MATLAB for Engineers (Beginner's Edition)

http://uiuc-cse.github.io/matlab-sp14/

Section 1, Session 1

Introduction (MATLAB, programming) • 15min

Variables & equations • 15min

Functions: Area of a circle & volume of a sphere • 5min

Functions, plotting: Fahrenheit/Celsius • 10min

Elementwise & matrix operators: Truss forces • 15min

Vectorization, control flow: Falling ballistic object • 15min

Plotting, vectorization: Dam rate of flow • 10min

Scripts, file operations: Fahrenheit/Celsius • 25min

Introduction (MATLAB, programming) • Computer Science • 10min

- How many of you have programmed at all before? Which languages?

In this section of MATLAB, we will be starting at a basic level: what is programming, what are variables, what are functions, etc. This is a traditional way to start programming:

```
disp('Hello world!')
```

It simply echoes (or prints) the contents to the screen. The material outside the quotes is a different color, indicating that it is a command or definition or equation. The quoted material is a *text string*, or a set of characters that is not processed as commands but simply manipulated as a whole.

So a *program* is a list of commands, data, logic, and comments which is used to control a computer.

- *Commands* are things like `disp` above: they simply tell the computer to do something it knows how to do.
- *Data* or *variables* are the raw numbers or text which are manipulated, calculated, or displayed.
- *Logic* is the control system, which tells the computer when to do what.
- *Comments* are human-readable blocks of text which explain what is going on to future readers of your code—including yourself.

MATLAB lets you either input statements one at a time into the interpreter (where we will start) or save a set of commands as a file, or script, which can be executed repeatedly as necessary.

Although we will see some elements of programming, many of you will not immediately start programming in a grand sense with MATLAB. Mostly you will use it for mathematical definitions and solutions, as well as plotting.

Variables & equations • Computer Science • 15min

A number in a computer is stored in a *binary representation* at a unique location in memory. A *variable* is the term we use to refer to both the contents of this location and the user-assigned name of those contents. For instance,

```
x = 5
```

creates a variable x that will henceforward refer to a memory location containing the binary code 00000101b. (The b means that the number is in binary—digress to binary representation if necessary.) Much of numerical computer programming consists of algebraically manipulating variables in *algorithms*, which are formulæ which describe how to manipulate the variables. (Variable names may be as long as you like, but it is good to make them as short as possible for their description.)

The standard arithmetic functions have defined operators. Try the following statements out and see if you can figure out what each one does.

```
x = 2
y = 3
x + y
```

```
x * -y
x / y
x ^ y
```

```
1:6
1:2:7
1j
```

There is also a logic in computer programming.

```
x = 2
y = 3
x == y
```

```
x ~= y
1||0
0||0
```

```
1||1
xor(1,1)
xor(0,0)
```

```
xor(0,1)
1 && 0
1 && 1
```

You will soon find that mathematically impossible statements are commonly written:

```
x = x + 1
```

which of course does not describe an algebraic statement but a command: make the *new* value of x equal to the *old* value of x plus one.

Much like a graphing calculator, there are also functions built in for many mathematical needs. The function browser helps you find what you want.

```
sin(5.43)
exp(1)
```

- Compute the following values in MATLAB. If you have trouble finding a function, use the function browser.

  $3\cos(5\pi/4)$

  $1+1/2+1/3+1/4+1/5+1/6$ *(use* `:` *)*

  $\sin^2(3\pi/2)$

  $\ln(2^5+1)$

  $7!$ *(use the function browser)*

  $$\begin{bmatrix} 5i & -4 \\ 2 & 7-i \end{bmatrix}\begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Note also that `ans` is a special variable, the result of the last calculation. It changes on every statement execution, though, so be careful using it.

Example: Area of a circle & volume of a sphere (functions) • Mathematics • 5min

Given the radius r, what is the area A of a circle? The classic formula is written

$$A(r) = \pi r^2$$

- To make this reproducible, let us create a function to store it. Open a new file and enter the following:

```
function [A] = areaOfCircle(r)
A = pi * r^2;
```

- Save this file as areaOfCircle.m. Test it on a few values. Now try it on `1:1:5`. What do you need to fix for this to work properly?
- Now compose a new function, `volOfSphere`, and make it work.

Example: Fahrenheit/Celsius (functions) • Mathematics • 25min

You are familiar with the conversion formula from Fahrenheit to Celsius:

$$T_F(T_C) = T_C \frac{180}{100} + 32$$

- Write a function which performs this conversion and save it in an appropriate file.

```
function Tf = TempC2F(Tc)
... your code here ...
```

- Now we wish to plot the conversions over a range -50℃–150℃.

```
tc = linspace(-50,150,1001)
tf = TempC2F(tc)
plot(tc,tf)
```

Introduction (MATLAB, programming) • Computer Science • 10min

Although we've seen a few features—operators, functions, even some basic plotting—the real power of MATLAB lies in its namesake: *MATrix LABoratory*. MATLAB is very good at letting you work with groups of numbers as either *matrices* (linear algebra) or *data arrays* (spreadsheet columns). (We saw this implicitly with the range notation, `2:2:16`.)

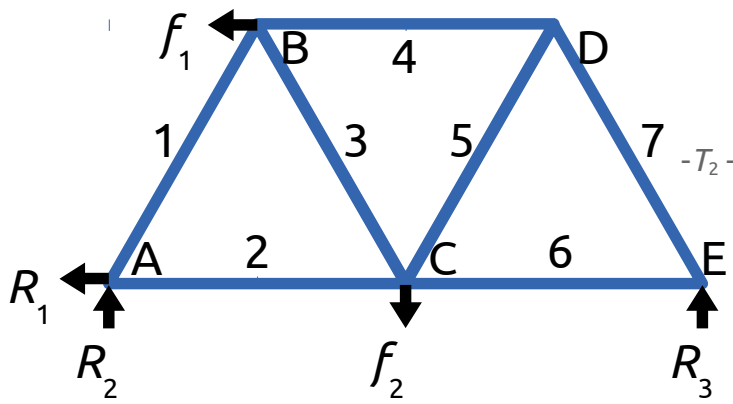
```
A=[1 2 3;4 5 6;7 8 9]   v = [1 3 4 2];       eye(3,3)              A=ones(3)
B=[7 8; 9 10; 11 12]    v'                   eye(3)                C=rand(3)
A*B                     rand(4)*v            ones(3)*ones(3,1)     A*C
A*A                     rand(4)*v'                                 A.*C
A.*A                    M = [1 2 6; 0 4 3]   (0:0.1:1)*5           A^2
                        M'                                         A.^2
```

- What is the result of each of the following commands?

```
A(2,2) = 100;
v(5) = 1;
B(1:2) = 1;
B(:,2) = 3;
B(4,4) = 4;
B(2,:) = 5;
```

Example: Truss forces (Elementwise & matrix operators) • Mathematics • 5min

A common problem in mechanics is the solution of forces in a truss. This is solved statically by the *method of joints*, in which an equation is written for each node of the truss and resulting set of linear equations is solved.



$$
\begin{aligned}
0.5\ T_1 + T_2 &= R_1 = f_1 \\
0.866\ T_1 &= -R_2 = -0.433\ f_1 - 0.5\ f_1 \\
-0.5\ T_1 + 0.5\ T_3 + T_4 &= -f_1 \\
0.866\ T_1 + 0.866\ T_3 &= 0 \\
-T_2 - 0.5\ T_3 + 0.5\ T_5 + T_6 &= 0 \\
0.866\ T_3 + 0.866\ T_5 &= f_2 \\
-T_4 - 0.5\ T_5 + 0.5\ T_7 &= 0
\end{aligned}
$$

Let us write the matrix form of this equation as $\mathbf{T}x = f$. You could write the solution to this formally as $x = \mathbf{T}^{-1}f$. While attractive formally, it is often far too expensive to calculate and store an inverse matrix in memory for large problems. Matrix inversion is a brute-force solution to a linear algebra problem. MATLAB has a number of clever ways to solve matrices built into it. The most frequent method is *not* to invert the matrix, but instead to use what is called *right division*, written $x = \mathbf{T}\backslash f$.

- Let $f_1 = 1000$ and $f_2 = 2000$. Write the governing equations in matrix form.

$$
\begin{pmatrix}
0.5 & 1 & 0 & 0 & 0 & 0 & 0 \\
0.866 & 0 & 0 & 0 & 0 & 0 & 0 \\
-0.5 & 0 & 0.5 & 1 & 0 & 0 & 0 \\
0.866 & 0 & 0.866 & 0 & 0 & 0 & 0 \\
0 & -1 & -0.5 & 0 & 0.5 & 1 & 0 \\
0 & 0 & 0.866 & 0 & 0.866 & 0 & 0 \\
0 & 0 & 0 & -1 & -0.5 & 0 & 0.5
\end{pmatrix}
x =
\begin{pmatrix}
f_1 \\
-0.433 f_1 - 0.5 f_2 \\
-f_1 \\
0 \\
0 \\
f_2 \\
0
\end{pmatrix}
$$

- You can either define **T** and *f* using the MATLAB *New Variable...* interface or load it from the file `truss-matrix.mat`.

```
load('truss-matrix.mat', 'T')
load('truss-matrix.mat', 'f')
```

- Solve the matrix using inversion:

```
x = inv(T)*f
```

- Solve the matrix in MATLAB using right division:

```
x = T\f;
```

- Compare the speed of these two methods using `tic` and `toc`.

```
tic; x = T\f; toc
```

Example: Falling ballistic object (vectorization, functions) • Physics • 15min

- You are familiar with the equation for an object moving in a gravitational field (a.k.a. falling). The formula for vertical position y as a function of time t may be written
  - $y(t) = y_0 + v\,t + a\,t^2$
- where y_0 is the initial position (typically ground level, $y_0 = 0$); v is the initial y-velocity; and a is the acceleration (in this case due to gravity, a = -9.8 m/s^2). The classic ballistics example would be an object fired into the air such that a bullet or shell has a high initial velocity. Let's update this by using a railgun, with a projectile velocity of 2.52km/s.
- First, calculate the values directly.

```
a=-9.8; %m/s^2
v=2520; %m/s
x0=0;
t=1;
y=a*t^2+v*t+x0;
```

This will rapidly become cumbersome if we require many values.

- So let's *vectorize* the formula, or allow it to work on many values simultaneously.

```
t=linspace(0,5,101);
y=a*t^2+v*t+x0;      % comment on this error or ask for input
y=a*t.^2+v*t+x0;
```

- Plot the vectorized formula: `plot(t,y);`
- Finally, let's abstract the formula out into its own function. This which will allow us to reuse it over and over again as necessary. Open a new blank file and write the function definition: `function y = a_fall(t,v,x0)`
- Test the function. When does a projectile traveling at 2.52km/s strike the ground?
- What is the maximum position (`max`)? Where is it (`[val, index] = max(y)`)?
- Modify the function to not yield y-values less than zero.
- Write another function, `v_fall`, to give the velocity at each position. Use this to find the time at which the highest point of the projectile is attained.

Example: Dam rate of flow (plotting, vectorization) • Civil Eng. • 10min

One equation for the volume rate of flow of water Q over the spillway of a dam is the formula

$$Q = C\sqrt{2g}\,B\left(H + \frac{v^2}{2g}\right)^{3/2}$$

where C is the discharge coefficient; B is the spillway width; and H is the depth of water passing over the spillway.

- Open the file `computedamrateofflow.m`. This has skeleton code for a function you will define.
- Define a discharge coefficient of 1.946 and the acceleration due to gravity in SI units.
- Calculate the value of Q. This should be in the form `Q = <something>`.
- Plot your function for B = 1m, v = 0.536m/s (the flow rate of the Mississippi river) against a vector H ranging from 1 to 10 at increments of 0.1.
- What would you change to make this also a function of discharge coefficient C?

Example: Fahrenheit/Celsius (scripting, file operations) • Mathematics • 25min

Recall our previous conversion function `TempC2F`.

- Write a script `templist.m` which generates a table of conversions for the range 0℃–200℃ and then writes that table to a file `templist.txt`.

```
cList = linspace(0,200,101);
fList = TempC2F(cList);

fileID = fopen('templist.txt','w');
fprintf(fileID,'%6s %12s\n','Tc','Tf');
fprintf(fileID,'%6.2f %6.2f\n',cList,fList);
fclose(fileID);
```

- Examine `templist.txt`. Is the output what you expect? Change the output line to the following:

```
tList = [cList; fList];
fprintf(fileID,'%6.2f %6.2f\n',tList);
```

(Note that the row-major order of MATLAB translates into a transposed column-major order in the output.)

- Next, let's read the data in and plot them.

```
tData = importdata('templist.txt')
plot(tData.data(:,1),tData.data(:,2))
```