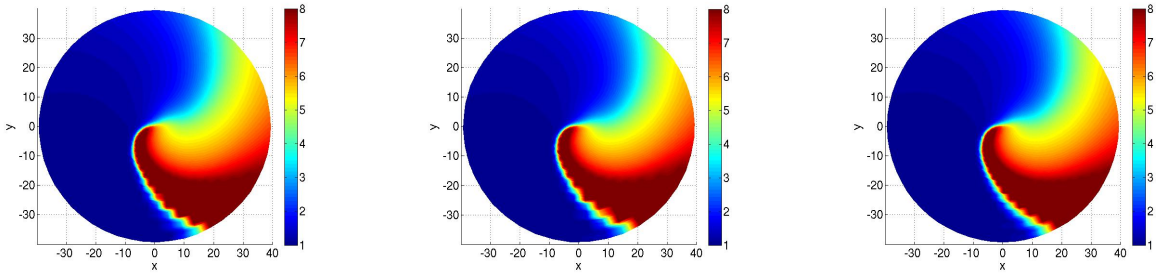


Figure 4.31: Temperature contours of single spirals for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 40$ at a. $D = 1.91$, b.$D = 1.88$ and c.$D = 1.86$

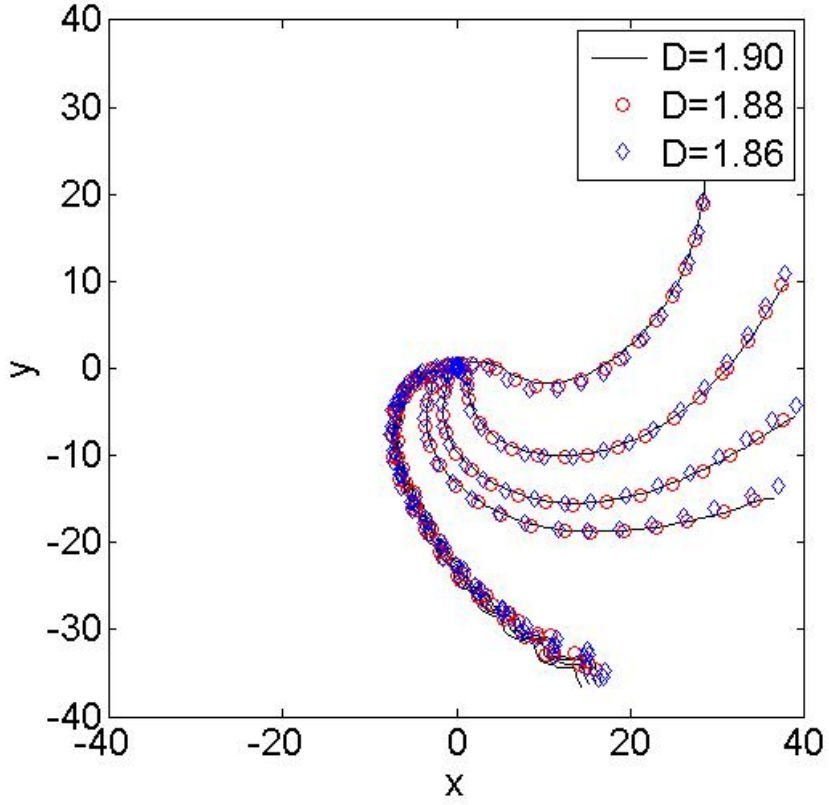


Figure 4.32: Overlay of spirals for $D = 1.90$, $D = 1.88$ and $D = 1.86$

4.5 Double Spiral

The double spirals in figure 4.33 are for $D = 1.85$ at times 120, 140 and 160. As in the case of the single spiral, the leading edge of the flame relative to the flow, is the one with the steeper temperature gradients and the temperature contours in the $z\theta$ plane, as shown in figures 4.34. Also, each arm of the double spiral is similar in shape to the single spiral discussed in the previous section. However, in contrast to the single spiral, the double spiral rotates in the same direction as the burner, and the curvature of the leading edge is concave. These differences in the dynamics and the shape of the spirals in the xy plane is attributed to interactions between the two arms of the spiral as depicted in the temperature contours shown in figure 4.34, where some tip interaction is apparent. Thus, along with the Damköhler number and the velocity profile, interactions between the spirals also affect the shape and dynamics of these flames.

It should also be mentioned that it appears from the flame contours in figures 4.34 and 4.12 that the shape of the flame bars in the $z\theta$ plane are equivalent although there is a curvature difference in the flame shape in the xy plane. This suggests that the mode of the instability only affects the lateral dynamics of the flame.

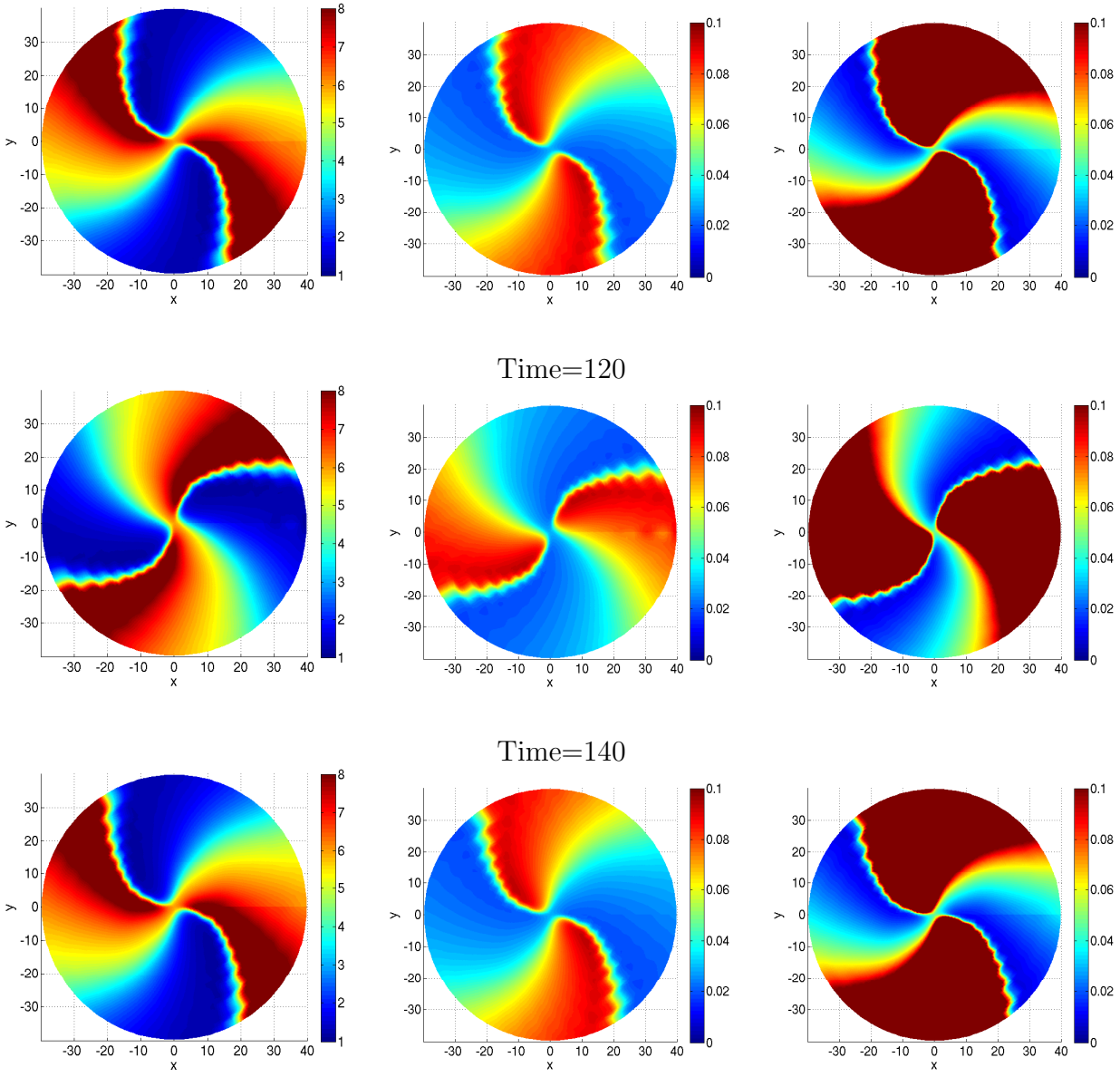
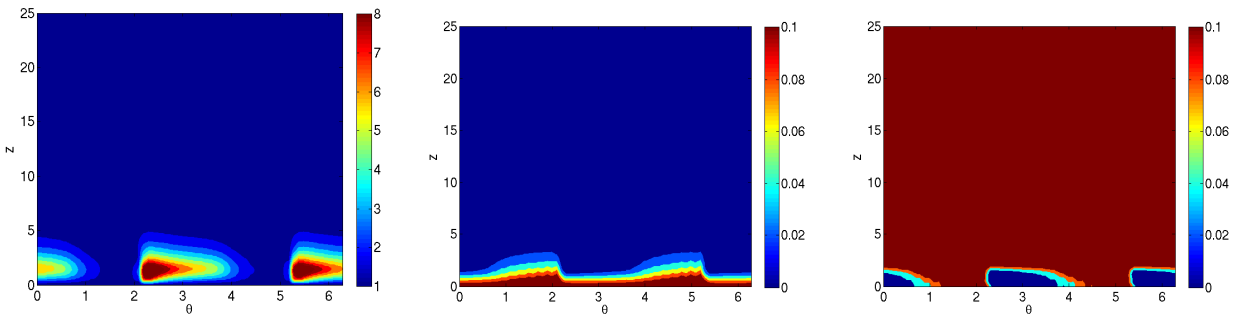


Figure 4.33: Temperature and species contours of a double spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.85$ at $time = 120,\ \ 140,\ \ 160$



T Y_f Y_o

Figure 4.34: Temperature and species contours in the $z\theta$ plane of a double spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.85$ at $time = 160$

4.6 Non-uniform Flames at a Mixture Strength Value of 5.0

Non-uniform flames are also simulated for a pure methane burner. The parameter values for the results in this section are, $\phi = 5.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.05$, $T_s = 1.0$, $\beta = 50$, and $Z_e = 5.43$. Figures 4.35 and 4.36 show the velocity profiles and the S-shaped response curve for this set of parameters. The extinction point, D_E, is approximately 5.56 and $D^* = 6.70$. As in the previous section, the non-uniform flames observed in this section include flame holes, single spirals and double spirals as shown in figures 4.37, 4.38 and 4.39 respectively. The geometry of these flames is very similar to the ones in the previous section but the dynamics are different. The single spiral rotates clockwise at an angular velocity of 0.07 $rad/time$ while the double spiral rotates counterclockwise at an angular velocity of 0.75 $rad/time$. Also, the range of Damköhler numbers within which the instabilities occur is larger. This suggests that the injection velocity and the mixture strength affect the range of Damköhler numbers within which the non-uniform flames appear.

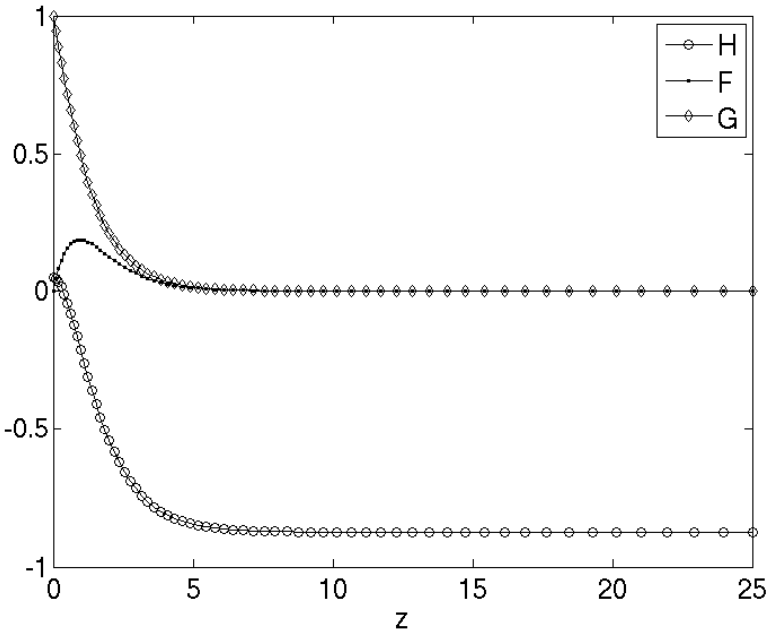


Figure 4.35: Non-dimensional Velocities

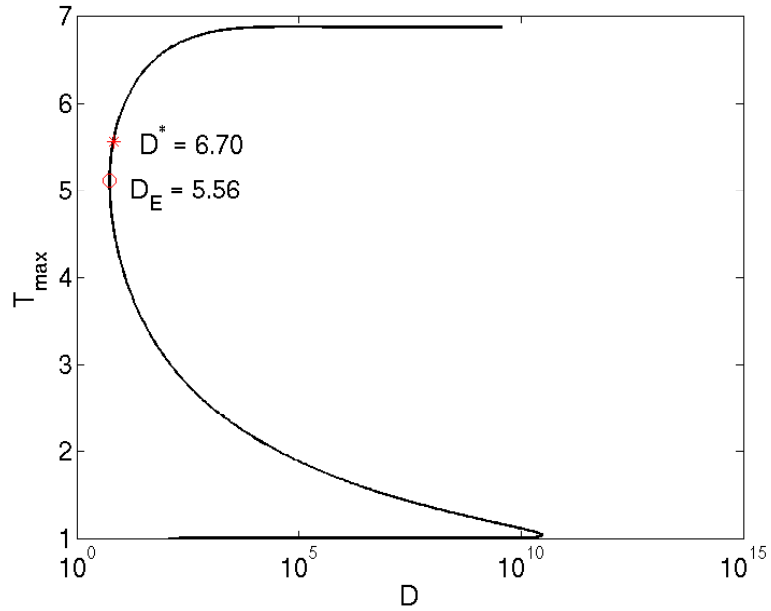


Figure 4.36: S-shaped Response

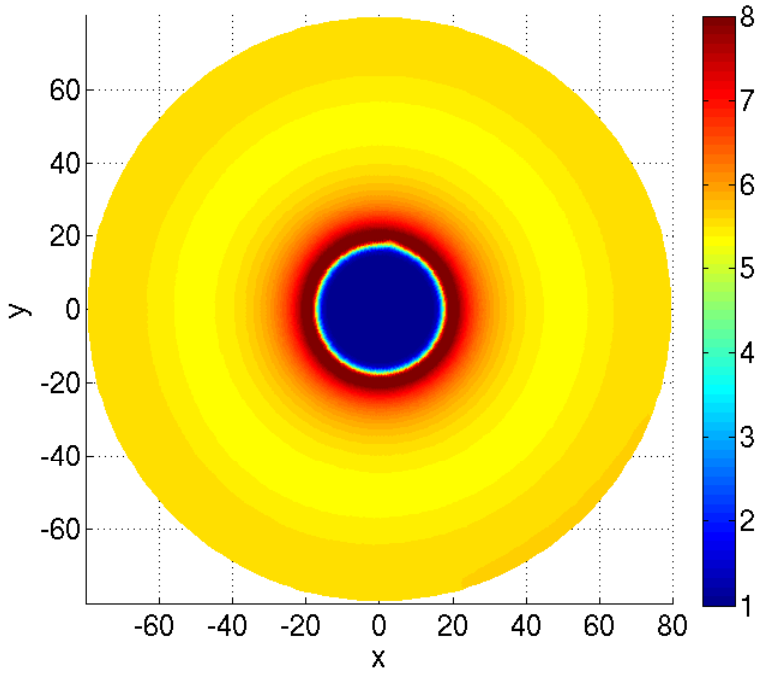


Figure 4.37: Temperature contour of a flame hole for $\phi = 5.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, f$\beta = 50$, $Z_e = 5.43$ and $D = 6.7$

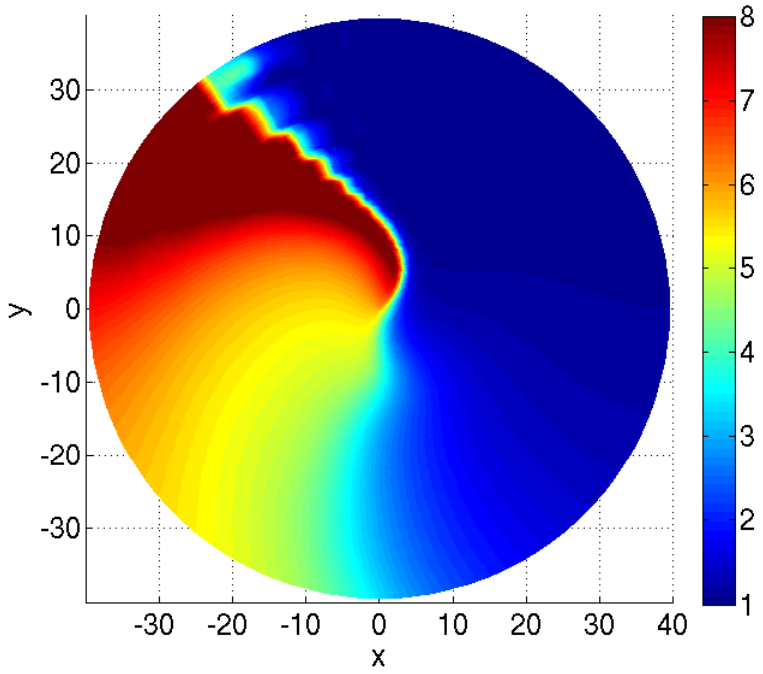


Figure 4.38: Temperature contour of a single spiral for $\phi = 5.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, f$\beta = 50$, $Z_e = 5.43$ and $D = 6.6$

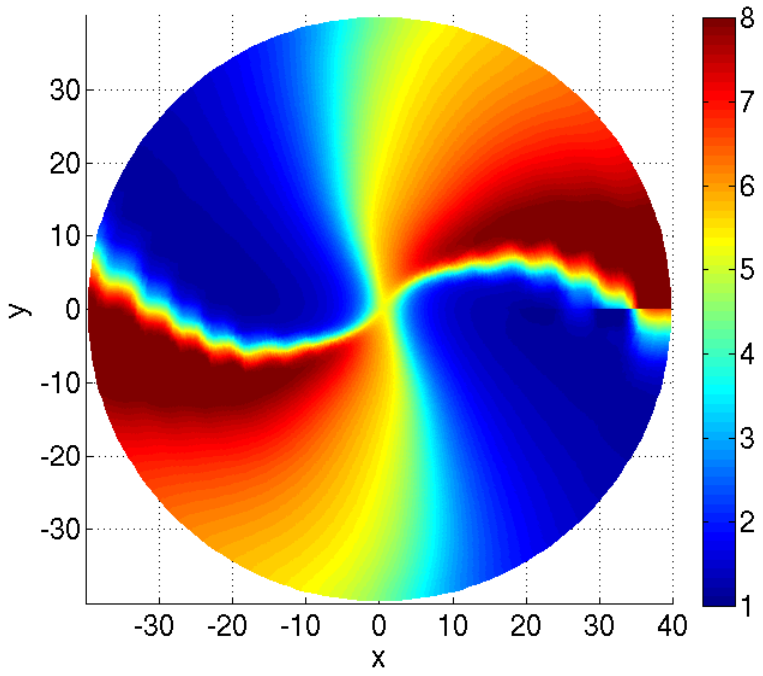


Figure 4.39: Temperature contour of a double spiral for $\phi = 5.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 50$, $Z_e = 5.43$ and $D = 6.2$

4.7 Effect of Parameters of Study on the Stability of the System

In the previous sections it has been shown that the Damköhler number is the crucial paramter affecting the stability of the system. However, other factors such as boundary conditions are also expected to have an effect, especially in terms of the range of Damköhler numbers within which non-uniform flames can be sustained. This is demonstrated in table 4.2 where the extinction Damköhler numbers are reported for four different parameter combinations. Here the heat release parameter is adjusted to fix the Burke-Schumann flame temperature. These values suggest that the injection velocity and the mixture strength have a considerable effect on the stability of the system while the effect of the temperature at the burner exit is small in comparison. Increasing the mixture strength leads to a shrinkage of the range whereas an increase in the injection velocity leads to an expansion. The trend in terms of the mixture strength is opposite to that observed by Kukuck and Matalon [18] for planar oscillations in diffusion flames. In the case of the spinning burner, an increase in the mixture strength leads to a shifting of the height of stoichiometry away from the burner surface to a region where the radial and angular velocities are lower in magnitude since the velocity profile remains unchanged. This reduction in the velocities in the vicinity of the flame sheet may have a stabilizing effect and so the Damköhler number range for instability is significantly reduced. An increase in the injection velocity, on the other hand, results in both a shift in the stoichiometric height and velocity profile. This dual change may be the cause of the expansion of the range.

Case	H_0	ϕ	T_s	D_E	D^*	D_E/D^*
1	0.10	2.0	1.0	1.48	1.96	0.76
2	0.10	2.0	1.4	1.46	1.95	0.75
3	0.10	5.0	1.0	1.78	1.87	0.95
4	0.05	5.0	1.0	5.25	6.32	0.83

Table 4.2: Effect of Parameters of Study on D^*.

4.8 Scalar and Cross Scalar Dissipation Rates of the Non-uniform Flames

The formation of flame patterns in the Von Karman boundary layer may have important implications for modeling turbulent flames, especially in terms of non-premixed and partially premixed turbulent combustion. Currently, turbulence models do not explicitly account for local vortices within a turbulent mixing layer. Flames in the vicinity of localized vortices may exhibit behavior similar to the flames in the boundary layer of a spinning burner. Thus it may be necessary to incorporate such behavior into existing turbulence models.

In turbulence modeling it is necessary to derive relationships between parameters such as a conserved scalar, that is not depleted by chemical reactions, and the reaction rate. The dissipation of the conserved scalar is a popular choice. In turbulent flames the conditional expectation of the dissipation of a conserved scalar, conditioned upon the value of the scalar, is proportional to the mean reaction rates [34]. Bilger [35] gives a relationship between the dissipation of a conserved scalar and the reaction rate valid in the fast chemistry limit,

$$w_i = \frac{1}{2}\rho\chi\frac{\partial^2 Y_i^e}{\partial Z^2}. \tag{4.11}$$

Although this model is only valid in the fast chemistry limit, dependencies like these are important and it is pertinent to derive similar relationships beyond the fast chemistry limit. Investigating the scalar dissipation profiles of the non-uniform flames which are typically a consequence of finite rate chemistry may provide some insight into expanding the scope of turbulence models. Thus, in this section, correlations between the dissipation of a conserved scalar and the flame patterns in the Von Karman boundary layer is explored.

The conserved scalars mentioned above are typically derived either from linear combinations of the species equations by elimination of the reaction rate term, or from elemental conservation. Linear relationships exist between all such conserved scalars if all the diffusivities are equal and there are two uniform reactant feeding streams [34],[36]. However, when the scalars are not linearly related the choice of the conserved scalar is dependent upon the particular problem.

The mixture fraction, a conserved scalar, is often utilized in the description of non-premixed combustion to determine the flame surface [37]. The expression for the mixture fraction is derived from the species governing equations by eliminating the reaction terms. For a reaction given by,

$$\nu'_F C_m H_n + \nu'_{O_2} O_2 \rightarrow \nu''_{CO_2} CO_2 + \nu''_{H_2O} H_2O$$

the mixture fraction is

$$Z = \frac{\nu Y_F - Y_{O_2} + Y_{O_2,2}}{\nu Y_{F,1} + Y_{O_2,2}}, \tag{4.12}$$

where, $\nu = \nu'_{O_2} W_{O_2} / \nu_F W_F$, the $W_i s$ are the molecular weights and the subscripts 1 and 2 represent values at the fuel supply and oxidizer boundaries respectively. The species mass fractions at the boundaries serve to normalize the mixture fraction so that it has a value of 1 at the fuel boundary and 0 at the oxidizer boundary. The stoichiometric mixture fraction is given by,

$$Z_{st} = \left[1 + \frac{\nu Y_{F,1}}{Y_{O_2,2}} \right]^{-1}. \tag{4.13}$$

In non-premixed combustion

$$Z(\vec{x}, t) = Z_{st} \tag{4.14}$$

is used to define the flame surface [37]. Steady combustion is achieved when a balance exists between the chemical heat release and the heat dissipated away from the stoichiometric surface by diffusion and convection. The effects of finite rate chemistry become evident when such a balance does not exist [38]. Hence it is helpful to define a diffusive time

$$\tau_\chi \approx \chi_s^{-1} \tag{4.15}$$

where χ_s is the scalar dissipation rate at the stoichiometric surface. The scalar dissipation rate represents a diffusion time scale imposed by the mixing field and is given by

$$\chi = 2D\nabla Z.\nabla Z, \tag{4.16}$$

where Z is the conserved scalar. Flame quenching occurs when the value of the instantaneous scalar dissipation rate exceeds a threshold, χ_q. The value of χ_q is obtained from the solution of one-dimensional steady strained diffusion flames and is used as a reference point for quenching in turbulent combustion modeling [38]. It is important to note that simulation results of Favier and Vervisch [38] indicate that the scalar dissipation rate needed to initiate quenching is higher than that necessary to maintain or propagate a quenched zone.

Another important factor that needs to be considered is the increase in the product of the reactant mass fractions in the regions bordering the quenched zones [38]. Under stoichiometric conditions this suggests that the flame itself may be a combination of partially premixed and non-premixed regions. Thus, it is useful to identify premixed regions so as to fully understand the structure of the flame and the quenching mechanism. The cross scalar dissipation rate is a parameter that may be used to distinguish between premixed and diffusion flames [38],[39]. The cross scalar dissipation rate

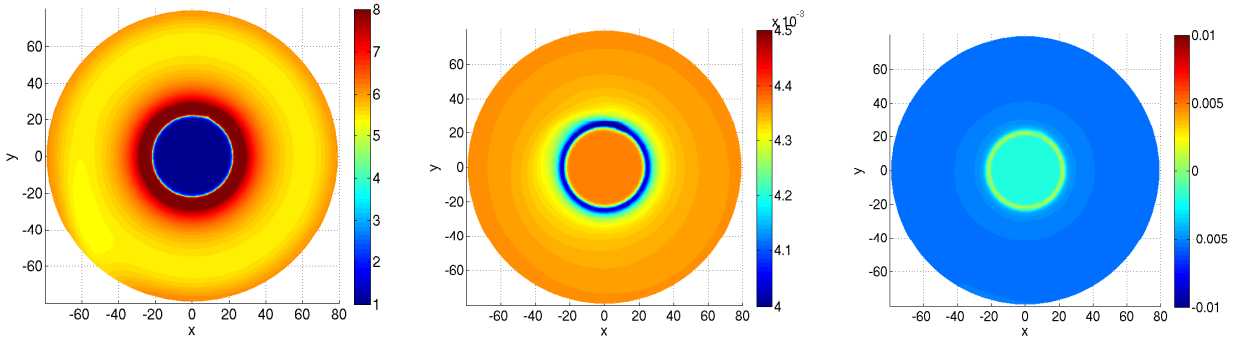
$$\chi_c = 2D\nabla Y_f.\nabla Y_o \tag{4.17}$$

is positive for a premixed flame and negative for a diffusion flame [38].

Figures 4.40-4.42 show the contours of the instantaneous scalar dissipation rates, and the cross scalar dissipation rates for a flame hole, a single spiral and a double spiral, in the xy plane. These contours are at the plane where the mass fractions are in stoichiometric proportions. It is evident from these figures that the instantaneous scalar dissipation rate is generally higher in the regions where quenching has occurred. In comparison to the temperature contours, in the quenched regions, the scalar dissipation rate has a value higher than 0.00435 for all three flames. Another interesting characteristic that is common among the flames is that regions of premixing, as indicated by positive values of χ_c are thin and are limited to regions bordering those with low scalar dissipation rates. With respect to the temperature contours these premixed regions are in areas where the temperature transitions rapidly from hot to cold values.

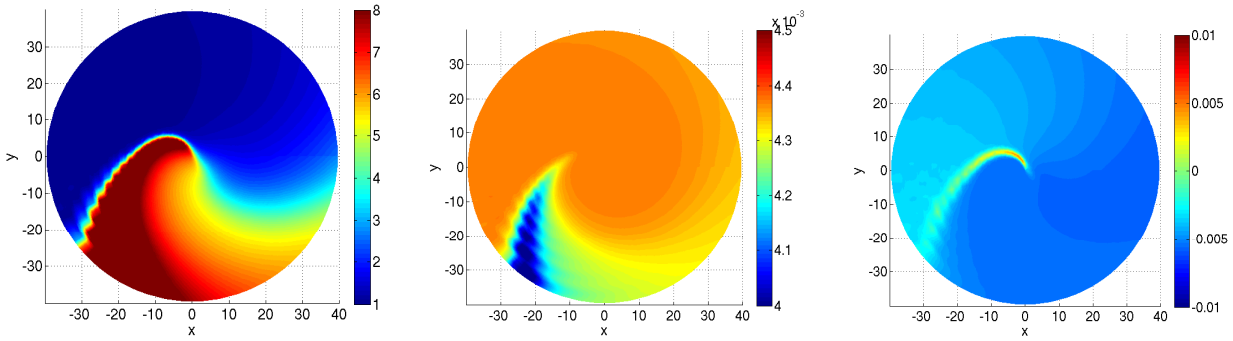
The curvatures of the contours of both the scalar dissipation and the cross scalar dissipation rates are similar to those of the temperature. It is also important to note that the maximum positive value of the cross scalar dissipation rate is higher in the case of the single spiral than that for the flame hole or the double spiral. This may be attributed to the higher angular velocity with which the single spiral rotates about the axis of the burner.

The contours of the scalar dissipation and the cross-scalar dissipation provide insight in terms of the structure of the flame and their relationship to the dissipation away from the stoichiometric surface. In other words, the chemical reaction in the quenched regions is not sufficient to balance the dissipation away from the stoichiometric plane. This leads to flame extinction because the Damköhler number at which these flame patterns are observed are



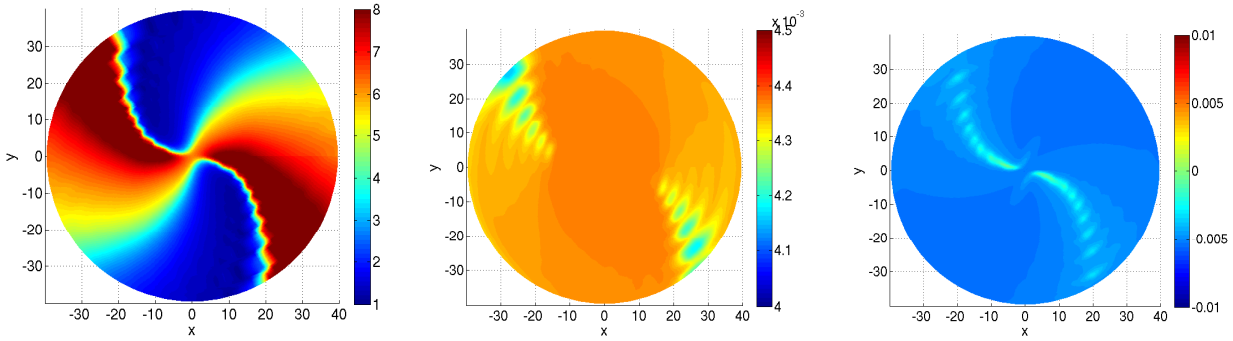
a. T b. χ c.χ_c

Figure 4.40: Temperature, scalar and cross scalar contours in the xy plane of a flame hole for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.95$



a. T b. χ c.χ_c

Figure 4.41: Temperature, scalar and cross scalar contours in the xy plane of a single spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.9$



a. T b. χ c.χ_c

Figure 4.42: Temperature, scalar and cross scalar contours in the xy plane of a double spiral for $\phi = 2.0$, $Y_{o\infty} = 0.2$, $H_0 = 0.1$, $T_s = 1.0$, $\beta = 25$, $Z_e = 5.43$ and $D = 1.85$

low enough such that finite rate chemistry can be expected. The largest Damköhler number at which a flame pattern can be sustained could thus be considered as the transition point where the chemistry is no longer infinitely fast.

Within the range of finite chemistry, however, several distinct flame patterns have been observed. These flame patterns are both functions of the nature of the perturbation and the Damköhler number. The scalar and cross scalar dissipation, however, do not provide much information about the dynamics of these flames. Therefore, it may be necessary to include other pertinent parameters in models for representing the effect of localized vortices in turbulent combustion. So further analysis is necessary to ascertain the driving factor for the dynamics of the flame patterns observed in this study.

4.9 Conclusion

In this study a simple model is utilized to gain some insights into the stability of diffusion flames supported by a spinning porous plug methane burner. The evolution of the flame holes and spirals indicate that diffusion flames supported by the rotating porous plug burner become unstable at near extinction Damköhler numbers on the upper branch of the S-response curve. It is shown that these multidimensional instabilities are thermo-diffusional in nature. The flames have three-dimensional characteristics and the spirals are dynamic in nature. They rotate about the axis of the burner and the single-armed spiral rotates clockwise while the double-armed spiral rotates counter clockwise. The primary factors affecting the shape and dynamics of the spirals are identified as the velocity field and the interaction between spiral flames. It is also found that the mixture strength and the injection velocity have a significant effect on the range of Damköhler numbers within which the system is prone to instability.

Appendix A

Nondimensionalizing the Governing Equations

Continuity

$$\frac{\partial \rho}{\partial t} + \frac{1}{r^*}\frac{\partial(\rho r^* u)}{\partial r^*} + \frac{1}{r^*}\frac{\partial(\rho v)}{\partial \theta} + \frac{\partial(\rho w)}{\partial z^*} = 0 \tag{A.1}$$

For steady flow with radial symmetry,

$$\frac{1}{r^*}\frac{\partial(\rho r^* u)}{\partial r^*} + \frac{\partial(\rho w)}{\partial z^*} = 0 \tag{A.2}$$

For constant density,

$$\frac{1}{r^*}\frac{\partial(r^* u)}{\partial r^*} + \frac{\partial(w)}{\partial z^*} = 0 \tag{A.3}$$

Using nondimensional variables:

$$r = \sqrt{\frac{b^*}{\nu_\infty}} r^*, \quad z = \sqrt{\frac{b^*}{\nu_\infty}} z^*, \quad \overline{T} = \frac{T}{T_\infty}, \quad \overline{Y_i} = \frac{Y_i}{Y_{f0}}$$

where, b^* is the angular velocity of the burner.

We seek a similarity solution of the form:

$$u = b^* r^* F, \quad v = b^* r^* G, \quad w = \sqrt{b^* \nu_\infty} H, \quad p = \rho b^* \nu_\infty P$$

where F, G, H and P are functions of z.

Non-dimensionalizing the equation we get,

$$\begin{aligned} &\frac{1}{r} b^* r F + b^* F + b^* r \frac{\partial F}{\partial r} + \frac{\partial}{\partial z} \sqrt{b^* \nu_\infty} \sqrt{\frac{b^*}{\nu_\infty}} H = 0 \\ &\rightarrow 2F + H' = 0 \end{aligned} \tag{A.5}$$

Momentum

r-Momentum:

Incompressible flow with constant transport properties,

$$\begin{aligned}&\rho\left(\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial r^*}+\frac{v}{r^*}\frac{\partial u}{\partial \theta}-\frac{v^2}{r^*}+w\frac{\partial u}{\partial z^*}\right)\\&=-\frac{\partial p}{\partial r^*}+\mu\left[\frac{\partial}{\partial r^*}\left(\frac{1}{r^*}\frac{\partial (r^*u)}{\partial r^*}\right)+\frac{1}{r^{*2}}\frac{\partial^2 u}{\partial \theta^2}-\frac{2}{r^{*2}}\frac{\partial v}{\partial \theta}+\frac{\partial^2 u}{\partial z^{*2}}\right]\end{aligned} \tag{A.6}$$

Using the steady equation with radial symmetry,

$$u\frac{\partial u}{\partial r^*}-\frac{v^2}{r^*}+w\frac{\partial u}{\partial z^*}=-\frac{\partial p}{\partial r^*}+\nu_\infty\left[\frac{\partial}{\partial r^*}\left(\frac{1}{r^*}\frac{\partial r^*u}{\partial r^*}\right)+\frac{\partial^2 u}{\partial z^{*2}}\right] \tag{A.7}$$

Using the similarity variables defined in Equation A:

$$\begin{aligned}&b^{*2}r^*F^2+b^{*2}r^*HF'-b^{*2}r^*G^2+\frac{b^*F}{r^*}-\frac{b^*F}{r^*}=\nu_\infty\frac{b^{*2}r^*F''}{\nu_\infty}\\&\therefore F''=F^2+F'H-G^2\end{aligned} \tag{A.8}$$

θ-Momentum: Steady equation with radial symmetry,

$$\rho\left(u\frac{\partial v}{\partial r^*}-w\frac{\partial v}{\partial z^*}+\frac{1}{r^*}uv\right)=\mu\left[\frac{\partial^2}{\partial r^*}\left(\frac{1}{r^*}\frac{\partial r^*u}{\partial r^*}\right)+\frac{\partial^2 u}{\partial z^{*2}}\right] \tag{A.9}$$

Using similarity variables defined in Equation A:

$$\begin{aligned}&r^*b^*FG+b^*r^*HG'+b^*r^*FG=\nu_\infty\left(\frac{b^*G}{r^*}-\frac{b^*G}{r^*}+\frac{b^*r^*}{\nu_\infty}G''\right)\\&\therefore G''=2FG+HG'\end{aligned} \tag{A.10}$$

z-Momentum: Steady equation with radial symmetry,

$$\rho\left(u\frac{\partial w}{\partial r^*}-w\frac{\partial w}{\partial z^*}\right)=-\frac{\partial p}{\partial z^*}+\mu\left[\frac{1}{r^*}\frac{\partial}{\partial r^*}\left(r^*\frac{\partial w}{\partial r^*}\right)+\frac{\partial^2 w}{\partial z^{*2}}\right] \tag{A.11}$$

Using similarity variables defined in Equation A:

$$\begin{aligned}
&b^*\sqrt{b^*\nu_\infty}HH'=\frac{1}{\rho}\left[-\rho b^*\sqrt{b^*\nu_\infty}P'+b^*\sqrt{b^*\nu_\infty}H''\right]\\
&\rightarrow H''-HH'-P'=0\\
&\therefore 2HF-2F'=P'
\end{aligned} \tag{A.12}$$

Energy Equation

$$\rho C_p \left(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial r^*} + \frac{v}{r^*}\frac{\partial T}{\partial \theta} + w\frac{\partial T}{\partial z^*} \right) - \left(\frac{\partial P}{\partial t} + u\frac{\partial P}{\partial r^*} + \frac{v}{r^*}\frac{\partial P}{\partial \theta} + w\frac{\partial P}{\partial z^*} \right)$$
$$-\lambda \left[\frac{1}{r^*}\frac{\partial}{\partial r^*}\left(r^*\frac{\partial T}{\partial r^*} \right) + \frac{1}{r^{*2}}\frac{\partial^2 T}{\partial \theta^2} + \frac{\partial^2 T}{\partial z^{*2}} \right] = -\sum \omega_i \Delta h, i$$
$$-\frac{1}{r^*}\left[\frac{\partial}{\partial r^*}\left(r^*\rho T \sum C_{pi} Y_i V i r^* \right) + \frac{\partial}{\partial \theta}\left(\rho T \sum C_{pi} Y_i V i \theta \right) + \frac{\partial}{\partial z^*}\left(\rho T \sum C_{pi} Y_i V i z^* \right) \right]$$
$$+2\left[\left(\frac{\partial u}{\partial r^*}^2 \right) + \left(\frac{1}{r^*}\frac{\partial v}{\partial \theta} + \frac{u}{r^*} \right)^2 + \left(\frac{\partial w}{\partial z^*} \right)^2 \right] + \left[r^*\frac{\partial}{\partial r^*}\left(\frac{v}{r^*} \right) + \frac{1}{r^*}\frac{\partial u}{\partial \theta} \right]^2$$
$$+\left[\frac{1}{r^*}\frac{\partial w}{\partial \theta} + \frac{\partial v}{\partial v}\partial z^* \right] + \left[\frac{\partial u}{\partial z^*} + \frac{\partial w}{\partial r^*} \right]^2 - \frac{2}{3}\left[\frac{1}{r^*}\frac{\partial}{\partial r^*}(r^* u) + \frac{1}{r^*}\frac{\partial v}{\partial \theta} + \frac{\partial w}{\partial z^*} \right]^2 \tag{A.13}$$

For low mach number flows:

$$\rho C_p \left(\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial r^*} + \frac{v}{r^*}\frac{\partial T}{\partial \theta} + w\frac{\partial T}{\partial z^*} \right) - \lambda \left(\frac{1}{r^*}\frac{\partial T}{\partial r^*} + \frac{\partial^2 T}{\partial r^{*2}} + \frac{1}{r^{*2}}\frac{\partial^2 T}{\partial \theta^2} + \frac{\partial^2 T}{\partial z^{*2}} \right) = Q\Omega \tag{A.14}$$

Nondimensionalizing,

$$\frac{\partial \overline{T}}{\partial \tau} + rF\frac{\partial \overline{T}}{\partial r} + G\frac{\partial \overline{T}}{\partial \theta} + H\frac{\partial \overline{T}}{\partial z} - \frac{1}{Pr}\left(\frac{1}{r}\frac{\partial \overline{T}}{\partial r} + \frac{\partial^2 \overline{T}}{\partial r^2} + \frac{1}{r^2}\frac{\partial^2 \overline{T}}{\partial \theta^2} + \frac{\partial^2 \overline{T}}{\partial z^2} \right) = \overline{\Omega} \tag{A.15}$$

Species Equation

$$\rho\left(\frac{\partial Y_i}{\partial t}+u\frac{\partial Y_i}{\partial r^*}+\frac{v}{r^*}\frac{\partial Y_i}{\partial \theta}+w\frac{\partial Y_i}{\partial z^*}\right)+\frac{1}{r^*}\frac{\partial}{\partial r^*}(r^*\rho Y_i V_{ir})+\frac{1}{r^*}\frac{\partial}{\partial \theta}(\rho Y_i V_{i\theta})+\frac{\partial}{\partial z^*}(\rho Y_i V_{iz^*})=-\alpha_i\Omega \tag{A.16}$$

where,

$$V_{ir^*}=-\frac{D}{Y_i}\frac{\partial Y_i}{\partial r^*},\qquad V_{i\theta}=-\frac{D}{Y_i r}\frac{\partial Y_i}{\partial \theta},\qquad V_{iz^*}=-\frac{D}{Y_i}\frac{\partial Y_i}{\partial z^*}$$

Simplifying,

$$\rightarrow \rho\left(\frac{\partial Y_i}{\partial t}+u\frac{\partial Y_i}{\partial r^*}+\frac{v}{r^*}\frac{\partial Y_i}{\partial \theta}+w\frac{\partial Y_i}{\partial z^*}\right)-\rho D\left(\frac{\partial^2 Y_i}{\partial r^{*2}}+\frac{1}{r^*}\frac{\partial Y_i}{\partial r^*}+\frac{1}{r^{*2}}\frac{\partial^2 Y_i}{\partial \theta^2}+\frac{\partial^2 Y_i}{\partial z^2}\right)=-\alpha_i\Omega \tag{A.18}$$

Nondimensionalizing,

$$\frac{\partial \overline{Y}_i}{\partial \tau}+rF\frac{\partial \overline{Y}_i}{\partial r}+G\frac{\partial \overline{Y}_i}{\partial \theta}+H\frac{\partial \overline{Y}_i}{\partial z}-\frac{1}{Sc_i}\left(\frac{1}{r}\frac{\partial \overline{Y}_i}{\partial r}+\frac{\partial^2 \overline{Y}_i}{\partial r^2}+\frac{1}{r^2}\frac{\partial^2 \overline{Y}_i}{\partial \theta^2}+\frac{\partial^2 \overline{Y}_i}{\partial z^2}\right)=-\alpha_i\overline{\Omega} \tag{A.19}$$

Appendix B

Code Verification

B.0.1 Grid Refinement Study

This section summarizes the results from the grid refinement study conducted to verify that the centered difference derivative routines being used in this study are fourth order accurate. For this grid refinement study, derivatives of some common functions are computed using the fourth order scheme. An average error is then calculated over all grid points according to

$$L2 = \frac{1}{n} \sum \sqrt{(f_{exact} - f_{numerical})^2}, \tag{B.1}$$

where n is the total number of grid points. This error is computed for a grid with grid refinement ratio of 2 between successive grids. The order of grid convergence is then computed

$$E = L2 = Ch^p + Higher\ order\ terms, \tag{B.2}$$

where h is the grid spacing, C a constant and p is the order of convergence. If the higher order terms are neglected, and a logarithm is taken,

$$Ln(E) = Ln(C) + pLn(h), \tag{B.3}$$

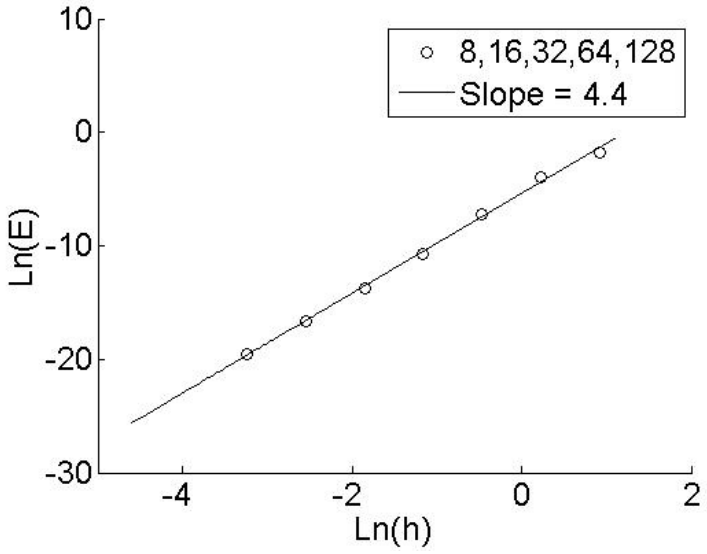


Figure B.1: Order of Grid Convergence

then the order of convergence is the slope of the plot of $Ln(E)$ versus $Ln(h)$. Figure B.1 is a plot of equation B.3 for several grids. The grids used for the study are listed in the legend. The solid line is a trend line derived using a least squares method. The slope of the trendline is also listed in the legend. As seen from the figure, the slope of the least squares fit curve is 4.4. Also, the average value of p obtained considering,

$$E_1 = Ch_1^p \tag{B.4}$$

$$E_2 = Ch_2^p \tag{B.5}$$

$$where, \qquad h_2 = h_1/2 \tag{B.6}$$

$$\rightarrow \frac{E_1}{E_2} = \left(\frac{h_1}{h_1/2}\right)^p = 2^p \tag{B.7}$$

$$\rightarrow p = \frac{\ln(E_1/E_2)}{\ln 2} \tag{B.8}$$

is $p = 4.52$. Thus, the derivative routines being used are fourth order accurate.

B.0.2 Benchmark Studies

In this section the solutions to some simple problems are summarized in order to validate the solver. In all these problems the initial and boundary conditions are stated and the steady state solutions are obtained using the solver developed for the current study.

Benchmark Problem 1

The first problem consists of a simple three-dimensional second order partial differential equation. This problem is used to verify that the basic functions of the solver are implemented properly.

$$T' = T_{x1x1} + T_{x2x2} + T_{x3x3} \tag{B.9}$$

This parabolic partial differential equation has the exact solution

$$T = 1 + e^{-3t}(sin(x1)sin(x2)sin(x3)). \tag{B.10}$$

At steady state the temperature converges to a value of 1.0 as the exponential term decays to 0. This partial differential equation is solved in the domain

$$0 < x1 < 2\pi \quad 0 < x2 < 2\pi \quad 0 < x3 < 2\pi.$$

and the initial conditions are calculated using equation B.10 with t=0,

$$T(0, x1, x2, x3) = 1 + sin(x1)sin(x2)sin(x3), \tag{B.11}$$

while the six boundaries are set to unity.

$$\begin{aligned} T(t, 0, x2, x3) &= 1.0 \quad T(t, 2\pi, x2, x3) = 1.0 \\ T(t, x1, 0, x3) &= 1.0 \quad T(t, x1, 2\pi, x3) = 1.0 \\ T(t, x1, x2, 0) &= 1.0 \quad T(t, x1, x2, 2\pi) = 1.0 \end{aligned} \tag{B.12}$$

Figures B.2 and B.3 show the relative percentage error for the solution for a 16 cubed and a 32 cubed grid. As seen from this figure, the relative error is very small at steady state. This indicates that the code is capable of solving the above three dimensional system with a relative error of 0.003% for a 32 cubed grid.

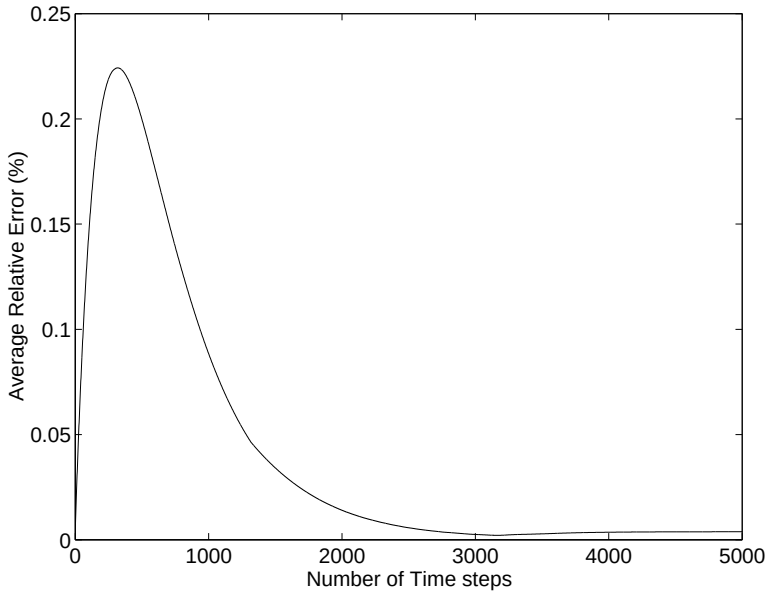


Figure B.2: Benchmark Problem 1, Grid=16 cubed

Benchmark Problem 2

This problem is considered to ensure that the boundary conditions are applied appropriately for a problem in cylindrical coordinates, where the reciprocal of r appears as a coefficient to one of the derivative terms. This, if not dealt with appropriately, causes a singularity at $r = 0$. Here, the variables in the runge kutta iteration are computed from $r = 0$ to $r = n - 1$ and the boundary condition is applied at these points. The problem is given by

$$T_t = \frac{1}{4}\left[\frac{1}{r}\frac{\partial T}{\partial r} + \frac{\partial^2 T}{\partial r^2}\right] + Q, \tag{B.13}$$

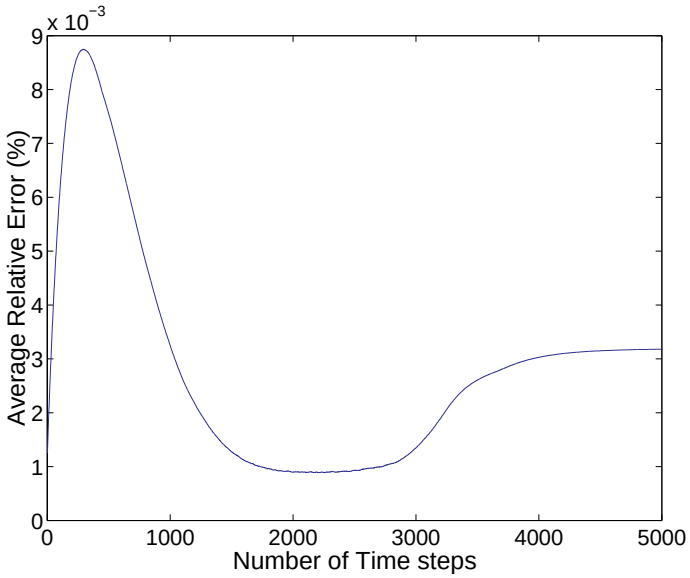


Figure B.3: Benchmark Problem 1, Grid = 32 cubed

where the source term is

$$Q = -0.0001r^2 e^{-0.01r^2} e^{-0.01t}. \tag{B.14}$$

The boundary conditions are

$$\frac{\partial T}{\partial r}(t, 0) = 0.0 \quad and \quad \frac{\partial T}{\partial r}(t, 40) = 0. \tag{B.15}$$

For this equation, the exact solution of T is

$$T = e^{-0.01r^2} e^{-0.01t}. \tag{B.16}$$

Figure B.4 shows the temperature profile after 5000 time steps from the numerical and exact solutions. Figure B.5 shows the absolute error at each r location, and as seen here the errors are very small. These plots show that the boundary conditions are applied appropriately and that the code is able to solve problems in cylindrical coordinates.

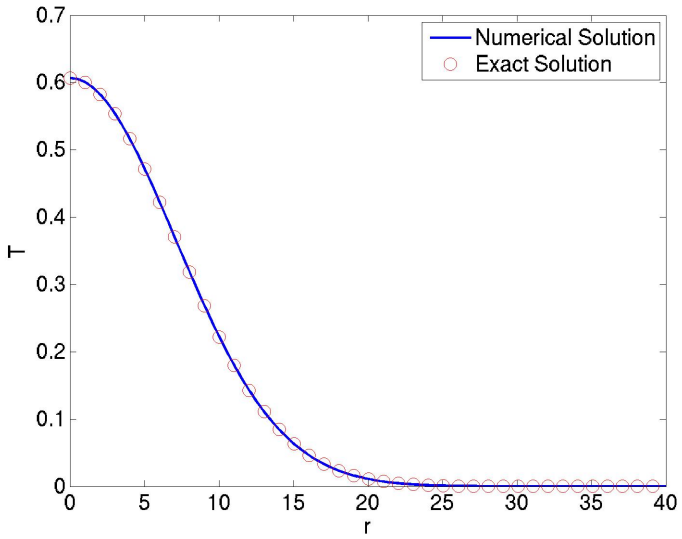


Figure B.4: Temperature Profile for Benchmark Problem 2

Benchmark Problem 3

The third problem involves solving simulataneously, partial differential equations for the species and the temperature. The governing equations are

$$T_t = \lambda_1(uT_{x1x1} + vT_{x2x2} + wT_{x3x3}) - \beta(T_{x1} + T_{x2} + T_{x3}) + Q_1, \tag{B.17}$$

$$Y_t = \lambda_2(Y_{x1x1} + Y_{x2x2} + Y_{x3x3}) - Q_1T + Q_2. \tag{B.18}$$

The boundary conditions are

$$T(t, 0, x2, x3) = 1.0 \quad T(t, 2\pi, x2, x3) = 1.0,$$

$$T(t, x1, 0, x3) = 1.0 \quad T(t, x1, 2\pi, x3) = 1.0,$$

$$T(t, x1, x2, 0) = 1.0 \quad T(t, x1, x2, 2\pi) = 1.0,$$

$$Y(t, 0, x2, x3) = 1.0 \quad Y(t, 2\pi, x2, x3) = 1.0,$$

$$Y(t, x1, 0, x3) = 1.0 \quad Y(t, x1, 2\pi, x3) = 1.0,$$

$$Y(t, x1, x2, 0) = 1.0 \quad Y(t, x1, x2, 2\pi) = 1.0,$$

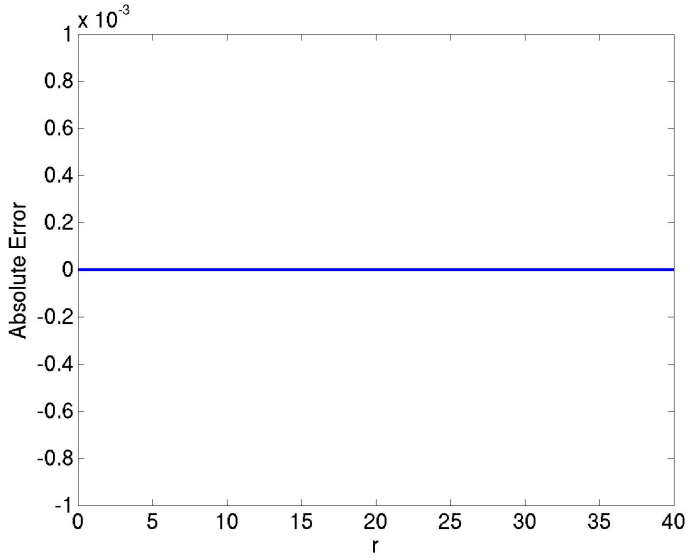


Figure B.5: Error for Benchmark Problem 2

and $u = v = w = 1.0$ for all grid points. If the exact solutions of T and Y are taken as

$$T = 1 + e^{-3t}(sin(x1)sin(x2)sin(x3)), \tag{B.19}$$

$$Y = 1 + e^{-3t}(sin(x1)sin(x2)sin(x3)), \tag{B.20}$$

and if $\lambda_1 = 1$ the value of Q_1 is

$$Q_1 = \beta(T_{x1} + T_{x2} + T_{x3}). \tag{B.21}$$

The value of Q_2 is

$$Q_2 = Q_1 T. \tag{B.22}$$

Figures B.6 and B.7 show the differences between the computed and analytical solutions, for the temperature and species respectively. These results show that after 2000 time steps the difference between the analytical results and the computed results is less that 0.01%.

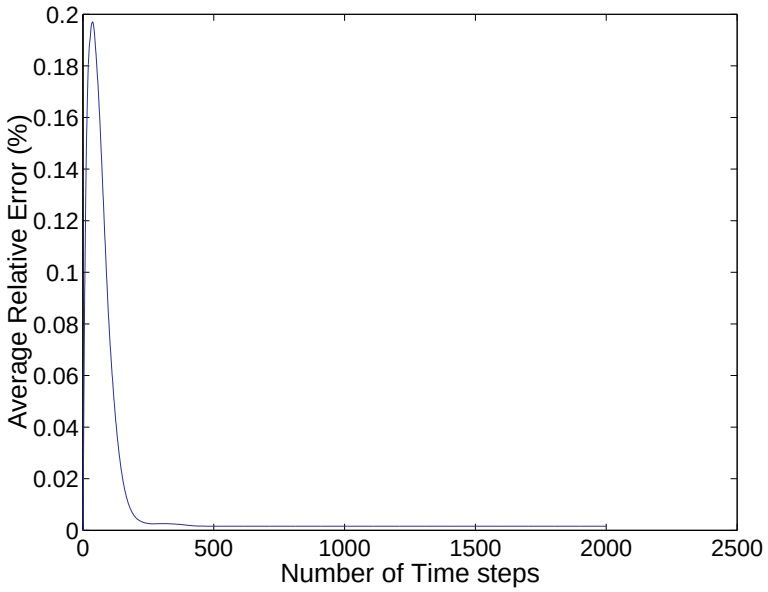


Figure B.6: Benchmark Problem 3, Error in the Temperature, Grid = 16 cubed

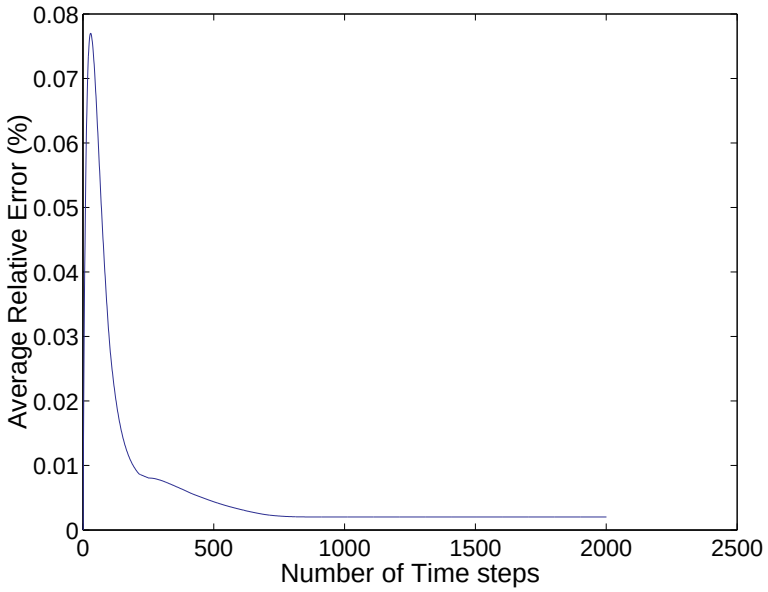


Figure B.7: Benchmark Problem 3, Error in the Species, Grid = 16 cubed

Appendix C

Normal Mode Analysis

The stability of the system is examined using a numerical normal mode analysis. The perturbed solutions are defined as

$$\begin{bmatrix} F(z) \\ G(z) \\ H(z) \\ T(r,\theta,z,t) \\ Y_f(r,\theta,z,t) \\ Y_o(r,\theta,z,t) \end{bmatrix} = \begin{bmatrix} F(z) \\ G(z) \\ H(z) \\ T_{ss}(z) \\ Y_{fss}(z) \\ Y_{oss}(z) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \hat{T}(z) \\ \hat{Y_f}(z) \\ \hat{Y_o}(z) \end{bmatrix} Ar^k e^{\omega\tau + ik\theta}, \tag{C.1}$$

where, $T_{ss}(z)$, $Y_{fss}(z)$, and $Y_{oss}(z)$ are the 1-dimensional steady state solutions, ω is the growth rate and k is the wavenumber. Detailed derivation of the separated form is included in the appendix. Upon substituting the above perturbed solutions into the field equations

and linearizing, equations for $\hat{T}(z)$, $\hat{Y}_f(z)$, and $\hat{Y}_o(z)$ are obtained

$$\omega\hat{T} + kF\hat{T} + ikG\hat{T} + H\hat{T}' - \hat{T}'' = \beta\hat{\Omega}, \tag{C.2}$$

$$\omega\hat{Y}_f + kF\hat{Y}_f + ikG\hat{Y}_f + H\hat{Y}_f' - \hat{Y}_f''/Le_f = \alpha_f\hat{\Omega}, \tag{C.3}$$

$$\omega\hat{Y}_o + kF\hat{Y}_o + ikG\hat{Y}_o + H\hat{Y}_o' - \hat{Y}_o''/Le_o = \alpha_o\hat{\Omega}, \tag{C.4}$$

where,

$$\hat{\Omega} = D\overline{D}e^{-Ze/T_{ss}}(Y_{oss}\hat{Y}_f + Y_{fss}\hat{Y}_o + Y_{fss}Y_{oss}Ze\hat{T}T_{ss}^{-2}). \tag{C.5}$$

The boundary conditions for the linearized system are

$$\hat{T}(0) = 0, \quad \hat{Y}_o'(0) = PeLe_o\hat{Y}_o(0), \quad \hat{Y}_f'(0) = PeLe_f\hat{Y}_f(0), \tag{C.6}$$

$$\hat{T}(\infty) = 0, \quad \hat{Y}_o(\infty) = 0, \quad and \quad \hat{Y}_f(\infty) = 0. \tag{C.7}$$

When discretized, equations C.2-C.7 define a matrix eigenvalue problem of the form

$$(\mathbf{A} + \omega\mathbf{B})\overrightarrow{X} = 0, \tag{C.8}$$

where $\mathbf{A}$ is

$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & - & 0 \\
0 & \frac{3}{2\Delta z}+PeLe_o & 0 & 0 & -\frac{4}{2\Delta z} & 0 & 0 & \frac{1}{2\Delta z} & 0 & - & 0 \\
0 & 0 & \frac{3}{2\Delta z}+PeLe_f & 0 & 0 & -\frac{4}{2\Delta z} & 0 & 0 & \frac{1}{2\Delta z} & - & 0 \\
\Pi 1 & 0 & 0 & \Gamma 1_1 & D\overline{D}e^{-ZeT_{ss1}^{-1}}Y_{f1} & D\overline{D}e^{-ZeT_{ss1}^{-1}}Y_{o1} & \overline{\Pi 1} & 0 & - & - & 0 \\
0 & \Pi 2 & 0 & \Lambda_1 & \Gamma 2_1 & D\overline{D}e^{-ZeT_{ss1}^{-1}}Y_{o1} & 0 & \overline{\Pi 2} & 0 & - & 0 \\
0 & 0 & \Pi 3 & \Lambda_1 & D\overline{D}e^{-ZeT_{ss1}^{-1}}Y_{f1} & \Gamma 3_1 & 0 & 0 & \overline{\Pi 3} & - & 0 \\
| & | & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & | & | & | \\
| & | & | & | & | & | & | & 1 & 0 & 0 \\
| & | & | & | & | & | & | & 0 & 1 & 0 \\
| & | & | & | & | & | & | 0 & 0 & 1
\end{bmatrix} \tag{C.9}$$

where,

$$\Pi 1 = H/2\Delta z + 1/\Delta z^2, \quad \overline{\Pi 1} = -H/2\Delta z + 1/\Delta z^2,$$

$$\Pi 2 = H/2\Delta z + 1/Le_f\Delta z^2, \quad \overline{\Pi 2} = -H/2\Delta z + 1/Le_f\Delta z^2,$$

$$\Pi 3 = H/2\Delta z + 1/Le_o\Delta z^2, \quad \overline{\Pi 3} = -H/2\Delta z + 1/Le_o\Delta z^2,$$

$$\Lambda_j = D\overline{D}Y_{ossj}Y_{fssj}ZeT_{ssj}^{-2},$$

$$\Gamma 1_j = -kF_{ssj} - ikG_{ssj} - 2\Delta z^{-2} + \beta Y_{fssj}Y_{ossj}D\overline{D}e^{-ZeT_{ssj}^{-1}}ZeT_{ssj}^{-2},$$

$$\Gamma 2_j = -kF_{ssj} - ikG_{ssj} - 2\Delta z^{-2} + \alpha_f Y_{ossj}D\overline{D}e^{-ZeT_{ssj}^{-1}}, \tag{C.10}$$

$$\Gamma 3_j = -kF_{ssj} - ikG_{ssj} - 2\Delta z^{-2} + \alpha_o Y_{fssj}D\overline{D}e^{-ZeT_{ssj}^{-1}}.$$

B is given by,

$$B(i,i) = 1, \quad for \quad i = 4, \quad to \quad i = 3N - 3,$$

$$B(i,i) = 0, \quad for \quad i = 1, \quad 3N - 2, \quad 3N - 1, \quad 3N,$$

$$B(i,j) = 0, \quad for \quad i, \neq j$$

and

$$\overrightarrow{X}^T = \left[\begin{array}{cccccccccccccccc} T_0 & Y_{o0} & Y_{f0} & T_1 & Y_{o1} & Y_{f1} & T_2 & Y_{o2} & Y_{f2} & - & - & - & - & T_N & Y_{oN} & Y_{fN} \end{array} \right]. \tag{C.11}$$

Here, N, is the total number of grid pts used in the discretization. Figure C.1 shows the progression of the largest eigenvalue from stable (< 0) to unstable (> 0) values, for wavenumbers of 1 to 7, $N = 50$. As seen from this figure, at high Damköhler numbers the largest eigenvalue for all the modes asymptotes to a negative value. As the Damköhler number decreases the eigenvalue transitions from this stable branch to an unstable one. The Damköhler number at which this transition occurs decreases as the wavenumber increases. Table C shows the Damköhler numbers for which the system becomes unstable alongside the Damköhler numbers that are observed in the three-dimensional simulations for a 64^3 grid. The extinction Damköhler number for this problem is $D_E = 1.46$

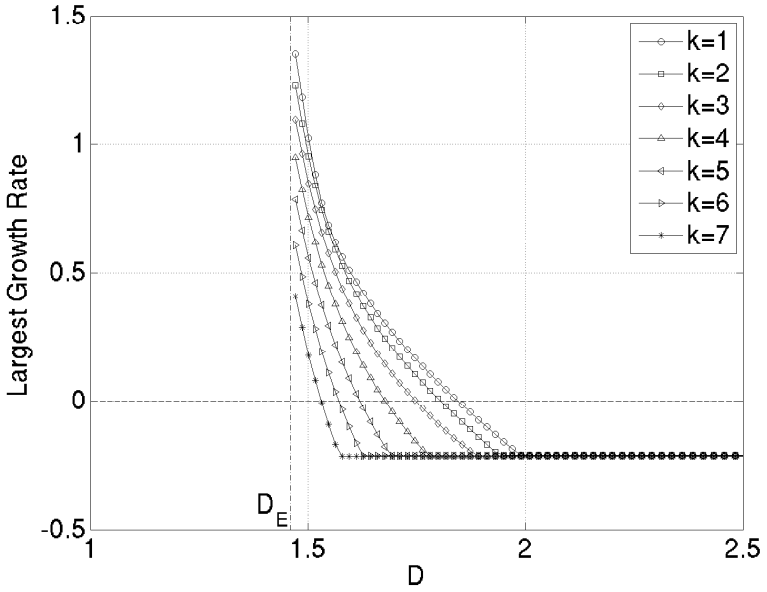


Figure C.1: The first eigenvalue as a function of D for several k's, $N = 50$

k	D($N = 50$)	D($N = 100$)	D(3-D, 64^3 Grid)
1	1.845	1.82	1.90
2	1.806	1.79	1.85
3	1.748	1.74	-
4	1.678	1.67	-
5	1.620	1.61	-
6	1.572	1.56	-
7	1.531	1.49	-

Table C.1: Transition Damköhler Numbers from the eigenvalue problem and from 3-D simulations for $k = 1$ to $k = 7$

Appendix D

Eigenvalue Problem: Separation of Variables

The non-dimensional system is given by,

$$F = \frac{u}{b_0 r}, \quad G = \frac{v}{b_0 r}, \quad H = \frac{w}{\sqrt{b_0 \nu}}, \tag{D.1}$$

where F, G, H are functions of z. The resulting non-dimensional equations are

$$H' = -2F, \tag{D.2}$$

$$F'' = F^2 + F'H - G^2, \tag{D.3}$$

$$G'' = 2FG + HG', \tag{D.4}$$

$$P' = 2HF - 2F'. \tag{D.5}$$

$$F(0) = 0, \quad G(0) = G_0 = 1, \quad H(0) = H_0, \quad P(0) = 0,$$

$$F(\infty) = 0, \quad G(\infty) = 0. \tag{D.6}$$

$$L[T, Y_i] - M[Pr^{-1}T, Pr^{-1}Le_i Y_i] = \Omega[\beta, -\alpha_i], \tag{D.7}$$

where

$$\mathbf{L} = \frac{\partial}{\partial t} + rF\frac{\partial}{\partial r} + G\frac{\partial}{\partial \theta} + H\frac{\partial}{\partial z},$$
$$\mathbf{M} = \frac{1}{r}\frac{\partial}{\partial r} + \frac{\partial^2}{\partial r^2} + \frac{1}{r^2}\frac{\partial}{\partial \theta^2} + \frac{\partial^2}{\partial z^2}.$$

The boundary conditions are,

$$z = 0: \quad T_0 = T_s, \quad \frac{\partial Y_f}{\partial z} = H_0 Pr Le_f (Y_f - 1), \quad \frac{\partial Y_o}{\partial z} = H_0 Pr Le_o Y_o,$$

$$z = \infty: \quad T_\infty = 1, \quad Y_f = 0, \quad Y_o = \phi^{-1},$$

$$r = 0: \quad \frac{\partial T}{\partial r} = 0, \quad \frac{\partial Y_i}{\partial r} = 0, \quad r = \infty: \quad \frac{\partial T}{\partial r} = 0, \quad \frac{\partial Y_i}{\partial r} = 0,$$

$$T|_{\theta=0} = T|_{\theta=2\pi}, \quad Y_i|_{\theta=0} = Y_i|_{\theta=2\pi}, \quad \frac{\partial T}{\partial \theta}|_{\theta=0} = \frac{\partial T}{\partial \theta}|_{\theta=2\pi}, \quad \frac{\partial Y_i}{\partial \theta}|_{\theta=0} = \frac{\partial Y_i}{\partial \theta}|_{\theta=2\pi}.$$

The perturbed solutions are defined as

$$\begin{bmatrix} F(z) \\ G(z) \\ H(z) \\ T(r,\theta,z,t) \\ Y_f(r,\theta,z,t) \\ Y_o(r,\theta,z,t) \end{bmatrix} = \begin{bmatrix} F(z) \\ G(z) \\ H(z) \\ T_{ss}(z) \\ Y_{fss}(z) \\ Y_{oss}(z) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \tilde{T}(r,\theta,z,t) \\ \tilde{Y}_f(r,\theta,z,t) \\ \tilde{Y}_o(r,\theta,z,t) \end{bmatrix}, \tag{D.9}$$

where, $T_{ss}(z)$, $Y_{fss}(z)$, and $Y_{oss}(z)$ are the 1-dimensional steady state solutions. From these equations it is evident that t and θ are readily separable if the perturbations are taken to be of the form

$$\begin{bmatrix} \tilde{T}(r,\theta,z,t) \\ \tilde{Y}_f(r,\theta,z,t) \\ \tilde{Y}_o(r,\theta,z,t) \end{bmatrix} = \begin{bmatrix} \check{T}(r,z) \\ \check{Y}_f(r,z) \\ \check{Y}_o(r,z) \end{bmatrix} e^{\omega t + ik\theta}. \tag{D.10}$$

The resulting equations are

$$\omega\check{T} + rF\check{T}_r + ikG\check{T} + H\check{T}_z = \left(\check{T}_r/r + \check{T}_{rr} - k^2\check{T}/r^2 + \check{T}_{zz}\right) + \beta\check{\Omega}, \tag{D.11}$$

$$\omega\check{Y}_f + rF\check{Y}_f r + ikG\check{Y}_f + H\check{Y}_f z = 1/Le_f\left(\check{Y}_f r/r + \check{Y}_{frr} - k^2\check{Y}_f/r^2 + \check{Y}_{fzz}\right) - \alpha_f\check{\Omega}, \tag{D.12}$$

$$\omega\check{Y}_o + rF\check{Y}_o r + ikG\check{Y}_o + H\check{Y}_o z = 1/Le_o\left(\check{Y}_o r/r + \check{Y}_{orr} - k^2\check{Y}_o/r^2 + \check{Y}_{ozz}\right) - \alpha_o\check{\Omega}, \tag{D.13}$$

where, $\check{\Omega} = D\overline{D}e^{-Ze/T_{ss}}\left(Y_{oss}\check{Y}_f + Y_{fss}\check{Y}_o + Y_{fss}Y_{oss}Ze\check{T}/T_{ss}^2\right)$. Now, taking the perturbations as

$$\begin{bmatrix} \check{T}(r,z) \\ \check{Y}_f(r,z) \\ \check{Y}_o(r,z) \end{bmatrix} = \begin{bmatrix} R(r)\hat{T}(z) \\ R(r)\hat{Y}_f(z) \\ R(r)\hat{Y}_o(z) \end{bmatrix}, \tag{D.14}$$

the equation for $\check{T}$ becomes

$$\omega + ikG + H\hat{T}'/\hat{T} - \hat{T}''/\hat{T} + \beta\hat{\Omega} = R''/R + R'/(rR) - rFR'/R - k^2/r^2. \tag{D.15}$$

In order for the above equation to be separable, rR'/R has to be a constant because $F = F(z)$.Thus,

$$rR'/R = const \tag{D.16}$$

$$\rightarrow R = Ar^{\alpha}. \tag{D.17}$$

Thus,

$$\omega + ikG + H\hat{T}'/\hat{T} - \hat{T}''/\hat{T} + \alpha F + \beta\hat{\Omega} = \alpha(\alpha-1)r^{\alpha-2}/r^{\alpha} + \alpha r^{\alpha-1}/(rr^{\alpha}) - k^2/r^2. \tag{D.18}$$

The right hand side of the above equation must equal a constant. Thus,

$$\alpha^2/r^2 - k^2/r^2 = const. \tag{D.19}$$

Since α and k are constants, for equation D.19 to be true for all r, the constant must be zero, and so,

$$\alpha = \pm k \tag{D.20}$$

and the equations for $\hat{T}$, $\hat{Y}_f$ and $\hat{Y}_o$ are

$$\omega\hat{T} + kF\hat{T} + ikG\hat{T} + H\hat{T}_z + \hat{T}_{zz} - \beta\hat{\Omega} = 0, \tag{D.21}$$

$$\omega\hat{Y}_f + kF\hat{Y}_f + ikG\hat{Y}_f + H\hat{Y}_{fz} + 1/Le_f\hat{Y}_{fzz} + \alpha_f\hat{\Omega} = 0, \tag{D.22}$$

$$\omega\hat{Y}_o + kF\hat{Y}_o + ikG\hat{Y}_o + H\hat{Y}_{oz} + 1/Le_o\hat{Y}_{ozz} + \alpha_o\hat{\Omega} = 0, \tag{D.23}$$

where, $\hat{\Omega} = D\overline{D}e^{-Ze/T_{ss}}\left(Y_{oss}\hat{Y}_f + Y_{fss}\hat{Y}_o + Y_{fss}Y_{oss}Ze\hat{T}/T_{ss}^2\right)$. Thus,

$$\check{T}(r,\theta,z,t) = \sum_{k=0}^{\infty} A_k r^{+k}\hat{T}(z)e^{\omega t+ik\theta}, \tag{D.24}$$

$$\check{Y}_f(r,\theta,z,t) = \sum_{k=0}^{\infty} A_k r^{+k}\hat{Y}_f(z)e^{\omega t+ik\theta}, \tag{D.25}$$

$$\check{Y}_o(r,\theta,z,t) = \sum_{k=0}^{\infty} A_k r^{+k}\hat{Y}_o(z)e^{\omega t+ik\theta}. \tag{D.26}$$

To satisfy the boundary condition, $\partial[\check{T},\check{Y}_i]/\partial r = 0$ as $r \to \infty$,

$$\sum_{k=0}^{\infty} kA_k r^{k-1}[\hat{T}(z),\hat{Y}_i(z)]e^{ik\theta} = 0, \tag{D.27}$$

and so

$$\sum_{k=0}^{\infty} kA_k r^k e^{ik\theta} = 0. \tag{D.28}$$

This condition dictates that the coefficients A_k be such that $r \to \infty$ be included in the region of convergence for $\sum_{k=0}^{\infty} kA_k r^k e^{ik\theta}$. Now, since $k \geq 0$, it follows that $|A_k r^k e^{ik\theta}| \leq |kA_k r^k e^{ik\theta}|$. Thus, from the theorems that if $\sum|u_n|$ converges and $|v_n| \leq |u_n|$, then $\sum|v_n|$ converges, and if $\sum|u_n|$ converges, then $\sum u_n$ converges, (where $\sum u_n = \sum_{n=1}^{\infty} u_n = u_1 + u_2.....$),

$\sum_{k=0}^{\infty} A_k r^k e^{ik\theta}$ should converge as $r \to \infty$.

Upon discretization using following expressions for the first and second derivatives at the interior and boundary points

$$\frac{dX_j}{dz} = (X_{j+1} - X_{j-1})/(2\Delta z), \tag{D.29}$$

$$\frac{d^2X_j}{dz^2} = (X_{j+1} - 2X_i + X_{j-1})/(\Delta z)^2, \tag{D.30}$$

$$\frac{dX_0}{dz} = (-3X_0 + 4X_1 - X_2)/(2\Delta z), \tag{D.31}$$

$$\frac{dX_N}{dz} = (-3X_N + 4X_{N-1} - X_{N-2})/(2\Delta z), \tag{D.32}$$

the temperature eqation becomes,

$$\begin{aligned} \omega\hat{T}_j = \quad & \hat{T}_j(-kF - ikG + D\overline{D}e^{-Ze/T_s s}Y_{fss}Y_{oss} - 2/(\Delta z)^2) \\ & +\hat{T}_{j-1}\{H/(2\Delta z) + 1/(\Delta z)^2\} \\ & +\hat{T}_{j+1}\{-H/(2\Delta z) + 1/(\Delta z)^2\} \\ & +D\overline{D}e^{-Ze/T_s s}(Y_{oss}\hat{Y}_f + Y_{fss}\hat{Y}_o). \end{aligned} \tag{D.33}$$

Similarly, discretized equations for the species and boundary conditions are found and the resulting system is given by equation C.8.

Appendix E

Source Code

Makefile

```
SHELL       = /bin/sh
.SUFFIXES:  .o .f90 .f

OSTYPE      = $(shell uname)

EXEC        = vkcf

OBJS        = vk_colsys.o dcolsys.o vkvar.o vkpar.o vk_in.o\
              gen_grid.o der_ord_4_2.o grid_map.o func_def.o\
              vk_cf.o vkcf_driver.o\

MX          = mod
CD          = \cd
LN          = ln -sf
RM          = rm -f
ECHO        = echo

#------------------ options for Blue Horizon (San Diego)
CXX         = g++
F90C        = mpif90
F77C        = mpif77
F90FLAGS    = -qsuffix=f=f90 -qinitauto -qthreaded -qtune=auto
F77FLAGS    = -O3 -Q -qinitauto -qthreaded -qstrict -qtune=auto -qzerosize
LD          = $(F90C)
LDFLAGS = -bind_at_load
LDLIBS = -lpmpich -lmpich -lmpe

all :    note  run
```

```
note:
	$(ECHO) "Building $(EXEC) on a $(OSTYPE) system. $(F90C)"
	-$(RM) $(EXEC)

$(EXEC):	$(OBJS)
	$(LD) $(LDFLAGS) $(F90FLAGS) -o $(EXEC) $(OBJS) $(LDLIBS)

run :		$(EXEC)

seq :
	-$(CD) orig_code;$(MAKE)

#-------------------------------- dependencies
vk_colsys.o: vk_colsys.f dcolsys.o
dcolsys.o: dcolsys.f
vkvar.o: vkvar.f90
vk_in.o: vk_in.f90 vkvar.o
vkpar.o: vkpar.f90 vkvar.o vk_in.o
gen_grid.o: gen_grid.f90 vkvar.o
der_ord_4_2.o: der_ord_4_2.f90 vkvar.o
grid_map.o: grid_map.f90 der_ord_4_2.o
func_def.o: func_def.f90 vkvar.o
vk_cf.o: vk_cf.f90 vkvar.o vkpar.o der_ord_4_2.o
vkcf_driver.o: vkcf_driver.f90 vkvar.o vk_in.o gen_grid.o der_ord_4_2.o grid_map.o func_def.o vk_cf.o vkpar.o

#-------------------------------- implicit rules
%.o: %.f90
	$(F90C) -c $(F90FLAGS) $<
%.o: %.f
	$(F77C) -c $(F77FLAGS) $<
.C.o :
	$(CXX) -c $(OFLAGS) $<
###%.o: %.f90
###	$(F90C) -c $(F90FLAGS) $< -o $@
###%.o: %.f
###	$(F77) -c $(FFLAGS) $< -o $@
#-------------------------------- clean

new :	clean note run



clean:	runclean
	-$(RM) -f *~ *.o *.mod *.s $(EXEC)
runclean:
	-$(RM) -f core fort.* $(EXEC)
#--------------------------------
```

Driver

```
PROGRAM vk
!
```

```
!=============================================================
! This is the top level driver for the parallel version
! of the three-dimensional fourth order Runge Kutta
! code used for solving the Spinning Disk Flame Sheet
! Problem
!
! Date : April 2005
!=============================================================
!
  USE vkvar
  USE vk_in
  USE gen_grid
  USE der_ord_4
  USE grid_map
  USE func_def
  USE vk_cf
  USE vkpar
!
!
  IMPLICIT NONE
!
  INTEGER :: var,nn,npts,k,j,i
  REAL*8 :: tt1, tt2, mod1,tout,time1,time2,total_time
!
! Initiate MPI
!==============
!
  call MPI_INIT(ierr)
  call MPI_COMM_RANK(MPI_COMM_WORLD, id, ierr)
  call MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
!
! Read Inputs
!=============
!
  CALL input
  CALL grid_assign
  CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
! Allocate arrays, set up grid and initial conditions and
! communicate with adjacent processors
!=======================================
!
  CALL allocate
  CALL grid
  if (irestart==0) CALL initial_conditions(nvar)
  if (irestart==1.or.irestart==2) CALL restart
  if (irestart==3) CALL initial_colsys
!
  CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
  CALL vk_buffer_comm(1,fsize)
  CALL init_runge
```

```
  CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
  if (id==0) then
     print 1, 'D=',Damk, 'Le_F=', Le1, 'Pr=', Pr,'Pe=',Pe
  end if
1 format(a6,f24.4,a6,f4.2,a6,f4.2,a6,f4.2)
! If there is a stretch map the derivaties
!============================
!
  CALL der_map
!
! Print pertinent parameters to screen
!============================
!
!
! Integrate until tstop is reached
!============================
!
  if (irestart==0) then
     if (id==0) open(UNIT=222,file='vkout.dat',status='replace')
     t = 0.0
  end if
!
  if (irestart==3) then
     if (id==0) open(UNIT=222,file='vkout.dat',status='replace')
     t = 0.0
  end if
!
  if (irestart==1) then
     if (id==0) open(UNIT=222,file='vkout.dat',status='old',position='append')
     t = tlast
  end if
!
  if (irestart==2) then
     if (id==0) open(UNIT=222,file='vkout.dat',status='replace')
     t = 0.0
  end if
!
  time1 = MPI_WTIME()
  DO nn=1,n_tstep
!
     tout = t + dt
    CALL rk54(t,tout)
    CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
    if (mod(nn,err_out_freq)==0) CALL track_error(t)
    if (nn==51000.and.pert.ne.0.0) CALL perturb
    if (mod(nn,cont_out_freq)==0.and.nn.ne.n_tstep) then
       CALL vk_communication(1,fsize)
       CALL output(t,nn)
    end if
!
```

```
  END DO
!                                                                              110
  time2 = MPI_WTIME()
  nn = nn-1
  CALL vk_communication(1,fsize)
  CALL output(t,nn)
!
  CLOSE(222)
!
!
! Gather the temperature and species values from all the processors
! and write them to the output file                                            120
!==============================================
!
  if (id ==0) print *, 'Wrote output'
!
  total_time = time2-time1
!
  if (id == 0) print 22, 'Time taken: ', total_time
!
22 format(2x,a12,1f36.20)
!                                                                              130
  print *, id, 'RUN COMPLETED'
!
! Deallocate all arrays and call for a stop
!============================
!
  deallocate(f,ferr,cc,cc1,Q1,fprime,VEL)
!
  call MPI_FINALIZE(ierror)
!
  STOP                                                                         140
!
END PROGRAM vk
```

Global Variables

```
MODULE vkvar
      IMPLICIT NONE
!
      SAVE
!
      INCLUDE "mpif.h"
!
      INTEGER, PARAMETER :: nx1max=100
      INTEGER, PARAMETER :: nx2max=100
      INTEGER, PARAMETER :: nx3max=100                                         10
      INTEGER, PARAMETER :: nmax=nx1max
      INTEGER, PARAMETER :: neqmax=6
      INTEGER :: n_tstep, nvar, n1, n2, n3,irestart,twrite
      REAL*8 :: t,tlast,pert,wnum,RHO, CP, U, V, W, LAMBDA, Q, OMEGA, STRETCH1, STRETCH2, STRETC
      REAL*8 :: T0, TINF,tprev,H0,b0
      REAL*8 :: dx1,dx2,dx3,rmt,rm,kny
```

```
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: ibnd,jbnd,kbnd
      INTEGER :: iord,err_out_freq,cont_out_freq
      INTEGER :: neqn,nx1,nx2,nx3
      REAL*8 :: x1min,x2min,x3min,x1max,x2max,x3max,dt,dt1                    20
      REAL*8 :: tend,cfl,tmax
      REAL*8 :: dx1min,dx2min,dx3min,x1lrxn,x2lrxn,x3lrxn,x1knx1,x2knx2
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: ack,bck,cck
      INTEGER :: ikutta,nstage,ikk,n2x,noutput2,noutput1,grd1,grd2,grd3
      REAL*8, DIMENSION(:), POINTER :: wf
      REAL*8, DIMENSION(:,:,:,:,:), TARGET, ALLOCATABLE :: cc1
      REAL*8, DIMENSION(:,:,:,:,:), TARGET, ALLOCATABLE :: cc
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: rkj
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: fprime
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE:: dfdx1,dfdx2,dfdx3    30
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: df2dx1,df2dx2,df2dx3
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: df2dx1dx2,df2dx1dx3,df2dx2dx3
      REAL*8, DIMENSION(:), POINTER :: b,d,rl,u1,a
      REAL*8 :: alp1st,a1st,b1st,alp2st,a2st,b2st
!
      REAL*8, DIMENSION(:,:,:), TARGET, ALLOCATABLE :: Q1
      REAL*8, DIMENSION(:,:,:), TARGET, ALLOCATABLE :: Q2
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: f
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: ferr
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: fx1                 40
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: ffx1
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: fx2
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: ffx2
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: fx3
      REAL*8, DIMENSION(:,:,:,:), TARGET, ALLOCATABLE :: ffx3
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: x1
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: x2
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: x3
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: xx1
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: xx2                       50
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: xx3
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: dxidx
      REAL*8, DIMENSION(:), TARGET, ALLOCATABLE :: dxi2dx2
      REAL*8, dimension(:,:,:,:,:), target, allocatable :: f_prime
      REAL*8, dimension(:,:,:,:,:), target, allocatable :: f_prime_periodic
      REAL*8, dimension(:,:,:,:,:), target, allocatable :: f_pprime_per
      REAL*8, dimension(:,:,:,:), target, allocatable :: f_periodic
      REAL*8, dimension(:,:,:,:,:), target, allocatable :: f_pprime
      REAL*8, dimension(:,:,:,:,:,:), target, allocatable :: f_cpprime
      REAL*8, dimension(:,:), target, allocatable :: VEL                     60
      INTEGER :: fsize
      REAL*8 :: Pr, Damk, beta, eq1, eps, Le1, Le2, alpha1, alpha2, bcval, Pe
      REAL*8 :: tstar, zstar
!
!     MPI related variables
      INTEGER :: nprocs, id, cart_comm, ierr,ierror, ghostlayer, mpilayer
      INTEGER :: x1size, x2size, x3size, totsize, velsize
      INTEGER :: sbx1_size,sbx2_size,sbx3_size
      INTEGER, dimension(:), TARGET, ALLOCATABLE :: allsize, disp, allrange
```

```
      INTEGER, dimension(2) :: loci, locj, lock                                    70
      INTEGER, dimension(3) :: comm, rid, rsize, coords, free, dims,periodic
      INTEGER, dimension(6) :: idnbr
      INTEGER :: err_chk
      INTEGER, dimension(3,2) :: dom_bound,sndim
      REAL*8,dimension(3) :: sndx
      REAL*8, dimension(:,:,:,:), target, allocatable :: out_f
      REAL*8, dimension(:), target, allocatable :: out_x1
      REAL*8, dimension(:), target, allocatable :: out_x2
      REAL*8, dimension(:), target, allocatable :: out_x3
      REAL*8, dimension(:), target, allocatable :: myres                           80
      REAL*8, dimension(:), target, allocatable :: allres
      REAL*8, dimension(:), target, allocatable :: send_buffx1_r
      REAL*8, dimension(:), target, allocatable :: send_buffx1_l
      REAL*8, dimension(:), target, allocatable :: send_buffx2_r
      REAL*8, dimension(:), target, allocatable :: send_buffx2_l
      REAL*8, dimension(:), target, allocatable :: send_buffx3_r
      REAL*8, dimension(:), target, allocatable :: send_buffx3_l
      REAL*8, dimension(:), target, allocatable :: recv_buffx1_l
      REAL*8, dimension(:), target, allocatable :: recv_buffx1_r
      REAL*8, dimension(:), target, allocatable :: recv_buffx2_l                   90
      REAL*8, dimension(:), target, allocatable :: recv_buffx2_r
      REAL*8, dimension(:), target, allocatable :: recv_buffx3_l
      REAL*8, dimension(:), target, allocatable :: recv_buffx3_r
!
    END MODULE vkvar
```

IO and Perturbations

```
MODULE vk_in
!
  CONTAINS
    SUBROUTINE input
!
!   This SUBROUTINE reads input parameters from file vk.dat
!============================================================
!
!                                                                              10
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: d1
      REAL*8 :: dummy
!
      open(Unit=14,file="vkcf.dat")
!
      read(14,'(a40)')                                                         20
      read(14,*) nvar,n_tstep,dt
      read(14,'(a40)')
      read(14,*) ikutta,iord
      read(14,'(a40)')
```

```
      read(14,*) H0,b0, eq1
      read(14,'(a40)')
      read(14,*) x1min,x2min,x3min
      read(14,'(a10)')
      read(14,*) x1max,x2max,x3max
      read(14,'(a40)')
      read(14,*) nx1,nx2,nx3
      read(14,'(a60)')
      read(14,*) STRETCH1, STRETCH2, STRETCH3
      read(14,'(a60)')
      read(14,*) Pr, Le1, Le2, Damk, alpha1, alpha2, beta, eps, bcval
      read(14,'(a60)')
      read(14,*) T0, TINF
      read(14,'(a80)')
      read(14,*) irestart, pert, wnum, err_out_freq, cont_out_freq
      read(14,'(a80)')
      read(14,*) grd1,grd2,grd3
      read(14,'(a80)')
      read(14,*) tstar, zstar
      close(14)
!
! Check for input errors
!==================
!
      err_chk = 0
      if (nx1.le.4.or.nx2.le.4.or.nx3.le.4) err_chk = 1
      if (n_tstep.le.0) err_chk = 2
      if (ikutta.le.0) err_chk = 3
      if (Pr.lt.0.or.Le1.lt.0.or.Le2.lt.0.or.Damk.lt.0) err_chk = 4
      if (alpha1.lt.0.or.alpha2.lt.0.or.eps.lt.0.or.bcval.lt.0) err_chk = 4
      if (irestart.lt.0.or.irestart.gt.3) err_chk = 5
!
      CALL ERROR
!
! Run vkcolsys in order to calculate the flowfield
! The velocities are stored in 'vel.dat'
!================================
!
      if (id ==0) call vkcolsys(nx3,STRETCH3,x3min,x3max,H0,b0)
      CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
      open(Unit=15,file="vel.dat")
!
!
!     Correct some variables
!============================
!
      fsize = nvar+3
!
    END SUBROUTINE input
!
!=====================================================
!=====================================================
```

```
!
    Subroutine restart
!                                                                              80
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: i,j,k,proc
      INTEGER,dimension(2) :: indi,indj,indk
      REAL*8, dimension(:,:,:,:),allocatable :: fres
!
      allocate(fres(3,0:sndim(3,1)-1,0:sndim(2,1)-1,0:sndim(1,1)-1))
      do i = 1,2                                                               90
            indi(i) = loci(i)-dom_bound(1,i)
            indj(i) = locj(i)-dom_bound(2,i)
            indk(i) = lock(i)-dom_bound(3,i)
         end do
!
!         if (id==0) then
            open(UNIT=301,file='res.dmp',status='old')
            do k = 0,sndim(3,1)-1
               do j = 0,sndim(2,1)-1
                  do i = 0,sndim(1,1)-1                                        100
!
                     read(301,*) fres(1,k,j,i), fres(2,k,j,i), fres(3,k,j,i)
!
                  end do
               end do
            end do
            read(301,*) tlast
            close(301)
!         end if
         call MPI_BARRIER(MPI_COMM_WORLD,ierr)                               110
!
         do k = indk(1),indk(2)
            do j = indj(1),indj(2)
               do i = indi(1),indi(2)
                  f(1,k,j,i) = fres(1,k,j,i)
                  f(2,k,j,i) = fres(2,k,j,i)
                  f(3,k,j,i) = fres(3,k,j,i)
                  ferr(1,k,j,i) = f(1,k,j,i)
               end do
            end do                                                             120
         end do
!
         deallocate(fres)
!
       end Subroutine restart
!
!=====================================================

!=====================================================
!                                                                              130
```

```
    Subroutine allocate
!
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: k
      REAL*8 :: aplace_holder
!
      ALLOCATE(f(1:fsize,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      ALLOCATE(ferr(1:1,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      ALLOCATE(fprime(1:fsize,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      ALLOCATE(rkj(1:nvar,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      ALLOCATE(cc(1:2*nvar,1:3,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      ALLOCATE(cc1(1:2,1:3,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      ALLOCATE(Q1(lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
!
        !      Obtain velocity information
!================================
!
      allocate(VEL(1:3,0:nx3-1))
      ALLOCATE(x3(0:nx3-1))
!
      do k = 0,nx3-1
!
         read(15,*) x3(k),&
              &VEL(3,k),VEL(1,k),VEL(2,k)
!
!
     end do
      close(15)
!
!     Assign value to Peclet Number
!=========================
      Pe = VEL(3,0)*Pr
!
    End Subroutine allocate
!
!=======================================================
!=======================================================
!
    Subroutine output(tt,step)
!
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER, target :: i,j,k
      INTEGER :: ind,proc,iout,tinteger
      INTEGER, pointer :: ii,jj,kk
      INTEGER, dimension(2) :: indi,indj,indk
      INTEGER, INTENT(IN) :: step
      REAL*8, INTENT(IN) :: tt
```

```
      character*20 :: oname,iname
!
      ind = 1
      if (id == 0) then
         oname = 'vkout_cont'
         iout = 122
         write(iname,23)trim(oname),tt/100                                    190
         print *, iname
23       format(a,f6.4,'.dat')
!
         open(UNIT=100,file=trim(iname),status='replace')
!
         do i = 0,nx1-1
            do j = 0,nx2-1
               do k = 0,nx3-1
!
                  if (sndim(1,2) == 1) ii => i                              200
                  if (sndim(2,2) == 1) jj => i
                  if (sndim(3,2) == 1) kk => i
                  if (sndim(1,2) == 2) ii => j
                  if (sndim(2,2) == 2) jj => j
                  if (sndim(3,2) == 2) kk => j
                  if (sndim(1,2) == 3) ii => k
                  if (sndim(2,2) == 3) jj => k
                  if (sndim(3,2) == 3) kk => k
!
                  write(100,200) out_f(nvar+1,kk,jj,ii) ,out_f(nvar+2,kk,jj,ii), out_f(nvar+3,kk,jj,ii) ,&   210
                       & out_f(1,kk,jj,ii), out_f(2,kk,jj,ii), out_f(3,kk,jj,ii)
!
                  ind = ind+1
!
               end do
            end do
         end do
!
         close(100)
      end if                                                                220
!
!
      if (id==0) then
         if (mod(step,50000)==0.or.step==n_tstep) then
            oname = 'res'
            write(iname,24)trim(oname),tt/100
            print *, iname
24          format(a,f6.4,'.dmp')
            open(UNIT=101,file=trim(iname),status='replace')
            open(UNIT=102,file='res.dmp',status='replace')                  230
!
            do k = 0,sndim(3,1)-1
               do j = 0,sndim(2,1)-1
                  do i = 0,sndim(1,1)-1
!
                     write(101,201) out_f(1,k,j,i), out_f(2,k,j,i), out_f(3,k,j,i)
```

```
                    write(102,201) out_f(1,k,j,i), out_f(2,k,j,i), out_f(3,k,j,i)
!
                  end do
                end do                                                    240
              end do
              write(101,*) t
              write(102,*) t
              close(101)
              close(102)
            end if
        end if
        deallocate(out_f)
!
        call MPI_BARRIER(MPI_COMM_WORLD,ierr)                             250
!
200   format(2x,3f10.6,3f12.8)
201   format(2x,3f22.18)
!
    End Subroutine output
!
!=================================================================
!=================================================================
!
    Subroutine track_error(curr_time)                                     260
!
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: i,j,k
      INTEGER,dimension(nvar) :: indz
      REAL*8, INTENT(IN) :: curr_time
      REAL*8,dimension(2) :: max_temp,max_temp_all,mtempavg,mtempavg_all
      REAL*8 :: err,err_all,qmax,qmaxall,fmax,fmaxall                     270
      REAL*8,dimension(nprocs) :: maxtemp_zval, mtempvalall, mtempavg_zval, mtempavgvalall
      INTEGER ::mind,mind2
      REAL*8, dimension(:,:,:), allocatable :: favg
!
      max_temp(1) = 0.0d0
      maxtemp_zval = 0.0d0
      mtempavg(1) = 0.0d0
      mtempavg_zval = 0.0d0
      qmax = 0.d0
      fmax = 0.d0                                                         280
      err = 0.0d0
      do i = 1,nvar
        indz(i) = 0
        if (sndim(i,2)-3==0) indz(i) = 1
      end do
!
      allocate(favg(lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
!
      DO k=lock(1)-dom_bound(3,1)+1,lock(2)-dom_bound(3,2)
```

```
      DO j = locj(1)−dom_bound(2,1)+1,locj(2)−dom_bound(2,2)                    290
        DO i = loci(1)−dom_bound(1,1)+1,loci(2)−dom_bound(1,2)
!
          favg(k,j,i) = (f(1,k,j,i)+f(1,k−1,j,i)+f(1,k,j−1,i)+f(1,k,j,i−1)&
            &+f(1,k−1,j−1,i)+f(1,k−1,j,i−1)+f(1,k,j−1,i−1)+f(1,k−1,j−1,i−1))/dfloat(8)
!
        end DO
      end DO
    end DO
!
      DO k=lock(1)−dom_bound(3,1),lock(2)−dom_bound(3,2)                        300
        DO j = locj(1)−dom_bound(2,1),locj(2)−dom_bound(2,2)
          DO i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)
!
            err = err+sqrt(((f(1,k,j,i) − ferr(1,k,j,i))**2))/TINF
            if (f(1,k,j,i).gt.max_temp(1)) then
              max_temp(1) = f(1,k,j,i)
              maxtemp_zval = f(nvar+3,k,j,i)
            end if
!
            if (favg(k,j,i).gt.mtempavg(1)) then                                310
              mtempavg(1) = favg(k,j,i)
              mtempavg_zval = (f(nvar+3,k,j,i)+&
                &f(nvar+3,k−indz(3),j−indz(2),i−indz(1)))/dfloat(2)
            end if
!
            if (Q1(k,j,i).gt.qmax) qmax=Q1(k,j,i)
            if (f(2,k,j,i).gt.fmax) fmax=f(2,k,j,i)
            ferr(1,k,j,i) = f(1,k,j,i)
!
          END DO                                                                320
        END DO
      END DO
      mtempavg(2) = id
      max_temp(2) = id
!
      CALL MPI_REDUCE(err,err_all,1,MPI_DOUBLE_PRECISION,&
        &MPI_SUM,0,MPI_COMM_WORLD,ierr)
      CALL MPI_REDUCE(max_temp,max_temp_all,1,MPI_2DOUBLE_PRECISION,&
        &MPI_MAXLOC,0,MPI_COMM_WORLD,ierr)
      CALL MPI_REDUCE(mtempavg,mtempavg_all,1,MPI_2DOUBLE_PRECISION,&           330
        &MPI_MAXLOC,0,MPI_COMM_WORLD,ierr)
      CALL MPI_REDUCE(qmax,qmaxall,1,MPI_DOUBLE_PRECISION,&
        &MPI_MAX,0,MPI_COMM_WORLD,ierr)
      CALL MPI_REDUCE(fmax,fmaxall,1,MPI_DOUBLE_PRECISION,&
        &MPI_MAX,0,MPI_COMM_WORLD,ierr)
!
      CALL MPI_GATHER(maxtemp_zval,1,MPI_DOUBLE_PRECISION,&
        &mtempvalall,1,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
!
      CALL MPI_GATHER(mtempavg_zval,1,MPI_DOUBLE_PRECISION,&                    340
        &mtempavgvalall,1,MPI_DOUBLE_PRECISION,0,MPI_COMM_WORLD,ierr)
!
```

```
        err_all = err_all/(nx1*nx2*nx3*1.d0)
        if (id==0) then
           mind = max_temp_all(2)+1
           mind2 = mtempavg_all(2)+1
           print 111, curr_time,err_all,max_temp_all(1),mtempvalall(mind),&
                &mtempavg_all(1),mtempavgvalall(mind2),qmaxall
           write(222,111) curr_time,err_all,max_temp_all(1),mtempvalall(mind)&
                &,mtempavg_all(1),mtempavgvalall(mind2)                          350
        end if
!
111     format(1x,1f10.5,2f10.6,1f6.3,1f10.6,1f6.3,2f10.6)
!
    End Subroutine track_error
!
!================================================
!================================================
!
!                                                                                360
    Subroutine ERROR
!
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: stat
!
         if (err_chk == 1) print *, 'Number of grid pts in all directions must be >= 5'
         if (err_chk == 2) print *, 'Number of time steps cannot be < 0'          370
         if (err_chk == 3) print *, 'ikutta is < 1'
         if (err_chk == 4) print *, 'All parameter values (Pr, Le. . . .) must be > 0'
         if (err_chk == 5) print *, 'Irestart must be 0 or 1'
         if (err_chk == 6) print *, 'Proc allocation error, pls check if nprocs > npts'
!
      if (err_chk > 0) CALL MPI_ABORT(MPI_COMM_WORLD,stat,ierr)
!
    End Subroutine ERROR
!================================================
!================================================                                380
!
  END MODULE vk_in
```

Grid Generation

```
MODULE gen_grid
contains
  SUBROUTINE grid
!
    USE vkvar
!
    IMPLICIT NONE
!
    INTEGER, target :: i,j,k
    INTEGER, pointer :: ii,jj,kk                                                 10
```

```
    INTEGER, dimension(2) :: indi,indj,indk
    REAL*8 :: s1, s2, s3
    REAL*8, dimension(3) :: dom,dx
    REAL*8, pointer, dimension(:) :: v1,v2,v3
!
    ALLOCATE(x1(0:nx1-1),xx1(0:nx1-1))
    ALLOCATE(x2(0:nx2-1),xx2(0:nx2-1))
!
!    SUBROUTINE sets up the grid for the 3D problem
!=================================================                    20
!
!    Caculate the min grid spacing for X1, X2 and X3
!=================================================
!
!    If dimension is periodic set value of boundary
!=================================================
!
    if (x1max==0.)x1max = 2.*acos(-1.)
    if (x2max==0.)x2max = 2.*acos(-1.)
    if (x3max==0.)x3max = 2.*acos(-1.)                                30
!
!
111 format(2x,3f6.3)
!
    dom(1) = (x1max-x1min)
    dom(2) = (x2max-x2min)
    dom(3) = (x3max-x3min)
!
    s1 = 1.
    s2 = 1.                                                           40
!
    do j=1,nx1-2
       s1 = s1+STRETCH1**(j)
    end do
!
    do k=1,nx2-2
       s2 = s2+STRETCH2**(k)
    end do
!
    s1 = dom(1)/s1                                                    50
    s2 = dom(2)/s2
!
    if (nx1.gt.0) then
       dx1 = dom(1)/((nx1-1.d0)*1.d0)
    else
       dx1 = 1.d0
    end if
!
        if (nx2.gt.0) then
           dx2 = dom(2)/((nx2-1.d0)*1.d0)                             60
        else
           dx2 = 1.d0
        end if
```

```
!
      if (nx3.gt.0) then
         dx3 = dom(3)/((nx3-1.d0)*1.d0)
      else
         dx3 = 1.d0
      end if
!                                                                      70
!      print *, 'dx', dx1,dx2,dx3
!      print *, nx1,nx2,nx3
!      if (id ==0) print *, 'dx3=', dx3
!
      DO i=0,nx1-1
         x1(i) = i*dx1*1.d0
      END DO
!
      DO j=0,nx2-1
         x2(j) = j*dx2*1.d0                                            80
      END DO
!
      xx1(0) = x1min
      do i = 1,nx1-1
         xx1(i) = xx1(i-1)+s1*STRETCH1**(i-1)
      end do
!
      xx2(0) = x2min
      do i = 1,nx2-1
         xx2(i) = xx2(i-1)+s2*STRETCH2**(i-1)                          90
      end do
!
      dx(1) = dx1
      dx(2) = dx2
      dx(3) = dx3
!
      do i = 1,3
         sndx(i) = dx(sndim(i,2))
      end do
                                                                       100
!      DO k =0,nx3-1
!      x3(k) = k*dx3
!      END DO
      do i = 1,2
         indi(i) = loci(i)-dom_bound(1,i)
         indj(i) = locj(i)-dom_bound(2,i)
         indk(i) = lock(i)-dom_bound(3,i)
      end do
!
!                                                                      110
      do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
         do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
            do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
               if (sndim(1,2) == 1) ii => i
               if (sndim(2,2) == 1) ii => j
```

```
                if (sndim(3,2) == 1) ii => k
                if (sndim(1,2) == 2) jj => i
                if (sndim(2,2) == 2) jj => j
                if (sndim(3,2) == 2) jj => k                                    120
                if (sndim(1,2) == 3) kk => i
                if (sndim(2,2) == 3) kk => j
                if (sndim(3,2) == 3) kk => k
!
                f(nvar+1,k,j,i) = xx1(ii)
                f(nvar+2,k,j,i) = xx2(jj)
                f(nvar+3,k,j,i) = x3(kk)

!
!                if (id==0) print 100, kk,jj,ii,f(nvar+1,k,j,i),f(nvar+2,k,j,i),f(nvar+3,k,j,i)   130
              end do
            end do
          end do
!
          100 format(2x,3i5,6f10.5)
          DEALLOCATE(x1,x2,x3)
!
      CONTAINS
!
!     Functions                                                                 140
!============
!
          FUNCTION BLOCK_LOW(proc_id,num_proc,num_elem)
            INTEGER :: BLOCK_LOW
            INTEGER, INTENT(IN) :: proc_id, num_proc
            REAL, INTENT(IN) :: num_elem
            BLOCK_LOW = proc_id*num_elem/num_proc
          END FUNCTION BLOCK_LOW
!
          FUNCTION BLOCK_HIGH(proc_id,num_proc,num_elem)                        150
            INTEGER :: BLOCK_HIGH
            INTEGER, INTENT(IN) :: proc_id, num_proc, num_elem
            BLOCK_HIGH = (proc_id+1)*num_elem/num_proc-1
          END FUNCTION BLOCK_HIGH
!
          FUNCTION BLOCK_SIZE(proc_id,num_proc,num_elem)
            INTEGER :: BLOCK_SIZE
            INTEGER, INTENT(IN) :: proc_id, num_proc
            REAL, INTENT(IN) :: num_elem
            BLOCK_SIZE = (proc_id+1)*num_elem/num_proc &                        160
                 &-proc_id*num_elem/num_proc
          END FUNCTION BLOCK_SIZE
!
      END SUBROUTINE grid
!
!
!*********************************************************************
!
!
```

```
      END MODULE gen_grid                                                    170

!      NONUNIFORM GRID
!      X1
!
!        xx1(0) = x1min
!        do i=1,nx1
!           xx1(i) = xx1(i−1)+s1*STRETCH1**(i−1)
!        end do
!
!                                                                           180
!        X2
!
!
!        xx2(0) = x2min
!        do i=1,nx2
!          xx2(i) = xx2(i−1)+s2*STRETCH2**(i−1)
!        end do
!
!
!       X3                                                                  190
!
!        xx3(0) = x3min
!        do i=1,nx3
!           xx3(i) = xx3(i−1)+s3*STRETCH3**(i−1)
!        end do
!
!     end if


                                                                            200
```

```
MODULE grid_map
!
  contains
!
    SUBROUTINE der_map
!
      USE vkvar
      USE der_ord_4
!
      IMPLICIT NONE                                                         10
!
      INTEGER :: i, j, k, l, ind
      INTEGER, dimension(3) :: indi, indj, indk
      REAL*8 :: a1, a2
!
      do i = 1,2
         indi(i) = loci(i) − dom_bound(1,i)
         indj(i) = locj(i) − dom_bound(2,i)
         indk(i) = lock(i) − dom_bound(3,i)
```

```
      end do                                                                    20
!
      do l = 1,nvar
!
         CALL prime(l,nvar+sndim(l,2))
         CALL pprime(l,nvar+sndim(l,2))
!
         do k = indk(1),indk(2)
            do j = indj(1), indj(2)
               do i = indi(1),indi(2)
!                                                                              30
                  cc1(1,l,k,j,i) = 1.d0/(f_prime(l,nvar+sndim(l,2),k,j,i))
                  cc1(2,l,k,j,i) = -f_pprime(l,nvar+sndim(l,2),k,j,i)/&
                       &((f_prime(l,nvar+sndim(l,2),k,j,i))**3)
!
                  a1 = abs(1.0d0 - cc1(1,l,k,j,i))
                  a2 = abs(0.0d0 - cc1(2,l,k,j,i))
!
                  if (a1.lt.1.0d-3) cc1(1,l,k,j,i) = 1.0d0
                  if (a2.lt.1.0d-3) cc1(2,l,k,j,i) = 0.0d0
!                                                                              40
               end do
            end do
         end do
!
      deallocate(f_prime,f_pprime)
!
   end do
!
      CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!                                                                              50
   12 format(a10,4f12.8)
!
 END SUBROUTINE der_map
!
!=============================================================
!=============================================================
!
  END MODULE grid_map
```

MPI Assignments and Domain Decomposition

```
MODULE vkpar
!
CONTAINS
!
SUBROUTINE grid_assign
!
! This subroutine initiates and completes all necessary allocations and
! functions needed for the parallel routine
!=============================================================
!                                                                              10
  USE vkvar
```

```
  USE vk_in
!
  IMPLICIT NONE
!
  INTEGER :: i,end_index,plc_hld,mid
  INTEGER, dimension(3) :: max_pts,min_pts
  INTEGER, dimension(3,3) :: ndim
  INTEGER, dimension(6) :: nrange
!                                                                              20
! In this subroutine, the procs are allocated such that the direction with the
! highest number of grid points is always assigned to the index i and the direction
! with the lowest number of grid points is assigned to the index k
! This was done to make the code perform faster, expecially in terms of do loops
!
! The array sndim is crucial in identifying the direction each index corresponds to
! The sndim(*,1) correspond the indices: 1=i, 2=j, 3=k
! The sndim(*,2) elements store the value of the directions: 1=r, 2=theta, 3=z
!==============================================================
!                                                                              30
  ndim(1,1) = nx1                    ! ndim stores the no. of grid pts and the periodicity info
  ndim(2,1) = nx2
  ndim(3,1) = nx3
!
  ndim(1,3) = grd1
  ndim(2,3) = grd2
  ndim(3,3) = grd3
  do i=1,3
     ndim(i,2) = 0
  end do                                                                       40
!
   if (x1max==0.) ndim(1,2) = 1
   if (x2max==0.) ndim(2,2) = 1
   if (x3max==0.) ndim(3,2) = 1
!
   if (nx1.ne.nx2.or.nx1.ne.nx3.or.nx2.ne.nx3) then
      max_pts = maxloc(ndim,1) ! identify the directions which have max and min grid pts
      min_pts = minloc(ndim,1)
   else
      max_pts(1) = 3                                                           50
      min_pts(1) = 1
   end if
!
   mid = 0
!
   do i = 1,3
      if (i.ne.min_pts(1).and.i.ne.max_pts(1)) mid = i
      periodic(i) = 0
   end do
!                                                                              60
!
   sndim(1,1) = ndim(max_pts(1),1) ! sndim(1) = i, sndim(2) = j, sndim(3) = k
   sndim(1,2) = max_pts(1)            ! sndim(2) stores the corresponding directions
   periodic(1) = ndim(max_pts(1),2) ! (r,theta,z) that above indices point to
```

```
   sndim(2,1) = ndim(mid,1)
   sndim(2,2) = mid
   periodic(2) = ndim(mid,2)
   sndim(3,1) = ndim(min_pts(1),1)
   sndim(3,2) = min_pts(1)
   periodic(3) = ndim(min_pts(1),2)                                    70
!
   if (grd1.eq.0.and.grd2.eq.0.and.grd3.eq.0) then
      if (mod(nprocs,8).eq.0) then
         dims(1) = 0
         dims(2) = 0
         dims(3) = 0
      else
         if (mod(nprocs,4).eq.0) then
            dims(2) = 0
            dims(3) = 1                                              80
         else
            dims(2) = 1
            dims(3) = 1
         end if
      end if
!
   else
      dims(1) = ndim(max_pts(1),3)
      dims(2) = ndim(mid,3)
      dims(3) = ndim(min_pts(1),3)                                   90
   end if
!
  ghostlayer = 2
  mpilayer = 1
!
  call MPI_DIMS_CREATE(nprocs,3,dims,ierr)
  call MPI_CART_CREATE(MPI_COMM_WORLD,3,dims,periodic,1,cart_comm,ierr)
  call MPI_COMM_RANK(cart_comm,id,ierr)
  call MPI_CART_SHIFT(cart_comm,0,1,idnbr(1),idnbr(2),ierr)
  call MPI_CART_SHIFT(cart_comm,1,1,idnbr(3),idnbr(4),ierr)          100
  call MPI_CART_SHIFT(cart_comm,2,1,idnbr(5),idnbr(6),ierr)
  call MPI_CART_COORDS(cart_comm,id,3,coords,ierr)
!
  do i = 1,3
     if (sndim(i,1).le.dims(i)) then
        err_chk = 6
        CALL ERROR
     end if
  end do
!                                                                    110
! Assign vectors for identifying boundary processors
!=================================================
!
  do i = 1,3
     dom_bound(i,1) = -ghostlayer
     dom_bound(i,2) = ghostlayer
  end do
```

```
!
  if (coords(1) == 0 .and. periodic(1) == 0) dom_bound(1,1) = 0
  if (coords(1) == dims(1)-1 .and. periodic(1) == 0) dom_bound(1,2) = 0
  if (coords(2) == 0 .and. periodic(2) == 0) dom_bound(2,1) = 0
  if (coords(2) == dims(2)-1 .and. periodic(2) == 0) dom_bound(2,2) = 0
  if (coords(3) == 0 .and. periodic(3) == 0 ) dom_bound(3,1) = 0
  if (coords(3) == dims(3)-1 .and. periodic(3) == 0) dom_bound(3,2) = 0
!
! Set up sub communicators for each dimension
!================================================
!
  free(1) = 1
  free(2) = 0
  free(3) = 0
!
  call MPI_CART_SUB(cart_comm,free,comm(1),ierr)
  call MPI_COMM_RANK(comm(1),rid(1),ierr)
  call MPI_COMM_SIZE(comm(1),rsize(1),ierr)
!
  free(1) = 0
  free(2) = 1
  free(3) = 0
!
  call MPI_CART_SUB(cart_comm,free,comm(2),ierr)
  call MPI_COMM_RANK(comm(2),rid(2),ierr)
  call MPI_COMM_SIZE(comm(2),rsize(2),ierr)
!
  free(1) = 0
  free(2) = 0
  free(3) = 1
!
  call MPI_CART_SUB(cart_comm,free,comm(3),ierr)
  call MPI_COMM_RANK(comm(3),rid(3),ierr)
  call MPI_COMM_SIZE(comm(3),rsize(3),ierr)
!
! Determine the start and end index for each processor in each dim
!=================================================================
!
  loci(1) = BLOCK_LOW(rid(1),rsize(1),sndim(1,1))
  loci(2) = BLOCK_HIGH(rid(1),rsize(1),sndim(1,1))
  locj(1) = BLOCK_LOW(rid(2),rsize(2),sndim(2,1))
  locj(2) = BLOCK_HIGH(rid(2),rsize(2),sndim(2,1))
  lock(1) = BLOCK_LOW(rid(3),rsize(3),sndim(3,1))
  lock(2) = BLOCK_HIGH(rid(3),rsize(3),sndim(3,1))
!
!
  x1size = BLOCK_SIZE(rid(1),rsize(1),sndim(1,1))
  x2size = BLOCK_SIZE(rid(2),rsize(2),sndim(2,1))
  x3size = BLOCK_SIZE(rid(3),rsize(3),sndim(3,1))
!
!
  totsize = x1size*x2size*x3size
!
```

```
! Buffers for Gathering
!======================
!
  allocate(allsize(nprocs))
  allocate(disp(nprocs))
  allocate(allrange(6*nprocs))
!
  nrange(1) = lock(1)
  nrange(2) = lock(2)
  nrange(3) = locj(1)                                                    180
  nrange(4) = locj(2)
  nrange(5) = loci(1)
  nrange(6) = loci(2)
!
  CALL MPI_ALLGATHER(totsize,1,MPI_INTEGER,allsize,1,&
       &MPI_INTEGER,cart_comm,ierr)
  CALL MPI_ALLGATHER(nrange,6,MPI_INTEGER,allrange,6,&
       &MPI_INTEGER,cart_comm,ierr)
!
  disp(1) = 0                                                            190
  do i = 2,nprocs
     disp(i) = allsize(i-1) + disp(i-1)
  end do
!
! Account for the ghost layers in the Boundary Processors
!========================================================
!
  if (dom_bound(1,1).ne.0) then
     loci(1) = loci(1) - ghostlayer
  end if                                                                 200
!
  if (dom_bound(1,2).ne.0) then
     loci(2) = loci(2) + ghostlayer
  end if
!
  if (dom_bound(2,1).ne.0) then
     locj(1) = locj(1) - ghostlayer
  end if
!
  if (dom_bound(2,2).ne.0) then                                          210
     locj(2) = locj(2) + ghostlayer
  end if
!
  if (dom_bound(3,1).ne.0) then
     lock(1) = lock(1) - ghostlayer
  end if
!
  if (dom_bound(3,2).ne.0) then
     lock(2) = lock(2) + ghostlayer
  end if                                                                 220
!
! Allocate buffers for final communications
!==========================================
```

```
!
  end_index = allsize(nprocs) + disp(nprocs)
  ALLOCATE(myres(totsize))
  ALLOCATE(allres(end_index))
!  ALLOCATE(out_f(1:fsize,0:x3size*rsize(3)-1,0:x2size*rsize(2)-1,0:x1size*rsize(1)-1))
!
! Print Decomposition information:                                                    230
!=========================
!
  if (id == 0) then
     print *, 'dim',sndim(1,2),':',dims(1),' dim',sndim(2,2),':', dims(2),&
          &' dim',sndim(3,2),':',dims(3)
     print *, 'Grid points / proc- nx1:', x1size,' nx2:',x2size,' nx3:',x3size
     print *, 'Total no of grid points :', nx1,'X',nx2,'X',nx3
  end if
!
  CONTAINS                                                                          240
    FUNCTION BLOCK_LOW(proc_id,num_proc,num_elem)
      INTEGER :: BLOCK_LOW
      INTEGER, INTENT(IN) :: proc_id, num_proc, num_elem
      BLOCK_LOW = proc_id*num_elem/num_proc
    END FUNCTION BLOCK_LOW
!
    FUNCTION BLOCK_HIGH(proc_id,num_proc,num_elem)
      INTEGER :: BLOCK_HIGH
      INTEGER, INTENT(IN) :: proc_id, num_proc, num_elem
      BLOCK_HIGH = (proc_id+1)*num_elem/num_proc-1                                  250
    END FUNCTION BLOCK_HIGH
!
    FUNCTION BLOCK_SIZE(proc_id,num_proc,num_elem)
      INTEGER :: BLOCK_SIZE
      INTEGER, INTENT(IN) :: proc_id, num_proc, num_elem
      BLOCK_SIZE = (proc_id+1)*num_elem/num_proc &
           &-proc_id*num_elem/num_proc
    END FUNCTION BLOCK_SIZE
!
!                                                                                   260
END SUBROUTINE grid_assign
!=====================================================
!=====================================================
!
!
 SUBROUTINE vk_communication(ind1,ind2)
!
   USE vkvar
!
   IMPLICIT NONE                                                                    270
!
   INTEGER :: i,j,k,index,procs,el,n
   INTEGER, INTENT(IN) :: ind1,ind2
!
   ALLOCATE(out_f(ind1:ind2,0:sndim(3,1)-1,0:sndim(2,1)-1,0:sndim(1,1)-1))
   do n = ind1,ind2
```

```
!
      index = 1
!
      do k = lock(1)−dom_bound(3,1),lock(2)−dom_bound(3,2)                    280
         do j = locj(1)−dom_bound(2,1),locj(2)−dom_bound(2,2)
            do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)
!
               myres(index) = f(n,k,j,i)
               index = index+1
!
            end do
         end do
      end do
!                                                                             290
      CALL MPI_ALLGATHERV(myres,totsize,MPI_REAL8,&
           &allres,allsize,disp,MPI_REAL8,&
           &MPI_COMM_WORLD,ierr)
!
      index = 1
!
      do procs = 0,nprocs−1
         el = 1 + procs*6
         do k = allrange(el),allrange(el+1)
            do j = allrange(el+2),allrange(el+3)                              300
               do i = allrange(el+4),allrange(el+5)
                  out_f(n,k,j,i) = allres(index)
                  index = index+1
               end do
            end do
         end do
      end do
!
   end do
!                                                                             310
 END SUBROUTINE vk_communication
!
!====================================
!====================================
!
!
    SUBROUTINE vk_buffer_comm(ind1,ind2)
!
!     This subroutince facilitates the communication
!     between adjacent processors                                             320
!=========================================================
!
!
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: ind,n,k,j,i,ii,jj,kk,arg1,arg2,p,tag1,tag2,dummy1, dummy2,stat
      real*8 :: test1, test2
```

```
      INTEGER, INTENT(IN) :: ind1,ind2                                          330
!
!     Allocate the send and receive buffers
!=============================================
!
      sbx1_size = (ind2-ind1+1)*(lock(2)-lock(1)+1+dom_bound(3,1)-dom_bound(3,2))&
           &*(locj(2)-locj(1)+1+dom_bound(2,1)-dom_bound(2,2))*ghostlayer
      sbx2_size = (ind2-ind1+1)*(lock(2)-lock(1)+1+dom_bound(3,1)-dom_bound(3,2))&
           &*(loci(2)-loci(1)+1+dom_bound(1,1)-dom_bound(1,2))*ghostlayer
      sbx3_size = (ind2-ind1+1)*(locj(2)-locj(1)+1+dom_bound(2,1)-dom_bound(2,2))&
           &*(loci(2)-loci(1)+1+dom_bound(1,1)-dom_bound(1,2))*ghostlayer        340
!
!
      ALLOCATE(send_buffx1_l(sbx1_size),recv_buffx1_r(sbx1_size))
      ALLOCATE(send_buffx1_r(sbx1_size),recv_buffx1_l(sbx1_size))
      ALLOCATE(send_buffx2_l(sbx2_size),recv_buffx2_r(sbx2_size))
      ALLOCATE(send_buffx2_r(sbx2_size),recv_buffx2_l(sbx2_size))
      ALLOCATE(send_buffx3_l(sbx3_size),recv_buffx3_r(sbx3_size))
      ALLOCATE(send_buffx3_r(sbx3_size),recv_buffx3_l(sbx3_size))
!
!                                                                               350
!     Communication in the i direction
!=========================================
!
!
      if (dims(1).gt.1.and.idnbr(2).gt.id.and.idnbr(2).gt.-1) then
         i = 1
!
         do ii = 2*ghostlayer-1,ghostlayer,-1
            do n = ind1,ind2
               do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)             360
                  do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
                     send_buffx1_r(i) = f(n,k,j,loci(2)-ii)
                     i = i+1
!
               end do
            end do
         end do
      end do
!                                                                               370
     CALL MPI_SENDRECV(send_buffx1_r,sbx1_size,&
            &MPI_REAL8,idnbr(2),0,recv_buffx1_r,sbx1_size,&
            &MPI_REAL8,idnbr(2),1,MPI_COMM_WORLD,stat,ierr)
!
      i = 1
!
      do ii = ghostlayer-1,0,-1
         do n = ind1,ind2
            do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
               do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)             380

                  f(n,k,j,loci(2)-ii) = recv_buffx1_r(i)
```

```
                i = i+1
!
              end do
            end do
          end do
        end do
!
      end if
!
      if (dims(1).gt.1.and.idnbr(1).lt.id.and.idnbr(1).gt.-1) then
!
      i = 1
!
      do ii = loci(1)+ghostlayer,loci(1)+2*ghostlayer-1
        do n = ind1,ind2
          do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
            do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
                send_buffx1_l(i) = f(n,k,j,ii)
                i = i+1
!
              end do
            end do
          end do
        end do
!
      CALL MPI_SENDRECV(send_buffx1_l,sbx1_size,&
           &MPI_REAL8,idnbr(1),1,recv_buffx1_l,sbx1_size,&
           &MPI_REAL8,idnbr(1),0,MPI_COMM_WORLD,stat,ierr)
!
      i = 1
!
      do ii = loci(1),loci(1)+1
        do n = ind1,ind2
          do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
            do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
                f(n,k,j,ii) = recv_buffx1_l(i)
                i = i+1
!
!
              end do
            end do
          end do
        end do
!
      end if
!
!
      call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
      if (dims(1).gt.1.and.periodic(1)==1.and.idnbr(2).lt.id) then
        i = 1
```

```
!
        do ii = 2*ghostlayer,ghostlayer+1,-1
           do n = ind1,ind2
              do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
                 do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)                    440
!
                    send_buffx1_r(i) = f(n,k,j,loci(2)-ii)
                    i = i+1
!
              end do
           end do
        end do
     end do
!
     CALL MPI_SENDRECV(send_buffx1_r,sbx1_size,&                                        450
          &MPI_REAL8,idnbr(2),4,recv_buffx1_r,sbx1_size,&
          &MPI_REAL8,idnbr(2),5,MPI_COMM_WORLD,stat,ierr)
!
     i = 1
!
     do ii = ghostlayer-1,0,-1
        do n = ind1,ind2
           do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
                                                                                        460
              f(n,k,j,loci(2)-ii) = recv_buffx1_r(i)
               i = i+1
!
              end do
           end do
        end do
     end do
!
   end if
!                                                                                       470
    if (dims(1).gt.1.and.periodic(1)==1.and.idnbr(1).gt.id) then
      i = 1
!
     do ii = loci(1)+ghostlayer+1,loci(1)+2*ghostlayer
        do n = ind1,ind2
           do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
                 send_buffx1_l(i) = f(n,k,j,ii)
                 i = i+1                                                                480
!
              end do
           end do
        end do
     end do
!
     CALL MPI_SENDRECV(send_buffx1_l,sbx1_size,&
          &MPI_REAL8,idnbr(1),5,recv_buffx1_l,sbx1_size,&
```

```
          &MPI_REAL8,idnbr(1),4,MPI_COMM_WORLD,stat,ierr)
!                                                                          490
      i = 1
!
      do ii = loci(1),loci(1)+1
         do n = ind1,ind2
            do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
               do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
                  f(n,k,j,ii) = recv_buffx1_l(i)
                  i = i+1
!                                                                          500
               end do
            end do
         end do
      end do
   end if
!
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
! Communication in the j direction
!====================================                                      510
!
!
   if (dims(2).gt.1.and.idnbr(4).gt.id.and.idnbr(4).gt.-1) then
!
      j = 1
!
      do ii = 2*ghostlayer-1,ghostlayer,-1
!
         do n = ind1,ind2
            do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)          520
               do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                  send_buffx2_r(j) = f(n,k,locj(2)-ii,i)
                  j = j+1
!
               end do
            end do
         end do
      end do
!                                                                          530
      CALL MPI_SENDRECV(send_buffx2_r,sbx2_size,&
          &MPI_REAL8,idnbr(4),id,recv_buffx2_r,sbx2_size,&
          &MPI_REAL8,idnbr(4),idnbr(4),MPI_COMM_WORLD,stat,ierr)
!
      j = 1
!
      do ii = ghostlayer-1,0,-1
         do n = ind1,ind2
            do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
               do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)       540
!
```

```
                f(n,k,locj(2)−ii,i) = recv_buffx2_r(j)
                j = j+1
!
             end do
          end do
       end do
    end do
 end if
!                                                                                    550
 if (dims(2).gt.1.and.idnbr(3).lt.id.and.idnbr(3).gt.−1) then
!
    j = 1
!
    do ii = locj(1)+ghostlayer,locj(1)+2*ghostlayer−1
       do n = ind1,ind2
          do k = lock(1)−dom_bound(3,1),lock(2)−dom_bound(3,2)
             do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)
!
                send_buffx2_l(j) = f(n,k,ii,i)                                       560
                j = j+1
!
             end do
          end do
       end do
    end do
!
    CALL MPI_SENDRECV(send_buffx2_l,sbx2_size,&
         &MPI_REAL8,idnbr(3),id,recv_buffx2_l,sbx2_size,&
         &MPI_REAL8,idnbr(3),idnbr(3),MPI_COMM_WORLD,stat,ierr)                      570
!
    j = 1
!
    do ii = locj(1),locj(1)+1
       do n = ind1,ind2
          do k = lock(1)−dom_bound(3,1),lock(2)−dom_bound(3,2)
             do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)

                f(n,k,ii,i) = recv_buffx2_l(j)
                j = j+1                                                              580
!
             end do
          end do
       end do
!
    end do
!
 end if
!
 CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)                                               590
!
 if (dims(2).gt.1.and.periodic(2)==1.and.idnbr(4).lt.id) then
     j = 1
!
```

```
        do ii = 2*ghostlayer,ghostlayer+1,-1
          do n = ind1,ind2
            do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
              do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                send_buffx2_r(j) = f(n,k,locj(2)-ii,i)                    600
                j = j+1
!
            end do
          end do
        end do
      end do
!
      CALL MPI_SENDRECV(send_buffx2_r,sbx2_size,&
           &MPI_REAL8,idnbr(4),4,recv_buffx2_r,sbx2_size,&
           &MPI_REAL8,idnbr(4),5,MPI_COMM_WORLD,stat,ierr)                 610
!
      j = 1
!
      do ii = ghostlayer-1,0,-1
        do n = ind1,ind2
          do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
            do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)

              f(n,k,locj(2)-ii,i) = recv_buffx2_r(j)
              j = j+1                                                     620
!
            end do
          end do
        end do
      end do
!
    end if
!
     if (dims(2).gt.1.and.periodic(2)==1.and.idnbr(3).gt.id) then
!                                                                         630
       j = 1
!
      do ii = locj(1)+ghostlayer+1,locj(1)+2*ghostlayer
        do n = ind1,ind2
          do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
            do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
              send_buffx2_l(j) = f(n,k,ii,i)
              j = j+1
!                                                                         640
            end do
          end do
        end do
      end do
!
      CALL MPI_SENDRECV(send_buffx2_l,sbx2_size,&
          &MPI_REAL8,idnbr(3),5,recv_buffx2_l,sbx2_size,&
```

```
          &MPI_REAL8,idnbr(3),4,MPI_COMM_WORLD,stat,ierr)
!
      j = 1                                                                    650
!
      do ii = locj(1),locj(1)+1
         do n = ind1,ind2
            do k = lock(1)−dom_bound(3,1),lock(2)−dom_bound(3,2)
               do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)
!
                  f(n,k,ii,i) = recv_buffx2_l(j)
                  j = j+1
!
               end do                                                          660
            end do
         end do
      end do
   end if
!
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
   !
!     Communication in the k direction
!=================================================================================  670
!
   if (dims(3).gt.1.and.idnbr(6).gt.id.and.idnbr(6).gt.−1) then
      k = 1
!
         do ii = 2*ghostlayer−1,ghostlayer,−1
            do n = ind1,ind2
               do j = locj(1)−dom_bound(2,1),locj(2)−dom_bound(2,2)
                  do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)
!
                     send_buffx3_r(k) = f(n,lock(2)−ii,j,i)                    680
                     k = k+1
!
               end do
            end do
         end do
      end do
!
     CALL MPI_SENDRECV(send_buffx3_r,sbx3_size,&
             &MPI_REAL8,idnbr(6),4,recv_buffx3_r,sbx3_size,&
             &MPI_REAL8,idnbr(6),5,MPI_COMM_WORLD,stat,ierr)                   690
!
      k = 1
!
      do ii = ghostlayer−1,0,−1
         do n = ind1,ind2
            do j = locj(1)−dom_bound(2,1),locj(2)−dom_bound(2,2)
               do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)

                  f(n,lock(2)−ii,j,i) = recv_buffx3_r(k)
                  k = k+1                                                      700
```

```
!
                end do
              end do
            end do
          end do
!
        end if
!
         if (dims(3).gt.1.and.idnbr(5).lt.id.and.idnbr(5).gt.-1) then
           k = 1                                                                    710
!
          do ii = lock(1)+ghostlayer,lock(1)+2*ghostlayer-1
            do n = ind1,ind2
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
                do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                  send_buffx3_l(k) = f(n,ii,j,i)
                  k = k+1
!
                end do                                                              720
              end do
            end do
          end do
!
          CALL MPI_SENDRECV(send_buffx3_l,sbx3_size,&
              &MPI_REAL8,idnbr(5),5,recv_buffx3_l,sbx3_size,&
              &MPI_REAL8,idnbr(5),4,MPI_COMM_WORLD,stat,ierr)
!
          k = 1
!                                                                                   730
          do ii = lock(1),lock(1)+1
            do n = ind1,ind2
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
                do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                  f(n,ii,j,i) = recv_buffx3_l(k)
                  k = k+1
!
                end do
              end do                                                                740
            end do
          end do
!
        end if
!
        call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
        if (dims(3).gt.1.and.periodic(3)==1.and.idnbr(6).lt.id) then
          k = 1
!                                                                                   750
          do ii = 2*ghostlayer,ghostlayer+1,-1
            do n = ind1,ind2
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
```

```
                do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                  send_buffx3_r(k) = f(n,lock(2)-ii,j,i)
                  k = k+1
!
                end do
              end do                                                    760
            end do
          end do
!
          CALL MPI_SENDRECV(send_buffx3_r,sbx3_size,&
               &MPI_REAL8,idnbr(6),4,recv_buffx3_r,sbx3_size,&
               &MPI_REAL8,idnbr(6),5,MPI_COMM_WORLD,stat,ierr)
!
          k = 1
!
          do ii = ghostlayer-1,0,-1                                     770
            do n = ind1,ind2
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
                do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)

                  f(n,lock(2)-ii,j,i) = recv_buffx3_r(k)
                  k = k+1
!
                end do
              end do
            end do                                                      780
          end do
!
        end if
!
        if (dims(3).gt.1.and.periodic(3)==1.and.idnbr(5).gt.id) then
          k = 1
!
          do ii = lock(1)+ghostlayer+1,lock(1)+2*ghostlayer
            do n = ind1,ind2
              do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)      790
                do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                  send_buffx3_l(k) = f(n,ii,j,i)
                  k = k+1
!
                end do
              end do
            end do
          end do
!                                                                       800
          CALL MPI_SENDRECV(send_buffx3_l,sbx3_size,&
               &MPI_REAL8,idnbr(5),5,recv_buffx3_l,sbx3_size,&
               &MPI_REAL8,idnbr(5),4,MPI_COMM_WORLD,stat,ierr)
!
          k = 1
!
```

```
      do ii = lock(1),lock(1)+1
         do n = ind1,ind2
            do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
               do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)                    810
!
                  f(n,ii,j,i) = recv_buffx3_l(k)
                  k = k+1
!
               end do
            end do
         end do
      end do
   end if
!                                                                                     820
   call MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
!  Dealocate all send and recv buffers
!=======================================
!
   DEALLOCATE(send_buffx1_l,recv_buffx1_r)
   DEALLOCATE(send_buffx1_r,recv_buffx1_l)
   DEALLOCATE(send_buffx2_l,recv_buffx2_r)
   DEALLOCATE(send_buffx2_r,recv_buffx2_l)
   DEALLOCATE(send_buffx3_l,recv_buffx3_r)                                           830
   DEALLOCATE(send_buffx3_r,recv_buffx3_l)
!
 END SUBROUTINE vk_buffer_comm
!
!=========================================================
!=========================================================
!
END MODULE vkpar
```

Derivative Routines

```
!
!      This package has the derivati'ye routines and the
!        Runge-Kutta routines for parallel implementation
!
!
!=================================================================================
!=================================================================================
!
MODULE der_ord_4
  contains                                                                         10
!
      SUBROUTINE prime(ivar,nvec)
!     This subroutine calculates first derivatives
!     The following format should be used when calling this subroutine
!     f_prime(Independent variable,Dependent variable,x3loc,x2loc,x1loc)
!     Independent variable: x1=1
!                           x2=2
```

```
!                       x3=3
!     Dependent variable: The variable for which the derivative is being calculated (T,Yi,u,v,w.....)
!     x3loc:              The x3 location                                                          20
!     x2loc:              The x2 location
!     x1loc:              The x1 location
!
      USE vkvar
!
      IMPLICIT NONE
!
      INTEGER :: a1,b1,c1, i, j, k,ii,jj,kk
      INTEGER, DIMENSION(:) :: indi(2),indj(2),indk(2)
      INTEGER, INTENT(IN) :: ivar, nvec                                                            30
      REAL*8 :: h, h1, gl
!
    ALLOCATE(f_prime(ivar:ivar,nvec:nvec,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
!
! Calculate dfdx1
!===============
!
       if (ivar == 1) then
          h = sndx(ivar)
         ! print *, 'h', h,ivar,sndim(3,2),sndx(3)                                               40
          h1 = 1.0/(12.0*h)
          a1 = 1
          b1 = 0
          c1 = 0
!
          do k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
             do j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
                kk = k - lock(1)
                jj = j - locj(1)                                                               50
                indi(1) = loci(1)-dom_bound(1,1) - loci(1)
                indi(2) = loci(2)-dom_bound(1,2) - loci(1)
!
                if (dom_bound(1,1) == 0) then
                   ii = 0;i = 0;
                   f_prime(ivar,nvec,k,j,i) = bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                   ii = 1;i = 1;
                   f_prime(ivar,nvec,k,j,i) = bnd_1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                   indi(1) = indi(1)+ghostlayer;
                end if                                                                         60
!
                if (dom_bound(1,2) == 0) then
                   ii = loci(2)-loci(1)-1;i = loci(2)-1
                   f_prime(ivar,nvec,k,j,i) = bnd_n1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                   ii = loci(2)-loci(1);i = loci(2)
                   f_prime(ivar,nvec,k,j,i) = bnd_n(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                   indi(2) = indi(2)-ghostlayer;
                end if
!
                do ii = indi(1),indi(2)                                                        70
```

```
                    i = ii+loci(1)
                    f_prime(ivar,nvec,k,j,i) = mid(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                  end do
!
               end do
            end do
!
         else
!
! Calculate dfdx2                                                    80
!================*
            if (ivar==2) then
               h = sndx(ivar)
               h1 = 1.0/(12.0*h)
               a1 = 0
               b1 = 1
               c1 = 0
!
               do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
                  do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)   90
!
                     kk = k - lock(1)
                     ii = i - loci(1)
                     indj(1) = locj(1)-dom_bound(2,1) - locj(1)
                     indj(2) = locj(2)-dom_bound(2,2) - locj(1)
!
                     if (dom_bound(2,1) == 0) then
                        jj = 0;j = 0;
                        f_prime(ivar,nvec,k,j,i) = bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                        jj = 1;j = 1;                                     100
                        f_prime(ivar,nvec,k,j,i) = bnd_1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                        indj(1) = indj(1) + ghostlayer
                     end if
!
                     if (dom_bound(2,2) == 0) then
                        jj = locj(2)-locj(1)-1;j = locj(2)-1;
                        f_prime(ivar,nvec,k,j,i) = bnd_n1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                        jj = locj(2)-locj(1);j = locj(2)
                        f_prime(ivar,nvec,k,j,i) = bnd_n(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                        indj(2) = indj(2) - ghostlayer                    110
                     end if
!
                     do jj = indj(1),indj(2)
                        j = jj+locj(1)
                        f_prime(ivar,nvec,k,j,i) = mid(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                     end do
!
                  end do
               end do
            else                                                          120
!
! Calculate dfdx3
!================*
```

```
            if (ivar==3) then
               h = sndx(ivar)
               h1 = 1.0/(12.0*h)
               a1 = 0
               b1 = 0
               c1 = 1
               do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)                    130
                  do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
                     jj = j - locj(1)
                     ii = i - loci(1)
                     indk(1) = lock(1)-dom_bound(3,1) - lock(1)
                     indk(2) = lock(2)-dom_bound(3,2) - lock(1)
!
                     if (dom_bound(3,1) == 0) then
                        kk = 0;k = 0;
                        f_prime(ivar,nvec,k,j,i) = bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1)       140
                        kk = 1;k = 1;
                        f_prime(ivar,nvec,k,j,i) = bnd_1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                        indk(1) = indk(1) + ghostlayer
                     end if
!
                     if (dom_bound(3,2) == 0) then
                        kk = lock(2)-lock(1)-1;k = lock(2)-1;
                        f_prime(ivar,nvec,k,j,i) = bnd_n1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                        kk = lock(2)-lock(1);k = lock(2);
                        f_prime(ivar,nvec,k,j,i) = bnd_n(f,nvec,ii,jj,kk,a1,b1,c1,h1)       150
                        indk(2) = indk(2) - ghostlayer
                     end if
!
                     do kk = indk(1),indk(2)
                        k = kk + lock(1)
                        f_prime(ivar,nvec,k,j,i) = mid(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                     end do
!
                  end do
               end do                                                                       160
            endif
         end if
      end if
   CONTAINS
!=======================================================
      ! Functions for calculating the above derivatives
!=======================================================
      !
      ! 0 th Boundary
      !_______________                                                                     170
   function bnd_0(p,nv1,ll1,mm1,nn1,r,s,t,u)
     REAL*8::bnd_0
     INTEGER :: l1, m1,n1
     REAL*8,intent(in),dimension(:,:,:,:) :: p
     INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
     real*8, intent(in) :: u
```

```
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
      bnd_0=(-3.0*p(nv1,n1+4*t,m1+4*s,l1+4*r)&                                  180
           &+16.0*p(nv1,n1+3*t,m1+3*s,l1+3*r)&
           &-36.0*p(nv1,n1+2*t,m1+2*s,l1+2*r)&
           &+48.0*p(nv1,n1+t,m1+s,l1+r)&
           &-25.0*p(nv1,n1,m1,l1))*u
    end function bnd_0
    !
    ! 0-1 Boundary
    !-------------
    function bnd_1(p,nv1,ll1,mm1,nn1,r,s,t,u)
      !                                                                         190
      ! Declare variables
      !
      REAL*8 :: bnd_1
      REAL*8,intent(in),dimension(:,:,:,:) :: p
      INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
      INTEGER :: l1, m1, n1
      real*8, intent(in) :: u
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!                                                                             200
      bnd_1 = (-3.0*p(nv1,n1-t,m1-s,l1-r)&
           &-10.0*p(nv1,n1,m1,l1)&
           &+18.0*p(nv1,n1+t,m1+s,l1+r)&
           &-6.0*p(nv1,n1+2*t,m1+2*s,l1+2*r)&
           &+p(nv1,n1+3*t,m1+3*s,l1+3*r))*u
!
    end function bnd_1
    !
    ! Middle
    !-------                                                                   210
    function mid(p,nv1,ll1,mm1,nn1,r,s,t,u)
      REAL*8 :: mid
      REAL*8, intent(in), dimension(:,:,:,:) :: p
      INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
      INTEGER :: l1, m1, n1
      real*8, intent(in) :: u
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
      mid = (p(nv1,n1-2*t,m1-2*s,l1-2*r)&                                       220
           &-8.0*p(nv1,n1-t,m1-s,l1-r)&
           &+8.0*p(nv1,n1+t,m1+s,l1+r)&
           &-p(nv1,n1+2*t,m1+2*s,l1+2*r))*u
!
    end function mid
    !
    ! N-1 boundary
    !-------------
    function bnd_n1(p,nv1,ll1,mm1,nn1,r,s,t,u)
```

```
      REAL*8::bnd_n1
      REAL*8,intent(in),dimension(:,:,:,:) :: p
      INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
      INTEGER :: l1, m1, n1
      real*8, intent(in) :: u
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
      bnd_n1=-(-3.0*p(nv1,n1+t,m1+s,l1+r)&
           &-10.0*p(nv1,n1,m1,l1)&
           &+18.0*p(nv1,n1-t,m1-s,l1-r)&
           &-6.0*p(nv1,n1-2*t,m1-2*s,l1-2*r)&
           &+p(nv1,n1-3*t,m1-3*s,l1-3*r))*u
!
    end function bnd_n1
    !
    ! Nth Boundary
    !-------------
    function bnd_n(p,nv1,ll1,mm1,nn1,r,s,t,u)
      REAL*8 :: bnd_n
      REAL*8,intent(in),dimension(:,:,:,:) :: p
      INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
      real*8, intent(in) :: u
      INTEGER :: l1, m1, n1
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
      bnd_n = -(-3.0*p(nv1,n1-4*t,m1-4*s,l1-4*r)&
           &+16.0*p(nv1,n1-3*t,m1-3*s,l1-3*r)&
           &-36.0*p(nv1,n1-2*t,m1-2*s,l1-2*r)&
           &+48.0*p(nv1,n1-t,m1-s,l1-r)&
           &-25.0*p(nv1,n1,m1,l1))*u
!
    end function bnd_n
!
  END SUBROUTINE prime
!
!=======================================================================
!=======================================================================
    SUBROUTINE pprime(ivar,nvec)
!   This subroutine calculates second derivatives
!   The following format should be used when calling this subroutine
!   dfdx(Independent variable,Dependent variable,x3loc,x2loc,x1loc)
!   Independent variable: x1=1
!                         x2=2
!                         x3=3
!   Dependent variable: The variable for which the derivative is being calculated (T,Yi,u,v,w.....)
!   x3loc:              The x3 location
!   x2loc:              The x2 location
!   x1loc:              The x1 location
!
      USE vkvar
!
```

```
      IMPLICIT NONE
!
      REAL*8 :: h, h1
      INTEGER :: a1,b1,c1,i,j,k,ii,jj,kk
      INTEGER, DIMENSION(:) :: gl(6),indi(2),indj(2),indk(2)
      INTEGER, INTENT(IN) :: ivar,nvec
!
    ALLOCATE(f_pprime(ivar:ivar,nvec:nvec,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))   290
!     df2dx1dx1
      if (ivar==1) then
        h = sndx(ivar)
        h1 = 1.0/(12.0*h*h)
        a1 = 1
        b1 = 0
        c1 = 0
!
        do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
          do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)                          300
!
            kk = k - lock(1)
            jj = j - locj(1)
            indi(1) = loci(1)-dom_bound(1,1) - loci(1)
            indi(2) = loci(2)-dom_bound(1,2) - loci(1)
!
            if (dom_bound(1,1) == 0) then
              ii = 0;i = 0;
              f_pprime(ivar,nvec,k,j,i) = bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1)
              ii = 1;i = 1;                                                                  310
              f_pprime(ivar,nvec,k,j,i) = bnd_1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
              indi(1) = indi(1) + ghostlayer
            end if
!
            if (dom_bound(1,2) == 0) then
              ii = loci(2)-loci(1)-1;i = loci(2)-1;
              f_pprime(ivar,nvec,k,j,i) = bnd_n1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
              ii = loci(2)-loci(1);i = loci(2)
              f_pprime(ivar,nvec,k,j,i) = bnd_n(f,nvec,ii,jj,kk,a1,b1,c1,h1)
              indi(2) = indi(2) - ghostlayer                                                 320
            end if
!
            do ii = indi(1),indi(2)
              i = ii + loci(1)
              f_pprime(ivar,nvec,k,j,i) = mid(f,nvec,ii,jj,kk,a1,b1,c1,h1)
            end do
!
          end do
        end do
                                                                                           330
      else
        if (ivar==2) then
!         df2dx2dx2
          h = sndx(ivar)
          h1 = 1.0/(12.0*h*h)
```

```
            a1 = 0
            b1 = 1
            c1 = 0
!
            do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)                    340
               do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
                  kk = k - lock(1)
                  ii = i - loci(1)
                  indj(1) = locj(1)-dom_bound(2,1) - locj(1)
                  indj(2) = locj(2)-dom_bound(2,2) - locj(1)
!
                  if (dom_bound(2,1) == 0) then
                     jj = 0; j = 0;
                     f_pprime(ivar,nvec,k,j,i) = bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1)       350
                     jj = 1; j = 1;
                     f_pprime(ivar,nvec,k,j,i) = bnd_1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                     indj(1) = indj(1) + ghostlayer

                  end if
!
                  if (dom_bound(2,2) == 0) then
                     jj = locj(2)-locj(1)-1;j = locj(2) - 1
                     f_pprime(ivar,nvec,k,j,i) = bnd_n1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                     jj = locj(2)-locj(1); j = locj(2)                                   360
                     f_pprime(ivar,nvec,k,j,i) = bnd_n(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                     indj(2) = indj(2) - ghostlayer
                  end if
!
                  do jj = indj(1),indj(2)
                     j  = jj +locj(1)
                     f_pprime(ivar,nvec,k,j,i) = mid(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                  end do
!
               end do                                                                   370
            end do
!
         else
!
            if (ivar == 3) then
!           df2dx3dx3
               h = sndx(ivar)
               h1 = 1.0/(12.0*h*h)
               a1 = 0
               b1 = 0                                                                   380
               c1 = 1
!
               do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)
                  do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
                     jj = j - locj(1)
                     ii = i - loci(1)
                     indk(1) = lock(1)-dom_bound(3,1) - lock(1)
```

```
                    indk(2) = lock(2)−dom_bound(3,2) − lock(1)
!                                                                                    390
                    if (dom_bound(3,1) == 0) then
                       kk = 0; k = 0;
                       f_pprime(ivar,nvec,k,j,i) = bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                       kk = 1; k = 1;
                       f_pprime(ivar,nvec,k,j,i) = bnd_1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                       indk(1) = indk(1) + ghostlayer
                    end if
!
                    if (dom_bound(3,2) == 0) then
                       kk = lock(2)−lock(1)−1; k = lock(2)−1                         400
                       f_pprime(ivar,nvec,k,j,i) = bnd_n1(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                       kk = lock(2)−lock(1); k = lock(2)
                       f_pprime(ivar,nvec,k,j,i) = bnd_n(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                       indk(2) = indk(2) − ghostlayer
                    end if
!
                    do kk = indk(1),indk(2)
                       k = kk+lock(1)
                       f_pprime(ivar,nvec,k,j,i) = mid(f,nvec,ii,jj,kk,a1,b1,c1,h1)
                    end do                                                           410
!
                 end do
              end do
!
           end if
        end if
     end if
!
  CONTAINS
!=============================================================================== 4
     !
     ! 0th Boundary
     !_____________
     function bnd_0(p,nv1,ll1,mm1,nn1,r,s,t,u)
       REAL*8::bnd_0
       REAL*8,intent(in),dimension(:,:,:,:) :: p
       INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
       real*8, intent(in) :: u
       INTEGER :: l1, m1, n1
!                                                                                   430
       l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
       bnd_0=(11.0*p(nv1,n1+4*t,m1+4*s,l1+4*r)&
            &−56.0*p(nv1,n1+3*t,m1+3*s,l1+3*r)&
            &+114.0*p(nv1,n1+2*t,m1+2*s,l1+2*r)&
            &−104.0*p(nv1,n1+t,m1+s,l1+r)&
            &+35.0*p(nv1,n1,m1,l1))*u
!
     end function bnd_0
     !                                                                              440
     ! 0−1 Boundary
```

```
      !-------------
      function bnd_1(p,nv1,ll1,mm1,nn1,r,s,t,u)
        REAL*8 :: bnd_1
        REAL*8,intent(in),dimension(:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
        real*8, intent(in) :: u
        INTEGER :: l1, m1, n1
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1                                    450
!
        bnd_1 = (11.0*p(nv1,n1-1*t,m1-1*s,l1-1*r)&
             &-20.0*p(nv1,n1,m1,l1)&
             &+6.0*p(nv1,n1+t,m1+s,l1+r)&
             &+4.0*p(nv1,n1+2*t,m1+2*s,l1+2*r)&
             &-p(nv1,n1+3*t,m1+3*s,l1+3*r))*u
!
      end function bnd_1
      !
      ! Middle                                                        460
      !-------
      function mid(p,nv1,ll1,mm1,nn1,r,s,t,u)
        REAL*8 :: mid
        REAL*8, intent(in), dimension(:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
        real*8, intent(in) :: u
        INTEGER :: l1, m1, n1
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!                                                                     470
        mid = (-p(nv1,n1-2*t,m1-2*s,l1-2*r)&
             &+16.0*p(nv1,n1-t,m1-s,l1-r)&
             &-30.0*p(nv1,n1,m1,l1)&
             &+16.0*p(nv1,n1+t,m1+s,l1+r)&
             &-p(nv1,n1+2*t,m1+2*s,l1+2*r))*u
!
      end function mid
      !
      ! N-1 Boundary
      !-------------                                                 480
      function bnd_n1(p,nv1,ll1,mm1,nn1,r,s,t,u)
        REAL*8::bnd_n1
        REAL*8,intent(in),dimension(:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
        real*8, intent(in) :: u
        INTEGER :: l1, m1, n1
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        bnd_n1=(11.0*p(nv1,n1+t,m1+s,l1+r)&                             490
             &-20.0*p(nv1,n1,m1,l1)&
             &+6.0*p(nv1,n1-t,m1-s,l1-r)&
             &+4.0*p(nv1,n1-2*t,m1-2*s,l1-2*r)&
             &-p(nv1,n1-3*t,m1-3*s,l1-3*r))*u
```

```
!
      end function bnd_n1
      !
      ! Nth Boundary
      !-------------
      function bnd_n(p,nv1,ll1,mm1,nn1,r,s,t,u)                                    500
        REAL*8 :: bnd_n
        REAL*8,intent(in),dimension(:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t
        real*8, intent(in) :: u
        INTEGER :: l1, m1, n1
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        bnd_n = (11.0*p(nv1,n1-4*t,m1-4*s,l1-4*r)&
             &+35.0*p(nv1,n1,m1,l1)&                                               510
             &-104.0*p(nv1,n1-t,m1-s,l1-r)&
             &+114.0*p(nv1,n1-2*t,m1-2*s,l1-2*r)&
             &-56.0*p(nv1,n1-3*t,m1-3*s,l1-3*r))*u
!
      end function bnd_n
      !
    END SUBROUTINE pprime
      !
!=====================================================================
!===================================================================== 520
      SUBROUTINE f1f2prime(ivar1,ivar2,nvec)
!     This subroutine calculates cross derivatives
!     The following format should be used when calling this subroutine
!     dfdx(Independent variable1,Independent variable2,Dependent variable,x3loc,x2loc,x1loc)
!     Independent variable: x1=1
!                           x2=2
!                           x3=3
!     Dependent variable: The variable for which the derivative is being calculated (T,Yi,u,v,w.....)
!     x3loc:              The x3 location
!     x2loc:              The x2 location                                          530
!     x1loc:              The x1 location
!
        USE vkvar
!
        IMPLICIT NONE
!
        INTEGER :: a1,b1,c1,var,i,j,k
        INTEGER, DIMENSION(:) :: gl(6),indi(2),indj(2),indk(2)
        INTEGER, INTENT(IN) :: ivar1,ivar2,nvec
        REAL*8 :: h,h1                                                            540
!
      ALLOCATE(f_cpprime(ivar1:ivar1,ivar2:ivar2,nvec:nvec,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
      call prime(ivar1,nvec)
      !      df2dx1dx2
      if (ivar1==1 .and. ivar2==2) then
         h = sndx(ivar1)
         h1 = 1.0/(12.0*h)
```

```
      a1 = 0
      b1 = 1
      c1 = 0                                                                    550
      do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
        do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
          if (dom_bound(2,1) == 0) then
            j = 0
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_0(f_prime,ivar1,nvec,i,j,k,a1,b1,c1,h1)
            j = 1
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_1(f_prime,ivar1,nvec,i,j,k,a1,b1,c1,h1)
          end if
!                                                                             560
          do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = mid(f_prime,ivar1,nvec,i,j,k,a1,b1,c1,h1)
          end do
!
          if (dom_bound(2,2) == 0) then
            j = locj(2)-1
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_n1(f_prime,ivar1,nvec,i,j,k,a1,b1,c1,h1)
            j = locj(2)
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_n(f_prime,ivar1,nvec,i,j,k,a1,b1,c1,h1)
          end if                                                                570
!
        end do
      end do
      deallocate(f_prime)
    end if
    if (ivar2==3 .and. ivar1==1 .or. ivar1==2 ) then
      !
      var = 1
      !     df2dx1dx3
      do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)                 580
        do j = locj(1) - dom_bound(2,1), lock(2) - dom_bound(2,2)
          do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
          end do
        end do
      end do
      h = sndx(ivar2)
      h1 = 1.0/(12.0*h)
      a1 = 0
      b1 = 0
      c1 = 1                                                                    590
      do j = locj(1) - dom_bound(2,1), lock(2) - dom_bound(2,2)
        do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
          if (dom_bound(3,1) == 0) then
            k = 0
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_0(f_prime,var,nvec,i,j,k,a1,b1,c1,h1)
            k = 1
            f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_1(f_prime,var,nvec,i,j,k,a1,b1,c1,h1)
          end if
!                                                                             600
```

```
            do k = lock(1) − dom_bound(3,1), lock(2) − dom_bound(3,2)
               f_cpprime(ivar1,ivar2,nvec,k,j,i) = mid(f_prime,var,nvec,i,j,k,a1,b1,c1,h1)
            end do
!
            if (dom_bound(3,2) == 0) then
               k = lock(2)−1
               f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_n1(f_prime,var,nvec,i,j,k,a1,b1,c1,h1)
               k = lock(2)
               f_cpprime(ivar1,ivar2,nvec,k,j,i) = bnd_n(f_prime,var,nvec,i,j,k,a1,b1,c1,h1)
            end if                                                                          610
!
         end do
       end do
       deallocate(f_prime)
    end if
  CONTAINS
!=================================================================
    !
    ! 0th Boundary
    !_____________                                                                          620
    function bnd_0(p,q,nv1,ll1,mm1,nn1,r,s,t,u)
      REAL*8::bnd_0
      INTEGER :: l1, m1,n1
      REAL*8,intent(in),dimension(:,:,:,:,:) :: p
      INTEGER, intent(in) :: nv1,q,ll1,mm1,nn1,r,s,t
      real*8, intent(in) :: u
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
      bnd_0=(−3.0*p(q,nv1,n1+4*t,m1+4*s,l1+4*r)&                                           630
           &+16.0*p(q,nv1,n1+3*t,m1+3*s,l1+3*r)&
           &−36.0*p(q,nv1,n1+2*t,m1+2*s,l1+2*r)&
           &+48.0*p(q,nv1,n1+t,m1+s,l1+r)&
           &−25.0*p(q,nv1,n1,m1,l1))*u
!
    end function bnd_0
    !
    ! 0−1 Boundary
    !_____________
    function bnd_1(p,q,nv1,ll1,mm1,nn1,r,s,t,u)                                            640
      REAL*8 :: bnd_1
      REAL*8,intent(in),dimension(:,:,:,:,:) :: p
      INTEGER, intent(in) :: nv1,q,ll1,mm1,nn1,r,s,t
      INTEGER :: l1, m1, n1
      real*8, intent(in) :: u
!
      l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
      bnd_1 = (−3.0*p(q,nv1,n1−t,m1−s,l1−r)&
           &−10.0*p(q,nv1,n1,m1,l1)&                                                       650
           &+18.0*p(q,nv1,n1+t,m1+s,l1+r)&
           &−6.0*p(q,nv1,n1+2*t,m1+2*s,l1+2*r)&
           &+p(q,nv1,n1+3*t,m1+3*s,l1+3*r))*u
```

```
!
      end function bnd_1
      !
      ! Middle
      !_______
      function mid(p,q,nv1,ll1,mm1,nn1,r,s,t,u)
        REAL*8 :: mid                                                    660
        REAL*8, intent(in), dimension(:,:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,q,ll1,mm1,nn1,r,s,t
        INTEGER :: l1, m1, n1
        real*8, intent(in) :: u
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        mid = (p(q,nv1,n1-2*t,m1-2*s,l1-2*r)&
             &-8.0*p(q,nv1,n1-t,m1-s,l1-r)&
             &+8.0*p(q,nv1,n1+t,m1+s,l1+r)&                              670
             &-p(q,nv1,n1+2*t,m1+2*s,l1+2*r))*u
!
      end function mid
      !
      ! N-1 Boundary
      !_____________
      function bnd_n1(p,q,nv1,ll1,mm1,nn1,r,s,t,u)
        REAL*8::bnd_n1
        REAL*8,intent(in),dimension(:,:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,q,ll1,mm1,nn1,r,s,t                   680
        INTEGER :: l1, m1, n1
        real*8, intent(in) :: u
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        bnd_n1=-(-3.0*p(q,nv1,n1+t,m1+s,l1+r)&
             &-10.0*p(q,nv1,n1,m1,l1)&
             &+18.0*p(q,nv1,n1-t,m1-s,l1-r)&
             &-6.0*p(q,nv1,n1-2*t,m1-2*s,l1-2*r)&
             &+p(q,nv1,n1-3*t,m1-3*s,l1-3*r))*u                          690
!
      end function bnd_n1
      !
      ! Nth Boundary
      !_____________
      function bnd_n(p,q,nv1,ll1,mm1,nn1,r,s,t,u)
        REAL*8 :: bnd_n
        REAL*8,intent(in),dimension(:,:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,q,ll1,mm1,nn1,r,s,t
        INTEGER :: l1, m1, n1                                            700
        real*8, intent(in) :: u
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        bnd_n = -(-3.0*p(q,nv1,n1-4*t,m1-4*s,l1-4*r)&
             &+16.0*p(q,nv1,n1-3*t,m1-3*s,l1-3*r)&
```

```
            &−36.0*p(q,nv1,n1−2*t,m1−2*s,l1−2*r)&
            &+48.0*p(q,nv1,n1−t,m1−s,l1−r)&
            &−25.0*p(q,nv1,n1,m1,l1))*u
!                                                                                    710
    end function bnd_n
!=====================================================================
!
  END SUBROUTINE f1f2prime
!
!=====================================================================
    SUBROUTINE prime_periodic(ivar,nvec)
!   This subroutine calculates first derivatives
!   The following format should be used when calling this subroutine
!   f_prime(Independent variable,Dependent variable,x3loc,x2loc,x1loc)               720
!   Independent variable: x1=1
!                         x2=2
!                         x3=3
!   Dependent variable: The variable for which the derivative is being calculated (T,Yi,u,v,w.....)
!   x3loc:              The x3 location
!   x2loc:              The x2 location
!   x1loc:              The x1 location
!
      USE vkvar
!                                                                                    730
      IMPLICIT NONE
!
      INTEGER :: a1,b1,c1,i,j,k,ii,jj,kk
      INTEGER, DIMENSION(:) :: gl(6),indi(2),indj(2),indk(2)
      INTEGER, INTENT(IN) :: ivar,nvec
      REAL*8 :: h, h1, test1
!
    ALLOCATE(f_prime_periodic(ivar:ivar,nvec:nvec,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
!
    ! Calculate dfdx                                                                 740
!================
    if (ivar==1) then
       h = sndx(ivar)
       h1 = 1.0/(12.0*h)
       a1 = 1
       b1 = 0
       c1 = 0
!
       do k = lock(1) − dom_bound(3,1), lock(2) − dom_bound(3,2)
          do j = locj(1) − dom_bound(2,1), locj(2) − dom_bound(2,2)                  750
             do i = loci(1)−dom_bound(1,1),loci(2)−dom_bound(1,2)
!
                kk = k − lock(1)
                jj = j − locj(1)
                ii = i − loci(1)
!
                f_prime_periodic(ivar,nvec,k,j,i) = &
                     & bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1,nx3,nx2,nx1)
!
```

```
            end do
          end do
        end do
      else
!
! Calculate dfdx2
!================*
        if (ivar==2) then
          h = sndx(ivar)
          h1 = 1.0/(12.0*h)
          a1 = 0
          b1 = 1
          c1 = 0
!
          do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
            do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)
              do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                kk = k - lock(1)
                jj = j - locj(1)
                ii = i - loci(1)
!
                f_prime_periodic(ivar,nvec,k,j,i) = &
                    & bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1,nx3,nx2,nx1)
!
              end do
            end do
          end do
!
        else
!
! Calculate dfdx3
!================
          if (ivar==3) then
            h = sndx(ivar)
            h1 = 1.0/(12.0*h)
            a1 = 0
            b1 = 0
            c1 = 1
!
            do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
              do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)
                do i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                  kk = k - lock(1)
                  jj = j - locj(1)
                  ii = i - loci(1)
!
                  f_prime_periodic(ivar,nvec,k,j,i) = &
                      & bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1,nx3,nx2,nx1)
!
                end do
              end do
```

```
            end do
!
          end if
        end if
      end if

    CONTAINS
!=============================================================                820
      ! Functions for calculating the above derivatives
!=============================================================
      !
      ! 0 th Boundary
      !---------------
      function bnd_0(p,nv1,ll1,mm1,nn1,r,s,t,u,nxx3,nxx2,nxx1)
        REAL*8::bnd_0
        INTEGER :: l1, m1,n1
        REAL*8,intent(in),dimension(:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t,nxx3,nxx2,nxx1                830
        real*8, intent(in) :: u
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        bnd_0=(-p(nv1,n1+2*t,m1+2*s,l1+2*r)&
             &+8.0*p(nv1,n1+t,m1+s,l1+r)&
             &-8.0*p(nv1,n1-1*t,&
             &m1-1*s,l1-1*r)&
             &+p(nv1,n1-2*t,&
             &m1-2*s,l1-2*r))*u                                                    840
      end function bnd_0
!
    END SUBROUTINE prime_periodic
!
!=====================================================================
!=====================================================================
      SUBROUTINE pprime_per(ivar,nvec)
!     This subroutine calculates first derivatives
!     The following format should be used when calling this subroutine
!     f_prime(Independent variable,Dependent variable,x3loc,x2loc,x1loc)          850
!     Independent variable: x1=1
!                           x2=2
!                           x3=3
!     Dependent variable: The variable for which the derivative is being calculated (T,Yi,u,v,w.....)
!     x3loc:              The x3 location
!     x2loc:              The x2 location
!     x1loc:              The x1 location
!
        USE vkvar
!                                                                                860
        IMPLICIT NONE
!
        INTEGER :: a1,b1,c1,i,j,k,ii,jj,kk
        INTEGER, INTENT(IN) :: ivar,nvec
        REAL*8 :: h, h1
```

```
      INTEGER, DIMENSION(:) :: gl(6),indi(2),indj(2),indk(2)
!
      ALLOCATE(f_pprime_per(ivar:ivar,nvec:nvec,lock(1):lock(2),locj(1):locj(2),loci(1):loci(2)))
!
    ! Calculate dfdx                                                                          870
!===============
      if (ivar==1) then
        h = sndx(ivar)
        h1 = 1.0/(12.0*h*h)
        a1 = 1
        b1 = 0
        c1 = 0
!
        do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
          do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)                          880
            do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
              kk = k - lock(1)
              jj = j - locj(1)
              ii = i - loci(1)
!
              f_pprime_per(ivar,nvec,k,j,i) = &
                  & bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1,nx3,nx2,nx1)
            end do
          end do                                                                             890
        end do
      else
!
! Calculate dfdx2
!===============
        if (ivar==2) then
          h = sndx(ivar)
          h1 = 1.0/(12.0*h*h)
          a1 = 0
          b1 = 1                                                                             900
          c1 = 0
          do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
            do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)
              do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
                kk = k - lock(1)
                jj = j - locj(1)
                ii = i - loci(1)
!
                f_pprime_per(ivar,nvec,k,j,i) = &                                            910
                    & bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1,nx3,nx2,nx1)
              end do
            end do
          end do
        else
!
! Calculate dfdx3
!===============
```

```
            if (ivar==3) then
               h = sndx(ivar)                                                    920
               h1 = 1.0/(12.0*h*h)
               a1 = 0
               b1 = 0
               c1 = 1
               do k = lock(1) - dom_bound(3,1), lock(2) - dom_bound(3,2)
                  do j = locj(1) - dom_bound(2,1), locj(2) - dom_bound(2,2)
                     do i = loci(1) - dom_bound(1,1), loci(2) - dom_bound(1,2)
!
                        kk = k - lock(1)
                        jj = j - locj(1)                                       930
                        ii = i - loci(1)
!
                        f_pprime_per(ivar,nvec,k,j,i) = &
                             & bnd_0(f,nvec,ii,jj,kk,a1,b1,c1,h1,nx3,nx2,nx1)
                     end do
                  end do
               end do
!
            endif
         end if                                                                940
      end if
!
      CONTAINS
!=============================================================
      ! Functions for calculating the above derivatives
!=============================================================
      !
      ! 0 th Boundary
      !---------------
      function bnd_0(p,nv1,ll1,mm1,nn1,r,s,t,u,nxx3,nxx2,nxx1)                 950
        REAL*8::bnd_0
        INTEGER :: l1, m1,n1
        REAL*8,intent(in),dimension(:,:,:,:) :: p
        INTEGER, intent(in) :: nv1,ll1,mm1,nn1,r,s,t,nxx3,nxx2,nxx1
        real*8, intent(in) :: u
!
        l1 = ll1+1;m1=mm1+1;n1=nn1+1
!
        bnd_0=(-p(nv1,n1+2*t,m1+2*s,l1+2*r)&
             &+16.0*p(nv1,n1+t,m1+s,l1+r)&                                     960
             &-30.*p(nv1,n1,m1,l1)&
             &+16.0*p(nv1,n1-t,m1-s,l1-r)&
             &-p(nv1,n1-2*t,m1-2*s,l1-2*r))*u
      end function bnd_0
!
   END SUBROUTINE pprime_per
!=============================================================*
!=============================================================*
      SUBROUTINE init_runge
!                                                                              970
      USE vkvar
```

```
!
      IMPLICIT NONE
!
      allocate(ack(ikutta),bck(ikutta),cck(ikutta))
!
      if (ikutta==1) then
!
!       1st order backward scheme
!========================                                          980
!
         nstage = 1
!
         ack(1) = 0.0d0
!
         bck(1) = 1.0d0
!
         cck(1) = 1.0d0
!
      end if                                                      990
!
      if (ikutta==3) then
!
!       (3,3) scheme of Williamson
!========================
!
         nstage = 3
!
         ack(1) = 0.0d0
         ack(2) = -5.0d0/9.0d0                                    1000
         ack(3) = -153.0d0/128.0d0
!
         bck(1) = 1.0d0/3.0d0
         bck(2) = 15.0d0/16.0d0
         bck(3) = 8.0d0/15.0d0
!
         cck(1) = 0.0d0
         cck(2) = 1.0d0/3.0d0
         cck(3) = 3.0d0/4.0d0
!                                                                 1010
      end if
!
      if (ikutta==5) then
!
!       (5,4) scheme of Carpenter
!========================
!
         nstage = 5
!
         ack(1) = 0.0d0                                           1020
         ack(2) = -0.417890474500d0
         ack(3) = -1.192151694643d0
         ack(4) = -1.697784692471d0
         ack(5) = -1.514183444257d0
```

```
!
        bck(1) = 0.1496590219993d0
        bck(2) = 0.3792103129999d0
        bck(3) = 0.8229550293869d0
        bck(4) = 0.6994504559488d0
        bck(5) = 0.1530572479681d0                                          1030
!
        cck(1) = 0.0d0
        cck(2) = 0.1496590219993d0
        cck(3) = 0.3704009573644d0
        cck(4) = 0.6222557631345d0
        cck(5) = 0.9582821306748d0
!
      end if
!
      END SUBROUTINE init_runge                                             1040
!
!
!=======================================================================
!=======================================================================
!
END MODULE der_ord_4
```

Integration

```
MODULE vk_cf
  CONTAINS
!
   SUBROUTINE rk54(tin,tout)
!
     USE vkvar
     USE vkpar
!
     IMPLICIT NONE
!                                                                         10
!    Runge−Kutta
!
     INTEGER :: i, j, k, n, ns
     REAL*8, INTENT(IN) :: tin, tout
     REAL*8 :: time_n, h
     INTEGER, dimension(2) :: indi,indj,indk
!
     h = tout−tin
!
     DO ns=1,ikutta                                                     20
!
        time_n = tin + cck(ns)*h
!
        CALL per_corr
        CALL vk_buffer_comm(1,nvar)
        CALL rhs(time_n)
!
        DO n=1,nvar
```

```
          DO k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
            DO j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
              DO i = loci(1)-dom_bound(1,1),loci(2)-dom_bound(1,2)
!
                rkj(n,k,j,i) =   ack(ns)*rkj(n,k,j,i) + h*fprime(n,k,j,i)
                f(n,k,j,i)   = f(n,k,j,i) + bck(ns)*rkj(n,k,j,i)
!
                if (f(n,k,j,i).lt.1.d-20) f(n,k,j,i) = 1.d-20
!
              END DO
            END DO
          END DO
        END DO
!
        CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
!
!       Set boundary conditions
!====================
!
       CALL bc(n)
!
!       Check if temperature or species values go below 0 or above 100
!       and set the floor at 0.0 and the ceiling at 100.0
!=================================================
!
     END DO
!
        CALL MPI_BARRIER(MPI_COMM_WORLD,ierr)
!
        t = tout
!
      END SUBROUTINE rk54
!
!=================================================================
!=================================================================
!
      SUBROUTINE rhs(time_curr)
!
        USE vkvar
        USE der_ord_4
!
        IMPLICIT NONE
!
        INTEGER :: i, j, k, l, n,zpos
        INTEGER, dimension(2) :: indi,indj,indk
        REAL*8, INTENT(IN) :: time_curr
!
!       Compute and assign respective derivative values
!=====================================
!
        indi(1) = loci(1) - dom_bound(1,1)
        indi(2) = loci(2) - dom_bound(1,2)
```

```
        indj(1) = locj(1) - dom_bound(2,1)
        indj(2) = locj(2) - dom_bound(2,2)
        indk(1) = lock(1) - dom_bound(3,1)
        indk(2) = lock(2) - dom_bound(3,2)
!
        DO n = 1,nvar
           DO l = 1,3
              IF (periodic(l)==0) then
!                                                                          90
                 CALL prime(l,n)
                 CALL pprime(l,n)
!
                 DO k = indk(1),indk(2)
                    DO j = indj(1),indj(2)
                       DO i = indi(1),indi(2)
!
                          cc(n,sndim(l,2),k,j,i) = &
                               &f_pprime(l,n,k,j,i)*(cc1(1,l,k,j,i)**2)&
                               &+f_prime(l,n,k,j,i)*cc1(2,l,k,j,i)                 100
                          cc(n+nvar,sndim(l,2),k,j,i) = f_prime(l,n,k,j,i)*cc1(1,l,k,j,i)
!
                       END DO
                    END DO
                 END DO
!
                 DEALLOCATE(f_prime,f_pprime)
!
              else                                                           110
!
                 call prime_periodic(l,n)
                 call pprime_per(l,n)
!
                 DO k = indk(1),indk(2)
                    DO j = indj(1),indj(2)
                       DO i = indi(1),indi(2)
!
                          cc(n,sndim(l,2),k,j,i) = f_pprime_per(l,n,k,j,i)
                          cc(n+nvar,sndim(l,2),k,j,i) = f_prime_periodic(l,n,k,j,i)  120
!
                       END DO
                    END DO
                 END DO
!
                 DEALLOCATE(f_prime_periodic,f_pprime_per)
!
              END IF
           END DO
        END DO                                                             130
!
!       Set domain boundary limits for the r direction since the
!       singularities arise at boundary pts
!=========================================
```

```
!
!
      if (sndim(1,2) == 1.and.dom_bound(1,1) == 0) indi(1) = loci(1)+1
      if (sndim(1,2) == 1.and.dom_bound(1,2) == 0) indi(2) = loci(2)-1
      if (sndim(2,2) == 1.and.dom_bound(2,1) == 0) indj(1) = locj(1)+1
      if (sndim(2,2) == 1.and.dom_bound(2,2) == 0) indj(2) = locj(2)-1                140
      if (sndim(3,2) == 1.and.dom_bound(3,1) == 0) indk(1) = lock(1)+1
      if (sndim(3,2) == 1.and.dom_bound(3,2) == 0) indk(2) = lock(2)-1
!
!     Compute the d/dt for all the variables
!=============================
!
      DO k = indk(1),indk(2)
        DO j = indj(1),indj(2)
          DO i = indi(1),indi(2)
!                                                                                 150
            if (sndim(1,2) == 3) zpos = i
            if (sndim(2,2) == 3) zpos = j
            if (sndim(3,2) == 3) zpos = k
!
            Q1(k,j,i) = Damk * (eps**3) / ((tstar*zstar)**2) &
                &* f(2,k,j,i)*f(3,k,j,i) * &
                &exp (eps*(dfloat(1)/tstar - 1.d0 / f(1,k,j,i)) )
!
            fprime(1,k,j,i) =(1.d0/Pr)*(cc(1,1,k,j,i)+&
                &cc(1,2,k,j,i)/(f(nvar+1,k,j,i)**2)&                             160
                &+cc(1,3,k,j,i))+&
                &(1.d0/(f(nvar+1,k,j,i)*Pr)-&
                &f(nvar+1,k,j,i)*VEL(1,zpos))*cc(4,1,k,j,i)&
                &-VEL(2,zpos)*cc(4,2,k,j,i)&
                &-VEL(3,zpos)*cc(4,3,k,j,i)&
                &+beta*Q1(k,j,i)
!
            fprime(2,k,j,i) = (1.d0/Pr/Le1)*(cc(2,1,k,j,i)+&
                &cc(2,2,k,j,i)/(f(nvar+1,k,j,i)**2)&
                &+cc(2,3,k,j,i))+&                                               170
                &(1.d0/(f(nvar+1,k,j,i)*Le1*Pr)-&
                &f(nvar+1,k,j,i)*VEL(1,zpos))*cc(5,1,k,j,i)&
                &-VEL(2,zpos)*cc(5,2,k,j,i)&
                &-VEL(3,zpos)*cc(5,3,k,j,i)&
                &-alpha1*Q1(k,j,i)
!
            fprime(3,k,j,i) = (1.d0/Pr/Le2)*(cc(3,1,k,j,i)+&
                &cc(3,2,k,j,i)/(f(nvar+1,k,j,i)**2)&
                &+cc(3,3,k,j,i))+&
                &(1.d0/(f(nvar+1,k,j,i)*Le2*Pr)-&                                180
                &f(nvar+1,k,j,i)*VEL(1,zpos))*cc(6,1,k,j,i)&
                &-VEL(2,zpos)*cc(6,2,k,j,i)&
                &-VEL(3,zpos)*cc(6,3,k,j,i)&
                &-alpha2*Q1(k,j,i)
!
!            print 11, fprime(1,k,j,i), fprime(2,k,j,i), fprime(3,k,j,i)
!
```

```
            END DO
          END DO
        END DO                                                    190
!
11      format(2x,3f12.8)
!
      END SUBROUTINE rhs
!
!================================== ====================
!=========================================================
!
      subroutine bc(invar)
!                                                                 200
        USE vkvar
!
        IMPLICIT NONE
!
        INTEGER :: i,j,k,n,a1,b1,c1,tsize,stat,ii,zpos,ffsize,r
        INTEGER, INTENT(IN) :: invar
        INTEGER, dimension(2) :: indi, indj, indk
        REAL*8,dimension(:),allocatable :: trans
        REAL*8,dimension(:,:,:),allocatable :: ff
        REAL*8 :: h1, h2,y0                                       210
!
!       B.C at z = 0:
!================
!
        indi(1) = loci(1) - dom_bound(1,1)
        indi(2) = loci(2) - dom_bound(1,2)
        indj(1) = locj(1) - dom_bound(2,1)
        indj(2) = locj(2) - dom_bound(2,2)
        indk(1) = lock(1) - dom_bound(3,1)
        indk(2) = lock(2) - dom_bound(3,2)                        220
!
        a1 = 0; b1 = 0; c1 = 0;
        if (sndim(1,2) == 3) a1 = 1
        if (sndim(2,2) == 3) b1 = 1
        if (sndim(3,2) == 3) c1 = 1
        if (sndim(1,2) == 3) ii = 1
        if (sndim(2,2) == 3) ii = 2
        if (sndim(3,2) == 3) ii = 3
!
        y0 = 0.22d0                                               230
!
        h1 = 1.d0/(12.d0*dx3)&
             &*cc1(1,ii,indk(1),indj(1),indi(1))
!
        if (sndim(1,2) == 3.and.dom_bound(1,1) == 0) then
!
          DO k = lock(1)-dom_bound(3,1),lock(2)-dom_bound(3,2)
            DO j = locj(1)-dom_bound(2,1),locj(2)-dom_bound(2,2)
!
              f(1,k,j,loci(1)) = T0                               240
```

```
                f(2,k,j,loci(1)) = (h1*(−3.d0*f(2,k,j,loci(1)+4)+&
                    &16.d0*f(2,k,j,loci(1)+3)−36.d0*f(2,k,j,loci(1)+2)&
                    &+48.d0*f(2,k,j,loci(1)+1))+Le1*Pe)/(Le1*Pe+25.d0*h1)
!
                f(3,k,j,loci(1)) = (h1*(−3.d0*f(3,k,j,loci(1)+4)+&
                    &16.d0*f(3,k,j,loci(1)+3)−36.d0*f(3,k,j,loci(1)+2)&
                    &+48.d0*f(3,k,j,loci(1)+1)))/(Le2*Pe+25.d0*h1)
!
            END DO
          END DO                                                          250
        END if
!
        if (sndim(2,2) == 3.and.dom_bound(2,1) == 0) then
!
          DO k = indk(1),indk(2)
            DO i = indi(1),indi(2)
!
                f(1,k,locj(1),i) = T0
                f(2,k,locj(1),i) = (h1*(−3.d0*f(2,k,locj(1)+4,i)+&
                    &16.d0*f(2,k,locj(1)+3,i)−36.d0*f(2,k,locj(1)+2,i)&          260
                    &+48.d0*f(2,k,locj(1)+1,i))+Le1*Pe)/(Le1*Pe+25.d0*h1)
!
                f(3,k,locj(1),i) = (h1*(−3.d0*f(3,k,locj(1)+4,i)+&
                    &16.d0*f(3,k,locj(1)+3,i)−36.d0*f(3,k,locj(1)+2,i)&
                    &+48.d0*f(3,k,locj(1)+1,i)))/(Le2*Pe+25.d0*h1)
!
            END DO
          END DO
        END if
!                                                                         270
        if (sndim(3,2) == 3.and.dom_bound(3,1) == 0) then
!
          DO j = indj(1),indj(2)
            DO i = indi(1),indi(2)
!
                f(1,lock(1),j,i) = T0
                f(2,lock(1),j,i) = (h1*(−3.d0*f(2,lock(1)+4,j,i)+&
                    &16.d0*f(2,lock(1)+3,j,i)−36.d0*f(2,lock(1)+2,j,i)&
                    &+48.d0*f(2,lock(1)+1,j,i))+Le1*Pe)/(Le1*Pe+25.d0*h1)
!                                                                         280
                f(3,lock(1),j,i) = (h1*(−3.d0*f(3,lock(1)+4,j,i)+&
                    &16.d0*f(3,lock(1)+3,j,i)−36.d0*f(3,lock(1)+2,j,i)&
                    &+48.d0*f(3,lock(1)+1,j,i)))/(Le2*Pe+25.d0*h1)
!
            END DO
          END DO
        END if
!
!       B.C at z = zinf
!==================                                                       290
!
        if (sndim(1,2)==3.and.dom_bound(1,2) == 0) then
!
```

```
          DO k = lock(1)−dom_bound(3,1),lock(2)−dom_bound(3,2)
            DO j = locj(1)−dom_bound(2,1),locj(2)−dom_bound(2,2)
!
!               f(1,k,j,loci(2)) = f(1,k,j,loci(2)−1)
!               f(2,k,j,loci(2)) = f(2,k,j,loci(2)−1)
!               f(3,k,j,loci(2)) = f(3,k,j,loci(2)−1)
                f(1,k,j,loci(2)) = TINF                                    300
                f(2,k,j,loci(2)) = 0.0d0
                f(3,k,j,loci(2)) = 1.0d0/eq1
!
            END DO
          END DO
        END if
!
        if (sndim(2,2)==3.and.dom_bound(2,2) == 0) then
!
          DO k = indk(1),indk(2)                                           310
            DO i = indi(1),indi(2)
!
!               f(1,k,locj(2),i) = f(1,k,locj(2)−1,i)
!               f(2,k,locj(2),i) = f(2,k,locj(2)−1,i)
!               f(3,k,locj(2),i) = f(3,k,locj(2)−1,i)
              f(1,k,locj(2),i) = TINF
              f(2,k,locj(2),i) = 0.0d0
              f(3,k,locj(2),i) = 1.0d0/eq1
!
            END DO                                                         320
          END DO
        end if
!
        if (sndim(3,2)==3.and.dom_bound(3,2) == 0) then
!
          DO j = indj(1),indj(2)
            DO i = indi(1),indi(2)
!
!               f(1,lock(2),j,i) = f(1,lock(2)−1,j,i)
!               f(2,lock(2),j,i) = f(2,lock(2)−1,j,i)                    330
!               f(3,lock(2),j,i) = f(3,lock(2)−1,j,i)
              f(1,lock(2),j,i) = TINF
              f(2,lock(2),j,i) = 0.0d0
              f(3,lock(2),j,i) = 1.0d0/eq1
!
            END DO
          END DO
        end if
!
!   B.C at r = 0                                                         340
!==============
        indi(1) = loci(1) − dom_bound(1,1)
        indi(2) = loci(2) − dom_bound(1,2)
        indj(1) = locj(1) − dom_bound(2,1)
        indj(2) = locj(2) − dom_bound(2,2)
        indk(1) = lock(1) − dom_bound(3,1)
```

```
      indk(2) = lock(2) - dom_bound(3,2)
!
      ffsize=0
      if (sndim(1,2)==1.and.dom_bound(1,1) == 0) then                                    350
!
         allocate(ff(3,indk(1):indk(2),indj(1):indj(2)))
         DO n = 1,nvar
            DO k = indk(1),indk(2)
               DO j = indj(1),indj(2)
                  ff(n,k,j)=0.d0
!
                  DO r = -2,2,1
                     if (sndim(2,2)==2) ff(n,k,j) = ff(n,k,j)+f(n,k,j+r,loci(1)+1)+f(n,k,j+r,loci(1))
                     if (sndim(3,2)==2) ff(n,k,j) = ff(n,k,j)+f(n,k+r,j,loci(1)+1)+f(n,k+r,j,loci(1))   360
                  END DO
                  ff(n,k,j) = ff(n,k,j)/dfloat(10)
                  f(n,k,j,loci(1)) = ff(n,k,j)
!
               END DO
            END DO
         END DO
         deallocate(ff)
!
      END if                                                                             370
!
      if (sndim(2,2)==1.and.dom_bound(2,1) == 0) then
!
         allocate(ff(3,indk(1):indk(2),indi(1):indi(2)))
!
         DO n = 1,nvar
            DO k = indk(1),indk(2)
               DO i = indi(1),indi(2)
                  ff(n,k,i)=0.d0
!                                                                                        380
                  DO r = -2,2,1
                     if (sndim(1,2)==2) ff(n,k,i) = ff(n,k,i)+f(n,k,locj(1)+1,i+r)+f(n,k,locj(1),i+r)
                     if (sndim(3,2)==2) ff(n,k,i) = ff(n,k,i)+f(n,k+r,locj(1)+1,i)+f(n,k+r,locj(1),i)
                  END DO
                  ff(n,k,i) = ff(n,k,i)/dfloat(10)
                  f(n,k,locj(1),i) = ff(n,k,i)
!
               END DO
            END DO
         END DO                                                                          390
         deallocate(ff)
      END if
!
      if (sndim(3,2)==1.and.dom_bound(3,1) == 0) then
!
         allocate(ff(3,indj(1):indj(2),indi(1):indi(2)))
!
         DO n = 1,nvar
            DO j = indj(1),indj(2)
```

```
            DO i = indi(1),indi(2)                                          400
               ff(n,j,i)=0.d0
!
                DO r = −2,2,1
                  if (sndim(1,2)==2) ff(n,j,i) = ff(n,j,i)+f(n,lock(1)+1,j,i+r)+f(n,lock(1),j,i+r)
                  if (sndim(2,2)==2) ff(n,j,i) = ff(n,j,i)+f(n,lock(1)+1,j+r,i)+f(n,lock(1),j+r,i)
               END DO
               ff(n,j,i) = ff(n,j,i)/dfloat(10)
               f(n,lock(1),j,i) = ff(n,j,i)

!                                                                           410
            END DO
          END DO
        END DO
        deallocate(ff)
    END if
!
!   B.C. at r = x2max
!=================
!
    if (sndim(1,2) == 1) ii = 1                                             420
    if (sndim(2,2) == 1) ii = 2
    if (sndim(3,2) == 1) ii = 3
!
    h1 = 1.d0/(12.d0*dx1)&
           &*cc1(1,ii,indk(2),indj(2),indi(2))
!
     if (sndim(1,2) == 1.and.dom_bound(1,2) == 0) then
!       DO n = 1,nvar
          DO k = indk(1),indk(2)
            DO j = indj(1),indj(2)                                          430
!
               if (sndim(2,2) == 3) zpos = j
               if (sndim(3,2) == 3) zpos = k
!
               f(1,k,j,loci(2)) = f(1,k,j,loci(2)−1)
               f(2,k,j,loci(2)) = ((−3.d0*f(2,k,j,loci(2)−4)+&
                    &16.d0*f(2,k,j,loci(2)−3)−36.d0*f(2,k,j,loci(2)−2)&
                    &+48.d0*f(2,k,j,loci(2)−1)))/(25.d0)
               f(3,k,j,loci(2)) = ((−3.d0*f(3,k,j,loci(2)−4)+&
                    &16.d0*f(3,k,j,loci(2)−3)−36.d0*f(3,k,j,loci(2)−2)&      440
                    &+48.d0*f(3,k,j,loci(2)−1)))/(25.d0)
!
            END DO
          END DO
!       END DO
    END if
!
    if (sndim(2,2) == 1.and.dom_bound(2,2) == 0) then
!       DO n = 1,nvar
          DO k = indk(1),indk(2)                                            450
            DO i = indi(1),indi(2)
!
```

```
                if (sndim(1,2) == 3) zpos = i
                if (sndim(3,2) == 3) zpos = k
!
                f(1,k,locj(2),i) = f(1,k,locj(2)−1,i)
                f(2,k,locj(2),i) = ((−3.d0*f(2,k,locj(2)−4,i)+&
                    &16.d0*f(2,k,locj(2)−3,i)−36.d0*f(2,k,locj(2)−2,i)&
                    &+48.d0*f(2,k,locj(2)−1,i)))/(25.d0)
                f(3,k,locj(2),i) = ((−3.d0*f(3,k,locj(2)−4,i)+&                    460
                    &16.d0*f(3,k,locj(2)−3,i)−36.d0*f(3,k,locj(2)−2,i)&
                    &+48.d0*f(3,k,locj(2)−1,i)))/(25.d0)
!
            END DO
          END DO
!       END DO
    END if
!
    if (sndim(3,2) == 1.and.dom_bound(3,2) == 0) then
!       DO n = 1,nvar                                                          470
          DO j = indj(1),indj(2)
            DO i = indi(1),indi(2)
!
                if (sndim(1,2) == 3) zpos = i
                if (sndim(2,2) == 3) zpos = j
!
                f(1,lock(2),j,i) = f(1,lock(2)−1,j,i)
                f(2,lock(2),j,i) = ((−3.d0*f(2,lock(2)−4,j,i)+&
                    &16.d0*f(2,lock(2)−3,j,i)−36.d0*f(2,lock(2)−2,j,i)&
                    &+48.d0*f(2,lock(2)−1,j,i)))/(25.d0)                        480
                f(3,lock(2),j,i) = ((−3.d0*f(3,lock(2)−4,j,i)+&
                    &16.d0*f(3,lock(2)−3,j,i)−36.d0*f(3,lock(2)−2,j,i)&
                    &+48.d0*f(3,lock(2)−1,j,i)))/(25.d0)
!
            END DO
          END DO
!       END DO
    END if
!
!                                                                              490
!
     END SUBROUTINE bc
!
!=====================================================
!=====================================================
!
     Subroutine per_corr
!
       USE vkvar
!                                                                              500
       IMPLICIT NONE
!
       INTEGER :: i,j,k,n
!
       if (periodic(1).ne.0.and.dims(1)==1) then
```

```
          do n = 1,nvar
            do k = lock(1) − dom_bound(3,1), lock(2) − dom_bound(3,2)
              do j = locj(1) − dom_bound(2,1), locj(2) − dom_bound(2,2)
!
                f(n,k,j,loci(1)) = f(n,k,j,loci(2)−ghostlayer−2)                    510
                f(n,k,j,loci(1)+1) = f(n,k,j,loci(2)−ghostlayer−1)
                f(n,k,j,loci(2)) = f(n,k,j,loci(1)+ghostlayer+2)
                f(n,k,j,loci(2)−1) = f(n,k,j,loci(1)+ghostlayer+1)
!
              end do
            end do
          end do
!
        end if
!                                                                                  520
        if (periodic(2).ne.0.and.dims(2)==1) then
          do n = 1,nvar
            do k = lock(1) − dom_bound(3,1), lock(2) − dom_bound(3,2)
              do i = loci(1) − dom_bound(1,1), loci(2) − dom_bound(1,2)
!
                f(n,k,locj(1),i) = f(n,k,locj(2)−ghostlayer−2,i)
                f(n,k,locj(1)+1,i) = f(n,k,locj(2)−ghostlayer−1,i)
                f(n,k,locj(2),i) = f(n,k,locj(1)+ghostlayer+2,i)
                f(n,k,locj(2)−1,i) = f(n,k,locj(1)+ghostlayer+1,i)
!                                                                                  530
              end do
            end do
          end do
!
        end if
!
       if (periodic(3).ne.0.and.dims(3)==1) then
         do n = 1,nvar
           do j = locj(1) − dom_bound(2,1), locj(2) − dom_bound(2,2)
             do i = loci(1) − dom_bound(1,1), loci(2) − dom_bound(1,2)              540
!
               f(n,lock(1),j,i) = f(n,lock(2)−ghostlayer−2,j,i)
               f(n,lock(1)+1,j,i) = f(n,lock(2)−ghostlayer−1,j,i)
               f(n,lock(2),j,i) = f(n,lock(1)+ghostlayer+2,j,i)
               f(n,lock(2)−1,j,i) = f(n,lock(1)+ghostlayer+1,j,i)
!
             end do
           end do
         end do
       end if                                                                      550
!
    End Subroutine per_corr
!
!=====================================================
!=====================================================
!=================================================
!
    Subroutine perturb
```

```
!
      USE vkvar                                                                    560
      USE vkpar
      USE vk_in
!
      IMPLICIT NONE
!
      INTEGER :: i,j,k,n
      INTEGER,dimension(2) :: indi,indj,indk
      REAL*8 :: sp
      sp = 1.d0
!                                                                                  570
      if (id==0) print *, 'Perturbing in Theta'
!
      do i = 1,2
         indi(i) = loci(i)-dom_bound(1,i)
         indj(i) = locj(i)-dom_bound(2,i)
         indk(i) = lock(i)-dom_bound(3,i)
      end do
!
      if (wnum.gt.0.d0) sp = 0.d0
      do n = 1,nvar                                                              580
         do k = indk(1),indk(2)
            do j = indj(1),indj(2)
               do i = indi(1),indi(2)
                  f(n,k,j,i) = f(n,k,j,i)*(1.d0+(1.d0-sp)*pert*(cos(wnum*f(5,k,j,i)))&
                       &+sp*pert*(0.01*cos(f(5,k,j,i))+cos(acos(-1.)-f(4,k,j,i)/x1max*acos(-1.))))
                  if (f(n,k,j,i).lt.1.d-20) f(n,k,j,i) = 1.d-20
               end do
            end do
         end do
      end do                                                                     590
      call track_error(t)
      CALL vk_communication(1,fsize)
      CALL output(t,1000)
!
    end Subroutine perturb
!
!=================================================
!
    END MODULE vk_cf
                                                                                 600
```

Sample Input

```
nvar    nstep   dt
3       10000   0.0002d0
ikutta  iord
5       2
H0      b       eq1
0.1d0   1.      2.d0
rmin    thetmin zmin
```

```
0.0001  0.0     0.0
rmax    thetmax zmax
40.     0.0     25.
nr      ntheta  nz
64      64      64
St1     St2     St3
1.0     1.0     1.04
Pr      Le1     Le2     Damk        a1      a2      beta    Ze      bcval
1.0     1.0     1.0     1.90d0      .25d0   1.0     25.d0   40.     1.0
T0      TINF
1.0     1.0
Irestart        thetapert       wnum    err ouput freq  Contour_out_freq
3               0.01            0.0     100             25000
nproc1  nproc2  nproc3
0       0       0
tstar   zstar
7.43    1.54
```

Notes:

thetamax: **if** 0, direction is taken to be periodic

Irestart:
- 0 − Initial conditions given in initial_conditions(nvar) in func_def.f90
- 1 − restart
- 2 − restart with time=0
- 3 − time=0, Initial conditions=temp_colsys.dat

therapert: Magnitude of perturbation (usually 0.01)

wnum:
- 0.0 − cos(r)
- 1.0 − cos(theta)
- 2.0 − cos(2theta)
- 3.0 − cos(3theta)
- .
- .
- .

err_output_freq: Interval at which to print to screen

Contour_out_freq: Interval at which to write output

nproc: Number of processors assigned to the respective direction. If all are set to 0 then the optimal combination is assigned.

References

[1] V. Nayagam and F. A. Williams. Rotating Spiral Edges in Von Karman Swirling Flows, Physical Review Letters **84**, 3 (2000).

[2] V. Nayagam and F. A. Williams. Pattern Formation in Diffusion Flames Embedded in Von Karman Swirling Flows, NASA/CP 2001-21082 (2001).

[3] V. Nayagam and F. A. Williams. Pattern Formation in Diffusion flames Embedded in Von Karman Swirling Flows, NASA/CR 2006-214057 (2006).

[4] M. H. Carpenter and C. A. Kennedy. Fourth-order 2N Storage Runge-Kutta Schemes, NASA Technical Memorandum 109112 (1994).

[5] R. Vance, M. Miklavic, and I. S. Wichman. On the stability of one-dimensional diffusion flames established between plane, parallel, porous walls, Combustion Theory and Modelling **5,**, 147 (2001).

[6] J. D. Buckmaster and G. S. S. Ludford, *Theory of Laminar Flames* (Cambridge University Press, 1982).

[7] P. J. Zandbergen and D. Dijkstra. Von Karman Swirling Flows, Annual Review of Fluid Mechanics **19**, 456 (1987).

[8] V. Nayagam and F. A. Williams. Diffusion Flame Extinction for a Spining Fuel Disk in an Oxidizer Counterflow, Proceedings of the Combustion Institute **28**, 2875 (2000).

[9] M. L. Shay and P. D. Ronney. Nonpremixed Edge Flames in Spatially Varying Straining Flows, Combustion and Flame **112**, 117 (1998).

[10] J. D. Buckmaster. Edge-Flames and their Stability, Combustion Science and Technology **115, 1-3**, 41 (1996).

[11] J. D. Buckmaster. Edge-flames, Progress in Energy and Combustion Science **28**, 435 (2002).

[12] G. L. Pellet, K. M. Isaac, W. M. J. Humphreys, L. R. Gartrell, W. L. Roberts, C. L. Dancey, and G. B. Northam. Velocity and Thermal Structure, and Strain-Induced Extinction of 14 to 100% Hydrogen-Air Counterflow Diffusion Flames, Combustion and Flame **112**, 575 (2002).

[13] C. E. Frouzakis, A. G. Tomboulides, J. Lee, and K. Boulouchos. From Diffusion to Premixed Flames in an H2/Air Opposed-Jet Burner: The Role of Edge Flames, Combustion and Flame **130**, 171 (2002).

[14] Z. Lu and S. Ghoshal. Flame Holes and Flame Disks on the Surface of a Diffusion Flame, Journal of Fluid Mechanics **513**, 287 (2004).

[15] J. D. Buckmaster, A. Nachman, and S. Taliaferro. The Fast-Time Instability of Diffusion Flames, Physica D: Nonlinear Phenomena **9, 3**, 408 (1983).

[16] J. S. Kim. Linear Analysis of Diffusional-Thermal Instability in Diffusion Flames with Lewis Numbers Close to Unity, Combustion Theory and Modelling **1**, 13 (1997).

[17] S. Cheatham and M. Matalon. A General Asymptotic Theory of Diffusion Flames with Application to Cellular Instabilities, Journal of Fluid Mechanics **414**, 105 (2000).

[18] S. Kukuck and M. Matalon. The Onset of Oscillations in Diffusion Flames, Combustion Theory and Modelling **5**, 217 (2001).

[19] J. D. Buckmaster and T. L. Jackson. Holes in Flames, Flame Isolas, and Flame Edges, Proceedings of the Combustion Institute **28**, 1957 (2000).

[20] M. Short and Y. Liu. Edge-flame Structure and Oscillations for Unit Lewis Numbers in a Non-premixed Counterflow, Combustion Theory and Modelling **8**, 425 (2004).

[21] B. S. Margolis. Bifurcation Phenomena in Burner-Stabilized Premixd Flames, Combustion Science and Technology **22**, 143 (1979).

[22] J. D. Buckmaster. Stability of the Porous Plug Burner Flame, SIAM Journal on Applied Matematics **43, 6**, 1335 (1983).

[23] R. Chen, B. G. Mitchell, and P. D. Ronney. Diffusive-Thermal Instability and Flame Extinction in Nonpremixed Combustion, Twenty-fourth Eighteenth Symposium (International) on Combustion pp. 213–221 (1992).

[24] S. P. Burke and T. E. W. Schumann. Diffusion Flames, 76th Meeting of the American Chemical Society **20, 10**, 998 (1928).

[25] M. Matalon. The Effect of Thermal Expansion on Flame Dynamics, 5th US Combustion Meeting (2007).

[26] M. Matalon. Intrinsic Flame Instabilities in Premixed and Nonpremixed Combustion, Annual Review of Fluid Mechanics (2007).

[27] M. Matalon and V. Kurdyumov. Effect of Thermal Expansion on Edge-Flames, 46th AIAA Aerospace Sciences Meeting and Exhibit (2008).

[28] M. Smooke, C. McEnally, L. Pfefferle, R. Hall, and M. Colket. Computational and Experimental Study of Soot Formation in a Coflow, Laminar Diffusion Flame, Combustion and Flame **117, Issues 1-2**, 117 (1999).

[29] G. Amantini and A. G. Jonathan H. Frank, Mitchell D. Smooke. Computational and Experimental Study of Standing Methane Edge Flames in the Two-Dimensional Axisymmetric Counterflow Geometry, Combustion and Flame **147**, 133 (2006).

[30] U. Asher, J. Christiansen, and R. D. Russel. Collocation Software for Boundary Value ODEs, ACM Transactions on Math Software **7**, 223 (1981).

[31] J. H. Williamson. Low Storage Runge-Kutta Schemes, Journal of Computational Physics (1980).

[32] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP* (McGraw-Hill, 2004).

[33] A. Gray, *Modern Differential Geometry of Curves and Surfaces with Mathematica* (CRC Press, Boca Raton, FL, 1997).

[34] R. W. Bilger. Turbulent Diffusion Flames, Annual Review of Fluid Mechanics **2**, 101 (1989).

[35] R. W. Bilger. The Structure of Diffusion Flames, Combustion Science and Technology **13**, 155 (1976).

[36] K. K. Kuo, *Principles of Combustion* (John Wiley and Sons, 1986).

[37] N. Peters, *Turbulent Combustion* (Cambridge University Press, 2000).

[38] V. Favier and L. Vervisch. Edge Flames and Partially Premixed Combustion in Diffusion Flame Quenching, Combustion and Flame **125**, 788 (2001).

[39] N. Chakraborty and E. Mastorakos. Numerical Investigation of Edge Flame Propagation Characteristics in Turbulent Mixing Layers, Physics of Fluids **18**, 105103 (2006).

Curriculum Vitae

Kishwar Hossain

550 W Stocker St
Glendale, CA 91202
773-787-672

Education

5/2003 - Present
University of Illinois at Urbana Champaign, Department of Aerospace Engineering, Urbana, IL
§§ Doctorate
§§ GPA: 3.7, Minor: Computational Science and Engineering

5/2000 - 5/2003
University of Illinois - Urbana Champaign, Department of Aerospace Engineering, Urbana, IL
§§ Master's Degree
§§ GPA: 3.6

8/1997 - 5/2000
Lafayette College, Easton, PA
§§ Bachelor's Degree
§§ GPA: 3.5, Minor: Mathematics
§§ Graduated with Honors, Excel Scholar

Experience

Research Assistant
10/2007 – Present
Computational Science and Engineering, UIUC, Urbana, IL
§§ Member of the Combustion and Energetic Materials Group.
§§ Conducting a study on burn rates for heterogeneous solid-propellant packs using a 3-step kinetic model to evaluate the robustness of the model.

Researcher
5/2004 – Present
UIUC, Department of Aerospace Engineering, Urbana, IL
§§ Development of numerical algorithms to study flame dynamics.
§§ Conducting a study on the stability, and extinction characteristics of diffusion flames supported by a spinning methane burner. It is found that non-uniform flames, namely flame holes and spirals appear at near extinction conditions. These flames are simulated using a

three-dimensional parallel combustion code in cylindrical coordinates.
§§ Developed FORTRAN based computational code for solving three-dimensional combustion problems.
§§ Adapted the above code to run on a cluster of parallel computers using the MPI protocol.

Analyst – Graduate Assistant
8/2003 - 8/2006
Champaign Simulation Center, Caterpillar, Champaign, IL
§§ Structural analysis of earth moving vehicles using finite element methods. Generated finite element meshes using software such as IDEAS and Hypermesh and conducted analysis using NASTRAN and ABAQUS.
§§ Conducted modal analysis to identify critical frequencies in newly developed machines.

Analyst - CAT PRACTICUM Internship
5/2006 - 8/2006
CATERPILLAR INC., Champaign, IL
§§ Performed analysis for the Structural/Optimization group at the Champaign Simulation Center.
§§ Conducted non-linear structural analysis.
§§ Conducted fatigue analysis.
§§ Conducted linear static analysis using finite element models for different components of Caterpillar machines for validation of simulations with test results and for identifying weak spots in new designs.
§§ Conducted non-linear static analysis to identify structural weaknesses in designs during the development of new machines.

Analyst - CAT PRACTICUM Internship
5/2005 - 8/2005
CATERPILLAR INC., Champaign, IL
§§ Performed analysis for the Structural/Optimization group at the center.
§§ Conducted non-linear structural analysis.
§§ Conducted modal analysis.

Research Assistant
8/2000 - 5/2003
UIUC, Department of Aerospace Engineering, Urbana, IL
§§ Member of the Smart Icing Systems group, SIS, a multidisciplinary group dedicated to the development of a semi-autonomous icing protection system for small aircraft.
§§ Developed numerical algorithms to enhance the envelope protection systems of aircrafts impaired under icing conditions.
§§ Developed C code to implement the algorithms in a flight simulator.
§§ Developed a neural network in MATLAB to characterize icing effects as a function of aerodynamic coefficients.

Teaching Assistant
8/2000 - 12/2000
UIUC, Department of Aerospace Engineering, Urbana, IL
§§ Instructor for AAE 260, the undergraduate fluid dynamics lab course.

Instructor, Illinois Aerospace Institute Summer Camp
Summer 2001, Summer 2002, Summer 2003
UIUC, Department of Aerospace Engineering, Urbana, IL
§§ Instructor for aerodynamics sessions.
§§ Instructor for glider building sessions.

Research Assistant
8/1997 - 5/2000
Lafayette College, Easton, PA
§§ Conducted research on the behavior of viscoelastic materials under tensile loads.
§§ Developed a diagnostic setup to study thermoforming using thermocouples in conjunction with the Labview diagnostic interface.
§§ Designed and manufactured a pseudo fluidized bed for coating prepregs with microparticles.

Publications

§§ Hossain K., Jackson T., Buckmaster J., Numerical Simulations of Flame Patterns Supported by a Spinning Methane Burner, Submitted to the 32nd International Symposium on Combustion, 2008.

§§ Hossain K., Jackson T., Buckmaster J., Three-dimensional Simulations of Flames Supported by a Spinning Porous Plug Burner, AIAA 2008-1047, 2008.

§§ Wang, X., Hossain K., Jackson T., The Three-dimensional Numerical Simulation of Aluminized Composite Solid Propellant Combustion, Combustion Theory and Modelling, 11, 4, 2007.

§§ Hossain K., Sharma V., Bragg M., Voulgaris P., Envelope Protection and Control Adaptation in Icing Encounters, AIAA 2003-0025, 2003.

§§ Merret J., Hossain K., Bragg M., Envelope Protection and Atmospheric Disturbances in Icing Encounter, AIAA 2002-0814, 2002.

§§ Hummel, S. Hossain K., Hayes G., Biaxial Stress Relaxation of High Impact Polystyrene (HIPS) Above the Glass Transition Temperature, Polymer Engineering and Science, 41, 3, pg 566, 2001.

Activities

§§ Member of the Graduate Student Advisory Committee, Department of Aerospace Engineering, UIUC, 2006-2007