Project 02

APSC 607 Fall 2017

Seth Goodman

October 27, 2017

1 Introduction

This project explores different numerical methods for calculating the definite integrals of functions. Two functions will be examined in the range between zero and two, using the Composite Trapezoidal Rule, Composite Midpoint Rule, Composite Simpson's Rule, as well as an adaptive implementation of Simpson's Rule. The behavior and characteristics of these methods will be reviewed by examining their accuracy using varying number of subintervals, **n**. For each **n**, the result will be evaluated with respect to the true integral value, using tolerance thresholds to determine sufficient levels of accuracy.

All computations are performed using MATLAB using the code accompanying this report. Section 2 will present the methods used in MATLAB to explore functions, as well as the outputs and results. Section 3 section of this report contains results and related outputs for each function along with observations and discussion. All figures and tables found in this report are available in the output subdirectory of the accompanying zip file. Additionally, all code and figures found in the zip file can be accessed via GitHub[1].

[1] `https://github.com/sgoodm/apsc607/tree/master/project_02`

2 Methods

The two unique functions which will be explored in this project, Functions **A** and **B**, are defined by Equations 1 and 2, respectively.

$$f(x) = e^{2x} * sin(3x) \tag{1}$$

$$f(x) = \frac{1}{x+4} \tag{2}$$

To establish a baseline for error comparisons, the true value for the integral of each function is first calculated using built in MATLAB tools. The integral is calculated both using the symbolic toolkit function **int** as well as the numerical function **integral**. The resulting values can be seen in Table 1.

Function	**Symbolic**	**Numeric**
A	-14.2139771298625	-14.2139771298625
B	0.405465108108164	0.405465108108164

Table 1: True values of integrals between zero and two

The true value for Functions A and B will be compared to the results of integration using the Trapezoidal, Midpoint, and Simpson's Rules for a range of subintervals. The range of subinterval values, **n**, will vary with each function and rule tested, in order to achieve accuracy within a tolerance of $1e^{-4}$ and $1e^{-8}$.

The tolerance is defined as the absolute difference between the result of numerical integration at a given **n** value and the true value of the integral. An alternative approach for measuring tolerance is to compare the difference between the integral resulting from subsequent values of **n**. Both methods will effectively illustrate the performance of the different methods of numerical integration, but comparing to the true value provides a independent metric of accuracy, rather than one which is dependent on the step size between values of **n**. Since the accuracy tolerance will not be based on subsequent values of **n**, a sample of values for **n** can be used rather than the full range. This will drastically improve the performance of the MATLAB calculations without sacrificing noticeable detail from the results. Figure 1 illustrates an example of the sampling points used for **n** ranging between 2 and 100,000. As a tradeoff for reduced computation time, the apparent value of **n** required to achieve a given tolerance can be inflated as seen in Table 2. Sampling may not be ideal in all scenarios, but works well as an exploratory technique when the magnitude of the upper limit of the **n** values is unknown and the potential number of **n** values which need to be tested is very large.

Tolerance	**Full Range**	**Sample**
$1e^{-4}$	644	768
$1e^{-8}$	64230	65536

Table 2: Example **n** needed to reach tolerance by sampling **n** vs using full range

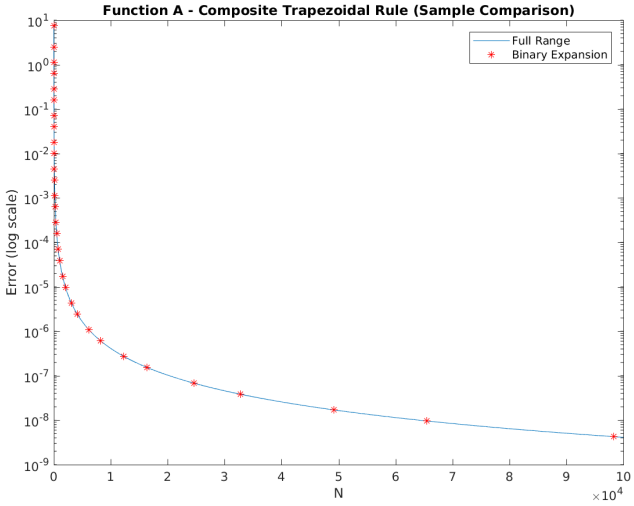

Figure 1: Sample Comparison

The value of **n** required to reach the specified tolerance is dependent on the function itself as well as the error term associated with each rule. The error terms for the Trapezoidal, Midpoint, and Simpson's Rules are defined in Equation 3, 4, and 5 respectively (Burden and Faires, 2010). As the Trapezoidal and Midpoint Rules have second order error terms, it is expected that they will require a greater value of **n** to produce results comparable to Simpson's Rule, which has a fourth order error term.

$$\frac{b-a}{6}h^2 f''(u) \tag{3}$$

$$\frac{b-a}{12}h^2 f''(u) \tag{4}$$

$$\frac{h^5}{90}f^{(4)}(\xi_j) \tag{5}$$

All values of **n** tested will be positive and even, as it is required for the Midpoint and Simpson's Rules. Although the Trapezoidal Rule can be used with odd intervals, using only even intervals will provide sufficient sample points for analysis. For most cases, the values of **n** will range from zero to three thousand, at intervals of ten. However, in certain cases where it takes roughly 100,000 subintervals to reach the lowest tolerance with the Trapezoidal or Midpoint Rules, a binary expansion will be used to generate values of **n** at which to sample the full range, rather than test at every possible step. Starting with two sub intervals ($n = 2$), if every even sub interval value up to 100,000 was tested it would require 50,000 points. By incorporating a binary expansion based approach in this scenario, a sample of only 20 different **n** values can generated

which are capable of sufficiently demonstrating the behavior of the integration rules.

Due to the higher order error term, Simpson's Rule can reach the desired tolerance much quicker and a simple range of $2:2:nmax$ can be used.

The functions implementing the Trapezoidal, Midpoint, and Simpson's Rules will be called over the range of values for **n** using the **arrayfun** function in MATLAB. This function accepts another function, defining our integration rule, and a vector, and simply repeatedly calls the specified value while iterating over the values in the vector. The call to **arrayfun** then return a vector (or set of vectors in this case) containing the results from each call to the integration function. A simplified example below show the **arrayfun** function being used to pass each **n** value in a vector called **nlist** to a trapezoidal function, returning a vector of the resulting output values.

```
[int_vals] = arrayfun(@(n) trapezoidal(n), nlist);
```

The resulting vector of integral values across varying **n** can compared with the true integral value generated earlier, to produce an error vector. The error vector is used to identify the value of **n** (and thus **h**) at which the integration rule produced results that were accurate within a desired tolerance. No **n** value needs to be specified for the adaptive approach because, as seen in Subsection 2.4, this approach iteratively checks (within the function) that a specified tolerance has been reached before producing a single output rather than using a set **n** value to produce a set of output values.

The remainder of this section will introduce the three composite numerical integration approaches (Trapezoidal, Midpoint, and Simpson's) and the adaptive approach using Simpson's Rule.

2.1 Composite Trapezoidal Rule

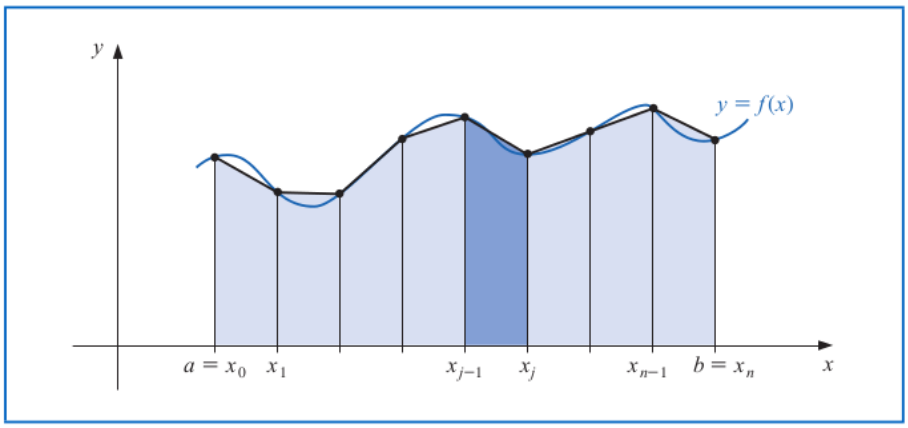

Figure 2: Trapezoidal Figure (Burden and Faires, 2010)

The Composite Trapezoidal Rule for n intervals, as seen in Figure 2, can be defined by Equation 6, given $h = (b-a)/n$ and $x_j = a + jh$, for each $j = 0, 1, \ldots, n$ (Burden and Faires, 2010).

$$\int_a^b f(x)dx = \frac{h}{2}\left[f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right] \tag{6}$$

In this equation, the integral of each individual subinterval is calculated simply as the area of a trapezoid:

$$area_{subinterval} = \frac{f(x_j) + f(x_{j+1})}{2}h \tag{7}$$

2.2 Composite Midpoint Rule

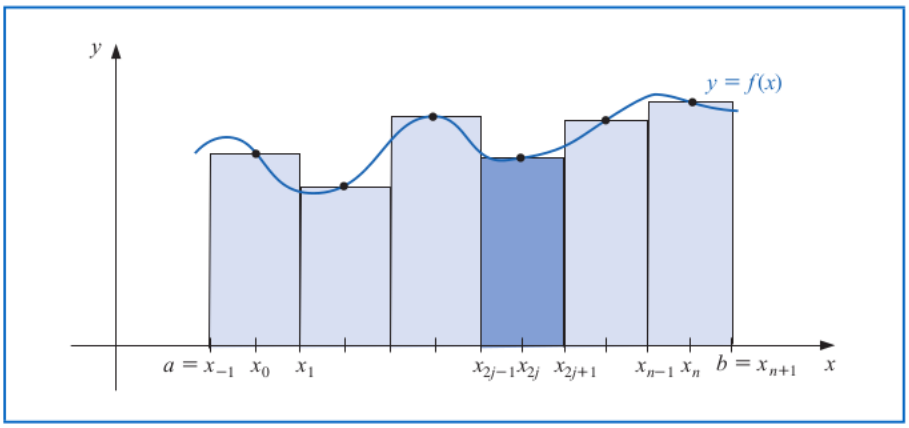

Figure 3: Midpoint Figure (Burden and Faires, 2010)

The Composite Midpoint Rule for $n+2$ intervals, as seen in Figure 3, can be defined by Equation 8, given $h = (b-a)/(n+2)$ and $x_j = a + (j+1)h$, for each $j = -1, 0, \ldots, n+1$ (Burden and Faires, 2010).

$$\int_a^b f(x)dx = 2h\sum_{j=0}^{n/2} f(x_2 j) \tag{8}$$

The area for each subinterval is similar to the Trapezoidal Rule, but instead of calculated the area of a trapezoid, the midpoint of the subinterval is used to calculate the area of a rectangle:

$$area_{subinterval} = f(\frac{x_{2j-1} + x_{2j+1}}{2})h \tag{9}$$

2.3 Composite Simpson's Rule

Unlike the Trapezoidal or Midpoint Rules, Simpson's Rule utilizes subintervals with three points, seen in Figure 4, resulting in a higher order error term as seen earlier in 5.

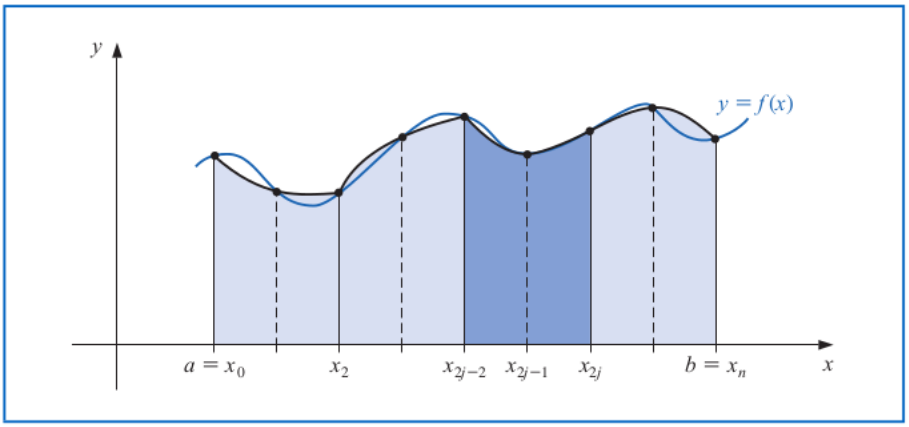

Figure 4: Simpson's Figure (Burden and Faires, 2010)

The Composite Simpson's Rule for n intervals, as seen in Figure 4, can be defined by Equation 10, given $h = (b-a)/n$ and $x0_j = a+jh$, $x1_j = a+jh+h/2$, $x2_j = a+jh+h$, for each $j = 0, 1, \ldots, n-1$.

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(a) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(b)\right] \tag{10}$$

2.4 Adaptive Simpson's Rule

An adaptive integration approach iteratively compares each subinterval (starting with the full range) with the next level of subintervals (splitting the current subinterval in half) until a specified tolerance value (difference between integration value of original subinterval and the sum of the integration of the subinterval halves) has been reached. This can be modified to calculate the integration using any method, though Simpson's Rule will be used through this project. A rough overview of this process is demonstrated using the following pseudocode.

```
integral_value = 0

subinterval_data = [
    {a: bound_min, b: bound_max},
    {...}, ...
]

number_of_subintervals_remaining = 1

while number_of_subintervals_remaining > 0

  active_interval = simpsons(a,b)

  left_half = simpsons(a, (b-a)/2)
  right_hal = simpsons((b-a)/2, b)
```

```
if active - (left_half+right_half) < tolerance

      tolerance reached
      integral_value += active_interval

else
      save left half definition
      number_of_subintervals_remaining += 1
      subinterval_data.append({aleft, bleft}

      save right half definition
      number_of_subintervals_remaining += 1
      subinterval_data.append({aleft, bleft}

      % next loop of while loop will process
      % this right half and continue halving
      % and working down right most subinterval
      % until tolerance reached then work backwards
      % towards the leftmost subinterval until all
      % subinterval which make upthe original range
      % have been processed
```

3 Results

This section will review the results of the different integration rules for Functions A and B. An overview of these results can be seen in Table 3 which provides the number of subintervals and size of each subinterval required to reach tolerances of 10^{-4} and 10^{-4} for each function-method combination. The rows of Table 3 for the adaptive method do not contain **h** values as they vary based on the adaptive response to the function behavior (details on this, along with distribution of **h** values in the adaptive results subsection). Figure 5 contains the true plots of both function, for reference.

Function	**Method**	**N:** 10^{-4}	**N:** 10^{-8}	**H:** 10^{-4}	**H:** 10^{-8}
a	trapezoidal	768	65536	0.002604	3.0517578e-05
a	midpoint	1024	98304	0.001949	2.0344638e-05
a	simpsons	42	322	0.047619	0.0062111
a	adaptive	24	278	-	-
b	trapezoidal	12	1082	0.166667	0.001848
b	midpoint	22	1522	0.083333	0.001312
b	simpsons	2	22	1	0.090909
b	adaptive	2	16	-	-

Table 3: Results

Table 4 shows the final error value and the number of subintervals required to achieve it. It should be noted that this minimum error is based purely on an arbitrary maximum **n** value in the list of **n** values that was only chosen to ensure the two specified tolerances were reached. Additional iterations with larger values of **n** and smaller **h** would yield improved accuracy in most cases, except where computational precision limits are reached (this case will be addressed along with the results of Simpson's Rule for Function A).

Function	**Method**	**N: Min Error**	**Min Error**
a	trapezoidal	98304	4.26887e-09
a	midpoint	98304	8.53747e-09
a	simpsons	2992	1.30384e-12
b	trapezoidal	2992	1.29289e-09
b	midpoint	2992	2.58233e-09
b	simpsons	2992	2.22044e-16

Table 4: Results

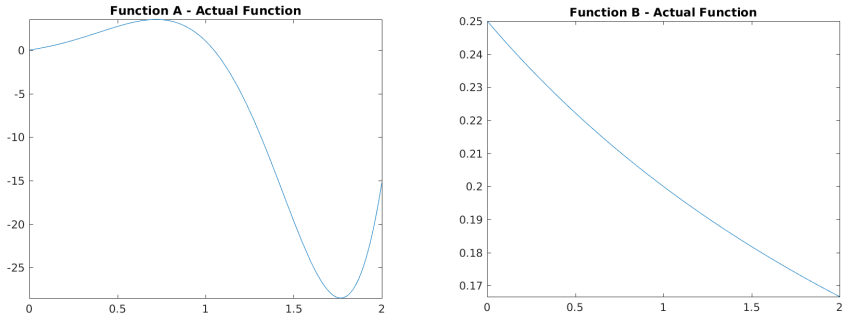

Figure 5: True plots of Function A and B between zero and two

3.1 Trapezoidal and Midpoint Results

The Composite Trapezoidal and Midpoint Rules performed comparably well to one another, as expected given their similarity - both conceptually and in terms of the equations used to implement them (and the associated error terms).

Function A require relatively large values of **n**, nearly 100,000 to reach the lower level tolerance of 10^{-8} and necessitated the use of the binary expansion sampling discussed in the *Methods* section earlier. Ultimately, the Trapezoidal and Midpoint Rules required an **n** values of 65,000 and 98,000, respectively, to achieve this tolerance level (seen in Table 3[2].

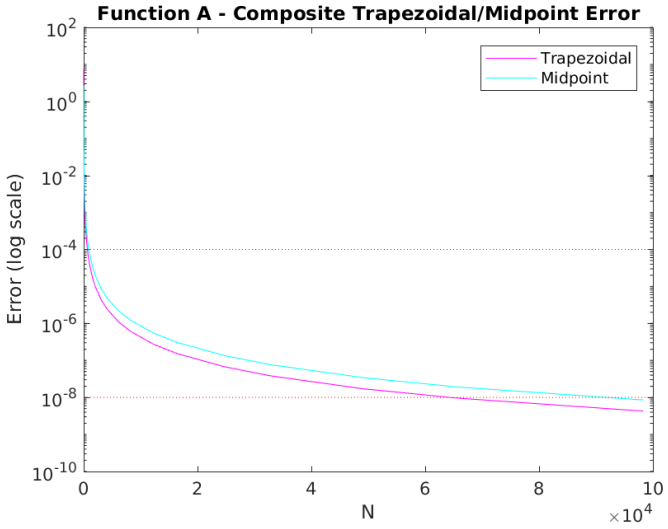

Figure 6: Plot of error from Trapezoidal and Midpoint integration of Function A

The value of **n** required to reach a tolerance of 10^{-8} for Function B was only 1082 and 1522 for the Trapezoidal and Midpoint Rules, respectively.

[2]As mentioned in the *Methods* section, the values of **n** listed in Table 3 for the Trapezoidal and Midpoint Rules used with Function A are slightly greater than what is actually required to reach the specified tolerance, due to the binary sampling.

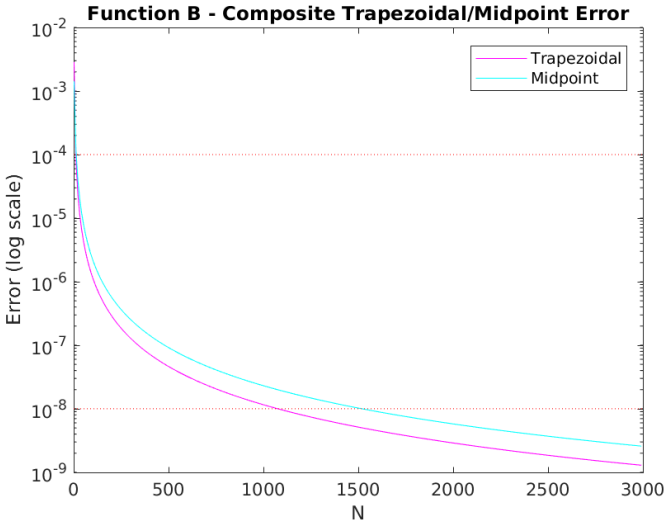

Figure 7: Plot of error from Trapezoidal and Midpoint integration of Function B

3.2 Simpson's Results

The effectiveness of Simpson's Rule as indicated by its higher order error term, discussed earlier, is clearly evident in the plot of the error for each function across the range of **n** values, seen in Figures 8 and 9.

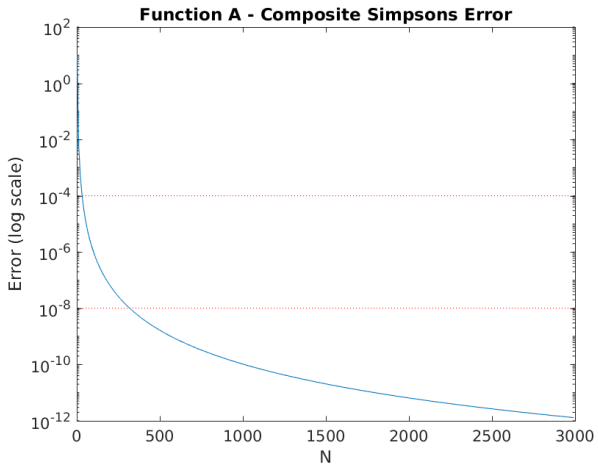

Figure 8: Plot of error from Simpson's integration of Function A

Although notably faster at reaching the tolerance levels of both 10^{-4} and 10^{-8} for Function A, it slows down as the error nears 10^{-12} - possibly an attribute of the function rather than Simpson's Rule. When used with Function B, Simpson's Rule reaches an error of around 10^{-15} in around only 1000 iterations. At this level of error, floating point precision and the limitations of the

precision that MATLAB can handle[3] becomes apparent. In Figure 9 the fluctuations causes by hitting this limit of precision can be seen between **n** values of 1000 and 3000.

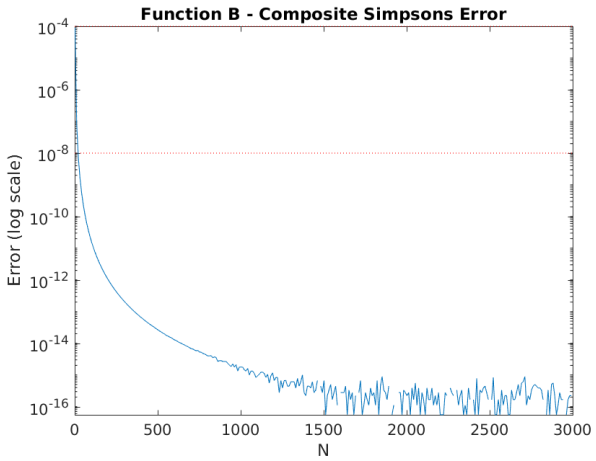

Figure 9: Plot of error from Simpson's integration of Function B

3.3 Adaptive Simpson's Results

Based on the true function plots in Figure 5 it is reasonable to expect that an adaptive approach may yield some benefit for Function A due to the variable nature of the function over this range, while Function B has a relatively minor and fairly consistent slope and would not be a good use case for an adaptive method.

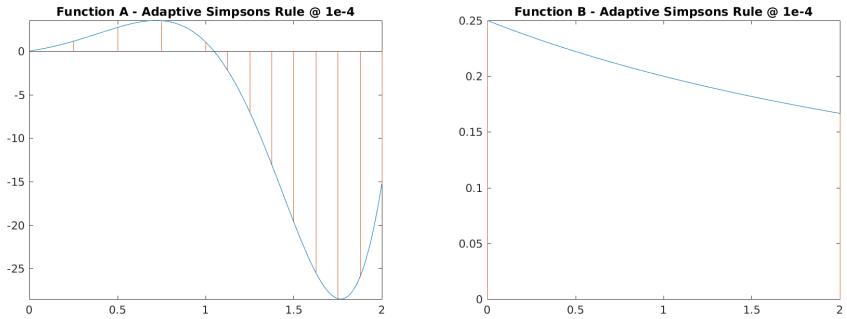

Figure 10: Subintervals generated by adaptive Simpson's Rule for Function A and B with tolerance of 10^{-4}

Figure 10 shows the subinterval pairs generated by the application of the adaptive Simpsons's Rule with a relatively high tolerance of 10^{-4}. Although there is some minor variation in the values of **h** for Function A, Function B

[3]See following link on MATLAB precision limitions (general limitations of floating point representations apply) https://www.mathworks.com/help/fixedpoint/ug/limitations-on-precision.html

required only a single subinterval pair to reach the tolerance and clearly did not need an adaptive approach. When the tolerance is reduced to 10^{-8}, Function B still has no variance in the values of **h** over the 16 subintervals generated, seen in Figure 13. Function A, however, does benefit slightly over the 278 subinterval pairs seen in Figure 11. The distribution of the size of subintervals can be seen in Figure 12.

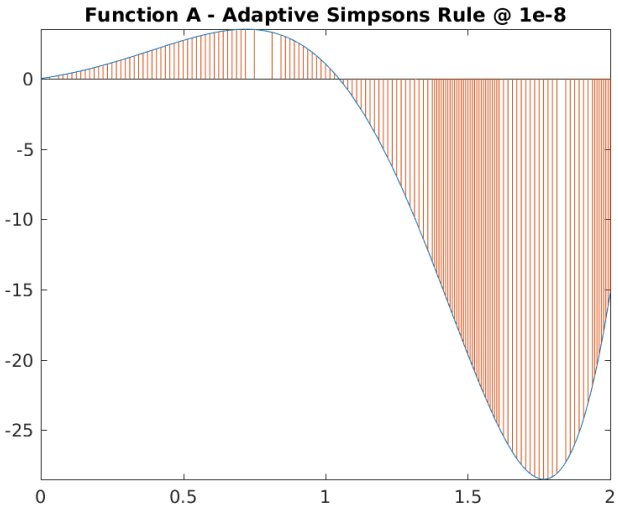

Figure 11: Subintervals generated by adaptive Simpson's Rule for Function A with tolerance of 10^{-8}

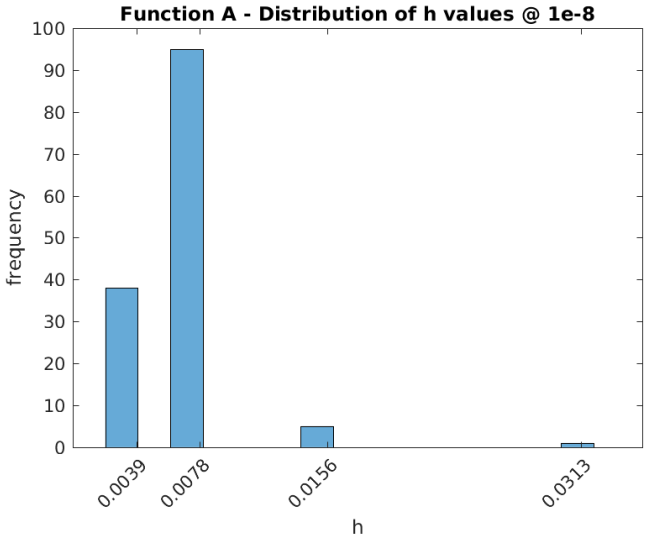

Figure 12: Distribution of the size of subintervals generated from adaptive Simpson's for Function A

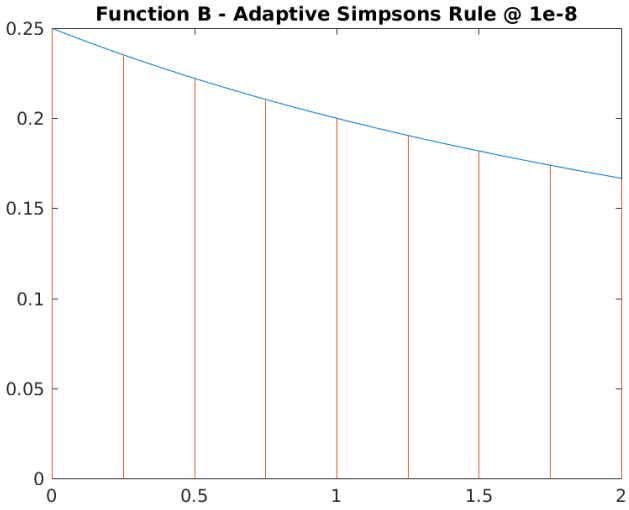

Figure 13: Subintervals generated by adaptive Simpson's Rule for Function B with tolerance of 10^{-8}

If the range over which the integral was taken is expanded, as seen in Figure 14, an adaptive approach becomes much more useful. When expanded with a lower bound of -2 and an upper bound of 3, Function A has a substantial portion of the range between -2 and 1 where the integral will be close to zero. The adaptive approach would reduce **h** over this range, and then between 1 and 3, where there is significant variation in the function and a larger relative integral, the value of **h** would increase as needed. Similarly, when Function B is expanded to -3.9 and 2, the integral between -3.9 and -3 is much larger relative to the integral between -3 and 2. In this case, the adaptive approach to integration would again prove useful[4].

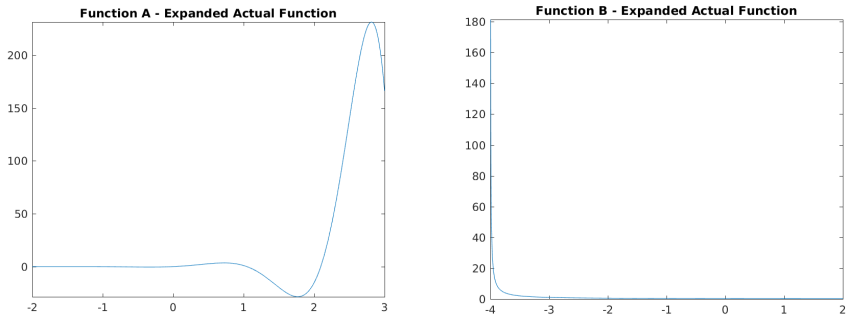

Figure 14: True plots of Function A and B over expanded range

[4]Function B has a vertical asymptote at -4 which introduces a complication when calculating the integral as it is technically infinite at -4. A practical approach to calculating this integral would be to split the calculation into one part up to just before -4 and another part start just after -4

References

R. Burden and J Faires. *Numerical Analysis*. Brooks/Cole, 9th edition edition, 2010.