Local Defect Correction using Cubic Splines for Incompressible Viscous Flows

MA499 Project II

Aditya Gupta[1] Vishal Kumar[2]

[1]130123051
Department of Mathematics

[2]130123043
Department of Mathematics

8 May, 2017

Outline

Outline

Problem Description

To devise a **Local Defect Correction** based approach using **Cubic Splines** for **Incompressible Viscous Flows.**

Outline

Motivation

- Infeasibility of analytical solution for many problems.

Motivation

- Infeasibility of analytical solution for many problems.
- Numerical methods are used to solve such problems.

Motivation

- Infeasibility of analytical solution for many problems.
- Numerical methods are used to solve such problems.
- Some Boundary Value Problems (BVP) have rapid variations in their solution in some parts of the domain.

Motivation

- Infeasibility of analytical solution for many problems.
- Numerical methods are used to solve such problems.
- Some Boundary Value Problems (BVP) have rapid variations in their solution in some parts of the domain.
- A single discretization scheme will require a very fine grid overall, leading to unnecessary calculations.

Motivation

- Infeasibility of analytical solution for many problems.
- Numerical methods are used to solve such problems.
- Some Boundary Value Problems (BVP) have rapid variations in their solution in some parts of the domain.
- A single discretization scheme will require a very fine grid overall, leading to unnecessary calculations.
- **LDC** uses different discretization schemes for coarser and finer grid, saving a lot of computations.

Outline

Description

LDC uses two different discretization schemes.

A general problem

A boundary value problem with domain Ω, like

$$Lu = f \tag{1}$$

with boundary condition

$$Bu = g \tag{2}$$

on

$$\Gamma = \delta\Omega \tag{3}$$

Discretization scheme

For our original BVP, the discretization scheme can be written as

Discretization scheme

For our original BVP, the discretization scheme can be written as

$$L_h u_{h,\Omega} = f_{h,\Omega} \tag{4}$$

Discretization scheme

For our original BVP, the discretization scheme can be written as

$$L_h u_{h,\Omega} = f_{h,\Omega} \tag{4}$$

Including boundary conditions on Γ .Here h is the step size in the underlying grid Ω_h. For a subdomain $\omega \subset \Omega$ the local BVP is

Discretization scheme

For our original BVP, the discretization scheme can be written as

$$L_h u_{h,\Omega} = f_{h,\Omega} \tag{4}$$

Including boundary conditions on Γ .Here h is the step size in the underlying grid Ω_h. For a subdomain $\omega \subset \Omega$ the local BVP is

$$Lu_\omega = f \tag{5}$$

Discretization scheme (...continued)

The discretization in this subdomain ω is given by

Discretization scheme (...continued)

The discretization in this subdomain ω is given by

$$L^{'}_{h'} u^{'}_{h',\omega} = f^{'}_{h',\omega} \tag{6}$$

Discretization scheme (...continued)

The discretization in this subdomain ω is given by

$$L'_{h'} u'_{h',\omega} = f'_{h',\omega} \tag{6}$$

L' and u' indicate the discretization may be different from L_h and u_h. h' subscript means a different step size from h.

Discretization scheme (...continued)

On the interior boundary Γ_1 we have

Discretization scheme (...continued)

On the interior boundary Γ_1 we have

$$u_\omega = u_\Omega \tag{7}$$

Discretization scheme (...continued)

On the interior boundary Γ_1 we have

$$u_\omega = u_\Omega \tag{7}$$

The discretization of equation (7) is given by:

$$u^{'}_{h^{'},\omega} = \gamma u_{h,\Omega} \tag{8}$$

Discretization scheme (...continued)

On the interior boundary Γ_1 we have

$$u_\omega = u_\Omega \tag{7}$$

The discretization of equation (7) is given by:

$$u'_{h',\omega} = \gamma u_{h,\Omega} \tag{8}$$

Where γ is an interpolation on $u_{h,\Omega}$ in $\Gamma_{1,h'}$. In our experiments its the cubic spline interpolation

Outline

Why Interpolation?

- LDC based discretization scheme use two grids.
  - **Coarser Grid** Actual physical boundaries.
  - **Finer Grid** At least 1 virtual boundary.

Why Interpolation?

- LDC based discretization scheme use two grids.
  - **Coarser Grid** Actual physical boundaries.
  - **Finer Grid** At least 1 virtual boundary.
- Cubic Splines can be used to interpolate missing boundary values.

Why Interpolation?

- LDC based discretization scheme use two grids.
  - **Coarser Grid** Actual physical boundaries.
  - **Finer Grid** At least 1 virtual boundary.
- Cubic Splines can be used to interpolate missing boundary values.

IMPROVED BOUNDARY INFORMATION! :)

Why Interpolation?

- LDC based discretization scheme use two grids.
  - **Coarser Grid** Actual physical boundaries.
  - **Finer Grid** At least 1 virtual boundary.
- Cubic Splines can be used to interpolate missing boundary values.

IMPROVED BOUNDARY INFORMATION! :)

Approach

- Acquire the boundary value of common points from coarser grid.

Why Interpolation?

- LDC based discretization scheme use two grids.
  - **Coarser Grid** Actual physical boundaries.
  - **Finer Grid** At least 1 virtual boundary.
- Cubic Splines can be used to interpolate missing boundary values.

IMPROVED BOUNDARY INFORMATION! :)

Approach

- Acquire the boundary value of common points from coarser grid.
- Interpolate missing values.

Why interpolation?

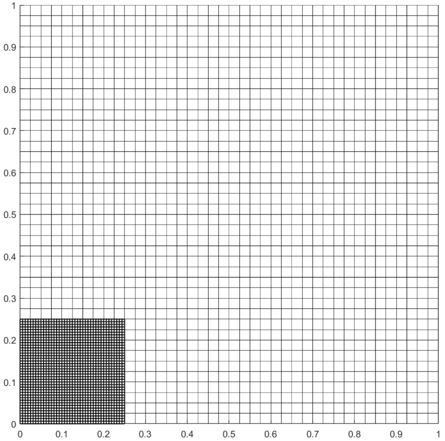

Figure 1: Grid structure for lid driven cavity

Outline

Description

- Piecewise polynomial interpolation

Description

- Piecewise polynomial interpolation
- High Degree polynomials can oscillate erratically because of a minor fluctuation over a small portion.

Description

- Piecewise polynomial interpolation
- High Degree polynomials can oscillate erratically because of a minor fluctuation over a small portion.
- Divide approximation interval into sub-intervals.

Description

- Piecewise polynomial interpolation
- High Degree polynomials can oscillate erratically because of a minor fluctuation over a small portion.
- Divide approximation interval into sub-intervals.
- Given n point(s) we approximate $n - 1$ piecewise cubic polynomial(s).

Outline

Why Cubic Polynomial ?

Smoothness!

Theorem

If S is the natural cubic spline function that interpolates a twice-continuously differentiable function f at knots $a = t_0 < t_1 < ... < t_n = b$ then

$$\int_a^b [S''(x)]^2 dx \leq \int_a^b [f''(x)]^2 dx$$

Constraints

Definition

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:
    1. $S''(x_0) = S''(x_n)$ (free boundary)
    2. $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)S'(x_n)$ (clamped boundary)

Outline

Natural Splines

The general form of cubic spline interpolant

$$a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3 \tag{9}$$

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

Theorem

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 by using **Levy Desplanques** theorem for dominant matrices.

Outline

The conjugate gradient

Algorithm for finding numerical solution of a system of linear equations $Ax = B$ such that A is symmetric and positive definite.

The conjugate gradient

Algorithm for finding numerical solution of a system of linear equations Ax = B such that A is symmetric and positive definite.
We convert the problem $Ax = b$ into a minimization problem.

$$f(x) = \frac{1}{2}x^T Ax - x^T b \qquad x \in \mathbb{R}^n \tag{10}$$

The conjugate gradient (contd...)

To solve the minimization problem, we perform gradient descent.

The conjugate gradient (contd...)

To solve the minimization problem, we perform gradient descent.
With simple gradient descent progress becomes very slow as the value of the numerical solution comes close to the optimum.

The conjugate gradient (contd...)

To overcome this problem we make sure that the respective gradient directions are conjugate to each other w.r.t A.

Definition

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 $u^T Av = 0$.

The conjugate gradient (contd...)

We start with an initial guess x_0, and calculate the gradient at that point. Our first conjugate vector is.

The conjugate gradient (contd...)

We start with an initial guess x_0, and calculate the gradient at that point. Our first conjugate vector is.

$$p_1 = -(Ax_0 - b) \tag{11}$$

The conjugate gradient (contd...)

We start with an initial guess x_0, and calculate the gradient at that point. Our first conjugate vector is.

$$p_1 = -(Ax_0 - b) \tag{11}$$

For the residual at every step

$$r_k = b - Ax_k \tag{12}$$

we have

The conjugate gradient (contd...)

$$p_{k+1} = r_k - \sum_{i \leq k} \beta_{ik} p_i \tag{13}$$

where

$$\beta_{ik} = \frac{r_k^T A p_i}{p_i^T A p_i} \tag{14}$$

gives the projector operator of r_k on p_i.

$$x_{k+1} = x_k + \alpha_{k+1} p_{k+1} \tag{15}$$

where α_{k+1} is step size.

Outline

The bi-conjugate gradient

The CG method is unsuitable for nonsymmetric systems.
In the BiCG 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 $r_1, r_2, ...r_{j-1}$.

The bi-conjugate gradient (contd...)

The residuals are updated as

$$r_i = r_{i-1} - \alpha_i A p_i \tag{16}$$

$$\hat{r}_i = \hat{r}_{i-1} - \alpha_i A^T \hat{p}_i \tag{17}$$

The two sequences of search directions are updated as

$$p_i = r_{i-1} + \beta_{i-1} p_{i-1} \tag{18}$$

$$\hat{p}_i = \hat{r}_{i-1} + \beta_{i-1} \hat{p}_{i-1} \tag{19}$$

This method has $O(n)$ time complexity.

The bi-conjugate gradient (contd...)

where

$$\alpha_i = \frac{\hat{r}_{i-1}^T \hat{r}_{i-1}}{\hat{p}_i^T A p_i} \tag{20}$$

and

$$\beta_i = \frac{\hat{r}_i^T \hat{r}_i}{\hat{r}_{i-1}^T r_{i-1}} \tag{21}$$

ensure that bi-orthogonality

$$\hat{r}_i^T r_j = \hat{p}_i^T A p_j = 0 \tag{22}$$

is maintained.

