MAP3121 - Exercício Programa 2 - 2018

Oscilações Econômicas e a Transformada de Fourier

Contextualização

Nesta tarefa, comparamos o uso de diferentes implementações da transformada de Fourier discreta. Primeiramente usamos sua forma literal:

$$F(x_j) = \sum_{k=0}^{2N-1} c_k e^{ikx_j}, \;\; j = 0, \dots, 2N-1$$

E a respectiva transformação inversa:

$$c_k = \frac{1}{2N} \sum_{k=0}^{2N-1} F(x_j) e^{-ikx_j}, \;\; k = 0, \dots, 2N-1$$

Entretanto, o cálculo desta forma é custoso e requer $O(N^2)$ operações. Para evitar a complexidade computacional desta implementação, portanto, usa-se a FFT (transformada rápida de fourier) que tem complexidade $O(N \log N)$.

Além da implementação recursiva da FFT indicada no enunciado, também comparamos o uso da implementação deste algoritmo na biblioteca numpy usando as funções rfft e irfft que se aproveitam de propriedades de funções reais para minimizar ainda mais o numero de operações necessárias.

Tarefas Iniciais e verificação

O primeiro passo neste exercício consiste na implementação dos algoritmos sugeridos e teste com funções indicadas. Abaixo indicamos os resultados e comentamos os gráficos obtidos.

Teste a

Neste teste temos que f = [5,-1,3,1].

Implementação ineficiente da Transformada de Fourier direta

==========

| (2.00000, 0.00000i)| (0.50000, 0.50000i)| (2.00000, 0.00000i)| (0.50000, -0.50000i)|

==========

Implementação recursiva da FFT direta

==========

| (2.00000, 0.00000i)| (0.50000, 0.50000i)| (2.00000, 0.00000i)| (0.50000, -0.50000i)|

==========

Fazendo c = (2, 1/2 + i/2, 2, 1/2 − i/2), que é o resultado anterior, e aplicando a transformada inversa, é possível observar que obtemos os mesmos resultados originais.

Implementação ineficiente da Transformada de Fourier inversa

==========

| (5.00000, 0.00000i)| (-1.00000, 0.00000i)| (3.00000, -0.00000i)| (1.00000, 0.00000i)|

==========

Implementação recursiva da FFT inversa

==========

| (5.00000, 0.00000i)| (-1.00000, 0.00000i)| (3.00000, 0.00000i)| (1.00000, -0.00000i)|

==========

Teste b

f = [6, 2, 5, 2, 11, 2, 8, 8]

Implementação ineficiente da Transformada de Fourier direta

==========

| (5.50000, 0.00000i)| (-0.09467, 0.90533i)| (0.50000, 0.75000i)| (-1.15533, 0.15533i)| (2.00000, 0.00000i)| (-1.15533, -0.15533i)| (0.50000, -0.75000i)| (-0.09467, -0.90533i)|

==========

Implementação recursiva da FFT direta

==========

| (5.50000, 0.00000i)| (-0.09467, 0.90533i)| (0.50000, 0.75000i)| (-1.15533, 0.15533i)| (2.00000, 0.00000i)| (-1.15533, -0.15533i)| (0.50000, -0.75000i)| (-0.09467, -0.90533i)|

==========

Implementação recursiva da FFT direta pelo numpy (rfft)

Note que são mostrados apenas os coeficientes de frequências positivas pois o input é real

==========

| (5.50000, 0.00000i)| (-0.09467, 0.90533i)| (0.50000, 0.75000i)| (-1.15533, 0.15533i)| (2.00000, 0.00000i)|

==========

Implementação ineficiente da Transformada de Fourier inversa

==========

| (6.00000, -0.00000i)| (2.00000, 0.00000i)| (5.00000, -0.00000i)| (2.00000, -0.00000i)| (11.00000, -0.00000i)| (2.00000, 0.00000i)| (8.00000, 0.00000i)| (8.00000, 0.00000i)|

==========

Implementação recursiva da FFT inversa

==========

| (6.00000, 0.00000i)| (2.00000, -0.00000i)| (5.00000, 0.00000i)| (2.00000, -0.00000i)| (11.00000, 0.00000i)| (2.00000, 0.00000i)| (8.00000, -0.00000i)| (8.00000, 0.00000i)|

==========

Implementação recursiva da FFT inversa pelo numpy (rfft)

==========

| (6.00000, 0.00000i)| (2.00000, 0.00000i)| (5.00000, 0.00000i)| (2.00000, 0.00000i)| (11.00000, 0.00000i)| (2.00000, 0.00000i)| (8.00000, 0.00000i)| (8.00000, 0.00000i)|

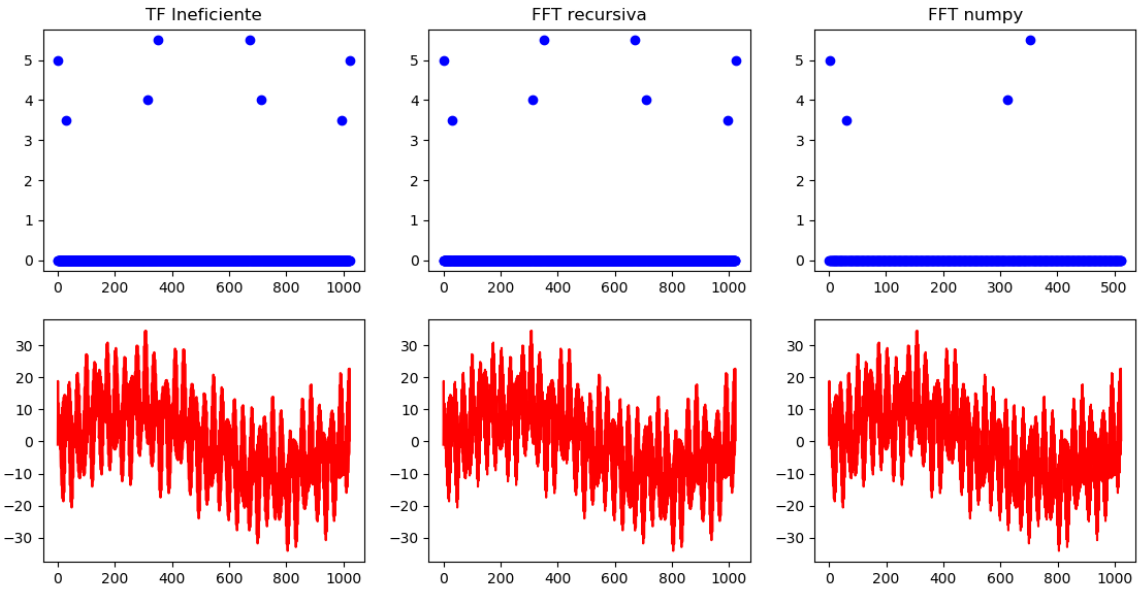
==========

Teste c

Neste teste, obtemos resultados mais interessantes e é possível fazer uma análise mais profunda. Usamos a seguinte função para gerar pontos:

$$F(x) = 10\sin(x) + 7cos(30x) + 11\sin(352x) - 8\cos(711x)$$

Construindo 1024 pontos de $F(x_j)$ e fazendo $x_j = \frac{j\pi}{512}$. Aplicamos a transformada neste vetor de pontos e obtemos o gráfico a seguir, que mostra para cada implementação da transformada, o módulo do espectro e o sinal reconstruído a partir da transformada inversa.



Nesta figura, há dois fatos interessantes que são levantados. O primeiro deles é a simetria do espectro com relação ao meio, que corresponde a frequência angular π. Em sinais reais, há uma simetria destes pontos com relação a frequência 0 (ou a frequência π devido a periodicidade). Por esta razão, inclusive, as funções rfft e irrft retornam um vetor com metade do tamanho, pois se aproveitam das propriedades de sinais reais para diminuir a quantidade de cálculos necessários para a transformada.

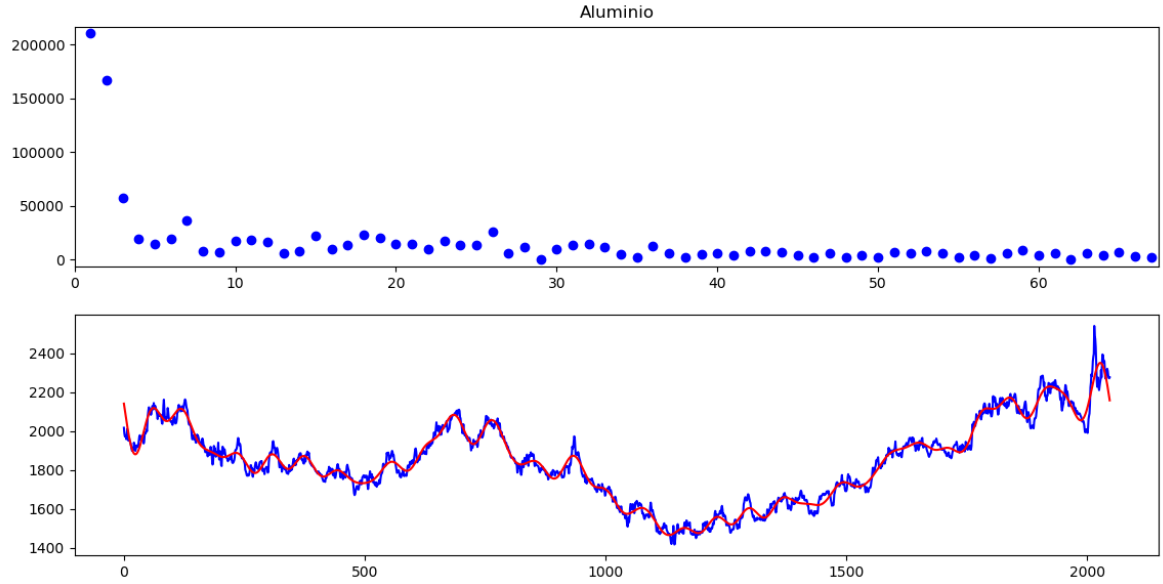
O outro aspecto relevante que podemos observar nesta figura é o fenômeno do rebatimento. Conforme a função original, vemos componentes nos índices 1, 30 e 353. Entretanto há uma componente em 311 também. Isto é devido ao rebatimento da frequência 711, que é superior a frequência de de Nyquist e não pode ser capturada corretamente pela transformada.

Testes da série temporal Alum.csv

Passa baixas

Usando K = 36 e filtro passa baixas obtemos os seguintes nas figuras a seguir. **Note que o espectro mostrado descarta a frequência 0 e está com zoom para que seja possível observar melhor as frequências. Além disto, é mostrado o espectro do sinal antes de utilizar o filtro passa baixas.**

Na figura abaixo mostramos o resultado para implementação com as funções do **numpy**



==========

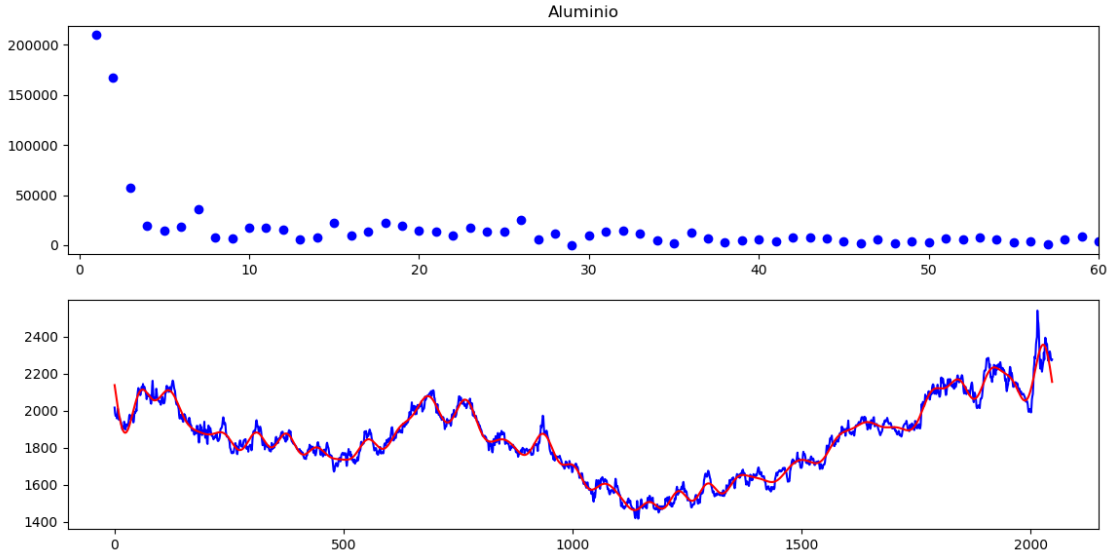
Tempo decorrido para transformada direta: 0.000000 ms

Tempo decorrido para transformada inversa: 0.000000 ms

Desvio Padrao da serie temporal: 201.609439

==========

Na figura abaixo mostramos o resultado para implementação com as funções **fft recursivas** com o algoritmo sugerido no enunciado



==========

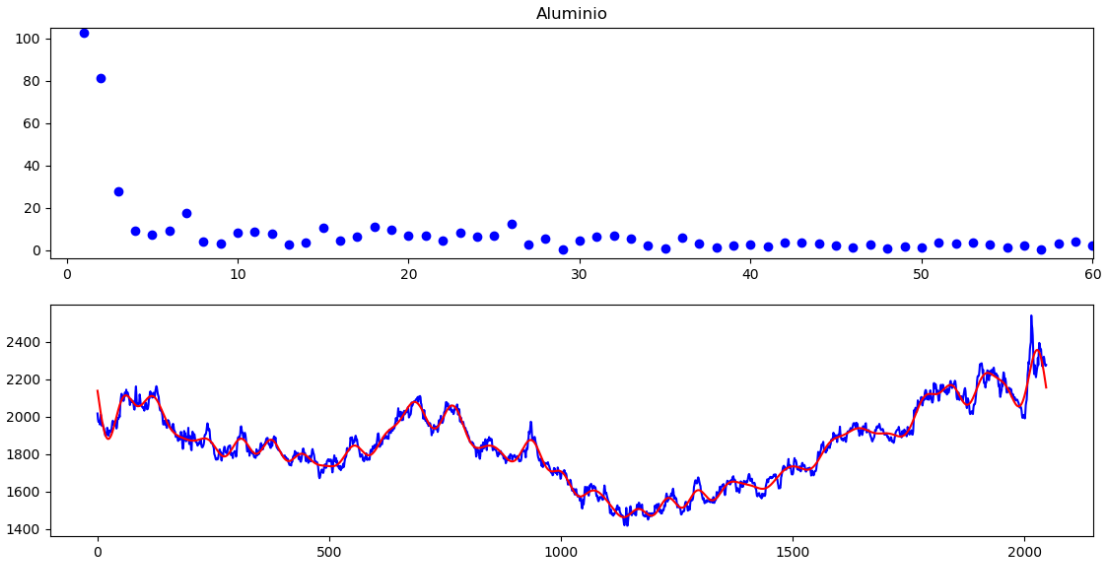
Tempo decorrido para transformada direta: 109.373331 ms

Tempo decorrido para transformada inversa: 124.990940 ms

Desvio Padrao da serie temporal: 201.609439

==========

Na figura abaixo mostramos o resultado para implementação com as funções **Transformadas lentas**



==========

Tempo decorrido para transformada direta: 21531.363010 ms

Tempo decorrido para transformada inversa: 18397.241831 ms

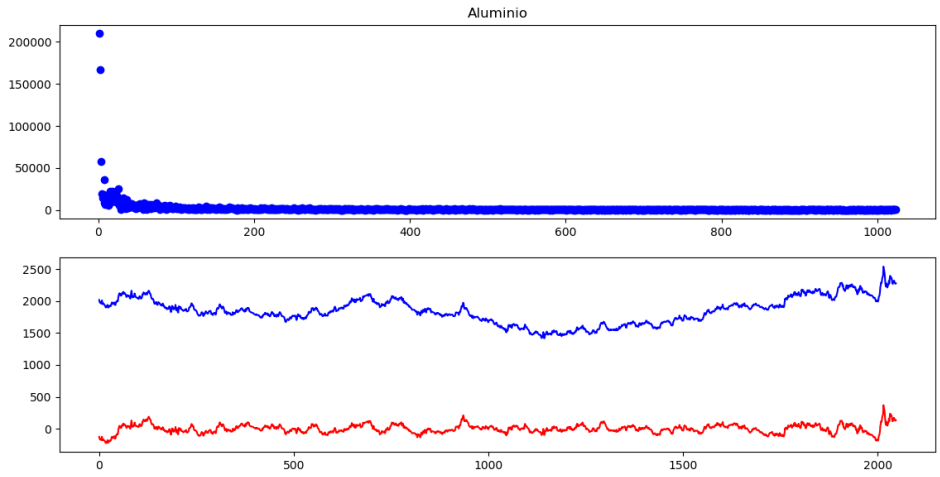
Desvio Padrao da serie temporal: 201.609439

==========

Passa Altas

Agora, iremos mostrar o uso do **filtro passa altas**, usando **K = 8**. Obtemos os resultados demonstrados a seguir. Note que o espectro mostrado descarta a frequência 0 e **não está com zoom** para que seja possível observar o resultado completo do vetor**.** Além disto, é mostrado o espectro do sinal antes de utilizar o filtro passa baixas.

Na figura abaixo mostramos o resultado para implementação com as funções do **numpy**



==========

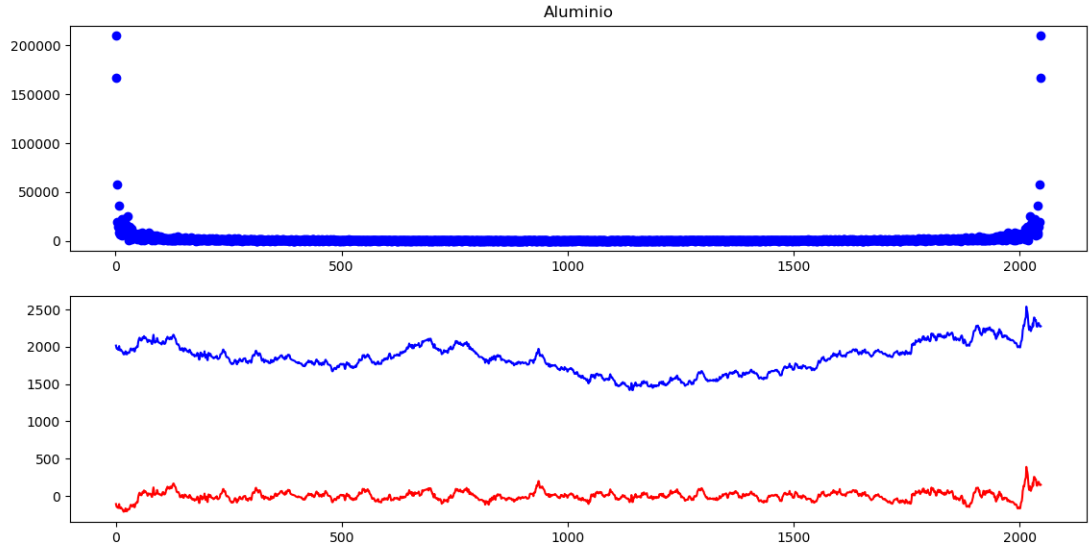
Tempo decorrido para transformada direta: 0.000000 ms

Tempo decorrido para transformada inversa: 0.000000 ms

Desvio Padrao da serie temporal: 201.609439

==========

Na figura abaixo mostramos o resultado para implementação com as funções **fft recursivas** com o algoritmo sugerido no enunciado



==========

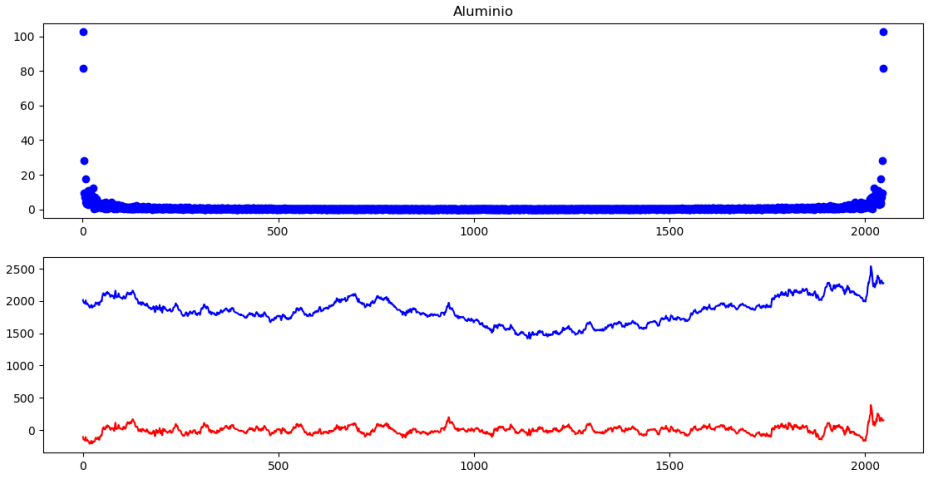
Tempo decorrido para transformada direta: 93.741894 ms

Tempo decorrido para transformada inversa: 109.975576 ms

Desvio Padrao da serie temporal: 201.609439

==========

Na figura abaixo mostramos o resultado para implementação com as funções **Transformadas lentas**



==========

Tempo decorrido para transformada direta: 20967.803001 ms

Tempo decorrido para transformada inversa: 17639.755487 ms

Desvio Padrao da serie temporal: 201.609439

==========