Signals & Systems Lab.- Manual (2)

A brief overview of:

**Signal Analysis using MATLAB**

By: Eng. Moustafa Adly
Contents

1. Basic signals ............................................................................................................................. 2
   1.1. Unit step ............................................................................................................................ 2
   1.2. Unit impulse ....................................................................................................................... 4

2. Operations on signals .............................................................................................................. 5
   2.1. Addition & Subtraction ....................................................................................................... 5
   2.2. Multiplication & Division .................................................................................................. 6
   2.3. Maximum & minimum ....................................................................................................... 7

3. Energy and power of signals .................................................................................................. 7

4. Transformation of the independent variable .......................................................................... 8
   ➢ Shifting (Delay & Advance)
   ➢ Reversing
   ➢ Scaling

5. Convolution & Deconvolution ............................................................................................... 10

6. Fourier Series Representation ............................................................................................... 10

7. Fourier Transform .................................................................................................................. 12

8. Image processing ................................................................................................................... 13
1. Basic Signals

1.1. Unit Step

The unit step function $u(t)$ is basically a mathematical function that is defined by:

$$u(t) = \begin{cases} 
1 & , t > 0 \\
0 & , t < 0 
\end{cases}$$

So, it is clear that the function is undefined at zero because of its discontinuity. The unit step function is defined in MATLAB as follows:

```matlab
>> t=-10:0.01:10; % step is small enough to represent continuous-time signal
>> f=heaviside(t); % the unit step function.
>> plot(t,f)
```

The heaviside function is defined easily such that it can shifted or reversed as follows:

```matlab
>> g=heaviside(t-3);
>> figure(1)
>> plot(t,g)
>> axis([-15 15 -1 2])

>> h=heaviside(t+4);
>> figure(2)
>> plot(t,h)
>> axis([-15 15 -1 2])

>> k=heaviside(-t);
>> figure(3)
>> plot(t,k)
>> axis([-15 15 -1 2])

>> l=heaviside(-2*t+2);
>> figure(4)
>> plot(t,l)
>> axis([-15 15 -1 2])
```
The discrete-time unit step function is defined as follows:

\[ u[n] = \begin{cases} 
1 & , n \geq 0 \\
0 & , n < 0 
\end{cases} \]

While, MATLAB uses the same continuous-time expression of \( u(t) \) for discrete-time unit step function causing an error in the definition of \( u[n] \) in MATLAB.

```matlab
>> n=-10:10;
>> figure(1)
>> f=heaviside(n);
% note that it is not defined at n=0 (mistake)
>> stem(n,f)
>> axis([-15 15 -1 2])
```

So, you should take care while using MATLAB with discrete time unit step functions. It is also possible to do shifting, reversing, and scaling for the discrete-time unit step function as shown in the following example.

```matlab
>> n=-10:10;
>> f1=heaviside(n+3);
>> f2=heaviside(-n+3);
>> f3=0.5.^n;
>> f4=f1.*f2.*f3;
>> subplot(221)
>> stem(n,f1)
>> grid
>> title('u[n+3]');
>> subplot(222)
>> stem(n,f2)
>> grid
>> title('u[-n+3]');
>> subplot(223)
>> stem(n,f3)
>> grid
>> title('u[n+3]*u[-n+3]');
>> subplot(224)
>> stem(n,f4)
>> grid
>> title('(0.5)^n*u[n+3]*u[-n+3]');
```
1.2. Unit Impulse

The unit impulse function is not a function in the strict sense mean while it is very useful in the mathematical analysis of signals and specially signal which has discontinuities or sudden changes.

It is defined as:

\[ \delta(t) = \begin{cases} \text{undefined(\infty)} & , t = 0 \\ 0 & , t \neq 0 \end{cases} \]

We usually care for the area of the impulse delta or dirac function as its width is usually \( \frac{1}{\Delta} \) and its height is \( \Delta \) where \( \Delta \) tends to zero keeping the area equal to 1.

MATLAB defines the delta function in the same manner, so you can’t see its value at \( t=0 \), also you can do shifting and reversing on it as in the following example:

\[
\begin{align*}
& >> t=-10:0.01:10; \\
& >> f=dirac(-t+5); \\
& >> plot(t,f)
\end{align*}
\]

The problem occurs in the definition of MATLAB for the discrete-time delta function as it uses the same continuous-time expression while the discrete-time delta is defined as follows:

\[
\delta[n] = \begin{cases} 1 & , t = 0 \\ 0 & , t \neq 0 \end{cases}
\]

And you can check that using the following code:

\[
\begin{align*}
& >> n=-10:10; \\
& >> f=dirac(-n+2); \\
& >> stem(n,f)
\end{align*}
\]

The most important properties of the delta function are the multiplication and convolution with other signals. The delta function can be multiplied by any other signal as follows:

\[
x(t) \times \delta(t - t_o) = x(t_o) \times \delta(t - t_o)
\]

\[
\int_{-\infty}^{\infty} x(t) \times \delta(t - t_o) dt = x(t_o)
\]
So, let’s try this example:

```matlab
>> syms t;
>> x=2*sin(t);
>> f=x.*dirac(t-pi/2)           % see the value of f.
>> g=int(f,t,-inf,inf)
```

Try also the following example:

```matlab
>> syms t;
>> f=heaviside(t-5);
>> diff(f,t)
>> g=dirac(t+2);
>> int(g,t)
>> int(g,t,-inf,inf)
```

The last example ensures that the delta function is the derivative of the unit step function and hence the integration of the delta function from –ve infinity till t leads to the unit step function while integrating the delta function from –ve infinity to +ve infinity is always equal to one which is the area under its curve.

### 2. Operations Signals

#### 2.1. Addition & Subtraction

Signals can be added and subtracted on condition which is that they be of the same length when defined in MATLAB in order to add or subtract the adjacent elements or values of the signals.

```matlab
>> n=-10:10;
>> x1= 0.5.^n;
>> x2= 0.8.^n;
>> x=x1+x2;
>> y=x1-x2;
>> subplot(221)
```
2.2. Multiplication & Division

Signals can be multiplied and divided on condition that they are of the same length. You should be aware not to divide by zero. Finally, it is very important to remind you of the `dot operator` that is used in vectors to access element by element not the whole matrix or vector because our signals will be horizontal vectors in most of cases.

```matlab
>> n=-10:10;
>> x1 = 0.6.^n;
>> x2 = 0.2.^n;
>> x = x1.*x2;
>> y = x1./x2;
>> subplot(221)
>> stem(n,x1)
>> subplot(222)
>> stem(n,x2)
>> subplot(223)
>> stem(n,x)
>> subplot(224)
>> stem(n,y)
```
2.3. Maximum & Minimum

You can get the minimum and the maximum of any signal simply using the `min` and `max` instructions. You can also search for all minimums and maximums using the `find` instruction as shown in the following example.

```matlab
>> n=-20:20;
>> x=cos(pi*n/4);
>> stem(n,x)
>> hold
>> xmin=min(x);
>> xmax=max(x);
>> pmin=find(x==xmin);
>> pmax=find(x==xmax);
>> ymin=x(pmin);
>> ymax=x(pmax);
>> stem(n(pmin),ymin,'filled')
>> stem(n(pmax),ymax,'filled')
```

3. Energy and power of signals

For any discrete-time signal you can calculate the average energy and average power using the following formulas:

\[
E_{av} = \sum_{N_1}^{N_2} |x[n]|^2 \\
P_{av} = \frac{1}{N_2 - N_1 + 1} \sum_{N_1}^{N_2} |x[n]|^2
\]
The values of $N_1$ & $N_2$ can be determined from the signal itself. For calculating the total energy and power you should extend $N_1$ & $N_2$ to infinity. Due to the limitations of computer simulations’ programs, we would use the following code for calculating the energy and power.

```
>> n=-10:10;
>> x=(0.5).^n.*heaviside(n);
>> E=sum(abs(x).^2)
>> Pav=E/length(x)
```

Another method for calculating the energy is as follows:

```
>> E=x.*(x.')
```

Task: Try to take a signal from a user and calculate its energy and power.

### 4. Transformation of the independent variable

It is now desired to focus on the transformation of the independent variable while shifting, reversing, and scaling. It will be covered first for the unit step functions and then it will be left for you as a task to do it for any general signal $x[n]$.

```
>> n=-10:10;
>> x=heaviside(n-1);
>> y=heaviside(n+2);
>> z=heaviside(-n+3);
>> k=heaviside(-n-2);
>> l=heaviside(2*n-1);
>> m=heaviside(0.5*n+1);
>> h=heaviside(-0.5*n-2);
>> g=heaviside(-2*n+3);
>> subplot(421)
>> stem(n,x)
>> subplot(422)
>> stem(n,y)
```
Problem:

A discrete time signal $x[n]$ is defined as follows:

$$x[n] = \begin{cases} 
\alpha^n & , \quad N1 \leq n \leq N2 \\
0 & , \quad \text{else where}
\end{cases}$$

It's required to develop a code that is responsible of doing the following:

1) Taking the values of $\alpha$, $N1$, and $N2$ from the user.
2) Plotting this signal in a reasonable range.
3) Calculating the energy and average power of the signal.
4) Plotting $x[an+b]$ in another figure, where $a$ is +1 or -1 according to the user choice and $b$ is the value of the shift specified also by the user.
5) Scaling the signal $x[n]$ and plotting the new scaled signal $x[cn]$, where $c$ is the scaling ratio ($c$ may be less than one or greater than one)

Finally, the code should plot all signals in one graph showing all transformations and indicating all signals’ titles.
5. Convolution

Convoluting two signals is very simple using MATLAB as follows. If it is required to convolute any two signals, you can use the `conv` instruction directly but you should care for the limits of the independent variable of the result as it will be the sum of the length of the independent variable of the convoluted signals.

```matlab
>> n1=-3:3;
>> x1=ones(1,length(n1));
>> n2=-2:2;
>> x2=(0.5).^n2;
>> y=conv(x1,x2);
>> n3=-5:5;
>> subplot(311)
>> stem(n1,x1)
>> subplot(312)
>> stem(n2,x2)
>> subplot(313)
>> stem(n3,y)
```

Note that the reverse operation is called deconvolution in which you provide the result of the convolution and one of the convoluted signals and will return the other convoluted signal.

Type:

```matlab
>> help deconv
```

6. Fourier Series Representation

MATLAB can be used for the calculation of the fourier series coefficient of any discrete-time signal using a predefined function called `fast fourier transform` or by developing a code based on the basic definition of the discrete-time Fourier series.
The formula used for calculating the discrete-time fourier series coefficient is:

\[ a_k = \frac{1}{N} \sum_{n=N}^{n=N} x[n] e^{-j k \omega_o n} \]

Where: \( N \) is the period of the signal \( x[n] \) and \( \omega_o = \frac{2\pi}{N} \).

We will start by calculating the fourier series coefficient using a function that we will develop based on the definition of the fourier series coefficient.

On MATLAB, type:

\[ \text{>> edit FScoeff} \quad \% \text{we will call the function FScoeff} \]

On the open editor type the following code:

```matlab
function [a]=FScoeff(x) %This function calculates the fourier series coeff of any signal. %User should define one period of the signal and call the %function for it. N=length(x); % the period of the signal x[n] n=0:N-1; for k=0:N-1 a(k+1)=(1/N)*sum(x.*exp(-j*k*2*pi*n/N)); end
```

then save the file on the work directory of the MATLAB.

On MATLAB, we will call the function as follows:

For example, the signal \( x[n]=\cos(\pi*n/4) \) is periodic of period \( N=8 \), so we will define the signal in one period and call the function for it as follows.

\[ \text{>> n=0:7;} \]
\[ \text{>> x=cos(pi*n/4);} \]
\[ \text{>> a=FScoeff(x)} \]
The result will be the fourier series coefficient of the cosine function:

Where: \( a_{1}=a_{-1}=0.5 \) but note that we calculate for positive \( k \) using the property of the discrete-time signals: \( a_{k}=a_{k+N} \) then we will have \( a_{1}=a_{-1+8}=a_{7}=0.5 \).
So the result will be all zeros except for the these two coefficients.

*Try also the signal \( x[n]=\sin(\pi*n/8) \) as a task.*

The other method for calculating the fourier series coefficient is as follows:

\[
a = \frac{1}{N} \text{fft}(x)
\]

where: \( N \) is the period of the signal \( x[n] \) and \( x \) is one period of the signal. The instruction is an abbreviation of fast fourier transform.

\[
\begin{align*}
>> n &= 0:7; \\
>> x &= \cos(\pi*n/4); \\
>> N &= \text{length}(x); \\
>> a &= (1/N) \text{fft}(x)
\end{align*}
\]

Compare the results of the function and the fft instruction.

### 7. Fourier Transform

You can find the frequency domain representation of any signal \( x(t) \) or \( x[n] \) using the fourier transform but in symbolic form using the `fourier` and then you can substitute a range of frequencies to plot the spectrum of the signal (its frequency domain representation).

For example if you would like to get the fourier transform of the following signal given below and plot its frequency spectrum:

\[
x(t) = te^{-2t}u(t)
\]
\begin{verbatim}
>> syms t n
>> x=t*exp(-2*t)*heaviside(t);
>> xw=fourier(x);
>> f=-pi:0.01:pi;          % for plotting x(jw) in the range [-pi: pi]
>> xf=subs(xw,f);
>> plot(f,xf)

You can do the same with laplace transform as follows:

>> xs=laplace(x)

Also, for Z transform you can use:

>> xz=ztrans(x)
\end{verbatim}

\textbf{8. Image processing}

It will be just an introduction for the image processing using MATLAB and you can depend on your self for further knowledge.

Images can be analyzed in MATLAB by reading them in three-dimensional matrices. The normal 2dimensional matrix is used to represent the image in number of pixels equal to the number of elements of the matrix and the 3\textsuperscript{rd} dimension is used to represent the color in RGB mode (Red-Green-Blue). If you cancel the 3\textsuperscript{rd} dimension the image will be in grey scale. You can now deal with the image as a matrix in which you can change any elements to find new colors and new image. Finally, note that the 0 represents black color while 255 represents white color because we use 8-bits for color representation as a binary word. The changes made can be shown, and saved in another file. Also you can check the properties of the file and the compression algorithm used for the image. It is highly recommended that you check the help of MATLAB in the used instructions for more details about the image processing for your further study.
The following code will assume that there is an image in partition C in your hard disk drive in a folder named \textit{PICS} and the name of the image is \textit{FLOWER.jpg} for example. The code will read the image and do some processing and finally it will save a lot of versions of the modified image in the same directory as follows:

\begin{verbatim}
>> A=imread('C:\PICS\FLOWER.jpg');
>> size(A) % A is a 3-D matrix
>> B=A(:,:,1);
>> imshow(B) % B is a 2-D matrix; grey scale
>> C=A(:,:,2);
>> imshow(C)
>> D=A(:,:,3);
>> imshow(D);
>> imwrite(B,'C:\PICS\FLOWER1.jpg');
>> imwrite(C,'C:\PICS\FLOWER2.jpg');
>> imwrite(D,'C:\PICS\FLOWER3.jpg');
>> imfinfo('C:\PICS\FLOWER.jpg')
>> A(1:10,:,1)=0;
>> imshow(A)
>> A(1:100,1:100,1)=255;
>> imshow(A)
>> E=rand(600,600,3);
>> imshow(E)
\end{verbatim}

Note that:

\begin{itemize}
  \item X OR 1=1
  \item X OR 0=X
  \item X AND 1=X
  \item X AND 0=0
\end{itemize}

Hence you can define a matrix of zeros or ones using the \texttt{ones} and \texttt{zeros} instructions and try anding and oring it with the grey scale image-matrix to follow how it will be black and white according to the results shown above.

\textbf{BEST WISHES}