****

**CE-580**

**COMPUTATIONAL TECHNIQUES**

**FOR**

**FLUID DYNAMICS**

**HOMEWORK #3**

**Turbulent Flow Between**

**Parallel Plates**

**Taha Yaşar Demir**

**1881978**

# Calculations

***Assumptions***

* Steady flow
* Incompressible flow
* Fully developed turbulent flow
* 1-D uniform flow

***RANS equation in x- direction***

|  |
| --- |
|  |
| Figure 1: Problem Domain |

1. (2) (3) (4) (5) (6) (7) (8) (9) (10) (11)

Simplify above equation according to our domain and assumptions

* Term (1) drops due to steady flow assumption
* Fully developed so there are no u gradients in x- direction term (2), (3),(9) drops
* Uniform flow, term (4) drops
* 2-D domain term (6),(7),(11) drops

Simplified equation is

Partial derivatives can be replaced by total derivatives since each dependent variable is dependent one space parameter

Using Boussinesq hypothesis for turbulent stress

Combining the viscous and turbulence resistance terms

Introducing for effective viscosity and Cp for pressure gradient

Direct integration of above term is not possible due to nonlinearity introduced with viscosity term

***Grid***

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

Grid is generated using constant ratio method

Ratio between any two neighbouring meshes is constan

Distance between the last two mesh point can be calculated as

Mesh distribution can be completed marching down from N to 1

***Discretization***

Governing equation can be discretized on domain as

Note = half term convention is i+1/2 => i+1 , i-1/2 => i

Collecting the coefficients of the unknown velocity terms

We can write above equation such as

***Turbulence Model***

To obtain turbulent viscosity can be obtained from mixing length theory

Where

can be calculated by integrating the momentum equation

Applying the boundary condition at y=H, calculated as

***Boundary Conditions***

A different approach from previous homework is taken to impose boundary conditions.

Consider below equation

At I=2, impose no slip boundary condition

at the wall, so first term drops

At i=N-1 , impose symmetry line, meaning . This makes turbulent velocity zero and cancels the coefficients in B term

When three diagonal system of equations are solved with modified coefficients at I=2 and I=N-1, boundary conditions will be imposed automatically.

***Error Calculations***

Errors are calculated at each iterations as

***Solution Parameters***

***Solution algorithm***

1. Initialize solution
   1. Generate grid
   2. Calculate laminar boundary layer
2. Start iteration 1 to 100
   1. Calculate turbulent stresses
      1. From laminar profile at first iteration
      2. From previous solution as the solution advances
   2. Calculate coefficients of discretized equation with boundary conditions
   3. Call Thomas algorithm to get a solution from three diagonal system of equations
   4. Extract the solution from Thomas subroutine
   5. Calculate error after the first iteration using new and old values of velocity distributions
   6. Output error at every iteration and final velocity profile
3. Stop iterations
4. Print out Cp and maximum velocity

# Results and Discussion

|  |
| --- |
|  |
| Figure 3: Velocity Profiles |

|  |  |
| --- | --- |
| Cp | U\_max (m/s) at centerline |
| -100 | 1.004 |
| -1000 | 3.553 |
| -10000 | 12.453 |

As seen from figure 3 velocity profiles are very steep and contains very high gradients near the wall zone. The high gradients near wall is an effect of viscous damping function. To catch the high gradients accurately we need very dense mesh close to wall regions. The velocity profile flattens away from the wall and gradients can be calculated with bigger steps. This is the logic behind the constant ratio mesh sizing.

|  |
| --- |
|  |
| Figure 4: u+ vs y+ |

Figure 4 is another way to view velocity profile using dimensionless quantities. Where

Log-Law also called Law of the Wall is calculated

The variation of u+ and y+ fits very well the log-law starting from y+=̃30 , Meaning above equation can be used to calculate velocities without finite differences up to the so called outer layer. But as y+ gets smaller there is buffer and viscous sublayer where this correlation does not hold. Thus we need very refined mesh to catch the solution in viscous sublayer. This requires first grid point is at where y+ < 5 , ideally at y+= 1. In our cases , grid is well refined at the close wall regions and viscous sublayer is well solved

|  |
| --- |
|  |
| Figure 5: Turbulent Viscosity Distribution with Different Cp |

Considering velocity gradients close to the wall zone, turbulent viscosities are expected to be large close to wall. But it is not the case as figure 5 shows. The reason of the small turbulent viscosities is the turbulence model. Viscous damping function and mixing length makes necessary corrections and region close to the wall turbulent stresses are damped. As going away from the wall , velocity gradients become smaller but gets into account so that we get such a turbulent viscosity distribution.

|  |
| --- |
|  |
| Figure 6: Error Variation Along Iteration Steps |

A laminar velocity profile is used for starting point of iterations. But in laminar flow the velocity gradients in y- direction is mush greater than a turbulent flow. The result is very high turbulent stresses to start with. High stresses cause almost constant and small velocities and next turbulent stresses will be very low. This will cause oscillations and convergence will be slow.To overcome this problem at first iteration get half of the laminar gradients to start with. And take average of turbulent viscous stresses with the old ones.

Thanks to averaging error magnitude drops to 1E-6 levels after 10 iteration as seen in figure 6. As solution marches there is fluctuations in error magnitude caused by round of and truncation errors.

# Fortran Code

|  |
| --- |
| program TurbulentBetweenParallelPlates  c Taha Yaşar Demir / 1881978  c CE-580 - Homework #3  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  common/coef/ A(mx), B(mx), C(mx), D(mx)  common/error/u\_old(mx),error(mx)  c sml = mixinglength , fu = viscous damping function, yp=y+, Ap=A+  c us = u\_star, rnu = kinematic viscosity, tau = wall shear  open(1,file='error.dat')  open(2,file='yplus.dat')  open(3,file='velpr.dat')  open(4,file='loglaw.dat')  open(6,file='vist.dat')  call init  do l=1,100  call stress(l) ! Evaluate stresses  call coefficients ! Get Three diagonal system coefficents  call THOMAS(2,N-1,A,B,C,D) ! Returns the solution in D vector  u(1) = 0. ! no-slip Boundary Condition  u(N) = D(N-1) ! Neumann Boundary at centerline  do k = 2,N-1 ! extract the solution  u(k) = D(k)  enddo  if (i.gt.1) call error\_cal(l) ! After first iteration calculate error  do k=1,N  u\_old(k) = u(k)  enddo  call output(l) ! write solution out in relative files  enddo  print\*, Cp, u(N) ! get max velocity at centerline  close(1)  close(2)  close(3)  close(4)  close(6)  stop  end |
| Main Program |

|  |
| --- |
| subroutine init  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  common/coef/ A(mx), B(mx), C(mx), D(mx)  H = 0.02 ! m  vis = 0.001 ! N.s/m^2  rho = 1000.0 ! kg/m^3  N = mx  print\*, "Enter the pressure coef. Cp :"  read(\*,\*) Cp  if (Cp.eq.-100.) then  Beta = 0.96  elseif(Cp.eq.-1000.) then  Beta = 0.94  else  Beta = 0.93  endif  call makegrid  call analytic  return  end |
| Initialize Solution Parameters and Get Grid and Laminar Solution |

|  |
| --- |
| subroutine makegrid  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  common/coef/ A(mx), B(mx), C(mx), D(mx)  real sum  sum = 0.0  do i=0,N-2  sum = sum+ Beta\*\*i  enddo  dy(N) = H/sum  y(N) = H  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  enddo  return  end |
| Calculate Gird Spacing and Coordinates |

|  |
| --- |
| subroutine analytic  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  do i=1,N  ua(i) = -500\*y(i)\*\*2 + 200\*y(i)  enddo  ua(1) = 0.  return  end |
| Laminar Solution |

