Ryerson University
Faculty of Engineering & Architectural Science
Department of Mechanical and Industrial Engineering

Course Number	AE8112
Course Title	Computational Fluid Dynamics and Heat Transfer
Semester/Year	Summer/Spring 2021
Instructor	Dr. Seth Dworkin

Problem Set 5

Submission Date	July 29, 2021
Programing Language Used	Fortran90

Student Name	Student Number
Ezeorah Godswill	501012886

Q1a Using explicit finite difference, we can discretize the given vorticity – velocity eqn, with the below Tylor's series expansion centered at ϕ_i

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_i = \frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{\Delta x^2} \quad \text{——(1.1)}$$

and $$\left.\frac{\partial\phi}{\partial x}\right|_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta x} \quad \text{——(1.2)}$$

Applying this to our governing equations, we will have, $\phi = \omega, v_r$ & v_z and for our 2-D coordinate

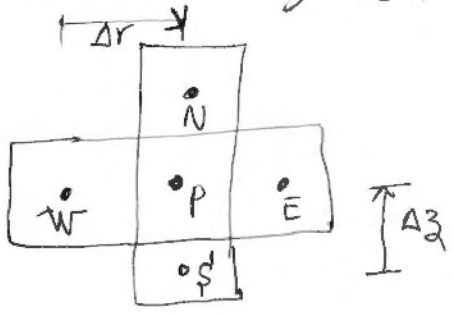
$$x = r \& z$$

Now considering vorticity transport eqn., so that after applying the product rule, we have,

$$\rho\frac{\partial\omega}{\partial t} = \mu\frac{\partial^2\omega}{\partial r^2} + \mu\frac{\partial^2\omega}{\partial z^2} + \mu\left(-\frac{1}{r^2}\omega + \frac{1}{r}\frac{\partial\omega}{\partial r}\right) - \rho v_r\frac{\partial\omega}{\partial r}$$

$$-\rho v_z\frac{\partial\omega}{\partial z} + \rho\frac{v_r\omega}{r}$$

So that applying the standard forward difference for the time discretization and eqn (1.1), (1.2), we get,



Q1 a) cont'd.:

$$\rho\left(\frac{\omega_p^{n+1}-\omega_p^n}{\Delta t}\right)=\mu\left(\frac{\omega_W^n-2\omega_p^n+\omega_E^n}{\Delta r^2}\right)+\mu\left(\frac{\omega_S^n-2\omega_p^n+\omega_N^n}{\Delta z^2}\right)-\frac{\mu\omega_p^n}{r_p^2}$$
$$+\frac{\mu}{r_p}\left(\frac{\omega_E^n-\omega_W^n}{2\Delta r}\right)-\rho v_r^n\left(\frac{\omega_E^n-\omega_W^n}{2\Delta r}\right)-\rho v_z^n\left(\frac{\omega_N^n-\omega_S^n}{2\Delta z}\right)$$
$$+\frac{\rho v_r^n\omega_p^n}{r_p}$$

now re-arranging for ω_p^{n+1} since we will be computing it's values from an we get, initial state ω_p^0.

$$\omega_p^{n+1}=\frac{\mu\Delta t}{\rho\Delta r^2}(\omega_W^n-2\omega_p^n+\omega_E^n)+\frac{\mu\Delta t}{\rho\Delta z^2}(\omega_S^n-2\omega_p^n+\omega_N^n)-\frac{\mu\Delta t\,\omega_p^n}{\rho r_p^2}$$
$$+\frac{\mu\Delta t}{2\rho r_p\Delta r}(\omega_E^n-\omega_W^n)-\frac{v_r^n\Delta t}{2\Delta r}(\omega_E^n-\omega_W^n)-\frac{v_z^n\Delta t}{2\Delta z}(\omega_N^n-\omega_S^n)$$
$$+\frac{v_r^n\omega_p^n\Delta t}{r_p}+\omega_p$$

expanding and grouping like terms, assuming $\mu, \rho = \text{const}$ we get,

$$\omega_p^{n+1}=\left(\frac{\mu\Delta t}{\rho\Delta r^2}-\frac{\mu\Delta t}{2\rho r_p\Delta r}+\frac{v_{rp}^n\Delta t}{2\Delta r}\right)\omega_W^n+\left(\frac{\mu\Delta t}{\rho\Delta z^2}+\frac{v_{zp}^n\Delta t}{2\Delta z}\right)\omega_S^n$$
$$-\left(\frac{2\mu\Delta t}{\rho\Delta r^2}+\frac{2\mu\Delta t}{\rho\Delta z^2}+\frac{\mu\Delta t}{\rho r_p^2}-\frac{v_{rp}^n\Delta t}{r_p}-1\right)\omega_p^n$$
$$+\left(\frac{\mu\Delta t}{\rho\Delta r^2}+\frac{\mu\Delta t}{2\rho r_p\Delta r}-\frac{v_{rp}^n\Delta t}{2\Delta r}\right)\omega_E^n+\left(\frac{\mu\Delta t}{\rho\Delta z^2}-\frac{v_{zp}^n\Delta t}{2\Delta z}\right)\omega_N^n$$

Q1 a | cont'd.:

For ease of computational setup, let have the above as,

$$\omega_p^{n+1} = a_W^n \omega_W^n + a_S^n \omega_S^n - a_P^n \omega_P^n + a_E^n \omega_E^n + a_N^n \omega_N^n \quad \text{——(2.2)}$$

This can be solved direcly from given initial ω_P^0

→ Now we discretize the poisson radial velocity eqn in similar way,

$$\frac{\partial^2 v_r}{\partial r^2} + \frac{\partial^2 v_r}{\partial z^2} = \frac{\partial \omega}{\partial z} + \frac{v_r}{r^2} - \frac{1}{r}\frac{\partial v_r}{\partial r} \quad \text{——(2.3)}$$

we can apply eqn (1.1) & (1.2) to discretize as,

$$\frac{v_{rW}^{n+1} - 2v_{rP}^{n+1} + v_{rE}^{n+1}}{\Delta r^2} + \frac{v_{rS}^{n+1} - 2v_{rP}^{n+1} + v_{rN}^{n+1}}{\Delta z^2} = \frac{\omega_N^{n+1} - \omega_S^{n+1}}{2\Delta z} + \frac{v_{rP}^{n+1}}{r_P^2}$$

multiplying by r_P^2 and re-arranging we get,

$$-\frac{1}{r_P}\left(\frac{v_{rE}^{n+1} - v_{rW}^{n+1}}{2\Delta r}\right)$$

$$\left(\frac{r_P^2}{\Delta r^2} - \frac{r_P}{2\Delta r}\right) v_{rW}^{n+1} + \frac{r_P^2}{\Delta z^2} v_{rS}^{n+1} - \left(\frac{2r_P^2}{\Delta r^2} + \frac{2r_P^2}{\Delta z^2} + 1\right) v_{rP}^{n+1} + \left(\frac{r_P^2}{\Delta r^2} + \frac{r_P}{2\Delta r}\right) v_{rE}$$

$$+ \frac{r_P^2}{\Delta z^2} v_{rN}^{n+1} = r_P^2 \frac{(\omega_N^{n+1} - \omega_S^{n+1})}{2\Delta z}$$

Assuming that ω_P^{n+1} from eqn (2.2), we can re-write the above as, (for ease of computational setup).

Q1 a| Cont'd.:

$$a_W v_{rW}^{n+1} + a_S v_{rS}^{n+1} - a_P v_{rP}^{n+1} + a_E v_{rE}^{n+1} + a v_{rN}^{n+1} = b^{n+1} \quad \text{——— (2.4)}$$

we will use Preconditioned Bi-CGSTAB to solve the above pentadiagonal linear system for v_r^{n+1}.

→ Now lets discretize the continuity eqn, applying the product rule we have,

$$\frac{\partial v_z}{\partial z} = -\frac{1}{r} v_r - \frac{\partial v_r}{\partial r}$$

using first order backward difference for $\frac{\partial v_z}{\partial z}$ and eq^n (1.2) for $\frac{\partial v_r}{\partial r}$, we get,

$$\frac{v_{zP} - v_{zS}}{\Delta z} = -\frac{v_{rP}^{n+1}}{r_P} - \left(\frac{v_{rE}^{n+1} - v_{rW}^{n+1}}{2\Delta r}\right)$$

re-arranging for v_{zP}^{n+1}, since v_{zS}^{n+1} can be known from our B.C., we will have,

$$v_{zP}^{n+1} = v_{zS}^{n+1} - \frac{\Delta z}{r_P} v_{rP}^{n+1} - \frac{\Delta z}{2\Delta r} v_{rE}^{n+1} + \frac{\Delta z}{2\Delta r} v_{rW}^{n+1}$$

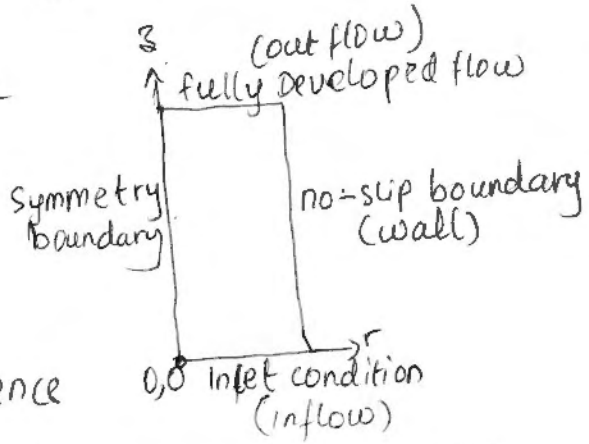
for ease of computational setup, we re-write the above as,

$$v_{zP}^{n+1} = v_z^{n+1} - a_P v_{rP}^{n+1} - a v_{rE}^{n+1} + a v_{rW}^{n+1} \quad \text{——— (2.5)}$$

This can be solved directly for the axial velocity v_{zP}^{n+1}

Q1 a) cont'd.)

>> Next, in order to apply the Boundary conditions (B.Cs), lets consider the four boundaries of our 2-D model as,



we have the vorticity for axisymmetric cylindric coordinate as,

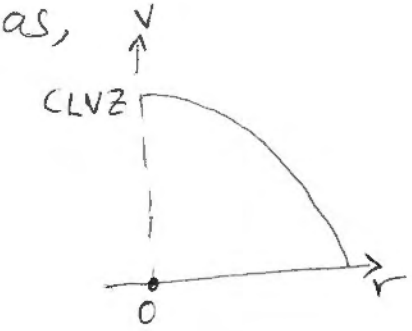
$$\omega = \frac{\partial v_r}{\partial z} - \frac{\partial v_z}{\partial r}$$

using first order one-sided difference for $\frac{\partial v_r}{\partial z}$, and a second-order centered difference for $\frac{\partial v_z}{\partial r}$, the above can be discretized as,

$$\omega_p^{n+1} = \frac{v_{rN}^{n+1} - v_{r_p}^{n+1}}{\Delta z} - \left(\frac{v_{zE}^{n+1} - v_{zW}^{n+1}}{2\Delta r} \right) \quad \text{——(2.6)}$$

and we have the half parabola eqn as,

$$v_z = CLVZ - CLVZ(r^2) \quad \text{——(2.7)}$$



So that for:

Inflow;

$$v_{zp} = CLVZ - CLVZ(r_p^2)$$

re-written as $v_{zp} = k_1$ (for computational ease

$$v_{rp} = 0$$

using eqn (2.6), we have,

$$\omega_p = \frac{v_{rN}}{\Delta z} - \left(\frac{v_{zE} - v_{zW}}{2\Delta r} \right)$$

re-written, $\omega_p = k_2$ (for computational ease

Q1a] cont'd.!

wall;

$$v_{zp} = 0$$

$$v_{rp} = 0$$

Since $\frac{\partial v_r}{\partial z} = 0$, we will have

$$\omega_p = \frac{v_{z,w}}{2\Delta r}$$

Symmetry;

$$v_{rp} = 0$$

$$\omega_p = 0$$

since $\frac{\partial v_z}{\partial z} = 0$, discretizing gives,

$$v_{zp} = v_{zs}$$

outflow;

we are give $\frac{\partial \omega}{\partial z} = \frac{\partial v_r}{\partial z} = \frac{\partial v_z}{\partial z} = 0$

and discretizing gives,

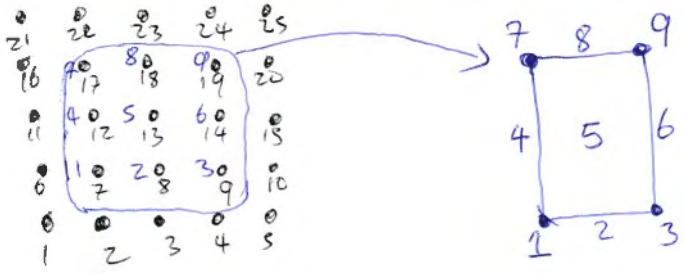
$$\omega_p = \omega_s$$

$$v_{rp} = v_{rs}$$

$$v_{zp} = v_{zs}$$

Q1a | cont'd;

>> Now because of the unique nature for solving over v_r using a linear solver, lets assume we have a sample of grid mesh,



So that considering these 9 regions we will have the following boundary for v_r,

@ [1] The bottom left region:

applying both the inflow and symmetry B.Cs to eqn (2.4)

$$-a_p v_{rp} + a_E v_{rE} + a_N v_{rN} = b$$

@ [2] The bottom region:

we apply only the inflow conditions to eqn (2.4),

$$a_W v_{rW} - a_p v_{rp} + a_E v_{rE} + a_N v_{rN} = b$$

@ [3] The bottom right region:

applying both the inflow and wall B.Cs. to eqn (2.4),

$$a_W v_{rW} - a_p v_{rp} + a_E v_{rE} + a_N v_{rN} = b$$

@ [4] The left region:

we will apply only the symmetry condition to eqn (2.4)

Q1 a) cont'd:

$$a_S v_{rS} - a_P v_{rP} + a_E v_{rE} + a_N v_{rN} = b$$

(a) [5] The middle region:

Here we just use eqn (2.4) directly

(a) [6] The right region:

we will only apply the wall condition to eqn (2.4)

$$a_W v_{rW} + a_S v_{rS} - a_P v_{rP} + a_N v_{rN} = b$$

(a) [7] The top left region

applying both the outflow and symmetry B.Cs to eqn (2.4)

$$a_S v_{rS} - a_P v_{rP} + a_E v_{rE} + a_N v_{rP} = b$$

simplifying gives, $a_S v_{rS} - (a_P + a_N) v_{rP} + a_E v_{rE} = b$

(a) [8] The top region:

we will only apply the outflow condition to eqn (2.4)

$$a_W v_{rW} + a_S v_{rS} - a_P v_{rP} + a_E v_{rE} + a_N v_{rP} = b$$

simplyfying gives,

$$a_W v_{rW} + a_S v_{rS} - (a_P - a_N) v_{rP} + a_E v_{rE} = b$$

(a) [9] The top right region:

applying both the outflow and wall B.Cs. to eqn (2.4)

$$a_W v_{rE} + a_S v_{rS} - a_P v_{rP} + a_N v_{rP} = b$$

Simplyfying gives,

$$a_W v_{rE} + a_S v_{rS} - (a_P - a_N) v_{rP} = b$$

Q1b] Using the boxed eqns from Q1a, we can write our program in fortran 90.

