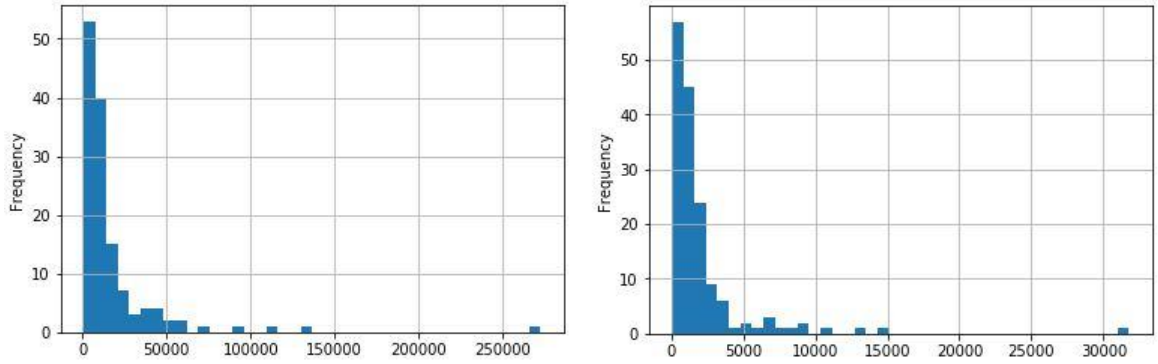
CSC521
Tzu-Hao Peng

Final
Problem 1

In problem 1, the data shows the recorded accidents of plant A and plant B over the last 4 years. There are 291 instances with 3 variables. First column indicates where it happened (plant A or plant B), second column indicates the day (0 is 4 years ago), and the third column indicates the loss caused by the accidents in dollars.

- **Without simulation**



First I am going to see the distribution on losses of each plant, above on the left is the histogram on losses of plant A, and on the right is the histogram on losses of plant B. From the histogram above, we could see that the distribution of both plants is right skewed. And most of the losses are between 0 to 50000 for plant A and 0 to 5000 for plant B

The first question is that we are going to calculate the average number of accidents per year in plant A and plant B. From the excel file, first I sorted plant A and plant B so the output would be clear to interpret. Next, I can compute the count of the accidents for plant A and plant B each year, the result is below:

	Year 1	Year 2	Year 3	Year 4
Plant A	37	30	36	32
Plant B	47	34	40	35

- What is the average number of accidents per year in plant A and plant B?

  The average number of accidents per year in plant A = (37+30+36+32)/4 = 135/4 = 33.75

  The average number of accidents per year in plant B = (41+34+40+35)/4 = 156/4 = 39

- What is the average loss per accident in plant A and plant B?

  Using excel, I can get the average loss per accident in plant A and plant B

  The average loss per accident in Plant A = 2358471/135 = 17470.16

  The average loss per accident in Plant B = 315582/156 = 2022.96

- What is the average loss in total per year in plan A and plant B?

  The average loss in total per year is to use the sum of the loss divided by the year,

  The average loss in total per year in Plant A = Avg loss per accident*Avg number of accidents per year = (2358471/135)*33.75 = 589617.75

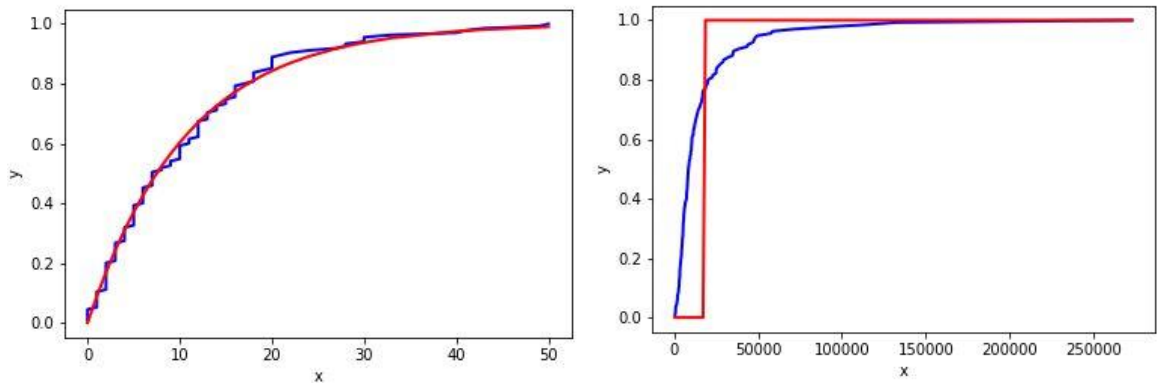
  The average loss in total per year in Plant B = Avg loss per accident*Avg number of accidents per year = (315582/156)*39 = 78895.5

- **Simulate Once**

Now assume the time interval between accidents is exponential and the natural log of a loss due to a single accident is a Gaussian (aka the loss is lognormal) implement a simulate once that simulates one year of losses for both plants.

