COMPUTER PROJECT #3

Shaun Harris
Department of Mechanical and Aerospace Engineering
Utah State University
Email: shaun.r.harris@gmail.com

ABSTRACT

A staggard grid Navier-Stokes solver is implemented to solve a driven cavity problem, and a channel flow problem. The Pressure, u-velocity and v-velocity are all staggard and solved for separatley. The necessary equations and the implemented code is provided in this paper.

NOMENCLATURE

u Velocity in the x-direction (m/s)
u_i Velocity in the x-direction referencing neighbor $i(N,E,S,W,P)$ lowercase (n,e,s,w) indicates averaged values
u_P^{old} Velocity in the x-direction from previous iteration on node center
v Velocity in the y-direction
v_i Velocity in the y-direction referencing neighbor $i(N,E,S,W,P)$ lowercase (n,e,s,w) indicates averaged values
v_P^{old} Velocity in the y-direction from previous iteration on node center
P Pressure
i' Correction term for $i(u,v,P)$
$a_{i,j}$ Coefficient for final discretized equation referencing neighbor $i(N,E,S,W,P)$ on $j(u,v,P)$ mesh
$\tilde{a}_{P,j}$ Coefficient for final discretized equation referencing center P on $j(u,v,P)$ mesh and divided by Ω correction factor
Ω non-linear correction factor for momentum equations
Ω_P linear correction factor for pressure equation
α Pressure blending factor

CONTENTS

1 INTRODUCTION

In order to solve the Navier Stokes equation in two dimensions a staggered grid approach was used. This where the u, v, and P values were all saved in separate locations on the grid. This allowed for many of the oscillations to be minimized and for the solution to converge to a correct solution.

Two cases were considered in this problem. These cases were a driven cavity and a channel flow. The inputs and requirements are shown on the problem outline.

The numerical method, application with code, and results are shown in the following sections.

2 NUMERICAL METHOD

In order to solve using this method, a staggered grid was utilized. Fig. 1 shows how the u, v, and P values were saved on the grid. The momentum equation is discretized from Eq. 1 to 2.

$$\frac{\partial(\rho uu)}{\partial x}+\frac{\partial(\rho vu)}{\partial y}=-\frac{\partial P}{\partial x}+\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial u}{\partial y}\right)+\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right) \tag{1}$$

$$\begin{aligned} u_P &= (1-\Omega)u_P^{old}+\frac{1}{\tilde{a}_{P,u}}\left[a_{E,u}u_E+a_{W,u}u_W+a_{N,u}u_N+a_{S,u}u_S+dy(P_w-P_e)\right] \\ &\textit{where:} \\ a_{E,u} &= max(-\rho u_e dy,0)+\mu dy/dx \\ a_{N,u} &= max(-\rho v_n dx,0)+\mu dx/dy \\ a_{W,u} &= max(\rho u_w dy,0)+\mu dy/dx \\ a_{S,u} &= max(\rho v_s dx,0)+\mu dx/dy \end{aligned} \tag{2}$$

The following equations (Eq. 3 and 4) show the discretized equations used for v velocity momentum.

$$\frac{\partial(\rho uv)}{\partial x}+\frac{\partial(\rho vv)}{\partial y}=-\frac{\partial P}{\partial x}+\frac{\partial}{\partial x}\left(\mu\frac{\partial v}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial x}\left(\mu\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial y}\left(\mu\frac{\partial v}{\partial y}\right) \tag{3}$$

