Lecture 5 Week 5

Mohamad Nassereddine

Lecture Contents

- Script
- Plotting
- Dynamic problem solving using Matlab
- Cell structure

Book Reference

The online textbook used in the Tutorials is: B. Hahn & D. Valentine, "Essential MATLAB for Engineers and Scientists", 7th Edition, Academic Press/Elsevier, 2019.

Chapters 9 & 12

Scripts

- Scripts allow programmers to become more productive by establishing a group of commands that can be executed in-one-go
  - Scripts usually have multiple statements with output suppressed (so they can operate silently)
  - Any outputs required can still be printed to the screen by not including the `;` character at the end of the line
  - When a script is run, all variables exist in the workspace.

A short program to calculate if a year is a leap year or not

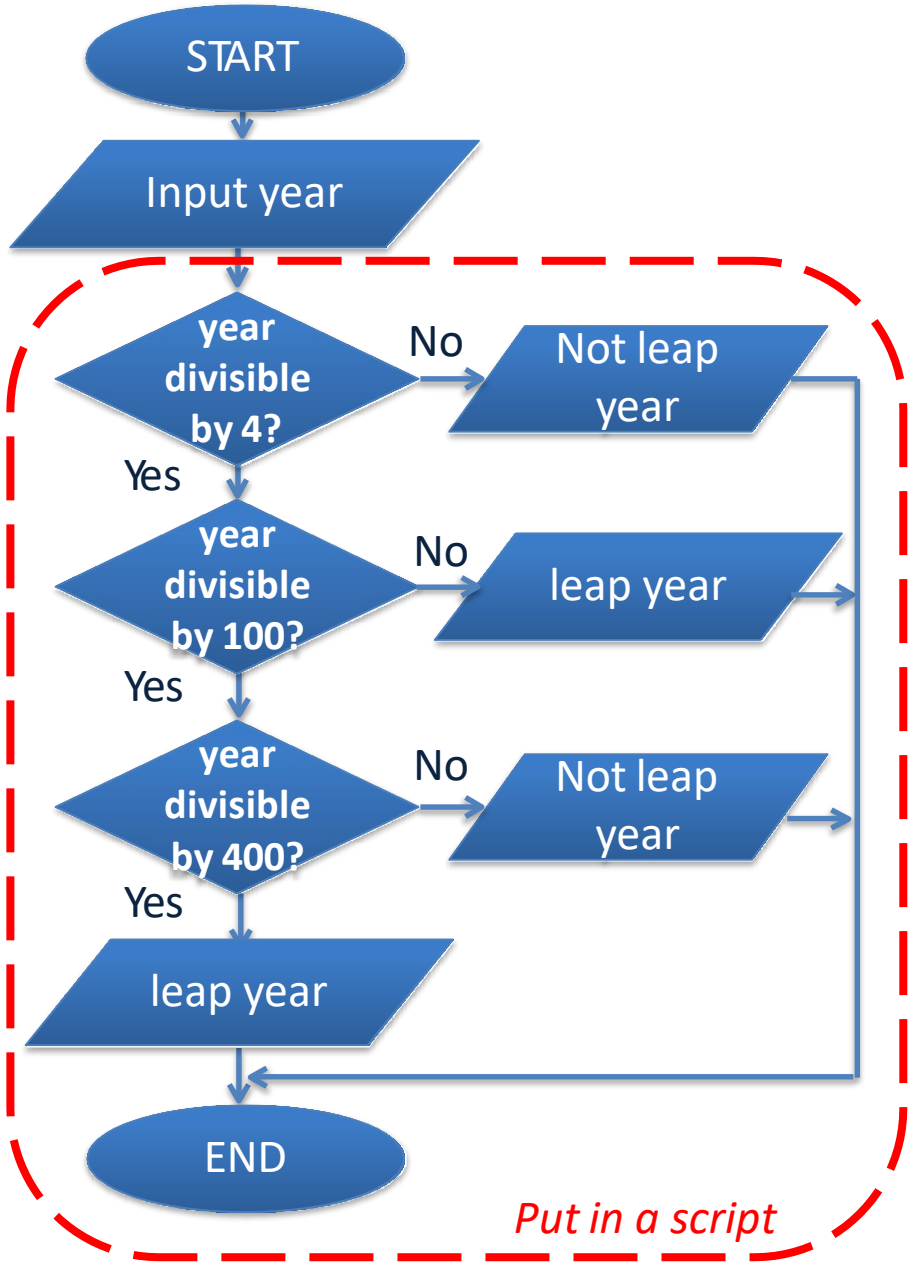
- Every **year** that is exactly **divisible** by four is a **leap year**, except for **years** that are exactly **divisible by 100**, but these centurial **years** are **leap years** if they are exactly **divisible** by 400.

- For example, the **years** 1700, 1800, and 1900 were not **leap years**, but the **years** 1600 and 2000 were.

Script Example

- A short program to calculate if a year is a leap year or not

- To test our algorithm multiple times, we can place it in a script and run from the command window with various values of '**`year`**'



Script Example

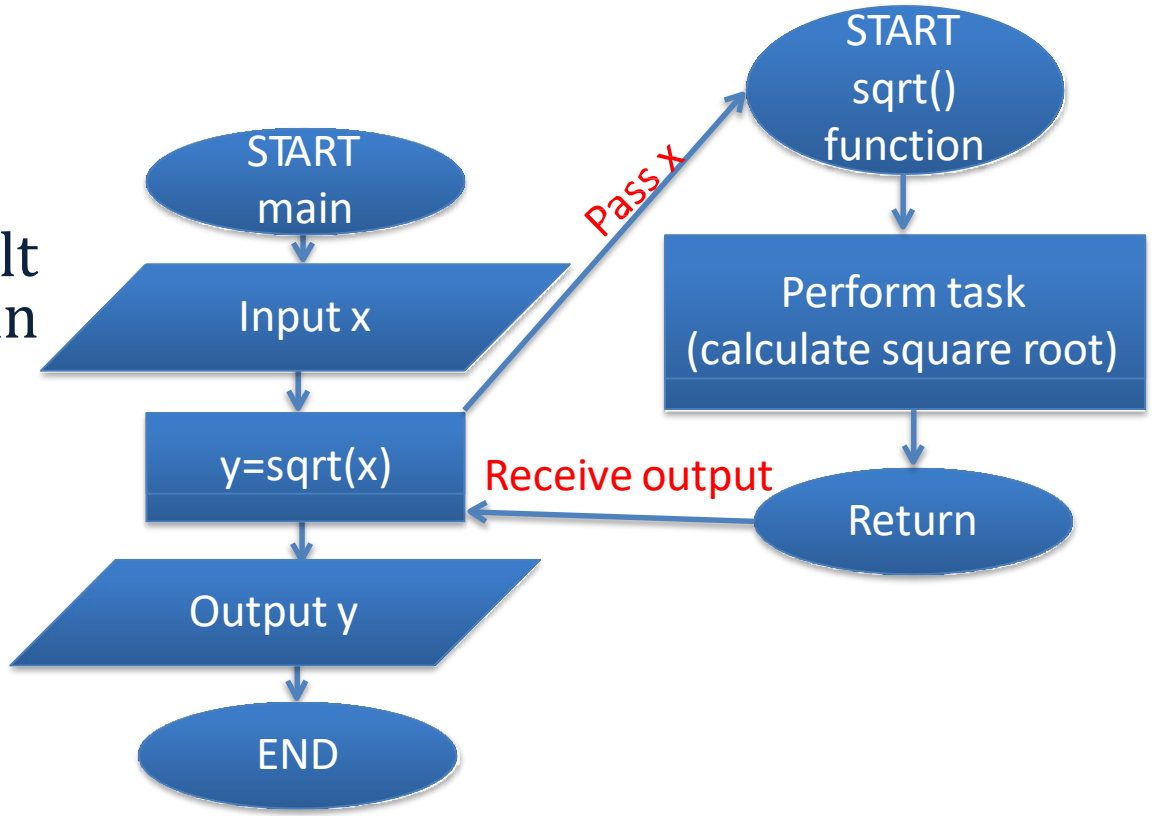
```
%% Work out if a year is a leap year:
%% Algorithm:
%%     - if year NOT divisible by 4, then common year
%%     - else, if year NOT divisible by 100, then leap year
%%     - else, if year NOT divisible by 400, then common year
%%     - else, then leap year
```

`mod (a,m)` = Remainder after division

```
year=input('Input the year: ')
if (mod(year,4) ~= 0)
    disp([num2str(year),' is a common year.']);
elseif (mod(year,100) ~= 0)
    disp([num2str(year),' is a LEAP year!!']);
elseif (mod(year,400) ~= 0)
    disp([num2str(year),' is a common year.']);
else
    disp([num2str(year),' is a LEAP year!!']);
end
```

Functions

- Functions (or sub- routines) allow for the easy re-use of code by applying the same operations to many values
- You have already been calling in-built functions, such as sqrt(). You pass an input and receive an output back.
- Today we will learn how to create our own functions.



