**Guy Dor**

**CSC 521 Section 901**

**Monte Carlo Algorithms Final Project**

**Problem 1**

**Approach, Methodology, and Solutions**

**General Approach:**

The primary assignment is to take data of losses from two plants over four years in order to model the distribution of losses of the individual plants, and their daily frequencies of loss over one year.

The explanation has also been given that the time between losses for both of two plants follow an exponential distribution, and that the log of the two plants’ losses follows a gaussian distribution (i.e. lognormal). This will allow me to model the simulation without relying on resampling or having to determine the distribution beforehand.

Initially, the data must be extracted and processed in python to determine the basic information about the data, including the sample means and standard deviations of the loss events and when they have occurred for each plant. From this information, the data can be modeled by pseudo-random number generators that use the information as parameters. The exponential distribution of time uses the mean represented by lambda, while the gaussian distribution of the log loss amounts uses the mean and standard deviation represented by mu and sigma. These pseudo-random number generators should be run twice, once for plant A, and once for plant B, as the distribution of losses and time between losses are independent for both plants.

Once the data has been modeled and the PRNGs are programmed to use the modeled parameters, the rest of the process is relatively simple. The simulate\_once() function is created in order to simulate one year of losses for both plants, in this case scaled to 365 days since the rate of loss is based on days per year. The simulation runs 365 days of randomly generated losses (or no losses if the exponential random variable generates outside the parameter to record a loss). Whenever the event of a loss is recorded, it also records a loss amount from the gaussian random number generator. Once the programmed “timer” falls outside 365 days, the simulation stops, aggregates the results, and generates the total estimated yearly loss for the company.

The simulate\_many() function is programmed to take the simulate\_once() function, run the simulation over a selected amount of iterations, and produce an estimated mean of the results (in this particular case, with a relative precision of 10%). The simulate\_many() function also has a bootstrap function that produces error intervals for the estimated mean.

Once the ability to run a simulate\_once() multiple times is achieved, the company can determine how to budget for a percentage of yearly losses (specifically 90%) by running and recording the simulate\_once() multiple times. The 90th percentile of the recorded results will produce the estimated 90% of total losses that the company could be subject to, therefore also being the amount that should be budgeted for to cover 90% of losses in a year.

**Parameter extraction Methodology and Solutions:**

To model the simulation I must first learn the parameters via python to model for the simulate\_once() function. To do so, I needed to code for which losses occurred in which year for which plant, as well as using logarithms and some other simple math to extract the means and standard deviations for the distributions of time and loss for both Plant A and Plant B.

The average number of accidents per year in plant A:

**33**

The average number of accidents per year in plant B:

**39**

The average loss per accident in plant A:

**$17,470.155555555557**

The average loss per accident in plant B:

**$2,022.9615384615386**

The average loss in total per year in plan A:

**$589,617.75**

The average loss in total per year in plan B:

**$78,895.50**

The average log loss per accident in plant A:

**9.146099829385644**

The average log loss per accident in plant B:

**6.95909167211235**

The standard deviation of log loss per accident in plant A:

**1.061510583198764**

The standard deviation of log loss per accident in plant B:

**1.133381024878168**

From the results above, and from the instructions information, we know that the distribution of time between losses for Plant A are exponential with a lambda parameter of 33/365 (scaling the rate of accidents per day), while Plant B has an exponential distribution of time between losses with a lambda parameter of 39/365. We also yield results for the distribution of losses for Plant A, which is lognormal with a mean of 9.146099829385644 and a sigma of 1.061510583198764, While Plant B’s distribution of losses is lognormal with a mean of 6.95909167211235 and a sigma of 1.133381024878168.

Mapping these parameters will help to simulate loss daily occurrences for each plant over a year.

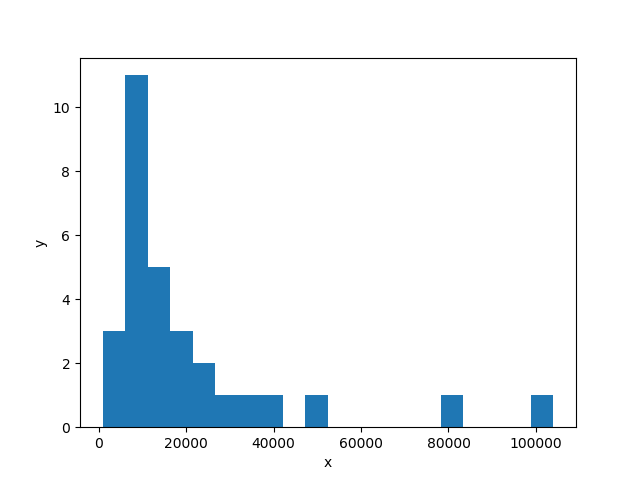
**Simulate\_once() Methodology and Solutions:**

In order to implement a simulate\_once() that simulates one year of losses for both plants, I entered into the function the parameters of log losses (mu and sigma) as well as the time parameters(lambda) of both individual plants independently. Once this was done, I modeled the time between losses of the two plants as random.expovariate(lambda) and the losses as exp(random.gaussian(mu,sigma)). The exp() function transforms the modeled log loss back into loss.

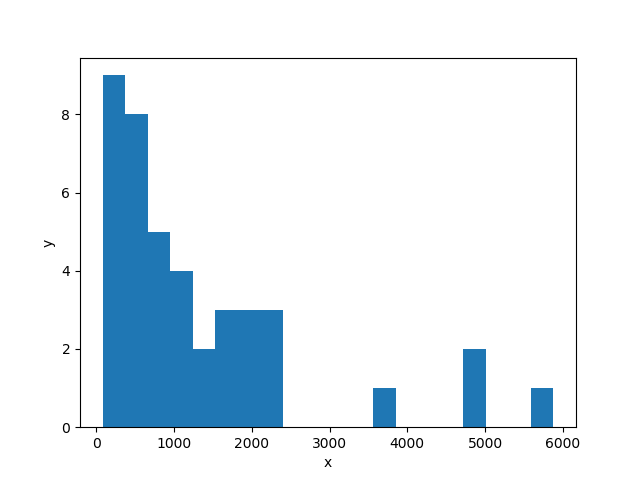
The modeled random variables are then looped for a virtual 365 days. In this loop, if the time iteration was within the randomly generated time variable, I recorded and aggregated a loss that is randomly generated from the gaussian variable. This is done independently for each plant.

Each simulate once obviously gives a slightly different answer, but in one iteration the total loss from both plants gives **$665,119**, which is very similar to the sum of the average total loss of the two plants. Below is two histograms of the distribution of losses of both plants from the simulate\_once() output.

Loss from Plant A

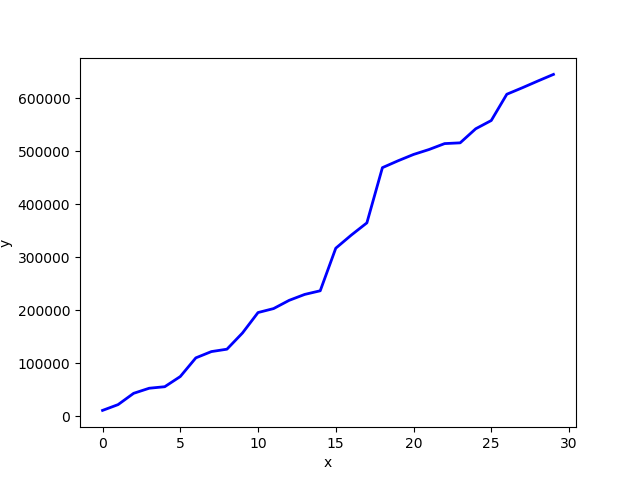


Loss from Plant B



After aggregating each simulated loss from the two plants for one year (365 days), the program then adds the two loss vectors together. The chart below is a running total of the total loss of both plants over time in a single simulation.

Losses vs Time



**Simulate\_many() Methodology and Solutions:**

In order to run a simulate\_many() with a relative precision of 10% and using bootstrapping, I implemented the MCEngine from the nlib package and created a class called LossSimulator(MCEngine). This class included the simulate\_once() function and iterated 5000 simulations. With a relative precision of 10%, it produced the result below:

**(588034.2067844407, 635101.8950825462, 686214.3608994514)**

This result means that with a relative precision of 10%, the average yearly total loss is **$635,101.90** with a bootstrap interval between **$588,034.21** and **$686,214.36**. Once again, the average is consistent with the sum of yearly average loss from both plants.

**Budgeting for loss Methodology and Solution:**

To determine how much should the company budget to make sure that it can cover these losses in 90% of the simulated scenarios, I simply ran the simulate\_once() 100 times and appended the aggregated yearly losses to a vector. After sorting the vector, and looking up the vector value at index 90, I received a result of **$773,442.95**. This result suggests that to cover 90% of losses, the company should budget $773,442.95 per year.

**Appendix**

```python

import csv

import math

from nlib import \*

import random

from math import exp, log

import datetime

```

```python

import os

os.getcwd()

```

'C:\\Users\\guy.dor\\Documents\\CSC 521'

```python

filename = 'accidents.csv'

history = []

log\_loss = []

Ahist=[]

Bhist=[]

Aplant=[]

Aday=[]

Ayear=[]

Aloss=[]

Alog\_loss=[]

Bplant=[]

Bday=[]

Byear=[]

Bloss=[]

Blog\_loss=[]

with open(filename) as myfile:

reader = csv.reader(myfile)

header = ['Plant','Day','Loss']

rows = [dict(zip(header,row)) for row in reader]

rows.reverse()

for k,row in enumerate(rows):

plant = row['Plant']

day = int(row['Day'])

loss = float(row['Loss'])

log\_loss = math.log(loss)

if 0<day<=365:

year=1

elif 366<day<=730:

year=2

elif 730<day<=1095:

year=3

elif day>1095:

year=4

history.append([plant, day, year, loss, log\_loss])

if plant=='A':

Ahist.append([plant, day, loss, log\_loss])

Aplant.append(plant)

Aday.append(day)

Ayear.append(year)

Aloss.append(loss)

Alog\_loss.append(log\_loss)

else:

Bhist.append([plant, day, loss, log\_loss])

Bplant.append(plant)

Bday.append(day)

Byear.append(year)

Bloss.append(loss)

Blog\_loss.append(log\_loss)

print plant, day, year, loss, log\_loss

```

B 1462 4 3626.0 8.19588539131

A 1455 4 11561.0 9.35539264368

A 1453 4 34881.0 10.4596975473

B 1452 4 1718.0 7.44891610254

B 1449 4 1904.0 7.55171221535

A 1443 4 9923.0 9.20261057391

B 1441 4 1218.0 7.10496544827

A 1440 4 4400.0 8.38935981991

A 1430 4 2767.0 7.92551897979

………………………………………………………………………………………

```python

#Average number of acidents/year in plant A

def accidents\_in\_year(n,plant):

accidents=0

for i in plant:

if i==n:

accidents+=1

return accidents

(accidents\_in\_year(1, Ayear)

+accidents\_in\_year(2, Ayear)

+accidents\_in\_year(3, Ayear)

+accidents\_in\_year(4, Ayear))/4

```

33

```python

#Average number of acidents/year in plant B

(accidents\_in\_year(1, Byear)

+accidents\_in\_year(2, Byear)

+accidents\_in\_year(3, Byear)

+accidents\_in\_year(4, Byear))/4

```

39

```python

#Average loss per accident in plant A

def Expected(val):

r=float(sum(val))/len(val)

return r

Expected(Aloss)

```

17470.155555555557

```python

#Average loss per accident in plant B

Expected(Bloss)

```

2022.9615384615386

```python

#Average Total loss per year in plant A

Expected(Aloss)\*len(Ahist)/4

```

589617.75

```python

#Average Total loss per year in plant B

Expected(Bloss)\*len(Bhist)/4

```

78895.5

```python

#Average Log loss per accident in plant A

Expected(Alog\_loss)

```

9.146099829385644

```python

#Average Log loss per accident in plant B

Expected(Blog\_loss)

```

6.95909167211235

```python

#Standard Deviation of Log loss per accident in plant A

sd(Alog\_loss)

```

1.061510583198764

```python

#Standard Deviation of Log loss per accident in plant A

sd(Blog\_loss)

```

1.133381024878168

```python

#Simulate once for one year of losses for plants A and B

def simulate\_once():

N = 1

max\_time = 365 # 1 year

xm\_a = 9.146099829385644 # mu of log loss A

sd\_a = 1.061510583198764 # sd of log loss A

lamb\_a = float(33)/365 # accidents per day

xm\_b = 6.95909167211235 # mu of log loss B

sd\_b = 1.133381024878168 # sd of log loss B

lamb\_b = float(39)/365 # accidents per day

time = [0.0] \* N

ta = 0 # timer for loss at plant A

tb = 0 # timer for loss at plant B

sima=0 # Loss aggregator for plant A

va=[]

simb=0 # Loss aggregator for plant A

vb=[]

hista=[]

histb=[]

while True:

ta = ta + random.expovariate(lamb\_a) # time between losses~exponential dist for plant A with lambda=lamb\_a

if ta > max\_time: break

for k in range(N):

if time[k] <= ta:

rva=(exp(random.gauss(xm\_a,sd\_a))) # Log Loss for A~N(xm\_a,sd\_a). exp() transforms log of Loss into Loss

sima+=rva

va.append(rva)

hista.append(sima)

else:

sima+=(0)

va.append(0)

hista.append(0)

while True:

tb = tb + random.expovariate(lamb\_b) # time between losses~exponential dist for plant B with lambda=lamb\_b

if tb > max\_time: break

for k in range(N):

if time[k] <= tb:

rvb=(exp(random.gauss(xm\_b,sd\_b))) # Log Loss for B~N(xm\_b,sd\_b). exp() transforms log of Loss into Loss

simb+=rvb

vb.append(rvb)

histb.append(simb)

else:

sima+=(0)

va.append(0)

hista.append(0)

total\_hist=[]

th=list(map(sum,zip(hista,histb)))

for i in th:

total\_hist.append(((th.index(i)),(i)))

Canvas().plot(total\_hist).save('Loss\_Timeplot.png')

Canvas().hist(va).save('simulated\_loss\_Plant\_A.png')

Canvas().hist(vb).save('simulated\_loss\_Plant\_B.png')

return (sima+simb)

simulate\_once()

```

665118.599115779

```python

#Simulate many for one year of losses for plants A and B

class LossSimulator(MCEngine):

def \_\_init\_\_(self, N):

self.N = N

def simulate\_once(self):

N = self.N

max\_time = 365 # 1 year

xm\_a = 9.146099829385644 # mu of log loss A

sd\_a = 1.061510583198764 # sd of log loss A

lamb\_a = float(33)/365 # accidents per day

xm\_b = 6.95909167211235 # mu of log loss B

sd\_b = 1.133381024878168 # sd of log loss B

lamb\_b = float(39)/365 # accidents per day

time = [0.0] \* N

ta = 0 # timer for loss at plant A

tb = 0 # timer for loss at plant B

sima=0 # Loss aggregator for plant A

simb=0 # Loss aggregator for plant A

while True:

ta = ta + random.expovariate(lamb\_a) # time between losses~exponential dist for plant A with lambda=lamb\_a

if ta > max\_time: break

for k in range(N):

if time[k] <= ta:

sima+=(exp(random.gauss(xm\_a,sd\_a))) # Log Loss for A~N(xm\_a,sd\_a). exp() transforms log of Loss into Loss

else:

sima+=(0)

while True:

tb = tb + random.expovariate(lamb\_b) # time between losses~exponential dist for plant B with lambda=lamb\_b

if tb > max\_time: break

for k in range(N):

if time[k] <= tb:

simb+=(exp(random.gauss(xm\_b,sd\_b))) # Log Loss for B~N(xm\_b,sd\_b). exp() transforms log of Loss into Loss

else:

simb+=(0)

return (sima+simb)/N

sim=LossSimulator(5000)

print sim.simulate\_many(rp=.1)

```

(588034.2067844407, 635101.8950825462, 686214.3608994514)

```python

# find 90% coverage of loss

v = []

for i in range(100):

v.append(sim.simulate\_once())

v.sort()

print v[90]

```

773442.953903