And we have our numeric integrated mass flow rate as,

$$\dot{m}_j = \sum_{i=1}^{Nr} (2\pi V(i)\, r\, t) \qquad \dot{m}_j = \sum_{i=1}^{Nr}$$

$$\boxed{\dot{m}_j = 2\pi\rho \sum_{i=1}^{Nr} V_{zi}\, r_i}$$

Results

>> First I ran the simulation with $Tol = 1e^{-5}$, and it seemed to be taking approximately more than 7 hours and more than 5 million iterations, yet the tolerance wasn't small enough.

>> So I reduced the tolerance to Tol= , which sufficiently small enough for a balance in both computational cost and slight accuracy.

>> Also I tried adaptive timestep size (by incrementing dt by t at each new iteration). This worked well at lesser number of iterations than the constant time step, only if the incrementing time (t) is small enough. Which in my case, I choose $t = 1e^{-7}$

for $dt = dt + t$

Otherwise the solution will explode if t is not small enough.

```
!***************Begin Header*******************************************************
!This program was written by Godswill Ezeorah, Student Number: 501012886 on July 20, 2021.
!This program solves an unsteady linear/non-linear equation using finite difference method
!and was written as a solution to AE8112 PS5 q1(b,c,d)
!**************************End Header*****************************************
program unsteady_Dfinite
    implicit none
    !Variable declaration
    DOUBLE PRECISION, dimension(:), ALLOCATABLE :: omg_i, V_r, V_z, r_i, z_i
    DOUBLE PRECISION, PARAMETER :: rh=8.3D-4, RR=1, L=10, mu=1.4D-4, CLVZ=5
    DOUBLE PRECISION, PARAMETER :: Pi = 4*atan(1.0)
    DOUBLE PRECISION :: dr, dz, dt, tols, rrr, start, finish
    INTEGER :: nr, nz, jj, i1, j1, l1, nnr, nnz

    call cpu_time(start) !gets the start time, for timing purpose
    open(2, file = 'PS5_Q1w.txt', status = 'unknown')
    open(3, file = 'PS5_Q1vr.txt', status = 'unknown')
    open(4, file = 'PS5_Q1vz.txt', status = 'unknown')
    Nr=21   !Number of grid points along radial direction
    Nz=201  !Number of grid points along axial direction
    nnr=nr-2; nnz=nz-2 !Number of grid points in the interior region
    dr=RR/(nr-1)
    dz=L/(nz-1)
    dt=1D-6     !given timestep
    tols=1D-5
    ALLOCATE(omg_i(nr*nz), V_r(nr*nz), V_z(nr*nz), r_i(nr), z_i(nz))

    !Initialization
    omg_i=0; V_r=0; V_z=0; r_i=0; z_i=0 !first initial consideration
    !rrr=0.75; omg_i=2*CLVZ*(rrr/RR**2); V_r=0; V_z=CLVZ*(1-(rrr/1)**2) !second initial consideration

    call unsteady(omg_i, V_r, V_z, r_i, z_i) !solves the unsteady system

    !Printing and writing results
    j1=nr*nz-(nr-1)
    l1=nr*nz
    jj=nz
    do i1 = j1, 1, -nr
        write(2,*) z_i(jj), omg_i(i1:l1)
        write(3,*) z_i(jj), V_r(i1:l1)
        write(4,*) z_i(jj), V_z(i1:l1)
        l1=l1-nr
        jj=jj-1
    end do
    call cpu_time(finish) !gets the end time
    write(2,*) r_i(1:nr); write(3,*) r_i(1:nr); write(4,*) r_i(1:nr)
```

```
    print *, "Computation Time = ", (finish-start)/(60*60),"hours"
    close(2);close(3);close(4)

    contains
!*******************************************************************************
    subroutine unsteady(oma, Vr, Vz, ri, zi)
        DOUBLE PRECISION, dimension(nr*nz) :: oma, Vz, omo, b, Vr
        DOUBLE PRECISION, dimension(nnr*nnz):: d, e, f, g, h, bi, Vri
        DOUBLE PRECISION, dimension(nr) :: ri
        DOUBLE PRECISION, dimension(nz) :: zi, mdot, som
        DOUBLE PRECISION :: aa,aw,as,ap,ae,an, k1, k2, k3, t, rp, zp
        DOUBLE PRECISION :: Mmdot, Amdot
        integer  :: i, j, k, nj, nk, ll, ij, iii
    !!This subroutine solves unsteady fluid flow through a pipe problem using vorticity method

        open(1, file = 'PS5_Q1.txt', status = 'unknown')
        !Variable initialization
        d=0; e=0; f=0; g=0; h=0
        t=1D-8; rp=0; zp=0
        b=0
        Vri=0
        Mmdot=1
        ij=1
        do i = 1,nz !Loop for creating the axial and radial gridpoint dimensions
            if ( i<=nr ) then
                ri(i)=rp
                zi(i)=zp
            else
                zi(i)=zp
            end if
            rp=rp+dr
            zp=zp+dz
        end do
        do while (Mmdot>tols) !Main loop for time step
            j=nr+1
            k=nr+nr
            ll=1
            omo=oma

            !The below loop solves the vorticity at each gridpoint
            do i = 1,nr*nz
                aw=mu*dt/(rh*dr**2)-mu*dt/(2*rh*ri(ll)*dr)+Vr(i)*dt/(2*dr)
                as=mu*dt/(rh*dz**2)+Vz(i)*dt/(2*dz)
                ap=2*mu*dt/(rh*dr**2)+2*mu*dt/(rh*dz**2)+mu*dt/(rh*ri(ll)**2)-Vr(i)*dt/ri(ll) -1
                ae=mu*dt/(rh*dr**2)+mu*dt/(2*rh*ri(ll)*dr)-Vr(i)*dt/(2*dr)
                an=mu*dt/(rh*dz**2)-Vz(i)*dt/(2*dz)
```

```
    k2=(Vr(i+nr)/dz)-((Vz(i+1)-Vz(i-1))/(2*dr))
    k3=Vz(i-1)/(2*dr)
    !The 9 if-statements below are for the 9 regions of our CV formulation
    if ( i==1 ) then !bottom left region
        oma(i)=0 !symmetry condition
        ll=ll+1
    else if ( i < nr ) then        !bottom region
        oma(i)=k2
        ll=ll+1
    else if ( i==nr ) then         !bottom right region
        oma(i)=k3
        ll=1
    else if ( i==nr*nz-(nr-1) ) then !top left region
        oma(i)=0 !symmetry condition
        ll=ll+1
    else if ( i==nr*nz ) then       !top right region
