Technical University of Cluj-Napoca

Faculty of Electronics, Telecommunications & Information Technology

Fast Fourier Transform and MATLAB Implementation

Student,

Gemănar Adina Alexandra

Group 2023

Second Year

Coordinator,

dr. ing. Mihaela Cîrlugea

dr. ing. Paul Farago

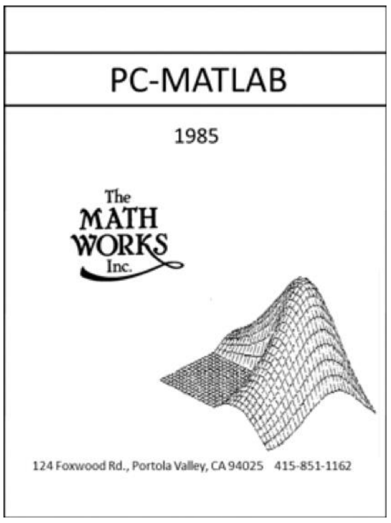
Table of contents

Introduction

A Brief History of MATLAB

The first MATLAB (the name is short for "Matrix Laboratory") was not a programming language. Written in Fortran in the late 1970s, it was a simple interactive matrix calculator built on top of about a dozen subroutines from the LINPACK and EISPACK matrix software libraries. There were only 71 reserved words and built-in functions. It could be extended only by modifying the Fortran source code and recompiling it.

The programming language appeared in 1984 when MATLAB became a commercial product. The calculator was reimplemented in C and significantly enhanced with the addition of user functions, toolboxes, and graphics. It was available initially on the IBM PC and clones; versions for Unix workstations and the Apple Macintosh soon followed.



In addition to the matrix functions from the calculator, the 1984 MATLAB included fast Fourier transforms (FFT). The Control System Toolbox appeared in 1985 and the Signal Processing Toolbox in 1987. Built-in support for the numerical solution of ordinary differential equations also appeared in 1987.

The first significant new data structure, the sparse matrix, was introduced in 1992. The Image Processing Toolbox and the Symbolic Math Toolbox were both introduced in 1993.

Several new data types and data structures, including single precision floating point, various integer and logical types, cell arrays, structures, and objects were introduced in the late 1990s.

Enhancements to the MATLAB computing environment have dominated development in recent years. Included are extensions to the desktop, major enhancements to the object and graphics systems, support for parallel computing and GPUs, and the "Live Editor", which combines programs, descriptive text, output and graphics into a single interactive, formatted document.

Today there are over 60 Toolboxes, many programmed in the MATLAB language, providing extended capabilities in specialized technical fields.

Most of the work you will do in MATLAB will be stored in files called scripts, or m-files, containing sequences of MATLAB commands to be executed over and over again.

Gauss and the history of the fast Fourier transform

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies.

The development of fast algorithms for DFT can be traced to Carl Friedrich Gauss's unpublished work in 1805 when he needed it to interpolate the orbit of asteroids Pallas and Juno from sample observations. His method was very similar to the one published in 1965 by James Cooley and John Tukey, who are generally credited for the invention of the modern generic FFT algorithm. While Gauss's work predated even Joseph Fourier's results in 1822, he did not analyze the computation time and eventually used other methods to achieve his goal.

James Cooley and John Tukey published a more general version of FFT in 1965 that is applicable when N is composite and not necessarily a power of 2. Tukey came up with the idea during a meeting of President Kennedy's Science Advisory Committee where a discussion topic involved detecting nuclear tests by the Soviet Union by setting up sensors to surround the country from outside. To analyze the output of these sensors, an FFT algorithm would be needed. In discussion with Tukey, Richard Garwin recognized the general applicability of the algorithm not just to national security problems, but also to a wide range of problems including one of immediate interest to him, determining the periodicities of the spin orientations in a 3-D crystal of Helium-3. Garwin gave Tukey's idea to Cooley (both worked at IBM's Watson labs) for implementation. Cooley and Tukey published the paper in a relatively short time of six months. As Tukey did not work at IBM, the patentability of the idea was doubted and the algorithm went into the public domain, which, through the computing revolution of the next decade, made FFT one of the indispensable algorithms in digital signal processing.

Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. The basic ideas were popularized in 1965, but some algorithms had been derived as early as 1805. In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime", and it was included in Top 10 Algorithms of 20th Century by the IEEE magazine Computing in Science & Engineering.

Theoretical presentation

A signal has one or more frequencies in it and can be viewed from two different standpoints: Time domain and Frequency domain.

Time-domain figure shows how a signal changes over time. Frequency-domain figure shows how much of the signal lies within each given frequency band over a range of frequencies.

In this project, I intended to illustrate the difference between the time domain and the frequency domain by using a tool called Fourier transform. A Fourier transform converts a signal in the time domain to the frequency domain(spectrum). An inverse Fourier transform converts the frequency domain components back into the original time domain signal.

The Fast Fourier Transform does not refer to a new or different type of Fourier transform. It refers to a very efficient algorithm for computing the DFT.

The time taken to evaluate a DFT on a computer depends principally principally on the number of multiplications multiplications involved involved. DFT needs N2 multiplications. FFT only needs Nlog2(N).

The central insight which leads to this algorithm is the realization realization that a discrete discrete Fourier Fourier transform transform of a sequence sequence of N points can be written in terms of two discrete Fourier transforms of length N/2.

Thus if N is a power of two, it is possible possible to recursively recursively apply this decomposition until we are left with discrete Fourier transforms of single points.

The Fourier transform is defined for a vector x with n uniformly sampled points by

$$y_{k+1} = \sum_{j=0}^{n-1} \omega^{jk} x_{j+1}.$$

ω=$e^{-2\pi i/n}$ is one of n complex roots of unity where i is the imaginary unit. For x and y, the indices j and k range from 0 to n−1.

The fft function in MATLAB® uses a fast Fourier transform algorithm to compute the Fourier transform of data.

The FFT operates by decomposing an N point time domain signal into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum. separate stages.

MATLAB offers many predefined mathematical functions for technical computing.

cos(x)	Cosine	abs(x)	Absolute value
sin(x)	Sine	angle(x)	Phase angle
exp(x)	Exponential	conj(x)	Complex conjugate
sqrt(x)	Square root	log(x)	Natural logarithm

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands The MATLAB command to plot a graph is plot(x,y)

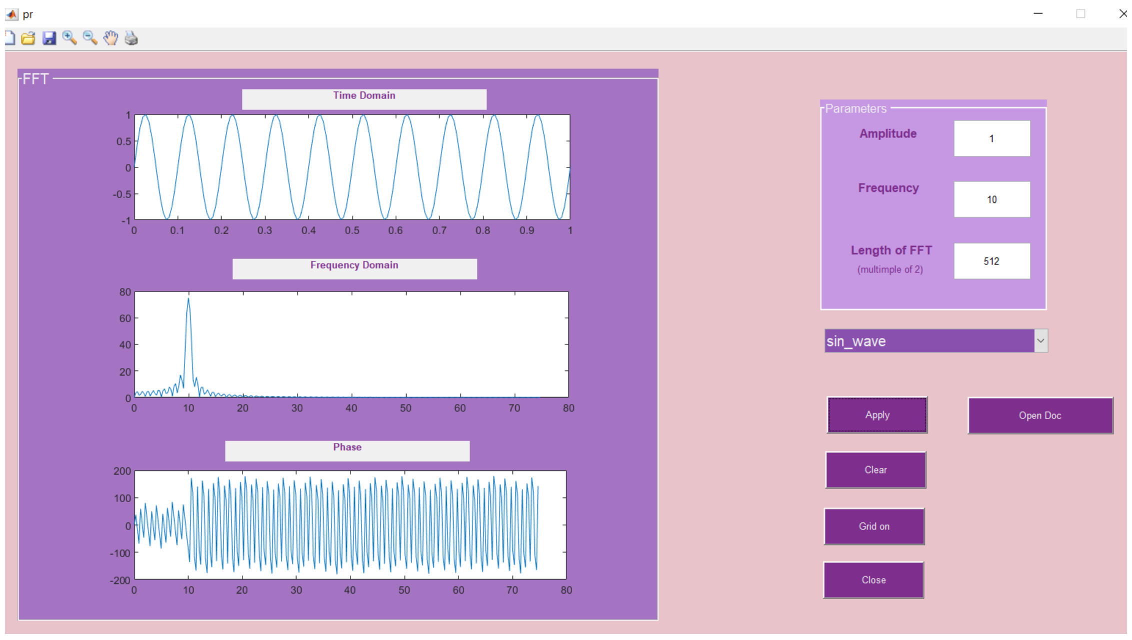
In my GUI there are 7 possible options for the type of the wave: sine, cosine, tangent, square, gaussian pulse, exponential decay, chirp signal. We can select one from the popupmenu.

I considered the frequency f , the amplitude A and the length N of the sine wave editable, in order to generate a sine wave in Matlab.

Experimental part

For example, I intend to generate a f=10 Hz sine wave whose minimum and maximum amplitudes are -1V and +1V respectively. Now that we have determined the frequency of the sinewave, the next step is to determine the sampling rate. Matlab is a software that processes everything in digital. In order to generate/plot a smooth sine wave, the sampling rate must be far higher than the prescribed minimum required sampling rate which is at least twice the frequency f – as per Nyquist Shannon Theorem.

I obtained the next figures, as I expected:



Parameters

Amplitude: 1

Frequency: 10

Length of FFT (multimple of 2): 512

sin_wave

Representing the given signal in frequency domain is done via Fast Fourier Transform (FFT) which implements Discrete Fourier Transform (DFT) in an efficient manner. Usually, power spectrum is desired for analysis in frequency domain. In a power spectrum, power of each frequency component of the given signal is plotted against their respective frequency.

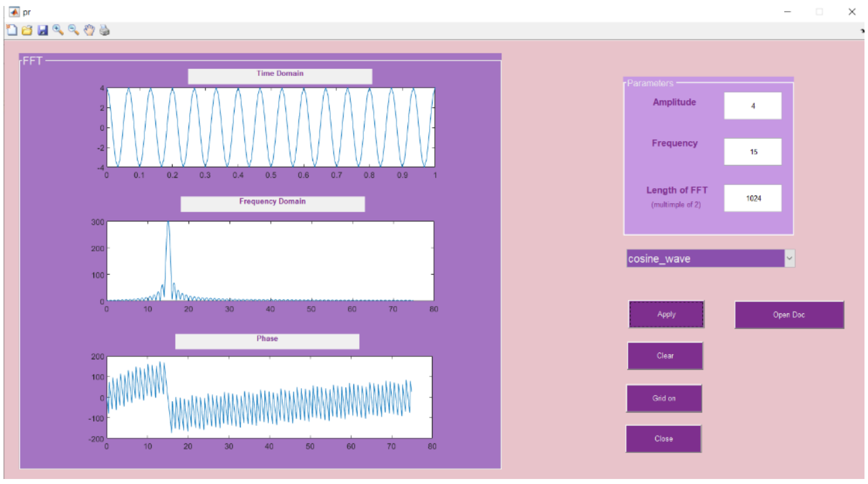
The command

```
X = fft(x,nfft);
```

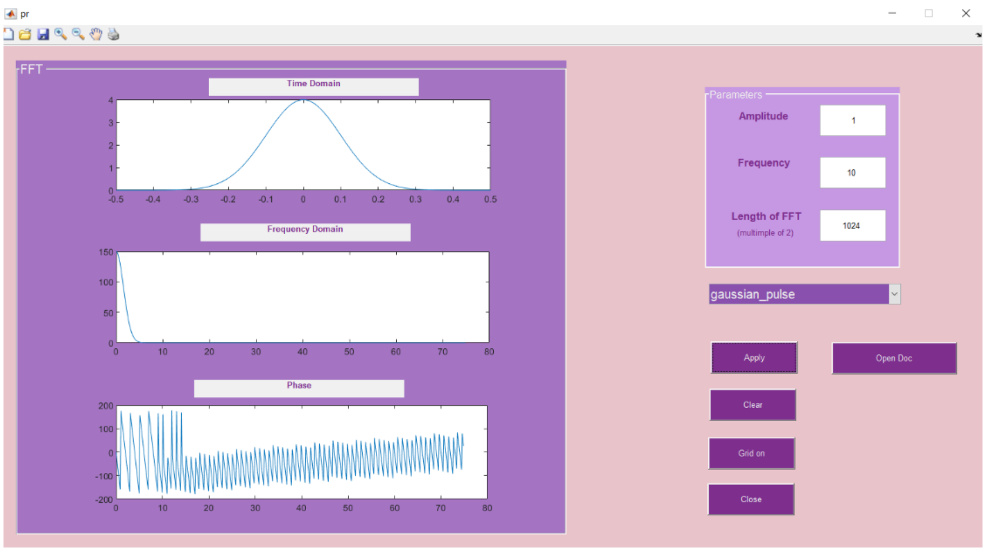
computes the N-point DFT.

The number of points in the DFT computation is taken as power of (2) for facilitating efficient computation with FFT. A value of N=512 is chosen here. It can also be chosen as next power of 2 of the length of the signal.

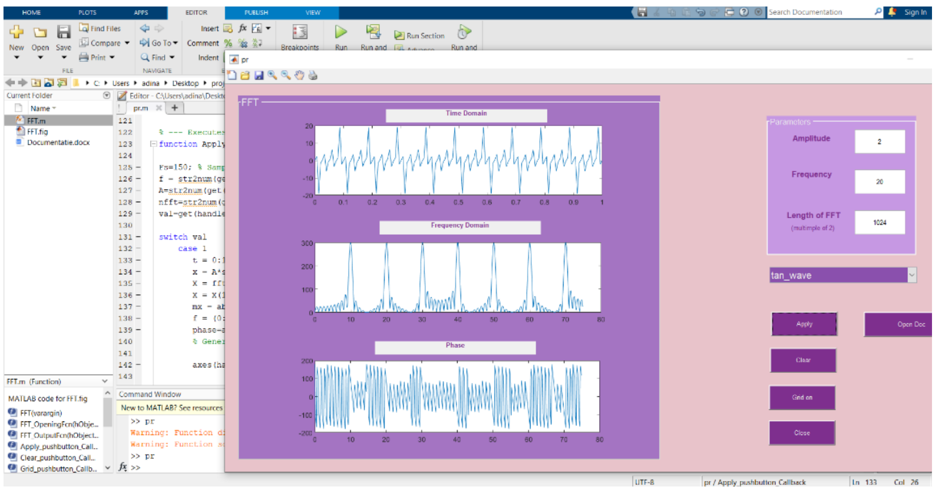
Another example is with the cosine wave. Let’s select the an amplitude of 4V, frequency 15 and N of 1024. When clicking *Apply* we are supposed to visualise the 3 graphs.



The next example is with *gaussian pulse*. The values of the parameters are 1 for the Amplitude, 10 for the frequency and 1024 for N. In the first graph we clearly see the Gaussian Pulse:



This is a picture of my interface:



```
% --- Executes
function Apply

Fs=150; % Samp
f = str2num(ge
A=str2num(get(
nfft=str2num(g
val=get(handle

switch val
    case 1
        t = 0:1
        x = A*s
        X = fft
        X = X(1
        mx = ab
        f = (0:
        phase=a
        % Gener

        axes(ha
```

```
New to MATLAB? See resources
>> pr
Warning: Function d
Warning: Function s
>> pr
>>
```

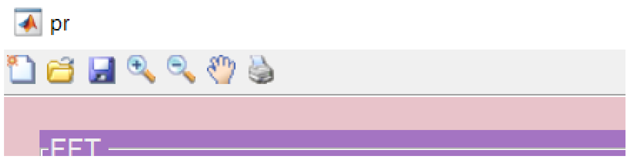
MATLAB enables you to add axis Labels and titles. I was able to do this in the created functions of the axes as follows:

```
function axes1_CreateFcn(hObject, eventdata, handles)
xlabel('Time(s)')
ylabel('Amplitude(V)')

function axes2_CreateFcn(hObject, eventdata, handles)
xlabel('Frequency(Hz)')
ylabel('Amplitude(db)')

function axes3_CreateFcn(hObject, eventdata, handles)
xlabel('Frequency(Hz)')
ylabel('Phase(deg)')
```

If you make a GUI in GUIDE, by default the toolbar and menubar are hidden. I made them visible by setting the 'MenuBar' and 'Toolbar' properties of the GUI figure:



I added to my figure the *New, Opne, Save, Zoom in, Zoom out, Pan* and *Print* tools.

My designed GUI containes 5 pushbuttons, each one with a different role, 2 editable text boxes, a pop-up menu, 2 axes, 2 panels, and 4 static text boxes.

I will explain each graphical component that appear, what it is and what role it has inside the project.

Edit boxes are designed to make the connection between the user and the code. Here we can insert the values for the frequency and amplitude.

Another pushbutton is the pop-up menu. Here we can select the type of the sinewave that has to be generated. Our options are sin wave, cos and tangent. Also, the next callback function has the role to execute the selection change in the pop-up menu:

```
function popupmenu1_Callback(hObject, eventdata,
handles)
val=get(handles.popupmenu1,'Value');
handles.v=val;
guidata(hObject,handles)
```

The *Apply* button is the principal graphical component because it has the role of calculating and ploting the signals, in both time and frequency domain.

The callback function of this button contains the next code:

```
switch val
    case 1
       t = 0:1/Fs:1;  % Time vector of 1 second
       x = A*sin(2*pi*t*f);
       X = fft(x,nfft);
       X = X(1:nfft/2); % FFT is symmetric, throw away
second half
       mx = abs(X); % Take the magnitude of fft of x
       f = (0:nfft/2-1)*Fs/nfft; % Frequency vector
       phase=angle(X)*180/pi; %phase spectrum (in deg.)
       % Generate the plot, title and labels.

       axes(handles.axes1);

       plot(t,x );                %plots signal(sin) in time
domain
       axes(handles.axes2);
       plot(f,mx);   %plots signal (sin) in freq domain &
% Take the magnitude of fft
       axes(handles.axes3);
       plot(f,phase); %plots phase

    case 2
       t = 0:1/Fs:1;
       x = A*cos(2*pi*t*f);
       X = fft(x,nfft);
       X = X(1:nfft/2);
       mx = abs(X);
       f = (0:nfft/2-1)*Fs/nfft;
       phase=angle(X)*180/pi; %phase spectrum (in deg.)
       axes(handles.axes1);
       plot(t,x);
       axes(handles.axes2);
       plot(f,mx);
       axes(handles.axes3);
       plot(f,phase);
    case 3
       t = 0:1/Fs:1;
       x = A*tan(2*pi*t*f);
       X = fft(x,nfft);
       X = X(1:nfft/2);
       mx = abs(X);
       f = (0:nfft/2-1)*Fs/nfft;
       phase=angle(X)*180/pi; %phase spectrum (in deg.)
       axes(handles.axes1);
```

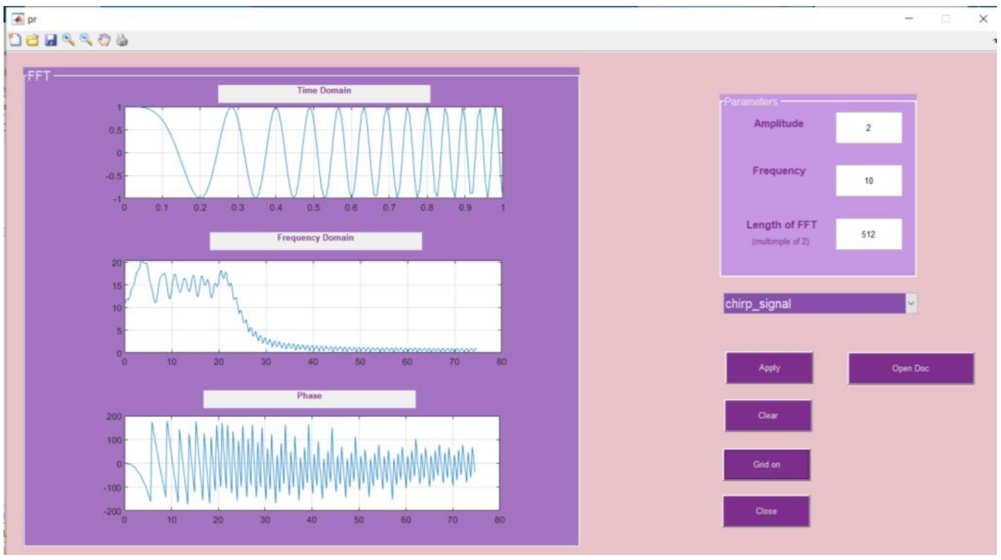
```
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 4
        t = -.5:1/Fs:.5;
        x =A*( 1/(sqrt(2*pi*0.01))*(exp(-t.^2/(2*0.01))));
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 5
        t = 0:1/Fs:1;
        x = A*sign(sin(2*pi*t*f));
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 6
        t = 0:1/Fs:1; % Time vector of 1 second
        x = 2*exp(-5*t);
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
     case 7
         t = 0:1/Fs:1; % Time vector of 1 second
         x = chirp(t,0,1,Fs/6);
         X = fft(x,nfft);
         X = X(1:nfft/2);
         mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);

 end
```

Switch switch_expression, case case_expression, end evaluates an expression and chooses to execute one of several groups of statements. Each choice is a case. When a case expression is true, MATLAB® executes the corresponding statements and exits the switch block.

I used this expression because in the pop-up menu I added 3 options for the type of the wave. That's why we need a piece of code for each.

Once *Grid on* button is pushed, it adds a grid to the graphs:



Amplitude 2

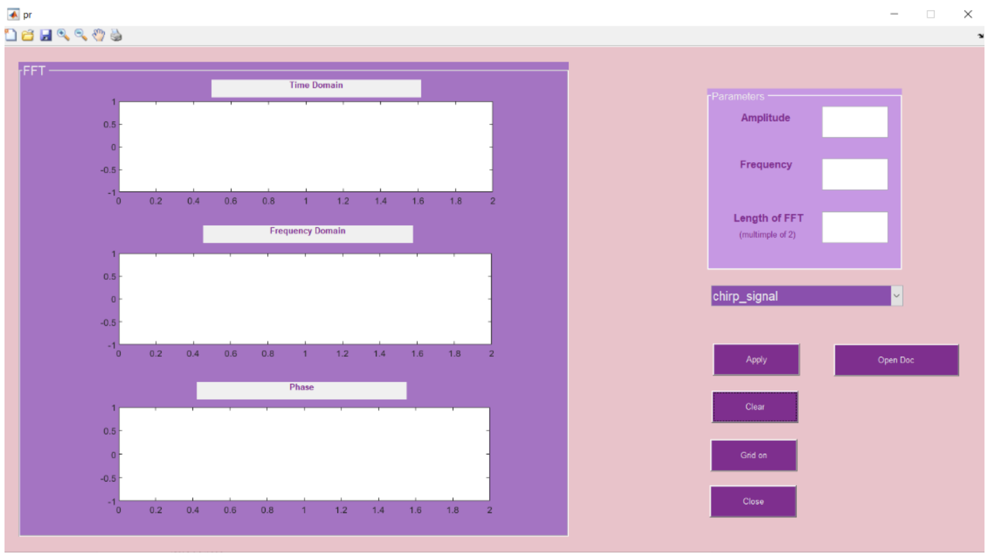
Frequency 10

Length of FFT (multimple of 2) 512

chirp_signal

The *Open* pushbutton opens the written pdf documentation, that we are able to do with the function `winopen`

If the *Clear* button is pressed, we can start again and add another values for the frequency and amplitude in order to obtain the waves.



For the *Close* button all I did was to add the word *close* to the callback function of the pushbutton. It has the role of closing the window.

```
function Close_pushbutton_Callback(hObject,
eventdata, handles)
close
```

Conclusion

The basic objective of this project is to use the Fast Fourier transform to convert a signal from the time domain to the frequency domain(spectrum) and its phase spectrum and it was successfully build using MATLAB.

Based on the fundamental theory of the Fourier transform, the code is arranged so we can obtain 3 graphs at the same time, with the possibility of choosing the type of wave, and also inserting the 3 parameters of the signal, the amplitude, the frequency and the length.

At a closer look, some improvements could be attributed in the future to this project, after further research and development. It is sometimes convenient to rearrange the output of the fft or fft2 function so the zero frequency component is at the center of the sequence. Also, the popupmenu could be improved by adding more options.

Bibliography

- https://dl.acm.org/doi/10.1145/3386331

- https://en.wikipedia.org/wiki/Fast_Fourier_transform

- E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. 1999. LAPACK Users' Guide (third ed.). Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, USA.

- https://www.mathworks.com/help/matlab/ref/fft.html

- https://personal.utdallas.edu/~dlm/3350%20comm%20sys/FFTandMatLab-wanjun%20huang.pdf

Appendix

```
function varargout = pr(varargin)
% PR MATLAB code for pr.fig
%      PR, by itself, creates a new PR or raises the
existing
%      singleton*.
%
%      H = PR returns the handle to a new PR or the
handle to
%      the existing singleton*.
%
%      PR('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in PR.M with the given
input arguments.
%
%      PR('Property','Value',...) creates a new PR or
raises the
%      existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before pr_OpeningFcn gets
called.  An
%      unrecognized property name or invalid value
makes property application
%      stop.  All inputs are passed to pr_OpeningFcn
via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.
Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
pr

% Last Modified by GUIDE v2.5 06-Jan-2021 14:45:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton,
...
                   'gui_OpeningFcn', @pr_OpeningFcn,
...
```

```
                   'gui_OutputFcn',  @pr_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before pr is made visible.
function pr_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)
% varargin   command line arguments to pr (see
VARARGIN)

% Choose default command line output for pr
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes pr wait for user response (see
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the
command line.
function varargout = pr_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args
(see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)

% Get default command line output from handles
structure
varargout{1} = handles.output;



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)

% Hints: get(hObject,'String') returns contents of
edit1 as text
%        str2double(get(hObject,'String')) returns
contents of edit1 as a double


% --- Executes during object creation, after setting
all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
```

```
% handles    structure with handles and user data
(see GUIDATA)

% Hints: get(hObject,'String') returns contents of
edit2 as text
%        str2double(get(hObject,'String')) returns
contents of edit2 as a double


% --- Executes during object creation, after setting
all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: edit controls usually have a white background
on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in Apply_pushbutton.
function Apply_pushbutton_Callback(hObject,
eventdata, handles)

Fs=150; % Sampling frequency
f = str2num(get(handles.edit1, 'string'));
%frequency from the edit box
A=str2num(get(handles.edit3, 'string')); %amplitude
from the edit box
nfft=str2num(get(handles.edit4, 'string')); %lenght
of FFT from the edit box
val=get(handles.popupmenu1, 'Value');

switch val
    case 1
       t = 0:1/Fs:1;  % Time vector of 1 second
       x = A*sin(2*pi*t*f);
       X = fft(x,nfft);
       X = X(1:nfft/2); % FFT is symmetric, throw
away second half
       mx = abs(X); % Take the magnitude of fft of x
```

```
        f = (0:nfft/2-1)*Fs/nfft; % Frequency vector
        phase=angle(X)*180/pi; %phase spectrum (in
deg.)
        % Generate the plot, title and labels.

        axes(handles.axes1);

        plot(t,x );              %plots signal(sin) in
time domain
        axes(handles.axes2);
        plot(f,mx);   %plots signal (sin) in freq
domain & % Take the magnitude of fft
        axes(handles.axes3);
        plot(f,phase); %plots phase

    case 2
        t = 0:1/Fs:1;
        x = A*cos(2*pi*t*f);
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in
deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 3
        t = 0:1/Fs:1;
        x = A*tan(2*pi*t*f);
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in
deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 4
        t = -.5:1/Fs:.5;
        x =A*( 1/(sqrt(2*pi*0.01))*(exp(-
t.^2/(2*0.01))));
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in
deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 5
        t = 0:1/Fs:1;
        x = A*sign(sin(2*pi*t*f));
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in
deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 6
        t = 0:1/Fs:1; % Time vector of 1 second
        x = 2*exp(-5*t);
        X = fft(x,nfft);
        X = X(1:nfft/2);
        mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in
deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);
    case 7
         t = 0:1/Fs:1; % Time vector of 1 second
         x = chirp(t,0,1,Fs/6);
         X = fft(x,nfft);
         X = X(1:nfft/2);
         mx = abs(X);
        f = (0:nfft/2-1)*Fs/nfft;
        phase=angle(X)*180/pi; %phase spectrum (in
 deg.)
        axes(handles.axes1);
        plot(t,x);
        axes(handles.axes2);
        plot(f,mx);
        axes(handles.axes3);
        plot(f,phase);

 end

 % hObject    handle to Apply_pushbutton (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    structure with handles and user data
 (see GUIDATA)


 % --- Executes on button press in Clear_pushbutton.
 function Clear_pushbutton_Callback(hObject,
 eventdata, handles)
 set(handles.edit1, 'string', ' ');
 set(handles.edit3, 'string', ' ');
 set(handles.edit4, 'string', ' ');
 axes(handles.axes1);
 plot(0);
 axes(handles.axes2);
 plot(0);
 axes(handles.axes3);
 plot(0);
 % hObject    handle to Clear_pushbutton (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    structure with handles and user data
 (see GUIDATA)


 % --- Executes on selection change in popupmenu1.
 function popupmenu1_Callback(hObject, eventdata,
 handles)
 val=get(handles.popupmenu1,'Value');
 handles.v=val;
 guidata(hObject,handles);
 % hObject    handle to popupmenu1 (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    structure with handles and user data
 (see GUIDATA)

 % Hints: contents = cellstr(get(hObject,'String'))
 returns popupmenu1 contents as cell array
 %        contents{get(hObject,'Value')} returns
 selected item from popupmenu1


 % --- Executes during object creation, after setting
 all properties.
 function popupmenu1_CreateFcn(hObject, eventdata,
 handles)
 % hObject    handle to popupmenu1 (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    empty - handles not created until after
 all CreateFcns called

 % Hint: popupmenu controls usually have a white
 background on Windows.
 %       See ISPC and COMPUTER.
 if ispc && isequal(get(hObject,'BackgroundColor'),
 get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
 end



 function edit3_Callback(hObject, eventdata, handles)
 % hObject    handle to edit3 (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    structure with handles and user data
 (see GUIDATA)

 % Hints: get(hObject,'String') returns contents of
 edit3 as text
 %        str2double(get(hObject,'String')) returns
 contents of edit3 as a double


 % --- Executes during object creation, after setting
 all properties.
 function edit3_CreateFcn(hObject, eventdata, handles)
 % hObject    handle to edit3 (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    empty - handles not created until after
 all CreateFcns called

 % Hint: edit controls usually have a white background
 on Windows.
 %       See ISPC and COMPUTER.
 if ispc && isequal(get(hObject,'BackgroundColor'),
 get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
 end



 function edit4_Callback(hObject, eventdata, handles)
 % hObject    handle to edit4 (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    structure with handles and user data
 (see GUIDATA)

 % Hints: get(hObject,'String') returns contents of
 edit4 as text
 %        str2double(get(hObject,'String')) returns
 contents of edit4 as a double


 % --- Executes during object creation, after setting
 all properties.
 function edit4_CreateFcn(hObject, eventdata, handles)
 % hObject    handle to edit4 (see GCBO)
 % eventdata  reserved - to be defined in a future
 version of MATLAB
 % handles    empty - handles not created until after
 all CreateFcns called

 % Hint: edit controls usually have a white background
 on Windows.
 %       See ISPC and COMPUTER.
 if ispc && isequal(get(hObject,'BackgroundColor'),
 get(0,'defaultUicontrolBackgroundColor'))
     set(hObject,'BackgroundColor','white');
 end


 % --- Executes on button press in Close_pushbutton.
```

```
function Close_pushbutton_Callback(hObject,
eventdata, handles)
close
% hObject    handle to Close_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)


% --- Executes on button press in Grid_pushbutton.
function Grid_pushbutton_Callback(hObject, eventdata,
handles)
grid on
% hObject    handle to Grid_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)


% --- Executes on button press in Open_pushbutton.
function Open_pushbutton_Callback(hObject, eventdata,
handles)
winopen('Documentatie.docx');
% hObject    handle to Open_pushbutton (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)


% --- Executes on mouse press over axes background.
function axes1_ButtonDownFcn(hObject, eventdata,
handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    structure with handles and user data
(see GUIDATA)


% --- Executes during object creation, after setting
all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
xlabel('Time(s)')
ylabel('Amplitude(V)')
% hObject    handle to axes1 (see GCBO)
```

```matlab
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1


% --- Executes during object creation, after setting
all properties.
function axes2_CreateFcn(hObject, eventdata, handles)
xlabel('Frequency(Hz)')
ylabel('Amplitude(db)')
% hObject    handle to axes2 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: place code in OpeningFcn to populate axes2


% --- Executes during object creation, after setting
all properties.
function axes3_CreateFcn(hObject, eventdata, handles)
xlabel('Frequency(Hz)')
ylabel('Phase(deg)')
% hObject    handle to axes3 (see GCBO)
% eventdata  reserved - to be defined in a future
version of MATLAB
% handles    empty - handles not created until after
all CreateFcns called

% Hint: place code in OpeningFcn to populate axes3
```