Lab assignment #2: Numerical Errors and Integration
Author: Yinshi Liu, Landon Wang
Contributions: Yinshi Liu wrote the entirety of Q1 and 2, Landon Wang wrote the entirety of Q3 and 4. The report was compiled by Yinshi Liu.

Q1a
Pseudocode

1. Create an array from loaded data
2. Calculate the total sum of the array using loops
   Create an variable named x_sum and set it to 0
   for each value of array, add the value to sum
   iterate for all values
3. Calculate the average value using the formula below:
$$x_{avg} = \frac{x_{sum}}{len(a)}$$
4. Calculate the value of $\sum_i^n \left(x_i - x_{avg}\right)^2$:
   Create a variable named sum_1 and set it to 0
   For each value of the array a, calculate $\left(x_i - x_{avg}\right)^2$ and add it to sum_1
   Iterate
5. Put everything together and calculate standard deviation using
$$\sigma = \sqrt{\frac{1}{n-1}\sum_i^n \left(x_i - x_{avg}\right)^2}$$
   Create a variable named sigma_1 and set it to the result of
$$\sqrt{\frac{1}{len(a)-1}sum\_1}$$
6. Call np.std() to find the true value. Calculate relative errors using
$$\frac{sigma_1 - np.std(a)}{std}$$
   and print results.

7. Calculate $\sum_i^n (x_i^2)$ using loops
   Create a value named sum_2 and set it to 0
   For each value in array, add x_i^2 to sum_2
   Iterate
8. Calculate x_{avg} using loops
   Create a value named sum_1 and set it to 0
   For each value in array, add x_i to sum_1
   Iterate.
   Divide sum_1 by the length of the array. This variable is named x_{avg} .
9. Calculate standard deviation using

$$\sqrt{\frac{1}{n-1}(\sum_{i}^{n} x_i^2 - nx_{avg}^2)}$$

Create a variable called var_2 and set it to the result of

$$\frac{1}{len(a)-1}(sum_2 - x_{avg}^2)$$

if var2 is negative, then print out a warning for negative variance. Stop the calculation. Else, calculate sigma_2 using $sigma_2 = \sqrt{var_2}$

10. Calculate relative error using

$$\frac{sigma_2 - np.std(a)}{std}$$

and print results.

Q1b

Below are the outputs of the code

```
sigma_1 = 0.0790105478190507 np.std = 0.07901054781905067
relative error = 3.512894971845343e-16
sigma_2 = 0.07901054823364538 np.std = 0.07901054781905067
relative error = 5.247333643835157e-09
```

According to the printed results, the relative error using equation 2 is greater in magnitude, with a relative error of ~10^{-9}, which is roughly 7 order of magnitudes larger than that of equation 1 (~10^{-16}).

Q1c

The output of the code is shown below.

```
sigma_1a = 0.981420374766329 np.std for a = 0.9814203747663289
magnitude of relative error with eqn 1 for a = 1.131241059560736e-16
sigma_2a = 0.9814203747663286 np.std = 0.9814203747663289
magnitude of relative error with eqn 2 for a 3.393723178682208e-16

sigma_1b = 1.0159511951360536 np.std for a = 1.0159511951360525
magnitude of relative error with eqn 1 for b = 1.092791691117091e-15
sigma_2b = 1.1387046694213918 np.std for a = 1.0159511951360525
magnitude of relative error with eqn 1 for b = 0.12082615274535963
```

As the size of the mean increases, the size of relative error using equation 2 compared to the np.std() method grew rapidly. On the other hand, the magnitude of relative error using equation 1 remained at a similar level.

The size of relative error for equation 2 has increased from ~10^{-16} to ~10^{-1}. This constitutes a growth of 15 orders of magnitude. The most likely explanation of this increase is the accumulation of numerical errors during the additions or subtractions, and the increased sizes of the mean with a small standard deviation has exaggerated this error. As the comparison between

the 2 equations has shown, the 1-pass method proves to be more unstable compared to that of the 2-pass.

Q1d

Changes to method 2: In order to reduce the magnitude of the relative error, the simplest approach is to change method to a 2-pass method by calculating x_{avg} separately. In this particular case, the value of x_{avg} can be obtained using the average() function in Q2b.

Pseudocode (for equation_2 only):

1. Calculate x_{avg}. Call the average function and return its output.
2. Calculate standard deviation using

$$\sqrt{\frac{1}{n-1}(\sum_{i}^{n} x_i^2 - n * x_{avg}^2)}$$

   Create a variable called var_2 and set it to the result of

$$\frac{1}{len(a)-1}(sum_2 - x_{avg}^2)$$

   if var2 is negative, then print out a warning for negative variance. Stop the calculation. Else, calculate sigma_2 using $sigma_2 = \sqrt{var_2}$

3. Calculate relative error using

$$\frac{sigma_2 - np.std(a)}{std}$$

   and print results.

Result from one of the trials

```
sigma_1a = 0.9926905544580432 np.std for a = 0.9926905544580434
magnitude of relative error with eqn 1 for a = 2.2367957862383e-16
sigma_2a = 0.9926905544580426 np.std = 0.9926905544580434
magnitude of relative error with eqn 2 for a 8.9471831449532e-16

sigma_1b = 1.0004008233423693 np.std for a = 1.0004008233423691
magnitude of relative error with eqn 1 for b = 2.2195563992357945e-16
sigma_2b = 1.0278756073241868 np.std for a = 1.0004008233423691
magnitude of relative error with eqn 1 for b = 0.02746377585938365
```

Compared to the result from Q2b, the magnitude of relative error has reduced from ~10^{-1} to ~10^{-2} according to multiple trials. While this is not a perfect solution, but a tenfold decrease of error is still significant.

Q2a

$$\int_0^1 \frac{4}{1+x^2}dx = 4\int_0^1 \frac{1}{1+x^2}dx = 4(\arctan(1) - \arctan(0)) = \pi$$

Q2b

Pseudocode:

1. Define function f(x), which calculates $\frac{4}{1+x^2}$ for x
2. Define function named trapezoidal, which requires variable N (number of slices), a (start point), b (end point).

   Calculate h (the width of the slices) using $\frac{b-a}{N}$
   Create variable named s, which represents the total area under the curve. Initialize the value using $0.5f(a) + 0.5f(b)$, which represents the first and last side of the trapezoid.

   For each value that falls in the range from 1 to N-1, call the f(x) function to compute the value of f(a + i*h). Add this value to s. This computes the sum of

   $$h[\frac{1}{2}f(a) + \frac{1}{2}f(b) + \sum_{i=1}^{N-1} f(a+ih)]$$

   outlined on Pg. 142 of the textbook.

   Multiply s to h (width of one slice). Return this result.

3. Define another function called Simpson, this requires the same variables as trapezoidal rule.

   Calculate h with the same method
   Create variables named even and odd, set both to 0
   For each integer value that is between 1 and N-1, check if it is even or odd.
   If the number is even, calculate $f(a+ih)$ and add it to even.
   If the number is odd, calculate $f(a+ih)$ and add it to odd.
   Iterate

   Return $(f(a) + f(b) + 4odd + 2even)\frac{h}{3}$, this calculates the value of

   $$\frac{h}{3}[f(a) + f(b) + 4\sum_{odd}^{N-1} f(a+ih) + 2\sum_{even}^{N-2} f(a+kh)]$$

   as outlined on Pg. 146 of the textbook.

4. Call both functions and print the result. Compare them with true value.

Result
The output of the program is shown below.

```
trapezoidal method = 3.1311764705882354
Simpson's Method = 3.14156862745098
true value = 3.141592653589793
```

According to the output, the Simpson's method is closer to the exact value compared to the trapezoidal method.
Note: in the code, the np.pi function is used for true value.

Q2c

Pseudocode

1. Create 2 variable named Simpson_ error and trapezoidal_error that calculates the difference between their respective results and the exact value.
2. Create a variable named n and set it to 1.
3. Start timing the trapezoidal method's performance.
   Call trapezoidal method and calculates the integral above with 2^n slices.
   While the error term is greater than 10^{-9}, add 1 to n
   Iterate.
   Print the final value of n and stop timing.
4. Start timing the Simpson method's performance.
   Call Simpson method and calculates the integral above with 2^n slices.
   While the error term is greater than 10^{-9}, add 1 to n
   Iterate.
   Print the final value of n and stop timing.
5. Print the time elapsed for both methods and n.

Results

The output of the code

```
results for trapezoidal method:
number of slices = 2^ 14 slices
time taken = 0.014214992523193 36 s
result = 3.1415926529689018

results for Simpson's method:
number of slices = 2^ 5 slices
time taken = 8.606910705566406e-05 s
result = 3.1415926535897953
```

According to the result, the Simpson's method takes both less time and iterations to reach the desired accuracy compared to the trapezoidal method. After multiple trials (not shown on this report), the time and slices required for Simpson's method are around $8*10^{-5}$ s and ~2^5 slices, and trapezoidal method requires a much larger ~$1.5*10^{-2}$ s and ~2^{14} slices.

Q2d

Pseudocode is based on the code for part c.

1. Call trapezoidal function with $N_1 = 16$, a = 0, and b = 1 to calculate I_1.
2. Call trapezoidal function with $N_2 = 32$, a = 0, and b = 1 to calculate I_2.
3. Calculate error term using the formula $\epsilon = \frac{1}{3}(I_2 - I_1)$ then print the result. This is the method outlined on Pg. 155 of the textbook.

Output

```
estimated error = 0.0001627603778620 0348
```

According the code output, the error term ϵ has a value of $1.6 * 10^{-4}$ in this particular case. Since $\epsilon = ch_2^2$, and $h_2 = \frac{1}{N_2} = \frac{1}{32}$, the error constant c has a value of $c = \frac{\epsilon}{h_2^2} = (0.0001627)(32^2) = 0.1667$.

Q2e

Derivation of the practical estimate of error term for Simpson's rule:

Since Simpson's rule is accurate to the 4th degree, the error term for I_1 would be:

$$\epsilon_1 = ch_1^4.$$

The relation between the true value I and I_1 would then be:

$$I = I_1 + ch_1^4.$$

Similarly for I_2 : $I = I_2 + ch_2^4$. Thus:

$$I_1 + ch_1^4 = I_2 + ch_2^4$$

since $h_1 = 2h_2$:

$$I_2 - I_1 = ch_1^4 - ch_2^4$$

$$I_2 - I_1 = 15ch_2^4$$

since $\epsilon_2 = ch_2^4$, therefore the estimation would be:

$$\epsilon_2 = \frac{1}{15}(I_2 - I_1)$$

Output of estimated error vs. actual error is shown below:

```
estimated error for Simpson= 1.55200948389241e-10
actual error for Simpson= 3.695665995451236e-11
```

Which actually shows that the estimated error is much larger compared to the real value. Notably, this does not happen with the trapezoidal method:

```
estimated error for trap. = 0.00016276037786200348
actual error for trap. = 0.00016276041481821935
```

which is indeed very similar to each other.

One possible reason that the estimate is that Simpson's rule divides the sections into 2 equal slices. This means it will double the intervals of N_1 and N_2. Therefore, one possible way to fix this would be to double the count of both N_1 and N_2.

Q3

A

We have the equation 5 from the lab manual:

$$B = \frac{2hc^{-2}v^3}{\exp\frac{hv}{kT} - 1} \tag{3.1}$$

By definition, we have:

$$W = \pi \int_0^\infty B dv \tag{3.2}$$

Plugin equation 3.1 into equation 3.2:

$$W = \pi \int_0^\infty \frac{2hc^{-2}v^3}{\exp\frac{hv}{kT} - 1} dv \tag{3.3}$$

Notice that we are expected to manipulate the equation 3.3 into the form of

$$W = C_1 \int_0^\infty \frac{x^3}{e^x - 1} dx \tag{3.4}$$

Since the components inside the exponential function is hard to manipulate and simplify, we let

$$x = \frac{hv}{kT} \tag{3.5}$$

Therefore

$$v = \frac{kT}{h}x \tag{3.6}$$

Hence

$$dv = \frac{kT}{h}dx \tag{3.7}$$

Now, change the variable according to equation 3.6 and 3.7

$$W = \pi \int_0^\infty \frac{2hc^{-2}\left(\frac{kT}{h}\right)^4 x^3}{e^x - 1}dx \tag{3.8}$$

Simplify, we have

$$W = \frac{2\pi k^4 T^4}{h^3 c^2}\int_0^\infty \frac{x^3}{e^x - 1}dx \tag{3.9}$$

Therefore, we arrived at

$$C_1 = \frac{2\pi k^4 T^4}{h^3 c^2} \tag{3.10}$$

B

From textbook section 5.8, we have learned that to solve a problem with infinite integral range, we can apply a change of variable such that the integral becomes a finite one. Here, we use:

$$z = \frac{x}{1+x} \tag{3.11}$$

Rearrange,

$$x = \frac{z}{1-z} \tag{3.12}$$

Then,

$$dx = \frac{dz}{(1-z)^2} \tag{3.13}$$

Apply this transformation to the equation 3.9, we have

$$W = C_1 \int_0^1 \frac{1}{(1-z)^2}\frac{\left(\frac{z}{1-z}\right)^3}{\exp\frac{z}{1-z} - 1}dz \tag{3.14}$$

Since the integral is independent of temperature, we can rewrite the equation 3.14 into a function of temperature and define the Stefan-Boltzmann constant:

$$\sigma = \frac{2\pi k^4}{h^3 c^2}\int_0^1 \frac{1}{(1-z)^5}\frac{z^3}{\exp\frac{z}{1-z} - 1}dz \tag{3.15}$$

In the code, we define the integrating function:

$$f(z) = \frac{1}{(1-z)^5} \frac{z^3}{\exp \frac{z}{1-z} - 1} \tag{3.16}$$

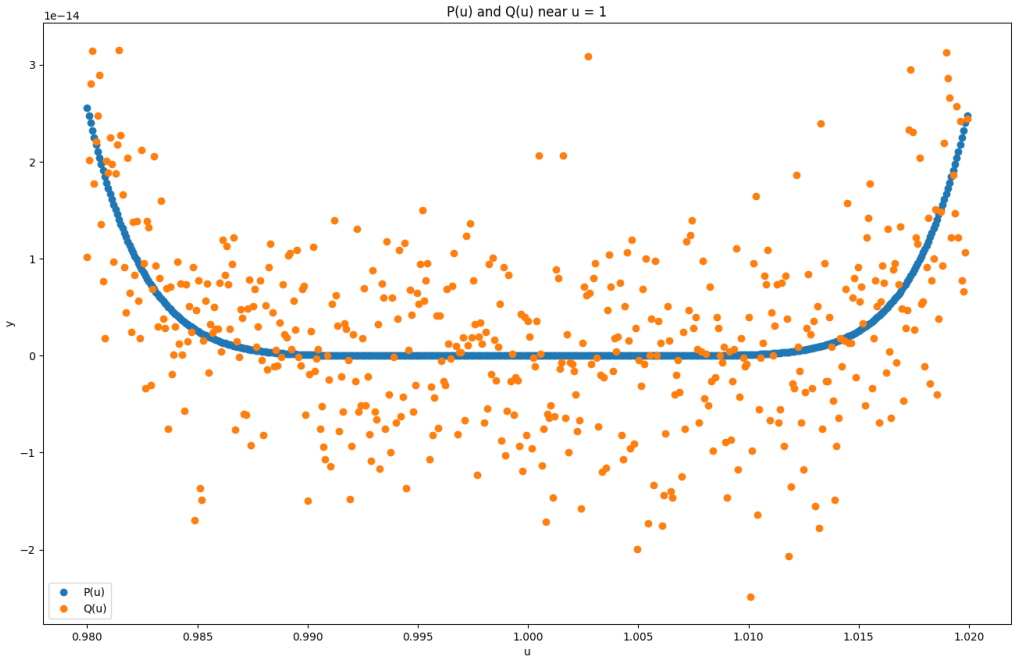
To simplify the coding, we calculate the integral itself using the trapezoid rule.

Pseudocode:
1. Setup the dataframe
2. Define the integral formula
3. Apply the numerical integration
4. Calculate the Stefan-Boltzmann constant
5. Define the function that takes T as input, calculate $W = \sigma T^4$ and outputs W

Notice that by using trapz, we encounter the case that we need the value of f(0) and f(1) which is not defined. Since we are calculating the area under the plot, the right side limit of f(0) and left side limit of f(1) should be used, which is zero for both cases.
We used the step number $n = 100$, which gives a $h = \frac{1}{100}$, therefore the error is in the magnitude of 10^{-4}.

C
The calculated $\sigma_c = 5.670374419508412 \times 10^{-8}$.
Compared to the scipy.constant.sigma $\sigma_{sp} = 5.670374419 \times 10^{-8}$, we have exceeded the 10^{-4} order accuracy.

Q4
A
We observed a scatter of points by using q(u) as the defining function, whereas p(u) performs very well as expected
This scattering of q(u) is likely due to rounding errors that limited by the computer's 64bits system. That is, every term will be saved to a 64bit number and added up together in the q(u) case which has a rounding error in order of 8 because of the 8 terms in the function. However, only save action is required for p(u) which only have rounding error in order 1
FIGURE 4.1



B

Figure 4.2 shows the scatter of p(u)-q(u) and figure 4.3 shows the distribution of the differential. The calculated standard deviation of the differential is $\sigma_c = 8.025\times10^{-15}$, and estimated error is $\sigma_{est} = 1.795\times10^{-29}$ according to the equation:

$$\sigma = C\sqrt{N}\sqrt{\overline{x^2}} \qquad 4.1$$

Where $C = 10^{-16}$ from the lab manual, $N = 500$ as the number of observations, x is the differential we calculated, which is squared, averaged and square rooted.

FIGURE 4.2

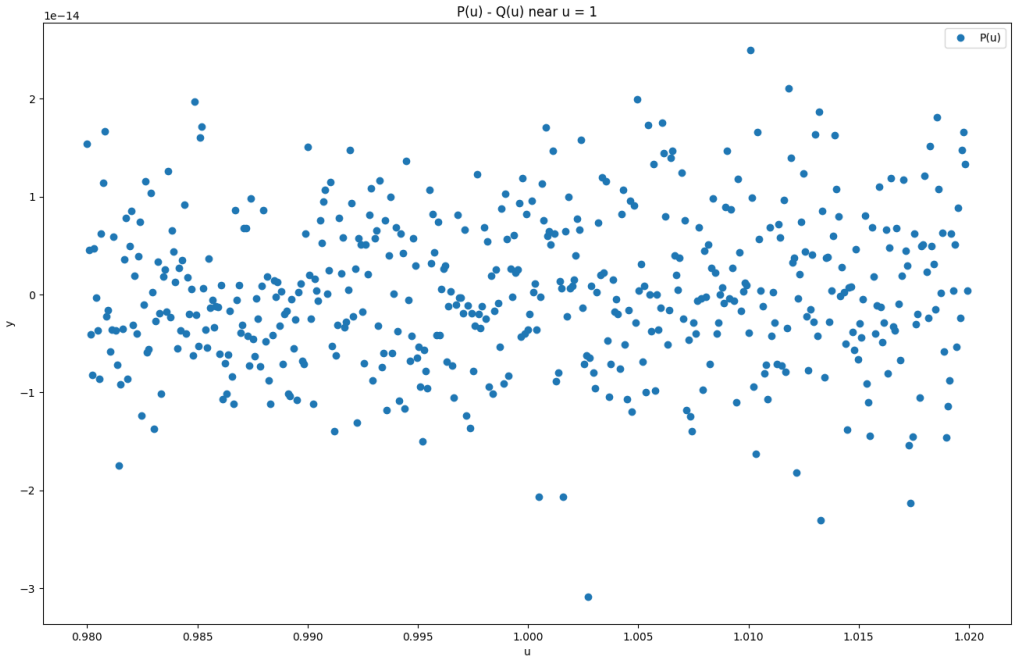
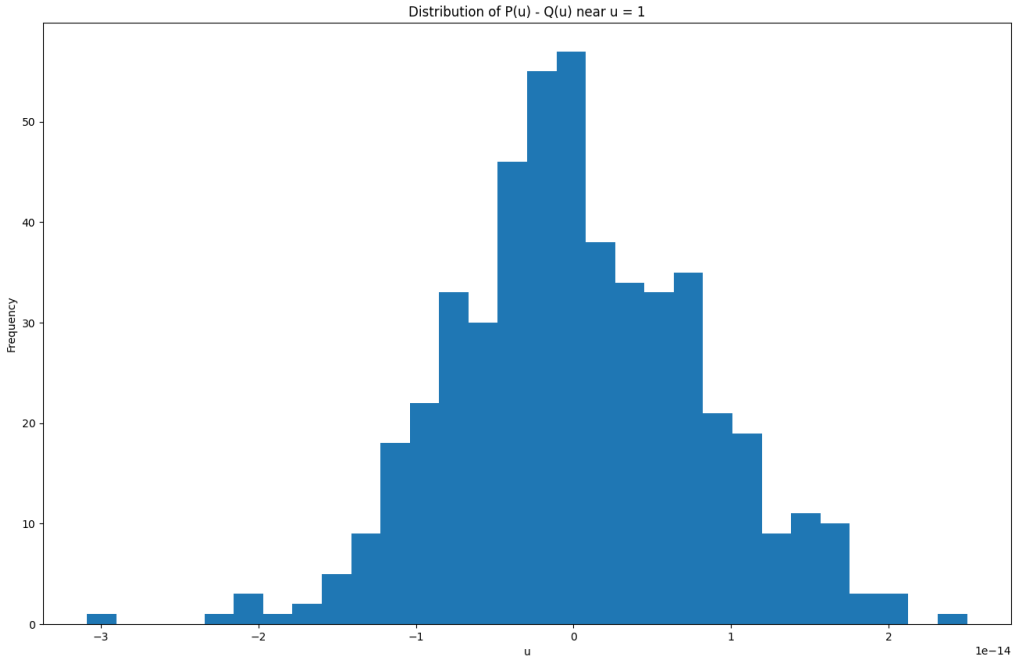


FIGURE 4.3



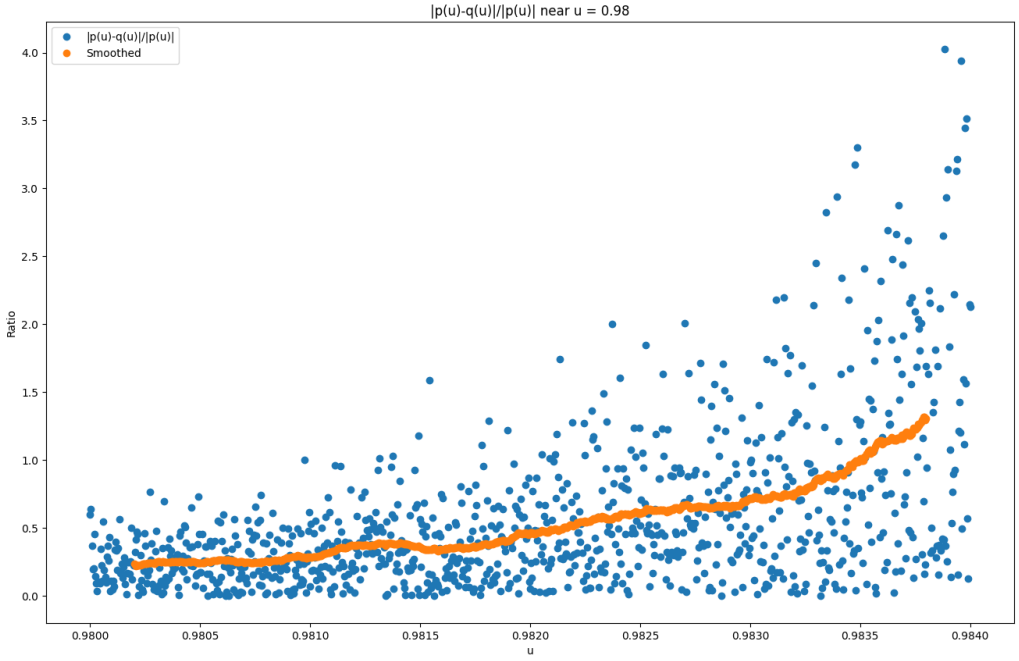
C

We did calculate a fractional error according to

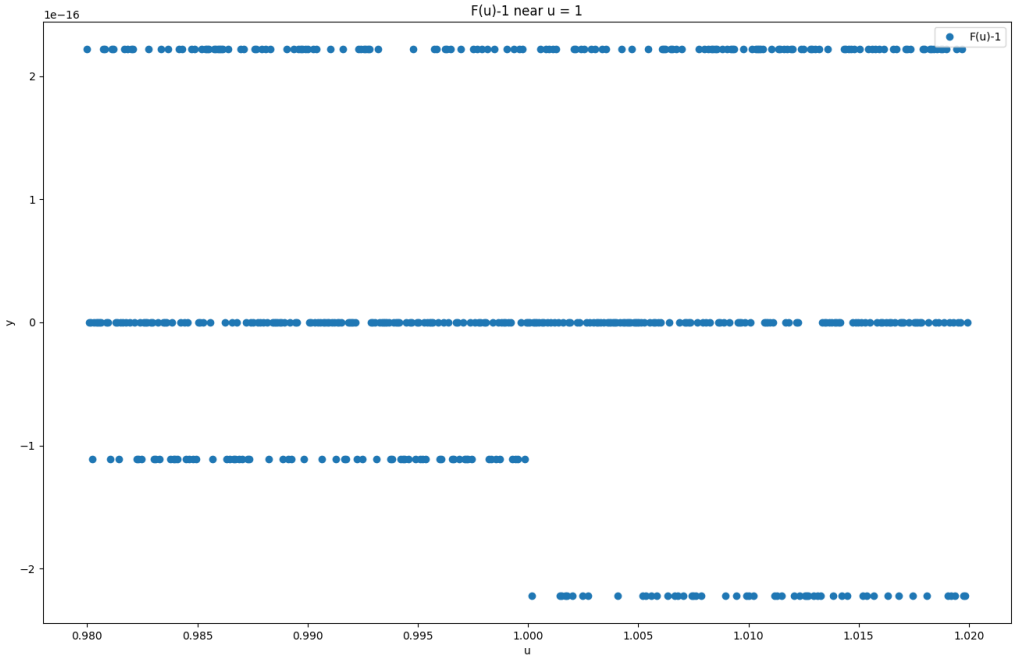
$$\frac{\sigma}{\sum x_i} = \frac{C}{\sqrt{N}} \frac{\sqrt{\overline{x^2}}}{\bar{x}} \qquad 4.2$$

Very close to 1 with $n = 150$ and $C = 1$.

Figure 4.4 shows the fractional errors in blue which is very scattered. After a smoothing shown in orange, we observed that $u \approx 0.98348$ gives a fractional error about 1.
FIGURE 4.4



D
Figure 4.5 shows the distribution of errors according to function f. We observed a quantized error with some smallest unknown units.
FIGURE 4.5


F(u)-1 near u = 1
1e−16
F(u)-1
2
1
0
−1
−2
y
0.980
0.985
0.990
0.995
1.000
1.005
1.010
1.015
1.020
u