|  |
| --- |
| subroutine stress(iter)  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  Ap = 26  rnu = vis/rho  tau = -Cp\*H  us = sqrt(tau/rho)  do k=2,N  yp(k) = yc(k)\*us/rnu  fu(k) = 1-exp(-yp(k)/Ap)  sml(k) = H\*(0.14-0.08\*(1-(yc(k)/H))\*\*2  & -0.06\*(1-(yc(k)/H))\*\*4)\*fu(k)  if (iter.eq.1) then  vist(k) = 0.5\*(rho\*sml(k)\*\*2)\*(ua(k)-ua(k-1))/dy(k)  else  vist(k) = 0.5\*(vist(k)+(rho\*sml(k)\*\*2)\*(u(k)-u(k-1))/dy(k))  endif  vise(k) = (vist(k)+vis)  enddo  return  end |
| Effective Stress Calculation |

|  |
| --- |
| subroutine coefficients  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  common/coef/ A(mx), B(mx), C(mx), D(mx)  do i=2,N-1  if (i.eq.2) then ! no slip Boundary Conditions  A(i) = 0  C(i) = -vise(i+1)/(dy(i+1)\*(dy(i+1)+dy(i))/2)  B(i) = (vise(i)/(dy(i)\*(dy(i+1)+dy(i))/2)) - C(i)  D(i) = -Cp  elseif (i.eq.N-1) then ! Neumann Boundary Condition u\_n=u\_n-1  A(i) = -vise(i)/(dy(i)\*(dy(i+1)+dy(i))/2)  C(i) = 0  B(i) = -A(i)  D(i) = -Cp  else  A(i) = -vise(i)/(dy(i)\*(dy(i+1)+dy(i))/2)  C(i) = -vise(i+1)/(dy(i+1)\*(dy(i+1)+dy(i))/2)  B(i) = -A(i) - C(i)  D(i) = -Cp  endif  enddo  return  end |
| 3-Diagonal System coefficients |

|  |
| --- |
| subroutine error\_cal(ite)  parameter (mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  common/coef/ A(mx), B(mx), C(mx), D(mx)  common/error/u\_old(mx),error(mx)  error(ite) = 0.  do i=1,N  error(ite) = error(ite) + (1/((N-1)\*u(N)))\*abs(u\_old(i)-u(i))  enddo  return  end |
| Error Calculation |

|  |
| --- |
| subroutine output(it)  parameter(mx=101)  common/flow/ rho, H, Cp, vis, vist(mx), vise(mx), u(mx), ua(mx)  common/grid/ Beta, y(mx),yc(mx),dy(mx),N  common/turb/ sml(mx), fu(mx), yp(mx), Ap, us, rnu, tau  common/error/u\_old(mx),error(mx)  real law(mx),up(mx)  write(1,\*) it,error(it)  if(it.eq.1) write(3,\*) u(1),y(1),ua(1)  if(it.eq.100) then  do i=2,N  law(i) = (1/0.41)\*log(yp(i))+5.1  up(i) = 0.5\*(u(i)+u(i-1))/us  write(2,\*) yp(i),up(i)  write(3,\*) u(i),y(i),ua(i)  write(4,\*) yp(i),law(i)  write(6,\*) vist(i),y(i)  enddo  endif  return  end |
| Write out the Solution |

|  |
| --- |
| subroutine THOMAS(il,iu,aa,bb,cc,ff)  c............................................................  c Solution of a tridiagonal system of n equations of the form  c A(i)\*x(i-1) + Bb(i)\*x(i) + C(i)\*x(i+1) = R(i) for i=il,iu  c the solution X(i) is stored in F(i)  c A(il-1) and C(iu+1) are not used  c A,Bb,C,R are arrays to bbe provided bby the user  c............................................................  parameter (mx=101)  dimension aa(mx),bb(mx),cc(mx),ff(mx),tmp(mx)  tmp(il)=cc(il)/bb(il)  ff(il)=ff(il)/bb(il)  ilp1 = il+1  do i=ilp1,iu  z=1./(bb(i)-aa(i)\*tmp(i-1))  tmp(i)=cc(i)\*Z  ff(i)=(ff(i)-aa(i)\*ff(i-1))\*z  enddo  iupil=iu+il  do ii=ilp1,iu  i=iupil-ii  ff(i)=ff(i)-tmp(i)\*ff(i+1)  enddo  return  end |
| Thomas Algorithm |