A CUBIC SPLINE BASED LOCAL DEFECT CORRECTION APPROACH FOR THE SIMULATION OF INCOMPRESSIBLE VISCOUS FLOWS

A Project Report Submitted

for the Course

MA499 Project II

by

Aditya Gupta | Vishal Kumar

(130123051 | 130123043)

to the

DEPARTMENT OF MATHEMATICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

April 2017

CERTIFICATE

This is to certify that the work contained in this project report entitled "**A cubic spline based local defect correction approach for the simulation of incompressible viscous flow**" submitted by **Aditya Gupta** (**Roll No.: 130123051**) and **Vishal Kumar** (**Roll No.: 130123043**) to Department of Mathematics, Indian Institute of Technology Guwahati towards the requirement of the course **MA499 Project II** has been carried out by them under my supervision.

Guwahati - 781 039 (Dr. Jiten C. Kalita)

April 2017 Project Supervisor

ABSTRACT

The project is focused on developing a numerical scheme by using local defect correction methods in conjunction with cubic splines for interpolating boundary conditions for the simulation of incompressible viscous flows.

In this project, we use the approach of solving a discretization scheme over the whole domain (coarse grid) and using the information to set boundary conditions for a small part of the domain (fine grid). We then solve the discretization over the fine grid to obtain more accurate results for the fine grid and update the coarse grid. We repeat this process until convergence. This saves us the immense computation overhead of using a single grid for the whole domain with small step size. In the first part of our project we used this method to obtain solutions for the steady state heat conduction problem and the Lid Driven Cavity problem.

For solving the Lid Driven cavity problem, we used a discretization scheme based on the $\psi - \omega$ formulation of the Navier-Stokes equations. The scheme required updating both ψ and ω parameters. We tried our back and forth approach with updating only ψ between interpolations, but still obtained results in excellent agreement with Ghia's benchmark results. Updating both ψ and ω was not only cumbersome, it also provided little improvement on our previous result.

So, in the second part we instead use a discretization scheme that only takes into account ψ and velocities u and v at a given point, eliminating the need for updating ω. We will use the BiCGStab as our iterative solver, unlike Gauss-Seidel in the first part.

Contents

List of Figures

Chapter 1

Introduction

The boundary value problems (BVP) we will explore in this paper show rapid variations in their solution in some parts of the domain. If we use a single uniform grid for the whole domain this we will require a very fine grid leading to a lot of unnecessary computations. Local Defect correction (LDC) offers a way to deal with such problems. It involves using different discretization schemes for the local and global problem, instead of using a single set of difference equations. In our case, we use a coarser grid as a global discretization scheme, which is combined with a finer grid (having smaller step size) used for the sub-domain.

1.1 The Local Defect Correction method

In this section we give a general introduction to the Local Defect Correction method.[1] We may have a boundary value problem with domain Ω, like

$$Lu = f \tag{1.1}$$

with boundary condition

$$Bu = g \tag{1.2}$$

on

$$\Gamma = \delta\Omega \tag{1.3}$$

where for example $L = -\Delta$. A discretization can be given by

$$L_h u_h = f_h \tag{1.4}$$

where h is grid parameter. The defect correction iteration for the given problem shall be

$$u_0^h = f_h L_h^1 \tag{1.5}$$

$$u_h^{i+1} = u * i_h - L_h^{-1}(L_h^{'} u_h^i - f_h^{'}) \tag{1.6}$$

for $i = 1, 2, ...$ The first step of this iteration can be written as

$$u_h^1 = u_h^0 - L_h^{-1}(L_h^{'} u_h^0 - f_h^{'}) \tag{1.7}$$

where d_h^0 is the defect $d_h^0 = L_h^{'} u_h^0 - f_h^{'}$ Let $\omega \Subset \Omega$ be a sub-region. Its characteristics function is

$$\chi_\omega(x) = \begin{pmatrix} 0 & x\epsilon\Omega\backslash\omega \\ 1 & x\epsilon\omega \end{pmatrix} \tag{1.8}$$

The local defect is

$$d_0^h = \chi_\omega(L_h^{'} u_h - f_h^{'}) \tag{1.9}$$

Together the equations (1.7) and (1.9) give us the local defect correction.

$$u_h^1 - u_h^* = L_h^{-1}\chi_\omega(L_h - L_h^{'})(u_h^0 - u_h^*) - L_h^{-1}\chi_\omega(L_h^{'} u_h^* - f'_h) + L_1^h(1 - \chi_\omega)L_h^{'}(u_h^0 - u_h^*) \tag{1.10}$$

The local discretization for our original BVP can be denoted by

$$L_h u_{h,\Omega} = f_{h,\Omega} \tag{1.11}$$

including the boundary conditions on Γ. The underlying grid is Ω_h . The local discretization problem in the sub-domain $\omega \subset \Omega$ can be written as $Lu_\omega = f$ If $\Gamma_0 = \overline{w} \cap \Gamma$ is not empty, u_ω must satisfy in Γ_0

$$Bu_\omega = g \tag{1.12}$$

On the interior part $\Gamma_1 = \overline{\omega} \backslash \Gamma$ we have

$$u_\omega = u_\Omega \tag{1.13}$$

The local discretization is given by

$$L^{'}_{h'} u^{'}_{h',\omega} = f^{'}_{h',\omega} \tag{1.14}$$

$L^{'}$ and $u^{'}$ indicate the discretization may be different from L_h and u_h. $h^{'}$ subscript means a different step size from h. The discretization of (1.13) is given by:

$$u^{'}_{h',\omega} = \gamma u_{h,\Omega} \tag{1.15}$$

1.2 An Algorithm for LDC

An algorithm to combine the local and global discretization[2] is given by
Start:

$$f^0_{h,\Omega} = f_{h,\Omega} \tag{1.16}$$

obtained from the global discretization.

Given $f^i_{h,\Omega}$: Compute the solution to the coarse grid problem to initialize $u^i_{h,\omega}$ on Ω_h

$$L_h u^i_{h,\omega} = f^i_{h,\Omega} \tag{1.17}$$

Compute the boundary values on $\Gamma_{1,h'}$

$$g^i = \gamma u^i_{h,\Omega} \tag{1.18}$$

The preceding two equations give the solution to the coarser grid problem. Solve this finer grid problem in $\omega_{h'}$

$$L^{'}_{h'} u^{'}_{h',\omega} = f^{'}_{h',\omega} \tag{1.19}$$

and

$$g^i = \gamma u^i_{h,\omega} \tag{1.20}$$

We have sub-grids $\omega^{"}_h \Subset \omega^{'}_h \Subset \omega$

Interpolate $u^i_{h,\omega}$ on $\omega^{'}_h = \omega^{'} \epsilon \Omega_h$

$$\overline{u}_{h,\omega'} = \pi u_{h',\omega} \tag{1.21}$$

Compute the defect in $w^{"}_h$

$$d_{h,w^{"}} = L_h \overline{u}_{h,\omega'} - f_{h,\Omega} \tag{1.22}$$

Now for the next iteration, define in the coarser grid

$$f^{i+1}_{h,\Omega} = f_{h,\Omega} + \chi_{\omega^{"}} d_{h,w^{"}} \tag{1.23}$$

1.3 Background

A lot of problems have already been experimented with the inclusion of LDC methods, for example diffusion equations [3] and combustion equations [4], to name a few. Also, [5] provides a close correspondence between LDC and Fast Adaptive Composite Grid (FAC) methods [6]. [7] provide a local defect correction technique for time-dependent problems, suitable for solving partial differential equations characterized by high activity.

Chapter 2

Interpolation

In our discretization scheme, we will use the cubic spline algorithm[8] to find the boundary values for the finer grid using values from the coarser grid. Piecewise polynomial approximation is used to approximate the value of a function on an interval using a set of given data points.

