hw07

November 8, 2021

1 Homework 07

Nathan LeRoy

[71]:
```
# better image quality
import matplotlib as mpl
%matplotlib inline
mpl.rcParams['figure.dpi'] = 300
```

1.1 Problem 1

Given that our system is an LSI system, we know that the impulse reponse is equivalent to substituting in an impulse ($\delta[n]$) for $x[n]$:

$$h[n] = \frac{1}{M_1 + M_2 + 1} \sum_{k=-M_1}^{M_2} \delta[n-k]$$

Which is equivalent to:

$$\frac{1}{M_1 + M_2 + 1} \Big(\delta[n + M_1] + \cdots + \delta[n - M_2] \Big)$$

Which we know, due to the definition of the impulse function will always be equal to **1** outside the interval $-M_1 \leq n \leq M_2$. Formally,

$$h[n] = \begin{cases} \frac{1}{M_1 + M_2 + 1} & -M_1 \leq n \leq M_2 \\ 0 & \text{otherwise} \end{cases}$$

1.2 Problem 2

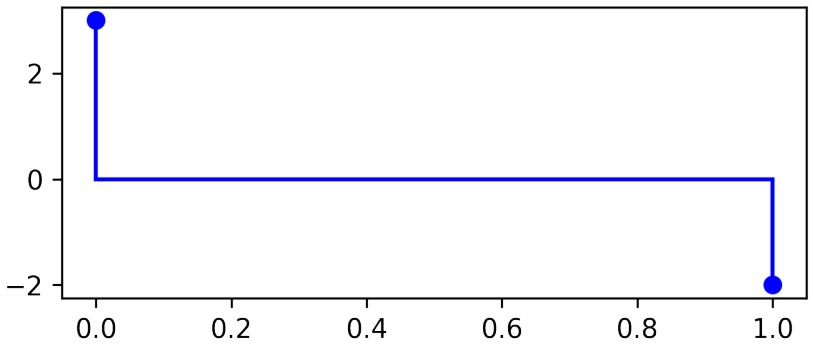
1.2.1 a.) $x[n] = 3\delta[n] - 2\delta[n-1]$

We can plot the function $x[n]$:

[72]:
```
import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (5,2)
```

```
x = [3, -2]

plt.stem(
    x,
    linefmt='b-',
    markerfmt="bo",
    basefmt="b-",
    label="$x[n]$"
)
```

[72]: <StemContainer object of 3 artists>



We know for an LSI system:

$$y[n] = \sum_{k=0}^{\infty} x[n]h[n-k]$$

Given our function $x[n]$, we know that $y[n]$ will be equal to 9 outside the interval $0 \leq n \leq 1$. Thus we need only consider $k = 0$ and $k = 1$:

$$y[n] = x[0]h[0] + x[1]h[n-1]$$
$$y[n] = 3h[n] - 2h[n-1]$$

And given the plot of $h[n]$, we can write out the analytical function for $h[n]$:

$$h[n] = \delta[n+1] + 3\delta[n] + 2\delta[n-1] - \delta[n-2] + \delta[n-3]$$

Thus,

$$y[n] = 3\Big(\delta[n+1]+3\delta[n]+2\delta[n-1]-\delta[n-2]+\delta[n-3]\Big) - 2\Big(\delta[n]+3\delta[n-1]+2\delta[n-2]-\delta[n-3]+\delta[n-4]\Big)$$

$$y[n] = 3\delta[n+1]+9\delta[n]+6\delta[n-1]-3\delta[n-2]+3\delta[n-3]-2\delta[n]-6\delta[n-1]-4\delta[n-2]+2\delta[n-3]-2\delta[n-4]$$

Simplifying, we obtain:

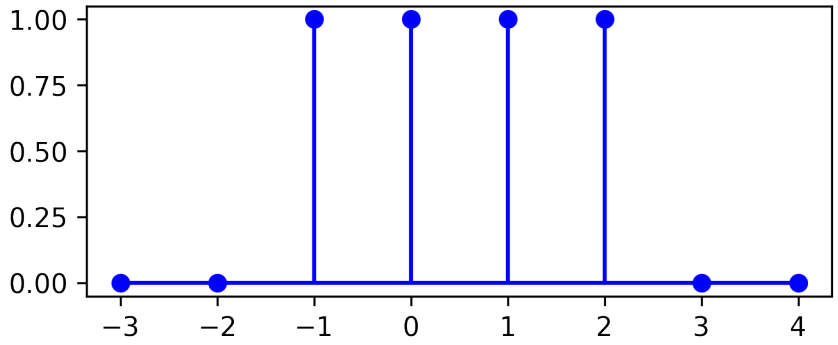
$$y[n] = 3\delta[n+1] + 7\delta[n] - 7\delta[n-2] + 5\delta[n-3] - 2\delta[n-4]$$

1.2.2 b.) $x[n] = u[n+1] - u[n-3]$

Given the definitions of step functions, we know that this system will be equal to 0 outside the interval $-1 \leq n \leq 2$. (note: $u[3+1] - u[3-3] = 1 - 1 = 0$). we can plot this function in python:

[73]:
```
locs = [-3, -2, -1, 0, 1, 2, 3, 4]
x = [0, 0, 1, 1, 1, 1, 0, 0]
plt.stem(
    locs,
    x,
    linefmt='b-',
    markerfmt="bo",
    basefmt="b-",
    label="$x[n]$"
)
```

[73]: <StemContainer object of 3 artists>



Again, given an LSI system, we can use the following definition to calculate the output $y[n]$:

$$y[n] = \sum_{k=0}^{\infty} x[n]h[n-k]$$

Which will be equal to 0 outside the interval $-1 \leq n \leq 2$ by looking at the above graph. As well, knowing $x[k]$ will always equal 1 inside that interval, our system can be described as:

$$y[n] = h[n+1] + h[n] + h[n-1] + h[n-2]$$

Using the above analytic definition for $h[n]$:

$$\begin{aligned} y[n] = &\Big(\delta[n+2] + 3\delta[n+1] + 2\delta[n] - \delta[n-1] + \delta[n-2]\Big) \\ &+\Big(\delta[n+1] + 3\delta[n] + 2\delta[n-1] - \delta[n-2] + \delta[n-3]\Big) \\ &+\Big(\delta[n] + 3\delta[n-1] + 2\delta[n-2] - \delta[n-3] + \delta[n-4]\Big) \\ &+\Big(\delta[n-1] + 3\delta[n-2] + 2\delta[n-3] - \delta[n-4] + \delta[n-5]\Big) \end{aligned}$$

Simplifying, we obtain:

$$y[n] = \delta[n+2] + 4\delta[n+1] + 6\delta[n] + 5\delta[n-1] + 5\delta[n-2] + 2\delta[n-3] + \delta[n-5]$$

1.2.3 c.) x[n] in figure.

We can extract $x[n]$ from the figure as such:

$$x[n] = -\delta[n+2] + 2\delta[n] + 2\delta[n-3]$$

Again, lets start at the begining. A very good place to start…

$$y[n] = \sum_{k=0}^{\infty} x[n]h[n-k]$$

We know $x[k]$ to be zero outside the domain $k = \{-2, 0, 3\}$, thus we can construct the following response:

$$y[n] = x[-2]h[n+2] + x[0]h[n] + x[3]h[n-3]$$

$$y[n] = -h[n+2] + 2h[n] + 2h[n-3]$$

Thus,

$$y[n] = -\Big(\delta[n+3] + 3\delta[n+2] + 2\delta[n+1] - \delta[n] + \delta[n-1]\Big)$$

$$+2\Big(\delta[n+1]+3\delta[n]+2\delta[n-1]-\delta[n-2]+\delta[n-3]\Big)+$$
$$2\Big(\delta[n-2]+3\delta[n-3]+2\delta[n-4]-\delta[n-5]+\delta[n-6]\Big)$$

$$\begin{aligned}y[n] &= -\delta[n+3]-3\delta[n+2]-2\delta[n+1]+\delta[n]-\delta[n-1]\\ &+2\delta[n+1]+6\delta[n]+4\delta[n-1]-2\delta[n-2]+2\delta[n-3]\\ &+2\delta[n-2]+6\delta[n-3]+4\delta[n-4]-2\delta[n-5]+2\delta[n-6]\end{aligned}$$

Simplifying, we obtain:

$$y[n] = -\delta[n+3]-3\delta[n+2]+7\delta[n]+3\delta[n-1]+8\delta[n-3]+4\delta[n-4]-2\delta[n-5]+2\delta[n-6]$$

1.3 Problem 3

1.3.1 a.)

The base equation can be written as:

$$y(t) = x(t) * \Big(\text{sub system}\Big) * h_5(t)$$

Where the sub system can be written as:

$$h_1(t) - (h_2(t) + h_3(t)) * h_4(t)$$

Which gives us:

$$y(t) = x(t) * \Big(h_1(t) - h_4(t) * (h_2(t) + h_3(t))\Big) * h_5(t)$$

1.3.2 b.)

The base equation can be written as:

$$y[n] = x[n] * h_1[n] * \Big(\text{sub system}\Big) * h_6[n]$$

Where the sub system can be written as:

$$h_3[n] * h_5[n] - h_2[n] * h_4[n]$$

Which gives us:

$$y[n] = x[n] * h_1[n] * \Big[h_3[n] * h_5[n] - h_2[n] * h_4[n] \Big] * h_6[n]$$

1.3.3 c.)

The base equation can be written as:

$$y(t) = x(t) * \Big(\text{sub system} \Big)$$

Where the sub-system can be written as:

$$(-h_1(t) + h_2(t)) * h_3(t) * h_4(t) + h_2(t)$$

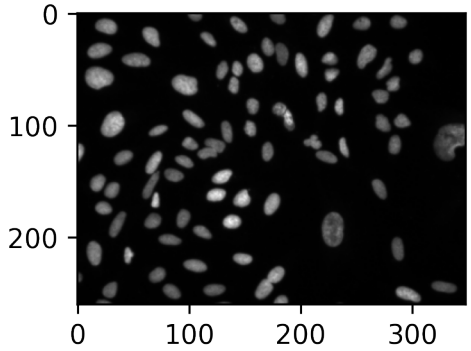
Which gives us:

$$y(t) = x(t) * \Big((h_2(t) - h_1(t)) * h_3(t) * h_4(t) + h_2(t) \Big)$$

1.4 Problem 2

We can read the input image like so:

[74]:
```
import numpy as np

im1 = plt.imread("data/cell_img.png")
im1 = np.dot(im1[..., :3], [0.299, 0.587, 0.114]) # select grayscale part of␣
↪image
plt.imshow(im1, cmap="gray") # display the image
plt.show()
```



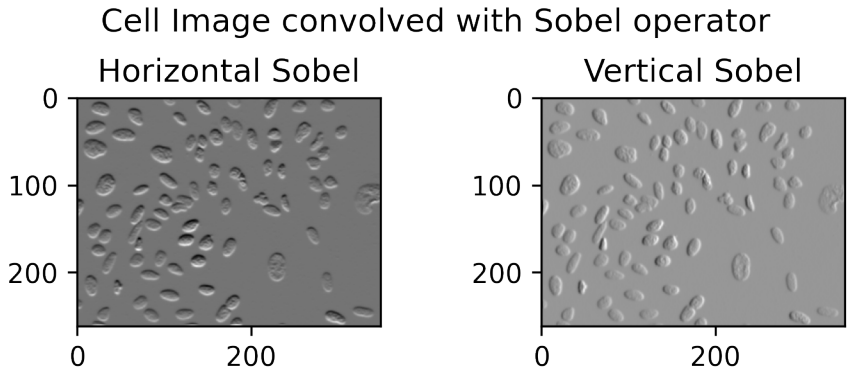
We can define both the horizontal and vertical Sobel operators like so:

[75]:
```
h_horizontal = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]]) # horizontal␣
↪Sobel operator
h_vertical = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]) # vertical Sobel␣
↪operator
```

We can now convolve the cell image using the `scipy.signal.convolve2d()` function.

[76]:
```
from scipy.signal import convolve2d

im1_hsobel = convolve2d(im1, h_horizontal)
im1_vsobel = convolve2d(im1, h_vertical)

fig, ax = plt.subplots(1,2)
fig.set_tight_layout({"pad": 0.5})
fig.suptitle("Cell Image convolved with Sobel operator")

ax[0].imshow(im1_hsobel, cmap="gray")
ax[0].set_title("Horizontal Sobel")

ax[1].imshow(im1_vsobel, cmap="gray")
ax[1].set_title("Vertical Sobel")
```

[76]:
```
Text(0.5, 1.0, 'Vertical Sobel')
```

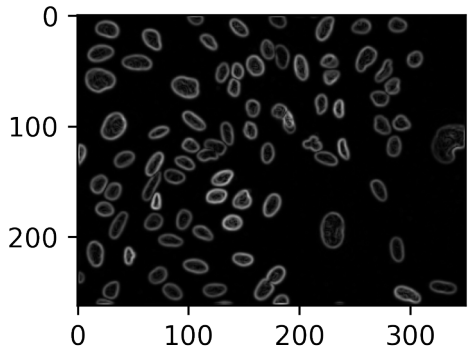


We can compute the final image using the pixelwise square root of the sum of the squares of the two processed images:

$$F_{i,j} = \sum_i \sum_j \sqrt{I_{h_{i,j}}^2 + I_{v_{i,j}}^2}$$

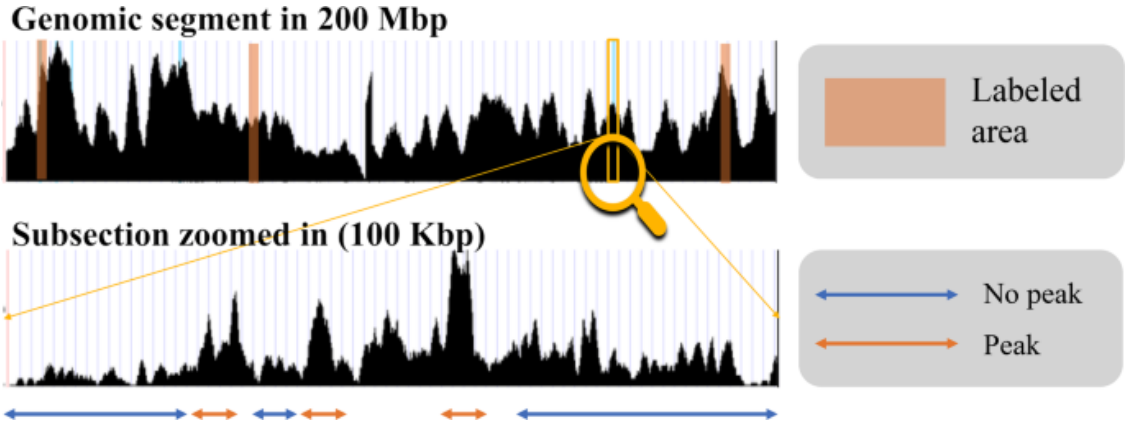
[77]:
```
final_im = np.sqrt(im1_hsobel**2 + im1_vsobel**2)

plt.imshow(final_im, cmap="gray")
```

[77]: <matplotlib.image.AxesImage at 0x1337193d0>



1.5 Problem 5

In the study og epigenomics, peak calling is a computational method used to identify areas in a genome that have been enriched with aligned reads as a consequence of performing a ChIP-sequencing or MeDIP-seq experiment. Given a genomic signal track of aligned reads, a scientist can find "peaks" of readily aligned sequences that might indicate an area of the genome that is particularly interesting to whatever experiment is being conducted (Typically this is interpreted as how *open* chromatin are).



Often, the genomic signal can come in quite noisey and hard to analyze. This is especially true among **single-cell** experiements, where sample read sizes can tend to be quite small and thus, our data is much more prone to noise and random fluctuations. Filters using mathematical convolutions can be applied to *smooth* outthese signal tracks and make peak calling easier for a researcher or software package. One such simple example is the **moving-average** filter:

$$\bar{x_i} = \frac{1}{M+1} \sum_{j=-M}^{M} x[i+j]$$

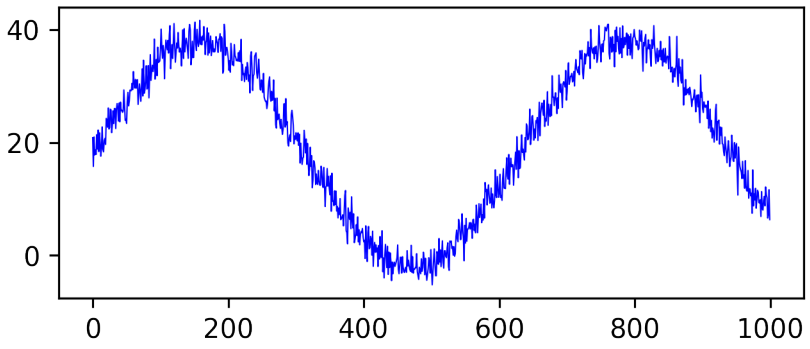
We can achieve this result across our entire dataset by convoluting our signal track using a "window of ones". To demonstrate this I'm going to generate a simulated sinusoidal signal track that we will smooth using our moving average convolution:

$$x[t]_{\text{smooth}} = \sum_{k=0}^{N} x[t][\frac{1}{W}, \ldots, \frac{1}{W}]$$

Where the length of the ones vector is the size of the window, W.

[78]:
```
x = np.array([i for i in range(1000)])

# simulated singal function
def f_x(x: np.array) -> np.array:
    A = 20 # amplitude
    return 20*np.sin(0.01*x) + 20 + np.random.normal(-A/10, A/10, len(x))

noisy = f_x(x)

# plot the result of
# our noist function
plt.plot(
    x,
    noisy,
    "b-",
    linewidth=0.5
)
```

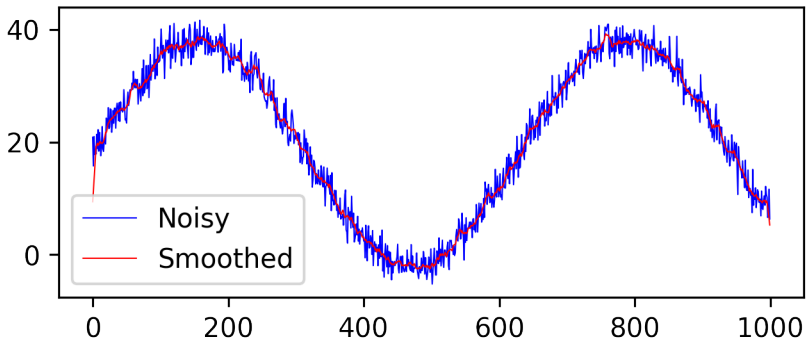
[78]:
```
[<matplotlib.lines.Line2D at 0x1337732b0>]
```



[79]:
```
from numpy import convolve

# run a convolution against our dataset usinga moving average
WINDOW = 10 # window size
```

```
# run smoothing convolution
smoothed = convolve(
    noisy,
    np.ones(WINDOW)/WINDOW,
    "same"
)

# plot noisy
plt.plot(
    x,
    noisy,
    "b-",
    linewidth=0.5,
    label="Noisy"
)

plt.plot(
    x,
    smoothed,
    "r-",
    linewidth=0.5,
    label="Smoothed"
)

plt.legend()
```

[79]: <matplotlib.legend.Legend at 0x1333fc610>



We can see the smoothed signal nicely overlayed on the noisy signal which gives as a better look at the signal track and facilitates nice analysis of the data.