COMPUTER PROJECT #2

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

ABSTRACT

A Convection problem is calculated in this project. Neglecting the diffusion terms, and setting density to one yields the governing equation of $\frac{\partial}{\partial x}(u\phi) + \frac{\partial}{\partial y}(v\phi) = 0$*. Having the domain of* $0 \leq x \leq 1$ *and* $0 \leq y \leq 1$ *and letting* $u = v = 1$ *we can implement the deferred correction method to solve the solution. The solutions for* $\beta = 0.0, 0.9, 1.0$ *are shown.*

NOMENCLATURE

u Velocity in the x-direction
v Velocity in the y-direction
ϕ General Parameter per unit mass
ϕ^L Lower order approximation of ϕ
ϕ^H Higher order approximation of ϕ
F_i Flux coefficient on the i face, (n, e, s, or w)
a_i Coefficient for final discretized equation on cell center i using lower order approximation
$\tilde{a}_i$ Coefficient for final discretized equation on cell center i using higher order approximation
β Blending factor for deferred correction method

INTRODUCTION

A convection of a step profile in a uniform incompressible flow is considered. This problem neglects diffusion and sets density to one. The governing equation now becomes like Eq. 1.

$$\frac{\partial}{\partial x}(u\phi) + \frac{\partial}{\partial y}(v\phi) = 0 \tag{1}$$

This equation was altered using a numerical method and then applied to a system of equations. The solution to the system of equations yielded the true values of ϕ for the convection problem.

The boundary conditions were provided as shown in Eq. 2. This provided for a more accurate convection problem.

$$\begin{aligned} \phi|_{Left\ Wall} &= 100 \\ \phi|_{Bottom\ Wall} &= 100 \\ \left.\frac{\partial \phi}{\partial x}\right|_{Right\ Wall} &= 0 \\ \left.\frac{\partial \phi}{\partial y}\right|_{Top\ Wall} &= 0 \end{aligned} \tag{2}$$

The numerical method and results are shown in the next sections.

NUMERICAL METHOD

The following steps for followed to achieve a numerical method for this problem.

1. Grid generation
2. Discretization of equations
3. Solution of equations

To generate the grid, and formulation represented by Fig. 1 was generated. This figure shows the point P and the corresponding neighbor cells N,E,S, and W.

To discretize the grid the volume integral of Eq. 1 over the cell volume P. The discretization is shown in Eq. 3.

$$
\begin{aligned}
\frac{\partial}{\partial x}(u\phi)+\frac{\partial}{\partial y}(v\phi) &= 0 \\
\iiint_V \frac{\partial}{\partial x}(u\phi)\ dx\,dy\,dz+\iiint_V \frac{\partial}{\partial y}(v\phi)\ dx\,dy\,dz &= 0 \\
(u\phi A)_e-(u\phi A)_w+(v\phi A)_n-(v\phi A)_s &= 0 \\
\text{Assume: } A_e=A_w=A_n=A_s &= A \\
(u\phi)_e-(u\phi)_w+(v\phi)_n-(v\phi)_s &= 0
\end{aligned} \tag{3}
$$

In order to solve for the face values shown in the equation we needed to incorporate upwinding and central differencing schemes. It was found that upwinding passed all the criteria and gave stable results, but yielded false diffusion results. While central differencing gave semi-stable results but did not pass the transitivenesses criteria. It did not account for the direction of the flow in the numerical analysis.

Both of these solutions could be combined, however, and yield a deferred correction method. These solutions were combined as shown in Eq. 4.

$$
\phi \approx \phi^L+\beta(\phi^H-\phi^L) \tag{4}
$$

Eq. 3 and 4 were combined to yield the final discretized equation in Eq. 5

$$
\begin{aligned}
\phi_P = \frac{a_W\phi_W+a_S\phi_S}{a_P} - \frac{\beta}{a_P}\,[&\tilde{a_P}\phi_P-\tilde{a_E}\phi_E-\tilde{a_W}\phi_W-\tilde{a_N}\phi_N-\tilde{a_S}\phi_S \\
&+a_P\phi_P+a_W\phi_W+a_S\phi_S] \\
\text{Where:}\qquad & \\
a_W &= u \\
a_E &= 0 \\
a_S &= v \\
a_N &= 0 \\
a_P &= a_W+a_S \\
\tilde{a_W} &= \frac{u}{2} \\
\tilde{a_E} &= \frac{-u}{2} \\
\tilde{a_S} &= \frac{v}{2} \\
\tilde{a_N} &= \frac{-v}{2} \\
\tilde{a_P} &= \tilde{a_W}+\tilde{a_E}+\tilde{a_S}+\tilde{a_N}
\end{aligned} \tag{5}
$$

This discretized equation was then applied to the grid to generate a system of equations. It was then iterated over until the solution fully converged. The various solutions are shown below.

