

# EECS16A Homework 12

## Question 3: Mechanical Correlation

### Part (c)

```
In [2]: import numpy as np
s1 = [2, -2, 2, -2]
s2 = [1, 2, 3, 4]

# Use the function np.correlate with mode='full' for linear cross correlation
print(np.correlate(s1, s2, mode="full"))
print(np.correlate(s2, s1, mode="full"))    # they are flipped horizontally

[ 8 -2  6 -4 -4 -2 -2]
[-2 -2 -4 -4  6 -2  8]
```

## Question 4: Audio File Matching

This notebook continues the audio file matching problem. Be sure to have song.wav and clip.wav in the same directory as the notebook.

In this notebook, we will look at the problem of searching for a small audio clip inside a song.

The song "Mandelbrot Set" by Jonathan Coulton is licensed under [CC BY-NC 3.0](http://creativecommons.org/licenses/by-nc/3.0/) (<http://creativecommons.org/licenses/by-nc/3.0/>).

If you have trouble playing the audio file in IPython, try opening it in a different browser. I encountered problem with Safari but Chrome works for me.

```
In [3]: # Part B
import numpy as np
print(np.correlate([-1, 1, 1, -1, 1, 1, -1, 1], [1, 1, -1], mode="full"))
print(np.correlate([-1, 1, 1, -1, 1, 1, -1, 1], [1, 1, 1], mode="full"))

[ 1 -2 -1  3 -1 -1  3 -1  0  1]
[-1  0  1  1  1  1  1  1  0  1]
```

```
In [4]: import numpy as np
import wave
import matplotlib.pyplot as plt
import scipy.io.wavfile
import operator
from IPython.display import Audio
%matplotlib inline

given_file = 'song.wav'
target_file = 'clip.wav'
rate_given, given_signal = scipy.io.wavfile.read(given_file)
rate_target, target_signal = scipy.io.wavfile.read(target_file)
given_signal = given_signal[:2000000].astype(float)
target_signal = target_signal.astype(float)
def play_clip(start, end, signal=given_signal):
    scipy.io.wavfile.write('temp.wav', rate_given, signal[start:end].astype(float))
    return Audio(url='temp.wav', autoplay=True)

def run_comparison(target_signal, given_signal, idxs=None):
    # Run everything if not called with idxs set to something
    if idxs is None:
        idxs = [i for i in range(len(given_signal)-len(target_signal))]
    return idxs, [vector_compare(target_signal, given_signal[i:i+len(target_signal)])
                  for i in idxs]

play_clip(0, len(given_signal))
```

Out [4]: -0:45

We will load the song into the variable `given_signal` and load the short clip into the variable `target_signal`. Your job is to finish code that will identify the short clip's location in the song. The clip we are trying to find will play after executing the following block.

```
In [5]: Audio(url=target_file, autoplay=True)
```

Out [5]: -0:01

Your task is to define the function 'vector\_compare' and run the following code. Because the song has a lot of data, you should use the provided examples from the previous parts of the problem before running the later code. Do your results here make sense given your answers to previous parts of the problem?

```
In [2]: def vector_compare(short_clip, segment_of_song):
        """This function compares two vectors, returning a number.
        The test vector with the highest return value is regarded as being
        return sum([short_clip[i] * segment_of_song[i] for i in range(len(

print("PART A:")
print(vector_compare(np.array([1,1,1]), np.array([1,1,1])))
print(vector_compare(np.array([1,1,1]), np.array([-1,-1,-1])))
print("PART C:")
print(vector_compare(np.array([1,2,3]), np.array([1,2,3])))
print(vector_compare(np.array([1,2,3]), np.array([2,3,4])))
print(vector_compare(np.array([1,2,3]), np.array([3,4,5])))
print(vector_compare(np.array([1,2,3]), np.array([4,5,6])))
print(vector_compare(np.array([1,2,3]), np.array([5,6,7])))
print(vector_compare(np.array([1,2,3]), np.array([6,7,8])))
```

PART A:

3

-3

PART C:

14

20

26

32

38

44

## Part (e)

Run the following code that runs `vector_compare` on every subsequence in the song- it will probably take at least 5 minutes. How do you interpret this plot to find where the clip is in the song?

```
In [ ]: import time

t0 = time.time()
idxs, song_compare = run_comparison(target_signal, given_signal)
t1 = time.time()
plt.plot(idxs, song_compare)
print ("That took %(time).2f minutes to run" % {'time':(t1-t0)/60.0} )
```

## Question 5: GPS Receivers

```
In [6]: %pylab inline
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
import sys
```

Populating the interactive namespace from numpy and matplotlib

```
In [7]: ## RUN THIS FUNCTION BEFORE YOU START THIS PROBLEM
## This function will generate the gold code associated with the satellite
## The satellite_ID can be any integer between 1 and 24
def Gold_code_satellite(satellite_ID):
    codelength = 1023
    registerlength = 10

    # Defining the MLS for G1 generator
    register1 = -1*np.ones(registerlength)
    MLS1 = np.zeros(codelength)
    for i in range(codelength):
        MLS1[i] = register1[9]
        modulo = register1[2]*register1[9]
        register1 = np.roll(register1,1)
        register1[0] = modulo

    # Defining the MLS for G2 generator
    register2 = -1*np.ones(registerlength)
    MLS2 = np.zeros(codelength)
    for j in range(codelength):
        MLS2[j] = register2[9]
        modulo = register2[1]*register2[2]*register2[5]*register2[7]*register2[9]
        register2 = np.roll(register2,1)
        register2[0] = modulo

    delay = np.array([5,6,7,8,17,18,139,140,141,251,252,254,255,256,257,258,259,260,261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290,291,292,293,294,295,296,297,298,299,300,301,302,303,304,305,306,307,308,309,310,311,312,313,314,315,316,317,318,319,320,321,322,323,324,325,326,327,328,329,330,331,332,333,334,335,336,337,338,339,340,341,342,343,344,345,346,347,348,349,350,351,352,353,354,355,356,357,358,359,360,361,362,363,364,365,366,367,368,369,370,371,372,373,374,375,376,377,378,379,380,381,382,383,384,385,386,387,388,389,390,391,392,393,394,395,396,397,398,399,400,401,402,403,404,405,406,407,408,409,410,411,412,413,414,415,416,417,418,419,420,421,422,423,424,425,426,427,428,429,430,431,432,433,434,435,436,437,438,439,440,441,442,443,444,445,446,447,448,449,450,451,452,453,454,455,456,457,458,459,460,461,462,463,464,465,466,467,468,469,470,471,472,473,474,475,476,477,478,479,480,481,482,483,484,485,486,487,488,489,490,491,492,493,494,495,496,497,498,499,500,501,502,503,504,505,506,507,508,509,510,511,512,513,514,515,516,517,518,519,520,521,522,523,524,525,526,527,528,529,530,531,532,533,534,535,536,537,538,539,540,541,542,543,544,545,546,547,548,549,550,551,552,553,554,555,556,557,558,559,560,561,562,563,564,565,566,567,568,569,570,571,572,573,574,575,576,577,578,579,580,581,582,583,584,585,586,587,588,589,590,591,592,593,594,595,596,597,598,599,600,601,602,603,604,605,606,607,608,609,610,611,612,613,614,615,616,617,618,619,620,621,622,623,624,625,626,627,628,629,630,631,632,633,634,635,636,637,638,639,640,641,642,643,644,645,646,647,648,649,650,651,652,653,654,655,656,657,658,659,660,661,662,663,664,665,666,667,668,669,670,671,672,673,674,675,676,677,678,679,680,681,682,683,684,685,686,687,688,689,690,691,692,693,694,695,696,697,698,699,700,701,702,703,704,705,706,707,708,709,710,711,712,713,714,715,716,717,718,719,720,721,722,723,724,725,726,727,728,729,730,731,732,733,734,735,736,737,738,739,740,741,742,743,744,745,746,747,748,749,750,751,752,753,754,755,756,757,758,759,760,761,762,763,764,765,766,767,768,769,770,771,772,773,774,775,776,777,778,779,780,781,782,783,784,785,786,787,788,789,790,791,792,793,794,795,796,797,798,799,800,801,802,803,804,805,806,807,808,809,810,811,812,813,814,815,816,817,818,819,820,821,822,823,824,825,826,827,828,829,830,831,832,833,834,835,836,837,838,839,840,841,842,843,844,845,846,847,848,849,850,851,852,853,854,855,856,857,858,859,860,861,862,863,864,865,866,867,868,869,870,871,872,873,874,875,876,877,878,879,880,881,882,883,884,885,886,887,888,889,890,891,892,893,894,895,896,897,898,899,900,901,902,903,904,905,906,907,908,909,910,911,912,913,914,915,916,917,918,919,920,921,922,923,924,925,926,927,928,929,930,931,932,933,934,935,936,937,938,939,940,941,942,943,944,945,946,947,948,949,950,951,952,953,954,955,956,957,958,959,960,961,962,963,964,965,966,967,968,969,970,971,972,973,974,975,976,977,978,979,980,981,982,983,984,985,986,987,988,989,990,991,992,993,994,995,996,997,998,999,1000,1001,1002,1003,1004,1005,1006,1007,1008,1009,1010,1011,1012,1013,1014,1015,1016,1017,1018,1019,1020,1021,1022,1023])
    G1_out = MLS1;
    shamt = delay[satellite_ID - 1]
    G2_out = np.roll(MLS2,shamt)

    CA_code = G1_out * G2_out

    return CA_code
```

## Part (a)

```
In [8]: def cross_correlation(array1, array2):  
        """ This function should return two arrays or a matrix with one row and one column.  
        The first array is the offset and other to the correlation value. array1 and array2 are  
        arrays of equal length.  
        Think of array1 as the received signal and array2 as the signature.  
        The function should return correlation values as well as the indices.  
        Hint: look up np.correlate  
        """  
  
        #correlated_array = #Your code here (it is just one line) np.correlate(array1, array2, 'full')  
        correlated_array = np.correlate(array1, array2, 'full')  
  
        #Since both the arrays start at 0, the last "shift" where the signals overlap is the negative of the length of array1  
        end_index = len(array1)  
  
        #Similarly, the first "shift" where the signals overlap is the negative of the length of array2  
        st_index = -len(array2) + 1  
  
        indices = np.arange(st_index, end_index)  
        return (indices, correlated_array)
```

```
In [9]: # Plot the auto-correlation of satellite 10 with itself. Fill in the
array_10 = Gold_code_satellite(10)

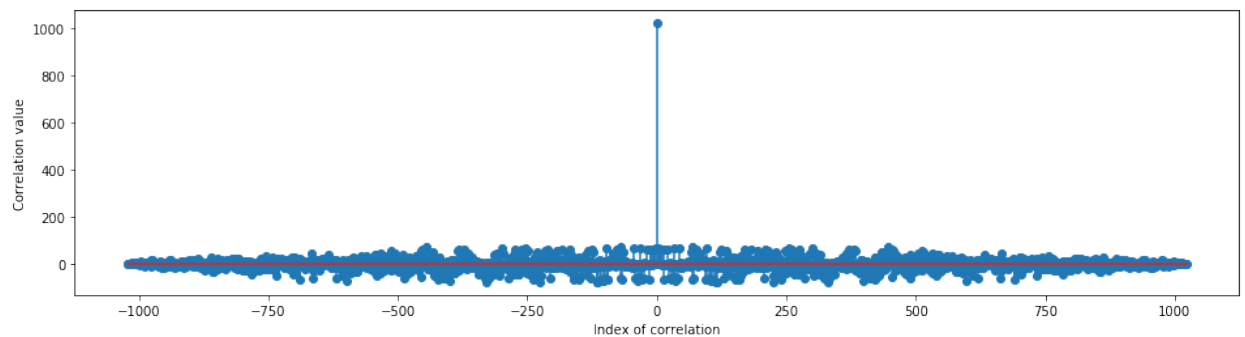
(ind_10, self_10) = cross_correlation(array_10, array_10)

plt.figure(figsize=(16, 4))
plt.stem(ind_10, self_10)
plt.xlabel("Index of correlation")
plt.ylabel("Correlation value")
```

/Users/manlai/miniconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:7: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use\_line\_collection" keyword argument to True.

```
import sys
```

Out[9]: Text(0, 0.5, 'Correlation value')



The autocorrelation peaks at 1023 when the signals are perfectly aligned (offset 0). The correlation of a Gold code with a shifted version of itself is not significant.

## Part (b)

Plot the cross correlation when array1 = satellite 13 and array2 = satellite10

```
In [10]: array_10 = Gold_code_satellite(10)
array_13 = Gold_code_satellite(13)

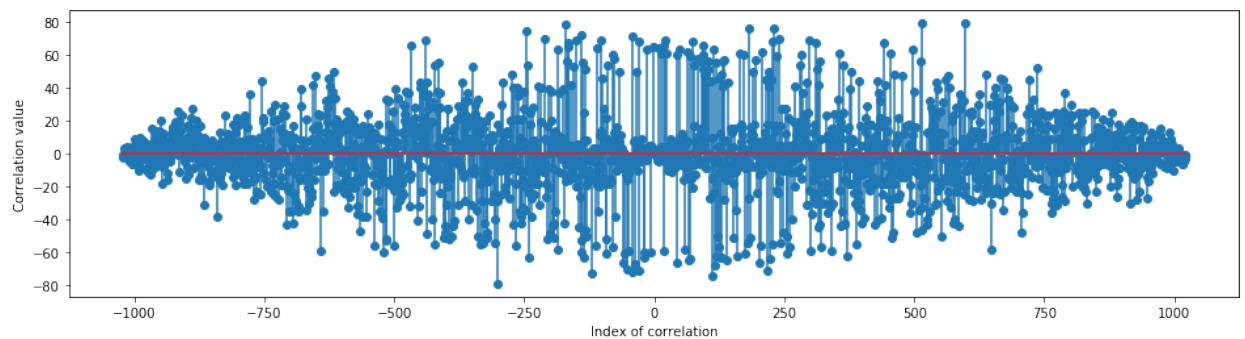
(ind_10, self_10) = cross_correlation(array_10, array_13)

plt.figure(figsize=(16, 4))
plt.stem(ind_10, self_10)
plt.xlabel("Index of correlation")
plt.ylabel("Correlation value")
```

/Users/manlai/miniconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:7: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use\_line\_collection" keyword argument to True.

```
import sys
```

```
Out[10]: Text(0, 0.5, 'Correlation value')
```



We see that the cross-correlation of a Gold code of any satellite with any other satellite is very low. This indicates that when given some unknown data, we can differentiate between different satellites.

## Part (c)

```
In [11]: ## THIS IS A HELPER FUNCTION FOR PART C THAT GENERATES +-1 RANDOM NOISE
def integernoise_generator(length_of_noise):
    noise_array = np.random.randint(2, size = length_of_noise)
    noise_array = 2 * noise_array - np.ones(size(noise_array))
    return noise_array

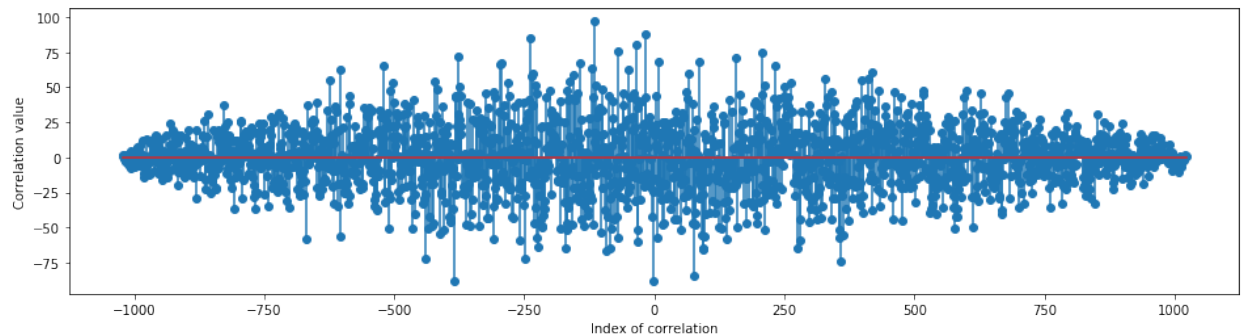
array_10 = Gold_code_satellite(10)
(ind_10, self_10) = cross_correlation(array_10, integernoise_generator)

plt.figure(figsize=(16, 4))
plt.stem(ind_10, self_10)
plt.xlabel("Index of correlation")
plt.ylabel("Correlation value")
```

/Users/manlai/miniconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:11: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use\_line\_collection" keyword argument to True.

```
# This is added back by InteractiveShellApp.init_path()
```

```
Out[11]: Text(0, 0.5, 'Correlation value')
```



We see that the cross-correlation of the Gold code of any satellite with integer noise is very low. This indicates that we can still figure out the presence of a satellite even if it is buried in noise.

## Part (d)



```
In [12]: ## THIS IS A HELPER FUNCTION FOR PART D THAT GENERATES REAL VALUED RANDOM NOISE
def gaussiannoise_generator(length_of_noise):
    noise_array = np.random.normal(0, 1, length_of_noise)
    return noise_array

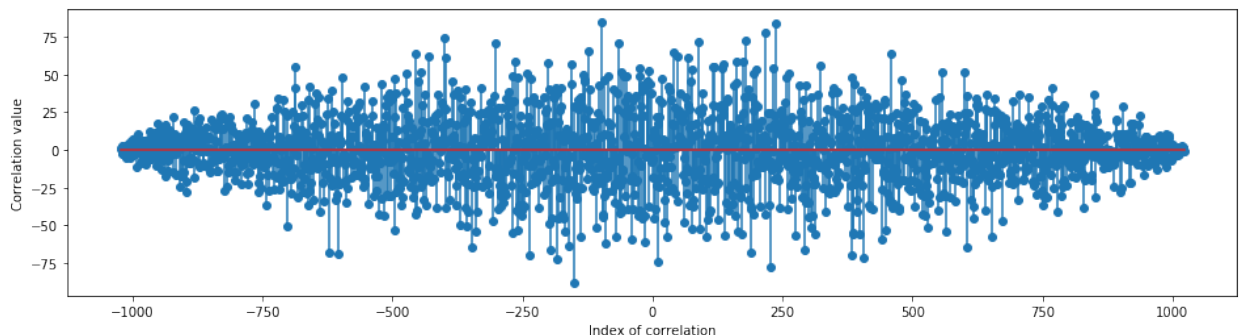
array_10 = Gold_code_satellite(10)
(ind_10, self_10) = cross_correlation(array_10, gaussiannoise_generator(length_of_noise))

plt.figure(figsize=(16, 4))
plt.stem(ind_10, self_10)
plt.xlabel("Index of correlation")
plt.ylabel("Correlation value")
```

/Users/manlai/miniconda3/lib/python3.7/site-packages/ipykernel\_launcher.py:10: UserWarning: In Matplotlib 3.3 individual lines on a stem plot will be added as a LineCollection instead of individual lines. This significantly improves the performance of a stem plot. To remove this warning and switch to the new behaviour, set the "use\_line\_collection" keyword argument to True.

# Remove the CWD from sys.path while we load stuff.

```
Out[12]: Text(0, 0.5, 'Correlation value')
```



We see that the Gold code of any satellite with Gaussian noise is very low. This indicates that we can still figure out the presence of a satellite even if it is buried in Gaussian noise.

## Part (e)

Hint: you can use a absolute value threshold of 800 for the cross-correlation to detect if a given satellite is present. `np.argwhere` may be useful for detecting peak locations.

```
In [13]: #Now let us see which signals are present in the data signal that is a
signal1 = np.load('data1.npy')
```

In [14]: *#Here try plotting the cross-correlations of data1.npy with a few of t*  
*#How can you detect if the satellite is present?*

In [15]: *## This helper function returns 1 if peak (greater than threshold or 1*  
*## You do not have to use this function as there are other solutions t*

```
def find_peak(signal, threshold):
    max_value = np.amax(signal)
    min_value = np.amin(signal)
    if max_value > threshold:
        ret_value = 1
    elif min_value < -1 * threshold:
        ret_value = 1
    else:
        ret_value = 0
    return ret_value
```

In [35]: *## USE 'np.load' FUNCTION TO LOAD THE DATA*  
*## USE DATA1.NPY AS THE SIGNAL ARRAY*  
 for i in range(1, 25, 1):  
 c = cross\_correlation(signal1, Gold\_code\_satellite(i))  
 if find\_peak(c[1], 800) != 0:  
 print("Satellite", i, "is present")

```
Satellite 4 is present
Satellite 7 is present
Satellite 13 is present
Satellite 19 is present
```

## Part (f)

In [36]: *## USE DATA2.NPY AS THE SIGNAL ARRAY*  
 signal2 = np.load('data2.npy')  
 for i in range(1, 25, 1):  
 c = cross\_correlation(signal2, Gold\_code\_satellite(i))  
 if find\_peak(c[1], 800) != 0:  
 print("Satellite", i, "is present")  
 print("The data transmitted:", [int(i/abs(i)) for i in c[1] if

```
Satellite 3 is present
The data transmitted: [1, -1, -1, -1, 1]
```

## Part (g)

```
In [38]: ## USE DATA3.NPY AS THE SIGNAL ARRAY

signal3 = np.load('data3.npy')
for i in range(1, 25, 1):
    c = cross_correlation(signal3, Gold_code_satellite(i))
    if find_peak(c[1], 800) != 0:
        print("Satellite", i, "is present")
```

Satellite 5 is present  
Satellite 20 is present

```
In [49]: ## We know that the data is 1, 1, -1, -1, -1, so we just find the posi
## plot the appropriate cross_correlation and find the location of the
## Do this for as many satellites as there are present
for i in [5, 20]:
    data = list(Gold_code_satellite(i))
    data = np.append(data, data)
    data = np.append(data, -3 * data)
    c = cross_correlation(data, signal3)
    print("Delay of satellite", i, "is", [j for j in range(len(c[1]))
```

Delay of satellite 5 is 1528  
Delay of satellite 20 is 1022

```
In [1]: # PROBLEM 6

# Part B
import numpy as np
print(np.correlate([1, 1, -1, 1, -1, -1, -1, 1, -1, 1], [1, 1, -1, 1,
[-1  0  1 -2  5 -1  1 -1  1 -5  2 -1  0  1]
```

In [ ]: