****

**CE-580**

**COMPUTATIONAL TECHNIQUES**

**FOR**

**FLUID DYNAMICS**

**HOMEWORK #2**

**Uniform Flow Between Two**

**Parallel Plates**

**Taha Yaşar Demir**

**1881978**

# CALCULATIONS

***Assumptions:***

* Steady State
* Laminar flow
* Uniform 1-D flow
* Fully developed
* 2-D domain

Conservation of momentum equation in x-direction

1. (2) (3) (4) (5) (6) (7) (8)

Considering steady laminar flow between two parallel plates

* Assuming steady state the term (1) becomes zero
* Fully developed flow means there is no variation of x-velocity along x direction so terms (2) and (6) drops
* Uniform flow, there is no y- and z- component of velocity so terms (3) and (4) drops
* In a 2-D domain the term (8) will be dropped

Momentum equation becomes

After arrangement

***Computational domain***

2-D Two parallel plates with a distance 2H with each other. The Domain is cut in half to reduce computational effort

|  |
| --- |
|  |
| *Figure 1:* Computational Domain |

***Solution Parameters***

***Discretization***

Using second order finite difference, momentum equation

Or

Above equation will result in a three-diagonal system of equations. To solve this equations Thomas algorithm will be used. Represent above equation in the form

Where

To keep the round-off errors down, it is sufficient that

Solution vector can be written in the form of

One index down of equation (2)

Substituting eq(3) for into eq(1) and solving for gives

This gives E and F terms

***Boundary Conditions***

The boundary condition at i=1 will determine E1 and F1, after which the recursion relations (5) and

(6) can be used to calculate all Ei and Fi up to i=N-1. Then uN is set from the boundary condition at

i=N and Eq.2 is used with the known Ei and Fi to solve recursively for ui from ui+1, marching down

from i=N-1 to i=1.

The boundary condition at i=1 is used to determine E1 and F1. Writing Eq.2 at i=1 gives

For a Dirichlet condition (u1 = constant) the above equation must hold for all possible values of u2,

Thus

In this problem we impose no slip condition, which implies B.C.1 is also 0.

The boundary condition at i=N is used to determine uN as follows:

For a Dirichlet condition: in part b) of the problem B.C.N is set to Umax(=5 m/s) which will be calculated in analytical calculation section

For Neumann condition:

In part c) of the problem we impose symmetry line which means zero gradient on centerline, so B.C.N will be 0.

***Analytical Solution***

Analytical velocity profile can be calculated by integrating momentum equation twice

First integration yields

On symmetry line the gradient will be zero, equation above equation zero at y=0.1

Integrate again

And imposing no slip condition at y=0

Final equation

Finally, errors are calculated and normalized as

# Results and Discussion

|  |
| --- |
|  |
| *Figure 2*: Analytical Velocity distribution |

Analytical solution gives typical laminar flow profile with maximum velocity 5 m/s

The main equation that we are solving a second order derivative, it can be approximated exactly if second order discretization is used. Which means we are not introducing any discretization error when using finite difference.

|  |
| --- |
|  |
| Figure 3: Error Distribution with respect to grid points (N) |

Although, we are not introducing any discretization errors, Figure 3 shows that we, actually, do. It can be seen at small iterations; the error is quite large. The reason is that we are, actually, using first order approximation for the Neumann boundary condition, so some discretization error is introduced. This error is reducing with the reduced step size () but around N=1000 round-off errors become dominant and error is increasing again.

In Dirichlet type boundary condition, as seen in figure 3, there is no discretization error. On small iteration numbers (grid point) there is almost no error introduced. The error seen is completely due to round off error and it increases as the number of operation increases.

Notice that the error magnitudes gets closer to each other as N increases, which means discretization error on Neumann b.c. become less and less and finally two methods almost have same error.

|  |
| --- |
|  |
| *Figure 4*: Centerline Velocity Variation with Umax= 5 m/s (For Neumann B.C.) |

Figure 4 confirms the behavior of the imposed Neumann B.C.. Up to N=1000 centerline velocity gets closer to analytical velocity (5 m/s) but after that point round off errors becomes dominant and solution oscillates.

