

# EFC1 - Sistemas LIT e Convolução

*Rafael Gonçalves (RA:186062)*

*March 21, 2018*

## Parte Teórica

a)

Dado que:

(A)  $h[n]$  é não nulo em  $[0, D - 1]$ , então  $h[n - k]$  é não nulo para  $n$  em  $[k, D + k - 1]$ .

(B)  $s[k]$  é não nulo em  $[0, K - 1]$ .

(C) Para todo  $n$ :

$$x[n] = \sum_{k=-\infty}^{\infty} s[k]h[n - k]$$

Por (C),  $x[n]$  é não nulo quando tanto  $s[k]$  quanto  $h[n - k]$  são não nulos.

Por (A), para cada  $k$ ,  $x[n]$  é não nulo para  $k \leq n \leq k + D - 1$ .

Por (B), para  $x[n]$  ser não nulo  $k$  está limitado tal que  $0 \leq k \leq K - 1$ .

Definindo:

$$x_k[n, k], \quad k \leq n \leq D - 1 + k \quad \text{e} \quad k = 1, 2, \dots, K - 1 \quad (1)$$

Ou seja:

$x_0$  é não nulo se  $n \in [0, D - 1 + 0] = [0, D - 1]$

$x_1$  é não nulo se  $n \in [1, D - 1 + 1] = [1, D]$

...

$x_{K-1}$  é não nulo se  $n \in [K - 1, D - 1 + K - 1] = [K - 1, D + K - 2]$

Como  $x[n] = \sum_{k=-\infty}^{\infty} x_k$  então  $x[n]$  é não nulo no intervalo em que pelo menos um  $x_k$  é não nulo. E isso pode ser expresso como a união dos intervalos em que  $x_k$ ,  $k = 1, 2, \dots, K - 1$  é não nulo:

$$n = [0, D - 1] \cup [1, D - 1 + 1] \cup \dots \cup [K - 1, D - 1 + K - 1] \quad (2)$$

Então:

$$0 \leq n \leq K + D - 2 \quad (3)$$

O que representa um número de amostras igual a:

$$P = K + D - 1 \quad (4)$$

b)

Dado

$$\mathbf{x} = \mathbf{H}\mathbf{s} \quad (5)$$

$$\mathbf{x} = [x_0, x_1, \dots, x_{P-1}]^T \quad (6)$$

$$\mathbf{x} \in \mathbb{R}^{P \times 1}, \quad \mathbf{H} \in \mathbb{R}^{P \times K}, \quad \mathbf{s} \in \mathbb{R}^{K \times 1}$$

Temos

$$x_i = \sum_{k=0}^{K-1} h_{ik} s_k, \quad i = 0, 1, \dots, P-1 \quad (7)$$

Para que  $x_i$  seja igual a  $x[n]$ :

$$x_i = \sum_{k=0}^{K-1} h_{ik} s_k = \sum_{k=-\infty}^{\infty} s[k] h[n-k] = x[n] \quad (8)$$

Como:

$$\sum_{k=-\infty}^{-1} s[k] h[n-k] = \sum_{k=K}^{\infty} s[k] h[n-k] = 0 \quad (9)$$

$$s_k = s[k] \quad (10)$$

Então:

$$x_i = \sum_{k=0}^{K-1} h_{ik} s[k] = \sum_{k=0}^{K-1} h[n-k] s[k] = x[n] \quad (11)$$

$$h_{ik} = h[n-k], \quad i = n \quad (12)$$

$$\mathbf{H} = \begin{bmatrix} h[0-0] & h[0-1] & \dots & h[0-(K-1)] \\ h[1-0] & h[1-1] & \dots & h[1-(K-1)] \\ \vdots & \vdots & \ddots & \vdots \\ h[P-1-0] & h[P-1-1] & \dots & h[P-1-(K-1)] \end{bmatrix} \quad (13)$$

$$\mathbf{H} = \begin{bmatrix} h[0] & h[-1] & \dots & h[1-K] \\ h[1] & h[0] & \dots & h[2-K] \\ \vdots & \vdots & \ddots & \vdots \\ h[P-1] & h[P-2] & \dots & h[P-K] \end{bmatrix} \quad (14)$$

Com cada elemento  $h_{ij} = 0$  para todo  $i, j$  se  $i-j < 0$  ou  $i-j \geq D$

## Parte Computacional

c)

Como:

$$x[n] = s[n] - 0.5s[n-1] \quad (15)$$

Se substituirmos  $s[n] = \delta[n]$  em (15), então:

$$h[n] = \delta[n] - 0.5\delta[n-1] \quad (16)$$

Ou ainda:

$$\mathbf{h} = [1, -0.5] \quad (17)$$

d)

Supondo:

$$\mathbf{y} = \mathbf{W}\mathbf{x} = \mathbf{s} \quad (18)$$

$$\mathbf{x} = \mathbf{H}\mathbf{s} \quad (19)$$

Temos:

$$\mathbf{y} = \mathbf{W}\mathbf{H}\mathbf{s} = \mathbf{s} \quad (20)$$

Portanto a resposta combinada é:

$$\mathbf{W}\mathbf{H} = \mathbf{I} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \quad (21)$$

E a saída do sistema é:

$$\mathbf{y} = \mathbf{I}\mathbf{s} = \mathbf{s} \quad (22)$$

e)

Rotina para implementação de forma vetorial de convolução (em python 3):

```
import numpy as np
# convolucao
def conv(h, x):
    H = getH(h, x)
    return np.dot(H, x)
# construo da matriz H a partir de h (resposta ao impulso) e x (entrada)
def getH(h, x):
    K = len(x)
    D = len(h)
    P = K + D - 1
    H = np.zeros([P,K])
    for i in range(P):
        for j in range(K):
            if (i - j) in range(D):
                H[i,j] = h[(i-j)]
            else:
                H[i,j] = 0
    return H
```

```
h = np.array([[1], [-0.5]])
w1 = np.array([1,0.5,0.5**2,0.5**3,0.5**4])
w2 = np.array([1,1.5,0.7,-0.2, 0.3])
```

```
print("w1:", '\n', conv(w1, h))
```

```
## w1:
## [[ 1.    ]
## [ 0.    ]
## [ 0.    ]
## [ 0.    ]
## [ 0.    ]
## [-0.03125]]
```

```
print("w2:", '\n', conv(w2, h))
```

```
## w2:
## [[ 1.   ]
## [ 1.   ]
## [-0.05]
## [-0.55]
## [ 0.4 ]
## [-0.15]]
```

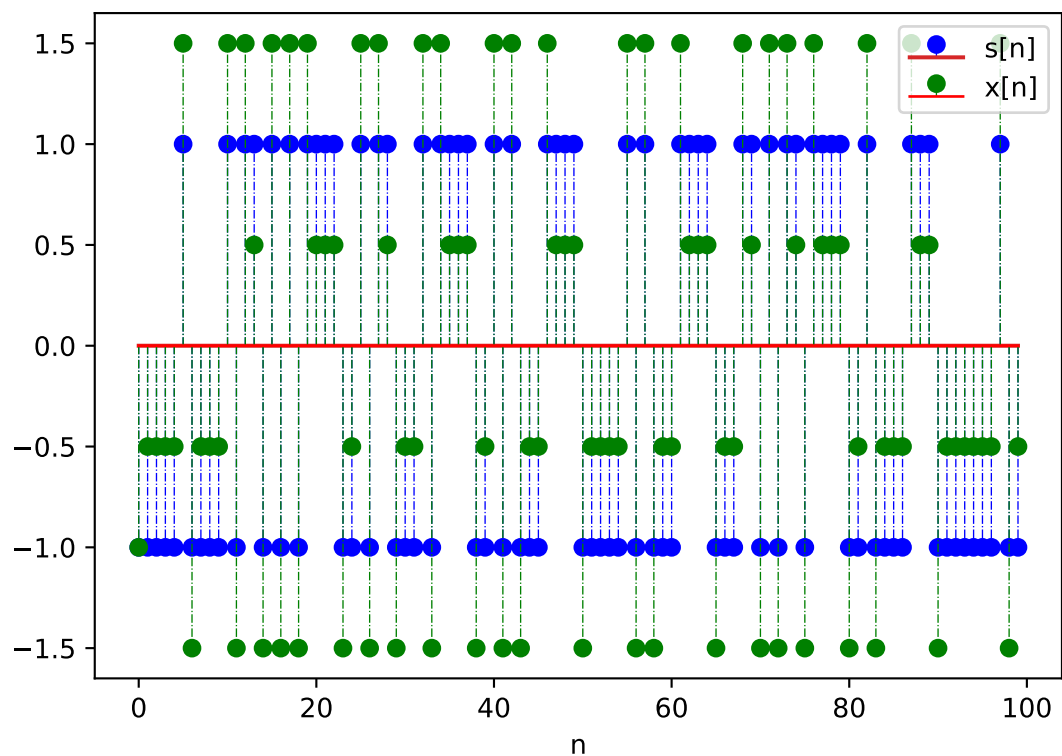
Estamos buscando construir um sistema (com resposta ao impulso  $w[n]$ , e entrada  $x[n]$ ) que devolva um sinal o mais semelhante possível com o sinal inicial sem o ruído (sinal  $s[n]$ ). Como  $h$  é a resposta quando  $s[n] = \delta[n]$  então a saída  $y$  deveria ser igual a  $\delta[n]$  ( $w[n] = 1$  para  $n = 0$  e  $w[n] = 0$  para  $n \neq 0$ ).

O filtro  $w1$  apresentou uma resposta muito próxima ao ideal (valor 1 em  $n = 0$ , e zero para todos os outros valores de  $n$ ). Já o filtro  $w2$  não apresentou uma boa resposta.

f)

```
# funcao signum
sign = lambda X: [e/abs(e) for e in X[0,:]]
# 100 amostras
s = np.array(sign(np.random.randn(1,100)))
#
x = conv(h,s)

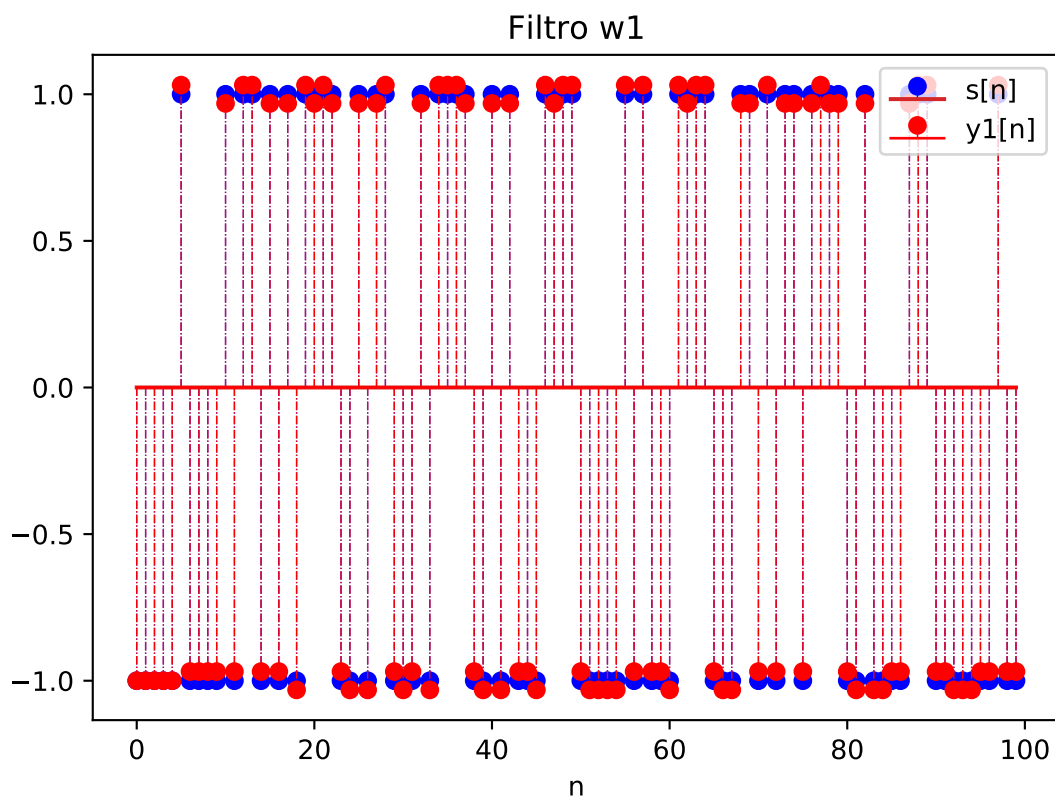
import matplotlib.pyplot as plt
import numpy as np
n = np.linspace(0, 99, 100)
markerline1, stemlines1, baseline = plt.stem(n, s, '-.')
markerline2, stemlines2, baseline = plt.stem(n, x[:-1], '-.')
plt.setp(baseline, 'color', 'r', 'linewidth', 1)
plt.setp(stemlines1, 'color', 'b', 'linewidth', 0.5)
plt.setp(stemlines2, 'color', 'g', 'linewidth', 0.5)
plt.setp(markerline1, 'color', 'b', 'linewidth', 1)
plt.setp(markerline2, 'color', 'g', 'linewidth', 1)
plt.xlabel('n')
plt.legend(['s[n]', 'x[n]'], loc=1)
plt.show()
```



g)

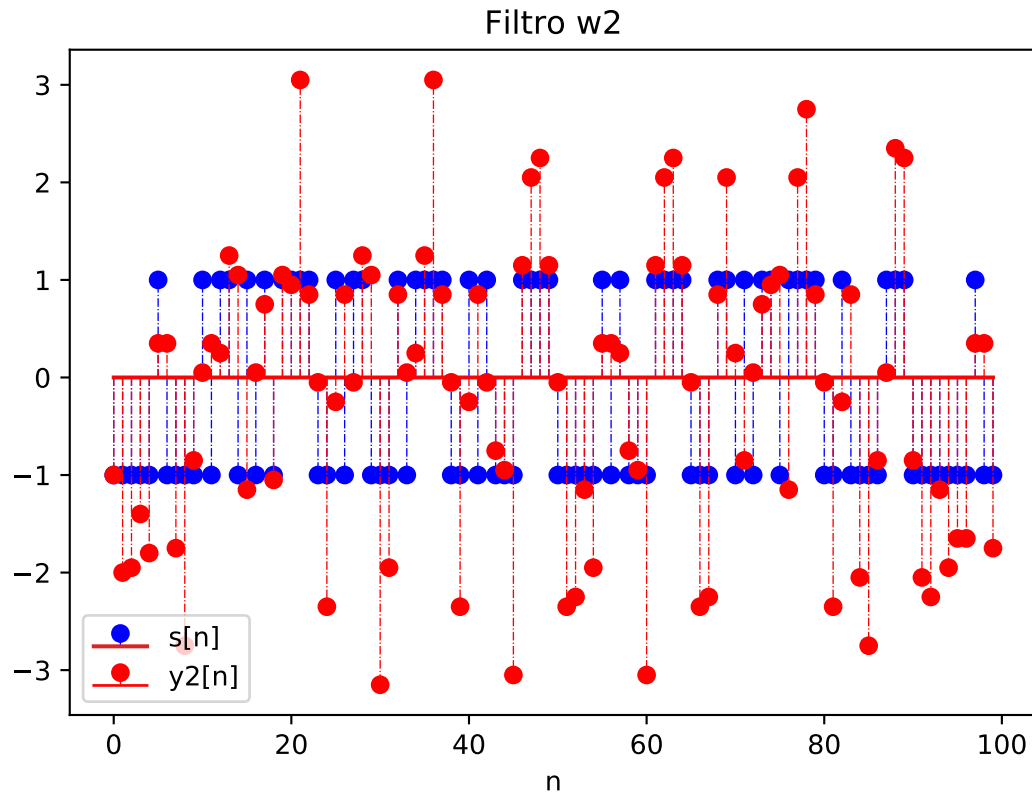
```
y1 = conv(w1,x)
y2 = conv(w2,x)
```

```
plt.clf()
n = np.linspace(0, 99, 100)
markerline1, stemlines1, baseline = plt.stem(n, s, '-.')
markerline2, stemlines2, baseline = plt.stem(n, y1[:100], '-.')
plt.setp(baseline, 'color', 'r', 'linewidth', 1)
plt.setp(stemlines1, 'color', 'b', 'linewidth', 0.5)
plt.setp(stemlines2, 'color', 'r', 'linewidth', 0.5)
plt.setp(markerline1, 'color', 'b', 'linewidth', 1)
plt.setp(markerline2, 'color', 'r', 'linewidth', 1)
plt.title('Filtro w1')
plt.xlabel('n')
plt.legend(['s[n]', 'y1[n]'], loc=1)
plt.show()
```



```
plt.clf()
markerline1, stemlines1, baseline = plt.stem(n, s, '-.')
markerline2, stemlines2, baseline = plt.stem(n, y2[:100], '-.')
plt.setp(baseline, 'color', 'r', 'linewidth', 1)
plt.setp(stemlines1, 'color', 'b', 'linewidth', 0.5)
plt.setp(stemlines2, 'color', 'r', 'linewidth', 0.5)
plt.setp(markerline1, 'color', 'b', 'linewidth', 1)
```

```
plt.setp(markerline2, 'color', 'r', 'linewidth', 1)
plt.title('Filtro w2')
plt.xlabel('n')
plt.legend(['s[n]', 'y2[n]'], loc=3)
plt.show()
```



Com base nos gráficos a saída que mais se aparenta com  $s[n]$  é a saída  $y1[n]$  (distância entre os pontos de  $y[n]$  e  $s[n]$  para um mesmo  $n$  é menor para  $y[n] = y1[n]$ ).

h)

Uma possível medida seria o MSE (Mean Squared Error), pois considera os erros para mais ou para menos com a mesma medida.

$$E = \frac{1}{N} \sum_{i=0}^N (s_i - y_i)^2 \quad (23)$$

```
mse = lambda A, B: sum([(e-f)**2 for e, f in zip(A,B)])/len(A)
print("MSE de y1: ", mse(s,y1))
```

```
## MSE de y1: 0.000927734375
```

```
print("MSE de y2: ", mse(s,y2))
```

```
## MSE de y2: 1.2956000000000003
```

Com base nos valores para o MSE, podemos ver que a saída  $y2[n]$  é muito mais próxima da entrada  $s[n]$  do que a saída  $y1[n]$  (erro entre a saída e o sinal  $s$  é menor para  $y2$ ).

i)

O filtro  $w1$  é mais adequado na tarefa de equalização. Tanto pelos gráficos, quanto pelo resultado do MSE, temos que o filtro  $w1$  obtém resultados muito melhores na tarefa de aproximar a saída  $y[n]$  em relação à entrada  $s[n]$ .