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	-1.44	-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.

Problem 1 continued on next page...

Problem1

Paul Mendoza

December 9, 2016

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

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

```
Table1
```

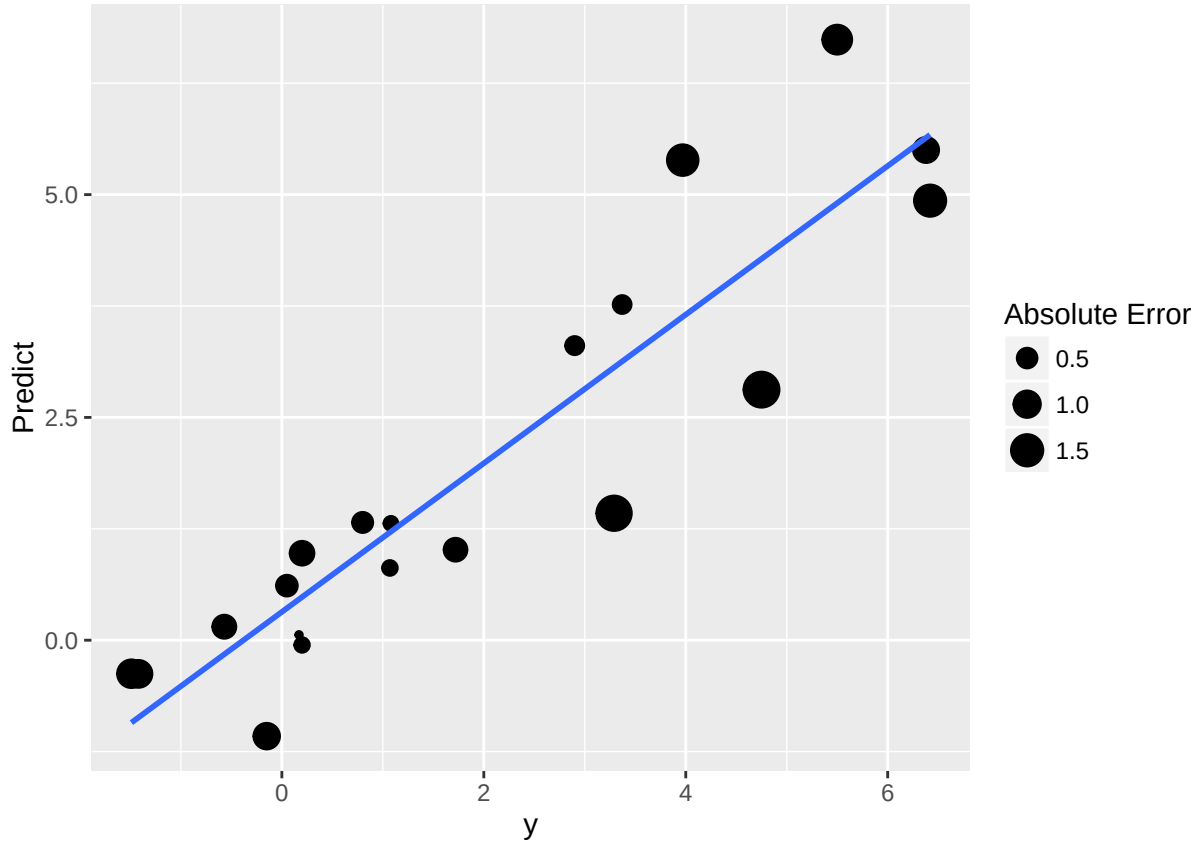
```
##       x1    x2     y
## 1   0.99  0.98  6.42
## 2  -0.75 -0.76  0.20
## 3  -0.50 -0.48  0.80
## 4  -1.08 -1.08 -0.57
## 5   0.09  0.09  4.75
## 6  -1.28 -1.27 -1.42
## 7  -0.79 -0.79  1.07
## 8  -1.17 -1.17  0.20
## 9  -0.57 -0.57  1.08
## 10 -1.62 -1.62 -0.15
## 11  0.34  0.35  2.90
## 12  0.51  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 -1.44 -1.49
```

1. Least Squares

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

```
##
## Call:
## lm(formula = y ~ x1 + x2, data = Table1)
##
## Coefficients:
## (Intercept)           x1           x2
##       2.607        9.707       -7.433
```

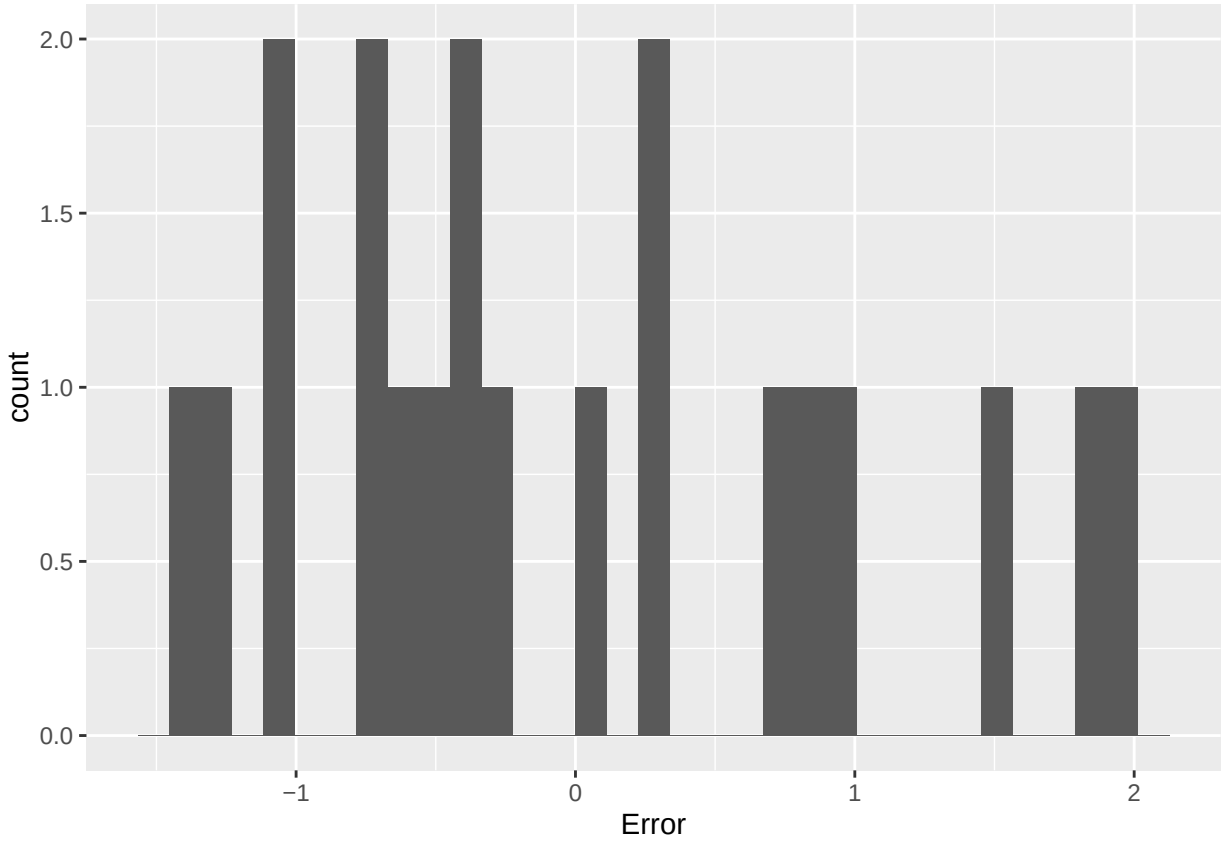
```
plotDF<-Table1
plotDF[,'Type']<-'Train'
plotDF$Predict<-predict(LeastFit,plotDF[,1:2])
plotDF$Error<-plotDF$y-plotDF$Predict
ggplot(plotDF,aes(x=y,y=Predict,size=abs(Error)))+geom_point()+
  scale_size("Absolute Error")+geom_smooth(method="lm",se=F,size=1)
```




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

```
##             Error
## Error 0.05088412
```

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

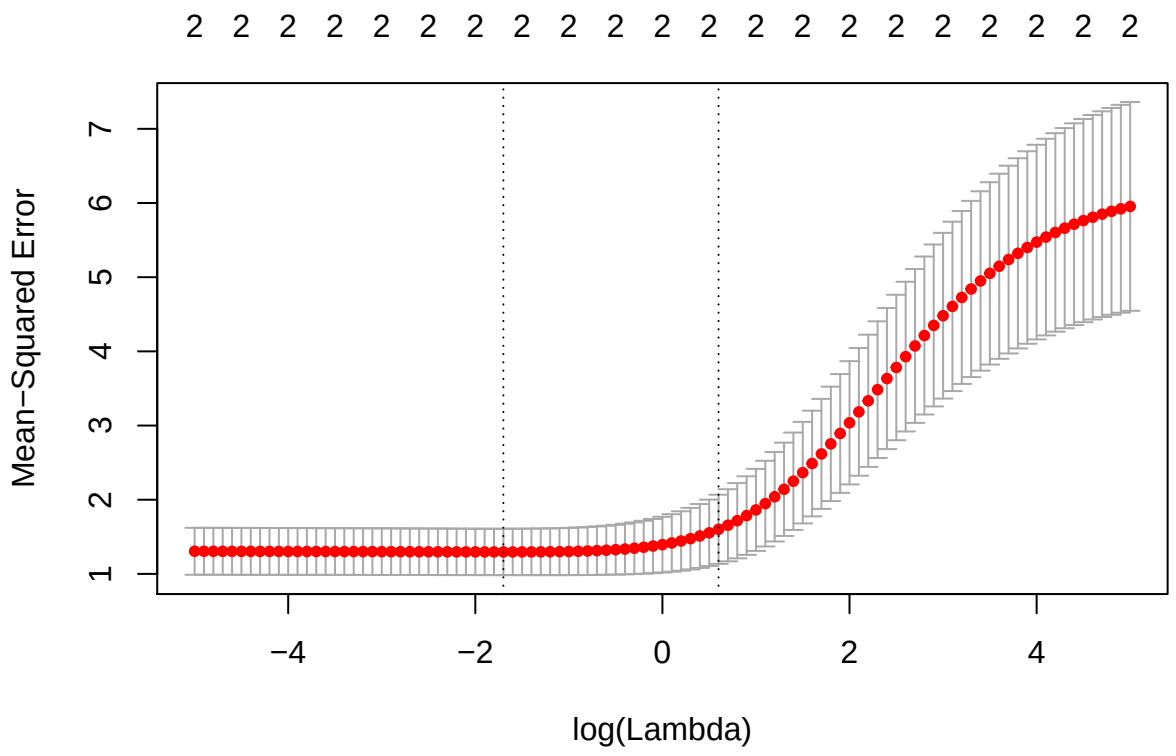



```
sensitivities <- coef(LeastFit)
sensDF <- data.frame(Method = 0, Var = 0, Value=0)
sensDF[1:length(sensitivities),'Method'] <- "Least-Squares"
sensDF[1:length(sensitivities),'Var'] <- names(sensitivities)
sensDF[1:length(sensitivities),'Value'] <- (sensitivities)
rowStart <- length(sensitivities)+1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

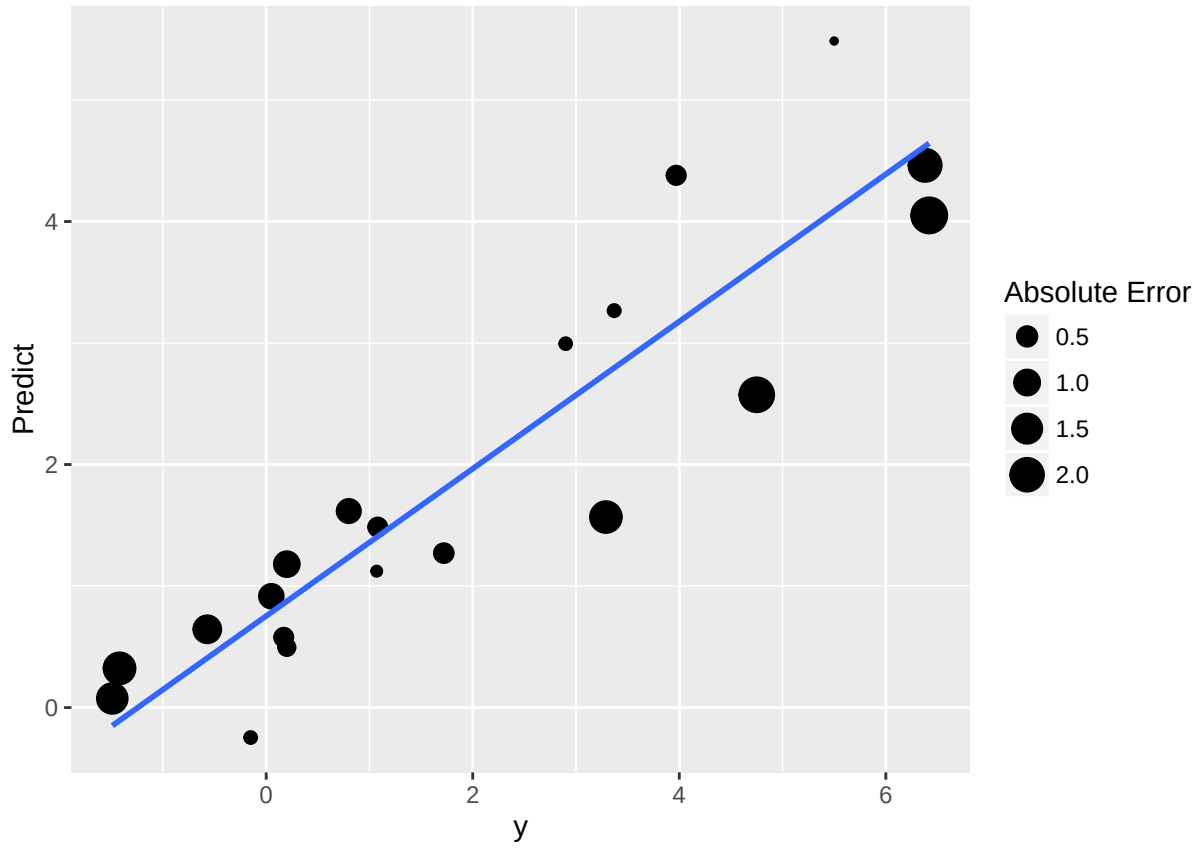
2. Ridge Regression

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

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

plot(crossValid)
```



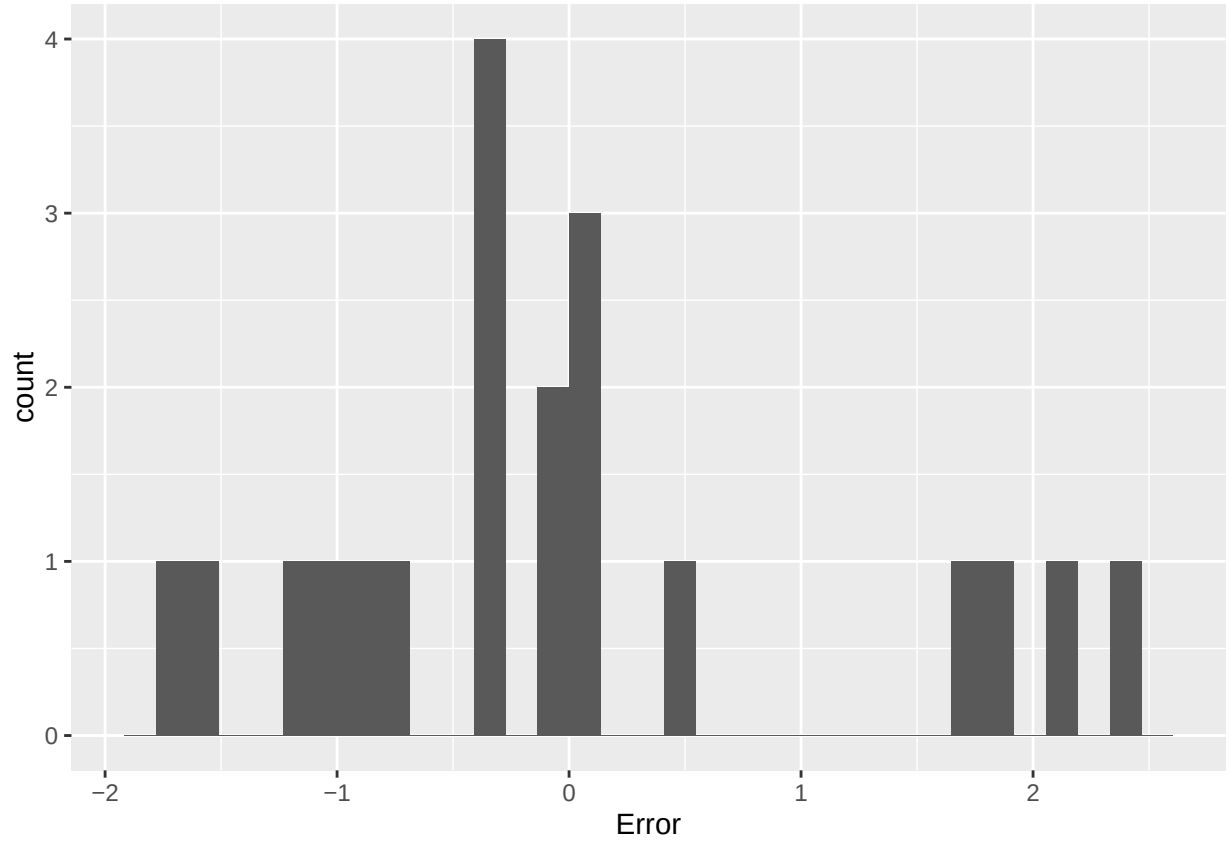

```
lambda <- crossValid$lambda.min
sensitivities <- coef(crossValid)
plotDF <- Table1
plotDF[,'Type'] <- 'Train'
plotDF[,"Predict" ]<- data.frame(Predict=predict(crossValid,as.matrix(plotDF[,1:2]),
                                                 lambda=lambda))
plotDF$Error <- plotDF$y-plotDF$Predict
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 0.05976789
```

