ME EN 6720: Homework 3

Christopher Mertin

Due Date: March 21, 2016

Question 1: Numerical Methods and the Vorticity-Streamfunction.....

Consider the flow in a lid driven cavity illustrated in the figure below. The cavity is square and flow in the cavity is driven by an upper plate moving at a constant velosity of U. The Reynolds number for the flow is $R_e = Uw/\nu = 100$. Solve for the steady state flow in the cavity using the vorticity-streamfunction method outlined in Lecture 12 (also see Ferziger pg. 181 and Tannehill *et. al*, pg. 650). Construct a second order finite difference solution for the problem (for both ω and Ψ) and solve it using the following schemes:

1. Jacobi Iterations
2. Gauss-Seidel
3. Gauss-Seidel with Successive Over-Relaxations (SOR)
4. Gauss-Seidel with a 1-level nested Multigrid Method

For **extra credit**, use a solver package for the following:

5. Preconditioned Conjugate Gradient Method (PCG)
6. Generalized Minimal Residual Method (GMRES)

Use a uniform mesh to solve your problem with $N_x = N_y = 50$ points in the horizontal x and vertical y directions respectively. Start the iterations with $u(x, y) = v(x, y) = 0$ (you can also explore using $u(x, y) = U$ and $v(x, y) = 0$). For a fixed convergence criteria (on the finite grid for multigrid methods), which scheme converges fastest? How do all the schemes compare to each other (measured by performance)? What is the optimum value of the relaxation coefficient for the SOR scheme? Present your results by plotting the streamlines for each scheme overlaid on a filled contour (or pseudo color) plot of the vorticity in the cavity. Also include a table comparing the number of iterations and total execution times for each method along with a short discussion. If you do the extra credit, with the PCG and GMRES methods, report at least the minimum total time.

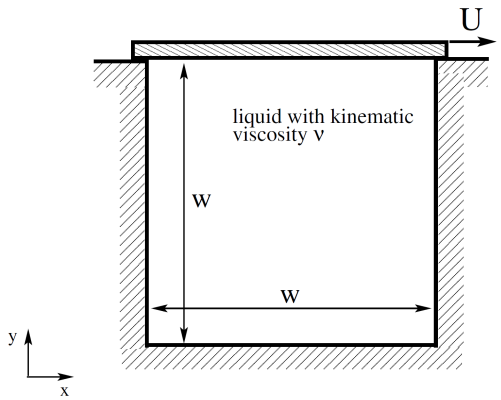

Figure 1: Given Problem

Solution: The 2D vorticity (ω) is represented as

$$\omega = \boldsymbol{\nabla} \times \mathbf{V} \tag{1}$$

with the scalar value being

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{2}$$

and the relation to the velocity from the streamfunction ψ is

$$\frac{\partial \psi}{\partial y} = u \tag{3}$$

$$\frac{\partial \psi}{\partial x} = -v \tag{4}$$

However, we also need a *time-evolution* scheme for how the system evolves. This is done with the *vorticity transport equation*, which is represented as

$$\frac{D\omega}{Dt} = \cancel{\frac{\partial \omega}{\partial t}} + (\vec{u} \cdot \boldsymbol{\nabla})\vec{\omega} = \nu \nabla^2 \omega \tag{5}$$

where the first term cancels in the case of incompressibility, leaving the 2D case as being

$$\frac{D\omega}{Dt} = u\frac{\partial \omega}{\partial x} + v\frac{\partial \omega}{\partial y} = \nu \left[\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right] \tag{6}$$

which from Equation (3) and Equation (4) we get

$$\frac{D\omega}{Dt} = \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \nu \left[\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2} \right] \tag{7}$$

We can also relate the streamfunction and the vorticity as being the Poisson equation of

$$\nabla^2 \psi = -\omega \tag{8}$$

Using these above relations, we can solve the Navier-Stokes Equations in streamfunction and vorticity form in the following steps:

1. Specify initial values for ω and ψ at $t = 0$
2. Solve the vorticity transport equation for ω at each interior point for $t + \Delta t$
3. Iterate for new values of ψ at all points by solving the Poisson Equation using the new values of ω
4. Find the velocity components from $u = \psi_y$ and $v = -\psi_x$
5. Determine the values of ω on the boundaries using ψ and ω values at the interior points
6. Return to step 2 if not converged

Before doing so though, we need to define the boundary conditions of the system. These can be seen in Figure 2.

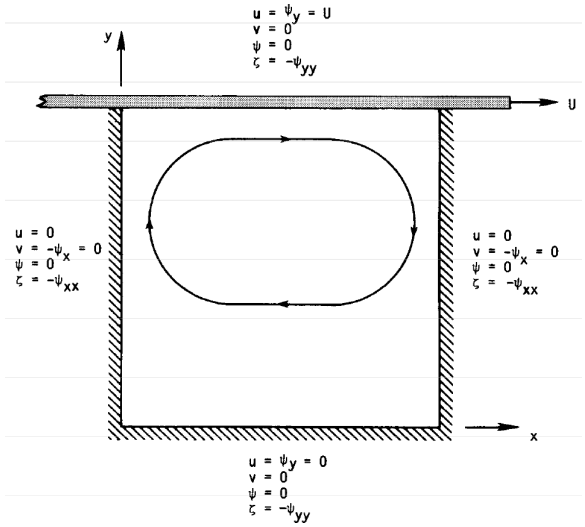

Figure 2: Driven Cavity Problem with Boundary Conditions, extracted from [1]

Finally, we need to discretize the above equations so that we can solve them numerically. When discritizing, we get the *vorticity transport equation* as being

$$\omega_{i,j}^{n+1} = \omega_{i,j}^{n} + \Delta t \left[\left(\frac{\psi_{i,j+1} - \psi_{i,j-1}}{\Delta} \right) \left(\frac{\omega_{i+1,j} - \omega_{i-1,j}}{\Delta} \right) - \left(\frac{\psi_{i+1,j} - \psi_{i-1,j}}{\Delta} \right) \left(\frac{\omega_{i,j+1} - \omega_{i,j-1}}{\Delta} \right) + \nu \left(\frac{\omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1} - 4\omega_{i,j}}{\Delta^2} \right) \right] \tag{9}$$

where Δ represents Δx and Δy since it's a uniform grid. The vorticity at the walls are

$$\omega_{N,j} = \frac{2\left(\psi_{i,j-1} - \psi_{i,j}\right)}{\Delta^2} + U\frac{2}{\Delta} \tag{10}$$

$$\omega_{1,j} = \frac{2\left(\psi_{i,j} - \psi_{i,j+1}\right)}{\Delta^2} + 0 \tag{11}$$

$$\omega_{i,1} = \frac{2\left(\psi_{i,j} - \psi_{i+1,j}\right)}{\Delta^2} + 0 \tag{12}$$

$$\omega_{i,N} = \frac{2\left(\psi_{i-1,j} - \psi_{i,j}\right)}{\Delta^2} + 0 \tag{13}$$

which comes about because we set $\psi = 0$ along the boundaries, so the $\psi_{i,j}$ terms cancel, and the second term is $\frac{\partial \psi_{i,j}}{\partial y}$ or $\frac{\partial \psi_{i,j}}{\partial x}$ depending on the boundary, which is zero for every boundary *except* the top. Finally, we can solve the Poisson Equation, Equation (8) in the following two ways, by Matrix and Matrix-free methods. For the Matrix, since we want the *boundary* values of ψ to be zero, on a 6×6 grid, the matrix would be of the form

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 4 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 4 \end{bmatrix} \tag{14}$$

where every $N-1$ rows have a zero in the upper diagonal element, and every N rows have a zero in the lower diagonal part. These zeros are used to *force* the boundary conditions of having a zero element. As can be seen, the majority of the terms are zero, so to speed up computations, only the non-zero diagonals can be stored and the matrix manipulations can be specified to this matrix structure to both reduce storage and computational cost. Alternatively, by discritizing Equation (8), we can use the Matrix-free method which produces the equation

$$\psi_{i,j} = \psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} + \Delta^2 \omega_{i,j} \tag{15}$$

where with Equation (14) or Equation (15) we can use any iterative method to solve the system. The Pseudocode for the most popular methods are defined below

Algorithm 1 Jacobi Iterative

Input: $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\vec{x}^{(0)} \in \mathbb{R}^{N \times 1}$, $\vec{b} \in \mathbb{R}^{N \times 1}$, tolerance, MAX iterations
Output: $\vec{x}^{(k)} = \mathbf{A}^{-1}\vec{b}$
  for $k = 1$ **to** MAX **do**
    for $i = 1$ **to** N **do**
      $x_i^{(k)} = \frac{-\sum_{j=1, j\neq i}^{N} \left(a_{i,j} x_j^{(k-1)}\right) + b_i}{a_{i,i}}$
    end for
    if $\left|\left|x^{(k)} - x^{(k-1)}\right|\right| < tol$ **then**
      return $x^{(k)}$
    end if
  end for

Algorithm 2 Gauss-Seidel

Input: $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\vec{x}^{(0)} \in \mathbb{R}^{N \times 1}$, $\vec{b} \in \mathbb{R}^{N \times 1}$, *tol*erance, MAX iterations
Output: $\vec{x}^{(k)} = \mathbf{A}^{-1}\vec{b}$
  for $k = 1$ **to** MAX **do**
    for $i = 1$ **to** N **do**
      $x_i^{(k)} = \frac{-\sum_{j=1}^{i-1}\left(a_{i,j}x_j^{(k)}\right) - \sum_{j=i+1}^{N}\left(a_{i,j}x_j^{(k-1)}\right) + b_i}{a_{i,i}}$
    end for
    if $\left|\left|x^{(k)} - x^{(k-1)}\right|\right| < tol$ **then**
      return $x^{(k)}$
    end if
  end for

Algorithm 3 Gauss-Seidel (SOR)

Input: $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\vec{x}^{(0)} \in \mathbb{R}^{N \times 1}$, $\vec{b} \in \mathbb{R}^{N \times 1}$, ζ SOR parameter, *tol*erance, MAX iterations
Output: $\vec{x}^{(k)} = \mathbf{A}^{-1}\vec{b}$
  for $k = 1$ **to** MAX **do**
    for $i = 1$ **to** N **do**
      $x_i^{(k)} = (1-\zeta)x_i + \frac{\zeta\left(-\sum_{j=1}^{i-1}\left(a_{i,j}x_j^{(k)}\right) - \sum_{j=i+1}^{N}\left(a_{i,j}x_j^{(k-1)}\right) + b_i\right)}{a_{i,i}}$
    end for
    if $\left|\left|x^{(k)} - x^{(k-1)}\right|\right| < tol$ **then**
      return $x^{(k)}$
    end if
  end for

Algorithm 4 MultiGrid [2]

Input: $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\vec{x} \in \mathbb{R}^{N \times 1}$, $\vec{b} \in \mathbb{R}^{N \times 1}$, ℓ level, ν_1, ν_2
Output: $\vec{x}^{(k)} = \mathbf{A}^{-1}\vec{b}$
  if $\ell = 0$ **then**
    Solve $\mathbf{A}\vec{x} = \vec{b}$ Iteratively {Gauss-Seidel}
  else
    $x_\ell = S_\ell^{\nu_1}(x_\ell, b_\ell)$
    $d_{\ell-1} = R_{\ell-1,\ell}(b_\ell - A_\ell x_\ell)$
    $v_{\ell-1} = \vec{0}$
    for $i = 0$ **to** γ **do**
      MultiGrid$(v_{\ell-1}, d_{\ell-1}, \ell - 1)$ {Recursive}
    end for
    $x_\ell = x_\ell + P_{\ell,\ell-1}v_{\ell-1}$
    $x_\ell = S_\ell^{\nu_2}(x_\ell, b_\ell)$
  end if

Algorithm 5 Pre-Conjugate Gradient [3]

Input: $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\vec{x}^{(0)} \in \mathbb{R}^{N \times 1}$, $\vec{b} \in \mathbb{R}^{N \times 1}$, $\mathbf{M}$ preconditioner, MAX iterations, tolerance
Output: $\vec{x} = \mathbf{A}^{-1}\vec{b}$

$i \leftarrow 0$
$r \leftarrow \vec{b} - \mathbf{A}\vec{x}$
$d \leftarrow \mathbf{M}^{-1}r$ {Common choice of $\mathbf{M} = \text{Diag}(\mathbf{A})$}
$\delta_{new} \leftarrow r^T d$
$\delta_0 \leftarrow \delta_{new}$
while $i < MAX$ **and** $\delta_{new} > tol^2 \delta_0$ **do**
  $q \leftarrow \mathbf{A}d$
  $\alpha \leftarrow \frac{\delta_{new}}{d^T q}$
  $x \leftarrow x + \alpha d$
  if i is divisible by 50 **then**
    $r \leftarrow b - \mathbf{A}x$
  else
    $r \leftarrow r - \alpha q$
  end if
  $s \leftarrow \mathbf{M}^{-1}r$
  $\delta_{old} \leftarrow \delta_{new}$
  $\delta_{new} \leftarrow r^T s$
  $\beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$
  $d \leftarrow s + \beta d$
  $i \leftarrow i + 1$
end while

Algorithm 6 GMRES [4]

Input: $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\vec{x}^{(0)} \in \mathbb{R}^{N \times 1}$, $\vec{b} \in \mathbb{R}^{N \times 1}$
Output: $\vec{x} = \mathbf{A}^{-1}\vec{b}$

$r_0 = b - \mathbf{A}x_0$
$\beta = ||r_0||_2$
$v_1 = r_0/\beta$
Define the $\mathbb{R}^{(m+1)\times m}$ matrix H_m and set elements $h_{i,j}$ to zero
for $j = 1$ **to** m **do**
  Compute $w_j = Av_j$
  for $i = 1$ **to** j **do**
    $h_{i,j} = (w_j, v_i)$
    $w_j = w_j - h_{i,j}v_i$
  end for
  $h_{j+1,j} = ||w_j||_2$
  if $h_{j+1,j} = 0$ **then**
    Compute y_j as the Minimizer of $||\beta e_1 - H_j y||_2^2$
    return $x_j = x_0 + V_j y_j$
  end if$v_{j+1} = w_j/h_{j+1,j}$
end for
Compute y_m as the Minimizer of $||\beta e_1 - H_m y||_2^2$
return $x_m = x_0 + V_m y_m$

All of these above algorithms can be found in `Matrix.h` which implements them for the given

pentagonal matrix that is defined above.

Unfortunately, I was unable to obtain the correct solution for the numerical results, which limits the amount that I can respond to this question. Using *any* of the above numerical methods produced a solution as portrayed in Figure 3

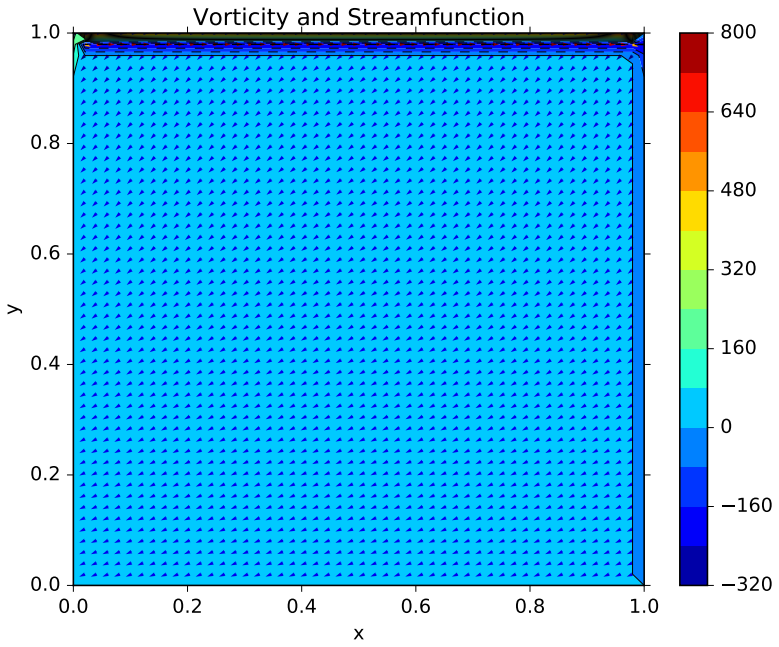

Figure 3: "Typical" results from the solver

I was unable to obtain results as it seems the boundary conditions are not "sticking" for some reason. Therefore, I was not able to test the iterative solvers on how quick they can solve the system. However, in lieu of this, I decided to test the number of iterations that each iterative solver took on an *arbitrary* system so that some results could be reported.

From Algorithm 3, there is an optimal parameter ζ which needs ot be determined to find out the most *efficient* choice. The table of the values of ζ with the number of iterations can be found in Table 1

ζ	Iterations
0.1	26
0.2	20
0.3	15
0.4	11
0.5	10
0.6	8
0.7	7
0.8	7
0.9	6
1.0	6

Table 1: Values of ζ for SOR

Note, when $\zeta = 1.0$, it is reduced down to the normal Gauss-Seidel method. From this table, the optimal value of ζ can be chosen. From here, the overall number of iterations from *all* the methods could be determined and can be seen in Table 2

Iterative Scheme	Iterations
Jacobi	11
Gauss-Seidel	6
SOR	6
PCG	114

Table 2: Iterations of Iterative Methods

For the above iterations, $\zeta = 0.9$ was chosen for the experiments. The preconditioner for PCG was chosen to be Diag($\mathbf{A}$) for the system. Changing this could vary the number of iterations in the solver greatly, though this was chosen as the "default" for the implementation.

References

[1] John C. Tannehill, *et. al*, *Computational Fluid Mechanics and Heat Transfer*

[2] Christian Wagner, *Introduction to Algebraic Multigrid*, `http://perso.uclouvain.be/alphonse.magnus/num2/amg.pdf`

[3] Jonathan Richard Shewchuck, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, `https://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`

[4] Jens Markus Melenk, `www.asc.tuwien.ac.at/~melenk/teach/iterative_SS07/part5.ps`