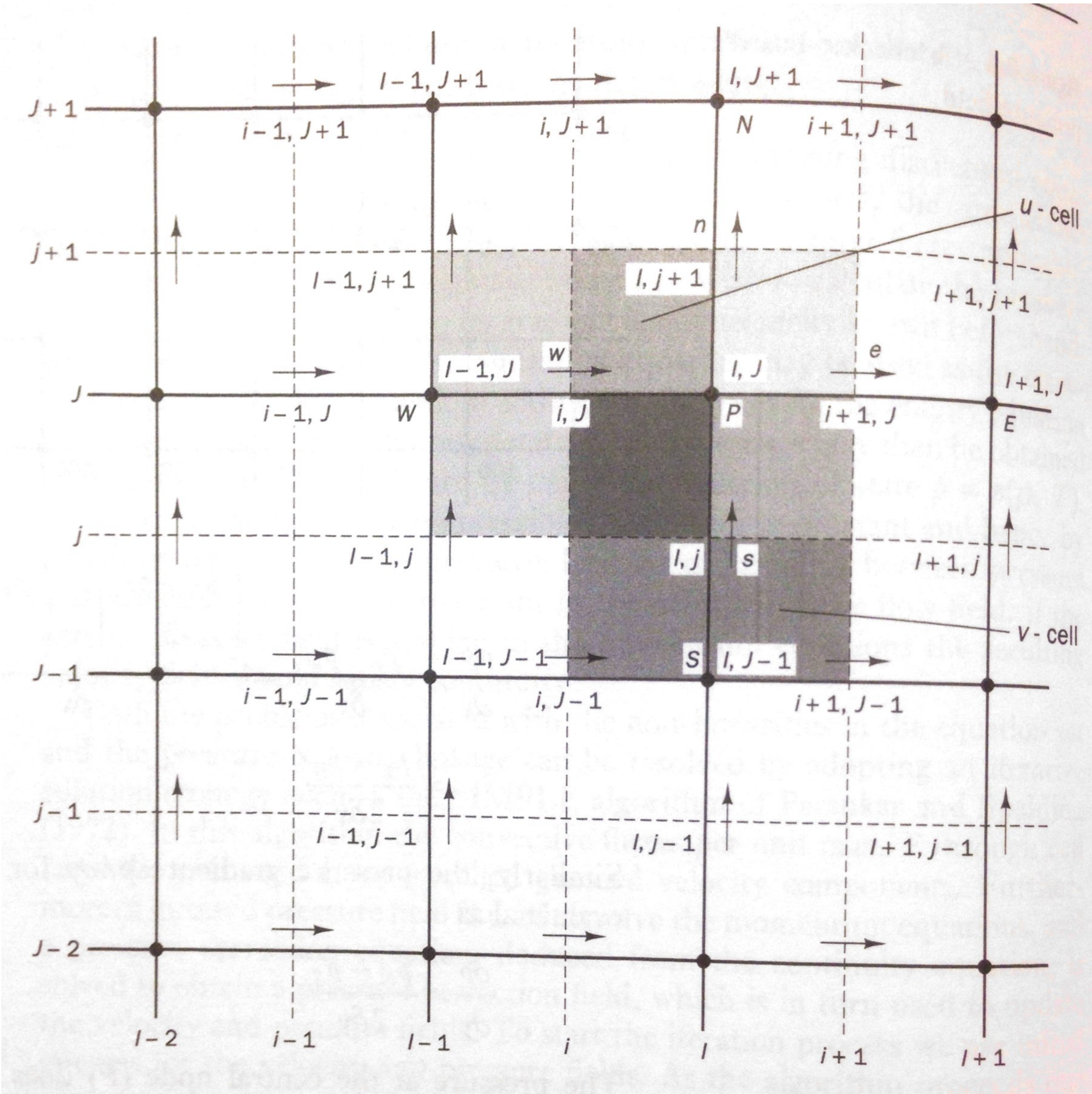


FIGURE 1. REPRESENTATION OF STENCIL FOR GRID GENERATION

$$v_P = (1-\Omega)v_P^{old} + \frac{1}{\tilde{a}_{P,v}}\left[a_{E,v}v_E + a_{W,v}v_W + a_{N,v}v_N + a_{S,v}v_S + dy(P_s - P_n)\right]$$

where:

$$\begin{aligned} a_{E,v} &= max(-\rho u_e dy, 0) + \mu dy/dx \\ a_{N,v} &= max(-\rho v_n dx, 0) + \mu dx/dy \\ a_{W,v} &= max(\rho u_w dy, 0) + \mu dy/dx \\ a_{S,v} &= max(\rho v_s dx, 0) + \mu dx/dy \end{aligned} \qquad (4)$$

It should be noted that these equations took into account the spacing on the boundary. That is, if there was a ghost node that was used in the coefficient calculations, then the $\mu dx/dy$ like terms became $\mu 2dx/dy$ terms on the North and South boundaries for the u momentum calculations.

The pressure was discretized from continuity, staggered control volume equations, and velocity correction terms. Thus, the continuity equation shown in Eq. 5 is discretized to Eq. 6.

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \tag{5}$$

$$\begin{aligned} P'_P &= P'_P + \frac{\Omega_P}{a_{P,P}} \left(a_{E,P}P'_E + a_{W,P}P'_W + a_{N,P}P'_N + a_{S,P}P'_S - S - a_{P,P}P'_P \right) \\ P &= P^{old} + \alpha P' \end{aligned} \tag{6}$$

It should be noted that momentum is non-linear so the relaxation factors used were $\Omega \approx 0.6$ and the $\Omega_P \approx 1.7$ while $\alpha \approx 0.3$ for the linear pressure equation.

These equations were implemented into a code structure shown in the diagram in Fig. 2. The code is referenced in Sec. A. It is also noted that the velocity correction terms were also implemented as shown in Eq. 7 and implemented as depicted in Fig. 2.

$$\begin{aligned} u'_P &= \frac{dy}{a_{W,u}}(P_W - P_P) \\ v'_P &= \frac{dx}{a_{S,u}}(P_S - P_P) \end{aligned} \tag{7}$$

3 RESULTS

3.1 Driven Cavity

The following plots show the u, v, P and the iterations required to converge. Additionally, the $x = 0.5$ u values are provided.

The below two plots show the u (left) and v (right) velocity contours.