RESULTS

The diagonal of the solution ϕ is shown in Fig. 2, 3, 4, 5, and 6.

Fig. 2 shows the solution for total upwinding. This is as if we did not implement any deferred correction method. It can be seen that the upwinding yielding undesired diffusion around the middle section. It was more noticeable for the coarser grids.

Fig. 3 shows the solution for the deferred correction method. It can be seen that this has noticeably less diffusion along the middle line. As we get more refined mesh we find that it gets closer and closer to the exact solution. Thus, the artificial diffusion is minimized by both having a finer mesh, and by increasing the β value.

Fig. 4 shows the solution for the central differencing approximation. We can see that it yields the most accurate results for no artificial diffusion. But it should be remembered that this scheme does not pass all the criteria for convection-diffusion solution. The solution here may not be accurate if diffusion is involved.

Fig. 5 shows the solution for the finest grid for all the values of β. We can easily see how the closer the β value gets to one the closer the solution gets to the exact solution.

Fig. 6 shows the contour plot of the finest grid with a blending factor $\beta = 1.0$.

CONCLUSION

The numerical method for approximating a convection problem has been shown here. The solution of ϕ is shown for various values of β for the deffered correction method. The artificial diffusion presented in the solution can be minimized by refining the grid, and by increasing the β value using the deferred correction method.

