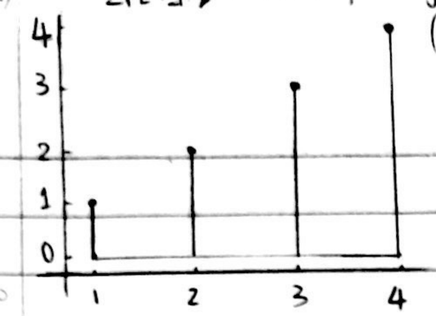
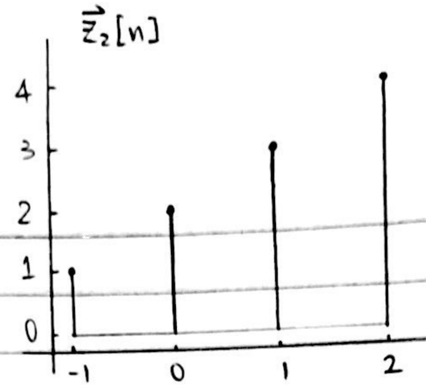
1. Mechanical Correlation

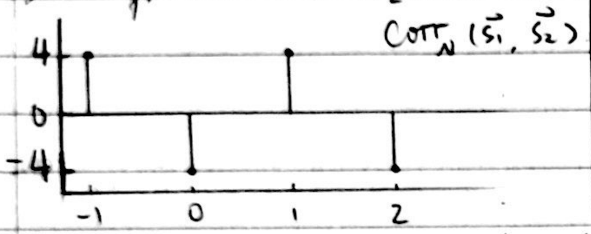
(a) $\vec{z}_1[n]$ ↙ 0 outside of the region drawn → (both signals extends to $\pm\infty$)

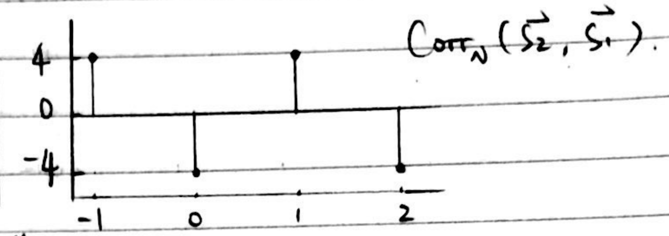


$\vec{z}_2[n]$



(b) Since each signal is periodic with period 4, so $corr(\vec{s}_1, \vec{s}_2)[0] = -4$.
$corr(\vec{s}_1, \vec{s}_2)[1] = 4$. In general, $corr(\vec{s}_1, \vec{s}_2)[2k] = -4$ $corr(\vec{s}_1, \vec{s}_2)[2k+1] = 4$
Similarly, $corr(\vec{s}_2, \vec{s}_1)[2k] = -4$, $corr(\vec{s}_2, \vec{s}_1)[2k+1] = 4$ ← where $k \in \mathbb{Z}$

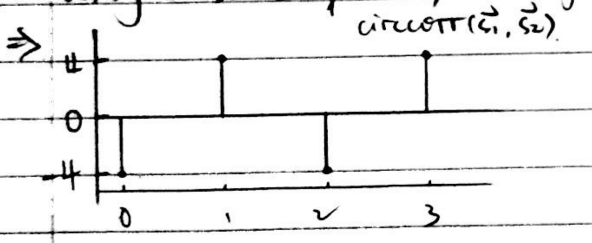


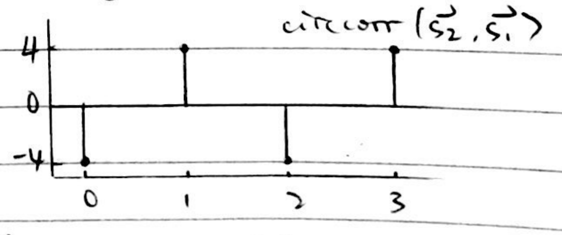


They're the same in values, but different concept-wise.
They're the mirror image of each other, with symmetry axis at 0 (no shift)

(c) Here, period $N = 4$. Now, we can calculate:
$circcorr(\vec{s}_1, \vec{s}_2)[0] = \sum_{i=0}^{3} \vec{s}_1[i] \vec{s}_2[i_4] = 2\cdot1 + (-2)\cdot2 + 2\cdot3 + (-2)\cdot4 = -4$.
$circcorr(\vec{s}_1, \vec{s}_2)[1] = \sum_{i=0}^{3} \vec{s}_1[i] \vec{s}_2[(i-1)_4] = 2\cdot3 + (-2)\cdot1 + 2\cdot2 + (-2)\cdot3 = 4$.
Similarly, $circcorr(\vec{s}_1, \vec{s}_2)[2] = -4$, $circcorr(\vec{s}_1, \vec{s}_2)[3] = 4$.
Thus. $circcorr(\vec{s}_1, \vec{s}_2) = [-4 \ 4 \ -4 \ 4]^T$.
Using a similar process, we can get $circcorr(\vec{s}_2, \vec{s}_1) = [-4 \ 4 \ -4 \ 4]^T$.

⇒



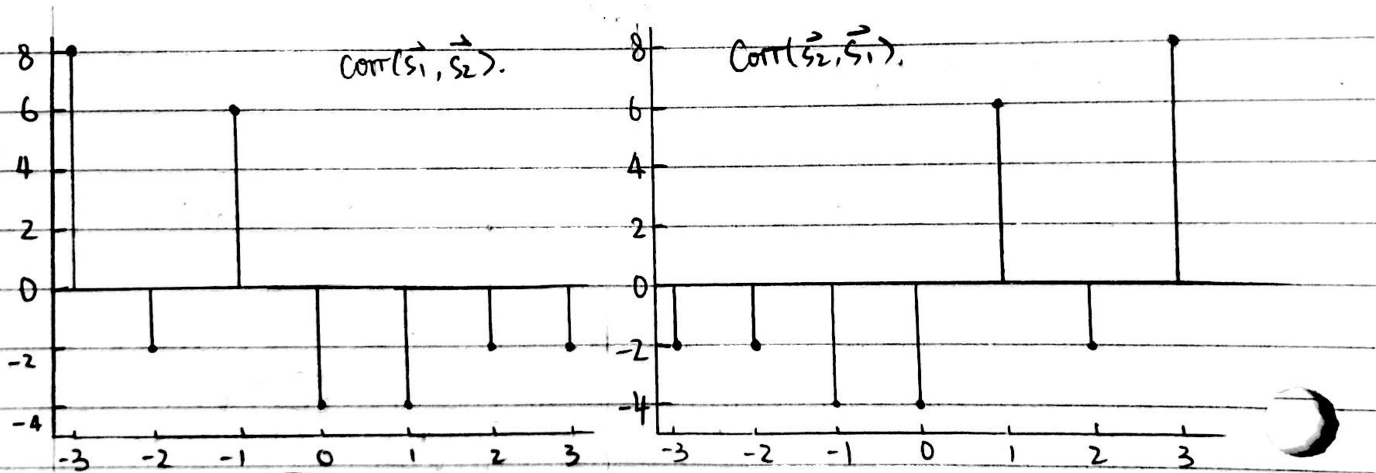


Again, they're the same in value, but different in concept (mirror with axis at 0)
They are related since they check for similarity between $\vec{s}_1, \vec{s}_2$ by shifting one of them.

(d) In this situation, we can calculate that (by definition):
$corr(\vec{s}_1, \vec{s}_2)[0] = corr(\vec{s}_2, \vec{s}_1)[0] = -4$, $corr(\vec{s}_1, \vec{s}_2)[1] = corr(\vec{s}_2, \vec{s}_1)[-1] = -4$.
$corr(\vec{s}_1, \vec{s}_2)[2] = corr(\vec{s}_2, \vec{s}_1)[-2] = -2$, $corr(\vec{s}_1, \vec{s}_2)[3] = corr(\vec{s}_2, \vec{s}_1)[-3] = -2$.
$corr(\vec{s}_1, \vec{s}_2)[-1] = corr(\vec{s}_2, \vec{s}_1)[1] = 6$, $corr(\vec{s}_1, \vec{s}_2)[-2] = corr(\vec{s}_2, \vec{s}_1)[2] = -2$
$corr(\vec{s}_1, \vec{s}_2)[-3] = corr(\vec{s}_2, \vec{s}_1)[3] = 8$
All other values of shifting (linear cross-correlation at other values)
are 0 since they're sum of 0s

(d) Cont.

both signals extends to $\pm\infty$, with 0 outside the region shown.



They are $\boxed{\text{not}}$ the same, but they are mirror image of each other, with the symmetrical "axis" at 0 (no-shift).

