Physics 20 - Numerical Integration

Shubh Agrawal, *Class of 2022*

October 15, 2018

Introduction

Several methods of computational numerical integration of basic functions are analyzed. Algorithm-centric Python programs are implemented for trapezoid approximation and Simpson's method; their error's dependence on contained variables is graphically studied through progressive simulations, and is used in improving code to provide values within specified error bounds. Predefined systems of integration, notably Romberg's method, are also taken up in comparison. Source code is available as attachment (`Assignment2.py`).

Extended Simpson's Formula

Here, the extended Simpson's formula for integration approximation is derived. The orders of the local and global errors are also derived.

The Simpson's rule is defined for one interval as follows:

$$I_{\text{simp}} \equiv H\Big(\frac{f(a)}{6} + \frac{4f(c)}{6} + \frac{f(b)}{6}\Big)$$

where $H = b - a$, and $c = (a+b)/2$. Writing $f(x)$ as a Taylor sum about a and integrating each side,

$$\begin{aligned} f(x) &= f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots \\ &= f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \frac{f''''(\eta)}{4!}(x-a)^4 \quad \text{with } x, \eta \in [a,b] \end{aligned}$$

Now,

$$I = f(a)H + f'(a)\frac{H^2}{2!} + f''(a)\frac{H^3}{3!} + f'''(a)\frac{H^4}{4!} + f''''(\eta)\frac{H^5}{5!}$$

Approximating using the Taylor series expression and putting $c - a = H/2$ and $b - a = H$,

$$\begin{aligned} I_{\text{simp}} &= \frac{f(a)}{6}H + \frac{4f(c)}{6}H + \frac{f(b)}{6}H \\ &= \frac{f(a)}{6}H + \Big(\frac{4f(a)H}{6} + f'(a)\frac{4H^2}{6\cdot 2} + \frac{f''(a)}{2!}\frac{4H^3}{6\cdot 4} + \frac{f'''(a)}{3!}\frac{4H^4}{6\cdot 8} + \frac{f''''(\eta)}{4!}\frac{4H^5}{6\cdot 16}\Big) \\ &\quad + \Big(\frac{f(a)H}{6} + \frac{f'(a)H^2}{6} + \frac{f''(a)H^3}{6\cdot 2!} + \frac{f'''(a)H^4}{6\cdot 3!} + \frac{f''''(\eta)H^5}{6\cdot 4!}\Big) \\ &= f(a)H + f'(a)\frac{H^2}{2!} + f''(a)\frac{H^3}{3!} + f'''(a)\frac{H^4}{4!} + f''''(\eta)\frac{5H^5}{24\cdot 4!} \end{aligned}$$

Then, $I - I_{\text{simp}} = f''''(\eta)\frac{H^5}{5!} - f''''(\eta)\frac{5H^5}{24\cdot 4!} \propto H^5$. Thus, it is derived that the local error under Simpson's Rule is of the fifth order, or $I = I_{\text{simp}} + O(H^5)$.

For deriving the *extended* Simpson's formula, the integral interval (a,b) is divided into N subintervals. However, it is noted that, in comparison to the trapezoid rule analysis, there are actually $2N$ divisions as each interval (a', b') is broken into two by c'. So, taking $h_N = (b-a)/N$, and $x_0 = a, x_1 = a + h_N/2, x_2 = a + h_N, \ldots, x_{2N} = b$,

$$\begin{aligned} \int_a^b f(x)\,dx &= \int_{x_0}^{x_2} f(x)\,dx + \int_{x_2}^{x_4} f(x)\,dx + \cdots + \int_{x_{2N-2}}^{x_{2N}} f(x)\,dx \\ &\simeq h_N\Big(\frac{f(x_0)}{6} + \frac{4f(x_1)}{6} + \frac{f(x_2)}{6}\Big) + \Big(\frac{f(x_2)}{6} + \frac{4f(x_3)}{6} + \frac{f(x_4)}{6}\Big) + \cdots + \Big(\frac{f(x_{2N-2})}{6} + \frac{4f(x_{2N-1})}{6} + \frac{f(x_{2N})}{6}\Big) \\ &= \frac{h_N}{6}\Big(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \cdots + 2f(x_{2N-2}) + 4f(x_{2N-1}) + f(x_{2N})\Big) \end{aligned}$$

The global error of the extended Simpson approximation would be

$$f''''(\eta')\frac{5h_N^5}{24\cdot 4!}N \propto h_N^5\frac{(b-a)}{h_N} \propto h_N^4 \qquad \text{with } \eta' \in [a,b]$$

Thus, the global error of the Simpson expression of the fourth order, or, $O(h_N^4)$.

Convergence rate for Trapezoid and Simpson's Formulas

A Python method was implemented to run models determining error bounds in integration approximations for either method. These models varied in the minimum and maximum subintervals studied. A overlaying graph was also plotted to not the difference of order between trapezoid (*second order*) and Simpson expressions (*fourth order*). The plots are plotted in log-log space to linearize the curve.

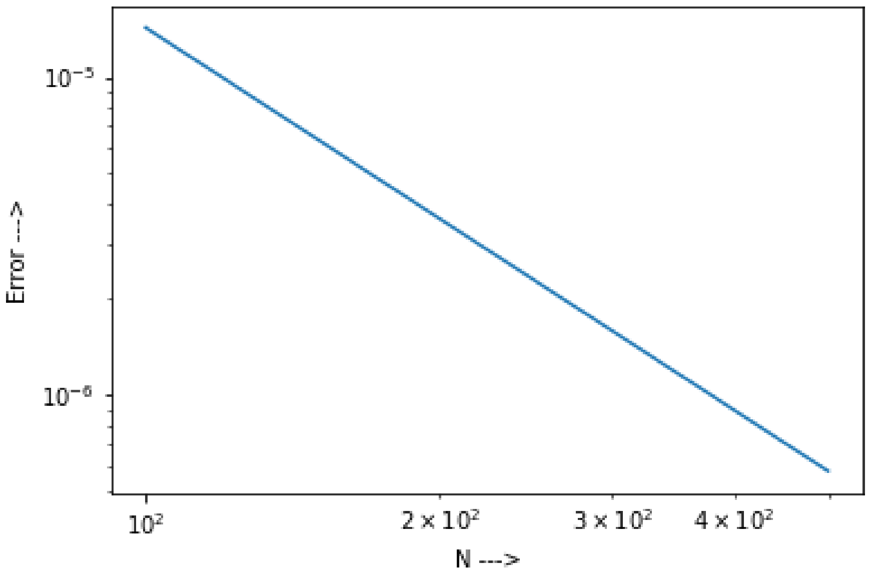

Figure 1: Convergence of error for trapezoid approximation of $\int_0^1 e^x$. Number of subintervals N ranges from 100 to 500.

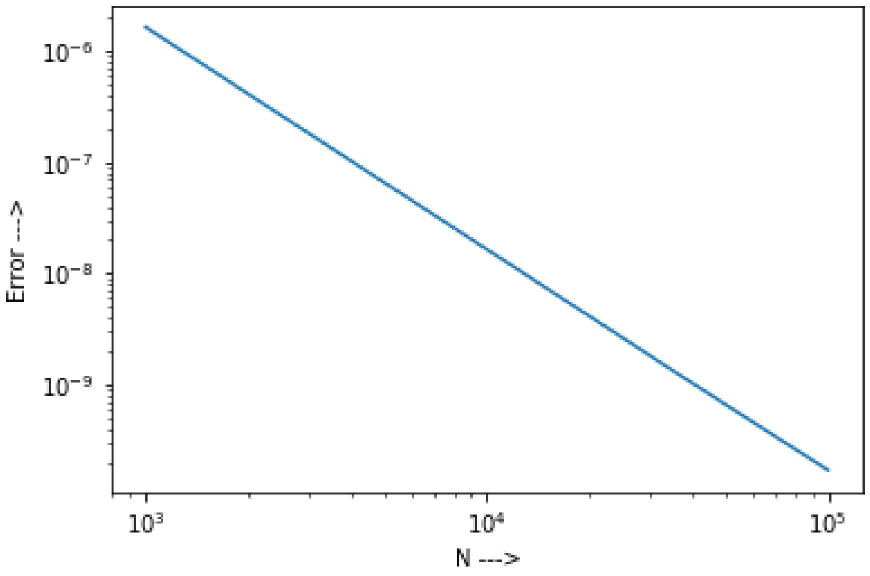

Figure 2: Convergence of error for trapezoid approximation of $\int_{-\pi/2}^{\pi/2} \cos x$. Number of subintervals N ranges from 1000 to 100000.

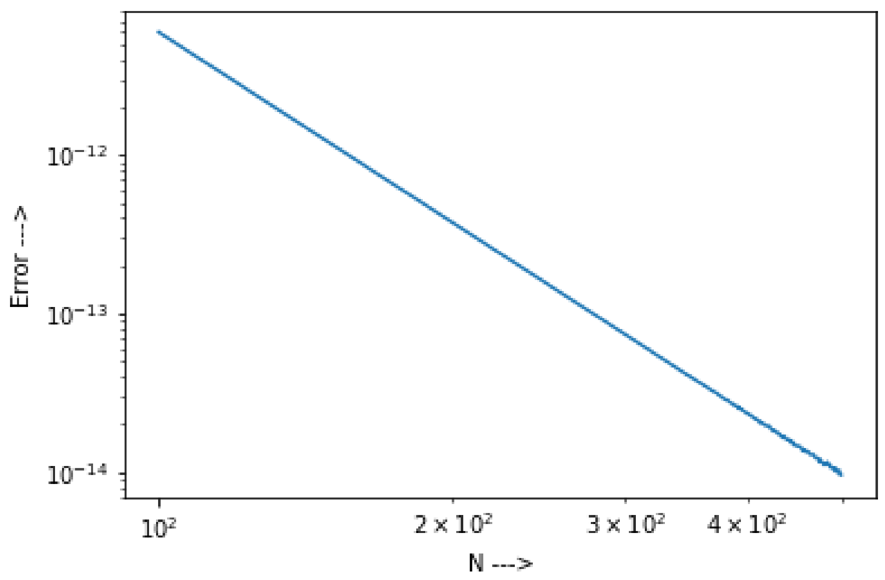

Figure 3: Convergence of error for Simpson approximation of $\int_0^1 e^x$. Number of subintervals N ranges from 100 to 500.

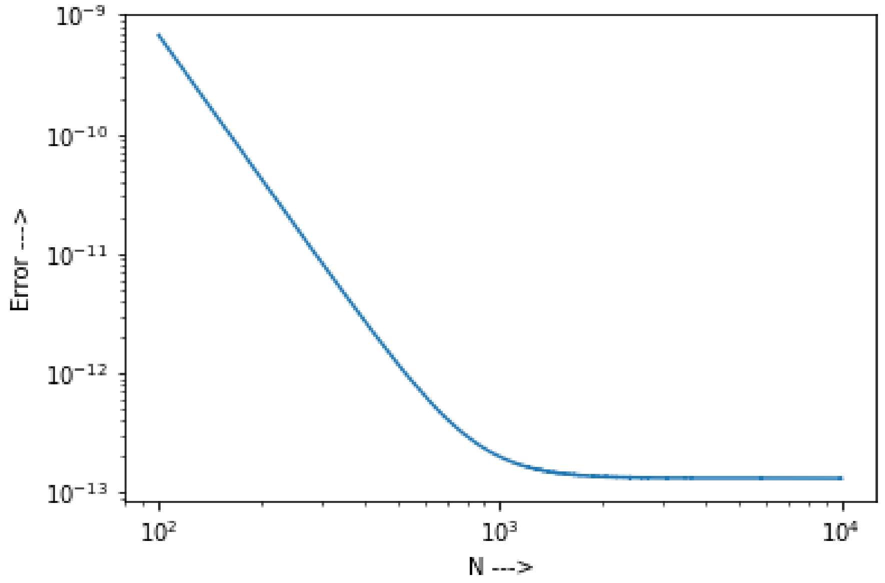

Figure 4: Convergence of error for Simpson approximation of $\int_0^\pi \sin x$. Number of subintervals N ranges from 100 to 10000. Note how the rate of error decrease seems to decay to constant error beyond third order N.

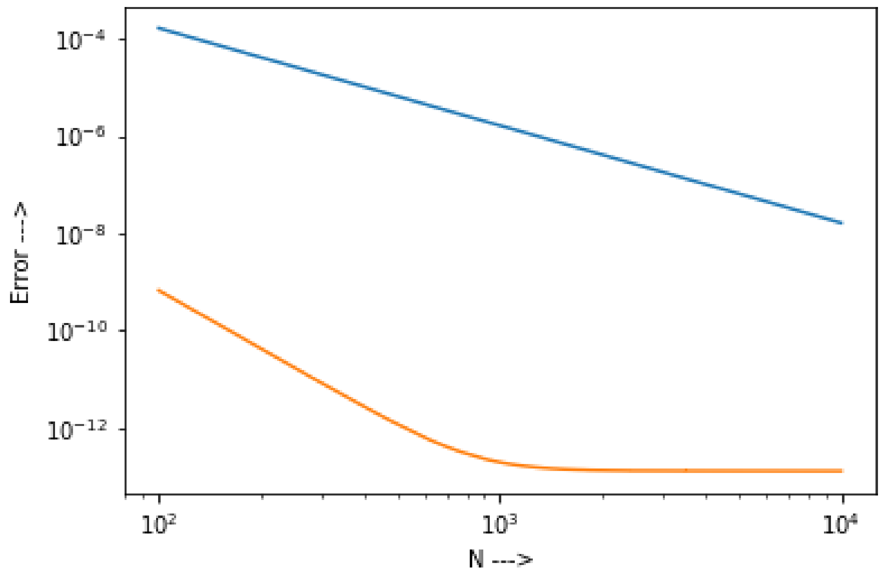

Figure 5: Legend: *orange* $\leftrightarrow$ Simpson; *blue* $\leftrightarrow$ Trapezoid.
Convergence of error for both approximations of integral of $\sin x$ from 0 to π. Number of subintervals N ranges from 100 to 10000. Note that the Simpson error bound is greatly smaller than the trapezoid error. Note how the magnitude of slope for Simpson is higher than that of the trapezoid method for the precise part, indicating that the former is a superior formula.

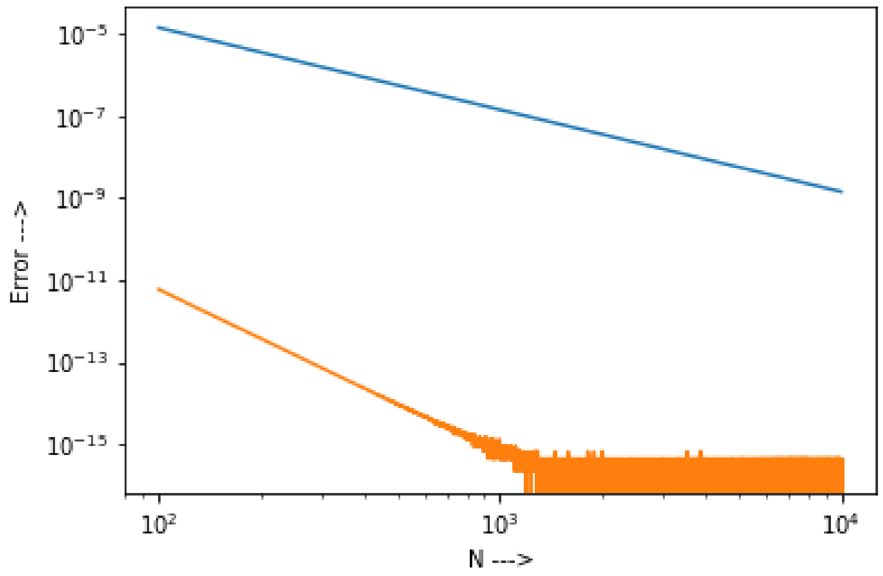

Figure 6: Legend: *orange* $\leftrightarrow$ Simpson; *blue* $\leftrightarrow$ Trapezoid.
Convergence of error for both approximations of integral of e^x from 0 to 1. Number of subintervals N ranges from 1000 to 100000. Note how the floating point imprecision is even more evident when the integral value is a bounding value too $(e - 1)$.

In Figures 4, 5, and 6, it is noted that the Simpson error seems to not decrease beyond a certain number of simulations. Floating point precision limits (the innate storage precision of `float` values in computer memory) do not allow greater accuracy, or simply put, the unit cannot measure the returned values of the function accurately to say anything conclusive. It is also notable that this is only observed in our Simpson plots at high N, as the trapezoid method gives several order of magnitudes (5) larger errors, which do not reach the limits of memory precision. The limit might be reachable by increasing the order of magnitude of N by the said order (5), which would take high computational power and time complexity, and is not therefore demonstrated here.

Comparison with `scipy.integrate` module

Similar analysis was carried out on the `quad` and `romberg` functions of the `scipy.integrate` module. Note that these two do not take number of subintervals as arguments and, thus, give a constant error bound (represented as straight lines parallel to the Y-axis on the graphs below) for all N.

The integral $\int_0^1 e^x$ is analyzed in all cases; wherever applicable, the number of divisions are taken fairly large to get lower error scales.

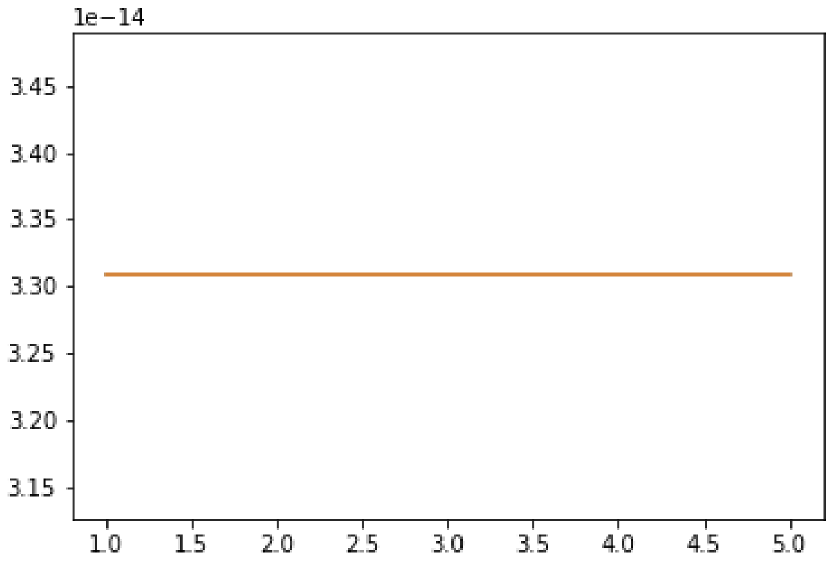

Figure 7: Romberg method error order (in linear space)

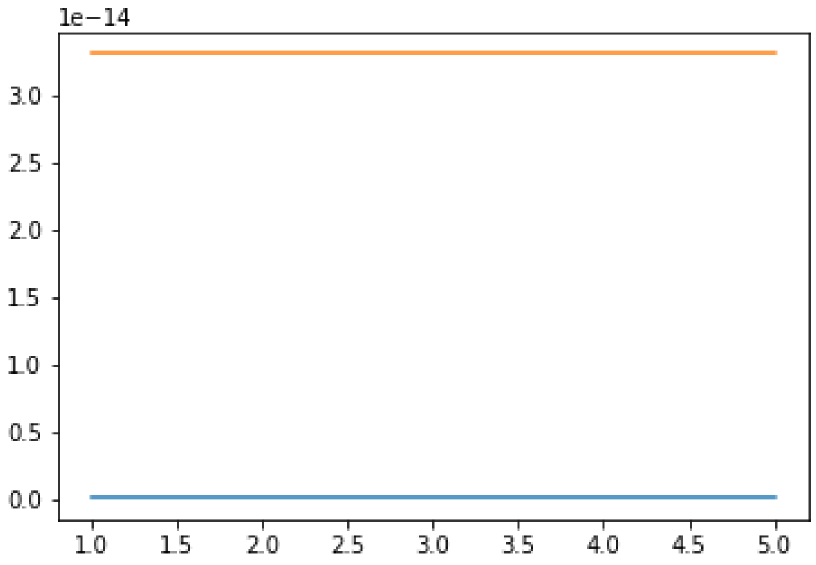

Figure 8: `romberg` (*orange*) method versus `quad` (*blue*) method (in linear space)

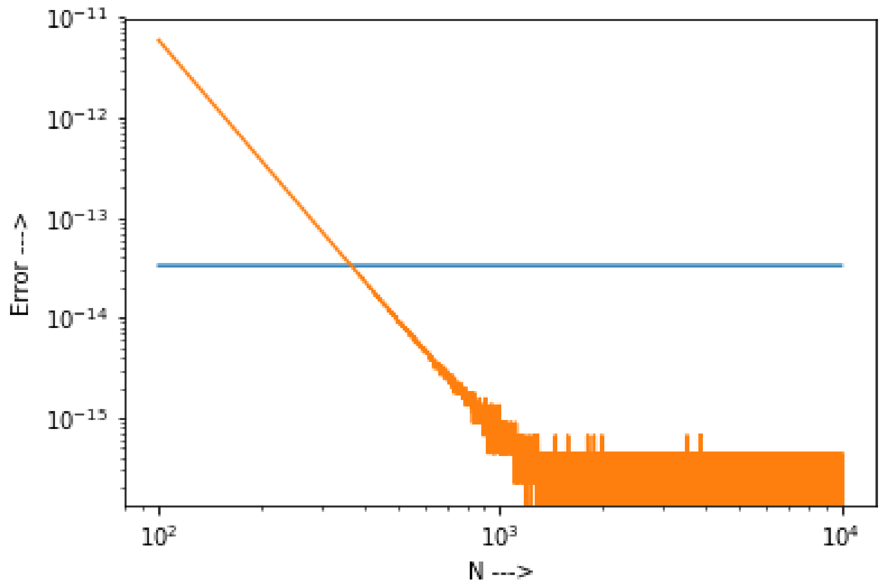

Figure 9: `romberg` method (*blue*) versus Simpson's formula (*orange*) (in logarithmic space), for N from 100 to 10000. Note that Simpson's error does (seem) to go below the romberg error, but, due to floating precision, it is irregular and imprecise.

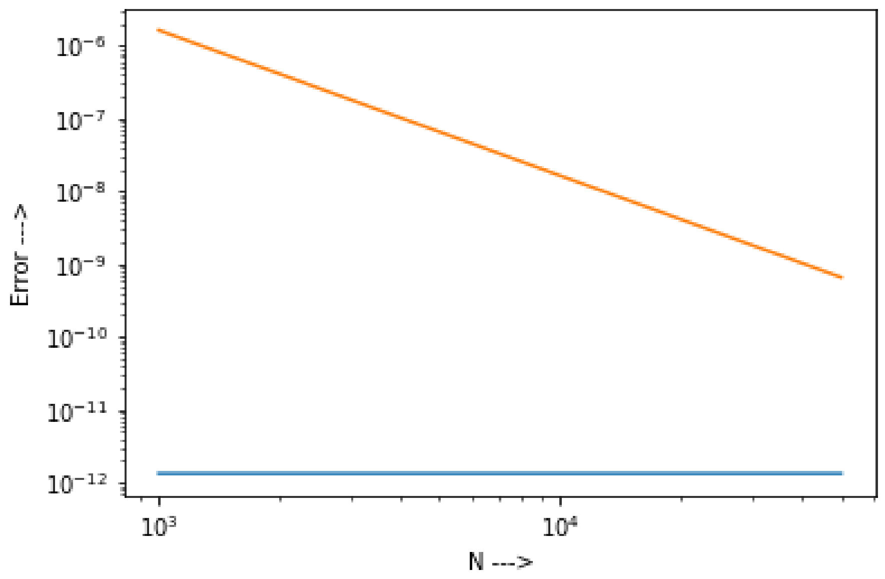

Figure 10: `romberg` method (*blue*) versus Simpson's formula (*orange*), for N from 100 to 5000, for integral of $\sin x$ over 0 to π. Note how the Simpson error does not oscillate when the actual value of integral is not a bounding (irrational-like) float value. (Here it is simply 2.0)

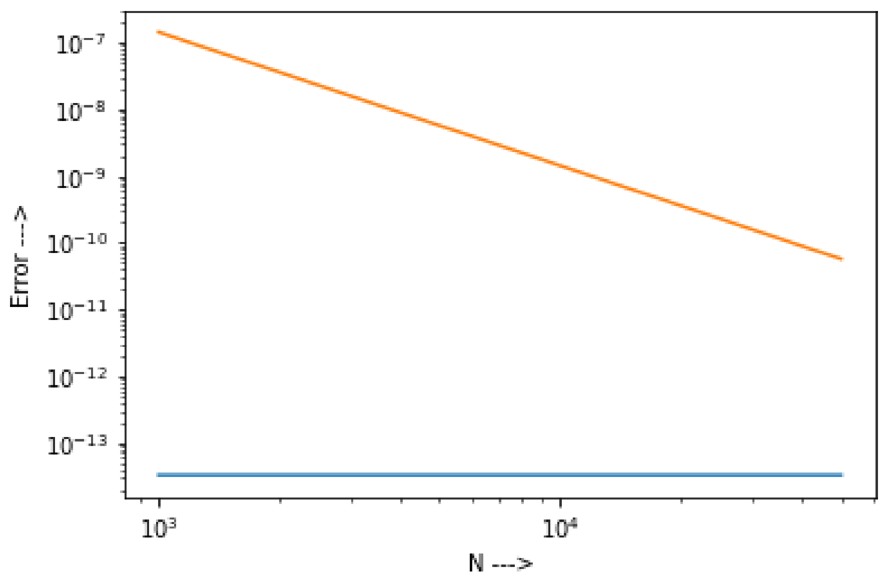

Figure 11: `romberg` (*blue*) method versus trapezoid formula (*orange*), range of N from 1000 till 50000. Note the large difference in order of magnitudes.

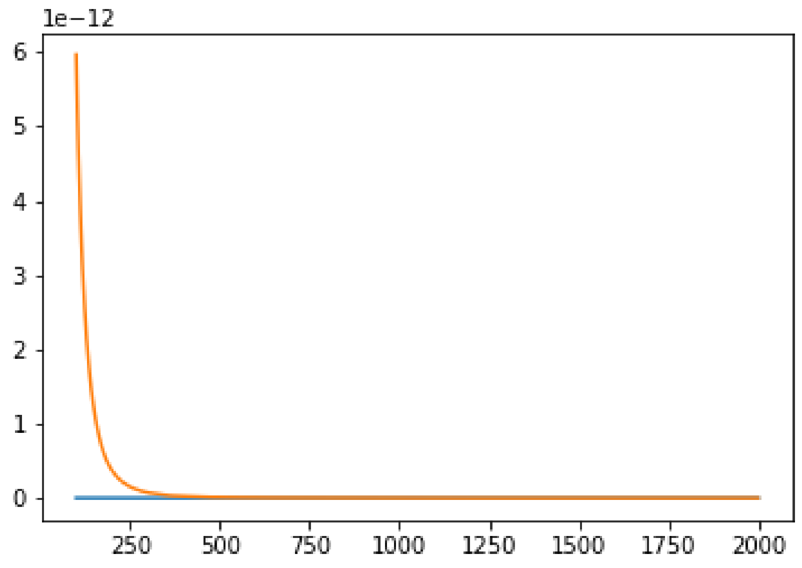

Figure 12: `quad` method (*blue*) versus Simpson's formula (*orange*), for N from 200 to 2000, in linear space, as quad errors are not always positive.

It is noted that, as expected, the `scipy.integrate` methods approximate the integral more accurately and faster (less memory manipulated) for practical purposes. Increasing N does cause the Simpson error to converge towards 0, yet due to floating precision, it is ineffective as compared to `quad` as all values, and `romberg` at lower values of N (which are usually used in fast computation).