Lab01

Heng Li, Lihao Wang

Question 1

(a) The Pseudo-code is shown as following:

```
#Pseudocode:
#1. First of first, import numpy
#2. Secondly, import speed of light data
#3. Then Write the two methods into functions
#4. Compute the relative errors:
    #calculate the correct answer
    #calculate std by method1
    #calculate std by method2
    #calculate and print the relative errors respect to the correct answer
```

*We don't think we need to check if there is square root of a negative number, because $\sum_{i=1}^{n} x_i^2 - n\bar{x}^2$ cannot be negative since quadratic mean cannot be smaller than arithmetic mean.

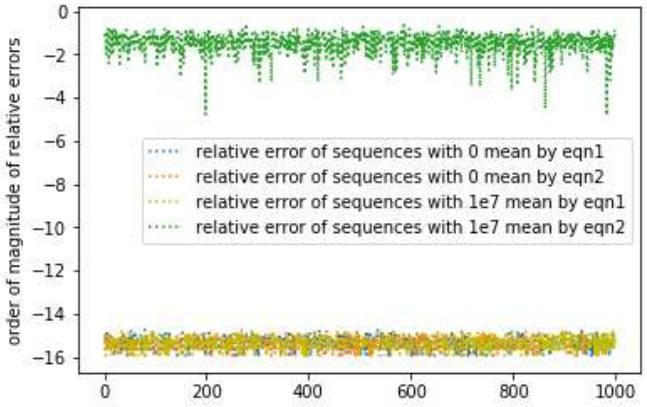
(b) Printed output:

```
For Michelsen's speed of light data
relative error from Eqn 1 is 3.512894971845343e-16
relative error from Eqn 2 is 7.265258724578925e-10
```

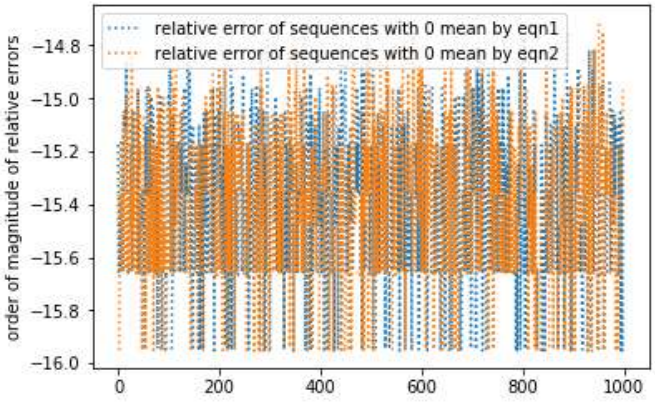
Answers to questions:

The result calculated by equation 2 has larger relative error in magnitude.

(c) We calculate the relative error of the two random sequences 1000 times and plot their order of magnitude (the graph on the right). From this graph, we could easily find out that the method using equation 2 causes much larger relative error of sequences with a large mean than using equation 1.



As for the sequences with 0 mean, relative error for the two equations is almost at the same level, which is shown in the graph on the right.



In conclusion, the equation 2 will brought larger relative error than equation 1, and the error would be larger and larger as the data has a larger mean. The reason this problem is caused is that the deviation of a single data point lost digits because the data is squared before subtracting the mean.

(d) Answer: As we mentioned in (c), the problem is caused by squaring the data before subtracting the mean. So, to solve this problem, we should subtract the mean of the dataset from the data. In conclusion, we think we should use the method1 function in our codes to get the smallest relative error no matter how large the mean value is.

Question 2.

(a)

i. Printed outputs:

```
the value using Trapezoidal's rule is 0.9999719125941186
the value using Simpson's rule is 0.9999770112979357
the value using erf function is 0.9999779095030014
```

From the results, we can easily see that Simpson's method gets more accuracy.

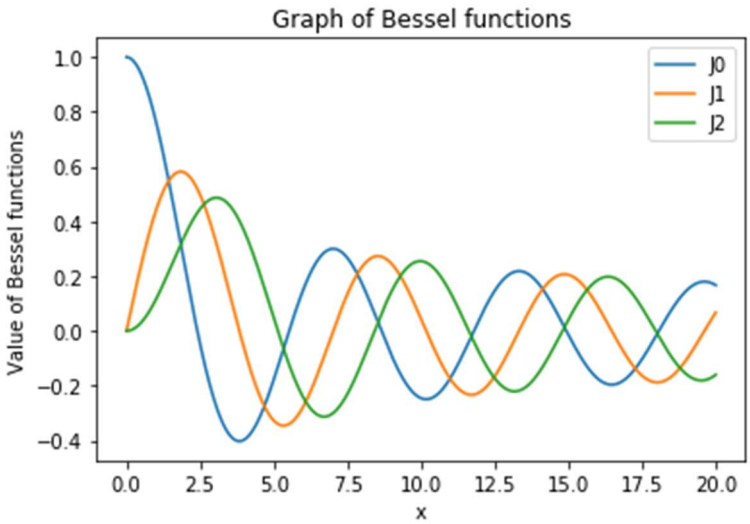
ii. It needs 4 slices for Trapezoidal's rule and it needs 3 slices for Simpson's rule. Trapezoidal's rule spent 0.0017287731170654297 sec and Simpson's rule cost 0.0004096031188964844 sec whereas scipy.erf only took 5.7220458984375e-05 sec.

iii. The practical estimation of error for Trapezoidal's rule when N=10 is 5.2556060231208806e-06. The practical estimation of error for Simpson's rule when N=10 is 1.9653264411445547e-07.

iv. Result of Euler-Maclaurin formula for Trapezoidal's rule when N=10 is 5.5534411839e-06. Result of Euler-Maclaurin formula for Simpson's rule when N=10 is 3.33206471034e-08. The reason why the two method give different results is that we treat the error O(h^2) as terms in h^2 but ignore the higher order terms.

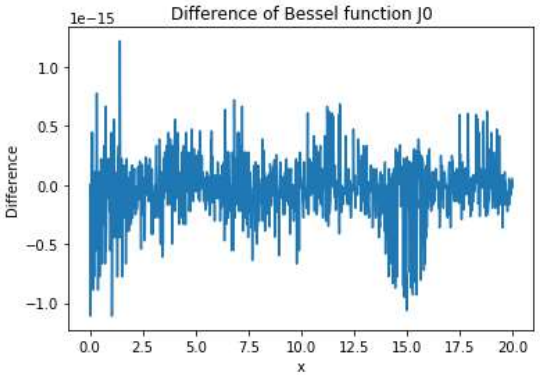
(b)

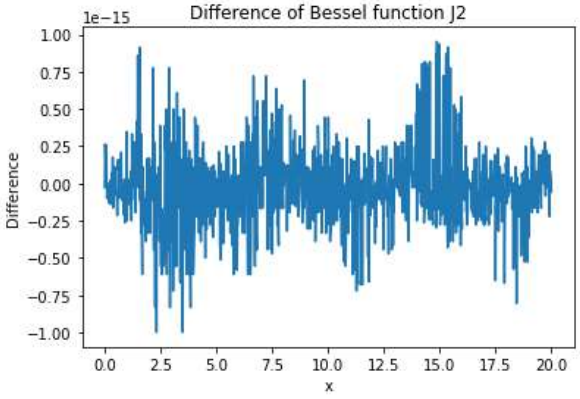
i) The graph of three Bessel functions is shown on the right.

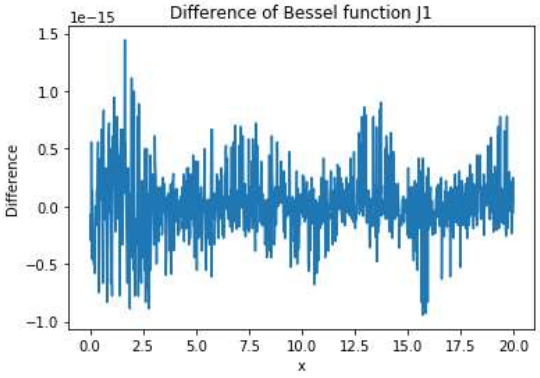


ii) The difference between our Bessel functions and those from scipy is shown on the right.

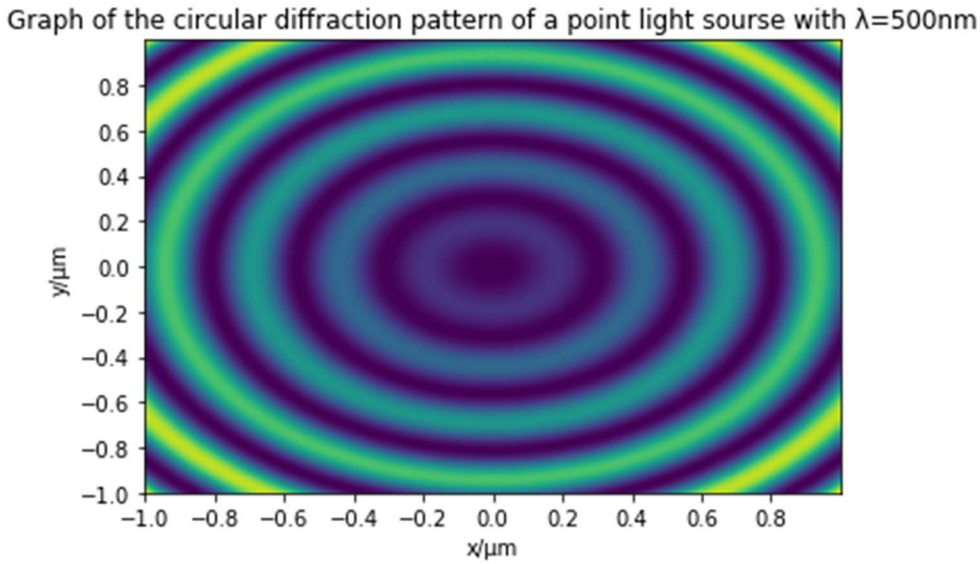
We could see that the difference is very small compared with their value. So our reproduction of Bessel functions is very accurate.







iii) The density plot is shown on the right.



Question 3

a) We use change of variables and Trapezoidal's rule with 100 slices to calculate the integral. The result is 6.493938607674634 with practical estimation of error 4.839602532058507e-07.

b) The result we get is 5.670366524853972e-08. And the value given by scipy.constant is (5.670367e-08, 'W m^-2 K^-4', 1.3e-13). It means the result we get fall into the uncertainty interval of Stefan-Boltzmann constant.