DEPAUL UNIVERSITY

Monte Carlo Simulation

Problem 1 – Final Project CSC521

Diana Bursac

June 2018

Contents

1.1 Introduction

This project explores Monte Carlo simulation that utilizes random numbers to model or imitate the real-world process or the mathematical model of the process. In the analysis we utilize the Problem1 data set that is available on the following link: http://mdipierro.github.io/DePaul/CSC521/accidents.csv. The data set contains recorded accidents in two plants over the last 4 years including exact days when the accidents happened and losses caused by the accidents. In the simulated model as an input value we use the parameters that were estimated from the real data set. The computation of the relevant parameters is described in Chapter 1.2. The random number generation according to the given probability distribution functions for exponential and log normal distribution is explained in Chapter 1.3. The results of the simulation are explained in the Chapter 1.5. We can see that the simulation confirmed some numerical calculations from Chapter 1.2. In addition, by simulating one year losses in the both plants, it was possible to answer questions related to budget planning since the company wanted to make sure that the planned budget is sufficient to cover losses in 90% of the simulated scenarios. The simulation also shows that the average yearly loss depends on the number of times we run 'simulate once' and the relative precision. In other words the number of Monte Carlo steps or relative precision can influence how precise the result of simulation is and if it converges to the true value. For the purpose of the simulation the MCEngine from nlib library is used. The functions of MCEngine are described in Chapter 1.4.

1.2 Data exploration

In this section, we analyze the losses in plant A and plant B, with data collected over a period of 4 years. In Figure 1 we can see that the accidents that caused the losses are happening on certain days and only one accident is happening per day. At first glance we can conclude that in plant A the losses are mostly in the range of up to $10,000.00 although some values are higher. For plant B most of the losses are in the range of $2,500.00.

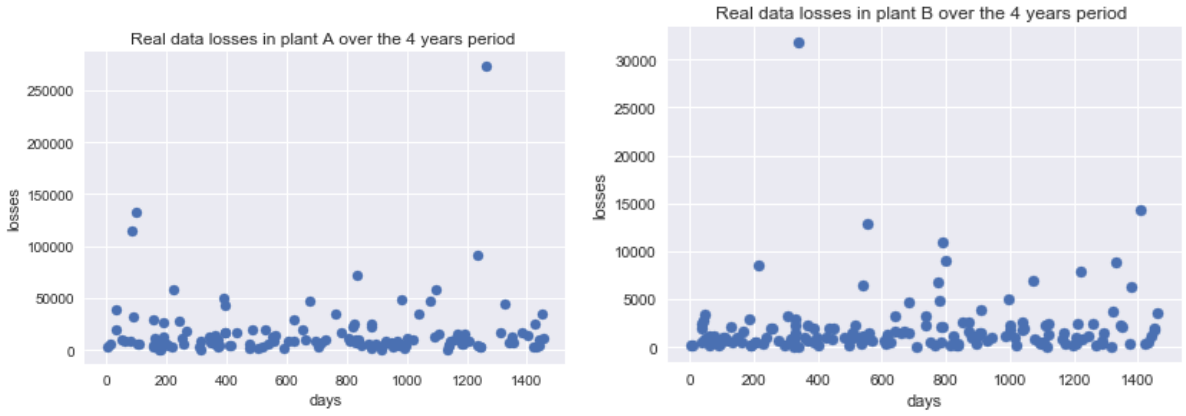

Figure 1 Real data loses over the 4 years period

From the real data set we can compute the parameters like mean, standard deviation and the average number of accidents in one year. We can use those parameters later on for the random

numbers generations used in the model where the simulation is performed based on the behavior of the real data set. This is later described in the Chapter 1.3. It would be also interesting to compute the average loss in one year and later on to compare that result with the simulated average yearly loss.
Since the data were recorded over a period of 4 years, the average loss in total per year is calculated when the total loss over the 4 years in each plant is divided by 4. Accordingly,

avg_loss_year in plant A = sum(lossA over 4 years)/4 = 589,617.75
avg_loss_year in plant B = sum(lossB over 4 years)/4 = 78,895.5

The average number of accidents per year is calculated as number of accidents over 4 years divided by 4:

average_no_accidents in plant A = sum(no.accidents A over 4 years)/4 = 33.75
average_no_accidents in plant B= sum(no.accidents B over 4 years)/4 = 39

The average loss per accident is calculated as the total loss in the plant over 4 years divided by the total number of accidents in the plant over 4 years:

avg_lossA_per_accident= sum(lossA over 4 years) / sum(no.accidents A over 4 years) = 17,470.16
avg_lossB_per_accident= sum(lossB over 4 years) / sum(no.accidents B over 4 years) = 2,022.96

In addition, we can calculate the parameters like mean of natural log loss and standard deviation of natural log loss. We are going to use these parameters in the model for generating random numbers that are going to simulate the log normal distribution of the losses. In Chapter 1.3 we use those parameters to visualize the cumulative distribution function of the log normal distribution and in Chapter 1.4 the generation of random numbers is explained.

muA = mean(log_lossA) = sum (log_loss A over 4 years) / number of lossA over 4 years = 9.146
muB = mean(log_lossB) = sum (log_loss B over 4 years) / number of lossB over 4 years = 6.959

sigmaA = mean(log_loss A)^2 – (muA)^2 = 1.0615
sigmaB = mean(log_loss B)^2 - (muB)^2 = 1.1333

The aforementioned calculations are performed in python code (see Chapter 1.7 Appendix).