Outline

The BiCGStab Algorithm

The BiCGStab has smoother convergence behavior than BiCG. According to the BiCG algorithm $r_i = P_i(A)r_0$ 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.

$$T_i(A)r_0 = (P_i(A) + \beta_{i+1}T_{i-1}(A))r_0 \tag{23}$$

and

$$P_i(A)r_0 = (P_{i-1}(A) - \alpha_i AT_{i-1}(A))r_0 \tag{24}$$

The BiCGStab Algorithm

In the BiCGStab algorithm we want a recurrence of the form

$$r_i = Q_i(A)P_i(A)r_0 \tag{25}$$

where Q_i is a polynomial of the form

$$Q_i(x) = (1 - \omega_1 x)(1 - \omega_1 x)...(1 - \omega_i x) \tag{26}$$

We can evaluate this recurrence as

$$Q_i(A)P_i(A)r_0 = (1 - \omega_i A)Q_{i-1}(A)(P_{i-1}(A) - \alpha_i A T_{i-1}(A)r_0) \tag{27}$$

from the BiCG algorithm we also get

$$Q_i(A)T_i(A)r_0 = Q_i(A)(P_i(A) + \beta_{i+1}T_{i-1}(A))r_0 \tag{28}$$

The BiCGStab Algorithm I

$x_0 \leftarrow initial - guess$
$r_0 \leftarrow b - Ax_0$
$\hat{r}_0 \leftarrow arbitrary - vector - s.t$
$\langle \hat{r}_0, r_0 \rangle \neq 0$
$\rho_0 = \alpha = \omega_0 = 1$
$v_0 = p_0 = 0$
for $i = 1, 2, \dots$ **do**
    $\rho_i = \langle \hat{r}_0, r_{i-1} \rangle$
    $\beta = (\rho_i / \rho_{i-1})(\alpha / \omega_{i-1})$
    $p_i = r_{i-1} + \beta(p_{i-1} - \omega_{i-1} v_{i-1})$
    $v_i = Ap_i$
    $\alpha = \rho_i / \langle \hat{r}_0, v_i \rangle$
    $s = r_{i-1} - \alpha v_i$
    $t = As$
    $\omega_i = \langle t, s \rangle / \langle t, t \rangle$
    $x_i = x_{i-1} + \alpha p_i + \omega_i s$

The BiCGStab Algorithm II

if x_i is accurate enough **then**
    break
end if
$r_i = s - \omega_i t$
end for

A General Numerical Scheme for LDC using Cubic Splines

Steps

1. The coarser grid is solved first(till convergence or for some fixed number of iterations) by initializing it with proper boundary conditions (and zero values for the interior points).

A General Numerical Scheme for LDC using Cubic Splines

Steps

1. The coarser grid is solved first(till convergence or for some fixed number of iterations) by initializing it with proper boundary conditions (and zero values for the interior points).
2. The common points between the finer and coarser grid are used to then update the finer grid for boundary conditions as well as interior points(a better initial approximation).

A General Numerical Scheme for LDC using Cubic Splines

Steps

1. The coarser grid is solved first(till convergence or for some fixed number of iterations) by initializing it with proper boundary conditions (and zero values for the interior points).
2. The common points between the finer and coarser grid are used to then update the finer grid for boundary conditions as well as interior points(a better initial approximation).
3. The remaining boundary points are then interpolated using cubic spline for the virtual boundaries.

A General Numerical Scheme for LDC using Cubic Splines

Steps

1. The coarser grid is solved first(till convergence or for some fixed number of iterations) by initializing it with proper boundary conditions (and zero values for the interior points).
2. The common points between the finer and coarser grid are used to then update the finer grid for boundary conditions as well as interior points(a better initial approximation).
3. The remaining boundary points are then interpolated using cubic spline for the virtual boundaries.
4. The finer grid is then solved using a similar scheme as used in the coarser grid in step 1.

A General Numerical Scheme for LDC using Cubic Splines

Steps

1. The coarser grid is solved first(till convergence or for some fixed number of iterations) by initializing it with proper boundary conditions (and zero values for the interior points).
2. The common points between the finer and coarser grid are used to then update the finer grid for boundary conditions as well as interior points(a better initial approximation).
3. The remaining boundary points are then interpolated using cubic spline for the virtual boundaries.
4. The finer grid is then solved using a similar scheme as used in the coarser grid in step 1.
5. The common points are now used to update the coarser grid and we move back to step 1. The process is repeated until we reach below tolerance on comparing solution on common points for both the grids.

Outline

2D Navier-Stokes equations in a square cavity

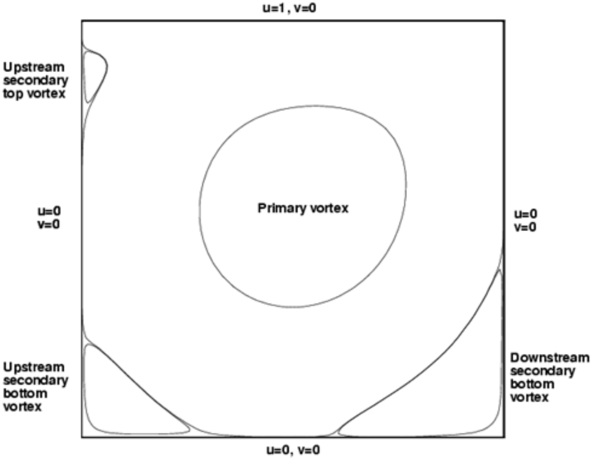

Figure 2: Lid Driven Square Cavity.

Variation in flow complexity with change in *Re*

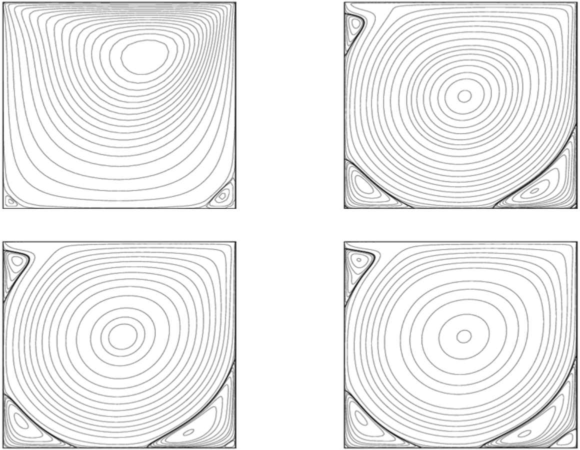

Figure 3: Flow for different values of *Re*.

Outline

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:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{29}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\nabla^2 u = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{30}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\nabla^2 v = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{31}$$

Non-Dimensionality and Variable Reduction

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:

$$\nabla^2 \psi = \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega \tag{32}$$

$$u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} = \frac{1}{Re} \nabla^2 \omega = \frac{1}{Re} \left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right) \tag{33}$$

Outline

DS1: Involving both $\psi - \omega$

- Numerically solving the 2D Navier-Stokes equations in a square cavity:.

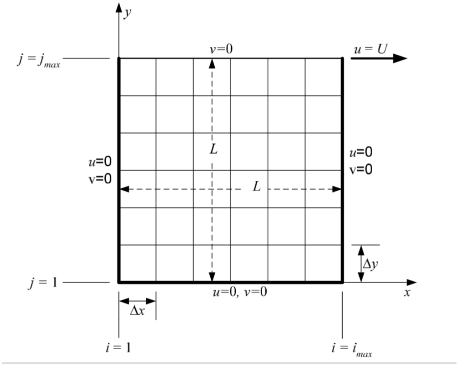

Figure 4: Lid Driven Square Cavity with an uniform grid for the whole domain, $Re = \frac{LU}{\nu}$.

Discretizing ψ

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

$$\frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{(\Delta x)^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{(\Delta y)^2} = -\omega_{i,j} \tag{34}$$

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

$$\psi_{i,j} = \frac{\psi_{i+1,j} + \psi_{i-1,j} + \beta^2(\psi_{i,j+1} + \psi_{i,j-1}) + (\Delta x)^2\omega_{i.j}}{2(1 + \beta^2)} \tag{35}$$

Discretizing ω

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

$$u_{i,j}\frac{\omega_{i+1,j}-\omega_{i-1,j}}{2\Delta x}+v_{i,j}\frac{\omega_{i,j+1}-\omega_{i,j-1}}{2\Delta y}=\frac{1}{Re}\left(\frac{\omega_{i+1,j}-2\omega_{i,j}+\omega_{i-1,j}}{(\Delta x)^2}+\frac{\omega_{i,j+1}-2\omega_{i,j}+\omega_{i,j-1}}{(\Delta y)^2}\right) \tag{36}$$

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

$$\omega_{i,j}=\frac{\omega_{i+1,j}+\omega_{i-1,j}+\beta^2(\omega_{i,j+1}+\omega_{i,j-1})-\frac{1}{2}\Delta x Re(\omega_{i+1,j}-\omega_{i-1,j})u_{i,j}-\frac{1}{2}\Delta x\beta Re(\omega_{i,j+1}-\omega_{i,j-1})v_{i,j}}{2(1+\beta^2)} \tag{37}$$

Boundary Conditions

Take $\psi = 0$ on all the walls.

$$\text{Left wall: } \omega_{1,j} = -\frac{2\psi_{2,j}}{(\Delta x)^2} \qquad 2 \leq j \leq j_{max-1} \tag{38}$$

$$\text{Right wall: } \omega_{imax,j} = -\frac{2\psi_{imax-1,j}}{(\Delta x)^2} \qquad 2 \leq j \leq j_{max-1} \tag{39}$$

$$\text{Bottom wall} \omega_{i,1} = -\frac{2\psi_{i,2}}{(\Delta y)^2} \qquad 2 \leq i \leq i_{max-1} \tag{40}$$

$$\text{Top wall} \omega_{i,jmax} = -\frac{2\psi_{i,jmax-1} + 2\Delta y}{(\Delta y)^2} \qquad 2 \leq i \leq i_{max-1} \tag{41}$$

These conditions were obtained by coupling with (35) and (37) the conditions of zero wall-tangential velocity at the side and bottom walls and a tangential velocity of U at the top wall.

Outline

DS2: Involving only ψ

- Its convenient to interpolate a single variable on the virtual boundaries of the finer grid instead of 2 to reduce the error which may be induced due to interpolation.

DS2: Involving only ψ

- Its convenient to interpolate a single variable on the virtual boundaries of the finer grid instead of 2 to reduce the error which may be induced due to interpolation.
- The paper by Prof. M. M. Gupta, and Prof. Jiten C. Kalita. “A new paradigm for solving NavierStokes equations: streamfunctionvelocity formulation.” published in the Journal of Computational Physics presents a numerical scheme involving only ψ (*pure streamfunction formulation*).

DS2: Involving only ψ

- Its convenient to interpolate a single variable on the virtual boundaries of the finer grid instead of 2 to reduce the error which may be induced due to interpolation.
- The paper by Prof. M. M. Gupta, and Prof. Jiten C. Kalita. "A new paradigm for solving NavierStokes equations: streamfunctionvelocity formulation." published in the Journal of Computational Physics presents a numerical scheme involving only ψ (*pure streamfunction formulation*).
- Avoids the difficulties associated with the computation of vorticity values, especially on solid boundaries, encountered when solving the streamfunctionvorticity formulations.

DS2: Involving only ψ

- Its convenient to interpolate a single variable on the virtual boundaries of the finer grid instead of 2 to reduce the error which may be induced due to interpolation.
- The paper by Prof. M. M. Gupta, and Prof. Jiten C. Kalita. "A new paradigm for solving NavierStokes equations: streamfunctionvelocity formulation." published in the Journal of Computational Physics presents a numerical scheme involving only ψ (*pure streamfunction formulation*).
- Avoids the difficulties associated with the computation of vorticity values, especially on solid boundaries, encountered when solving the streamfunctionvorticity formulations.
- Avoids the difficulties associated with solving pressure equations of the conventional velocitypressure formulations of the NavierStokes equations.

Biharmonic equation and obtaining the discretization

Dirichlet problem for the biharmonic equation

$$\frac{\partial^4 \phi}{\partial x^4} + 2\frac{\partial^4 \phi}{\partial x^2 \partial y^2} + \frac{\partial^4 \phi}{\partial y^4} = f(x,y), (x,y) \in \Omega \tag{42}$$

$$\phi = g_1(x,y), \frac{\partial \phi}{\partial n} = g_2(x,y), (x,y) \in \partial\phi \tag{43}$$

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.
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.

PDE

$$\frac{\partial^4\psi}{\partial x^4} + 2\frac{\partial^4\psi}{\partial x^2\partial y^2} + \frac{\partial^4\psi}{\partial y^4} - Reu\left[\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right] - Rev\left[\frac{\partial^3\psi}{\partial y^3} + \frac{\partial^3\psi}{\partial x^2\partial y}\right] = 0 \tag{44}$$

Discretizing the PDE

The discretization can be rewritten as

$$-28\psi_{i,j}+8[\psi_{i+1,j}+\psi_{i_1,j}+\psi_{i,j+1}+\psi_{i,j-1}]-[\psi_{i+1,j+1}+\psi_{i-1,j+1}+\psi_{i-1,j-1}+\psi_{i+1,j-1}]$$

$$-3h[u_{i,j+1}-u_{i,j-1}-v_{i+1,j}+v_{i-1,j}]-\frac{Reh}{4}u_{i,j}[2(\psi_{i+1,j}-\psi_{i-1,j})-\psi_{i+1,j+1}$$

$$+\psi_{i-1,j+1}+\psi_{i-1,j-1}-\psi_{i+1,j-1}+2h(v_{i+1,j}-2v_{i,j}+v_{i-1,j})]-\frac{Reh}{4}v_{i,j}[2(\psi_{i,j+1}-\psi_{i,j-1})-$$

$$\psi_{i+1,j+1}-\psi_{i-1,j+1}+\psi_{i-1,j-1}+\psi_{i+1,j-1}-2h(u_{i,j+1}-2u_{i,j}+u_{i,j-1})]=0 \quad (45)$$

$$u_{i,j}=\frac{3}{4h}[\psi_{i,j+1}-\psi_{i,j-1}]-\frac{1}{4}[u_{i,j+1}+u_{i,j-1}] \quad (46)$$

and

$$v_{i,j}=\frac{3}{4h}[\psi_{i+1,j}-\psi_{i-1,j}]-\frac{1}{4}[v_{i+1,j}+v_{i-1,j}] \quad (47)$$

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

$$A\Psi=f(Re,\mathbf{u},\mathbf{v}) \quad (48)$$

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

Outline

LDC Grid for Numerical Simulation

The following grid structure was used to apply local defect correction to the lower left corner of the global domain.

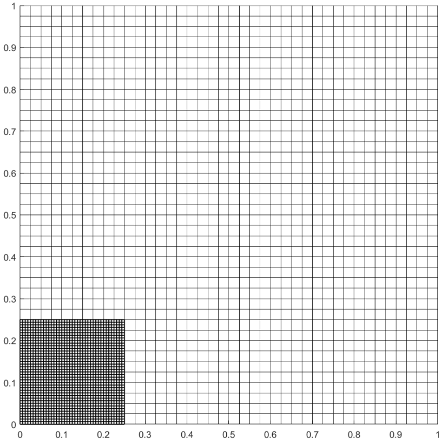

Figure 5: Grid structure for lid driven cavity

Outline

Proposed Numerical Schemes

We devised 5 schemes which we referred to as the following:

Devised Schemes

- *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 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.

Routines and sub-routines involved

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

Routines and sub-routines involved

1. `coarse_initialize()`: All the variables are initialized with the appropriate initial value(if known, like boundary conditions) or with some initial 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.

Routines and sub-routines involved

1. `coarse_initialize()`: All the variables are initialized with the appropriate initial value(if known, like boundary conditions) or with some initial 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_initialize()`: This function works in the same way as the above described `coarse_initialise()` function but for the variables and parameters associated with the finer grid.

Routines and sub-routines involved

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.

Routines and sub-routines involved

7. `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.
8. `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.

Routines and sub-routines involved

10. `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.
11. `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.
12. `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.

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 1 Coarse Convergence - Fine Convergence

```
function CCFC
    SOLVE_COARSER(tol_coarse)
    UPDATE_FINER_FROM_COARSER
    SOLVE_FINER(tol_fine)
end function
```

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

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 following is the pseudo-code for the same:

Algorithm 2 Back and Forth with Coarse Convergence - Fine Convergence

```
function BNF_CCFC(max_iter, tol_coarse, tol_fine)
    i ← 0
    while i ≤ max_iter do
        SOLVE_COARSER(tol_coarse)
        UPDATE_FINER_FROM_COARSER
        SOLVE_FINER(tol_fine)
        UPDATE_COARSER_FROM_FINER
        i ← i + 1
    end while
end function
```

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 3 Back and Forth with Coarse Convergence - Fine Finite.

```
function BNF_CFFF(max_outer_iter, iter_coarse, iter_fine)
    i ← 0
    while i ≤ max_outer_iter do
        SOLVE_COARSER(iter_coarse)
        UPDATE_FINER_FROM_COARSER
        SOLVE_FINER(iter_fine)
        UPDATE_COARSER_FROM_FINER
        i ← i + 1
    end while
end function
```

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 4 Back and Forth with Coarse Finite - Fine Finite

```
function BNF_CCFF(max_outer_iter, tol_coarse, iter_fine)
    i ← 0
    while i ≤ max_outer_iter do
        SOLVE_COARSER(tol_coarse)
        UPDATE_FINER_FROM_COARSER
        SOLVE_FINER(iter_fine)
        UPDATE_COARSER_FROM_FINER
        i ← i + 1
    end while
end function
```

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

Algorithm 5 Back and Forth with Coarse Finite - Fine Finite and subsequent convergence.

function BNF_CFFF_CONV(*max_outer_iter*, *tol_coarse*, *tol_fine*, *iter_coarse*, *iter_fine*)
    $i \leftarrow 0$
    while $i \leq$ *max_outer_iter* **do**
        SOLVE_COARSER(*iter_coarse*)
        UPDATE_FINER_FROM_COARSER
        SOLVE_FINER(*iter_fine*)
        UPDATE_COARSER_FROM_FINER
        $i \leftarrow i + 1$
    end while
    SOLVE_COARSER(*tol_coarse*)
    SOLVE_FINER(*tol_fine*)
end function

Outline

Discretization Scheme 2

Setup:

Here, we applied a 81×81 *finer* grid in the bottom left corner instead of a global finer 321×321 grid. (Extremely efficient in terms of number of grid points!)

Re was taken as 100.

Magnification was taken to be 4.

Coarser grid numerical solutions

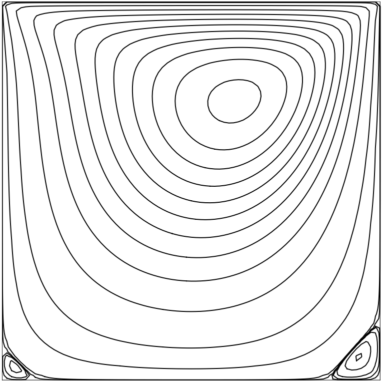

Figure 6: Contour plot of the whole domain.

Bottom left corner (BL_1) from coarser grid

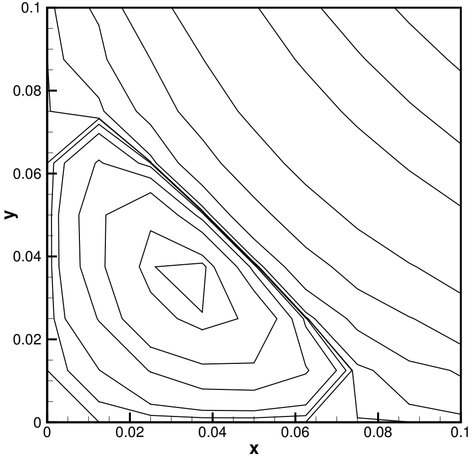

Figure 7: Contour plot of the bottom left corner (coarser grid).

Why do we need new improvised back and forth routines?

Applying previous back and forth scheme

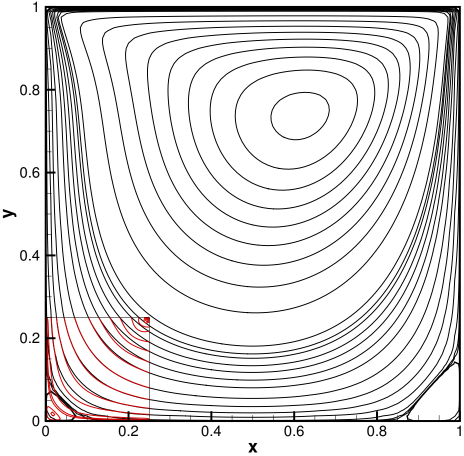

Figure 8: Combined contour plot of the whole domain.

Applying previous back and forth scheme

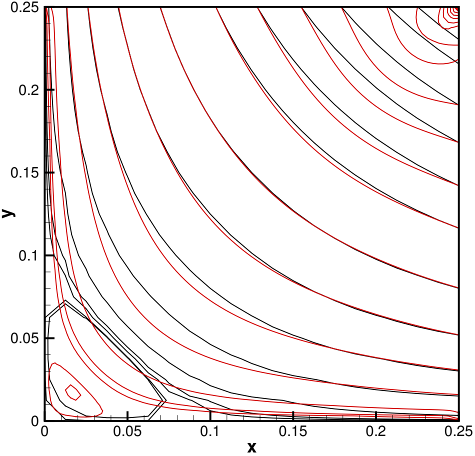

Figure 9: Conbined contour plot of the bottom left corner.

Applying new improvised schemes

Applying proposed schemes on BL_1

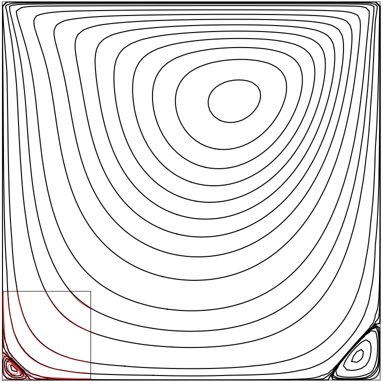

Figure 10: Combined contour plot of the whole domain.

Bottom left corner (BL_1) from finer grid

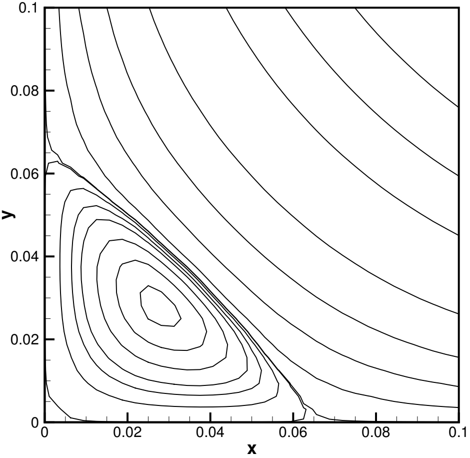

Figure 11: Contour plot of the bottom left corner (finer grid).

Comparing contour plots and smoothness

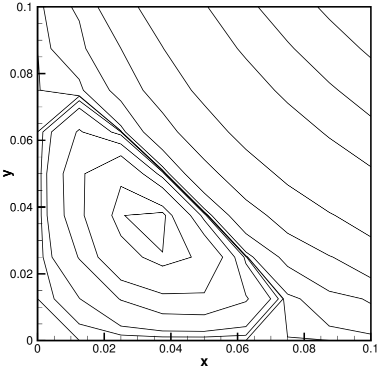

Figure 12: Contour plot of BL_1 from the coarser grid.

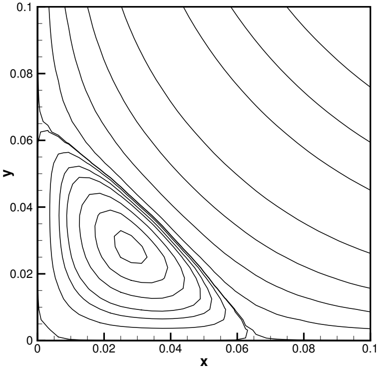

Figure 13: Contour plot of BL_1 from the finer grid.

Time comparisions

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.

Outline

Discretization Scheme 1

Setup:

Here, we applied a 81×81 *finer* grid in the bottom left corner instead of a global finer 321×321 grid. (Extremely efficient in terms of number of grid points!)

Re was taken as 100.

Magnification was taken to be 4.

Coarser Grid

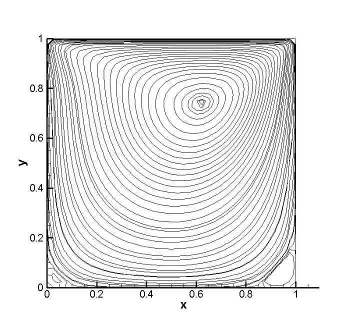

Figure 14: Contour plot of the global domain.

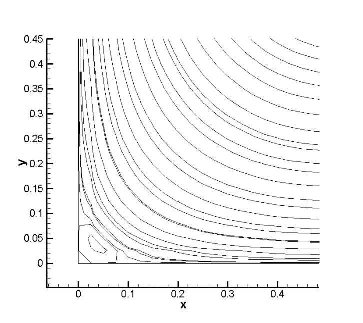

Figure 15: Contour plot of the lower left region

Observation:

Using a uniform coarser grid global domain does not give us good results in high gradient regions

Virtual BC $\simeq$ Physical BC

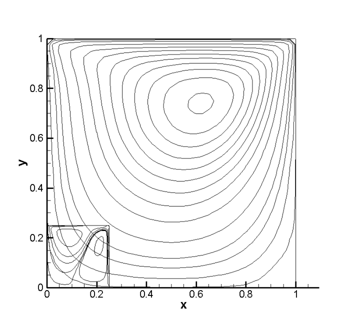

Figure 16: Contour plot of the whole domain.

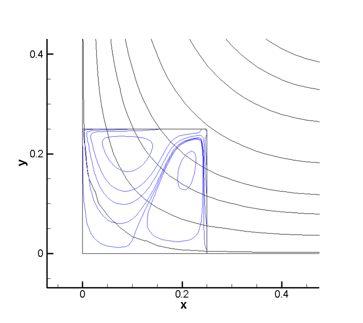

Figure 17: Contour plot of the lower right region.

Observation:

Expected results! This gives us a solution where the finer grid now corresponds to a new global domain with actual physical boundaries.

Fixed ψ and ω on boundaries

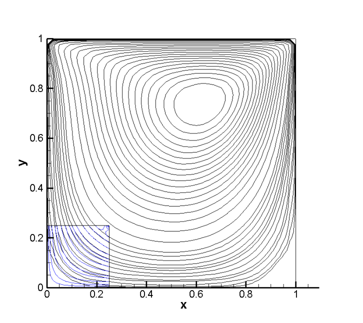

Figure 18: Contour plot of the whole domain.

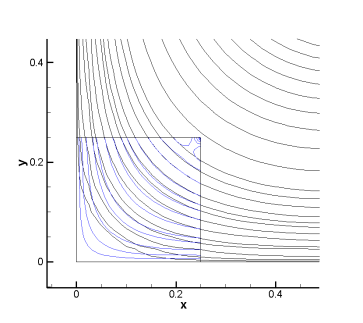

Figure 19: Contour plot of the lower right region.

Observation:

Since there is no refinement in the boundary values of the both ψ & ω, the solution resembles the overall solution but the error is significantly high.

Fixed ψ and varying ω

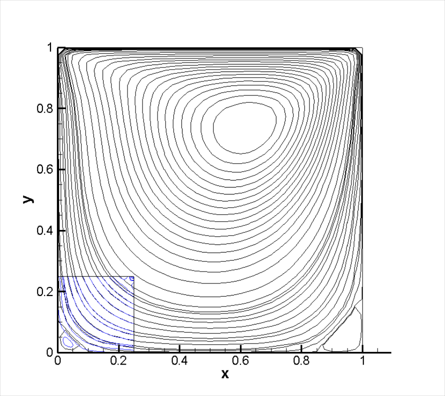

Figure 20: Contour plot of the whole domain.

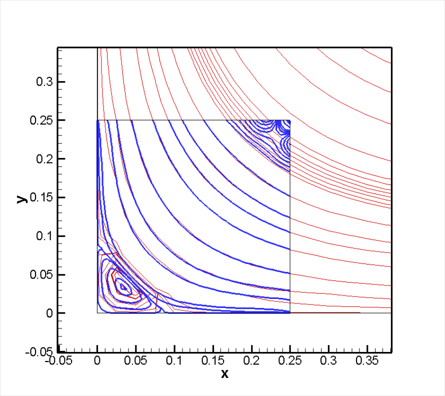

Figure 21: Contour plot of the lower right region.

Fixed ψ and varying ω (...continued)

Observation I:

When solving for finer grid, we use the discretization for ω and we update ω every time by using the new *refined* values from the interior of the finer grid and old *fixed* values from the coarser grid. This significantly improved the solution in the region of interest matching with the set benchmarks (will be discussed later).

Observation II:

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).

Moreover, the error prevails due to the interpolation in the values of ψ which are not refined over the course of the solution. Doing this, will further improve the quality of numerical solution.

BL_1 comparision

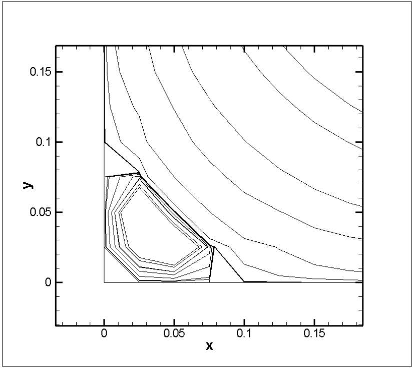

Figure 22: First BL (BL_1) from coarser grid .

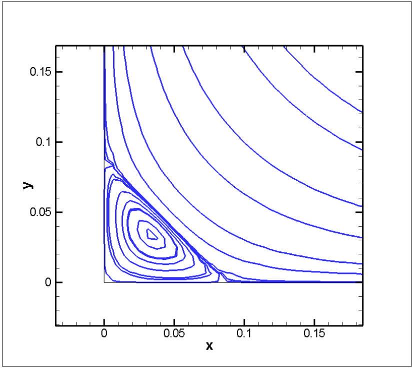

Figure 23: First BL (BL_1) from finer grid.

Comparing with Benchmark

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 $\times 10^{-2}$
(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

Grid Point Comparison

The solution mentioned in the paper uses a single 129×129 (16641) global grid whereas we evaluate by using two 41×41 grids (3262).

Immediate Improvement Strategy

- Tuning hyper parameters for over and under relaxation.
- Trying different grid structures and sizes.
- Updating ψ as well.

Future Work

- **Dynamic adaptation** to the hyper parameters involved in our schemes like relaxation parameters, outer iterations and tolerance values.

Future Work

- **Dynamic adaptation** to the hyper parameters involved in our schemes like relaxation parameters, outer iterations and tolerance values.
- Using an **ensemble** model like we use in Machine Learning to get better results.

Future Work

- **Dynamic adaptation** to the hyper parameters involved in our schemes like relaxation parameters, outer iterations and tolerance values.
- Using an **ensemble** model like we use in Machine Learning to get better results.
- Efficiently finding **tertiary vortices** with significantly lesser computation.

Future Work

- **Dynamic adaptation** to the hyper parameters involved in our schemes like relaxation parameters, outer iterations and tolerance values.
- Using an **ensemble** model like we use in Machine Learning to get better results.
- Efficiently finding **tertiary vortices** with significantly lesser computation.
- Extending to **transient** domains.

References I

Jiten C Kalita and D C Dalal and Anoop K Das
A class of higher order compact schemes for the unsteady two-dimensional convection-diffusion equations with variable coefficients
International Journal for Numerical Methods in Fluids, 38, 2002

Anoop K Das
On discretization schemes for the Lid-driven cavity problem
Private report

Jiten C Kalita and Swapan K Pandit and D C Dalal
A fourth-order accurate compact scheme for the solution of steady NavierStokes equations on non-uniform grids
Computers and fluids, 37, 2007

References II

U N Ghia and K N Ghia and C T Shin
High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method
Journal of Computational Physics, 48, 1982

Hackbusch, Wolfgang
Local defect correction method and domain decomposition techniques
Defect correction methods, 1984

Richard L Burden and J Douglas Faires
Numerical analysis.
Brooks/Cohle, Cengage Learning, 2011

THANK YOU :)