Introduction to Working with Sampled Waveforms in MATLAB

J. Smith

June 24, 2008

(Excerpted – Mar 2009 from: J. Smith
Introduction to Fourier Transforms and MATLAB – 2008-06-24)

1.0 Introduction

This is an excerpt from a document written by Jim Smith on using MATLAB to perform Fourier analysis. It includes the first part of the original document on how to use MATLAB to generate and plot sampled waveforms.

2.0 Synthesizing Signals in MATLAB

If real world signal data is not available, we can easily generate signal data using basic MATLAB features. One pedagogical advantage of generating our own signals initially is that we can easily control and play with the signal characteristics and examine their effect on the FFT results.

The following are terms of signal technology that we will be dealing with.

time	**t** - independent variable time (sec)	
peak value	**r** - max height of the wave peak	
period of wave	**T** - time to complete a full wave	(sec)
frequency:	**f** - number of full waves per sec	(cycles/sec = Hz)
angular speed:	**ω** - speed of vector rotation	(radians/sec)
vector angle:	θ - current vector angle	(radians)
phase angle:	**Φ** - starting angle of vector	(radians)
dc level	**dc** – dc level of the signal	

Note that the angular speed (also called the angular frequency) comes from the fact that the end of a vector that rotates one complete revolution (2π radians of rotation) describes a sine wave on a time plot.

The following equations relate the terms together.

$$T = \frac{1}{f} \qquad \left(\frac{\sec}{cycle} = \sec\right) \qquad (1.1)$$

$$f = \frac{\omega}{2\pi} \qquad \left(\frac{radians/\sec}{radians/cycle} = \frac{cycles}{\sec} = Hz\right) \qquad (1.2)$$

$$\omega = \frac{2\pi}{T} \qquad \left(\frac{radians/cycle}{\sec/cycle} = \frac{radians}{\sec}\right)$$

(1.3)

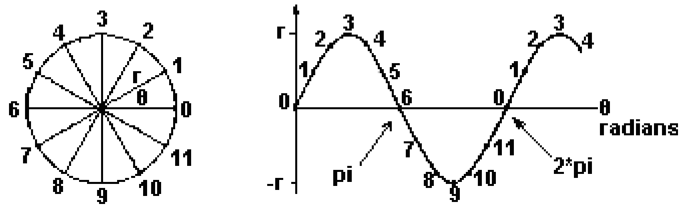
2.1 Synthesizing Sine Waves

To generate a single sine wave, we can use the fact that if one rotates a vector of length r through a complete circle (2π radians) and plots the end position of the vector as:

$$y = r * \sin\theta \qquad (1.4)$$

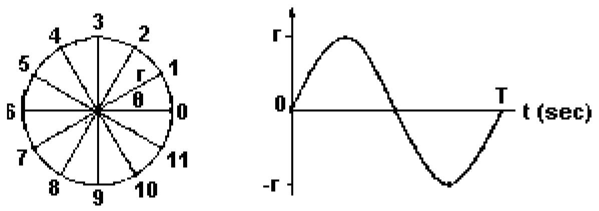
versus the angle θ as shown below, the resultant graph is that of a sine wave.

This is shown in the diagram below in which twelve positions of the vector are mapped on to the resulting plot. Of course the maximum and minimum values of the sine curve are r and –r respectively. The sine wave crosses the θ axis at $\theta = \pi$ and $\theta = 2\pi$.



This rotating vector way of visualizing signal generation is very powerful and makes it easier to understand what is happening in signal analysis.

However, with signals we want to talk in terms of time, not angles. The following diagram shows the same rotating vector but with a time-based plot to its right.



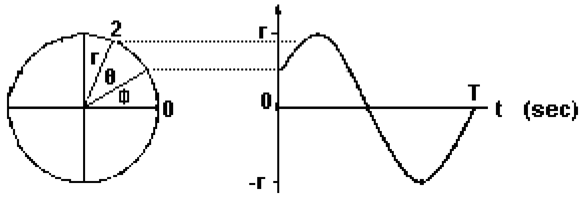
The relationship between angle and time is that one complete revolution of the vector (one complete wave) takes place in T seconds. The relationships between angle and time are:

$$\theta = \omega t \qquad (1.5)$$

$$\theta = 2\pi f t \qquad (1.6)$$

$$\theta = \frac{2\pi t}{T} \qquad (1.7)$$

Now, in the real world when we start looking at signals there is no guarantee that all waves will start at an angle of zero. In fact there is a guarantee that most signals will not start at a zero angle at time zero. In order to model these cases we have to introduce a phase offset angle (starting angle) Φ to the rotating vector. The following diagram shows a phase offset angle of Φ and the corresponding impact on the generated waveform which starts further into its cycle.



Thus the waveform is actually generated from

$$\text{Vector angle} = \theta + \Phi \qquad \text{(radians)} \qquad (1.8)$$

Additionally, a wave form may also have an offset or DC level to it which is just the addition of a constant value. If we use equation (1.8) for the angle and add a DC term, equation (1.4) now becomes:

$$y = \sin(\theta + \Phi) + dc \qquad (1.9)$$

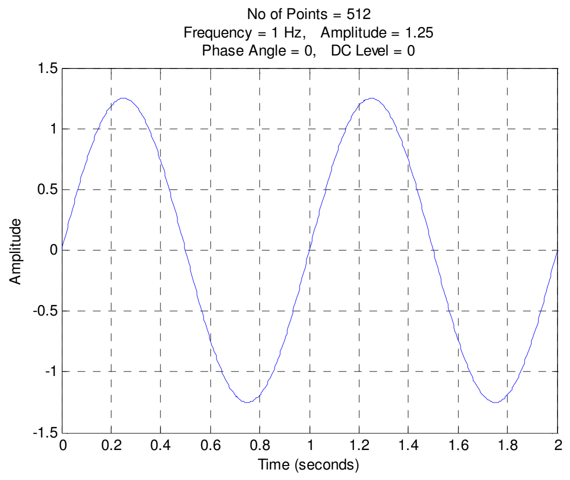
or equivalently,

$$y = \sin(2\pi f\, t + \Phi) + dc \qquad (1.10)$$

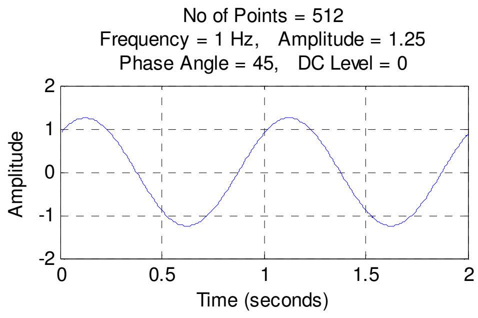
The following program uses equation (1.10) to generate a sine wave for plotting. The simulation generates N data points over the simulation limits of `tstart` and `T` (here T is the period of data sample).

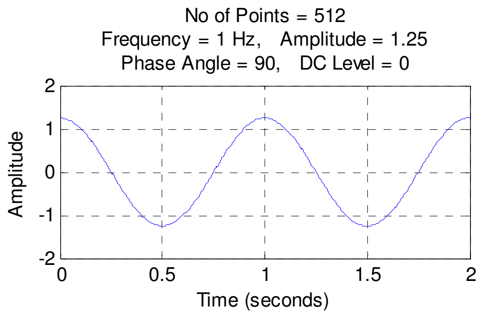
```
% Example Program 1
% Simple program to generate and plot a single sine wave

clear; clc;       % close workspace & command window
close all;        % close figures

% Set the sine wave parameters and time parameters

N = 512;          % number of data points
f = 1;            % wave frequency in cycles/sec (Hz)
r = 1.25;         % amplitude of the sine wave
phi = 0;          % phase angle
dc = 0;           % dc offset

tstart = 0;       % simulation start (sec)
T = 2;            % simulation end time (sec)

% Calculate the time base, the angles and the curve values

dT = (T-tstart)/(N-1);
t = [tstart:dT:T];                 % calculate time base values
theta = 2*pi*f*t;                  % calculate angle values
y = r*sin(theta + phi/180*pi) + dc;    % calculate amplitudes

% Plot the signal

plot(t,y); grid on;
title({['No of Points = ',num2str(N)],...
       ['Frequency = ',num2str(f),' Hz,',...
                        '   Amplitude = ',num2str(r)],...
       ['Phase Angle = ',num2str(phi),...
                        ',    DC Level = ',num2str(dc)]});
xlabel('Time (seconds)'); ylabel('Amplitude');
```

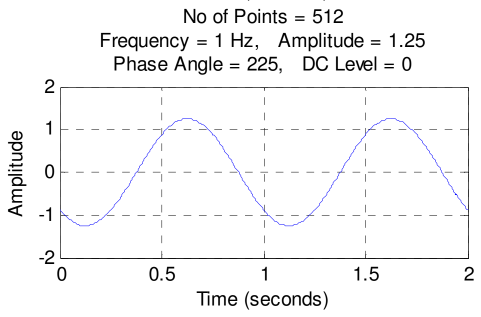
The graph from the above program follows. The graph parameters are shown at the top of the graph. The plot is a sine wave with a frequency 1 Hz over a time period of 2 sec.

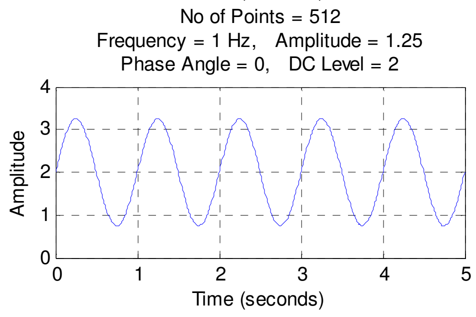


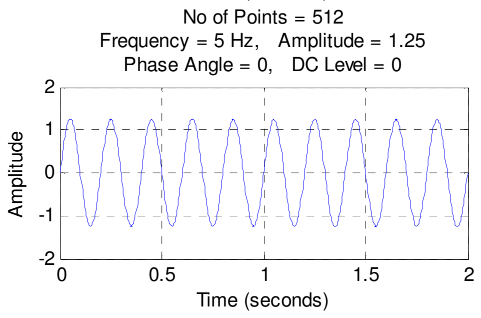
The following graphs were generated from a similar program. It varied the sine wave and graphing parameters as indicated in the graphs. The first three graphs vary the phase angle by 45°, 90° and 225° respectively. As seen in graph two, a sine wave with a phase shift of 90° is in fact a cosine wave. Graph three has a DC level of 2 which raises the curve and although graphs 5 and 6 look identical, graph 6 has twice the frequency and half the simulation time.

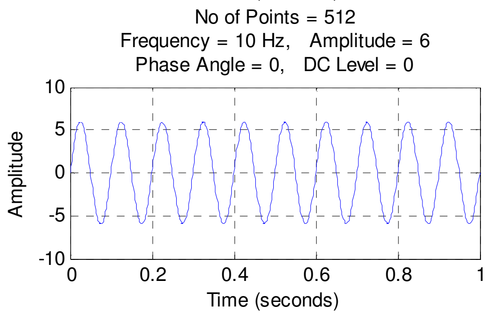












2.2 Synthesizing Square Waves

If one has the MATLAB Signal Processing Toolbox there are a number of functions to help generate different periodic waves and pulse trains. If one does not have that toolbox then it is possible to write a function to generate these other waves.

For instance if there is, in the directory, a file called squarewave.m which contains the function squarewave as shown below:

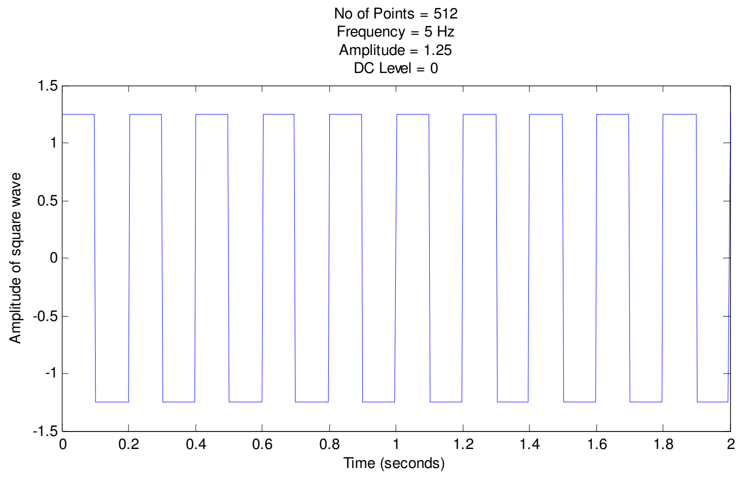
```
% This function calculates a square wave

function [value] = squarewave(t)
    value = mod(t,2*pi);
    for i = 1:length(t)
        if (value(i) < pi) value(i) = 1;
        else value(i) = -1;
```

```
            end
        end
    end
```

and if one substitutes the following statement

```
            y = r*squarewave(theta + phi/180*pi) + dc;
```

into Example Program 1, in place of the statement that calculates the sine values, then the plot below will be generated. Note that the grid was turned off to make the curve more visible.



2.3 Synthesizing Triangle Waves

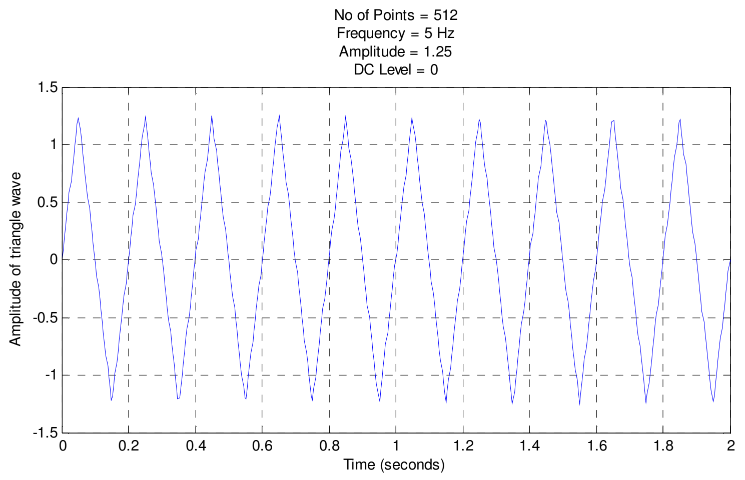
As another example, if in the directory, there also exists a file called trianglewave.m containing the trianglewave function as shown below.

```
    % This function calculates a triangle wave

    function [value] = trianglewave(t)
        value = mod(t,2*pi);
        for i = 1:length(t)
            if     (value(i) < pi/2)   value(i) = value(i)*2/pi;
            elseif (value(i) < 3*pi/2) value(i) = 2-2*value(i)/pi;
            else                       value(i) = 2*value(i)/pi-4;
            end
        end
    end
```

and the following statement is used in Example Program 1:

```
            y = r*trianglewave(theta + phi/180*pi) + dc;
```

one gets the following graph.



There may be more efficient ways to write these functions but the way shown is quite explicit and clear and can be applied to many complex situations.

2.4 Synthesizing Signals and Nyquist

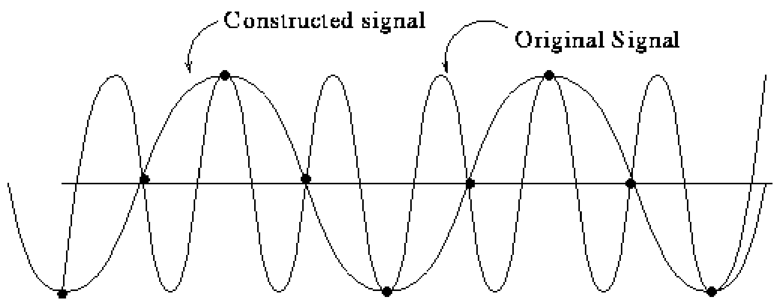
When acquiring real-world signal data, there is a question of how many data samples should be taken. One wants enough data points to accurately capture all the signal information but not an unnecessary excess of such points. The considerations here are storage requirements, processing time and cost of signal acquisition equipment.

The same considerations also exist when synthesizing signal data. A decision on the number of data points to use depends entirely on how fast the signal is changing which is a function of the signal frequencies.

The **Nyquist Theorem** addresses this issue. It states that in order to capture enough information about a signal to be able to reproduce it, we must sample the signal at least twice the frequency of the highest frequency component.

Said another way, the highest frequency which can be accurately represented is less than one-half of the sampling rate. This highest frequency allowed for a given sampling rate is called the **Nyquist frequency**.

If we sample at a frequency less than the **Nyquist rate**, the samples will reconstruct another signal of lower frequency. This phenomenon of a higher frequency signal behaving as a lower frequency signal is called **aliasing**. The following figure shows aliasing.



Thus the Nyquist Theorem explains why compact discs are recorded at a sampling frequency of 44 kHz while the human ear can only hear sounds up to about 20 kHz. This gives a ratio between the sampling frequency and the analysis (playback) frequency of about 44/20 = 2.2. Thus in this case, the phrase 'over at least twice the frequency' means 2.2 times.

3.0 Storing Signal Data to Disk

In most situations, signal data will not be generated within the same program as the one that does the analysis but rather the signal data will be received and analyzed in real-time or stored to stored to disk for later processing.

If the data comes from disk, the required information for analysis is the signal data itself and some timing information for the data. One way to do this is to begin the data file with a number for the sampling frequency of the data and then the actual signal data itself. If this method is followed then the user may read any portion of the data file and still extract proper timing information for the Fourier Transform function.

The data period represented by the extracted data is calculated from the sampling frequency and the number of points according to the formula of Section 2.0, namely T = N/fs.

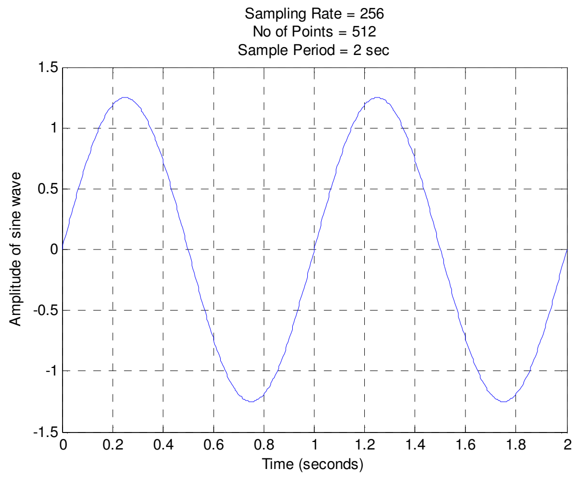
The following program is a modification of program 1 to write the signal data to a file.

```
% Example Program 2
% This program writes signal data to a file.

clear; clc;       % clear workspace & command window
close all;        % close figures

% Set the sine wave parameters and time parameters

N = 512;          % number of data points
f = 1;            % wave frequency in cycles/sec (Hz)
r = 1.25;         % amplitude of the sine wave
phi = 0;          % phase angle
dc = 0;           % dc offset

tstart = 0;       % simulation start (sec)
T = 2;            % simulation end time (sec)

% Calculate the time base, the angles and the curve values

dT = (T-tstart)/(N-1);
t = [tstart:dT:T];               % calculate time base values
theta = 2*pi*f*t;                % calculate angle values
y = r*sin(theta + phi/180*pi) + dc;      % calculate amplitudes

% Open an output file and write the data to it.

fs = N/T;
fid = fopen('Sine data,f=1.txt','w');         % Open the file
fprintf(fid,'%6.2f\n',fs);               % Write the sample rate
fprintf(fid,'%12.8f\n',y);              % Write the signal data
status = fclose(fid);                   % Close the file
```

This program reads in the signal data from the above file and graphs it.

```
% Example Program 3
% This programs reads in signal data from a
% file and plots the data

clear; clc;                 % clear workspace & command window
close all;                  % close figures

% Open the file and read the data

fid = fopen('Sine data.txt','r');  % Open the file
fs = fscanf(fid,'%g',1);           % read the sampling rate

y = fscanf(fid,'%g');              % read the signal data
status = fclose(fid);

N = length(y);
T = N/fs;
t = [0:T/(N-1): T];

% Plot the signal

plot(t,y); grid on;
title({['Sampling Rate = ',num2str(fs)],...
       ['No of Points = ',num2str(N)],...
       ['Sample Period = ',num2str(T),' sec']});
xlabel('Time (seconds)'); ylabel('Amplitude of sine wave');
```

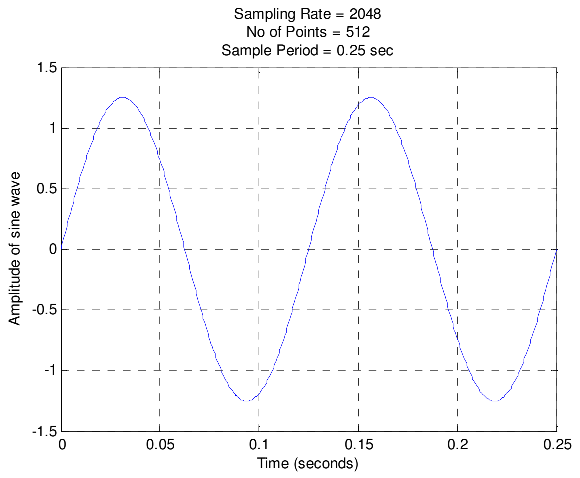
The output from the program follows. It is exactly the same as the plot from program 1.



Note that for the above signal the frequency, amplitude and DC level information is in the signal itself and is not explicitly available without some sort of analysis. The sampling rate provides the context for the signal data, since from the sampling rate and the number of points, the period for the signal can be obtained. In this case, T = N / fs = 512 / 256 = 2 sec.

To take this thought further, if program 2 is changed such that f = 8 HZ and T = 0.25 sec and another data file is generated, then the following graph results. This graph and its data file are exactly the same as for the above example except that the sampling rate in the file is 2048 Hz instead 256 Hz.

Now when program 3 reads in this new data this new sampling rate provides the context for the interpretation of the signal. In this case, T = N / fs = 512 / 2048 = 0.25 sec.



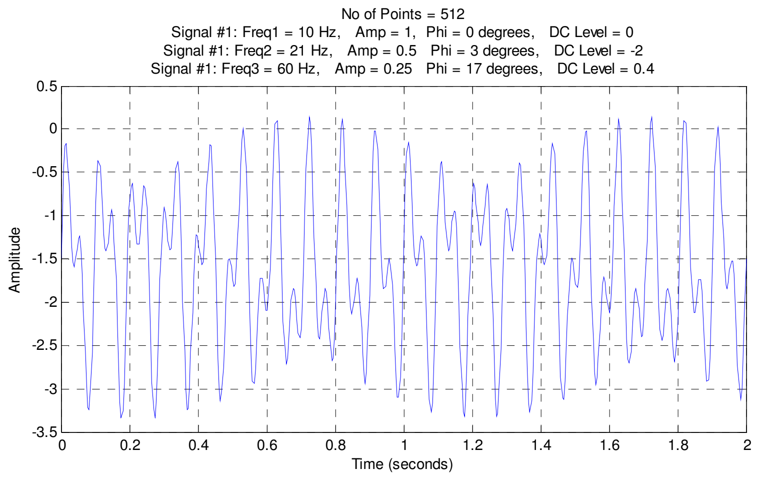
4.0 Synthesizing Signals with Waves of Different Frequencies

Using the techniques of the above sections we can write a MATLAB program to generate a signal of many different wave parameters. The following program generates three separate signals, each with different frequencies, amplitudes, phase angles and DC levels. The composite signal is formed by just adding these three signals together.

```
% Example Program 4
% Synthesize several sine waves with different parameters

clear; clc; clear all;          % close workspace & command window
close all;                      % close figures

% Set up wave parameters

N = 512;
f = 1; r = 1.25; phi = 0; dc = 0;

% Set up simulation parameters

tstart = 0;          % simulation start (sec)
T = 2;               % simulation end time (sec)

% Calculate the time base, the angles and the curve values

dT = (T-tstart)/(N-1);
```

```
t = [tstart:dT:T];                % calculate time base values
theta = 2*pi*f*t;                 % calculate angle values
y = r*sin(theta + phi/180*pi) + dc;       % calculate amplitudes

% Set the signal parameters

f1 = 10;  r1 = 1.0;  phi1 =  0; dc1 =  0;
f2 = 21;  r2 = 0.5;  phi2 =  3; dc2 = -2;
f3 = 60;  r3 = 0.25; phi3 = 17; dc3 = .4;

% Calculate the angles and the sine signals and
% add them together

theta = 2*pi*f1*t + phi1/180*pi; y1 = r1*sin(theta) + dc1;
theta = 2*pi*f2*t + phi2/180*pi; y2 = r2*sin(theta) + dc2;
theta = 2*pi*f2*t + phi3/180*pi; y3 = r3*sin(theta) + dc3;
yt = y1 + y2 + y3;

% Plot the composite signal

plot(t,yt); grid on;
title({['No of Points = ',num2str(N)],...
       ['Signal #1: Freq1 = ',num2str(f1),' Hz,    ',...
                          'Amp = ', num2str(r1),',   ',...
                          'Phi = ', num2str(phi1), ' degrees,    ',...
                          'DC Level = ', num2str(dc1)],...
       ['Signal #1: Freq2 = ',num2str(f2),' Hz,    ',...
                          'Amp = ', num2str(r2),'    ',...
                          'Phi = ', num2str(phi2),' degrees,    ',...
                          'DC Level = ', num2str(dc2)],...
       ['Signal #1: Freq3 = ',num2str(f3),' Hz,    ',...
                          'Amp = ', num2str(r3),'    ',...
                          'Phi = ', num2str(phi3),' degrees,    ',...
                          'DC Level = ', num2str(dc3)]});
xlabel('Time (seconds)'); ylabel('Amplitude');
```

The following figure shows the output of the above program. As can be seen, the signal is quite complicated.


No of Points = 512
Signal #1: Freq1 = 10 Hz, Amp = 1, Phi = 0 degrees, DC Level = 0
Signal #1: Freq2 = 21 Hz, Amp = 0.5 Phi = 3 degrees, DC Level = -2
Signal #1: Freq3 = 60 Hz, Amp = 0.25 Phi = 17 degrees, DC Level = 0.4
Amplitude
Time (seconds)