AERO70008 Coursework 2

CID: 01906145

Due: 3rd February 2025

Task 1 [5%]

This task simply required running the provided code. The simulation is described by the parameters in Table 1.

Table 1: Simulation parameters for Task 1

Re	nx	ny	nt	Scheme	CFL	Spatial order
200	129	129	10000	Adams-Bashforth	0.25	2

The vorticity field

$$\omega_z = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} \tag{1}$$

is visualised in Figure 1. Positive vorticity, as defined here, accordingly corresponds with counter-clockwise rotations. It serves to, in some sense, visualise the wake behind the cylinder and is indicative of losses in the flow.

Initially, the wake is (almost) symmetric in the transverse direction y. As the simulation progresses in time, the wake advects until it reaches the succeeding cylinder (at nt = 5000). There is a subsequent roll-up of the vorticity sheets (at nt = 7500), after which the vortices are finally shed and advected downstream before re-attaching to the following cylinders.

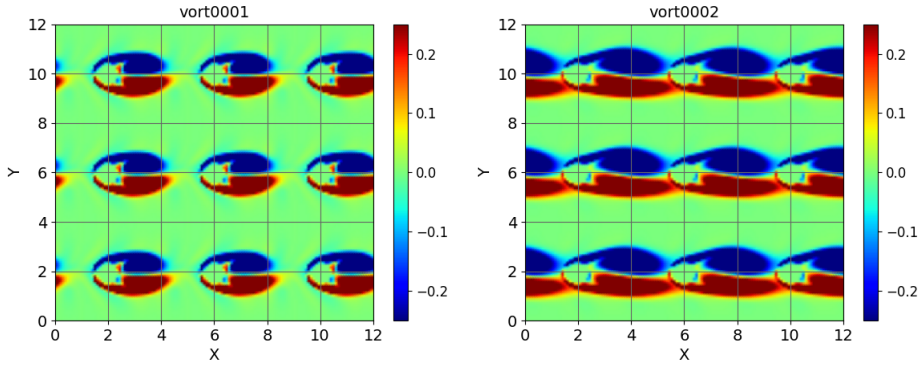
Task 2 [10%]

This task involved increasing the CFL number of the previous simulation from 0.25 to 0.75. The simulation is described by the parameters in table 2.

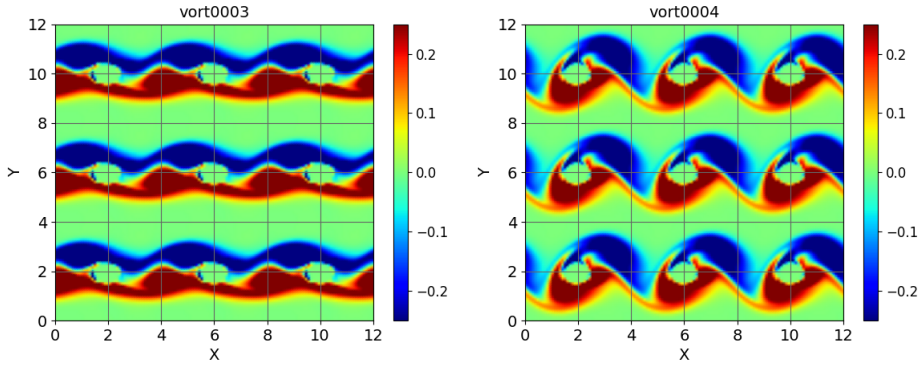
Table 2: Simulation parameters for Task 2

Re	nx	ny	nt	Scheme	CFL	Spatial order
200	129	129	10000	Adams-Bashforth	0.75	2

This simulation diverged due to an unstable numerical scheme (recall that divergence and stability are linked through the Lax equivalence theorem). The



(a) nt = 2500 (b) nt = 5000



(c) nt = 7500 (d) nt = 10000

Figure 1: Visualisation of vorticity field for Task 1 simulation (original code)

CFL number,

$$CFL = \frac{a\Delta t}{\Delta x} \tag{2}$$

where a is the propagation or convection velocity, is the parameter used to control the stability of a particular numerical scheme. Increasing the CFL number means that the physical solution propagates further in space (relative to the numerical solution). The first-order upwind numerical scheme is conditionally stable for CFL numbers less than one (meaning that the numerical solution must propagate at least the same distance as the physical one), i.e.

$$a\Delta t \leq \Delta x \tag{3}$$

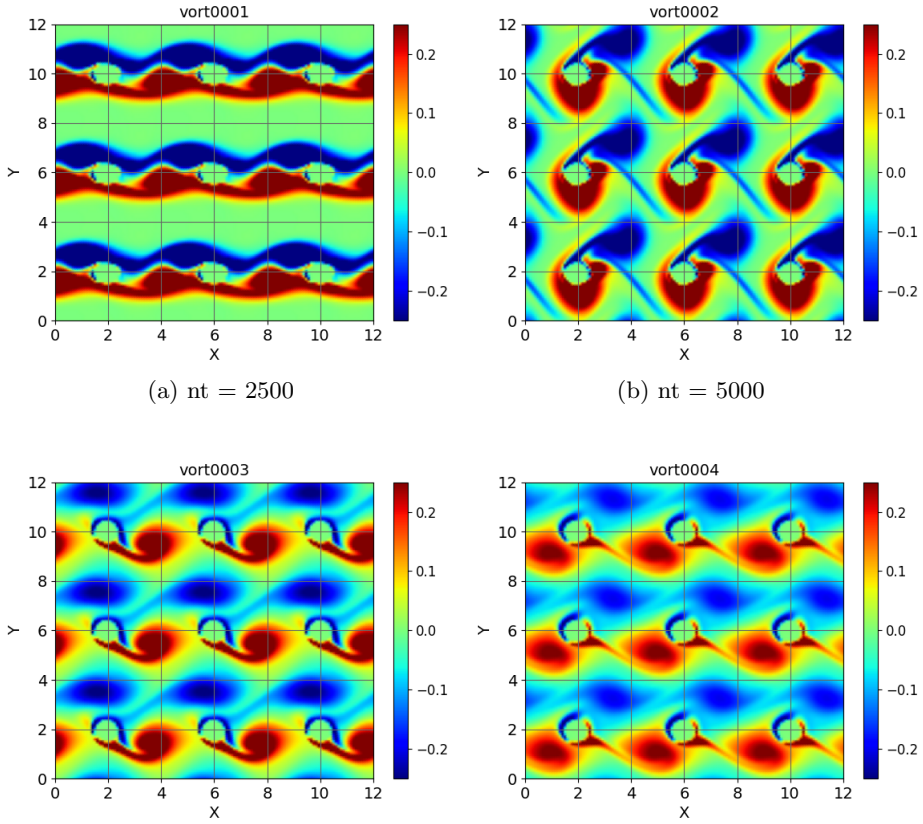
Note, however, that this stability condition is scheme-dependent. Through trial-and-error, the condition for stability of this scheme is found to be 0.35 (rounded to the nearest 0.05).

Task 3 [15%]

This task involved the implementation of an RK3 (third-order Runge-Kutta) time integration. The simulation is described by the parameters in table 3.

Table 3: Simulation parameters for Task 3

Re	nx	ny	nt	Scheme	CFL	Spatial order
200	129	129	10000	RK3	0.75	2



(a) nt = 2500

(b) nt = 5000

(c) nt = 7500

(d) nt = 10000

Figure 2: Visualisation of vorticity field for Task 3 simulation (RK3 temporal integration)

By replacing the Adams-Bashforth temporal integration with the third-order Runge-Kutta scheme, the finite-difference scheme is stabilised at higher CFL numbers (it is stable at least to a CFL of 0.75). The numerical solution is

presented in Figure 2. This means that simulations employing RK3 can be run with larger time steps (which are less expensive!). The trade-off, however, is that each full time step requires two additional sub-time steps.

Subroutine rkutta

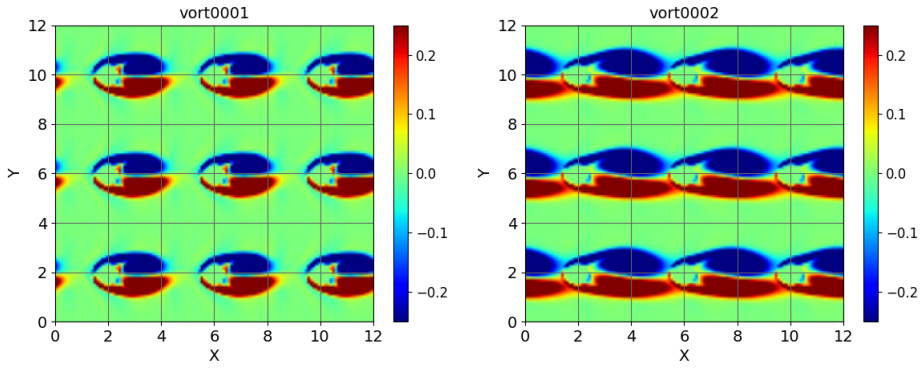
```
subroutine rkutta(rho,rou,rov,roe,fro,gro,fru,gru,frv,grv,&
     fre,gre,nx,ny,ns,dlt,coef,k)
!
!############################################################

  implicit none
!
  real(8),dimension(nx,ny) :: rho,rou,rov,roe,fro,gro,fru,gru,frv
  real(8),dimension(nx,ny) :: grv,fre,gre
  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.*dlt
   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)
     enddo
  enddo

  return
end subroutine rkutta
```

Task 4 [15%]

This task involved the implementation of higher-order spatial derivatives (fourth-order instead of second-order). The simulation is described by the parameters in table 4.

Table 4: Simulation parameters for Task 4

Re	nx	ny	nt	Scheme	CFL	Spatial order
200	129	129	10000	RK3	0.25	4



(a) nt = 2500

(b) nt = 5000

(c) nt = 7500

(d) nt = 10000

Figure 3: Visualisation of vorticity field for Task 4 simulation (fourth-order spatial schemes)

Increasing the order of accuracy of the spatial scheme does not negatively impact the stability of the solution. In fact, it i) permits a solution that is

qualitatively equivalent to the one obtained with the second-order scheme, and ii) permits a solution that is qualitatively equivalent to the one obtained with the Adams-Bashforth time integration. Figures 1 and 3 may be compared for evidence of this.

Subroutine derix4

```
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-1,j)=udx*(-phi(1,j)+8*phi(nx,j)-8*phi(nx-2,j)+phi(nx-3,j))
     dfi(nx,j)=udx*(-phi(2,j)+8*phi(1,j)-8*phi(nx-1,j)+phi(nx-2,j))
  enddo

  return
end subroutine derix4
```

Subroutine deriy4

```
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 i=1,nx
     dfi(i,1)=udy*(-phi(i,3)+8*phi(i,2)-8*phi(i,ny)+phi(i,ny-1))
     dfi(i,2)=udy*(-phi(i,4)+8*phi(i,3)-8*phi(i,1)+phi(i,ny))
     do j=3,ny-2
        dfi(i,j)=udy*(-phi(i,j+2)+8*phi(i,j+1)-8*phi(i,j-1)+phi(i,j-2))
     enddo
     dfi(i,ny-1)=udy*(-phi(i,1)+8*phi(i,ny)-8*phi(i,ny-2)+phi(i,ny-3))
     dfi(i,ny)=udy*(-phi(i,2)+8*phi(i,1)-8*phi(i,ny-1)+phi(i,ny-2))
  enddo

  return
end subroutine deriy4
```

Subroutine derxx4

```
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-1,j)=udx*(-phi(1,j)+16*phi(nx,j)-30*phi(nx-1,j)+16*phi(nx-2,j)-phi(nx-3,j))
     dfi(nx,j)=udx*(-phi(2,j)+16*phi(1,j)-30*phi(nx,j)+16*phi(nx-1,j)-phi(nx-2,j))
  enddo

  return
end subroutine derxx4
```

Subroutine deryy4

```
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 i=1,nx
     dfi(i,1)=udy*(-phi(i,3)+16*phi(i,2)-30*phi(i,1)+16*phi(i,ny)-phi(i,ny-1))
     dfi(i,2)=udy*(-phi(i,4)+16*phi(i,3)-30*phi(i,2)+16*phi(i,1)-phi(i,ny))
     do j=3,ny-2
        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
     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))
     dfi(i,ny)=udy*(-phi(i,2)+16*phi(i,1)-30*phi(i,ny)+16*phi(i,ny-1)-phi(i,ny-2))
  enddo

  return
end subroutine deryy4
```

Task 5 [10%]

This task involved commenting on the long-term behaviour of simulation 1; this was examined by running a simulation with 10x the number of timesteps as in Task 1. The simulation is described by the parameters in Table 5.

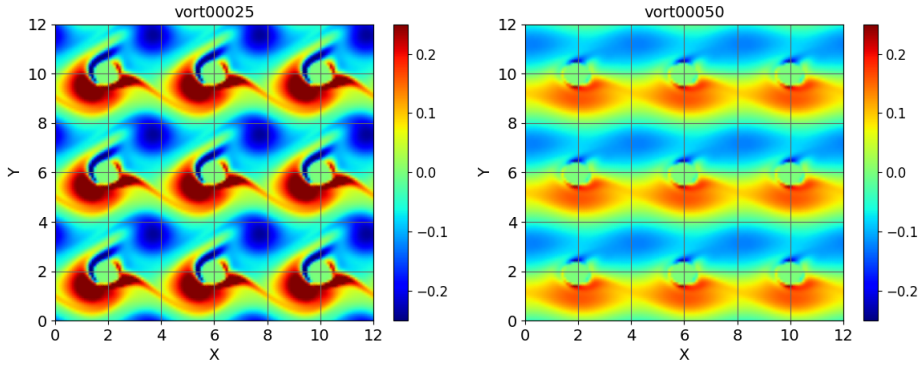
Table 5: Simulation parameters for Task 5

Re	nx	ny	nt	Scheme	CFL	Spatial order
200	129	129	100000	Adams-Bashforth	0.25	2

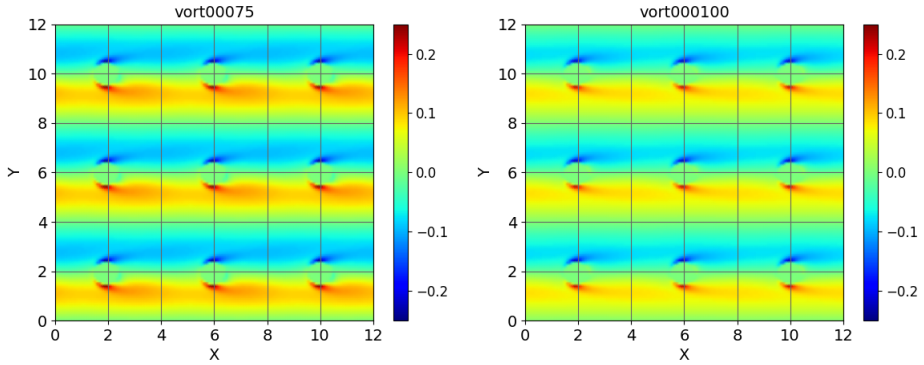
Vortex shedding is enabled by the initial transverse perturbations, which undergo a period of transient growth. However, since the flow is forced by a uniform streamwise velocity, it is expected that these will eventually be damped in time.

Figure 4 illustrates that the enstrophy (and thereby vorticity) dissipates over time, resulting in a distinct and approximately constant region of separation behind the cylinder.

[**Link to gif in github**]



(a) nt = 25000 (b) nt = 50000



(c) nt = 75000 (d) nt = 100000

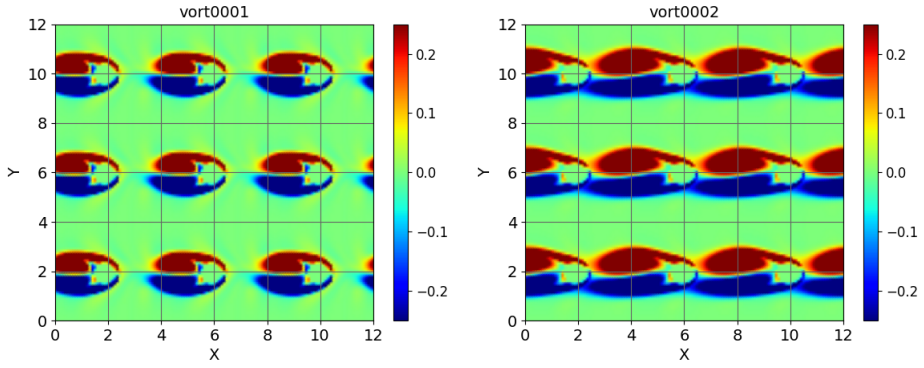
Figure 4: Visualisation of vorticity field for Task 5 simulation (long-term behaviour)

Task 6 [20%]

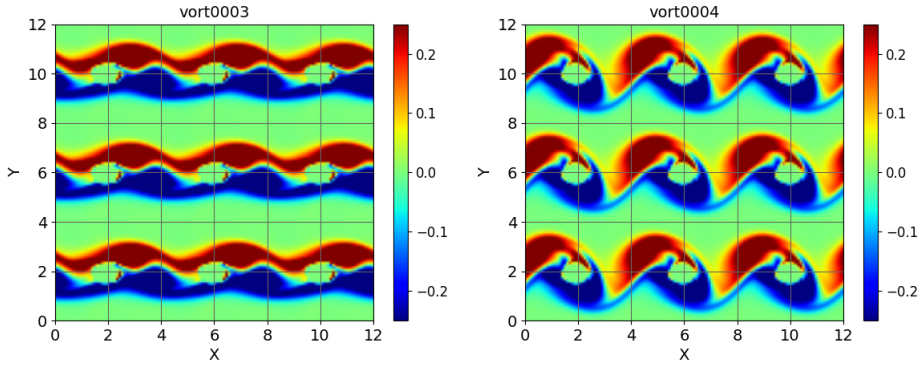
This task involved reversing the streamwise direction of the flow (of the Task 1 simulation). Two modifications were made to the initialising subroutine initl. The sign change on the initial streamwise velocity is self-explanatory, while the modification $i \rightarrow nx - i$ was due to the fact that the flow enters from the right hand side of the domain. Figures 1 and 5 demonstrate the appropriateness of these modifications for reversing the flow.

Before

```
...
uuu(i,j)=uu0
```



(a) nt = 2500 (b) nt = 5000



(c) nt = 7500 (d) nt = 10000

Figure 5: Visualisation of vorticity field for Task 6 simulation (flow reversal)

```
vvv(i,j)=0.01*(sin(4.*pi*(i)*dlx/xlx)&
     +sin(7.*pi*(i)*dlx/xlx))*&
     exp(-(j*dly-yly/2.)**2)
...
```

After

```
...
uuu(i,j)=-uu0
vvv(i,j)=0.01*(sin(4.*pi*(nx-i)*dlx/xlx)&
     +sin(7.*pi*(nx-i)*dlx/xlx))*&
     exp(-(j*dly-yly/2.)**2)
...
```

Task 7 [25%]

This task involved the replacement of period boundary conditions with inlet/outlet conditions, as well as the modification of the size of the domain and resolution of the simulation.

In both subroutines derix and derxx, the second-order central difference scheme was retained for the interior points, while second-order forward and backward difference schemes were implemented for the inlet and outlet boundary points. A new subroutine boundary was also introduced. For the inlet, the same Dirichlet conditions as in the subroutine initl, while for the outlet, the linear advection equation was employed.

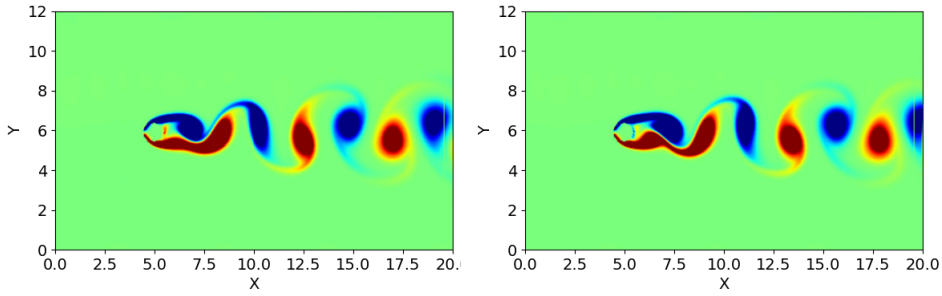


Figure 6: Visualisation of vorticity field for Task 7 simulation (single cylinder with inflow/outflow boundary conditions)

[Link to gif in github]

subroutine derix

```
...
do j=1,ny
      ! second-order, forward-difference for the first-derivative
      dfi(1,j)=udx*(-3*phi(1,j)+4*phi(2,j)-phi(3,j))
      do i=2,nx-1
         dfi(i,j)=udx*(phi(i+1,j)-phi(i-1,j))
      enddo
      ! second-order, backward-difference for the first-derivative
      dfi(nx,j)=udx*(3*phi(nx,j)-4*phi(nx-1,j)+phi(nx-2,j))
enddo
...
```

subroutine derxx

```
    ...
    do j=1,ny
        ! second-order, forward-difference for the second-derivative
        dfi(1,j)=udx*(2*phi(1,j)-5*phi(2,j)+4*phi(3,j)-phi(4,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
        ! second-order, backward-difference for the second-derivative
        dfi(nx,j)=udx*(2*phi(nx,j)-5*phi(nx-1,j)+4*phi(nx-2,j)-phi(nx-3,j))
    enddo
    ...
```

subroutine boundary

```
!######################################################################
!
subroutine boundary(uuu,vvv,rho,eee,pre,tmp,rou,rov,roe,nx,ny,dlt,&
   xlx,yly,xmu,xba,gma,uu0)
!
!######################################################################

   implicit none

   real(8),dimension(nx,ny) :: uuu,vvv,rho,eee,pre,tmp,rou,rov,roe
   real(8) :: xlx,yly,xmu,xba,gma,roi,cci,d,tpi,chv,uu0
   real(8) :: pi,dlx,dly,dlt,ct6,ct7,ct8
   integer :: j,ny,nx

   call param(xlx,yly,xmu,xba,gma,roi,cci,d,tpi,chv,uu0)

   dlx=xlx/nx
   dly=yly/ny
   ct6=(gma-1.)/gma
   ct7=gma-1.
   ct8=gma/(gma-1.)
   pi=acos(-1.)

   do j=1,ny
      ! inlet
      uuu(1,j)=uu0
      vvv(1,j)=0.01*(sin(4.*pi*(1)*dlx/xlx)&
           +sin(7.*pi*(1)*dlx/xlx))*&
           exp(-(j*dly-yly/2.)**2)
      tmp(1,j)=tpi
```

```
      eee(1,j)=chv*tmp(1,j)+0.5*(uuu(1,j)*uuu(1,j)+vvv(1,j)*vvv(1,j))
      rho(1,j)=roi
      pre(1,j)=rho(1,j)*ct6*tmp(1,j)
      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)

      ! outlet
      uuu(nx,j)=(rou(nx,j)-uu0*dlt/dlx*(rou(nx,j)-rou(nx-1,j)))/rho(nx,j)
      vvv(nx,j)=(rov(nx,j)-uu0*dlt/dlx*(rov(nx,j)-rov(nx-1,j)))/rho(nx,j)
      tmp(nx,j)=tpi
      eee(nx,j)=chv*tmp(nx,j)+0.5*(uuu(nx,j)*uuu(nx,j)+vvv(nx,j)*vvv(nx,j))
      rho(nx,j)=roi
      rou(nx,j)=rho(nx,j)*uuu(nx,j)
      rov(nx,j)=rho(nx,j)*vvv(nx,j)
      roe(nx,j)=rho(nx,j)*eee(nx,j)
      pre(nx,j)=ct7*(roe(nx,j)-0.5*(rou(nx,j)*uuu(nx,j)+&
             rov(nx,j)*vvv(nx,j)))
      tmp(nx,j)=ct8*pre(nx,j)/(rho(nx,j))
   enddo

   return
end subroutine boundary
!######################################################################
```