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 9

Intro to Scientific Computing

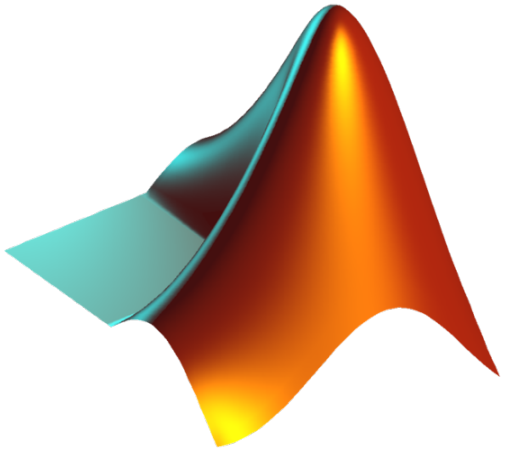
- Have learnt the basics of programming including variables, loops, conditionals, files and arrays.
- Have learnt programming through the basic C language
- Introduced MATLAB and how to manipulate matrices and plot data

TODAY

- Further programming in MATLAB
  - Control flow $\Rightarrow$ loops and conditionals
  - Functions

MATLAB

In the last lecture we introduced the MATLAB tool, including how to manipulate matrices, solve linear systems and plot data. As outlined in the lecture, MATLAB can be entirely run from the command window, however, like a standard programming language can also be coded in a script (and this is preferred).
During the course, we have considered basic programming concepts such as for and while loops, if statements and functions. While we have learnt the concepts, we have only done so through the prism of the C language. All of these concepts, however, can also be coded in MATLAB, and this is all outlined in this lecture.



Remember, MATLAB is more of a development environment/package, and is not suitable for hardcore numerical computing or fast computing times. Before coding in MATLAB you should always ask whether MATLAB really is a suitable tool to use because of the functionality it has, or that you are using it because it appears simpler to use than coding a C (, FORTRAN, Java, C++, etc. etc.) program.

Control Flow - IF

In MATLAB, we can use conditional statements (if, else if, else) to branch off during the executation of our code. The syntax is similar to C, except without all of the brackets:

```
if <expression>
  <statements>
elseif <expression>
  <statements>
else
  <statements>
end
```

- There are no `()` to define the expression
- There are no $\{\}$ to define start and end of statements
- `end` is used to define the end of a block of code

Control Flow - IF

As an example, consider the maximum/minimum aileron angle problem we first visited in Lecture 3. The problem is that the aileron angle should not exceed some maximum and minimum limits:

```
if aileronIN > 30.0
  aileronOUT = 30.0;
elseif aileronIN < -30.0
  aileronOUT = -30.0;
else
  aileronOUT = aileronIN;
end
```

Control Flow - IF

Like in C, the full `if-elseif-else` statement can be simplifed down to if, `if-else` or `if-elseif` as appropriate.

Common boolean operators are similar to those found in C:

Expression	True iff
`a < b`	`a` is strictly less than `b`
`a > b`	`a` is strictly greater than `b`
`a <= b`	`a` is less than or equal to `b`
`a >= b`	`a` is greater than or equal to `b`
`a == b`	`a` is equal to `b`
`a ~= b`	`a` is not equal to `b`
`a & b`	both `a` and `b` are true
`a | b`	either `a` or `b` is true

Control Flow - FOR and WHILE

Again, like in C, we can loop over blocks of code. The `for` loop can loop over a block of text a known number of times, and the `while` loop will go until a Boolean test is satisfied:

```
for <var1>=<expression>
  <statements>
end
```

```
while <expression>
  <statements>
end
```

`break` can be used to exit loops.

Control Flow - FOR and WHILE

Again, like in C, we can loop over blocks of code. The `for` loop can loop over a block of text a known number of times, and the `while` loop will go until a Boolean test is satisfied:

```
for i=1:N
  x(i)=0;
  for j=0: 0.1: 1
    x(i)=x(i)+j;
  end
end
```

```
while x<5
  x=x+0.1;
end
```

`break` can be used to exit loops.

Control Flow - FOR and WHILE

Loops can also be used to increase the size of vectors (or in general, arrays). For example, you may wish to perform a calculation using a different input and save both of these to vectors for plotting later.

E.g. the following code increases the entries in the vectors x and y by 1 each time through the loop

```
i=0;
imax=10;
while i<=imax
  i=i+1;
  x(i)=0.1*(i-1);
  y(i)=x(i)^2;
end
plot(x,y)
```

```
i=1   x = 0.0
i=2   x = 0.0 0.1
i=3   x = 0.0 0.1 0.2
```

Control Flow - Vectorization

We can take advantage of the knowledge that MATLAB is optimized for coding with vectors, and often avoid unnecessary looping (and get massive performance increases).

An example is to plot $\sin(x), 0 \leq x \leq \pi$, which requires lots of values of x to get a smooth curve:

time=0.54s

```
n=0;
for i=0:0.000001:pi
  n=n+1;
  x(n)=i;
  y(n)=sin(x(n));
end
plot(x,y)
```

time=0.048s

```
x=[0:0.000001:pi];
y=sin(x);
plot(x,y)
```

More details can be found at:

http://uk.mathworks.com/help/matlab/matlab_prog/vectorization.html

MATLAB functions

In MATLAB, we can also write functions, like in C. This allows us to split up our problem in a divide and conquer approach. However, to create a function we must write it in a separate file to our main script. The name of the file must be the same name as the function.

```
function [<out1>, <out2>, ... ] = <funcname>(<in1>, <in2>, ... )
%<funcname> Summary of this function goes here
%   Detailed explanation goes here
<statements>
end
```

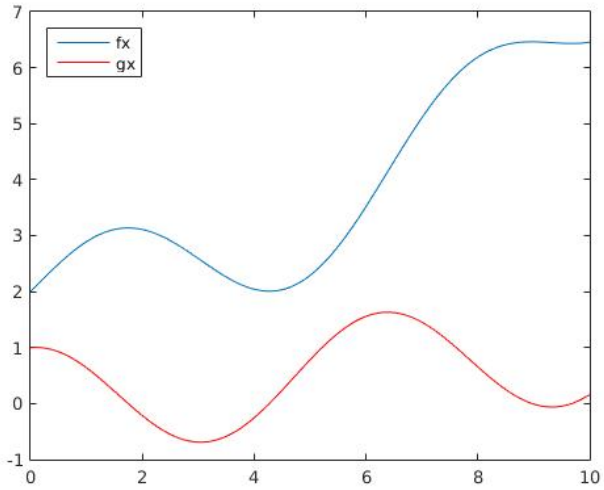
For example, calculate the lift of a wing using a function contained in the file `lift.m`:

```
function [Lforce] = lift(rho,vel,area,CL)
%LIFT Calculate the lift of a wing
Lforce=0.5*rho*vel*vel*area*Cl;
end
```

The comments in the header are then accessible in the `help` command.

A further example is estimating the gradient, of some function:

```
clear all;
close all;
x=[0: 0.01: 1];
fx=myfunc(x);
gx=mygrad(x);
plot(x,fx)
hold on
plot(x,gx,'r')
```



myfunc.m

```
function [fx] = myfunc(x)
%MYFUNC Return some function of x
fx = 2.0 + (0.05*.x*.x) + sin(x);
end
```

mygrad.m

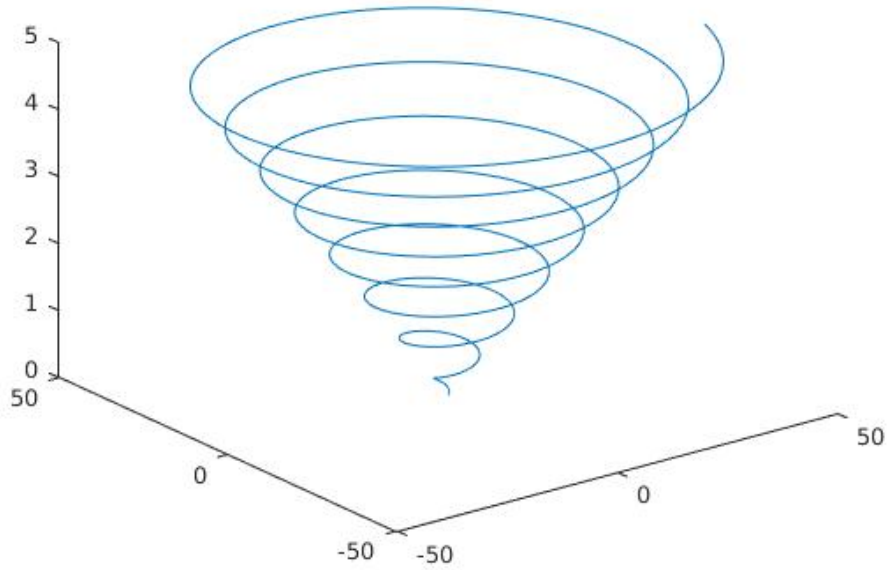
```
function [gx] = mygrad(x)
%MYGRAD Return gradient of myfunc at x
dx=0.001;
gx=(myfunc(x+dx)-myfunc(x-dx))/(2*dx);
end
```

Extra Info - 3D Plotting

Extra Info: 3D line plots

Matlab can also plot line graphs in 3D as well as 2D. The syntax of the `plot3` command is similar to the standard `plot` command. For example, plotting a spiral:

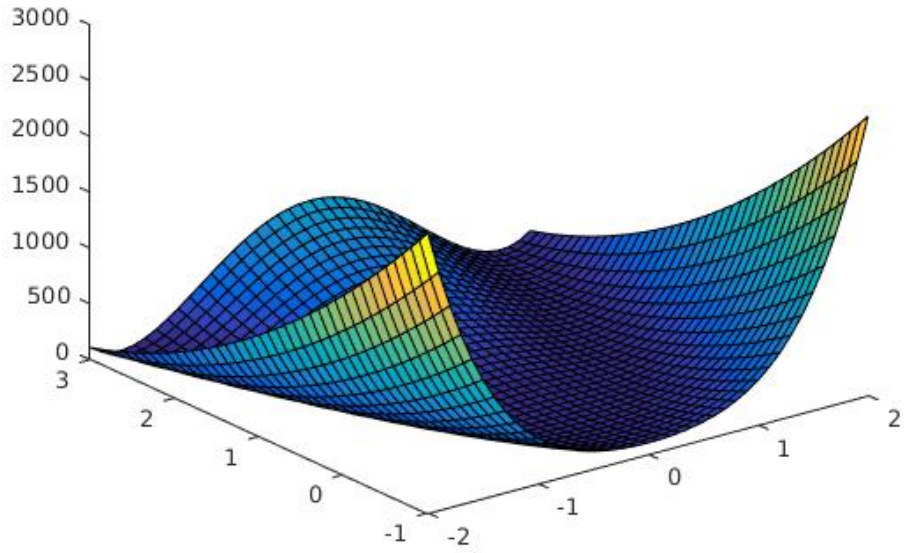
```
t=[0:0.01:50];
x=cos(t).*t;
y=sin(t).*t;
z=0.1*t;
plot3(x,y,z)
```



Extra Info: 3D surface plots

If we want to visualise some function that has two variables (the function can be plotting on the vertical axis). We must firstly create a structured grid on which to plot the surface, and MATLAB has an inbuilt `meshgrid` command for this:

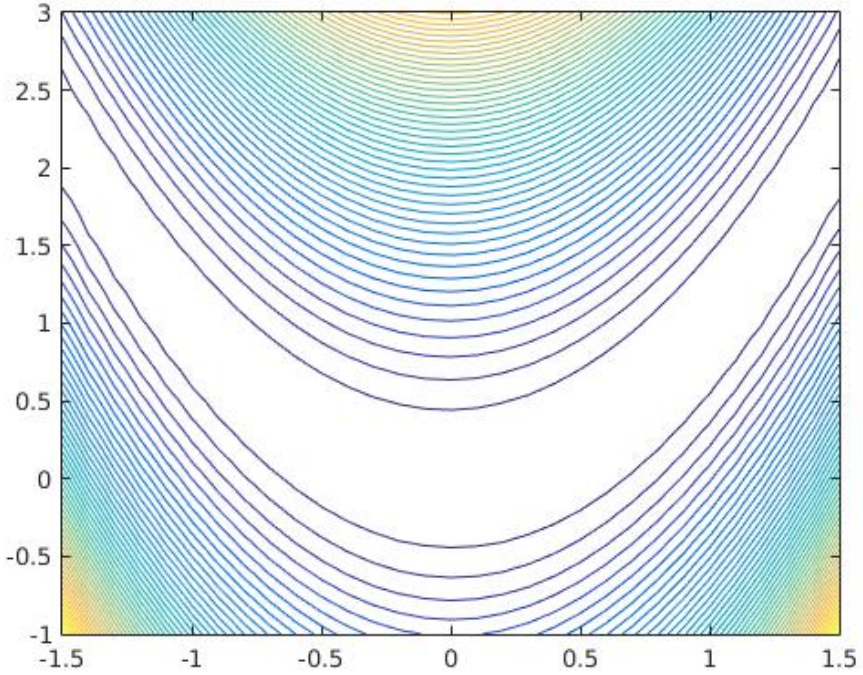
```
x=-2:0.1:2;
y=-1:0.1:3;
[X,Y]=meshgrid(x,y);
Z =(1-X).^2+100*(Y-X.^2).^2;
surf(X,Y,Z)
```



Extra Info: 3D contour plots

Alternatively, we can do a contour plot. The `contour` function takes the same argument as the `surf` function:

```
x=-2:0.1:2;
y=-1:0.1:3;
[X,Y]=meshgrid(x,y);
Z =(1-X).^2+100*(Y-X.^2).^2;
contour(X,Y,Z,50)
```



Summary

- Have introduced using MATLAB
- Have considered how to manipulate matrices in MATLAB and plot data

LAB

- Curve fitting data