NUEN 647
Uncertainty Quantification for Nuclear Engineering
Homework 3

Due on Saturday, December 10, 2016

Dr. McClarren

Paul Mendoza

Contents

Problem 1

Fit the data in Table 1 to a linear model using

(a) Least squares

(b) Ridge Regression

(c) Lasso Regression

Table 1: Data to fit linear model $y = a + bx_1 + cx_2$

	x_1	x_2	y
1	0.99	0.98	6.42
2	-0.75	-0.76	0.20
3	-0.50	-0.48	0.80
4	-1.08	-1.08	-0.57
5	0.09	0.09	4.75
6	-1.28	-1.27	-1.42
7	-0.79	-0.79	1.07
8	-1.17	-1.17	0.20
9	-0.57	-0.57	1.08
10	-1.62	-1.62	-0.15
11	0.34	0.35	2.90
12	0.51	0.51	3.37
13	-0.91	-0.92	0.05
14	1.85	1.86	5.50
15	-1.12	-1.12	0.17
16	-0.70	-0.70	1.72
17	1.19	1.18	3.97
18	1.24	1.23	6.38
19	-0.52	-0.52	3.29
20	-1.41	-1.41	-1.49

Be sure to do cross-validation for each fit, and for each method present your best estimate of the model.

Did this problem in R, and appended the PDF on the following pages.

Problem1

Paul Mendoza

December 9, 2016

```r
require(magrittr)
require(dplyr)
require(ggplot2)
require(glmnet)
```

Fit the data in Table 1 to a linear model using:

```r
Table1
```

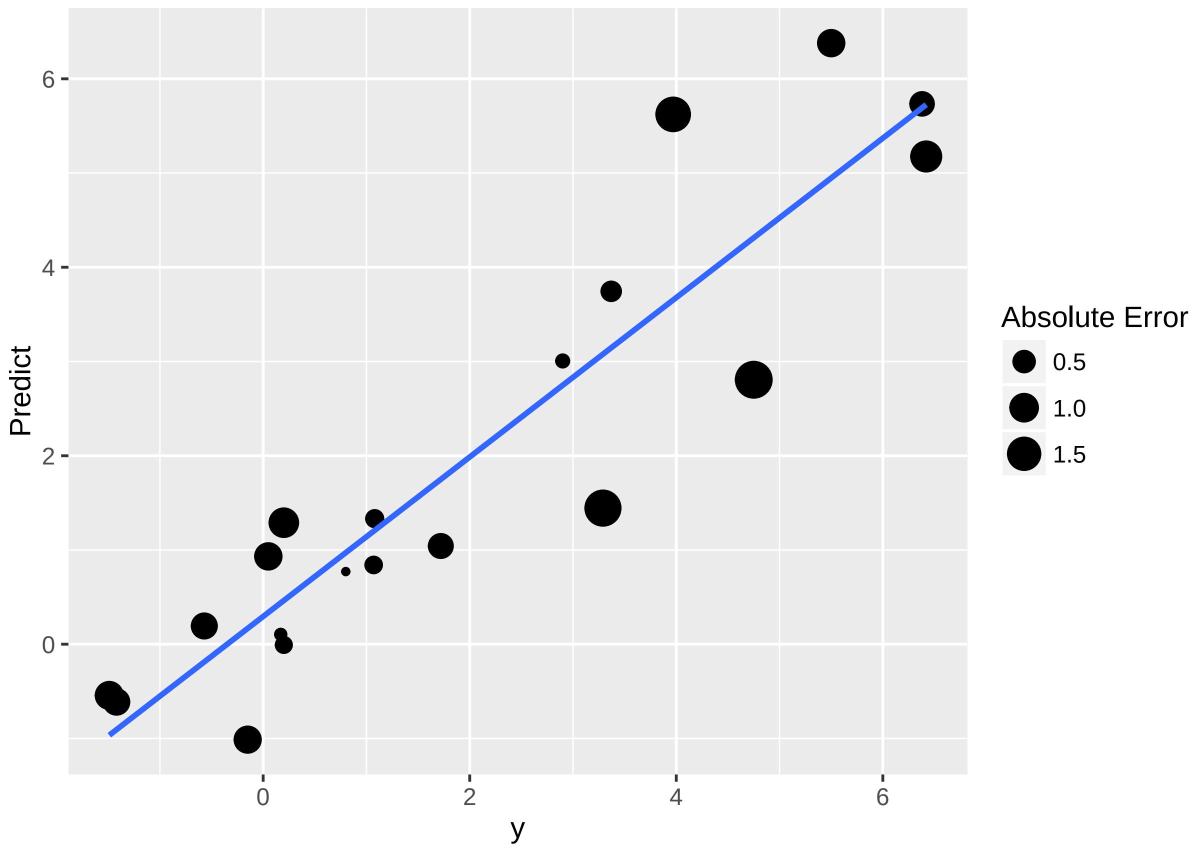
```
##       x1    x2     y
## 1   0.99  0.98  6.42
## 2  -0.75 -0.76  0.20
## 3  -0.50 -0.48  0.80
## 4  -1.08 -1.08 -0.57
## 5   0.09  0.09  4.75
## 6  -1.28 -1.27 -1.42
## 7  -0.79 -0.79  1.07
## 8  -1.17 -1.17  0.20
## 9  -0.57 -0.57  1.08
## 10 -1.62 -1.62 -0.15
## 11  0.34  0.35  2.90
## 12  0.51  0.51  3.37
## 13 -0.91 -0.92  0.05
## 14  1.85  1.86  5.50
## 15 -1.12 -1.12  0.17
## 16 -0.70 -0.70  1.72
## 17  1.19  1.18  3.97
## 18  1.24  1.23  6.38
## 19 -0.52 -0.52  3.29
## 20 -1.41 -1.41 -1.49
```