2.1 Cubic Splines

Definition 2.1.1. Given a function f defined on $[a, b]$ and set of n+1 points such that

$$a < x_0 < x_1 < x_2 ... < x_n < b$$

A cubic spline interpolant S for the function f satisfies the following conditions:

1. $S(x)$ is a cubic polynomial, denoted $S_j(x)$ for every jth interval $[x_j, x_{j+1}]$, $j = 0, 1, 2, ..., n-1$

2. $S_j(x_{j+1}) = S_j(x_{j+1})$ for all $j = 0, 1, 2, ..., n-2$

3. $S_j^{'}(x_{j+1}) = S_j^{'}(x_{j+1})$ for all $j = 0, 1, 2, ..., n-2$

4. $S_j^{''}(x_{j+1}) = S_j^{''}(x_{j+1})$ for all $j = 0, 1, 2, ..., n-2$

5. One of the following boundary conditions is satisfied:

   (a) $S^{''}(x_0) = S^{''}(x_n)$ (free boundary)

   (b) $S^{'}(x_0) = f^{'}(x_0)$ and $S^{'}(x_n) = f^{'}(x_n)S^{'}(x_n)$ (clamped boundary)

2.2 Construction of a cubic spline

We take the general form of the cubic spline interpolant for each interval as follows:

$$a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \tag{2.1}$$

Clearly, the we need to determine $4n$ constants to construct the interpolant based on n nodes. We apply the conditions mentioned in the definition of the cubic spline to each S_j to find the values of corresponding a_j, b_j, c_j, d_j for every $j = 0, 1, ..., n-2$. Since $S_j(x_j) = a_j = f(x_j)$, we can use condition 3 to get

$$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) \tag{2.2}$$

for every $j = 1, ..., n-1$ we introduce the notation

$$h_j = x_{j+1} - x_j \tag{2.3}$$

We can define $a_n = f(x_n)$, giving us the equation

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \tag{2.4}$$

which holds for all $j = 0, 1, ..., n-1$ Similarly we can define $b_n = S^{'}(x_n)$ (trivial). If we take the first derivative of $S_j(x)$ we see

$$S_j^{'}(x) = b_j + 2c_j(x - x_j) + 3d_j(x - x_j)^2 \tag{2.5}$$

Hence $S_j^{'}(x_j) = b_j$ for all $j = 0, 1, ..., n-1$. Using condition 4 we get for all $j = 0, 1, ..., n-1$

$$b_{j+1} = b_j + 2c_jh_j + 3d_jh_j^2 \tag{2.6}$$

Similarly we can define $c_n = S^{"}(x_n)/2$ and use condition 5 to get for all j = 0,1, ... ,n-1 the following equation

$$c_{j+1} = c_j + 3d_jh_j \tag{2.7}$$

From the last three equations, for all j = 0,1, ... ,n-1 we get

$$a_{j+1} = \frac{a_j + b_jh_j + h_j^2(2c_j + c_{j+1})}{3} \tag{2.8}$$

and

$$b_{j+1} = b_j + h_j(c_j + c_{j+1}) \tag{2.9}$$

By reducing index j by 1 we get

$$b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j) \tag{2.10}$$

Solving for b_j we get

$$b_j = \frac{(a_{j+1} - a_j)}{h_j} - \frac{(2c_{j+1} + c_j)h_j}{3} \tag{2.11}$$

We can get b_{j-1} by reducing the index j in the preceding equation

$$b_{j-1} = \frac{(a_j - a_{j-1})}{h_{j-1}} - \frac{(2c_j + c_{j-1})h_{j-1}}{3} \tag{2.12}$$

Substituting the values for b_j and b_{j-1} gives us for $all j = 1, ..., n-1$ this set of linear equations

$$h_{j-1} + c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = 3\frac{(a_{j+1} - a_j)}{h_j} - 3\frac{(a_j - a_{j-1})}{h_{j-1}} \tag{2.13}$$

Notice that in this system we already know h_j and a_j for $j = 1, 2, ..., n-1$. Once c_j for $j = 1, 2, ..., n-1$ are calculated the remaining coefficients can be easily determined by the equations listed above. The following section demonstrates that these coefficients can indeed be uniquely calculated if one of the boundary conditions mentioned in the definition of the cubic spline is imposed.

2.3 Natural splines

The following theorem proves that for any given set of n points, there will always exist a natural spline interpolant.

Theorem 2.3.1. *If a function f is defined at nodes $x_0, x_1, ..., x_n$ such that $a < x_0 < x_1 ... < x_n < b$, then there exists a unique natural spline interpolant S for the given function passing through the points $(x_0, f(x_0)), (x_1, f(x_1))...(x_n, f(x_n))$ satisfying the natural boundary condition $S^{''}(a) = 0$ and $S^{''}(b) = 0$.*

Proof. Imposing the boundary condition gives us

$$c_n = S^{''}(x_n) = 0 \tag{2.14}$$

$$S''(x_0) = 0 = 2c_0 + 6d_0(x_0 - x_0) \tag{2.15}$$

Hence $c_0 = x_0$. We get a system of linear equations of the form $AX = B$ where

$$A = \begin{bmatrix} 1 & 0 & 0 & . & . & . & . & . & 0 \\ h_0 & 2(h_0+h_1) & h1 & 0 & . & . & . & . & 0 \\ 0 & h_1 & 2(h_1+h_2) & h_2 & 0 & . & . & . & 0 \\ 0 & 0 & . & . & . & . & . & . & 0 \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & . & . & . & . & . & 0 & 0 & 1 \end{bmatrix}$$

Notice that A has three non-zero diagonals.

$$B = \begin{bmatrix} 0 \\ 3(a_2-a_1)/h_1 \\ . \\ . \\ 3\frac{a_n-a_{n-1}}{h_{n-1}-3(a_{n-1}-a_{n-2})/h_{n-2}} \\ 0 \end{bmatrix}$$

and

$$X = \begin{bmatrix} c_0 \\ c_1 \\ . \\ . \\ c_n \end{bmatrix}$$

We can see that matrix A is clearly diagonally dominant, i.e. in every row the diagonal entry is greater than the sum of all other elements in the same row. Hence, by Levy–Desplanques theorem[9] we can conclude that the matrix A is singular and the system $AX = B$ has a unique solution. Hence, the theorem is proved. The natural spline[8] has been used in our implementation of the discretization processes. $\square$

Chapter 3

Lid Driven Cavity(LDC) for incompressible fluids

Having gained confidence in our implementation of local defect correction in conjunction with the steady state heat conduction equations in the first part of our project, we moved forward to the famous LDC problem(figure 3.1)

3.1 Problem Description

The problem essentially consists of numerically solving the 2D NS equations in a square cavity. The left, right and bottom walls of the cavity are stationary while the top wall moves from the left to right (in the positive x-direction) with a velocity U. Because of the no-slip condition for a viscous flow, the x- and y-velocities of the fluid are zero at the left, right and the bottom walls. The no-slip condition at the top wall requires that y-velocity of the fluid is zero but the x-velocity equals the wall velocity U. As a consequence, as the top wall moves from the left to right, so does the fluid in contact with that wall. Because of viscosity, this momentum in the x-direction diffuses in the transverse direction to the interior of the cavity and after some time the fluid in the whole cavity is set into motion (note that if the fluid had no viscosity, the wall motion would not have affected the fluid in the cavity at all and it would have remained stationary).

In this simple geometric setting, highly complex flow patterns arise as the Reynolds number (Re) increases. At relatively low Reynolds numbers, a large primary vortex (or eddy) is seen to form. As Reynolds number increases smaller vortices, called secondary, tertiary, quaternary etc. start developing at various corners of the cavity.

Theoretically, at high Reynolds numbers, an infinite series of vortices, ranging from the primary eddy (having the dimension of the cavity) to the very very small ones, are generated. Multiplicity of scales is characteristic of high-Re flows.

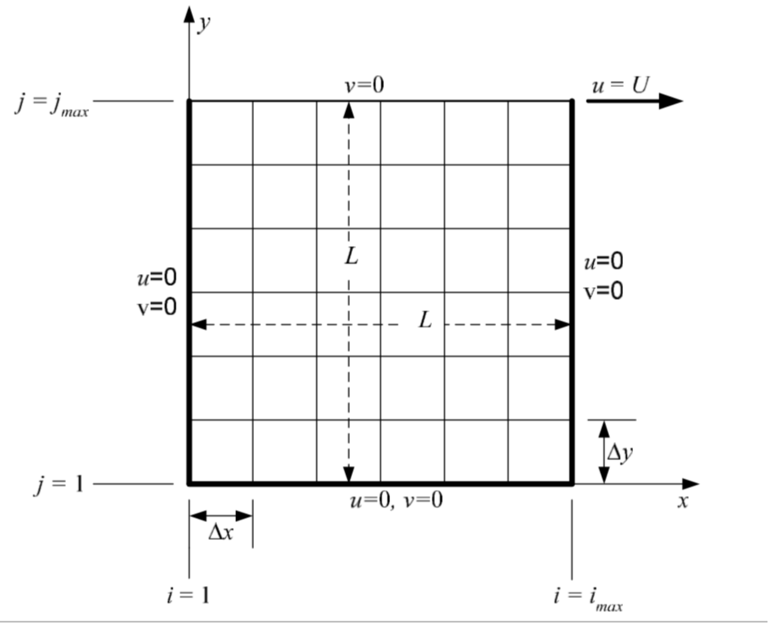

Figure 3.1: Lid Driven Square Cavity with an uniform grid for the whole domain, $Re = \frac{LU}{v}$.

3.2 Governing Equations

The governing equations for the present flow configuration are the two-dimensional (2D) incompressible Navier-Stokes (NS) equations. The 2D NS equations in the **non-dimensional** primitive-variables form are given by:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{3.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\nabla^2 u = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{3.2}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\nabla^2 v = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{3.3}$$

For a steady 2D flow, it is possible to have an equivalent non-dimensional system of two equations known as the streamfunction-vorticity (or $\psi - \omega$) form of the NS equations:

$$\nabla^2\psi = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{3.4}$$

$$u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \frac{1}{Re}\nabla^2\omega = \frac{1}{Re}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right) \tag{3.5}$$

Chapter 4

Local Defect Correction in the case of Lid Driven Cavity

4.1 Local Defect Correction

The following grid structure (with varying resolutions) was used to apply local defect correction to the lower left corner of the global domain (figure 4.1).

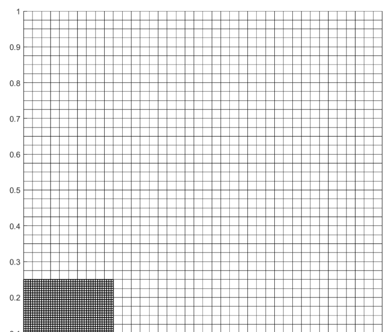

Figure 4.1: Grid structure for lid driven cavity

This was done to check the validity of the proposed schemes. We then used finer grids in other 'high activity' regions as well.

Chapter 5

Discretization schemes

In this chapter we describe the discretization scheme we have used to solve the Navier-Stokes (NS) equations. The following scheme is based on the streamfunction-vorticity formulation of the NS equations.

5.1 The streamfunction-vorticity formulation

The streamfunction-vorticity (or $\psi - \omega$) form of the NS equations is as follows

$$\nabla^2\psi = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = -\omega \tag{5.1}$$

$$u\frac{\partial\omega}{\partial x} + v\frac{\partial\omega}{\partial y} = \frac{1}{Re}\nabla^2\omega = \frac{1}{Re}\left(\frac{\partial^2\omega}{\partial x^2} + \frac{\partial^2\omega}{\partial y^2}\right) \tag{5.2}$$

5.2 ***DS1:*** Discretization involving both $\psi - \omega$

We now discretize our equations involving ψ and ω.

5.2.1 Interior points

Using 5 point formula, we get the following discretization for the interior points:

For ψ

We use second-order accurate central differencing for all the space derivatives.The discretized ψ-equation at point (i, j) is written as:

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta y)^2} = -\omega_{i,j} \tag{5.3}$$

We introduce the notation $\beta = \frac{\Delta x}{\Delta y}$. Upon rearrangement of equation (4.6) we shall get

$$\psi_{i,j} = \frac{\psi_{i+1,j} + \psi_{i-1,j} + \beta^2(\psi_{i,j+1} + \psi_{i,j-1}) + (\Delta x)^2\omega_{i.j}}{2(1+\beta^2)} \tag{5.4}$$

For ω

The discretized form of the ω equation at (i, j) point is

$$u_{i,j}\frac{\omega_{i+1,j} - \omega_{i-1,j}}{2\Delta x} + v_{i,j}\frac{\omega_{i,j+1} - \omega_{i,j-1}}{2\Delta y} = \frac{1}{Re}(\frac{\omega_{i+1,j} - 2\omega_{i,j} + \omega_{i-1,j}}{(\Delta x)^2} + \frac{\omega_{i,j+1} - 2\omega_{i,j} + \omega_{i,j-1}}{(\Delta y)^2}) \tag{5.5}$$

Upon rearrangement after multiplying on both sides by $(\Delta x)^2$ we get

$$\omega_{i,j} = \frac{\omega_{i+1,j} + \omega_{i-1,j} + \beta^2(\omega_{i,j+1} + \omega_{i,j-1}) - \frac{1}{2}\Delta x Re(\omega_{i+1,j} - \omega_{i-1,j})u_{i,j} - \frac{1}{2}\Delta x\beta Re(\omega_{i,j+1} - \omega_{i,j-1})v_{i,j}}{2(1+\beta^2)} \tag{5.6}$$

5.2.2 Boundary conditions

For ψ

Take $\psi = 0$ on all the walls (any real number can be used; but 0 is better for the sake of comparison as most papers use this value)

For ω

Left wall: $\omega_{1,j} = -\frac{2\psi_{2,j}}{(\Delta x)^2} \& 2 \leq j \leq j_{max-1}$
Right wall: $\omega_{imax,j} = -\frac{2\psi_{imax-1,j}}{(\Delta x)^2} \& 2 \leq j \leq j_{max-1}$
Bottom wall: $\omega_{i,1} = -\frac{2\psi_{i,2}}{(\Delta y)^2} \& 2 \leq i \leq i_{max-1}$
Top wall: $\omega_{i,jmax} = -\frac{2\psi_{i,jmax-1}+2\Delta y}{(\Delta y)^2} \& 2 \leq i \leq i_{max-1}$

These conditions were obtained by coupling with (5.4) and (5.6) the conditions of zero wall-tangential velocity at the side and bottom walls and a tangential velocity of U at the top wall.

5.3 *DS2:* Discretization involving only ψ

For eliminating the need to update ω, we use the following finite difference formulation [10]. For discretization we use a uniform two-dimensional grid with step size h in either direction. The approximation at a point (x, y) utilizes the value of ψ at 8 nearest neighbours of the point in the compact stencil, and the velocities u, v at its nearest neighbours.

$$\begin{gathered}
-28\psi(x,y) + 8[\psi(x+h,y) + \psi(x-h,y) + \psi(x,y+h) + \psi(x,y-h)] \\
- [\psi(x+h,y+h) + \psi(x-h,y+h) + \psi(x-h,y-h) + \psi(x+h,y-h)] \\
- 3h[u(x,y+h) - u(x,y-h) - v(x+h,y) + v(x-h,y)] \\
- \frac{Reh}{4}u(x,y)[2(\psi(x+h,y) - \psi(x-h,y)) - \psi(x+h,y+h) + \psi(x-h,y+h) \\
+ \psi(x-h,y-h) - \psi(x+h,y-h) + 2h(v(x+h,y) - 2v(x,y) + v(x-h,y))] \\
- \frac{Reh}{4}v(x,y)[2(\psi(x,y+h) - \psi(x,y-h)) - \psi(x+h,y+h) - \psi(x-h,y+h) + \\
\psi(x-h,y-h) + \psi(x+h,y-h) - 2h(u(x,y+h) - 2u(x,y) + u(x,y-h))] = 0
\end{gathered} \tag{5.7}$$

The velocities u and v can be approximated as

$$u(x,y) = \frac{3}{4h}[\psi(x,y+h) - \psi(x,y-h)] - \frac{1}{4}[u(x,y+h) + u(x,y-h)] \tag{5.8}$$

and

$$v(x,y) = \frac{3}{4h}[\psi(x+h,y) - \psi(x-h,y)] - \frac{1}{4}[v(x+h,y) + v(x-h,y)] \tag{5.9}$$

5.3.1 Biharmonic equation and obtaining the discretization

The Dirichlet problem for the biharmonic equation is

$$\frac{\partial^4 \phi}{\partial x^4} + 2\frac{\partial^4 \phi}{\partial x^2 \partial y^2} + \frac{\partial^4 \phi}{\partial y^4} = f(x, y), (x, y) \in \Omega \tag{5.10}$$

$$\phi = g_1(x, y), \frac{\partial \phi}{\partial n} = g_2(x, y), (x, y) \in \partial\phi \tag{5.11}$$

where ϕ is a closed two-dimensional convex domain with boundary $\partial\phi$.
There are a few second and fourth order compact finite difference approximations for the biharmonic equation on a 9 point stencil[11]. The compact approach involves discretizing the biharmonic equation using not just the grid values of ϕ but also the values of the gradients ϕ_x, and ϕ_y at selected points in the stencil. This approach has a lot of advantages such as there is no need for any further approximations or special adjustments near the boundaries. Also the values ϕ_x and ϕ_y are already available over the whole grid and need not be approximated.
We can extend the same approach to derive our scheme for NS equations. For this we transform the streamfunction-vorticity formulation into a fourth order PDE.

$$\frac{\partial^4 \psi}{\partial x^4} + 2\frac{\partial^4 \psi}{\partial x^2 \partial y^2} + \frac{\partial^4 \psi}{\partial y^4} - Reu\left[\frac{\partial^3 \psi}{\partial x^3} + \frac{\partial^3 \psi}{\partial x \partial y^2}\right] - Rev\left[\frac{\partial^3 \psi}{\partial y^3} + \frac{\partial^3 \psi}{\partial x^2 \partial y}\right] = 0 \tag{5.12}$$

We treat the velocities u and v in the above equation as locally fixed at the grid point (x, y) for deriving the discretization. We obtain the scheme as described in

5.13. The discretization can be rewritten as

$$-28\psi_{i,j}+8[\psi_{i+1,j}+\psi_{i_1,j}+\psi_{i,j+1}+\psi_{i,j-1}]-[\psi_{i+1,j+1}+\psi_{i-1,j+1}+\psi_{i-1,j-1}+\psi_{i+1,j-1}]$$
$$-3h[u_{i,j+1}-u_{i,j-1}-v_{i+1,j}+v_{i-1,j}]-\frac{Reh}{4}u_{i,j}[2(\psi_{i+1,j}-\psi_{i-1,j})-\psi_{i+1,j+1}+\psi_{i-1,j+1}$$
$$+\psi_{i-1,j-1}-\psi_{i+1,j-1}+2h(v_{i+1,j}-2v_{i,j}+v_{i-1,j})]-\frac{Reh}{4}v_{i,j}[2(\psi_{i,j+1}-\psi_{i,j-1})-$$
$$\psi_{i+1,j+1}-\psi_{i-1,j+1}+\psi_{i-1,j-1}+\psi_{i+1,j-1}-2h(u_{i,j+1}-2u_{i,j}+u_{i,j-1})]=0 \quad (5.13)$$

$$u_{i,j}=\frac{3}{4h}[\psi_{i,j+1}-\psi_{i,j-1}]-\frac{1}{4}[u_{i,j+1}+u_{i,j-1}] \quad (5.14)$$

and

$$v_{i,j}=\frac{3}{4h}[\psi_{i+1,j}-\psi_{i-1,j}]-\frac{1}{4}[v_{i+1,j}+v_{i-1,j}] \quad (5.15)$$

The approximation has truncation error of order $O(h^2)$. The system of equations rising from this finite difference scheme takes the form

$$A\Psi=f(Re,\mathbf{u},\mathbf{v})(5.16)$$

We will solve this system of equations using the BiCGStab method.

