HW# 3

Aaron Segura
Frederick Lee

October 2, 2018

1 Task 1: Trapezoidal Rule

1.1 What

The trapezoidal rule belongs to the class of Newton-Cotes quadrature rules which approximate integrals using equidistant grids. The term quadrature is used because we will be approximating the integral by using the area created by the equidistant grids that take the form of a quadrilateral or four-sided figure. By definition, for a set of equidistant grid points $x_i = X_L + ih, i = 0, 1, ..., nh = \frac{X_R - X_L}{n}$ on the interval $[X_R, X_L]$, the Trapezoidal rule is as follows:

$$\int_{X_L}^{X_R} f(x)\,dx \approx h\left(\frac{f(x_0) + f(x_n)}{2} + \sum_{i=1}^{n-1} f(x_i)\right) \tag{1}$$

Here we will write a Python program that uses the trapezoidal rule to approximate the definite integral:

$$\int_{-1}^{1} e^{cos(kx)}\,dx \tag{2}$$

where $k = \pi$ and $k = \pi^2$ for $n = 2, 3, ..., N$ with n being the number of intervals used to approximate the integral and N being the max number of intervals within a desired approximation tolerance. In other words, we used a value of N such that the absolute error estimate

$$(\delta_{abs})_{n+1} = |I_{n+1} - I_n|, \tag{3}$$

is less than 10^{-10}. In our report we plot the error obtained using the method against n using a logarithmic scale for both axes.

Again, this method is known as the Trapezoidal rule because $\int_a^b f(x)\,dx$ is being approximated by $f(x)$ as it is evaluated at values on the limits of the interval multiplied by the interval width which would be proportional to the area of a trapezoid. In other words, the integral $\int_a^b f(x)\,dx$ can be approximated by approximating the area under its curve with the area of a trapezoid as shown in Figure 1 below.

In this task, we will use a composite of the trapezoidal rule such that the trapezoidal rule integral approximation I_n has n intervals of width h, where $h = \frac{b-a}{n}$. We then compare this approximation of the integral to the next trapezoidal rule integral approximation I_{n+1} that has h of smaller size and $n+1$ intervals. Iterations of this process will be implemented until the difference between these two integral approximations defined in (2) is within a user defined tolerance. This procedure will be used for both cases where $k=\pi$ and $k=\pi^2$.

In this case, using a composite Newton-Cotes method such as the Trapezoidal rule, the order of convergence rate is typically s or $s+1$, where s is the number of points in each interpolating spline. Here, $s=2$ for the trapezoidal rule since we have two points for each spline. Additionally, the trapezoidal rule is a linearly convergent approximation of the definite integral as we increase the size of n intervals decreasing h.

1.2 How

To calculate the approximation of the integral, we define a function *int_trap* which has inputs f, a, b, and n which will approximate the integral using the Trapezoidal Rule.

$$int\_trap(f(x), a, b, n) = \left(\frac{h}{2}\right)\left(f(a) + 2\sum_{k=1}^{N-1} f(x_k) + f(b)\right) \tag{4}$$

where $h=(b-a)/n$, as stated previously, and $x_k = a + kh$.

Here, $f(x)$ is the integrand in the integral being approximated by the Trapezoidal Rule with n being a specified partition count, and the lower and upper limits of integration being a and b, respectively. Note that given n partitions, there will be a total of $n+1$ function evaluations when approximating the integral using the Trapezoidal rule. This fact can be used for comparison purposes when evaluating the time efficiency between two different methods of integral approximations. We will do this in a later task by comparing the number of n used to best approximate the convergence rates within the desired tolerance for each k (discussed in Task 3 in relation to the two quadrature methods: Trapezoidal and Gaussian).

Finding the minimal partition count for each k value was accomplished through an iterative process in which the increasing values of n are used as inputs in the int_trap function. The condition checked is whether the given absolute difference, calculated using equation (3), is within the given tolerance 10^{-10}. When this condition is satisfied, the iteration process through which increasing n values are generated halts and the minimal satisfying n, along with the error calculated using equation (3), is printed to the console.

Below, we provide a visualization of the trapezoidal integration given the minimal number of partitions n needed to achieve an absolute error value within the given tolerance 10^{-10}.

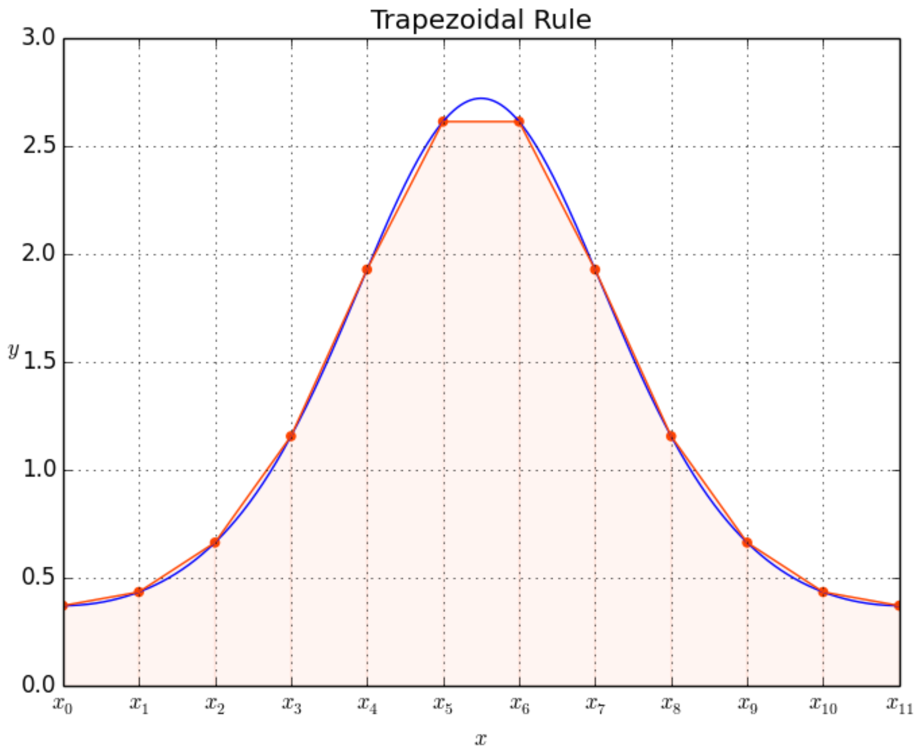

Figure 1: Visualization of Trapezoidal Rule applied to $f(x)$ for $k = \pi$

As you can see, the Trapezoidal Rule roughly approximates the area under the curve of our function within the integral (2) when $k = \pi$ which can be interpreted as an approximation of the definite integral. A Trapezoidal Rule plot of $f(x)$ when $k = \pi^2$ was not included here because the number of n intervals was so great that an accurate depiction was not possible since the number of n intervals is well over 2000. Below, we show the error plot for both k values at each n including several referential convergence curves which will be used for comparison.

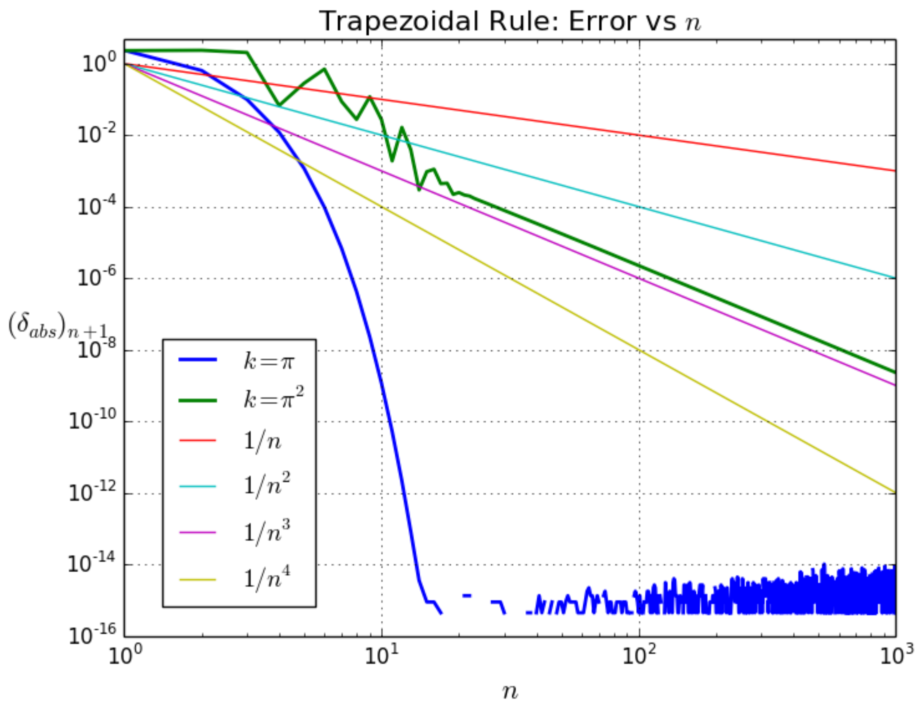

Figure 2: Plot of $(\delta_{abs})_{n+1}$ for $f(x)$ with respect to n (Trapezoidal Rule)

1.3 Why

The Newton-Cotes composite Trapezoidal rule is a useful numerical method because there are cases where we need to evaluate the definite integral of a function but that function may have an antiderivative that is difficult to compute analytically or does not have an explicit antiderivative. In other words, the definite integral may not be easily attained in that it may be computationally intensive or nearly impossible to compute the integral. This is the case for our function $f(x) = e^{cos(kx)}$ for both when $k = \pi$ and $k = \pi^2$, in that we cannot analytically derive an antiderivative for this function. Another reason to use numerical methods to compute a definite integral would be that there may be an integrand $f(x)$ that is known for a certain number of points obtained by a sampling process and the embedded system or computer application may need a numerical integration method to approximate the definite integral of the data.

Here, we have shown that it is possible to use a composite of the Newton-Cotes Trapezoidal rule to approximate the definite integral $\int_{-1}^{1} e^{cos(kx)}\, dx$.

We found that the Trapezoidal rule does so differently for different values of k, where $k = \pi$ takes fewer N intervals to approximate the integral than $k = \pi^2$. The approximation of the definite integral in the case of $k = \pi$, converges faster than expected relative to the integral approximation of $k = \pi^2$ because of the periodicity of the function when $k = \pi$ is the same magnitude as our limits of integration $b - a$ which is 2. This is explained in further detail below.

As stated previously, we found something that is quite interesting. We obtained a higher-order accurate approximation of the integral when $k = \pi$ in that it took fewer partitions n to achieve the desired absolute error tolerance level. This can be explain by the Euler-Maclaurin Expansion and the fact that our integrand is a periodic function, whose period is 2 when $k = \pi$ and our interval length, $b - a$, is also 2 in the case where $a = -1$, $b = 1$. This is important because when using the Euler-Maclaurin Expansion terms cancel in such a way that when applying Richardson Extrapolation repeatedly for the composite Trapezoidal Rule we get a higher order accurate approximation. Here, we express the Euler-Maclaurin Expansion as follows:

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n}[f(x_{i-1}) + f(x_i)] + f(b)\right] + \sum_{r=1}^{k} c_r h^{2r}[f^{(2r-1)}(b) - f^{(2r-1)}(a)] - \left(\frac{h}{2}\right)^{2k} \sum_{r=1}^{k} \int_{x_{i-1}}^{x_i} q_{2k}(t) f^{(2k)}(x)\,dx \tag{5}$$

where for each $i = 1, 2, ..., n, t = -1 + \frac{2}{h}(x - x_{i-1})$, and the constants

$$c_r = \frac{q_{2r}(1)}{2^{2r}} = -\frac{B_{2r}}{(2r)!}, r = 1, 2, ..., k \tag{6}$$

are closely related to the *Bernoulli numbers.* It can be seen the from this expansion that the error $E_{trap}(h)$ in the composite Trapezoid Rule, like the centered difference approximation of the derivative, has the form, :

$$E_{trap}(h) = K_1 h^2 + K_2 h^4 + K_3 h^6 + ... + O(h^{2k}) \tag{7}$$

where the constants K_i are independent of h, provided that the integrand is at least $2k$ times continuously differentiable. So when applying Richardson Extrapolation repeatedly using the composite Trapezoidal Rule at different spacing, we can obtain a higher order accurate approximation of the integral.

Note that Richardson extrapolation is a method where in which we can use some $F(h)$ and $F(h/q)$, with $q > 0$ to approximate $F(0)$ with higher order accuracy with $F(h)$ taking the form of $F(h) = a_0 + a_1 h^p + O(h^r), r > p$. The exact value of $F(0)$ being a_0 and we choose of value for h and compute $F(h)$ and $F(h/2)$ for some positive integer q then we get the $O(h^r)$ and solve a system of two equations for unknowns a_0 and a_1 to obtain an approximation that is r-th in order of accuracy. Then if we described the error in the approximation in the same then we can describe the error of our original approximation $F(h)$ and repeat this process to obtain a more accurate approximation since our function $f(x) = e^{cos(kx)}$ when $k = \pi$ has a period of $b - a$ in our case $a = -1$ and $b = 1$

So, from the Euler-Maclaurin Expansion in relation to the composite Trapezoidal Rule applied here, we obtain the higher-order accurate approximations of the integral because the terms involving the derivatives of the integrand at a and b cancel out with the Euler-Maclaurin Expansion becoming:

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n}[f(x_{i-1}) + f(x_i)] + f(b)\right] + \left(\frac{h}{2}\right)^{2k} \sum_{r=1}^{k} \int_{x_{i-1}}^{x_i} q_{2k}(t) f^{(2k)}(x)\,dx \tag{8}$$

and as a result increases our convergence to approximate the definite integral. In other words, when the integrand is a periodic function and the limits of integration form a period $b - a$ of the integrand, this causes the terms involving the derivatives of the integrand at a and b to disappear when $f(x)$ is $2k$ times differentiable as shown in equation (8) above. The composite Trapezoidal Rule approximation with spacing h becomes $O(h^{2k})$ rather than $O(h^2)$. So, it follows that if $f(x)$ is infinitely differentiable, such as in this case as a finite linear combination of sines and cosines, then this method will have an exponential order of accuracy as h approaches zero and the error will converge to zero more rapidly than any power of h^p for $p > 0$. This is in agreement with our findings in Figure 2 where we plot $(\delta_{abs})_{n+1}$ for $f(x)$ with respect to n.

From the error plot above, we also can see that the rate of convergence is slightly below the order of $O(h^3)$ for our function $f(x)$ when $k = \pi^2$. Since the limits of integration of this function do not form a period of the function, the observed exponential convergence when $k = \pi$ does not apply. So we can then say that the Trapezoidal rule has an error convergence rate on the order of $O(h^3)$ for periodic functions whose integration limits are not a period of the function.

2 Task 2: Gauss Quadrature

2.1 What

According to the Gaussian quadrature rule, we can approximate the definite integral of a function by using a weighted sum of function values at specified points or nodes within the domain being approximated $[a, b]$. In Newton-Cotes quadrature rules, it assumed that the value of the integrand is known at equally spaced points. However, with Gaussian quadrature, the location of the grid-points x_i, referred to as quadrature nodes, are not equally spaced. In Gaussian quadrature, we will evaluate the function $f(x)$ for each of these quadrature nodes and use quadrature weights w_i which are chosen such that the order of the approximation to the weighted integral will be maximized. Therefore, by summing the product of the function evaluated at the quadrature nodes and the quadrature weights for each node, we can accurately approximate the definite integral. This can expressed by the following equation for Gaussian Quadrature method:

$$\int_{-1}^{1} f(x)w(x)\,dx \approx \sum_{i=0}^{n} f(x_i)w_i \tag{9}$$

Here, the weight function $w(x)$ is assumed to be positive and integrable and the optimal nodes x_i turn out to be the roots of orthogonal polynomials associated with the weighted function $w(x)$. For the case $w(x) = 1$, we will use Legendre polynomials and obtain the weights and nodes by calling the function *lglnodes(n)*. We will use the function located in the *homeworks/hw3/lglnode.py* from the course repository to compute the integral.

We then plot the error against n using a logarithmic scale for both axes. For Gauss quadrature the error decreases as $\epsilon(n) \sim C^{-\alpha n}$.

2.2 How

To implement the Gaussian Quadrature, we create a *for* loop for i in range of the length of our function list with a nested inner loop having a conditional statement of N in range from 1 to the max number of $n + 1$. Using the *lglnodes* function from the course repository, we used quadrature nodes and weights to approximate the integral of our two functions in our function list. The approximation of the integral was found by summing values of the function evaluated at each node multiplied by the quadrature weights for each node.

Here, we note that given an input of n, the function *lglnodes* will return a tuple of nodes and weights, each of size $n+1$. These nodes are the inputs for the given function that will be integrated, meaning that a call to *lglnodes* with input n will result in an integral approximation that involves $n + 1$ function evaluations. In other words, using the *lglnodes* function with n as our input we will generate $n+1$ number of nodes and weights and use a total of $n + 1$ function evaluations in the process.

Included below is a visualization of Gaussian Quadrature for our function $f(x)$ when $k = \pi$. For this plot, we used two color associated scales on the vertical axes. The blue curve represents our function and is associated with the left vertical axis. The black vertical lines represent the function evaluated at each Legendre-Gauss-Lobatto node returned by calling the *lglnodes* function input of $n = 19$ and is associated with right right axis. Here, n is the minimal value that results in an absolute error value less than tolerance 10^{-10}. Note that the height of each vertical line is determined by multiplying each function evaluation by their associated quadrature weights. The sum of each vertical line length approximates the definite integral.

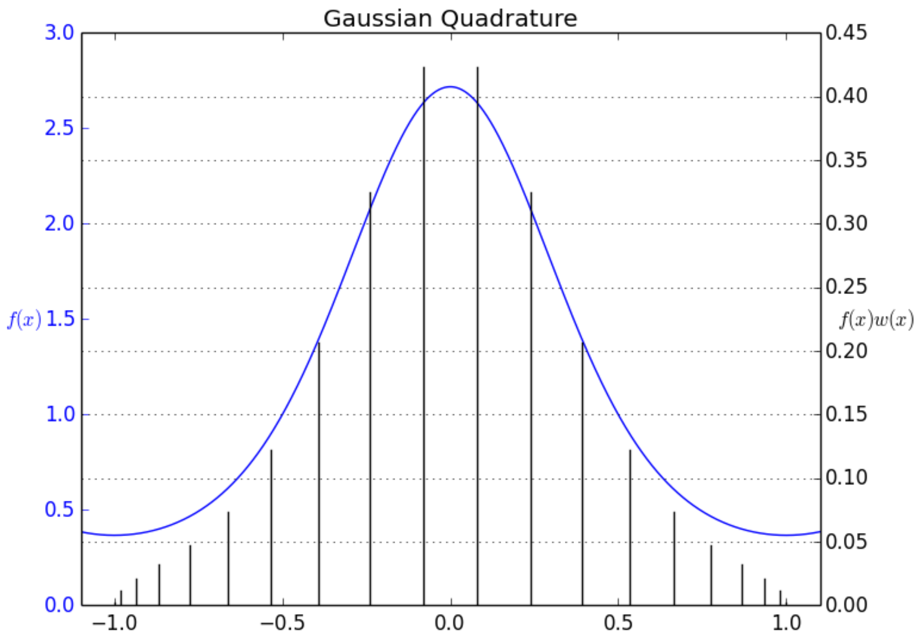

Figure 3: Visualization of Gaussian Quadrature

Below, we plot our error using equation (3) for every n. Additionally, for each k value, we report the smallest value of n that resulted in an error less than the given tolerance 10^{-10}. We can see that as n increases the error values for each function decreases and approaches zero. Here, we also include guiding convergence curves for the functions 2^{4-n}, $2^{2(4-n)}$, and $2^{3(4-n)}$ which can be used for comparison for each function where $k = \pi$ and $k = \pi^2$.

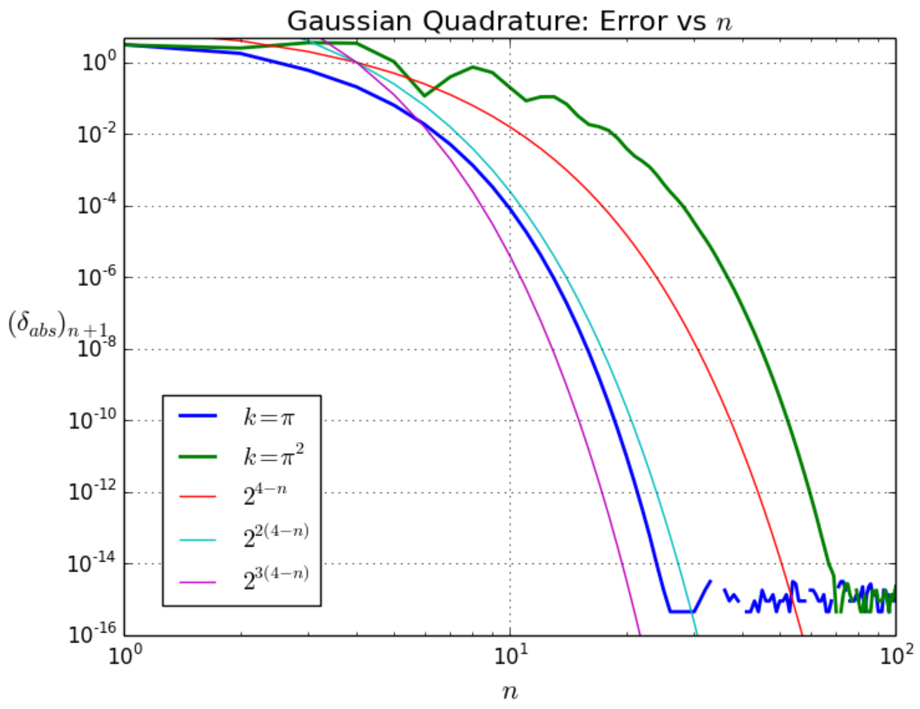

Figure 4: Plot of $(\delta_{abs})_{n+1}$ with respect to n (Gaussian Quadrature)

2.3 Why

From the Gaussian Quadrature error vs n plot above, we can see that the rate of convergence for $k = \pi$ is very close to $2^{2(4-n)}$ which would be of the order $O(C^{-\alpha n})$, where $C = 4$ and $\alpha = 1$. We effectively approximate the definite integral of the function $f(x)$ to less than our user defined tolerance as n increases. Here, we are using Legendre-Gauss-Lobatto(LGL) nodes and weights as well as the LGL Vandermonde matrix while we call the function *lglnodes* in python. The LGL nodes are the zeros of $(1 - x^2)P'_N(x)$ and are useful for numerical integration and spectral methods.

In our visualization of Gaussian Quadrature, we see that the nodes are not multiplied by the same weighted values and are not evenly spaced along the interval $[-1, 1]$. This effectively illustrates how Gaussian Quadrature can be used to approximate a definite integral i.e. by using variable weights appiled to unevenly spaced grid points. It is important to note that the LGL nodes are generated by using the specified points whose derivation is beyond the scope of this assignment but provide great accuracy when used to approximate a definite integral.

It can be observed from the plot above that Gauss Quadrature seems to be exponentially convergent for both values of k in the function (2), apparently outclassing Trapezoidal integration in terms of convergence rates.

3 Task 3: Cost Comparison

3.1 What

In numerical computing it is beneficial to choose the cheapest method for a given task to save computational power and time for the user. Here, we compare the computational cost of Gauss quadrature and the trapezoidal method. One way to count the cost of a method is to find how many function evaluations are required for the quadrature result to be within a given error tolerance, in this case 10^{-10}. We will then determine the number of evaluations for both methods regarding our two cases $k = \pi$ and $k = \pi^2$ concluding which rule is likely to be more efficient in a general setting.

3.2 How

In the tabulation below, we show the minimum number of required function evaluations in order to obtain an error within our desired tolerance level of 10^{-10} using (3) for each k value with respect to both quadrature methods.

Method	$k = \pi$	$k = \pi^2$
Trapezoid	12	2843
Gaussian	20	54

Table 1: Comparison between Trapezoidal rule and Gaussian Method

3.3 Why

By comparing methods of approximating an integral in number of function evaluations required to achieve a given error tolerance, we can then select the more time efficient method of approximation saving the computational power required to complete such a task. This will benefit the user by reducing the amount of resources used to accomplish this task within a desired tolerance or error level.

In comparing the time efficiency of these methods, we use the number of required function evaluations as the main criterion. The table above seems to suggest that, in general, Gaussian quadrature is a more time efficient quadrature strategy, though the special case of integrating a period of a periodic function seems to favor trapezoidal integration. Empirically, it is not clear how advantageous Gaussian quadrature can be in a general setting from testing two functions of the same general form. Additionally, we cannot conclude the order of convergence for non-periodic functions in the general sense in regards to using Trapezoidal rule.

These numerical methods are useful because there are cases where we need to evaluate a definite integral of a function computationally but are unable to do so since the function may not have an antiderivative or is difficult to compute. Here we have shown a case where numerical methods can achieve high accuracy with low relative computational cost. Some topics to explore in the future would include comparing other quadrature strategies and applying quadrature to functions of multiple variables, as well as integrating other types of single variable functions, including non-periodic functions.

References

[1] http://www.math.usm.edu/lambers/mat772/fall10/lecture9.pdf