(e) Checked. Should use mode $\boxed{\text{"full"}}$.

f. Since we've shown that the inner product is the cosine of the angle between any two unit vectors, and since $\mathrm{corr}_N(\vec{x},\vec{x})[k]$ is just the inner product of two vectors, so $\frac{|\mathrm{corr}_N(\vec{x},\vec{x})[k]|}{\|\vec{x}\|^2} \leq \cos 0 = 1 \; \forall k \in \mathbb{Z}$. Then, $\mathrm{corr}_N(\vec{x},\vec{x})[0] = \|\vec{x}\|^2$, with $|\mathrm{corr}_N(\vec{x},\vec{x})[k]| \leq \|\vec{x}\|^2$, so $\mathrm{corr}_N(\vec{x},\vec{x})[0] \geq |\mathrm{corr}_N(\vec{x},\vec{x})[m]|$ for all m. Q.E.D.

2. (a). I observe a graph with a bunch of small y-value indices from x = -1000 to 1000, and one very large y-value at x = 0. (very high autocorrelation at 0 and low everywhere else)

(b). I see all y-values bounded by the range -80 to 80, which is relatively small compared to result in (a). In other, very low cross-correlation

(c). Again, the cross-correlation is very low (bounded by -100 to 75). This means that we have a strong ability to identify satellites.

(d). Again, very small y-values ⇒ the cross-correlation is small (-75 to 75)

(e). The satellites present are 4, 7, 13, 19

(f). Satellite 3, and message is [1 -1 -1 -1 1]

(g). Satellites 5 and 20. Delay is 506.

3\. (a). No, sim_1 wouldn't be a good similarity measure.
because absolute value measures more of the distance (and thus the differences) between the two vectors, which is the opposite of what we want.

Yes, sim_2 would be good. because correlation measures the cosine of the angle between the vectors. The smaller the angle (i.e. the greater the similarity), the larger the score.
Thus, $\langle \vec{x_c}, \frac{\vec{S_A}}{\|S_A\|} \rangle$ is a good similarity measure.

(b). We can setup the system of linear equations with the information:

$$\begin{bmatrix} 40\% & 33\% & 22\% & 5\% \\ 70\% & 10\% & 10\% & 10\% \\ 20\% & 10\% & 15\% & 55\% \\ 5\% & 2\% & 20\% & 73\% \end{bmatrix} \cdot x_c = \begin{bmatrix} T_{food}\ \% \\ T_{movies}\ \% \\ T_{art}\ \% \\ T_{book}\ \% \end{bmatrix}$$

(c) Algorithm 1

1. procedure PROMOTION(M_{food}, M_{movies}, M_{art}, M_{books}, $\vec{S_{A_1}}$, ..., $\vec{S_{A_N}}$).

⇒

2. $T_{food} = M_{food} / (M_{food} + M_{movies} + M_{art} + M_{books})$
3. $T_{movies} = M_{movies} / (M_{food} + M_{movies} + M_{art} + M_{books})$
4. $T_{art} = M_{art} / (M_{food} + M_{movies} + M_{art} + M_{books})$.
5. $T_{books} = M_{books} / (M_{food} + M_{movies} + M_{art} + M_{books})$.
6. Setup and solve: $\begin{bmatrix} 0.4 & 0.33 & 0.22 & 0.05 \\ 0.7 & 0.1 & 0.1 & 0.1 \\ 0.2 & 0.1 & 0.15 & 0.55 \\ 0.05 & 0.02 & 0.2 & 0.73 \end{bmatrix} \cdot \vec{x_c} = \begin{bmatrix} T_{food} \\ T_{movies} \\ T_{art} \\ T_{books} \end{bmatrix}$

   Solve for $\vec{x_c}$

7. ...
8. ... (algorithm on the hw pdf).
9. ...
10.

(d). First we calculate the spending percentage vector. $\vec{T_c} = [T_{food}\ T_{movies}\ T_{art}\ T_{books}]^T$.

where $T_{food} = 6/(6+4+1+5) = 0.375 = 37.5\%$

similarly $T_{movies} = 25\%$. $T_{art} = 6.25\%$. $T_{books} = 31.25\%$

Now, using IPython, we figured out that:

$$\vec{x_c} = \begin{bmatrix} 0.0782 \\ -2.1456 \\ 4.9815 \\ -0.8833 \end{bmatrix}$$

Then, using IPython, we figure out the best promotions with our similarity score in part (a), which is:

$$\Rightarrow \vec{S_{A_3}} = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ \frac{5}{2} \\ -\frac{1}{2} \end{bmatrix}$$

e, $\boxed{\underline{No}}$ Using IPython, we found out that the 4x4 spending distribution matrix is full rank. In other words, it's invertible.

Thus, for any customer with percentage vector $\vec{T_c} = [T_{food}\ T_{movies}\ T_{art}\ T_{books}]^T$, we have $\text{spending} \cdot \vec{x_c} = \vec{T_c}$, so $\vec{x_c} = \text{spending}^{-1} \cdot \vec{T_c}$ is unique.

which implies that for all customers, the system has a unique solution.

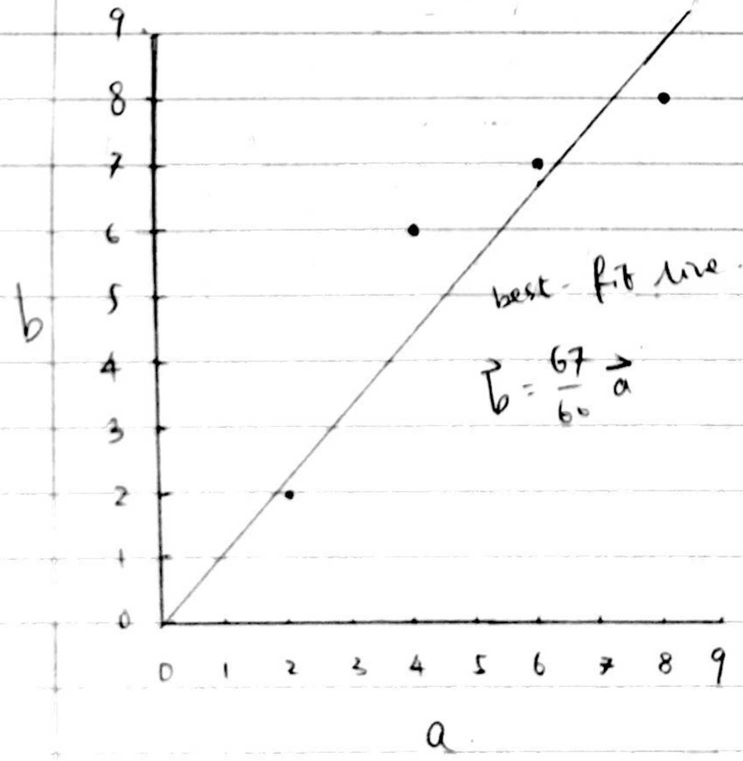
4. (a) Since $\vec{a} = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix}$, $\vec{b} = \begin{bmatrix} 2 \\ 6 \\ 7 \\ 8 \end{bmatrix}$, so $\vec{a}^T \vec{a} = 120$, $\vec{a}^T \vec{b} = 4 + 24 + 42 + 64$

$= 134$

So $x = (\vec{a}^T\vec{a})^{-1} \vec{e}^T \vec{b} = 120^{-1}\ 124 = \boxed{\frac{67}{60}}$.

Squared error $e_1 = \| \vec{b} - \vec{a}x \|^2 = \| [-\frac{7}{30} \quad \frac{23}{15} \quad \frac{3}{10} \quad -\frac{14}{15}] \|^2$

$= \left(\sqrt{\frac{101}{30}} \right)^2 = \frac{101}{30} \approx \boxed{3.37}$



(b) Since $A = \begin{bmatrix} 2 & 1 \\ 4 & 1 \\ 6 & 1 \\ 8 & 1 \end{bmatrix}$, so $A^T = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix}$, and with $\vec{b} = \begin{bmatrix} 2 \\ 6 \\ 7 \\ 8 \end{bmatrix}$

So $A^T A = \begin{bmatrix} 120 & 20 \\ 20 & 4 \end{bmatrix}$, so $(A^T A)^{-1} = \frac{1}{120 \cdot 4 - 20^2} \cdot \begin{bmatrix} 4 & -20 \\ -20 & 120 \end{bmatrix}$

$$= \begin{bmatrix} \frac{1}{20} & -\frac{1}{4} \\ -\frac{1}{4} & \frac{3}{2} \end{bmatrix}$$

$$\Rightarrow (A^T A)^{-1} \cdot A^T = \begin{bmatrix} -\frac{3}{20} & -\frac{1}{20} & \frac{1}{20} & \frac{3}{20} \\ 1 & \frac{1}{2} & 0 & -\frac{1}{2} \end{bmatrix}$$

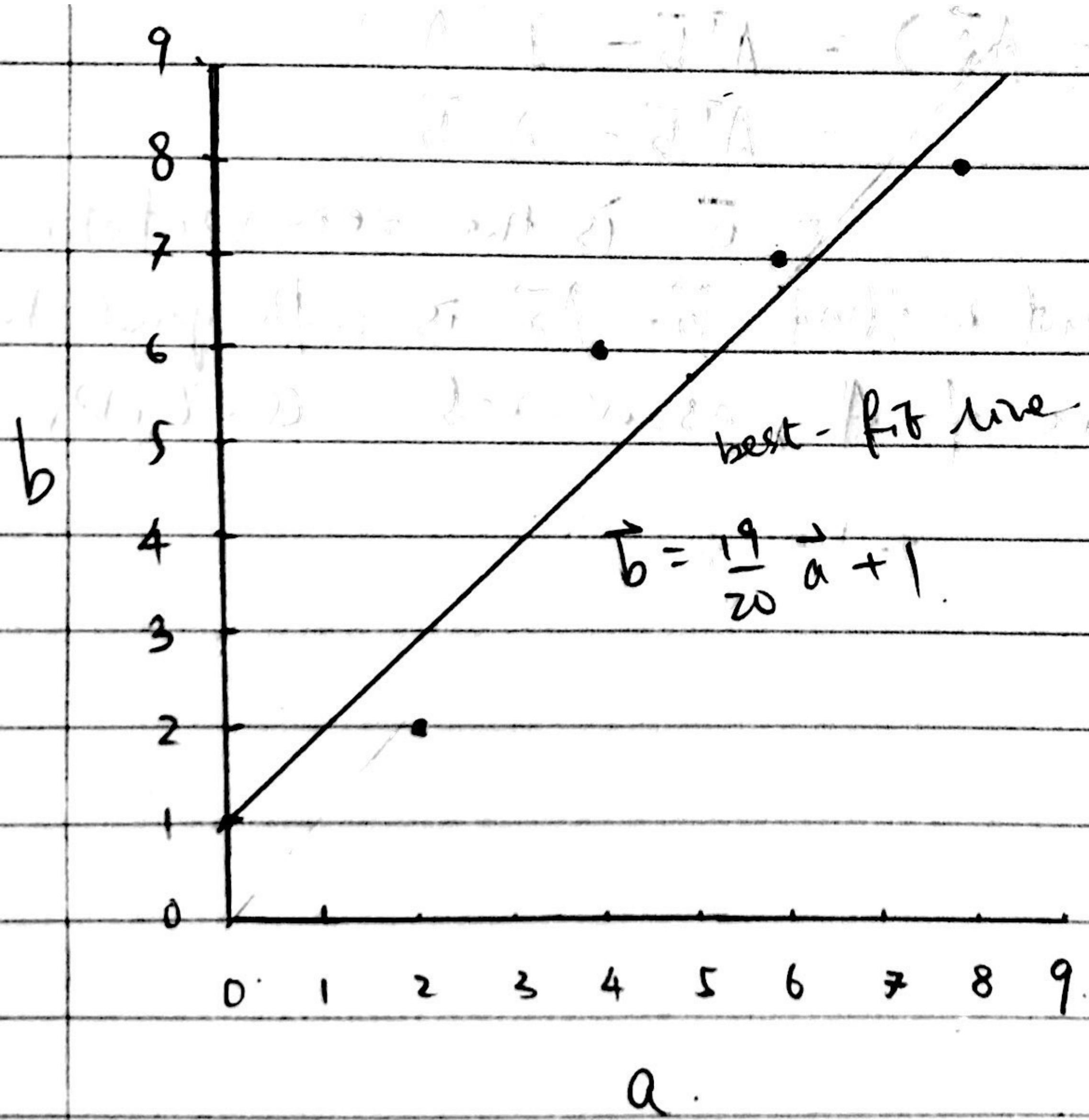
Thus, $\vec{x} = (A^T A)^{-1} A^T \vec{b} = \begin{bmatrix} \frac{19}{20} \\ 1 \end{bmatrix} \Rightarrow \boxed{x_1 = \frac{19}{20} \quad x_2 = 1}$.

Here, Squared error $e_2 = \left\| \vec{b} - (x_1 \vec{a} + x_2) \right\|^2 = \left\| [2\ 6\ 7\ 8]^T - [\frac{29}{10}\ \frac{24}{5}\ \frac{67}{10}\ \frac{43}{5}]^T \right\|^2$

$$= \left\| [-\frac{9}{10}\ \ \frac{6}{5}\ \ \frac{3}{10}\ \ -\frac{3}{5}] \right\|^2$$

$$= \left(\sqrt{\frac{270}{100}}\right)^2 = \frac{27}{10} = \boxed{2.7}$$

Since $e_2 < e_1$, so $\boxed{\text{Yes}}$, it's a better fit. (also by the plot).



best-fit line
$\vec{b} = \frac{19}{20}\vec{a} + 1$
b
a
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9

(c). From the notes we know that $\vec{\hat{x}} = (A^TA)^{-1} A^T \vec{b}$.

So $\vec{b} - A\vec{\hat{x}} = \vec{b} - A(A^TA)^{-1}A^T\vec{b}$.

Now, consider $A^T(\vec{b} - A\vec{\hat{x}}) = A^T(\vec{b} - A(A^TA)^{-1}A^T\vec{b})$

$= A^T\vec{b} - A^TA(A^TA)^{-1}A^T\vec{b}$.

Since $(A^TA)\cdot(A^TA)^{-1} = I$, the identity matrix,

so $A^T(\vec{b} - A\vec{\hat{x}}) = A^T\vec{b} - I \cdot A^T\vec{b}$

$= A^T\vec{b} - A^T\vec{b}$

$= \vec{0}$ is the zero vector.

which is equivalent to that $\vec{b} - A\vec{x}$ is orthogonal to the columns of A, as desired. Q.E.D.

5. Find an orthorgonal vector to $\vec{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$.

Let $\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}$ be orthorgonal to $\vec{v}$. so $\langle \vec{u}, \vec{v} \rangle = u_1 + 2u_2 + 3u_3 = 0$.

There are infinitely-many solutions since we have 3 variables and 1 equations. One solution is $u_1 = u_2 = 1$ $u_3 = -1$.

so $\vec{u} = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$ is orthorgonal to $\vec{v}$.

6. I worked alone without getting any help.

EE16A Homework 12

Question 1: Mechanical Correlation

Part (e)

In [5]:
```
## your code here
import numpy as np

s1 = np.array([2, -2, 2, -2])
s2 = np.array([1, 2, 3, 4])

print('corr[s1, s2]:', np.correlate(s1, s2, "full"))
print('corr[s2, s1]:', np.correlate(s2, s1, "full"))
```

```
corr[s1, s2]: [ 8 -2  6 -4 -4 -2 -2]
corr[s2, s1]: [-2 -2 -4 -4  6 -2  8]
```

Question 2: GPS Receivers

In [7]:
```
%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 [8]:
```
## RUN THIS FUNCTION BEFORE YOU START THIS PROBLEM
## This function will generate the gold code associated with the satel
## 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]*re
        register2 = np.roll(register2,1)
        register2[0] = modulo

    delay = np.array([5,6,7,8,17,18,139,140,141,251,252,254,255,256,25
    G1_out = MLS1;
    shamt = delay[satellite_ID - 1]
    G2_out = np.roll(MLS2,shamt)

    CA_code = G1_out * G2_out

    return CA_code
```

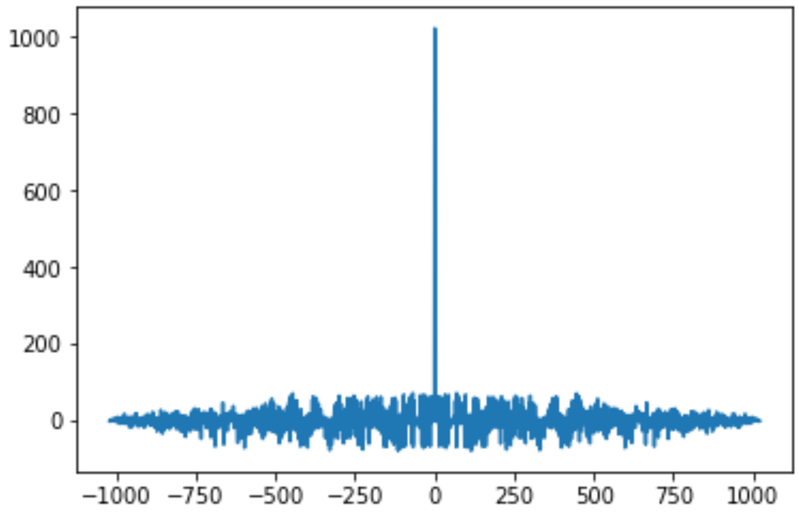
Part (a)

In [17]:
```
def array_correlation(array1, array2):
    """ This function should return two arrays or a matrix with one row
    the offset and other to the correlation value
    """
    ## YOUR CODE HERE
    correlation = np.correlate(array1, array2, 'full')
    length = max(len(array1), len(array2))
    offset = np.array(range(-length+1, length))
    return offset, correlation
    ## Use np.correlate with "FULL". Check out the documentation page.

# Plot the auto-correlation of satellite 10 with itself. Your signal sh
# at offset = 0.
# Use plt.plot or plt.stem to plot.

# YOUR CODE HERE
sate = Gold_code_satellite(10)
a, b = array_correlation(sate, sate)
plt.plot(a, b)
```

Out[17]: [<matplotlib.lines.Line2D at 0x1167349e8>]

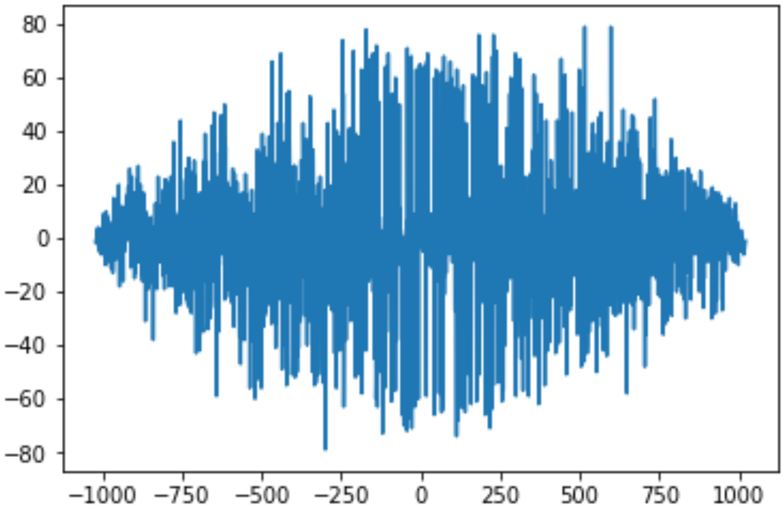


Part (b)

In [18]:
```
# YOUR CODE HERE
sate10 = Gold_code_satellite(10)
sate13 = Gold_code_satellite(13)

a, b = array_correlation(sate10, sate13)
plt.plot(a, b)
```

Out[18]: [<matplotlib.lines.Line2D at 0x11679fcc0>]



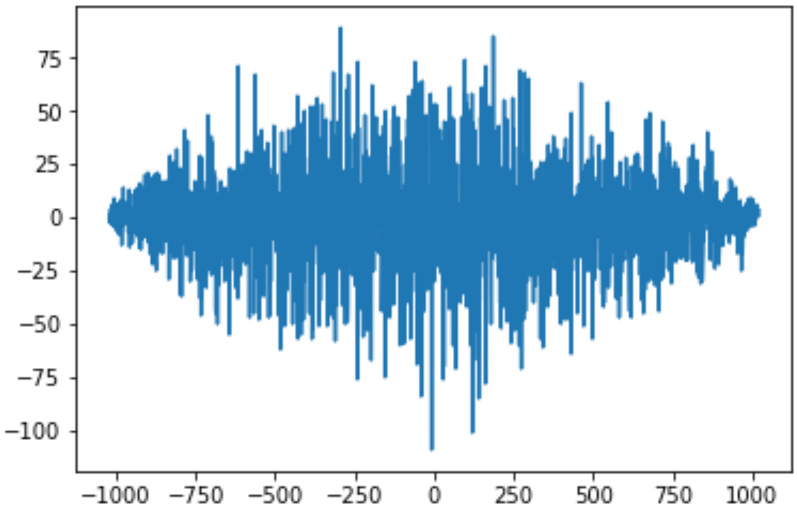
Part (c)

In [20]:
```
## THIS IS A HELPER FUNCTION FOR PART C
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

# YOUR CODE HERE
sate10 = Gold_code_satellite(10)
random = integernoise_generator(len(sate10))

a, b = array_correlation(sate10, random)
plt.plot(a, b)
```

Out[20]: [<matplotlib.lines.Line2D at 0x1169d4e80>]



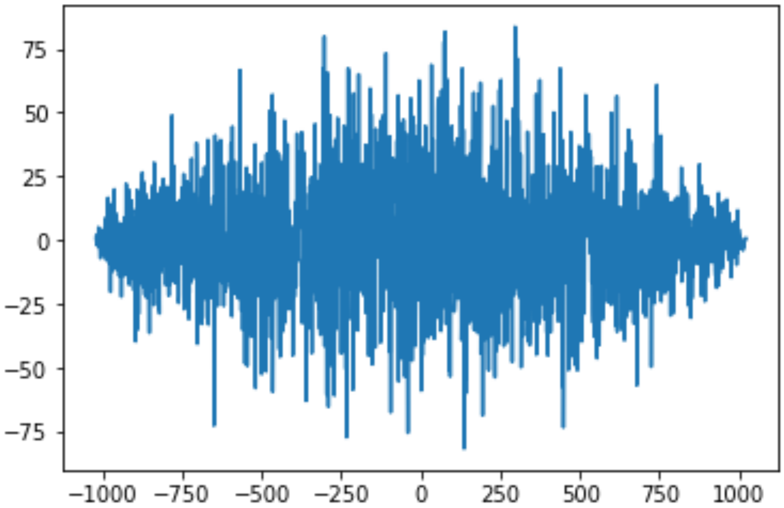
Part (d)

In [23]:
```
## THIS IS A HELPER FUNCTION FOR PART D
def gaussiannoise_generator(length_of_noise):
    noise_array = np.random.normal(0, 1, length_of_noise)
    return noise_array