Creating Functions

- When a function is called, the variables in the workspace are separate from the variables inside the function, unless those variables are passed to or from the function as inputs or outputs.

- The first line of a function must always be in the form:

```
function [output1, output2, …] = function_name(input1, input2, …)
```

Creating Functions

- Now let's modify the **`leapyear`** program to create two functions:
  - First function: keep output as is, within the function
  - Second function: return a logical value instead
    - In this case, the calling function or command can determine how the output should be represented

```
function Leapyear(year)

%% Work out if a year is a leap year:
%% Algorithm:
%%      - if year NOT divisible by 4, then common year
%%      - else, if year NOT divisible by 100, then leap year
%%      - else, if year NOT divisible by 400, then common year
%%      - else, then leap year


year=input('Input the year: ')
if (mod(year,4) ~= 0)
    disp([num2str(year),' is a common year.']);
elseif (mod(year,100) ~= 0)
    disp([num2str(year),' is a LEAP year!!']);
elseif (mod(year,400) ~= 0)
    disp([num2str(year),' is a common year.']);
else
    disp([num2str(year),' is a LEAP year!!']);
end
```

```
function [ TrueOrFalse ] = testLeapYear( year )

    %% Work out if a year is a leap year:
    %% Algorithm:
    %%      - if year NOT divisible by 4, then FALSE
    %%      - else, if year NOT divisible by 100, then TRUE
    %%      - else, if year NOT divisible by 400, then FALSE
    %%      - else, then TRUE

year=input('Input the year to be checked: ')
    if ( mod(year,4) ~= 0)
        TrueOrFalse = logical(false);
    elseif ( mod(year,100) ~= 0)
        TrueOrFalse = logical(true);
    elseif ( mod(year,400) ~= 0)
        TrueOrFalse = logical(false);
    else
        TrueOrFalse = logical(true);
    end
end
```

Calling functions

- A function cannot be called by clicking 'Run' because the inputs have to be defined. Instead, a function can be called in the command window or inside a script/function using the function name and parentheses (you have already been doing this).

- The input and output variable names can be different inside and outside of the function, only the position matters.

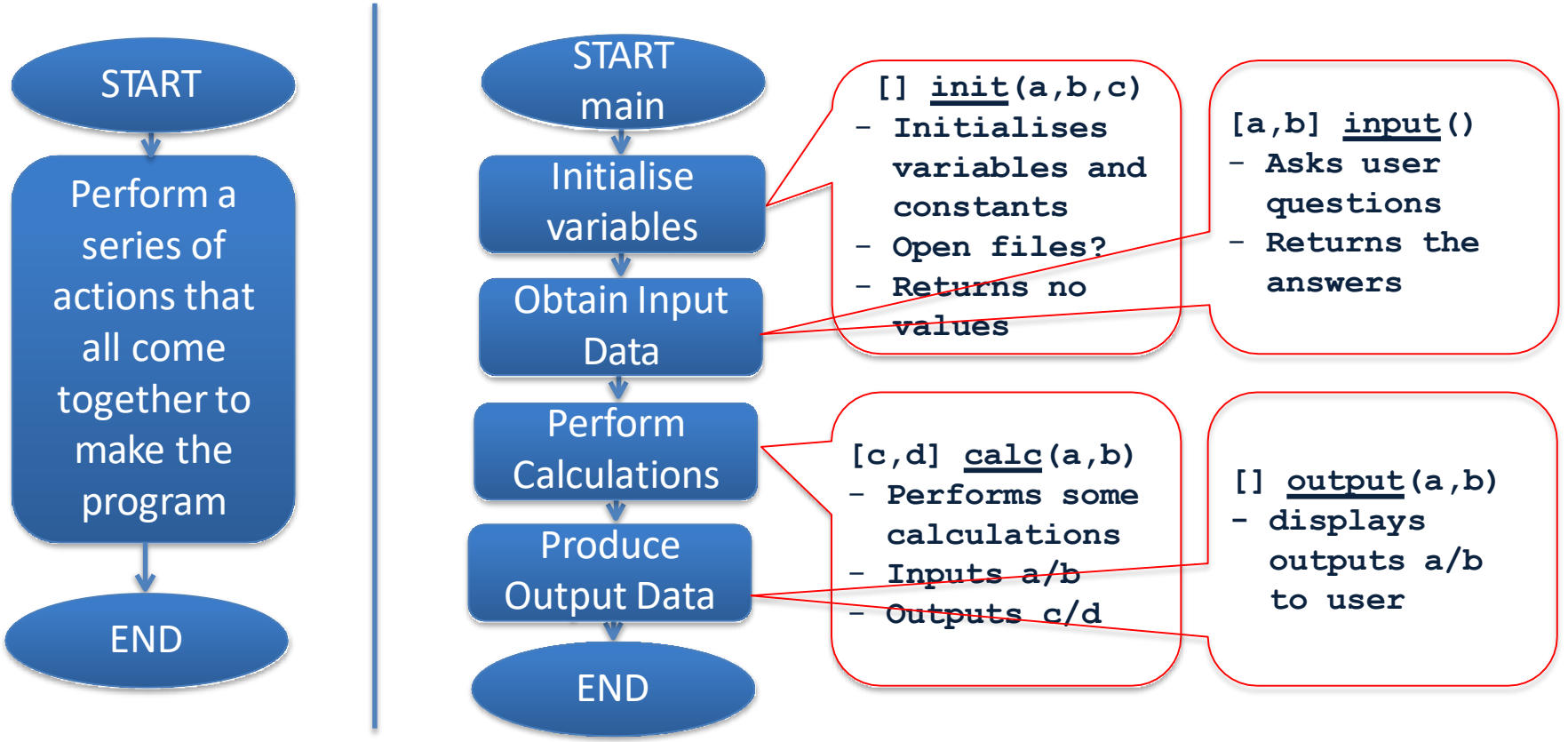
function[output1, output2, …]=function_name(input1, input2, …)

```
testLeapYear(year)
```

Modular Programming

- In our example, the second function definition allows for a greater re-use of the function titled **`testLeapYear`**
- Modular programming is a software design technique used to create compact, re-usable sections of code "modules"
  - The more modular, the more re-usable functions
  - The less modular, the less re-usable functions

Non-modular vs Modular



Modular programming

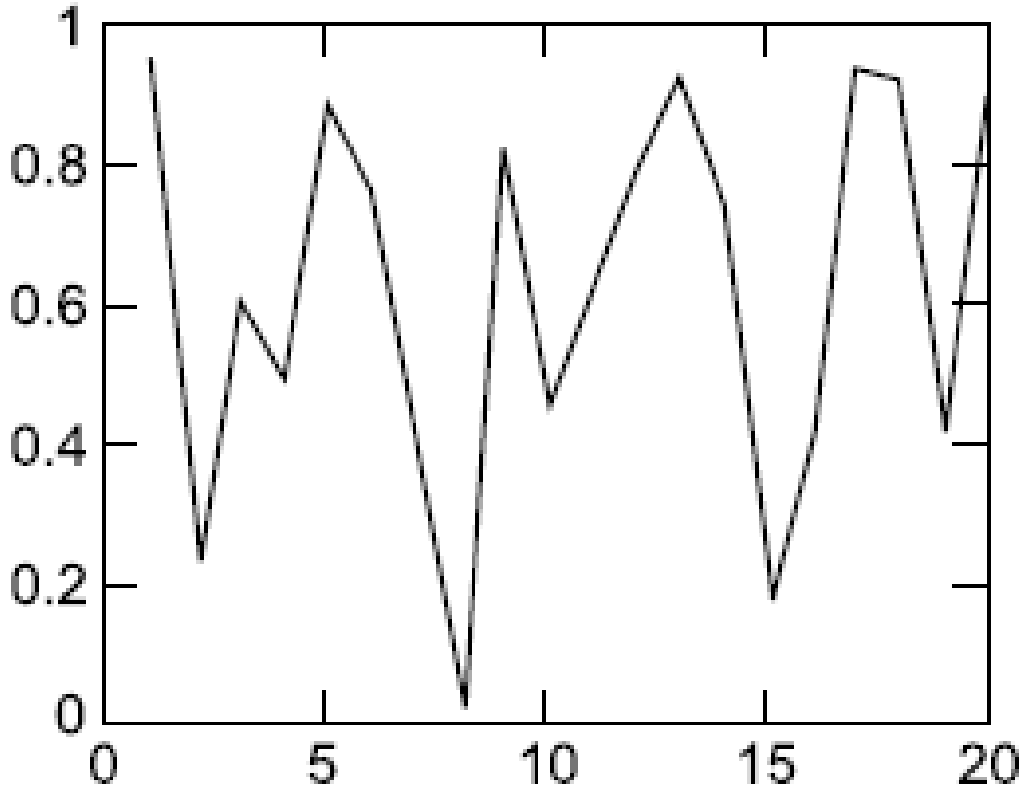
- Each function must be carefully designed and ensure that all parties understand the inputs and outputs
- Functions can be called from other functions
- Modular programming is particularly important when writing software in a team

Plotting

- This section introduces MATLAB's high-level 2-D and 3-D plotting facilities.
- It should be stressed that the treatment of graphics in this section is of necessity brief, and intended to give you a glimpse of the richness and power of MATLAB graphics.
- For the full treatment, you should consult help on the functions mentioned in this section, as well as the comprehensive list of graphics functions in Help document which you open up by clicking the '?' and typing graphics in the search tool.

