MA348 Numerical Analysis, Integration

David Jefts

April 24th, 2019

Introduction

The purpose of this lab is to use compare the effectiveness of different algorithms at estimating the value of an integral, $\int_a^b f(x)\,\mathrm{d}x$, where a is 0, b is 0.8, and $f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$.

$$\int_{0.0}^{0.8} 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5\,\mathrm{d}x$$

The two methods used to solve the above definite integral were the Newton-Cotes Trapezoidal Method and the Newton-Cotes Simpson method. The Trapezoid method was used with 2-, 4-, and 6-subintervals, while the Simpson method was used with five subintervals.

Theory-Analysis

The Newton-Cotes Trapezoidal method is an integration method based on splitting up an integral into n subintervals, drawing a straight line between each pair of $f(x_n)$ and $f(x_{n+1}$ values to create a series of trapezoids. These trapezoids can then be used to estimate the integral value of the function. The area of a trapezoid can be represented by the function $A = \frac{h}{2}left(a+b)$ where h is the width of the trapezoid, a is the height of the left trapezoid edge, and b is the height of the right trapezoid edge. The fourth edge of the trapezoid is not needed to find the area. By replacing a with x_i and b with x_{i+1} then creating a summation from $i = 0$ to $i = n$, the function can be rewritten as:

$$I = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right]$$

The middle sum term of the bracketed equation is multiplied by two because each subinterval's y-value will be used twice, except for the first and

last points.

Simpson's integral estimation equation is an extension of the Newton-Cotes method, but adds the midpoint of each subinterval into the estimation function to make the estimate more accurate by using three points on each subinterval instead of two. It uses a Lagrange polynomial interpolation to generate a standard second-degree curve along these three points and then estimates the area. The function for this interpolated line is much more complicated than the trapezoid area function:

$$y_i = f(x_i)\frac{(x - x_{i+1})(x - x_{i+2})}{(x_i - x_{i+1})(x_i - x_{i+2})} + f(x_{i+1})\frac{(x - x_{i+1})(x - x_{i+2})}{(x_{i+1} - x_i)(x_{i+1} - x_{i+2})} + f(x_{i+2})\frac{(x - x_{i+1})(x - x_{i+2})}{(x_{i+2} - x_i)(x_{i+2} - x_{i+1})}$$

When summed from $i = 0$ to $i = n$, the equation is simplified to:

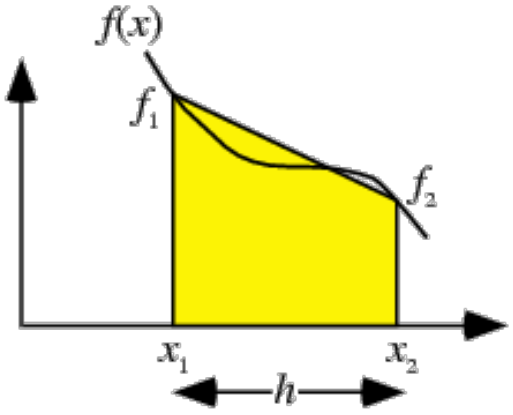
$$I = \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,5\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,6\ldots}^{n-1} f(x_i) + f(x_n)\right]$$

For the purpose of testing it was assumed that the actual value of the integral is 1.640533333.

Numerical Solution

All three integration methods were coded in Python, with this report created in LaTeX.

The Newton-Cotes Trapezoidal method is a method of integration based upon breaking up the integral into multiple subintervals and then drawing trapezoids that estimate the area under the function curve, as shown below:



$f(x)$ is the function line, h is the width of each subinterval (also known as the δx), x_1 and x_2 are the endpoints of the subinterval and f_1 and f_2 are the corresponding function values. The yellow area is the trapezoid that represents the integral estimation for that subinterval and is represented by:

$$I = \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right]$$

This is a summation of all of the trapezoids between each limit of the integral when there is n subintervals with each x value representing the vertical lines dividing the subintervals. h is equivalent to a

The Simpson formula is very similar to the Trapezoidal formula, except that it uses the midpoint of each subinterval as well as the end points when estimating the integral value.

$$I = \frac{h}{3}\left[f(x_0) + 4\sum_{i=1,3,5\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,6\ldots}^{n-1} f(x_i) + f(x_n)\right]$$

Results and Discussion

The table depicting the various integral values is shown below:

Method	**Value**	**Percentage Error**
By-Hand Integration	1.6405333333333333	0.0000%
Trapezoid Rule (2 points)	0.5312000000000001	67.620%
Trapezoid Rule (4 points)	0.6240000000000006	61.964%
Trapezoid Rule (6 points)	0.8981816186556900	45.251%
Trapezoid Rule (8 points)	1.1246500000000006	31.446%
Simpson Rule (4 points)	0.587733333333333	64.174%
Simpson Rule (6 points)	0.9335601280292622	43.094%
Simpson Rule (8 points)	1.1645000000000005	29.017%

Based on these results, the Simpson Method converges faster and generally has a lower error when using a similar amount of points as the Trapezoid Rule. However, the error for the Simpson method increases by 20-30% when there is an odd number of points, so on average the Trapezoid Rule is more effective. However when the number of points is predetermined to be even then the Simpson Method will be more accurate.

Conclusions

Both the Simpson Method and the Trapezoid Rule are very effective at estimating definite integral functions but the Simpson Method routinely resulted in more accurate integrals. To contrast that, the Trapezoidal Rule was considerably easier to implement and works correctly regardless of the number of subintervals, while the Simpson Method requires an even number of points to produce an accurate measurement. In conclusion, the Simpson Method is more accurate, but only when the number of subintervals is even.

Appendix A

```
4   def f(x):
5       return 0.2 + 25 * x - 200 * x**2 + 675 * x**3 - 900 * x**4 + 400 * x**5
6       # return exp(3 * x) * sin(2 * x)
7
8
9   def trap_rule(n, h, x):
10      n -= 1
11      two = 0
12      for i in range(1, n - 1):
13          two += f(x[i])
14
15      return (h / 2) * (f(x[0]) + 2 * two + f(x[n]))
16
17
18  def simpson(n, h, x):
19      n -= 1
20      two = 0
21      for i in range(1, n - 1, 2):
22          two += f(x[i])
23      three = 0
24      for i in range(2, n - 2, 2):
25          three += f(x[i])
26      return (h / 3) * (f(x[0]) + 4 * two + 2 * three + f(x[n]))
27
28
29  a = 0.0
30  b = 0.8
31  t = 4
32  h = (b - a) / t
33  x = [0]
34  for i in range(t - 1):
35      x.append(x[len(x) - 1] + h)
36  print("%d x values:" % len(x), x)
37
38  actual = 1.640533333
39  trap = trap_rule(t, h, x)
40  simp = simpson(t, h, x)
41
42  print("\nExpected:", actual)
43  print("Trapezoid:", trap)
44  print("Simpson:", simp)
45  print("\nTrapezoid Error:", abs(trap-actual)/actual * 100)
46  print("Simpson Error:", abs(simp-actual)/actual * 100)
```