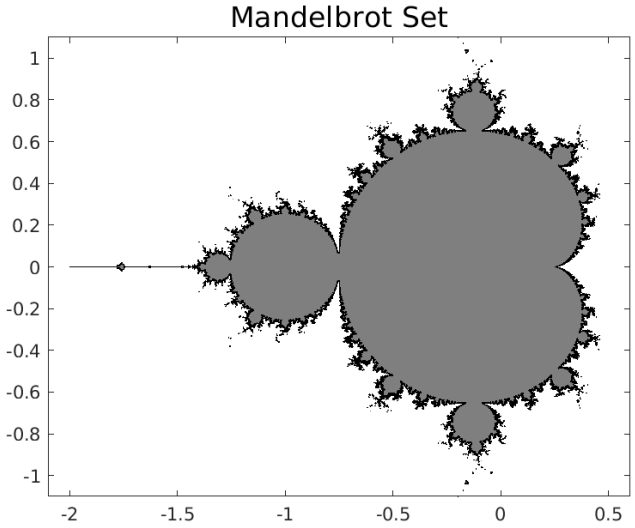
CS319: Scientific Computing (with MATLAB)

Niall Madden

DRAFT: Flow control; Matrices and Vectors

Week 3: **9am and 4pm**, 25 Jan 2023



Source: MATLAB Guide

Lab times

	Mon	Tue	Wed	Thu	Fri
9 – 10			LECTURE		
10 – 11			LAB 1		
11 – 12			LAB 2		
12 – 1					
1 – 2					
2 – 3					
3 – 4					
4 – 5			LECTURE		

- Attend either (or both) Labs 1 or 2.
- Can anyone attend neither?
- Any requests for another hour?

Send Niall an email!

Bitbucket git repo

You should now have access to the CS319 git repository at https://bitbucket.org/niallmadden/2223-cs319/src. If not, check your email for an invitation, or send me an email.

Today in CS319:

This morning.

1. 1: Output/Input
   - Output
   - format
   - fprintf
   - Input
2. 2: Flow of control – `if`
3. 3: `while` loops
   - Example: Newton's method

Other reading:

- Chapter 1 of The MATLAB Guide: https://doi-org.nuigalway.idm.oclc.org/10.1137/1.9781611974669
- Chapter 3 of Learning MATLAB: https://doi-org.nuigalway.idm.oclc.org/10.1137/1.9780898717662

There are several different ways of getting output from MATLAB:

- When running a command, just omit the semi-colon from the line.
- Use the `disp()` function which outputs a single variable.
- The results of the above two can be controlled using the `format` function. Try

```
disp(pi)
format long
disp(pi)
format shortE
disp(pi)
```

There are other uses of the `format` instruction. Use `>> doc format` to read more.

However, the most useful may be:

```
format compact
```

Not as useful, but fun:

```
format rat
```

To reset:

```
format default
```

Mostly, we will use the `fprintf()` function, which is a little like a `f`-string in Python (and almost identical to `printf()` in C).

This is especially useful, because we can mix text and variable values, can specify how many decimal places, to output to, etc. Also, this is used to write to files.

Syntax:

- First argument is always a string (text that starts and ends with either a single or double quote).
- That string may contain a **conversion character**: `%` followed by a letter, e.g., `%f`
- The `%f` is replaced with the value of the second argument.
- Further conversion characters and arguments are allowed.

Common conversion characters:

- `f` fixed-point representation of a float.
- `e` or `E` exponent notation
- `g` or `G` let MATLAB guess if `f` or `e`
- `c` or `s` single character or string
- `d` or `i` integer is better.

fprintf("%e", 0.0123)

gives 1.23e-02

You can also set the

- field width
- precision.

%w.pf

Examples:

```
fprintf('pi = %f\n', pi);      % pi = 3.141593
fprintf('pi = %7.1f\n', pi);   % pi =      3.1
fprintf('pi = %7.3f\n', pi)    % pi = ··3.142
fprintf('pi = %7.5f\n', pi)
fprintf('pi = %7.7f\n', pi)
fprintf('pi = %7.6e\n', pi)
```

In that previous example, `\n` is an "escape character". It causes a newline to be printed.

Other escape characters:

- `\t`
- `\\`
- `%%`

Since MATLAB is an interactive system, reading input in a script is not very common. But if we must:

```
x = input('Tell me something: ')
```

Assumes x is a float

```
x = input('Tell me something: ', 's')
```

Treats x as a string

Later: fscanf for files.

2: Flow of control – `if`

`if` statements are used to conditionally execute part of your code.

Syntax: `if/else`:

```
if( exprn )
    statements to execute if exprn evaluates is true
else
    statements if exprn evaluates as 0
end
```

- The `else` statement is optional, but good practice.
- The `end` statement is needed.
- Indentation is good practice, but not required.

2: Flow of control – `if`

The argument to `if()` is a **logical expression**.

Example

- Equality: `x == 8` or `m == 'c'` (single = used for assignment).
- Inequality: `y ~= x`
- Less than: `y < 1`
- Less than or equal to: `z <= pi`
- Greater than: `X > 9`
- Greater than or equal to: `q123 >= 1/2`

More complicated examples can be constructed using the operators

- **AND** `&&`
- **OR** `||`.

2: Flow of control – if

Eg01_EvenOdd.m

```
Number = input("Please enter an integrer: ");
if ( mod(Number,2) == 0)
   fprintf("%d is an even number.\n",  Number);
else
   fprintf("%d is an odd number.\n",  Number);
end
```

mod (a, b) is the remainder on dividing a by b.

More complicated examples are possible:

Syntax: `if/elseif/else`:

```
if( exp1 )
    statements to execute if exp1 evaluates is true
elseif ( exp2 )
    statements run if exp1 is "false" but exp2 is "true"
else
    "catch all" statements if both exp1 and exp2 false.
end
```

Can have multiple elseif statements.

2: Flow of control – if

Eg02_Grades.m

```
%% Eg02_Grades.m
%   Date   : Jan 2023
%   What   : Example of using if-elseif-else
NumberGrade = input("Please enter the grade (
    percent): ");
if ( NumberGrade >= 70 )
   LetterGrade = 'A';
elseif ( NumberGrade >= 60 )
   LetterGrade = 'B';
elseif ( NumberGrade >= 50 )
   LetterGrade = 'C';
elseif ( NumberGrade >= 40 )
   LetterGrade = 'D';
else
   LetterGrade = 'E';
end
fprintf("%2d%% corresponds to a %c grade\n", ...
   NumberGrade, LetterGrade);
```

2: Flow of control – if

The other main flow-of-control structure is

`switch / case / otherwise`

It has limited used (I find), since it doesn't involve any relational operators. But it can be helpful if you have set some parameter in your code.

Eg03_Switch.m

```
x = [12, 5, 59, 24];
plottype = 'pie'; % One of 'bar', 'pie', 'pie3'
switch plottype
   case 'bar'
      bar(x)
      title('Bar Graph')
   case {'pie','pie3'}
      pie3(x)
      title('Pie Chart')
   otherwise
      warning('Unexpected plot type. No plot
         created.')
```

3: `while` loops

A `loop` is a programming structure that allows for some piece of code to be repeated.

There are two main types of loop in MATLAB:

- `while`: preform a set of instructions as long as a given logical statement holds true;
- `for`: for each element in a vector, preform a set of instructions.

3: while loops

Syntax: while:

```
while( exp1 )
    statements to execute so long as exp1 evaluates is true
end
```

Eg04_Countdown_while.m

```
c = 10;
while (c>0)
    fprintf("%i... ", c);
    c=c-1;
end
fprintf("Zero!\n");
```

Output: 10... 9... 8... 7... 6... 5... 4... 3... 2... 1... Zero!

One of the most classic problem in scientific computing is solving nonlinear equations: given a function f, find x such that $f(x) = 0$.

And one of the most important methods for solving this is **Newton's Method:** if x_k is a good estimate for x, then

$$x_{k+1} = x_k - f(x_k)/f'(x_k),$$

$$f'(x_k) \simeq \frac{f(x_{k+1}) - f(x_k)}{x_{k+1} - x_k}.$$

Now set $f(x_{k+1}) = 0$ and solve for x_{k+1}

To implement this method we need to know how many iterations to preform. Since we are trying to solve $f(x) = 0$, we can use $f(x_k)$ is a good measure for how good an estimate x_k is. That is, we iteration `while` $|f(x_k)|$ is greater than some chosen value.

We also need to know how to define functions in MATLAB. We'll study that in detail next week, but for now we just need to know that the syntax is:

```
>> fun_name = @(x)(formula for function in x)
```

The point term here is the use of the `@` symbol.

We can plot functions defined in this with using `fplot()`.

. . . . . .

In the following example, we'll use Newton's method to solve

$$f(x) = x^2 - 2$$

for $x > 0$. That is, we are estimating $\sqrt{2}$.

E.g,

```
f = @(x)( x.^2 - 2)
```

Eg05Newton.m

```
f = @(x)x.^2-2;
df = @(x)2*x;
fplot(f, [0,3]);

xk = 1;
k = 0;
fprintf("k=%2d, xk=%f, f(xk)=%8.2e\n", ...
    k, xk, f(xk))
while (abs(f(xk)) > 1.0e-6)
    k=k+1;
    xk = xk - f(xk)/df(xk);
    fprintf("k=%2d, xk=%f, f(xk)=%8.2e\n", ...
        k,xk,f(xk))
end
```

MORE NOTES WILL BE ADDED BEFORE THE 4PM CLASS.

Finished here 10am