        oma(i)=k3 !wall condition
    else if ( i==j ) then         !left region
        oma(i)=0 !symmetry condition
        j=j+nr
        ll=ll+1
    else if ( i==k ) then         !right region
        oma(i)=k3 !wall condition
        k=k+nr
        ll=1
    else if ( i < (nr*nz)-nr ) then    !Interior region
        oma(i)=aw*omo(i-1)+as*omo(i-nr)-ap*omo(i)+ae*omo(i+1)+an*omo(i+nr)
        ll=ll+1
    else                           !top region
        oma(i)=oma(i-nr) !outflow condition
        ll=ll+1
    end if
end do
!print *,'Oma=', oma(1:nr*nz)

!The below Loop composes the radial velocity at the boundary
j=nr+1
k=nr+nr
ll=1
do i = 1,nr*nz
    !The 9 if-statements below are for the 9 regions of our formulation
    if ( i==1 ) then !bottom left region
        Vr(i)=0 !inflow or symmetry condition
        b(i)=(ri(ll)**2)*(0-0)/(2*dz)
        ll=ll+1
    else if ( i < nr ) then        !bottom region
```

```
        Vr(i)=0 !inflow condition
        b(i)=(ri(ll)**2)*(oma(i+nr)+((Vz(i+1)-Vz(i-1))/(2*dr)))/(2*dz)
        ll=ll+1
    else if ( i==nr ) then         !bottom right region
        Vr(i)=0 !wall conditions
        b(i)=(ri(ll)**2)*(oma(i+nr)-(Vz(i-1)/(2*dr)))/(2*dz)
        ll=1
    else if ( i==nr*nz-(nr-1) ) then !top left region
        Vr(i)=0 !symmetry condition
        b(i)=ri(ll)**2*(0-0)/(2*dz)
        ll=ll+1
    else if ( i==nr*nz ) then       !top right region
        Vr(i)=0
        b(i)=(ri(ll)**2)*(oma(i)-oma(i-nr-nr))/(2*dz)
    else if ( i==j ) then          !left region
        Vr(i)=0 !symmetry conditions
        b(i)=ri(ll)**2*(0-0)/(2*dz)
        j=j+nr
        ll=ll+1
    else if ( i==k ) then          !right region
        Vr(i)=0 !wall conditions
        b(i)=(ri(ll)**2)*(oma(i+nr)-oma(i-nr))/(2*dz)
        k=k+nr
        ll=1
    else if ( i < (nr*nz)-nr ) then    !Interior region
        b(i)=(ri(ll)**2)*(oma(i+nr)-oma(i-nr))/(2*dz)
        ll=ll+1
    else                           !top region
        b(i)=(ri(ll)**2)*(oma(i)-oma(i-nr-nr))/(2*dz)
        ll=ll+1
    end if
end do
nj=nnr+1; nk=nnr+nnr
j=nr+1; k=1
ll=2
do i = 1,nnr*nnz !This loop composes the interior radial velocity pentadiagonal vectors
    aw=(ri(ll)**2)/(dr**2) - ri(ll)/(2*dr)
    aa=(ri(ll)**2)/(dz**2)
    ap=(2*ri(ll)**2)/(dr**2) + (2*ri(ll)**2)/(dz**2) + 1
    ae=(ri(ll)**2)/(dr**2) + ri(ll)/(2*dr)
    !The 9 if-statements below are for the 9 regions of our formulation
    if ( i==1 ) then !bottom left region
        f(i)=-ap
        g(i+1)=ae
        h(i+nnr)=aa
        bi(i)=b(j+k)
```

```
      k=k+1
      ll=ll+1
    else if ( i < nnr ) then        !bottom region
      e(i-1)=aw
      f(i)=-ap
      g(i+1)=ae
      h(i+nnr)=aa
      bi(i)=b(j+k)
      k=k+1
      ll=ll+1
    else if ( i==nnr ) then         !bottom right region
      e(i-1)=aw
      f(i)=-ap
      h(i+nnr)=aa
      bi(i)=b(j+k)
      j=j+nr
      k=1
      ll=2
    else if ( i==nnr*nnz-(nnr-1) ) then !top left region
      d(i-nnr)=aa
      f(i)=-ap+aa
      g(i+1)=ae
      bi(i)=b(j+k)
      k=k+1
      ll=ll+1
    else if ( i==nnr*nnz ) then       !top right region
      d(i-nnr)=aa
      e(i-1)=aw
      f(i)=-ap+aa
      bi(i)=b(j+k)
    else if ( i==nj ) then         !left region
      d(i-nnr)=aa
      f(i)=-ap
      g(i+1)=ae
      h(i+nnr)=aa
      bi(i)=b(j+k)
      nj=nj+nnr
      k=k+1
      ll=ll+1
    else if ( i==nk ) then         !right region
      d(i-nnr)=aa
      e(i-1)=aw
      f(i)=-ap
      h(i+nnr)=aa
      bi(i)=b(j+k)
      nk=nk+nnr
```

```
            j=j+nr
            k=1
            ll=2
        else if ( i < (nnr*nnz)-nnr ) then    !Interior region
            d(i-nnr)=aa
            e(i-1)=aw
            f(i)=-ap
            g(i+1)=ae
            h(i+nnr)=aa
            bi(i)=b(j+k)
            k=k+1
            ll=ll+1
        else                            !top region
            d(i-nnr)=aa
            e(i-1)=aw
            f(i)=-ap+aa
            g(i+1)=ae
            bi(i)=b(j+k)
            k=k+1
            ll=ll+1
        end if
    end do
    !For computational efficiency, the, A penta-diagonal matrix is split to 5 vectors
    call Bi_CGSTAB_P(d,e,f,g,h,bi,Vri) !Solves the linear system at each time-step
    j=nr+1
    k=nr+nr
    ll=1
    do i = 1,nr*nz!This loop adds the Bi_CGSTAB solved interior to the boundary radial velocity

        if ( i>nr+1 .and. i/=j .and. i/=k) then
            Vr(i)=Vri(ll)
            ll=ll+1
        else if ( i==nr*nz-(nr-1) ) then !interior region
            ll=nnr*nnz-(nnr-1)
        else if ( i==j ) then         !left region
            j=j+nr
        else if ( i==k ) then         !right region
            k=k+nr
        end if
     end do
     !print "(a5,25f10.7)",'Vr=', Vr(1:nr*nz)

     !The below loop solves the axial velocity at each grid point
     j=nr+1
     k=nr+nr
     ll=1
```

```
do i = 1,nr*nz
    aa=dz/(2*dr)
    ap=dz/ri(ll)
    k1=CLVZ-CLVZ*ri(ll)
    !The 9 if-statements below are for the 9 regions of our CV formulation
    if ( i==1 ) then !bottom left region
        Vz(i)=k1 !inflow condition
        ll=ll+1
    else if ( i < nr ) then        !bottom region
        Vz(i)=k1
        ll=ll+1
    else if ( i==nr ) then         !bottom right region
        Vz(i)=0
        ll=1
    else if ( i==nr*nz-(nr-1) ) then !top left region
        Vz(i)=Vz(i-nr) !symmetry condition
        ll=ll+1
    else if ( i==nr*nz ) then       !top right region
        Vz(i)=0 !wall conditions
    else if ( i==j ) then         !left region
        Vz(i)=Vz(i-nr) !symmetry conditions
        j=j+nr
        ll=ll+1
    else if ( i==k ) then         !right region
        Vz(i)=0 !wall conditions
        k=k+nr
        ll=1
    else if ( i < nr*nz-nr ) then    !Interior region
        Vz(i)=Vz(i-nr)-ap*Vr(i)-aa*Vr(i+1)+aa*Vr(i-1)
        ll=ll+1
    else                           !top region
        Vz(i)=Vz(i-nr) !outflow condition
        ll=ll+1
    end if
end do
!print "(a5,25f8.5)",'Vz=', Vz(1:nr*nz)
k=0
do j = 1, nz !this loop solves the mass flow-rate along each z-gridpoint
    mdot(j) = 0.d0
    do i = 1, nr
        k=k+1
        mdot(j) = mdot(j) + 2.d0*pi*rh*Vz(k)*ri(i)
    end do
end do

do i = 1,Nz
```

```
            som(i)=abs(mdot(i)-mdot(1))
        end do
        !print "(a5,25f8.3)",'som=', som(1:nz)
        !print "(a5,25f8.3)",'mdot=', mdot(1:nz)
        Mmdot=maxval(som)/mdot(1)
        Amdot=(sum(som)/nz)/mdot(1)

        !for printing results
        print "(i9,E12.3,E12.3)",ij, Mmdot, Amdot
        if (ij == 1) then
            Mmdot=1
            iii=101
        elseif (ij <= 100) then
            write(1,*) ij, Mmdot, Amdot
        elseif (ij == iii) then
            print *, achar(27)//"[2J" !clears the console
            iii=iii+5
        elseif (ij >= 3D6) then !set the maximum number of iteration
            Mmdot=tols
        end if

        !dt=dt+t !Adaptive time-step
        ij=ij+1
    end do

    close(1)
end subroutine unsteady
!*******************************************************************************
!*******************************************************************************
subroutine Bi_CGSTAB_P(d,e,f,g,h,b,x)
    implicit none
    DOUBLE PRECISION, DIMENSION(nnr*nnz), INTENT(OUT) :: x
    DOUBLE PRECISION, dimension(nnr*nnz), INTENT(IN):: d, e, f, g, h, b
    DOUBLE PRECISION, DIMENSION(nnr*nnz) ::d1,e1,g1,h1,d2,e2,g2,h2
    DOUBLE PRECISION, DIMENSION(nnr*nnz) ::   p, r, r0, y, z, v, s, t, k, ks, kt
    DOUBLE PRECISION :: rho0, rho, w, alpha, beta, r_check, tol, nan
    INTEGER :: i
!!This subroutine solves a penta-diagonal linear system using
!!The Preconditioned Bi_CGSTAB Algorithm by Van Der Vorst

    !Variable initialization
    d2=0; e2=0; g2=0; h2=0
    !since we are dealing with vectors, I have done the multiplication of two vectors,
    !using this pattern
    !this multiples each vectors with x, for A*x
    d1=d(1:nnr*nnz)*x(1:nnr*nnz);e1=e(1:nnr*nnz)*x(1:nnr*nnz);
```

```
 g1=g(1:nnr*nnz)*x(1:nnr*nnz);h1=h(1:nnr*nnz)*x(1:nnr*nnz);
!This circularlly shifts the vectors to the appropiate position, before addition
d2(nnr+1:nnr*nnz)=d1(1:nnr*nnz-nnr);e2(2:nnr*nnz)=e1(1:nnr*nnz-1);
 g2(1:nnr*nnz-1)=g1(2:nnr*nnz)
h2(1:nnr*nnz-nnr)=h1(nnr+1:nnr*nnz)
!Hence A*x = d2+e2+f(1:nnr*nnz)*x(1:nnr*nnz)+g2+h2
r0=b-(d2+e2+f(1:nnr*nnz)*x(1:nnr*nnz)+g2+h2)
r=r0
w=1; alpha=1; rho0=1; r_check=1
v=0; p=0;  k=0
tol=10E-8
!For the inverse of K
do i = 1, nnr*nnz
    k(i)=1/f(i)
end do
i=0
!main algorithm loop
do while (r_check>tol)
    i=i+1
    rho=dot_product(r0,r)
    beta=(rho/rho0)*(alpha/w)
    rho0=dot_product(r0,r)
    p=r+beta*(p-w*v)
    y=k(1:nnr*nnz)*p(1:nnr*nnz)
    d1=d(1:nnr*nnz)*y(1:nnr*nnz);e1=e(1:nnr*nnz)*y(1:nnr*nnz);
     g1=g(1:nnr*nnz)*y(1:nnr*nnz);h1=h(1:nnr*nnz)*y(1:nnr*nnz);
    d2(nnr+1:nnr*nnz)=d1(1:nnr*nnz-nnr);e2(2:nnr*nnz)=e1(1:nnr*nnz-1);
     g2(1:nnr*nnz-1)=g1(2:nnr*nnz)
    h2(1:nnr*nnz-nnr)=h1(nnr+1:nnr*nnz)
    v=(d2+e2+f(1:nnr*nnz)*y(1:nnr*nnz)+g2+h2)
    alpha=rho/dot_product(r0,v)
    if (rho==0) then
        alpha=0
    end if
    s=r-alpha*v
    z=k(1:nnr*nnz)*s(1:nnr*nnz)
    d1=d(1:nnr*nnz)*z(1:nnr*nnz);e1=e(1:nnr*nnz)*z(1:nnr*nnz);
     g1=g(1:nnr*nnz)*z(1:nnr*nnz);h1=h(1:nnr*nnz)*z(1:nnr*nnz);
    d2(nnr+1:nnr*nnz)=d1(1:nnr*nnz-nnr);e2(2:nnr*nnz)=e1(1:nnr*nnz-1);
     g2(1:nnr*nnz-1)=g1(2:nnr*nnz)
    h2(1:nnr*nnz-nnr)=h1(nnr+1:nnr*nnz)
    t=(d2+e2+f(1:nnr*nnz)*z(1:nnr*nnz)+g2+h2)
    d1=d(1:nnr*nnz)*s(1:nnr*nnz);e1=e(1:nnr*nnz)*s(1:nnr*nnz);
     g1=g(1:nnr*nnz)*s(1:nnr*nnz);h1=h(1:nnr*nnz)*s(1:nnr*nnz);
    d2(nnr+1:nnr*nnz)=d1(1:nnr*nnz-nnr);e2(2:nnr*nnz)=e1(1:nnr*nnz-1);
     g2(1:nnr*nnz-1)=g1(2:nnr*nnz)
```

```
            h2(1:nnr*nnz-nnr)=h1(nnr+1:nnr*nnz)
            ks=(d2+e2+f(1:nnr*nnz)*s(1:nnr*nnz)+g2+h2)
            d1=d(1:nnr*nnz)*t(1:nnr*nnz);e1=e(1:nnr*nnz)*t(1:nnr*nnz);
             g1=g(1:nnr*nnz)*t(1:nnr*nnz);h1=h(1:nnr*nnz)*t(1:nnr*nnz);
            d2(nnr+1:nnr*nnz)=d1(1:nnr*nnz-nnr);e2(2:nnr*nnz)=e1(1:nnr*nnz-1);
             g2(1:nnr*nnz-1)=g1(2:nnr*nnz)
            h2(1:nnr*nnz-nnr)=h1(nnr+1:nnr*nnz)
            kt=(d2+e2+f(1:nnr*nnz)*t(1:nnr*nnz)+g2+h2)
            nan=dot_product(kt,ks)
            w=nan/dot_product(kt,kt)
            if (nan==0) then
                w=0
            end if
            x=x+alpha*y+w*z
            r=s-w*t
            r_check=norm2(r)
        end do
    end subroutine Bi_CGSTAB_P
!*******************************************************************************

end program unsteady_Dfinite
```

