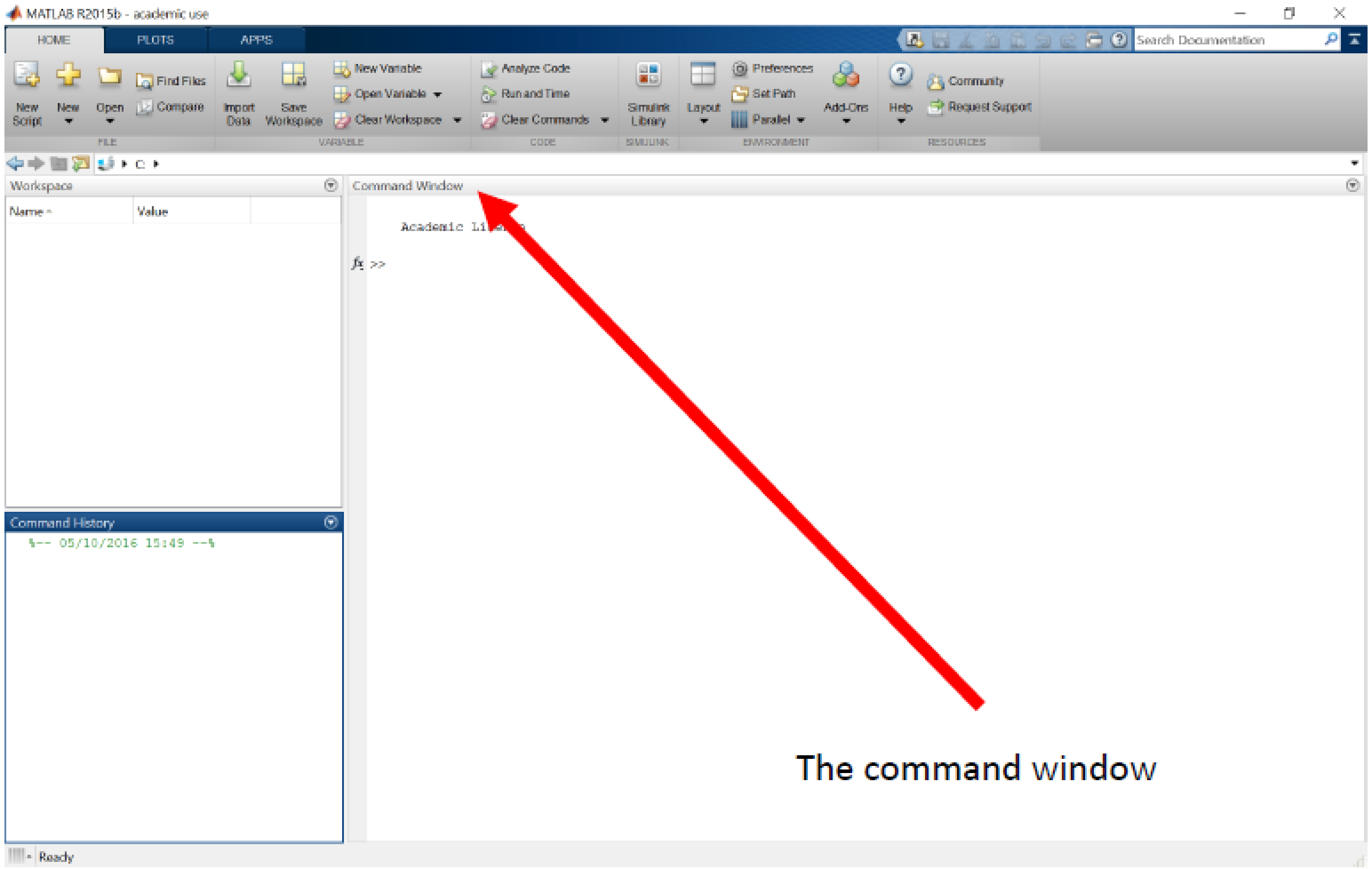
Introduction to MATLAB

MATLAB Screen

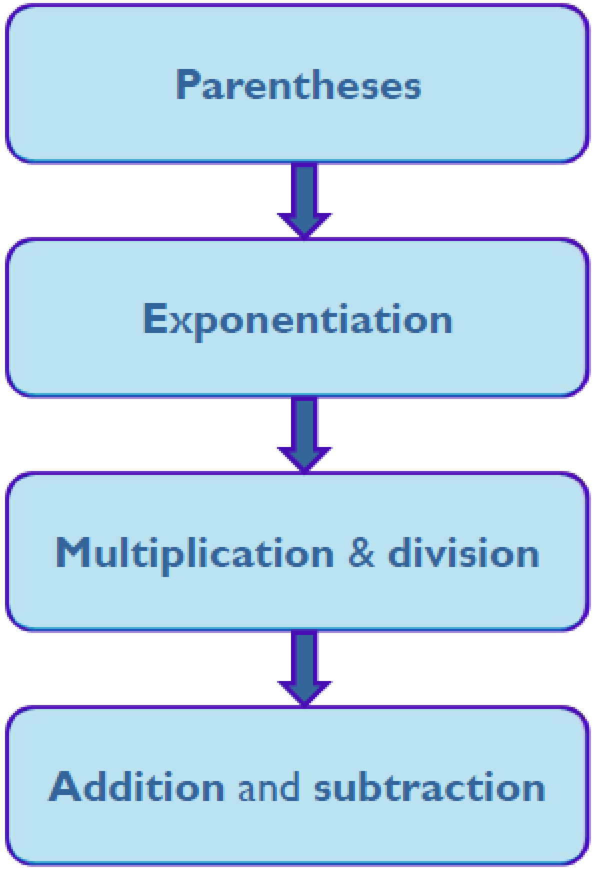


The exact layout can differ from machine to machine, but the windows are always labelled!

Matlab as interactive calculator

```
>> 8/10
ans=
      0.8000
>> 5*ans
ans=
      4
>> r=8/10
r =
      0.8000
>> r
r =
      0.8000
>> s=20*r
s =
      16
```

Order of Precedence of Arithmetic Operators



Note: if precedence is equal, evaluation is performed from left to right.

Scalar Arithmetic Operations

Symbol	Operation	Mathematical Syntax	Matlab Syntax
^	Exponentiation	a^b	a^b
*	Multiplication	ab	a*b
/	Forward Division	a/b	a/b
\	Backward Division	a\b	a\b
+	Addition	a+b	a+b
-	Subtraction	a-b	a-b

Examples of Precedence

```
>> 8 + 3*5
ans=
     23
>> 8 + (3*5)
ans=
     23
>> (8 + 3)*5
ans=
     55
>> 4^2-12- 8/4*2
ans=
     0
>> 4^2-12- 8/(4*2)
ans=
```

Commonly Used Mathematical Functions

Function	Matlab Syntax
e^x	exp(x)
$\sqrt{x}$	sqrt(x)
ln x	log(x)
$\log_{10} x$	log10(x)
cos x	cos(x)
sin x	sin(x)
tan x	tan(x)
$\cos^{-1} x$	acos(x)
$\sin^{-1} x$	asin(x)
$\tan^{-1} x$	atan(x)
|x|	abs(x)

NOTES

- *Trigonometric functions in Matlab use radian measure*
- *$\cos^2(x)$ is written (cos(x))^2 in Matlab*

Row Vectors

- Row vector: comma or space separated values between brackets

```
row = [1 2 5.4 -6.6]
row = [1, 2, 5.4, -6.6];
```

- Command Window:

```
>> row = [1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

Column Vectors

- Column vector: semicolon separated values between brackets

```
col = [4;2;7;4]
```

- Command Window:

```
>> col = [4;2;7;4]

col =

    4.0000
    2.0000
    7.0000
    4.0000
```

Size & Length

- You can tell the difference between a row and a column vector by:
- ➢ Looking in the workspace
- ➢ Displaying the variable in the command window
- ➢ Using the `size` function:

```
>> size(row)

ans =

     1     4
```

```
>> size(column)

ans =

     4     1
```

- To get a vector's length, use the `length` function:

```
>> length(row)

ans =

     4
```

```
>> length(column)

ans =

     4
```

Other Methods for Creating Vectors

- The colon operator (:) easily generates a large vector of **regularly spaced** elements.

`x = [m:q:n]` → to create a vector `x` of values with a spacing = `q`
The first value is `m`, the last value is `n` *if* `m-n` *is an integer multiple of* `q`. If not, the last value is less than `n`.

The number of elements = `((n-m)/q)+1`

- Examples:

`x=[0:2:8]` creates the vector x=[0,2,4,6,8]
`x=[0:2:7]` creates the vector x=[0,2,4,6]

- Default increment:

If the increment `q` is omitted, it is assumed to be `1`.

`y = [-3:2]` produces the vector y=[-3,-2,-1,0,1,2]

Other Methods for Creating Vectors

- The `linspace` command also creates a linearly spaced row vector, but instead you specify the number of elements rather than the increment.

`y = linspace(x1,x2,n)` where `x1` and `x2` are the lower and upper limits and `n` is the number of points.

Here the increment = `(x2-x1)/(n-1)`

- Examples:

`linspace(5,8,31)` is equivalent to [5:0.1:8]

- Default increment:

If `n` is omitted, the spacing is `1`.

Automatic Initialization

- Identity Matrix (I)

  `eye(n), eye(m,n)`

- All-ones matrix

  `ones(n), ones(m,n), ones(size(A))`

- All-zeros matrix

  `zeros(n), zeros(m,n), zeros(size(A))`

- Matrix of Random numbers (between 0 and 1)

  `rand(n), rand(m,n), rand(size(A))`

- Matrix of Not a Number

  `nan(n), nan(m,n), nan(size(A))`

- Matrix with elements only in the diagonal:

  `diag([x1,x2,…,xn])`

Basic Array Functions

Function	Description
size(A)	Returns a row vector [m n] containing the size of the mxn array A
sort(A)	Sorts each column of the array A in ascending order and returns an array of same size as A
sum(A)	Sums the elements in each column of the array A and returns a row vector containing the sums
inv(A)	Computes the inverse of array A
diag(A)	Returns the elements along the main diagonal of A
fliplr(A)	Flips array A about it central column
flipud(A)	Flips array A about it central row

: find(A), max(A) , min(A), cat(n,A,B,C)

Some Vector Functions

- The **transpose** operator turns a column vector into a row vector and vice versa.

