Group Members: Jason Carlson, Alex Reust, Kevin de Szendeffy, Henrique Cheng

HW3

In [3]:
```
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt
import scipy as sp
import scipy.stats as sps
import math
from time import time
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
import itertools
```

Problem 1

Let Y be a random variable with a normal distribution with parameters 0, Σ (mean zero and variance Σ)

In [4]:
```
sigma = np.array([
    [1.0,  0.8,  0.8,  0.8,  0.8],
    [0.8,  1.0,  0.8,  0.8,  0.8],
    [0.8,  0.8,  1.0,  0.8,  0.8],
    [0.8,  0.8,  0.8,  1.0,  0.8],
    [0.8,  0.8,  0.8,  0.8,  1.0]
], dtype=float)
sigma
```

Out[4]:
```
array([[1. , 0.8, 0.8, 0.8, 0.8],
       [0.8, 1. , 0.8, 0.8, 0.8],
       [0.8, 0.8, 1. , 0.8, 0.8],
       [0.8, 0.8, 0.8, 1. , 0.8],
       [0.8, 0.8, 0.8, 0.8, 1. ]])
```

a) What is the marginal distribution of (Y_1, Y_2)?

(Y_1, Y_2) ~ $N(0, X)$ where X equals:

In [5]:
```
X = sigma[[0,1],:][:,[0,1]]
X
```

Out[5]:
```
array([[1. , 0.8],
       [0.8, 1. ]])
```

b) What is the conditional distribution of $(Y_1, Y_2)|Y_2 = .23, Y_4 = -.65, Y_5 = -.3$?

[8]:
```
x = np.transpose([0.23, -0.65, -0.3])
```

First, we note it is equivalent to find the conditional distribution of (Y_4, Y_5) given $Y_1 = 0.23$, $Y_2 = -0.65$, and $Y_3 = -0.3$. Then we define L_1, A, and L_2 as follows.

[9]:
```
L1 = L[0:3, 0:3]
L2 = L[3:5, 3:5]
A = L[3:5, 0:3]
```

Thus the conditional distribution of (Y_4, Y_5) given $Y_1 = 0.23$, $Y_2 = -0.65$, and $Y_3 = -0.3$ is $N(\mu_{45|123}, \Sigma_{45|123})$ where $\mu_{45|123}$ is

[10]:
```
A@np.linalg.inv(L1)@x
```

[10]:
```
array([-0.22153846, -0.22153846])
```

and $\Sigma_{45|123})$ is

[11]:
```
L2@np.transpose(L2)
```

[11]:
```
array([[0.26153846, 0.06153846],
       [0.06153846, 0.26153846]])
```

c) What s L in $\Sigma = LL'$?

In [8]:
```
L = sp.linalg.cholesky(sigma, lower=True)
L
```

Out[8]:
```
array([[1.        , 0.        , 0.        , 0.        , 0.        ],
       [0.8       , 0.6       , 0.        , 0.        , 0.        ],
       [0.8       , 0.26666667, 0.53748385, 0.        , 0.        ],
       [0.8       , 0.26666667, 0.16537965, 0.51140831, 0.        ],
       [0.8       , 0.26666667, 0.16537965, 0.12033137, 0.49705012]])
```

d) What is L^{-1}?

In [9]:
```
L_inverse = np.linalg.inv(L)
L_inverse
```

Out[9]:
```
array([[ 1.        ,  0.        ,  0.        ,  0.        ,  0.        ],
       [-1.33333333,  1.66666667,  0.        ,  0.        ,  0.        ],
       [-0.82689823, -0.82689823,  1.86052102,  0.        ,  0.        ],
       [-0.60165684, -0.60165684, -0.60165684,  1.95538472,  0.        ],
       [-0.47338107, -0.47338107, -0.47338107, -0.47338107,  2.01186954]])
```

e) What is A in $A = PD^{\frac{1}{2}}$.

In [10]:
```
D, P = np.linalg.eig(L)
D_half = np.sqrt(D)
A = np.matmul(P, D_half)
A
```

Out[10]:
```
array([0.14562795, 0.32198369, 0.58162993, 1.15640372, 3.45859482])
```

1.0.6 f)

[17]:
```
ysim = L @ np.random.normal(size=(5, 10000))
df = pd.DataFrame(ysim).T
ybar = df.apply(np.mean, axis=0).values
```

[18]:
```
sigmahat = np.zeros(Sigma.shape)
for i, r in df.iterrows():
    hold = (r.values-ybar)[:, np.newaxis]
    sigmahat += hold @ hold.T
sigmahat/= df.shape[0]
```

[19]:
```
print(ybar)
print(sigmahat)
```

```
[0.0096619  0.00767152 0.00338953 0.00290112 0.00266758]
[[0.99358867 0.80595063 0.80931672 0.79806052 0.80461073]
 [0.80595063 1.01499931 0.8164054  0.80298267 0.81187738]
 [0.80931672 0.8164054  1.01646978 0.81051097 0.81420219]


 [0.79806052 0.80298267 0.81051097 1.00016302 0.80852963]
 [0.80461073 0.81187738 0.81420219 0.80852963 1.01087406]]
```

[20]:
```
mu_diff = np.linalg.norm(ybar-np.zeros(p))
sigma_diff = np.linalg.norm(sigmahat - Sigma)
print("n = ", 10000)
print("Mu diff: ", mu_diff)
print("Sigma diff: ", sigma_diff)
```

```
n =  10000
Mu diff:  0.013387528088009627
Sigma diff:  0.05057825761213993
```

1.0.7 g)

[21]:
```
ysim = L @ np.random.normal(size=(5, 50))
df = pd.DataFrame(ysim).T
ybar = df.apply(np.mean, axis=0).values
sigmahat = np.zeros(Sigma.shape)
for i, r in df.iterrows():
    hold = (r.values-ybar)[:, np.newaxis]
    sigmahat += hold @ hold.T
sigmahat/= df.shape[0]
mu_diff = np.linalg.norm(ybar-np.zeros(p))
sigma_diff = np.linalg.norm(sigmahat - Sigma)
print("n = ", 50)
print("Mu diff: ", mu_diff)
print("Sigma diff: ", sigma_diff)
```

```
n =  50
Mu diff:  0.20541184830511416
Sigma diff:  1.0208336116865688
```

Note: the following code (for problem 2) was written in R.

Problem 2

The algorithm seems correct. Here is the original function followed by a vectorized version

```
logL=function(Y,mu,Sigma)
{
   n = nrow(Y); p=ncol(Y)
   dS = det(Sigma)
   retval = -.5*n*log(dS)
   Si = solve(Sigma)
   for(i in 1:n) {
      yi = matrix(Y[i,]-mu,ncol=1)
      retval = retval - .5 * t(yi) %*% Si %*% yi
   }
   return(retval)
}

logLVec=function(Y,mu,Sigma)
{
  n = nrow(Y); p=ncol(Y)
  dS = det(Sigma)
  retval = -.5*n*log(dS)
  Si = solve(Sigma)
  Ybar=colMeans(Y)
  Yminmu=(t(Y)-mu)
  retval=retval-.5*(sum(diag(t(Yminmu)%*%Si%*%Yminmu)))
  return(retval)
}

#Check the speed with "sample2" from problem 1
library(microbenchmark)
```

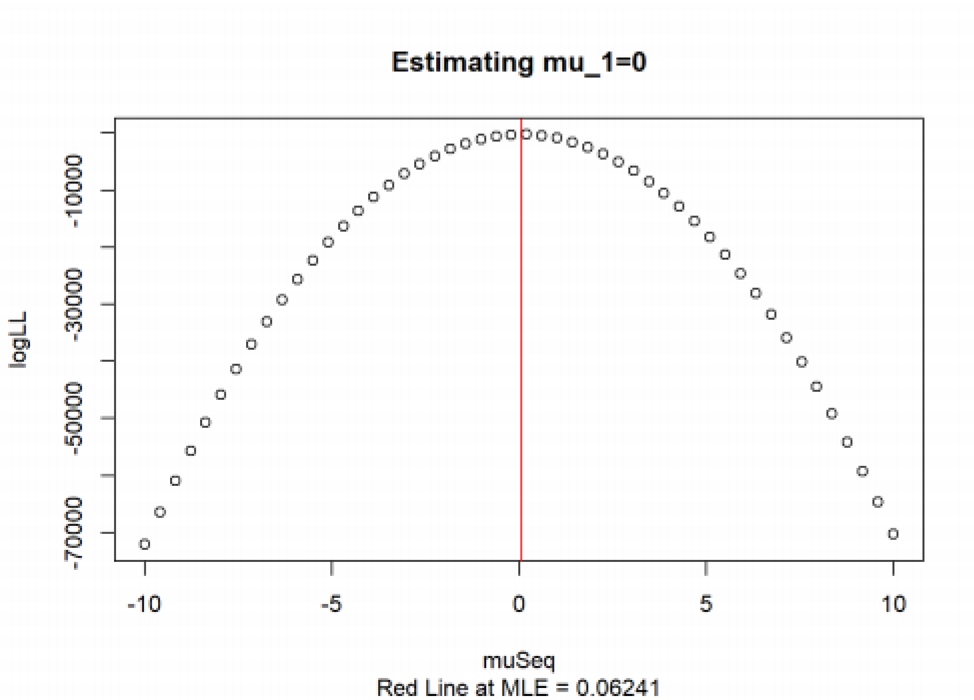
```
## Warning: package 'microbenchmark' was built under R version 4.0.3
```

```
microbenchmark(logL(sample2,mu,Sigma),logLVec(sample2,mu,Sigma))
```

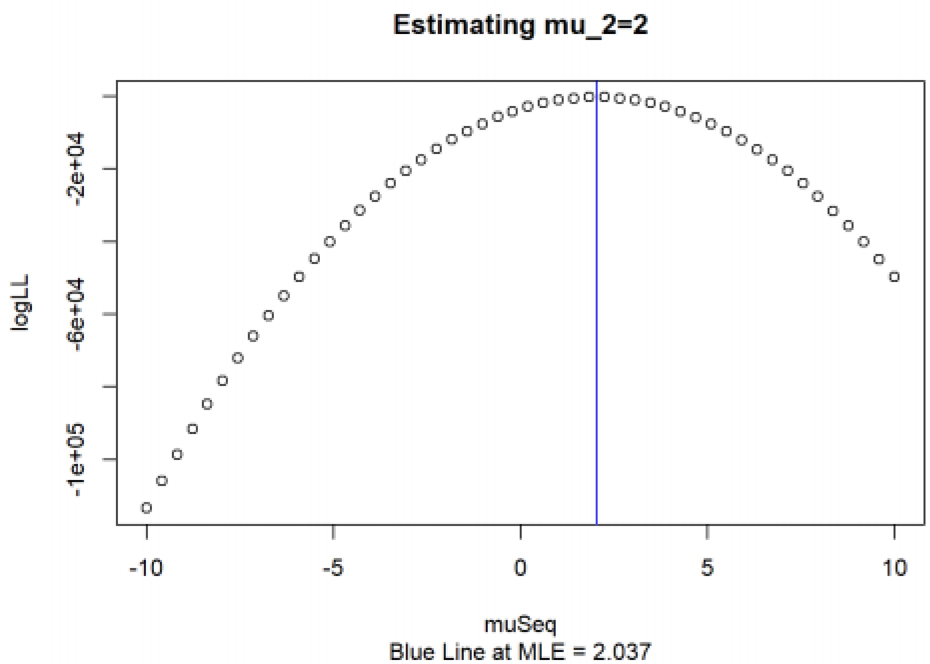
```
## Unit: microseconds
##                          expr   min     lq     mean  median      uq     max neval
##     logL(sample2, mu, Sigma) 697.7 910.9 1400.821 1088.85 1167.70 32255.5   100
##  logLVec(sample2, mu, Sigma) 126.4 170.6 1105.052  198.70  227.45 82991.8   100
```

The stats on the vectorized version of the code are much better. Now on to some plotting!

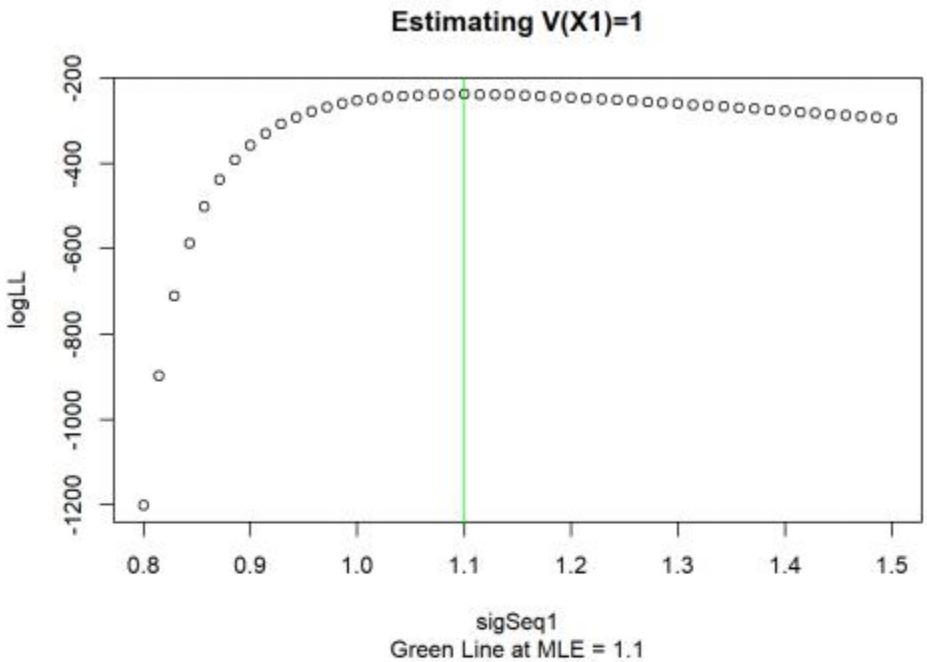
```
mu2=c(0,2)
Sigma2=Sigma[1:2,1:2]
n3=500
set.seed(3)
sample3=mvrnorm(n3,mu2,Sigma2)
mu2Est=colMeans(sample3)
sig2Est=cov(sample3)
muSeq=seq(from=-10,to=10,length.out=50)
logLLmu1=rep(NA,50)
logLLmu2=rep(NA,50)
for(i in 1:length(muSeq))
{
  logLLmu1[i]=logLVec(sample3,c(muSeq[i],mu2Est[2]),sig2Est)
  logLLmu2[i]=logLVec(sample3,c(mu2Est[1],muSeq[i]),sig2Est)
}

plot(muSeq,logLLmu1,main="Estimating mu_1=0",ylab="logLL",
     sub=paste("Red Line at MLE =",signif(mu2Est[1],4)))
abline(v=mu2Est[1],col="red")
```



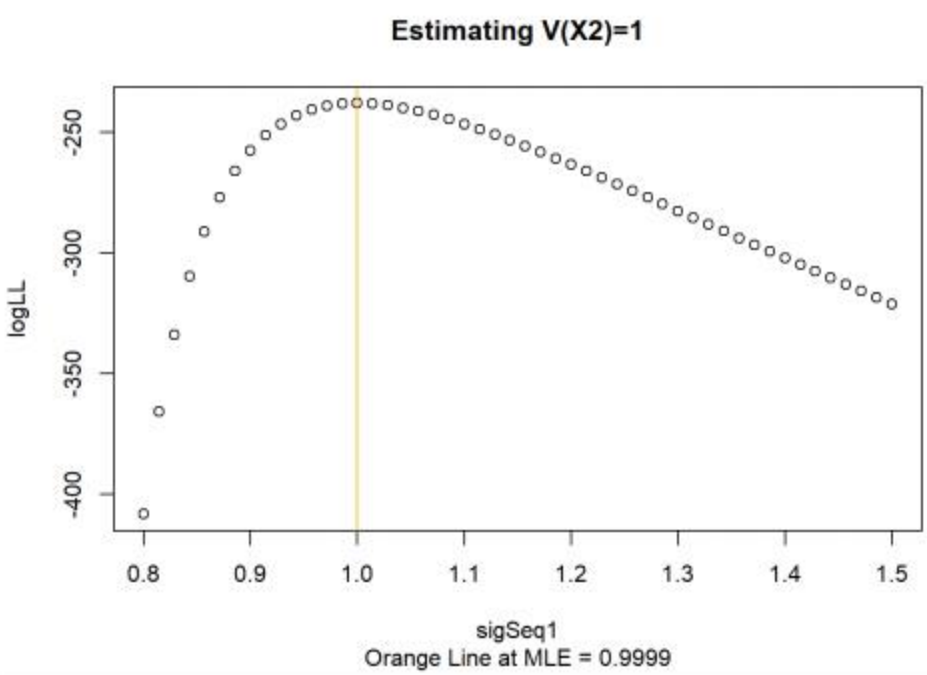

```
plot(muSeq,logLLmu2,main="Estimating mu_2=2",ylab="logLL",
     sub=paste("Blue Line at MLE =",signif(mu2Est[2],4)))
abline(v=mu2Est[2],col="blue")
```



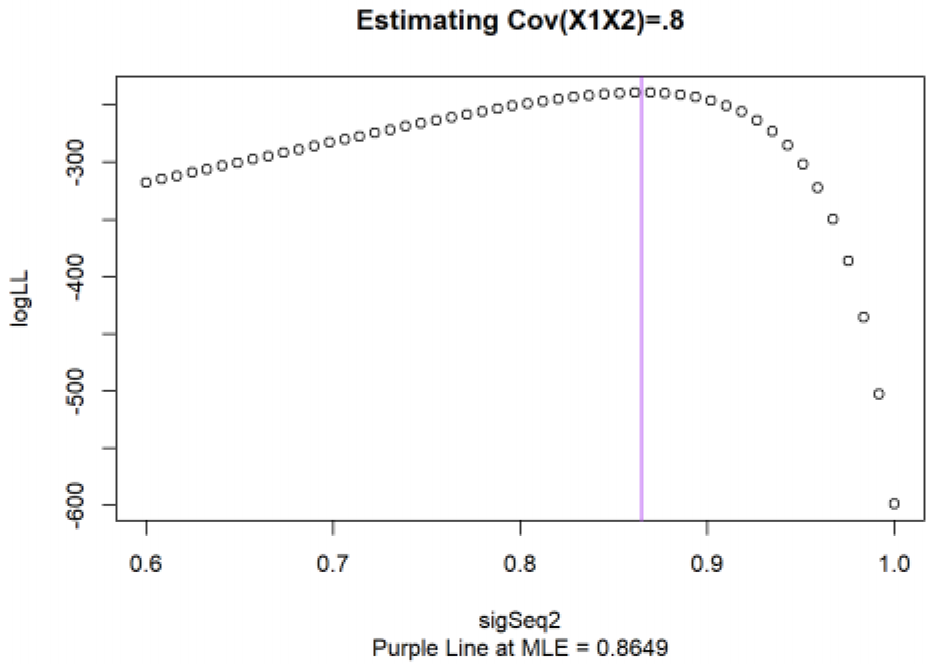

```
sigSeq1=seq(from=.8,to=1.5,length.out=50)
sigSeq2=seq(from=.6,to=1,length.out=50)
logLLVX1=rep(NA,50)
logLLVX2=rep(NA,50)
logLLCovX1X2=rep(NA,50)
Sigma2Ed1=Sigma2Ed2=Sigma2Ed3=sig2Est
for(i in 1:length(sigSeq1))
{
  Sigma2Ed1[1,1]=sigSeq1[i]
  logLLVX1[i]=logLVec(sample3,mu2Est,Sigma2Ed1)
  Sigma2Ed2[2,2]=sigSeq1[i]
  logLLVX2[i]=logLVec(sample3,mu2Est,Sigma2Ed2)
  Sigma2Ed3[1,2]=Sigma2Ed3[2,1]=sigSeq2[i]
  logLLCovX1X2[i]=logLVec(sample3,mu2,Sigma2Ed3)
}

plot(sigSeq1,logLLVX1,main="Estimating V(X1)=1",ylab="logLL",
     sub=paste("Green Line at MLE =",signif(sig2Est[1,1],4)))
abline(v=sig2Est[1,1],col="green")
```




```
plot(sigSeq1,logLLVX2,main="Estimating V(X2)=1",ylab="logLL",
     sub=paste("Orange Line at MLE =",signif(sig2Est[2,2],4)))
abline(v=sig2Est[2,2],col="orange")
```




```
plot(sigSeq2,logLLCovX1X2,main="Estimating Cov(X1X2)=.8",ylab="logLL",
     sub=paste("Purple Line at MLE =",signif(sig2Est[1,2],4)))
abline(v=sig2Est[1,2],col="purple")
```


