

Matlab

Ciro Donalek

donalek@astro.caltech.edu



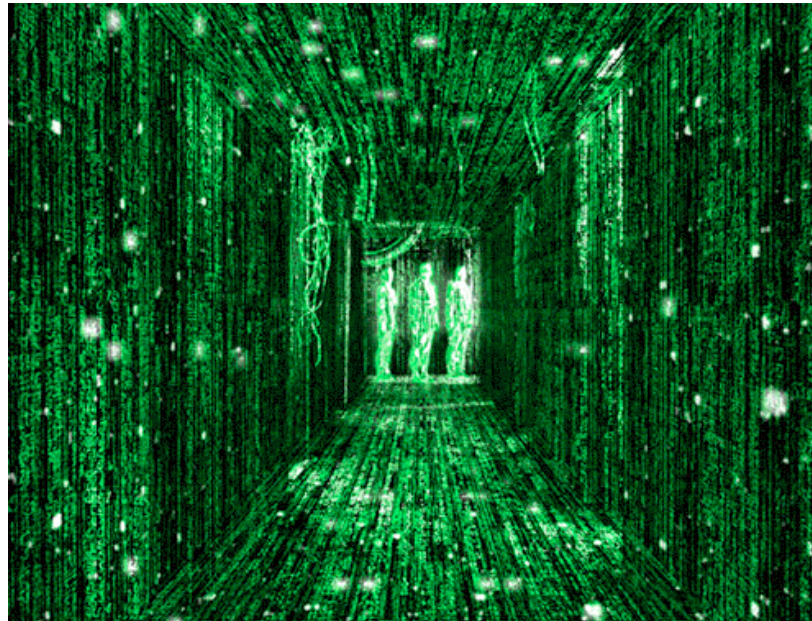
Summary

- 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 and improving
- 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

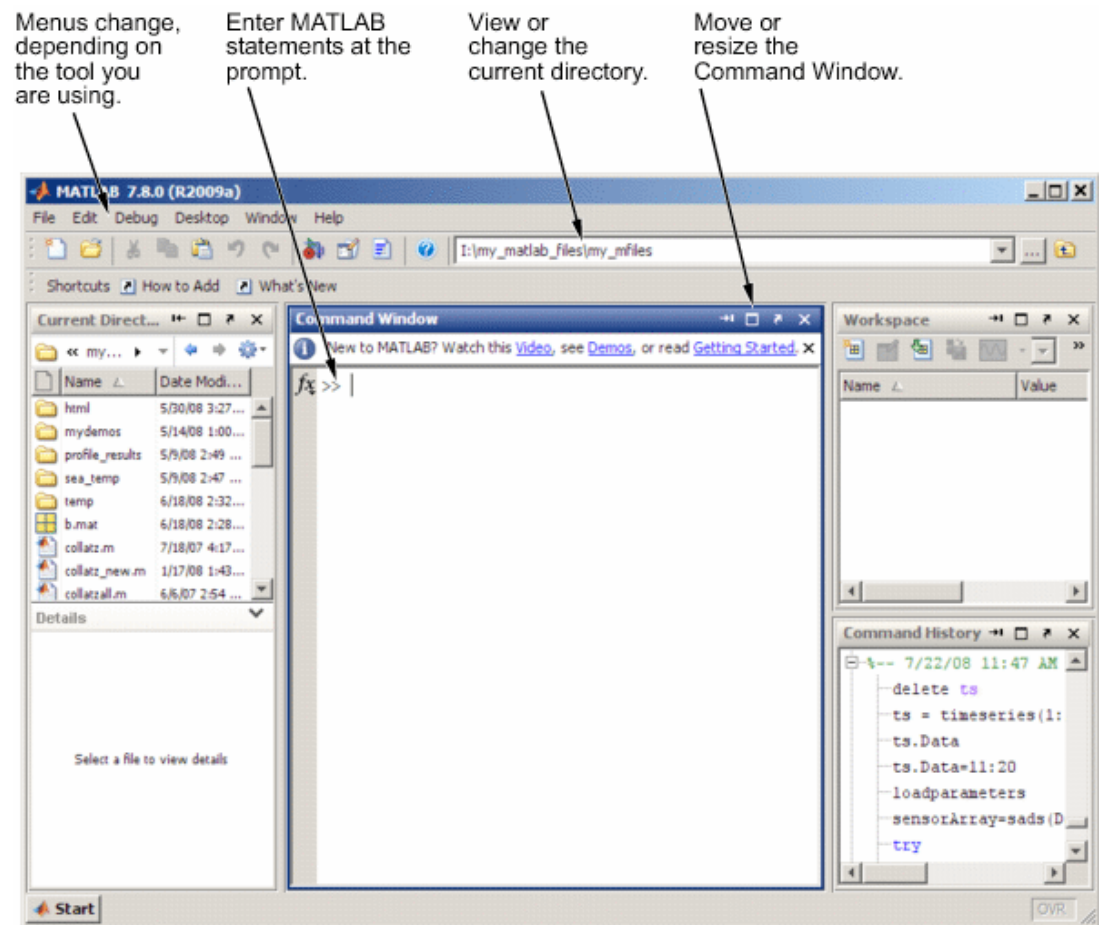
Proprietary software: distributed by MathWorks: <http://www.mathworks.com>
Latest version: 2009a (released March 6th, 2009)

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

Support for 64 bits
architectures.

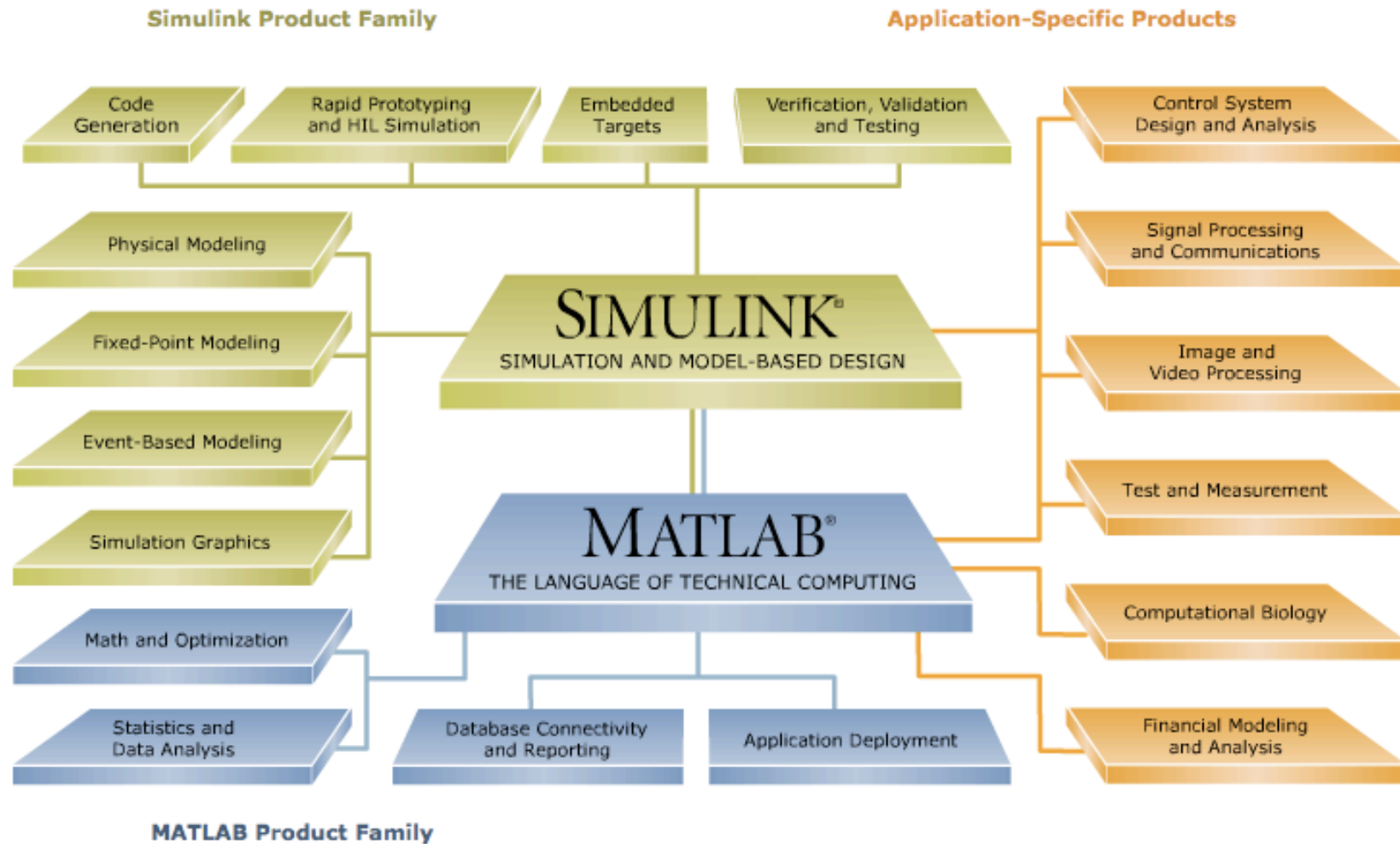
* Caltech personnel can
download it for free at:

<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



Special Variables

- `ans` (*answer*)
 - 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` (*infinite*)
  - `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

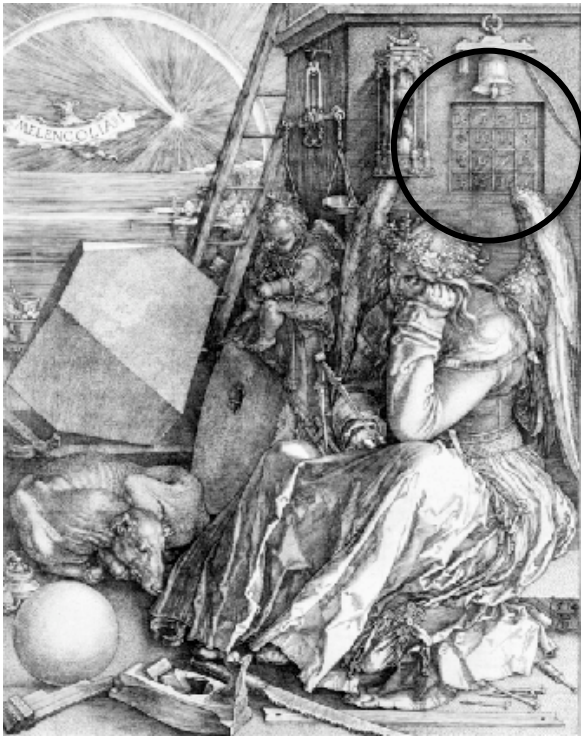
- Powerful way to access data stored in matrices.
- The element in row  $i$  and column  $j$  of  $A$  is denoted by  $A(i,j)$ .
- The colon operator `:` is one of the most important MATLAB operators.
- These features permit to minimize the use of loops (**which slows MATLAB**) and to make code simple.

```
>> 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.



|    |    |    |    |
|----|----|----|----|
| 16 | 3  | 2  | 13 |
| 5  | 10 | 11 | 8  |
| 9  | 6  | 7  | 12 |
| 4  | 15 | 14 | 1  |



# Other Data Structures

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

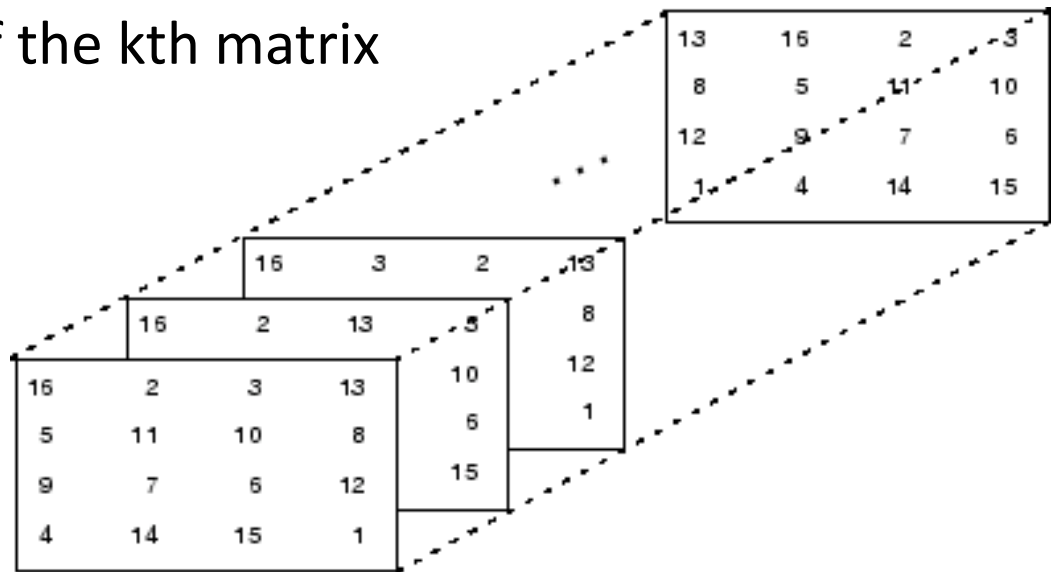
- Multidimensional Arrays
- Structures
- Cell Arrays
- Function Handlers



# 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  $k$ th 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

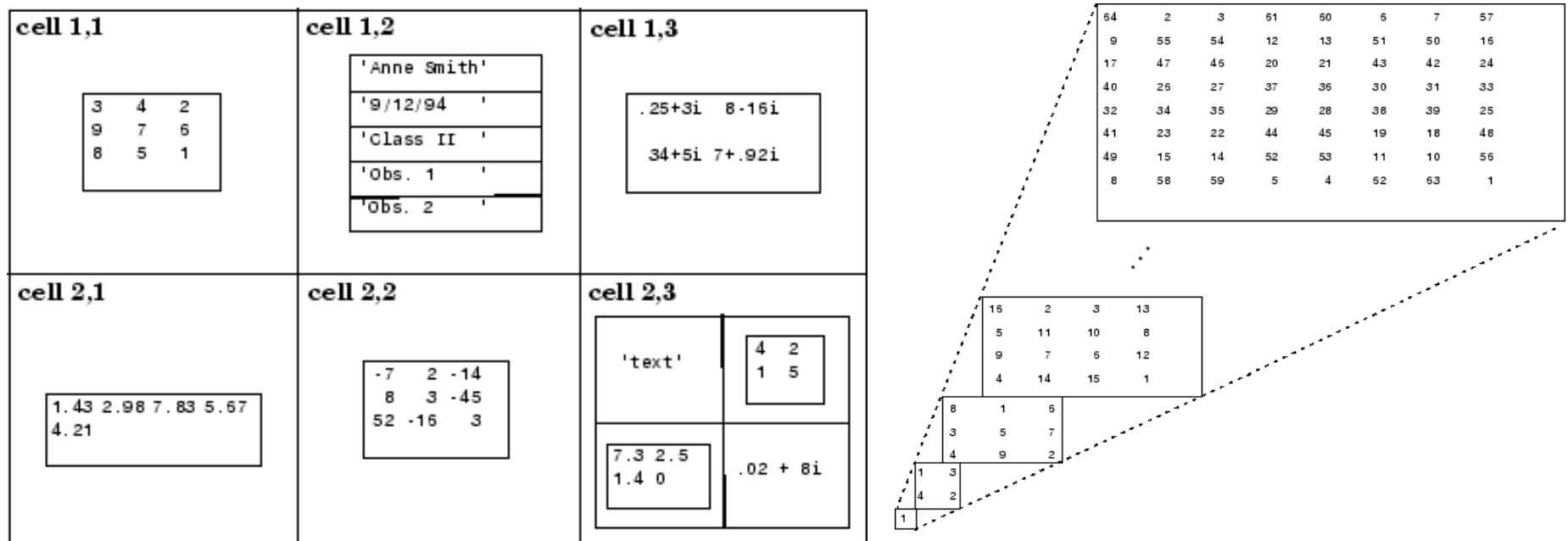
- Cell Array

- 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]
```



# 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)...

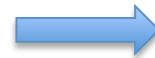
```
if RELATION
 statement
end
```

```
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?





# 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!**



# 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.



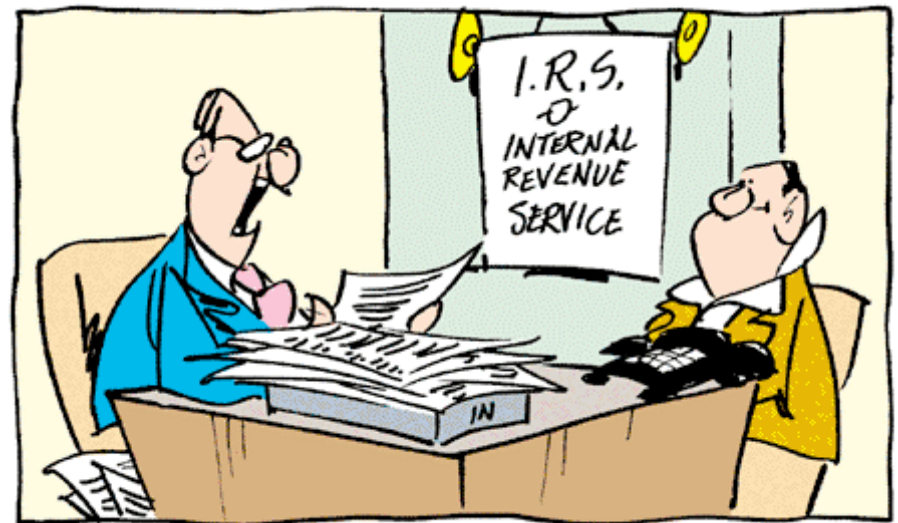
# M-files: Functions

- 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!"