1.3 Exponential and Log Normal distributions

In the given model two assumptions are made. In the first it was assumed that the time interval between the accidents is an exponential distribution ~ Exp(λ). This practically means that a Poisson process is considered for modelling the occurrence of accidents in the plant. For the Poisson process the probability of two accidents / loss events occurring at the same instant is "negligible" or equal to zero. In addition, the number of accidents occurring in a finite interval

after time ‘t’ does not depend on the number of accidents occurred before time ‘t’. In particular, this means we can simulate a Poisson process with intensity λ by simply generating the time intervals between Poisson events, Xi , where Xi ~ Exp(λ). The intensity of the Poisson process (λ) measures the average number of loss events per time unit. Since our simulation is focused on a one-year time interval, in this project λ means the average number of accidents in one year. The cumulative distribution function (CDF) of the exponential distribution is given by the formula:

$F(x) = \int_0^x p(x)dx = \int_0^x \lambda * e^{-\lambda x}\,dx = 1 - e^{-\lambda x}$, where $p(x) = \lambda * e^{-\lambda x}$ is the probability density function (PDF) of exponential distribution.

In Figure 2 below we can see the comparison between the CDF of the time intervals between the accidents captured from the given data set in both plants and the CDF distribution that was modeled by using a formula for the exponential distribution F(x) as described above.
The CDF distribution of the generated exponential distributed x variables equals to CDF distribution of the time intervals. The parameter λ in the formula is calculated from the original data set as described in the Chapter 1.2.
Average number of accidents in year for the plant A is λ(A) = 33.75 or per day λ(A) = 33.75/365.
Average number of accidents in year for the plant B is λ(A) = 39 or per day λ(A) = 39/365.
The time intervals between the two consecutive accidents in plant A varies from 0 to 50 days while in plant B varies from 0 to 41 days.

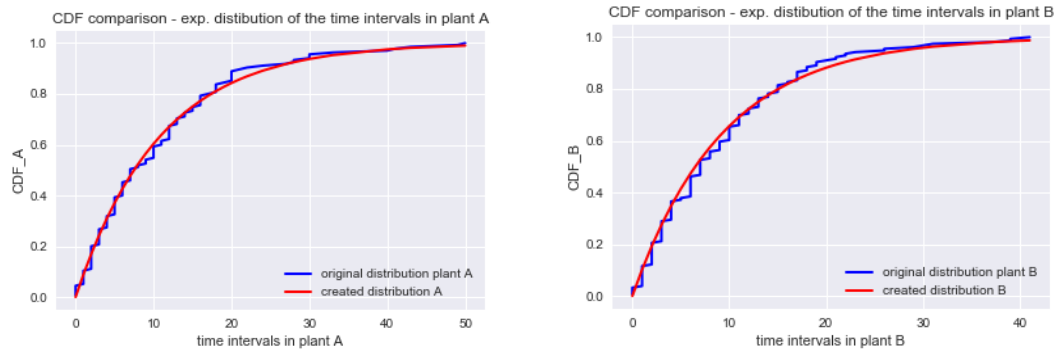

Figure 2 CDF comparison of exponential distributions of real data vs model

In Figure 3 we can see the histogram of time intervals for plant A and plant B computed from the original data set:

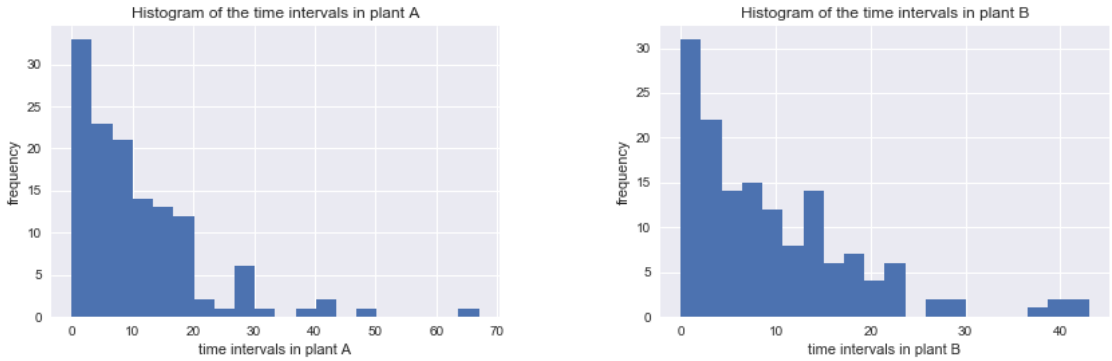

Figure 3 Histogram of real data time intervals

In the given model the second given assumption is that loss is a log normal. In other words it means that the natural log of a loss due to a single accident has a Gaussian distribution ~ N(μ, σ^2). Likewise, if a random variable X is log-normally distributed, then Y = ln(X) has a normal distribution. If Y has a normal distribution, then the exponential function of Y, X = exp(Y), also has a log-normal distribution.
The cumulative distribution function (CDF) of the log normal distribution can be calculated by using formulas listed below where F(x) is the CDF of variable x and ϕ (phi) is the CDF of N(0,1) distribution

$$F(x) = \frac{d}{dx}\Pr(X \le x) = \frac{d}{dx}\Pr(lnX \le lnx) = \frac{d}{dx}F(lnx) = \frac{d}{dx}\ \phi\left(\frac{lnx-\mu}{\sigma}\right)$$

In general, the cumulative distribution function of the standard normal distribution *N(0,1) is given by the formula:*

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{\frac{-t^2}{2}}\ dt$$

The related error function is defined as the probability of a random variable with normal distribution of mean 0 and variance 1/2 falling in the range [-x,x]. The error function is given by the formula:

$$erf(x) = \frac{2}{\sqrt{\pi}}\int_{-\infty}^{x} e^{-t^2}\ dt$$

Finally, CDF of log normal distribution is:

$$F(x) = \frac{1}{2}\left[\ 1 + erf\left(\frac{lnx-\mu}{\sigma\sqrt{2}}\right)\right],$$

where parameters μ and σ are respectively, the mean and standard deviation of the variable's x natural logarithm.