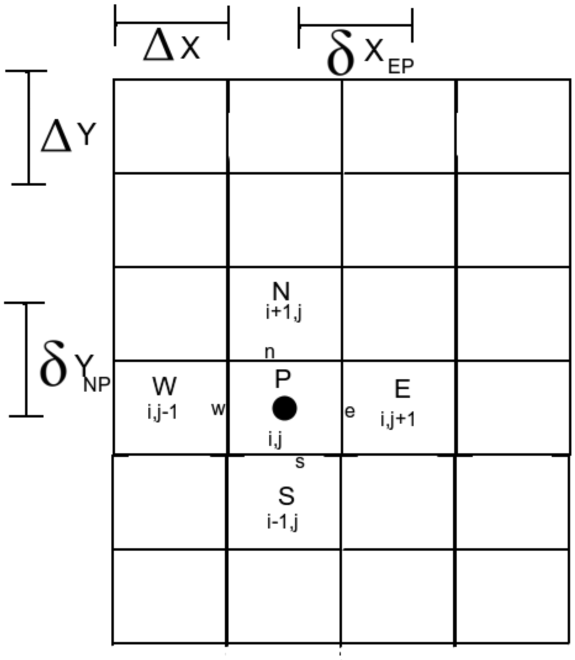

FIGURE 1. REPRESENTATION OF STENCIL FOR GRID GENERATION

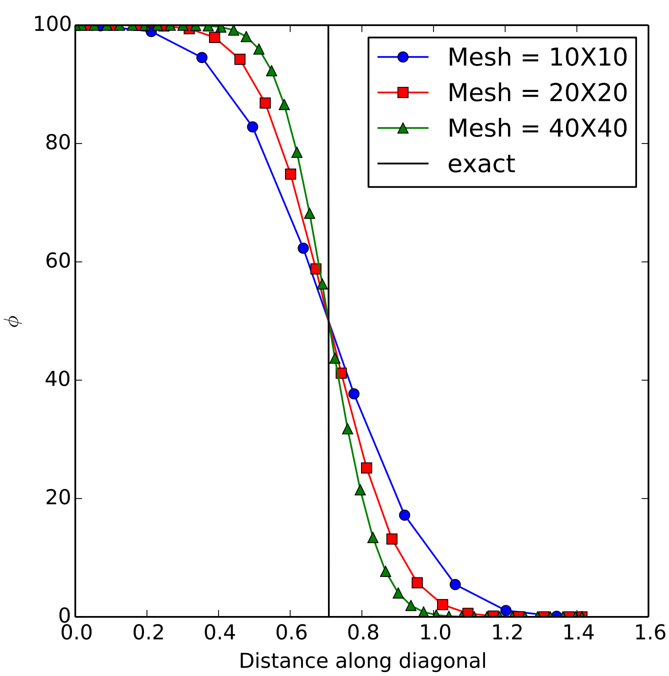

FIGURE 2. DIAGONAL OF $\beta = 0$ UPWINDING

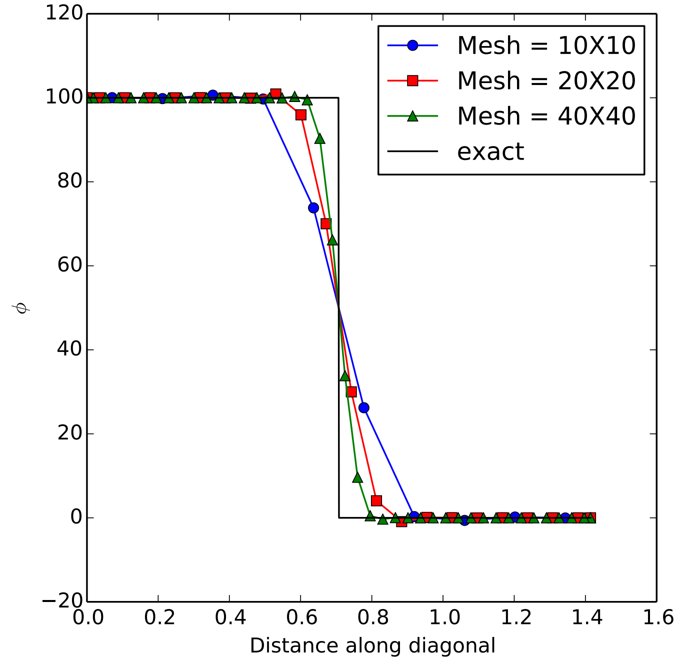

FIGURE 3. DIAGONAL OF $\beta = 0.9$ DEFERRED CORRECTION METHOD