1. Least Squares

```r
LeastFit<-lm(formula = y~x1+x2,data=Table1)
LeastFit
```

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = Table1)
##
## Coefficients:
## (Intercept)           x1           x2
##       2.606       38.133      -35.900
```

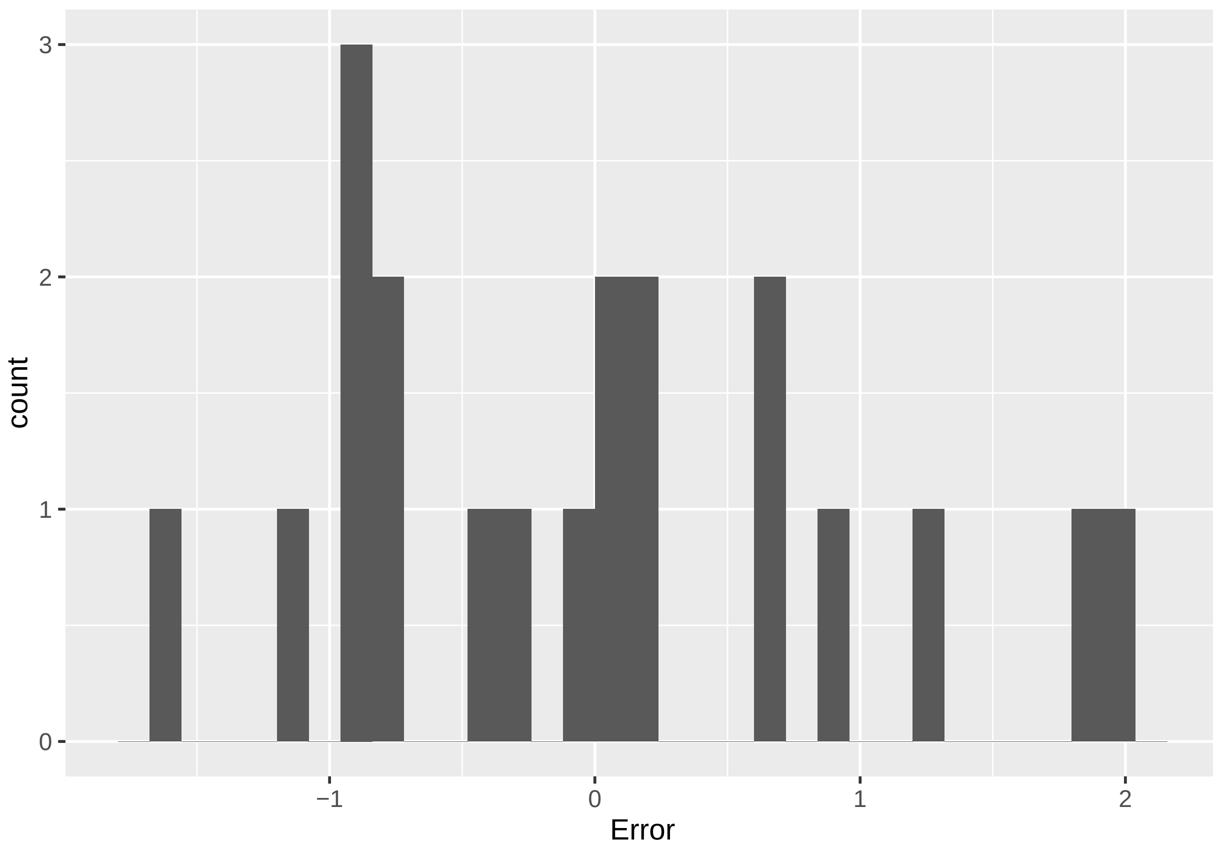
```r
plotDF<-Table1
plotDF[,'Type']<-'Train'
plotDF$Predict<-predict(LeastFit,plotDF[,1:2])
plotDF$Error<-plotDF$y-plotDF$Predict
ggplot(plotDF,aes(x=y,y=Predict,size=abs(Error)))+geom_point()+
  scale_size("Absolute Error")+geom_smooth(method="lm",se=F,size=1)
```




```r
sqrt(var(data.frame(plotDF %>% filter(Type=="Train") %>% select(Error))))/20
```

```
##            Error
## Error 0.04902635
```

```r
ggplot(plotDF,aes(x=Error)) + geom_histogram()
```

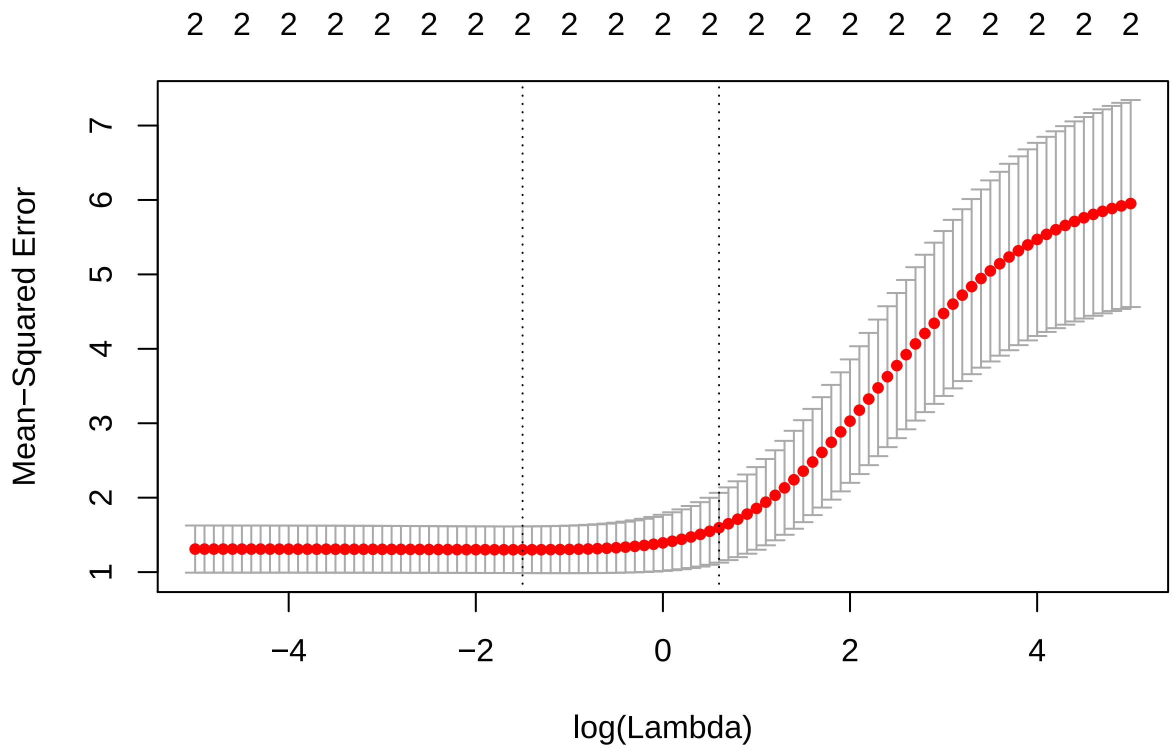



```r
sensitivities <- coef(LeastFit)
sensDF <- data.frame(Method = 0, Var = 0, Value=0)
sensDF[1:length(sensitivities),'Method'] <- "Least-Squares"
sensDF[1:length(sensitivities),'Var'] <- names(sensitivities)
sensDF[1:length(sensitivities),'Value'] <- (sensitivities)
rowStart <- length(sensitivities)+1
```

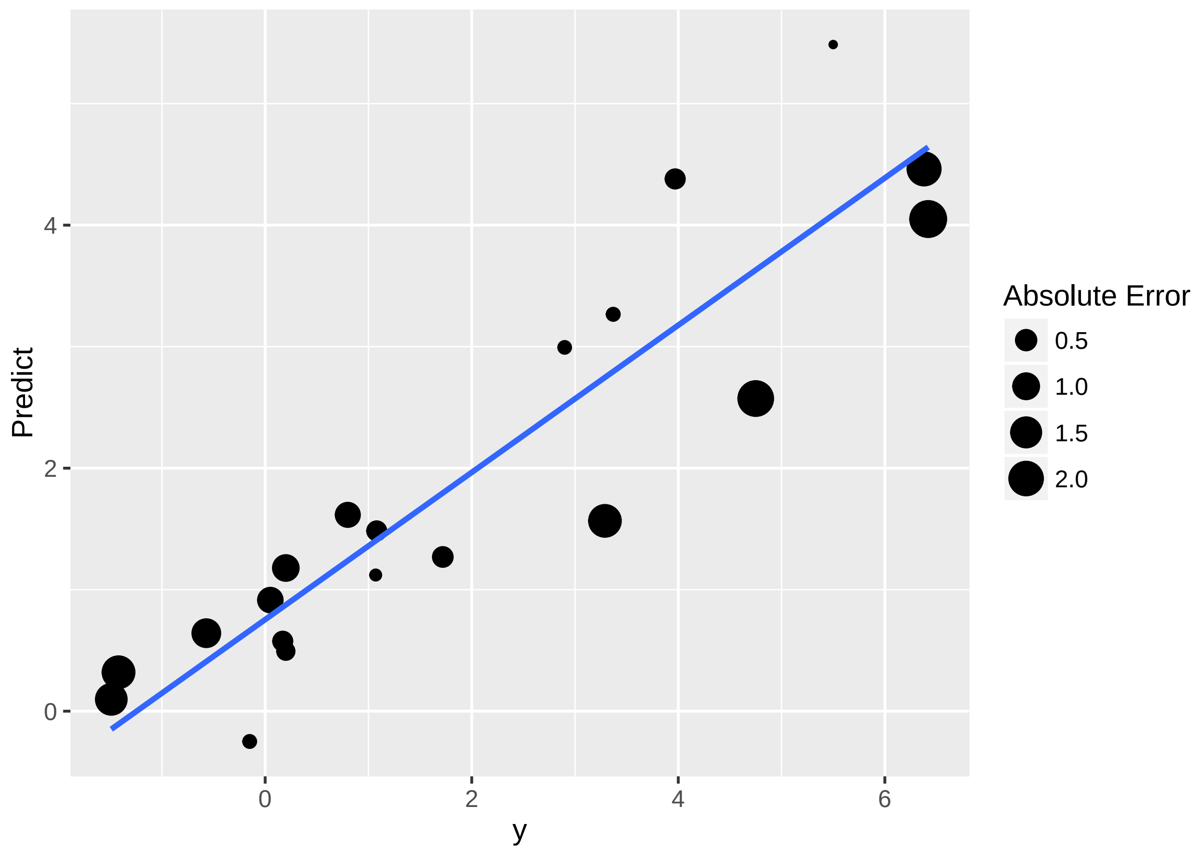
2. Ridge Regression

Ridge regression sets alpha=0, which adds damping to the coefficients