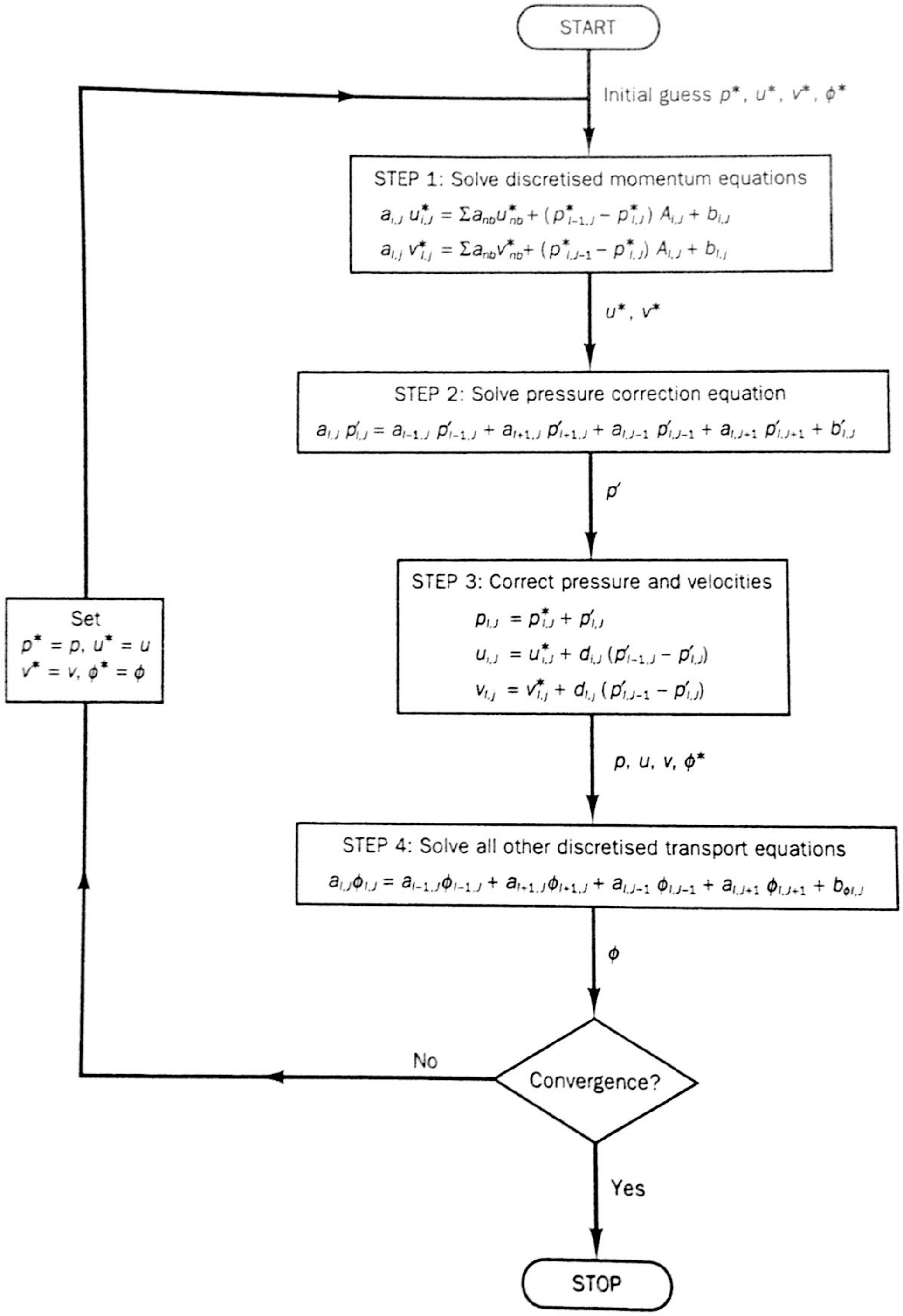
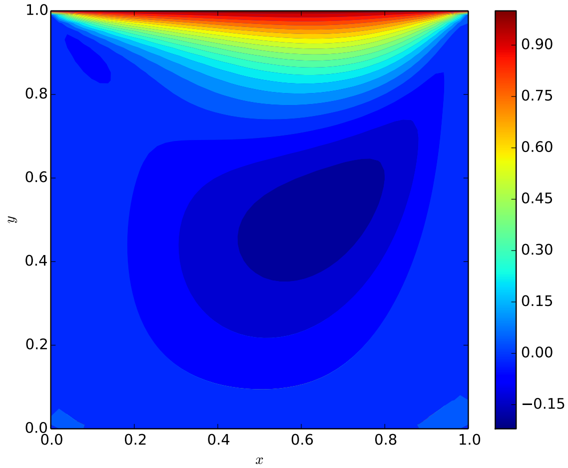
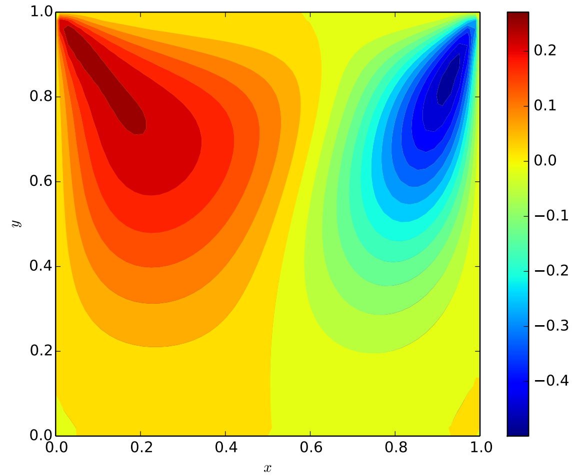
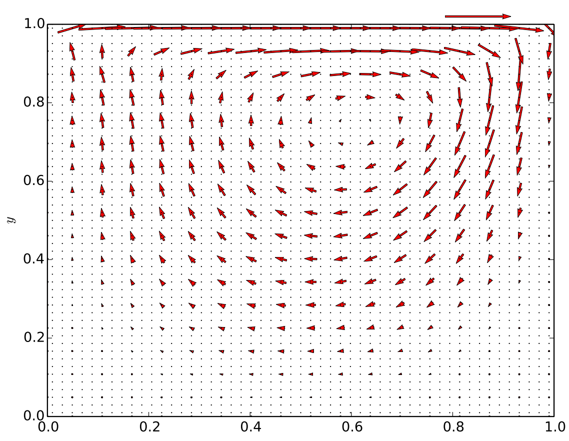


