HW 11 Problem 1

a. Signal 1: $\begin{bmatrix}2\\-2\\2\\-2\\-2\\-2\\2\\-2\\2\\2\end{bmatrix}\begin{bmatrix}2\\-2\\2\\-2\\-2\\2\\-2\\2\\2\\-2\end{bmatrix}$ ⇒ Using iPython ⇒ $\begin{bmatrix}40\\-8\\8\\-8\\8\\-40\\8\\-8\\8\\-8\end{bmatrix}$

Signal 2: $\begin{bmatrix}1\\2\\3\\4\\5\\6\\7\\6\\5\\4\end{bmatrix}$ ⇒ Using iPython ⇒ $\begin{bmatrix}217\\208\\193\\176\\161\\156\\161\\176\\193\\208\end{bmatrix}$

b. Cross-Correlate

$\begin{bmatrix}2\\-2\\2\\-2\\-2\\-2\\2\\-2\\2\\2\end{bmatrix}\begin{bmatrix}1\\2\\3\\4\\5\\6\\7\\6\\5\\4\end{bmatrix}$ ⇒ Using iPython ⇒ $\begin{bmatrix}-6\\6\\10\\22\\18\\6\\-6\\-10\\-22\\-18\end{bmatrix}$

Problem 2

a. The values are fairly small until the massive spike to ~1000. Therefore, the autocorrelation at the 0 offset is large.

b. The values are very small and only have a range from -80 – 80.

c. The shape of the graph is very similar to that of the cross-correlation between satellite 10 and 13. This means that the random signal does not match satellite 13. We can identify the satellites by doing cross-correlation and observe which one is highest.

d. The graph is still similar to part (b) and (c). This means that the gaussian noise creates "static" or noise that jumbles up the values.

e. The satellites are **4, 17, 13 and 19.**

f. The satellite is Gold code satellite 3.
The sequence is [1, -1, -1, -1, 1]

g. I expect the values of the cross-correlation to be significantly higher with the proper satellite than the other ones.

The satellites are 5 and 20. The relative delay is **506 samples**

Problem 3

a. The largest inner products is when $j \neq 0$ is always, seemingly less than 0.1.

b. Yes, we can find the delay b/c the peaks occur at different times.
This works b/c we are able to see at which time the signal $\vec{y}$ is most similar to $\vec{s}_1^{(j)}$

c. The $\langle \vec{s}_1, n \rangle$ is very small and is never higher than 0.4. This closely relates to the signal b/c they are around the same values.

d. Yes we can find the delay. This works b/c there is only one peak and all the other values.

e. The cross-correlation still works for $\vec{y} = \vec{s}_1^{(j)} + \vec{n}$ b/c there is still a distinct peak.

However, the cross-correlation does not $\vec{y} = \vec{s}_1^{(j)} + 10\vec{n}$ b/c there is too much noise for there to be a distinct label.

f. Yes, we can find the delays b/c there are two different distinct points where peaks, occur.

g. Yes, we can find signal 1 b/c there is a distinct point where a peak occurs.
However, we cannot find signal 2 b/c there is too much noise

h. Yes, this does works. This works b/c the signal s_1 has a larger peak and thus the cross-correlation becomes more distinct. Thus when you subtract s_1, s_2 becomes clearer.

Problem 3 cont.

i Yes we can use inner-products to find the amplitudes b/c the signal is normalized so we get an amplitude that is not multiplied by some factor.

They are very close b/c they are off by very small margins.

j The amplitudes are still very close b/c the amplitudes are off by only a small margin.

Problem 4.

a. $A = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix}$ $b = \begin{bmatrix} 3 \\ 6 \\ 7 \\ 8 \end{bmatrix}$

$A^TA = \begin{bmatrix} 2 & 4 & 6 & 8 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} = 4+16+36+64 = 120$

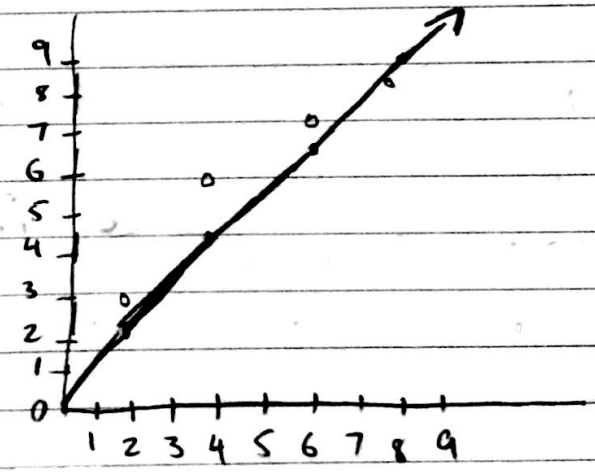
$A^Tb = \begin{bmatrix} 2 & 4 & 6 & 8 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \\ 7 \\ 8 \end{bmatrix} = 6+24+42+64 = 30+106 = 136$

$x = \frac{136}{120} = \frac{68}{60} = \frac{34}{30} = \boxed{\frac{17}{15}}$

$\frac{17}{15} \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix} = \begin{bmatrix} 34/15 \\ 68/15 \\ 102/15 \\ 136/15 \end{bmatrix}$

$\left(\begin{bmatrix} 3 \\ 6 \\ 7 \\ 8 \end{bmatrix} - \begin{bmatrix} 34/15 \\ 68/15 \\ 102/15 \\ 136/15 \end{bmatrix} \right)^2 =$

Error: $\boxed{3.867}$ $\frac{58}{15}$



b. $A = \begin{bmatrix} 2 & 1 \\ 4 & 1 \\ 6 & 1 \\ 8 & 1 \end{bmatrix}$ $A^TA = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 4 & 1 \\ 6 & 1 \\ 8 & 1 \end{bmatrix} = \begin{bmatrix} 120 & 20 \\ 20 & 4 \end{bmatrix}$

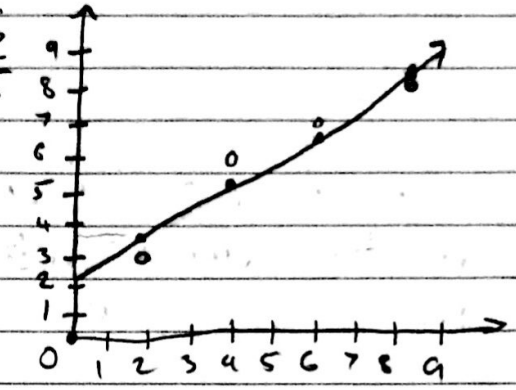
$A^TA^{-1} = \frac{1}{20} \begin{bmatrix} 1 & -5 \\ -5 & 30 \end{bmatrix}$

$A^Tb = \begin{bmatrix} 2 & 4 & 6 & 8 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 6 \\ 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 136 \\ 24 \end{bmatrix}$

$(A^TA)^{-1}A^Tb = \frac{1}{20} \begin{bmatrix} 1 & -5 \\ -5 & 30 \end{bmatrix} \begin{bmatrix} 136 \\ 24 \end{bmatrix} = \begin{bmatrix} 4/5 \\ 2 \end{bmatrix}$

b. cont. $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4/5 \\ 2 \end{bmatrix}$ $A\hat{x} = \hat{b} = \begin{bmatrix} 2 & 1 \\ 4 & 1 \\ 6 & 1 \\ 8 & 1 \end{bmatrix} \begin{bmatrix} 4/5 \\ 2 \end{bmatrix} = \begin{bmatrix} 18/5 \\ 26/5 \\ 34/5 \\ 42/5 \end{bmatrix}$

Error $= \left\| \begin{bmatrix} 3 \\ 6 \\ 7 \\ 8 \end{bmatrix} - \begin{bmatrix} 18/5 \\ 26/5 \\ 34/5 \\ 42/5 \end{bmatrix} \right\|^2 = \boxed{1.2}$ $\frac{6}{5}$



This plot fits the data much better b/c the error is lower

c. Suppose $\hat{b} = A\hat{x} = A(A^TA)^{-1}A^Tb$

Consider $\vec{x}_2 = A\vec{x}$

$\vec{x}_2^T(\vec{b} - \hat{b}) = \vec{x}^T A^T(\vec{b} - A(A^TA)^{-1}A^Tb) = \vec{x}^T(A^Tb - A^TA(A^TA)^{-1}A^Tb)$

$= x^T(A^Tb - A^Tb) = x^T(0)$. B/C $A^T(\vec{b} - \hat{b}) = 0$, we can show that the error vector is $\perp$ to the columns of A.

Problem 5

a. $n \begin{bmatrix} | & | & | \\ x^2+y^2 & x & y \\ | & | & | \end{bmatrix}$ $\quad 3$

$A^TA \quad 3\times n \quad n\times 3$

$$A = \begin{bmatrix} 0.5661 & 0.3 & -0.69 \\ 1.0069 & 0.5 & 0.87 \\ 1.5496 & 0.9 & -0.86 \\ 1.7744 & 1 & 0.88 \\ 2.1124 & 1.2 & -0.82 \\ 2.6593 & 1.5 & 0.64 \\ 3.24 & 1.8 & 0 \end{bmatrix} \quad \vec{x} = \begin{bmatrix} a \\ d \\ e \end{bmatrix} \quad \vec{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \vec{e} = \vec{b} - \hat{b} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{bmatrix}$$

b. $A = \begin{bmatrix} x^2 & xy & y^2 & x & y \end{bmatrix} = \begin{bmatrix} 0.09 & -0.207 & 0.4761 & 0.3 & -0.69 \\ 0.25 & 0.435 & 0.7569 & 0.5 & 0.87 \\ 0.81 & -0.774 & 0.7396 & 0.9 & -0.86 \\ 1 & 0.88 & 0.7744 & 1 & 0.88 \\ 1.44 & -0.984 & 0.6724 & 1.2 & -0.82 \\ 2.25 & 0.96 & 0.4096 & 1.5 & 0.64 \\ 3.24 & 0 & 0 & 1.8 & 0 \end{bmatrix}$

$$b = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \vec{x} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \end{bmatrix}$$

$$\vec{e} = \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{bmatrix} = \vec{b} - \hat{b} = \vec{b} - A\hat{x}$$

c. $\frac{\|c\|}{N} = 0.1375$

d. $\frac{\|e\|}{N} = 0.01285$

The error is significantly lower than the error for the circle. This concludes that the ellipse technique is better.

Problem 6

Question.

Given $x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ and $y = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$, find the best-fit curve in the form $y = \alpha_1 x + \alpha_2$.

Solution:

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}$$

$$A^TA = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 14 & 6 \\ 6 & 3 \end{bmatrix} \qquad A^Tb = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 19 \\ 8 \end{bmatrix}$$

$$A^TA^{-1} = \frac{1}{6} \begin{bmatrix} 3 & -6 \\ -6 & 14 \end{bmatrix} \qquad (A^TA)^{-1}A^Tb = \frac{1}{6} \begin{bmatrix} 3 & -6 \\ -6 & 14 \end{bmatrix} \begin{bmatrix} 19 \\ 8 \end{bmatrix} = \begin{bmatrix} 3/2 \\ -1/3 \end{bmatrix}$$

Solution $\boxed{y = \frac{3}{2}x - \frac{1}{3}}$

7.

I worked by myself with no study group. I went to office hours both Friday and Monday in order to get my work done.

prob11

November 20, 2017

1 EE16A Homework 11

1.1 Question 1: Mechanical Correlation

In [1]:
```
# Any IPython code here
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
import sys

a = np.array([2, -2, 2, -2, -2, -2, 2, -2, 2, 2])
b = np.array([2, -2, 2, -2, -2, -2, 2, -2, 2, 2])


c = []

for x in range(a.shape[0]):
    value = np.dot(np.transpose(a), b)
    c.append(value)
    b = np.roll(b, 1)

print(c)

d = np.array([1, 2, 3, 4, 5, 6, 7, 6, 5, 4])
e = np.array([1, 2, 3, 4, 5, 6, 7, 6, 5, 4])

f = []

for x in range(d.shape[0]):
    value = np.dot(np.transpose(d), e)
    f.append(value)
    e = np.roll(e, 1)

print(f)
```

```
Populating the interactive namespace from numpy and matplotlib
[40, -8, 8, -8, 8, -40, 8, -8, 8, -8]
```

```
[217, 208, 193, 176, 161, 156, 161, 176, 193, 208]
```

```
In [2]: signal1 = np.array([2, -2, 2, -2, -2, -2, 2, -2, 2, 2])
        signal2 = np.array([1, 2, 3, 4, 5, 6, 7, 6, 5, 4])
        cross_corr = []
        for x in range(signal1.shape[0]):
            value = np.dot(np.transpose(signal1), signal2)
            cross_corr.append(value)
            signal2 = np.roll(signal2, 1)

        print(cross_corr)
```

```
[-6, 6, 10, 22, 18, 6, -6, -10, -22, -18]
```

1.2 Question 2: GPS Receivers

```
In [3]: %pylab inline
        import numpy as np
        import matplotlib.pyplot as plt
        import scipy.io
        import sys
```

```
Populating the interactive namespace from numpy and matplotlib
```

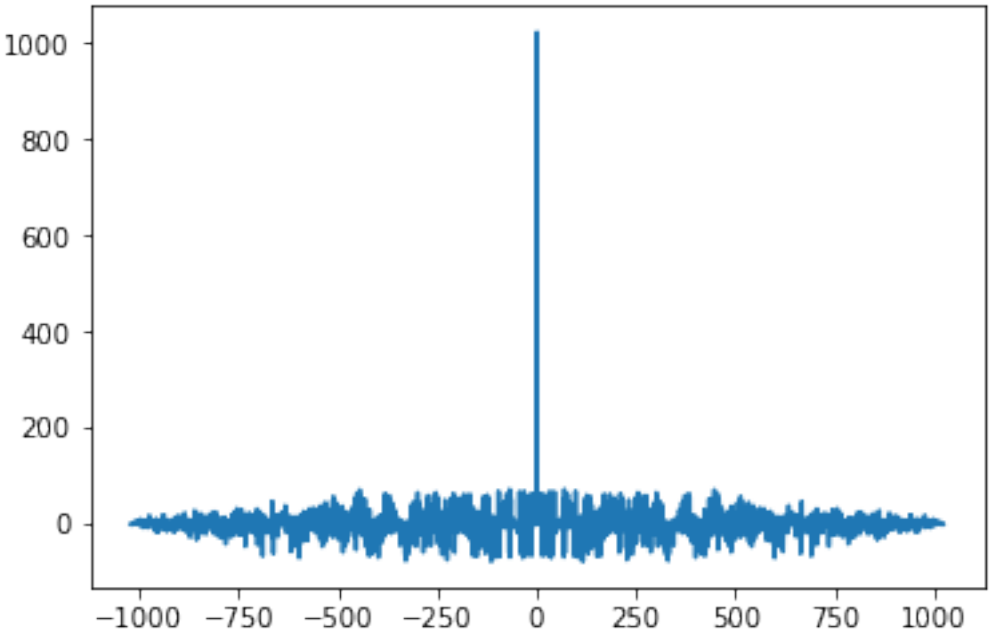
```
/Users/aatifjiwani/anaconda3/lib/python3.6/site-packages/IPython/core/magics/pylab.py:161: User
`%matplotlib` prevents importing * from pylab and numpy
  "\n`%matplotlib` prevents importing * from pylab and numpy"
```

```
In [4]: ## RUN THIS FUNCTION BEFORE YOU START THIS PROBLEM
        ## This function will generate the gold code associated with the satellite ID using li
        ## 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[8]*regis
        register2 = np.roll(register2,1)
        register2[0] = modulo

    delay = np.array([5,6,7,8,17,18,139,140,141,251,252,254,255,256,257,258,469,470,471
    G1_out = MLS1;
    shamt = delay[satellite_ID - 1]
    G2_out = np.roll(MLS2,shamt)

    CA_code = G1_out * G2_out

    return CA_code
```

1.2.1 Part (a)

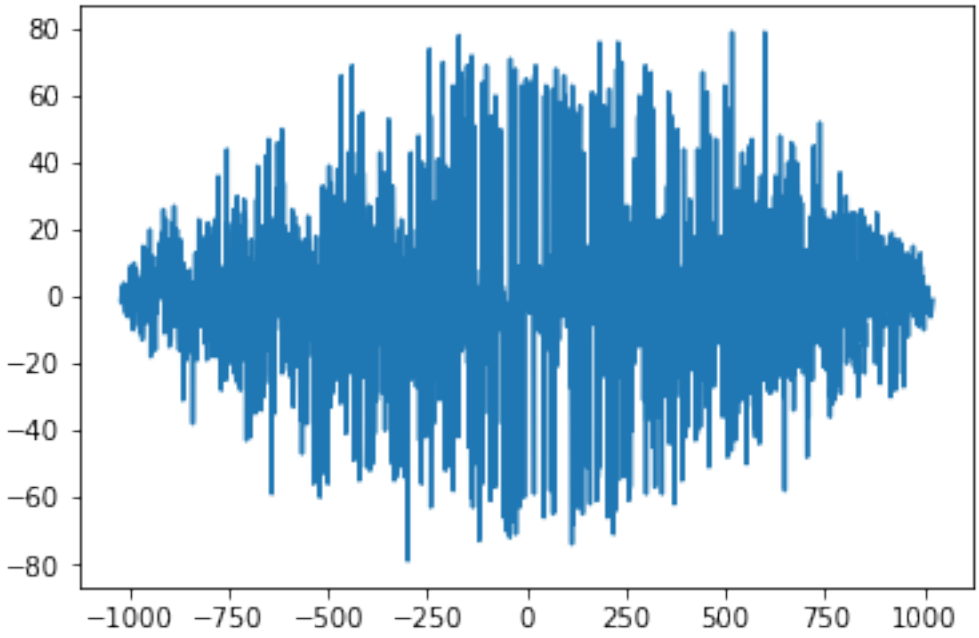
```
In [5]: def array_correlation(array1, array2):
            """ This function should return two arrays or a matrix with one row corresponding
            the offset and other to the correlation value
            """
            ## YOUR CODE HERE
            ## Use np.correlate with "FULL". Check out the documentation page.
            correlate = np.correlate(array1, array2, "FULL")
            offset = [(k - len(correlate)/2 + 1) for k in range(len(correlate))]
            return correlate, offset


        # Plot the auto-correlation of satellite 10 with itself. Your signal should be centere
        # at offset = 0.
        # Use plt.plot or plt.stem to plot.
        array1 = Gold_code_satellite(10)
        array2 = Gold_code_satellite(10)
        x, y = array_correlation(array1, array2)
        plt.plot(y, x)


        # YOUR CODE HERE

Out[5]: [<matplotlib.lines.Line2D at 0x118326160>]
```



1.2.2 Part (b)

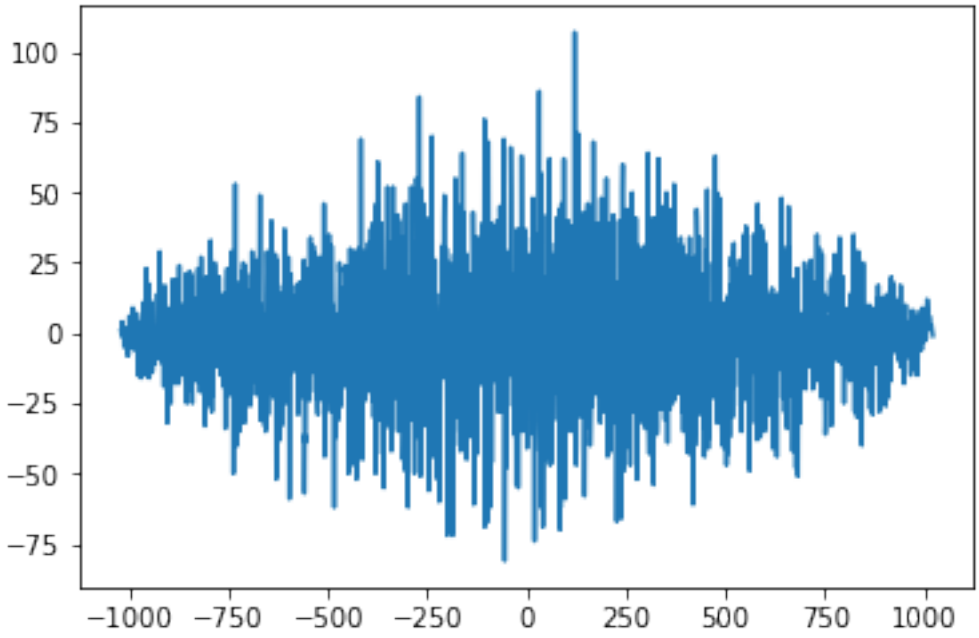
```
In [6]: # YOUR CODE HERE
        array1 = Gold_code_satellite(10)
        array2 = Gold_code_satellite(13)
        x, y = array_correlation(array1, array2)
        plt.plot(y, x)
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x1184a4898>]
```



