Project 02

APSC 607 Fall 2017

Seth Goodman

October 26, 2017

1 Introduction

This project explores different numerical methods for calculating the definite integrals of functions. Each 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 the effectiveness of the resulting value for the integral when using varying number of subintervals, **n**. For each **n**, the result will be evaluated with respect to the true integral value using defined tolerances thresholds.

All computations are performed using MATLAB using the code accompanying this report (in a zip file). The following *Methods* section will present the methods used in MATLAB to explore functions, as well as the outputs and results. The *Results* section of this report contains the outputs for each function along with related 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

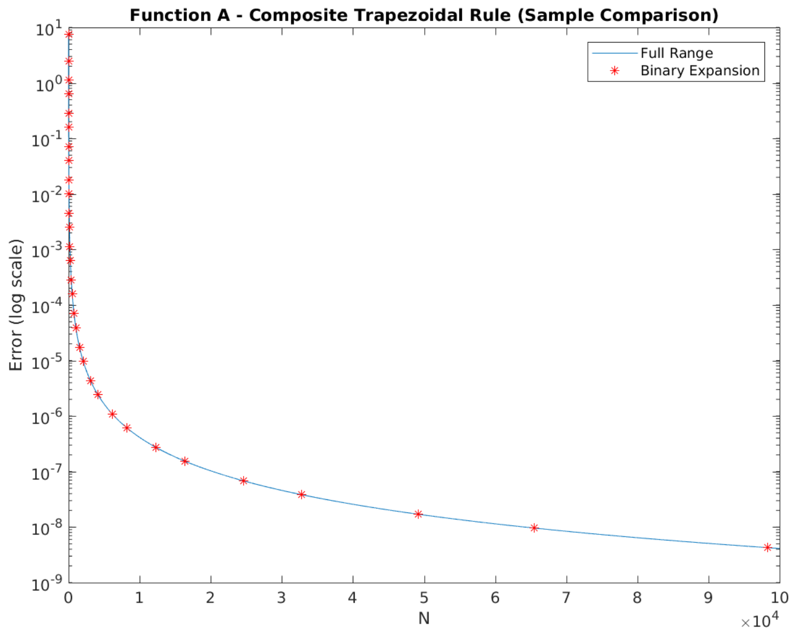

Figure 1: Sample Comparison

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. In order to test **n** over a sufficient range for the Trapezoidal and 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, 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

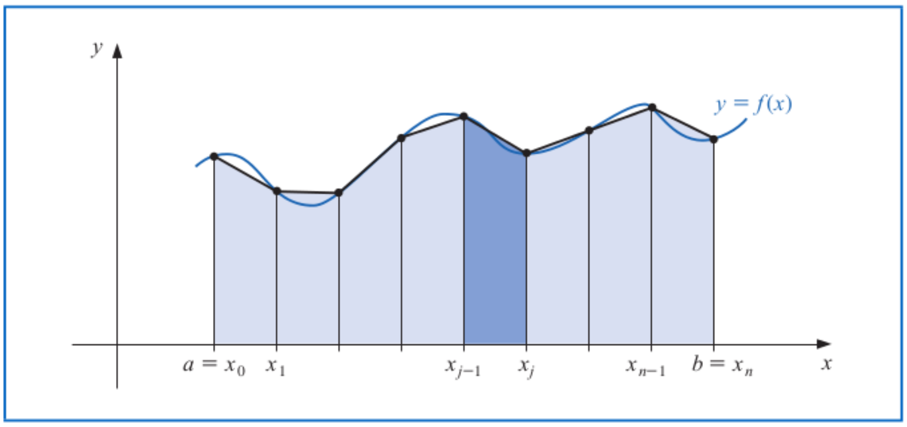

Figure 2: Trapezoidal Figure (Burden and Faires, 2010)

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 $j = 0, 1, \dots, n$ (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

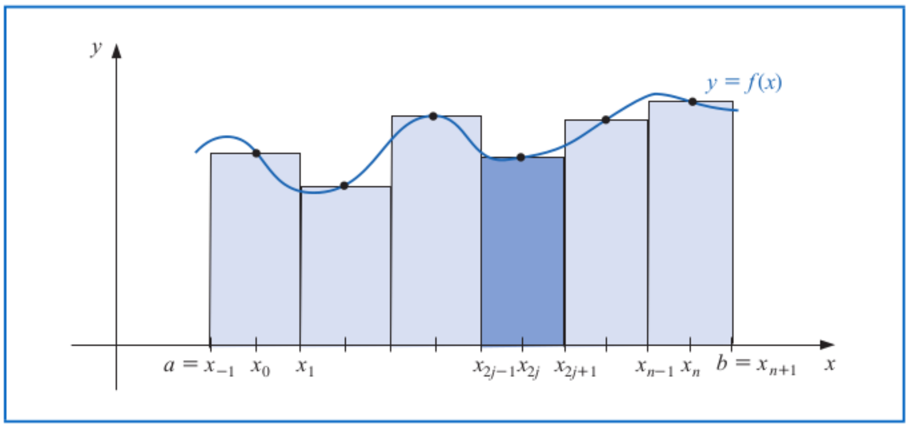

Figure 3: Midpoint Figure (Burden and Faires, 2010)

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).

$$\int_a^b f(x)dx = 2h\sum_{j=0}^{n/2} f(x_2j) \tag{8}$$

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

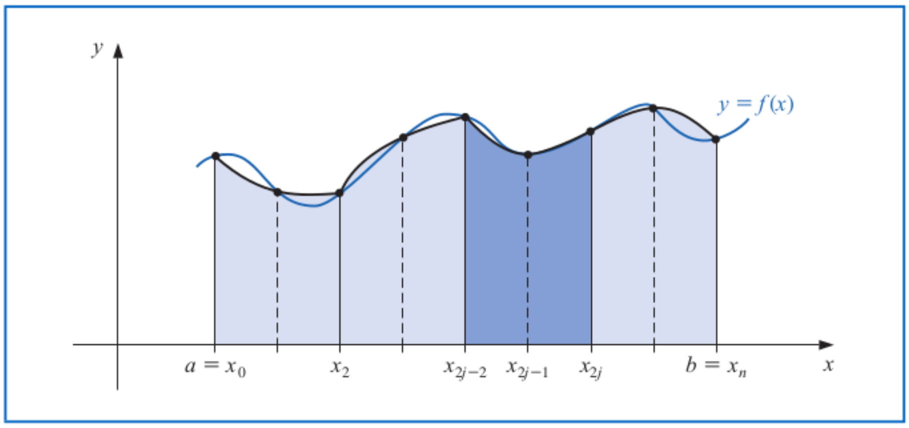

Figure 4: Simpson's Figure (Burden and Faires, 2010)

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 $j = 0, 1, \ldots, n-1$.

$$\int_a^b f(x)dx = \sum_{j=0}^{n-1} \left[\frac{h}{6} [f(x0_j) + 4f(x1_j) + f(x2_j)] \right] \tag{10}$$

2.4 Adaptive Simpson's Rule

```
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
        % continuehalving and working down right most subinterval until
        % then work backwards towards the leftmost subinterval until a
```

3 Results

TEXT

Something about minimum error in footnote[2].

3.1 A

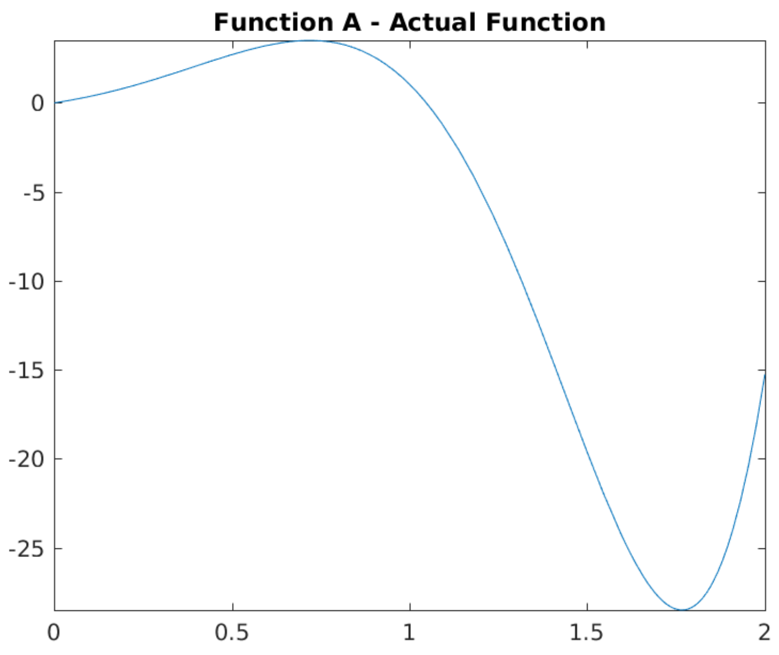

Figure 5: caption text a

[2]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

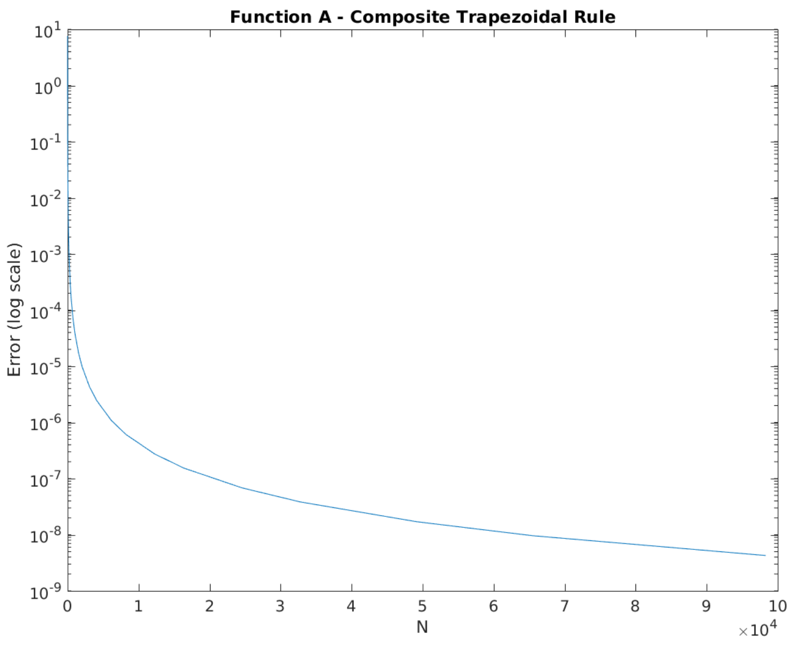

Figure 6: caption text a

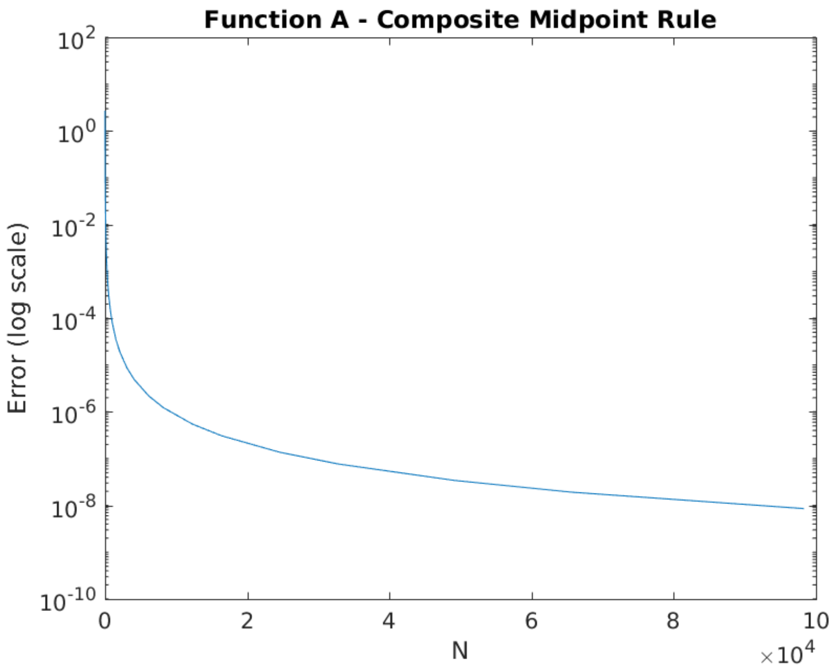

Figure 7: caption text a

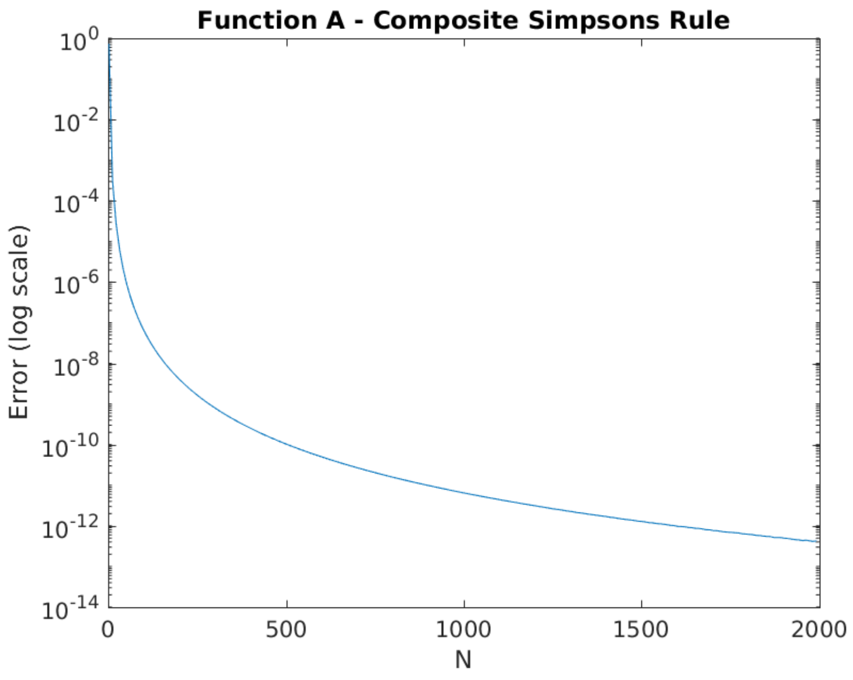

Figure 8: caption text a

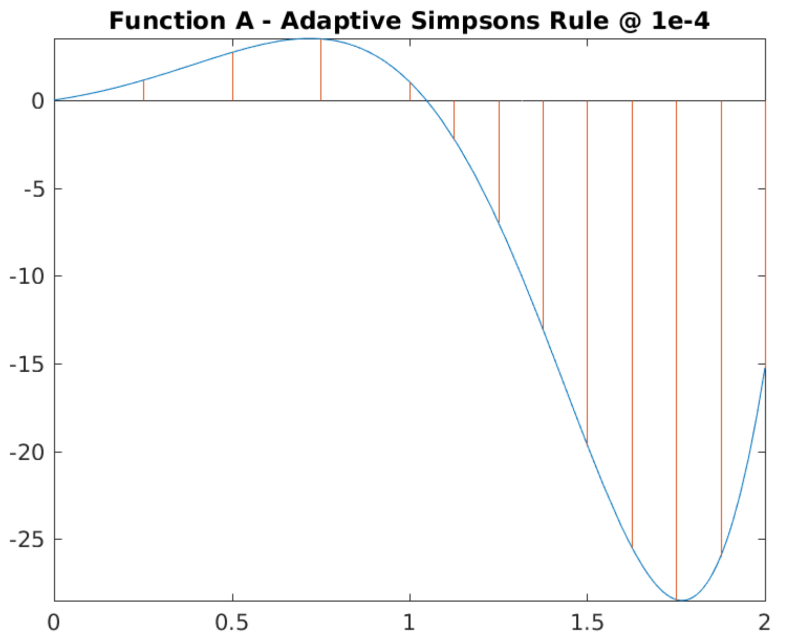

Figure 9: caption text a

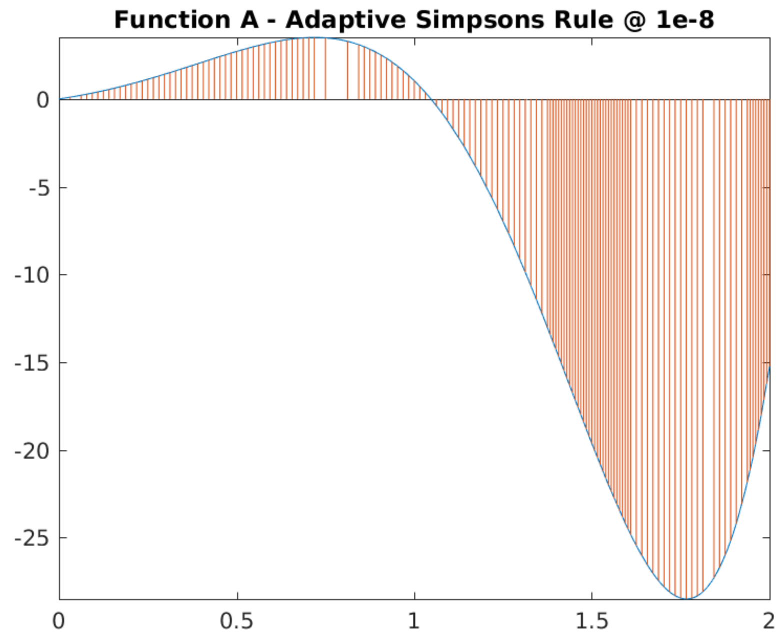

Figure 10: caption text a

3.2 B

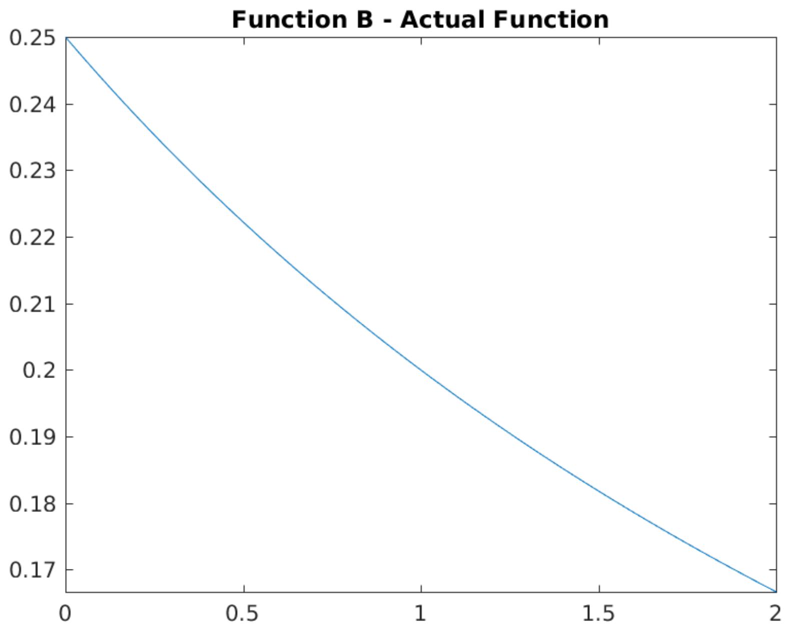

Figure 11: caption text a

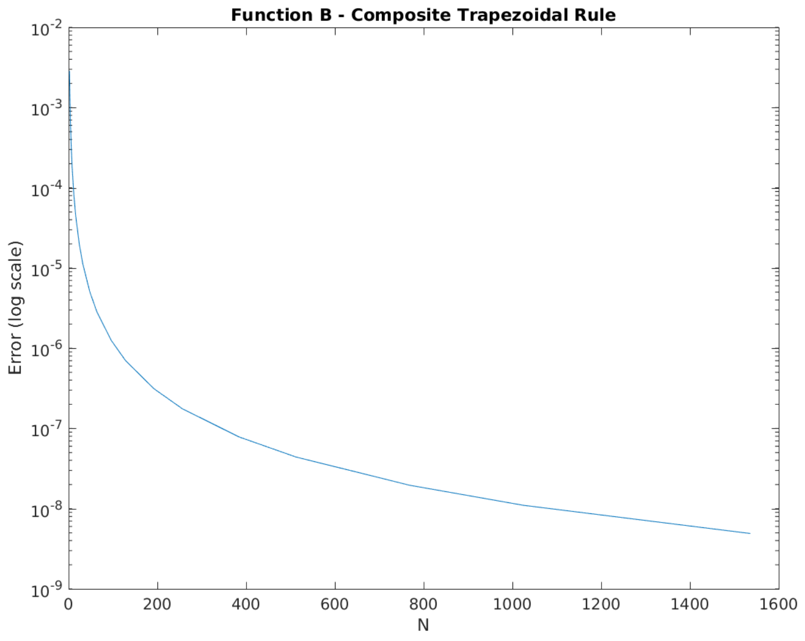

Figure 12: caption text a

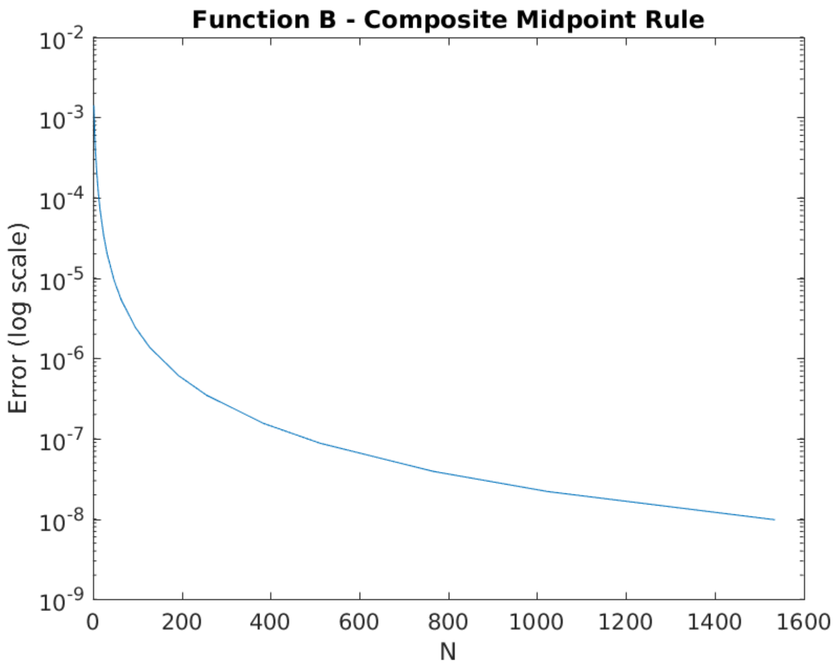

Figure 13: caption text a

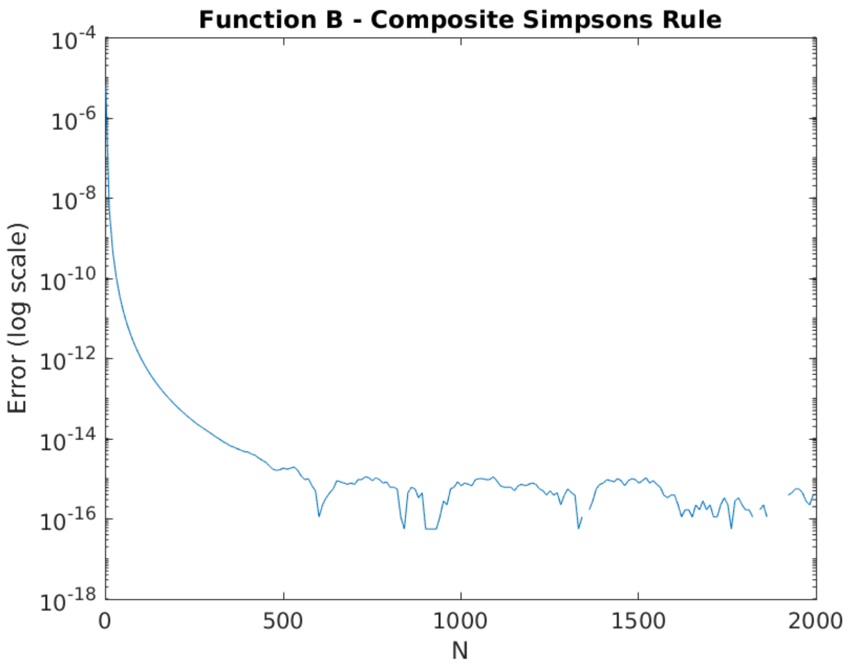

Figure 14: caption text a

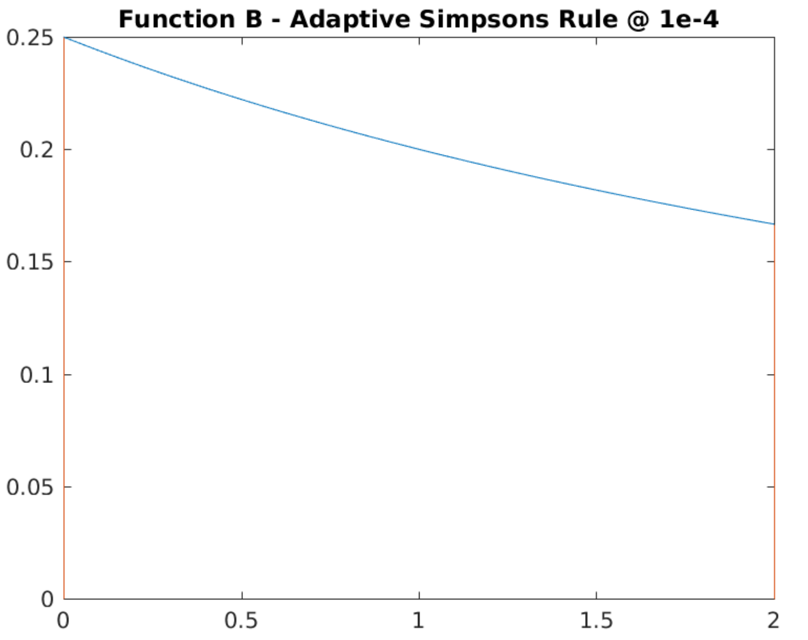

Figure 15: caption text a

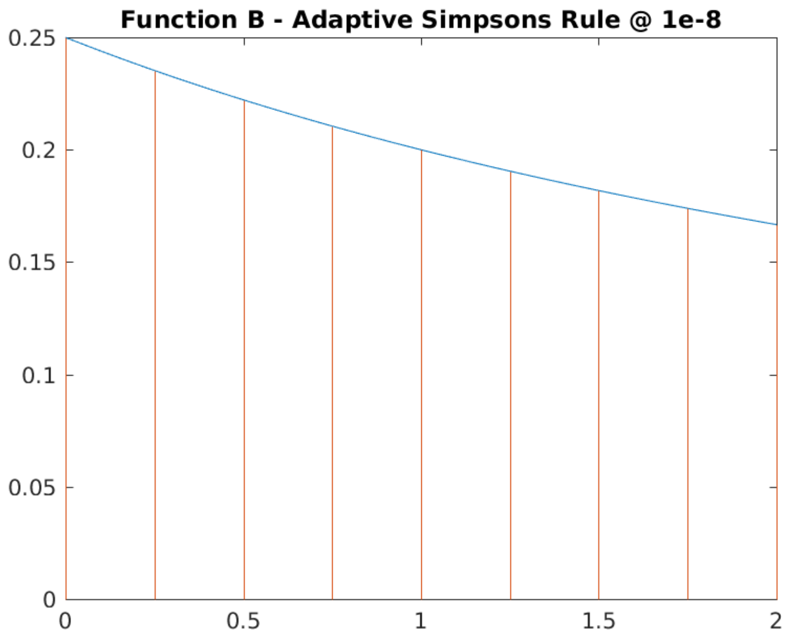

Figure 16: caption text a

3.3 Expanded Range

integration over an expanded range and the potential utility of adaptive approaches in such a scenario

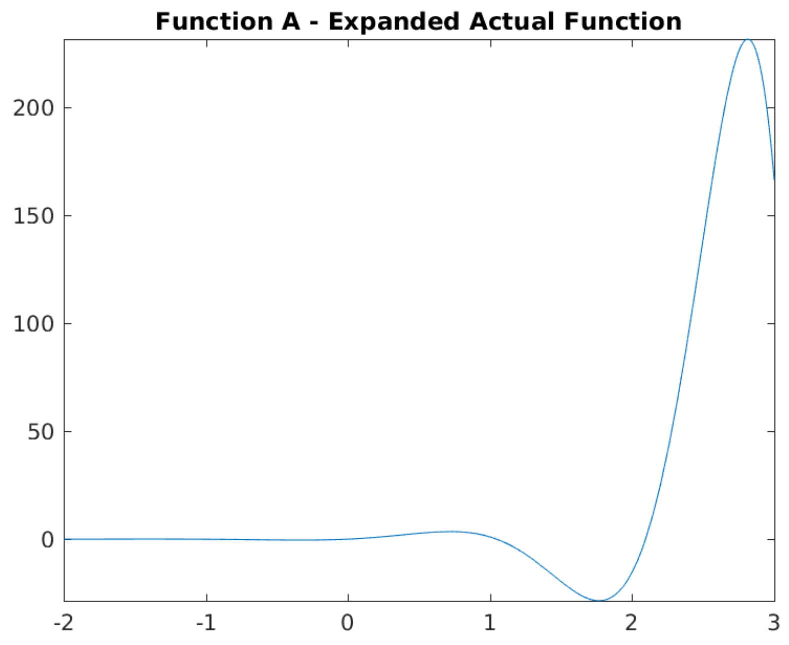

Figure 17: caption text a

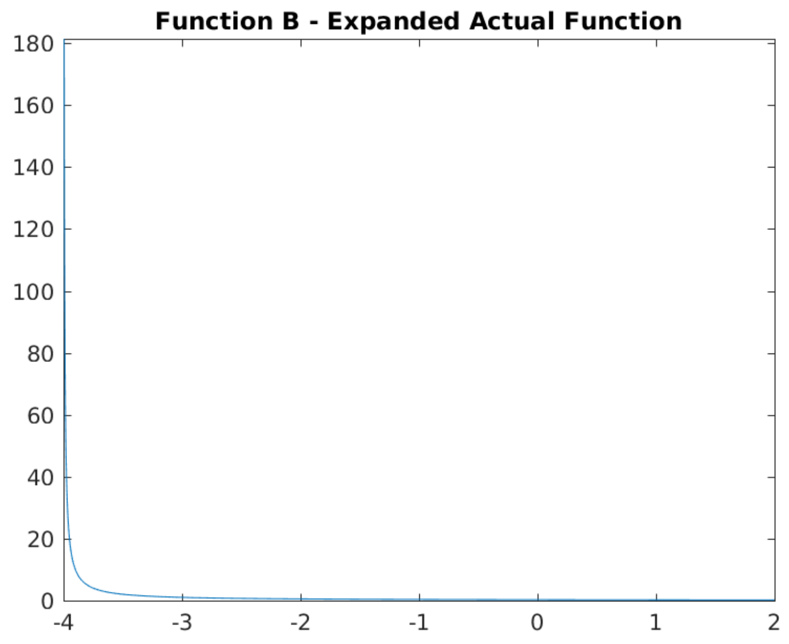

Figure 18: caption text a

References

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