

notebook2

April 4, 2016

<h1>Introdução histórica</h1>

<p>Historicamente falando a FFT tem sua peculiaridade,começando em 1805 quando Gauss tentou traçar a
<hr>

<h1>Transformada de Fourier</h1>

<p> A motivação para a transformada de Fourier veio com o estudo da série de Fourier. No estudo da s
 <div style="text-align:center">caso a imagem não apareça, aces
<p>No primeiro frame da animação, a função f é solucionada usando a série de Fourier : con

Teste da transformada rápida de Fourier (FFT)

Nessa parte do trabalho iremos testar como o cálculo da FFT irá se comportar com as bibliotecas **Numpy**
e **Scipy**

$$y[k] = \sum_{n=0}^{N-1} W_n x[n]$$
$$W_n = e^{-2\pi j \frac{kn}{N}}$$

Iremos calcular a FFT de um vetor com 4096 valores aleatórios usando as bibliotecas propostas e assim
iremos analisar o tempo de compilação de cada.

In [1]: *#Definindo as Bibliotecas que iremos usar*

```
import cmath as mt
from tabulate import tabulate
import time
import scipy as sc #criando um objeto sc da classe scipy
import numpy as nm #criando um objeto nm da classe numpy
import matplotlib.pyplot as plt #criando um objeto plt da classe matplotlib.pyplot
%matplotlib inline
import random
import warnings
warnings.filterwarnings('ignore')
```

In [15]: *#-----Definindo quais serão os 4096 números-----*

```
N = 4096 #tendo 4096 amostras
x=[[int(1000*random.random()) for i in range(N)]] #Gerando 4096 amostras com N variando de 0-
```

#-----Usando o Scipy para FFT-----

```
start_time = time.time()
y=sc.fft(x)
```

```

timeScipy=(time.time() - start_time)
print("Para o Scipy teremos o tempo de %s segundos ---" % timeScipy)

#-----Usando o Numpy para FFT-----

start_time = time.time()
y1=nm.fft.fft(x)
timeNumpy=(time.time() - start_time)
print("Para o Numpy teremos o tempo de %s segundos ---" % timeNumpy)

```

Para o Scipy teremos o tempo de 0.0009996891021728516 segundos ---
 Para o Numpy teremos o tempo de 0.0010004043579101562 segundos ---

```

In [18]: start_time = time.time()
N=4096
y=0
x=[]
# Definindo o valores aleatórios do vetor X
for i in range(N):
    x.append(int(1000*random.random()))
# Implementação da FFT
for i in range(N):
    z=mt.cos(2*mt.pi*y/N)-1j*mt.sin(2*mt.pi*y/N)
    x[i]=z
    y=y+1
timeMyFFt=(time.time() - start_time)
print("Para a código de FFT que desenvolvemos teremos o tempo de %s segundos ---" % timeMyFFt)

```

Para a código de FFT que desenvolvemos teremos o tempo de 0.014504432678222656 segundos ---

Teste da transformada rápida de Fourier inversa (IFFT)

Nessa parte do trabalho iremos testar como o cálculo da IFFT irá se comportar com as bibliotecas **Numpy** e **Scipy**

$$x[n] = \frac{1}{N} \times \sum_{n=0}^{N-1} W_n y[k]$$

$$W_n = e^{2\pi j \frac{km}{N}}$$

Iremos calcular a IFFT de um vetor com 4096 valores aleatórios usando as bibliotecas propostas e assim iremos analisar o tempo de compilação de cada.

```

In [4]: #-----Definindo quais serão os 4046 números-----

```

```

N = 4096 #tendo 4046 amostras
x=[[int(1000*random.random()) for i in range(N)]] #Gerando 4046 amostras com N variando de 0-1

#-----Usando o Scipy para IFFT-----

start_time = time.time()
y=sc.ifft(x)
timeScipy2=(time.time() - start_time)
print("Para o Scipy teremos o tempo de %s segundos ---" % timeScipy2)

```

#-----Usando o Numpy para IFFT-----

```
start_time = time.time()
y1=nm.fft.ifft(x)
timeNumpy2=(time.time() - start_time)
print("Para o Numpy teremos o tempo de %s segundos ---" % timeNumpy2)
```

Para o Scipy teremos o tempo de 0.0 segundos ---

Para o Numpy teremos o tempo de 0.0 segundos ---

```
In [19]: start_time = time.time()
        N=4096
        y=0
        x=[]
        # Definindo o valores aleatórios do vetor X
        for i in range(N):
            x.append(int(1000*random.random()))
        # Implementação da IFFT
        for i in range(N):
            z=1/N*mt.cos(2*mt.pi*y/N)+1j*mt.sin(2*mt.pi*y/N)
            x[i]=z
            y=y+1
        timeMyIFFt=(time.time() - start_time)
        print("Para a código de IFFT que desenvolvemos teremos o tempo de %s segundos ---" % timeMyIFFt)
```

Para a código de IFFT que desenvolvemos teremos o tempo de 0.016506195068359375 segundos ---

0.1 # Exemplo

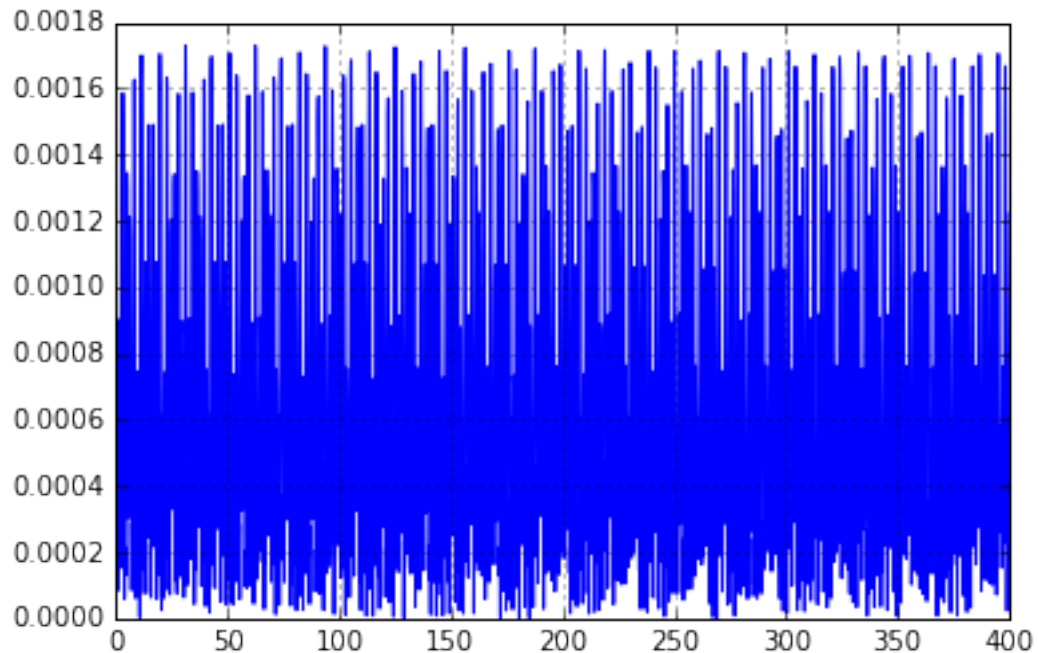
Para poder exemplificar o uso da FFT e da IFFT de uma forma ainda mais clara, iremos utilizar a soma de dois cossenos, onde:

$$2 \times \cos(30x \times 5\pi) + 1.54 \times \sin(20x \times 2\pi)$$

Teremos $N = 4096$ amostras aleatórias e um período de $T = \frac{1}{800}$, assim poderemos testar como o **Scipy** e o **Numpy** irão trabalhar com a parte matemática e gráfica do resultado. Lembrando que anteriormente testamos as duas bibliotecas para calcular a FFT e a IFFT de um vetor com 4096 valores aleatórios.

```
In [6]: N = 4096
        T = 1.0 / 800.0
        x = nm.linspace(0.0, N*T, N)
        y = 2*nm.cos(5*nm.pi*(30*x) ) + 1.54*nm.cos(2*nm.pi*(20*x) )

        xf = nm.linspace(0.0, 1.0/(2.0*T), N/2)
        plt.plot(xf, 2.0/N * nm.abs(y[0:N/2]))
        plt.grid()
        plt.show()
```

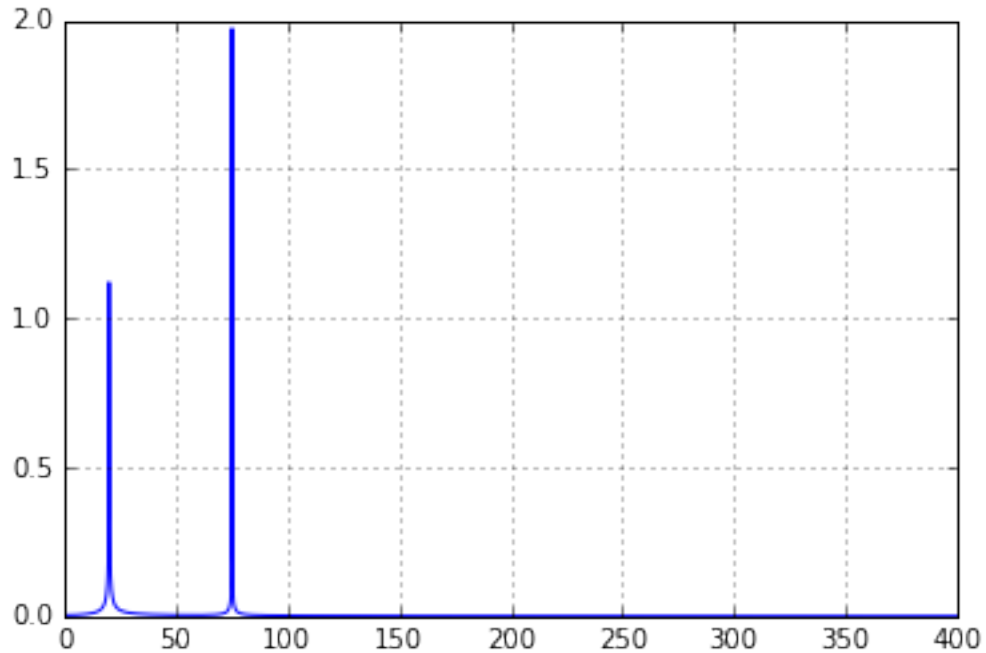


Usando FFT

Resolução do exemplo usando scipy

```
In [7]: start_time = time.time()
        N = 4096
        T = 1.0 / 800.0
        x = sc.linspace(0.0, N*T, N)
        y = 2*nm.cos(5*nm.pi*(30*x) ) + 1.54*nm.cos(2*nm.pi*(20*x) )
        yf = sc.fft(y)
        xf = sc.linspace(0.0, 1.0/(2.0*T), N/2)

        plt.plot(xf, 2.0/N * nm.abs(yf[0:N/2]))
        plt.grid()
        plt.show()
        timePlotScipyFft=time.time() - start_time
        print("Tempo de compilação para a biblioteca Numpy é de %s segundos ---" % timePlotScipyFft)
```

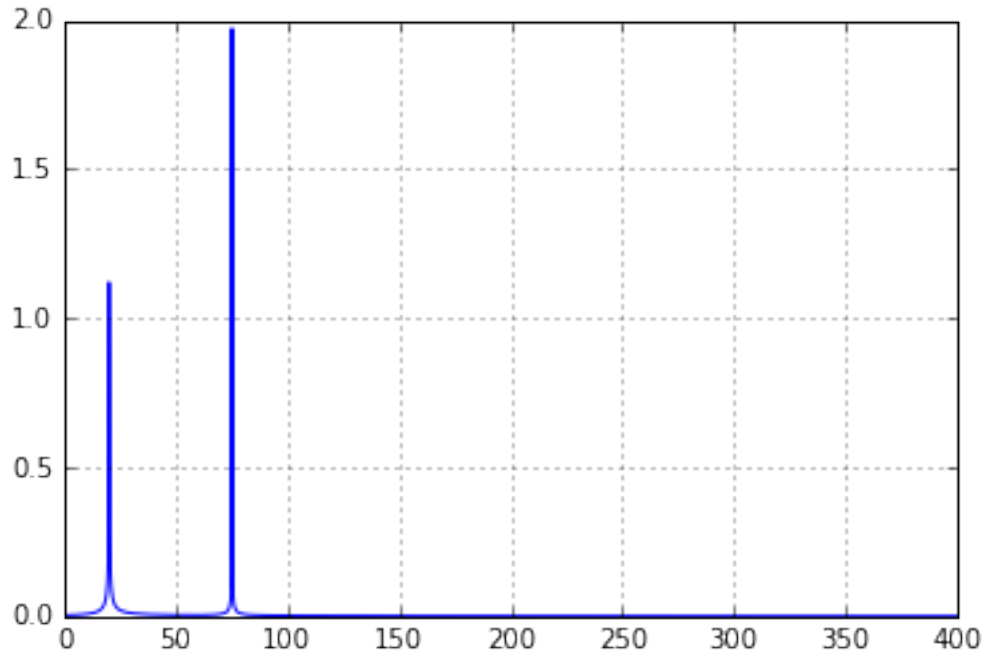


Tempo de compilação para a biblioteca Numpy é de 0.14168572425842285 segundos ---

Resolução do exemplo usando numpy

```
In [8]: start_time = time.time()
        N = 4096
        T = 1.0 / 800.0
        x = nm.linspace(0.0, N*T, N)
        y = 2*nm.cos(5*nm.pi*(30*x) ) + 1.54*nm.cos(2*nm.pi*(20*x) )
        yf = nm.fft.fft(y)
        xf = nm.linspace(0.0, 1.0/(2.0*T), N/2)

        plt.plot(xf, 2.0/N * nm.abs(yf[0:N/2]))
        plt.grid()
        plt.show()
        timePlotNumpyFft=time.time() - start_time
        print("Tempo de compilação para a biblioteca Numpy é de %s segundos ---" % timePlotNumpyFft)
```



Tempo de compilação para a biblioteca Numpy é de 0.1432499885559082 segundos ---

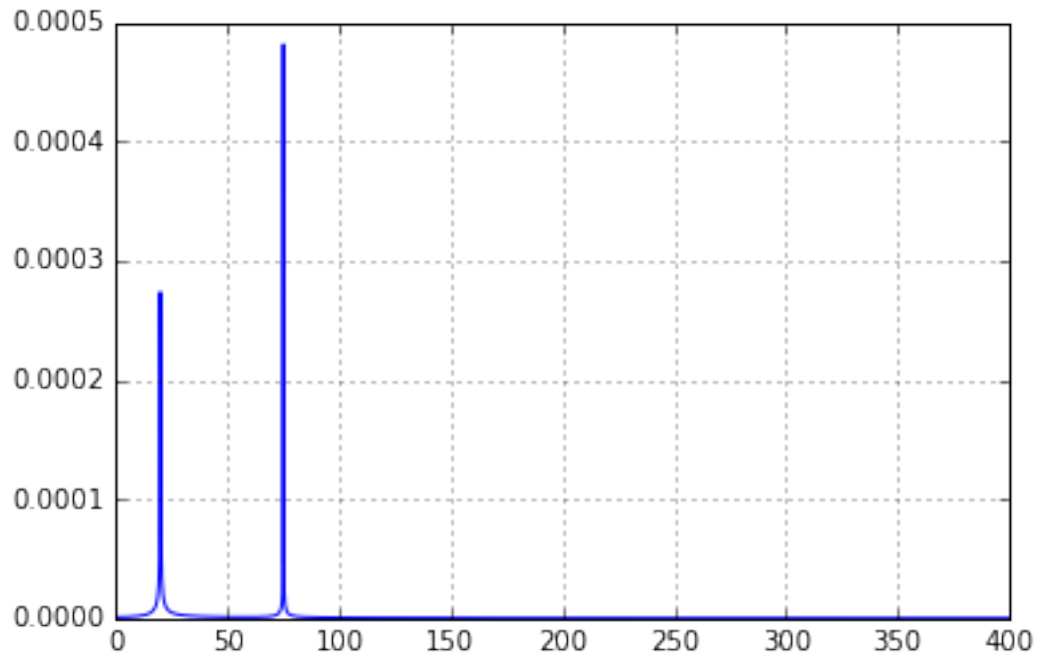
Resolução do exemplo usando numpy

Usando IFFT

Resolução do exemplo usando scipy

```
In [9]: start_time = time.time()
        N = 4096
        T = 1.0 / 800.0
        x = sc.linspace(0.0, N*T, N)
        y = 2*nm.cos(5*nm.pi*(30*x) ) + 1.54*nm.cos(2*nm.pi*(20*x) )
        yf = sc.ifft(y)
        xf = sc.linspace(0.0, 1.0/(2.0*T), N/2)

        plt.plot(xf, 2.0/N * nm.abs(yf[0:N/2]))
        plt.grid()
        plt.show()
        timePlotScipyIfft= time.time() - start_time
        print("Tempo de compilação para a biblioteca Scipy é de %s segundos ---" % timePlotScipyIfft)
```

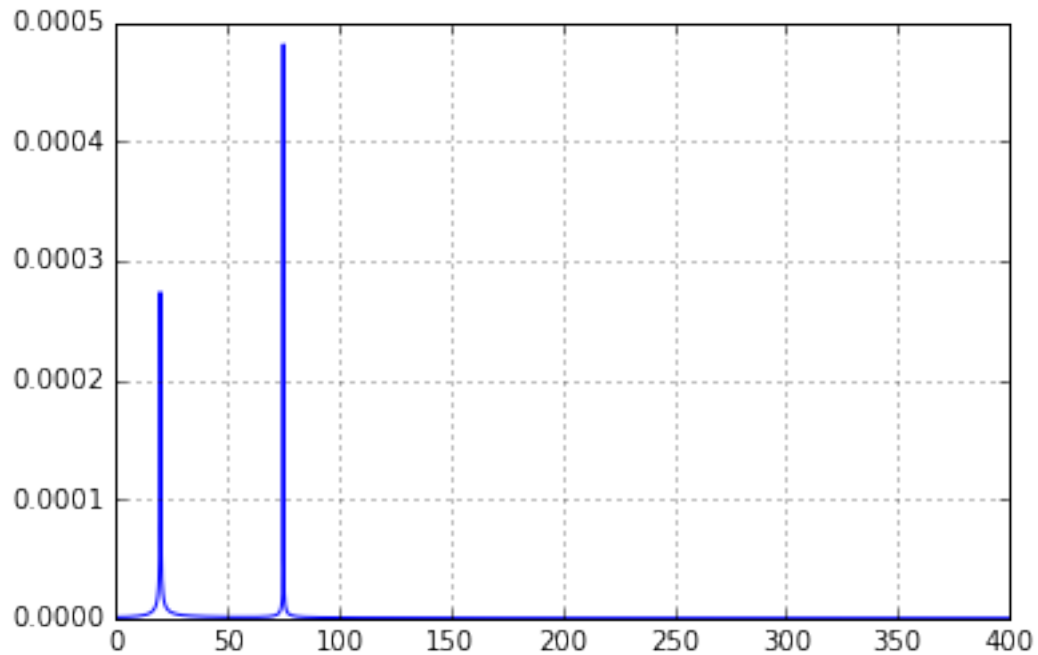


Tempo de compilação para a biblioteca Scipy é de 0.21123075485229492 segundos ---

Resolução do exemplo usando numpy

```
In [10]: start_time = time.time()
         N = 4096
         T = 1.0 / 800.0
         x = nm.linspace(0.0, N*T, N)
         y = 2*nm.cos(5*nm.pi*(30*x) ) + 1.54*nm.cos(2*nm.pi*(20*x) )
         yf = nm.fft.ifft(y)
         xf = nm.linspace(0.0, 1.0/(2.0*T), N/2)

         plt.plot(xf, 2.0/N * nm.abs(yf[0:N/2]))
         plt.grid()
         plt.show()
         timePlotNumpyIfft= time.time() - start_time
         print("Tempo de compilação para a biblioteca Numpy é de %s segundos ---" % timePlotNumpyIfft)
```



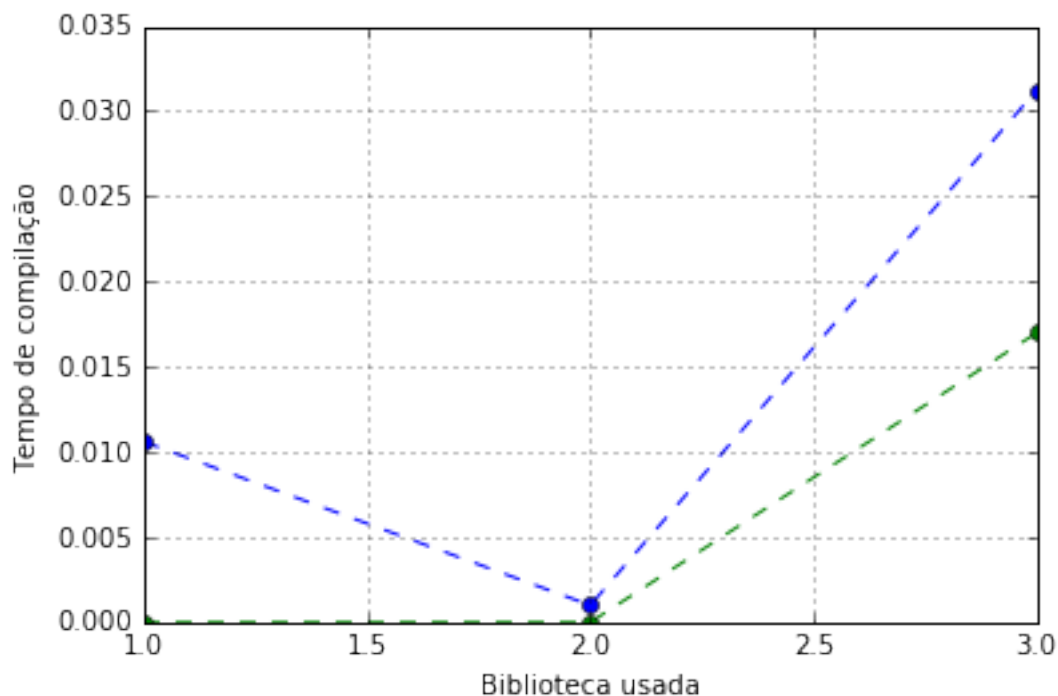
Tempo de compilação para a biblioteca Numpy é de 0.15835905075073242 segundos ---

0.2 # Conclusão

Para o teste com o vetor de 4096 valores aleatórios temos o gráfico abaixo

```
In [11]: k=[1,2,3]
         c=[timeScipy,timeNumpy,timeMyFFt]
         b=[timeScipy2,timeNumpy2,timeMyIFFt]
         plt.grid()
         plt.xlabel('Biblioteca usada')
         plt.ylabel('Tempo de compilação')
         plt.plot(k,c,'--o',k,b,'--o')
```

```
Out[11]: [<matplotlib.lines.Line2D at 0x1eddfa177f0>,
          <matplotlib.lines.Line2D at 0x1eddf901860>]
```

Onde temos que o verde representa o tempo de compilação da IFFT e o azul, temos o tempo de compilação da FFT onde a mesma tende a ter um tempo de compilação menor. A FFT tende nesse caso a ter um tempo de compilação menor, pois o intuito dela é converter um sinal do domínio do tempo para o domínio da frequência.

```
In [12]: table = [{"Scipy - FFT",timeScipy},["Numpy - FFT",timeNumpy],["MyFFt",timeMyFFt],["Scipy - IFFT",timeScipy2],["Numpy - IFFT",timeNumpy2],["MyIFFT",timeMyIFFt]]
          headers = ["Biblioteca usada", "Tempo de compilação (segundos)"]
          print(tabulate(table, headers, tablefmt="grid"))
```

Biblioteca usada	Tempo de compilação (segundos)
Scipy - FFT	0.0105989
Numpy - FFT	0.000999928
MyFFt	0.0312486
Scipy - IFFT	0
Numpy - IFFT	0
MyIFFT	0.0170002

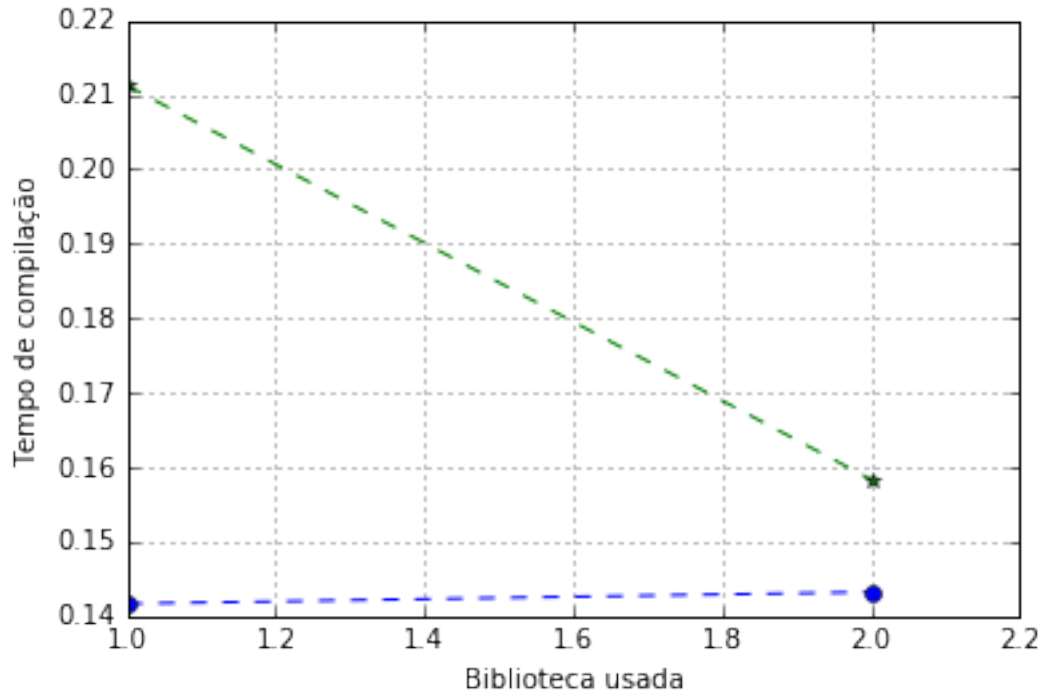
Tabela referente ao tempo gasto para que cada biblioteca compile o cálculo da FFT e IFFT

Conclusão do exemplo

Agora iremos construir um gráfico para melhor visualizarmos de como as bibliotecas se comportam com a resolução completa -cálculo matemático e construção do gráfico- da soma dos cossenos.

```
In [13]: k=[1,2]
         c=[timePlotScipyFft,timePlotNumpyFft]
         b=[timePlotScipyIfft,timePlotNumpyIfft]
         plt.grid()
         plt.xlabel('Biblioteca usada')
         plt.ylabel('Tempo de compilação')
         plt.plot(k,c,'--o',k,b,'--*')
```

```
Out[13]: [<matplotlib.lines.Line2D at 0x1eddf9d6128>,
          <matplotlib.lines.Line2D at 0x1eddfdfa1d0>]
```



Como o esperado, o tempo gasto para poder calcular a FFT tende a ser menor que o tempo gasto para calcular a IFFT

```
In [14]: table = [
["Scipy - FFT",timePlotScipyFft],["Numpy - FFT",timePlotNumpyFft],
["Scipy - IFFT",timePlotScipyIfft],["Numpy - IFFT",timePlotNumpyIfft]]
headers = ["Biblioteca usada", "Tempo de compilação (segundos)"]
print(tabulate(table, headers, tablefmt="grid"))
```

```
+-----+-----+
| Biblioteca usada | Tempo de compilação (segundos) |
+-----+-----+
| Scipy - FFT      | 0.141686 |
+-----+-----+
| Numpy - FFT      | 0.14325  |
+-----+-----+
| Scipy - IFFT     | 0.211231 |
+-----+-----+
| Numpy - IFFT     | 0.158359 |
+-----+-----+
```

Tabela referente ao tempo gasto para que cada biblioteca compile o cálculo e a construção do gráfico da soma dos cossenos, usando FFT e IFFT

Problemas encontrados

Para esse projeto tivemos alguns problemas no decorrer do desenvolvimento. O problema a ser destacado foi o uso e a instalação do pyFFTW, não conseguimos instalar e nem usar, tentamos tanto no windows como no linux, não obtivemos sucesso

<h1>Referências</h1>

Taneja, HC (2008), "Chapter 18: Fourier integrals and Fourier transforms", Advanced Engineering Mat.

www.astro.if.ufrgs.br/med/imagens/fourier.htm

www.en.wikipedia.org/wiki/Fourier_transform#CITEREFTaneja2008

www.docs.scipy.org/doc/numpy-1.10.1/index.html

www.cis.rit.edu/class/simg716/Gauss_History_FFT.pdf

www2.math.ethz.ch/education/bachelor/seminars/fs2008/nas/woerner.pdf

www.en.wikipedia.org/wiki/File:Fourier_transform_time_and_frequency_domains_(small).gif

www.pypi.python.org/pypi/tabulate

www.researchgate.net/post/Why_do_we_use_IFFT