EE 16A: Homework 12

David J. Lee
3031796951
dssd1001@berkeley.edu

December 6, 2016

1. **Worked With...**

   Ilya (3031806896), James Zhu (3031793129)
   I worked alone on Friday morning, then met up with Ilya and James to discuss on Saturday afternoon.

2. **Mechanical Gram-Schmidt**

(a) *Find U, whose columns form orthonormal basis for col space of V.*

$$V = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Find $\vec{u_1}$:
Since $\vec{v_1}$ is already a unit vector,

$$\vec{u_1} = \vec{v_1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

Find $\vec{u_2}$:
A vector $\vec{e_2}$ orthogonal to $\vec{u_1}$ is:

$$\begin{aligned} \vec{e_2} &= \vec{v_2} - proj_{\vec{u_1}} \vec{v_2} \\ &= \vec{v_2} - \langle \vec{v_2}, \vec{u_1} \rangle \vec{u_1} \\ &= \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}^T - \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \end{bmatrix}^T \end{aligned}$$

Thus the orthonormal $\vec{u_2}$ is:

$$\begin{aligned} \vec{u_2} &= \frac{\vec{e_2}}{\|\vec{e_2}\|} \\ &= \begin{bmatrix} 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 & 0 \end{bmatrix}^T \end{aligned}$$

Find $\vec{u_3}$:
Similarly,

$$\begin{aligned} \vec{e_3} &= \vec{v_3} - proj_{span(\vec{u_1}, \vec{u_2})} \vec{v_3} \\ &= \vec{v_3} - proj_{\vec{u_1}} \vec{v_3} - proj_{\vec{u_2}} \vec{v_3} \\ &= \vec{v_3} - \langle \vec{v_3}, \vec{u_1} \rangle \vec{u_1} - \langle \vec{v_3}, \vec{u_2} \rangle \vec{u_2} \\ &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix}^T - \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T - \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \end{bmatrix}^T \\ &= \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T \end{aligned}$$

Then $\vec{u_3}$ is:

$$\begin{aligned} \vec{u_3} &= \frac{\vec{e_3}}{\|\vec{e_3}\|} \\ &= \begin{bmatrix} 0 & 0 & 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^T \end{aligned}$$

Putting it together, we get

$$\boxed{U = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{2}} \\ 0 & 0 & \frac{1}{\sqrt{2}} \end{bmatrix}}$$

Now, show that projecting a vector $\vec{w} = \begin{bmatrix} 1 & -1 & 0 & -1 & 1 \end{bmatrix}^T$ onto U and onto V are the same:
We can easily see that

$$V(V^TV)^{-1}V^T\vec{w} = U(U^TU)^{-1}U^T\vec{w} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{bmatrix}^T$$

b) Orthonormalise matrix V :

$$u_1 = v_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

$$u_2 = v_1 - \frac{\langle v_2, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ -1 \\ -1 \end{bmatrix}$$

$$u_3 = v_3 - \frac{\langle v_3, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 - \frac{\langle v_3, u_2 \rangle}{\langle u_2, u_2 \rangle} u_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Now verify $V(V^TV)^{-1}V^Tw = U(U^TU)^{-1}U^Tw$:

$$U^Tw = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad U^TU = \begin{bmatrix} 3 & 0 \\ 0 & 4 \end{bmatrix}, \quad (U^TU)^{-1} \begin{bmatrix} 1/3 & 0 \\ 0 & 1/4 \end{bmatrix}$$

$$U(U^TU)^{-1}U^Tw = \begin{bmatrix} 1/2 \\ 1/2 \\ 0 \\ -1/2 \\ -1/2 \end{bmatrix} = V(V^TV)^{-1}V^Tw$$

c) 1)

$$V = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{2} & 0 \\ 0 & 1/\sqrt{2} & 0 \\ 0 & 0 & 1/\sqrt{2} \\ 0 & 0 & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & \frac{2}{\sqrt{2}} & \frac{2}{\sqrt{2}} \\ 0 & 0 & \frac{2}{\sqrt{2}} \end{bmatrix}$$

$$= [q_1 \ q_2 \ q_3] \begin{bmatrix} v_1 q_1 & v_2 q_1 & v_3 q_1 \\ 0 & v_2 q_2 & v_3 q_2 \\ 0 & 0 & v_3 q_3 \end{bmatrix} = QR$$

2)

$$V = \begin{bmatrix} 1 & 1 & -1 \\ 0 & 1 & -1 \\ 1 & 0 & 0 \\ 1 & -1 & 1 \\ 0 & -1 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{3}} & \frac{1}{2} \\ 0 & \frac{1}{2} \\ \frac{1}{\sqrt{3}} & 0 \\ \frac{1}{\sqrt{3}} & -\frac{1}{2} \\ 0 & -\frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{3}} & 0 & 0 \\ 0 & 2 & -2 \end{bmatrix} = QR$$

3) a) $b^k = \ln\left(\frac{\ln(1-p)}{\ln 0.95}\right) + 26.1931$

$A_1^k = \ln(AGE^k)$

$A_2^k = \ln(TC^k)$

$A_3^k = \ln(HDL^k)$

$A_4^k = \ln(SBP^k)$

$A_5^k = DIA$

$A_6^k = SMK$

$\boxed{b^k = x^T A^k}$

b) $b = \begin{bmatrix} b_1 \\ \vdots \\ b^n \end{bmatrix}$, $A = \begin{bmatrix} A_1^1 & A_2^1 & A_3^1 & A_4^1 & A_5^1 & A_6^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ A_1^n & A_2^n & A_3^n & A_4^n & A_5^n & A_6^n \end{bmatrix}$

c) $\bar{x} = \begin{bmatrix} 2.322 \\ 1.24 \\ -0.668 \\ 2.685 \\ 0.705 \\ 0.513 \end{bmatrix}$

d) $E^2 = |b - \hat{b}|^2 = \boxed{0.695}$

e) $\boxed{\text{iPython}}$

f) $\boxed{\text{Linear}}$

g) Perturbation increases error to 77.205

h) $\boxed{\text{iPython}}$

i) $\boxed{\text{iPython}}$

All also in iPython

4) ~~a~~ iPython.

5) a) $\boxed{\text{Yes}}$, $\boxed{\text{Yes}}$

b) $\boxed{\text{No.}}$, consider $\vec{a} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\vec{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$,

let $\left.\begin{aligned} \vec{x}_1 &= \vec{a} + \vec{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\ \vec{x}_2 &= 2\vec{a} + 2\vec{b} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \end{aligned}\right\} \vec{x}_1 \not\perp \vec{x}_2.$

c) $M = [\vec{x}_1, \vec{x}_2]$ $\quad$ $V = QR$

$= \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}$

$u_1 = m_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, $u_2 = m_1 - \frac{\langle m_2, u_1 \rangle}{\langle u_1, u_1 \rangle} u_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - 2\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$

$\therefore \cancel{\boxed{Q = \begin{bmatrix} 1/\sqrt{2} & -\frac{1}{\sqrt{2}} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}}}$

One example of $Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ LOL,

but this is just a 2D example, in the problem of troll sound, the vectors would ~~not~~ necessarily be 2-dimensional, they could be 1000-dimensions. Just something to keep in mind.

d-e) iPython.

6) Q: Given a set of three vectors $\{\vec{v}_1, \vec{v}_2, \vec{v}_3\}$ that is linearly independent of each other, ~~and all are of unit length~~, find an orthonormal set of vectors $\{\vec{w}_1, \dots, \vec{w}_n\}$.

Solution:

Step 1 Find unit vector $\vec{w}_1$ such that $\text{span}(\{\vec{w}_1\}) = \text{span}(\{\vec{v}_1\})$.

$$\vec{w}_1 = \frac{\vec{v}_1}{\|\vec{v}_1\|}.$$

Step 2 Find $\vec{w}_2$ s.t. $\text{span}(\{\vec{w}_1, \vec{w}_2\}) = \text{span}(\{\vec{v}_1, \vec{v}_2\})$ and orthogonal to $\vec{w}_1$.

$$\vec{e}_2 = \vec{v}_2 - (\vec{v}_2^T \vec{w}_1)\vec{w}_1$$

$$\vec{w}_2 = \frac{\vec{e}_2}{\|\vec{e}_2\|}$$

Step 3 Find $\vec{w}_3$

$$\vec{e}_3 = \vec{v}_3 - (\vec{v}_3^T \vec{w}_2)\vec{w}_2 - (\vec{v}_3^T \vec{w}_1)\vec{w}_1 ,$$

$$\vec{w}_3 = \frac{\vec{e}_3}{\|\vec{e}_3\|}$$

And we're done generalizing.

hw12

November 22, 2016

1 Problem Set 11 Code

```
In [144]: %pylab inline
          import numpy as np
          import matplotlib.pyplot as plt
```

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

2 The Framingham Risk Score Revisited

2.1 Part a

```
In [145]: # Importing medical data
          import scipy.io

          # LOADS IN THE MEDICAL DATA IN THE FORM OF A PYTHON DICTIONARY.
          # Data credit: CDC http://www.cdc.gov/nchs/nhanes.htm
          data = scipy.io.loadmat('CVDdata.mat')

          #UNPACKING DATA INTO COLUMN VECTORS
          AGE = data['AGE']
          TC = data['TC']
          HDL = data['HDL']
          SBP = data['SBP']
          DIA = data['DIABETIC']
          SMK = data['SMOKER']
          p = data['pNoisy']

          # Write expressions for b, A1, A2, A3, A4, A5, A6
          # It will help to use the identity log_n(z) = log(z)/log(n)

          b = np.log(np.log(1 - p)/np.log(0.95)) + 26.1931

          A1 = np.log(AGE)
          A2 = np.log(TC)
          A3 = np.log(HDL)
          A4 = np.log(SBP)
          A5 = DIA
          A6 = SMK
```

2.2 Part b

```
In [146]: # Write expressions for b and A
          # the function np.hstack will be helpful for constructing A
```

```
b = np.array(b.ravel())
A = np.hstack((A1,A2,A3,A4,A5,A6))
```

2.3 Part c

```
In [147]: # Write an expression for xhat

          xhat = np.dot(np.linalg.inv(np.dot(A.T, A)), np.dot(A.T, b))

          print("The estimated values for x:" + str(xhat.T))
```

```
The estimated values for x:[ 2.34338912  1.2405098  -0.6692395   2.68521474  0.70530453  0.5132915 ]
```

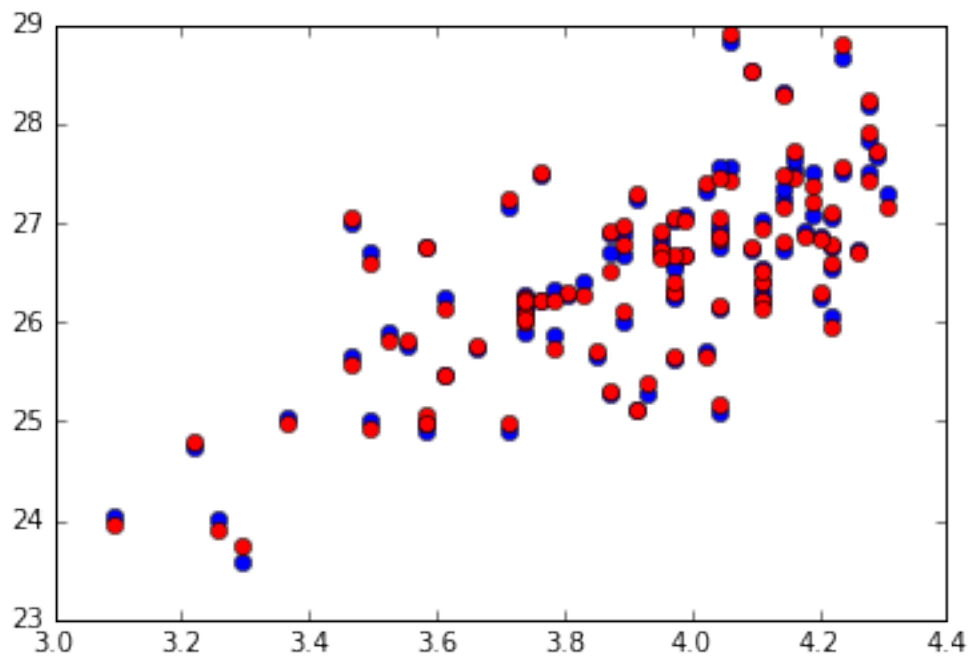
2.4 Part d

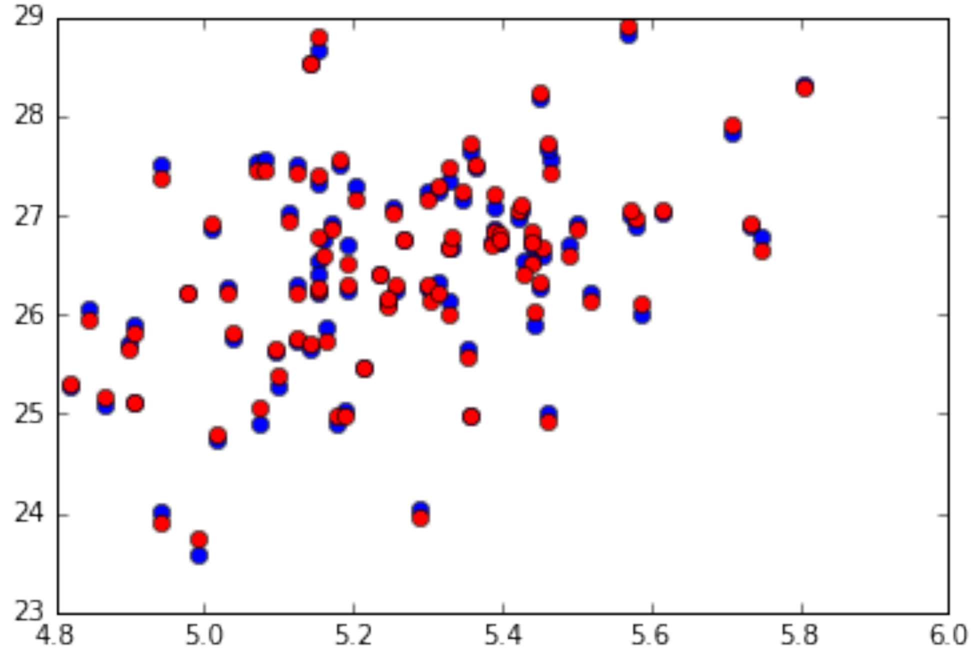
```
In [148]: # the model estimate bhat, and the squared error e2
          bhat = np.dot(A, xhat)
          e2 = np.linalg.norm(bhat - b)**2

          print("E^2 = " + str(e2))
```

