Matlab

Ciro Donalek

donalek@astro.caltech.edu

Summary

- Introduction to Matlab
- Programming in Matlab
  - data types, variables
  - loops vs Built-In Functions
- Matrix and Array Operations
  - Submatrices and Column Notation
- M-Files
  - script, functions
- Visualization
- Performance evaluation
- Advanced Use
  - Compiler, Database connection, How to build a GUI...
- Third Party toolboxes

What is Matlab?

- MATLAB is an interactive, **matrix**-based system for scientific and technical computing.

- It is matrix oriented, that means that all the matrix operations are highly optimized.
- It integrates computation, visualization and programming in an easy-to-use environment.

Typical Uses

Matlab is used in many fields for:

- Math and Computation
- Algorithm Development
- Data Acquisition
- Modeling, Simulation, and Prototyping
- Data Analysis, Exploration, and Visualization
- Scientific and Engineering Graphics
- Application Development, including Graphical User Interface building

Getting Matlab

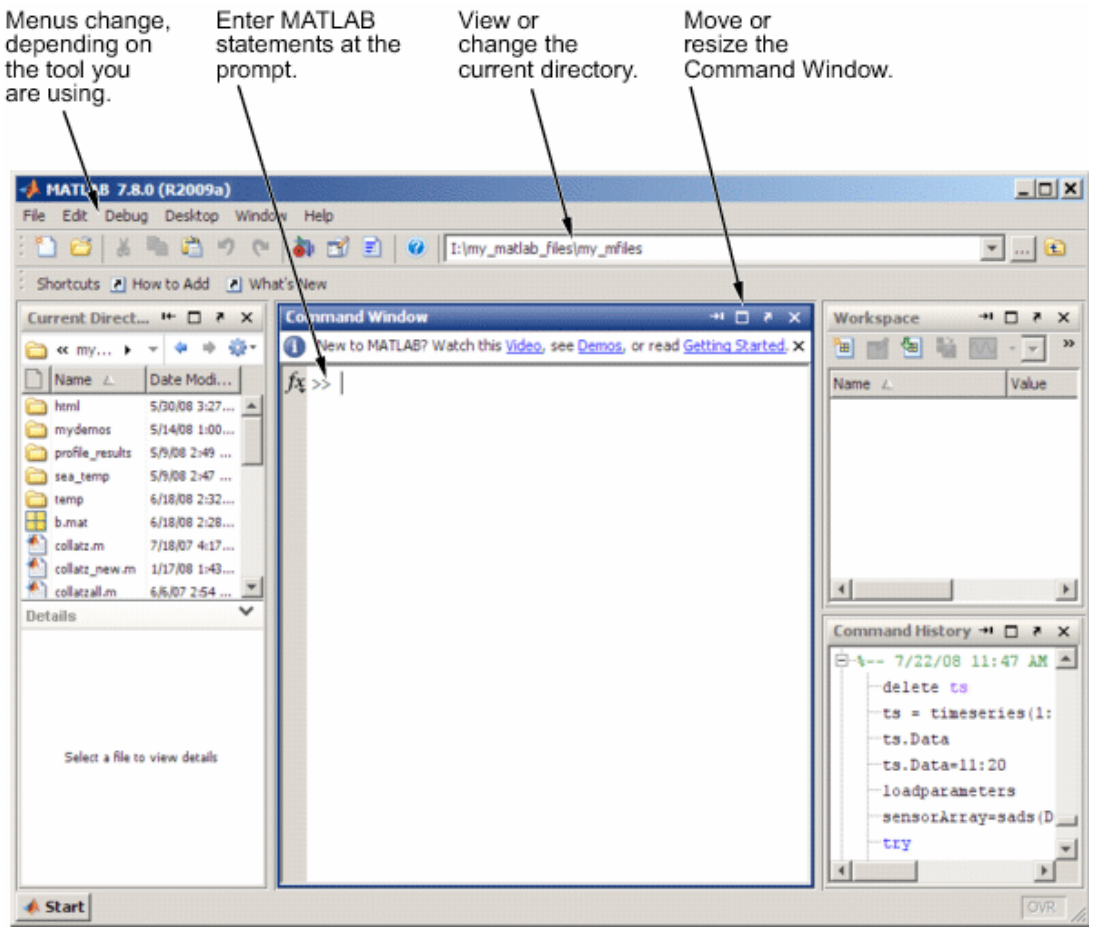
Proprietary software: distributed by MathWorks: http://www.mathworks.com
Latest version: 2009a (released March 6th, 2009)

Multiplatform: available for Linux, Mac, Solaris, Windows.

Support for 64 bits architectures.

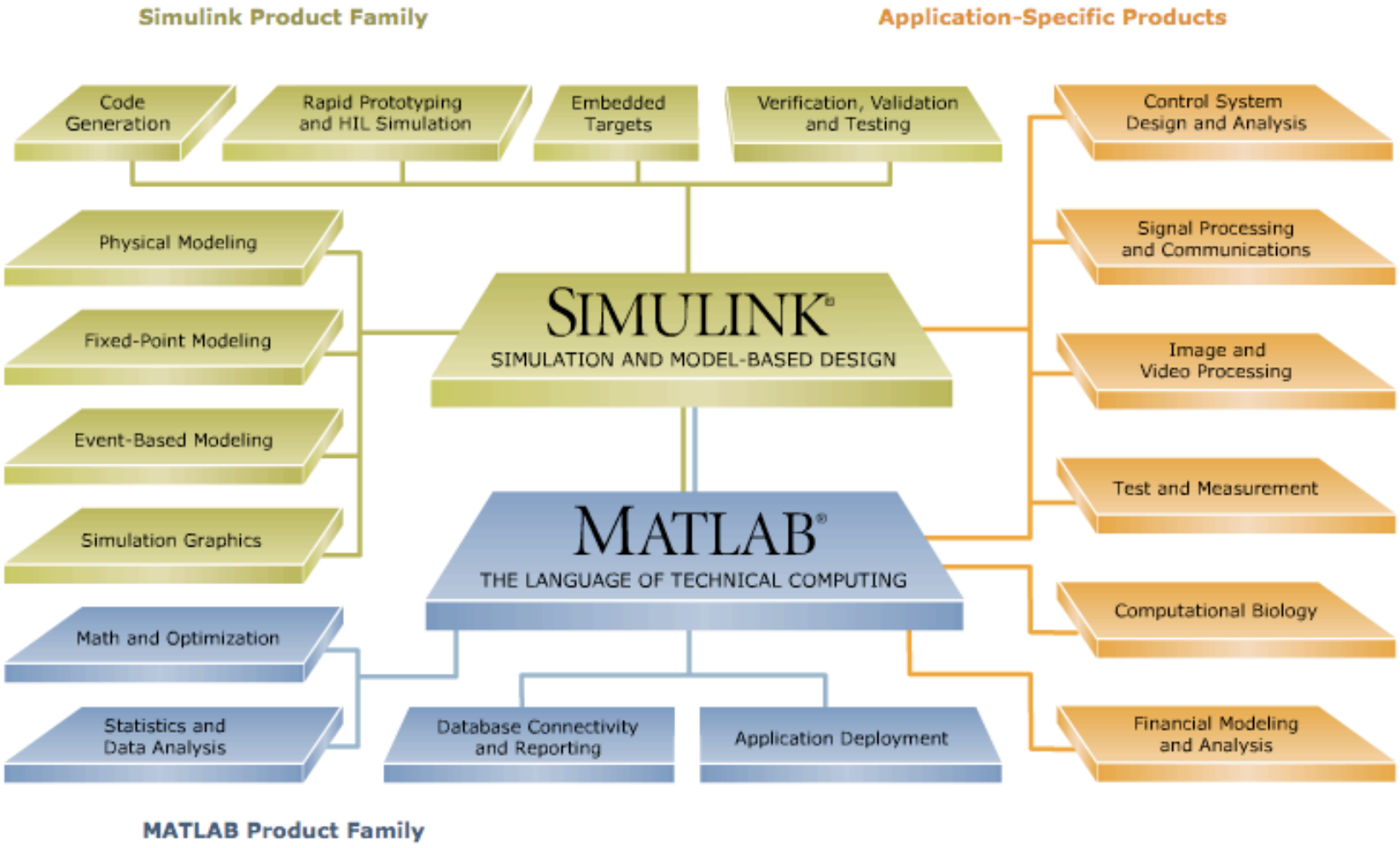
* Caltech personnel can download it for free at:

http://software.caltech.edu/



Matlab Family Products

- Available many toolboxes for specific applications.



Programming in Matlab

- The Language
  - The MATLAB language is a high-level matrix oriented language with:
    - control flow statements (loops, conditionals)
    - functions, data structures, input/output, and object-oriented programming features
    - it has high-level functions for two-dimensional and three-dimensional data visualization, image processing, animation...

