Computing assignment:
AEM-ADV19 Computational Fluid Dynamics

Student name	Wenqi Wu	Student CID	01540965
	Wenze Liu Chen		01526073

1. The figures below show the simulation result of 2D heat exchanger which was based on circular cylinders using a second-order Adams-Bashforth time scheme with CFL = 0.25 and Re = 0.25. Different colour represents different vorticity intensity, the green colour implies null vorticity, red and blue colour represent positive and negative vorticity respectively. The area above the cylinder tends to have negative vorticity and the area below tends to have negative vorticity. Since a periodic boundary condition has been imposed on the boundaries, it can be seen from the figures, the vorticity field has a repeating pattern around nine cylinders.

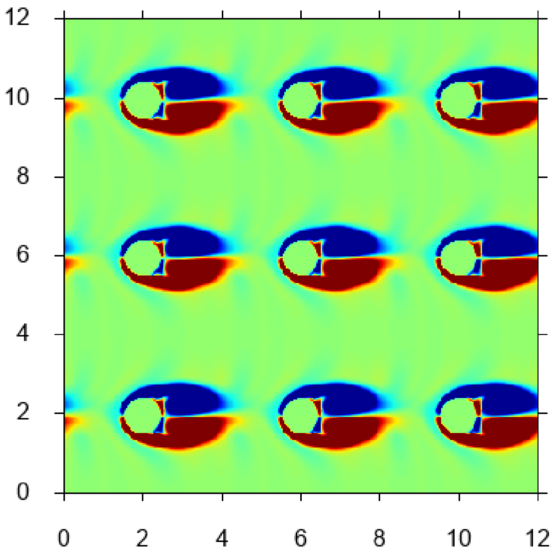

Figure 1. Vorticity filed at nt=2500.

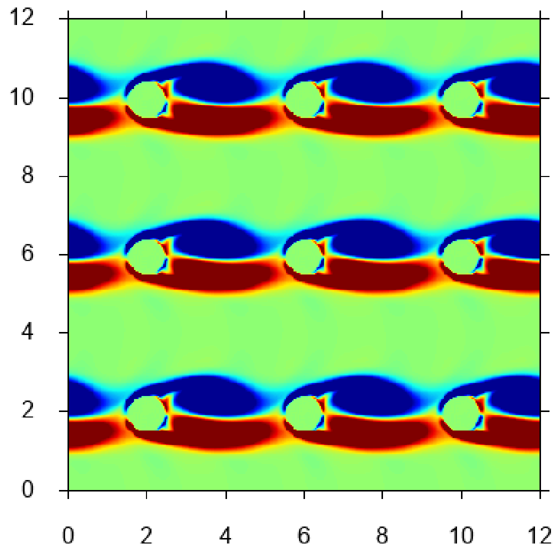

Figure 2. Vorticity filed at nt=5000.

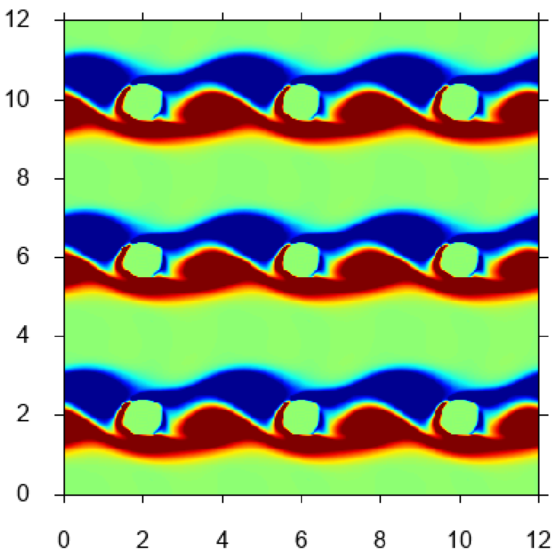

Figure 3. Vorticity filed at nt=7500.

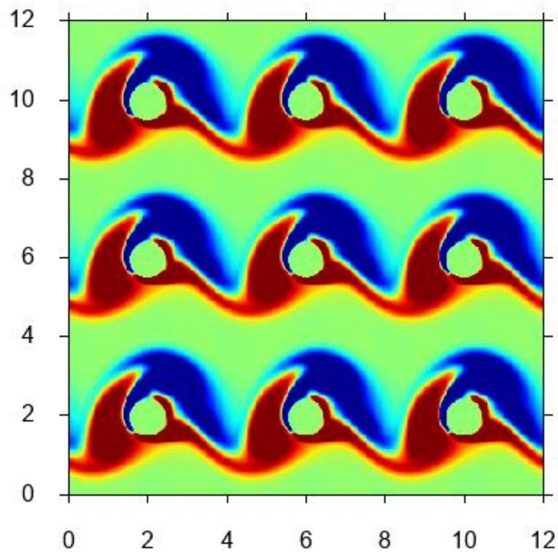

Figure 4. Vorticity filed at nt=10000.

2.In this question, CFL number was changed from 0.25 to 0.75, as a result, the simulation was not able to run. It gave a non-convergent solution. Simulation failed because of the use of inappropriate CFL number. Increasing the CLF number exceeds the stable limit of CFL condition for the Adams-Bashforth time scheme. When CFL number is 0.75, the scheme became unstable since the domain of dependence of physical solution was not contained in the domain of dependence of numerical scheme. CFL condition is a necessary condition to produce a convergent solution according to Lax Equivalent Theorem.

3. Simulation was run using 3rd Runge-Kutta time scheme with a CFL number equals to 0.75, the results are shown in figures below. By using Runge-Kutta scheme, simulation is able to produce a stable and convergent result with higher CFL number. In RK scheme, the numerical solution updated in one time step, which has three sub-steps, is equivalent to the solution update in three time steps in AB scheme. This is demonstrated by comparing the results from two different schemes, the results in RK scheme at nt=2500 is equal to those in AB scheme at nt=7500.

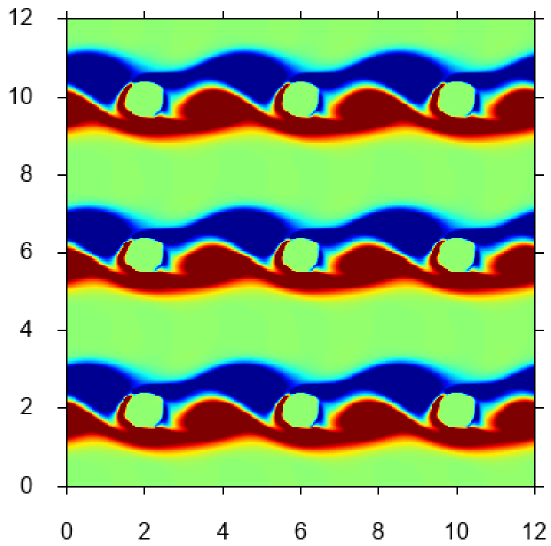

Figure 5. Vorticity filed at nt=2500.

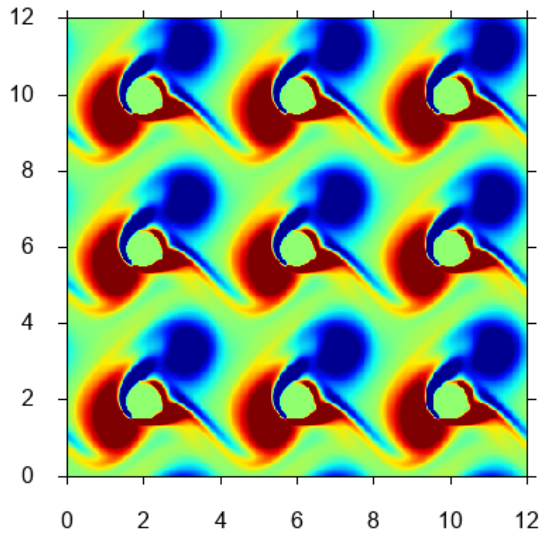

Figure 6. Vorticity filed at nt=5000.

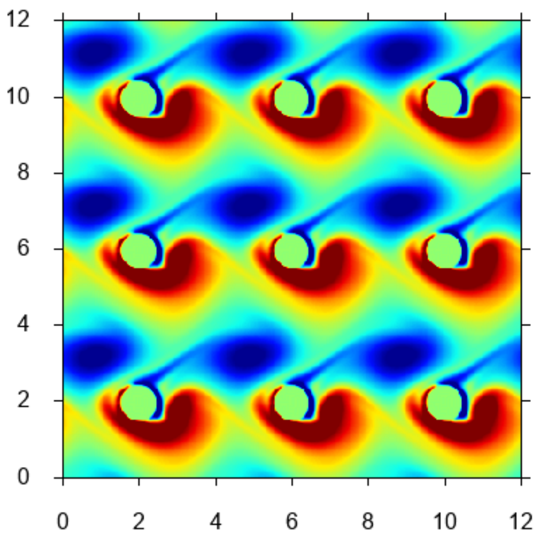

Figure 7. Vorticity filed at nt=7500.

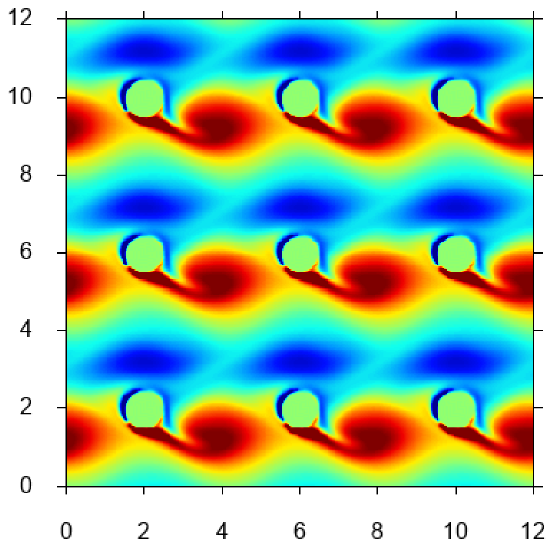

Figure 8. Vorticity filed at nt=10000.

```
!#######################################################################
!
subroutine rkutta(rho,rou,rov,roe,fro,gro,fru,gru,frv,grv,&
     fre,gre,ftp,gtp,nx,ny,ns,dlt,coef,scp,k)
!
!#######################################################################

  implicit none
!
  real(8),dimension(nx,ny) :: rho,rou,rov,roe,fro,gro,fru,gru,frv
  real(8),dimension(nx,ny) :: grv,fre,gre,scp,ftp,gtp
  real(8),dimension(2,ns) :: coef
  real(8) :: dlt
  integer :: i,j,nx,ny,ns,k
!
!coefficient for RK sub-time steps
       coef(1,1)=(8./15.)*(dlt)
       coef(1,2)=(5./12.)*(dlt)
       coef(1,3)=(3./4.)*(dlt)
       coef(2,1)=0.
       coef(2,2)=(-17./60.)*(dlt)
       coef(2,3)=(-5./12.)*(dlt)

  do j=1,ny
    do i=1,nx
       rho(i,j)=rho(i,j)+coef(1,k)*fro(i,j)+coef(2,k)*gro(i,j)
       gro(i,j)=fro(i,j)

       rou(i,j)=rou(i,j)+coef(1,k)*fru(i,j)+coef(2,k)*gru(i,j)
       gru(i,j)=fru(i,j)

       rov(i,j)=rov(i,j)+coef(1,k)*frv(i,j)+coef(2,k)*grv(i,j)
       grv(i,j)=frv(i,j)

       roe(i,j)=roe(i,j)+coef(1,k)*fre(i,j)+coef(2,k)*gre(i,j)
       gre(i,j)=fre(i,j)

       scp(i,j)=scp(i,j)+coef(1,k)*ftp(i,j)+coef(2,k)*gtp(i,j)
       gtp(i,j)=ftp(i,j)
    enddo
  enddo

  return
end subroutine rkutta
!#######################################################################
```

4. Fourth-order centred schemes for the first and second order derivatives in two spatial directions were coded in the simulation. The results are shown in the figures below. Higher order derivatives schemes give more accurate results. However, by comparing the results of 4th order scheme with the results of 2nd order scheme, no significant difference was found between them. The main reason for this is that the mesh size was not changed, when changing to higher order shemes, finer mesh should be used in order to unleah the effect of higher order schemes.

