Blake Van Dyken

Assignment 3

Methods Used:

- I used the below methods to compute this Integral

$$\int_0^{2\pi} 1 + \sin(x) \cdot \cos\left(\frac{2x}{3}\right) \cdot \sin(4x)\, dx$$

- For all below, N = 1 to 1024, a = 0, b = 2pi, f(x) = 1 + sin(x)*cos(2x/3) * sin(4x)
- **Composite Midpoint Rule**
  - This approximates the integral by N applications of a midpoint between a and b.
  - I used this formula $\int_a^b f(x)dx \approx \sum_{i=1}^{N} w_i f(x_i)$
  - Delta_x = $\frac{b-a}{N}$
  - $x_i = a + (i - .5)\Delta x$
  - $w_i = \Delta x$
- **Composite Trapezoid Rule**
  - This approximates the integral by N applications by drawing a trapezoid between a and b and finding the area.
  - I used this formula $\int_a^b f(x)dx \approx \sum_{i=1}^{N} w_i f(x_i)$

$$x_i = a + (i-1)\Delta x$$

$$w_i = \begin{cases} \frac{\Delta x}{2}, & i = 1, N \\ \Delta x, & i = 2, \dots, N-1 \end{cases}$$

$$\text{where } \Delta x = \frac{b-a}{N-1}$$

- **Composite Simpson Formula**
  - Uses quadratic Lagrange polynomial to approximate the polynomial at the lines between a and b which basically finds another known polynomial to draw between the points to approximate it better.
  - I used the formula $$\int_a^b f(x)dx \approx \sum_{i=1}^{N} \int_a^b g(x)dx = \sum_{i=1}^{2N+1} w_i f(x_i)$$

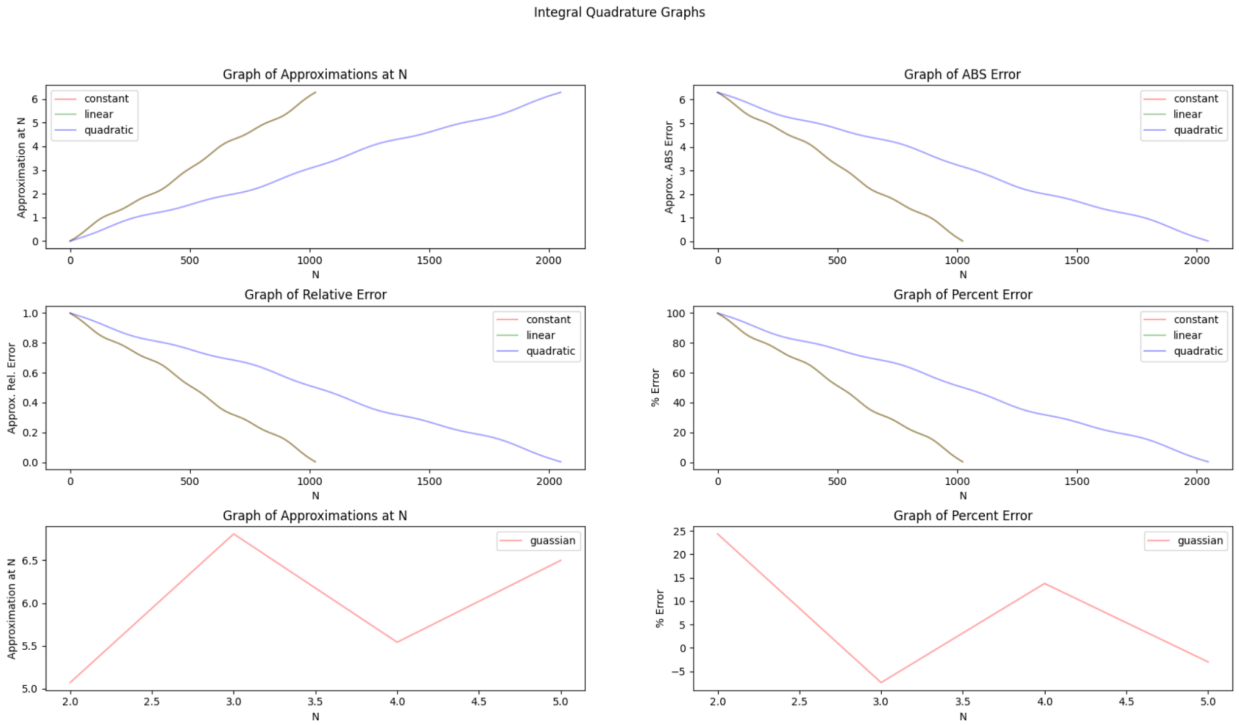
$$w_i = \begin{cases} \frac{\Delta x}{3} & : \quad i = 1, 2N+1 \\ \frac{4\Delta x}{3} & : \quad i = 2, \dots, 2N \quad (i \text{ even}) \\ \frac{2\Delta x}{3} & : \quad i = 3, \dots, 2N-1 \quad (i \text{ odd}) \end{cases}$$

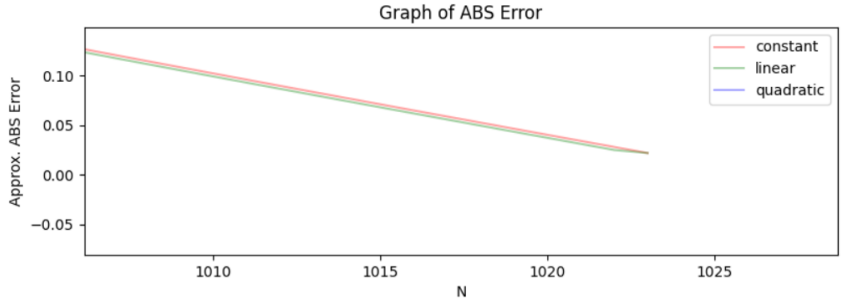
$$x_i = a + (i-1)\Delta x$$

$$\text{where } \Delta x = \frac{b-a}{2N}$$

- I used the following formula below to calculate my errors
  - **ABS Error** = True value - computed value
  - **Relative Error** = True value - computed value / True value
  - **Percent Error** = True value - computed value / True value * 100

Results and Figures:






```
True ans:  6.305171
Constant ans:  6.2831853071795924
Linear ans:  6.283185307179597
Quadratic ans:  6.2852306148967685
N =  2  guass =  5.071054755798437
N =  3  guass =  6.807774066063649
N =  4  guass =  5.5424044072742555
N =  5  guass =  6.4983310917111705
```

The true solution for the given integral is 6.305171, and using the three different Newton-Cote methods does not give us an exact answer that is the same, but they do get very close without having to do difficult integration work. I graphed all three types of errors but all can explain the same thing. Thus, looking at the percent error for the 3 Newton-Cote Formulas we can see that as N increases, we get a smaller (percent) error. The first thing we notice is that the quadratic interpolant gives us a good approximation overall just like the other two Newton-Cote methods, but it takes 2*N+1 more work/steps to calculate so it is the slowest of the formulas. The next important thing to notice is that the constant and linear interpolant formulas are very similar to each other, however, if we zoom in on the graph (which is the second image) we can see that the linear plot lies below the constant one and is thus marginally better towards the end of the graph where N is larger. If we were to zoom in along the beginning and middle of the graph we will see the linear and constant lines fluctuate above each other. Therefore, the linear plot converges the fastest and we can see that on the percent error graph or the theoretical error graphs and on the graph of the actual approximated values.

For the Gaussian quadrature formula, we only used a few values for N, and we get more of a range of values the answer could be between but we get nothing we can use. It is a high-order function, as it uses gauss-Legendre's to compute its answer, but the larger the N, the larger the degree polynomial and equation gets, thus it gets more computationally expensive. Looking at the Gaussian percent error graph compared to the Newton Cotes percent error graph, we can see that the Gauss graph fluctuates

more and gives us a higher error. It does not do a good job because we only do work up to N = 5, but if we were to increase N to a much larger number and computed much higher degree polynomials for the equation, the result would be more accurate; however, it would take too much time and resources to compute.

Sources:

All my code is submitted with the project and on GitHub here:

https://github.com/iPupkin/Theory-3200