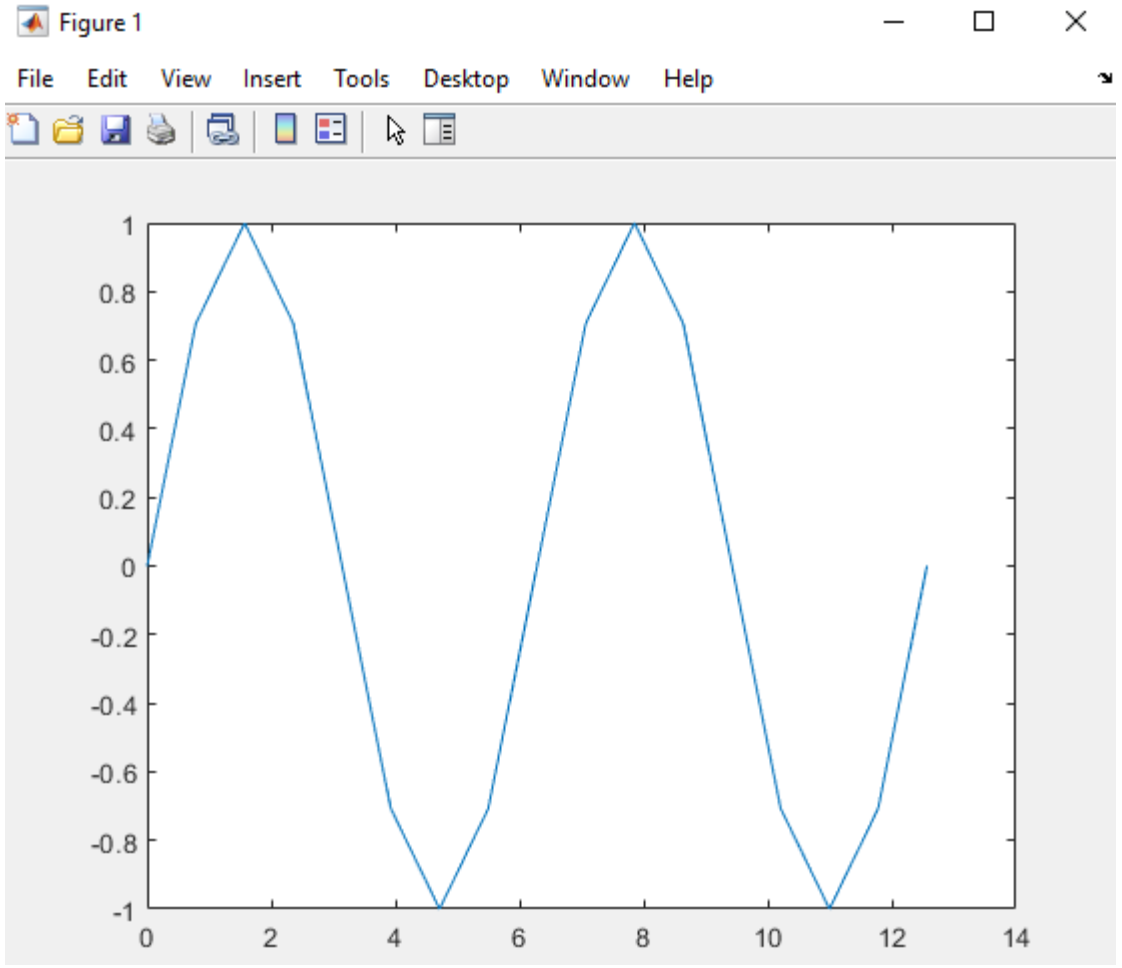
Plotting:
Basic 2 D Graph

- Graphs (in 2-D) are drawn with the plot statement. In its simplest form, it takes a single vector argument as in plot(y). In this case the elements of y are plotted against their indexes, e.g., plot(rand(1, 20)) plots 20 random numbers against the integers 1–20, joining successive points with straight lines. If y is a matrix, its columns are plotted against element indexes.



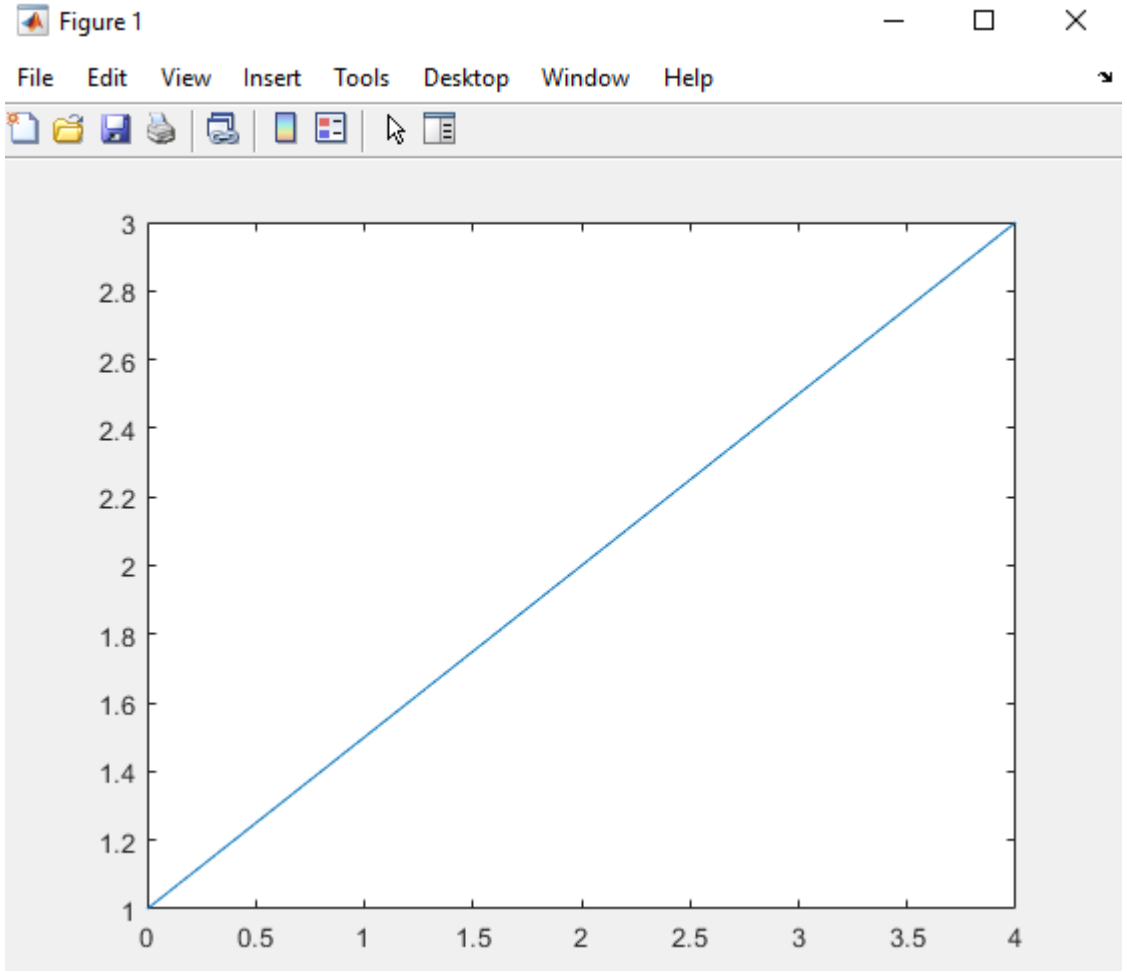
Plotting:
Basic 2 D Graph

- the most common form of plot is plot(x, y) where x and y are vectors of the same length
- x = 0:pi/40:4*pi;
- plot(x, sin(x))