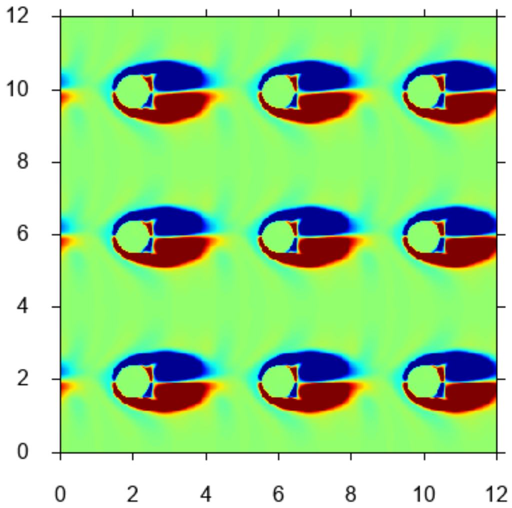

Figure 9. Vorticity filed at nt=2500 (fourth order).

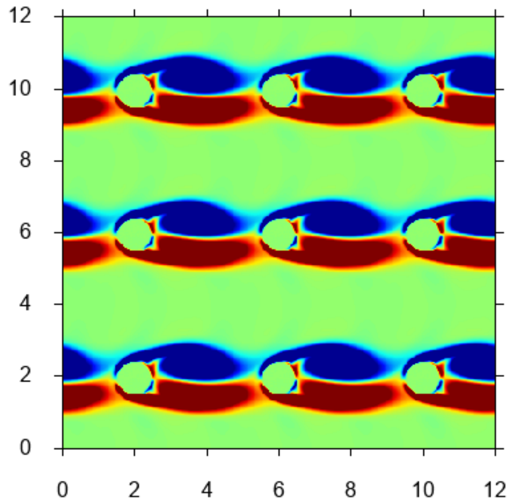

Figure 10. Vorticity filed at nt=5000 (fourth order).

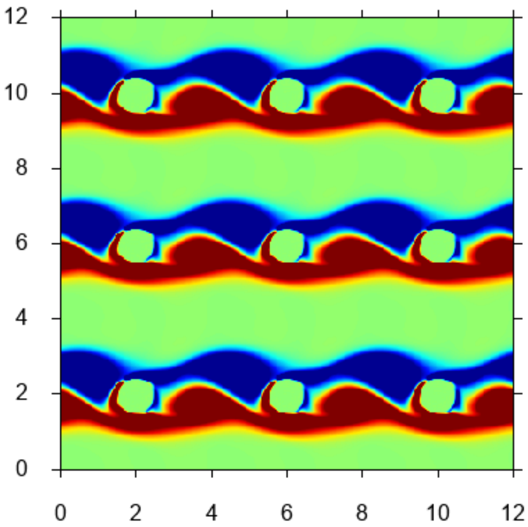

Figure 11. Vorticity filed at nt=7500 (fourth order).

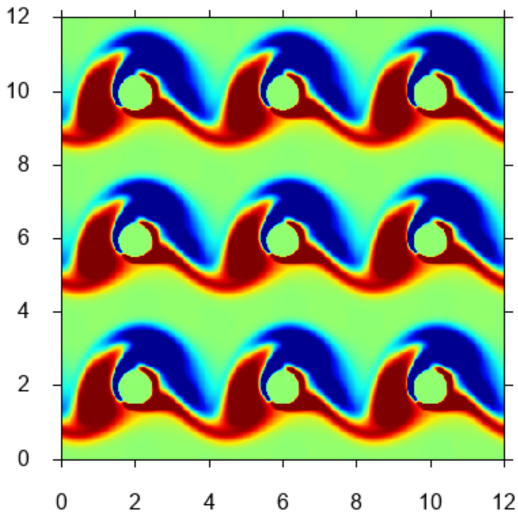

Figure 12. Vorticity filed at nt=10000 (fourth order).

```
!###################################################
!
subroutine derix4(phi,nx,ny,dfi,xlx)
!
!Fourth-order first derivative in the x direction
!###################################################

  implicit none

  real(8),dimension(nx,ny) :: phi,dfi
  real(8) :: dlx,xlx,udx
  integer :: i,j,nx,ny

  dlx=xlx/nx
  udx=1./(12.*dlx)
  do j=1,ny
  dfi(1,j)=udx*(-phi(3,j)+8.*phi(2,j)-8.*phi(nx,j)+phi(nx-1,j))
  dfi(2,j)=udx*(-phi(4,j)+8.*phi(3,j)-8.*phi(1,j)+phi(nx,j))
  do i=3,nx-2
  dfi(i,j)=udx*(-phi(i+2,j)+8.*phi(i+1,j)-8.*phi(i-1,j)+phi(i-2,j))
  enddo
  dfi(nx,j)=udx*(-phi(2,j)+8.*phi(1,j)-8.*phi(nx-1,j)+phi(nx-2,j))
  dfi(nx-1,j)=udx*(-phi(1,j)+8.*phi(nx,j)-8.*phi(nx-2,j)+phi(nx-3,j))
  enddo

  return
end subroutine derix4
!###################################################

!###################################################
!
subroutine deriy4(phi,nx,ny,dfi,yly)
!
!Fourth-order first derivative in the y direction
!###################################################

  implicit none

  real(8),dimension(nx,ny) ::  phi,dfi
  real(8) :: dly,yly,udy
  integer :: i,j,nx,ny

  dly=yly/ny
  udy=1./(12.*dly)
  do j=3,ny-2
  do i=1,nx
  dfi(i,j)=udy*(-phi(i,j+2)+8.*phi(i,j+1)-8.*phi(i,j-1)+phi(i,j-2))
  enddo
  enddo
  do i=1,nx
  dfi(i,1)=udy*(-phi(i,3)+8.*phi(i,2)-8.*phi(i,ny)+phi(i,nx-1))
  dfi(i,2)=udy*(-phi(i,4)+8.*phi(i,3)-8.*phi(i,1)+phi(i,ny))
  dfi(i,ny)=udy*(-phi(i,2)+8.*phi(i,1)-8.*phi(i,ny-1)+phi(i,ny-2))
  dfi(i,ny-1)=udy*(-phi(i,1)+8.*phi(i,ny)-8.*phi(i,ny-2)+phi(i,ny-3))
  enddo

  return
end subroutine deriy4
```

```
!###################################################

!###################################################
!
subroutine derxx4(phi,nx,ny,dfi,xlx)
!
!Fourth-order second derivative in y direction
!###################################################

  implicit none
  real(8),dimension(nx,ny) ::  phi,dfi
  real(8) :: dlx,xlx,udx
  integer :: i,j,nx,ny

  dlx=xlx/nx
  udx=1./(12.*dlx*dlx)
  do j=1,ny
  dfi(1,j)=udx*(-phi(3,j)+16.*phi(2,j)-30.*phi(1,j)+16.*phi(nx,j)-phi(nx-1,j))
  dfi(2,j)=udx*(-phi(4,j)+16.*phi(3,j)-30.*phi(2,j)+16.*phi(1,j)-phi(nx,j))
  do i=3,nx-2
  dfi(i,j)=udx*(-phi(i+2,j)+16.*phi(i+1,j)-30.*phi(i,j)+16.*phi(i-1,j)-phi(i-2,j))
  enddo
  dfi(nx,j)=udx*(-phi(2,j)+16.*phi(1,j)-30.*phi(nx,j)+16.*phi(nx-1,j)-phi(nx-2,j))
  dfi(nx-1,j)=udx*(-phi(1,j)+16.*phi(nx,j)-30.*phi(nx-1,j)+16.*phi(nx-2,j)-phi(nx-3,j))
  enddo

  return
end subroutine derxx4
!###################################################

!###################################################
!
subroutine deryy4(phi,nx,ny,dfi,yly)
!
!Fourth-order second derivative in the y direction
!###################################################

  implicit none
  real(8),dimension(nx,ny) ::  phi,dfi
  real(8) :: dly,yly,udy
  integer :: i,j,nx,ny

  dly=yly/ny
  udy=1./(12.*dly*dly)
  do j=3,ny-2
  do i=1,nx
  dfi(i,j)=udy*(-phi(i,j+2)+16.*phi(i,j+1)-30.*phi(i,j)+16.*phi(i,j-1)-phi(i,j-2))
  enddo
  enddo
  do i=1,nx
  dfi(i,1)=udy*(-phi(i,3)+16.*phi(i,2)-30.*phi(i,1)+16.*phi(i,ny)-phi(i,nx-1))
  dfi(i,2)=udy*(-phi(i,4)+16.*phi(i,3)-30.*phi(i,2)+16.*phi(i,1)-phi(i,ny))
  dfi(i,ny)=udy*(-phi(i,2)+16.*phi(i,1)-30.*phi(i,ny)+16.*phi(i,ny-1)-phi(i,ny-2))
  dfi(i,ny-1)=udy*(-phi(i,1)+16.*phi(i,ny)-30.*phi(i,ny-1)+16.*phi(i,ny-2)-phi(i,ny-3))
  enddo
  return
end subroutine deryy4
!###################################################
```

