

# ENEL 563 Biomedical Signal Analysis (Fall 2005)

## Introduction to Matlab and guidelines for plotting effective graphs

Matlab is a tool for technical and scientific computation. It consists of a high-level programming language, a development environment, and a computing engine designed for numerical and symbolic computation. Matlab facilitates the development of numerical algorithms due to its extensive library of numerical functions, its interpreted language, and its support for vectorized operations.

### 1 Matlab programming fundamentals

#### 1.1 Simple calculations using Matlab

After starting Matlab, a prompt symbol “>>” is displayed. Matlab is ready to accept commands. Simple numerical expressions can be typed directly into the command line:

```
>> ((42 * 36) / (10 + 4) - 110) ^ 3
```

```
ans =
```

```
    -8
```

```
>> 54 / 0
```

```
Warning: Divide by zero.
```

```
ans =
```

```
    Inf
```

```
>> 0 / 0
```

```
Warning: Divide by zero.
```

```
ans =
```

```
    NaN
```

```
>> pi
```

```
ans =
```

```
    3.1416
```

```
>> pi;
```

```
>> 'Hello world'
```

```
ans =
```

```
Hello world
```

Observe the usage of the numerical operators. The “^” represents the exponential operator. Matlab will handle a division by zero by either producing an `Inf` (infinite) or a `NaN` (“not-a-number”) result. Some Matlab functions (e.g. plotting functions) may not accept these special numbers as input, causing Matlab to print an error message and stop processing.

Notice also the usage of the semicolon at the end of the line: it prevents Matlab from printing the outcome of a calculation on the screen (which is highly desirable when working with vectors and matrices containing thousands or millions of values!). Also observe that strings are enclosed by single quotes.

Matlab also performs calculations using complex numbers:

```
>> sqrt(-2)

ans =

    0 + 1.4142i

>> abs(3 + j*4)

ans =

    5

>> angle(3 + j*4)

ans =

    0.9273

>> 5 * cos(0.9273)

ans =

    3.0000

>>
```

Note: Matlab’s trigonometric functions require all angle quantities to be expressed in radians.

Matlab contains an extensive library of built-in functions: you can discover Matlab’s functions using the Help menu. The command `help` also provides information on how to use Matlab’s functions, for instance:

```
>> help exp
EXP      Exponential.
        EXP(X) is the exponential of the elements of X, e to the X.
        For complex Z=X+i*Y, EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y)).

        See also expm1, log, log10, expm, expint.
```

Overloaded functions or methods (ones with the same name in other directories)

```
help sym/exp.m
```

Reference page in Help browser  
doc exp

**Try this:** Compute  $\exp(j\pi) + 1$ . What was the expected result? What is the result provided by Matlab? How do you interpret Matlab's result?

Some useful Matlab commands (**Try this:** use `help` to learn how to use them):

Command	Purpose
<code>cd</code>	change directory
<code>pwd</code>	prints the current directory
<code>mkdir</code>	make new directory
<code>ls</code>	list files
<code>edit</code>	Matlab's text editor
<code>clc</code>	clears the screen
<code>clear</code>	erases variables from memory
<code>disp</code>	prints a string on the screen
<code>input</code>	reads input from the keyboard

## 1.2 Matlab scripts

Instead of just typing isolated commands on the prompt, you can write a Matlab script to automate your calculations. The script is a text file whose name has a “.m” extension, and contains a sequence of Matlab commands. In order to call the script from the command line or from another script, use the script's file name (without the “.”). A script may also contain comments, which start with the symbol “%”. Here is an example of a simple script:

```
%  
% Program to compute the volume of a sphere  
% The volume of a sphere is given by the formula  
%  
% volume = (4/3) * pi * radius^3  
%  
  
% Read the radius  
radius = input('Enter the sphere radius (in meters): ');  
  
% Compute the volume  
volume = (4/3) * pi * radius^3; % This is just another comment  
  
% Display the calculated volume  
output_string = ...  
    sprintf('The sphere volume is %f cubic meters.', volume);  
disp(output_string);
```

**Try this:** Type the program above into a Matlab script file named `sphere_volume.m` (use the command “`edit sphere_volume.m`”). Execute the script by typing “`sphere_volume`” in the command line.

