Design and Computing 1

Introduction to Scientific Computing

Daniel Poole
Department of Aerospace Engineering
University of Bristol
`d.j.poole@bristol.ac.uk`

2017

LECTURE 5

Intro to Scientific Computing

- Have used programs to perform simple calculations
- Conditionals can be used to branch off during a program
- Loops are used to perform calculations multiple times
- Can split a problem down and program using divide-and conquer using functions

TODAY

- Reading and writing from files
- Writing to specific formats
- Using scientific plotting software - Tecplot 360

File I/O

File I/O

Up until now, the only way we have given our programs variable inputs (i.e. not hard-coded into the program) is by asking the user for information from the screen. Likewise, the only way we have displayed output is also via the screen. However, consider the case of, say, a computational fluid dynamics (CFD) solver. A code as complicated as this has many inputs, including a large lattice mesh file, and has a very complicated output–a flowfield.

In cases where either there are a large number of inputs, or the input is say a geometry, asking the user to input this from the screen is obviously going to cause problems. Likewise, if the output is some complicated visual structure, then obviously the basic terminal screen we use to run our codes does not provide sufficient fidelity to visualise such structures. In these cases (and almost all cases) we need to read and write from and to files.

For clarification, a file can be any document in any format, however, to make it considerably easier, code developers normally talk about simple ASCII text files, like the files you have been writing your codes in.

File Handling

In all programming languages, whether we are reading or writing, we need to tell the program to open the file, do something to the file, and then close the file.

OPEN

```
FILE *<filepointer>;
<filepointer> = fopen("<filename>","<access>");
```

DO STUFF

CLOSE

```
fclose(<filepointer>);
```

File Handling

For example:

```
#include <stdio.h>
int main () {
  FILE *fid;
  fid = fopen("myfile.txt","w");
  fclose(fid);
  return(0);
}
```

`fid` is a special pointer variable (it has type `FILE *`). Every time we want to access the file, we must include this variable such that the program knows to which file we are referring. This also means we can have multiple files open at any one time (obviously having different variable names).

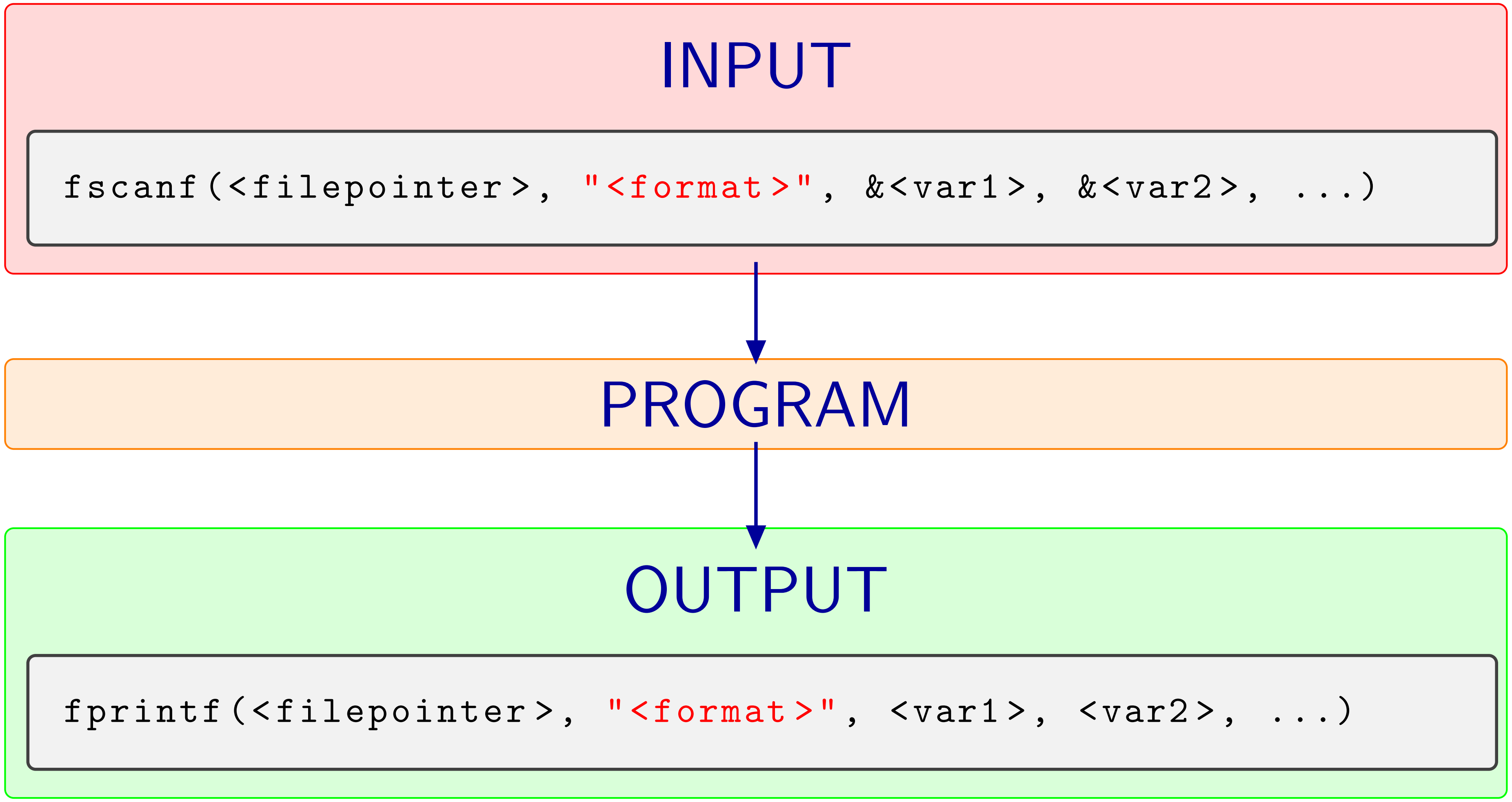
File Handling

`"<access>"` in `fopen` specifies what we are planning on doing with the file. The following are options:

`"r"`	open for reading (file must exist)
`"w"`	open for writing (**file will be overwritten if it exists**)
`"a"`	open for appending to end of file (file need not exist)
`"r+"`	open for reading and writing, start at beginning
`"w+"`	open for reading and writing (overwrite file)
`"a+"`	open for reading and writing (append if file exists)

File I/O

To read and write from and to the file we have opened, we use commands similar to what we haven previously seen, except now they are:



```
fscanf(<filepointer>, "<format>", &<var1>, &<var2>, ...)
```

```
fprintf(<filepointer>, "<format>", <var1>, <var2>, ...)
```

File I/O

So we have now introduced `fscanf` and `fprintf`, which differ from the normal read and print from and to screen commands (`scanf` and `printf`) by including the file pointer variable as the first argument. These can be used as:

```
#include <stdio.h>
int main () {
  FILE *fid1, *fid2;
  int a;
  double b;
  fid1 = fopen("myfile1.txt","r");
  fid2 = fopen("myfile2.txt","w");
  fscanf(fid1, "%i, %lf", &a, &b);
  fprintf(fid2, "%i, %lf", a, b);
  fclose(fid1);
  fclose(fid2);
  return(0);
}
```

Safe File Access

We must also be careful when accessing files, particularly if reading, as the file may not exist or may be locked by another program. In this case we should always check that the file is available for accessing (`fopen` returns `NULL` if an error occurs):

```
/* Open file */
fid = fopen("myfile1.txt","r");
/* Continue if file open ok */
if(fid != NULL){
    fscanf(fid1, "%i, %lf", &a, &b);
    fclose(fid1);
}
else {
    /* Print error if not ok */
    printf("Could not open file!\n");
}
```

Handling End of File

Often we may not know the length of the given file in advance. Our program may crash if we try to read past the end of a file (this is an obvious error to debug if we do get it), hence we should also include a check to stop this.

`feof` returns 0 if the file is not at the end

```
/* Open file */
fid = fopen("myfile1.txt","r");
/* Continue if file open ok */
if(fid != NULL){

   /* Read to end of file */
   while(feof(fid) == 0) {
     fscanf(fid1, "%i, %lf", &a, &b);
   }

   fclose(fid1);
}
```

File I/O Summary

We have now outlined the major aspects of accessing files for reading and writing. The main commands you will use during the labs for this are:

`fopen`	open file
`fclose`	close file
`fscanf`	read from file
`fprintf`	write to file
`feof`	check file end not reached

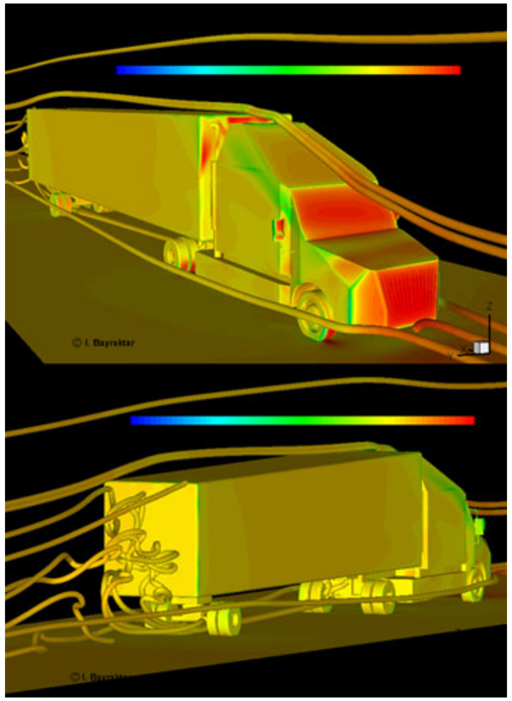
Tecplot 360

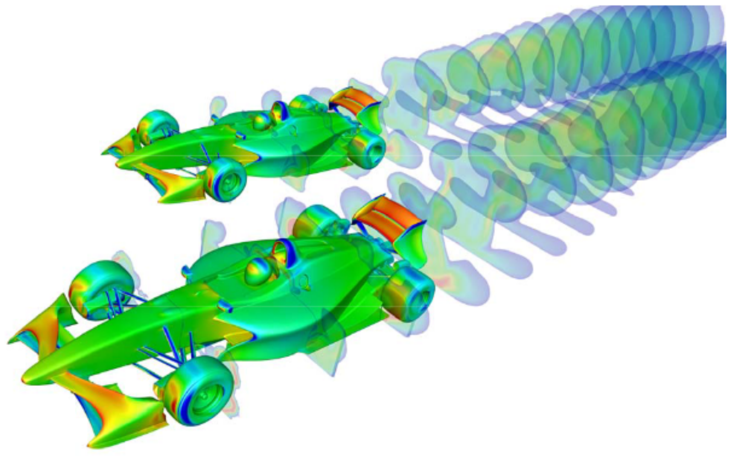
Writing Formatted Data

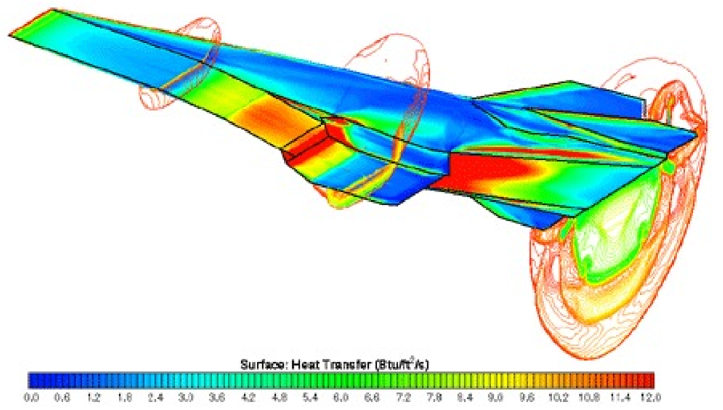
When writing data to files, we may often be doing this for use by another program, most commonly for visualisation. Almost all visualisation software requires data to be input in a given specific format. It is therefore important that you get used to writing data in specific formats for visualisation.

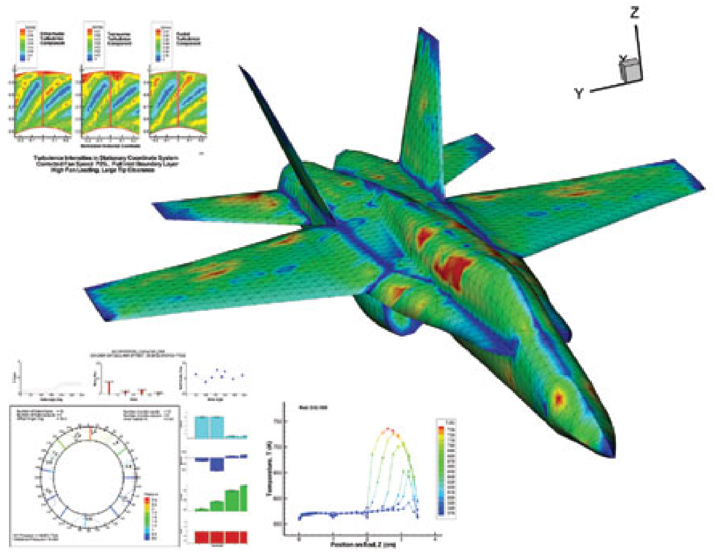
A common visualisation suite used in scientific computing is called Tecplot. Tecplot is a powerful visual data analysis tool used commonly for producing two-dimensional plots, two- and three-dimensional visualization of flow and stress fields and complicated animations. In the next few slides we will introduce the basic functions of Tecplot, and the common ones that you may use. We will only scratch the surface of what Tecplot can actually do, however, this should give you a flavour of the sort of tools available to the engineer to visualise solutions.

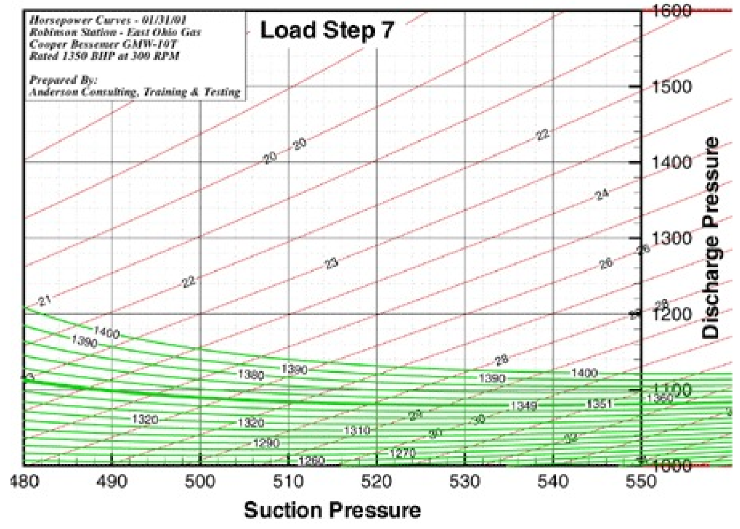
Tecplot 360


© I. Bayraktar




Surface: Heat Transfer (Btu/ft^2/s)
0.0 0.6 1.2 1.8 2.4 3.0 3.6 4.2 4.8 5.4 6.0 6.6 7.2 7.8 8.4 9.0 9.6 10.2 10.8 11.4 12.0


Z
Y
X


Horsepower Curves - 01/31/01
Robinson Station - East Ohio Gas
Cooper Bessemer GMW-10T
Rated 1350 BHP at 300 RPM
Prepared By:
Anderson Consulting, Training & Testing
Load Step 7
Discharge Pressure
Suction Pressure
480 490 500 510 520 530 540 550
1000 1100 1200 1300 1400 1500 1600

Tecplot Format

Tecplot can visualise 2D plots, 2D and 3D flowfield solutions, and many more things. The exact details of the 2D and 3D flowfield solutions we will not go into large detail on as this can get vary complicated. For a complete breakdown of what Tecplot can do, we can refer to the user manuals:

`http://www.tecplot.com/support/documentation/`

In general, though, a Tecplot input file has a `.plt` extension to denote it is for Tecplot (Tecplot can also read in a large array of other data formats, such as CAD, FLUENT, NASTRAN, etc. etc.). The typical layout of the input file is:

```
VARIABLES = "<var1>" "<var2>" ... "<varN>"
ZONE <details>
<v11> <v12> ... <v1N>
...
<vM1> <vM2> ... <vMN>
ZONE <details>
...
```

Tecplot Format

The header contains the variable names. These will form the axis of any plots used, as well as being used as a basis for things such as contour plots.

Tecplot data is then divided into zones, where each zone can be plotted separately on the same graph.

- `T="<zonename>"`: We can name each zone (if this is absent then tecplot performs its default naming)

- `SOLUTIONTIME=<framenumber>` If doing videos then we must create separate zones for each different frame of the video where each zone is assigned a frame number

DEMO: using Tecplot

Tecplot Format - 2D plots

Two-dimensional line plots are the simplest data format for Tecplot. For example, consider three points in the $x - y$ axis. This can be written as:

```
VARIABLES = "x" "y"
0.0  0.0
1.0  2.0
2.0  2.5
```

Once this has been created, we load into Tecplot using:

- `File / Load data files`
- `Tecplot Data Loader`
- `<Select file> / Ok`
- `<Select plot type> / Ok`

Tecplot Format - 2D plots

Two-dimensional line plots are the simplest data format for Tecplot. For example, consider three points in the $x - y$ axis. This can be written as:

```
VARIABLES = "x" "y"
0.0 0.0
1.0 2.0
2.0 2.5
```

Tecplot has a massive customisability, so we can alter our figure to make it look suitable for what we want e.g.

- `Plot / Axis`
- `Plot / Line Legend`
- `Plot / Mapping Style`
- `Mapping Layers`

Tecplot Format - 2D plots

A useful feature of Tecplot is the abaility to save your layout. This is especially useful as often it can take a long time to get a figure layout that looks good for what you want. The layout is saved as a `.lay` file:

- `File / Save Layout`

And can be used to load other data that is **formatted in the same way as the original data file** used to create the layout.

The layout file is simply a text file so can be edited manually if needed.

Tecplot Format - 2D plots

Two-dimensional line plots are the simplest data format for Tecplot. For example, consider three points in the $x - y$ axis. This can be written as:

```
VARIABLES = "x" "y"
0.0 0.0
1.0 2.0
2.0 2.5
```

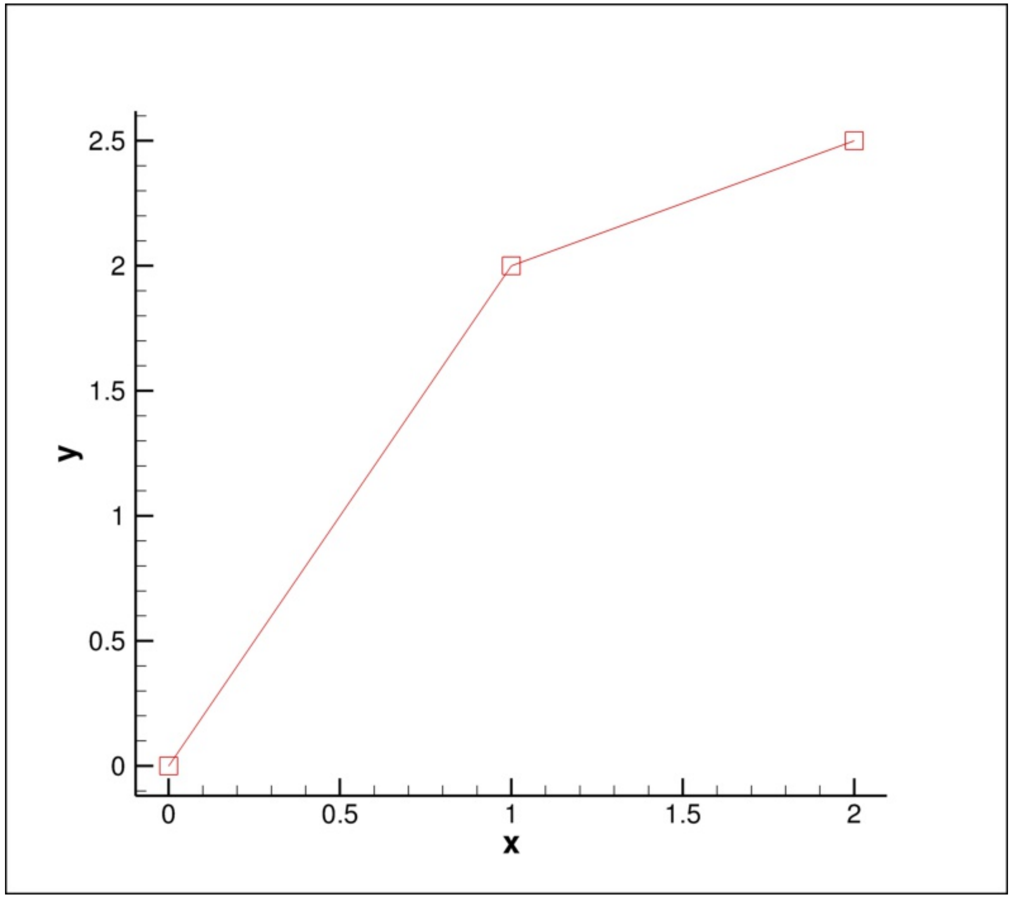
For efficiency, Tecplot's screen output is an unrendered approximation to the figure. We must therefore output the figure to a file, which we can use later in a document. Again, Tecplot has a myriad of output formats, though the most common you will use are .eps, .jpg, .png.

NOTE: if using .jpg or .png, it is recommended to use antialiasing of about level 5. It will increase the rendering time of your image, but allow you to get a much sharper image for fewer pixels.

Tecplot Format - 2D plots

Two-dimensional line plots are the simplest data format for Tecplot. For example, consider three points in the $x - y$ axis. This can be written as:

```
VARIABLES = "x" "y"
0.0 0.0
1.0 2.0
2.0 2.5
```



Tecplot Format - 2D plots

We can have multiple zones, or even multiple variables (within Tecplot we then decide which zones to display and which variables to plot):

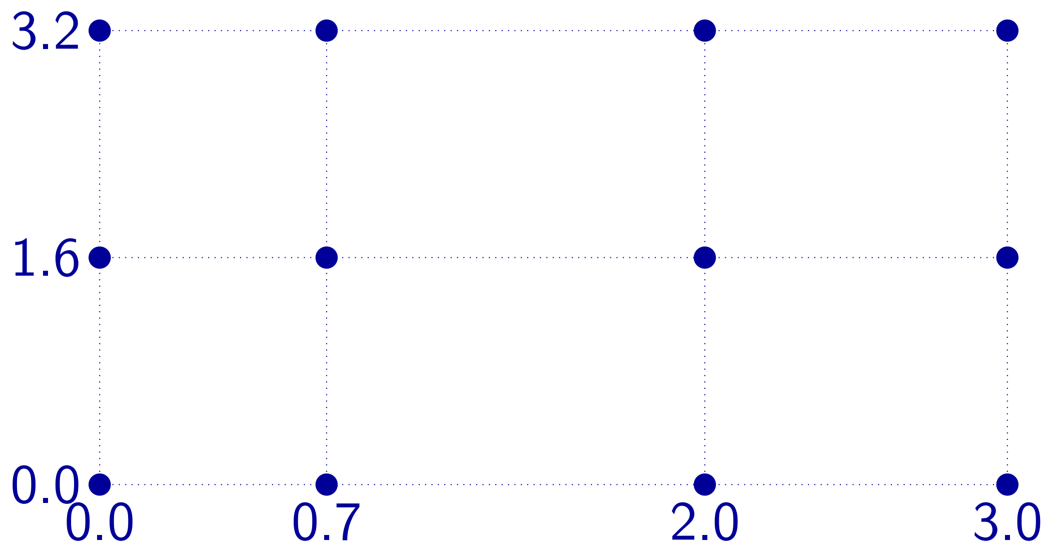
```
VARIABLES = "x" "y"
ZONE T="Zone 1"
0.0  0.0
1.0  2.0
2.0  2.5
ZONE T="Zone 2"
1.0  0.0
2.0  1.5
3.0  2.5
```

```
VARIABLES = "x" "y" "z"
ZONE T="Zone 1"
0.0  0.0  0.0
1.0  2.0  2.0
2.0  2.5  1.5
ZONE T="Zone 2"
1.0  0.0  2.0
2.0  1.5  1.6
3.0  2.5  2.9
```

Tecplot Format - For Reference

Structured grids:

When importing structured grids of data (this is common in simulation), the zone details also need extra information on how the structured grid is arranged. Structured grids are made up of `ni` columns and `nj` rows of points. For example if `ni=4` and `nj=3`



Would require the information of:

```
ZONE  I=4  J=3  F=POINT
```

Tecplot Format - For Reference

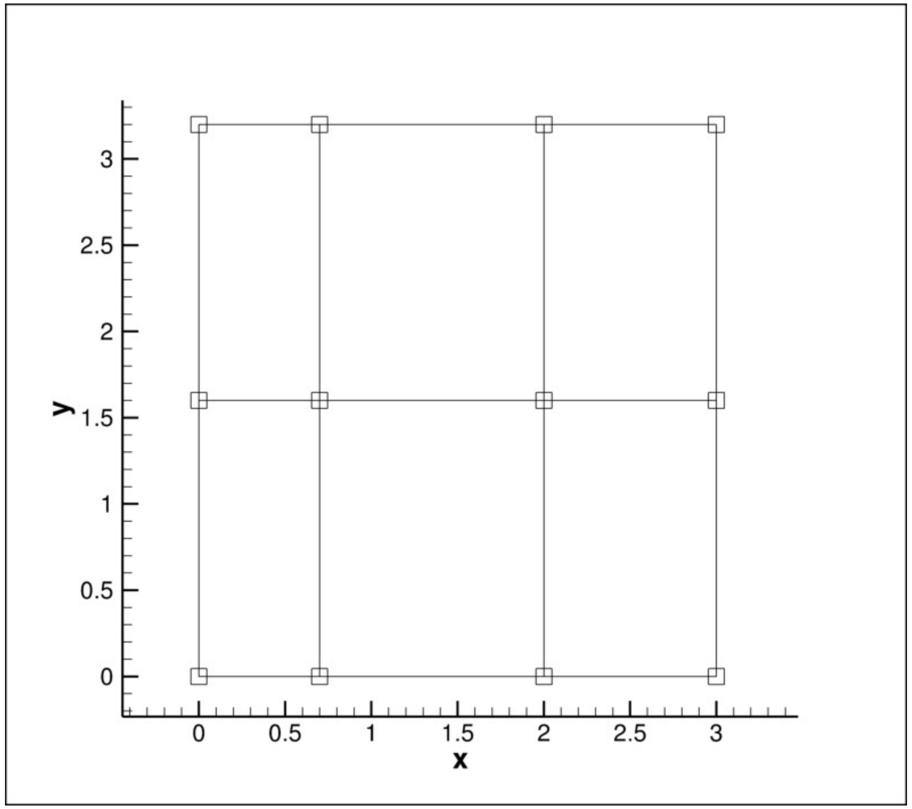
Where the `POINT` keyword says that the data is made up of points. If we were working in 3D then we would also need `K=<nk>`. The data structure must then be given as blocks of row data, where each block is a column. The file would then be:

```
VARIABLES = "<v1>" "<v2>"
ZONE I=<ni> J=<nj> F=POINT
<v1(1,1)>        <v2(1,1)>
 ...
<v1(ni,1)>       <v2(ni,1)>
<v1(1,2)>        <v2(1,2)>
 ...
<v1(ni,2)>       <v2(ni,2)>
 ...
 ...
<v1(ni,nj-1)>    <v2(ni,nj-1)>
<v1(1,nj)>       <v2(1,nj)>
 ...
<v1(ni,nj)>      <v2(ni,nj)>
```

Tecplot Format - For Reference

Where the `POINT` keyword says that the data is made up of points. If we were working in 3D then we would also need `K=<nk>`. The data structure must then be given as blocks of row data, where each block is a column. The file would then be:

```
VARIABLES = "x" "y"
ZONE I=4 J=3 F=POINT
0.0  0.0
0.7  0.0
2.0  0.0
3.0  0.0
0.0  1.6
0.7  1.6
2.0  1.6
3.0  1.6
0.0  3.2
0.7  3.2
2.0  3.2
3.0  3.2
```



Summary

- Have introduced the commands needed to assess and read and write files
- Have introduced the scientific plotting software Tecplot
- Have used basic Tecplot functionality

LAB

- Reading from and writing to files
- Plotting data in Tecplot