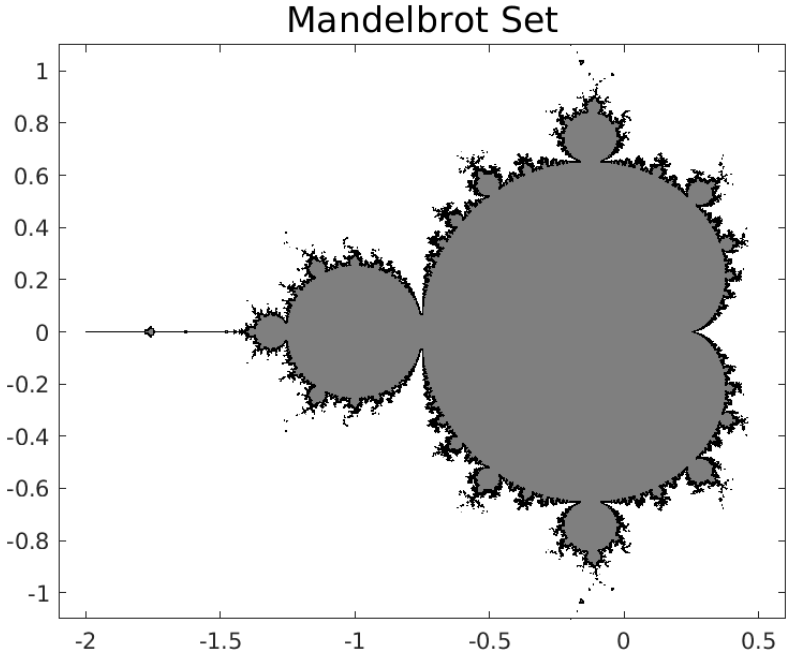
Annotated slides from 4pm class.

CS319: Scientific Computing (with MATLAB)

Niall Madden

# Flow control; Matrices and Vectors

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




Source: MATLAB Guide

# Today, in CS319:

9 am

1. 1: Output/Input
   - Output
   - format
   - fprintf
   - Input
2. 2: Flow of control – `if`
3. 3: `while` loops
   - Example: Newton's method
4. 4: `for` loops
5. 5: Vectors and matrices
   - Common matrices
6. 6. Vectors
   - Accessing elements
   - Vector indexing
   - Useful functions
7. 7. Matrices
   - Vector indexing
   - Operations
   - Special matrices
   - Last example

Pm

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

# 4: `for` loops

A `for` loop is used when we want to

1. Repeat the execution of a block of code a fixed number of times; or
2. Execute a block of code for each element in a (row) vector.

These two applications are actually the same, but we'll treat them separately for now...

**Syntax: `for`: fixed number of iterations**

```
for i=1:N
    statements to executed N times
end
```

Python: for i in range(1, N+1)

In the next example, we will use the `nthprime()` function to display the first 10 prime numbers.

Eg06Primes.m

```
%% File   : Eg05_Countdown.m
%  Date   : Jan 2023 (CS319 Week 03)
%  What   : Use a for loop to display 1st 10 primes
for i=1:10
   fprintf("The %2d-th prime is %2d\n", i, nthprime(i));
end
```

E..g, 3:0 is empty.

Here is a slightly more general version:

**Loop over integers from $a$ to $b$**

```
for i=a:b
    // code to execute inside loop
    // First time, i=a
    // Next, i=a+1
    // ...
    // Last:  i=b
end
```

**Explanation of $a : b$ and of $a : h : b$**

Row vector
- starting at a
- increasing by 1 at each step
- not exceeding b.

E.g, -1.5:3 is the vector [-1.5, -0.5, 0.5, 1.5, 2.5]

# 4: for loops

Here is a slightly more general version:

**Loop over integers from $a$ to $b$**

```
for i=a:b
    // code to execute inside loop
    // First time, i=a
    // Next, i=a+1
    // ...
    // Last:  i=b
end
```

**Explanation of $a : b$ and of $a : h : b$**

- start at a
- increases by h at each step

* does not exceed b.

E.g, 0:2:10 == [0, 2, 4, 6, 8, 10]

# 4: for loops

**Example:** we'll re-do the `while` count-down example using a `for`-loop.

Eg07Countdown.m

```
for i=10:-1:1
   disp(i);
end
disp(' Zero!');
```

10:-1:1 = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

- Start at 10
- Change by −1 each step.
- End at 1.

# 4: `for` loops

1. In the most general use of `for`, the syntax is
   `for Index = ListOfValues`
2. `ListOfValues` is a row `vector`, that is, a $1 \times n$ matrix.
3. At each step through the loop, `Index` takes the next element of the list.
4. If the list is empty, nothing is done.
5. The instructions iterated are between for `for` and `end` lines.
6. Can also use `continue` or `break`, but it is rarely necessary.

# 5: Vectors and matrices

MATLAB stands for "matrix laboratory". The core goal of the original version of MATLAB was to be a "matrix calculator" (https://dl.acm.org/doi/10.1145/3386331

So working with matrices and vectors is simpler than in just about any other language.

In fact, if you assign a single number to a variable, it is stored as a $1 \times 1$ matrix.

Similarly, vectors are just

- $1 \times n$ matrix for a row vectors
- $n \times 1$ matrix for a column vectors

As well as these notes, you should read Chapter 3 of "Learning MATLAB": https://epubs-siam-org.nuigalway.idm.oclc.org/doi/pdf/10.1137/1.9780898717662.ch2

# 5: Vectors and matrices

The simplest way to define a matrix is to list its entries:

- List the entries between square brackets
- Place a space or comma between columns;
- Place a semicolon at the end of rows

```
>> A = [1, 2, 3; -1, -2, 0; 0, 1, 2]
A =
     1     2     3
    -1    -2     0
     0     1     2
```

# 5: Vectors and matrices

```
>> b = [5 5 5]  % commas are optional
b =
     5     5     5
```

```
>> c = [-1; -2; -3]
c =
    -1
    -2
    -3
```

Use `whos` to check the size of these arrays. You can also use the **size** function: `size(A)`

# 5: Vectors and matrices

It is easy to combine matrices and vectors to make larger ones. With the examples above, we could set

```
>> X=[A; b]
```

or

```
>> Y=[A, c]
```

**What happens if you try?**

```
>> X=[A, b] % note comma
```

or

```
>> Y=[A; c] % note semicolon
```

In the case of certain special or common matrices, there are functions to construct them:

- `I=eye(N)` makes the $N \times N$ identity matrix

```
>> I=eye(3)
I =
      1      0      0
      0      1      0
      0      0      1
```

- The zero matrix: `Z = zero(m,n)` sets $Z$ to be an $m \times n$ matrix, all of whose entries are zero.

```
>> Z = zeros(1,4)
Z =
     0      0      0      0
```

- `ones(m,n)` returns the $m \times n$ matrix, all of whose entries are 1.

```
>> ones(3,1)
ans =
     1
     1
     1
```

- Random arrays:
  - `rand(n)` or `rand(m,n)` : uniformly distributed entries in [0,1]
  - `randn(n)` or `randn(m,n)` : entries are normally distributed random numbers (mean of 0).
  - `randi(k,n)` or `randi(k,m,n)`

An n-by-n matrix of random integers between 1 and k.

# Accessing elements

Use round brackets, `(` and `)`, to access a particular element of a vector or array.

In MATLAB, all arrays are indexed from 1.

That means, the first element of any vector, `v`, is `v(1)`.

And the first element of any matrix, `A`, is `A(1,1)`.

There is a special keyword `end` to access the final element of a vector, so that you don't have to know how many elements it has:

```
>> v = [1 2 3 4]
v =
     1     2     3     4
>> v(1)
ans =
     1
>> v(end)
ans =
     4
```

# Vector indexing

A very powerful feature of MATLAB is that you can use integer vectors to access multiple entries at once.

```
>> v = randn(1, 5)
v =
    0.3035   -0.6003    0.4900    0.7394    1.7119
>> v([3,2])
ans =
    0.4900   -0.6003
```

Vector indexing can also be used for setting values:

```
>> x = 1:10
x =
  1     2     3     4     5     6     7     8     9    10
>> x(2:2:10)=0
x =
  1     0     3     0     5     0     7     0     9     0
```

→ set every second entry to zero.

It takes a little getting used to, but one can also use logical indexing. For example, suppose we have the vector $v$ with entries

```
v = [1, -2, 3, -4, 5, -6, -7, 8]
```

and we want to change all the negative entries to 0. Here are two ways to do that

```
for i=1:length(v)
    if (v(i) <0)
        v(i)=0
    end
end
```

length(v) is the number of entries in the vector v.

Or, in a single line:

```
>> v(v<0)=0
v =
     1     0     3     0     5     0     0     8
```

- `find(v)` returns the index of all non-zero entries of v. E.g.,

```
>> v = [1, 0, 0, -2, 0, 0, 3];
>> find(v)
ans =
     1     4     7
```

- `max(v)` and `min(v)`
- `mean(v)` and `median(v)`
- And many others!

Finished here at 5pm, but skipped ahead to do the Mandelbrot example.

MANDEL.m

```
%% MANDEL Mandelbrot set.
% Taken from Listing 1.4 of The MATLAB Guide, 3rd Ed
% https://epubs-siam-org.nuigalway.idm.oclc.org/doi/pdf/10.1137/1.9781611974669.ch1
h = waitbar(0,'Computing...');
x = linspace(-2.1,0.6,2001);
y = linspace(-1.1,1.1,2001);
[X,Y] = meshgrid(x,y);
C = complex(X,Y);
Z_max = 1e6; it_max = 50;
Z = C;
for k = 1:it_max
   Z = Z.^2 + C;
   waitbar(k/it_max)
end
close(h)
contourf(x,y,double(abs(Z)<Z_max))
colormap([1 1 1; 1/2 1/2 1/2]) % Gray inside, white outside.
title('Mandelbrot Set','FontSize',16,'FontWeight','normal')
```