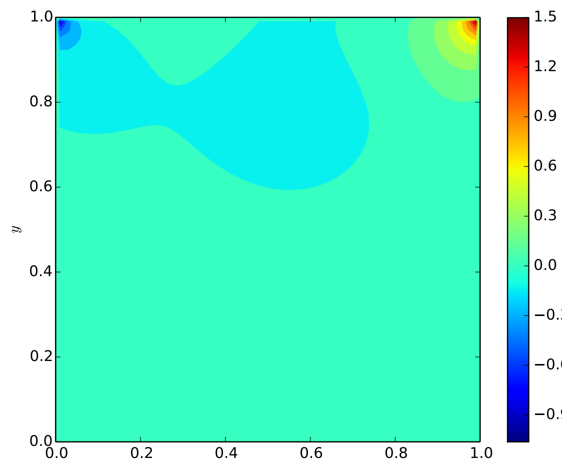
FIGURE 2. Outline of code structure



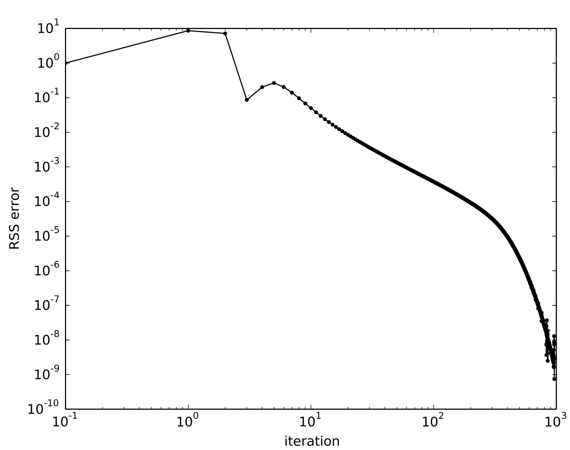


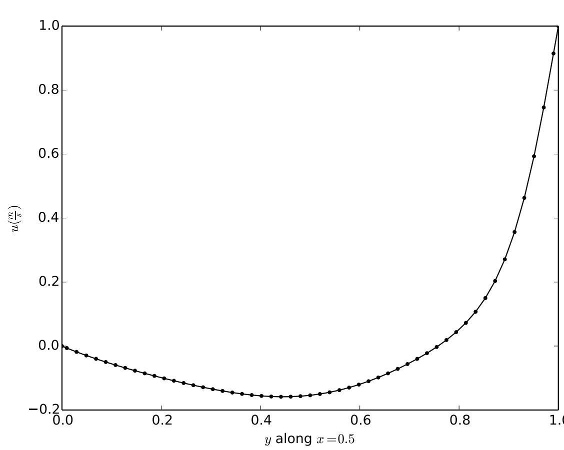
The below two plots show the vector plots of velocity magnitude (left) and the pressure contour (right).





The below two plots show the error vs iterations (left) and the $x = 0.5$ u values (right).





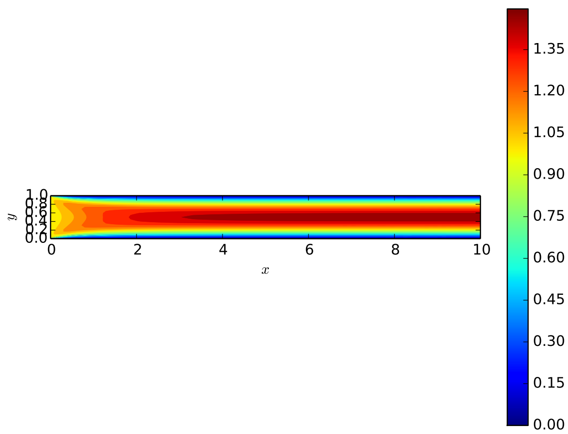
The final mass imbalance for this 51X51 cell simulation was shown to be 1.60E-017.

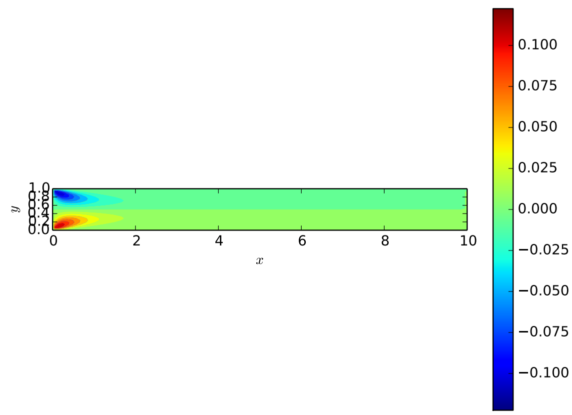
3.2 Channel Flow

For this solution to function properly, the proper boundary condition needed to be applied. Since the channel was long enough, the boundary condition to be applied was in the u momentum solver. We just set the outside condition to be equal to the flow directly upstream. This allowed for flow to flow outside of the wall boundary.

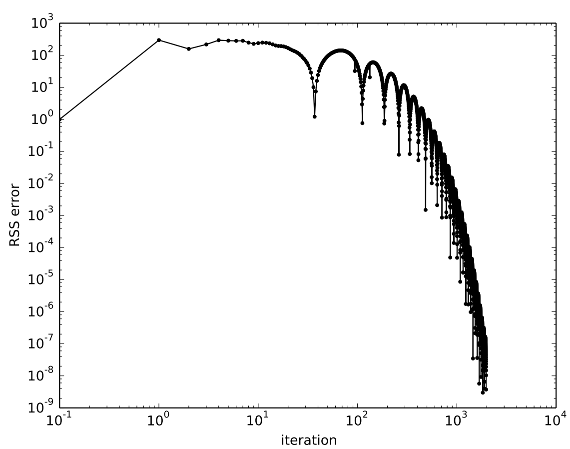
In addition to the plot provided in the above section. The u velocity for the centerline of the duct is shown. The wall shear stress is also shown along both the upper and lower walls from the inlet to the outlet.

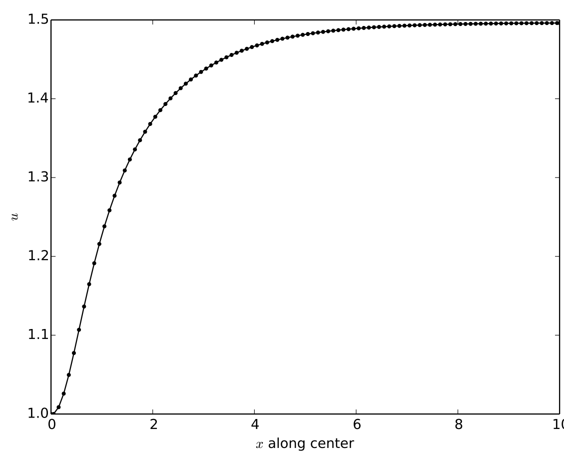
The below two plots show the u (left) and v (right) velocity contours.



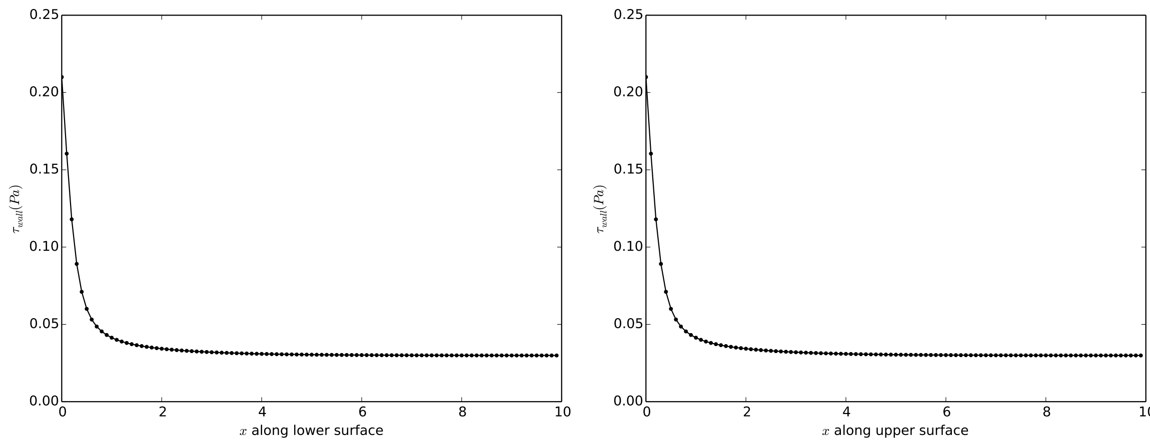


The below two plots show the error vs iterations (left) and the $y = 0.5$ u values (right).





The below two plots show the wall shear stress on the lower (left) and upper (right) surfaces.



The final mass imbalance for this 21X100 cell problem was shown to be 7.44E-023

4 CONCLUSION

We have demonstrated a computational fluid dynamics solver for a driven cavity and for a channel flow study. We have shown the staggered grid approach using two dimensional u, v, and P solvers.

A Code

A.1 Subroutines

```
MODULE types
    !purpose: define data type struct
    IMPLICIT NONE
    ! Properties of fluid flow
    REAL    :: Omega = 0.6 ! Relaxation factor for momentum non-linear
    REAL    :: OmegaP= 1.7 ! Relaxation factor for pressure correction linear
    REAL    :: alpha = 0.3 ! relaxation factor for pressure correction
    REAL    :: mu = 0.01   ! dynamic viscosity
    REAL    :: rho= 1.     ! density
    REAL    :: Convergence = 1.e-14
    REAL    :: Convergence2= 1.e-9
    INTEGER :: max_iter = 1000000
    INTEGER :: max_iter2= 700
    INTEGER :: max_iter3= 7000
    TYPE::dat
        REAL::xu,yv,xp,yp
        REAL::u,v,u_old,v_old !u,v is in bottom left corner, or south and west sides of cell
        REAL::APu,AEu,ANu,ASu,AWu,Apv,AEv,ANv,ASv,AWv,APp,AEp,ANp,AWp,ASp
        REAL::P,Pp,P_old
        REAL::S ! source terms
        INTEGER::n
    END TYPE dat
CONTAINS
    SUBROUTINE set_xy (strct,dx,dy,nx,ny,x,y)
        real,intent(in)    :: dx,dy,x,y
        integer,intent(in) :: nx,ny   ! size of strct in x and y directions
        type(dat),dimension(0:,0:),intent(inout)::strct ! data contained from 0:nx-1 where cells 0 and nx-1 are boundary nodes (cell volume approaches 0 on boundary nodes)
        integer :: i,j,n  ! for do loops and n is counter for cell number
        real    :: xi,yi  ! x and y values for each cell
        ! left boundary
        strct(0,:)%xp= 0.
        strct(0,0)%yp= 0.
```

```fortran
        strct(0,1:ny-1)%yp= reshape((/ (i*dy - dy/2. ,i=1,ny-1) /),(/ ny-1/))
        strct(0,ny+0)%yp= y
        ! bottom boundary
        strct(0,0)%xp= 0.
        strct(1:nx-1,0)%xp= reshape((/ (i*dx - dx/2. ,i=1,nx-1) /),(/ nx-1/))
        strct(nx+0,0)%xp= x
        strct(:,0)%yp= 0.
        ! right boundary
        strct(nx+0,:)%xp= x
        strct(nx+0,:)%yp= strct(0,:)%yp
        ! top boundary
        strct(:,ny+0)%xp= strct(:,0)%xp
        strct(:,ny+0)%yp= y

        n=1                     ! cell number 1
        DO i=1,ny-1             ! 1 to ny-2 for boundary nodes (we only are iterating through the middle values)
            yi = i*dy - dy/2.   ! y coordinate
            DO j=1,nx-1
                xi = j*dx - dx/2.       ! x coordinate
                strct(j,i)%n = n        ! input n node
                strct(j,i)%xp= xi       ! x coordinate to strct
                strct(j,i)%yp= yi       ! y coordinate to strct
                n=n+1                   ! count cell numbers up one
            END DO
        END DO
        ! set xu and yv to similar values (but for the staggard grids of each)
        strct%xu = strct%xp - dx/2.
        strct%yv = strct%yp - dy/2.
        strct(:,ny)%yv=y      !top
        strct(:,0)%yv=0.      !bottom
        strct(0:1,:)%xu=0.    !left
        strct(nx,:)%xu=x      !right
    END SUBROUTINE set_xy

    SUBROUTINE mom_uv(strct,dx,dy,nx,ny)
        ! requires uniform grid of dx and dy spacing
        REAL,INTENT(IN)       :: dx,dy
        INTEGER,INTENT(IN)    :: nx,ny! size of strct in x and y directions
        TYPE(dat),DIMENSION(0:nx+1,0:ny+1),INTENT(INOUT)::strct ! data contained from 0:nx+1 where cells 0 and nx+1 are boundary nodes (cell volume approaches 0 on boundary nodes)
        REAL      :: mdot ! temporary value for mass flow values
        INTEGER   :: i,j,iter=0!loop iterators
        REAL      :: error=1.,error2=1.

        ! mdot and Au values
        !$OMP PARALLEL DO
        DO i=1,nx
            DO j=1,ny
                mdot              =   rho*(strct(i+1,j  )%u_old+strct(i  ,j  )%u_old)/2.*dy ! east face
                strct(i,j)%AEu    =   max(-mdot,0.) + mu*dy/dx
                mdot              =   rho*(strct(i-1,j+1)%v_old+strct(i  ,j+1)%v_old)/2.*dx ! north face
                IF (j==ny) THEN
                    strct(i,j)%ANu    =   max(-mdot,0.) + mu*2.*dx/dy
                ELSE
                    strct(i,j)%ANu    =   max(-mdot,0.) + mu*dx/dy
                END IF
                mdot              =   rho*(strct(i-1,j  )%u_old+strct(i  ,j  )%u_old)/2.*dy ! West face
                strct(i,j)%AWu    =   max( mdot,0.) + mu*dy/dx
                mdot              =   rho*(strct(i-1,j  )%v_old+strct(i  ,j  )%v_old)/2.*dx ! south face
                IF (j==1) THEN
                    strct(i,j)%ASu    =   max( mdot,0.) + mu*2.*dx/dy
                ELSE
                    strct(i,j)%ASu    =   max( mdot,0.) + mu*dx/dy
                END IF
                strct(i,j)%APu    =   strct(i,j)%AEu + &
```

```fortran
                    strct(i,j)%ANu + &
                    strct(i,j)%AWu + &
                    strct(i,j)%ASu
                strct(i,j)%APu      = strct(i,j)%APu/Omega
            END DO
        END DO
        !$OMP END PARALLEL DO

        ! mdot and Av values
        !$OMP PARALLEL DO
        DO i=1,nx
            DO j=1,ny
                mdot            =   rho*(strct(i+1,j-1)%u_old+strct(i+1,j  )%u_old)/2.*dy ! east face
                IF (i==nx) THEN
                    strct(i,j)%AEv      =   max(-mdot,0.) + mu*2.*dy/dx
                ELSE
                    strct(i,j)%AEv      =   max(-mdot,0.) + mu*dy/dx
                END IF
                mdot            =   rho*(strct(i  ,j+1)%v_old+strct(i  ,j  )%v_old)/2.*dx ! north face
                strct(i,j)%ANv      =   max(-mdot,0.) + mu*dx/dy
                mdot            =   rho*(strct(i  ,j-1)%u_old+strct(i  ,j  )%u_old)/2.*dy ! West face
                IF (i==1) THEN
                    strct(i,j)%AWv      =   max( mdot,0.) + mu*2.*dy/dx
                ELSE
                    strct(i,j)%AWv      =   max( mdot,0.) + mu*dy/dx
                END IF
                mdot            =   rho*(strct(i  ,j-1)%v_old+strct(i  ,j  )%v_old)/2.*dx ! south face
                strct(i,j)%ASv      =   max( mdot,0.) + mu*dx/dy
                strct(i,j)%APv      =   strct(i,j)%AEv + &
                    strct(i,j)%ANv + &
                    strct(i,j)%AWv + &
                    strct(i,j)%ASv
                strct(i,j)%APv      = strct(i,j)%APv/Omega
            END DO
        END DO
        !$OMP END PARALLEL DO

        ! solve u-momentum
        error2 = 1.
        DO iter=1,max_iter
            error2=error
            error = 0.
            DO i=2,nx
                DO j=1,ny
                    strct(i,j)%u = (1.-Omega)*strct(i,j)%u_old &
                        + &
                        (1./strct(i,j)%APu) &
                        * (&
                        strct(i  ,j  )%AEu*strct(i+1,j  )%u +   &
                        strct(i  ,j  )%ANu*strct(i  ,j+1)%u +   &
                        strct(i  ,j  )%AWu*strct(i-1,j  )%u +   &
                        strct(i  ,j  )%ASu*strct(i  ,j-1)%u +   &
                        (strct(i-1,j)%P_old-strct(i  ,j)%P_old) &
                        *dy                                     &
                        )
                    error = error + (strct(i,j)%u - strct(i,j)%u_old)**2
                END DO
            END DO
            strct(nx+1,:)%u=strct(nx,:)%u
            error=sqrt(error)
            IF (abs(error - error2)<Convergence) EXIT   ! error stops changing convergence
        END DO
        WRITE(*,*) sum(rho*dy*strct(0,:)%u)/sum(rho*dy*strct(nx+1,:)%u),iter
        WRITE(*,*) iter,abs(error-error2)

```

```fortran
        ! solve v-momentum
        error2 = 1.
        DO iter=1,max_iter
            error2=error
            error = 0.
            DO i=1,nx
                DO j=2,ny
                    strct(i,j)%v = (1.-Omega)*strct(i,j)%v_old &
                        + &
                        (1./strct(i,j)%APv) &
                        * (&
                        strct(i  ,j  )%AEv*strct(i+1,j  )%v +&
                        strct(i  ,j  )%ANv*strct(i  ,j+1)%v +&
                        strct(i  ,j  )%AWv*strct(i-1,j  )%v +&
                        strct(i  ,j  )%ASv*strct(i  ,j-1)%v +&
                        (strct(i,j-1)%P_old-strct(i  ,j)%P_old)  *&
                        dy                                       &
                        )
                    error = error + (strct(i,j)%v - strct(i,j)%v_old)**2
                END DO
            END DO
            error=sqrt(error)
            IF (abs(error - error2)<Convergence) EXIT   ! error stops changing convergence
        END DO
        WRITE(*,*) iter,abs(error-error2)
    END SUBROUTINE mom_uv

    SUBROUTINE vel_correction(strct,dx,dy,nx,ny)
        ! requires uniform grid of dx and dy spacing
        REAL,INTENT(IN)     ::  dx,dy
        INTEGER,INTENT(IN)  ::  nx,ny! size of strct in x and y directions
        TYPE(dat),DIMENSION(0:nx+1,0:ny+1),INTENT(INOUT)::strct ! data contained from 0:nx+1 where cells 0 and nx+1 are boundary nodes (cell volume approaches 0 on boundary nodes)
        INTEGER ::  i,j,iter=0 !loop iterators
        REAL    :: error,error2
        REAL    ::  S_sum
        !$OMP PARALLEL DO
        DO i=1,nx
            DO j=1,ny
                IF (i==nx) THEN
                    strct(i,j)%AEp     =    0.
                ELSE
                    strct(i,j)%AEp     =    rho*dy*dy/strct(i+1,j)%APu
                END IF
                IF (j==ny) THEN
                    strct(i,j)%ANp     =    0.
                ELSE
                    strct(i,j)%ANp     =    rho*dx*dx/strct(i,j+1)%APv
                END IF
                IF (i==1) THEN
                    strct(i,j)%AWp     =    0.
                ELSE
                    strct(i,j)%AWp     =    rho*dy*dy/strct(i,j)%APu
                END IF
                IF (j==1) THEN
                    strct(i,j)%ASp     =    0.
                ELSE
                    strct(i,j)%ASp     =    rho*dx*dx/strct(i,j)%APv
                END IF
                strct(i,j)%APp         =    strct(i,j)%AEp + &
                    strct(i,j)%ANp + &
                    strct(i,j)%AWp + &
                    strct(i,j)%ASp
            END DO
        END DO
```

```fortran
        !$OMP END PARALLEL DO
        error  =1.
        error2 =1.
        DO iter=1,max_iter2
            error2=error
            error=0.
            S_sum = 0.
            !$OMP PARALLEL DO
            DO i=1,nx
                DO j=1,ny
                    strct(i,j)%S = &      ! source terms
                        (rho*strct(i+1,j  )%u-rho*strct(i,j)%u)*dy&
                        + (rho*strct(i  ,j+1)%v-rho*strct(i,j)%v)*dx
                END DO
            END DO
            !$OMP END PARALLEL DO
            DO i=1,nx
                DO j=1,ny
                    strct(i,j)%Pp = strct(i,j)%Pp + (OmegaP/strct(i,j)%APp)&
                        *(&
                        + strct(i,j)%AEp*strct(i+1,j  )%Pp&
                        + strct(i,j)%AWp*strct(i-1,j  )%Pp&
                        + strct(i,j)%ANp*strct(i  ,j+1)%Pp&
                        + strct(i,j)%ASp*strct(i  ,j-1)%Pp&
                        - strct(i,j)%S                     &
                        - strct(i,j)%APp*strct(i  ,j  )%Pp&
                        )
                END DO
            END DO
            !$OMP PARALLEL DO
            DO i=1,nx
                DO j=1,ny
                    strct(i,j)%P=strct(i,j)%P_old+alpha*strct(i,j)%Pp
                END DO
            END DO
            !$OMP END PARALLEL DO
            DO i=1,nx
                DO j=1,ny
                    error = error + (strct(i,j)%P - strct(i,j)%P_old)**2
                    S_sum = S_sum + strct(i,j)%S**2
                    IF (ISNAN(strct(i,j)%Pp)) THEN
                        WRITE(*,*) "error on ",i,j
                        STOP
                    END IF
                END DO
            END DO
            IF (abs(error - error2)<Convergence) THEN   ! error stops changing convergence
                EXIT
            END IF
        END DO
        WRITE(*,*) iter,S_sum,abs(error-error2) ! output iterations along with RSS of source term
        !$OMP PARALLEL DO
        DO i=2,nx
            DO j=1,ny
                strct(i,j)%u=strct(i,j)%u + (strct(i-1,j  )%Pp - strct(i  ,j  )%Pp) *dy/strct(i,j)%APu
            END DO
        END DO
        !$OMP END PARALLEL DO
        !$OMP PARALLEL DO
        DO i=1,nx
            DO j=2,ny
                strct(i,j)%v=strct(i,j)%v + (strct(i  ,j-1)%Pp - strct(i  ,j  )%Pp) *dx/strct(i,j)%APv
            END DO
        END DO
        !$OMP END PARALLEL DO
```

```
    END SUBROUTINE vel_correction

    SUBROUTINE Solve_NS(strct,dx,dy,nx,ny)
        REAL,INTENT(IN)     ::  dx,dy
        INTEGER,INTENT(IN)  ::  nx,ny! size of strct in x and y directions
        TYPE(dat),DIMENSION(0:nx+1,0:ny+1),INTENT(INOUT)::strct ! data contained from 0:nx+1 where cells 0 and
            nx+1 are boundary nodes (cell volume approaches 0 on boundary nodes)
        INTEGER ::  i,j,iter=0!loop iterators
        REAL    ::  error2=1.,error_RSS=0.

        open(unit=8,file="output/iter.txt")
        108 FORMAT(2ES16.7)
        WRITE(8,108) 0.1,1.
        DO iter=1,max_iter3
            ! step 1 solve discretised momentum equations
            CALL mom_uv(strct,dx,dy,nx,ny)

            ! step 2 Solve pressure correction equation
            ! step 3 Correct pressure and velocities
            CALL vel_correction(strct,dx,dy,nx,ny)

            ! step 4 Solve all other discretised transport equations
            ! not implemented

            ! if no convergence, then iterate
            error2 = error_RSS
            error_RSS = 0.
            DO i=1,nx
                DO j=1,ny
                    error_RSS = error_RSS + (strct(i,j)%u-strct(i,j)%u_old)**2
                    error_RSS = error_RSS + (strct(i,j)%v-strct(i,j)%v_old)**2
                    error_RSS = error_RSS + (strct(i,j)%P-strct(i,j)%P_old)**2
                END DO
            END DO
            error_RSS = sqrt(error_RSS)

            ! reset values
            strct%u_old  = strct%u
            strct%v_old  = strct%v
            strct%P_old  = strct%P

            ! if converged then stop
            WRITE(8,108) REAL(iter),abs(error_RSS-error2)
            IF (abs(error_RSS-error2) <= Convergence2) THEN
                WRITE(*,*) "converged on iteration and error big loop = ",iter,abs(error_RSS-error2)
                EXIT
            ELSE
                WRITE(*,*) "iteration and error big loop = ",iter,abs(error_RSS-error2)
            END IF
        END DO
        close(8)
    END SUBROUTINE Solve_NS
END MODULE types
```

