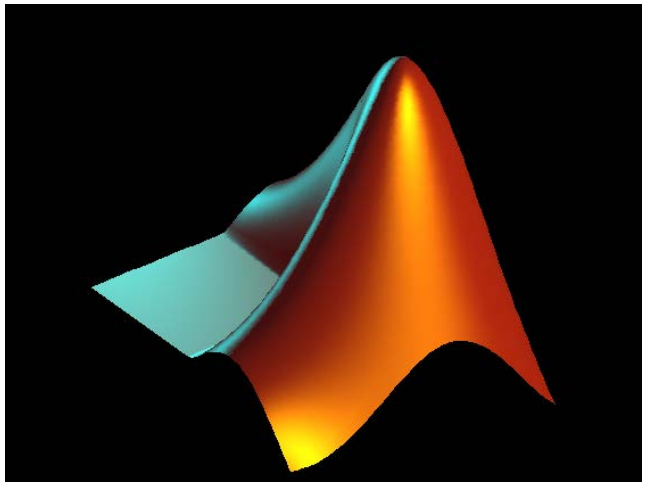
Computational Mathematics with MATLAB
Topic 4



FOR Loops and Loop Plots

Outline

- FOR loop introduction and examples
- randomisation with `randperm` and `randsample`
- single-line FOR loops
- initialising array variables
- formula vectorisation
- loop plotting
- RGB colours
- polygon plot examples

FOR Loops

A basic FOR loop repeats one or more statements a fixed number of times.

The structure of a typical FOR loop is

```
for counter = start_value : step_size : end_value
    one or more statements;
end
```

- the default step size is 1
- the **for** and **end** keywords are automatically highlighted in blue
- statements inside the loop should be suppressed (;)

Example 1 - Permutations

```
% FOR loop example 1

%  number of iterations
n=5;
% vector length
l=8;
for i=1:n
    % random arrangement of integers from 1 to 8
    v=randperm(l);
    disp(v)
end

% v is overwritten in each iteration of the loop
% the results of the 1st n-1 iterations are NOT stored
```

`randperm(n)` returns a row vector including the integers from 1 to *n* in random order

`randperm(n, k)` returns a row vector including *k* unique integers selected randomly from the interval [1, *n*]

Example 2 - Permutations

```
% FOR loop example 2

% number of iterations
n=5;
% length of vectors
l=8;
% storage array for the permutations
m=zeros(n,l);

for i=1:n
    % the rows of m are overwritten 1-by-1 in the loop
    m(i,:)=randperm(l);
end
disp(m)

% all n permutations are now stored in matrix m
```

Example 3 – National Lottery

```
% Example 3 - simulate weekly lottery draws
% UK National Lottery Main Draw: 6 numbers out of [1,59]

n=input('Number of weeks: ');
% create a storage array for the winning numbers
% n weeks -> n rows
% 6 numbers selected each week -> 6 columns
m=zeros(n,6);

for i=1:n
    m(i,:)=randsample(59,6);
end
disp(m)

disp('Total numbers drawn: ');
disp(n*6)
disp('The number of unique winning numbers is: ');
u=length(unique(m));
disp(u)
% This means that (59-u) numbers were not selected even once
```

the randsample function

`randsample(n, k)` returns a sample of *k* unique integers chosen randomly from [1, *n*] in row vector format.

This type of sampling is called **sampling without replacement.**

`randsample(v, k)` returns a vector including *k* elements selected *randomly and without replacement* from the elements of vector v

To change the sampling method to **sampling with replacement** (which means that any element can be selected any number of times), use

```
randsample(n, k, true)
randsample(v, k, true)
```

the randsample function

```
% create a sequence of unique integers
>> l=randperm(50,12)

l =  41   45    7     24    19    8    26    27   10   22    40    20

% take a sample without replacement
>> a=randsample(l, 8)

a =  7    26    24     8    10    45    27    40

% take a sample with replacement
b=randsample(l, 8, true)

b =  41     8    41    22    10     7    10    26


% One of the following commands will give an error message. Why?
>> randsample(l,100)
>> randsample(l,100, true)
```

the unique function

unique(A) returns the unique elements of A in order

```
A =
     5     2     1     2     4     5
     1     4     5     5     5     4

>> unique(A)
ans =           % ascending order
        1
        2
        4
        5

% string example
>> s='fgjsdfgjkshgdsf';
>> unique(s)
ans =
        dfghjks   % alphabetical order
```

Single-Line FOR Loops

It is possible to write a FOR loop in a single line.

Example: generate a vector of the first 8 powers of 2.

Try this in the command window:

```
>> for i=1:8, v(i)=2^i; end, disp(v)
2     4     8    16    32    64   128   256
```

The following variations will not work:

```
>> clear all
>> for i=1:8, v(i)=2^i  end, disp(v)
>> for i=1:8, v(i)=2^i; end  disp(v)
```

Statements within the same line must be separated by commas or semicolons. Statements inside the loop should be ended with ;
Use , inside the loop only when you are testing the loop.

Single-Line FOR Loops

Testing the loop:

```
>> clear all
>> for i=1:8, v(i)=2^i, end, disp(v)

v =
     2
v =
     2     4
v =
     2     4     8
v =
     2     4     8    16
v =
     2     4     8    16    32
v =
     2     4     8    16    32    64
v =
     2     4     8    16    32    64   128
v =
     2     4     8    16    32    64   128   256
     2     4     8    16    32    64   128   256
```

Vector v is replaced with a vector of a different size in each iteration of the loop – this is not an efficient way of using computing resources!

Initialising Variables

Initialising a variable means pre-allocating the correct amount of memory for the variable.

In our current example, we can use zeros(1,8) or ones(1,8) or any other 1 x 8 built-ins to initialise vector v.

Each iteration of the loop will now change 1 element only. Try the following:

```
>> v=zeros(1,8); for i=1:8, v(i)=2^i, end, disp(v)

v =
     2     0     0     0     0     0     0     0
v =
     2     4     0     0     0     0     0     0
.
.
     2     4     8    16    32    64   128   256
```

Formula vectorisation

Problem: generate a vector of the first 8 powers of 2.

Formula vectorisation allows us to repeatedly evaluate an expression without a FOR loop. Try the following:

```
>> v=2.^(1:8), disp(v)
     2     4     8    16    32    64   128   256
```

This is an example of elementwise exponentiation with a scalar base and a vector exponent. This means evaluating the expression

```
[ 2 2 2 2 2 2 2 2].^[ 1 2 3 4 5 6 7 8]
```

```
>> u=2*ones(1,8), disp(u)
     2     2     2     2     2     2     2     2
>> disp(u.^(1:8))
     2     4     8    16    32    64   128   256
```

Newton's Method (cannot be vectorised)

```
% Newton's method of finding square roots
% textbook example, see Section 2.7 (p59)

% change the format to show more digits
format long

k=input('Approximate the square root of: ');
x=k/2;
% number of iterations
n=10;
% vector to store each approximation
v=zeros(n,1);

for i=1:n
    x=(x+k/x)/2;
    v(i)=x;
end
disp(v)
disp('Answer using the sqrt function: ')
disp(sqrt(k))

% restore default format
format
```

Nested Loops

A nested loop is a loop within another loop.

Example

```
A=zeros(4);
for i=1:4
    for j=1:4
        A(i, j)=5*i^2-j^2, % use , here to test the loop
    end
end
disp(A)
```

- Matrix A is generated in 4 x 4 = 16 iterations.
- In the first step we enter the outer loop where i is set to 1. The inner loop is then executed for each value of j from 1 to 4. Then i is incremented to 2 and the inner loop runs completely again (and so on).
- It is common practice to indent statements that belong together by the same number of spaces to indicate blocks/groups.

Practice Problem

Use any method to create a sign matrix (recall determinant calculations using the Laplace expansion method in Week 1)

Example (*5 x 5*)

1	-1	1	-1	1
-1	1	-1	1	-1
1	-1	1	-1	1
-1	1	-1	1	-1
1	-1	1	-1	1

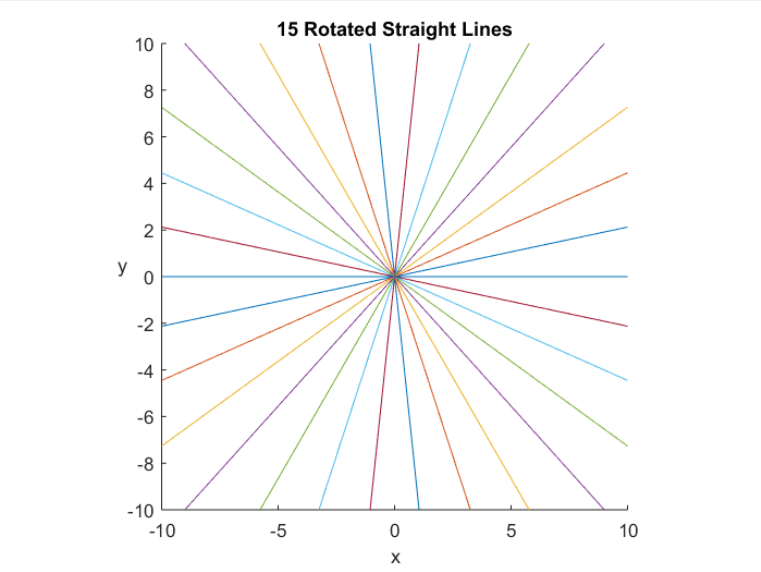
Loop Plot 1

```
% 1-loop plot example – variable gradient

close all

hold on
x=-10:0.01:10;
for i=1:15
y=tan(i*pi/15)*x;
plot(x,y)
end

% settings
axis equal
axis([-10,10,-10,10])
xlabel('x'), ylabel('y', 'rot', 0)
title('15 Rotated Straight Lines')
```



MATLAB's default plot colour order:

http://uk.mathworks.com/help/matlab/graphics_transition/why-are-plot-lines-different-colors.html

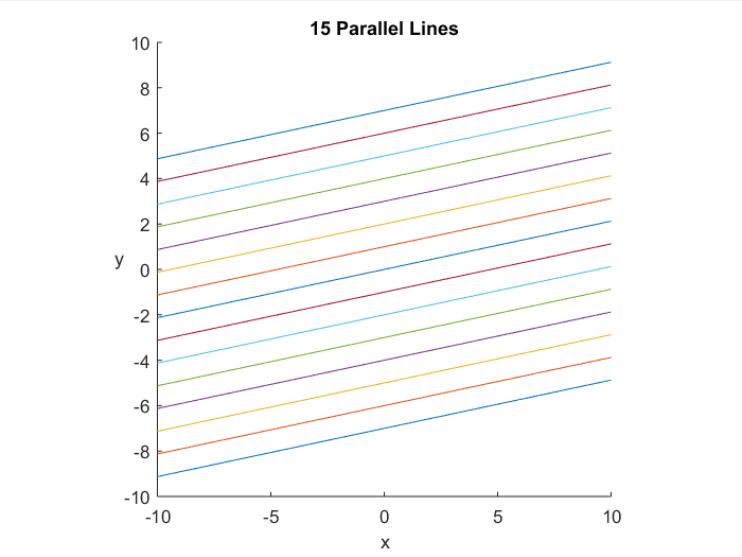
Loop Plot 2

```
% 1-loop plot example – variable intercept

close all

hold on
x=-10:0.01:10;
for i=1:15
y=tan(pi/15)*x+i-8;
plot(x,y)
end

% settings
axis equal
axis([-10,10,-10,10])
xlabel('x'), ylabel('y', 'rot', 0)
title('15 Parallel Lines')
```



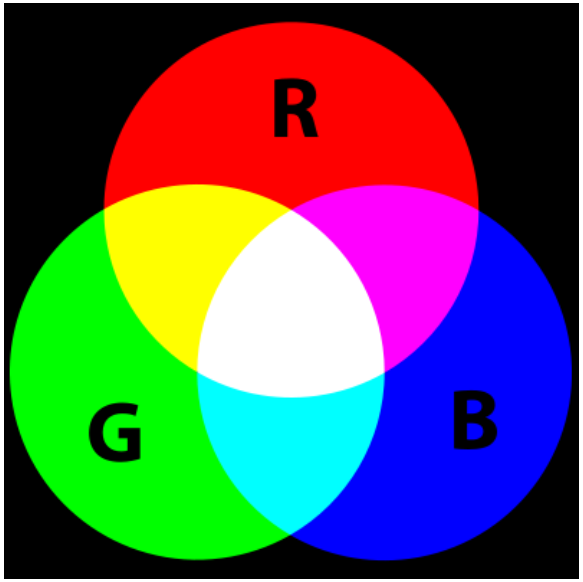
MATLAB's default plot colour order:

http://uk.mathworks.com/help/matlab/graphics_transition/why-are-plot-lines-different-colors.html

RGB Colours

The RGB colour system combines various levels of Red, Green and Blue to create a wide range of colours.

In MATLAB, the level of each component is specified on a scale from 0 to 1.



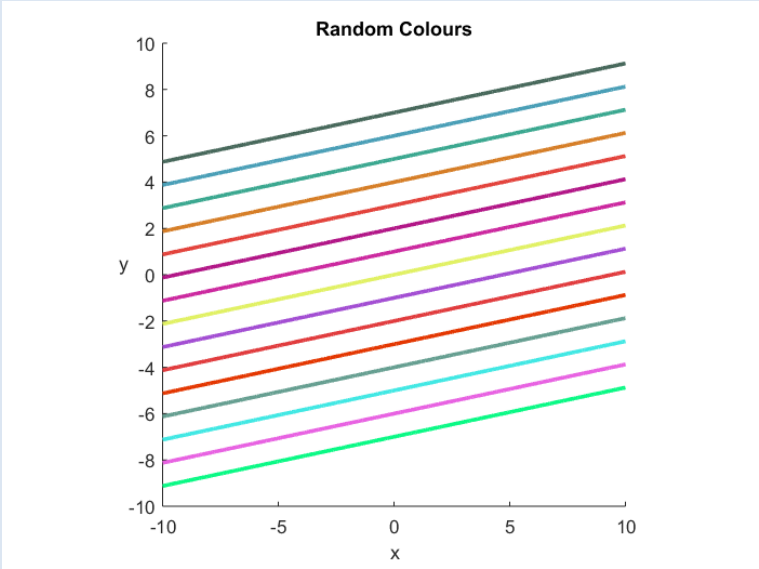
source: wikipedia

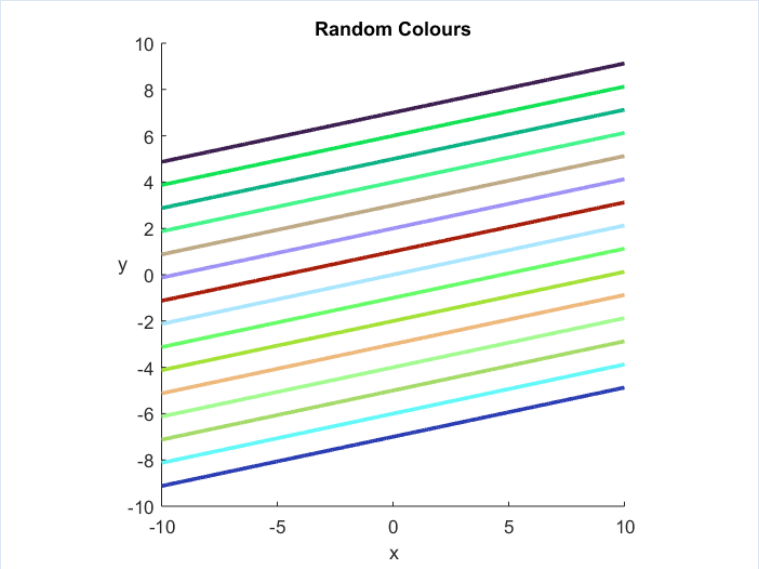
Examples	R	G	B
Black	[0	0	0]
White	[1	1	1]
Red	[1	0	0]
Yellow	[1	1	0]
Pink	[1	0	1]
Grey	[x	x	x]

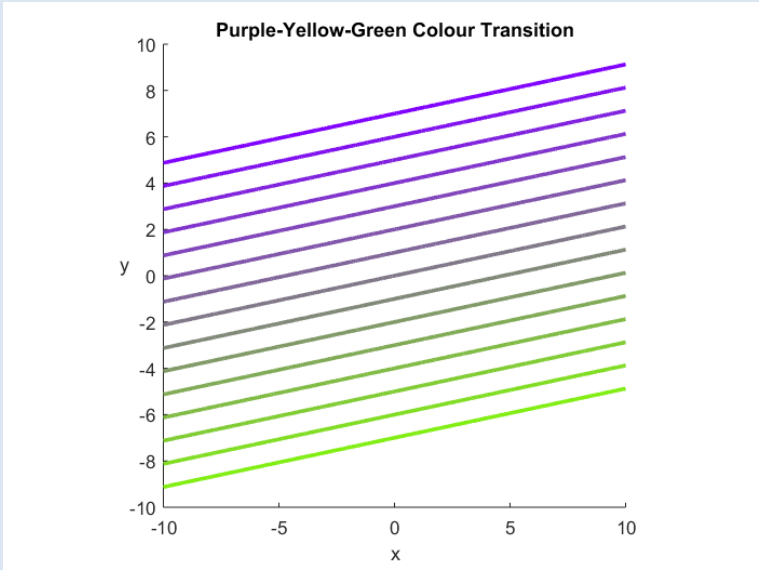
RGB triplets are 1 x 3 (row) vectors whose elements correspond to the amount of Red, Green and Blue contained in the colour they represent.

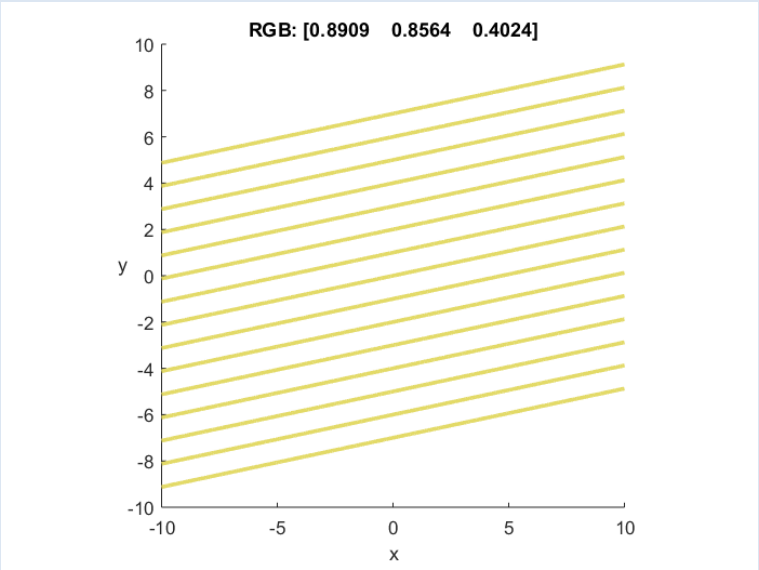
The 8 pre-defined colours can also be specified with their short or long names. For more detail, click here

RGB Colour Plot Examples









Custom RGB Colours

Test the following plot commands in Loop Plot 2

```
plot(x,y,'color', [0.5 0 1])
plot(x,y,'color', [0.5 0 1], 'linewidth', 2 )
% to create thick purple lines
```

Use the **rand** function to create random colours:

```
plot(x,y,'color', rand(1,3), 'linewidth', 2 )
% to choose a random colour in each iteration of the loop


>> disp(rand(1,3))
    0.7724    0.2280    0.3709 % deep pink (fuchsia) shade

>> disp(rand(1,3))
    0.8909    0.8564    0.4024 % pale mustard-yellow shade
```

Custom RGB Colours

Try the following plot command in Loop Plot 2:

```
plot(x,y, 'color', [1/2, 1-i/15,  i/15], 'linewidth', 2)
```

```
colour: 1:   0.5     0.93333    0.066667
colour: 2:   0.5     0.86667     0.13333
colour: 3:   0.5         0.8         0.2
colour: 4:   0.5     0.73333     0.26667
colour: 5:   0.5     0.66667     0.33333
colour: 6:   0.5         0.6         0.4
colour: 7:   0.5     0.53333     0.46667
colour: 8:   0.5     0.46667     0.53333
colour: 9:   0.5         0.4         0.6
colour: 10:  0.5     0.33333     0.66667
colour: 11:  0.5     0.26667     0.73333
colour: 12:  0.5         0.2         0.8
colour: 13:  0.5     0.13333     0.86667
colour: 14:  0.5    0.066667     0.93333
colour: 15:  0.5           0           1
```

←
Colour sequence created with the RGB vector

`[1/2, 1-i/15,i/15]`

Red: constant
Green: decreasing
Blue: increasing

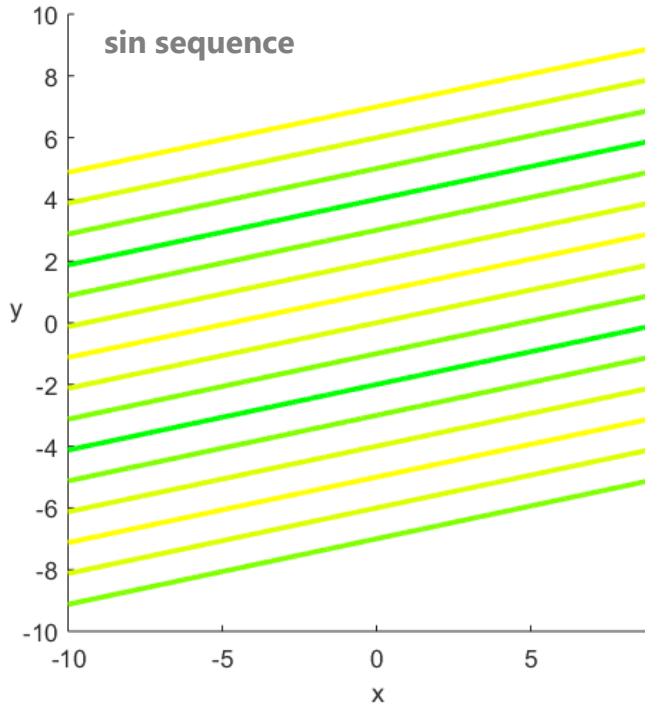
Alternating Colours

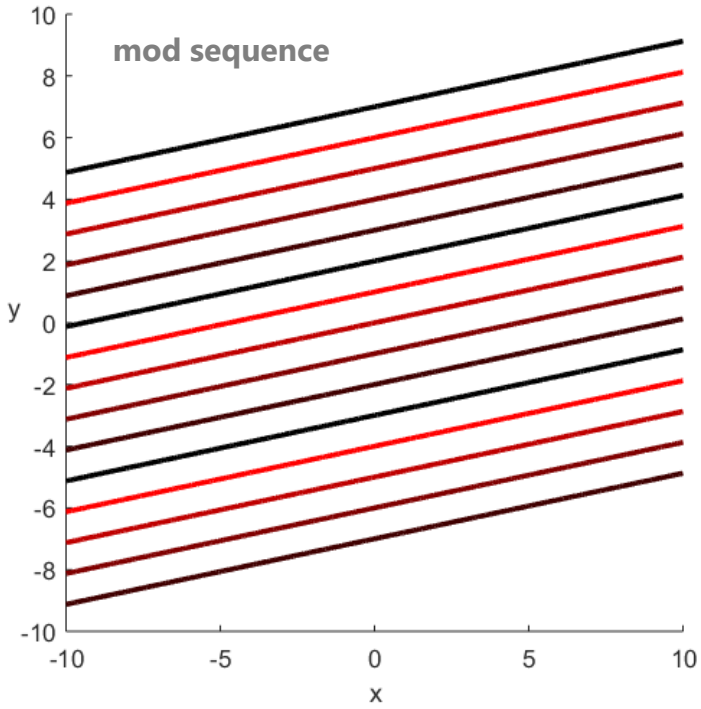
Try the following plots command in Loop Plot 2:

```
plot(x,y, 'color', [abs(sin(i*pi/6)) 1 0], 'linewidth', 2)
```

```
plot(x,y, 'color', [mod(i,5)/4, 0 0], 'linewidth', 2')
```

Both commands generate repeated colour sequences:





Polygon Plot Examples

`fill(x, y, c)` creates a ***c*-coloured 2D polygon** where the x and y coordinates of the vertices are stored in **vectors x and y**, respectively.

c can be an RGB vector or a string specifier (e.g. `'g'`).

The string `'color'` cannot be included inside the fill function.

MATLAB will always close the polygon – i.e. connects the 1st and last vertices.

Examples

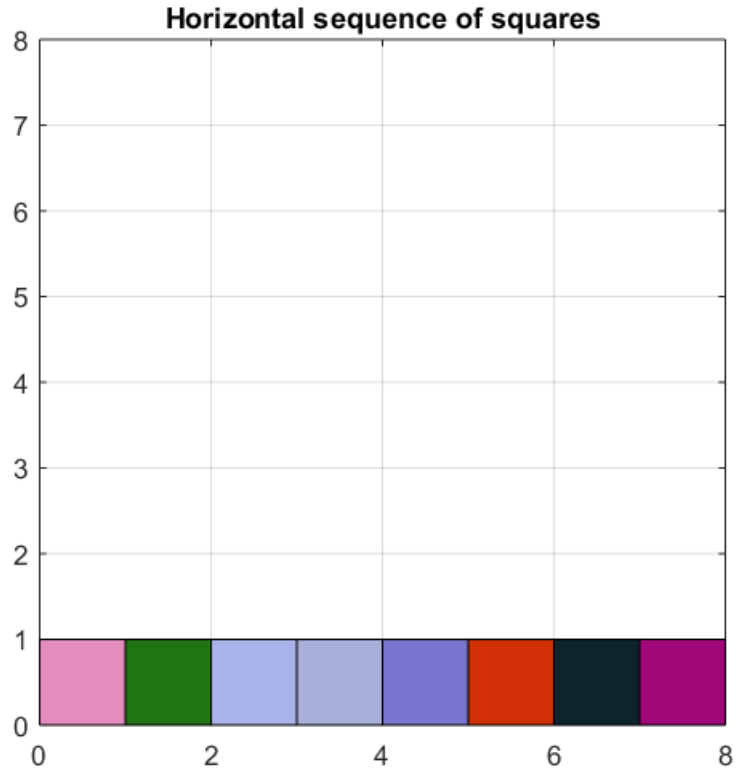
```
>> fill([0 0 1 1], [0 1 1 0], 'r')
% this is a red square

>> fill([0 1 0 1], [0 1 1 0], 'r')
% this is something else
```

Polygon Loop Plot Examples

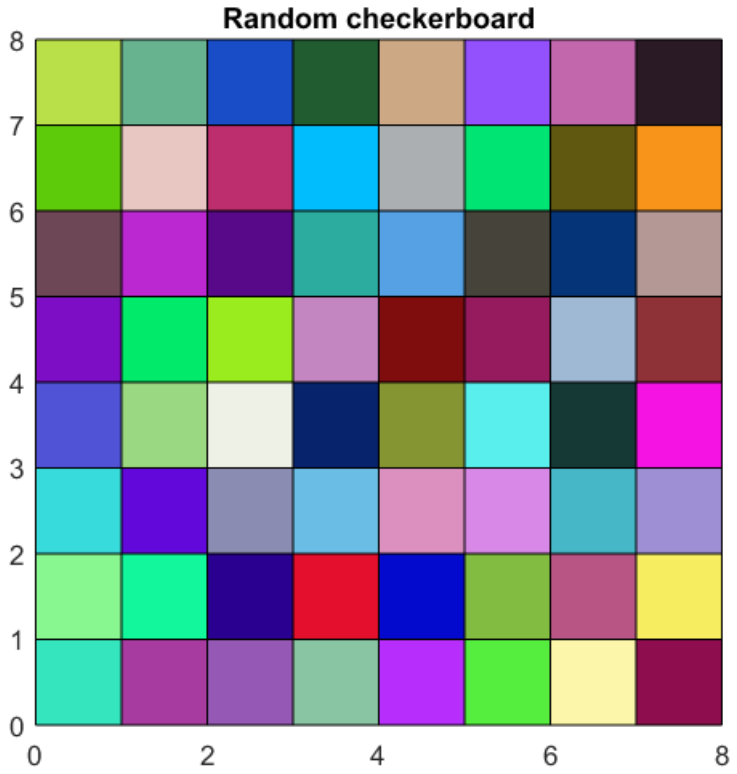
1-Loop Example

The counter controls the horizontal position (8 possible positions)



2-Loop Example

One of the counters controls the horizontal position, the other the vertical (64 combinations)



1-Loop Example

```
% horizontal sequence of squares

close all
y=[0 1 1 0]; % the y coordinate vector will not change

hold on
for i=0:7
    x=[i i i+1 i+1];
fill(x,y, rand(1,3))
end
title('Horizontal sequence of squares')
axis equal tight
axis([0 8 0 8])
box on
```

Can you change this into a checkerboard?

It usually helps to write out the results of the first few iterations (until a pattern starts to emerge).

common mistakes using plot & fill

When using RGB colour triplets:
include the `'color'` specifier in **`plot`** but do not include it in **`fill!`**
Forgetting commas is also a common mistake.

```
% correct function calls
plot(x,y, 'r')
plot(x,y, 'color', [0.5 1 0.2])
fill(x,y, 'r')
fill(x,y, [0.5 1 0.2])
```

```
% incorrect function calls
plot(x,y, [0.5 1 0.2])
fill(x,y, 'color', [0.5 1 0.2])
```

British vs American Spelling

color: *American English*
(MATLAB accepts **color** only; the textbook also uses American English)
colour: *British English*
(the spelling used in our course materials, except in copies of MATLAB code)