Chapter 6

The BiCGStab method

We used the **Biconjugate Gradient method (Stabilised)** or **BiCGStab** [12] as our iterative solver. The Biconjugate Gradient method is a generalisation of the Conjugate Gradient method for non-symmetric linear systems.

6.1 The Conjugate Gradient method

The Conjugate Gradient (CG) method is an algorithm for finding numerical solution of a system of linear equations Ax = B such that A is symmetric and positive definite.

6.1.1 Description of CG as a direct method

Consider a nxn matrix A which is symmetric ($A = A^T$) and positive definite (i.e $uAu^T > 0 \; \forall u \in \mathbb{R}^n$).

Definition 6.1.1. Conjugate Consider two vectors u and v s.t. u,v $\in \mathbb{R}^n$. The vectors u and v are **conjugate** w.r.t. the matrix A iff $u^T A v = 0$.

We can define the expression $u^T A v$ as an inner product and use the fact that A

is symmetric and positive definite to obtain the following result. Here we define the operator

$$\langle u, v \rangle_A = \langle Au, v \rangle = \langle u, A^T v \rangle = \langle u, Av \rangle = u^T A v \tag{6.1}$$

Hence, we can say that two vectors u and v are conjugate iff they are orthogonal w.r.t this inner product. Consider $A \in \mathbb{R}^{n \times n}$. We can find a set

$$P = \{p_1, p_2, ..., p_n\} \tag{6.2}$$

consisting of n mutually conjugate vectors w.r.t the matrix A s.t. P forms a basis for $\mathbb{R}^n$. We can represent the solution to the system of equations $Ax = b$ as

$$x = \sum_{i=1}^{n} \alpha_i p_i \tag{6.3}$$

We need to find the values of the respective $\alpha_i, i = 1, 2...n$ to get the vector x. Starting from the original system and multiplying both sides by p_k^T for some $k \in \{1, 2, ..n\}$ we get

$$p_k^T b = p_k^T A x \tag{6.4}$$

which gives

$$p_k^T b = p_k^T A \sum_{i=1}^{n} \alpha_i p_i \tag{6.5}$$

Since all p_i are conjugate of p_k except for $i = k$, hence only one term will remain in the sum.

$$p_k^T b = p_k^T \alpha_k A p_k \tag{6.6}$$

$$\alpha_k = \frac{p_k^T b}{p_k^T A p_k} \tag{6.7}$$

This gives us a way to calculate the solution for a system of equations, provided 6.2 is known. However, its not a trivial task to calculate the set P. In practice, the conjugate gradient is computed as an iterative method.

6.1.2 CG as an iterative method

We convert the problem $Ax = b$ into a minimization problem. Consider the function

$$f(x) = \frac{1}{2}x^T Ax - x^T b \qquad x \in \mathbb{R}^n \tag{6.8}$$

If A is symmetric and positive definite, the function is convex. The minimum of the function will be found where derivative of the function is equal to zero, i.e $Ax - b = 0$. Hence, both the problems have the same solution. To solve the minimization problem, we will simply perform gradient descent. The problem with simple gradient descent is that progress becomes very slow as the value of the numerical solution comes close to the optimum. To overcome this problem we make sure that the respective gradient directions are conjugate to each other w.r.t A. The CG method takes $O(n)$ time to terminate where A is a $n \times n$ matrix. We use a linear combination of all gradient directions in CG, unlike in gradient descent where we only use the current gradient direction.
We start with an initial guess x_0, an calculate the gradient at that point. Let p_1 vector be the negative of gradient at x_0. It will be our first conjugate vector.

$$p_1 = -(Ax_0 - b) \tag{6.9}$$

Now we take a step in direction of p_1

$$x_1 = x_0 + \alpha_1 p_1 \tag{6.10}$$

We need to make sure the successive p_i are conjugate to the previous directions. We ensure that by using a process similar to Gram-Schmidt orthonormalisation.

For the residual at every step

$$r_k = b - Ax_k \tag{6.11}$$

we have

$$p_{k+1} = r_k - \sum_{i \leq k} \beta_{ik} p_i \tag{6.12}$$

where

$$\beta_{ik} = \frac{r_k^T A p_i}{p_i^T A p_i} \tag{6.13}$$

gives the projector operator of r_k on p_i.

$$x_{k+1} = x_k + \alpha_{k+1} p_{k+1} \tag{6.14}$$

where the step size or α_k is defined by 6.7.

This orthogonalization process can be further simplified. For that we define the error as

$$e_k = x - x_k = \sum_{i=k+1}^{n} \alpha_i p_i \tag{6.15}$$

Its a trivial observation that

$$r_k = b - Ax_k = Ae_k \tag{6.16}$$

Theorem 6.1.2. *The residual r_k,where $k \in \{1, 2...n\}$ is orthogonal to all previous p_j i.e $j \leq k$.*

Proof. Multiplying 6.15 on both sides by $p_j^T A (j \neq k)$ we get

$$p_j^T A e_k = \sum_{i=k+1}^{n} \alpha_i p_j^T A p_i \tag{6.17}$$

using 6.16 we get

$$p_j^T r_k = \sum_{i=k+1}^{n} \alpha_i p_j^T A p_i \tag{6.18}$$

For every $j \leq k$ the rhs is zero since p_i form a conjugate set. Hence our result is proved. $\square$

Theorem 6.1.3. *The residual r_j,where $k \in \{1, 2...n\}$ is orthogonal to all previous r_k i.e $k < j$.*

Proof. Multiplying 6.12 on both sides by r_j, where $j \leq k$ we get

$$p_{k+1}^T r_j = r_k^T r_j + \sum_{i \leq k} \beta_{ik} p_i^T r_j \tag{6.19}$$

Using the theorem we just proved, we get

$$0 = r_k^T r_j + 0 \tag{6.20}$$

Hence our result is proved. $\square$

We can now define the step size α in terms of residuals. From the previous theorem

$$r_{k+1}^T r_k = 0 \tag{6.21}$$

substituting from 6.14 and 6.11 we get

$$(b - Ax_k)^T r_k - \alpha - k + 1(Ap_{k+1})^T r_k = 0 \tag{6.22}$$

This can be evaluated to give

$$\alpha_{k+1} = \frac{r_k^T r_k}{p_{k+1}^T A p_{k+1}} \tag{6.23}$$

From 6.14 we can get

$$b - Ax_{k+1} = b - Ax_k - \alpha_{k+1}Ap_{k+1} \tag{6.24}$$

$$r_{k+1} = r_k - \alpha_{k+1}Ap_{k+1} \tag{6.25}$$

We want to find the value of β_{jk}. For this we multiply by r_j^T

$$r_j^T r_{k+1} = r_j^T r_k - \alpha_{k+1} r_j^T Ap_{k+1} \tag{6.26}$$

from 6.13 we get

$$\alpha_{k+1}\beta_{k+1,j}p_{k+1}^T Ap_{k+1} = -r_j^T r_{k+1} + r_j^T r_k \tag{6.27}$$

Using 6.23 and shifting index k we get

$$\beta_{k,j} = \frac{-r_j^T r_k + r_j^T r_{k-1}}{r_{k-1}^T r_{k-1}} \tag{6.28}$$

Since the residuals are orthogonal, for $k < j$ $\beta_{kj} = 0$ and

$$\beta_{kk} = -\frac{r_k^T r_k}{r_{k-1}^T r_{k-1}} \tag{6.29}$$

Hence

$$p_{k+1} = r_k\beta_{kk}p_k \tag{6.30}$$

6.2 The Biconjugate gradient method

The CG method is unsuitable for nonsymmetric systems as the residual vectors cannot be made orthogonal with short recurrences. In the BiCG [12] method, we replace the orthogonal sequence of residuals by two mutually orthogonal sequences.

The residual r_j is orthogonal w.r.t $\hat{r}_0$, $\hat{r}_1$, ... $\hat{r}_{j-1}$. And vice versa $\hat{r_j}$ is orthogonal w.r.t $r_1, r_2, ...r_{j-1}$. We update the sequence of residuals as

$$r_i = r_{i-1} - \alpha_i A p_i \tag{6.31}$$

$$\hat{r}_i = \hat{r}_{i-1} - \alpha_i A^T \hat{p}_i \tag{6.32}$$

The two sequences of search directions are updated as

$$p_i = r_{i-1} + \beta_{i-1} p_{i-1} \tag{6.33}$$

$$\hat{p}_i = \hat{r}_{i-1} + \beta_{i-1} \hat{p}_{i-1} \tag{6.34}$$

where

$$\alpha_i = \frac{\hat{r}_{i-1}^T \hat{r}_{i-1}}{\hat{p}_i^T A p_i} \tag{6.35}$$

and

$$\beta_i = \frac{\hat{r}_i^T \hat{r}_i}{\hat{r}_{i-1}^T r_{i-1}} \tag{6.36}$$

ensure that bi-orthogonality

$$r_i^T r_j = \hat{p}_i^T A p_j = 0 \tag{6.37}$$

is maintained.

6.2.1 The stabilised variant

From the previous equations we can easily show that $r_i = P_i(A) r_0$ and $p_{i+1} = T_i(A) r_0$ where $P_i(A)$ and $T_i(A)$ are both i^{th} degree polynomials in A. According to the BiCG algorithm

$$T_i(A)r_0 = (P_i(A) + \beta_{i+1}T_{i-1}(A))r_0 \tag{6.38}$$

and

$$P_i(A)r_0 = (P_{i-1}(A) - \alpha_i A T_{i-1}(A))r_0 \tag{6.39}$$

In the BiCGStab algorithm we want a recurrence of the form

$$r_i = Q_i(A)P_i(A)r_0 \tag{6.40}$$

where Q_i is a polynomial of the form

$$Q_i(x) = (1 - \omega_1 x)(1 - \omega_1 x)...(1 - \omega_i x) \tag{6.41}$$

We can evaluate this recurrence as

$$Q_i(A)P_i(A)r_0 = (1 - \omega_i A)Q_{i-1}(A)(P_{i-1}(A) - \alpha_i A T_{i-1}(A)r_0) \tag{6.42}$$

from the BiCG algorithm we also get

$$Q_i(A)T_i(A)r_0 = Q_i(A)(P_i(A) + \beta_{i+1}T_{i-1}(A))r_0 \tag{6.43}$$

The BiCG constants like ρ, α_i, β_i (see the following algorithm) can be calculated by inner products in terms of these newly generated vectors.
The BiCGStab algorithm is given as

Algorithm 1 BiCGStab

```
1: **Input:** number of agents
2: $x_0 \leftarrow initial - guess$
3: $r_0 \leftarrow b - Ax_0$
4: $\hat{r}_0 \leftarrow arbitrary - vector - s.t$
5: $\langle \hat{r}_0, r_0 \rangle \neq 0$
6: $\rho_0 = \alpha = \omega_0 = 1$
7: $v_0 = p_0 = 0$
8: **for** $i = 1, 2, ...$ **do**
9:     $\rho_i = \langle \hat{r}_0, r_{i-1} \rangle$
10:    $\beta = (\rho_i / \rho_{i-1})(\alpha / \omega_{i-1})$
11:    $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$
12:    $v_i = Ap_i$
13:    $\alpha = \rho_i / \langle \hat{r}_0, v_i \rangle$
14:    $s = r_{i-1} - \alpha v_i$
15:    $t = As$
16:    $\omega_i = \langle t, s \rangle / \langle t, t \rangle$
17:    $x_i = x_{i-1} + \alpha p_i + \omega_i s$
18:    **if** $x_i$ is accurate enough **then**
19:        break
20:    **end if**
21:    $r_i = s - \omega_i t$
22: **end for**
```

Like CG, BiCGStab takes $O(n)$ time. Its advantage over BiCG is that it provides smoother convergence.

Chapter 7

Proposed Numerical Schemes

We experimented with and formalised various numerical schemes whereby we utilised cubic spline as our interpolation method and pure stream-function formulation. The primary mechanism behind all the schemes is a back and forth routine such that the values of the finer grid (eventually) be in accordance with the actual (coarser) values even due to the error introduced due to interpolation.
The schemes devised are referred to as the following:

- *Scheme 1 (NS1)*: Coarse Convergence - Fine Convergence {baseline scheme}
- *Scheme 2 (NS2)*: Back & Forth + Coarse Convergence - Fine Convergence
- *Scheme 3 (NS3)*: Back & Forth + Coarse Finite - Fine Finite
- *Scheme 4 (NS4)*: Back & Forth + Coarse Finite - Fine Finite + Coarse Convergence - Fine convergence
- *Scheme 5 (NS5)*: Back & Forth + Coarse Convergence - Finer Finite

The following routines are involved in association with the back and forth protocol:

1. `coarse_initialise()`: All the variables are initialised with the appropriate initial value(if known, like boundary conditions) or with some inital guess and the parameters(tolerance, max iter., relaxation parameter etc.) are set accordingly.

2. `coarse_solver()`: The values of coarse grid are evaluated till convergence is achieved (up to a set tolerance value) using a standard numerical scheme (refer here). The solver can take either the tolerance value or the number of the max out iterations.

3. `fine_initialise()`: This function works in the same way as the above dexcribed `coarse_initialise()` function but for the variables and parameters associated with the finer grid.

4. `fine_solver()`: Here, again we use the same numerical scheme as used in step 2 to obtain the values of the finer grid. This solver can take either the tolerance value or the number of the max out iterations.

5. `update_finer_from_coarse()`: This function call first updates the values of the points of the finer grid common to the coarser grid with the values from the coarser grid. The *remaining virtual boundary points* are then interpolated using cubic splines.

6. `update_coarser_from_finer()`: This function call works in the same manner as the previous update function, the only difference here being the update takes places from the finer grid to the common points in the coarser grid.

Now, we go through each of the devised schemes by simply explaining the routines involved in each of them.

7.1 Scheme 1: Coarse Convergence - Fine Convergence

This scheme is the most basic of all. Here, no back and forth mechanism is deployed. This formed the baseline for devising and comparing other schemes. The pseudo-code is as follows:

Algorithm 2 Coarse Convergence - Fine Convergence

```
1: function CCFC
2:     SOLVE_COARSER($tol\_coarse$)
3:     UPDATE_FINER_FROM_COARSER
4:     SOLVE_FINER($tol\_fine$)
5: end function
```

7.2 Scheme 2: Back and Forth with Coarse Convergence - Fine Convergence

This scheme is the back and forth version of the previously described *Scheme 1*. Here, a new function call `update_coarser_from_finer()` updates the values of the coarser grid which are common to the finer grid with the most recent values from finer grid. The additional thing in this scheme is the back and forth routine. The following is the pseudo-code for the same:

Algorithm 3 Back and Forth with Coarse Convergence - Fine Convergence

```
1: function BNF_CCFC($max\_iter$, $tol\_coarse$, $tol\_fine$)
2:     $i \leftarrow 0$
3:     while $i \leq max\_iter$ do
4:         SOLVE_COARSER($tol\_coarse$)
5:         UPDATE_FINER_FROM_COARSER
6:         SOLVE_FINER($tol\_fine$)
7:         UPDATE_COARSER_FROM_FINER
8:         $i \leftarrow i + 1$
9:     end while
10: end function
```

Next up, we introduce schemes where the grids are not solved till convergence up to a tolerance limit but are solved for a fixed number of outer iterations. This was done with the primary motive to bring in resonance between the solution convergence of the coarser and finer grid rather than letting them run independent of each other.

7.3 Scheme 3: Back and Forth with Coarse Convergence - Fine Finite.

In this scheme, we work with the usual back and forth setup as used in the *Scheme 2* with a modification such that finer grid is solved numerically only for a fixed number of outer iterations rather than till convergence up to a tolerance limit. The following is the pseudo-code for the same:

Algorithm 4 Back and Forth with Coarse Convergence - Fine Finite.

```
1: function BNF_CFFF($max\_outer\_iter$, $iter\_coarse$, $iter\_fine$)
2:     $i \leftarrow 0$
3:     while $i \leq max\_outer\_iter$ do
4:         SOLVE_COARSER($iter\_coarse$)
5:         UPDATE_FINER_FROM_COARSER
6:         SOLVE_FINER($iter\_fine$)
7:         UPDATE_COARSER_FROM_FINER
8:         $i \leftarrow i + 1$
9:     end while
10: end function
```

7.4 Scheme 4: Back and Forth with Coarse Finite - Fine Finite

Again, we slightly modify the *Scheme 3* such that now both the finer and coarser grid are solved for a fixed number of outer iterations. **The main motivation here was that the finer and the coarser grid come in *agreement* with each other before even converging.** The following is the pseudo-code for the same:

Algorithm 5 Back and Forth with Coarse Finite - Fine Finite

```
1: function BNF_CCFF($max\_outer\_iter$, $tol\_coarse$, $iter\_fine$)
2:     $i \leftarrow 0$
3:     while $i \leq max\_outer\_iter$ do
4:         SOLVE_COARSER($tol\_coarse$)
5:         UPDATE_FINER_FROM_COARSER
6:         SOLVE_FINER($iter\_fine$)
7:         UPDATE_COARSER_FROM_FINER
8:         $i \leftarrow i + 1$
9:     end while
10: end function
```

7.5 Scheme 5: Back and Forth with Coarse Finite - Fine Finite and subsequent convergence.

This scheme is an intuitive extension of the *Scheme 4*. Here, we run the coarser and finer grids till convergence after the finite steps of back and forth iterations.

