****

**CE-580**

**COMPUTATIONAL TECHNIQUES**

**FOR**

**FLUID DYNAMICS**

**HOMEWORK #4**

**Turbulent Pipe Flow**

**Taha Yaşar Demir**

**1881978**

# Calculations

***Assumptions***

* Steady, Uniform Flow
* Turbulent Flow
* Smooth Pipe
* Axisymmetric Domain

Simplified momentum equation in radial coordinates

***Discretization***

Using FTCS explicit scheme;

The algebraic equation for grid/node is

Where

A recurrence formula can be formed

Where

As the solution converges the difference between time steps will go to zero, in other words, when solution converges. This is how transient(explicit) works. Flow solved is still steady, but the solution is transient until it converges.

For turbulence modelling, mixing length theory is used. Turbulent viscosity is calculated as

Where

Where wall shear is calculated at the middle of first grid point that is in the viscous sublayer. We can use shear stress definition in the viscous sublayer

And effective viscosity would be

The other unknown in the momentum equation is pressure coefficient. Which can be calculated from the wall shear as

***Computational Domain***

|  |
| --- |
|  |
| Figure 1: Computaional Domain |

***Grid***

|  |
| --- |
|  |
| Figure 2: Grid |

Grid is generated with constant ratio method

Ratio between any two neighboring meshes is constant

Distance between the last two mesh point can be calculated as

Mesh distribution can be completed marching down from N to 1

***Boundary Conditions***

Since this is an explicit solution, implementing Dirichlet boundary conditions are easy. It can be done equation the boundary values on the upper and lower limits

***Initial condition***

An initial velocity distribution to start the solution procedure is get from the power law.

***Error***

We can consider the term as error term, using this a general residual error can be calculated

***Outputs***

Required outputs can be calculated once the solution is converged

Discharge

Above integral can be taken numerically as such

Average velocity

Reynolds Number

Friction factors

From experimental data, using Swamee-Jain formula

Since we assumed smooth pipe we can drop term and equation becomes

From the converged solution, using Darcy’s friction factor

Finally, diffusion number is calculated as

***Input Data***

***Program Algorithm***

1. Initialize the data using above data
   1. Calculate Grid
   2. Get initial velocity distribution using power law
2. İmplement boundary conditions
3. Start solution loop
   1. Calculate stresses
      1. Wall shear, Cp, turbulent viscosity, effective viscosity
      2. Shear stresses calculated at midpoints and indexed such as
   2. Calculate Coefficients
   3. Update the Velocity profile
   4. Calculate error
   5. Calculate required output parameters on last iteration
4. End loop after 100000 iterations

# Results and Discussion

|  |
| --- |
|  |
| Figure 3: y+ vs u+ |

Figure 3 shows typical turbulent boundary layer behavior. It has same values with the log layer in the outer layer region and starts to differ in buffer and viscous sublayer. Note that y+ starts at almost 2 which is in the viscous sublayer range. This allows us to calculate wall shear using shear stress definition .

|  |
| --- |
|  |
| Figure 4: Error magnitudes at each iteration |

Since it a transient solution, convergence takes a lot of iteration. Figure 4 illustrates the error distribution at each iteration. It takes almost a hundred thousand iteration for residual error to go below . Also, toward the end of iterations error magnitude starts to fluctuate due to added up round-off errors. One hundred thousand iterations is a sweet spot that gives enough accuracy before solution blows up due to numerical errors.

|  |
| --- |
|  |
| Figure 5: Diffusion number variation w.r.t. y+ |

To ensure stability of the solution we must choose diffusion number less than 0.5. From the definition of diffusion number . Stability condition depends on the mesh size. As seen from figure 5 the biggest diffusion number is at the first grid point where the mesh is most refined. In other words, is smallest. It is the critical point where the biggest times step is to be determined. This also means, we don’t have to use same time step across the solution domain. Bigger time steps could be taken where the is larger and reach convergence faster.

|  |
| --- |
|  |
| Figure 6: Ratio of turbulent viscosity to kinematic viscosity along y |
|  |

Figure 5 shows the distribution of turbulent viscosity. The grid is refined enough that first grid point is in viscous sublayer which means kinematic viscosity is dominant. On the contrary, away from the wall turbulent viscosity becomes dominant quickly. The reason behind this behavior is mixing length theory which gives accurate results when first grid point is in the viscous sublayer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Discharge (m^3/s)** | **Average Velocity (m/s)** | **Reynolds Number** | **Swamee-Jain Friction Factor(fm)** | **Darcy's Friction Factor(fd)** | **Error %** |
| **1.3387E-02** | **1.7039E+00** | **1.7039E+05** | **1.6028E-02** | **1.5966E-02** | **3.9016E-01** |
| Table 1: Calculated Results | | | | | |

Swamee-Jain formula is valid for . With resultant Re we can use this formula. Error between experimental friction factor and calculated one is only 3.9%. Considering experimental data reflects real life results and all the assumptions done, it could be said that mixing length theory gives very accurate results. To improve results, we may go to y+ 1 and perhaps use different turbulence modelling.

# Program Listing

|  |
| --- |
| program TurbulentPipeFlow  c.. Taha Yaşar Demir /1881978  c.. CE-580 / homework #4  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  common/coef/ A(mx), C(mx), eps(mx), dT  common/error/u\_old(mx),rel\_err,iteration  open(11,file='yplus.dat')  open(22,file='error.dat')  open(33,file='difno.dat')  open(44,file='turbv.dat')  open(55,file='loglaw.dat')  iteration = 100000  call init  u(1) = 0.  u(N) = u\_max  do i=1,iteration  call stress  call coefficients  call update  if (i.gt.1) then  call error\_calc(test)  call output(i)  endif  enddo  close(11)  close(22)  close(33)  close(44)  close(55)  stop  end |
| Main Program |

|  |
| --- |
| subroutine init  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  common/coef/ A(mx), C(mx), eps(mx), dT  common/error/u\_old(mx),rel\_err,iteration  u\_max = 2.  vis = 1E-6  Beta = 0.82  rho = 1000.  Rad = 0.05  Beta = 0.82  N = mx  dT = 3.5E-4  call makegrid  call analytic  return  end |
| Solution Initializaiton |

|  |
| --- |
| subroutine makegrid  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  common/coef/ A(mx), C(mx), eps(mx), dT  common/error/u\_old(mx),rel\_err,iteration  real sum  sum = 0.0  do i=0,N-2  sum = sum + Beta\*\*i  enddo  dy(N) = Rad/sum  y(N) = Rad  do i=N,2,-1  y(i-1) = y(i)-dy(i)  dy(i-1) = Beta\*dy(i)  yc(i) = (y(i)+y(i-1))/2.  r(i) = Rad - y(i)  rc(i) = Rad - yc(i)  enddo  r(1) = Rad  y(1) = 0.  return  end |
| Generation of Grid |

|  |
| --- |
| subroutine analytic  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  do i=2,N-1  u(i) = u\_max\*(y(i)/Rad)\*\*(1./7.)  enddo  return  end |
| Analytic Solution |

|  |
| --- |
| subroutine stress  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  tau = rho\*vis\*(u(2)-u(1))/dy(2)  us = sqrt(tau/rho)  Ap = 26.  do i=2,N  yp(i) = yc(i)\*us/vis  fu(i) = 1-exp(-yp(i)/Ap)  sml(i) = Rad\*(0.14-0.08\*(1-(yc(i)/Rad))\*\*2  + -0.06\*(1-(yc(i)/Rad))\*\*4)\*fu(i)  vist(i)= (sml(i)\*\*2.)\*((u(i)-u(i-1))/dy(i))  vise(i)= vist(i) + vis  enddo  Cp =-2.\*(tau/Rad)  return  end |
| Stress Calculation |

|  |
| --- |
| subroutine coefficients  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  common/coef/ A(mx), C(mx), eps(mx), dT  real term  do i=2,N-1  term = (dy(i) + dy(i+1))  A(i) = ( 2.\*rc(i+1)) / (r(i)\*term\*dy(i+1) )  C(i) = ( 2.\*rc(i)) / (r(i)\*term\*dy(i) )  eps(i) = dT\*( -(Cp/rho) + A(i)\*vise(i+1)\*(u(i+1)-u(i))  + -C(i)\*vise(i)\*(u(i)-u(i-1)) )  enddo  return  end |
| Calculation of Coefficients |

|  |
| --- |
| subroutine update  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/coef/ A(mx), C(mx), eps(mx), dT  do i=2,N-1  u(i) = u(i) + eps(i)  enddo  return  end |
| Velocity Profile Update |

|  |
| --- |
| subroutine error\_calc  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/coef/ A(mx), C(mx), eps(mx), dT  common/error/u\_old(mx),rel\_err,iteration  real ttest  rel\_err = 0.0  do i=2,N-1  rel\_err = rel\_err + ((1./(N-2)/u\_max)\*abs(eps(i)))  enddo  ttest = rel\_err  return  end |
| Relative Error Calculaiton |

|  |
| --- |
| subroutine output(iter)  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  common/coef/ A(mx), C(mx), eps(mx), dT  common/error/u\_old(mx),rel\_err,iteration  real U\_p(mx),law(mx),Dis,V\_ave,Re,fm,fd  integer iter  write(22,\*) iter,rel\_err  if (iter.eq.iteration) then  do k=2,N  yp(k) = y(k)\*us/vis  U\_p(k) = u(k)/us  write(11,\*) yp(k),U\_p(k)  law(k) = (1./0.41)\*log(yp(k))+5.1  write(55,\*) yp(k),law(k)  if (k.lt.N) then  df(k) = (0.5\*(vise(k)+vise(k+1))\*dT)/(dy(k)\*\*2)  write(33,\*) yp(k),df(k)  endif  write(44,\*) vist(k)/vis, y(k)/Rad  enddo  call output\_par(Dis,V\_ave,Re,fm,fd)  print\*, "Discharge:Average Vel:Reynolds:fm:fd"  print\*, Dis,V\_ave,Re,fm,fd  endif  return  end |
| Outpur the Paramaters |

|  |
| --- |
| subroutine output\_par(discha,ave,reynolds,fr\_m,fr\_d)  parameter (mx=31)  common/flow/ rho,Rad,Cp,vis,vist(mx),vise(mx),u(mx),u\_max  common/grid/ Beta, y(mx),yc(mx),dy(mx),r(mx),rc(mx),N,df(mx)  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, tau  common/coef/ A(mx), C(mx), eps(mx), dT  common/error/u\_old(mx),rel\_err,iteration  real discha,ave,reynolds,fr\_m,fr\_d,pi  discha = 0.  pi = 22./7.  do i=2,N  discha = discha + 2.\*pi\*rc(i)\*((u(i)+u(i-1))/2.)\*dy(i)  enddo  ave = discha/(pi\*Rad\*\*2)  Reynolds = 2.\*ave\*Rad/vis  fr\_m= 0.25/((log10(5.74/(Reynolds\*\*0.9)))\*\*2)  fr\_d= (8.\*tau)/(rho\*ave\*\*2)  return  end |
| Calculating Output Parameters |