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

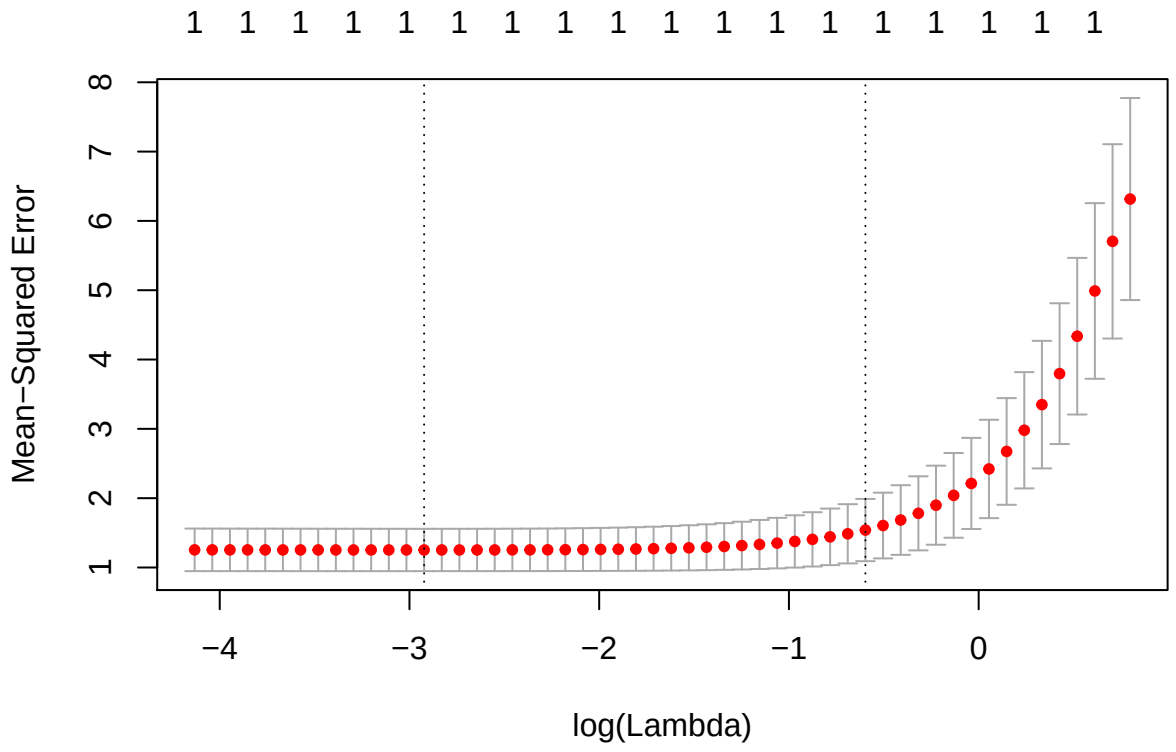



```
sensDF[rowStart:(rowStart + length(sensitivities)-1),'Method'] <- "Ridge"
sensDF[rowStart:(rowStart+length(sensitivities)-1),'Var']<-t(t(rownames(sensitivities)))
sensDF[rowStart:(rowStart +length(sensitivities)-1),'Value']<-as.numeric(sensitivities)
rowStart <- rowStart + length(sensitivities)
```

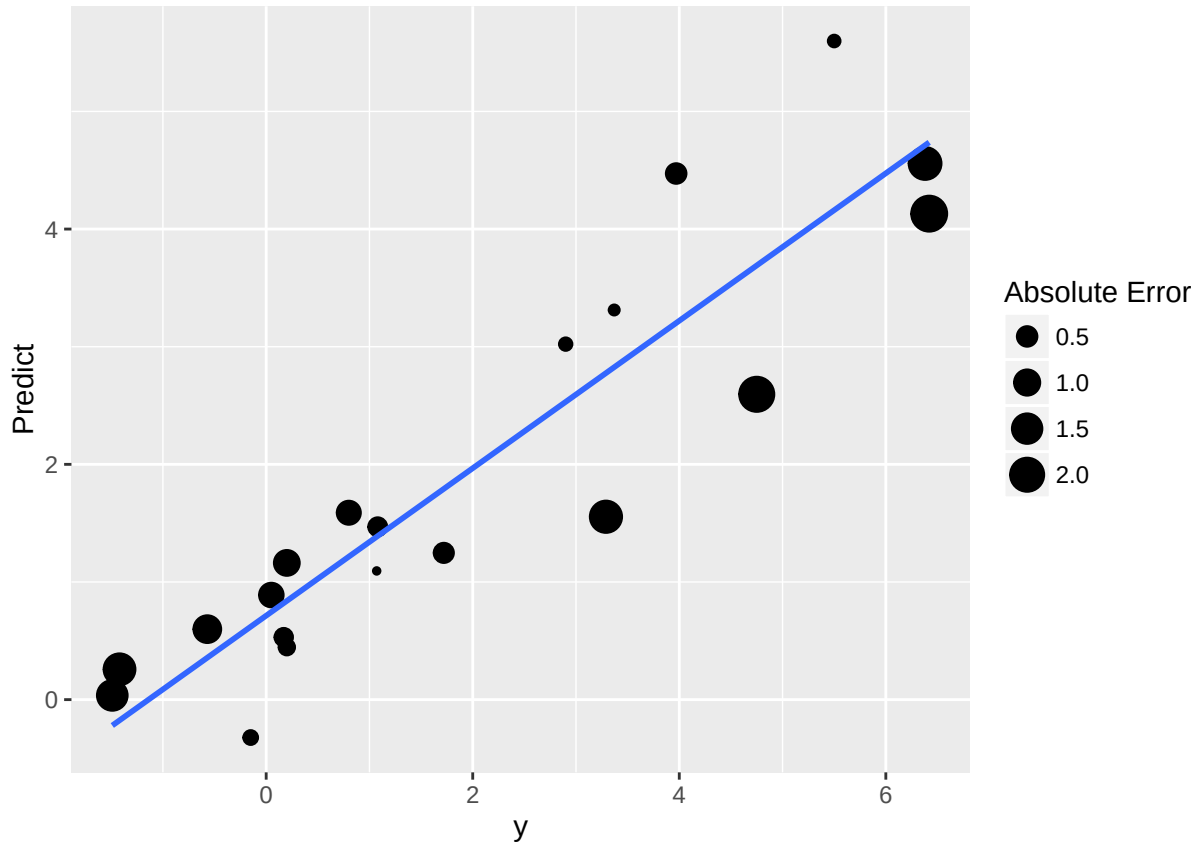
3. Lasso Regression

Lasso regression sets alpha=1

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



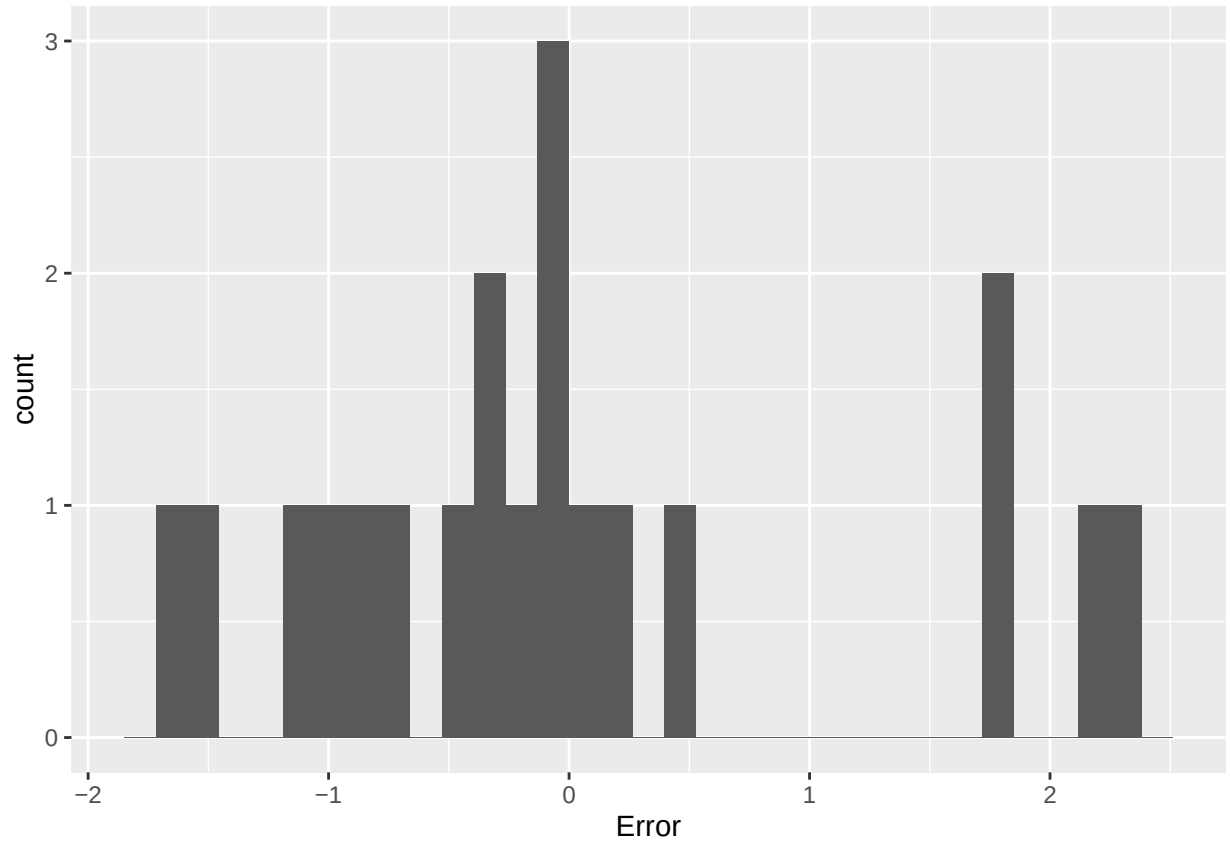

```
lambda <- crossValid$lambda.min
sensitivities <- coef(crossValid)
plotDF <- Table1
plotDF[,'Type'] <- 'Train'
plotDF[,"Predict" ]<- data.frame(Predict=predict(crossValid,as.matrix(Table1[,1:2]),
                                                 lambda=lambda))

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
```

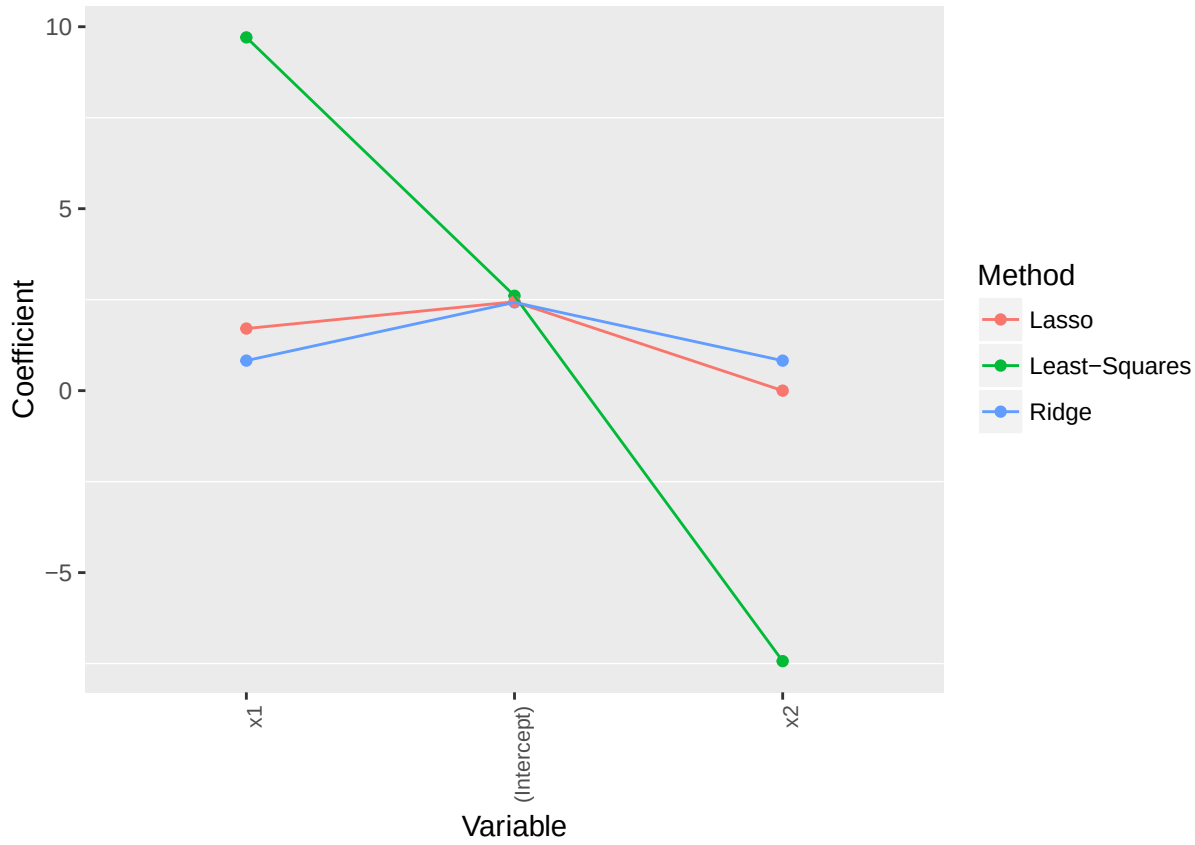
```
ggplot(plotDF,aes(x=Error)) + geom_histogram()
```




```
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

```
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,D and Q.

Derive the adjoint operator

See Ch 6 for help more background. Define the operator $\mathcal{L}$ as

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

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

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

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

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 (all terms have a D in them and cancel):

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

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 (removing the negative sign for simplicity):

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

Problem 2 continued on next page...

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:

The QoI has been defined as,

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

with $p = \frac{D}{L^2}$. Recall the original system being,

$$\mathcal{L}\phi = Q.$$

If we define an adjoint system as,

$$\mathcal{L}^\dagger\phi^\dagger = p,$$

then the QoI can be represented as,

$$QoI = (\phi, p) = (Q, \phi^\dagger)$$

This is because,

$$(\mathcal{L}\phi, \phi^\dagger) = (\phi, \mathcal{L}^\dagger\phi^\dagger)$$
$$(Q, \phi^\dagger) = (\phi, p)$$

The first line was proved in the first part of the problem, and the second line substitutes q for $\mathcal{L}\phi$ on the LHS of the equation and p for $\mathcal{L}^\dagger\phi^\dagger$ on the RHS.
If the solution for $\phi^\dagger$ were known, then this problem would be a lot easier. Lets see if we can find it.

$$\mathcal{L}^\dagger\phi^\dagger = p$$
$$-D \bigtriangledown^2 \phi^\dagger + \frac{D}{L^2}\phi^\dagger = \frac{D}{L^2}$$
$$\phi^\dagger = 1 + L^2 \bigtriangledown^2 \phi^\dagger$$

Where the solution is $\phi^\dagger = 1 + exp(\frac{x}{L}) + exp(\frac{y}{L}) + exp(\frac{z}{L})$

Problem 2 continued on next page...

Now the QoI is,

$$\begin{aligned} QoI &= (Q, \phi^\dagger) \\ &= \int_0^X dx \int_0^Y dy \int_0^Z dz\ Q\phi^\dagger \\ &= \int_0^X dx \int_0^Y dy \int_0^Z dz\ Q\left(1 + e^{\frac{x}{L}} + e^{\frac{y}{L}} + e^{\frac{z}{L}}\right) \\ &= QXYZ + QYZL(e^{X/L} - 1) + QXZL(e^{Y/L} - 1) + QXYL(e^{Z/L} - 1) \end{aligned}$$

The sensitivity to the QoI will be determined with a simple derivative of the QoI with respect to particular variables.

$$\frac{\delta QoI}{\delta X} = \boxed{QYZ + QYZe^{X/L} + QZL(e^{Y/L} - 1) + QYL(e^{Z/L} - 1)}$$

$$\frac{\delta QoI}{\delta Y} = \boxed{QXZ + QZL(e^{X/L} - 1) + QXZe^{Y/L} + QXL(e^{Z/L} - 1)}$$

$$\frac{\delta QoI}{\delta Z} = \boxed{QXY + QYL(e^{X/L} - 1) + QXL(e^{Y/L} - 1) + QXYe^{Z/L}}$$

$$\frac{\delta QoI}{\delta L} = \boxed{Q(YZe^{X/L}(1 - e^{-X/L} - \frac{X}{L}) + XZe^{Y/L}(1 - e^{-Y/L} - \frac{Y}{L}) + XYe^{Z/L}(1 - e^{-Z/L} - \frac{Z}{L}))}$$

$$\frac{\delta QoI}{\delta D} = \boxed{0}$$

$$\frac{\delta QoI}{\delta Q} = \boxed{XYZ + YZL(e^{X/L} - 1) + XZL(e^{Y/L} - 1) + XYL(e^{Z/L} - 1)}$$

Problem 3

For the random variable X $\sim$ 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

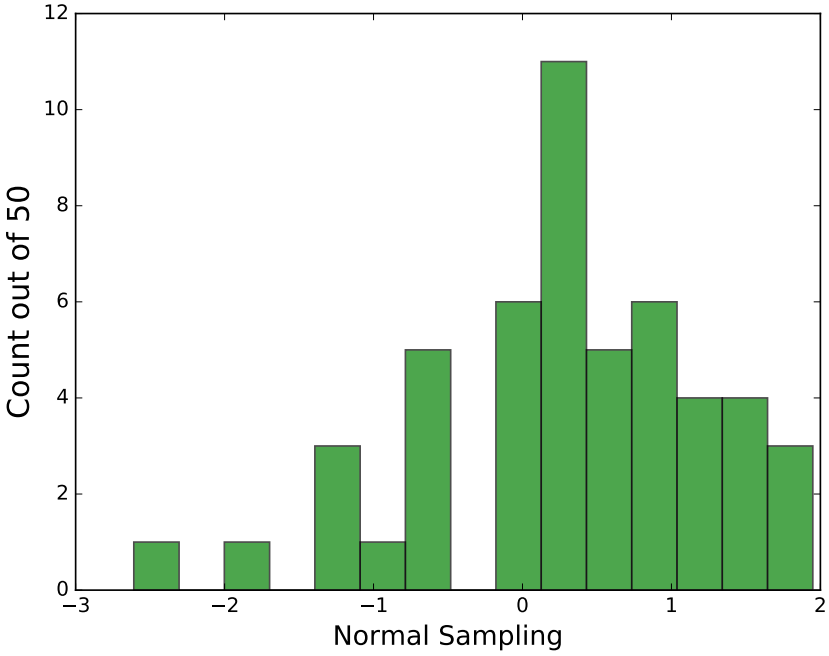
Simple random sampling samples U(0,1) and plugs this value into the inverse CDF. Stratified sampling separates U(0,1) into equal bins and samples "Randomly" in each bin. Van der Corput sequences divides an interval into a a number of equal subintervals.

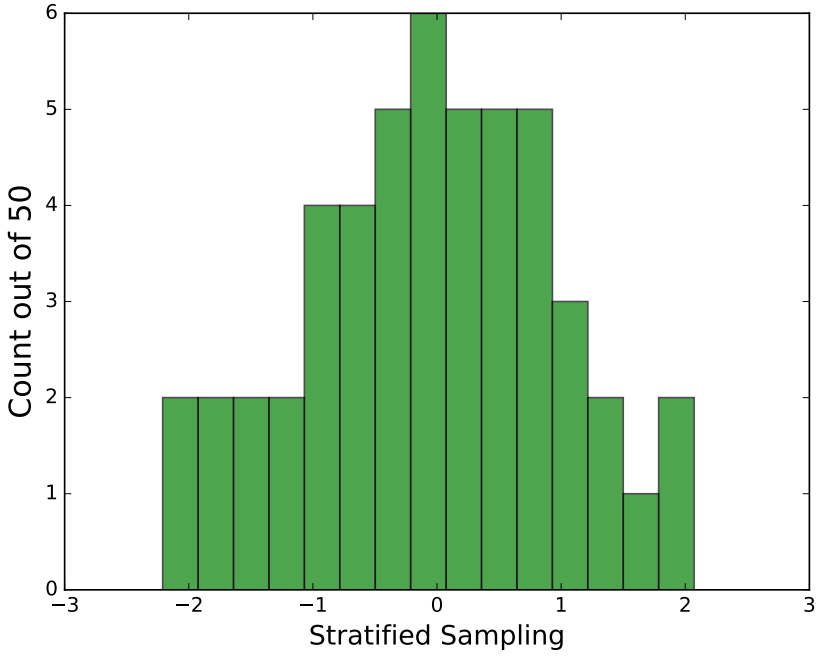
For example, the ordinary van der Corput sequence in base 3 is given by 1/3, 2/3, 1/9, 4/9, 7/9, 2/9, 5/9, 8/9, 1/27.

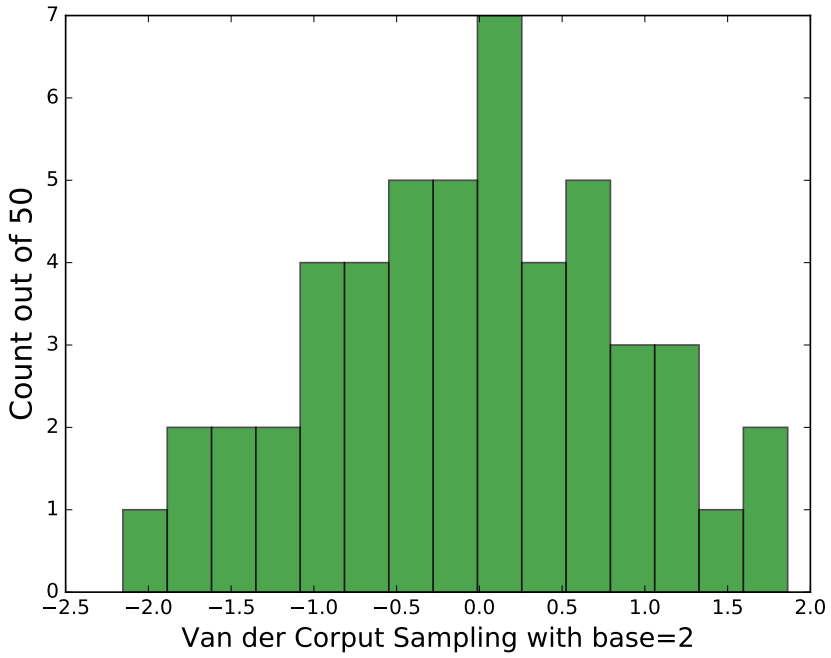
Listing 1: Script for Problem

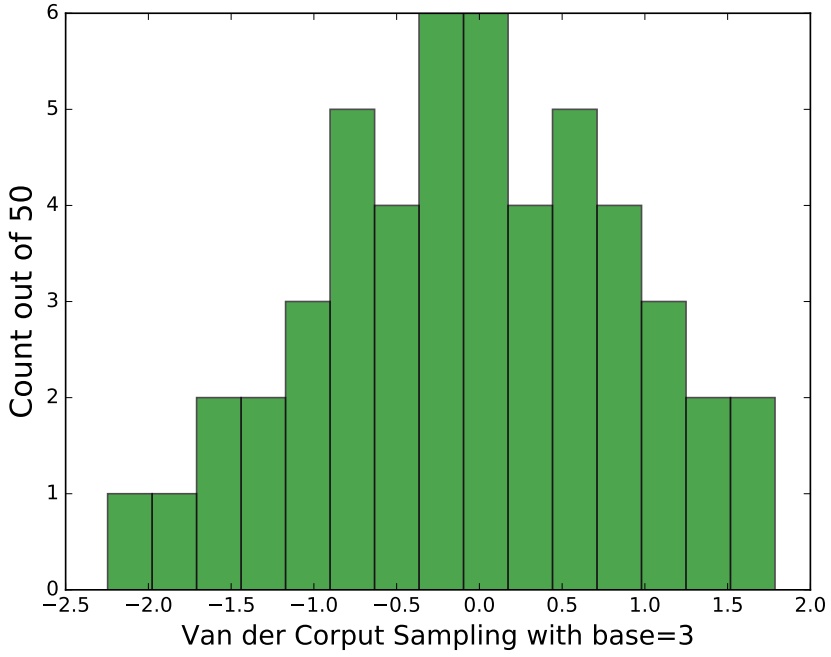
```
#!/usr/bin/env python3

#################################################################
#################### 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


Count out of 50
Van der Corput Sampling with base=3

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.

Problem 4 continued on next page...

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

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

Problem 4 continued on next page...

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 Hypercube and Halton

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

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

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

Problem 4 continued on next page...

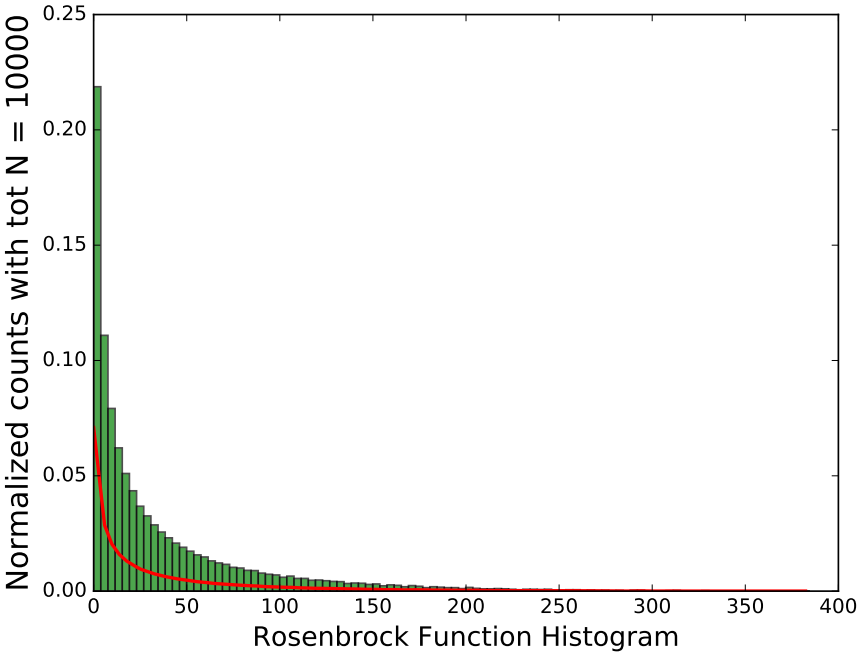
```
from random import shuffle

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

N=100  #Samples
Nbins=30 #Hist Plot
Nstrata=10
filename="HaltonStrat.pdf"

Xlabel="Halton Sampling"
#RandomNumbersX=fun.np.random.uniform(0,1,N)
#RandomNumbersY=fun.np.random.uniform(0,1,N)
#RandomNumbersX=fun.Rstrat(N,Nstrata)  #strat sampling
#RandomNumbersY=fun.Rstrat(N,Nstrata)
#l=lhsmdu.sample(2,N)  #Hyper cube sampling
#RandomNumbersX=l[0].A1
#RandomNumbersY=l[1].A1
RandomNumbersX=fun.Rvdc(N,2)     #Halton sequence
RandomNumbersY=fun.Rvdc(N,3)     #shuffled the list (tried notto)
#shuffle(RandomNumbersX)
#shuffle(RandomNumbersY)

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

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

f=fun.Rosen(X,Y)

#Plot the data for Rosenbrok, and plot fitted PDF
Xlabel="Rosenbrock Function Histogram"
(n,bins,ax,fig)=fun.HIST(Xlabel,f,Nbins,N)
(ax,fig)=fun.HISTDataToPDF(n,bins,ax,fig)
fun.plt.savefig(filename)

#Find the probability of f being less than 10
PGreater=sum(i<10 for i in f)/N

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


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

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

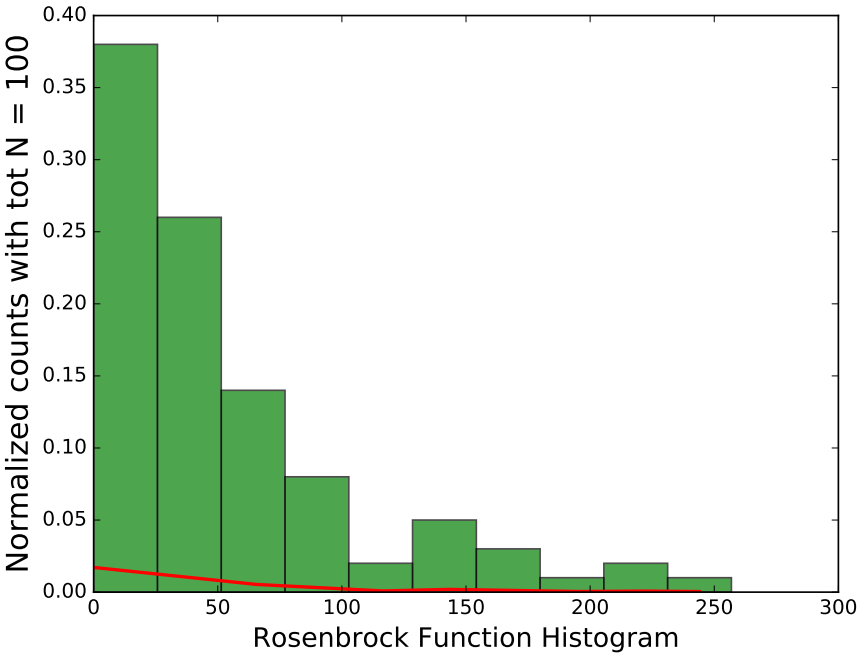
My PDF integrates to 1.01, which I'm okay with. The table below summarizes my results.

Table 2: Different Sampling Techniques

Method	p_{fail} 50 points	p_{fail} 100 points
Stratified	0.30	0.28
First Order	0.268	
Latin Hyper	0.36	0.26
Halton	0.24	0.25
Normal (10^5)	0.26017	

It should be noted that I shuffled the stratified samples, because otherwise there would be some correlation between the numbers. The first order listing in the table is not from the code, but the calculation in the first part of the problem above (did not use 50 points). I realized I could have used a more complicated first-order second moment reliability method, but this is A reliability method.




Normalized counts with tot N = 100
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
50
100
150
200
250
300
Rosenbrock Function Histogram

Problem 5

Consider the exponential integral function, $E_n(x)$,

$$E_n(x) = \int_1^\infty \frac{e^{-xt}}{t^n} dt$$

This function is involved in the solution to many pure-absorbing transport problems. Use this function to solve the transport problem,

$$\mu \frac{\delta \psi}{\delta x} + \sigma \psi = 0,$$

$$\psi(0, \mu > 0) = 1, \psi(10, \mu < 0) = 0,$$

for the scaler flux $\phi(x) = \int_{-1}^{1} \psi(x, \mu) d\mu$. Assume that $\sigma \sim \ GAM(10, 0.1)$. Use a PCE expansion to estimate the distribution, mean, and variance of $\phi(x)$ at $x = 1, 1.5, 3, 5$. Also, plot the mean value of ϕ as a function of x.

Using the integrating factor approach we proceed as follows:

$$\frac{\delta \psi}{\delta x} + \frac{\sigma \psi}{\mu} = 0$$
$$\frac{\delta}{\delta x}\left[e^{\frac{\sigma x}{\mu}} \psi\right] = 0$$

Integrating from 0 to x for $\mu > 0$, we obtain (yes I am copying some old notes)

$$\begin{aligned} \psi(x, \mu > 0) e^{\frac{\sigma x}{\mu > 0}} - \psi(0, \mu > 0) e^{\frac{\sigma \cdot 0}{\mu > 0}} &= 0 \\ \psi(x, \mu > 0) e^{\frac{\sigma x}{\mu > 0}} - 1 &= 0 \\ \psi(x, \mu > 0) e^{\frac{\sigma x}{\mu > 0}} &= 1 \\ \psi(x, \mu > 0) &= e^{\frac{-\sigma x}{\mu > 0}} \end{aligned}$$

Integrating from x to 10 for $\mu < 0$, we obtain

$$\begin{aligned} \psi(10, \mu < 0) e^{\frac{\sigma 10}{\mu < 0}} - \psi(x, \mu < 0) e^{\frac{\sigma \cdot x}{\mu < 0}} &= 0 \\ -\psi(x, \mu < 0) e^{\frac{\sigma \cdot x}{\mu < 0}} &= 0 \\ \psi(x, \mu < 0) &= 0 \end{aligned}$$

Here we can note that there is no flux traveling to the left, and will focus on the flux traveling to the right. Integrating over all $\mu > 0$, and using a substitution of $z = 1/\mu$ and $d\mu = -z^{-2}$:

Problem 5 continued on next page...

$$\begin{aligned}\phi^+(x) &= 2\pi \int_0^1 \psi(x, \mu > 0) d\mu \\ &= 2\pi \int_0^1 e^{-\frac{\sigma x}{\mu}} d\mu \\ &= 2\pi \int_\infty^1 -\frac{e^{-\sigma x z}}{z^2} dz \\ &= 2\pi \int_1^\infty \frac{e^{-\sigma x z}}{z^2} dz \\ &= 2\pi E_2(\sigma x)\end{aligned}$$

Now to use PCE expansion to estimate the distribution, mean, and variance of $\phi(x)$. First off, I want to say that I have no idea whats going on. Next, because we have a gamma distribution, I suppose we should use Laguerre Polynomials. Where:

$$\phi(\sigma x) = 2\pi \sum_{n=0}^{\infty} c_n L_n^{(\alpha)}(\beta x \sigma)$$

and

$$c_n = 2\pi \frac{n!}{\Gamma(n+\alpha+1)} \int_0^\infty E_2\left(\frac{x \cdot z}{\beta}\right) z^\alpha e^{-z} L_n^{(\alpha)}(z) dz$$

It should be noted that z is the standardized gamma distribution, with $z = \beta\sigma$, σ being our original gamma distribution.

In order to estimate the coefficients in the expansion we have to evaluate a wonderful integral. In order to estimate the integral, will use Gauss-Laguerre quadrature (like I have any idea what that is) to have as few evaluations of the integrand as possible.

The quadrature rule has the form

$$\int_0^\infty f(z) z^\alpha e^{-z} dz \approx \sum_{i=1}^{n} w_i f(z_i)$$

Where z_i are the n roots of $L_n^{(\alpha)}(z)$, and the weights are given by:

$$w_i = \frac{\Gamma(n+\alpha) z_i}{n!(n+\alpha)(L_{n-1}^\alpha(z_i))^2}$$

Looking at the quadrature rule, I think $f(z)$, in our instance would have to be

$$f(z) = E_2\left(\frac{x \cdot z}{\beta}\right) L_n^{(\alpha)}(z)$$

This is potentially confusing about the two n indice, so I'll put it all on one line, and hope its correct, if not then the only person I can blame is myself. The notes aren't very clear on what $f(z)$ is.

$$\begin{aligned}c_n &\approx \frac{n!}{\Gamma(n+\alpha+1)} \sum_{i=1}^{n'} w_i f(z_i) \\ c_n &\approx \frac{n!}{\Gamma(n+\alpha+1)} \sum_{i=1}^{n'} \frac{\Gamma(n'+\alpha) z_i}{n'!(n'+\alpha)(L_{n'-1}^\alpha(z_i))^2} E_2\left(\frac{x \cdot z_i}{\beta}\right) L_n^{(\alpha)}(z_i)\end{aligned}$$

Problem 5 continued on next page...

Where n' will increase until the summation is not changing, this is potentially confusing (for me). n' will start at 1. At which point the Laguerre polynomial $L_1^\alpha(x)$ will have a single root. The 'summation' will be a single term. Then n' will increase to 2, where there are two roots. The 'summation' will sum results from those two roots, **and not use the previous summation** (except to compare - not to add onto). The summation from $n' = 1$ and $n' = 2$ will be compared, if there is no difference (note there could be some zero terms) then the n' stops increasing, and the most recent summation is what we use moving forward.

Also n is the constant we are solving for. This way, the $L_n^\alpha(z_i)$ term on the right side will only be zero for when $n' = n$ (I hope this is correct).

According to the notes, the variance is:

$$\mathrm{Var}(G) = \sum_{n=1}^{\infty} \frac{\Gamma(n+\alpha+1)}{\Gamma(\alpha+1)n!} c_n^2$$

And c_0 is:

$$\begin{aligned} c_0 &= \int_0^\infty E_2\left(\frac{x\cdot z}{\beta}\right)\frac{z^\alpha e^{-z}}{\Gamma(\alpha+1)}dz = E[g(X)] \\ &\approx \sum_{i=1}^{n'} \frac{\Gamma(n'+\alpha)z_i}{n'!(n'+\alpha)(L_{n'-1}^\alpha(z_i))^2} E_2\left(\frac{x\cdot z_i}{\beta}\right) \end{aligned}$$

To check if this is correct, we can change the E_2 term for $cos(z/\beta)$ with Z $\sim$ G(1,2) and check to see if c_n converge to what Dr. McClarren has in his notes...which I'm hoping are correct.

Listing 3: Code for Calculation

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

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

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


x=5;alpha=1;beta=2;NumofC=10
alpha=10-1;beta=1/0.1
################################################################
################### Monte Calculations #########################
################################################################

N=100000  #Samples

RandomNumbers=fun.Rvdc(N,2)     #Halton sequence
Samples=fun.gammad.ppf(q=RandomNumbers,a=alpha+1,
                               scale=1/beta)
#SamplesSigma=fun.np.random.gamma(shape=alpha+1,scale=1/beta,size=N)
```

Problem 5 continued on next page...

```
MSolution=[] #Monte Solution
for i in range(0,len(Samples)):
    MSolution.append(fun.Fnear(Samples[i]*x))



################################################################
############### Deterministic Calculations ####################
################################################################

cn=[]
for n in range(0,NumofC):
    cn.append(fun.Determine_cn(n,alpha,beta,x))

Var=0
for n in range(1,NumofC):
    Coef=fun.gammaf(n+alpha+1)/(fun.gammaf(alpha+1)*fun.fact(n))
    Var=Var+Coef*(cn[n]**2)

DSolution=[] #Deterministic Solution with Monte Sampling
for i in range(0,len(Samples)):
    DSolution.append(fun.PolyChaos(cn,alpha,Samples[i]*beta))



################################################################
################## Printing and Plotting #######################
################################################################

fun.Print("Monte",MSolution)
fun.Print("Chaos",[cn[0],Var])
fun.Print("Monte+Chaos",DSolution)

#Plot the data, and plot fitted PDF
Nbins=100 #Hist Plot
filename="meanx_"+str(x)+".pdf"
Xlabel="Distribution at x = "+str(x)
(n,bins,ax,fig)=fun.HIST(Xlabel,DSolution,Nbins,N)
#(ax,fig)=fun.HISTDataToPDF(n,bins,ax,fig)
fun.plt.savefig(filename)

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

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

Problem 5 continued on next page...

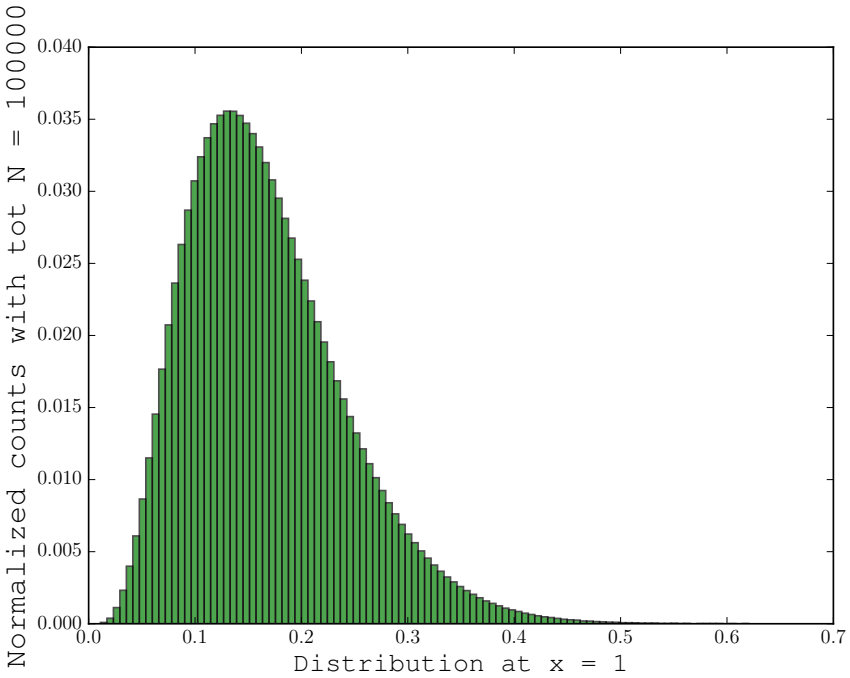
Table 3: Compare to the Dr. MC

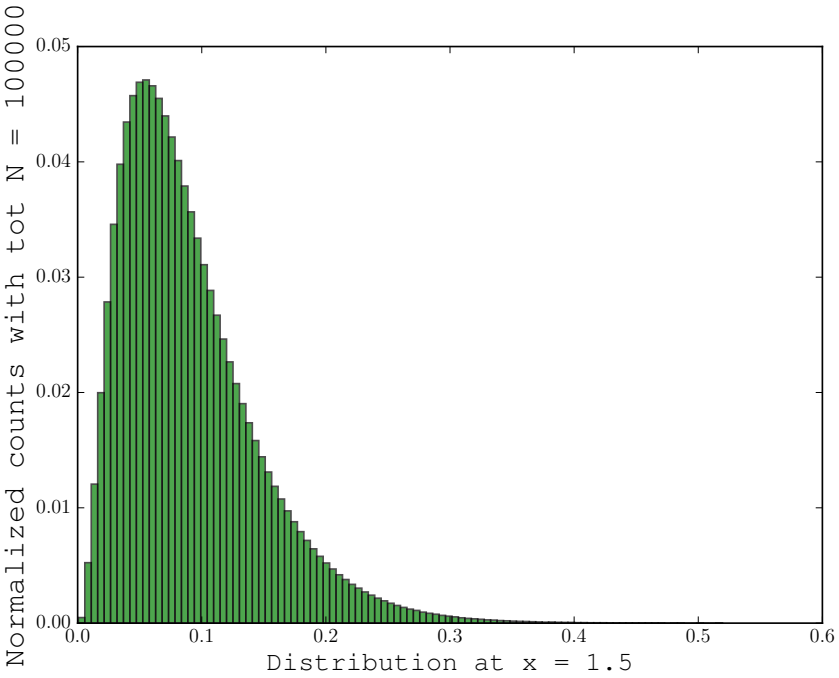
C_n	MC notes	Code
0	0.48	0.48
1	0.35	0.35
2	0.04	0.04
3	-0.05	-0.05
4	-0.03	-0.03
5	-0.00	-0.00

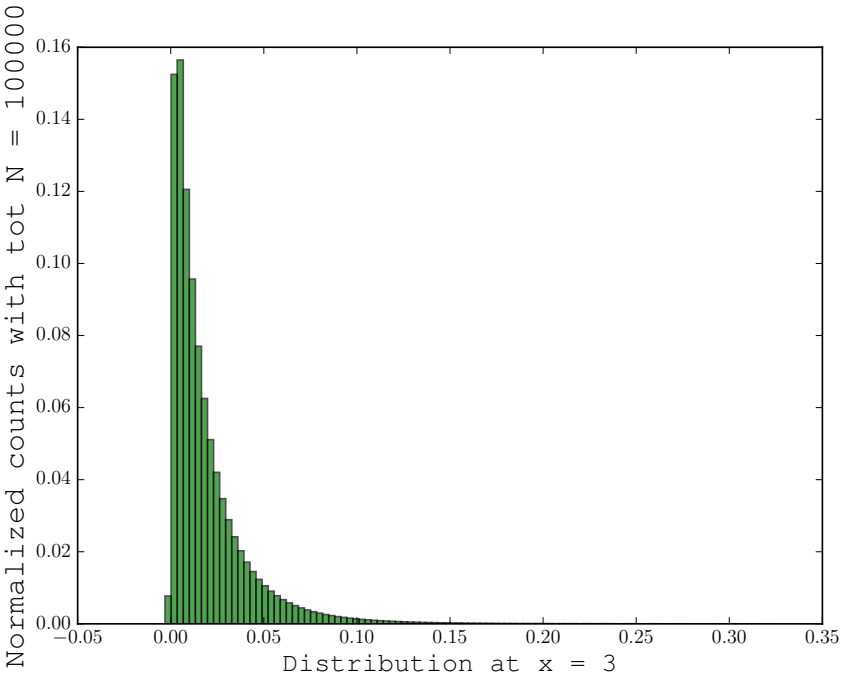
I am glad that works, as soon as I try to extend this to our problem it broke down. The reason is, the example given in lecture, with cos(x) needs a modification on the α and β terms so that my code gets the same answers ($\alpha + 1$ and $1/\beta$). When working with the homework problem, the terms don't need the modification, and if we do modify, we get REALLY bad answers..Also it should be noted that I don't include the 2π in the code, but because these answers match up close with the next problem, I leave the 2π out.

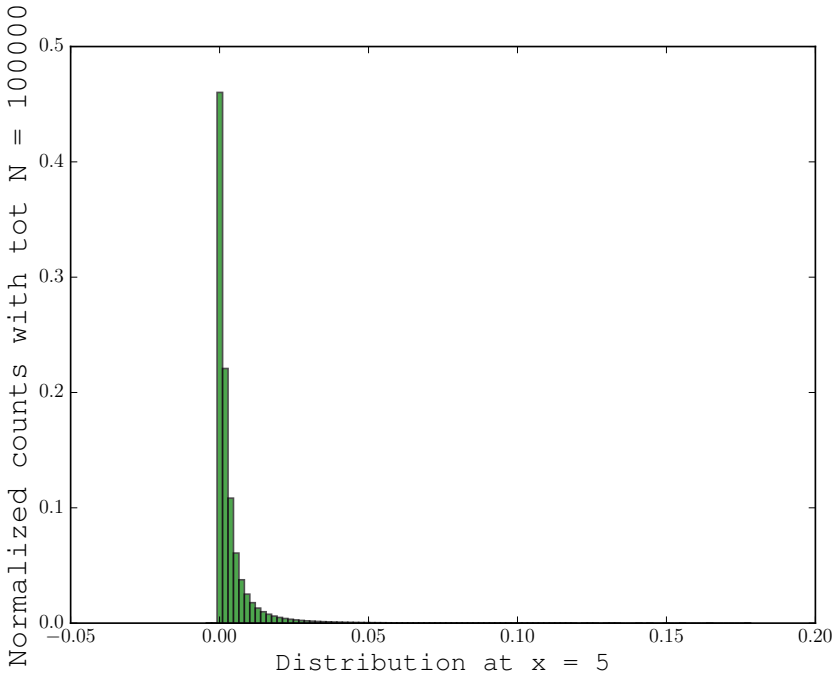
Table 4: Different Sampling Techniques

Location	Mean	Variance	Agree with MCNP
1	0.166714961646	0.00541820218281	Yes
1.5	0.090024688979	0.00303591212914	Yes
3	0.0192062954143	0.000475125069726	Yes
5	0.00362613412053	5.14761627202e-05	Yes




Normalized counts with tot N = 100000
0.05
0.04
0.03
0.02
0.01
0.00
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Distribution at x = 1.5


Normalized counts with tot N = 100000
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.05
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
Distribution at x = 3

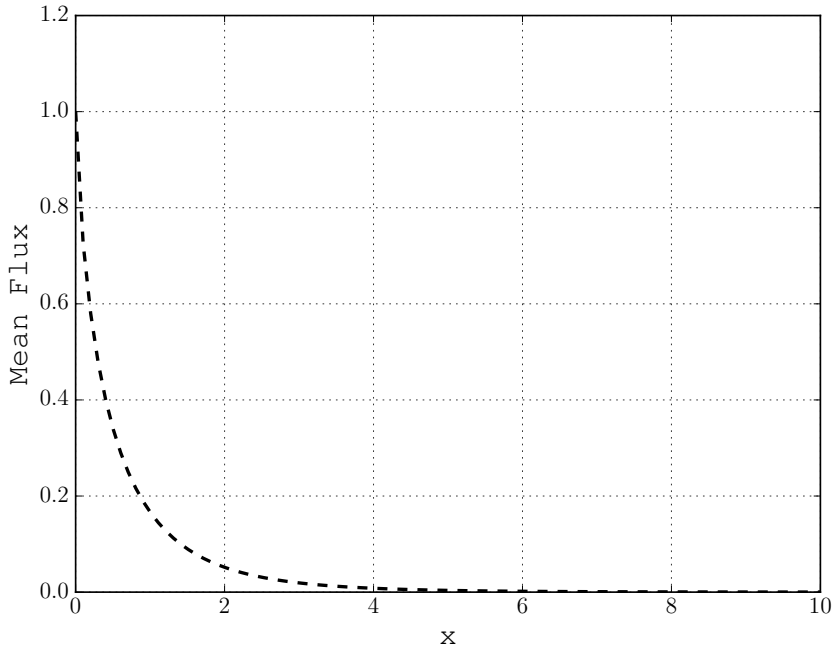


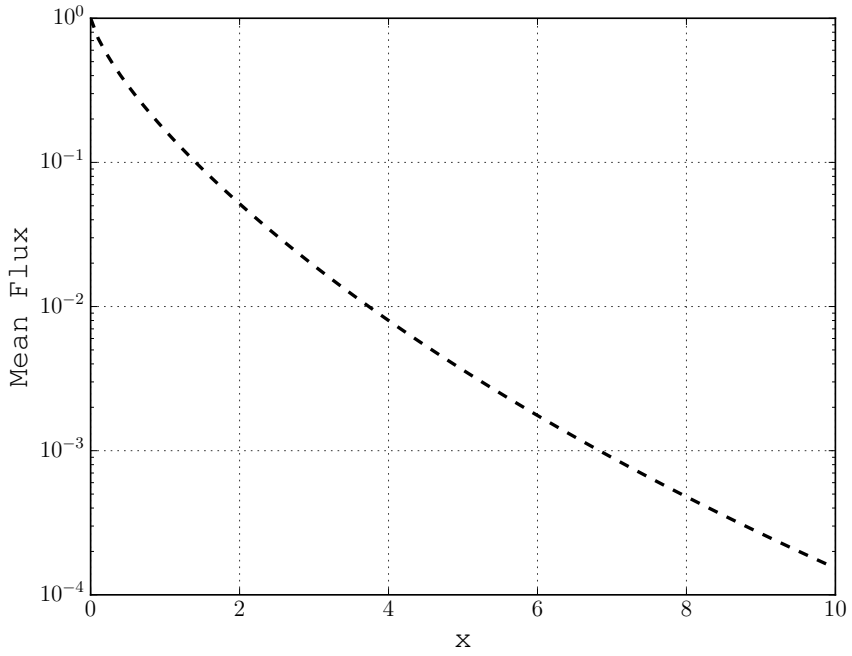
Listing 4: Code for plot of mean as a function of x

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

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

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


x=5;alpha=1;beta=2;NumofC=10
alpha=10-1;beta=1/0.1
################################################################
#################### Monte Calculations ########################
################################################################

N=100000  #Samples

RandomNumbers=fun.Rvdc(N,2)     #Halton sequence
Samples=fun.gammad.ppf(q=RandomNumbers,a=alpha+1,
                           scale=1/beta)
#SamplesSigma=fun.np.random.gamma(shape=alpha+1,scale=1/beta,size=N)

MSolution=[] #Monte Solution
for i in range(0,len(Samples)):
    MSolution.append(fun.Fnear(Samples[i]*x))
```

```
################################################################
############### Deterministic Calculations ####################
################################################################

cn=[]
for n in range(0,NumofC):
    cn.append(fun.Determine_cn(n,alpha,beta,x))

Var=0
for n in range(1,NumofC):
    Coef=fun.gammaf(n+alpha+1)/(fun.gammaf(alpha+1)*fun.fact(n))
    Var=Var+Coef*(cn[n]**2)

DSolution=[] #Deterministic Solution with Monte Sampling
for i in range(0,len(Samples)):
    DSolution.append(fun.PolyChaos(cn,alpha,Samples[i]*beta))


################################################################
################## Printing and Plotting #######################
################################################################

fun.Print("Monte",MSolution)
fun.Print("Chaos",[cn[0],Var])
fun.Print("Monte+Chaos",DSolution)

#Plot the data, and plot fitted PDF
Nbins=100 #Hist Plot
filename="meanx_"+str(x)+".pdf"
Xlabel="Distribution at x = "+str(x)
(n,bins,ax,fig)=fun.HIST(Xlabel,DSolution,Nbins,N)
#(ax,fig)=fun.HISTDataToPDF(n,bins,ax,fig)
fun.plt.savefig(filename)

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

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

Plot for the mean log scale and normal.


Mean Flux
x
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0
2
4
6
8
10


Mean Flux
x
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
2
4
6
8
10

Problem 6

You perform a measurement of a beam of radiation satisfying the boundary condition in problem 5 hitting a slab, and somehow are able to measure the scalar flux at $x = 1, 1.5, 3, 5$:

Table 5: Measured and calculated flux

Location	Measured	Calculated	Variance
1	0.201131	0.166714961646	0.00541820218281
1.5	0.110135	0.090024688979	0.00303591212914
3	0.0228748	0.0192062954143	0.000475125069726
5	0.0032849	0.00362613412053	5.14761627202e-05

Using the prior distribution for σ from problem 5, and the experimental data just given, derive a posterior distribution for σ (i.e calibrate σ). You may assume that the measurement has an error distributed by N(0,σ=0.001).

There has to be a better way to do this, but my brain is too fried to think about it. I am going to run the code from problem 5, vary the alpha and beta parameters, and calculate the error adjusted error,

$$E = \sum_{j=1}^{2} \sum_{i=1}^{4} \frac{\text{New Calculation}_i - \text{Original or Measured}_{i,j}}{\text{Original or Measured}_{i,j}}$$

and see which gives me the smallest value for E. The above equation was summed over the four measurement points. Original or Measured refers to either the original calculation from problem 5, for the measured data (why the first summation has an upper limit of 2). The error was not incluced because some of the calculated values variances are really small.

After looking for a parameter that would minimize the error between measured values and calculated values, I got $\alpha =$ and $\beta =$.

Table 6: Measured and calculated flux

Location	Measured	Calculated	Variance
1	0.201131		
1.5	0.110135		
3	0.0228748		
5	0.0032849		

Scripts and plots for choosing the parameter are below.

Listing 5: Code for Calculation

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

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

```

```
import time
start_time = time.time()
import FUN as fun
from uncertainties import ufloat
from uncertainties.umath import *
from uncertainties import unumpy as unp

################################################################
######################### Set up ###############################
################################################################
NumofC=6

output=open("All_Calcs.csv","w")
print("alpha,beta,SumErr",file=output)

#Loop through all possibilities
alphaL=fun.np.linspace(1,20,3);
betaL=fun.np.linspace(1,20,3);
xL=[1,1.5,3,5];
Measured=[0.201131,0.110135,0.0228748,0.0032849]
MeasuredVar=[0.001,0.001,0.001,0.001]
Prior=[0.166714961646,0.090024688979,0.0192062954143,0.00362613412053]
PriorVar=[0.00541820218281,0.00303591212914,0.000475125069726,5.14761627202e-05]

################################################################
#################### Brute Force ###############################
################################################################

SumErrs=[]
for alpha in alphaL:
    for beta in betaL:
        ErrorS=0
        for i in range(0,len(xL)):
            x=xL[i]

            #cn=[]
            #for n in range(0,NumofC):
            #    cn.append(fun.Determine_cn(n,alpha,beta,x))
            c0=fun.Determine_cn(0,alpha,beta,x)
            #Var=0
            #for n in range(1,NumofC):
            #    Coef=fun.gammaf(n+alpha+1)/
            #(fun.gammaf(alpha+1)*fun.fact(n))
            #    Var=Var+Coef*(cn[n]**2)

            #Calculate err
            #M=ufloat(Measured[i],MeasuredVar[i])
            #C=ufloat(Prior[i],PriorVar[i])
            #C2=ufloat(cn[0],Var)
            M=Measured[i]
            C=Prior[i]
            C2=c0
            #if C.std_dev<1e-6:
```

Problem 6 continued on next page...

```
            #    C.std_dev=1e-3
            #if C2.std_dev<1e-6:
            #    C2.std_dev=1e-3
            Error=abs(C2-M)/M+abs(C2-C)/C
            #Error=Error.nominal_value/Error.std_dev
            ErrorS=ErrorS+Error

        SumErrs.append(ErrorS)
        print(str(alpha)+","+str(beta)+','+str(Error),file=output)


############################################################
################### Find Minimum ###########################
############################################################
print("")

#Find minimium Summed Error
Min=min(SumErrs)
Count=0
for alpha in alphaL:
    for beta in betaL:
        if SumErrs[Count]==Min:
            print("Min of Summed Errors: %.2f, " % Min +
                  "alpha: %.3f, " % alpha +
                  'beta: %.3f, ' % beta)
            alpharun=alpha
            betarun=beta

        Count=Count+1

############################################################
################## Print Minimum ###########################
############################################################

print("")

for i in range(0,len(xL)):
    x=xL[i]
    alpha=alpharun;beta=betarun

    cn=[]
    for n in range(0,NumofC):
        cn.append(fun.Determine_cn(n,alpha,beta,x))

    Var=0
    for n in range(1,NumofC):
        Coef=fun.gammaf(n+alpha+1)/(fun.gammaf(alpha+1)*fun.fact(n))
        Var=Var+Coef*(cn[n]**2)

    fun.Print(x,"Chaos",[cn[0],Var])

print("")
###################### Time To execute ################
```

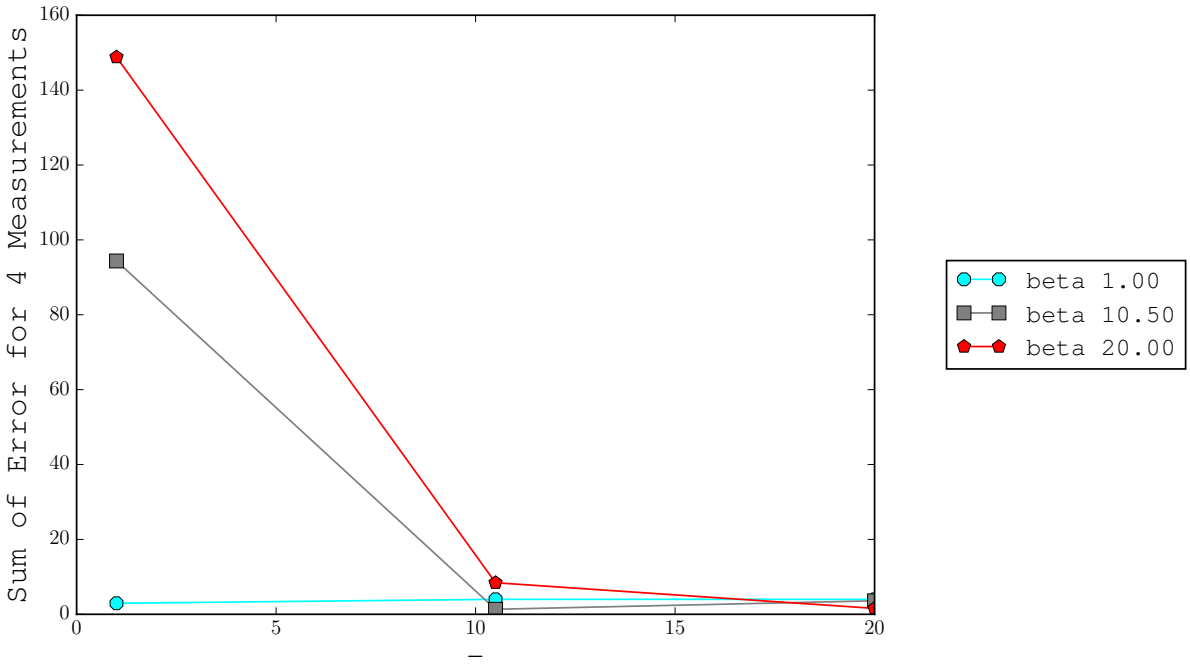
Problem 6 continued on next page...

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

Listing 6: Code for plotting

```
#!/usr/bin/env python3
"""
This will plot how the error behaves as a function of flux
parameters. The error is the difference between the calculated
1-group cross section and the x-section ORIGEN reports
"""
#########################################################################
######################### Import Packages ###############################
#########################################################################

import time
start_time = time.time()
import Functions_Plot as f
import pandas as pd
import copy

#########################################################################
######################### Data Frame Work ###############################
#########################################################################

#Load up dataframe
df=pd.read_csv('All_Calcs.csv',sep=',',index_col=False)
ColumnNames=df.columns.values #Probably wont use

#Set X and Y values for plotting ['alpha','beta']
X='alpha'    #Should be one of the variables
#ALSO IF YOU CHANGE CHANGE Xlabel!!!!!!!!!!!!!!!!!!
Y='SumErr'  #Should be an error term

#Three variables that I am...well varying
Variables=["alpha","beta"]
Printing=["%.2f","%.2f"]

XScale="linear"     # 'linear' or 'log'
#'$\alpha$ Parameter'   For alpha
#'$\beta$ Parameter'   For beta
Xlabel='-'  # X label
YScale="linear"  # 'linear' or 'log'
Ylabel='Sum of Error for 4 Measurements'

#Remove the one variable I am looking at, group by others
del Printing[Variables.index(X)]
Variables.remove(X)

UniqueOther1=pd.unique(df[Variables[0]])

#Set up plot
fig=f.plt.figure(figsize=f.FigureSize)
ax=fig.add_subplot(111)
```

```
50
Check=0
for U1 in UniqueOther1:
    Label=Variables[0]+" "+Printing[0] % U1
    #Take all rows with U1 (filter)
55  dfHold=df[df[Variables[0]] == U1]
    #Take all rows with U2 (filter again)
    Error=dfHold[Y].values
    Xplot=dfHold[X].values
    (fit,ax)=f.plot(Xplot,Error,ax,Check,Label,
60                  fig,Ylabel,Xlabel,XScale,YScale)
    Check=Check+1

ax=f.Legend(ax)
f.plt.savefig(X+"_vs_"+Y+'.pdf')
65

#############################################################################
######################### Time to Execute ###################################
#############################################################################

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



Long Problem 1

Using a discretization of your choice, solve the equation

$$\frac{\delta u}{\delta t} + v\frac{\delta u}{\delta x} = D\frac{\delta^2 u}{\delta x^2} - \omega u$$

for $u(x,t)$ on the spatial domain $x \in [0,10]$ with periodic boundary conditions $u(0^-) = u(10^+)$, and initial conditions

$$u(x,0) = \begin{cases} 1, & x \in [0,2.5] \\ 0, & \text{otherwise} \end{cases}$$

Use the solution to compute the total reactions.

$$\int_5^6 dx \int_0^5 dt\ \omega u(x,t).$$

Compute scaled sensitivity coefficients and sensitivity indices for normal random variables:

(a) $\mu_v = 0.5,\ \sigma_c = 0.1$

(b) $\mu_D = 0.125,\ \sigma_D = 0.03$

(c) $\mu_\omega = 0.1,\ \sigma_\omega = 0.05$

How do these results change with changes in Δx and Δt?

n for time, i for x

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} + v\frac{u_i^{n+1} - u_{i-1}^{n+1}}{\Delta x} = D\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} - wu_i^{n+1}$$

Grouping i on LHS:

$$U_i^{n+1}\left[\frac{1}{\Delta t} + \frac{v}{\Delta x} + \frac{2D}{\Delta x^2} + w\right]$$

Grouping $i+1$ on LHS:

$$U_{i+1}^{n+1}\left[\frac{-D}{\Delta x^2}\right]$$

Grouping $i-1$ on LHS:

$$U_{i-1}^{n+1}\left[-\frac{v}{\Delta x} - \frac{D}{\Delta x^2}\right]$$

RHS:

$$\frac{u_i^n}{\Delta t}$$

Boundary conditions set A(1,I)=A(1,1) and A(I,1)=A(I,I).

The code for calculating the reaction rate is shown below and is used for problems 3, and 4, but is only shown here. Specific code for other calculations will be shown.

Listing 7: Code for RXNs

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

############################################################
#################### Import packages #######################
############################################################
```

Long Problem 1 continued on next page...

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

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

#Samples, X-1 bins, t-1 time steps
N=1;NX=101;Nt=11

#Problem boundaries
x=fun.np.linspace(0,10,NX)
dx=x[1]-x[0]
t=fun.np.linspace(0,5,Nt)
dt=t[1]-t[0]

#Generate Samples;         loc = mean; scale=STD
v=fun.np.random.normal(loc=0.5,scale=0.1,size=N)
D=fun.np.random.normal(loc=0.125,scale=0.03,size=N)
w=fun.np.random.normal(loc=0.1,scale=0.05,size=N)

#Build A
A=fun.BuildA(v,D,w,dt,dx,NX)

#Build Matrix with solutions
u=fun.np.zeros([NX,Nt])
for i in range(0,NX):
    if x[i]<=2.5:
        u[i,0]=1

#Solve the system
for i in range(1,len(t)):
    u[:,i] = fun.np.linalg.solve(A,u[:,i-1]*(1/dt))


#Integrate the system
Integral=fun.Integrate(u,5,6,x,t)
RXNRate=Integral*w

print(RXNRate)
#################### Time To execute ################

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

Long Problem 1 continued on next page...

The scaled sensitivity coefficient, and sensitivity index are defined as

$$\text{Scaled Sensitivity Coeff}|_i = \mu_i \frac{\delta \text{QoI}}{\delta \theta_i}\bigg|_{\bar{\theta}}$$

$$\text{Sensitivity Index}|_i = \sigma_i \frac{\delta \text{QoI}}{\delta \theta_i}\bigg|_{\bar{\theta}}$$

Where the QoI is evaluated at mean values for parameters and a single parameter is varied off the mean. The code for calculating these values is shown below

Listing 8: Code for Sensitivities

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

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

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

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

#Generate Samples;         loc = mean; scale=STD
#v=fun.np.random.normal(loc=0.5,scale=0.1,size=N)
v=[0.5,0.6];dp=v[1]-v[0];mu=0.5;sigma=0.1;Param='v'
#D=fun.np.random.normal(loc=0.125,scale=0.03,size=N)
D=[0.125,0.125];#dp=D[1]-D[0];mu=0.125;sigma=0.03;Param='D'
#w=fun.np.random.normal(loc=0.1,scale=0.05,size=N)
w=[0.1,0.1];#dp=w[1]-w[0];mu=0.1;sigma=0.05;Param='w'



#Samples, X-1 bins, t-1 time steps
N=1;
NX=[51,101,201,251,301,351,401,451];Vary='x';Nt=[11];l=0
#Nt=[11,21,31,41,51,61,71,81,91,101];Vary='t';NX=[401];j=0


Scales=[];Indice=[];dxs=[]  #The dxs could hold dts

for j in range(0,len(NX)):   #Loop over space
#for l in range(0,len(Nt)):    #Loop over time

    #Problem boundaries
    x=fun.np.linspace(0,10,NX[j])
    dx=x[1]-x[0]
    t=fun.np.linspace(0,5,Nt[l])
    dt=t[1]-t[0]
```

Long Problem 1 continued on next page...

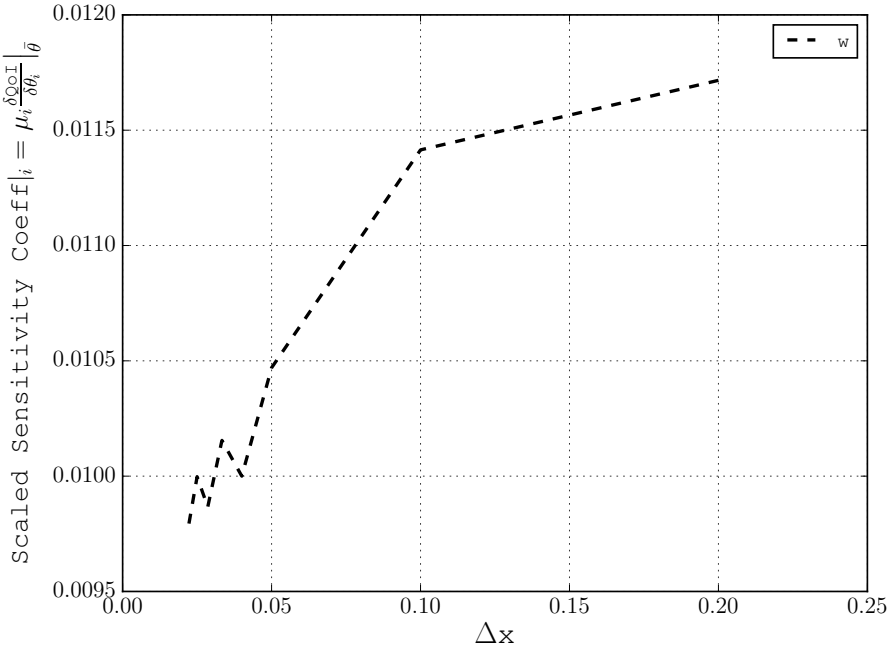
```
    RXNs=[];

    for i in range(0,2):
        #Build A
        A=fun.BuildA(v[i],D[i],w[i],dt,dx,NX[j])
        #Calculate QoI
        RXNRate=fun.SolveQoI(NX[j],Nt[l],x,t,A,dt,w[i])
        RXNs.append(RXNRate)

    Scaled=((RXNs[1]-RXNs[0])/(dp))*mu
    Index=((RXNs[1]-RXNs[0])/(dp))*sigma

    Scales.append(Scaled)
    Indice.append(Index)
    if Vary=='x':
        dxs.append(dx)
    elif Vary=='t':
        dxs.append(dt)

#Plot the data, and plot fitted PDF
for i in range(0,2):
    fig = fun.plt.figure()
    ax = fig.add_subplot(111)
    if i==0:
        (fig,ax)=fun.Plot(dxs,Scales,Param,fig,ax,'k--',Vary,Scaled=True)
        filename='scaled_'+Param+Vary+'.pdf'
    else:
        (fig,ax)=fun.Plot(dxs,Indice,Param,fig,ax,'k--',Vary,Scaled=False)
        filename='index_'+Param+Vary+'.pdf'

    handles,labels=ax.get_legend_handles_labels()
    Lfont={'family':'monospace',
           'size':12}
    ax.legend(handles,labels,loc='best',fontsize=12)
    fun.plt.savefig(filename)


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

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

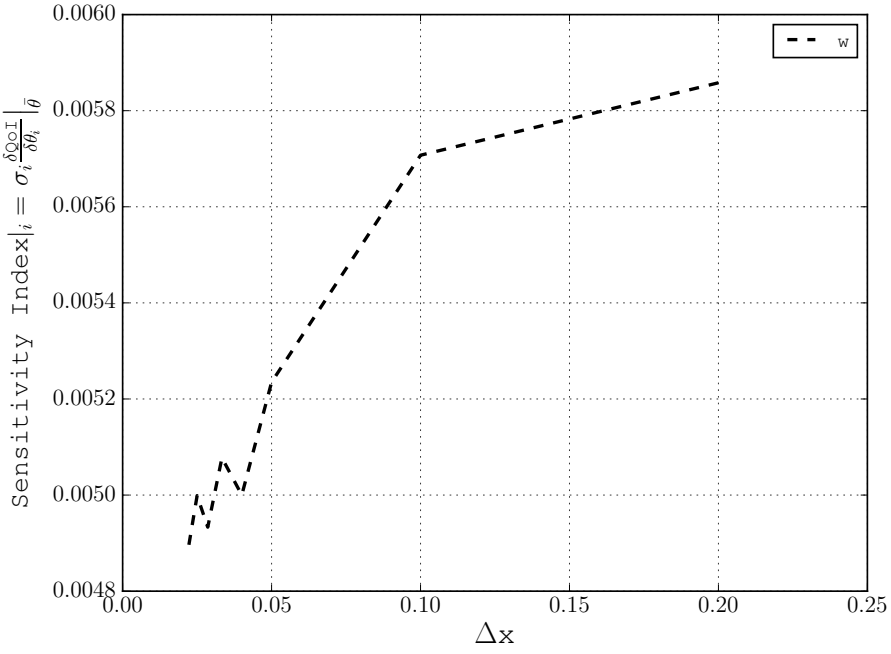
Table 7: Sensitivity Parameters dx = 0.1 dt = 0.5

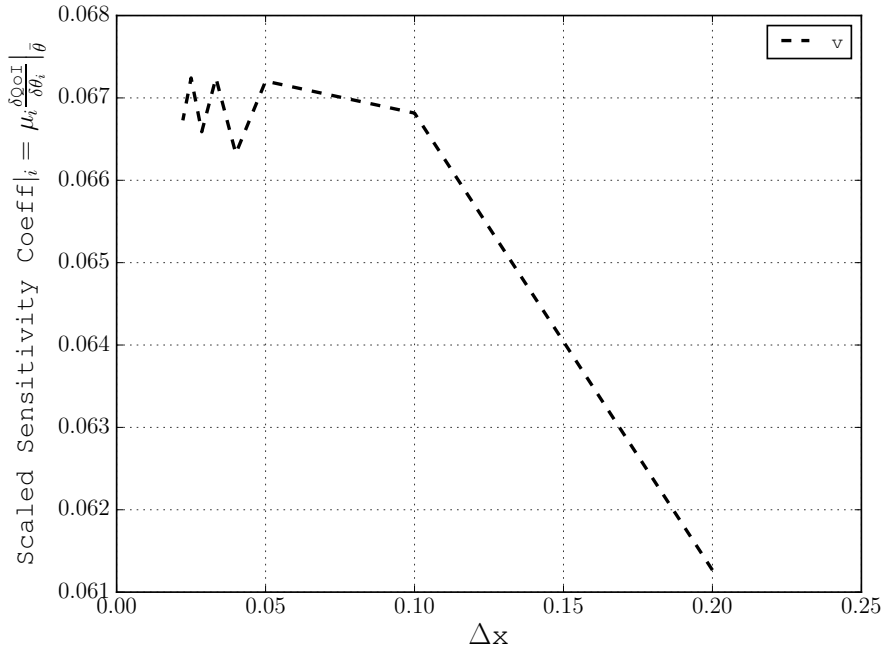
Parameter	Scaled Sensitivity Coeff	Sensitivity Index
v	0.0668156428348	0.013363128567
D	0.00872948872873	0.0020950772949
ω	0.0114146347046	0.00570731735231

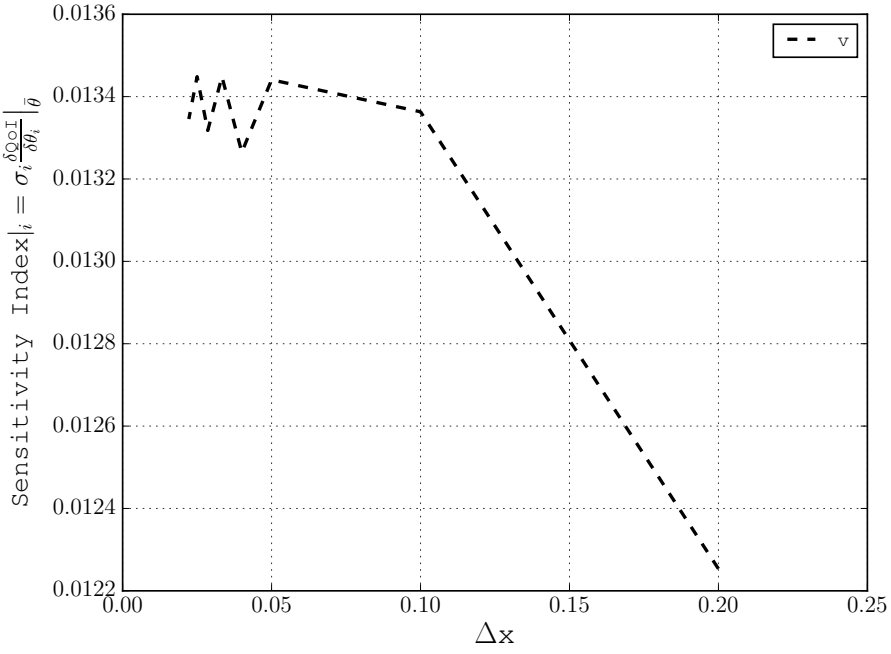
All these plots were produced by increasing the value of a single parameter by one standard deviation. Other variables were held at their mean. When Δx was being varied, the time divisions were held at 11, and when

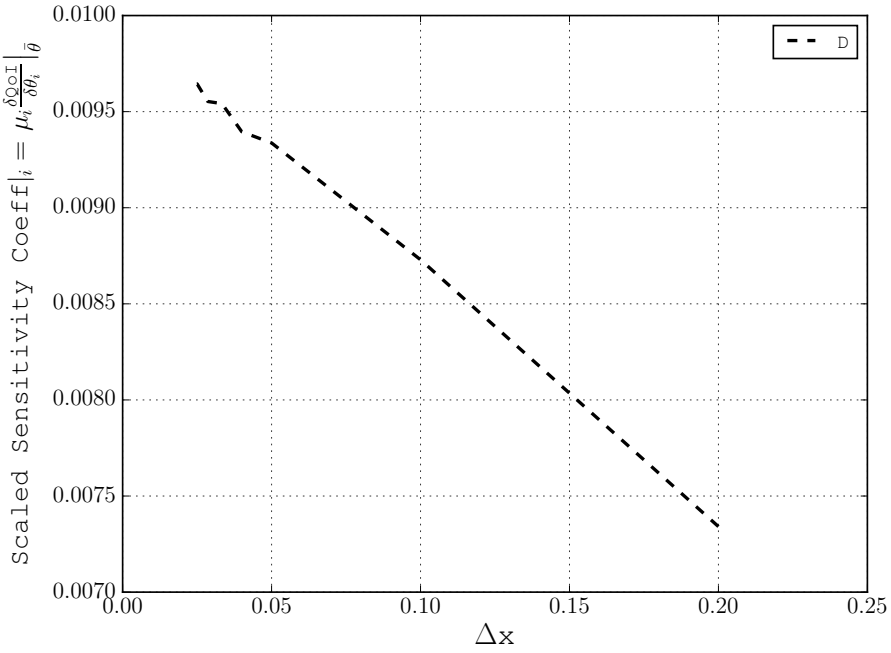
Δt was being varied, the space divisions were held at 401. Also I apologize for all the plots, there is definately a smarter way to do this.

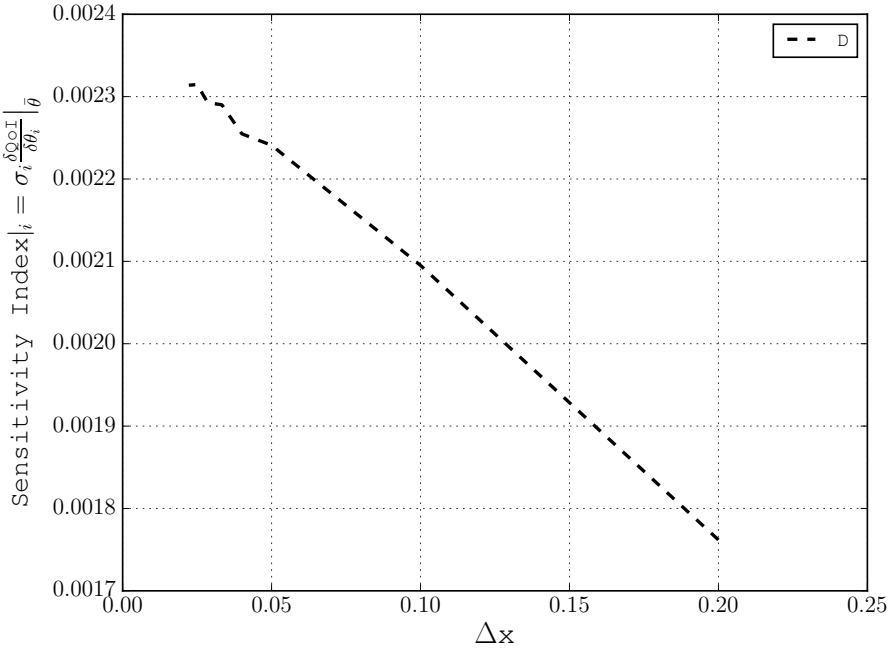


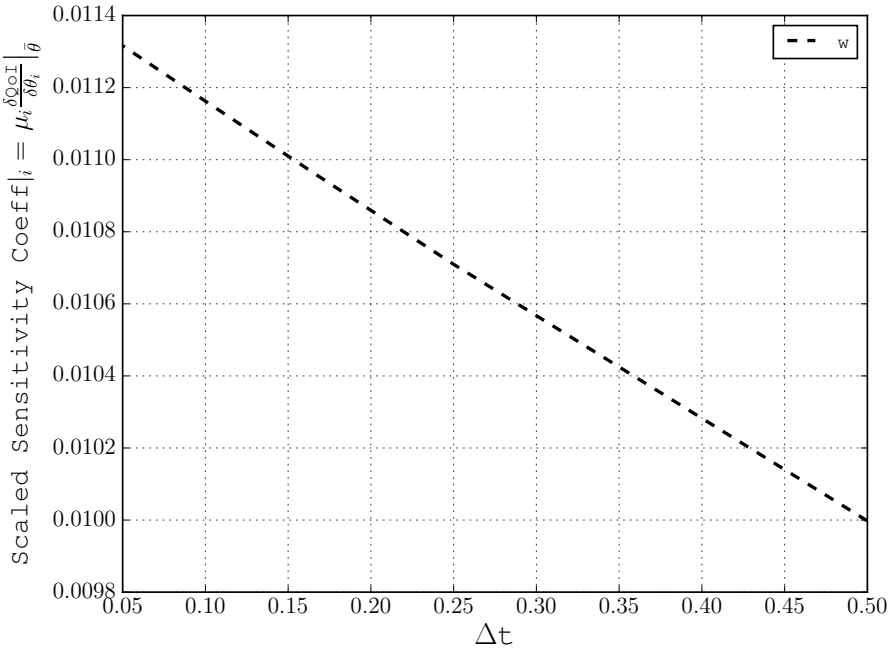


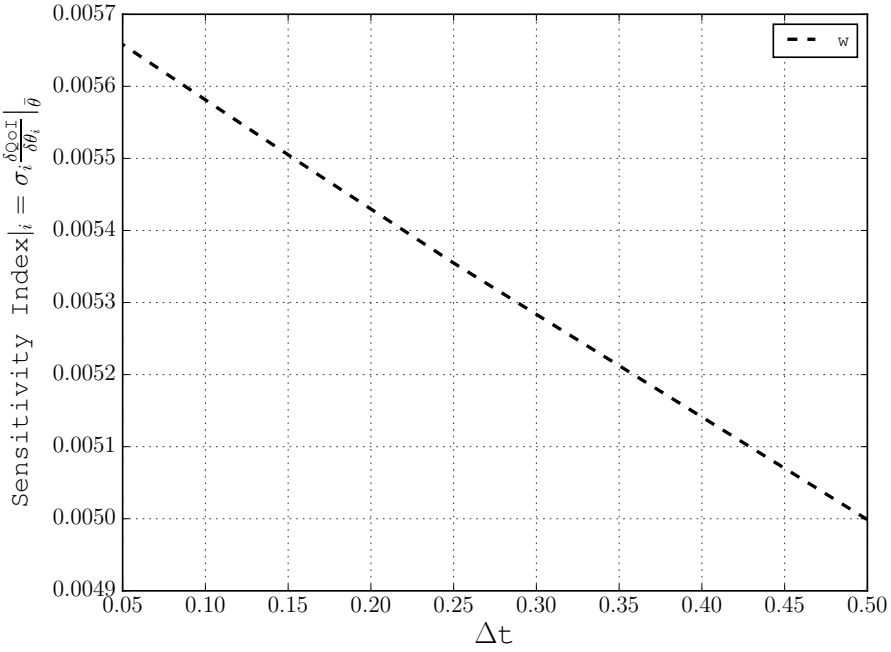

Scaled Sensitivity Coeff|$_i = \mu_i \frac{\delta Q_{OI}}{\delta \theta_i}\big|_{\bar{\theta}}$
0.068
0.067
0.066
0.065
0.064
0.063
0.062
0.061
0.00
0.05
0.10
0.15
0.20
0.25
Δx
v

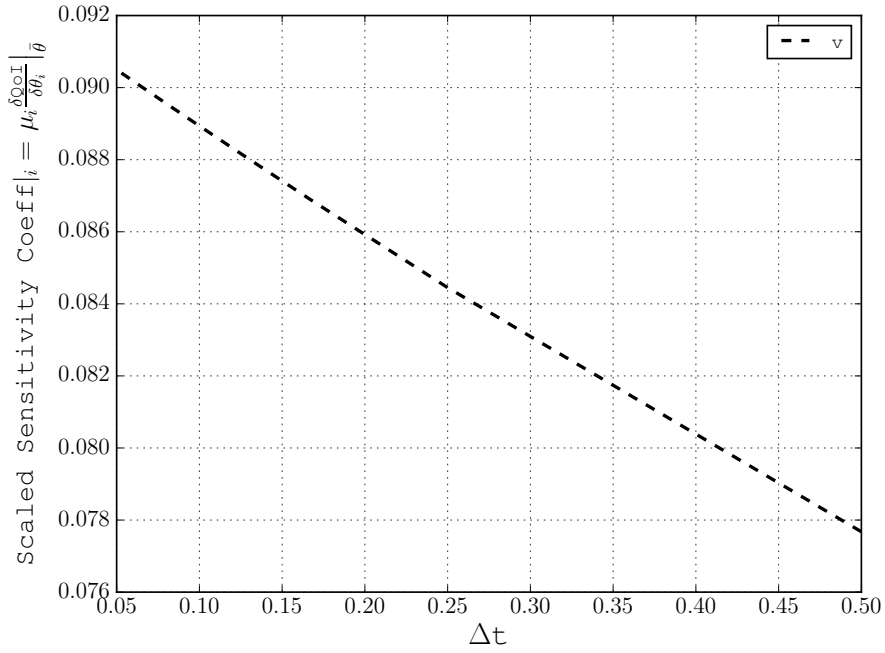

Sensitivity Index|$_i = \sigma_i \frac{\delta Q_{OI}}{\delta \theta_i}\big|_{\bar{\theta}}$
0.0136
0.0134
0.0132
0.0130
0.0128
0.0126
0.0124
0.0122
0.00
0.05
0.10
0.15
0.20
0.25
Δx
v

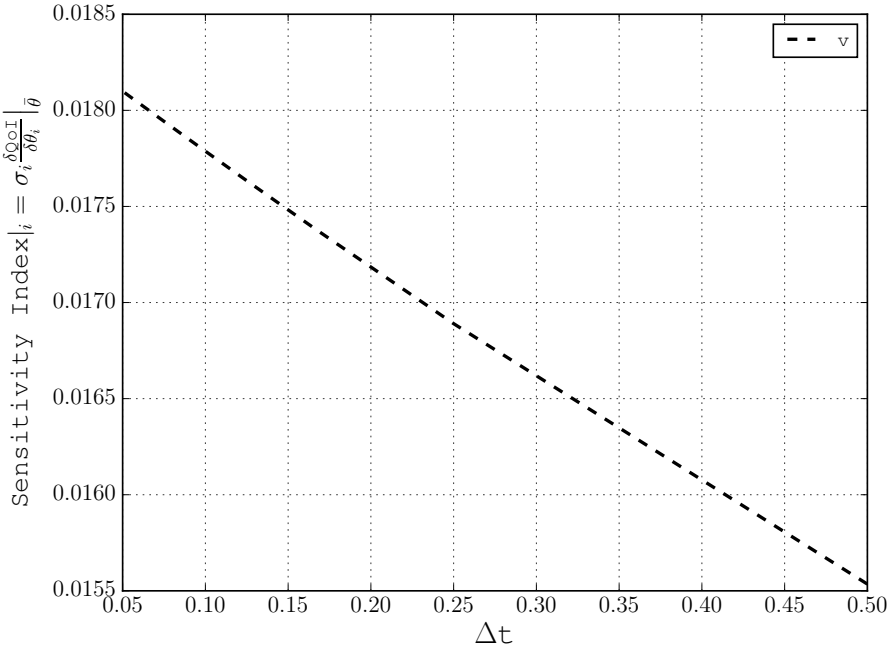

0.0100
0.0095
0.0090
0.0085
0.0080
0.0075
0.0070
0.00
0.05
0.10
0.15
0.20
0.25
Δx
D
Scaled Sensitivity Coeff|$_i = \mu_i \frac{\delta QoI}{\delta \theta_i}\big|_{\bar{\theta}}$

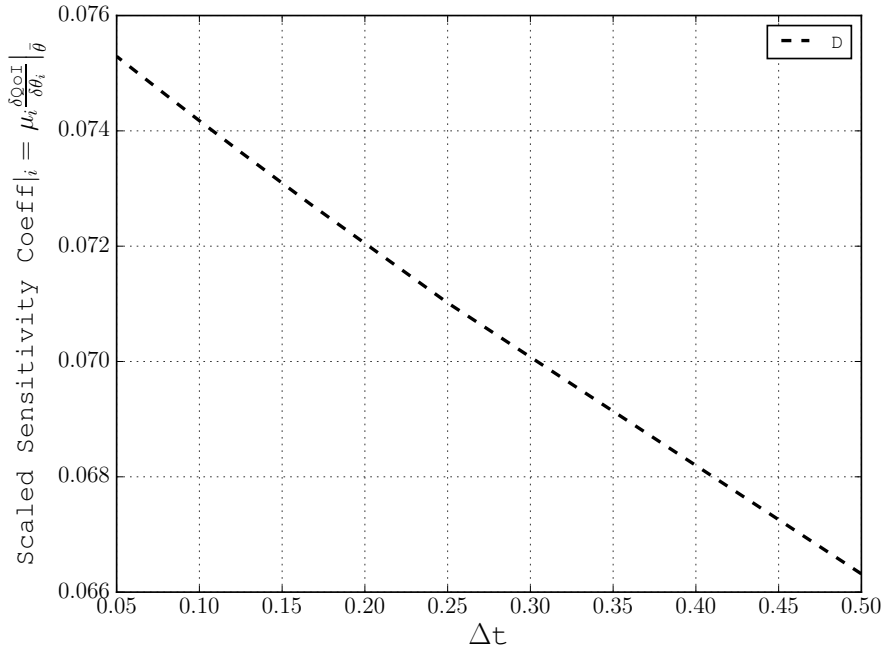

0.0024
0.0023
0.0022
0.0021
0.0020
0.0019
0.0018
0.0017
0.00
0.05
0.10
0.15
0.20
0.25
Δx
D
Sensitivity Index|$_i = \sigma_i \frac{\delta QoI}{\delta \theta_i}\big|_{\bar{\theta}}$

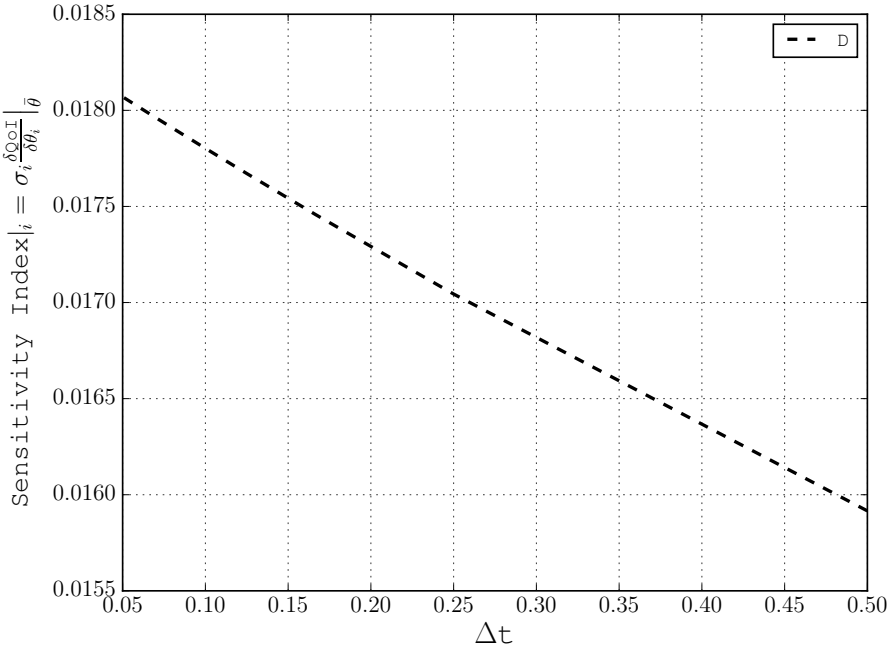

Scaled Sensitivity Coeff$|_i = \mu_i \frac{\delta Q o I}{\delta \theta_i}\big|_{\bar{\theta}}$
0.0114
0.0112
0.0110
0.0108
0.0106
0.0104
0.0102
0.0100
0.0098
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt
w


Sensitivity Index$|_i = \sigma_i \frac{\delta Q o I}{\delta \theta_i}\big|_{\bar{\theta}}$
0.0057
0.0056
0.0055
0.0054
0.0053
0.0052
0.0051
0.0050
0.0049
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt
w


v
0.092
0.090
0.088
0.086
0.084
0.082
0.080
0.078
0.076
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt
Scaled Sensitivity Coeff|$_i = \mu_i \frac{\delta Q_{OI}}{\delta \theta_i}\big|_{\bar{\theta}}$


v
0.0185
0.0180
0.0175
0.0170
0.0165
0.0160
0.0155
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt
Sensitivity Index|$_i = \sigma_i \frac{\delta Q_{OI}}{\delta \theta_i}\big|_{\bar{\theta}}$


0.076
0.074
0.072
0.070
0.068
0.066
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt
D
Scaled Sensitivity Coeff|$_i = \mu_i \frac{\delta Q_{OI}}{\delta \theta_i}\big|_{\bar{\theta}}$


0.0185
0.0180
0.0175
0.0170
0.0165
0.0160
0.0155
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt
D
Sensitivity Index|$_i = \sigma_i \frac{\delta Q_{OI}}{\delta \theta_i}\big|_{\bar{\theta}}$

Long Problem 3

Using a discretization of your choice, solve the equation

$$\frac{\delta u}{\delta t} + v\frac{\delta u}{\delta x} = D\frac{\delta^2 u}{\delta x^2} - \omega u$$

for $u(x,t)$ on the spatial domain $x \in [0,10]$ with periodic boundary conditions $u(0^-) = u(10^+)$, and initial conditions

$$u(x,0) = \begin{cases} 1, & x \in [0,2.5] \\ 0, & \text{otherwise} \end{cases}$$

Use the solution to compute the total reactions

$$\int_5^6 dx \int_0^5 dt\ \omega u(x,t).$$

Sample values of parameters using a uniform distribution centered at the mean with upper and lower bounds $\pm 10\%$ for the following variables:

(a) $\mu_v = 0.5$

(b) $\mu_D = 0.125$

(c) $\mu_\omega = 0.1$

and sample values of the following parameters in their given ranges:

(a) $\Delta x \sim [0.001, 0.5]$

(b) $\Delta t \sim [0.001, 0.5]$

Using regression, estimate the sensitivities to each parameter.
This python script was used to generate the data:

Listing 9: Code for Problem

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

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

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

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

N=1000;tend=5;xend=10;

output=open("output","w")
print("v,D,w,dt,dx,RXNRate",file=output)

```

```
NCount=1
while NCount<=N:

    #Generate Samples;        loc = mean; scale=STD
    v=np.random.uniform(0.5-0.5*0.1,0.5+0.5*0.1)
    D=np.random.uniform(0.125-0.125*0.1,0.125+0.125*0.1)
    w=np.random.uniform(0.1-0.1*0.1,0.1+0.1*0.1)
    dt=np.random.uniform(0.001,0.5)
    dx=np.random.uniform(0.001,0.5)

    NX=int(xend/dx)
    Nt=int(tend/dt)

    print("space_divs time_divisionts = "+str(NX*Nt)+" NCount = "+str(NCount))

    #Problem boundaries
    x=fun.np.linspace(0,xend,NX)
    dx=x[1]-x[0]
    t=fun.np.linspace(0,tend,Nt)
    dt=t[1]-t[0]

    #Build A
    A=fun.BuildA(v,D,w,dt,dx,NX)
    #Calculate QoI
    RXNRate=fun.SolveQoI(NX,Nt,x,t,A,dt,w)
    try:
        if np.isnan(RXNRate):
            continue
    except TypeError:
        continue
    if(RXNRate<0 or RXNRate>100):
        print("Skip print fragile solve")
        continue

    print(",".join([str(v),str(D),str(w),str(dt),str(dx),str(RXNRate)]),
          file=output)
    NCount=NCount+1

###################### Time To execute ################
output.close() #important or ORIGEN2 wont run
print("--- %s seconds ---" % (time.time() - start_time))
```

Th script below was used to fit a regression model

$$RXNs = v + D + w + dt + dx + \text{Intercept}$$

the coefficients of which are an estimate on

$$\frac{\delta \text{QoI}}{\delta \theta_i}$$

where i is a particular parameter.

Listing 10: Regression with Pandas

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

```
#################################################################
#################### Import packages ############################
#################################################################

import time
start_time = time.time()
import Functions as fun
import numpy as np
import pandas as pd

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

Df=pd.read_csv('output')
Vars=['v','D','w','dt','dx']
Df2=Df[Vars]
model=pd.ols(y=Df['RXNRate'],x=Df2)

print(model.summary().as_latex())

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

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

Table 8: Sensitivity Parameters dx = 0.1 dt = 0.5

Variable	Coef	Std Err	t-stat	p-value
v	0.1945	8.5335	0.02	0.9818
D	-25.2844	33.3347	-0.76	0.4483
w	76.5716	42.3267	1.81	0.0707
Δt	-7.2201	1.8227	-3.96	0.0001
Δx	30.8771	2.6986	11.44	0.0000
intercept	-5.1421	7.2558	-0.71	0.4787

The R^2 value for this fit is 0.12, which isn't very good. Also these fitted values are fairly large, much larger than estimated in the previous problem. The reason behind this is probably due to least squares fitting, which in problem 1, was shown to give large values for coefficients.

Another issue is the 'sneaky weeding' out of the data I did in my code. Sometimes the QoI was either negative, or very large...or both. There was also cases where there was an overflow error, which may not depend on the total number of divisions in the problem (failed in this manner with smaller total bin numbers - but maybe there were few spatial divisions and a ton of time divisions, anyway it seemed random). Both these instances, weird results and overflow errors, were filtered out and not used in the regression model.

Long Problem 4

Using a discretization of your choice, solve the equation

$$\frac{\delta u}{\delta t} + v\frac{\delta u}{\delta x} = D\frac{\delta^2 u}{\delta x^2} - \omega u$$

for $u(x,t)$ on the spatial domain $x \in [0,10]$ with periodic boundary conditions $u(0^-) = u(10^+)$, and initial conditions

$$u(x,0) = \begin{cases} 1, & x \in [0,2.5] \\ 0, & \text{otherwise} \end{cases}$$

Use the solution to compute the total reactions

$$\int_5^6 dx \int_0^5 dt\ \omega u(x,t).$$

Compute the probability that this quantity of interest is greater than 0.035 using LHS sampling of 50 points, 50 points of a Halton sequence, and a first-order second moment method using the following distributions

(a) $\mu_v = 0.5,\ \sigma_c = 0.1$

(b) $\mu_D = 0.125,\ \sigma_D = 0.03$

(c) $\mu_\omega = 0.1,\ \sigma_\omega = 0.05$

How do these results change with changes in Δx and Δt?

Will assume a normal distribution and filter out any negative results. Also I do not know how to do a first-order second moment method in this context. These plots were produced by holding two parameters at the mean value, and sampling the third with one of the two methods asked for in the problem.

Varying the D parameter with Δx produced two different plots for the two different sampling methods. After rerunning both those cases I concluded that there is either a bug in my code, or one of the Halton sampling scheme didn't sample in the failure area as well as the LHS.

Listing 11: Code for Problem - run many times

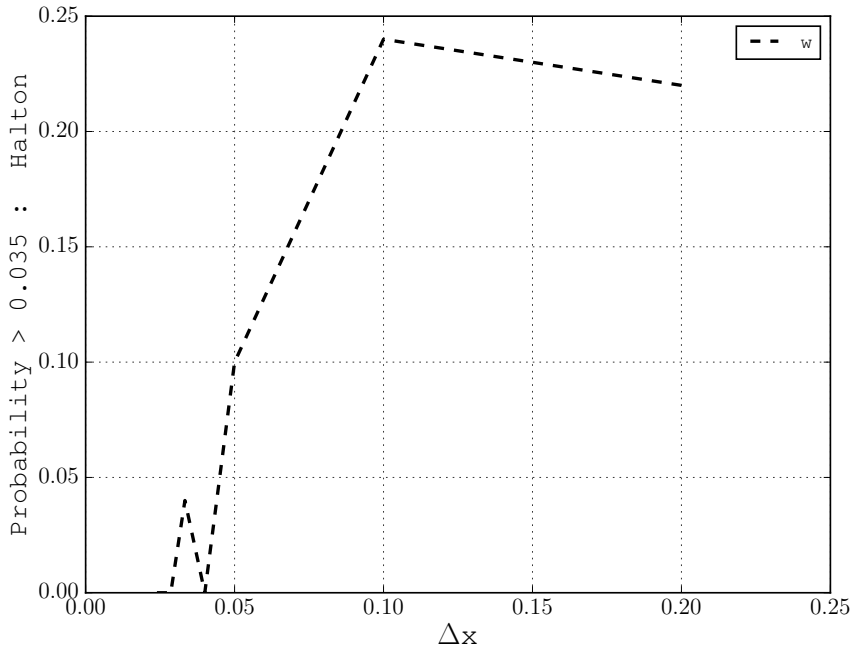
```
#!/usr/bin/env python3

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

import time
start_time = time.time()
import Functions as fun
from scipy.stats import norm
import lhsmdu
import numpy as np

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

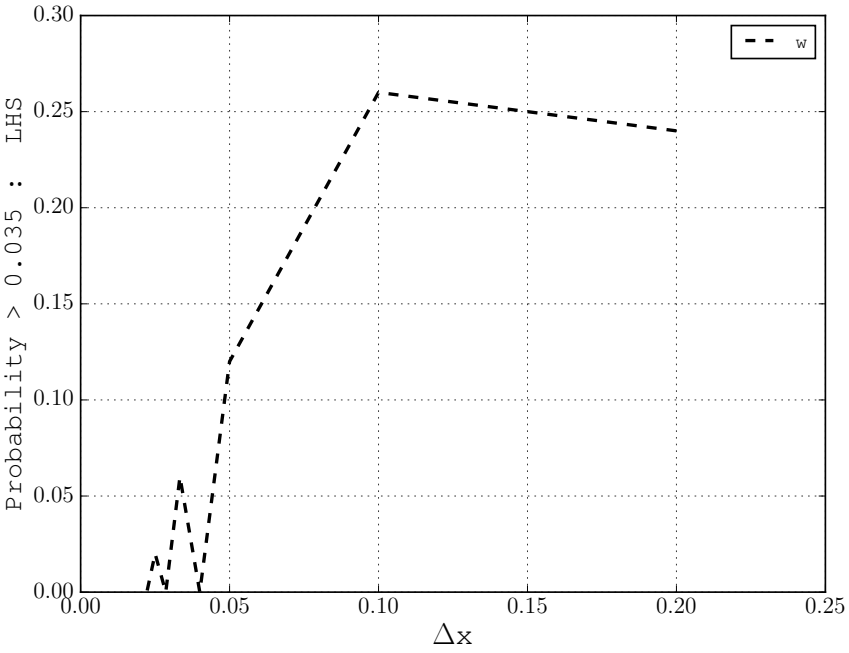
Long Problem 4 continued on next page...

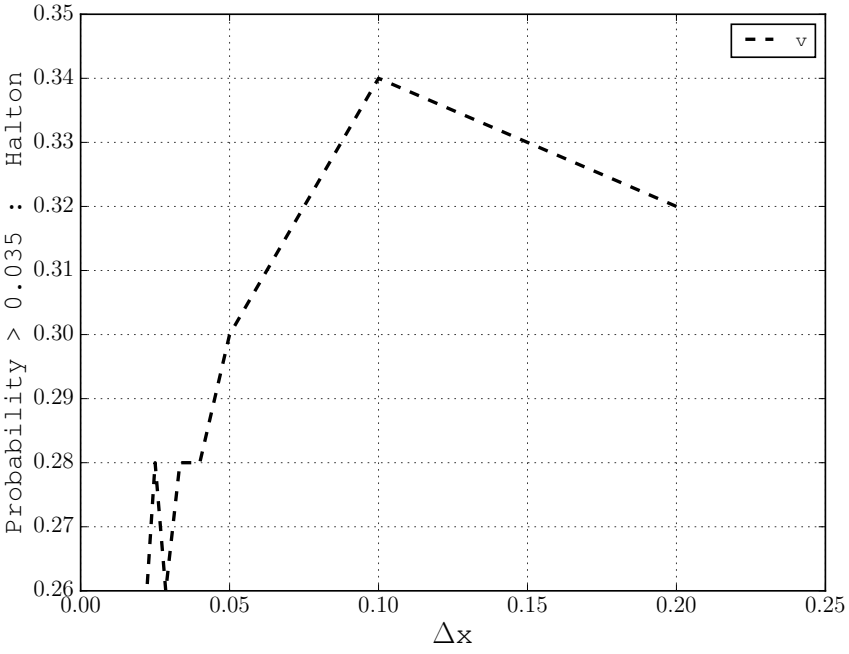
```
N=50
#RandN=fun.Rvdc(N,2);Name='Halton';Halton=True #Halton Sequence
l=lhsmdu.sample(1,N)  #Hyper cube sampling
RandN=l[0].A1;Name='LHS';Halton=False

#Generate Samples;        loc = mean; scale=STD
v=np.ones(N)*0.5
#mu=0.5;sigma=0.1;Param='v';v=norm.ppf(RandN,mu,sigma)
D=np.ones(N)*0.125
mu=0.125;sigma=0.03;Param='D';D=norm.ppf(RandN,mu,sigma)
w=np.ones(N)*0.1
#mu=0.1;sigma=0.05;Param='w';w=norm.ppf(RandN,mu,sigma)

NX=[51,101,201,251,301,351,401,451];Vary='x';Nt=[11];l=0
#Nt=[11,21,31,41,51,61];Vary='t';NX=[401];j=0

AllProbs=[];dxs=[]

for j in range(0,len(NX)):   #Loop over space
#for l in range(0,len(Nt)):    #Loop over time

    #Problem boundaries
    x=fun.np.linspace(0,10,NX[j])
    dx=x[1]-x[0]
    t=fun.np.linspace(0,5,Nt[l])
    dt=t[1]-t[0]

    RXNs=[];

    for i in range(0,N):
        #Build A
        A=fun.BuildA(v[i],D[i],w[i],dt,dx,NX[j])
        #Calculate QoI
        RXNRate=fun.SolveQoI(NX[j],Nt[l],x,t,A,dt,w[i])
        RXNs.append(RXNRate)

    #Find the probability of f being greater than 0.035
    PGreater=sum(i>0.035 for i in RXNs)/N

    try:
        if np.isnan(RXNRate):
            continue
    except TypeError:
        continue
    if(RXNRate<0 or RXNRate>100):
        print("Skip print fragile solve")
        continue

    AllProbs.append(PGreater)
    if Vary=='x':
        dxs.append(dx)
    elif Vary=='t':
```

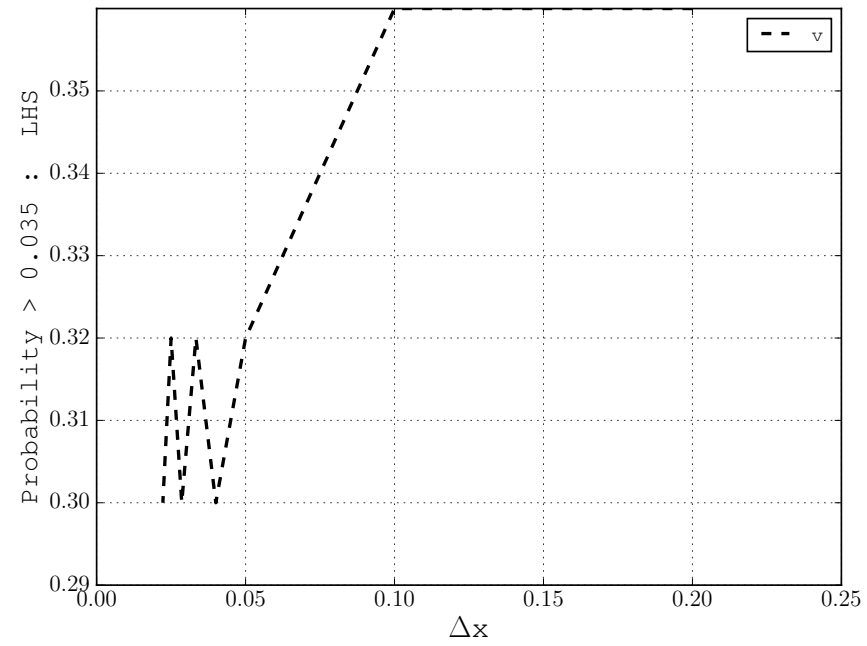
Long Problem 4 continued on next page...

