Term-project in
381C: Computational Physics

Visualization of the motion of vortices in 2D

Andreas P. Breu

May 9, 1997

Contents

1 Theory of complex stream functions

1.1 Flow field and stream function

I want to start up with a short introduction to complex stream functions in fluid mechanics and what are their limits.

The general differential equation describing any flow field is the Navier-Stokes equations, which can be derived from local law of conservation of momentum and the local law of conservation of mass,

$$\rho\left[\frac{\partial}{\partial t}+(\vec{u}\,\vec{\nabla})\right]u_i=-\frac{\partial}{\partial x_i}p+\mu\frac{\partial}{\partial x_i}\left[\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right]+\lambda\delta_{ij}\frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j}u_i \tag{1}$$

where $\vec{u}(\vec{r},t)$ denotes the flow field, ρ the density of the fluid, and p the pressure. λ and μ are both variables for the viscosity of the fluid. Further more assuming in good approximation that the fluid is incompressible the flow field has to satisfy the following equation:

$$\nabla\vec{u}=0 \tag{2}$$

Since the Navier-Stokes equations are three coupled nonlinear differential equations, just in a view cases a solution can be obtained analytically like the Poiseuille flow through a pipe. But in a lot cases the fluids can be treated as invicid. Then the equations (1) reduce to:

$$\rho\left[\frac{\partial}{\partial t}+(\vec{u}\,\vec{\nabla})\right]u_i=-\frac{\partial}{\partial x_i}p \tag{3}$$

They are also called Euler equations. Further more in the case of an ideal fluid, invicid and incompressible, which is also irrotational,

$$\nabla\times\vec{u}(\vec{r},t)$$

we can find a scalar function $\phi(\vec{r},t)$, such that

$$\nabla\phi(\vec{r},t)=\vec{u}(\vec{r},t) \tag{4}$$

ϕ has then the nice property to satisfy the Laplacian equation, which is obvious, if you substitute $\vec{u}$ by (4) in (2):

$$\nabla^2\phi=0$$

Hence the solution for the flow field is easier to find then by solving the Navier-Stokes equations. ϕ is called the velocity potential. Another simplification of the problem is achieved by restricting the flow to 2 dimensions. Equation (4) can be written as follows

$$\begin{aligned} \nabla\phi(\vec{r},t) &= (\vec{e}_x\frac{\partial}{\partial x} + \vec{e}_y\frac{\partial}{\partial y})\phi \\ &= u(\vec{r},t)\vec{e}_x + v(\vec{r},t)\vec{e}_y \end{aligned} \tag{5}$$

Lines of constant ϕ are called equipotential lines. Since we are restricted to the 2D case we are able to define analog to ϕ a function $\psi(\vec{r},t)$ which satisfies then

$$\begin{aligned} u &= \frac{\partial\psi}{\partial y} \\ v &= -\frac{\partial\psi}{\partial x} \end{aligned} \tag{6}$$

If you plug this definition into the equation for irrotationality, you will find that ψ also has to satisfy the Laplacian equation. But what is the meaning of the new function? The answer is obtained by setting $\psi = const$ and taking the total derivative

$$d\psi = \frac{\partial\psi}{\partial x}dx + \frac{\partial\psi}{\partial y}dy$$

and substitute the partial derivatives by (6).

$$d\psi = -v\,dx + u\,dy \stackrel{!}{=} 0$$

After a little bit of arithmetic we get

$$\frac{dx}{u} = \frac{dy}{v}$$

which is exactly the definition of the streamlines. Hence $\psi = const$ are streamlines and therefore ψ is called stream function.

Another property of the stream function and the velocity potential is that the combination of both satisfy the Cauchy-Riemann equations. This can easily be seen if you combine (5) and (6):

$$\frac{\partial\phi}{\partial x} = \frac{\partial\psi}{\partial y} \tag{7}$$

$$\frac{\partial\phi}{\partial y} = -\frac{\partial\psi}{\partial x} \tag{8}$$

Therefore it is just a small further step to define a analytical function $F(z)$, called complex stream function, with $z = x + i\,y$ which completely describes the flow field:

$$\begin{aligned} F(z) &= \phi + i\,\psi && (9) \\ \frac{dF}{dz} &= u - i\,v && (10) \end{aligned}$$

1.2 Examples for complex stream functions

The three basic examples are:

1. uniform flow field
2. sink and source
3. and the vortex

The uniform flow field is just given by

$$F(z) = cz$$

with a complex constant. The absolute value of c is the value of the velocity.

The other examples are covered by a more elaborated function, which is basically the same for all three cases:

$$F(z) = c\ln(z) \qquad (11)$$

Source and sink are expressed with a real constant c, proportional to the mass flow. c is positive for the source and negative for the sink. In the case of the vortex c is pure imaginary.

$$\begin{aligned} c &= -i\,\frac{\Gamma}{2\pi} && (12) \\ \Gamma &= \oint \vec{u}d\vec{l} \end{aligned}$$

The circulation Γ denotes the strength of the vortex. Taking the derivative like (10), setting $z = Re^{i\,\theta}$, and decomposing the result in radial and angular part you will find that there exist just a angular flow velocity.

$$\begin{aligned} u_R &= 0 \\ u_\theta &= \frac{c}{R} \end{aligned}$$

Free Vortex

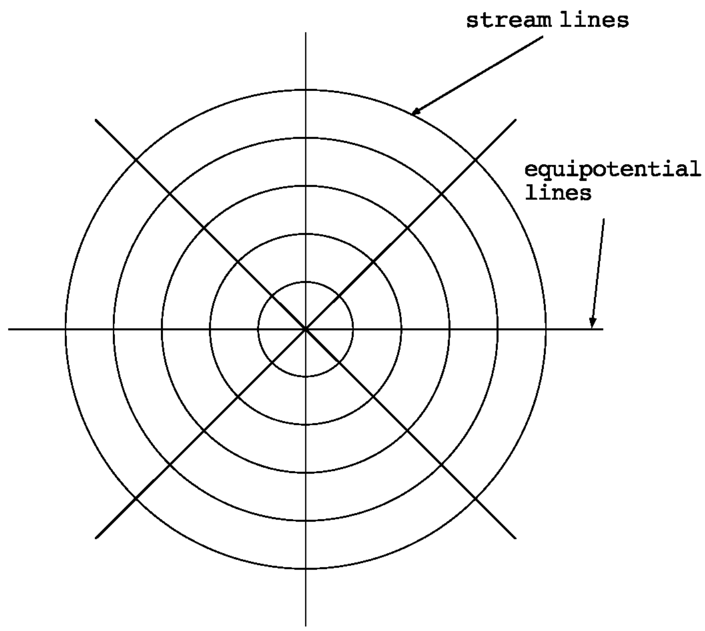

Figure 1: vortex

Hence a vortex is just a circular stream where the stream lines are concentric circles and the equipotential lines are radial lines (Fig. 1). For positive Γ the rotation is counterclockwise and otherwise for negative Γ.

This flow field is stationary in time and so not a interesting case. More interesting is the dynamics of vortices, which is given if one vortex interacts with a flow field due to sinks, sources, or other vortices. The basic principle for the interaction is the superposition of different parts of flow fields. Hence for two vortices you ad up the complex stream functions.

$$F(z) = -i\frac{\Gamma_1}{2\pi}\ln(z - z_1) - i\frac{\Gamma_2}{2\pi}\ln(z - z_2)$$

According to (10) the flow field is

$$\begin{aligned} u - iv &= -i\frac{\Gamma_1}{2\pi}\frac{-i(x - x_1) - (y - y_1)}{(x - x_1)^2 + (y - y_1)^2} \\ &\quad -i\frac{\Gamma_2}{2\pi}\frac{-i(x - x_2) - (y - y_2)}{(x - x_2)^2 + (y - y_2)^2} \end{aligned}$$

$z_1 = x_1 + i\,y_1$ and $z_2 = x_2 + i\,y_2$ are the center of the vortices. Evaluating the flow field due to the other components at one center of a vortex gives the equations of motion of the center of the vortex. Then each vortex can be treated as a point particle. For example the equations of motion for the vortex at z_1 would be

$$
\begin{aligned}
\frac{d\,x_1}{dt} &= \tilde{u}(x_1, y_1) = -\frac{\Gamma_2}{2\pi}\frac{y_1 - y_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2} \\
\frac{d\,y_1}{dt} &= \tilde{v}(x_1, y_1) = +\frac{\Gamma_2}{2\pi}\frac{x_1 - x_2}{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\end{aligned}
$$

1.3 Boundary conditions

Boundary conditions like walls, where the flow perpendicular has to vanish, are modeled by a appropriate number of image vortices, like image charges in electrostatics. The circulation has the same absolute value but opposite sign. The position is determined by imaging the real vortex at the wall, also like the image charge in electrostatics. As an example with a wall identical to the y-axes, using the previous results for the equations of motion with $x_1 = d$ as distance of the wall, $x_2 = -d$, and $\Gamma = \Gamma_1 = -\Gamma_2$ the equations are

$$
\begin{aligned}
\frac{d\,x_1}{dt} &= 0 \\
\frac{d\,y_1}{dt} &= -\frac{\Gamma}{2\pi}\frac{1}{2d}
\end{aligned}
$$

In the case of a corner, where a wall parallel to the positive x-axes and a wall parallel to the positive y-axes intersect, it is necessary to take one image vortex due to the "x-wall", one due to the "y-wall", and one due to the corner at $z = -x_1 - i\,y_1$ with the same circulation as the real vortex.

To get the right boundary conditions for parallel walls sums over a infinite number of image vortices has to be done. Hence for a box with side length L the sums have to cover a entire grid of image vortices (Fig. 2). The flow field looks then like

$$
\begin{aligned}
\frac{dF}{dz} &= -i\frac{\Gamma}{2\pi}\left[\frac{1}{(x-x_0)+i(y-y_0)}\right. && (13)\\
&+ \sum_{\substack{j=-\infty \\ k=-\infty}}^{\infty} \frac{1}{(x-(2Lj-x_0))+i(y-(2Lk-y_0))} && (14)\\
&- \sum_{\substack{j=-\infty \\ k=-\infty}}^{\infty} \frac{1}{(x-(2Lj-x_0))+i(y-(2Lk+y_0))} && (15)\\
&+ \sum_{\substack{j=-\infty \\ k=-\infty \\ /\{j=0,k=0\}}}^{\infty} \frac{1}{(x-(2Lj+x_0))+i(y-(2Lk+y_0))} && (16)\\
&- \left.\sum_{\substack{j=-\infty \\ k=-\infty}}^{\infty} \frac{1}{(x-(2Lj+x_0))+i(y-(2Lk-y_0))}\right] && (17)
\end{aligned}
$$

2 Implementation

Since this problem is similar to the n-body problem in problem 3 of homework set 3 I used this code as starting point and changed first the subroutine fcn.f to the match the new system of coupled differential equations. The integration is done by the LAPACK routine lsoda. The file fcn.f consists now of 5 fcn subroutines for 5 different scenarios of walls:

1. fcnfv: “free” vortices, no walls
2. fcn1w: one wall which is identical with the y-axes. The image vortices are place in the +x region
3. fcnc: describes a corner with walls identical with the positive x-axes and the positive y-axes
4. fcnpw: this subroutine models 2 parallel walls, one identical with the y-axes and a parallel one at $x = xlength$

5. fcnb: ...and finally the box described by walls a $x = 0$ and $x = xlength$ and $y = 0$ and $y = ylength$

The infinite sums have be replaced by finite ones from $-maxx$ to $maxx$ and $-maxy$ to $maxy$. The default value is $maxx = maxy = 10$, so that in the case of the box 399 image vortices are included. This value was chosen since the calculated positions are accurate up to the tenth digit after the decimal point and the calculations for two vortices in a box take something around 5 minutes on the Center for Relativity computer “einstein” (vortex 3.0 0.05 4 3 <twob.in). Further multipols of image vortices haven't been included, yet. The convergence was tested analytically with the criterion

$$\frac{a_{n+1}}{a_n} < \varepsilon \qquad \text{if } n > N$$

I also proved the convergence by letting on vortex circle several times in the box (vortex 20.0 0.05 4 2 <one.in, Fig. 3). A plot with gnuplot of its position showed no major deviation from a stable trajectory.

Further more I extended the subroutine “dvfrom” by Matthiew Choptuik in “dvmat” to be able to read matrices. This was necessary to read in the circulations and initial positions of the vortices.

Three different types of output forms can be chosen. First if “¡oform¿” has the value 1 (also default) the output has the required form of pp2d. For the value 2 just the position of the vortices is written to standard out. I needed this to control the convergence of the integration with gnuplot as mentioned before. If the value is 3 the main program vortex calls the subroutine “sf”, which calculates the value of the stream function on a grid specified by *level*, *xlength*, and *ylength* and writes the value to standard out. The last value is the the number of output times. I used this output to create the movies with “Mathematica”.

3 Results

There exist not many results since I spend most time in trouble shouting with the problem of convergence. It turned out to be the major problem of the project. Especially the reason for this is the convergence of the infinite sums like

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{n}$$

Therefore the integration diverges as soon as one sign in “fcnb” is changed.

Existing results are that the basic fluid mechanic problems interaction between two vortices, with one wall, one corner can be well simulated. Also the motion of one vortex in a box on a closed trajectory can be calculated without large mistakes. First runs with two vortices in a box showed some kind of chaotic behaviour, but I haven't checked so far, if it is a real effect of and artificial one due to the finite sums. Further exploration will be done and posted on my home page.

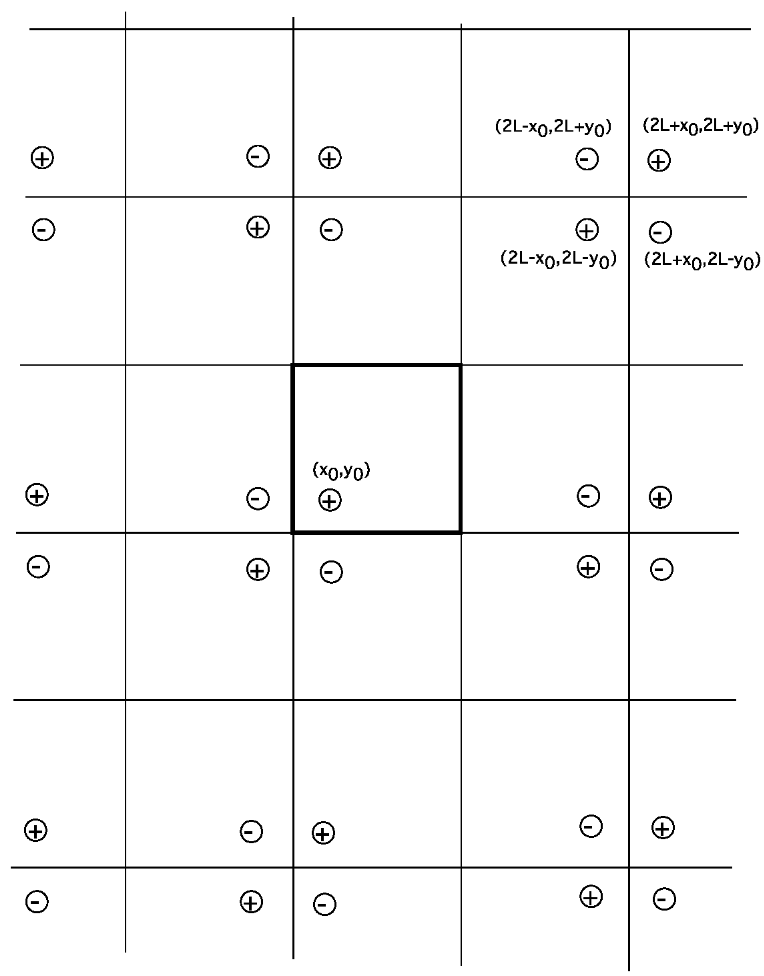

Figure 2: box and image vortices

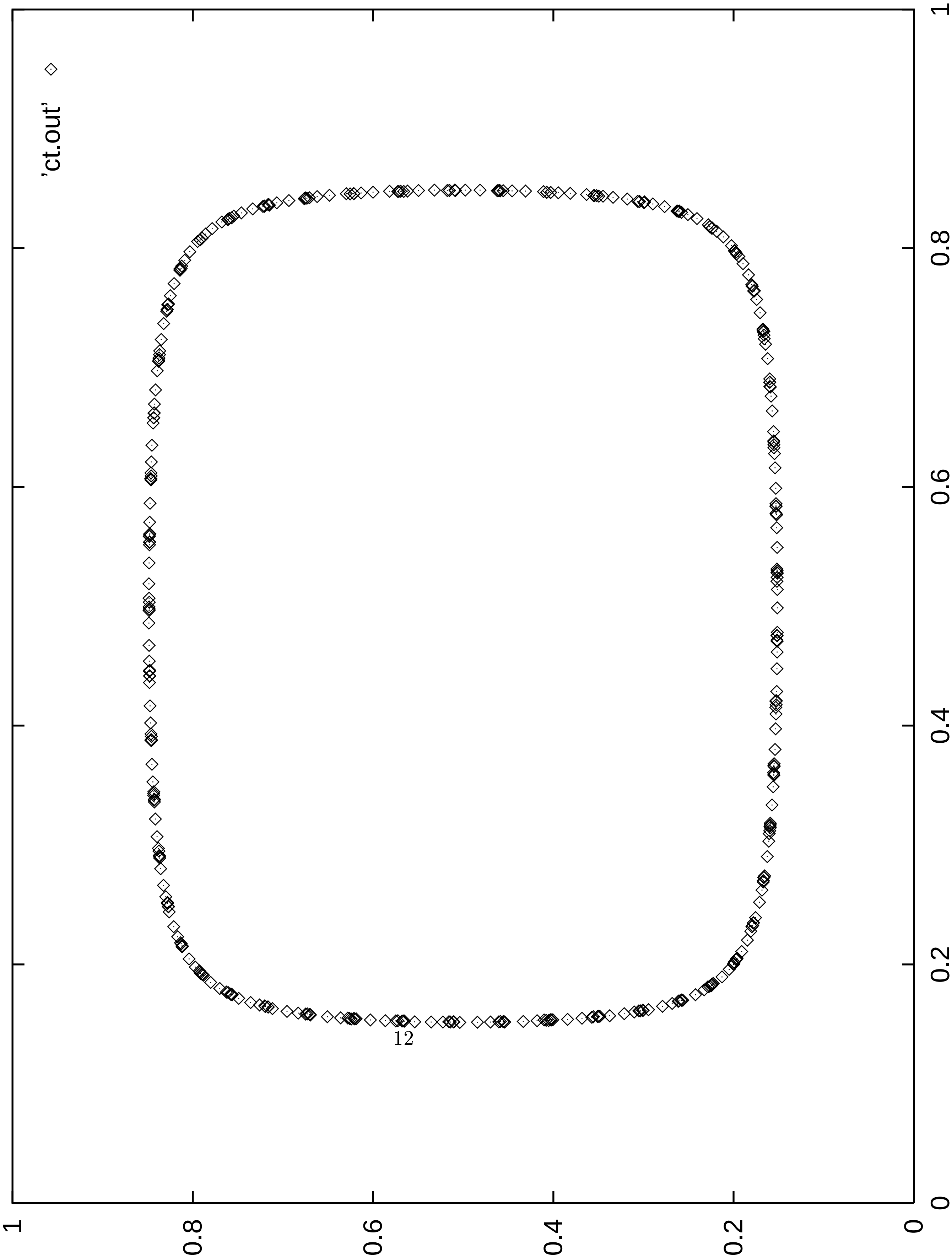

Figure 3: vortex in a box, about 30 oscillation

A Main program

```
c==========================================================
c     This program integrates the problem n vortices in a
c     rectangle of side length l. It uses LSODA for the
c     integration. The integration is performed in <dt out>
c     time steps from t=0 to t=tfinal.
c
c        usage: vortex <t final> <dt out> <walls>
c                      [<oform> <xwd> <ywd> <xorig> <yorig>
c                       <maxx> <maxy> <tol>]
c
c     <walls> gives the configuration of the walls:
c
c        0  no walls
c        1  one infinite wall identical with the y-axes
c        2  two perpenticular walls identical with the
c           x-axes and y-axes (corner)
c        3  two infinite parallel walls, one identical with
c           the y-axes, the second at +<xwd>
c        4  rectangular box with side length <xwd> and <ywd>
c
c     <oform> specifies the form of the output of the
c     calculation
c
c        1  (default) puts out the number of vortices, their
c           circulation, the time step and the position of
c           the vortices at that time step, such that the
c           data can be viewed with pp2d by Matt Choptuik
c        2  just puts out the positions of the vortices.
c           The purpose of this form is the control of the
c           trajectory of a single vortex for example with
c           ghost view.
c        3
c     tol is the optional tolerance for LSODA.
c==========================================================

      program vortex

      implicit     none

c----------------------------------------------------------
c     argument parsing variables
c----------------------------------------------------------
      integer      iargc,       i4arg
      real*8       r8arg
c----------------------------------------------------------
c     commandline variables
c----------------------------------------------------------
```

```
      real*8       tol
      real*8       r8_never,     default
      parameter  ( r8_never   = -1.0d-8 )
      parameter  ( default    =  1.0d-6 )

      real*8       tfinal,       dt_out,        t_out
      real*8       t,            l_default
      parameter  ( l_default  = 1.0d0 )

      integer      oform
      integer      wall_never,   oform_default
      integer      maxxy_default
      parameter  ( wall_never     = -1 )
      parameter  ( oform_default  =  1 )
      parameter  ( maxxy_default  = 10 )

      character    cdnm*5
      parameter  ( cdnm       = 'vortex' )
c------------------------------------------------------------
c     other variables
c------------------------------------------------------------
      integer      eps
      integer      iout,    nout
c------------------------------------------------------------
c     mathematical constants
c------------------------------------------------------------
      real*8       pi
      parameter  ( pi = 3.141592654 )
c------------------------------------------------------------
c     input variables
c
c     maxcol       the number of input parameters per line/
c                  vortex. It consists out of the strength
c                  of the vortex, characterized by the
c                  circulation gamma and its position in the
c                  complex plane
c     maxvort      is the maximum number of vortices which
c                  are handled by the program.
c------------------------------------------------------------

      external dvmat
      external sf

      include 'fcn.inc'

      integer      nvort,      ncol
      integer      maxvort,    maxcol
      parameter  ( maxcol      =   3 )
      parameter  ( maxvort     =  30 )
```

```
      real*8       mat(maxvort, maxcol)
      real*8       position(2*maxvort)

      integer      i, j, k, l,  sig

c-----------------------------------------------------------
c     variables for the call of LSODA
c-----------------------------------------------------------

      external       fcnfv,      fcn1w,        fcnc
      external       fcnpw,      fcnb,         jac

      integer        neq

      integer        itol,           itask,      istate
      integer        iopt,           lrw,        liw

      parameter    ( liw = 20 + 2*maxvort )
      parameter    ( lrw = 22 + 2*maxvort*max(16, maxvort + 9))

      integer        iwork(liw)
      integer        jt

      real*8         rtol,       atol
      real*8         rwork(lrw)
c-----------------------------------------------------------
c     xorig and yorig set the origin of the system. This is
c     the lower left corner of the box. A change of the
c     value effects only the output, not the calculation,
c     which is performt in a rectangle from (0,0) to (1,1).
c-----------------------------------------------------------
      real*8         xorig,      yorig,      orig_default
      parameter    ( orig_default = 0.0d0 )
c-----------------------------------------------------------
c     commantline argument parsing of the estimated
c     eigenvalue lambda and the optional tolerance
c-----------------------------------------------------------

      If((iargc() .lt. 3) .OR. (iargc() .gt. 6)) go to 900
         tfinal   = r8arg(1, r8_never)
         dt_out   = r8arg(2, r8_never)
         walls    = i4arg(3, wall_never)
         oform    = i4arg(4, oform_default)
         xlength  = r8arg(5, l_default)
         ylength  = r8arg(6, l_default)
         xorig    = r8arg(7, orig_default)
         yorig    = r8arg(8, orig_default)
         maxx     = i4arg(9, maxxy_default)
```

```
         maxy     = i4arg(10, maxxy_default)
         tol      = r8arg(11, default)

      If(( walls .lt. 0 ) .or. ( walls .gt. 4 )) then
         write(0,*) 'A system of walls must be defined:'
         write(0,*)
         write(0,*) '  0    free vortices, no walls'
         write(0,*) '  1    one wall, identical with the y-axes'
         write(0,*) '  2    two perpenticular wall, identical '
     &        // 'with the +x and +y axes'
         write(0,*) '  3    two parallel walls'
         write(0,*) '  4    rectangular box'
      end if
      If ( tfinal .eq. r8_never ) go to 1000
      If ( dt_out .eq. r8_never ) go to 1100
      If ( tol .le. 0 ) Then
         write(0,*) 'Warning: The tolerance must be positive '//
     &              'and real. The default value of 1.0d-8 '//
     &              'is used.'
      END IF

c------------------------------------------------------------
c     Get the masses and initial values from
c     standard input ...
c------------------------------------------------------------

      call dvmat('-',mat,nvort,ncol,maxvort,maxcol)

      if( nvort .le. 0 ) then
         write(0,*)
         write(0,*) 'vortex: No initial values read from standard input'
         write(0,*) 'creat a file or type the values of the form:'
         write(0,*)
         write(0,*) '   number_of_lines 3'
         write(0,*) '   gamma x0_1 y0_1'
         write(0,*) '        ...'
         write(0,*) '   gamma x0_i y0_i'
         write(0,*)
         write(0,*) 'write the values for one particle in one line'
         write(0,*)
         stop
      end if

c------------------------------------------------------------
c     setting the variables to the initial values
c------------------------------------------------------------
      do i = 1, nvort

c-----  circulation  -------------------------------------
```

```
        gamma(i) = mat(i,1)/2.0d0*pi

c-----  position of the i-th vortex  ----------------------

        position(2*(i-1)+1) = mat(i,2)
        position(2*(i-1)+2) = mat(i,3)
      end do

      neq = 2*nvort
      xlength = 1.0d0
      ylength = 1.0d0
c------------------------------------------------------------
c     seting the variables for LSODA (compare example
c     of the online reference)
c------------------------------------------------------------

      itol   = 1
      rtol   = tol
      atol   = tol
      itask  = 1
      istate = 1
      iopt   = 0
      jt     = 2

c------------------------------------------------------------
c     set the varibales for the loop control
c------------------------------------------------------------

      nout = int(tfinal/dt_out)
      t_out = 0

c------------------------------------------------------------
c     send starting point to stdout
c------------------------------------------------------------
      if( oform .eq. 1 ) then
         write(*,*) nvort
         do i = 1, nvort
            write(*,*) abs(gamma(i))
         end do
         write(*,*) t_out
         do i = 1, nvort
            write(*,600) position(2*(i-1)+1) + xorig,
     &           position(2*(i-1)+2) + yorig, 0
 600        format(1p,10E25.16)
         end do

      else if( oform .eq. 2 ) then
         do i = 1, nvort
```

```
           write(*,601) position(2*(i-1)+1) + xorig,
     &           position(2*(i-1)+2) + yorig
 601       format(1p,10E25.16)
        end do

      else if( oform .eq. 3 ) then
        call sf(neq, position)
      end if

c-----------------------------------------------------------
c     integration of the ODE with the package LSODA
c-----------------------------------------------------------

      DO iout = 1, nout
        istate = 1
        t    = dt_out*(iout -1)
        t_out = dt_out*iout


        if( walls .eq. 0 ) then
           call lsoda(fcnfv,neq,position,t,t_out,itol,rtol,atol,
     &           itask,istate,iopt,rwork,lrw,iwork,liw,
     &           jac,jt)
        else if( walls .eq. 1 ) then
           call lsoda(fcn1w,neq,position,t,t_out,itol,rtol,atol,
     &           itask,istate,iopt,rwork,lrw,iwork,liw,
     &           jac,jt)
        else if( walls .eq. 2 ) then
           call lsoda(fcnc,neq,position,t,t_out,itol,rtol,atol,
     &           itask,istate,iopt,rwork,lrw,iwork,liw,
     &           jac,jt)
        else if( walls .eq. 3 ) then
           call lsoda(fcnpw,neq,position,t,t_out,itol,rtol,atol,
     &           itask,istate,iopt,rwork,lrw,iwork,liw,
     &           jac,jt)
        else if( walls .eq. 4 ) then
           call lsoda(fcnb,neq,position,t,t_out,itol,rtol,atol,
     &           itask,istate,iopt,rwork,lrw,iwork,liw,
     &           jac,jt)
        end if

        IF( istate .lt. 0 ) go to 2000

c-----------------------------------------------------------
c     output of the new particle positions
c-----------------------------------------------------------
      if( oform .eq. 1 ) then
        write(*,*) t_out
        do i = 1, nvort
```

```
            write(*,500) position(2*(i-1)+1) + xorig,
     &           position(2*(i-1)+2) + yorig, 0
 500        format(1p,10E25.16)
         end do

      else if( oform .eq. 2 ) then
         do i = 1, nvort
            write(*,501) position(2*(i-1)+1) + xorig,
     &           position(2*(i-1)+2) + yorig
 501        format(1p,10E25.16)
         end do

      else if( oform .eq. 3 ) then
         call sf(neq, position)
      end if
c-----------------------------------------------------------
c     end of the integration loop
c-----------------------------------------------------------
      END DO

      if( oform .eq. 3 ) then
         write(*,*)  iout+1
      end if

      stop

c-----------------------------------------------------------
c     error messages
c-----------------------------------------------------------

 900  continue
         write(0,*)
         write(0,*) '    usage: vortex <t final> <dt out> '
     &        // '<walls> [<oform> <xwd> <ywd>'
         write(0,*) '           <xorig> <yorig> <maxx> '
     &        // '<maxy> <tol>]'
         write(0,*)
      stop

 1000 continue
         write(0,*)
         write(0,*) 'tfinal must be a positive real constant'
         write(0,*) 'it is the final time of the integration'
         write(0,*)
      stop

 1100 continue
         write(0,*)
         write(0,*) 'dt must be a positive real constant'
```

```
         write(0,*) 'dt are the time steps in the integration'
         write(0,*)
      stop

 2000 continue
      if( oform .ne. 1 ) then
         write(*,*)  iout+1
      end if
         write(0,*)
         write(0,*) 'LSODA didnt succeed. Error: ',istate
         write(0,*)
      stop

      end
```

B fcn.f

```
c------------------------------------------------------------
c     fcn.f supports the subroutine fcn, which implements
c     the differntial equations for lsoda. It also implements
c     the subroutine jac for a jacobian matrix, optinally
c     used by lsoda.
c------------------------------------------------------------
c     This special subroutine simulates free vortices
c------------------------------------------------------------
      subroutine fcnfv(neq, t, r, rdot)

         implicit none
         include 'fcn.inc'

         integer     i, j
         integer     neq,     nvort
c------------------------------------------------------------
c     maxx and maxy must be large then 5. 8*maxx and 8*maxy
c     is the number if image vortices in x and y direction
c------------------------------------------------------------
         real*8      sum,      denominator
         real*8      t,        r(neq),       rdot(neq)
c------------------------------------------------------------
c     implementation of the ode's
c------------------------------------------------------------
         nvort = neq / 2

         do i = 1, nvort
c------------------------------------------------------------
c     interaction with the other vortices
c------------------------------------------------------------
            rdot(2*(i-1)+1) = 0
```

```
            rdot(2*(i-1)+2) = 0

            do j = 1, nvort
               If(i .ne. j) then
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1) - r(2*(j-1)+1)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          -r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
               end if
            end do
         end do
c-----------------------------------------------------------
c     end of fcnfv
c-----------------------------------------------------------
         end


c-----------------------------------------------------------
c     This special subroutine simulates vortices at one
c     infinite wall, which is the y axes
c-----------------------------------------------------------
      subroutine fcn1w(neq, t, r, rdot)
```

```
      implicit none
      include 'fcn.inc'

      integer     i, j
      integer     neq,     nvort
c-----------------------------------------------------------
c     maxx and maxy must be large then 5. 8*maxx and 8*maxy
c     is the number if image vortices in x and y direction
c-----------------------------------------------------------
      real*8      sum,      denominator
      real*8      t,        r(neq),       rdot(neq)
c-----------------------------------------------------------
c     implementation of the ode's
c-----------------------------------------------------------
      nvort = neq / 2

      do i = 1, nvort
c-----------------------------------------------------------
c     reset of the variables
c-----------------------------------------------------------
         rdot(2*(i-1)+1) = 0
         rdot(2*(i-1)+2) = 0
c-----------------------------------------------------------
c     set the new values
c-----------------------------------------------------------
         do j = 1, nvort
            If(i .ne. j) then
c-----------------------------------------------------------
c     interaction with the other vortices
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
               rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
               rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 + gamma(j)
```

```
     &                   *( r(2*(i-1)+1) - r(2*(j-1)+1)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     interaction with the "other" image vortices
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
                 rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        + r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
                 rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+1) + r(2*(j-1)+1)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        + r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     end of interaction with other image vortices
c-----------------------------------------------------------


               else if ( i .eq. j ) then
c-----------------------------------------------------------
```

```
c     interaction with the "own" image vortices
c-------------------------------------------------------------
c     x component stays constant
c-------------------------------------------------------------
c     y component
c-------------------------------------------------------------
                 rdot( 2*(i-1)+2 ) = rdot( 2*(i-1)+2 )
     &                - gamma(i) / (2 * r( 2*(i-1)+1 ))


              end if
           end do
        end do
c-------------------------------------------------------------
c     end of fcn1w
c-------------------------------------------------------------
        end


c-------------------------------------------------------------
c     This special subroutine simulates vortices at two
c     perpenticular walls (a corner), which are the positive
c     x and y axes
c-------------------------------------------------------------
      subroutine fcnc(neq, t, r, rdot)

        implicit none
        include 'fcn.inc'

        integer     i, j
        integer     neq,     nvort
c-------------------------------------------------------------
c     maxx and maxy must be large then 5. 8*maxx and 8*maxy
c     is the number if image vortices in x and y direction
c-------------------------------------------------------------
        real*8      sum,      denominator
        real*8      t,        r(neq),       rdot(neq)
c-------------------------------------------------------------
c     implementation of the ode's
c-------------------------------------------------------------
        nvort = neq / 2

        do i = 1, nvort
c-------------------------------------------------------------
c     interaction with the other vortices
c-------------------------------------------------------------
           rdot(2*(i-1)+1) = 0
           rdot(2*(i-1)+2) = 0
```

```
            do j = 1, nvort
               If(i .ne. j) then
c-----------------------------------------------------------
c     interaction with the other vortices
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1) - r(2*(j-1)+1)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     interaction with the "other" image vortices
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
c     interaction with image vortices in the -x y plane
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        + r(2*(j-1)+1)
     &                      )**2.0d0
```

```
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-------------------------------------------------------------
c     interaction with image vortices in the x -y plane
c-------------------------------------------------------------
                rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &               + gamma(j)
     &                 *( r(2*(i-1)+2) + r(2*(j-1)+2)
     &                  )
     &                 /(
     &                    ( r(2*(i-1)+1)
     &                      - r(2*(j-1)+1)
     &                    )**2.0d0
     &                    + ( r(2*(i-1)+2)
     &                        + r(2*(j-1)+2)
     &                    )**2.0d0
     &                  )
c-------------------------------------------------------------
c     interaction with image vortices in the -x -y plane
c-------------------------------------------------------------
                rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &               - gamma(j)
     &                 *( r(2*(i-1)+2) + r(2*(j-1)+2)
     &                  )
     &                 /(
     &                    ( r(2*(i-1)+1)
     &                      + r(2*(j-1)+1)
     &                    )**2.0d0
     &                    + ( r(2*(i-1)+2)
     &                        + r(2*(j-1)+2)
     &                    )**2.0d0
     &                  )
c-------------------------------------------------------------
c     y component
c-------------------------------------------------------------
c     interaction with image vortices in the -x y plane
c-------------------------------------------------------------
                rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &               - gamma(j)
     &                 *( r(2*(i-1)+1) + r(2*(j-1)+1)
     &                  )
     &                 /(
     &                    ( r(2*(i-1)+1)
     &                      + r(2*(j-1)+1)
     &                    )**2.0d0
     &                    + ( r(2*(i-1)+2)
     &                        - r(2*(j-1)+2)
```

```
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     interaction with image vortices in the x -y plane
c-----------------------------------------------------------
                 rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                - gamma(j)
     &                  *( r(2*(i-1)+1) - r(2*(j-1)+1)
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - r(2*(j-1)+1)
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         + r(2*(j-1)+2)
     &                     )**2.0d0
     &                   )
c-----------------------------------------------------------
c     interaction with image vortices in the -x y plane
c-----------------------------------------------------------
                 rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                + gamma(j)
     &                  *( r(2*(i-1)+1) + r(2*(j-1)+1)
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       + r(2*(j-1)+1)
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         + r(2*(j-1)+2)
     &                     )**2.0d0
     &                   )
c-----------------------------------------------------------
c     end of interaction with other image vortices
c-----------------------------------------------------------


              else if ( i .eq. j ) then
c-----------------------------------------------------------
c     interaction with the "own" image vortices
c-----------------------------------------------------------
c     image vortex in the -x y plane:
c     x component stays constant
c     y component
c-----------------------------------------------------------
                 rdot( 2*(i-1)+2 ) = rdot( 2*(i-1)+2 )
     &                - gamma(i) / (2 * r( 2*(i-1)+1 ))
c-----------------------------------------------------------
c     image vortex in the x -y plane:
```

```
c     y component stays constant
c     x component
c-----------------------------------------------------------
                 rdot( 2*(i-1)+1 ) = rdot( 2*(i-1)+1 )
     &                + gamma(i) / (2 * r( 2*(i-1)+2 ))
c-----------------------------------------------------------
c     image vortex in the -x -y plane:
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
                 rdot( 2*(i-1)+1 ) = rdot( 2*(i-1)+1 )
     &                - gamma(i)
     &                  *(2 * r( 2*(i-1)+2 ))
     &                  /(
     &                     (2 * r( 2*(i-1)+1 ))**2.0d0
     &                     + (2 * r( 2*(i-1)+2 ))**2.0d0
     &                   )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
                 rdot( 2*(i-1)+2 ) = rdot( 2*(i-1)+2 )
     &                + gamma(i)
     &                  *(2 * r( 2*(i-1)+1 ))
     &                  /(
     &                     (2 * r( 2*(i-1)+1 ))**2.0d0
     &                     + (2 * r( 2*(i-1)+2 ))**2.0d0
     &                   )
c-----------------------------------------------------------
c     end of interaction of "own image vortices
c-----------------------------------------------------------
              end if
           end do
        end do
c-----------------------------------------------------------
c     end of fcnc
c-----------------------------------------------------------
        end


c-----------------------------------------------------------
c     This special subroutine simulates vortices at two
c     infinite parallel walls parallel to the y axis
c-----------------------------------------------------------
      subroutine fcnpw(neq, t, r, rdot)

        implicit none
        include 'fcn.inc'

        integer     i, j, k, l
```

```
        integer     neq,     nvort
c-----------------------------------------------------------
c     maxx and maxy must be large then 5. 8*maxx and 8*maxy
c     is the number if image vortices in x and y direction
c-----------------------------------------------------------
        real*8      sum,      denominator
        real*8      t,        r(neq),       rdot(neq)
c-----------------------------------------------------------
c     implementation of the ode's
c-----------------------------------------------------------
        nvort = neq / 2

        do i = 1, nvort
c-----------------------------------------------------------
c     interaction with the other vortices
c-----------------------------------------------------------
           rdot(2*(i-1)+1) = 0
           rdot(2*(i-1)+2) = 0

           do j = 1, nvort
              If(i .ne. j) then
c-----------------------------------------------------------
c     interaction with the other vortices
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
                 rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                - gamma(j)
     &                  *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - r(2*(j-1)+1)
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         - r(2*(j-1)+2)
     &                     )**2.0d0
     &                   )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
                 rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                + gamma(j)
     &                  *( r(2*(i-1)+1) - r(2*(j-1)+1)
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - r(2*(j-1)+1)
     &                     )**2.0d0
```

```
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(i-1)+2)
     &                      )**2.0d0
     &                    )
c------------------------------------------------------------
c     interaction with the "other" image vortices
c------------------------------------------------------------
                  do k = -maxx, maxx
c------------------------------------------------------------
c     x component
c------------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1) )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )


                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1) )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c------------------------------------------------------------
c     y component
c------------------------------------------------------------
                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          - r(2*(j-1)+1) )
     &                    )
     &                   /(
```

```
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1) )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )


                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          + r(2*(j-1)+1) )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1) )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-------------------------------------------------------------
c     end of interaction with other image vortices
c-------------------------------------------------------------
               end do

         else if( i .eq. j ) then
c-------------------------------------------------------------
c     interaction with "own" image vortices
c-------------------------------------------------------------
c     Since the walls are parallel to the y axis, just the
c     y coordinate is affected
c-------------------------------------------------------------
            do k = -maxx, maxx
               if( k .ne. 0 ) then
                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 - gamma(j)
     &                   /( 2 * r(2*(i-1)+1)
     &                      - 2 * k * xlength
     &                    )


                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   /( -2 * k * xlength
```

```
     &                    )
               end if
            end do
         end if
      end do
      end do
c-----------------------------------------------------------
c     end of fcnpw
c-----------------------------------------------------------
      end


c-----------------------------------------------------------
c     The subroutine fcnb is for the vortices in a box
c-----------------------------------------------------------
      subroutine fcnb(neq, t, r, rdot)

         implicit none
         include 'fcn.inc'

         integer     i, j, k, l
         integer     neq,     nvort
c-----------------------------------------------------------
c     maxx and maxy must be large then 5. 8*maxx and 8*maxy
c     is the number if image vortices in x and y direction
c-----------------------------------------------------------
         real*8      sum,      denominator
         real*8      t,        r(neq),       rdot(neq)
c-----------------------------------------------------------
c     implementation of the ode's
c-----------------------------------------------------------
         nvort = neq / 2

         do i = 1, nvort
c-----------------------------------------------------------
c     interaction with the other vortices
c-----------------------------------------------------------
            rdot(2*(i-1)+1) = 0
            rdot(2*(i-1)+2) = 0

            do j = 1, nvort
               If(i .ne. j) then
c-----------------------------------------------------------
c     interaction with the other vortices
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
```

```
     &                   *( r(2*(i-1)+2) - r(2*(j-1)+2)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
                  rdot(2*(i-1)+2)= rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1) - r(2*(j-1)+1)
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - r(2*(j-1)+1)
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - r(2*(j-1)+2)
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     interaction with the "other" image vortices
c-----------------------------------------------------------
                  do k = -maxx, maxx
                     do l = -maxy, maxy
c-----------------------------------------------------------
c     x component
c-----------------------------------------------------------
c     influence due to term 1
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+2)
     &                      - ( 2 * l * ylength
     &                          + r(2*(j-1)+2)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
```

```
     &                          - ( 2 * l * ylength
     &                              + r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     influence due to term 2
c-----------------------------------------------------------
                 rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                - gamma(j)
     &                  *( r(2*(i-1)+2)
     &                     - ( 2 * l * ylength
     &                         - r(2*(j-1)+2)
     &                       )
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - ( 2 * k * xlength
     &                           - r(2*(j-1)+1)
     &                         )
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         - ( 2 * l * ylength
     &                             - r(2*(j-1)+2)
     &                           )
     &                     )**2.0d0
     &                   )
c-----------------------------------------------------------
c     influence due to term 3
c-----------------------------------------------------------
                 rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                + gamma(j)
     &                  *( r(2*(i-1)+2)
     &                     - ( 2 * l * ylength
     &                         - r(2*(j-1)+2)
     &                       )
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - ( 2 * k * xlength
     &                           + r(2*(j-1)+1)
     &                         )
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         - ( 2 * l * ylength
     &                             - r(2*(j-1)+2)
     &                           )
     &                       )**2.0d0
     &                   )
```

```
c-----------------------------------------------------------
c     influence due to term 4
c-----------------------------------------------------------
                 rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                - gamma(j)
     &                  *( r(2*(i-1)+2)
     &                     - ( 2 * l * ylength
     &                         + r(2*(j-1)+2)
     &                       )
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - ( 2 * k * xlength
     &                           + r(2*(j-1)+1)
     &                         )
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         - ( 2 * l * ylength
     &                             + r(2*(j-1)+2)
     &                           )
     &                        )**2.0d0
     &                   )
c-----------------------------------------------------------
c     y component
c-----------------------------------------------------------
c     influence due to term 1
c-----------------------------------------------------------
                 rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                - gamma(j)
     &                  *( r(2*(i-1)+1)
     &                     - ( 2 * k * xlength
     &                         - r(2*(j-1)+1)
     &                       )
     &                   )
     &                  /(
     &                     ( r(2*(i-1)+1)
     &                       - ( 2 * k * xlength
     &                           - r(2*(j-1)+1)
     &                         )
     &                     )**2.0d0
     &                     + ( r(2*(i-1)+2)
     &                         - ( 2 * l * ylength
     &                             + r(2*(j-1)+2)
     &                           )
     &                     )**2.0d0
     &                   )
c-----------------------------------------------------------
c     influence due to term 2
c-----------------------------------------------------------
```

```
              rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          - r(2*(j-1)+1)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              - r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c------------------------------------------------------------
c     influence due to term 3
c------------------------------------------------------------
              rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          + r(2*(j-1)+1)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              - r(2*(j-1)+2)
     &                            )
     &                        )**2.0d0
     &                    )
c------------------------------------------------------------
c     influence due to term 4
c------------------------------------------------------------
              rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          + r(2*(j-1)+1)
```

```
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              + r(2*(j-1)+2)
     &                            )
     &                        )**2.0d0
     &                    )
c------------------------------------------------------------
c     end of interaction with "other" image vortices
c------------------------------------------------------------
c--------  l loop ------------------------------------------
                      end do
c--------  k loop ------------------------------------------
                    end do


         else if( i .eq. j ) then
c------------------------------------------------------------
c     interaction with "own" image vortices
c------------------------------------------------------------
            do k = -maxx, maxx
               do l = -maxy, maxy
c------------------------------------------------------------
c     x component
c------------------------------------------------------------
c     influence due to term 1
c------------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+2)
     &                      - ( 2 * l * ylength
     &                          + r(2*(j-1)+2)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
```

```
     &                              + r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     influence due to term 2
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+2)
     &                      - ( 2 * l * ylength
     &                          - r(2*(j-1)+2)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              - r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     influence due to term 3
c-----------------------------------------------------------
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+2)
     &                      - ( 2 * l * ylength
     &                          - r(2*(j-1)+2)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              - r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
```

```
c     influence due to term 4
c------------------------------------------------------------
                if(( k .ne. 0 ) .or. ( l .ne. 0 ))then
                  rdot(2*(i-1)+1) = rdot(2*(i-1)+1)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+2)
     &                      - ( 2 * l * ylength
     &                          + r(2*(j-1)+2)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              + r(2*(j-1)+2)
     &                            )
     &                        )**2.0d0
     &                    )
c---  (( k .ne. 0 ) .or. ( l .ne. 0 ))  --------------------
                end if
c------------------------------------------------------------
c     y component
c------------------------------------------------------------
c     influence due to term 1
c------------------------------------------------------------
                  rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          - r(2*(j-1)+1)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              + r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c------------------------------------------------------------
```

```
c     influence due to term 2
c-----------------------------------------------------------
                  rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          - r(2*(j-1)+1)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            - r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              - r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     influence due to term 3
c-----------------------------------------------------------
                  rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 - gamma(j)
     &                   *( r(2*(i-1)+1)
     &                      - ( 2 * k * xlength
     &                          + r(2*(j-1)+1)
     &                        )
     &                    )
     &                   /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1) )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              - r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                    )
c-----------------------------------------------------------
c     influence due to term 4
c-----------------------------------------------------------
                if(( k .ne. 0 ) .or. ( l .ne. 0 ))then
                  rdot(2*(i-1)+2) = rdot(2*(i-1)+2)
     &                 + gamma(j)
     &                   *( r(2*(i-1)+1)
```

```
     &                      - ( 2 * k * xlength
     &                          + r(2*(j-1)+1)
     &                        )
     &                   )
     &                  /(
     &                      ( r(2*(i-1)+1)
     &                        - ( 2 * k * xlength
     &                            + r(2*(j-1)+1)
     &                          )
     &                      )**2.0d0
     &                      + ( r(2*(i-1)+2)
     &                          - ( 2 * l * ylength
     &                              + r(2*(j-1)+2)
     &                            )
     &                      )**2.0d0
     &                   )
c---  (( k .ne. 0 ) .or. ( l .ne. 0 ))  --------------------
                end if
c------------------------------------------------------------
c     end of interaction with other image vortices
c------------------------------------------------------------
c--------  l loop ------------------------------------------
               end do
c--------  k loop ------------------------------------------
            end do

c------------------------------------------------------------
c     end of vortex loops
c------------------------------------------------------------
c---  ( i .eq. vs .ne. j )  --------------------------------
            end if
c--------  j loop ------------------------------------------
         end do
c--------  i loop ------------------------------------------
      end do


C      mutltipols !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!



c------------------------------------------------------------
c     interaction with image vortices between -maxx and -5,
c     5 and maxx, -maxy and -5, and 5 and maxy. For faster
c     convergence only the quatrupol moments are included
c------------------------------------------------------------
c            do j = -maxx, maxx
```

```
c               do k = -maxy, maxy
c                  if ((j .lt. -5) .or. (j .gt. 5) .or.
c     &                 (k .lt. -5) .or. (k .gt. 5)) then
c                     denominator = 1/2.0d0*
c     &                 ((r(2*(i-1)+1)-2.0d0*j*xlength)**2.0d0
c     &                 +(r(2*(i-1)+2)-2.0d0*k*ylength)**2.0d0)
c     &                 **-3.0d0
c
c                     rdot(2*(i-1)+1)= rdot(2*(i-1)+1)
c     &                      +gamma(i)*8*r(2*(i-1)+1)*r(2*(i-1)+2)*
c     &                      ( 3*(r(2*(i-1)+1)+2*j*xlength)**2.0d0
c     &                        *(r(2*(i-1)+2)+2*k*ylength)
c     &                        -(r(2*(i-1)+2)+2*k*ylength)**3.0d0)
c     &                        *denominator
c
c                     rdot(2*(i-1)+2)= rdot(2*(i-1)+2) +
c     &                      gamma(i)*8*r(2*(i-1)+1)*r(2*(i-1)+2)*
c     &                      ( -3*(r(2*(i-1)+1)+2*j*xlength)
c     &                        *(r(2*(i-1)+2)+2*k*ylength)**2.0d0
c     &                        +(r(2*(i-1)+1)+2*j*xlength)**3.0d0)
c     &                        *denominator
c                  end if
c               end do
c            end do
c------------------------------------------------------------
c     end of the multipol loop
c------------------------------------------------------------
c         end do



c------------------------------------------------------------
c     end of fcnb
c------------------------------------------------------------
         end


c------------------------------------------------------------
c     Implements Jacobian  (optional) ...
c------------------------------------------------------------
      subroutine jac
         implicit    none

         include    'fcn.inc'

         return
      end
```

C sf.f

```
c==========================================================
c     The subroutine sf calculates the value of the stream
c     function on a grid with 2**level points and writes the
c     result to stdout
c==========================================================
      subroutine sf(npos, r)

      implicit     none

      include 'fcn.inc'

      integer      npos,    nvort
      real*8       r(npos)

      integer      i,       j,        k,        l
      integer      nx,      ny,       level
      parameter  ( level = 4 )

      real*8       hx,      hy
      real*8       sfv


      nvort = npos/2
      nx = 2**level + 1
      ny = 2**level + 1
      hx = xlength/nx
      hy = ylength/ny

c----------------------------------------------------------
c     free vortices
c----------------------------------------------------------
      If( walls .eq. 0 ) then
         do i = 0, nx
            do j = 0, ny
               sfv = 0
               do k = 1, nvort
                  if( ( i*hx - r(2*(k-1)+1) .ne. 0 ) .or.
     &                ( j*hy - r(2*(k-1)+2) .ne. 0 ) ) then
                     sfv = sfv
     &                  - gamma(k)
     &                    * log( ( i*hx - r(2*(k-1)+1)
     &                           )**2.0d0
     &                           + ( j*hy - r(2*(k-1)+2)
     &                             )**2.0d0
     &                         )
                  end if
               end do
```

```
c               write(*,*) i*hx, j*hy, sfv
               write(*,*) sfv
           end do
         end do
         write(*,*)


c-----------------------------------------------------------
c     one wall
c-----------------------------------------------------------
      else if( walls .eq. 1 ) then
         do i = 0, nx
            do j = 0, ny
               sfv = 0
               do k = 1, nvort
                  if( ( i*hx - r(2*(k-1)+1) .ne. 0 ) .or.
     &                ( i*hy - r(2*(k-1)+2) .ne. 0 ) ) then
                     sfv = sfv
     &                  - gamma(k)
     &                    * log( ( i*hx - r(2*(k-1)+1)
     &                           )**2.0d0
     &                           + ( j*hy - r(2*(k-1)+2)
     &                             )**2.0d0
     &                         )
                      sfv = sfv
     &                  + gamma(k)
     &                    * log( ( i*hx + r(2*(k-1)+1)
     &                           )**2.0d0
     &                           + ( j*hy - r(2*(k-1)+2)
     &                             )**2.0d0
     &                         )
                 end if
               end do
c               write(*,*) i*hx, j*hy, sfv
               write(*,*) sfv
            end do
         end do
         write(*,*)


c-----------------------------------------------------------
c     one corner
c-----------------------------------------------------------
      else if( walls .eq. 2 ) then
         do i = 0, nx
            do j = 0, ny
               sfv = 0
               do k = 1, nvort
                  if( ( i*hx - r(2*(k-1)+1) .ne. 0 ) .or.
```

```
     &                ( i*hy - r(2*(k-1)+2) .ne. 0 ) ) then
                     sfv = sfv
     &                 - gamma(k)
     &                   * log( ( i*hx - r(2*(k-1)+1)
     &                          )**2.0d0
     &                          + ( j*hy - r(2*(k-1)+2)
     &                            )**2.0d0
     &                        )
                      sfv = sfv
     &                 + gamma(k)
     &                   * log( ( i*hx + r(2*(k-1)+1)
     &                          )**2.0d0
     &                          + ( j*hy - r(2*(k-1)+2)
     &                            )**2.0d0
     &                        )
                      sfv = sfv
     &                 + gamma(k)
     &                   * log( ( i*hx - r(2*(k-1)+1)
     &                          )**2.0d0
     &                          + ( j*hy + r(2*(k-1)+2)
     &                            )**2.0d0
     &                        )
                      sfv = sfv
     &                 - gamma(k)
     &                   * log( ( i*hx + r(2*(k-1)+1)
     &                          )**2.0d0
     &                          + ( j*hy + r(2*(k-1)+2)
     &                            )**2.0d0
     &                        )
                 end if
               end do
c               write(*,*) i*hx, j*hy, sfv
               write(*,*) sfv
            end do
         end do
         write(*,*)


c------------------------------------------------------------
c     one two parallel walls
c------------------------------------------------------------
      else if( walls .eq. 3 ) then
         do i = 0, nx
            do j = 0, ny
               sfv = 0
               do k = 1, nvort
                  if( ( i*hx - r(2*(k-1)+1) .ne. 0 ) .or.
     &                ( i*hy - r(2*(k-1)+2) .ne. 0 ) ) then
                     sfv = sfv
```

```
     &                  - gamma(k)
     &                    * log( ( i*hx - r(2*(k-1)+1)
     &                           )**2.0d0
     &                           + ( j*hy - r(2*(k-1)+2)
     &                             )**2.0d0
     &                         )

                do l = -maxx, maxx
                  sfv = sfv
     &                  + gamma(k)
     &                    * log( ( i*hx
     &                             - ( 2 * l * xlength
     &                                 - r(2*(k-1)+1)
     &                               )
     &                           )**2.0d0
     &                           + ( j*hy - r(2*(k-1)+2)
     &                             )**2.0d0
     &                         )

                  sfv = sfv
     &                  + gamma(k)
     &                    * log( ( i*hx
     &                             - ( 2 * l * xlength
     &                                 - r(2*(k-1)+1)
     &                               )
     &                           )**2.0d0
     &                           + ( j*hy - r(2*(k-1)+2)
     &                             )**2.0d0
     &                         )
               end do
             end if
           end do
c           write(*,*) i*hx, j*hy, sfv
           write(*,*) sfv
        end do
     end do
     write(*,*)


c------------------------------------------------------------
c     box
c------------------------------------------------------------
      else if( walls .eq. 4 ) then
         do i = 0, nx
            do j = 0, ny
               sfv = 0
               do k = 1, nvort
                  if( ( i*hx - r(2*(k-1)+1) .ne. 0 ) .or.
     &                ( i*hy - r(2*(k-1)+2) .ne. 0 ) ) then
```

```
                 sfv = sfv
     &              - gamma(k)
     &                * log( ( i*hx - r(2*(k-1)+1)
     &                       )**2.0d0
     &                       + ( j*hy - r(2*(k-1)+2)
     &                         )**2.0d0
     &                     )

c-----------------------------------------------------------
c     influence of the image vortices
c-----------------------------------------------------------
                do l = -maxx, maxx
                  sfv = sfv
     &              + gamma(k)
     &                * log( ( i*hx
     &                         - ( 2 * l * xlength
     &                             - r(2*(k-1)+1)
     &                           )
     &                       )**2.0d0
     &                       + ( j*hy
     &                           - ( 2 * l * xlength
     &                               + r(2*(k-1)+2)
     &                             )
     &                         )**2.0d0
     &                     )


                  sfv = sfv
     &              - gamma(k)
     &                * log( ( i*hx
     &                         - ( 2 * l * xlength
     &                             - r(2*(k-1)+1)
     &                           )
     &                       )**2.0d0
     &                       + ( j*hy
     &                           - ( 2 * l * xlength
     &                               - r(2*(k-1)+2)
     &                             )
     &                         )**2.0d0
     &                     )


                  sfv = sfv
     &              + gamma(k)
     &                * log( ( i*hx
     &                         - ( 2 * l * xlength
     &                             + r(2*(k-1)+1)
     &                           )
     &                       )**2.0d0
```

```
     &                           + ( j*hy
     &                               - ( 2 * l * xlength
     &                                   - r(2*(k-1)+2)
     &                                 )
     &                             )**2.0d0
     &                         )


                     sfv = sfv
     &                 - gamma(k)
     &                   * log( ( i*hx
     &                             - ( 2 * l * xlength
     &                                 + r(2*(k-1)+1)
     &                               )
     &                           )**2.0d0
     &                           + ( j*hy
     &                               - ( 2 * l * xlength
     &                                   + r(2*(k-1)+2)
     &                                 )
     &                             )**2.0d0
     &                         )
                  end do
                end if
              end do
c              write(*,*) i*hx, j*hy, sfv
              write(*,*) sfv
           end do
        end do
        write(*,*)

c------------------------------------------------------------
c     end of If walls ...
c------------------------------------------------------------
      end if

c------------------------------------------------------------
c     end of subroutine sf
c------------------------------------------------------------
      end
```

D dvmat.f

```
c============================================================
c
c     Subroutine, which reads several values from on line
c     and returns the values in a 2 dim array. The idea of
c     this subroutine is based on dvfrom.f by M. Choptuik
```

```
c
c     In the first line of the file must(!) consist of the
c     dimesions of data to read in. The first integer nlines
c     is the total number of lines in the file without(!)
c     the line of the dimensions. The second integer has
c     the value of the number of values per line in the file
c
c     <nlines> <ncolumns>
c     <data>
c
c     Example:
c
c     2 3
c     -1.0 0.01 3.0
c     0.1 2.02 0.33
c
c     Fortran reserves the following unit numbers:
c
c     5      terminal input (stdin)
c     6      terminal output (stdout)
c     0      error output on Unix systems (stderr)
c==========================================================

      subroutine dvmat(fname,mat,nlines,ncol,maxlines,maxcol)

      implicit    none

c----------------------------------------------------------
c        The integer functions 'indlnb' and 'getu' are
c        defined in the 'p329f' library.
c----------------------------------------------------------
         integer          indlnb,      getu
c----------------------------------------------------------
c     fname    :  name of the input file. If fname has the
c                 value of '-' the input is red from stdin
c     mat      :  2 dim output array
c     nlines   :  returns the number of lines
c     ncol     :  returns the number of elements in each line
c     maxlines :  maximum of lines
c     maxcol   :  maximum of elements in on line
c----------------------------------------------------------

      character*(*)    fname
      integer          nlines,       ncol,      maxlines
      integer          maxcol
      integer          i, j
      real*8           mat(maxlines, maxcol),   line(maxcol)

      integer          inunit,       error,     rc
```

```
      integer          ustdin
      parameter      ( ustdin = 5 )
      integer          iline,        icol

c-----------------------------------------------------------
c     set the input unit
c-----------------------------------------------------------
      if( fname .eq. '-' ) then
         inunit = ustdin
      else
c-----------------------------------------------------------
c           Get an available unit number
c-----------------------------------------------------------
            inunit = getu()
c-----------------------------------------------------------
c           Open the file for formatted I/O
c-----------------------------------------------------------
            open(inunit,file=fname(1:indlnb(fname)),
     &           form='formatted',status='old',iostat=rc)
            if( rc .ne. 0 ) then
c-----------------------------------------------------------
c              Couldn't open the file, print error message
c              and return.
c-----------------------------------------------------------
               write(0,*) 'dvfrom: Error opening ',
     &                    fname(1:indlnb(fname))
               return
            end if
      end if
c-----------------------------------------------------------
c     read in the dimensions of the matrix
c-----------------------------------------------------------
      read(inunit,*,iostat=error, end=200) nlines, ncol
      If( nlines .gt. maxlines ) then
         write(0,*)
         write(0,*) '   To many lines. This program can only '
     &        //'handle ',maxlines
         write(0,*)
         stop
      else if( ncol .gt. maxcol ) then
         write(0,*)
         write(0,*) '   To many columns. This program can only '
     &        //'handle ',maxcol
         write(0,*)
         stop
      end if
c-----------------------------------------------------------
c     read in the entries of the matrix
c-----------------------------------------------------------
```

```
      do i = 1, nlines
         read(inunit,*,iostat=error, end=200)
     &        (line(j), j=1, ncol)
         if( error .ne. 0 ) then
            write(0,*)
            write(0,*) '   An error occured while reading '
     &           // 'the matrix entries.'
            write(0,*) '   Please check your data file.'
            write(0,*)
            stop
         end if
         do j = 1, ncol
            mat(i,j) = line(j)
         end do
      end do
c------------------------------------------------------------
c     end of read sequence
c------------------------------------------------------------

 200     continue
c------------------------------------------------------------
c        If we are reading from a file, close the file.
c        This releases the unit number for subsequent use.
c------------------------------------------------------------
         if( inunit .ne. ustdin ) then
            close(ustdin)
         end if

      return

      end
```

E Makefile