ENGN2020 – HOMEWORK9

Problem 1

Part (a)

The code of Metropolis algorithm is shown as below.

```
def metropolis(initial_guess = 1.0, steps = 500,maxStep = 0.1):
    accept = 0

    T = 300
    kb = 1.38064852e-23
    evToJ = 1.60218e-19
    l_old = initial_guess

    result = np.zeros((steps))
    means = np.zeros((steps))
    stds = np.zeros((steps))
    acceptNums = np.zeros((steps))

    for i in range(steps):
        #get energy of old l
        El_old = get_energy(l_old)
        #random jump to a new l
        l_new = l_old+np.random.uniform(-1,1)*maxStep
        #get the energy of new l
        El_new = get_energy(l_new)

        #calculate r by two energy, note that r is for logP
        r = -(El_new-El_old)*evToJ/kb/T

        #if P(new)>P(old)
        if r>0:
            #save new l
            accept = accept + 1
            l_old = l_new
            result[i] = l_new
        else:
            #generate a new random value from [0,1]
            prob = np.random.uniform(0,1)
            #see if the we accept the new step
            if math.exp(r) > prob:
                accept = accept + 1
                l_old = l_new
                result[i] = l_new
            else:
                result[i] = l_old

        #get min
        current = result[:i+1]

        current_mean = np.mean(current)
        means[i] = current_mean
        #get std

        current_std = np.std(current)
        stds[i] = current_std

        #get accept ratio
        acceptNums[i] = accept/(i+1)

    return result,means,stds,acceptNums
```

Choose 0.5 Å as the start point, and set T by 300K and max step size by 0.1 Å.
The figure of average of ***l*** versus step number is show in Fig.1.

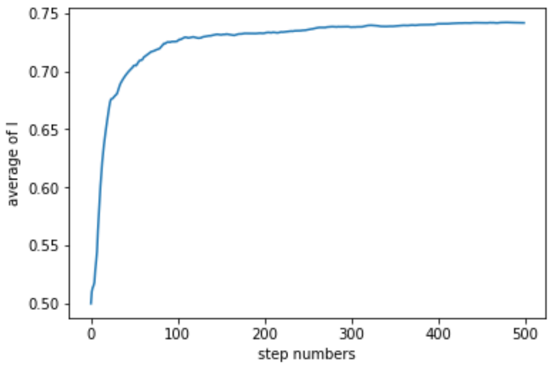


Fig 1. The figure of average of l versus step number

The figure of standard deviation of ***l*** versus step number is show in Fig.2.

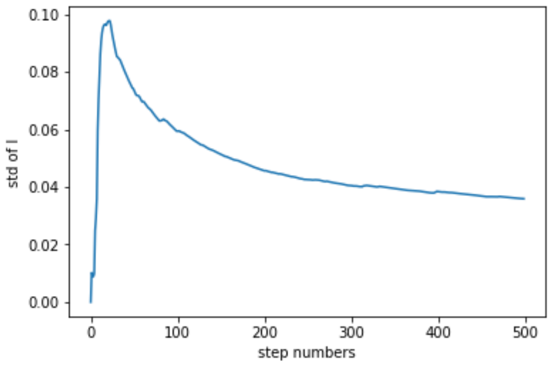


Fig 2. The figure of standard deviation of l versus step number

The figure of fraction of steps accepted versus step number is show in Fig.3.

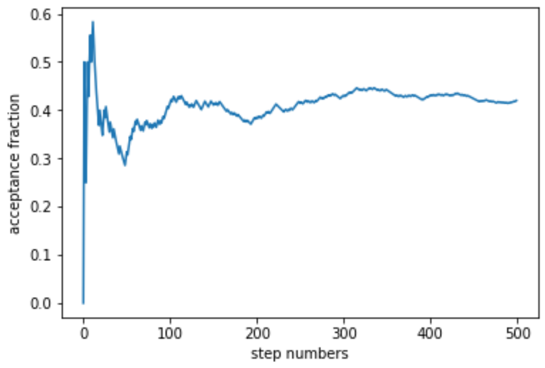


Fig 3. The figure of standard deviation of l versus step number

The histogram of ***l*** after 500 steps is shown in Fig.4.

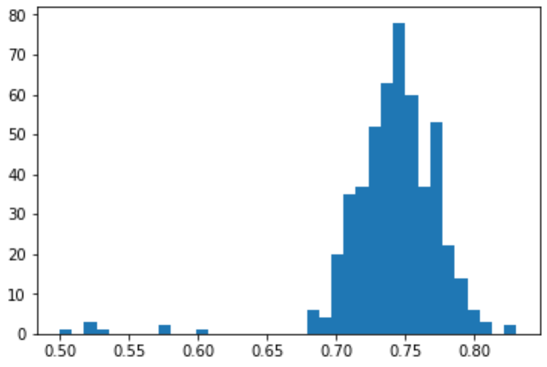


Fig 4. The figure of standard deviation of l versus step number

From the histogram, we can find that samples distribute normally at nearly 0.75 Å, where the corresponding energy is the smallest.

Part (b)

Under different maximum step size, namely 0.01 Å, 0.1 Å, 1 Å, 10 Å, the plots of average of l against step number is

shown in Fig.5.
As the maximum step size increases, the algorithm converges more quickly. However, when the step size is too big, sometimes, the algorithm couldn`t converge to the right result. The system becomes instable.

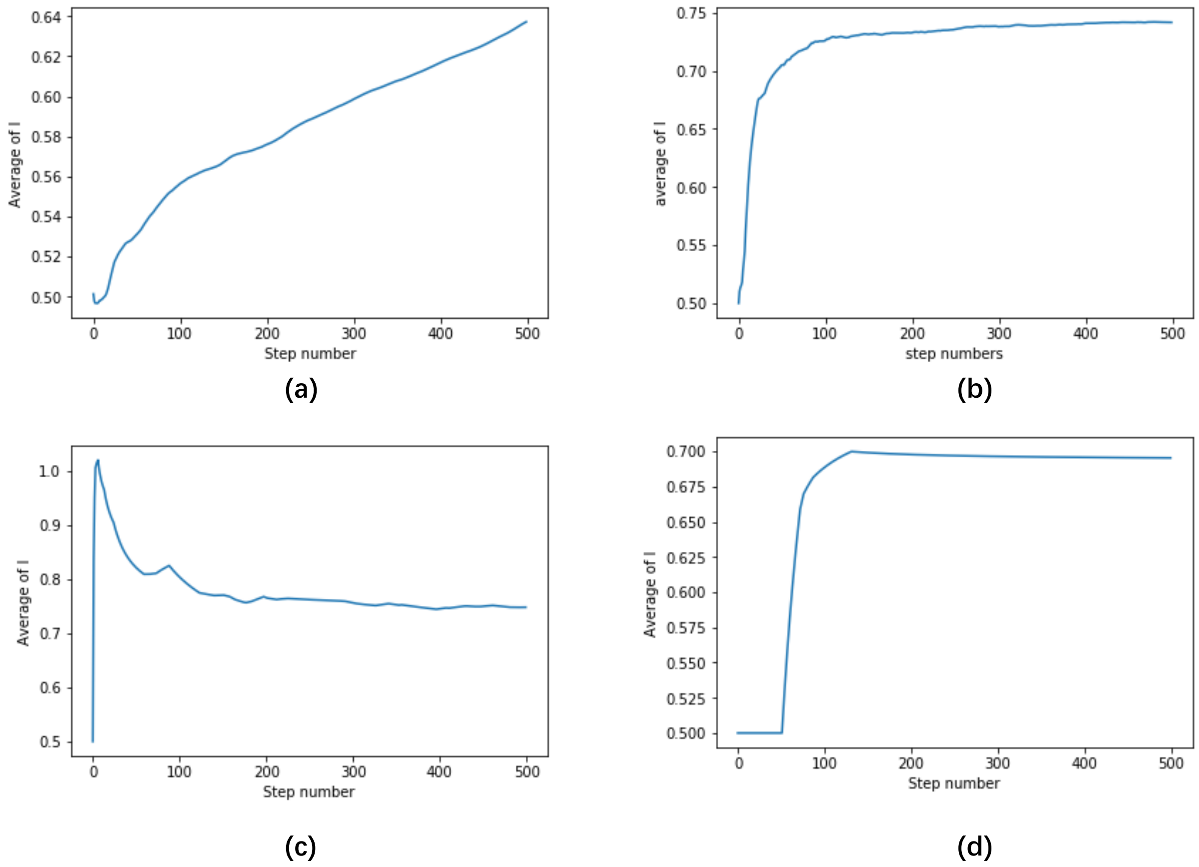


Fig 5. Plots of average of I against step number under different maximum step sizes. **(a)**0.01 Å**(b)**0.1 Å**(c)**1 Å**(d)**10 Å
0.1 Å seems a very optimal maximum step size, the algorithm converges quite fast, and according to my experiments, when the initial guess is between 0.3 Å to 2 Å, every time the algorithm can converge to the same result.

Part (c)

Set the initial guess by 3 Å, and change the temperature from 300 K to 1500K, the fraction of broken bonds is shown in Fig.6.

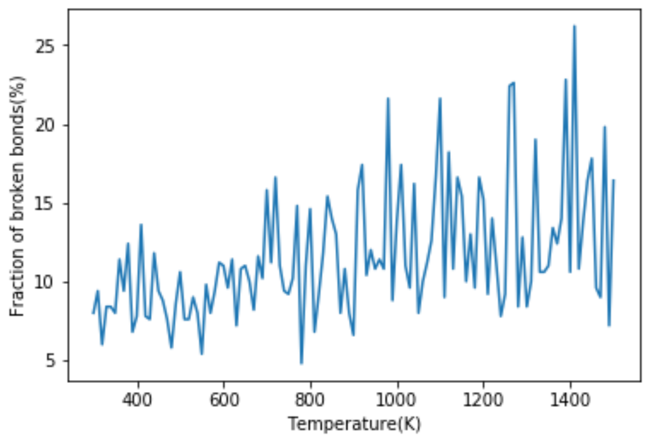


Fig 6. Plots of fraction of broken bonds against temperature
From the figure, we can find that as the temperature increases, the fraction of broken bonds also tends to increase.

Problem 2

$$Nu = \alpha_0(Re)^{\alpha_1}(Pr)^{\alpha_2}$$

Where:

$$Nu = \frac{hD}{k}$$
$$Pr = \frac{\mu \hat{C}_p}{k}$$
$$Re = \frac{\rho v_f D}{\mu}$$

To obtain a linear model, we take the base-10 logarithm:

$$log_{10}Nu = log_{10}\alpha_0 + \alpha_1 log_{10}Re + \alpha_2 log_{10}Pr$$

Based on the given data, the base-10 logarithm is shown in Table 1.

Table.1 The data after base-10 logarithm

log_{10}Nu	log_{10}Re	log_{10}Pr
0.29393681	0	-0.1366771
-0.0464336	-1	-0.1366771
-0.3704885	-2	-0.1366771
0.39963911	0	0.17609126
0.06149018	-1	0.17609126
-0.2580609	-2	0.17609126

By following the tutorial of the ***sampyl*** library, the parameter vector $b = [\, log_{10}\alpha_0, \alpha_1, \alpha_2]$ can be calculated by the following code.

```
import sampyl as smp
from sampyl import np

X = np.array([[1, 0, -0.13667714],
              [1, -1, -0.13667714],
              [1, -2, -0.13667714],
              [1, 0, 0.176091259],
              [1, -1, 0.176091259],
              [1, -2, 0.176091259]])

y = np.array([[0.293936814],
              [-0.046433586],
              [-0.370488466],
              [0.399639115],
              [0.061490177],
              [-0.258060922]])

def logp(b, sig):
    #define the model
    model = smp.Model()

    # Predicted value
    y_hat = np.dot(X, b)

    # Log-likelihood
    model.add(smp.normal(y, mu=y_hat, sig=sig))

    # log-priors
    model.add(smp.exponential(sig),
              smp.normal(b, mu=0, sig=100))

    return model()

start = smp.find_MAP(logp, {'b': np.ones(3), 'sig': 1.})
nuts = smp.NUTS(logp, start)
chain = nuts.sample(2100, burn=100)
import matplotlib.pyplot as plt
plt.plot(chain.b)
```

A chain of 2000 samples is calculated, and the figure of those samples is shown in Fig.7.

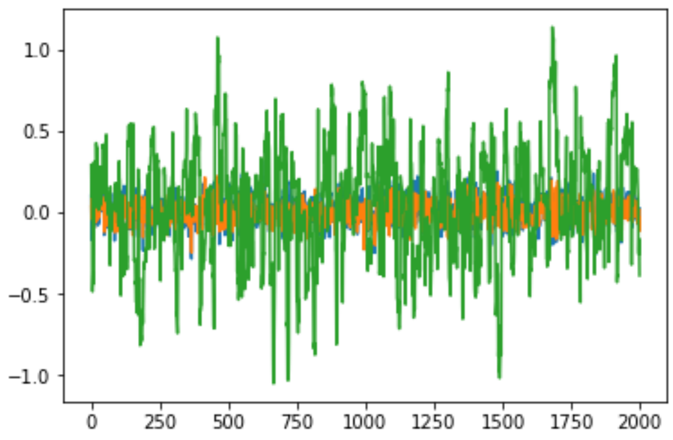
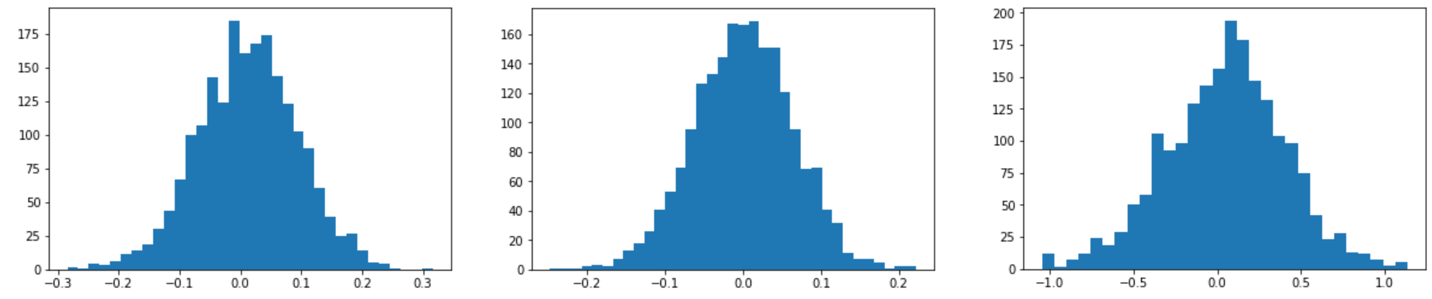


Fig 7. The figure of those samples of the parameters

The histograms of each parameter in vector b is shown in Fig.8.



(a) (b) (c)

Fig 8. The histograms of each parameter.**(a)** $log_{10}\alpha_0$**(b)** α_1**(c)** α_2

The code below is used to calculate the 95% confidence section.

```
from scipy.stats import norm
import math

def get_confidence_interval(data):
    mu, std = norm.fit(data)
    length = len(data)
    return mu-1.96*std/math.sqrt(length),mu+1.96*std/math.sqrt(length)
```

After calculation, the 95% confidence section of $log_{10}\alpha_0$ is [0.00976, 0.01621].

Therefore, the the 95% confidence section of α_0 is [1.0098, 1.0163].

The 95% confidence section of α_1 is [-0.00281, 0.00236]

The 95% confidence section of α_2 is [-0.01998, 0.01055]