Unlike the C programming language, Matlab commands cannot be divided into more than one line, unless ellipsis (“...”) are used.

### 1.3 Control flow commands

The following program illustrates the usage of the `if`, `while`, `for`, and `switch` commands:

```
%
% Program to compute the factorial of a given integer
%

% Read the input integer
done = false;
while ~done
    N = input('Enter an integer between 0 and 10: ');

    if fix(N) ~= N
        disp('The number must be an integer!');
    elseif (N < 0) || (N > 10)
        disp('The number is out of bounds!');
    else
        done = true;
    end
end

% Compute the factorial function
if (N == 0) % Trivial case
    f = 0;
elseif (N == 1) % Trivial case
    f = 1;
else % General case, compute the factorial in a for loop
    f = 1;

    for k = 2:N
        f = f * k;
    end
end

disp(sprintf('The factorial of %d is %d', N, f));

% More interesting information about the given number
switch(f)
    case 2
        disp('An even prime number!');
    case 6
        disp('A perfect number!');
    otherwise
        disp('A boring number.');
```

Observe the syntax for the `for` loop, which causes the loop to execute once for each integer between 2 and  $N$ .

**Try this:** Learn about the relational operators using “`help relop`”

**Try this:** The code above is in the file `fact.m`. Study the code, execute it, until you are familiar with the various control flow commands (`for`, `if`, etc).

**Try this:** What happens if you run “`help fact`”?

## 1.4 Functions

Like many other programming languages, Matlab allows the user to develop new functions. Matlab’s functions have some special characteristics:

- Arguments are always passed by value, not by reference.
- The function can have many output arguments.

The code for a function must be saved in a “.m” file. A single file may contain several functions, but only the first function on the file is accessible externally (from other functions and scripts, for instance). The remaining functions are local: they can only be used within the same file. The first line of the “.m” file must contain the syntax definition for the function. The name of the new function is the name of the “.m” file (similar to scripts).

Here is an example of a function (file `gaussian.m`):

```
function y = gaussian(x, mu, sigma)
% gaussian: compute the Gaussian p.d.f.
%
% The function y = gaussian(x, mu, sigma) computes
% the value of the Gaussian p.d.f. with parameters
% mu and sigma, at the value x, and returns the
% computed value in y
%
% If sigma is zero, then y is undefined, and a NaN
% is returned

if (sigma == 0)
    y = NaN;
else
    aux1 = 1 / (sigma * sqrt(2 * pi));
    aux2 = (x - mu)/sigma;

    y = aux1 * exp(-(1/2)*aux2^2);
end
```

and here is how to use it:

```
>> help gaussian
gaussian: compute the Gaussian p.d.f.

The function y = gaussian(x, mu, sigma) computes
the value of the Gaussian p.d.f. with parameters
mu and sigma, at the value x, and returns the
computed value in y

If sigma is zero, then y is undefined, and a NaN
is returned
```

```
>> gaussian(3,0,1)
```

```
ans =
```

```
    0.0044
```

```
>> gaussian(3,0,0)
```

```
ans =
```

```
NaN
```

Here is another example of a function:

```
function [surface, volume] = sphere_measure(radius)
surface = sphere_surface(radius);
volume = sphere_volume(radius);
```

```
function surface = sphere_surface(radius)
surface = 4 * pi * radius^2;
```

```
function volume = sphere_volume(radius)
volume = (4/3) * pi * radius^3;
```

Observe that the functions `sphere_surface` and `sphere_volume` are local functions.

Matlab functions can be called recursively, for instance:

```
function f = fact_recursive(n)

if (n <= 2)
    f = n;
else
    f = n * fact_recursive(n - 1);
end
```

## 1.5 Advanced programming in Matlab

There are many advanced topics on Matlab programming that will not be covered in this short tutorial, including:

- anonymous functions, nested functions, function references;
- structures and cells;
- object-oriented programming;
- interface with C and C++

## 2 Matrices and vectors in Matlab

Matlab is short for “Matrix laboratory”. The Matlab language was developed by Cleve Moler in the 1970s, with the intention of facilitating access to the linear algebra Fortran packages LINPACK and EISPACK, without requiring that his students learn Fortran. Matlab became popular among universities, and in 1984 the company MathWorks was founded. Therefore, it is no surprise that Matlab’s strength resides in its powerful matrix manipulation capabilities. Compare the following C and Matlab programs to compute the dot product of two  $N$ -dimensional vectors:

C program	Matlab program
<pre>double dot_product(double u[],                   double v[],                   int N) {     int k;     double val;      val = 0.0;     for (k = 0; k &lt; N; k++)     {         val += u[k] * v[k];     }      return(val); }</pre>	<pre>function val = dot_product(u,v) val = sum(u .* v); % .* : array multiplication  or simply  function val = dot_product(u,v) val = u' * v; % u' : transpose of u</pre>

The creation and manipulation of matrices will be discussed in the following sections.

### 2.1 Creating matrices and vectors

The next table lists some examples of matrix and vector creation:

Creating a $2 \times 3$ matrix	<pre>&gt;&gt; A = [1 5 7; 2 9 8]  A =       1     5     7      2     9     8</pre>
Creating an empty $2 \times 3$ matrix	<pre>&gt;&gt; A = zeros(2,3)  A =       0     0     0      0     0     0</pre>

<p>Creating an empty <math>3 \times 1</math> column vector</p>	<pre>&gt;&gt; v = zeros(3,1)  v =      0     0     0</pre>
<p>Creating a <math>1 \times 3</math> row vector filled with ones</p>	<pre>&gt;&gt; v = ones(1,3)  v =      1    1    1</pre>
<p>Creating sequences using the colon (":") operator</p>	<pre>&gt;&gt; v = 1:5  v =      1    2    3    4    5  &gt;&gt; v = -1.1:0.4:0.2  v =  -1.1000  -0.7000  -0.3000   0.1000  &gt;&gt; v = 0.2:-0.4:-1.1  v =      0.2000   -0.2000   -0.6000   -1.0000</pre>

<p>Creating an array with more than two dimensions</p>	<pre>&gt;&gt; A = zeros(2,2,3)  A(:,:,1) =      0    0     0    0  A(:,:,2) =      0    0     0    0  A(:,:,3) =      0    0     0    0</pre>
<p>Creating an empty matrix (zero dimension)</p>	<pre>&gt;&gt; A = []  A =      []</pre>

## 2.2 Matrix manipulation

Accessing matrix and vector elements:

<p>Accessing a single matrix element</p>	<pre>&gt;&gt; A = [1 2 3; 4 5 6]; v = [1 2 3 4]; &gt;&gt; A(2,1)  ans =      4  &gt;&gt; v(3)  ans =      3  &gt;&gt; v(1,3)  ans =      3</pre>
--	--

Accessing multiple matrix elements at once

```
>> A = [1 2 3; 4 5 6]
```

```
A =
```

```
    1    2    3
    4    5    6
```

```
>> A([1 2], [1 3])
```

```
ans =
```

```
    1    3
    4    6
```

```
>> A(1,:)
```

```
ans =
```

```
    1    2    3
```

```
>> A(:,2)
```

```
ans =
```

```
    2
    5
```

```
>> A(:, :)
```

```
ans =
```

```
    1    2    3
    4    5    6
```

Matlab is very flexible with respect to memory allocation and matrix size: a matrix can be extended or shortened effortlessly:

Joining two matrices	<pre>&gt;&gt; A = [1 2 ; 3 4] A =      1     2      3     4 &gt;&gt; B = [5 6 ; 7 8] B =      5     6      7     8 &gt;&gt; C = [A B] C =      1     2     5     6      3     4     7     8 &gt;&gt; D = [A ; B] D =      1     2      3     4      5     6      7     8</pre>
----------------------	--

<p>Inserting and removing new elements in the middle of a vector</p>	<pre>&gt;&gt; v = 4:7 v =      4     5     6     7  &gt;&gt; v = [v(1:3) 42 v(4)] v =      4     5     6    42     7  &gt;&gt; v(2:4) = [] v =      4     7</pre>
<p>Circular rotation of a vector</p>	<pre>&gt;&gt; v = 1:5 v =      1     2     3     4     5  &gt;&gt; v_right = [v(end) v(1:(end - 1))] v_right =      5     1     2     3     4  &gt;&gt; v_left = [v(2:end) v(1)] v_left =      2     3     4     5     1</pre>

These examples are impressive when compared with any equivalent implementation in the C language.

### 2.3 Matrix computations

The numerical operators “+”, “-”, and “\*” have the familiar meanings in the context of linear algebra. The operator “'” means matrix transposition.

Scalar multiplication	<pre>&gt;&gt; A = [1 2 ; 3 4]; &gt;&gt; B = 3 * A  B =       3     6      9    12</pre>
Matrix multiplication	<pre>&gt;&gt; A = [1 2 3 ; 4 5 6]  A =       1     2     3      4     5     6  &gt;&gt; B = [0 -1 ; 1 0]  B =       0    -1      1     0  &gt;&gt; C = B * A  C =      -4    -5    -6      1     2     3  &gt;&gt; D = A * B ??? Error using ==&gt; mtimes Inner matrix dimensions must agree.</pre>

<p>Raising a matrix to a power</p>	<pre>&gt;&gt; A = [1 2 ; 3 4]  A =       1     2      3     4  &gt;&gt; A^3  ans =      37    54     81   118  &gt;&gt; A * A * A  ans =      37    54     81   118</pre>
<p>Transposing a matrix</p>	<pre>&gt;&gt; A = [1 2 3 ; 4 5 6]; &gt;&gt; A'  ans =       1     4      2     5      3     6</pre>

Often, an operation must be performed between the individual elements of two matrices. The operators that perform these “pointwise” calculations have a dot before the usual numerical operator, as shown in the following example:

Array multiplication versus matrix multiplication

```
>> A = [1 2 3 ; 4 5 6]

A =

     1     2     3
     4     5     6

>> B = [5 5 5 ; 10 10 10]

B =

     5     5     5
    10    10    10

>> C = A * B
??? Error using ==> mtimes
Inner matrix dimensions must agree.

>> D = A .* B

D =

     5    10    15
    40    50    60
```

Use `help arith` to learn more about the arithmetical operators.

### 3 Creating graphs in Matlab

Matlab has an extensive library for producing high-quality graphs. In this section, we will explore an example of a 2-D plot of a simple signal (file `plot_example.m`), and its Discrete Fourier Transform (DFT).

First of all, it is always a good practice to clear the memory of any variables prior to the execution of the script, as shown in the first lines of the `plot_example.m` file.

```
%
% Example of how to use the plot command
%
```

```
clear all;
close all;
```

Next, the signal  $y(t) = 5 \sin(2\pi 3(t - 0.5)) + 2 \sin(2\pi 5t)$  is generated over a duration of 5 seconds, with a sampling rate of 1kHz. This signal contains two pure sine waves at the frequencies 3Hz and 5Hz.

```
%
% A signal is represented in Matlab as a vector.
```

```

% First, generate the abscissa values. Then,
% generate the corresponding ordinate values
%
fs = 1000;      % sampling frequency in Hz
T = 1/fs;      % sampling interval
tf = 5;        % end time

t = 0 : T : tf;          % abscissa
y = 5*sin(2*pi*3*(t - 0.5)) + ... % ordinate
    2*sin(2*pi*5*t);

```

The next section of code shows how to plot the signal, give a title to the graph, and label the axis. Figure 1 shows the plotted signal. **Important: any plot must contain the title, axis labels, and a proper abscissa scale.** Throughout this course, you are required to always follow this guideline when producing your plots.

The command `figure` creates a new blank figure window, and the subsequent `plot` command generates the graph in this figure window. Each new graph requires a new figure window: otherwise, the new graph will overwrite the old graph in the last figure window left open.

```

%
% Plotting the signal
%
figure;
plot(t,y);

title('Plot example');
xlabel('time [s]');
ylabel('signal amplitude');

```

The `fft` command returns a vector of complex numbers which is the DFT of the given discrete-time signal. Observe that  $Y(1)$  corresponds to the DC frequency value,  $Y(2)$  corresponds to  $f = f_s/N$  in Hz,  $Y(3)$  corresponds to  $f = 2f_s/N$ , and so on ( $f_s$ : sampling frequency,  $N$ : number of samples in the signal and/or the DFT array).

```

%
% Taking the Discrete Fourier Transform of
% the test signal
%
Y = fft(y);
mag_Y = abs(Y);
N = length(y);

```

Now let us analyze some possible ways to present the DFT magnitude in a graph. In the first method, the frequency axis covers the range  $[0, f_s]$ , starting at DC ( $f = 0\text{Hz}$ ). Figure 2 shows the resulting graph. The problem with this graph is that all the significant information is compressed at the edges of the graph, making it difficult to obtain any useful knowledge out of this plot.

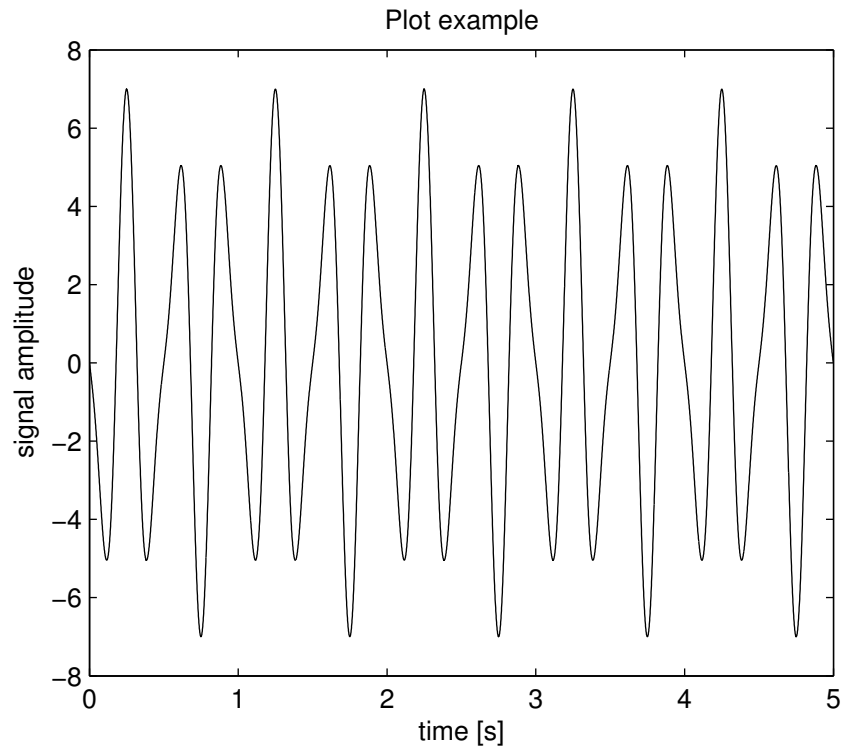


Figure 1: A simple test signal.

```

%
% Plotting the DFT magnitude, method 1.
% The frequency axis spans the whole unit circle
% starting at f = 0.
%
f = [0 : (N-1)] * (fs/N);

figure;
plot(f, mag_Y);
title('DFT magnitude, method 1');
xlabel('Frequency [Hz]');
ylabel('Magnitude');

```

Another method for the visualization of the DFT magnitude is to place the DC frequency at the center of the graph. In this case, the frequency axis spans the range  $[-f_s/2, f_s/2]$ . Figure 3 shows the resulting plot. Notice that the magnitude spectrum is symmetric about the origin, as expected. Observe also that this graph presents the same problem as in the first method: all the useful information is compressed in a narrow area, making it indistinguishable.

```

%
% Plotting the DFT magnitude, method 2
% The frequency axis spans the whole unit circle

```

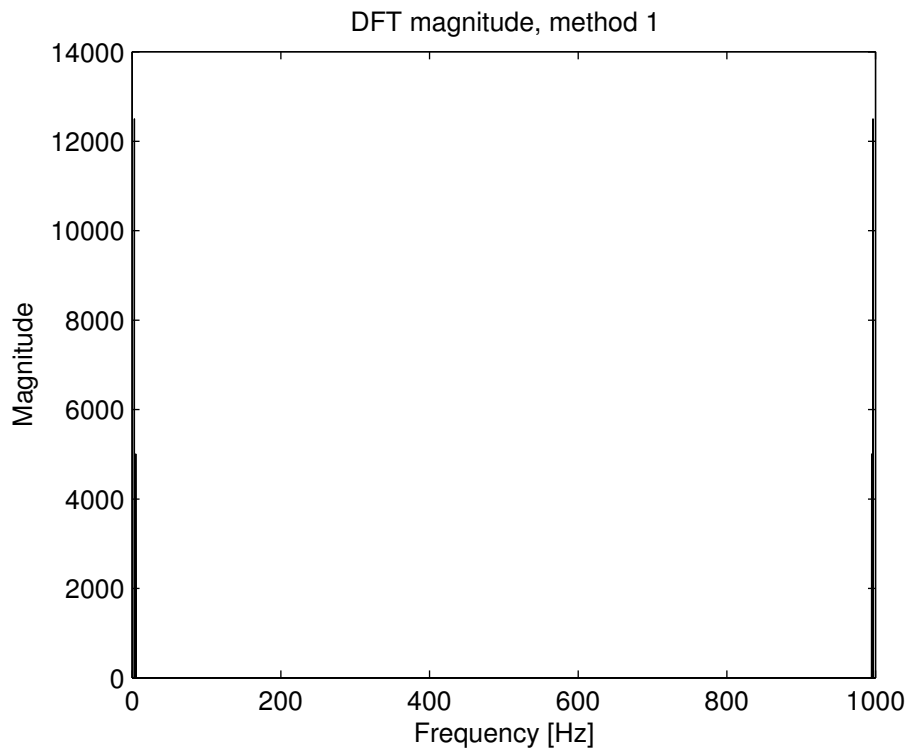


Figure 2: Magnitude spectrum of the simple signal in Figure 1, first method.

```
% Zero frequency is placed in the middle of the plot
% using the fftshift command on the mag_Y vector
% and using proper generation of the abscissa.
%
p = ceil(N/2);
f = [(p - N) : (p - 1)] * (fs/N);
mag_Y_shifted = fftshift(mag_Y);

figure;
plot(f, mag_Y_shifted);
title('DFT magnitude, method 2');
xlabel('Frequency [Hz]');
ylabel('Magnitude');
```

Because the magnitude spectrum is symmetric about the origin, there is no need to display the spectrum for both the positive and the negative frequencies. Therefore, it may be advantageous to plot only the positive frequencies, as illustrated in the code below. The resulting graph is shown in Figure 4.

```
%
% Plotting the DFT magnitude, method 3.
% The frequency axis spans half of the unit circle
% (positive frequencies and DC).
%
```

