A User's Guide To The Felisa-Nektar System

By

Nicholas Hubscher
Aeronautics III

Department of Aeronautical Engineering
Imperial College of Science, Technology and Medicine

June 1997

Written with the help and advice of Dr. S. Sherwin
FOREWORD
During work on my project, looking at numerical simulations of flow over low speed aerofoils, I found that I had to spend considerable amounts of time learning how to use the Felisa-Nektar system and becoming familiar with its intricacies. It struck me that the system, although very powerful and flexible, was by nature a rather complex tool use. I felt that I had been spending unnecessarily large amounts of time, which could have been used producing important results, becoming familiar with its workings instead. There was nothing inherently difficult in learning to use the system but the lack of information on details such as the uses of the various files, what were reasonable parameter values and how to use the different programs did prove to be a significant hindrance. It is for this reason that I have written this guide providing information, which will allow a new user to get up to speed more quickly. I am sure that having had it at my disposal would have helped me greatly.

N. W. Hubbscher
June 1997
CONTENTS

Title Page Page 1
Foreword Page 2
Contents Page 3
Introduction Page 4
The Felisa-Nektar System Page 5

1) Creating an aerofoil shape Page 6

2) Felisa and Mesh Generation Page 7
   - 2.1 Creating the .dat file Page 8
   - 2.2 Creating the .bac files
     - 2.2.1 Point Source Page 9
     - 2.2.2 Line Source
   - 2.3 mg2d and creating the .grid file
   - 2.4 xp2d and viewing the mesh
   - 2.5 Creating the .body file
   - 2.6 pet2nek2d and creating the .rea file
   - 2.7 nektool2d and manipulating the mesh

3) Nektor and flow computation Page 10
   - 3.1 nektar2d and creating the output files
   - 3.2 nektar3d and 3D simulations
   - 3.3 Running Nektor with Codine
   - 3.4 force2d and viewing force vs. time plots
   - 3.5 vort2d, nek2tec2d and creating the .tec file Page 11
   - 3.6 The .ring file and its use

4) Flow images and making movies Page 13
   - 4.1 Tecplot and visualising the flow-field
   - 4.2 Making and playing movies
   - 4.3 Recording videos of movies

Bibliography Page 14
Appendix A Page 15
Appendix B Page 16
INTRODUCTION

The intention of this document is to give an overview of the Felisa-Nektar system. It is aimed to be a user-friendly manual that will be of assistance to a student who is learning to use the system, or to a beginner. I have tried to make explanations of the various files and programs as simple as possible. Reference to existing projects using the system should also provide insight as to how the system is used in practice.

By following this guide, a user unfamiliar with the system’s intricacies should be able to produce meaningful results with confidence. This guide is not an explanation of the complex workings of the code itself. However, it deals with each stage, step by step, with useful pointers for avoiding commonly encountered problems and achieving good results.

In this document I have assumed that the user is looking to produce data referring to aerofoils in flow-fields and tailored the guide to this end. However, a user examining different objects will still find the information in this document valuable.

SYSTEM OVERVIEW

The following page contains a diagram of the system and its components, which is constantly referred to throughout this document. In short, the system comprises of two main programs, which combine to give a very flexible CFD package.

The first part is Felisa, a mesh generator written by Dr. J. Piero. It has its own thorough, if slightly rigorous documentation, which can be found in Appendix A. This guide should be referred to in conjunction with that documentation.

The other part of the system is Nektar, a Navier-Stokes solver written by Dr. S. Sherwin. Its normal use and functions are explained thoroughly in this guide.

The other components of the system are linking programs and programs for interpreting the system’s output.
The Felisa-Nektar System

.bac → .dat
  ▼
  mg2d
  ▲
  .gri
  ▼
  xp2d
  ▼
  .h
  ▼
  fel2nek2d -m .bdy -o .rea .gri
  ▲
  nektar2d (-chk) -r10 -t0.0 .rea
  ▲
  nekttool2d -r .rea
  ▲
  .his
  ▼
  .fld/.chk
  ▼
  force2d .f
  ▲
  vort2d -r .rea -o .tec .fld/chk "or"
  ▲
  nek2tec2d -r .rea -o .tec .fild/chk
  ▲
  .tec
  ▼
  preplot .tec
  ▲
  .plt
1) Creating an aerofoil shape

This section is not strictly pertinent to the Felisa-Nektar system. However, it will significantly assist a user in creating necessary aerofoil data at various incidences. Its inclusion can thus be justified on the grounds that aerofoils are a very common application of the system.

Let us consider, for example, the creation of a mesh for a NACA0012 aerofoil. To input its geometry into the .dat file we first need to know the locations of a number points along its upper and lower surfaces. The following equation gives this data for a NACA0012 aerofoil:

\[ y = \pm \frac{12}{0.2} \left( 0.2969\sqrt{x} + 0.126x - 0.3516x^2 + 0.2843x^3 - 0.1015x^4 \right) \]

If the need arises to create meshes for this aerofoil at different incidences, we can perform a simple trigonometric rotation of the points on the aerofoil surfaces using a spreadsheet.

\[ x = r \cos \theta \]
\[ y = r \sin \theta \]

A spreadsheet similar to the one shown below can be used to give co-ordinates of points on the aerofoil surfaces at different incidences.

<table>
<thead>
<tr>
<th>Incidence</th>
<th>x</th>
<th>y</th>
<th>r</th>
<th>Angle</th>
<th>New x</th>
<th>New y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-5.0000</td>
<td>0.1000</td>
<td>0.0468</td>
<td>0.1104</td>
<td>25.0926</td>
<td>0.0955</td>
<td>0.0554</td>
</tr>
<tr>
<td>-6.0000</td>
<td>0.2000</td>
<td>0.0574</td>
<td>0.2201</td>
<td>16.0970</td>
<td>0.1942</td>
<td>0.0746</td>
</tr>
</tbody>
</table>
2) *Felisa* and Mesh Generation

*Felisa* requires two files to create a mesh. Roughly speaking the .dat file contains details of the boundary of the mesh and the object(s) within it. The .bac files contains data relating to the size and nature of the mesh elements themselves.

2.1 Creating the .dat file

The purpose of the .dat file is the definition of the geometry of the boundaries of the flow regime and any objects, such as aerofoils, within it. Care must be taken to enter data in the correct format and replacing numbers in an existing file rather than writing the file from scratch best serves this purpose.

An example of the file for a NACA0012 at 0° incidence is shown on the right. The values should be compared to those indicated in the *Felisa* manual.

`nreq` should always be kept at 1. `nfn` is the total number of points used in describing the flow boundary and shapes within it. The boundary (assuming it is rectangular) should therefore contribute 4 points to this total, with the remainder coming from points on the aerofoil itself. `nbs` is the total number of independent curves which form the boundaries of the flow-field and the objects within it. The number 7 comes from the 4 edges of the flow-field, plus the trailing edge of the aerofoil which is straight of finite length and the upper and lower surfaces.

Next, co-ordinates of the points that lie on the curves or which form the flow boundaries are entered against arbitrary ID numbers.

Numerical interpolation in the code will produce a best curve between these points and in order to produce accurate aerofoil surfaces it is best to use at least 15 points per surface.

<table>
<thead>
<tr>
<th>nreq</th>
<th>nfn</th>
<th>nbs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>18</td>
</tr>
<tr>
<td>19</td>
</tr>
<tr>
<td>20</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>23</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>25</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>29</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>34</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>36</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>38</td>
</tr>
<tr>
<td>39</td>
</tr>
</tbody>
</table>

**BOUNDARY segments**

1 2 0 1
36 37
2 2 0 2
37 39
3 2 0 3
38 39
4 0 2 0 1
39 35
5 2 0 4 19 35
6 1 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 7 10 0 3
1 2 3 4 5 6 7 8 9 10 11 11 13 14 15 16 17 18

**region**

1 7
1 2 3 4 5 6 7
The boundary segments are then described. A boundary segment is a curve or line that forms part of the flow boundary or the shape around which the flow is being modelled. Each segment is described by four variables, the first is its own ID number (each subsequent curve's ID number should increment by one). The second variable is the total number of points that are being used to describe the curve—a straight line would thus need only 2. The smooth curve of an aerofoil surface obviously requires more. The third variable should be 0 for all cases. The value of the fourth variable is important when the mesh is used as input to the Nekter code. This number should match up with the number in the .bldy file that describes the role of that particular curve in the flow-field. See the description of the .bldy file for more details on this matter. The line underneath these four variables lists the ID numbers of the points that lie on and thus describe the curve. Care must be taken to follow the convention in which segments and the points within them are listed in an anticlockwise sense if they form the flow boundaries and clockwise if they form an object in the flow. This is explained in more detail in the Felisa handout.

Finally, the region definitions follow. The first variable always has the value 1 and the second is equal to the total number of component segments (nbs), in this case 7. On the line below, the ID numbers of these curve segments are listed.

### 2.2 Creating the .bac file

This file is concerned with describing the detail of the mesh to be formed inside the flow-field. It is simple but gives the user power to vary areas of mesh density.

An example of the file for a NACA0012 at 0° incidence is shown on the right. Again, reference should be made to the relevant drawings and diagrams in the Felisa manual.

<table>
<thead>
<tr>
<th>np</th>
<th>ne</th>
<th>n1</th>
<th>n2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-5</td>
<td>-5</td>
<td>1.0 1.0 0 1.1 1.0</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>-5</td>
<td>1.0 1.0 0 1.1 1.0</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>5</td>
<td>1.0 1.0 0 1.1 1.0</td>
</tr>
<tr>
<td>4</td>
<td>-5</td>
<td>5</td>
<td>1.0 1.0 0 1.1 1.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

source point

1. 0. 0. 0.1 0.3 0.5

course line

1. 0.9 0.0 0.1 0.3 .5

4.0 0.0 0.1 0.3 .5

np is the number of background mesh nodes. The best way to create the mesh, assuming the flow region to be rectangular is to place the mesh directly over this flow-field. This value will be 4 since each node will then correspond with a corner of the flow-field. ne is the number of background mesh elements, best always kept as 2. The two remaining values, for nbs and nls, give the total number of point and line sources respectively that the user wishes to define. Details of these sources are described later on in the file.

The lines below describe the boundary lines of the mesh area. As mentioned above, it is simple and easy to follow the edges of the flow-field. The first two numbers in the line, after the ID number of the point, are the x and y co-ordinates respectively of this edge. The following two digits locate the x direction in vector terms (1,0) and should always read 1 and 0. The next column of number describes the distance between mesh nodes emanating from the point in the direction defined. So, a value of 0.25 in this column will guarantee a mesh node every 0.25 units in the x-direction. The next three columns set up the y direction. Hence, 0 and 1 followed by a mesh spacing on the mesh boundary in the y direction of 0.5 would give half unit spacing in this direction.

Obviously, it is impossible for requests for different spacing in the same direction for the same boundary line from two different lines of the file to be accurately granted! In this case, the code will either give a fair balance between the requests or in some cases may crash, failing to produce a mesh altogether.

The next line in the file defines the triangle's nodes. The first value is the incremental triangle ID, and the next three are its node points. Again, the diagram in the Felisa manual better explains this.
The file now requires details of any point or line sources:

2.2.1 Point source
A point source is a point from which dimensions of triangles are defined. For example if we need to know the flow in fine detail around a particular point, we will want to ensure there are enough triangles in the area to ensure that a good resolution is realised.
The details of each source are described underneath the point's ID number. In the case of our aerofoil, good resolution compared with the remainder of the flow-field is required around the leading edge.
The first two values are the x and y co-ordinates of the source, in our case 0 and 0 (the location of our leading edge). The next column describes the length of triangles at that point. The next column gives the distance over which that length of triangles is to be maintained; in our case a length of 0.3. The final column gives the distance from the point by which the spacing should have increased to twice the spacing adjacent to the point. In our case we have chosen 0.5, a relatively slow increase in triangle size.

2.2.2 Line source
A line source definition is based along the same lines as the point source. A line source can be considered as two point sources, only the effect of the sources on surrounding triangles only occurs in the region between the two points. In our aerofoil we require good resolution in the wake of the flow, and around the aerofoil surfaces so a line source beginning just before the trailing edge and terminating 3 aerofoil lengths downstream is of benefit. Triangle sizes required along this line are defined at each end in exactly the same manner as the point source. When conflict arises in demanded triangle sizes, the code will give a "best fit" distribution.

2.3 mg2d and creating the .gri file
Creating the mesh itself involves running the program mg2d as can be seen from the Felisa-Nektar flow-chart. This program uses the .dat and .bac files described above to create a .gri file which contains the details of the mesh. The command line for this program is simply mg2d and the program requires the name of the mesh. (eg. NACA12 if the files are called NACA12.dat and NACA12.bac).

The program will inform the user of how many elements have been created and one should aim to create meshes with less than 1000 elements, otherwise the user will experience unnecessary delays in the later stages of computation. The number of elements can be altered by changing the intensity of elements both at the flow boundaries and at the sources. As mentioned earlier, some conflicts of meshing requirements may cause the program to run unsuccessfully. Altering the mesh characteristics may help to avoid this.

The physical content of the .gri file is of no real use to the user and cannot be reasonably altered manually.

2.4 xp2d and viewing the mesh
The mesh can be viewed using the program xp2d. Once this program has been started, it requests the input of the name of the mesh file. This is the .gri file (the product of mg2d), for example NACA12.gri.

In short, to view the mesh the user should respond with "no" to all of the program's initial questions and then select the option to view the file. Commands for zooming, unzooming, and printing the mesh are displayed by the program and are useful for checking that elements are of the desired size in the right places.
An example of the mesh created by the above files can be seen below:

The mesh, as seen, can be manually altered further at a later stage once it has been incorporated into a .rea file. The powerful application nekttool2d allows the user to alter fine details of the mesh and is described later on.

2.5 Creating the .bdy file
This file is needed in creating the .rea file, which is the input for the Nektar code. This file along with the program fel2nek2d described later forms the link between Felisa and Nektar.

The .bdy file informs the code of the action of the flow on the various mesh boundaries and also explains the requirement for the so far unexplained column in the curve entries in the .dat file.

Each of the solid lines or curves forming the mesh boundary or some object within it has some effect on the flow. The aerofoil surfaces and trailing edge are walls where fluid velocity is assumed to be zero. The upstream and upper and lower boundary walls all have the same velocity of fluid through them. The downstream boundary wall will, however, see a range of velocities. Each type of boundary has a code associated with it to inform Nektar of its function.
It is again best to alter existing .bdy files and they are fairly self-explanatory:

Region 1 is a velocity region "V". In our example we assume steady uniform flow in the x direction only at the upstream, top and bottom boundaries of the mesh, where flow is considered to be sufficiently far away from the aerofoil as to be unaffected by its presence. To ensure no conflicts result from boundary proximity, the object in the flow should be at least 5 chord lengths behind,
below and above these boundaries. The values 1.0 and 0.0 indicate the vector direction of the flow at these points. The corresponding code at these boundaries in the .dat file is "1".

Region 2 is an outflow region "O". This is the downstream boundary and will experience a range of velocities across it. There is no minimum distance at which it should fall behind the aerofoil, the only criterion is the length of wake the user wishes to visualise. The corresponding code for these boundaries in the .dat file is "2".

Region 3 is a curved wall region "W". Nektar makes a distinction between curved and straight walls since curved walls require further clarification as can be seen at the bottom of this file. In the case of the aerofoil, the co-ordinates of leading and trailing edge are required, along with its thickness. The corresponding code for these boundaries in the .dat file is "4".

Region 4 is a straight wall region also "W". Unlike the curved wall, this needs no further clarification. The corresponding code at these boundaries in the .dat file is "4".

2.6 fel2nek2d and creating the .rea file

fel2nek2d creates the .rea file from the .bay and .gri files. The .rea file contains the necessary data for Nektar to run. An example is shown on the right. The majority of the .rea file is created automatically, but those parameters that determine the specific output of the Nektar code must be completed manually once fel2nek2d has run and are described below:

KINVIS: is the kinematic viscosity of the fluid in the flow. This is simply 1/Re, since ρ is assumed to be 1. So in this case, Re = 1000.

DT & NSTEPS: It is best at this stage to explain the concept of time units and time-steps:

In our flow we are interested in non-dimensional time. The best way to understand this is that one time unit is the time taken for the flow to cover the distance of the chord length of the aerofoil (assuming that this is of unit length in the mesh). In oscillating flows it usually takes around 5 time units before oscillations begin and steady oscillations are seen after around 11 time units. Hence, for a simple run, a period of around 20 time units is the best to aim for. The product of the time-step and the number of time-steps required advise the code of the required number of time units:

\[
\text{time units} = \text{time-step} \times \text{number of steps} = DT \times NSTEPS
\]
DT, the time-step, is the length of time in between which the code calculates physical values at points in the mesh. If it is 0.00125, it will recalculate data at points each time the flow moves a further 0.125% of the chord length.

Time-step selection is probably the most difficult part of using this system. For a given requirement of time units, too short a time-step will result in an unnecessarily lengthy computation, since NSTEPS (see later) will be consequently high. Too high a time-step, and the computation will crash, as incorrectly high velocity values are initially produced around the aerofoil's leading edge. In relation to the potential time lost to the user as a result of a poor selection, a short trial and error experiment to find the correct value is worth making before starting the computation proper. An explanation of how to do this can be found in the section explaining the use of nektd.

The main factors effecting the choice of DT (and flow speeds around leading edges) are the Reynolds number of the flow, polynomial order of the computation (see later) and the incidence of an aerofoil in the flow. Mesh resolution can also be a factor. As Re increases, DT must usually also decrease. Roughly, if Re doubles, DT should halve. An increase by 2 in the polynomial order NORDER (see later), usually also requires a halving of DT. The increase of incidence of an aerofoil in the flow usually also requires a drop in DT for successful start-up. Mesh resolution can effect DT in both directions, for a given DT both a finer or courser mesh maybe required to aid a successful start-up. Again, there are no recommended values for DT's and trial and error is the best method to find the best one. However, as a very rough guide for a 20 time unit requirement, a 0° incidence aerofoil at Re=1000 and polynomial order 5 would be best run at DT=1.25×10⁻³. A 5° aerofoil at Re=1000 and polynomial order 7 would be best run with DT=6.25×10⁻⁴.

So, NSTEPS, as previously explained, is simply (time units)/DT. Again, it is best kept as low as possible to reduce computation time. This is obviously achieved by making DT as high as possible.

NORDER: This is the polynomial order that the code uses. The higher the polynomial order, the better the resolution of the flow (i.e., physical properties will be more consistent throughout the flow-field). In practice only odd numbers can be effectively used and thus sensible values are limited to 5 or 7. 7 will give better results.) However, as a result of it frequently forcing the user to reduce the time-step (DT), it may not always be preferable. In some cases, such as those with high Re and incidences, this may make the computation time unacceptably long, and in other cases, only polynomial order 5 will only work without crashing.

IOTSTEP: This value determines after how many time-steps Nektar dumps the physical values throughout the mesh at that point in time into an output file. There are two types of output files that can be created. One is a check file (.chk) and the other a field file (.fld). If a check file is required, this is specified in the Nektar command line (see later). A .chk file gives data for the mesh every IOTSTEP time-steps and is replaced each time with the fresh dump. The .fld file, however, accumulates the dumps and they can be extracted individually later. For example if NSTEPS=16000 and IOTSTEP=1000, the .fld file will contain 16 dumps at 1000, 2000, 3000 time-steps etc. Useful data begins to emerge after around 0.5 time units, so if a check of the output is required on a run of NSTEPS=16000, IOTSTEP should be set at around 400, depending on the exact value of DT. For the purposes of making movies of flows—movies are a sequence of a number of individual picture frames—a .fld file should be used to retain the flow data over the time period. A reasonable quality movie should comprise of at least 15 dumps so IOTSTEP, in this case, could be set at 1000 giving 15 such dumps.

EOTYPE: This value determines the type of equation that Nektar is solving. It is best kept at 0 so that the code solves the rotational Navier-Stokes equations.

A User's Guide To The Freista-Nektar System

Page 12

Nicholas Hubscher
HISSTEP: This value determines how frequently details are dumped into the force (.fco) file. At every time-step Nektar calculates the forces on the aerofoil in the x and y directions. This file, amongst other things, is useful for checking the stability of the run (described later). Every HISSTEP a line is added to the .fco file. If it is the forces upon the aerofoil which are being examined, it is worthwhile having a low number, perhaps 2 in NSTEPS=16000, giving a good history of the forces on the aerofoil with time, which can themselves be plotted (see later). If the purpose of the run is to make movies and forces are of no interest, HISSTEP can be set in the region of 10 upward to avoid large .fco files. A reasonably low figure is still required to ensure the run can be monitored for stability during start-up. However, when the test run is terminated and the user is confident of stability this can be reset as high as NSTEPS, giving virtually no .fco file. With runs of high NSTEPS, HISSTEP should be kept relatively high, since a .fco file with HISSTEP=1 for NSTEPS=64000 is over 10Mbytes in size! HISSTEP also determines, in the same manner, how often data is added to the .his file (see later).

In certain circumstances one may wish to restart or continue with a run that has already began. In order to do this a .fld file or a .chk file from the previous run is required. It is for this reason that these files are worth keeping despite their potential size. The are a number of reasons for a restart being needed. Perhaps an initial condition, such as the Reynolds number has to be changed. We know that increasing the Reynolds number often requires a decrease in time-step to avoid initial problems. To avoid a permanently low time-step throughout the run, which will result in a long run time, we can restart after say 1000 time-steps with the higher value of Re. Another reason could be a crash of the computer systems, and the run can be restarted from the point of the last .chk file or .fld entry. Towards the end of the .rea file, which is shown heavily abbreviated above, “0 INITIAL CONDITIONS” must be changed to read “2 INITIAL CONDITIONS” and the following two lines must be inserted immediately underneath:

```
Restart...
NACA12.chk ... or NACA12.fld
```

This done, nktar2d can be restarted and run as normal. However, in the case of a change of Re it is important to allow a short time for the output to adjust to the change before taking output data as sacred!

2.7 nkttool2d and manipulating the mesh
As mentioned earlier, once incorporated into the .rea file, the mesh shape itself can be manipulated by the program nkttool2d to increase or decrease mesh resolution should this be required. This program is powerful, simple to use and a self-explanatory menu can be found by right clicking on the mesh once it has been displayed by the program. The main advantages are that more triangles can be created when greater mesh definition is required, usually around the leading edge and in the boundary layers of aerofoils. The location of individual vertices of mesh triangles can also be altered. Such mesh changes can also lead to an increase in the stability of the run along with giving improved results. In addition, nkttool2d visually indicates the different flow boundaries specified in the .bdy file, thus allowing the user to ensure that the code will be aware of the correct status of each boundary. It makes good sense to monitor whether or not this process is creating many more elements, since significant additions could slow down processing at a later stage.
3) Nektar and flow computation
Although aspects of Nektar's input and output data have already been discussed in some detail, the following components of the system relate to Nektar itself. They also deal with its specific output and the manipulation of this data to useful ends.

3.1 nekta2d and creating the .chk, the .fld, the .his and the .fco files
nekta2d creates these files which are described in some detail above during the course of its run. The option -chk indicates that .chk files, and not .fld files should be produced.

.chk and .fld files are used to create pictures and movies of the flow. This process is described later. Other than their headers, which explain at which time unit they were created or refer to, the user cannot manipulate or view the actual content of these files.

<table>
<thead>
<tr>
<th>Force acting on body</th>
</tr>
</thead>
<tbody>
<tr>
<td># Time (Px-press, Px-visc) Px</td>
</tr>
<tr>
<td>0.006000</td>
</tr>
<tr>
<td>0.010000</td>
</tr>
<tr>
<td>0.015000</td>
</tr>
<tr>
<td>0.020000</td>
</tr>
<tr>
<td>0.025000</td>
</tr>
</tbody>
</table>

.fco files, see example above, are text files which contain the x and y forces on the aerofoil at each time-step. In this case we can see the relatively large initial forces produced before they settle down to become more consistent. In viewing force/time plots (described later) it may be necessary to manually remove these lines containing high forces in order to hold the graph range within realistic limits.

.his files can be configured to contain history of physical data at given points in the mesh. The required points have to be advised individually in the .rea file. This function can be useful for monitoring flow at a given point over the time of the run rather than just at every IOSTEPS. The lines in the .rea file required to set up these dumps are not easy to explain and are specific to certain cases. If required, it is best to seek advice from Dr. Sherwin who wrote the code! Data dumps to the .his file will be made every HISSTEPS.

Running nekta2d successfully requires knowledge of its start-up and run procedures. As mentioned earlier, it is good practice to ensure that the run is stable. This can be achieved by starting the program in a shell, then monitoring the .fco file and the lines being added to it in another. If it is going to do so, the run usually produces incorrectly high forces in its initial stages, stay within around 100 time-steps depending on the length of each step. If the run fails, a shorter time-step should be selected. If the forces settle down (i.e. they begin to increase or decrease by an orderly time-step), this is a sign of stability.

Since nekta2d in the shortest run cases takes at least a day to complete a run, a program allowing the user to log off whilst the code is being run is necessary. This software is known as Codine and is described later.

It is extremely difficult to predict how long any given run will take to complete since this depends not only on the number of time-steps required (NISTEPS), but also upon the computing power available.
Also, if a machine is running other processes simultaneously processing will be consequently delayed. However, as a very rough estimate, a run of \texttt{NSTEMPS}=16000 on a machine with large memory capacity running no other jobs, could be expected to last for 20 hours. The length of the run would double with a doubling of \texttt{NSTEMPS} or as a result of a 50% reduction in machine CPU availability.

3.2 \texttt{nekto}2\texttt{d} and 3D simulations

Under certain circumstances a flow will be predominately 3D. This should occur around aerofoils with incidence greater than 8\degree. It has been found that the \texttt{nekto}2\texttt{d} code cannot resolve these flows accurately, and that the forces found by this program massively overestimate those that are actually found on aerofoils. The 3D version of the code, \texttt{nekto}F, and its associated application \texttt{vortF} can be used instead to resolve and interpret the data in these flows.

An example of the .rea file required for \texttt{nekto}F is shown on the right. Many factors are similar to those found in the 2D .rea file. The additional parameters are introduced to induce noise in all directions in the mesh and thus “kick” the flow out of its normal 2D pattern into 3 dimensions.

These parameters should be used for similar runs. The command line is the same as for \texttt{nekto}2\texttt{d}, only the following two switches should be used to try to give well defined 3 dimensional flow: -Z4 -R1000

Unfortunately the 3D code is very slow to run on a normal system and the above run took around two weeks to produce data for just 3 time units! Even in this case, the flow did not have the opportunity to change into 3 dimensions, which normally occurs at a far later stage.

3.3 Running \texttt{Nekto} with \texttt{Codine}

\texttt{Codine} is a job manager which runs programs for users whilst they are not logged on. Its use is important in running \texttt{Nekto} successfully and this is the reason for its inclusion in this document. The basic shell program on the right, which could be named \texttt{NACA12.cod}, instructs the \texttt{Codine} manager how to run the job. It is also worth referring to the \texttt{Codine} manual in Appendix B.

After writing the program (best modified from an existing one), the user must make sure it is executable by typing:

```
chmod +x NACA12.cod
```

In the program, the first line advises in which type of shell the program should be run. This should not be changed. The second and third request e-mails to confirm the beginning and completion of the run. The fourth line advises the manager to look for the executable program in the current directory. The fifth line stipulates the file to which, what would otherwise be visual output, should be sent. The machine and the area in which the program is to be run is then determined in the fifth line. The final line is the execution command of the program with all the necessary options, in this case to produce a .chk file.

The .out file can become large during the run and contains essentially unnecessary information. As soon as it has been created, it can be safely deleted.

A User’s Guide To The Felisa-Nekto System

Nicholas Hubscher
The are other important control commands for Codine.

`qstat`, typed at the prompt, lists all free computers so the user can select one on which to run. It also lists the jobs that are running and their ID numbers.

`qsub NACA12.cod` submits the job to the Codine manager.

`qdel ID#` stops a job running, although it will stop automatically once it has been completed.

It is also advisable to arrange an account on a machine with its own hard disk space, since running across the network can cause instability. More space can be allocated on a hard disk, and the job is best run in the batch (`b_computer`) queues.

3.4 force2d and viewing force vs. time plots

`force2d` can be used for plotting forces with time - the content of `.fce` files. An example of such a plot can be seen below. `force2d` runs on machine `cub`, but Dr. Sherwin must arrange a user account. This done, the user can type "sm" in a shell. They will then be invited to enter a command and typing `force2d NACA12.fce` at the prompt gives the plot.

On first viewing their `.fce` file the user will probably notice a large initial peak, caused by the high forces on start-up (mentioned previously). In order to bring the graph within the range of the remainder of the comparatively smaller forces, the lines at the beginning of the `.fce` files referring to these forces should be deleted.

The graph can be saved as a postscript file, if its inclusion was to be required in a document or for other purposes. After typing "sm", the user should enter the following commands:

```
dev postfile NACA0012.ps
force2d NACA0012.fce
devXI1
```

This will output the `NACA0012.ps` postscript file. Conversion to other picture file formats can be carried out with the `snapshot` application described later in the section on making movie videos.

It will be apparent to the user that forces usually settle into a steady pattern after around 10-11 time units. Any force data for the aerofoil should be taken from this point onwards. This plot is particularly useful for calculating C_l and C_D for aerofoils. Forces on the aerofoil can be averaged over a number of steady cycles if they are oscillatory or taken as the steady values if constant. The user has only to apply the lift and drag equations to find the desired co-efficient. If the speed of the flow has been set at 1ms^-1 (in the `.bdy` file) and the aerofoil chord length set at 1m (in the `.dat` file) this calculation is relatively simple. (Fluid density is assumed to be 1 Kgm^-3).
In some cases it may be useful to compare two force plots on the same set of axes. This can be done quite simply from the two .fce files to be plotted. The are simply mixed together to form a new file which is plotted instead. This can be done using the "cat" application by typing:

```
cat plot1.fce plot2.fce > comparison.fce
```

The new .fce file can then be plotted in the normal way as can be seen in the two quite similar plots on the right.

3.5 vor12d, nek2tec2d and creating the .tec file

The dumps of physical values in the .chk and .f1d files must be converted to a format that can be interpreted and then viewed by a data visualisation/graphics program. For this purpose we use Tecplot (described later), vor12d and nek2tec2d make up the first stage in this two-step process and either one program or the other is used. Both applications transfer the same basic information from the data files to the .tec file. However, vor12d also transfers information about vorticity in the flow-field.

Consequently its output files are slightly larger, this being the only reason for the user choosing to use nek2tec2d instead.

Vorticity is a useful property to plot in the flow-field. It describes the rotation of a fluid element at a given point in the field. Hence it highlights boundary layers and vortices themselves very clearly. The other property giving a comprehensive view of events is the pressure distribution (produced by both programs), although vorticity is clearly the better property to plot. Details concerned with the actual plotting of flows follow later in the section on Tecplot.

To run these programs, the commands described in the system overview will suffice in the case of interrogating .chk files. However, since each dump in a .f1d file refers to a different point in time in the flow, these dumps have to be extracted into separate files to make multiple .tec files (ie one for each dump in the .f1d file). This is best achieved by running the fairly self-explanatory shell program on the right, NACA0012.mov.

```
#!/bin/sh
set i = 1
while (i <= 36)
vor12d -R -m N12.rng -r N12.rea -$5i N12.f1d | preplot -N12_i.plt
@ i+=1
end
```

The first line specifies the shell in which the program should run. The next line sets up a basic loop. The third line should be adjusted so that i<= the number of dumps in the .f1d file.

The fourth line invokes either nek2tec2d or vor12d as requested with the usual options, in this case including a range option (described later). Since we are interrogating a .f1d file, the -$5i switch selects the particular dump we wish to process, since dumps are numbered sequentially. The "|" symbol is known as a pipe and it indicates that the output of the previous program should be piped directly into the next. The next application is preplot, which is described later, and whose output is, in this case, is requested to form sequentially numbered .plt files.
The fifth line completes the loop, and the program will progress to extract the next dump in the .fld file and produce a .plt file from it.

3.6 The .rnx file and its use
An example of a .rnx file can be seen on the right. It is a basic file that indicates the regions of the flow-field to be extracted from .fld or .chk files. Its use is optional and if the switches activating it (R -m NACA12.rnx) are not included in the command line, output will include data for the full area of the specified flow-field. However, if the user is sure that not all the flow-field is of interest, then in order to speed up future processing of the data and save space, parts can be omitted. The .rnx file simply gives the range required in the x direction of the field, in this case from -1.5 to 4, followed by the requirement in the y direction on the following line.

3.7 preplot and creating the .plt file
To convert the .tec file into the .plt file, which is the required input of Tecplot, the data visualisation program, the program preplot is used. The command line is as shown in the system diagram.
As described in the section above, the use of preplot is often controlled by the .mov program for the convenience of making the large number of .plt files required to make a movie with relative ease.

4) Flow images and making movies
Now that the flow has been calculated and resolved throughout the mesh, it must be changed into images from which it can be viewed either as stills or pieced together to make movies.

4.1 Tecplot and visualising the flow-field
Tecplot is a data visualisation program that converts .plt files into viewable images that can be then manipulated. It is difficult to describe the use of this application in a document such as this, but fortunately the program is quite user friendly. Here are a few hints that may help to initially familiarise a user with the application:
.plt files are loaded from the Load Datafiles menu item. Once loaded, to view a property in the flow, the Contour button on the upper-panel should be chosen, giving the choice of properties to plot. Vorticity is a useful property to select if vor12d has been used in the production of the .tec file. Vorticity is selected by choosing Wz from the menu offered. Also from the same panel, Boundary and Mesh should be de-selected.
To plot the flow inside the elements, the Contour Attributes menu item should be selected. Click on Zone Number and Select All elements. Then click on Contour Plot Type and select Flood. Also, to get varying colour gradients in the flow, select Contour Levels. Set New Levels in the region of -12 and 12.
This done, select Redraw from the upper panel and the flow should be plotted around the aerofoil. Many other functions, such as zooming etc. are available to the user and are self-explanatory. Images can be also converted to normal picture files. This is done using the Export menu item.
4.2 Making and playing movies
As mentioned before, a movie is a number of pictures played in sequence. To create a .rm (movie) file, the user must play the Tecplot macro NACA0012.mcr seen on the right. This is done with the Macro menu item. This takes .plt files in sequence and adds their details to the .rm file. The user must first edit the macro to contain the correct name of the run and to adjust the value nplots to (1+ the number of .plt files that are to be incorporated into the movie). In order to ensure that all movie frames are similar in size, the user should load the first .plt file and adjust the image to the size, range and content they wish to see in the movie. They should then save these characteristics as the .sty (style) file by using the Copy Style to File menu item. When the macro loads its images, it will automatically put them into the required style before adding them to the movie file.

Once the macro has completed its task, the user can view the resultant movie with the framer application by typing framer NACA0012.rm. The movie will then run through and to restart it once it has completed, the “l” key should be used to tell the program to loop through the images. Full instructions for framer are displayed on the screen once it has been started.

4.3 Recording videos of movies
This is a multistage process in which images have to be reformatted and then compiled into a movie before being recorded onto a video cassette.

An application called snapshot is used to “capture” individual images from the screen when they are displayed by framer. snapshot produces a window that can be dragged onto the area of the picture to be captured. By right clicking on the snapshot control panel, the user has the option to save the contents of the capture window as a .rgb file. Pressing the + key on the framer image loads the next image and each image should be recorded as a separate file.

To create a movie, the Silicon Graphics O2 machine (firecrest) should be used in conjunction with a video recorder (available from Mr. J. Wye).

To create a movie file type:

makemovie -o nameofmovie.mov 1.rgb 1.rgb 1.rgb 2.rgb 2.rgb 2.rgb 3.rgb 3.rgb 3.rgb etc.

The reason for each file being included three times is to ensure that each image is displayed by the movie for a longer time than normal, since otherwise the movie would run too quickly.

Next, the application moviemaker should be run. In the initial dialogue box, select Custom and input the following 3 values in descending order in the columns: 768, 576, 5. Next select the Import Movie menu item and open the nameofmovie.mov movie file. Titles and effects can be created in this useful movie editor. Once the user is satisfied with the movie, they should select the menu item Export and save the sequence as a Quicktime Movie.
This movie can then be viewed on screen using the *movieplayer* application. To record this movie onto a tape the *mediarecorder* application also should be started. Under the *Options* menu, *Video Panel* should be selected and the *Composite Video Input* and *Video Output* bars should be changed to the *Square PAL* settings. Now the *Utilities* menu should be chosen and *Live Video Output* selected. A window will appear and should be placed over the *movieplayer* window.

Another window in *mediarecorder* monitors what is visible in the recording window. Once the user has checked the *movieplayer* run once more to be ensure that the desired output is seen correctly in this monitor window, the record button on the video, whose channel must be set to *AU*, should be pressed and the movie restarted.

The video should now record successfully, and can be played back using the *Live Video Input* menu item.
BIBLIOGRAPHY

Books:
Abbott, Ira H. & Von Doenhoff, Albert E. - Theory of Wing Sections

Projects:
Gouthenoire, Cedric - Computational Study of Mesh and Expansion Repartition Blockage Effects in the Flow Around a Circular Cylinder