5. A simulation with the same conditions as those in question 1 has been run for 100000 time-step and visualization of results has been made at each 10000 time-step. As can be observed from the figure 13 to 21, when time-step reached 6000, the vorticity field remains the same. This implies that running the simulation for a very long time will lead to a steady vorticity field. The field becomes independent of the time, in other words, there is no more changes in the vorticity over time.

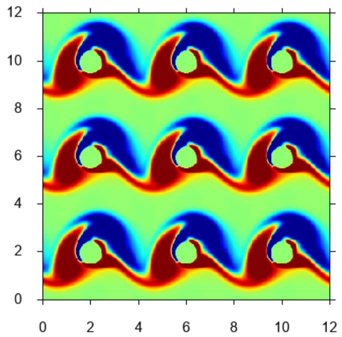

Figure 13. Vorticity at nt= 10000.

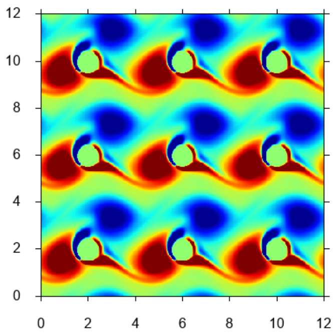

Figure 14. Vorticity at nt= 20000.

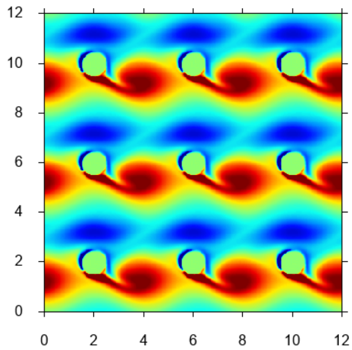

Figure 15. Vorticity at nt= 30000.

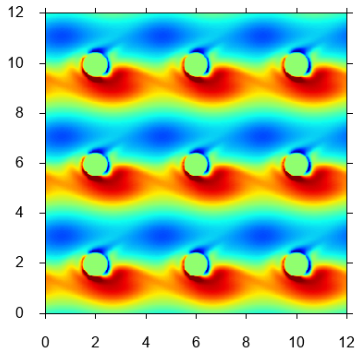

Figure 16. Vorticity at nt= 40000.

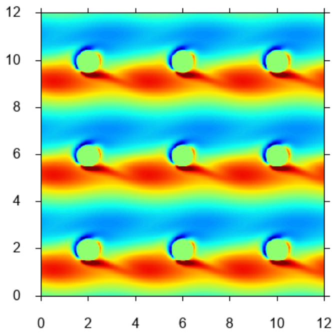

Figure 17. Vorticity at nt= 50000.

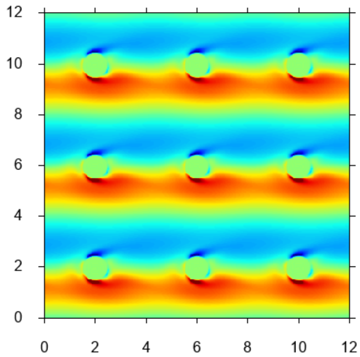

Figure 18. Vorticity at nt= 60000.

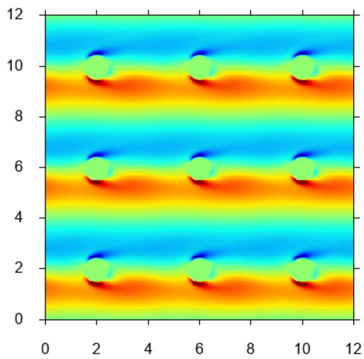

Figure 19. Vorticity at nt= 70000.

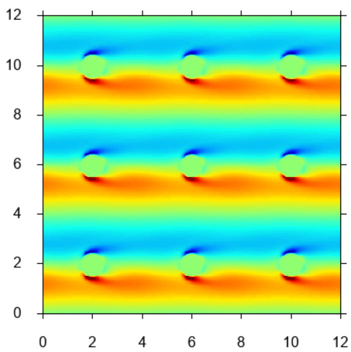

Figure 20. Vorticity at nt= 80000.

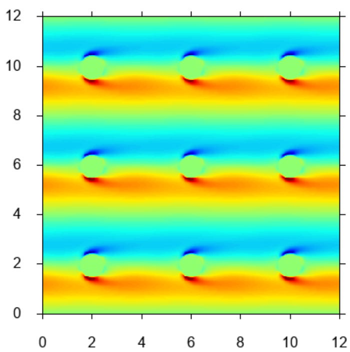

Figure 21. Vorticity at nt= 90000.

