A FRAMEWORK FOR THE ANALYSIS OF COUPLED-PHYSICS MODELS USING ADAPTIVE MULTI-LEVEL TECHNIQUES

by

Erhan Turan

B.S., M.E., Boğaziçi University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Mechanical Engineering

Boğaziçi University

2010

canım anneme...

ACKNOWLEDGEMENTS

I would like to thank to my advisor Dr. Ali Ecder for his continuous support and guidance. I also appreciate the assistances of Dr. Can Özturan and Dr. Fazıl Önder Sönmez. I'm thankful to Dr. Ali Cemal Benim whom with I had the opportunity to deepen my knowledge in the area of fluid dynamics. Dr. Emre Aksan trusted me and also gave me a chance to look things differently; I appreciate it.

There are so many people who trained me throughout my journey. I can't thank them one by one but their names deserved to be pronounced so thank you, Dr. Akın Tezel, Dr. Arsev Eraslan, Veysel Durmuş, Gülfidan Karanfil, Suzan Osmanoğlu and Naci Kahveci. You made me to think freely.

What I did and what I'm going to do will not match with the unconditional support and love of my family; my mother Songül Yürek, my brother Erman Turan, my father Kenan Turan and also my dear Aunts. I'm happy to have such kind relatives. Truly, I always regarded my friends as a part of my family. Thank you, Özkan Aydın, Erkan Bağcivan, Altuğ Melik Başol, Yalın Kaptan, Hicran Kırılmaz, Gülsad Küçük, Mehmet Orhan, Can Özcan, Övül Özgü Özsoy, Miray Şimşek, Uğur Türk and Utku Cemal Ünlü. I'm so lucky to have such lovely friends like you guys.

And finally I want to say that I'm really grateful to all of you who never doubted me and believed in my dreams.

ABSTRACT

A FRAMEWORK FOR THE ANALYSIS OF COUPLED-PHYSICS MODELS USING ADAPTIVE MULTI-LEVEL TECHNIQUES

The object of this study is to develop a computational framework to analyze coupled-physics problems within the context of multi-level methods. Adaptive solution strategies in conjunction with Newton-Krylov and Domain Decomposition Methods are used to investigate different problems. Two model coupled-physics problems are selected for simulation; a fluid-structure interaction problem and a multiphase flow problem. First problem is on the deformation of a bimetallic strip exposed to natural convection. Two non-conforming and overlapping domains are created to handle the changes on the boundaries so that the deflection of the solid is applied only some portion of the fluid region. Displacements on the strip are calculated using decoupled thermoelasticity with plane strain assumption. In the second problem, collapse of a water column into the air is modeled. The interface is tracked using the Volume of Fluid method and the results are compared against experimental studies. To let the physics interact with each other and to unify different numerical solution methods, a solver called DEMONA (Decomposition Enhanced Mechanics Optimized Numerical Analysis) is developed which is verified on numerous benchmark problems. A new technique, based on an idea to reduce the solution sets is implemented into the solver, as well. With this methodology, the unknowns are filtered using various reduction criteria which are either applied in run-time or decided prior to the computations so that a specific solution approach is employed. Consequently, an adaptive structure is attained and different solution techniques are allowed to be tested with a single model definition.

ÖZET

BAĞLI-MODEL PROBLEMLERİNİN ADAPTİF ÇOKLU-AĞ YÖNTEMİ İLE ÇÖZÜLMESİ İÇİN SAYISAL BİR ALTYAPI

Bu çalışmanın amacı bağlı-model problemlerinin çoklu-seviye yöntemleri açısından analiz edilmesini sağlayan sayısal bir altyapının geliştirilmesidir. Çeşitli problemlerin incelenmesinde, Newton-Krylov ve Alan Ayrıştırma metotları ile bütünleşik adaptif çözüm stratejileri kullanılmıştır. İki bağlı-fizik problemi, bir akışkan-katı etkileşimi problemi ve bir çok fazlı akış problemi, simülasyon için seçildi. İlk problem doğal konveksiyona maruz kalmış iki metalli bir çubuğun şekil değiştirmesi üzerinedir. Uyumsuz ve örtüşen iki alan oluşturularak sınırlardaki değişikliklerin üstesinden gelinmiştir böylece katıdaki eğilmenin akış bölgesinin sadece bir kısmına uygulanması sağlanmıstır. Çubuk üstündeki deplasmanlar ayrışmış termoelastisite kullanarak düzlemsel gerinim kabulü ile hesaplanmıştır. İkinci problemde, bir su kütlesinin hava içine çöküşü modellenmiştir. Arayüz Akışkan Hacmi yöntemi ile takip edilmiştir ve sonuçlar deneysel çalışmalar ile karşılaştırılmıştır. Fiziklerin birbiri ile etkileşmesi ve farklı sayısal çözüm yöntemlerinin birleştirilmesi için muhtelif sayıda karşılaştırma problemleriyle doğrulanan DEMONA (Ayrıştırmayla Geliştirilmiş Mekanik Optime Sayısal Analiz), adında bir çözücü geliştirilmiştir. Çözüm kümelerinin kücültülmesine dayanan yeni bir teknik de çözücüye eklenmiştir. Bu yaklaşımla bilinmeyenler, ya calışma anında ya da hesaplamalardan önce kullanılacak özel çözüm yöntemine göre belirlenmiş çeşitli küçültme kriterleri ile süzülmüştür. Sonuç olarak, adaptif bir yapı sağlandı ve farklı çözüm yöntemlerinin tek bir model tanımı ile test edilmesi mümkün kılındı.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

A	Coefficient Matrix
B	Nodal Matrix
c_p	Specific heat
D	Material Stiffness Matrix
E	Young's Modulus
e	Error
F	Nonlinear Function
$\tilde{F}$	Phase Fractions
f_x	Forcing Function in x direction
f_y	Forcing Function in y direction
$\mathbf{g}$	Gravitational vector
g_y	Gravity in y direction
H	Hessenberg Matrix, dimensionless height
h	Spacing
K	Stiffness Matrix
k	Conductivity
J	Jacobian Matrix
l	Fill-in Parameter
M	Preconditioner
N	Number of Total Unknowns
Nu	Nusselt Number
P_L	Left Preconditioner
P_R	Right Preconditioner
Pr	Prandtl Number
p	Pressure
Re	Rayleigh Number
Re	Reynolds Number
r	Residual
T	Temperature, dimensionless time

t	Real time
u	Velocity in x direction
u_{wall}	Wall Velocity
$\mathbf{u}$	Velocity vector
$\mathbf{V}$	Velocity vector
V	Orthogonal Basis
v	Velocity in y direction
w	Velocity in z direction
x	Solution vector
Z	Dimensionless base length
α_b	Backtracking Parameter
α^d	Artificial interface BC control parameter
α_j	Evaluation parameter for the diagonal of the Jacobian
β	Artificial Compressibility
β_b	Backtracking Parameter
β_g	Norm of the initial residual in GMRES
ϵ	Eccentricity
ε	Perturbation Parameter
γ	Adaptive Forcing Factor Parameter
η	Forcing Parameter, machine epsilon
λ	Damping Parameter, Bulk Viscosity
μ	Viscosity
ν	Poisson's Ratio
Φ	Viscous dissipation
ψ	Stream-Function
ψ_{wall}	Wall Stream-Function
ρ	Density
$\mathcal{S}_G$	Global Set
$\mathcal{S}_L$	Local Set
ω	Vorticity

ω_x	X-Vorticity
ω_y	Y-Vorticity
ω_z	Z-Vorticity
Ω	Domain
τ	Pseudo Time
$\partial\Omega$	Boundary of the Domain

AC	Artificial Compressibility
AMG	Algebraic Multigrid
ASPIN	Additive Schwarz Preconditioned Inexact Newton
BC	Boundary Condition
BiCGSTAB	Bi-Conjugate Gradient Stabilized
CG	Conjugate Gradient Method
CRS	Compresses Row Storage
DDM	Domain Decomposition Method
DEMONA	Decomposition Enhanced Mechanics Optimized Numerical Analysis
DOF	Degree of Freedom
EBE	Element-by-Element
FAS	Full Approximation Scheme
FEA	Finite Elements Analysis
FMG	Full Multigrid
FSI	Fluid-Structure Interaction
GM	Geometric Multigrid
GMRES	Generalized Minimum Residual
GS	Gauss-Seidel
IM	Pointer to the dofMatrix of currentDomain
IMG	Pointer to the dofMatrix's of all domains in the globalSet
IML	Pointer to the dofMatrix's of all domains in the localSet
IMT	Pointer to the dofMatrix's of all domains in the totalSet
IN	Inexact Newton Method
IV	Pointer to the dofVector of currentDomain

MF	Matrix-Free
MG	Multigrid
ML	Multilevel
NGS	Nonlinear Gauss-Seidel
NK	Newton-Krylov
NKS	Newton-Krylov-Schwarz
NM	Newton's Method
NS	Nonlinear Solver (Smoother)
NSE	Navier-Stokes Equations
PBP	Physics Based Preconditioner
PDE	Partial Differential Equations
RK	Runge-Kutta
SFV	Stream Function - Vorticity
SNE	System of Nonlinear Equations
SOR	Successive Overrelaxation
VOF	Volume of Fluid

1. INTRODUCTION

The world around us, from its tiniest sand to the top of a huge mountain or from the dew on a leaf to a sudden shower, is a large system with all its elements and has its own rules. There is a continuous interaction between the components of the system: Sun heats the water; water turns into vapor; vapor becomes a cloud and as it rains, the Water Cycle is completed. As scientists, we are curious to understand the logic behind all phenomena. In reality, there is a complex mechanism behind the things we observe which look so complicated and interestingly so natural. For us, it is usual that heated water evaporates. In reality, water molecules speed up with increasing internal energy and after some point the connections between macro molecules are lost and water changes phase.

The logic we say is actually the physics and inevitable developments in the technology enforce the scientists to dig those physical phenomena more and more. It is a fact that no single real incident can be explained by single "physics". Since years, analysts, especially engineers, worked with simplified models in order to bring the complexity of the problems into reasonable i.e. solvable levels. Although, at first sight, such an approach fitted perfectly to several classical problems, there is a bunch of observations which cannot be explained with reduced models. Hence, more than one "physics" must be handled at the same time. Current advances in science give us the opportunity to simulate coupled-physics problems. However, the question is, how can we manage these improvements and utilize them to analyze multi-"physics" environments.

Multi-physics is a notion that leads to a unified model to simulate different physics. Fluid-Structure Interactions, combustion problems, aeroacoustics, etc. are some examples of coupled-physics problems. Aeroelasticity, for instance, is a Fluid-Structure Interaction (FSI) problem; which studies the effects aerodynamics forces, created by the flow field, on bodies which are elastic (Fung, 1969). Induction heating is another example in which flow of the current creates heat energy in the metal.

Real life applications like those two definitely involve more than one "physics". Even the simplest things that we define as natural are examples of coupled-physics events: as we pour water into a glass, a two-phase flow is happening that contains both air and water. Another example is the study of arteries for patient specific treatment. This problem includes the study of blood flow, as well as the deformation of the artery walls (Calo *et al.*, 2008). Blood pressure is the main reason of the tension on the arteries which affects the health conditions of the patient in the long run since a corrupt wall or embolism will deteriorate the blood flow. Hence, physics do have relations between them that designate the coupled phenomena. The level of coupling determines the complexity of the problem. If there is a strong coupling, then effective methods should be devised to cope with the problems. However, that would be just tailoring of specific solution method for that kind of multi-"physics" problem. Although such methods work best under those specified conditions, any other coupled-"physics" problem will require a new method, which will turn the solution process into a tedious nightmare. Universal techniques that can deal with these kinds of complicated systems can be quite useful. From that point of view, adaptive multi-level methods promise a sound theory to simulate multi-disciplinary formulation. Additionally, contemporary approaches, like Krylov subspace Methods or Multigrid formulation do reduce the computational time which is extremely useful in multi-physics problems.

Use of multi-level idea for solving partial differential equations is first introduced by Brandt (1977) although there was some preliminary work on this subject before. In his work, use of "multigrid" and also "adaptive" grids is suggested. Brandt defined the terminology of multigrid and also gave a convergence theory on multigrid as well as some techniques to use adaptive solution. This first paper is especially written for elliptic PDEs (Partial Differential Equations) *i.e.* the Poisson problem in Cartesian coordinates, but soon it is applied to a wide range of PDEs including the Navier-Stokes Equations.

Brandt also introduced the Full Approximation Scheme (FAS). Rather than applying multigrid to the linearized problems, FAS' intent is to overcome the difficulties of the nonlinear problems by handling them directly (Section 5.3). In FAS, fine grid

solutions are restricted to coarse grids and then a coarse grid guess is established. After the determination of the error, the fine grid solution is updated. That's the fundamental difference between the linear multigrid and FAS: in linear multigrid, the error is solved directly, but in FAS the coarse grid error is calculated from a coarse grid estimate.

Later, Brandt (1980) published a paper which discusses multi-level adaptive methods on fluid dynamics problems . He studied the incompressible Navier-Stokes problems discretized on staggered grids. Vanka (1986) studied Navier-Stokes Equations with primitive variables by utilizing Block-Implicit Multigrid method. He used Finite Differences to discretize the problem. SIMPLE method was utilized for computational model. From multi-level point of view, he used Full multigrid (FMG) with FAS. Later Bai and Brandt (1987) published an article about the application of multi-level on local mesh refinement. At that year, Briggs published the Multigrid tutorial, and introductory material for application and theory of multigrid which is updated later by other collaborators (Briggs *et al.*, 2000).

Rüde (1993) published a paper on fully adaptive multigrid techniques. For refinement purposes, he suggested a couple of methods, also Multi-Level Adaptive Technique (MLAT) of Brandt but stuck on Fast Adaptive Composite Technique (FAC) which was proposed by McCormick (1989) in his book. He first extended the multigrid idea for local multigrid schemes. An improvement followed with the usage of bordered multilevel technique which eliminates use of a fine grid out of the local fine grid. After that, he states two level FAC including a global and a local domain. McCormick also showed how to employ FAC as a preconditioner and stated the relation of the method with domain decomposition (whole domain over the local domain can be thought as an overlapping system).

Another aspect of this thesis is use of the Domain Decomposition Method. Domain Decomposition is important in two aspects (Section 6). First of all, it a good alternative to multilevel techniques to enhance the solution of nonlinear problems. Secondly, a multi-physics problem is decomposed into single physics such that each physics

can be seen a sub-domain of the global problem. After the introduction of multi-level adaptive techniques, domain decomposition techniques is also started to be investigated on real life problems. The method is reinvented by Lions after more than 100 years of Schwarz's pioneering work (Schwarz, 1869).

A comparative study on Domain Decomposition applied to elliptic PDEs is performed by Keyes and Gropp (1987). Later Börgers and Widlund (1989) use domain decomposition method to solve an internal combustion problem. Strikwerda and Scarbnick (1993) used domain decomposition techniques to analyze incompressible flow problems. They solved an external Stokes' Equation problem around a circle. The geometry is modeled in such a way that two overlapping subdomains are used. One is a rectangular domain that excludes the circle and another is a polar domain where the effect of the far field conditions are carried into the domain with data transfer from the rectangular geometry - which is an example of Alternating Schwarz Algorithm. In this paper, interpolation techniques are presented to couple both domains.

In 1996, Smith, Bjorstad and Gropp published their book on domain decomposition with the application of parallel multi-level methods (Smith *et al.*, 1996). They generally discussed conforming domains (except Schwarz's problem which is nonconforming). Additive and multiplicative multi-level Schwarz methods are compared with full multigrid. Theoretical background of Domain Decomposition is explained by Quarteroni and Valli (1999). In addition to these milestones, many papers regarding to the application of multi-level are published. Commercial analysis packages do provide multigrid as a solution method now. Mesh partitioning methods also take advantage of multi-level methods (Karypis and Kumar, 1998).

Another important technique to deal with nonlinear problems is the Newton-Krylov (NK) Method (Section 4.2.2). In this technique, Newton's Method is used to find a solution for the nonlinear system. A linear problem based on the used of the Jacobian is solved with Krylov Subspace Methods like BiCGStab (van der Vorst, 1992) or GMRES (Saad and Schultz, 1986). An advantage of Krylov solvers is that the Jacobian is not needed; any procedure that computes the matrix-vector product is enough

to perform iterations. This property leads to the so called Matrix-free methodology at which Jacobian vector products are approximated with directional derivatives without ever forming the matrix. This method is first incorporated by Brown and Hindmarsh (1987) to solve stiff ODE problems. Later, Brown and Saad (1990) extended the idea to solve PDE. In this work, damping strategies based on the work of Dennis and Schnabel (1983) are introduced to improve the convergence of the Newton's Method. Inexact Newton (IN) Method of Dembo *et al.* (1982) is also used in this work. Theory of global convergence of Inexact Newton Methods is presented by Eisenstat and Walker (1994). A study on the selection of damping parameter is performed by Eisenstat and Walker (1996). Shadid *et al.* (1997) applied IN method on Navier-Stokes Equations. Knoll and Rider (1999) used Multigrid as a preconditioner on the Newton-Krylov Method. Solution of compressible flow problems and parallel aspects of Newton-Krylov method is discussed by Gropp *et al.* (2000). NK methods have many aspects to be discussed which can be found in the review article of Knoll and Keyes (2004).

1.1. Multi-level

Multi-level idea incorporates the use of different grid levels to solve partial differential equations. Multigrid can utilize these levels to correct smooth errors in a fine grid using a coarser one. Also a coarse grid can act as an initial guess for finer grid like in FAS (Brandt, 1977). Multi-level can also be used for adaptive grids. The grids can be set to be denser where more accuracy is needed. Hence, several domains can be put on top of each other which in turn provide nicer results on overlapping parts comparing to a single grid alone (Figure 1.1). A sub domain can also act as a local error corrector in terms of multigrid (Brandt, 1980). More on multigrid is covered in section 5.

1.2. Adaptive Techniques

Adaptive idea is already contained within the context of the multi-level techniques. Adaptive grid is placed such that a region in need of highly-accurate results

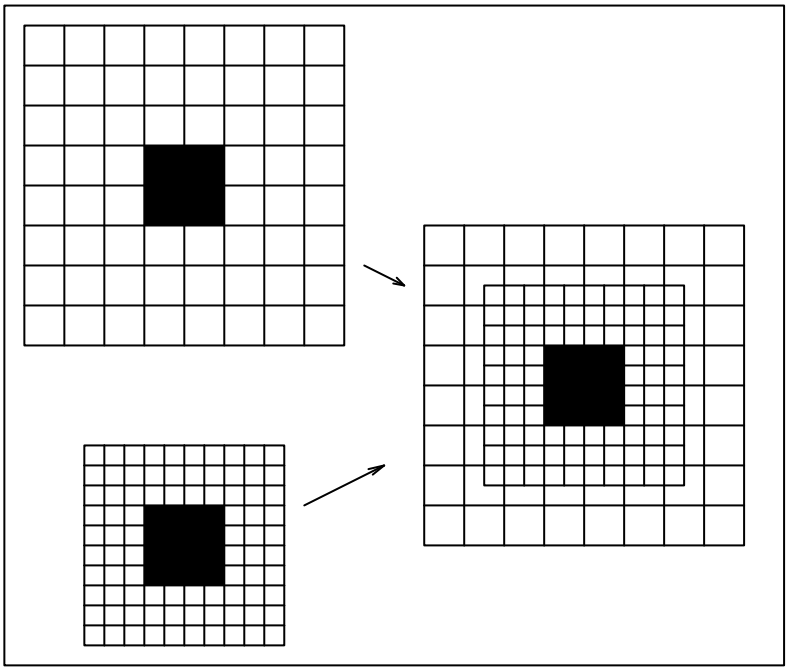


Figure 1.1. Overlapping Domains

is covered. Out of this region generally has a coarser grid but that coarse area is corrected with the results of finer grid. There are two different approaches in performing adaptive techniques (Trottenberg *et al.*, 2000). Static (or predefined) refinement is held before the solution. The analyst has already decided on critical areas where accuracy is crucial. In Dynamic (or self-adaptive) refinement, however, the solution process decides on the adaptation of the grid. If for instance, a shock wave is needed to be determined, then the process searches for sharp gradients and refines the grid around the sonic line. Static refinement is simpler comparing to dynamic counterpart, since the self-adaptive method must define a new grid system at each iteration level. If the problem is time dependent, then both ideas can be combined; static refinement in each time level and dynamic adaptation between time levels.

1.3. Multi-physics

In the analysis of coupled-physics problems, there are two directions. First idea is to use Arbitrary Lagrangian Eulerian Methods (Donea, 1983). In this approach, some portion of the fluid domain is solved with Lagrangian frame rather than Eulerian. For deformed meshes, the velocities of the grid cells are also considered which leads to the

geometric conservation. Another idea is to employ Space-Time Finite element Methods (Hughes *et al.*, 1983). Tezduyar and Behr (1992) studied the idea for deforming domains. In this technique, boundaries are in motion relative to the other nodes so their final location is decided at the end of each time step. A monolithic approach based on Space-Time Finite elements is demonstrated by Hübner *et al.* (2004) for incompressible flow problems.

Piperno *et al.* (1995) analyzed aeroelastic problems with partitioned procedures which is essentially a staggered solution method to simulate both the solid and fluid parts. Transfer of deformation and load is discussed by Farhat *et al.* (1998). Later, Farhat and Lesoinne (2000) demonstrated parallel application of staggered solution techniques. Felippa *et al.* (2001) generalized partitioned idea and applied to aeroelastic problems. A study on discretization of multi-physics problems is presented by Bailey *et al.* (1999). In this study, Finite Volume method is employed both for fluid and solid mechanics. Guruswamy (2002) reviewed interface procedures for coupled-physics problems. Multilevel Newton-Krylov analysis of Multi-physics is performed by Yotov (2001). In this work, multiphase flow is simulated for porous media. Wall *et al.* (2008) applied Alternating Schwarz Method to solve an FSI problem, flow over a perfectly elastic beam. One of the domains work with Eulerian coordinate system whereas the deforming subdomain is analyzed with Lagrangian point of view. Inner domain is conforming with the solid domain as a result the displacements can be transferred to the fluid geometry with ease.

Considering these facts, the object of this study is to develop a computational framework to study coupled-physics problems using adaptive multi-level techniques. Instead of focusing on a specific multi-physics problem, different kinds of physical phenomena will be investigated with a general point of view. In order to use different numerical techniques with one discrete model definition, an idea based on solution set will be introduced. As a result, adaptation will be an option to improve the solution. Furthermore, unification of different methodologies will be established. The proposed structure will be verified with several benchmark problems and will be applied on two multi-phyiscs problems; an FSI problem and a multiphase flow problem.

2. MATHEMATICAL MODELING

In this study, multiphysics models are to be analyzed with various numerical techniques. In order to model the physics, we need to understand the function of each physics, first. Only then, we can observe the behaviors of the physics in a multidisciplinary environment.

In a multiphysics problem, at least two-physics are present in the system. In an FSI problem, for instance, the physics are fluid and the solid parts. In a combustion problem, on the other hand, both reaction kinetics and fluid mechanics should be solved including the energy equation. Other type of connections between different physics are listed by Dehning and Wolf (2006). This thesis is focused on fluid equations plus any other theoretical frame that ends up in a coupled-physics problem. Therefore, it is useful to start with Fluid Mechanics.

2.1. Physics

2.1.1. Fluid Mechanics

Governing equations of Fluid Mechanics are called Navier-Stokes (NS) equations. NS Equations are time dependent, coupled, partial differential equations (Tannehill *et al.*, 1997). The continuity equation, also known as mass conservation, is given in Equation 2.1:

$$\frac{\partial \rho}{\partial t} + \boldsymbol{\nabla} \cdot \rho \mathbf{V} = 0 \tag{2.1}$$

Other equations are the momentum equations given in Equation 2.2

$$\rho \left(\frac{\partial \mathbf{V}}{\partial t} + \mathbf{V} \cdot \nabla \mathbf{V} \right) = -\boldsymbol{\nabla} p + \boldsymbol{\nabla} \cdot \tau_{ij} + \rho \mathbf{g} \tag{2.2}$$

In Equation 2.2 definition of the viscous stresses are related to the fluid type. For linear Newtonian fluids, the stresses are defined as in Equation 2.3:

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) + \delta_{ij} \lambda \boldsymbol{\nabla} \cdot \mathbf{V} \tag{2.3}$$

To define the bulk viscocity, λ, Stokes' assumption can be used as in Equation 2.4

$$\lambda = -\frac{2}{3}\mu \tag{2.4}$$

If the fluid is Non-Newtonian several stress models can be used. It is generally assumed that the viscosity is a function of the rate of shear stress (Bird *et al.*, 1987). Hence, any model that relates the shear rate with viscosity can be used to model a Non-Newtonian fluid. Generalized Newton Model, Maxwell Model, Oldroyd B model are some examples.

When Navier-Stokes Equations are in concern, energy equation should also be included as in Equation 2.5.

$$\rho c_p \left(\frac{\partial T}{\partial t} + \mathbf{V} \cdot \nabla T \right) = \boldsymbol{\nabla} \cdot k \boldsymbol{\nabla} T + \Phi \tag{2.5}$$

For three dimensional flow problems, there are mainly five unknowns: p, $\mathbf{V}$ and T. In this study, incompressible flow problems are considered. When Mach number is less than 0.3, then the fluid can be considered as incompressible i.e. the density remains the same. If a natural convection problem is to be studied, then Boussinesq Approximation (Bejan, 2004) can be used to model density changes. Constant density can be eliminated from the continuity equation. Unfortunately, this simplification complicates the numerical solution of Navier-Stokes Equations since pressure is missing in the continuity equation: it acts as a constrain to be satisfied by the flow field. Various solution procedures are suggested to date. Important ideas are Marker and Cell method of Harlow and Welch (1965), Artificial Compressibility (AC) method of Chorin (1967), SIMPLE and its variants by Patankar (1980) as well as PISO by Issa (1985). For finite elements, either adaptation of these techniques or finite element spe-

cific formulations (like CBS scheme of Massarotti *et al.* (1998) or penalty formulation Drikakis and Rider (2005)) can be used, however, in this study only finite difference and finite volume discretizations are considered on fluid mechanics. For the details of the techniques that are not mentioned or not described here, please refer to excellent books on Computational Fluid Dynamics of Tannehill *et al.* (1997), Ferziger and Peric (2002) and Versteeg and Malalasekera (2007).

Among the suggested solution algorithms, AC of Chorin is used as the primitive-variables based solver, and yet other methods can also implemented with proper definition of the physics of the solver. Selection of Artificial Compressibility is based on two criteria. It is easy to implement and its numerical character is suitable with the Newton-Krylov solver as well as Nonlinear Multigrid. Before giving the details of the method, it is important to talk about two other approaches like stream function - vorticity formulation and velocity - vorticity (VV) formulations which are also used to model incompressible fluids.

2.1.1.1. Stream Function Vorticity Formulation. An important solution technique for two dimensional problems is the stream function - vorticity (SFV) approach. In this method, instead of solving in primitive variables, u, v, p, the problem is solved for stream function, ψ, and vorticity, ω. An obvious advantage is the reduction in the number of unknowns. If primitive variables are needed, then they can be found with post-processing and generally the need is only at some portion of the domain, for example at the surface. Then, instead of solving for whole domain, simple formulations can be used to find the necessary values (Tannehill *et al.*, 1997). Definition of the Stream Function and the Vorticity is given in Table 2.1.

Table 2.1. Velocity and vorticity definitions

cartesian	$u_x = \frac{\partial \psi}{\partial y}$	$u_y = -\frac{\partial \psi}{\partial x}$	$\Omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$
polar	$u_r = \frac{1}{r}\frac{\partial \psi}{\partial \theta}$	$u_\theta = -\frac{\partial \psi}{\partial r}$	$\Omega = \frac{1}{r}\frac{\partial}{\partial r}\left(r u_\theta\right) - \frac{1}{r}\frac{\partial u_r}{\partial \theta}$

The Equations for the formulation are given in 2.6. Reynolds number, Re, is one the most important physical parameters in fluid mechanics. A small Re means that viscous forces are dominant. As Re increases, convective terms start to dominate and the equations become highly nonlinear.

$$\frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} + \omega = 0 \tag{2.6a}$$

$$\frac{\partial\psi}{\partial t} + \frac{\partial\psi}{\partial y}\frac{\partial\omega}{\partial x} - \frac{\partial\psi}{\partial x}\frac{\partial\omega}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2}\right) = 0 \tag{2.6b}$$

Boundary conditions are very simple for stream function but needs special attention for vorticity. Starting with stream function, it can be said that for closed domains like lid driven cavity, it can set to be 0. If there are more than one face, then each face is connected to a constant that should be calculated with proper formulations. Similar case also occurs if external flow around two distinct object is studied. In that case, stream function constant can be calculated with the technique explained by Liu and Wang (2004).

Definition of the vorticity boundary conditions is a bit tricky compared to the Stream function. Thom's formula in Equation 2.7 is a basic first order boundary condition based on the values of ψ. Its derivation is given in by Tannehill *et al.* (1997). ψ_{wall-1} is the first interior node from the wall into the domain. If the wall is stationary, then $u_{wall} = 0$. High order boundary conditions can be found in Roache (1998).

$$\omega_{wall} = \frac{2\left(\psi_{wall} - \psi_{wall-1} - h \times u_{wall}\right)}{h^2} \tag{2.7}$$

When velocity field is needed, then it has to be solved using the resulting stream function field. If values at the neighborhood of the boundary is needed, then one way finite differences can be used to calculate the variables. If the pressure is to be found, then the Pressure Poisson's equation can be used to determine the field (Tannehill *et al.*, 1997). It is also possible to generate a stream function only equation by eliminating

the vorticity. Biharmonic form of this equation requires high order discretization since fourth order derivatives are present. That also means that two boundary conditions at each boundary node should be given to close the problem. In this thesis, this form is not used.

2.1.1.2. Velocity Vorticity Formulation. Another useful idea to solve incompressible flow problems is the use of velocity - vorticity formulation. In this method, only velocity and vorticity components are solved and pressure does not exist as a variable. In 2D, only unknowns are u, v, and ω i.e. ω_z (more unknowns compared to $\psi - \omega$ formulation but same number of unknowns of the original system). Although, this formulation is useful in 2D, in 3D there are total 6 unknowns: $u, v, w, \omega_x, \omega_y, \omega_z$ and this means there are 2 extra unknowns compared to u, v, w, p. However, the problem still possess nice features since the problems[1] because of the pressure are avoided and the velocity components are solved directly. The Equations are given in 2.8 (Fletcher, 1991). If the problem is 2D, Equation 2.8c and ω_x, ω_y are omitted. Also, 2.8d gives only one equation. In 3D, however, 2.8d gives a set of three equations.

$$\nabla^2 u = \frac{\partial \omega_y}{\partial z} - \frac{\partial \omega_z}{\partial y} \tag{2.8a}$$

$$\nabla^2 v = \frac{\partial \omega_z}{\partial x} - \frac{\partial \omega_x}{\partial z} \tag{2.8b}$$

$$\nabla^2 w = \frac{\partial \omega_x}{\partial y} - \frac{\partial \omega_y}{\partial x} \tag{2.8c}$$

$$\nabla \times \mathbf{u} = \omega \tag{2.8d}$$

2.1.1.3. Artificial Compressibility Formulation. Use of Artificial Compressibility (AC) is first suggested by Chorin (1967). In AC method, steady NS Equation is turned into a pseudo-time problem by adding $\frac{\partial}{\partial \tau}$ terms on each equation. Continuity is treated specially, an extra pressure derivative is added into Equation 2.9a. β is called the compressibility parameter and $\sqrt{\beta}$ is analogous to speed of sound (Tannehill *et al.*,

[1] Checkerboard instability, definition of boundary conditions and compatibility (Ferziger and Peric, 2002)

1997). Nonconservative form of 2D AC problem is given in Equation set 2.9.

$$\frac{1}{\beta}\frac{\partial p}{\partial \tau} + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0 \tag{2.9a}$$

$$\frac{\partial u}{\partial \tau} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{2.9b}$$

$$\frac{\partial v}{\partial \tau} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{2.9c}$$

Boundary conditions for u and v are no slip boundary conditions. If collocated finite difference discretization is used, then an equation for thre pressure can be used as in 2.10 so that the system is closed. Here, n is the surface normal and $\tilde{u}_n$ is the velocity component normal to the wall. For staggered grid or cell centered finite volume methods, homogenous neumann boundary condition can be used for pressure.

$$\frac{\partial p}{\partial n} = \frac{1}{Re}\frac{\partial^2 \tilde{u}_n}{\partial n^2} \tag{2.10}$$

Main problem in AC is the selection of the β. Although there are some guidelines to select a proper value (Tannehill *et al.*, 1997), (Drikakis and Rider, 2005), most of the contributors agree that an optimized parameter can be found only by numerical experiments. Unfortunately, this selection is heavily dependent on the solution algorithms as well as the time integrator i.e. explicit vs. implicit. In this study, implicit formulation is used which is suitable for Newton-Krylov and that's why selection of $\Delta\tau$ is not an important issue since it leads to steady state solution only with no time accuracy in intermediate steps. However, as a rule of thumb $\Delta\tau$ should be selected to be small such that the CFL condition is satisfied. Later, its value can be increased as the behavior of the flow field becomes apparent. While playing with $\Delta\tau$, the product $\beta d\tau$ should be kept constant (Tannehill *et al.*, 1997). As a matter of fact, this product should be large enough so that the artificial wave can travel. In this thesis, an experimentation for β is carried out, as well. The results are presented in section 9.1.2.3, however, it can be said beforehand that Re and β are inversely proportional and for constant Re number and $\Delta\tau$, total number of time steps to reach steady state is grid size independent. The formulation can also be made time accurate with dual time stepping. In this idea, the equations are allowed to march in both pseudo and real time spaces.

Staggered grid is first used to solve simple problems and most of the techniques introduced to solve incompressible flow problems are first tested on staggered grids and these grids can also be used in the solver as explained in section 8.6.2. However, use of staggered grids requires a different kind of a data structure and also for complex domains, definition of staggered grids is a tedious. In three dimensional problems with unstructured grids, another approach should be followed: collocated arrangement. In collocated arrangement (section 8.6.1), all variables are stored at the same location, on the cell center, for instance. The problem with collocated arrangement is the checker-board pressure field which is not observed in staggered grids. Although SIMPLE is not used in this study, techniques given for collocated SIMPLE can also be applied on AC. Main idea is to connect the velocities with the pressure variables. For basic collocated discretization, Rhie-Chow interpolation technique can be used to eliminate the instabilities in the pressure field (Versteeg and Malalasekera, 2007).

One of the first example of dual time stepping is performed by Soh (1987). He investigated internal incompressible flow problems. In his work, he tried to formulate an equation to select β. In his sample calculation, it is observed that an optimized β works well and converges in less pseudotime steps compared to any other choices.

Another study on transient problems is carried out by Rogers *et al.* (1991). They analyzed both steady and unsteady problem on 3D curvilinear coordinates systems. Real time formulation is applied with second order backward Euler method and pseudo time is formulated with first order implicit integration. For steady state problems, they suggested a $\beta \in [10^0, 10^1]$ and for transient problems $\beta \in [10^2, 10^3]$

Ramshaw and Mousseau (1990) offered a modification on AC formulation with the introduction of a bulk viscosity, b. Its use is incorporated with the addition of the mass conservation equation into the momentum equations. Mathematically, this new problem is equivalent since $\nabla \cdot u$ is zero. From the numerical point of view, however, it is shown that the convergence can be accelerated. In this thesis, such a improvement is not observed - use of implicit formulation rather then explicit integration might a reason for this failure.

2.1.1.4. Non-isothermal Flows. In non-isothermal flows, energy equation is also solved in addition to the momentum equations. For incompressible flows, the temperature field is uncoupled from the velocity field if the problem is a forced convection problem. In that case, temperature does not need to be solved at the same time. Once the velocity field is found, the energy equation can be solved easily. On the other hand, for free convection problems equations should be solved simultaneously since momentum is coupled with energy filed through the buoyancy term. In natural convection problems, this coupling is accomplished with the introduction of the Boussinesq approximation where the density allowed to be change only for body force. In this thesis, non-isothermal problems are also investigated with Artificial Compressibility. Application of forced convection problems do not bring any complications.

Mandan *et al.* (2000) investigated use of AC for laminar and turbulent flow problems. For β values, they used the suggestion of Turkel. In the sample runs, it is shown that AC can be used to model Natural convection problems. As the authors knowledge, there is no other work on Natural convection problems with AC. Additionally, there is not enough evidence that the selection of Turkel is best for natural convection problems. A similar formulation of Soh (1987) can be extended to free convection problems.

2.1.1.5. Multiphase Flows and Volume-of-fluid Methodology. Volume-of-fluid, Vof, is first suggested by Hirt and Nichols (1981) for solving two fluid problems. In this formulation a fraction, $\tilde{F}$, is defined to identify the fluids. $\tilde{F} = 0$ designates the first fluid and $\tilde{F} = 1$ shows the other. In this idea, an additional advection equation is solved for $\tilde{F}$ and hypothetical $\tilde{F} = 0.5$ line is assumed to be the interface. Because of the nature of the equation, the interface cannot be tracked directly whereas in Marker and Cell method of Harlow and Welch (1965) "mark" the location of the interface. The fuzzy region around the interface can either be narrowed with a fine mesh or with a high order method. Former idea can easily be adapted with application of domain decomposition. The equation in two dimensions is given below in Equation

9.24. Pseudotime derivative is also included to combine VOF with AC.

$$\frac{\partial \tilde{F}}{\partial t} + \frac{\partial \tilde{F}}{\partial \tau} + u\frac{\partial \tilde{F}}{\partial x} + v\frac{\partial \tilde{F}}{\partial y} = 0 \tag{2.11}$$

2.1.2. Solid Mechanics

When dealing with Fluid-Structure Interaction problems, one has to solve the solid mechanics problems, as well. For 2D linear problems, plane stress and plane strain problems can be analyzed. Normally, Elasticity equations can be solved with finite differences, however, since the problems are generally involved with deformations, solutions domain might not be suitable for FD (w/o mapping) hence use of Finite elements is an excellent choice for solid mechanics. Analysis of elasticty problems with finite elements is a mature research area. Timoshenko and Goodier (1970) is an excellent reference for elasticity. There are also numerous book on finite elements of solids like Cook (1995), Bathe (1996) and Pepper and Heinrich (2006).

Equation 2.12 states the equilibrium and Equation 2.13 gives the relation between the strains and the displacements Timoshenko and Goodier (1970). Depending on the assumption of plane stress or plane strain different constitutive equations can be given (Ugural and Fenster, 1994). In this thesis, plane strain is used to model the deflection of beams (Equation 9.18)

$$\begin{aligned} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial x} &= -f_x \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial x} &= -f_y \end{aligned} \tag{2.12}$$

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{2.13}$$

2.1.3. Elastohydrodynamics

Analysis of Lubrication process is examined with Elastohydrodynamics. Tribology mainly involved with the solution of Navier-Stokes Equation in a thin gap. For this kind of a geometry, Reynolds Equation can be used to model the flow field.

2.1.3.1. Hydrodynamics Lubrication. Hydrodynamic lubrication problems are governed with Reynolds Equation. Reynolds modified Navier-Stokes Equations with the assumption that the flow is slow and viscous, and the gap that the lubricant passes through is too narrow (Reynolds, 1886). Reynolds Equation is the basic equation of Hydrodynamic Lubrication used frequently in the analysis of Journal Bearings. Dimensionless form of the model is given in Equation 2.14

$$\frac{\partial}{\partial\theta}\left(\xi\frac{\partial P}{\partial\theta}\right)+\frac{1}{k^2}\frac{\partial}{\partial Y}\left(\xi\frac{\partial P}{\partial Y}\right)-\frac{\partial H}{\partial\theta}=0 \tag{2.14}$$

This problem resembles the Bratu problem, if the variation of film thickness, H, is known in advance. A typical assumption is that of Equation 2.15 in which H varies throughout the domain (in fact in θ-direction only). Also note that $\xi = H^3$. The parameter ϵ is a dimensionless measure of the eccentricity.

$$H(\theta) = 1 + \epsilon cos(\theta) \tag{2.15}$$

If H is known, then the problem is no longer nonlinear however it still has some important properties. First of all, the k term, ratio of the thickness to the radius of the bearing, results in anisotropy if its value is different than unity. Secondly, the cavity condition as explained in section 9.4.3 complicates the problem.

2.2. Discretizations

2.2.1. Finite Differences

Finite difference is a relatively simple discretization procedure based on truncated Taylor Series Expansion. On basic coordinate systems, like cartesian or polar, the implementation is straight forward, however, for special orthogonal coordinate systems, respective metrics should be applied carefully. For complex domains, a mapping i.e. grid generation (algebraic or pde based) is necessary hence the computations are more involved with the calculation of the Jacobian of the transformation. Detailed analysis is given in many text books on computational fluid mechanics (Anderson, 1995),(Hoffmann and Chiang, 2000), (Date, 2005).

Strikwerda (1989) gave a comprehensive study on finite differences and their application on PDE's. Until the formulation of finite volume methods, finite differences were used in the analysis of fluid dynamics. On the other, finite volume dominates numerical flow studies because of its ability to be applied on complex domains. Still, finite difference is used especially on simple problems. In this thesis, basic problems are modeled with finite differences to test the performance of the solver. Additionally, application of Newton-Krylov and Multigrid is easier with finite differences.

2.2.2. Finite Volumes

Finite volume method is an important discretization method in numerical analysis. It is based on the integral form of equations. In terms of fluid mechanics, finite volume possess desirable properties such that the continuity is satisfied within the fluid. For a control volume, the integral forms are still exact, the approximation comes into the picture when computing the features at the cell faces, both for velocities and scalar variables. Here, Finite Volume also relies on Taylor series expansion.

In this study, only cell centered formulations are carried out rather than vertex centered methodology. For diffusive terms, centered finite differences are used. For

convective terms, both first order upwind and QUICK(Leonard, 1979) schemes are employed.

$$v_f = \frac{3}{8}v_D + \frac{6}{8}v_U - \frac{1}{8}v_{UU} \tag{2.16}$$

In QUICK face velocity, v_f, is calculated with Equation 2.16. Selection of nodes is decided on the sign of the face velocities. The list of possible combinations is given in Appendix A. Several modifications are offered in the literature like SMART(Gaskell and Lau, 1988) or SHARP(Leonard, 1988). These techniques limit the face velocities since QUICK is not monotone, i.e. face velocity might be out of range of $[v_C, v_E]$. In this thesis SMART is used since its use with preconditioned Newton-Krylov solver is already proved to be successful (Knoll, 1998).

2.2.3. Finite Elements

Finite Elements are frequently used in solving PDE's, particularly in solid mechanics as well as in the analysis of fluid flow[2] . From a Multiphysics point of view FEA has many advantages so it will be a wise idea to implement FEA to investigate the "other physics". Since we generally have to deal with deformed geometries, discretizations using Finite Differences will always be problematic especially to treat the irregular geometries. However, Finite Elements do not have any difficulty in dealing with boundaries since the elements does not have to be regular so they can be transformed to fit. Each element approximates the change of any property with its nodes. High number of nodes within an element increases the accuracy as well as the computational cost (p-version). Instead of high order elements, more element with low order (linear variation, for instance) can also be employed (h-version). However, this approach might also affect the computations since a region in the domain could be needing more elements to capture sharp changes or discontinuities. As a result combining those two (hp-version) could provide an optimized performance Akin (2005) by providing adaptation[3] . Adaptivity is already an important research area as shown in

[2]Instead of a Galerkin approach, the so called Petrov-Galerkin (PG) is used where the trial functions and weighting functions do differ (They are the same in Galerkin).

[3]There is also the so-called r-version where distribution of nodes are changed Tannehill *et al.* (1997)

Schaefer (2006) and also in the title of this thesis the keyword "Adaptive" is used.

In this study, 2D heat-transfer and elasticity problems are solved using finite elements. Linear triangulars and bilinear quadrilaterals are used in the analysis. Formulation for conduction problem is given below. Details of elasticity problem is given by Pepper and Heinrich (2006). Discrete system is given in Equation 2.17 where K is the stiffness matrix, u is the nodal unknowns and f is the forcing function. Material stiffnes matrix, D in Equation 2.18 is defined by 2.20. N in Equations 2.18, 2.19 is the nodal shape functions ($N = [N_1, N_2, N_3, N_4]$). B is given in Equation 2.21 (if a triangular element is used then the last column is ignored). Q is the source term, q is the heat flux and h is the convective heat transfer coefficient. When these equations are observed, it can be said that the discretization can be performed systematically by evaluating each part of the integrals.

$$\mathbf{Ku} = \mathbf{f} \tag{2.17}$$

$$K = \int_{\Omega} B^T D B d\Omega + \int_{\Gamma} h N^T N d\Gamma \tag{2.18}$$

$$f = \int_{\Omega} Q N^T d\Omega - \int_{\Gamma} q N^T d\Gamma + \int_{\Gamma} h T_{\infty} N^T d\Gamma \tag{2.19}$$

$$D = \begin{bmatrix} k_{xx} & 0 \\ 0 & k_{yy} \end{bmatrix} \tag{2.20}$$

$$B = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \end{bmatrix} \tag{2.21}$$

In elasticity problem, one has to change B, D, N and also respective boundary formulation. For completeness it is worth to show D and B matrices for a plane-stress problem

(Smith and Griffiths, 2004).

$$D = \frac{E}{1-\upsilon^2}\begin{bmatrix} 1 & \upsilon & 0 \\ \upsilon & 1 & 0 \\ 0 & 0 & \frac{1-\upsilon}{2} \end{bmatrix} \tag{2.22}$$

$$B = \begin{bmatrix} \frac{\partial N_1}{\partial x} & & \frac{\partial N_2}{\partial x} & & \frac{\partial N_3}{\partial x} & & \frac{\partial N_4}{\partial x} & \\ & \frac{\partial N_1}{\partial y} & & \frac{\partial N_2}{\partial y} & & \frac{\partial N_3}{\partial y} & & \frac{\partial N_1}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} \tag{2.23}$$

2.3. Challenges in Multiphysics

The challenge in scientific computing is always the same, a numerical solver should be fast and robust, and perform reasonably well in large scale. This object is, however, not easy to fulfill, especially in multiphysics. Besides the complications arising in the solution of nonlinear PDE's, main concern in coupled-physics problems is that different physics have different numerical characteristics. Each of them should be handled carefully i.e. they should be treated with proper solution methodologies that is suitable for themselves. Unfortunately, this treatment, alone is not enough to close the issue, the way of the coupling between the physics is another level complexity for the analysis. Additionally, implementation of various solution methodologies are not easy to combine in single solver since they will eventually require different algorithms which will lead to different data structures.

When the coupling between the physics are considered, two types of approaches are utilized; weak and strong couplings. In the industry, generally weak-couplings are used since there are specialized packages for specific physics (like Fluent[4] for fluid mechanics and Abaqus for solid mechanics) and they provide cutting edge solution

[4]Merger of Fluent with Ansys is not just to improve the capabilities of the CFD solver. Also, multiphysics problems, like Fluid-Structure Interactions are simplified and the coupling is handled internally

algorithms for their area of expertise. In this way, latest developments in the numerical techniques for the simulation of the particular physics can be implemented on the multiphysics problems. Major concern in weak coupling, on the other hand, is that an interface is required to transfer data across programs. Those packages may provide their own interfaces for simple couplings or an analyst can define a user-specified script to interact with other softwares but if we require something more pronounced, the genuine solution is to use an additional software that manages different programs and synchronizes the data transfer, like Mpcci (2010) (Mesh-based parallel Code Coupling Interface). Another problem is that the stability is limited and convergence requires a reasonable amount of computational time.

Monolithic algorithms, i.e stronger couplings are more robust yet difficult to model because tedious efforts are required to deal with complicated equations and generalization of them is not straight forward - they are mostly applicable for a tailored multiphysics problem. Which coupling is to be used is really user dependent, the researcher could use any kind of methodology that is suitable for the model. However, a solver designed to analyze multiphysics problems should able to work regardless of the coupling type.

Another difficulty in multiphysics simulation is the discretization of the governing equations. Take FSI problems as an example. Although there are exceptions, fluids are mostly modeled with finite volumes and solids with finite elements. Solution algorithms do have differences on both systems therefore conformity in between becomes important. It is also possible to use a single type of method to transform continuous problems into discrete ones but as stated before, contemporary methodologies are available only for frequently used approaches there a bunch of studies are for finite volumes in fluids and finite elements in solids. Considering this fact, a coupled physics solvers better be perform in more than one discretization options. In fact, it should focus on the discrete system rather than how we get them. In other words, we should concentrate on nonlinear systems and their solutions. As a result, universal solvers like multilevel should be preferred instead optimized numerical solvers.

One characteristics of multiphysics is that the problems are usually transient. The issue is that physics might have varying time scales so marching in time would not be performed simultaneously for all physics, the interactions will be considered only at the synchronization points. That means, a transient solver should work on different time lines.

To accommodate multiphysics problems, a solver called DEMONA - Decomposition Enhanced Mechanics Optimized Numerical Analysis is created. Before going into the details of DEMONA, we should summarize some of the solvers that are available to use and free to use i.e. opensource.

2.3.1. Petsc

Petsc - Portable, Extensible Toolkit for Scientific Computation (Balay *et al.*, 2010) is one of the flagships of the scientific community. Main feature of Petsc is that it provides scalable parallel solvers to model scientific problems. Petsc is programmed in C but Fortran codes are also available. Petsc includes linear and nonlinear solvers both for steady and transient problems. Computationally, its foundations are based on MPI and BLAS and numerically on preconditioned Newton-Krylov methods. The advantage of Petsc is that any vector or matrix defined by the user is inherently parallel. Design of a parallel solver is simple with the use of Petsc-specific call routines. Although tutorials are available to be followed by novice users, development of a complex model or implementation of a new solution technique requires experience in Petsc. Some of the features of DEMONA are already available in Petsc and they are introduced years ago however new ideas presented in this thesis -like use of sets to change the solution algorithm or direct coupling of different physics could only be realized with a new solver. In DEMONA, index sets (not with the same meaning of index sets used in petsc) is specifically designed to eliminate some of the unknowns so that a particular solution algorithm can be implemented without modifying the model or using special flags to distinguish between different methodologies.

2.3.2. Sundials

Sundials - SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers (Hindmarsh *et al.*, 2005) is a software package that contains different solver for various numerical problems. Kinsol is the solver in Sundials that is designed to solve nonlinear problems. In employs Inexact Newton-Krylov method to deal with large scale problems. It is based on the solver released by Brown and Saad (1990). Like Petsc, Sundials is also programmed in C, however, interfaces for Fortran codes are included. Sundials has another solver named Cvode which is specifically designed for ordinary differential equations. This solver is suitable to analyse time dependent problems.

2.3.3. Trilinos

Trilinos is a project of Sandia National Laboratories (Heroux *et al.*, 2005) which aims to collect different softwares to analyze large scale scientific problems. Similar to other two projects, solvers are included to the project. What is different than other two packages is that it also includes tools to perform preprocessing like grid generation. Additionally, discretization can be perfomed with an additional tool provided by Trilinos. NOX is the solver that deal with Newton-Krylov Method. It is programmed in C++, thus NOX is object oriented. Trilinos also offers preconditioners to be used in the solution of linear iterative solvers.

2.3.4. Other Packages

Other noteworthy efforts in scientific computing are hypre (Lawrence Livermore National Laboratory, 2010), Sparskit of Saad (2010) and Nitsol of Pernice and Walker (1998). Hypre is a suit for high performance preconditioners. Sparkit is a collection of subroutines to solve sparse linear problems. Nitsol is a solver for nonlinear systems based on Inexact Newton-Krylov Methods.

3. DEMONA

In order to deal with the challenges of multi-physics problems, a solver is developed. Decomposition Enhanced Mechanics Optimized Numerical Analysis, DEMONA in short, is an attempt to collect several numerical methods under the same framework such that they can be applied on various coupled-physics problems. Consequently, DEMONA is designed to be a computational test bed for hybrid solvers. Essentially, the analyst is not limited with a narrow set of solution algorithms, several methods are already implemented but one is free to add new ideas. DEMONA requires similar inputs compared to commercial packages like use of a mesh in preprocessing phase. Additionally, the discrete model has to be given manually by the user. While doing that, least amount of work is required to be able to use varying methodologies with the same discretization. On the other hand, implementation of a discrete model will be different than usual approaches so details are to come later in this section.

Keyword optimized, refers to selection of suitable solution techniques for a specific kind of coupled-physics problem rather being an optimized solver for every kind of problems. As DEMONA is an effort to use a number of different techniques, it cannot contain every kind of solution approaches and also it is not here to analyze all types of model problems. Before introducing the features of the solver, some of the assumptions made during the design phase should be brought in. DEMONA is designed only for structured grids but general quadrilaterals are also allowed. Unstructured grids are implemented in the solution phase of the solids via finite elements but some of the DEMONA specific routines are not applicable. In reality, basic solution methods like Newton-Krylov can be used for unstructured grids with minor modification on the solver but the intention is to let the models be tested with all of the numerical techniques and this is possible only for structured meshes.

Real time and pseudo time computations are possible in DEMONA. However, only implicit and explicit Euler Method is implemented as time integrator. Runge-Kutta like multistage algorithms are not included. Numerical solvers are selected as

Newton-Krylov (NK) and Multigrid (MG) in conjunction with domain decomposition. Only iterative linear solvers are utilized. Direct solution methods, like Gaussian Elimination is not a part of the solver but backward and forward substitution algorithms are present since they are frequently used in preconditioning. Parallel processing is performed only for shared memory computations i.e. by the use of OpenMP. In terms of multi-physics, test problems presented for coupled phenomena features weak couplings rather than strong connections. However, numerical couplings could be set stronger and besides monolithic approaches are also possible in DEMONA with proper definitions of the models.

One of the main difficulties in designing a solver that is applicable for NK and MG at the same time is that NK methods work well with vectors but MG methods need scalar forms of the models. First idea that comes to one's mind is to combine them at the same solver might be having two separate discrete definitions. However, this requires a tedious amount of work when a discrete model is in development. Even a minor change in the discretization should be reflected to both systems which is an error prone operation. Another idea might be use of just a vector to characterize the model and extract scalars from the vector as it is evaluated with the current iterate. Taking values one-by-one is alone inefficient and besides in MG the so-called "smoothers" (explained in section 5) sweep the entire domain and consecutive evaluation of the vectorial system will be time consuming. To overcome these difficulties, a new technique is proposed. DEMONA features a number of new solution methods that are based on the use of "sets" which is realized with the use of an index matrix called as `IM`. At the beginning, use of sets are essentially planned to reduce the size of the nonlinear systems through elimination some of the unknowns temporarily during intermediate iterations but the method came out to be a more general idea. A number of computational techniques could be embraced within the set scheme. The discussion on DEMONA will start with an introduction to "sets" followed by the use of `IM`.

3.1. Sets

One important aspect of DEMONA is the use of sets. The "Sets" defines a particular portion of the model that is to be solved. Normally, that does not make any sense since we are trying to solve all of the unknowns; however, use of sets is a generalization of the global problem. Depending on the solution algorithm or the flow of the solution process, a subset of the all unknowns is selected. To be formal we call this set as local set and denote it as $\mathcal{S}_L$. Similarly, the set of the all unknowns to be solved is called the global set, $\mathcal{S}_G$. The advantage of use of a subset is that the number of unknowns is reduced. If we compare a fine grid and a coarse grid, the condition number of a fine grid is larger for which the convergence will take more iterations. With analogy, we can think that the reduced set of unknown might have a better condition number. Naturally we should ask, whether this assumption is valid and also is the condition number directly related to the convergence. Starting from the latter, a small condition generally means better convergence i.e. if the spectral radius (calculated over the eigenvalues) is small, then convergence is to be archived with less iteration. On the other hand, there are cases, especially in nonlinear problems, where small eigenvalues does not result in better convergence. Even in these cases, however, use of preconditioning (section 7.1) cures the model so linear solution is performed with success. The question, whether a reduced set has better convergence properties is not answered here rather it is discussed frequently in the thesis. Key issue in testing the performance of the sets is that, proposed model with those eliminated unknowns may perform well in solving that particular set but the overall performance might be deteriorated. To be fair in comparing the performance of the sets, a distinction made for the type of the sets and the idea is divided into two types as seen in Figure 3.1: run-time sets and algorithmic sets. In short, run-time sets are decided while the iterations

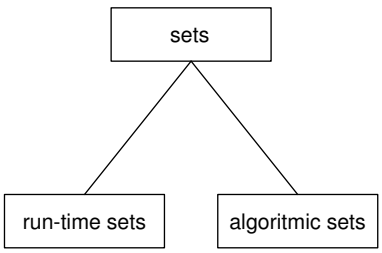


Figure 3.1. Type of the sets

on the nonlinear problem continue. From one iteration to another the set might be changed. In algorithmic-sets, on the other hand, the sets are defined prior solving the unknowns. Actually, there is an analogy similar to dynamic-static adaptation: run time sets are dynamically created while algorithmic sets are static and does not change until the solution of that particular set is found. In terms of hybrid solution techniques, combination of those two is also possible. Details of these sets will be given as the index matrix, `IM` concept (section 3.2) of DEMONA is introduced.

$$\mathcal{S}_L \subseteq \mathcal{S}_G \tag{3.1}$$

It is useful to make some remarks on the set idea. From the definition of the sets one obvious relation can be stated as in Equation 3.1. If nothing special is performed, then local set is equivalent to the global set and this is what mostly done in scientific analysis. When the local set is a subset of the global set, then computations are performed on the local level but with a global point of view. That means, the solution of the model should stop when the results of the global set i.e. all of the unknowns are satisfactory (i.e. the norm of the residual is below a given tolerance). This part should be stated clearer to avoid confusion. We are solving a local set so we have to stop somewhere to get a local solution. Hence, a local stopping criterion is needed. Still, we are done only if the global set converges. Central difficulty is that this concept is not easy to generalize and the application is dependent on the selected solution methodology. Then again, another definition for global set is required since main concern is not to solve just one domain for one physics. Multiple physics with a number of domains might be present in a multiphysics model. Here we make a distinction between the model and the global set. Model is everything with its physics, domains, unknowns vector and geometry. A global set is a part of the model that is to be examined. To get rid of the contradiction which reads as "global set said to be the set of all unknowns before but now it is not actually - it is also a subset of the model" further clarification is necessary. Global sets are defined with the modeling algorithm and local sets with the numerical algorithm. In order to grasp the idea behind this statement, we give an example. Consider Figure 3.2. This is an illustration of an FSI problem; a vertical slab is placed on free stream and allowed to be deflected. The pressure field on the

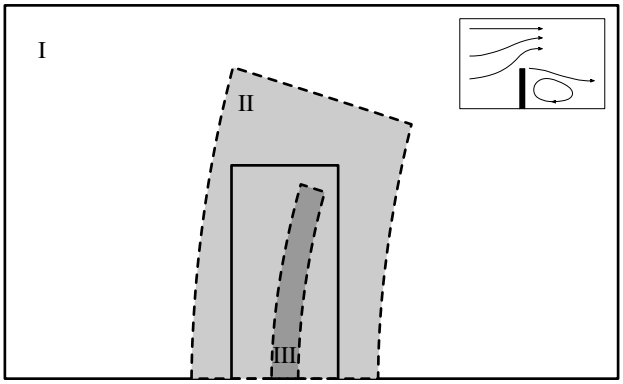


Figure 3.2. Multiphysics example

structure will deform the geometry, as well as the boundaries of the flow domain. When the domain of the flow field is changed, the grid should be regenerated. As in this thesis it is frequently used, overlapping domains are here to eliminate the need for re-mesh or at least it will be performed only in the interior domain (designated as II). Domain I, Ω_I, excludes the solid (designated with III) and some neighborhood of it. Domain I can have a structured mesh, whereas domain II should adapt itself to the deformation of the structure. For the setup of the solver, we will say that the model is made of two physics with three domains. First physics is the flow field and includes domains I and II. These two domains are overlapping and non-conforming (please refer to section 6 or appendix D for definitions). Second physics is the solid with domain III. Domain II and III might be conforming or not but they are non-overlapping. From numerical point of view this problem can be solved in different ways. In the first idea, physics can be treated partitioned, i.e. we start with fluid physics and consider solid and then fluid again. These visits will continue until we reach convergence. For this idea, the model includes all three domains and but the sets not. When we are solving the flow field then $\Omega_I, \Omega_{II} \in \mathcal{S}_G$ and also $\Omega_{III} \notin \mathcal{S}_G$. When we move on the solid domain, then the relation switches: $\Omega_{III} \in \mathcal{S}_G$ and $\Omega_I, \Omega_{II} \notin \mathcal{S}_G$. Another solution technique could be involved with the solution of each three domains one by one. In this case, one domain will be on the global set and the other two not. As a summary, a global set defines a subset of the model that is solved at a time. Now, we can proceed with a more formal definition of the local set. Let's say Ω_I is the domain of interest with N number of unknowns and dof number of local freedom. If fluid equations are to be solved coupled using Newton's Method, then $\mathcal{S}_L$ includes all of the unknowns and

$\mathcal{S}_L \equiv \mathcal{S}_G$. If, however, Picard's method is to be used, where each degree of freedom is solved one by one in a segregated manner, then $\mathcal{S}_L \subset \mathcal{S}_G$ such that $size\,(\mathcal{S}_L) = \frac{N}{dof}$. Another example could be use of Gauss-Seidel Method. If point-wise GS is employed then $\mathcal{S}_L$ includes only 1 unknown and a block version will work on dof number of unknowns, locally.

Certainly, with the selection of the solution algorithm, different sets can be generated. Details of the sets idea will be revealed as new methodologies are introduced. One thing is clear; use of sets requires a new approach to store the unknowns and a work-around to evaluate systems of equations. Creation of sets requires use of the index matrices (`IM`s) which is explained next. With the help of `IM`, indices of the set unknowns could be stored in a tree like structure. This part of the discussion is postponed after the explanation of the indexing.

3.2. Index Matrix

Numbering of unknowns is important in numerical analysis. Before the solution starts, each unknown should be assigned to an index which designates the location of the current unknown on the solution vector. With proper indexing (like nested dissection (Saad, 2003)), the convergence of a problem could be improved but in turn selecting the appropriate variables to be used in the discretization will be not straightforward and an auxiliary array might be necessary to associate the unknowns with the indices. If the domain is a full structured grid of rectangles, then indexing is a simple task, not only in Cartesian, also in orthogonal coordinate systems and even in mapped grids. Since the structure is straightforward, the indices of the neighboring nodes or cell can be determined directly. In Figure 3.3 for example, the index is calculated by $(j-1)*n_i+i$, n_i being the number of grid points in the i direction ($n_i = 5$ in this figure). For a multi-dof problem with dof number of degree-of-freedom per computational node, the index of the first dof can be calculated with $((j-1)*n_i+i-1)*dof+1$. In this formulation it is assumed that the unknowns are ordered lexicographically, i.e. starting from $(i,j) = [1,1]$ sweeping first over i's and then j's. Another assumption is that all local unknowns are numbered before proceeding with the next node. In other words,

the unknowns are sorted as $\psi_1, \omega_1, \psi_2, \omega_2 .... \psi_n, \omega_n$ for the stream function - vorticity formulation, for example. Another variant is also possible like $\psi_1, \psi_2, ... \psi_n, \omega_1, \omega_2, ... \omega_n$ which needs a different indexing formulation and results in a different sparsity pattern.

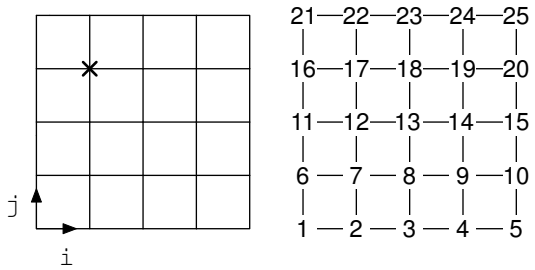


Figure 3.3. Simple rectangular grid

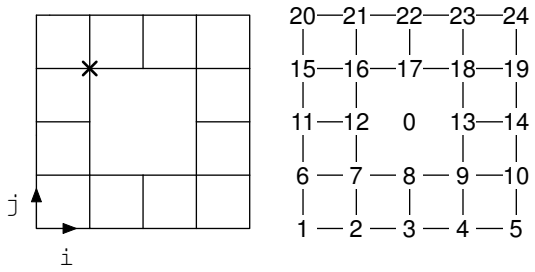


Figure 3.4. Rectangular grid with a hole

There are instances at which the domain of concern cannot be represented with a full rectangular shape, i.e. holes are present. Then relative index formulation will not work. In this case, a matrix should be used to point the indices of the unknowns which is formed beforehand. When we compare Figures 3.3 and 3.4, we observe that the point designated by an $\times$ has an index of 17 and 16, respectively. Proposed formulation fails on the second grid. Hence, an array is needed to refer to the indices of the unknowns. The numbers on the right hand side of the figures can be readily used to fill a matrix which we call as the index matrix, in short `IM`. For single dof problems, i and j are enough to call the index as $\texttt{IM}(i, j)$. In order to find the index of point $\times$, we need the value of $\texttt{IM}(2, 4)$. The difficulty is that multi-dof models require an extra evaluation to find the indices of each degree of freedom. Consequently, $\texttt{IM}(i, j) * dof + 1$ should be used to find the index of the first dof. At this stage, use of two indices[5] gives the impression that no further improvement is needed. As a matter of fact, this idea can

[5]This is for 2D. In 3D, three indices are needed.

also be used for pseudo rectangular grid like the one in Figure 3.5. Another apparent advantage of this form is that reordering algorithms can be applied with ease; only IM should be modified and access to the indices will still be performed with $\texttt{IM}(i, j)$. Unfortunately, the method with its current form is not flexible enough. For example,

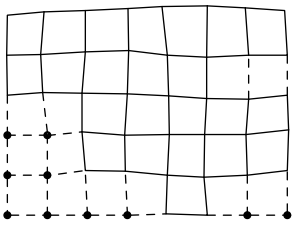


Figure 3.5. Non-uniform grid

the second variant proposed to number multi-dof problems requires more complex indexing formulations and that gets even more complicated if reordering algorithms are used. To simplify the process, a third index is introduced which symbolizes the *dof* such that $\texttt{IM}(i, j, t)$ will be enough to operate. The index of a third dof at point $\times$ can now be reached with $\texttt{IM}(2, 4, 3)$.

3.3. Evaluation of Functions

Numerical solution of Partial Differential Equations (PDEs) requires discretization of the governing equations. When the continuous model is converted into a discrete model, a system is generated which is mostly a nonlinear relation between the unknowns of the problems. A discrete system is in fact a vector. Analytically, the problem is either linear or nonlinear, however for brevity, no distinction is made for linear problems and every model problem treated as nonlinear and represented with $F(x)$, x being the solution vector. F is a vector that includes all local discretization functions, f_p at each point for each dof. This is one of the features of scientific computing; the computations are performed on the basis of vectors. The solution is a vector at the first place. Furthermore, even if matrices are needed, their values are also kept in different vectors (storage concerns) and the connections between those arrays define the matrix in an abstract level. As a result, a solver should work comfortably with vectors. Despite its limited capability in object oriented programming, FORTRAN is a very efficient programming language in numerical analysis since it is designed to be so.

Definition of and operations on vectors are straightforward in FORTRAN 90. Hence DEMONA is programmed in FORTRAN. The nonlinear function, F is located at the center of the solver. All of the computations are carried out through F. The referral of F in DEMONA is `discrete`. As the name implies, it serves as the discrete form of the model problem. Basic form of `discrete` for a simple diffusion problem like in Equation 3.2 discretized with Finite Differences as in Equation 3.3 is expressed in Table 3.6 (where the declarations of the variables are ignored). C, W, E, N and S represent center, west, east, north and south nodes, respectively. i and j are compatible with Figure 3.3 where use of `IM` is avoided. Since boundary conditions are homogenoues Dirichlet type, no discretization is necessary.

$$\Omega : \nabla^2 T + 1 = 0, \qquad \partial\Omega : T = 0 \tag{3.2}$$

$$F_i(x) = \frac{T_W + T_W + T_N + T_S - 4T_C}{h^2} + 1 \tag{3.3}$$

Main observation about the function is that it is written into a module. Use of modules has several appealing features which can be found in Chapman (1998). To summarize, the size of the function is not have to be an input into `discrete`. This avoids use of interface codes of FORTRAN (Chapman, 1998). Furthermore, a variable defined in the module (not in the `discrete`) is a global variable for each routine that works on the module. This is advantageous if several variables should be used in different routines so current value of that particular variable could be used efficiently. If we would like to evaluate the same problem with IM, we have to make the observation that f_p can also be represented as $f_{i,j,t}$. Hence, the placement of the local discretizations on F is realized with the use of formulation: $F(\texttt{IM}(i,j,t))=f_{i,j,t}$. This form is given in Figure 3.7. Main difficulty is, however, vector computations are not suitable for all of the solution methodologies. Stationary Methods for instance, requires scalar definitions of the problems. In other words, local computations are needed rather than global computations. In Gauss-Seidel method for instance, all discrete points should be visited one-by-one and recently computed variables are used as soon as they needed. Although, Jacobi Method or Kacmarz's Method (Kaczmarz, 1937) could be applied in

```
module problem
    contains
    function discrete(x)
            discrete=x
            do j=2,n-1
                do i=2,n-1
                    index=(j-1)*n+i
                    TC=x(index)
                    TW=x(index-1)
                    TE=x(index+1)
                    TN=x(index+n)
                    TS=x(index-n)
                    discrete(index)=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0
                end do
            end do
    end function discrete
end module problem
```

Figure 3.6. CODE Listing for Equation 3.3

```
...
    function discrete(x)
            discrete=x
            do j=2,n-1
                do i=2,n-1
                    TC=x(IM(i,j,1))  ! 1 to denote the first dof
                    TW=x(IM(i-1,j,1))
                    TE=x(IM(i+1,j,1))
                    TN=x(IM(i,j+1,1))
                    TS=x(IM(i,j-1,1))
                    discrete(IM(i,j,1))=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0
                end do
            end do
    end function discrete
```

Figure 3.7. CODE Listing for Equation 3.3 with `IM`

matrix-free concept as explained in Appendix B, a more general form of discrete is needed. An improved form should also be working on block versions of the smoothers, as well. Proposed new form is strongly connected to the set idea. Let's examine the structure of `discrete` shown is Figure 3.8. This form constitutes for all physics will all of their domains so separate `discrete` files for each physics are avoided. When Figure 3.8 is studied, it can be said that there are loops to sweep over each discrete points. On the other hand, definition of the discretizations for each dof is split. This split form is essential for set idea. Consider Figure 3.9 where a second dof U is introduced using the same form of Equation 3.3. This setup has two apparent drawbacks. Firstly, there is no

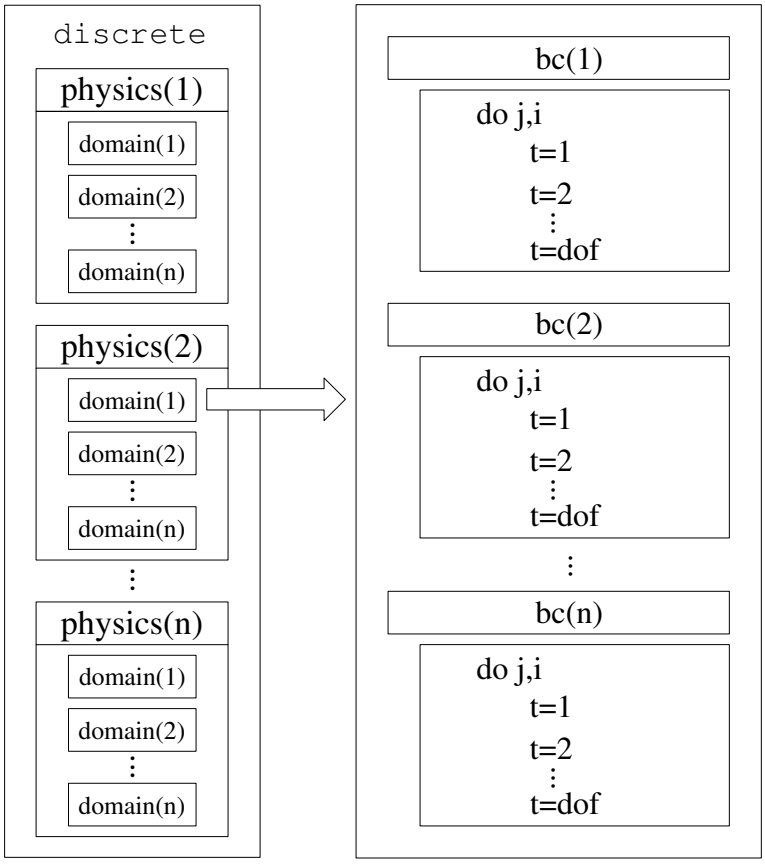


Figure 3.8. Structure of `discrete`

check for $\texttt{IM}(i,j,t) = 0$ case, if a point outside the domain is to be calculated, then the loop should cycle without performing any computation. Secondly, the value of discrete for all unknowns are evaluated here, however if a particular unknown is not a member of the current set, then the computations should be avoided. One idea to eliminate the former issue might be using a if statement such in Figure 3.10. In this code, the logical statement is based on the first unknown ($\texttt{IM}(i,j,\mathbf{1})$) and the rest of the unknowns are

```
...
    function discrete(x)
            discrete=x
            do j=2,n-1
                do i=2,n-1
                    TC=x(IM(i,j,1))  ! 1 to denote the first dof
                    TW=x(IM(i-1,j,1))
                    TE=x(IM(i+1,j,1))
                    TN=x(IM(i,j+1,1))
                    TS=x(IM(i,j-1,1))
                    UC=x(IM(i,j,2))  ! 2 to denote the second dof
                    UW=x(IM(i-1,j,2))
                    UE=x(IM(i+1,j,2))
                    UN=x(IM(i,j+1,2))
                    US=x(IM(i,j-1,2))
                    discrete(IM(i,j,1))=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0
                    discrete(IM(i,j,2))=(UW+TE+UN+US-4.d0*UC)/h**2.d0+1.d0
                end do
            end do
    end function discrete
```

Figure 3.9. CODE Listing for Equation 3.3 with `IM` for 2 dof

```
...
    function discrete(x)
            discrete=x
            do j=2,n-1
                do i=2,n-1
                    if ((i,j,1).eq.0) cycle
                    ...
                    !perform computations
                end do
            end do
    end function discrete
```

Figure 3.10. CODE Listing for Equation 3.3 with `IM` for 2 dof

not considered. This is actually a valid operation; if one of the unknowns is out of the domain, the others are also out of domain. The disadvantage of this idea is that the condition should be tested for every index in the domain. Furthermore, `discrete` is called several times by the numerical algorithms so that kind of a modification is in fact a bottleneck for the computations. This issue is an indication to change the arrangement of the function. For the latter case, nothing can be done with the current form of `discrete` since there is an ambiguity on the state of each dof - it is not obvious whether a dof is in the set or not. The index of one particular unknown can be set zero explicitly using IM, but this will take the point out of domain rather taking out of the set. Obviously, a different mechanism is needed for the routine.

3.4. First Case Study

Consider Figure 3.11. Assume that the points marked with black circles are selected somehow as the set for the computations. What we want is a new structure that will evaluate local function, f_p only at those points. Examining the figure, it can be figured out that all rows are not fully covered, even $j = 4$ has no points in the set. In this model, size($\mathcal{S}_L$) = 11 and size($\mathcal{S}_G$) = 25. Therefore, `discrete` should return a vector with 11 elements only.

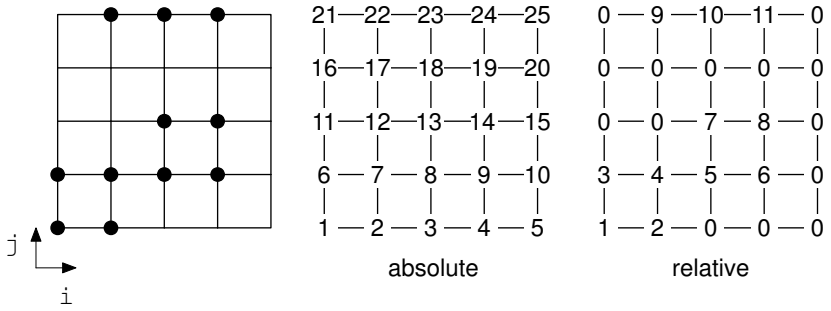


Figure 3.11. Demonstration of the set idea

In order to construct the set idea, we have to advance step by step. To start, we have to organize the unknowns. The inventory is presented in Table 3.1. i indices are given for each j with their respective absolute and relative indices. Absolute index is for the identification of the node on $\mathcal{S}_G$ and relative index is for $\mathcal{S}_L$. One imminent outcome is that i's are actually members of j's. For a given row, the columns can be

stored in vectors with varying lengths. An analogy with compressed matrix storage can also be made, so that two vectors suffice to define $\mathcal{S}_L$ as in Figure 3.12. j-pointer designates the starting point of a new row on the i-values vector.

Table 3.1. Index inventory

j	i's of j	absolute indices	relative indices
1	1, 2	1, 2	1, 2
2	1, 2, 3, 4	6, 7, 8, 9	3, 4, 5, 6
3	3, 4	13, 14	10, 11
5	2, 3, 4	22, 23, 24	12, 13, 14

Although, such a similarity can reduce to efforts to design of the solver, use of multiple-dof models and utilization of `bc` concept (introduced later), requires relatively complicated routines. Still, to increase the performance of the computations, i and j's can be compressed in single vectors like in Figure 3.12. However, pedagogical concerns are also taken into consideration while developing the solver such that the introduction of a discrete model becomes simpler. If the user can visualize the geometry with its grid elements, discretizations can be carried into the program without extra difficulty. Hence, for the ease of implementation, some performance improvement is sacrificed.

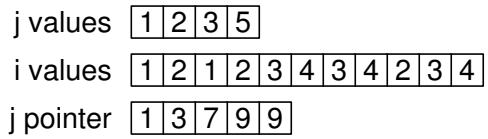


Figure 3.12. Analogy to CRS (Saad, 2003)

Instead of using single vectors, a tree-like structure can be used to define the indices. For this current local set, the tree can be formed as in Figure 3.13. `jVec` stores the indices of the j's. i indices for each j are stored in vectors called also `j`. These j vectors vary in size so a two dimensional array for all indices cannot be created. Instead, `type` property of FORTRAN is used to declare these vectors. Types allow creating combination of FORTRAN intrinsic variables, a real and integer can be combined in a new variable type (Chapman, 1998). Assuming we are operating on the "domain" as

depicted in the figure, jVec can be accessed as `domain%jVec`. "%" is used to reach the sub element of any type definition. In this example, "domain" is a type that contains `jVec` and `j`'s.

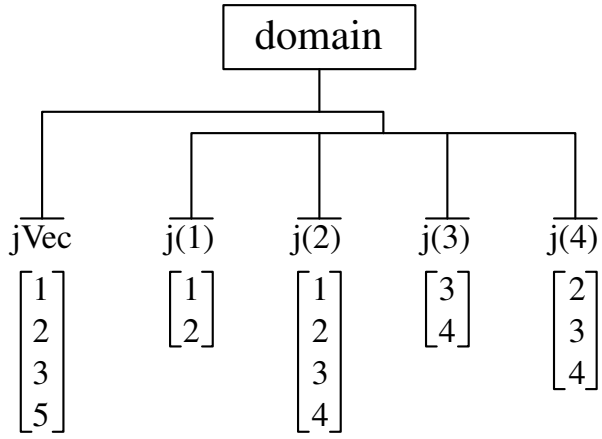


Figure 3.13. Set Tree

In a similar fashion, the entries of individual `j` vectors are reached with `domain%j(n)%iVec` n being the index of a particular j on `jVec`. Here, another vector, `iVec` is used. `iVec` is employed for convenience so that different row's on `IM` can be differentiated. An additional note is that, `j` is actually a `type` so `iVec` is a member of each `j` vector. To clarify, two examples might be used. First example is to call the indices of fifth row. Fifth row is the fourth member of the set: `domain%jVec(4)=5`. To find second column of fifth row, following call is needed: `domain%j(4)%iVec(2)=3`. Second example is conversion of Table 3.7 to Figure 3.14. As observed, the flow of the do-loops is changed they operate of myI and myJ rather than working on i and j, directly. These indices are extracted indirectly from `iVec` and `jVec` thus extra code lines are added under the do-lines. If $\mathcal{S}_L \equiv \mathcal{S}_G$ then `jVec` contains all rows and `iVec` contains all columns. When the form given in Table 3.14 (on page 40)is used, `discrete` can be defined regardless of the knowledge of the current working set. What becomes important is the formation of the tree structure.

As we progress towards the final form of discrete, some additional obstacles should be taken out. What is missing in the present form is the distinctions of the boundary conditions. Till this end, the problem was involved with homogenous boundary conditions. The declaration $discrete = x$ automatically defines the boundary conditions. Normally, separate loops are needed for each boundary line. Consider Figure 3.15. In this model, bottom boundary condition is change to homogenous Neumann boundary condition so an additional loop is needed for the definition of the discretization.

```
...
    function discrete(x)
            discrete=x

            do myJ=1,size(domain%jVec) ! access to jVec
                j=domain%jVec(myJ)

                do myI=1,size(domain%j(myJ)%iVec)
                    i=domain%j(myJ)%iVec(myI) ! access to iVec of j(mjJ)

                    TC=x(IM(i,j,1))
                    TW=x(IM(i-1,j,1))
                    TE=x(IM(i+1,j,1))
                    TN=x(IM(i,j+1,1))
                    TS=x(IM(i,j-1,1))
                    discrete(IM(i,j,1))=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0

                end do
            end do
    end function discrete
...
```

Figure 3.14. Conversion of the code on Table 3.7 to set format

On the other hand, the tree structure cannot differentiate boundary conditions with interiors. As in Figure 3.16, some unknowns of the set are on the boundary. However, created set includes all points of $\mathcal{S}_L$. That means there is a need for an additional

layer between the "domain" and j's which is called as `bc`. Although the code in Figure 3.15 is fairly simple, the boundary conditions are still defined as extra code lines. So, introduction of `bc` will not require extra effort to implement a discretized function - only the solver is getting more complex.

```
...
    function discrete(x)
            discrete=x

            do j=1,1 ! instead of this loop, j=1 could be written explicitly
                do i=1,n
                    TC=x(IM(i,j,1))
                    TN=x(IM(i,j+1,1))
                    discrete(IM(i,j,1))=(TN-TC)/h
                end do
            end do

            do j=2,n-1
                do i=2,n-1
                    TC=x(IM(i,j,1))
                    TW=x(IM(i-1,j,1))
                    TE=x(IM(i+1,j,1))
                    TN=x(IM(i,j+1,1))
                    TS=x(IM(i,j-1,1))
                    discrete(IM(i,j,1))=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0
                end do
            end do
    end function discrete
...
```

Figure 3.15. CODE Listing for Equation 3.3 with `IM`

In DEMONA, interior nodes are flagged with 1 and numbering of boundary conditions start from 2. Then different `jVec`'s will be defined for each `bc`. With bc(1) to denote the interior and bc(2) to boundaries, `j` indices can be defined as `domain%bc(1)%jVec=(2,3)`

and `domain%bc(2)%jVec=(1,2,5)`. Note that $j = 2$ is defined for both bc structure. This is because second also include one boundary point. For brevity, bc based tree is also demonstrated in Figure 3.17. Figure 3.8 on page 35 is now clearer with the explanation of the `bc` concepts. Before moving into the next section `bc` version of previous code on Figure 3.15 is presented in Figure 3.18.

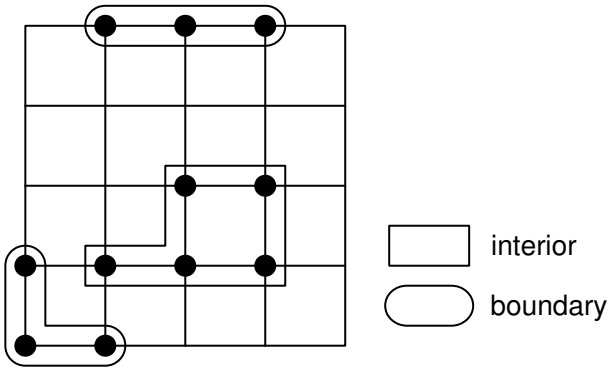


Figure 3.16. Interior and boundary nodes

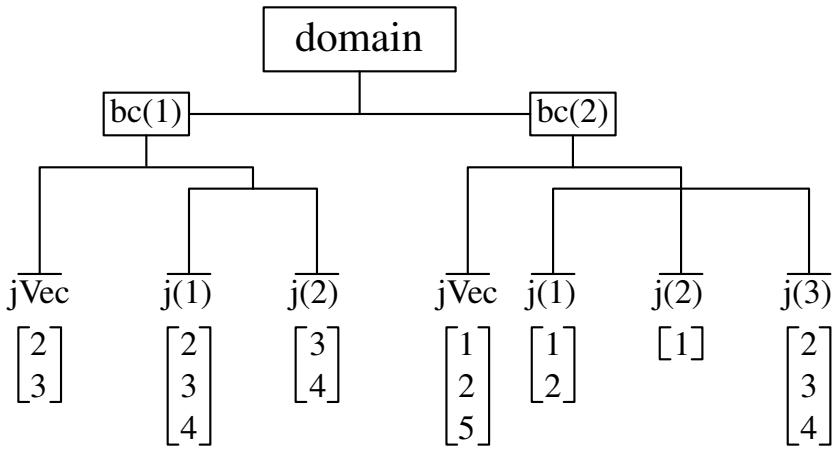


Figure 3.17. Set Tree with `bc`

```
...
    function discrete(x)
            discrete=x

            myBC=2

            do myJ=1,size(domain%bc(myBC)%jVec) ! access to jVec
                j=domain%bc(myBC)%jVec(myJ)

                do myI=1,size(domain%bc(myBC)%j(myJ)%iVec)
                    i=domain%bc(myBC)%j(myJ)%iVec(myI) ! access to iVec of j(mjJ)
                    TC=x(IM(i,j,1))
                    TN=x(IM(i,j+1,1))
                    discrete(IM(i,j,1))=(TN-TC)/h
                end do
            end do

            myBC=1

            do myJ=1,size(domain%bc(myBC)%jVec) ! access to jVec
                j=domain%bc(myBC)%jVec(myJ)

                do myI=1,size(domain%bc(myBC)%j(myJ)%iVec)
                    i=domain%bc(myBC)%j(myJ)%iVec(myI) ! access to iVec of j(mjJ)

                    TC=x(IM(i,j,1))
                    TW=x(IM(i-1,j,1))
                    TE=x(IM(i+1,j,1))
                    TN=x(IM(i,j+1,1))
                    TS=x(IM(i,j-1,1))
                    discrete(IM(i,j,1))=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0

                end do
            end do

    end function discrete
```

Figure 3.18. CODE Listing for Equation 3.3 with `IM` using `bc`

3.5. Index Matrix - Revisited

Previous test problem exemplifies one thing: `IM` should be modified. Use of absolute and relative indices and necessity to associate nodes with bc types, `IM` is expanded into fourth dimension as $\texttt{IM}(i, j, t, layer)$. This latest level is denoted as "layer". For the time being, three elements of this layer are `absolute`, `relative` and `boundary`. The content of absolute and relative layer of the model problem in section 3.4 is already given in Figure 3.11 but for convenience all layers including the `boundary` layer is presented in Figure 3.19. When all three layers are used together, formation of

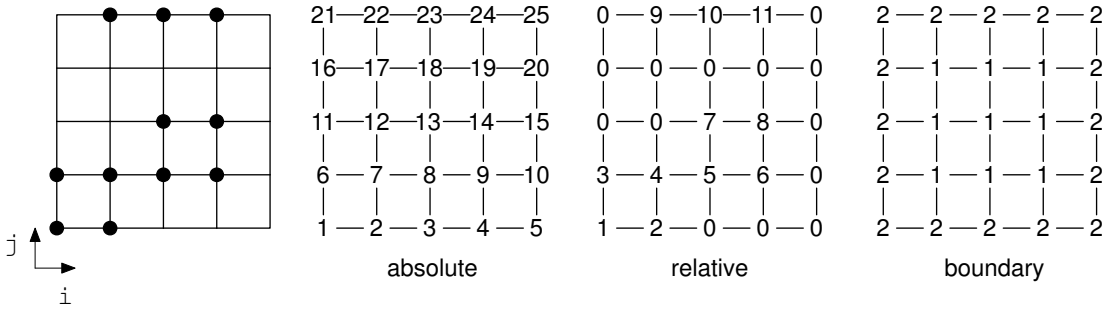


Figure 3.19. Entries of the layers

the set is straightforward. Everything seems to be appropriate except one thing:How do we know which points are in the set and how can be mark those points in $\mathcal{S}_L$? The answer for the latter question is making use of a new layer: `flag` layer. This layer is a binary layer, in other words the entries are either 0 or 1. 0 designates an unknown out of the set and 1 in-set. Using a kind of a special mechanism (which is the answer for the former question) the unknowns are flagged. Later the set routine sweeps all unknowns and checks whether the particular unknown is a member of $\mathcal{S}_L$. If not, the variable is still an element of $\mathcal{S}_G$ but will not be used in subsequent computations. For each unknown in $\mathcal{S}_L$, first `bc`'s are created. Later `jVec` and `iVec` entries are adjusted. `flag` layer for the case model is given in Figure 3.20. Reflection of these changes on IM to discrete is demonstrated in Figure 3.21. In this code segment, variables on the solution vector are called from the `absolute` layer. On the contrary, function is evaluated for `relative` layer. Refer to Figure 3.19 and note that x value at $i = 2$, $j = 2$ is x_7. Local function is, however, f_4. Since sets are to reduce the number of variables in the computations, numbering of the local functions is be carried out over `relative` layer.

With the addition of `flag` layer, `IM` is finalized to be a four dimensional array. As a summary; first two indices denote the location of the node on the grid, third index is for local dof. Final index is to select the suitable layer. For multiple-dof problems both absolute and relative numbering can be accessed with t on `IM`, nonetheless an extra level is required for the tree. This is discussed in the second case study (section 3.6).

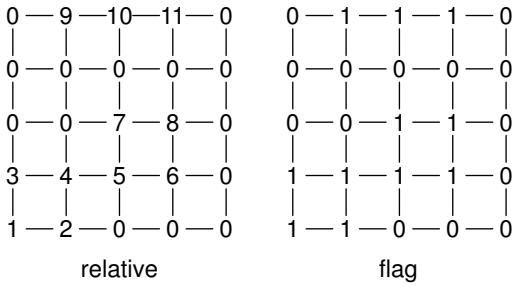


Figure 3.20. Relative values and respective `flag` layer

```
        myBC=1

        do myJ=1,size(domain%bc(myBC)%jVec) ! access to jVec
            j=domain%bc(myBC)%jVec(myJ)

            do myI=1,size(domain%bc(myBC)%j(myJ)%iVec)
                i=domain%bc(myBC)%j(myJ)%iVec(myI) ! access to iVec of j(mjJ)

                TC=x(IM(i,j,1,absolute))
                TW=x(IM(i-1,j,1,absolute))
                TE=x(IM(i+1,j,1,absolute))
                TN=x(IM(i,j+1,1,absolute))
                TS=x(IM(i,j-1,1,absolute))
                discrete(IM(i,j,1,relative))=(TW+TE+TN+TS-4.d0*TC)/h**2.d0+1.d0

            end do
        end do
end function discrete
```

Figure 3.21. CODE Listing for Equation 3.3 with expanded `IM`

3.6. Second Case Study

This is the extension of the first case study in section 3.4 to a 2-dof problem. To reflect the application of `bc` idea obvious, it is assumed that each dof has different types of boundary conditions as showed in Figure 3.22. For first variable, T, boundaries are homogenous Dirichlet type whereas for U, right and top sides are insulated. The $\times$ at the top-right corner is excluded from the discretization and its values is taken as the average of the nodes around it. Let's say a set has to be formed for the unknowns

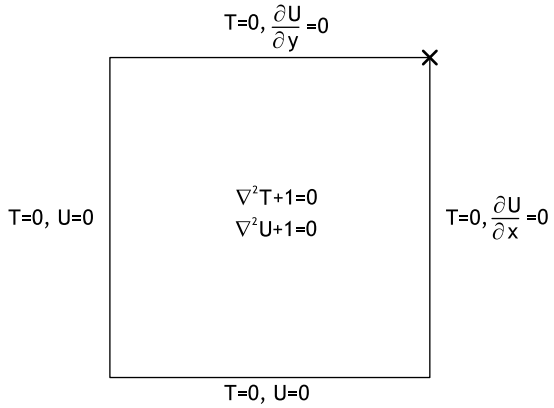


Figure 3.22. Boundary conditions of the problem

indicated in Figure 3.23. As observed, unknowns are placed randomly and also number of set nodes for both variables are not the same. In this example, it will be clear that this is not a concern, set idea works well with unequal distributions. In Figure 3.23, both `relative` and `boundary` layers should be explained. Starting with relative layer, numbering of the unknowns is performed for both dof's at the same time. While sweeping on the domain and over all degrees of freedom, next unknown on the line is numbered. Consequently, for a particular variable layer, unknowns are not in order. Still, relative numbering does not affect subsequent computations. In `boundary` layer, 5 different bc types are assigned. Also, layers for both T and U are the same. When Figure 3.22 is reexamined, it would be obvious that two types of boundary conditions are enough for T, particulary $bc = 1$ for interior and $bc = 2$ for all sides. On the other hand, U requires at least 5 types: $bc = 1$ for interior, $bc = 2$ for left and bottom, $bc = 3$ for right, $bc = 4$ for top and $bc = 5$ for top-right corner. At the time DEMONA is designed, there were two choices. Either bc's can be defined for all unknowns such that all possible combination is passed on to each variable meaning that 5 types will

be used for T and U. Or as a second idea, bc types of all unknowns will be summed up and brought on to the layers. That would mean bc's 1-2 to be associated with T and 3-7 with U. We went with the first idea. Although second idea could produce a shorter code, it is always to keep same part of the domains together even if the discretized definition of a particular variable is repeated elsewhere. With this form of the `boundary` layer implementation of `discrete` is easier. In `flag` level, entries related to the nodes are 1 and the rest is 0. A couple of examples for the set tree is presented in Table 3.2.

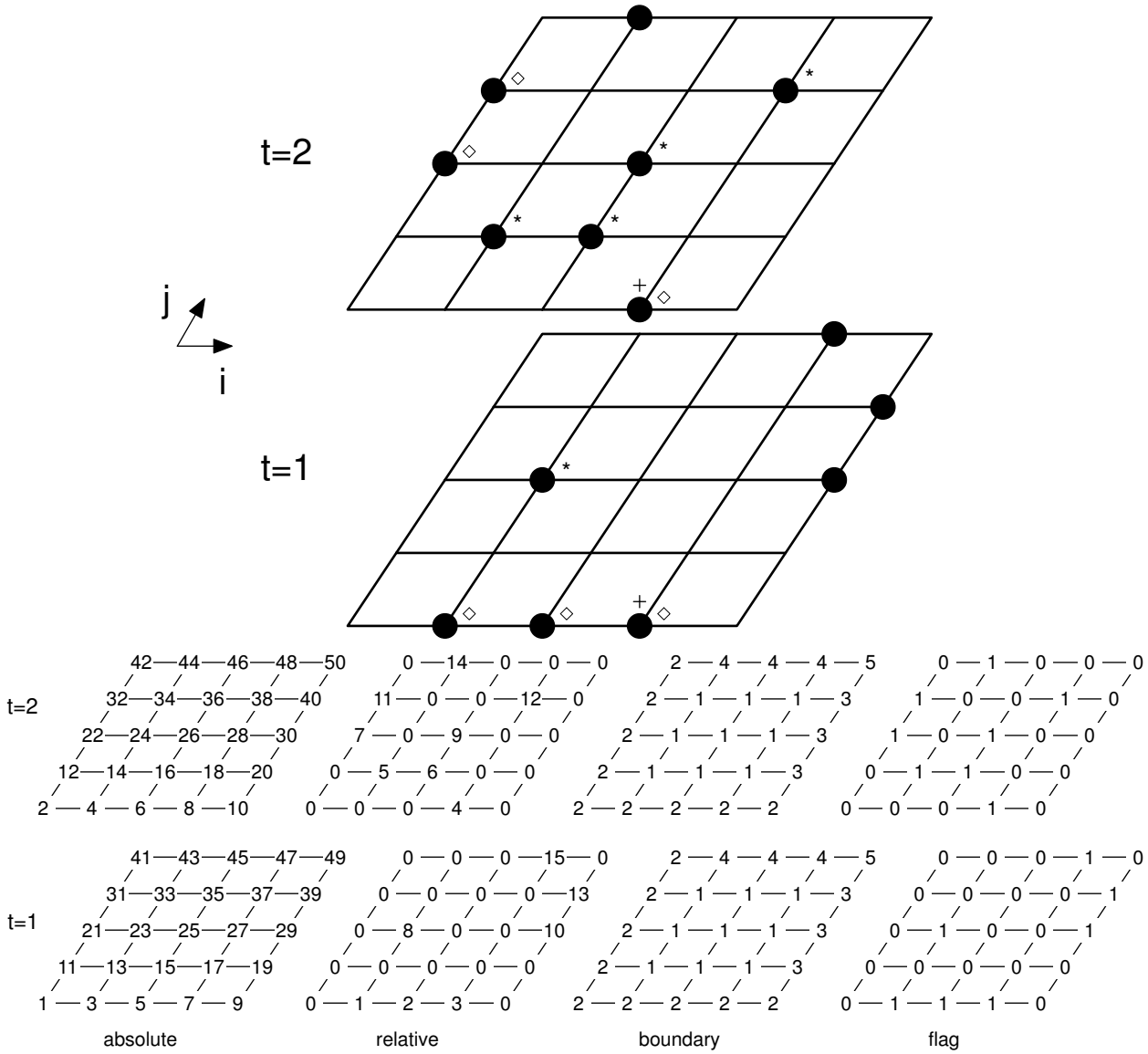


Figure 3.23. Second case study and entries of IM

While adding the multi-dof problems into DEMONA, there were two choices; either the "domain" could branch first on variables then `bc`'s or the variables might be at the end of the line - after `iVec`'s. In DEMONA, second option is preferred. During the development phase, it is discovered that starting with variables before `bc`'s requires a completely different structure where most of the routines would need a remake. Instead, unknown variables are assigned to be child members of i's. Thus, j's are decided by considering all that's why `domain%bc(1)%jVec` has three elements. If we were be using a variable based idea, then such a structure might be generated: `domain%t(1)%bc(1)%jVec=3` and `domain%t(2)%bc(1)%jVec=2,3,4`. On the contrary, with current choice, `tVec`'s are introduced as in Table 3.2.

Table 3.2. Second Case Study

domain%bc(1)%jVec	2, 3, 4 ($*$)
domain%bc(2)%jVec	1, 3, 4 ($\diamond$)
domain%bc(2)%j(1)%iVec	2, 3,4 ($\diamond$)
domain%bc(2)%j(1)%i(3)%tVec	1, 2 ($+$)

3.7. Data Structure of Demona

Up to this point, only one domain is considered and set is explained over simple problems. In multiphysics problems, however, there are more than one physics and each physics might have more than one domain. In a model, physics and domains are related as in Figure 3.24. In DEMONA, the model is called as `myModel`. The model contains a number of physics. To reach any physics, keyword `myModel%physics(p)` has to be used. If we set the index, p, properly, then we have access to all of the information of that particular physics. Each physics consists of several important parameters like `numberOfDomains` or `pDof`, total number of unknowns of that physics. The unknowns are kept in vector, `x` as `myModel%x` rather having a separate solution vector for each domain. Local `x` vectors of domains are pointers to the `x` of `myModel`. Beside `x`, a temporary vector `tempx` is also stored in the program. `tempx` is frequently used in

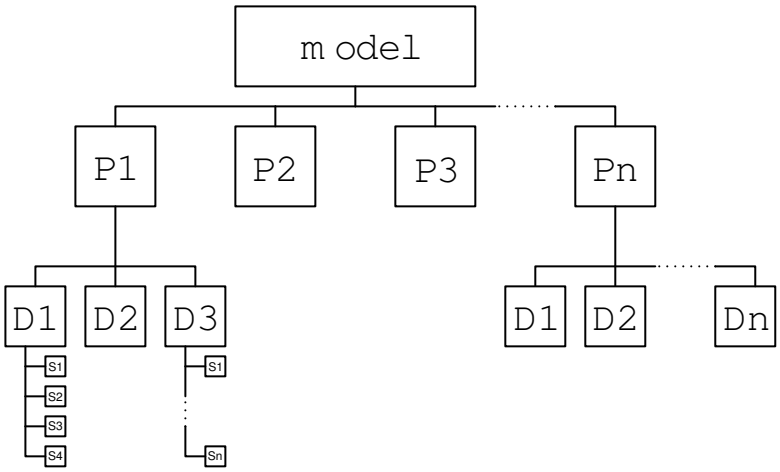


Figure 3.24. Structure of DEMONA

numerical algorithms, especially in Newton's Method to test the update. Moreover, two additional vectors are stored for real time and pseudo time computations: `xRTO` and `xPTO`, respectively.

Domains are accessed as `myModel%physics(p)%domain(d)`. Domains also have their own particular set of features like `dDof`, total number of unknowns of that domain and many more. All of the properties are listed in Appendix C. Number of unknowns for each node is also defined in the physics and inherited by the domains. That means, number of dof of all domains, that are member of a physics is the same. If a domain is to be analyzed with different number of local variables, that it should be defined as a domain of a new physics. In other words, physics is a collection of domains with same number of local dof's.

In DEMONA, nodes define the geometry and dofs define the unknowns. Index matrices, i.e. `IM`'s are stored in `domain`'s. In DEMONA they are named as `dofMatrix` to be able to separate nodes (which is called by `nodeMatrix`) with unknowns. This is because of the differences in discretization processes. In Finite Differences and Finite Elements nodes and unknowns are collocated. In Finite Volume on the other hand, nodes that define a volume do not necessarily represent the unknowns. This is especially true for cell-centered discretizations. `dofMatrix`'s are referenced by IM which is a pointer:`IM=>myModel%physics(p)%domain(d)%dofMatrix`. Similarly, `IN` is used to point the `nodeMatrix`. The solver has a simple rectangular mesh generator and the

coordinates of the nodes are stored in `domain`'s `xCoor` and `yCoor` vectors. If we need the x coordinate of the i,j, node then,

```
myModel%physics(p)%domain(d)%xCoor(...
...myModel%physics(p)%domain(d)%dofMatrix(i,j,1,relative))
```

should be called. However, this designation is a bit complex and prone to typos. So, instead of full address of the variables, pointers can also be used here and such

```
myxVec=>myModel%physics(p)%domain(d)%xCoor)
myGlobalNode=>myModel%physics(p)%domain(d)%indexMatrix(i,j,1,relative)
```

Then he same call can be made with `myxVec(myGlobalNode)`. Here, it is worth to note that, already written codes can be used such that the definitions are changed as pointers and the pointers are associated at the beginning of the routines.

Demona has three main parts namely, preprocessor, processor and postprocessor. These routines use different number of subroutines and modules where some of them are problem dependent and some of them are problem independent. In the preprocessor part, the system is initialized and the `domains`' `dofMatrix`'s and coordinates are generated. In the processor part, the problem is solved. The interactions between domains as well as physics are organized here. If a FMG type solution routine is needed, then the sub domains are also created here. Time dependent implicit solvers with Newton-Krylov and Multigrid methodologies are controlled in processor, as well. The postprocessor arranges the result for plotting and also can perform additional post-analysis like calculation of friction coefficient of extraction of some data to be used in next computations or to be compared with other studies in the literature. Additional detail on DEMONA is given in section 8 as well as in proceeding sections.

4. NUMERICAL METHODS

Governing equations of multi-physics problems are nonlinear, coupled partial differential equations. Discretization of the models results in nonlinear system of equations which have to be solved with proper numerical techniques. This section serves as an introduction to methods for solving linear as well as nonlinear systems of equations. The techniques presented here are essential to understand the methodologies that are used in this thesis. The section starts with the solution of linear systems and then it extends to nonlinear problems. Newton's method and its variations will be explained in conjunction with linear iterative methods. Linearization of nonlinear problems is also considered. DEMONA related implementation details are given if necessary. To clarify the terminology, a glossary is given in Appendix D where some of the important keywords are explained. It is a good idea to look and come back time to time, if the jargon looks unfamiliar. Additionally, refer to Figure 4.3 for overview of the methods.

In this study main object is to use iterative methods rather than direct solution techniques or analytical means. Iterative methods are powerful tools in scientific computing. For large scale computations two directions are present. One is the use of Newton-Krylov techniques in which Newton's Method (along with its inexact variants) is the nonlinear solver and Krylov Subspace methods are used to deal with the linear correction equation. Second idea is to use multigrid (section 5) whose generalization yields to multilevel techniques. In multigrid, a smoother is used to attack the problem and the various grid levels are used to reduce the error. Before giving the details of nonlinear solvers, linears solvers should be examined, first.

4.1. Solution of Linear Systems

This part illustrates some of the known iterative solvers that are frequently used in this thesis as well as in the literature. Our object is to find x, the solution of Equation 4.1. A is a matrix that contains the coefficients of each linear equation and b is a vector filled with the constants that system is equal to. In finite elements, this

equation is generally presented as $Ku = f$ where K is called the stiffness matrix and f as the forcing function which lead us to the solution of nodal values, u.

$$Ax = b \tag{4.1}$$

Linear problems are frequently encountered in scientific computing. Usually it is a sub problem contained by a nonlinear system. A good understanding of linear systems is necessary to cope with complex problems. Underlying idea of this section is to bring in the iterative methods but first direct solvers deserve some attention. A direct solver is a kind of solution techniques that gives an answer with no error[6] . The result should be as accurate as the machine precision. Most famous direct method is the Gaussian Elimination; where the progress of the technique is from the diagonal of the first row to the diagonal of the last row by eliminating part of the columns under the diagonals. It might be necessary to perform pivoting for avoiding zero diagonals. When last row is reached, then the solution can be found by backward substitution. The idea is simple yet has its own difficulties. This method is extremely useful for small problems, however, as the degree-of-freedom of the model increases, the solution takes much more time since this approach is based on $O(n^3)$ operations. High dof cannot be dealt with Gaussian Elimination. When the problem results in a sparse system - which is the case for discretized equations, then the zero entries are filled with intermediate values and stay there until the solution is get. Since main issue in solving large problems is the storage of the unknowns, special methods like Compressed Row Storage (CSR) (Barrett *et al.*, 1994) are used to reduce the number of entries and the fill in's deteriorate the structure of the storage scheme. Of course, depending on the problem - the structure of the matrix could be guessed beforehand and specific fast solvers can be devised. This kind of solvers, like frontal solvers or TDMA (when ADI is used as the linear operator) are frequently used in commercial packages till reasonable dof's. However, another issue reveals itself: round-off errors start to accumulate and spoil the solution. At the end, the solution should be exact; however, it might not satisfy the linear problem when substituted. Those problems force us to use iterative methods instead of direct

[6]This is the numerical error; discretization error is definitely introduced when continuous problem is converted in to a discrete problem by means of FD, FV, FE or any other technique

methods. Main advantage of iterative methods are that, they generate new solution at each iteration - so round-off errors are out of question unless very strict convergence tolerances are used. Some iterative methods are simple to implement and some of them not. However, the convergence properties are enhanced for advanced techniques. Another benefit of iterative methods is that they can be used together to improve the performance of the solver.

4.1.1. Iterative Methods

As explained by Barrett *et al.* (1994), iterative linear solvers can be divided into two categories: Stationary and Non-Stationary solvers. Stationary solvers are based on different forms of splitting of matrix A in Equation 4.1. Jacobi, Gauss-Seidel (GS), and Successive Over Relaxation (SOR) are basic examples of stationary solvers. Non-stationary solvers, however, performs matrix-vector computations based on A. Krylov Subspace solvers fall into this category; Conjugate Gradient (CG), GMRES and BiCGStab are frequently used Krylov solver in numerical science community.

General form of Stationary methods can be given in 4.2 following the notation of Barrett *et al.* (1994). B is the iteration matrix and its form is taken from A. c is a vector and its form is adjusted with Equation 4.1. At the beginning of iterations, B and c are set and they remain the same until the problem is solved meaning their values does not change with iteration number. That's why they called stationary Methods.

$$x^{k+1} = Bx^k + c \tag{4.2}$$

Various forms of B and c for different solvers are given in 4.1. Splitting for A is given as in Equation 4.3 where D is the diagonal and L and U are strictly lower and upper triangulars of A, respectively. Main characteristic of Jacobi method is that the x^k kept the same until $k+1$'th iteration finishes. Gauss-Seidel on the other hand starts using the values on x^{k+1} as soon as they are calculated. This approach improves the solution of the linear system. SOR is a modification on Gauss-Seidel method. Relaxation parameter ω can be set such that a further improvement over GS

is achieved. Although, a value for ω can be selected in $(0, 2)$, underrelaxation ($\omega < 1$) might be necessary to ensure convergence on stiff problems. This is especially true if GS is used directly on a nonlinear problem with scalar Newton's Method. Similar to the SOR case, Jacobi Method can also be used with damping. In that case, $x_i^{k+1} = \omega x_i^{k+1} + (1-\omega)\, x_i^k$. Underdamped Jacobi method definitely requires more iteration to converge and normally one is doubtful with its use. Actually, as will be seen in section 5, it is an excellent solver for Multigrid.

$$A = D - L - U \tag{4.3}$$

In table 4.1, inverse forms of the matrices are given. In reality, no inverse is computed.

Table 4.1. Common Stationary Methods (Barrett *et al.*, 1994)

Method	**B**	**c**
Jacobi	$D^{-1}(L+U)$	$D^{-1}b$
GS	$(D-L)^{-1}U$	$(D-L)^{-1}$
SOR	$(D-\omega L)^{-1}(\omega U + (1-\omega)D)$	$\omega(D-\omega L)^{-1}$

The methods can only be given in matrix form as in the table, however, in practice, the relaxations are matrix independent i.e. the computations are handled entry based. In fact, one should strictly avoid taking inverses of matrices when dealing with large scale problems mainly because taking an inverse is actually applying Gaussian Elimination on the system and that takes too much time. Additionally, linear systems generated with discretizations tend to be ill-posed and computations of inverses might fail. Another pitfall is on storage of the elements of A^{-1}. Even if A is sparse, its inverse is not sparse as observed in Figure 4.1. A on the left is the discretization of 2D Poisson's equation on 9x9 grid. As seen from the figure, A has only five diagonals but A^{-1} is filled with values[7] . Of course, it would be the best if $B = I$ and $c = A^{-1}b$ such that with a zero initial guess ($x^0 = 0$) the solution could be found in just one step. Although this ideal case is like a pipe dream for real scale problems, it is not fully

[7]of course some of the entries are very small compared to A, this is the motivation behind ILUT preconditioner (Saad, 2003) where a drop tolerance is used to eliminate those entries

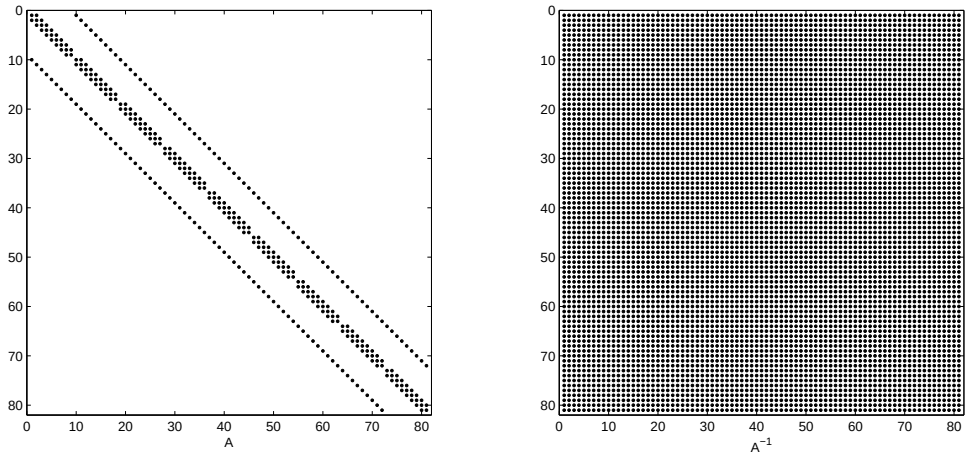


Figure 4.1. Spy plot of A and A^{-1}

disregarded. Essentially, finding matrices that resembles A^{-1} brings us to the notion of preconditioners explained in section 7.1 and in that respect stationary iterative solvers are excellent candidates of being preconditioners.

There are also more stationary iterative methods than these three ideas, red-black GS, symmetric versions of GS and SOR are also used at which sweeps on the domain are done forward and backward repeatedly. Incomplete LU decomposition is also a good solver but it is generally used as a preconditioner so details are given in section 7.1. A review of iterative linear solvers is given by Saad and van der Vorst (2000). More details on iterative linear solvers can be found in Barrett *et al.* (1994), Kelley (1995), Greenbaum (1997), van der Vorst (2003) and Saad (2003).

4.1.2. Krylov Subspace Methods

Krylov subspace methods based on orthogonalization procedures. A subspace is generated with the powers of A and a vector v. Several techniques can be used to generate and orthogonal basis (Saad, 2003). In GMRES, Full Orthogonalization

Method (FOM) and CG, modified Gram-Schmidt or Householder version of Arnoldi Orthogonalization are used. BiCGStab and alike solution methods (BiCG, CGS), however, based on Lanczos' Biorthogonalization. Implementation of these solvers is not trivial but Krylov Solvers have the ability to perform matrix-free computations which justifies their use. Although the concept is explained later in this chapter, it could be defined here. Matrix-free computations is based on the fact matrix A in Equation 4.1 is not needed anymore. All of the Krylov solvers work on A (some of them also need A^T) but actually they do not need it explicitly. Rather, they need the result of a matrix-vector product. This product can be calculated w/o a reference onto A. This is particularly useful in solving nonlinear system. Only a vector - nonlinear function, F is enough to perform all of the computations contained in the algorithm of any Krylov solver (except the cases where A^T is needed).

For real problems, i.e. if A is not complex, there are two possibilities for the structure of the coefficient matrix; it is either symmetric or non-symmetric. When isotropic equidistant diffusion problems are discretized, then the resulting A is symmetric. For symmetric problems, Conjugate Gradient (CG) is the best choice (van der Vorst, 2003) among all other Krylov Solvers. In the case of a non-symmetric problem - which is the case in computational fluid dynamics - there are many possibilities. In this thesis, only two of them are used, Bi-Congugate Gradient Stabilized (BiCGStab) and Generalized Minimum Residual (GMRES), because of their attractive features. Other solvers include BiCG,CGS, QMR, TFQMR, MINRES, ORTHOMIN, ORTHODIR and many others (Broyden and Vespuci, 2004) might perform well in specific problems, however, to be general, we stick on those two techniques. There are a vast amount of work on these solvers, nevertheless, essentials can be found in references Saad (2003), van der Vorst (2003) and recent progress is described in the review paper of Simoncini and Szyld (2007) and references there in.

4.1.2.1. Bi-Conjugate Gradient Stabilized. BiCGStab is based on Lanczos' Biorthogonalization. It is first proposed by van der Vorst (1992) as a smoothly converging variant of Conjugate Gradient Squared. CGS is an improvement on BiCG method

in which need for A^T is eliminated. However, there recurrence in CGS based on the square of the matrix-polynomials which deteriorate the convergence history, error can accumulate in powers of two. Van der Vorst idea is to replace one of the polynomials with another form that can stabilize the solution. As mentioned before, two vectors are selected at the beginning of the iterations. First vector is the initial residual, r^0, and generally the second vector is selected equal to r^0. In this method, residual polynomial is stabilized by introduction of a parameter ω which is selected such that steepest descent is achieved in the direction of the residual. Use of recurrence relations is useful in numerical science since storage is one of the main concerns. Main advantage of BiCGStab is that it requires only two Matrix-Vector products per iteration and stores only four vectors during the computations. Downside of the method is that it does not promise a non-increasing residual and during the iterations, residual might increase unexpectedly but not as dramatic as BiCG or CGS and can be controlled with proper preconditioning. BiCGStab can be applied on non-symmetric linear problems and generally it is considered as a rival of GMRES because its flexibility in storage. If the storage is the main concern, BiCGStab is an excellent choice(Barrett *et al.*, 1994). As any other Lanczos' based method, BiCGStab might also fail during the iterations (w=0 case for instance). Preconditioning helps to avoid the difficulties but more robust solvers can always be used, GMRES for instance.

4.1.2.2. Generalized Minimum Residual. GMRES of Saad and Schultz (1986) is a powerful Krylov solver for non-symmetric problems. It is based on a least squares minimization of the residual. Main advantage of GMRES is that it is not required to update the solution vector during the iterations and the update can be performed at the end of the computations. The convergence is monitored with the norm of the residual which is a by product of the algorithm and does not have to be calculated explicitly. Another advantage of GMRES is that it promises a non-increasing residual norm. In GMRES, a set of orthonormal vectors is created with Arnoldi orthogonalization process and stored as columns of V. First column of V is the normalized residual vector. The dot products are stored in Hessenberg matrix, which is later transformed such that the residual of the system is minimized.

A useful relationship can be derived to investigate the residual as in Equation 4.4.

$$b - Ax = V_{m+1}(\beta_g e_1 - \bar{H}_m y) \tag{4.4}$$

Then it can be shown that in order to minimize the residual, one should work on Equation 4.5 and find y_m

$$y_m = argmin||\beta_g e_1 - \bar{H}_m y||_2 \tag{4.5}$$

In order to facilitate Equation 4.5, some plane rotations must be applied on Hessenberg matrix. Rotation angles for i'th transformation are determined from the Hessenberg matrix that is rotated $i - 1$ times. At the end, Equation 4.5 turns into Equation 4.6 where $\bar{R}_i$ denotes the transformed H matrix after i'th transformation. $\bar{R}_i$ is an upper-triangular with an extra row full of zeros.

$$y_m = argmin||g_i - \bar{R}_i y||_2 \tag{4.6}$$

Last element of $\bar{g}_i$, denoted as γ_{i+1} is the residual. The residual can be monitored after each orthogonalization process. It can be shown that a relation for the residual can be formulated as in Equation 4.7

$$\gamma_{i+1} = (-1)^i \beta_g \prod_{j=1}^{i} \sin\theta_j \tag{4.7}$$

Since sin function is bounded between $0 < \sin\theta_j < 1$, the residual will not increase throughout the process. Happy breakdown can occur if the angle of the transformation becomes 0 so the result will be exact. The procedure is completed with Equation 4.8

$$x = x_0 + V_m y \tag{4.8}$$

where $V_m y$ is the m'th Krylov subspace K_m. An important drawback of the algorithm is that need for storage increases linearly as the orthogonalization proceeds. For a system with N unknowns, GMRES guarantees the exact solution in N steps, however generally the procedure continues till m number of orthogonalization with $m \ll N$. When m is reached but the tolerance criteria is not fulfilled, then an intermediate result can be calculated with Equation 4.8 and the process may begin from scratch with same number of steps. This is called restarting where it is denoted as GMRES(m). Number of the restarts depends on the implementation, however, as long as the convergence criteria not satisfied, iterations should continue. Full algorithm and details of the implementation can be found in Saad and Schultz (1986); Saad (2003).

4.1.3. Split Linearized Forms

Before moving into the analysis of nonlinear problems, three linearization processes should be explained. Alternating Direction Implicit is a method where the governing equations are solved implicitly in one spatial direction and treated explicitly in all other directions. Consequently, n-dimensional problem is solved in n steps per iteration. This method is extremely useful to get rid of the problems with the nonlinearities specifically in convective terms. Main issue in ADI is that the convergence is achieved in more iterations but it can be used with Multigrid such that its smoothing properties is exploited. As an example, consider the conduction equation in 4.9.

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{4.9}$$

When x-components are considered, operator on the first step is as in Equation 4.10. In this formulation, we are trying to solve for unknowns at level $m + 1/2$ with the assumption that the unknowns at level m are known. The resulting system has three components per row so TDMA can be used to solve the problem. When we move to second coordinate, then the formulation is Equation 4.11.

$$\frac{T_{i-1,j}^{m+1/2} - 2T_{i,j}^{m+1/2} + T_{i+1,j}^{m+1/2}}{dx^2} + \frac{T_{i,j-1}^{m} - 2T_{i,j}^{m} + T_{i,j+1}^{m}}{dy^2} = 0 \tag{4.10}$$

$$\frac{T_{i-1,j}^{m+1/2} - 2T_{i,j}^{m+1/2} + T_{i+1,j}^{m+1/2}}{dx^2} + \frac{T_{i,j-1}^{m+1} - 2T_{i,j}^{m+1} + T_{i,j+1}^{m+1}}{dy^2} = 0 \tag{4.11}$$

In this case, we are solving for level $m+1$ i.e. for y-direction and assume that variables at level $m + 1/2$ are known. In DEMONA, ADI can be applied only with the use of special parameter that multiplies x and y derivatives separately. Computations stop when m and $m + 1$ level results are similar.

Picard's technique is also a kind of a splitting. However, this time, instead of splitting the connection between the spatial directions, local dof's are separated. Taking the stream function - vorticity formulation as an example, Equation 2.6a is solved first by taking vorticity, ω, constant and Equation 2.6b is solved with freezing the values of stream function, ψ. In this way, coupled nonlinearity in the convective terms are eliminated. In Picard's Method, an iteration finishes after dof number of intermediate computations. For steady state SFV formulation, first step of Picard is like in Equation 4.12 where vorticity at level k is taken as constant. This form of equation is, in fact same as Poisson's Equation with varying coefficients all over the domain.

$$\frac{\psi_{i-1,j}^{k+1/2} - 2\psi_{i,j}^{k+1/2} + \psi_{i+1,j}^{k+1/2}}{dx^2} + \frac{\psi_{i,j-1}^{k+1/2} - 2\psi_{i,j}^{k+1/2} + \psi_{i,j+1}^{k+1/2}}{dy^2} + \omega_{i,j}^{k} = 0 \tag{4.12}$$

Second step is involved with the solution of vorticity transport given in Equation 4.13. This time ψ values are regarded as constants. In this formulation, resulting system is linear in deed. Nonlinear coupling in convective terms are linearized so any linear solver can be used to find the solution.

$$\left(\frac{\psi_{i,j+1}^{k+1/2} - \psi_{i,j-1}^{k+1/2}}{2dy}\right)\left(\frac{\omega_{i+1,j}^{k+1} - \omega_{i-1,j}^{k+1}}{2dx}\right) - \left(\frac{\psi_{i+1,j}^{k+1/2} - \psi_{i-1,j}^{k+1/2}}{2dx}\right)\left(\frac{\omega_{i,j+1}^{k+1} - \omega_{i,j-1}^{k+1}}{2dy}\right)$$
$$-\frac{1}{Re}\left(\frac{\omega_{i-1,j}^{k+1} - 2\omega_{i,j}^{k+1} + \omega_{i+1,j}^{k+1}}{dx^2} + \frac{\omega_{i,j-1}^{k+1} - 2\omega_{i,j}^{k+1} + \omega_{i,j+1}^{k+1}}{dy^2}\right) = 0 \tag{4.13}$$

Please note that, use of ADI linearizes SFV formulation as well. This is mainly because the convective terms are cross terms wrt the coordinate system. Hence, either directional or component wise splitting works for this problem. On the other hand, if we

are after solving the flow variables, p, u, v directly in primitive variables then neither Picard nor ADI is enough to linearize the convective terms. When Equation 2.9 is examined, it is apparent that for example, $u\frac{\partial u}{\partial x}$ term stays nonlinear. This observation brings us to a third kind of a splitting: Operator Splitting (OS).

In operator splitting, the separation of the field variables is considered based on their physical meanings. Taking the primitive variable formulation of incompressible Navier Stokes Equation, $\frac{\partial u_i}{\partial x_j}$ terms in convective parts are actually carried by u_1 and u_2 (i.e. with u and v). Hence, considering x-momentum Equation 2.9b as an example, $u\frac{\partial u}{\partial x}$ term can be treated as $u_k\frac{\partial u_d}{\partial x}$, u_k being the kinematic velocity and u_d being the dynamic velocity (Ilıcak *et al.*, 2007). Later, the constraint will be enforced such that both velocities are equal to each other. Use of OS in this model problem increase number of unknowns by two: $p, u, v \rightarrow p, u_k, u_d, v_k, v_d$.

Before ending this section, few remarks should be stated. These linearized forms can be solved directly, or with the aim of stationary or non-stationary iterative methods. Depending on the solution process, modifications on the discrete operators are possible like in ADI as explained in Hoffmann and Chiang (2000). These methods require a lot of iterations to converge but highly nonlinear problems can be relaxed with the use of these techniques. Damping parameters can be used to accelerate the computations. In Picard, underdamping is frequently used, especially in primitive forms, so that the convergence is slow but more likely. Slowly converging problems can be accelerated with the use of Multigrid (section 5). All of these techniques can be combined; for example, each Picard solve could be performed in ADI basis. Finally, using these ideas hybrid solvers can be generated. In hybrid idea, solution process starts with these methods and then at some point we switch to direct coupled solvers like Newton's Method. In this setup, split formulations improve the initial guess.

4.2. Solution of Non-linear Systems

Although many engineering problems can be defined with linear PDE's, like heat conduction or diffusion, most of the problems are nonlinear, especially in Fluid Mechanics. For example, convective terms in Navier-Stokes Equations are the main reason for the complications arising in the computation of the flow field. Most of the effort is given to treat these terms in such a way that the nonlinearity is some what relaxed (like in ADI or PICARD) and also to solve the arising equations efficiently. In this thesis, nonlinearity is attacked directly with the use of Newton-Krylov Techniques (section 4.2.2) as well as Full Approximation Scheme (section 5.3). Although, this thesis is based on Newton's Method, another type of solution technique is also used to solve nonlinear equations, Secant Method (Dennis and Schnabel, 1996). For scalar equation, Secant Method provides a formulation without a derivative. Resulting algorithm is not as fast as Newton's Method, however, the formulation can be defined in a recursive manner where the system matrices can be used with Broyden's update. A comprehensive study on Fast Secant Method is provided by Deuflhard *et al.* (1990).

4.2.1. Newton's Method

Newton's Method[8] is one of the famous methods for solving nonlinear problems. Newton's method basically works on an initial iterate that is updated via a linear correction equation to find next candidate solution vector. When the vector containing each nonlinear function is denoted as F then x is to be found that satisfies Equation 4.14. Newton's method is generally formulated for scalar problems but its derivation for problems with more than one unknown is straightforward. $F(x^0 + \Delta x)$ can be expanded for x^0 using the Taylor series as in Equation 4.15.

$$F(x) = 0 \tag{4.14}$$

[8]This method is also called as Newton-Raphson or Newton-Raphson-Simpson or even Newton-Simpson Method (Deuflhard, 2005)

$$F(x^0 + \Delta x) = F(x^0) + F'(x^0)\Delta x + H.O.T. \tag{4.15}$$

By neglecting the high order terms and assuming that $x = x^0 + \Delta x$ satisfies Equation 4.14 then we get an equation for Δx. Replacing $F'(x)$ with $J(x)$ the correction can be defined as in Equation 4.16. Here $J_{ij} = \frac{\partial f_i}{\partial x_j}$ is the Jacobian evaluated at x^0.

$$J(x^0)\Delta x = -F(x^0) \tag{4.16}$$

Equation 4.16 is similar to Equation 4.1. That means, every solution method applicable for linear systems can also be used for this correction equation. Depending on the linear solver, Newton Method can be called Newton-Jacobi, Newton-GS, Newton-Krylov or even Newton-Multigrid(MG).Newton's method has a superior advantage compared to any other method: it has a quadratic convergence rate if the iterate is close to the exact solution. Otherwise, numerous tricks should be used to make the method globally convergent (section 4.2.4). As seen in Figure 4.2 the method converges rapidly at the end of the computation. 87'th step being the last step, the calculation of the updates proceeds slowly till 85'th step and then quadratic convergence rate is observed. If one could have given the solution at the 85'th step as an initial guess, then it would be observed that the method converges just in three steps. However, such a lucky guess is generally not possible (except the strategies explained in section 7.3) so utilization of a global convergence method is crucial. In this figure, global convergence methods are effective from the first step to 85'th step.

There are, however, a bunch of disadvantages of Newton's Method which we try to eliminate with the techniques offered throughout this chapter. First of all, at each iteration a linear system should be solved. If a simple method like, Jacobi, is selected as the default linear solver, then more computations will be needed to find x. As a result, the linear solvers should be as fast as possible. Another drawback is that at each step the Jacobian should be evaluated and stored. As mentioned before, stationary methods do not really need a coefficient matrix but the definition of the Jacobian should be given analytically or computed by numerical means. Analytic

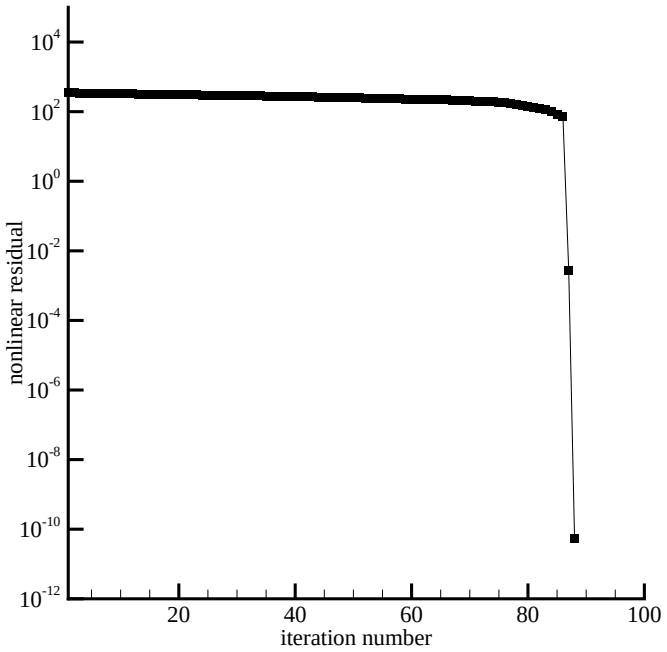


Figure 4.2. Nonlinear convergence history of a SF-VOR equation for $Re = 100$ in a 33x33 grid

evaluations are normally the ultimate way of computing the Jacobian. Unfortunately, except simple problems, implementation of the Jacobian is a tedious task and error prone for multi-dof nonlinear equations. Additionally, there is no clear cut method to calculate the derivative of functions with discontinuities. Furthermore, a modification on F, use of a new or high order discretization for instance, will require an update on the analytic Jacobian - that lacks flexibility for the nonlinear solver, as well as the study.

Numerical calculation of the Jacobian one-by-one is also problematic. First of all, calculating each entry will take at least $n \times (n + 1)$ operations for scalar functions. Yet F is defined as a vector so n times more computations will be needed and $n - 1$ times operations are wasted. Instead, each column of the Jacobian can be calculated by performing a matrix-free multiplication with a vector filled with 1's only. As a result, n times evaluation of F will suffice. Still there is a lot of computation and this can be reduced with color-based computation as presented in section 7.1.1. Selection

of the variation parameter in the numerical differentiation is another concern that is addressed in section 4.2.2. All what is said forces us to change the direction of the linear solver. If one can use a method that does not really need the Jacobian, than we get rid of a lot of computational load. This observation leads us to the Newton-Krylov Techniques.

Stopping criteria of the Newton's Method is based on the 2-norm of $F(x)$. Whether an absolute or relative convergence will be pursued is analyst dependent. Kelley (1995) proposed a combination of both cases as in (4.17). It might happen that the update Δx is too small but there observed no relative reduction in the norm of F. That would mean that a local minimum is found instead of a global one. In that case, the problem should start with a different initial guess. So, the change in the update should also be monitored (Dennis and Schnabel, 1996). A variant of the step size tolerance is given by Shadid *et al.* (1997).

$$||F(x^k)||_2 \leq tol_{rel}||F(x^0)||_2 + tol_{abs} \tag{4.17}$$

4.2.2. Newton-Krylov Techniques

In Newton-Krylov Method, Equation 4.16 is solved with one of the Krylov Methods. Motivation behind Newton-Krylov techniques is that while performing matrix-vector products, the so called Matrix-Free methodology can be employed. In matrix-free computations, the Jacobian, J is never needed (except preconditioning which is explained in section 7.1). This provides two apparent advantages: Jacobian will neither be computed nor stored. Avoiding formation of the Jacobian provides huge flexibility on the analysis, such that changes on the models can be applied with minimum effort. Matrix-Free (MF) approach reduces the complexity of introducing a new problem. Only the discretization information is provided to the solver. As a result, the robustness of the solver becomes very important since this "black-box" like solver should handle a wide range of problems. Meanwhile, a clear definition should be made. Matrix-free means there is no stiffness matrix stored in the problem. In that sense, stationary

linear solvers are also matrix-free. The point where stationary and non-stationary methods differ is that in NK, the computations are performed at once. That means, NK is vector based rather being scalar based. Vector computing is advantageous in parallelization but that does not necessarily mean that stationary methods are not parallelizable - Jacobi Method is very parallel computing friendly. These comments reveal the dilemma of numerical computing on the decision of methods to be used. Interestingly, the direction of studies in the literature is to combine them efficiently. So, implementation of NK in such a way that stationary solver can also be utilized is a challenge. Before asking why's, how's should be answered.

In Newton-Krylov Techniques, during the computation of Δx, the solver frequently needs the result of the product Jv, v being any vector used in the Krylov solver. The resulting vector of this matrix-vector multiplication can be approximated with Fréchet derivative as in Equation 4.18. In this formulation, F is evaluated twice, in the first call of the function, x is perturbed by an amount of εv and in the second call x is used directly.

$$Jv \approx \frac{F(x + \varepsilon v) - F(x)}{\varepsilon} \tag{4.18}$$

Equation 4.18 is first order. High order formulations till sixth order are given by Turner and Walker (1992). Most important aspect of this formulation is the selection of the perturbation parameter, ε. Its selection is important for the robustness of the nonlinear solver. For well-scaled problems, $\sqrt{\eta}$ performs well η being the machine epsilon. On the other hand, this selection does not use any information on x or v. To date, many suggestions are made on the calculation of ε. One of the frequently used formula is the one presented by Brown and Saad (1990), which is an extension of definition of ε for a scalar computation by Dennis and Schnabel (1996)[9] . Several choices used in this study is presented in Table 4.2. In this table, *typ* is the typical size (Dennis and Schnabel, 1996), and $\eta = epsilon(0.d0)$ is the machine epsilon. Gropp *et al.* (2000) suggested to use $1e-6$ instead of $\sqrt{\eta}$ in compressible flow problems. According to this table, ε should always be calculated unless choice 3 of Qin *et al.* (2000) is used. Continuous

[9]There seem to be a conflict on the reference dates, however, the original print is published in 1983

Table 4.2. Selection of the perturbation parameter ε

choice	source	ε
1	Brown and Saad (1990)	$\frac{\sqrt{\eta}}{\|\|v\|\|_2} max\left(\|x^T v\|, typx^T v\right) sign\left(x^T v\right)$
2	Knoll (1998)	$\frac{1}{N\|\|v\|\|_2}\sum_{i=1}^{N}(a\|x_i\|+a),\ a=1e-6$
3	Qin *et al.* (2000)	$\frac{\sqrt{\eta}}{\|\|x\|\|_2}$
4	Gropp *et al.* (2000)	choice 1 with $\sqrt{\eta}=1e-6$
5	epsilon(0.d0)$^{1/2}$	$\sqrt{\eta}$
6	epsilon(0.0)	η_{sp}

calculation of this parameter is time consuming. In this thesis, calculated values of epsilon are monitored and it is observed that its value varies a lot. So selection of an average values seems to be irrelevant. On the other hand, choice 1 and 2 generally perform well compared to other choices when dealing with highly nonlinear problems. Hence, one should either sacrifice on some computational performance or keep the process robust. Since we need both, the selection procedure require more attention. For time dependent problems, like in AC problem even it is in pseudo time, it could be said that the variation of ε is minimal. Additionally, when both x and $F(x)$ is scaled, subsequently we have more control on the parameter. After then, use of an average ε seems logical.

One remark should be stated for linear problems. In theory, linear problems should converge in one Newton iteration since the Jacobian is linear and calculated vector, Δx updates to a converged x. However, in some cases, more than 1 iteration are needed (more than 2 is not rare). There are two reasons for that, first is the use of Inexact Newton. Linear system is not solved accurate enough to find x. This problem can be solved by providing Newton convergence tolerance directly to the linear solver (i.e. full Newton rather than inexact Newton). However, even in this case, Newton's Method might need a number of iteration for the convergence of the linear problem. This brings us to the second issue, eps is not calculated properly. Constant *eps* like in choices 5 and 6 might deteriorate the convergence. Choice 1 and 2 are reliable on

linear problems, as well. Of course, scaling of the unknowns as well as the nonlinear function helps to reduce the iterations, however, use of an adaptive *eps* is vital for a robust solver.

Going back to Equation 4.18, $F(x)$ is actually is calculated once in the Krylov solver. Since its value is not changing, it can be calculated and stored at the beginning of the linear iterations. So it is fair to say that, for first order formulations only one extra evaluation of F is needed per matrix-vector product. If the formulation is second order, then two computations are needed. To generalize, as much of evaluation is needed as the order of the directional differencing. Turner and Walker (1992) performed some computational experiments with GMRES are performed showing that high order approximation used at the beginning of the solver and low order approximations at the orthogonalization procedure following with high order subspace correction yields nearly the same result as of high order formulation used everywhere. This is especially important since most of the computations are done during the orthogonalization process. Also, low order approximation can be carried out single precision rather than double precision. Overall, computational effort is reduced both function calls and storage. Same idea is also proposed by Pernice and Walker (1998) for the NITSOL solver.

In Newton-Krylov framework, the linear system is solved till the convergence tolerance is reached. In that sense, update Δx is not found exactly as compared to a direct solver. Hence, this approach can be treated as an inexact version of Newton's Method. As a matter of fact Inexact Newton does have a generally meaning as explained in the next section. A review on Jacobian-free techniques is given by Knoll and Keyes (2004) where valuable information is given in every aspect of NK computation, like globalization techniques or preconditioning.

Before closing this section, the Element-by-Element (EBE) method used in Finite Element Analysis should also be stated. In EBE approach one can avoid forming the Global Stiffness Matrix and keep only the local matrices. This is especially useful if Krylov methods are used to solve the unknowns. Matrix-Vector multiplications can be computed with local contributions only. This idea is not matrix-free computation in

reality. It is just a natural way of storing the values with low cost. What is apparent in EBE is that calculation of the local stiffness matrices can be performed on parallel computers for decomposed domains. Pioneering work on EBE is carried out by Winget and Hughes (1985). Preconditioning issues are discussed in section 7.1

4.2.3. Inexact Newton

Solution of linear system comes to a stop as the norm of the residual is less than the given tolerance. However, selection of the tolerance is not obvious. It is a fact that, for initial guesses not so close to the exact solution, the calculated updates are rough estimates. So, do we really need to calculate each update as accurate as any other updates? When a strict tolerance is followed for each step, then oversolving will happen. Oversolving is defined as the solution of the linear step beyond the necessary amount of accuracy. That means, an update on iterate x^k will not be much affected by the extra accuracy involved in Δx. On the other hand, than the updates should be accurate enough to reflect the small changes on the solution vector, as the problem is about to converge. To sum up, it could be said that a mechanism should be devised to adjust the convergence tolerances of the linear equations. This idea is called Inexact Newton Method proposed by Dembo *et al.* (1982). Stopping criteria is defined with the inequality given in 4.19 where η is called the forcing parameter. For a converging method, $\eta < 1$ should be provided. It is a useful idea to decrease η as the nonlinear iterations proceeds which yields superlinear convergence (Brown and Saad, 1990) but quadratic convergence rate can only be achieved with appropriate selection of the forcing parameter.

$$||J(x^k)\Delta x + F(x^k)|| \leq \eta_k ||F(x^k)|| \tag{4.19}$$

Eisenstat and Walker (1996) gave two different choices to select η. Choice 1 is given in Equation 4.20 and choice 2 in Equation 4.21 with $0 \leq \gamma \leq 1$ and $1 < \alpha \leq 2$.

$$\eta_k = \frac{\left|\, \left\|F(x^k)\right\| - \left\|F(x^{k-1}) + J(x^{k-1})\Delta x^{k-1}\right\| \,\right|}{\|F(x^{k-1})\|} \tag{4.20}$$

$$\eta_k = \gamma \left(\frac{\| F(x^k) \|}{\| F(x^{k-1}) \|} \right)^{\alpha} \tag{4.21}$$

In order to prevent η being too small when the iterate is not close to the solution, safeguarding is suggested for both choices. Calculated η value, either with Equation 4.20 or 4.21, is modified when convergence rates are greater than 0.1. At the beginning of the Newton iterations, the convergence is slow but safe and towards the end, quadratic convergence is recovered as observed by Shadid *et al.* (1997). It is also suggested that η should not exceed η_{max} so after safeguarding, η_k is set as $min(\eta_k, \eta_{max})$ (Shadid *et al.*, 1997).

$$Choice\,1 : \eta_k = max\left(\eta_k, \eta_{k-1}^{(1+\sqrt{5})/2}\right) \quad \texttt{If } \eta_{k-1}^{(1+\sqrt{5})/2} > 0.1$$

$$Choice\,2 : \eta_k = max\left(\eta_k, \gamma\eta_{k-1}^{\alpha}\right) \quad \texttt{If } \gamma\eta_{k-1}^{\alpha} > 0.1$$

4.2.4. Global Convergence

Global convergence is ability to find a solution even if the initial guess is far away the exact solution vector. It is already mentioned that Newton's locally convergent with a good guess, otherwise it fail to find an acceptable solution: either the solution diverges or a non-physical solution is found which is probably a local minimum but not a global one. To get rid of these circumstances, update of the iterates need a mechanism. In the simplest case, the updates can be damped like in Equation 4.22. λ is the damping parameter which is $0 < \lambda \leq 1$. For the time being assume that Equation 4.16 is solved exactly. In this case, full step, Δx can place the new iterate further away the solution. Highly nonlinear problems fall into this category if no educated guess is used. Decreasing the step by half, might help the solver to run smoothly even a mild increase on $\| F \|$ could be allowed for the first couple of steps. Unfortunately, this convergent sequence of updates will reduce the convergence rate. Hence, selection of λ should be automated such that at the initiation of the computations, a relatively small λ should be used and allowed to increase with increasing iteration numbers. As a rule of thumb, one should always test the full step and if the test does not fail, then

$\lambda = 1$ should be used. The question is what kind of test(s) should be applied. Since a decrease in $\| F \|$ is expected, checking whether the norm is increased or not seem to be a plausible tactic. On the other hand, experienced showed that this criteria is too strict to be fulfilled. Additionally, a decreasing norm does not necessarily mean that the problem will converge. It might step in a local minimum and stay there (Dennis and Schnabel, 1996).

$$x^{k+1} = x^k + \lambda \Delta x \tag{4.22}$$

Instead, Goldstein-Armijo conditions (also known as $\alpha - \beta$ conditions) given in Equation 4.23 should be met as explained by Dennis and Schnabel (1996) which translate as average rate of decrease of $\| F \|$ is enough to continue with the iterations. f is a local quadratic model defined as $\frac{1}{2}F(x)^T F(x)$. In fact it is equal to the half of $\| F \|$ where $\frac{1}{2}$ is kept in the formulations for algebraic convenience.

$$f(x + \lambda \Delta x) \leq f(x) + \alpha_b \lambda \nabla f(x)^T \Delta x \tag{4.23a}$$

$$f(x + \lambda \Delta x) \geq f(x) + \beta_b \lambda \nabla f(x)^T \Delta x \tag{4.23b}$$

Brown and Saad (1990) suggested to increase λ two fold if β-condition is not met, however, in most of the problems α-condition is hard to satisfy. Value of λ is determined by a line search backtracking algorithm presented by Dennis and Schnabel (1996). Defining $\tilde{f}(\lambda) \triangleq f(x + \lambda \Delta x)$, a minimum λ could be find using $\tilde{f}(0), \tilde{f}(1), \tilde{f}'(0)$ on a parabolic model. Minimum of the parabola is at the point given in Equation 4.24

$$\lambda = -\frac{\tilde{f}'(0)}{2\left(\tilde{f}(1) - \tilde{f}(0) - \tilde{f}'(0)\right)} \tag{4.24}$$

If this point satisfies 4.23a then $\lambda \Delta x$ can be accepted. If the condition is not met, then another quadratic fit can be calculated with $\tilde{f}(0), \tilde{f}(\lambda), \tilde{f}'(0)$. Dennis and Schnabel (1996) suggested to use a cubic fit instead since four distinct conditions are already calculated: $\tilde{f}(0), \tilde{f}(1), \tilde{f}(\lambda), \tilde{f}'(0)$. What is known is that $\tilde{f}'(0) = \nabla f(x)^T \Delta x$. Since computations are based on F vector, the gradient of f should be replaced by a equiv-

alent form given in the Equation 4.25 (Brown and Saad, 1990). Test of α-condition is based on the transpose of ∇f so J^T is never used. The form of line search suggested by Brown and Saad (1990) is valid if Δx is solved exactly. Since Inexact Newton method is used together with Krylov solver, there is a residual at the end of the calculations. Hence, backtracking should be modified as explained by Brown and Saad (1990) such that the residual, r of the linear solver is also considered. Consequently, final form of $\tilde{f}'(0)$ is set as Equation 4.26.

$$\nabla f(x) = J(x)^T F(x) \tag{4.25}$$

$$\tilde{f}'(0) = -F(x)^T F(x) - F(x)^T r \tag{4.26}$$

Line search backtracking procedures with Inexact Newton Method is also considered by Eisenstat and Walker (1994), Shadid *et al.* (1997), Gropp *et al.* (2000), Pawlowski *et al.* (2006). Another globalization idea is the use of Trust Region methods (Brown and Saad, 1990). In this study, only line search backtracking is followed rather the trust region methods as a damping strategy mainly because the computational performance of trust region methods are not convincing and inferior compared to line search (Brown and Saad, 1990),(Pawlowski *et al.*, 2006). Also, search direction of trust region methods are affected when F is scaled whereas line search is unaffected.

There are some points to discuss here. If we allow mild increase in the F norm (with just using quadratic backtrack) then the solution is found in less steps. Forcing the residual being non-increasing is a strict rule and it can be relaxed this way. On the other hand, use of pseudo time and continuation, backtracks are performed only couple of times and Newton proceeds with full step (i.e. lambda=1). Still, cubic backtrack is applied to be general since with increasing nonlinearity, rise of the norm of the nonlinear residual cannot be prevented.

4.3. Nonlinear Solvers Chart

Figure on page 74 summarizes a class of solver to cope with nonlinear problem. In this chart, the relations between different solvers are also apparent. For linear solvers, stationary methods are accelerated by Krylov solvers or Subspace methods are preconditioned by simple iterative methods. Furthermore, Newton's Method can be preconditioned with Multigrid or FAS can utilize Newton's Method as a linear solver. There, interior iteration can also be preconditioned with all prescribed methods. One can also start with FAS if a good initial guess is not available and then might switch to Newton's Method for faster convergence. Different kinds of hybrid solvers can be used considering the solvers presented. Boxes with dashed lines describe the ideas that could be tested in the future. In the next chapters, Multigrid method and Domain decomposition will be analyzed in detail. Later, the discussion will continue with Advanced Techniques.

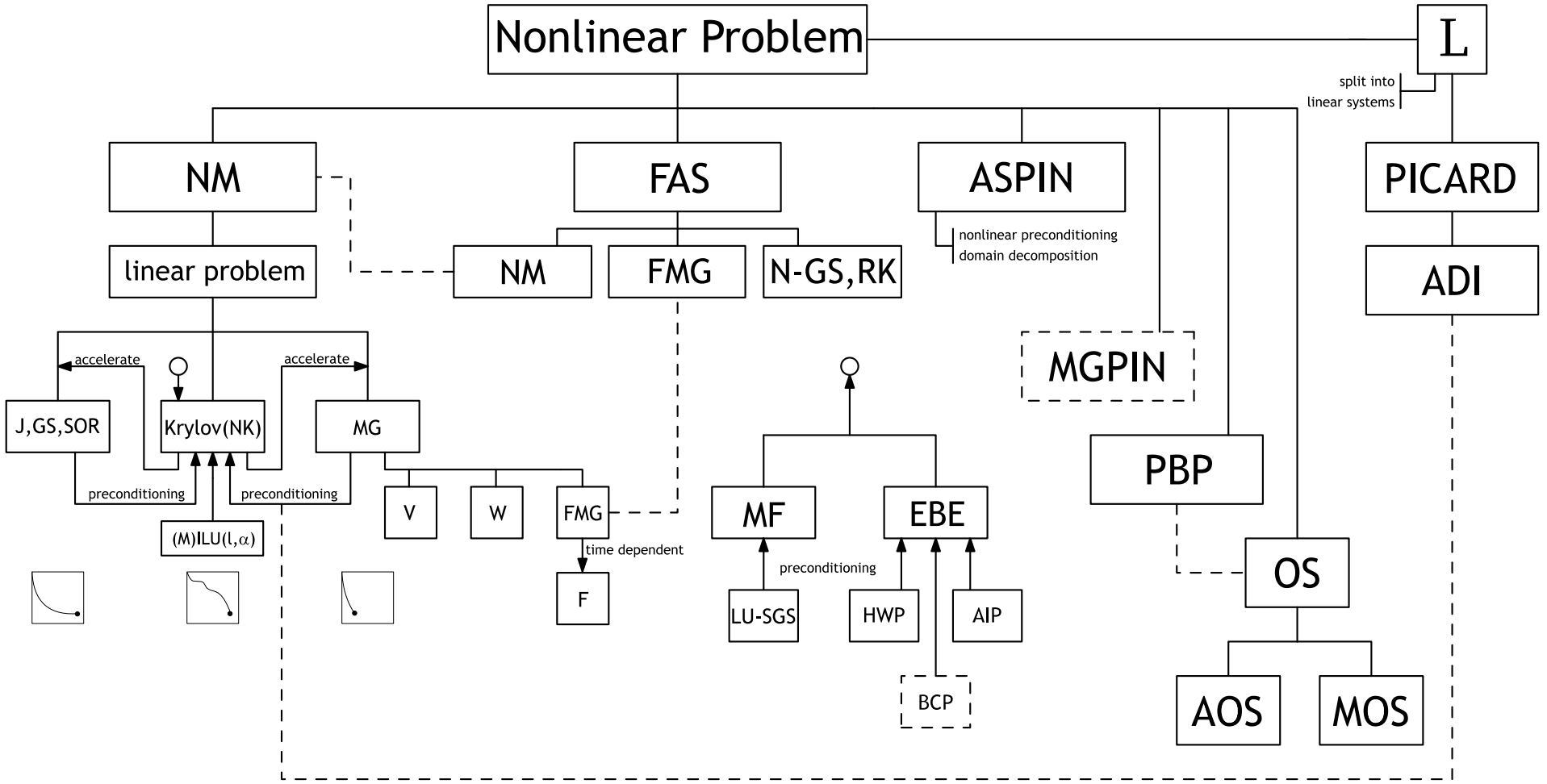


Figure 4.3. Nonlinear Solvers

5. MULTIGRID

Some iterative methods like Jacobi or Gauss-Seidel suffer from the so called smooth error which designates the error components that cannot be eliminated within reasonable[10] number of iterations. This issue is related to the error modes; high frequency errors are eliminated with success after a couple iterations, however, low frequency ones still keep prevailing hence keyword *smoother*[11] is used for those kind of methods. At this point, Multigrid comes into the picture as an idea to improve the convergence of smoothers in such a way that the low frequency error modes are also treated effectively. To explain the idea, it might be useful to use a linear problem in matrix form. Following the formulation of Briggs *et al.* (2000), a basic problem can be defined as in Equation 5.1.

$$Au = f \tag{5.1}$$

A is the coefficient matrix, u is the vector of unknowns. For completeness, a forcing function f is also kept on the right hand side. When an iterative method is applied on the system, the final iterate, which is an approximation for u, is defined as v. The error can be defined as $e = u - v$. Here, the residual equation should be given as in Equation 5.2.

$$r = f - Av \tag{5.2}$$

When f is replaced with Equation 5.1 and the definition of error is applied on the resulting system, we get an equation for the error:

$$Ae = r \tag{5.3}$$

[10]The definition is vague here because of the features of the discrete problem; it depends on the discretization, degrees of freedom, numerical parameters and convergence criteria.

[11]On the contrary, methods like CG or GMRES are called *roughers* in Multigrid context (Douglas and Douglas, 1993)

Equation 5.3 is actually equivalent to Equation 5.1: If the residual, r is calculated at the beginning with initial guess $u^0 = v^0$, one can iterate on Equation 5.3 to get an approximation for error, e, and then u can be updated as $u = u^0 + e$. The final result will be the same as if Equation is 5.1 relaxed as much as Equation 5.3. That means, error correction can be used to improve the approximate solutions. Unfortunately, Equation 5.3 is as expensive as 5.1 when kept at the same solution level. However, if the error is smooth then it can also be represented on a coarser grid. In Figure 5.1, low and high frequency errors are compared on fine and coarse grids. On the left of the figure, one can observe that the form of the error on fine grid is nearly conserved on coarse grid. On the other hand, representation of a high frequency error is completely different on the coarse grid as seen on the right of Figure 5.1. We can conclude that, if the error is smooth Equation 5.3 might be relaxed on a coarse grid such that the computational load is decreased. This is the essential idea of Multigrid. The two-grid

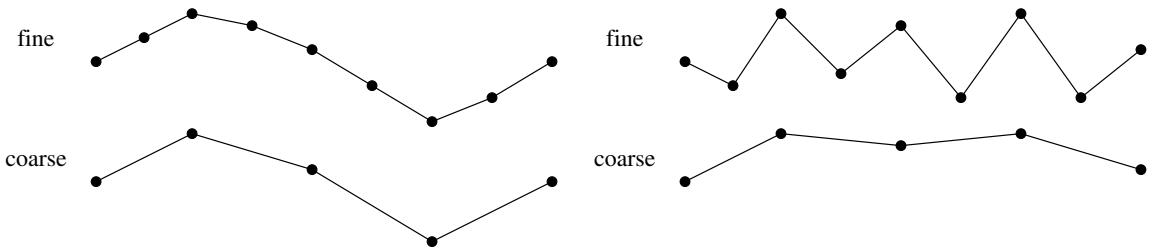


Figure 5.1. Errors on fine and coarse grids: low frequency on the left and high frequency on the right

cycle, also called $V2$, proceeds as follows. ν_1 number of iterations are performed for u^h. Then the residual on fine grid, r^h is found with Equation 5.2. After that, coarse grid residual, $r^{2h} = \mathcal{I}_h^{2h} r^h$ is found with the restriction operation $\mathcal{I}_h^{2h}$ as explained in section 5.1. Later, Equation 5.4 is used to solve the error on the coarse grid.

$$A^{2h} e^{2h} = r^{2h} \tag{5.4}$$

What is now left is to carry the coarse grid solution, e^{2h} into fine grid as e^h and update $u^h = v^h + \mathcal{I}_{2h}^h e^{2h}$ where $\mathcal{I}_{2h}^h$ is the prolongation operator section . This completes two grid cycle but there is more to say. After the update of the fine grid solution, it is customary to perform ν_2 number of iterations. In Multigrid terminology, ν_1 is named

as presmoothing and ν_2 as postsmoothing. After postsmoothing the same cycle can be repeated over and over until the desired tolerance criteria is fulfilled. While performing Multigrid, the iterations are based on pre- and postsmoothing iterations rather than checking the smoothness of the solution. Of course, performance of the procedure is heavily dependent of the smoothness of the problem and there are different ways to understand whether it is proper to move on a coarser grid (Hackbusch, 1985), however, one can still rely on past experiences to set ν_1 and ν_2 properly.

One might ask whether the coarse grid solution suffers from smooth error, too. The answer is yes. Although it occurs in later iterations compared to fine grid, smooth error also reduces the performance of the coarse grid solution. To correct that, one can use a coarser grid as well. This is essential idea of Multigrid. Each coarse grid is used to correct a fine grid above it so each smooth error component can be eliminated in respective grid level. As a result, $V2$ can be generalized as $V3, V4, ..., Vn$, n being the coarsest level. If level n is too coarse, then the system can be solved with direct methods like Gaussian Elimination so there will be no error in the solution while moving back to the finer grid. Visits to coarse grids can be in different orders. One form is already introduced as *V cycle*. Other form is the *W-cycle* as in Figure 5.2.

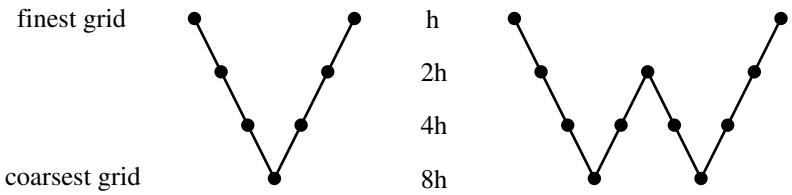


Figure 5.2. V and W cycles

5.1. Intergrid Operations

Values from fine and coarse grids should be transferred in multigrid. Use of proper operations is essential for multigrid to function correctly. As stated previously, a smooth result on a fine grid should also look smooth in a coarse grid so that the representation of the values are correct. When data transfer from a coarse grid to fine grid is needed, the operation is called *prolongation*. Values for fine grid are generated

with interpolations of coarse grid data. On the other hand, carrying values from a fine grid to a coarse one is called *restriction*. Mainly, two methods are available to be used in restriction. First one is called injection and in this approach only the values that is collocated on both grids are carried directly. Other method is full-weighting and the values that are discarded in injection are also shared on coarse grid point according to their weight. There is a rule for the order of the transfer operators and for second order partial differential equations, bilinear interpolation should be used along with full-weighting (Chen, 2005).

5.2. Full Multigrid

It is natural to ask how to improve the initial guess for the discrete problem. As seen previously, coarse grids are frequently used in multigrid. Same grids can also be used to enhance the starting vector of the smoothers. This idea is called Full-Multigrid - FMG. Computations start with coarsest grid than one can reach in a given V-cycle of the finest grid. As the first grid is solved, then the solution is interpolated into a finer grid and then a V2 cycle is realized by considering the same old coarse grid but this time to calculate the error correction. Interpolations to finer grids and coarse grid corrections with V cycle's continues till the finest grid of the problem. FMG shortens the calculation of the result with incorporating both multigrid and improved initial guesses. Another advantage of FMG is that an approximation for the error can be found although the exact error is unknown. To achieve this, solutions at different levels are compared for shared nodes. Hence, in FMG one can decide whether that particular solution level is good enough to declare mesh convergence. Use of coarse grids to have better initial guesses is a good idea, however, there is the problem of resolving the physics of the problem. If the starting grid is too coarse, then the approximations fail to provide good initial guesses for finer grids and multigrid performance degrades. That's why, depending on the problem, there are limits on the selection of the coarsest grid. The so called short-FMG is used to incorporate better solutions by avoiding extremely coarse grids (Drikakis and Rider, 2005). This issue is usually encountered in convection dominated flow problems. There is really no remedy to speed up the convergence of the coarsest grid at the start of FMG, when a smoother

used but use NK just at the beginning is a good practice. FMG can also be applied

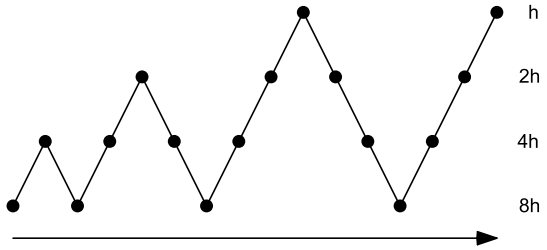


Figure 5.3. Full Multigrid

on time dependent problems. For transient cases, *F-cycle* is used in multigrid. In this cycle, continous reduction is made on the number of grids. Refering to Figure 5.3, all of the values are restricted from level h to level $8h$. Thereafter, computations on new time step can proceed with another FMG cycle. Some notes on transient multigrid is also given in section 5.4.

5.3. Full Approximation Scheme

One of the powerful aspects of multigrid is that the ideas can be extended for nonlinear problems. Full Approximation Scheme (FAS) proposed by Brandt (1977) which is later generalized by Hackbusch (1985) is an efficient nonlinear solver especially when combined with FMG. In different words, FAS-FMG present an appealing alternative to Newton based solution algoritms. In FAS, coarse grids are still in use to improve the solutions on the finer grids, however, in contrast to the linear case, instead of solving the error directly (which cannot be done in nonlinear systems[12]) coarse grid unknowns are solved with a slight modification on the system. Details are as follows.

Define a nonlinear problem as $A(u) = f$. $A(u)$ is a function of u and not necessarily linear. f is kept on the right hand side even if its value is zero. Remember that in Newton's Method, everything on RHS is carried to left side as the definition of the problem. FAS on the finer level is similar to $F(x) = 0$ case, however, the procedure requires nonzero forcing functions on coarser levels. Computations start with smoothing

[12]There are, however, some exceptions; discretization and smoother may give an explicit definition of the error. See the examples in Barrett *et al.* (1994) Chpt. 6

the fine grid solution u (Briggs *et al.*, 2000). After ν_1 amount of presmoothing steps, approximation v^h is found. Next step is to calculate the coarse grid residual, r^{2h} as in Equation 5.5.

$$r^{2h} = \mathcal{I}_h^{2h}\left(f^h - A^h(v^h)\right) \tag{5.5}$$

Later, v^h is restricted and the coarse grid problem is formed like in Equation 5.6

$$A^{2h}(u^{2h}) = A^{2h}(v^{2h}) + r^{2h} \tag{5.6}$$

After the computation of the coarse grid solution u^{2h}, error on the coarse grid can be calculated as $e^{2h} = u^{2h} - v^{2h}$. Prolongation of the error and update of the fine grid solution reads as

$$v^h = v^h + \mathcal{I}_{2h}^h e^{2h} \tag{5.7}$$

This two grid cycle defines basic FAS operation. There are several points to be explained. First of all, it is not stated what smoothing in nonlinear setting means. As the solver, Newton's Method can be used, but since a smooth solution is needed rather than a rough solution, the extension of stationary solvers on to nonlinear problems could be used. Pointwise Nonlinear Gauss-Seidel (NGS) is the default solver of FAS. In NGS, a local model is construted and solved with a scalar Newton's method. The central variable is treated as the main unknown and for this unknown local f and df are calculated. Generally, one newton step suffices for the computations but for highly nonlinear case, number of steps can be increased with a cost of more computational load. Damping can also be used to provide a steady convergence. Beside NGS, different solvers can also be used like ADI, Runge-Kutta, Strongly Implicit Procedure (Ghia *et al.*, 1982) and many others. The so called box-smoothing is generally prefered on the solution of fluid dynamics problems on staggered grids (Trottenberg *et al.*, 2000). In box-smoothing, every variable on a given cell are solved at the same time with the inversion of a small jacobian (for 2D problems, 5×5 Jacobian is neeeded.)

As a second observation, selection of the coarse grid solution is not obvious. On the procedure above, the coarse grid vector v^{2h} is found by simply restricting v^h. If only a V or W cycle is used, this choice is reasonable. On the other hand, when FMG is used, then solution on previous coarse grid levels can be used directly without restricting the fine solution. In some cases, coarse grid problem should be modified physically since the definition of the fine grid problem might be ill-posed on a coarse level (Brandt, 1980),(Vanka, 1986).

Third remark is on the nonlinear problem $A(u)$. In Equation 5.6, residual on the right hand side is updated with $A^{2h}(v^{2h})$. Obviously, the solver should differentiate A with no forcing function, f with A that includes the forcing function. f is needed to perform smoothing but not needed to update the coarse grid residual. Consequently, the updates on the rhs can be defined systematically with the introduction of the τ-correction (Briggs *et al.*, 2000). In τ-correction, Equation 5.6 is replaced with an equivalent form as in 5.8 where τ_h^{2h} is defined in Equation 5.9. If τ is zero, than original equation on the coarse grid is recovered. On the other hand, that will mean that the fine grid is solved. Generally, τ is nonzero so a correction is calculated with the designated algoritm. Extension of FAS to more grid levels is similar to the linear multigrid.

$$A^{2h}(u^{2h}) = f^{2h} + \tau_h^{2h} \tag{5.8}$$

$$\tau_h^{2h} = A^{2h}\left(\mathcal{I}_h^{2h} v^h\right) - \mathcal{I}_h^{2h} A^h(v^h) \tag{5.9}$$

5.4. Parabolic Multigrid

Multigrid can be applied on transient problems as well. However, parallelization is not obvious for computations. It is natural that unsteady problems are sequential (parareal algorithm of Lions *et al.* (2001) promises time parallelization, though) Hackbusch (1984) proposed a method based on multigrid, which he called parabolic

multigrid, at which several time steps can be calculated simultaneously. This method can be seen as decomposition in time for implicit formulations (Ferziger and Peric, 2002). Horton (1992) applied the idea on Navier-Stokes Equations using SIMPLE.

5.5. Algebraic Multigrid

Previous sections on Multigrid is based on geometric definitions of fine and coarse grids - that's why it is also called Geometric Multigrid (GM). On the other hand, there are bunch of problems with unstructured grids. For that kind of problems, three ingredients of Multigrid should be redefined: prolongation operator, restriction operator and coarse grid selection. Normally, the problem is always the definition of the coarse grid equation. In GM, there is a hierarchy of grids but for unstructured grids, such a basis is not easy to construct.

Algebraic Multigrid (AMG) is an attempt so get rid of these difficulties. In AMG, instead of a geometric smoothing, algebraic smoothing is defined. This definition relies purely on mathematical formulations. Restriction and prolongation can be kept in abstract level by using variational forms like Galerkin condition even without a coarse grid. Much effort on AMG is on the selection of coarser level. The so called agglomeration method attract a lot of attention in multigrid community. In Agglomeration multigrid, a series of operations are performed to join the grids. Group of volumes are also regrouped on coarser levels. In this thesis, AMG is not used. More information can be found in books of Briggs *et al.* (2000) and Trottenberg *et al.* (2000).

6. DOMAIN DECOMPOSITION

Domain decomposition techniques have different definitions according to their use (Smith *et al.*, 1996). As a preconditioner, it divides the global domain into small pieces. It defines the relationships between other sub-domains and as well as the global domain. Another definition of domain decomposition can be given for the perturbation of the PDEs. Asymptotic methods can be used to solve the problem in outer and inner domains separately which are later set to be match at their boundaries. In aerodynamics for instance, the computational domain around an airfoil can be divided into two. The narrow region in the vicinity of the airfoil can be used to solve the full NS Equation whereas, out of this region Euler Equations can be used. As a result Boundary Layer Theory can be applied with domain decomposition. Also it can be used in parallel computation to distribute the information on processor network. This distribution is not solver dependent; it just states the way in which the data must be handled.

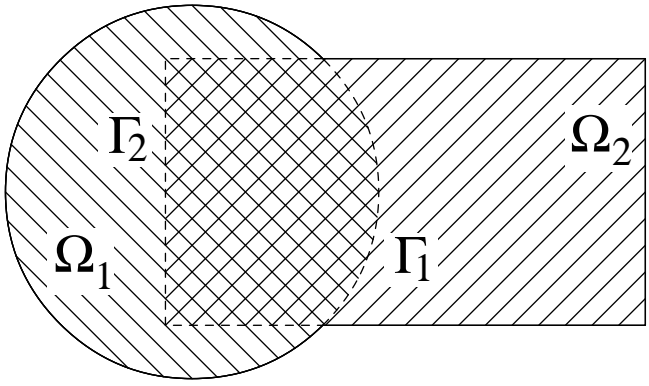


Figure 6.1. Schwarz's idea

In engineering applications, domain decomposition methods are first use to divide large problems into small sub problems so they can be studied without exceeding limits of the computer hardware. Early studies on decomposition is of Kron (1953) and Przemieniecki (1963). However, the form of the domain decomposition defined today is first investigated by Schwarz (1869). Schwarz analyzed the Poisson's Equation on

an irregular domain to prove the existence of a solution and used a domain similar to Figure 6.1. While doing that, he divided the domain into two simple geometries - a circle, Ω_1, and a rectangle, Ω_2. When interface conditions are defined on the overlap of the geometries then solution on both sub-domains can be iterated until a final solution is found. For simplicity, nonhomogenous Dirichlet BC's are used on artificial boundary conditions, Γ_1 and Γ_2. As observed form the figure, artificial boundaries are in the interiors of other domains. Consequently, the values of the articial boundaries are calculated via interpolating the values of nearest nodes of particular domains. Convergence can be claimed is the results of two consecutive visits are similar or the values on the overlap do no change much. Formerly, this idea is called Alternating Schwarz Method and it is regarded as the method to be used on nonmatching domains. Non-matching (also called non-conforming) means that the nodes of the domains do not coincide. In both domains, different discretization ideas can also be used; Smith *et al.* (1996) utilized FD and FE in a simple test problem where FD is used on a structured domain and FE is used to analyze a circular domain with triangular elements. In DEMONA, that means two physics are used. As stated before, different equations are contained within different physics.

The idea of Schwarz is improved in such a way that the resulting system acts as successful preconditioner. Two variants are suggested: Additive and Multiplicative Schwarz Methods (Smith *et al.*, 1996). In multiplicative version, calculated solution on a domain are directly used on another domain. In additive version, however, data exchange is performed at the end of all computations. This procedure reminds us the relation between Gauss-Seidel and Jacobi where an analogy becomes apparent. Unfortunately, additive variant is not a standalone solver. It works only, if used as a preconditioner to a Krylov solver. Efstathiou and Gander (2003) demonstrated with a simple 1D example why additive Schwarz is not converging. It should be also noted that the Restrictive version of AS, RAS has favorable convergence properties on AS (Cai and Sarkis, 1999). In Figure 6.2, an example for conforming grids is given. In this model, instead solving Ω directly, one can divide the problem into four conforming and overlapping domains. With current selection of domains total number of unknowns of each subproblem is less compared to Ω. In multiplicative version, when the computa-

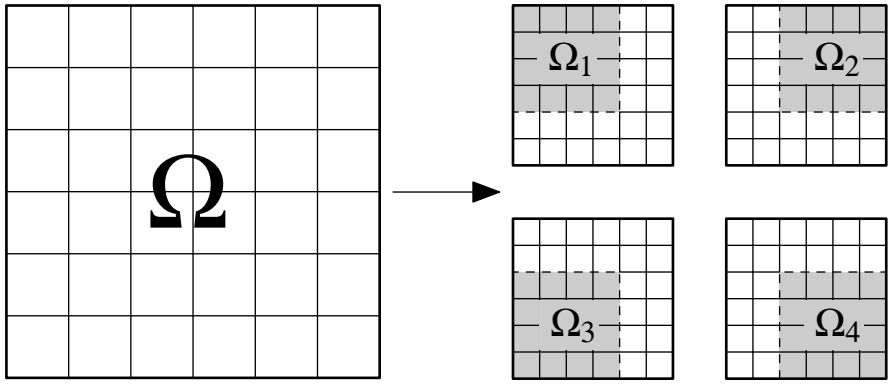


Figure 6.2. Matching grids

tions starts with say, Ω_1 then some of the artificial bc values of Ω_2 will utilize the recent solution of Ω_1. Later, Ω_3 will make use of the solution of previous two solves. Finally, Ω_4 will exploit all previous solution. In additive version, however, recent values will not be used and data transfer will occur only when all domains are visited once.

This form of domain decomposition is a kind of a solver. Each subdomain is a distinct problem by its own so any technique introduced before can be employed to find the local results. As a preconditioner, on the other hand, use of subdomains differ. Suppose there are two conforming overlapping domains. If $Ax = b$ is formed in such a way that two sub domains are embedded into a single matrix considering the artificial boundary conditions, then this matrix will be different than the one would be constructed with the global domain. Apparently, some of the unknowns of Ω do exist in both subdomains. Here x is created with combining both x_{Ω_1} and x_{Ω_2}. Then for x_Ω it can be said that $x_\Omega \subseteq x$. Now, define the restriction $A_{\Omega_i} = R_i A R_i^T$ such that the part of A that is related to sub domain Ω_i is extracted. Then multiplicative Schwarz method can be written as in Equation 6.1 (Smith *et al.*, 1996). In the first pass, first domain is solved. In the second pass, second domain is solved and the updated values on the overlap are directly used. This procedure can be defined in one pass if B_i is defined as $B_i = R_i^T \left(R_i A R_i^T\right)^{-1} R_i$. Then it reads as in Equation 6.2. If the update is to be performed after each sub domain is solved, then additive Schwarz method is applied. Equation 6.3 gives the single step iteration of this variant. The multipliers, i.e. $(B_1 + B_2 - B_2 A B_1)$ in Equation 6.2 and $(B_1 + B_2)$ in Equation 6.3 can be treated as preconditioner on the residual $b - Ax$. Then, in the solution of the Ω problem,

the combinations of B matrices can be used to improve the convergence of the Krylov solvers.

$$x^{k+1/2} = x^k + R_1^T \left(R_1 A R_1^T\right)^{-1} R_1 \left(b - Ax^k\right) \tag{6.1a}$$

$$x^{k+1} = x^{k+1/2} + R_2^T \left(R_2 A R_2^T\right)^{-1} R_2 \left(b - Ax^{k+1/2}\right) \tag{6.1b}$$

$$x^{k+1} = x^k + \left(B_1 + B_2 - B_2 A B_1\right)\left(b - Ax^k\right) \tag{6.2}$$

$$x^{k+1} = x^k + \left(B_1 + B_2\right)\left(b - Ax^k\right) \tag{6.3}$$

Beside this one level formulation, methods for multilevel are also available (Smith *et al.*, 1996). Theory of domain decomposition as well as applications can be found on reference books Quarteroni and Valli (1999), Wohlmuth (2001), Toselli and Widlund (2005), Mathew (2008).

Before proceeding with the next section, it is useful to denote that different BC's can be used to transfer data between domains. Till here, only artificial Dirichlet BC's are considered. On the other hand, boundary conditions can be generalized as in Equation 6.4. If α^d is 1, then Dirichlet BC's are used. Taking the value as 0 will yield Neumann BC's. Any other value in between will be a general Robin Type BC. In decomposed physics, appropriate interface conditions should also be applied to model the problem to pursue compatibility.

$$\alpha_1^d x_1^k + (1 - \alpha_1^d)\frac{\partial x_1^k}{\partial n} = \alpha_2^d x_2^{k-1} + (1 - \alpha_2^d)\frac{\partial x_2^{k-1}}{\partial n} \tag{6.4}$$

7. ADVANCED TECHNIQUES

Numerical methods introduced before work reasonably well for a wide range of problems. In should be noted that some form of acceleration is already introduced in preceding chapters, selection of the forcing parameter η can recover quadratic convergence of Newton's Method or Multigrid can be used to accelerate the solution of both the smoothers as well as Krylov solver as preconditioner. In this section, however, we will go one step further and explain advanced techniques to enhance the convergence. These ideas can be regarded as improvements on Newton's Method but can be applied to other approaches, as well.

7.1. Preconditioning

Preconditioning is an important element of numerical analysis. Krylov Subspace Solvers generally fail to convergence w/o the application of a proper preconditioner since the original systems fail to have well-posed spectral properties. In other words, some of the eigenvalues are so large which leads to no convergence or even to divergence. Basic form of a preconditioned equation is given in 7.1 in which the preconditioning matrix, M is an example of forward type preconditioners (Chen, 2005). Equation 7.1 is equivalent to Equation 4.1 in the sense that both have the same solution. For the ideal case, $M = A^{-1}$, however, this is not practical since calculation of the inverse of A means the solution of 4.1 is found already. Drawbacks of direct solution methods are addressed before in section 4.1 hence instead of using the inverse, we have to come up with some form which is cheap to calculate and easy to implement and also we have to pursue $M \approx A^{-1}$ as much as we can.

$$M^{-1}Ax = M^{-1}b \tag{7.1}$$

First idea is to use stationary methods like Jacobi or Gauss-Seidel Methods. They are simple to use yet their performance reduces for stiff problems. Another form

of preconditioners is the incomplete LU decompositions (Saad, 2003). In ILU(l), an LU decomposition is computed for A such that elements that does not belong to the pattern are eliminated. This approach is useful since the same storage are of A is enough. This basic form of ILU is called ILU(0) where no fill-ins are allowed (l=0). Fill-in's are extra entries stored in ILU. Number of fill-ins improves the iterations but more storage is requires which cannot be easily guess beforehand unless a symbolic evaluation is performed. Also, evaluation take more times as l increases. Another variant of ILU is Modified-ILU (MILU). In MILU, discarded elements are added to the diagonal such that row sums are conserved which is an important property for matrix-vector products. MILU can be generalized as MILU(α,l) $\alpha \in [0, 1]$ being the multiplier of the sum of the dropped elements. ILU-Threshold (ILUT) is a different version of ILU where some of the elements are dropped whether they belong to the set of A or not. Depending on the drop tolerance, some of the entries of the decomposition is kept. Central issue for ILUT is that the storage requirements cannot be estimated - it can change during the iterations. On the other hand, ILUT is a powerful preconditioner (Saad, 2003). Other version of ILU preconditioners are given by Benzi (2002), van der Vorst (2003), Saad (2003) and Chen (2005).

Prior to the investigation of other preconditioning options, implementation details of previously defined preconditioners will be given. Assume that A is decomposed as $A \approx M = LU$. Then, $M^{-1} = U^{-1}L^{-1}$. When this is substituted into Equation 7.1 new form is as in Equation 7.2. Even though there are inverses, they are not calculated. In the analysis, instead of 7.2, 7.3 is utilized. Let's say that in Krylov solver, A is to be multiplied with a vector v such that $w = Av$. If $U^{-1}L^{-1}w = z$ and $L^{-1}w = q$ then implementation is as in Equation 7.3.

$$U^{-1}L^{-1}Ax = U^{-1}L^{-1}b \tag{7.2}$$

$$Solve \quad Lq = w \tag{7.3a}$$

$$Solve \quad Uz = q \tag{7.3b}$$

In 7.3a and 7.3b there are actually no relaxation. Since L and U are lower and upper triangulars, respectively, forward substitution is used for 7.3a and backward substitution is performed on 7.3b. Normally, diagonals of both L and U should be stored, however, during the LU decomposition the diagonal of L is normalized so 1's are assumed to be on the main diagonal.

The form given in 7.1 left preconditioner. Left preconditioners change the residual vector, as well. That means, the norm of the residual is scaled wrt the preconditioner. This is an issue when testing for convergence. An alternative formulation is the right preconditioning where the residual is kept as is. A third possibility is use of split preconditioners where L and U are applied are pre and post multipliers on A. In GMRES, preconditioning is applied during Arnoldi Orthogonalization. If the preconditioner is left, initial residual vector is operated with the preconditioner. For right preconditioner, however, the preconditioner is applied on Equation 4.8. GMRES has a modified algorithm called Flexible GMRES (FGMRES) which allows to use different preconditioner at each step of the iterations (Saad, 1993).

Other type of preconditioners - the inverse type preconditioners read as in Equation 7.4. In these type preconditioners, the inverse of A is directly approximated. Approximate inverse preconditioners are more effective if there is reasonable error involved in the computation of the incomplete decomposition. For sparse matrices, the quest for a approximate inverse starts with Equation 7.5 considering a right preconditioner. Refering to Saad (2003), F is a function to be minimized on Frobenius norm. One should find an M that gives a small F value. Equation 7.5 can be rewritten as in 7.6. There are two possibilities to compute M, either a global calculation can be made for minimization or each component can be minimized by itself. Both cases are

explained in detail by Saad (2003) and Chen (2005).

$$MAx = Mb \tag{7.4}$$

$$\mathrm{F}\,(M) = ||I - AM||_F^2 \tag{7.5}$$

$$||I - AM||_F^2 = \sum_{i}^{N} ||e_i - Am_i||_2^2 \tag{7.6}$$

In Matrix Free computations, preconditioning can be described as in equations 7.7-7.9 where both left (P_L) and right (P_R) preconditioner are presented for convenience.

$$P_L^{-1} J(x)v = P_L^{-1} \frac{f(x + \varepsilon P_R^{-1} y) - f(x)}{\varepsilon} = z \tag{7.7}$$

$$P_R v = y \tag{7.8}$$

$$P_L z = Jv \tag{7.9}$$

Normally, P_L and P_R should be evaluated from J when applying for example a GS preconditioner or Incomplete LU decompositions (ILU). However, if J is not at hand, as in Newton-Krylov techniques, an approximation for the preconditioner should be found. First idea is to perform a numerical Jacobian calculation if an analytical form is not available where all entries are calculated. Unfortunately, this will increase the storage requirements as well as computational work. At this point, it might reasonable to use a coarse-grid approximation where the total number of unknowns is reduced

so less storage will be enough. Since stationary methods are good smoothers, use of coarse grids can be generalized as multigrid preconditioning (Knoll and Rider, 1999). Another option could use a low-order approximation of the problem. In that sense a high accuracy result can be preconditioned with a basis discrete form which requires less computation (Knoll, 1998). Still there will be a need to compute the Jacobian. Two issues should be resolved. How to calculate the Jacobian with vector function F and how to store the matrix. The latter can be handled with special storage schemes like Compresses Row Storage (CRS) or other techniques presented in Barrett *et al.* (1994) and Saad (2003). Once the Jacobian stored in sparse forms, ILU decomposition should also performed sparse. Former issue on the other hand requires more attention.

7.1.1. Calculation of the Jacobian

By default, the Jacobian is calculated with coloring scheme and only for preconditioning purposes. It is stored in CRS format. For structured grids, this is a relatively simple task but for unstructured grids or high order calculations, graph theory should be implemented to color the domain as in Gebremedhin *et al.* (2005). Automatic differentiation is another way of computing the Jacobian (Griewank and Walther, 2008). Lastly, complex differentiation can also be used which provides robust and accurate formulation as suggested by Martins *et al.* (2003). These last two ideas are not addressed in this study and left as future work.

Essential idea in this method is to color of each node. For 2D problems, a 5-point stencil requires $n = 5 \times unk$ number of colors, unk being number of unknowns per node. Every node on the domain is assigned to a color in such a way that all points that share a stencil have different colors. This is illustrated in Figure 7.1. After the appointment of colors, $F(x^c)$ is evaluated n+1 number of times. In each n evaluations, i'th colored unknowns of x are perturbed by an amount of ε to generate x^c's. "+1" evaluation is for $F(x^c = x)$. Now the entries of the jacobian can be approximated as in 7.10. Bottleneck in this formulation is checking of each color for each row. Then again, the operation is extremely fast even for high number of unknowns.

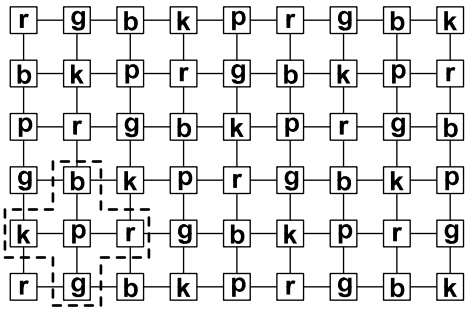


Figure 7.1. Coloring of nodes

$$J_{ij} \approx \frac{F_{c_o f_j}(i) - F(i)}{\epsilon} \tag{7.10}$$

7.1.2. Computational Tricks

There are several options in calculating the preconditioners. First of all, instead of calculating a preconditioner at each Newton Step, it can be calculated in every 2 or 3 steps depending on the problem. In some cases, a constant preconditioner calculated at the beginning might also work, however, for highly nonlinear problems this may result with breakdown of the solver. Also, as the solver is near at the solution, i.e. converge is imminent, then the jacobian can be fixed since x will not change much. Another idea is as prescribed before, use of low-order preconditioner for a high order problem. For example, an incompressible flow problem modeled with QUICK scheme will require a 9-point stencil. On the other hand, use of first order upwind can reduce the stencil into 5 point at which color based jacobian evaluation can be employed. For Navier-Stokes Equations the preconditioner can be selected from the diffusive terms only. Since diffusive terms are linear, they will not be needed to be recomputed. To increase the neighborhood of convergence, a combination of freezed Jacobian and diffusive jacobian can be utilized. The convective terms can be reevaluated at every 2-3 Newton steps.

There are more to say on preconditioners. Different types of preconditioners that are not described here can be found in the review paper of Benzi (2002) and on the book of Chen (2005). Still, some advanced forms of preconditioners can be described as in the following sections.

7.1.3. Physics-Based Preconditioning

Some preconditioning methods based on the system matrix or the geometry of the solution domain are discussed in preceding sections. Physics-Based Preconditioning brings a new aspect into the picture. It focuses on the governing equation of the problem. Mousseau *et al.* (2000) first applied this type of preconditioning on radiation diffusion problems. They utilized a different operator splitting technique on their coupled equations 7.11 and 7.12.

$$\frac{\partial E}{\partial t} - \frac{\partial}{\partial x}\left(D_r \frac{\partial E}{\partial x}\right) = \sigma_a(T^4 - E) \tag{7.11}$$

$$\frac{\partial T}{\partial t} - \frac{\partial}{\partial x}\left(D_r \frac{\partial T}{\partial x}\right) = -\sigma_a(T^4 - E) \tag{7.12}$$

In order to apply this preconditioning, first a method for solving the equations are discussed. By looking at equations 7.11 and 7.12 one can say that LHS of them are independent of each other. However the coupling between E and T is stated at the RHS. Suggested method first sets the RHS to zero and solves for intermediate values of E^* and T^*. Since they are decoupled now, the variables can be solved separately. Next step is to set the diffusive parts of LHS to zero and solve the new coupled system. In this step, T^4 term is linearized as $T_n^3 T_{n+1}$ where n is the previous step (not the intermediate step) and $n+1$ is the step that we look for its solution. After the 2×2 system is solved, then time level $n+1$ assumed to be reached. Here we can think the time steps as diffusion time level and radiation time level. The results are found with Newton-Krylov solvers. Other applications are also available in the literature Chacon *et al.* (2002); Reisner *et al.* (2003).

7.1.4. Multi-pole Based Preconditioning

The basic Idea behind Multi-pole is clustering far away point into one single point, which represents the properties of that particular region. It is first used for N-Body

problems, for the analysis of the attractions between electrical charges in a particular region for example. For N charged particle, there are $N!$ relations. If the number of the charges are not much, then the analysis can even done by hand. However, while N is getting larger and larger, the analysis is becoming cumbersome. So, instead of considering each particle, a bunch of particles can be studied together.

Main concern is how to cluster the points. Several approaches are suggested. Anderson, for example, suggested the use of a ring around the pole in question (Guillaume *et al.*, 2003). Guillaume *et al.* (2003) used multi-pole in Block Constant Matrix Preconditioners. This preconditioner has the property that the system matrix A is divided into submatrices. Each matrix is constant, which is determined by means of procedures promises residual minimization. The minimization takes place after the decision of the block constant matrix is made. That means, the pattern of the bcm will be arranged. The pattern is decided by means of three different parameter, L_c, L_i, L_f. L_c is the coarsest level of refinement. If it is equal to N, then the matrix is full, i.e. every sub-matrix is $1x1$. L_f on the other hand, gives the finest level of the refinement. A L_f value equal to 1 will give a full constant matrix. A major drawback in this method is that the matrix must be reordered in order to achieve an efficient algorithm. Guillaume *et al.* (2003) suggested different methods but used recursive bisection algorithm.

7.1.5. Nonlinear Preconditioning

Non-linear preconditioners are more complex approaches to attack the problem. Often, the formulation of the preconditioner must be decided prior solving the problem. The geometry and the equations affect the selection of the preconditioner. In nonlinear preconditioning, a system of nonlinear equation, $\bar{F}(\bar{x}) = 0$, is changed into $\bar{\Phi}(\bar{x}) = 0$ where both $\bar{F}$ and $\bar{\Phi}$ have the same solution. For a nonlinear system, a formulation like in Equation 7.13 can be derived.

$$F(x + \delta_i(x)) = 0 \tag{7.13}$$

Then we need to come up with a solution for $\delta_i(x)$ such that the residual becomes zero in region i (Equation 7.14)

$$F_i(x + \delta_i(x)) = 0 \tag{7.14}$$

These ideas can be incorporated in Additive Schwarz Preconditioned Inexact Newton (ASPIN) algorithm Cai and Keyes (2002). In this algorithm, first the nonlinear residual is calculated. To do that, first Equation 7.14 is solved for δ_i's (with zero initial guess). Then the global residual is found via Equation 7.15. Before calculating the update, the inexact Newton direction, p in Equation 7.15, is found via IN algorithm given in Cai and Keyes (2002). During the computation, the approximate Jacobian is found with the decomposition. The form of the Jacobian as well as the preconditioner will depend on the number of subdomains and their orientations.

$$\Phi(x) = \sum_i \delta_i(x) \tag{7.15}$$

$$\sum_i^N J_{S_i}^{-1} Jp = \Phi \tag{7.16}$$

7.1.6. Element-by-Element Preconditioners

First idea on EBE preconditioning is proposed by Winget and Hughes (1985) with the so called HW preconditioner where an early work of the same authors is expanded (Hughes *et al.*, 1983). The preconditioner is a variant of Approximate inverse preconditioners where the inverses are expanded with Neumann series. At the same year, Nouromid and Parlett (1985) investigated EBE preconditioning with splitting techniques. Law (1986) use EBE techniques to parallelize the computations. Later, Tezduyar and Liou (1989) introduced Group EBE (GEBE) preconditioners to be used on incompressible flow problems. Papadrakakis and Dracopoulos (1991) presented

partial preconditioning based on EBE computations. Since then, EBE techniques are frequently used in scientific computing (Sheu *et al.*, 2000).

7.2. Switched Evolution Relaxation

An important property of implicit Newton-Krylov techniques is that there is virtually no limit for the selection of the time step i.e. even infinity can be selected. However, since at the beginning of the computation, the initial guess is far way from the solution, Newton's method might converge to an unrealistic solution or even might diverge. So, it is a good idea to start with small time steps like explicit solution techniques. However, as iterations pass the time step can be increased to reach steady state faster since as we approach the steady state, the solution will not change much.

Switched Evolution Relaxation, SER[13] , technique can be used to accelerate time dependent problems by modifying the time steps (Mulder and van Leer, 1985). Formulation in Equation 7.17 is adapted from Blazek (2005), σ being the CFL number, n being the time level and R being the residual of respective time level. α_{SER} is added to diversify the selection of the time step where the proposed value is $\alpha_{SER} = 1$. However, Gropp *et al.* (2000) suggested that new value should not exceed 200% and not be less than 10% of the previous value and the time step should be modified accordingly. If $\alpha_{SER} = 0.5$, then clipping of the new value might be omitted since similar trends are observed. A sample computation is given in Figure 7.2 for stream function - vorticity formulation using Inexact Newton method with ilu(0) preconditioned GMRES(200) as the linear solver. Constant time steps are compared against $\alpha_{SER} = 0.5, 1.0$ w/o clipping. As observed from the figure, significant improvement is achieved with varying time steps. $\alpha_{SER} = 1.0$ w/o clipping case is the fastest option but experienced showed that for problem with higher nonlinearities, clipping should be performed.

$$\sigma^{n+1} = \sigma^n \left(\frac{||R^{n-1}||_2}{||R^n||_2} \right)^{\alpha_{SER}} \tag{7.17}$$

[13] also called successive evolution-relaxation (Gropp *et al.*, 2000)

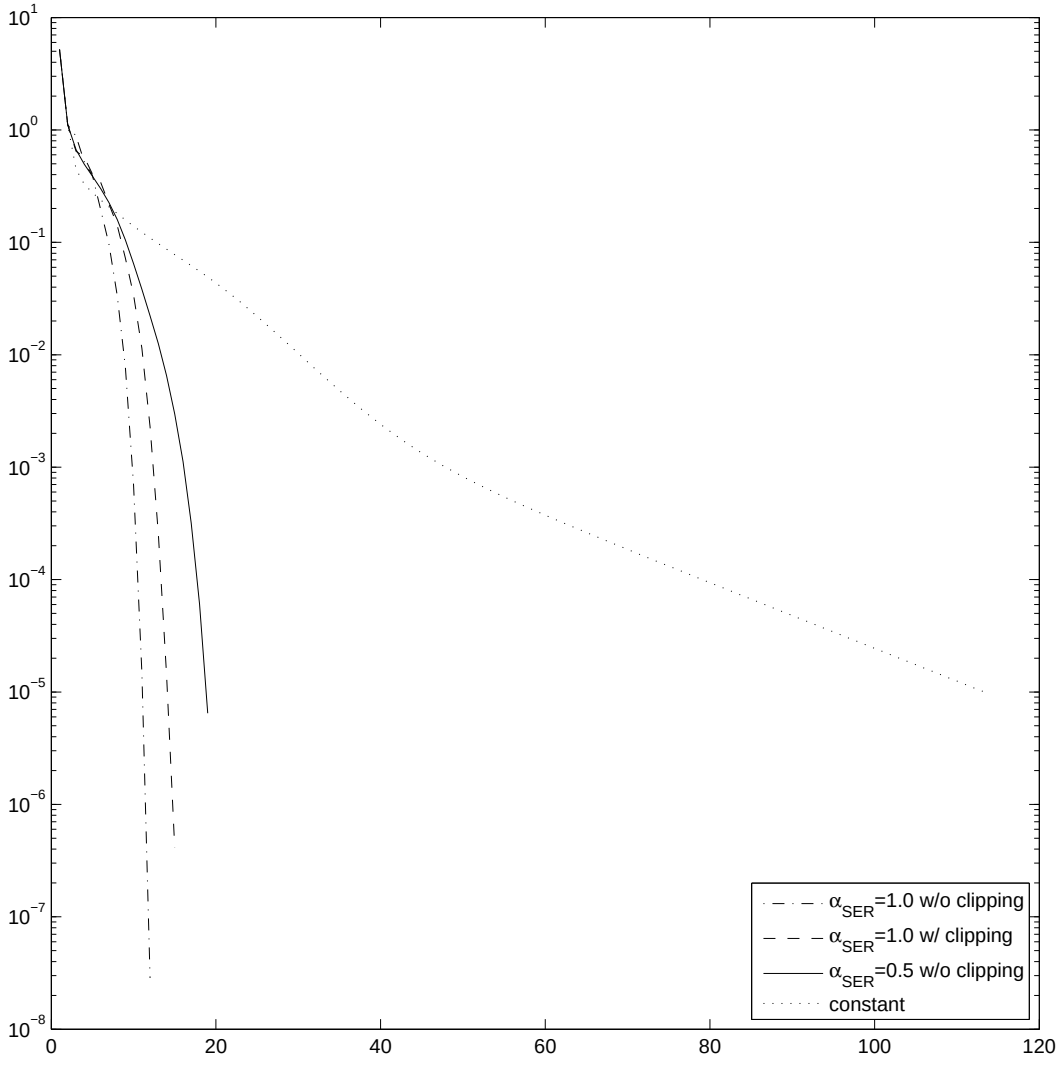


Figure 7.2. $Re = 1000, d\tau = 1.0, grid : 65 \times 65$

7.3. Improvement of Initial Guesses

Unfortunately, use of the methods described before is not enough to cope with difficult problems. The success of the solution process is dependent on the selection process. As observed from Figure 4.2, a bad initial guess requires a lot of Newton iterations whereas a simple estimate as in Figure 7.3 enhanced the convergence. There are several methods to cure the initial guess Knoll and Keyes (2004). One approach is to apply mesh continuation. Since solution is easier for a small problem, a coarse mesh can be used to generate a solution to be transferred to a finer grid. Such an approach is already described in the context of FMG. Another idea is use of pseudo-transient continuation. Even if the problem is steady, a time derivative will provide diagonal dominance which in turn improves the solution of linear steps. Time increments can be adjusted so that the steady state can be reached in less steps (section 7.2). Artificial Compressibility is naturally provides a continuation in time. One other way might to start with FAS to and switch to Newton in later steps. For this new hybrid solver, the jacobian would not be evaluated a lot. Lastly, physical parameters based continuation can be done as explain in the next section.

7.3.1. Parameter continuation

For highly nonlinear problems, it is not wise to start the problem from scratch. A bad initial guess may cause divergence when Newton's Method is used even its inexact variants are tried. Additionally, quadratic convergence can only be achieved if the iterate is around the exact solution. A reasonable method to create a good initial guess is use of parameter based continuation or homotopy. This approach basically uses a problem dependent parameter to estimate a new guess for current problem using previous solutions. Re number in NSE, or the eccentricity, ϵ, in Reynolds Equation are examples of parameters that can be used in conjunction with Continuation Methods. Continuation, also known as Path Following Method, travels along a path and assumes a finite amount of arc length to reach the next point. They are also used to capture the physics in bifurcation problems. Buckling phenomena is for instance solved with Riks' Method in Abaqus Following Knoll and Keyes (2004), if we are after solving a

system of nonlinear equations, $\mathbf{F}(\mathbf{u}, \lambda)$, then

$$\mathbf{u}_{k+1}^{0} = \mathbf{u}_{k}^{m} + \frac{\partial \mathbf{u}_{k}^{m}}{\partial \lambda_k} \Delta\lambda \tag{7.18}$$

where $\Delta\lambda$ denotes $(\lambda_{k+1} - \lambda_k)$ and superscripts denote the number of iteration at problems k and $k+1$, respectively. If problem k is solved after m number of iterations, then an initial guess for problem $k+1$ can be calculated using Equation 7.18. $\frac{\partial \mathbf{u}_{k}^{m}}{\partial \lambda_k}$ in this equation is solved using the formulation in 7.19.

$$\left(\frac{\partial \mathbf{F}_k}{\partial \mathbf{u}_{k}^{m}}\right)\left(\frac{\partial \mathbf{u}_{k}^{m}}{\partial \lambda_k}\right) = -\left(\frac{\partial \mathbf{F}_k}{\partial \lambda_k}\right) \tag{7.19}$$

In the so called "boot strapping" Knoll and Keyes (2004) the second part of rhs of Equation (7.18) might be ignored, however, it will be more elegant if Equation (7.19) is not avoided. A note about Equation (7.19) is that it resembles the Newton update equation and can be solved via the methods introduced in previous chapters. The main difference is that the rhs of Equation (7.19) is not $\mathbf{F}$ directly but its derivative wrt continuation parameter λ .

In Figure 7.3 one can see that the number of newton updates for $Re = 100$ is decreased to 4 if the result of $Re = 1$ is used as an initial guess. Total 8 steps sufficies for convergence. This dramatic decrease in the number of iterations results only a slight increase in the initial norm of $Re = 100$ problem. As the number of unknowns increases, it can be expected that the initial norm will be larger.

7.3.2. Implementation in Demona

Equation 7.19 can readily be solved with any matrix-free krylov solver introduced in preceding chapters. However, there is a catch in the evaluation of the RHS i.e. $\frac{\partial \mathbf{F}_k}{\partial \lambda_k}$. This term is the derivative of the nonliear system wrt the parameter λ. Numerically, this term can be directly evaluated using finite differences but the parameter λ should be perturbed. Generally, physical parameters like Reynolds number or Rayleigh number are defined with fixed variables. In Demona, they are also defined as

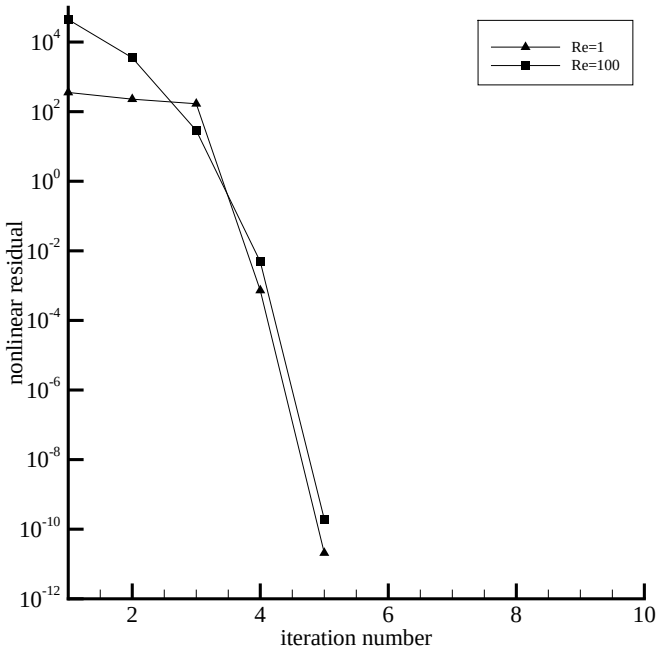


Figure 7.3. Nonlinear convergence history of a SF-VOR equation for $Re = 1$ and $Re = 100$ in a 33x33 grid

a feature of the physics via the variable `physicalParameters` which can be accessed as `myModel%physics(currentPhysics)%physicalParameters(i)`, i being the index of the parameter. To be general, `physicalParameters` is kept as a vector since in some problems more than one physical variables are present like Ra and Pr numbers in natural convection. However, continuation is performed only over the main parameter, i.e. the Ra number. Actually, Homotopy can be carried out for more than one variables but in Demona only the main parameter is considered. Consequently, only this variable is perturbed by an amount of ε to calculate the respective terms. This operation is done by subroutine `continuation`. Note that solution of Equation 7.19 is as expensive as the linear steps in Newton's method so use of preconditioners is essential.

8. FUNDAMENTALS OF DEMONA

In this section, more on DEMONA will be presented. The discussion will start with former definitions of, model, physics and domain. Each of this three components has distinct properties that are essential for efficient use of DEMONA. This section also serves to explain the implementation details of the numerical methods explained before. Additionally, new solution methodologies will be introduced.

There are four main routines in DEMONA, these are namely, `initializeModel`, `preprocessor`, `processor` and `postprocessor`. They are invoked in the same order. `initializeModel` is used to read the user input files and allocate the structure of DEMONA using the number of physics and number of domains per physics. In `preprocessor` part, size of the unknowns is calculated and absolute indices of the variables are determined. These absolute indices does never change during run time. Index matrix, `IM` is created here for the first time. All of the layers are initialized in this part of DEMONA. Additionally, simple rectangular meshes are created here. If a mesh to be modified, some parts of the preprocessor is executed again. `processor` is where all of the computations are performed. Solution is performed according to the `modelSolver` provided by the user. Time dependent problems are also controlled here. `postprocessor` is to create output for the solution as well to perform some post-computations with the data in hand.

DEMONA has a more complex structure than explained in preceding chapters. A model is designated with keyword `myModel`. It is the top level of all variables. All other declarations are members of `myModel`. Important variables of `myModel` is given in Figure 8.1. Solution vectors (`x` and `tempx`) with time histories (`xRTO`, `xPTO`) are stored in `myModel`. Their size is not known before hand, thus they are created in run-time. As observed in `myModel`, `physics` is also a `type` as demonstrated in Figure 8.2. `x` and additional time vector in `physics` are just pointers to the original `x` in `myModel`. In other words,

`myModel%physics(p)%x=>myModel%x(lowerBound:upperBound).`

Bounds of `physics` p is decided during the initialization phase of DEMONA. With this approach, we can work on any physics with comfort although the real work is done on `x` of `myModel`. And yet, `domain` is also another `type` defined in `physics`. As in Figure 8.3, index matrix, `IM` is defined within the `domain` structure. Since for collocated finite volume problems nodes and location of the unknowns do not coincide (observe Figure 8.6.1) node and dof matrices are separated. `domain` has also an `x` which simplifies the implementation of a discretized model even further. `x` of the `domain` is a pointer to `myModel%x`, as well.

In section 3.3, evaluation of `discrete` is performed with `IM`. `IM` is a pointer for `dofMatrix` and is shorter than `myModel%physics(p)%domain(d)%dofMatrix`. Hence, for physics p, and domain d of p, `IM` can be declared as

`IM=>myModel%physics(p)%domain(d)%dofMatrix`

```
type DemonaModel
    type(phy),allocatable :: physics(:)
    integer :: numberOfPhysics
    real(8),allocatable :: x(:)
    real(8),allocatable :: tempx(:)
    real(8),allocatable :: xPTO(:)
    real(8),allocatable :: xRTO(:)
end type DemonaModel
```

Figure 8.1. Elements of `myModel`

8.1. Pointers in DEMONA

Pointers are frequently used in DEMONA. First reason is to simplify coding; use of `type`'s forces us to use them since the definition of all members of the model is long. Also, some of the variables are arguments of some other variables so calling

```
type phy
    type(dom),allocatable :: domain(:)
    real(8),pointer :: x(:)
    real(8),pointer :: xPTO(:)
    real(8),pointer :: xRTO(:)
    integer :: numberOfDomains
    integer :: dof
    integer :: dFX
    real(8) :: refXcoor
    real(8) :: refYcoor
    real(8) :: Lx
    real(8) :: Ly
end type phy
```

Figure 8.2. Elements of `physics`

```
type dom
    integer, allocatable :: nodeMatrix(:,:,:)
    integer, allocatable :: dofMatrix(:,:,:,:)
    integer :: allBC
    integer :: nx
    integer :: ny
    integer :: numberOfGhostCells
    integer totalDof
    real(8),pointer :: x(:)
    real(8),pointer :: xPTO(:)
    real(8),pointer :: xRTO(:)
    real(8),allocatable :: xCoor(:)
    real(8),allocatable :: yCoor(:)
    real(8) relXcoor
    real(8) relYcoor
end type dom
```

Figure 8.3. Elements of `domain`

each parameter with is full address is a tedious task. Second use of pointers is related to the numerical methods. As explained before, there are two main vectors, `x` (`myModel%x`) and `tempx` (`myModel%tempx`). `x` is where the solution of the model is kept. Pointers to those vectors are `y` and `tempy`, respectively. To be clear, `y=>myModel%x` and `tempy=>myModel%tempx`. Operations are always performed on `x` or some portion of it. However, during the computations some calculations should be performed by modifying `x` i.e. summing it with another vector (like Δ`x` in Newton's Method). Such computations are carried out on `tempx` rather than `x` to protect the current iterate. `tempx` is also used to select the perturbation parameter ε. Computations are generally based on the use of functions and it may occur that `tempx` is not needed at all. One might call any function `func(x+v)` so that `x` is not modified. In DEMONA, however, `x` is never used as argument of functions. All routines that needs `x` has direct access to it. The computations are adjusted with another pointer $\bar{\texttt{x}}$ which points either `x` or `tempx` depending on the iteration process. During the iterations, `discrete`(0) will use `x` and `discrete`(1) `tempx`. That means, while working with Krylov subspaces, only `discrete`(0) is used. At the time `x` is to be updated with Δ`x` the operation in Figure 8.4 is performed. Use of two pointers like `y` and $\bar{\texttt{x}}$ for `myModel%x` is not a contradic-

```
tempy=y
tempy=tempy+deltax
f=discrete(1)
!calculate 2-norm of f and decide whether deltax is acceptable
! if acceptable, y=tempy
! if not, perform damping strategies with tempy
```

Figure 8.4. Application of `discrete`(1) in Newton's Method

tion, actually. It is also used for pedagogical reason. As observed in given examples in section 3.3, always `x` is used for computations. The `x` used in DEMONA is actually a pointer; it is the $\bar{\texttt{x}}$ explained before. Depending on the argument of `discrete`, it either performs on `y` or `tempy`. Consider Figure 8.5.

```
function discrete(opt)

    real(8), pointer  :: x(:)

    if (opt.eq.0) then
        x=>y        ! which actually points to myModel%x
    elseif (opt.eq.1) then
        x=>tempy    ! which actually points to myModel%tempx
    end if

    !continue with x
    ...
end function
```

Figure 8.5. Use of pointer `x` in `discrete`

8.2. Definition of a Model Problem

In order to set up a new problem in to DEMONA, several user routines should be provided. First of all, information about the model, as well as the physics and domains should be provided. This is achieved by using `modelInfo`, `physicsInfo` and `domainInfo` files. In the foregoing discussion `currentPhysics` will refer to the physics that the code is working on. Similarly, `currentDomain` will depict the domain that receives attention.

8.2.1. modelInfo

This is just to set the number of physics of the current model. Parameter used to define the physics is `numberOfPhysics`. It is an element of `myModel` and accessed with `myModel%numberOfPhysics`.

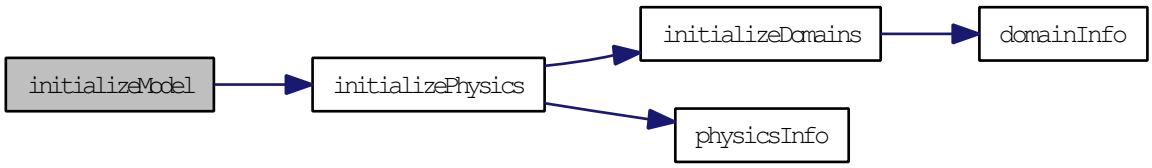


Figure 8.6. `initializeModel`

8.2.2. physicsInfo

In this routine, basic information about the physics is stated. Number of domains per physics is defined using `numberOfDomains`. `dof` of the physics is also defined here. Bounding box dimension are given with `Lx`, `Ly`. Reference coordinates are useful to place both other physics and domains of the `currentPhysics`. `dFX` is the information about the discretization. The options are `dFD`, `dFV`, `dFE` for finite difference, finite volume and finite element, respectively. `damping` is to be used in linear split forms like Picard so to stabilize the convergence. Final declaration is the `numberOfPhysicalParameters` variable. This is to define the physical variables of the physics like, Reynolds number, Re or Rayleigh number, Ra. Normally, this definition is enough to perform computations. However, in some cases, we might be utilizing continuation method; there we can read an initial guess and correct it using the parameter based computations (section 7.3.1). A sample declaration for a SFV model is given in Figure 8.7. As observed, damping is applied only for Vorticity Transport Equation (Equation 2.6b). It is also apparent that the solution of $Re = 100$ case is used as an initial case for current problem with $Re = 1000$. Note that, the variables shown in the table are in fact pointers. Pointers are associated with their counterparts in the definition of the structure. For example, `dof=>myModel%physics(currentPhysics)%dof`.

8.2.3. domainInfo

This routine is shorther than the physicsInfo. In here, number of nodes in each coordinate direction is given. Reference coordinates are replaced with relative coordinates; placement of the domain is adjusted with respect to the physics' reference coordinates (Figure 8.8). For single domain problems, the domains can use the di-

```
select case(currentPhysics)
    case(1) ! definitions for the first physics
        numberOfDomains=1
        dof=2
        dFX=dFD ! finite difference
        TecplotVariables='VARIABLES="x","y","str","vor"'
        refXcoor=0.d0
        refYcoor=0.d0
        Lx=1.d0
        Ly=1.d0
        damping(1)=1.0d0  ! damping parameter for stream function
        damping(2)=0.01d0 ! damping parameter for vorticity
        numberOfPhysicalParameters=1 ! Reynolds number
        myModel%physics(currentPhysics)%physicalParameters=0.d0
    case(2) ! definitions for the second physics
        ...
end select
```

Figure 8.7. Sample for `physicsInfo`

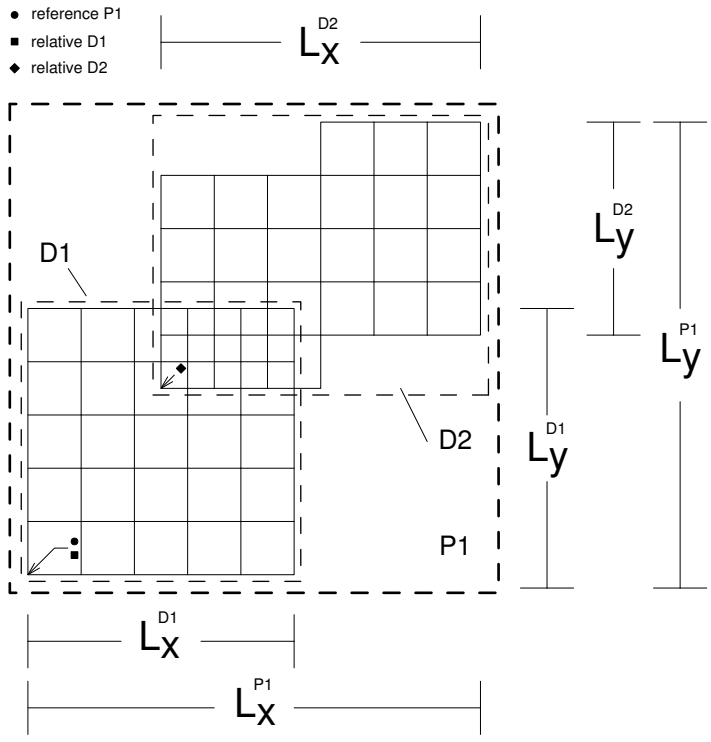


Figure 8.8. Definitions of Physics and Domains

mension of the physics or new dimension can be defined for domains. In reality, the dimensions of the domains are important not the physics. However, if there are more than one domains in the physics, then their locations can be automated with reference to the physics. If finite volume is active, then number of nodes in each direction is first cut by one and then expanded by the number of ghost cells, `numberOfGhostCells`. Reduction by 1 is necessary since nodes actually define the geometry of the cells, unknowns are located at the center of the volumes for collocated arrangement. As a result, number of unknowns is $n_i - 1$, i being the coordinate direction. To implement the boundary conditions, it is sometimes necessary and also useful to employ ghost cells. For high order upwind schemes, more than one ghost cells might be needed (2 for QUICK, Appendix A). Hence, number of unknowns are expanded to cover ghost cells, too. An example is given in Figure 8.9. Pointers are also used in domainInfo, as well. Here, the declarations are even longer since domains are elements of the physics's: `nx=>myModel%physics(currentPhysics)%domain(currentDomain)%nx`. As a note, domains should be defined per physics; separate domainInfo files for each physics is not used. Consequently, information given in the table is valid only for one physics. Values of other physics are to be stated in the same routine, as well.

8.2.4. addBC

This is the routine where the BC's are defined. This is closely related to the discretization of the model problem. Boundary conditions should be defined for each domain, however, even domains are different, the distribution of the BC might be similar, as a result same definition can be used for those domains. In this function, `boundary` layer of `IM` of the domains are filled. The entries will guide the set as well as `discrete`. As denoted in section 3.6, `boundary` layer is decided by considering all of the local dof's. `boundary` layer is filled similar to Figure 3.23.

8.2.5. model

`myModel` organizes the relation between physics's and domains. `model`, on the other hand, is the discretized form of the problem. `model` is a fortran module (section

```
select case(currentPhysics)
    case(1)
        select case(currentDomain)
            case(1) ! definitions for the first domain
                nx=65
                ny=65
                allBC=6
                relXcoor=0.d0
                relYcoor=0.d0
                !Lx=1.d0
                !Ly=1.d0
                !or
                Lx=myModel%physics(currentPhysics)%Lx
                Ly=myModel%physics(currentPhysics)%Ly
                !numberOfGhostCells=2
            case(2) ! definitions for the second domain
            ...
            end select
    case(2)
        ! domainInfo for second physics
end select
```

Figure 8.9. Sample for `domainInfo`

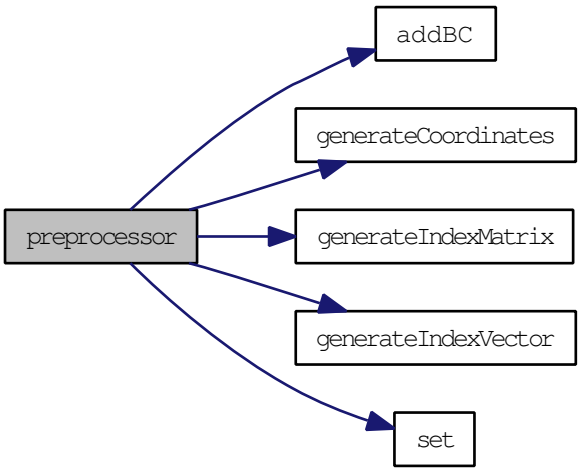


Figure 8.10. `preprocessor`

3.3) that contains `discrete`. The structure of `discrete` is given in Figure 3.8. However, there are some details to be discussed in here. Observe Figure 8.11. Each physics is defined separately (without using a do loop) using the variable `modelPhysics`. `endIndex` is either 0 or 1. If 0, that would mean that the physics in concern is not active currently. The same idea is also used for domains as well as BC's. Now, we look in to the loops. The form is stated in Figure 8.12. If domains are solved sequentially, then always one of the `endIndex`'s is active and others are passive. If the domains are to be solved in a coupled fashioned, then all of the loops are visited. Note that, `endIndex`'s are part of the global set, $\mathcal{S}_G$. Which boundaries are to be visited is on the other hand, is decided by local set, $\mathcal{S}_L$ and its implementation is like in Figure 8.14. `myBC` is to define that following code lines are valid for that particular bc number until a new `myBC` is defined. Figure 8.14 gives an example for SFV formulation where `myBC=2` is the left boundary condition. Note that, if a particular boundary is not in local set, then `maxJ` of that BC is set to 0, so the entire loop is skipped and computations proceed with the next bc on the line. There are, however, endIndex's in the i-loop. These variables are used to include local dof into $\mathcal{S}_L$ or not and act per i basis. Thus, run-time sets can be used to eliminate some of the unknowns. This form is also helpful to realize Picard like solution strategies. Another point is that, the update of `discrete(IM)` is performed at the end, all previous computations are carried our for RHS. With this idea, one can use additional right hand side vectors to the solution (like in FAS in section 5.3). Moreover, only respective dof's equations are assigned to `discrete` when local sets are in concern.

8.2.6. modelSkip

In varios problems, some portions of the domains are not filled, i.e. there might be holes or missing part in the problem. However, `nx` and `ny` of the domain is always defined for the full bounding domain. `modelSkip` is a function to designate which nodes are not in the domain. It is called in `preprocessor`. While the variables are assigned to absolute numbering, those points are skipped. If a node has a zero value as the absolute index, then that point will not be a part of any set. `modelSkip` could make use of the information of the domains so skipped nodes can be adjusted automatically

```
function discrete(opt)
...
modelPhysics=1

do myP=1,myModel%endIndex(modelPhysics)
    ! perform computations on the domains of physics 1
end do

modelPhysics=2

do myP=1,myModel%endIndex(modelPhysics)
    ! perform computations on the domains of physics 2
end do
...
end function discrete
```

Figure 8.11. `discrete` and physics

wrt to the number of increment in both coordinate direction and dimensions of the geometry.

8.2.7. modelSolver

modelSolver states the solver to be used in `processor`. Currently it needs an interface to reach the solver, one can either create a new solver or use the ones prepared in the code. Some template solvers of DEMONA are `RunNewton`, `RunPicard`, `RunGS`, `RunMS`, `RunHybrid`.

8.2.7.1. RunNewton. This is just an interface to call Newton's Method. Main property of this routine is that it is also called from other solver interfaces.

8.2.7.2. RunPicard. Picard's method is executed using this routine. In Picard's Method, first IM is modified, then `Newton` is called to solve the resulting system.

```
do myP=1,myModel%endIndex(modelPhysics)
    modelDomain=1

    IM=>myModel%physics(modelPhysics)%domain(modelDomain)%dofMatrix

    do myD=1,myModel%physics(modelPhysics)%endIndex(modelDomain)
        ! perform computations on the bc's of domain 1
    end do

    modelDomain=2

    IM=>myModel%physics(modelPhysics)%domain(modelDomain)%dofMatrix

    do myD=1,myModel%physics(modelPhysics)%endIndex(modelDomain)
        ! perform computations on the bc's of domain 2
    end do
    ...
end do
```

Figure 8.12. `discrete` and domains

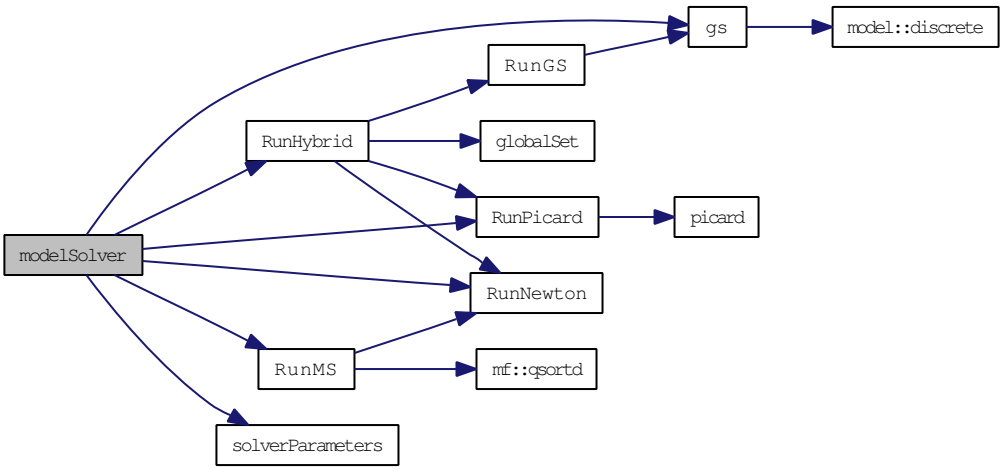


Figure 8.13. `modelSolver`

```
!short hand notation for myModel%physics(modelPhysics)%domain(modelDomain): MMM

do myD=1,myModel%physics(modelPhysics)%endIndex(modelDomain)
    myBC=2
    maxJ=>MMM%set%bc(myBC)%maxJ
    jVec=>MMM%set%bc(myBC)%jVec

    do myJ=1,maxJ
        j=jVec(myJ)
        maxI=>MMM%set%bc(myBC)%j(myJ)%maxI
        iVec=>MMM%set%bc(myBC)%j(myJ)%iVec

        do myI=1,maxI
            i=iVec(myI)
            maxT=>MMM%set%bc(myBC)%j(myJ)%i(myI)%maxT
            tVec=>MMM%set%bc(myBC)%j(myJ)%i(myI)%tVec
            endIndex=>MMM%set%bc(myBC)%j(myJ)%i(myI)%endIndex

            t=1 !unk1
            do myT=1,endIndex(t)
                STFC=x(IM(i,j,t,absolute))
                RHS(t)=STFC-0.d0
            end do

            t=2 !unk2
            do myT=1,endIndex(t)
                STFC=x(IM(i,j,t-1,absolute))
                VORC=x(IM(i,j,t,absolute))
                STFE=x(IM(i+1,j,t-1,absolute))
                RHS(t)=VORC-2.d0*(STFC-STFE)/dx**2.d0
            end do

            do myT=1,maxT
                t=tVec(myT)
                discrete(IM(i,j,t,relative))=rhs(t)
            end do
        end do
    end do

    myBC=3
    ...
end do
```

Figure 8.14. `discrete` and bc's

8.2.7.3. RunGS. Gauss-Seidel is accessed using this routine. GS can worked either point wise or dof-wise.

8.2.7.4. RunMS. This routine is to use Multiplicative Schwarz Method. In this routine, the domain in concern is divided into small parts using the variables `xPartition` and `yPartition` defined by the user. Later, the size of the domain could be expanded by using the `overlap` value. There are four options to sweep the domains. Forward, reverse, random or user-specified. Forward starts from the first sub-domain and reverse start from the last subdomain. When random is used, then at each MS iterations, sub domains are visited in a random fashion. As a last resort, user can specify its own visiting order.

8.2.7.5. RunHybrid. In hybrid solvers, different solvers are combined together to create a new solver. One example is to start computations with RunPicard and then continue with RunNewton. This way, an improved initial guess will be provided for Newton's Method. RunPicard can be replaced with RunGS or RunMS to test different approaches.

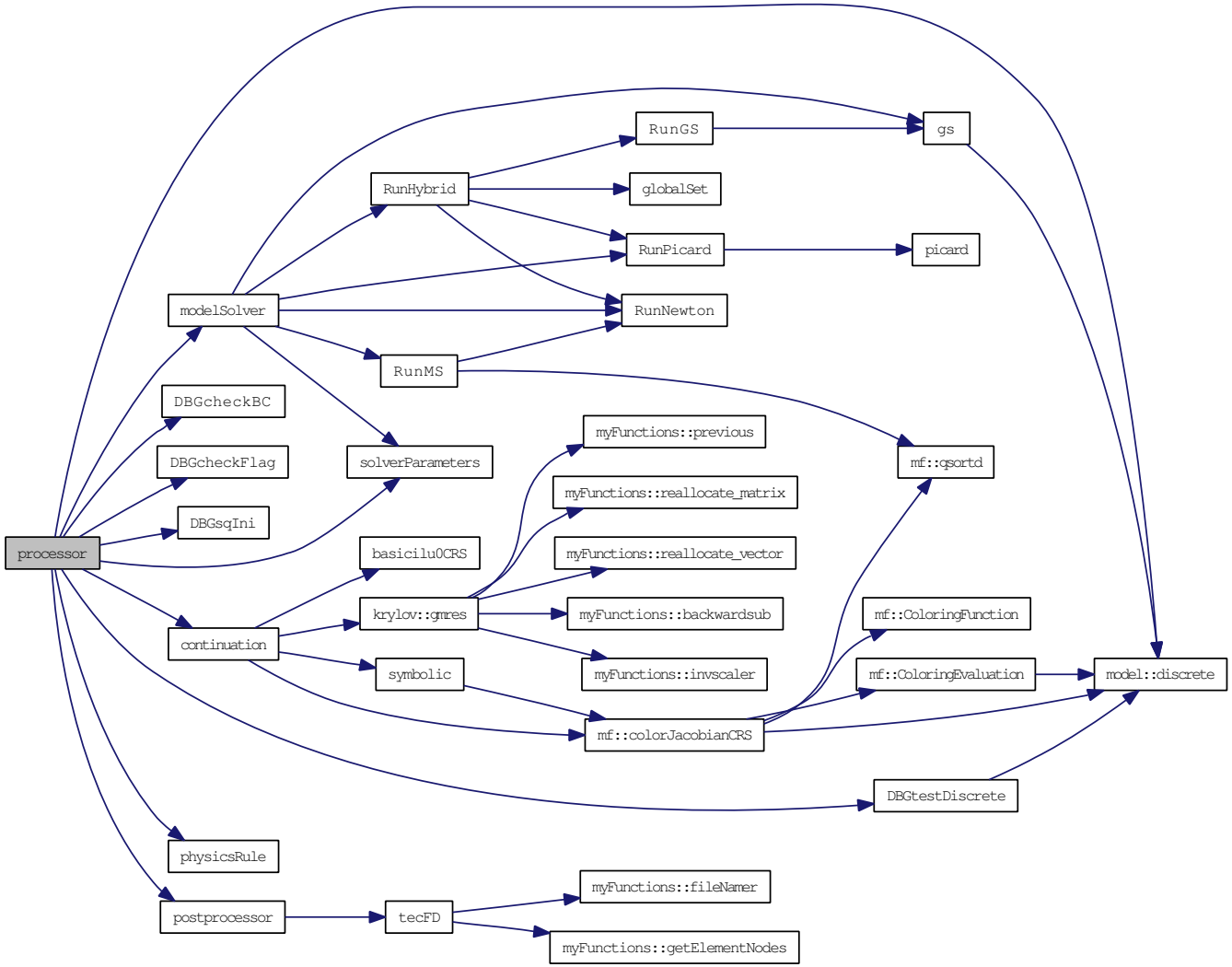


Figure 8.15. `processor`

8.3. Implementation of the Solvers

8.3.1. Newton's Method

Call graph of Newton's Method in DEMONA is given in Figure 8.3.1. Top of the graph shows the routines that computes color based Jacobian and ilu0 preconditioner. `krylov::gmres` is the selected linear solver in this example. ε that is necessary to perform matrix-free computations is calculated in `mf:MatVec` where `MatVec` is an interface to perform Matrix-Vector products and `mf` is the module related to matrix-free calculations. Newton's Method is called frequently, whether a Newton-Krylov solver is invoked or Gauss-Seidel is called. When NK is the solver, than full problem is attacked. If Gauss-Seidel is to be used, then Newton's Method acts locally and solver a dof×dof Jacobian.

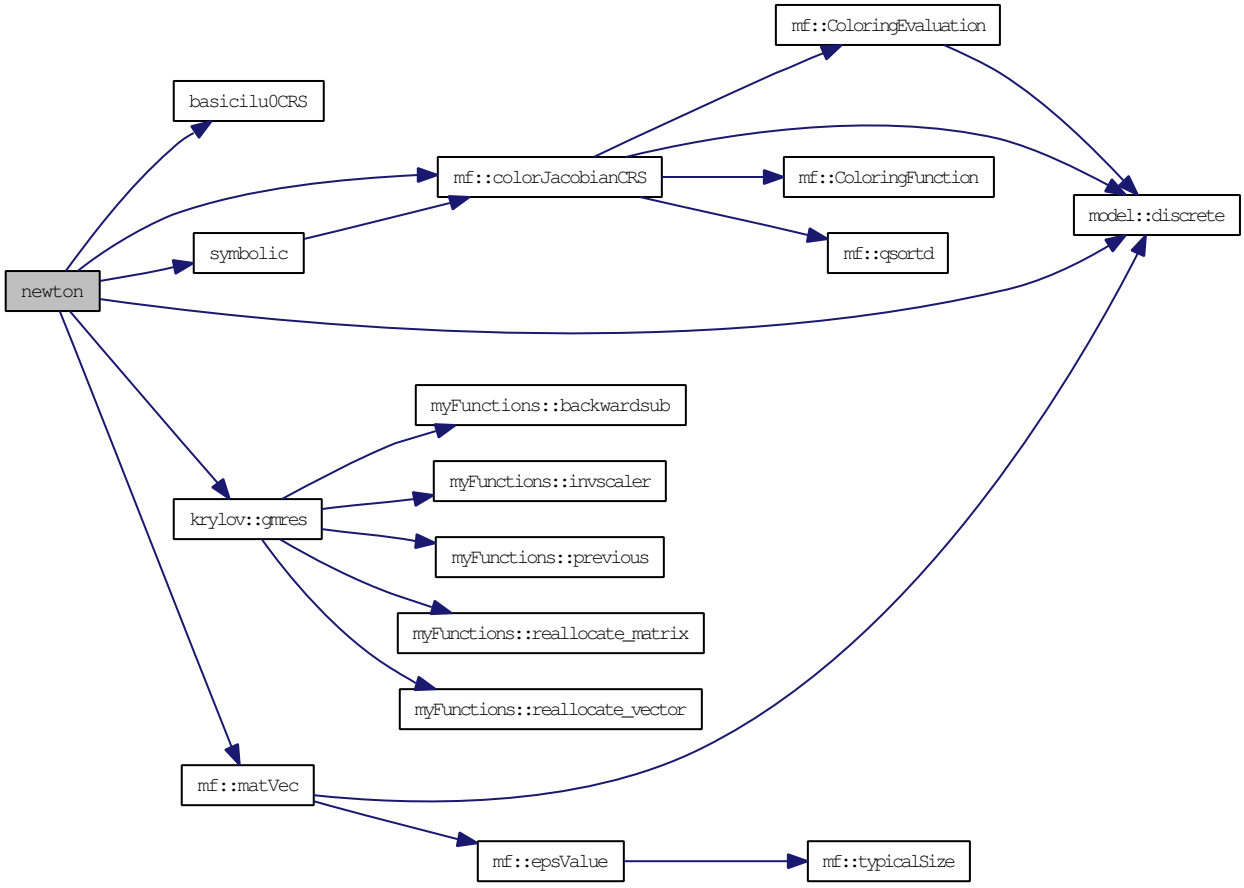


Figure 8.16. Newton's Method in DEMONA

8.3.2. Application of Picard's Method

As explained before, in Picard's Method only one of the dof's is solved at a time. One Picard iteration ends when all dof's are visited. Algorithm for Picard method in Demona is like in Figure 8.17. In applying Picard's method, it is extremely useful to use a damping factor to stabilize the solution process. If damping is avoided, then `oldValues` vector will not needed since y will be updated directly in the solver, however, `oldValues` are kept for a robust solver. As an option, allocation of the `oldValues` vector can be avoided if damping is 1. In the solver, dof based damping is also used; that means different damping factors can be employed for each dof. In SFV formulation for instance, damping factor of the stream function equation is 1 but for vorticity transport it is kept small (≈0.01). Values of the damping factors are user input and defined in `physicsInfo`. Additionally, damping factor might be allowed to increase for a converging computation so the computations end in less amount of iterations but in sample computations it is observed that varying factors sometimes does not help and the calculation diverge.

```
do picardIter=1,maxPicardIter

    do i=1,dof
        IM(:,:,:,flag)=0 ! initialize the flag layer
        IM(:,:,i,flag)=1 ! activate only i'th dof
        call currentSet
        oldValues=y(setVec)

        "solve the problem"

        y(setVec)=damping*y(setVec)+(1-damping)*oldValues
    end do

end do
```

Figure 8.17. CODE Listing for Picard's Method

In Figure 8.17, two loops are shown. First loop is to control maximum Picard iteration. In the second loop, dof's are selected one-by-one. First `IM` is initialized to zero and then only i'th unknown layer is set to 1. This declaration is enough to set Picard solver. In the "solve the problem" line, Newton-Krylov (even if the resulting sub problem is linear) or Multigrid can be used. By default, Newton's Method is implemented.

In Picard's method, only one of the dof is solved per intermediate iteration, only. One can also come up with different strategies. For example, first step can be solved with both ψ and ω in the system and the second step alone with ω or vice versa. Another idea might be useful in non-isothermal forced convection problems, for instance. In contrast to free convection, temperature field is not coupled with the velocity field. General rule is to solve the velocity field first and then to calculate the temperature distribution. Since the velocities are known, energy equation reduces to a linear problem and the solution can be found in just one Newton step. On the other hand, this would require to define a new physics. Instead, one can solve all four equations simultaneously although it is not necessary physically. This approach, however, increase the size of the unknowns so more iterations would be required for linear solvers. New approach is to define all equations but to keep only three equations in the set till convergence. That means, energy equation will be out-of-set. In can be included in to the set, or better it can be the only equation in the set as soon as the three-equation coupled model is solved. For various problems, different combinations of the equations can be tested for improved convergence.

8.3.3. Application of Block Smoothers

In block Jacobi and GS methods, dof's at a grid point are solved at the same time rather solving them one by one. Sweeps are performed on J and I component of the current set. Flag layer is declared like following:

```
do myJ=1,maxJ
    j=Jvec(myJ)
    do myI=1,maxI
        i=j(myJ)%iVec(myI)

        IM(i,j,:,flag)=1 ! all local dofs at point i,j added to the set

        call currentSet

        oldValues=y(setVec)

        "solve the problem"

        y(setVec)=damping*y(setVec)+(1-damping)*oldValues

    end do

    IM(i,j,:,flag)=0
end do
```

Figure 8.18. CODE Listing for Picard's Method

Again damping is introduced through oldValues vector whose length is dof. Damping is especially useful when smoothers are solved with multigrid. In this thesis, a variant of GS is introduced. Second last line reads as `IM(i,j,:,flag)=0`. Here, the `IM` is initialized to zero again. Instead, we can ignore this line and increase the number of unknowns dof by dof so 1's start to accumulate in the `IM`. As a result, the set will expand until the last node. At this final attempt, the system will be full. Motivation is to improve the initial guess for the full system. Another variant could also be use of more than one node, i.e. a set of 4 nodes for example could be used to solve the problem. In fact, this is the application of the multiplicative schwarz idea which is explained next. When we look from domain decomposition window, GS is just a special case of multiplicative schwarz method where the domain is partitioned into $\frac{N}{dof}$ number of subdomains. Same analogy is valid for jacobi and additive schwarz method.

A variant of line relaxation can also be implemented with a modification on the loop such that full line of unknowns are added in to the set i.e. IM(:,j,:,flag)=1. Instead of block solves, dof-by-dof solution is also possible by visiting each dof separately. Yet another variant is an "accumulated flag" version of GS method. In this approach, the 1's are not initialized to 0 and kept in the flag layer. Therefore, number of unknowns in the local set increases. At the end, it will be equal to the the full set and we expect and improved initial guess for Newton's Method. A variant of accumulation might be starting from different point in the domain instead of the first point. This would like dropping inks onto different spots on the water. As the ink fragment are diffused on the surface of the water, 1-flag's will be diffused in the flag-layer. A final note, red-black Gauss-Seidel can be implemented easily with reordering the indices.

8.4. Application of Domain Decomposition Method in DEMONA

One of the advantages of DEMONA is that application of Domain Decomposition Method is easy. This simplicity is because of the *flag* layer of IM. When the flag layer is adjusted in such a way that only one of the sub domains is in the set, then the solver works only for that domain.

Default artificial boundary condition for the interfaces is non-homogenous Dirichlet BC. In fact, this is not stated explicitly, nature of the set approach automatically provides this boundary condition. On the other hand, in some cases Neumann or Robin BC's should also be considered either for all dof's or for some of them . To use these boundary conditions, we have to define their discretization beforehand in the model. Even if they are absent in global set, $\mathcal{S}_G$, they will be used in localSet, $\mathcal{S}_L$. When setting a sub domain using the flag layer, boundary layer should also be modified to reflect the changes on

The user should provide three basic parameters basically, `xPartition`, `yPartition` and `overlap`. x and y partitions defines the number of sub domains in the prescribed directions, respectively. Overlap determines the amount of extension of the sub domain on the other part of the whole domain. As overlap increases, DDM converges

faster. Generally, a nonzero overlap enhances the computations a lot by just providing a mild increase in the number of sub domain unknowns. If we need to solve a single domain, conforming DDM is an excellent alternative. Using those three parameters and Multiplicative Schwarz approach. Additive Schwarz needs extra storage for new results so they are not used as soon as they are calculated.

From this point of view, parallel processing is also straight forward. Each processor will work on its own flag layer and communicate for updated values. Of course Additive Schwarz is more advantageous since communications are necessary only at the end of one sub domain sweep. Besides Newton's Method, other nonlinear solvers like FAS or Picard can be used in sub domain solutions. Or even hybrid solvers can be tested like first Picard and then Newton's Method. If the user desires, different solvers for each sub domain can be adapted.

8.5. Set Rules

Definition of a new set can be realized only if IM is formed properly. For algorithmic sets, the rule for sets are given by the algorithm. For run-time sets, however, a criteria should be determined. Before examining some of the ideas, a few observations are to be made. In calculating the norm of nonlinear residual norm, $normR$, 2-norm is used. This norm is the square root of sum of the squares of all local residuals. When all local residuals, $normR_l$ are compared against to the norm of the residual then all of them all smaller than $normR$. So, this cannot be used as a set rule. Here, instead of monitoring the $normR$, one can select an intermediate or a set of intermediate tolerances that are greater than the nonlinear convergence criteria but gets smaller as iterations proceeds and be equal to the stopping tolerance at the final stage. If damping strategies are not used, is some cases it might happen that $normR$ increases or it might not reduced as fast as we anticipated. In these situations, comparison of local norms to $normR$ seems to be more appropriate. In order to achieve that, we might use RMS of the residual instead of 2-norm so an average value could be determined. Then all unknowns smaller then the RMS norm can be dropped from the set. This norm can also be useful to decide on the intermediate norms of the previous ideas.

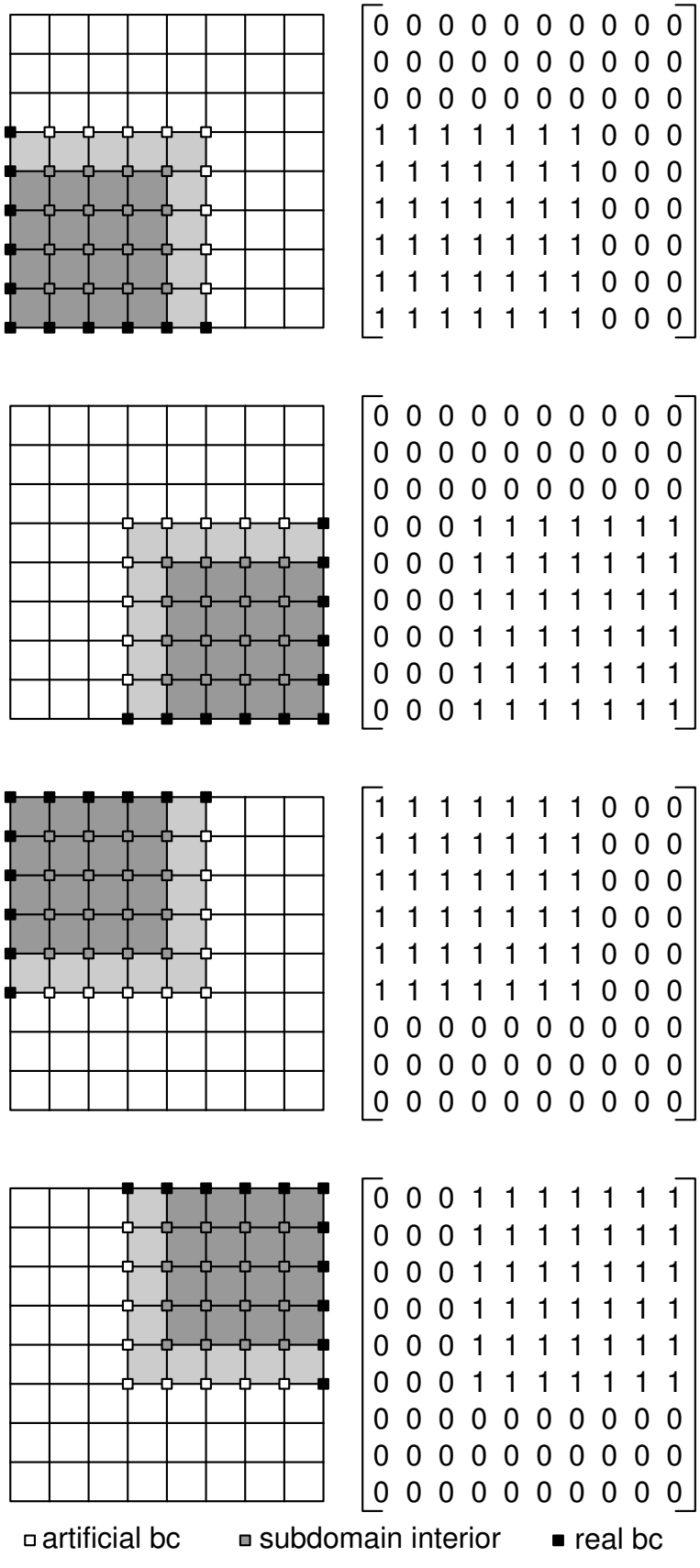


Figure 8.19. Four subdomains with their flag layers, overlap=1

When the set rule is ready, then all of the unknowns in $\mathcal{S}_G$ will be scanned and flag layer of each unknown will be set to either 0 or 1 for out-of-set and in-set variables, respectively. Depending on the problem, isolating a point might deteriorate the solution of the system. In these instances, some of the neighboring points could be kept in the set. To accomplish this task, one can keep an eye on the gradients to capture large changes on the solution field and include more points than the set rule allows.

Set idea to perform a residual norm based computation is the generalization of the ideas stated before. In this method, a random point cloud and dof cloud is selected from the domain. Depending on the norm of the each dof (different local dofs can be compared with different tolerances), unknowns that not satisfy the criteria will be flagged like they are punished and stayed for detention after school. They are solved again until they are as successful as others. In this way, we avoid that some points are reduced fast in the residual norm but the others not. Another analogy can be made considering a group of people walking in the forest. dome of them - youngsters perhaps - might be fast compared to the main group. In that case, these youngsters should wait until those others catches them so no one can be lost. In ordinary computations, however, it is like Tour de France -no one waits the others- who reaches first win that stage, however, for that stage to be completed everyone should reach to the end (program converged) or some of them might quit (divergence). Next stage (on the next day) could not be started if half of the bikers are not finished (new iteration or new time level can not be started.)

Selection of the set tolerance is also important. At given intervals - maybe at each Newton step, all f norms can be checked against the convergence tolerance or against an intermediate tolerance which is larger than the convergence criteria (like half power 10-4 vs 10-2). Here an intermediate or a series of intermediate tolerances (like every intermediate tolerance is a check point on a race track) seems reasonable, since a converged point might have a modified norm if the points around, or points around the points around or etc. change because of the computations on the reduced set. Hence, even if they are regarded as converged before, they might be solved again

to fulfill the criteria. So intermediate check would promise smoother convergence. We can end the metaphors with this analogy, if those youngster finish their tour on the woods early, and if the old ones are too late - then these youngsters might be worried and go back to look the others until they found them. Then, they walk the same way again but all together. Intermediate controls, on the other hand, reduces the risks of go backs or at least returns would not be long.

As a set is decided either algorithmically or run time a vector called `setVec` is created. `setVec` has a size of `currentSetLength` which amounts the total number of unknowns in the set. The entries of this vector is the index values of the unknowns in the set. Referring to Figure 3.19 on page 44, the entries of setVec are (1,2,6,7,8,9,13,14,22,23,24). When `x` is to be updated, then `x(setVec)` is used to operate only on the elements of the set. Note that size of Δx vector is `currentSetLength` since it is solved for the unknowns on the set.

8.6. Other Aspects of DEMONA

8.6.1. Collocated Grids

For finite difference the numbering is lexicographic starting from the bottom left node. In finite volume, default numbering is also lexicographic and in this order: First the interior zone then the first ghost cell layer and afterwards the second ghost cell layer. In ghost cells, the order is, bottom, top, left and right as seen in Figure 8.6.1.

Downside of the use of an index matrix is the indirect addressing. This method has drawbacks when it comes to the performance of the solver. That's why pointers are used frequently to improve the computations.

8.6.2. Staggered Grids

Staggered grids have nice feature in the numerical solution of incompressible fluid mechanics. First of all, continuity is satisfied within a computational cell and resulting

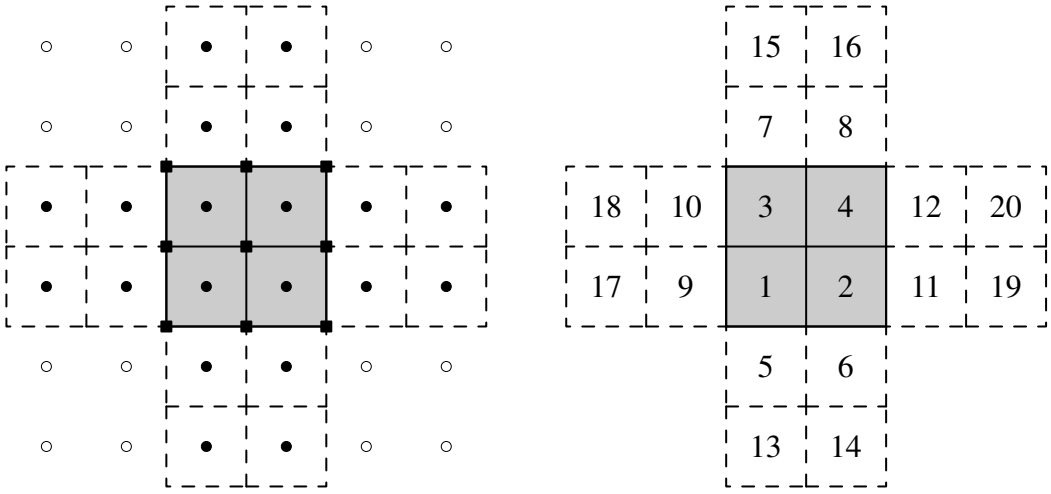


Figure 8.20. Finite Volume grid. Black circles are active dof matrix elements and white circles are passive dof elements. Black squares are the elements of the node matrix

pressure fields is smooth where spurious oscillations are absent. On the other hand, grid structure of such grids is not easy to establish which in turn complicated the numerical analysis. In Demona, staggered grids can also arranged with dof and node matrices in exchange to an increase in the dimension of the dof matrix. For a cell, the locations of the variables are illustrated in Figure 8.6.2 As observed from Figure 8.6.2

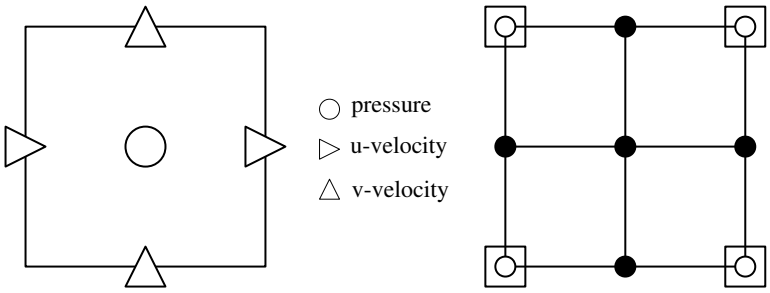


Figure 8.21. Variables on the staggered grid (left) and `IM` lines (right).

for a staggered cell Node matrix is $2x2$ whereas dof matrix is $3x3$ with 4 passive entries. Furthermore, active entries are not fully active. For instance, `IM`(2,1,:,:,1) is equal to 0 since at that point pressure is not defined but `IM`(2,1,:,:,3) is 1 since v-velocity is present there. Similarly `IM`(1,2,:,:,2)=2 and `IM`(2,2,:,:,1)=3. So to say, only 1/3 of each active entry is a valid indexing point. If a non-isothermal problem was to be solved, then temperature would be stored at the center like pressure and `IM`(2,2,4,1)=4

8.7. Use of other Solution Techniques

Use of ADI is suggested in section 4.1.3 to linearize nonlinear problems. Unfortunately, implementation of the idea is not straightforward. The problem is that same unknown set is used with different discretizations. Normally, different discretizations would mean different physics, so for a 2D problem, definition of two physics can barely allow the user to perform ADI. In this approach, however, same unknown set is declared twice, which is redundant. The idea is to use more boundary values to be defined in the `boundary` layer of `IM` in the `addBC` routine. For a n-dimensional problem, n multiple of total BC definitions should be used. With switching from one bc set to other, discretization can be varied and the computations could be performed on the same unknown vector.

SIMPLE like solvers can also be used in DEMONA. Main feature of SIMPLE method is that, velocity components are solved together excluding the pressure field. Later, pressure is updated separately. Implementation of this solver can be realized with a PICARD like technique such that, instead of solving each variable one-by-one, velocities are handled as a group and pressure alone. As a result, periodic set definition between these two groups allow the use of SIMPLE and its variants.

8.8. Transient Problems

In DEMONA transient problems are can be treated both implicity or explicitly (or even semi-implicitly) while defining some of the variables in the stencil either from `y` or `yRTO`. On the other hand, the solver designed for implicit problems so NK is still the default solver for both cases i.e. ode solvers like Runge-Kutta are not implemented. When explicit formulation is used, then all variables are written from previous levels expect the time derivative. In that case, the Jacobian of the nonlinear problem is just a multiple of the Identity matrix and the problem is linear (unless time dependence is linear - nonlinear time terms are not present in the models used in this thesis and yet they can be treated, as well.) So, Newton's Method becomes just an elementary operation to change the solution set. In the implicit approach, however, sparse and

varying Jacobians are used as in steady-state problems. Consequently, Newton-Krylov solvers should be used to solve Jacobian matrix related linear equations.

8.9. Final Notes

As mentioned in section 3, this solver is an attempt to test various couplings and solution techniques to solve coupled-field problems. Optimization refers to the decision on the solution approach rather than the computation time.Whom is after designing a tailored software for a specific problem is free to eliminate some of the bottlenecks of the program like index matrices or decision trees.

9. RESULTS AND DISCUSSION

In this section, results of many problems are given. The discussion start with linear test problem. Here, the performance of linear solvers are compared. Both Newton-Krylov (even if the problems are linear) and Multigrid methods are employed. Domain decomposition is also investigated. Later, nonlinear test problems are demonstrated. First, the Bratu problem is presented. Later, the lid driven cavity problem is investigated with stream function - vorticity, velocity - vorticity and Artificial Compressibility formulations. In the first two approaches, only finite differences are used. In AC, however, staggered grid and finite volume approaches are also considered. The problems are tested with Newton-Krylov and FMG-FAS solvers.

The discussion continues with sample transient problems. Later, some benchmark problems are considered. Multiphysics problems start with the analysis of a multiphase flow using the Volume-of-Fluid technique. An fsi benchmark is followed by the analysis of a bimetallic slab in a non-isothermal flow.

9.1. Test Problems

9.1.1. Linear Test Problems

Linear test problems are used to observe the performance of both stationary and nonstationary iterative methods. Jacobi and SOR are used to compare the convergence histories with and without the application of multigrid. Additionally, Newton-Krylov techniques are used to analyze selected test problems for both GMRES and BiCGStab with diagonal scaling and ilu(k) predecontioners.

9.1.1.1. Poisson's Equation. Poisson's Equation given in 9.1 is one of the simplest linear test problems. Temperature distribution is sought within a simple domain. For no source term, i.e. $\lambda = 0$, the equation is called Laplace's Equation and a solution

exist only with appropriate boundary conditions since there is no internal heating or cooling in the system. All discretizations can be tested with this problem, so sample solutions for all cases are given(fv and fe is not added yet). If the boundary conditions are all Neumann type boundary conditions (called Natural Type in FE) then there are infinitely many solutions and a constraint is needed to fix a solution. Use of the constraint is also a good test for DEMONA's constraint capability. With playing the source term, problems with known solutions can be generated as in Barrett *et al.* (1994) so results can be compared with exact solution to comment on the error. Nonlinear source terms can also be defined like in Bratu problem in section 9.1.2.2.

$$\nabla^2 T + \lambda = 0 \tag{9.1}$$

First, Newton-Krylov solver is used to solve Poisson's Equation for various source terms. Since the problem is linear, different source values does not affect the linear solver. On 129x129 grid, for $\lambda = 1$ 173 linear iterations and for $\lambda = 10$, 167 linear iterations are performed in 3 Newton steps. Since the problem is linear, Newton's Method should solve the problem in just 1 step. This unexpected behavior is because of the computational parameters of the Inexact Newton Method. Initial norm of the NK solver is determined with η_0 which is generally taken as 0.9. When one step convergence is needed, then the linear tolerance of the first NK solve should be equal to the global convergence tolerance but use of (4.19) with $\eta_0 = 0.9$ will provide a loose tolerance. Hence, it will take a couple of steps to reach convergence. In this problem, if a strict tolerance is used for linear solves (i.e 1e-5), then Newton converges in 1 step with 63 linear iterations. A constant and relatively small (1e-4) forcing parameter also requires 3 Newton steps.

In left preconditioning, the residual of the linear system is scaled with the preconditioner. If the absolute value of the residual is needed (for example in backtracking) then the residual vectors should be explicitly calculate and multiplied with the preconditioner at each step. This increases computational load. To overcome some of the difficulties, Right preconditioned solvers stop if the absolute residual norm is small

enough. For left preconditioned solvers, on the other hand, a relative convergence criteria is used. That's why, when the tolerance is set to be equal to the global tolerance, then right preconditioned solvers converge in 1 Newton step, however, left preconditioned NK solver need 2 Newton steps. That's why, while comparing BiCGStab and GMRES, only right preconditioning is used (!!! BiCGStab-RP is nor working properly, results are not included).

In Figure 9.1, convergence history of GMRES(100)-ILU(0) is given. Tolerance is set as $1e-5$. Two observations can be made. Firstly, more iterations are needed for increasing grid numbers. Secondly, as promised, GMRES gave a non-increasing residual history. For 257x257 case, one restart is made but residual continued to drop. In Figure 9.2, performance of BiCGStab with no preconditioning is presented. At same iterations, norm of the residual is increased. Irregular pattern of BiCGStab is clear for no preconditioning case.

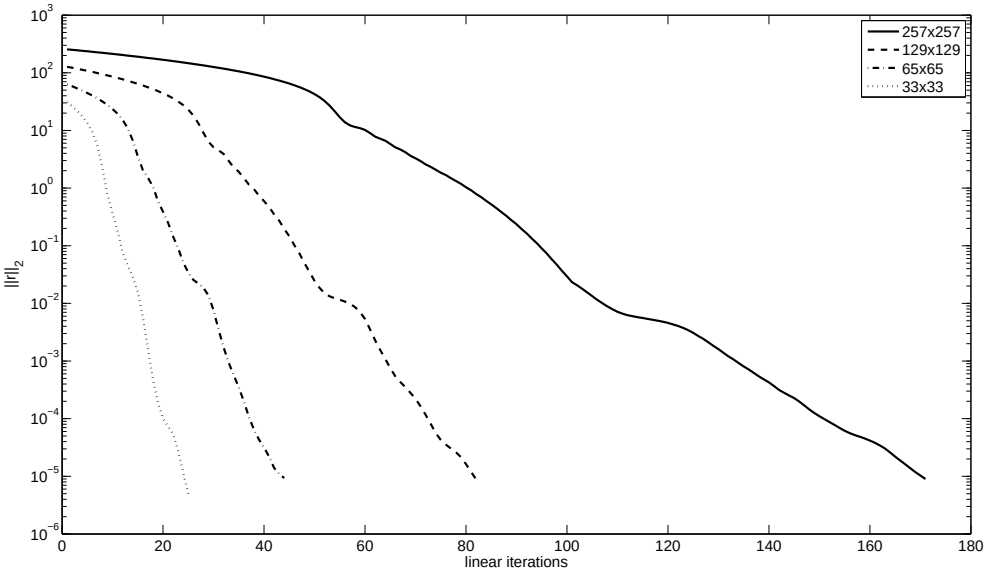


Figure 9.1. Residual history of GMRES

In Figure 9.3 V2 performance is compared against no multigrid solution. For non-mg case, damped-Jacobi needs more iterations compared to no damped case. On the other hand, multigrid improved thr convergence in both cases. Not surprisingly,

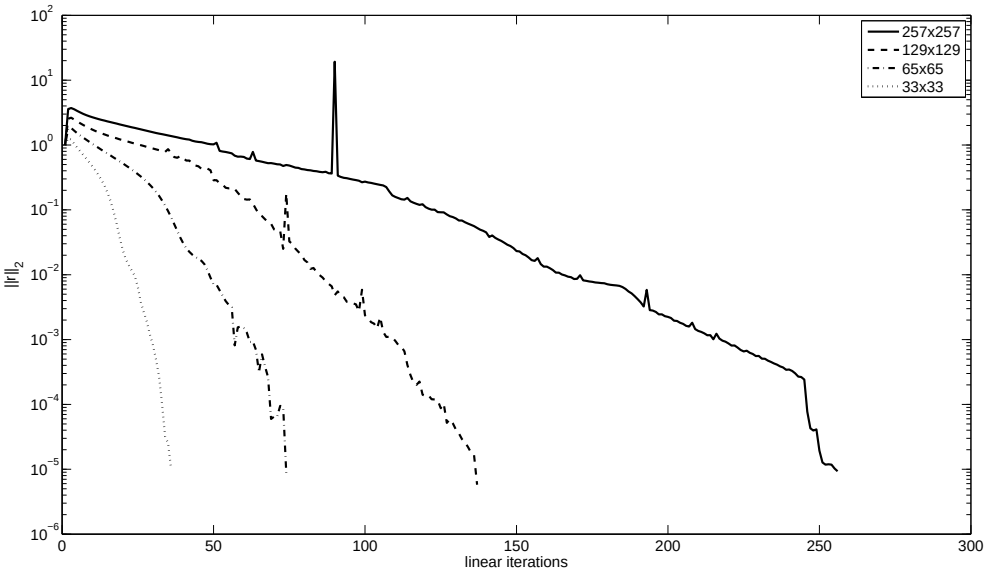


Figure 9.2. Residual history of BiCGStab

damped Jacobi showed significant improved. This is because $\omega = \frac{4}{5}$ is the optimal parameter for smoothing (Saad, 2003). FMG is also applied on the problem. FMG performance is given in table 9.1. It can been that the convergence of all grids in themselves is independent of the source term. Also, as grid size increases, number of iterations stays the same. This is one of the important features of Multigrid. An interesting point is that for 3x3, only 1 iteration is executed. This is quite natural since at that grid level, only one unknown is present. So, GS can solve that point exactly. As a matter of fact, the residual at that point is zero, so no error is carried to finer levels. That also shows that, as V cycle reach to coarsest mesh, the equation is always solved exactly. In Figure 9.4 convergence histories of different grid levels (excluding 3x3) for $\lambda = 7$ is given. Here also, the behavior on the table is verified.

Table 9.1. FMG Results for Poisson's Equation

$\lambda \setminus$ grid	129x129	65x65	33x33	17x17	9x9	5x5	3x3
1	10	10	10	10	9	7	1
3	11	11	11	11	10	8	1
5	11	12	12	12	10	8	1
7	12	12	12	12	11	8	1

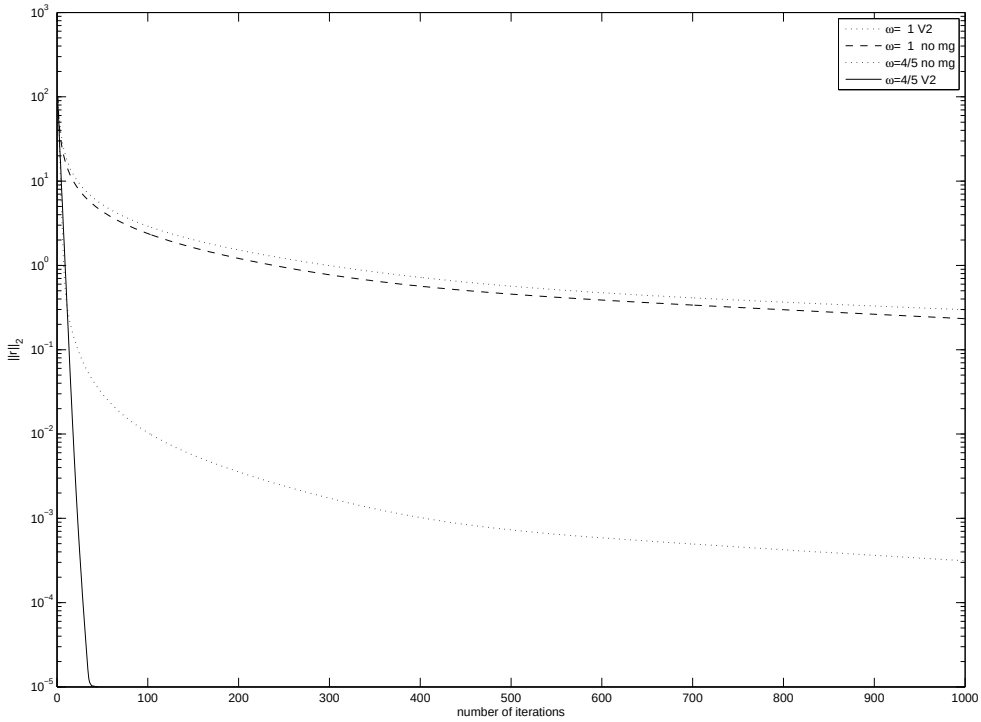


Figure 9.3. Multigrid convergence for Poisson's Problem

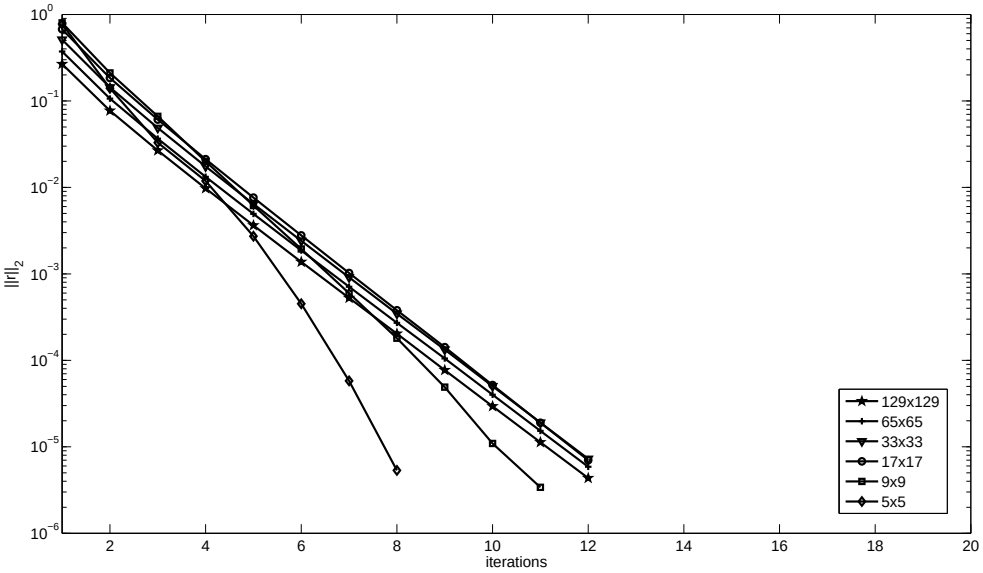


Figure 9.4. Residual norms on various grid levels for $\lambda = 7$

9.1.1.2. Hydrodynamic Lubrication. Hydrodynamics lubrication with complementarity condition changes the behavior of the problem and turns it into a nonlinear problem. As a result, it is a good test for Newton-Krylov problems and offers a smooth pass to nonlinear problems with no explicit nonlinearity defined in the Equation. Figure 9.5 shows a one-domain solution for k=1 and $\epsilon = 0.2$. Referring to two Figures 9.9 and **??** one can say that, the transition from positive pressure to negative pressure will be sharper if k and ϵ are large. Considering this fact, a two-domain adaptive solution is achieved for k=2 and $\epsilon = 0.4$ as seen in Figure 9.6. Figure 9.7 gives results for the cavity

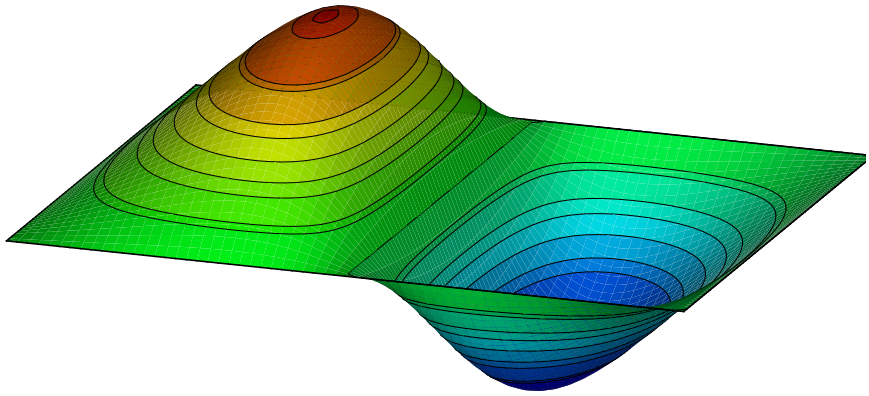


Figure 9.5. Results of the Reynolds Equation for ϵ=0.2 and k=1 without cavitation condition (grid:65x65)

condition. This condition is added to the system since the pressure cannot be negative (which is allowed before) at which the lubricant will evaporate. To resolve this problem two methods could be used (Venner and Lubrecht, 2000). First Method assumes that the negative pressure will be found on the second half of the model (obvious from Figure 9.5) so only the first half will be solved. On the other hand the second model still deals with the entire domain but sets $P = 0$ if pressure is found negative at any point on the domain. Results using the second method is given in Figure 9.6. As can be seen

from the figure, zero pressure line does not start exactly from the half of the domain but shifts a little towards right. Achieved speedup for 4 threads is 2.88 as in Figure 9.8

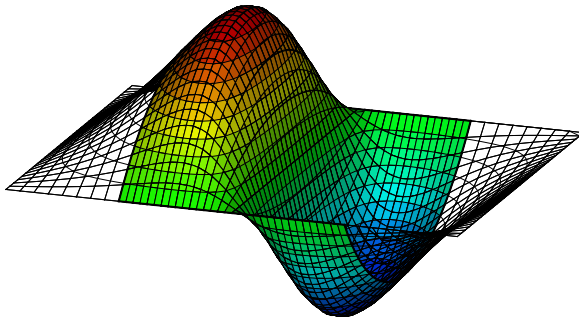


Figure 9.6. Results of the Reynolds Equation for ϵ=0.4 and k=2 without cavitation condition (grids:33x33)

FMG-FAS is used for both cases, namely cavitation and no cavitation. With increasing eccentricity, ϵ, solver performs nearly the same. A change in k on the other hand result in more fine level iterations. On the coarsest level, both problems converge just in 1 steps with no error since the problem is linear even if the complimentarity (if $P < 0$, $P = 0$) condition is activated. When Tables 9.2 and 9.3 examined, it can be said that complimentarity problem does not affect the solver much although a rise in the number of fine grid iterations is evident. It should be noted that, implementation of this constraint is harder in multigrid compared to NK framework. In Newton-Krylov Problem, setting $P = 0$ as soon as $P < 0$ is enough for the rest of the computations. On the other hand, operators should be changed in FMG. In other words, a flag should be used to define a negative pressure point and that the relaxation should be modified considering the constraint. Consequently, definition of the residual as well as $A(v^h)$ changes for that point. Finally, these changes should be performed only for the fine grid, if coarse grid corrections are equated to zero for negative values, than the problem will not converge. Residual graphs for both cases are given in Figures 9.11 and 9.12. When these figures are compared with that of Poisson's Equation, convergence of only

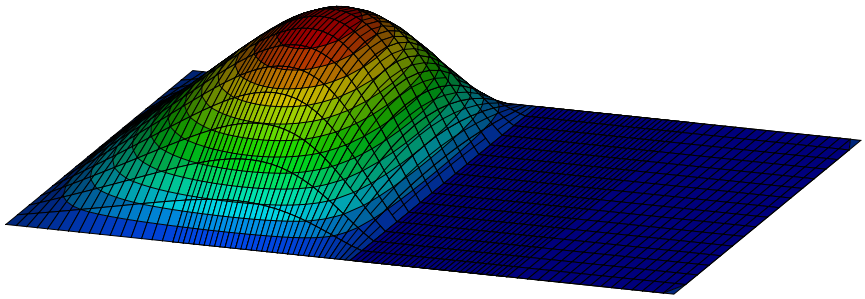


Figure 9.7. Results of the Reynolds Equation for ϵ=0.2 and k=1 with cavi- tation Results of the Reynolds Equation for ϵ=0.2 and k=1 with cavitation condition (grids:65x17)

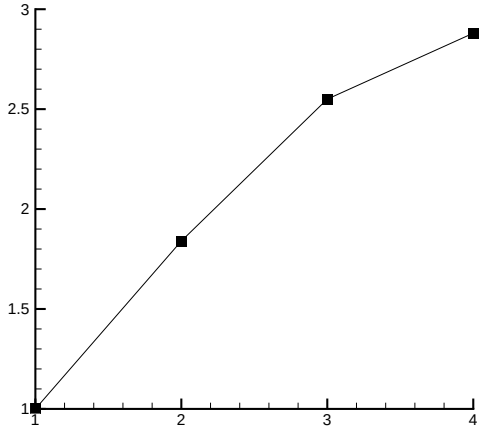


Figure 9.8. Speed-up on 91x34 grid with cavitation condition

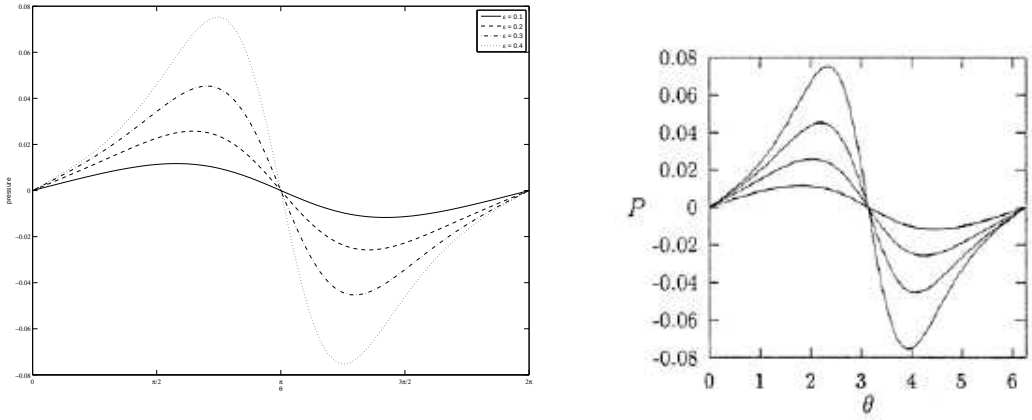


Figure 9.9. Distribution of dimensionless pressure along the center line (Y=1/2) and comparison with Venner and Lubrecht (2000)(right)

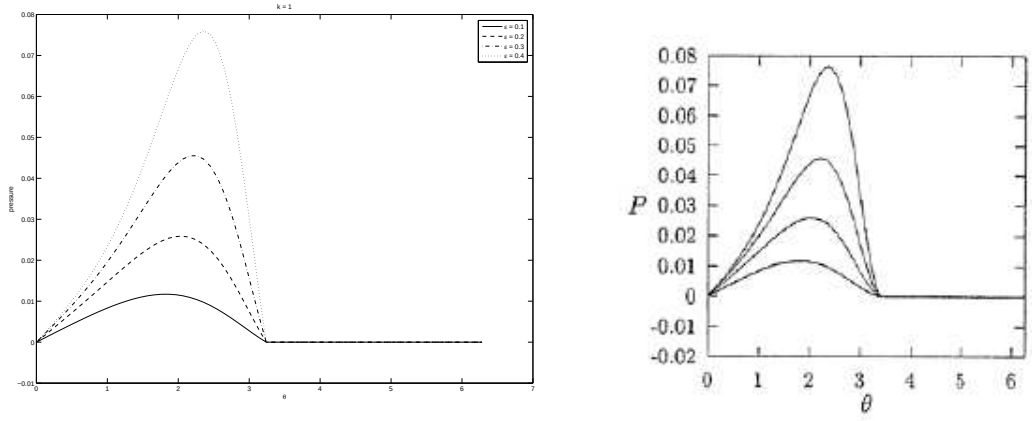


Figure 9.10. Distribution of dimensionless pressure along the center line (Y=1/2) with cavitation condition and comparison with Venner and Lubrecht (2000) (right)

final grid level do coincide. In previous problem 17x17 grid was enough to estimate the behavior of finer grids, here however that is observed only after 65x65.

Table 9.2. FMG Results for Reynolds Equation (no cavitation)

$\epsilon, k \setminus$ grid	129x129	65x65	33x33	17x17	9x9	5x5	3x3
0.1,1	10	11	11	10	9	6	1
0.2,1	12	12	12	11	10	6	1
0.1,2	31	32	29	24	17	7	1
0.2,2	35	36	33	28	19	8	1

Table 9.3. FMG Results for Reynolds Equation (with cavitation)

$\epsilon, k \setminus$ grid	129x129	65x65	33x33	17x17	9x9	5x5	3x3
0.1,1	11	12	12	11	8	5	1
0.2,1	12	13	13	12	9	6	1
0.1,2	32	33	30	23	15	4	1
0.2,2	36	36	33	25	16	4	1

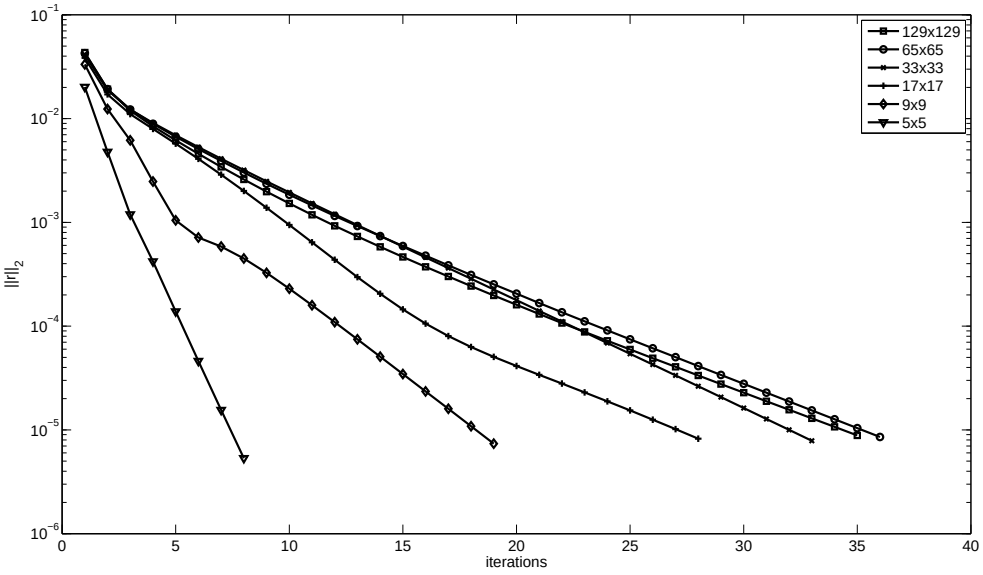


Figure 9.11. Residual norms on various grid levels (no cavitation)

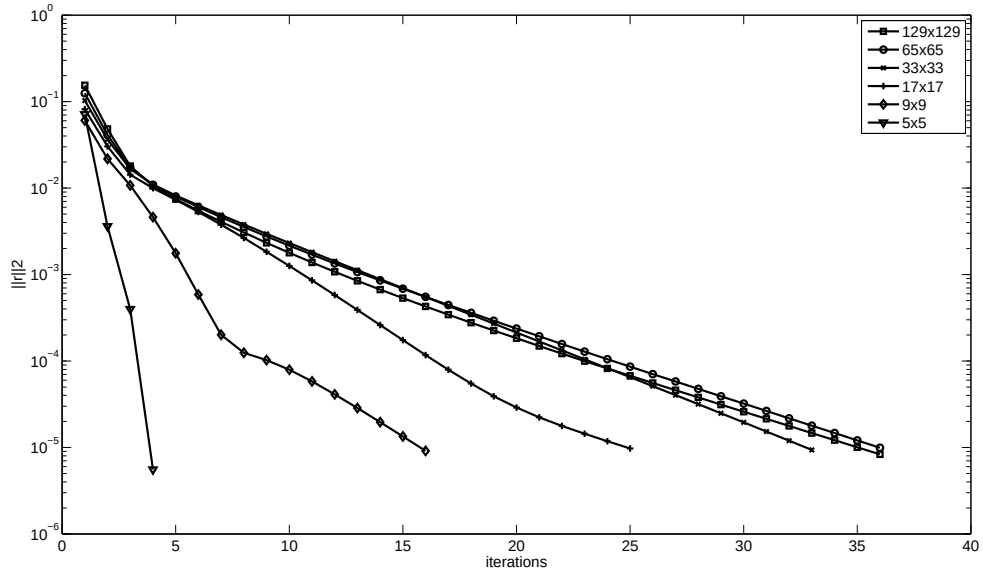


Figure 9.12. Residual norms on various grid levels (with cavitation)

9.1.2. Non-linear Test Problems

9.1.2.1. Bratu Problem. Bratu problem given in Equation 9.2 is the first real nonlinear test problem used in this thesis. It has only one unknown - temperature. It is a basic model for combustion (need ref here). Although its setup is simple, its solution is not trivial since the problem is multivalued for $\lambda \leq \lambda_{critical}$ and with no solution for $\lambda > \lambda_{critical}$.

$$\nabla^2 T + \lambda e^T = 0 \tag{9.2}$$

The form of the equation can be generalized as in Equation 9.3 to test the solver in a wide range of possibilities by varying α and λ (Brown and Saad, 1990).

$$\nabla^2 T + \alpha \frac{\partial T}{\partial x} + \lambda e^T = 0 \tag{9.3}$$

The problem is discretized with finite differences and modeled in a unit square domain. Both Newton's Method and Multigrid employed on the problem. Static adaptation is carried out to increase the resolution of the solution. Furthermore, domain decomposition is used to divide the domain into small pieces where multiplicative domain decomposition is utilized. Parallel performance with the use of OpenMP dialects is provided at the end of the section, as well where the operations in the loops are divided into threads.

Computation start with Newton-Krylov solver. Inexact Newton-GMRES(100) solver with both η choices is tried. In Table 9.4, left most number on the cell gives the number on Newton steps. Values in [ ] shows the total linear iterations for Jacobi preconditioner, ILU(0) as left and right preconditioners, respectively. In this table, only choice 1 is tabulated, it is observed that choice 2 requires more linear step. If a particular case has a different number of Newton iterations, it is denoted as a subindex of the linear iterations. As can been seen from the table, number of Newton steps is not grid size dependent, on the contrary, total number of linear iterations do increase if more unknowns are present. Both ILU(0) preconditioners behaved almost the same and

Jacobi preconditioner required more linear iterations. Success of ILU(0) is more obvious for larger λ values. The solver is also tested to compare the inexact solver with ordinary

Table 9.4. NK Results for Bratu Problem - η of (4.20)

$\lambda \setminus$ grid	129x129	65x65	33x33	17x17	9x9	5x5	3x3
1	3 [424|140|148]	3 [216|79| 84]	3 [109|46|44]	3 [54|27|26]	2 [14| 9|10]	2 [6| 6| 6]	2 [2|2|2]
3	3 [558_4|202|219]	4 [287|108|109]	3 [99|39|38]	3 [50|23|23]	3 [23|13|14]	3 [9| 8| 9]	3 [3|3|3]
5	4 [537|187|189]	4 [272|103|102]	4 [117|56|54]	4 [69|31|30]	4 [30|19|20]	3 [9|11| 8]	4 [4|4|4]
6	5 [552|233|239]	5 [333|137|132]	4 [160_5|50|50]	4 [65|28|29]	4 [30|17|18]	4 [12|10|12]	fail
6.8	7 [931|351|453_8]	7 [488|180|177]	7 [170|97|99]	8 [138|64|64]	fail	fail	fail

solver i.e. against constant linear tolerance for each Newton iterations. Computations are carried out for 129x129 case for $\lambda = 6.8$ with ILU(0) as right preconditioner. If linear tolerance is kept as $1e-5$ at each step, 680 total linear iterations are needed. If eta is kept constant as $\eta = 1e-4$ then 548 linear iterations are performed. Choice 2 on the other hand used 458 total linear iterations. Although each problem converges in 8 nonlinear steps, only inexact Newton with forcing parameter of Equation 4.20 requires less linear iterations.

Continuation over λ is also considered only with bootstrapping. Same values on the table are selected as the parameters. As observed from Figure 9.13 number of linear iterations are reduced in general, however, there is no significant decrease. Only at the final solution, 7 Newton iterations are performed instead of 8. Figure also tells us that the continuation should be handled more carefully such that reaching the final case needs less computational load in previous cases.

FMG-FAS solution are performed for a wide range of λ values as in Table 9.5. One observation is that, after first couple of coarse grids, finer grids converge in same number of iterations. This verifies that mesh size free convergence can be attained. For relatively small λ values, coarse grids are extremely useful, unfortunately as λ increases, coarse grids becomes useless since there is no solution for the discrete problems at those levels. If a coarse grid fails to converge, than a finer grid requires more iterations since the initial guess is not improved. Furthermore, as problem get stiffer, the number of iterations to satisfy the convergence criteria also increases. That's why for $\lambda = 6.8 \approx \lambda_c$ case, 17x17 grid needs around 7000 iterations. In Figure 9.14 convergence history of

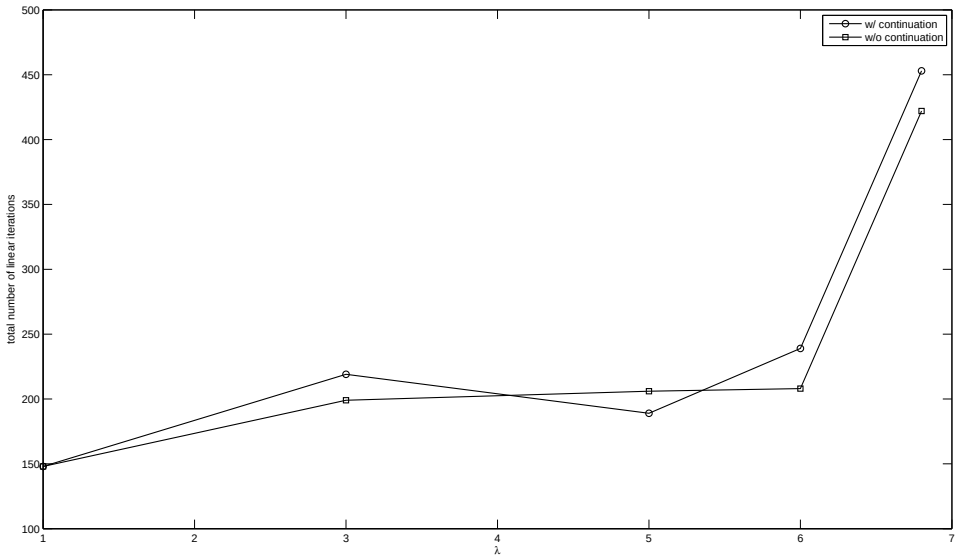


Figure 9.13. Continuation comparison

Table 9.5. FMG Results for Bratu Problem

$\lambda \setminus$ grid	129x129	65x65	33x33	17x17	9x9	5x5	3x3
1	10	10	10	10	9	8	2
3	11	11	11	11	10	8	3
5	11	12	12	12	11	8	4
6	11	12	12	12	11	40	fail
6.8	12	13	89	7039	fail	fail	fail

$\lambda = 6$ is given. It is clear from the figure that starting with grid 33x33, all finer grids come to a stop in about same number of iterations. Please note, that does not mean that all of the solutions finishes in same amount of time. Iterations are the same but obviously a finer grid will be executed in more time. In Poisson's Equation, coarsest grid could be solved exactly. In Bratu problem, one Newton step seems to be not enough to solved the problems exactly. As we observe the figure, we can say guess that 2 local Newton's are needed for $\lambda = 1$ case and 4 steps for $\lambda = 4$ case. When these changes made, it is observed that the coarsest grid performed only one global iteration but the residuals are not exactly zero (4.12e-9, 3.58e-11 respectively). Actually, these values are same as 1 local step iterations! As a result, increasing number of Newton steps for all points of the domain seems to be a waste of computation for this problem (On the other, it pays off later for more nonlinear problems like lid driven cavity problem). Of course, it would be a good idea to decide on local Newton steps. Normally, the residual might be calculated as soon as the point is updated. However, checking every point locally after each GS sweep is also time consuming. In that sense, deciding on the part of the domains that require more accuracy and applying local fine grids seems more reasonable.

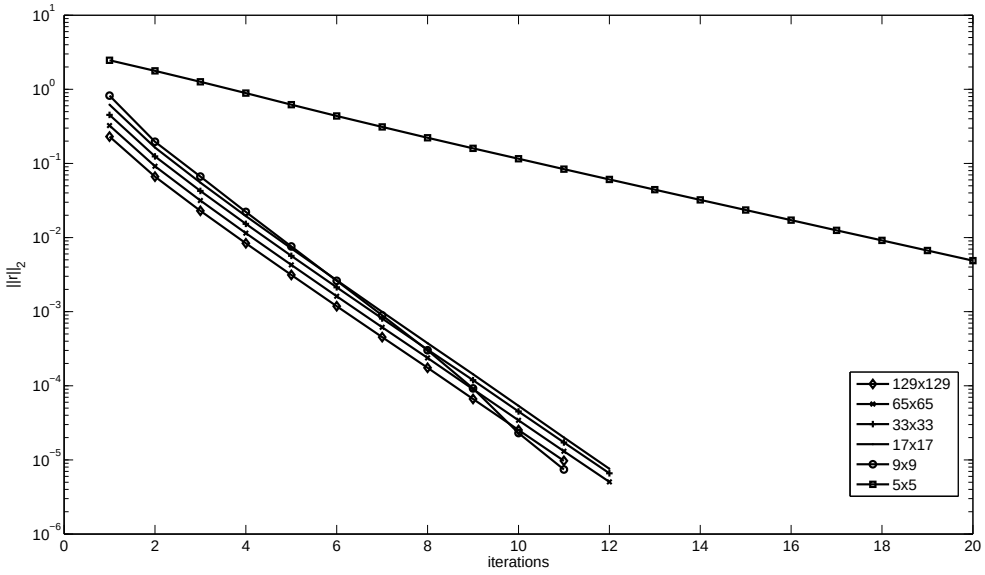


Figure 9.14. Residual norms on various grid levels for $\lambda = 6$

Static adaptation is performed for $\lambda = 6.8$ case. Two 33x33 grids are used, one is global and the other one is local. The solution is plotted in Figure 9.15. Coarse domain covers the entire domain where the local grid solution is used to improve to solution at the center to read the maximum temperature value.

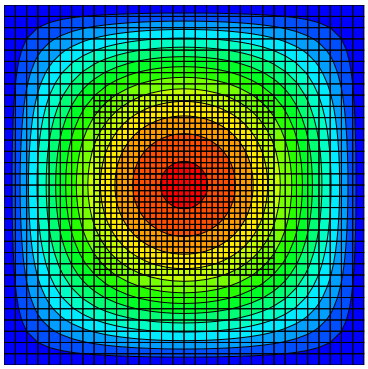


Figure 9.15. Results of the Bratu problem for λ=6.8

Next figure gives some details about the global and local solves. It is observed that solutions match at the interface and smoother in the fine region.

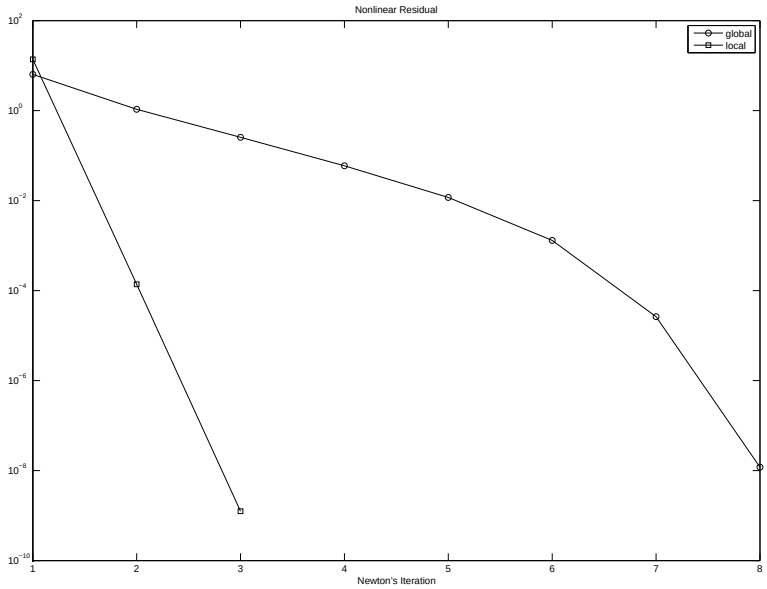


Figure 9.16. Nonlinear residuals

Parallel computations based on OpenMP is tested for Bratu Problem. Speed-up results are given in Figure 9.17. Intel Fortran Compiler 10.1 is used on Windows XP 64 bit with optimization flags $/O2$ and $/QxT$. Speed-up is calculated using $S(N) = \frac{T(1)}{T(N)}$. Maximum Speedup is achieved with 4 threads. The calculations are based on NK solver. Parallelization is performed while evaluating $F(x)$.

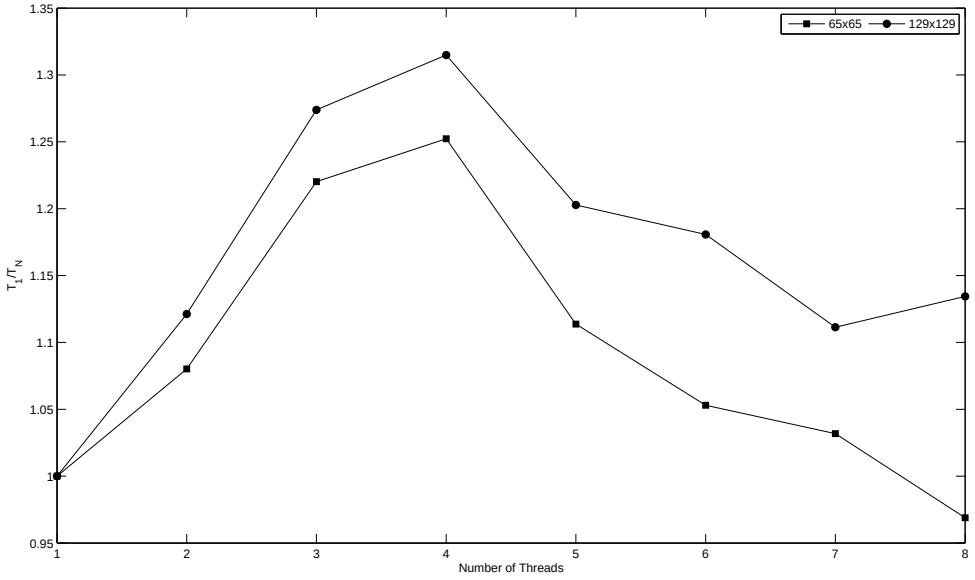


Figure 9.17. Speed-up on different grids

9.1.2.2. Lid Driven Cavity with SF-VOR Formulation. Lid driven cavity problem is the one of the most frequently used benchmark problem in computational fluid dynamics. Ghia *et al.* (1982) analyzed the problem with $\psi - \omega$ formulation using FAS. They used Strong Implicit Procedure as the smoother. Results up to $Re = 10000$ are presented so most of the studies uses the work as a reference. In this thesis, comparison with this study is also made. Equation set 2.6 is discretized with finite differences. Thom's formula is used for vorticity boundary conditions. The test are made on the basis of Re number as well as the grid sizes. An important point is that when line search backtracking is used, then maximum number of Newton iterations should be set to a large number like 200. It is already mentioned that even for $Re = 100$ on 33x33 grid required 87 Newton step for zero initial guess (Figure 4.2). If 200 iterations

is reached then the solver stops even there is no convergence. For those problems, the solution is restarted with improved initial guesses. The grid sizes are important in analyzing convective dominated flows. A minimum grid size should be selecting so that the physics of the problem is reflected. For very coarse meshes, the discrete problems might not have a solution. In this problem, 17x17 is selected as the coarsest mesh. Newton-GMRES(100) with ILU(0) as right preconditioner is used. Numerical Jacobian is evaluated coloring based. Choice 1 with safeguarding is used and $\eta_0 = 0.9$. As stopping criteria (4.17) with $tol_{rel} = 1e-8$, $tol_{abs} = 1e-5$ is preferred.

From Table 9.8 one can see that $Re = 1000$ case cannot converge in given 200 iterations. It would have converged if the maximum Newton iterations kept higher, however, a nonlinear solver should also find a solution in a reasonable period of time so we should not wait too long to find a solution. As a matter of fact, continuation is quite effective as in table 9.8. Number of Newton iterations are significantly reduced. For no continuation cases, line search backtracking is successful to prevent divergence. When the iterate is close enough to the exact solution, quadratic convergence is achieved. Convergence histories are given in Figure 9.18. For continuation, line search backtracking is never needed except grids 33x33 and 17x17. Those grids could not give a result since for $RE = 1000$, grid resolution is not enough to capture the physics. Hence, backtracking algorithm tries to find a solution but cannot get anything.

Table 9.6. FMG Results for SF-VOR Problem

$Re \setminus$ grid	129x129	65x65	33x33	17x17
1	11 (2026)	10 (411)	8 (151)	8 (54)
10	23 (6215)	25 (1743)	24 (751)	20 (320)
100	137 (58995)	105 (11014)	84 (3867)	64 (1192)
1000	200F (153023)	200F (30797)	200F (10807)	200F (4822)

FAS-FMG performance is listed in Table 9.8. Same idea as in NK case is also followed here. Minimum grid is set to 17x17. Furthermore, fine grid is prescribed as two levels higher. This is called short-FMG (Drikakis and Rider, 2005). The reason, while using

Table 9.7. FMG Results for SF-VOR Problem with continuation

$Re \setminus$ grid	129x129	65x65	33x33	17x17
1	11 (2026)	10 (411)	8 (151)	8 (54)
10	3 (426)	3 (191)	3 (83)	3 (41)
100	4 (1312)	5 (391)	5 (159)	4 (66)
1000	13 (15690)	34 (25458)	200F (18107)	200F (11138)

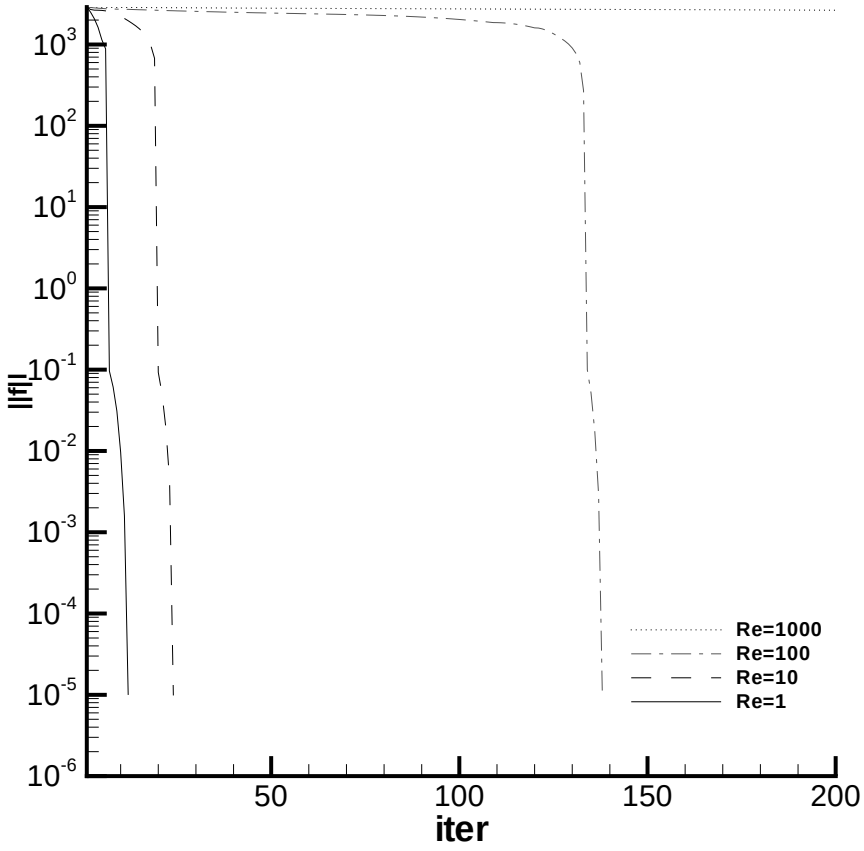


Figure 9.18. Nonlinear residual history

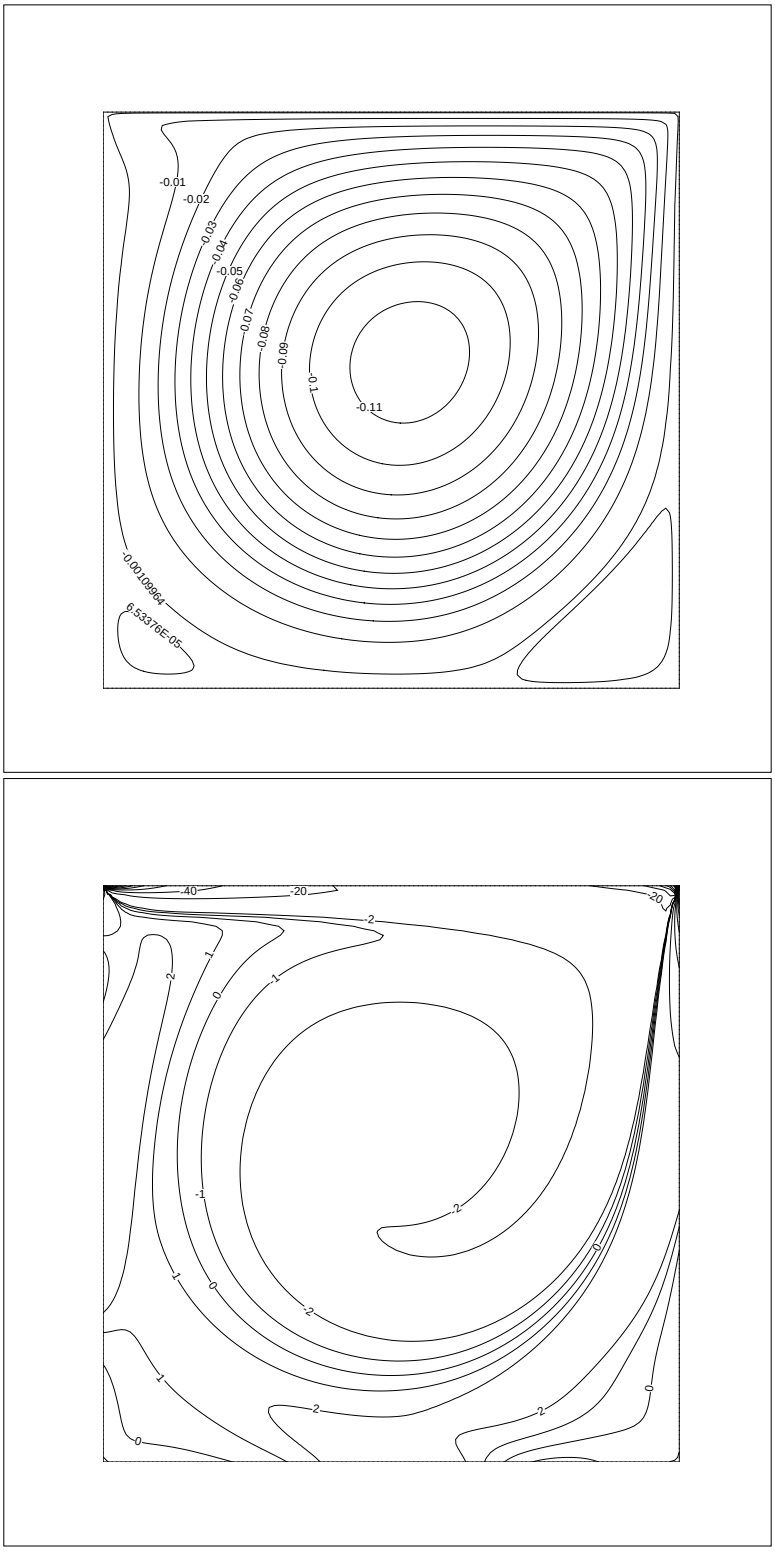


Figure 9.19. Stream function and vorticity for Re=1000

coarse grid correction, is that the phyiscs on the coarsest level will be different on the finest level. For example, cell Peclet numbers will be different. So, if the coarsest mesh is too coarse for the finest level, then Multigrid does not converge and it stalls. In Figure 9.20 convergence history of $Re = 100$ case is given. Since coarsest mesh is relatively fine, more than 200 iterations are needed for convergence. For convective dominated problems, it might be a good idea to reduce the tolerance of the coarsest mesh as suggested by Vanka (1986). In this problem, tolerance of the 17x17 grid is reduced by one order and number of maximum linear iterations are increased.

Table 9.8. FMG Results for SF-VOR Problem

$Re \setminus$ grid	129x129	65x65	33x33	17x17
1	stall	44	41	212
10	stall	44	42	213
100	stall	48	48	239
100	50	48	918	x
400 ($\omega = 0.5$)	stall	87	3657	fail

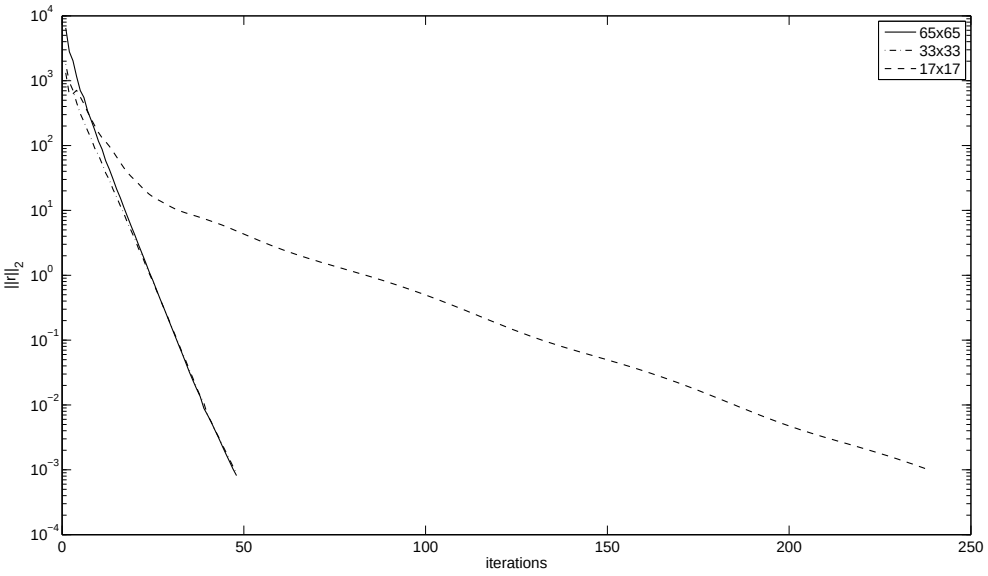


Figure 9.20. Residual norms on various grid levels for $Re = 100$

Domain decomposition idea, with static adaptive patches are also tested for Re=500 to get the result depicted in Figure 9.21. In this sample run, one global domain, four subdomain plus an additional subdomain at right bottom corner is used. The patches are added to the corners in a predefined manner since it is known that the vortex formation is critical at those corners. When Figure 9.22 is examined, it could be found out that there is a tiny vortex at the corner. Mathematically speaking, one can expect infinitely many vortices at the corner, as the grid is refined in such a way that the value of the stream function is decreased and approaches to zero. Physically, however, this is not true from two different aspects. Firstly, as grid is refined, the mesh size will not be compatible with the governing equations since the continuum assumption is compelled. Secondly, the computer model is free of impurities, yet the domain's surface will not be smooth and the vortex formation will be damped in reality.

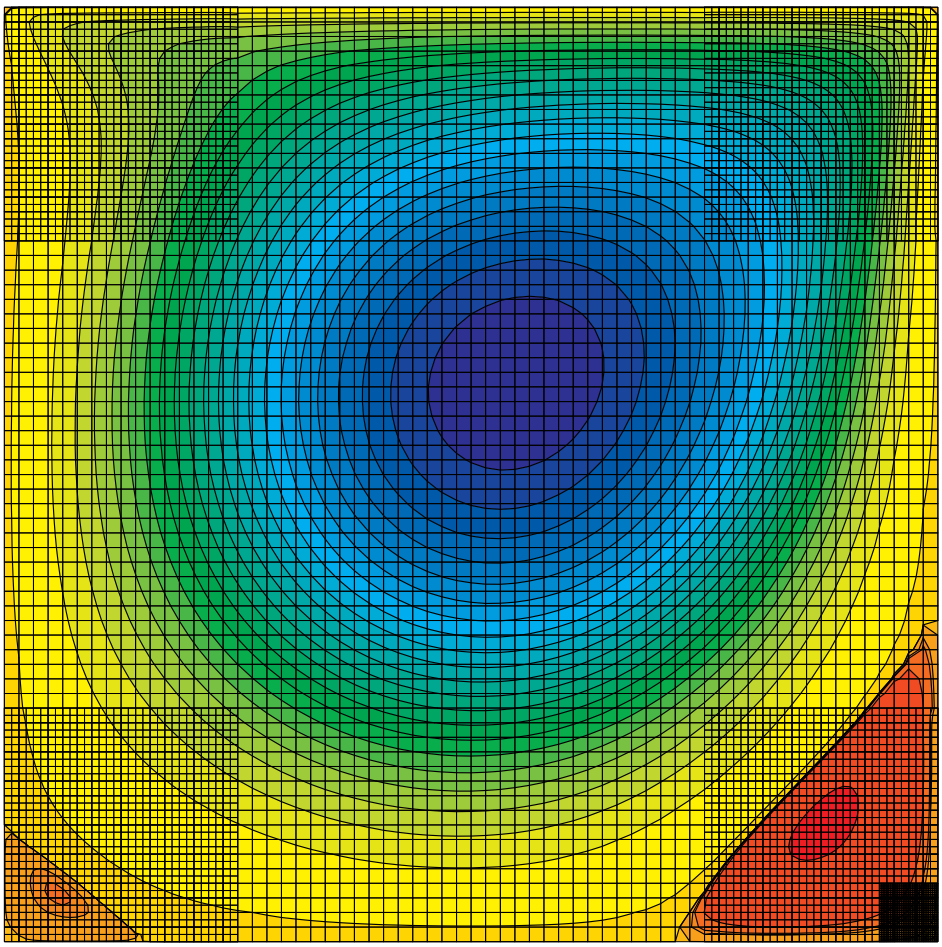


Figure 9.21. Stream function for $Re = 500$

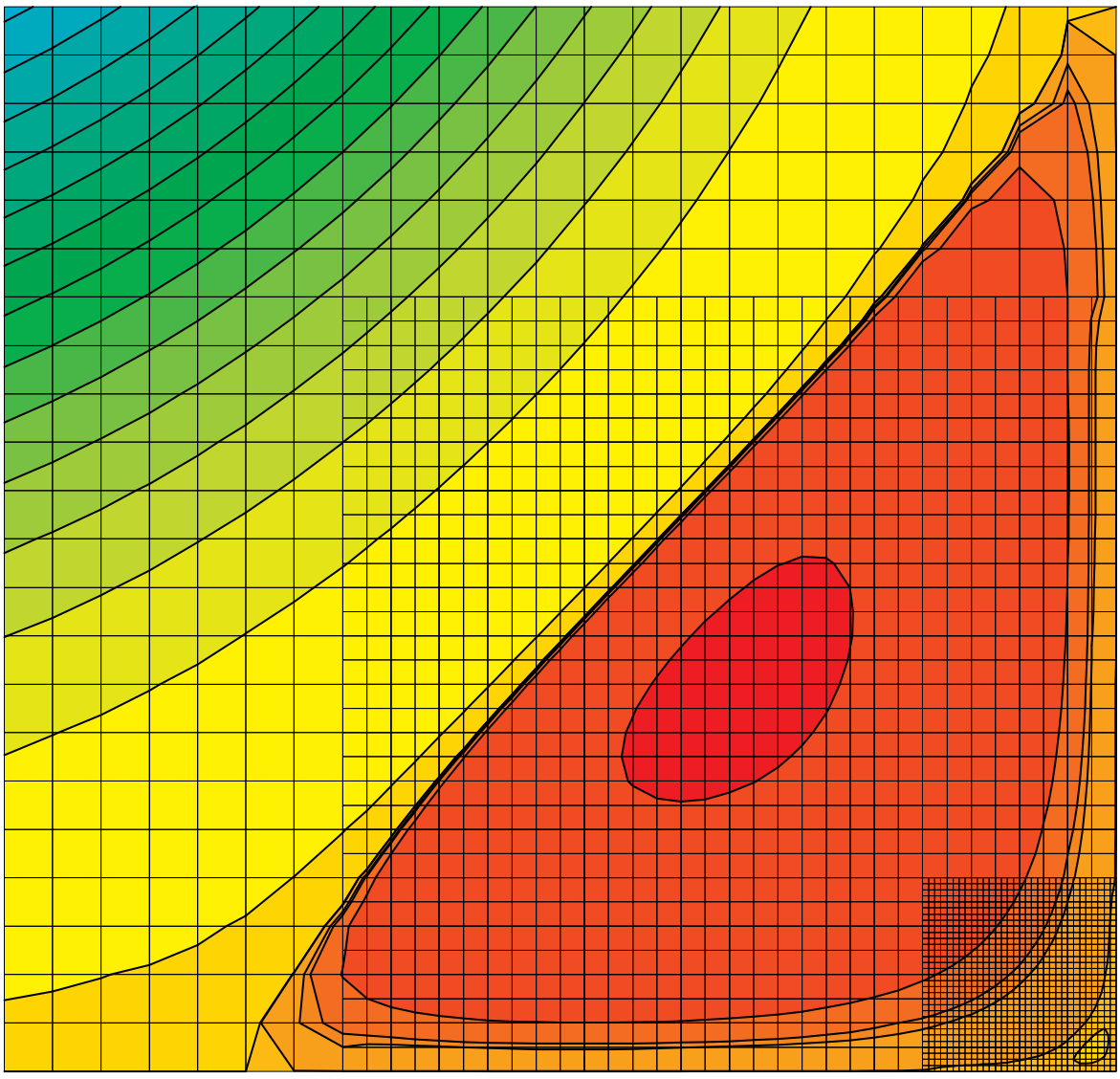


Figure 9.22. Stream function for $Re = 500$ at right bottom corner

9.1.2.3. Lid Driven Cavity with AC Formulation. Lid driven cavity problem is also solved with AC formulation. Both finite difference and finite volume are used but oscillatory pressure field is smoother in cell centered finite volume discretization so computations are performed for that formulation. QUICK upwinding is used for convective terms so two layers of ghost cells are implemented out of the solution domain. It is already discussed that the performance of the solver is dependent on the artificial compressibility parameter β. That's why a parametric study is done before proceeding.

Parametric study on β is also performed. In order to understand the convergence behaviour of AC based on β as well as $d\tau$, numerical experiments are made. Grid sizes are selected as 33x33,65x65 and 129x129. An optimum β is defined as the parameters that let the solver converge in less pseudotime iterations. One of the important observations of the study is that at the optimized *beta* value, number of time iterations are the same for all grid sizes. On the other hand, optimum β is different for meshes. In Figure 9.23 variation of time iterations with respect to $d\tau$ is given. One outcome is that for small $d\tau$ values, more time steps are needed. Another result is that as Re increases, more time steps are needed, as well (similar trend is also observed for $Re = 10, 1000$, however, some of the run's are missing). It is also understood that, for every beta value larger than the optimum β, number of time iterations stays the same. This is an important observation. In order words, the optimum β is actually β_{min} of that particular grid, RE and $d\tau$. Final observation is that, although number of time iterations are increased for increasing RE number, the value of β_{min} decreases. This is demonstrated in Figure 9.24.

Computations up to $Re = 1000$ are performed and validated with Ghia *et al.* (1982) as in Figure 9.1.2.3. Velocity boundaries are given on the cell faces as the averages of the nearest cell-center values whereas the pressure conditions are taken as homogenous Neumann boundary conditions. Time history of the steady AC problem is given in Figure 9.1.2.3. In these finite volume computations, the α-diagonal idea is used to test the solver (appendix B). When the problem is preconditioned with the correct diagonal, then the time history is smoother towards convergence but more run-time is needed because of the computation of the jacobian through the nonlinear

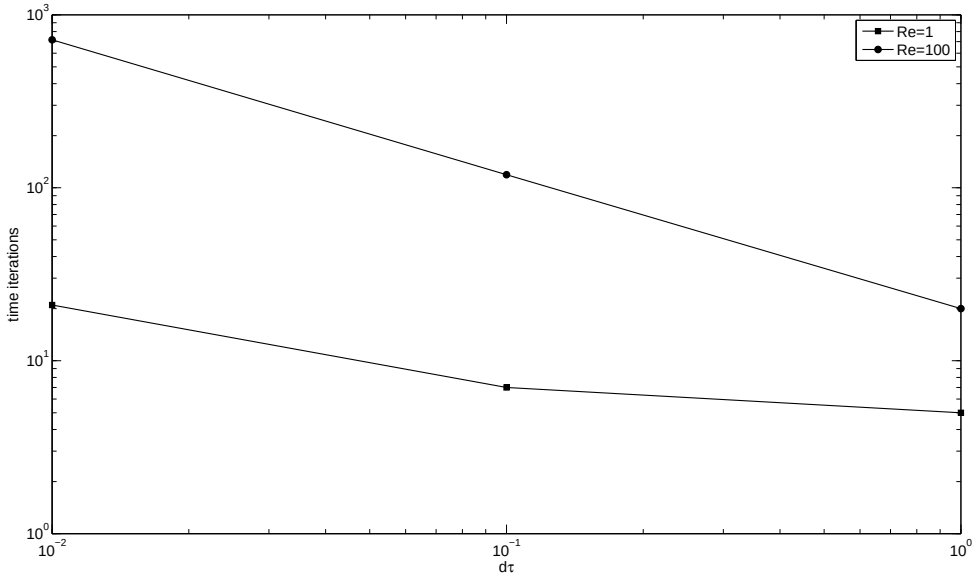


Figure 9.23. time iterations vs. $d\tau$

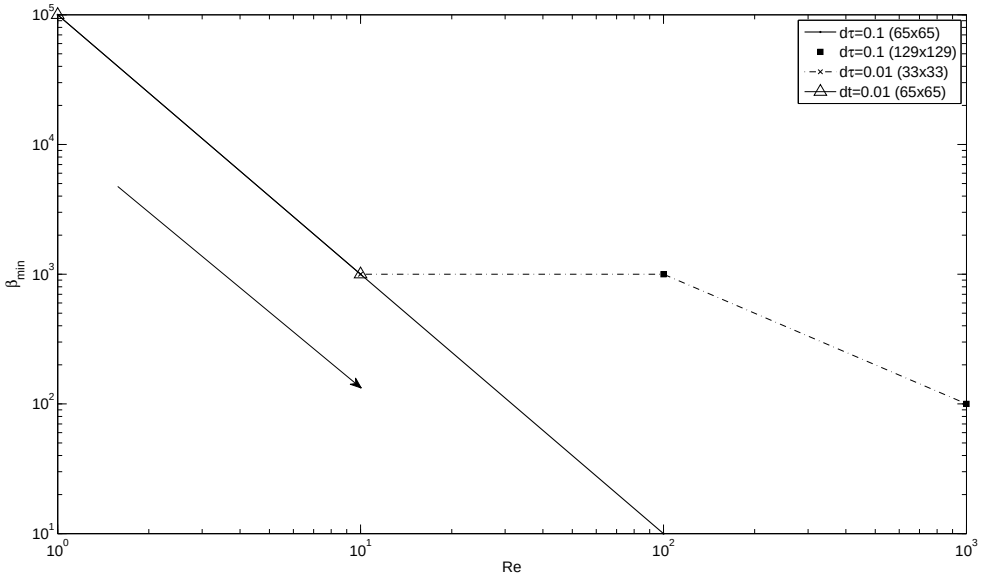


Figure 9.24. β_{min} vs. Re

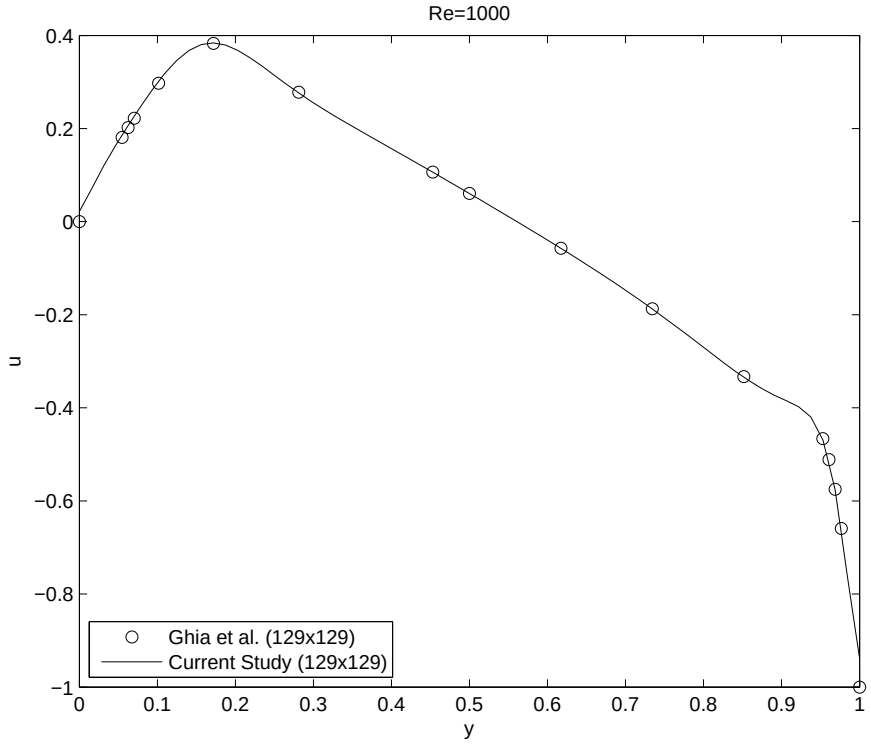


Figure 9.25. Comparison of the u-velocity on y=0.5 line

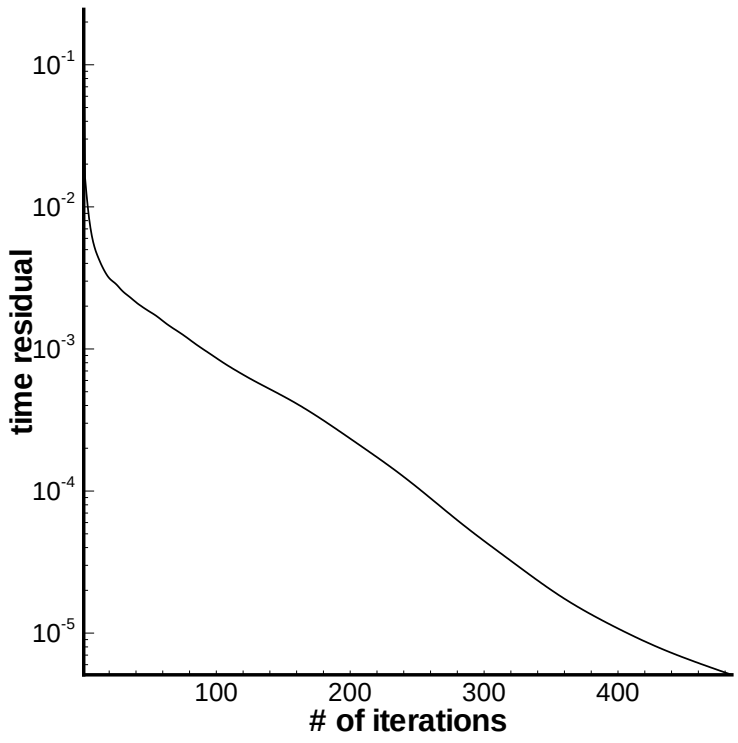


Figure 9.26. time history

function. If $\alpha = 0.5$ is used, then the time history smears towards the steady state but converges. The results in both cases are the same but the α case runs way faster compared to the original system. Of course, with the implementation of the color-based jacobian evaluation, the original problem can also been accelerated.

9.1.3. Transient Test Problems

A Benchmark problem on transient lid driven cavity is presented by Mohamad (1998). In this section, transient results are compared with this reference. In Figure 9.27, the history of u velocity on different y locations are given. In this problem, the lid is oscillating with a frequency of π. $Re = 500$.

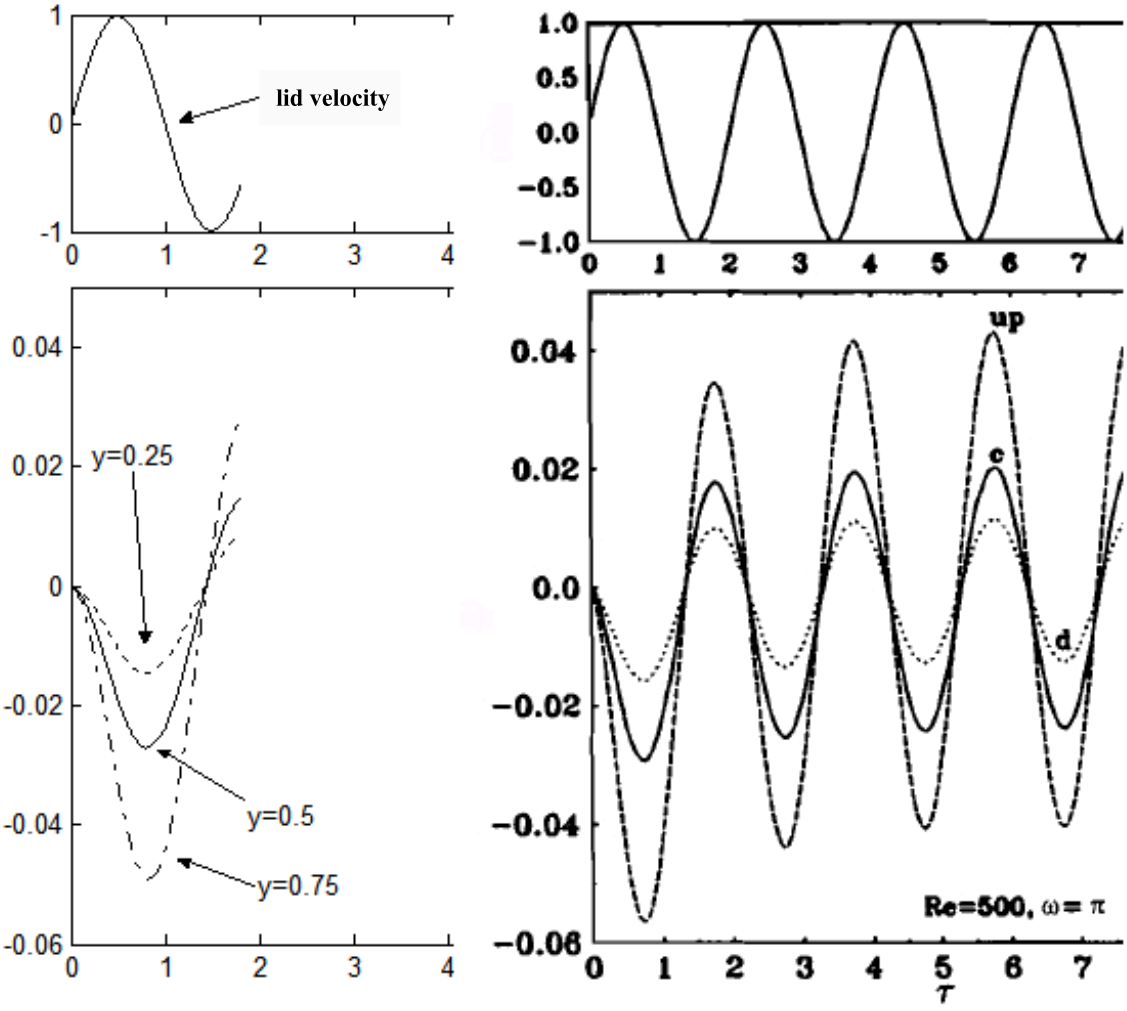


Figure 9.27. u velocity on various locations

9.2. Finite Element Computations

In this thesis, an application problem based on a thermoelasticity problem is analyzed (section 9.4.1). Decoupled formulation in thermoelasticity requires the solution of a heat conduction problem as well as the solution of the deformation on the body. For that purpose, a finite element solver, yavru[14] , is also developed. The solver is tested on several problems and compared against the result of ANSYS.

In this section, some results on basic heat conduction problems are presented. Figures 9.31 and 9.28 are two examples compared with test problems VM98 and VM100, respectively (Ansys, 2004). In Figure 9.31, a simple heat transfer problem in a tapered fin is solved where T_{wall} bc is 1100 F and T_∞ is 100 F. Figure 9.28 presents the results of a chimney problem where the inner temperature is 100 F and outer temperature is 0 F. Number of elements are kept small to compliy with the benchmark problems. However, the results are in accordance with Ansys' computations.

Additional computations besides these two test problems are given in Figures 9.34, 9.32, 9.33. In those results, calculations on complex geometries with unstructured triangular and quadrilateral elements presented. Figure 9.34 is a 2D model of a heatsink mounted on a CPU. The CPU assumed to be at 100 C at all times. The temperature distribution on the solid with $T_\infty = 10$ C is plotted. In this model, only triangular elements are used. On the other hand, Figure 9.32 is solved with unstructured quadrilateral elements. As observed from the figure, although the mesh is not symmetric, the contour lines are relatively smooth and symmetric. Last two figures are on a sample problem generated by the union of four circles. Figure 9.33 presents the temperature distribution of the complex geometry whereas Figure 9.35 compares the results of yavru with ANSYS.

[14] called yavru because it is not well developed to test advanced problems in finite elements

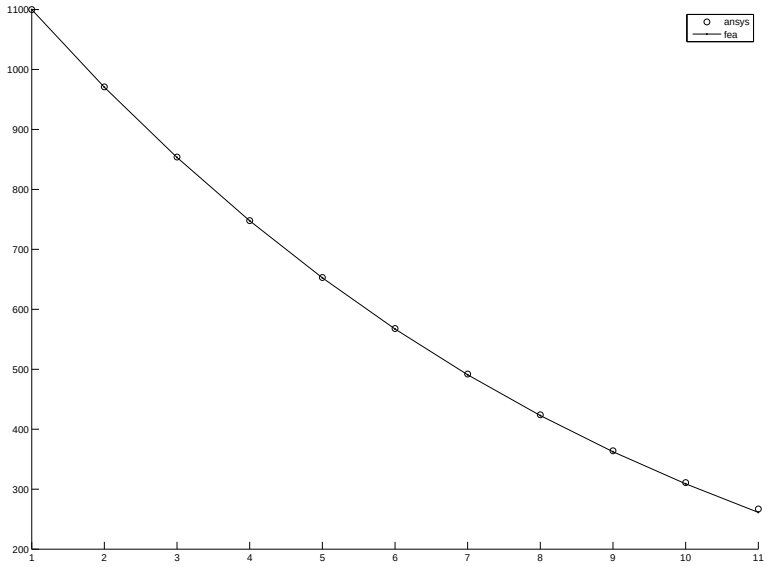


Figure 9.28. Comparison of Benchmark VM98 with ANSYS

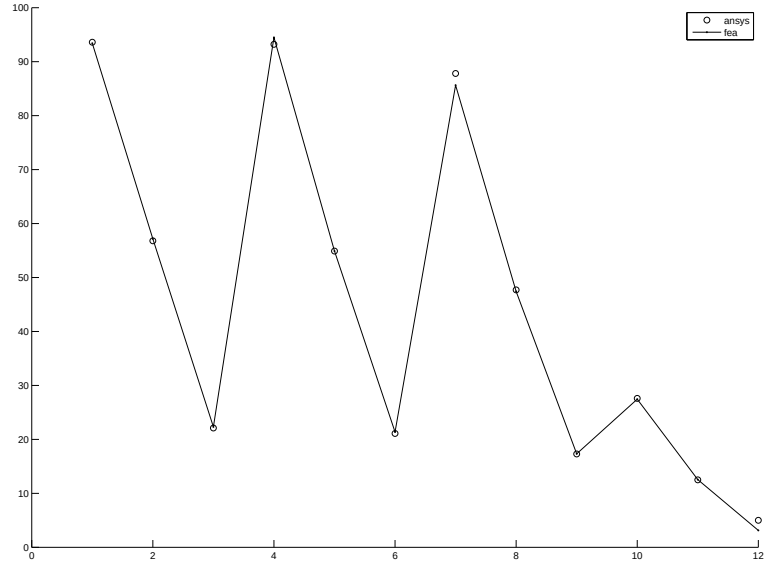


Figure 9.29. Comparison of Benchmark VM100 with ANSYS

Figure 9.30. Temperature distribution for VM100

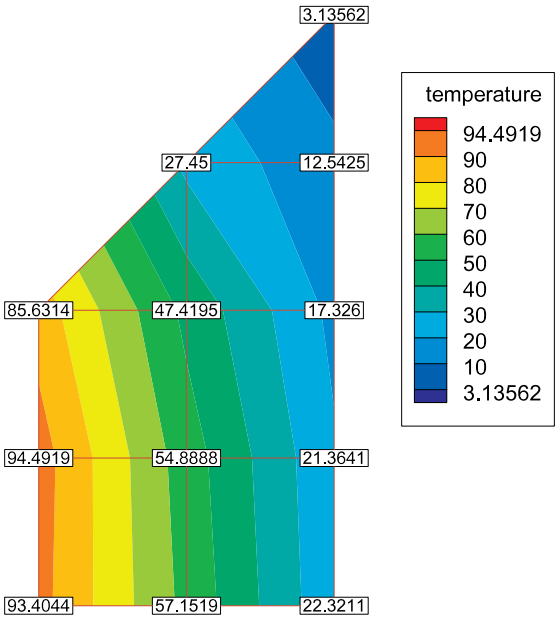


Figure 9.31. Temperature distribution for VM100

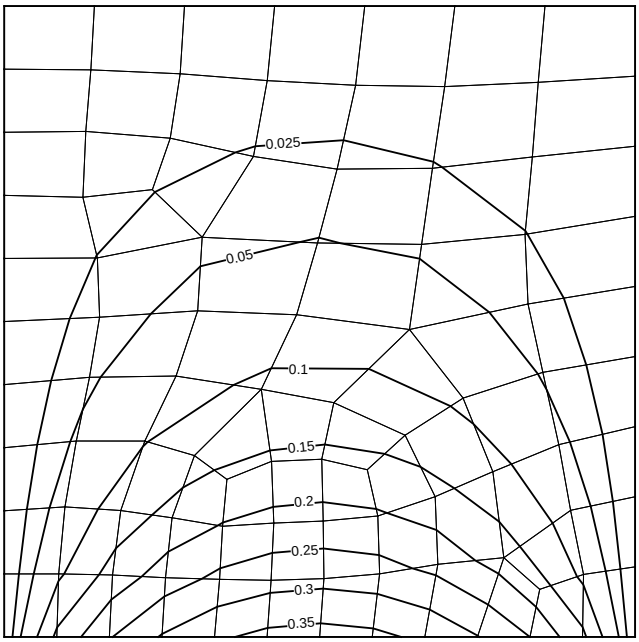


Figure 9.32. Temperature field on unit square. Homogenous Dirichlet on three sides. $q'' = 1W/m^2$ at the bottom

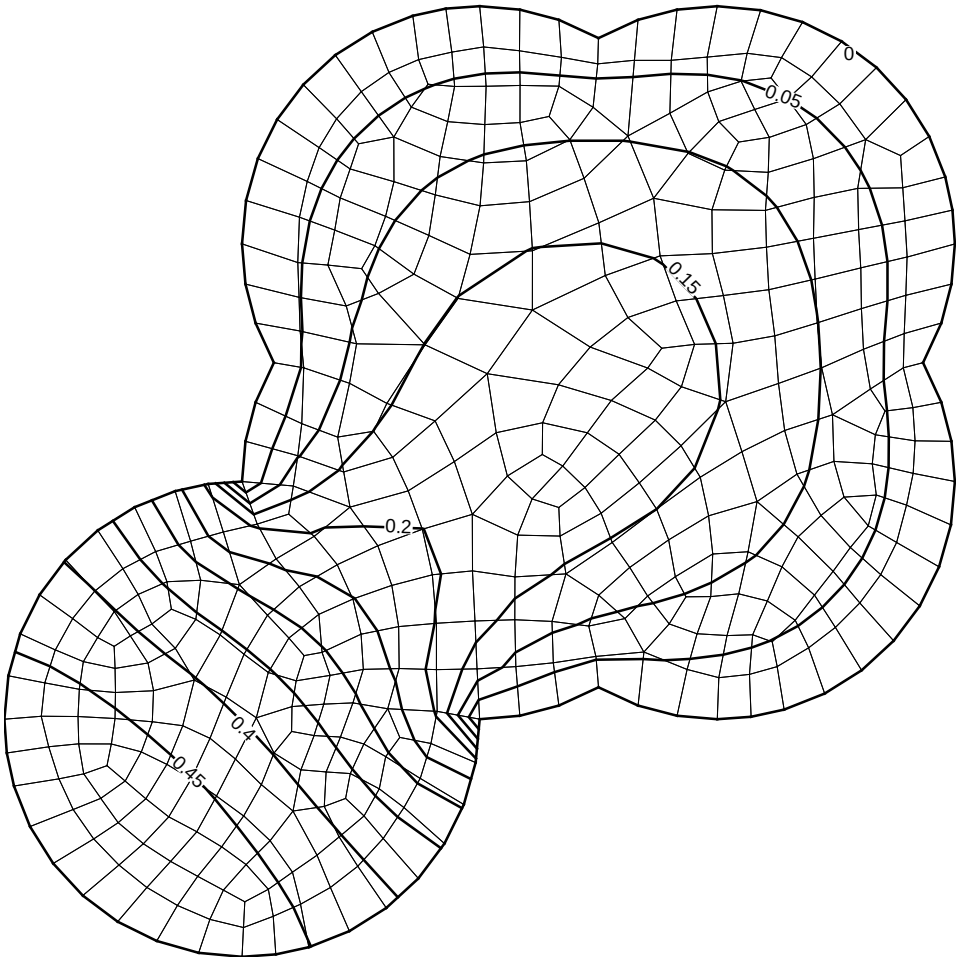


Figure 9.33. Temperature field on a complex domain. Homogenous Dirichlet on long Perimeter. Insulation on short side. Uniform source, $Q = 1W/m^3$ all over the domain

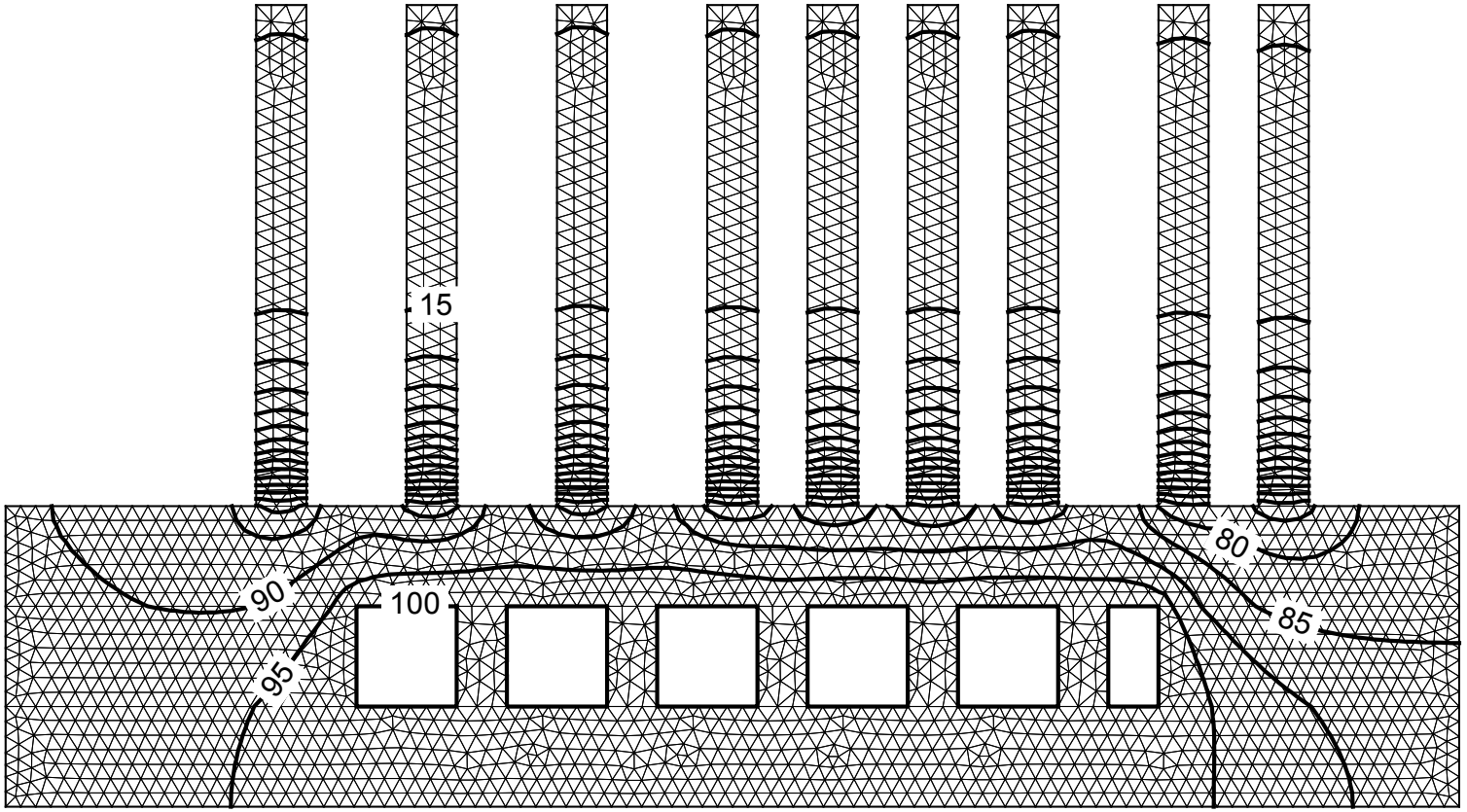


Figure 9.34. Temperature field on a heat sink. Convection BC over the fins, $T = 10^oC$, h=5 W/m^2K. $T = 100^oC$ on the source.

With 3148 elements

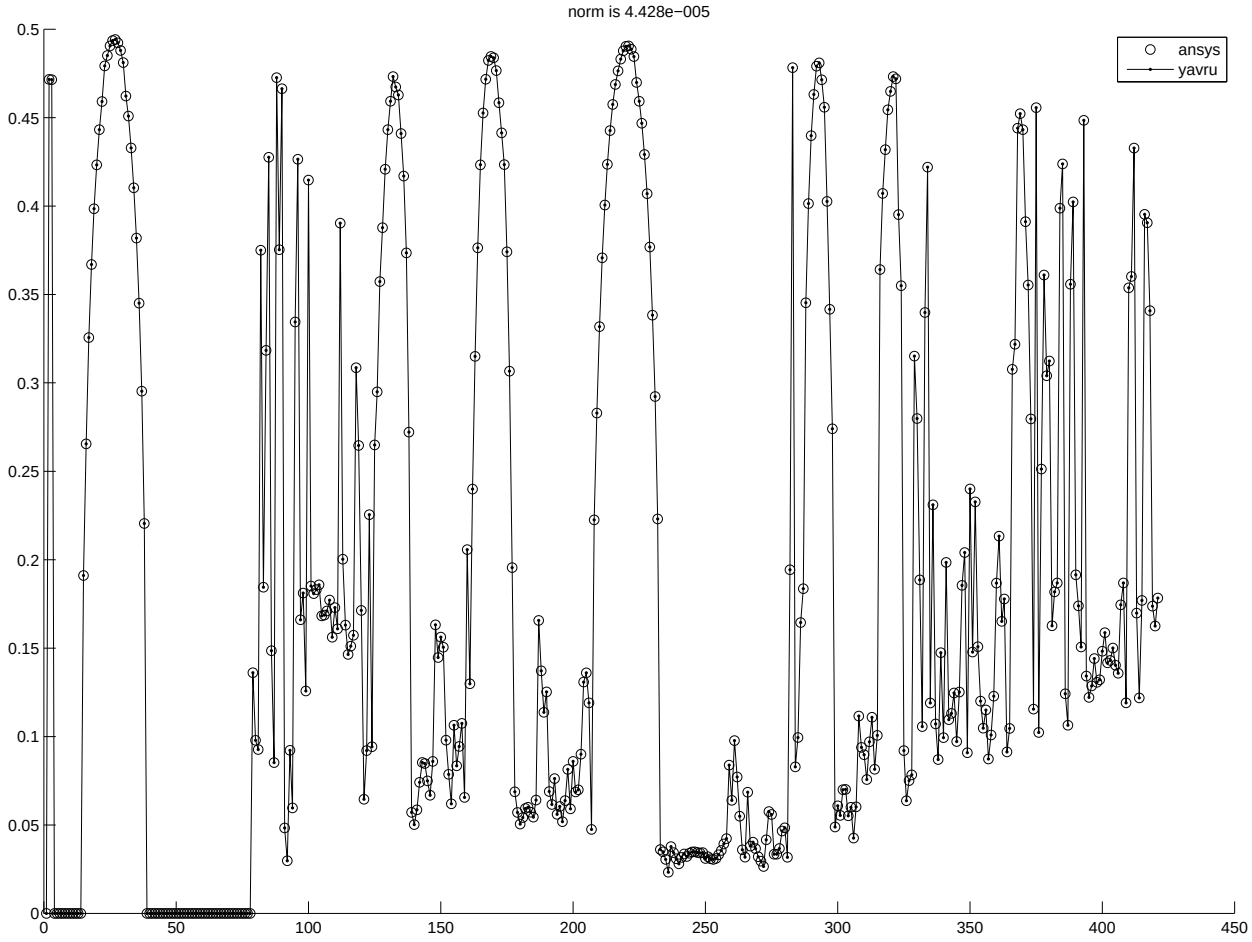


Figure 9.35. Comparison of the result in figure 9.33 with ANSYS

9.3. Set Benchmark

A one-dimensional problem is selected to examine the behavior of set selections. Test problem is taken from Lanzkron *et al.* (1996) which reads as in Equation 9.4[15] with boundary conditions $u(0) = u(1) = 0$. The exact solution is given in Equation 9.5.

$$-u'' + u^3 + \left(4 \times 10^8 \left(x - 0.5\right)^2 - 2 \times 10^4\right) u - 10^9 e^{-3\left(\frac{x-0.5}{0.01}\right)^2} = 0 \tag{9.4}$$

$$u(x) = 10^3 e^{-\left(\frac{x-0.5}{0.01}\right)^2} \tag{9.5}$$

The solution of the equation has a peak around $x = 0.5$ which complicates the solution. In Lanzkron *et al.* (1996), the set is determined before solving the problem: For a 100 node grid, the initial set is fixed as nodes 49-52. In current study, these points are automatically determined using set rules. The solution of the problem is given in Figure 9.36. In Figure 9.37, Newton's Method is compared with the solution based

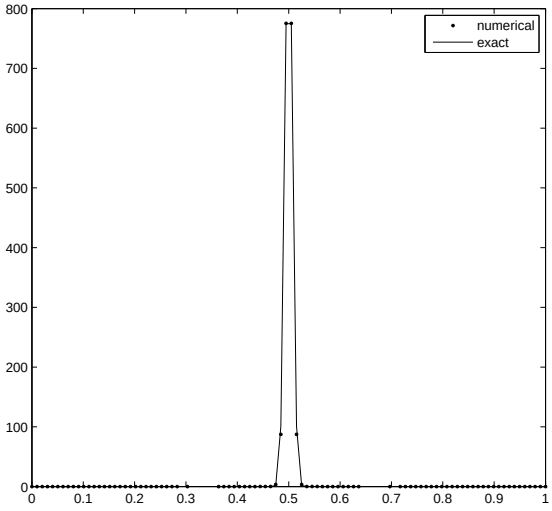


Figure 9.36. Solution of the test problem

on the set idea. Excluding the initial norms, Newton's Method converges in 7 steps

[15] I believe there are typos in the equation given in the reference. The exact solution does not satisfy the ODE unless it is corrected as in Equation 9.4

whereas the set idea uses only 2 global iterations. In the sub solves of the set approach 6 and 3 Newton steps are performed, respectively. Newton's Method required 34 total linear iterations where the set idea needs 33 linear iterations. Although, set method is 1 linear iteration shy, more linear iterations per nonlinear iteration is needed compared to Newtons' method (16.5 vs. 4.9). In the first step of the set idea, the local set is identical to those points used in Lanzkron *et al.* (1996), only 4 points (49:52) are kept in $\mathcal{S}_L$. In the second iteration, however, the local set is expanded to 12 points (45:56). In both cases, the local set is around $x = 0.5$ which makes sense because of the spike in the problem. This comparison is based by using average of the nonlinear residuals with

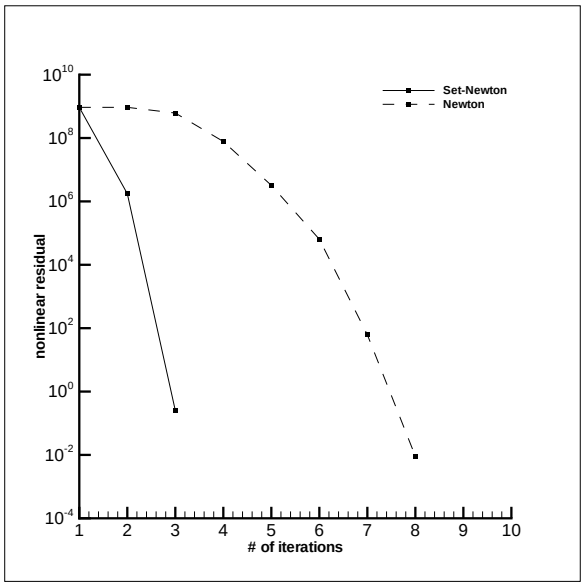


Figure 9.37. Comparison of the set idea with Newton's Method

$\alpha = 0.01$. When $\alpha = 1$, then set still converges in two steps but requires 35 total linear iterations. Use of RMS of $\mathbf{f}$ performs the same as the average values. Use of the step size criterion did not result a reduction in the set and the computations are performed as Newton's Method. When the number of unknowns is increased, it is observed that

Table 9.9. Comparison in various grid sizes

grid size	Newton Time	Newton Iters	Set Time	Set Iters	Set size
500	0.4 s	11(105)	0.25 s	3(137)	20,36,26
1000	0.92 s	10(159)	0.51 s	3(293)	40,68,59
5000	111.17 s	13(1422)	17.38 s	4(2101)	196,311,287,498

the set idea requires more linear iterations, however, the computations are performed in less amount of time. In Table 9.9, the methods are compared on different grid sizes where iters denotes the nonlinear iterations and the numbers in the parenthesis are total number of linear steps. Set size shows the size of the local set at each set iteration. The sets are created using the average values with $\alpha = 0.001$. The efficiency of the proposed idea is apparent over the Newton's method when computational times are considered. The results in Table resembles 9.9 the results in Lanzkron *et al.* (1996) but speedup over the Newton's Method is not constant as in the reference.

9.4. Applications

In this section, different multiphysics problems are introduced. In these problems, various aspects of the solution techniques are analyzed. First problem is an example of a Fluid-Structure Interaction problem. In the model problem, a bimetallic strip is to be deflected because of the thermal gradient on the solid which is a result of the natural convection around the structure. Because of the differences in the thermal expansion coefficients of two halves, the beam deforms so does the flow domain. Second problem is an extension of the multiphase test problem introduced in section9.5. Here, there is a water column that is to be collapsed into the air. The object is to determine the flow field and also to track the interface between water and air.

9.4.1. Analysis of a bimetallic slab in non-isothermal flow

Main object of this study is to investigate a Fluid-Structure Interaction (FSI) problem using Newton-Krylov Techniques. As a test problem, a bimetallic slab exposed to a non-isothermal flow field is selected. If the metals on the body have different coefficients of thermal expansion, then the structure deflects in the case of a temperature change. In this study, natural convection is the reason for the temperature gradient on the slab. Solution of incompressible Navier-Stokes Equations are weakly coupled with linearized quasi-static thermoelasticity problem. Stream function - vorticity approach is used to analyze the governing equations of the flow problem and finite differences are used for discretization. Finite element method is preferred to solve the deflection

of the solid which is modeled with plane strain assumption. Matrix-free methodology is utilized in the Newton-Krylov solver. Inexact Newton method is employed along with preconditioned GMRES as the linear solver. Domain decomposition is utilized both to ease the solution of the deformed flow geometry and to separate both physics. Several computations are performed for various Rayleigh numbers and various results are presented.

9.4.1.1. Description of the Problem. The problem consists of two physics: fluid and solid domains. For the sake of the analysis, the equations of both physics should be presented. First, we can start with the fluid mechanics. Governing equations of fluid mechanics are the Navier-Stokes Equations. General form of incompressible Navier-Stokes Equations is given in 9.6-9.8 Bejan (2004) where $\mathbf{u}$ denotes the velocity field, p is the pressure, T is the temperature and ρ, μ, k, c_p, denote the density, dynamic viscosity, conductivity and specific heat, respectively. Equation 9.6 is the mass conservation whereas Equation 9.7 is the momentum equations for each velocity components and Equation 9.8 is the energy equation. For a forced convection problem, the energy and momentum equations are decoupled. Once the flow field is determined, the temperature field can easily be extracted.

$$\nabla \cdot \mathbf{u} = 0 \tag{9.6}$$

$$\rho \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{f} \tag{9.7}$$

$$\rho c_p \mathbf{u} \cdot \nabla T = k \nabla^2 T \tag{9.8}$$

On the other hand, in a free convection problem the variation of the density cannot be ignored. The fluid motion in natural convection is driven by the buoyancy. Although the density is not constant all over the domain, an assumption on the vari-

ation of the density can be made through the Boussinesq approximation Bejan (2004) given in Equation 9.9, β being the thermal expansion coefficient of the fluid and subscript r denotes reference values. With the introduction of Boussinesq assumption, density is taken constant throughout the entire equation set except the body forces.

$$(\rho - \rho_r) \approx \rho_r \beta (T - T_r) \tag{9.9}$$

Use of this approximation couples the momentum and energy equations so two velocities (u, v), pressure (p), and the temperature (T), should be solved at once. However, instead of solving these primary unknowns, one could use the stream function (ψ) - vorticity (Ω) formulation to determine the flow field. The advantage of following this methodology rather than the primitive variables (u, v, p, T) is that complications regarding to the treatment of the pressure can be eliminated, and in two dimensional coordinate systems, this approach is computationally more efficient because of reduced number of unknowns. Non-dimensional form of the equations in both cartesian and polar coordinates are given in 9.10-9.15. The polar form is also introduced since it will be used in the deflected parts of the fluid domain.

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\Omega \tag{9.10}$$

$$Pr[\frac{\partial^2 \Omega}{\partial x^2} + \frac{\partial^2 \Omega}{\partial y^2}] + RaPr\frac{\partial T}{\partial x} - [\frac{\partial \psi}{\partial y}\frac{\partial \Omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \Omega}{\partial y}] = 0 \tag{9.11}$$

$$[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}] - [\frac{\partial \psi}{\partial y}\frac{\partial T}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial T}{\partial y}] = 0 \tag{9.12}$$

$$\frac{\partial^2 \psi}{\partial r^2} + \frac{1}{r}\frac{\partial \psi}{\partial r} + \frac{1}{r^2}\frac{\partial^2 \psi}{\partial \theta^2} = -\Omega \tag{9.13}$$

$$Pr[\frac{\partial^2\Omega}{\partial r^2}+\frac{1}{r}\frac{\partial\Omega}{\partial r}+\frac{1}{r^2}\frac{\partial^2\Omega}{\partial\theta^2}] - [\frac{\partial\psi}{\partial\theta}\frac{\partial\Omega}{\partial r}-\frac{\partial\Omega}{\partial\theta}\frac{\partial\psi}{\partial r}] + RaPr[\frac{1}{r}\frac{\partial T}{\partial\theta}cos(\theta)+\frac{\partial T}{\partial r}sin(\theta)]=0 \quad (9.14)$$

$$[\frac{\partial^2 T}{\partial r^2}+\frac{1}{r}\frac{\partial T}{\partial r}+\frac{1}{r^2}\frac{\partial^2 T}{\partial\theta^2}]-\frac{1}{r}[\frac{\partial\psi}{\partial\theta}\frac{\partial T}{\partial r}-\frac{\partial T}{\partial\theta}\frac{\partial\psi}{\partial r}]=0 \quad (9.15)$$

The most important parameter in these equations is the Rayleigh number, Ra. The problem gets harder to analyze as Ra number increases. The reason for this complication is that the nonlinearity increases as the convective terms dominate the flow conditions. Prandtl number, Pr, is the other physical parameter and compares the velocity and thermal boundary layers, however, in this present study it is taken as unity.

$$Ra=\frac{g\beta H^3(T_0-T_r)}{\alpha\nu} \qquad Pr=\frac{\nu}{\alpha} \quad (9.16)$$

Boundary condition for stream function, ψ, is set to zero, however, for vorticity, Ω, Equation 9.17 should be implemented where subscript $wall$ denotes a node on the wall and $wall-1$ denotes an interior node in the vicinity of the wall.

$$\Omega_{wall}=\frac{2\left(\psi_{wall}-\psi_{wall-1}\right)}{\Delta n^2} \quad (9.17)$$

Bottom and top walls are assumed to be insulated therefore homogenous Neumann boundary conditions are implemented for the dimensionless temperature as seen in Figure 9.38. Temperature of left wall is unity ($T=1$) and right wall is kept as zero ($T=0$). Temperature boundary conditions on the metallic slab are complicated. At the first solve of the convection problem, the surface of the solid is also taken as zero. Later, the temperature values are determined by the coupling algorithm.

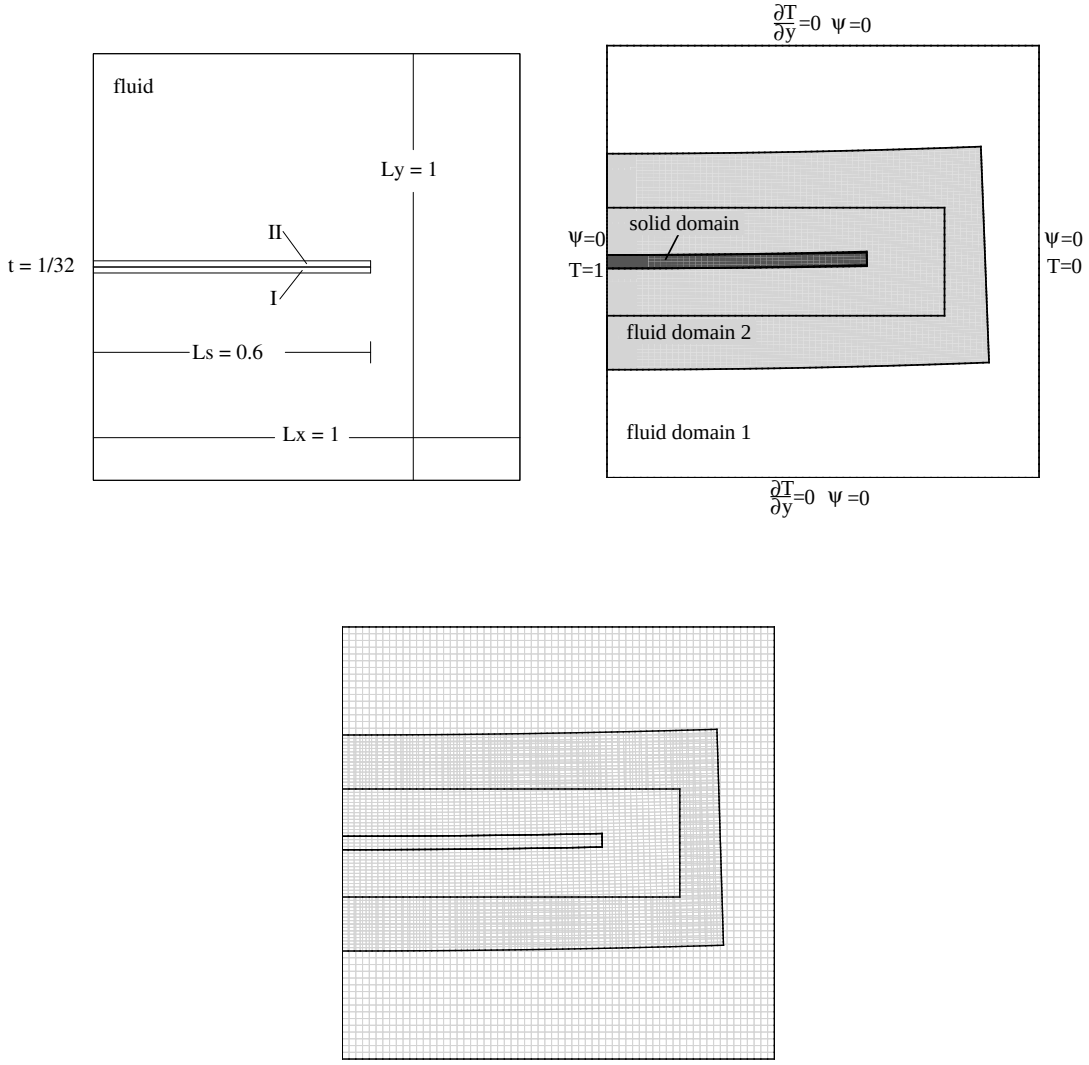


Figure 9.38. Description of the geometry and the boundary conditions and mesh of the domains

Second physics of the problem is the solid part which is a bimetallic slab composed of two different metals. Main feature of a bimetallic slab is the difference between the coefficient of thermal expansion, α of two materials. In the presence of a temperature gradient, the metal will bend since the elongations of the layers will be in different amounts. Heat transfer and elasticity are coupled with each other, however with a linearized assumption, both fields can be decoupled and the deflection and the temperature distribution can be solved with iterations between each systems. This model problem is in two dimensions (infinitely long in third dimension) as a result a plane-strain assumption is appropriate for solving the displacement field. Equation 2.12 states the equilibrium and Equation 2.13 gives the relation between the strains and the displacements Timoshenko and Goodier (1970). u and v in Equation 2.13 should not be confused with the velocity components in Navier-Stokes Equations. Since the fluid flow is now analyzed with stream function (ψ) and vorticity (ω) formulation, u and v throughout this study will point the displacements in x and y directions, respectively.

In order to perform a displacement based finite element analysis, the constitutive equation should also be given. For an isotropic homogenous material the constitutive equation, $\sigma = \mathbf{D}\epsilon$, is given in Equation 9.18 where the material stiffness matrix, $\mathbf{D}$, is valid for plane strain Pepper and Heinrich (2006).

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{E(1-\nu)}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{9.18}$$

In order to calculate the thermal stresses, the constitutive relation should be modified as $\sigma = \mathbf{D}\left(\epsilon - \epsilon_T\right)$, and the effect of thermal expansion through the thermal strain, ϵ_T, should be included. For plain strain problems, thermal strain is defined as in Equation 9.19 at which α is the thermal expansion coefficient and ΔT is the temperature difference Cook (1995) . It can be seen that only normal strain components

are effected by the change in the temperature.

$$\varepsilon_T = (1+\nu)\begin{Bmatrix} \alpha\Delta T \\ \alpha\Delta T \\ 0 \end{Bmatrix} \tag{9.19}$$

Although the governing equations of the fluid model are discretized with finite differences, the displacements are calculated with finite elements using bilinear quadrilaterals. The equations that leads a system like $\mathbf{Ku} = \mathbf{F}$ are given in the Equation 9.20 and 9.21

$$\mathbf{K} = \int_A \mathbf{B}^T\mathbf{D}\mathbf{B}dA \tag{9.20}$$

$$\mathbf{F} = \int_A \mathbf{B}^T\mathbf{D}\varepsilon_T dA\mathbf{N}^T\mathbf{f}dA + \int_S \mathbf{N}^T\mathbf{t}ds \tag{9.21}$$

with,

$$\mathbf{N} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \times \mathbf{N}$$

where N_i is the shape function of the i'th node, and ϵ_T is the thermal strain Pepper and Heinrich (2006). The solid is attached to the left side of the wall from its right end and free on the other end. Both degrees of freedom, u and v are kept zero.

Because of the deformed geometry, it is also convenient to solve the heat conduction on the solid with finite elements. The procedure is similar to the elastic case, $\mathbf{K_C T} = \mathbf{F_C}$ can be calculated from Equation 9.22 and 9.23, where subscript C denotes a conduction problem and q is the heat flux, and k_{xx}, k_{yy} are thermal conductivity in

x and y coordinates, respectively.

$$\mathbf{K_C} = \int_A \mathbf{B_C^T D_C B_C} dA \tag{9.22}$$

$$\mathbf{F_C} = -\int_S q\mathbf{N_C^T} dS \tag{9.23}$$

with,

$$\mathbf{B_C} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \end{bmatrix}, \mathbf{D_C} = \begin{bmatrix} k_{xx} & 0 \\ 0 & k_{yy} \end{bmatrix}, \mathbf{N_C^T} = \begin{bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{bmatrix}$$

9.4.1.2. Coupling Procedure. In this study, a partitioned analysis Felippa *et al.* (2001) is performed and there is a two-way weak coupling between the physics. On the other hand, the thermoelastic coupling between the energy and elasticity problems on the slab is a one-way coupling: The temperature distribution is calculated on the structure and then the deformation is calculated via thermal loads on the body by solving the displacement field. Later, instead of updating the temperature profile for the deflected geometry, the computation continues with recalculation of the flow field at which the temperature values on the nodes of the deformed solid is applied on the inner domain of the flow field for Equation 9.15. Although thermoelasticity is nonlinear in a general sense because of the thermodynamic considerations (Nicholson (2003)), the analysis followed in this study performed well because the deformations stays in the linear elastic region and the problem is treated quasi-static because the computations are handled in an iterative fashion by visiting both physics, and final convergence is needed only at the end.

Focusing on the multi-physics, the coupling between two physics is mainly established with heat flow. Assuming an initial temperature for the slab, the free convection problem is solved with non-homogenous Dirichlet boundary condition where first solve is attempted with the initial temperature as the boundary condition. Temperature distribution inside the structure is computed with non-homogenous Neumann boundary conditions calculated within the values of the fluid domain. Thereafter, the temperature values on the surface of the solid body is used as the boundary condition of the flow problem. Once a temperature distribution on the structure is calculated and the deformation is determined, this temperature field is treated as the initial state for the next computations.

When the solid deforms, the boundaries of the flow should be modified accordingly. This can normally be accomplished with re-generation of the mesh after each deflection. Instead of this approach, a method based on domain decomposition is followed. Basically, the domain of the fluid is divided into two overlapping, non-conforming grids as observed in Figure 9.38. Outer grid preserves its shape while the inner adjust itself with the deflection of the solid part. Iterations between domains will continue until convergence is achieved. Two remarks should be stated before proceeding with the next section. The pressure on the solid as a result of the fluid is ignored mainly because the pressure is not significant compared to a forced convection problem. Only thermal conditions are effective on the solid. It should also be mentioned that an average radius of curvature is to be found since the deformed sub-domain is assumed to be in polar coordinates. To facilitate the new geometry, an average radius of curvature must be calculated with a least-squares approximation. This way, the new geometry can be constructed with the average curvature. If some portion of the solid deforms unexpectedly more than the rest, then the extra deflection can be damped with this strategy. In this study, the problems regarding to the advection of the deforming mesh are not addressed mainly because we are after finding a steady state solution and intermediate problems are treated as being steps towards the steady physics.

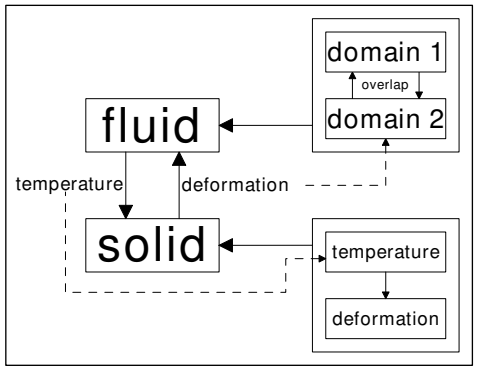


Figure 9.39. Solution Algorithm

9.4.2. Numerical Methods

In this study, several methods are covered to solve the problem. Systems of nonlinear equations are solved with Newton's Method and linear problems are handled with GMRES, one of many Krylov solvers. Newton-Krylov methodology is improved with the matrix-free idea so the formation of the Jacobian is avoided. Additionally, domain decomposition is employed and particularly in two different ways. At first, it is used to separate the physics of the problem into fluid and solid domains. Secondly, it is used to simplify the analysis of the fluid domains where the deformations are confined in a subdomain so that it is conformable with the solid domain.

In finite element method, global assembly of the stiffness matrix is avoided and the computations are done in element-by-element (EBE) basis Smith and Griffiths (2004). In EBE approach, only local stiffness matrices are stored. If a multiplication is necessary, then the vector to be multiplied is operated only the respective part of the local stiffness matrices. In this approach, storage considerations can be reduced significantly and the computations can be done in parallel processors.

When the Krylov solvers are in concern, it should be said that they perform poorly without a proper preconditioner. A compromise could be made when selecting the preconditioner. If a simple method like Jacobi or Gauss-Seidel is selected, then the application of the preconditioner does not require that much of a computational power. On the other hand, the number of iterations that is necessary to fulfill the

convergence criteria is still high. More advanced methods like incomplete-LU decompositions with fill-ins, ILU(ℓ), are superior to those methods in terms of the iteration numbers, however, they require more space and time for the calculations. One should also pay attention to the matrix-free approach. Although the Jacobian matrix is not explicitly required in GMRES and in other Krylov solvers, an approximate matrix should be formed to perform the preconditioning. This matrix can either be formed from a low-order approximation of the problem or the Jacobian matrix might be calculated at the beginning of nonlinear iterations and kept the same for several other updates. In this study, Jacobi preconditioner is picked as the preconditioner and it performed well for the range of Rayleigh numbers investigated.

9.4.2.1. Domain decomposition method. Domain decomposition method simplifies the analysis in several aspects. Firstly, the number of the unknowns on each domain is smaller compared to the whole domain. On the other hand, when total number of unknowns of all subdomains are in concern, the degree of freedom of this new problem exceeds the original full problem. However, the ability of using parallel processing in domain decomposition can resolve this problem. When small problems are realized in parallel, each subdomain can be analyzed in different processor and with proper load-balancing, and inter-processor communications, the overall analysis can be optimized to succeed the full problem.

In this study, domain decomposition is used to fulfill another necessity that arises from the deformation of the solid boundary; regeneration of the grid. When the solid part deflects, the fluid domain should adapt to the change. This can be handled in a single domain either by grid generation all over the flow area or by moving the mesh in some amounts. Then again, use of domain decomposition plays an important role and limits the generation of the grid only for the inner domain. In this way, several different cases can be analyzed simultaneously by just varying the conditions only in the inner problem.

It should be said that, although an overlapping grids methodology is followed, the amount of overlap is changing as the metal deflects. This change also affects the convergence properties of the problem. When the geometry is deflected, the problem is getting complicated so this overlap can be used to overcome the difficulties (or balance the computations) since it is known that the amount of overlap improves the domain decomposition analysis in Alternating Schwarz method (Smith *et al.*, 1996). After deflections of the domains, the data transfer between subdomains is achieved with bilinear interpolation. Calculated stream function, vorticity and temperature values are applied directly regardless of the coordinate system.

Strikwerda and Scarbnick (1993) recommended a strategy for interpolations between two different coordinate systems: during the interpolation variables should be transformed first and then interpolation should be performed rather than first interpolation and then transformation. In this study, velocities are not used so such a methodology can be ignored since the variables in this study, ψ, Ω, T, all are scalar. As a result, data interchange between domains is carried out by simple bilinear interpolations.

9.4.3. Results and Discussion

Several computations are performed for $10^0 \leq Ra \leq 10^6$. Pr is kept unity, except the validation study. Newton's method is said to be converged for 10^{-5} order change on the initial residual. Maximum number of nonlinear iterations is set as 20. If the domain solve is successful in 20 steps, then the solver passes to the other subdomain. GMRES(200) with Jacobi preconditioning is used as the linear solver. Iterations between fluid sub-domains is stopped if 2-Norm of the difference of two consecutive solves is less than 10^{-4}. It is observed that more strict tolerances do not affect the convergence of the domain decomposition method. For the coupling, the iterations between physics pronounced to be at the steady state if the change in the radius of convergence is less than 10^{-4}. In section 9.4.3.1, the algorithm is tested against the study of Bilgen Bilgen (2005). In section 9.4.3.2, free convection results without deformations is presented. Last section is devoted for the results of the coupled problem.

9.4.3.1. A Comparison. Before solving the natural convection problem in the main geometry, a study is performed to compare the results with Bilgen (2005). To do that, a thin strip is prepared with length 0.5 and thickness 0.01. As given in Bilgen (2005), the dimensionless conductivity of the fin is selected as 30 and Pr number is set to 0.7. The computations are performed for $Ra = 10^6$. Figure 9.40 demonstrates the flow field. On the left, the results of the current study is presented. On the right, result of Bilgen (2005) is given. As observed from the figures, the plots are very similar. Another important aspect of the comparison is that the result is taken from a domain decomposition solve. Similar to the main problem, also one interior and one outer domain is used. The plot reveals the fact that domain decomposition performs well.

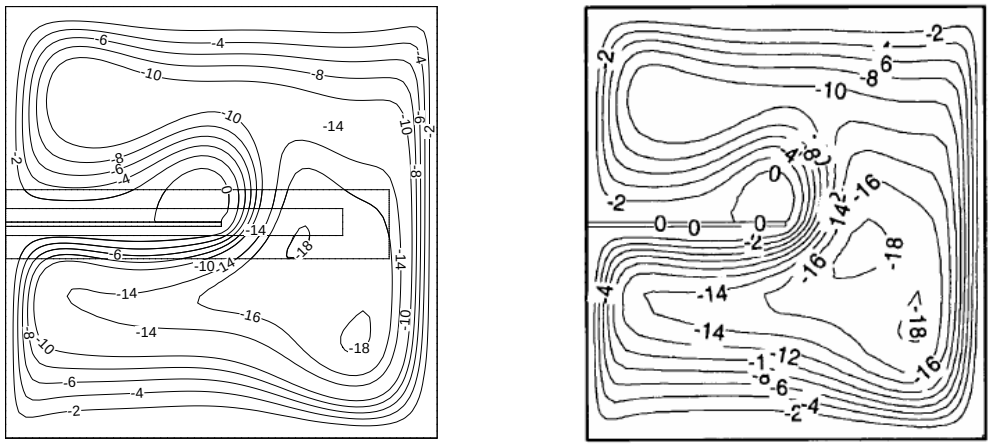


Figure 9.40. Streamlines for $Ra = 10^6$ and $Pr = 0.7$. Left figure current study, right figure Bilgen (2005)

9.4.3.2. Natural Convection. In the first part, the problem is solved for a fixed radius of curvature, $\rho = 2.5$. Such a value is not encountered in the analysis of the FSI problem, however, it is given here to show that the solver is capable of handling large changes in the geometry. Results presented here are for $Ra = 10^4$ and $Ra = 10^5$. For

this case, dimensionless temperature values of the inner boundary of the polar domain is given as unity. Stream function contours as well as the temperature distribution for

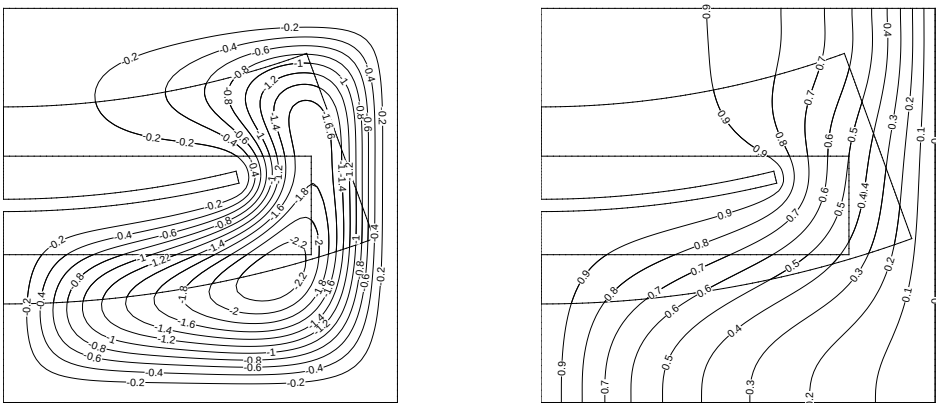


Figure 9.41. Stream function and Temperature distribution for $Ra = 10^4$ and $\rho = 2.5$

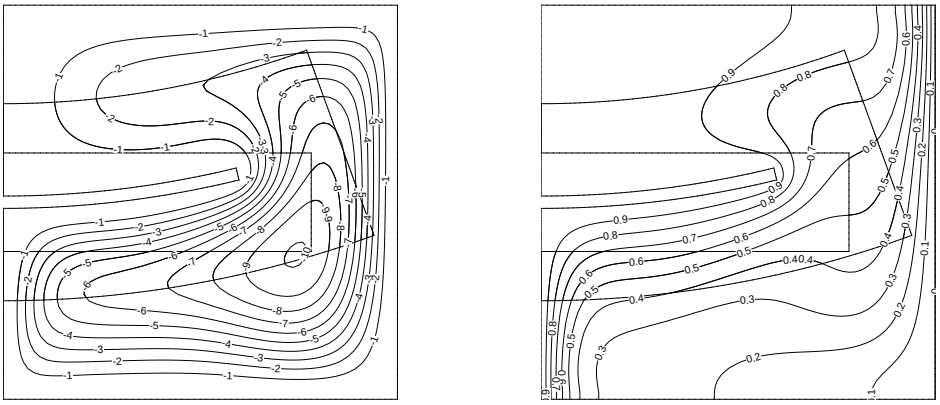


Figure 9.42. Stream function and Temperature distribution for $Ra = 10^5$ and $\rho = 2.5$

$Ra = 10^4$ and $Ra = 10^5$ are given in Figure 9.41 and 9.42, respectively. The boundaries of the subdomains are clearly visible. As one can notice, the solutions are in agreement in both domains and the contour lines passes smoothly from one domain to another. In Figure 9.43, a typical nonlinear convergence plot is shown for $Ra = 10^5$. After an initial increase in the nonlinear norm, the residual starts to decrease in a steady rate and drops gradually in last steps. In this part of the study, continuation over the parameter Ra is followed so an improved initial guess is used for Newton's method. If an initial

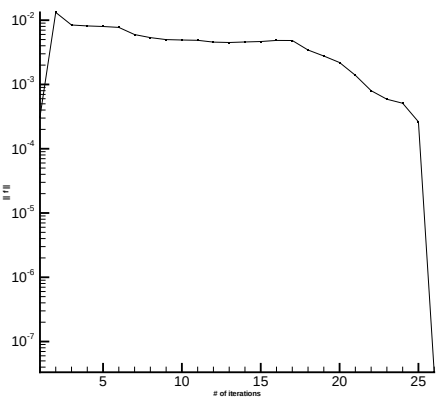


Figure 9.43. Typical non-linear convergence history for $Ra = 10^5$

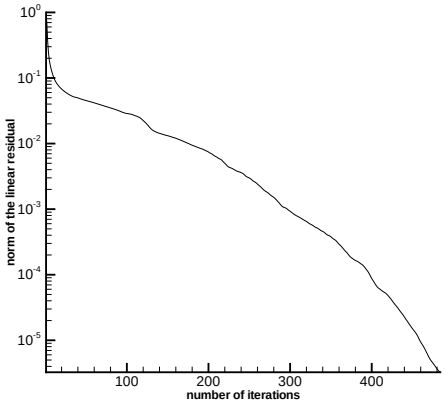


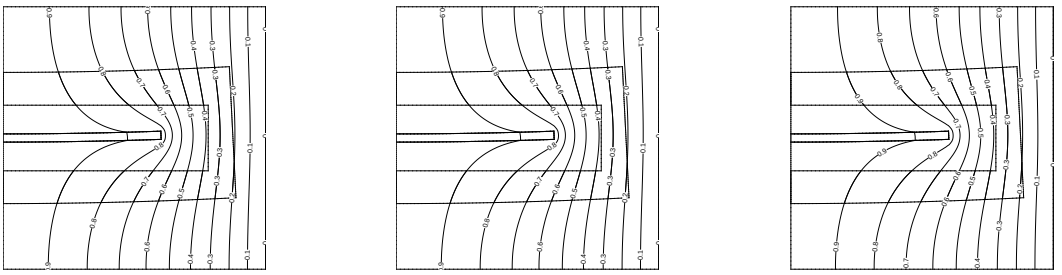
Figure 9.44. Typical linear convergence history for GMRES at $Ra = 10^5$

guess is not available, global convergence strategies like line search - backtracking can be employed (Brown and Saad, 1990). In Figure 9.44, linear convergence of GMRES is presented. As observed from the figure, the performance of Jacobi preconditioner is satisfactory.

9.4.3.3. Natural convection with deforming solid. In this section, the results of the main problem is given. For the solid part, the material properties given in Table 9.10 are used. As observed from the table, Aluminum's thermal expansion coefficient is greater then that of Steel's. Hence, as Aluminum being on the bottom layer of the slab, a temperature rise within the body will deflect the solid in positive y direction. The equations in the solid part are solved dimensional, however, their values are kept the same of those of non-dimensional forms. On the other hand, the temperature is not taken equal. A temperature rise in the order of 1 would not generate enough deflection to deform the structure. That's why the values calculated in the flow field are scale with 100. As a result, 100 times larger temperature difference is applied on the system which is physically reasonable. The geometrical description of the problem is given in Figure 9.38. Several computations are made for $Ra = 10^0, 10^1, 10^2, 10^3, 10^4, 10^5, 10^6$. To be fair on the comparison charts, the computations for all Ra numbers started from zero initial guesses.

Table 9.10. Material Properties Incropera and DeWitt (2001),Ugural and Fenster (1994)

	Aluminum (I)	Steel (II)
Modulus of Elasticity (E)	72 GPa	200 GPa
Poisson's ratio (ν)	0.35	0.28
Coefficient of Thermal Expansion(α)	23e-6 $°C^{-1}$	12e-6 $°C^{-1}$
Thermal Conductivity (k)	237 $W/m \cdot K$	40 $W/m \cdot K$



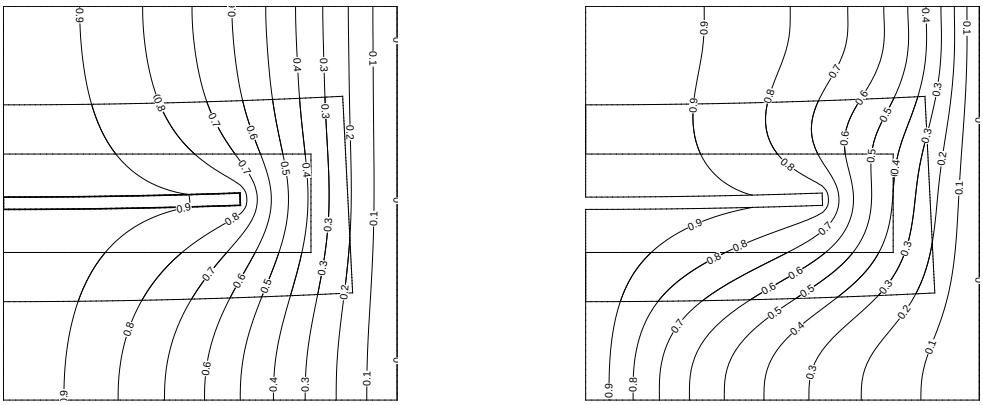
a) b) c)

Figure 9.45. Temperature distributions **a**) $Ra = 10^0$, **b**) $Ra = 10^1$, **c**) $Ra = 10^2$

In Figure 9.45 temperature profiles for $Ra = 10^0, 10^1, 10^2$ are shown. It is clear that all three solutions are very similar. Radius of curvatures for all three cases are equal and read as 17.304. All three cases, needs 12 visits between fluid and solid to converge. Total number of subdomain solves in the flow problem is 63 for $Ra = 10^0, 10^1$ and 66 for $Ra = 10^2$.

Results of $Ra = 10^3$ also resembles previous plots (Figure 9.46). Radius of curvature is 17.304 and 12 iterations are needed for physics convergence, as well. 67 total visits are observed for the solution of the free convection problem. Temperature profile of $Ra = 10^4$ is slightly different and convection starts to dominate the flow. Radius of curvature is not much different than previous cases and equal to 17.503. Although, total number of solves are equal to $Ra = 10^4$ with 67 visits, 13 iterations are needed between fluid and solid physics for a converged solution.

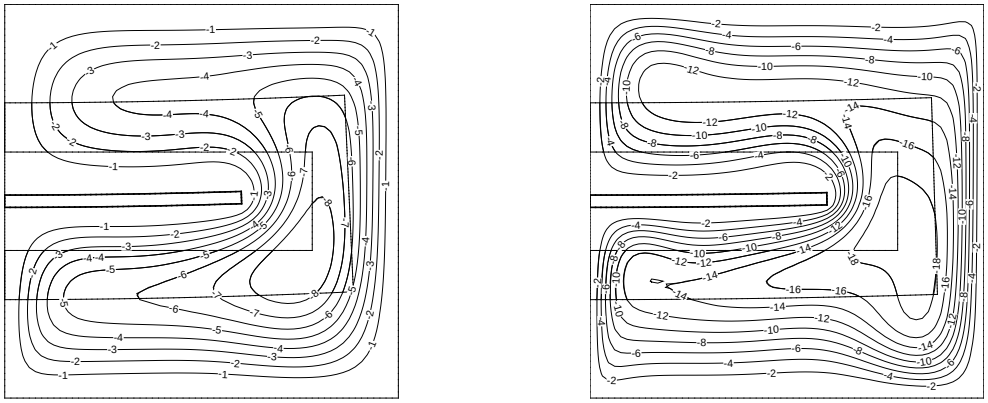
$Ra = 10^5$ and $Ra = 10^6$ problems are numerically more challenging than the aforementioned case because the nonlinearity is increased and more domain solves are



a) b)

Figure 9.46. Temperature distributions **a**) $Ra = 10^3$, **b**) Ra=10^4

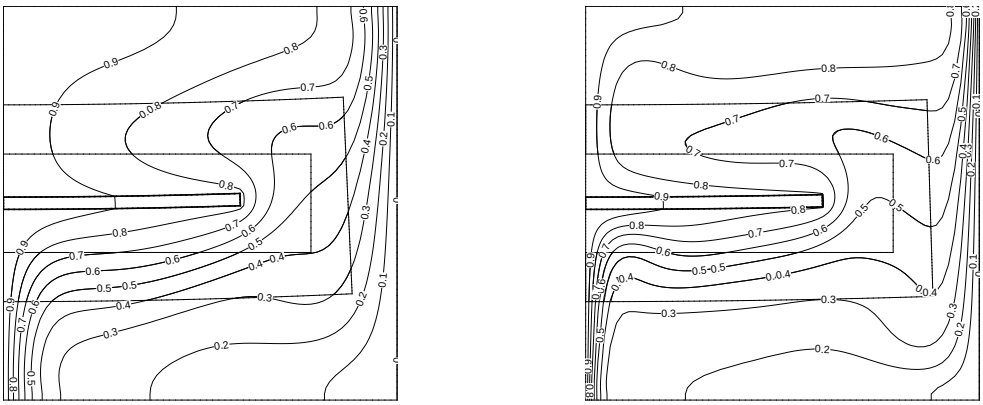
needed. Stream function contours are given in Figure and temperature distributions are given in Figure . Radius of curvatures are 19.372 and 28.868, respectively. $Ra = 10^5$ case converges in 18 physics iterations and performs 84 subdomain solves to get there. $Ra = 10^6$ converges in 40 iterations and needs 183 subdomain solves.



a) b)

Figure 9.47. Stream function contours **a**) $Ra = 10^5$, **b**) Ra=10^6

In Figure 9.49 the temperature distribution on the center line i.e on the interface of the slab for $Ra = 10^3, 10^4, 10^5, 10^6$ is given. As seen from the figure, the tip of the beam gets colder with increasing Ra number. This is expected because convection starts to dominate and cools the metal. This figure excludes temperature variation of



a) b)

Figure 9.48. Temperature distributions **a**) $Ra = 10^5$, **b**) Ra=10^6

smaller Ra number since they follow nearly the same profile of $Ra = 10^3$. Another outcome of this result is that temperature difference from the initial state decreases as Ra increases. That means the thermal load on the metal will be smaller for large Ra numbers. A consequence of this reality should be that the radius of curvature should be larger since the displacement is smaller amounts. Hence, this comment is validated in Figure 9.50. The figure shows that the radius of curvature is nearly constant till $Ra = 10^3$.

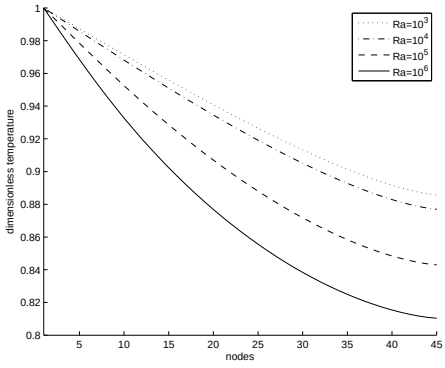


Figure 9.49. Centerline temperature profile

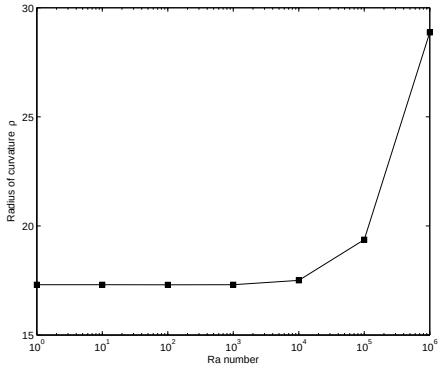


Figure 9.50. Radius of curvature

During the iterations between the physics, several undershoots and overshoots are observed as the system tries to reach the steady state. For small Ra numbers,

final radius of curvature is decided in less iterations, however, for larger values, more iterations are necessary because of the underdamped solution. This issue is depicted in Figure 9.53. This outcome is supported with Figure 9.51 where one can observe that the number of iterations between the fluid and solid increases as Ra number increases. Figure 9.52 shows the total number sub-domain solves with respect to Ra number. Here also we observe what is expected, as convection dominates (with increasing Ra number), the problem becomes more nonlinear and the number of domain visits increases. In Figure 9.54, the convergence of the error in the nonlinear least square problem is presented. For large Ra numbers, convergence history suffers from unexpected peaks but eventually the norm of the error decreases for all Ra numbers studied in this work. That means, when a problem is finished, the temperature values as well as the deformation of the slab comes to an end and the curvature is compatible for all elements in the structure. As an extreme case, a temperature scale of 500 is also considered. Computations are held for $Ra = 10^4$ and the plots are given in Figure 9.55 and 9.56. Radius curvature is calculated as 3.677 and total of 70 subdomain solves are performed in 11 physics iterations. Compared to former $Ra = 10^4$ problem, it can be said that less physics iterations are performed, however, total number of domain visits is increased.

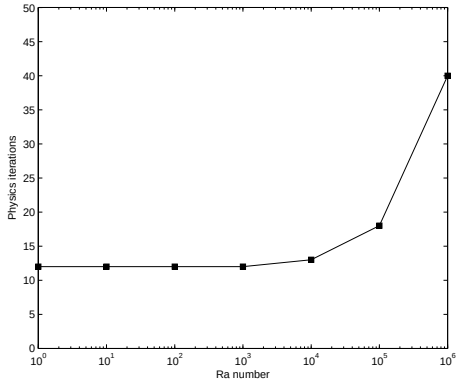


Figure 9.51. Iterations between physics

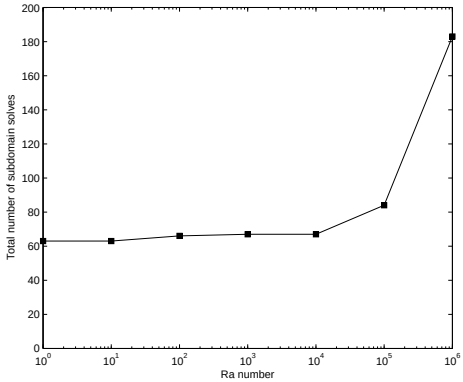


Figure 9.52. Total number of visits

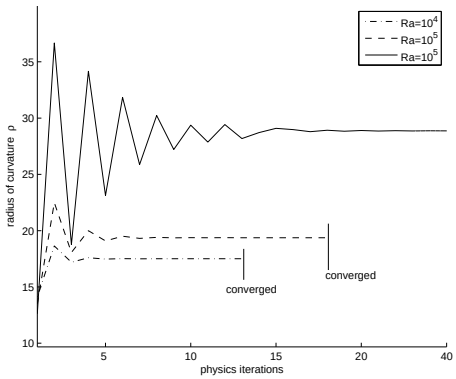


Figure 9.53. Convergence of radius of curvature

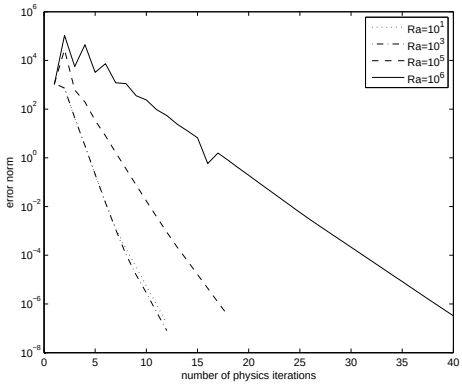


Figure 9.54. Convergence of nonlinear least squares

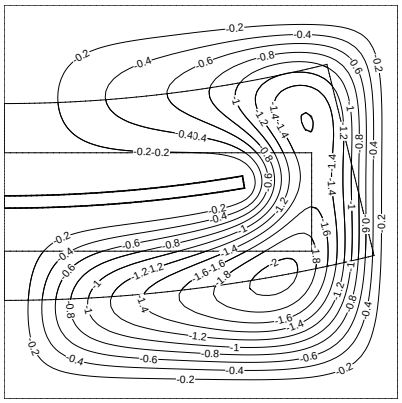


Figure 9.55. Stream function contours

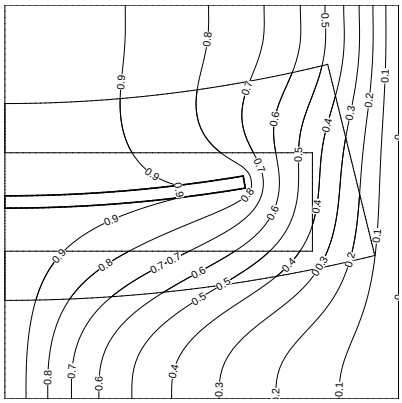


Figure 9.56. Temperature distribution

9.4.4. Summary

In this study, a fluid-structure problem is investigated such that a flow field around a bimetallic strip is to be found. The strip is allowed to deform hence, a strategy based on domain decomposition is used to reflect this change in the geometry into the fluid domain. The flow region is divided into two overlapping subdomains. Outer domain is kept the same and analyzed in cartesian coordinates, however, the inner domain is modified such that it is conformable with the deformed strip. Inner domain is studied with polar coordinates. The formation of the deflected geometry is based on the computation of the radius of curvature, an average over the deflected elements.

Fluid part is analyzed with stream function - vorticity approach for various Rayleigh numbers and discretized with finite differences. On the contrary, solid part is examined with plane strain assumption and solved through finite elements. The coupling between physics is achieved with heat flow. After the solution of the fluid equations, heat fluxes through the inner boundaries of the flow region is calculated. Then a conduction equation, with non-homogenous Neumann boundary conditions is solved for the solid part. After the calculation of the temperature field, thermal deformation of the strip is computed. When the fluid domain is visited again, recently computed temperature values on the solid is implemented as non-homogenous Dirichlet boundary conditions.

Nonlinear equations are solved with Newton-GMRES methodology. Matrix-free approach is followed to avoid the formation of the Jacobian matrix. Element-by-element computations are held during the solution of the finite element equations. Proposed solution algorithm is tested, first for a fixed radius of curvature to show that the solver can handle large changes in the flow domain. In the second part, the calculations are enhanced with the consideration of thermoelasticity. It is observed that the geometry changes when the thermal strain are implemented and the inner domains adapt themselves to deformed body.

To take the study one step further, one can analyze solids with different properties, like orthotropic for instance. In reality, elastic deformation occurs only in small amounts. In this study, it is shown that proposed method works well even for small radius of curvatures. As a result, problems with large deformations like in plasticity, might also be investigated. Additionally, a transient analysis can be conducted to understand the stability of the structure. In terms of the computational aspects, nonlinear preconditioning and multigrid can be employed. Parallel implementations should also be considered to reveal the power of domain decomposition.

9.5. Collapse of a water dam

Collapse of a water column w/ and w/o an obstacle (also known as the dam break problem) is a well known benchmark problem for two-fluid flows. In this section, volume-of-fluid methods is tested with the solver against the experimental work of Martin and Moyce (1952).

9.5.1. Description of the Problem

In this study, different than the all previous studies performed in the literature, VOF method is used in conjunction with AC to model the flow field. Since two phase flow is a transient problem, dual time stepping AC should be used to solve the governing equations. Real time increment, Δt is selected such that the physics of the system is conserved (for stability Courant number can be kept small even an implicit time marching is used), on the other hand, pseudo time increment $\Delta\tau$ can be selected as large as possible to avoid excessive number of artificial time steps. Additionally, an averaging based on the fraction, F, is to be examined whether it enhances the convergence of the discrete system. The problem is transient and solved in dimensional form. For completeness, the equations should be presented. ρ_m and μ_m are the mean values that

are found with F using Equations 9.25.

$$\frac{1}{\beta}\frac{\partial p}{\partial \tau} + \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0 \tag{9.24a}$$

$$\rho_m \left(\frac{\partial u}{\partial t} + \frac{\partial u}{\partial \tau} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} \right) = -\frac{\partial p}{\partial x} + \mu_m \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \tag{9.24b}$$

$$\rho_m \left(\frac{\partial v}{\partial t} + \frac{\partial v}{\partial \tau} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} \right) = -\frac{\partial p}{\partial y} + \mu_m \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + \rho_m g_y \tag{9.24c}$$

$$\frac{\partial \tilde{F}}{\partial t} + \frac{\partial \tilde{F}}{\partial \tau} + u\frac{\partial \tilde{F}}{\partial x} + v\frac{\partial \tilde{F}}{\partial y} = 0 \tag{9.24d}$$

$$\rho_m = \rho_1 \tilde{F} + \rho_2 \left(1 - \tilde{F} \right) \tag{9.25a}$$

$$\nu_m = \mu_1 \tilde{F} + \mu_2 \left(1 - \tilde{F} \right) \tag{9.25b}$$

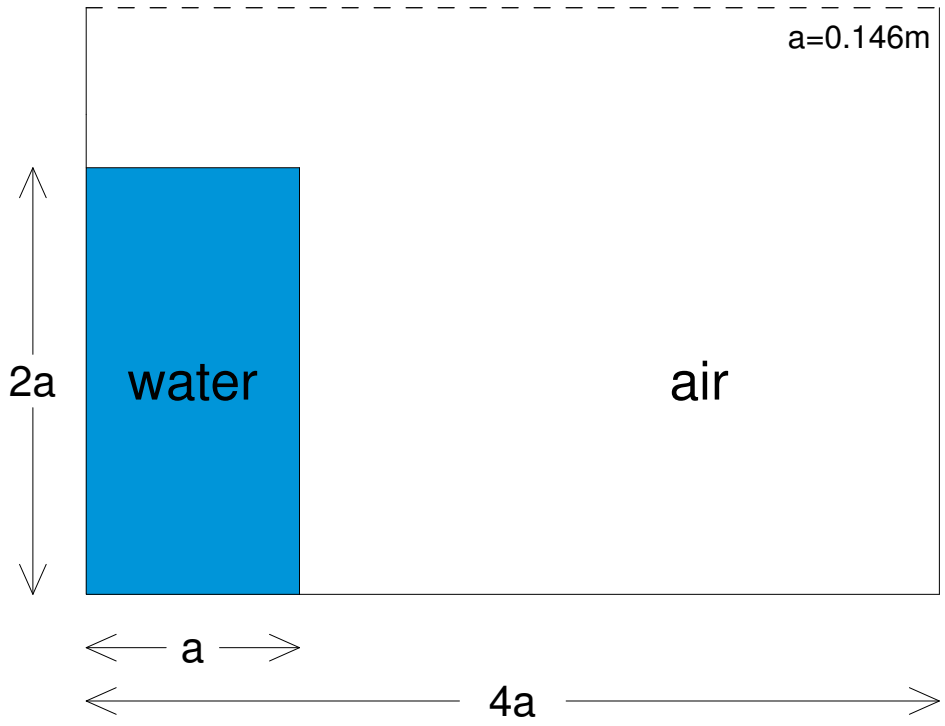


Figure 9.57. Geometry of the original problem

9.5.2. Results

Figure 9.58 shows the interface between water and air in various time points. To track the interface, $F = 0.5$ contour is followed. In reality the interface is a diffuse interface and accurate representation requires high order methods or fine grids. In this thesis, another study on the definition of convective fluxes is performed. Several Total Variation Diminishing (TVD) schemes and also SMART limited QUICK scheme is used to examine the changes on the interface. The comparison is made in Figure 9.59 at $t = 0.1sec$. As observed, Superbee predicts the interface in a narrower band. Still, the TVD schemes perform pretty successful.

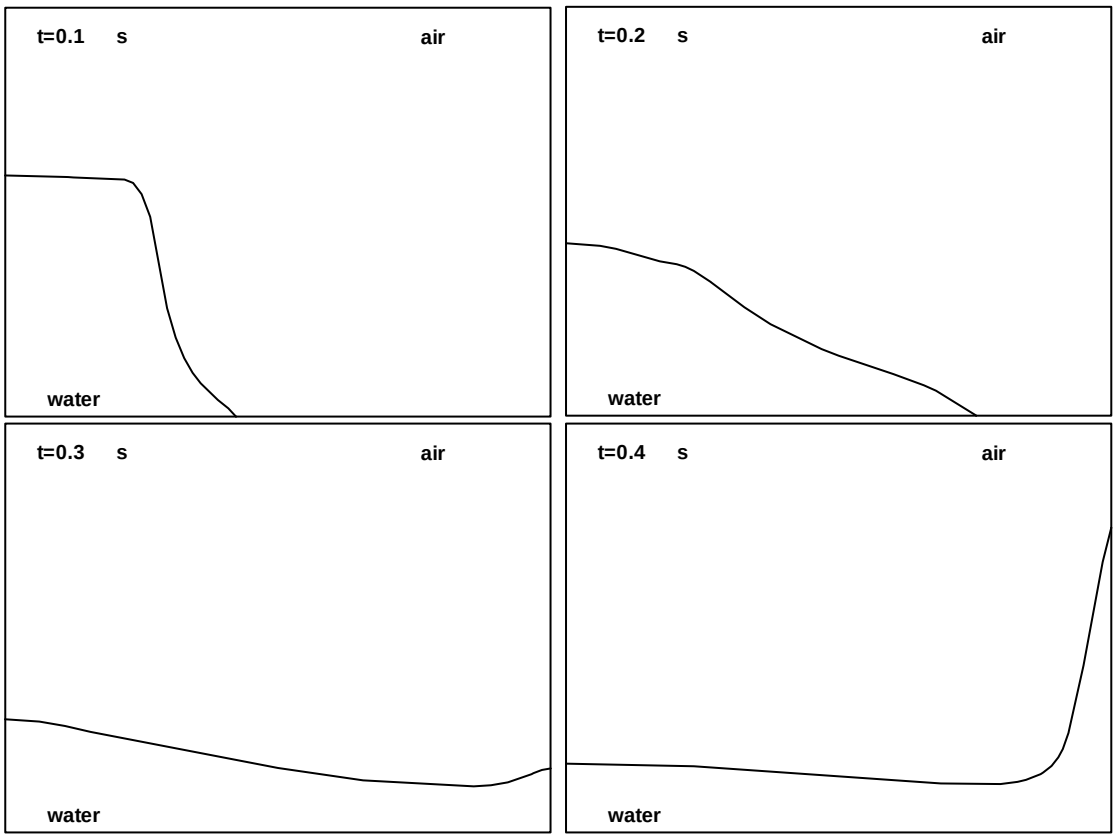


Figure 9.58. Fraction of the fluids in different time points.

Figure 9.60 compares water-air simulation with the experiments of Martin and Moyce (1952) and numerical calculations of Ubbink (1997). In this figure, $Z = z/a$ where z is the base of the water column at various time points and is equal to a initially. T is the dimensionless time variable such that $T = t\sqrt{\frac{2g}{a}}$. When the plots are examined,

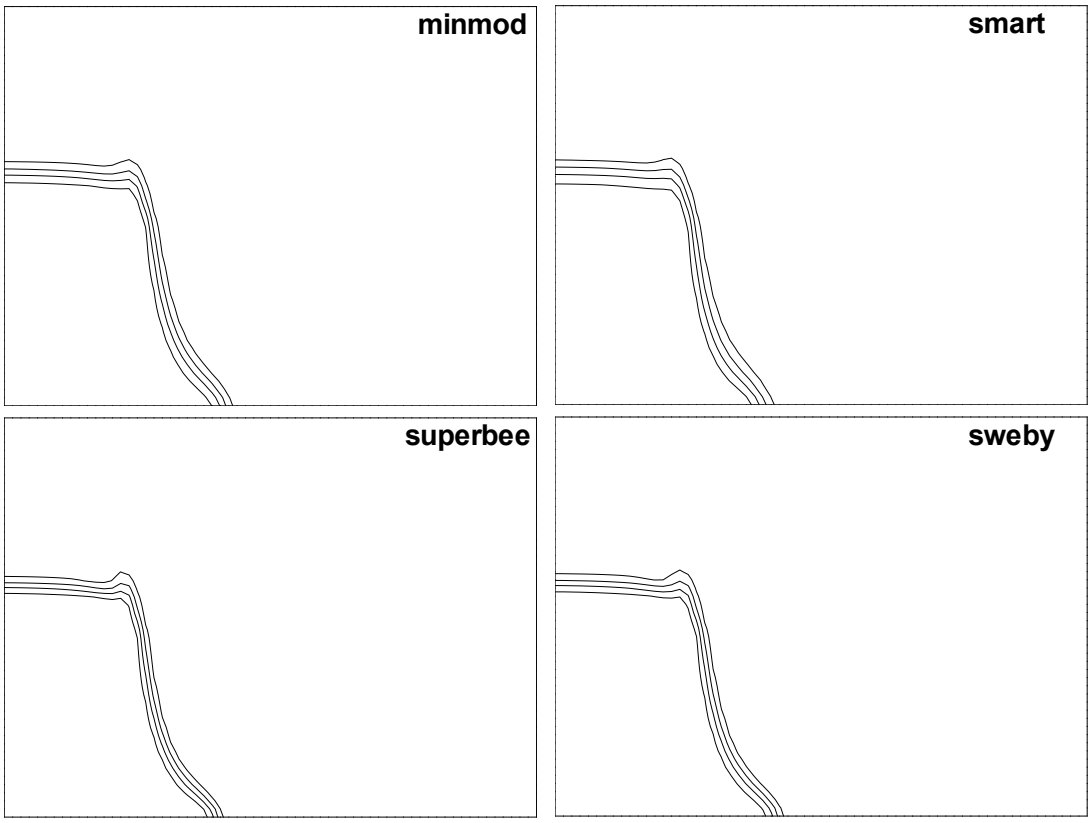


Figure 9.59. Flux limiters

it can be understood that the computations are in accordance. The difference between the numerical and experimental studies is that the experiment is actually 3D where end effects do change the flow. Figure 9.61 is another comparison with the studies in concern. Again, the results are validated. Here, dimensionless parameter H is defined as $H = h/2a$ where h is the height of the column in time. For this figure, the definition of T is different; $T = t\sqrt{\frac{g}{a}}$.

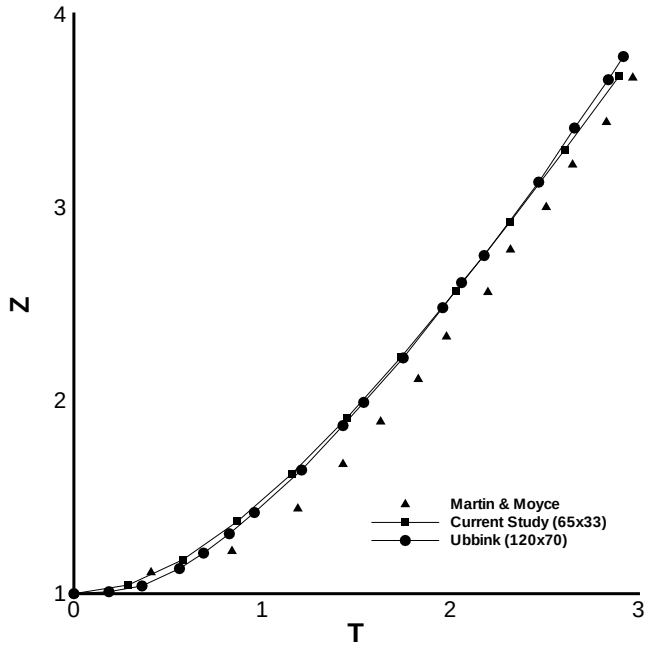


Figure 9.60. Comparison of the results with Martin and Moyce (1952) and Ubbink (1997)

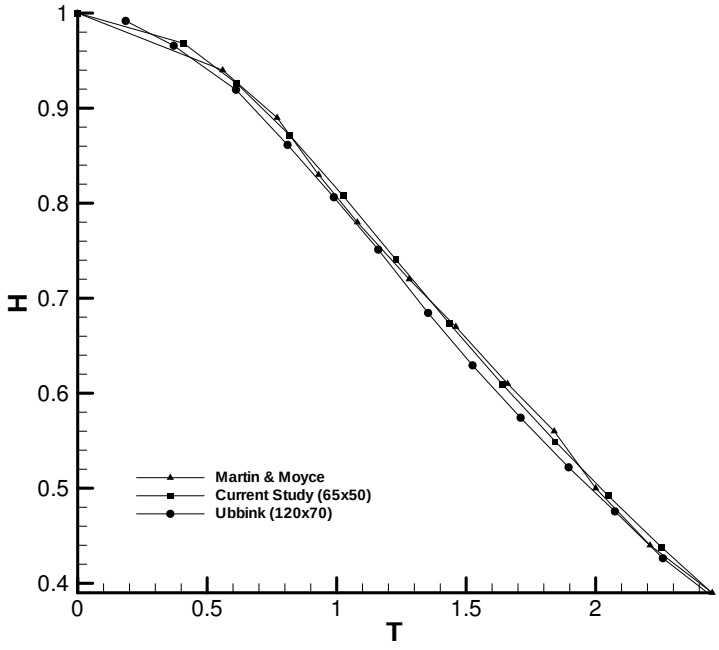


Figure 9.61. Comparison of the results with Martin and Moyce (1952) and Ubbink (1997)

10. CONCLUSION

Analysis of coupled-physics problems is a challenge in scientific computing. If a single physics is considered, it can be said that each physics has unique properties that is described by the mathematical model of the problem. Models in various research areas are examined for many years in scientific community. When multiphysics problems are in concern, interactions between physics and also variety of different couplings require multi-physics specific solution approaches which is quite different than mono-physics. This necessity can be realized by studying each physics by its own and by introducing formulations to define the relations between physics. Main advantage of this approach is that cutting edge numerical solution techniques developed for particular physics can be applied in a multi-physics environment, as well. However, this weakly-coupled approach suffers from reduced efficiencies in many aspects since several iterations are required for between physics. Strong-couplings, on the other hand, promise more success since the mathematical model is developed to define the features of that multi-physics problem. Here, however, the model is only for that particular case - tailored program cannot be extended considering a new problem. On top of these difficulties, we are also keen to use contemporary solution methods like Newton-Krylov and Multigrid which are also suitable for multiprocessor environments. Consequently, the object was to create a new computational framework to analyst multiphysics problems.

In this thesis, a contribution is made into the field such that contemporary solution techniques are applied on various multiphysics problems. A solver is presented where all physics of a model are contained in a single computational framework. The solver has several features such that different solution algorithms can be implemented without sacrificing on the definition of the discrete model. Both Newton-Krylov and Multigrid is used to deal with resulting nonlinear systems. Domain decomposition is used to enhance the solution process.

One of the main difficulty in using Newton-Krylov and Multigrid solvers is the definition of the discrete model. Krylov subspace solvers work on vectors, on the other

hand, multigrid requires scaler definition of the discretization. Jacobi method can be used as a vector operation within the context of matrix-free methodology, however, generalization of the solvers would require a different approach. For that reason, the solver called Demona is based on an idea that uses different solution sets. With the use of this approach, it is showed that different solution techniques can be analyzed by just providing one definition for the discretization.

Several computations are presented to state the performance of Newton-Krylov, multigrid and domain decomposition solvers. The calculations started with test problems and continued with applications. With the test problems, the solution methodologies introduced in this study is verified. Poisson's Equation, Hydrodynamic Lubrication, Bratu problem, and lid driven cavity problems are selected as benchmark problems. Additionally, solution on finite element and finite volume techniques are given.

Application problems are selected from multiphysics problems. One particular case is the analysis of a bimetallic strip that is exposed to natural convection. When the temperature varies within the slab, the body is deflected. The deformation on the fluid boundary is captured by decomposing the domain into two subdomain where one of them remains fixed and the other one changes it shape with the solid. The deflection on the solid is calculated with decoupled thermoelasticity.

Another model to analyze multiphysics problems was the multiphase problem. Collapse of a water dam is studied with the use of Artificial Compressibility and Volume of Fluid technique. It is showed that both formulations can be used together to model multiphase problems. Furthermore, the solver is capable of performing computations in dual-time step to analyst real-time problems.

Although, the new solution approach has several appealing properties, the performance in parallel processing is poor. The performance is tested for OpenMP and it is observed that set computations are not efficient. Old solve is capable of good speedup computations yet Demona fails to improve the performance. Main reason is

the selected data structure. The solver might be revisited to focus on new programming aspects that will work on new data tree's which are multicore friendly.

To conclude, multiphyiscs problems can be analyzed within the framework of Demona which can create adaptive solution strategies without sacrificing the generality of model problems.

10.1. Future Work

Improvements on the solver can be categorized in two sections: numerical and physical. From numerical point of view, ability to work with unstructured grids could be implemented. In the thesis, triangles are considered only in finite elements, their use in CFD is also important. Use of unstructured grid also requires more general solvers like Algebraic Multigrid (AMG). Tensor-Krylov methods can be tested in multiphysics problems. Extension of the code for 3D is actually, straight forward. Only discrete model will be longer and set idea should be modified for the extra dimension. In finite elements, use of Element-By-Element (EBE) techniques is also essential. EBE deserves more attention in parallel processing especially in terms of preconditioning.

Current study investigated only incompressible laminar flow; however, there is a variety of problems both in compressible and turbulent flow regimes. Proposed solution techniques can be tested for these harder problems. Couplings between physics can also be investigated. Weak coupling is used in this thesis but monolithic approaches could also be examined.

Multiscale problems are another hot topic in scientific computing. As the name implies, the problem that is investigated is involved with more than one physical scale. In solidification problem for instance, the formation of the solid is related to the flow field and the boundary conditions. The way that the fluid cools to solid actually affects microscopic features of the structure. Improved mechanical properties require knowledge of both physical levels. Hence, this study on multiphysics can be extended towards multiscale problems at which different governing equations should be used for

different scales but their interaction should be well posed so that the model is more realistic.

APPENDIX A: QUICK Scheme

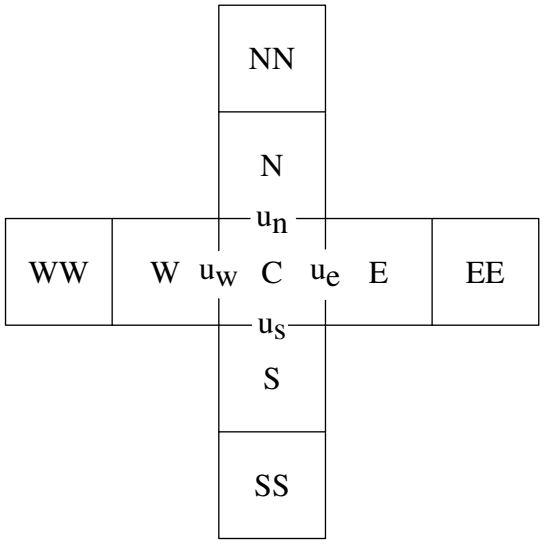


Figure A.1. Volumes used in QUICK Scheme

In Table A.1, nodes of the QUICK formulation are presented. D, U, UU mean downwind, upwind, and secondary upwind nodes, respectively. Designation of the location used in the table is shown in Figure A.1.

Table A.1. Nodes of the QUICK Scheme

face velocity	D $\left(\frac{3}{8}\right)$	U $\left(\frac{6}{8}\right)$	UU -$\left(\frac{1}{8}\right)$
$u_e > 0$	E	C	W
$u_e < 0$	C	E	EE
$u_w > 0$	C	W	WW
$u_w < 0$	W	C	E
$u_n > 0$	N	C	S
$u_n < 0$	C	N	NN
$u_s > 0$	C	S	SS
$u_s > 0$	S	C	N

APPENDIX B: Fast computation of the Diagonal of the Jacobian

For some nonlinear system of equations, the central variable can be separated from the nodal equation. For example consider the discretized Bratu problem (section 9.1.2.2) given below.

$$\frac{T_w + T_e - 2T_c}{dx^2} + \frac{T_n + T_s - 2T_c}{dy^2} + \lambda e^{T_c} = 0 \tag{B.1}$$

By examining the equation we can say that the nonlinearity is only for the central node and there are no cross terms. In order to calculate the diagonal of the Jacobian, Equation (B.1) can be modified like following:

$$\alpha \left[\frac{T_w + T_e}{dx^2} + \frac{T_n + T_s}{dy^2} \right] - \frac{2T_c}{dx^2} - \frac{2T_c}{dy^2} + \lambda e^{T_c} = 0 \tag{B.2}$$

Providing $\alpha = 0$ and performing a matrix-free multiplication with a vector, v composed of 1's only, we can extract the diagonal with just one function evaluation.[16] Keeping $\alpha = 1$ at the rest of the computations will yield the exact SNE. Same idea can also be used for Stream Function - Vorticity Approach given in section 2.1.1.1. Equations can be written in following discrete form where the first order partial derivatives in convective terms (i.e. the LHS of the second equation) are written with central differences.

$$\alpha \left[\frac{\psi_w + \psi_e}{dx^2} + \frac{\psi_n + \psi_s}{dy^2} \right] - \frac{2\psi_c}{dx^2} - \frac{2\psi_c}{dy^2} + \alpha\omega_c = 0 \tag{B.3a}$$

$$\alpha RE \left\{ \left(\frac{\psi_n - \psi_s}{2dy} \right) \left(\frac{\omega_e - \omega_w}{2dx} \right) - \left(\frac{\psi_e - \psi_w}{2dx} \right) \left(\frac{\omega_n - \omega_s}{2dy} \right) \right\} = \alpha \left[\frac{\omega_w + \omega_e}{dx^2} + \frac{\omega_n + \omega_s}{dy^2} \right] - \frac{2\omega_c}{dx^2} - \frac{2\omega_c}{dy^2} \tag{B.3b}$$

Same idea can be exploited, again. If the central variable is not separable, then one has two choices. The first one still exploits the same idea with a minor change: Pre-

[16] Actually two evaluations needed when referring to Equation 4.18. However, $\mathbf{F}(\mathbf{u})$ is already calculated at the start of the any Krylov routine. As a result, second evaluation is not an extra cost.

conditioner system is different then the real SNE. For example, if convective terms in Stream Function - Vorticity approach are discretized with forward differences, then the following formulation in (B.4b) will provide a diffusive preconditioner for $\alpha = 0$. It will be not the same diagonal of the Jacobian but the Jacobian of the Preconditioner.

$$\alpha \left[\frac{\psi_w + \psi_e}{dx^2} + \frac{\psi_n + \psi_s}{dy^2} \right] - \frac{2\psi_c}{dx^2} - \frac{2\psi_c}{dy^2} + \alpha\omega_c = 0 \tag{B.4a}$$

$$\begin{aligned} \alpha RE \left\{ \left(\frac{\psi_n - \psi_s}{2dy} \right) \left(\frac{\omega_e - \omega_c}{dx} \right) - \left(\frac{\psi_e - \psi_w}{2dx} \right) \left(\frac{\omega_n - \omega_c}{dy} \right) \right\} \\ = \alpha \left[\frac{\omega_w + \omega_e}{dx^2} + \frac{\omega_n + \omega_s}{dy^2} \right] - \frac{2\omega_c}{dx^2} - \frac{2\omega_c}{dy^2} \end{aligned} \tag{B.4b}$$

For this same idea, one can also try vary α such that $0 \leq \alpha \leq 1$. This is like having an analogy with MILU(α,0) where 0 denotes no fill-in's.

APPENDIX C: Demona Variables

Most important variables of Demona are summarized here. Details and additional variables are given in the manual. Overview of the subroutines used in the solver is depicted in Figure C.1.

C.1. Model Variables

physics Structure for the physics.

x Solution vector of the model. Pointer y is used in DEMONA to shorten its definition.

xPTO Solution vector of the model in pseudo time. Pointer yPTO is used in DEMONA to shorten its definition.

xRTO Solution vector of the model in real time. Pointer yRTO is used in DEMONA to shorten its definition.

tempx Temporary solution vector of the model.

setVec Set vector of the model.

C.2. Physics Variables

domain Structure for the domain.

x Solution vector of the physics.

xPTO Solution vector of the physics in pseudo time.

xRTO Solution vector of the physics in real time.

refXcoor x coordinate of the reference point.

refYcoor y coordinate of the reference point.

dFX Discretization type.

totalDof Total number of degree of freedoms.

totalNode Total number of degree of nodes.

discOrder discretization order

discOrder evaluation order

C.3. Domain Variables

set Structure for the set.

nodeMatrix Three dimensional node array.

dofMatrix Four dimensional index array.

nodeVector Vector representation of nodeMatrix.

dofVector Vector representation of dofMatrix.

allBC Number BC definitions.

numberOfGhostCells Number of ghost cells.

totalDof Total number of degree of freedoms.

totalNode Total number of degree of nodes.

x Solution vector of the domain.

xPTO Solution vector of the domain in pseudo time.

xRTO Solution vector of the domain in real time.

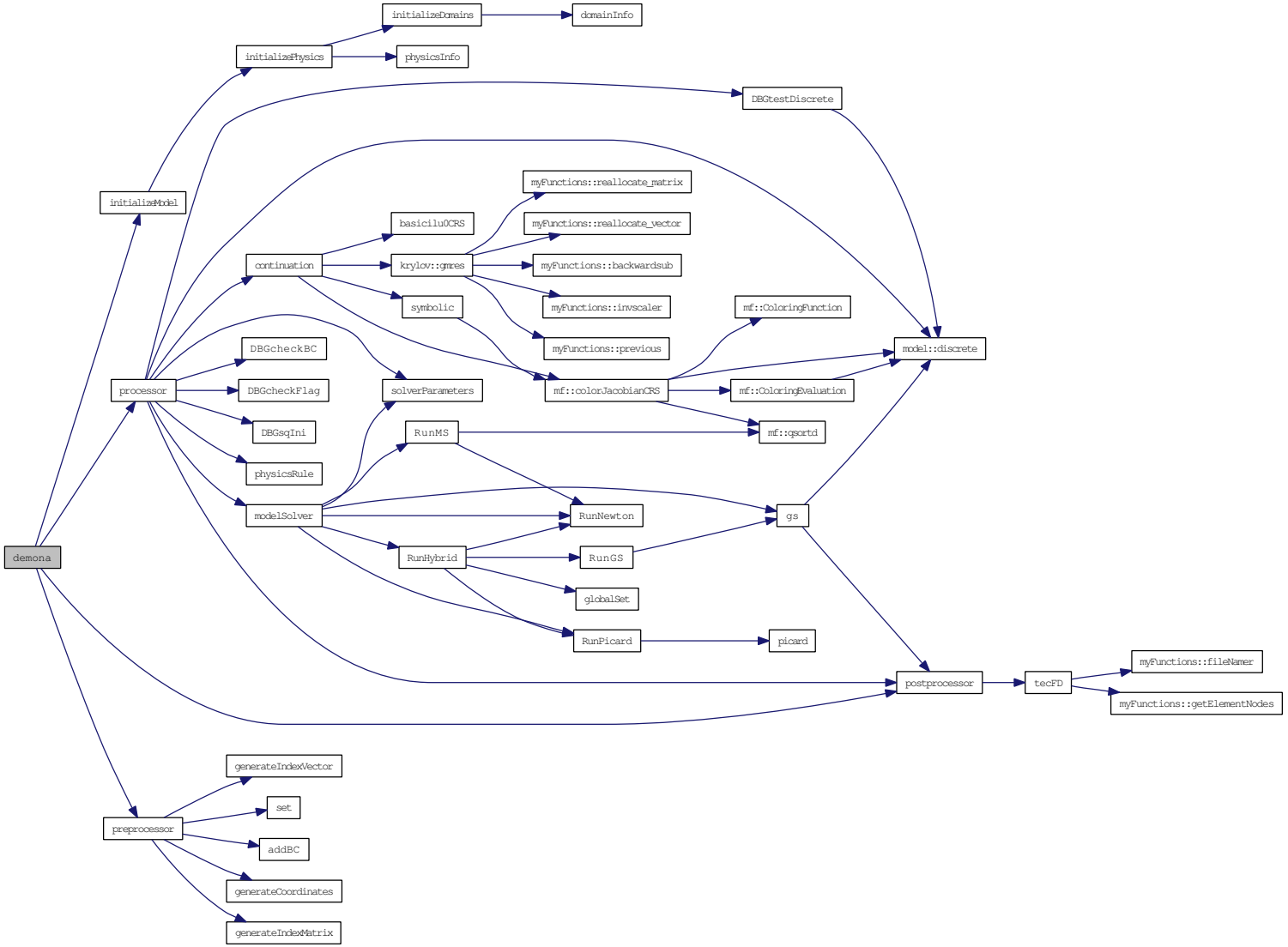


Figure C.1. DEMONA

APPENDIX D: Glossary

accelerator: advanced methods to be incorporated with basic iterative procedures that suffer from slow convergence rates. Krylov subspace methods are common examples of accelerators, for instance GMRES accelerates the Jacobi Method.

aspin: Additive Schwarz preconditioned inexact Newton's method utilizes domain decomposition. Resulting system acts as a nonlinear preconditioner.

domain decomposition: Division of the domain into some number of overlapping or non-overlapping domains. Different domains can be modeled with different equations. Also, it is well suited to parallel computation.

fas: Full approximation scheme is used to solve nonlinear equations directly by taking the advantage of the Multigrid idea. It is a good alternative to inexact Newton's methods however it can also utilized as a preconditioner of linear solves of Newton's method, as well.

inexact newton: Inexact Newton algorithm is used to solve systems of nonlinear equations if a proper initial guess is not at hand. It solves the updates approximately to a given tolerance which may vary in each step. Parameters define the linear tolerance which can be found by methods like backtracking or trust region.

inner iterations: in Newton's method.

matrix-free: Matrix-free methods avoid the formation of the Jacobian matrix when newton like methods are employed. It is used in conjunction with Krylov methods; methods that need matrix-vector products only except preconditioning.

multigrid: A method to improve the fine grid solution of pde's with coarse or a set of coarser grid corrections. Appropriate to use with smoothers. Can be applied on to linear, as well as nonlinear problems. Besides being a direct solution method, it can also act as a preconditioner.

multilevel: Utilization of different levels to improve the solution process. Levels can be interpreted in terms of the grid configurations.

multiscale: Approach to model problems that involve more than one scale. Turbulence is a good example of a multiscale method, after the solution of the global domain, the so called eddies can be handled in small scales.

non-conforming: Non-conforming domain decompositions occurs if same grid points on overlap domains are not shared. Information between domains is carried with interpolation operations.

non-overlapping: If decomposed domains only share interfaces (or surfaces in 3D), then they said to be non-overlapping. Each domain is a neighbor of surrounding domains

overlapping: if some portions of decomposed domains are intersected, then the domains are said to be overlapping. Fully overlapping occurs if a domain is a subset of some other domain.

outer iterations: In Newton's method.

preconditioner: methods that reduces the spectral radius of the system to be solved. It can be a matrix directly multiplied to the coefficients matrix or handling of a problem such a way that the stiffness of the system is reduced. For instance, Jacobi Method preconditiones GMRES.

smoothers: methods that eliminates the high frequency error but cannot reduce the low frequency (smooth) error. Smooth errors can be handled with Multigrid on a coarser grid at which they turn into high frequency error.

REFERENCES

Akin, J. E., 2005, *Finite Element Analysis with Error Estimators* , Elsevier, Amsterdam.

Anderson, J. D., 1995, *Computational Fluid Dynamics - The Basic with Applications* , McGraw Hill, New York.

Ansys, A., 2004, *ANSYS 9 Verification Manual* , ANSYS, http://www.ansys.com.

Bai, D. and A. Brandt, 1987, "Local mesh refinement multilevel techniques", *SIAM Journal on Scientific and Statistical Computing*, Vol. 8, pp. 109–134.

Bailey, C., G. A. Taylor, M. Cross and P. Chow, 1999, "Discretisation procedures for multi-physics phenomena", *Journal of Computational and Applied Mathematics*, Vol. 103, pp. 3–17.

Balay, S., K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and H. Zhang, 2010, *PETSc Web page* , Http://www.mcs.anl.gov/petsc.

Barrett, R., M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. V. der Vorst, 1994, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* , SIAM, Philadelphia, second ed.

Bathe, K. J., 1996, *Finite element procedures* , Prentice Hall, New Jersey.

Bejan, A., 2004, *Convection Heat Transfer* , Wiley, Hoboken, third ed.

Benzi, M., 2002, "Preconditioning techniques for large linear systems: A survey", *Journal of Computational Physics*, Vol. 182, pp. 418–477.

Bilgen, E., 2005, "Natural convection in cavities with a thin fin on the hot wall", *International Journal Of Heat And Mass Transfer*, Vol. 48, pp. 333–390.

Bird, R. B., R. C. Armstrong and O. Hassager, 1987, *Dynamics of Polymeric Liquids* , Wiley, New York, second ed.

Blazek, J., 2005, *Computational Fluid Dynamics:Principles and Applications* , Elsevier, Oxford, second ed.

Börgers, C. and O. B. Widlund, 1989, "A domain decomposition laplace solver for internal combustion engine modeling", *SIAM Journal on Scientific and Statistical Computing*, Vol. 10, pp. 211–226.

Brandt, A., 1977, "Multi–level adaptive solutions to boundary–value problems", *Mathematics of Computation*, Vol. 31, pp. 333–390.

Brandt, A., 1980, "Multilevel adaptive computations in fluid dynamics", *AIAA Journal*, Vol. 18, pp. 1165–1172.

Briggs, W. L., V. E. Hemson and S. F. McCormick, 2000, *A Multigrid Tutorial* , SIAM, Philadelphia, second ed.

Brown, P. N. and A. C. Hindmarsh, 1987, "Matrix-free methods for stiff systems of odes", *SIAM Journal of Numerical Analysis*, Vol. 24, pp. 610–638.

Brown, P. N. and Y. Saad, 1990, "Hybrid krylov methods for nonlinear-systems of equations", *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, pp. 450–481.

Broyden, C. G. and M. T. Vespuci, 2004, *Krylov Solvers for Linear Algebraic Systems* , Elsevier, Amsterdam.

Cai, X. C. and D. E. Keyes, 2002, "Nonlinearly preconditioned inexact newton algorithms", *SIAM Journal on Scientific Computing*, Vol. 24, pp. 183–200.

Cai, X. C. and M. Sarkis, 1999, "A restricted additive schwarz preconditioner for general sparse linear systems", *SIAM Journal on Scientific Computing*, Vol. 21, pp. 239–247.

Calo, V. M., N. F. Brasher, Y. Bazilevs and T. J. R. Hughes, 2008, "Multiphysics model for blood flow and drug transport with application to patient-specific coronary artery flow", *Computational Mechanics*, Vol. 43, pp. 161–177.

Chacon, L., D. A. Knoll and J. M. Finn, 2002, "An implicit, nonlinear reduced resistive mhd solver", *Journal of Computational Physics*, Vol. 178, pp. 15–36.

Chapman, S. J., 1998, *Fortran 90/95 for Scientists and Engineers* , McGraw-Hill, Boston.

Chen, K., 2005, *Matrix Preconditioning Techniques and Applications* , Cambridge University Press, Cambridge.

Chorin, A. J., 1967, "A numerical method for solving incompressible viscous flow problems", *Journal of Computational Physics*, Vol. 2, pp. 12–26.

Cook, R. D., 1995, *Finite Element Modeling for Stress Analysis* , Wiley, New York.

Date, A., 2005, *Introduction to Computational Fluid Dynamics* , Cambridge University Press, Cambridge.

Dehning, C. and K. Wolf, 2006, *Why do Multiphysics Analysis* , NAFEMS, Glasgow.

Dembo, R. S., S. C. Eisenstat and T. Steihaug, 1982, "Inexact newton methods", *SIAM Journal on Numerical Analysis*, Vol. 19, pp. 400–408.

Dennis, J. E. and R. B. Schnabel, 1983, *Numerical methods for unconstrained optimization and nonlinear equations* , Prentice Hall, New Jersey.

Dennis, J. E. and R. B. Schnabel, 1996, *Numerical methods for unconstrained optimization and nonlinear equations* , SIAM, Philadelphia.

Deuflhard, P., 2005, *Newton Methods for Nonlinear Problems* , Springer, Berlin, second ed.

Deuflhard, P., R. Freund and A. Walter, 1990, "Fast secant metohds for the iterative solution of large nonsymmetric linear systems", *Impact of Computing in Science and Engineering*, Vol. 2, pp. 244–276.

Donea, J., 1983, "Arbitrary lagrangian eulerian finite element methods", In T. Belytschko and T. Hughes ((editors)), *Computational methods for transient analysis*, Vol. 1 of *Mechanics and Mathematical Methods*, pp. 473–516. North-Holland, New York.

Douglas, C. C. and J. Douglas, 1993, "A unified convergence theory for abstract multigrid or multilevel algorithms, serial and parallel", *SIAM Journal on Numerical Analysis*, Vol. 30, pp. 136–158.

Drikakis, D. and W. Rider, 2005, *High-Resolution Methods for Incompressible and Low-Speed Flows* , Springer, Berlin.

Efstathiou, E. and M. J. Gander, 2003, "Why restricted additive schwarz converges faster than additive schwarz", *BIT Numerical Mathematics*, Vol. 43, pp. 945–959.

Eisenstat, S. C. and H. F. Walker, 1994, "Globally convergent inexact newton methods", *SIAM Journal on Optimization*, Vol. 4, pp. 393–422.

Eisenstat, S. C. and H. F. Walker, 1996, "Choosing the forcing terms in an inexact newton method", *SIAM Journal on Scientific and Statistical Computing*, Vol. 17, pp. 16–32.

Farhat, C. and M. Lesoinne, 2000, "Two effcient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems", *Computer Methods In Applied Mechanics And Engineering*, Vol. 182, pp. 499–515.

Farhat, C., M. Lesoinne and P. Le Tallec, 1998, "Load and motion transfer algorithms for fluid/structure interaction problems with non-matching discrete interfaces: Momentum and energy conservation, optimal discretization and application to aeroelasticity", *Computer Methods In Applied Mechanics And Engineering*, Vol. 157, pp. 95–114.

Felippa, C. A., K. C. Park and C. Farhat, 2001, "Partitioned analysis of coupled mechanics systems", *Computer Methods In Applied Mechanics And Engineering*, Vol. 190, pp. 3247–3270.

Ferziger, J. H. and M. Peric, 2002, *Computational Methods for Fluid Dynamics* , Springer, Berlin, third ed.

Fletcher, C. A. J., 1991, *Computational techniques for fluid dynamics* , Springer, Berlin.

Fung, Y. C., 1969, *The Theory of Aeroelasticity* , Dover, New York, revised republication of the 1955 ed.

Gaskell, P. H. and A. K. C. Lau, 1988, "Curvature-compensated convective transport: Smart, a new boundedness preserving transport algorithm", *International Journal For Numerical Methods In Fluids*, Vol. 8, pp. 617–641.

Gebremedhin, A. H., F. Manne and A. Pothen, 2005, "What color is your jacobian? graph coloring for computing derivatives", *SIAM Review*, Vol. 47, pp. 629–705.

Ghia, U., K. N. Ghia and C. T. Shin, 1982, "High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method", *Journal of Computational Physics*, Vol. 48, pp. 387–411.

Greenbaum, A., 1997, *Iterative Methods for Solving Linear Systems* , SIAM, Philadelphia.

Griewank, A. and A. Walther, 2008, *Evaluating Derivatives:Principles and Techniques of Algorithmic Differentiation* , SIAM, Philadelphia, second ed.

Gropp, W., D. E. Keyes, L. C. McInnes and M. D. Tidriri, 2000, "Globalized newton-krylov-schwarz algorithms and software for parallel implicit cfd", *International Journal of High Performance Computing Applications*, Vol. 14, pp. 102–136.

Guillaume, P., A. Huard and C. L. Calvez, 2003, "A block constant approximate inverse for preconditioning large linear systems", *SIAM Journal on Matrix Analysis and Applications*, Vol. 24, pp. 822–851.

Guruswamy, G. P., 2002, "A review of numerical fluids/structures interface methods for computations using high-fidelity equations", *Computers and Structures*, Vol. 80, pp. 31–41.

Hackbusch, W., 1984, "Parabolic multi–grid methods", In R. Glowinski and J. Lions ((editors)), *Computing Methods in Applied Sciences and Engineering VI*, pp. 189–197. North–Holland, Amsterdam.

Hackbusch, W., 1985, *Multi-grid methods and applications* , Springer-Verlag, Berlin, first ed.

Harlow, F. H. and J. E. Welch, 1965, "Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface", *Physics of Fluids*, Vol. 8, pp. 2182–&.

Heroux, M. A., R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams and K. S. Stanley, 2005, "An overview of the trilinos project", *ACM Transactions on Mathematical Software*, Vol. 31, pp. 397–423.

Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker and C. S. Woodward, 2005, "Sundials: Suite of nonlinear and differential/algebraic equation solvers", *ACM Transactions on Mathematical Software*, Vol. 31, pp. 363–396.

Hirt, C. W. and B. D. Nichols, 1981, "Volume of fluid (vof) method for the dynamics of free boundaries", *Journal of Computational Physics*, Vol. 39, pp. 201–225.

Hoffmann, K. A. and S. T. Chiang, 2000, *Computational Fluid Dynamics for Engineers* , Engineering Education System, Austin.

Horton, G., 1992, "The time-parallel multigrid method", *Communications In Applied Numerical Methods*, Vol. 8, pp. 585–595.

Hübner, B., E. Walhorn and D. Dinkler, 2004, "A monolithic approach to fluidstructure interaction using spacetime finite elements", *Computer Methods In Applied Mechanics And Engineering*, Vol. 193, pp. 2087–2104.

Hughes, T. J. R., I. Levit and J. Winget, 1983, "An element-by-element solution algorithm for problems of structural and solid mechanics", *Computer Methods In Applied Mechanics And Engineering*, Vol. 36, pp. 241–254.

Ilıcak, M., A. Ecder and E. Turan, 2007, "Operator splitting techniques for the numerical analysis of natural convection heat transfer", *International Journal of Computer Mathematics*, Vol. 84, pp. 783–793.

Incropera, F. P. and D. P. DeWitt, 2001, *Introduction to Heat Transfer* , Wiley, New York, fourth ed.

Issa, R. I., 1985, "Solution of the implicitly discretized fluid flow equations by operator-splitting", *Journal of Computational Physics*, Vol. 62, pp. 40–65.

Kaczmarz, S., 1937, "Angenherte auflsung von systemen linearer gleichungen", *Bulletin International de l'Academie Polonaise des Sciences et des Lettres, series A*, Vol. 35, pp. 335–357.

Karypis, G. and V. Kumar, 1998, "A fast and high quality multilevel scheme for partitioning irregular graphs", *SIAM Journal on Scientific Computing*, Vol. 20, pp. 359–392.

Kelley, C. T., 1995, *Iterative Methods for Linear and Nonlinear Equations* , SIAM, Philadelphia.

Keyes, D. E. and W. Gropp, 1987, "A comparison of domain decomposition techniques for elliptic partial differential equations", *SIAM Journal on Scientific Computing*, Vol. 8, pp. 166–202.

Knoll, D. A., 1998, "An improved convection scheme applied to recombining divertor plasma flows", *Journal of Computational Physics*, Vol. 142, pp. 473–488.

Knoll, D. A. and D. E. Keyes, 2004, "Jacobian–free newton-krylov methods: A survey of approaches and applications", *Journal of Computational Physics*, Vol. 193, pp. 357–397.

Knoll, D. A. and W. J. Rider, 1999, "A multigrid preconditioned newton-krylov method", *SIAM Journal on Scientific Computing*, Vol. 21, pp. 691–710.

Kron, G., 1953, "A set of principles to interconnect the solutions of physical systems", *Journal of Applied Physics*, Vol. 24, pp. 965–980.

Lanzkron, P. J., D. J. Rose and J. T. Wilkes, 1996, "An analysis of approximate nonlinear elimination", *SIAM Journal of Scientific Computing*, Vol. 17, pp. 538–559.

Law, K. H., 1986, "A parallel finite element solution method", *Computers and Structures*, Vol. 23, pp. 845–858.

Lawrence Livermore National Laboratory, 2010, *Hypre-high performance preconditioners* , Http://www.llnl.gov/CASC/hypre/.

Leonard, B. P., 1979, "Accurate convective modelling procedure based on quadratic upstream interpolation", *Computer Methods in Applied Mechanics and Engineering*, Vol. 19.

Leonard, B. P., 1988, "Simple high-accuracy resolution program for convective modelling of discontinuities", *Computer Methods in Applied Mechanics and Engineering*, Vol. 8.

Lions, J. L., Y. Maday and G. Turinici, 2001, "A parareal in time discretization of pde's", *Comptes Rendus De L Academie Des Sciences Serie I-Mathematique*, Vol. 332, pp. 1661–668.

Liu, J. G. and C. Wang, 2004, "High order finite difference methods for unsteady incompressible flows in multi-connected domains", *Computers & Fluids*, Vol. 33, pp. 223–255.

Mandan, J. C., L. Agrawal and A. G. Marathe, 2000, "Computations of incompressible flows with natural convection using pseudocompressibility approach", *Journal Of Thermophysics And Heat Transfer*, Vol. 14, pp. 606–609.

Martin, J. C. and W. J. Moyce, 1952, "An experimental study of the collapse of liquid columns on a rigid horizontal plane", *Philos. Trans. Roy. Sot. London Ser. A*, Vol. 244, pp. 312–&.

Martins, J. R. R. A., P. Sturdza and J. J. Alonso, 2003, "The complex-step derivative approximation", *ACM Transactions on Mathematical Software*, Vol. 29, pp. 245–262.

Massarotti, N., P. Nithiarasu and O. C. Zienkiewicz, 1998, "Characteristic-based-split (cbs) algorithm for incompressible flow problems with heat transfer", *International Journal of Numerical Methods for Heat and Fluid Flow*, Vol. 8, pp. 969–990.

Mathew, T., 2008, *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations* , Springer, Berlin.

McCormick, S. F., 1989, *Multilevel adaptive Methods for Partial Differential Equations* , SIAM, Philadelphia.

Mohamad, A. A., 1998, "Benchmark solution for unsteady state cfd problems", *Numerical Heat Transfer, Part A: Applications*, Vol. 34, pp. 653–672.

Mousseau, V. A., D. A. Knoll and W. J. Rider, 2000, "Physics-based preconditioning and the newton-krylov method for non-equilibrium radiation diffusion", *Journal of Computational Physics*, Vol. 160, pp. 743–765.

Mpcci, 2010, *Mesh-based parallel Code Coupling Interface* , http://www.mpcci.de.

Mulder, W. A. and B. van Leer, 1985, "Experiments with implicit upwind methods for the euler equations", *Journal of Computational Physics*, Vol. 59, pp. 232–246.

Nicholson, D. W., 2003, *Finite element analysis : thermomechanics of solids* , CRC Press, Boca Raton.

Nouromid, B. and B. N. Parlett, 1985, "Element preconditioning using splitting techniques", *SIAM Journal on Scientific and Statistical Computing*, Vol. 6, pp. 761–770.

Papadrakakis, M. and M. C. Dracopoulos, 1991, "A global preconditioner for the element-by-element solution methods", *Computer Methods in Applied Mechanics and Engineering*, Vol. 88, pp. 275–286.

Patankar, S. V., 1980, *Numerical Heat Transfer and Fluid Flow* , Hemisphere Publishing, Washington.

Pawlowski, R. P., J. N. Shadid, J. P. Simonis and H. F. Walker, 2006, "Globalization techniques for newtonkrylov methods and applications to the fully coupled solution of the navierstokes equations", *SIAM Review*, Vol. 48, pp. 700–721.

Pepper, D. W. and J. C. Heinrich, 2006, *The finite element method : basic concepts and applications* , Taylor & Francis, New York, second ed.

Pernice, M. and H. F. Walker, 1998, "Nitsol: A newton iterative solver for nonlinear systems", *SIAM Journal on Scientific and Statistical Computing*, Vol. 19, pp. 302–318.

Piperno, S., C. Farhat and B. Larrouturou, 1995, "Partitioned procedures for the transient solution of coupled aeroelastic problems part i: Model problem, theory and

two-dimensional application", *Computer Methods in Applied Mechanics and Engineering*, Vol. 124, pp. 79–112.

Przemieniecki, J. S., 1963, "Matrix structural analysis of substructures", *AIAA Journal*, Vol. 1, pp. 138–147.

Qin, N., D. K. Ludlow and S. T. Shaw, 2000, "A matrix-free preconditioned newton:gmres method for unsteady navierstokes solutions", *International Journal for Numerical Methods in Fluids*, Vol. 33, pp. 223–248.

Quarteroni, A. and A. Valli, 1999, *Domain Decomposition Methods for Partial Differential Equations* , Oxford University Press, Oxford.

Ramshaw, J. D. and V. A. Mousseau, 1990, "Accelerated artificial compressibility method for steady-state incompressible flow calculations", *Computers and Fluids*, Vol. 18, pp. 361–367.

Reisner, J., A. Wyszogrodzki and D. Knoll, 2003, "An efficient physics-based preconditioner for the fully implicit solution of small-scale thermally driven atmospheric flows", *Journal of Computational Physics*, Vol. 189, pp. 30–44.

Reynolds, O., 1886, "On the theory of lubrication and its application to mr. beauchamp tower's experiments, including an experimental determination of the viscosity of olive oil", *Philosophical Transactions of the Royal Society of London*, Vol. 177, pp. 157–234.

Roache, P. J., 1998, *Fundamentals of Computational Fluid Dynamics* , Hermosa Publishers, Albuquerque.

Rogers, S. E., D. Kwak and C. Kiris, 1991, "Steady and unsteady solutions of the incompressible navier-stokes equations"', *AIAA Journal*, Vol. 29, pp. 603–610.

Rüde, U., 1993, "Fully adaptive multigrid methods", *SIAM Journal on Numerical Analysis*, Vol. 30, pp. 230–248.

Saad, Y., 1993, "A flexible inner-outer preconditioned gmres algorithm", *SIAM Journal on Scientific and Statistical Computing*, Vol. 14, pp. 461–469.

Saad, Y., 2003, *Iterative Methods for Sparse Linear Systems* , SIAM, Philadelphia, second ed.

Saad, Y., 2010, *Sparskit* , Http://www-users.cs.umn.edu/ saad/software/SPARSKIT/.

Saad, Y. and M. H. Schultz, 1986, "A generalized minimal residual algorithm for solving nonsymmetric linear systems", *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, pp. 856–869.

Saad, Y. and H. A. van der Vorst, 2000, "Iterative solution of linear systems in the 20th century", *Journal of Computational and Applied Mathematics*, Vol. 123, pp. 1–33.

Schaefer, M., 2006, *Computational Engineering Introduction to Numerical Methods* , Springer, Berlin.

Schwarz, H. A., 1869, "Ueber einige abbildungsaufgaben", *Gesammelte Mathematische Abhandlungen*, Vol. 11, pp. 65–83.

Shadid, J. N., R. S. Tuminaro and H. F. Walker, 1997, "An inexact newton method for fully coupled solution of the navier-stokes equations with heat and mass transport", *Journal of Computational Physics*, Vol. 137, pp. 155–185.

Sheu, T. W. H., M. M. T. Wang and S. F. Tsai, 2000, "Element-by-element parallel computation of incompressible navier-stokes equations in three dimensions", *SIAM Journal on Scientific and Statistical Computing*, Vol. 21, pp. 1387–1400.

Simoncini, V. and D. B. Szyld, 2007, "Recent computational developments in krylov subspace methods for linear systems", *Numerical Linear Algebra with Applications*, Vol. 14, pp. 1–59.

Smith, B., P. Bjorstad and W. Gropp, 1996, *Domain Decomposition* , Cambridge Publications, Philadelphia.

Smith, I. M. and D. V. Griffiths, 2004, *Programming the Finite Element Method* , Wiley, Hoboken, fourth ed.

Soh, W. Y., 1987, "Time-marching solution of incompressible navierstokes equations for internal flow", *Journal of Computational Physics*, Vol. 70, pp. 232–252.

Strikwerda, J. C., 1989, *Finite difference schemes and partial differential equations* , Wadsworth & Brooks, Pacific Grove.

Strikwerda, J. C. and D. Scarbnick, 1993, "A domain decomposition method for incompressible viscous flow", *SIAM Journal on Scientific and Statistical Computing*, Vol. 14, pp. 49–67.

Tannehill, J. C., D. A. Anderson and R. H. Pletcher, 1997, *Computational Fluid Mechanics and Heat Transfer* , Taylor and Francis, Philadelphia, second ed.

Tezduyar, T. E. and M. Behr, 1992, "A new strategy for finite element computations involving moving boundaries and interfaces - the deforming-spatial-domain/spcae-time procedure: I. the concept and the preliminary numerical tests", *Computer Methods in Applied Mechanics and Engineering*, Vol. 94, pp. 339–351.

Tezduyar, T. E. and J. Liou, 1989, "Grouped element-by-element iteration schemes for incompressible flow computations", *Computer Physics Communications*, Vol. 53, pp. 441–453.

Timoshenko, S. P. and J. N. Goodier, 1970, *Theory of Elasticity* , McGraw-Hill, New York, third ed.

Toselli, A. and O. Widlund, 2005, *Domain Decomposition Methods - Algorithms and Theory* , Springer, Berlin.

Trottenberg, U., C. W. Oosterlle and A. Schueller, 2000, *Multigrid* , Academic Press, London.

Turner, K. and H. F. Walker, 1992, "Efficient high accuracy solutions with gmres(m)", *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, pp. 815–825.

Ubbink, O., 1997, *Numerical prediction of two fluid systems with sharp interfaces* , University of London, London.

Ugural, A. C. and S. K. Fenster, 1994, *Advanced Strength and Applied Elasticity* , Prentice Hall, New Jersey, third ed.

van der Vorst, H. A., 1992, "Bi cgstab: A fast and smoothly converging variant of bicg for the solution of nonsymmetric linear systems", *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, pp. 631–644.

van der Vorst, H. A., 2003, *Iterative Krylov Methods for Large Linear Systems* , Cambridge University Press, Cambridge.

Vanka, S., 1986, "Block-impicit multigrid solution of navier-stokes equations in primitive variables", *Journal of Computational Physics*, Vol. 65, pp. 138–158.

Venner, C. H. and A. A. Lubrecht, 2000, *Multilevel Methods in Lubrication* , Elsevier, Amsterdam.

Versteeg, H. K. and W. Malalasekera, 2007, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method* , Prentice Hall, New York, second ed.

Wall, W. A., P. Gamnitzer and A. Gerstenberger, 2008, "Fluid-structure interaction approaches on fixed grids based on two different domain decomposition ideas", *International Journal of Computational Fluid Dynamics*, Vol. 22, pp. 411–427.

Winget, J. M. and T. J. R. Hughes, 1985, "Solution algorithms for nonlinear transient heat conduction analysis employing element-by-element iterative strategies", *Computer Methods in Applied Mechanics and Engineering*, Vol. 52, pp. 711–815.

Wohlmuth, B. I., 2001, *Discretization Methods and Iterative Solvers Based on Domain Decomposition* , Springer, Berlin.

Yotov, I., 2001, "A multilevel newtonkrylov interface solver for multiphysics couplings of flow in porous media", *Numerical Linear Algebra With Applications*, Vol. 8, pp. 551–570.