Estimating Cov(X1X2)=.8
logLL
-300
-400
-500
-600
0.6
0.7
0.8
0.9
1.0
sigSeq2
Purple Line at MLE = 0.8649

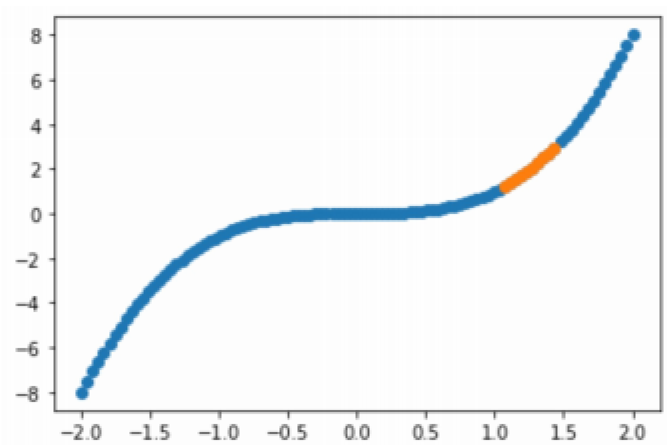
Note: The code for problem 3 is in Python.

Problem 3: Gaussian Process

In [19]:
```
# Generate the entire dataset
n = 100
x = np.linspace(-2, 2, num=n)
fx = x**3
# Index the known vs uknown data
obs = list(range( 75))
obs.extend(list(range(86, 100)))
y = list(range(76,86))
print(y)
X = x[obs]
# Plot the data
plt.scatter(x, fx)
plt.scatter(x[y], fx[y])
```

```
[76, 77, 78, 79, 80, 81, 82, 83, 84, 85]
```

Out[19]: `<matplotlib.collections.PathCollection at 0x1dcff94cee0>`



In [20]:

```
# test some values of sigf, l
l = 0.1
sigmaf = 0.1
distances = x - x[:, np.newaxis]
sigma = sigmaf**2 * np.exp(-(distances**2/l**2/2))
```

In [21]:

```
# Calculate Y | X
sig11 = sigma[obs, :][:, obs]
sig22 = sigma[y, :][:, y]
sig12 = sigma[obs, :][:, y]
sig21 = sig12.T

sig11inv = LA.inv(sig11)
sigbar = sig22 - LA.multi_dot([sig21, sig11inv, sig12])
mubar = sig21 @ sig11inv @ fx[obs]
```

In [22]:

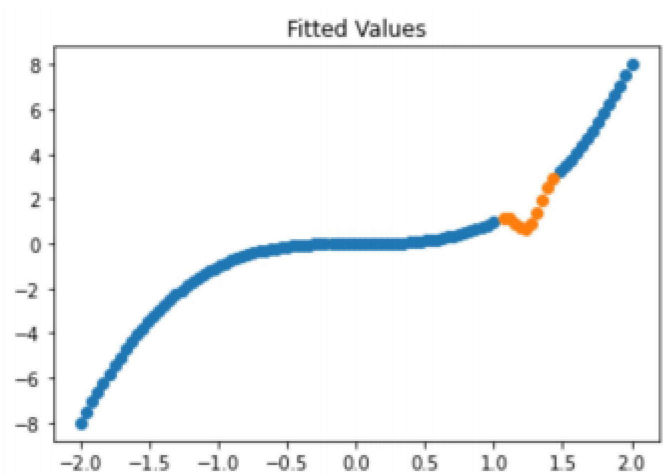
```
# Fit yhat
yhat_samp = np.random.multivariate_normal(mubar, sigbar, 1000)
# Take the posterior mean
yhat = np.mean(yhat_samp, axis=0)
print(yhat)
```

```
[1.14793154 1.0992842  0.92641469 0.73076406 0.67872514 0.89508971
 1.3777657  1.98497146 2.52840516 2.91899081]
```

In [23]:

```
plt.scatter(x[obs], fx[obs])
plt.scatter(x[y], yhat)
plt.title("Fitted Values")
```

Out[23]: `Text(0.5, 1.0, 'Fitted Values')`



The values (.1, .1) were chosen because many other nearby values resutled in correlation matrices that were not full rank. Larger values resulted in worse fits.

$Y | X$ is a good generic approximation of the true values of Y