Elementos de Processamento de Sinais: Lista de Exercícios #1

Data de entrega: Quinta, 11 de Julho, 2019

Prof. Sergio Lima Netto, Segundas e Quartas: 08:00–10:00

Vinicius Mesquita de Pinho

Questão 1

Primeiro, geramos uma senoide com frequência de $f_c = 80$ Hz, amostrada com frequência $f_s = 1$ kHz. A mesma pode ser vista na Figura 1.
O nosso sinal é então:

$$x(n) = \sin(\theta) = \sin(2\pi f_c n).$$

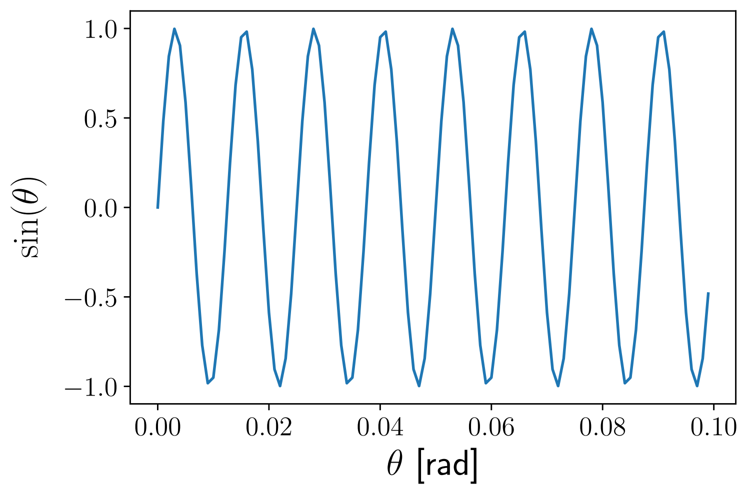


Figura 1: Seno em função do ângulo θ, representado de 0 a 0.1 radianos.

O próximo passo foi gerar um ruído gaussiano. Como uma espécie de teste de sanidade, temos o histograma apresentado na Figura 2. A curva vermelha foi plotada a partir da equação da PDF da Gaussiana, e podemos verificar visualmente que os dados gerados em azul seguem a uma distribuição normal.
O ruído será representado por $\zeta(n)$.

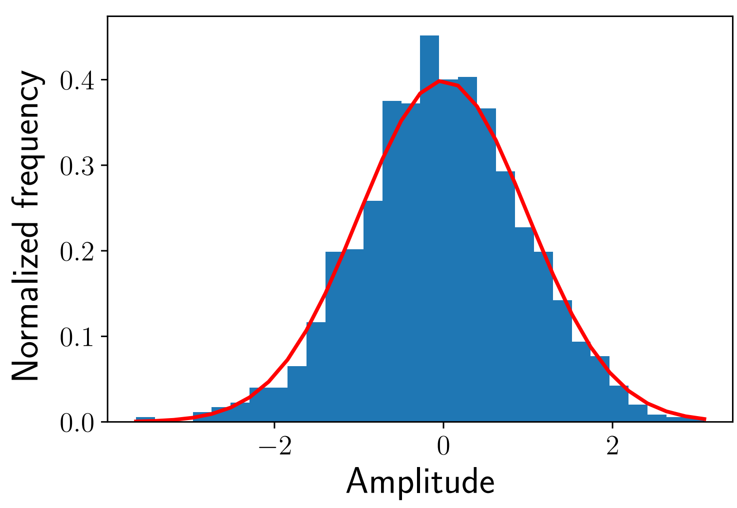


Figura 2: Histograma do ruído gaussiano gerado.

A Figura 3 é o resultado de $x(n)$ da Figura 1 e do ruído representado no histograma da Figura 2. A operação pode ser escrita como:

$$y(n) = x(n) + \zeta(n).$$

Colocamos uma ganho $k = 0.2$ para demonstrar o resultado da adição do ruído e ainda conseguirmos visualizar a senoide, dada a amplitude do ruído $\zeta(k)$ como vista na Figura 2.

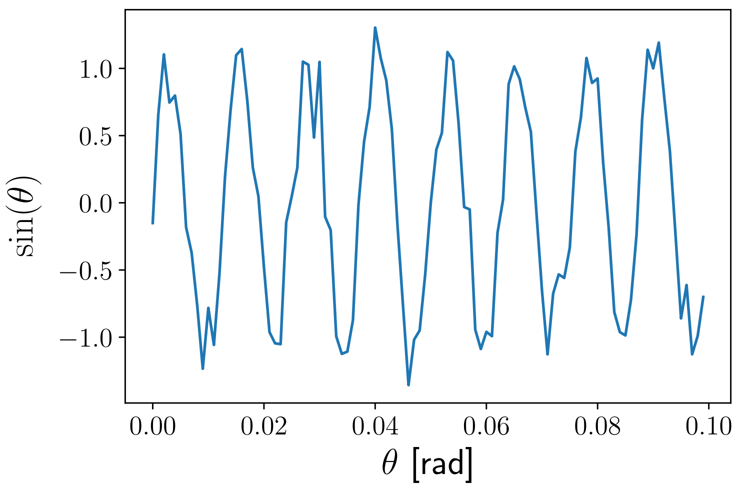


Figura 3: Função seno adicionada do ruído gaussiano. O ganho no ruído neste caso é de $k = 0.2$.

Sendo a DFT de $y(n)$ expressa como:

$$Y(m) = \sum_{l=0}^{L-1} y(l) W_L^{li}, \text{ para } 0 \leq m \leq L-1, \tag{1}$$

na Figura 4 temos a parte positiva $|Y(f)|$, onde f é a frequência em Hz.
Para obtermos uma estimativa da frequência f_c da senoide $x(n)$, analisar de forma visual e verificar que em $f = 80$ Hz encontra-se um pico de $|Y(f)|$, portanto podemos dizer que a estimativa para a frequência da senoide é $\hat{f}_c = 80$ Hz.

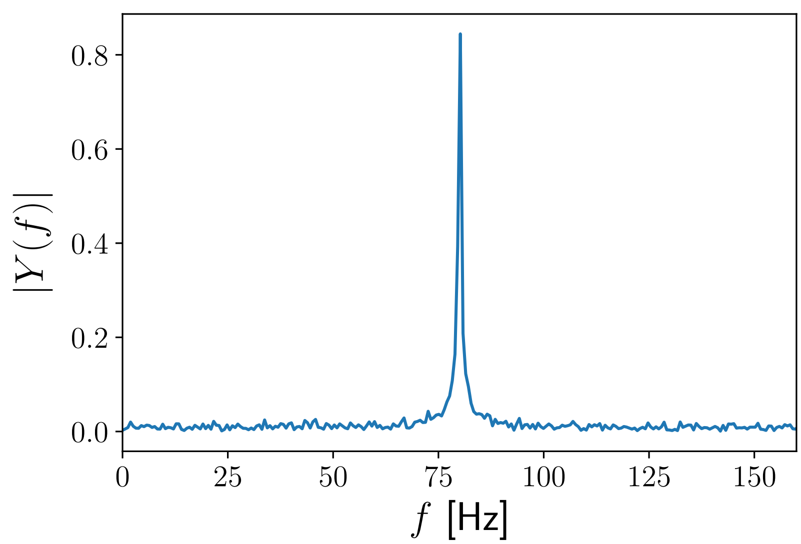
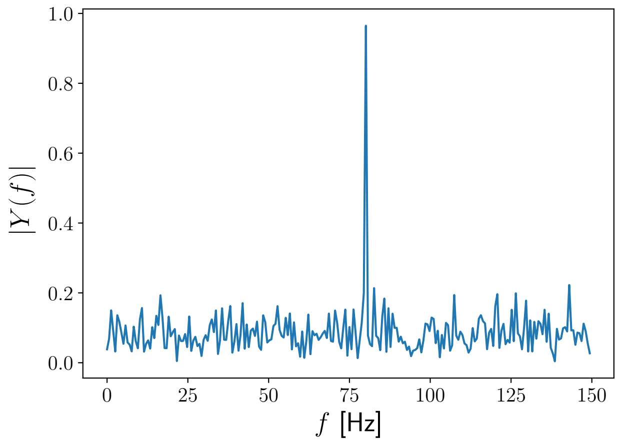


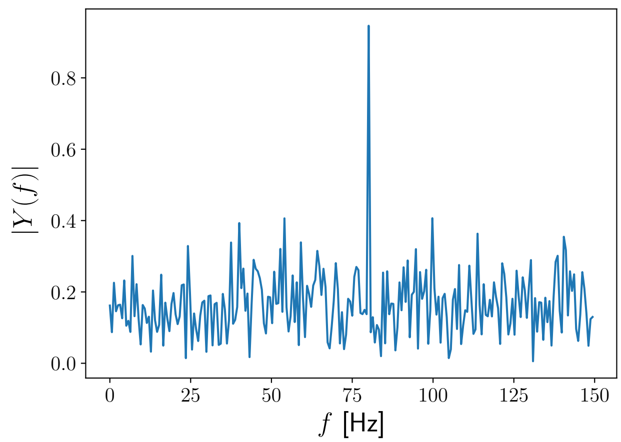
Figura 4: Módulo da DFT do sinal adicionado de ruído.

Para automatizarmos o processo, podemos construir um algoritmo que identifique o máximo de $|Y(f)|$ e nos retorne em qual f ele ocorre. Para o exemplo ilustrado na Figura 4, a resposta foi $\hat{f}_c = 80.2$ Hz.

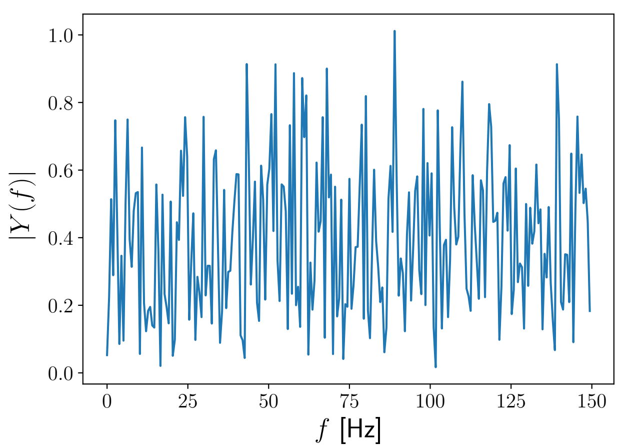
Porém este algoritmo nem sempre nos informará uma resposta correta. Quando o ganho k do ruído aumenta, perdemos a referência do pico. Como podemos ver na Figura 5, que exemplifica este fato com diferentes valores para o ganho k. Portanto, a estimação da frequência por este meio não traz um resultado satisfatório quando o ruído tem uma amplitude tão grande quanto o pico do valor absoluto da DFT.



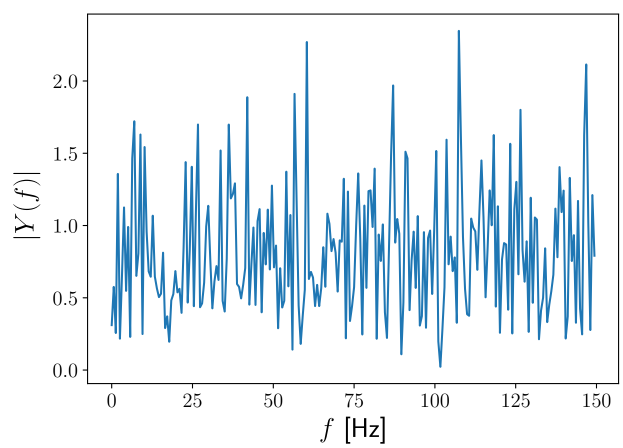
(a) $k = 1$, $\hat{f}_c = 80.08$ Hz



(b) $k = 2$, $\hat{f}_c = 80.08$ Hz



(c) $k = 5$, $\hat{f}_c = 88.98$ Hz



(d) $k = 10$, $\hat{f}_c = 107.42$ Hz

Figura 5: Efeito do aumento do ganho do ruído k na estimativa $\hat{f}_c$ da frequência da senoide.

Para resolvermos este problema, podemos fazer um pré-processamento no sinal recebido, como por exemplo fazer o realce do sinal desejado, com um filtro levando em conta a estatística do ruído e a correlação dele com o sinal. Depois do realce do sinal, podemos assim estimar a frequência da portadora.

Questão 2

Esta questão tem como objetivo calcular a convolução no tempo e na frequência. Observar a necessidade de *zero-padding* para o caso da frequência.

Para este exercícios, vamos definir dois vetores:

$$\mathbf{x} = [1 \quad 2]^{\mathrm{T}} \quad \text{e} \quad \mathbf{h} = [1 \quad 0]^{\mathrm{T}}.$$

O tamanho os dois é de $L_x = L_h = 2$. Portanto, é de se esperar que a convolução entre os dois sinais tenha um tamanho $L_y = L_x + L_h - 1 = 3$.
Usando a função do Numpy chamada convolve, o resultado que nos é apresentado é:

$$\mathbf{y} = \mathbf{x} * \mathbf{h} = [1 \quad 2 \quad 0]^{\mathrm{T}}.$$

Portanto, L_y de fato é 3.
Podemos também fazer a conta escrevendo $\mathbf{x}$ como uma matriz Toeplitz, que fica com a forma:

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{bmatrix}$$

Ao fazermos $\mathbf{y} = \mathbf{Xh}$ temos:

$$\mathbf{y} = \mathbf{Xh} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \mathbf{x} * \mathbf{h} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}$$

Também podemos fazer essas mesmas operações no domínio da frequência. Primeiro, vamos explorar o caso em que não fazemos o *zero-padding*. Analisaremos o caso em que temos $\mathbf{x}$ e $\mathbf{h}$ tais que:

$$\mathbf{x} = [1 \quad 2]^{\mathrm{T}} \qquad \mathbf{h} = [1 \quad 0]^{\mathrm{T}}$$

Usando a função *fftconvolve* do Scipy como um *sanity check*, o resultado é:

$$\mathbf{y}' = \begin{bmatrix} 1 & 2 \end{bmatrix}^{\mathrm{T}}.$$

Agora vamos fazer de outra forma. Vamos calcular a DFT de $\mathbf{x}$ e de $\mathbf{h}$, multiplicá-los e calcular a IDFT do resultado.
Para ambos, a matriz de DFT é:

$$\mathbf{W}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Então temos:

$$\mathbf{X}' = \mathbf{W}_2\mathbf{x} = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

e

$$\mathbf{H}' = \mathbf{W}_2\mathbf{h} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Portanto,

$$\mathbf{Y}' = \mathbf{X}' \odot \mathbf{H}' = \begin{bmatrix} 3 \\ -1 \end{bmatrix}.$$

Agora, para fazermos a IDFT, a matriz será:

$$\mathbf{W}_2^{-1} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & -0.5 \end{bmatrix}.$$

Questão 2 continua na próxima página. . .

Portanto:

$$\mathbf{y}' = \mathbf{W}_2^{-1}\mathbf{Y}' = \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

Ok, este resultado não é esperado para a convolução linear. Para atingirmos o resultado esperado para convolução linear é necessário o *zero-padding*. Neste caso, como resultado da convolução linear tem que ter tamanho $L_y = 3$, os dois vetores vão ser acrescidos de 1 zero no final. Portanto, teremos:

$$\tilde{\mathbf{x}} = [1 \quad 2 \quad 0]^{\mathrm{T}} \qquad \tilde{\mathbf{h}} = [1 \quad 0 \quad 0]^{\mathrm{T}}$$

Usando agora a função *fftconvolve* do Scipy, temos o resultado esperado:

$$\mathbf{y} = [1 \quad 2 \quad 0]^{\mathrm{T}}.$$

Fazendo novamente o cálculo pela matriz de DFT. A matriz $\mathbf{W}_3$ neste caso é:

$$\mathbf{W}_3 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -0.5 - \mathrm{j}0.866 & -0.5 + \mathrm{j}0.866 \\ 1 & -0.5 + \mathrm{j}0.866 & -0.5 - \mathrm{j}0.866 \end{bmatrix}.$$

Então temos:

$$\tilde{\mathbf{X}} = \mathbf{W}_3\tilde{\mathbf{x}} = \begin{bmatrix} 3 \\ -\mathrm{j}1.732 \\ +\mathrm{j}1.732 \end{bmatrix},$$

$$\tilde{\mathbf{H}} = \mathbf{W}_3\tilde{\mathbf{h}} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Portanto,

$$\tilde{\mathbf{Y}} = \tilde{\mathbf{X}} \odot \tilde{\mathbf{H}} = \begin{bmatrix} 3 \\ -\mathrm{j}1.732 \\ +\mathrm{j}1.732 \end{bmatrix}.$$

Agora, para fazermos a IDFT, a matriz é

$$\mathbf{W}_3^{-1} = \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & -1/6 + \mathrm{j}0.289 & -1/6 - \mathrm{j}0.289 \\ 1/3 & -1/6 - \mathrm{j}0.289 & -1/6 + \mathrm{j}0.289 \end{bmatrix}.$$

Por fim:

$$\mathbf{y} = \tilde{\mathbf{y}} = \mathbf{W}_3^{-1}\tilde{\mathbf{Y}} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}.$$

Agora vamos analisar o caso em que um dos sinais é muito maior do que o outro. Primeiro vamos fazer um *overlap and add* manual. Depois vamos comparar tempos de processamentos entre as formas diferentes de cálculo da convolução.

Primeiramente, vamos fazer um caso para os seguintes vetores:

$$\mathbf{x} = [1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 7 \quad 19 \quad 0]^{\mathrm{T}} \quad \text{e} \quad \mathbf{h} = [1 \quad 2]^{\mathrm{T}}.$$

O que queremos calcular é: $\mathbf{y} = \mathbf{x} * \mathbf{h}$, para isso faremos o processo a seguir.

Dividindo $\mathbf{x}$ em blocos de dois elementos.

$$\mathbf{x}_1 = [1 \quad 2]^{\mathrm{T}}, \qquad \mathbf{x}_2 = [3 \quad 4]^{\mathrm{T}}, \qquad \mathbf{x}_3 = [5 \quad 7]^{\mathrm{T}} \quad \text{e} \quad \mathbf{x}_4 = [19 \quad 0]^{\mathrm{T}}.$$

Ao fazermos a convolução linear de cada bloco com $\mathbf{h}$, teremos $\mathbf{y}_i = \mathbf{h} * \mathbf{x}_i,\ i = 1, 2, 3$ e 4.:

$$\mathbf{y}_1 = [1 \quad 1 \quad 4]^{\mathrm{T}}, \qquad \mathbf{y}_2 = [3 \quad 10 \quad 8]^{\mathrm{T}}, \qquad \mathbf{y}_3 = [5 \quad 17 \quad 14]^{\mathrm{T}}, \quad \text{e} \quad \mathbf{y}_4 = [19 \quad 38 \quad 0]^{\mathrm{T}}.$$

Para o nosso problema original, o tamanho do resultado final deve ser: $L_{\mathbf{y}} = L_{\mathbf{x}} + L_{\mathbf{h}} - 1 = 8 + 2 - 1 = 9$. Vemos que $L_{\mathbf{y}_1} + L_{\mathbf{y}_2} + L_{\mathbf{y}_3} + L_{\mathbf{y}_4} = 12$. Para conseguirmos o resultado correto, temos que fazer o *overlap and add* como mostrado pelo livro-texto, que é basicamente concatenar os vetores e somar com sobreposição de $L_{\mathbf{h}} - 1$ elementos, neste caso teremos um *overlap* de 1 unidade. Então o resultado final é:

$$\mathbf{y} = [1 \quad 4 \quad 7 \quad 10 \quad 13 \quad 17 \quad 33 \quad 38 \quad 0]^{\mathrm{T}}.$$

Este caso foi para mostrar como o *overlap and add* pode ser feito. Para vermos diferenças no tempo de processamento, realizei algumas medições no Python. Criei vetores $\mathbf{x}$ de tamanho 1024 que convoluí com vetores $\mathbf{h}$ de tamanho 4, ambos com números gerados de forma aleatória. Realizei a operação 1 milhão de vezes, e então calculei o tempo por operação. Para as mesmas configurações, fiz a operação de convolução de linear com *zero-padding* e calculando a DFT e sua inversa.
Como para ambas eu mesmo fiz o código, o que significa que ele não é otimizado, os tempos por operação tornam-se bem altos. Tivemos 10 milissegundos para a operação com *overlap and add* e 5 milissegundos para cada operação com a matriz de DFT. Acredito que pela matriz de DFT foi mais rápido porque o Python tem jeitos espertos de calcular essa inversa por conta da estrutura da matriz de DFT.