```
>> a = [1 2 3 4];
>>transpose(a)
>>a'
```

- You can **sum** or **multiply** the elements of a vector

```
>> a = [1 2 3 4];
>>s = sum(a)
>>p = prod(a)
```

Addition and Subtraction

- Addition and Subtraction are element-wise operations; sizes must match (unless one is a scalar)

```
>> r1=[12 3 32 -11];
>> r2=[2 11 -30 32];
>> c1=[12;1;-10;0];
>> c2=[3;-1;13;33];
>> a=r1+r2

a =

    14    14     2    21

>> b=c1+c2

b =

    15
     0
     3
    33
```

```
>> r1+c1
Error using  +
Matrix dimensions must
agree.

>> r1+c1'

ans =

    24     4    22   -11
```

Element-wise Functions

- All the functions that work on scalars also work on vectors.

```
>> t = [1 2 3];
>> f = exp(t)
```

Is the same as

```
>> f = [exp(1) exp(2) exp(3)]
```

- If in doubt, check a function's help file to see if it handles vectors element-wise.

- Operators `(* / ^)` have two modes of operation:

→ Element-by-Element

→ Standard

Operators: Element-by-Element

- To do element-wise operations, use the dot: `(.*   ./   .^)`
- BOTH dimensions must match, unless one is scalar.

```
>> a = [1 2 3];
>> b = [4 2 1];
>> a.*b %Okay
>> a*b %Error
```

$$\begin{bmatrix}1 & 1 & 1\\ 2 & 2 & 2\\ 3 & 3 & 3\end{bmatrix} .* \begin{bmatrix}1 & 2 & 3\\ 1 & 2 & 3\\ 1 & 2 & 3\end{bmatrix} = \begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 6\\ 3 & 6 & 9\end{bmatrix}$$

$$3\times 3 .* 3\times 3 = 3\times 3$$

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} .* \begin{bmatrix}4\\ 2\\ 1\end{bmatrix} = ERROR$$

$$\begin{bmatrix}1\\ 2\\ 3\end{bmatrix} .* \begin{bmatrix}4\\ 2\\ 1\end{bmatrix} = \begin{bmatrix}4\\ 4\\ 3\end{bmatrix}$$

$$3\times 1 .* 3\times 1 = 3\times 1$$

$$\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} .\hat{}\ 2 = \begin{bmatrix}1^2 & 2^2\\ 3^2 & 4^2\end{bmatrix}$$

Can be any dimension

Operators: Standard

- Standard Multiplication **(*)** is either a dot product or an outer-product.

→ Inner dimensions MUST match.

- Standard exponentiation **(^)** can only be done on square matrices or scalars.
- Standard Division is NOT recommended, multiply by matrix inverse instead.

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$

$$1 \times 3 * 3 \times 1 = 1 \times 1$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 6 & 12 & 18 \\ 9 & 18 & 27 \end{bmatrix}$$

$$3 \times 3 * 3 \times 3 = 3 \times 3$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} ^\wedge 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Must be square to do powers

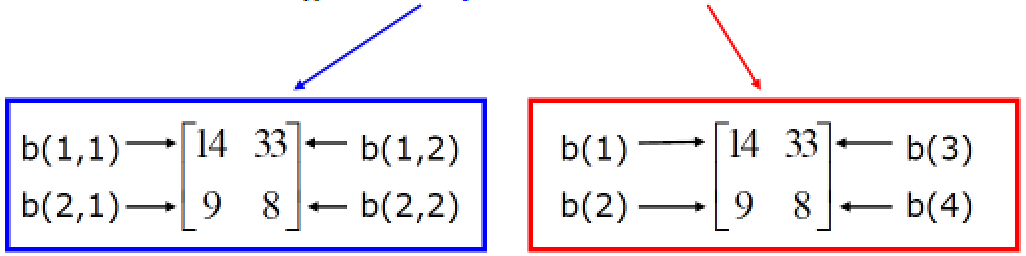
Matrix Indexing

- Matrices can be indexed in two ways:

→ Using **subscripts** (rows and columns)

→ Using **Linear Indices** (as if the matrix is a vector)

- Matrix Indexing: **Subscripts** or **Linear Indices**



- Picking Submatrices:

```
>> A = rand(5)
>> A(1:3,1:2)
>> A([1 5 3],[1 4])
```

Plotting

- Example:

```
>> x = linspace(0,4*pi,10);
>> y = sin(x);
```

- Plot values against their indexes:

```
>> plot(y)
```

- Usually we want to plot y versus x:

```
>> plot(x,y)
```

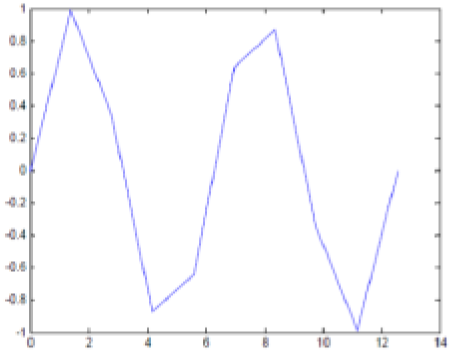
What does plot do?

- **plot** generates dots at each (x,y) pair and then connects the dots with lines.
- To make the plot of a function look smoother, evaluate at more points:

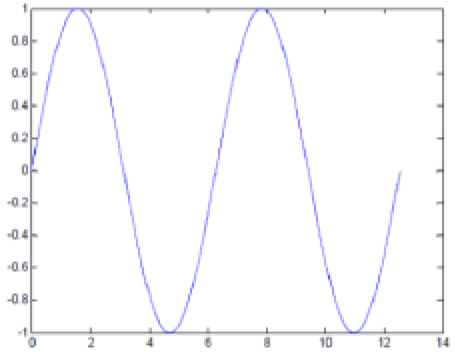
```
>> x = linspace(0,4*pi,1000);
>> y = sin(x);
>> plot(x,y)
```

- x and y vectors must be of the same size or else you will get an error.

10 x values:

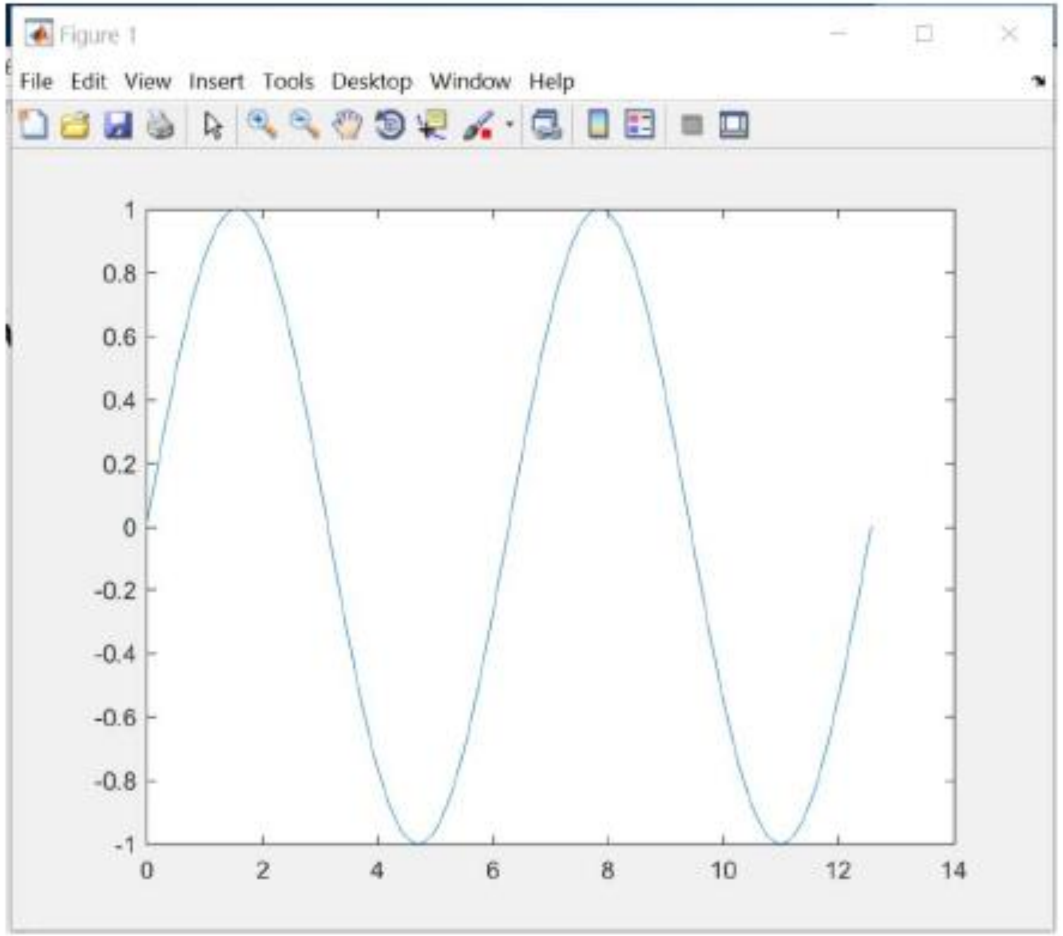


1000 x values:



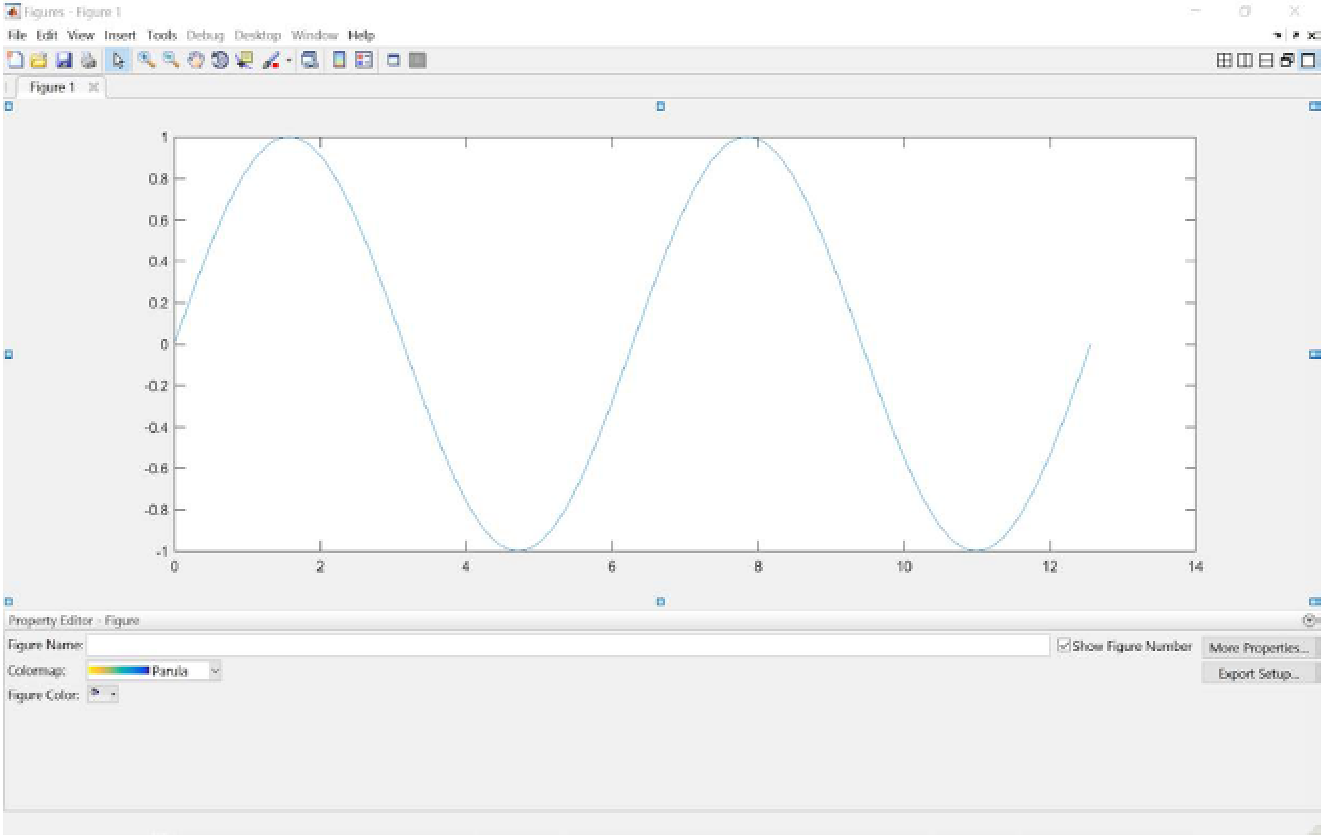