A.2 Main program

```
! user defined variables to define finite volume
! x and y direction # of cells
#define max_x   101
#define max_y   21
! x and y number of cells plus 1
#define max_xp   102
#define max_yp   22
! x and y number of cells plus 2 (to account for boundary nodes)
#define max_x2p   103
#define max_y2p   23
```

```
! length of x and y
#define len_x 10.
#define len_y 1.

PROGRAM project3
    USE types !use module defined by types
    IMPLICIT NONE
    ! declare variables
    INTEGER ::  i,j!,iter!,max_x=20,max_y=20
    REAL     ::  dx,dy
    REAL     ::  Lu,Ru,Tu,Bu ! boundary condition u velocity values
    TYPE(dat),DIMENSION(0:max_xp,0:max_yp)::data ! 22 if you count edges (thin cell)
    REAL     ::  TIME1,TIME2  ! for time of computation

    ! set dx and dy and gamma and coefficients (without dividing by delta x between node centers)
    dx=len_x/REAL(max_x)
    dy=len_y/REAL(max_y)
    ! initialize data and x,y for middle values
    CALL set_xy(data,dx,dy,max_xp,max_yp,len_x,len_y)

    !! initialize BC's
    ! BC's
    Lu = 1.
    Ru = 0.
    Tu = 0.
    Bu = 0.
    data%u          = 0.  ! initialize all data
    ! left Boundary
    data(1,:)%u     = Lu
    data(0,:)%u     = Lu
    ! bottom boundary
    data(:,0)%u     = Bu
    ! right boundary
    data(max_xp,:)%u= Ru
    ! top boundary
    data(:,max_yp)%u= Tu

    ! initialize u
    data%u_old   = data%u
    ! initialize v
    data%v_old  = 0.
    data%v      = 0.
    ! initialize P values
    data%Pp=0.
    data%P_old=0.
    data%P     =0.

    ! solving Navier-Stokes 2-D using the staggered grid method
    CALL CPU_TIME(TIME1)
    CALL Solve_NS(data,dx,dy,max_x,max_y)
    CALL CPU_TIME(TIME2)
    WRITE(*,*) "CPU Time = ",TIME2-TIME1

    !output
    ! user will need to specify size of
    open(unit= 9,file="output/x.txt")
    open(unit=10,file="output/y.txt")
    open(unit=11,file="output/xu.txt")
    open(unit=12,file="output/yv.txt")
    open(unit=13,file="output/u.txt")
    open(unit=14,file="output/v.txt")
    open(unit=15,file="output/P.txt")
    open(unit=16,file="output/u_spot.txt")
    open(unit=17,file="output/tau_upper.txt")
    open(unit=18,file="output/tau_lower.txt")
```

```
    open(unit=19,file="output/u_center.txt")
    100 FORMAT (max_x2p ES16.7)
    101 FORMAT (2ES16.7)
    102 FORMAT (max_xp ES16.7)
    WRITE( 9,100) ( data(:,i)%xp,i=0,max_yp )
    WRITE(10,100) ( data(:,i)%yp,i=0,max_yp )
    WRITE(11,100) ( data(:,i)%xu,i=0,max_yp )
    WRITE(12,100) ( data(:,i)%yv,i=0,max_yp )
    WRITE(13,100) ( data(:,i)%u ,i=0,max_yp )
    WRITE(14,100) ( data(:,i)%v ,i=0,max_yp )
    WRITE(15,100) ( data(:,i)%P ,i=0,max_yp )
    DO i=0,max_xp
        !IF (data(i,1)%xu <= 0.51 .AND. data(i,1)%xu>=0.49) THEN
        IF (data(i,1)%xu <= 0.41 .AND. data(i,1)%xu>=0.39) THEN
            DO j=0,max_yp
                WRITE(16,101) data(i,j)%u,data(i,j)%yp
            END DO
        END IF
    END DO
    WRITE(17,102) ( mu*(data(i,max_y)%u-data(i,max_yp)%u)/dy ,i=0,max_x )
    WRITE(18,102) ( mu*(data(i,1     )%u-data(i,0      )%u)/dy ,i=0,max_x )
    DO i=0,max_xp
        !IF (data(i,1)%xu <= 0.51 .AND. data(i,1)%xu>=0.49) THEN
        DO j=0,max_yp
            IF (data(i,j)%yp <= 0.51 .AND. data(i,j)%yp>=0.49) THEN
                WRITE(19,101) data(i,j)%u,data(i,j)%xp
            END IF
        END DO
    END DO
    close(9);close(10);close(11);close(12);close(13);close(14);close(15);close(16);close(17);close(18);close(19)
END PROGRAM project3
```