Coordinate transformations and two exact solutions of the axisymmetric Navier-Stokes equations

Les Hatton
Computing Laboratory, University of Kent*

October 5, 2003

Abstract

An existing separable solution of the full unsteady axisymmetric Navier-Stokes equations is presented in a form which can be repeated using open source or widely available numerical libraries. A new separable solution of the same equations is also presented and investigated numerically using shooting methods.

`$Date: 2003/10/01 14:06:04 $`

1 Overview

Coordinate transformations are one of the most elegant concepts in physics. The mathematical description of a problem in one coordinate system can be extraordinarily complex and yet with a coordinate transformation, a very simple equation can result. Of course, such equations can often by solved whereas the initial equation can not. Examples abound. On a very simple level, consider the difference between the representation of a circle in Cartesian coordinates:-

$$x^2 + y^2 = a^2 \tag{1}$$

compared with its polar representation:-

$$r = a \tag{2}$$

Moving up the scale of complexity, the Joukowski transformation [1] transforms an aerofoil shape into a circle. The problem of calculating fluid flow around an aerofoil is then reduced to the problem of calculating the flow around a circle which is much simpler.

*L.Hatton@kent.ac.uk, lesh@oakcomp.co.uk

2 Transforming the axisymmetric Navier-Stokes equations

The axisymmetric Navier-Stokes equations are a formidably complex set of equations which govern the movement of a fluid. They are used to model all kinds of vortex flow such as occur in aerodynamics and meteorology. There are very few exact solutions known and the solution space normally has to be explored by numerical methods on the full set of equations, itself a very difficult problem. For a velocity field:-

$$u = u(r,z,t); v = v(r,z,t); w = z\varpi(r,z,t) \tag{3}$$

where u is the radial velocity, v is the swirl velocity, w is the axial velocity and r and t are the radius and time respectively, the full unsteady, viscous, axisymmetric Navier-Stokes equations are:-

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial r} + w\frac{\partial u}{\partial z} - \frac{v^2}{r} = -\frac{1}{p}\frac{\partial p}{\partial r} + \nu\{\frac{\partial}{\partial r}(\frac{1}{r}\frac{\partial(ru)}{\partial r}) + \frac{\partial^2 u}{\partial z^2}\} \tag{4}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial r} + w\frac{\partial v}{\partial z} + \frac{uv}{r} = -\nu\{\frac{\partial}{\partial r}(\frac{1}{r}\frac{\partial(rv)}{\partial r}) + \frac{\partial^2 v}{\partial z^2}\} \tag{5}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial r} + w\frac{\partial w}{\partial z} - \frac{v^2}{r} = -\frac{1}{p}\frac{\partial p}{\partial z} + \nu\{\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial(w)}{\partial r}) + \frac{\partial^2 w}{\partial z^2}\} \tag{6}$$

The continuity equation, (fluid cannot disappear) is represented by:-

$$\frac{\partial u}{\partial r} + \frac{u}{r} + \frac{\partial w}{\partial z} = 0 \tag{7}$$

As is the usual practice, the axisymmetric nature of the equations allows us to rewrite them in terms of two dependent variables, the stream function Ψ and the circulation function K, defined by

$$(u,v,w) = \frac{1}{r}(-\frac{\partial\Psi}{\partial z}, K, \frac{\partial\Psi}{\partial r}) \tag{8}$$

Eliminating the pressure p from the radial and axial equations above, gives the following two equations:-

$$-\frac{1}{r}D^2(\frac{\partial\Psi}{\partial t}) + \frac{\partial(\frac{1}{r^2}D^2\Psi,\Psi))}{\partial(r,z)} - \frac{1}{r^3}\frac{\partial K^2}{\partial z} = -\frac{\nu}{r}D^4\Psi \tag{9}$$

where

$$\frac{\partial(f,g)}{\partial(x,y)} = (\frac{\partial f}{\partial x})_y(\frac{\partial g}{\partial y})_x - (\frac{\partial f}{\partial y})_x(\frac{\partial g}{\partial x})_y \tag{10}$$

and

$$\frac{\partial K}{\partial t} - \frac{1}{r}\frac{\partial(K,g\Psi)}{\partial(r,z)} = \nu D^2 K \tag{11}$$

3 Transforming the axisymmetric Navier-Stokes equations

The point of the earlier section on coordinate simplification now becomes apparent. The formidably complex equations of the previous section can be reduced to coupled ordinary differential equations using the following transformations, [2], [3].

$$\eta = \frac{r^2}{4\nu t}; \xi = \frac{z}{2\sqrt{\nu t}} \tag{12}$$

$$\frac{\Psi}{4\nu\sqrt{\nu t}} = f(\eta)F(\xi); \frac{K}{K_c} = h(\eta)H(\xi) \tag{13}$$

where K_c is a constant with the dimensions of circulation.

Two exactly separable solutions now present themselves.

3.1 Solution 1

We let f = $-\eta$ and h = η. The full N-S equations of the previous section then reduce after one integration to the set of coupled ordinary differential equations

$$\frac{d^3F}{d\xi^3} + 2\xi\frac{d^2F}{d\xi^2} + 4\frac{dF}{d\xi} + 4F\frac{d^2F}{d\xi^2} - 2(\frac{dF}{d\xi})^2 + \frac{\Omega^2H^2}{2} = 2 + \frac{\Omega^2}{2} \tag{14}$$

and

$$\frac{d^2H}{d\xi^2} + 2\xi\frac{dH}{d\xi} + 4H = 4(H\frac{dF}{d\xi} - F\frac{dH}{d\xi}) \tag{15}$$

where $\frac{\Omega}{4t}$ is the angular velocity. The resulting velocity field in the fluid is given by

$$u = \frac{r}{2t}\frac{dF(\xi)}{d\xi}; v = \frac{\Omega r}{4t}H(\xi); w = -2\sqrt{\frac{\nu}{t}}F(\xi) \tag{16}$$

These equations are easy to solve using shooting methods and were solved by [3] and the solutions prove useful in modelling the air flow in the centre of an intense vortex such as a tornado in contact with a non-slipping surface. One important prediction is the existence of an axial stagnation point aloft where the axial flow reverses. This has been observed in a number of tornadoes. The relevant boundary conditions are

$$F(0) = (\frac{dF}{d\xi})_{\xi=0} = H(0) = 0; F(\xi) \longrightarrow \xi - C, H(\xi) \longrightarrow 1, \xi \longrightarrow \infty \tag{17}$$

Using the notation F" = $\frac{d^2F}{d\xi^2}$, Figure 1 shows F"(0) and H'(0) for a range of Ω^2. The corresponding velocity fields are shown in Figure 2.

3.2 Solution 2

There is a second solution to the full equations. These are given by choosing F = ξ and H = ξ in the original equations and lead to the following solution

$$2\eta^3\frac{d^4f}{d\eta^4} + 2\eta^2(\eta + 2)\frac{d^3f}{d\eta^3} + 2\eta^2(f\frac{d^3f}{d\eta^3} - f\frac{d^2f}{d\eta^2}) + 4\eta^2\frac{d^3f}{d\eta^3} = \frac{\Omega^2h^2}{8} \tag{18}$$

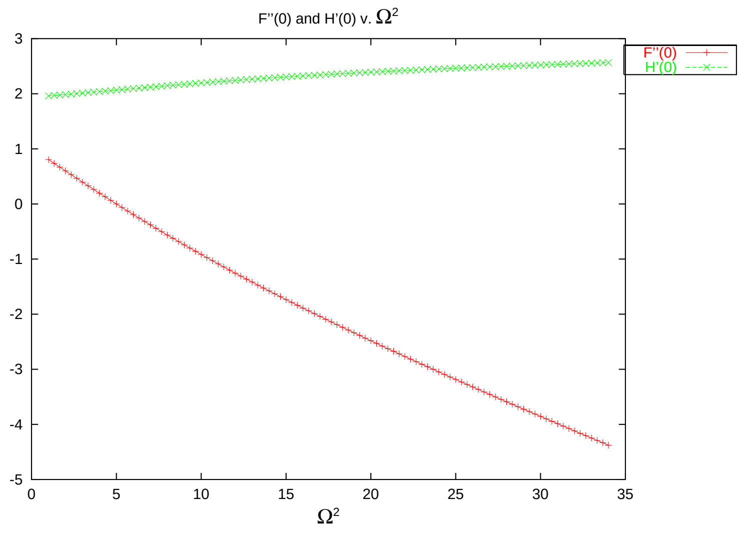

Figure 1: A plot of F”(0) and H’(0) against Ω^2

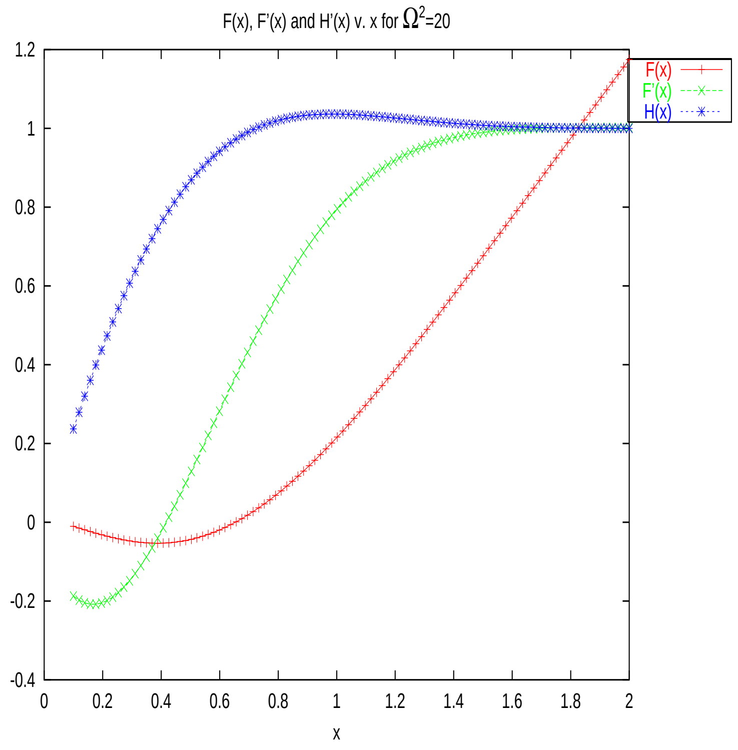

Figure 2: Combined plot of F’(x) (radial), H’(x) (tangential) and F(x) (axial) velocity fields.

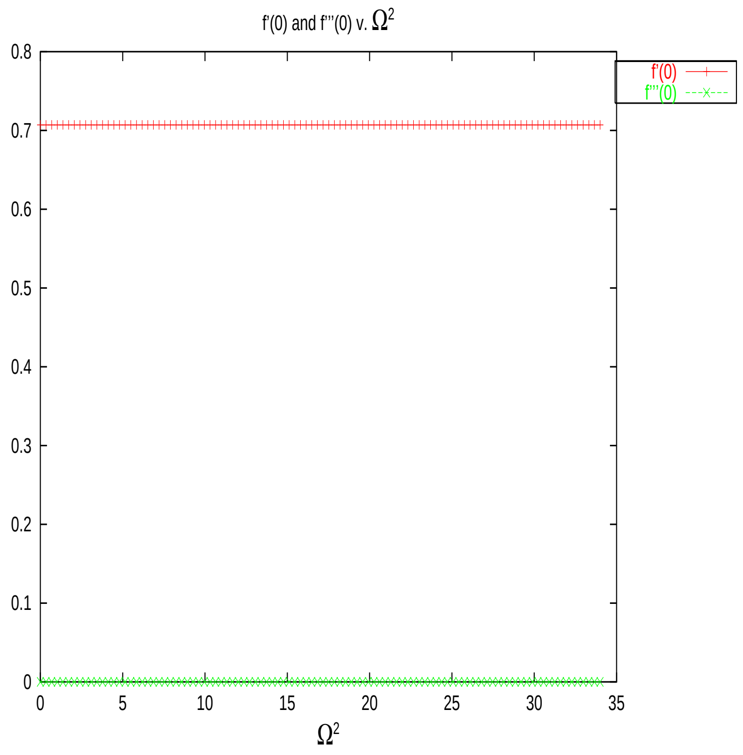

Figure 3: A plot of f'(0) and f"'(0) against Ω^2

and

$$\eta\frac{d^2h}{d\eta^2} + (\eta + f)\frac{dh}{d\eta} = (\frac{df}{d\eta} - \frac{1}{2})h \tag{19}$$

The resulting velocity field has the form

$$u = -\frac{2\nu}{r}f(\eta); v = \frac{K_c z}{r\sqrt{\nu t}}h(\eta); w = \nu z\frac{df}{d\eta} \tag{20}$$

3.2.1 Boundary conditions for solution 2

At the axis, f(0) = h(0) = 0 for zero axial and radial flow. In addition, we require that $\frac{\partial w}{\partial r} = 0$ which gives f"(0) = 0. Away from the axis, we have various choices most of which lead to trivial flows, but an example of a non-trivial although very simple flow is one for which (u,w) are independent of ν as $\nu \longrightarrow \infty$. The boundary conditions used are $f'(\eta) \longrightarrow \frac{1}{\eta}, f'''(\eta) \longrightarrow 0, h'(\eta) \longrightarrow 0$. At the time of writing, no more interesting flows had been found.

The resulting solutions are shown as Figure 3 and the corresponding flow patter in Figure 4.

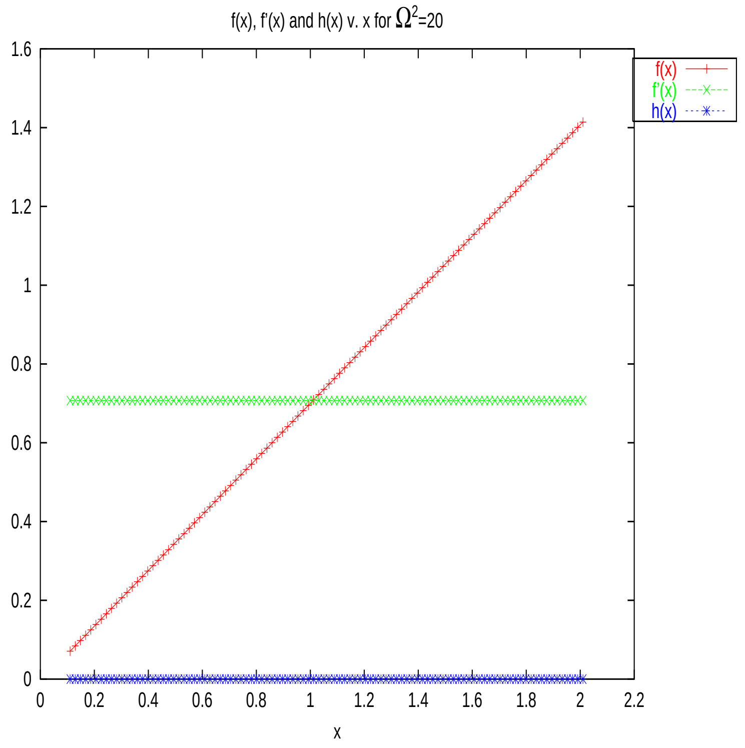

Figure 4: Combined plot of f(x) (radial), h(x) (tangential) and f'(x) (axial) velocity fields.

References

[1] Curle N. and Davies H.J. (1968) *Modern Fluid Dynamics Vol 1: incompressible flow* Van Nostrand, London.

[2] Bellamy-Knights P.B.K. (1974) *An axisymmetric boundary layer solution for an unsteady vortex above a plane* Tellus XXVI.

[3] Hatton L. (1975) *Stagnation point flow in a vortex core* Tellus XXVII.

[4] Press, W.H., Teukolsky, S.A, Vettering, W.T., Flanner, B.P. (1999) *Numerical Recipes in C: second edition* Cambridge University Press, ISBN 0-521-43108-5

A Appendix: driver and plotting programs

So that these results can be independently repeated, the driver programs are shown here. These were run against Numerical Recipes C library v. 2.08, [4] on a SuSE Linux 8.2 machine and the results plotted using gnuplot, (http://www.gnuplot.org). The plotting program is also shown. This produces four enhanced postscript files which appear in this paper as Figs 1-4.

A.0.2 Driver program for solution 1

```
/*
 *   Program in C to simulate the results of solution 1.
 *
 *   Revision: $Revision: 1.1 $
 *   Date:     $Date: 2003/10/01 14:06:04 $
 */
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define   EPS       1.0E-6
#define   GNUPLOT_SEPARATOR   printf("\n\n");
/*
 *   Pull in the Numerical Recipes libraries.
 */
#include "nrutil.h"
#include "nr.h"
/*
 *   Variables local to my equations.
 */
float     omegasq;

#define   MAXPLOT        10000
float     ind1[MAXPLOT];
float     dep1[MAXPLOT];
float     dep2[MAXPLOT];
float     dep3[MAXPLOT];
```

```
int       nplot;              /*   Index into plotted arrays     */
/*
 *   Communication variables for shooting.
 */
int       nvar = 5;      /*   Number of equations in equivalent  */
                         /*   first order system.                */
float     x1   = 0.0;    /*   Start of integration               */
float     x2   = 2.0;    /*   End of integration                 */
/*------------------------------------------------------------------
 *   Transformation used in the user-supplied functions, derivs,
 *   load and score.
 *
 *   Let y1 = F, y2 = F', y3 = F'', y4 = H, y5 = H'.
 *------------------------------------------------------------------*/
/*
 *   First of all transform the equations into a set of first order
 *   equations for the format required by the shooter.
 */
void
derivs(
     float     x,
     float     y[],
     float     dydx[]
)
{
/*
 *   The equations are, (p.56 of Ph.D):-
 *
 *   F''' + 2xF'' + 4F' + 4FF'' - 2(F')**2 + 0.5 *(WH)**2 = 2 + 0.5W**2
 *   H'' + 2xH' + 4H = 4(F'H - FH')
 */
     dydx[1]   = y[2];
     dydx[2]   = y[3];
     dydx[3]   = - 2.0 * x * y[3] - 4.0 * y[2] - 4.0 * y[1] * y[3]
                      + 2.0 * y[2] * y[2] - 0.5 * omegasq * y[4] * y[4]
                      + 2.0 + 0.5 * omegasq;
     dydx[4]   = y[5];
     dydx[5]   = -2.0 * x * y[5] - 4.0 * y[4] + 4.0 * y[2] * y[4]
                      - 4.0 * y[1] * y[5];
}
/*------------------------------------------------------------------*/
/*
 *   Supply the boundary conditions at the start point, in this case
 *   x = 0.
 *
 *   The start boundary conditions are, (p.55 of Ph.D):-
 *
 *   F(0) = F'(0) = H(0) = 0
 */
void
```

```
load(
     float     xs,
     float     v[],
     float     y[]
)
{
/*
 *   These are easy.
 */
     y[1]      = 0.0;         /*   F(0)      = 0.0     */
     y[2]      = 0.0;         /*   F'(0)     = 0.0     */
     y[4]      = 0.0;         /*   H(0)      = 0.0     */
/*
 *   These are parameterised from the v vector.
 */
     y[3]      = v[1];        /*   F''(0)    = ?       */
     y[5]      = v[2];        /*   H'(0)     = ?       */
}
/*------------------------------------------------------------------*/
/*
 *   Test the parameterised start values against their values
 *   at the end point.
 *
 *   The end boundary conditions are, (p.55 of Ph.D):-
 *
 *   F -> x -C, H-> 1 as x -> inf
 *   so F''(x) -> 0, H'(0) -> 0 as x -> inf
 */
void
score(
     float     xf,
     float     y[],
     float     f[]
)
{
/*
 *   f[1] corresponds to v[1] which is the behaviour of F''.
 *   f[2] corresponds to v[2] which is the behaviour of H'.
 *
 *   The f[] is a discrepancy vector.  The shooting algorithm forces
 *   the elements of f[] towards zero so that the discrepancy is zero.
 *   Hence if we simply put in the function values here, the statement
 *   f[1] = y[3] will converge when y[3] is closest to zero which is the
 *   required behaviour for y[3] = F'' as x -> inf.
 */
     f[1]      = y[3];
     f[2]      = y[5];
}
/*------------------------------------------------------------------*/
void
```

```
gnuplot_out(
     char      *string,
     float     x[],
     float     y[]
)
{
     int                 i;

     printf("%s\n", string);

     for( i = 0; i < nplot; ++i )
     {
          printf("%g\t%g\n", x[i], y[i] );
     }
}
/*-------------------------------------------------------------------*/
int
main(void)
{
     const     int       n2 = 2;
     float               *v;
     float               *ystart;
     int                 check = 0;
     float               dx = 0.1;
     float               h1;
     float               x;
     float               hmin = 0.0;
     int                 nok, nbad;
/*
 *   Create the disposable vector.
 */
     v         = vector( 1, n2 );
/*
 *   Part 1: Find F''(0) and H'(0) v. omegasq.
 *   ==========================================
 *   First guess the start values for v[1] = F'' and v2 = H'.
 */
     v[1]      = 1.0;
     v[2]      = 2.0;
/*
 *   For loop on real variables, Yuk.
 */
     nplot     = 0;

     for( omegasq = 0.0; omegasq < 35.0; omegasq += 1.0 )
     {
          newt( v, n2, &check, shoot );

          if ( check )
          {
                printf("# ... did NOT converge\n");
            }
            else
            {
/*
 *              Store the values in the plot arrays.
 */
                dep1[nplot]    = v[1];
                dep2[nplot]    = v[2];
                ind1[nplot++]  = omegasq;

                if ( nplot == MAXPLOT )  break;
            }
        }
/*
 *   Output the results in gnuplot format.  These have to
 *   be done one after the other.
 */
     gnuplot_out( "# Omega**2  F''(0)", ind1, dep1 );

     GNUPLOT_SEPARATOR;

     gnuplot_out( "# Omega**2  H'(0)", ind1, dep2 );
/*
 *   Part 2: Integrate axial, radial and tangential velocity fields
 *   for 1 value of omega squared where there is an axial stagnation
 *   point aloft.
 */
     omegasq   = 20.0;
     ystart    = vector( 1, nvar );

     nplot     = 0;
/*
 *   Shoot to get the starting values.
 */
     newt( v, n2, &check, shoot );

     load( 0.0, v, ystart );

     h1   = (x2-x1) * .01;
/*
 *   Now integrate out step by step.
 */
     for( x = x1; x < x2; x += dx )
     {
          odeint( ystart, nvar, x, x+dx, EPS, h1, hmin,
                    &nok, &nbad, derivs, rkqs );

          dep1[nplot]    = ystart[1];
          dep2[nplot]    = ystart[2];
          dep3[nplot]    = ystart[4];
          ind1[nplot++]  = x+dx;

          if ( nplot == MAXPLOT )  break;
     }

     GNUPLOT_SEPARATOR;

     gnuplot_out( "# x  F(x)", ind1, dep1 );

     GNUPLOT_SEPARATOR;

     gnuplot_out( "# x  F'(x)", ind1, dep2 );

     GNUPLOT_SEPARATOR;

     gnuplot_out( "# x  H(x)", ind1, dep3 );

     free_vector(ystart, 1, nvar );
     free_vector(v, 1, n2 );

     exit(EXIT_SUCCESS);
}
```

A.0.3 Driver program for solution 2

The only significant differences were in the setup functions which are shown below.

```
/*
 *   Program in C to simulate the results of solution 2.
 *
 *   Revision: $Revision: 1.1 $
 *   Date:     $Date: 2003/10/01 14:06:04 $
 */
...
int       nvar = 6;      /*   Number of equations in equivalent  */
                         /*   first order system.                */

...
/*-----------------------------------------------------------------
 *   Transformation used in the user-supplied functions, derivs,
 *   load and score.
 *
 *   Let y1 = F, y2 = F', y3 = F'', y4 = H, y5 = H'.
 *-----------------------------------------------------------------*/
/*
 *   First of all transform the equations into a set of first order
 *   equations for the format required by the shooter.
 */
```

```
void
derivs(
     float     x,
     float     y[],
     float     dydx[]
)
{
/*
 *   The equations are, (p.74 of Ph.D):-
 *
 *   2x**3f'''' + 2x**2(x+2)f''' + 2x**2(ff'''-f'f'') + 4x**2f''= W**2/8h**2
 *   xh'' + xh' + 0.5h = hf' -h'f
 */
     dydx[1]   = y[2];
     dydx[2]   = y[3];
     dydx[3]   = y[4];
     dydx[4]   = - ((x+2) * y[4])/x - (y[1]*y[4] - y[1]*y[3])/x - 2.0*y[3]/x
                    + (omegasq * y[5] * y[5])/(16.0*pow(x,3.0));
     dydx[5]   = y[6];
     dydx[6]   = - y[6] - (y[1] * y[6])/x + ((y[2] - 0.5) * y[5])/x;
}
/*------------------------------------------------------------------*/
/*
 *   Supply the boundary conditions at the start point, in this case
 *   x = 0.
 *
 *   The start boundary conditions are, (p.74 of Ph.D):-
 *
 *   f(0) = f''(0) = h(0) = 0
 *   (f''(0) corresponds to dw/dr = 0 on axis.)
 */
void
load(
     float     xs,
     float     v[],
     float     y[]
)
{
/*
 *   These are easy.
 */
     y[1]      = 0.0;         /*   f(0)      = 0.0     */
     y[3]      = 0.0;         /*   f''(0)    = 0.0     */
     y[5]      = 0.0;         /*   h(0)      = 0.0     */
/*
 *   These are parameterised from the v vector.
 */
     y[2]      = v[1];        /*   f'(0)     = ?       */
     y[4]      = v[2];        /*   f'''(0)   = ?       */
     y[6]      = v[3];        /*   h'(0)     = ?       */
}
/*----------------------------------------------------------------*/
/*
 *   Test the parameterised start values against their values
 *   at the end point.
 *
 *   The end boundary conditions are, (p.55 of Ph.D):-
 *
 *   F -> x -C, H-> 1 as x -> inf
 *   so F''(x) -> 0, H'(0) -> 0 as x -> inf
 */
void
score(
     float     xf,
     float     y[],
     float     f[]
)
{
/*
 *   f[1] corresponds to v[1] which is the behaviour of f'.
 *   f[2] corresponds to v[2] which is the behaviour of f'''.
 *   f[3] corresponds to v[3] which is the behaviour of h'.
 *
 *   The f[] is a discrepancy vector.  The shooting algorithm forces
 *   the elements of f[] towards zero so that the discrepancy is zero.
 *   Hence if we simply put in the function values here, the statement
 *   f[1] = y[3] will converge when y[3] is closest to zero which is the
 *   required behaviour for y[3] = F'' as x -> inf.
 */
     f[1]      = y[2] - 1.0 / sqrt(xf);
     f[2]      = y[4];
     f[3]      = y[6];
}
...
```

A.0.4 Plotting program for both solutions

The gnuplot program was used to produce Figs 1-4. The program is shown below.

```
#
#    Gnuplot script to make all figures for this project.
#
#    Revision: $Revision: 1.1 $
#    Date:     $Date: 2003/10/01 14:06:04 $
#-----------------------------------------------------------------------
#
#    First create the combined diagram of F''(0) and H'(0) for solution 1.
#    ==================================================================
#
```

```
#    Divert the output into the eps file.
#
set  term      postscript eps enhanced color
set  output    "fig1.eps"
#
#    Annotate.  To get fancy symbols you have to use the enhanced mode
#    in postscript.
#
#    Note the set-show aspect of gnuplot.  If you don't show it you
#    don't see it.
#
set  xlabel    '{/Symbol=20 W}^2'
show xlabel

set  title     "F''(0) and H'(0) v. {/Symbol=20 W}^2"
show title
#
#    Ask for a legend.  The titles in the legend are set by the titles in
#    the plot command which follows.
#
set  key       outside box
show key

plot 'sol1.dat' index 0 smooth bezier title "F''(0)" with linespoints, '' index 1 smooth b

show tics

#
#    Close the file.
#
set  output
#
#    Now create the combined flow diagram.
#    ===================================
#
#    Divert the output into the eps file.
#
set  output    "fig2.eps"

#
#    Annotate.  To get fancy symbols you have to use the enhanced mode
#    in postscript.
#
#    Note the set-show aspect of gnuplot.  If you don't show it you
#    don't see it.
#
set  xlabel    'x'
show xlabel

set  title     "F(x), F'(x) and H'(x) v. x for {/Symbol=20 W}^2=20"
```

```
show title
#
#    Ask for a legend.  The titles in the legend are set by the titles in
#    the plot command which follows.
#
set  key       outside box
show key

plot 'sol1.dat' index 2 smooth bezier title "F(x)" with linespoints, '' index 3
smooth bezier title "F'(x)" with linespoints, '' index 4 smooth bezier title "H(x)"
with linespoints

show tics

#
#    Close the file.
#
set  output
#
#    First create the combined diagram of F''(0) and H'(0) for solution 2.
#    ==================================================================
#
#    Divert the output into the eps file.
#
set  term      postscript eps enhanced color
set  output    "fig3.eps"
#
#    Annotate.  To get fancy symbols you have to use the enhanced mode
#    in postscript.
#
#    Note the set-show aspect of gnuplot.  If you don't show it you
#    don't see it.
#
set  xlabel    '{/Symbol=20 W}^2'
show xlabel

set  title     "f'(0) and f'''(0) v. {/Symbol=20 W}^2"
show title
#
#    Ask for a legend.  The titles in the legend are set by the titles in
#    the plot command which follows.
#
set  key       outside box
show key

plot 'sol2.dat' index 0 smooth bezier title "f'(0)" with linespoints, '' index 1
smooth bezier title "f'''(0)" with linespoints

show tics
```

```
#
#    Close the file.
#
set  output
#
#    Now create the combined flow diagram.
#    ====================================
#
#    Divert the output into the eps file.
#
set  output    "fig4.eps"

#
#    Annotate.  To get fancy symbols you have to use the enhanced mode
#    in postscript.
#
#    Note the set-show aspect of gnuplot.  If you don't show it you
#    don't see it.
#
set  xlabel    'x'
show xlabel

set  title     "f(x), f'(x) and h(x) v. x for {/Symbol=20 W}^2=20"
show title
#
#    Ask for a legend.  The titles in the legend are set by the titles in
#    the plot command which follows.
#
set  key       outside box
show key

plot 'sol2.dat' index 2 smooth bezier title "f(x)" with linespoints, '' index 3
smooth bezier title "f'(x)" with linespoints, '' index 4 smooth bezier title "h(x)"
with linespoints

show tics

#
#    Close the file.
#
set  output
```