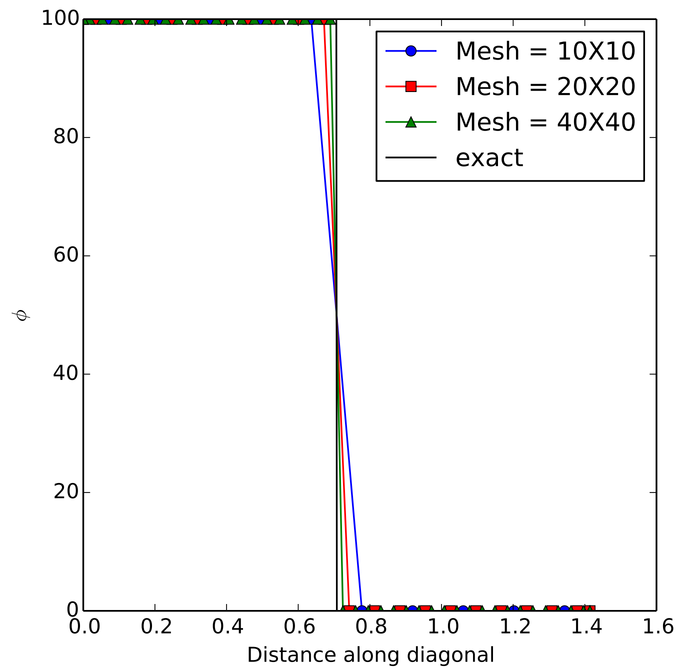

FIGURE 4. DIAGONAL OF $\beta = 1$. CENTRAL DIFFERENCING

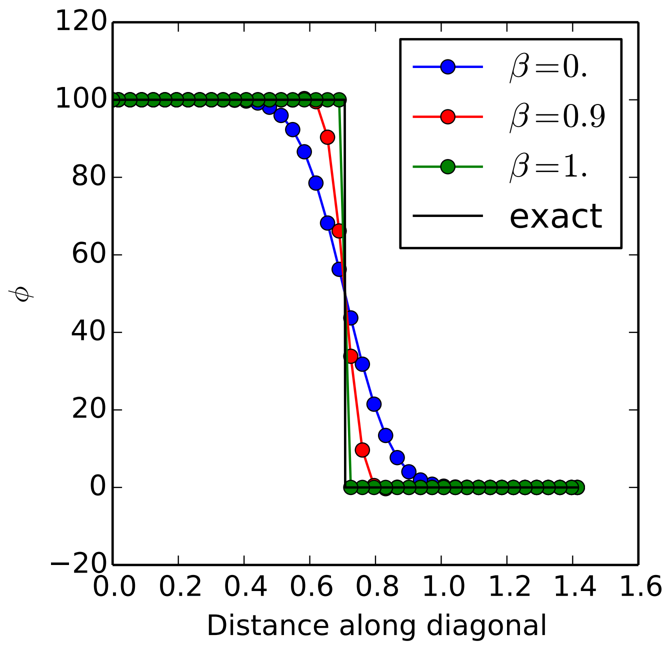

FIGURE 5. DIAGONAL OF 40X40 MESH DENSITY

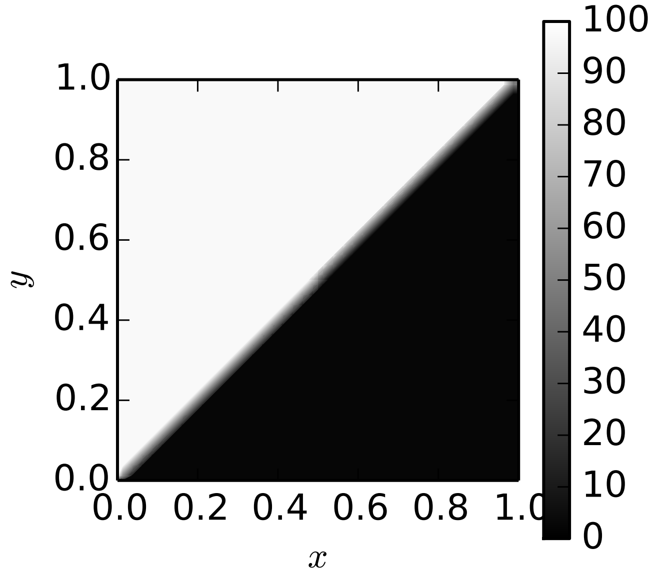

FIGURE 6. CONTOUR PLOT OF PHI FOR MESH 40X40 AND $\beta = 1$.

