Incompressible, Irrotational Fluid Flow Around A Cylinder In A Duct

An application of finite element methods

This experiment concerned the accurate simulation, via the finite element method, of the behaviour of an incompressible irrotational fluid flow around a cylinder in a duct. Our aims were to find the conditions under which the grid performs reasonably well and then to examine how the width of the duct affects the flow (in comparison to the analytical solution for an isolated cylinder). It was discovered that while a reasonable mesh can be easily defined, a really accurate mesh is difficult to predict. Also, the simulation was found to fail for high and low duct widths, and ways to improve the overall performance of the simulation are discussed.

Andrew N Jackson

15th November 1995

Introduction

This experiment aims to model a simple physical problem using finite element methods, thus giving an example of the implementation such methods require whilst showing what kind of problem-dependant refinements can be made to the general method to make the solution more accurate.

The problem we shall consider here is that of ideal flow around a cylinder in a duct (see fig. 1-1).

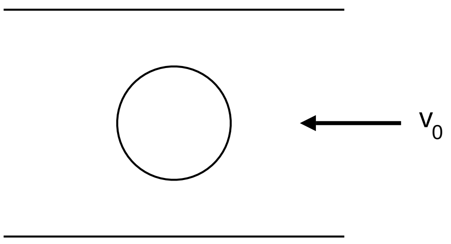

Figure 1-1

In this simple example, ideal flow means the flow of an incompressible and irrotational fluid, giving rise to symmetric flow over the cylinder (no turbulence can form within an ideal fluid such as this). We can describe the flow either by means of a velocity potential,ϕ , or by the stream function, φ, both of which satisfy Laplace's equation:

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

for ideal flow, subject to the appropriate boundary conditions. Either approach can be used, but in this experiment we shall calculate the velocity potential.

The finite element approach works by breaking the two dimensional structure in figure 1-1 down into a number of elements for each of which we shall calculate the velocity potential, ie one value of the velocity potential applies to the whole of the area an element occupies. The number of elements we use and the way they are arranged has a crucial effect on the success of the model, and the first aim of this experiment is to identify a reasonable working grid for the problem. The second aim is to analyse the effect which varying the duct width has on the fluid flow, which can then be compared to the analytical result for fluid flow over a cylinder in the absence of a duct.

Theory

The theory for the implementation of the finite element method, in this case the Galerkin residual method, can be broken down into six steps, each of which is detailed below. Each step will be explained both in general and in terms of how it applies to this specific problem.

Step 1 • Discretisation of the Structure

As we break the structure down for simulation, it is important that we should allow the grid to follow the physical form of the domain as closely as we can, as a more accurate definition of the problem will give us a better solution. In the case of our problem, it is clear that the part of the flow around the cylinder would be best described by polar coordinates, while the duct flow would be better described in cartesian coordinates. These two coordinate systems are combined in our mesh to give the general form shown in the schematic below (fig. 2-1).

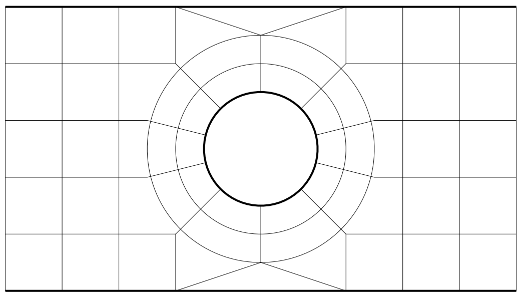
Figure 2-1

The schematic above is only intended to portray the general form of the grid, it's exact form will be defined in the next step.

As mentioned before, our irrotational flow is left-right symmetric, and as we can see the problem is also top-bottom symmetric. This means that we only need to calculate the velocity potential for one quarter of the domain. For our calculation we shall use the top-right hand quarter of the problem domain.

Step 2 • Identifying the Interpolation Model

The field variable for an element is calculated by interpolation of the nodal values, and the interpolation model we wish to use defines the number of nodes each element requires. In this experiment we will use the simplex form, which requires only a linear interpolation. This means that in 2-D each element just requires three points for interpolation, so we will be using the simplest 2-D surface unit, the triangle. This means the grid take the form shown in figure 2-2.

[See overleaf]

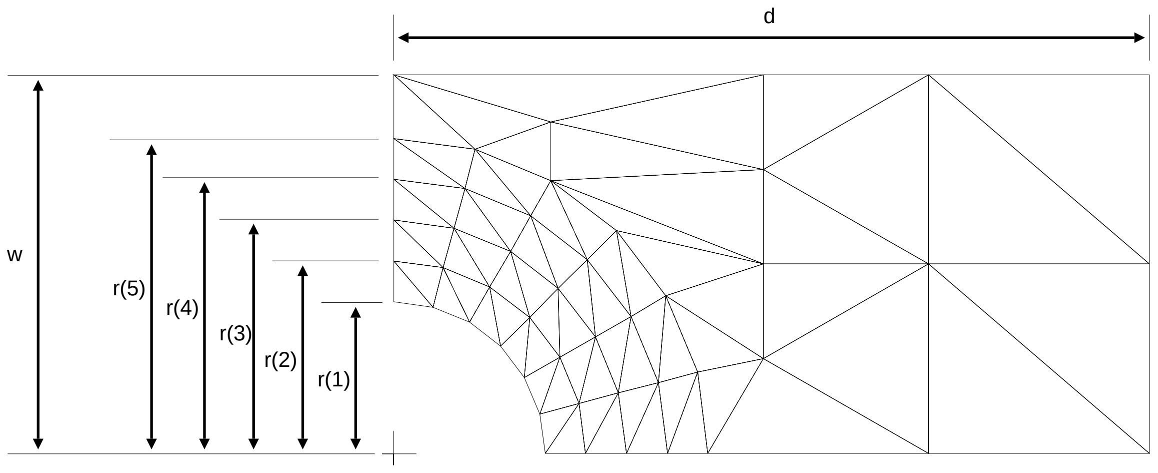

Figure 2-2

Once the domain of the problem is described like this, any particular problem can be defined by the 7 parameters indicated above (ie r(1-5), w, d). The only other parameter required to completely define the problem is the velocity of the fluid as it comes into the duct.

Now our interpolation model has been chosen, the value of the velocity potential for any particular element with 3 nodes is:

$$\phi^{(e)}(x,y)=\sum_{i=1,3} N_i(x,y)\,\phi_i^{(e)} \qquad \text{(i)}$$

Where N refers to the interpolation functions and ϕ refers to the nodal points. If the elements nodal points i=1,2,3 refers to the domain nodal points i,j,k, then:

$$\underline{N}(x,y)=\begin{bmatrix} N_i(x,y) \\ N_j(x,y) \\ N_k(x,y) \end{bmatrix}^T = \begin{bmatrix} (a_i+b_ix+c_iy)/2A^{(e)} \\ (a_j+b_jx+c_jy)/2A^{(e)} \\ (a_k+b_kx+c_ky)/2A^{(e)} \end{bmatrix}^T \qquad \text{(ii)}$$

and

$$\phi^{(e)}=\begin{bmatrix} \phi_i \\ \phi_j \\ \phi_k \end{bmatrix} \qquad \text{(iii)}$$

where

$$a_i = x_jy_k - x_ky_j \;:\; b_i = y_j - y_k \;:\; c_i = x_k - x_j$$
$$a_j = x_ky_i - x_iy_k \;:\; b_j = y_k - y_i \;:\; c_j = x_i - x_k$$
$$a_k = x_iy_j - x_jy_i \;:\; b_k = y_i - y_j \;:\; c_k = x_j - x_i$$

and the element area

$$A^{(e)} = {}^1/_2\,(x_iy_j + x_jy_k + x_ky_i - x_iy_k - x_jy_i - x_ky_i)$$

In other words the spacial interpolation function is defined by the ratio of the nodewise area to the total element area.

Step 3 • Defining the Problem in Matrix Form

In order to solve our Laplace equation, we must express it in finite element terms and then integrate:

$$\iint_{S^{(e)}} N_i \nabla^2 \phi_i^{(e)} \, dS = 0 \tag{iv}$$

Integrating by parts

$$\iint_{S^{(e)}} N_i \nabla^2 \phi_i^{(e)} \, dS = -\iint_{S^{(e)}} \nabla\phi_i^{(e)}.\nabla N_i \, dS + \int_{C^{(e)}} N_i \nabla\phi_i^{(e)} \, d\underline{C} \tag{v}$$

The integration is carried out to reduce the order of the equation. This is necessary for the problem to be solvable by this method. Once in this form, the calculation can be reduced to a matrix equation made up of the spatial interpolation terms (the stiffness matrix $\underline{K}$), the potential terms (the matrix $\underline{\phi}$) and the boundary condition terms (the load matrix $\underline{p}$):

$$\underline{K}^{(e)} \underline{\phi}^{(e)} = \underline{p}^{(e)} \tag{vi}$$

where

$$\underline{K}^{(e)} = \iint_{S^{(e)}} B^T D B.dS$$

$$\underline{p}^{(e)} = \int_{C_2^{(e)}} \underline{v}_0 \left[N^T\right].dC_2$$

and

$$B = \begin{bmatrix} \delta N_i/\delta x & \delta N_j/\delta x & \delta N_k/\delta x \\ \delta N_i/\delta y & \delta N_j/\delta y & \delta N_k/\delta y \end{bmatrix}$$

and

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The $\underline{p}$ matrix contains all the contributions from the second part of the expansion in equation (v), however, when the problem is solved all these terms should be zero apart from the boundary value terms. This means we neglect all the terms which do not concern elements on the boundaries. The $\underline{v}_0$ and C_2 terms for $\underline{p}$ in the above set of equations refers to the input velocity on the right-hand side boundary of the duct problem (v_0 in figure 1-1). This boundary is known as a Neumann boundary, as are the upper and lower boundaries (but no flow can cross these, so they do not contribute to $\underline{p}$). Further information on the implementation of the boundary conditions will be given in step 5.

