First Access Your Account

- Log into your accounts
  - Username or login = hpc_user**X**
  - Where x = sign in serial number 1 – 47
  - Password = **cacds2014**

  - Use your web browser
    - Firefox, Chromium or Google chrome

- Slides could be downloaded from URL below

```
cp   /share/apps/tutorials/intro2matlab_Part1.pdf     ~
cp   /share/apps/tutorials/intro2matlab.zip     ~
cd ~;  unzip intro2matlab.zip ; cd intro2matlab
module add matlab
matlab
```

Accessing Tutorial Materials

First login into the cluster:

TYPE AND EXECUTE COMMANDS IN RED!!!

```
cd
```

```
cp /share/apps/tutorials/intro2matlab.zip   ~
```

```
unzip intro2matlab.zip
```

```
cd intro2matlab
```

Introduction to Matlab Part I

Amit Amritkar

Center for Advanced Computation and Data Systems
(CACDS)
http://cacds.uh.edu

http://support.cacds.uh.edu

University of Houston
Houston, TX

Overview

- Introduction to Matlab
    - What is Matlab
    - Variables, arrays, matrices, indexing, etc
    - Operators
    - Flow control
    - Using of M-files

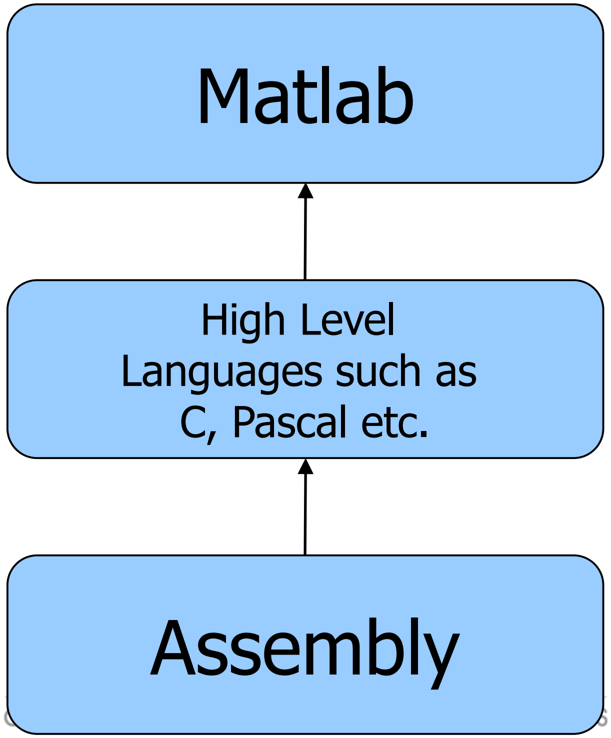
CACDS@UH

- About
  - CACDS provides high performance computing resources and related services at the University of Houston
- Mission
  - To provide state-of-the-art Tier One HPC resources and expertise propelling research in the Houston-Galveston region to a competitive advantage in education and business. Through training, CACDS aims to develop a highly-talented diverse HPC workforce as well as help promote interdisciplinary research excellence and provide competitive HPC, data analysis and scientific visualization resources.

What is Matlab?

- MATLAB is a high-level programming language for scientific computing and data visualization built around an interactive programming environment.
- Matlab is NOT "Math" lab
- The name stands for MATrix LABoratory since originally it was developed for matrix computation.
  - Think everything in matrix
  - Many specialized toolboxes for making things easier for us

How high ?



Matlab

High Level Languages such as C, Pascal etc.

Assembly

What Is Matlab? (cont.)

- Full programming language.
- Strong on matrix manipulation and graphics.
- Optional toolboxes for statistics, image processing, signal processing, etc.
- Interfaces with C, Fortran, and Java.
- Can create stand-alone executable files.
    - HHsim, a Hodgkin-Huxley simulator developed by Dave Touretzky with help from Jon Johnson, is distributed as a stand-alone executable. (Source is also available.)

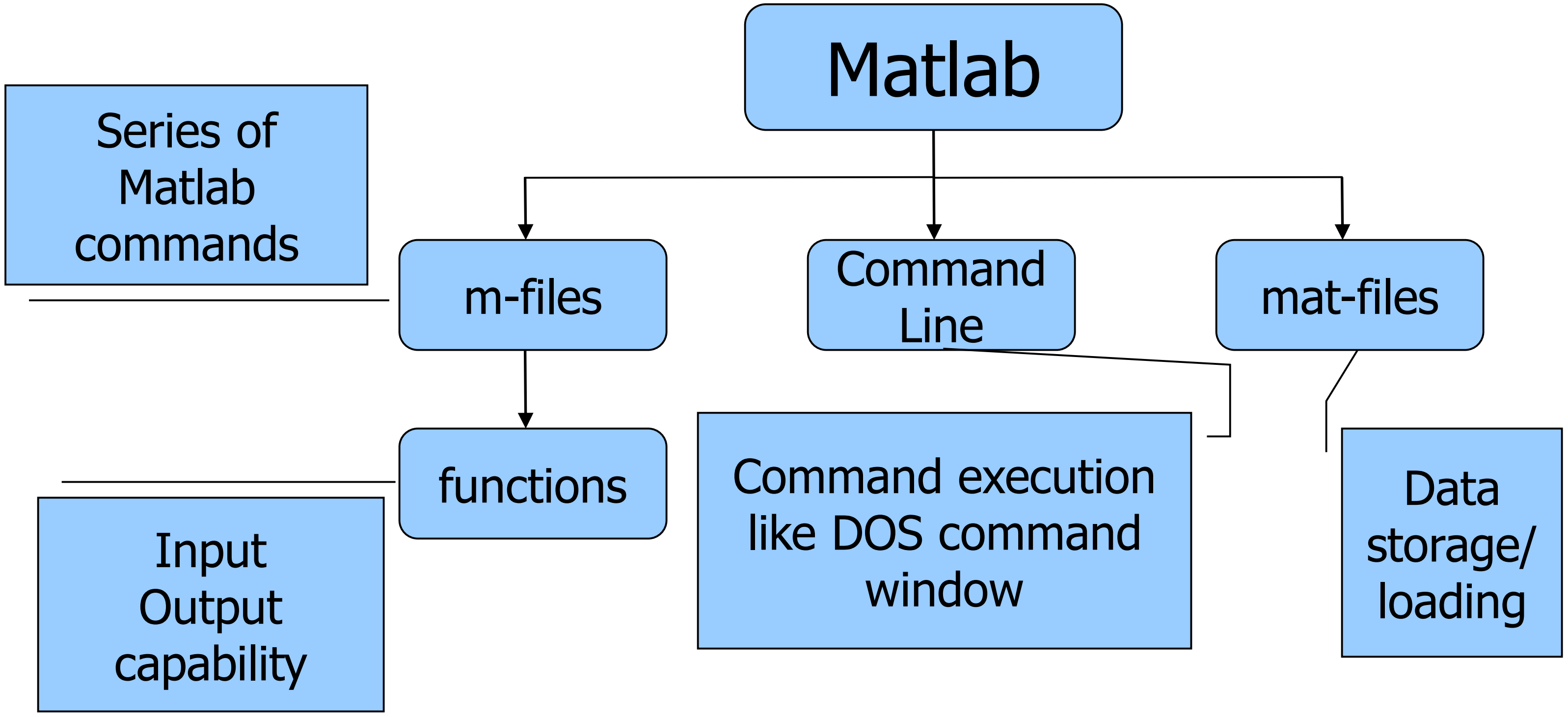
Why Should You Learn Matlab?

- Data analysis:
  - Much more versatile than a spreadsheet.
  - Extensive statistics toolbox.
  - Third party domain applications e.g. SPM uses Matlab.
    - Statistical Parametric Mapping (SPM)
- Graphics:
  - Many ways to visualize your data – even animations!
  - Produce great figures for your papers.
- Modeling and simulation:
  - Used for finite element method.

What Are We Interested In ?

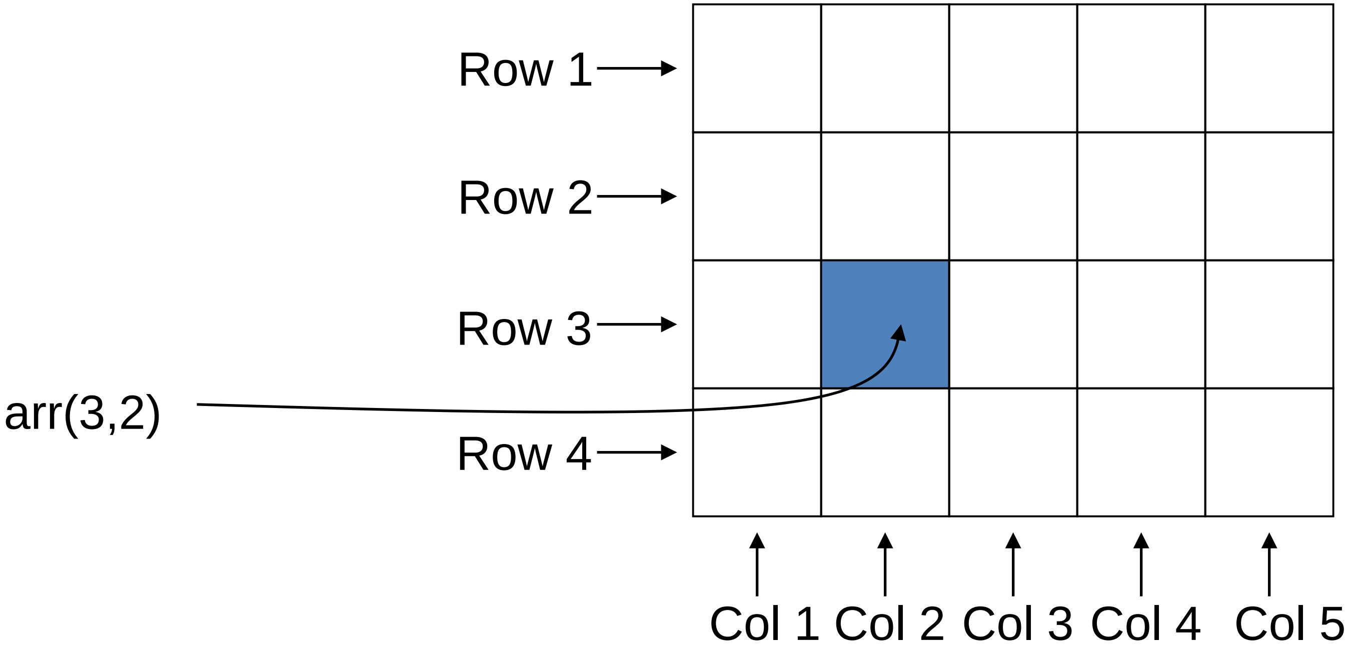
Ours is a short introduction to Matlab

The features we are going to require are



Variables and Arrays

- **Array:** A collection of data values organized into rows and columns, and known by a single name.



Row 1

Row 2

Row 3

Row 4

arr(3,2)

Col 1 Col 2 Col 3 Col 4 Col 5

Arrays

- The fundamental unit of data in MATLAB
- Scalars are also treated as arrays by MATLAB (1 row and 1 column).
- Row and column indices of an array start from 1.
- Arrays can be classified as **vectors** and **matrices**.

Vector and Matrix

- **Vector:** Array with one dimension

- **Matrix:** Array with more than one dimension

- **Size** of an array is specified by the number of rows and the number of columns, with the number of rows mentioned first (For example: n x m array).

  Total number of elements in an array is the product of the number of rows and the number of columns.

MATRIX BASICS

$$a=\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$$

3x2 matrix → 6 elements

$$b=[1 \quad 2 \quad 3 \quad 4]$$

1x4 array → 4 elements, **row vector**

$$c=\begin{bmatrix}1\\ 3\\ 5\end{bmatrix}$$

3x1 array → 3 elements, **column vector**

a(2,1)=3 b(3)=3 c(2)=3

Row # Column #

Variables

- A region of memory containing an array, which is known by a **user-specified name**.
- Contents can be used or modified at any time.
- **Variable names must begin with a letter**, followed by any combination of letters, numbers and the underscore (_) character. Only the first 31 characters are significant.
- The MATLAB language is Case Sensitive. NAME, name and Name are all different variables.

Give meaningful (descriptive and easy-to-remember) names for the variables. Never define a variable with the same name as a MATLAB function or command.

Common Types Of MATLAB Variables

- double: 64-bit double-precision floating-point numbers.
  - All variables are created with double precision unless specified
  - They can hold real, imaginary or complex numbers in the range from ±10-308 to ±10308 with 15 or 16 decimal digits.

```
>> var = 1 + i;
```

- char: 16-bit values, each representing a single character
  - The char arrays are used to hold character strings.

```
>> comment = ‘This is a character string’ ;
```

- The type of data assigned to a variable determines the type of variable that is created.

Initializing Variables in Assignment Statements

An assignment statement has the general form

var = expression

% Try these examples:

```
>> var = 40 * i;
>> var2 = var / 5;
>> array = [1 2 3 4];
>> x = 1;  y = 2;
>> a = [3.4];
>> b = [1.0 2.0 3.0 4.0];
>> c = [1.0; 2.0; 3.0];
>> d = [1, 2, 3; 4, 5, 6];
>> e = [1, 2, 3
        4, 5, 6];
```

```
>> a2 = [0 1+8];
>> b2 = [a2(2) 7 a];
>> c2(2,3) = 5;
>> d2 = [1 2];
>> d2(4) = 4;
```

';' semicolon suppresses the automatic echoing of values.

Initializing Variables in Assignment Statements

- Arrays are constructed using square brackets and semicolons
- All of the elements of an array are listed in row order
- The values in each row are listed from left to right and they are separated by blank spaces or commas
- The rows are separated by semicolons or new lines
- The number of elements in every row of an array must be the same.
- The expressions used to initialize arrays can include algebraic operations and all or portions of previously defined arrays

Initializing with Shortcut Expressions

Syntax *first: increment: last*

- **Colon operator:** a shortcut notation used to initialize arrays with thousands of elements

```
>> x = 1 : 2 : 10;
>> angles = (0.01 : 0.1 : 1) * pi;
```

- **Transpose operator:** (′) swaps the rows and columns of an array

```
>> f  = [1:4]′;
>> g = 1:4;
>> h = [ g′  g′ ];
```

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

Initializing with Shortcut Expressions Concatenation of Arrays

- `x = [1 2], y = [4 5], z=[ 0 0]`

```
A = [ x y]

      1   2   4   5

B = [x ; y]

  1 2
  4 5
```

C = [x y ; z]

Error:

??? Error using ==> vertcat CAT arguments dimensions are not consistent.

Initializing With Built-in Functions

- zeros(n)
- zeros(n,m)
- zeros(size(arr))
- ones(n)
- ones(n,m)
- ones(size(arr))
- eye(n)
- eye(n,m)
- randn(n,m)
- rand(n)
- length(arr)
- size(arr)

```
>> a = zeros(2);
>> b = zeros(2, 3);
>> c = [1, 2; 3, 4];
>> d = zeros(size(c));
```

```
>> r = 1 + 3.*randn(1,100);
```

mean ~ 1 & std ~3

Initializing with Keyboard Input

- The **input** function displays a prompt string in the Command Window and then waits for the user to respond.

```
my_val = input( 'Enter an input value: ' );

in1 = input( 'Enter data: ' );
```

- To enter a string

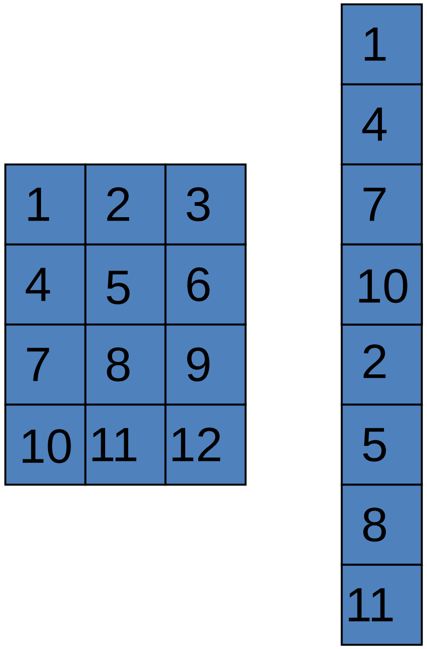
```
in2 = input( 'Enter data: ' ,'s');
```

Multidimensional Arrays

- A two dimensional array with m rows and n columns will occupy mxn successive locations in the computer’s memory. MATLAB always allocates array elements in **column major order.**

a= [1 2 3; 4 5 6; 7 8 9; 10 11 12];

a(5) = a(1,2) = 2



1	2	3
4	5	6
7	8	9
10	11	12

1
4
7
10
2
5
8
11

- A 2x3x2 array of three dimensions

c=zeros(2,3,2);

c(:, :, 1) = [1 2 3; 4 5 6];

c(:, :, 2) = [7 8 9; 10 11 12];

Size of a Matrix

```
pts=randn(5,4)

whos pts

size(pts)

length(pts)
```

- equivalent to max(size(pts))

Subarrays and Subscripting

```
V = [10 20 30 40 50];

V(3)

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

```
M =
          1    2    3
          4    5    6
          7    8    9
```

```
M(2,2)
M(2)
M(6)
```

access in **column-major** order

Subarrays and Subscripting

- It is also possible to select and use subsets of MATLAB arrays.

```
arr1 = [1.1  -2.2  3.3  -4.4  5.5];
arr1(3) is 3.3
arr1([1  4]) is the array [1.1  -4.4]
arr1(1 : 2 : 5) is the array [1.1  3.3  5.5]
```

- For two-dimensional arrays, a colon can be used in a subscript to select all of the values of that subscript.

```
arr2 = [1  2  3; -2  -3  -4; 3  4  5];
arr2(1, :)
arr2(:, 1:2:3)
```

Subarrays and Subscripting
Matrix Slices

```
V=[0:2:10]

V(2:4)

V(2:end)

M=rand(5);

M(1:2, 2:3)

M( : )

M(: , :)
```

Subarrays and Subscripting

- The **end** function: When used in an array subscript, it returns the highest value taken on by that subscript.

```
arr3 = [1 2 3 4 5 6 7 8];
arr3(5:end) is the array [5 6 7 8]
arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12];
arr4(2:end, 2:end)
```

- Using subarrays on the left hand-side of an assignment statement:

```
arr4(1:2, [1 4]) = [20  21;  22  23];
(1,1) (1,4) (2,1) and (2,4) are updated.
arr4 = [20  21;  22  23];   all of the array is changed.
```

Subarrays and Subscripting

- Assigning a Scalar to a Subarray: A scalar value on the right-hand side of an assignment statement is copied into every element specified on the left-hand side.

```
>> arr4 = [1 2 3 4; 5 6 7 8; 9 10 11 12];
>> arr4(1:2, 1:2) = 1
arr4 =
        1    1    3    4
        1    1    7    8
        9   10   11   12
```

Expanding a Matrix

```
a = [1 2 3]

a = [a 4]

a(7) = 5

a(end+1) = 6

b = [a ;  a.^2]
```

Efficiency tip:

Use ZEROS(rows,cols) to preallocate large arrays instead of growing them dynamically.

Reshaping a Matrix

```
M = reshape(1:12,  4,  3)

M'

M' '  or  (M')'
```

Exercise (1)

Create the following matrix using **only** the colon, reshape, and transpose operators.

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \\ 13 & 14 & 15 \end{matrix}$$

Deleting Rows or Columns

```
M=rand(5)

M(: , 3) = [ ]

M(2, :) = [ ]

size([ ])
```

Reduction Operators

```
M = rand(5, 3)

sum(M)         sum along first  dimension
sum(M, 2)      sum along second dimension

sum, prod, min, max, mean, var

min(min(M))
min( M(:) )
```

Expanding with REPMAT

REPMAT is often used to expand a vector to fit the shape of a matrix.
Example: adjusting a dataset to have zero mean.

```
M = rand(5,3);
avgs = mean(M);
Mavgs = repmat(avgs, 5, 1)
Mzero = M - Mavgs;
sum(Mzero)
```

```
Mavgs =

   0.6220    0.7448    0.5099
   0.6220    0.7448    0.5099
   0.6220    0.7448    0.5099
   0.6220    0.7448    0.5099
   0.6220    0.7448    0.5099


ans =

  1.0e-15 *

   0.3331    0.3331    0.1110
```

Exercise (2)

- Suppose we want the *rows* of M to sum to zero, instead of the columns.

- How would you do this, *without* using the transpose operator?

Special Values

- MATLAB includes a number of predefined special values. These values can be used at any time without initializing them.
- These predefined values are stored in ordinary variables. They can be overwritten or modified by a user.
- If a new value is assigned to one of these variables, then that new value will replace the default one in all later calculations.

```
>> circ1 = 2 * pi * 10;
>> pi = 3;
>> circ2 = 2 * pi * 10;
```

Never change the values of predefined variables.

Special Values

- pi: π value up to 15 significant digits
- i, j: sqrt(-1)
- Inf: infinity (such as division by 0)
- NaN: Not-a-Number (division of zero by zero)
- clock: current date and time in the form of a 6-element row vector containing the year, month, day, hour, minute, and second
- date: current date as a string such as *16-Feb-2004*
- eps: epsilon is the smallest difference between two numbers
- ans: stores the result of an expression

Changing The Data Format

```
>> value = 12.345678901234567;
   format short           → 12.3457
   format long            → 12.34567890123457
   format short e         → 1.2346e+001
   format long e          → 1.234567890123457e+001
   format short g         → 12.346
   format long g          → 12.3456789012346
   format rat             → 1000/81
```

Disp function

The disp(*array*) function

```
>> disp( 'Hello' )
Hello
>> disp(5)
     5
>> disp( [ 'Bilkent ' 'University' ] )
Bilkent University
>> name = 'Alper';
>> disp( [ 'Hello ' name ] )
Hello Alper
```

Conversion Functions

The num2str() and int2str() functions

```
>> d = [ num2str(16) '-Feb-' num2str(2004) ];
>> disp(d)
16-Feb-2004
>> x = 23.11;
>> disp( [ 'answer = ' num2str(x) ] )
answer = 23.11
>> disp( [ 'answer = ' int2str(x) ] )
answer = 23
```

Changing The Data Format

The fprintf(*format*, *data*) function

– %d	integer
– %f	floating point format
– %e	exponential format
– %g	either floating point or exponential format, whichever is shorter
– \n	new line character
– \t	tab character

Changing The Data Format

```
>> fprintf( 'Result is %d', 3 )
Result is 3
>> fprintf( 'Area of a circle with radius %d is %f', 3, pi*3^2 )
Area of a circle with radius 3 is 28.274334
>> x = 5;
>> fprintf( 'x = %3d', x )
x =   5
>> x = pi;
>> fprintf( 'x = %0.2f', x )
x = 3.14
>> fprintf( 'x = %6.2f', x )
x =   3.14
>> fprintf( 'x = %d\ny = %d\n', 3, 13 )
x = 3
y = 13
```

MATLAB Files

Data files

- save *filename var1 var2 …*

```
>> save myfile.mat  x  y                → binary
>> save myfile.dat  x  –ascii           → ascii
```

- load *filename*

```
>> load myfile.mat                      → binary
>> load myfile.dat  –ascii              → ascii
```

MATLAB Operations

- *variable_name* = *expression*;

addition	a + b	→	a + b
subtraction	a - b	→	a - b
multiplication	a x b	→	a * b
division	a / b	→	a / b
exponent	a^b	→	a ^ b

Hierarchy of operations

- x = 3 * 2 + 6 / 2
- Processing order of operations is important
  - parentheses (starting from the innermost)
  - exponentials (from left to right)
  - multiplications and divisions (from left to right)
  - additions and subtractions (from left to right)

```
>> x = 3 * 2 + 6 / 2
x =
     9
```

Built-in MATLAB Functions

result = function_name(input);

- abs, sign
- log, log10, log2
- exp
- sqrt
- sin, cos, tan
- asin, acos, atan
- max, min
- round, floor, ceil, fix
- mod, rem

- help elfun → help for elementary math functions

Types Of Errors İn MATLAB Programs

- Syntax errors

  Fix: Check spelling and punctuation

- Run-time errors

  Fix: Check input data

  Can remove “;” or add “disp” statements

- Logical errors

  Fix

  –Use shorter statements

  –Check typos

  –Check units

  –Ask your friends, assistants, instructor, …

Linear Algebra

★ A+B

★ A-B

A' for the transpose of A

★ inv(A) for the inverse of A

★ det(A) for determinant of A

★ diag(A) for a vector equal to the diagonal elements of A

* must be used with appropriate sized matrices, e.g., square matrices to inv and det and conformable matrices

Element Wise Operations

A= magic(3)

```
8   1   6
3   5   7
4   9   2
```

A.^2

```
64    1   36
 9   25   49
16   81    4
```

Note A*A =

```
91   67   67
67   91   67
67   67   91
```

Other element-wise operators: + − .* ./ .^

Logical Operations

Operators: ==   ~=   <   >   <=   >=

Can't use != as in Java or C

Logical values:

0 means “*false*”

1 (or any non-zero number) means “*true*”

a = (3 >= 1 : 5)   What are the type and size of a?

```
a =

   1   1   1   0   0
```

Conditional Statements And Looping

any & all

```
 a=randn(2)
 0.47785  -0.52364
  -1.46193   1.29783
b=randn(2)
  -0.68481  -1.20778
  -0.10133  -0.85876


any (a<b)
1 0

all (a<b)
0 0
```

Conditional Statements And Looping

If & else

```
if expression
...
end
```

```
% Example 1:
X=2; if X>0, A=1/X; else A=0, end
```

```
if expression
...
else
...
end
```

```
%Example 2:
x = 5;
if (x > 10 )
y = 10;
elseif (x < 0)
y = 0;
else
y = x;
end
```

While loop

Syntax

```
while expression
...statements
end
```

Example 1:

```
while  ((a>3) & (b==5))
     Some Matlab Commands;
end
```

Example 2:
How quickly can a random accumulator reach 5?

```
accum = 0; steps = 0;
while accum < 5
  steps = steps + 1;
  accum = accum + rand(1);
end
steps, accum
```

Used when you don't know how many times the loop is to be executed

Conditional Statements And Looping

For loop

Syntax:

```
for index = values
   statements
end
```

```
% Example:
n=10; X=[1:n]; X(:)=0
for i=2:n
 X(i)=3*X(i-1)+1
end
```

Batch/Script file

Use of M files

- Batch or script file in Matlab is simple. Simply put your code in a file with extension .m
- Every symbol will be visible in the current workspace
- Good for initial development

Functions

- Functions are exactly the same as batch files except they hide all variables excluding those you return

```
function out1=function_name(in1)
function out1=function_name(in1,in2,in3)
function [out1,out2]=function_name(in1,in2)
```

You should write this command at the beginning of the m-file and you should **save the m-file with a file name same as the function name**

Functions

file isprime1.m

```
function bool=isprime1(n)
bool=1;
for i=2:floor(sqrt(n))
        if mod(n,i)==0
                bool=0; break;
        end
end
```

in Matlab terminal

```
>> type isprime1.m
function bool=isprime1(n)
bool=1;
for i=2:floor(sqrt(n))
        if mod(n,i)==0
                bool=0; break;
        end
end
>> bool=isprime1(5);
```

Function pointers
lambda functions:

```
>> add=@(x,y) x+y
add =
   @(x,y)x+y

>> add(1,2)
ans =
   3
```

```
>> addone=@(x)
add(1,x);
>> addone(3)
ans =
   4
```

Scripts vs. Functions

Scripts take no input arguments and produce no return values.

Scripts operate in the workspace of their caller.

If called from the command line, scripts operate in the **base workspace**.

If called from within a function, scripts operate in the function's **local workspace** and can see and modify its local variables.

Scripts vs. Functions

Functions can take zero or more arguments and return zero or more values.

Functions operate in their own **local workspace**.

Variables created inside a function are local to that function.

Local variables disappear when the function returns.

File I/O

- fopen
- fread
- fwrite
- fprintf

More or less same as C

- save/load

```
>> save tmp a b c
>> load tmp
>> save tmp2 a -ascii
```

Profiling

- Using `tic/toc`

```
>> tic; parfor i=1:3; c(:,i)=eig(rand(1000)); end; toc
Elapsed time is 0.862877 seconds.


>> tic; for i=1:3; c(:,i) = eig(rand(1000)); end; toc
Elapsed time is 2.036540 seconds.
```

- Matlab has its own profiling tools
  - `help profile`

Basic Plotting with Matlab

```
N = 1000;
f = zeros(N+1,1);
t = zeros(N+1,1);
for i = 1:N
        t(i) = (i-1)/N;
        f(i) = sin(2*pi*t(i));
end
plot(t,f);
title('The Sine Function'); xlabel('t', 'FontSize',14);
ylabel('sin(2 pi t)','FontSize',14);
print  -djpeg  myfig.jpg
print  -depsc  -r300  myfig.ps
print  -dtiff  myfig.tiff
```

Basic Plotting + Vectorization

```
N = 1000;
t = [1:N]/N;
f = sin(2*pi*t);

plot(t,f);
title('The Sine Function'); xlabel('t', 'FontSize',14);
ylabel('sin(2 pi t)','FontSize',14);
print  -djpeg  myfig2.jpg
print  -depsc  -r300  myfig2.ps
print  -dtiff  myfig2.tiff
```

Histograms

```
dat = randn(10000, 1);

hist(dat)

hist(dat, 50)

b = hist(dat, 6)

bar(b)
print  -dtiff  myhistogram.tiff
```

3D Plots

```
[X,Y] = meshgrid(-2:.2:2, -2:.2:2);
Z = X .* exp(-X.^2 - Y.^2);
surf(X,Y,Z)
xlabel ('X','Fontsize',18)
ylabel ('Y','Fontsize',18)
zlabel ('Z','Fontsize',18)
title ('3D X,Y,Z Plot of Z = X .* exp(-X.^2 - Y.^2)','Fontsize',18)
print  -dtiff  myfig2.tiff
```

Exercise (3)

- Create a vector, t, that ranges from 0 to 1 with 1000 equally spaced points. Plot the functions cos(2*pi*t). label the x and y axis

Ways To Learn Matlab

- Tutorial videos at mathworks.com
- Built-in demos:
  - doc demo
- Browse the online documentation
- Dozens of books:
  - Amazon.com reports 4,600 search results!
- Matlab Central: user community site
  - http://www.mathworks.com/matlabcentral
- Useful programs by John Burkardt –
  - http://people.sc.fsu.edu/~jburkardt/m_src/m_src.html
- Questions to support@mathworks.com

Matlab Documentation

help cos
doc cos

clf, peaks
click on rotate3D icon
which peaks
edit peaks

lookfor rotate

Yes!
You CAN see
our
source code!

Browsing Online Documentation

Help pulldown menu
> Product Help

In the help browser:
> MATLAB
> > Statistics Toolbox
> > > User's Guide
> > > > Probability Distributions
> > > > > Supported Distributions
> > > > > > Beta

MATLAB BASICS

Summary

help *command*	→ Online help
lookfor *keyword*	→ Lists related commands
which	→ Version and location info
clear	→ Clears the workspace
clc	→ Clears the command window
diary *filename*	→ Sends output to file
diary on/off	→ Turns diary on/off
who, whos	→ Lists content of the workspace
more on/off	→ Enables/disables paged output
Ctrl+c	→ Aborts operation
…	→ Continuation
%	→ Comments

Introduction to Matlab Part II

- parallel computing
- job submission to a HPC cluster
- compiling M code
- ….

- will try to tailor it as per your requests

Questions?

- My office – PGH 223
- Email – aramritkar@uh.edu

- Please leave your feedback!