- **Plant A**

Because we have to simulate one year of losses for both plants, so we am going to do the simulation separately. First, for plant A, I made the exponential plot of day and the Gaussian plot of losses to see the distribution.



The plot on the left is the exponential plot of day and the plot on the right is the Gaussian plot of losses. The red line indicates the exponential distribution and the Gaussian distribution, and the blue line indicates the actual data of plant A. From the plots above, we could see that our data is followed by the distribution. After seeing the distribution of plant A, we could make the simulation.

Simulate once is to make the simulation only one time to see the one year of losses. To implement that, first we have to calculate the mean (mu), minimum (xM), and the alpha (mu/(mu-xM)). After doing it by python, mu = 17470.16, xM = 490.00, and alpha = 1.03, we could use it to do the sumulate_once function. I generated the random number of day followed by the exponential distribution and generated the random number of losses followed by the Gaussian distribution. And set the day < 365 because we only need to calculate one year of losses.

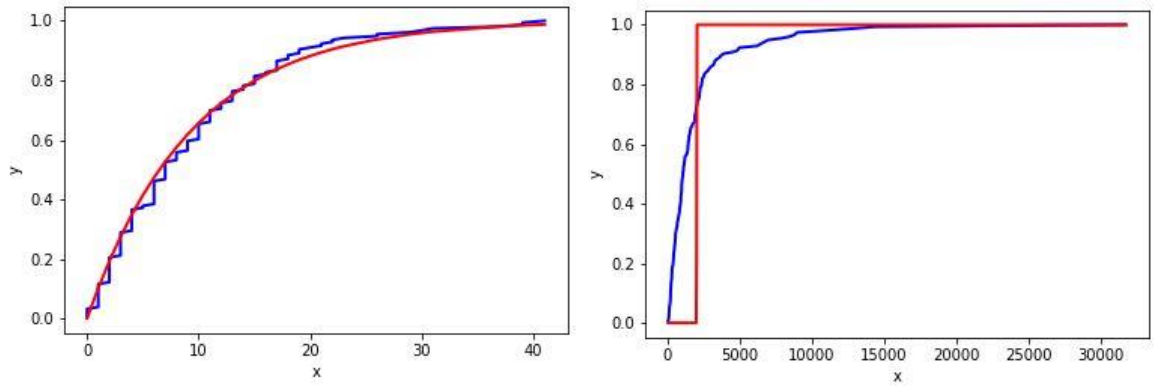
```
One year of losses for plant A = 444873.272218
```

As a result, one year of losses for plant A = 444873.27 dollars, and the result fluctuated because we were using the random numbers. The full results breakdown for simulate once of plant A can be found in the Appendix - [28].

- **Plant B**

For plant B, the exponential plot of day on the left and the Gaussian plot of losses on the

right shows that the data of plant B is followed by the exponential distribution and the Gaussian distribution. We could make the simulation.



After calculated plant B by python, we could know that mean (mu) = 2022.96, minimum (xM) = 46.00, and alpha (mu/(mu-xM)) = 1.02. To implemented it into the simulate_once function. First, I generated the random number of day followed by the exponential distribution and generated the random number of losses followed by the Gaussian distribution. And again, set the day < 365 because we only need to calculate one year of losses.

```
One year of losses for plant B = 73105.1447163
```

As a result, one year of losses for plant B = 73105.14 dollars. Again, it fluctuated because of the random numbers. The full results breakdown for simulate once of plant B can be found in the Appendix - [247].

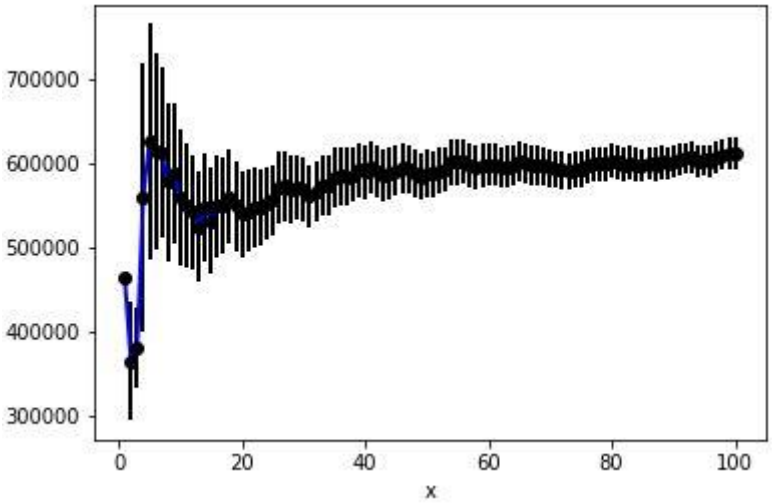
- **Simulate Many**

Running simulate many, what is the average yearly loss with a relative precision of 10%? Report the bootstrap errors in your result.

- **Plant A**

Simulate Many is to simulate once repeatedly, computes average and error analysis, checks convergence, and computes bootstrap error for the result. Since we've already done simulate once previously, we have to compute the average from the result of every simulate

once. I've set the simulation into 10000 times, so the program would run the simulation for 10000 years and compute the average yearly loss with a relative precision of 10%. The error bar plot of loss convergence is below:



I've only plot 100 years so we could see it clearly. From the plot, we could see the yearly loss convergence with a relative precision of 10% for 100 years. Every dots indicates one year of losses for plant A. The trend is rising very fast in the beginning but become stable after 40 years. We could assume that the more simulation you make, the more precisely the result will be. After doing the computation, the average loss with a relative precision of 10% for plant A is 617179.60 dollars. The plot is fluctuated because of the random numbers, but the trend is all becoming stable after doing more simulations. The full results breakdown for simulate many of plant A can be found in the Appendix - [112].

- **Bootstrap of plant A**

The purpose of the bootstrap algorithm is computing the error in an average $\mu = (1/N) \sum xi$ without making the assumption that the xi are Gaussian. As long as we know the mean and the standard deviation of plant A, we could compute the bootstrap error. The mean for plant A is 17470.16 with the 29862.67 standard deviation. After doing the computation, the result is below:

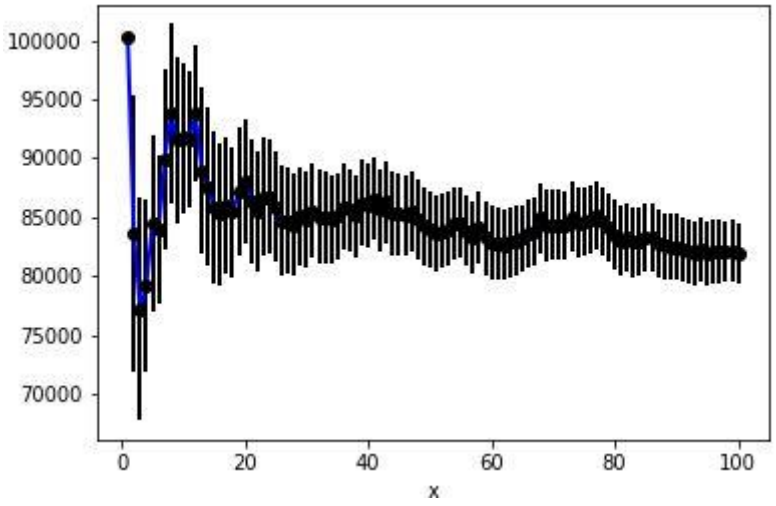
```
confidence interval(at 50%) is 17713.447348, 17947.522282, 18170.989826
confidence interval(at 68%) is 17636.792182, 17947.522282, 18222.454465
confidence interval(at 80%) is 17536.078501, 17947.522282, 18373.955658
confidence interval(at 90%) is 17366.435052, 17947.522282, 18368.267219
```

I've done bootstrap method for the confidence interval at 50%, 68%, 80%, and 90% and did the simulation for 10000 years. Because we had 10000 random Gaussian numbers, with average 17470.16 and standard deviation 29862.67, we expect μ to be close to 17470.16. We get 17947.52. The bootstrap tells us that with 68% probability, the true average of these numbers is indeed between 17636.79 and 18222.45. The uncertainty (18222.45 – 17636.79)/2 = 292.83 is compatible with 29862.67/ √ 10000 = 29862.67/100 = 298.63. The full results breakdown for the bootstrap of plant A can be found in the Appendix - [200].

- Confidence interval (at 50%): The uncertainty (18170.99 – 17713.45)/2 = 228.77 is compatible with 29862.67/ √ 10000 = 29862.67/100 = 298.63.
- Confidence interval (at 68%): The uncertainty (18222.45 – 17636.79)/2 = 292.83 is compatible with 29862.67/ √ 10000 = 29862.67/100 = 298.63.
- Confidence interval (at 80%): The uncertainty (18373.96 – 17536.08)/2 = 418.94 is compatible with 29862.67/ √ 10000 = 29862.67/100 = 298.63.
- Confidence interval (at 90%): The uncertainty (18368.27 – 17366.44)/2 = 500.92 is compatible with 29862.67/ √ 10000 = 29862.67/100 = 298.63.

- **Plant B**

For plant B, again, simulated for 10000 times. Tried to see the loss convergence of plant B for 10000 years. The plot is below:



I've only plot for 100 years so the visualization would be clear. The plot is pretty similar with plant A, it is rising rapidly in the beginning but become stable after doing more simulations. As a result, the average loss with a relative precision of 10% for plant B is 80680.16 dollars. Also, the plot is fluctuated due to the random numbers, but it will become stable after doing more simulations. The full results breakdown for simulate many of plant B can be found in the Appendix - [322].

- **Bootstrap of plant B**

For plant B, the mean is 2022.96 and the standard deviation is 3336.62. I also did the simulation for 10000 years. The confidence interval at 50%, 68%, 80%, and 90% is below:

```
confidence interval(at 50%) is 2050.031293, 2073.630775, 2090.889093
confidence interval(at 68%) is 2039.699405, 2073.630775, 2109.831806
confidence interval(at 80%) is 2037.357105, 2073.630775, 2118.037323
confidence interval(at 90%) is 2023.702626, 2073.630775, 2151.580622
```

The bootstrap tells us that with 68% probability, the true average of these numbers is indeed between 2039.70 and 2109.83. The uncertainty (2109.83 – 2039.70)/2 = 35.065 is compatible with 3336.62/ √ 10000 = 3336.62/100 = 33.37. The full results breakdown for bootstrap of plant B can be found in the Appendix - [410].

- Confidence interval (at 50%): The uncertainty (2090.89 – 2050.03)/2 = 20.43 is compatible

with 3336.62/ √ 10000 = 3336.62/100 = 33.37.

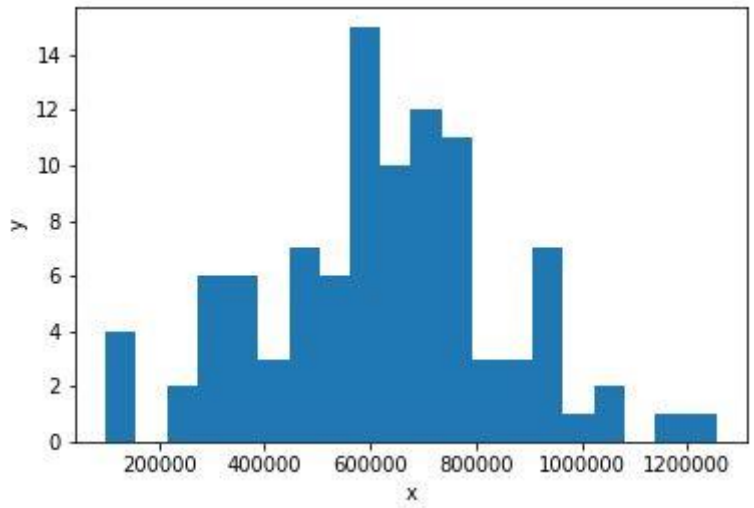
- Confidence interval (at 68%): The uncertainty (2109.83 – 2039.70)/2 = 35.065 is compatible with 3336.62/ √ 10000 = 3336.62/100 = 33.37.
- Confidence interval (at 80%): The uncertainty (2118.04 – 2037.36)/2 = 40.34 is compatible with 3336.62/ √ 10000 = 3336.62/100 = 33.37.
- Confidence interval (at 90%): The uncertainty (2151.58 – 2023.70)/2 = 63.94 is compatible with 3336.62/ √ 10000 = 3336.62/100 = 33.37.

- **90% of the simulated scenarios**

How much should the company budget to make sure that it can cover these losses in 90% of the simulated scenarios?

➢ **Plant A**

Below is the histogram for 100% of the scenario for one year of loss for plant A:



From the histogram above, we could see that the distribution is pretty much normally distributed. It is followed by the Gaussian distribution.

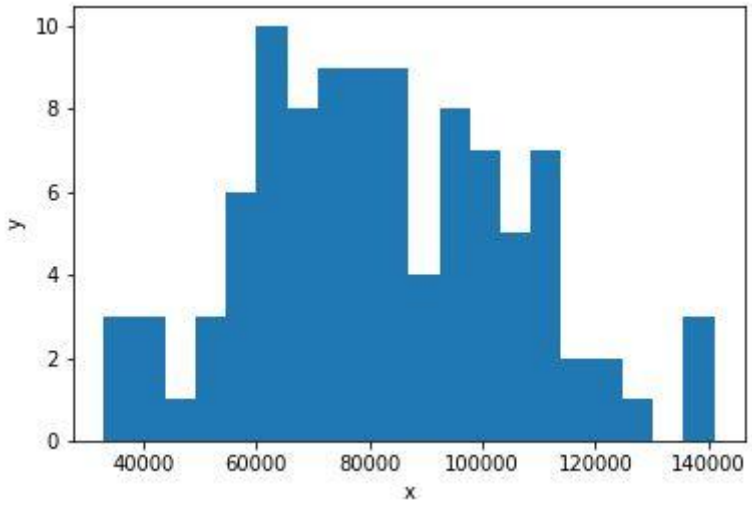
```
90% scenarios = 938504.316699
```

To cover the losses in 90% of the simulated scenarios, the company budget should be

938504.32 dollars. The full results breakdown for the 90% of the simulated scenarios of plant A can be found in the Appendix - [92].

- **Plant B**

Below is the histogram for 100% of the scenario for one year of loss for plant B:



From the histogram above, we could see that the distribution is also normally distributed, it is followed by the Gaussian distribution.

```
90% scenarios = 113341.637298
```

To cover the losses in 90% of the simulated scenarios, the company budget should be 113341.64 dollars. The full results breakdown for the 90% of the simulated scenarios of plant B can be found in the Appendix - [302].

- **Conclusion**

In conclusion, we've done without simulation, simulate once, and simulate many algorithms to compute the losses for both plant A and plant B. Below is the summary of what we've done so far for both plants within these methods,

	Plant A	Plant B
Without simulation	589617.75	78895.5
Simulate once for one year	444873.27	73105.14
Simulate many for the yearly average losses	607179.6	80680.16
90% scenarios	938504.32	113341.64

If we only do the calculation without any simulation, we are calculating the actual data from the csv file. However, if we are doing the simulation using this dataset, we are going to generate the random numbers using the basic statistics such as mean, standard deviation…etc from the dataset for plant A and plant B. And the result is above, the result of simulate once may not be very correct because the algorithm only simulate one time for one year of loss. Simulate many would be more precisely because it do the simulation many times, I would say the more simulation you do, the more accurate of the results you have. In this problem, I did simulate many for 10000 times, and the result is pretty close to the result without simulation. And the company budget should be about 900 thousand dollars for plant A and 100 thousand dollars for plant B to make sure it can cover these losses in 90% of the simulated scenarios.

Appendix

```
1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. %matplotlib inline
5. import random
6. from nlib import *
7. import csv
8. from math import *
9.
10. problem1 = pd.read_csv("accidents.csv", header = None)
11. problem1
12.
13. problem1.columns=['X','Y','Z']
14. problem1
15.
16. problem1.sort_values(by=['X'])
17.
18. problem1['Z'][problem1['X']=='A'].plot(kind = "hist", bins = 40, grid = True)
19. problem1['Z'][problem1['X']=='B'].plot(kind = "hist", bins = 40, grid = True)
20.
21. problem1.groupby('X').count()
22.
23. avg_accdA = float(135/4)
24. avg_accdA
25. avg_accdB = float(156/4)
26. avg_accdB
27.
28. #Plant A
29.
30. def F(v):
31.         v.sort()
32.         n = len(v)
33.         data = []
34.         for k in range(n):
35.                 point = (v[k], float(k+1)/n)
36.                 data.append(point)
```

```
37.         return data
38.
39. def read_csv(filename='accidentsA.csv'):
40.         day = []
41.         loss = []
42.         with open(filename) as myfile:
43.                 reader = csv.reader(myfile)
44.                 previous = 0.0
45.                 for row in reader:
46.                         row = map(float, row)
47.                         day.append(row[0] - previous)
48.                         loss.append(row[1])
49.                         previous = row[0]
50.         return day, loss
51.
52. def get_parameters():
53.         day, loss = read_csv()
54.         lamb = float(len(day))/1462
55.
56.         def F_exponential(x, lamb=lamb):
57.                 return 1.0-exp(-lamb*x)
58.
59.         points = F(day)
60.         points2 = [(x,F_exponential(x)) for (x,y) in points]
61.
62. #Canvas().plot(F(day)).save('F_dayA1.png')
63.         Canvas().plot(points,color='blue').plot(points2,color='red').save('F_dayA2.png')
64. #Canvas().plot(F(loss)).save('F_lossA.png')
65.
66.         mu = sum(loss)/len(loss)
67.         xM = min(loss)
68.         alpha = mu/(mu-xM)
69.         print 'mu=%f, xM=%f, alpha=%f' % (mu, xM, alpha)
70.
71.         def F_gauss(x, xM = xM, alpha=alpha):
72.                 return 0.5+0.5*erf((x-mu)/(alpha*(2.0**0.5)))
73.
74.         points = F(loss)
```

```
75.         points2 = [(x,F_gauss(x)) for (x,y) in points]
76.         Canvas().plot(points,color='blue').plot(points2,color='red').save('F_lossA1.png')
77.
78.         return lamb, xM, alpha
79.
80. lamb, xM, alpha = get_parameters()
81.
82. #Simulate once
83.
84. t = 0
85. total_loss = 0
86. while t<365:
87.         t = t + random.expovariate(lamb)
88.         amount = random.gauss(17470.155556, 29862.673153)
89.         total_loss = total_loss+amount
90. print 'One year of losses for plant A =', total_loss
91.
92. #90% scenarios
93.
94. def simulate_once():
95.         t = 0
96.         total_loss = 0
97.         while t<365:
98.                 t = t + random.expovariate(lamb)
99.                 amount = random.gauss(17470.155556, 29862.673153)
100.                  total_loss = total_loss+amount
101.         return total_loss
102.
103. scenarios = []
104. for k in range(100):
105.         y = simulate_once()
106.         scenarios.append(y)
107.
108. Canvas().hist(scenarios).save('scenariosA.png')
109. scenarios.sort()
110. print '90% scenarios =', scenarios[int(100*0.9)]
111.
112. #Simulate many
```

```
113.
114. def read_csv(filename='accidentsA.csv'):
115.     day = []
116.     loss = []
117.     with open(filename) as myfile:
118.         reader = csv.reader(myfile)
119.         previous = 0.0
120.         for row in reader:
121.             row = map(float, row)
122.             day.append(row[0] - previous)
123.             loss.append(row[1])
124.             previous = row[0]
125.     return day, loss
126.
127. def get_parameters():
128.     day, loss = read_csv()
129.     lamb = float(len(day))/1460
130.
131.     def F_exponential(x, lamb=lamb):
132.         return 1.0-exp(-lamb*x)
133.
134.     points = F(day)
135.     points2 = [(x,F_exponential(x)) for (x,y) in points]
136.
137. #Canvas().plot(F(day)).save('F_dayA1.png')
138.     Canvas().plot(points,color='blue').plot(points2,color='red').save('F_dayA2.png')
139. #Canvas().plot(F(loss)).save('F_lossA.png')
140.
141.     mu = sum(loss)/len(loss)
142.     xM = min(loss)
143.     alpha = mu/(mu-xM)
144.     print 'mu=%f, xM=%f, alpha=%f' % (mu, xM, alpha)
145.
146.     def F_gauss(x, xM = xM, alpha=alpha):
147.         return 0.5+0.5*erf((x-mu)/(alpha*(2.0**0.5)))
148.
149.     points = F(loss)
150.     points2 = [(x,F_gauss(x)) for (x,y) in points]
```

```
151.        Canvas().plot(points,color='blue').plot(points2,color='red').save('F_lossA1.png')
152.
153.        return lamb, xM, alpha
154.
155. lamb, xM, alpha = get_parameters()
156.
157. class Simulator(object):
158.
159.        def _init_(self):
160.                pass
161.
162.        def simulate_once(self):
163.                t = 0
164.                total_loss = 0
165.                while t<365:
166.                        t = t + random.expovariate(lamb)
167.                        amount = random.gauss(17470.155556, 29862.673153)
168.                        total_loss = total_loss+amount
169.                return total_loss
170.
171.        def simulate_many(self, ap=100, ns=100):
172.                sy=0.0
173.                sy2=0.0
174.                self.history = []
175.                for k in range(1,ns+1):
176.                        print k
177.                        y = self.simulate_once()
178.                        sy = sy + y
179.                        sy2 = sy2 + y*y
180.                        y_bar = sy/k
181.                        y2_bar = sy2/k
182.                        sigma = (y2_bar - y_bar**2)**0.5
183.                        dy_bar = sigma/k**0.5
184.                        self.history.append((k,y_bar,dy_bar))
185.                        if dy_bar<ap and k>100:
186.                                return y_bar
187.                return y_bar
188.                #raise ArithmeticError
```

```
189.
190. sim = Simulator()
191. print sim.simulate_many(ap=0.1, ns=10000)
192. history = sim.history[:100]
193. Canvas().plot(history).errorbar(history).save('loss_convergenceA.png')
194.
195. problem1A = pd.read_csv("accidentsA.csv", header = None)
196. problem1A
197.
198. problem1A.describe()
199.
200. #Bootstrap of plant A
201.
202. def resample(v):
203.         return [random.choice(v) for k in range(len(v))]
204.
205. def bootstrap(scenarios, confidence=68):
206.         # len(scenarios) == 1000
207.         samples = []
208.         for x in range(100):
209.                 samples.append(mean(resample(scenarios)))
210.         samples.sort()
211.         # len(samples) == 100
212.         i = int((100-confidence)/2)
213.         j = 99-i
214.         mu_plus = samples[j]
215.         mu_minus = samples[i]
216.         return mu_minus, mu_plus
217.
218. ### START OF CUSTOM LOGIC
219.
220. def main():
221.         mu = 17470.155556
222.         sigma = 29862.673153
223.
224.         def simulate_once():
225.                 return random.gauss(mu, sigma) # simulate_once
226.
```

```
227.    def simulate_many(ap=0.1, ns=10000): # is simulate_many
228.        v = []
229.        for k in range(ns):
230.            x = simulate_once()
231.            v.append(x)
232.            sigma = sd(v)
233.            dmu = sigma/sqrt(k+1)
234.            if k>100 and dmu < ap:
235.                return v
236.        return v
237.
238.    scenarios = simulate_many()
239.
240.    mu = mean(scenarios)
241.    for confidence in (50, 68, 80, 90):
242.        mu_minus, mu_plus = bootstrap(scenarios, confidence)
243.        print 'confidence interval(at %i%%) is %f, %f, %f' % (confidence, mu_minus, mu,
    mu_plus)
244.
245. if __name__ == '__main__': main()
246.
247. #Plant B
248.
249. def read_csv(filename='accidentsB.csv'):
250.    day = []
251.    loss = []
252.    with open(filename) as myfile:
253.        reader = csv.reader(myfile)
254.        previous = 0.0
255.        for row in reader:
256.            row = map(float, row)
257.            day.append(row[0] - previous)
258.            loss.append(row[1])
259.            previous = row[0]
260.    return day, loss
261.
262. def get_parameters():
263.    day, loss = read_csv()
```

```
264.        lamb = float(len(day))/1460
265.
266.        def F_exponential(x, lamb=lamb):
267.              return 1.0-exp(-lamb*x)
268.
269.        points = F(day)
270.        points2 = [(x,F_exponential(x)) for (x,y) in points]
271.
272. #Canvas().plot(F(day)).save('F_dayB1.png')
273.        Canvas().plot(points,color='blue').plot(points2,color='red').save('F_dayB2.png')
274. #Canvas().plot(F(loss)).save('F_lossB.png')
275.
276.        mu = sum(loss)/len(loss)
277.        xM = min(loss)
278.        alpha = mu/(mu-xM)
279.        print 'mu=%f, xM=%f, alpha=%f' % (mu, xM, alpha)
280.
281.        def F_gauss(x, xM = xM, alpha=alpha):
282.              return 0.5+0.5*erf((x-mu)/(alpha*(2.0**0.5)))
283.
284.        points = F(loss)
285.        points2 = [(x,F_gauss(x)) for (x,y) in points]
286.        Canvas().plot(points,color='blue').plot(points2,color='red').save('F_lossB1.png')
287.
288.        return lamb, xM, alpha
289.
290. lamb, xM, alpha = get_parameters()
291.
292. #Simulate once
293.
294. t = 0
295. total_loss = 0
296. while t<365:
297.        t = t + random.expovariate(lamb)
298.        amount = random.gauss(2022.961538, 3336.619195)
299.        total_loss = total_loss+amount
300. print 'One year of losses for plant B =', total_loss
301.
```

```
#90% for plant B

def simulate_once():
    t = 0
    total_loss = 0
    while t<365:
        t = t + random.expovariate(lamb)
        amount = random.gauss(2022.961538, 3336.619195)
        total_loss = total_loss+amount
    return total_loss

scenarios = []
for k in range(100):
    y = simulate_once()
    scenarios.append(y)

Canvas().hist(scenarios).save('scenariosB.png')
scenarios.sort()
print '90% scenarios =', scenarios[int(100*0.9)]

#Simulate many

def read_csv(filename='accidentsB.csv'):
    day = []
    loss = []
    with open(filename) as myfile:
        reader = csv.reader(myfile)
        previous = 0.0
        for row in reader:
            row = map(float, row)
            day.append(row[0] - previous)
            loss.append(row[1])
            previous = row[0]
    return day, loss

def get_parameters():
    day, loss = read_csv()
    lamb = float(len(day))/1462
```

```
340.
341.        def F_exponential(x, lamb=lamb):
342.            return 1.0-exp(-lamb*x)
343.
344.        points = F(day)
345.        points2 = [(x,F_exponential(x)) for (x,y) in points]
346.
347. #Canvas().plot(F(day)).save('F_dayB1.png')
348.        Canvas().plot(points,color='blue').plot(points2,color='red').save('F_dayB2.png')
349. #Canvas().plot(F(loss)).save('F_lossB.png')
350.
351.        mu = sum(loss)/len(loss)
352.        xM = min(loss)
353.        alpha = mu/(mu-xM)
354.        print 'mu=%f, xM=%f, alpha=%f' % (mu, xM, alpha)
355.
356.        def F_gauss(x, xM = xM, alpha=alpha):
357.            return 0.5+0.5*erf((x-mu)/(alpha*(2.0**0.5)))
358.
359.        points = F(loss)
360.        points2 = [(x,F_gauss(x)) for (x,y) in points]
361.        Canvas().plot(points,color='blue').plot(points2,color='red').save('F_lossB1.png')
362.
363.        return lamb, xM, alpha
364.
365. lamb, xM, alpha = get_parameters()
366.
367. class Simulator(object):
368.
369.        def _init_(self):
370.            pass
371.
372.        def simulate_once(self):
373.            t = 0
374.            total_loss = 0
375.            while t<365:
376.                t = t + random.expovariate(lamb)
377.                amount = random. gauss(2022.961538, 3336.619195)
```

```
                total_loss = total_loss+amount
            return total_loss

    def simulate_many(self, ap=100, ns=100):
            sy=0.0
            sy2=0.0
            self.history = []
            for k in range(1,ns+1):
                print k
                y = self.simulate_once()
                sy = sy + y
                sy2 = sy2 + y*y
                y_bar = sy/k
                y2_bar = sy2/k
                sigma = (y2_bar - y_bar**2)**0.5
                dy_bar = sigma/k**0.5
                self.history.append((k,y_bar,dy_bar))
                if dy_bar<ap and k>100:
                    return y_bar
            return y_bar
            #raise ArithmeticError

sim = Simulator()
print sim.simulate_many(ap=0.1, ns=10000)
history = sim.history[:100]
Canvas().plot(history).errorbar(history).save('loss_convergenceB.png')

problem1B = pd.read_csv("accidentsB.csv", header = None)
problem1B

problem1B.describe()

#Bootstrap of plant B

def resample(v):
    return [random.choice(v) for k in range(len(v))]

def bootstrap(scenarios, confidence=68):
```

```
416.        # len(scenarios) == 1000
417.        samples = []
418.        for x in range(100):
419.            samples.append(mean(resample(scenarios)))
420.        samples.sort()
421.        # len(samples) == 100
422.        i = int((100-confidence)/2)
423.        j = 99-i
424.        mu_plus = samples[j]
425.        mu_minus = samples[i]
426.        return mu_minus, mu_plus
427.
428. ### START OF CUSTOM LOGIC
429. def main():
430.        mu = 2022.961538
431.        sigma = 3336.619195
432.
433.        def simulate_once():
434.            return random.gauss(mu, sigma) # simulate_once
435.
436.        def simulate_many(ap=0.1, ns=10000): # is simulate_many
437.            v = []
438.            for k in range(ns):
439.                x = simulate_once()
440.                v.append(x)
441.                sigma = sd(v)
442.                dmu = sigma/sqrt(k+1)
443.                if k>100 and dmu < ap:
444.                    return v
445.            return v
446.
447.        scenarios = simulate_many()
448.
449.        mu = mean(scenarios)
450.        for confidence in (50, 68, 80, 90):
451.            mu_minus, mu_plus = bootstrap(scenarios, confidence)
452.            print 'confidence interval(at %i%%) is %f, %f, %f' % (confidence, mu_minus, mu,
    mu_plus)
```

453.

454. if __name__ == '__main__': main()