```r
crossValid <- cv.glmnet(as.matrix(Table1[,1:2]),
                        as.matrix(Table1$y),alpha = 0,
                        lambda=exp(seq(-5,5,by=0.1)))

plot(crossValid)
```



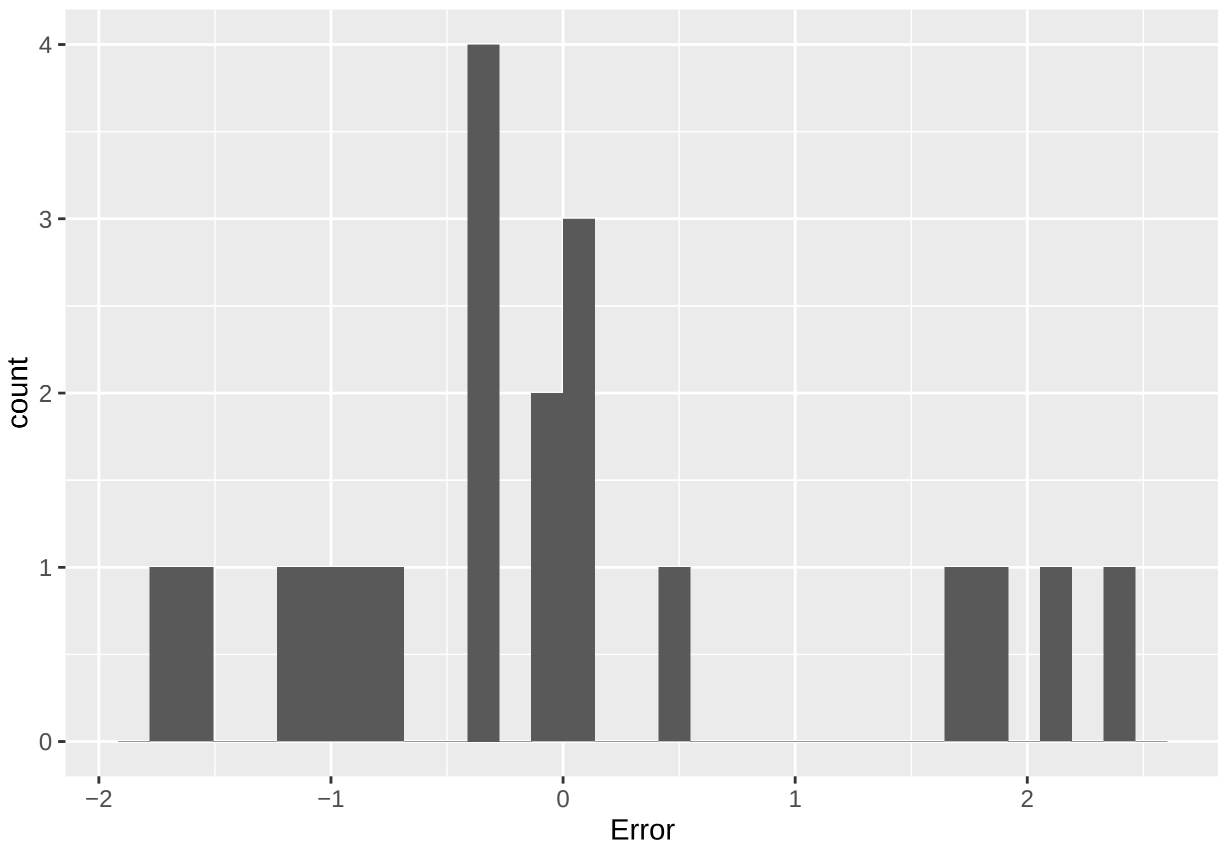

```
lambda <- crossValid$lambda.min
sensitivities <- coef(crossValid)
plotDF <- Table1
plotDF[,'Type'] <- 'Train'
plotDF[,"Predict" ]<- data.frame(Predict=predict(crossValid,as.matrix(plotDF[,1:2]),
                                                lambda=lambda))
plotDF$Error <- plotDF$y-plotDF$Predict
ggplot(plotDF,aes(x=y,y=Predict,size=abs(Error))) + geom_point() +
scale_size("Absolute Error") + geom_smooth(method="lm",se=F,size=1)
```




```
sqrt(var(data.frame(plotDF %>% filter(Type=="Train") %>% select(Error))))/20
```

```
##             Error
## Error 0.05982956
```

```
ggplot(plotDF,aes(x=Error)) + geom_histogram()
```

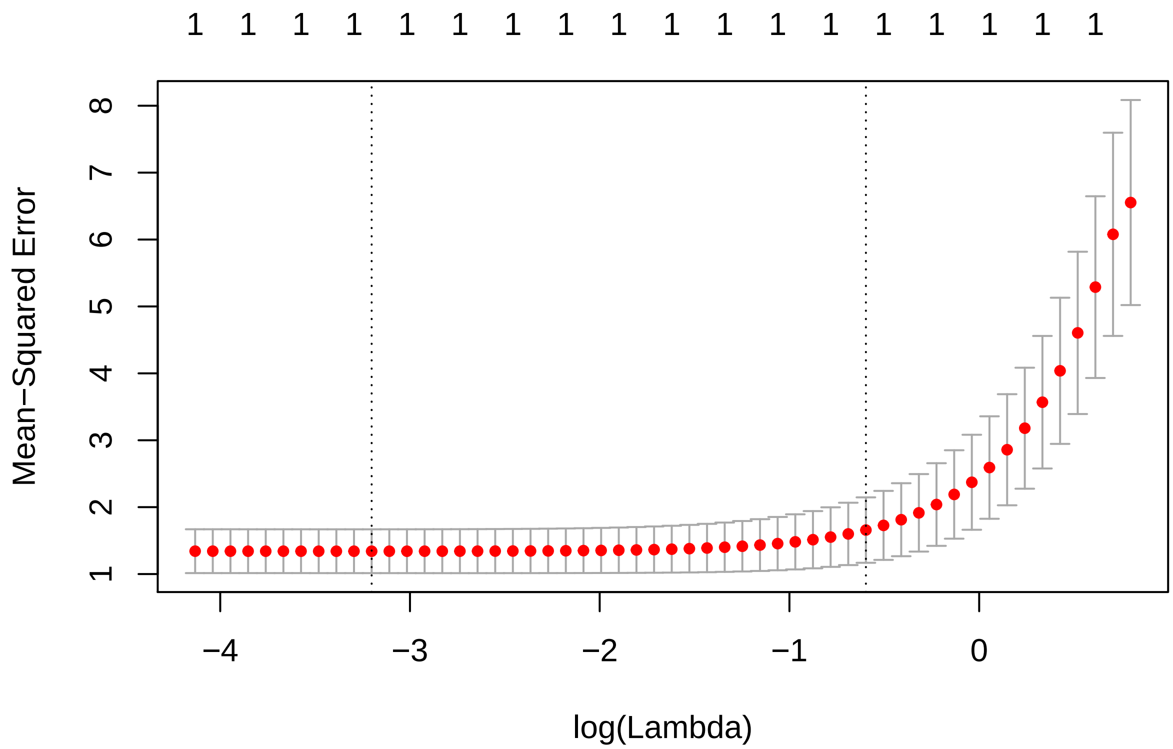



```
sensDF[rowStart:(rowStart + length(sensitivities)-1),'Method'] <- "Ridge"
sensDF[rowStart:(rowStart+length(sensitivities)-1),'Var']<-t(t(rownames(sensitivities)))
sensDF[rowStart:(rowStart +length(sensitivities)-1),'Value']<-as.numeric(sensitivities)
rowStart <- rowStart + length(sensitivities)
```

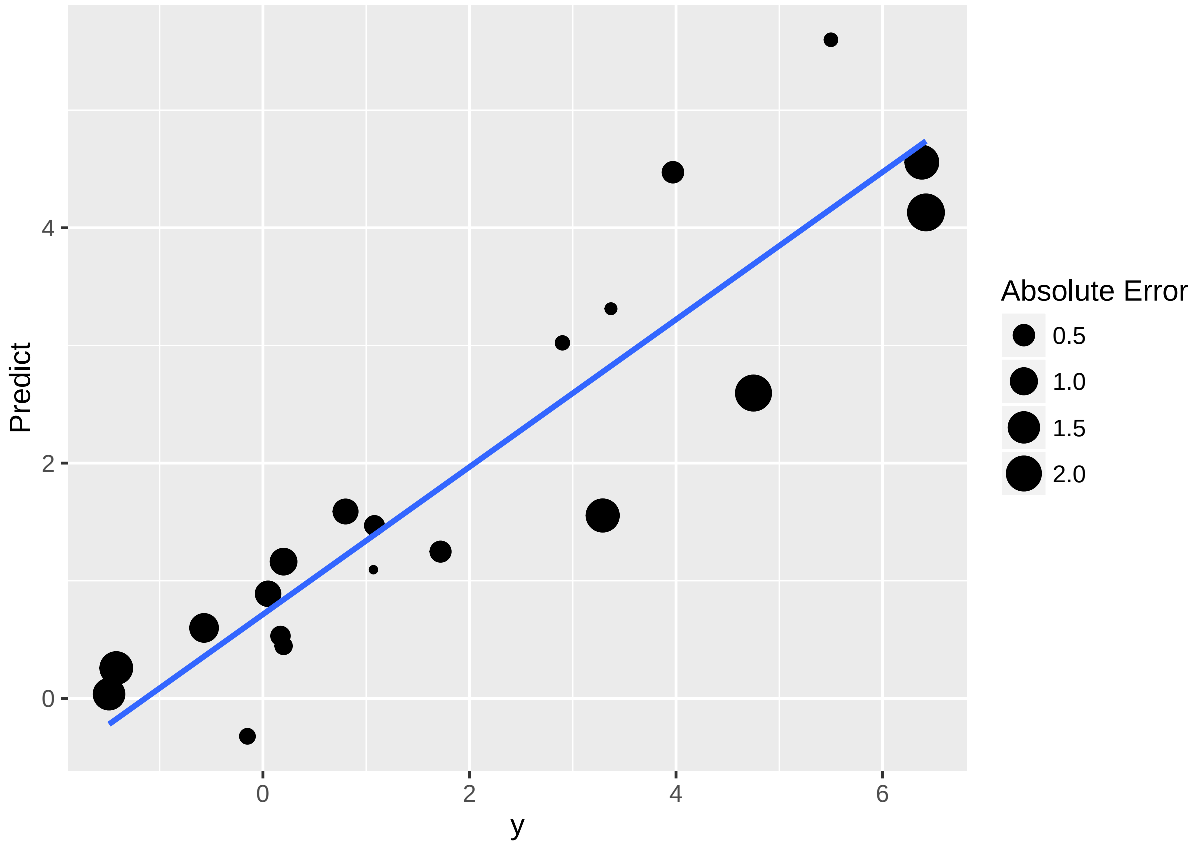
3. Lasso Regression

Lasso regression sets alpha=1

```
crossValid <- cv.glmnet(as.matrix(Table1[,1:2]),as.matrix(Table1$y),alpha = 1)
plot(crossValid)
```



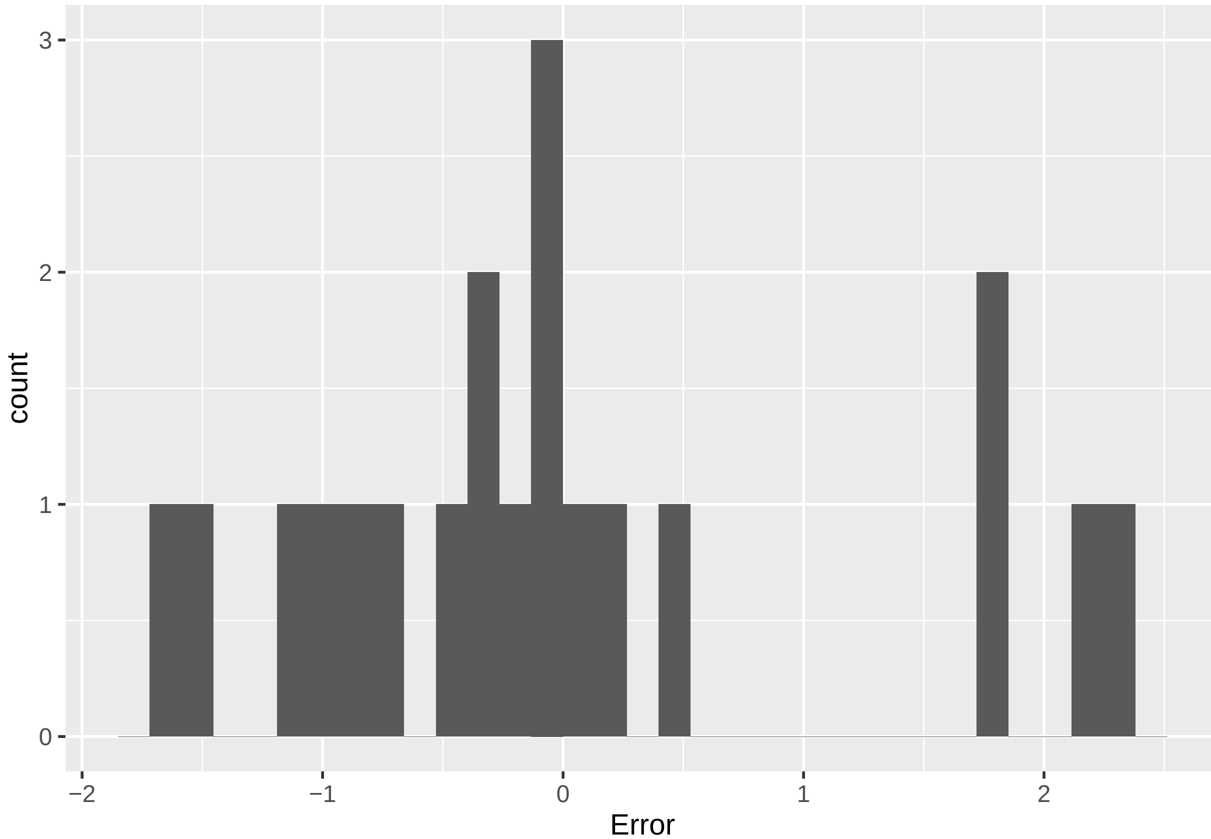

```
lambda <- crossValid$lambda.min
sensitivities <- coef(crossValid)
plotDF <- Table1
plotDF[,'Type'] <- 'Train'
plotDF[,"Predict" ]<- data.frame(Predict=predict(crossValid,as.matrix(Table1[,1:2]),
                                                 lambda=lambda))

plotDF$Error <- plotDF$y-plotDF$Predict
ggplot(plotDF,aes(x=y,y=Predict,size=abs(Error))) + geom_point() +
scale_size("Absolute Error") + geom_smooth(method="lm",se=F,size=1)
```




```
sqrt(var(data.frame(plotDF %>% filter(Type=="Train") %>% select(Error))))/20
```

```
##             Error
## Error 0.05832321
```

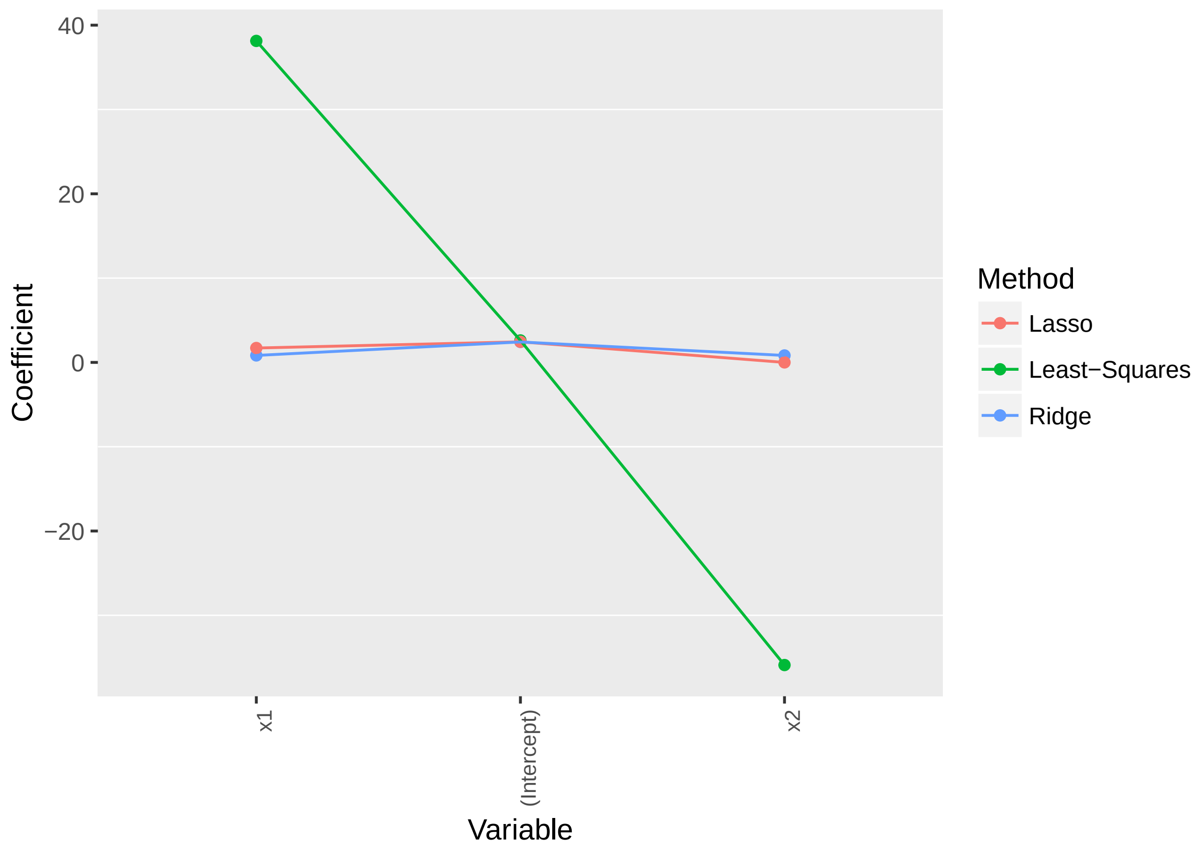
```
ggplot(plotDF,aes(x=Error)) + geom_histogram()
```




```r
sensDF[rowStart:(rowStart + length(sensitivities)-1),'Method'] <- "Lasso"
sensDF[rowStart:(rowStart+length(sensitivities)-1),'Var']<-t(t(rownames(sensitivities)))
sensDF[rowStart:(rowStart +length(sensitivities)-1),'Value']<-as.numeric(sensitivities)
rowStart <- rowStart + length(sensitivities)
```

Compare Methods

```r
ggplot(sensDF,aes(x=reorder(Var,-Value),y=Value,color=Method,group=Method)) +
  geom_point() + geom_line() +
  theme(panel.grid.major = element_blank(),
        axis.text.x = element_text(angle = 90, hjust = 1, size=8))+
  scale_x_discrete("Variable") + scale_y_continuous("Coefficient")
```



The Lasso and Ridge are both more bounded in their coefficents.

Problem 2

Derive the adjoint operator for the equation

$$-\bigtriangledown^2 \phi(x,y,z) + \frac{1}{L^2}\phi(x,y,z) = \frac{Q}{D}$$

$$\phi(0,y,z) = \phi(x,0,z) = \phi(x,y,0) = \phi(X,y,z) = \phi(x,Y,z) = \phi(x,y,Z) = C$$

Compute the sensitivity to the QOI:

$$QoI = \int_0^X dx \int_0^Y dy \int_0^Z dz \frac{D}{L^2}\phi(x,y,z)$$

for X,Y,Z,L,, and Q.

Derive the adjoint operator

Define the operator $\mathcal{L}$ as

$$\mathcal{L} = \bigtriangledown^2 + \frac{1}{L^2}$$

and the adjoint $\mathcal{L}^\dagger$ as

$$\mathcal{L}^\dagger = \bigtriangledown^2 + \frac{1}{L^2}$$

$$\phi^\dagger(0,y,z) = \phi^\dagger(x,0,z) = \phi^\dagger(x,y,0) = \phi^\dagger(X,y,z) = \phi^\dagger(x,Y,z) = \phi^\dagger(x,y,Z) = C$$

Setting:

$$\left|\frac{\delta\phi^\dagger}{\delta x}\right|_{x=0} = \left|\frac{\delta\phi}{\delta x}\right|_{x=0}$$

and

$$\left|\frac{\delta\phi^\dagger}{\delta x}\right|_{x=X} = \left|\frac{\delta\phi}{\delta x}\right|_{x=X}$$

and similar for the other two dimentions. Also define the inner product as:

$$(u,v) = \int_0^X dx \int_0^Y dy \int_0^Z dz \ uv$$

Proof, in order to prove that this is an adjoint operator for the above equation, it needs to be shown that $(\mathcal{L}\phi, \phi^\dagger) = (\phi, \mathcal{L}^\dagger\phi^\dagger)$.

Equivalent to:

$$\int_0^X dx \int_0^Y dy \int_0^Z dz \left(\phi^\dagger \bigtriangledown^2 \phi + \phi^\dagger \frac{\phi}{L^2}\right) = \int_0^X dx \int_0^Y dy \int_0^Z dz \left(\phi \bigtriangledown^2 \phi^\dagger + \phi \frac{\phi^\dagger}{L^2}\right) \tag{1}$$

The terms

$$\int_0^X dx \int_0^Y dy \int_0^Z dz \left(\phi^\dagger \frac{\phi}{L^2}\right) = \int_0^X dx \int_0^Y dy \int_0^Z dz \left(\phi \frac{\phi^\dagger}{L^2}\right)$$

are equal. For the other term with, $\bigtriangledown^2$, we can expand to:

$$\int_0^X dx \int_0^Y dy \int_0^Z dz \left(\phi^\dagger \left[\frac{\delta^2\phi}{\delta x^2} + \frac{\delta^2\phi}{\delta y^2} + \frac{\delta^2\phi}{\delta z^2}\right]\right)$$

Focusing on the x terms, noting that y and z will have the same derivation. Integration by parts, with $u = \phi^\dagger$, $du = \frac{\delta\phi^\dagger}{\delta x}dx$, and $v = \frac{\delta\phi}{dx}$, $dv = \frac{\delta^2\phi}{\delta x^2}dx$ yields.

$$\int_0^Y dy \int_0^Z dz \left(\int_0^X dx\ \phi^\dagger \frac{\delta^2\phi}{\delta x^2}\right) = \int_0^Y dy \int_0^Z dz \left(\left|\phi^\dagger \frac{\delta\phi}{\delta x}\right|_{x=0}^{x=X} - \int_0^X dx\ \frac{\delta\phi}{\delta x}\frac{\delta\phi^\dagger}{\delta x}dx\right)$$

Performing another integration by parts, with $u = \frac{\delta\phi^\dagger}{\delta x}$, $du = \frac{\delta^2\phi^d ag}{\delta x}dx$ and, $v = \phi$, $dv = \frac{\delta\phi}{\delta x}dx$.

$$= \int_0^Y dy \int_0^Z dz \left(\left|\phi^\dagger \frac{\delta\phi}{\delta x}\right|_{x=0}^{x=X} - \left|\phi\frac{\delta\phi^\dagger}{\delta x}\right|_{x=0}^{x=X} + \int_0^X dx\ \phi\frac{\delta^2\phi^\dagger}{\delta x^2}dx\right)$$

At the boundaries, both ϕ and $\phi^\dagger$ are a constant, and the derivatives of both at the boundaries are equal, and therefore those terms cancel, leaving

$$\int_0^Y dy \int_0^Z dz \left(\int_0^X dx\ \phi\frac{\delta^2\phi^\dagger}{\delta x^2}dx\right)$$

which is equal to the x component of the $\bigtriangledown^2$ term of the RHS of equation 1 above.

Compute the sensitivity to the QoI:

I don't know if this is the right way to go about this, but if you modify the first equation listed in this problem to:

$$\phi = L^2\left[\frac{Q}{D} + \bigtriangledown^2\phi\right]$$

and use it as a substitution, then we get something like this,

$$\begin{aligned}
\text{QoI} &= \int_0^X dx \int_0^Y dy \int_0^Z dz\ \frac{D}{L^2}\phi \\
&= \int_0^X dx \int_0^Y dy \int_0^Z dz\ \frac{D}{L^2}\left(L^2\left[\frac{Q}{D} + \bigtriangledown^2\phi\right]\right) \\
&= \int_0^X dx \int_0^Y dy \int_0^Z dz\ \left[Q + D\bigtriangledown^2\phi\right]
\end{aligned}$$

If Q and D both are independent of space, then

$$\text{QoI} = QXYZ + D\left[\int_0^Y dy \int_0^Z dz\ \left|\frac{\delta\phi}{\delta x}\right|_{x=0}^{x=X} + \int_0^X dx \int_0^Z dz\ \left|\frac{\delta\phi}{\delta y}\right|_{y=0}^{y=Y} + \int_0^X dx \int_0^Z dz\ \left|\frac{\delta\phi}{\delta z}\right|_{z=0}^{z=Z}\right]$$

I'm not sure how the adjoint helps me though.

Problem 3

For the random variable X ∼ N(0,1) draw fifty samples and generate histograms using the following sampling techniques

(a) Simple random sampling

(b) Stratifid sampling

(c) A van der Corput sequence of base 2

(d) A van der Corput sequence of base 3

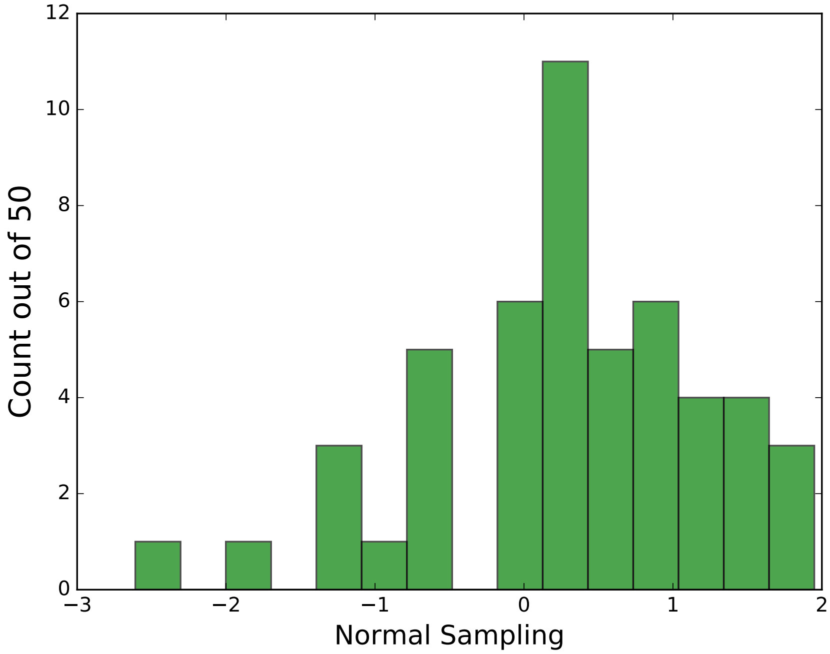
Simple random sampling samples U(0,1) and plugs this value into the inverse CDF. Stratified sampling separates U(0,1) into equal bins and samples "Randomly" in each bin. Van der Corput sequences divides an interval into a a number of equal subintervals.

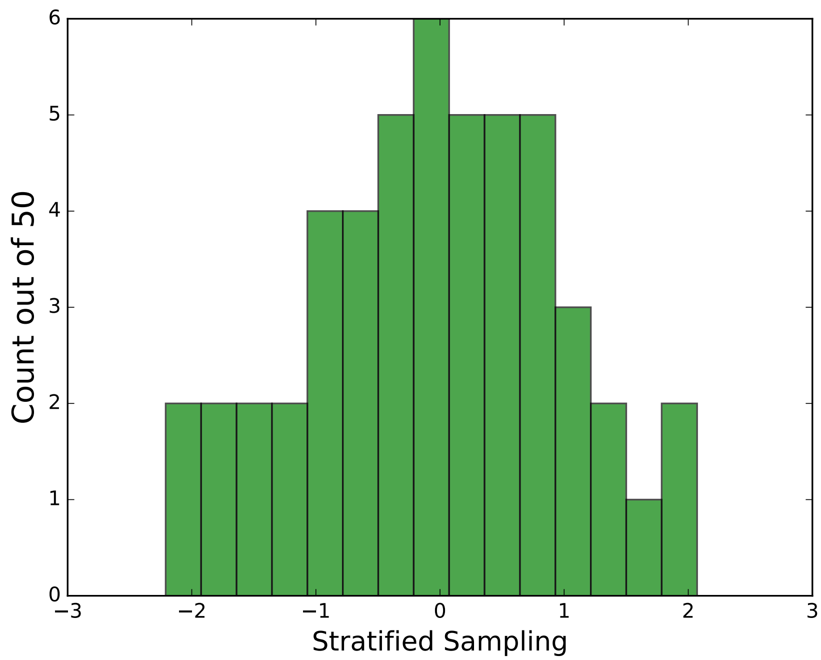
For example, the ordinary van der Corput sequence in base 3 is given by 1/3, 2/3, 1/9, 4/9, 7/9, 2/9, 5/9, 8/9, 1/27.

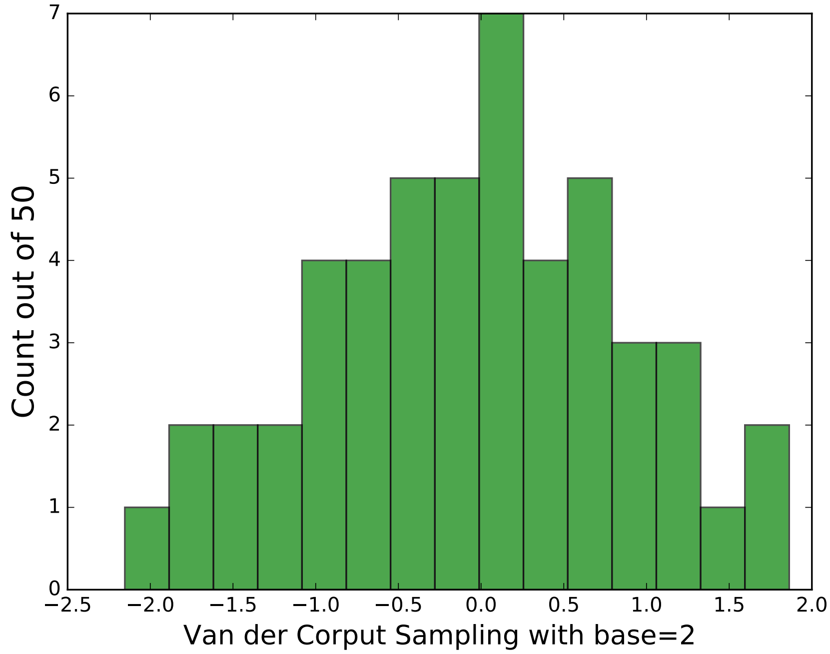
Listing 1: Script for Problem

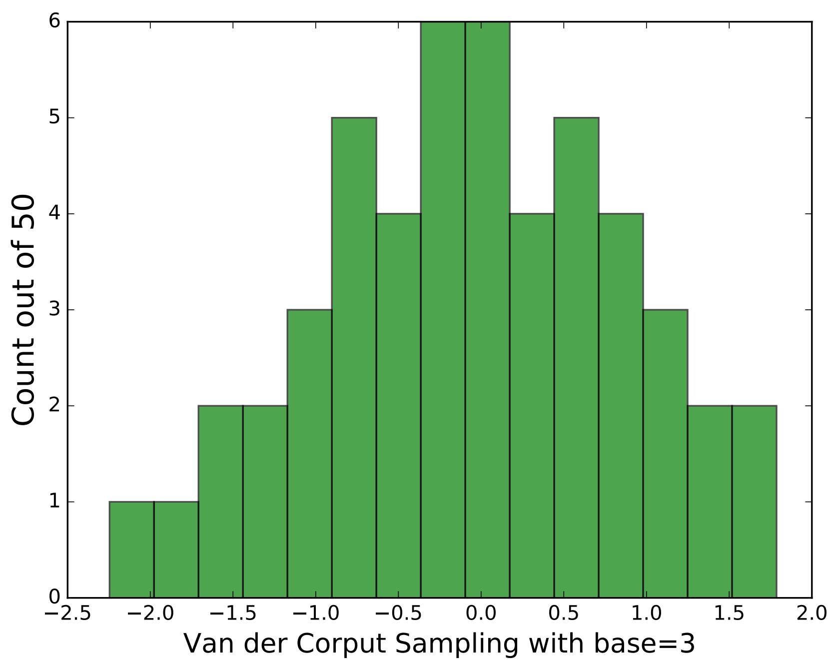
```
#!/usr/bin/env python3

################################################################
#################### Import packages #########################
################################################################

import numpy as np
import matplotlib.pyplot as plt
import time
start_time = time.time()
from scipy.stats import norm

################################################################
####################### Functions ############################
################################################################

#Van der Corput sequence function found online
def vdc(n, base=2):
    vdc, denom = 0,1
    while n:
        denom *= base
        n, remainder = divmod(n, base)
        vdc += remainder / denom
    return vdc

################################################################
##################### Calculations ###########################
################################################################

#Make sure Nstrata <= N

N=50  #samples
```

```
Nbins=15  #hist plot
Nstrata=49
filename="V2Norm.pdf"
#Stratified or Normal or Van der Corput
Xlabel="Van der Corput Sampling with base=2"
RandomNumbers=[]
vanBase=2;van=True

#Sampling for normal and stratified
if not van:
    Nloop=int(N/Nstrata)*Nstrata
    for i in range(0,int(N/Nstrata)):
        for j in range(0,Nstrata):
            RandomNumbers.append(np.random.uniform(low=j/Nstrata,
                                    high=(j+1)/Nstrata,size=1))
    #If N/Nstrata doesn't divide evenly
    if Nloop<N:
        for j in range(0,N-Nloop):
            RandomNumbers.append(np.random.uniform(low=j/Nstrata,
                                high=(j+1)/Nstrata,size=1))
#Sampling for van
if van:
    for i in range(0,N):
        RandomNumbers.append(vdc(i+1,vanBase))

#Sample the inverse of the CDF of the standard normal
#distribution
Samples=norm.ppf(RandomNumbers)

#Generate histogram
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlabel(Xlabel,fontsize=16)
ax.set_ylabel('Count out of '+str(N),fontsize=18)
ax.hist(Samples,Nbins,color='green',alpha=0.7,edgecolor='black')
#ax.set_xlim(-500,500)
plt.savefig(filename)

#################### Time To execute ################

print("--- %s seconds ---" % (time.time() - start_time))
```


Count out of 50
Normal Sampling
0
2
4
6
8
10
12
−3
−2
−1
0
1
2


Count out of 50
Stratified Sampling
0
1
2
3
4
5
6
−3
−2
−1
0
1
2
3


Count out of 50
Van der Corput Sampling with base=2
0
1
2
3
4
5
6
7
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Count out of 50
Van der Corput Sampling with base=3
0
1
2
3
4
5
6
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Problem 4

Consider the Rosenbrock function $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$. Assume that $x = 2t - 1$, where T $\sim$ $B(3, 2)$ and $y = 2s - 1$, where S $\sim$ $B(1.1, 2)$. Estimate the probability that $f(x, y)$ is less than 10 using:

(a) a first-order second-moment reliability method

(b) Latin hypercube sampling using 50 points

(c) A Halton sequence using 50 points

Compare this with the probability you calculate using 10^5 random samples. (*Hint: Matlab has a built-in function for sampling beta R.V.'s "betard"*).

A first-order second-moment reliability method

Will use a gaussian approximation as shown in section 7.3 in the course notes (FORM).

In this method the performance function ($Z(x, y)$) is a function of the random variables x and y, and is defined such that the failure surface is the location where $Z = 0$ (top of page 119 Ch 7) such that $Z < 0$ represents failure and $Z > 0$ represents success. For the case above, this would mean that our performance function is:

$$\begin{aligned} Z &= 10 - f(x, y) \\ &= 10 - (1 - x)^2 + 100(y - x^2)^2 \end{aligned}$$

Next, the probability of failure is defined as:

$$p_{fail} = 1 - \Phi\left(\frac{\mu_Z}{\sigma_Z}\right)$$

Where Φ is the CDF for a standard normal, μ_Z and σ_Z are the mean and standard deviation of Z. As a reminder, if Z were a standard normal, defined such that any part of Z that is less than 0 is failure. Then because $\mu_Z = 0$ there would be a 50% chance of failure. If Z were a non standard normal, then the distance from zero would be normalized (by dividing by σ_Z) to units of σ, and as μ_Z increases, the chance of failure would continue to decrease, which make sense, as the mean of the performance function moves further and further away from the failure point ($Z = 0$), the probability of failure decreases.

I am trying to spell this out for myself, because I was really confused about this. There are two things I would like to point out to future self. First, Z may not be normal, which is why this method is only exact if Z is normal, otherwise its an approximation. Second, if μ_Z were less than 0, then the equation for the probability of failure should (I think) change to

$$p_{fail} = 0.5 + \Phi\left(\frac{|\mu_Z|}{\sigma_Z}\right)$$

Also third, I am not sure if Z has to be a typical PDF or CDF, will let you know after I do some of the math McClarren gave.

The mean for Z was defined as

$$\mu_Z \approx g(\mu_{x,y})$$

where g is the function we defined as Z (first part of the taylor expansion of Z) evaluated at the mean values of x and y. The standard deviation of Z was defined as the the second part of the taylor expansion of Z (without covariances because we are going to assume that all random variables are independent), namely

$$\sigma_Z^2 = \left(\left| \frac{\delta g}{\delta x} \right|_{\mu_x} \sigma_x \right)^2 + \left(\left| \frac{\delta g}{\delta y} \right|_{\mu_y} \sigma_y \right)^2$$

For what I defined as Z above,

$$\frac{\delta g}{\delta x} = -2(x-1) - 400x(x^2 - y)$$
$$\frac{\delta g}{\delta y} = -200(y - x^2)$$

Also for a beta R.V

$$\mu = \frac{a}{a+b}$$
$$\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}$$

Calculations

For the random variable t used in $x = 2t - 1$

mean

$$\mu_t = \frac{3}{3+2} = \mathbf{0.6}$$
$$\mu_x = 2 * \mathbf{0.6} - 1 = \boxed{0.2}$$

standard deviation

$$\sigma_t^2 = \frac{3 \cdot 2}{(3+2)^2(3+2+1)} = \mathbf{0.04}$$
$$\sigma_x^2 = \left| \frac{\delta x}{\delta t} \right|_{\mu_T}^2 \sigma_t^2$$
$$= 2^2 \cdot \mathbf{0.04} = \boxed{0.16}$$

For the random variable s used in $y = 2s - 1$

mean

$$\mu_s = \frac{1.1}{1.1+2} = \mathbf{0.354839}$$
$$\mu_y = 2 * \mathbf{0.354839} - 1 = \boxed{-0.290323}$$

standard deviation

$$\sigma_s^2 = \frac{1.1 \cdot 2}{(1.1+2)^2(1.1+2+1)} = \mathbf{0.055836}$$
$$\sigma_y^2 = \left|\frac{\delta y}{\delta s}\right|_{\mu_s}^2 \sigma_S s^2$$
$$= 2^2 \cdot \mathbf{0.055836} = \boxed{0.223345}$$

For the partial derivative terms

$$\left|\frac{\delta g}{\delta x}\right|_{\mu_X} = -2(x-1) - 400x(x^2-y) = -2(0.2-1) - 400 \cdot 2(0.2^2 - (-0.290323)) = \boxed{-24.8258}$$

$$\left|\frac{\delta g}{\delta y}\right|_{\mu_y} = -200(y - x^2) = -200(-0.290323 - 0.2^2) = \boxed{66.0646}$$

For Z and the probability of failure

$$\mu_Z = 10 - (1-0.2)^2 + 100(-0.290323 - 0.2^2)^2 = \boxed{20.2713}$$

$$\sigma_Z^2 = 24.8^2 \cdot 0.16 + 66.0646^2 \cdot 0.223345 = 1073.41$$
$$\sigma_Z = \boxed{32.7629}$$

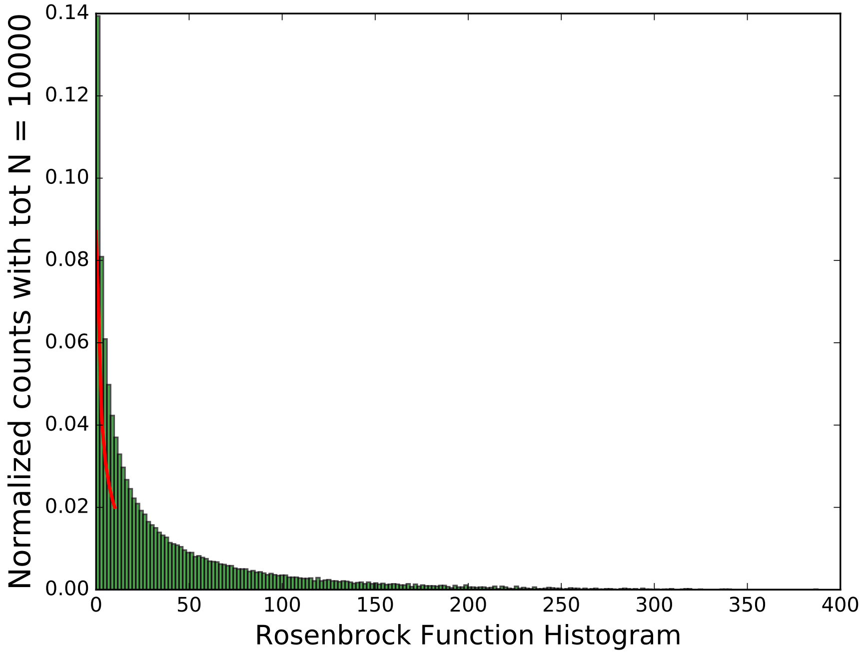
$$p_{fail} = 1 - \Phi\left(\frac{20.27}{32.7629}\right)$$
$$= \boxed{\mathbf{0.268}}$$

Listing 2: Script for Problem

```
#!/usr/bin/env python3

###################################################################
#################### Import packages ##############################
###################################################################

import time
start_time = time.time()
import Functions as fun

```

```
##################################################################
##################### Calculations ###########################
##################################################################

N=10000  #Samples
Nbins=200 #Hist Plot
Nstrata=1000
filename="Histf.pdf"

Xlabel="Stratified Sampling both samples"
RandomNumbersX=fun.Rstrat(N,Nstrata)
RandomNumbersY=fun.Rstrat(N,Nstrata)

Samplest=fun.beta.ppf(RandomNumbersX,3,2)
Sampless=fun.beta.ppf(RandomNumbersY,1.1,2)

ut=fun.betau(3,2)
us=fun.betau(1.1,2)
ot=fun.betao(3,2)
os=fun.betao(1.1,2)
ux=fun.X(ut)
uy=fun.Y(us)


print("X mean is "+str(ux))
print("Y mean is "+str(uy))
print("X sig is "+str((ot**0.5)*2))
print("Y sig is "+str((os**0.5)*2))
print("Mean Z is "+str(fun.Z())
quit()

X=fun.X(Samplest)
Y=fun.Y(Sampless)

f=fun.Rosen(X,Y)

Xlabel="Rosenbrock Function Histogram"
(I)=fun.HIST(Xlabel,f,Nbins,filename,N)

PGreater=sum(i<10 for i in f)/N

print("The probability of being less than 10 is: "+str(PGreater[0]))




##################### Time To execute ################

print("--- %s seconds ---" % (time.time() - start_time))
```

The output is 0.3795 probability of being less than 10. Here is a PDF of the function f generated from the code above.


Normalized counts with tot N = 10000
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250
300
350
400
Rosenbrock Function Histogram