Algorithm 6 Back and Forth with Coarse Finite - Fine Finite and subsequent convergence.

```
1: function BNF_CFFF_CONV($max\_outer\_iter$, $tol\_coarse$, $tol\_fine$, $iter\_coarse$, $iter\_fine$)
2:     $i \leftarrow 0$
3:     while $i \leq max\_outer\_iter$ do
4:         SOLVE_COARSER($iter\_coarse$)
5:         UPDATE_FINER_FROM_COARSER
6:         SOLVE_FINER($iter\_fine$)
7:         UPDATE_COARSER_FROM_FINER
8:         $i \leftarrow i + 1$
9:     end while
10:    SOLVE_COARSER($tol\_coarse$)
11:    SOLVE_FINER($tol\_fine$)
12: end function
```

Chapter 8

Results and Plots

We have already presented the results (and are attached later in this section for reference purposes) of local defect correction using *DS1* (involving both $\psi - \omega$). It was observed that even without a comprehensive back and forth mechanism we were able to obtain sufficiently good results(in agreement with the benchmarks). However, same was not the case when we dealt with *DS2* in which only a single variable ψ was involved, and thus required a improved numerical scheme. We then devised the 5 numerical schemes as mentioned in the previous section and here we present the results. We shall also discuss the significant improvement we have achieved in terms of computational efficiency by avoiding the overkill required to compute the finer grid on the whole domain.

8.1 For *DS2*

8.1.1 Coarser grid numerical solutions

The following is the contour plot obtained by using a single (coarser 81×81) grid over the global domain (without any local defect correction):

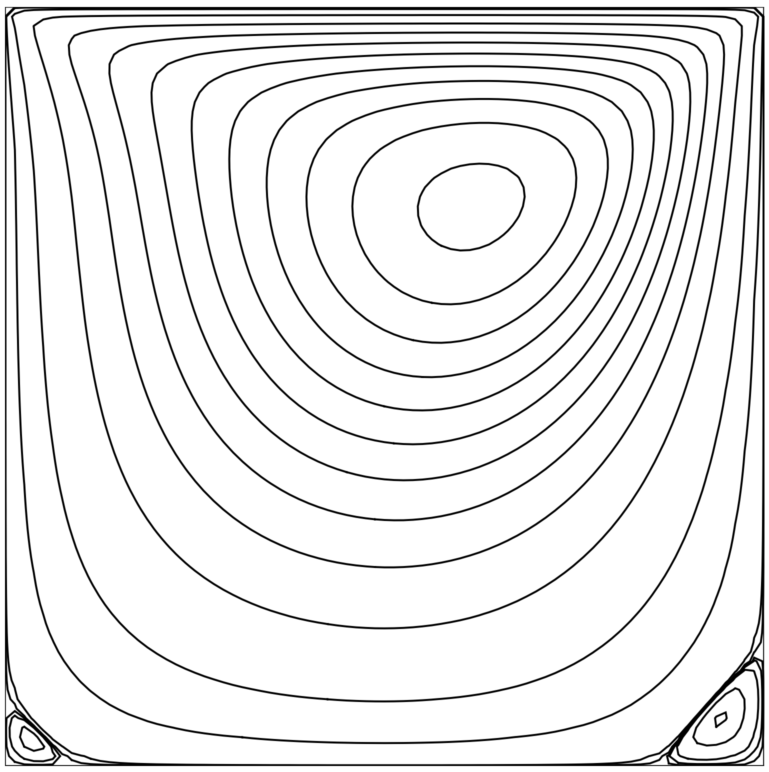

Figure 8.1: Contour plot of the whole domain.

8.1.2 Bottom left corner (BL_1) from coarser grid

This is to show that using just a coarser grid globally gives results which do not capture the activity in the whole domain in a complete sense. The following is the contour plot of the bottom left corner by just using a global coarser 81×81 grid:

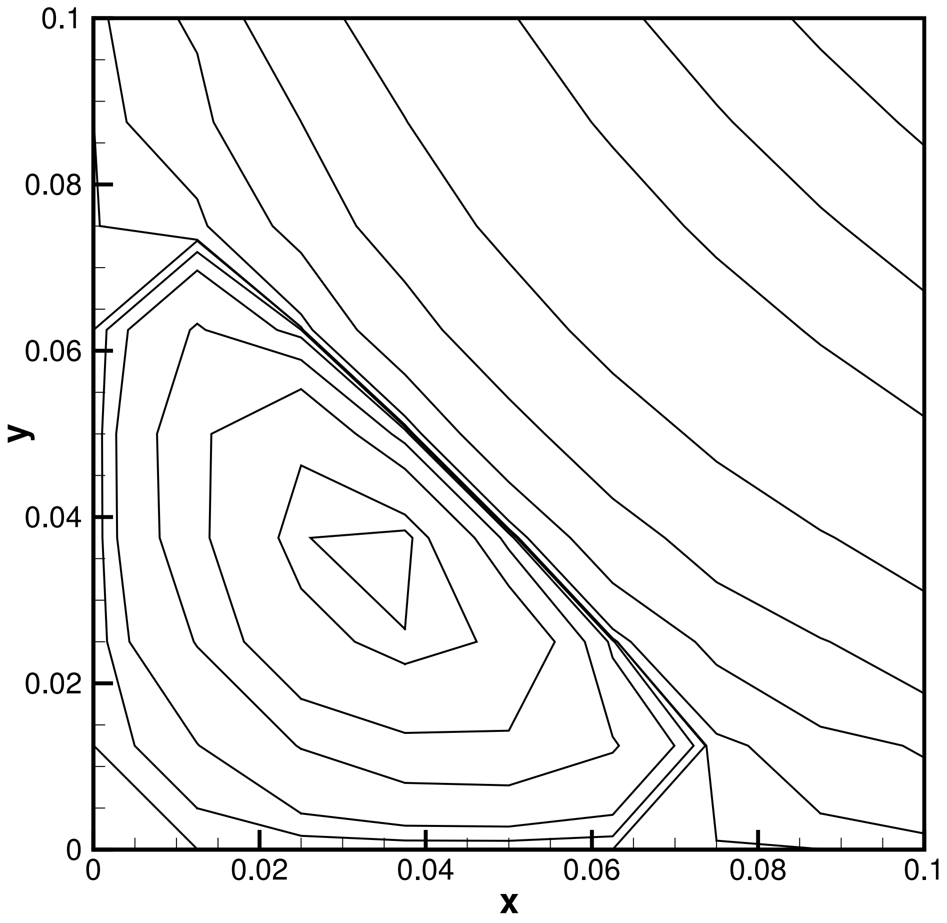

Figure 8.2: Contour plot of the bottom left corner (coarser grid).

8.1.3 Applying proposed schemes on BL_1

Now, we present the contour plots after applying our proposed numerical schemes and the significant improvement attained without doing expensive computations (in terms of time). The following is global combined contour plot showing that the local defect correction applied to the bottom left corner produced results(in red) which were in accordance with the coarser solutions.:

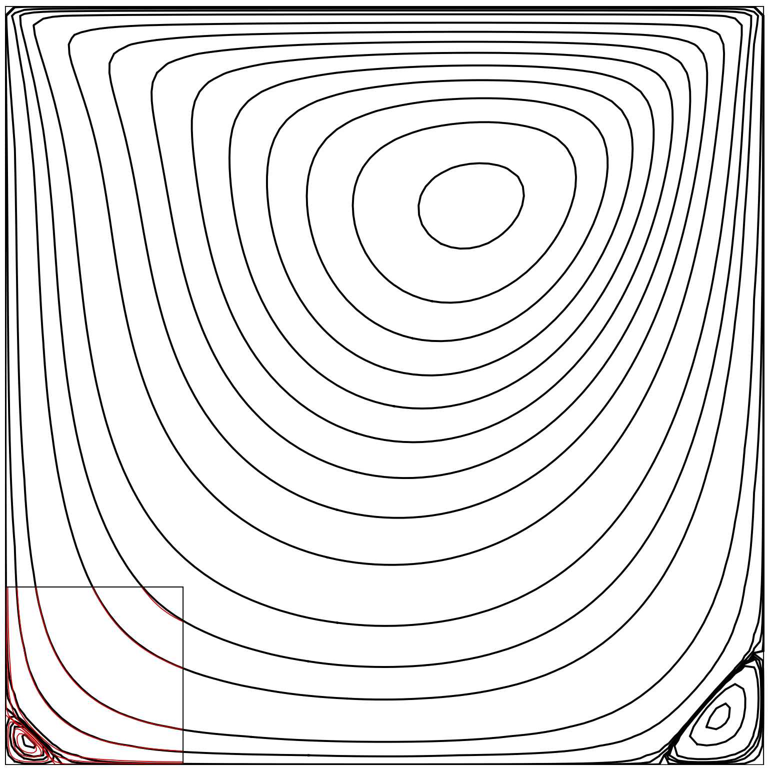

Figure 8.3: Combined contour plot of the whole domain.