In Figure 4 below we can see the comparison between the CDF of the losses captured from the given data set over the four years in both plants and the CDF distribution that was modeled by using the CDF formula for the log normal distribution F(x) as described above.
The CDF distribution of the generated log normal distributed x variables equals the CDF distribution of the losses. The parameters μ and σ in the formula are calculated from the original data set as described in Chapter 1.2 as the mean and the standard deviation of the log values of the variable loss in plantA and plantB.
Therefore, in plant A μA = 9.146 and sigmaA = 1.0615 while in the plantB μB = 6.959 and sigmaB = 1.1333.

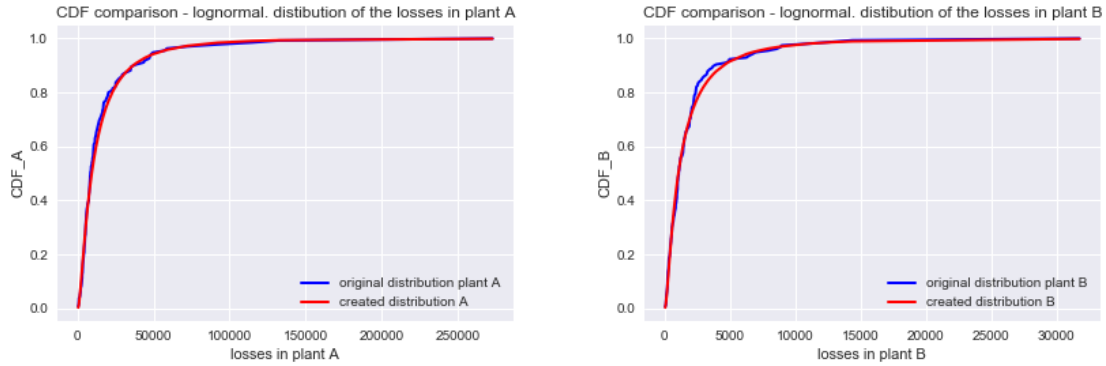

Figure 4 CDF comparisons of log normal distributions of real data vs. model

In the Figure 5 we can see the histogram of the losses for plant A and plant B computed from the original data set:

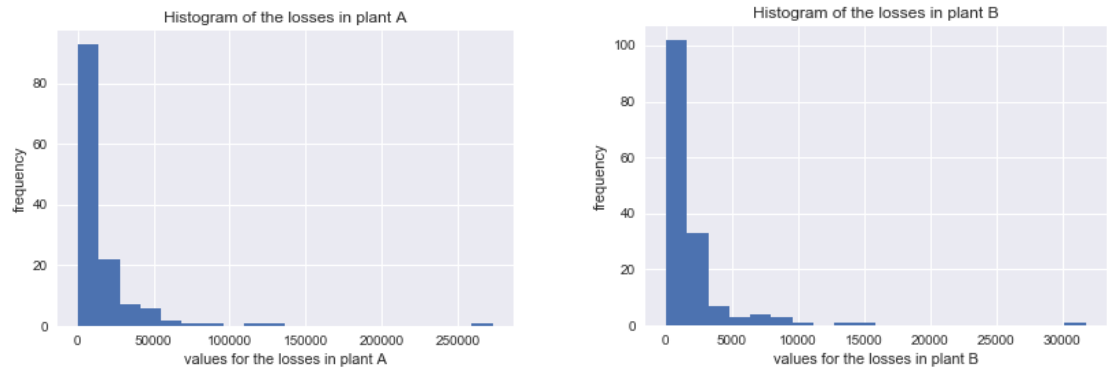

Figure 5 Histogram of real data losses

1.4 Monte Carlo Simulation

1.4.1 Simulate Once

By having determined all the parameters as described in the Chapters 1.2 and 1.3 and by assuming that the time interval between accidents is exponential and that the natural log of a loss due to a single accident is a Gaussian, we can perform the Monte Carlo simulation.
The basic characteristics of Monte Carlo simulations are the two functions.

The first function is the 'simulate once' that uses random variables to model the unknown time intervals and loss occurrences for one year the both plants. I used the pseudo-random number generator for the exponential distribution that simulates the time intervals between the accidents:

- random.expovariate (lamb) where the values for 'lambda' (the average number of accidents per year) in both plants are given in Chapter 1.2

In addition, the random generator for the log normal distribution is used to simulate the losses:

- random.lognormvariate (mu, sigma), where the values for 'mu' and 'sigma' in both plants are explained and calculated in Chapter 1.2

If we take the exponential of the random Gaussian distribution, we can get the same results for the loss simulation as with random.lognormvariate(mu, sigma):

- exp(random.gauss(mu, sigma)

The result of the 'simulate once' is the total loss collected over the period of one year. The total loss collected over the period of one year is also an average yearly loss.

1.4.2 Simulate many and Bootstrap

The second function of the Monte Carlo simulation is the 'simulate many' that repeats 'simulate once' a number of times 'N' until the stop criteria is reached. The result of the 'simulate many' is the average of the all N total yearly losses that were simulated N times until the stopping criteria of 10% relative precision is reached according to the project requirement. The simulate many can also average the results of 'simulate once' until they converge but that usually requires a higher value of N since the relative precision of 10 % can be reached earlier before the real convergence happens. This is described in Chapter 1.5. In other words, the more simulate once is run the closer we get to the true result.

The result of Monte Carlo computation is an average of the all simulated total losses:

- $\mu_{data} = \frac{1}{N} \sum xi$, where xi is the average yearly loss from the simulate once, N is the number of times that simulate once was run in the simulate many and μ_{data} is the average captured from the data.

The error of this average, abs (μ_{data} - μ), that tells how much we can trust the difference between the average captured from the data (μ_{data}) and the real average (μ), is presented by the formula:

- $\sigma_{\mu\ data} = \frac{\sigma}{\sqrt{N}}$ where $\sigma = \sqrt{\frac{1}{N} \sum (xi - \mu\ data)^2}$, where σ is the standard deviation of xi.

This formula always assumes that the average yearly loss (xi) has a Gaussian distribution. Since this is not true in our case, we use the Bootstrap approach to find the error of the average without making an assumption that xi has a Gaussian distribution.
With the Bootstrap approach we find 100 different data samples obtained by resampling the vector of the N average yearly losses which resulted from running 'simulate once' N times. Then the mean of each sample is calculated (μ_{sample}). From there it is not difficult to sort 100 mu

samples into the vector w= [$\mu_{\text{sample 1}}$, $\mu_{\text{sample 2}}$, $\mu_{\text{sample 3}}$ ... $\mu_{\text{sample 100}}$] and to find that the true average (μ) is between μ^- and μ^+ with a 68% of probability or with 68% of confidence level :

- $\mu^- < \mu < \mu+$, where μ^- is the lower confidence limit and μ^+ is the upper confidence limit from the sorted vector w
- the uncertainty is BootStrap error = ($\mu+$ - $\mu-$) / 2

In other words, the true average is bounded by the average captured from the data and the sigma captured from the data with 68% the confidence interval:

- $\mu_{\text{data}} - \sigma_{\mu\text{ data}} < \mu < \mu_{\text{data}} + \sigma_{\mu\text{ data}}$

In this project all the steps described above are achieved by using a generic program MCEngine that is part of the nlib library.

1.5 Results of the simulation

The result of simulation depends on the number of times we run 'simulate once' and the relative precision of 10 %. The relative precision is an indication that 'simulate many' should be stopped once we achieve the following condition:

$\sigma_{\mu\text{ data}} < \mu_{\text{data}} * 0.1$

In other words, it means that we trust the mean value within 10%.

Before I considered relative precision for the stopping criteria I wanted to check how the average yearly loss converges to the true value in the both plants, see Figure 6:

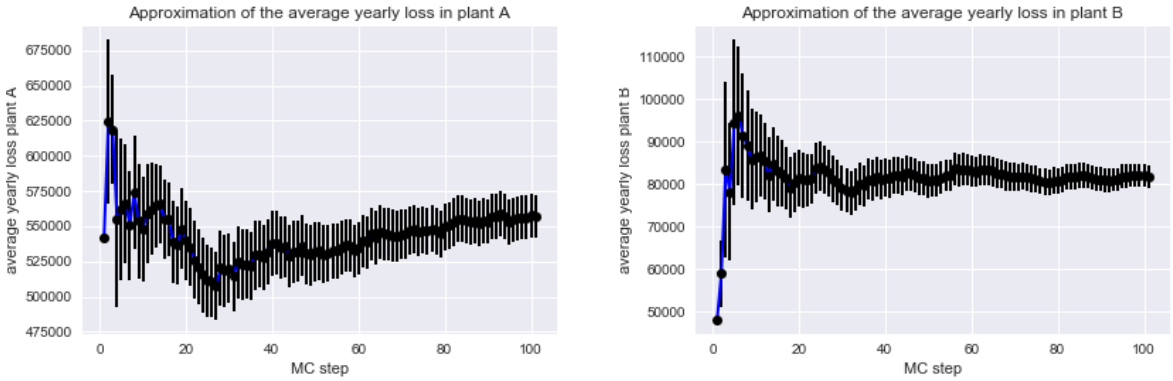

Figure 6 Average yearly loss convergence

We can see that after 100 Monte Carlo steps the average yearly loss in plant A converges to 565000 and in plant B the average yearly loss converges to 80000. Those values are compatible with calculated values for the average loss per year in the Chapter 1.2.

1.5.1 Experiment 1: number of MC steps >5 and relative precision 10%

Now, I wanted to see how many MC steps were required to achieve the relative precision of 10 %. For this purpose, I used the stopping criteria in 'simulate many' MCEngine: number of MC steps >5.

	MC step	μ-	μ+	μ_{data}	$\sigma_{\mu\,data}$	BootStrap Error
Plant A	6	567,279.60	689,161.19	620,806.50	55,156.36	60,940.80
Plant B	9	80,337.72	98,817.26	90,701.49	8,887.00	9,239.77

Table 1 Confidence interval 68% (simulate many: stopping criteria k>5)

According to Table 1 the relative precision of 10% for 68% CI is achieved in plant A after only 6 iterations while in plant B after only 9 Monte Carlo steps. We can also see that Bootstrap error is different than $\sigma_{\mu\,data}$ values in both plant A and plant B, although the difference is not significant. This is expected since $\sigma_{\mu\,data}$ is the error in the mean in for a Gaussian distribution while the Bootstrap error is relevant for any distribution.

	MC step	μ-	μ+	μ_{data}	BootStrap Error	
Plant A	7	468,326.60	672,590.46	543,830.10	102,131.92	
Plant B	14	65,925.60	92,620.46	77,232.67	13,347.43	

Table 2 Confidence interval 95% (simulate many: stopping criteria k>5)

According to Table 2 the relative precision of 10% for 95% CI is achieved in plant A after only 7 iterations while in plant B after 14 iterations. Since the confidence interval is wider with 95% CI in comparison to 68% CI, the Bootstrap error is higher as well. I was expecting to see that μ_{data} was the same in Table 1 and Table 2 but there is an obvious difference. The reason is that number of MC steps is low. According to Figure 6 in the first 20 steps we can expect bigger fluctuations of the average loss values so the mean of the data might fluctuate and be higher at the beginning before it converges.

1.5.2 Experiment 2: number of MC steps >100

Now, I wanted to increase the number of MC steps. For this purpose, I used the stopping criteria in 'simulate many' MCEngine: number of MC steps >100.
If we run 'simulate once' at least 100 times like in Table 3 and Table 4, we can notice that the average value computed from the data (μ_{data}) is very similar for CI 68% and CI 95%. The Bootstrap error is higher for the wider confidence interval. We can also notice that Bootstrap errors are lower in Experiment 2 then the Bootstrap errors in Experiment 1 when the number of MC steps was set to >5. This is a logical consequence since the relative precision is lower when the number of MC steps is higher (rp is below 10%) and with a higher number of MC steps the μ_{data} value is more similar to the true mean value, therefore the error in the mean is smaller. In addition, the values for μ_{data} are lower in Experiment 2 since they converge towards true values while in Experiment 1 they fluctuate a lot in the first 20 MC steps.

	MC step	μ-	μ+	μ_{data}	$\sigma_{\mu\,data}$	BootStrap Error
Plant A	101	538,990.18	573,291.62	556,438.73	16,321.00	17,150.72
Plant B	101	75,272.14	78,890.14	77,306.74	1,966.00	1,809.00

Table 3 Confidence interval 68% (simulate many: stopping criteria k>100)

	MC step	μ-	μ+	μ_{data}	BootStrap Error	
Plant A	101	524,922.55	608,321.23	559,751.47	41,699.34	
Plant B	101	72,109.81	79,360.00	76,146.00	3,625.04	

Table 4 Confidence interval 95% (simulate many: stopping criteria k>100)

1.5.3 Budget Planning

One of the answers that the simulation provides is how to plan the budget to make sure that company can cover the losses in 90% of the simulated scenarios.
If we run 'simulate once' at least 5 times (Experiment 1) in order to achieve the relative precision of 10% and repeat that scenario ('simulate many') 100 times we are going to get a vector of 100 average yearly losses scenarios according to Figure 7. In the sorted vector the value with the index of 90 indicates that all values from the previous 90 scenarios (index from 0 - 89) are lower. For plant A the company should reserve a budget of $ 620,757.289 to cover losses in 90% of scenarios with a relative precision of 10 % while for the plant B the planned budget should be $86,268.75.

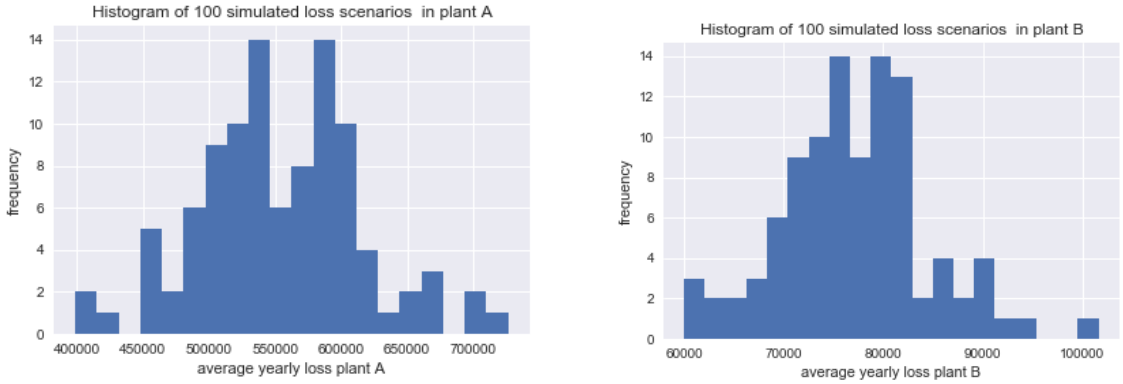

Figure 7 Histogram of 100 simulated loss scenarios when 'simulate once' is run at least 5 times

If we run 'simulate once' at least 100 times (Experiment 2) and repeat that scenario 100 times according to Figure 8, the values at index 90 for the plant A and plant B are lower than in the previous example when the simulate once was run as in Experiment 1. For plant A the company should reserve a budget of $572,152.36 to cover the losses in 90% of scenarios while for plant B the planned budget should be $ 81,004.25.
Since the number of MC steps is low in Experiment 1, average yearly loss values fluctuate significantly in the first 20 steps. Therefore, the planned budget with relative precision of 10% is higher in comparison to the scenario when the number of MC steps is greater than 100.

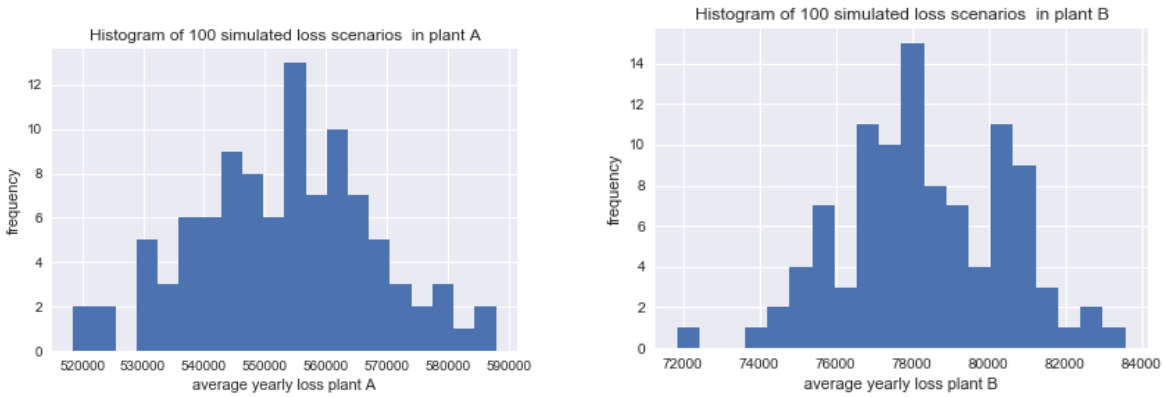

Figure 8 Histogram of 100 simulated loss scenarios when 'simulate once' is run at least 100 times

In Figure 9 we can see the simulated losses over a period of 1 year (losses vs time). There is a clear similarity with Figure 1 where real losses are presented over the period of 4 years. Like in Figure 1, the simulated loses in plant A are mostly in the range of up to $10,000.00 although some values are higher. For plant B most of the simulated losses are in the range of up to $2,500.00. Figure 10 shows accumulated losses over the period of one year.

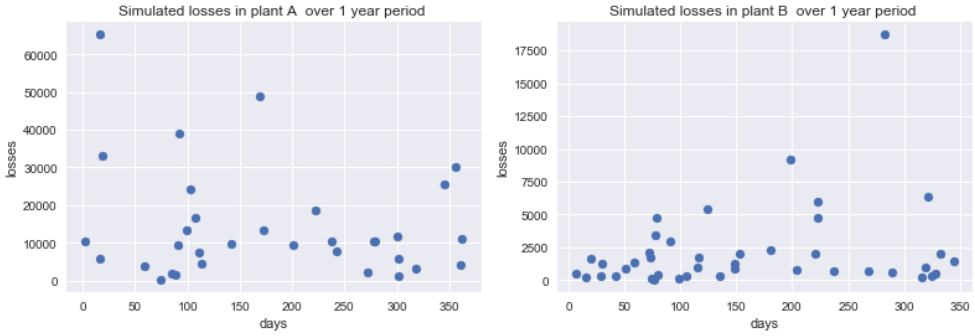

Figure 9 Simulated losses over 1-year period

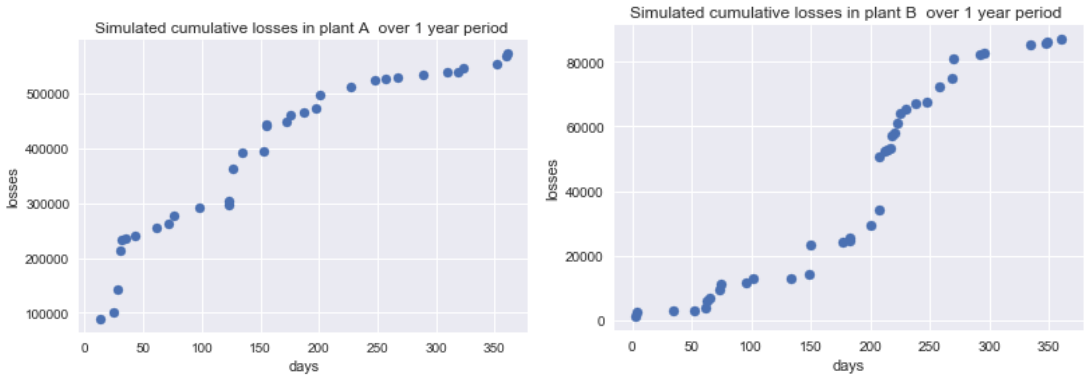

Figure 10 Accumulative simulated losses over 1-year period

For detailed calculation of the budget planning please see Chapter 1.7 Appendix.

1.6 Conclusion

In this paper the analysis shows that simulated results are closer to the true values by running the Monte Carlo simulation more times (Experiment 2). The higher values for the relative precision (rp = 10%) usually stops the simulation in an early phase so the Bootstrap error is higher (Experiment1). At the same time the relative precision of 10% implies that company should reserve more money in order to cover the losses in 90% scenarios in comparison to scenario where the relative precision is smaller but the simulation is run more times. I think that a 10% relative precision gives the company the opportunity to plan the budget with a certain reserve, which is a good approach. It was also interesting to see that simulated data captured the distribution of the real data since the CDF functions were almost identical. In addition, the range of simulated losses values is comparable with the real data set.

1.7 Appendix: Python Code

reading the file and parameters calculation according to Chapter 1.2

```
from nlib import Canvas, mean, sd, MCEngine
import csv
from math import exp, log, sqrt, erf
import random

import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)

def read_csv(filename = 'accidents.csv'):
    '''this function reads the accidents file and returns
    the vector of losses, the vector of days when the accident happened,
    and the vector of time intervals between the accidents per each plant over
    the 4 years' period'''
    dayA=[]
    dayB=[]
    lossA=[]
    lossB=[]
    time_intervals_A=[]
    time_intervals_B=[]
    previousA = 0.0
    previousB = 0.0
    with open (filename) as myfile:
        reader = csv.reader(myfile)
        for row in reader:
            if row[0] == 'A':
                dayA.append(row[1])
                lossA.append(row[2])
                dayA = map(float, dayA)
                lossA = map(float, lossA)
                time_intervals_A.append(float(row[1])-previousA)
                previousA = float(row[1])
            elif row[0] =='B':
                dayB.append(row[1])
                lossB.append(row[2])
                dayB = map(float, dayB)
                lossB = map(float, lossB)
                time_intervals_B.append(float(row[1])-previousB)
                previousB = float(row[1])
    return dayA, lossA, dayB, lossB, time_intervals_A, time_intervals_B

dayA, lossA, dayB, lossB, time_intervals_A, time_intervals_B = read_csv(filename =
'accidents.csv')
```

Chapter 1.2

```
def avg_loss(day,loss):
    '''this function calculates the average loss per year'''
    avg_loss_year = sum(loss)/4
    return avg_loss_year


def avg_accidents(day,loss):
    '''this function calculates the average number of accidents per year'''
    avg_accidents_year = float(len(day))/4
    return avg_accidents_year

def avg_loss_accident(day, loss):
    '''this function calculates the average loss per accident '''
    avg_loss_per_accident = sum(loss) / len(day)
    return avg_loss_per_accident


avg_lossA_year = avg_loss(dayA, lossA)
avg_lossB_year = avg_loss(dayB, lossB)
avg_accidentsA_year = avg_accidents(dayA,lossA)
avg_accidentsB_year = avg_accidents(dayB,lossB)
avg_lossA_per_accident = avg_loss_accident(dayA, lossA)
avg_lossB_per_accident = avg_loss_accident(dayB, lossB)


def get_parameters(loss):
    '''this function returns the mean and the sigma of log loss'''
    loss_log = []
    for value in loss:
        value1 = log(value)
        loss_log.append(value1)
    mu = mean(loss_log)
    sigma = sd(loss_log)
    return loss_log, mu, sigma

lossA_log, muA, sigmaA = get_parameters(lossA)
lossB_log, muB, sigmaB = get_parameters(lossB)


lambA = avg_accidentsA_year
lambB = avg_accidentsB_year
```

#Data visualization

```
# loss histogram of the original data set
Canvas(title='Histogram of the losses in plant A', xlab='values for the losses in plant A',
ylab='frequency').hist(lossA).save('lossA_hist.png')
Canvas(title='Histogram of the losses in plant B', xlab='values for the losses in plant B',
ylab='frequency').hist(lossB).save('lossB_hist.png')


# time intervals histogram of the original data set
Canvas(title='Histogram of the time intervals in plant A',xlab='time intervals in plant
A',ylab='frequency').hist(time_intervals_A).save('time_intervals_A.png')
Canvas(title='Histogram of the time intervals in plant B',xlab='time intervals in plant
B',ylab='frequency').hist(time_intervals_B).save('time_intervals_B.png')


# Scatter Real data losses in plant A/B over the 4 years period (loss vs time)
points_T_L_A = [(dayA[i],lossA[i]) for i in range(len(lossA))]
points_TA = [x[0] for x in points_T_L_A]
points_LA = [x[1] for x in points_T_L_A]
plt.scatter(points_TA, points_LA)
plt.title("Real data losses in plant A over the 4 years period ")
plt.xlabel("days")
plt.ylabel("losses")
plt.show()
```

```
points_T_L_B = [(dayB[i],lossB[i]) for i in range(len(lossB))]
points_TB = [x[0] for x in points_T_L_B]
points_LB = [x[1] for x in points_T_L_B]
plt.scatter(points_TB, points_LB)
plt.title("Real data losses in plant B over the 4 years period ")
plt.xlabel("days")
plt.ylabel("losses")


def simulate_once(lamb, mu, sigma): # simulate one day
'''this function returns accumulated simulated losses, simulated losses
    and simulated time/days'''
    t = 0.0 # start
    accum_loss_sim=[]
    accum_loss = 0.0
    time_sim = []
    loss_sim = []
    while True: # t is 1 year
        delta = random.expovariate(lamb) # time interval between accidents
        t = t+delta
        if t>1: break
        loss = random.lognormvariate(mu,sigma)
        accum_loss = loss+accum_loss
        loss_sim.append(loss)
        time_sim.append(t*365)
        accum_loss_sim.append(accum_loss)
    return accum_loss_sim, loss_sim, time_sim

accum_lossA_sim, lossA_sim, timeA_sim = simulate_once(lambA, muA, sigmaA)
accum_lossB_sim, lossB_sim, timeB_sim = simulate_once(lambB, muB, sigmaB)

# Scatter Simulated losses in plant A/B  over 1 year period (loss vs time)
points_T_L_A_sim = [(timeA_sim[i],lossA_sim[i]) for i in range(len(lossA_sim))] # time, loss
points_TA_sim = [x[0] for x in points_T_L_A_sim] # time A
points_LA_sim = [x[1] for x in points_T_L_A_sim] # loss A
points_T_L_B_sim = [(timeB_sim[i],lossB_sim[i]) for i in range(len(lossB_sim))] # time, loss
points_TB_sim = [x[0] for x in points_T_L_B_sim] # time B
points_LB_sim = [x[1] for x in points_T_L_B_sim] # loss B

plt.scatter(points_TA_sim, points_LA_sim)
plt.title("Simulated losses in plant A  over 1 year period ")
plt.xlabel("days")
plt.ylabel("losses")
plt.show()

plt.scatter(points_TB_sim, points_LB_sim)
plt.title("Simulated losses in plant B  over 1 year period ")
plt.xlabel("days")
plt.ylabel("losses")
plt.show()

# Scatter Simulated cumulative losses in plant A/B  over 1 year period (cumulative loss vs time)

points_T_L_A_sim_accum = [(timeA_sim[i],accum_lossA_sim[i]) for i in range(len(accum_lossA_sim))]
# time, loss
points_TA_sim_accum = [x[0] for x in points_T_L_A_sim_accum] # time A
points_LA_sim_accum = [x[1] for x in points_T_L_A_sim_accum] # loss A
points_T_L_B_sim_accum = [(timeB_sim[i],accum_lossB_sim[i]) for i in range(len(accum_lossB_sim))]
# time, loss
points_TB_sim_accum = [x[0] for x in points_T_L_B_sim_accum] # time B
points_LB_sim_accum = [x[1] for x in points_T_L_B_sim_accum] # loss B
plt.scatter(points_TA_sim_accum, points_LA_sim_accum)
plt.title("Simulated cumulative losses in plant A  over 1 year period ")
plt.xlabel("days")
plt.ylabel("losses")
plt.show()

plt.scatter(points_TB_sim_accum, points_LB_sim_accum)
plt.title("Simulated cumulative losses in plant B  over 1 year period ")
plt.xlabel("days")
```

```
plt.ylabel("losses")
plt.show()
```

#Chapter 1.3

```
def F(v):
    v.sort()
    n = len(v)
    data = []
    for k in range(n):
        point = (v[k], float(k+1)/n)
        data.append(point)
    return data

def F_exponential(x, lamb):
    return 1.0 - (exp(-float(lamb/365*x)))

def F_lognormal(x, mu, sigma):
    return 0.5 + 0.5*erf((log(x) - mu)/(sigma*sqrt(2.0)))


# CDF time intervals: the original data vs model plant A

points_time_A = F(time_intervals_A) # original distribution based on the given data set
points_time_A_sim = [(x,F_exponential(x, lambA)) for (x,y) in points_time_A] # this is what we
have created
Canvas(title='CDF comparison - exp. distibution of the time intervals in plant A',xlab='time
intervals in plant A',ylab='CDF_A').plot(points_time_A, color ='blue', legend = 'original
distribution plant A').plot(points_time_A_sim, color='red', legend = 'created distribution
A').save('time_intervals_comparison_A.png')


# CDF time intervals: original data vs model plant B

points_time_B = F(time_intervals_B) # original distribution based on the given data set
points_time_B_sim = [(x,F_exponential(x, lambB)) for (x,y) in points_time_B] # this is waht we
have created
Canvas(title='CDF comparison - exp. distibution of the time intervals in plant B',xlab='time
intervals in plant B',ylab='CDF_B').plot(points_time_B, color ='blue', legend = 'original
distribution plant B').plot(points_time_B_sim, color='red', legend = 'created distribution
B').save('time_intervals_comparison_B.png')


# CDF loss: original data vs modeled plant A

points_loss_A = F(lossA) # original distribution based on the given data set
points_loss_A_sim = [(x,F_lognormal(x, muA, sigmaA)) for (x,y) in points_loss_A] # this is waht
we have created
Canvas(title='CDF comparison - lognormal. distibution of the losses in plant A',xlab='losses in
plant A',ylab='CDF_A').plot(points_loss_A, color ='blue', legend = 'original distribution plant
A').plot(points_loss_A_sim, color='red', legend = 'created distribution
A').save('loss_comparison_A.png')

# CDF loss: original data vs modeled plant B

points_loss_B = F(lossB) # original distribution based on the given data set
points_loss_B_sim = [(x,F_lognormal(x, muB, sigmaB)) for (x,y) in points_loss_B] # this is waht
we have created
Canvas(title='CDF comparison - lognormal. distibution of the losses in plant B',xlab='losses in
plant B',ylab='CDF_B').plot(points_loss_B, color ='blue', legend = 'original distribution plant
B').plot(points_loss_B_sim, color='red', legend = 'created distribution
B').save('loss_comparison_B.png')
```

#Monte Carlo simulation Chapter 1.5

```
class ServerSimulator(MCEngine):
    def __init__(self, lamb, mu, sigma):
        self.lamb = lamb
```

```
        self.mu = mu
        self.sigma = sigma
    def simulate_once(self): # we give our own simulate once, simulate_many is the same
        lamb = self.lamb
        mu = self.mu
        sigma = self.sigma
        total_loss = 0.0
        max_time = 1 # we simulate 1 year
        t = 0.0 # start of simulation
        total_loss = 0.0
        while True: # t is 1 year
            delta = random.expovariate(lamb) # time interval between accidents
            t = t+delta
            if t>max_time: break
            loss = random.lognormvariate(mu,sigma)
            total_loss = loss+total_loss
        return total_loss

simA = ServerSimulator(lambA, muA, sigmaA)
simB = ServerSimulator(lambB, muB, sigmaB)

muA_minus, mu_A, muA_plus = simA.simulate_many(ap =0, rp=0.1, ns = 1000) # rp = 10%
muB_minus, mu_B, muB_plus = simB.simulate_many(ap = 0, rp=0.1, ns =1000)
dmuA = (muA_plus-muA_minus)/2 # this tells how much I can trust mu, upper estimate-lower estimate
dmuB = (muB_plus-muB_minus)/2

print mu_A, dmuA
print mu_B, dmuB

historyA = simA.history[:1000]
historyB = simB.history[:1000]

Canvas(title='Approximation of the average yearly loss in plant A', xlab='MC step', ylab='average
yearly loss plant A').plot(historyA).errorbar(historyA).save('lossA_avg_convergence.png')
Canvas(title='Approximation of the average yearly loss in plant B', xlab='MC step', ylab='average
yearly loss plant B').plot(historyB).errorbar(historyB).save('lossB_avg_convergence.png')
```

#Planning company's budget Chapter 1.5

```
vA = []
vB = []
for i in range(100):
    muA_minus, mu_A, muA_plus = simA.simulate_many(ap =0, rp=0.1, ns = 1000)
    muB_minus, mu_B, muB_plus = simB.simulate_many(ap =0, rp=0.1, ns = 1000)
    vA.append(mu_A)
    vB.append(mu_B)
vA.sort()
vB.sort()
print vA, vA[90]
print vB, vB[90]

Canvas(title='Histogram of 100 simulated loss scenarios  in plant A', xlab='average yearly loss
plant A', ylab='frequency') .hist(vA).save('vA_100scenarios.png')
Canvas(title='Histogram of 100 simulated loss scenarios  in plant B', xlab='average yearly loss
plant B', ylab='frequency') .hist(vB).save('vB_100scenarios.png')
```