1.2.3 Part (c)

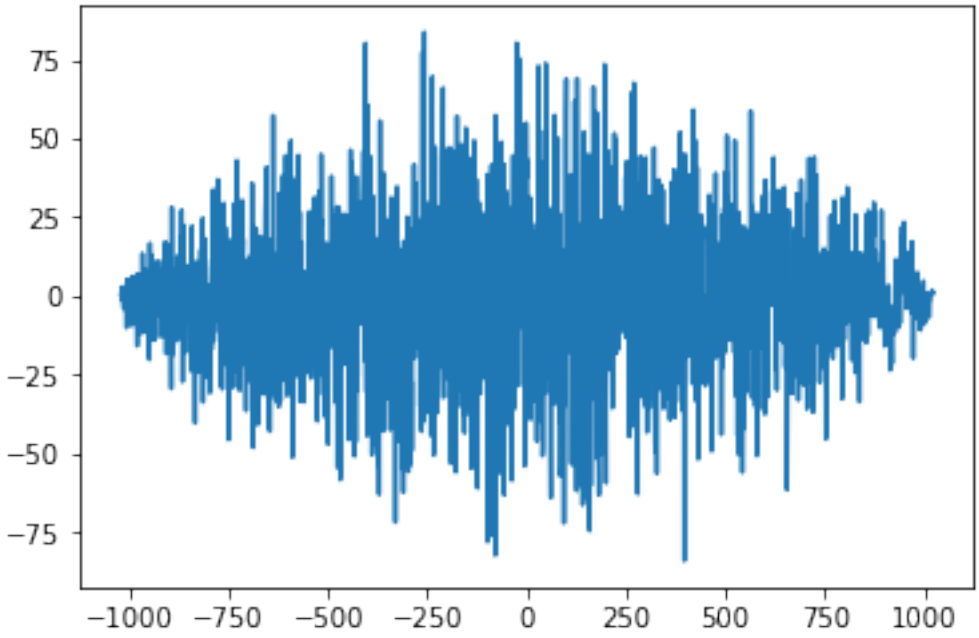
```
In [7]: ## THIS IS A HELPER FUNCTION FOR PART C
        def integernoise_generator(length_of_noise):
            noise_array = np.random.randint(2, size = length_of_noise)
            noise_array = 2 * noise_array - np.ones(size(noise_array))
            return noise_array

        # YOUR CODE HERE
        randomarray = integernoise_generator(1023)
        array1 = Gold_code_satellite(10)
        x, y = array_correlation(array1, randomarray)
        plt.plot(y, x)

Out[7]: [<matplotlib.lines.Line2D at 0x1186f7cc0>]
```



1.2.4 Part (d)

In [8]:
```
## 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
randomarray = gaussiannoise_generator(1023)
array1 = Gold_code_satellite(10)
x, y = array_correlation(array1, randomarray)
plt.plot(y, x)
```

Out[8]:
```
[<matplotlib.lines.Line2D at 0x1187a7f28>]
```

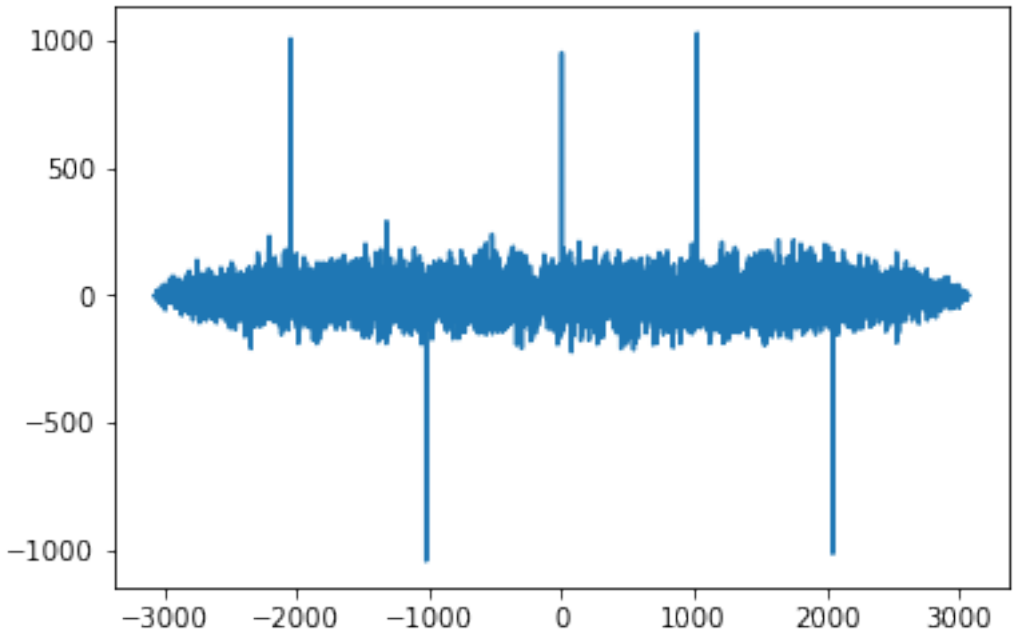


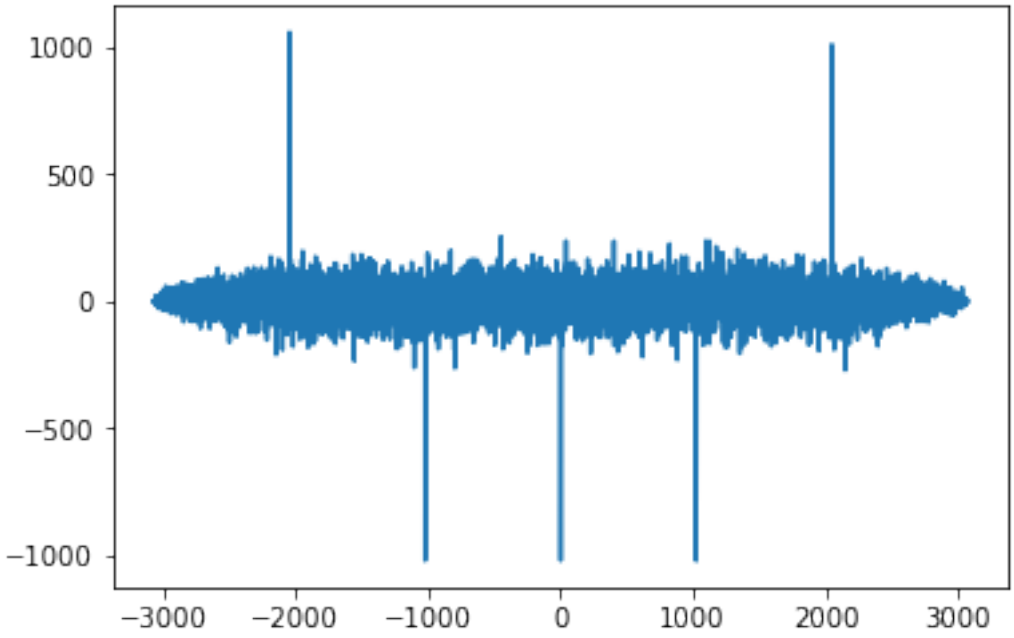
1.2.5 Part (e)

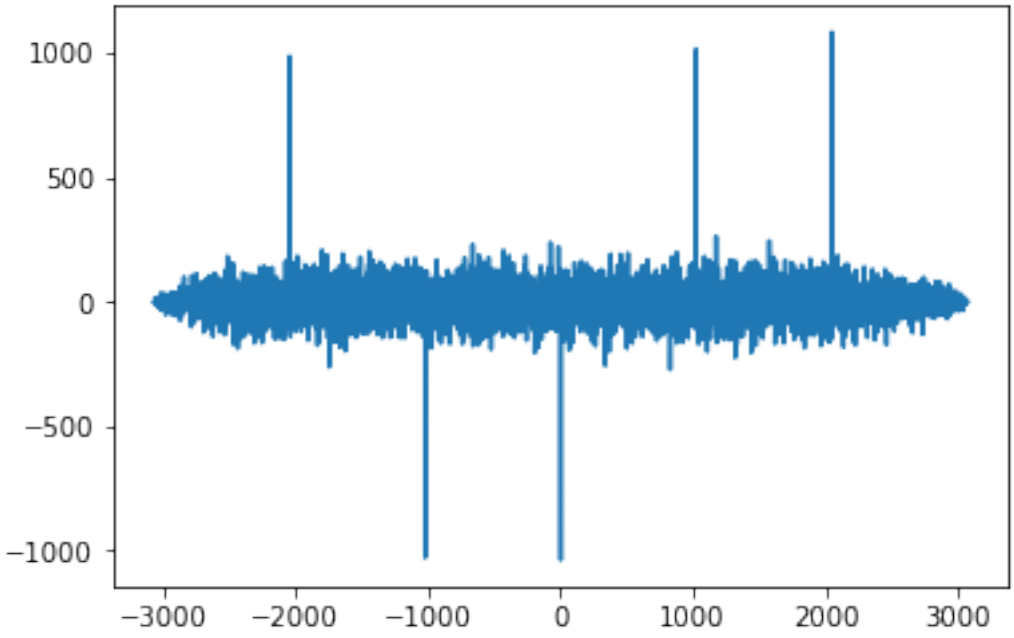
```
In [9]: ## USE 'np.load' FUNCTION TO LOAD THE DATA
        ## USE DATA1.NPY AS THE SIGNAL ARRAY

        # YOUR CODE HERE
        data = np.load("data1.npy")
        for x in range(1, 25):
            gold_satellite = Gold_code_satellite(x)
            x, y = array_correlation(data, gold_satellite)
            print(max(x))

        # 4 7 13 19
        arr = [4, 7, 13, 19]
        for x in arr:
            gold_satellite = Gold_code_satellite(x)
            x, y = array_correlation(data, gold_satellite)
            plt.plot(y, x)
            plt.figure()
```

