Simpson's Rule for Integration

Morgan Buterbaugh and Jaylee Lassinger

December 2023

1 Introduction

In numerical mathematics, there is a numerical integration method called Simpson's rule. This method of integration is more accurate than other numerical approximations such as the Fundamental Theorem of Calculus, Trapezoid Rule, and Newton-Cotes. Simpson's Rule is based on dividing the region of interest into intervals and then calculating the average value of each interval [5].

There are two different Simpson's Rules, the 1/3 and 3/8 rules. Simpson's 1/3 Rule is based on quadratic interpolation quadrature, while Simpson's 3/8 Rule is based on cubic interpolation [2]. In this paper, we will explore both of the Simpson's Rules in one dimension, two dimensions, and three dimensions. We aim to find the difference between the two different Simpson's Rules in each and see which one of the two rules is more accurate.

2 Simpson's Rule in 1 Dimension

Simpson's Rule in one dimension approximates the numerical integration of a single integral. Consider the two following rules.

2.1 Original Simpson's 1/3 Rule in 1 Dimension

Simpson's 1/3 Rule approximates polynomials based on quadratic interpolation quadrature. It is an extension of the trapezoidal rule, in which the integrand is approximated by a second-order polynomial. Simpson's 1/3 Rule can be derived from using Newton's divided difference polynomial, Lagrange polynomial, and the method of coefficients [2]. The formula for Simpson's 1/3 Rule is the following [3]:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right].$$

If we let $h = \frac{b-a}{2}$, then we get

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f(a+h) + f(b)\right].$$

2.2 Original Simpson's 3/8 Rule in One Dimension

Simpson's 3/8 Rule approximates polynomials based on cubic interpolation quadrature, as opposed to quadratic interpolation [4]. It is a uniformly sampled integration [1]. The formula for Simpson's 3/8 Rule is the following [3]:

$$\int_a^b f(x)dx \approx \frac{b-a}{8}\left[f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right].$$

If we let $h = \frac{b-a}{3}$, then we get

$$\int_a^b f(x)dx \approx \frac{3h}{8}\left[f(a) + 3f(a+h) + 3f(a+2h) + f(b)\right].$$

2.3 Composite Simpson's 1/3 Rule

If a function oscillates a lot or lacks derivatives at certain points, Simpson's 1/3 Rule will no longer produce accurate results. To work with functions like this, the interval $[a, b]$ must be

broken down into n, smaller sub-intervals so that the function is smooth over the interval. The number of sub-intervals, n, must be divisible by 2 for the Composite Simpson's 1/3 Rule. After the number of sub-intervals is established, Simpson's 1/3 Rule will be applied to each sub-interval and the sum of the results gives the approximation for the integral over the whole interval [3]. Composite Simpson's 1/3 Rule is given by [3]:

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left[f(a) + 4f(x_1) + 2f(x_2) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right].$$

If we let $h = \frac{b-a}{n}$, then we get

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4f(x_1) + 2f(x_2) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right].$$

2.4 Composite Simpson's 3/8 Rule

The same logic for Composite Simpson's 1/3 Rule is applied to Composite Simpson's 3/8 Rule as well [2]. However, n should be divisible by 3 now. The equation for Composite Simpson's 3/8 Rule is given by:

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left[f(a) + 2f(x_1) + 2f(x_2) + 3f(x_3) + ... + 3f(x_{n-3}) + 2f(x_{n-2}) + 2f(x_{n-1}) + f(x_n)\right]$$

If we let $h = \frac{b-a}{n}$, then we get

$$\int_a^b f(x)dx \approx \frac{3h}{8}\left[f(a) + 2f(x_1) + 2f(x_2) + 3f(x_3) + ... + 3f(x_{n-3}) + 2f(x_{n-2}) + 2f(x_{n-1}) + f(x_n)\right].$$

These rules now allow us to increase the accuracy by increasing the number of sub-intervals that are present.

2.5 Explanation of Simpson's Rule Code in 1 Dimension

NOTE: All code is located in the appendix.

2.5.1 Simpson's 1/3 Rule Code

The first step in the code is to define the function we wish to integrate, this can be any function. From here we can start defining Simpson's 1/3 Rule, which we call $Simpson13()$. We defined a function that takes in four parameters, f, a, b, and n, where f is the function that we wish to take the integral of, a is the lower bound of integration, b is the upper bound of integration, and n is the number of sub-intervals that one wishes to have, as long as it is divisible by 2. Next, we have to define the number of time steps h and determine what it is equal to, which is discussed above. Thus, $h = \frac{b-a}{n}$ [6]. We now need to define a variable in which the value of the integral is going to be stored in, which we call *integration*. To begin, since the first value and the last value are not multiplied by any weights we set $integration = f(a) + f(b)$, which are the two endpoint values. Next, a for loop is needed to sum all of the middle terms and decide which values are being multiplied by 4 and which values are multiplied by 2. The values that are multiplied by 4 are the odd i values in the for loop. This would mean that we would add the value of $4f(a + i * h)$ to *integration*. If the i value is even then the value of $2f(a + i * h)$ is added to *integration*. It is important to note that the value that is being plugged into the function f is not the value of i but rather $a + i * h$. Following the for loop the final step is to take the value of *integration* and multiply it by the $\frac{h}{3}$ factor, and return the value of the integral. The last step is setting the parameters for $Simpson13(f, a, b, n)$ to be our chosen values.

2.5.2 Simpson's 3/8 Rule Code

The first step in the code is to define the function we wish to integrate, this can be any function. From here we can start defining Simpson's 3/8 Rule, which we call $Simpson38()$. We defined a function that takes in four parameters, f, a, b, and n, where f is the function

that we wish to take the integral of, a is the lower bound of integration, b is the upper bound of integration, and n is the number of sub-intervals that one wishes to have, as long as it is divisible by 3. Next, we have to define the number of time steps h and determine what it is equal to, which we previously discussed. Thus, $h = \frac{b-a}{n}$. We now need to define a variable in which the value of the integral is going to be stored in, which we call *integration*. To begin, since the first value and the last value are not multiplied by any weights we set $integration = f(a) + f(b)$, which are the two endpoint values. Next, a for loop is needed to sum all of the middle terms and to decide which values are being multiplied by 2 and which values are multiplied by 3. The values that are going to be multiplied by 2 are the i values in the for loop that are divisible by 3. This would mean that we would add the value of $2f(a+i*h)$ to *integration*. If the i value is not divisible by 3 then the value of $3f(a+i*h)$ is added to *integration*. It is important to note that the value that is being plugged into the function f is not the value of i but rather $a+i*h$. Following the for loop the final step is to take the value of *integration* and multiply it by the $\frac{3h}{8}$ factor, and return the value of the integral. The last step is setting the parameters for $Simpson38(f, a, b, n)$ to be our chosen values.

3 Mathematical Modeling Example for Simpson's Rule in 1 Dimension

The mathematical modeling problem we will be working to solve is:

A researcher studied the population dynamics of robins in southern California. Over the span of five years, the researcher collected irregular data on the robin population at various time intervals. The population counts at these intervals are as follows, where t is in years:

t	Number of robins
0	100
1	120
2	150
3	200
4	180
5	250

Using both Simpson's Rules, estimate the total growth in the robin population over the five-year period.

NOTE: All code is located in the appendix.

3.1 Solving with Simpson's 1/3 Rule

To begin, the first step is to define the time intervals and the corresponding population counts as separate lists. We also have to define a function that interpolates between the data points, call it *interpolate_pop*, which checks to make sure if the end time matches any of the recorded times. If not, we have to compute a linear interpolation to estimate the population between the two closest known time intervals. We created a for loop that will do this calculation for us in which we update the time difference and population difference, find the slope, and calculate the interpolate population by taking the current population count and adding the slope multiplied by the time given minus the current time. Next, we have to modify Simpson's 1/3 Rule, that we previously coded, in terms of this model. The function takes in the starting time, $t0$, the ending time tn, and the number of sub-intervals n. It calculates the step size by taking the ending time minus the start time all divided by the number of sub-intervals. We now need to define a variable in which the value of the integral is going to be stored in, which we call *integration*. For this, we use the two endpoint values $t0$ and tn. The same process for figuring out if the points are even or odd and the final value

of the integration is the same in the code. Next, we define our parameters $t0$, tn, and n. The start time is 0 years and we set the end time as 5 years, with the number of sub-intervals set to 150. The result showed that the estimated total population growth is 823.3851851851852, or about 824 robins over the five-year period. We also tried setting the end time to 4.5 years and keeping the start time and the number of intervals the same and we got the estimated total population growth to be 707.6179, or 708 robins.

3.2 Solving with Simpson's 3/8 Rule

The same exact steps were followed from Simpson's 1/3 Rule to solve this problem except for the fact that we modified Simpson's 3/8 Rule in our code. We set the start time as 0 years, the end time as 5 years, with the number of sub-intervals set to 150. The result showed that the estimated total population growth is 825 robins over the five-year period. We also tried setting the end time to 4.5 years and keeping the start time and the number of intervals the same and we got the estimated total population growth to be 708.7578749999999, or 709 robins.

3.3 Comparing the Two Rules

When the start time is 0 years, the end time is 5 years, and the number of sub-intervals is 150, Simpson's 1/3 Rule approximated 824 robins, while Simpson's 3/8 Rule approximated 825 robins over the five-year period. Thus, the Simpson's 3/8 Rule is more accurate, but only by a little. Also, the same goes for when we change the end time to 4.5 years, keeping all the other variables the same. Simpson's 1/3 Rule gave 708 robins, while Simpson's 3/8 Rule gave 709 robins. Again, we can see that Simpson's 3/8 Rule does a better job of estimating the total population of robins.

4 Simpson's Rule in 2 Dimensions

NOTE: All code is located in the appendix.

Once we figured out Simpson's 1/3 Rule and Simpson's 3/8 Rule in one dimension, our goal was to expand this idea into two dimensions. After a while, we were able to come up with a way to do such calculations of a double integral using both of Simpson's Rules by taking the one-dimensional code and implementing it onto the other variable, y. Hence, we can see that the codes are similar in the aspect of the calculations that are being done to figure out what coefficient goes with what value of the function.

4.1 Explanation of Simpson's 1/3 Rule in Two Dimensions

Using the knowledge of Simpson's 1/3 rule in one dimension, we were able to compute the two dimensional equation to be:

$$\int_{a_x}^{b_x}\int_{a_y}^{b_y} f(x,y)dydx \approx \frac{h_x h_y}{9}\sum_{i=0}^{m}\sum_{j=0}^{n} w_{i,j} f(x_{i,j}, y_{i,j}),$$

where $h_x = \frac{bx-ax}{m}$, $h_y = \frac{by-ay}{n}$, $w_{i,j}$ is the weight, or coefficient, that gets multiplied to the function $f(x_{i,j}, y_{i,j})$, $x_{i,j}$ and $y_{i,j}$ represent the x points at i, j and the y points at i, j respectively. The fraction $\frac{h_x h_y}{9}$ is now over 9 instead of 3, because we are taking $\left(\frac{1}{3}\right)^2$ to get $\frac{1}{9}$.

To code this, we must again define the function of interest that we want to take the integral of. From this point, we are going to implement the same idea as in Simpson's 1/3 Rule in one dimension. The function $SimpsonsDouble13()$ is defined taking in parameters $(g, ax, bx, ay, by, nx, ny)$ where g is the function that is defined, ax and bx are the integral bounds of x, ay and by are the integral bounds of y, and nx and ny are the number of sub-intervals wanted for each x and y which must be divisible by 2, typically set to the same number. We now have two different values of h, one for the x values and one for the y values, they are defined the same as before, with $hx = \frac{bx-ax}{nx}$ and $hy = \frac{by-ay}{ny}$. From here, we used

the same idea as the one dimensional Simpson's Rule when finding the coefficients. It can be seen that it is broken into two parts, a part for the x coefficient and a part for the y coefficient. They both however work the in exact same way. In the for loops if i or j are at one of the endpoints, then the coefficient is going to be equal to 1. If i or j is divisible by 2, meaning the remainder when dividing by 2 is equal to 0, then the coefficient is going to be equal to 2. If i or j is not divisible by 2 and is not an endpoint then the coefficient is going to be 4. From here, to figure out what the coefficient is going to be in front of the function $g(x, y)$, we have to multiply the x coefficient by the y coefficient. When discussing $g(x, y)$, x and y are defined as $x = ax + i * hx$ and $y = ay + j * hy$. From here we can sum our integral together by taking the coefficient and multiplying it by $g(x, y)$. We now need to multiply $g(x, y)$ by h, but since we are in two dimensions, we now have two different values, one for h with respect to x, and one for h with respect to y. Thus, the final step is to take the variable *integration*, which is the sum of $g(x, y)$ multiplied by the coefficients, and multiply it by $\frac{hx*hy}{9}$. This is the final answer for the integral. The last step is to input what values for all of the parameters for $SimpsonsDouble13(g, ax, bx, ay, by, nx, ny)$ and call the function.

4.2 Explanation of Simpson's 3/8 Rule in Two Dimensions

Using the knowledge of Simpson's 3/8 rule in one dimension and the same reasoning with Simpson's 1/3 rule in two dimensions, we were able to compute the two dimensional equation to be:

$$\int_{a_x}^{b_x} \int_{a_y}^{b_y} f(x, y)dydx \approx \frac{9h_x h_y}{64} \sum_{i=0}^{m} \sum_{j=0}^{n} w_{i,j} f(x_{i,j}, y_{i,j}),$$

where $h_x = \frac{bx-ax}{m}$, $h_y = \frac{by-ay}{n}$, $w_{i,j}$ is the weight, or coefficient, that gets multiplied to the function $f(x_{i,j}, y_{i,j})$, $x_{i,j}$ and $y_{i,j}$ represent the x points at i, j and the y points at i, j respectively.The fraction $\frac{9h_x h_y}{64}$ is now 9 over 64 instead of 3 over 8, because we are taking $\left(\frac{3}{8}\right)^2$ to get $\frac{9}{64}$.

Coding Simpson's 3/8 Rule in two dimensions does not differ from Simpson's 1/3 Rule in two dimensions too much when it comes to updating the code to approximate the double integral. The differences lie in that nx and ny, the number of sub-intervals wanted for each x and y, must be divisible by 3, and when determining the coefficients for $g(x, y)$. For Simpson's 3/8 Rule in two dimensions, it is still seen that it is broken into two parts, a part for the x coefficient and a part for the y coefficient. In the for loops if i and j are at one of the endpoints, then the coefficient is going to be equal to 1. If i or j is divisible by 3, meaning the remainder when dividing by 3 is equal to 0, then the coefficient is going to be equal to 2. If i or j is not divisible by 3 and is not an endpoint then the coefficient is going to be 3. From here the only remaining difference in the codes is the value that *integration* is multiplied by at the end. Instead of it being $\frac{hx*hy}{9}$, it is now $\frac{9*hx*hy}{64}$.

5 Mathematical Modeling Example for Simpson's Rule in 2 Dimension

A company is analyzing the temperature distribution across a rectangular metal plate. The temperature, in Fahrenheit, at any point (x, y) on the plate is given by the function $T(x, y) = 100 - x^2 - 2y^2$, where x and y are distances in meters.

The plate's temperature is to be estimated by computing the double integral of $T(x, y)$ over a specific region. Consider the region R bounded by $0 <= x <= 4$ and $0 <= y <= 2$. Define a grid of 6 x 6 points within the region R to approximate the temperature distribution. Using both Simpson's rule in two dimensions to compute an estimate of the average temperature across the plate within the given region R.

5.1 Solving with Simpson's 1/3 Rule

The first step is to define the given temperature equation. Then, we have to modify Simpson's 1/3 Rule in two dimensions that we previously coded, in terms of this model. The function

takes in the temperature equation, the bounds for x and y, and the number of sub-intervals for both x and y. This function calculates the step size for both x and y by taking the upper bound minus the lower bound of each respective variable all divided by the number of sub-intervals for that specific variable. We now need to define a variable in which the value of the integral is going to be stored in, which we call *integration*. This will be set it equal to 0 because if the bounds are all 0 the double integral will be equal to 0. We then calculated the weights, or coefficients, for both x and y that will be multiplied together to create an overall weight that will then be multiplied by the temperature function. We then updated the integration to be the weight multiplied by the temperature function, and then multiplied this by the step sizes for x and y all divided by 9. We set the lower bound for x to 0, the upper bound for x to 4, the lower bound for y to 0, the upper bound for y to 2, the number of sub-intervals for x to 6, and the number of sub-intervals for y to 6. The result we got is 735.9999999999999 degrees Fahrenheit, so the average temperature across the plate within the given region using Simpson's 1/3 Rule for two dimensions is around 736 degrees Fahrenheit.

5.2 Solving with Simpson's 3/8 Rule

The same steps were followed from Simpson's 1/3 Rule in two dimensions to solve this problem except for the fact that we modified Simpson's 3/8 Rule in two dimensions in our code. We again set the lower bound for x to 0, the upper bound for x to 4, the lower bound for y to 0, the upper bound for y to 2, the number of sub-intervals for x to 6, and the number of sub-intervals for y to 6. The result we got is 735.9999999999999 degrees Fahrenheit, so the average temperature across the plate within the given region using Simpson's 3/8 Rule for two dimensions is around 736 degrees Fahrenheit.

5.3 Comparing the Two Rules

We see that we got the same exact answer when we used both Simpson's 1/3 Rule and Simpson's 3/8 Rule in two dimensions. This is due to the fact that we have a higher order of accuracy and the convergent rates are the same (see section 7). Thus both rules are very accurate.

6 Simpson's Rule in Three Dimensions

NOTE: All code is located in the appendix.

Following Simpson's 1/3 Rule and Simpson's 3/8 Rule in two dimensions, our next goal was to figure out how they both would work in three dimensions. We were able to come up with a way to do such calculations of a triple integral using both of Simpson's Rules by taking the two-dimensional code and implementing it onto the other variable, z.

6.1 Explanation of Simpson's 1/3 Rule in Three Dimensions

Using the knowledge of Simpson's 1/3 rule in two dimensions, we computed the three dimensional equation to be:

$$\int_{a_x}^{b_x}\int_{a_y}^{b_y}\int_{a_z}^{b_z} f(x,y,z)dzdydx \approx \frac{h_x h_y h_z}{27}\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{o} w_{i,j,k} f(x_{i,j,k}, y_{i,j,k}, z_{i,j,k}),$$

where $h_x = \frac{bx-ax}{m}$, $h_y = \frac{by-ay}{n}$, $h_z = \frac{bz-az}{o}$ $w_{i,j,k}$ is the weight, or coefficient, that gets multiplied to the function $f(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$, $x_{i,j,k}$, $y_{i,j,k}$, and $z_{i,j,k}$ represent the x points at i, j, k, the y points at i, j, k, and the z points at i, j, k respectively. The fraction $\frac{h_x h_y h_z}{27}$ is now over 27 instead of 9, because we are taking $\left(\frac{1}{3}\right)^3$ to get $\frac{1}{27}$.

To begin coding Simpson's 1/3 Rule in three dimensions, we still carry over the ideas from one dimension and two dimensions. When defining the function $SimpsonsTriple13$ we now have more parameters which are $(g, ax, bx, ay, by, az, bz, nx, ny, nz)$, g is the function

we wish to take the integral of, ax and bx are the limits of integration for x, ay and by are the limits of integration for y, az and bz are the limits of integration for z and lastly nx, ny, and nz are the number of sub-intervals for all three variables, which must be divisible by 2, typically set to the same value. Now when defining the h values, we still have hx and hy equal to the same equations as in the two dimensional case, however, we now have the addition of hz which is equal to $\frac{bz-az}{nz}$. The next step is to find the coefficients, it is the same as the two dimensional case, but again we have the addition of the z component. The coefficients for x, y, and z have the same rules that applied in the two dimensional case. The following step which is summing the components to figure out the approximation of the integral is the same, but we now have to add the z component. Thus we multiply $g(x, y, z)$ by all of the coefficients and add it to our variable *integration* which is summing all the values. The final step is to take the variable *integration* and multiply it by $\frac{hx*hy*hz}{27}$. This is then the final answer for the integral approximation. The last step is to input what values are wanted for all of the parameters for $SimpsonsTriple13$ and call the function.

6.2 Explanation of Simpson's 3/8 Rule in Three Dimensions

Using the knowledge of Simpson's 3/8 rule in two dimensions, we computed the three dimensional equation to be:

$$\int_{a_x}^{b_x}\int_{a_y}^{b_y}\int_{a_z}^{b_z} f(x,y,z)dzdydx \approx \frac{27h_xh_yh_z}{512}\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{o} w_{i,j,k}f(x_{i,j,k}, y_{i,j,k}, z_{i,j,k}),$$

where $h_x = \frac{bx-ax}{m}$, $h_y = \frac{by-ay}{n}$, $h_z = \frac{bz-az}{o}$ $w_{i,j,k}$ is the weight, or coefficient, that gets multiplied to the function $f(x_{i,j,k}, y_{i,j,k}, z_{i,j,k})$, $x_{i,j,k}$, $y_{i,j,k}$, and $z_{i,j,k}$ represent the x points at i, j, k, the y points at i, j, k, and the z points at i, j, k respectively. The fraction $\frac{27h_xh_yh_z}{512}$ is now over 512 instead of 8, because we are taking $\left(\frac{3}{8}\right)^3$ to get $\frac{27}{512}$.

The same as in the two dimensional case, there is not many things that are needed to be updated in our code in order to make the Simpson's 1/3 Rule in Three Dimensions into

Simpson's 3/8 Rule. The differences lie again when figuring out what coefficients are to be used. Similarly, as in the Simpson's 3/8 Rule in Two Dimensions, the x, y, and now z coefficients are found with the same rules that applied then. Note that a third for loop was added in order to account for the z variable. The only difference between Simpson's 1/3 Rule and Simpson's 3/8 Rule in three dimensions that is left to discuss is what factor to multiply the variable *integration* by at the end. The new factor is $\frac{27*hx*hy*hz}{512}$. Finally, the last step is to call the function named $SimpsonsTriple38$ and input the desired parameters. Note that the number of sub-intervals, n must still be divisible by three in order to get a correct approximation.

7 Convergence Rates

We decided it was important to take a look at all six rules that we created and calculate the convergence rate of each one. The code for this works the same in all cases. For the three functions in the three separate dimensions that use Simpson's 1/3 Rule, we changed the number of sub-intervals to be $10, 20, 40, 80$, and then 160 and noted what the output values are for the function. For the three functions in the three separate dimensions that use Simpson's 3/8 Rule, we changed the number of sub-intervals to be $9, 18, 36, 72$, and then 144 we noted what the output values are for the function. The reason that the number of sub-intervals are different between the two different Simpson's Rules is because of the two different properties that the number of sub-intervals have to have in each of the two rules. For Simpson's 1/3 Rule, the number of sub-intervals must be divisible by two while in Simpson's 3/8 Rule, they must be divisible by three. We used a calculator to find the exact value that the integral would be for each of the equations that were looked at. Following that we created an error vector that was calculated by taking the absolute value of the approximated value and subtracting from it the true value at each of the sub-interval values. From here we were able to use code that we previously had done in homework problems and class to

find the convergence rate of each method.

In the one dimensional case, we found the convergence rate of Simpson's 1/3 Rule to be 2 while Simpson's 3/8 Rule was 4. In the two dimensional case, we found that the convergence rate for both methods was 4. Similarly, we found the convergence rate for both methods in three dimensions to be 4 as well. In the one dimensional case, it is easy to see through different functions and sub-interval values that Simpson's 3/8 Rule gave a better approximation each time. The convergence rates found in this case show that there is a reasoning behind this, as Simpson's 3/8 Rule has a larger convergence rate. When moving into two and three dimensions, the convergence rates then equal each other explaining why the approximations of each method were not far off from each other.

8 Conclusion

We were able to successfully create a program that can approximate the integral of a function in one dimension, two dimensions, and three dimensions using Simpson's 1/3 Rule and Simpson's 3/8 Rule. Even though there are numerous websites out there to give the formula for the two rules in one dimension, it took a while to see the correlation to the two dimensional and three dimensional case since the formula for such calculations is hard to find. Once we were able to see how the dimensions were based off of each other and how to add to the program in order to add more dimensions, it made sense as to what was happening and why. We were also able to successfully calculate the convergence rates of the functions by creating error vectors from the data the created functions gave and what the true solution is equal to. We also saw how one dimensional and two dimensional Simpson's 1/3 Rule and Simpson's 3/8 Rule are able to solve mathematical modeling questions.

References

[1] Difference between simpsons rule and 3/8 rule. Accessed: November 29, 2023.

[2] Byju's. Simpson's rule. Accessed: November 27, 2023.

[3] freeCodeCamp. Simpson's rule: the formula and how it works. Accessed: November 28, 2023.

[4] Geeks For Geeks. Difference between simpson 's 1/3 rule and 3/8 rule. Accessed: November 27, 2023.

[5] Quingkai Kong, Timmy Shiauw, and Alexandre Bayen. *Python Programming and Numerical Methods*. Elsevier, 2020. Accessed: November 28, 2023.

[6] C. D. Lane. Simpson rule with python. Accessed: November 27, 2023.

Appendix

Listing 1: Python code for Simpson's 1/3 Rule in 1 Dimension.

```
import numpy as np
#Define the function you want to integrate
def f(x):
    return np.exp(-x)*np.sin(x)

#Defining Simpson's 1/3 Rule
def Simpson13(f,a,b,n):
    h = ((b - a)/(n))     #step size

    #Give an intial value to what the answer is going to be
    #It is the following because the first and the last values are
    #not multiplied by any value
    integration = f(a) + f(b)



    #Creating a for loop to figure out what terms get
    #multiplied by what value
    #For the odd points:
    for i in range(1,n,2):
        integration += 4.0 * f(a + i * h)

    #For the even points
    for i in range(1,n-1,2):
        integration += 2.0 * f(a + i * h)
```

```
    integration *= h / 3.0

    return integration

lowLim = 0
upLim = 10
subInt = 160

result = Simpson13(f,lowLim,upLim,subInt)

print("The 1 dimensional approximation using Simpson's 1/3 Rule: ",
result)
```

Listing 2: Python code for Simpson's 1/3 Rule Convergence Rate in 1 Dimension.

```
import numpy as np
ErrorVec = [0.12147566, 0.03911301, 0.01026395, 0.00259551, 0.00065092]
def GetConvergence(vec,factor=2):
    for i in range(len(vec)-1):
        print("Convergence =", (np.log(vec[i]/vec[i+1]))/np.log(factor))
GetConvergence(ErrorVec)
```

Listing 3: Python code for Simpson's 3/8 Rule in 1 Dimension.

```
import numpy as np
#Define the function you want to integrate
def f(x):
    return np.exp(-x)*np.sin(x)

#Defining Simpson's 3/8 Rule
```

```
def Simpson38(f,a,b,n):
    h = ((b - a)/(n))    #step size

    #Give an intial value to what the answer is going to be
    #It is the following since the first and the last values are not
    #multiplied by any value
    integration = f(a) + f(b)

    #Creating a for loop to figure out what terms get multiplied by
    #what value
    for i in range(1,n):
        #if i is divisible by 3 then do the following
        if i%3 == 0:
            integration += 2 * f(a + i*h)
        #if i is not divisible by 3 do the following
        else:
            integration += 3 * f(a + i*h)

    #Multiply the integration value by 3h/8
    integration = integration * 3 * h / 8

    return integration

lowLim = 0
upLim = 10
subInt = 144
```

```
result = Simpson38(f,lowLim, upLim, subInt)
print("The 1 dimensional approximation using Simpson's 3/8 Rule is: ",
result)
```

Listing 4: Python code for Simpson's 3/8 Rule Convergence Rate in 1 Dimension.

```
import numpy as np
ErrorVec2 = [.0455739712, .0026766908, .0001541234682, .00000938804861,
.00000058283454]
def GetConvergence(vec,factor=2):
    for i in range(len(vec)-1):
        print("Convergence =", (np.log(vec[i]/vec[i+1]))/np.log(factor))
GetConvergence(ErrorVec2)
```

Listing 5: Python code for Mathematical Modeling Example Using Simpson's 1/3 Rule in 1 Dimension.

```
import numpy as np
#Defining the time intervals and corresponding population counts
time_int = [0, 1, 2, 3, 4, 5]
pop_count = [100, 120, 150, 200, 180, 250]

#Defining a function that interpolates between the data points
def interpolate_pop(t):
    #Checking to see if the time t matches any recorded time
    if t in time_int:
        return pop_count[time_int.index(t)]

    #If not, do a linear interpolation to estimate the population at time t
    else:
```

```
        for i in range(len(time_int) - 1):
            if time_int[i] < t < time_int[i + 1]:
                #Performing a linear interpolation between two closest known
                time_diff = time_int[i + 1] - time_int[i]
                pop_diff = pop_count[i + 1] - pop_count[i]
                slope = ((pop_diff)/(time_diff))
                interpolated_pop = pop_count[i] + slope * (t - time_int[i])
                return interpolated_pop

#Defining the function for population growth over time using the interpolatio
def pop_growth(t):
    return interpolate_pop(t)

#Modifing the Simpson's 1/3 Rule function to use the population growth
def Simpson13(t0,tn,n):
    h = ((tn - t0)/(n))

    integration = pop_growth(t0) + pop_growth(tn)

    for i in range(1,n,2):
        integration += 4.0 * pop_growth(t0 + i * h)
    for i in range(1,n-1,2):
        integration += 2.0 * pop_growth(t0 + i * h)

    integration *= h / 3.0

    return integration
```

```
t0 = 0  #Starting time
tn = 4.5  #Ending time
n = 150 #Number of sub-intervals

result = Simpson13(t0, tn, n)
print("The estimated total population growth using Simpson's 1/3 Rule:", res
```

Listing 6: Python code for Mathematical Modeling Example Using Simpson's 3/8 Rule in 1 Dimension.

```
import numpy as np
#Defining the time intervals and corresponding population counts
time_int = [0, 1, 2, 3, 4, 5]
pop_count = [100, 120, 150, 200, 180, 250]

#Defining a function that interpolates between the data points
def interpolate_pop(t):
    #Checking to see if the time t matches any recorded time
    if t in time_int:
        return pop_count[time_int.index(t)]

    #If not, do a linear interpolation to estimate the population at time t
    else:
        for i in range(len(time_int) - 1):
            if time_int[i] < t < time_int[i + 1]:
                #Performing a linear interpolation between two closest known
                time_diff = time_int[i + 1] - time_int[i]
```

```
            pop_diff = pop_count[i + 1] - pop_count[i]
            slope = ((pop_diff)/(time_diff))
            interpolated_pop = pop_count[i] + slope * (t - time_int[i])
            return interpolated_pop

#Defining the function for population growth over time using the interpolatio
def pop_growth(t):
    return interpolate_pop(t)

#Modifing the Simpson's 3/8 Rule function to use the population growth
def Simpson38(t0,tn,n):
    h = ((tn - t0)/(n))

    integration = pop_growth(t0) + pop_growth(tn)

    for i in range(1,n):
        if i%3 == 0:
            integration += 2 * pop_growth(t0 + i*h)
        else:
            integration += 3 * pop_growth(t0 + i*h)

    integration *=  (3 * h) / 8

    return integration

t0 = 0  #Starting time
tn = 4.5  #Ending time
```

```
n = 150 #Number of sub-intervals

result = Simpson38(t0, tn, n)
print("The estimated total population growth using Simpson's 3/8 Rule:", res
```

Listing 7: Python code for Simpson's 1/3 Rule in 2 Dimension.

```
import numpy as np
#Define the function you want to integrate
def g(x, y):
    return np.exp(-x*y) + np.sin(x) - np.cos(y)

#Defining Simpson's 1/3 Rule for two dimensions
def SimpsonsDouble13(g, ax, bx, ay, by, nx, ny):
    hx = (bx - ax) / nx
    hy = (by - ay) / ny
    #Set the summing variable equal to 0
    integration = 0.0

#Creating for loops to figure out the coefficients
    for i in range(0,nx+1):
        for j in range(0, ny+1):
            x = ax + i * hx
            y = ay + j * hy

            #end points
            if i==0 or i==nx:
                coefficientx =  1.0
```

```
            #divisible by 2
            elif i%2 == 0:
                coefficientx =  2.0
            # not an end point or divisible by 2
            else:
                coefficientx =  4.0

            if j==0 or j==ny:
                coefficienty =  1.0
            elif j%2 ==0:
                coefficienty =  2.0
            else:
                coefficienty =  4.0
            #calculating the whole coefficient
            coefficient = coefficientx * coefficienty
            #multiply the coefficient by the function
            integration += coefficient*g(x,y)

    #multiply the integration value by (hx*hy)/9
    integration *= (hx * hy) / 9.0

    return integration

# Define the region of integration [ax, bx] x [ay, by]
ax, bx = 0, 3       # Limits in x
ay, by = 1, 5       # Limits in y
nx, ny = 10,10      # Number of divisions x and y must be divisible by 2
```

```
result = SimpsonsDouble13(g,ax,bx,ay,by,nx,ny)
print("The 2 dimensional approximation using Simpson's 1/3 Rule is:",
result)
```

Listing 8: Python code for Simpson's 1/3 Rule Convergence Rate in 2 Dimension.

```
import numpy as np
ErrorVec = [.0073704237, .000508831031, .000032678638, .000002056683,
.000000128768]
def GetConvergence(vec,factor=2):
    for i in range(len(vec)-1):
        print("Convergence =", (np.log(vec[i]/vec[i+1]))/np.log(factor))
GetConvergence(ErrorVec)
```

Listing 9: Python code for Simpson's 3/8 Rule in 2 Dimension.

```
import numpy as np
#Define the function you want to integrate
def g(x, y):
    return np.exp(-x*y) + np.sin(x) - np.cos(y)

#Defining Simpson's 3/8 Rule for two dimensions
def SimpsonDouble38(g,ax,bx,ay,by,nx,ny):
    hx = (bx - ax) / nx
    hy = (by - ay) / ny
    #Set the summing variable equal to 0
    integration = 0.0
    #Creating for loops to figure out the coefficients
    for i in range(0,nx+1):
```

```
    for j in range(0, ny+1):
        x = ax + i * hx
        y = ay + j * hy

        #end points
        if i==0 or i==nx:
            coefficientx = 1.0
        #divisible by 3
        elif i%3 == 0:
            coefficientx = 2.0
        # not an endpoint or divisible by 3
        else:
            coefficientx = 3.0

        if j==0 or j==ny:
            coefficienty = 1.0
        elif j%3 ==0:
            coefficienty = 2.0
        else:
            coefficienty = 3.0
        #calculating the whole coefficient
        coefficient = coefficientx * coefficienty
        #multiply the coefficient by the function
        integration += coefficient*g(x,y)
#multiply the integration value by (9*hx*hy)/64
integration *= (9 * hx * hy)/64
return integration
```

```
# Define the region of integration [ax, bx] x [ay, by]
ax, bx = 0, 3        # Limits in x
ay, by = 1, 5        # Limits in y
nx, ny = 144,144      # Number of divisions x and y must be divisible by 3

result =  SimpsonDouble38(g,ax,bx,ay,by,nx,ny)
print("The 2 dimensional approximation using Simpson's 3/8 Rule is: ",
result)
```

Listing 10: Python code for Simpson's 3/8 Rule Convergence Rate in 2 Dimension.

```
import numpy as np
ErrorVec2 = [.0216705439, .0016602848, .000110575533, .000007029057,
.000000441212]
def GetConvergence(vec,factor=2):
    for i in range(len(vec)-1):
        print("Convergence =", (np.log(vec[i]/vec[i+1]))/np.log(factor))
GetConvergence(ErrorVec2)
```

Listing 11: Python code for Mathematical Modeling Example Using Simpson's 1/3 Rule in 2 Dimension.

```
import numpy as np
#Defining the given temperature equation
def T(x,y):
    return 100 - x**2 - 2*y**2

#Modifing the Simpson's 1/3 Rule function to use the temperature equation
def SimpsonsDouble13(T,ax,bx,ay,by,nx,ny):
```

```
hx = ((bx - ax)/nx)
hy = ((by - ay)/ny)

integration = 0.0

for i in range(0,nx+1):
    for j in range(0, ny+1):
        x = ax + i * hx
        y = ay + j * hy

        if i==0 or i==nx:
            coefficientx = 1.0
        elif i%2 == 0:
            coefficientx = 2.0
        else:
            coefficientx = 4.0

        if j==0 or j==ny:
            coefficienty = 1.0
        elif j%2 ==0:
            coefficienty = 2.0
        else:
            coefficienty = 4.0

        coefficient = coefficientx * coefficienty
        integration += coefficient*T(x,y)
```

```
    integration *= (hx * hy) / 9.0

    return integration

#Define the region of integration 0<=x<=4 x 0<=y<=2
ax, bx = 0, 4       #Bounds on x
ay, by = 0, 2       #Bounds on y
nx, ny = 6, 6       #Number of sub-intervals x and y, must be divisible by 2

result = SimpsonsDouble13(T, ax, bx, ay, by, nx, ny)
print("The average temperature across the plate witin the given region using
```

Listing 12: Python code for Mathematical Modeling Example Using Simpson's 3/8 Rule in 2 Dimension.

```
import numpy as np
#Defining the given temperature function
def T(x,y):
    return 100 - x**2 - 2*y**2

#Modifing the Simpson's 3/8 Rule function to use the temperature equation
def SimpsonDouble38(T,ax,bx,ay,by,nx,ny):
    hx = ((bx - ax)/nx)
    hy = ((by - ay)/ny)

    integration = 0.0

    for i in range(0,nx+1):
```

```
        for j in range(0, ny+1):
            x = ax + i * hx
            y = ay + j * hy

            if i==0 or i==nx:
                coefficientx =  1.0
            elif i%3 == 0:
                coefficientx =  2.0
            else:
                coefficientx =  3.0

            if j==0 or j==ny:
                coefficienty =  1.0
            elif j%3 ==0:
                coefficienty =  2.0
            else:
                coefficienty =  3.0

            coefficient = coefficientx * coefficienty
            integration += coefficient*T(x,y)

    integration *= (9 * hx * hy)/64

    return integration

#Define the region of integration 0<=x<=4 x 0<=y<=2
ax, bx = 0, 4       #Limits in x
```

```
ay, by = 0, 2        #Limits in y
nx, ny = 6, 6        ##Number of sub-intervals x and y, must be divisible by 3

result = SimpsonDouble38(T,ax,bx,ay,by,nx,ny)
print("The average temperature across the plate witin the given region using
```

Listing 13: Python code for Simpson's 1/3 Rule in 3 Dimension.

```
import numpy as np
#Correct Running Code for Simpsons 1/3 in Three Dimensions

#Define the function you want to integrate
def g(x, y, z):
    return np.exp(x*y) + np.sin(z)

#Defining Simpson's 1/3 Rule for three dimensions
def SimpsonsTriple13(g, ax, bx, ay, by, az, bz,  nx, ny, nz):
    hx = (bx - ax) / nx
    hy = (by - ay) / ny
    hz = (bz - az) / nz
    #Set the summing variable equal to 0
    integration = 0.0

    #Creating for loops to figure out the coefficients
    for i in range(0,nx+1):
        for j in range(0, ny+1):
            for k in range (0, nz+1):
                x = ax + i * hx
```

```
y = ay + j * hy
z = az + k * hz

#end points
if i==0 or i==nx:
  coefficientx =  1.0
#divisible by 2
elif i%2 == 0:
  coefficientx =  2.0
#not an end point or divisible by 2
else:
  coefficientx =  4.0

if j==0 or j==ny:
  coefficienty =  1.0
elif j%2 ==0:
  coefficienty =  2.0
else:
 coefficienty =  4.0

if k==0 or k==nz:
  coefficientz = 1.0
elif k%2 == 0:
  coefficientz = 2.0
else:
  coefficientz = 4.0
```

```
                #calculating the whole coefficient
                coefficient = coefficientx * coefficienty * coefficientz
                #multiply the coefficient by the function
                integration += coefficient*g(x,y,z)

    #multiply the integration value by (hx*hy)/27
     integration *= (hx * hy * hz) / 27.0

     return integration

# Define the region of integration [ax, bx] x [ay, by] x [az,bz]
ax, bx = 0, 2       # Limits in x
ay, by = 0, 1       # Limits in y
az, bz = 0, 1       # Limits in z
nx, ny, nz  = 160,160,160    # Number of divisions x and y and z

result = SimpsonsTriple13(g, ax, bx, ay, by, az, bz,  nx, ny, nz)
print("The 3 dimensional approximation using Simpson's 1/3 Rule is:",
result)
```

Listing 14: Python code for Simpson's 1/3 Rule Convergence Rate in 3 Dimension.

```
import numpy as np
ErrorVec = [.0000490905509, .0000011960549, .0000000748012,
.0000000046762, .0000000002921]
def GetConvergence(vec,factor=2):
    for i in range(len(vec)-1):
        print("Convergence =", (np.log(vec[i]/vec[i+1]))/np.log(factor))
```

```
GetConvergence(ErrorVec)
```

Listing 15: Python code for Simpson's 3/8 Rule in 3 Dimension.

```
import numpy as np
#Define the function you want to integrate
def g(x, y, z):
    return np.exp(x*y) + np.sin(z)

#Defining Simpson's 3/8 Rule for three dimensions
def SimpsonsTriple38(g, ax, bx, ay, by, az, bz,  nx, ny, nz):
    hx = (bx - ax) / nx
    hy = (by - ay) / ny
    hz = (bz - az) / nz
    #Set the summing variable equal to 0
    integration = 0.0

  #Creating for loops to figure out the coefficients
    for i in range(0,nx+1):
        for j in range(0, ny+1):
            for k in range (0, nz+1):
                x = ax + i * hx
                y = ay + j * hy
                z = az + k * hz

                #end points
                if i==0 or i==nx:
                  coefficientx =  1.0
```

```
            #divisible by 3
            elif i%3 == 0:
              coefficientx =  2.0
            #not an endpoint or divisible by 3
            else:
              coefficientx =  3.0

            if j==0 or j==ny:
              coefficienty =  1.0
            elif j%3 ==0:
              coefficienty =  2.0
            else:
              coefficienty =  3.0

            if k==0 or k==nz:
              coefficientz = 1.0
            elif k%3 == 0:
              coefficientz = 2.0
            else:
              coefficientz = 3.0

            #calculating the whole coefficient
            coefficient = coefficientx * coefficienty * coefficientz
            #multiply the coefficient by the function
            integration += coefficient*g(x,y,z)

    #multiply the integration value by (27*hx*hy)/512
```

```
    integration *= (27.0*hx * hy * hz) / 512.0

    return integration

# Define the region of integration [ax, bx] x [ay, by] x [az,bz]
ax, bx = 0, 2       # Limits in x
ay, by = 0,1        # Limits in y
az, bz = 0,1        # Limits in z
nx, ny, nz  = 144,144,144  # Number of divisions x and y and z and
#they must be divisible by 3

result = SimpsonsTriple38(g, ax, bx, ay, by, az, bz,  nx, ny, nz)
print("The 3 dimensional approximation using Simpson's 3/8 Rule is:",
result)
```

Listing 16: Python code for Simpson's 3/8 Rule Convergence Rate in 3 Dimension.

```
import numpy as np
ErrorVec2 = [.0000651648105, .0000040966307, .0000002564385,
.000000016034, .0000000010024]
def GetConvergence(vec,factor=2):
    for i in range(len(vec)-1):
        print("Convergence =", (np.log(vec[i]/vec[i+1]))/np.log(factor))
GetConvergence(ErrorVec2)
```