Problem Set 2

Research method Problem Set 2 due Mon 22th Oct, 17:00

LIU BOWEN (1004028)

For Problem Set 1, I use the following packages:

```
import matplotlib.pyplot as plt

import numpy as np

import scipy as sp

import scipy.stats as ss

import scipy.stats as ss

import math
```

Problem 1

Answer:

a)

Up to now, I have leant three different methods to calculate standard deviation or standard error, namely, analytical method, perturbation and Monte Carlo.

For the analytical approach, I need to calculation two partial derivation for this problem(r and Q). According to the previous equation, I can obtain the Equation 1.

$$t = In(\frac{Q * r}{P_0} + 1)/r$$

Equation 1

The code is as follow, the rDerivation() and qDerivation() calculate the partial derivation of r and Q. stdDeviat() firstly calculate the deviation and then sqrt() the result.

```
def rDerivation(Q, r, P0):
    return ((Q*r) / (Q*r + P0) - np.log(Q*r/P0 + 1)) / r**2
def qDerivation(Q, r, P0):
    return (1.0/r) * (1.0 / (Q*r/P0 + 1)) * (r/P0)
def stdDeviat(Q, r, dQ, dr, P0):
    analytical_ans = rDerivation(Q, r, P0)**2 * dr**2 +
qDerivation(Q, r, P0)**2 * dQ**2
    return np.sqrt(analytical_ans)
```

As absolute r is 0.2% therefore dr = 0.002, Q is 10% relative so dQ should be 1000*10%=100, and P0 = 5.

```
Q, r, dQ, dr, P0 = 1000, 0.027, 100, 0.002, 5
print 'Problem 1-a result:',    stdDeviat(Q, r, dQ, dr, P0)
```

The answer is shown in Figure 1:

```
Problem 1-a result: 4.181196700587151
```

Figure 1 Analytical standard deviation

b)

As for MC method, I need to define tFunc() to represent the function. Then, randomly generated Q and r each time.

```
def tFunc(Q, r, P0):
    return (np.log((Q*r/P0)+1)) / r
def MC(Num):
    t =[]
    for sample in range(Num):
```

```
            Q = ss.norm.rvs(1000, 100)
            r = ss.norm.rvs(0.027, 0.002)
            P0 = 5.0
            t.append(tFunc(Q, r, P0))
        stdDeviat = np.std(t, ddof=1)
        return stdDeviat
Num = 1000000
print MC(Num)
```

When I input Num = 10^6 the standard deviation is shown in Figure 2:

```
4.227597698707504
```

Figure 2 MC method

Problem 2

Before Monte-Carlo, I firstly need linear fit to calculate the intercept, C. From the equation, I regard the Ea and C as the gradient and intercept therefore I define xValue() and yValue() to calculate each value of x and value of y respectively. The origin data comes from the table with the number of eight pair. After that, I leverage the numpy.polyfit() function. This function fits a polynomial p(x) = p[0] * x**deg + ... + p[deg] of degree *deg* to points (x, y) and return values are the polynomial coefficients, and highest power first. In fact, there are many other ways to fit, such as scipy.stats.linregress()

```
kb = 8.6173303e-5
def xValue(Tc):
```

```
    return (-1 / (kb * Tc))
def yValue(phi, Tc):
    return np.log(phi / Tc**2)
def calculateC(Tc_array, pfi_array):
    x_value = []
    y_value = []
    for i in range(len(Tc_array)):
        x_value.append(xValue(Tc_array[i]))
        y_value.append(yValue(pfi_array[i], Tc_array[i]))
    gradient, C = np.polyfit(x_value, y_value, 1)
    x_mean = np.mean(x_value)
    y_mean = np.mean(y_value)
    return C, x_mean, y_mean
Tc_array = [440.6, 440.3, 439.7, 438.2, 437.3, 434.4, 431.7, 429.7];
pfi_array = [4.5, 3.4, 3.2, 2.7, 2.1, 1.0, 0.8, 0.5]
C, x_mean, y_mean = calculateC(Tc_array, pfi_array)
print 'The intercept C:', C
```

The value of C is shown in Figure 3:

```
The intercept C: 69.31194966606428
```

Figure 3 The value of intercept C

a)

When I solve it by Monte Carlo algorithem, I firsly need to calculate the mean of Tc and pfi. There are two ways to get mean of Tc and pfi. One is calculate the mean from Tc_array[] and pfi_array[]. The other is calculate the mean from the eight-pair data from xValue() and yValue() then calculate the mean of Tc and pfi. Shown in code below, as xValue equals (-1 / (kb * Tc)) therefore mean of Tc can be done by (-1 / (x_mean*kb)) where x_mean is the 8 data from xValue(). After that, the mean of Tc and pfi is shown in Figure 4.

```
def Ea(Tc, pfi, C):
    return kb * Tc * (C - np.log(pfi / np.square(Tc)))
meanTc = -1 / (x_mean*kb)
meanpfi = math.exp(y_mean)* meanTc**2
print 'The Tc mean:', meanTc
print 'The pfi mean:', meanpfi
stdDeviatTc = 0.2
stdDeviatpfi = 0.1
num = 10000
Ea_value = []
def MCEa(Tc, stdTc, phi, stdphi, C, N_experiments):
    for i in range(N_experiments):
        Tc_random = ss.norm.rvs(Tc, stdTc)
        phi_random = ss.norm.rvs(phi, stdphi)
        Ea_value.append(Ea(Tc_random, phi_random, C))
    mean = np.mean(Ea_value)
    stdDeviat = np.std(Ea_value)
```

```
    stdError = stdDeviat / math.sqrt(N_experiments)

    return mean, stdDeviat, stdError
```

The mean of Tc and pfi is as below:

```
The Tc mean: 436.45332650059544
The pfi mean: 1.8015714666282996
```

Figure 4 Mean of Tc and pfi

b)

```
mean, stdDeviat, stdError = MCEa(meanTc, stdDeviatTc, meanpfi, stdDeviatpfi, C, num)

print 'The Ea mean:', mean

print 'The Ea standard deviation:', stdDeviat

print 'The Ea standard error:', stdError

num_bins = 100

plt.hist(Ea_value, num_bins, facecolor='red', edgecolor='black')
```

Continue the code, the histogram of 10000 simulations is shown in Figure 5.

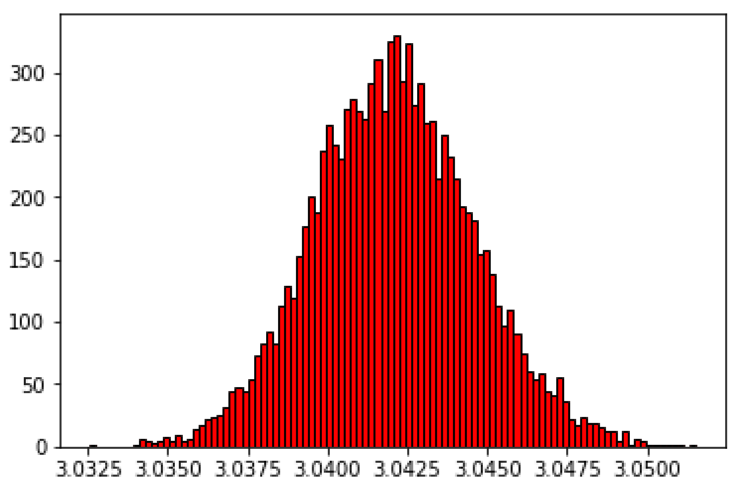

Figure 5. Histogram of 10000 simulations

c) and d)

The Figure 6 shows the result of standard deviation and standard error. Noted that the sample size is 10000 therefore the standard error = standard deviation / sqrt(10000).

```
The intercept C: 69.31194966606428
The Tc mean: 436.45332650059544
The pfi mean: 1.8015714666282996
The Ea mean: 3.042061650163298
The Ea standard deviation: 0.0025278742262418255
The Ea standard error: 2.5278742262418255e-05
```

Figure 6. Standard deviation and Standard error

e)

I calculate the min and max of Ea by two-tails confidence interval. Use norm.ppf() to obtain the value for probability = (1-0.95)/2.

```
def EaBounds(meanEa, stdDeviaEa, confidence):

    z = ss.norm.ppf((1+confidence)/2)

    minEa = meanEa - stdDeviaEa * z

    maxEa = meanEa + stdDeviaEa * z

    return minEa, maxEa

minEa, maxEa = EaBounds(mean, stdDeviat, 0.95)

print 'The minimum Ea:', minEa

print 'The maximal Ea:', maxEa
```

The code is as above and the min and max value are shown in Figure 7.

```
The minimum Ea: 3.036994557769688
The maximal Ea: 3.0470498066517084
```

Figure 7. The min and max Ea