# YOUR CODE HERE
noise = gaussiannoise_generator(1023)
sate10 = Gold_code_satellite(10)

a, b = array_correlation(sate10, noise)
plt.plot(a, b)
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x116bd0b38>]
```



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 [126]:
```
## USE 'np.load' FUNCTION TO LOAD THE DATA
## USE DATA1.NPY AS THE SIGNAL ARRAY
data = np.load('data1.npy')

present = []

for index in range(1, 25):
    sate = Gold_code_satellite(index)
    _, b = array_correlation(sate, data)
    if np.any(abs(b) >= 800):
        present.append(index)

if len(present) > 0:
    print(present)

# YOUR CODE HERE
```

```
[4, 7, 13, 19]
```

Part (f)

In [131]:
```
## USE DATA2.NPY AS THE SIGNAL ARRAY
data = np.load('data2.npy')

present = 0
for index in range(1, 25):
    sate = Gold_code_satellite(index)
    a, b = array_correlation(sate, data)
    if np.any(abs(b) >= 800):
        present = index
        print('Satellite', present)

bits = []
for i in range(0, 5):
    sate = Gold_code_satellite(present)
    cur_data = data[i*1023:(i+1)*1023]

    _, b = array_correlation(cur_data, sate)

    if np.any(b >= 800):
        bits.append(1)
    elif np.any(b <= -800):
        bits.append(-1)

print('Message:', bits)

# YOUR CODE HERE
```

```
Satellite 3
Message: [1, -1, -1, -1, 1]
```

Part (g)

In [136]:
```
## USE DATA3.NPY AS THE SIGNAL ARRAY
data = np.load('data3.npy')

present = []

for index in range(1, 25):
    sate = Gold_code_satellite(index)
    a, b = array_correlation(sate, data)
    if np.any(abs(b) >= 800):
        present.append(index)

if len(present) > 0:
    print('Satellites:', present)

offset = []
for sate_num in present:
    sate = Gold_code_satellite(sate_num)

    actual_data = np.append(sate, sate)
    actual_data = np.append(actual_data, -actual_data)
    actual_data = np.append(actual_data, -sate)

    a, b = array_correlation(actual_data, data)
    offset.append(np.argwhere(abs(b) >= 800)[0][0])

print('Offsets are:', offset)
delay = abs(offset[0] - offset[1])
print('Relative Delay is:', delay)

# YOUR CODE HERE
```

```
Satellites: [5, 20]
Offsets are: [1528, 1022]
Relative Delay is: 506
```

Question 3: Retail Store Marketing

Part (d)

In [145]:
```
spending = np.array([
    [0.40, 0.33, 0.22, 0.05],
    [0.70, 0.10, 0.10, 0.10],
    [0.20, 0.10, 0.15, 0.55],
    [0.05, 0.02, 0.20, 0.73]
])

T = np.array([0.375, 0.25, 0.0625, 0.3125])

x = np.linalg.solve(spending, T)
print(x)
```

```
[ 0.07819672 -2.14557377  4.98147541 -0.88327869]
```

In [166]:
```
sA = np.array([
    [1/2, 1/2, -1/2, 1/2],
    [2/3, -1/2, 1/2, 1/3],
    [-1/2, -1/2, 5/2, -1/2],
    [0, 1/2, 0, 1/2]
])

similarity = []
for promo in sA:
    similarity.append(np.correlate(promo/np.linalg.norm(promo), x))

max_sim = np.max(similarity)
index = np.argwhere(similarity == max_sim)[0][0]
print('Use promo sA', index+1, ':', sA[index])
```

```
Use promo sA 3 : [-0.5 -0.5  2.5 -0.5]
```

Part (e)

In [173]:
```
rank = np.linalg.matrix_rank(spending)

if rank < spending[0].size:
    print('Linearly dependent')
else:
    print('Full rank! (Linearly independent)')
```

```
Full rank! (Linearly independent)
```

In [ ]: