Numerical Methods

MTH4002

Laboratory 01: Octave/Matlab Tutorial

Dr. Kundan Kumar
Associate Professor
Department of ECE

Faculty of Engineering (ITER)
S‘O’A Deemed to be University, Bhubaneswar, India-751030

Outline

1. Introduction
2. Start, quit, getting help
3. Variables
4. Matrix
5. Strings
6. Plotting
7. Control structures
8. Function
9. References

Introduction

- Octave is the "open-source Matlab"
- Octave is a great gnuplot wrapper

www.octave.org

www.mathworks.com

- Octave and Matlab are both, high-level languages and mathematical programming environments for:
  - Visualization
  - Programming, algorithm development
  - Numerical computation: linear algebra, optimization, control, statistics, signal and image processing, etc.
- Beware: Octave/Matlab programs can be slow.

Introduction

- Matlab-Octave comparison:
  - Matlab is more flexible/advanced/powerful/costly
  - Octave is for free (GPL license)
  - There are minor differences in syntax
- This tutorial:
  - This tutorial applies to Octave and Matlab unless stated otherwise!
- Current versions (autumn 2020):
  - Octave 5.2.0
  - Matlab 9.9

Installation Guide

- Open the website

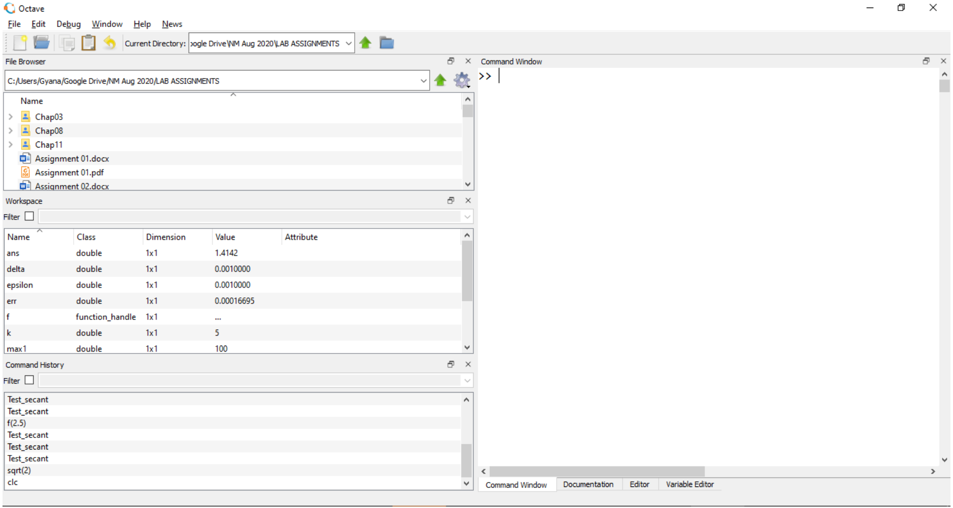
  “https://www.gnu.org/software/octave/download.html”

- Download the file

  “octave-5.2.0_1-w64-installer.exe” (for windows-64)
  “octave-5.2.0_1-w32-installer.exe” (for windows-32)

- Double click on the downloaded file to start the installation and follow the instruction for complete installation.

Installation Guide



Current Directory: ›ogle Drive\NM Aug 2020\LAB ASSIGNMENTS

C:/Users/Gyana/Google Drive/NM Aug 2020/LAB ASSIGNMENTS

Name
- Chap03
- Chap08
- Chap11
- Assignment 01.docx
- Assignment 01.pdf
- Assignment 02.docx

Name	Class	Dimension	Value	Attribute
ans	double	1x1	1.4142	
delta	double	1x1	0.0010000	
epsilon	double	1x1	0.0010000	
err	double	1x1	0.00016695	
f	function_handle	1x1	...	
k	double	1x1	5	
max1	double	1x1	100	

```
Test_secant
Test_secant
f(2.5)
Test_secant
Test_secant
Test_secant
sqrt(2)
clc
```

```
>>
```

Start, Quit, Getting Help

- To start Octave type the shell command octave, double-click Octave.app or whatever your OS needs. You should see the prompt:

```
octave:1\>
```

- To start Octave GUI double-click on the desktop shortcut GNU Octave GUI.
- If you get into trouble, you can interrupt Octave by typing ctrl-c.
- To exit Octave, type quit or exit.
- To get help, type help or doc.
- To get help on a specific command (=built-in function), type help command
- Examples: `help size`, `help plot`, `help figure`, `help inv`, ...
- To get help on the help system, type `help help`

Start, Quit, Getting Help

- In the help text of Matlab functions, function names and variables are in capital letters.
    - □ Don't get confused! The (case-sensitive) naming convention specifies lowercase letters for built-in commands. It is just a way to highlight text.
- Example: `help round` returns ROUND Round towards nearest integer. ROUND(X) rounds the elements of X to the nearest integers. See also `floor`, `ceil`, `fix`. [...]
- Octave texts are mixed, in lower- and uppercase.

Octave as calculator

+,-,*,/	operators
^	exponential
log	natural log
exp	natural exponent
sin,cos,tan	trigonometric function
asin,acos,atan	inverse trigonometric function

- Examples:

```
2+3 = 5
3*4 = 12
log(1) = 0
sin(pi/2) = 1
```

Variables and Data Types

- Matrices (real and complex)
- Strings (matrices of characters)
- Structures
  - □ Vectors? It's a matrix with one column/row
  - □ Scalars? It's a matrix of dimension 1x1
  - □ Integers? It's a double (you never have to worry)
  - □ Boolean? It's an integer (non-null=true, 0=false)
- Almost everything is a matrix!
- Matlab and Octave both support Object Oriented Programming.

Variables and Data Types

- Creating a Matrix
  - □ Simply type:

```
octave:1> A = [8, 2, 1; 3, -1, 4; 7, 6, -5]
```

  - □ Octave will respond with a matrix in pretty-print:

```
A =

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

- More on matrices, further down this tutorial.

Variables and Data Types

- Creating a Character string:

```
octave:4> str = 'Hello World'
```

- Creating a Structure using instance:

```
octave:5> data.id = 3;
octave:6> data.timestamp = 1265.5983;
octave:7> data.name = 'sensor 1 front';
```

Variables and Data Types

- Creating a Array of Structures
  - □ Oh, a new measurement arrives. Extend struct by:

```
octave:8> data(2).id = 4;
octave:9> data(2).timestamp = 1268.9613;
octave..> data(2).name = 'sensor 1 front';
```

  - □ Octave will respond with:

```
data =
{
    1x2 struct array containing the fields:
     id
     timestamp
     name
}
```

Variables and Data Types

- Display Variables: Simply type its name

```
octave:1> a = 4
```

- Suppress Output: Add a semicolon

```
octave:2> a;
octave:3> sin(phi);
```

Applies also to function calls.

Variables and Data Types

- Variables have no permanent type. `s = 3` followed by `s = 'octave'` is fine
- Use `who` (or the more detailed `whos`) to list the currently defined variables. Example output:

```
Variables in the current scope:
    Attr Name        Size                   Bytes  Class
    ==== ====        ====                   =====  =====
         A           3x3                       72  double
         a           1x1                        8  double
         ans        21x1                      168  double
         s           1x5                        5  char
         v           1x21                      24  double
```

Variables and Data Types

- Numerical Precision: Variables are stored as double precision numbers in IEEE floating point format.

`realmin`	`Smallest positive floating point number: 2.23e-308`
`realmax`	`Largest positive floating point number: 1.80e+308`
`eps`	`Relative precision: 2.22e-16`

Variables and Data Types

- Control Display of Float Variables

`format short`	`Fixed point format with 5 digits`
`format long`	`Fixed point format with 15 digits`
`format short e`	`Floating point format, 5 digits`
`format long e`	`Floating point format, 15 digits`
`format short g`	`Best of fixed or floating point with 5 digits (good choice)`
`format long g`	`Best of fixed or floating point with 15 digits`

- See `help format` for more information

Variables and Data Types

- Talking about Float Variables...

`ceil(x)`	`Round to smallest integer not less than x`
`floor(x)`	`Round to largest integer not greater than x`
`round(x)`	`Round towards nearest integer`
`fix(x)`	`Round towards zero`

- If `x` is a matrix, the functions are applied to each element of `x`.

Creating a Matrix

- Simply type:

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

  To delimit columns, use comma or space. To delimit rows, use semicolon.

- The following expressions are equivalent

  ```
  A = [8 2 1;3 -1 4;7 6 -5]
  A = [8,2,1;3,-1,4;7,6,-5]
  ```

- Octave will respond with a matrix in pretty-print:

  ```
  A =

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

Creating a Matrix

- Alternative Example:

```
>> phi = pi/3;
>> R = [cos(phi) -sin(phi); sin(phi) cos(phi)]
R =
        0.50000  -0.86603
        0.86603   0.50000
```

Creating a Matrix from Matrices

```
>> A = [1 1 1; 2 2 2];  B = [33; 33];
```

- Column-wise

```
>> C = [A B]
C =
      1     1     1     33
      2     2     2     33
```

- Row-wise:

```
>> D = [A; [44 44 44]]
D =
      1     1     1
      2     2     2
      44    44    44
```

Indexing

- Always ”row before column”!

```
aij = A(i,j)        Get an element
r = A(i,:)          Get a row
c = A(:,j)          Get a column
B = A(i:k,j:l)      Get a submatrix
```

- Useful indexing command end:

```
>> data = [4 -1 35 9 11 -2];
>> v = data(3:end)
v =
    35   9   11   -2
```

Indexing

- Colon ':', two meanings:
  - □ Wildcard to select entire matrix row or column

    ```
    A(3,:), B(:,5)
    ```

  - □ Defines a range in expressions like

    ```
    indices = 1:5        Returns row vector 1,2,3,4,5
    steps = 1:3:61       Returns row vector 1,4,7,...,61
    t = 0:0.01:1         Returns vector 0,0.01,0.02,...,1
    ```

- Useful command to define ranges: `linspace`

Indexing

- Assigning a Row/Column: All referenced elements are set to the scalar value.

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

- Adding a Row/Column: If the referenced row/colum doesn't exist, it's added.

```
>> A(4,:) = 4
A =
     1    2    3    4    5
     2    2    2    2    2
    -3   -3   -3   -3   -3
     4    4    4    4    4
```

Indexing

- Deleting a Row/Column: Assigning an empty matrix [] deletes the referenced rows or columns. Examples:

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

Sizes

- Get Size

`nr = size(A,1)`	Get number of rows of A
`nc = size(A,2)`	Get number of columns of A
`[nr nc] = size(A)`	Get both (remember order)
`l = length(A)`	Get whatever is bigger
`numel(A)`	Get number of elements in A
`isempty(A)`	Check if A is empty matrix []

- Octave only:

`nr = rows(A)`	Get number of rows of A
`nc = columns(A)`	Get number of columns of A

Operations

- Matrix Operations

`3*A`	Multiply by scalar
`A*B + X - D`	Add and multiply
`B = A’`	Transpose A
`inv(A)`	Invert A
`s = v’*Q*v`	Mix vectors and matrices
`det(A)`	Determinant of A
`[v lambda] = eig(A)`	Eigenvalue decomposition
`[U S V] = svd(A)`	Singular value decomposition
	many many more...

Operations

- Vector Operations (With x being a column vector)

```
s = x'*x        Inner product, result is a scalar
X = x*x'        Outer product, result is a matrix
e = x*x         Gives an error
```

- Element-Wise Operations (for vectors/matrices)

```
s = x.+x        Element-wise addition
p = x.*x        Element-wise multiplication
q = x./x        Element-wise division
e = x.^3        Element-wise power operator
```

Vector Functions

- Useful Vector Functions

`sum(v)`	`Compute sum of elements of v`
`cumsum(v)`	`Compute cumulative sum of elements of v`
`prod(v)`	`Compute product of elements of v`
`cumprod(v)`	`Compute cumulative product of elements of v`
`diff(v)`	`Compute difference of subsequent elements [v(2)-v(1) v(3)-v(2) ...]`
`mean(v)`	`Mean value of elements in v`
`std(v)`	`Standard deviation of elements`

Vector Functions

- Useful Vector Functions

`min(v)`	Return smallest element in v
`max(v)`	Return largest element in v
`sort(v,'ascend')`	Sort in ascending order
`sort(v,'descend')`	Sort in descending order
`find(v)`	Return vector of indices of all non-zero elements in v. Great in combination with vectorized conditions. Example: `ivec = find(datavec == 5).`

Special Matrices

- Special Matrices

```
A = zeros(m,n)       Zero matrix of size m x n
B = ones(m,n)        Matrix of size m x n with all 1's
I = eye(n)           Identity matrix of size n
D = diag([a b c])    Diagonal matrix of size 3 x 3
                     with a,b,c in the main  diagonal
```

- Just for fun

```
M = magic(n)         Magic square matrix of size
                     n x n. (All rows and columns
                     sum up to the same number)
```

Special Matrices

- Random Matrices and Vectors

```
R = rand(m,n)         Matrix with m x n uniformly
                      distributed random numbers
                      from interval [0..1]
N = randn(m,n)        Row vector with m x n normally
                      distributed random numbers
                      with zero mean, unit variance
v = randperm(n)       Row vector with a random
                      permutation of the numbers 1 to n
```

Multi-Dimensional Matrices

- Matrices can have more than two dimensions.
- Create a 3-dimensional matrix by typing, e.g.,

```
>> A = ones(2,5,2)
```

- Octave will respond by

```
A =
ans(:,:,1) =
    1    1    1    1    1
    1    1    1    1    1
ans(:,:,2) =
    1    1    1    1    1
    1    1    1    1    1
```

Multi-Dimensional Matrices

- All operations to create, index, add, assign, delete and get size apply in the same fashion
- Examples:

```
[m n l] = size(A)
A = rand(m,n,l)
m = min(min(min(A)))
aijk = A(i,j,k)
A(:,:,5) = -3
```

Matrix Massage

- Matrix Massage

```
reshape(A,m,n)             Change size of matrix A to
                           have dimension m x n. An
                           error results if A does not
                           have m x n elements
circshift(A,[m n])         Shift elements of A m times
                           in row dimension and n  times
                           in column dimension
shiftdim(A,n)              Shift the dimension of A by n.
                           Generalizes transpose for
                           multi-dimensional matrices
```

Matrix Massage

- Examples: Let `P = [x1; y1; x2; y2; ...]` be a 2nx1 column vector of `n` `(x,y)`-pairs. Make it a column vector of `(x,y,theta)`-tuples with all `theta` values being `pi/2`:
  - □ Make it a 2xn matrix

    ```
    >> P = reshape(P,2,numel(P)/2);
    ```

  - □ Add a third row, assign pi/2

    ```
    >> P(3,:) = pi/2;
    ```

  - □ Reshape it to be a 3nx1 column vector

    ```
    >> P = reshape(P,numel(P),1);
    ```

Strings

- Most Often Used Commands

strcat	Concatenate strings
int2str	Convert integer to a string
num2str	Convert numbers to a string
sprintf	Write formatted data to a string. Same as C/C++ fprintf for strings.

- Example

```
s = strcat('At step ',int2str(k),', p = ',num2str(p,4))
```

Given that strings are matrices of chars, this is also

```
s = ['At step ' int2str(k) ', p = ' num2str(p,4)]
```

Octave responds with

```
s = At step 56, p = 0.142
```

Strings

- Octave/Matlab has virtually all common string and parsing functions.
- You are encouraged to browse through the list of commands or simply type help command :

```
strcmp, strncmp, strmatch, char, ischar, findstr,
strfind, str2double, str2num, num2str, strvcat,
strtrim, strtok, upper, lower,
```

and many more...

Plotting in 2D

`plot(x,cos(x))`

Display x,y-plot
Creates automatically a figure window.
Octave uses gnuplot to handle graphics.

`figure(n)`

Create figure window 'n'
If the figure window already exists,
brings it into the foreground
(= makes it the current figure)

`figure`

Create new figure window with
identifier incremented by 1.

Several Plots

- Series of x,y-patterns: plot(x1,y1,x2,y2,...), e.g.

```
>> plot(x,cos(x),x,sin(x),x,x.^2)
```

- Add legend to plot: command legend

```
>> legend('cos(x)','sin(x)','x^2')
```

- Alternatively, hold on does the same job:

```
>> hold on; plot(x,cos(x));
>> plot(x,sin(x));
>> plot(x,x.^2);
```

Frequent Plotting Commands

`clf`	Clear figure
`hold on`	Hold axes. Don't replace plot with new plot, superimpose plots
`grid on`	Add grid lines
`grid off`	Remove grid lines
`title('Exp1')`	Set title of figure window
`xlabel('time')`	Set label of x-axis
`ylabel('prob')`	Set label of y-axis
`subplot`	Put several plot axes into figure

Controlling Axes

`axis equal`	Set equal scales for x-/y-axes
`axis square`	Force a square aspect ratio
`axis tight`	Set axes to the limits of the data
`a = axis`	Return current axis limits [xmin xmax ymin ymax]
`axis([-1 1 2 5])`	Set axis limits (freeze axes)
`axis off`	Turn off tic marks
`box on`	Adds a box to the current axes
`box off`	Removes box

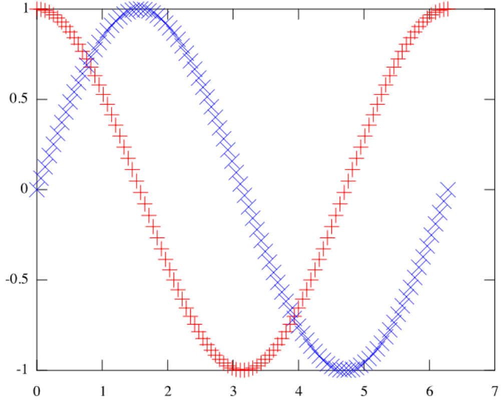
Choosing Symbols and Colors

- In `plot(x,cos(x),'r+')` the format expression ’r+’ means red cross.
- There are a number of line styles and colors, see `help plot`.
- Example:

```
>> x = linspace(0,2*pi,100);
>> plot(x,cos(x),'r+',x,sin(x),'bx');
```

produces this plot:

Plot result



```
>> plot(x,cos(x),'r+',x,sin(x),'bx');
```

Plotting parameters

- Adjusting the axes

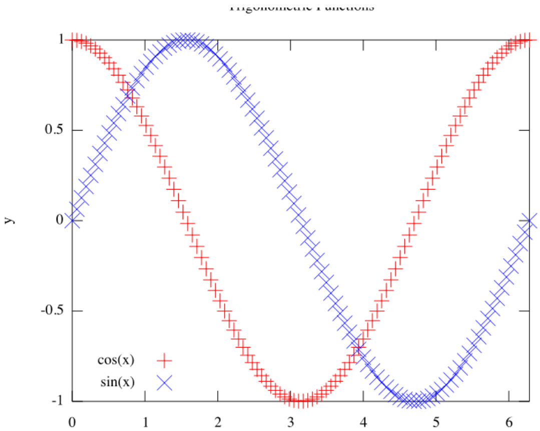
```
>> axis([0 2*pi -1 1])
```

  (try also axis tight)

- Adding a legend, labels and a title

```
>> legend('cos(x)','sin(x)','Location','Southwest')
>> title('Trigonometric Functions')
>> xlabel('x')
>> ylabel('y')
```

Plot result



```
plot(x,cos(x),'r+',x,sin(x),'bx');
```

Plotting parameters

- Controlling Color and Marker Size

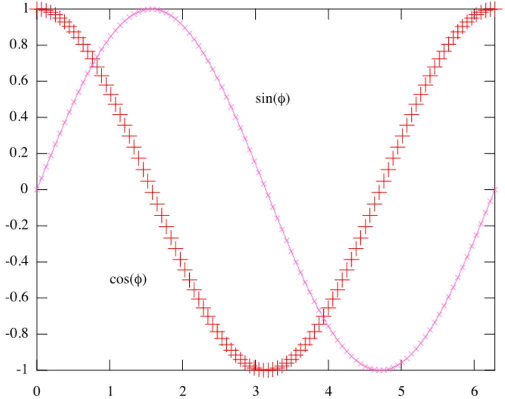
```
octave:2> plot(x,cos(x),'r+',x,sin(x),'-x',...
    'Color',[1 .4 .8],'MarkerSize',2)
octave:3> axis tight
```

- Adding Text

```
octave:4> text(1,-0.5,'cos(\phi)')
octave:5> text(3,0.5,'sin(\phi)')
```

Note the LateX syntax!

Plot result



```
plot(x,cos(x),'r+',x,sin(x),'-x','Color',[1 .4 .8],'MarkerSize',2
```

Exporting Figures

- Exporting Figures

```
print -deps myPicBW.eps         Export B/W .eps file
print -depsc myPic.eps          Export color .eps file
print -djpeg -r80 myPic.jpg     Export .jpg in 80 ppi
print -dpng -r100 myPic.png     Export .png in 100 ppi
```

- See `help print` for more devices including specialized ones for Latex.
- `print` can also be called as a function. Then, it takes arguments and options as a comma-separated list. E.g.:

```
print('-dpng','-r100','myPic.png');
```

More commands

- This tutorial cannot cover the huge variety of graphics commands in Octave/Matlab.
- You are encouraged to browse through the list of commands or simply type `help command`:

  `hist, bar, pie, area, fill, contour, quiver, scatter, compass, rose, semilogx, loglog, stem, stairs, image, imagesc`
- and many more...

3D Plots

- Plotting in 3D

`plot3`	Plot lines and points in 3d
`mesh`	3D mesh surface plot
`surf`	3D colored surface plot

- Most 2d plot commands have a 3D sibling. Check out, for example,

`bar3`, `pie3`, `fill3`, `contour3`, `quiver3`, `scatter3`, `stem3`

Programming

- Programming in Octave/Matlab is Super Easy. However, keep the following facts in mind:
  - Indices start with 1 !!!

    ```
    >> v = 1:10
    >> v(0)
    error: subscript
    ```

  - Indices must be either positive integers or logicals.
  - Octave/Matlab is case-sensitive.
- Text Editors
  - Use an editor with m-file syntax highlighting/coloring.

if-else

- if Statement

```
if condition,
    then-body;
elseif condition,
    elseif-body;
else
    else-body;
end
```

The `else` and `elseif` clauses are optional. Any number of `elseif` clauses may exist.

Switch-cases

- switch Statement

```
switch expression
    case label
        command-list;
    case label
        command-list;
    ...
    otherwise
        command-list;
end
```

- Any number of case labels are possible.

Loops statement

- Octave's `while` statement looks like this:

```
while (condition)
    body
endwhile
```

- Example

```
%% Fibonacci sequence.
fib = ones (1, 10);
i = 3;
while (i <= 10)
    fib (i) = fib (i-1) + fib (i-2);
    i++;
endwhile
```

Loops statement

- The for statement

```
for var = expression
    body
endfor
```

- Example

```
fib = ones (1, 10);
for i = 3:10
    fib(i) = fib(i-1) + fib(i-2);
endfor
```

Loops statement

- Within Octave is it also possible to iterate over matrices or cell arrays using the for statement. For example consider

```
disp ("Loop over a matrix")
for i = [1,3;2,4]
    i
endfor

disp ("Loop over a cell array")
for i = {1,"two";"three",4}
    i
endfor
```

Break Statement

- The break statement jumps out of the innermost while, do-until, or for loop that encloses it. The break statement may only be used within the body of a loop.

```
%% finds the smallest divisor and identifies prime numbers
num = 103;
div = 2;
while (div*div <= num)
    if (rem (num, div) == 0)
        break;
    endif
    div++;
endwhile
if (rem (num, div) == 0)
    printf ("Smallest divisor of %d is %d\n", num, div)
else
    printf ("%d is prime\n", num);
endif
```

Continue Statement

- The continue statement, like break, is used only inside while, do-until, or for loops. It skips over the rest of the loop body, causing the next cycle around the loop to begin immediately. Contrast this with break, which jumps out of the loop altogether. Here is an example:

```
% print elements of a vector of random integers that are even.
vec = round (rand (1, 10) * 100);
% print what we're interested in:
for x = vec
    if (rem (x, 2) != 0)
        continue;
    endif
    printf ("%d\n", x);
endfor
```

Increment Operators (Octave only!)

- Increment operators increase or decrease the value of a variable by 1.

`i++`	`Increment scalar i by 1`
`i--`	`Decrement scalar i by 1`
`A++`	`Increment all elements of matrix A by 1`
`v--`	`Decrement all elements of vector v by 1`

- There are the C/C++ equivalent operators `++i` , `--A` .

Comparison Operators

- All of comparison operators return a value of 1 if the comparison is true, or 0 if it is false. Examples: `i == 6`, `cond1 = (d > theta)`
- For the matrix-to-matrix case, the comparison is made on an element-by-element basis. Example: `[1 2; 3 4] == [1 3; 2 4]` returns `[1 0; 0 1]`
- For the matrix-to-scalar case, the scalar is compared to each element in turn. Example: `[1 2; 3 4] == 2` returns `[0 1; 0 0]`.

Comparison Operators

- Special comparison operators

```
any(v)      Returns 1 if any element of
            vector v is non-zero (e.g. 1)
all(v)      Returns 1 if all elements in
            vector v are non-zero (e.g. 1)
```

- For matrices, any and all return a row vector with elements corresponding to the columns of the matrix.

```
any(any(C))     Returns 1 if any element of
                matrix C is non-zero (e.g. 1)
all(all(C))     Returns 1 if all elements in
                matrix C are non-zero (e.g. 1)
```

Relational Operators

```
x < y        True if x is less than y
x <= y       True if x is less than or equal to y
x == y       True if x is equal to y
x >= y       True if x is greater than or equal to y
x > y        True if x is greater than y
x ~= y       True if x is not equal to y

x != y       True if x is not equal to y (Octave only)
x <> y       True if x is not equal to y (Octave only)
```

Logical operations

- Boolean Expressions

```
B1 & B2     Element-wise logical and
B1 | B2     Element-wise logical or
~B          Element-wise logical not
!B          Element-wise logical not (Octave only)
```

- Short-circuit operations: evaluate expression only as long as needed (more efficient).

```
B1 && B2    Short-circuit logical and
B1 || B2    Short-circuit logical or
```

Recommended Naming Conventions

- Underscore-separated or lowercase notation for functions Examples: `intersect_line_circle.m`, `drawrobot.m`, `calcprobability.m`
- UpperCamelCase for scripts Examples: `LocalizeRobot.m`, `MatchScan.m`
- Note: Matlab/Octave commands are all in lowercase notation (no underscores or dashes) Examples: `continue`, `int2str`, `isnumeric`

Functions

- Complicated Octave/Matlab programs can often be simplified by defining functions.
- Functions are typically defined in external files, and can be called just like built-in functions.
- In its simplest form, the definition of a function named name looks like this:

```
function name
    body
endfunction
```

- Get used to the principle to define one function per file (text files called m-file or .m-file)

Passing/return parameters

- Normally, you will want to pass/return some information to the functions you define. Simply write

```
function ret_var = name (arg_list)
    body
endfunction
```

- `arg-list` is a comma-separated list of input arguments `arg1, arg2, ...,argn`
- `ret-var` is a comma-separated list of output arguments. Note that `ret-var` is a vector enclosed in square brackets `[arg1, arg2, ...,argm]`.

Example functions

```
function [mu sigma] = calcmoments(data)
    mu = mean(data);
    sigma = std(data);
endfunction

function [haspeaks i] = findfirstpeak(data, thresh)
    indices = find(data > thresh);
    if isempty(indices),
        haspeaks = 0; i = [];
    else
        haspeaks = 1; i = indices(1);
    endelse
endfunction
```

Local Variables, Variable Number of Arguments

- Of course, all variables defined within the body of the function are local variables.

`varargin`	`Collects all input argument in a cell array Get them with varargin{i}`
`varargout`	`Collects all output argument in a cell array. Get them with varargout{i}`
`nargin`	`Get the number of input args.`
`nargout`	`Get the number of output args.`

- See `help varargin`, `help varargout` for details.

Functions and their m-File

- When putting a function into its m-file, the name of that file must be the same as the function name plus the .m extension.
- Examples: `calcmoments.m`, `findfirstpeak.m`
- To call a function, type its name without the .m extension. Example: `[bool i] = findfirstpeak(myreadings, 0.3);`
- Comments in Octave/Matlab start with % Make use of them!

Document your Function/Script

- You can add a help text to your own functions that appears upon help command.
- The first block of comment lines in the beginning of an m-file is defined to be help text.
- Example:

```
% NORMANGLE Put angle into a two-pi interval.
% AN = NORMANGLE(A,MIN) puts angle A into the interval
% [MIN..MIN+2*pi[. If A is Inf, Inf is returned.
% v.1.0, Dec. 2003, Kai Arras.

function an = normangle(a,mina);
    if a < Inf,
    [...]
```

Setting Paths

`path`	Print search path list
`addpath('dir')`	Prepend the specified directory to the path list
`rmpath('dir')`	Remove the specified directory from the path list
`savepath`	Save the current path list

References

- Octave/Matlab Tutorial, Kai Arras Social Robotics Lab. http://srl.informatik.uni-freiburg.de/downloadsdir/Octave-Matlab-Tutorial.pdf
- Numerical Methods Using MATLAB by Matthews and Fink, Pearson
- Applied Numerical Methods with MATLAB for Engineers and Scientists, Third Edition Steven C. Chapra, McGraw-Hill
- Numerical Methods in Engineering with MATLAB, Jaan Kiusalaas, Cambridge University Press

Thank you!