PROJECT for EE 483

COMMUNICATIONS SYSTEMS I - Fall 2004

Computer Assignment 1 - Get Started

Assigned: 09/09/2004   Due: 09/21/2004

Part I : Basic Operations

Exercise 1 : Vectors

Matrices and vectors make up the heart of MATLAB computations. A vector `v` that contains the numbers between `a` and `b` that are `d` apart is created using the command:

```
v = a:d:b;
```

(a) Create a vector `x` containing the numbers in the range [0,2]. Use increments of 0.02. Plot `x` using the command `plot(x)`.

To create an all-zero or all-one vector you can use the `zeros` and `ones` commands, respectively. For example, by typing

```
w = ones(1,100);
```

you create a vector containing 100 elements equal to 1.

(b) Create and plot a vector `y` containing half the number of elements as vector `x` (round up the number of elements to the closer larger integer). All the elements of `y` should be equal to 0. **Note**: You can check the length of `x` using the command `length`.

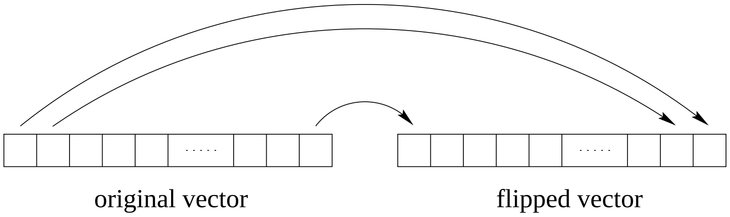
(c) Create and plot an all-one vector `z` with the same length as `y`.

Two vectors `v1` and `v2` can be concatenated by typing

```
[v1,v2];
```

(d) Create a ramp waveform by concatenating vectors `y`, `x` and `z` (in that order). Store the result in vector `s` and plot `s`.

The command `fliplr` flips a vector (see the following figure).



(e) Flip vector `s` and plot the result.

You can access specific elements of a vector `v` by typing

```
v(r);
```

where `r` is a vector containing the indices of the elements you wish to access. For example, with the commands:

```
r=1:2:length(v);
vodd=v(r);
```

we store in `vodd` the odd-numbered elements of `v`.

(f) Plot the second third of the vector created in section 1-(e).

Exercise 2 : Arithmetic expressions and functions

In MATLAB complex arithmetic expressions can be formed using the following operators:

Operator	Operation	Example
`+`	Addition	`x+y`
`-`	Subtraction	`x-y`
`*`	Multiplication w/ scalar	`2*x`
`.*`	Element-by-element multiplication	`x.*y`
`.^`	Power	`x.^2`

The functions listed below are very useful in signals and systems and are also defined in MATLAB:

Function	
`cos`	Cosine
`sin`	Sine
`tan`	Tangent
`exp`	Exponential
`abs`	Absolute value (magnitude for complex numbers)
`angle`	Phase

See also the MATLAB manual for more functions.
As an example,

```
cos(x)
```

returns a vector of the same length as `x`, containing the cosine of the elements in `x`.
A function can be plotted using the command

```
plot(x,y);
```

where `y` contains the values of the function evaluated at the data points contained in `x`.

(a) Plot the function

$$f(x) = 7\sin(x^2),$$

where x is in the range -2 to 2. Use increments of 0.01.

(b) Define

$$g(x) = 3\cos(x^3).$$

Plot the function $g(x) + f(x)$ as well as the function $f(x) - g(x)$.

Exercise 3 : Plotting

MATLAB provides several commands to customize the look of a plot. These commands include:

Command	
`title('Title Text')`	Puts `Title Text` as the plot title.
`xlabel('Label Text')`	Puts `Label Text` as the x-axis label.
`ylabel('Label Text')`	Puts `Label Text` as the y-axis label.
`axis([xl,xu,yl,yu])`	Specifies the axis limits.
	`xl` : The lower limit of the x-axis.
	`xu` : The upper limit of the x-axis.
	`yl` : The lower limit of the y-axis.
	`yu` : The upper limit of the y-axis.

(a) Plot the function $g(x)$ defined in the previous exercise. Title the plot `Plot of function g(x)`, label the x-axis and the y-axis as `x` and `g(x)`, respectively. Use the following limits for the axes:

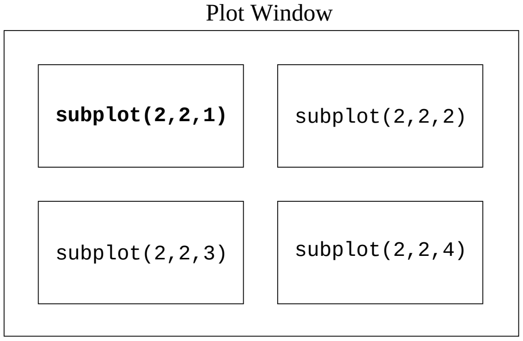
x-axis: Lower limit: -1, Upper Limit: 1.

y-axis: Lower limit: 1, Upper Limit: 4.

If you wish to have several plots shown at once on different sets of axes, use `subplot`. For example,

```
subplot(2,2,1)
```

divides the plot window into 4 parts (*subplots*) as shown in the figure and specifies the upper left part as the current subplot.



All subsequent `plot`, `title`, `xlabel`, `ylabel`, and `axis` commands will affect that subplot *only*. You may choose another subplot by executing again the `subplot` command. For example, to choose the lower left subplot you type:

```
subplot(2,2,3)
```

You can restore the original plot setting (i.e. no subplots) by typing `clf` .

(b) The complex function $f(t)$ has the form:

$$f(t) = 0.5t^2 e^{-j2\pi t}.$$

Plot (in two different subplots) the real and imaginary parts (see note below) as a function of time t, from $t = -3$ to $t = 3$ (seconds) in 0.01 second increments.

(c) Plot (in two different subplots) the magnitude and phase of f.

Note: It is very easy to enter a complex number in MATLAB. As an example, the complex number $1 + \pi j$ is entered as

```
1 + pi*j
```

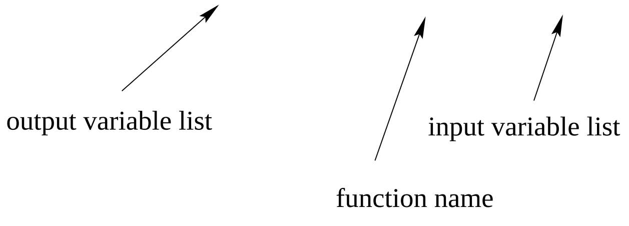
Make sure that you *do not* assign a new value to the predefined variables `j` and `pi`. The functions `real` and `imag` give the real and the imaginary part of a complex number.

Part II : User defined functions,M-Files and Program Control

Exercise 4 : User defined functions

MATLAB allows the user to create his own functions. To create a function `newfun` a text file must be created with the same filename as the function and the extension `.m`, i.e. `newfun.m`. The first line of this file has the form:

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



We note that any number of input and output variables can be used. As an example, the following file `oddeven.m` defines the function `oddeven` that returns the odd-numbered and even-numbered elements of an arbitrary length vector.

```
function [oddels,evenels]=oddeven(v)

oddels=v(1:2:length(v));
evenels=v(2:2:length(v));
```

(a) *Sinusoidal waveform generator.*
Let `sinegen` be a function whose output is a sine wave $y(t) = \sin(2\pi ft)$. The inputs to the function should be the frequency f and the time vector t. Generate the `sinegen` function (i.e. the `sinegen.m` file) and plot the function with t ranging from 0 to 0.5 in 0.001 increments, and a frequency of $25Hz$.

(b) Let `ecos` be a function whose output is $y(t) = e^{-\pi t}\cos(50\pi t)$. Generate the `ecos` function and plot the function, with t ranging from 0 to 0.5 in 0.001 increments.

Exercise 5 : M-Files and Program Control

Lists of MATLAB functions and commands can be stored in a single file with the extension `.m`. These files are called *M-Files.* The commands stored in an M-File can be executed by typing the filename (without the `.m` extension) at the command prompt. As an example, the M-File `expon.m`:

```
t=-2:0.01:2;
y=exp(-abs(t));
plot(t,y);
```

plots the function $y(t) = e^{-|t|}$ with t ranging from -2 to 2 in 0.01 increments, when the user types `expon` at the command prompt. Note, that a file that defines a new function is a special case of an M-File.

(a) Create an M-File that plots the output of the sine wave generator for $f = 10Hz$ and $f = 15Hz$. Each output should be plotted in a different subplot with appropriate titles and axis labels.

As in all high level languages MATLAB includes an *if-then-else* structure that allows a group of commands to be executed when a certain condition is satisfied. For example, the following statements in an M-File will increase the variable `x` by 1 if it is positive and decrease it if it is negative:

```
if x<0
   x=x-1;
else
   x=x+1;
end
```

MATLAB includes several relational operators. For example:

Operator	
==	Equal
~=	Not equal
<=	Less than or equal
>	Greater than
<	Less than
>=	Greater than or equal

The logical operators defined in MATLAB include:

Operator	
|	OR
&	AND
~	NOT

A loop for repeating the same task for different parameter values can be created using the `for` statement. For example, let a difference equation be given as follows:

$$x_n = 0.5x_{n-1} + x_{n-2}, n = 3, \ldots, 10 \text{ and } x_1 = 0.25,\ x_2 = 0.5,$$

Then we may use the following M-File to calculate the first 10 values of the sequence x_n:

```
x=zeros(1,10);   % We initialize the vector x.
x(1)=0.25;
x(2)=0.5;        % We state the initial conditions.
for I=3:10
    x(I)=0.5*x(I-1)+x(I-2);
end
```

Loops for repeating the same task while a certain condition is satisfied can be created using the `while` statement. As an example, the function `firstneg` defined below returns either the position of the first negative element of the input vector or 0 if there is no negative element:

```
function p=firstneg(x)

p=1;
while p~=length(x)+1 & x(p)>=0
      p=p+1;
end
if p>length(x)
   p=0;
end
```

(b) Create a function **meancomp** that compares the mean value of the odd numbered elements with the mean value of the even numbered elements and returns:

0, if they are equal,
1, if the mean of the odd numbered elements is greater,
2, if the mean of the even numbered elements is greater.

Note

Your report should include all the plots you are asked to create in exercises 1,2, 3, and all the plots and M-files you are asked to create in exercises 4 and 5.