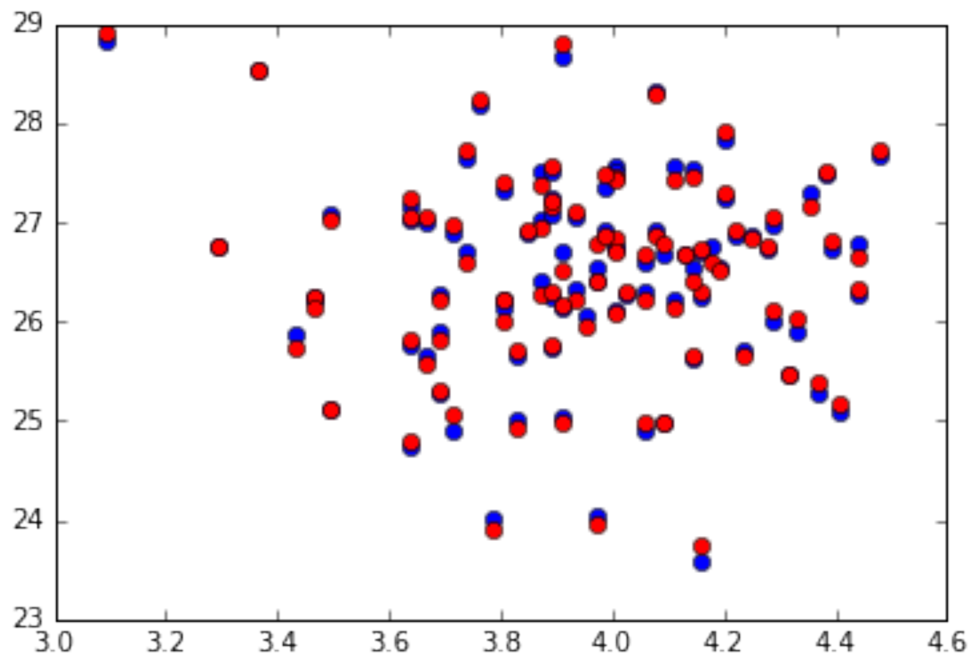
```
E^2 = 0.695069973457
```

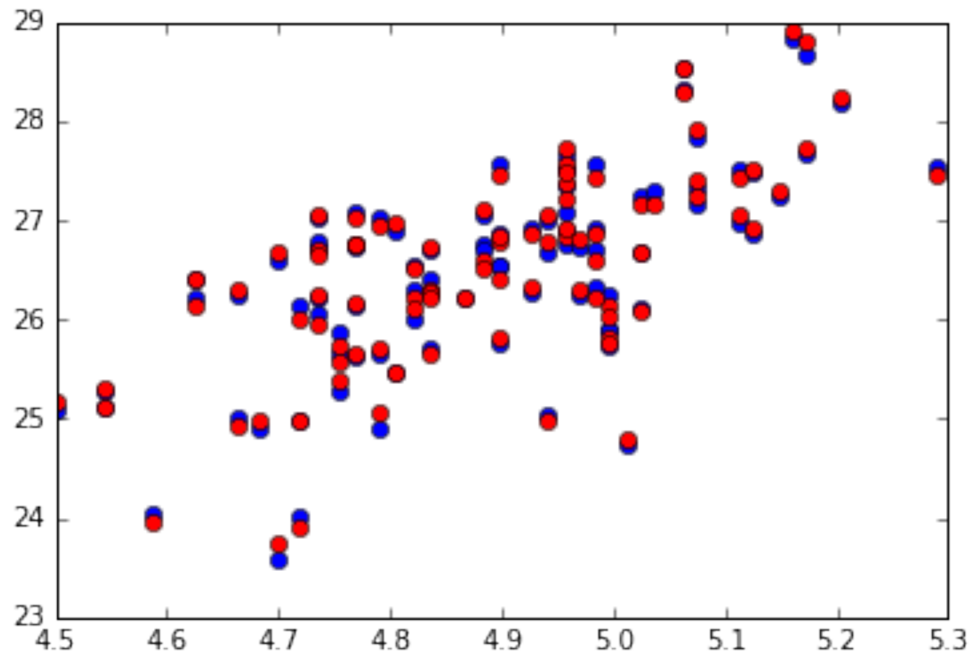
2.5 Part e

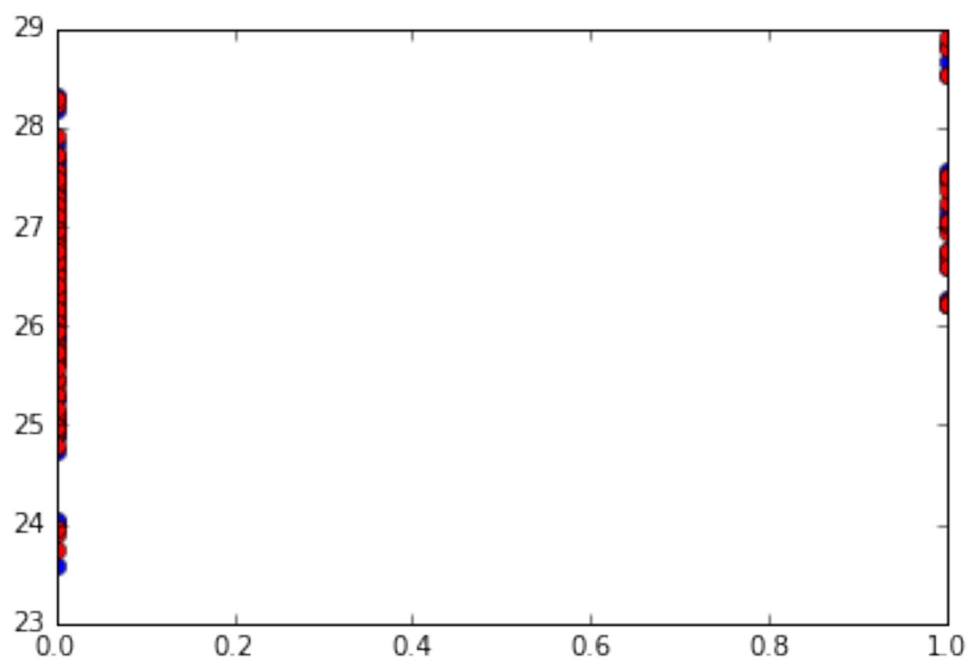
```
In [151]: # Linear plots, pick an index below (0,1,2,etc). This code will plot b and bhat vs Ai
          for i in range(len(A[0])):
              plt.plot(A[:,i],b,'ob')
              plt.plot(A[:,i],bhat,'or')
              plt.show()
```

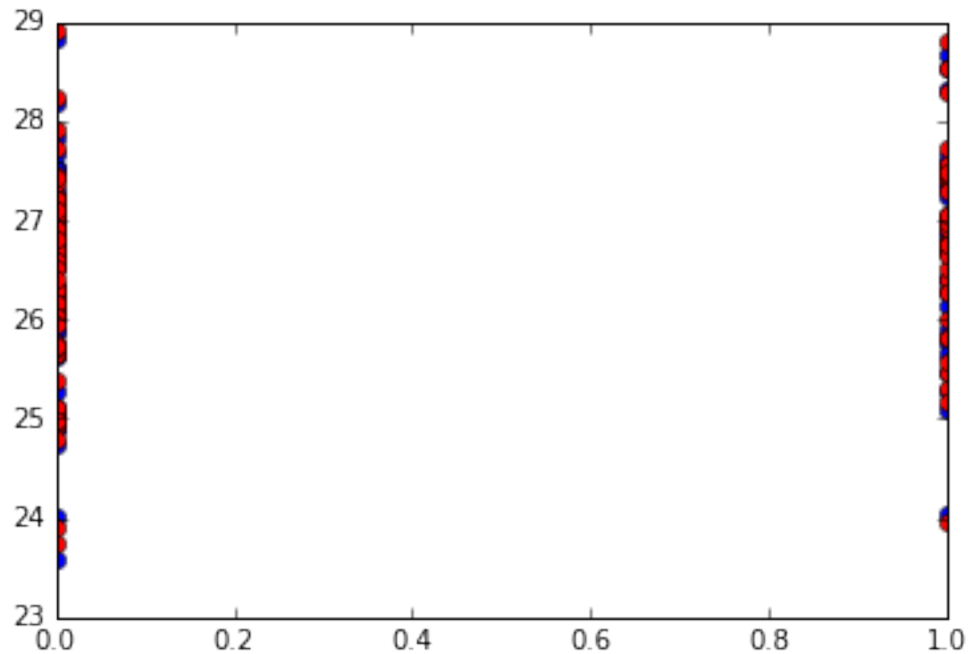



29
28
27
26
25
24
23
4.8
5.0
5.2
5.4
5.6
5.8
6.0


29
28
27
26
25
24
23
3.0
3.2
3.4
3.6
3.8
4.0
4.2
4.4
4.6


29
28
27
26
25
24
23
4.5
4.6
4.7
4.8
4.9
5.0
5.1
5.2
5.3

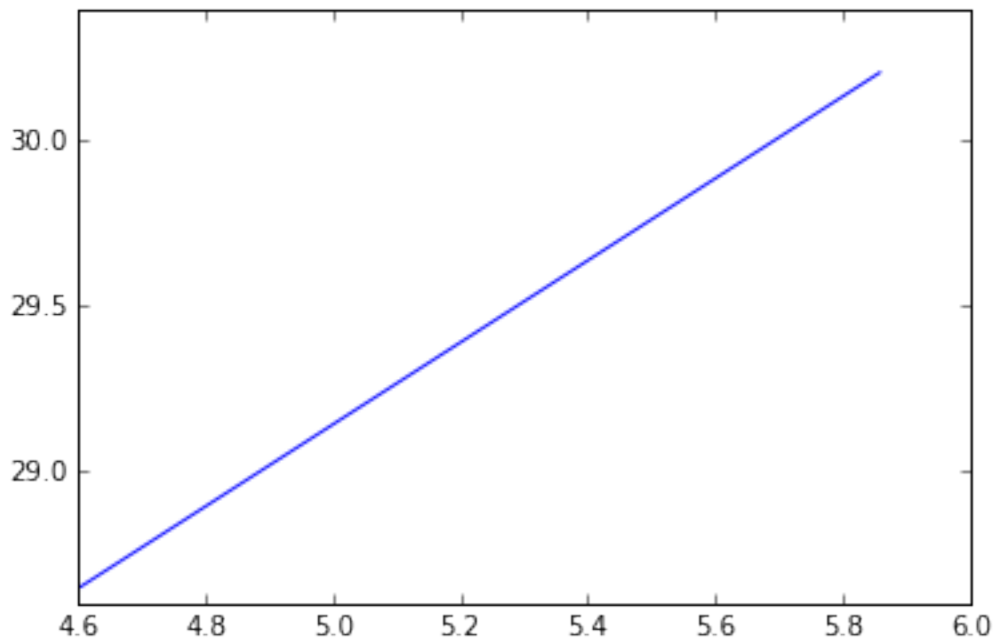

29
28
27
26
25
24
23
0.0
0.2
0.4
0.6
0.8
1.0



2.6 Part f

In [152]:
```
# Here are the values for the test plot
age_test = 55
tc_test_vector = np.linspace(100,350,(350-100+1))
hdl_test = 25
sbp_test = 220
dia_test = 1
smk_test = 1

A2_test = np.zeros(tc_test_vector.size)
b_test = np.zeros(tc_test_vector.size)

for ind in range(tc_test_vector.size):
    tc_test = tc_test_vector[ind];
    # Use the values for age_test, tc_test, hdl_test, sbp_test, dia_test
    # and smk_test to calculate the next value for b_test (y axis value)
    # and A2_test (x_axis value)
    a = np.array([np.log(age_test), np.log(tc_test), np.log(hdl_test), np.log(sbp_test), dia_
    b_test[ind] = np.dot(a, xhat)
    A2_test[ind] = np.log(tc_test)

plt.plot(A2_test,b_test,'-b')
```

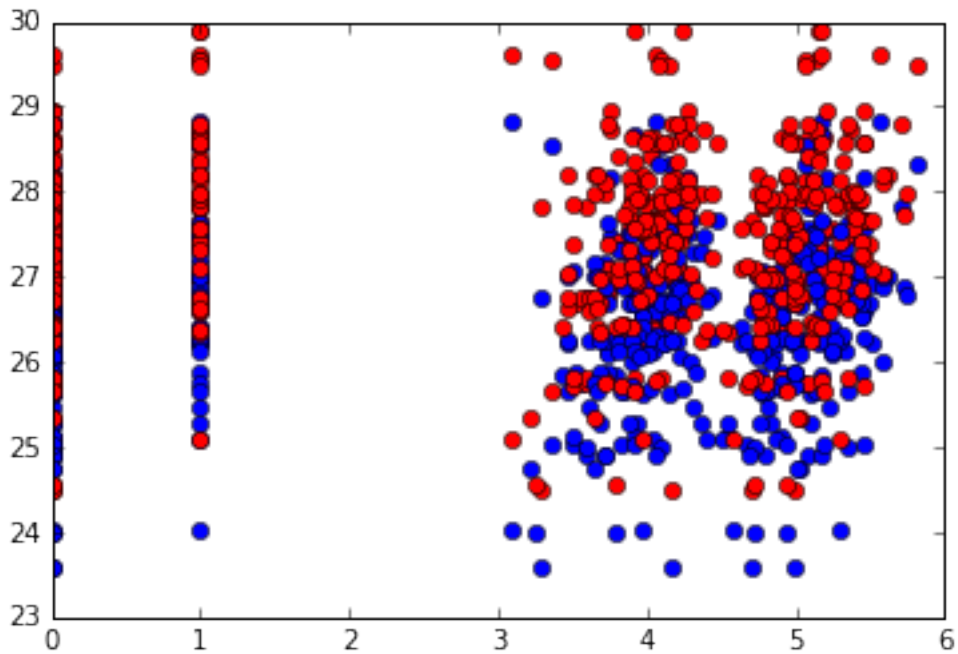
Out[152]: [<matplotlib.lines.Line2D at 0x10a18f048>]



2.7 Part g

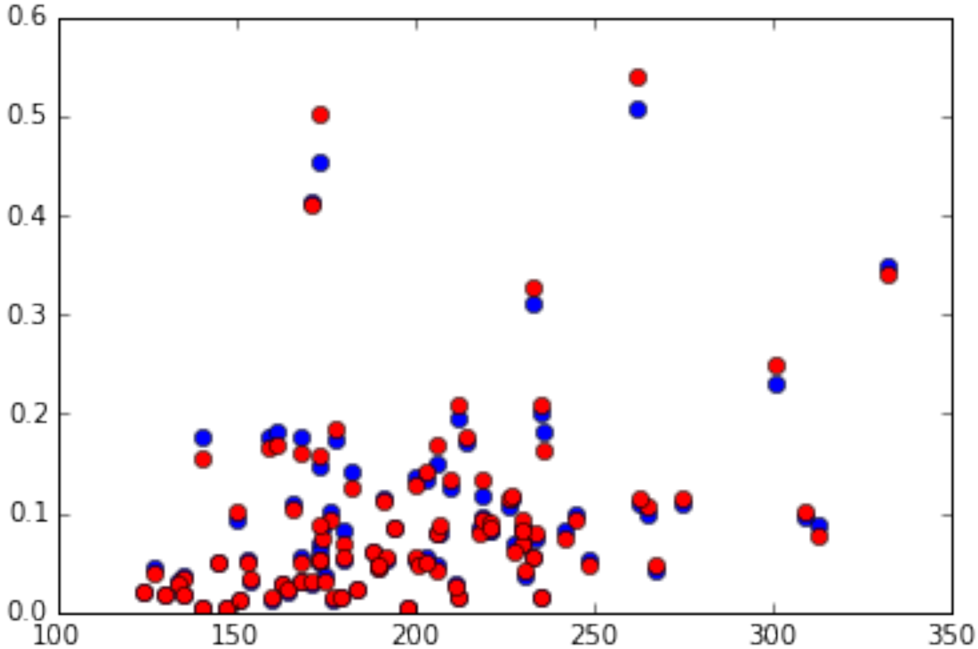
```
In [162]: # Perturb xhat from the solution above, store into x_perturbed and replot.

          # Use the following example expression with different pertrubations.
          x_perturbed = xhat+np.array([.1, 0.2, 0.2, -0.3, 0.1, 0.32])

          # What are the new estimated b values in terms of x_perturbed?
          b_perturbed = np.dot(A,x_perturbed)

          # Plot again
          for i in range(len(A[0])):
              plt.plot(A[:,i],b,'ob')
              plt.plot(A[:,i],b_perturbed,'or')

          # What is the new sum of squared errors (after perturbing)?
          e2_perturbed = np.linalg.norm(b_perturbed - b)**2

          print("E^2 after perturbing" + str(e2_perturbed))
```

```
E^2 after perturbing:77.3312748655
```



2.8 Part h

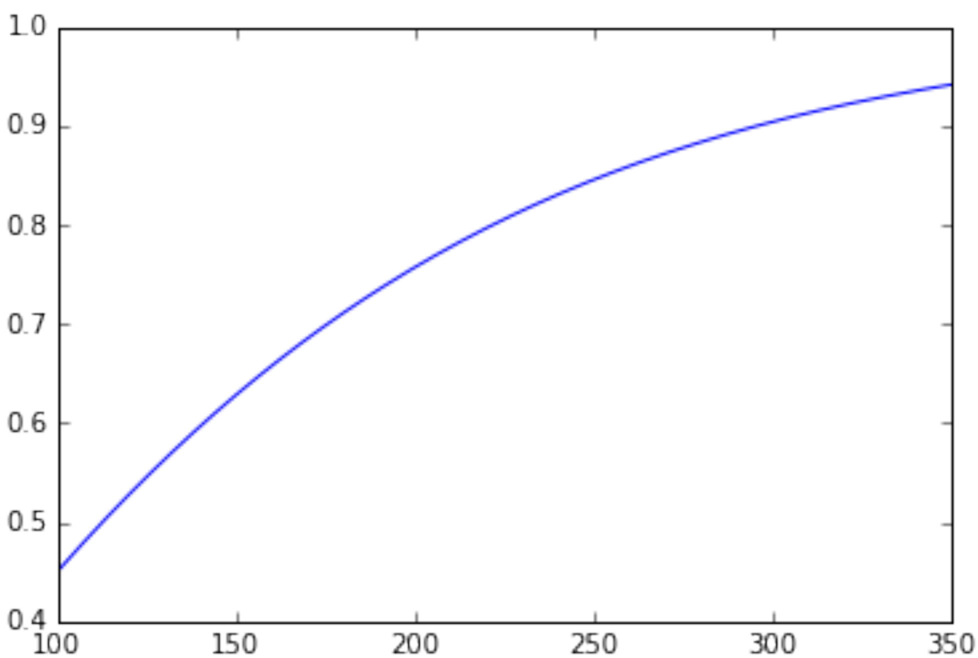
```
In [163]: # Nonlinear plots, pick an index below (0,1,2,etc). This code will plot b and bhat vs Ai
          i = 1

          # Write an expression for estimated p values here
          p_estimated = 1- np.exp(np.log(0.95)*np.exp(bhat - 26.1931))

          plt.plot(np.exp(A[:,i]),p,'ob')
          plt.plot(np.exp(A[:,i]),p_estimated,'or')

Out[163]: [<matplotlib.lines.Line2D at 0x108bf8588>]
```



2.9 Part i

In [164]:
```
# transform b_test into p_test
p_test = 1- np.exp(np.log(0.95)*np.exp(b_test - 26.1931))

plt.plot(tc_test_vector,p_test,'-b')
```

Out[164]: [<matplotlib.lines.Line2D at 0x106d85358>]



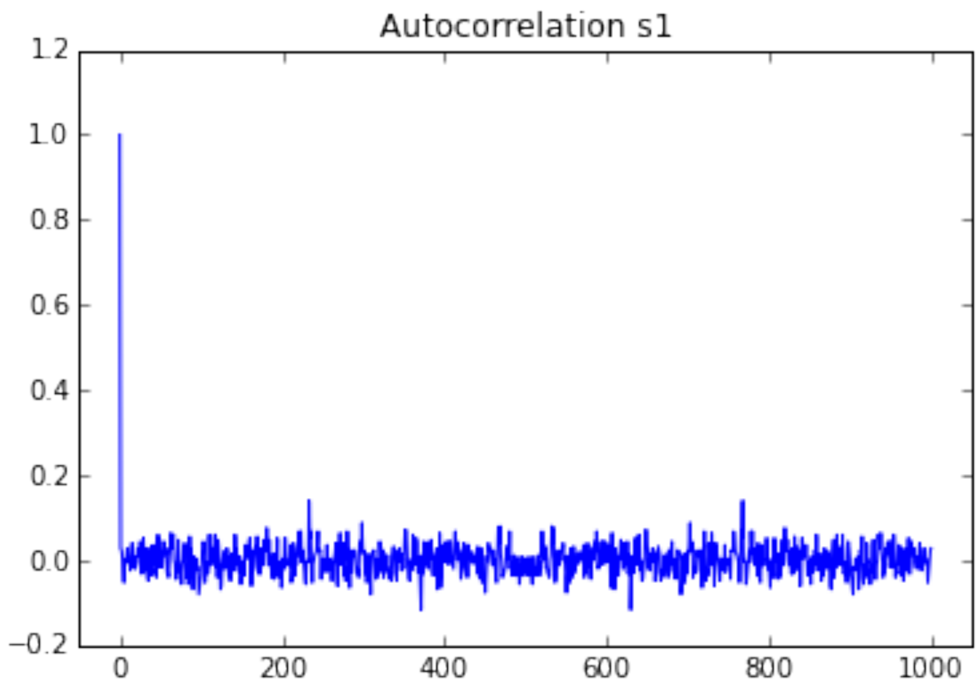
3 Finding Signals in Noise

In [165]:
```
# Run this first
%matplotlib inline
import numpy as np
import scipy as sp
import scipy.linalg as la
import pylab as plt
import numpy.random

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 shifted v
    C = la.circulant(f)
    corr = C.T.dot(g)
    return corr
```

4 (a)

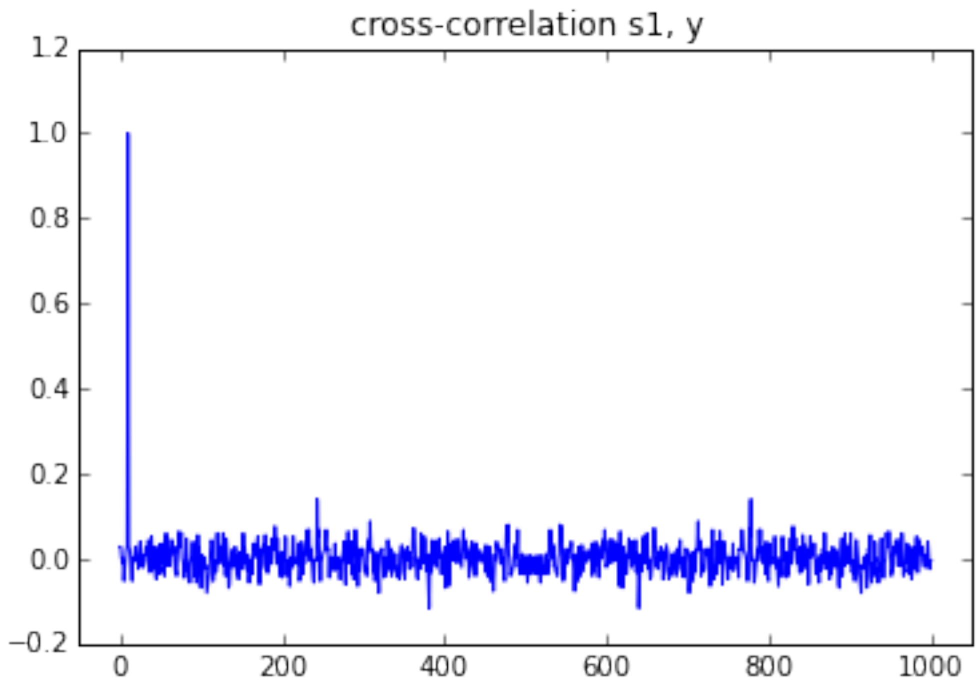
In [171]:
```
# generate a random normalized vector for s1
# (running this cell again will generate a new random vector)
s1 = rand_normed_vector(N)

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

# The inner-prouct <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()
```

```
0.028
0.028
```



5 (b)

```
In [172]: 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 s1)
          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
```

6 (c)

```
In [173]: # 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)))
```

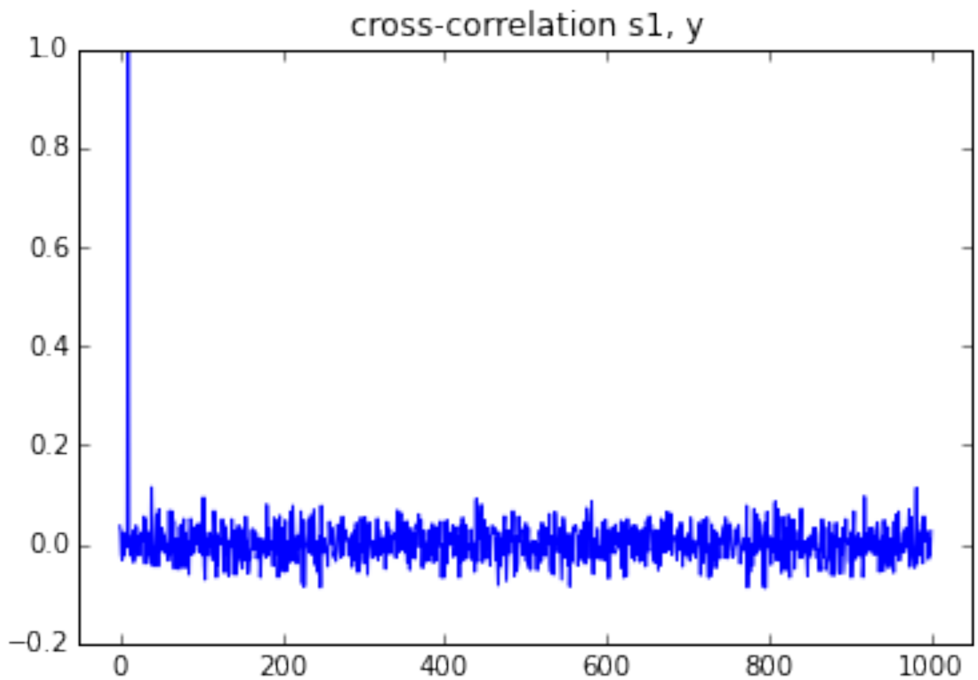
```
0.07
```

7 (d)

This is the code from part (b), but with the received signal y additionally corrupted by noise

```
In [174]: 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)
```



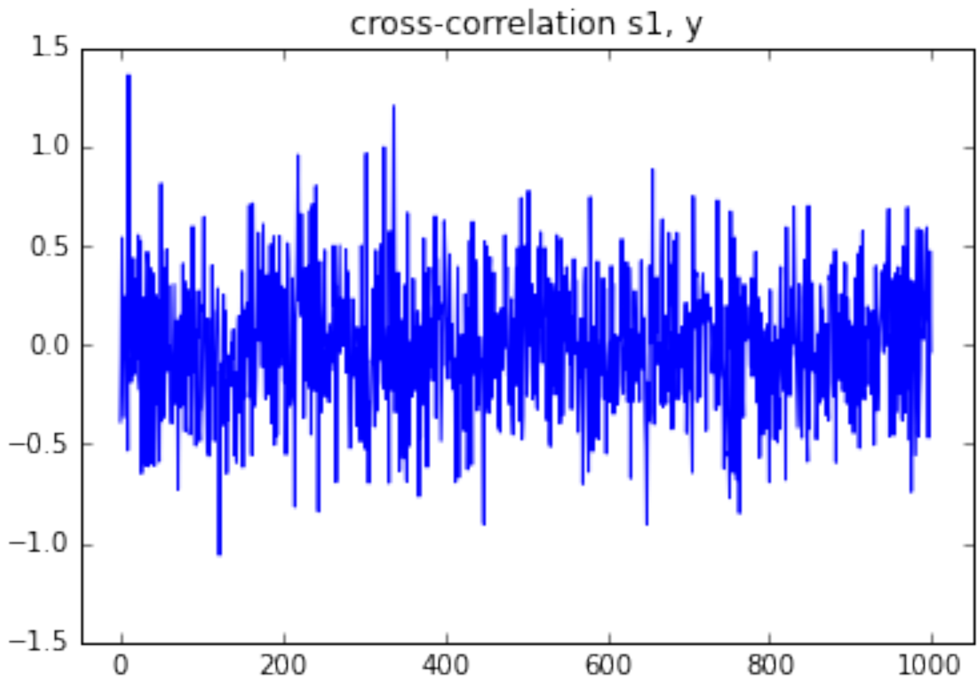

```
Out[174]: 10
```

8 (e)

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

```
In [175]: ## CODE HERE
          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)
```

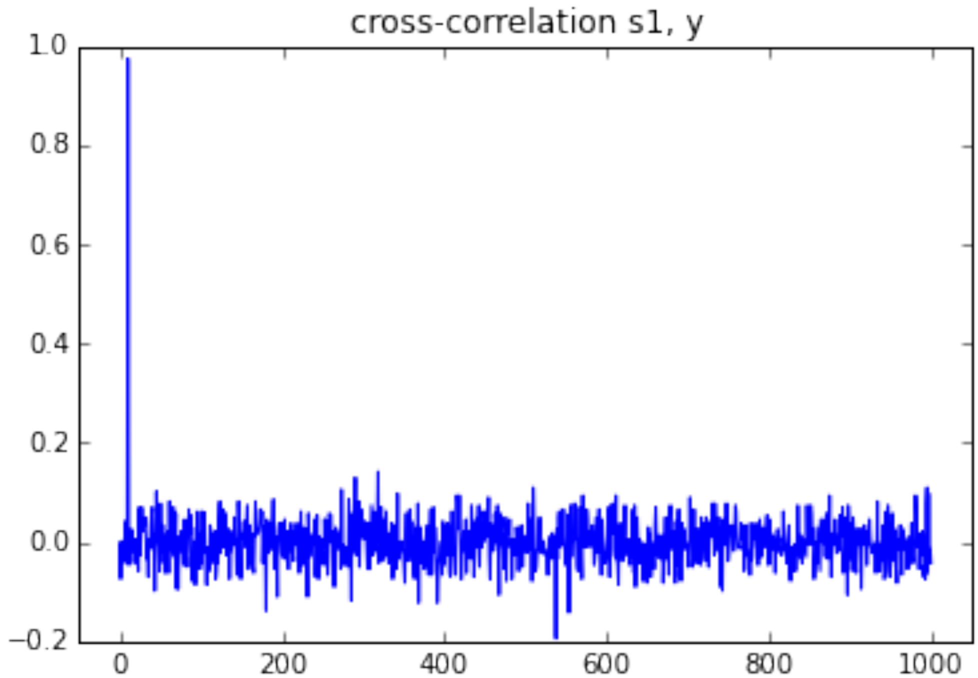


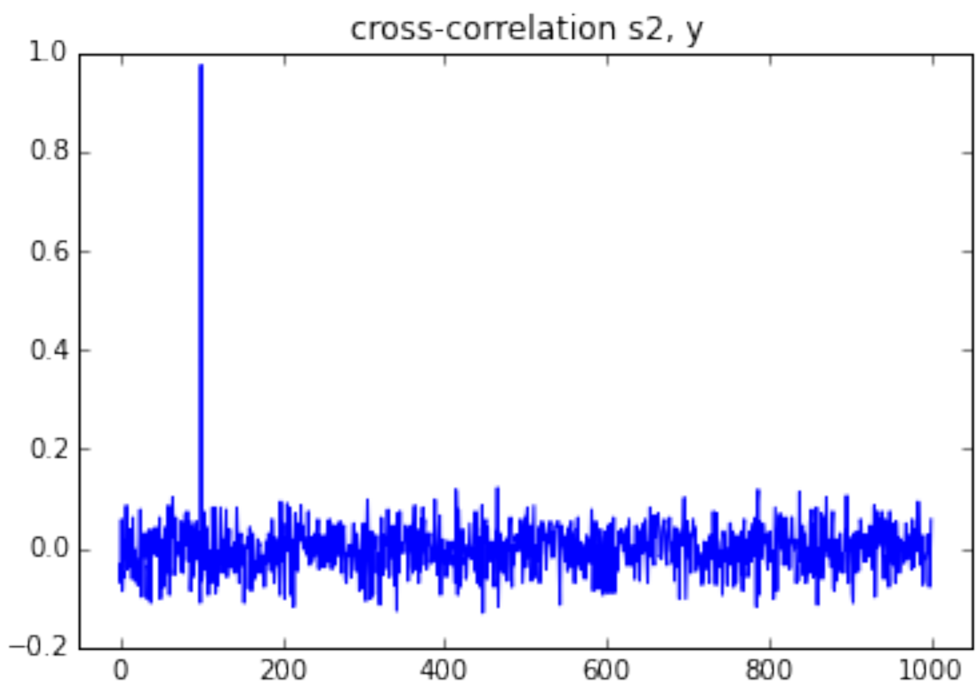
Out[175]: 10

9 (f)

In [176]:
```
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)
```



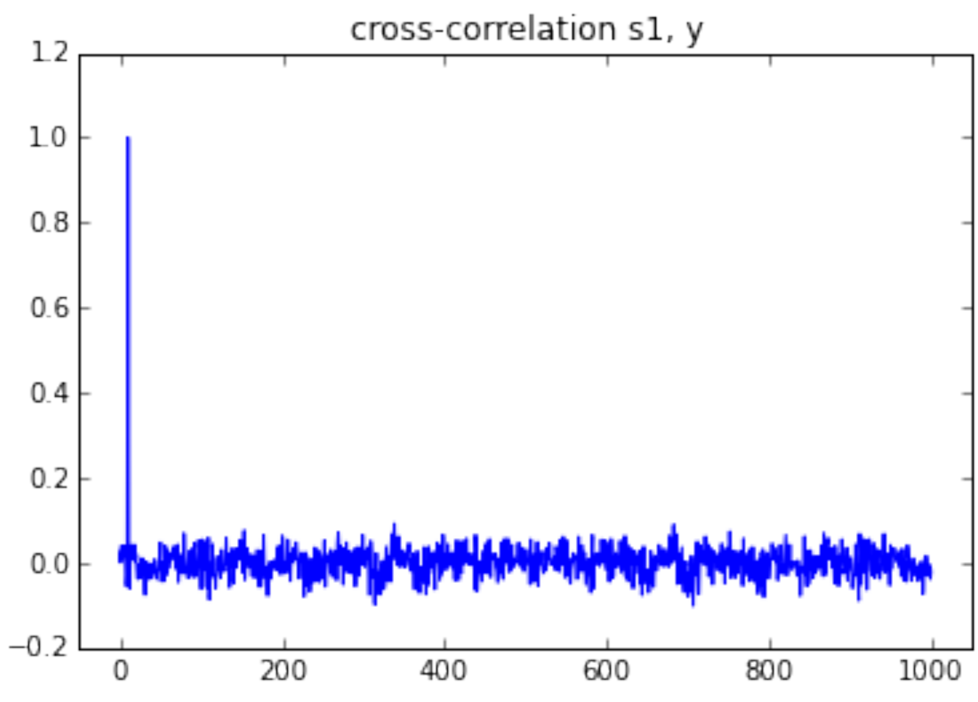


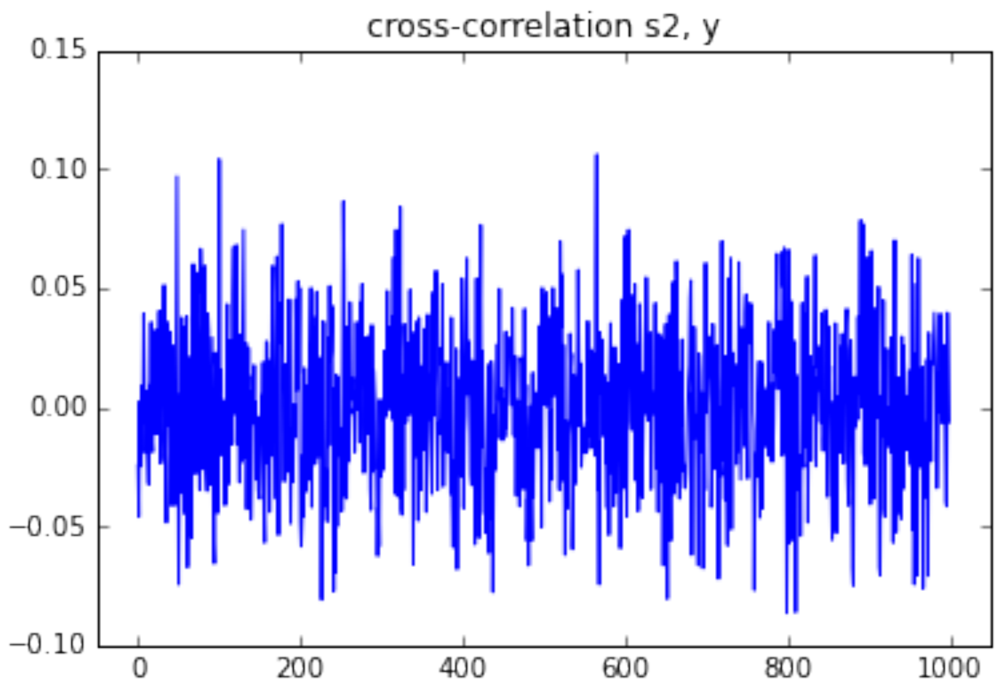
```
10 100
```

10 (g)

This is the same code as part (f), but with 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 [177]:
```python
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()
```

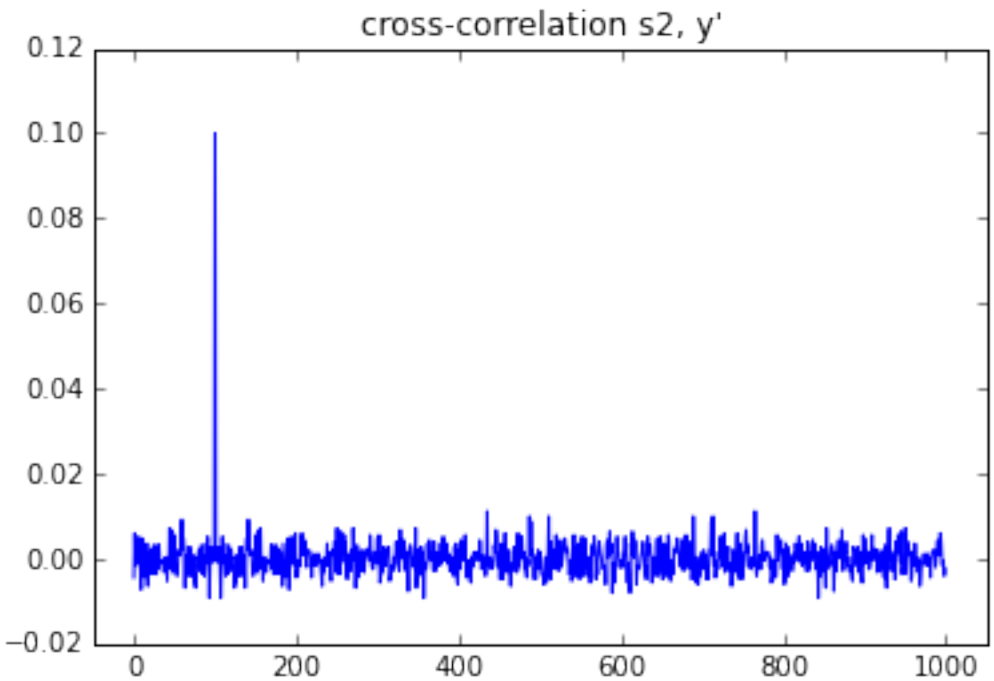




11 (h)

In [178]:
```
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 max correl
print(k)
```

10




```
100
```

12 (i)

```
In [179]: 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
0.708 0.5112
```

13 (j)

This is the same code as part (i), but with noise added to the received signal y.

```
In [180]: 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
0.7014 0.5052
```

14 (k)

```
In [181]: # Given the shifts j, k, setup the matrix A and vector b.

          # Hint: use np.roll(...) to circularly shift vectors.
          # For example, "np.roll(s1, j)" shifts the vector s1 by j indices.
          # A has columns c1, c2 which you should FILL IN BELOW.
          c1 = np.roll(s1,j)
          c2 = np.roll(s2, k)
          A = np.array([c1, c2]).T

          b = y

          # Solve to find the linear least-square solution of Ax ~ b (minimizing error ||Ax - b||)
          xhat = la.inv(A.T.dot(A)).dot(A.T).dot(b)
          print(xhat)
```

```
[ 0.69839394  0.50100964]
```

15 (l)

```
In [182]: # Load the signal vectors from file.
          npzfile = np.load("signals.npz")
          y, s1, s2, s3 = [npzfile[f] for f in ['y','s1','s2','s3']]
```

Try to find the delays and coefficients of the three signals s1, s2, s2, from your received signal y. Hint: Make use of the provided code in the previous parts. This should be possible by mostly copy/pasting code. In particular, remember:

- "np.roll(s1, 123)" circularly shifts vector s1 by 123
- "np.argmax(corr)" finds the index of the maximum entry in vector "corr".

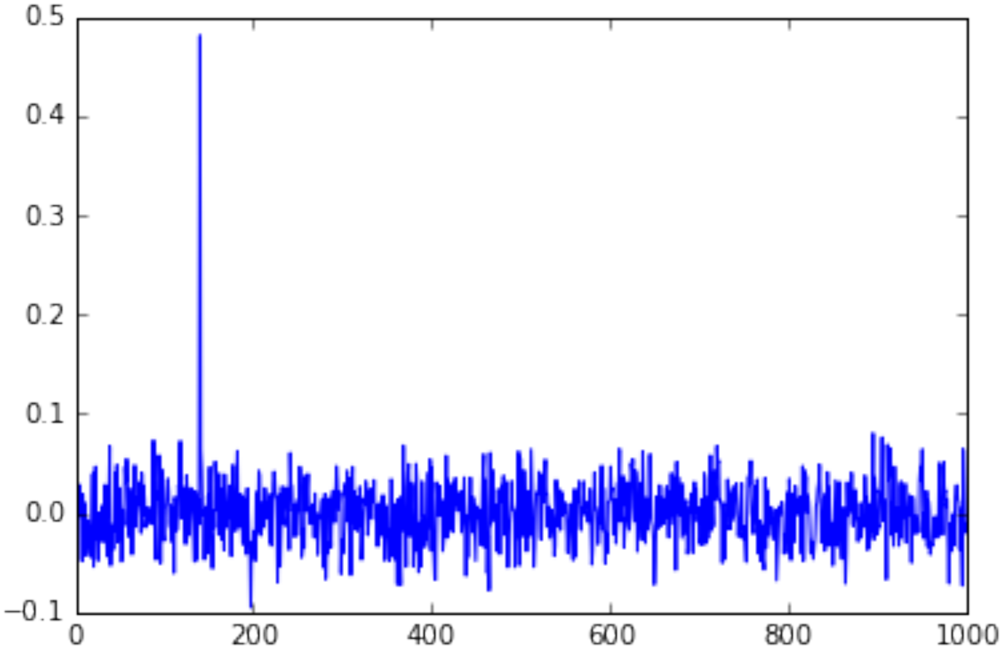
Once you have found candidate delays j, k, l, try running the following function. You should recognize the output.

```
In [183]: # Test your j,k,l by running this function:
          def test(j,k,l):
              return [chr(int(i)) for i in (np.array([j,k,l])/20 + 60)]
```

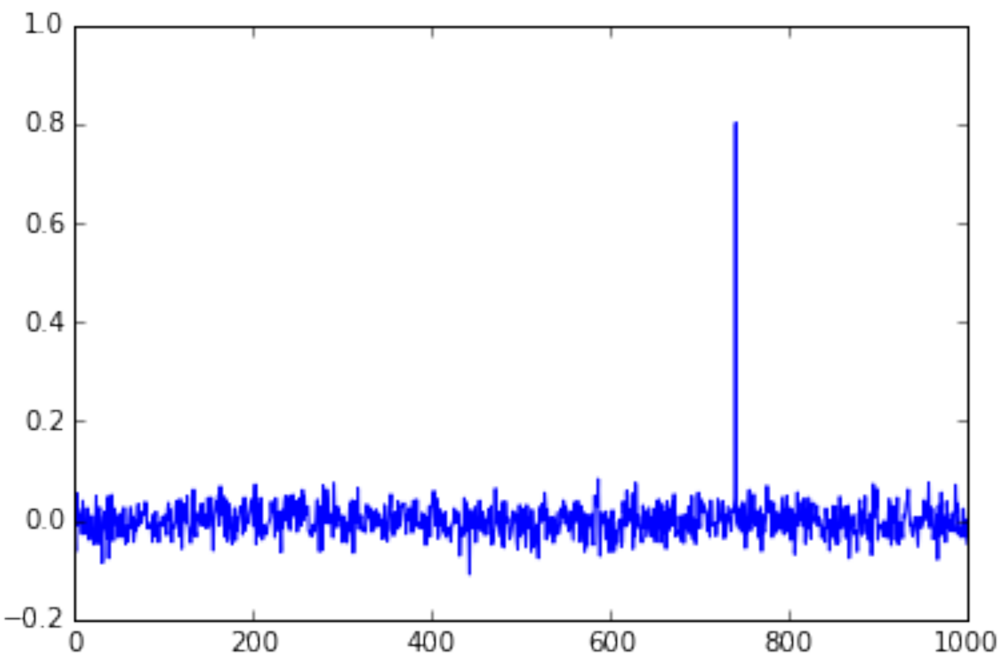
```
In [184]: ## TRY TO FIND THE SIGNALS HERE.

          print("\n Find delays")
          corr_s1_y = cross_corr(s1, y)
          j = np.argmax(corr_s1_y) # find the first signal delay
          print(j)
          plt.plot(corr_s1_y)
          plt.show()

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

          corr_s3_y = cross_corr(s3, y)
          l = np.argmax(corr_s3_y) # find the third signal delay
          print(l)
          plt.plot(corr_s3_y)
          plt.show()

          print("Signal 2 is the loudest so remove its contribution.")
          c1 = np.roll(s1,j)
          c2 = np.roll(s2, k)
          c3 = np.roll(s3, l)
          A = np.array([c1, c2, c3]).T

          b = y

          # Solve to find the linear least-square solution of Ax ~ b (minimizing error ||Ax - b||)
          xhat = la.inv(A.T.dot(A)).dot(A.T).dot(b)

          # subtract out the contribution of second signal
          y_prime = y - xhat[1]*c2


          print("\n Then repeat process of finding delays")
          corr_s1_y = cross_corr(s1, y_prime)
          j = np.argmax(corr_s1_y)
          print(j)
          plt.plot(corr_s1_y)
          plt.show()


          corr_s3_y = cross_corr(s3, y_prime)
```

```python
l = np.argmax(corr_s3_y)
print(l)
plt.plot(corr_s3_y)
plt.show()

print("Signal 1 is loud so subtract its contribution.")
c1 = np.roll(s1,j)
c2 = np.roll(s2, k)
c3 = np.roll(s3, l)
A = np.array([c1, c2, c3]).T

b = y

# Solve to find the linear least-square solution of Ax ~ b (minimizing error ||Ax - b||)
xhat = la.inv(A.T.dot(A)).dot(A.T).dot(b)

# subtract out the contribution of the first signal
y_prime = y - xhat[1]*c2 -xhat[0]*c1


print("Now just solve for signal 3")
corr_s3_y = cross_corr(s3, y_prime)
l = np.argmax(corr_s3_y)
print(l)
plt.plot(corr_s3_y)
plt.show()

c1 = np.roll(s1,j)
c2 = np.roll(s2, k)
c3 = np.roll(s3, l)
A = np.array([c1, c2, c3]).T

b = y

# Solve to find the linear least-square solution of Ax ~ b (minimizing error ||Ax - b||)
xhat = la.inv(A.T.dot(A)).dot(A.T).dot(b)
print("The new coefficients we get are: ", xhat)


test(j,k,l)
```

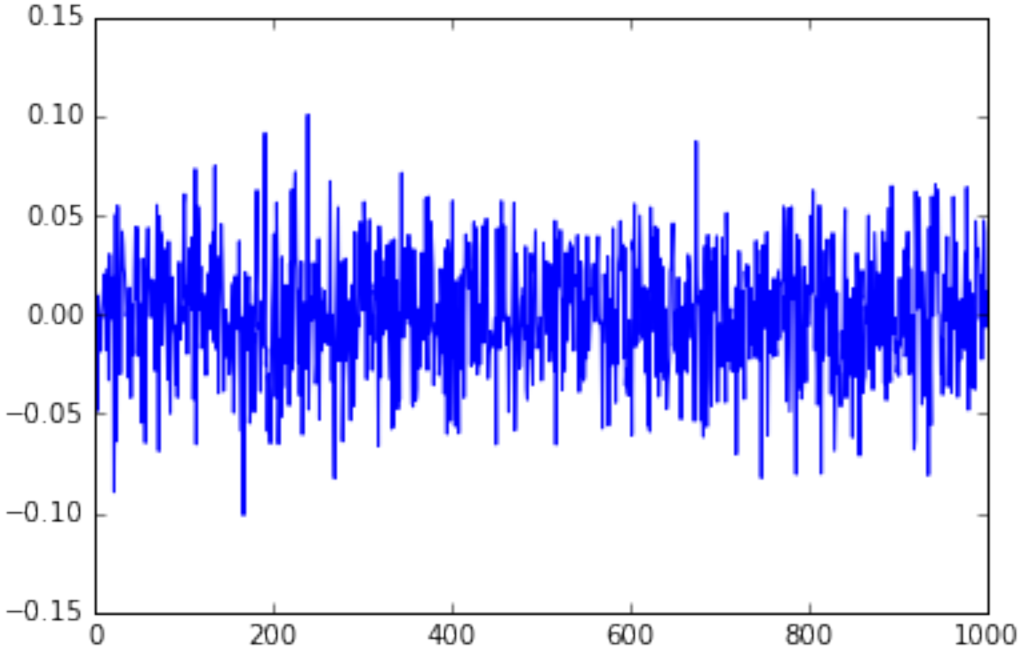
```
Find delays
140
```


0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0
200
400
600
800
1000

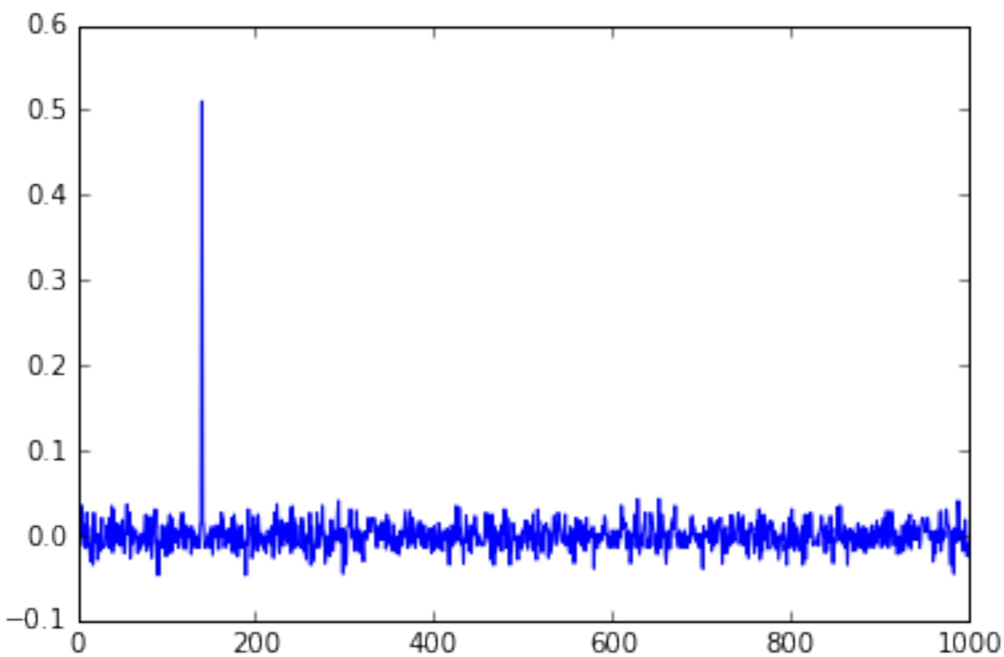
740


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000

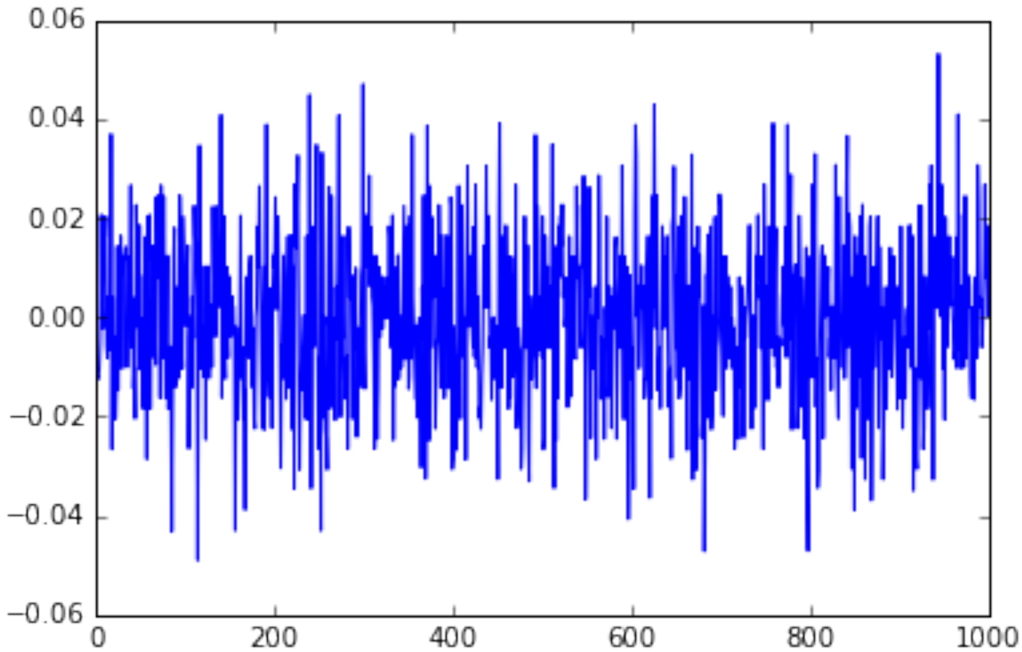
239



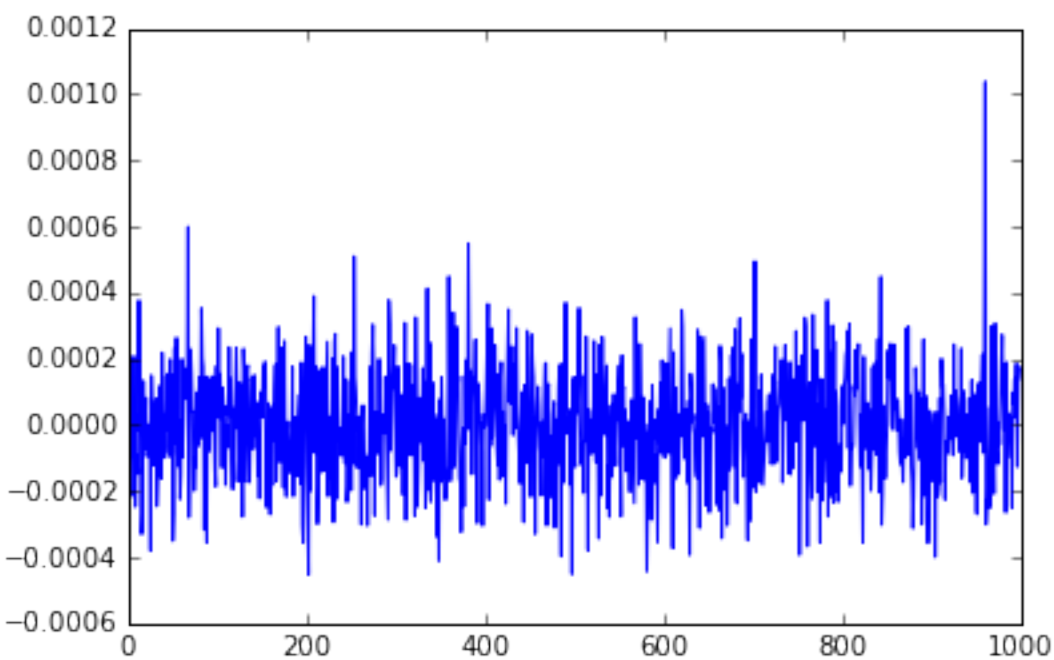

```
Signal 2 is the loudest so remove its contribution.

 Then repeat process of finding delays
140
```



```
943
```




```
Signal 1 is loud so subtract its contribution.
Now just solve for signal 3
960
```



```
The new coefficients we get are:  [ 0.51019847  0.81994888  0.00103934]
```

```
Out[184]: ['C', 'a', 'l']
```

16 Deconstructing Trolls

```
In [185]: import numpy as np
          import matplotlib.pyplot as plt
          import wave as wv
          import scipy
          from scipy import io
          import scipy.io.wavfile
          from scipy.io.wavfile import read
          from IPython.display import Audio
          import warnings
          warnings.filterwarnings('ignore')
          sound_file_1 = 'm1.wav'
          sound_file_2 = 'm2.wav'
          rate1,corrupt1 = scipy.io.wavfile.read('m1.wav')
          rate2,corrupt2 = scipy.io.wavfile.read('m2.wav')
```

Just as last time, let's listen to the inputs.

```
In [186]: Audio(url='m1.wav', autoplay=False)
```

```
Out[186]: <IPython.lib.display.Audio object>
```

```
In [187]: Audio(url='m2.wav', autoplay=False)
```

```
Out[187]: <IPython.lib.display.Audio object>
```

In the cell below, complete the function to find the vectors $\vec{a}$ and $\vec{b}$. Make sure $\vec{a}$ is the original speech and not the troll.

```
In [189]: def remove_troll(m1, m2):
              ##Your code here


              a =
              b =
              return a, b
```

```
          File "<ipython-input-189-531044194be5>", line 5
        a =
           ^
     SyntaxError: invalid syntax
```

Run the cell below to test your function

```
In [190]: a, b = remove_troll(corrupt1, corrupt2)
          Audio(data=a, rate=rate1)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-190-336f73bed4ad> in <module>()
----> 1 a, b = remove_troll(corrupt1, corrupt2)
      2 Audio(data=a, rate=rate1)


NameError: name 'remove_troll' is not defined
```

Let's now compare our output here to the output from Homework 1. Read through the block of code below and comment on it's output

```
In [191]: ## First let's compute the original vectors representing the speakers using the technique in
          a_u = np.sqrt(2)/(1+np.sqrt(3))
          a_v = np.sqrt(6)/(1+np.sqrt(3))
          b_u = np.sqrt(2)/(1+np.sqrt(3))
          b_v = -1*np.sqrt(2)/(1+np.sqrt(3))
          s1 = a_u*corrupt1 + a_v*corrupt2
          s2 = b_u*corrupt1 + b_v*corrupt2

          ## Here we will compute various dot products to see which vectors are orthogonal.
          ## Note that we normalize the vectors before we compare, this is because we want
          ## to get rid of any scaling.
          print("Dot product of the two original speaker outputs ", np.dot(s1/np.linalg.norm(s1), s2/np
          print("Dot product of calculated a and b ", np.dot(a/np.linalg.norm(a), b/np.linalg.norm(b)))
```

```
Dot product of the two original speaker outputs  -0.00379040459598
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-191-be4cba61913b> in <module>()
     11 ## to get rid of any scaling.
     12 print("Dot product of the two original speaker outputs ", np.dot(s1/np.linalg.norm(s1), s2/
---> 13 print("Dot product of calculated a and b ", np.dot(a/np.linalg.norm(a), b/np.linalg.norm(b)


ValueError: shapes (6,) and (1000,) not aligned: 6 (dim 0) != 1000 (dim 0)
```

```
In [ ]:
```