Expanding $\underline{K}^{(e)}$ and $\underline{p}^{(e)}$ using $\underline{N}_i^{(e)}$ (equation (ii)) we get the final result that:

$$\underline{K}^{(e)} = \frac{1}{4A^{(e)}} \begin{bmatrix} b_i^2+c_i^2 & b_i b_j + c_i c_j & b_i b_k + c_i c_k \\ b_i b_j + c_i c_j & b_j^2+c_j^2 & b_j b_k + c_j c_k \\ b_i b_k + c_i c_k & b_j b_k + c_j c_k & b_k^2+c_k^2 \end{bmatrix} \tag{vii}$$

and

$$\underline{p}^{(e)} = \frac{1}{2}\left\{ V_{oij} l_{ij} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} + V_{ojk} l_{jk} \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + V_{oki} l_{ki} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right\} \qquad \text{(viii)}$$

where V_{oij}, V_{ojk} and V_{oki} are the baoundary conditions for the velocity of fluid leaving the edges ij, jk and ki respectively, and l_{ij}, l_{jk} and l_{ki} are the lengths of those edges.

Step 4 • Assembling the Matrices for the Whole Domain

Having created the $\underline{K}$ and $\underline{p}$ matrices for each element, these must be put together to create matrices which describe the whole of the domain. Such matrices must have the form:

For a domain of n nodes:

$$\underline{K}_{IJ}\ \underline{\phi}_I = \underline{p}_I$$

Where $\underline{K}_{IJ}$ is and n×n square matrix, and
$\underline{\phi}_I$, $\underline{p}_I$ are column matrices of n elements each.

Thus the domain is described in terms of the grid nodes. It is relatively simple to assemble the domain matrices from the element matrices, all we have to do is sum the contributions from each element into the domain matrices by mapping the i, j, k indicies of equation (vii) and (viii) onto the I and J indicies of $\underline{K}_{IJ}$ and the I index of $\underline{p}_I$.

Step 5 • Including the Boundary Conditions

The boundaries of the cylinder/duct break down into two kinds, Neumann and Dirichlet (see figure 2-3 below).

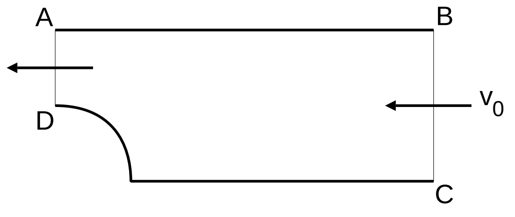

Figure 2-3

AB, BC and CD are Neumann boundaries and are defined as in step 3:

$\delta\phi/\delta n = 0$	on AB and CD,
$\delta\phi/\delta n = v_0$	on BC.

On these boundaries we know what the velocity should be, because we define v_0 and because no fluid is allowed to cross AB and CD (these boundaries have been taken care of in step 3). However, on the AD boundary, although we don't know what the velocity is, we do know what the potential is, ie the arbitrary zero potential. This is an example of a Dirichlet boundary and must be handled a little differently.

We must modify the node-wise matricies $\underline{K}_{IJ}$ and $\underline{p}_I$ for all the nodes on AD. If I refers to a node on AD, then

$$\phi_I = \phi_o$$

thus defining the arbitrary zero potential. We then modify $\underline{K}_{IJ}$ and $\underline{p}_I$ such that:

$$\underline{K}_{II} = 1 \quad ; \quad \underline{p}_I = \phi_o \quad \text{and} \quad \underline{K}_{IJ} = 0 \quad \text{for all } J \neq I.$$

We may also simplify the matrix calculation by inserting the constant value in the other equations, and retaining the symmetry of the stiffness matrix. Thus:

$$\underline{p}_J = \underline{p}_J - \underline{K}_{JI}\,\phi_I \quad \text{and} \quad \underline{K}_{JI} = 0 \quad \text{for } J \neq I.$$

Step 6 • Solving the Domain Matrix Equation

As the above steps yield a symmetric matrix we carry out the solution of this set of simultaneous equations by a Cholesky method. For more details on the form of this method, see:

MacKeown P K & Newman D J, 1987 , *Computational Techniques in Physics* (Adam Higler).
See §6.5 and Exercise 6.3.

Or more in depth,

Bath K-J, 1982, *Finite Element Procedures in Engineering Analysis* (Englewood Cliffs, NJ: Prentice Hall).
Discussed in §8.2.

Analytical Results for Flow Over a Cylinder in the Absence of a Duct

It can be shown via complex variable means that the flow around an isolated cylinder can be dcscribed as:

$$\phi = v_0\, x\, [1+R^2/(x^2+y^2)]$$

This result leads to the following equation for the pressure on the cylinder of the form p = p(●), which can be compared to the output of the PRESS subroutine:

$$p = p_0 + {}^1/_2\, v_0^2\, (1 - 4 \sin^2 \bullet) \qquad \text{(ix)}$$

Method

To aid the solution of our finite element problem, we have the routines detailed below. Most are for input and output purposes, whilst others help with the actual steps of the calculation.

• MENU	• Routine to ask the user to define the problem in terms of r(1-5), d,w and v_o (see step 2 of the theory section, page 3).
• STGRID	• Set up the definition of the grid from the parameters passed from MENU (this routine performs steps 1 and 2 of the calculation).
• PTGRID	• Plots the grid on the screen with elements and nodes numbered (for reference).
• DECOMP • SOLVE	• The first routine decomposes the stiffness matrix and the second solves for the velocity potential matrix. ie together these routines perform step 6.
• VELOC	• Calculates the velocity at each node from the velocity potential data (for the TRACE routine).
• TRACE	• Plots the finite element grid and then superimposes the streamlines around the cylinder, as calculated from the velocity data.
• PRESS	• Plots the pressure along the surface of the cylinder as p against ●.

[These routines are included in full in appendix B.]

So, in order to solve for the velocity potential, routines must be written to carry out steps 3,4 and 5. ie we must assemble equations (vii) and (viii) for each element, create the domain matrix from these and then add in the boundary conditions. To do this we need to know how the routines pass data to each other, which in this case is through the use of dummy variables of the following form:

• NN	• an integer holding the number of nodes.
• NE	• an integer holding the number of elements.
• NB	• an integer holding the bandwidth of the $\underline{K}$ matrix (see later).
• NODE(NE,3)	• an array listing the three nodes for each of the NE elements.
• XC(NN)	• an array listing the x coordinates of each of the NN nodes.
• YC(NN)	• an array listing the y coordinates of each of the NN nodes.
• NCON(NE,2)	• an array listing nodes on a boundary for which velocity is specified for each element (0 if node is not on boundary.
• ICON(NE)	• an array identifying is an element adjoins a boundary on which velocity is specified (1 if true, 0 otherwise).
• q(NE)	• boundary outflow velocity for an element, non-zero only if an element joins a Neumann boundary.
• PS(NN)	• nodal value of potential where specified by boundary conditions (set to-1000.0 if value is not preset).
• P(NN)	• the load vector.
• PLOAD(NN)	• the final solution vector.
• A(NE)	• area of element.

• GK(NN,NB) • the stiffness matrix. As mentioned in the theory, K has a symmetric form. However, because there are only a certain number of nodes that are directly connected to each other, then K must form a band matrix, the bandwidth of which depends on how efficiently the nodes are numbered with respect to the way they are interconnected. Hence we are passed the bandwidth NB from STGRID, because the arrangement of the mesh defines it. The point of this being that we can make large savings in the memory required to store the stiffness matrix by only storing the non-zero band elements. The transformation from $\underline{K}_{IJ}$ to GK(NN,NB) is illustrated below (fig. 3-1).

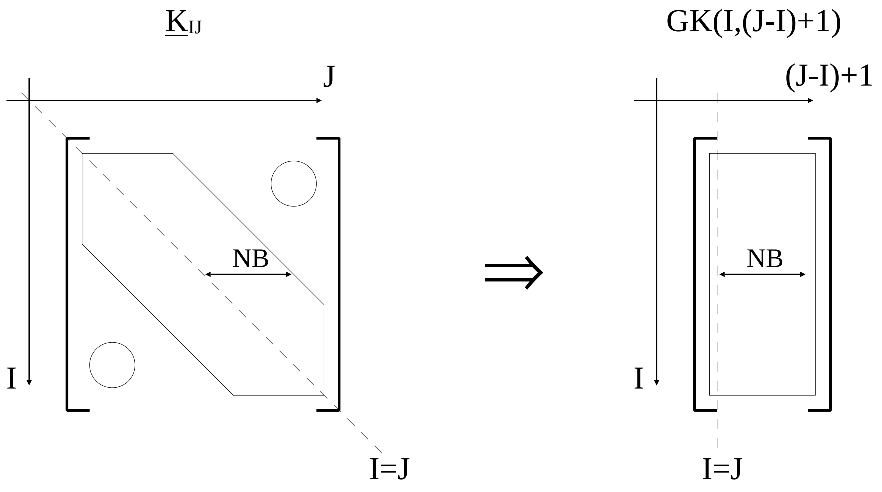

Figure 3-1

It is clear that this transformation requires us to check that J ≥ I before adding data into GK.

Given these routines and data, we first need a control routine which defines space for all the data in the list above and then calls all the routines which make up the calculation, passing the required array pointers etcetera. This is simple enough and just requires the knowledge of the fact that there will be up to 50 nodes and up to 70 elements. This control routine is given in appendix A.

My routine to carry out steps 3,4 and 5 has the structure described below.

Subroutine DEFINEKP

•

Define names and dimensions of dummy arrays as in the control procedure, along with local variables.

•

Loop through all elements in GK and P and set them all to zero.

•

Initiate DO loop over all the elements in order to fill up GK and P (let [e] denote the current element).

•

Look up i,j,k as the numbers of the nodes for this element from NODE([e],1/2/3)

•

Calculate the b and c coefficients for i,j,k (see page 3)
and put this into arrays of the type b(1/2/3), c(1/2/3).
Also calculate the area of this element, area([e]).

•

Initiate a DO loop for I = 1,2,3.

•

Initiate another DO loop for J = 1,2,3.

•

Let ia = NODE([e],I)
Let ib = NODE([e],J)

•

IF ib ≥ ia THEN GK(ia,(ib-ia)+1 = GK(ia,(ib-ia)+1) +
(b(I)×b(J) + c(I)×c(J)) / (4.0×area([e]))

•

Close DO loop for J.

•

Close DO loop for I.

•

IF ICON([e])=1 THEN this element has a Neumann boundary so we must modify P. Looking at the boundary nodes:
Let na = NCON([e],1)
Let nb = NCON([e],2)
$l_{ab} = \sqrt{((xc(nb)-xc(na))^2 + (yc(nb)-yc(na))^2)}$
And so from equation (viii):
$P(na)=P(na) + (q([e])\times l_{ab}) / 2$
$P(nb)=P(nb) + (q([e])\times l_{ab}) / 2$

•

Close DO loop over elements for GK and P.

•

Now adding the Dirichlet boundary conditions (See pages 5 and 6):

•

Initiate a DO loop over all nodes, current node [N].

•

IF ps([N])≠ -1000.0 THEN:

•

Initiate a DO loop nodes from [N] to NN,
current node [n].

•

IF [N] = [n] THEN GK([N],1) = 1.0
IF [N] ≠ [n] THEN GK([N],([n]-[N])+1) = 0.0

•

Close DO loop for [n].

•

Let P([N]) = PS([N]).

•

```
Initiate DO loop over all nodes, current node [n].
•
IF [N] ≠ [n] THEN p([n]) = p([n]) -
GK([n],ABS(([N]-[n]))+1) × ps([N]).
•
Close DO loop over [n].
•
Initiate DO loop over nodes from 1 to [N]-1, current node
[n].
•
GK([n],([N]-[n])+1) = 0.0
•
Close DO loop over [n].
•
ENDIF
•
Close DO loop over all nodes [N].
•
Copy contents of P into calculation array PLOAD:
•
Initiate DO loop over all nodes, current node [N].
•
PLOAD([N]) = P([N])
•
Close DO loop over all nodes [N].
•
End subroutine.
```

The full FORTRAN code implementation of the above is given in appendix A. Now the code works, we can move on to the objectives of the experiment:

1 • Identification of a Reasonable Working Grid

In order to work out how effective a particular mesh has been we must have some way how working out how bad the results of a particular calculation are. This is not an easy problem. as direct calculation of an error measure such as how close $\nabla^2\phi$ actually is to zero is rather difficult from the information we have. However, we can use one of the characteristics of fluid flow to give us a reasonably good idea of the error in a calculation. The characteristic used in my code is the principle of conservation of mass. For an incompressible system such as this, the volume of fluid that leaves the left hand side of the grid for any particular time interval must equal the volume of fluid coming in at the right hand side, ie if the gap for the fluid to pass through at one side is smaller than the other, then the fluid must pass quickest through that narrowest side. Formally, if the velocity of fluid in is v_{in} and fluid out is v_{out}, where the width if the incoming duct is d_{in} and the outgoing duct is d_{out}, then the velocity multiplied the width is a constant, ie:

$$v_{in}\, d_{in} = v_{out}\, d_{out}$$

Thus we can workout what the average velocity should be over the left hand edge and compare this to the value calculated from the above equation. The FORTRAN routine which carries out this error estimation, called TESTFLOW, is given in appendix A.

So now we can alter the values of r(2-5) to see how the spacing of the elements affects the validity of the calculation. The only problem with this is that for any given problem (ie a given set of values for r(1), w, d and v_0) we need to alter 4 different variables before each calculation (r(2-5)). This is a rather messy way to go about things (and rather prone to typing errors) so I used a more methodical approach.

For this first objective, I used a linearly spaced grid. In other words, when a value of r(2) was set from the MENU subroutine, I altered the routine to set r(3-5) as shown below:

$$r(3) = 2 \times (r(2) - r(1)) + r(1)$$
$$r(4) = 2 \times (r(3) - r(2)) + r(2)$$
$$r(5) = 2 \times (r(4) - r(3)) + r(3)$$

ie the polar elements are evenly spaced (r(2) - r(1) between each one).

Now the objective is to find a value of r(2) which minimises the error, given these parameters:

r(1)	=	1.0
w	=	2.5
d	=	5.0
v_0	=	1.0

The actual magnitude of v_0 is arbitrary as the velocity potential result is the important result, and so altering v_0 will not alter the streamlines/potential result.

2 • Change in flow behaviour due to alteration of the duct width, w

This aim requires us to examine the pressure variation over the cylinder (from PRESS) for a range of duct widths (w), and to compare this to the distribution found in the absence of the duct, described by equation (ix). The only complication for this is that the value of r(2) found in part 1 (above) need not suffice for a range of values of w, and in fact may force the grid outside the walls of the duct for some values. Therefore it is necessary to find the best value of r(2) for each value of w, by the same prodecure as part 1. After doing this is may be possible to find a relationship between duct width and the best value for r(2).

3 • Advantages of Non-Linear Grid Spacing Over Linear Spacing

As a further investigation, the advantages of a non-linear spacing between the r(2-5) values can be examined. This involves carrying out part 2 again, but instead of using the linear relationship given above, we use the non-linear relationship below:

$$r(3) = 2 \times (r(2) - r(1)) + r(1)$$
$$r(4) = 2 \times (r(3) - r(1)) + r(1)$$
$$r(5) = 2 \times (r(4) - r(1)) + r(1)$$

The results under these conditions can then be compared to the results from part 2.

Results

1 • 1 • Identification of a Reasonable Working Grid

The systematic experimentation with values of r(2) to minimise the error lead to the results shown in table 1 below.

r(2)	r(3)	r(4)	r(5)	Error %
1.1	1.2	1.3	1.4	37.37
1.2	1.4	1.6	1.8	12.89
1.225	1.450	1.675	1.900	7.99
1.25	1.50	1.75	2.00	3.41
1.26	1.52	1.78	2.04	1.65
1.265	1.530	1.795	2.060	0.79
1.2675	1.5350	1.8025	2.0700	0.36
1.27	1.54	1.81	2.08	0.07
1.2725	1.5450	1.8175	2.0900	0.49
1.275	1.550	1.825	2.100	0.91
1.28	1.56	1.84	2.12	1.75
1.3	1.6	1.9	2.2	5.00

Table 1

Hence, for the given duct problem the best value for r(2) is 1.27. The resultant streamline plot is given in figure 4-1 below (along with the grid specified by setting r(2) to 1.27).

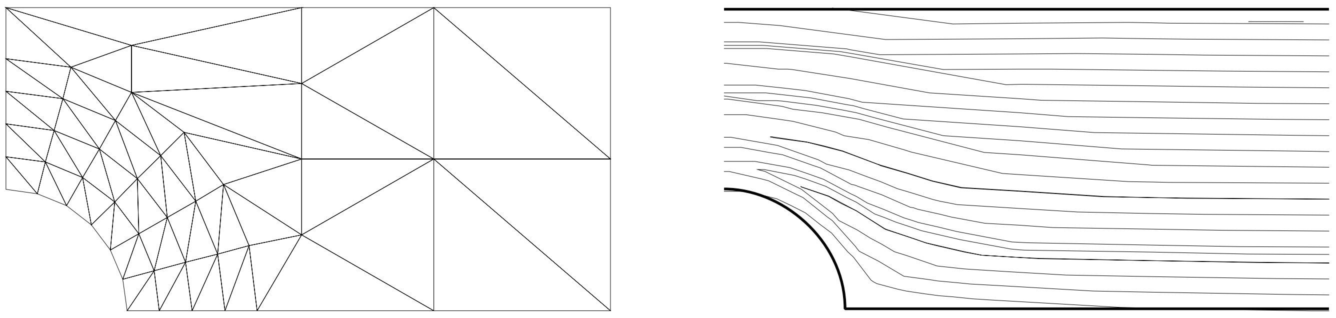

Figure 4-1

To give an idea of how the grid proportions relate to the success of the calculation, figure 4-2 (overleaf) shows the resultant streamlines for r(2) = 1.1 (error = 37.37%).

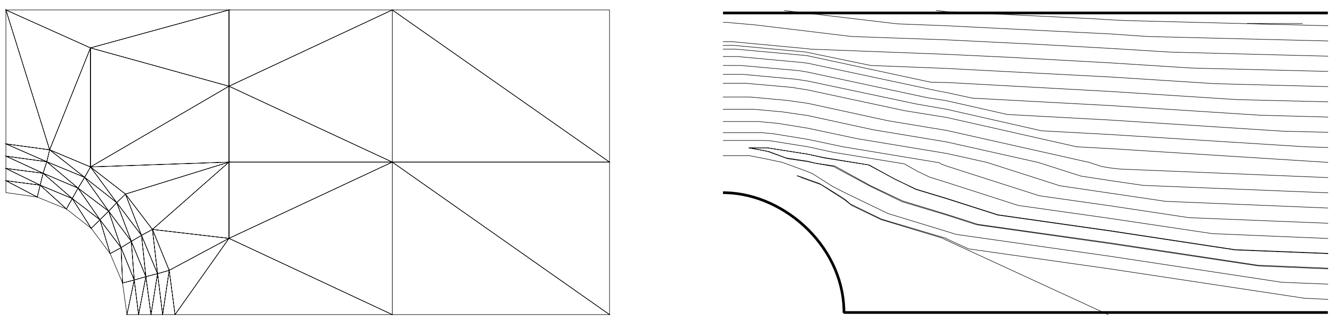

Figure 4-2

In this case the densely packed elements have lead to the flow being unable to follow the body streamline and creating a low pressure area in front of the cylinder. It can be seen from these results that reasonable accuracy can be obtained by just having the elements evenly spaced over the Dirichlet boundary, but for a good level of accuracy we need to hunt down the best value for r(2).

2 • Change in flow behaviour due to alteration of the duct width, w

In order to get the best results possible, I used trial and error to find the best values for r(2) for each set of r(1), w, d and v_0 values, for a range of duct widths (w). It was not possible to examine duct widths below 2.00 as this caused the code to crash out during the process of solving the matrix equation. I examined the w interval 2.00 - 3.50 ,as isolated cylinder behaviour occurs in this range. The pressure on the cylinder for the 2.00[0.25]3.50 interval is shown in figure 4-3.

w	Best r(2)	Error %
2.00	1.2499	50.13
2.10	1.274	24.14
2.20	1.299	7.61
2.25	1.3124	1.16
2.30	1.315	0.56
2.40	1.295	0.21
2.50	1.27	0.07
2.75	1.21	0.67
3.00	1.17	0.43
3.25	1.13	0.49
3.50	1.09	0.20

Table 2

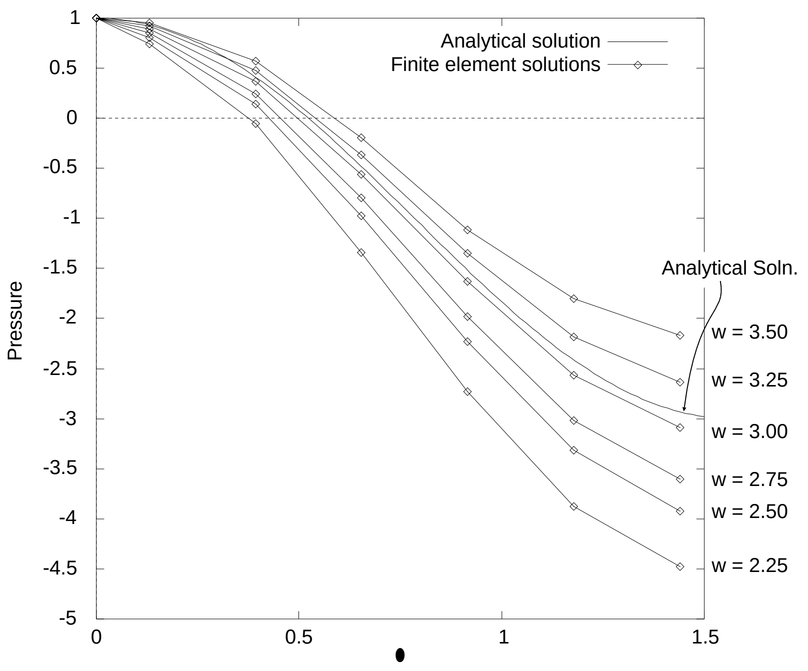

Figure 4-3

The pressure plot only includes the evenly spaced values of w in order to make the system behaviour more apparent. The lines of increasing w are roughly evenly spread, resulting in the pressure at the uppermost point of the cylinder ($\theta = \pi/2$) holding an approximately linear relationship with the duct width. This corresponds to the decrease in the average velocity outflow as the duct widens (see bottom of page 10). Figure 4-3 also reveals the inaccuracy of the simulation at large w, in that although the pressure tends toward the isolated cylinder relationship at first, it then goes beyond it ($w > 3.00$). This is an error due to trying to fit very few elements over a large area, which cannot be picked up by my error calculation (see figure 4-4).

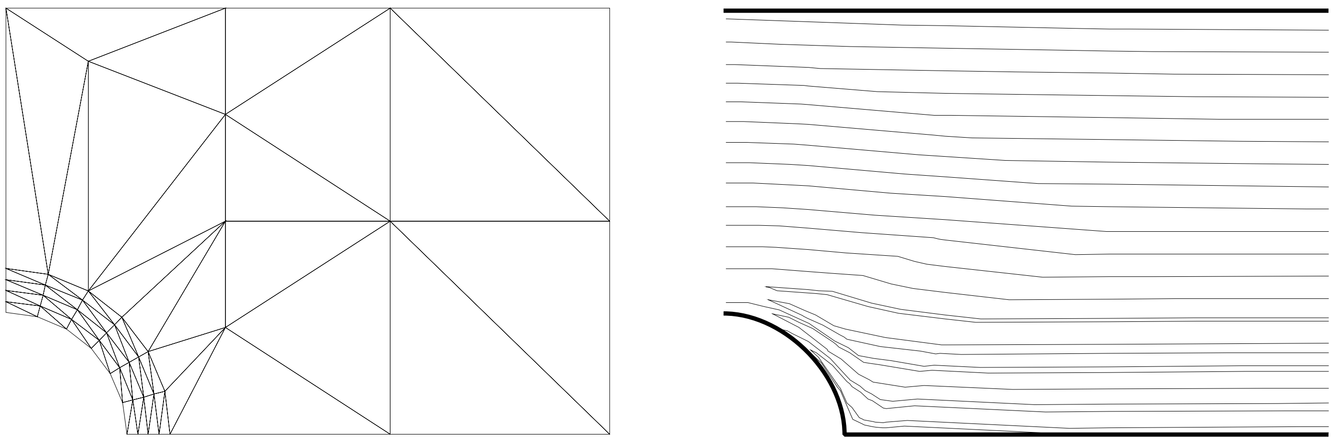
Figure 4-4

From this it can be seen that for large w, there are very few elements (relative to the area they cover) to handle the flow beyond the cylinder, so the simulation becomes innaccurate.

Table 2 also gives the relationships between w, r(2) and the error as shown in figures 4-5 and 4-6 below.

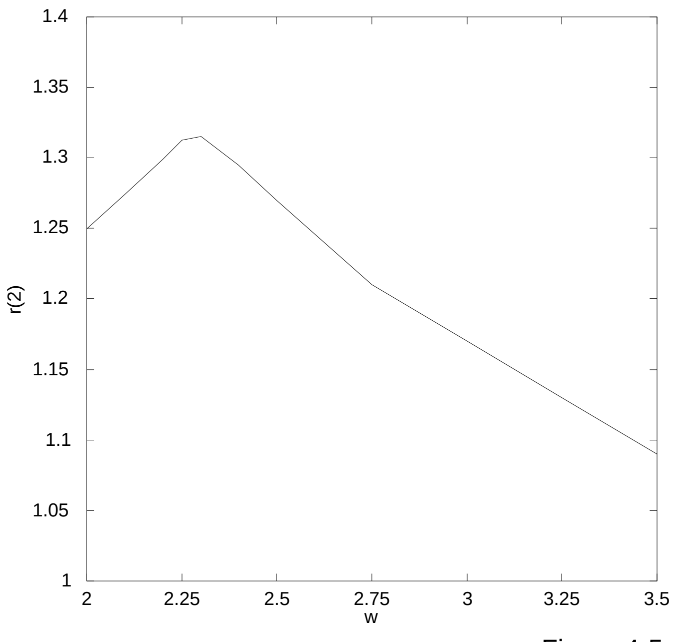

Figure 4-5

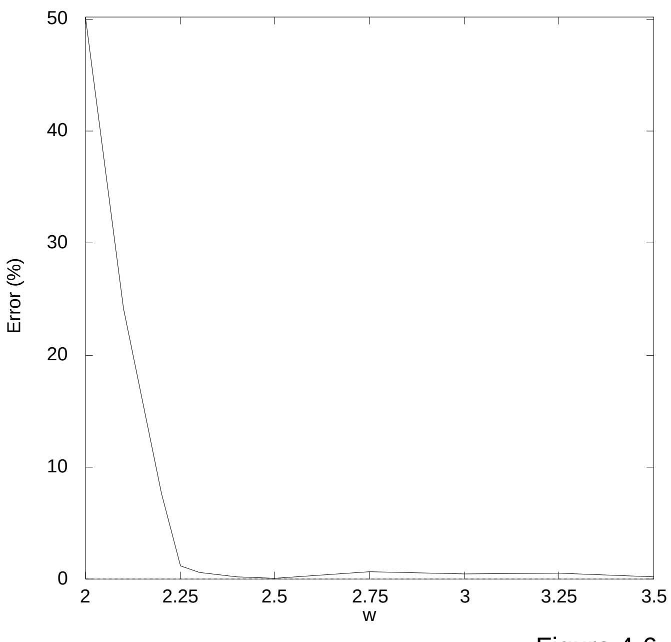

Figure 4-6

As you can see the turning point in figure 4-5 corresponds to the sudden increase in the error in the calculation (figure 4-6) at $w \approx 2.25$. This is because the best arrangement of elements, as defined by r(2), is now being restricted by the duct wall, the densely packed elements this creates around the cylinder force the streamlines into the duct wall, ie an illegal compression of the fluid occurs. It is this fault that the error routine detects.

Figure 4-5 also illustrates the linear relationship between the element spacing and the duct width for larger w, which using a least squares fit method (on the low error data points only) is:

$$r(2) = -0.176\,w + 1.702$$

Although this is only valid for high w within the realm of the simulation, due to the simulation not acting as it should for high w (see earlier).

3 • Advantages of Non-Linear Grid Spacing Over Linear Spacing

In order to evaluate the advantages or otherwise of the non-linear grid arrangement, the same procedure as in 2 (above) can be carried out for the interval in w of 2.00-3.50 (this routine also failed to give results below w = 2.00). The results are as shown in table 3.

w	Best r(2)	Error %
2.00	1.1249	50.66
2.10	1.137	24.43
2.20	1.15	7.20
2.25	1.157	1.09
2.30	1.156	0.36
2.40	1.147	0.13
2.50	1.135	0.07
2.75	1.108	0.08
3.00	1.084	0.04
3.25	1.064	0.06
3.50	1.046	0.05

Table 3

By combining the r(2) against w and error against w plots for both the linear and non-linear grids on the same sets of axis, we can directly compare the two methods:

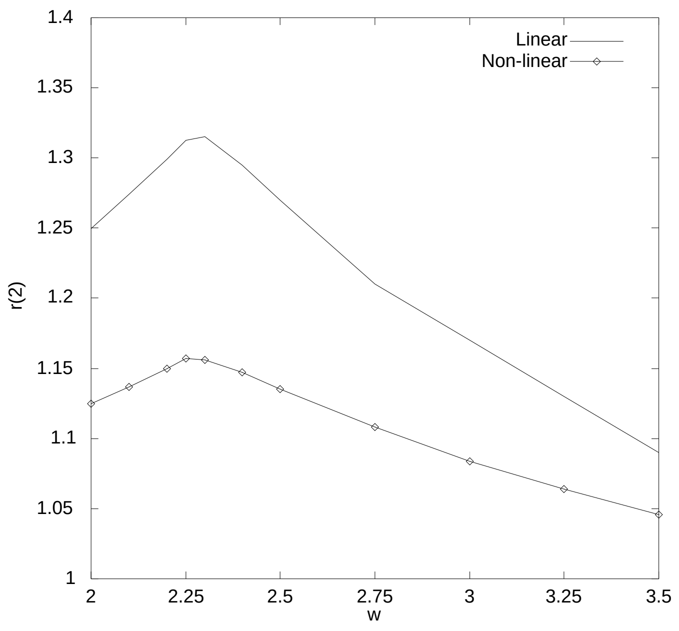

Figure 4-7

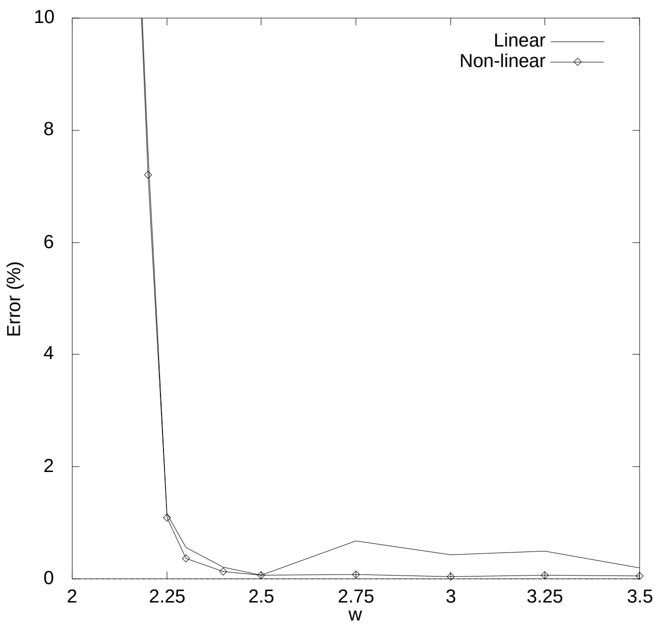

Figure 4-8

The general behaviour is the same in both cases, including the position of the turning point. The only difference in figure 4-7 is that the whole curve is squashed down lower (as expected, as the non-linear relationship covers more ground over the 4 radial elements for a given value of r(2)). The least squares fit linear relationship for this is:

$$r(2) = -0.09\,w + 1.358$$

The only advantage in using a non-linear grid is illustrated in figure 4-8. It is clear that for most of the range of w shown, the non-linear grid arrangement is more accurate than the linear one. Yet again this is only correct within the realm of the simulation, as shown by the behaviour of the system at large w (see figure 4-9). An example of a non linear grid is given in figure 4-10, and is for the w = 2.50 simulation from table 3.

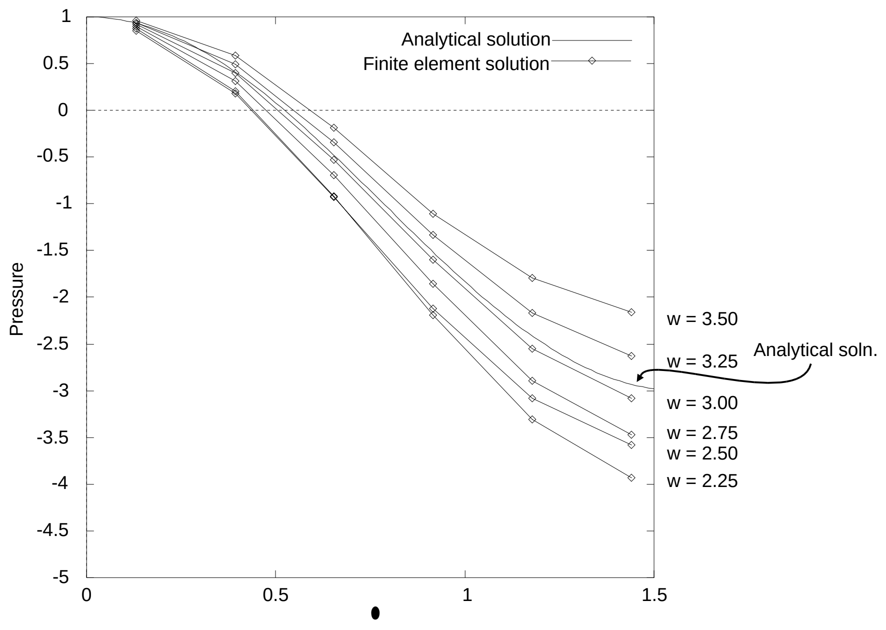

Figure 4-9

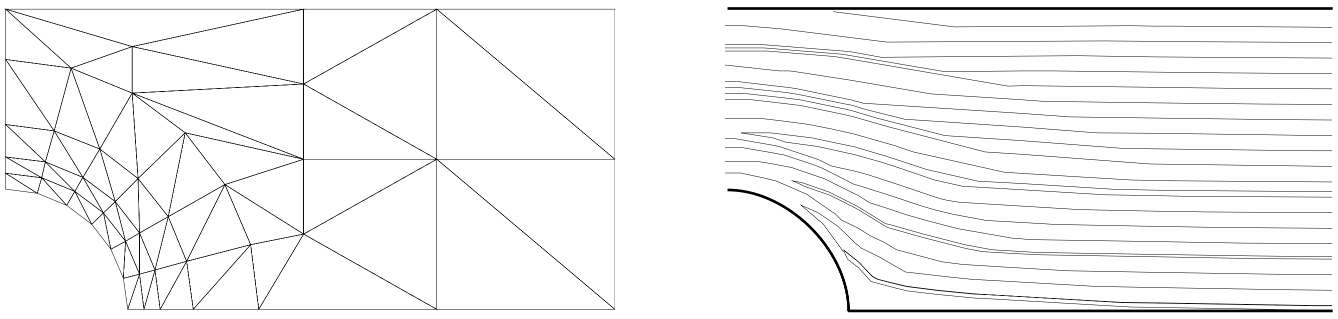

Figure 4-10

Conclusion

As we have seen, this simulation was reasonably successful within a small range of duct widths (ie from 2.25 to 3.00 for a 1.00 radius cylinder), and a reasonable grid for this range can be predicted using the linear relationships shown. However, the simulation failed in the upper and lower ranges of w due to having a grid which is unsuitable for the given problem, and so no general solution for the ideal grid proportions over a large range of w could be found. I would suggest two ways in which the performance of simulation could be improved:

i) There were often problems caused by the compactness of the elements over the leading edge of the cylinder, as this arrangement does not reflect the nature of the actual flow (especially true for high pressure flow, ie low duct width). I would therefore suggest making the polar elements elliptical, so that they spread out in front of the cylinder and slowly introduce the flow to the cylinders existence.

ii) For the larger ranges of w, simply increasing the number of elements over the whole domain would improve the systems performance. This would also help the low w range calculations.

These conclusions illustrate the way in which the set up of the mesh is the important element for specific fluid problems. The actual calculation and matrix routines can be generalised to any solution of Laplace's equation, but defining the right kind of mesh for a problem is absolutely crucial to the success of the finite element method.

Appendix A • My FORTRAN code

```
C Procedure calling control routine for Finite Element project.
C
C By A N Jackson esq.  (c) 1995
C
      program FinElem
      dimension node(70,3),xc(50),yc(50),icon(70),ncon(70,2),
     +ps(50),q(70),gk(50,70),p(50),pload(50),area(70),vx(70),
     +vy(70)
      common /bkdata/ r(5),w,d,qq
      integer nn,ne,nb
      data icon /70*0/
      data ncon /140*0/
c File i/o...
      OPEN(UNIT=13, FILE='JACK.OUT', STATUS='UNKNOWN')
c
      call menu
      call stgrid(nn,ne,nb,node,xc,yc,icon,ncon,ps,q)
      call ptgrid(nn,ne,node,xc,yc)
c Call up my code...
      call definekp(nn,ne,nb,node,xc,yc,icon,ncon,ps,q,gk,
     +p,pload,area)
c Solve...
      call decomp(nn,nb,gk)
      call solve(nn,nb,1,gk,pload)
      call veloc(nn,ne,node,xc,yc,pload,vx,vy)
c Show results...
c      call trace(nn,ne,node,xc,yc,icon,ncon,vx,vy)
      call press(ne,vx,vy)
c Check Dirichlet outflow...
      call testflow(nn,ne,node,xc,yc,icon,ncon,ps,q,vx,vy)
c      call potplot(ne,node,xc,yc,pload)
c Fin file i/o...
      CLOSE (UNIT=13)
      end
c
c My routines
c
      subroutine potplot(ne,node,xc,yc,pload)
      dimension node(70,3),xc(50),yc(50),pload(50)
      integer n,i,j
      common /bkdata/ r(5),w,d,qq
c Title in file...
      WRITE(13,*) '#'
      WRITE(13,*) '# Element potential plot (x,y,phi)'
      WRITE(13,*) '# w=',w
      WRITE(13,*) '#'
      WRITE(13,*) char(13)
c
      DO i=1,ne
       DO j=1,4
        IF (j.LE.3) n=node(i,j)
        IF (j.GT.3) n=node(i,j-3)
        WRITE(13,666) xc(n),yc(n),pload(n)
       END DO
       WRITE(13,*) char(13)
      ENDDO
      WRITE(13,*) char(13)
 666  FORMAT(3F12.6)
      return
      end
c
      subroutine testflow(nn,ne,node,xc,yc,icon,ncon,ps,
     +q,vx,vy)
      dimension node(70,3),xc(50),yc(50),icon(70),ncon(70,2),
     +ps(50),q(70),vx(70),vy(70)
      dimension edge(2)
      common /bkdata/ r(5),w,d,qq
      real v,dx,dy,l,vorl,vres,ratio,error
      integer i,j,edge
c Title in file...
      WRITE(13,*) '#'
      WRITE(13,*) ‘# Dirichelt boundary velocity plot (y,v)’
      WRITE(13,*) ‘# w=’,w
      WRITE(13,*) ‘#’
      WRITE(13,*) char(13)
c
      ratio=w/(w-r(1))
      vorl=0.0
      DO i=1,ne
       itest=0
       DO j=1,3
        n=node(i,j)
        IF (ps(n).LT.-1000.0 .OR. ps(n).GT.-1000.0) THEN
         itest=itest+1
         edge(itest)=n
        ENDIF
       ENDDO
       IF (itest.EQ.2) THEN
        v=SQRT(vx(i)*vx(i)+vy(i)*vy(i))
        dx=xc(edge(2))-xc(edge(1))
        dy=yc(edge(2))-yc(edge(1))
        l=SQRT(dx*dx+dy*dy)
        vorl=vorl+v*l
        WRITE(13,666) 0.5*(yc(edge(1))+yc(edge(2))),v
        WRITE(*,666) 0.5*(yc(edge(1))+yc(edge(2))),v
       ENDIF
      ENDDO
      vres=vorl/(w-r(1))
      error=abs((ratio-vres)/ratio)*100
      write(13,*) char(13)
      write(13,665) error
      write(*,665) error
      write(13,666) w,error
  665  FORMAT(1X,’# Error =+/-’,F5.2,’%’)
  666  FORMAT(1X,F12.6,2X,F12.6)
      return
      end
c
      subroutine definekp(nn,ne,nb,node,xc,yc,icon,
     +ncon,ps,q,gk,p,pload,area)
      dimension node(70,3),xc(50),yc(50),icon(70),ncon(70,2),
     +ps(50),q(70),gk(50,70),p(50),pload(50),area(70)
      dimension bcoeff(3),ccoeff(3)
      common /bkdata/ r(5),w,d,qq
      real addon,la2b,dx,dy
      integer nn,ne,nb,inn,ine,inb,ii,jj,kk,i,j,ia,ib
c  First zero all elements in the K matrix:
      DO inn=1,nn
       DO inb=1,nb
        gk(inn,inb)=0.0
       ENDDO
        p(inn)=0.0
      ENDDO
c  Now fill up the K matrix element by element
      DO ine=1,ne
       ii=NODE(ine,1)
       jj=NODE(ine,2)
       kk=NODE(ine,3)
c  Work out area of element
       area(ine)=(xc(kk)-xc(ii))*(yc(jj)-yc(ii))
     +          -(xc(jj)-xc(ii))*(yc(kk)-yc(ii))
       area(ine)=0.5*abs(area(ine))
c  Set up b and c coefficent arrays
       bcoeff(1)=yc(jj)-yc(kk)
       bcoeff(2)=yc(kk)-yc(ii)
       bcoeff(3)=yc(ii)-yc(jj)
       ccoeff(1)=xc(kk)-xc(jj)
       ccoeff(2)=xc(ii)-xc(kk)
       ccoeff(3)=xc(jj)-xc(ii)
c  Now fit all this into K
       DO i=1,3
        DO j=1,3
         addon=bcoeff(i)*bcoeff(j)+ccoeff(i)*ccoeff(j)
         ia=NODE(ine,i)
         ib=NODE(ine,j)
         IF (ib.GE.ia) THEN
```

```
           gk(ia,(ib-ia)+1)=gk(ia,(ib-ia)+1)+(addon/(4.0*area(ine)))
          ENDIF
         ENDDO
        ENDDO
c  And set up the P matrix too
       If (ICON(ine).EQ.1) THEN
        nodea=NCON(ine,1)
        nodeb=NCON(ine,2)
        dx=xc(nodeb)-xc(nodea)
        dy=yc(nodeb)-yc(nodea)
        la2b=SQRT(dx*dx+dy*dy)
        p(nodea)=p(nodea)+0.5*q(ine)*la2b
        p(nodeb)=p(nodeb)+0.5*q(ine)*la2b
       END IF
c  On to next element(kp)...
       ENDDO
c  Now add Dirichlet boundary conditions...
       DO inn=1,nn
        IF (ps(inn).GT.-1000.0 .OR. ps(inn).LT.-1000.0) THEN
c  Modify K
         DO j=inn,nn
          IF (j.EQ.inn) THEN
           GK(inn,1)=1.0
          ELSE
           GK(inn,(j-inn)+1)=0.0
          ENDIF
         ENDDO
c  Modify P
         p(inn)=ps(inn)
c  Furthur simplify...
         DO j=1,nn
          IF (j.NE.inn) then
           p(j)=p(j)-gk(j,abs((inn-j))+1)*ps(inn)
          ENDIF
         ENDDO
         DO j=1,inn-1
          gk(j,(inn-j)+1)=0.0
         ENDDO
        ENDIF
c On to next node(dirichlet)...
       ENDDO
c Copy p into pload
       DO inn=1,nn
        pload(inn)=p(inn)
       ENDDO
       return
       end
c
```

Appendix B • FORTRAN routines supplied for the experiment

```
      subroutine menu
      integer hasckey
      common /bkdata/ r(5),w,d,qq
      r(1)=1.0
      r(2)=1.1
      r(3)=1.2
      r(4)=1.3
      r(5)=1.4
      w=2.40
      d=5.0
      qq=1.0
  100 write (*,1000) r(1)
 1000 format ('  0 Radius of cylinder               ',1pe11.3)
      write (*,1001) r(2)
 1001 format ('  1 Radius of first set of elements ',1pe11.3)
      write (*,1002) r(3)
 1002 format ('  2 Radius of second set of elements',1pe11.3)
      write (*,1003) r(4)
 1003 format ('  3 Radius of third set of elements ',1pe11.3)
      write (*,1004) r(5)
 1004 format ('  4 Radius of fourth set of elements',1pe11.3)
      write (*,1005) w
 1005 format ('  5 Width of channel                ',1pe11.3)
      write (*,1006) d
 1006 format ('  6 Length of duct                  ',1pe11.3)
      write (*,1007) qq
 1007 format ('  7 Inflow velocity                 ',1pe11.3)
      print *,'  Do you wish to change any parameter (y or n)'
  101 i=hasckey()
      if (i.ne.78.and.i.ne.89.and.i.ne.110.and.i.ne.121) go to 101
      if (i.eq.89.or.i.eq.121) then
  110 print *,'  Which parameter do you wish to change (insert no.)'
  111 i=hasckey()
      if (i.lt.48.or.i.gt.55) go to 111
      print *,' Insert new value      '
      read (*,*) x
      write (*,*) i,x
      if (i.ge.48.and.i.le.52) r(i-47)=x
      if (i.eq.53) w=x
      if (i.eq.54) d=x
      if (i.eq.55) qq=x
c AJ bit in here *************************************************************
      if (i.eq.49) then
       r(3)=2*(r(2)-r(1))+r(1)
       r(4)=2*(r(3)-r(1))+r(1)
       r(5)=2*(r(4)-r(1))+r(1)
       write(*,*) 'New non-linear arrangement set.'
      endif
c ***************************************************************************
      print *,'  Do you wish to change any other parameter (y or n)'
  112 i=hasckey()
      if (i.ne.78.and.i.ne.89.and.i.ne.110.and.i.ne.121) go to 112
      if (i.eq.89.or.i.eq.121) go to 110
      call hprepit
      call hfinish
      go to 100
      endif
      return
      end
      subroutine stgrid(nn,ne,nb,node,xc,yc,icon,ncon,ps,q)
      dimension node(70,3),xc(50),yc(50),icon(70),ncon(70,2),
     +ps(50),q(70)
      dimension sint(0:6),cost(0:6)
      dimension snode(22,3)
      common /bkdata/ r(5),w,d,qq
      data cost
     1/1.0,0.965925826,0.866025403,0.707106781,0.5,0.258819045,0.0/
      data sint
     1/0.0,0.258819045,0.5,0.707106781,0.866025403,0.965925826,1.0/
      data (snode(i,1),i=1,22) /-6,-5,-4,-4,-3,-2,-2,-2,-1,-1,04,03,
     +-6,01,01,02,03,03,07,08,08,09/
      data (snode(i,2),i=1,22) /-5,-4,01,-3,-2,02,03,-1,04,00,05,04,
     +01,07,02,03,08,06,08,10,09,11/
      data (snode(i,3),i=1,22) /01,01,02,02,02,03,04,04,05,05,06,06,
     +07,08,08,08,09,09,10,11,11,12/
c Set the elements within the arcs
      m=0
      n=-1
      jj=1
      do 1 j=2,5
      if (r(j).lt.r(1)) go to 1
      n=n+1
      jj=jj+1
      do 10 i=1,6
      m=m+2
      n=n+1
      node((m-1),1)=n
      node((m-1),2)=n+7
      node((m-1),3)=n+8
      node(m,1)=n
      node(m,2)=n+1
      node(m,3)=n+8
   10 continue
    1 continue
c  Set the remaining elements off the circular arcs
      m0=12*(jj-1)
      n0=7*jj
      ne=m0+22
      nn=n0+12
      do 11 i=1,22
      m=m0+i
      node(m,1)=n0+snode(i,1)
      node(m,2)=n0+snode(i,2)
      node(m,3)=n0+snode(i,3)
   11 continue
c Identify the boundary nodes on which the outflow velocity is specified
      icon(ne-2)=1
      ncon((ne-2),1)=n0+snode(20,2)
      ncon((ne-2),2)=n0+snode(20,3)
      icon(ne)=1
      ncon(ne,1)=n0+snode(22,2)
      ncon(ne,2)=n0+snode(22,3)
c Position the nodes on the circular arcs
      n=0
      do 2 j=1,5
      if (r(j).lt.r(1)) go to 2
      do 20 i=0,6
      n=n+1
      xc(n)=r(j)*cost(i)
      yc(n)=r(j)*sint(i)
   20 continue
    2 continue
c  Position the remaining nodes off the circular arcs
      xc(n0+1)=r(jj)+0.125*(d-r(jj))
      yc(n0+1)=0.25*w
      xc(n0+2)=xc(n0+1)
      yc(n0+2)=0.5*w
      xc(n0+3)=xc(n0+1)
      yc(n0+3)=0.75*w
      xc(n0+4)=0.5*r(jj)
      yc(n0+4)=0.875*w
      xc(n0+5)=0.0
      yc(n0+5)=w
      xc(n0+6)=xc(n0+1)
      yc(n0+6)=w
      xc(n0+7)=0.5*(r(jj)+d)
      yc(n0+7)=0.0
      xc(n0+8)=xc(n0+7)
      yc(n0+8)=yc(n0+2)
      xc(n0+9)=xc(n0+7)
      yc(n0+9)=w
      xc(n0+10)=d
      yc(n0+10)=0.0
      xc(n0+11)=d
      yc(n0+11)=yc(n0+2)
      xc(n0+12)=d
      yc(n0+12)=w
      nb=0
      do 3 m=1,ne
      nb=max0(nb,iabs(node(m,1)-node(m,2)),
     +iabs(node(m,2)-node(m,3)),iabs(node(m,3)-node(m,1)))
    3 continue
c  Insert the boundary conditions
c  The fixed values of the potential on the symmetry line
      do 40 i=1,nn
      ps(i)=-1000.0
      if (abs(xc(i)).lt.1.0e-10) ps(i)=0.0
   40 continue
c  The outflow velocities on the exit line
      do 41 i=1,ne
      if (icon(i).ne.0) q(i)=-qq
   41 continue
      return
      end
      subroutine ptgrid(nn,ne,node,xc,yc)
      dimension node(70,3),xc(50),yc(50)
      character*2 file
      integer hasckey,hrsx,hrsy
      common /bkdata/ r(5),w,d,qq
c AJ bit
      WRITE(13,*) ‘#’
      WRITE(13,*) ‘# Data for mesh plotting...’
      WRITE(13,*) ‘# w=’,w
      WRITE(13,*) ‘#’
      WRITE(13,*) char(13)
      call hprepit
      call hdefwin(1,0.0,d,0.0,w,50,750,50,550)
      call hsetwin(1)
      call hwinfram(’ ‘)
      do 10 i=1,ne
      call hmoveto(xc(node(i,3)),yc(node(i,3)))
      do 11 j=1,3
      call hsetcol(7)
      call hlineto(xc(node(i,j)),yc(node(i,j)))
c AJ bit in here ************************************************************
       write(13,666) xc(node(i,j)),yc(node(i,j))
  666  FORMAT(1X,F12.6,2X,F12.6)
c **************************************************************************
      npx=hrsx(xc(node(i,j)))
      npx=npx-(npx-400)/35
      npy=hrsy(yc(node(i,j)))
      npy=npy-(npy-250)/40
      call hmovpix(npx,npy)
      write (file,’(i2)’) node(i,j)
      call hsetcol(2)
      call hrgwrite (file,1)
      call hmoveto(xc(node(i,j)),yc(node(i,j)))
   11 continue
c AJ bit in here ************************************************************
       write(13,666) xc(node(i,1)),yc(node(i,1))
       write(13,*) char(13)
c **************************************************************************
      call hmoveto
     1(0.3333333*(xc(node(i,1))+xc(node(i,2))+xc(node(i,3))),
     2 0.3333333*(yc(node(i,1))+yc(node(i,2))+yc(node(i,3))))
      write (file,’(i2)’) i
      call hsetcol(3)
      call hrgwrite(file,1)
   10 continue
      call hsetcol(1)
      call hmoveto(0.0,w)
      call hlineto(d,w)
      call hmoveto(0.0,r(1))
      do 12 m=1,100
      x=0.01*r(1)*float(m)
      y=sqrt(dim((r(1)*r(1)),(x*x)))
   12 call hlineto(x,y)
      call hsetwin(-1)
      call hmovpix(300,10)
      call hrgwrite(’Press any key to continue’,1)
   13 continue
      if (hasckey().eq.0) go to 13
      call hfinish
       WRITE(13,*) char(13)
      return
      end
      subroutine trace(nn,ne,node,xc,yc,icon,ncon,vx,vy)
      dimension node(70,3),xc(50),yc(50),icon(70),ncon(70,2),
     +vx(70),vy(70)
      logical f1,f2
      integer hasckey
      common /bkdata/ r(5),w,d,qq
c AJ bit
      WRITE(13,*) ‘#’
      WRITE(13,*) ‘# Data for streamline plotting...’
      WRITE(13,*) ‘# w=’,w
      WRITE(13,*) ‘#’
c
      call hprepit
      call hdefwin(1,0.0,d,0.0,w,50,750,50,550)
      call hsetwin(1)
      call hsetcol(7)
      call hwinfram(’ ‘)
      do 9 i=1,ne
      call hmoveto(xc(node(i,3)),yc(node(i,3)))
      do 9 j=1,3
      call hlineto(xc(node(i,j)),yc(node(i,j)))
    9 continue
      do 10 m=1,20
      x0=d
      y0=0.05*w*float(m)
      call hsetcol(4)
      call hmoveto(x0,y0)
c AJ bit in here ************************************************************
      write(13,*) char(13)
      write(13,666) x0,y0
  666  FORMAT(1X,F12.6,2X,F12.6)
c ******************************************************************************
      do 100 n=1,ne
c  Identify the starting element for the path trace
      if (icon(n).ne.0) then
        if (y0.le.amax1(yc(ncon(n,1)),yc(ncon(n,2)))
     1 .and.y0.ge.amin1(yc(ncon(n,1)),yc(ncon(n,2)))) go to 101
      endif
  100 continue
  101 nc1=ncon(n,1)
      nc2=ncon(n,2)
           if (node(n,1).ne.ncon(n,1).and.node(n,1).ne.ncon(n,2)) then
        n3=node(n,1)
        n1=node(n,2)
        n2=node(n,3)
      else if (node(n,2).ne.ncon(n,1).and.node(n,2).ne.ncon(n,2)) then
        n3=node(n,2)
        n1=node(n,3)
        n2=node(n,1)
      else
        n3=node(n,3)
        n1=node(n,1)
        n2=node(n,2)
      endif
c  The entry nodes to the element are N1 and N2
c  Find the exit face and co-ordinates
c  The intersection with the face (1,3)
c  Test to see if face is more nearly horizontal or vertical
  110 if (abs(xc(n1)-xc(n3)).gt.abs(yc(n1)-yc(n3))) then
        xx=vy(n)*(xc(n3)-xc(n1))-vx(n)*(yc(n3)-yc(n1))
c  Horizontal face
        if (abs(xx).gt.1.0e-10) then
          x13=((xc(n3)*yc(n1)-xc(n1)*yc(n3))*vx(n)+
     1    (x0*vy(n)-y0*vx(n))*(xc(n3)-xc(n1)))/xx
          y13=yc(n1)+(yc(n3)-yc(n1))/(xc(n3)-xc(n1))*(x13-xc(n1))
c  Check for an allowed intersection
          f1=(x13.le.amax1(xc(n1),xc(n3)))
     1  .and.(x13.ge.amin1(xc(n1),xc(n3)))
        else
          f1=.false.
        endif
      else
c  Vertical face
        yy=vx(n)*(yc(n3)-yc(n1))-vy(n)*(xc(n3)-xc(n1))
        if (abs(yy).gt.1.0e-10) then
          y13=((yc(n3)*xc(n1)-yc(n1)*xc(n3))*vy(n)+
     1    (y0*vx(n)-x0*vy(n))*(yc(n3)-yc(n1)))/yy
          x13=xc(n1)+(xc(n3)-xc(n1))/(yc(n3)-yc(n1))*(y13-yc(n1))
c  Check for an allowed intersection
          f1=(y13.le.amax1(yc(n1),yc(n3)))
     1  .and.(y13.ge.amin1(yc(n1),yc(n3)))
        else
          f1=.false.
        endif
      endif
c  The intersection with the face (2,3)
c  Test to see if face is more nearly horizontal or vertical
      if (abs(xc(n2)-xc(n3)).gt.abs(yc(n2)-yc(n3))) then
c  Horizontal face
        xx=vy(n)*(xc(n3)-xc(n2))-vx(n)*(yc(n3)-yc(n2))
        if (abs(xx).gt.1.0e-10) then
          x23=((xc(n3)*yc(n2)-xc(n2)*yc(n3))*vx(n)+
     1    (x0*vy(n)-y0*vx(n))*(xc(n3)-xc(n2)))/xx
          y23=yc(n2)+(yc(n3)-yc(n2))/(xc(n3)-xc(n2))*(x23-xc(n2))
c  Check for an allowed intersection
          f2=(x23.le.amax1(xc(n2),xc(n3)))
     1  .and.(x23.ge.amin1(xc(n2),xc(n3)))
        else
          f2=.false.
        endif
      else
c  Vertical face
        yy=vx(n)*(yc(n3)-yc(n2))-vy(n)*(xc(n3)-xc(n2))
        if (abs(yy).gt.1.0e-10) then
          y23=((yc(n3)*xc(n2)-yc(n2)*xc(n3))*vy(n)+
     1    (y0*vx(n)-x0*vy(n))*(yc(n3)-yc(n2)))/yy
          x23=xc(n2)+(xc(n3)-xc(n2))/(yc(n3)-yc(n2))*(y23-yc(n2))
c  Check for an allowed intersection
          f2=(y23.ge.amax1(yc(n2),yc(n3)))
     1  .and.(y23.lt.amin1(yc(n2),yc(n3)))
        else
          f2=.false.
        endif
      endif
c  The actual intersection has the larger value of X less than X0
c  If there are allowed intersections with both faces, one must be
c  close to point on entry, or both at the node point at exit
      if (f1.and.f2) f1=x13.lt.x23
      if (f1) then
c  Next entry face (1,3)
        xx=x13
        yy=y13
        n2=n3
      else
c  Next entry face (2,3)
        xx=x23
        yy=y23
        n1=n3
      endif
c  Draw path increment
      call hlineto(xx,yy)
c AJ bit in here ************************************************************
      write(13,666) xx,yy
c ***************************************************************************
      do 120 i=1,ne
      if (i.ne.n) then
        if  ((n1.eq.node(i,1).or.n1.eq.node(i,2).or.n1.eq.node(i,3))
     1  .and.(n2.eq.node(i,1).or.n2.eq.node(i,2).or.n2.eq.node(i,3)))
     2  go to 121
      endif
  120 continue
c  Check to see if edge of mesh is reached
      go to 10
c  Move to next element
  121 n=i
      n3=(node(n,1)+node(n,2)+node(n,3))-(n1+n2)
      x0=xx
      y0=yy
      go to 110
   10 continue
      call hsetwin(-1)
      call hmovpix(300,10)
      call hrgwrite('Press any key to continue',1)
   12 continue
      if (hasckey().eq.0) go to 12
      call hfinish
       WRITE(13,*) char(13)
      return
      end
      subroutine press(ne,vx,vy)
      dimension vx(ne),vy(ne)
      integer hasckey
      common /bkdata/ r(5),w,d,qq
c AJ bit
      WRITE(13,*) '#'
      WRITE(13,*) '# Data for pressure plot (theta,press)'
      WRITE(13,*) '# w=',w
      WRITE(13,*) '#'
      WRITE(13,*) char(13)
      q2=qq*qq
      p=(q2-vx(12)*vx(12)+vy(12)*vy(12))/q2
      p=amin1(-3.0,float(int(p)-1))
      call hprepit
      call hsetcol(7)
      call hdefwin(1,0.0,1.570796327,p,1.0,150,650,150,450)
      call hsetwin(1)
      call hplotax('Theta','Press')
      call hsetcol(2)
      call hmoveto(0.0,1.0)
      do 1 n=2,12,2
      v2=vx(n)*vx(n)+vy(n)*vy(n)
      p=(q2-v2)/q2
      theta=0.130899693*float(n-1)
      call hlineto(theta,p)
c AJ bit in here ***********************************************************
      write(13,666) theta,p
  666  FORMAT(1X,F12.6,2X,F12.6)
c ******************************************************************************
    1 continue
      call hsetcol(1)
      call hmoveto (0.0,1.0)
      do 10 m=1,100
      th=1.570796327e-2*float(m)
      pr=1.0-4.0*sin(th)**2
      call hlineto(th,pr)
   10 continue
      call hsetwin(-1)
      call hmovpix(300,10)
      call hrgwrite('Press any key to continue',1)
   11 continue
      if (hasckey().eq.0) go to 11
      call hfinish
      WRITE(*,*) char(13)
      return
      end
      subroutine decomp(n,nb,a)
      dimension a(50,15)
c
c  This subroutine decomposes the symmetric matrix A in an upper
c  triangular matrix U and its transpose by the Cholesky method.
c  The symmetric matrix is stored as a band matrix of width NB and
c  length N in the upper right hand triangle with the diagonal
c  elements stored in A(I,1).
c  The resultant matrix is stored in A.
c
c  The sums are performed in double precision
      double precision diff
      a(1,1)=sqrt(a(1,1))
      do 5 k=2,nb
    5 a(1,k)=a(1,k)/a(1,1)
      do 25 k=2,n
      kp1=k+1
      km1=k-1
c  Calculate the diagonal elements
      diff=a(k,1)
      do 10 jp=1,km1
      icol=k+1-jp
      if (icol.le.nb) diff=diff-a(jp,icol)*a(jp,icol)
   10 continue
      a(k,1)=sqrt(diff)
      do 20 j=2,min0(nb,(n+1-k))
c  Caculate the off-diagonal elements
      diff=a(k,j)
      do 15 jp=1,km1
      icol=k+1-jp
      jcol=k+j-jp
      if ((jcol.le.nb).and.(icol.le.nb)) diff=diff-a(jp,icol)*a(jp,jcol)
   15 continue
   20 a(k,j)=diff/a(k,1)
   25 continue
      return
      end
      subroutine solve(n,nb,m,a,b)
      dimension a(50,15),b(50,1)
c
c  The Cholesky decomposed matrix A is used to calculate the solution
c  vectors for sert of right-hand side vectors B.
c  The decomposed matrix is stored as a band matrix of width NB and
c  length NN in the upper right hand triangle with the diagonal
c  elements stored in A(I,1).
c  The calculation is performed for M solution vectors stored in B.
c
      double precision diff
      do 100 j=1,m
c  The first pass
      b(1,j)=b(1,j)/a(1,1)
      do 30 i=2,n
c  Scan through the range forwards
      diff=b(i,j)
      do 20 k=2,nb
c  Add in the contributions from the lower elements already calculated
      irow=i+1-k
      icol=i+1-irow
c  Check the element of A exists
      if ((irow.ge.1).and.(icol.le.nb)) diff=diff-a(irow,icol)*b(irow,j)
   20 continue
      b(i,j)=diff/a(i,1)
   30 continue
c  The second pass
      b(n,j)=b(n,j)/a(n,1)
      do 60 ii=2,n
c  Scan through the range backwards
      i=n+1-ii
      diff=b(i,j)
c  Add in the contributions from the upper elements already calculated
      do 50 k=2,nb
      ik=i-1+k
c  Check the element of A exists.
      if (ik.le.n) diff=diff-a(i,k)*b(ik,j)
   50 continue
      b(i,j)=diff/a(i,1)
   60 continue
  100 continue
      return
      end
      subroutine veloc(nn,ne,node,xc,yc,pload,vx,vy)
      dimension node(70,3),pload(50,1),xc(50),yc(50),vx(70),vy(70)
c
c  Subroutine to calculate the constant velocity components VX and VY
c  in each element from the nodal values of the potential.
c
      do 1 i=1,ne
c  Identify each node - its postion and geometrical factors
      n1=node(i,1)
      n2=node(i,2)
      n3=node(i,3)
      xi=xc(n1)
      xj=xc(n2)
      xk=xc(n3)
      yi=yc(n1)
      yj=yc(n2)
      yk=yc(n3)
c  Calculate the shape matrix coefficients bi and ci for each node
      bi=yj-yk
      bj=yk-yi
      bk=yi-yj
      ci=xk-xj
      cj=xi-xk
      ck=xj-xi
c  Calculate the velocity components from the derivatives
      vx(i)=(bi*pload(n1,1)+bj*pload(n2,1)+bk*pload(n3,1))/
     1(bi*xi+bj*xj+bk*xk)
      vy(i)=(ci*pload(n1,1)+cj*pload(n2,1)+ck*pload(n3,1))/
     1(ci*yi+cj*yj+ck*yk)
    1 continue
      return
      end
```