Answer to Question 1 and 2 in Lab 2:

Question1a:

The reason we need to condition check the negative value for equation 6 is that if the sum is negative, the square root of the sum will be a complex number, which may cause error. The pseudo code for equation (5) and (6) are shown below:

Pseudo code for (5):

```
def Pesudo_for_5:
        data_point = np.loadtxt( cdata.txt )
        x_mean = (sum of data in data_point) / (number of element in data_point)
        result = 0
        for i=0 to i = size of data_point:
                result += (data_point[i] - size of data_point * x_mean)^2
        estimate_std = squareroot( result / (size of data_point - 1))
        Calculate std using np.std(data_set)
        calculate relative error using (x-y)/y and return it
```

Pseudo code for (6):

```
def Pesudo_for_6:
        data_point = np.loadtxt(cdata.txt)
        x_mean = (sum of data in data_point) / (number of element in data_point)
        result = 0
        for i=0 to i = size of data_point:
                result += data_point[i] - size of data_point * x_mean^2
        if result is negative:
                print error
                return 0
        estimate_std = squareroot( result / (size of data_point - 1))
        Calculate std using np.std(data_set)
        calculate relative error using (x-y)/y and return it
```

Question1b:

The code for this question is written in the file Question_1_and_2.py as *method_1_1b()* and *method_2_1b()* where the first one is the implementation for equation (5) and the other one is the implementation of equation (6). The relative error (output) for the implementation of equation (5) is:

377606.01478819316

And the relative error (output) for the implementation of equation (6) is:

37949.932903619876

Compare both results, the implementation of equation (5) have a greater magnitude in relative error.

Question1c:

Same as above, the code for this question is written in the file Question_1_and_2.py, the output for the equation (5) using the generated input is:

4.3862741278688407e-16 and 5.713504331290894e-16

And for equation (6):

0.2106840695854981 and 460296131.7119557

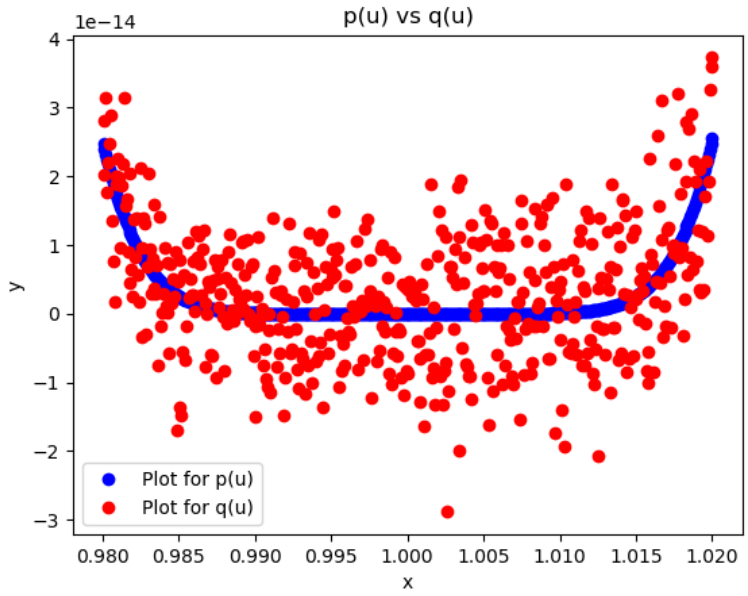
In general, the equation (6) implementation have a greater magnitude for relative error compare to equation (5) implementation. We suspect that the one-pass methods involve more step of calculation compare to the other method, especially on the summation part. Instead of subtracting the number and then square them, equation (6) square each of the element and then subtracts them, which will introduce some extra rounding error during the process.

Question1d:

One way to work around this is to make the data smaller by shifting them towards zero. Ideally the mean of the data after shitting should be zero. Since shifting data by constant amount does not affect the deviation, the deviation should remain the same, however, since the data is smaller, there will be more bits to be used to represent decimal points since the integer part is very small, which improves the accuracy. By using this modification, the relative error to the one-pass method drops from 37949.932903619876 to 106.8796951249224, for cdata.txt with shift -299 to the data. The fixed one-pass equation is implemented as method *fixed_one_pass* in the python file.

Question2a:

The plot of p(u) and q(u) around 1 is shown below:



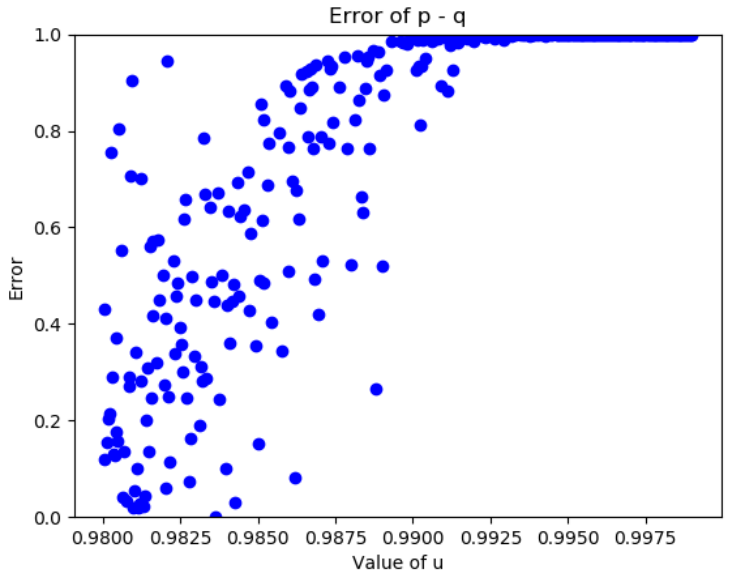
q(u) is the expansion of p(u), and the plot of q(u) is significantly greater than p(u), we conclude that since q(u) have to calculate each power term and add them up, the rounding off error for floating point accumulate as the power was calculated for each term in q(u) and result in the scattered red dot on the graph.

Question2b:

From the plot of the histogram, we notice that the value of x- axis of histogram has approximately the same magnitude. To calculate using equation (7) in the lab manual, we calculate the mean of x mean first. We calculate the x square mean by square each term of q(u), add them together and divide them by the number of the terms in p(u), and do that for each of the u we pick, then we divide it by how many data point we have. For the number N in equation (7), we take the sum of 1 to 8 +8. We plus 8 here because there is also p(u) there. And the output for the program is 2.6529746846076516e-14, compare to the standard deviation provided by the numpy.std (8.319904710810596e-15), they are nearly at the same magnitude.

Question2c:

The plot to problem 2c is shown blow, and indeed as u approaches 1, the error goes up to 100%



Question2d:

The plot for f(u) – 1 for u near 1.0 is shown below, by comparing with equation 4.5 at pg. 131, the error has the same magnitude as equation 4.5 predict. However, notice that the error has some kind pattern. Most of the pattern is concentrated around y = 2.2e-16, -1.1e-16, -2.2e-16; We suspect that it is due to rounding error where some float was rounded to the same value as the others.