Here, another 81×81 $finer$ grid was applied in the bottom left corner instead of a global finer 321×321 grid.

8.1.4 Bottom left corner (BL_1) from finer grid

The following is the contour plot of the bottom left secondary vortex (BL_1) as obtained from the finer grid. Here, the improvement in terms of smoothness (due to more number of points) is significantly visible.

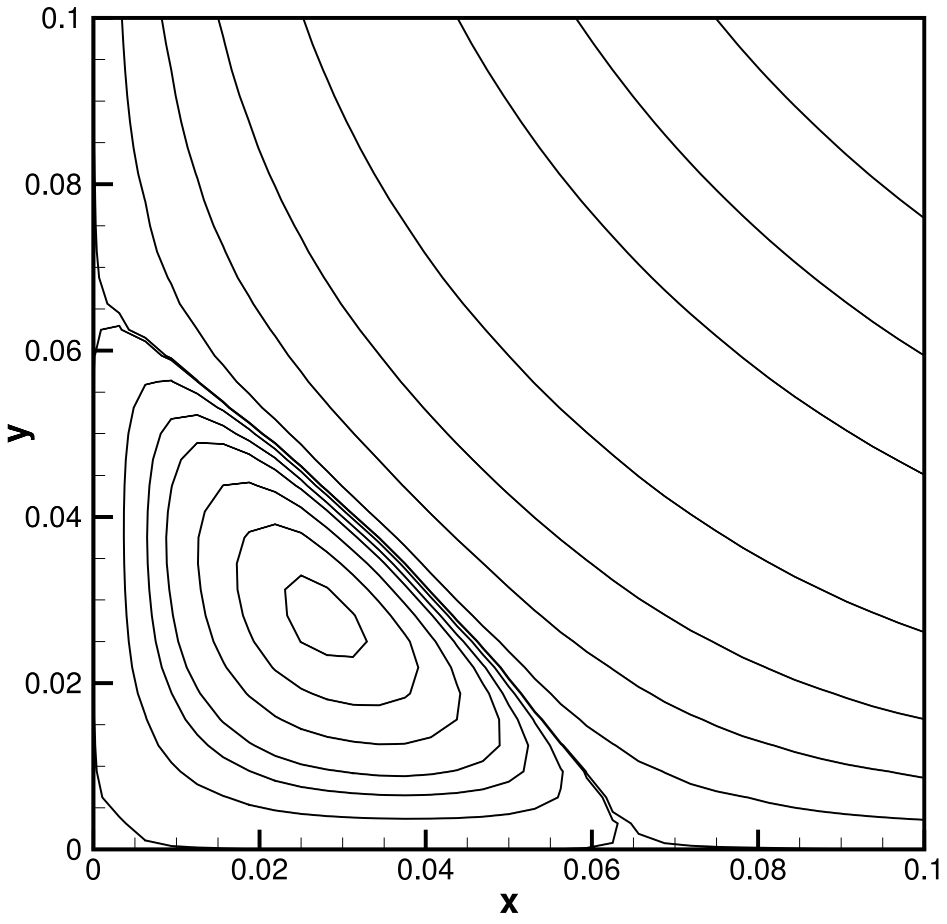

Figure 8.4: Contour plot of the bottom left corner (finer grid).

8.1.5 Comparisons

Contour plots and smoothness

We now compare the numerical results and the contour plot obtained subsequently. The crucial point to be noted here that the results in bottom left corner are obtained by just using another 81×81 local finer grid rather than a 321×321 global finer grid. This saves us from the enormous amount of computations required to get similar *smoother* results.

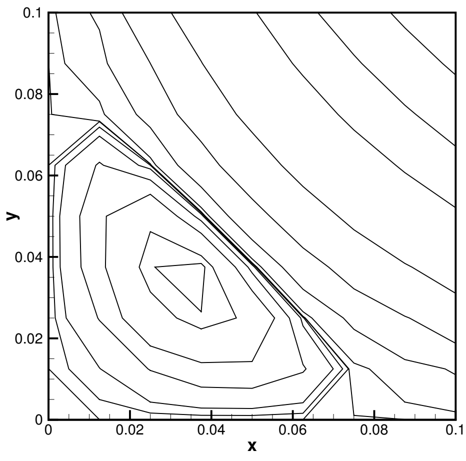

Figure 8.5: Contour plot of BL_1 from the coarser grid.

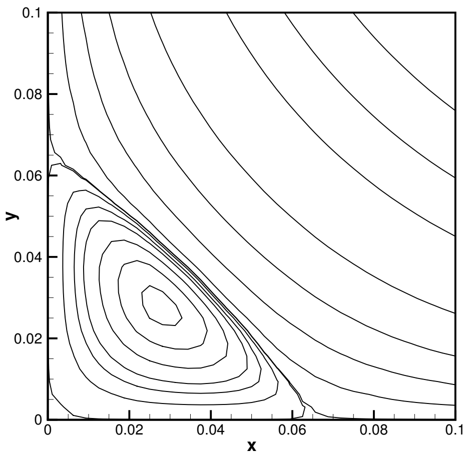

Figure 8.6: Contour plot of BL_1 from the finer grid.

Time comparisons

The regular program involving *DS2* for a uniform 321×321 grid takes around **14 hours** to reach convergence. However, our proposed numerical scheme achieves the result within around **23 minutes**. All the computations were carried out using a Intel C++ compiler on a 4GB RAM PC.

8.2 For *DS1*

The following plots were obtained from standard discretization of the stream-function and vorticity equations by using just the coarser grid on the global domain.

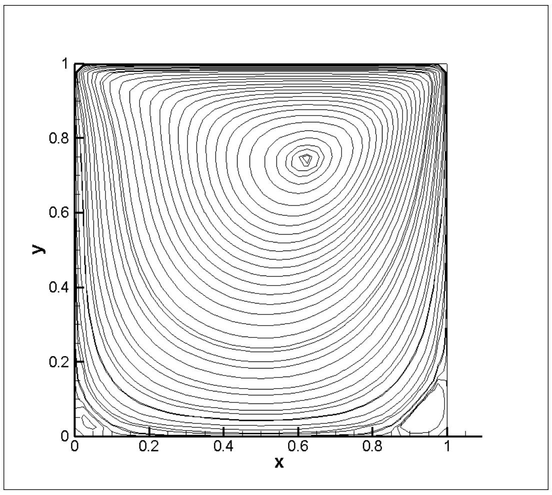

Figure 8.7: Contour plot of the global domain.

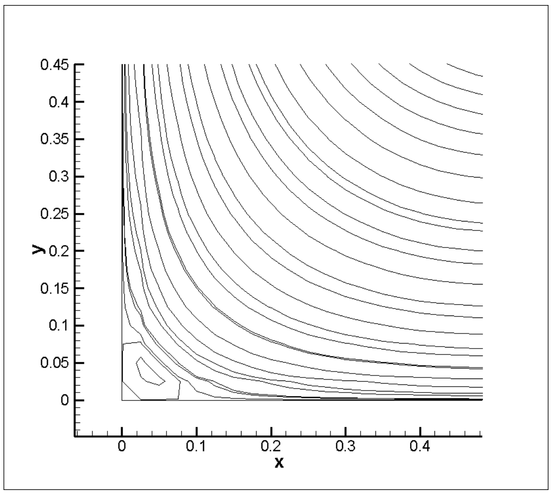

Figure 8.8: Contour plot of the lower left region

Observation: Using a uniform coarser grid global domain does not give us good results in high gradient regions(figure 8.7). And making the grid finer over the whole region increases the computation cost. Also since our main purpose is to just improve the results in the high gradient regions, application of an uniform finer grid is an overkill. For the coarser grid ψ has a Dirichlet BC and ω has Neumann BC. For the finer grid we experiment with different setting of the BCs.:

8.2.1 Using physical boundary conditions on all the virtual boundaries

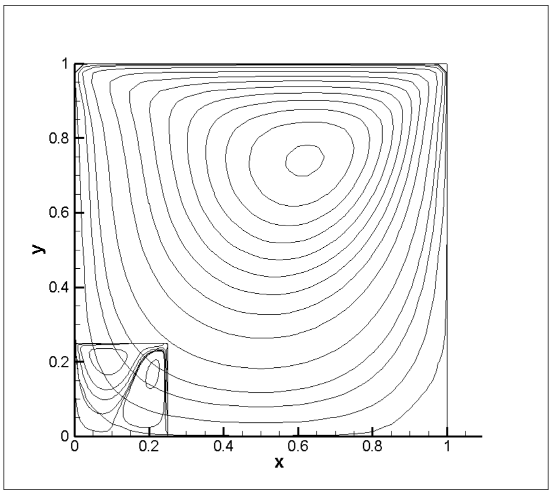

Figure 8.9: Contour plot of the whole domain.

