ME 310
Project 4 11.01.2023

MIDDLE EAST TECHNICAL UNIVERSITY

DEPARTMENT OF MECHANICAL ENGINEERING

ME 310 NUMERICAL METHODS

FALL 2022

PROGRAMMING PROJECT 4

Murat Berk Buzluk 2377653

Gökberk Çiçek 2377810

Contents

ME 310

Project 4 11.01.2023

Abstract

ME310 lecture students create this report to develop proper programs to solve an integral problem with the trapezoidal and Gauss-quadrature rules. Python is used in programming because the broader application area of Python makes it preferable for students to learn. We have learned how to code and deal with different problems caused by other functions. All the mistakes that we have done while writing the code led us to a better understanding of the course objectives. F.py file contains equations, fp.py contains boundary conditions and e237765 was the main code.

1. INTRODUCTION

F.py file contains equations, fp.py contains boundary conditions and e237765 was the main code.
Some integration methods have been driven to solve the complicated function's integrals, which are not solved analytically. In this programming project, the Gauss quadrature and the Trapezoidal rules will be applied to solve the integral of any function $f(x)$, in the domain of $[a,b]$. This specified domain will be divided into n segments, $n = 1,2,3 \dots$. The number of segments is increased one by one in the Gauss quadrature method, while it is increased by a multiple of two in the Trapezoidal rule. The algorithm will increase the number of segments until the desired accuracy is obtained.

2. HAND CALCULATIONS

- Newton-Cotes are the most common numerical integration arrangement that replaces a complicated function with an approximating function:

$$I = \int_a^b f(x)dx \cong \int_a^b f_n(x)dx \qquad \textbf{\textit{(1)}}$$

Where $f_n(x)\, is\ a\ polynomial\ of\ the\ form$

$$f_n(x) = a_0 + a_1x + \cdots . a_{n-1}x^{n-1} + a_nx_n \qquad \textbf{\textit{(2)}}$$

n is the order of the polynomial.

- For instance, if $n = 1$, approximating function is a first-order polynomial or a straight-line function.

Trapezoidal rule:

- The trapezoidal rule is the first of the Newton-Cotes closed form (the beginning and the endpoints are known) integration formulas, and it is applied by approximating with the integral of first-order polynomials:

$$I = \int_a^b f(x)dx \cong \int_a^b f_1(x)dx \tag{3}$$

$$f_1(x) = f(a) + \frac{f(b) - f(a)}{b - a}(x - a) \tag{4}$$

$$I \cong \int_a^b [\, f(a) + \frac{f(b) - f(a)}{b - a}(x - a)]dx \tag{5}$$

$$I \cong (b - a)\frac{f(a) + f(b)}{2} \tag{6}$$

- True error of the trapezoidal rule is:

$$E_t = -\frac{1}{12}f''(\zeta)(b - a)^3 \tag{7}$$

Where ζ is located in between $a\ and\ b$ points.

- Trapezoidal rule is working with zero error for linear functions.

The Multiple-Application Trapezoidal Rule

- To improve the accuracy of the trapezoidal rule, the number of segments should be increased in the domain of $[a, b]$.
- Defining

$$h = \frac{b-a}{n} \tag{7}$$

and applying the trapezoidal rule for each segment yields:

$$I \cong h\frac{f(x_0)+f(x_1)}{2} + h\frac{f(x_1)+f(x_2)}{2} + h\frac{f(x_2)+f(x_3)}{2} \ldots . + h\frac{f(x_{n-1})+f(x_n)}{2} \tag{8}$$

$$I \cong \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right] \tag{9}$$

Or it may be expressed as

$$I \cong \frac{b-a}{2n}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right] \tag{10}$$

- True error can be obtained as

$$E_t = -\frac{(b-a)^3}{12n^3}\sum_{i=1}^{n} f''(\zeta_i) \tag{11}$$

- To simplify this expression, $f''(\zeta_i)$ may be defined as

$$f''(\zeta_i) \cong \bar{f}''(x) = \frac{\int_a^b f(x)dx}{b-a} \tag{12}$$

$$I = \int_a^b f(x)dx$$

Then,

$$E_a = -\frac{(b-a)^2}{12n^2} I \qquad \textit{(13)}$$

Gauss quadrature rule:

- The Gauss quadrature rule aims to determine the coefficients of an equation form

$$I \cong c_0 f(x_0) + c_1 f(x_1) \qquad \textit{(14)}$$

- Unlike the trapezoidal rule, $x_o \; and \; x_1$ points are unknown. Therefore, there are four unknowns in the two-point Gauss-Legendre formula
- To determine these unknowns, assuming fits the defined integral of a constant, a linear, a quadratic, and a cubic function.

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} 1dx = 2 \qquad \textit{(15)}$$

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} xdx = 0 \qquad \textit{(16)}$$

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} x^2dx = \frac{2}{3} \qquad \textit{(17)}$$

$$c_0 f(x_0) + c_1 f(x_1) = \int_{-1}^{1} x^3dx = 0 \qquad \textit{(18)}$$

- When equations given above are solved simultaneously,

$$c_0 = c_1 = 1 \tag{19}$$

$$x_0 = -\frac{1}{\sqrt{3}} \cong -0.5773 \tag{20}$$

$$x_1 = \frac{1}{\sqrt{3}} \cong 0.5773 \tag{21}$$

Which can be substituted into an equation and yields

$$I \cong f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) \tag{22}$$

- To solve the integral for the different endpoints instead of $[-1,1]$, defining a new variable x_d, yields

$$x = \frac{(b+a) + (b-a)x_d}{2} \tag{23}$$

And differentiated to obtain

$$dx = \frac{b-a}{2} dx_d \tag{24}$$

- When x and dx are substituted by x_d and dx_d, the integral can be solved in the domain $[a, b]$.

Higher-point Gauss quadrature

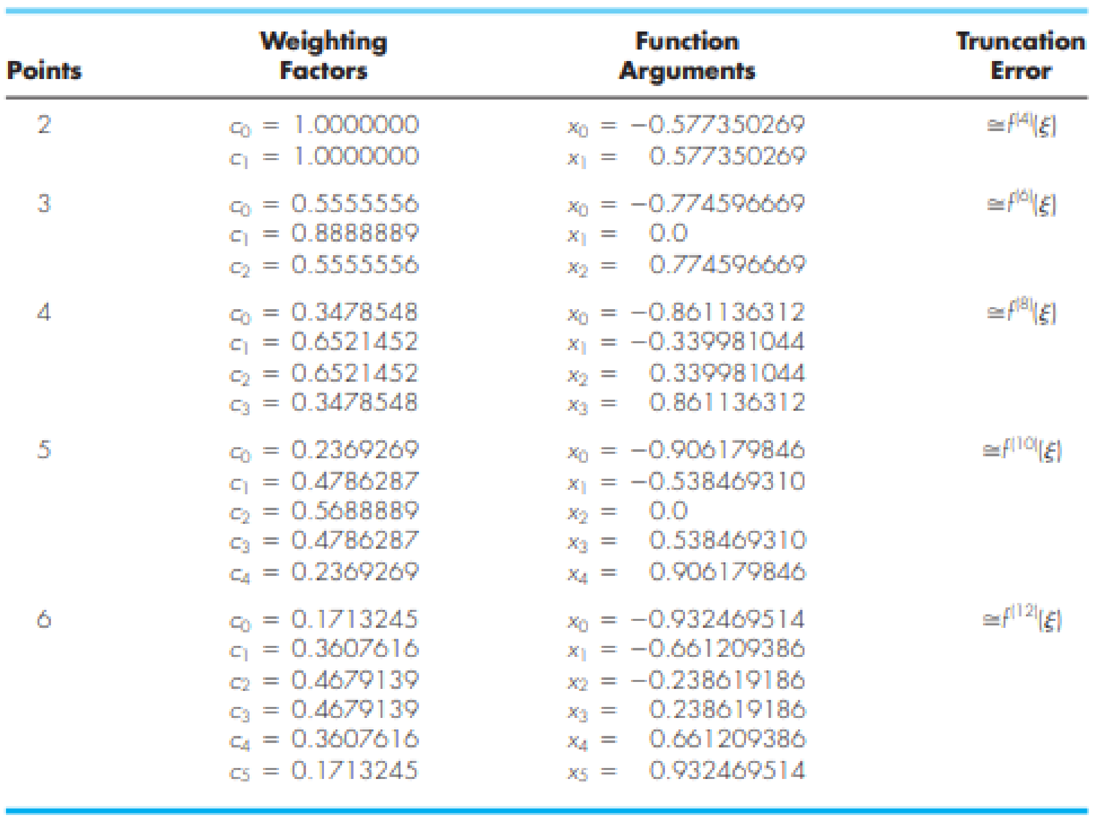
- Beyond two points, the integral form with higher points can be defined as

$$I \cong c_0 f(x_0) + c_1 f(x_1) + \cdots + c_{n-1} f(x_{n-1}) \tag{25}$$

  Where n: the number of points.
- Table 1, retrieved from Numerical Methods for Engineers by Chapra and Canale, provides weighting factors c and function arguments x in Gauss-Legendre formulas

- The error for the Gauss-Legendre formulas is specified generally by (Carnahan et al., 1969)

$$E_t = \frac{2^{n+3}[(n+1)!]^4}{(2n+3)[(2n+2)!]^3} f^{(2n+2)}(\zeta) \tag{26}$$



Points	Weighting Factors	Function Arguments	Truncation Error
2	$c_0 = 1.0000000$	$x_0 = -0.577350269$	$\cong f^{(4)}(\xi)$
	$c_1 = 1.0000000$	$x_1 = 0.577350269$	
3	$c_0 = 0.5555556$	$x_0 = -0.774596669$	$\cong f^{(6)}(\xi)$
	$c_1 = 0.8888889$	$x_1 = 0.0$	
	$c_2 = 0.5555556$	$x_2 = 0.774596669$	
4	$c_0 = 0.3478548$	$x_0 = -0.861136312$	$\cong f^{(8)}(\xi)$
	$c_1 = 0.6521452$	$x_1 = -0.339981044$	
	$c_2 = 0.6521452$	$x_2 = 0.339981044$	
	$c_3 = 0.3478548$	$x_3 = 0.861136312$	
5	$c_0 = 0.2369269$	$x_0 = -0.906179846$	$\cong f^{(10)}(\xi)$
	$c_1 = 0.4786287$	$x_1 = -0.538469310$	
	$c_2 = 0.5688889$	$x_2 = 0.0$	
	$c_3 = 0.4786287$	$x_3 = 0.538469310$	
	$c_4 = 0.2369269$	$x_4 = 0.906179846$	
6	$c_0 = 0.1713245$	$x_0 = -0.932469514$	$\cong f^{(12)}(\xi)$
	$c_1 = 0.3607616$	$x_1 = -0.661209386$	
	$c_2 = 0.4679139$	$x_2 = -0.238619186$	
	$c_3 = 0.4679139$	$x_3 = 0.238619186$	
	$c_4 = 0.3607616$	$x_4 = 0.661209386$	
	$c_5 = 0.1713245$	$x_5 = 0.932469514$	

Table 1: Weighting factors c and function arguments x used in Gauss-Legendre formulas

3. NUMERICAL RESULTS

- The operated function is $f(x) = e^x$

The trapezoidal rule results:

Segment number (n)	Integral value	Approximate error $(\%)$
1	8.332091516	0.69
2	7.860573807	0.16
4	7.740871532	0.04
8	7.71082937	0.01
16	7.7033115	0.25
32	7.701431573	0.063
64	7.700961563	0.016
128	7.700844059	0.0039
256	7.700814682	0.00098
512	7.700807338	0.00024
1024	7.700805502	$6.1\text{x}10^{-5}$
2048	7.700805043	$1.5\text{x}10^{-5}$
4096	7.700804929	$3.8\text{x}10^{-6}$

Table 2: The numerical results of the trapezoidal rule, including the segment number, integral results, and approximated error value

- The number of function evaluations is ***8180*** while using the trapezoidal rule.

The Gauss-quadrature rule results:

Segment number (n)	Integral value	Approximate error $(\%)$
1	7.700801197	-
2	7.700804831	$4.7x10^{-5}$
3	7.700804885	$7.0x10^{-7}$

Table 3: The numerical results of the Gauss-quadrature rule, including the segment number, integral results, and approximated error value

- The approximated error is calculated as

$$E_a = \frac{|I_{i+1} - I_i|}{I_i} 100\%$$

  Because the error calculation formula depends on the number of segments n, it causes high-number operations.

- Note that the result from the built-in function is 7.700804890362478 in the python program.

4. GRAPHIC OF NUMERICAL RESULTS

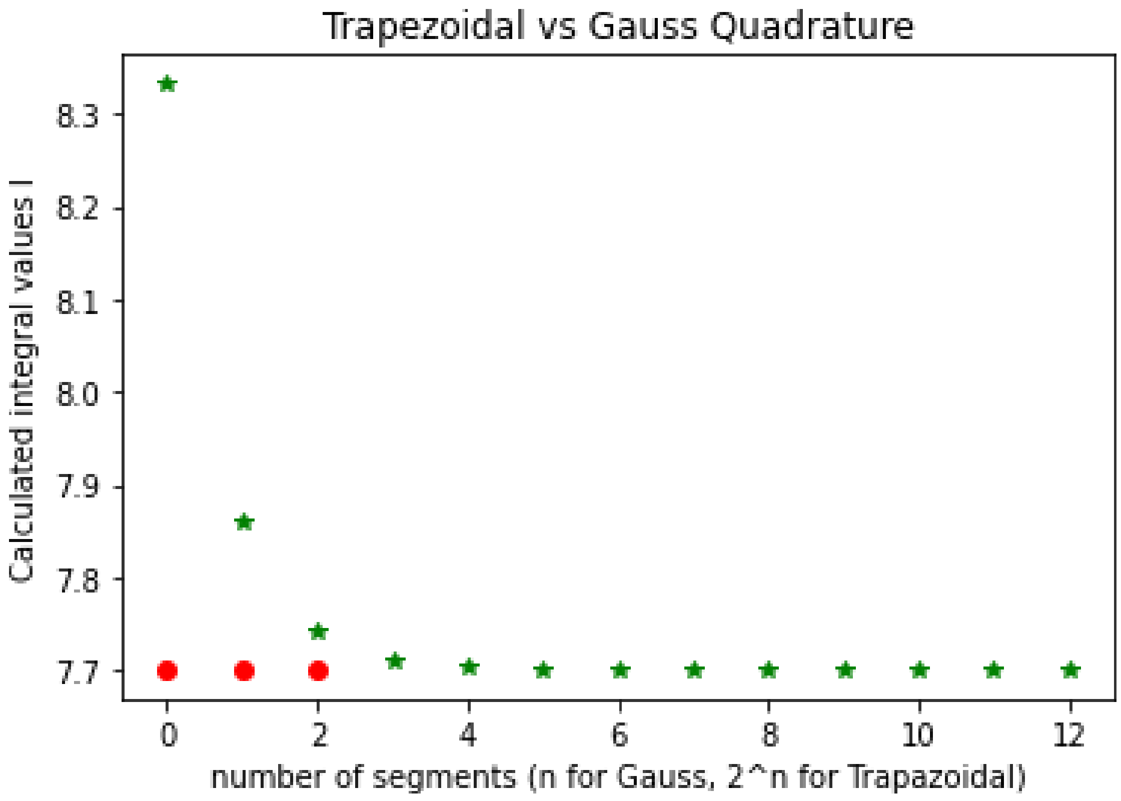


Figure 4: The distributed point graph of Integral results vs. number of segments

- 🔴 ***Gauss-quadrature results***
- 🟢 ***Trapezoidal method results***

Figure 4 is drawn according to table 2 and table 3, and it shows that the closes result to the actual result is obtained by the Gauss quadrature rule in $n = 2$ segments, while it is obtained by the trapezoidal rule in $n = 4096$ segments. (The figure can only show the first 12 segments.). The last result of the Gauss quadrature integration is I =7.700804885. The last result of the trapezoidal integration is I =7.700804929 which is lesser close to real result.

5. DISCUSSION AND CONCLUSION

In this programming project, the trapezoidal and the Gauss-quadrature rule are adapted into Python programming to solve any selected function $f(x)$. In this example, the function is selected as $f(x) = e^x$.

The code aims to find the Integral of function $f(x)$ by applying the two methods mentioned before with an increasing number of segments n. The built-in function obtains the actual result, which is $I = 7.700804890362478$. In the numerical results section, one may conclude that as the number of segments increases, the obtained result gets closer to the actual result for both methods. In other words, approximated error values become more comparable to zero. When these two methods are compared, the Gauss-quadrature method provides a faster convergence speed to the actual result and fewer iterations than the trapezoidal rule. Also, the final result of the Gauss quadrature rule is closer than the result of the trapezoidal rule, which means the approximated error value in the Gauss quadrature rule is less than the trapezoidal rules. Therefore, using the Gauss quadrature rule for evaluating the integral of the function $f(x) = e^x$ provides a less time-consuming and better result compared to the trapezoidal rule.

7. Appendices

Code of the Program

1. F

```
import math
def f(x):
    return math.exp(x)
```

2. Fb

```
a = 1.5
b = 2.5
E_s = 0.0000001
```

3. e2377653

```
import numpy as np
from f import f
from fp import a
from fp import b
from fp import E_s
import matplotlib.pyplot as plt
n = 0
E_a = 1
I_plot = []
f_a = f(a)
f_b = f(b)
count = 2 ## counting started from 2 because of f_a and f_b
print(' Trapezodial Rule:')
while E_a>E_s:
    Sum = 0
    I = 0
    if n == 0:
        n=1
    else:
        n = n*2
    h = (b-a)/n
    for i in range(1,n):
        Sum = Sum + f(a+h*i)
        count += 1
    I = (h/2)*(f_a+2*Sum+f_b)
    I_plot.append(I)
    E_a = abs(((h**2)/12)*I)
    E_a_p = E_a*100
    print('Segment n: %1.4g' %n,'\t\t Integral values: %1.10g'%I,
          '\t\tApproximate Error: %1.2g' %E_a_p)

print('Trapezodial Rule number of function evaluation: ', count)

n = 0
E_a = 1
I_plot2 = []
c1 = 5/9
c2 = 8/9
c3 = 5/9
x1 = -np.sqrt(3/5)
x2 = 0
x3 = np.sqrt(3/5)
print(' \n Gauss Quadrature Rule:')

while E_a>E_s:
    I_old = I
    I = 0
    n +=1
    for i in range(1,(n+1)):
        x_l = a+((b-a)/n)*(i-1)
        x_u = x_l +(b-a)/n
        I = I + (x_u-x_l)/2*(c1*f((x_u+x_l)/2+(x_u-x_l)/2*x1)
                         +c2*f((x_u+x_l)/2+(x_u-x_l)/2*x2)
                         +c3*f((x_u+x_l)/2+(x_u-x_l)/2*x3))
    I_plot2.append(I)
    if n == 1:
        E_a = 1
        print('Segment n: %1.4g' %n,'\t\t Integral values: %1.10g'%I,
              'Approximate Error: -' )
    else:
        E_a = (I-I_old)/I
        E_a_p = E_a*100
        print('Segment n: %1.4g' %n,'\t\t Integral values: %1.10g'%I,
              'Approximate Error: %1.2g' %E_a_p)

import scipy.integrate as spi
integrand = lambda x : np.exp(x)
```

```
a = 1.5
b = 2.5

result, none = spi.fixed_quad(integrand, a, b, n=5)
print('\n Result from built in function is: ', result)

plt.plot(I_plot, 'g*', I_plot2, 'ro')
plt.title('Trapezoidal vs Gauss Quadrature')
plt.xlabel('number of segments (n for Gauss, 2^n for Trapazoidal)')
plt.ylabel('Calculated integral values I')
plt.show()
```