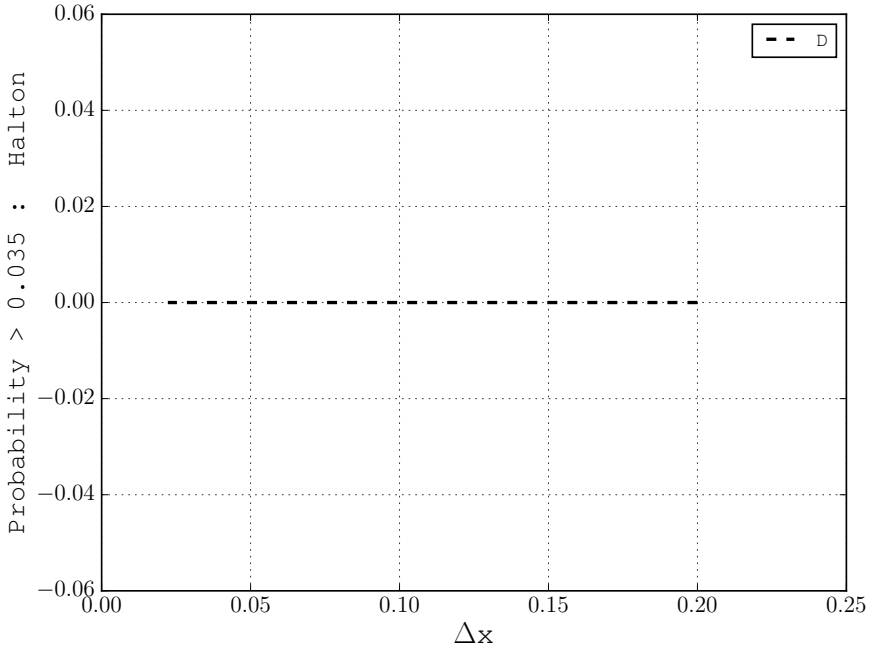
```
        dxs.append(dt)

#Plot the data
fig = fun.plt.figure()
ax = fig.add_subplot(111)
(fig,ax)=fun.Plot(dxs,AllProbs,Param,fig,ax,'k--',Vary,Scaled=Halton)
filename=Name+'_'+Param+Vary+'.pdf'
handles,labels=ax.get_legend_handles_labels()
Lfont={'family':'monospace',
       'size':12}
ax.legend(handles,labels,loc='best',fontsize=12)
fun.plt.savefig(filename)


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

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



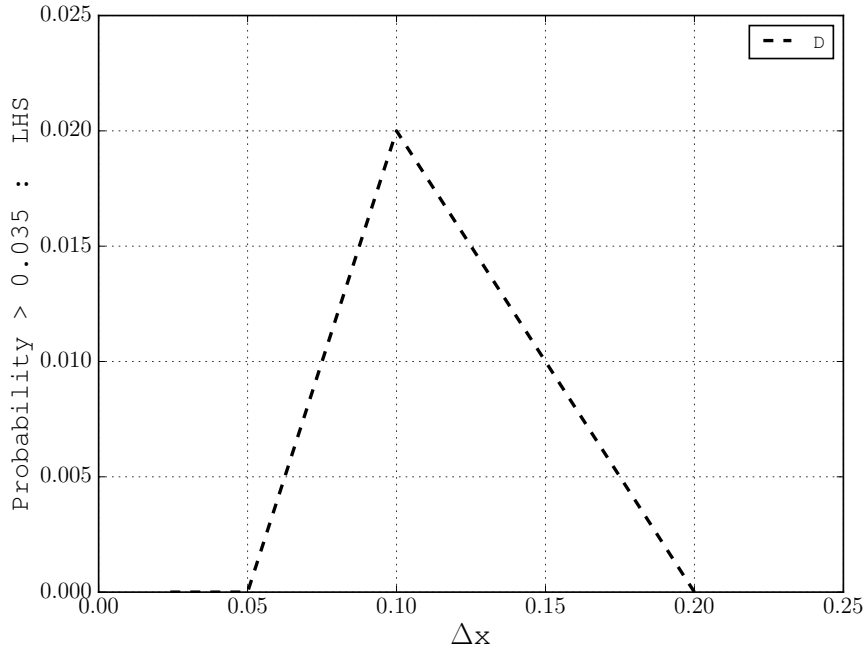

w
Probability > 0.035 : LHS
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Δx

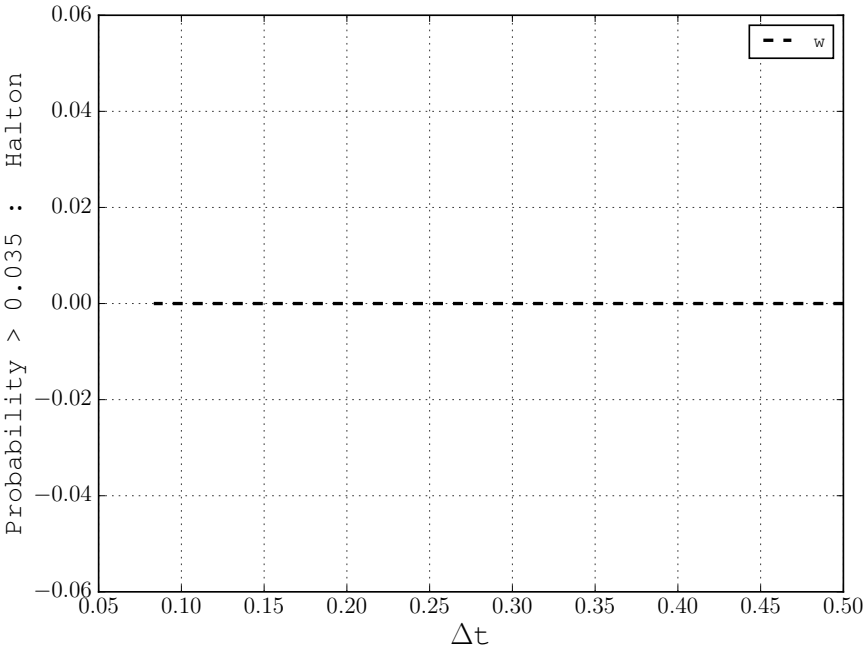

v
Probability > 0.035 : Halton
0.26
0.27
0.28
0.29
0.30
0.31
0.32
0.33
0.34
0.35
0.00
0.05
0.10
0.15
0.20
0.25
Δx

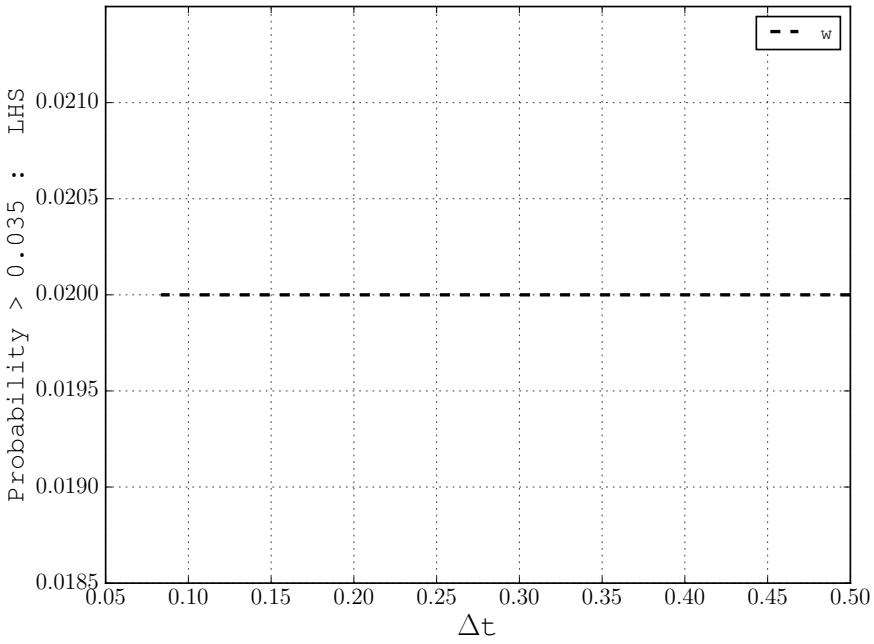


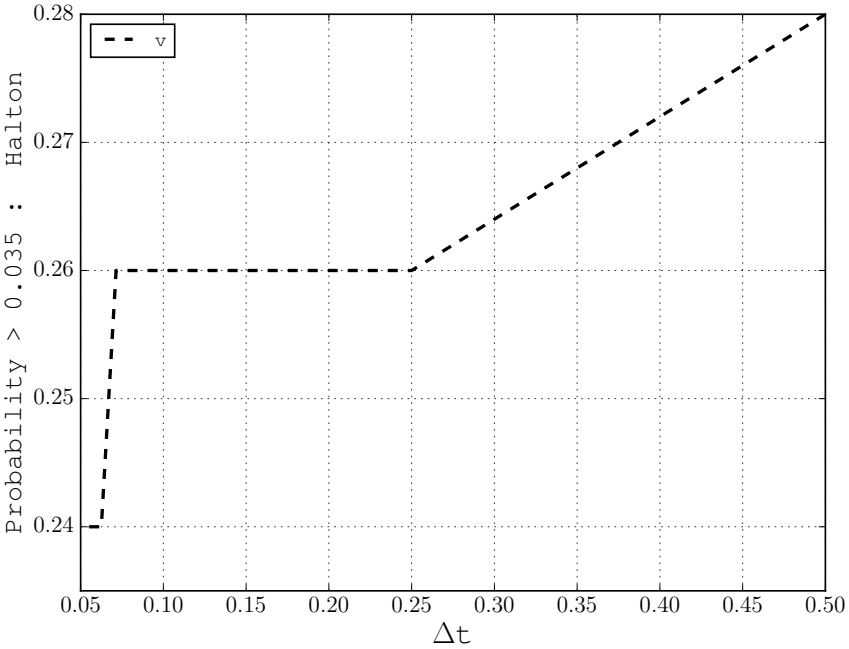


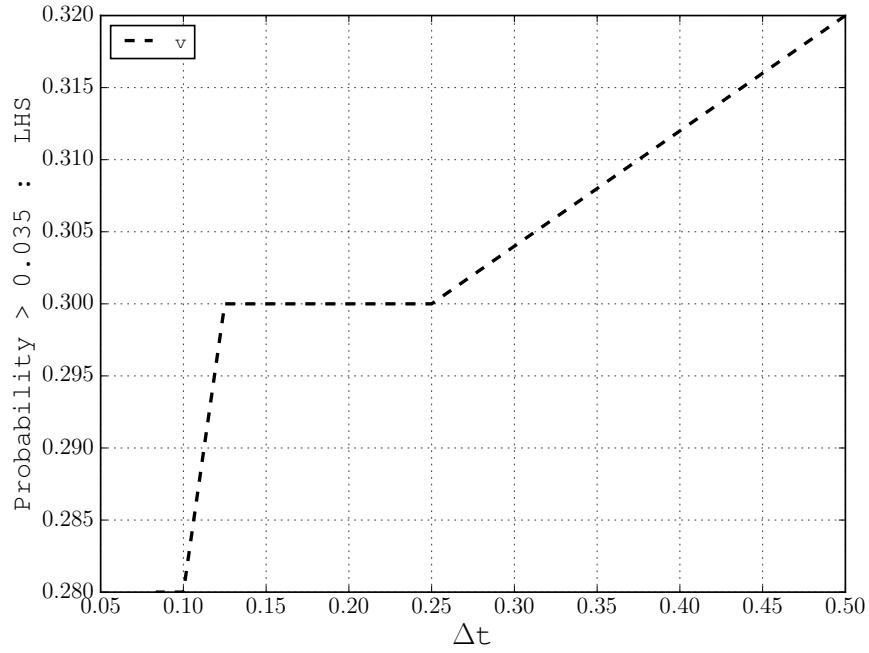
Long Problem 4 continued on next page...

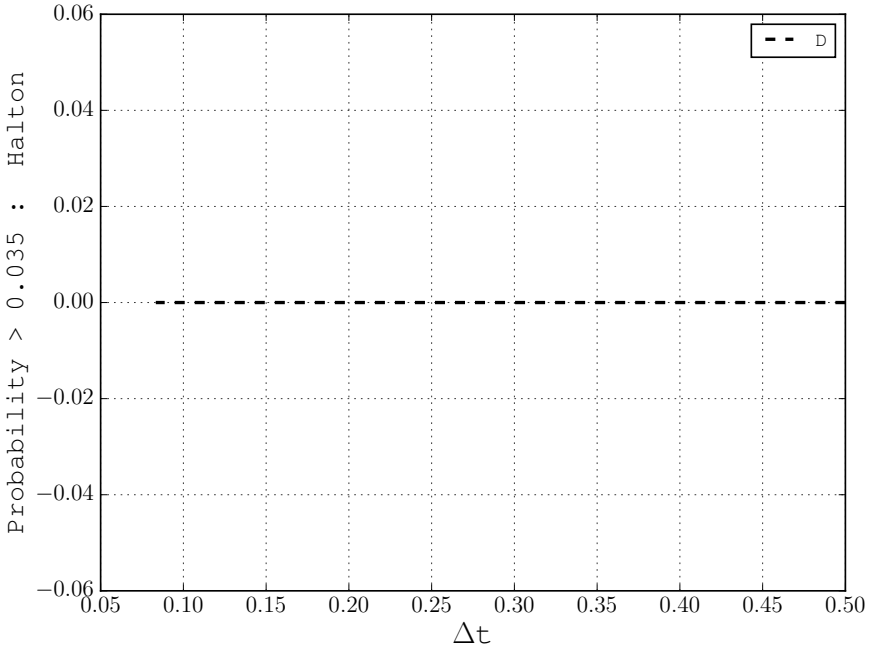

D
Probability > 0.035 : LHS
0.025
0.020
0.015
0.010
0.005
0.000
0.00
0.05
0.10
0.15
0.20
0.25
Δx

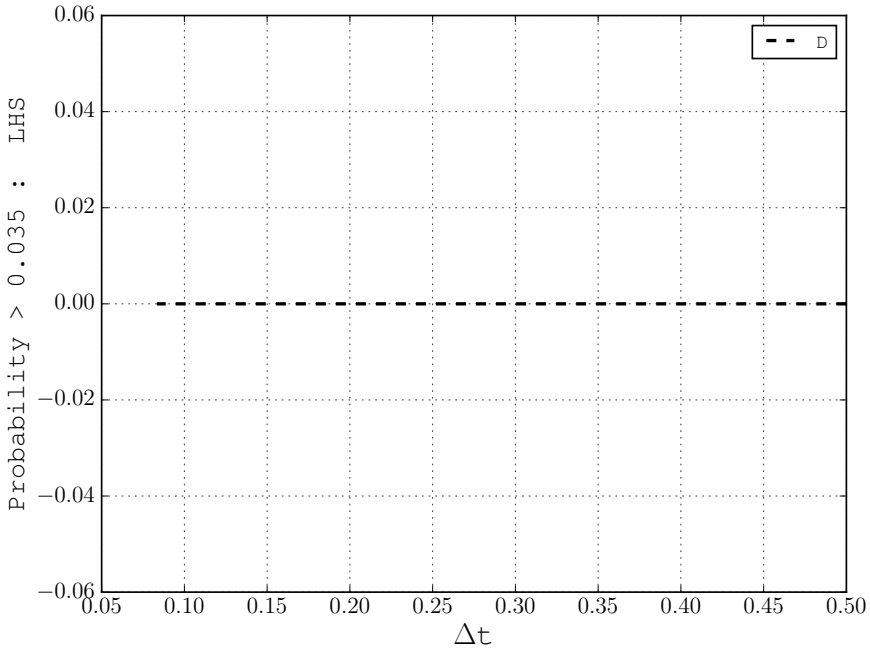

w
Probability > 0.035 : Halton
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt


w
Probability > 0.035 : LHS
0.0210
0.0205
0.0200
0.0195
0.0190
0.0185
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt


v
Probability > 0.035 : Halton
0.28
0.27
0.26
0.25
0.24
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt


v
Probability > 0.035 : LHS
0.320
0.315
0.310
0.305
0.300
0.295
0.290
0.285
0.280
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt


D
Probability > 0.035 : Halton
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt


D
Probability > 0.035 : LHS
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
Δt