ELLIPTICAL INSTABILITY IN THE PLANETARY FLUID CORES

ALI MORADI
Bachelor of Science, Amirkabir University of Technology, 2011

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Physics and Astronomy
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

Dedication

This dissertation is lovingly dedicated to my mother, Maryam Zaree. Her support, encouragement, and constant love have sustained me throughout my life.

Abstract

Elliptical instability may be excited in any rotating flow with elliptically deformed streamlines. Investigating this instability in containers with spheroidal or ellipsoidal boundaries is of geophysical and astrophysical interest as many stars and planets are either rotating ellipsoidal fluid bodies or have substantial fluid cores which are either ellipsoidal, in the absence of a solid inner core, or ellipsoidal shells such as the Earth's fluid core; elliptical instability may be excited in these bodies as a result of the gravitational pull of a secondary body such as a moon or a large asteroid orbiting these bodies. In this thesis, the nonlinear evolution of elliptical instability in an inviscid incompressible rotating triaxial ellipsoid is numerically studied using the least-square finite element method. After validating the method by reproducing some known results, it is applied to other configurations in order to investigate some open questions on this subject, namely, the effects of the oblateness of the ellipsoid and the frequency ratio of the orbital speed of the secondary body on the evolution of the elliptical instability. We have found that if the parameters of the system, i.e. the flattening ratio and the frequency ratio of the background rotation, are in the range of the spin-over instability, a repetitive three-dimensional rigorous motion is maintained indefinitely; otherwise, instability may be excited initially, once the streamlines become elliptical, for certain ranges of the system parameters; however, as time elapses the motion becomes two dimensional with small displacement amplitudes in x- and y- directions.

Acknowledgments

I would like to express my deepest appreciation and thanks to my supervisor, Professor Behnam Seyed-Mahmoud, for his continuous support of my research, his guidance, patience, motivation, and enthusiasm. Without his support and guidance this dissertation would not have been possible. *Thank you*.

I would like to thank my committee members, Professor Saurya Das and Professor Amir Akbary-Majdabadno, for their encouragement and insightful suggestions helped improve the quality of this research.

I also wish to thank Dr. John Brigham (assistant professor of the Structural Engineering and Mechanics department at University of Pittsburgh) and his PhD student, Mohammad Ahmadpoor, who kindly meshed the geometry for us using ABAQUS software.

Many thanks are also due to Dr. David Cebron (post-doc at ETH Zurich) for helpful discussions on the problem.

I am also grateful to the School of Graduate Studies for the opportunity they provided for me to work on this project. I also thank Compute Canada for they provided me with the resources required to conduct the numerical simulations.

There are way too many people inside and outside the university who have supported, motivated, and helped me through this process. *Thank you all*.
Financial support was provided by the School of Graduate Studies, and the Natural Sciences and Engineering Research Council (NSERC) of Canada, the Discovery Grants Program.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The term "elliptical instability" is the name given to the linear instability mechanism by which three-dimensional flows can be generated in regions of two-dimensional elliptical streamlines (see Aldridge et al., 1997; Gledzer and Ponomarev, 1978; Kerswell and Malkus, 1998; Seyed-Mahmoud et al., 2000; Kerswell, 2002). Elliptical instability is developed when an otherwise circular flow is deformed into an elliptical one in such a way that the elliptical streamlines conserve their figure in the reference frame attached to the elliptical perturbation, as long as the flow remains linear. The amplitude of the flow then grows exponentially and the flow becomes unstable. This phenomenon may be developed in the planetary fluid cores as a result of the gravitational pull of a secondary body such as a moon or an asteroid (figure 1.1).

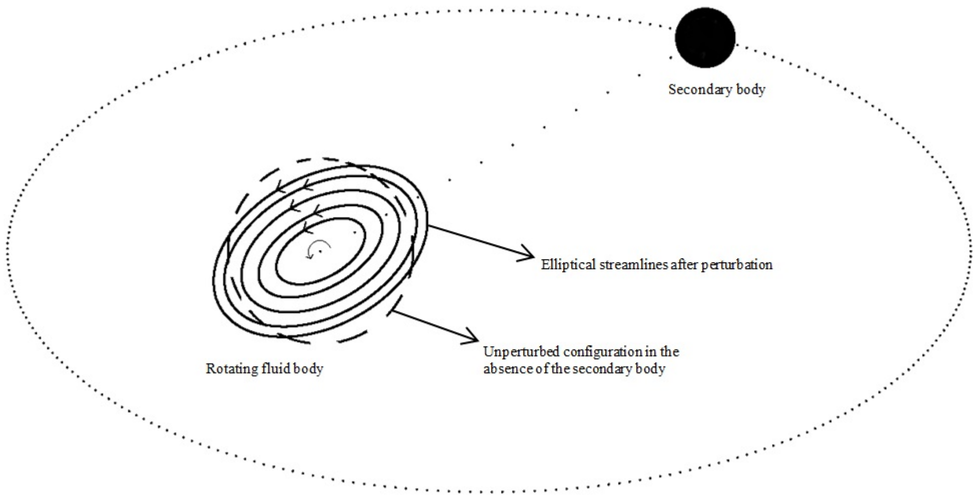

Secondary body
Elliptical streamlines after perturbation
Unperturbed configuration in the absence of the secondary body
Rotating fluid body

Figure 1.1: Deformation of the streamlines as a result of the gravitational pull of a secondary body.

In the case of the flows with circular streamlines, the Coriolis force acts as a restoring force and excite a type of waves known as inertial waves. In elliptical instability, the elliptical deformation couples inertial waves in pair and may excite instability if certain criteria are met.

To have a better picture of the elliptical instability, imagine a deformable-boundary cylinder filled with water and mounted on a turntable, with the central axis of the cylinder aligned with the center of the turntable. If the turntable rotates at the rate Ω for a long time, a dyed fluid element inside will follow a circular path with its center on the central axis of the cylinder. At any angular position the velocity of the dyed element is $r\Omega$, with r being the shortest distance from the central axis of the cylinder to the element and the velocity vector is always tangent to the circle, i.e. to the boundary of the cylinder, and perpendicular to the rotation axis. Now imagine two long rollers which are parallel to the central axis of the cylinder and are fixed in the laboratory frame, press against the cylinder at the two ends of the diameter of the cylinder along the x axis (see Malkus, 1989). This mechanism causes the lateral circular cross section of the cylinder to become elliptical with the semi-major axis along the y axis and semi-minor along the x (the x and y axes are perpendicular to the rotation axis and are fixed in the laboratory frame). Recall that the cylinder's boundary now rotates past the rollers, as the rollers are stationary. Since the fluid is incompressible, the streamlines all become elliptical parallel to the boundary. Now, the dyed element would follow an elliptical streamline and as such, in order to conserve its angular momentum, it slows down as it approaches the x axis and speeds up as it approaches the y axis, a process which causes the amplitude of the motion to grow exponentially and eventually break down. This phenomenon may also be excited in an ellipsoid as long as the rotation axis is along the middle axis of the ellipsoid.

The elliptical instability, whose existence is related to a parametric resonance of inertial waves, is well-known in aeronautics (see for instance Leweke and Williamson, 1998; Billant et al., 1999; Lacaze et al., 2007), and more generally in the field of vortex dynamics. Pierrehumbert (1986) calculated the growth rate of a three-dimensional perturbation on the two-dimensional velocity field associated with an elliptical vortex in an incompressible inviscid flow with zero normal-flow boundary condition. Based on the assumption of periodicity in the axial direction, spectral methods were applied to solve the linear eigenvalue problem. Leweke and Williamson (1998) studied coupled elliptical instability of a vortex pair experimentally and numerically. The instability of a multipolar vortex in a deformable cylinder is investigated in the works of Eloy et al. (2000, 2003). Le Dizès (2000) expanded the elliptical instability equations to account for Coriolis effects and higher order symmetries in the local stability of the core of a non-axisymmetric vortex in a rotating frame. Le Dizès and Laporte (2002) used direct numerical and large-eddy simulations to study elliptical instability in a two-vortex flow. Meunier et al. (2002) studied elliptical instability in the context of merging of a pair of equal two-dimensional co-rotating vortices. Lacaze et al. (2007) studied elliptical instability in a strained Batchelor vortex[1]. Their work was followed by the paper of Roy et al. (2008) in which elliptical instability is investigated in two strained Batchelor vortices. Schaeffer and Le Dizès (2010) studied nonlinear dynamics of elliptical instability of a single strained vortex and a system of two counter-rotating vortices.

Greenhill (1879) and Poincaré (1910) were among the firsts to study rotating fluid ellipsoids. They concluded that the rotation of a fluid ellipsoid is stable about the minor and major axis but unstable about the intermediate axis. Since its rediscovery in the mid-1970s, the elliptical instability has received considerable attention, theoretically, experimentally and numerically. Gledzer et al. (1976) and Gledzer and Ponomarev (1978) described the instability of a rotating fluid ellipsoid with spatially non-uniform vorticity field. Vladimirov and Vostretsov (1986) developed a model for the instability of a steady flow in an ellipsoid by applying a perturbation method based on the ratio of the equatorial ellipticity and the height of the ellipsoid. Kerswell (1993) studied the dynamics of the fluid instability of

[1]Batchelor vortex is a self-similar solution of the Navier-Stokes equation obtained using a boundary layer approximation.

a precessing rotating spheroidal container. Kerswell (1994) investigated the elliptical instability in a rotating spheroid in the presence of Ohmic and viscous decays. Mason and Kerswell (1999) solved the flow in an elliptically deformed cylinder by spectral methods using a non-orthogonal elliptico-polar coordinate system; the nonlinear temporal evolution of two different modes of the elliptical instability was calculated with no-slip boundary conditions on the sidewalls and stress-free conditions on the top and the bottom of the cylinder. Lacaze et al. (2004) studied the spin over instability in a slightly deformed sphere experimentally and analytically, and developed a nonlinear model in order to describe the observation. Ou et al. (2007) studied the stability of self-gravitating compressible ellipsoidal fluid configurations and observed the occurrence of the elliptical instability. Le Bars et al. (2007) conducted experiments to study the effects of the Coriolis force on the elliptical instability in cylindrical and spherical containers. They concluded that in addition to the simple spin-over modes driven by both precession and elliptical instability, complex motion can be excited due to Coriolis force. An experimental and numerical study of the nonlinear evolution of the elliptical instability in a rotating viscous fluid ellipsoids is given in Cébron et al. (2010a). They used a finite element method to investigate the effects of the equatorial ellipticity, length of the ellipsoid along the rotation axis, background rotation, and obliquity of the ellipsoid on the growth rate of the elliptical instability and the modes selected. Clausen and Tilgner (2013) investigated the elliptical instability of compressible flow in a rotating ellipsoid in the linear regime and found that although the compressibility of the fluid affects the growth rate of a certain mode combination, its influence in negligible for the growth rate maximized over all possible modes. The nonlinear evolution of libration driven elliptical instability was investigated numerically and experimentally for the first time by Cébron et al. (2012b) and Noir et al. (2012).

Elliptical instability in spheroidal and triaxial ellipsoidal shells has also been the topic of some studies. For instance, Aldridge et al. (1997) studied the elliptical instability in a thick spherical rotating shell experimentally and theoretically. Seyed-Mahmoud et al.

(2000) applied numerical techniques to compute the frequencies and the growth rates of the instability in both triaxial ellipsoid and ellipsoidal shell geometries using a linear Galerkin method, projecting the flow on a selected number of inertial waves. Seyed-Mahmoud et al. (2004) conducted experiments on spheroidal thick rotating fluid shells and observed the excitation of the elliptical instability and identified the modes predicted using theoretical methods. Lacaze et al. (2005) calculated the growth rate of the spin-over instability experimentally in a rotating spheroidal thick shell and conducted a linear stability analysis and determined the growth rate of the spin-over instability as a function of the Ekman number. Furthermore, they predicted the saturated amplitude by a simple nonlinear model for the spin-over instability amplitude. Cébron et al. (2010b) studied the stability of a rotating flow inside a fixed ellipsoid with an imposed temperature difference between inner and outer boundaries.

The presence of elliptical instability in planetary and stellar systems, elliptically deformed by gravitational tides due to the presence of nearby astronomical bodies, has been suggested for several decades. It could for instance be responsible for the surprising existence of a magnetic field in Io[2] (Kerswell and Malkus, 1998; Lacaze et al., 2006; Herreman et al., 2009) and for fluctuations in the Earth's magnetic field on a typical timescale of 10,000 years (Aldridge et al., 1997). It may also have a significant influence on the evolution of binary stars (e.g. Rieutord, 2003). Arkani-Hamed et al. (2008) have proposed that the Mars' dynamo, which was responsible for the magnetic field frozen in the planet's crust, was also energized by an elliptical instability developed in the Mars' fluid core as a result of the gravitational pull of a giant asteroid orbiting Mars early in the planet's history. Eventually the asteroid crashed into Mars and the planet's dynamo diminished. Cébron et al. (2012a) developed a numerical geodynamo based on elliptical instability for an incompressible viscous tri-axial ellipsoid for the first time. They studies the magnetic field induced by the elliptical instability developed as a result of the interactions of different in-

[2]Io is the innermost of the four Galilean moons of the planet Jupiter and, with a diameter of 3,642 kilometres (2,263 mi), the fourth-largest moon in the Solar System.

ertial modes and computed the decay rates of magnetic field when the external magnetic field is suddenly shut down.

It is now well understood that Earth's magnetic field is generated in the Earth's fluid core (Bullard and Gellman, 1954; Olson, 1983; Roberts, 1987) with the model that best explains this field called the geomagnetic dynamo or geodynamo. Although the existence of the geodynamo is now widely accepted, the source of energy required to drive this system is not well established. Due to the extreme conditions of the Earth's core, numerical simulation of the geodynamo with realistic parameters is still beyond the today's computing power. Among the non-dimensional parameters, the Ekman number, E, is of major importance and has been chosen as a barometer of the difference between numerical simulations and the real core conditions, where E is of the order of 10^{-15} (Sakuraba and Roberts, 2009; Kageyama et al., 2008). In simulations where a strong, Earth-like field was generated, unrealistically high Ekman numbers, as low as of the order of 10^{-6}, were used (Glatzmaier and Roberts, 1995; Kageyama et al., 1995; Kuang and Bloxham, 1997; Takahashi et al., 2005). In most of the geodynamo models, fixed temperature boundary condition is employed for the core-surface boundary. Using this boundary condition, some of the recent numerical solution with a more realistic Ekman number, as low as of the order of 10^{-7}, have generated a non-polar (Kageyama et al., 2008) or dipolar but comparatively weak magnetic field (Christensen and Aubert, 2006; Takahashi et al., 2008). Sakuraba and Roberts (2009) suggested that even with a small Ekman number, of order of 10^{-7}, by using a laterally uniform heat flux condition at the core's surface, a comparatively strong dipolar magnetic field can be generated. The current geodynamo models are based on the work of Glatzmaier and Roberts (1995) in which the geodynamo is based on thermal convection in the core. However, if Earth's core is predominantly adiabatic, which is the case for most of the existing Earth models (e.g., PREM); it is unlikely that thermal convection would produce enough energy to drive the geodynamo. Even if thermal convection does occur in the core, an elliptical instability, if excited, would contribute to the flow field and, therefore, is a problem which must be considered.

It is also well established that the magnitude of Earth's magnetic field fluctuates with time (Merrill and McElhinny, 1983; Roberts, 1987), and Earth's magnetic field reverses itself on the time scale of approximately 100,000 years (McFadden et al., 1991). The process of growth and decay of the amplitude of motion and the break-down of the flow, once the tidal instability is excited, could provide an explanation for the fluctuations in Earth's magnetic field on a geomagnetic time-scale. Indeed, the collapse of the flow after it grows to a certain magnitude could conceivably provide an explanation for the geomagnetic reversals (Seyed-Mahmoud et al., 2004).

In this work, we have studied the elliptical instability in an incompressible inviscid triaxial ellipsoid using least-square finite element method. We have found that if the parameters of the system, i.e. the flattening ratio of the ellipsoid and the frequency ratio of the background rotation, are in the range of the spin-over instability, a repetitive three-dimensional rigorous motion is maintained indefinitely; otherwise, instability may be excited initially, once the streamlines become elliptical, for certain ranges of the system parameters; however, as time elapses the motion becomes two dimensional with small displacement amplitudes in x- and y- directions.

In chapter 2, a description of the least-square finite element method and the reason that this method was chosen for this problem is given. In chapter 3, the OpenMP Library and its concepts are reviewed briefly. In chapter 4, the equations and the results are presented. Finally, in chapter 5, some conclusions are drawn.

Chapter 2

The Least-Square Finite Element Method

Practically every phenomenon in nature may be described using the laws of physics, in terms of mathematical problems. Any such modeling involves two major processes: mathematical formulation of the physical phenomenon and the analytical/numerical analysis of the model.

Modeling a physical process requires background in related subjects and certain mathematical tools. Although finding a model, for most physical processes, is not extremely difficult, solving the equations using exact methods may be an arduous job. In such cases, approximate methods provide an alternative means of finding solutions.

The finite element method is one of the most general techniques for the numerical solution of differential equations. It has been amazingly successful. Perhaps no other family of approximation methods has had a greater impact on the theory and application of numerical methods during the twentieth century (Jiang, 1998). The finite element method has been used extensively in the fields of engineering and applied sciences.

In finite element method, instead of dealing directly with differential equations, the variational forms of the continuous boundary and initial value problems are derived. The solution is in the integral of a quantity of a domain. The integral of a function over an arbitrary domain can be divided into sum of integrals over many small, interconnected subdomains called finite elements. If the finite elements are small enough, the local behavior of the solution can be sufficiently described using polynomial approximations. The most important advantages of the finite element method can be outlined as:

Arbitrary Geometries. Since the finite element method is independent of geometry, it can be applied to domains of complex geometries with arbitrary boundary conditions.

Unstructured meshes. In finite element analyses a global coordinate transformation is not needed. Finite elements can be placed anywhere in physical domain. In engineering, the original design is usually modified to meet different requirements. In finite element method, elements can be added or deleted without changing the global data structure. If iterative solvers are employed, the element and nodal numbering can be arbitrary without sacrificing efficiency.

Flexible and general purpose format of program. Due to clear structure and versatility of the finite element method, general purpose software can be developed for application.

Mathematical foundation. Researchers and scientists have worked extensively on the mathematical theorys. As a result, the finite element method now has a solid and enriched mathematical basis which has added to its reliability.

Since this method is based on a variational principle, there are three major groups of finite element methods: the Rayleigh-Ritz method, the Galerkin method and the Least-Square method.

In Rayleigh-Ritz method, the total potential energy is minimized. Consequently, the best approximation solution can be found using this method since the difference between the finite element solution and the exact solution is minimized with respect to a certain energy norm. In application to phenomena governed by self-adjoint, second- or fourth-order elliptic diffusion-type equations, the Rayleigh-Ritz method has shown great success.

In Galerkin method the weighted residual form is employed. To demonstrate the basic idea of Galerkin method, a mathematical problem defined by a set of differential equations is reviewed:

$$\mathbf{Au} = \mathbf{f} \quad \text{in } \Omega, \tag{2.1a}$$

$$\mathbf{Bu} = \mathbf{0} \quad \text{on } \Gamma, \tag{2.1b}$$

where $\mathbf{A}$ is the linear partial differential operator, $\mathbf{B}$ is the boundary operator, $\mathbf{u}$ is the dependent unknown vector, $\mathbf{f}$ is the force vector, Ω is the domain, and Γ is the boundary of Ω.

The weighted residual form can be approximated as follows. First, the function is approximated by a set of unknown parameters $\mathbf{u}_j$ and trial functions $\Phi_j(\mathbf{x})$, where $\mathbf{x}$ is the independent variables vector,

$$\mathbf{u} \simeq \Phi_j \mathbf{u}_j \quad j = 1, \ldots, n \tag{2.2}$$

Then, the algebraic equation is formed as a "weighted residual",

$$\int_\Omega \mathbf{v}_i^T (\mathbf{Au} - \mathbf{f}) d\Omega + \int_\Gamma \overline{\mathbf{v}}_i^T \mathbf{Bu} d\Omega = 0 \tag{2.3}$$

where $\mathbf{v_i}$ and $\overline{\mathbf{v}}_\mathbf{i}$ are appropriately chosen test functions, and T denotes the transpose. In the conventional Galerkin method, the choice is

$$\mathbf{v}_i = \overline{\mathbf{v}}_\mathbf{i} = \Phi_\mathbf{i} \tag{2.4}$$

The resulting system of equations for self-adjoint and positive definite operators are the same for both the Galerkin and the Rayleigh-Ritz formulation. The Galerkin method is more general than the Rayleigh-Ritz method since it is also applicable to non-self-adjoint equations such as arise in fluid mechanics. However, since in first order problems, such as convection dominated transport, Galerking method leads to non-self-adjoint problems, solution of the Galerkin method are usually corrupted by spurious oscillations. These can only be overcome by severe mesh refinements which clearly cripple the practical advantage of the method. The classic Galerkin method is not also the best choice for high-speed compressible flow and shallow water wave problems where the flow is governed by nonlinear first-order hyperbolic equations.

The Galerkin mixed method was developed to overcome the difficulties arise for the solution of elliptic problems. For instance, incompressible irrotational flow problems are governed by Laplace or Poisson equations of the potential. Usually, the primal variable (i.e. the potential) is solved by the Rayleigh-Ritz method, then a *posteriori* numerical differentiation is required to obtain the dual variables (i.e. the velocity components). Because of this *posteriori* numerical differentiation, the velocity accuracy is one order lower than that of the potential and it is not continuous across the element boundary. The mixed Galerkin method was formulated based on the first-order differential equations in order to obtain better accuracy for both primal and dual variables. However, unfortunately, at least for second-order elliptic problems, it does not work as expected for a couple of reasons. First, instead of facing the original simple minimization problem, a difficult saddle-point problem should be solved and to meet the Ladyzhenskaya-Babuska-Brezzi (LBB) condition, different elements should be used to interpolate primal and dual variables. Consequently, better accuracy for dual variables may not be achieved. Second, while the Rayleigh-Ritz method produces symmetric positive-definite matrices, the mixed Galerkin method leads to non-positive-definite matrices which have been hard to solve for large-scale problems.

The least-square finite element method (LSFEM) is based on the minimization of the residuals in a least-square sense. The method seeks the minimizer of the following function:

$$I(\mathbf{u}) = \int_{\Omega} (\mathbf{A}\mathbf{u} - \mathbf{f})^2 d\Omega \tag{2.5}$$

within the constraint of a given boundary condition (2.1b). The least-square solution is calculated from the following variational statement:

$$\int_{\Omega} (\mathbf{A}\Phi_{\mathbf{i}})^{T} (\mathbf{Au} - \mathbf{f}) d\Omega \tag{2.6}$$

While the Galerkin formulation (2.3) is only conceptual without much practical meaning and hence in most cases needs further problem-related mathematical manipulation to achieve a realistic computational formulation for a particular problem, the least square formulation (2.6), is a final mathematical formulation as well as computational formulation for any problem. Even though the LSFEM formulation is simple, it has many advantages:
Universality. Unlike finite difference or other finite element methods which have different schemes for different types of differential equations, the LSFEM has a unified formulation for the numerical solution of all types of partial differential equations. As long as the equations have a unique solution, the LSFEM can always find a good approximate solution independent of type of equations.
Efficiency. In many areas of engineering and applied science, the governing partial differential equations are either of first-order or can be turned into a set of first-order equations. Although, it is difficult to deal with first-order equations by conventional methods since such equations generally lead to non-symmetric matrices for all other methods, LSFEM is naturally suited for first-order differential operators. Also, for the case of linear partial differential equations, the LSFEM always produces symmetric positive-definite matrices which can be efficiently solved by matrix-free iterative methods, such as the preconditioned conjugate gradient method.
Robustness. In traditional schemes, special treatments are necessary for different problems, such as upwinding, artificial dissipation, staggered grid or non-equal-order elements, artificial compressibility, operator-splitting, operator-preconditioning, and etc. However, all these treatments are redundant when the LSFEM is invoked. Also, the LSFEM can be applied without the limitation of the LBB condition, so equal-order elements can be used which makes programming much easier.

Optimality. In many cases it can be rigorously proven than the LSFEM solution is the best approximation. In other words, the error of the LSFEM solution has the same order as the interpolation error.

Concurrent simulation of multiple physics. Since the LSFEM formulation is a unified method for finding the approximate solution of differential equations in diverse physical phenomena, one-code may be developed for concurrent analysis of different disciplines like conjugate heat transfer or solid-fluid interaction.

General-purpose coding. Thanks to very general setting of the LSFEM formulation, it can be programmed systematically and as a result for new applications, only subroutines for calculating the coefficients, the load vector, and the boundary conditions for the first-order system need to be added to the code. Therefore, the time, cost and programming errors can be immensely reduced in code development.

In summary, the LSFEM is a simple and universal method. Although, there is nothing beyond the finite element interpolation and the least-square principle in the LSFEM, it offers a lot of advantages which make it one of the most attractive methods to handle real-world problems.

2.1 An Example

In order to demonstrate that the classic Galerkin method leads to oscillatory solution, and why the least-square method does not need upwinding and is utterly suitable for the solution of first-order differential equation, a very simple one-dimensional model problem is reviewed.

Consider a first-order scalar ordinary differential equation in the interval [0,1]

$$\begin{aligned} u'(x) &= f(x), \\ u(0) &= 0 \end{aligned} \tag{2.7}$$

where $u' = du/dx$, and f is a given continuous function. Ω is a bounded one-dimensional domain. If all $u(x), u'(x), \ldots, d^m u/dx^m$ are continuous in the closed interval $\overline{\Omega}$, then u is said to belong to $C^m(\Omega)$. By integrating $u' = f$ once, it is easy to check whether this problem has a unique classic solution u or not. The smoothness and regularity of the classic solution $u(x)$ depends on the smoothness of f. If $f \in C^m(\Omega), m \geq 0$, then the classic solution $u \in C^{m+1}(\Omega)$.

2.1.1 Function Spaces $\mathbf{H^m}(\Omega)$

The problem (2.7) shall now be converted into a variational problem. For this, new function spaces $H^m(\Omega)$ which are larger than $C^m(\Omega)$ should be considered. The space of "square integrable" functions on Ω is defined as:

$$H^0(\Omega) = L_2(\Omega) = \{u : u \text{ is defined on } \Omega \text{ and } \int_\Omega u^2 dx < \infty\}. \tag{2.8}$$

The norm of u is denoted by

$$\|u\|_0 = \left\{ \int_\Omega |u(x)|^2 dx \right\}^{1/2}. \tag{2.9}$$

Continuous functions and piecewise continuous functions are typical functions belonging to $H^0(\Omega)$. Note that "square integrable" is a stronger requirement than "integrable".

If u is continuous on $\overline{\Omega}$, the derivative $u'(x) \in H^0(\Omega)$ and $u(x) - u(x_o) = \int_{x_o}^{x} u'(t)dt$, then u is said to belong to the function space $H^1(\Omega)$. The corresponding norm is

$$\|u\|_1 = \left\{ \int_\Omega (|u(x)|^2 + |u'(x)|^2)dx \right\}^{1/2} \tag{2.10}$$

A semi-norm is also defined for the function space $H^1(\Omega)$:

$$|u|_1 = \left\{ \int_\Omega |u'(x)|^2 dx \right\}^2.$$

Generally if the function $u \in C^{m-1}(\Omega)$ and $u^{(m)} \in H^0(\Omega)$, then it is said that $u \in H^m(\Omega)$.

2.1.2 The Basic Lemma of Variational Principles

Assume that $f(x) \in C^0(\Omega)$ and

$$\int_\Omega f(x)\phi(x)dx = 0 \quad \forall \phi \in C^0(\Omega),$$

then $f(x) = 0$.

2.1.3 The Classic Galerkin Method Global Approximation

The Galerkin method is a member of the class of weighted residual methods. Assume an approximate solution written as

$$u(x) = a_1\phi_1(x) + a_2\phi_2(x) + \ldots + a_n\phi_n(x) \quad x \in \Omega = [0, 1] \tag{2.11}$$

where the basis functions $\phi_1, \ldots, \phi_n$ are known. Of course, $u(x)$ should satisfy the boundary condition $u(0) = 0$. The coefficients a_j are to be determined. If the approximate solution

(2.11) is substitute into (2.7), it will not generally be identically zero. Hence we can write

$$R(x) = u'(x) - f(x) \tag{2.12}$$

where $R(x)$ is called the equation residual.

In weighted residual method, the coefficient a_j are determined by requiring that the integral of the weighted residual over the computational domain is zero.

$$\int_\Omega v(x)R(x)dx = 0 \tag{2.13}$$

If for any test function $v(x) \in C^0(\Omega)$, the integral in (2.13) is zero, because of the basic lemma of variational principles, the residual R will be identical to zero. Nonetheless it is unnecessary to test R by all C^0 functions in [0, 1]. In practice, due to the fact that the approximate solution $u(x)$ has only n degrees of freedom, n test functions would be enough. Different choices for the test functions $v(x)$ in (2.13) lead to different methods. In Galerkin method, the test functions are from the same family as the approximate functions. If the approximate functions form a complete set, the residual is orthogonal to every member of the complete set.

2.1.4 The Least-Square Method Global Approximation

In the least-square method, the coefficients a_j in (2.11) are determined by minimizing the integral of the square of the residual (2.12) over the computational domain. Thus, a quadratic function is constructed

$$\begin{aligned} I(u) &= \|R(u)\|_0^2 = \int_0^1 \{u'(x) - f(x)\}^2 dx, \\ I &: V \to \mathbb{R} \end{aligned} \tag{2.14}$$

over all $u \in V = \{u \in H^1(0,1) : u = 0 \text{ at } x = 0\}$. For $u \in V$ to be a minimizer of the functional I in (2.14), the first variation of I should vanish at all u for all admissible v.

$$\frac{d}{dt} I(u+tv)\Big|_{t=0} \equiv 2\int_0^1 \{u' - f(x)\}v'dx = 0 \quad \forall v \in V,$$

or

$$(u', v') = (f, v'). \tag{2.15}$$

2.1.5 One-Dimensional Finite Elements

In the finite element method, continuous piecewise polynomials are chosen as trial functions. A finite-dimensional subspace V_h of the space V consists of piecewise linear functions. The nodes

$$0 = x_0 < x_1 < x_2 < \ldots < x_{n-1} < x_n = 1$$

are used to divide the interval [0,1] into n elements $e_j = (x_{j-1}, x_j), j = 1, 2, \ldots, n$ of length $h_j = x_j - x_{j-1}$. The quantity h, $h = \max\, h_j$, is a measure of how fine the mesh is. A trial function u_h is constructed such that it is linear on each element e_j, and it is continuous on [0,1] and satisfy the boundary condition $u_h(0) = 0$. These functions constitute the subspace V_h and $V_h \subset V$. On each element $e_j = (x_{j-1}, x_j)$, $u_h(x)$ can be expressed as

$$u_h(x) = \psi_1^{(j)}(x)u_{j-1} + \psi_2^{(j)}(x)u_j \quad x \in e_j, \tag{2.16}$$

in which the interpolation functions

$$\psi^{(j)}(x) = \begin{pmatrix} \psi_1^{(j)} \\ \psi_2^{(j)} \end{pmatrix} = \begin{pmatrix} (x_j - x)/h_j \\ (x - x_{j-1})/h_j \end{pmatrix}. \tag{2.17}$$

Thus, on the entire domain [0,1], $u_h(x)$ can be written as

$$u_h(x) = \phi_0(x)u_0 + \phi_1(x)u_1 + \cdots + \phi_n(x)u_n \tag{2.18}$$

in which

$$\phi_j(x) = \begin{cases} \psi_2^{(j)}(x), & x \in e_j; \\ \psi_1^{(j+1)}(x), & x \in e_{j+1}; 1 \le j \le n-1 \\ 0, & \text{otherwise}; \end{cases}$$

$$\phi_0(x) = \begin{cases} \psi_1^{(1)}(x), & x \in e_1; \\ 0, & \text{otherwise}; \end{cases} \tag{2.19}$$

$$\phi_n(x) = \begin{cases} \psi_2^{(n)}(x), & x \in e_n; \\ 0, & \text{otherwise}; \end{cases}$$

2.1.6 The Classic Galerkin Finite Element Method

The Galerkin finite element method for the first-order equation (2.7) can be expressed as follows: Find $u_h \in V_h$ such that

$$\int_0^1 \left\{ u_h' - f(x) \right\} v_h dx = 0 \quad \forall v_h \in V_h. \tag{2.20}$$

Since

$$u_h(x) = \sum_{j=1}^{n} \phi_j(x) u_j, \tag{2.21}$$

and (2.20) must be valid by taking $v_h = \phi_i(x),\ i = 1, 2, \ldots, n$ as the test function, (2.20) can be written as

$$\sum_{j=1}^{n} (\phi_i, \phi_j') u_j = (\phi_i, f) \quad i = 1, 2, \ldots, n, \tag{2.22}$$

(2.22) is a system of linear algebraic equations with n equations in n unknowns $u_1, u_2, \ldots, u_n$. In matrix form

$$\mathbf{KU} = \mathbf{F}, \tag{2.23}$$

where the global matrix $\mathbf{K} = (K_{ij})$ is $n \times n$ matrix with entries $K_{ij} = (\phi_i, \phi'_j)$, the global unknown vector $\mathbf{U}^T = (u_1, u_2, \ldots, u_n)$ and the global force vector $\mathbf{F}^T = (F_1, F_2, \ldots, F_n)$ with $F_i = (\phi_i, f)$.The global matrix $\mathbf{K}$ and the global force vector $\mathbf{F}$ are assembled from the element matrices and the element force vectors, respectively.

In the global matrix $\mathbf{K}$ the entries $K_{ij} = (\phi_i, \phi_j)'$ can easily be computed. First, $(\phi_i, \phi'_j) = 0$ if $|i-j| > 1$, since in this case for all $x \in [0,1]$ either $\phi_i(x) = 0$ or $\phi_j(x) = 0$. Therefore, the matrix $\mathbf{K}$ is tri-diagonal. For $i = 1, 2, \ldots, n-1$

$$\begin{aligned} K_{ii} = (\phi_i, \phi'_i) &= \int_{x_{i-1}}^{x_i} \frac{(x - x_{i-1})}{h_i} \cdot \frac{1}{h_i} dx + \int_{x_i}^{x_{i+1}} \frac{(x_{i+1} - x)}{h_{i+1}} \cdot \frac{-1}{h_{i+1}} dx \\ &= \frac{1}{2} - \frac{1}{2} = 0, \end{aligned}$$

and for $i = 2, \ldots, n-1$

$$\begin{aligned} K_{i,i-1} = (\phi_i, \phi'_{i-1}) &= \int_{x_{i-1}}^{x_i} \frac{(x - x_{i-1}}{h_i} \cdot \frac{-1}{h_i} dx = -\frac{1}{2}, \\ K_{i,i+1} = (\phi_i, \phi'_{i+1}) &= \int_{x_i}^{x_{i+1}} \frac{(x_{i+1} - x)}{h_{i+1}} \cdot \frac{1}{h_{i+1}} dx = \frac{1}{2}, \end{aligned}$$

and

$$K_{nn} = (\phi_n, \phi'_n) = \int_{x_{n-1}}^{x_n} \frac{(x - x_{n-1})}{h_n} \cdot \frac{1}{h_n} = \frac{1}{2}$$

F_j cannot be accurately computed unless the analytical expression of the function $f(x)$ is given. For the purpose of illustration, one-point Simpson rule is used to perform the quadrature. For $i = 1, 2, \ldots, n-1$

$$\begin{aligned} F_i = (\phi_i, f) &= \int_{x_{i-1}}^{x_i} \frac{(x - x_{i-1})}{h_i} \cdot f dx + \int_{x_i}^{x_{i+1}} \frac{(x_{i+1} - x)}{h_{i+1}} \cdot f dx \\ &\simeq \frac{1}{2} f_i h_i + \frac{1}{2} f_i h_{i+1} \end{aligned}$$

and

$$F_n = (\phi_n, f) = \int_{x_{n-1}}^{x_n} f \cdot \frac{(x - x_{n-1})}{h_n} dx \simeq \frac{1}{2} f_n h_n$$

In case of uniform mesh with $h_i = h = 1/n$, the system (2.22) takes the form

$$\frac{1}{2} \begin{pmatrix} 0 & 1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ -1 & 0 & 1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & -1 & 0 & 1 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & 0 & -1 & 0 & 0 \\ 0 & \cdot & \cdot & \cdot & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \cdot \\ \cdot \\ u_{n-1} \\ u_n \end{pmatrix} = h \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ \cdot \\ \cdot \\ f_{n-1} \\ 0.5 f_n \end{pmatrix} \tag{2.24}$$

(2.24) is identical to the standard central difference approximation obtained by putting

$$u' \simeq \frac{u_{i+1} - u_{i-1}}{2h}$$

The global stiffness[3] matrix is non-symmetric with zeros on the diagonal, and has odd-even decoupling which leads to oscillatory solutions. This problem cannot be overcome even by mesh refining. In order to circumvent this bad approximation problem, one-sided or

[3]The stiffness matrxi represents the system of linear equations that must be solved in order to ascertain an approximate solution to the differential equation.

upwind differencing can be used

$$u' \simeq \frac{u_i - u_{i-1}}{h} = \frac{u_{i+1} - u_{i-1}}{2h} - \frac{h}{2}\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2},$$

which is equivalent to adding some numerical dissipation into the central difference scheme. That is, instead of (2.7), the following second-order equation is solved by central difference method

$$u' - \frac{h}{2}u'' = f.$$

2.1.7 The Least-Square Finite Element Method

According to (2.15) the least-square finite element method is formulated as follows: Find $u_h \in V_h$ such that

$$\int_0^1 \left\{u_h' - f(x)\right\} v_h' dx = 0 \quad \forall v_h \in V_h. \tag{2.25}$$

Since

$$u_h(x) = \phi_1(x)u_1 + \cdots + \phi_n(x)u_n, \tag{2.26}$$

(2.25) can be written as

$$\sum_{j=1}^{n} (\phi_i', \phi_j')u_j = (\phi_i', f) \quad i = 1, 2, \ldots, n, \tag{2.27}$$

or in matrix form as

$$\mathbf{KU} = \mathbf{F}$$

One-point Simpson quadrature can be used to calculate force vector elements for $i = 1, 2, \ldots, n-$

1

$$\begin{aligned} F_i = (\phi_i', f) &= \int_{x_{i-1}}^{x_i} \frac{1}{h_i} f dx + \int_{x_i}^{x_{i+1}} \frac{-1}{h_{i+1}} f dx \\ &\simeq \frac{(f_{i-1} + f_i)}{2} \frac{1}{h_i} h_i - \frac{(f_i + f_{i+1})}{2} \frac{1}{h_{i+1}} h_{i+1} \\ &= -\frac{(f_{i+1} - f_{i-1})}{2}, \end{aligned}$$

and

$$F_n = (\phi_n', f) = \int_{x_{n-1}}^{x_n} \frac{1}{h_n} f dx \simeq \frac{(f_{n-1} + f_n)}{2}.$$

The global matrix **K** is tri-diagonal, so

$$\begin{aligned} K_{ii} = (\phi_i', \phi_i') &= \textstyle\int_{x_{i-1}}^{x_i} \frac{1}{h_i^2} dx + \int_{x_i}^{x_{i+1}} \frac{1}{h_{i+1}^2} dx \\ &= \tfrac{1}{h_{i+1}} \quad i = 1, 2, \ldots, n-1, \end{aligned}$$

$$K_{i,i+1} = (\phi_i', \phi_{i-1}') = -\textstyle\int_{x_i}^{x_{i+1}} \frac{1}{h_{i+1}^2} dx = -\frac{1}{h_{i+1}} \quad i = 1, 2 \ldots, n-1$$

$$K_{i,i-1} = (\phi_i', \phi_{i-1}') = -\textstyle\int_{x_{i-1}}^{x_i} \frac{1}{h_i^2} dx = -\frac{1}{h_i} \quad i = 2, \ldots, n-1,$$

and

$$K_{nn} = (\phi_n', \phi_n') = \textstyle\int_{x_{n-1}}^{x_n} \frac{1}{h_n^2} dx = \frac{1}{h_n}$$

The matrix **K** is symmetric and positive-definite.

In the case of uniform mesh with $h_i = h/1/n$, the system (2.27) takes the form

$$-\frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ -1 & 2 & -1 & 0 & \cdot & \cdot & \cdot & 0 \\ 0 & -1 & 2 & -1 & \cdot & \cdot & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot & 0 & -1 & 2 & -1 \\ 0 & \cdot & \cdot & \cdot & 0 & 0 & -1 & 1 \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \cdot \\ \cdot \\ u_{n-1} \\ u_n \end{pmatrix} = \frac{1}{2h}\begin{pmatrix} f_2 - f_0 \\ f_3 - f_1 \\ f_4 - f_2 \\ \cdot \\ \cdot \\ f_n - f_{n-2} \\ -f_n - f_{n-1} \end{pmatrix} \tag{2.28}$$

The left-hand side of (2.28) can be described as a standard central difference for u'' and the right-hand side as a central difference for f'. (2.28) shows that the LSFEM implementation of the first-order differential equations leads to a symmetric and positive-definite system of linear algebraic equations.

2.1.8 A Numerical Example

For further comparison of LSFEM with the Galerkin method and the finite difference method, numerical solution using LSFEM, the central finite difference method, the upwinding finite difference method, the Galerkin method, and the exact solution of a first-order differential problem is presented:

$$\begin{aligned} &u'(x) = \left[1 - \exp\left(-\tfrac{1}{\varepsilon}\right)\right]^{-1} \tfrac{1}{\varepsilon} \exp\left(-\tfrac{1-x}{\varepsilon}\right) \quad x \in [0,1], \\ &u(0) = 0 \end{aligned} \tag{2.29}$$

with $0 < \varepsilon << 1$. The exact solution of this problem is given by

$$u(x) = 1 - \left[1 - \exp\left(\frac{1}{-\varepsilon}\right)\right]^{-1} \left[1 - \exp\left(-\frac{1-x}{\varepsilon}\right)\right].$$

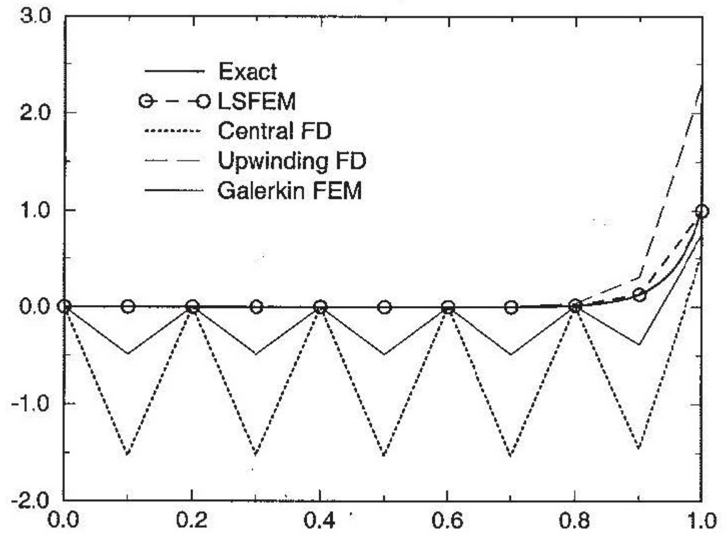


Figure 2.1: Solution of (2.29) with $\varepsilon = 0.05$.
(Jiang, 1998, chap. 2)

The Galerkin method is applied using piecewise linear elements on a uniform mesh with length $h = 0.1$. To calculate the right-hand side term of (2.24), each element is divided into 10 segments and the Simpson quadrature is used. As it is obvious in figure 2.1, the Galerkin solution oscillates violently in the whole domain and is not close to the exact solution.

The leas-square method is also applied and the same Simpson quadrature is used to calculate (ϕ_i', f). Albeit other methods do not work very well for this problem, the least-square solution with only 10 linear elements is very smooth and accurate.

For further reference, the solutions of the central finite difference method and the up-winding finite difference method are also shown in Figure 2.1.

2.2 General Formulation of LSFEM

In this section the general formulation of the least-square finite element method for steady-state boundary-value problem is given. In the case of time-dependent problems, a finite difference method may be used to discretize time derivatives. Consequently, the problems are converted into boundary-value problems at each time step.

$$\begin{aligned} \mathbf{Au} &= \mathbf{f} \quad \text{in } \Omega, \\ \mathbf{Bu} &= \mathbf{g} \quad \text{on } \Gamma, \end{aligned} \tag{2.30}$$

where **A** is a first-order partial differential operator:

$$\mathbf{Au} = \sum_{i=1}^{n_d} \mathbf{A_i} \frac{\partial \mathbf{u}}{\partial x_i} + \mathbf{A_0 u}, \tag{2.31}$$

in which $\Omega \in \mathbb{R}^{n_d}$ is a bounded domain with a piecewise smooth boundary Γ, n_d represents the number of space dimensions, $\mathbf{u}^T = (u_1, u_2, \ldots, u_m)$ is a vector of m unknown functions of $\mathbf{x} = (x_1, \ldots, x_{n_d})$; $\mathbf{A_i}$ and $\mathbf{A_0}$ are $N_{eq} \times m$ matrices which continuously depend on $\mathbf{x}$; $\mathbf{f}$ is a given vector-valued function; $\mathbf{B}$ is a boundary algebraic operator, and $\mathbf{g}$ is a given vector-valued function on the boundary. $\mathbf{g}$ may be assumed to be null without loss of generality.

Suppose that $\mathbf{f} \in \mathbf{L}_2(\Omega)$. $\mathbf{V}$ is an appropriate subspace of the Hilbert space $\mathbf{L}_2(\Omega)$. The functions in $\mathbf{V}$ satisfy the boundary condition:

$$\mathbf{Bv} = \mathbf{0} \quad \text{on } \Gamma. \tag{2.32}$$

The first-order differential operator **A** maps the subspace **V** into $\mathbf{L}_2(\Omega)$:

$$\mathbf{A} : \mathbf{V} \to \mathbf{L}_2(\Omega). \tag{2.33}$$

For an arbitrary trial function $\mathbf{v} \in \mathbf{V}$, the residual function is defined as:

$$\mathbf{R} = \mathbf{A}\mathbf{v} - \mathbf{f} \quad \text{in } \Omega. \tag{2.34}$$

If $\mathbf{v}$ is not equal to the exact solution $\mathbf{u}$, the residual $\mathbf{R}$ is not equal to zero. The squared distance between $\mathbf{A}\mathbf{v}$ and $\mathbf{f}$ will be nonnegative:

$$||\mathbf{R}||_0^2 = \int_\Omega (\mathbf{A}\mathbf{v} - \mathbf{f})^2 d\Omega \geq 0. \tag{2.35}$$

Hence a solution $\mathbf{u}$ to the problem (2.30) can be defined as a member of $\mathbf{V}$ that minimizes the squared distance between $\mathbf{A}\mathbf{v}$ and $\mathbf{f}$:

$$0 = ||\mathbf{R}(\mathbf{u})||_0^2 \leq ||\mathbf{R}(\mathbf{v})||_0^2 \quad \forall \mathbf{v} \in \mathbf{V}. \tag{2.36}$$

The least-square method tries to find the minimizer of the squared distance $||\mathbf{A}\mathbf{v} - \mathbf{f}||_0^2$ in $\mathbf{V}$. The quadratic functional in (2.35) can be written as:

$$I(\mathbf{v}) = ||\mathbf{A}\mathbf{v} - \mathbf{f}||_0^2 = (\mathbf{A}\mathbf{v} - \mathbf{f}, \mathbf{A}\mathbf{v} - \mathbf{f}). \tag{2.37}$$

For $\mathbf{u} \in \mathbf{V}$ to be a minimizer of the functional I in (2.37), a necessary condition is that its first variation vanishes at $\mathbf{u}$, that is:

$$\frac{d}{dt} I(\mathbf{u} + t\mathbf{v})\bigg|_{t=0} \equiv 2\int_\Omega (\mathbf{A}\mathbf{v})^T (\mathbf{A}\mathbf{u} - \mathbf{f}) d\Omega = 0 \quad \forall \mathbf{v} \in \mathbf{V}.$$

Thus, the least-square method is leading to the variational formulation:
Find $\mathbf{u} \in \mathbf{V}$ such that

$$B(\mathbf{u}, \mathbf{v}) = F(\mathbf{v}) \quad \forall \mathbf{v} \in \mathbf{V}, \tag{2.38}$$

where

$$B(\mathbf{u},\mathbf{v}) \equiv (\mathbf{A}\mathbf{u},\mathbf{A}\mathbf{v}),$$

$$F(\mathbf{v}) \equiv (\mathbf{f},\mathbf{A}\mathbf{v}).$$

The bilinear form in (2.38) is symmetric. For a well-posed[4] problem (2.30), the operator **A** is bounded below. Consequently, when discretized, (2.38) always lead to a symmetric positive-definite matrix.

In finite element analysis, first the domain is subdivided into a union of finite elements. Then an appropriate finite element basis is introduced. N_n denotes the number of nodes for one element and ψ_j denotes the elements shape functions. If for all unknown variables the same finite element is used, the expansion can be written in each element

$$\mathbf{u}_h^e(\mathbf{x}) = \sum_{j=1}^{N_n} \psi_j(\mathbf{x}) \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}_j , \tag{2.39}$$

where $(u_1, u_2, \cdots, u_m)$ are the nodal values at the jth node, and h denotes the mesh parameter.

By introducing the finite element approximation defined in (2.39) into the variational statement (2.38), the linear algebraic equations can be derived

$$\mathbf{K}\mathbf{U} = \mathbf{F}, \tag{2.40}$$

where the **U** is the global vector of nodal values. The global matrix **K** (stiffness matrix) is

[4]According to Hadamard (1902), a problem is well-posed if it has a solution, the solution is unique, and the solution depends continuously on data and parameters.

assembled from the element matrices

$$\mathbf{K}_e = \int_{\Omega_e} \left(\mathbf{A}\psi_1, \mathbf{A}\psi_2, \ldots, \mathbf{A}\psi_{N_n}\right)^T \left(\mathbf{A}\psi_1, \mathbf{A}\psi_2, \ldots, \mathbf{A}\psi_{N_n}\right) d\Omega, \tag{2.41}$$

in which $\Omega_e \subset \Omega$ is the domain of the eth element, and T denotes the transpose, and the vector $\mathbf{F}$ is assembled from the element vectors

$$\mathbf{F}_e = \int_{\Omega_e} \left(\mathbf{A}\psi_1, \mathbf{A}\psi_2, \ldots, \mathbf{A}\psi_{N_n}\right)^T \mathbf{f} d\Omega, \tag{2.42}$$

in (2.41) and (2.42)

$$\mathbf{A}\psi_j = \sum_{i=1}^{n_d} \frac{\partial \psi_j}{\partial x_i} \mathbf{A_i} + \psi_j \mathbf{A_0}. \tag{2.43}$$

2.3 The Gaussian Quadrature

In the LSFEM computation, Gaussian quadrature is used to compute the integrals. Equivalently, the following summation of weighted squared residuals is minimized:

$$I(\mathbf{v}_h) = \sum_{i=1}^{N_{elem}} \{ \sum_{l=1}^{N_{Gauss}} w_l \mathbf{R}^2(\zeta_l, \eta_l) |J(\zeta_l, \eta_l|\}, \tag{2.44}$$

in which, N_{Gauss} is the number of Gaussian points, w_l is the weighting factor in the Gaussian quadrature, and (ζ_l, η_l) is the location of Gaussian points in the master element, and J is the determinant of the Jacobian matrix of the coordinate transformation.

Besides a determined set of residual equations, LSFEM is also able to find a unique solution for an overdetermined set of equations with a specific weighting matrix in the least-squares sense. Different solutions can be obtained by using different weighting matrices.

Thus, in order for LSFEM to find an approximate solution, the set of residual equations must be either determined or overdetermined. This requirement can be expressed as:

$$N_{ele} \times N_{Gauss} \times N_{eq} \geq N_{node} \times m - N_{bc}, \tag{2.45}$$

where, N_{eq} is the total number of equations, N_{node} is the total number of nodes, m is the number of degrees of freedom at each node, N_{bc} is the total number of given nodal values at boundary nodes.

Satisfying requirement (2.45), which depends on the chosen number of Gaussian points, is essential in implementation of LSFEM. When the order of Gaussian quadrature is too low and this condition is not satisfied, the stiffness matrix is singular and consequently cannot be inverted by any direct solver. Also, the iterative solution does not converge in this case. On the other hand, if the order of Gaussian quadrature is too high, the solution will be inaccurate. In this case, the system will be extremely overdetermined which means that too many residual equations is forced to be zero at Gaussian points with too few adjustable unknowns. Another consequence of requirement (2.45) is that in general the linear triangular and linear tetrahedral elements are not suitable for LSFEM.

2.4 Incompressible Inviscid Flows

Incompressible inviscid flows are governed by the incompressible Euler equations. Although, the physical model from which the Euler equations are derived is simple, numerical solution of them is formidable. As stated before, the Galerkin finite element method method and the central difference method lead to non-self-adjointness and generate oscillatory solution.

In this section, first the Euler equation for an inviscid, incompressible and homogeneous flow, which is irrotational, is presented. Then, a splitting method is used to construct the solution of the equations in time domain.

2.4.1 Incompressible Euler Equations

Consider fluid flow in a fixed bounded domain Ω with a piecewise smooth boundary Γ in the presence of conservative external forces $\mathbf{f}$ acting on the fluid. The flow of an incompressible inviscid fluid is governed by the incompressibility equation:

$$\nabla \cdot \mathbf{u} = 0 \quad \text{in } \Omega \times (0,T], \tag{2.46}$$

and the momentum equation:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \nabla p = \mathbf{f} \quad \text{in } \Omega \times (0,T], \tag{2.47}$$

where $\mathbf{u}(\mathbf{x},t)$ is the velocity vector, $p(\mathbf{x},t)$ is the pressure divided by the constant density of the fluid.

To complete the statement of the problem, suitable boundary and initial conditions need to be prescribed. A common boundary condition is specifying the normal component of velocity:

$$\mathbf{n} \cdot \mathbf{u} = g \quad \text{on } \Gamma \times (0,T], \tag{2.48}$$

where g is a given function of the location on the boundary at the time t. g should satisfy the global conservation condition:

$$\int_{\Gamma} g d\Gamma = 0 \quad \text{in } (0,T]. \tag{2.49}$$

The Initial condition specifies the velocity field at $t = 0$:

$$\mathbf{u}(\mathbf{x},t) = \mathbf{u}_0(\mathbf{x}) \quad \text{in } \Omega \tag{2.50}$$

Also, the initial velocity should be divergence-free:

$$\nabla \cdot \mathbf{u}_0 = 0 \quad \text{in } \Omega \tag{2.51}$$

Lastly, the boundary and initial data should be compatible:

$$g|_{t=0} = \mathbf{n} \cdot \mathbf{u}_0 \tag{2.52}$$

2.4.2 Least-Square Finite Element Formulation of Euler Equations

To approximate the solution, LSFEM can be applied to Euler equations. For illustration, the LSFEM formulation of the Euler equations in two dimensions is presented.

The time-dependent incompressible Euler equations in two dimensional Cartesian coordinates can be written as:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \quad \text{in } \Omega \times (0,T], \tag{2.53}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} = f_x \quad \text{in } \Omega \times (0,T], \tag{2.54}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial x} = f_y \quad \text{in } \Omega \times (0,T], \tag{2.55}$$

The equations can be easily discretized by using finite difference method, for instance, the θ method.

With time Step $\Delta t = t^{n+1} - t^n$, by knowing $(\mathbf{u}, p)^n$ for the previous time step, the solution $(\mathbf{u}, p)^{n+1}$ for the current time step is determined from

$$\frac{\partial u^{n+1}}{\partial x} + \frac{\partial v^{n+1}}{\partial y} = 0 \quad \text{in } \Omega, \tag{2.56}$$

$$\begin{aligned}\frac{u^{n+1} - u^n}{\Delta t} &+ \theta\left(u^{n+1}\frac{\partial u^{n+1}}{\partial x} + v^{n+1}\frac{\partial u^{n+1}}{\partial y} + \frac{\partial p^{n+1}}{\partial x}\right)\\ &+ (1-\theta)\left(u^n\frac{\partial u^n}{\partial x} + v^n\frac{\partial u^n}{\partial y} + \frac{\partial p^n}{\partial x}\right) = f_x^{n+1} \quad \text{in } \Omega,\end{aligned} \tag{2.57}$$

$$\begin{aligned}\frac{v^{n+1} - v^n}{\Delta t} &+ \theta\left(u^{n+1}\frac{\partial v^{n+1}}{\partial x} + v^{n+1}\frac{\partial v^{n+1}}{\partial y} + \frac{\partial p^{n+1}}{\partial y}\right)\\ &+ (1-\theta)\left(u^n\frac{\partial v^n}{\partial x} + v^n\frac{\partial v^n}{\partial y} + \frac{\partial p^n}{\partial y}\right) = f_y^{n+1} \quad \text{in } \Omega,\end{aligned} \tag{2.58}$$

where the superscript 'n' denotes the previous time-step and 'n+1' the current one.

Usually θ can be either 1/2 or 1. $\theta = 1/2$ represents the Crank-Nicolson scheme and gives second-order accuracy $O(\Delta t^2)$, and is used for true transient problems; and $\theta = 1$ corresponds to the backward-Euler scheme and provides first-order accuracy $O(\Delta t)$, and is mainly used for a time-marching approach to seek steady-state solutions. Both schemes are unconditionally stable. Hence they do not impose any limitation on the size of the time step.

The nonlinear terms in (2.56)-(2.58) can be linearized by Newton's method. In order to guarantee time-accuracy at each time-step, Newton's linearization is used until convergence is reached. For instacne, $u^{n+1}\partial u^{n+1}/\partial x$ is approximated as:

$$\begin{aligned}(u^{n+1})_{[k+1]}\frac{\partial (u^{n+1})_{[k+1]}}{\partial x} &\approx (u^{n+1})_{[k+1]}\frac{\partial (u^{n+1})_{[k+1]}}{\partial x}\\ &+ (u^{n+1})_{[k+1]}\frac{\partial (u^{n+1})_{[k]}}{\partial x} - (u^{n+1})_{[k]}\frac{\partial (u^{n+1})_{[k]}}{\partial x},\end{aligned} \tag{2.59}$$

where k denotes the iteration count of the linearization and $(u^{n+1})_{[0]} = u^n$.

After linearization, the system of equations (2.56)-(2.58) can be written in standard matrix form as:

$$\mathbf{A}_1 \frac{\partial \underline{\mathbf{u}}}{\partial x} + \mathbf{A}_2 \frac{\partial \underline{\mathbf{u}}}{\partial y} + \mathbf{A}_0 \underline{\mathbf{u}} = \underline{\mathbf{f}} \tag{2.60}$$

where

$$\underline{\mathbf{u}} = \begin{pmatrix} u \\ v \\ p \end{pmatrix}_{[k+1]}^{n+1} , \quad \mathbf{A}_0 = \begin{pmatrix} 0 & 0 & 0 \\ \frac{1}{\Delta t} + \theta \frac{\partial u}{\partial x} & \theta \frac{\partial u}{\partial y} & 0 \\ \theta \frac{\partial v}{\partial x} & \frac{1}{\Delta t} + \theta \frac{\partial v}{\partial y} & 0 \end{pmatrix}_{[k]}^{n+1} ,$$

$$\mathbf{A}_1 = \begin{pmatrix} 1 & 0 & 0 \\ \theta u & 0 & 1 \\ 0 & \theta u & 0 \end{pmatrix}_{[k]}^{n+1} , \quad \mathbf{A}_2 = \begin{pmatrix} 0 & 1 & 0 \\ \theta v & 0 & 0 \\ 0 & \theta v & 1 \end{pmatrix}_{[k]}^{n+1} ,$$

$$\underline{\mathbf{f}} = \begin{pmatrix} 0 \\ f_x^{n+1} + \frac{u^n}{\Delta t} - (1-\theta)(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x})^n + \theta(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y})_{[k]}^{n+1} \\ f_y^{n+1} + \frac{v^n}{\Delta t} - (1-\theta)(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y})^n + \theta(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y})_{[k]}^{n+1} \end{pmatrix} .$$

Finally, the first-order system (2.60) can be applied to a meshed geometry and the velocity and pressure can be calculated using the determined (2.38).

Chapter 3

Introduction to the Parallel Computing with OpenMP

In this thesis, an LSFEM formulation of the elliptical instability problem in a rotating ellipsoid is numerically solved. Due to the size of the problem, we have implemented the OpenMP programming model in order to reduce the wall clock time and use the available resources, such as multi-cores computers and non-local resources, to maximum capacity. In this chapter an introduction to parallel programming models is presented.

3.1 Parallel Computing

Traditionally, most of the programs have been written for ***serial*** computation. In serial computing, software run on a single processor. A problem is broken into a discrete series of instructions which are executed one after the other, and only one instruction is executed at any moment in time.

On the other hand, in ***parallel*** computing, multiple processing units can be used simultaneously to solve a problem. In parallel computing, a problem is broken into independent parts which can be assigned to different processors to execute simultaneously. Parallel computing can be implemented in a single computer with multiple processors, a number computers connected by a network, or a combination of the two.

Parallel computing has been around for a few decades; however, thanks to recent development of hardware technology which has provided the average consumer with multi-core and multi-processor computers at a reasonable price, interest in parallel computing has grown outside of the high-performance computing community.

The main reasons to use parallel computing are to save time and/or money, and to be able to solve larger problems. In serial computing, in order to decrease the run time of a program without changing the code, the speed of the processor needs to be increased. However, there are some limitations on how fast a processor can be build, both physical and economical. There is a limit on how fast data can be move through hardware which pose a constraint on the speed of the processor. Moreover, it is significantly more expensive to make a single processor faster than using multiple fast processors. Also, due to the size and complexity of many problems and the limits of computer memory, it is impossible or impractical to solve these problem on a single computer.

Furthermore, parallelization has enabled the computing community to use concurrent non-local resources. A serial computer can only execute one instruction at a time. Multiple computing resources can execute many programs simultaneously. This feature of parallel computing has motivated many organizations to build computer clusters. Multiple computers can be connected through a local area network (LAN) and united as a single logical unit. The networked computers act as a single, much more powerful machine. Clusters offer much better performance, larger storage capacity, and wider availability of resources over that of a single computer. The user can add or remove computer resources to meet the size and time requirements for his workloads. Furthermore, using a wide area network or the Internet, computer jobs can be launched at anytime and from anywhere.

3.2 Limits and Costs of Parallel Computing

Ideally, the increase in speed from parallelization would be linear; doubling the number the processors should halve the wall clock time, and doubling it again should halve the wall clock time for a second time, and so on and so forth. However; in reality very few algorithms can reach the optimal speed-up. Almost every program has some sections which need to be executed in series. Amdahl (1967) determined the potential speed-up of a parallel program to be

$$SU = \frac{1}{1-P}, \tag{3.1}$$

where SU is the speed-up and P is the fraction of the program that can be parallelized. Thus, if none of the code can be parallelized, serial code, the speed-up would be zero; and if all of the code can be parallelized, the speed-up would be infinity (in theory). By introducing the number of processors executing the parallel fraction of the code, (3.1) can be modified as follows:

$$SU = \frac{1}{\frac{P}{N}+S} \tag{3.2}$$

in which, N is the number of processors. The effects of the Amdahl's law are shown in figure 3.1. Significant speed-up cannot always be achieved by using more processors to run a code.

Parallel programs are generally more complex than serial programs. Besides the execution of multiple instructions at the same time, most of the time, data flows between the instruction as well. Complexity of parallelized program is present in every aspect of the program development cycle, such as design, coding, debugging, tuning, and maintenance.

Due to standardization of several Application Programming Interfaces (APIs), such as MPI[5], POSIX threads, and OpenMP, parallel programs have become more portable than before. However, all of the portability issues associated with serial programs, like vendor

[5]Message Passing Interface

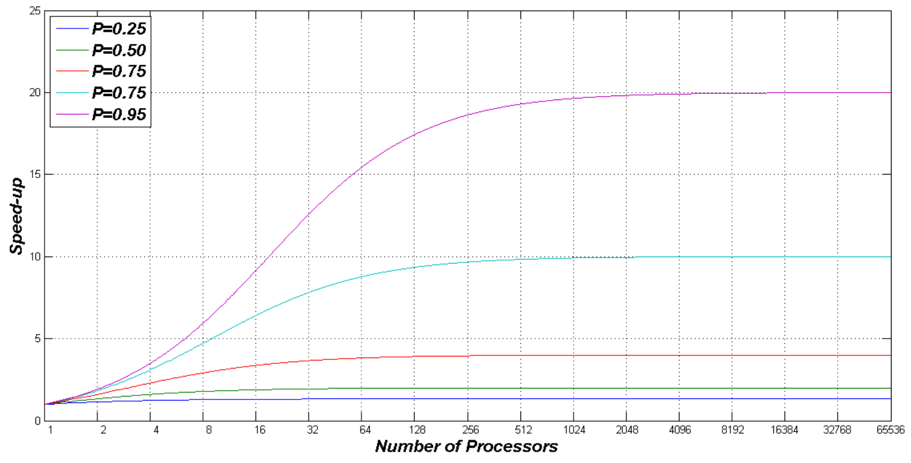


Figure 3.1: Scalability of parallelization.

enhancements, apply to parallel programs. Furthermore, operating systems and hardware architectures play key roles in portability of parallel codes.

Another issue with parallel programs is its demand for more resources. Considering the need to replicate data and for overheads associated with parallel support libraries and subsystems, the required amount of memory can be greater for parallel programs.

Also, in reality the Amdahl's law (3.2) is not correct for a large number of processors. Parallelization beyond a certain point causes the program to run slower. This phenomenon is generally due to communication bottleneck. As more processing nodes are added, each processing node spends progressively more time doing communication than useful processing. At some point, the communications overhead created by adding another processing node surpasses the increased processing power that node provides, and parallel slowdown occurs. Parallel slowdown is more serious in short parallel programs.

3.3 Parallel Computer Memory Architectures

3.3.1 Shared Memory

Shared memory computers (SMPs) refer to computers with multiprocessing design in which all processors have access to all memory[6] as global address space. In SMPs, while sharing the same memory resources, multiple processors can operate independently. Consequently, changes in a memory location by a processor are visible to all processors.

Originally, the term SMP was used to refer to a symmetric multiprocessor system design. In a symmetric design, all processors have access to any memory location at the same speed. Figure 3.2 shows an example of such design. This architecture is also called *uniform memory access* (UMA) or sometimes *cache-coherent uniform memory access* (cc-UMA). Cache coherency, achieved at the hardware level, means that if one processor updates a location in shared memory, all the other processors know about the update.

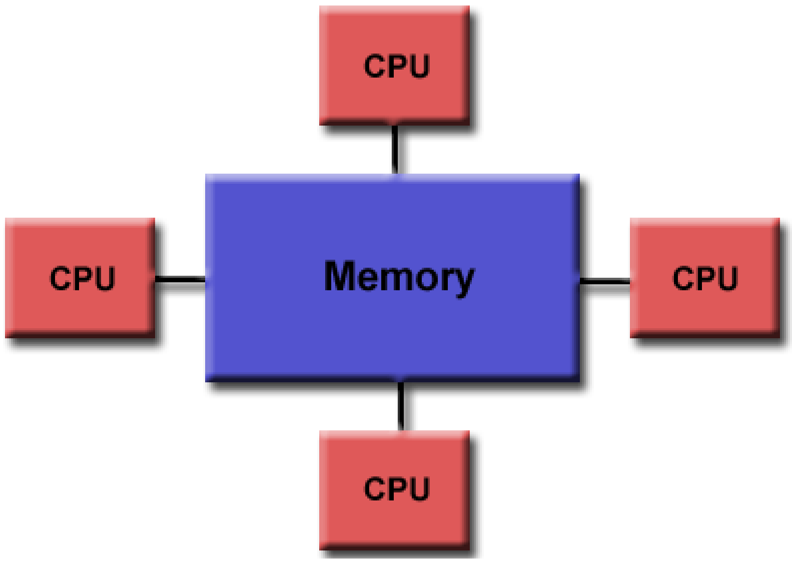


Figure 3.2: An example of symmetric shared memory configuration.

Due to evergrowing need for larger memory, another architecture has been designed for multiprocessing computer called *non-uniform memory access* (*NUMA*). In NUMA designs,

[6]In computer hardware, shared memory refers to a (typically large) block of random access memory (RAM) that can be accessed by several different central processing units (CPUs) in a multiple-processor computer system.

different processes have different access time to memory locations depending on their distance from the memory; the nearer a processor to the memory, the faster it can access the memory. A NUMA system is often made by linking two or more symmetric multiprocessor machines. If cache coherency is maintained in a NUMA design, it may also be called *cache-coherent non-uniform memory access* (*cc-NUMA*). An example of NUMA architecture is given in figure 3.3.

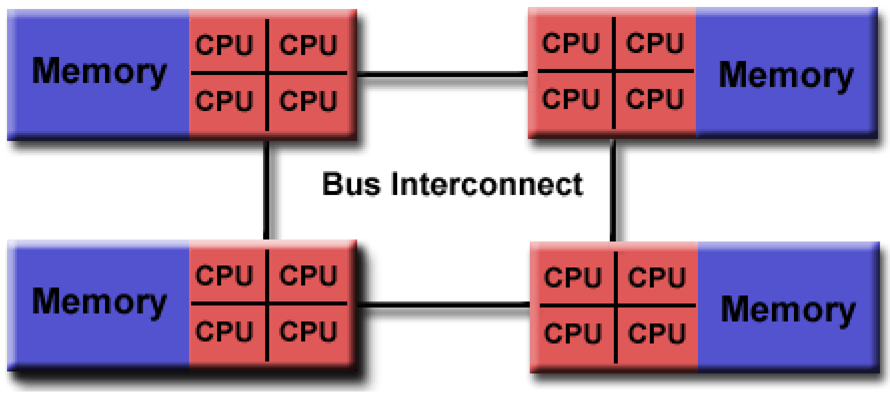


Figure 3.3: An example of *non-uniform memory access* configuration.

Although SMPs provide a user-friendly programming perspective to memory and a fast and uniform way for data sharing between tasks due to the proximity of memory to CPUs, their main disadvantage is the lack of scalability between memory and CPUs. While adding more processors to the system does not increase the amount of memory, it increases traffic on the shared memory-CPU path and causes a bottleneck which will slow down a program. Also, writing to the global memory should be handled carefully. Incorrect results can be produced when multiple processors tries to read and write on a memory location simultaneously. This is known as a *race condition*.

3.3.2 Distributed Memory

Distributed memory architecture refers to a multiprocessing system in which each processor has its own private memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.

In this design, each processor can only operate on its local data and changes to the local memory do not affect the memory of other processor. Consequently, the concept of cache coherency does not apply. If a processor needs remote data, it is the programmer task to define how and when data is communicated. Furthermore, programmer is responsible for synchronization between tasks. An example of distributed memory architecture is given in figure 3.4.

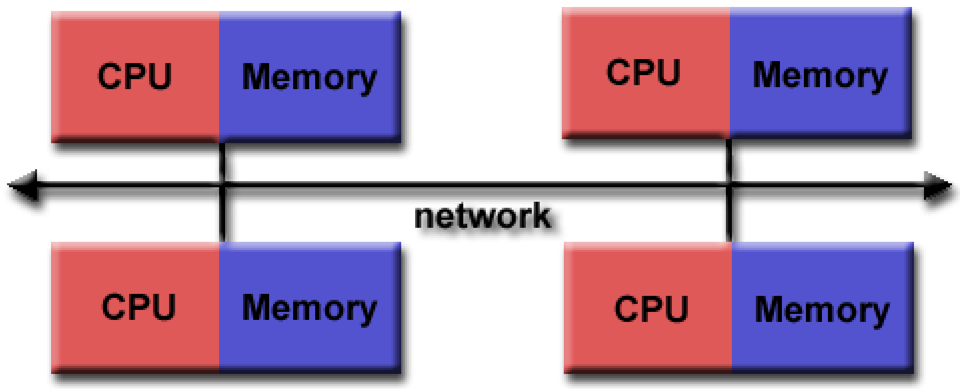


Figure 3.4: An example of distributed memory architecture.

While distributed memory architecture provides scalability between memory and CPUs and a significant decrease in the cost, it makes programming of a program much more complex. The programmer is responsible for many of the details associated with data communication between processors. Moreover, since the memory access is not uniform anymore, it takes longer to access data in a remote memory than a local one.

3.3.3 Hybrid Distributed-Shared Memory

Each of shared and distributed memory architectures offers some advantages but at a cost. Hence, computer vendors have designed a hybrid architecture that offers the advantages of both shared and distributed memory systems. In this hybrid design, shared memory components are linked together. The distributed memory component is the networking of multiple shared memory units. Like distributed memory architecture, since the shared memory components have direct access to their own memory, network communication is required to move data from one unit to another. The fastest and largest computers in the world use this hybrid architecture and the current trends suggest that this type of memory architecture will continue to dominate the high end of computing for the foreseeable future. An example of a hybrid design is given in figure 3.5.

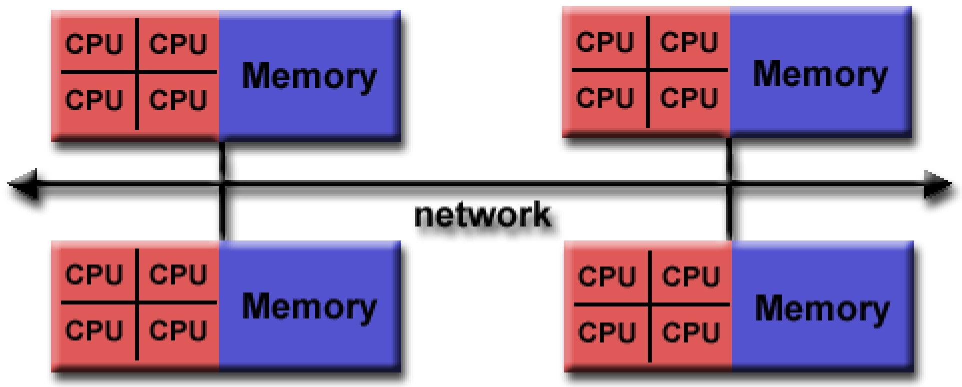


Figure 3.5: An example of hybrid memory architecture.

3.4 Cache Memory

The growing discrepancy in processors and memory speed has forced vendors to build computer systems with hierarchical memory systems. In other words, processors are getting faster and faster while the speed to access and write data to memories has not increased at the same rate. Thus, vendors have provided the processors with small, expensive, and very fast memories called cache memory, or “cache” for short. Each cache is private to its processor and feed them with data very quickly. Hence, even in SMPs, not all of memory is shared.

Once the processor wants to do a computation, data is copied into cache from main memory. Cache memory is very small compared to main memory; therefore, a new block of data may supersede data previously copied in. If data required by the processor is not available in cache, there will be a delay while they are retrieved from main memory. The results of the computations in the cache stay there until the program updates the main memory with these new data.

The consequence of cache memory in SMPs is that since a processor stores the results of local computation in its own private cache, the new values are only accessible to code executing on that processor. Even after copying the new data to main memory, some processors may still have a copy of old values in their cache and continue to use them.

There are a number of strategies developed to overcome the *memory inconsistency problem*. A *cache coherent* system ensures that updates to data made by one processor are known to the program running on other processors. The programmer is responsible to follow standards of the model to ensure the cache coherency of a program.

3.5 Parallel Programming Models

Just like there are different architectures of parallel hardware, there are different models of parallel programming. OpenMP is an API that support shared-memory multi-processing programming. In OpenMP model, the program is executed on one or more processors which share the some or all of the available memory. In shared-memory approach, the program executed by multiple independent threads[7]. Besides the shared data, threads may also have additional private data. Shared-memory programming model should provide the user with means for starting up threads, assigning work to them, coordinating their access to shared data, and the ability to ensure that certain operations are executed by only one thread at a time (Chapman et al., 2008).

Several different models, generally referred as “message passing”, have been developed for distributed memory architecture. In a message passing model, the program is executed by one or more processors with their own memory. If a processor needs the results of another processor, the data will be sent through the network as messages. A message passing model should provide means to initiate and manage the participating processors, as well as operations for sending and receiving messages.

Parallel programming APIs differ in the features provided, approach and complexity of their implementation. While some of the models are a collection of library routines with which the programmer may specify some or all of the details of parallel execution (e.g. GA (Nieplocha et al., 1996) and Pthreads (Lewis et al., 1998) for shared-memory programming and MPI (Gropp et al., 1999) for MPPs[8]), others such as OpenMP (Chapman et al., 2008) and HPF (Koelbel et al., 1994) provide the complier with additional information which utilize them to generate the parallel code.

[7]“execution states that are able to process an instruction stream”; Chapman et al. (2008, chap. 1)

[8]Massively Parallel Processors

3.6 OpenMP Programming Model

OpenMP was first published by the OpenMP Architecture Review Board (ARV) in 1997 in order to establish a standardized API for writing parallel programs in C, C++ and FORTRAN executable on shared-memory systems with UMA or NUMA architecture. OpenMP provides programmer with a simple set of directives for writing parallel codes on shared-memory systems. Unlike message passing libraries which require the whole program to be parallelized, OpenMP allows incremental parallelization.

However, OpenMP has its own restrictions, too. A code parallelized only by OpenMP is not useful for distributed memory systems. Also, OpenMP does not guarantee to make the most efficient use of shared memory. OpenMP does not provide the complier with a means to check for data dependencies, data conflicts, race conditions, or deadlocks.

Parallelization using OpenMP can be as simple as inserting compiler derivatives in a serial program or as sophisticated as inserting subroutines to set multiple levels of parallelism, locks and even nested locks.

OpenMP execute parallel code using a fork-join model (figure 3.6). The program starts with as a serial program with a single process, the *master thread*. As soon as a parallel region is encountered, the master thread creates a team of parallel threads. Then, the program segment inside the parallel region is executed in parallel among the various threads. Once the team threads finish the statement in parallel region, they synchronize and terminate, leaving only the master thread. The number of the threads that execute a parallel region depends on the availability of resources and the programmers decision. Also, in order to offer more efficient use of resources, OpenMP API provides the programmer with an environment variable to dynamically change the number of threads used to execute parallel regions.

In summary, OpenMP equips programmers with simple yet effective means to develop parallelized program executable on shared-memory systems. Although, the level of parallelization of OpenMP may not be enough to execute very long and complex programs

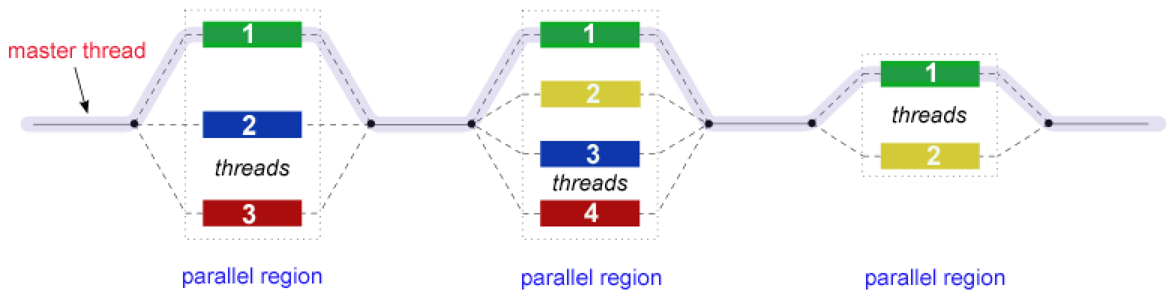


Figure 3.6: The Fork and Join Model.

efficiently, it provides the computing community with a powerful tool to handle medium-sized problems and utilize common resources as much as possible.

Chapter 4

Formulation and Results

4.1 Numerical Model

In this study, a triaxial ellipsoid containing an incompressible inviscid fluid is considered. The ellipsoid is rotating at a constant angular velocity Ω about its axis (Oz). Initially, we assume that there is no background rotation; therefore, the geometry is preserved in the inertial frame. The outer boundary of the ellipsoid is given by

$$\left(\frac{x'}{a}\right)^2 + \left(\frac{y'}{b}\right)^2 + \left(\frac{z'}{c}\right)^2 = 1 \tag{4.1}$$

where a, b, and c are the radii of the ellipsoid along the x-axis, y-axis, and z-axis respectively (see figure 4.1). Note that throughout this chapter, primed symbols are used for dimensional quantities and unprimed symbols are used for their dimensionless counterparts..

Instead of the radius of the ellipsoid along each axis, the following parameters are used to define the geometry

$$R = \sqrt{\frac{a^2+b^2}{2}}, \quad \varepsilon = \frac{a^2-b^2}{a^2+b^2}, \quad \text{and} \quad \zeta = \frac{c}{R}$$

where R is the mean equatorial radius, ε is the equatorial ellipticity, and ζ is the flattening of the ellipsoid.

With respect to the inertial reference frame, the initial velocity and pressure (the solid body rotation) satisfy

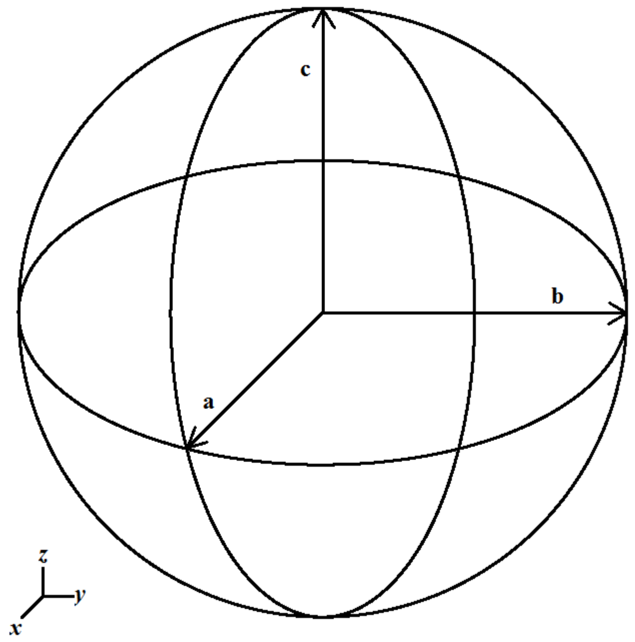


Figure 4.1: Sketch of the boundaries of the problem under consideration.

$$\mathbf{u}_0' = \Omega\left(-\frac{a}{b}y'\hat{\boldsymbol{i}} + \frac{b}{a}x'\hat{\boldsymbol{j}}\right), \tag{4.2a}$$

$$p_0' = \frac{1}{2}\rho\Omega^2\left(x'^2 + y'^2\right) \tag{4.2b}$$

where $\left(\hat{\boldsymbol{i}}, \hat{\boldsymbol{j}}, \hat{\boldsymbol{k}}\right)$ are the unit vectors associated with the cartesian coordinates (x', y', z'). The initial velocity (4.2a) and pressure (4.2b) are the solution of the momentum and continuity equations

$$\mathbf{u}_0' \cdot \nabla'\mathbf{u}_0' = -\frac{1}{\rho}\nabla' p_0', \tag{4.3a}$$

$$\nabla' \cdot \mathbf{u}_0' = 0 \tag{4.3b}$$

and satisfy the boundary condition

$$\hat{\mathbf{n}} \cdot \mathbf{u}'_0 = 0 \quad \text{on } S' \tag{4.4}$$

in which S' is the ellipsoid surface (Seyed-Mahmoud, 1999).

The dynamical equations describing the motion of the fluid ellipsoid are then defined by the following system of equations:

$$\frac{\partial \mathbf{u}'}{\partial t'} + \mathbf{u}' \cdot \nabla' \mathbf{u}' + \frac{1}{\rho} \nabla' p' = 0, \tag{4.5a}$$

$$\nabla' \cdot \mathbf{u}' = 0, \tag{4.5b}$$

$$\hat{\mathbf{n}} \cdot \mathbf{u}' = 0 \quad \text{on } S' \tag{4.5c}$$

where $\mathbf{u}'$, t', p', and ρ denote the velocity, time, reduced pressure, and density. Equations (4.5a) and (4.5b) are those of conservation laws for momentum and mass respectively and equation (4.5c) is the impermeability boundary condition.

Applying the following dimensionless quantities, the ellipsoidal boundary is mapped into a sphere and elliptical streamlines are mapped into circular ones:

$$x = \frac{x'}{a}, y = \frac{y'}{b}, z = \frac{z'}{c}, u = \frac{u'}{a\Omega}, v = \frac{v'}{b\Omega}, w = \frac{w'}{c\Omega}, t = t'\Omega, p = \frac{p'}{\rho\Omega^2 R^2} \tag{4.6}$$

where (u, v, w) are the components of the displacement vector in x, y and z directions respectively. In the new coordinate system, the equations (4.5a)-(4.5c) become:

$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}+w\frac{\partial u}{\partial z}+\frac{1}{1+\varepsilon}\frac{\partial p}{\partial x}=0, \tag{4.7a}$$

$$\frac{\partial v}{\partial t}+u\frac{\partial v}{\partial x}+v\frac{\partial v}{\partial y}+w\frac{\partial v}{\partial z}+\frac{1}{1-\varepsilon}\frac{\partial p}{\partial y}=0, \tag{4.7b}$$

$$\frac{\partial w}{\partial t}+u\frac{\partial w}{\partial x}+v\frac{\partial w}{\partial y}+w\frac{\partial w}{\partial z}+\frac{1}{\zeta^2}\frac{\partial p}{\partial z}=0, \tag{4.7c}$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}+\frac{\partial w}{\partial z}=0, \tag{4.7d}$$

$$\hat{\mathbf{n}}\cdot\mathbf{u}=0 \quad \text{on } S \tag{4.7e}$$

where S is the surface of the unit sphere.

In this research, formulation (2.38) is implemented in a code written in FORTRAN (Appendix B), parallelized using the OpenMP library, in order to solve system of equation (4.7a)-(4.10e). The time domain is discretized using Crank-Nicolson method which is unconditionally stable and provides second-order accuracy $O\left(\Delta t^2\right)$. Also, the nonlinear terms are linearized by Newton's method. In order to guarantee time-accuracy at each time step, Newton's linearization is used until convergence is reached. The geometry is meshed with quadratic tetrahedron elements using commercial software, ABAQUS; and the integrals are calculated using the second order Gaussian Quadrature Method. Finally, since the least-square finite element method leads to symmetric and positive-definite system of algebraic equation, Conjugate Gradient Method (see Appendix A) is applied to solve for the velocity and pressure fields at each iteration.

One of the key parameters in studying elliptical instability is the growth rate of the instability (σ). It is well established that for the inviscid incompressible fluid contained in an ellipsoid with $\zeta = 1$, the non-dimensional growth rate in the linear refime is equal to the half of the ellipticity value (Gledzer and Ponomarev, 1992; Kerswell, 1994; Seyed-Mahmoud et al., 2000; Lacaze et al., 2004). In order to calculate the growth rate and investigate the evolution of the instability, a norm is needed. Considering that due to the elliptical instability, perturbations in all velocity components and the pressure are induced, the average of the absolute of the perturbations in the whole domain, $U = 1/V \iiint\limits_V |u - u_0| d\tau$, $W = 1/V \iiint\limits_V |w| d\tau$, $P = 1/V \iiint\limits_V |p - p_0| d\tau$, are considered as the norms. Note that the base flow has no component in the z-direction, so there is only perturbation in that direction. The growth rates of the instability calculated from our approach which are consistent with previous studies, are given in table 4.1 for an ellipsoid with $\zeta = 1$ and different ellipticities.

Table 4.1: Growth rate of the instability of an ellipsoid with $\zeta = 1$ and different ellipticities.

ε	σ
0.010	0.005
0.025	0.013
0.050	0.025
0.075	0.038
0.100	0.050
0.150	0.075

Lacaze et al. (2004) showed that the nonlinear spin-over instability of fluid contained in an ellipsoid with $\zeta = 1$ is a heteroclinic[9] oscillation. Hence, after reaching the peak, W decays to zero and the z component of the velocity changes direction. The first cycle of the perturbations, plotted using our approach, is given in figure 4.2 for an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 1$.

As it is shown in figure 4.2, in one cycle, while norms of the x component and y component of the velocity (U and V) go through one cycle of growth and decay, the norm of the z component of the velocity (W) and the norm of the pressure (P) go through two cycles of the same. However, there is an important difference between the two cycles involving W. In the second cycle, the perturbed flow is growing in the opposite direction of the first one. Although the nonlinear terms have no noticeable effects on the exponential growth while the amplitudes of perturbations are small, at some point the nonlinear terms become

[9]A heteroclinic cycle is a collection of solution trajectories that connects sequences of equilibria, periodic solutions or chaotic invariant sets (Palacios, 2007).

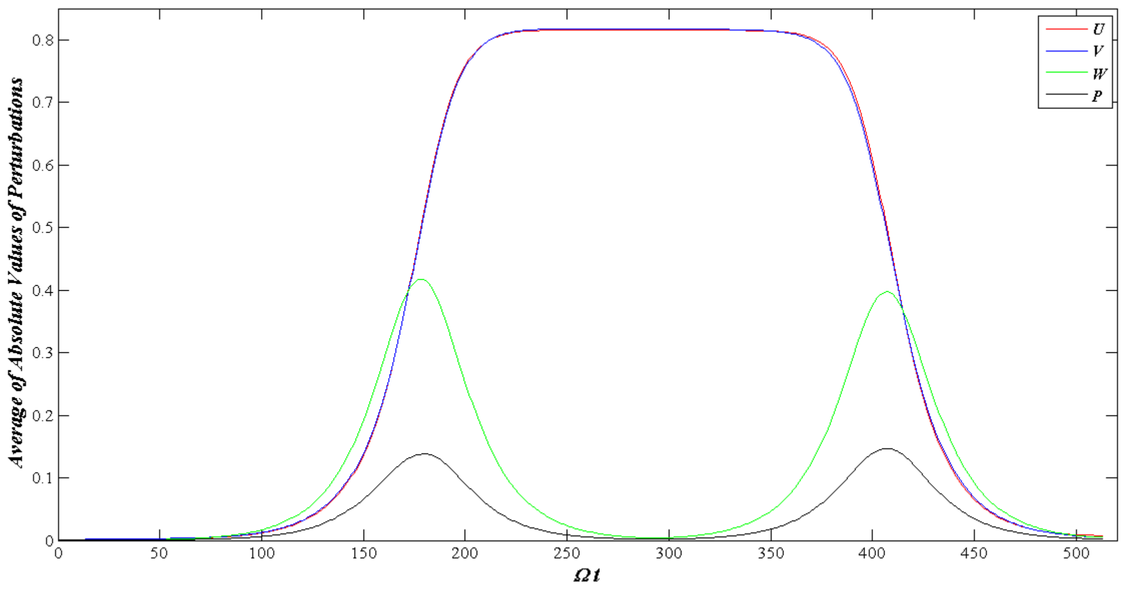
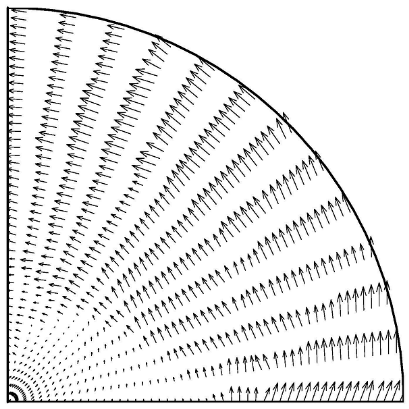


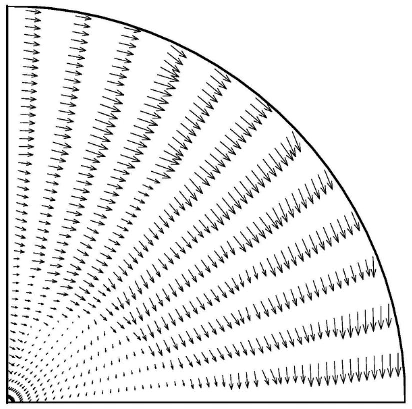
Figure 4.2: Average of absolute values of dimensionless perturbations for an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 1$.

dominant and change the sign of the curvatures. Furthermore, figure 4.2 shows that once W and P reach their peaks, the signs of curvature of U and V change. In other words, the local maximums of W and P and the inflection points of U and V happen around the same time. Also, while the magnitude of velocity in the z-direction decays and grows again in the opposite direction, U and V stay in a saturation state and only start to decay once the nonlinear terms become dominant again in the z-direction, the third inflection points of W.

The velocities of the flow in different planes at certain times are given in figures 4.3-4.5. The fluid body starts with a two dimensional flow, at solid body rotation. At some point, U and V which grow in the opposite direction of the basic flow, become greater than the basic flow and change the direction of rotation of the fluid inside the container in the equatorial plane. U and V start to decay around the time of the third inflection point of W and the flow goes back to the initial basic flow at the end of the cycle. The shapes of the flow in the y-z plane are identical as the (2, 1, 1) inertial mode (figure 4.6) which suggest that this instability is due to the excitation of the (2, 1, 1) and (2, 1, -1) modes, also known as the spin-over instability.

Another parameter that might be of interest is the period of the instability cycle. The


(a) The basic flow ($\Omega t = 0$).


(b) At the saturation state ($\Omega t = 293$).

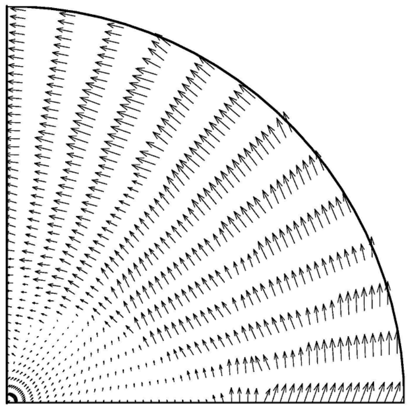
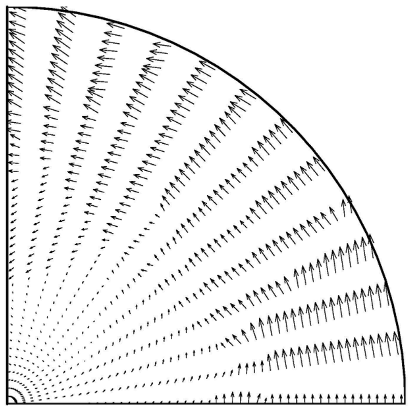

(c) At the end of the cycle ($\Omega t = 514$).

Figure 4.3: The flows in the x-y plane of an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 1$.


(a) At the first maximum of W ($\Omega t = 178$).

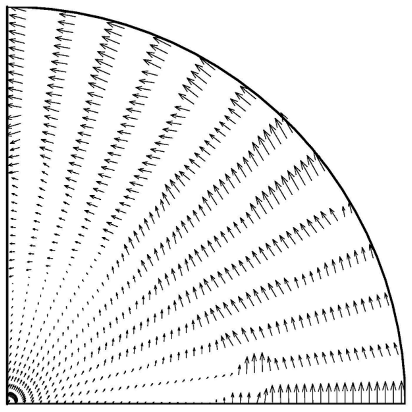
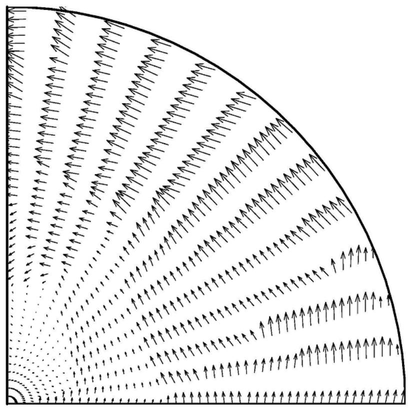

(b) At the second maximum of W ($\Omega t = 407$).

Figure 4.4: The flows in the x-z plane of an ellipsoid with $\varepsilon = 0.1$. and $\zeta = 1$


(a) At the first maximum of W ($\Omega t = 178$).

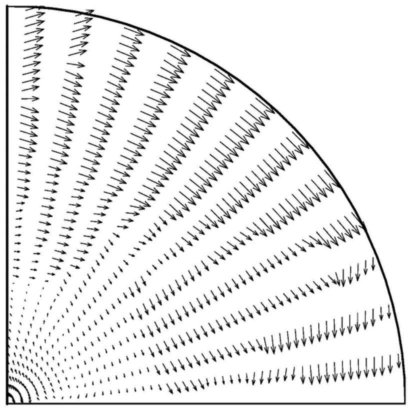

(b) At the second maximum of W ($\Omega t = 407$).

Figure 4.5: The flows in the y-z plane of an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 1$

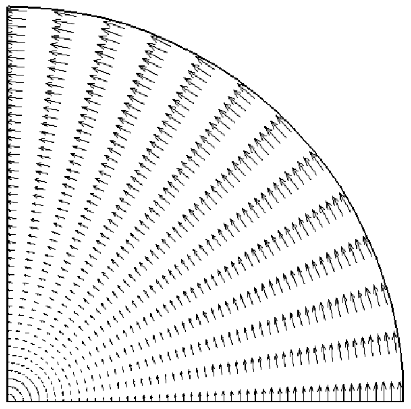


Figure 4.6: The flow pattern of the (2, 1, 1) mode.

period of the spin over instability for an ellipsoid with $\zeta = 1$ as a function of its ellipticity is given in figure 4.7. Considering the approximation error, it is safe to assume that the period of the instability is directly related to the inverse of the ellipticity.

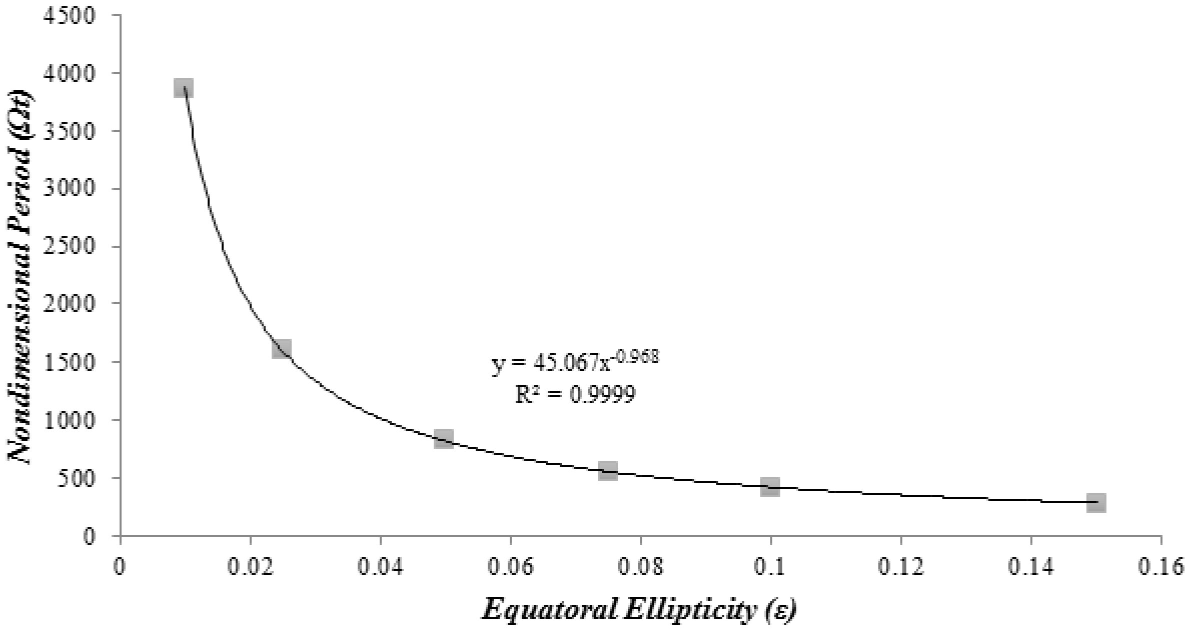


Figure 4.7: The period of the instability of a rotating contained fluid ellipsoid with $\zeta = 1$ as a function of its ellipticity

4.2 Elliptical Instability of a Flattened Ellipsoid

Any rotating fluid body in hydrostatic equilibrium is flattened along the rotation axis. For instance, according to the WGS 84 standard, the reciprocal of the flattening of the Earth is equal to 298.257223563. Kerswell (1994) calculated the first 60 subharmonic resonances and their growth rates for an ellipsoid with different flattenings. Seyed-Mahmoud et al. (2000) investigated the instability in ellipsoidal shells in linear regime and were able to determine the growth rate of the instability and the interacting modes.

According to Gledzer and Ponomarev (1992), the growth rate of the spin-over instability in an ellipsoid with the equatorial ellipticity ε and the flattening ζ is:

$$\sigma = \sqrt{\frac{(1+\varepsilon-\zeta^2)(-1+\varepsilon+\zeta^2)}{(1+\varepsilon+\zeta^2)(1-\varepsilon+\zeta^2)}} \tag{4.8}$$

Hence, in the limit of $\sqrt{(1-\varepsilon)} < \zeta < \sqrt{(1+\varepsilon)}$ the spin-over instability would be excited. Note that for an ellipsoid without flattening ($\zeta = 1$) with small ellipticity the expression $\sigma = \varepsilon/2$ is recovered. Our numerical results are in complete agreement with equation (4.8) (see table 4.2).

Table 4.2: Growth rate of the spin-over instability computed for an ellipsoid with equatorial ellipticity $\varepsilon = 0.1$ for different flattenings, ζ (column 1), using equation (4.8) (column 2) and our approach (column 3).

Flattening ζ	Growth rate from equation (4.8)	Growth rate from this work
0.95	0.012	0.012
0.96	0.032	0.032
0.97	0.042	0.041
0.98	0.047	0.046
0.99	0.050	0.049
1.00	0.050	0.050
1.01	0.049	0.049
1.02	0.045	0.045
1.03	0.039	0.038
1.04	0.028	0.028
1.047	0.013	0.013

Cébron et al. (2010a) investigated the instability of a rotating incompressible viscous fluid body for flattenings outside of the spin-over range and found that other modes may interact to excite the instability. The growth rate of the instability decreases significantly once the flow is in the nonlinear regime. Therefore, if a fluid of small viscosity is subjected to elliptical instability then the amplitude of the flow increases exponentially as long as the growth rate is larger than the damping rate due to viscosity. Once in the nonlinear regime, if the growth rate and the damping rate are the same then the flow remains at that amplitude, the saturation state. For an inviscid fluid, however, we found that outside the spin-over range even if the flow grows initially, it breaks down and decays to the initial state and remains in that state. For some flattenings, the flow becomes unstable and grows exponentially. Eventually, U, V, and P go into saturation states but once the maximum amplitude is reached, W starts to decay back to zero and remains in that state. The norms of perturbations for an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 0.77$ is given in figure 4.8.

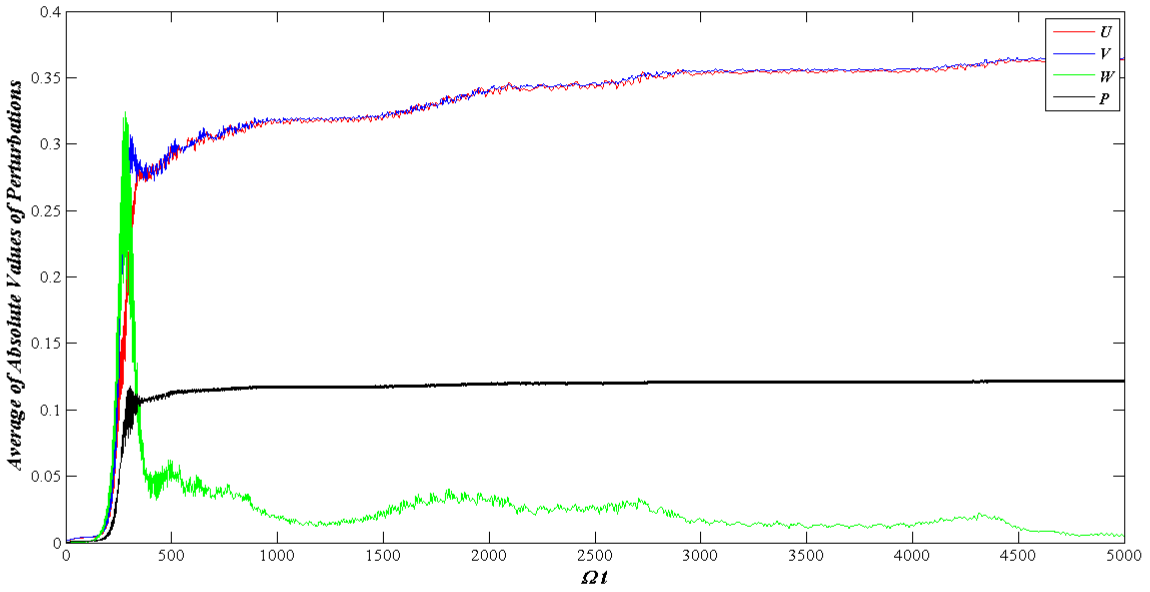
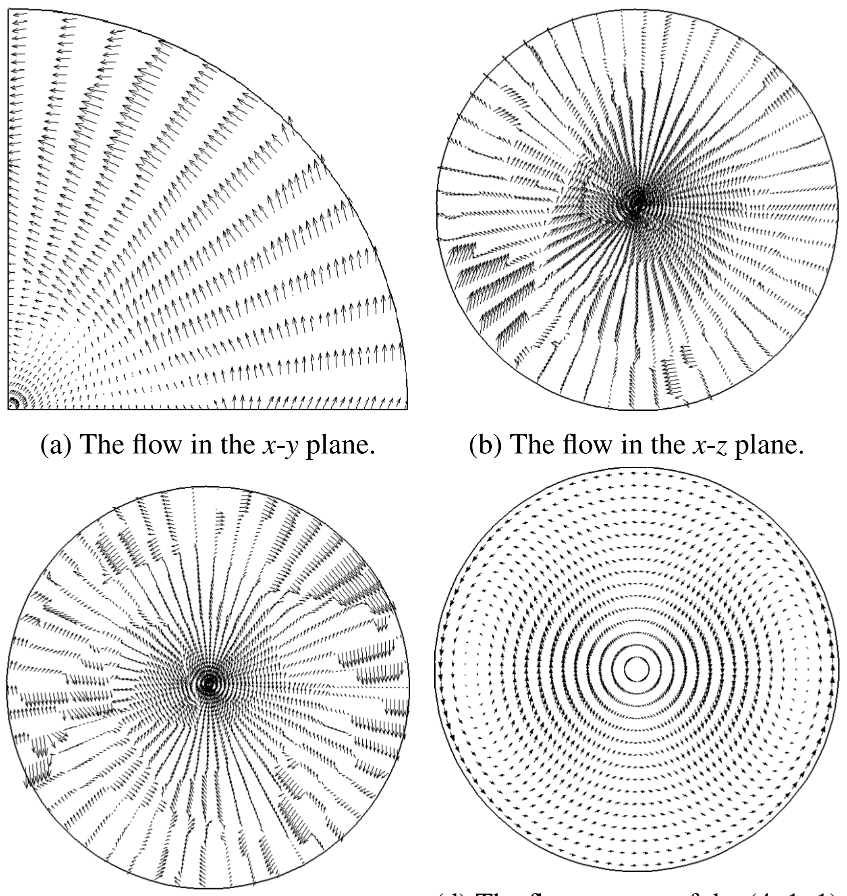


Figure 4.8: The norm of perturbations for an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 0.77$.