Appendix A: Code

```fortran
! user defined variables to define finite volume
! x and y direction # of cells
#define max_x 40
#define max_y 40
! x and y number of cells plus 1
#define max_xp 41
#define max_yp 41
! x and y number of cells plus 2 (to account for boundary nodes)
#define max_x2p 42
#define max_y2p 42
MODULE types
    !purpose: define data type struct
    IMPLICIT NONE
    TYPE:: dat
        REAL::x,y,u,v,phi,phi_new
        INTEGER::n
    END TYPE dat
CONTAINS
    SUBROUTINE set_xy (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:nx-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,n   ! for do loops and n is counter for cell number
        REAL    ::  xi,yi   ! x and y values for each cell
        n=1                          ! cell number 1
        DO i=1,ny-2                  ! 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-2
                xi = j*dx - dx/2.        ! x coordinate
                strct(i,j)%n = n         ! input n node
                strct(i,j)%x = xi        ! x coordinate to strct
                strct(i,j)%y = yi        ! y coordinate to strct
                strct(i,j)%phi = 0.      ! phi value initialized guess
                strct(i,j)%phi_new = 0. ! phi value initialized guess for next iteration
                n=n+1                    ! count cell numbers up one
            END DO
        END DO
    ! left boundary
    strct(:,0)%x = 0.
    strct(0,0)%y = 0.
    strct(1:max_x,0)%y = RESHAPE((/ (i*dy - dy/2. ,i=1,max_x) /),(/ max_x /))
    strct(max_xp,0)%y = 1.
    ! top boundary
    strct(0,0)%x = 0.
    strct(0,1:max_y)%x = RESHAPE((/ (i*dx - dx/2. ,i=1,max_y) /),(/ max_y /))
    strct(0,max_yp)%x = 1.
    strct(0,:)%y = 0.
    ! right boundary
    strct(:,max_yp)%x = 1.
    strct(:,max_yp)%y = strct(:,0)%y
    ! bottom boundary
    strct(max_xp,:)%x = strct(0,:)%x
    strct(max_xp,:)%y = 1.
    END SUBROUTINE set_xy
END MODULE types



PROGRAM project2
    USE types !use module defined by types
    IMPLICIT NONE
    ! declare variables
    INTEGER ::  i,j,iter,k!, max_x=20, max_y=20
    REAL    ::  dx,dy,gamma,ae,aw,an,as,ap,error
    REAL    ::  Lphi,Rphi,Tphi,Bphi ! boundary condition phi values
    REAL    ::  Fe,Fw,Fn,Fs ! flux terms = rho * u where rho = 1
    REAL    ::  aet,awt,ast,ant,apt ! higher order interpolation scheme coefficients
    REAL    ::  beta        ! used for combining lower order and higher order interpolation schemes
    REAL    ::  u=2.,v=2.   ! velocity values
    TYPE(dat),DIMENSION(0:max_xp,0:max_yp)::phi ! 22 if you count edges (thin cell)
    REAL    :: TIME1,TIME2 ! for time of computation
    REAL(KIND=4),DIMENSION(2):: TIMEA  ! for time of computation
    ! set dx and dy and gamma and coefficients (without dividing by delta x between node centers)
    dx=1./REAL(max_x)
    dy=1./REAL(max_y)
    gamma = 1.
    ! initialize phi data and x,y for middle values
    CALL set_xy(phi,dx,dy,max_x2p,max_x2p)
    !! initialize BC's
    ! BC's
    Lphi = 100.
    Rphi = 0.
    Tphi = 100.
    Bphi = 0.
    ! left Boundary
    phi(:,0)%phi = Lphi
    phi(:,0)%phi_new = Lphi
    ! bottom boundary
    phi(0,:)%phi = Bphi
    phi(0,:)%phi_new = Bphi
    ! right boundary
    phi(:,max_yp)%phi = Rphi
    phi(:,max_yp)%phi_new = Rphi
    ! top boundary
    phi(max_xp,:)%phi = Tphi
    phi(max_xp,:)%phi_new = Tphi
    ! point SOR method to solve for the exact values of phi using the BC (only loop through inner values)
    ! solving using the deferred correction method
    Fe = 1.*u
    Fw = 1.*u
    Fn = 1.*v
    Fs = 1.*v
    beta = 1.
    ae = 0.
    an = 0.
    aw = Fw
    as = Fs
    ap = as + aw
    aet= -Fe/2.
    awt= Fw/2.
    ant= -Fn/2.
    ast= Fs/2.
    apt= aet + awt + ant + ast
    open(unit=5,file="output/convergence.txt")
    CALL CPU_TIME(TIME1)
    DO iter=0,100000000
        error = 0.
        DO j=1,max_x
            DO i=1,max_y
                phi(i,j)%phi_new = (aw*phi(i,j-1)%phi_new + as*phi(i-1,j)%phi_new)/ap &
                    -beta/ap * (&
                     apt*phi(i  ,j  )%phi_new &
                    -aet*phi(i  ,j+1)%phi_new &
                    -awt*phi(i  ,j-1)%phi_new &
                    -ant*phi(i+1,j  )%phi_new &
                    -ast*phi(i-1,j  )%phi_new &
                    -ap *phi(i  ,j  )%phi_new &
                    +aw *phi(i  ,j-1)%phi_new &
                    +as *phi(i-1,j  )%phi_new )
```

```
                IF (ISNAN(phi(i,j)%phi_new)) THEN
                    WRITE(*,*) "You really stink at this dude! error on ",i,j," value of phi_new. On iter = ",iter
                    WRITE(*,*) phi(i,j)%phi_new
                    STOP
                END IF
                error = error + (phi(i,j)%phi-phi(i,j)%phi_new)**2  ! RSS the error for each iteration
            END DO
        END DO
        ! BC ghost node values
        phi(max_yp,:)%phi_new = phi(max_y,:)%phi_new
        phi(:,max_xp)%phi_new = phi(:,max_x)%phi_new
        error = SQRT(error)                              ! sqrt to have RSS of error
        write(5,*) iter,error
        phi%phi = phi%phi_new                            ! set old phi to the new iteration guess
        IF (error .lt. 1.E-11) THEN
            WRITE(*,*) 'Did in ',iter,' iterations'
            WRITE(*,*) 'With RSS error = ',error
            EXIT                                         ! exit loop if error is small enough
        END IF
    END DO

    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/phi.txt");
    open(unit=13,file="output/line.txt")
    100 FORMAT (max_x2p F14.6)
    WRITE( 9,100) ( phi(i,:)%x  ,i=0,max_xp )
    WRITE(10,100) ( phi(i,:)%y  ,i=0,max_xp )
    WRITE(11,100) ( phi(i,:)%phi  ,i=0,max_xp )
    k=max_xp
    DO i=0,max_yp
        !WRITE(13,'(2F14.6)') REAL(k)*(sqrt(2.))/REAL(max_x), phi(i,k)%phi
        WRITE(13,'(2F14.6)') phi(i,k)%x*sqrt(2.), phi(i,k)%phi
        k=k-1
    END DO
    close(9);close(10);close(11);close(5);close(13)
    WRITE(*,*) "Wall Time = ",etime(TIMEA)
END PROGRAM project2
```