- Mathematical Function Library
  - This library is a vast collection of computational algorithms ranging from elementary functions, like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, and fast Fourier transforms.

- It can be used for both small and large applications.

Variables

- MATLAB does not require any type declarations or dimension statements.
  - new variable name: automatically creates the variable and allocates the appropriate amount of storage

```
>> num=3; % creates a double "num" by default
```

  - if the variable already exists: MATLAB changes its contents (and allocates new storage if needed)

```
>> num=[1 3]; % num becomes an array
```

- Variable name (case sensitive)
  - any length but uses only the first N (*namelengthmax*)
  - letter, followed by any number of letters, digits, or underscores
  - type *iskeyword* to see the list of keywords in Matlab

Special Variables

- ans (*ans*wer)
  - when you do not specify an output variable, MATLAB uses the variable ans to store the results of a calculation

```
>> 2 + 3
>> ans =
    5
```

- Inf (*inf*inite)
- NaN
  - returns the IEEE arithmetic representation for Not-a-Number
  - marker for missing observation
- pi, eps, realmin, realmax, etc.

Numerical and Logical Types

- Numerical
  - integer, floating point, complex (`x = 2 + 3i;`)
  - short, long
  - all numbers are internally stored as long
  - `format`: change the way how the numbers are displayed

- Logical (boolean)
  - true: 1 , false: 0

- Char
  - integer value converted to its Unicode equivalent
  - a string is a vector of characters
  - the actual characters displayed depend on the character set encoding for a given font

All MATLAB data types are implemented as object-oriented classes.

Everything is a... Matrix

- In the MATLAB a matrix is a rectangular array of numbers.
  - scalars can be seen as 1-by-1 matrices;
  - array (vectors) are matrices with just one row or one column.

- MATLAB allows you to work with entire matrices quickly and easily.
- **It is usually best to think of everything as a matrix.**

Entering Matrices

Matrices can be introduced in several different ways:

- Entered by an explicit list of elements

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

- Generated by built-in statements and functions

```
>> A = rand(3)
```

- Created in M-files using built-in functions
- Loaded from external data files

Entering Matrices

Basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon ; to indicate the end of each row.
- Surround the entire list of elements with square brackets [ ].

```
>> A = [1,2,3; 4,5,6]
```

Matrix and Array Operations

- Matrix operations apply also to scalars (1-by-1 matrices).

- If the sizes of the matrices are incompatible for the matrix operation, an error message will result.
  For example, * is the usual matrix product, while .* is the element by element product.

+	addition
-	subtraction
*	multiplication
^	power
'	transpose
\	left division
/	right division

The "matrix division". If A is an invertible square matrix and b is a compatible column:

```
>> x = A \ b % left division, solution of A*x=b
>> x = b / A % right division, solution of x*A=b
```

Matrix Building Functions

- Used to build matrices with specific properties

eye	identity matrix
zeros	matrix of zeros
ones	matrix of ones
diag	see below
triu	upper triangular part of a matrix
tril	lower triangular part of a matrix
rand	randomly generated matrix
hilb	Hilbert matrix
magic	magic square
toeplitz	see `help toeplitz`

```
>> MB1 = zeros (3) % build a 3x3 matrix with all 0
>> MB2 = [ones(3), zeros(3,2); zeros(2,3), eye(2)] % 5x5
matrix - concatenation
```

Matrix and Vectors functions

eig	eigenvalues and eigenvectors
chol	cholesky factorization
svd	singular value decomposition
inv	inverse
lu	LU factorization
qr	QR factorization
hess	hessenberg form
schur	schur decomposition
rref	reduced row echelon form
expm	matrix exponential
sqrtm	matrix square root
poly	characteristic polynomial
det	determinant
size	size
norm	1-norm, 2-norm, F-norm, <infinity>-norm*
cond	condition number in the 2-norm
rank	rank

max	sum	median	any
min	prod	mean	all
sort		std	

```
% find the maximum value in a
% matrix
>> A=[7 2 3;
      4 5 2];
>> max_vec = max(A)
   [7 5 3]
>> max2 = max(max_vec)
   7
>> max2 = max(max(A))
   7

% when an array operation is
% applied to a matrix, it is
% applied to each column,
% the result is an array!
```

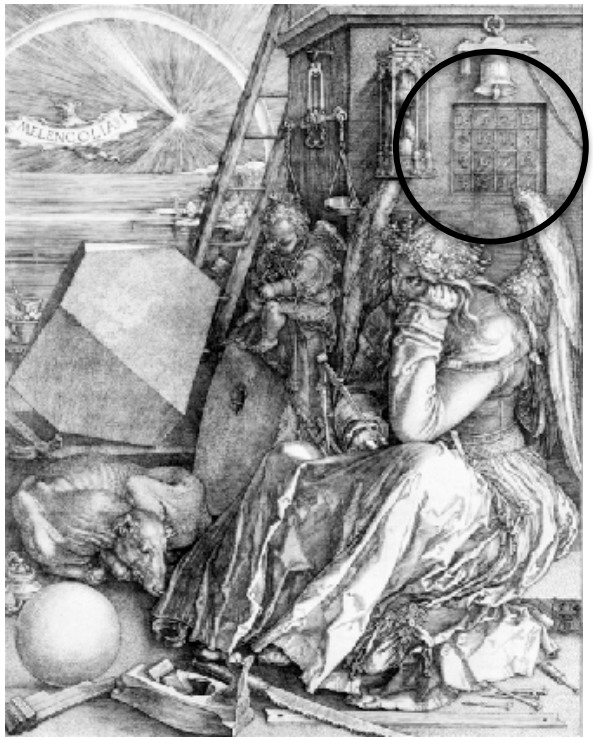
Submatrices and Column notation

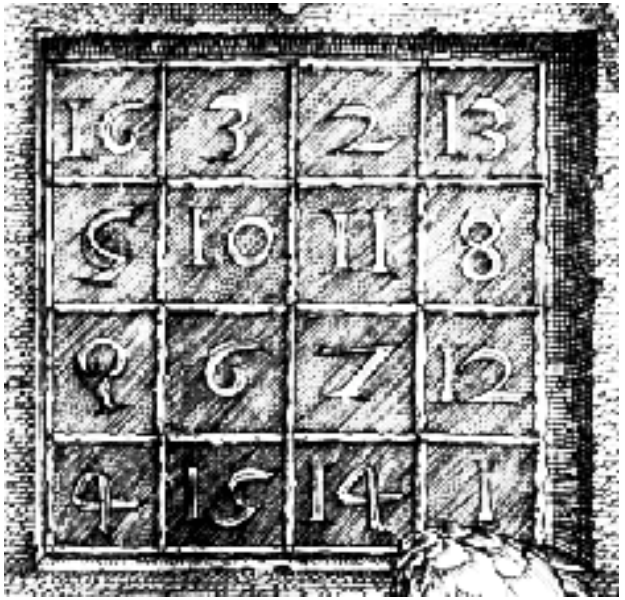
- Powerful way to access data stored in matrices.
- The element in row *i* and column *j* of A is denoted by A(*i,j*).
- The colon operator **:** is one of the most important MATLAB operators.
- These features permit to minimize the use of loops (which slows MATLAB) and to make code simple.

```
>> a=[2:2:10]; % create an array
>> A(1:3,3) % access part of a matrix
>> A([2 4],:) % access 2nd and 4th row, all the columns
>> A(:,2:end) % all rows, column 2 to the last
>> A(:,[2 4 5])=B(:,1:3)
```

Example: Magic Matrix

- Known for over 4000 years (found in ancient Egypt and India) was seen for the first time in European art in the renaissance engraving Melencolia I by the German artist and amateur mathematician Albrecht Dürer.
- Known as a magic square, was believed having magical properties. It does turn out to have some fascinating characteristics worth exploring.





Other Data Structures

Matrices can be used to store homogenous data, other useful structures are:

- Multidimensional Arrays
- Structures
- Cell Arrays
- Arrays and Structures
- Function Handles

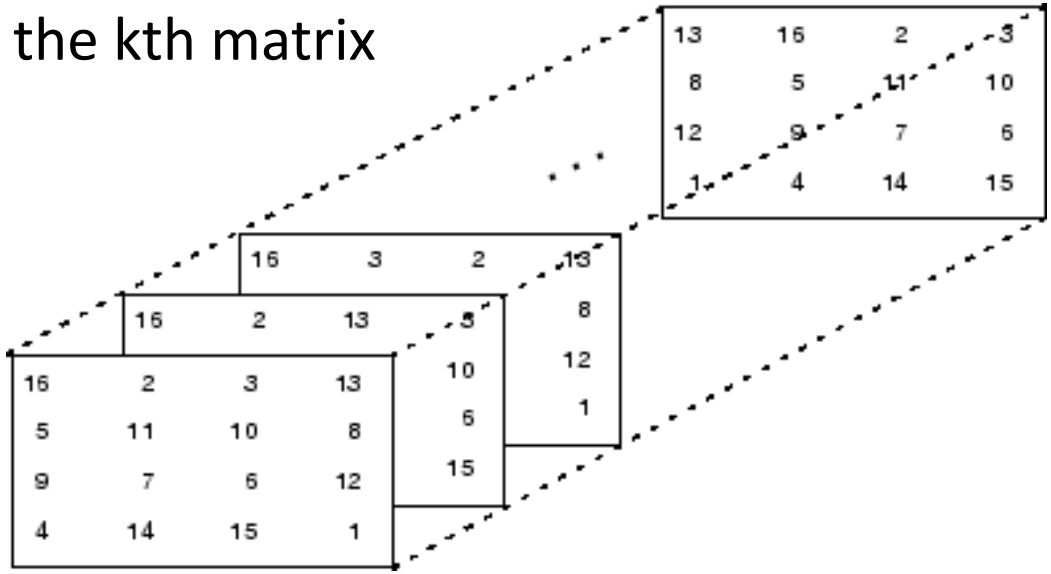
Multidimensional Arrays

- Multidimensional Arrays
  - arrays with more than two subscripts;

    >> B=rand(2,3,2);
  - used to represent a sequence of matrices, A(*k*), or samples of a time-dependent matrix, A(*t*)
    - the (i, j)th element of the kth matrix

      is denoted by A(*i,j,k*)

If we have a magic matrix that change over time, we can access the values at a certain time k using a A(i,j,k)



Structures

Structures and cell arrays, provide a way to store dissimilar types of data in the same array.

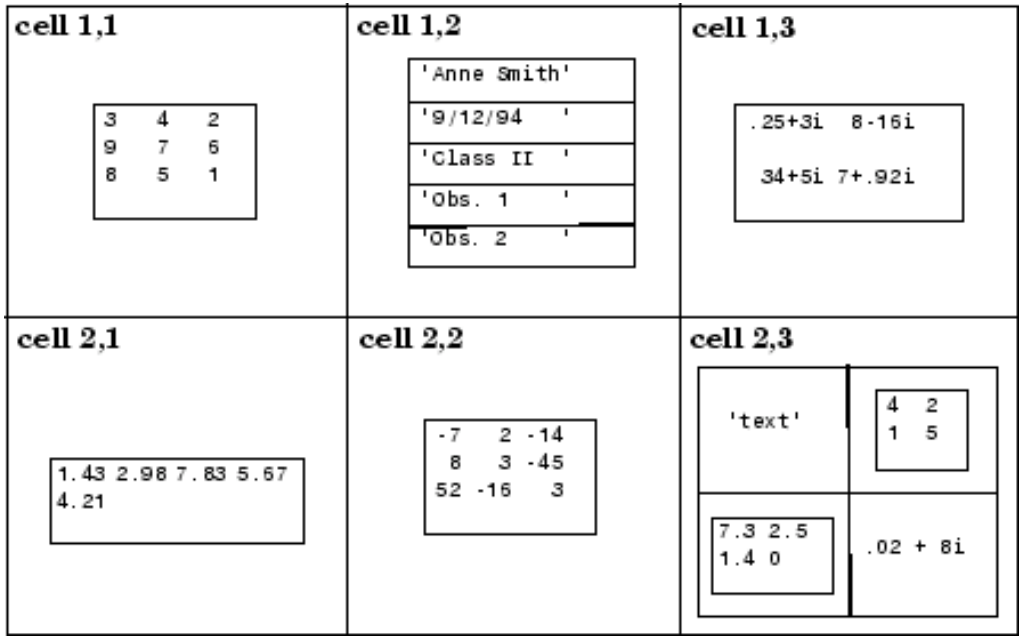
- Structures
  - multidimensional MATLAB arrays with elements accessed by textual field designators

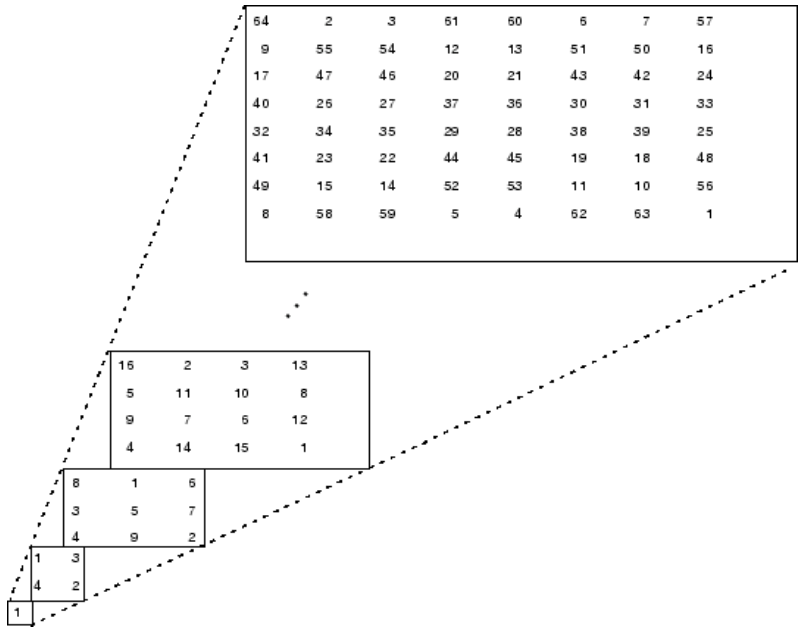
```
>> my=[];
>> my.age=35;
>> my.country='Italy';
my =
      age: 35
      country: 'Italy'
```

Cell Arrays

- Cell Array
  - a cell array is a collection of containers called cells in which you can store different types of data

```
>> my={35, 'Italy', [7 9 11 13 15]}
my =
    [35]    'Italy'    [1x5 double]
```





Arrays and Structures

- Structures of Arrays vs. Arrays of Structures

```
% Structures of Arrays
a.x = 1:10;
a.y = sin(a.x);
```

Each field is an array.
Good way to keep data that are related together.
Easy to pass in one structure when using functions.

```
% Arrays of Structures
people(1).name ='Ciro';
people(1).age = 35;
people(1).country='Italy'
people(2).name = 'Matusalem';
people(2).alt_name='Methuselah'
people(2).age = 969;
people(2).country='Unknown';
```

Each element of the array is a structure.
Useful when you have multiple copies of the same dataset: you can access easily a particular record.

Function Handle

- A function handle is a callable association to a MATLAB function:

  h = @functionname

- With function handles, you can:
  - pass a function to another function
  - call functions outside of their normal scope
  - save the handle in a MAT-file to be used in a later MATLAB session
  - use anonymous functions

Function Handles: Examples

- associate a function handler to a function

```
my_f = @stat
```

- anonymous functions

```
my_sqr = @(x) x.^2;
```

- array of function handles

```
trigFun = {@sin, @cos, @tan}; % cell array
plot(trigFun{2}(-pi:0.01:pi))
```

Flow Control

Matlab has loops and conditionals: for, while, if-else(if), continue, break, try, catch.

When applied to scalars, a relation is actually 1 or 0 depending on whether the relation is true or false.

When applied to matrices of the same size, a relation is a matrix of 0's and 1's depending on the value of the relation between corresponding entries.

<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
==	equal
~=	not equal.

&	and
|	or
~	not.

```
>> a=[2:2:10]
   2  4  6  8  10
>> a>6
   0  0  0  1  1
```

Loops and Conditionals

- For loop: the syntax is highlighted in the boxes.

```
x = []; n=5;
for i = 1:n
  x=[x,i^2];
end
```

```
x = []; n=5;
for i = 1:n
  x(i)=i^2;
end
```

```
x = []; n=5;
for i = n:-1:1
  x(i)=i^2;
end
```

- If... else (elseif)...

```
if RELATION
  statement
end
```

```
if a==b
   statement
end
```

```
if (isequal(A,B))
   statement
end
```

```
if A~=B
   statement
end
```

```
if any(any(A~=B))
   statement
end
```

are they different?
why 2 “any” in the second one?

Avoid and Speed Up Loops

- Built-In Functions
- Vectorization
- Pre-Allocation

Loops vs. Built-in Functions

- When programming with Matlab you should always try to avoid loops and conditionals and use built-in functions.
- Loops slow down Matlab while Built-in functions are highly optimized
- Eg. “for” vs “find”

```
random=rand(1,100000); % goal: create a new array with only elements > 0.5
```

```
 % solution 1: for loop
real1=[];
j=1;
n=size(random,2)
for i=1:n
   if (random(i)>0.5)
      real1(j)=random(i);
      j=j+1;
   end
end
```

```
% solution 2: built-in function (find)
real2=[];
real2=random(find(random(:)>0.5));
```

MUCH FASTER!

Vectorization

```
% Vectorizing a double FOR loop that creates a
% matrix by computation.
% Replace with element by element matrix operations
A = rand(1000);
B = rand(1000);

tic
for j = 1:1000
   for k = 1:1000;
     X1(j,k) = sqrt(A(j,k)) * B(j,k);
   end
end
sum1=sum(sum(X1))
toc

%using vectorization -> MUCH FASTER!!!
tic
X2 = sqrt(A).* B;
sum2=sum(sum(X2))
toc
```

Preallocation

- If you cannot vectorize a piece of code, you can make your for loops go faster by preallocating any arrays in which output results are stored.
- Without the preallocation the MATLAB interpreter enlarges the output vector by one element each time through the loop.
  - Vector preallocation eliminates this step and results in faster execution.

```
% using preallocation
A = rand(1000); B = rand(1000);
tic
X3=zeros(1000,1000);
for j = 1:1000
    for k = 1:1000;
      X3(j,k) = sqrt(A(j,k)) * B(j,k);
    end
end
sum3=sum(sum(X3))
toc
```

Logical Subscripting

- Another useful notation is the logical subscripting.
- The logical vectors created from logical and relational operations can be used to reference subarrays.

```
% extract a subarray from x with no NANs
x = [2.1 1.7 1.6 1.5 NaN 1.9];
y= x(isfinite(x))
y =
    [2.1 1.7 1.6 1.5 1.9]
```

M-files

- When you write a MATLAB function or script, you save it to a file called an M-file (named after its .m file extension).
- There are two types of M-files: script files and function files.
- M-Files can be executed calling their filename.
- Don't forget to add the path to your codes to tell Matlab where they are!
- Always use comments (%) and give meaningful (not only for you!) names to functions and variables.

M-files: script

- A script file consists of a sequence of MATLAB statements.
- If the filename is *myfile.m* the MATLAB command *myfile* will cause the statements in the file to be executed.
- **Variables in a script file are global** and will change the value of variables of the same name in the environment of the current MATLAB session.
- An M-file can reference other M-files, including referencing itself recursively.

M-files: Functions

- Function files actually provide a way to add new functionality to Matlab.
- You can create new functions specific to your problem which will then have the same status as other MATLAB functions.
- **Variables in a function file are by default local**; it is possible to declare variables as global.
- When the function ends the local variables are not available anymore.

Be organized...

- Save your Work
  - save
  - diary
    - *DIARY FILENAME* causes a copy of all subsequent command window input and most of the resulting command window output to be appended to it
- Write Documentation
  - help
- Clean your Space
  - clc
  - clear all , clear *name_var*
  - close all

"Everything seems to be in order, darn it!"

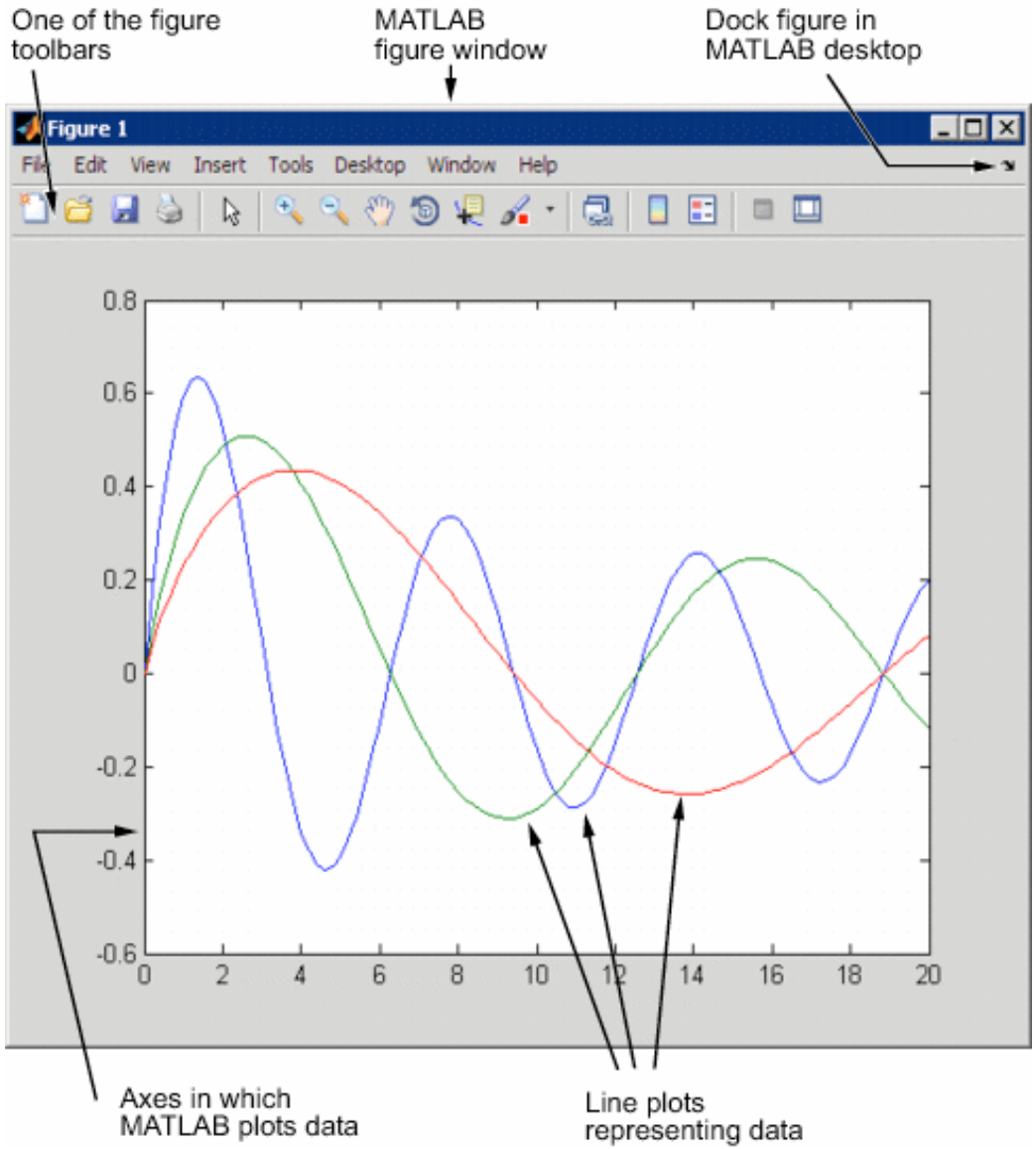
Figures and Plots

- Matlab gives a very easy way to generate and annotate many kinds of 2-D and 3-D plots.

- A figure is a MATLAB window that contains graphic displays (usually data plots) and UI components.
  - by default, figure windows are resizable and include pull-down menus and toolbars
  - you create figures explicitly with the figure function, and implicitly whenever you plot graphics and no figure is active

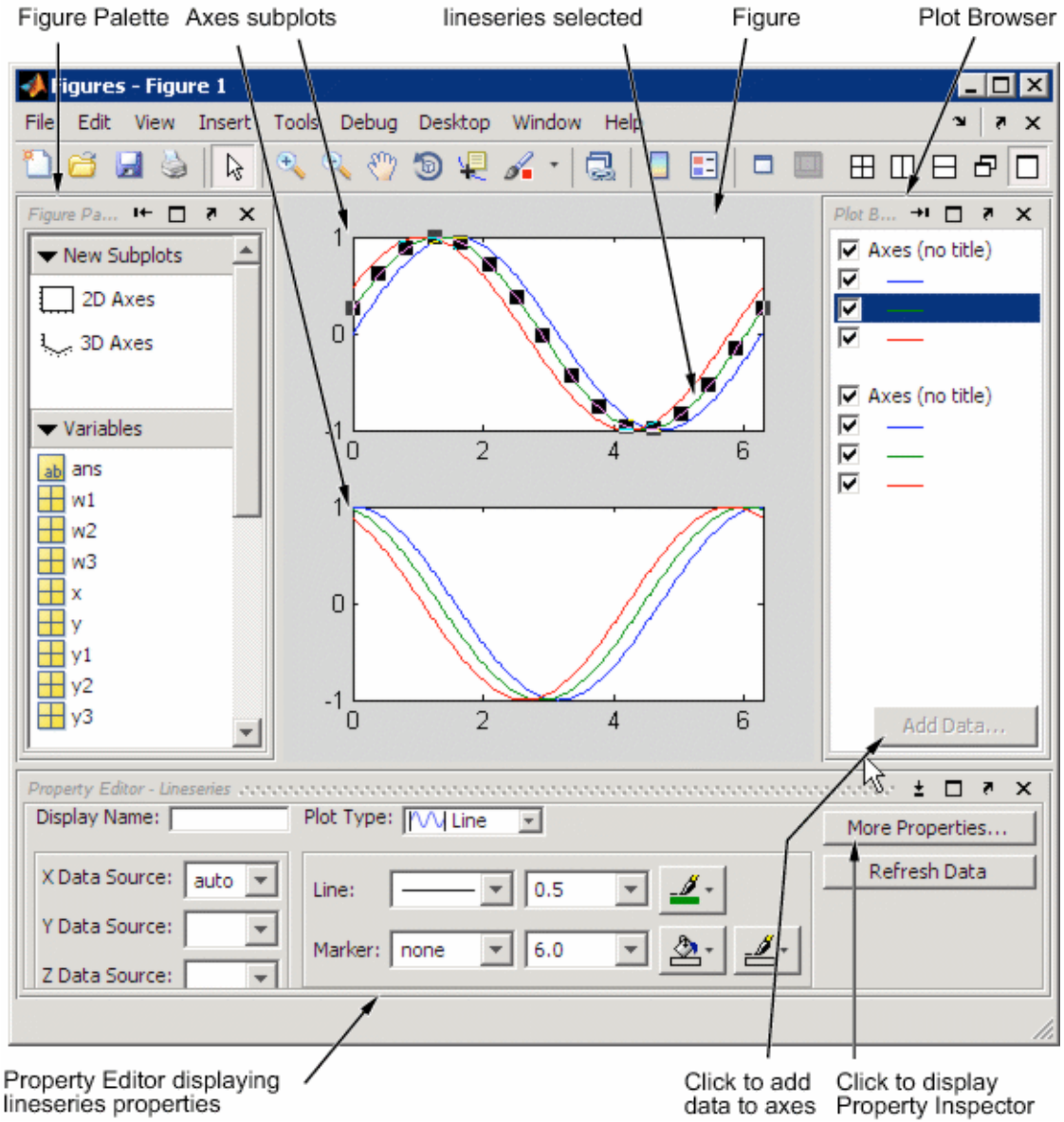
- A plot is any graphic display you can create within a figure window
  - each plot is created within a 2-D or a 3-D data space called an axes.
  - can display tabular data, geometric objects, surface and image objects, and annotations such as titles, legends, and colorbars.
  - figures can contain any number of plots.

Anatomy of a Graph




```
x=[0:.2:20];
y=sin(x)./sqrt(x+1);
y(2,:)=sin(x/2)./sqrt(x+1);
y(3,:)=sin(x/3)./sqrt(x+1);
plot(x,y)
legend('s1', 's2', 's3');
```

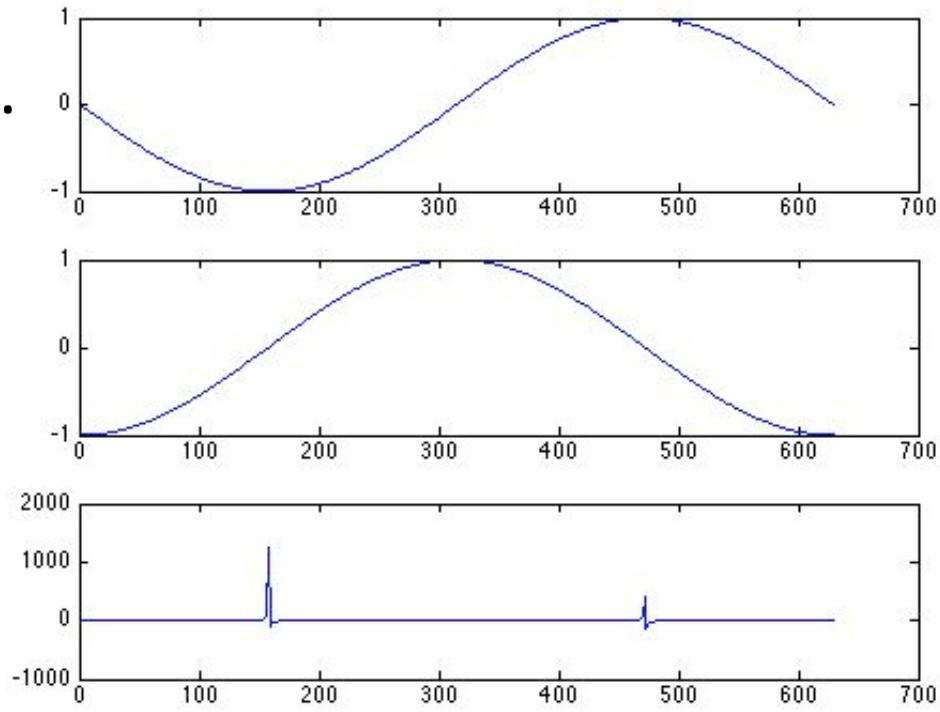
Interactive Plotting



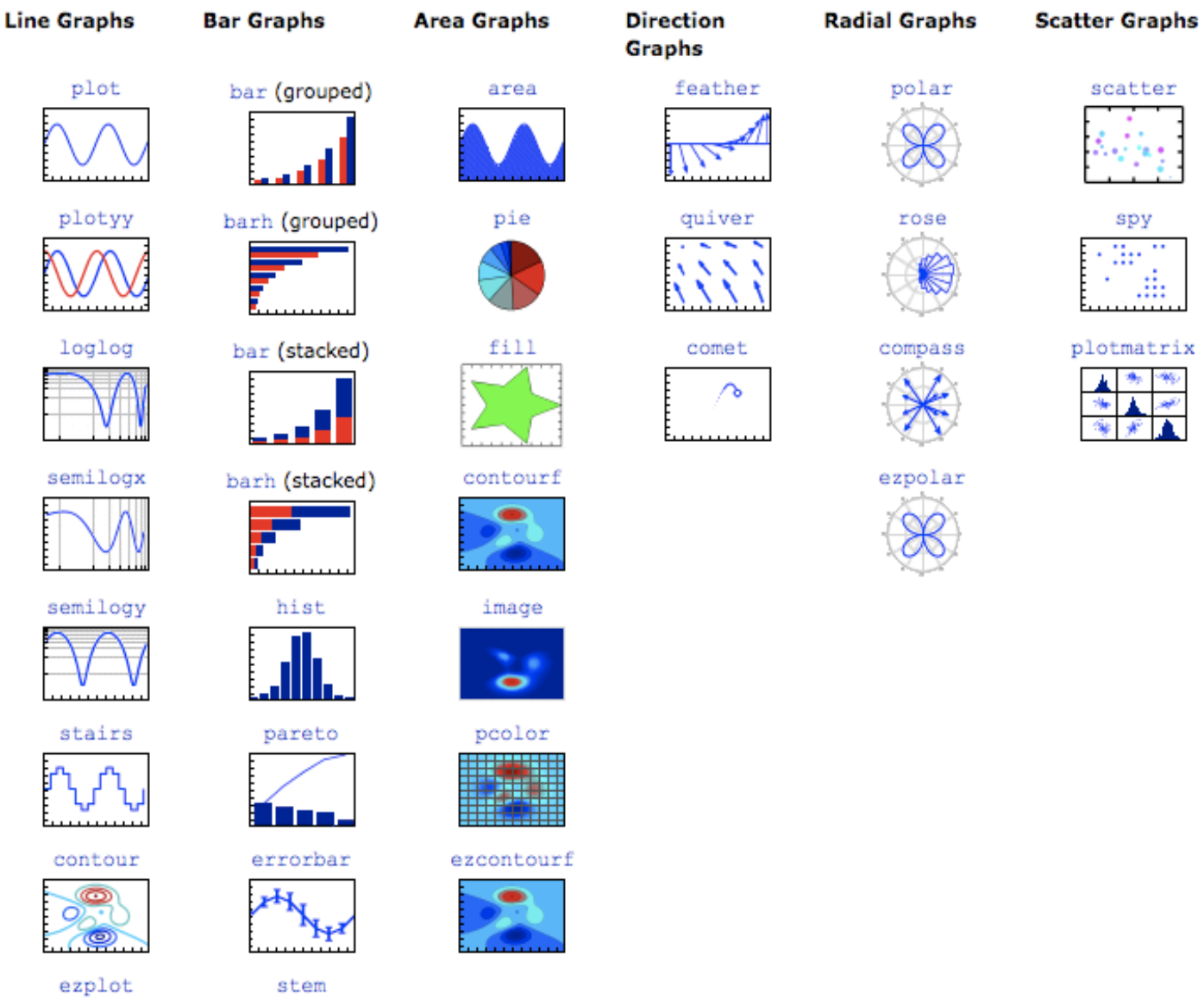
Plot Properties can be set:
- via script, arguments
- using the interactive plot

Subplots

- SUBPLOT Create axes in tiled positions.
  - H = SUBPLOT(m,n,p) breaks the Figure Window into an m-by-n matrix of small axes, selects the p-th axes for the current plot, and returns the axis handle.



2D-Plots


Line Graphs
Bar Graphs
Area Graphs
Direction Graphs
Radial Graphs
Scatter Graphs
plot
plotyy
loglog
semilogx
semilogy
stairs
contour
ezplot
bar (grouped)
barh (grouped)
bar (stacked)
barh (stacked)
hist
pareto
errorbar
stem
area
pie
fill
contourf
image
pcolor
ezcontourf
feather
quiver
comet
polar
rose
compass
ezpolar
scatter
spy
plotmatrix

Representing a matrix as a surface

The meshgrid function transforms the domain specified by two vectors, x and y, into matrices X and Y.

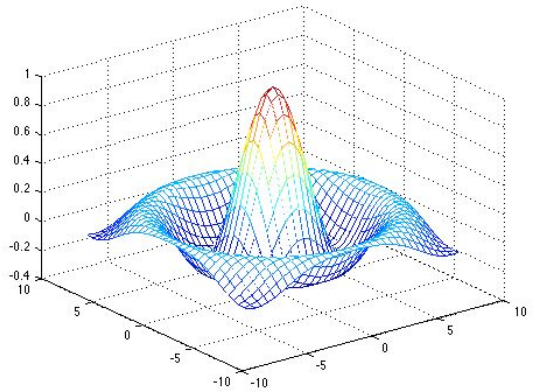
The rows of X are copies of the vector x and the columns of Y are copies of the vector y.

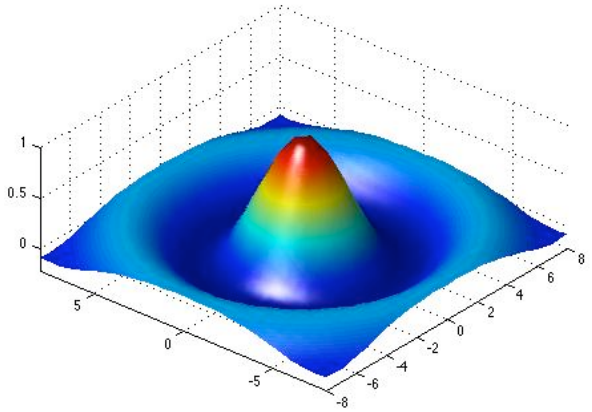
These matrices can be used to evaluate functions of two variables:

```
% consider the sin(r)/r
[X,Y] = meshgrid(-8:.5:8);

% The matrix R contains the distance
% from the center of the matrix
% Adding eps prevents the divide by zero
R = sqrt(X.^2 + Y.^2) + eps;
% values of the sinc function
Z = sin(R)./R;
mesh(X,Y,Z)

% lights and view adjustment
figure;
surf(X,Y,Z,'FaceColor','interp',...
    'EdgeColor','none',...
    'FaceLighting','phong')
daspect([5 5 1])
axis tight
view(-50,30)
camlight left
```





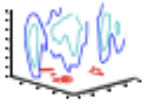
3D-Plots

Line Graphs

plot3

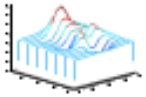
contour3

contourslice



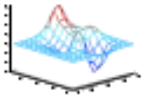
ezplot3

waterfall

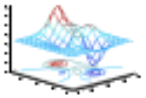


Mesh Graphs and Bar Graphs

mesh

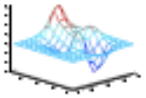


meshc

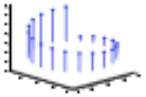


meshz

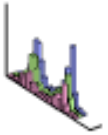
ezmesh



stem3



bar3



Area Graphs and Constructive Objects

pie3

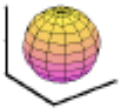
fill3

patch

cylinder

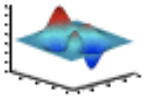
ellipsoid

sphere

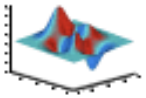


Surface Graphs

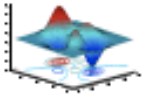
surf



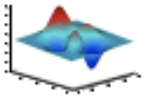
surfl



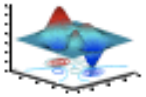
surfc



ezsurf

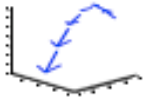


ezsurfc



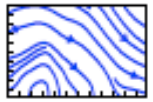
Direction Graphs

quiver3



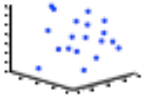
comet3

streamslice

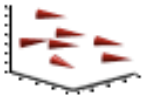


Volumetric Graphs

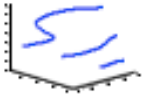
scatter3



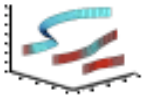
coneplot



streamline



streamribbon



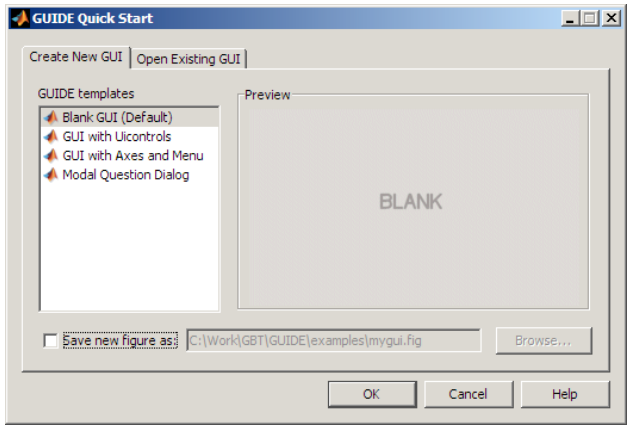
streamtube

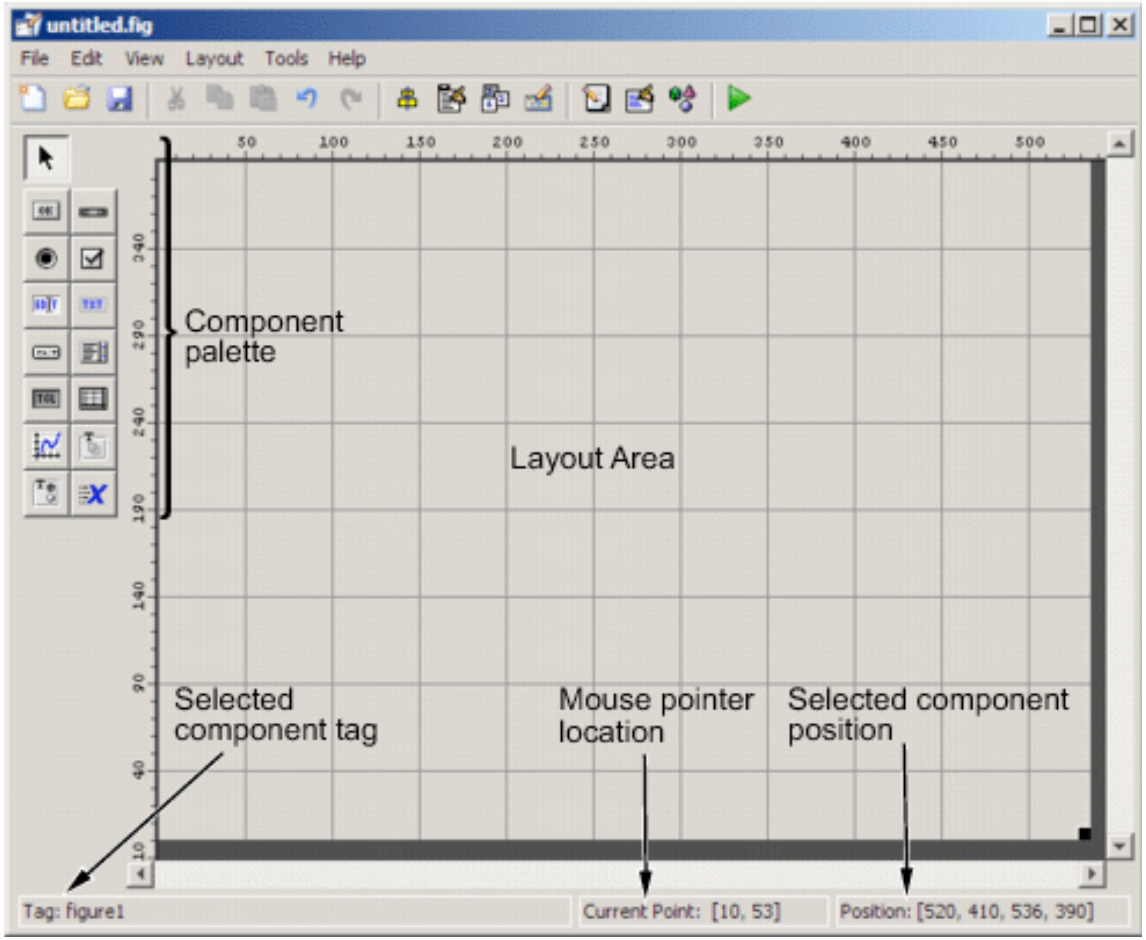
Graphical User Interface

- GUIDE: Graphical User Interface Development Environment
  - provides a set of tools for creating GUIs
  - simplify the process to design and build GUI
  - has many components: panels, buttons, text fields, sliders, menus... (drag and drop)
  - automatically generates an M-file that controls how the GUI operates.
    - The M-file initializes the GUI and contains a framework for the most commonly used callbacks for each component
      - command executed when a user clicks a GUI component

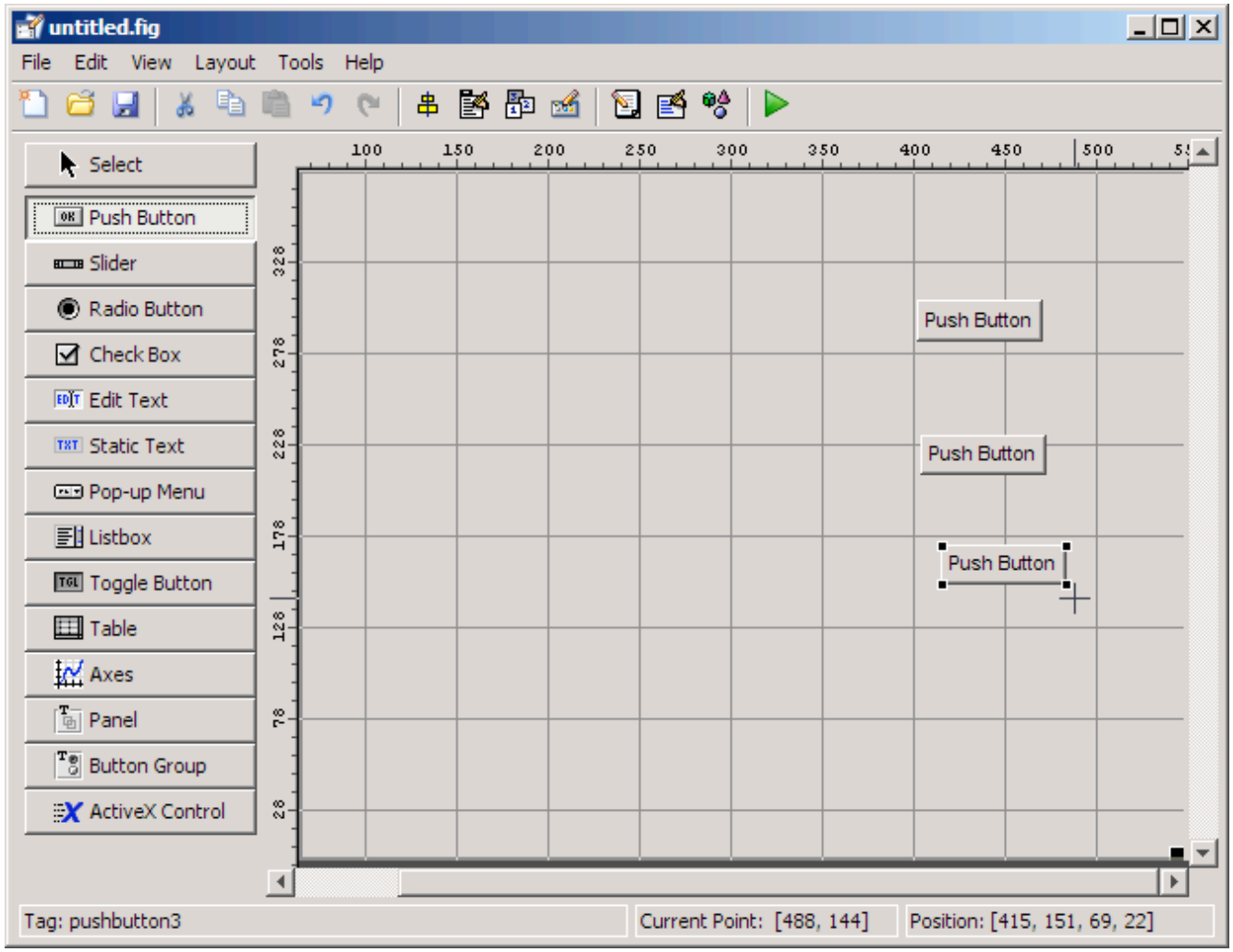
Start GUIDE

- To start GUIDE: *guide*



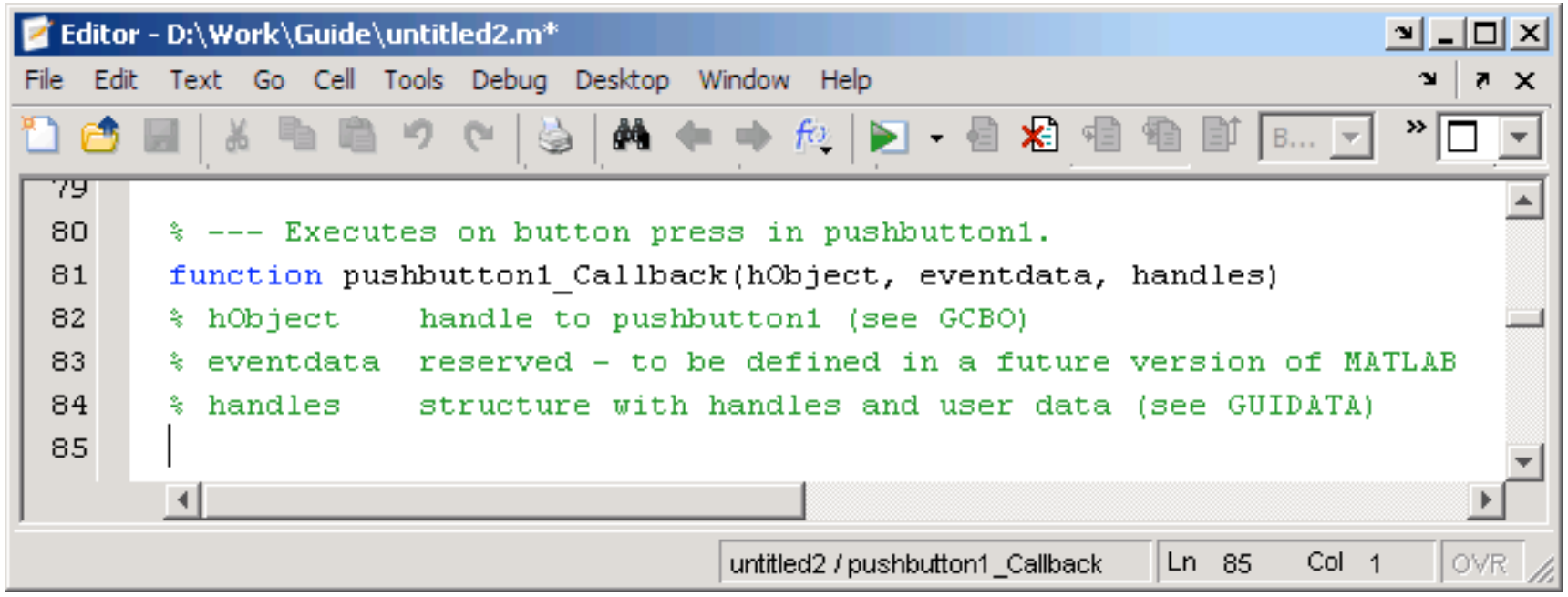


Adding Elements

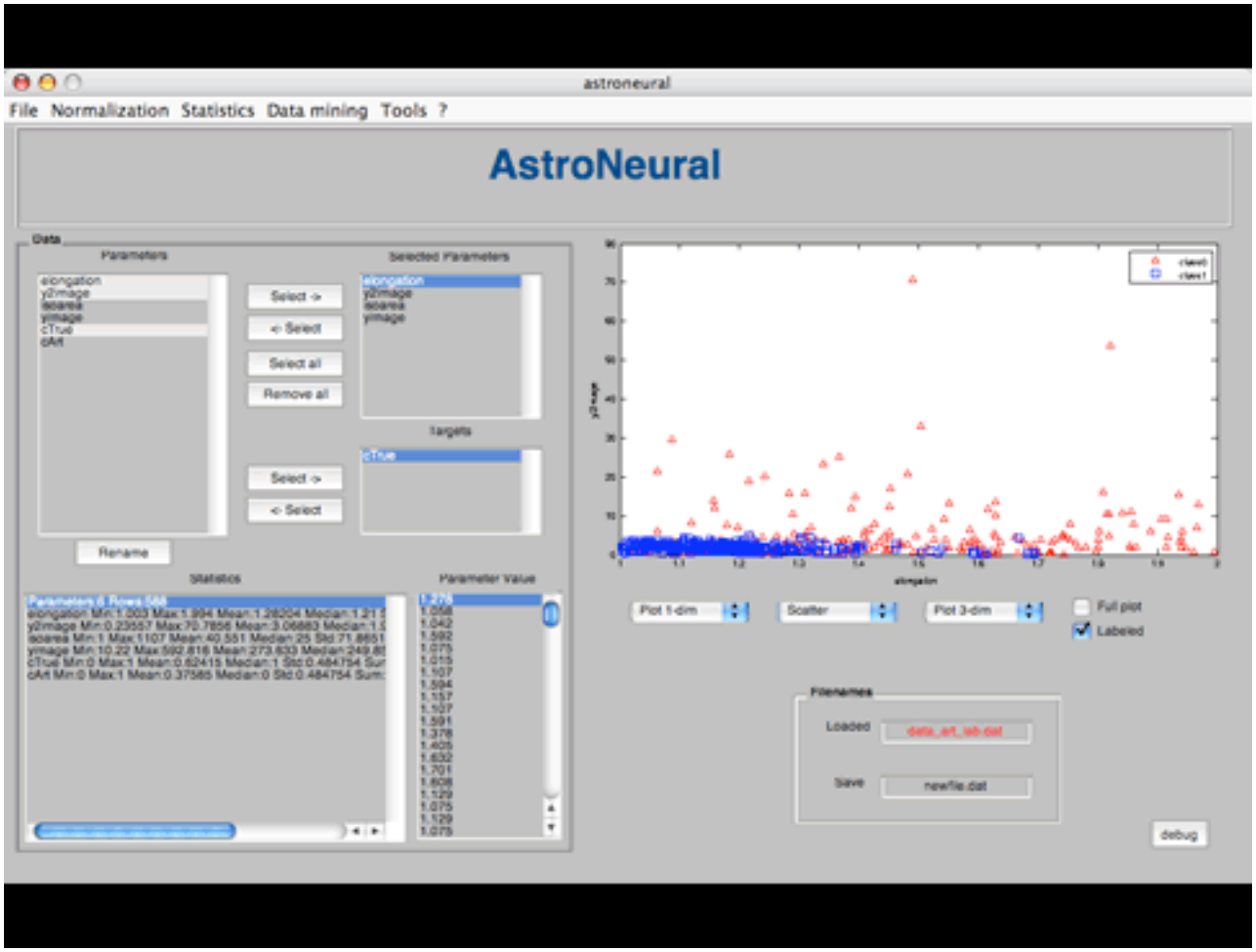


Programming a GUI

- After laying out the GUI and setting component properties, the next step is to program the GUI.
- You program the GUI by coding one or more callbacks for each of its components.



A more advanced GUI

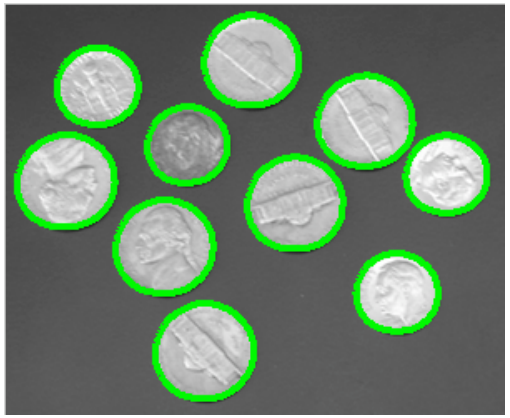


Coin Count

```
% requires the Image Processing Toolbox
I = imread('coins.png'); % read an image
imshow(I)                % show the image
BW = im2bw(I);           % convert the image in B/W
figure, imshow(BW)                % show the B/W image
dim = size(BW)           % size of the image in pixels
col = round(dim(2)/2)-90;  % determine the starting location
row = min(find(BW(:,col))) % for the boundary tracing
boundary = bwtraceboundary(BW,[row, col],'N');
```

```
% bwboundaries: by default finds the boundaries of all objects in an image
% including objects inside other objects.
% In the binary image used in this example, some of the coins contain
% black areas that bwboundaries interprets as separate objects.
% To ensure that bwboundaries only traces the coins,
% use imfill to fill the area inside each coin.

BW_filled = imfill(BW,'holes');
boundaries = bwboundaries(BW_filled);
fprintf('Number of coins: %d \n', size(boundaries,1));
```



Performances: evaluation

- A good first step to speeding up your programs is to find out where the bottlenecks are.

- Stopwatch Timer Function
  - to get an idea on how long the program takes to run
  - tic, toc
    - `>> tic, rand(100000,1), toc`
    - `>> tic, rand(100000,1);, toc`
  - measuring "small" programs
    - run the program repeatedly in a loop
    - average to find the time for a single run

Profiler and Code Analyzer

- The M-File Profiler Utility
  - graphical user interface that shows you where your program is spending its time during execution.
  - help you to determine where you can modify your code to make performance improvements
  - to start the Profiler, you can type
    - `>> profile viewer`
    - or select Desktop > Profiler in the MATLAB Command

- The M-Lint Code Analyzer
  - checks your code for problems and recommends modifications to maximize performance and maintainability
    - `>> mlint myfile`

Improving Performances

- Techniques to improve performances
  - vectorizing loops
  - preallocating arrays
  - multithreading
  - parallel for
  - functions are faster than script
  - avoid large background processes
  - ...

Other useful Toolboxes

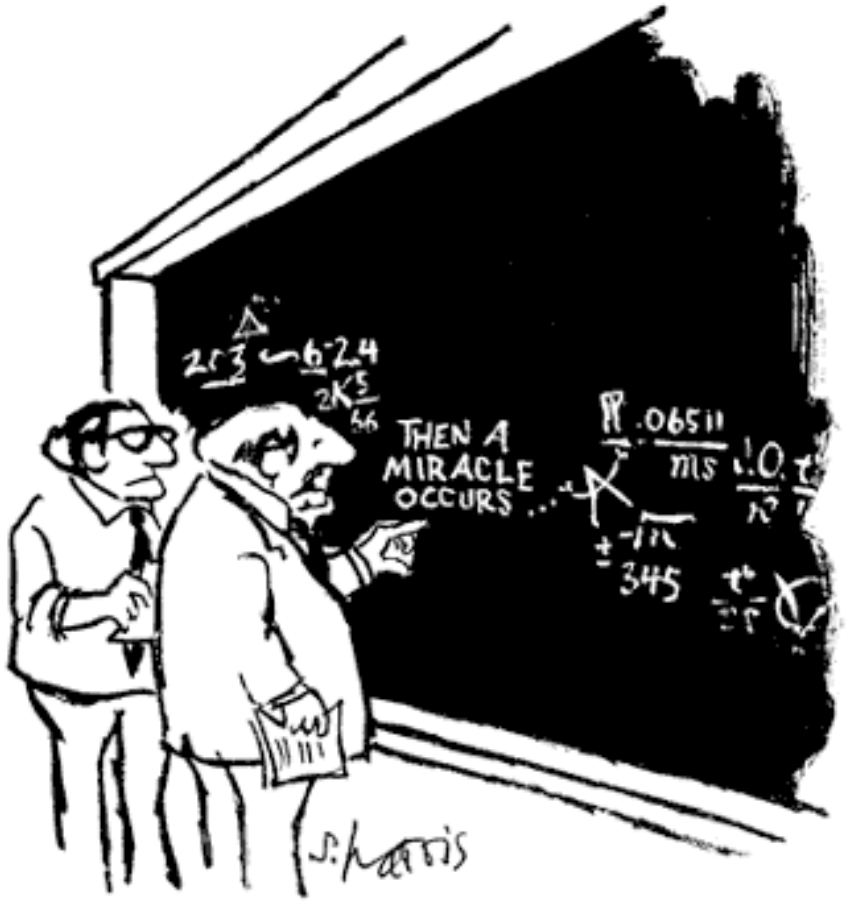
- Matlab Compiler
  - can compile M-files, MEX-files or other MATLAB code
  - can generate:
    - standalone applications on UNIX, Windows, Mac
    - C and C++ shared libraries
- Database Toolbox
  - enables to exchange data with and any ODBC/JDBC-compliant database
  - Visual Query Builder: allows to query stored data without needing to learn SQL
- Parallel Toolbox
- Neural Network Toolbox

Third Party Toolboxes

- There are many third party toolboxes, most of them can be downloaded for free.

  - Netlab: Neural Networks Toolbox
    - http://www.ncrg.aston.ac.uk/netlab/index.php
  - SOM: Self Organizing Maps Toolbox
    - http://www.cis.hut.fi/projects/somtoolbox/
  - BNT: Bayesian Networks Toolbox
    - http://www.cs.ubc.ca/~murphyk/Software/BNT/bnt.html

Send your comments...



"I think you should be more explicit here in step two."