# Numerical Integration Methods

Andrew Struthers

May 2022

## 1 Introduction

In this project, we were expected to use numerical integration methods to estimate the integral of

$$g(x) = \frac{1}{1+e^{-3x}}$$

on the interval $[-1, 3]$. We will use Compozite Trapezoidal and Composite Simpson's methods of numerical integration.

## 2 Analysis

First, we will calculate the integral of $g(x)$ exactly. Observe:

$$\begin{aligned} g(x) &= \int_{-1}^{3} \frac{1}{1+e^{-3x}}\,dx \\ &= \int_{-1}^{3} \frac{e^{3x}}{1+e^{3x}}\,dx \end{aligned} \tag{1}$$

Now we do u-substitution, so let $u = e^{3x}+1$. Then $\frac{du}{dx} = 3e^{3x}$ and $dx = \frac{3^{-3x}}{3}du$. Substituting back into the equation above we have:

$$\begin{aligned} &= \int_{-1}^{3} \frac{e^{3x}}{1+e^{3x}} * \frac{e^{-3x}}{3}\,du \\ &= \frac{1}{3}\int_{-1}^{3} \frac{1}{1+e^{3x}}\,du \\ &= \frac{1}{3}\int_{-1}^{3} \frac{1}{u}\,du \end{aligned} \tag{2}$$

$$
\begin{aligned}
&= \frac{1}{3}ln(u)\Big|_{-1}^{3} \\
&= \frac{ln(u)}{3}\Big|_{-1}^{3} \\
&= \frac{ln(e^{3x}+1}{3}\Big|_{-1}^{3} \\
&= \frac{ln(e^{3(3)})}{3} - \frac{ln(e^{3(-1)})}{3} \\
&\approx 2.98384329566880
\end{aligned} \tag{3}
$$

Looking at the error term for the Composite Trapezoid Method, we have the formula:

$$\frac{g''(\xi)h^2(b-a)}{12}$$

where $a \le \xi \le b$ and $h$ is the step size between our bounds. We will be looking at 3 cases, $n = 11, n = 41$, and then calculating a value of $n$ such that our error is less than $10^{-4}$. The second derivative of $g(x)$ has a maximum value of $\approx 0.866$. The corresponding $h$-value for 11 points is $h = 0.4$. Therefore the error is bounded by $\frac{0.866*0.4^2*(3-(-1)}{12} \approx 0.0462$. Likewise, the $h$-value for $n = 41$ is $h = 0.1$, so the error is bounded by $\frac{0.866*0.1^2*4}{12} \approx 0.00289$. Using the above error formula, we can calculate the appropriate $h$-value such that our error is less than $10^{-4}$. We get $h = \sqrt{\frac{10^{-4}*12}{(3-(-1))0.866}} \approx 0.0186$. We can get our number of points by taking $n = \frac{3-(-1)}{0.0186} + 1$ and rounding it up to the nearest whole number to get $n = 216$. This is the number of points needed to get an error less than $10^{-4}$ using the Composite Trapezoidal method.

Similarly, looking at the error term for the Composite Simpson's Method, we have the formula:

$$\frac{g''''(\xi)h^4(b-a)}{180}$$

where $a \le \xi \le b$ and $h$ is the step size between our bounds. We will again be looking at 3 cases, $n = 11, n = 41$, and then calculating a value of $n$ such that our error is less than $10^{-4}$. The fourth derivative of $g(x)$ has a maximum value of $\approx 10.342$. The corresponding $h$-value for 11 points is $h = 0.4$. Therefore the error is bounded by $\frac{10.342*0.4^4*(3-(-1)}{180} \approx 0.00588$. Likewise, the $h$-value for $n = 41$ is $h = 0.1$, so the error is bounded by $\frac{10.342*0.1^4*4}{180} \approx 0.000023$. Using the above error formula, we can calculate the appropriate $h$-value such that our error is less than $10^{-4}$. We can see from $n = 41$ that our error bound is already less than $10^{-4}$, so we should expect an $n$-value less than 41. We get $h = \sqrt[4]{\frac{10^{-4}*180}{(3-(-1))10.342}} \approx 0.1444$. We can get our number of points by taking $n = \frac{3-(-1)}{0.1444} + 1$ and rounding it up to the nearest whole number to get $n = 29$. This is the number of points needed to get an error less than $10^{-4}$ using the Composite Simpson's method.

We know that the Composite Simpson's method will be more accurate than the Composite Trapezoid method due to the error term. We can see in the error term for the Simpson's method that it behaves like $O(h^4)$ whereas the Trapezoid error behaves like $O(h^2)$. As $h \to 0$, the error for Simpson's will go to zero faster than for Trapezoid. This means that we can get a smaller error faster using Simpson's method, so we know that the Simpson's method is more accurate. Because of this and the analysis of our step sizes and their resulting error bounds found above, I am going to predict that the Simpson's will be much better of an approximation than the Trapezoid, and it will use less points to get an answer within the tolerance.

## 3 Computer Program

In Figure 1, we can see the output of the code when computing the integral with the Composite Trapezoidal Method with $n = 11, n = 41, n = 216$ points. Notice the only value that was within tolerance was $n = 216$.

```
=-=-=-=-=-=
Composite Trapezoidal:
=-=-=-=-=-=
Required h = 0.0186121401926444
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
  n          h     Real Value    Calculated Value    Absolute Error    Relative Error   Within Tolerance
---  ---------   ------------   -----------------   ---------------   ---------------   -----------------
 11  0.4             2.98385             2.98207        0.00177056        0.0593382     False
 41  0.1             2.98385             2.98373        0.00011251        0.00377064    False
216  0.0186047       2.98385             2.98384        3.89846e-06       0.000130652   True

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

Figure 1: Composite Trapezoidal Method at $n = 11, n = 41, n = 216$

In Figure 2, we can see the output of the code when computing the integral with the Composite Simpson's Method with $n = 11, n = 41, n = 29$ points.

```
=-=-=-=-=-=
Composite Simpson's:
=-=-=-=-=-=
Required h = 0.144427164687629
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
  n         h     Real Value    Calculated Value    Absolute Error    Relative Error  Within Tolerance
---  --------  ------------  ------------------  ----------------  ----------------  ------------------
 11  0.4            2.98385              2.9841       0.000251352        0.00842375   False
 41  0.1            2.98385             2.98384       4.92819e-07       1.65162e-05   True
 29  0.142857       2.98385             2.98384       2.05454e-06       6.88553e-05   True

=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

Figure 2: Composite Simpson's Method at $n = 11, n = 41, n = 29$

# 4 Results

Hello

# References


R.L. Burden and J.D. Faires. *Numerical Analysis*. Cengage Learning, 2010. ISBN 9781133169338. URL https://books.google.com/books?id=Dbw8AAAAQBAJ.

J.F. Epperson. *An Introduction to Numerical Methods and Analysis*. Wiley, 2013. ISBN 9781118626238. URL https://books.google.com/books?id=O1U5W5hzvCoC.

```python
import sys
import math
import sympy as sp
import numpy as np
from tabulate import tabulate

#calculate the nth derivative of fx
#returns a list of [fx, fx', fx'', ... , fx^(n)]
def n_deriv(fx, n):
   x = sp.symbols('x')
   f = fx
   deriv_list = [f]
   for i in range(1, n + 1):
         df_i = deriv_list[-1].diff(x).replace(sp.Derivative, lambda *args: f(x))
         deriv_list.append(df_i)
   return deriv_list

#takes a function, bounds, and step size and finds the x value where the function is maximized
def find_max_error(f_n, bounds, step):
   max_y = None
   max_x = None
   x = sp.symbols('x')
   for i in np.arange(bounds[0], bounds[1] + step, step):
      y = abs(f_n.subs(x, i))
      if (max_y is None and max_x is None) or y > max_y:
         max_y = y
         max_x = i
   return max_x

#takes parameters h, n, f, a, b, and a boolean that checks if we want Simpson's method or not
#returns the calculated numerical integral of f
def numerical_integral(h, n, f, a, b, simpsons):
   x = sp.symbols('x')
   integral = 0

   #iterate through all points
   for i in range(n):
      #calculate xi by taking the start point + step size * current index
      xi = a + i*h
      xi = round(xi, 10)

      #checks to see if we want to use Simpson's method or not
      if simpsons:
         #i==0 or i==n-1 corresponds to the two end points
         if i == 0 or i == n - 1:
            integral += f.subs(x, xi)
         #multiply the even terms by 2
         elif i%2 == 0:
            integral += 2 * f.subs(x, xi)
         #multiply the odd terms by 4
         else:
            integral += 4 * f.subs(x, xi)
      #if not using Simpson's, we are using Trapezoidal method
      else:
         #check for end points
         if i == 0 or i == n - 1:
            integral += f.subs(x, xi)
         #multiply non-end points by 2
         else:
            integral += 2 * f.subs(x, xi)

   if simpsons:
      integral *= h/3
   else:
      integral *= h/2
```

```python
    return integral

"""
defining our initial variables
"""
x = sp.symbols('x')

#define our function 1/(1+e^(-3x))
f = sp.sympify(1/(1+sp.exp(-3*x)))

tolerance = float(input("Enter max tolerance: "))
a = float(input("a = "))
b = float(input("b = "))

real = sp.integrate(f, (x, a, b))

"""
finds where the second and fourth derivatives are maximized
used for calculating our h-values
"""
second_deriv = n_deriv(f, 2)[-1]
fourth_deriv = n_deriv(f, 4)[-1]

second_deriv_max = find_max_error(second_deriv, [a, b], 0.01)
second_deriv_max = abs(second_deriv.subs(x, second_deriv_max))

fourth_deriv_max = find_max_error(fourth_deriv, [a, b], 0.01)
fourth_deriv_max = abs(fourth_deriv.subs(x, fourth_deriv_max))

"""
calculate h-values, interals, and points using the Composite error terms
"""
h_t = ((tolerance*12)/((b-a)*second_deriv_max))**(1/2)
h_s = ((tolerance*180)/((b-a)*fourth_deriv_max))**(1/4)

interval_t = math.ceil((b-a)/h_t)

#simpsons needs an even interval (resulting in odd # of points)
interval_s = math.ceil((b-a)/h_s)
if interval_s%2 != 0:
    interval_s += 1

points_t = interval_t+1
points_s = interval_s+1

#check to make sure the math we did above is actually correct and in tolerance
assert second_deriv_max*((b-a)/points_t)**2*((b-a)/12) <= tolerance
assert fourth_deriv_max*((b-a)/points_s)**4*((b-a)/180) <= tolerance


"""
setting up our method of calculating each integral
"""
points = [points_t, points_s]
simps = [False, True]
h_vals = [h_t, h_s]

#for loop that uses either the trapezoid method or simpsons method because I didn't want to copy/paste
for calc_type in range(len(points)):
    print("\n=-=-=-=-=-=\nComposite {0}:\n=-=-=-=-=-=".format("Trapezoidal" if calc_type == 0 else "Simpson's"))
    print("Required h = {0}".format(h_vals[calc_type]))
    print("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=")

    #sets up our n values
    n_vals = [11, 41, points[calc_type]]
    err_vals = [0 for _ in range(len(n_vals))]
```

```python
data = [[] for _ in range(len(n_vals))]

#iterates through n values and calculates integral with each n
for i in range(len(n_vals)):
    n = n_vals[i]

    #calculate our h-value for this specific n
    h = ((b-a)/(n-1))

    integral = numerical_integral(h, n, f, a, b, simps[calc_type])

    #calculates absolute and relative error
    abs_err = abs(real - integral)
    rel_err = abs(abs_err/real)*100

    data_row = [n, h, real, integral, abs_err, rel_err, abs_err <= tolerance]
    data[i] = data_row

#prints out table of information
print(tabulate(data, headers=["n",
                              "h",
                              "Real Value",
                              "Calculated Value",
                              "Absolute Error",
                              "Relative Error",
                              "Within Tolerance"]))

print("\n=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=\n")
```