Q1 b) cont'd!

>> For computation time, I observed that slight adjustment to the tolerance value greatly affects the computation time, ~~For instance~~ if adaptive timestep is not used. Even after 7 hours of simulation on a constant timstep (dt), the tol=1e-5 is not still reached. But I was able to simulate for 3million iterations at just 3.088 hours using vector only Bi-CGSTAB instead of the conventional matrix type. This improved my computational time greatly.

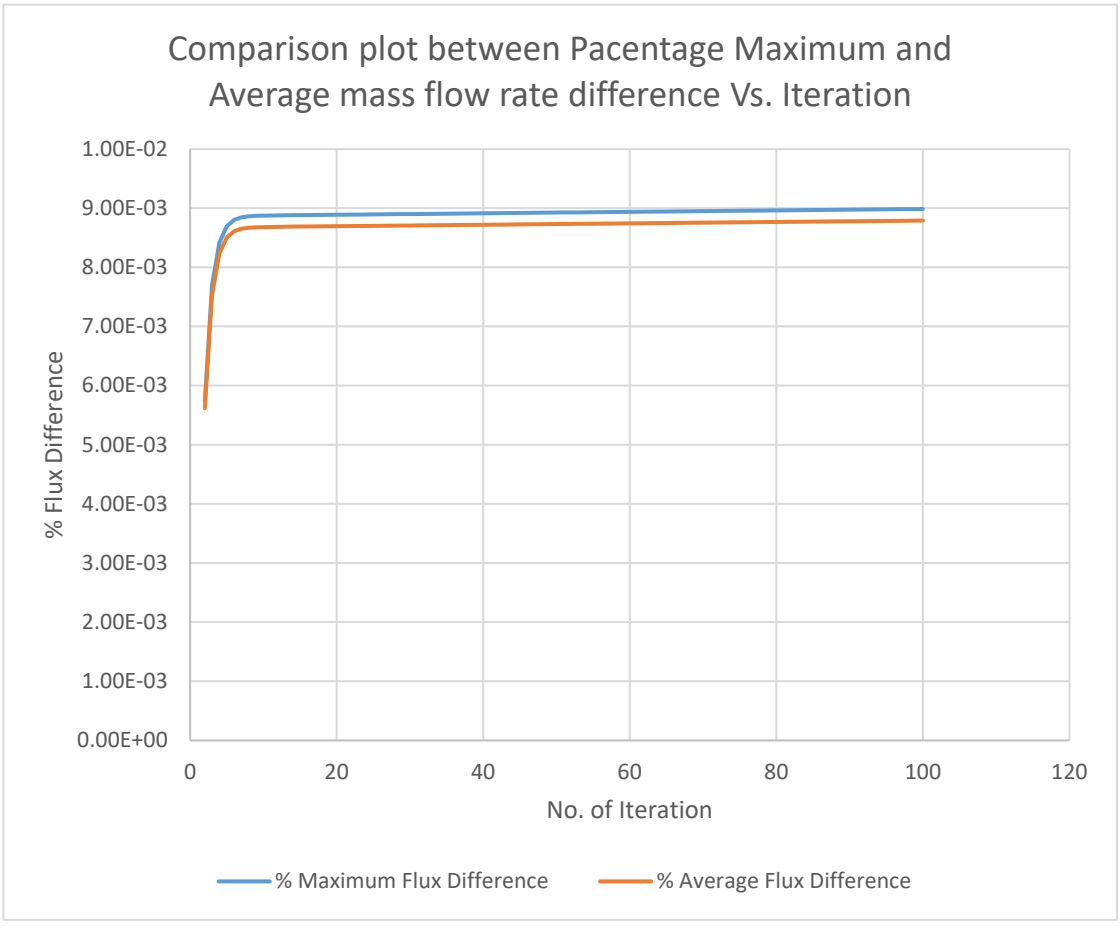
>> The accuracy which is a direct function of our percentage maximum/average flow rate difference (i.e based on ~~the~~ conservation law), will be as accurate as we want it to be, in the expense of our computational time.

But what I've been able to deduce from our simulation, is that the solution gets ~~more~~ slightly inaccurate ~~as we increas~~ at new iterations up to about 50,000 iterations. And then it starts to get accurate again and on-ward in expense of computation time.

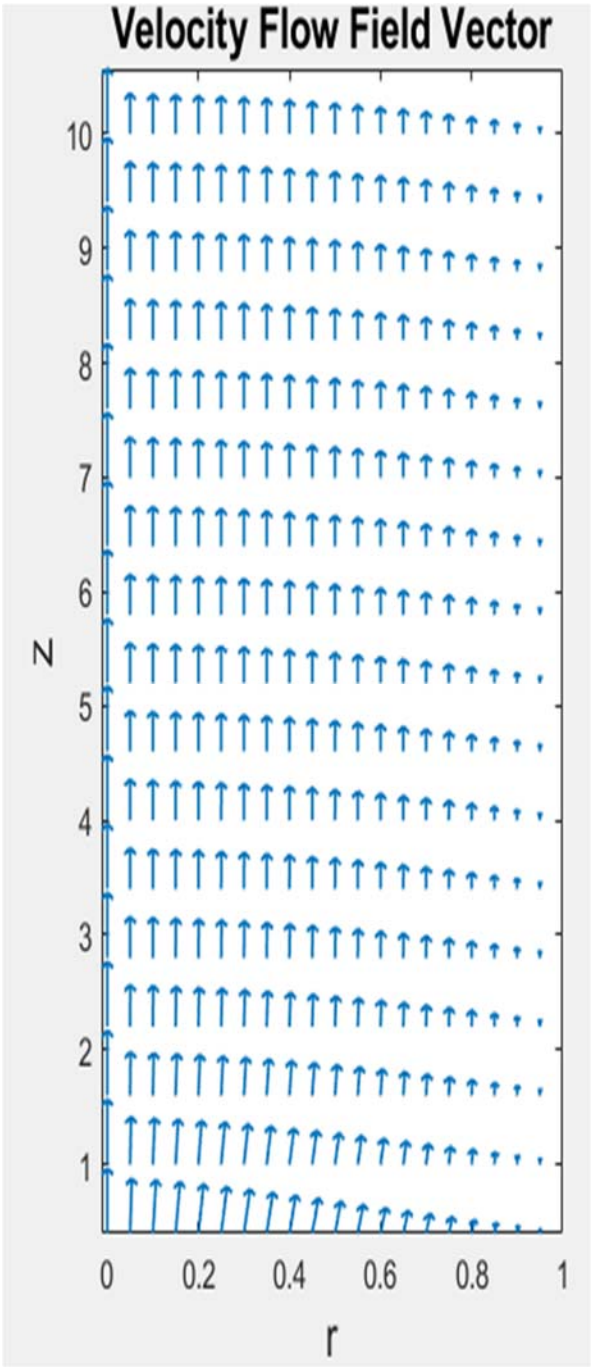
Q1c) >> From below plot, we can see that both percentage Maximum and average flow rate difference have similar behaviour against the number of iteration. And thus either one of them can be used to checks for the accuracy of our solution. (This was taken from samples of just 100 iterations).

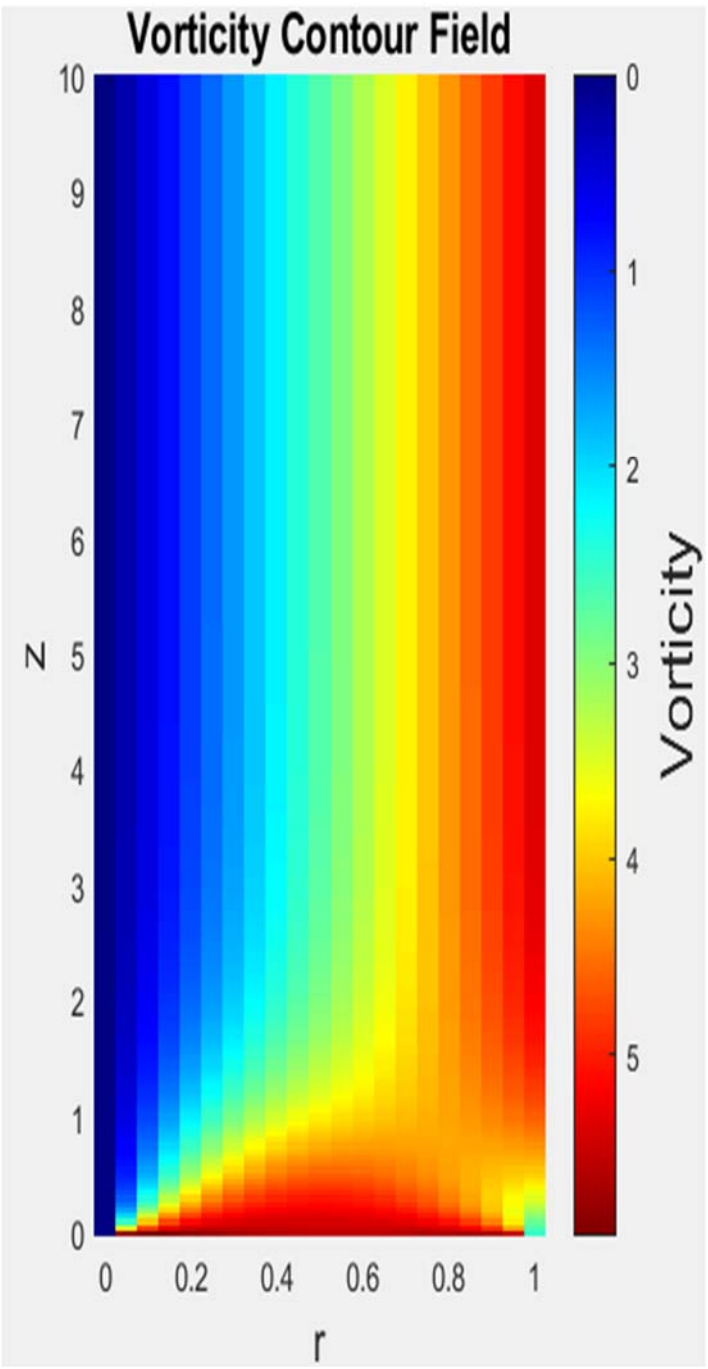
>> And setting our tolerance to $tol = 8e^{-3}$ is small enough for us to understand the behaviour of the solution (ie. after it begins to converge again, from about 50,000 iterations). With this we got about 3 million iterations. And the vector plot for the flow field, as while as the contour plot for the vorticity is shown below;

Q1c] cont'd.]


Comparison plot between Pacentage Maximum and Average mass flow rate difference Vs. Iteration
1.00E-02
9.00E-03
8.00E-03
7.00E-03
6.00E-03
5.00E-03
4.00E-03
3.00E-03
2.00E-03
1.00E-03
0.00E+00
% Flux Difference
0
20
40
60
80
100
120
No. of Iteration
% Maximum Flux Difference
% Average Flux Difference

Q1c] cont'd.]






```
Computational Time = 3.0881987847222221 hours
```

Q1 d| If we assume an initial guess of;

$$v_r = 0 \quad , \quad v_z = CLVZ\left[1 - \left(\frac{r_{guess}}{R}\right)^2\right] \quad , \quad \omega = 2(CLVZ)\left[\frac{r_{guess}}{R^2}\right]$$

lets assume R to be our overall radius $R = 1cm$ and let's play with $0 \leq r \leq 1cm$ (we can call this r_guess).

>> The sensitivity of varying this r_guess is shown in the table below. Which at $r_{guess} = 0.5$ we can see the solution converging very at just 9 iteration.

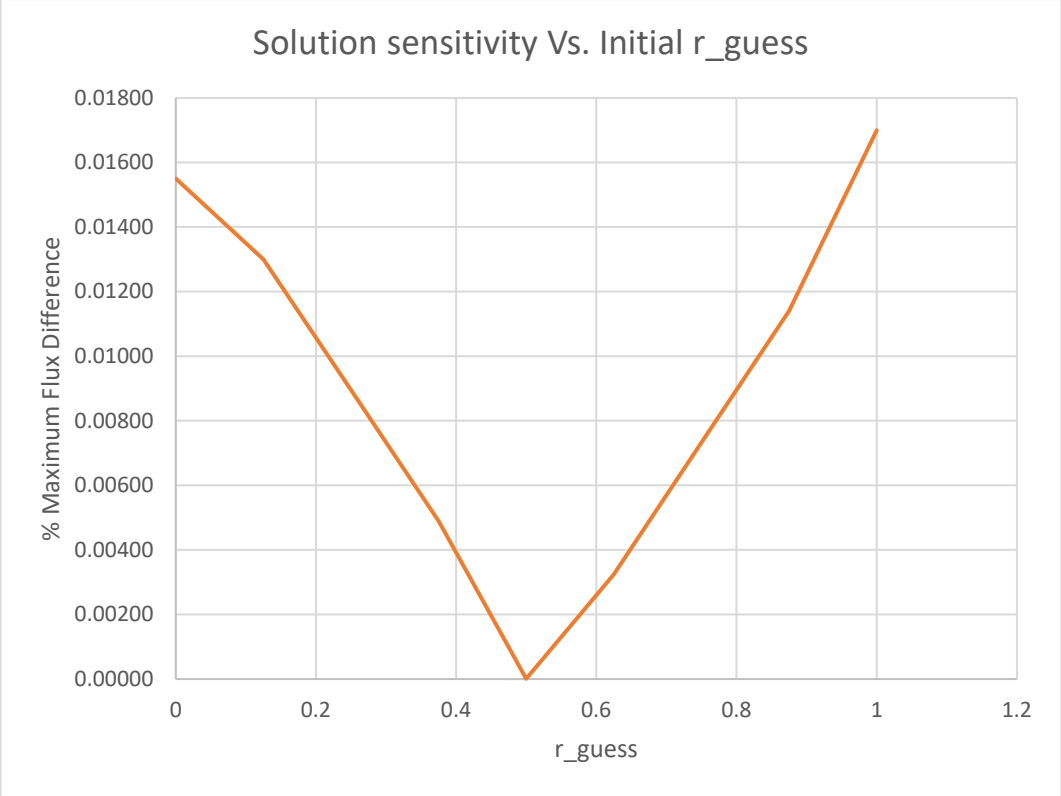
>> But the strange part is with the vector and contour plot which is shown below for $r_{guess} = 0.5$ and it looks much different from the solution if

$$v_r = 0 \; , v_z = 0 \; \& \; \omega = 0 \; .$$

I also observed that $r_{guess} = 0$ is yields similar results to our previous guess of $v_r = v_z = \omega = 0$.

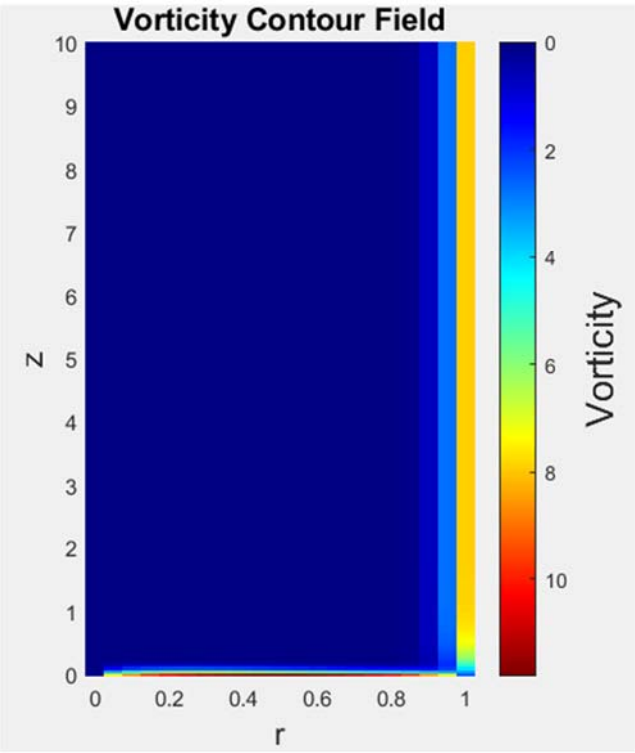
Q1d] cont'd.]

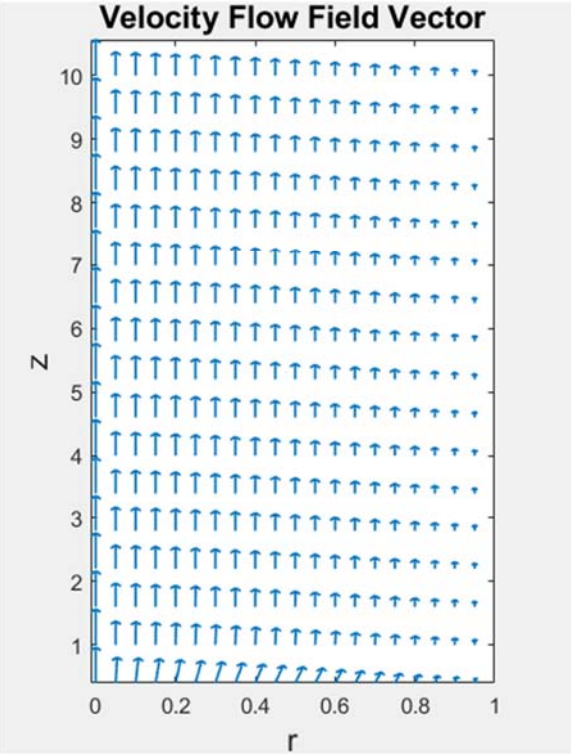
No. of Iteration	r_guess	% Maximum Flux Difference
10000	1	0.01550
10000	0.125	0.01300
10000	0.25	0.00895
10000	0.375	0.00489
9	0.5	0.00001
10000	0.625	0.00324
10000	0.75	0.00732
10000	0.875	0.01140
10000	1	0.01700



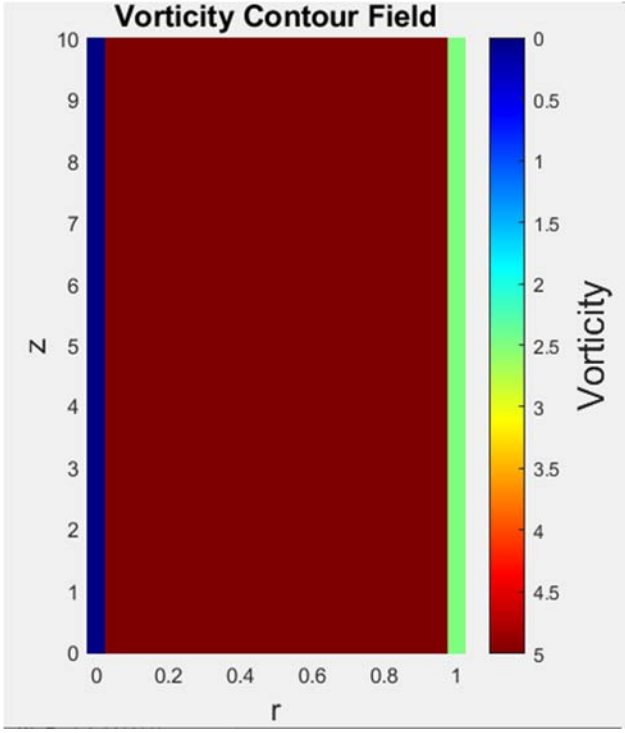
Q1d] cont'd.]

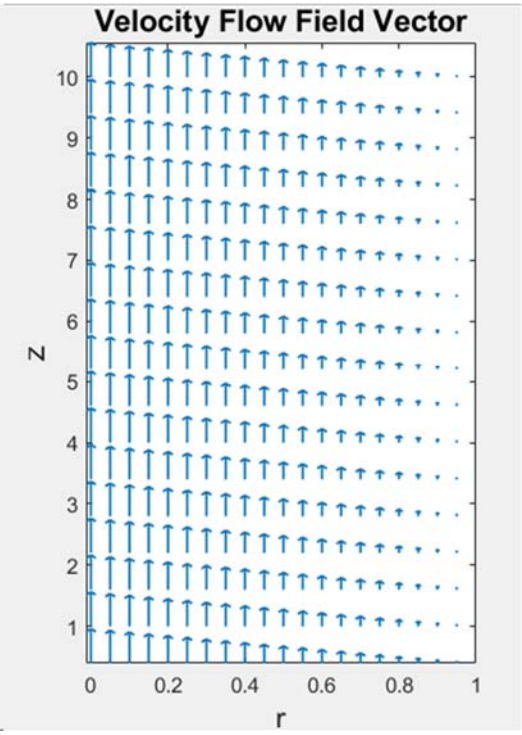
Plot when $r_{guess} = 0$:





Plot when $r_{guess} = 0.5$:





Q1d] cont'd.]

Plot when $r_{guess} = 1$: