Mathematical Physics III

Integration Module

Lab Report for Assignment _2

College Roll no.	:	2020PHY1097
University Roll no.	:	20068567031
Name	:	Kabir Sethi
Unique Paper Code	:	32221401
Paper Title	:	Mathematical Physics III Lab
Course and Semester	:	B.Sc.(Hons.) Physics, Sem IV
Due Date	:	20 February, 2022
Date of Submission	:	20 February, 2022
Lab Report File Name	:	2020PHY1097_A2a

Partners :

Name	:	Akarsh Shukla
Roll Number	:	2020PHY1216
Name	:	Brahmanand Mishra
Roll Number	:	2020PHY1184

Shri Guru Tegh Bahadur Khalsa College, University of Delhi
New Delhi-110007, India.

Contents

1 Introduction

let us consider that we have to evaluate the value of the integral:

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

over a finite interval [a, b]. The integrand f(x) is assumed to be real-values and smooth. The approximation of an integral by a numerical method is commonly referred to as quadrature.

2 Newton Cotes Quadrature Rule

In numerical analysis, the Newton–Cotes quadrature rules are a group of formulas for numerical integration based on evaluating the integrand at equally spaced points.
The Newton-Cotes formulas are an extremely useful and straightforward family of numerical integration techniques.
To integrate a function f(x) over some interval [a,b], divide it into n equal parts such that:
$f_n = f(x_n)$ and $h = \frac{b-a}{n}$

Then find polynomials which approximate the tabulated function, and integrate them to approximate the area under the curve. To find the fitting polynomials, use Lagrange interpolating polynomials the resulting formulas obtained are called Newton-Cotes formulas, or quadrature formulas.

There are two classes of Newton–Cotes quadrature: they are called **closed** when $x_0 = a$ and $x_n = b$, i.e., **they use the function values at the interval endpoints**, and **open** when $x_0 > a$ and $x_n < b$, i.e., **they do not use the function values at the endpoints.**
The Newton cotes Quadrature rule in general form is given by:

$$\int_a^b f(x)dx = nh\Big[y_0 + \frac{n}{2}\Delta y_0 + \frac{n(2n-3)}{12}\Delta^2 y_0 + \frac{n(n-2)^2}{24}\Delta^3 y_0 + ...\Big]$$

In short the difference is given in the table below:

Open	**Closed**
if the end points are not included in the integral	if the end points are included in the integral

3 Trapezoidal Rule

It is a a technique for approximating the definite integral in numerical analysis. The trapezoidal rule is an integration rule used to calculate the area under a curve by dividing the curve into small trapezoids. The summation of all the areas of the small trapezoids will give the area under the curve.
since , the $\int_a^b f(x)dx$ is approximated as:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i$$

where, w_i are called weights and x_i are called function argument or nodes.
By fixing the function argument x_i,
we can find the the unknown weights(w_i) such that the rule or formula is exact for a polynomial of degree n and we have the exact expression in form of :

$$\int_a^b f(x)dx = (b-a)[\frac{f(a)+f(b)}{2}]$$

3.1 Derivation of Trapezoidal Rule

The integral $\int_a^b f(x)dx =$ is approximated at:

$$\int_a^b f(x)dx \simeq w_0 f(x_0) + w_1 f(x_1)$$

Fixing or selecting the function argument as:

$$x_0 = a, x_1 = b$$

Now, the unknown weights w_0, w_1 are to be determined
Let, $f(x) = a_0 + a_1 x$ be a polynomial of degree 1 , or a straight line , the formula is exact for a polynomial of degree 1. Therefore, we have exact expression:

$$\int_a^b f(x)dx = w_0 f(a) + w_1 f(b) \tag{1}$$

But , from above

$$\int_a^b f(x)dx = \int_a^b (a_0 + a_1 x)dx$$

$$= a_0 x\Big|_a^b + a_1 \frac{x^2}{2}\Big|_a^b$$

$$\int_a^b f(x)dx = a_0(b-a) + a_1(\frac{b^2-a^2}{2}) \tag{2}$$

Now,

$$f(x) = a_0 + a_1 x$$

$$f(a) = a_0 + a_1 a$$

$$f(b) = a_0 + a_1 b$$

from equation 1:

$$\int_a^b f(x)dx = w_0(a_0 + a_1 a) + w_1(a_0 + a_1 b)$$

rearranging the terms :

$$\int_a^b f(x)dx = a_0(w_0 + w_1) + a_1(w_0 a + w_1 b) \tag{3}$$

comparing equation 2 and 3 :

$$w_0 = b - a, w_0 a + w_1 b = \frac{b^2 - a^2}{2}$$

solving above equations we get ,

$$w_0 = \tfrac{b-a}{2} \quad \text{and} \quad w_1 = \tfrac{b-a}{2}$$

putting the values of w_0 and w_1 in equation 1:

$$\int_a^b f(x)dx = \frac{b-a}{2} f(a) + \frac{b-a}{2} f(b)$$

$$\int_a^b f(x)dx = b - a[\frac{f(a) + f(b)}{2}]$$

$$\int_a^b f(x)dx = h\Big[\frac{f(a) + f(b)}{2}\Big]$$

where, h= b-a
The formula that we obtained above is called Trapezoidal Formula or rule.

3.2 composite trapezoidal rule

It is often the case that the function we are trying to integrate is not smooth over the interval. Typically, this means that either the function lacks derivatives at certain points. In these cases, Simpson's rule may give **very poor** results.
One common way of handling this problem is by breaking up the interval $[a, b]$ into small sub intervals. Integration rule is then applied to each sub interval, with the results being summed to produce an approximation for the integral over the entire interval.This process of getting more exact result is known as **Composite Rule**.
The composite trapezoidal rule is a method for approximating a definite integral by evaluating the integrand at n points. Let [a,b] be the interval of integration with a partition $a = x_0 < x_1 < x_2 < x_3 < x_4 < ..... < x_n = b$. Then the formal rule is given by

$$\int_a^b f(x)dx \approx \frac{1}{2} \sum_{j=1}^{n} (x_j - x_{j-1})[f x_{j-1} + f(x_j)]$$

The composite trapezoidal rule can also be applied to a partition which is uniformly spaced $x_j - x_{j-1} = h$ for all j $\in (1, ..., n)$. In this case, the formal rule is given by:

$$\int_a^b f(x)dx \approx \frac{h}{2}\Big[f(a) + 2\sum_{j=1}^{n-1}[f(a + jh) + f(b)\Big]$$

3.3 Error in Trapezoidal Method

The error in Trapezoidal Method can be obtained in following way. Let $y = f(x)$ be a continuous, well behaved and possess continuous derivative in $[x_0, x_n]$. Expanding y in a Taylor's series around $x = x_0$ we obtain:

$$\int_{x_0}^{x_1} y \ , dx = \int_{x_0}^{x_1} [y_0 + (x - x_0)y_0' + (x - x_0)^2 y_0'' + ...]dx$$

$$= hy_0 + \frac{h^2}{2}y_0' + \frac{h^3}{6}y_0'' + ... \tag{4}$$

Similarly,

$$\frac{h}{2}(y_0 + y_1) = [\frac{h}{2}(y_0 + y_0 + hy_0' + \frac{h}{2}y_0'' + \frac{h^3}{6}y_0''' + ...]$$
$$= hy_0 + \frac{h}{2}y_0' + \frac{h}{4}y_0'' + ... \tag{5}$$

from equation 1 and 2 , we obtained

$$\int_{x_0}^{x_1} y \ , dx - \frac{h}{2}(y_0 + y_1) = -\frac{1}{12}h^3 y_0'' + ... \tag{6}$$

Which is in the interval $[x_0, x_n]$. Proceeding in a similar manner we obtains the error in remaining sub intervals, viz $[x_1, x_2]$,$[x_2, x_3]$,.... and $[x_{n-1}, x_n]$. We thus have

$$E = -\frac{1}{12}h^3(y_0'' + y_1'' + y_2'' + ... + y_{n-1}'' \tag{7}$$

Where **E** is the *total error* . Assuming that y"($\bar{x}$) is the largest value of n quantities on the right hand side of *equation* 4 ,we obtained:

$$= -\frac{1}{12}h^3 n y''(\bar{x}) = -\frac{b-a}{12}h^2 y''(\bar{x})$$

4 Simpson $\frac{1}{3}$ Rule

We have several numerical methods to approximate an integral, such as Riemann's left sum, Riemann's right sum, midpoint rule, trapezoidal rule, Simpson's rule, etc. But among these, Simpson's rule gives the more accurate approximation of a definite integral. If we have f(x) = y, which is equally spaced between [a,b], the Simpson's rule formula is:

$$\int_a^b f(x) \approx \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + ... + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)$$

4.0.1 Derivation of Simpson Rule

The integral $\int_a^b f(x)dx =$ is approximated at:

$$\int_a^b f(x)dx \simeq w_0 f(x_0) + w_1 f(x_1) + w_2 f(x_2)$$

Fixing or selecting the function argument as:

$$x_0 = a, x_1 = \frac{a+b}{2}, x_2 = b$$

These are equispaced.
The unknown weights w_0, w_1 and w_2 are to be determined.
Let, $f(x) = a_0 + a_1 x + a_2 x^2$ be a polynomial of degree 2 as a parabola , the formula is exact for a polynomial of degree 2. Therefore, we have exact expression:

$$\int_a^b f(x)dx = w_0 f(a) + w_1 f(\frac{a+b}{2}) + w_2 f(b) \tag{8}$$

But , from above

$$\int_a^b f(x)dx = \int_a^b (a_0 + a_1x + a_2x^2)dx$$

$$= a_0x\Big|_a^b + a_1\frac{x^2}{2}\Big|_a^b + a_2\frac{x^3}{3}\Big|_a^b$$

$$\int_a^b f(x)dx = a_0(b-a) + a_1(\frac{b^2-a^2}{2}) + a_2(\frac{b^3-a^3}{3}) \tag{9}$$

Now,

$$f(x) = a_0 + a_1x + a_2x^2$$

$$f(a) = a_0 + a_1a + a_2a^2$$

$$f(b) = a_0 + a_1b + a_2b^2$$

$$f(\frac{a+b}{2}) = a_0 + a_1\frac{a+b}{2} + a_2\Big(\frac{a+b}{2}\Big)^2$$

from equation 8:

$$\int_a^b f(x)dx = w_0(a_0 + a_1a + a_2a^2) + w_1(a_0 + a_1\frac{a+b}{2} + a_2\Big(\frac{a+b}{2}\Big)^2) + w_2(a_0 + a_1b + a_2b^2)$$

rearranging the terms :

$$\int_a^b f(x)dx = a_0(w_0+w_1+w_2)+a_1(w_0a+w_1\frac{a+b}{2}+bw_2)+a_2(a_2^2w_0+\Big(\frac{a+b}{2}\Big)^2w_1+b^2w_2) \tag{10}$$

comparing equation 9 and 10 :

$$w_0 + w_1 + w_2 = b - a, w_0a + w_1\frac{a+b}{2} + w_2b^2 = \frac{b^2-a^2}{2}$$

,

$$a^2w_0 + \Big(\frac{a+b}{2}\Big)^2w_1 + b^2w_2 = \frac{b^3-a^3}{3}$$

solving above equations we get ,

$$w_0 = \tfrac{b-a}{6} \quad , \quad w_1 = 2\tfrac{b-a}{3} \quad \text{and} \quad w_2 = \tfrac{b-a}{6}$$

Putting back the values of w_0, w_1 and w_2 in equation :

Therefore, $\int_a^b f(x)dx = \frac{b-a}{6}f(a) + 2\Big(\frac{b-a}{3}\Big)f(\frac{a+b}{2}) + \frac{b-a}{6}f(b)$

$$\int_a^b f(x)dx = \frac{b-a}{6}\Big[f(a) + 4f(\frac{a+b}{2}) + f(b)\Big]$$

This formula derived above is the simpson‘s $\frac{1}{3}$ formula .

4.1 Composite Simpson Rule

As we know that the Simpson rule gives most accurate result when the function to be integrated is Smooth function.

Suppose that the interval [a,b] is split up into n sub-intervals, with n an even number. Then, the composite Simpson's rule is given by:

$$\int_a^b f(x)dx \approx \frac{h}{3}\sum_{j=1}^{1/2}\Big[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})\Big]$$

$$= \frac{h}{3}\Big[f(x_0) + 2\sum_{j=1}^{n/2-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\Big]$$

Where, $x_j = a + jh$ for $j = (0, 1, ..., n-1, n)$ and $h = \frac{(b-a)}{n}$

4.2 Simpson $\frac{3}{8}$ Rule

Since we know that the Newton cotes Formula in the general form is :

$$\int_a^b f(x)dx = nh\Big[y_0 + \frac{n}{2}\Delta y_0 + \frac{n(2n-3)}{12}\Delta^2 y_0 + \frac{n(n-2)^2}{24}\Delta^3 y_0 + ...\Big]$$

Putting $n = 3$ in above formula:

$$\int_a^b f(x)dx = \int_a^b f_3(x)dx$$

where, f_3 is a polynomial of degree 3.
Putting n and neglecting all the differences above the 3^{rd} order, from Newton's formula we have:

$$\int_{x_0}^{x_3} f(x)dx = 3h\Big[y_0 + \frac{3}{2}\Delta y_0 + \frac{9}{2}\Delta^2 y_0 + \frac{1}{8}\Delta^3 y_0\Big]$$

using, $\Delta y_0 = y_1 - y_2$, $\Delta^2 y_0 = y_2 - 2y_1 + y_0$ and $\Delta^3 y_0 = y_3 - 3y_2 + 3y_1 - y_0$

$$\int_{x_0}^{x_3} f(x)dx = 3h\Big[y_0 + \frac{3}{2}(y_1 - y_2) + \frac{9}{2}(y_2 - 2y_1 + y_0) + \frac{1}{8}(y_3 - 3y_2 + 3y_1 - y_0)\Big]$$

$$= \frac{3h}{8}\Big[y_0 + 3y_1 + 3y_2 + y_3\Big]$$

Similarly,

$$\int_{x_3}^{x_6} f(x)dx = \frac{3h}{8}\Big[y_3 + 3y_4 + 3y_5 + y_6\Big]$$

and summing on. summing up all these we get :

$$\int_{x_0}^{x_n} f(x)dx = \frac{3h}{8}\Big[(y_0+3y_1+3y_2+y_3)+(y_3+3y_4+3y_5+y_6)+.....+(y_{n-3}+3y_{n-2}+3y_{n-1}+y_6)\Big]$$

$$\boxed{\int_{x_0}^{x_n} f(x)dx = \frac{3h}{8}\Big(y_0 + 3y_1 + 3y_2 + 2y3 + 3y_4 + 3y_5 + 2y_6 + ... + 2y_{n-3} + 3y_{n-2} + 3y_{n-1} + y_n)\Big]}$$

This rule, is called **Simpson $\frac{3}{8}$ rule**

4.3 Composite Simpson's 3/8 rule

Dividing the interval [a,b] into n sub-intervals of length $h = \frac{(b-a)}{n}$ and introducing the nodes $x_i = a + ih$, we have

$$\int_a^b f(x)\,dx \approx \frac{3h}{8}\Big[f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) + 3f(x_4) + 3f(x_5) + 2f(x_6) + \cdots + 3f(x_{n-2}) + 3f(x_{n-1}) + f(x_n)\Big]$$

$$= \frac{3h}{8}\left[f(x_0) + 3\sum_{i\neq 3k}^{n-1} f(x_i) + 2\sum_{j=1}^{n/3-1} f(x_{3j}) + f(x_n)\right] \quad \text{for } k \in N_0.$$

4.4 Error in Simpson Rule

The error in Simpson rule is given by:

$$\int_a^b y dx = \frac{h}{3}[y_0 + 4(y_1 + y_3 + y_5 + ... + y_{n-1}) + 2(y_2 + y_4 + y_6 + ... + y_{n-2}) + y_n]$$

$$= -\frac{b-a}{180}h^4 y^{iv}(\bar{x})$$

where $y^{iv}(\bar{x})$ is the largest value of the fourth derivatives.

4.5 Geometrical Interpretation of Integration Method

Recall that the definite integral(It is the sum of the product of the lengths of intervals and the height of the function that is being integrated with that interval) of a sufficiently nice function f on the interval [a, b] is denoted by:

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

This above term have the geometric meaning of a signed area between the graph of f and x-axis. The word **signed** means that I take this area with the sign "plus" if f(x) ¿ 0 and with the sign "minus" if f(x) ¡ 0.

4.6 Condition for Number of Intervals

In **Trapezoidal rule** the given interval can be divided in any number of equal sub intervals say **n** number of intervals.
In **Simpson 1/3 rule** the given interval should be an **even** number .
In **Simpson 3/8 rule** the given interval should be a **multiple of 3** .

4.7 Relation between Number of Interval and Error

Since we know that the truncation error decreases as number of intervals(n) increase or h decreases.

But when be make very small than it the approximating error much smaller than the rounding off error which makes no sense. So decreasing h or increasing number of sub intervals will gives us the more accurate result up to the value at which the **Truncation** and **Rounding off** error are **approximately equal.**

5 Gauss Quadrature Method

An n-point Gaussian quadrature rule, is a quadrature rule constructed to yield an exact result for polynomials of degree $2n-1$ or less by a suitable choice of the nodes x_i and weights w_i for $i = 1, \cdots, n$. The rule is stated as:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

This exact rule is known as the Gauss-Legendre quadrature rule. The quadrature rule will only be an accurate approximation to the integral above if $f(x0)$ is well-approximated by a polynomial of degree $2n-1$ or less on $[-1, 1]$.

5.1 Relation Between Gauss Quadrature and orthogonal Polynomials:

For the simplest integration problem stated above, i.e., f(x) is well-approximated by polynomials on [-1, 1], the associated orthogonal polynomials are Legendre polynomials, denoted by $P_n(x)$. With the n^{th} polynomial normalized to give $P_n(1)$, the i^{th} Gauss node, xi, is the i^{th} root of P_n and the weights are given by the formula:

$$w_i = \frac{2}{(1-{x_i}^2)([P`_n(x_i)]^2)}$$

5.2 Difference between Newton cotes and Gauss quadrature

Newton Cotes forms pick equally spaced points in the interval of integration, Gaussian quadrature picks the best points. For this reason Gaussian quadrature is more accurate and uses less panels. This means less function evaluations and therefore less chance of round off error and better speed.

5.3 Two- Point Gauss Quadrature

The two-point Gauss quadrature rule is an extension of the trapezoidal rule approximation where the arguments of the function are not predetermined as a and b , but as unknowns x_1 and x_2 .

The general form of the Integration formula is

$$I = w_0 f(x_0) + w_1 f(x_1)$$

where , w_0 , $f(x_0)$,w_1 and $f(x_1)$ are all unknown
4 unknowns $\Rightarrow$ we can fit 3rd degree polynomial exactly

$$f(x) = Ax^3 + Bx^2 + cx + D$$

Figure 1: f(x)

substituting it in general formula, we get

$$\int_{-1}^{1} Ax^3 + Bx^2 + cx + D\,dx = w_0[Ax_0^3 + Bx_0^2 + Cx_0 + D] + w_1[Ax_1^3 + Bx_1^2 + Cx_1 + D]$$

$$A\frac{x^4}{4} + B\frac{x^3}{3} + C\frac{x^2}{2} + Dx\Big|_{-1}^{1} = w_0[Ax_0^3 + Bx_0^2 + Cx_0 + D] + w_1[Ax_1^3 + Bx_1^2 + Cx_1 + D]$$

combining the like terms of the right hand side:

$$\Rightarrow A[w_0x_0^3 + w_1x_1^3] + B[w_0x_0^2 + w_1x_1^2 - \frac{2}{3}] + c[w_0x_0 + w_1x_1] + D[w_0 + w_1 - 2]$$

In order for this to be true for any 3^{rd} degree polynomial (i.e all arbitrary coefficients A,B,C,D) must be zero

$$\begin{aligned} w_ox_o^3 + w_1x_1^3 &= 0 \\ w_0x_0^2 + w_1x_1^2 &= \frac{2}{3} \\ w_ox_o + w_1x_1 &= 0 \\ w_0 + w_1 &= 2 \end{aligned} \tag{11}$$

The solution of these equations gives

$$w_0 = w_1 = 1$$

and

$$x_0 = -\sqrt{\frac{1}{3}}, x_1 = +\sqrt{\frac{1}{3}}$$

All polynomials of degree 3 or less will be exactly integrated with a Gauss-Legendre 2 point formula.

5.4 How to solve$\int_{-1}^{1} f(x)dx$

As explained above in the Gauss Quadrature Method that the general form of the method is :

$$\int_{-1}^{1} f(x)dx = c_1f(x_1) + c_2f(x_2) + ... + c_nf(x_n)$$

for n=1,

$$\int_{-1}^{1} f(x)dx \approx c_1f(x_1)$$

is exact for polynomials of degree upto 2n-1=1 (for n=1).
that is x_0 and x_1.
$x_0 \rightarrow$

$$\int_{-1}^{1} 1dx = 2 = c_1$$
$$c_1 = 2$$

$x_1 \rightarrow$

$$\int_{-1}^{1} xdx = 0 = c_1 x_1$$
$$x_1 = 0$$

Thus, the 1-point quadrature formula becomes:

$$\boxed{\int_{-1}^{1} f(x)dx = 2f(0)}$$

For n=2 i had explained it in above section 5.3

5.5 How to solve $\int_a^b f(x)dx$

Any integral with limits [a,b] can be converted into an integral with limits [-1,1].
Let, $x = mt + c$
If x=a, then t=-1,
If x=b, then t=+1,
such that,

$$a = m(-1) + c$$
$$b = m(+1) + c$$

Solving these we get,

$$m = \frac{b-a}{2}$$
$$c = \frac{b+a}{2}$$

Therefore, we get:

$$x = \frac{b-a}{2}t + \frac{b+a}{2}$$
$$\rightarrow dx = \frac{b-a}{2}dt$$

Substituting the values of x and dx in the integral we get

$$\boxed{\int_a^b f(x)dx = \int_{-1}^{1} f(\frac{b-a}{2}x + \frac{b+a}{2})\frac{b-a}{2}dx}$$

5.6 n-point Gauss Quadrature Rule:

As explained above in the Gauss Quadrature Method (5.3) that the general form of the method is :

$$\int_{-1}^{1} f(x)dx = w_1 f(x_1) + w_2 f(x_2) + ... + w_n f(x_n)$$

From above it is clear that we have $2n$ numbers of unknowns and so as to find the values we take $(2n-1)^{th}$ degree polynomial. so, the above can be written in general form as:

$$\boxed{\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i)}$$

6 Discussion

- The trapezoidal gives accurate value for linear function proved by the verification part
- The Simpson gives correct value for every verification function
- The Gauss quadrature also gives correct value for each verification function
- The graph obtained for newton cotes shows that Simpson gives more correct value of π than Trapezoidal and as m values increase each method approaches the correct value of π

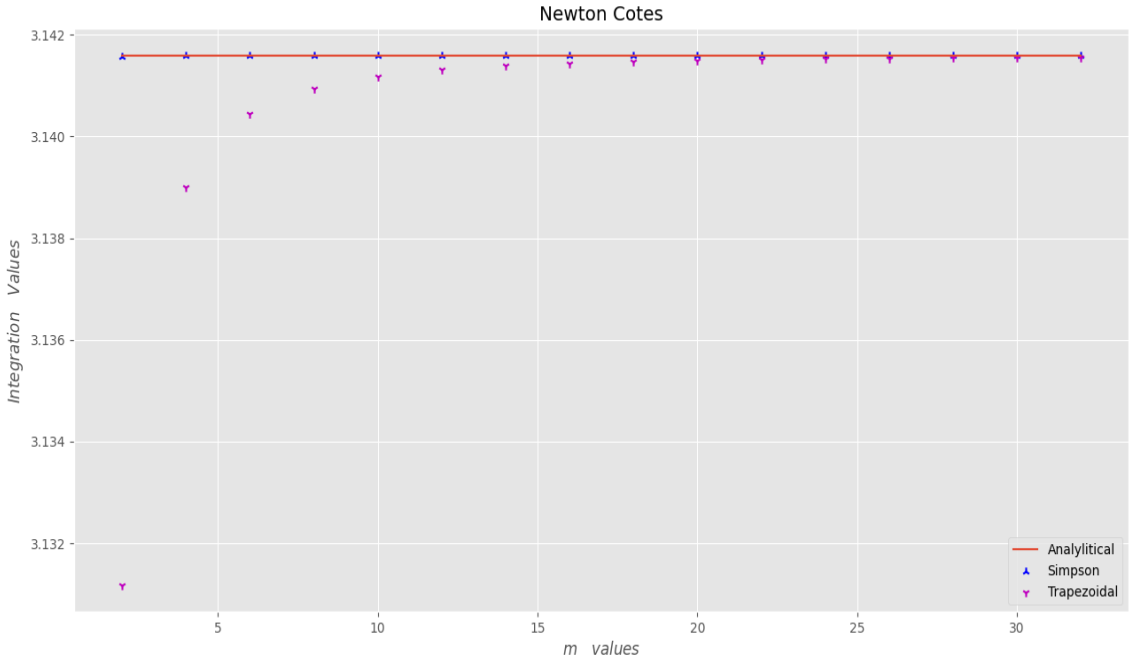

Figure 2: Sine Series

- The graphs above show that the value of error decreases as the value of m increases or value of step size decreases

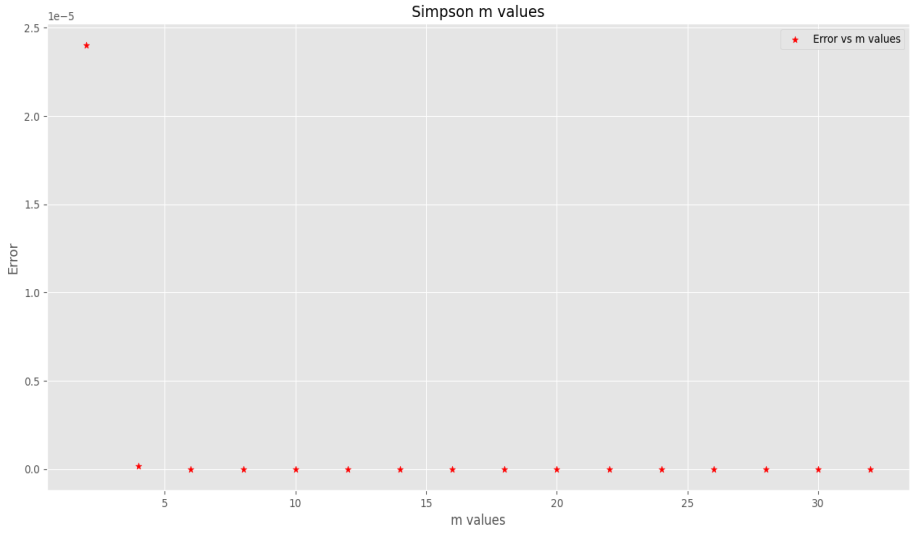

Figure 3: simpson˙m˙vs˙error˙values

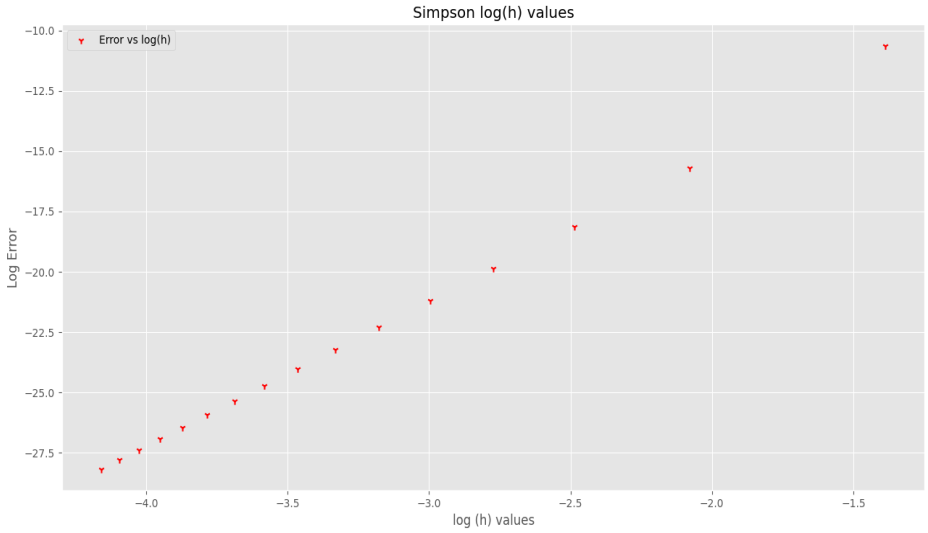

Figure 4: simpson˙log(h)˙vs˙error˙values

- The graphs show that the value of error decreases as the value of m increases or value of stepsize decreases

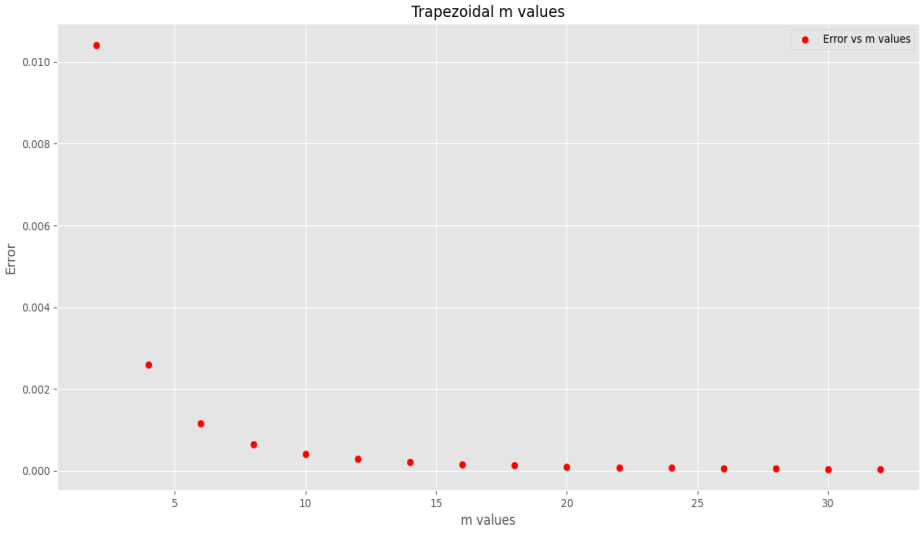

Figure 5: Trapezoidal˙m˙vs˙error˙values

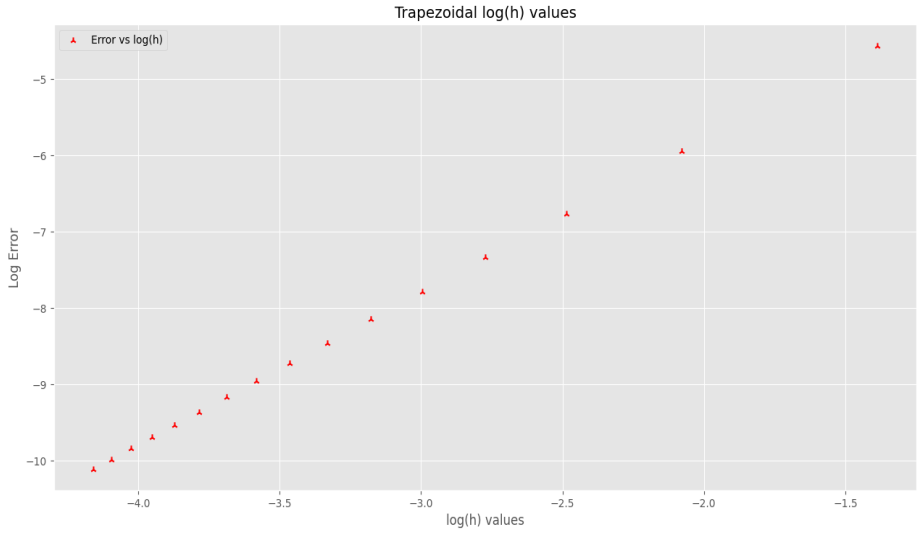

Figure 6: Trapezoidal˙log(h)˙vs˙error˙values

- The value of pi tends to actual value of π as the value of m and n increases. Although this method overestimates the value of π in contrast to newton cotes which underestimates the value of π

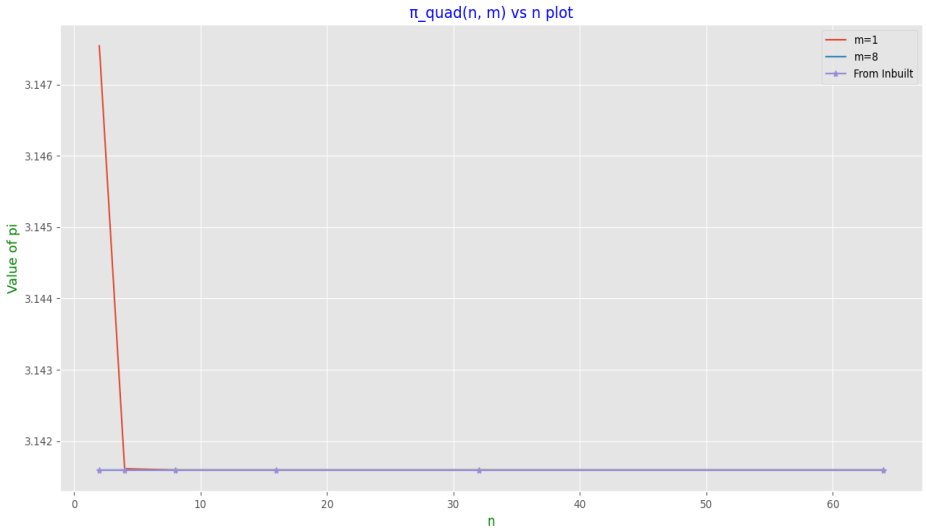

Figure 7: Gauss Quadrature Pi value for n

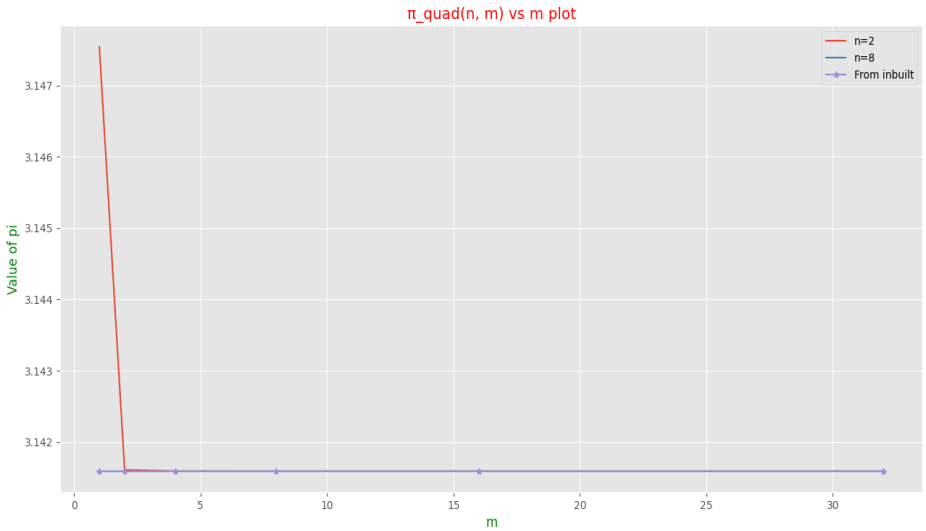

Figure 8: Gauss Quadrature Pi value for m

- Error for quadrature also decreases as the value of m or n increases.

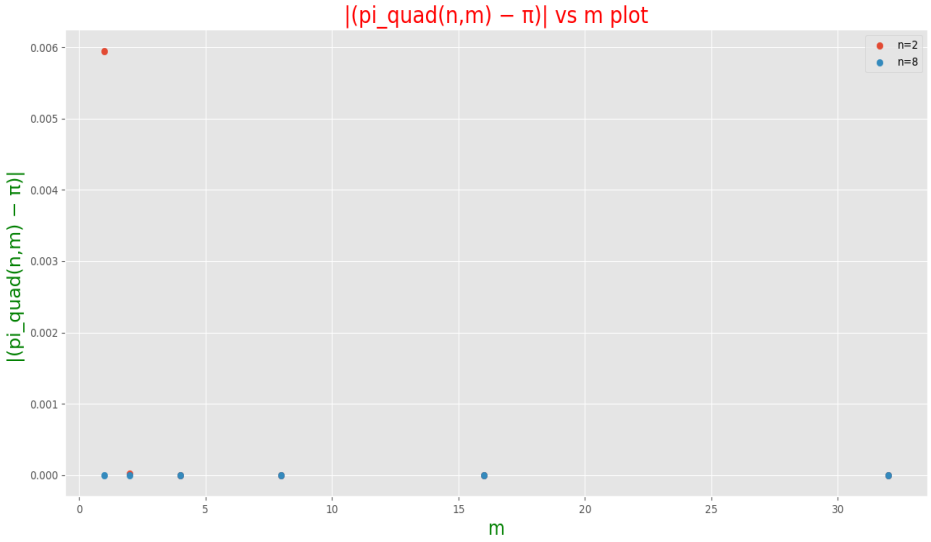

Figure 9: Gauss Quadrature Error for m

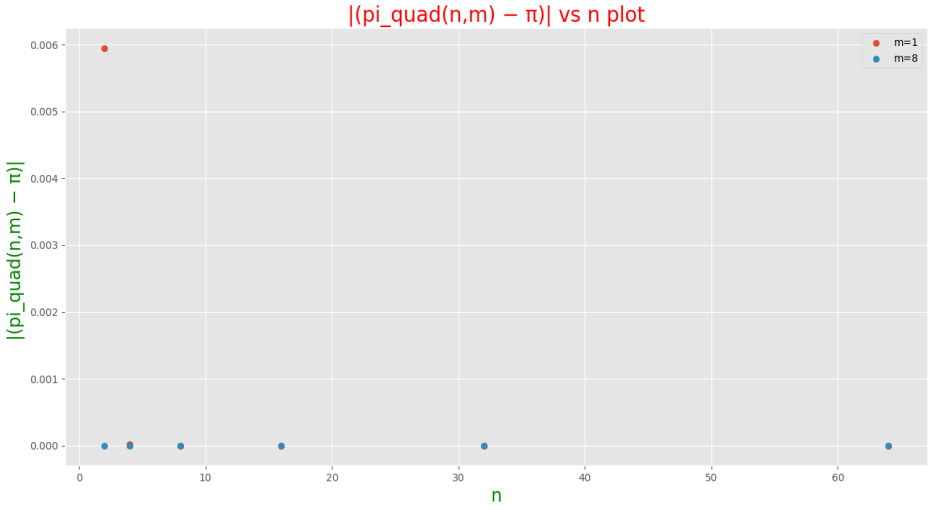

Figure 10: Gauss Quadrature Error for n

7 Algorithm

Algorithm 1 Tolerance

Data: INPUT -: limits of integration, number of sub intervals, function to be integrated, tolerance, maximum number of sub intervals that can be taken to achieve tolerance starting value of integration, a string to decide trap or simpson method, a weight function, and an optional i¨odd parameter for simpson integration

Result: OUTPUT -: returns the value of integration and relative error at the end of integration or an array of consisting of tolerance not reached and False bool value

for k = n, ..., Nmax **do**

$$stepsize = \frac{length \quad of \quad interval}{number \quad of \quad sub \quad interval}$$

  for i = 0, 1....., in x values **do**
    if $i < 2 * *k$ **then**
      $appendweight * y_{2*i+1}$
    end
  end
  if str = "simp" **then**

$$out = (sumofintegrationarray) * \frac{step \quad size}{3} + \frac{out1}{2} - \frac{i_{odd}}{6} * Stepsize$$

  end
  if str = 'trap' **then**

$$out = sum \quad of \quad integration \quad array * \frac{step \quad size}{2} + \frac{out1}{2}$$

  end

$$relative \quad error_i = \frac{out - out1}{out}$$

  if $rel < tolerance$ **then**
    return [out, error]
  end
  else
    out1 = out
    if str = simp **then**
      i¨odd = sum of integration array
    end
  end
end
return [tolerance not reached]

Algorithm 2 MyTrap

Data: INPUT -: limits of integration, number of sub intervals, function to be integrated, tolerance, maximum number of sub intervals that can be taken to achieve tolerance

Result: OUTPUT -: returns the value of integration

23

$$stepsize = \frac{length \quad of \quad interval}{number \quad of \quad sub \quad interval}$$

; `/* now form an array of y values using function from x values */`

24

$$y_i = f(x_i)$$

`/* now initialise the integration array with end points */`

25 **for** j = 1, ..., -1 in y values **do**

26 | append 2*j in integration array

27 **end**

28 `/* performing integration */`

29 `/* calculating the value of integration */`

30

$$out = sum \quad of \quad integration \quad array * \frac{step \quad size}{2}$$

`/* in below steps tolerance is checked with the help of tolerance function */`

31 **if** no of sub intervals = Nmax **then**

32 | return out1

33 **end**

34 check = tolerance function **if** second element of check is false **then**

35 | print tolerance can not be reached for given Nmax

36 **end**

37 **else**

38 | return check

39 **end**

Algorithm 3 MySimp

Data: INPUT -: limits of integration, number of sub intervals, function to be integrated, tolerance, maximum number of sub intervals that can be taken to achieve tolerance

Result: OUTPUT -: returns the value of integration

$$stepsize = \frac{length \quad of \quad interval}{number \quad of \quad sub \quad interval}$$

; /* now form an array of y values using function from x values */

$$y_i = f(x_i)$$

/* now initialise the integration array with end points */

$i_{odd} = 0$ /* performing integration */

for j = 0, 1....., -2 in y values **do**

  $i = i + 1$ **if** i is odd **then**

    $i_{odd} = i_{odd} + 4 * y_i$

  end

  if i is even **then**

    append 2*y˙i in integration array

  end

  append 2*j in integration array

end

/* calculating the value of integration */

$$out = sum \quad of \quad integration \quad array + i_{odd} * \frac{step \quad size}{3}$$

/* in below steps tolerance is checked with the help of tolerance function */

if no of sub intervals = Nmax **then**

  return out1

end

check = tolerance function **if** second element of check is false **then**

  print tolerance can not be reached for given Nmax

end

else

  return check

end

Algorithm 4 Gauss Legendre Quadrature

function MyLegGauss(f, a, b, n, m)

▷ f: the function of which the integral to be found
▷ a : lower limit of integral
▷ b: Upper limit of the Integral
▷ n: the degree of formula, ex: 2-point
▷ m: the number of sub intervals
▷ The Integral is $\int_a^b f(x)\,dx$

[x,w] = np.polynomial.legendre.leggauss(n) ▷ Taking the values of nodes and weights for a n-point formula

I=0 ▷ Initialize the I variable

h=(b-a)/(m) ▷ Width of each strip for given m subintervals

$h\_ar = np.arange(a, b + h, h)$ ▷ Creating a array which contain the m terms starting from a to b with formula $a_n = a_{n-1} + h$ **for** k in range(0,len(h_ar)-1): **do**

end

For loop to divide the integral into m subintervels

$half = float(h\_ar[k + 1] - h\_ar[k])/2.$ ▷ Changing the limits to -1 and 1

$mid = (h\_ar[k + 1] + h\_ar[k])/2.$ **for** c in range(n): **do**

end

To calculate the Integral by Gauss-Legendre n-point formula

$I+ = half * (w[c] * f(half * x[c] + mid))$

return I

end function

Algorithm 5 Gauss Legendre Quadrature with Tolerance

function MyLegGauss_tol($f, a, b, n, m, tol = None$)
▷ f: the function of which the integral to be found
▷ a : lower limit of integral
▷ b: Upper limit of the Integral
▷ n: the degree of formula, ex: 2-point
▷ m: the number of subintervals
▷ tol: The given tolerance
▷ The Integral is $\int_a^b f(x)\,dx$

$err = []$ **if** tol is None: **then**
end
If the tolerance is not given then it will give the result without the use of tolerance
return MyLegGauss(f,a,b,n,m) **else**
:
end
for d in range(2,m+1): **do**
end
Loop for comparing the d^{th}&$d-1^{th}$ values
value_1 = MyLegGauss(f,a,b,n,d-1)
value_2 = MyLegGauss(f,a,b,n,d)
$rel = abs((value_2 - value_1)/value_1)$ ▷ Calculating absolute error
$err.append(rel)$ **if** rel ¡= tol: **then**
end
If the error is less then or equal to given tolerance
print("The value of relative error is : ", rel, "and m is :", d)▷ it will break the loop
break ▷ and give us desired values

return [value_2, rel]

end function

8 Python Code

```
1
2 import numpy as np
3
4
5 def MySimp(a,b, N, f, tol, Nmax = None): # using previous values for
    simpson
6     err  = []
7     h1 = (b - a) / (2 * N)
8     x_values = np.linspace(a, b, 2 * N + 1)
9     y1 = f(x_values)
10
11     integrated_array1 = [y1[0], y1[-1]]
12     i_odd = 0
13     for i in range(len(x_values) - 2):
14         i = (i + 1)
15         #g = i % 2
16         if i %2 == 1: i_odd = i_odd + (4 * y1[i])
17         elif i%2 == 0: integrated_array1.append(2 * y1[i])
18     out1 = (sum(integrated_array1) + i_odd) * (h1 / 3)
19     if not Nmax:
20         return out1
21     check = tolerance(a, b, f, N, Nmax, out1, tol, "simp", 4, i_odd)
22     if check[1] == False:
23         return ["Tolerance not reached ", err]
24     else:
25         return check
26
27
28 def MyTrap(a, b, N, f, tol,Nmax = None):  # using previous values for
    tarapezoidal
29     err = []
30     h1 = (b - a) / (2 * N)
31     x_values = np.linspace(a, b, (2 * N) + 1)
32
33     y1 = f(x_values)
34
35     integration_array1 = [y1[0], y1[-1]]
36     temp = y1[1:-1]
37     for e in temp:
38         g = 2 * e
39         integration_array1.append(g)
40     out1 = sum(integration_array1) * (h1 / 2)
41     if not Nmax:
42         return out1
43     check = tolerance(a, b, f, N, Nmax, out1, tol, "trap", 2)
44     if check[1] == False:
45         return ["Tolerance not reached ", err]
46     else:
47         return check
48
49
50 def tolerance(a, b, f, N, Nmax, out1, tol, str, weight, i_odd=None):
51     err = []
52     for k in range(N, Nmax):
53         integration_array = []
54         h = (b - a) / (2 ** (k + 1))
```

```
        x_values = np.linspace(a, b, 2 ** (k + 1) + 1)
        for i in range(len(x_values)):
            if i < 2 ** k: integration_array.append(weight * f(x_values
    [(2 * i) + 1]))
        if str == "simp": out = sum(integration_array) * (h / 3) + (out1
    / 2) - (i_odd / 6) * h
        elif str == "trap": out = sum(integration_array) * (h / 2) + (
    out1 / 2)
        rel = abs((out - out1) / out)
        err.append(rel)
        if rel < tol:
            #print("The value of relative error is : ", rel, "and k is
    :", k, " and n is :", 2 ** k)
            return [out, err]
        else:
            out1 = out
            if str == "simp": i_odd = sum(integration_array)
    return ["tolerance can not be reached", False]


def MyLegGauss(f, a, b, n, m):
    [x, w] = np.polynomial.legendre.leggauss(n)
    I = 0
    h = (b - a) / (m)
    h_ar = np.arange(a, b + h, h)
    for k in range(0, len(h_ar) - 1):
        half = float(h_ar[k + 1] - h_ar[k]) / 2.
        mid = (h_ar[k + 1] + h_ar[k]) / 2.
        for c in range(n):
            I += half * (w[c] * f(half * x[c] + mid))
    return I


def MyLegGauss_tol(f, a, b, n, m, tol=None):
    err = []
    if tol is None:
        if n is list:
            d = []
            for v in n:
                u = MyLegGauss(f, a, b, v, m)
                d.append(u)
            return d
        elif m is list:
            c = []
            for w in m:
                g = MyLegGauss(f, a, b, n, w)
                c.append(g)
            return c
        elif n is not list and m is not list:
            return MyLegGauss(f, a, b, n, m)
    else:
        for d in range(2, m + 1):
            value_1 = MyLegGauss(f, a, b, n, d - 1)
            value_2 = MyLegGauss(f, a, b, n, d)
            rel = abs((value_2 - value_1) / value_1)
            err.append(rel)
            if rel <= tol:
                print("The value of relative error is : ", rel, "and m is
```

```
    :", d)
109                 break
110
111         return [value_2, rel]
```

```
1  from MyIntegration import *
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import pandas as pd
5  from matplotlib import style
6  from scipy import integrate
7  import tabulate
8  import math
9  style.use("ggplot")
10
11
12 def pifunc(x):
13     return 4/(1 + x**2)
14
15
16
17 ############################
18 # Questions
19 # 3(b) iii
20 @np.vectorize
21 def f1(x):
22     return 1
23
24
25 def f2(x):
26     return x
27
28 def f3(x):
29     return x ** 2
30
31 # verification for trapezoidal
32 print("verification for My Trap function for all three functions are :")
33 print("function: ", "Inbuilt Value")
34 print(MyTrap(1,2, 2, f1, 0.5 * 1e-3),integrate.quad(f1,1,2)[0])
35 print(MyTrap(1,2, 2, f2, 0.5 * 1e-3),integrate.quad(f2,1,2)[0])
36 print(MyTrap(1,2, 2, f3, 0.5 * 1e-3),integrate.quad(f3,1,2)[0])
37
38
39 # verification for trapezoidal
40 print("verification for My Simp function for all three functions are :")
41 print("function ", "Inbuilt Value")
42 print(MySimp(1,2, 2, f1, 0.5 * 1e-3),integrate.quad(f1,1,2)[0])
43 print(MySimp(1,2, 2, f2, 0.5 * 1e-3),integrate.quad(f2,1,2)[0])
44 print(MySimp(1,2, 2, f3, 0.5 * 1e-3),integrate.quad(f3,1,2)[0])
45
46
47
48 m_values = [2*i for i in range(1, 17)]
49 h_values = [1/(2*i) for i in m_values]
50 my_pi_Trap = np.array([MyTrap(0,1, i, pifunc, 0.5*1e-3 ) for i in
       m_values])
51 my_pi_simp = np.array([MySimp(0,1, i, pifunc, 0.5*1e-3 ) for i in
       m_values])
```

```
print("The Value of my_pi_Trap is :")
print(my_pi_Trap)
print("The Value of my_pi_simp is :")
print(my_pi_simp)
print("The value of inbuilt pi is :")
print(np.pi)
error_trap = [np.pi - i for i in my_pi_Trap]
error_simp = [np.pi - i for i in my_pi_simp]


pi_by_trap = MyTrap(0,1,1,pifunc, 0.5*1e-5, 16)
print(pi_by_trap[0])
pi_by_simp = MySimp(0,1,1,pifunc, 0.5*1e-5, 16)
print(pi_by_simp[0])


def scatter_plot(ax,a, b,title, markers, legend, xlabel , ylabel):
    for i in range(len(a)):
        ax.scatter(a[i], b[i], color="red", marker=markers[i], label=
    legend[i])
        ax.title(title[i])
        ax.xlabel(xlabel[i])
        ax.ylabel(ylabel[i])
        ax.legend()
        ax.grid('white')
        ax.show()




def plot(ax, title,xlabel ,ylabel, x, y, y1, y2, legend):
    ax.scatter(x,y1,  c = 'b', marker= '2', label = legend[0])
    ax.scatter(x, y2,c = 'm', marker = '1', label = legend[1])
    ax.plot(x,y, label = legend[2])
    ax.title(title)
    ax.xlabel(xlabel)
    ax.ylabel(ylabel)
    ax.grid('white')
    ax.legend()


plot(plt, 'Newton Cotes', '$ m \quad values$', '$ Integration \quad
    Values $', m_values,[np.pi for i in range(1, 17)],my_pi_simp
      , my_pi_Trap, ['Simpson', 'Trapezoidal', 'Analylitical'] )

plt.show()
markers = ['*', 'o', '1', '2']
a = [m_values, m_values, np.log(h_values), np.log(h_values)]
b = [error_simp, error_trap, np.log(error_simp), np.log(error_trap)]
leg = ['Error vs m values', 'Error vs m values', 'Error vs log(h)', '
    Error vs log(h)']
Title = ["Simpson m values","Trapezoidal m values", "Simpson log(h)
    values", 'Trapezoidal log(h) values']
xlabel = ['m values', 'm values', 'log (h) values', 'log(h) values']
ylabel = ['Error', 'Error', 'Log Error', 'Log Error']

scatter_plot(plt,a, b,Title,markers, leg, xlabel, ylabel)

```

```
# for value till 5 digit:
value, value_for_n = [], []
value.append(MyTrap(0,1, 1, pifunc, 0.5*1e-5, 16 )[0]), value_for_n.
    append(len(MyTrap(0,1,1,pifunc, 0.5*1e-5,16)[1]))
value.append(MySimp(0,1, 1, pifunc, 0.5*1e-5, 16)[0]), value_for_n.append
    (len(MySimp(0,1,1,pifunc, 0.5*1e-5,16)[1]))
er = [(np.pi - i)/np.pi for i in value]
dict = {"Methods ": [ "Trapezoidal","Simpson"],"my_pi(n)" : value, "n":
    value_for_n, "Error" : er  }
data = pd.DataFrame(dict)
print(data)


def f4(x):
    return (4 / (1 + x ** 2))


n_a = [2, 4, 8, 16, 32, 64]
m_a = [1, 2, 4, 8, 16, 32]
g = []
pi = np.array([math.pi] * len(n_a))

for m in m_a:
    s = []
    for n in n_a:
        s.append(MyLegGauss(f4, 0, 1, n, m))
    g.append(s)
d = np.array(g).reshape(6, 6)

np.savetxt('pi quad-1097a.dat', d, delimiter=',')
data = {"n": n_a, "m=1": g[0], "m=2": g[1], "m=4": g[2], "m=8": g[3], "m
    =16": g[4], "m=32": g[5]}
print('Question e (i)')
print(pd.DataFrame(data))

# Question e (ii)
plt.plot(n_a, g[0], label="m=1")
plt.plot(n_a, g[3], label="m=8")
plt.plot(n_a, pi, label="From Inbuilt", marker='*')
plt.xlabel("n", c="green")
plt.ylabel("Value of pi", c="green")
plt.title("\u03C0_quad(n, m) vs n plot", c="blue", )
plt.legend()
plt.grid('white')
plt.show()

y1 = []
y2 = []
for i in range(len(n_a)):
    y1.append(d[i][0])
    y2.append(d[i][2])

# Question e (iii)
plt.plot(m_a, y1, label="n=2", )
plt.plot(m_a, y2, label="n=8", )
plt.plot(m_a, pi, label="From inbuilt", marker="*")
plt.xlabel("m", c="green", )
plt.ylabel("Value of pi", c="green", )
```

```
plt.title("\u03C0_quad(n, m) vs m plot", c="red", )
plt.legend()
plt.grid('white')
plt.show()

# (iii)
e1 = []
e2 = []
e3 = []
e4 = []

e_t = []
e_s = []
for q, w, t, u in zip(g[0], g[3], y1, y2):
    e1.append(abs(q - math.pi))
    e2.append(abs(w - math.pi))
    e3.append(abs(t - math.pi))
    e4.append(abs(u - math.pi))

plt.scatter(n_a, e1, label="m=1", marker="o")
plt.scatter(n_a, e2, label="m=8", marker="o")
plt.xlabel("n", c="green", fontsize=17)
plt.ylabel("|(pi_quad(n,m)     \u03C0)| ", c="green", fontsize=17)
plt.title("|(pi_quad(n,m)     \u03C0)| vs n plot", c="red", fontsize=20)
plt.legend()
plt.grid('white')
plt.show()

plt.scatter(m_a, e3, label="n=2", marker="o")
plt.scatter(m_a, e4, label="n=8", marker="o")
plt.xlabel("m", c="green", fontsize=17)
plt.ylabel("|(pi_quad(n,m)     \u03C0)| ", c="green", fontsize=17)
plt.title("|(pi_quad(n,m)     \u03C0)| vs m plot", c="red", fontsize=20)
plt.legend()
plt.grid('white')
plt.show()


print('Question 3 (b) iii')
print('Verification of The function MyLegQuad')
t1=[ ['For F(x)= 1',MyLegGauss(f1,1,2,2,1)],
    ['For F(x)= 1 by Inbuilt',integrate.quad(f1,1,2)[0]],
    ['For F(x)= x',MyLegGauss(f2,1,2,2,1)],
    ['For F(x)= x by Inbuilt',integrate.quad(f2,1,2)[0]],
    ['For F(x)= x^2',MyLegGauss(f3,1,2,2,1)],
    ['For F(x)= x^2 by Inbuilt',integrate.quad(f3,1,2)[0]]]


print(tabulate.tabulate(t1))
```

References

[1] Introductory Methods of Numerical Analysis by S.S. Sastry

[2] Mathematical Methods for Physics and Engineering by Riley , Hobson and Bence

[3] Wolfram Mathworld
https://mathworld.wolfram.com

[4] Wikipedia
https://en.wikipedia.org/wiki

[5] Youtube
https://youtu.be/6x28YeHIkc4
https://www.youtube.com/watch?v=BXSEW2B_1UU
https://www.youtube.com/watch?v=qAVk2P5-Nvc

[6] Stackexchange
https://math.stackexchange.com/questions/870374/
comparison-of-newton-cotes-quadrature-and-gaussian-quadrature-formulas