From the results of this work and those of Lacaze et al. (2004) we deduce that in order for the instability to repeat itself (i.e., to have a periodic instability), perturbations in all component of the flow need to decay to zero and the initial state of the flow be recovered. In



(a) The flow in the *x*-*y* plane. (b) The flow in the *x*-*z* plane.

(c) The flow in the *y*-*z* plane. (d) The flow pattern of the (4, 1, 1) mode

Figure 4.9: The flow in different planes of an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 0.77$ at the beginning of the saturation state ($\Omega t = 279$)

the cases where the parameters of the system fall outside those of the spin-over instability, however, the some components of the flow go into saturation and the initial flow is not recovered. That may be the reason W does not grow a second time. For flattening outside the spin-over range, the shape of the flow in the *y*-*z* plane is completely different from that of the spin-over instability. That supports the argument that outside the spin-over range other modes beside the (2, 1, 1) and (2, 1, -1) are interacting and becoming elliptical unstable. For instance for $\zeta = 0.77$ the displacement patterns are very similar to that for the (4, 1, 1) mode (fig. 4.9). The displacement patterns for the (4, 1, 1) is also given in fig. 4.9d for reference.

Once the instability is excited, the perturbed flow in the planes perpendicular to the axis of rotation always grows in the opposite direction of that of the basic flow. Since, for flattening outside the spin-over range, the perturbations in the x and y components of the velocity become saturated, if the flattening of a rotating fluid ellipsoid is out of the spin-over range, in the long term the flow becomes two dimensional with smaller amplitude than the initial flow and does not support a repetitive three dimensional motion.

The growth rates of the instability of the motion at the onset of the instability, for equatorial ellipticity set at 0.1, is plotted as a function of the flattening in fig. 4.10.

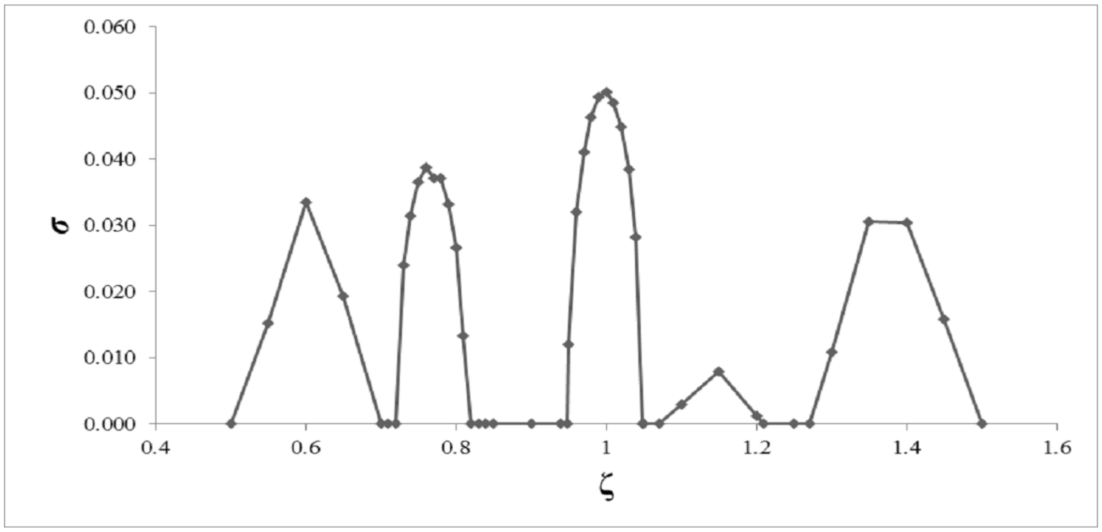


Figure 4.10: The growth rate of the instability at the onset versus the flattening of the ellipsoid for an equatorial ellipticity $\varepsilon = 0.1$.

4.3 Influence of a Background Rotation

In general, the secondary bodies such as the moons and asteroids orbit the fluid bodies. In these cases, the elliptical figure of the streamlines is preserved, as long as the flow is linear, in a frame attached to the secondary body. The effects of this rotation have been studied theoretically, numerically and experimentally (for instance Vladimirov and Vostretsov, 1986; Craik, 1989; Gledzer and Ponomarev, 1992; Kerswell, 1994; Leblanc and Cambon, 1997; Le Dizès, 2000; Seyed-Mahmoud et al., 2000; Cébron et al., 2010a; Le Bars et al., 2010). Hereafter we refer to this rotation as the background rotation. The background rotation may either stabilize or destabilize the motion depending on its magnitude.

In the presence of a background rotation, the reference frame is attached to the secondary body and the Coriolis term is included. (4.5) takes the form

$$\frac{\partial \mathbf{u}'}{\partial t'} + \mathbf{u}' \cdot \nabla' \mathbf{u}' + \frac{1}{\rho} \nabla' p' + 2\Omega_c \times \mathbf{u}' = 0, \tag{4.9a}$$

$$\nabla' \cdot \mathbf{u}' = 0, \tag{4.9b}$$

$$\hat{\mathbf{n}} \cdot \mathbf{u}' = 0 \quad \text{on } S' \tag{4.9c}$$

which can be nondimensionalized as follows

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{1}{1+\varepsilon}\frac{\partial p}{\partial x} - 2\frac{\Omega}{\Omega_c}\left(\sqrt{\frac{1-\varepsilon}{1+\varepsilon}}\right) v = 0, \tag{4.10a}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{1}{1-\varepsilon}\frac{\partial p}{\partial y} + 2\frac{\Omega}{\Omega_c}\left(\sqrt{\frac{1+\varepsilon}{1-\varepsilon}}\right) u = 0, \tag{4.10b}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{1}{\zeta^2}\frac{\partial p}{\partial z} = 0, \tag{4.10c}$$

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{4.10d}$$

$$\hat{\mathbf{n}} \cdot \mathbf{u} = 0 \quad \text{on } S \tag{4.10e}$$

Cébron et al. (2010a) found an expression for the growth rate of the spin-over instability in the presence of a background rotation as a function of ellipticity and the flattening of the ellipsoid. If the background rotation has a magnitude $\Omega_c = N\Omega$, the growth rate of the spin-over instability of an ellipsoid with ellipticity ε and flattening ζ is:

$$\sigma = \sqrt{\frac{\left(1+\varepsilon-\zeta^2+2N\sqrt{1-\varepsilon^2}\right)\left(-1+\varepsilon+\zeta^2-2N\sqrt{1-\varepsilon^2}\right)}{(1+\varepsilon+\zeta^2)(1-\varepsilon+\zeta^2)}}. \tag{4.11}$$

Hence, for a given ellipticity and flattening there is a range of the background rotation frequency ratio

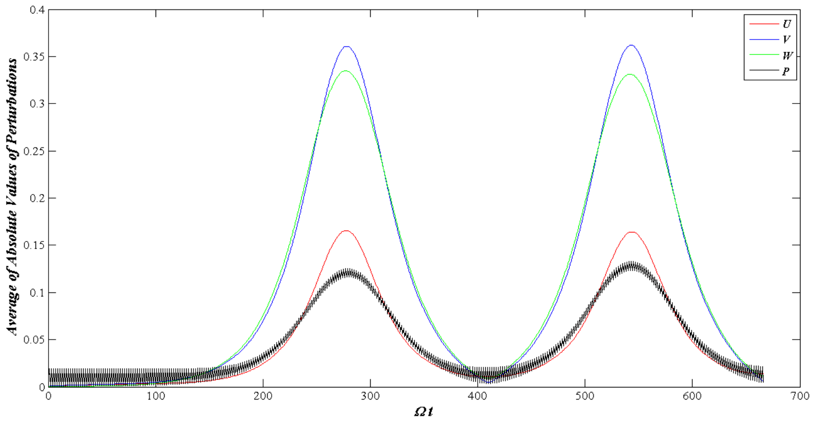
$$\frac{-\left(1+\varepsilon-\zeta^2\right)}{2\sqrt{1-\varepsilon^2}} < N < \frac{-1+\varepsilon+\zeta^2}{2\sqrt{1-\varepsilon^2}} \tag{4.12}$$

in which the spin-over instability may be excited. For example, for an ellipsoid with $\varepsilon = 0.1$ the spin-over instability would be excited in the range of $-0.05025 < N < 0.05025$. In table 4.3, we show the growth rate of the instability calculated using equation (4.11) (column 2) and that computed using our approach (column 3), for different N.

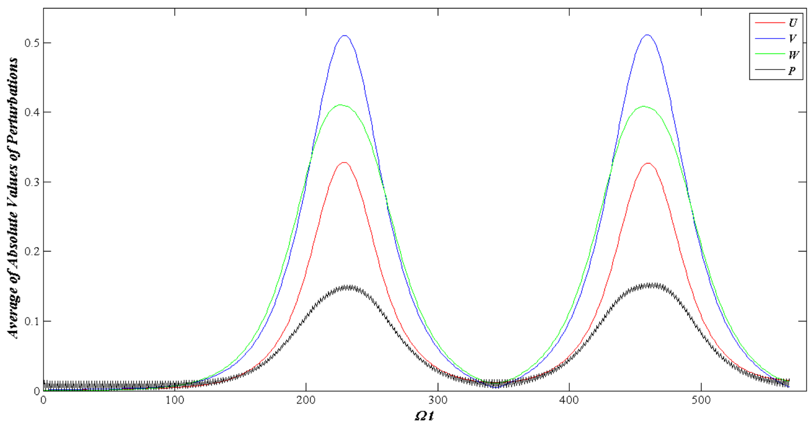
Table 4.3: Growth rate of the spin-over instability for an ellipsoid with $\varepsilon = 0.1$ in the presence of a background rotation calculated from equation (4.11), σ_c, and our approach, σ_m.

N	σ_c	σ_m
-0.04	0.0303	0.0304
-0.03	0.0402	0.0402
-0.02	0.0459	0.0459
-0.01	0.0491	0.0491
0.00	0.0501	0.0500
0.01	0.0491	0.0491
0.02	0.0459	0.0459
0.03	0.0402	0.0402
0.04	0.0303	0.0304

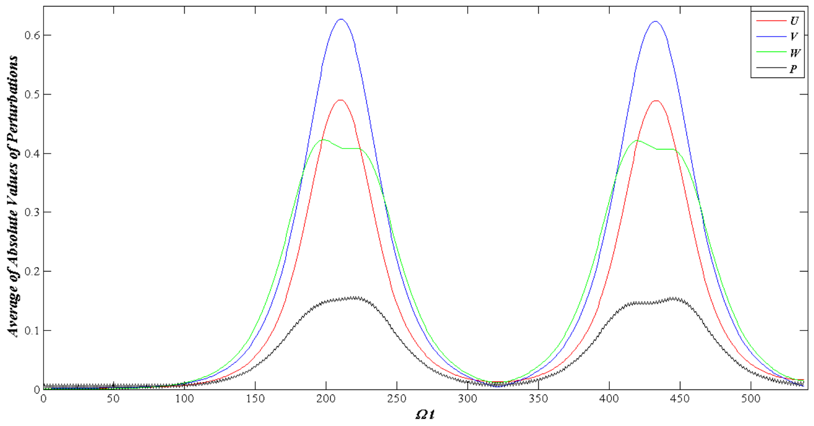
The evolution of norms of the perturbations in this range is given in figure 4.11. It is clear from figure 4.11 that the evolution of the perturbation norms in the presence of a background rotation is quite different from the case of a stationary elliptical deformation. When the elliptical deformation is stationary, U and V have equal values and grow and decay only once in a cycle. They remain stationary for almost a third of the cycle period. However in the presence of a background rotation, U and V no longer reach a saturation state, but grow to a maximum amplitude and then decay to zero twice per cycle. Furthermore, in the presence of a background rotation, U and V do not have the same value. As the absolute value of N increases, so does the gap between the maximum values of U and V. Furthermore, if the fluid body and the secondary body rotate in the same direction, V becomes smaller than U; and if they rotate in opposite directions then U becomes smaller than V. In both cases, however, W changes direction after decaying to zero. For small values of N, W has four local maximums in each cycle. As the absolute value of N increases, the two local maximums in each half of a cycle become closer and at some point they come together. Also, by introducing a background rotation the maximum value of the perturbation norms changes. The greater the absolute value of N, the smaller the maximum value of perturbation norms. In fact at some point the perturbed flow in the x- and y- directions become so small that the flow remains at near solid body rotation in these directions. The displacement vectors in different planes for the instability of an ellipsoid, with $\varepsilon = 0.1$ and $\zeta = 1$, in the presence of a background rotation are shown in figure 4.12-4.14.



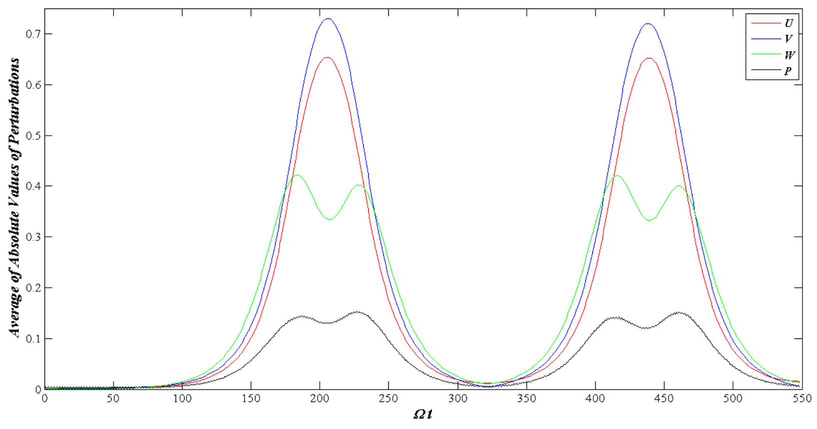
(a) $N = -0.04$.



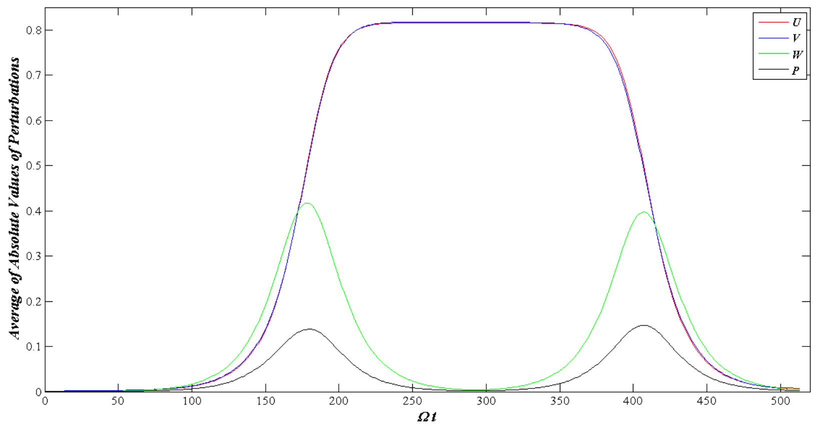
(b) $N = -0.03$.



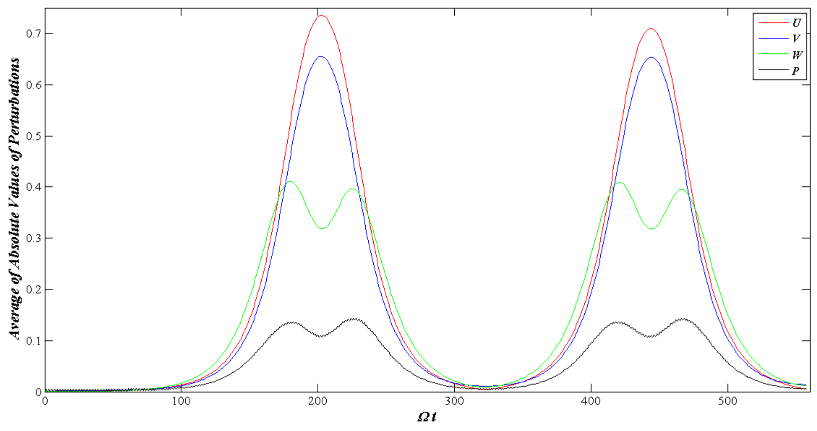
(c) $N = -0.02$.



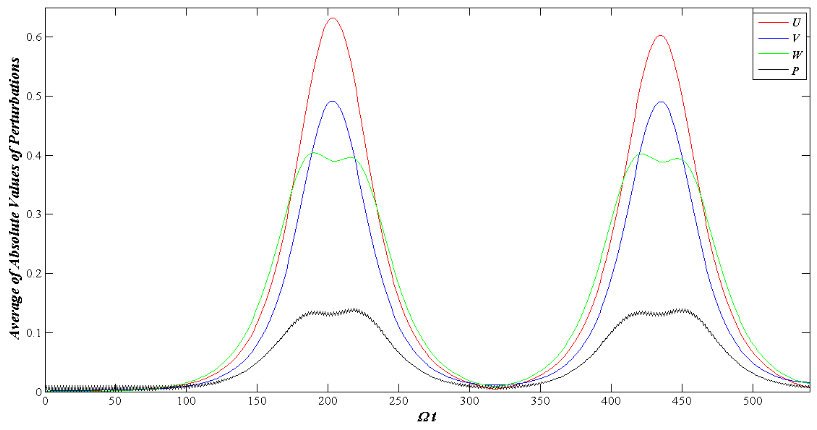
(d) $N = -0.01$.



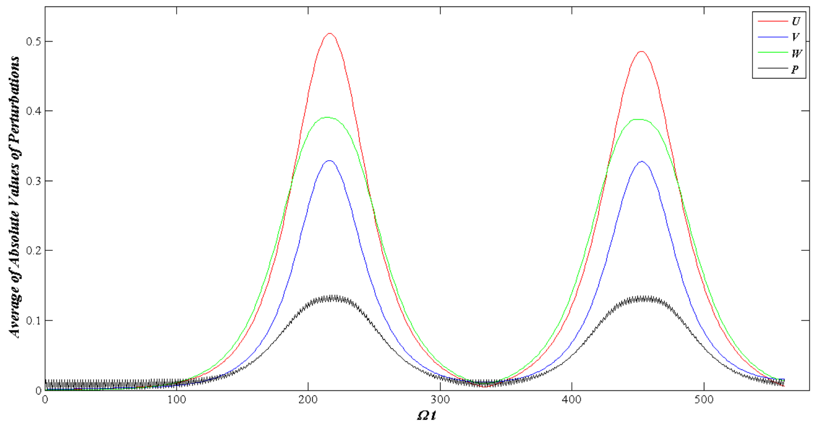
(e) $N = 0.00$.



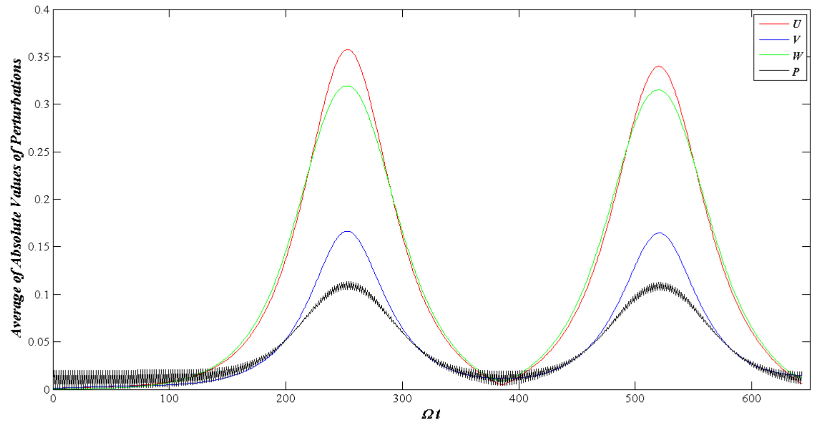
(f) $N = 0.01$.



(g) $N = 0.02$.

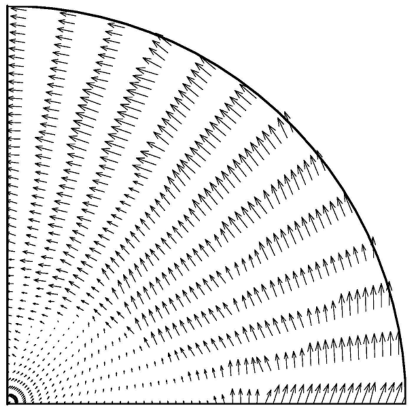


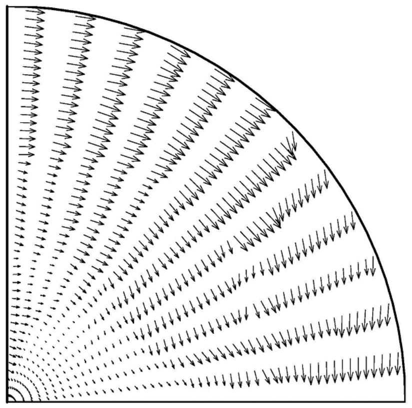
(h) $N = 0.03$.

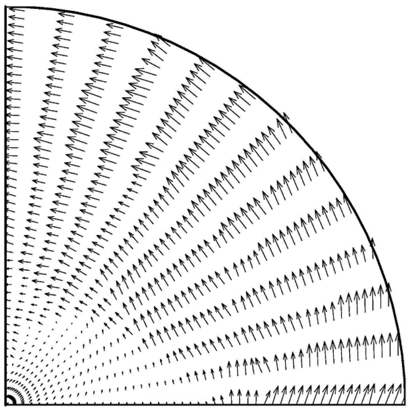


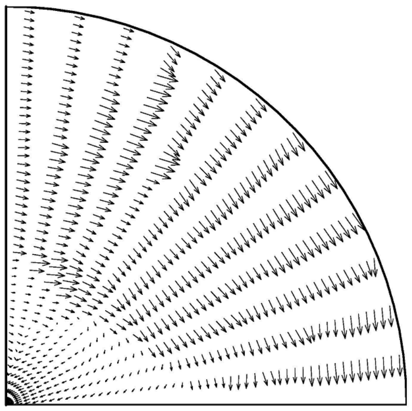
(i) $N = 0.04$.

Figure 4.11: Evolution of norms of perturbations of the spin-over instability in an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 1$ in the presence of a background rotation.


(a) The initial flow ($\Omega t = 0$).


(b) The first maximum point of U and V ($\Omega t = 203$).


(c) The second minimum point of U and V ($\Omega t = 322$).


(d) The second maximum point of U and V ($\Omega t = 444$).

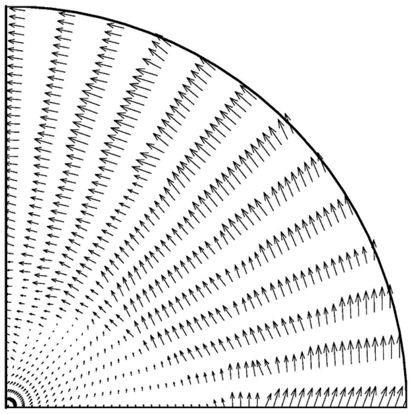
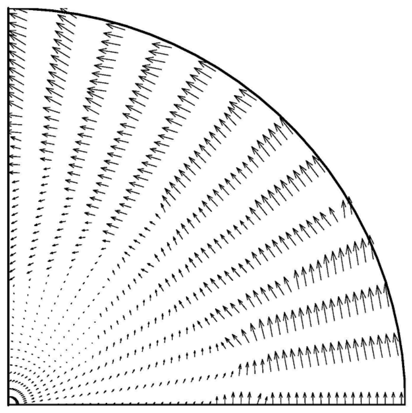
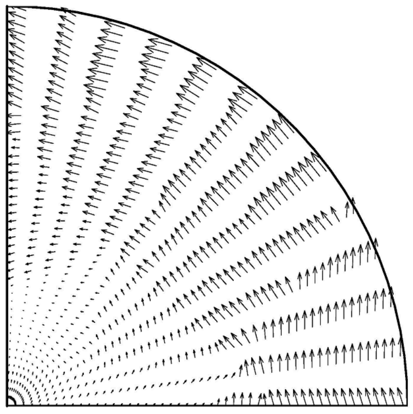
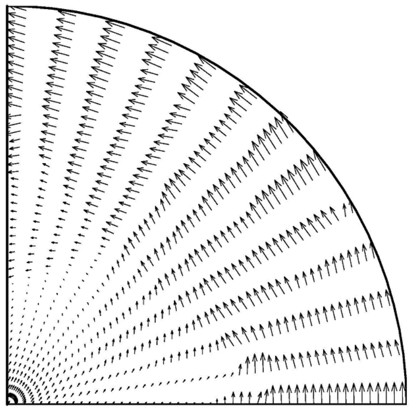
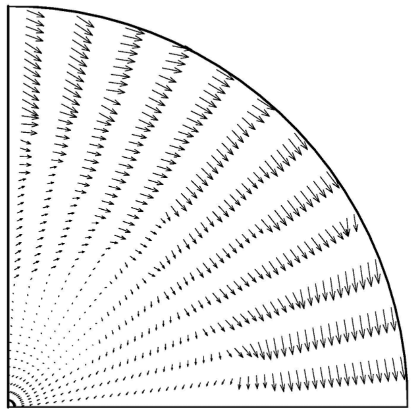

(e) The third minimum point of U and V ($\Omega t = 558$).

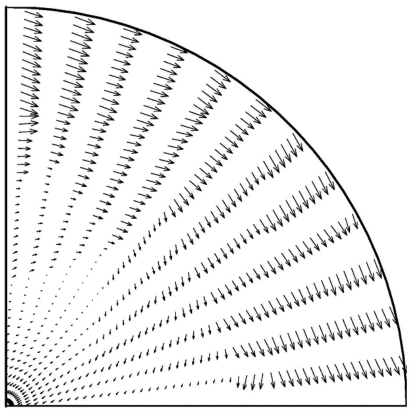
Figure 4.12: The displacement vectors in x-y planes during the instability of a fluid ellipsoid with $\varepsilon = 0.1$ and in the presence of a background rotation with rotational frequency ratio $N = 0.01$.


(a) The first local maximum point of W $(\Omega t = 180)$.


(b) The second local minimum point of W $(\Omega t = 203)$.


(c) The second local maximum point of W $(\Omega t = 225)$.


(d) The third local maximum point of W $(\Omega t = 420)$.


(e) The fourth local minimum point of W $(\Omega t = 444)$.

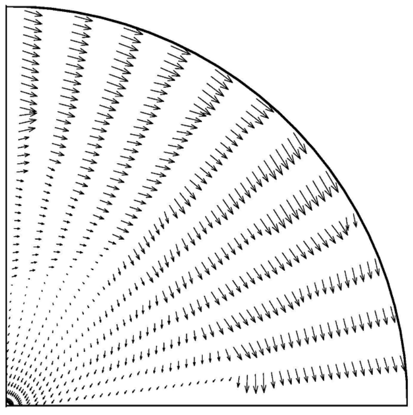
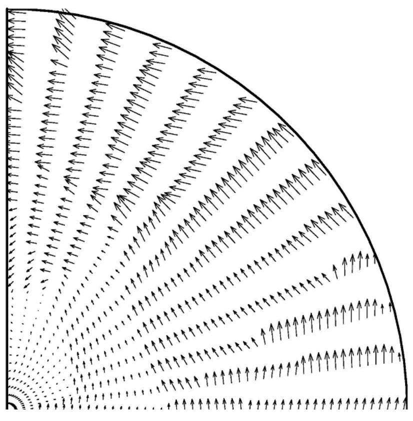
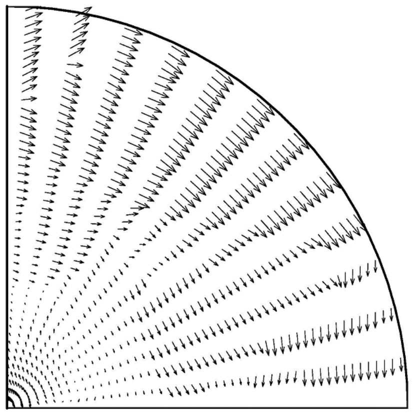
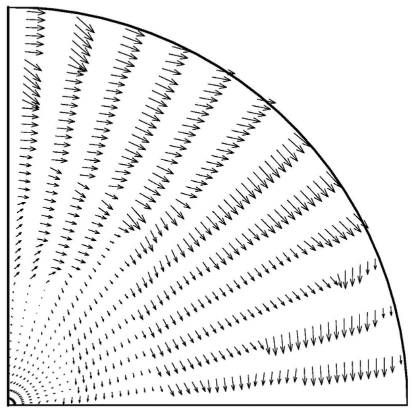

(f) The fourth local maximum point of W $(\Omega t = 466)$.

Figure 4.13: The displacement vectors in x-z planes during the instability of a fluid ellipsoid with $\varepsilon = 0.1$ and in the presence of a background rotation with rotational frequency ratio $N = 0.01$.


(a) The first local maximum point of W ($\Omega t = 180$).


(b) The second local maximum point of W ($\Omega t = 225$).


(c) The third local maximum point of W ($\Omega t = 420$).

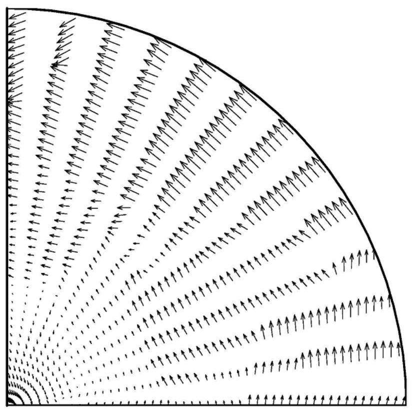

(d) The fourth local maximum point of W ($\Omega t = 466$).

Figure 4.14: The displacement vectors in y-z planes during the instability of a fluid ellipsoid with $\varepsilon = 0.1$ and in the presence of a background rotation with rotational frequency ratio $N = 0.01$.

The elliptical instability may not be excited, if the frequency ratio, N, is outside the spin-over range. As mentioned earlier, the background rotation may stabilize or destabilize the motion. Our results show that it is possible that the flow becomes unstable outside the spin-over range. However, initially the motion grows exponentially but U, V and P go to saturated states after they reach maximums, while W decays to zero and remains in that state. The evolution of the perturbation norms for an ellipsoid with $\varepsilon = 0.1$ and $\zeta = 1$ in the presence of a background rotation with rotational frequency ratio $N = 0.3$, which is outside the spin-over range, is given in figure 4.15.

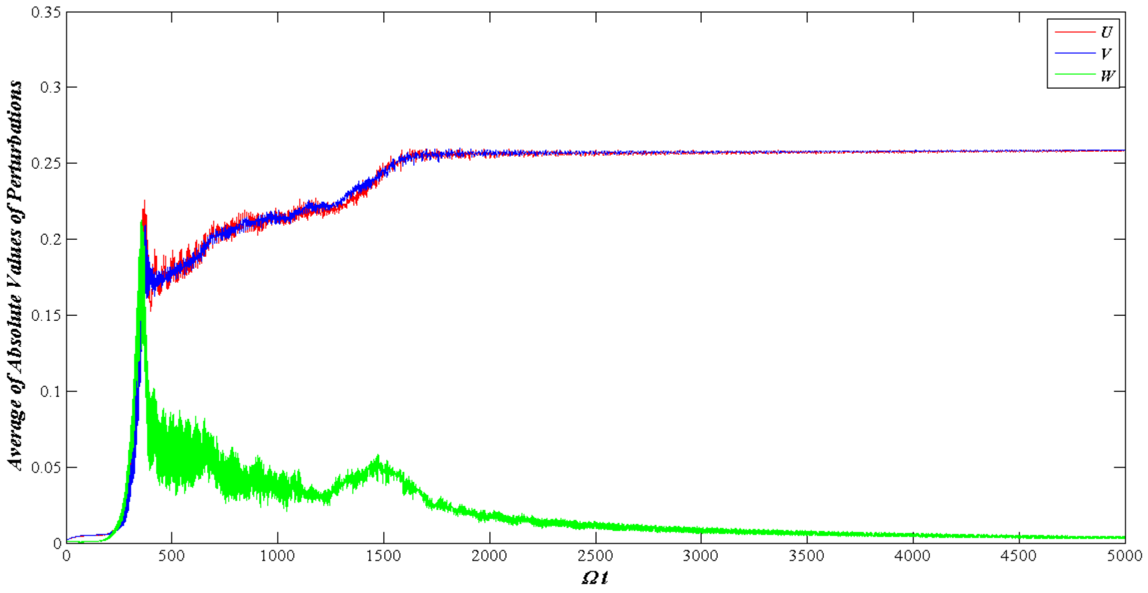


Figure 4.15: The evolution of perturbation norms for an ellipsoid with $\varepsilon = 0.1$ in the presence of a background rotation with rotational frequency ratio $N = 0.3$.

Chapter 5

Discussion

In this thesis, we have presented a numerical study of the elliptical instability in the nonlinear regime for a rotating triaxial ellipsoid containing incompressible inviscid and homogeneous fluid. We have investigated the effects of the oblateness of the ellipsoid and the presence of a background rotation on the growth rate and the evolution of the instability. Motivation for this work came from the possibility that elliptical instability may be excited in rotating fluid bodies such as rotating stars and planetary fluid cores. Although this phenomenon has been the topic of intense experimental, theoretical and computational research for the past several decades, to the best of our knowledge, this is the first time that inviscid incompressible fluid ellipsoids are thoroughly investigated in the nonlinear regime. The only other work that considers nonlinear effects is that of Lacaze et al. (2004) who use a semi analytical approach to study the spin-over instability that is restricted to a spheroidal geometry. In their work it is necessary to assume a priori that the spin-over modes interact to excite the instability. Other researches, such as Gledzer and Ponomarev (1992), Kerswell (1993), Kerswell and Malkus (1998) in the case of ellipsoids or Seyed-Mahmoud et al. (2000) in the case of ellipsoidal shells, have ignored the nonlinear terms and studied the onset of the elliptical instability.

In the previous works, this phenomenon was treated mathematically as the superposition of the inertial modes of the rotating fluid body in order to compute the growth rate of the instability. In this work we have used a finite element method to directly solve the nonlinear equations of motion describing the instability and have relaxed the above-mentioned assumption.

We have shown that if the parameters of the system, the oblateness of the ellipsoid and the rotational frequency ratio of the background rotation, are in the spin-over range, a repetitive three-dimensional motion may be maintained indefinitely; otherwise, instability might be excited initially but thereafter the motion would become two dimensional with small displacement amplitudes in x- and y- directions.

If there is a planet with the same geometrical parameters as the Earths cores, $\zeta \approx 0.99745$, $\varepsilon = 5 \times 10^{-8}$, and $\Omega_c/\Omega \approx 0.034$, but without a solid inner core, the spin-over would not be excited in this planet's fluid core. Hence, if it were purely inviscid, there would be no three-dimensional motion after a short period of time in its history.

Seyed-Mahmoud et al. (2000) performed a linear analysis and showed that the flow would become unstable in the linear regime for any flattening and background rotation. They concluded that for any set of two or more inertial modes, there is a critical flattening where the instability would be excited. A preliminary investigation showed that the instability becomes excited in some ranges of the system parameters (the flattening and the background rotation frequency). In other words, for each set of inertial modes which meet the criteria for instability, there is a range of values for the flattening and background rotation frequency where the modes become elliptically unstable. For the intervals between the unstable ranges, the instability does not become excited at all and a two-dimensional motion is maintained indefinitely (for instance see figure 4.10). Our results are consistent with Cébron et al. (2010a). We plan to explore this matter further.

We have not solved the elliptical instability problem for ellipsoidal fluid shells yet. However, based on the previous works, a scientific guess is that the motion in a rotating ellipsoidal fluid shell might be similar to that of an ellipsoid.

Bibliography

Aldridge, Keith; Seyed-Mahmoud, Behnam; Henderson, Gary, and van Wijngaarden, William. Elliptical instability of the earth's fluid core. *Physics of the earth and planetary interiors*, 103(3):365–374, 1997.

Amdahl, Gene M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485. ACM, 1967.

Arkani-Hamed, J; Seyed-Mahmoud, B; Aldridge, KD, and Baker, RE. Tidal excitation of elliptical instability in the martian core: Possible mechanism for generating the core dynamo. *Journal of Geophysical Research: Planets (1991–2012)*, 113(E6), 2008.

Billant, Paul; Brancher, Pierre, and Chomaz, Jean-Marc. Three-dimensional stability of a vortex pair. *Physics of Fluids*, 11:2069, 1999.

Bullard, Edward and Gellman, H. Homogeneous dynamos and terrestrial magnetism. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, pages 213–278, 1954.

Cébron, D; Le Bars, M; Maubert, P, and Le Gal, P. Magnetohydrodynamic simulations of the elliptical instability in triaxial ellipsoids. *Geophysical & Astrophysical Fluid Dynamics*, 106(4-5):524–546, 2012a.

Cébron, David; Le Bars, Michael; Leontini, Justin; Maubert, Pierre, and Le Gal, Patrice. A systematic numerical study of the tidal instability in a rotating triaxial ellipsoid. *Physics of the Earth and Planetary Interiors*, 182(1):119–128, 2010a.

Cébron, David; Maubert, Pierre, and Le Bars, Michael. Tidal instability in a rotating and differentially heated ellipsoidal shell. *Geophysical Journal International*, 182(3):1311–1318, 2010b.

Cébron, David; Le Bars, Michael; Noir, J, and Aurnou, JM. Libration driven elliptical instability. *Physics of Fluids*, 24:061703, 2012b.

Chapman, Barbara; Jost, Gabriele, and Van Der Pas, Ruud. *Using OpenMP: portable shared memory parallel programming*, volume 10. The MIT Press, 2008.

Christensen, UR and Aubert, Julien. Scaling properties of convection-driven dynamos in rotating spherical shells and application to planetary magnetic fields. *Geophysical Journal International*, 166(1):97–114, 2006.

Clausen, N and Tilgner, A. Elliptical instability of compressible flow in ellipsoids. *arXiv preprint arXiv:1312.5084*, 2013.

Craik, ADD. The stability of unbounded two-and three-dimensional flows subject to body forces: some exact solutions. *Journal of Fluid Mechanics*, 198:275–292, 1989.

Eloy, Christophe; Le Gal, Patrice, and Le Dizès, Stéphane. Experimental study of the multipolar vortex instability. *Physical review letters*, 85(16):3400, 2000.

Eloy, Christophe; Le Gal, Patrice; Le Dizès, Stéphane, and others, . Elliptic and triangular instabilities in rotating cylinders. *Journal of Fluid Mechanics*, 476:357–388, 2003.

Glatzmaier, Gary A and Roberts, Paul H. A three-dimensional self-consistent computer simulation of a geomagnetic field reversal. 1995.

Gledzer, EB and Ponomarev, VM. Finite-dimensional approximation of the motions of an incompressible fluid in an ellipsoidal cavity. *Akademiia Nauk SSSR Fizika Atmosfery i Okeana*, 13:820–827, 1978.

Gledzer, EB and Ponomarev, VM. Instability of bounded flows with elliptical streamlines. *Journal of Fluid Mechanics*, 240:1–30, 1992.

Gledzer, EB; Dolzhanskii, FV; Obukhov, AM, and Ponomarev, VM. An experimental and theoretical study of the stability of motion of a liquid in an elliptical cylinder. *Academy of Sciences, USSR, Izvestiya, Atmospheric and Oceanic Physics*, 11:617–622, 1976.

Greenhill, AG. On the rotation of a liquid ellipsoid about its mean axis. In *Proc. Cambridge Philos. Soc*, volume 3, pages 233–246, 1879.

Gropp, William D; Lusk, Ewing L, and Skjellum, Anthony. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. the MIT Press, 1999.

Hadamard, Jacques. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, 13(49-52):28, 1902.

Herreman, Wietze; Le Bars, Michael, and Le Gal, Patrice. On the effects of an imposed magnetic field on the elliptical instability in rotating spheroids. *Physics of Fluids*, 21(4): 046602–046602, 2009.

Jiang, Bo-nan. *The least-squares finite element method: theory and applications in computational fluid dynamics and electromagnetics*. Springer, 1998.

Kageyama, Akira; Sato, Tetsuya, and others, . Computer simulation of a magnetohydrodynamic dynamo. ii. *Physics of Plasmas*, 2:1421, 1995.

Kageyama, Akira; Miyagoshi, Takehiro, and Sato, Tetsuya. Formation of current coils in geodynamo simulations. *Nature*, 454(7208):1106–1109, 2008.

Kerswell, Richard R. Elliptical instability. *Annual review of fluid mechanics*, 34(1):83–113, 2002.

Kerswell, Richard R and Malkus, Willem VR. Tidal instability as the source for io's magnetic signature. *Geophysical research letters*, 25(5):603–606, 1998.

Kerswell, RR. The instability of precessing flow. *Geophysical & Astrophysical Fluid Dynamics*, 72(1-4):107–144, 1993.

Kerswell, RR. Tidal excitation of hydromagnetic waves and their damping in the earth. *Journal of Fluid Mechanics*, 274:219–241, 1994.

Koelbel, Charles H; Loveman, David B; Schreiber, Robert S; Steele Jr, Guy L, and Zosel, Mary E. The high performance fortran handbook. scientific and engineering computation, 1994.

Kuang, Weijia and Bloxham, Jeremy. An earth-like numerical dynamo model. *Nature*, 389 (6649):371–374, 1997.

Lacaze, Laurent; Le Gal, Patrice; Le Dizès, Stéphane, and others, . Elliptical instability in a rotating spheroid. *Journal of Fluid Mechanics*, 505:1–22, 2004.

Lacaze, Laurent; Le Gal, Patrice, and Le Dizès, Stéphane. Elliptical instability of the flow in a rotating shell. *Physics of the Earth and Planetary Interiors*, 151(3):194–205, 2005.

Lacaze, Laurent; Herreman, Wietze; Le Bars, Michael; Le Dizès, Stéphane, and Le Gal, Patrice. Magnetic field induced by elliptical instability in a rotating spheroid. *Geophysical and Astrophysical Fluid Dynamics*, 100(4-5):299–317, 2006.

Lacaze, Laurent; Ryan, Kris; Le Dizès, Stéphane, and others, . Elliptic instability in a strained batchelor vortex. *Journal of Fluid Mechanics*, 577:341, 2007.

Le Bars, Michael; Le Dizès, Stéphane; Le Gal, Patrice, and others, . Coriolis effects on the elliptical instability in cylindrical and spherical rotating containers. *Journal of Fluid Mechanics*, 585:323–342, 2007.

Le Bars, MICHAEL; Lacaze, Laurent; Le Dizès, Stéphane; Le Gal, Patrice, and Rieutord, Michel. Tidal instability in stellar and planetary binary systems. *Physics of the Earth and Planetary Interiors*, 178(1):48–55, 2010.

Le Dizès, Stéphane. Three-dimensional instability of a multipolar vortex in a rotating flow. *Physics of Fluids*, 12:2762, 2000.

Le Dizès, Stéphane and Laporte, Florent. Theoretical predictions for the elliptical instability in a two-vortex flow. *Journal of Fluid Mechanics*, 471(1):169–201, 2002.

Leblanc, Stéphane and Cambon, Claude. On the three-dimensional instabilities of plane flows subjected to coriolis force. *Physics of Fluids (1994-present)*, 9(5):1307–1316, 1997.

Leweke, T and Williamson, CHK. Cooperative elliptic instability of a vortex pair. *Journal of Fluid Mechanics*, 360(1):85–119, 1998.

Lewis, Bil; Berg, Daniel J, and others, . *Multithreaded programming with Pthreads*, volume 2550. Sun Microsystems Press, 1998.

Malkus, Willem VR. An experimental study of global instabilities due to the tidal (elliptical) distortion of a rotating elastic cylinder. *Geophysical & Astrophysical Fluid Dynamics*, 48(1-3):123–134, 1989.

Mason, DM and Kerswell, RR. Nonlinear evolution of the elliptical instability: an example of inertial wave breakdown. *Journal of Fluid Mechanics*, 396:73–108, 1999.

McFadden, PL; Merrill, RT; McElhinny, MW, and Lee, Sunhee. Reversals of the earth's magnetic field and temporal variations of the dynamo families. *Journal of Geophysical Research: Solid Earth (1978–2012)*, 96(B3):3923–3933, 1991.

Merrill, Ronald T and McElhinny, Michael W. *The Earth's magnetic field: Its history, origin and planetary perspective*, volume 13. Cambridge Univ Press, 1983.

Meunier, Patrice; Ehrenstein, Uwe; Leweke, Thomas, and Rossi, Maurice. A merging criterion for two-dimensional co-rotating vortices. *Physics of Fluids*, 14:2757, 2002.

Mueller, Jennifer L and Siltanen, Samuli. *Linear and Nonlinear Inverse Problems with Practical Applications*, volume 10. SIAM, 2012.

Nieplocha, Jaroslaw; Harrison, Robert J, and Littlefield, Richard J. Global arrays: A nonuniform memory access programming model for high-performance computers. *The Journal of Supercomputing*, 10(2):169–189, 1996.

Noir, Jerome; Cébron, David; Le Bars, Michael; Sauret, Alban, and Aurnou, JM. Experimental study of libration-driven zonal flows in non-axisymmetric containers. *Physics of the Earth and Planetary Interiors*, 204:1–10, 2012.

Olson, Peter. Geomagnetic polarity reversals in a turbulent core. *Physics of the earth and planetary interiors*, 33(4):260–274, 1983.

Ou, Shangli; Tohline, Joel E, and Motl, Patrick M. Further evidence for an elliptical instability in rotating fluid bars and ellipsoidal stars. *The Astrophysical Journal*, 665(2):1074, 2007.

Palacios, A. Heteroclinic cycles. *Scholarpedia*, 2(1):2352, 2007.

Pierrehumbert, RT. Universal short-wave instability of two-dimensional eddies in an inviscid fluid. *Physical review letters*, 57:2157–2159, 1986.

Poincaré, H. Sur la précession des corps déformables. *Bulletin Astronomique, Serie I*, 27: 321–356, 1910.

Rieutord, Michel. Evolution of rotation in binaries: physical processes. *arXiv preprint astro-ph/0308313*, 2003.

Roberts, PH. Origin of the main field: dynamics. *Geomagnetism, Vol. 2, p. 251-306*, 2: 251–306, 1987.

Roy, Clément; Schaeffer, Nathanaël; Le Dizès, Stéphane, and Thompson, Mark. Stability of a pair of co-rotating vortices with axial flow. *Physics of Fluids*, 20:094101, 2008.

Sakuraba, Ataru and Roberts, Paul H. Generation of a strong magnetic field using uniform heat flux at the surface of the core. *Nature Geoscience*, 2(11):802–805, 2009.

Schaeffer, Nathanaël and Le Dizès, Stéphane. Nonlinear dynamics of the elliptic instability. *Journal of Fluid Mechanics*, 646:471–480, 2010.

Seyed-Mahmoud, Behnam. *Elliptical instability in rotating ellipsoidal fluid shells: applications to the Earth's fluid core*. PhD thesis, York University Toronto, Ontario, Canada, 1999.

Seyed-Mahmoud, Behnam; Henderson, Gary, and Aldridge, Keith. A numerical model for elliptical instability of the earth's fluid outer core. *Physics of the Earth and Planetary Interiors*, 117(1):51–61, 2000.

Seyed-Mahmoud, Behnam; Aldridge, Keith, and Henderson, Gary. Elliptical instability in rotating spherical fluid shells: application to earths fluid core. *Physics of the Earth and Planetary Interiors*, 142(3):257–282, 2004.

Shewchuk, Jonathan Richard. An introduction to the conjugate gradient method without the agonizing pain, 1994.

Takahashi, Futoshi; Matsushima, Masaki, and Honkura, Yoshimori. Simulations of a quasi–taylor state geomagnetic field including polarity reversals on the earth simulator. *Science*, 309(5733):459–461, 2005.

Takahashi, Futoshi; Matsushima, Masaki, and Honkura, Yoshimori. Scale variability in convection-driven mhd dynamos at low ekman number. *Physics of the Earth and Planetary Interiors*, 167(3):168–178, 2008.

Vladimirov, VA and Vostretsov, DG. Instability of steady flows with constant vorticity in vessels of elliptic cross-section. *Journal of Applied Mathematics and Mechanics*, 50(3): 279–285, 1986.

Appendix A

An Introduction to the Conjugate Gradient Method

A.1 Introduction

The Conjugate Gradient Method is one of the most favorite iterative algorithms for solving a sparse, symmetric, and positive-definite linear system of equations of the form

$$Ax = b \tag{A.1}$$

where x is an unknown vector, b is a known vector, and A is a known, symmetric, positive-definite matrix.

A matrix A is positive-definite if

$$x^T Ax > 0 \tag{A.2}$$

for every nonzero vector x.

A quadratic form is a scalar, quadratic function of a vector with the form

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c \tag{A.3}$$

where A is a matrix, x and b are vectors, and c is a scalar constant.

Theorem A.1 (Shewchuk, 1994)**.** *The solution to $Ax = b$ minimizes the quadratic form.*

Proof. Suppose A is symmetric. Let x be a point that satisfy $Ax = b$, and let e be an error term. Then

$$\begin{aligned}
f(x+e) \quad &= \frac{1}{2}(x+e)^T A (x+e) b^T (x+e) + c \\
&= \frac{1}{2}x^2 Ax + e^T Ax + \frac{1}{2}e^T Ae - b^T x - b^T e + x \\
&= \frac{1}{2}x^T Ax - b^T x + c + e^T b + \frac{1}{2}e^T Ae - b^T e \\
&= f(x) + \frac{1}{2}e^T Ae.
\end{aligned}$$

If A is positive-definite, then the latter term is positive for all $e \neq 0$; therefore x minimizes f. $\square$

Another way to look at Theorem A.1 is in terms of the gradient of the quadratic form. The gradient of (A.3) which points in the direction of greatest increase of $f(x)$, is defined to be

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}. \tag{A.4}$$

By applying equation (A.4) to equation (A.3)and assuming that A is symmetric the following equation can be derived

$$f'(x) = Ax - b \tag{A.5}$$

The linear system (A.1) can be solved by setting the gradient to zero. Hence, the solution to (A.1) is a critical point of $f(x)$. Furthermore, this solution is a minimum of $f(x)$, if A is positive-definite, too.

The Steepest Descent and Conjugate Gradient methods were developed in order to minimize the quadratic form (A.3).

A.2 The Method of Steepest Descent

The Steepest Descent Method starts at an arbitrary point x_0 and takes a series of steps $x_1, x_2, \cdots$ until a close enough approximation to the solution x is found. The method of Steepest Descent takes a step in the direction in which f decrease most quickly, i.e. the opposite direction of $f'(x_i)$.

The error $e_i = x_i - x$ is a vector that denotes the difference between a step and the solution. The residual $r_i = b - Ax_i$ which can also be written as $r_i = -Ae_i$, is the error transformed by A into the same space as b and indicates the difference between b calculated at each step and the correct value of b. Since the opposite direction of $f'(x_i)$ is the direction of steepest descent and $r_i = -f'(x_i)$, the residual is also the direction to the next step.

Starting at x_0, the first step is along the direction of steepest descent

$$x_1 = x_0 + \alpha r_0. \tag{A.6}$$

The α that minimizes f along the above line can be calculated by setting the directional derivative $\frac{d}{d\alpha} f(x_1)$ to zero. By the chain rule,

$$\begin{aligned} \frac{d}{d\alpha} f(x_1) &= f'(x_1)^T \frac{d}{d\alpha} x_1 \\ &= f'(x_1)^T r_0. \end{aligned}$$

Ergo, at the value of α where r_0 and $f'(x_1)$ are orthogonal, f is minimized.

α can be calculated as follow

$$\begin{aligned} f'(x_1) r_0 &= 0 \\ r_0^T r_0 &= 0 \\ (b - Ax_1)^T r_0 &= 0 \\ (b - A(x_0 + \alpha r_0))^T r_0 &= 0 \\ (b - Ax_0)^T r_0 - \alpha (Ar_0)^T r_0 &= 0 \\ \alpha (Ar_0)^T r_0 &= (b - Ax_0)^T r_0 \\ \alpha r_0^T A r_0 &= r_0^T r_0 \\ \alpha &= \frac{r_0^T r_0}{r_0^T A r_0} \end{aligned}$$

Finally, the method of Steepest Descent can be summarized as follows,

$$r_i = b - Ax_i, \tag{A.7a}$$

$$\alpha_i = \frac{r_i^T r_i}{r_i^T A r_i}, \tag{A.7b}$$

$$x_{i+1} = x_i + \alpha r_i. \tag{A.7c}$$

This iterative procedure should be continued until it converges.

An efficient implementation of the method of Steepest Descend is (Shewchuk, 1994)

```
$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$\delta \Leftarrow r^T r$
While $i < i_{max}$ and $\delta > \varepsilon^2 \delta_0$ do
    $q \Leftarrow Ar$
    $\alpha \Leftarrow \frac{\delta}{r^T q}$
    $x \Leftarrow x + \alpha r$
    $r \Leftarrow r - \alpha q$
    $\delta \Leftarrow r^T r$
    $i \Leftarrow i + 1$
```

where i_{max} is the maximum number of iterations and $\varepsilon < 1$ is the error tolerance.

This algorithm stops when the maximum number of iterations has been exceeded, or when $\|r_i\| \leq \varepsilon \|r_0\|$.

A.3 The Method of Conjugate Directions

The problem with the method of Steepest Descent is that it often takes steps in the same direction as earlier steps. The method of Conjugate Directions was developed to overcome this problem. In this method a set of A-orthogonal search directions $d_0, d_1, \cdots, d_{n-1}$ is chosen. Only one step is taken in each search direction; and, after n steps, the solution x will be reached.

By definition, two vectors d_i and d_j are A-orthogonal, or conjugate, if

$$d_i^T A d_j = 0.$$

For each step, a point is chosen

$$x_{i+1} = x_i + \alpha_i d_i. \tag{A.8}$$

The x_{i+1} should minimize the quadratic form along the search direction d_i. Hence, the directional derivative is equal to zero.

$$\begin{aligned}
\frac{d}{d\alpha} f(x_{i+1}) &= 0 \\
f'(x_{i+1})^T \frac{d}{d\alpha} x_{i+1} &= 0 \\
-r_{i+1}^T d_i &= 0 \\
d_i^T A e_{i+1} &= 0.
\end{aligned}$$

Thus, for x_{i+1} to be the minimum point along the search direction d_i, e_{i+1} should be A-orthogonal to d_i. Using this condition α_i can be found as follows

$$\begin{aligned}
d_i^T e_{i+1} &= 0 \\
d_i^T (e_i + \alpha_i d_i) &= 0 \\
\alpha_i &= -\frac{d_i^T e_i}{d_i^T d_i}.
\end{aligned} \tag{A.9}$$

Theorem A.2 (Mueller and Siltanen, 2012). *Let $\{d_i\}_{i=0}^{n-1}$ be a set of nonzero A-orthogonal vectors. For any $x_0 \in \mathbb{R}^n$ the sequence $\{x_k\}$ generated by*

$$x_{k+1} = x_k = \alpha_k d_k, \qquad k \geq 0$$

with

$$\alpha_k = \frac{d_i^T r_i}{d_i^T A d_i}$$

converges to the unique solution x^* of $Ax = b$ after n step. That is $x_n = x^*$

Proof. Since the d_i are linearly independent, we can write $x^* - x_0 = \alpha_0 d_0 + \alpha_1 d_1 + \cdots +$

$\alpha_{n-1}d_{n-1}$, multiplying by A and then d_k^T to get

$$\alpha_k = \frac{d_k^T A\left(x^* - x_0\right)}{d_k^T A d_k}.$$

From the iterative process leading to x_k we have

$$x_k - x_0 = \alpha_0 d_0 + \cdots + \alpha_{k-1} d_{k-1}$$

we see that

$$d_k^T A\left(x_k - x_0\right) = \sum_{i=0}^{k-1} \alpha_i d_k^T A d_i,$$

which implies by the A-orthogonality of the d_i that $d_k^T A x_k = d_k^T A x_0$. Substituting this into the expression for α_k gives

$$\begin{aligned}
\alpha_k &= \frac{d_k^T A x^* - d_k^T A x_k}{d_k^T A d_k} = \frac{d_k^T b - d_k^T A x_k}{d_k^T A d_k} \\
&= \frac{d_k^T\left(b - A x_k\right)}{d_k^T A d_k} = \frac{r_k^T d_k}{d_k^T A d_k}.
\end{aligned}$$

□

The next thing to do is to find a set of A-orthogonal search directions $\{d_i\}$. The *conjugate Gram-Schmidt process* provides a simple way to generate them.

Consider a set of n linearly independent vectors $u_0, u_1, \cdots, u_{n-1}$. Set $d_0 = u_0$, and for $i < 1$

$$d_i = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_k, \tag{A.10}$$

where the β_{ik} are defined for $i > k$. The value of β_{ik} can be found by using the A-orthogonality of search directions:

$$\begin{aligned}
d_i^T A d_j &= u_i^T A d_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T A d_j \\
0 &= u_i^T A d_j + \beta_{ij} d_j^T A d_j, \qquad i > j \\
\beta_{ij} &= -\frac{u_i^T A d_j}{d_j^T A d_j}
\end{aligned}$$

The problem with the conjugate Gram-Schmidt process is that all the old search vectors must be kept in memory to construct new one. Also, to generate the full set $O\left(n^3\right)$ operations are needed. The Conjugate Gradient Method was developed to cure these disadvantages.

A.4 The Method of Conjugate Gradients

In the Conjugate Gradient Method, the search directions are determined at each step rather than being specified beforehand. At each step, the current gradient vector is calculated and its negative is added to the linear combination of the previous vectors in order to determine the new search direction. In other words, in the Conjugate Gradient Method, the search directions are constructed by conjugating of the residuals.

The method of Conjugate Gradient is:

$$\begin{aligned}
d_0 &= r_0 = b - Ax_0, \\
\alpha_i &= \frac{r_i^T r_i}{d_i^T A d_i}, \\
x_{i+1} &= x_i + \alpha_i d_i, \\
r_{i+1} &= r_i - \alpha_i A d_i, \\
\beta_{i+1} &= \frac{r_{i+1}^T r_{i+1}}{r_i^T r_i}, \\
d_{i+1} &= r_{i+1} + \beta_{i+1} d_i.
\end{aligned}$$

Conjugate Gradient Method eliminates the need to store old search vectors and reduces both the space complexity and time complexity per iteration from $O\left(n^2\right)$ to $O(m)$, where m is the number of nonzero entries of A.

An efficient implementation of the Conjugate Gradient Method is citepshewchuk1994introduction

```
$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow r$
$\delta_{new} \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta new$
While $i < i_{max}$ and $\delta > \varepsilon^2 \delta_0$ do
    $q \Leftarrow Ad$
    $\alpha \Leftarrow \frac{\delta_{new}}{d^T q}$
    $x \Leftarrow x + \alpha d$
    $r \Leftarrow r - \alpha q$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow r^T r$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $d \Leftarrow r + \beta d$
    $i \Leftarrow i + 1$
```

where i_{max} is the maximum number of iterations and $\varepsilon < 1$ is the error tolerance.

This algorithm stops when the maximum number of iterations has been exceeded, or when $\|r_i\| \leq \varepsilon \|r_0\|$.

Appendix B

The FORTRAN Code Developed to Solve the Elliptical Instability Problem in a Tri-Axial Ellipsoid

```
PROGRAM LSFEM
USE OPM_LIB

IMPLICIT NONE

INTEGER :: i, j, k, m, n, nb_nodes, itt, nonitt=0, IERR, n_c=0, n_r=0, n_temp
INTEGER, DIMENSION(10) :: l=0
! Degrees of Freedom
INTEGER, PARAMETER :: nu=3940
! Number of Nodes
INTEGER, PARAMETER :: nn=985
! Nodes Vector
INTEGER, DIMENSION(nn) :: n_nodes=0
! X, Y, and Z Coordinates of Nodes
DOUBLE PRECISION, DIMENSION(985) :: x_nodes=0.0d0, y_nodes=0.0d0, z_nodes=0.0d0
! Array containing Node Numbers of Boundary Nodes
INTEGER, DIMENSION(nn) :: b_nodes=0
! Universal Coordinates of Nodes of an Element
DOUBLE PRECISION, DIMENSION(10) :: x=0.0d0,y=0.0d0,z=0.0d0
! Number of Elements
INETEGER, PARAMETER :: ne=586
! Connectivity Matrix
INTEGER, DIMENSION(ne,11) :: connectivity=0
! Stiffness Matrix
DOUBLE PRECISION, DIMENSION(:,:), ALLOCATABLE :: stiffness
! The Solution Vector
DOUBLE PRECISION, DIMENSION(nu) :: unknowns=0.0d0
! Force Vector
DOUBLE PRECISION, DIMENSION(nu) :: force=0.0d0
DOUBLE PRECISION, DIMENSION(nu) :: diff=0.0d0
DOUBLE PRECISION :: max_diff,max_diff_old=0.0d0, diff1=0.0d0, diff2=0.0d0
! Initial Velocity and Pressure
DOUBLE PRECISION, DIMENSION(nu) :: unknowns0=0.0d0
! Velocities and Pressure at n+1_k step
DOUBLE PRECISION, DIMENSION(nu) :: unknowns_n1_k=0.0d0
! Velocities and Pressure at n step
DOUBLE PRECISION, DIMENSION(nu) :: unknowns_n=0.0d0
! Nodal Value of Velocities in an Element at n+1_k Step
DOUBLE PRECISION, DIMENSION(10) :: uele_n1_k=0.0d0, vele_n1_k=0.0d0, wele_n1_k=0.0d0,
pele_n1_k=0.0d0
! Nodal Value of Unknows in an Element at n step
DOUBLE PRECISION, DIMENSION(10) :: uele_n=0.0d0, vele_n=0.0d0, wele_n=0.0d0,
pele_n=0.0d0
! Stiffness Matrix for a Node
DOUBLE PRECISION, DIMENSION(4,4) :: kij=0.0d0
! Force Vector for a Node
DOUBLE PRECISION, DIMENSION(4) :: fi=0.0d0
! Conversion Matrix for Imposing Boundary Condition
DOUBLE PRECISION, DIMENSION(4,4) :: conv=0.0d0
DOUBLE PRECISION, PARAMETER :: PI=2*DACOS(0.0d0)
! Temporary Matrices for Imposing Boundary Condition on Stiffness Matrix
DOUBLE PRECISION, DIMENSION(4,4) :: bnode_stiff=0.0d0, temp_stiff=0.0d0
! Temporary Vector for Imposing Boundary Condition on Force Vector
DOUBLE PRECISION, DIMENSION(4) :: bnode_force=0.0d0, temp_force=0.0d0
! Temporary Arrays for Converting Back the Nodal Values to Cartesian Coordinates
DOUBLE PRECISION, DIMENSION(4) :: bnode_unk=0.0d0, temp_unk=0.0d0
! Angles for Converting to Curvilinear Coordinates
DOUBLE PRECISION :: phi=0.0d0, alpha=0.0d0
! Equatorial Ellipticity
DOUBLE PRECISION :: eps=0.1d0
CHARACTER(500) :: filename = ‘ ’
INTEGER (KIND=4) it, it_max, job
DOUBLE PRECISION :: tol, bnrm2, rnrm2
DOUBLE PRECISION :: p (nu), q (nu), r (nu), zz (nu)
INTEGER, DIMENSION(:,:), ALLOCATABLE :: ij
DOUBLE PRECISION, DIMENSION(:), ALLOCATABLE :: values
INTEGER, DIMENSION(:), ALLOCATABLE :: col, row
DOUBLE PRECISION, DIMENSION(nu) :: temp
DOUBLE PRECISION:: d0
DOUBLE PRECISION :: p_k=0.0d0, p_k1=0.0d0, diff_p, diff_p_old=0.0d0

write(*,*) ‘The Program Starts’

OPEN(UNIT=8, FILE=‘/global/scratch/moradi/NBR0.00/nodes.data’, STATUS=‘OLD’, AC-
TION=‘READ’)
```

```
OPEN(UNIT=9, FILE=‘/global/scratch/moradi/NBR0.00/elements.data’, STATUS=‘OLD’,
ACTION=‘READ’)

DO k=1,985
READ(8,*) n_nodes(k),x_nodes(k),y_nodes(k),z_nodes(k)
END DO

write(*,*) ‘Done Reading Nodes Coordinates’

CLOSE(UNIT=8)

DO k=1,586
READ(9,*) connectivity(k,1), connectivity(k,2), connectivity(k,3), connectivity(k,4), con-
nectivity(k,5), connectivity(k,6), connectivity(k,7), connectivity(k,8), connectivity(k,9), con-
nectivity(k,10), connectivity(k,11)
END DO

write(*,*) ‘Done Reading Connectivity Matrix’

CLOSE(UNIT=9)

i=0
DO k=1,985
IF (ABS(1-(x_nodes(k)**2+y_nodes(k)**2+z_nodes(k)**2))<=0.00001) THEN
i=i+1
b_nodes(i)=n_nodes(k)
END IF
END DO
nb_nodes=i
write(*,*) ‘Nubmber of boundary Nodes’, nb_nodes
write(*,*) ‘Done Finding Boundary Nodes’

!$omp parallel do default(none) shared(unknowns0,y_nodes,x_nodes,eps) private(k)
DO k=1,985
unknowns0(k*4-3)=-y_nodes(k)
unknowns0(k*4-2)=x_nodes(k)
unknowns0(k*4-1)=0.0d0
unknowns0(k*4)=1.0d0/2.0d0*((1+eps)*x_nodes(k)**2+(1-eps)*y_nodes(k)**2)
END DO
!$omp end parallel do

write(*,*) ‘Done Calculating Initial Velocities and Pressure’

OPEN(UNIT=13, FILE=‘/global/scratch/moradi/NBR0.00/data00000.data’, STATUS=‘REPLACE’,
ACTION=‘WRITE’)
```

```
DO k=1,985
WRITE(13,102) k, unknowns0(k*4-3), unknowns0(k*4-2), unknowns0(k*4-1), unknowns0(k*4)
102 FORMAT(‘ ’,I5,4F40.20)
END DO
CLOSE(UNIT=13)

OPEN(UNIT=13, FILE=‘/global/scratch/moradi/NBR0.00/unknowns_n.data’, STATUS=‘REPLACE’,
ACTION=‘WRITE’)
DO k=1,985
WRITE(13,103) unknowns0(k*4-3), unknowns0(k*4-2), unknowns0(k*4-1), unknowns0(k*4)
103 FORMAT(‘ ’,4F30.20)
END DO
CLOSE(UNIT=13)

OPEN(UNIT=13, FILE=‘/global/scratch/moradi/NBR0.00/unknowns_n1_k.data’, STATUS=‘REPLACE’,
ACTION=‘WRITE’)
DO k=1,985
WRITE(13,103) unknowns0(k*4-3), unknowns0(k*4-2), unknowns0(k*4-1), unknowns0(k*4)
END DO
CLOSE(UNIT=13)

itt=1
ittloop: DO

ALLOCATE(stiffness(3940,3940), stat=IERR)
IF (IERR /= 0) THEN
WRITE(*,*) ’allocation failed’
PAUSE
END IF

OPEN(UNIT=13, FILE=‘/global/scratch/moradi/NBR0.00/unknowns_n.data’, STATUS=‘OLD’,
ACTION=‘READ’)
DO k=1,985
READ(13,*) unknowns_n(k*4-3), unknowns_n(k*4-2), unknowns_n(k*4-1), unknowns_n(k*4)
END DO
CLOSE(UNIT=13)
OPEN(UNIT=13, FILE=‘/global/scratch/moradi/NBR0.00/unknowns_n1_k.data’, STATUS=‘OLD’,
ACTION=‘READ’)
DO k=1,985
READ(13,*) unknowns_n1_k(k*4-3), unknowns_n1_k(k*4-2), unknowns_n1_k(k*4-1), un-
knowns_n1_k(k*4)
END DO
CLOSE(UNIT=13)
```

```
DO i=1,3940
DO j=1,3940
stiffness(i,j)=0.0d0
END DO
force(i)=0.0d0
END DO


! Calculating Universal Stiffness Matrix and Force Array
kloop: DO k=1,586

m1loop: DO m=1,10
l(m)=connectivity(k,m+1)
x(m)=x_nodes(l(m))
y(m)=y_nodes(l(m))
z(m)=z_nodes(l(m))
uele_n(m)=unknowns_n(l(m)*4-3)
vele_n(m)=unknowns_n(l(m)*4-2)
wele_n(m)=unknowns_n(l(m)*4-1)
pele_n(m)=unknowns_n(l(m)*4)
uele_n1_k(m)=unknowns_n1_k(l(m)*4-3)
vele_n1_k(m)=unknowns_n1_k(l(m)*4-2)
wele_n1_k(m)=unknowns_n1_k(l(m)*4-1)
END DO m1loop

iloop: DO i=1,10
jloop: DO j=1,10

CALL Stiffness_Force(i,j,x,y,z,uele_n1_k,vele_n1_k,wele_n1_k,uele_n,vele_n,wele_n,pele_n,kij,fi)

m2loop: DO m=1,4
nloop: DO n=1,4
stiffness(l(i)*4+m-4,l(j)*4+n-4)=stiffness(l(i)*4+m-4,l(j)*4+n-4)+kij(m,n)
END DO nloop
END DO m2loop

END DO jloop

m3loop: DO m=1,4
force(l(i)*4+m-4)=force(l(i)*4+m-4)+fi(m)
END DO m3loop

END DO iloop

END DO kloop
```

```
write(*,*) 'Done Calculating Stiffness Matrix'

! Imposing The Boundary Condition
k2loop: DO k=1,nb_nodes

i=b_nodes(k)
alpha=DACOS(z_nodes(i))

outer: IF ((ABS(x_nodes(i)) <= 1.0d-7) .AND. (y_nodes(i) > 1.0d-7)) THEN
phi=PI/2.0d0

ELSE IF ((ABS(x_nodes(i)) <= 1.0d-7) .AND. (y_nodes(i) < -1.0d-7)) THEN
phi=3.0d0*PI/2.0d0

ELSE IF ((ABS(x_nodes(i)) <= 1.0d-7) .AND. (ABS(y_nodes(i)) <= 1.0d-7)) THEN
phi=0.0d0

ELSE

inner: IF ((x_nodes(i) > 0.0d0) .AND. (y_nodes(i) >= 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))

ELSE IF ((x_nodes(i) < 0.0d0) .AND. (y_nodes(i) <= 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))+PI

ELSE IF ((x_nodes(i) > 0.0d0) .AND. (y_nodes(i) < 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))+2.0d0*PI

ELSE IF ((x_nodes(i) < 0.0d0) .AND. (y_nodes(i) > 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))+PI

END IF inner

END IF outer

conv(1,1)=SIN(alpha)*COS(phi)
conv(1,2)=COS(alpha)*COS(phi)
conv(1,3)=-SIN(phi)
conv(1,4)=0.0d0

conv(2,1)=SIN(alpha)*SIN(phi)
conv(2,2)=COS(alpha)*SIN(phi)
conv(2,3)=COS(phi)
conv(2,4)=0.0d0
conv(3,1)=COS(alpha)
conv(3,2)=-SIN(alpha)
conv(3,3)=0.0d0
conv(3,4)=0.0d0

conv(4,1)=0.0d0
conv(4,2)=0.0d0
conv(4,3)=0.0d0
conv(4,4)=1.0d0

j2loop: DO j=1,985

DO m=1,4
DO n=1,4
bnode_stiff(m,n)=stiffness(4*i+m-4,4*j+n-4)
END DO
END DO

temp_stiff=MATMUL(TRANSPOSE(conv),bnode_stiff)

DO m=1,4
DO n=1,4
stiffness(4*i+m-4,4*j+n-4)=temp_stiff(m,n)
END DO
END DO

END DO j2loop

j3loop: DO j=1,985

DO m=1,4
DO n=1,4
bnode_stiff(m,n)=stiffness(4*j+m-4,4*i+n-4)
END DO
END DO

temp_stiff=MATMUL(bnode_stiff,conv)

DO m=1,4
DO n=1,4
stiffness(4*j+m-4,4*i+n-4)=temp_stiff(m,n)
END DO
END DO
END DO j3loop

DO m=1,4
bnode_force(m)=force(4*i+m-4)
END DO

temp_force=MATMUL(TRANSPOSE(conv),bnode_force)

DO m=1,4
force(4*i+m-4)=temp_force(m)
END DO

END DO k2loop

! Zeroring Boundary Elements in Stiffness Matrix
!$omp parallel do default(none) shared(nb_nodes,b_nodes,stiffness,force) &
!$omp private(k,i,j)
DO k=1,nb_nodes

i=b_nodes(k)
DO j=1,3940
IF (j == 4*i-3) THEN
stiffness(4*i-3,j)=1.0d0
ELSE
stiffness(4*i-3,j)=0.0d0
END IF
END DO

DO j=1,3940
IF (j == 4*i-3) THEN
stiffness(j,4*i-3)=1.0d0
ELSE
stiffness(j,4*i-3)=0.0d0
END IF
END DO

force(4*i-3)=0.0d0
END DO
!$omp end parallel do

! Finding the Indices of the Nonzero Entries of Stiffness Matrix and Force Vector
IF (itt==1 .AND. nonitt==0) THEN

DO i=1,3940
n_temp=0
```

```
DO j=1,3940
IF (ABS(stiffness(i,j))>=1.0e-10) THEN
n_c=n_c+1
n_temp=n_temp+1
IF (n_temp==1) THEN
n_r=n_r+1
END IF
END IF
IF ((j==3940) .AND. (n_temp==0)) THEN
n_c=n_c+1
n_r=n_r+1
END IF
END DO
END DO
n_r=n_r+1

ALLOCATE(ij(n_c,2), values(n_c), col(n_c), row(n_r))

k=0
m=0
DO i=1,3940
n_temp=0
DO j=1,3940
IF (ABS(stiffness(i,j))>=1.0e-10) THEN
k=k+1
ij(k,1)=i
ij(k,2)=j
col(k)=j
n_temp=n_temp+1
IF (n_temp==1) THEN
m=m+1
row(m)=k
END IF
END IF
IF (j==SIZE(stiffness(:,1)) .AND. (n_temp==0)) THEN
ij(k,1)=i
ij(k,2)=j
col(k)=j
m=m+1
row(m)=k
END IF
END DO
END DO
row(n_r)=k+1
IF (n_r-1/=nu) THEN
```

```
write(*,*) 'n_r-1 not equal to nu'
stop
END IF

END IF

!$omp parallel do default(none) shared(values,stiffness,ij,n_c) private(i)
DO i=1,n_c
values(i)=stiffness(ij(i,1),ij(i,2))
END DO
!$omp end parallel do

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!! The Implemented Conjugate Gradient Algorthm !!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!$omp parallel shared(nu,unknowns,unknowns_n)
!$omp workshare
unknowns(1:nu)=unknowns_n1_k(1:nu)
!$omp end workshare
!$omp end parallel

! Parameters for the stopping test.
it = 0
it_max = nu*4
tol = 1.0D-6
bnrm2 = sqrt ( sum ( force(1:nu)**2 ) )

! Set parameters for the CG code.
job = 1

! Repeatedly call the CG code, and on return, do what JOB tells you.
DO

CALL cg ( nu, force, unknowns, r, zz, p, q, job )

! Compute q = A * p.
IF ( job == 1 ) THEN

!$omp parallel do private(i,j,d0)
DO i=1,n_r-1
d0=0.0e0
DO j=row(i),row(i+1)-1
d0=d0+values(j)*p(col(j))
```

```
END DO
q(i)=d0
END DO
!$omp end parallel do

! Solve M * z = r
ELSE IF ( job == 2 ) THEN

!$omp parallel do private(i)
DO i = 1, nu
zz(i) = r(i) / stiffness(i,i)
END DO
!$omp end parallel do

! Compute r = r - A * x.
ELSE IF ( job == 3 ) THEN

!$omp parallel do private(i,j,d0)
DO i=1,n_r-1
d0=0.0e0
DO j=row(i),row(i+1)-1
d0=d0+values(j)*unknowns(col(j))
END DO
temp(i)=d0
END DO
!$omp end parallel do

!$omp parallel shared(nu,r,temp)
!$omp workshare
r(1:nu) = r(1:nu) - temp(1:nu)
!$omp end workshare
!$omp end parallel

! Stopping test.
ELSE IF ( job == 4 ) THEN

rnrm2 = sqrt ( sum ( r(1:nu)**2 ) )
IF ( bnrm2 == 0.0D+00 ) THEN
IF ( rnrm2 <= tol ) THEN
EXIT
END IF
ELSE
IF ( rnrm2 <= tol * bnrm2 ) THEN
EXIT
END IF
END IF

it = it + 1

IF ( it_max <= it ) THEN
write ( *, '(a)' ) ' '
write ( *, '(a)' ) ' Iteration limit exceeded.'
write ( *, '(a)' ) ' Terminating early.'
EXIT ittloop
END IF

END IF

job = 2

END DO

write ( *, '(a)' ) ' '
write ( *, '(a,i5)' ) ' Number of iterations was ', it
write ( *, '(a,g14.6)' ) ' Estimated error is ', rnrm2

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! The Equation are Solved !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Converting the Velocities of Boundary Nodes Back to Cartesian Coordinates
!$omp parallel do default(none) shared(nb_nodes,b_nodes,x_nodes,y_nodes,z_nodes,unknowns)
&
!$omp private(k,i,alpha,phi,conv,m,bnode_unk,temp_unk)
k3loop: DO k=1,nb_nodes

i=b_nodes(k)
alpha=DACOS(z_nodes(i))

outer2: IF ((ABS(x_nodes(i)) <= 1.0d-7) .AND. (y_nodes(i) > 1.0d-7)) THEN
phi=PI/2.0d0

ELSE IF ((ABS(x_nodes(i)) <= 1.0d-7) .AND. (y_nodes(i) < -1.0d-7)) THEN
phi=3.0d0*PI/2.0d0

ELSE IF ((ABS(x_nodes(i)) <= 1.0d-7) .AND. (ABS(y_nodes(i)) <= 1.0d-7)) THEN
phi=0.0d0
```

```
ELSE

inner2: IF ((x_nodes(i) > 0.0d0) .AND. (y_nodes(i) >= 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))

ELSE IF ((x_nodes(i) < 0.0d0) .AND. (y_nodes(i) <= 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))+PI

ELSE IF ((x_nodes(i) > 0.0d0) .AND. (y_nodes(i) < 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))+2.0d0*PI

ELSE IF ((x_nodes(i) < 0.0d0) .AND. (y_nodes(i) > 0.0d0)) THEN
phi=DATAN(y_nodes(i)/x_nodes(i))+PI

END IF inner2

END IF outer2

conv(1,1)=SIN(alpha)*COS(phi)
conv(1,2)=COS(alpha)*COS(phi)
conv(1,3)=-SIN(phi)
conv(1,4)=0.0d0

conv(2,1)=SIN(alpha)*SIN(phi)
conv(2,2)=COS(alpha)*SIN(phi)
conv(2,3)=COS(phi)
conv(2,4)=0.0d0

conv(3,1)=COS(alpha)
conv(3,2)=-SIN(alpha)
conv(3,3)=0.0d0
conv(3,4)=0.0d0

conv(4,1)=0.0d0
conv(4,2)=0.0d0
conv(4,3)=0.0d0
conv(4,4)=1.0d0

DO m=1,4
bnode_unk(m)=unknowns(4*i+m-4)
END DO

temp_unk=MATMUL(conv,bnode_unk)

DO m=1,4
```

```
unknowns(4*i+m-4)=temp_unk(m)
END DO

END DO k3loop
!$omp end parallel do
write(*,*) 'Done Reversing Boundary Condition'

!$omp parallel do default(none) shared(unknowns,unknowns_n1_k) private(k) reduction(+:diff1,diff2)
DO k=1,985
diff1=diff1+(unknowns(k*4-3)-unknowns_n1_k(k*4-3))**2+(unknowns(k*4-2)-unknowns_n1_k(k*4-
2))**2&
+(unknowns(k*4-1)-unknowns_n1_k(k*4-1))**2+(unknowns(k*4)-unknowns_n1_k(k*4))**2
diff2=diff2+unknowns(k*4-3)**2+unknowns(k*4-2)**2+unknowns(k*4-1)**2+unknowns(k*4)**2
END DO
!$omp end parallel do

max_diff=DSQRT(diff1/diff2)
diff1=0.0d0
diff2=0.0d0
diff_p=-1.0d-4

IF (max_diff <= 0.00005 .and. nonitt >= 2) THEN
max_diff_old=max_diff
max_diff=1.0d-8
END IF

OPEN(UNIT=13, FILE='/global/scratch/moradi/NBR0.00/unknowns_n1_k.data', STATUS='REPLACE',
ACTION='WRITE')
DO k=1,985
WRITE(13,111) unknowns(k*4-3),unknowns(k*4-2),unknowns(k*4-1),unknowns(k*4)
111 FORMAT(' ',4F40.20)
END DO
CLOSE(UNIT=13)

IF (max_diff <= 1.0d-5 .AND. diff_p<0.0d0) THEN
OPEN(UNIT=13, FILE='/global/scratch/moradi/NBR0.00/unknowns_n.data', STATUS='REPLACE',
ACTION='WRITE')
DO k=1,985
WRITE(13,112) unknowns(k*4-3),unknowns(k*4-2),unknowns(k*4-1),unknowns(k*4)
112 FORMAT(' ',4F40.20)
END DO
CLOSE(UNIT=13)

write(filename,'(a,i5.5,a)') '/global/scratch/moradi/NBR0.00/data', itt, '.data'
OPEN(UNIT=13,FILE=filename,STATUS='REPLACE',ACTION='WRITE')
```

```
DO k=1,985
WRITE(13,106) k, unknowns(k*4-3), unknowns(k*4-2), unknowns(k*4-1), unknowns(k*4)
106 FORMAT(‘ ’,I5, 4F40.20)
END DO
CLOSE(UNIT=13)

IF (itt==5000) EXIT ittloop

write(*,*) ‘Itteration Number: ’, itt, max_diff
IF (max_diff_old /= max_diff) THEN
write(*,*) ‘Old Max Diff: ’, max_diff_old
END IF
itt=itt+1
nonitt=0

ELSE
nonitt=nonitt+1
WRITE(*,*) ‘Time-Marching Iteration Number: ’, itt
WRITE(*,*) ‘Linearization Iteration Number: ’, nonitt, max_diff

END IF

DEALLOCATE(stiffness)

END DO ittloop

END PROGRAM LSFEM

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!! Subroutine for Calculating Stiffness Matrix and Force Array !!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE Stiffness_Force(i, j, x, y, z, uele_n1_k, vele_n1_k, wele_n1_k, uele_n, vele_n,
wele_n, pele_n, kij, fi)

IMPLICIT NONE

INTEGER, INTENT(IN) :: i,j
INTEGER :: k,l,m
! Universal Coordinates of Nodes of an Element
DOUBLE PRECISION, DIMENSION(10),INTENT(IN) :: x, y, z
! Interpolation Functions
DOUBLE PRECISION, DIMENSION(10) :: psi=0.0d0
```

```
! Natural Coordinates
DOUBLE PRECISION, DIMENSION(4) :: zeta=0.0d0
! Gaussian Points
DOUBLE PRECISION, DIMENSION(4,3) :: gp=0.0d0
DOUBLE PRECISION, DIMENSION(3) :: lc=0.0d0
! Derivatives of Interpolation Functions with Respect to Universal Coordinates
DOUBLE PRECISION, DIMENSION(10,3) :: dpsidlc=0.0d0
DOUBLE PRECISION, DIMENSION(10) :: dpsidx=0.0d0, dpsidy=0.0d0, dpsidz=0.0d0
DOUBLE PRECISION, DIMENSION(3,10) :: A=0.0d0
DOUBLE PRECISION, DIMENSION(10,3) :: B=0.0d0
! Jacobian Matrix and its Inverse
DOUBLE PRECISION, DIMENSION(3,3) :: Jac=0.0d0, Jinv=0.0d0
! Determinant of Jacobian Matrix
DOUBLE PRECISION :: detJ
! Geometry Properties
DOUBLE PRECISION :: eps=0.1d0, chi=1.0d0
! Finite Direference Coefficient
DOUBLE PRECISION :: theta=0.5d0
! Time Step
DOUBLE PRECISION :: delta=1.0d0
! Nodal Values of Velocities in an Element at n+1_k Step
DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: uele_n1_k, vele_n1_k, wele_n1_k
! Nodal Values of Unknows in an Element at n Step
DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: uele_n, vele_n, wele_n, pele_n
! Value of Velocities in an Element at n+1_k Step
DOUBLE PRECISION :: u_n1_k=0.0d0, v_n1_k=0.0d0, w_n1_k=0.0d0
! Value of Unknows in an Element at Step n
DOUBLE PRECISION :: u_n=0.0d0, v_n=0.0d0, w_n=0.0d0, p_n=0.0d0
! Derivatives of Velocities in an Element at n+1_k Step
DOUBLE PRECISION :: dudx_n1_k=0.0d0, dudy_n1_k=0.0d0, dudz_n1_k=0.0d0, dvdx_n1_k=0.0d0,
dvdy_n1_k=0.0d0&
, dvdz_n1_k=0.0d0, dwdx_n1_k=0.0d0, dwdy_n1_k=0.0d0, dwdz_n1_k=0.0d0
! Derivative of Velocities and Pressure in an Element at Step n
DOUBLE PRECISION :: dudx_n=0.0d0, dudy_n=0.0d0, dudz_n=0.0d0, dvdx_n=0.0d0,
dvdy_n=0.0d0, dvdz_n=0.0d0&
, dwdx_n=0.0d0, dwdy_n=0.0d0, dwdz_n=0.0d0, dpdx_n=0.0d0, dpdy_n=0.0d0, dpdz_n=0.0d0
DOUBLE PRECISION :: x_ele=0.0d0,y_ele=0.0d0
! Stifness Matrix
DOUBLE PRECISION, DIMENSION(4,4), INTENT(OUT) :: kij
! Force Vector
DOUBLE PRECISION, DIMENSION(4), INTENT(OUT) :: fi
DOUBLE PRECISION, DIMENSION(4,4) :: a_psi_i, a_psi_j
DOUBLE PRECISION, DIMENSION(4) :: f
! Frequency ration of the Background Rotation
DOUBLE PRECISION :: NBR=0.00d0
```

```
DO k=1,4
DO l=1,4
kij(k,l)=0.0d0
END DO
fi(k)=0.0d0
END DO

gp(1,1)=0.138196601125011d0
gp(1,2)=0.138196601125011d0
gp(1,3)=0.138196601125011d0

gp(2,1)=0.585410196624969d0
gp(2,2)=0.138196601125011d0
gp(2,3)=0.138196601125011d0

gp(3,1)=0.138196601125011d0
gp(3,2)=0.585410196624969d0
gp(3,3)=0.138196601125011d0

gp(4,1)=0.138196601125011d0
gp(4,2)=0.138196601125011d0
gp(4,3)=0.585410196624969d0

gaussloop: DO m=1,4

DO l=1,3
lc(l)=gp(m,l)
END DO

zeta(1)=1-lc(1)-lc(2)-lc(3)
zeta(2)=lc(1)
zeta(3)=lc(2)
zeta(4)=lc(3)

! Interpolation Functions and Their Derivatives
psi(1)=zeta(1)*(2.0d0*zeta(1)-1)
psi(2)=zeta(2)*(2.0d0*zeta(2)-1)
psi(3)=zeta(3)*(2.0d0*zeta(3)-1)
psi(4)=zeta(4)*(2.0d0*zeta(4)-1)
psi(5)=4.0d0*zeta(1)*zeta(2)
psi(6)=4.0d0*zeta(2)*zeta(3)
psi(7)=4.0d0*zeta(3)*zeta(1)
psi(8)=4.0d0*zeta(1)*zeta(4)
psi(9)=4.0d0*zeta(2)*zeta(4)
psi(10)=4.0d0*zeta(3)*zeta(4)

dpsidlc(1,1)=-3.0d0+4.0d0*lc(1)+4.0d0*lc(2)+4.0d0*lc(3)
dpsidlc(1,2)=-3.0d0+4.0d0*lc(1)+4.0d0*lc(2)+4.0d0*lc(3)
dpsidlc(1,3)=-3.0d0+4.0d0*lc(1)+4.0d0*lc(2)+4.0d0*lc(3)

dpsidlc(2,1)=4.0d0*lc(1)-1.0d0
dpsidlc(2,2)=0
dpsidlc(2,3)=0.0d0

dpsidlc(3,1)=0.0d0
dpsidlc(3,2)=4.0d0*lc(2)-1.0d0
dpsidlc(3,3)=0.0d0

dpsidlc(4,1)=0.0d0
dpsidlc(4,2)=0.0d0
dpsidlc(4,3)=4.0d0*lc(3)-1.0d0

dpsidlc(5,1)=4.0d0-8.0d0*lc(1)-4.0d0*lc(2)-4.0d0*lc(3)
dpsidlc(5,2)=-4.0d0*lc(1)
dpsidlc(5,3)=-4.0d0*lc(1)

dpsidlc(6,1)=4.0d0*lc(2)
dpsidlc(6,2)=4.0d0*lc(1)
dpsidlc(6,3)=0.0d0

dpsidlc(7,1)=-4.0d0*lc(2)
dpsidlc(7,2)=4.0d0-4.0d0*lc(1)-8.0d0*lc(2)-4.0d0*lc(3)
dpsidlc(7,3)=-4.0d0*lc(2)

dpsidlc(8,1)=-4.0d0*lc(3)
dpsidlc(8,2)=-4.0d0*lc(3)
dpsidlc(8,3)=4.0d0-4.0d0*lc(1)-4.0d0*lc(2)-8.0d0*lc(3)

dpsidlc(9,1)=4.0d0*lc(3)
dpsidlc(9,2)=0.0d0
dpsidlc(9,3)=4.0d0*lc(1)

dpsidlc(10,1)=0.0d0
dpsidlc(10,2)=4.0d0*lc(3)
dpsidlc(10,3)=4.0d0*lc(2)

! Jacobian Matrix and Its Determinant

DO k=1,3
DO l=1,10
A(k,l)=dpsidlc(l,k)
END DO
END DO

DO k=1,10
B(k,1)=x(k)
B(k,2)=y(k)
B(k,3)=z(k)
END DO

Jac=MATMUL(A,B)

! Calling the Subroutine to Calculate Inverse of Jacobian

CALL M33INV(Jac, Jinv, detJ)

! Derivatives of Interpolation Functions with Respect to Universal Coordinates

DO k=1,10
dpsidx(k)=Jinv(1,1)*dpsidlc(k,1)+Jinv(1,2)*dpsidlc(k,2)+Jinv(1,3)*dpsidlc(k,3)
dpsidy(k)=Jinv(2,1)*dpsidlc(k,1)+Jinv(2,2)*dpsidlc(k,2)+Jinv(2,3)*dpsidlc(k,3)
dpsidz(k)=Jinv(3,1)*dpsidlc(k,1)+Jinv(3,2)*dpsidlc(k,2)+Jinv(3,3)*dpsidlc(k,3)
END DO

! Calling Subroutines to Calculate the Velocities and Pressure in an Element

CALL uelem(uele_n1_k,psi,u_n1_k)
CALL uelem(vele_n1_k,psi,v_n1_k)
CALL uelem(wele_n1_k,psi,w_n1_k)

CALL uelem(uele_n,psi,u_n)
CALL uelem(vele_n,psi,v_n)
CALL uelem(wele_n,psi,w_n)
CALL uelem(pele_n,psi,p_n)

CALL uelem(x,psi,x_ele)
CALL uelem(y,psi,y_ele)

! Calling Subroutines to Calculate Derivatives of Velocities and Pressure in an Element

CALL dudx(dpsidx,uele_n1_k,dudx_n1_k)
CALL dudy(dpsidy,uele_n1_k,dudy_n1_k)
CALL dudz(dpsidz,uele_n1_k,dudz_n1_k)
CALL dudx(dpsidx,vele_n1_k,dvdx_n1_k)
CALL dudy(dpsidy,vele_n1_k,dvdy_n1_k)
CALL dudz(dpsidz,vele_n1_k,dvdz_n1_k)

CALL dudx(dpsidx,wele_n1_k,dwdx_n1_k)
CALL dudy(dpsidy,wele_n1_k,dwdy_n1_k)
CALL dudz(dpsidz,wele_n1_k,dwdz_n1_k)

CALL dudx(dpsidx,uele_n,dudx_n)
CALL dudy(dpsidy,uele_n,dudy_n)
CALL dudz(dpsidz,uele_n,dudz_n)

CALL dudx(dpsidx,vele_n,dvdx_n)
CALL dudy(dpsidy,vele_n,dvdy_n)
CALL dudz(dpsidz,vele_n,dvdz_n)

CALL dudx(dpsidx,wele_n,dwdx_n)
CALL dudy(dpsidy,wele_n,dwdy_n)
CALL dudz(dpsidz,wele_n,dwdz_n)

CALL dudx(dpsidx,pele_n,dpdx_n)
CALL dudy(dpsidy,pele_n,dpdy_n)
CALL dudz(dpsidz,pele_n,dpdz_n)

! Stiffness Matrix

a_psi_i(1,1)=dpsidx(i)
a_psi_i(1,2)=theta*(u_n1_k*dpsidx(i)+v_n1_k*dpsidy(i)+w_n1_k*dpsidz(i))+(delta+theta*dudx_n1_k)*psi(
a_psi_i(1,3)=theta*(dvdx_n1_k+2.0d0*DSQRT((1.0d0+eps)/(1.0d0-eps))*NBR)*psi(i)
a_psi_i(1,4)=theta*dwdx_n1_k*psi(i)

a_psi_i(2,1)=dpsidy(i)
a_psi_i(2,2)=theta*(dudy_n1_k-2.0d0*DSQRT((1.0d0-eps)/(1.0d0+eps))*NBR)*psi(i)
a_psi_i(2,3)=theta*(u_n1_k*dpsidx(i)+v_n1_k*dpsidy(i)+w_n1_k*dpsidz(i))+(delta+theta*dvdy_n1_k)*psi(
a_psi_i(2,4)=theta*dwdy_n1_k*psi(i)

a_psi_i(3,1)=dpsidz(i)
a_psi_i(3,2)=theta*dudz_n1_k*psi(i)
a_psi_i(3,3)=theta*dvdz_n1_k*psi(i)
a_psi_i(3,4)=theta*(u_n1_k*dpsidx(i)+v_n1_k*dpsidy(i)+w_n1_k*dpsidz(i))+(delta+theta*dwdz_n1_k)*psi

a_psi_i(4,1)=0.0d0
a_psi_i(4,2)=theta/(1.0d0+eps)*dpsidx(i)
a_psi_i(4,3)=theta/(1.0d0-eps)*dpsidy(i)
a_psi_i(4,4)=theta/chi**2*dpsidz(i)
```

```
a_psi_j(1,1)=dpsidx(j)
a_psi_j(1,2)=dpsidy(j)
a_psi_j(1,3)=dpsidz(j)
a_psi_j(1,4)=0.0d0

a_psi_j(2,1)=theta*(u_n1_k*dpsidx(j)+v_n1_k*dpsidy(j)+w_n1_k*dpsidz(j))+(delta+theta*dudx_n1_k)*psi
a_psi_j(2,2)=theta*(dudy_n1_k-2.0d0*DSQRT((1.0d0-eps)/(1.0d0+eps))*NBR)*psi(j)
a_psi_j(2,3)=theta*dudz_n1_k*psi(j)
a_psi_j(2,4)=theta/(1.0d0+eps)*dpsidx(j)

a_psi_j(3,1)=theta*(dvdx_n1_k+2.0d0*DSQRT((1.0d0+eps)/(1.0d0-eps))*NBR)*psi(j)
a_psi_j(3,2)=theta*(u_n1_k*dpsidx(j)+v_n1_k*dpsidy(j)+w_n1_k*dpsidz(j))+(delta+theta*dvdy_n1_k)*psi
a_psi_j(3,3)=theta*dvdz_n1_k*psi(j)
a_psi_j(3,4)=theta/(1.0d0-eps)*dpsidy(j)

a_psi_j(4,1)=theta*dwdx_n1_k*psi(j)
a_psi_j(4,2)=theta*dwdy_n1_k*psi(j)
a_psi_j(4,3)=theta*(u_n1_k*dpsidx(j)+v_n1_k*dpsidy(j)+w_n1_k*dpsidz(j))+(delta+theta*dwdz_n1_k)*psi
a_psi_j(4,4)=theta/chi**2*dpsidz(j)

kij=kij+MATMUL(a_psi_i,a_psi_j)*detJ/24.0d0

! Force Vector

f(1)=0.0d0
f(2)=u_n*delta-(1.0d0-theta)*(u_n*dudx_n+v_n*dudy_n+w_n*dudz_n+1.0d0/(1.0d0+eps)*dpdx_n&
-2.0d0*DSQRT((1.0d0-eps)/(1.0d0+eps))*NBR*v_n)+theta*(u_n1_k*dudx_n1_k+v_n1_k*dudy_n1_k+w_n
f(3)=v_n*delta-(1.0d0-theta)*(u_n*dvdx_n+v_n*dvdy_n+w_n*dvdz_n+1.0d0/(1.0d0-eps)*dpdy_n&
+2.0d0*DSQRT((1.0d0+eps)/(1.0d0-eps))*NBR*u_n)+theta*(u_n1_k*dvdx_n1_k+v_n1_k*dvdy_n1_k+w_n
f(4)=w_n*delta-(1.0d0-theta)*(u_n*dwdx_n+v_n*dwdy_n+w_n*dwdz_n+(1.0d0/chi**2)*dpdz_n)&
+theta*(u_n1_k*dwdx_n1_k+v_n1_k*dwdy_n1_k+w_n1_k*dwdz_n1_k)

fi=fi+MATMUL(a_psi_i,f)*detJ/24.0d0

END DO gaussloop

END SUBROUTINE Stiffness_Force

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutine for Calculating U in an Element !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
SUBROUTINE uelem(uele,psi,u)

IMPLICIT NONE

DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: uele
DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: psi
DOUBLE PRECISION, INTENT(OUT) :: u
INTEGER :: k

u=0.0d0
DO k=1,10
u=u+uele(k)*psi(k)
END DO

END SUBROUTINE uelem

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutine for Calculating dudx in an Element !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE dudx(dpsidx,uele,dudx_out)

IMPLICIT NONE

DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: dpsidx, uele
DOUBLE PRECISION,INTENT(OUT) :: dudx_out
INTEGER :: k

dudx_out=0.0d0
DO k=1,10
dudx_out=dudx_out+uele(k)*dpsidx(k)
END DO

END SUBROUTINE dudx

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutine for Calculating dudy in an Element !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE dudy(dpsidy,uele,dudy_out)
```

```
IMPLICIT NONE

DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: dpsidy, uele
DOUBLE PRECISION,INTENT(OUT) :: dudy_out
INTEGER :: k

dudy_out=0.0d0
DO k=1,10
dudy_out=dudy_out+uele(k)*dpsidy(k)
END DO

END SUBROUTINE dudy

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutine for Calculating dudz in an Element !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE dudz(dpsidz,uele,dudz_out)

IMPLICIT NONE

DOUBLE PRECISION, DIMENSION(10), INTENT(IN) :: dpsidz, uele
DOUBLE PRECISION,INTENT(OUT) :: dudz_out
INTEGER :: k

dudz_out=0.0d0
DO k=1,10
dudz_out=dudz_out+uele(k)*dpsidz(k)
END DO

END SUBROUTINE dudz

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Conjugate Gradient Method Subroutine !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE cg ( n, b, x, r, z, p, q, job )
USE OMP_LIB

IMPLICIT NONE
```

```
integer ( kind = 4 ) n

real ( kind = 8 ) alpha
real ( kind = 8 ) b(n)
real ( kind = 8 ) beta
integer ( kind = 4 ) iter
integer ( kind = 4 ) job
real ( kind = 8 ) p(n)
real ( kind = 8 ) pdotq
real ( kind = 8 ) q(n)
real ( kind = 8 ) r(n)
real ( kind = 8 ) rho
real ( kind = 8 ) rho_old
integer ( kind = 4 ) rlbl
real ( kind = 8 ) x(n)
real ( kind = 8 ) z(n)

! Some local variables must be preserved between calls.
save iter
save rho
save rho_old
save rlbl

! Initialization.
! Ask the user to compute the initial residual.
if ( job == 1 ) then

!$omp parallel shared(n,b,r)
!$omp workshare
r(1:n) = b(1:n)
!$omp end workshare
!$omp end parallel

job = 3
rlbl = 2

! Begin first conjugate gradient loop.
! Ask the user for a preconditioner solve.
else if ( rlbl == 2 ) then

iter = 1

job = 2
rlbl = 3
```

```
! Compute the direction.
! Ask the user to compute ALPHA.
! Save A*P to Q.
else if ( rlbl == 3 ) then

rho = dot_product ( r, z )

if ( 1 < iter ) then
beta = rho / rho_old
!$omp parallel shared(n,z,p)
!$omp workshare
z(1:n) = z(1:n) + beta * p(1:n)
!$omp end workshare
!$omp end parallel
end if

!$omp parallel shared(n,p,z)
!$omp workshare
p(1:n) = z(1:n)
!$omp end workshare
!$omp end parallel

job = 1
rlbl = 4

! Compute current solution vector.
! Ask the user to check the stopping criterion.
else if ( rlbl == 4 ) then

pdotq = dot_product ( p, q )
alpha = rho / pdotq
!$omp parallel shared(n,x,p,r,q)
!$omp workshare
x(1:n) = x(1:n) + alpha * p(1:n)
r(1:n) = r(1:n) - alpha * q(1:n)
!$omp end workshare
!$omp end parallel

job = 4
rlbl = 5

! Begin the next step.
! Ask for a preconditioner solve.
else if ( rlbl == 5 ) then
```

```
rho_old = rho
iter = iter + 1

job = 2
rlbl = 3

end if

return
end

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!! Compute the Inverse and Determinant of a 3x3 Matrix !!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE M33INV (A, AINV, DET)

IMPLICIT NONE

DOUBLE PRECISION, DIMENSION(3,3), INTENT(IN) :: A
DOUBLE PRECISION, DIMENSION(3,3), INTENT(OUT) :: AINV
DOUBLE PRECISION, INTENT(OUT) :: DET

DOUBLE PRECISION, PARAMETER :: EPS = 0.1D-10
DOUBLE PRECISION, DIMENSION(3,3) :: COFACTOR


DET = A(1,1)*A(2,2)*A(3,3)-A(1,1)*A(2,3)*A(3,2)-A(1,2)*A(2,1)*A(3,3)+A(1,2)*A(2,3)*A(3,1)&
+A(1,3)*A(2,1)*A(3,2)-A(1,3)*A(2,2)*A(3,1)

IF (ABS(DET) .LE. EPS) THEN
AINV = 0.0D0
WRITE(*,*) 'Singular Jacobian Matrix'
RETURN
END IF

COFACTOR(1,1) = +(A(2,2)*A(3,3)-A(2,3)*A(3,2))
COFACTOR(1,2) = -(A(2,1)*A(3,3)-A(2,3)*A(3,1))
COFACTOR(1,3) = +(A(2,1)*A(3,2)-A(2,2)*A(3,1))
COFACTOR(2,1) = -(A(1,2)*A(3,3)-A(1,3)*A(3,2))
COFACTOR(2,2) = +(A(1,1)*A(3,3)-A(1,3)*A(3,1))
COFACTOR(2,3) = -(A(1,1)*A(3,2)-A(1,2)*A(3,1))
COFACTOR(3,1) = +(A(1,2)*A(2,3)-A(1,3)*A(2,2))
COFACTOR(3,2) = -(A(1,1)*A(2,3)-A(1,3)*A(2,1))
COFACTOR(3,3) = +(A(1,1)*A(2,2)-A(1,2)*A(2,1))

AINV = TRANSPOSE(COFACTOR) / DET

RETURN

END SUBROUTINE M33INV

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! Subroutine for Matrix Vector Multiplication !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

SUBROUTINE SpMxV(values,col,row,x,n_c,n_r,v)
IMPLICIT NONE

INTEGER, INTENT(IN) :: n_c, n_r
INTEGER :: i,j
DOUBLE PRECISION, DIMENSION(n_c), INTENT(IN) :: values
DOUBLE PRECISION :: d0
INTEGER, DIMENSION(n_c), INTENT(IN) :: col
INTEGER, DIMENSION(n_r), INTENT(IN) :: row
DOUBLE PRECISION, DIMENSION(3940), INTENT(IN) :: x
DOUBLE PRECISION, DIMENSION(3940), INTENT(OUT) :: v

DO i=1,n_r
d0=0.0e0
DO j=row(i),row(i+1)-1
d0=d0+values(j)*x(col(j))
END DO
v(i)=d0
END DO

END SUBROUTINE SpMxV
```