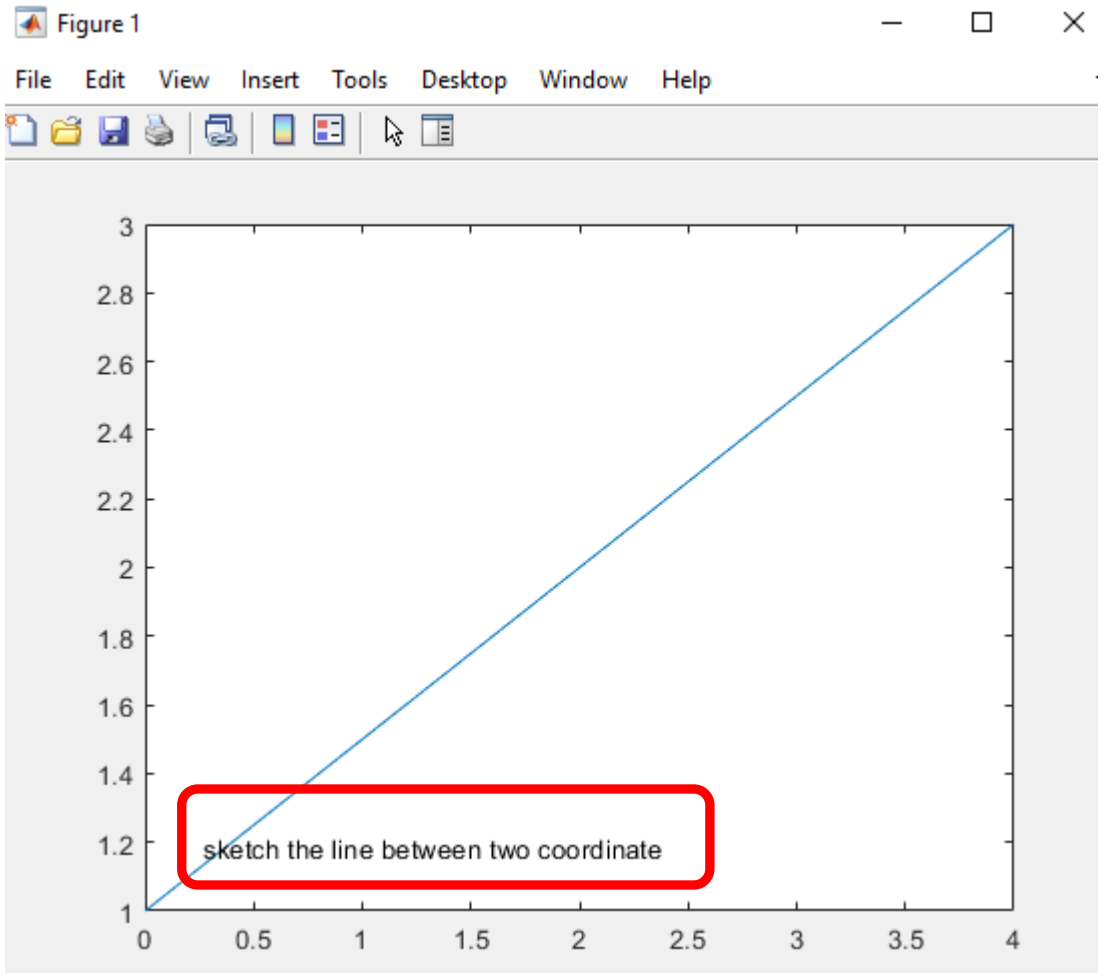
Plotting:
Basic 2 D Graph

- Straight-line graphs are drawn by giving the *x* and *y* co-ordinates of the endpoints in two vectors. For example, to draw a line between the point with cartesian co-ordinates (0, 1) and (4, 3) use the statement plot([0 4], [1 3])



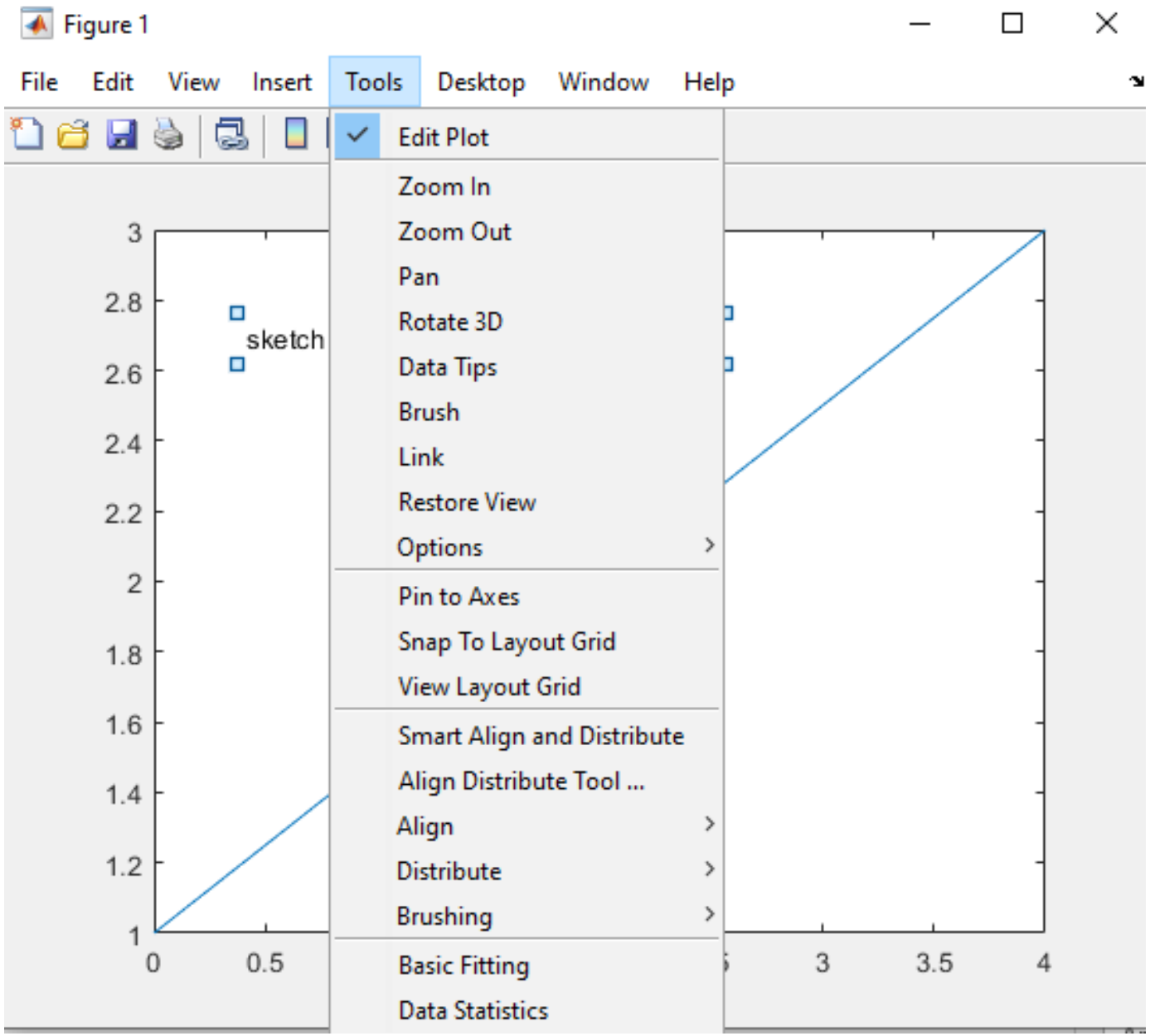
Plotting:
Basic 2 D Graph-Labels

- Graphs may be labeled with the following statements:
  - gtext('text')
- writes a string ('text') in the graph window.
- gtext puts a cross-hair in the graph window and waits for a mouse button or keyboard key to be pressed. The cross-hair can be positioned with the mouse or the arrow keys.
- For example, gtext('X marks the spot')



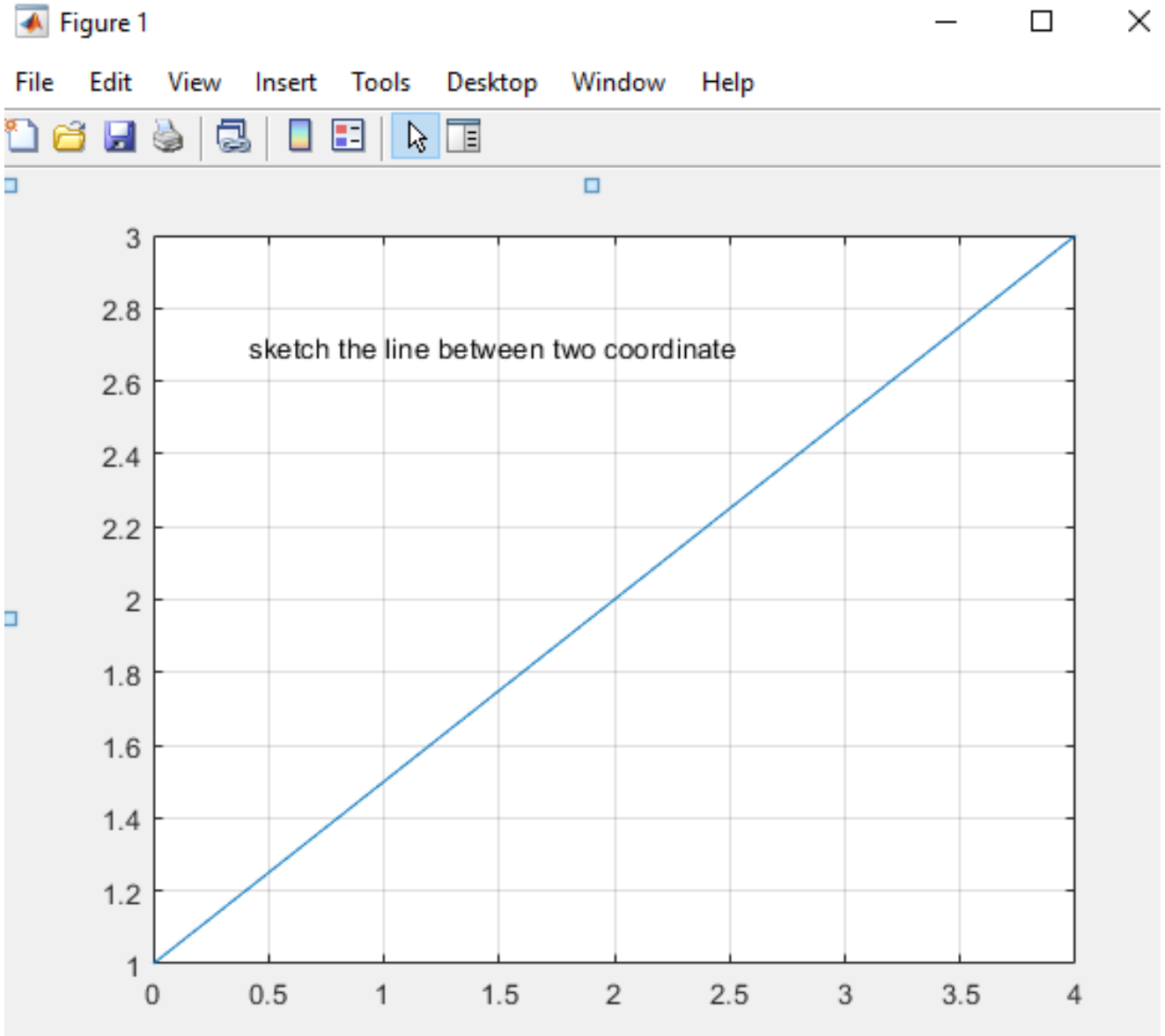
Plotting:
Basic 2 D Graph-Labels

- Text may also be placed on a graph interactively with **Tools -> Edit Plot** from the figure window.



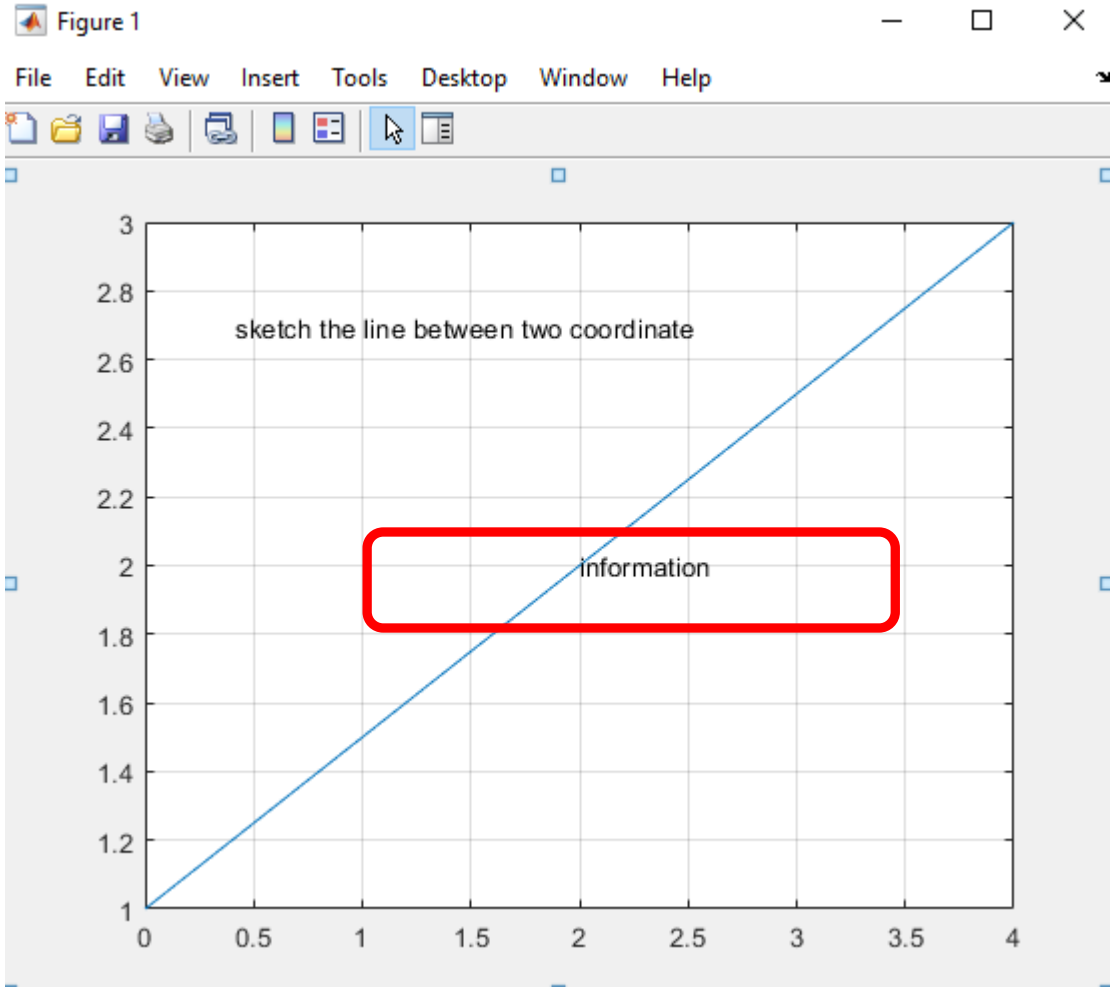
Plotting:
Basic 2 D Graph-Labels

- Grid
  - adds/removes grid lines to/from the current graph. The grid state may be toggled.



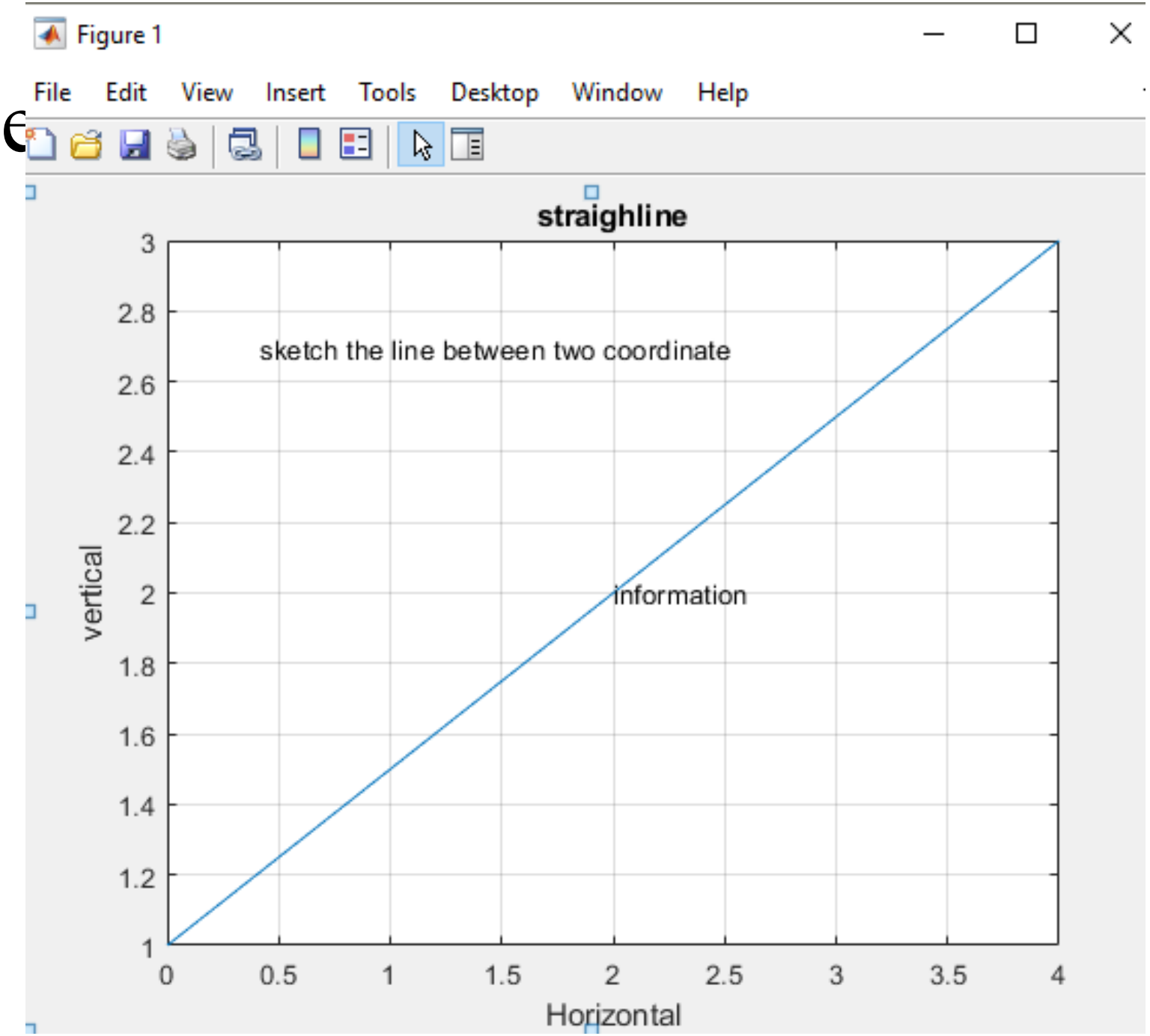
Plotting:
Basic 2 D Graph-Labels

- Text(x, y, 'text')
  - writes text in the graphics window at the point specified by x and y.
  - If x and y are vectors, the text is written at each point. If the text is an indexed list, successive points are labeled with corresponding rows of the text.
- Text(2,2,"Information")



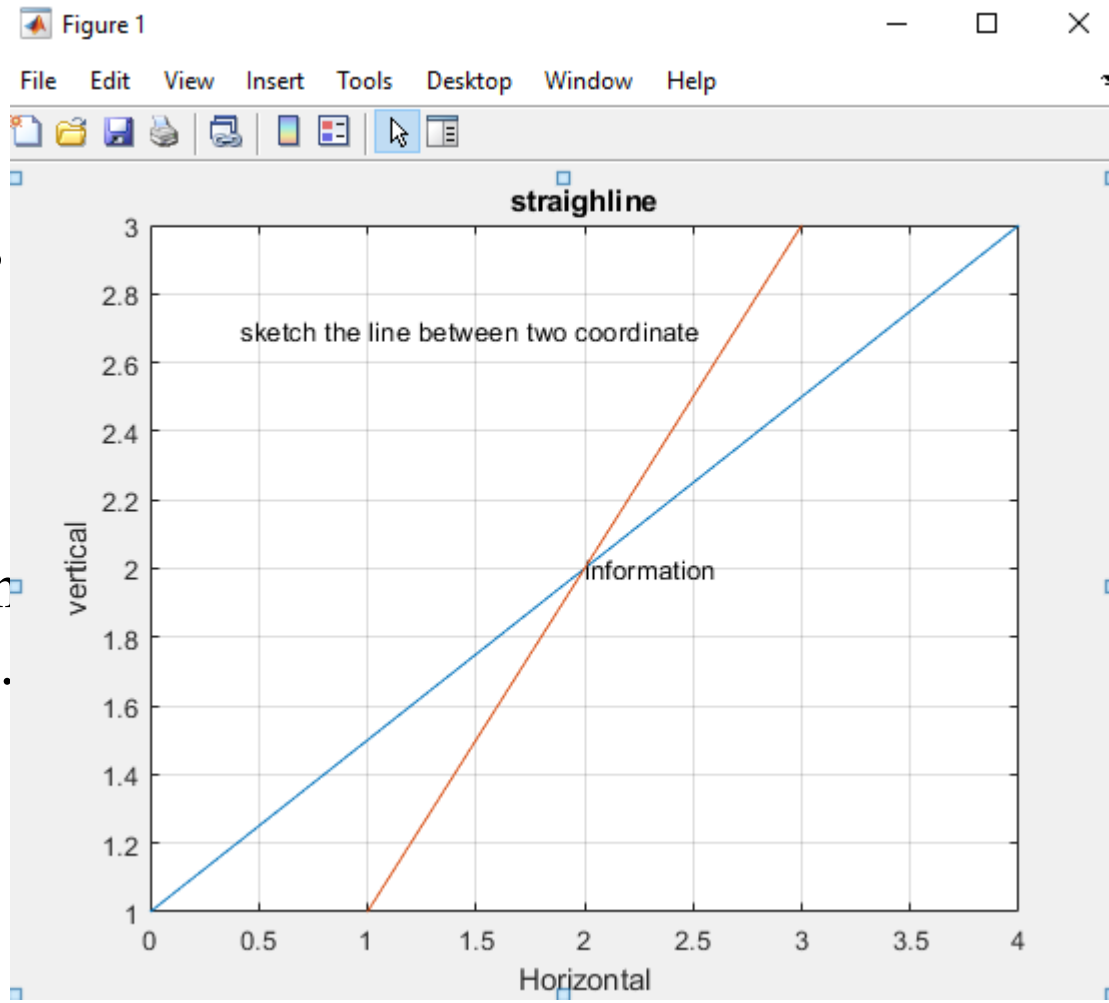
Plotting:
Basic 2 D Graph-Labe

- title('text')
  - writes the text as a title on top of the graph.
- xlabel('horizontal')
  - labels the *x*-axis.
- ylabel('vertical')
  - labels the *y*-axis.



Plotting:
Basic 2 D Graph-Multiple Plots on the same Axis

- There are at least three ways of drawing multiple plots on the same set of axes (which may however be rescaled if the new data falls outside the range of the previous data).
  - The easiest way is simply to use hold to keep the current plot on the axes. All subsequent plots are added to the axes until hold is released, either with hold off, or just hold, which toggles the hold state.
  - >>hold
  - Current plot held
  - >> plot([1 3],[1 3])
  - >> hold off



Plotting:
Basic 2 D Graph-Multiple Plots on the same Axis

- The second way is to use plot with multiple arguments, e.g., plot(x1, y1, x2, y2, x3, y3, ...)
- plots the (vector) pairs (x1, y1), (x2, y2), etc. The advantage of this
- method is that the vector pairs may have different lengths. MATLAB automatically selects a different color for each pair.

- The third way is to use the form plot(x, y) where x and y may both be matrices, or where one may be a vector and one a matrix.

Plotting:
Basic 2 D Graph

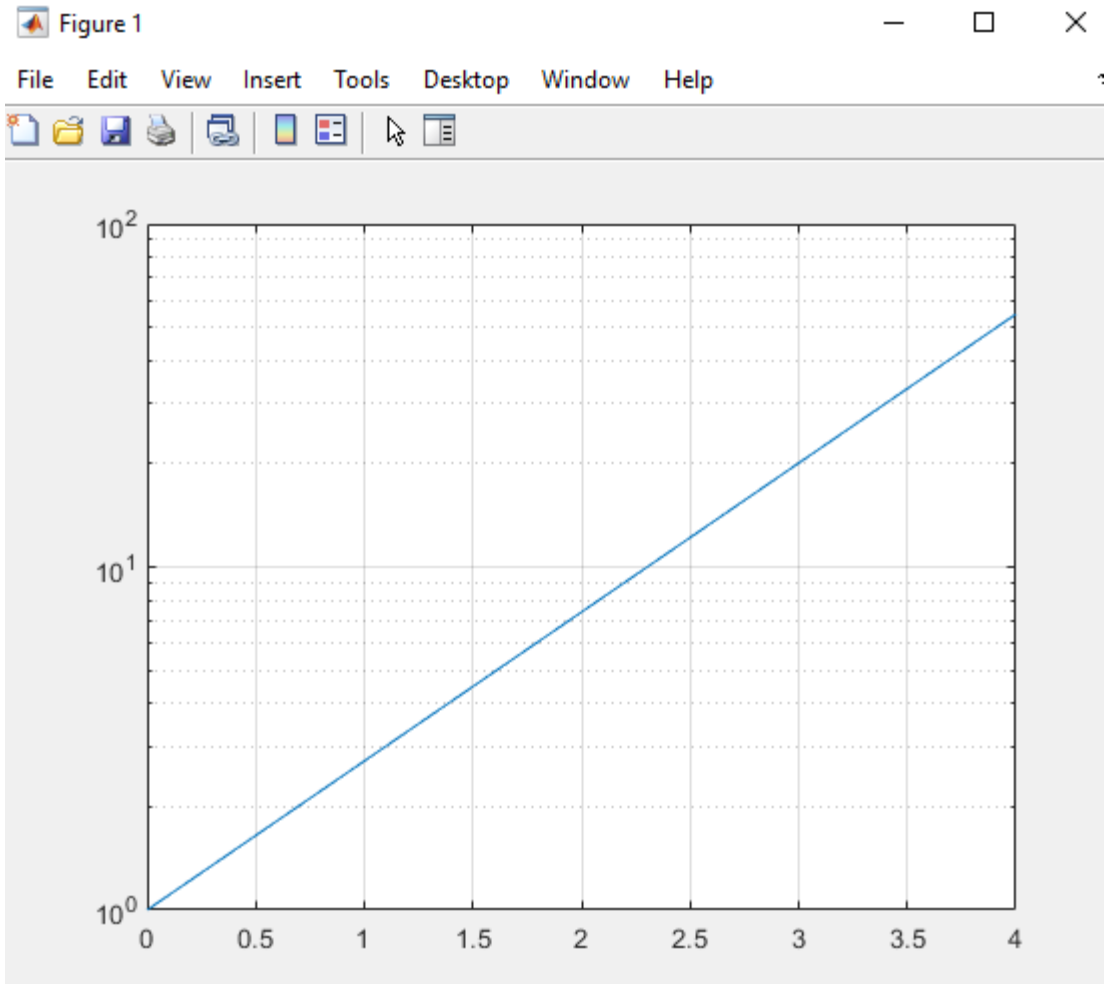
- Line styles, markers and colors may be selected for a graph with a string argument to plot, e.g.,
- plot(x, y, '--')
  - joins the plotted points with dashed lines, whereas
- plot(x, y, 'o')
  - draws circles at the data points with no lines joining them.
- You can specify all three properties, e.g.,
- plot(x,sin(x), x, cos(x), 'om--')
  - plots sin*(x)* in the default style and color and cos*(x)* with circles joined with dashes in magenta.
- The available colors are denoted by the symbols c, m, y, k, r, g, b, w. You can have fun trying to figure out what they mean, or you can use help plot to see the full range of possible symbols.

Plotting:
Basic 2 D Graph-Axis limit

- Whenever you draw a graph with MATLAB it automatically scales the axis limits to fit the data. You can override this with
  - axis([xmin, xmax, ymin, ymax])
- which sets the scaling on the *current* plot, i.e., draw the graph first, then reset the axis limits.
- Let us try an example

Plotting:
Basic 2 D Graph-Logarithm Plot

- The command
- semilogy(x, y)
- plots y with a log10 scale and x with a linear scale. For example, the statements
  - x = 0:0.01:4;
  - semilogy(x, exp(x)), grid



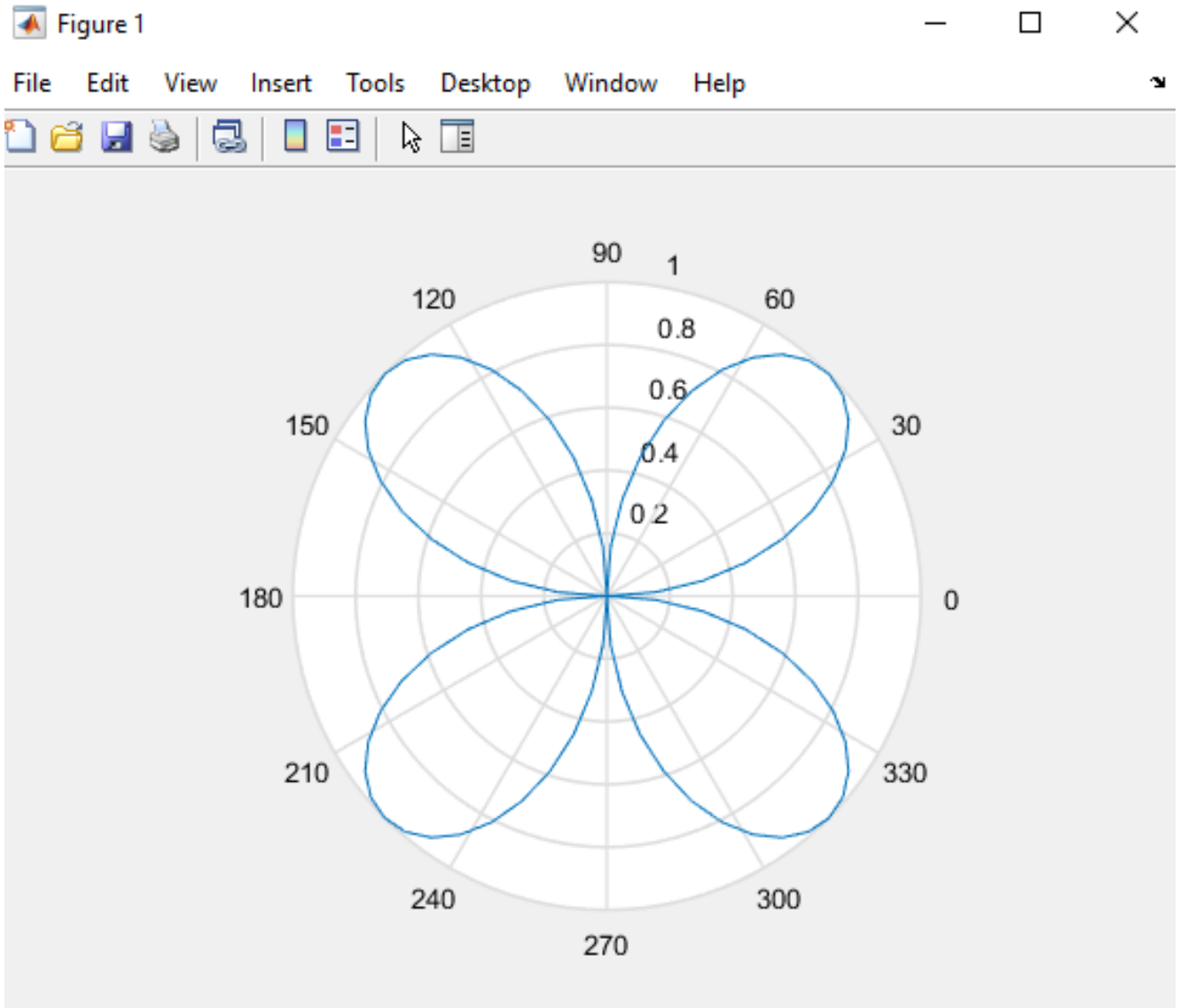
Plotting:
Basic 2 D Graph-Polar Plot

- The point (x, y) in cartesian co-ordinates is represented by the point (θ, r) in
- *polar* co-ordinates, where
- $x = r\cos(\theta)$,
- $y = r\sin(\theta)$,
- and θ varies between 0 and 2π radians ($360\circ$).
- The command
  - polar(theta, r)

Plotting:

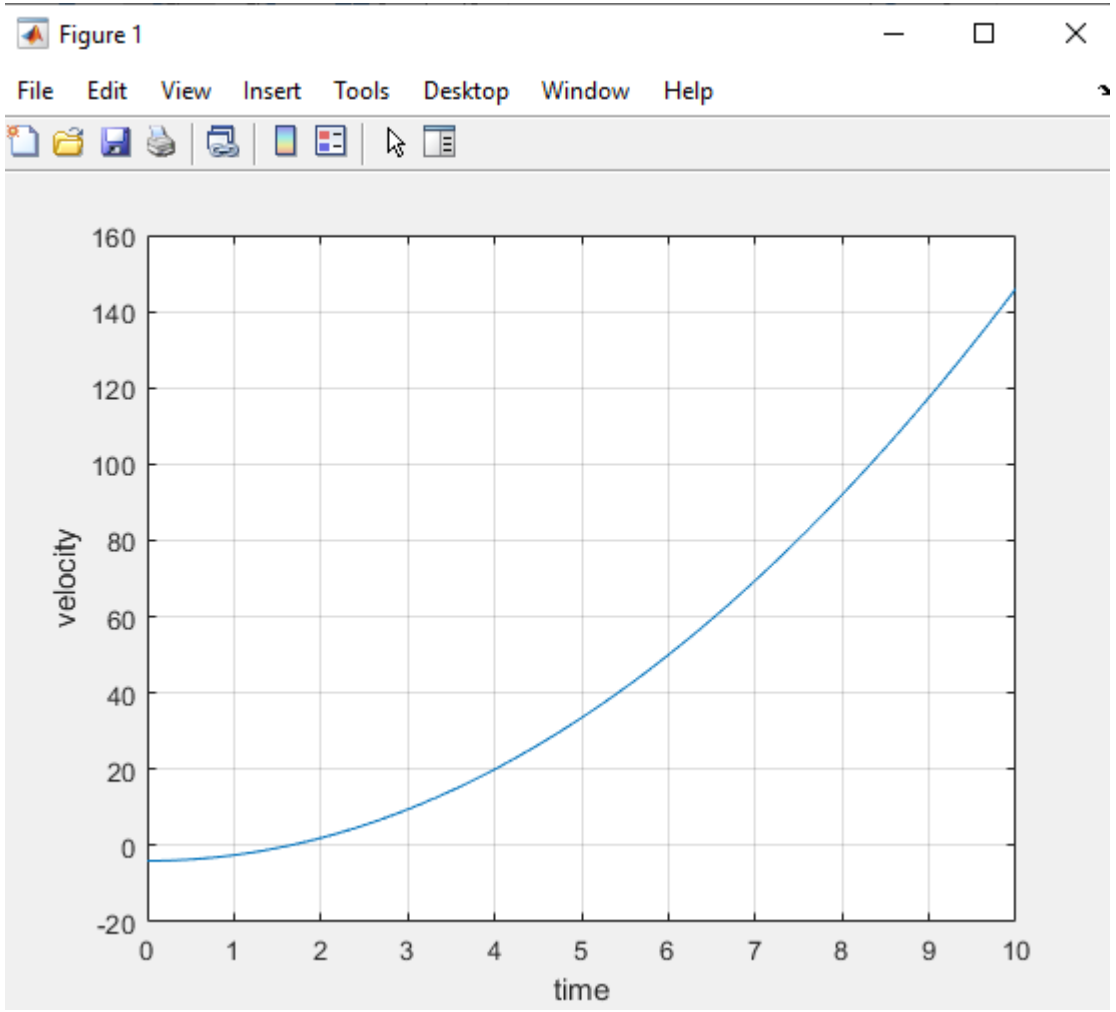
Basic 2 D Graph-Polar Plot

- >> x=0:pi/40:2*pi;
- >> polar(x,sin(2*x)),grid



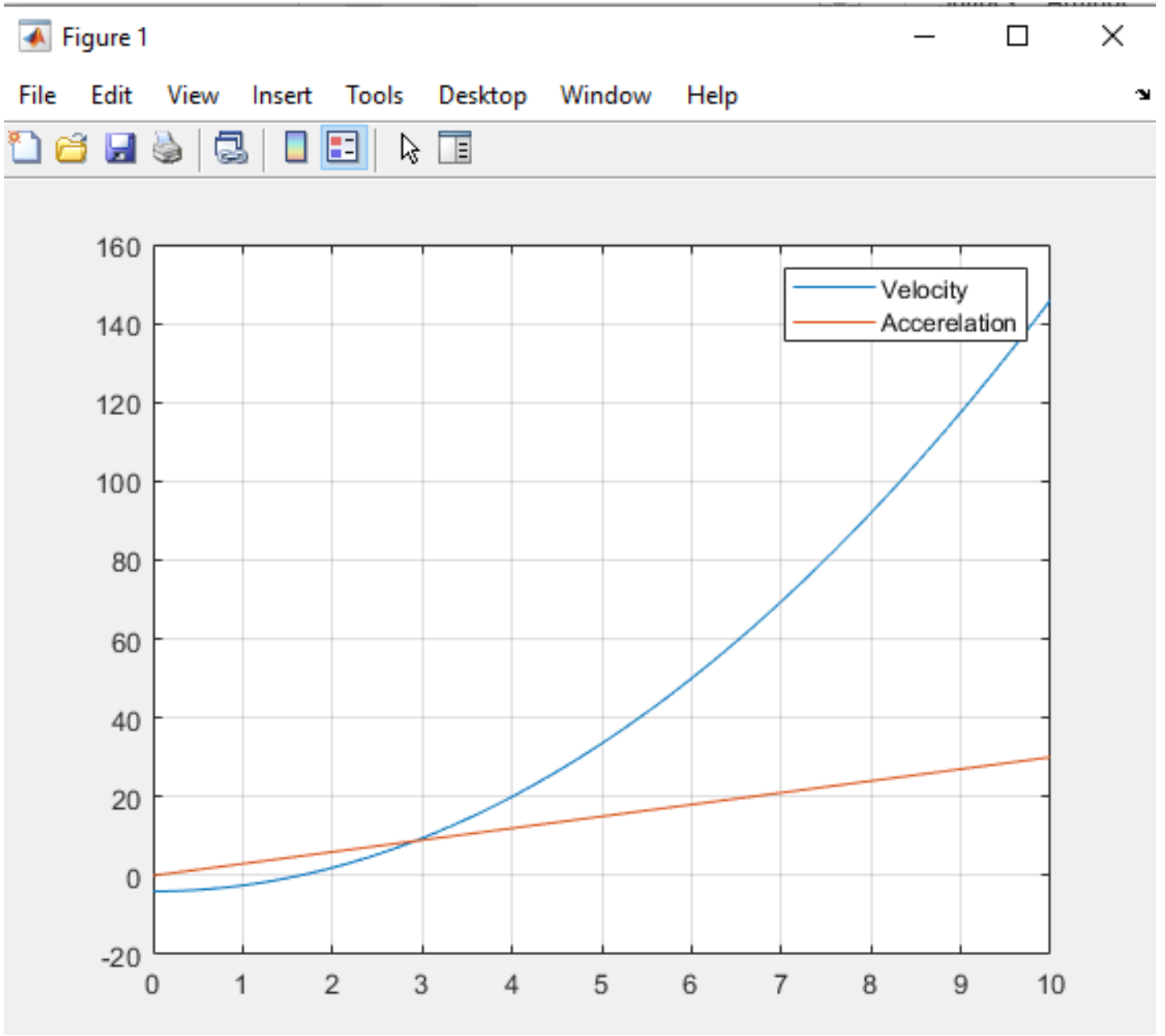
Matlab:
Dynamics

- The position of a particle travelling along a straight path is given by the equation $s = 0.5t^3 - 4t^2 + 6t$ m, where t is measured in seconds.
- Determine the maximum acceleration and the maximum velocity of the particle during the time interval $0 \le t \le 10$ seconds.
  - >> t=0:0.01:10;
  - >> s=0.5*t.^3-4*t+6;
  - >> v=1.5*t.^2-4;
  - >> a=3*t;
  - >> plot(v,t),grid
  - >> plot(t,v),grid
  - >> xlabel("time")
  - >> ylabel("velocity")



Matlab: Dynamics

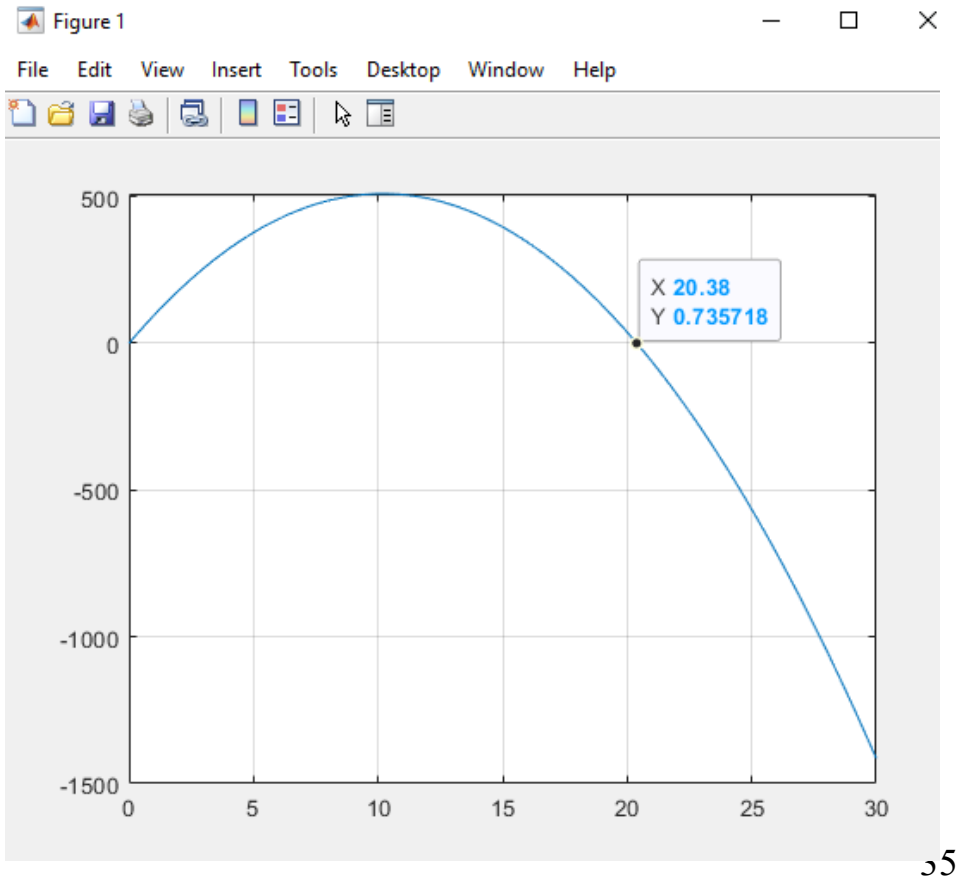
- Plot acceleration
  - >> t=0:0.01:10;
  - >> s=0.5*t.^3-4*t+6;
  - >> v=1.5*t.^2-4;
  - >> v=1.5*t.^2-4;
  - >> a=3*t;
  - >> plot(v,t),grid
  - >> plot(t,v),grid
  - >> xlabel("time")
  - >> ylabel("velocity")
  - >> plot(t,a),grid
  - >> plot(t,v,t,a),grid
  - >> legend("Velocity","Acceleration")



Matlab:

Dynamics

- An object is moving in projectile motion. The object left origin at a speed of 200m/s and an angle 30 degrees.
- $Y = 200\sin(30)\,t + (-4.905)t^2$
- Plot Y
- >> t=0:0.01:30;
- >> y=100*t-4.905*t.^2;
- >> plot(t,y), grid



Matlab:
Dynamics

- An object is moving in projectile motion. The object left origin at a speed of 200m/s and an angle α.
- Determine the angle α if the object reaches the ground after 5 seconds?
- $Y = 200\sin(\alpha)\,t + (-4.905)t^2$
- $\sin(\alpha) = \frac{Y+4.905t^2}{200t} = \frac{0+4.905t^2}{200t}$
- t=0.1:0.1:10;
- a=4.905*t.^2;
- b=200*t;
- >> c=a./b;
- >> plot(t,c),grid
- >> d=asin(c)
- plot(t,d), grid