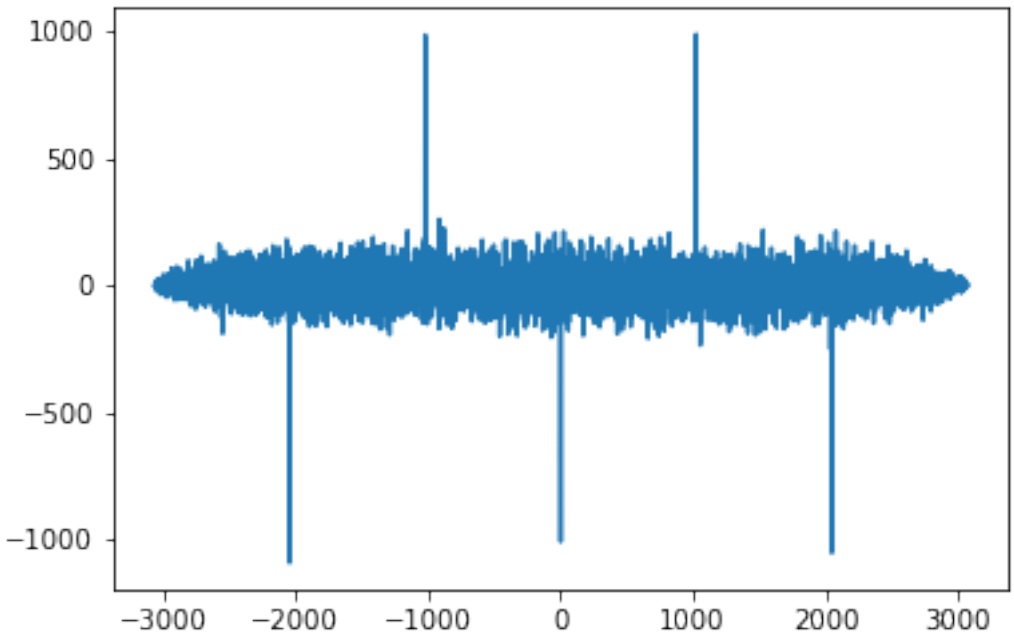
```
306.066515658
245.099653007
299.365433408
1029.71556601
292.718281052
279.928206052
```

```
1057.68837212
278.575821539
216.606272817
287.189517393
253.64845426
247.331691242
1084.14887121
265.069084198
243.774824523
268.518923482
282.272830391
242.824606907
987.991784761
246.872065605
243.393295861
280.289230731
246.266997465
268.470755486
```




1000
500
0
−500
−1000
−3000
−2000
−1000
0
1000
2000
3000


1000
500
0
−500
−1000
−3000
−2000
−1000
0
1000
2000
3000




```
<matplotlib.figure.Figure at 0x118cc1d68>
```

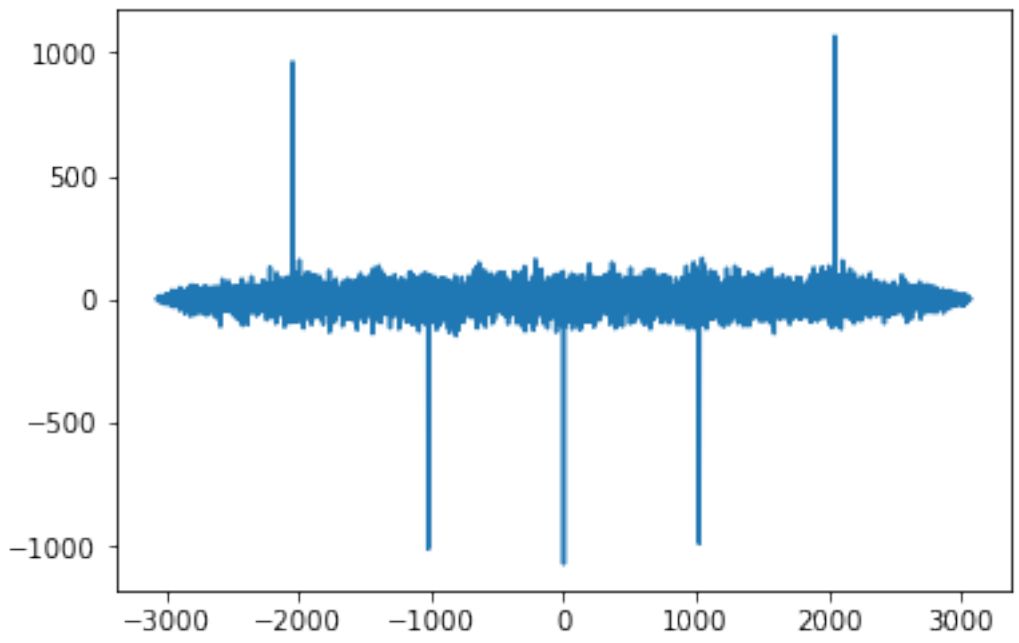
1.2.6 Part (f)

```
In [10]: ## USE DATA2.NPY AS THE SIGNAL ARRAY

         # YOUR CODE HERE
         data = np.load("data2.npy")
         for x in range(1, 25):
             gold_satellite = Gold_code_satellite(x)
             x, y = array_correlation(data, gold_satellite)
             print(max(x))

         #3
         gold_satellite = Gold_code_satellite(3)
         x, y = array_correlation(data, gold_satellite)
         plt.plot(y, x)
         plt.figure()
```

```
157.940844998
177.330855364
1065.46213414
164.914018878
151.638961506
188.315106572
```

```
160.721501081
165.141125454
166.902354093
167.810004862
154.59619961
162.591649795
154.740096351
179.257269302
168.611703473
169.144643862
150.688491882
155.144874353
167.769166479
160.924503569
165.64737499
153.485615358
205.436619607
173.461766874
```

Out[10]: <matplotlib.figure.Figure at 0x10df2e358>



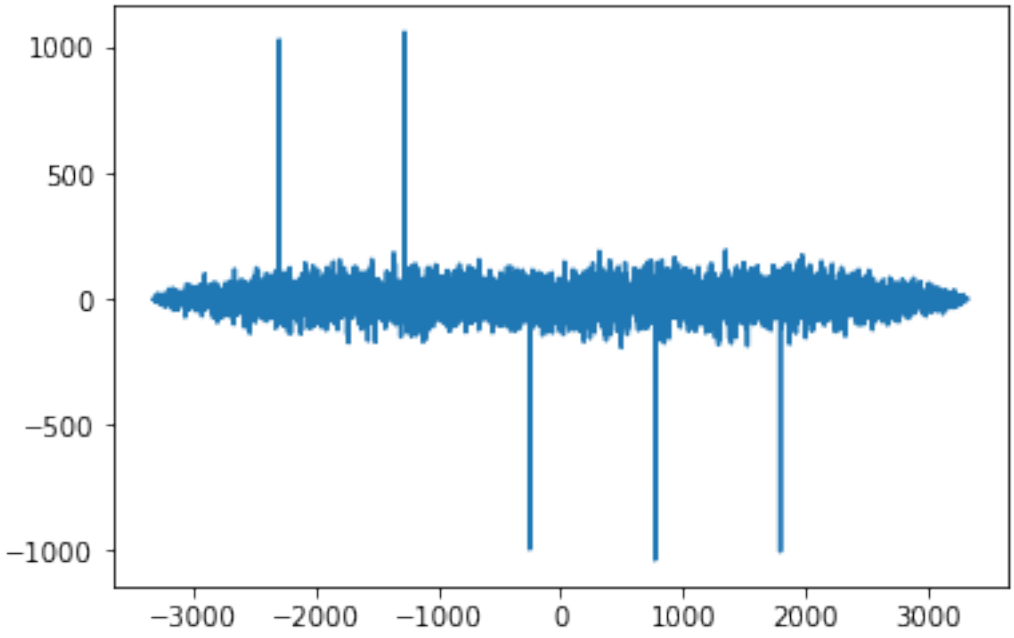
<matplotlib.figure.Figure at 0x10df2e358>

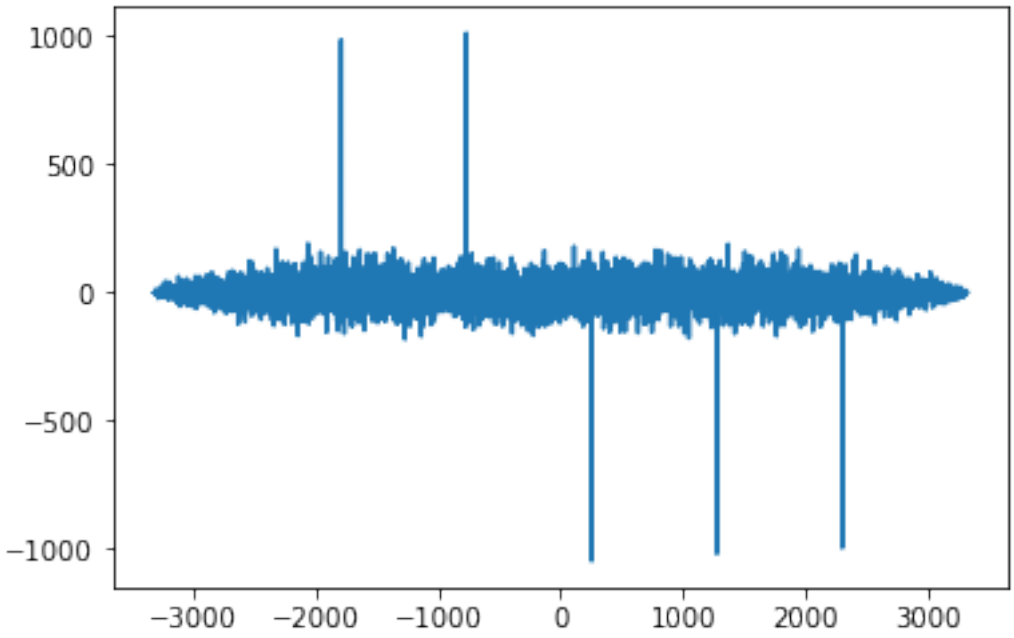
1.2.7 Part (g)

In [11]:
```
## USE DATA3.NPY AS THE SIGNAL ARRAY

# YOUR CODE HERE
data = np.load("data3.npy")
for x in range(1, 25):
    gold_satellite = Gold_code_satellite(x)
    x, y = array_correlation(data, gold_satellite)
    print(max(x))

#5 20

gold_satellite = Gold_code_satellite(5)
x_4, y_4 = array_correlation(data, gold_satellite)
plt.plot(y_4, x_4)
plt.figure()
print(np.argmax(x_4))

gold_satellite = Gold_code_satellite(20)
x_20, y_20 = array_correlation(data, gold_satellite)
plt.plot(y_20, x_20)
plt.figure()

print(np.argmax(x_20))
```

```
203.412060513
183.041201823
197.684194839
192.900391581
1062.8879826
196.667473199
196.837348295
194.558853532
184.680362336
212.502300883
196.245526004
205.581948421
190.460621463
202.327612241
201.688660911
202.418580195
205.51628162
203.954304576
224.216875585
1011.07973291
194.44925157
256.761024881
183.660513653
```

205.584553721
2045
2551





```
<matplotlib.figure.Figure at 0x1192d3860>
```

1.3 Question 3: Finding Signals in Noise

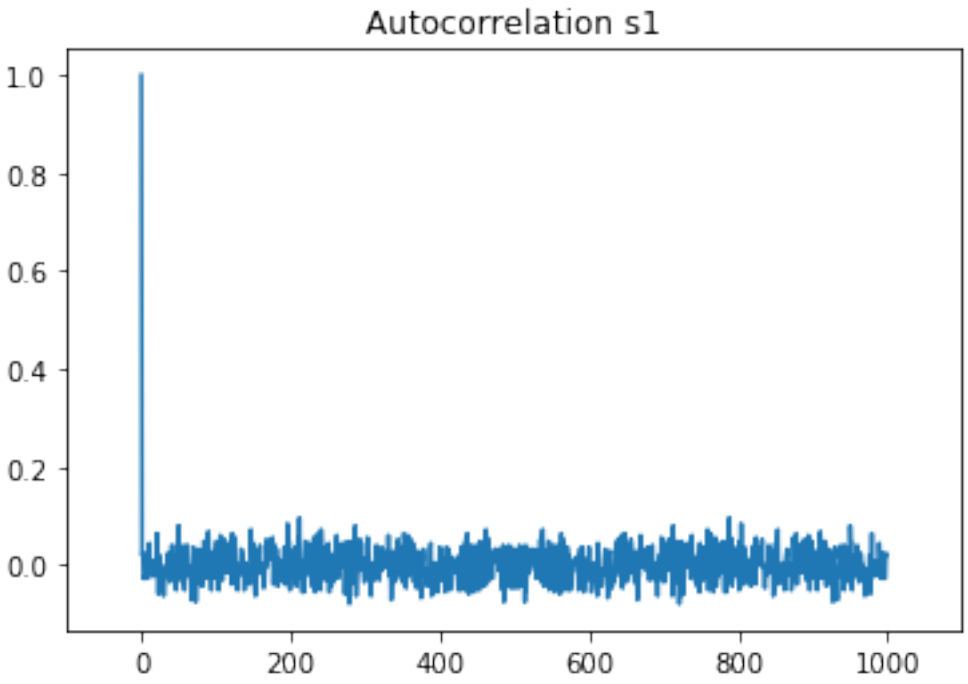
```
In [12]: %matplotlib inline
         import numpy as np
         import scipy as sp
         import scipy.linalg as la
         import pylab as plt
         import numpy.random

In [13]: N = 1000

         def rand_vector(n): # returns a random {+1, -1} vector of length n
             return np.random.randint(2, size=n)*2 - 1.0

         def rand_normed_vector(n): # returns a random normalized vector of length n
             x = rand_vector(n)
             return x / la.norm(x)

         def cross_corr(f, g):
             # returns the cross-correlation (a vector of all the inner products of 'g' with s
             C = la.circulant(f)
             corr = C.T.dot(g)
             return corr
```

1.3.1 Part (a)

```
In [14]: # Generate a random normalized vector for s1
         # Running this cell again will generate a new random vector
         s1 = rand_normed_vector(N)

         # Compute all the inner products of s1 with shifted versions of s1
         # (i.e., the cross-correlation of s1 with s1)
         corr = cross_corr(s1, s1)

         # The inner product <s1, s1^(1)> is:
         print(corr[1])

         # np.roll circularly shifts the signal,
         # so the above inner product could be computed as:
         print(np.dot(s1, np.roll(s1,1)))

         # Plot the autocorrelation:
         plt.title("Autocorrelation s1")
         plt.plot(corr)
```

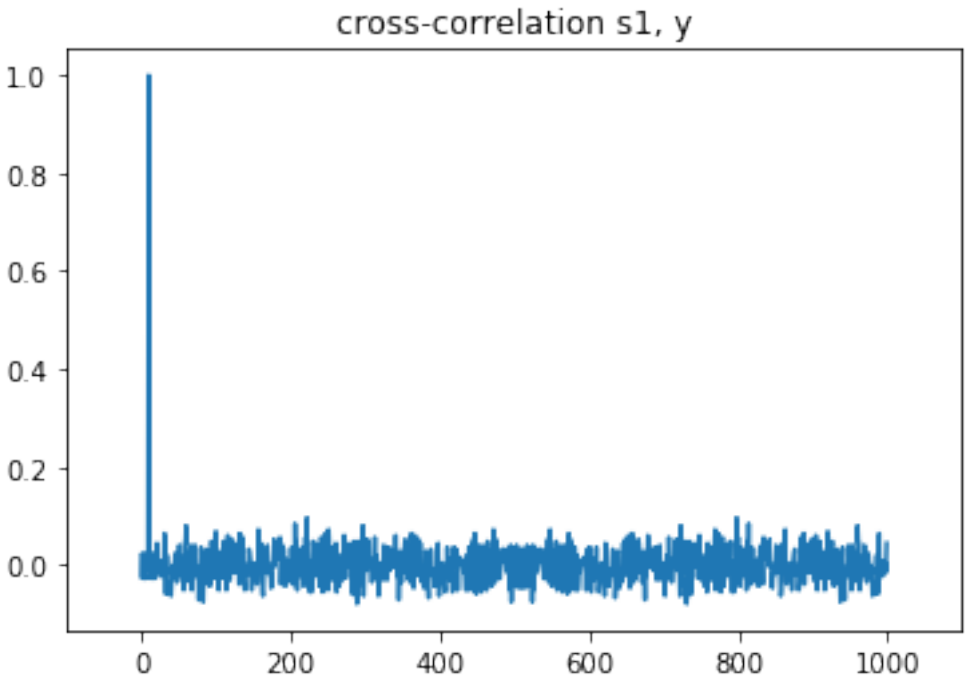
```
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])

plt.show()
```

```
0.02
0.02
```



1.3.2 Part (b)

```
In [15]: y = np.roll(s1, 10) # Received y = s1 shifted by 10

         # Compute the cross-correlation (all the inner products of y with shifted versions of
         corr = cross_corr(s1, y)

         # Plot
         plt.title("cross-correlation s1, y")
         plt.plot(corr)

         x1,x2,y1,y2 = plt.axis()
         plt.axis([x1-50,x2+50,y1,y2])
         plt.show()

         # Find the index of maximum correlation (inner product)
         print(np.argmax(corr))
```




```
10
```

1.3.3 Part (c)

```
In [16]: # Generate a random normalized vector for s1
         # and a random normalized vector for n
         # Running this cell again will generate new random vectors
         s1 = rand_normed_vector(N)
         n = rand_normed_vector(N)

         print(np.abs(np.dot(s1, n)))
```

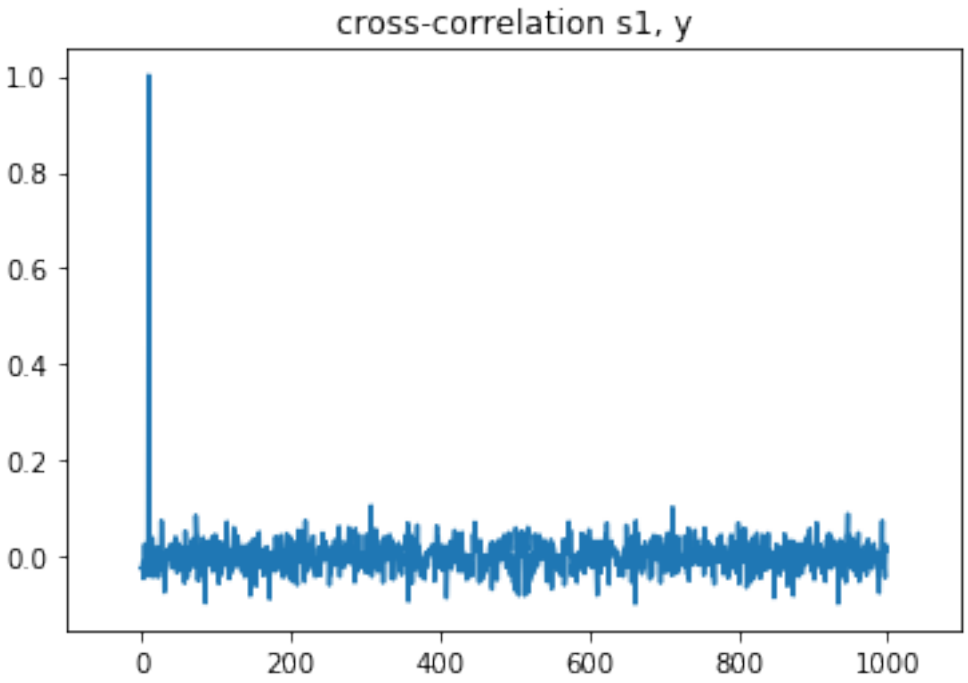
```
0.034
```

1.3.4 Part (d)

This is the code from part (b) but with the received signal $\vec{y}$, which is corrupted by noise.

```
In [17]: s1 = rand_normed_vector(N)
         n = rand_normed_vector(N)
         y = np.roll(s1, 10) + 0.1*n

         corr = cross_corr(s1, y)
```

```
plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
np.argmax(corr)
```



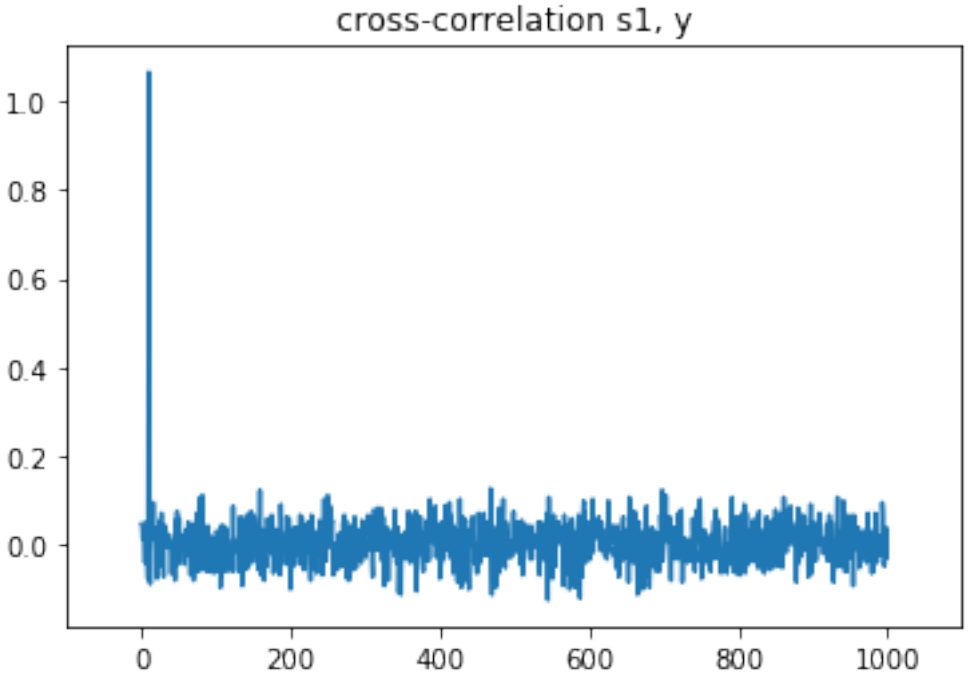
Out[17]: 10

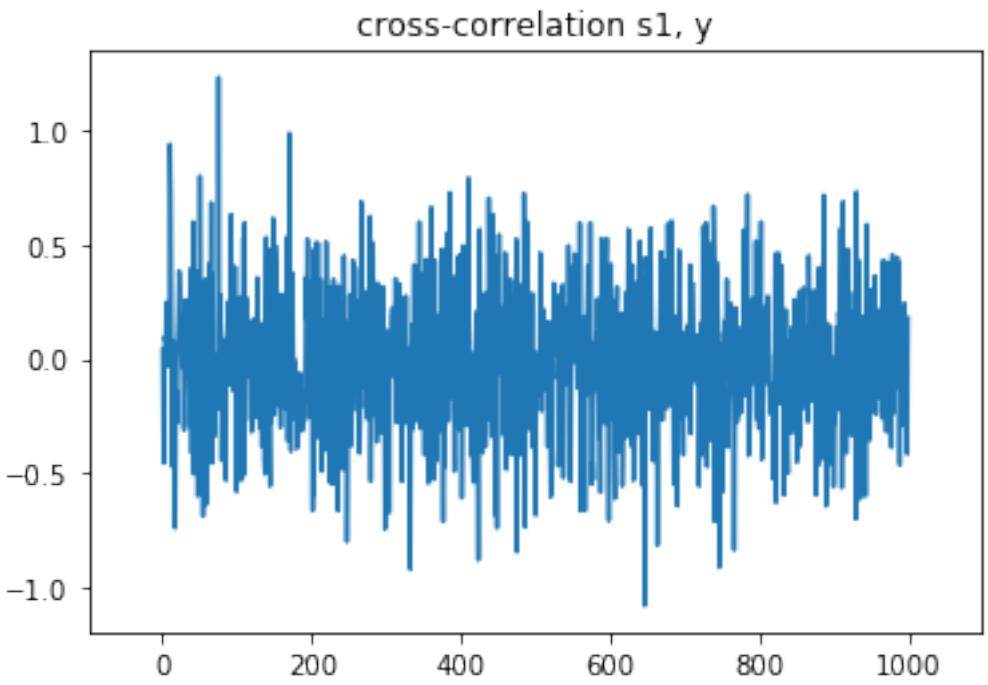
1.3.5 Part (e)

Copy the code provided for part (d), but modify it appropriately, so that the noise is higher. You should generate two cross-correlation plots, one for each noise level in the question. (You can just copy the code from part (d) twice.)

In [18]:
```
s1 = rand_normed_vector(N)
n = rand_normed_vector(N)
y = np.roll(s1, 10) + n

corr = cross_corr(s1, y)
```

```
plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
np.argmax(corr)

s1 = rand_normed_vector(N)
n = rand_normed_vector(N)
y = np.roll(s1, 10) + 10*n

corr = cross_corr(s1, y)

plt.title("cross-correlation s1, y")
plt.plot(corr)

x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

# Find the index of maximum correlation (inner product)
np.argmax(corr)
```



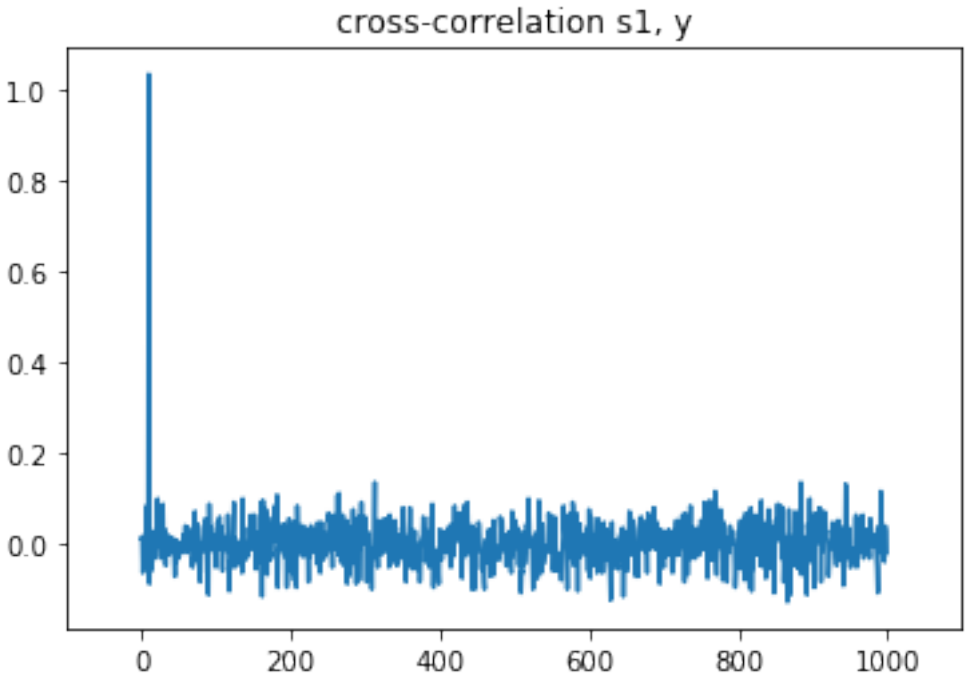


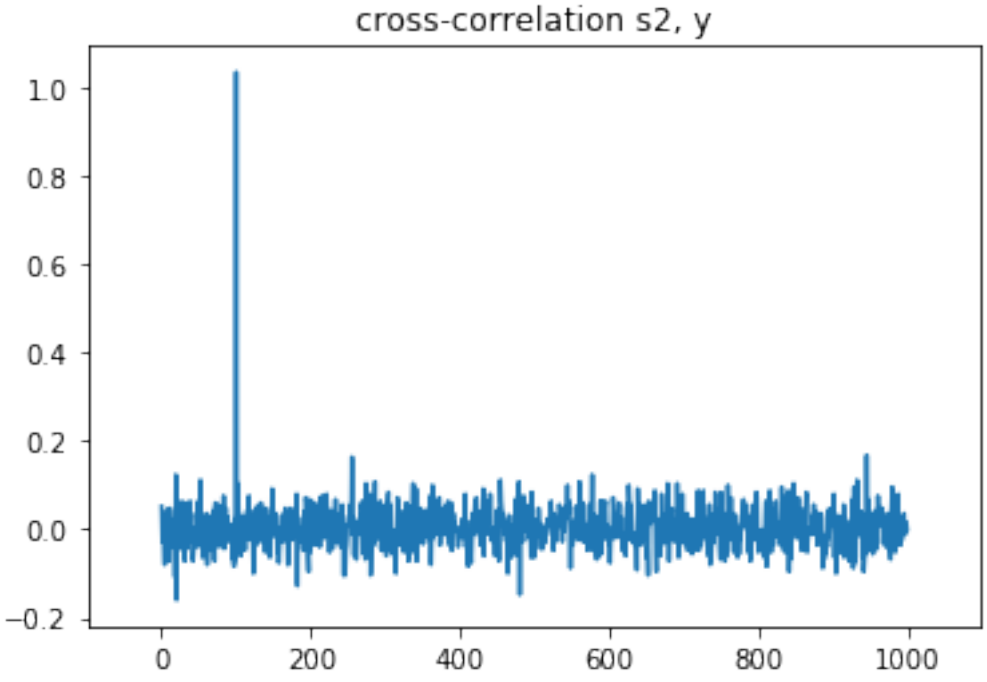
Out[18]: 75

1.3.6 Part (f)

```
In [19]: s1 = rand_normed_vector(N)
         s2 = rand_normed_vector(N)

         y = np.roll(s1, 10) + np.roll(s2, 100)

         # Compute cross-correlations:
         corr_s1_y = cross_corr(s1, y)
         corr_s2_y = cross_corr(s2, y)

         # Plot cross-correlations:
         plt.title("cross-correlation s1, y")
         plt.plot(cross_corr(s1, y))
         x1,x2,y1,y2 = plt.axis()
         plt.axis([x1-50,x2+50,y1,y2])
         plt.show()

         plt.title("cross-correlation s2, y")
         plt.plot(cross_corr(s2, y))
         x1,x2,y1,y2 = plt.axis()
         plt.axis([x1-50,x2+50,y1,y2])
         plt.show()
```

```
j = np.argmax(corr_s1_y) # find the first signal delay (max index of correlation)
k = np.argmax(corr_s2_y) # find the second signal delay
print(j, k)
```



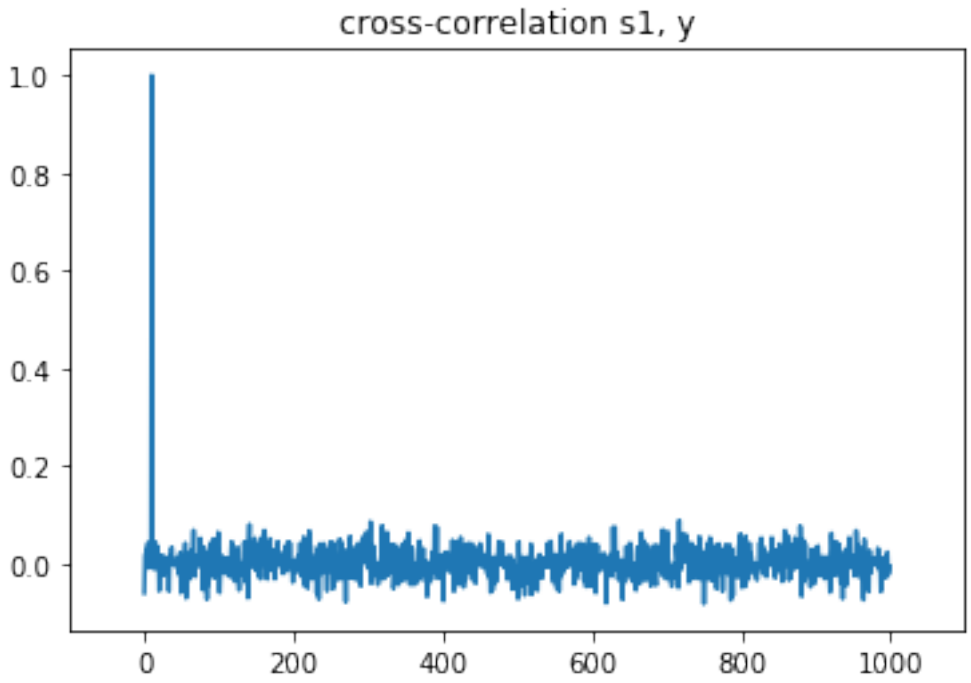


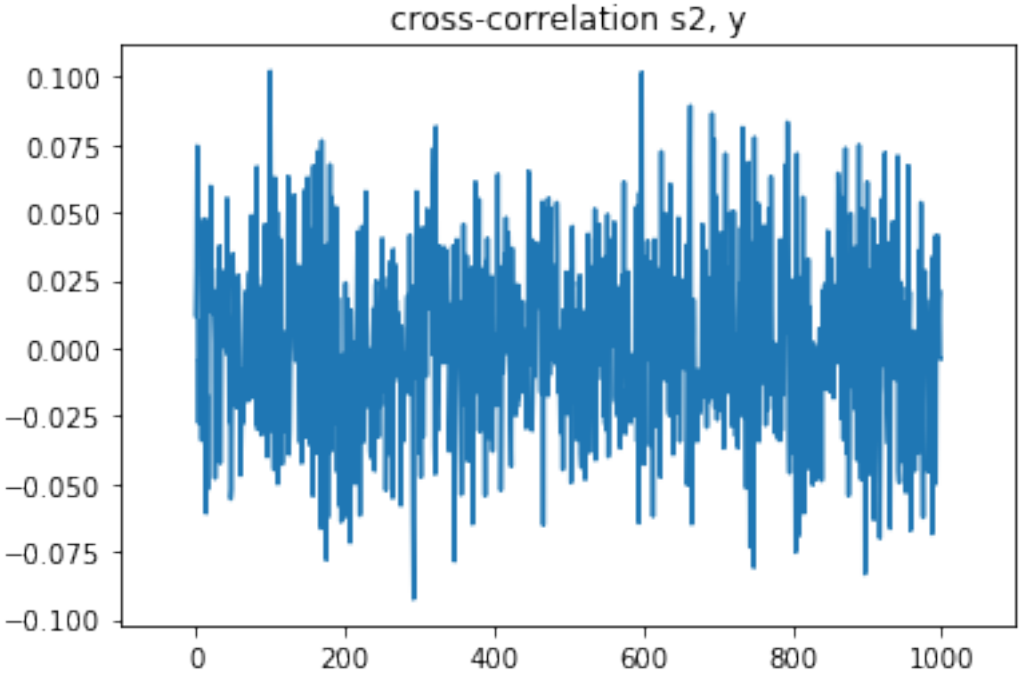
```
10 100
```

1.3.7 Part (g)

This is the same code as in part (f) but with a slight modification to how the received signal y generated. Run the below cell a few times to test for different choices of random signals.

```
In [20]: s1 = rand_normed_vector(N)
         s2 = rand_normed_vector(N)

         y = np.roll(s1, 10) + 0.1*np.roll(s2, 100)

         # Compute cross-correlations:
         corr_s1_y = cross_corr(s1, y)
         corr_s2_y = cross_corr(s2, y)

         # Plot cross-correlations:
         plt.title("cross-correlation s1, y")
         plt.plot(cross_corr(s1, y))
         x1,x2,y1,y2 = plt.axis()
         plt.axis([x1-50,x2+50,y1,y2])
         plt.show()

         plt.title("cross-correlation s2, y")
         plt.plot(cross_corr(s2, y))
         x1,x2,y1,y2 = plt.axis()
         plt.axis([x1-50,x2+50,y1,y2])
         plt.show()
```

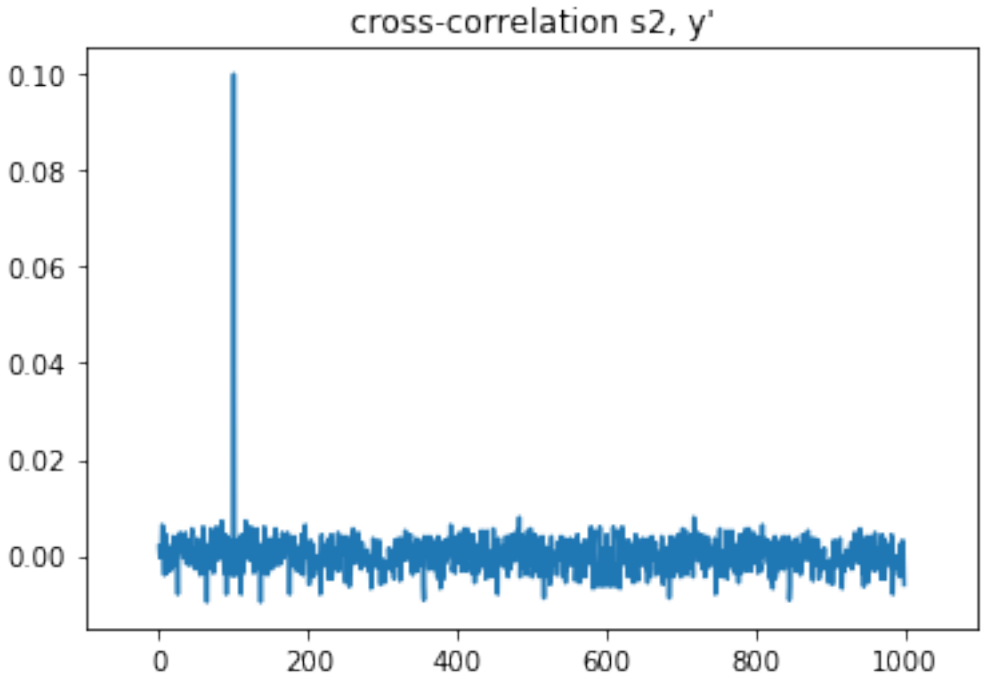

cross-correlation s1, y
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
600
800
1000


cross-correlation s2, y
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
0
200
400
600
800
1000

1.3.8 Part (h)

In [21]:
```
corr_s1_y = cross_corr(s1, y)
j = np.argmax(corr_s1_y) # find the first signal delay
print(j)

# Subtract out the contribution of the first signal
y_prime = y - np.roll(s1, j)

# Correlate the residual against the second signal
corr_s2_y = cross_corr(s2, y_prime)

# Plot
plt.title("cross-correlation s2, y'")
plt.plot(corr_s2_y)
x1,x2,y1,y2 = plt.axis()
plt.axis([x1-50,x2+50,y1,y2])
plt.show()

k = np.argmax(corr_s2_y) # find the second signal delay by looking at the index of ma
print(k)
```

10



100

1.3.9 Part (i)

```
In [22]: s1 = rand_normed_vector(N)
         s2 = rand_normed_vector(N)

         y = 0.7*np.roll(s1, 10) + 0.5*np.roll(s2, 100)

         corr_s1_y = cross_corr(s1, y)
         j = np.argmax(corr_s1_y) # find the first signal delay

         corr_s2_y = cross_corr(s2, y)
         k = np.argmax(corr_s2_y) # find the second signal delay

         print(j, k)

         # Once we have found the shifts, estimate the coefficients as inner products:
         a1 = np.dot(y, np.roll(s1, j))
         a2 = np.dot(y, np.roll(s2, k))

         print(a1, a2)
```

```
10 100
0.698 0.4972
```

1.3.10 Part (j)

This is the same code as in part (i) but with noise added to the received signal $\vec{y}$.

```
In [23]: s1 = rand_normed_vector(N)
         s2 = rand_normed_vector(N)
         n = rand_normed_vector(N)

         y = 0.7*np.roll(s1, 10) + 0.5*np.roll(s2, 100) + 0.1*n

         corr_s1_y = cross_corr(s1, y)
         j = np.argmax(corr_s1_y) # find the first signal delay

         corr_s2_y = cross_corr(s2, y)
         k = np.argmax(corr_s2_y) # find the second signal delay

         print(j, k)

         # Once we have found the shifts, estimate the coefficients as inner products:
         a1 = np.dot(y, np.roll(s1, j))
         a2 = np.dot(y, np.roll(s2, k))

         print(a1, a2)
```

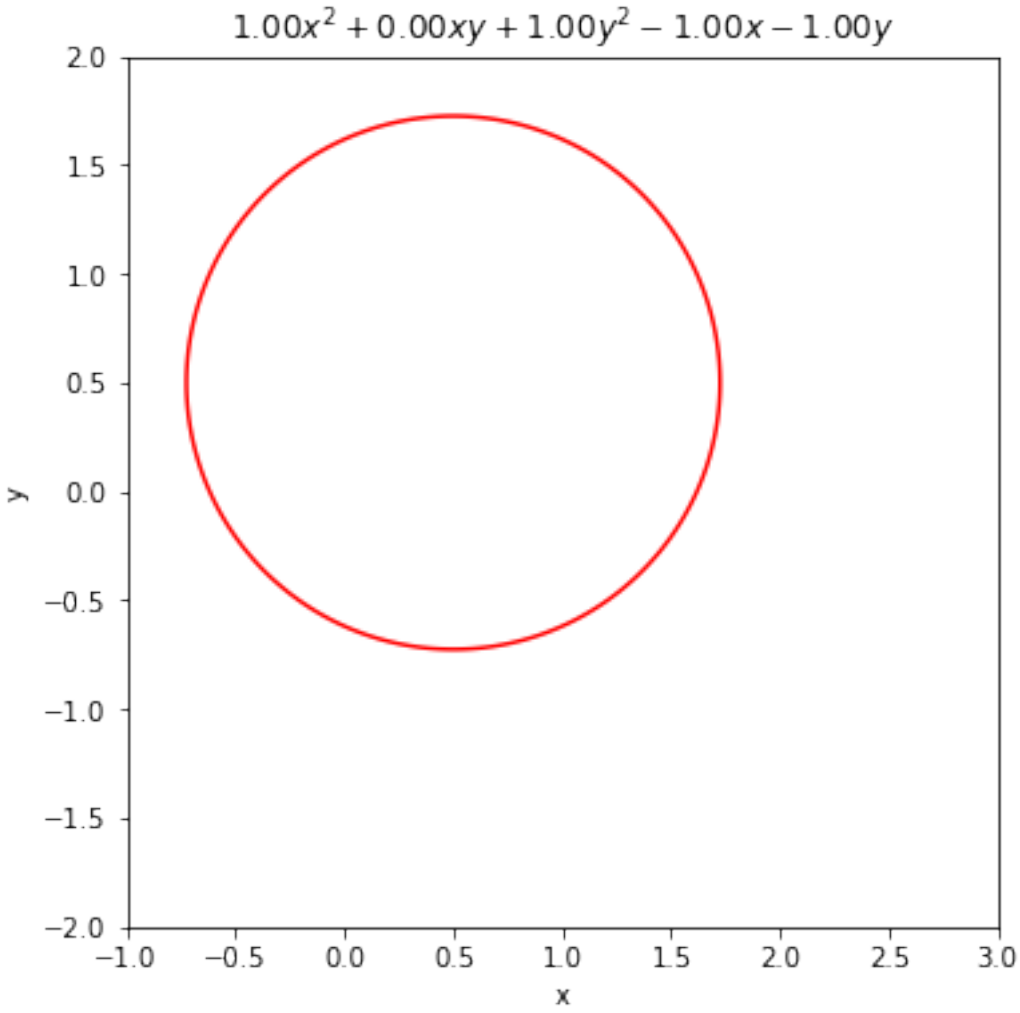
```
10 100
0.704 0.5096
```

1.4 Question 5: Image Analysis

```
In [24]: def plot_circle(a, d, e):
             """
             You can use this function to plot circles with parameters a,d,e.
             The parameters are described in the homework pdf.

             You can comment out the line that starts with `plt.title`
             because this makes assumptions regarding the title of your plot.
             """
             is_circle = d**2 + e**2 - 4*a > 0
             assert is_circle, "Not a circle"

             XLIM_LO = -1
             XLIM_HI = 3
             YLIM_LO = -2
             YLIM_HI = 2
             X_COUNT = 400
             Y_COUNT = 400

             x = np.linspace(XLIM_LO, XLIM_HI, X_COUNT)
             y = np.linspace(YLIM_LO, YLIM_HI, Y_COUNT)
             x, y = np.meshgrid(x, y)
             f = lambda x,y: a*(x**2 + y**2) + d*x + e*y

             c1 = plt.contour(x, y, f(x,y), [1], colors='r')
             plt.axis('scaled')
             plt.xlabel('x')
             plt.ylabel('y')
             plt.title(r'${:.2f}(x^2 + y^2) {:+.2f}x {:+.2f}y$'.format(a,d,e))

In [25]: def plot_ellipse(a, b, c, d, e):
             """
             You can use this function to plot ellipses with parameters a-e.
             The parameters are described in the homework pdf.

             You can comment out the line that starts with `plt.title`
             because this makes assumptions regarding the title of your plot.
             """
             is_ellipse = b**2 - 4*a*c < 0
             assert is_ellipse, "Not an ellipse"

             XLIM_LO = -1
             XLIM_HI = 3
```

```
    YLIM_LO = -2
    YLIM_HI = 2
    X_COUNT = 400
    Y_COUNT = 400

    x = np.linspace(XLIM_LO, XLIM_HI, X_COUNT)
    y = np.linspace(YLIM_LO, YLIM_HI, Y_COUNT)
    x, y = np.meshgrid(x, y)
    f = lambda x,y: a*x**2 + b*x*y + c*y**2 + d*x + e*y

    c1 = plt.contour(x, y, f(x,y), [1], colors='r')
    plt.axis('scaled')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(r'${:.2f}x^2 {:+.2f}xy {:+.2f}y^2 {:+.2f}x {:+.2f}y$'.format(a,b,c,d,e)
```

In [26]:
```
# Here is an example of plot_ellipse.
# This plots (x-1)**2 + (y-1)**2 = 1,
# which is a circle centered at (1,1).

plt.figure(figsize=(6,6))
plot_ellipse(1, 0, 1, -1, -1)
```



You may find plt.scatter useful for plotting the points.

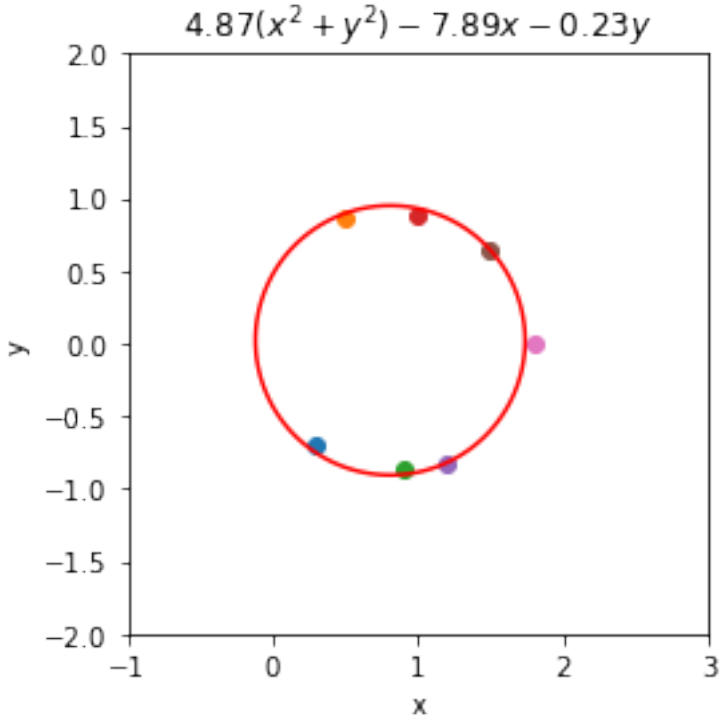
1.4.1 Part (c)

```
In [27]: # YOUR CODE HERE
         a = np.array([[0.5661, 0.3, -0.69],
                       [1.0069, 0.5, 0.87],
                       [1.5496, 0.9, -0.86],
                       [1.7744, 1, 0.88],
                        [2.1124, 1.2, -0.82],
                        [2.6593, 1.5, 0.64],
                        [3.24, 1.8, 0]])
         b = np.array([1, 1, 1, 1, 1, 1, 1])
         aTa = np.dot(np.transpose(a), a)
         aTb = np.dot(np.transpose(a), b)
         x = np.dot(np.linalg.inv(aTa), aTb)
```

```
print(x)

plot_circle(4.86902644, -7.88536506, -0.226141826)

b_hat = np.dot(a, x)
b_minus_b_hat = np.subtract(b, b_hat)
print(np.linalg.norm(b_minus_b_hat) / 7)

plt.scatter(0.3, -0.69)
plt.scatter(0.5, 0.87)
plt.scatter(0.9, -0.86)
plt.scatter(1, 0.88)
plt.scatter(1.2, -0.82)
plt.scatter(1.5, 0.64)
plt.scatter(1.8, 0)
```

```
[ 4.86902644 -7.88536506 -0.22614826]
0.13749580546
```

Out[27]: <matplotlib.collections.PathCollection at 0x118d07c18>

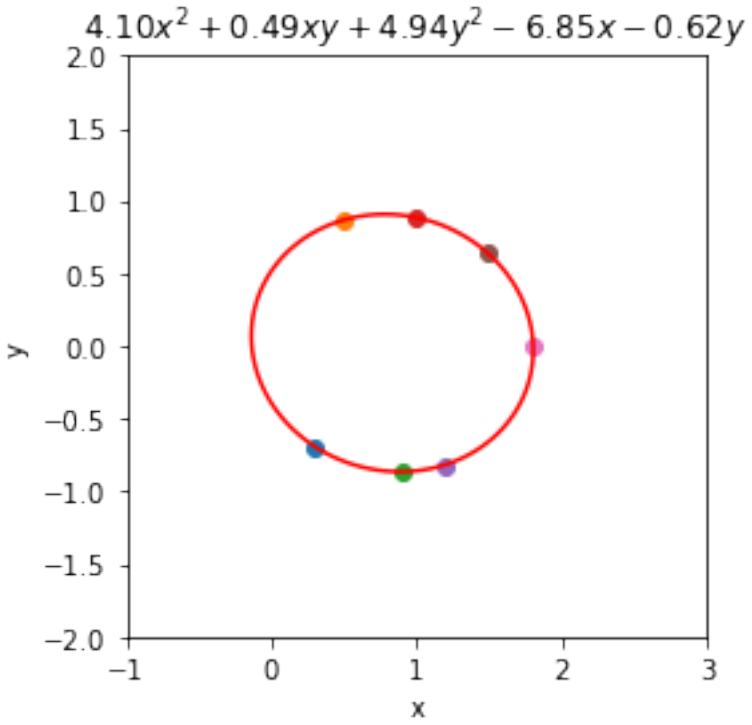


1.4.2 Part (d)

In [28]:
```
# YOUR CODE HERE
a = np.array([
    [0.09, -0.207, 0.4761, 0.3, -0.69],
    [0.25, 0.435, 0.7569, 0.5, 0.87],
    [0.81, -0.774, 0.7396, 0.9, -0.86],
    [1, 0.88, 0.7744, 1, 0.88],
    [1.44, -0.984, 0.6724, 1.2, -0.82],
    [2.25, 0.96, 0.4096, 1.5, 0.64],
    [3.24, 0, 0, 1.8, 0]

])

b = np.array([1, 1, 1, 1, 1, 1, 1])
aTa = np.dot(np.transpose(a), a)
aTb = np.dot(np.transpose(a), b)
x = np.dot(np.linalg.inv(aTa), aTb)
print(x)

plot_ellipse(4.10382951, 0.48711384, 4.93938449, -6.85032284, -0.62259529)

b_hat = np.dot(a, x)
b_minus_b_hat = np.subtract(b, b_hat)
print(np.linalg.norm(b_minus_b_hat) / 7)

plt.scatter(0.3, -0.69)
plt.scatter(0.5, 0.87)
plt.scatter(0.9, -0.86)
plt.scatter(1, 0.88)
plt.scatter(1.2, -0.82)
plt.scatter(1.5, 0.64)
plt.scatter(1.8, 0)
```

```
[ 4.10382951  0.48711384  4.93938449 -6.85032284 -0.62259259]
0.0128538290872
```

Out[28]: <matplotlib.collections.PathCollection at 0x11895b668>



In [ ]: