intro-convolucao

December 18, 2024

1 Introdução aos filtros lineares

Complemento para leitura da Sec. 3.5 do livro texto.

1.1 Convolução 1D e 2D

Os filtros lineares empregam uma operação baseada em uma vizinhança, isto é, em um conjunto de pixels ao redor de um pixel central. Esta operação, em alguns casos, pode ser realizada pela operação da convolução, que é largamente utilizada em processamento de sinais. Por esta razão, este material cobre brevemente os fundamentos da convolução e sua aplicação em processamento de imagens.

1.1.1 Problema modelo - suavização de curvas

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

Importação de dados de Covid-19

[4]:
```
url = "https://raw.githubusercontent.com/datasets/covid-19/main/data/
 ↪countries-aggregated.csv"
import io
import requests
s=requests.get(url).content
df=pd.read_csv(io.StringIO(s.decode('utf-8')))
```

Exploração da base de dados

[5]:
```
print (df.head())
```

```
         Date      Country  Confirmed  Recovered  Deaths
0  2020-01-22  Afghanistan          0          0       0
1  2020-01-23  Afghanistan          0          0       0
2  2020-01-24  Afghanistan          0          0       0
3  2020-01-25  Afghanistan          0          0       0
4  2020-01-26  Afghanistan          0          0       0
```

[6]:
```
print (df.tail())
```

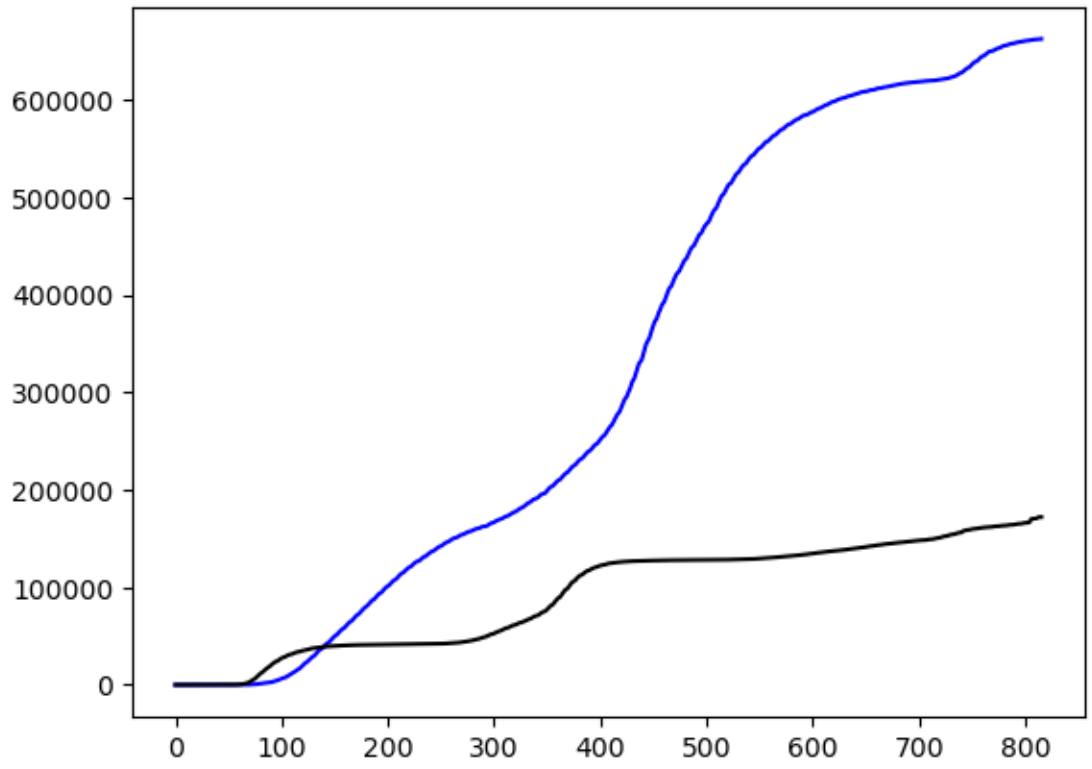
```
             Date    Country  Confirmed  Recovered  Deaths
161563 2022-04-12  Zimbabwe     247094          0    5460
161564 2022-04-13  Zimbabwe     247160          0    5460
161565 2022-04-14  Zimbabwe     247208          0    5462
161566 2022-04-15  Zimbabwe     247237          0    5462
161567 2022-04-16  Zimbabwe     247237          0    5462
```

Dados de mortes - Brasil e Reino Unido

[7]:
```
brasil = np.array(df[df['Country']=='Brazil']['Deaths'])
uk = np.array(df[df['Country']=='United Kingdom']['Deaths'])
df[df['Country']=='Brazil'].to_csv('brasil.csv')
df[df['Country']=='United Kingdom'].to_csv('uk.csv')
```

[8]:
```
plt.plot(range(len(brasil)),brasil,'-b')
plt.plot(range(len(uk)),uk,'-k')
```

[8]: [<matplotlib.lines.Line2D at 0x742ce80cf220>]

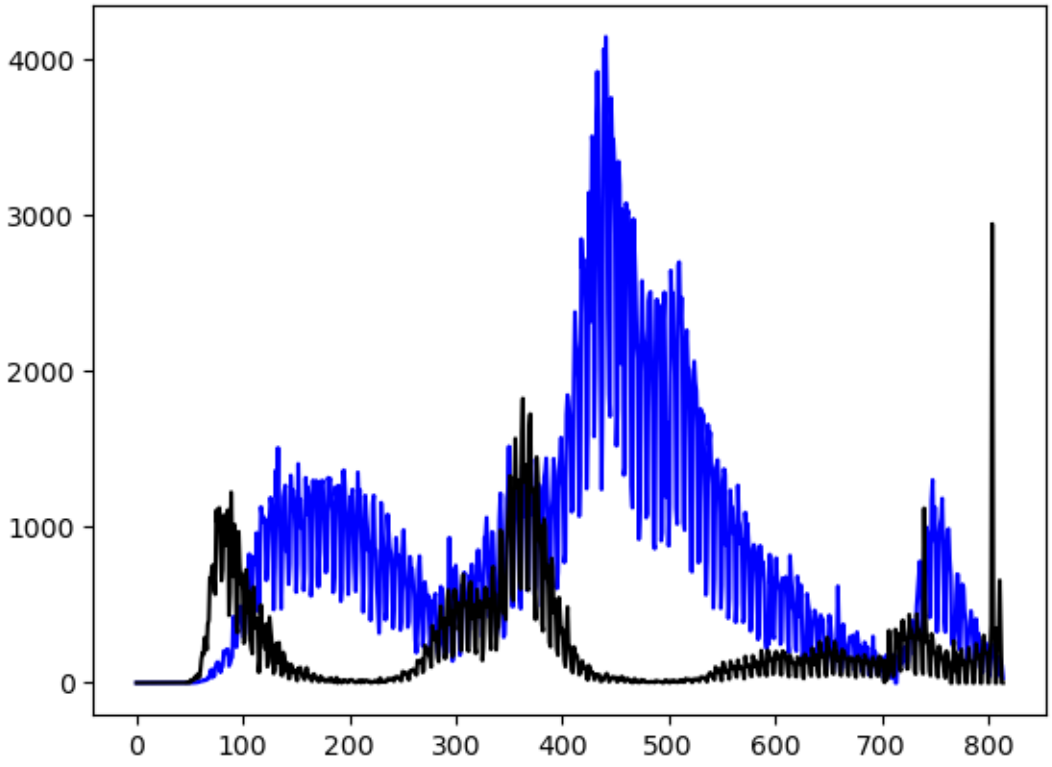


Obtenção dos dados diários

[9]:
```
brasil = brasil[1:]-brasil[:-1]
uk = uk[1:]-uk[:-1]
```

[10]:
```
plt.plot(range(len(brasil)),brasil,'-b')
plt.plot(range(len(uk)),uk,'-k')
```

[10]: [<matplotlib.lines.Line2D at 0x742cdfeacc40>]

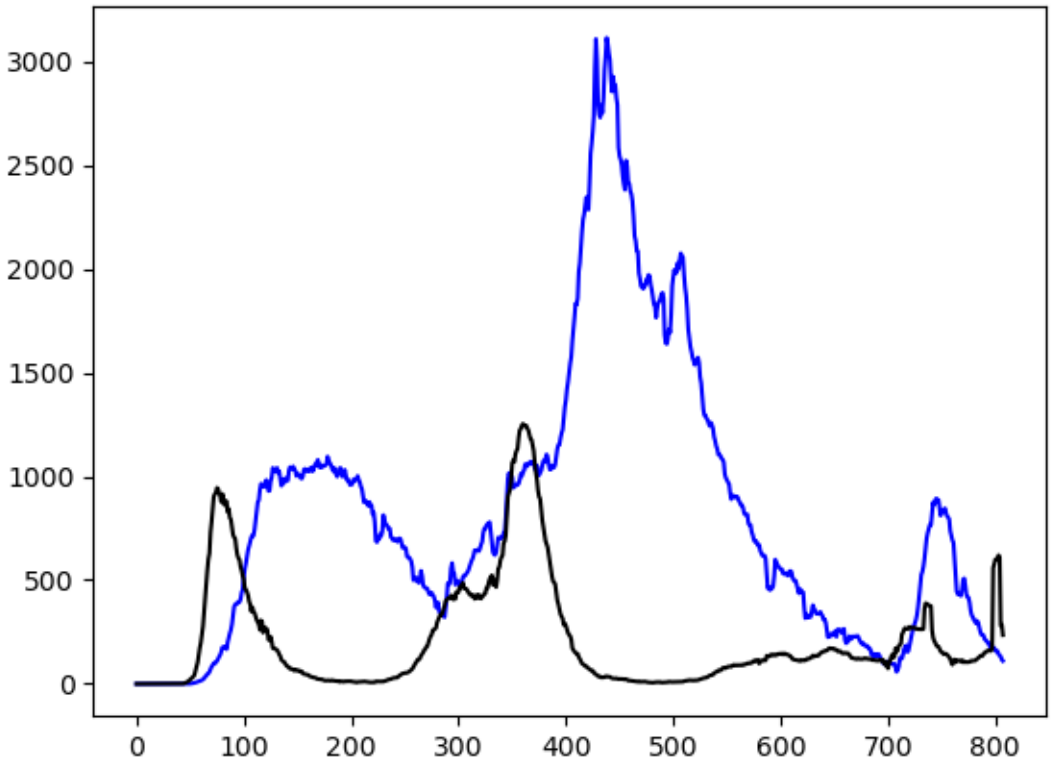


Aplicação da média móvel (janela de tamanho 7 - uma semana)

[11]:
```
brasil_m7 = np.array ([np.mean(brasil[i:(i+7)]) for i in range (len(brasil)-7)])
uk_m7 = np.array ([np.mean(uk[i:(i+7)]) for i in range (len(uk)-7)])
```

[12]:
```
plt.plot(range(len(brasil_m7)),brasil_m7,'-b')
plt.plot(range(len(uk_m7)),uk_m7,'-k')
```

[12]: [<matplotlib.lines.Line2D at 0x742cdb5a7280>]



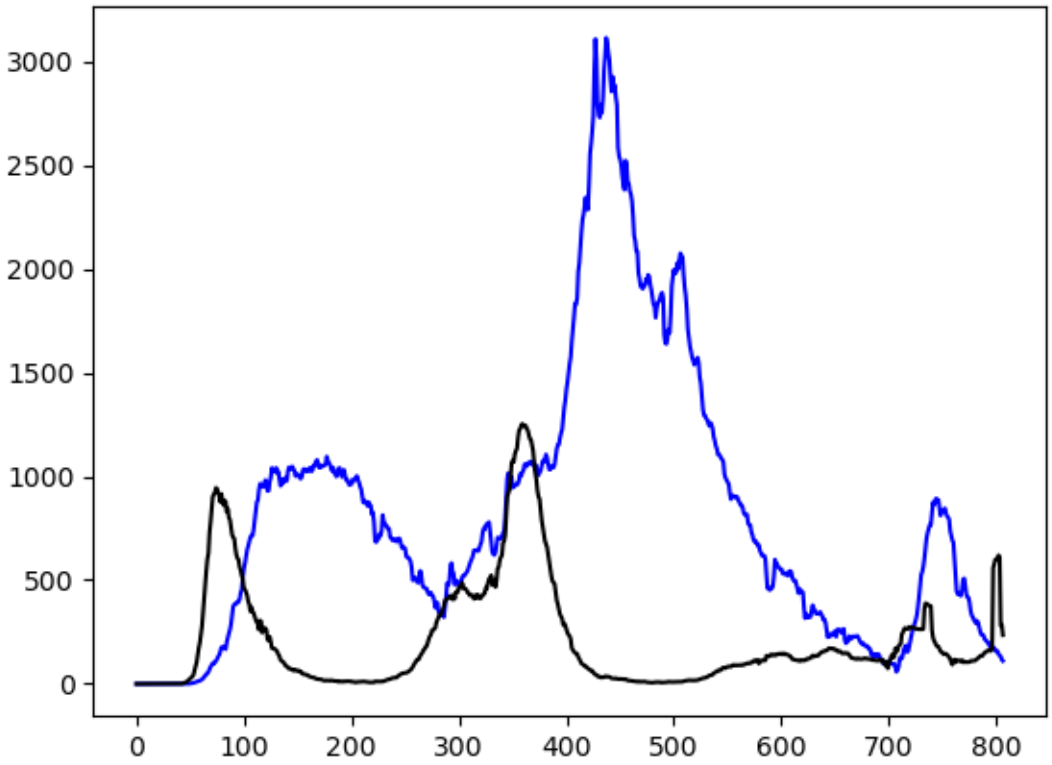
1.1.2 Emprego da convolução

[13]:
```
from scipy import signal
width = 7
msk = np.ones(width) / width
```

[14]:
```
brasil_mw = signal.convolve (brasil,msk)
uk_mw = signal.convolve (uk,msk)
```

[15]:
```
plt.plot(range(len(brasil_mw[width:-width])),brasil_mw[width:-width],'-b')
plt.plot(range(len(uk_mw[width:-width])),uk_mw[width:-width],'-k')
```

[15]:
```
[<matplotlib.lines.Line2D at 0x742cd8725730>]
```



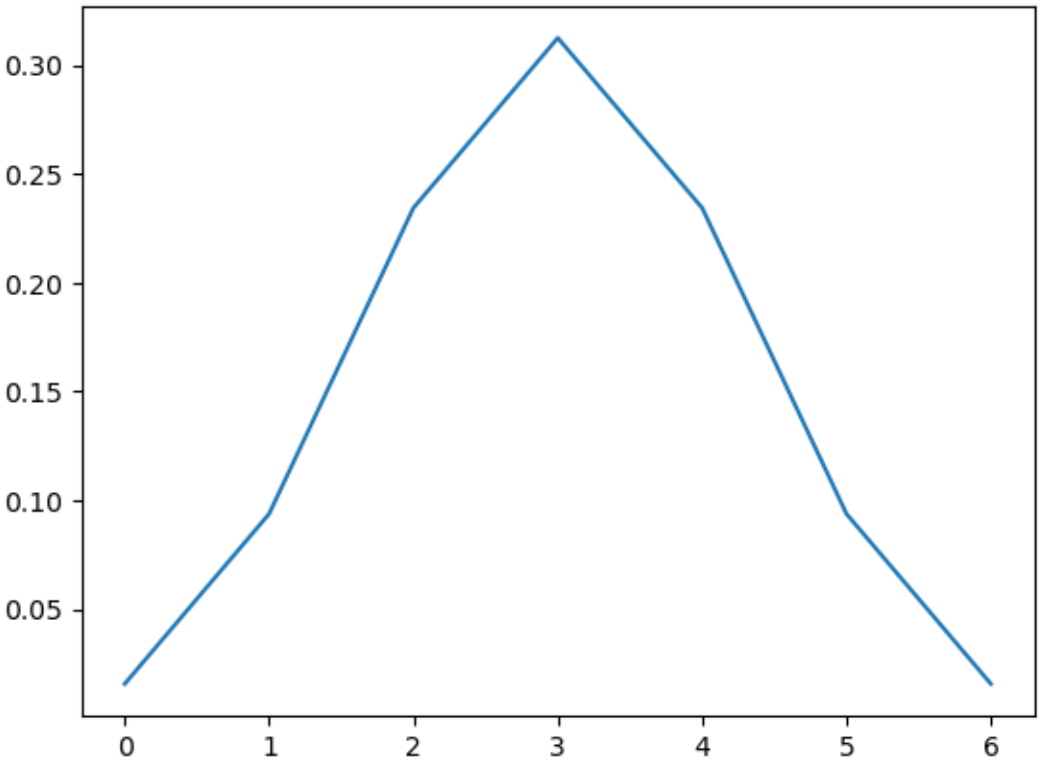
1.1.3 Novo filtro para as séries temporais

[16]:
```
from scipy.special import binom
width = 6
msk = np.array ([binom (width,k) for k in range (width+1)])/2**width
```

$$p(x;n,p) = \left(\begin{array}{c} n \\ p \end{array} \right) x^p (n-x)^q$$

[17]:
```
plt.plot(msk)
```

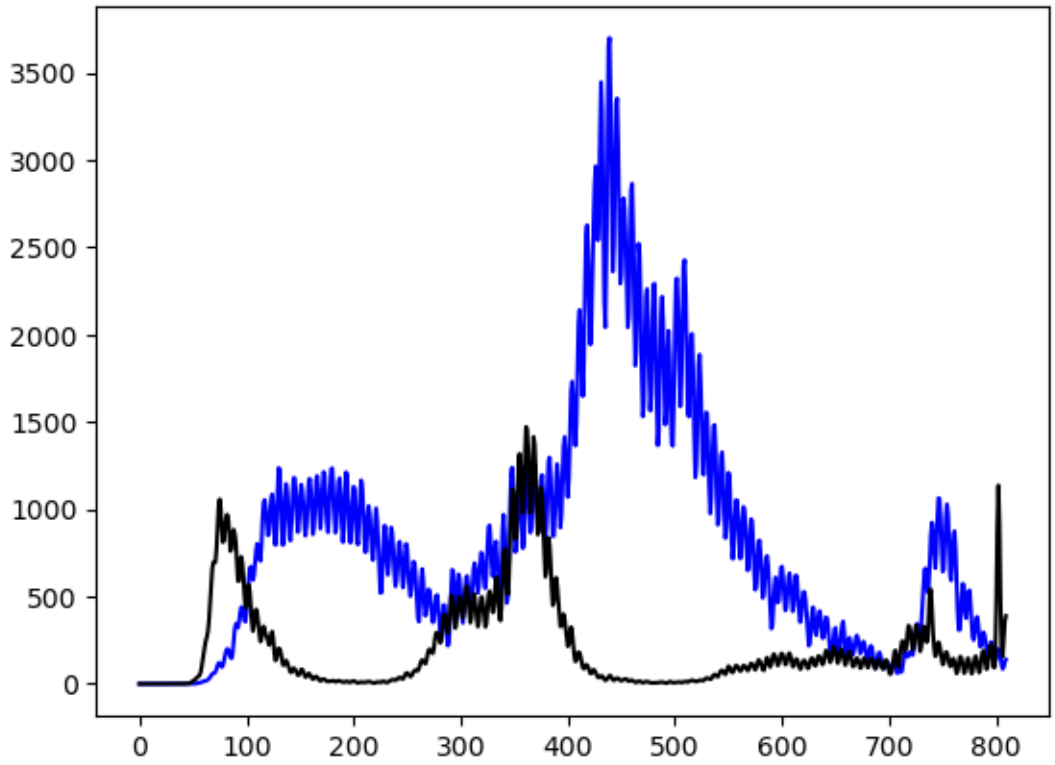
[17]: [<matplotlib.lines.Line2D at 0x742cd870e880>]



[18]:
```
brasil_mw = signal.convolve (brasil,msk)
uk_mw = signal.convolve (uk,msk)
```

[19]:
```
plt.plot(range(len(brasil_mw[width-1:-width])),brasil_mw[width-1:-width],'-b')
plt.plot(range(len(uk_mw[width-1:-width])),uk_mw[width-1:-width],'-k')
```

[19]:
```
[<matplotlib.lines.Line2D at 0x742cd86797f0>]
```



Operação da Convolução

Definição Sejam $x(t)$ e $h(t)$ dois sinais contínuos no tempo, a convolução destes sinais, representada como $x(t) * h(t)$, corresponde à operação:

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

Caso discreto Para sinais discretos, $x(n)$ e $h(n)$, a convolução entre eles é definida como:

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

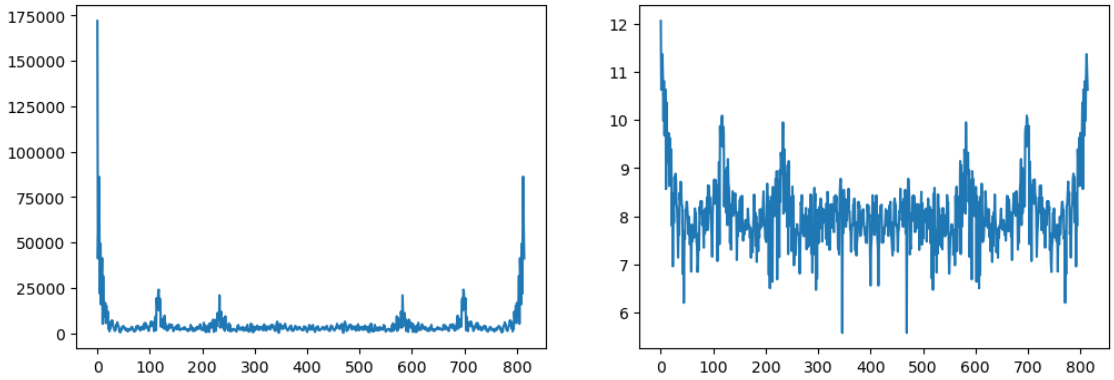
Convolução e Transformada de Fourier (Teorema da Convolução) Sejam $\mathcal{F}$ e $\mathcal{F}^{-1}$ a representação da transformada de Fourier e de sua inversa, respectivamente. O **Teorema da Convolução**, relação fundamental para modelagem de filtros para processamento de sinais, estabelece que:

$$y(t) = x(t) * h(t) = \mathcal{F}^{-1}\{\mathcal{F}\{x(t)\}\mathcal{F}\{h(t)\}\}$$

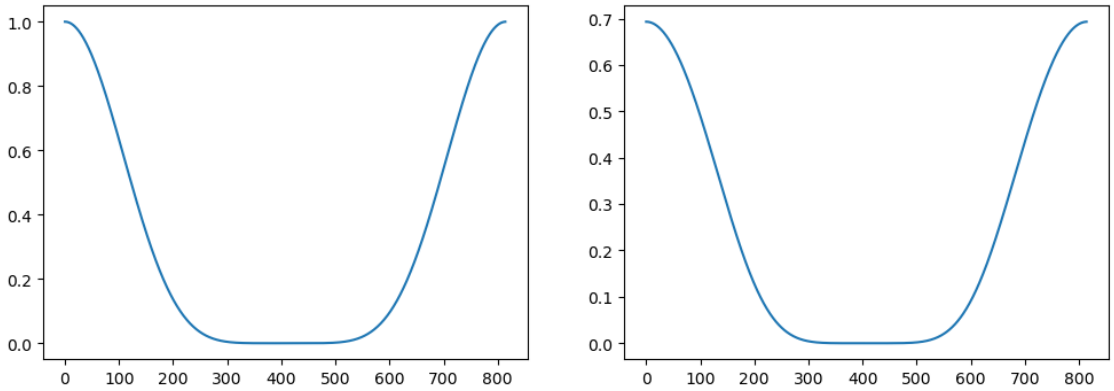
Exemplo - Coronavírus *dataset*

[20]:
```
from scipy.fftpack import fft,ifft
x = uk
xf = fft(x)
```

```
_,ax = plt.subplots (1,2,figsize=(12,4))
ax[0].plot(np.abs(xf))
ax[1].plot(np.log(np.abs(xf)+1))
plt.savefig('xf.png')
```

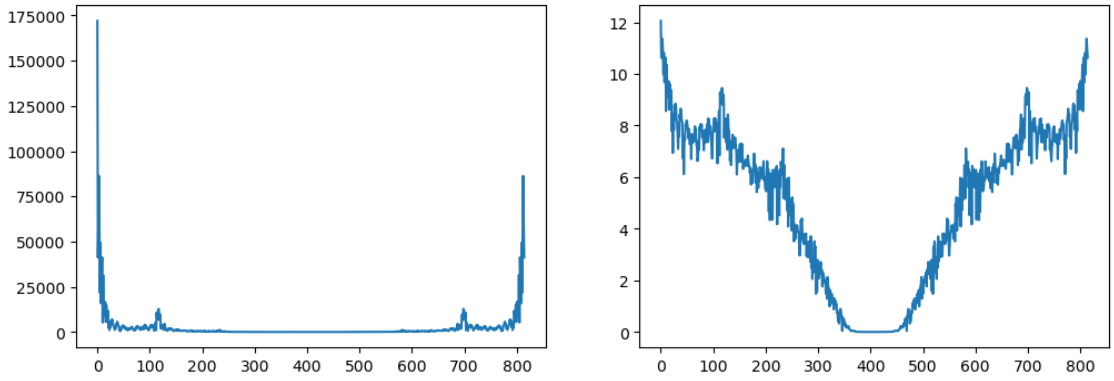


[21]:
```
h = np.zeros(len(uk))
h[:np.ceil(len(msk)/2).astype('int')] = msk[:np.ceil(len(msk)/2).
  ↪astype('int')][::-1]
h[-np.floor(len(msk)/2).astype('int'):] = msk[np.ceil(len(msk)/2).astype('int'):
  ↪][::-1]
hf = fft(h)
_,ax = plt.subplots (1,2,figsize=(12,4))
ax[0].plot(np.abs(hf))
ax[1].plot(np.log(np.abs(hf)+1))
plt.savefig('hf.png')
```

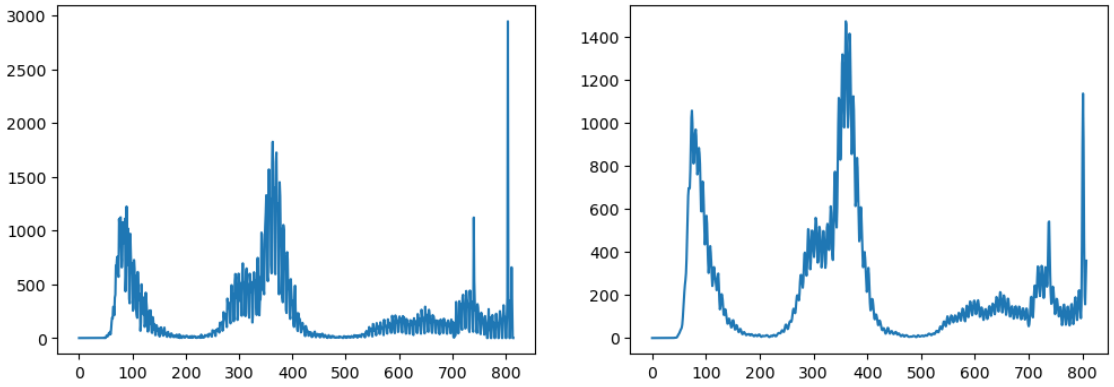


[22]:
```
yf = hf*xf
_,ax = plt.subplots (1,2,figsize=(12,4))
ax[0].plot(np.abs(yf))
```

```
ax[1].plot(np.log(np.abs(yf)+1))
plt.savefig('yf.png')
```



[23]:
```
y = np.real(ifft(yf))
_,ax = plt.subplots (1,2,figsize=(12,4))
ax[0].plot(uk)
ax[1].plot(y[len(msk)//2:-len(msk)//2])
plt.savefig('y.png')
```



1.1.4 Convolução 2D - Processamento de Imagens

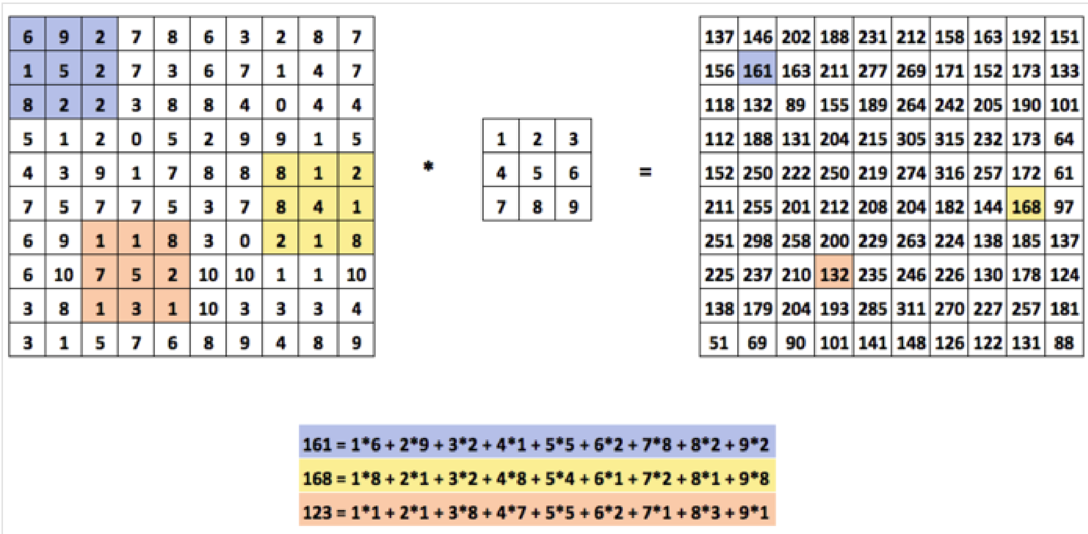
Dadas duas imagens $G_{M\times N}$ e $H_{m\times n}$, a convolução de G e H, é definida como

$$G(x,y) = F(x,y) * H(x,y) = \sum_{i=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} F(x-i, y-j)H(i,j)$$

Observação A convolução produz uma imagem resultante com resolução espacial menor tendo em vista que, nas regiões de borda, parte do filtro extrapola a cobertura da imagem. No exemplo abaixo é empregada a técnica *zero padding*, que aumenta a dimensão da entrada preenchendo as novas linhas e colunas com zeros, antes de aplicar a convolução, para que a imagem resultante tenha a mesma dimensão da original.

[24]:
```python
# https://elo7.dev/convolucao/
from IPython.display import Image
Image('convolucao.png',width=640,height=480)
```

[24]:



Exemplos

Filtro Box

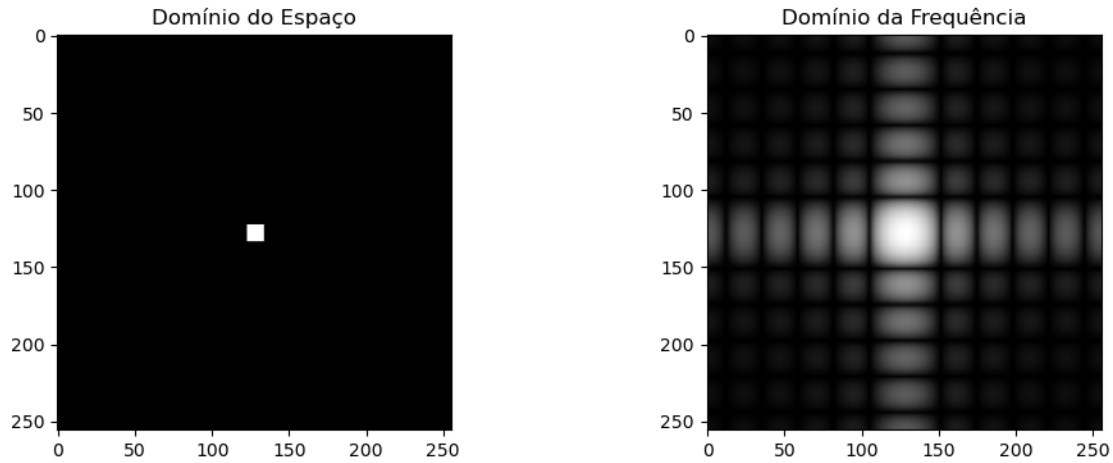
[25]:
```python
N = 11
box = np.ones((N,N))/N*2
print (box)
```

```
[[0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
  0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
 [0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
```

```
 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
[0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]
[0.18181818 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818
 0.18181818 0.18181818 0.18181818 0.18181818 0.18181818]]
```

[26]:
```
from scipy.fftpack import fft2
H=np.zeros((256,256))
H[128-N//2:128+N//2+1,128-N//2:128+N//2+1]=box
_,ax = plt.subplots (1,2,figsize=(12,4))
plt.subplot(121)
plt.imshow(H,cmap='gray')
plt.title ('Domínio do Espaço')
aux = np.array([1,-1]*128).reshape(256,1)
aux = aux.dot(aux.T)
box_f = fft2 (H*aux)
plt.subplot(122)
plt.title ('Domínio da Frequência')
plt.imshow(np.log(np.abs(box_f)+1),cmap='gray')
```

[26]:
```
<matplotlib.image.AxesImage at 0x742cd83495e0>
```



[27]:
```
# http://www.cse.psu.edu/~rtc12/CSE486/lecture04.pdf
from IPython.display import Image
Image('box-filter.png',width=640,height=480)
```

[27]:

Detecção de bordas

Filtro Laplaciano

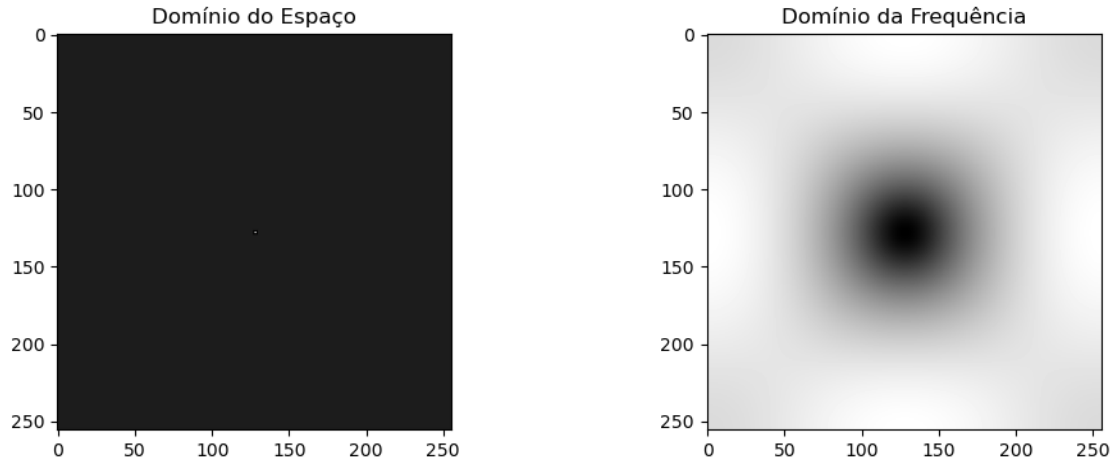
[28]:
```
laplace = np.array([[-1,-1,-1],
                    [-1,8,-1],
                    [-1,-1,-1]])
print (laplace)
```

```
[[-1 -1 -1]
 [-1  8 -1]
 [-1 -1 -1]]
```

[29]:
```
H=np.zeros((256,256))
N = 3
H[128-N//2:128+N//2+1,128-N//2:128+N//2+1]=laplace
_,ax = plt.subplots (1,2,figsize=(12,4))
plt.subplot(121)
plt.imshow(H,cmap='gray')
plt.title ('Domínio do Espaço')
aux = np.array([1,-1]*128).reshape(256,1)
```

```
aux = aux.dot(aux.T)
box_f = fft2 (H*aux)
plt.subplot(122)
plt.title ('Domínio da Frequência')
plt.imshow(np.log(np.abs(box_f)+1),cmap='gray')
```

[29]: <matplotlib.image.AxesImage at 0x742cd82fe2b0>



[30]:
```
# https://elo7.dev/convolucao/
Image('passa-altas.png',width=640,height=480)
```

[30]: