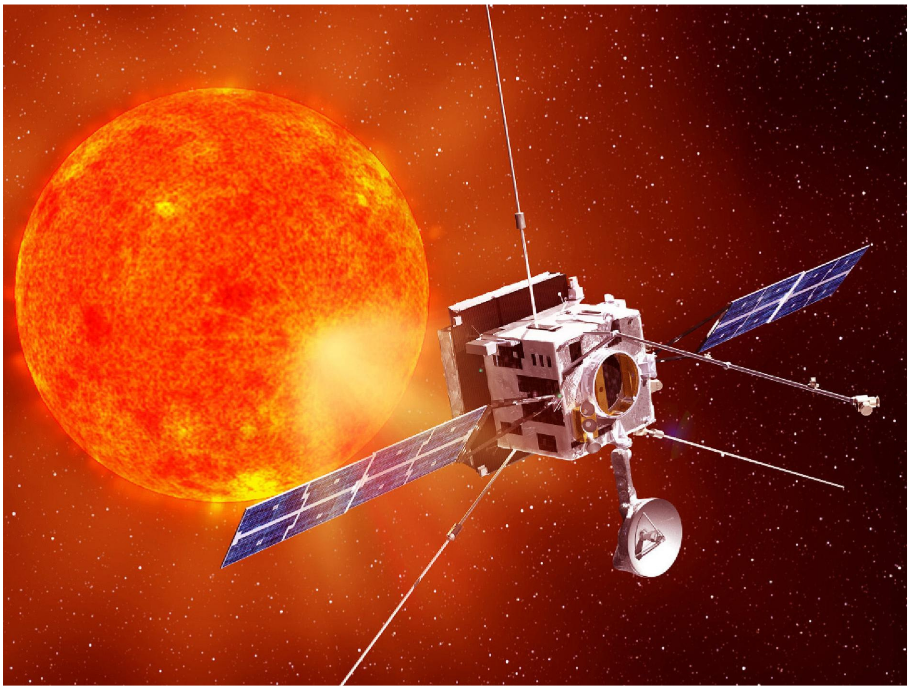
PHY571A - Projet de Recherche en Laboratoire

Embedded signal processing on space probe



January 2013 – Mars 2013

Pedro Luiz Coelho Rodrigues
X2010

Sous l'orientation de Alexis JEANDET et Paul LEROY

1 Table of Contents

2 Introduction

The space plasma physics of the LPP (Laboratoire de Physique des Plasmas), at the École Polytechnique, is responsible for the development and for the conception of an embedded data analyzer for the Solar Orbiter space probe, which launch is in 2017.

The LFR (Low Frequency Receiver) analyzer has the task of digitizing and processing the Electric and Magnetic Fields signals at frequencies below 10kHz (with 8 acquisition channels). The onboard processing of the analog waveforms allows the probe to send to earth only a few parameters, limiting dramatically the telemetry allocation of the instrument.

The research project that has been proposed by the LPP is the study of the functioning of the LFR analyzer and the development of an emulator for the system. This emulator, which has been developed in Python, will serve to validate the processing chain implemented in the analyzer and also to build tests cases to develop and validate the prototype of the instrument.

Furthermore, this emulator will allow the study of the influence of the numerical approximations done during the signal processing in the LFR analyzer.

This project was planned to be done during the two periods of the "Projet de Recherche en Laboratoire" course. The following report describes the works that were done in this period.

3 Methodology

In order to achieve the proposed goals for this project, it was necessary to do some initial bibliographic research. This activity started with the study of basic notions of discrete signal processing, in order to understand the basic ideas behind the structure of the LFR analyzer. An extensive comprehension of the LFR specifications was also needed. For that, time was given for the reading of the specification document [1], as well as for questioning.

Then, a programming language was chosen for the development of the project, and it was Python. Time was given to learn the use of this new tool. The idea behind this development in Python is to embed the written scripts in an interpreter developed by the researchers of the LPP, making it part of a much bigger environment used for the development of the LFR analyzer.

The final goal was to be able to code Python functions that would do exactly the same kind of operations executed in the LFR analyzer, respecting even the limited number of bits of resolution (16 bits) for the processed signals. This would allow the researchers to effectively compare the results from the prototypes with the ones in the Python program.

3.1 Discrete signal processing

3.1.1 The Fourier transform

A signal is a physical quantity, which varies with respect to time and space, and contains information from source to destination. Some examples of signals are audio, radar and videos. In the case of the LFR Analyzer, the signals that we manipulate are Electric and Magnetic fields, which are obtained by the sensors of the Solar Orbiter probe (one triaxial searchcoil magnetometer and three electric potential sensors).

A signal that varies with time may be associated with a function $f(t)$. It is often very useful to define the Fourier transform $\hat{f}(t)$ of this function, which is

$$\hat{f}(\omega) = \int_{-\infty}^{+\infty} f(t) \, . \, e^{-i\omega t} \, dt$$

The Fourier transform of a signal gives us information about its behavior in the frequency domain, which is something quite helpful.

3.1.2 Filtering

A very important property of the Fourier transform is its relation to the convolution of two signals, as we shall see below.

When a signal $x(t)$ enters a linear time-invariant system, the output obtained is $y(t)$, and it is given by

$$y(t) = \int_{-\infty}^{+\infty} h(t-\tau) \cdot x(\tau) d\tau$$

where the $h(t)$ function is called the impulsive response, and may be obtained by entering an impulse function $x(t) = \delta(t)$ in the system.

The operation above, which gives the output $y(t)$ in function of the input $x(t)$ and the impulse response of the system $h(t)$, is called convolution.

A very important theorem of the Fourier analysis states that the Fourier transform of $y(t)$ can be easily calculated by the product of the Fourier transforms of $x(t)$ and $h(t)$. So we have

$$\hat{y}(\omega) = \hat{x}(\omega) \cdot \hat{h}(\omega)$$

3.1.3 Discrete-time signals

Until now, we have only seen the behavior of signals in the continuous time domain, where we can define integrals and derivatives. However, in order to do digital signal processing, it is impossible to use continuous functions, we need to discretize them. And this is done by means of an operation called sampling.

Sampling a function $f(t)$ means to fix a value of T , which we shall call "period of sampling", and create a new function that relates to $f(t)$ by the following equation

$$f_s[n] = f(nT)$$

An example of a sampled signal $f(t)$ is shown in the graph below:

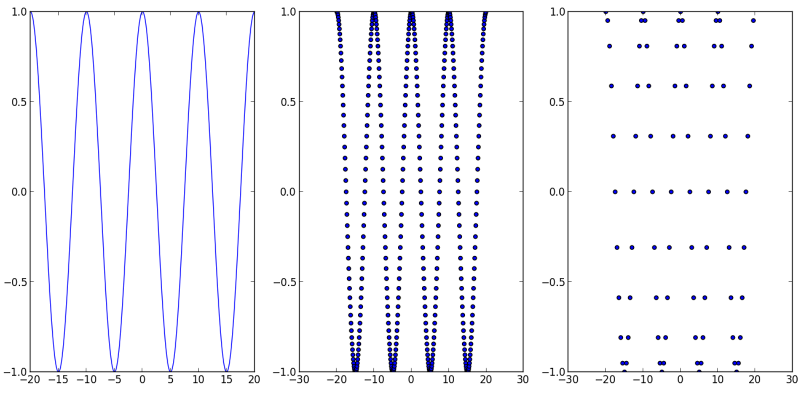


Figure 1 – The result of the sampling of a signal with two different sampling periods T

As we may see in the graph on **Figure 1**, different sampling periods give different results for the function $f_s[n]$. It should be quite intuitive to see that with bigger sampling periods, we "lose more information" about the original signal. This loss of information sometimes implicates the impossibility of reconstruction of the original continuous signal.

This kind of situation, where the sampling frequency (which is the inverse of the period of sampling) is not big enough, is called aliasing. Every circuit designer must take aliasing effects into account when he wants to sample signals.

The figure **Figure 2** shows an example of aliasing, where the real signal (in red) was sampled in such a way that the resulting sampled signal (in blue) is a signal of much smaller frequency.

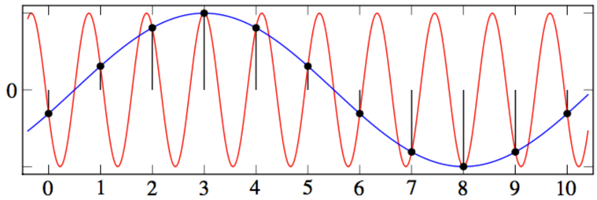


Figure 2 - Example of aliasing

One way to avoid aliasing is to use a frequency of sampling at least two times bigger than the highest frequency of the signal. This result follows from the theorem of Nyquist, which we won't demonstrate in this text.

3.1.4 Discrete Fourier transform

Once working in the discrete time domain, it becomes quite natural to define a discrete Fourier transform, for discrete values of frequency. The DFT (Discrete Fourier Transform) of a discrete signal $f[n]$ in N points is defined by the following equation:

$$\hat{f}[k] = \sum_{n=0}^{N-1} f[n] e^{\frac{-2i\pi kn}{N}}$$

Note that it is important to precise the number N of points in which the DFT is going to be calculated, since it changes the frequency resolution of the transform (the difference in frequency corresponding to two consecutive values of *k*).

The values of the DFT $\hat{f}[k]$ are equivalent to 1/T times the sampled values of the Fourier transform $\hat{f}(\omega)$, where each *k* corresponds to a frequency value of $\frac{k}{TN}$ in Hertz.

$$\hat{f}[k] = \frac{1}{T}\hat{f}\left(\frac{2k\pi}{NT}\right)$$

However, one of the biggest limitations for the calculation of the DFT in digital systems is the number of operations needed to perform it, which is proportional to N^2 (N is the number of points used to do the DFT). Because of this polynomial complexity, for many years the idea of doing Digital Signal Processing was something seen as impractical.

Everything changed when Cooley and Tukey, in an article from 1965, proposed a new algorithm for calculating the DFT. The touch of genius was to use the "redundant" characteristics of the complex exponentials in such a way that the complexity of the calculation could be reduced to something proportional to N.log(N).

The main idea of the algorithm was to use a “divide and conquer” method to recursively calculate the values of the Fourier transform, so that in the end the time of processing would be much smaller than before.

3.2 The LFR analyzer

3.2.1 Presentation of the system

Solar Orbiter is a mission from the ESA (European Spatial Agency) that will allow scientists to know a lot more about the internal Heliosphere, by doing in-situ measurements that have never been done before.

Different antennas will detect different aspects of the Magnetic and Electric fields from the Sun, generating a total of 11 different input analog signals for the LFR analyzer.

These signals can’t be directly sent to earth, since Solar Orbiter has a very limited bandwidth for sending data. This means that a lot of signal processing has to be previously done. This signal processing, which is done by the LFR analyzer, extracts the basic fundamental parameters of the input signals and allows the system to send only essential information back to earth.

The figure below shows the LFR block diagram, where we see the 11 analog input signals: BIAS_1, BIAS_2, BIAS_3, BIAS_4, BIAS_5, VHF_1, VHF_2, VHF_3, SCM_1, SCM_2, SCM_3. The signals with the prefix SCM_ are coming from the searchcoil magnetometer. All other signals are raw electric potentials (prefix VHF_) and mixed electric potential/electric field signals (prefix BIAS_). Only 8 signals are actually used for processing, as we can see by the red switches in the diagram.

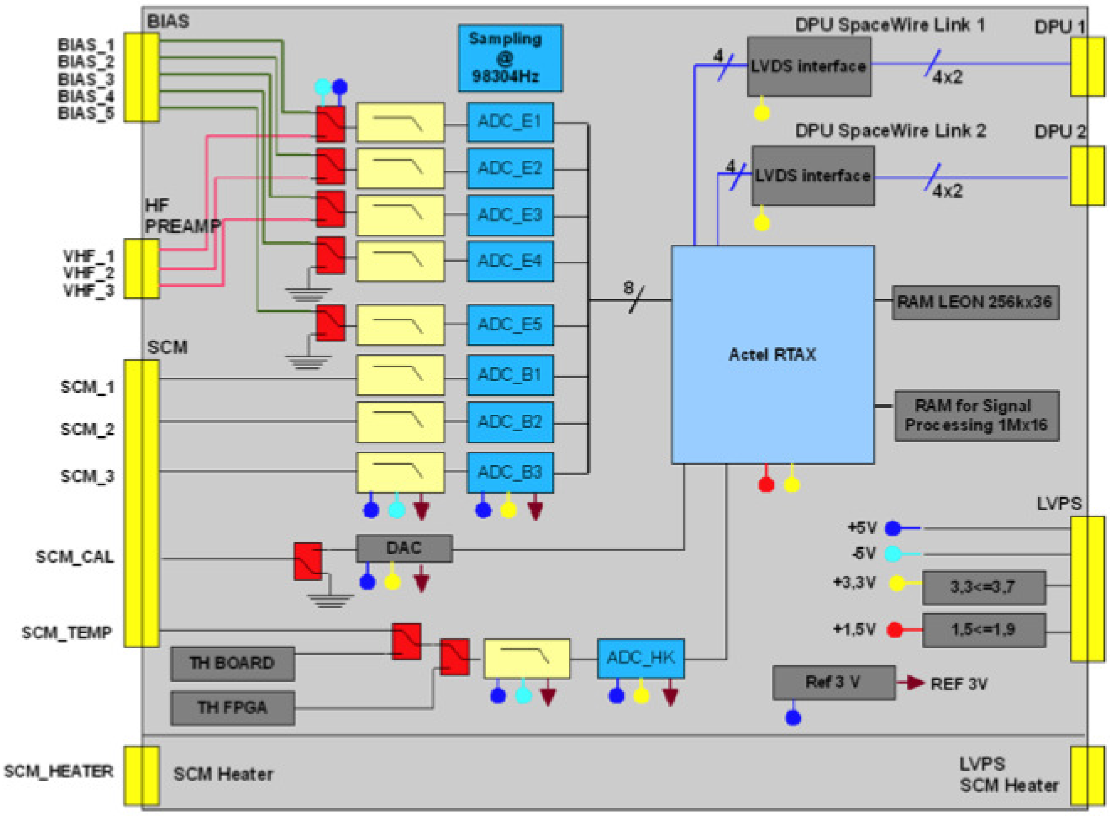


Figure 3 - LFR Block diagram

Knowing that we're only interested on signals of frequency between 0 and 10kHz, our input signals pass a first stage of anti-aliasing filtering, which guarantees a gain of -96dB for frequencies over 98304Hz, as well as a flat frequency response between 0 and 10kHz.

3.2.2 Digital data streams

After that, the filtered signal is sampled at a frequency of 98304Hz, and then converted to integer values through an Analog-to-Digital converter of 16 bits. Once in a digital form, a digital anti-aliasing filter, whose frequency response is shown in the figure below, filters the digital signal.

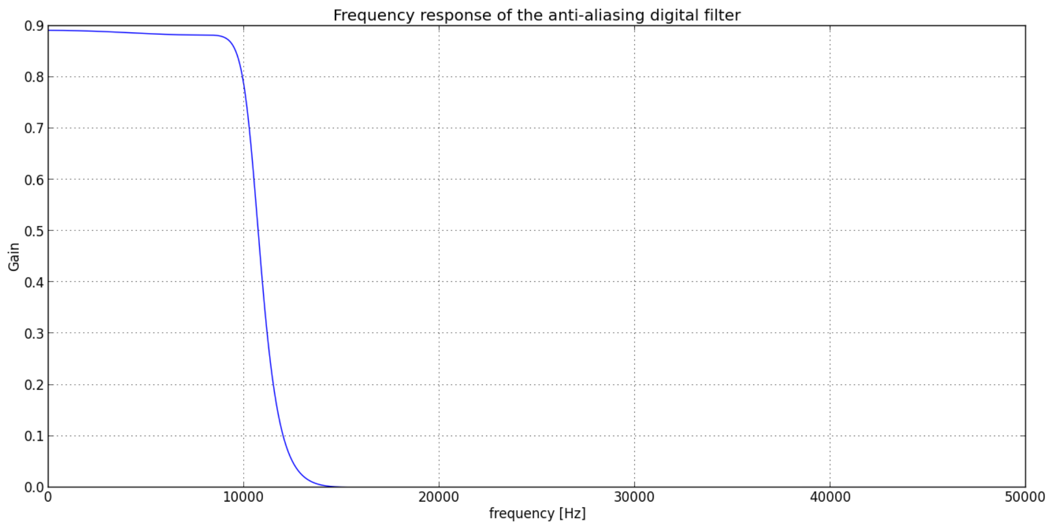


Figure 4 - Frequency response of the digital filter

This filtered digital signal is then down-sampled to a frequency of 24576Hz, which is one fourth of the original sampling frequency (which means that the procedure is simply that of taking one out of four samples, considering that the anti-aliasing digital filter is working well).

The signals that are really treated by the FGPA are these signals of frequency $f_0 = 24576$ Hz, as shown in the figure below.

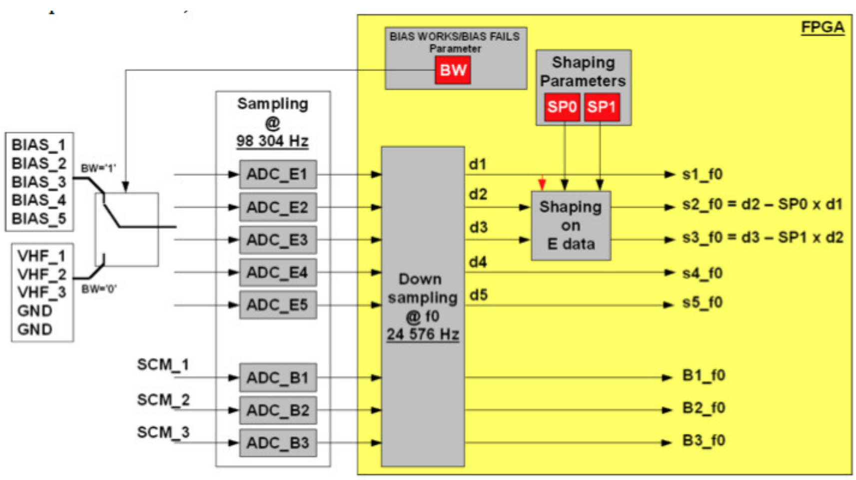


Figure 5 - Signals entering the FPGA

To obtain an optimized frequency analysis over the entire frequency band (from quasi DC to 10kHz), these signals initially at a frequency of f_0 are downsampled by digital filtering. Three sub-sampling frequencies are used: f1 = 4096 Hz, f2 = 256 Hz and f3 = 16Hz.

3.2.3 Data processing

Finally, the streams of data, each at one of the possible frequencies f0, f1, f2 or f3, are processed by the FPGA. It's important to notice that we only deal with 6 components: two signals of Electric Field, three signals of Magnetic Field and one signal with the Electric Potential. The FFT of these signals is calculated and from that we obtain the Spectral Matrices (SM). Notice that no processing is done over the signals with sampling frequency of 16Hz.

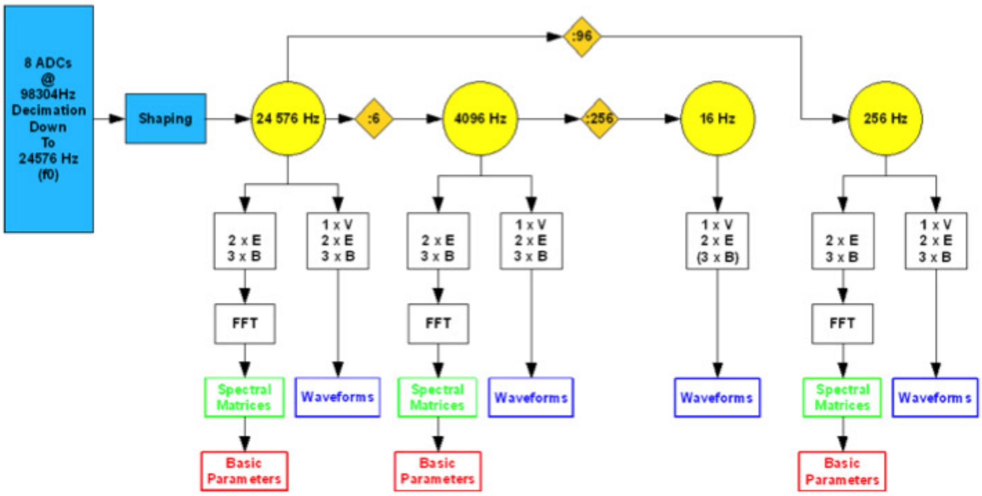


Figure 6 - Block diagram of the data processing

The Spectral Matrices are defined as follow

$$\mathbf{SM}\left(\omega_j^{(m)}\right) = \begin{bmatrix} B_1B_1^* & B_1B_2^* & B_1B_3^* & B_1E_1^* & B_1E_2^* \\ cc & B_2B_2^* & B_2B_3^* & B_2E_1^* & B_2E_2^* \\ cc & cc & B_3B_3^* & B_3E_1^* & B_1B_1^* \\ cc & cc & cc & E_1E_1^* & E_1E_2^* \\ cc & cc & cc & cc & E_2E_2^* \end{bmatrix}$$

Where the $\mathbf{B_i}$ and $\mathbf{E_j}$ are the Fourier transforms of the signals of Electric field and Magnetic field and the $\omega_j^{(m)}$ are the pulsation samples for the FFT done with N_{FFT} samples on the signal sampled at frequency f_m .

$$\omega_j^{(m)} = j\frac{2\pi f_m}{N_{FFT}}$$

From the Spectral Matrices we can calculate the Average Spectral Matrices, given by

$$\mathbf{ASM}\left(\omega_j^{(m)}\right) = \frac{1}{N_{SM}^{(m)}}\sum_{k=1}^{N_{SM}^{(m)}} \mathbf{SM}\left(\omega_j^{(m)}\right)$$

Other basic parameters may also be calculated from the **SM**. They represent the values of the normal wave vector, the wave's ellipticity, the wave's degree of polarization, the z-component of the normalized Poynting flux, the phase velocity estimator, the auto-correlations and normalized cross correlations.

Finally, depending on the chosen operation mode, these basic parameters and averaged spectral matrices are sent back to earth, as well as the signal's waveform.

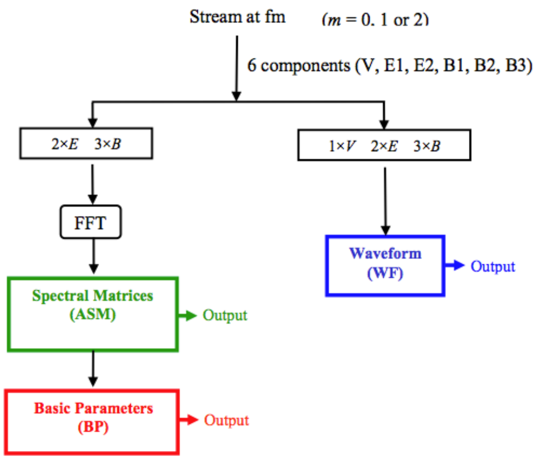


Figure 7 - Block diagram of the data processing for signals sampled at frequency f_m (m = 0, 1,2)

3.3 The basic parameters

As it was presented in the introduction section, the task of the LFR is to process signals corresponding to the Electric and Magnetic fields of electromagnetic waves, which are detected by the Solar Orbiter. There are three antennas for the detection of the Electric field and three Search Coil Magnetometers, which are responsible for the detection of Magnetic Fields.

An electromagnetic wave is a wave that propagates through space by means of an oscillating magnetic field. As Faraday's law of induction tells us, an oscillating magnetic field creates an electric field. This electric field, which is also oscillating, creates an oscillating magnetic field, following Ampère's law. This coupling between the magnetic and the electric fields creates an electromagnetic wave, which will propagate through space and have some specific properties that we shall present.

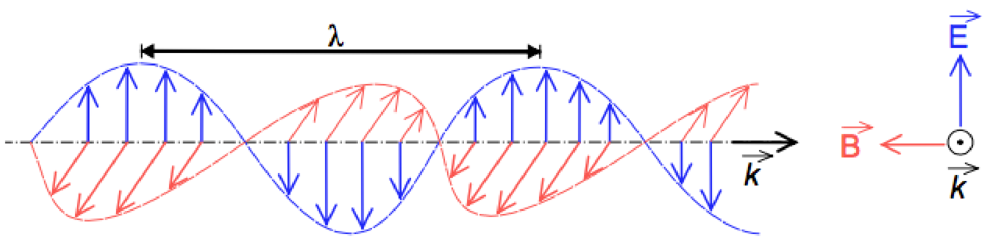


Figure 8 - An electromagnetic wave

As we can see in the figure above, the vector $\vec{k}$ tells us the direction of propagation of the electromagnetic wave. There's a fundamental relationship between $|\vec{k}|$ and λ, the wavelength, which is:

$$|\vec{k}| = \frac{2\pi}{\lambda}$$

Furthermore, as we can see in the figure, the magnetic field vector $\vec{B}$ is always orthogonal to the electric field vector $\vec{E}$ and the vector $\vec{k}$, as it may predicted from Faraday's law of induction.

Another interesting characteristic from an electromagnetic wave is its polarization. Polarization is the property of waves that can oscillate with more than one orientation. The degree of polarization (DOP) of an electromagnetic wave tells us the portion of the wave that is polarized. A DOP of 100% means that the wave is perfectly polarized whereas a DOP of 0% means that it is totally unpolarized.

To make the concept of polarization more clear, let's take a wave that propagates in the z-axis and that has its electric field in the XY plane. This electric field may be mathematically described then by the equation:

$$\vec{E}(\vec{r},t) = \mathbf{Re}\,\{(A_x\,, A_y\,.e^{i\varphi}\,,0).e^{i(kz-\omega t)}\}$$

Which is then the real part of a rotating complex vector (also called a phasor). The values of A_x and A_y will determine which kind of polarization the electromagnetic wave will have (remembering that the magnetic field is easily deduced from the electric field).

There are three main kinds of polarization: the linear polarization, which happens when the electric field oscillates only in one axis (the X or the Y), the circular polarization, when the electric field rotates in the XY plane in a circular manner and the elliptical polarization, which is a oscillation in the XY plane that is more general than the circular one. The figures below illustrate these three kinds of polarization:

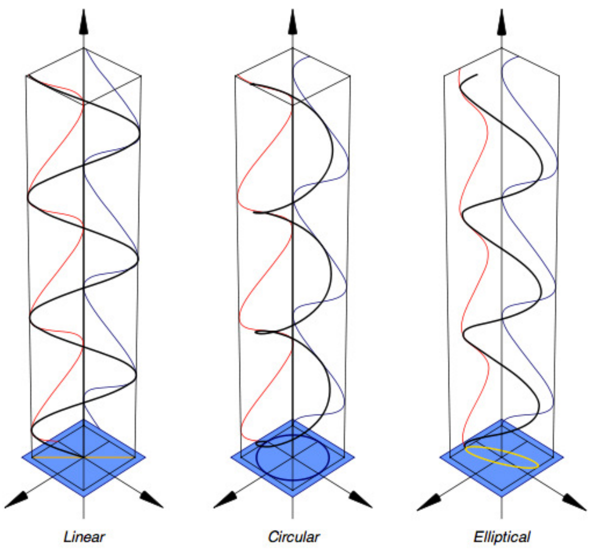


Figure 9 - Types of polarization of an electromagnetic wave

The Poynting vector is defined by

$$\vec{S} = \frac{1}{\mu}\vec{E} \times \vec{B}$$

It represents the directional energy flux density of energy transfer per unit area of an electromagnetic field. In the case of an electromagnetic wave, the Poynting vector always points in the direction of propagation, while oscillation in magnitude.

With all that said, it's time to finally present what kind of calculation and processing that is done in the LFR Analyzer.

Starting from the input signals, corresponding to the magnetic and the electric fields, the LFR calculates a Spectral Matrices, as it was explained in section 3.2.3. From these spectral matrices the LFR calculates a set of basic parameters:

- The power spectrum of the magnetic field and the power spectrum of the electric field.

These basic parameters show the way that the energy of the detected electromagnetic wave is spread in the frequency domain, for the electric and the magnetic fields.

- The normal wave vector.

This basic parameter tells the direction of propagation of the detected electromagnetic wave.

- The degree of polarization and the ellipticity of the wave.

These basic parameters refer to the degree of polarization of the electromagnetic wave (which was defined before) and the relationship of the semi-axes of the ellipse described by the electric and magnetic fields in the XY plane.

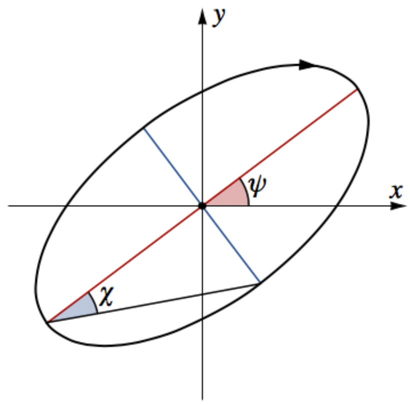


Figure 10 - Ellipse in the XY plane

- The z-component of the normalized Poynting flux.

As it was described before, the Poynting vector gives information about the way that the energy of an electromagnetic wave is being propagated. This basic-parameter gives us information about the z-component of the Poynting vector (which is the only component of interest).

- The <u>phase velocity estimator</u>.

This basic parameter gives us an estimate of the phase velocity of the detected electromagnetic wave, which is a relationship between the angular frequency and the vector the wave's k vector.

3.4 Quantization effects

Besides of having to sample signals in order to process them in the LFR Analyzer, which means working with discrete-time signals, it is necessary to quantize their values. This quantization transforms values that would belong to the set of real numbers and match them with an integer value.

This quantization doesn't come without lost, since the precision of an integer-based representation of numbers is limited to a certain number, which is not capable of describing the infinite possible values from the set of real numbers.

That said, quantization works by precising a maximum (V_{max}) and a minimum (V_{min}) value for the input number (who's in the set of real numbers) and then deciding into how many sub-intervals we wish to divide the interval [V_{min} , V_{max}].

As an example, if we take V_{max} = +1 and V_{min} = 0 and decide to have 2^3 sub-intervals, then each of our sub-intervals will have a size of 2^{-3}, which means we will be able to differentiate input values as 0.375 and 0.5 . However, two input values that are very close to each other (by "very" close we mean a distance of less than 2^{-3}) will not be differentiated, and they will be mapped to the same integer value, such as the values 0.375 and 0.380.

The figure below shows an example of signal that was sampled in time and then quantized. We can see clearly the effects of doing such an operation and the unavoidable lost of information associated with it.

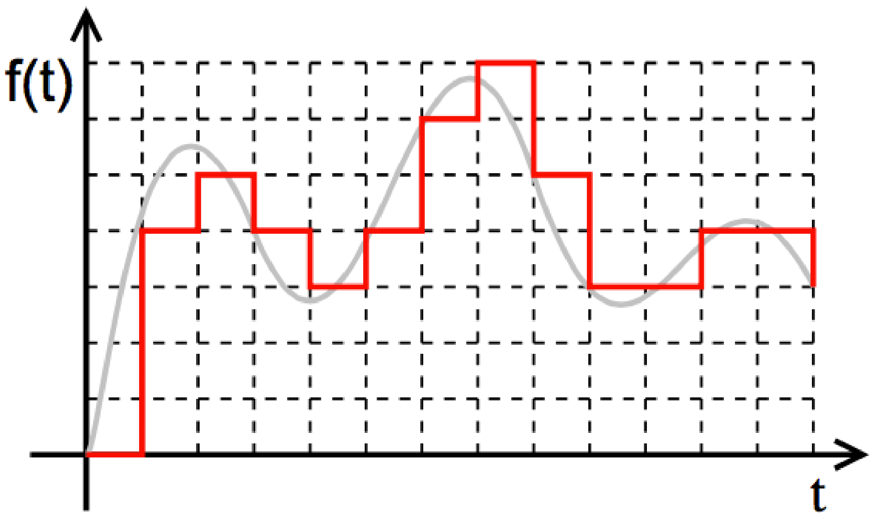


Figure 11 - Sampling and the quantization of a signal

Although we're saying that this mapping of values is done from the real set to the integer set of numbers, a digital system actually works with binary numbers. This means that the manipulations and calculations happening in a computer or a FPGA (such as the one used in the LFR analyzer) are actually being done with binary numbers. So, for example, if we say that a quantization has a precision of 3 bits, this means that the mapping will be done from the input value to the set of numbers given by B = {000,001,010,011,100,101,110,111}. Taking V_{max} = +1 and V_{min} = 0, we'll have:

000 = 0.000
001 = 0.125
010 = 0.250
011 = 0.375
100 = 0.500
101 = 0.625
110 = 0.750
111 = 0.875

As we can see, the V_{max} is actually never mapped into a binary number.

This example takes only values that are positive (for the input and for the mapped binary numbers). However, it is very often necessary to be able to represent negative values as well, in order to do subtractions and other kinds of manipulations with negative numbers. That said, a way of representing negative values must be chosen. The most common is the "two's complement" one, which gives us a representation like the one below:

0111 = 7	1111 = -1
0110 = 6	1110 = -2
0101 = 5	1101 = -3
0100 = 4	1100 = -4
0011 = 3	1011 = -5
0010 = 2	1010 = -6
0001 = 1	1001 = -7
0000 = 0	1000 = -8

As we can see, a negative number is obtained by inverting all the bits from its positive counterpart and adding one. This gives us a range going from -8 up to +7. In the case of the LFR Analyzer, the numbers are stored with a two's complement system using 16 bits of precision, which means that the integers can go from -32768 up to +32767.

It is quite clear then that the higher the number of sub-intervals, the more precise we can be in our quantization. For instance, if we take 2^{16} sub-intervals, as it is done in the LFR analyzer, we will have a very good precision, but it will still not be perfect, and some kind of errors will appear.

The analysis of these errors and the way that they influence the final results obtained by the LFR, that is, the Basic Parameters and the Spectral Matrices, is the main subject of this project. Attention was paid to the way in which each operation was done in the processing chain in the LFR, and operations were always made over integer with a varying number of bits of precision (that were taken from the LFR's specification).

3.5 Python

3.5.1 Presentation of the language

Python is an interpreted programming language, whose design philosophy emphasizes code readability. Supporting object-oriented programming, as well as other programming paradigms, Python has become a very popular programming language for scientific computing.

Although MATLAB continues to be the most common choice for scientific computing, researchers have slowly begun to choose Python as an alternative to Mathwork's product. The main reasons for that are:

- Python is free;
- Python was developed in such a way that programs are easily read and comprehensible;
- Because of its layout, its easier to transform ideas into code;
- Portability: because Python is free, codes may run anywhere.

However, in order to do the same kind of job that MATLAB is capable of doing, the user must install three extra packages into his Python system: Numpy (BSD License), Scipy (library based on Numpy, BSD License) and Matplotlib (Python Software Foundation License).

Numpy and Scipy are two libraries that allow the user to use complex routines like FFT, integrals, derivatives, matrix multiplication, etc. Another advantage of Python in comparison to MATLAB is that it's possible to see exactly how each function was written. This allows the developer to be completely sure how each of the functions that he's using are being calculated.

Matplotlib is a very helpful package responsible for all kind of plotting in Python, using commands that resemble a lot to those used in MATLAB. For instance, all the graphs shown in this report were plotted using Matplotlib.

All that said, Python is also a multi-use language, capable of doing much more than scientific calculation, something that MATLAB is not capable of.

3.5.2 First works with Python

The first works done with Python were some simple scripts to test ways of using the Scipy and Numpy packages for digital signal processing. Tests with Matplotlib were also done.

An example of results obtained with Python is shown on **Figure 12**, where we see the effects on the FFT of a signal when multiplying it by a by a window function, a very common operation used in practice.

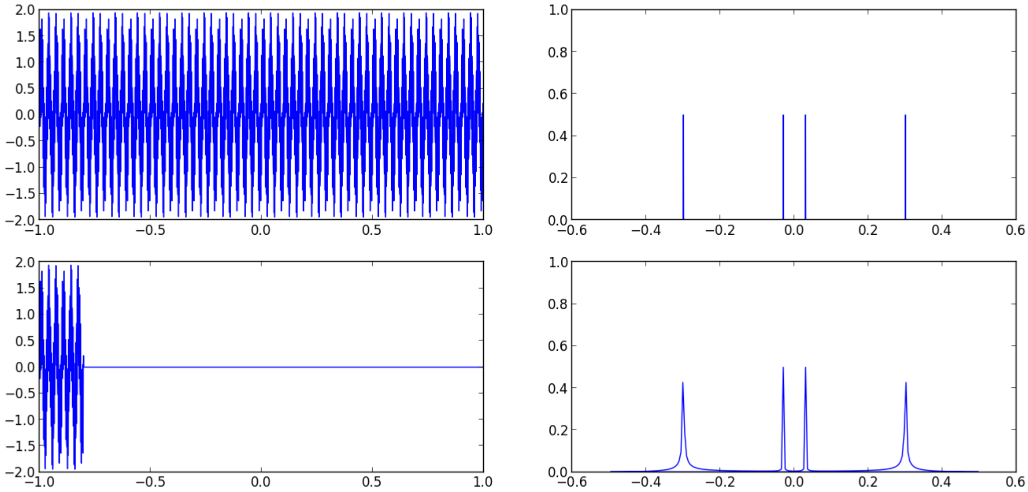


Figure 12 - Effects of windowing a signal

The figure below shows how the Python script for this example looks like.

```python
import numpy as np
import matplotlib.pyplot as plt
#Time parameters
time_step = 0.001
t_ini = -1
t_fin = +1
time_vec = np.arange(t_ini,t_fin,time_step)
#Input signal
f1 = 30
f2 = 300
input_sig1 = np.sin(2 * np.pi * f1 * time_vec)
input_sig2 = np.sin(2 * np.pi * f2 * time_vec)
input_sig = input_sig1 + input_sig2
#FFT of the input signal
fft_input_sig = np.fft.fft(input_sig)/len(input_sig)
freqs1 = np.fft.fftfreq(len(input_sig))
#Window parameters
window = np.zeros(len(input_sig))
window[0:199] = 1
#Output signal
output_sig = input_sig * window
#FFT of the output signal
fft_output_sig = np.fft.fft(output_sig[0:199])/len(output_sig[0:199])
freqs2 = np.fft.fftfreq(len(output_sig[0:199]))
#Plotting
plt.figure(1)
plt.subplot(221)
plt.plot(time_vec,input_sig)
plt.subplot(222)
plt.plot(freqs1, abs(fft_input_sig))
plt.ylim(0,1)
plt.subplot(223)
plt.plot(time_vec,output_sig)
plt.subplot(224)
plt.plot(freqs2, abs(fft_output_sig))
plt.ylim(0,1)
```

Figure 13 - Python scrip

3.5.3 Developed functions and classes

The next step was to use Python's possibility of doing Object Oriented Programs. The idea was to develop classes and functions that would to the exact kind of calculations that the filters of the LFR analyzer would do.

Although the digital filters from the LFR Analyzer are usually of order much bigger than two, their basic components are filters of order two, put in cascade. Knowing this, the classes representing these filters were done for second order filters, with the possibility of putting them in a series configuration (see **Figure 14** for a schematic view of such a second order digital filter).

Attention was paid to the fact that convolution should be coded from scratch and not simply taken from the Numpy package. The reason for this is that, by coding functions exactly as they are being executed in the LFR analyzer, we will obtain exactly the same results as seen in practice. Even the rounding errors will be the same, which is something we're interested in. The Fast Fourier Transform routine should also be coded, for the same reasons discussed above, but that shall be done in the next period of activities (from January to March of 2013).

A class called *Filter_Coefficients* was created. The idea behind that was to create something like a list, where the user may insert the six coefficients of a digital second order filter. The way of creating an instance of this class is with the constructor *Filter_Coefficients*(a_0,a_1,a_2,b_0,b_1,b_2).

$$a_0y[i] + a_1y[i-1] + a_2y[i-2] = b_0x[i] + b_1x[i-1] + b_2x[i-2]$$

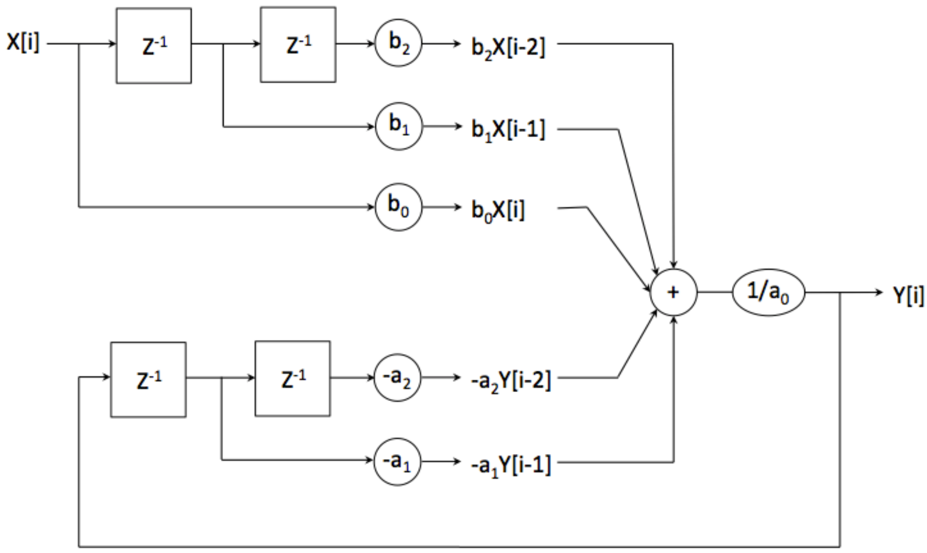


Figure 14 - Digital filter of order two

The next step was to effectively create the class for a second-order filter, where all the basic functions involving him and an input could be used. The created class is called *Filter* and it may be instantiated by calling it with a constructor using a *Filter_Coefficients* variable. The instantiation of a second order filter is presented on **Figure 15**.

```
#Coefficients of the filter
Clp = Filter_Coefficients(1,-0.7,0,0.3,0,0)
#Instantiation of the filter
Flp = Filter(Clp)
```

Figure 15 - Creating a filter

Once the filter has been created, the user may convolve an input signal by using the *convolution* function of the Filter class. Other functions that were developed for this class are the *print_coefficients*, which prints all the filter coefficients, and *impulsive_response* functions, which gives the response of a filter when we have an impulse function (δ(x)) in the input.

Another developed class is the one called *FilterGlobal*. The intention with this class was to allow the user to cascade several different second-order filters so as to get a filter equivalent to the one used in the LFR analyzer. The same kind of functions that we can use on a simple filter may be used for this kind of Global Filter. The instantiation of a higher order filter is presented on **Figure 16**.

```
#Creation of the Higher Order Filter
Filters = [Flp,Flp,Flp,Flp,Flp]
F = FilterGlobal(Filters)
```

Figure 16 - Creating a cascade filter

Finally, one thing that was taken into account when developing the functions of the *Filter* and *FilterGlobal* classes was the fact that input signals might be very long, making it necessary to do signal processing in "slices" of the input vector. This was made possible by using internal variables in the classes, in such manner that the "joining ends" of each slice would be correctly calculated.

The next step for the development on Python was the creation of functions that would calculate the basic parameters. These functions were firstly done using floating-point computation, which is the way that computers have to approximate the manipulation of real numbers. The results obtained with this kind of calculation were used as a way to validate the later calculations with integers, where the precision is much smaller (16 bits).

Functions that would create spectral matrices and calculate the corresponding basic parameters were put into action, as well as other functions allowing the plotting and saving of results.

An important created function was *SpectralMatrice_Monochromatic(a,b,n,E_para,f,fs,l)* , which allows us to test and verify the calculations of basic parameters for a monochromatic electromagnetic wave, which is the most basic case of wave that the LFR might come to analyze.

This kind of wave could be described with an equation for its magnetic field, given by:

$$\vec{B}(\vec{r},t) = \mathbf{Re}\left\{(a,b,0).\,e^{i\left(\frac{2\pi}{l}z-2\pi f t\right)}\right\}$$

Where we can easily identify some of the input parameters: 'a' and 'b', which tells us information about the ellipticity of the wave and 'l', which is simply the wavelength. The frequency 'f' is the frequency of oscillation of the wave, which cannot be simply deduced by the linear relation of $c = l.f$ because we're not necessarily in a non-dispersive media, and we can't easily deduce 'k' (which is $\frac{2\pi}{l}$) from 'ω' (which is $2\pi.f$). The other frequency taken as input, f_s , is the frequency of sampling of the original signal (the signals are actually decimated in the LFR, so that we analyze them with different sampling frequencies)

The electric field of the electromagnetic field may be written as having two components, which we'll call $\vec{E}_{para}$ and $\vec{E}_{orth}$.

$$\vec{E} = \vec{E}_{para} + \vec{E}_{orth}$$

By doing the right algebra (which we won't do in this text) one can see that $\vec{E}_{orth}$ can be obtained from the magnetic field by the following relationship:

$$\vec{E}_{orth} = -\, v_{\varphi} \,.\, \vec{\mathbf{n}} \times \vec{B}$$

The other component of the electric field, $\vec{E}_{para}$, shall be taken as an input parameter, representing the degree of polarization of the medium where the electromagnetic wave is being propagated. The $\vec{n}$ vector is one of the input parameters, corresponding to the direction of propagation of the monochromatic wave.

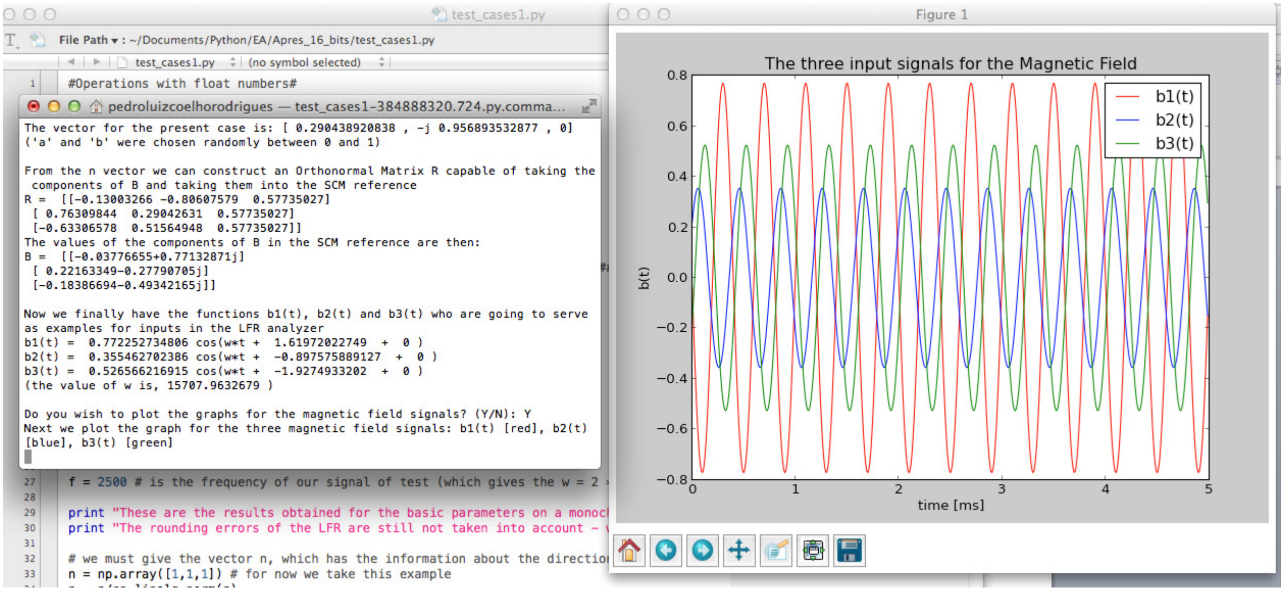


Figure 17 - Example of results obtained

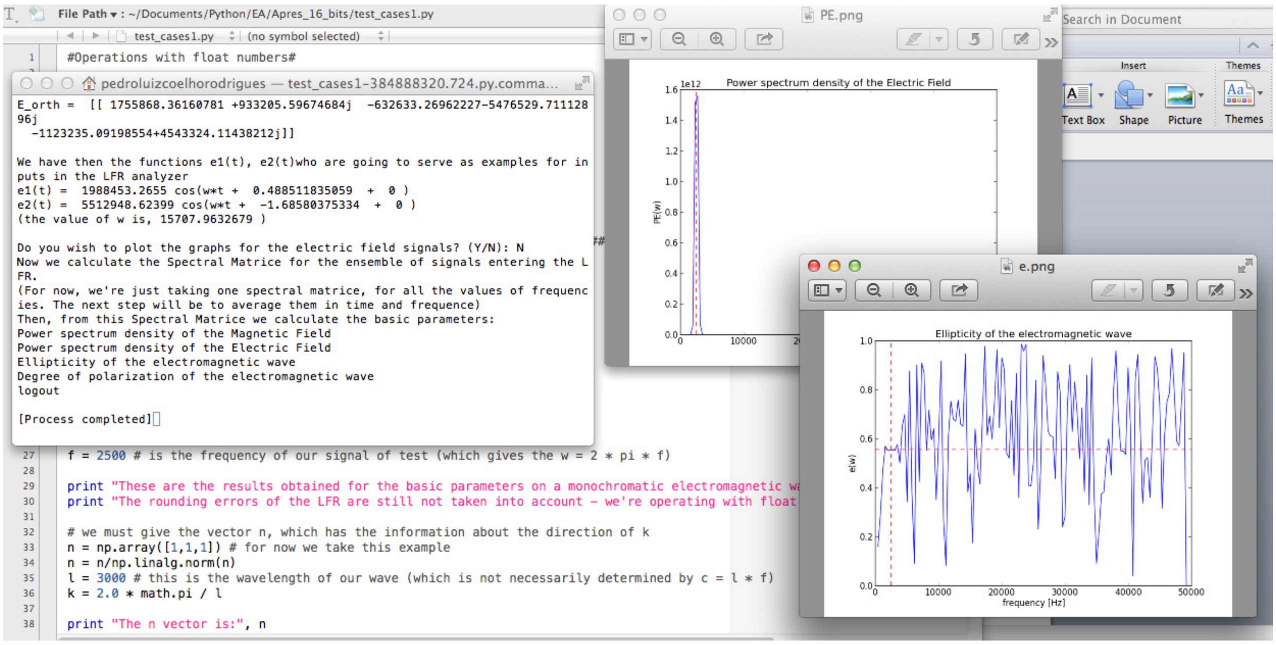


Figure 18 - Example of results obtained

The adaptation of the functions so that they would use a limited number of resolution bits (16 bits) in each calculation was done right after having finished to code the basic parameters in floating-point numbers.

The very first routines to be coded were the ones responsible for the quantization of a floating-point number into an integer, using N bits of precision (a special function for the recurrent case of N= 16 bits was also developed). Functions for the truncation of numbers were also developed.

The next step was to develop a function that would calculate the Fourier transform of an input signal. In order to do so, the Cooley-Tukey algorithm (already mentioned before) was implemented, doing manipulations over integer numbers. The results were validated by comparison with the values obtained with Numpy's function for FFT calculation.

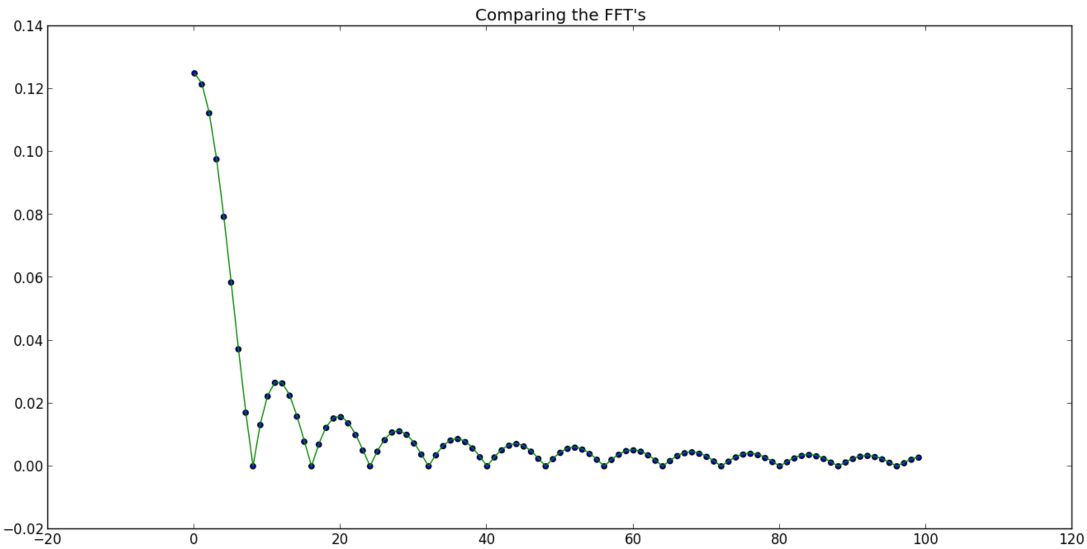


Figure 19 - Results for the floating-point (green) and integers (blue) calculations of the FFT of a rectangular pulse

As we can see from the figure above, the precision of 16 bits gives a very good dynamic for the FFT over integer numbers (the values of the integers results are scaled down to match the same scale as the floating-point results).

Once the functions for quantization and FFT were finalized, the development of functions for the calculations of basic parameters with integer numbers was put into practice. Different bit-precision was used for different basic parameters, according to the LFR Analyzer's specification. The obtained results, and a comparison with the ones obtained when using the floating-point manipulation, will be presented in the "Results" section of this report.

Finally, a routine for the actual simulation of the calculations in the LFR Analyzer began to be coded. The objective of this routine was to simulate all the steps of filtering and decimation (the reduction of the sampling frequency) that happened in the LFR Analyzer. This would obviously take into account the integer numbers manipulation that we have mentioned several times: For the basic parameters calculation as well as the digital filtering steps.

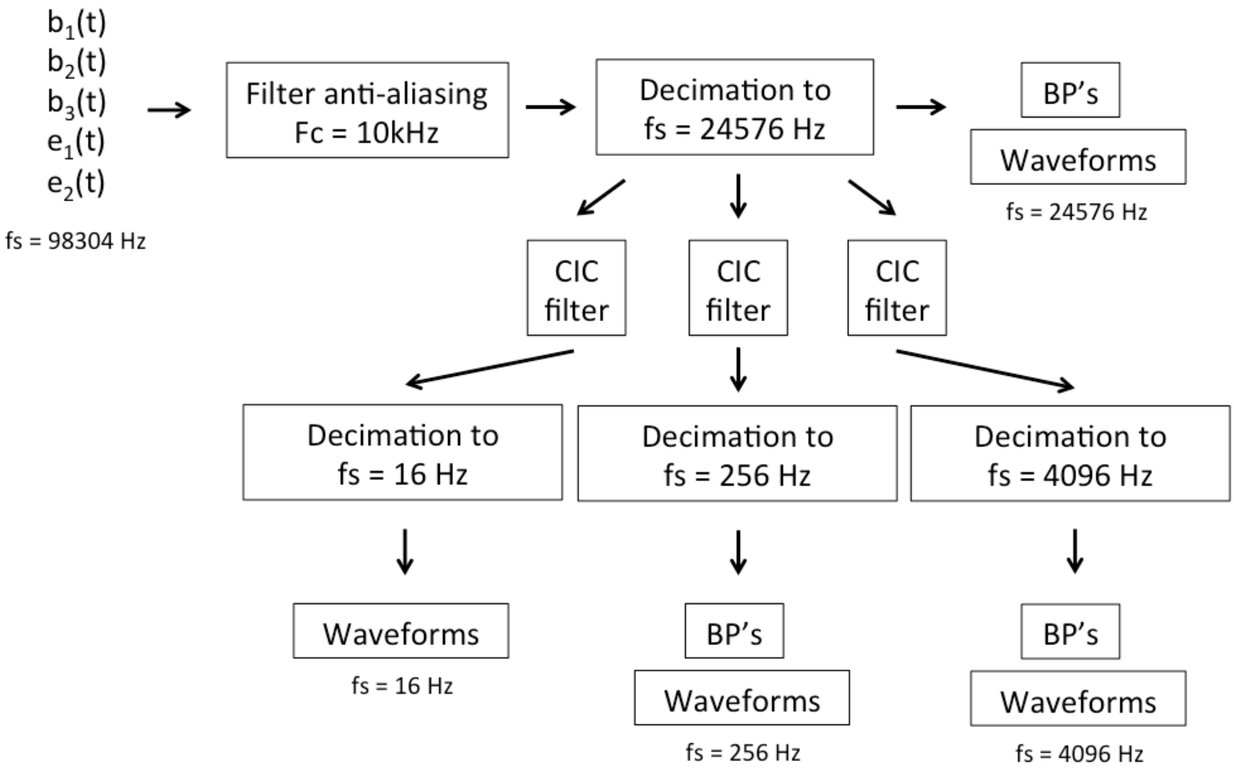


Figure 20 - Processing chain scheme

However, the blocks of CIC filters have not yet been done. These CIC filters are used to prevent the aliasing effects of the decimation from f_s = 24576Hz to f_s = 4096, 256, 16 Hz. Once these functions will be coded, the total simulation of the LFR Analyzer calculations will be possible. Nevertheless, the validation over the channel of f_s = 24576 Hz already gives us the certainty that the results over the other channels are correct.

4 Results

4.1 Results with the LFR analyzer filter

Tests were done in order to validate the classes and functions related to the filtering functions. These tests were made on filters with the same coefficients as those used in the digital anti-aliasing filter from the LFR Analyzer (the one used before down-sampling to 24576Hz). The resulting impulse response, and its Fourier transform, was compared with the one obtained in MATLAB's DSP toolbox. As we can see in the figure below, the developed functions have exactly the same results as the ones obtained in MATLAB, except for a constant scale factor.

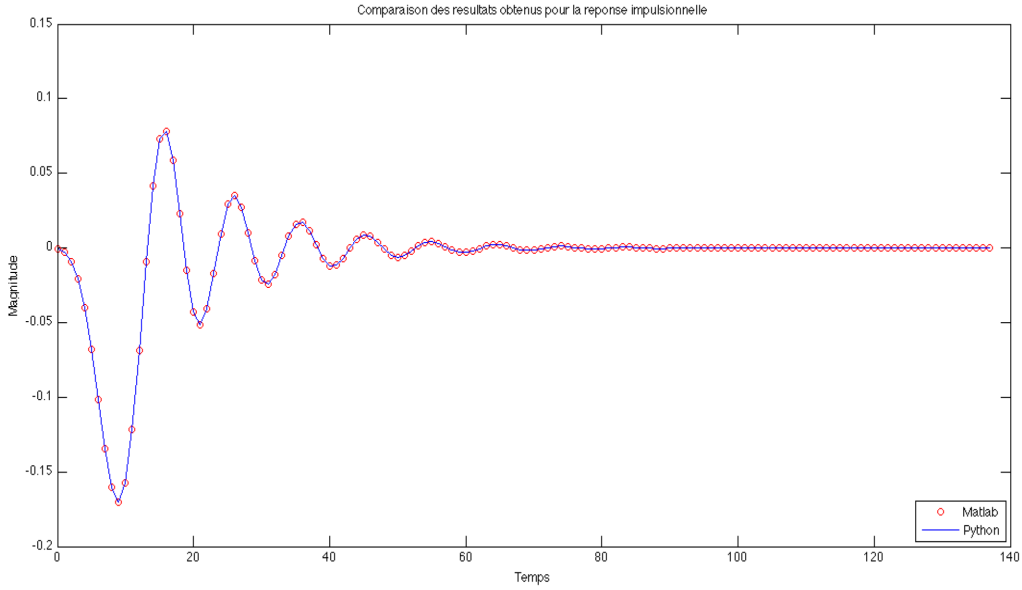


Figure 21 - Comparing the results obtained with Matlab and Python

4.2 Comparison of the floating-point and the integer calculation of BP's

Tests were done in order to validate the functions that calculate the basic parameters using integers. The precision of the input signals was always 16 bits, but for each basic parameter the number of bits was different, as it was specified in the LFR's specifications.

The most basic case of electromagnetic wave that could have been detected by the LFR analyzer was a monochromatic wave, with only one frequency and a well-defined polarization. This was exactly the kind of wave we used to validate the simulations.

For instance, taking an electromagnetic wave with the following parameters (which were already explained before),

'a' = 0.94276955701
'b' = 0.333444991527
n = [0.57735027 0.57735027 0.57735027]

l = 30000
f = 1500
E_para = [0, 0, 0]

One would obtain the following input signals for the LFR Analyzer:

b1(t) = 0.536200492766 cos(w*t + -0.400065319944)
b2(t) = 0.759731032119 cos(w*t + -3.07988596072)
b3(t) = 0.367828479583 cos(w*t + 0.768603365142)
e1(t) = 27709029.0141 cos(w*t + -2.85397345026)
e2(t) = 13460599.4161 cos(w*t + 2.02955904359)
(the value of w is 9424.77796077)

In order to make the input signals closer to the ones we would expect to see in reality, we add a white noise to each input, before starting the quantization and calculation of basic parameters.

After a quantization over 16 bits, with limits equal to

-1 and +1 for the magnetic field and
-10^7 and $+10^7$ for the electric field

The basic parameters were calculated using the quantized and the original signals (that were represented over floating-point numbers). They were then compared, as we can see in the following graphs for some of these parameters (the values for the integers were all scaled down so that they would match the values in the floating-point case).

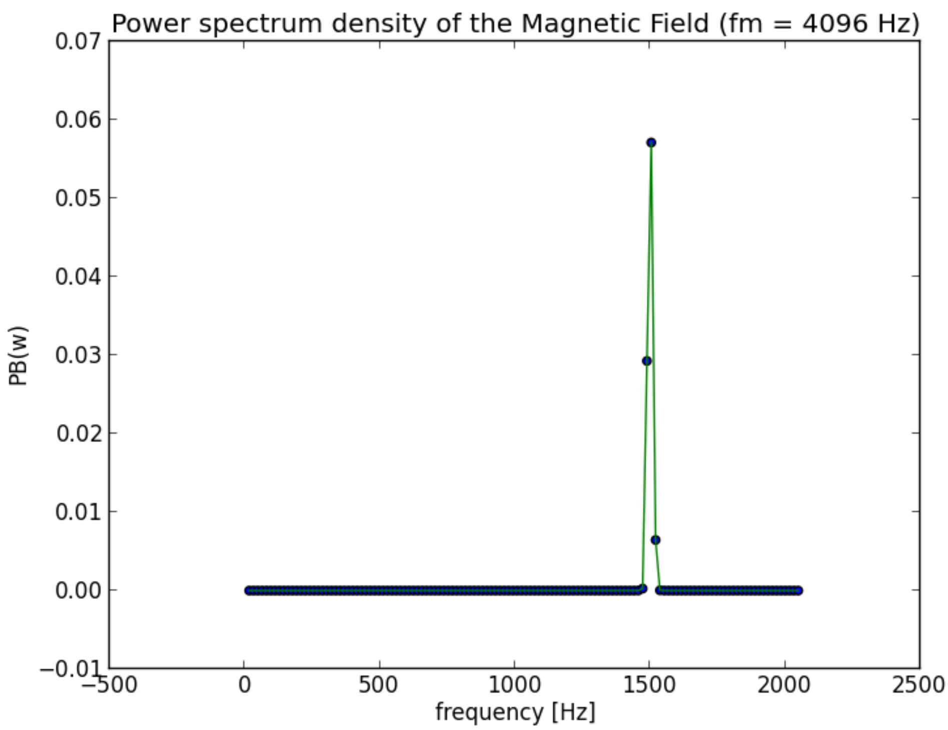


Figure 22 - Power spectrum density for the Magnetic Field (in green the floating-point results and blue the integers results)

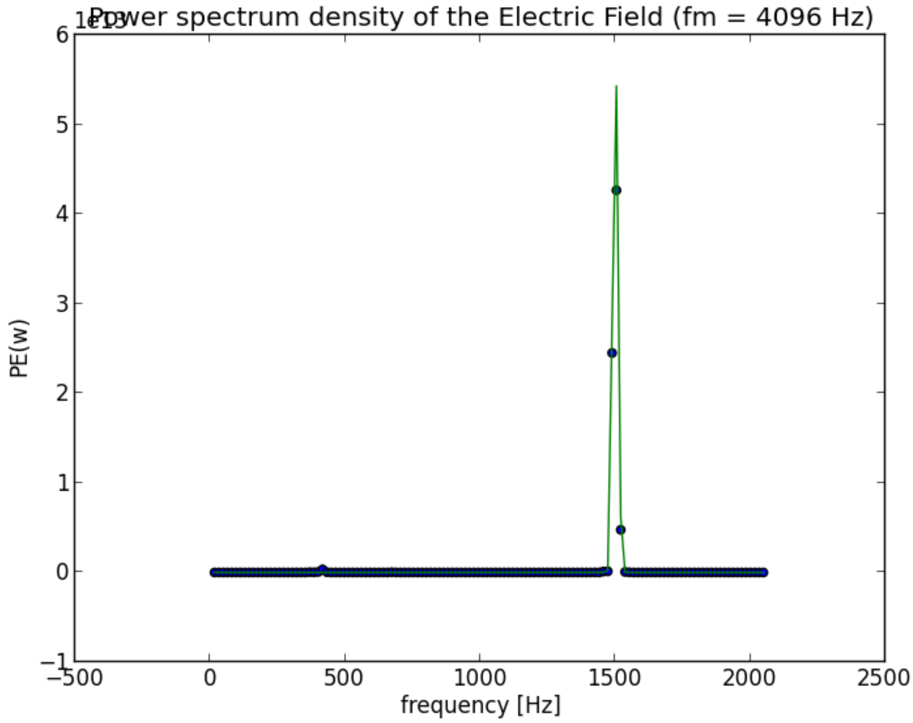


Figure 23 - Power spectrum density for the Electric Field (in green the floating-point results and blue the integers results)

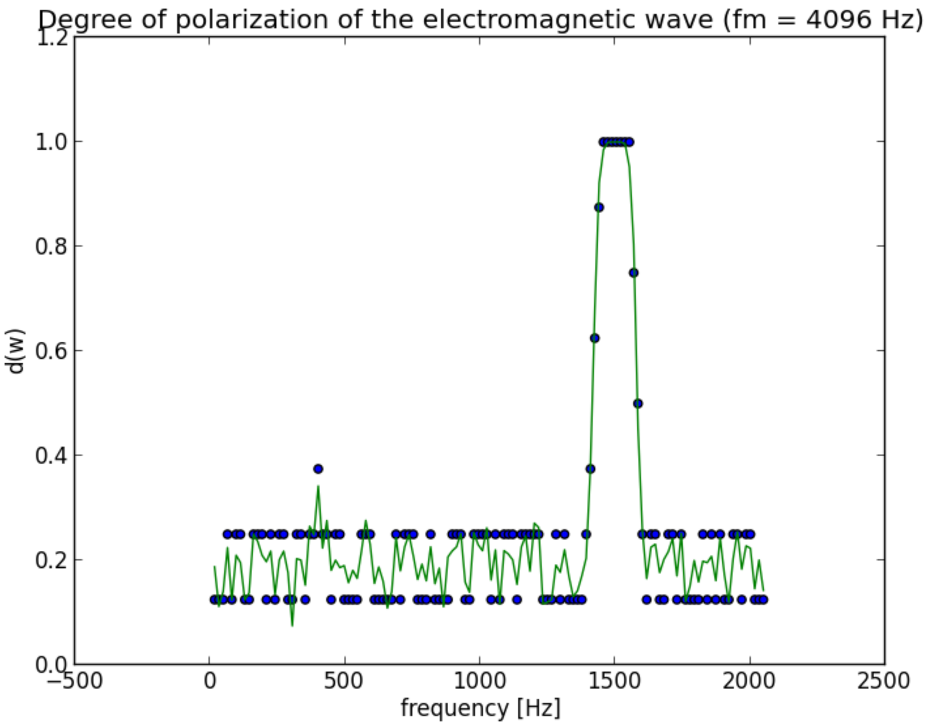


Figure 24 – Degree of polarization of the wave (in green the floating-point results and blue the integers results)

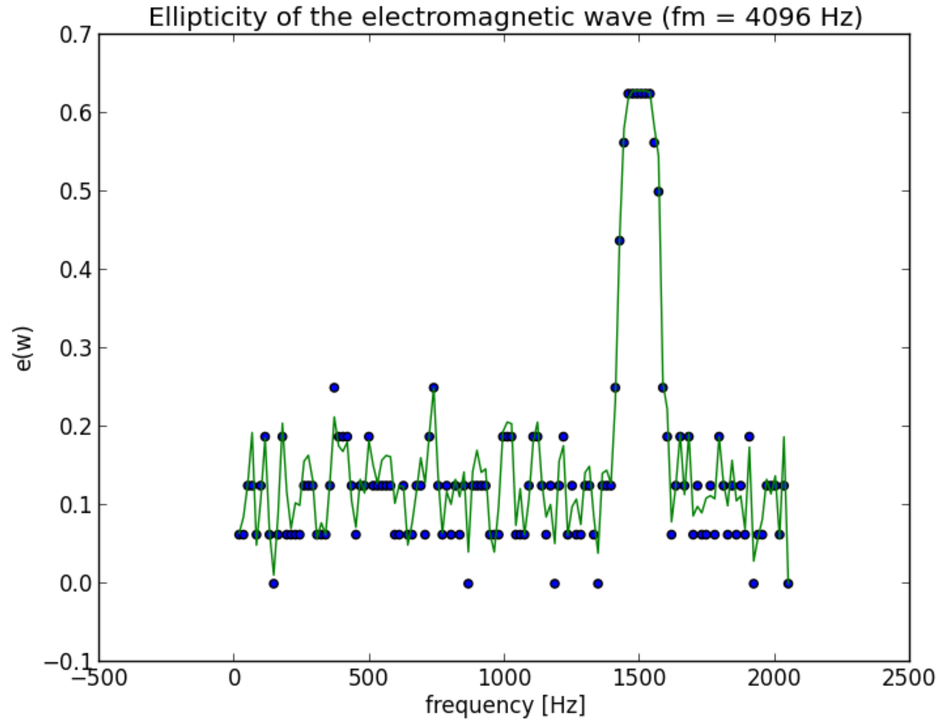


Figure 25 – Ellipticity of the wave (in green the floating-point results and blue the integers results)

As we can see in the shown graphs, there's a very good correspondence between the values obtained in the floating-point and the integers calculations, especially for the values where we would really expect to have some kind of information, which is the region of 1500 Hz.

With all that said, we can conclude that the calculations over integers are consistent with their expected values, which ensures us that further simulations will be possible and that their results shall really correspond to what we would see in the LFR Analyzer.

5 Conclusion

This project was a really good way to learn different aspects of a project in electronics, informatics and physics. It allowed me to see the interactions between physicists, who gave the specifications and desired parameters, and the engineers, who should analyze the specifications and say if (and how) they could be implemented.

The types of limitations and constraints that spatial projects must take into account were also something really interesting to learn, since it demands a special kind of engineering that I would never learn in class.

The manipulation of integers and the observation of quantization effects was also a very interesting subject to work on, since it influences a host of other applications and situations where electronics is used.

Finally, the learning of a new programming language was something really enriching and helped me to adapt and learn “how to learn” a new language.

6 References

[1] LFR Analyzer Documentation. Reference RPW-MEB-LFR-SPC-00003 (8 Mars 2012)

Discrete Signal Processing, Alan V. Oppenheim

Documentation for Python: http://www.python.org
Documentation for Scipy: http://www.scipy.org
Documentation for Numpy: http://numpy.scipy.org
Documentation for Matplotlib: http://matplotlib.org

http://upload.wikimedia.org/wikipedia/commons/3/35/Onde_electromagnetique.svg
http://en.wikipedia.org/wiki/File:Polarisation_(Linear).svg
http://en.wikipedia.org/wiki/File:Polarisation_(Circular).svg
http://en.wikipedia.org/wiki/File:Polarisation_(Elliptical).svg
http://en.wikipedia.org/wiki/File:Polarisation_ellipse2.svg
http://upload.wikimedia.org/wikipedia/commons/9/9a/Digital.signal.svg