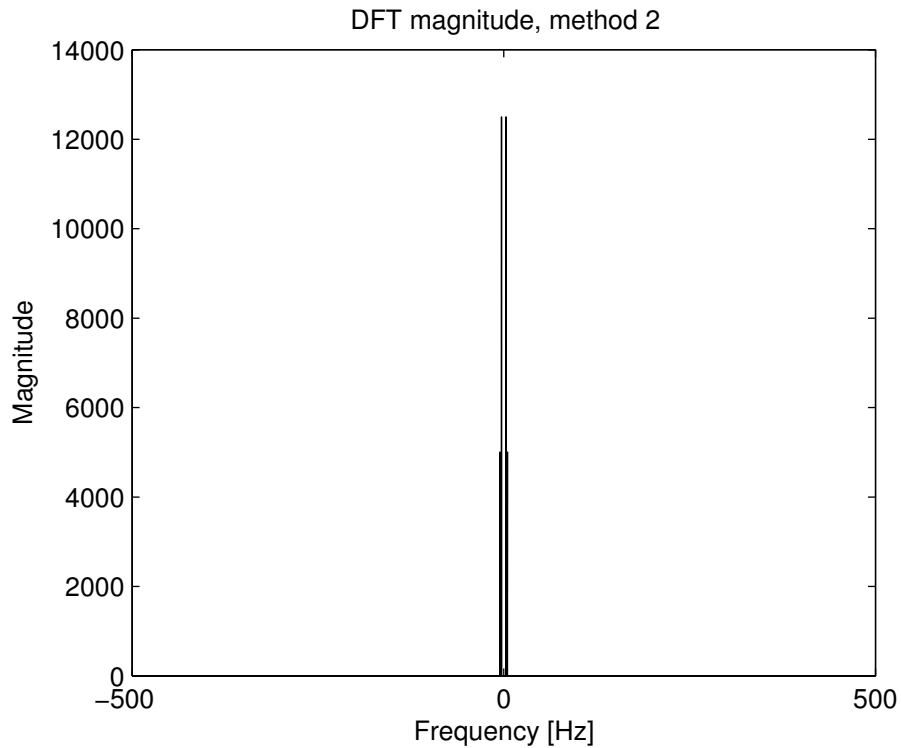


Figure 3: Magnitude spectrum of the simple signal in Figure 1, second method.

```
p = ceil(N/2);
f = [0:(p - 1)] * (fs/N);
mag_Y_pos_freq = mag_Y(1:p);

figure;
plot(f, mag_Y_pos_freq);
title('DFT magnitude, method 3');
xlabel('Frequency [Hz]');
ylabel('Magnitude');
```

The command `axis` can be used to expand an area of interest in the graph. The interesting and useful information in the graph is finally revealed, as shown in Figure 5. Now, it is clear that the simple test signal contains two frequency components, placed at 3Hz and 5Hz, as expected.

```
%
% Plotting the DFT magnitude, method 3b.
% The frequency axis spans half of the unit circle
% (positive frequencies and DC).
% The command "axis" is used to highlight
% the important part of the graph.
% The limits used in the "axis" command are obtained
% from visual inspection of the figure in any of the
% preceding methods.
```

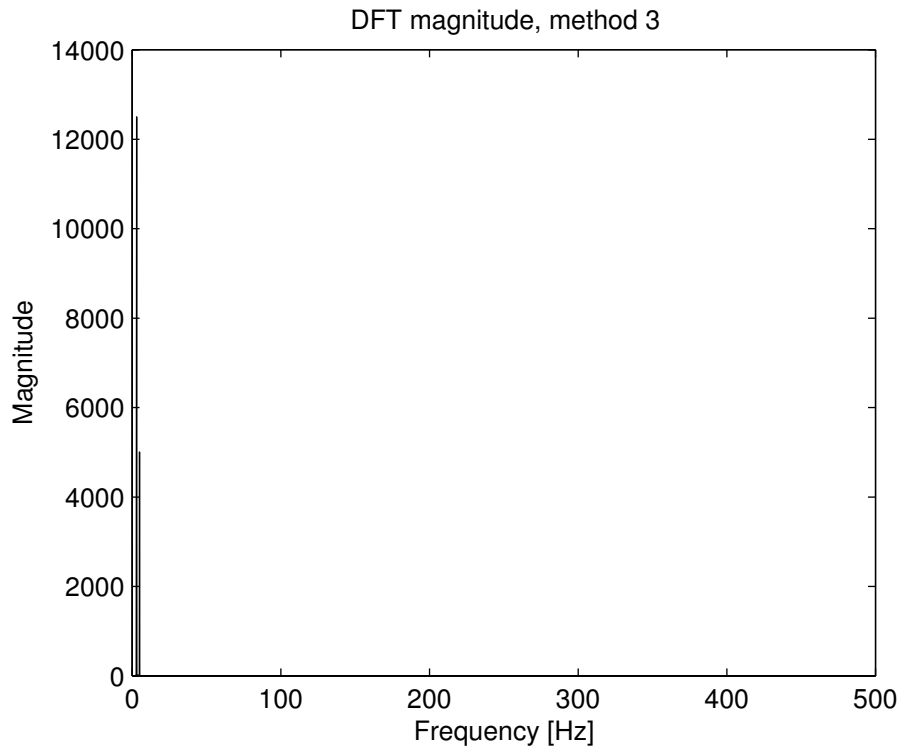


Figure 4: Magnitude spectrum of the simple signal in Figure 1, third method.

```
%
figure;
plot(f, mag_Y_pos_freq);
title('DFT magnitude, method 3b');
xlabel('Frequency [Hz]');
ylabel('Magnitude');

f_min = 0;
f_max = 10;
mag_min = 0;
mag_max = 14000;

axis([f_min f_max mag_min mag_max]);
```

There are many other topics related to plotting that will not be covered in this tutorial, including multiple plots in the same graph, and changing line styles. Use the Matlab documentation (`help plot`) to learn about these subjects.

## 4 Final remarks

This short tutorial presented an introduction to Matlab programming, operations with matrices and vectors, and plotting signals and spectra. This tutorial is not exhaustive, and it is the student's responsibility to

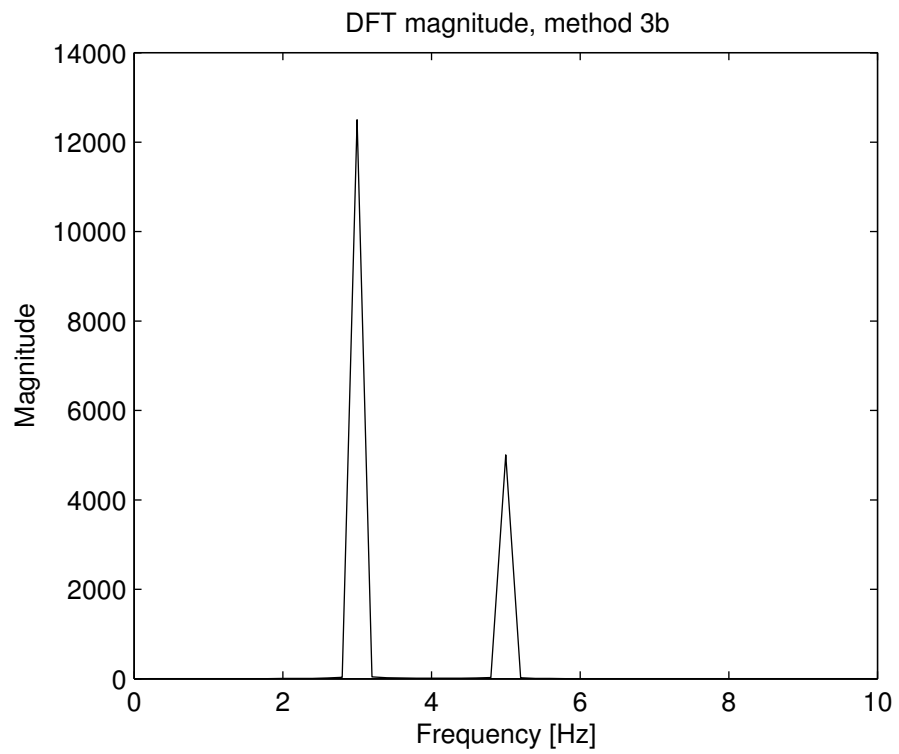


Figure 5: Magnitude spectrum of the simple signal in Figure 1. The area of interest in the graph is expanded using the `axis` command.

study Matlab further, as required.

The section on plotting graphs presented information on how to plot a magnitude spectrum appropriately. An engineer must know how to present graphical information effectively: therefore, the laboratory reports in this course will be graded on the quality of the graphs, among other criteria. The following guidelines must be followed in all of the laboratory reports:

- All graphs must have the proper axis labeling and title, and the correct abscissa values for time and frequency.
- When a signal is provided along with its sampling rate, the time and frequency axis must have the proper measurement units (time in seconds, frequency in Hertz). Multiples of the measurement units (such as millisecond, kHz, etc) must be chosen when appropriate, and the abscissa values must be modified accordingly (e.g. divide the frequency in Hertz by 1000 to express the same values in kHz).
- The graph must be informative: use the `axis` command to zoom in on regions of interest of the graph for greater clarity, if necessary. Remember that a graph is a tool for communication of technical knowledge: unintelligible graphs are useless.
- When plotting multiple functions in the same graph, use different line styles.
- Each graph must have a caption, describing the graph.
- Each graph must be referred to in the report text. The same is valid for tables, equations, and other complementary information to the report text. use cross-references to relate various plots.