6. Code for to simulate a single cylinder.

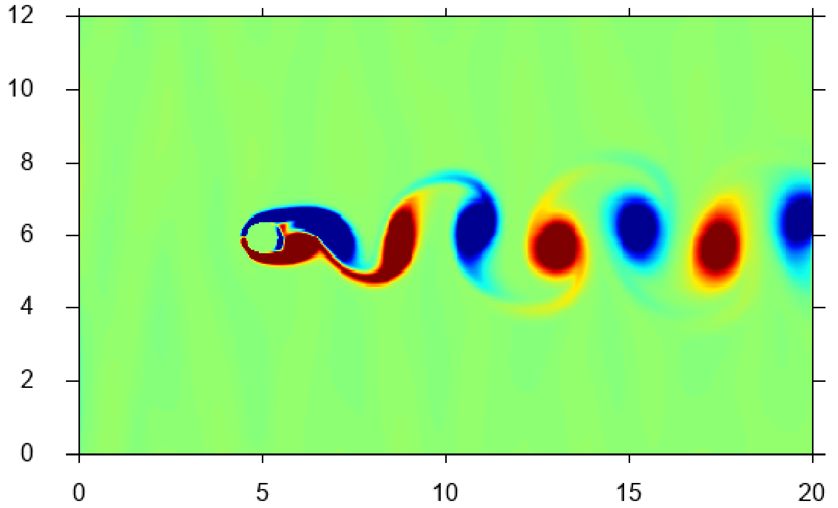

Figure 22: Flow around a single cylinder at nt = 32500

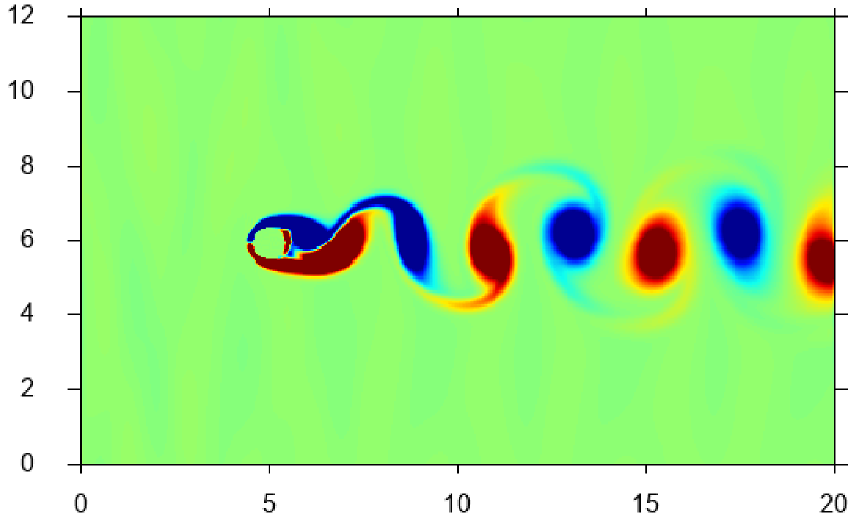

Figure 23: Flow around a single cylinder at nt = 45000

```
!###################################################################
!
subroutine boundary(uuu,vvv,rho,eee,rou,rov,roe,nx,ny,&
xlx,yly,xmu,xba,gma,chp,dlx,scp,dlt)
!###################################################################
implicit none
real(8),dimension(nx,ny) :: uuu,vvv,rho,eee,rou,rov,roe,scp
real(8) :: xlx,yly,xmu,xba,gma,chp,roi,cci,d,tpi,chv,uu0
real(8) :: dlt,dlx
integer :: nx,ny,i,j
!################################################################################
call param(xlx,yly,xmu,xba,gma,chp,roi,cci,d,tpi,chv,uu0)

!inlet boundary condition
do j=1,ny
rho(1,j)=roi
rou(1,j)=rho(1,j)*uuu(1,j)
rov(1,j)=rho(1,j)*vvv(1,j)
```

```
roe(1,j)=rho(1,j)*eee(1,j)
scp(1,j)=1.
enddo
!outlet boundary condition
do j=1,ny
rho(nx,j)=rho(nx,j)-uu0*(dlt/dlx)*(rho(nx,j)-rho(nx-1,j))
rou(nx,j)=rou(nx,j)-uu0*(dlt/dlx)*(rou(nx,j)-rou(nx-1,j))
rov(nx,j)=rov(nx,j)-uu0*(dlt/dlx)*(rov(nx,j)-rov(nx-1,j))
roe(nx,j)=roe(nx,j)-uu0*(dlt/dlx)*(roe(nx,j)-roe(nx-1,j))
scp(nx,j)=scp(nx,j)-uu0*(dlt/dlx)*(scp(nx,j)-scp(nx-1,j))
enddo
return
end subroutine boundary
!###################################################################

!##################################################
subroutine derix(phi,nx,ny,dfi,xlx)
!First derivative in the x direction
!##################################################
implicit none
real(8),dimension(nx,ny) :: phi,dfi
real(8) :: dlx,xlx,udx
integer :: i,j,nx,ny
dlx=xlx/nx
udx=1./(dlx+dlx)
do j=1,ny
dfi(1,j)=(1./dlx)*(phi(2,j)-phi(1,j))
do i=2,nx-1
dfi(i,j)=udx*(phi(i+1,j)-phi(i-1,j))
enddo
dfi(nx,j)=(1./dlx)*(phi(nx,j)-phi(nx-1,j))
enddo
return
end subroutine derix
!##################################################

!##################################################
subroutine derxx(phi,nx,ny,dfi,xlx)
!Second derivative in y direction
!##################################################
implicit none
real(8),dimension(nx,ny) ::  phi,dfi
real(8) :: dlx,xlx,udx
integer :: i,j,nx,ny
dlx=xlx/nx
udx=1./(dlx*dlx)
do j=1,ny
dfi(1,j)=udx*(phi(3,j)-(phi(2,j)+phi(2,j))+phi(1,j))
do i=2,nx-1
dfi(i,j)=udx*(phi(i+1,j)-(phi(i,j)+phi(i,j))&
+phi(i-1,j))
enddo
dfi(nx,j)=udx*(phi(nx,j)-(phi(nx-1,j)+phi(nx-1,j))+phi(nx-2,j))
enddo

return
end subroutine derxx
!##################################################
```