Formatting Plots

- To format the plot click **Edit → Figure Properties**



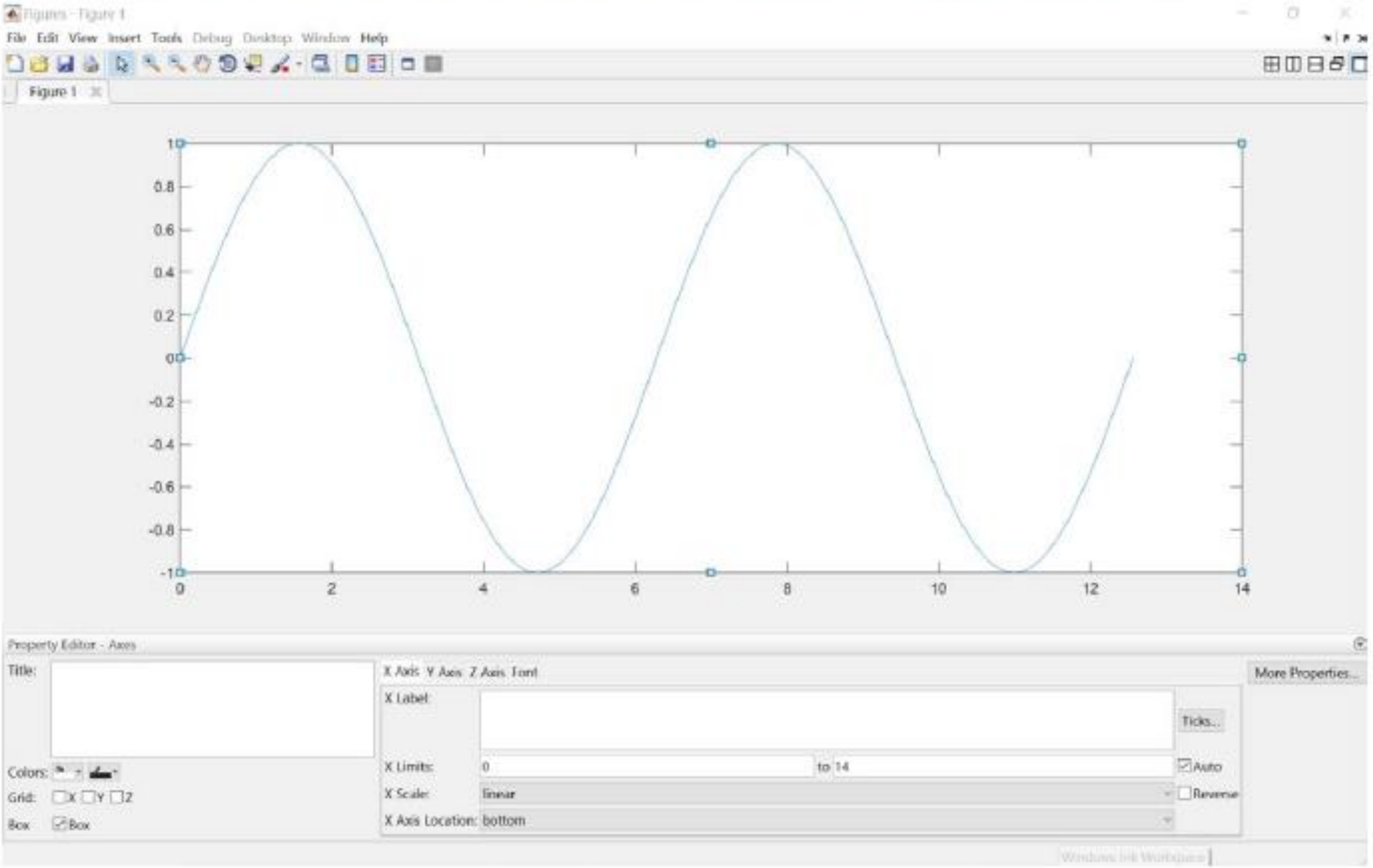
Formatting Plots

- You will get the following screen, you can change the Figure name.



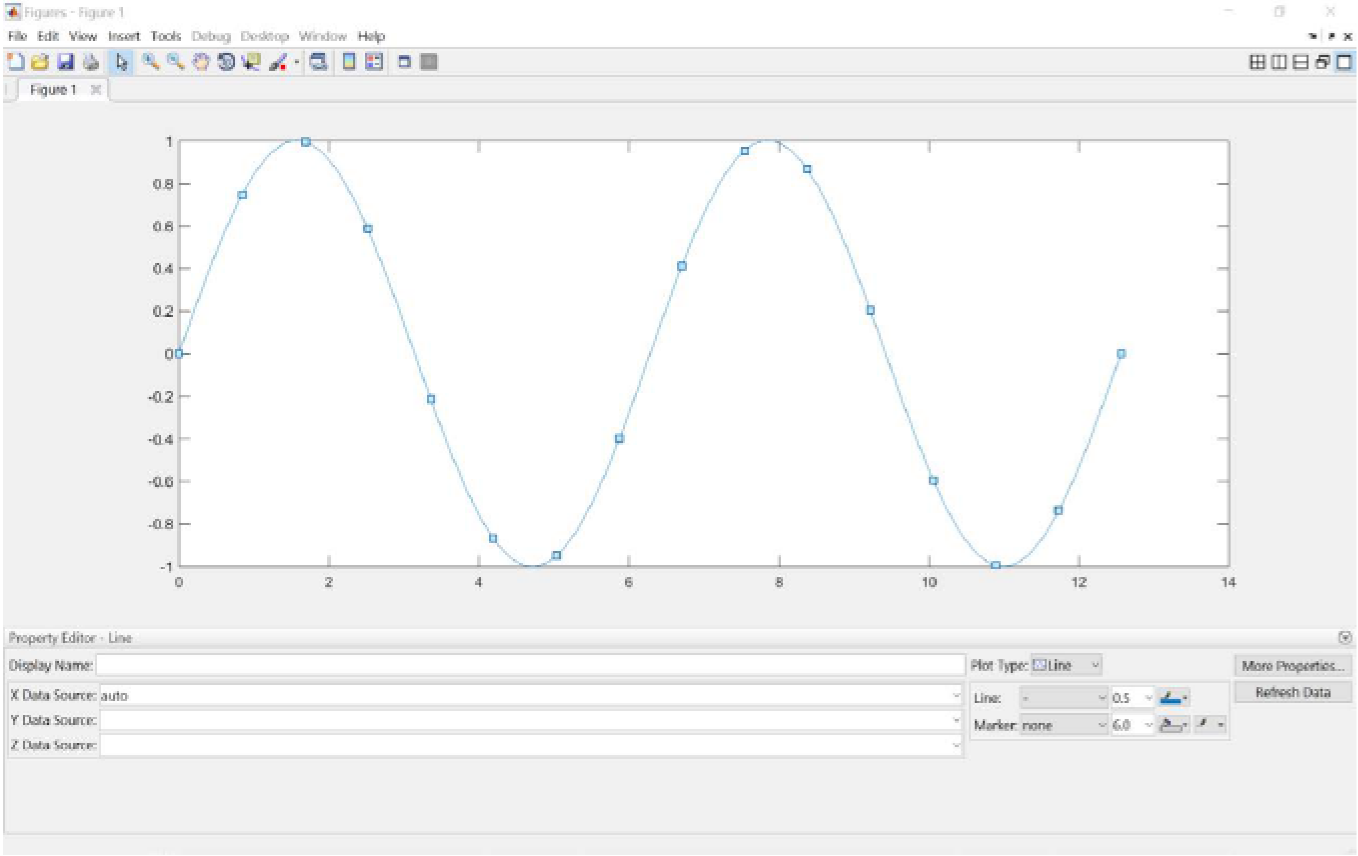
Formatting Plots

- Click on one of the axes to get the following screen, you can give the plot