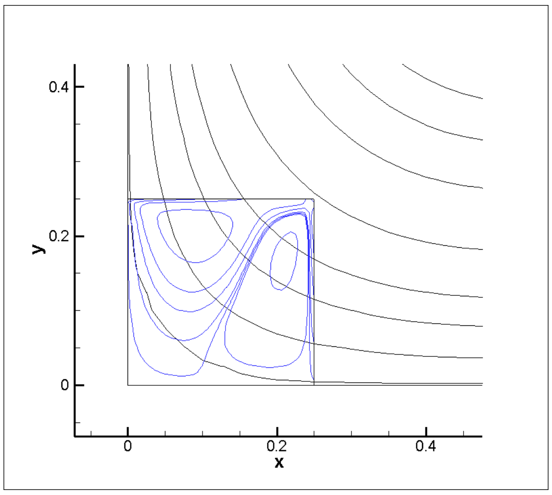

Figure 8.10: Contour plot of the lower left region.

As expected, we obtain results which would have been obtained if the finer domain were the whole global domain and the virtual boundaries were the actual physical boundaries.

Consequently, this gives us a solution where the finer grid now corresponds to a new global domain with actual physical boundaries.(figure 8.10) Since in the estimating ω on the actual physical boundaries we assumed the exterior point to be an image and the same generalization is not true for the interior point(s) which now act as points on the virtual boundary, this method is erroneous.

8.2.2 Keeping both ψ and ω fixed

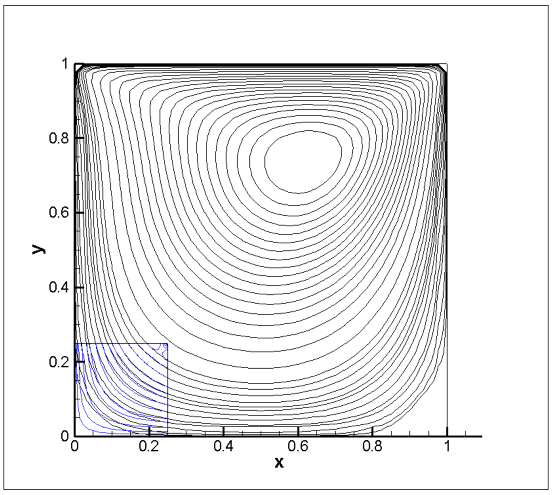

Figure 8.11: Contour plot of the whole domain.

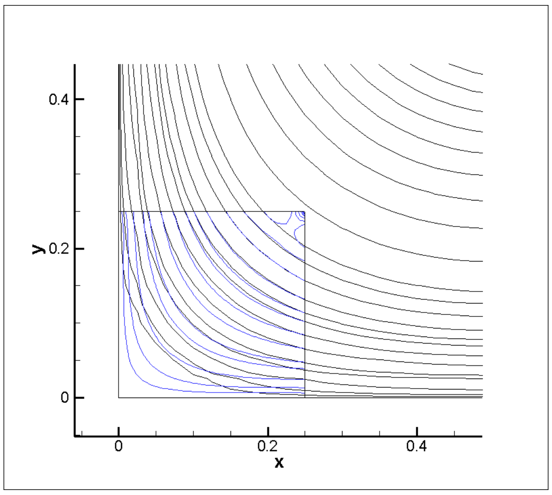

Figure 8.12: Contour plot of the lower left region.

Here, both the variables were held constant (initial interpolation by cubic splines) on the boundary. Since there is no refinement in the boundary conditions of the both ψ and ω, the error which arises due to the initial interpolation from cubic spline on the boundary prevails throughout the course of solution. Thus, the solution resembles the overall solution but the error is significantly high.

8.2.3 Keeping ψ fixed and varying ω

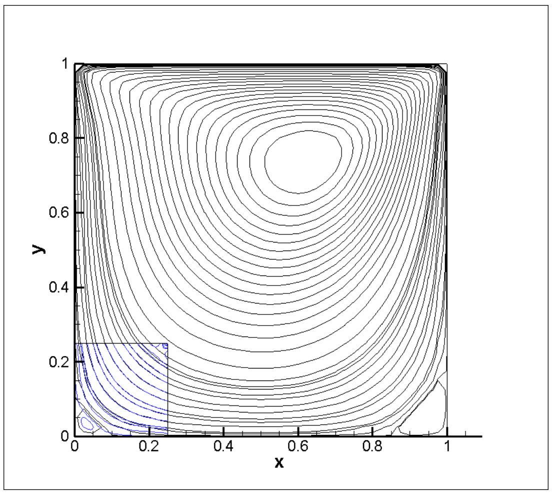

Figure 8.13: Contour plot of the whole domain.

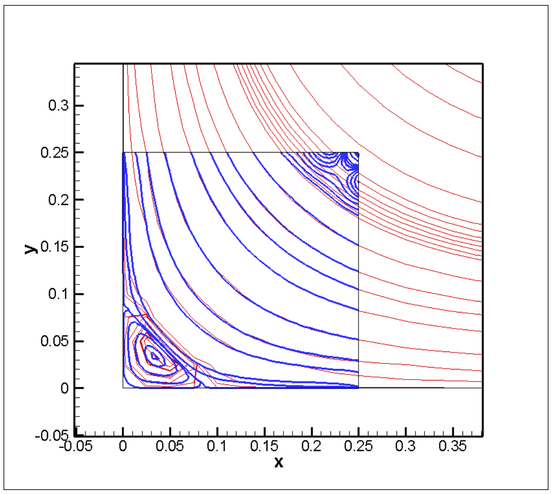

Figure 8.14: Contour plot of the lower left region.

Using the discretization for ω, we update ω every time by using the new values from the interior of the finer grid and old(fixed) values from the coarser grid to properly satisfy the neumann boundary conditions of ω. As can be observed from the contour plot this method proves superior from the previous two, as was expected(figure 8.14). But still there are some 'unwanted vortices' in the upper right corner. The reason of which could be ascribed to error which exists due to cubic spline interpolation since the top right corner's result are affected values from both the top and the right boundaries (both are virtual in our case). And the rest three corners have at least one true boundary.

Moreover, the error prevails due to the interpolation in the values of ψ which are not refined over the course of the solution.

8.2.4 Comparison with benchmark

We compared our results for the bottom left (BL) primary vortex with the benchmark solution. The following table summarises our findings.

Parameter	**Ghia's benchmark**	**Finer grid**	**Coarser grid**
ψ_{max}	1.74877 $\times 10^{-6}$	1.733538 $\times 10^{-6}$	2.488781 $\times 10^{-6}$
$\omega_{v.c.}$	-1.55509 $\times 10^{-2}$	-1.547339 $\times 10^{-2}$	-1.68443 $\times 10^{-2}$
Location (x, y)	0.0313,0.0391	0.03125,0.03125	0.03578,0.03729
H_L	0.0781	0.07892	0.7652
V_L	0.0781	0.07976	0.8078

The results we have obtained are very close to the benchmarks and hence look very promising. We seek to make further improvements to our methods not only in terms of accuracy, but also computation time in due course.

Bibliography

[1] K. W.Hackbusch, "Local defect correction method and domain decomposition techniques," in *Defect correction methods theory and applications*, K. B. . H. J. Stetter, Ed. New York: Springer-Verlag Wien, ch. 6, pp. 90–95.

[2] M. J. H. Anthonissen, *Local Defect Correction Techniques: Analysis and Application to Combustion*, ser. First Edition. Eindhoven University of Technology,, 2001.

[3] M. Anthonissen, R. Mattheij, and J. ten Thije Boonkkamp, "Convergence analysis of the local defect correction method for diffusion equations," *Numerische Mathematik*, vol. 95, no. 3, pp. 401–425, 2003.

[4] M. J. H. Anthonissen, *Local defect correction techniques: analysis and application to combustion.* Eindhoven University of Technology, 2001.

[5] P. J. Ferket and A. A. Reusken, "Further analysis of the local defect correction method," *Computing*, vol. 56, no. 2, pp. 117–139, 1996.

[6] S. McCormick, "Fast adaptive composite grid (fac) methods: Theory for the variational case," in *Defect correction methods.* Springer, 1984, pp. 115–121.

[7] R. Minero, M. Anthonissen, and R. Mattheij, "A local defect correction technique for time-dependent problems," *Numerical Methods for Partial Differential Equations*, vol. 22, no. 1, pp. 128–144, 2006.

[8] R. L. Burden and J. D. Faires., *Numerical analysis*, ser. Ninth Edition. Boston: Brooks/Cohle, Cengage Learning, 2011.

[9] R. A. Horn and C. R. Johnson, *Matrix analysis*, ser. Second Edition. Cambridge University Press, 2013.

[10] M. M. Gupta and J. C. Kalita, "A new paradigm for solving navier-stokes equations: stream-function-velocity formulation," *Journal of Computational Physics*, vol. 207, 2005.

[11] I. Altas, J. Dym, M. M. Gupta, and R. Manohar, "Multigrid solution of automatically generated high-order discretizations for the biharmonic equation," *SIAM Journal on scientific computing*, vol. 19(5), 1998.

[12] V. D. Vorst, "Bicgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems," *SIAM journal on scientific computing*, vol. 13(2), 1992.