UNIVERSITY COLLEGE DUBLIN

SCHOOL OF ELECTRICAL, ELECTRONIC & COMMUNICATION ENG.

EEEN30110 SIGNALS AND SYSTEMS

Experiment 3SS0

INTRODUCTION TO MATLAB

1. Objective:

To introduce Matlab as a tool for the analysis of signals and systems.

2. Background Information:

The numerical package to be employed in this module is **Matlab**. We introduce some elementary Matlab commands in this experiment. Matlab version 7.1 is described but the majority of described commands are the same in earlier and later versions. Matlab has hundreds of useful commands so the discussion below is far from complete. To enter a command in the **command window** type the command after the **>>** prompt and press the return key. Matlab will immediately execute the command and return the results.

1. *Help*:

Matlab has a simple in-built help facility. Type **help function_name** at the command prompt to get information on the function **function_name**. For example, typing **help roots** gives information on the Matlab function **roots** and typing **help eig** gives information on the Matlab function **eig**. Of course this help facility is only helpful if one knows the relevant function name. If you wish to see the actual Matlab code for a function the command is **type function_name**. So, for example, **type roots** gives a full list of the commands in the Matlab function roots. You will see that the function help file occurs at the start of the code, but unless you are already familiar with Matlab you will probably not derive any further useful information from this command. The type command will give no information on the very basic functions.

Test Problem: **log2** is a Matlab command. What does it do ?

2. *Basic Commands*:

Matlab can act like a scientific calculator (albeit far more powerful than any conventional scientific calculator).

Multiply 3 by 4:
```
>> 3*4
```

Divide 3 by 4:
```
>> 3/4
```

Add 3 and 4:
```
>> 3+4
```

Subtract 4 from 3:

```
>> 3-4
```

Raise 3 to the power of 4:

```
>> 3^4
```

Extract 4th root of 3:

```
>> 3^0.25
```

Extract square root of 3:

```
>> 3^0.5
```

or

```
>> sqrt(3)
```

Test Problem: Evaluate $\dfrac{-6 \pm \sqrt{6^2 - (4 \times 8)}}{2}$.

Most mathematical functions have the obvious names. For example sine has the Matlab name **sin**. A very incomplete list:

Function Name/Mathematical Symbol	Matlab Command Name
sine sin	**sin**
cosine cos	**cos**
tangent tan	**tan**
arcsine $\sin^{-1}$	**asin**
arccosine $\cos^{-1}$	**acos**
arctangent $\tan^{-1}$	**atan**
hyperbolic sine sinh	**sinh**
hyperbolic cosine cosh	**cosh**
hyperbolic tangent tanh	**tanh**
arc-hyperbolic sine $\sinh^{-1}$	**asinh**
arc-hyperbolic cosine $\cosh^{-1}$	**acosh**
arc-hyperbolic tangent $\tanh^{-1}$	**atanh**
exponential exp	**exp**
natural logarithm ln or $\log_e$	**log**
logarithm to the base 10 $\log_{10}$	**log10**
modulus $\lvert\ \rvert$	**abs**
argument Arg	**angle**
complex conjugate *	**conj**

If a function has an input argument (as the vast majority do) it must follow the function name in round brackets. For example:

```
>> sin(3.1)
```

The majority of functions also have output arguments. These may be assigned to named variables. For example:

```
>> a=sin(3.1)
```

```
a =

   0.0416
```

assigns the value 0.0416 to variable a. This variable will continue to have this value until some new value is assigned to it or the Matlab session is terminated. Functions will accept variables as inputs provided those variables have previously had values assigned to them. For example:

```
>> a^2

ans =

   0.0017
```

Test Problem: Evaluate $\tanh^{-1}\left(\sqrt{\ln\left(\frac{6}{5}\right)}\right)$

By default Matlab works to quite a large number of decimal places but only shows the first four when printing to the screen. If desired you can change the number of decimal digits printed by using the **format** command. Type **help format** for more information. Matlab stores a number of the decimal digits of the mathematical number π under the variable name **pi**. To find the value of a variable just type the variable name at the command prompt and press the return key.

Test Problem: Find the value of π as stored by Matlab to 14 decimal digits. Compare with the well-known rational approximation $\frac{22}{7}$. To how many digits do the two agree ? Compare with the somewhat less well known rational approximation $\frac{355}{113}$. How many digits of agreement are there in this case ?

You may suppress printing to the screen by using the semi-colon command. For example:

```
>> sin(3.1)

ans =

   0.0416
```

Absent the semi-colon Matlab prints to the screen.

```
>>  sin(3.1);
>>
```

With the semi-colon present no printing to the screen occurs. Of course this also means that you do not know the answer. Suppression of printing only makes sense when you redirect the output of the function to a variable. For example:

```
>> b=sin(3.1);
>> b

b =
   0.0416
```

A variable “b” is created containing the answer sin(3.1) which previously was directed to the temporary, default variable “ans”. To find out the value of the variable b, just type b and hit the return key.

Matlab treats complex numbers no differently from real numbers. You may create a complex variable in Matlab thus:

```
>>  z=1+2i

z =

  1.0000 + 2.0000i
```

By default Matlab uses the variable name i to refer to the square-root of -1. The vast majority of Matlab functions will accept complex input. For example:

```
>>  sin(z)

ans =

  3.1658 + 1.9596i
```

Test Problem: Evaluate $e^{\pi i}$ and $\tanh^{-1}\left(\sqrt{\dfrac{\ln(4)-\exp(-2)}{\sin(6)}}\right)$

3. *Matrices*

The principle advantage of Matlab lies in its ability to handle matrices. The simplest method of creating a matrix variable in Matlab is to list the elements. The list must be subtended by square brackets. Individual elements are separated by spaces and rows are separated by semi-colons. For example:

```
>> A=[1 4 -2;9 0 5;-8 7 -1]

A =

    1    4   -2
    9    0    5
   -8    7   -1
```

It is convenient to be able to generate certain special matrices without having to use the list method. The Matlab commands **eye**, **ones** and **zeros** are helpful in this respect. Use the help command for more information.

Having created a matrix variable (and assuming a square matrix) single commands in Matlab allow one to invert the matrix (**inv**), determine its eigenvalues and eigenvectors (**eig**) and calculate its determinant (**det**). For example:

```
>>  inv(A)

ans =

    0.1228    0.0351   -0.0702
    0.1088    0.0596    0.0807
   -0.2211    0.1368    0.1263

>>  det(A)

ans =

  -285
```

The **eig** command can be used in two ways. One generates the eigenvalues only. For example:

```
>>  eig(A)

ans =

  -10.7061
    3.9266
    6.7795
```

The second generates both the eigenvalues and the corresponding eigenvectors. It is invoked by collecting the output of the function using two output arguments. For example:

```
>>  [V,D]=eig(A)

V =

    0.3321   -0.4407    0.4935
   -0.6137    0.1222    0.8349
    0.7163    0.8893    0.2438


D =

  -10.7061         0         0
         0    3.9266         0
         0         0    6.7795
```

One does not have to call the output arguments V and D. The important point is that there must be two of them. For example invoking the **eig** command with only one output argument

```
>>  D=eig(A)

D =

  -10.7061
    3.9266
    6.7795
```

has reverted back to the same output data as above (i.e. the eigenvalues only) when no output argument was employed. All that has changed is that the output data has been placed in the variable D rather than being placed in the temporary, default variable ans.

Given two matrix variables A and B one may add them **A+B**, subtract them **A-B**, matrix-multiply them **A*B**, right matrix divide them **A/B** (i.e. form the product A*B^{-1}) or left matrix divide them **A\B** (i.e. form the product A^{-1}*B). One may matrix-multiply A by itself in either of two ways **A*A** or **A^2**. One may find the transpose of A, **A'**. One may multiply matrix A by real number r, **r*A**.

Test Problem: Find the inverse, the determinant, the eigenvalues and the corresponding eigenvectors of the matrix $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & -4 & -2 \end{bmatrix}$. Present the inverse as a matrix with rational elements by using the **format** command.

Test Problem: Using either the matrix inverse command, **inv**, or the left matrix divide command, \ , solve the following system of equations

$$x + 2y = 1$$
$$-2x + y + z = 0$$
$$3x + 2y + 4z = -3$$

presenting the solution as rational numbers.

4. *Signals and basic graphics*:

In the Matlab command window type

```
>> t = [0:0.01:9.99];
```

This generates the vector t of 1000 elements (illustrating another method of generating matrix variables of a special kind). The first element is 0. The last element is 9.99. Each element is 0.01 above its predecessor. As noted before, the semi-colon at the end prevents Matlab from echoing the 1000 elements to the screen, useful in this case when there is a lot of data.

The command

```
>> length(t)

ans =

  1000
```

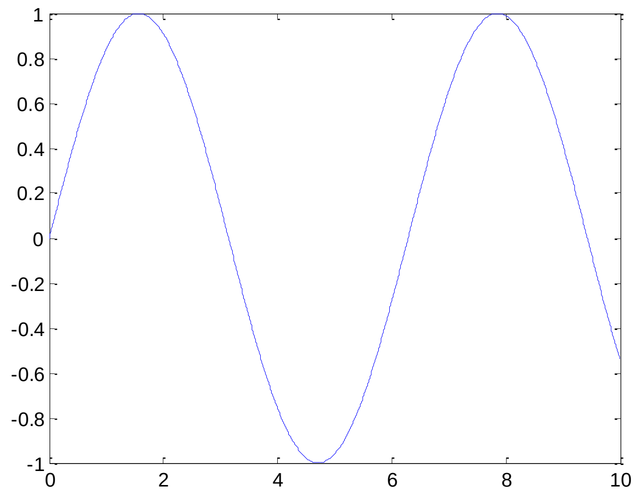
yields the number of elements as 1000.

The majority of basic Matlab functions will accept input arguments which are matrices. So, for example, the command

```
>> f = sin(t);
```

generates a vector of 1000 elements, each element being the sine of the corresponding element of the vector t. We thereby obtain 1000 samples of the sinusoidal signal sin(t), uniformly distributed over the time interval [0,0.99]. Note that Matlab has generated all 1000 elements of vector f with one command and without employing a loop. To see what we have achieved we can plot the 1000 samples of the sinusoidal signal

```
>> plot(t,f)
```



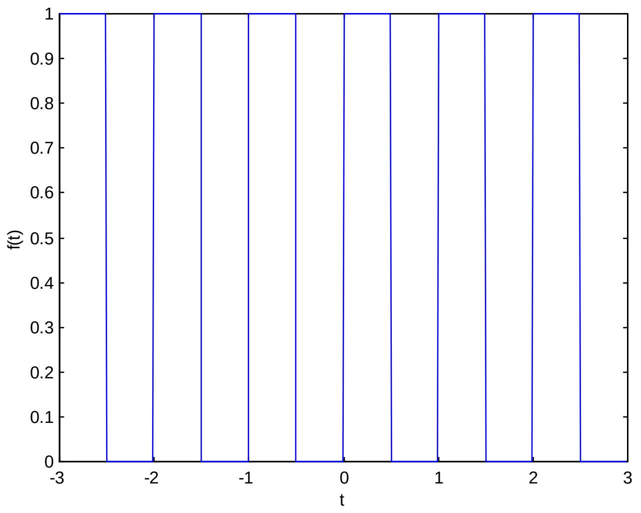
giving 1000 points (t,sin(t)) joined by lines. The period of sin(t) is 2π which is approximately 6.28 so the time interval of 9.99 sec encompasses a little over one and a half cycles.

All signals are represented in Matlab as vectors made up of uniformly-spaced samples of the signal, i.e. the signal evaluated at specific times (sample times) which are equally spaced.

Consider the periodic continuous-time signal with period of 1 sec defined by

$$f(t) = \begin{cases} 1 & 0 \le t < 0.5 \\ 0 & 0.5 \le t < 1 \end{cases}$$

over one period. A few (actually six) cycles of this signal appear as follows:



There is no standard Matlab function which can be used to directly evaluate this signal. We must take a different approach to above. To start let us try to generate one cycle.

```
>> f = [ones(1,50) zeros(1,50)];
```

You can employ the commands **help ones** and **help zeros** to understand this step if it is not intuitively clear. We have thereby generated 100 samples of one period of the signal. One way (and certainly not the best way) to generate six cycles is as follows:

```
>> f = [f f f f f f];
```

which simply replicates the cycle six times. To generate the vector of sample times

```
>> t = [-3:0.01:2.99];
```

The gap between sample times is chosen as 0.01 since the gap must be this value if 100 sample times are to fit uniformly spaced between 0 and 0.99. The vector t is now a vector of 600 times uniformly spaced between -3 and 2.99 while the vector f is the vector of values of the signal f (the square-wave) evaluated at these sample times. To see what we have achieved we can plot the 600 samples of the square-wave signal

```
>> plot(t,f)
```

This gives the same plot as above except for axis labels. If you wish to add axis labels or a title the relevant commands are **xlabel**, **ylabel**, and **title**. You can use the standard help method to get more information on the syntax and use of these commands (e.g. **help xlabel** for information on **xlabel**).

Before completing this section we consider one more example of signal generation. Suppose we have a periodic signal f(t) with period 2 given by:

$$f(t) = -t^3 \quad -1 \le t \le 1$$

over one period. To acquire a Matlab representation of this signal we take the second approach considered above and seek firstly a representation of just one cycle.

```
>> t = [-1:0.01:0.99];
```

generates 200 sample times uniformly distributed between -1 and 0.99.

```
>> f = -t.^3
```

This generates the signal over one cycle. The command .^ denotes term by term power. Specifically here .^3 denotes term by term cube. Similarly .* denotes term by term multiplication and ./ denotes term by term division. To understand these *term by term* arithmetical commands consider the following examples:

```
>> P = [1 4 6];
>> Q = [1 3 9];
>> P.*Q

ans =

    1    12    54
```

The first term is the first term of P multiplied by the first term of Q, etc.

```
>> P./Q
```

```
ans =

   1.0000    1.3333    0.6667
```

The first term is the first term of P divided by the first term of Q, etc.

```
>> P.^2

ans =

    1    16    36
```

The first term is the first term of P raised to the power of 2, etc.

To generate six cycles of the signal f above the command

```
>> f = [f f f f f f];
```

will do (although it is not the most elegant way).

Test Problem: Create Matlab variables t and f which provide a Matlab representation of four cycles of the signal $f(t)$ with period 3 sec given by $f(t) = 2t - \frac{1}{2}t^2$, $-1 \le t \le 2$. Your representation should include 150 samples per cycle. Plot the four cycles of the signal. Using the Matlab commands **size** and **length** find the size and length of the variables t and f. With reference to the help files for these commands explain why the numbers attained are to be expected.

5. *Polynomials*:

Matlab can store a polynomial as a vector. For example in Matlab the vector [1 3 4 2] can represent the polynomial $s^3 + 3s^2 + 4s + 2$ (note the order of decreasing powers of s). Of course it can also just represent a vector, so as the user you must be careful to keep clear in your own mind how a vector is to be interpreted. One is commonly interested in the roots (i.e. the zeros) of a polynomial. The Matlab command **roots** permits the ready evaluation of the roots of a polynomial. For example:

```
>> roots([1 3 4 2])

ans =

     -1      +    1i
     -1      -   1i
     -1
```

gives the roots of polynomial $s^3 + 3s^2 + 4s + 2$.

One may wish to add and subtract polynomials or to multiply them by constants. Convince yourself that the normal matrix addition, subtraction and scalar multiplication commands achieve this. You may also wish to multiply two polynomials. This is a little more sophisticated and requires a special command **conv** (short for convolution). For example the algebraic multiplication

$$\left(s^3 + 3s^2 + 4s + 2\right)\left(3s^2 - s + 1\right) = 3s^5 + 8s^4 + 10s^3 + 5s^2 + 2s + 2$$

is achieved in Matlab by

```
>> >> conv([1 3 4 2],[3 -1 1])

ans =

     3     8    10     5     2     2
```

Associated with any square matrix there is a special polynomial called the characteristic polynomial whose roots are the eigenvalues of the matrix. The Matlab command **poly** permits evaluation of the characteristic polynomial of a matrix. For example, using the matrix from above:

```
>>  A=[1 4 -2;9 0 5;-8 7 -1]

A =

     1     4    -2
     9     0     5
    -8     7    -1

>> poly(A)

ans =

   1.0000   -0.0000  -88.0000  285.0000
```

Interpreting Matlab syntax we conclude that the characteristic polynomial is $s^3 - 88s + 285$. In Mathematics lectures it is more likely that you would have chosen the variable λ and given the answer as $\lambda^3 - 88\lambda + 285$ but of course this is a notational, not a logical change. Matlab of course does not choose any name for the variable, it simply records the coefficients.

Test Problem: Find the roots of $s^7 + 28s^6 + \frac{643}{2}s^5 + \frac{8143}{4}s^4 + \frac{126085}{16}s^3 + \frac{308975}{16}s^2 + \frac{217875}{8}s + \frac{28125}{2}$.

Test Problem: Find the sum and product of the polynomials $s^3 + 5s - 1$ and $2s^4 - 9s^3 + 4s$.

You will probably come across a problem in forming the sum of the polynomials. Matlab can only add matrices of equal dimensions and in the special case of vectors this means that it can only add vectors of equal length. There is, however, a simple way around this problem which is already hinted at if you have correctly entered the polynomials in Matlab notation. Can you see it ?

Test Problem: Find the characteristic polynomial of the matrix $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -3 & -4 & -2 \end{bmatrix}$. Find the roots of this polynomial and confirm that they equal the eigenvalues as previously calculated.

6. *Rational Polynomials*:

Matlab can represent a rational polynomial simply as a pair of vectors, one representing the numerator and one the denominator. For example the rational polynomial

$$H(s) = \frac{1}{s^2 + 0.1s + 1} = \frac{N(s)}{D(s)}$$

can be recorded in Matlab as

```
>> N = [1];
>> D = [1 0.1 1];
```

Rational polynomials will occur often in this module. It will happen that it is of interest to make the special choice of complex variable $s = i\omega$ (where i is the square root of -1 (Matlab's default interpretation of variable i) and ω is real) and then to plot $|H(i\omega)|$ vs ω. To achieve this here we must first generate a vector of values of ω at which we would like to evaluate the rational polynomial.

```
>> w = [0:0.1:5];
```

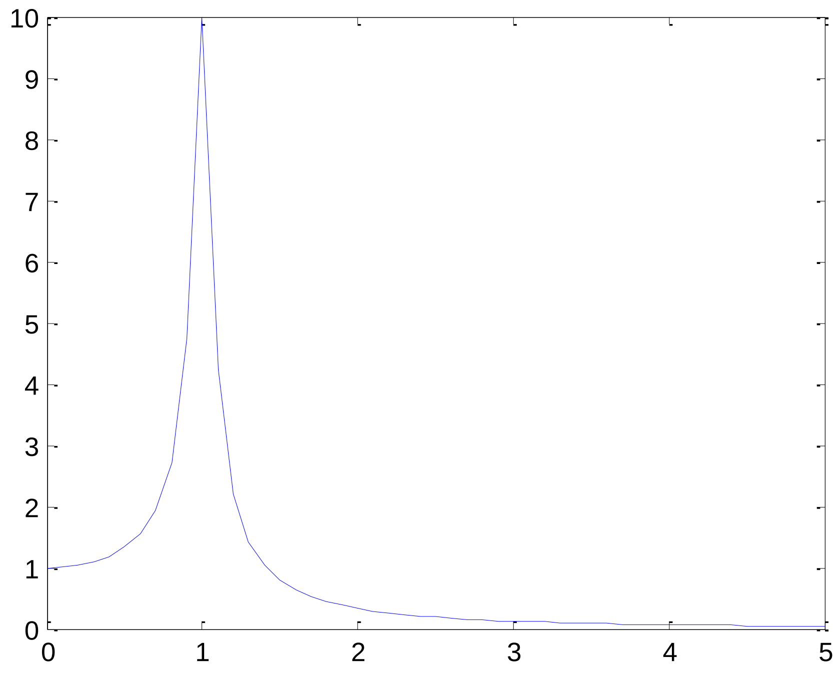
generates uniformly spaced values from 0 to 5.

```
>> H = polyval(N,i*w)./polyval(D,i*w);
```

The command **polyval(P,v)** evaluates the polynomial P (stored in Matlab notation) at all of the elements of the vector v. The command ***** is multiplication (in this case by i). The command ./ is the "term by term division" considered before. The command above therefore computes the vector $H(i\omega)$ for each of the uniformly-spaced values of ω between 0 and 5 included in the vector w. Finally the command

```
>> plot(w,abs(H))
```

produces the graph



which will be considered in lectures. This series of commands has permitted us to plot what we will call the "frequency response" of a system.

There is a much more sophisticated way that Matlab can represent a rational polynomial. This is to create a *system variable*. It is achieved by the command **tf**.

```
>> System = tf(N,D)

Transfer function:
      1
---------------
s^2 + 0.1 s + 1
```

Matlab has called the rational polynomial a "transfer function" and the command name is **tf** (standing for transfer function) for reasons which the module notes will clarify. Given a rational polynomial $H(s) = {}^{N(s)}\!/_{D(s)}$ the roots of the numerator polynomial *N* are called the zeros and the roots of the denominator polynomial *D* are called the poles. They may be found using the **roots** command of course. But if the rational polynomial is stored as a system variable it is easier to find them by using the **zero** and **pole** commands respectively.

Test Problem: Create a system variable *H* to represent the rational polynomial $\frac{2s^2+5s+1}{s^3+7s+4}$. Find the zeros and the poles of this rational polynomial and plot the frequency response.

Every rational polynomial can be represented by a partial fraction expansion. The command in Matlab which numerically determines the partial fraction expansion of a rational polynomial is **residue**. Consider rational polynomial:

$$F(s) = \frac{s^2-6}{s^3+4s^2+3s}$$

Firstly we must introduce this rational polynomial to Matlab as a pair of polynomials (numerator and denominator) using Matlab notation:

```
>> N = [1 0 -6]
>> D = [1 4 3 0]
```

Although it is unnecessary, if one wishes one may create an associated system variable:

```
>> Sys_example = tf(N,D)

Transfer function:
    s^2 - 6
-----------------
s^3 + 4 s^2 + 3 s
```

This confirms that no mistake was made in entering polynomials N and D.

Now enter the command:

```
>> [R, P, K] = residue(N,D)

R =
  0.5000
  2.5000
 -2.0000


P =

  -3
  -1
   0


K =

   []
```

Note in particular how the output of the command is to be collected. The vector R is the vector of *residues*. The vector P is the vector of corresponding poles. The vector K is the *direct term*, a polynomial stored in Matlab notation as a vector. This term is non-existent in this case. The values for R, P and K returned by **residue** are interpreted as saying that the partial fraction expansion for $F(s)$ is given by:

$$F(s) = \frac{0.5}{s+3} + \frac{2.5}{s+1} + \frac{-2}{s}$$

Partial fraction expansions will be covered in the module notes. You may use the help command to find out more about Matlab's **residue** command.

Test Problem: Find the partial fraction expansion of the rational polynomial $F(s) = \frac{5s^3 - 6s - 3}{s^3(s+1)^2}$.

7. *Managing Matlab Plots*:
Copy Matlab figure to clipboard as follows: **Matlab Figure Window > Edit Menu > Copy Figure**. The figure will take up considerable space. To reduce space save in Windows Metafile format (.wmf) rather than bitmap (.bmp) as follows: **Matlab Figure Window > Edit Menu > Copy Options > Metafile**. Once copied to the clipboard the figure may be pasted into word with the **Paste** command.