Fortran uses 32 bits to store Real type variables, as seen from the results this may be not sufficient in some cases. To increase precision and get rid of round-off errors, double precision can be used. It gives enough accuracy to eliminate round-off error. As seen from figure 5, double precision eliminates errors and oscillations.

|  |
| --- |
|  |
| *Figure 4*: Centerline Velocity Variation with Umax= 5 m/s (For Neumann B.C.)  For Double Precision |

# Appendix

|  |
| --- |
| c.. Taha Yaşar Demir / 1881978  c.. Ce-580 / Homework #2  program ParallelPlates  parameter(mx=5001)  double precision u(mx), y(mx), A(mx), B(mx), C(mx), D(mx)  double precision E(mx), F(mx)  double precision H, Cp, nu, dy, Umax  double precision BC1, BC2  double precision ua(mx)  double precision ee  c real u(mx), y(mx), A(mx), B(mx), C(mx), D(mx), E(mx), F(mx)  c real H, Cp, nu, dy, Umax  c real BC1, BC2  c real ua(mx)  c real ee  integer N  character boundary  c-----------------------------------------------------------------------  c..Variable names:  c u(x):velocity vevtor , y(x):distance from wall  c A,B,C,D: coefficient vectors  c E,F: coefficient of thomas algorithm  c H: Half of the distance between plates, Umax: Analutical Max velocity  c Cp: Pres. Coef , nu: viscousity ,dy:delta y  c ua: Analytical velocity distribution vector  c N: Toral grid points  c ee: normalized error  c-----------------------------------------------------------------------  c..Create Output Files  open(1,file='Error.dat')  open(2,file='center\_vel.dat')  open(3,file='analytic.dat')  c..Get the centerline boundary information  print\*, "Specify centerline B.C"  print\*, "d => Drichlet, n => Neumann"  read\*, boundary  H = 0.1 !m  Cp = -1. !Pa/m  nu = 1E-3 !N.s/m^2  Umax = 5. !m/s  c..Specify boundary conditions  BC1 = 0.0 ! wall B.C => 0 means no slip cond.  if (boundary.eq."d") then ! initiate boundary conditions  BC2 = Umax ! centerline B.C  elseif(boundary.eq."n") then  BC2 = 0.0 ! centerline B.C => zero means 0 gradient=symmetry  Endif  c..Set first thomas coefficients  E(1) = 0.0  F(1) = BC1  c..Main program loop  do N=10,5000,10  dy = H/(N-1)  do i=2,N-1 ! Coefficient vectors  A(i) = 1.0  B(i) = 2.0  C(i) = 1.0  D(i) =-(Cp\*dy\*\*2)/nu  y(i+1) = y(i)+dy  enddo  call thomas(N,A,B,C,D,E,F) ! Returns Thomas Coefficients  if (boundary.eq."d") then ! set Umax based on B.C  u(N) = BC2  elseif(boundary.eq."n") then  u(N) = (F(N-1)+dy\*BC2)/(1-E(N-1))  endif  do k=N-1,1,-1 ! Get the velocity distribution recursively  u(k) = E(k)\*u(k+1) + F(k) ! Numerical solution  enddo  call Analytic(N,dy,y,ua) ! Get analytic solution  call Error(N,Umax,u,ua,ee) ! Calculate normalized error  call Output(N,ee,u(N),ua,y) ! Output the data  enddo  close(1)  close(2)  close(3)  stop  end |
| Main Fortran Code |

|  |
| --- |
| c..Thomas Algorithm  subroutine thomas(nn,ca,cb,cc,cd,ce,cf)  parameter(mx=1001)  c real ca(mx),cb(mx),cc(mx),cd(mx),ce(mx),cf(mx)  double precision ca(mx),cb(mx),cc(mx),cd(mx),ce(mx),cf(mx)  integer nn  do i=2,nn-1  ce(i) = ca(i)/(cb(i)-cc(i)\*ce(i-1))  cf(i) = (cd(i)+cc(i)\*cf(i-1))/(cb(i)-cc(i)\*ce(i-1))  enddo  return  end |
| Thomas Algorithm to Solve Tridiagonal System |

|  |
| --- |
| c..Analytic Solution  subroutine Analytic(p,deltaY,ydist,uu)  parameter(mx=5001)  c real uu(mx), ydist(mx), deltaY  double precision uu(mx), ydist(mx), deltaY  integer p  ydist(1) = 0.  do i=1,p  uu(i) = -500\*ydist(i)\*\*2 + 100\*ydist(i)  ydist(i+1) = ydist(i) + deltaY  enddo  return  end |
| Calculation of Analytic Solution |

|  |
| --- |
| c..Error Calculation  subroutine Error(gnum,max,nu,ru,err)  parameter(mx=5001)  c real max,nu(mx),ru(mx),err  double precision max,nu(mx),ru(mx),err  integer gnum  err = 0.  do k=2,gnum  err = err + abs(nu(k)-ru(k))  enddo  err = (err)/((gnum-1)\*max)  return  end |
| Error Calculation |

|  |
| --- |
| c..Oupput Data  subroutine Output(grid,e\_r,center\_vel,analytic\_u,yy)  parameter(mx=5001)  c real e\_r,center\_vel,analytic\_u(mx),yy(mx)  double precision e\_r,center\_vel,analytic\_u(mx),yy(mx)  integer grid  write(1,\*) grid, e\_r  write(2,\*) grid, center\_vel  if(grid.eq.100) then  do k=1,grid  write(3,\*) yy(k), analytic\_u(k)  enddo  endif |
| Output Subroutine |