a title, label the axes, change the limits of the axes, change the fonts …



Formatting Plots

- Click on the line to get the following screen, you can change the line type, thickness and colour. Also, you can show data points.



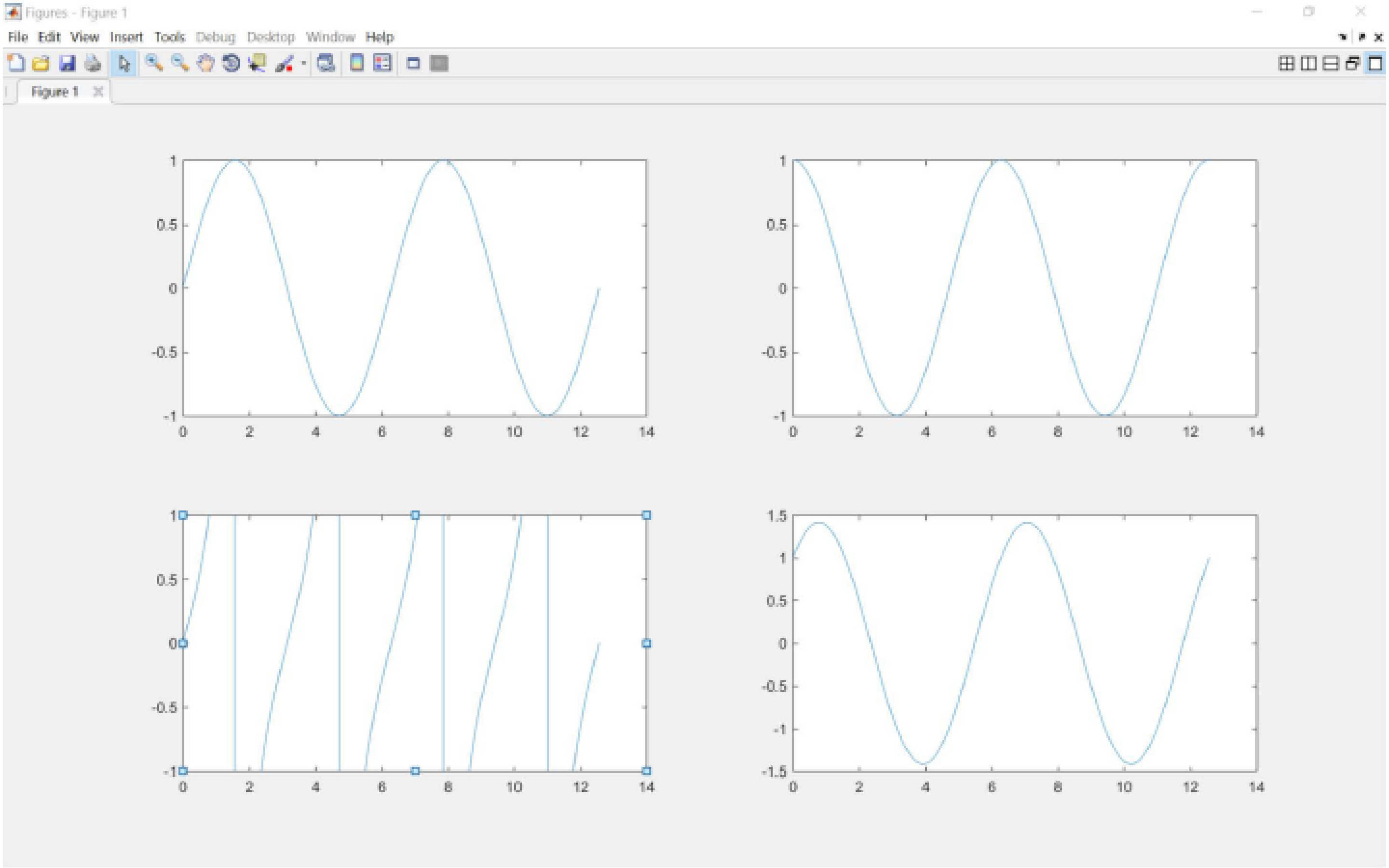
Multiple Plots in One Figure

- To have multiple plots in one figure:

```
>> x = linspace(0,4*pi,1000);
>> subplot(2,2,1)
```
→ Creates a figure with 2 rows and 2 columns of plots, and activates the first axis for plotting

```
>> plot(x,sin(x))
>> subplot(2,2,2)
>> plot(x,cos(x))
>> subplot(2,2,3)
>> plot(x,tan(x))
>> subplot(2,2,4)
>> plot(x,sin(x)+cos(x))
```

Multiple Plots in One Figure



Systems of Linear Equations

- Given a system of linear equations:

➢ x+2y-3z=5

➢ -3x-y+z=-8

➢ x-y+z=0

- Construct matrices so the system is described by **Ax=B**

```
>> A = [1 2 -3;-3 -1 1;1 -1 1];
>> B = [5;-8;0];
```

- Solve the system with a single line of code:

`>> x = A\B` Or inv(A) *b

Polynomials

- Many functions can be well described by a high-order polynomial.

- MATLAB represents a polynomial by a vector of coefficients

- Examples:
- `P = [1 0 -2]` represents the polynomial x^2-2
- `P = [2 0 0 0]` represents the polynomial $2x^3$

Polynomial Operations

- P is a vector of length N+1 describing an N-th order polynomial
- To get the roots of a polynomial :

```
r = roots(P)
```

- To get the polynomial from the roots:

```
P = poly(r)
```

- To evaluate a polynomial at a point:

```
y0 = polyval(P,x0)
```

- To evaluate a polynomial at many points:

```
y = polyval(P,x)
```

x and y are of the same size

Solve the equation $3x^2 - 2x - 4 = 0$.

```
p = [3 -2 -4];
r = roots(p)
```

```
r =

    1.5352
   -0.8685
```

Solve the equation $x^4 - 1 = 0$.

Solve the equation $x^4 - 1 = 0$.

```
p = [1 0 0 0 -1];
r = roots(p)
```

```
r =

  -1.0000 + 0.0000i
   0.0000 + 1.0000i
   0.0000 - 1.0000i
   1.0000 + 0.0000i
```

Command Window

```
>> r=[1,2]

r =

     1     2

>> p=poly(r)

p =

     1    -3     2

>> r=[0,5];
>> p=poly(r)

p =

     1    -5     0
```

Examples

The polynomial $p(x) = 3x^2 + 2x + 1$ is evaluated at $x = 5$, 7, and 9 with

```
p = [3 2 1];
polyval(p,[5 7 9])
```

which results in

```
ans =

    86   162   262
```

Polynomial Fitting

- MATLAB makes it very easy to fit polynomials to data.
- Given data vectors X = [-1 0 2] and Y = [0 -1 3]

```
P2 = polyfit(X,Y,2)
```

➢ Finds the best second order polynomial that fits the points (-1,0), (0,-1) and (2,3) poly.m*

```
clc
x=[-1 0 2];
y=[0 -1 3];
p2=polyfit(x,y,2);
plot(x,y,'o')
hold on
x=-3:0.1:3;
plot(x,polyval(p2,x),'r-');
```

Symbolic Math Toolbox

- Do not do nasty calculations by hand.

- Symbolics vs. Numerics

	Advantages	Disadvantages
Symbolic	• Analytical Solutions • Lets you understand things from the solution form	• Sometimes cannot be solved • Can be overly complicated
Numeric	• Always gives a solution • Can make solutions accurate • Easy to code	• Hard to extract a deeper understanding • Numerical methods sometimes fail • Can take a while to compute

Symbolic Variables

- Symbolic variables are a type, like **double** or **char**.
- To make symbolic variables, use `sym`:

```
>> a = sym('1/3');
>> b = sym('4/5');
```

➢ See `help sym` for a list of tags

- Or use `syms`

```
>> syms x y real
```

➢ Shorthand for `x = sym('x','real'); y = sym('y','real');`

Symbolic Expressions

- Multiply, add, subtract and divide expressions

```
>> d = a*b

d =

4/15

>> expand((a-c)^2)

ans =

c^2 - (2*c)/3 + 1/9

>> factor(ans)

ans =

(3*c - 1)^2/9
```