Computational Physics III: Report 1
Fourier transforms and analysis

Due on March 28, 2019

Nicolas Lesimple

Contents

Problem 1

The Fourier transform **relates real and reciprocal space representations of a function**. While the conventional Fourier transform operates with continuous functions f(x), the discrete Fourier transform (DFT) operates with discretized data f_n,n $\in N$.

The Fourier transform (FT) decomposes a function of time (a signal) into frequencies. The FT of a function of time is itself a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, and whose complex argument is the phase offset of the basic sinusoid in that frequency. This transformation is called the frequency domain representation of the original signal.

In mathematics, **the discrete Fourier transform (DFT) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time** Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. **The DFT is therefore said to be a frequency domain representation of the original input sequence.** If the original sequence spans all the non-zero values of a function, its DTFT is continuous (and periodic), and the DFT provides discrete samples of one cycle. If the original sequence is one cycle of a periodic function, the DFT provides all the non-zero values of one DTFT cycle. One of the biggest advantages is that since it deals with a finite amount of data, **it can be implemented in computers by numerical algorithms or even dedicated hardware.** These implementations usually employ efficient fast Fourier transform (FFT) algorithms.

In the first problem of this assignment, **a function which implements the discrete Fourier transform (DFT) had to be designed.** The core of the function is described in Listing 1 just below. The implementation was done thanks to the definition of the Fourier transform that has been applied thanks to a for loop : $\hat{f}[m] = \sum_{n=0}^{N-1} f[n] e^{\frac{i*2\pi*m*n}{N}}$. To complete the process, N*N multiplications and N*(N-1) additions are needed. Moreover, different possible input arrays were handled by transposing the input array if this one was a column vector, or by throwing an error if the input had a 2D shape or even more dimension.

Listing 1: A Matlab script which does the Discrete Fourier transform calculation of Problem 1

```
function result = mydft(input_array)
% This function called mydft computes the discrete Fourier transform of a 1D signal.
% Arguments :
%   - input_array (1D array complex) : data to transform ;
% Returns :
%   - 1D array complex, transformed data of the same shape as an input array .

% We check the format of the input to allow line or column vector
input_size = size(input_array);
if input_size(1)> 1
    input_array = transpose(input_array);
end

% We check if the input array is 1D. If not, an error is throw.
nD = size(input_size);
if (min(input_size)>1)~=0 || (nD(2)>2)~=0
    msg = 'The input array is in 2D or more.';
    error(msg);
end

% We first declare the variables needed to solve the problem
N = length(input_array);
result = zeros(1, N);
Sum=0;
```

```
% We apply the calculation algorithm of the DFT
for k=1:N
    for l=1:N
        Sum=Sum+input_array(l)*exp(-2*pi*1i*(l-1)*(k-1)/N);
    end
    result(k)=Sum;
Sum=0;
end

% If the input was not in the optimal format, we change it during the
% process and thus there we give him back is original form.
if input_size(1)> 1
    result = transpose(result);
end
return
```

Then, **a verification of the DFT implementation was done.** In fact, the concept of testing was applied by transforming simple model functions and comparing the results with MatLab 's function *fft()*. A convenient way to do that is to use the assert function of Matlab. The script test.m was supplied (see Listing 2). By running the test and see that all the tests were passed, we make sure that the implementation handled correctly different possible input arrays and that the DFT calculation is right. **In addition, an additional test was added to check if our function allows well wrong input, like multi-dimensional input_array.** In fact, in that case, an error is raised. Test 4 illustrates the proper functioning of this module. The all test script can be found in the Listing2 just below.

Listing 2: A script which does the test of the Discrete Fourier transform implementation of Problem 1

```
% This script tests mydft.m for correctness
tic

fprintf('Test 1: Gaussian ...');
sample = exp(-linspace(-4,4,100).^2);
assert(all(abs(mydft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 1.1: Gaussian with a different Matlab dimension ...');
sample = exp(-linspace(-4,4,100).^2)';
assert(all(abs(mydft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 1.2: Gaussian complex ...');
sample = 1i*exp(-linspace(-4,4,100).^2);
assert(all(abs(mydft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 2: sawtooth ...');
sample = linspace(-1,1,100);
assert(all(abs(mydft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 3: sin and sin2 ...');
sample = sin(linspace(-pi,pi,100));
assert(all(abs(mydft(sample) - fft(sample))<1e-10));
sample = sin(2*linspace(-pi,pi,100));
assert(all(abs(mydft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');
```

```
30
31 fprintf('Test 4: 2D input\n');
32 sample = sin(linspace(-pi,pi,100));
33 sample = [sample, sample];
34 testCase = matlab.unittest.TestCase.forInteractiveUse;
35 verifyError(testCase,@() dft(sample),'MATLAB:UndefinedFunction');
36
37 fprintf('\nAll tests passed!\n');
38
39 timeElapsed = toc;
40 fprintf('\nTime to pass all the tests : ')
41 disp(timeElapsed)
```

```
>> test_dft
Test 1: Gaussian ...    passed
Test 1.1: Gaussian with a different Matlab dimension ...        passed
Test 1.2: Gaussian complex ...  passed
Test 2: sawtooth ...    passed
Test 3: sin and sin2 ...        passed
Test 4: 2D input
Verification passed.

All tests passed!

Time to pass all the tests :     1.3610
```

Figure 1: Illustration of the testing's function output.

As a conclusion, the Figure 1 **illustrate the fact that the Discrete Fourier transform calculation with Matlab was implemented in mydft.m script and that the module passed all the tests.** Thus, the Discrete Fourier Transform module should work properly and **it implies that this code is reliable.**

Problem 2

In Fourier transform nuclear magnetic resonance spectroscopy, **free induction decay (FID) is the observable NMR signal generated by non-equilibrium nuclear spin magnetization precessing about the magnetic field (conventionally along z).** Specifically, nuclear spins are excited by means of a radio frequency pulse and the signal due to their relaxation is recorded. This non-equilibrium magnetization can be induced, generally by applying a pulse of resonant radio-frequency.

If the magnetization vector has a non-zero component in the xy plane, then theprecessing magnetization will induce a corresponding oscillating voltage in a detection coil surrounding the sample. **This time-domain signal is typically digitized and then Fourier transformed in order to obtain a frequency spectrum of the NMR signal.** The transitions between the quantum states of nuclear spins of protons in an applied magnetic field are probed to determine the structure of molecules and solids. mThis this exactly what we need to do in this problem. In fact, the Fourier transform of the FID results to the NMR spectrum, that is the intensity of the signal as a function of the chemical shift. Nuclear spins in different chemical environments would correspond to different peak positions, where the area of the peaks in the spectrum is proportional to the number of protons of the corresponding type.

(1)

The columns in the data file FID.dat contain the real and imaginary part of the free induction decay of a molecule with formula $C_2H_6O_2$. Importation of the data were made using *importdata()* and *complex()* Matlab function. **First, importation of the file in a two-column matrix was done and then the merge of the real and imaginary parts into a single complex vector was achieve.** It gives an array composed of complex numbers.

(2)

The time interval between two consecutive data points is $\Delta t = 83.2\mu s$. **Association of the time to each data point was made by creating time vector with the same shape, where the order and thus indirectly the index allowed to link the two array.** The setting of the initial time was put to zero. It means that for the n_{th} point we have $t_n = n * \Delta t$. In Listing 3, the code allowing the import of the data and the time vector creation is presented.

Listing 3: Small subpart of the code of Exercice 2 allowing the import of the data and the time vector creation :

```
% Question 1 : Import the file in a two-column matrix and merge the real
% and imaginary parts into a single complex vector.
data = importdata('./FID.dat.txt');
data_complex = complex(data(:,1), data(:,2));

% Question 2 : Associate time to each data point, setting the initial
% time to zero.
delta_t = 83.2*10^-6;
data_size = size(data_complex);
data_size = data_size(1,1);
time_list = linspace(0,data_size,data_size)'*delta_t;
```

The real and imaginary parts of the free induction decay were plotted as a function of time. It allows us to check if the theorical decay exists. In fact, **Free induction decay (FID)** refers a short-lived sinusoidal electromagnetic signal which appears immediately following the 90 degree pulse. It was induced in the receiver coil by the rotating component of the magnetization vector in the x-y plane which was crossing the coil loops perpendicularly. This sinusoidal signal

Problem 2 [(2)] continued on next page...

should decays exponentially with a time constant. **As the Figure 3 shows, we succeed to obtain this theoretical exponential decay.**

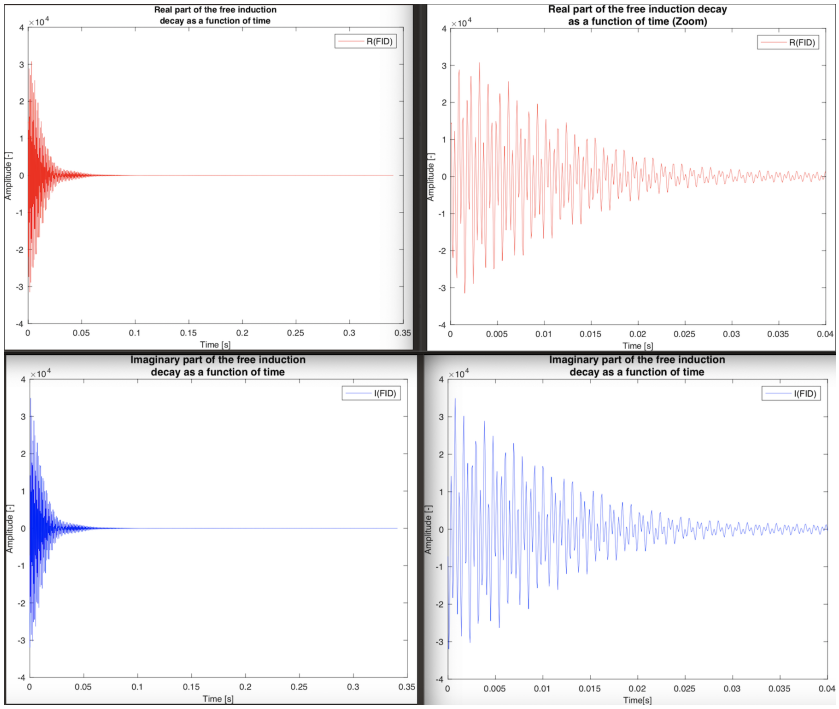


Figure 2: **Illustration of the Free induction decay coming from experimental data**
TopLeft: Real part of the FID - TopRight: Zoom of real part FID
BottomLeft: Imaginary part of the FID - BottomRight: Zoom of imaginary part FID

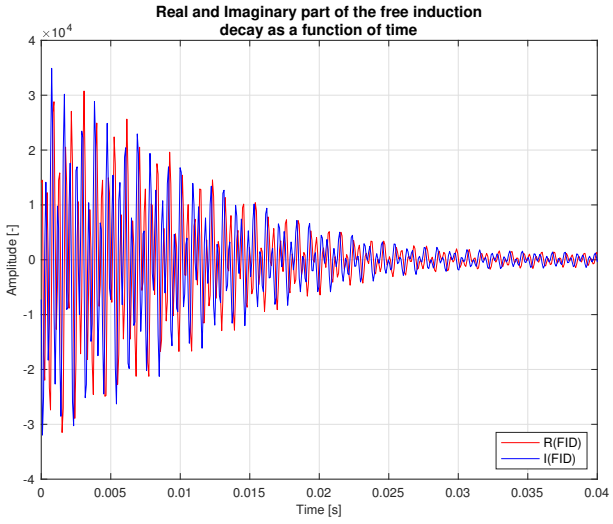


Figure 3: **Illustration of the Free induction decay coming from experimental data:** Superposition of the 2 previous zoomed graphs

(3)

Calculation of the discrete frequencies f_n was done by definition : in fact, the discrete frequencies f_n spans the interval $[f_c\ ,\ f_c]$ with f_c being the Nyquist frequency. F_c, the Nyquist frequency, is define

with this equation : $f_c = \frac{1}{2*\Delta t} = 6.0096e+3$. **The Nyquist frequency is a type of sampling frequency, used a lot in signal processing, that is defined as half of the rate of a discrete signal processing system. It is the highest frequency that can be coded for a particular sampling rate so that the signal can be reconstructed.** It is based on the Nyquist Theory, which applies to many different fields where data is captured. In general terms, the Nyquist Theory is the minimum number of resolution elements required to properly describe or sample a signal. In order to reconstruct (interpolate) a signal from a sequence of samples, sufficient samples must be recorded to capture the peaks and the trough of the original waveform.

(4)

Using this frequency vector f_n, the chemical shift δ_n (in ppm), was calculated. It's defined thanks to the following formula : $\delta_n = \frac{(f_n - f_0)*10^6}{\nu_0}$ with $f_0 = 1067.93Hz$ being the internal reference and $\nu_0 = 800.224MHz$ being the operating frequency of NMR spectrometer. Thus one chemical shift can be linked to on frequency.

In a second time, the calculation of the discrete Fourier transform of the complex FID was performed using the previously implemented function *mydft()*. Even if, the function passed all the tests previously, comparison of the result of the Fourier transform obtained using *mydft()* function with the *fft()* function of MatLab was completed in two ways :

- **Assert function** : The same procedure as in the previous tests was completed with the assert function. The threshold of this assertion has been set to 10^{-7}. Thus, the two Fourier transforms have the same values if we accept an error threshold of 10^{-7}.
- **Relative error** : A plot of the relative error in function of the frequency of each point was done. The relative error was calculated in the following way : $relative\_error = abs(\frac{mydft - fft}{fft})$. The plot in figure 4 illustrates that the relative error stays low, as the order of magnitude of the error is 10^{-12}. Moreover, an increase of the error can be observed around the -1000Hz and 5000Hz. By converting these two frequencies into ppm values, the matching between the frequencies where the relative error increase and between the peaks visible in the Power spectrum of the DFT is perfect. Thus, it means that, at these frequencies, the values of the DFT increased and became extreme. Matlab function should have a nicer and more complex algorithm to deal with the kind of data points allowing a better accuracy compare to us, as our DFT algorithm was made with a naive approach. Thus it makes sense.

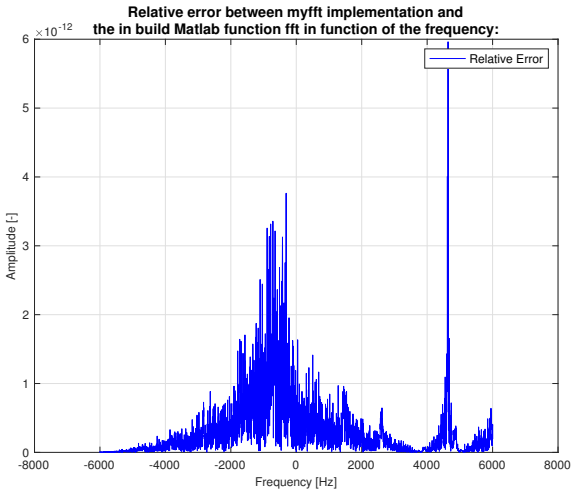


Figure 4: Relative error between our implemented algorithm of the fft calculation named mydft() and the in-build Matlab function fft()

In a third time, **the power spectrum of the Discrete Fourier Transform was calculated.** The power spectrum of a time series describes the distribution of power into frequency components composing that signal. According to Fourier analysis, any physical signal can be decomposed into a number of discrete frequencies, or a spectrum of frequencies over a continuous range. The statistical average of a certain signal or sort of signal (including noise) as analyzed in terms of its frequency content, is called its spectrum. When the energy of the signal is concentrated around a finite time interval, especially if its total energy is finite, one may compute the power spectrum, which applies to signals existing overall time. This refers to the spectral energy distribution that would be found. The power spectrum was computed thanks to the following formula: $PS = abs(\frac{FFT(signal)^2}{n})$, where n is the number of data points and signal correspond to the experimental measures. The code summing up the several processes made in question 3 and 4 is showed in Listing 4 :

Listing 4: Code allowing computation of f_n and f_c and the chemical shift and the discrete Fourier transform of the complex FID

```
% Question 3: Calculate the discrete frequencies fn which span the
% interval [-fc, fc] with fc = 1/2delat_t being the Nyquist frequency.
fc = (1/(2*delta_t));
fn = linspace(-fc,fc,data_size);

% Question 4 : calculate the chemical shift (in ppm) and plot the
% asked graphs
f0 = 1067.93;
mu0 = 800.224*10^6;
dirac_n = ((fn - f0)*10^6)/(mu0);

% Question 4 : Perform the discrete Fourier transform of the complex FID.
mydft_fid = mydft(data_complex);
fft_fid = fft(data_complex);

% Question 4 : Compare the result of the Fourier transform obtained
% with the fft function of MatLab.
dif = abs((mydft_fid - fft_fid)./fft_fid);
assert(all(abs(mydft_fid - fft_fid)<1e-6));
fprintf('\tCheck up for mydft passed\n');

% Question 4 : Calculus of the power spectrum using an existing tool
power_spectrum = (abs(mydft_fid).^2)/data_size;
```

The last part of this question was the creation of different plots. **All the plots in this entire report were made with similar code to the one described in Listing 5. It allows a nice printing of graph with labeled axis, title, legend and allows the saving in .eps format.**

Listing 5: Small subpart of the code of Exercice 2 allowing the creation of one plot.

```
figure(1)
plot(time_list, real(data_complex), '-r', 'LineWidth', 0.1)
title1 = title({'Real part of the free induction',...
            'decay as a function of time'});
set(title1,'FontName','Arial','FontSize',10);
leg1 = legend('R(FID)','Location','NorthEast');
set(leg1,'FontName','Arial','FontSize',10)
xlabel('Time [s]', 'FontName', 'Arial', 'FontSize', 10);
ylabel('Amplitude [-]','FontName','Arial','FontSize',10);
grid on;
hold off
```

```
filename='./plot/question_2_real_part.eps';
print(gcf,'-depsc',filename)
```

The resulting plots of the real part, the imaginary part (Figure 5) and the power spectrum (Figure 6) of the Discrete Fourier Transform of the FID as a function of the chemical shift δ_n (in ppm) will be analyzed in more details in the following subsections.

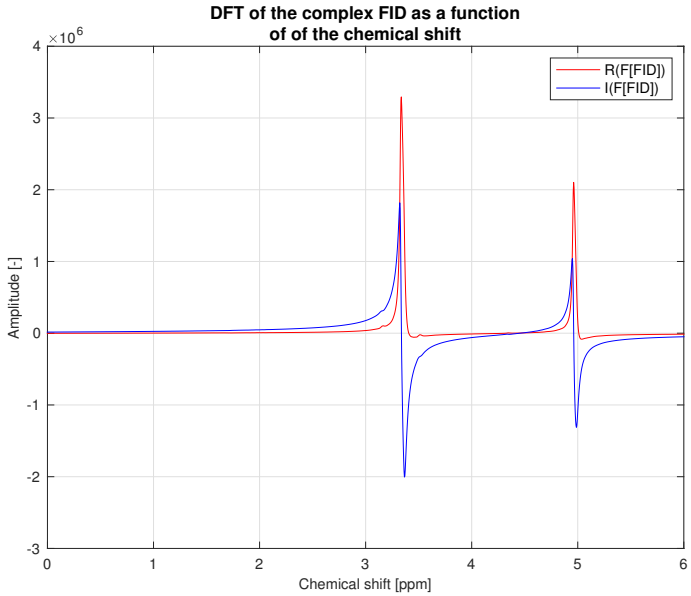


Figure 5: Real and Imaginary part of the Discrete Fourier Transform of the Free induction decay coming from experimental data in function of the chemical shift

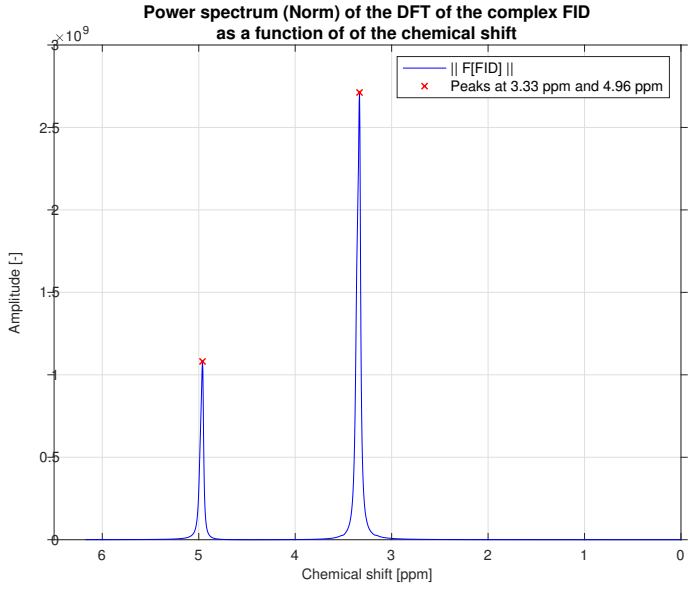


Figure 6: **Power Spectrum** of the Discrete Fourier Transform of the Free induction decay coming from experimental data in function of the chemical shift

(5)

Nuclear Magnetic Resonance (NMR) spectroscopy is an analytical chemistry technique used in quality control and research for determining the content and purity of a sample as well as its molecular structure. For example, NMR can quantitatively analyze mixtures containing known compounds. **For unknown compounds, NMR can either be used to match against spectral libraries or to infer the basic structure directly. Once the basic structure is known, NMR can be used to determine molecular conformation in solution as well as studying physical properties at the molecular level.** The principle behind NMR is that many nuclei have spin and all nuclei are electrically charged. If an external magnetic field is applied, an energy transfer is possible between the base energy to a higher energy level (generally a single energy gap). The energy transfer takes place at a wavelength that corresponds to radio frequencies and when the spin returns to its base level, energy is emitted at the same frequency. The signal that matches this transfer is measured in many ways and processed in order to yield an NMR spectrum for the nucleus concerned.

The precise resonant frequency of the energy transition is dependent on the effective magnetic field at the nucleus. This field is affected by electron shielding which is in turn dependent on the chemical environment. As a result, information about the nucleus' chemical environment can be derived from its resonant frequency. Chemical shift is associated with the Larmor frequency of a nuclear spin to its chemical environment. It is important to understand the trend of a chemical shift in terms of NMR interpretation: The proton NMR chemical shift is affected by nearness to electronegative atoms (O, N, halogen.) and unsaturated groups (C=C, C=O, aromatic). Electronegative groups move to the downfield (left; increase in ppm). Unsaturated groups shift to downfield (left) when affecting nucleus is in the plane of the unsaturation, but reverse shift takes place in the regions above and below this plane. **1H chemical shift plays a role in identifying many functional groups**. Application of this methodology will be done to analyze our experimental results.

Figure 6 shows the power spectrum of the DFT of the free induction decay of protons in the applied magnetic field. **2 different peaks are observable in the spectrum. It means that two different types of protons with a distinct chemical environment the investigated molecule should have.** In fact, the highest points of the peaks were labeled thanks to an investigation of the graph with Matlab tools. One can be found at 3.33 ppm and the other one can be found at 4.96 ppm. A difference in the amplitude of the peak is visible: the peak at 3.33 ppm seems to have between 2 and 3 times a highest amplitude compare to the smaller peak present at 4.96 ppm.

(6)

By definition, **the integral of the peak in the power spectrum scales as the square of the number of protons**. So by knowing the value of this integral, the number of protons of each type can be found. The selection of the points corresponding to each peak is done by giving some limits on the x-axis. Then, the ratio between the number of protons of different types is calculated thanks to Matlab. To do that, the investigation was done using the brush/select data command. The inbuilt function *trapz()* is used to find the integral under the curve. Thanks to this procedure, the ratio between the number of protons of different types is 1.7049. This means that if there are 10 protons corresponding to 4.96 ppm, we should found 17 protons from the 3.33 ppm type. In Listing 6, the code allowing this calculation is presented.

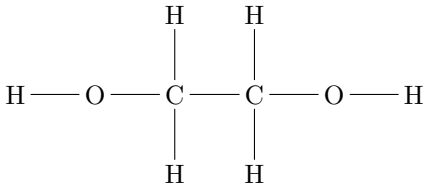
Listing 6: Subpart of code allowing the computation of the ratio between the proportion of the two different type of electron.

```
% Question 6 : What is the ratio between the number of protons of
% different types?
ind_first_peak = find(dirac_n(1,:)>3.2 & dirac_n(1,:)<3.5);
```

```
ind_second_peak = find(dirac_n(1,:)>4.8 & dirac_n(1,:)<5.2);
big_peak = sqrt(trapz(power_spectrum(ind_first_peak,1)));
small_peak = sqrt(trapz(power_spectrum(ind_second_peak)));
ratio = big_peak/small_peak;
fprintf('\tThe ratio of the integral of the two peaks is :\n');
disp(ratio)
```

(7)

From the power spectrum of the DFT of the FID data points, we know that there are **two different types of protons** with distinct chemical environments in the investigated molecule. In addition, by knowing that the integral of the peak in the power spectrum scales as the square of the number of protons, we were able to find out a **ratio of 1.7** giving relative information on the number of proton of each type in the molecule. Moreover, from the statement of the problem, we know that the columns in the data file FID.dat contain the real and imaginary part of the free induction decay of a molecule **with formula C2H6O2**. Another important information is that C, O and H atoms are able to **form covalent bonds with 4, 2 and 1 neighbors, respectively.** Thus, we can deduce that the molecule has the following structural formula :



In fact, this molecule follows the several mandatory properties found during the analysis :

- The raw formula is C2H6O2.
- C, O, and H atoms have the right number of covalent bonds (4 for each C, 2 for each O and 1 for each H).
- There are two different types of protons with distinct chemical environments in the investigated molecule: the 4 H link to the C 'in the middle' of the molecule and the 2 H link to the O at the end of both sides of the molecule.
- The ratio of the proportion of each type of proton is 2 which is near the experimental ratio we found (1.7). In fact, there are 4 protons in the more electro-positive environment and 2 proton in another more electro-negative environment due to the presence of the oxygen atom (which have a huge impact on electro-negativity). This makes sense with the power spectrum, as the smaller peak is the one with the more electro-negative environment (because of the ppm in higher).

To conclude, **the structural formula molecule shown above is inadequacy with the different experimental results. The only difference is in term of the ratio. This slight difference (2 instead of 1.7) can be explained due to some experimental mistakes during the procuration of the data.**

Problem 3

The CooleyTukey algorithm, is the most common fast Fourier transform (FFT) algorithm. It re-expresses the discrete Fourier transform (DFT) of an arbitrary composite size N = N1*N2 in terms of N1 smaller DFTs of sizes N2, recursively, to reduce the computation time to O(N log N) for highly composite N.

A radix-2 decimation-in-time FFT is the simplest and most common form of the CooleyTukey algorithm. Radix-2 divides a DFT of size N into two interleaved DFTs (hence the name "radix-2") of size N/2 with each recursive stage. This method first computes the DFTs of the even-indexed inputs ($x_{2m} = x_0, x_2, \ldots, x_{N-2}$) and of the odd-indexed inputs ($x_{2m+1} = x_1, x_3, \ldots, x_{N-1}$) and then combines those two results to produce the DFT of the whole sequence. This idea can then be performed recursively to reduce the overall runtime to O(N log N). This simplified form assumes that N is a power of two; since the number of sample points N can usually be chosen freely by the application, this is often not an important restriction.

Thus radix-2 recursive algorithm performs the discrete Fourier transform in an efficient way, hence belonging to the family of Fast Fourier transform (FFT) algorithms. The price paid for computational efficiency is a more sophisticated implementation involving recursion and workarounds for odd-sized arrays. The results of a discrete Fourier transform (DFT) and the Matlab *fft()* are exactly the same.

The goal of this problem was to write a function that implements the radix-2 variant of FFT algorithm. In our case, we will consider only the simplest case of $N = 2^r$ sampling points, where r is a natural number. **To implement the recursive part of this algorithm we coded another function called *myfft_module()*.** The function *myfft()* is calling *myfft_module()* and then *myfft_module()* is calling *myfft()* and so on. It allows the implementation of the recursive steps. **Then, a function *Wn()* was implemented to calculate the exponential terms.** The function is described in the Listing 7 just below.

Listing 7: Implementation of the Radix-2 Fast Fourier Transform.

```
function result = myfft(input_array)
  % This funciton called myfft computes discrete Fourier transform with a radix 2.
  %
  % Arguments :
  %   - input array (1D array complex, size 2N, N>0): data to transform ;
  %
  % Returns :
  %   - 1D array complex, transformed data of the same shape as an input array .

  % We check the format of the input to allow line or column vector
  input_size = size(input_array);
  if input_size(1)> 1
      input_array = transpose(input_array);
  end

  % We check if the input array is 1D. If not, an error is throw.
  nD = size(input_size);
  if (min(input_size)>1)~=0 || (nD(2)>2)~=0
      msg = 'The input array is in 2D or more.';
      error(msg);
  end

  % We check if N is a power of two
  if mod(log2(input_size(2)),2)~=0 && mod(log2(input_size(2)),2)~=1
      msg = 'The number of data is not a power of two.';
      error(msg);
  end

```

```
29    n = length(input_array);
30    N = pow2(ceil(log2(n)));
31    input_array = [input_array, zeros(1, N - n)]; % Create the new arry to store data
32    result = myfft_module(input_array); % Call the module declared below
33    result = result(1:n); % Fill up the results
34
35    % If the input was not in the optimal format, we change it during the
36    % process and thus there we give him back is original form.
37    if input_size(1)> 1
38      result = transpose(result);
39    end
40  return
41
42  function result_module = myfft_module(input_array_module)
43    % This functions allows the recursive call of myfft function
44    % It uses another function called W()
45    n = length(input_array_module);
46    % When n is 1 we have nearly finish
47    if (n == 1)
48      result_module = input_array_module;
49    else
50      % Odd and even number are calculate separtly to optimize the
51      % computation time
52      f_even = input_array_module(1:2:n);
53      f_odd = input_array_module(2:2:n);
54      % The recursive part of the algorithm is made thank to the call to the
55      % function myfft again
56      X1 = myfft(f_even);
57      X2 = myfft(f_odd).*Wn(n);
58      % Calculate and store the results
59      F1 = X1 + X2;
60      F2 = X1 - X2;
61      result_module = [F1 F2];
62    end
63  return
64
65  function w = Wn(n)
66    % This function allows the calculation of specific exponential terms
67    % added to the result
68    m = n/2;
69    w = exp(-2*pi*1i.*(0:1:m-1)/n);
70  return
```

Then, **a verification of the FFT implementation was done.** In fact, the concept of testing can be applied by transforming simple model functions and comparing the results with MatLabś function fft. A convenient way to do that was to use the assert function of Matlab. The script test.m was supplied (similar to the one in Listing 2). By running the test and see that all the tests were passed, we make sure that the implementation handled correctly different possible input arrays and that the FFT calculation is right. **In addition, an additional test 4 was added to check if our function allows well wrong input like multi-dimensional input array.** In fact, in that case, an error is raised. Test 4 illustrates the proper functioning of this module. Test 5 was also added. It checked if our module knows how to treat input the number of data points is not a power of two. In this case, an error is printed and the program is stopped. In fact, this implementation consider only the simplest case of $N = 2^r$ sampling points, where r is a natural number, as asked in the exercise. The all test script can be found in the Listing 8 just below.

Listing 8: Tests of the implementation of the Radix-2 Fast Fourier Transform.

```
clear all;
close all;
clc;

% This script tests myfft.m for correctness
tic

fprintf('Test 1: Gaussian ...');
sample = exp(-linspace(-4,4,16).^2);
assert(all(abs(myfft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 1.1: Gaussian with a different Matlab dimension ...');
sample = exp(-linspace(-4,4,16).^2)';
assert(all(abs(myfft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 1.2: Gaussian complex ...');
sample = 1i*exp(-linspace(-4,4,128).^2);
assert(all(abs(myfft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 2: sawtooth ...');
sample = linspace(-1,1,128);
assert(all(abs(myfft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 3: sin and sin2 ...');
sample = sin(linspace(-pi,pi,128));
assert(all(abs(myfft(sample) - fft(sample))<1e-10));
sample = sin(2*linspace(-pi,pi,128));
assert(all(abs(myfft(sample) - fft(sample))<1e-10));
fprintf('\tpassed\n');

fprintf('Test 4: 2D input\n');
sample = sin(linspace(-pi,pi,10));
sample = [sample, sample];
testCase = matlab.unittest.TestCase.forInteractiveUse;
verifyError(testCase,@() dft(sample),'MATLAB:UndefinedFunction');

fprintf('Test 5: Power of 2\n');
sample = sin(linspace(-pi,pi,100));
sample = [sample, sample];
testCase = matlab.unittest.TestCase.forInteractiveUse;
verifyError(testCase,@() dft(sample),'MATLAB:UndefinedFunction');

fprintf('All tests passed!\n');
timeElapsed = toc;
fprintf('Time to pass all the tests : ')
disp(timeElapsed)
```

Moreover, this algorithm was implemented to achieve one goal: perform the discrete Fourier transform in a more efficient way. Thus, thanks to Matlab function named *tic-toc()* **the execution of both *myfft()* and *mydft()* function were calculated**. The time taken for one execution of the simpler straightforward

implementation of DFT *mydft()* should be longer than the one taken our efficient implementation of the FFT. The figure 9 resume the computation time of each module in function of the number of data points. **As we expected, when the number of data points increases, the computation time of myfft() stays low when the one for mydft() increase exponentially.** In fact, the computation time of myfft should be asymptotically O(N log(N)). The code used to make this plot in described in Listing 9.

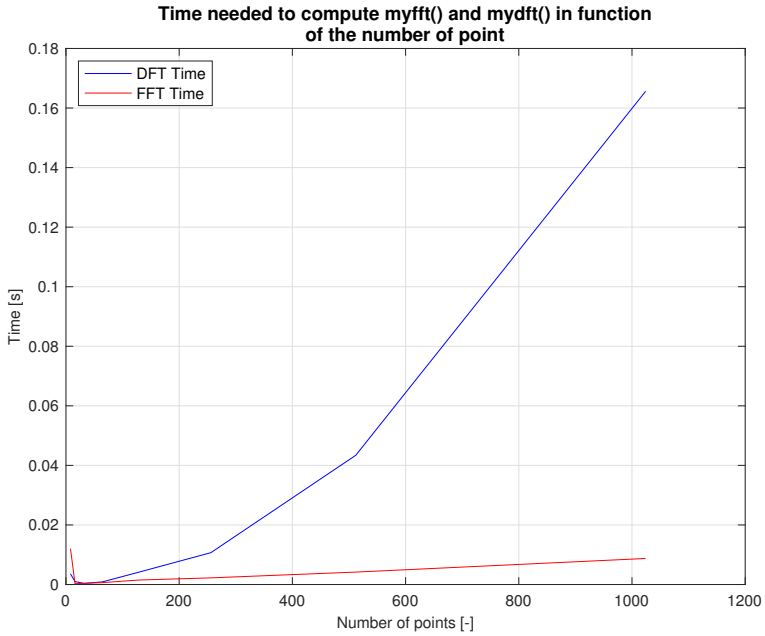


Figure 7: **Time Comparison** between our implementation of the FFT and a simpler straightforward implementation of DFT done in the first problem

Listing 9: Code allowing the comparison of the time needed to compute mydft() and a myfft()

```
clear all;
close all;
clc;

tic
number_of_point = [8,16,32,64, 128, 256, 512,1024]
dft = zeros(8,1);
fft = zeros(8,1);

for i=1:8
    sample = exp(-linspace(-4,4,number_of_point(i)).^2);
    tic
    myfft(sample);
    fft(i,1) = toc;
    tic
    mydft(sample);
    dft(i,1)  = toc;
end

figure(1)
plot(number_of_point, dft, '-b', 'LineWidth', 0.1)
hold on;
plot(number_of_point, fft, '-r', 'LineWidth', 0.1)
title1 = title({'Time needed to compute myfft() and mydft() in function',...
```

```
                'of the number of point'});
set(title1,'FontName','Arial','FontSize',12);
leg1 = legend('DFT Time','FFT Time','Location','NorthWest');
set(leg1,'FontName','Arial','FontSize',10)
xlabel('Number of points [-]','FontName','Arial','FontSize',10);
ylabel('Time [s]','FontName','Arial','FontSize',10);
grid on;
hold off
filename='./plot/time_comparison.eps';
print(gcf,'-depsc',filename)
```

```
Test 1: Gaussian ...    passed
Test 1.1: Gaussian with a different Matlab dimension ...        passed
Test 1.2: Gaussian complex ...  passed
Test 2: sawtooth ...    passed
Test 3: sin and sin2 ...        passed
Test 4: 2D input
Verification passed.
Test 5: Power of 2
Verification passed.
All tests passed!
Time to pass all the tests :     2.3570
```

Figure 8: Illustration of the testings function output for myfft() module

As a conclusion, **the Discrete Fourier transform calculation using the radix-2 method was implemented in myfft.m script and this module passed all the tests (as you can see in Figure 8). Moreover, the computation time well improved with this implementation. Thus, the Discrete Fourier Transform radix-2 module should work properly and it implies that this code is reliable.**

Problem 4

Hexagonal boron nitride (h-BN) is a two-dimensional material that is closely related to graphene. It forms the same honeycomb lattice, but instead of carbon atoms h-BN is composed of alternating boron (B) and nitrogen (N) atoms. When two honeycomb layers are placed on top of each other (form a bilayer), the orientation of the respective lattices can differ. The mutual orientation of the lattices in the two layers is described by the twist angle .

The goal of this exercise is to analyze the data using the Fourier transform tools and to find mutual orientation of the two layers in the right part of the image as well as the lattice constant of the honeycomb lattice.

(1)

The first task needed to be overcome is the loading of the image corresponding to the transmission electron microscopy picture of the bilayer of an h-BN sample. The module *imread()* of Matlab was used to load the RGB (Red-Green-Blue) image as a 3D array. In a second time, this image was converted into a 2D array. This conversion was succeeded by transforming the RGB image into a gray-scale picture thanks to the module *rgb2gray()*. The function *double()* was also used to convert the table to double precision. In addition, normalization of each pixel values has been done by subtracting the minimum values and by dividing by the maximum value. In figure 9, you can see the resulting picture after this small process.

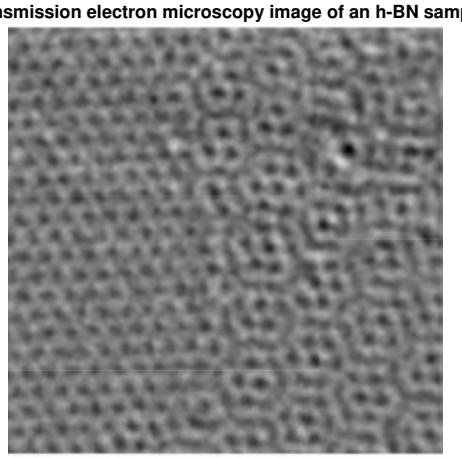


Figure 9: Image corresponding to the transmission electron microscopy picture of an h-BN sample. The left part of the image contains only one layer, while the right part is a bilayer.

This process was done thanks to the code presented in the Listing 10.

Listing 10: Code allowing the loading of a picture

```
% Question 1 : load the image as a red-green-blue (RGB) 3three-dimensional
% array with imread
img_gray = imread('BN.png');
% Question 1 : convert it to two-dimensional grayscale array with
% rgb2gray and double matlab function
img_gray = double(rgb2gray(img_gray));
img_gray = img_gray - min(min(img_gray));
img_gray = img_gray/max(max(img_gray));
% Question 1 :  plot the image with imshow
figure(1)
```

```
imshow(img_gray);
title('Transmission electron microscopy image of an h-BN sample');
filename='./plot/img_gray.eps';
print(gcf,'-depsc',filename)
```

(2)

The Fourier transform of the Transmission Electron Microscopy (TEM) image was calculate using the Matlab functions *fft2()* in a first step and *fftshift()* in a second time. *fft2()* function returns the two-dimensional Fourier transform of matrix X. *fftshift()* shift zero-frequency component to center of spectrum. The results of the *fft2()* module is shown in the figure 10 and the ones of the both command is shown in figure 11.

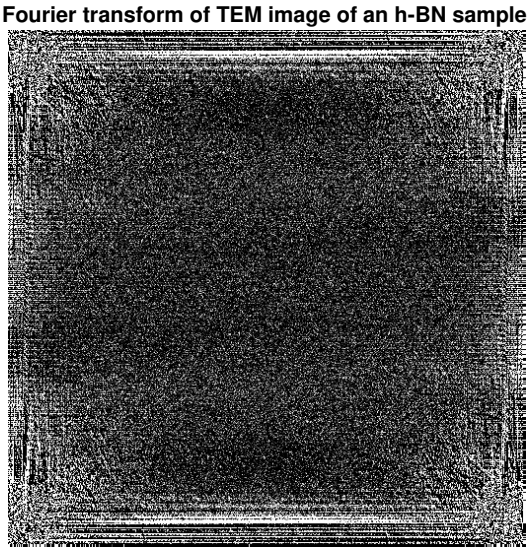


Figure 10: 2D Fourier Transform of the TEM picture

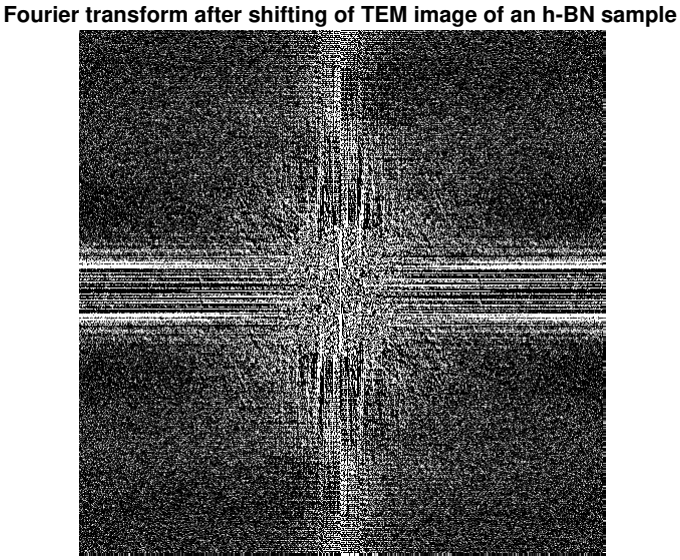


Figure 11: 2D Fourier Transform of the TEM picture after the shift of the zero-frequency component to center of spectrum.

As you can see on the figure 11, **the hexagonal pattern formed by the ensemble of Bragg peaks is not visible. In order to see clearly the Bragg peaks in the Fourier transformed image, the contrast needed to be tuned.** To do so either apply a linear operation and then a logarithmic transformation to the pixel intensities could allowed good results. The linear operation used was to divide the values of all the pixel of the shifted Fourier transform by a factor 3000 and then the absolute values function was applied. To conclude, the square of these pixels has been taken to construct the output image. The resulting picture of the process is illustrated in Figure 12.

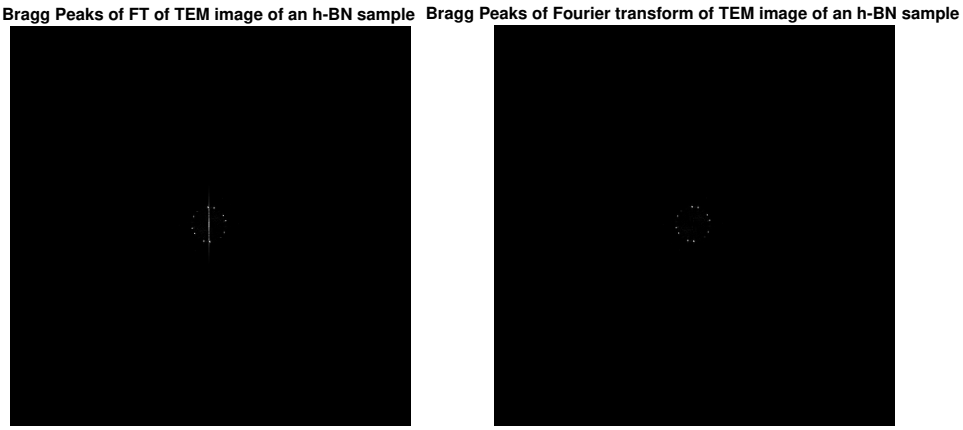


Figure 12: 2D Fourier Transform of the TEM picture.
Left : Picture after the shift and the application of the linear and logarithmic operation.
Right : Left picture after the removing of the noise.

These transformations allow the appearance of the hexagonal pattern formed by the ensemble of Bragg peaks. However, a line is present in the middle of the pattern. This curve needed to be removed as it's adding noise in our Bragg peaks analysis. To remove it, the index of the column, where all these white pixels are present, have been found and each value of this column was set to 0. In figure 12 and 13, the resulting image is presented where the hexagonal pattern formed by the ensemble of Bragg peaks can clearly be seen. **Thus, 6 peaks being composed by pair of high intensity points are visible, meaning that there are two different hexagons, each made by six points. We can switch to each part of the hexagon by moving by modulo $\frac{\pi}{3}$**

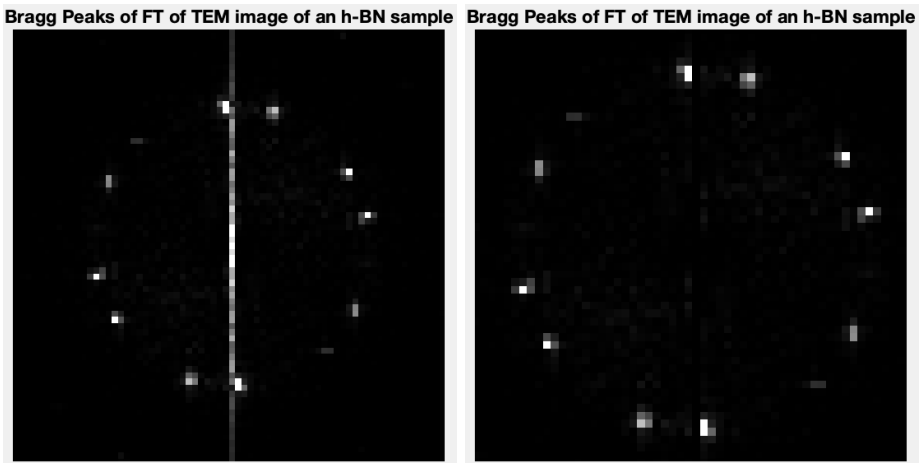


Figure 13: Zoom of the 2D Fourier Transform of the TEM picture.
Left : Picture after the shift and the application of the linear and logarithmic operation.
Right : Left picture after the removing of the noise.

The previous pictures were created thanks to the following part of code presented in Listing 11.

Listing 11: Code allowing the visualization of the hexagonal pattern formed by the ensemble of Bragg peaks.

```
% Question 2 : Calculate the Fourier transform of the TEM image
% using the fft2 and fftshift functions of Matlab.
img_fft2 = fft2(img_gray);
img_fftshift = fftshift(img_fft2);
% Question 2 : Visualize the result using imshow
% Question 2 : Calculate the Fourier transform of the TEM image
% using the fft2 and fftshift functions of Matlab.
img_fft2 = fft2(img_gray);
img_fftshift = fftshift(img_fft2);
% Question 2 : Visualize the result using imshow
figure(2)
imshow(img_fft2);
title('Fourier transform of TEM image of an h-BN sample');
filename='./plot/fft2.eps';
print(gcf,'-depsc',filename)
figure(3)
imshow(img_fftshift);
title('Fourier transform after shifting of TEM image of an h-BN sample');
filename='./plot/img_fftshift.eps';
print(gcf,'-depsc',filename)
% Question 2 : Make sure the resulting image clearly shows the hexagonal
% pattern formed by the ensemble of Bragg peaks.
img_perfect = (abs(img_fftshift/3000).^2);
img_perfect(:,263) = 0; % we supress the white line that makes non sense
figure(4)
imshow(img_perfect)
title('Bragg Peaks of FT of TEM image of an h-BN sample');
filename='./plot/bragg_peak.eps';
print(gcf,'-depsc',filename)
```

(3)

In physics, Bragg's law gives the angles for coherent and incoherent scattering from a crystal lattice. When X-rays are incident on an atom, they make the electronic cloud move, as does any electromagnetic waves. These crystals, at certain specific wavelengths and incident angles, produced intense peaks of reflected radiation. Both neutron and X-ray wavelengths are comparable with inter-atomic distances (~ 150 pm) and thus are an excellent probe for this length scale. This result was explained by modeling the crystal as a set of discrete parallel planes separated by a constant parameter. **It was proposed that the incident X-ray radiation would produce a Bragg peak if their reflections off the various planes interfered constructively.** The interference is constructive when the phase shift is a multiple of 2π.

Bragg diffraction occurs when radiation, with a wavelength comparable to atomic spacings, is scattered in a specular fashion by the atoms of a crystalline system and undergoes constructive interference. For a crystalline solid, the waves are scattered from lattice planes separated by the interplanar distance. When the scattered waves interfere constructively, they remain in phase since the difference between the path lengths of the two waves is equal to an integer multiple of the wavelength. **The effect of the constructive or destructive interference intensifies because of the cumulative effect of reflection in successive crystallographic planes of the crystalline lattice.** In addition to this constructive interference, a repetition of lattice happened for some points due to the fact that each lattice has a different frequency of apparition in function of x and y

coordinates. **Thus a high occurrence in addition to constructive interference** in the Bragg plan allows the apparition of this kind of peaks.

A diffraction pattern is obtained by measuring the intensity of scattered waves as a function of scattering angle. Very strong intensities known as Bragg peaks are obtained in the diffraction pattern at the points where the scattering angles satisfy Bragg condition. In fact, sharp Bragg peaks mean long correlation length in real space.

(4)

To use our Fourier transform's image of TEM where the hexagonal pattern formed by the ensemble of Bragg peaks can clearly be seen, a Fourier filter that allows distinguishing the honeycomb lattices of the two layers was designed. The filter should retain only the first Bragg peaks and not the low-frequency part (center of the Fourier-transformed image). A color filter was designed in this purpose which goal was to filter those color-codes distortion domains (bandpass). The idea of the filter was that after applying the filter to the image, all pixels belonging to one hexagon would be segmented in one color and all the other pixels of Bragg peaks belonging to the other hexagon would be segmented in another color (Figure 14).

This color filter would be designed as an HSV image : HSV (hue, saturation, value) is alternative representations of the RGB color model, designed to more closely align with the way human vision perceives color-making attributes. In these models, colors of each hue are arranged in a radial slice, around a central axis of neutral colors which ranges from black at the bottom to white at the top. The HSV representation models the way paints of different colors mix together, with thesaturationdimension resembling various shades of brightly colored paint, and the value dimension resembling the mixture of those paints with varying amounts of black or white paint.

Figure 14: Example of a color-coding filter.

To do that, first, in a separated process, the color filter was made. **The central pixels of each peaks belonging to Bragg peak were studied in the filtered Fourier Transform image.** Thanks to the tool data cursor of Matlab, the coordinates of the points belonging to the first and second hexagon have been found and store in several tables ($pts_{x1}, pts_{y1}, pts_{x2}, pts_{y2}$). Then, for each point, means for a pair of x and y coordinates, **the angle and the radius made by this point with the center of the circle** (which was defined to be the center of the image after using *fftshift()* function) were calculated and stored in different tables. **These information were needed to be able to distinguish the hexagonal pattern formed by the ensemble of Bragg peaks thanks to if condition.** In fact, after initialization of the 3D HSV filter with the same size as our image for each dimension, an iteration on each pixel was done and thanks to the previously calculated information, **if condition were used to defined which pixels needed to be changed.** For each pixel, as before, the angle and the radius made with the center of the image were calculated. If a pixel has a radius similar to one radius of collection of first peak's radius stored in the table rayon_1 (similar means ± 2.5) and if this pixel has an angle similar to one angle of collection of first peak's angle stored in the table angle_1 (similar means $\pm - 0.017$) then this pixel is part of the first hexagon and his value need to be changed. For example, we put the value of the third component of the HSV filter to 1 for such a pixel. This corresponds to a red label on the filter. If the pixel did not belong

to the first peaks, then with the same method, the program test if he is part of the second hexagon. If so, another color label is made by changing HSV value for this pixel. In our case, the green color was used. If the pixel does not belong to one of the two hexagons, then the pixel is not changed and remains black due to our initialization.

Thus, a Fourier color filter was created to allow to distinguish the honeycomb lattices of the two layers. In this filter, pixels corresponding to each hexagon were labeled and other pixels values were set to a black color. Labeled means that the corresponding pixels in the HSV filter were set to a specific color. **This process was done thanks to information found in the filtered Fourier Transform image.** The function allowing the creation of such filter is presented in the Listing 12 just below. In addition, the HSV filter transform in RGB scale can be seen in Figure 15.

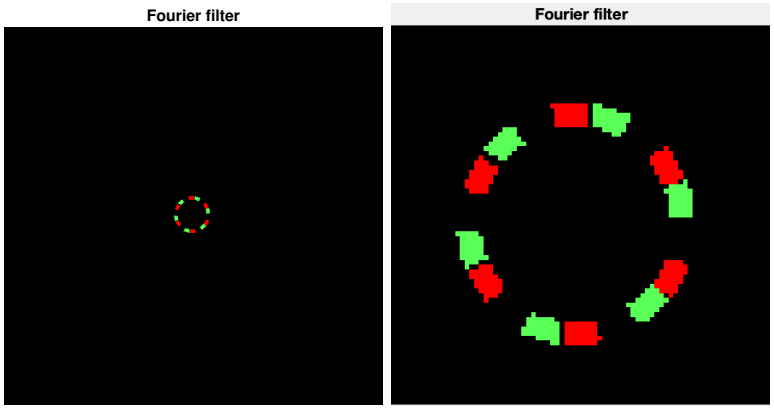


Figure 15: RGB Fourier Color Filter : The left part correspond to the entire filter when the right part is a zoom on the area of interest.

Listing 12: Code allowing the creation of the color filter with the analysis of the Bragg peaks image.

```
function rgb_filter = create_filter()
    % Function allowing the creation of the color filter.
    % This color filter will be apply on the Fourier Transform of th BN
    % image.
    % This filter will allows to distinguish the honeycomb lattices of
    % the two layers.

    N = 525; % Number of pixels per dimension on the image
    pts_x_1 = [262,282,283,264,244,243]; % X coordinate of the first hexagone
    pts_y_1 = [240,251,274,286,275,253]; % Y coordinate of the first hexagone
    pts_x_2 = [270,285,278,256,241,248]; % X coordinate of the first hexagone
    pts_y_2 = [241,258,280,285,268,246]; % Y coordinate of the first hexagone
    rayon_1 = zeros(6, 1);
    angle_1 = zeros(6, 1);
    angle_2 = zeros(6, 1);
    rayon_2 = zeros(6, 1);
    for i = 1:6
        angle_1(i) = atan2(pts_y_1(i)-N/2 , pts_x_1(i)-N/2);
        angle_2(i) = atan2(pts_y_2(i)-N/2 , pts_x_2(i)-N/2);
        rayon_1(i) = sqrt((pts_y_1(i)-N/2)^2 + (pts_x_1(i)-N/2)^2);
        rayon_2(i) = sqrt((pts_y_2(i)-N/2)^2 + (pts_x_2(i)-N/2)^2);
    end

```

```
    % Cartesian coordinates
    [x ,y] = meshgrid(1:N,1:N);
    % Polar coordinate
    phi = atan2(y-N/2 ,x-N/2);
    hsv = zeros(N,N,3);
    hsv (: ,: ,1) = 1; hsv (: ,: ,2) = 1; hsv(:,:,3)=0;
    for k=1:N % Iteration on the pixel of the X-axis
        for l=1:N % Iteration on the pixel of the Y-axis
            rayon_tmp = sqrt((N/2 - l)^2 + (N/2 - k)^2);
            for n=1:6 % Iteration on each point of the two hexagones
                % Fill up of the pixel corresponding to the first hexagone
                if (rayon_tmp < rayon_1(n) + 2.5)
                &&(rayon_tmp > rayon_1(n) - 2.5)
                &&(mod(phi(k,l)/3/pi,1) > mod(angle_1(n)/3/pi,1)-0.017)
                &&(mod(phi(k,l)/3/pi , 1 ) < mod(angle_1(n)/3/pi , 1 )+0.017)
                    hsv(k,l,3) = 1;
                % Fill up of the pixel corresponding to the second hexagone
                elseif (rayon_tmp < rayon_2(n) + 2.5)
                &&(rayon_tmp > rayon_2(n) - 2.5)
                &&(mod(phi(k,l)/3/pi,1) > mod(angle_2(n)/3/pi,1)-0.017)
                &&(mod(phi(k,l)/3/pi,1) < mod(angle_2(n)/3/pi,1)+0.017)
                    hsv(k,l,1)=119/360; hsv(k,l,2)=0.66;hsv(k,l,3)=1;
                end
            end
        end
    end
    % Convert the HSV filter to RGB and display it
    rgb_filter = hsv2rgb(hsv);
end
```

Then, the filter was applied to the original Fourier transform picture. To see a nice output image, a linear transformation has had to be done: in fact, the FT image was first divided by 100 before multiplying it with the filter. Then the absolute value function was used and the square of the resulting matrix was shown. The code allowing this process is shown in Listing 13. The resulting image is also presented in figure 16 where we can see that the segmentation worked.

Listing 13: Code allowing the creation and the application of the color filter on the Bragg peaks image.

```
% Question 4 : Design a Fourier filter that allows to distinguish
% the honeycomb lattices of the two layers.
rgb_filter = create_filter();
figure(5)
imshow(rgb_filter);
title('Fourier filter');
filename='./plot/rgb_filter.eps';
print(gcf,'-depsc',filename)
% Question 4 : Apply the color filter to our image and plot it
color_coded_FT = (abs(img_fftshift/100.*rgb_filter).^2);
figure(6)
imshow(color_coded_FT);
title('Application of the color filter on the FT');
filename='./plot/filter_bragg.eps';
print(gcf,'-depsc',filename)
```

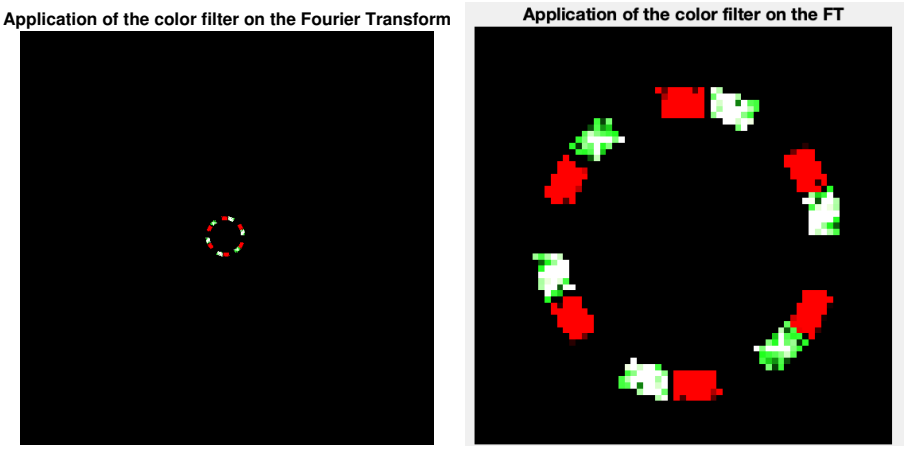


Figure 16: TEM image after application of the color filter on the filtered Fourier Transform image.
Left : Entire picture.
Right : Zoom on the Bragg peaks.

(5)

An image where the honeycomb lattices of the two layers can be distinguished is available. To achieve these output, the inverse Fourier transform (using first *ifft2()*, and then *fftshift()* of MatLab) was performed on the FT image multiply by the filter. The resulting image is shown in Figure 14. This picture makes sense as the left part of the image contains only one layer, while the right part is a bilayer. This fact can effectively be seen in Figure 14. This picture is a nice illustration of the effect of the honeycomb bilayer.

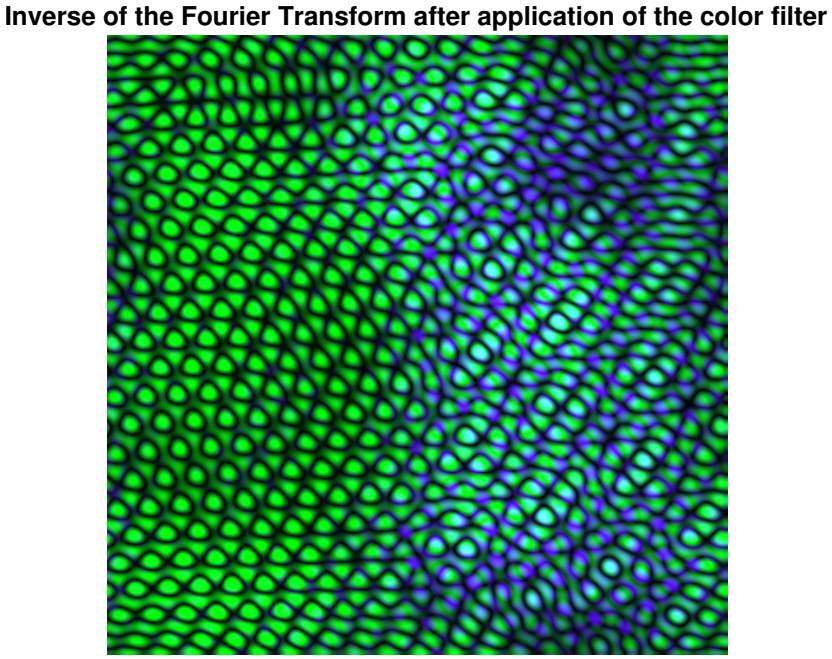


Figure 17: Inverse Fourier Transform output of Fourier Transform image after the application of the color filter.

This results were achieve thanks to the code presented in the Listing 14.

Listing 14: Code allowing the Inverse Fourier Transfom

```
% Question 4 : Design a Fourier filter that allows to distinguish
% the honeycomb lattices of the two layers.
rgb_filter = create_filter();
% Question 5 : Perform inverse Fourier transform (using first fftshift,
% and then ifft2 of MatLab)
color_coded_FT_shift = fftshift(img_fftshift.*rgb_filter);
color_coded_IFT = 10*abs(ifft2(color_coded_FT_shift));
print(gcf,'-depsc',filename)
figure(7)
imshow(color_coded_IFT);
title('Inverse of the Fourier Transform after application of the color
filter');
filename='./plot/output_grayscale.eps';
print(gcf,'-depsc',filename)
```

(6)

The goal of this question was to estimate the lattice constant of the honeycomb lattice of h-BN in Angstroms. By definition, the lattice constant is defined by the following formula :
$B = \frac{2*\pi}{distance\_between\_two\_peaks} = \frac{4*\pi}{\sqrt{3}*A}$ where A is the Lattice constant.
Starting from the Fourier image in Figure 13, the radius of the first peaks in pixels can be found. In Figure 18, you can see the illustration of the distance corresponding to the radius of the first peaks. Thanks to Matlab with the data cursor tool, the distance was found :
$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}\sqrt{(282 - 262)^2 + (251 - 240)^2} = 22.825[pixels]$.
Thus, the constant B can also be found :
$B = B[pixel] = \frac{2*\pi}{distance} = 0.275[pixels^{-1}]$
Consequently, the lattice constant A can now be calculated :
$A = \frac{4*\pi}{\sqrt{3}*B} = \frac{4*\pi}{\sqrt{3}*0.275} = 26.38[pixels]$.
The last thing to do is to convert the lattice constant from pixels unit to Angstroms by knowing that $1A = 10^{-10}m$. The picture is a picture of 525*525 pixels. Moreover, the image area is 5 nm $\times$ 5 nm. We can deduce from these information the size of one pixel : $pixel\_size = 5nm/525 = 9.5238 * 10^{-12}m/pixel = 9.52pm/pixel$.
Thus here is the estimate of the lattice constant of the honeycomb lattice of h-BN in Angstroms :
$lattice\_constant = 26.38[pixels] * 9.52[pm/pixel] = 251.13[pm] = 2.51 * 10^{-10} = 2.51[A]$

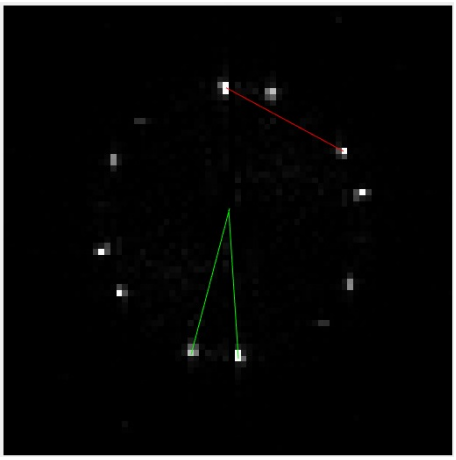


Figure 18: Fourier Transform Image of the TEM image illustrating the distance and the angle used for several calculation in question 6 and 7.
Red : Distance used to calculate the Lattice Constant.
Green : Angle used to calculate the twist angle.

(7)

An estimation of the twist angle for the honeycomb bilayer in the right part of the image can be done. **In fact, the left part of the BN image contains only one layer, while the right part is a bilayer characterized by a certain twist angle.** In Figure 19, the twist angle for the honeycomb bilayer is represented.

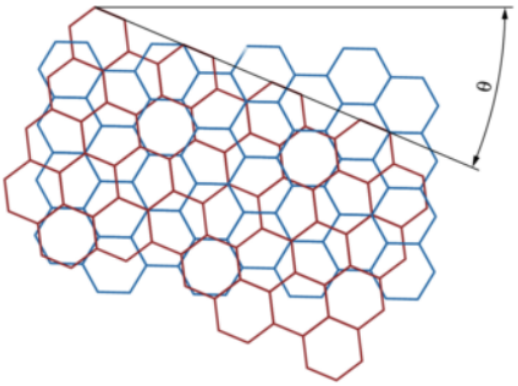


Figure 19: Schematic representation of twisted bilayer of the two honeycomb lattices.

This angle can be directly calculated from the Fourier Transform image which corresponds to Figure 18. In fact, the angle shown in green on this figure corresponds to the twist angle for the honeycomb bilayer. By using *atan2()* Matlab function, the angles corresponding to each point has been found compared to the origin of the circle (center of the image). Then, these two angles were subtracted to find the angle illustrate in green in the figure. This values which was 0.345 has been converted to degree $0.345 * \frac{180}{\pi} = 19.8$. **Thus, the estimation of the twist angle for the honeycomb bilayer in the right part of the image has a value of 20 degrees.**