Integration
Lab Report for Assignment No. 2(a)

SHASHVAT JAIN
(2020PHY1114)

ANKUR KUMAR
(2020PHY1113)

HARSH SAXENA
(2020PHY1162)

S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

February 20, 2022

Submitted to Dr. Mamta
"32221401 - MATHEMATICAL PHYSICS III"

Contents

1 Theory

1.1 Newton Cotes Quadrature

Q1.Explain the Newton Cotes Quadrature rules. What is the difference between open and closed Newton Cotes? Use method of undetermined coefficients to derive the $Trapezoidal$, $Simpson_{1/3}$ and $Simpson_{3/8}$ rules for integration.

- In numerical integration, the Newton-Cotes formulae, also called Newton-Cotes quadrature rules or simply Newton-Cotes rules, are a group of formulae for numerical integration(also called quadrature) based on evaluating the integrand at equally spaced points.
- They are named after Isaac Newton and Roger Cotes.
- 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 $h = (b - a)/n$. 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 are called Newton-Cotes formulas, or quadrature formulas.
- Newton-Cotes formulas may be "closed" if the interval $[x_1, x_n]$ is included in the fit, "open" if the points $[x_2, x_{(n-1)}]$ are used, or a variation of these two. If the formula uses n points (closed or open), the coefficients of terms sum to n-1.

Trapezoidal Rule

Using method of undetermined coefficients,

$$\int_a^b f(x)dx \approx c_1 f(a) + c_2 f(b) \tag{1}$$

We use a polynomial of first order for approximation. A basis for polynomials of degree 1 is [1,x].

$$\int_a^b 1dx = b - a = c_1 + c_2 \tag{2}$$

$$\int_a^b xdx = \frac{b^2 - a^2}{2} = c_1 a + c_2 b \tag{3}$$

Solving equation 2,3, for c_1, c_2 we get,

$$c_1 = c_2 = \frac{b-a}{2}$$

Putting these values in equation 1 we get,

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

which is the required result.

Simpson$_{1/3}$ Rule

Using method of undetermined coefficients,

$$\int_a^b f(x)dx \approx c_1 f(a) + c_2 f(\frac{a+b}{2}) + c_3 f(b) \tag{4}$$

We use a polynomial of second order for approximation. A basis for polynomials of degree 2 is $[1, x, x^2]$.

$$\int_a^b 1dx = b - a = c_1 + c_2 + c_3 \tag{5}$$

$$\int_a^b xdx = \frac{b^2 - a^2}{2} = c_1 a + c_2(\frac{a+b}{2}) + c_3 b \tag{6}$$

$$\int_a^b x^2 dx = \frac{b^3 - a^3}{3} = c_1 a^2 + c_2(\frac{a+b}{2})^2 + c_3 b^2 \tag{7}$$

Solving equation 5,6,7 for c_1, c_2, c_3 we get,

$$c_1 = c_3 = \frac{b-a}{6}, c_2 = \frac{4(b-a)}{6}$$

Putting these values in equation 4 we get,

$$\int_a^b f(x)dx \approx (\frac{b-a}{6})f(a) + (\frac{4(b-a)}{6})f(\frac{a+b}{2}) + (\frac{b-a}{6})f(b)$$

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

which is the required result.

Simpson$_{3/8}$ Rule

Using method of undetermined coefficients,

$$\int_a^b f(x)dx \approx c_1 f(a) + c_2 f(\frac{2a+b}{3}) + c_3 f(\frac{a+2b}{3}) + c_4 f(b) \tag{8}$$

We use a polynomial of third order for approximation. A basis for polynomials of degree 3 is $[1, x, x^2, x^3]$.

$$\int_a^b 1dx = b - a = c_1 + c_2 + c_3 + c_4 \tag{9}$$

$$\int_a^b xdx = \frac{b^2 - a^2}{2} = c_1 a + c_2(\frac{2a+b}{3}) + c_3(\frac{a+2b}{3}) + c_4 b \tag{10}$$

$$\int_a^b x^2 dx = \frac{b^3 - a^3}{3} = c_1 a^2 + c_2(\frac{2a+b}{3})^2 + c_3(\frac{a+2b}{3})^2 + c_4 b^2 \tag{11}$$

$$\int_a^b x^3 dx = \frac{b^4 - a^4}{4} = c_1 a^3 + c_2(\frac{2a+b}{3})^3 + c_3(\frac{a+2b}{3})^3 + c_4 b^3 \tag{12}$$

Solving equation 9 10,11,12 for c_1, c_2, c_3, c_4 we get,

$$c_1 = c_4 = \frac{b-a}{8}, c_2 = c_4 = \frac{3(b-a)}{8}$$

Putting these values in equation 8 we get,

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

which is the required result.

Q2.Explain how these can be made more accurate by increasing number of intervals and write the composite rules for each of them.

For the Newton–Cotes rules to be accurate, the step size h needs to be small, which means that the interval of integration $[a, b]$ must be small itself, which is not true most of the time. For this reason, one usually performs numerical integration by splitting $[a, b]$ into smaller subintervals, applying a Newton–Cotes rule on each subinterval, and adding up the results. This is called a composite rule.
It can be seen that truncation error decreases as number of intervals (or panels) increases or h decreases.

Trapezoidal Composite Rule

$$\int_a^b f(x)dx = \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right)$$

where $h = \dfrac{b-a}{n}$ and $x_i = a + ih$

Simpson$_{1/3}$ Composite Rule

$$\int_a^b f(x)dx = \frac{h}{3}\left(f(a) + 2\sum_{i=1}^{n/2-1} f(x_{2i}) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + f(b)\right)$$

where $h = \dfrac{b-a}{n}$ and $x_i = a + ih$

Simpson$_{3/8}$ Composite Rule

$$\int_a^b f(x)dx = \frac{3h}{8}\left(f(a) + 3\sum_{i\neq 3k}^{n-1} f(x_i) + 2\sum_{i=1}^{n/3-1} f(x_{3i}) + f(b)\right)$$

where $h = \dfrac{b-a}{n}$, $x_i = a + ih$ and $k \in \mathbb{N}_0$

Q3.Discuss the geometrical interpretation of these. What are the conditions on number of intervals for each of them?

Trapezoidal Rule

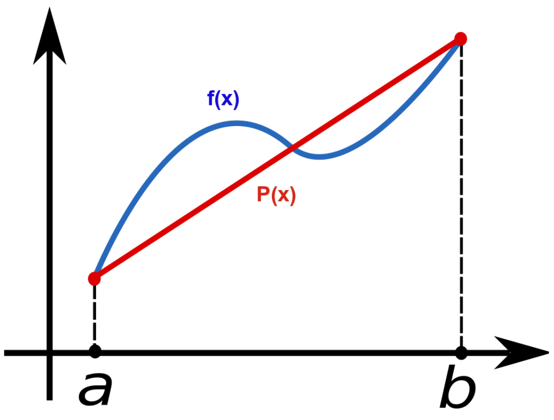


Figure 1: Trapezoidal Rule

In this rule the curve $f(x)$ is approximated by a straight line between two points. The integration is equal to the area under this line(Area of Trapezium).
For composite form the curve can be divided into n intervals where $n \in \mathbb{N}$.

Simpson$_{1/3}$ Rule

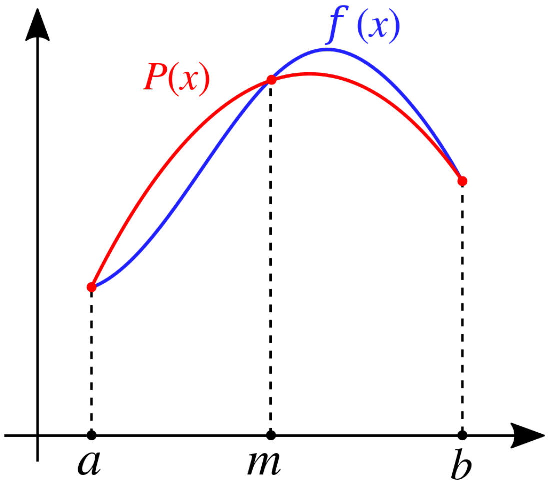


Figure 2: Simpson$_{1/3}$ Rule

In this rule the curve $f(x)$ is approximated by a parabola(2nd order polynomial) using 3 points. The integration is equal to the area under this curve(Area under parabola).
For composite form the curve can be divided into n even number of intervals.

Simpson$_{3/8}$ Rule

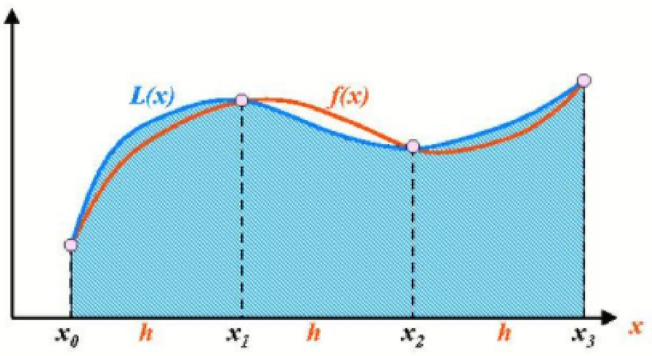


Figure 3: Simpson$_{3/8}$ Rule

In this rule the curve $f(x)$ is approximated by a polynomial of order 3 using 4 points. The integration is equal to the area under this curve(Area under polynomial).
For composite form the curve can be divided into n number of intervals where n is a multiple of 3.

Q4.Also discuss the error term in each of these. Can you keep on increasing the number of intervals to reduce the error?

Trapezoidal Rule

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

Simpson$_{1/3}$ Rule

$$\epsilon = (b-a)\frac{h^4}{180}f^{(4)}(\xi)$$

Simpson$_{3/8}$ Rule

$$\epsilon = (b-a)\frac{h^4}{80}f^{(4)}(\xi)$$

Where $a < \xi < b$

As we increase the number of intervals the error indeed goes on decreasing but only up to a certain point. There is no point in making h so small that the approximation error becomes much smaller than the rounding error. Decrease in h will only be beneficial up to the point at which the truncation and rounding errors are roughly equal.
Round-off error = machine epsilon $\times$ the value of integral.

1.2 Legendre Gauss Quadrature

Q1.What are Gauss quadrature methods for evaluating integrals? How are Gauss Quadrature methods different from the Newton Cotes Methods?

The term *Quadrature* refers to methods in which the points where the function is evaluated are chosen, and the weights calculated so that the formula is exact for polynomials of as high a degree as possible. The most basic of these methods is *Gaussian Quadrature.*

Gauss quadrature deals with integration over a symmetrical range of x from -1 to $+1$. The important property of Gauss quadrature is that it yields exact values of integrals for polynomials of degree up to $2n-1$. Gauss quadrature uses the function values evaluated at a number of interior points (hence it is an open quadrature rule) and corresponding weights to approximate the integral by a weighted sum.

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

Some differences from Newton Cotes are that in Gaussian Quadrature the points at which the function is calculated are not evenly spaced but instead chosen to give maximum accuracy. Secondly, the accuracy of n-point Gaussian Quadrature is $2n-1$ whereas a Newton-Cotes rule on n nodes is exact for polynomials of degree at most $n-1$. 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.

Q2.Explain how the Gauss quadrature method is closely linked with a set of orthogonal polynomials.

It turns out that the zeros of these orthogonal set of polynomials when chosen as the abscissa gives the highest accuracy. Here is why the zeros of orthogonal polynomials (in particular, Legendre polynomials) make a good choice of sample points. Given a general polynomial p of degree $2n-1$, we can use long division to obtain $p = qL_n + r$ where L_n is the Legendre polynomial of degree n, and both q and r have degree $< n$. Then,

$$\int_{-1}^{1} p(x) = \int_{-1}^{1} q(x)L_n(x)dx + \int_{-1}^{1} r(x)dx = \int_{-1}^{1} r(x)dx$$

because L_n is orthogonal to any polynomial of degree less than n. Since the term qL_n contributes zero to the integral, it should also contribute zero to our quadrature formula. Choosing the sample points to be the zeros of L_n achieves this goal.
Thus, one only needs to find the weights that give the exact value of $\int_{-1}^{1} r(x)dx$(where deg r $\leq n-1$) to obtain a quadrature formula that is exact for all degrees up to $2n-1$.

Q3.Explain Legendre Gauss Quadrature methods for evaluation of integral $\displaystyle\int_{-1}^{1} f(x)dx$. How will the formula change for the integration $\displaystyle\int_{a}^{b} f(x)dx$?

Gauss–Legendre quadrature is a form of Gaussian quadrature for approximating the definite integral of a function. For integrating over the interval $[-1,1]$, the rule takes the form:

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

Where

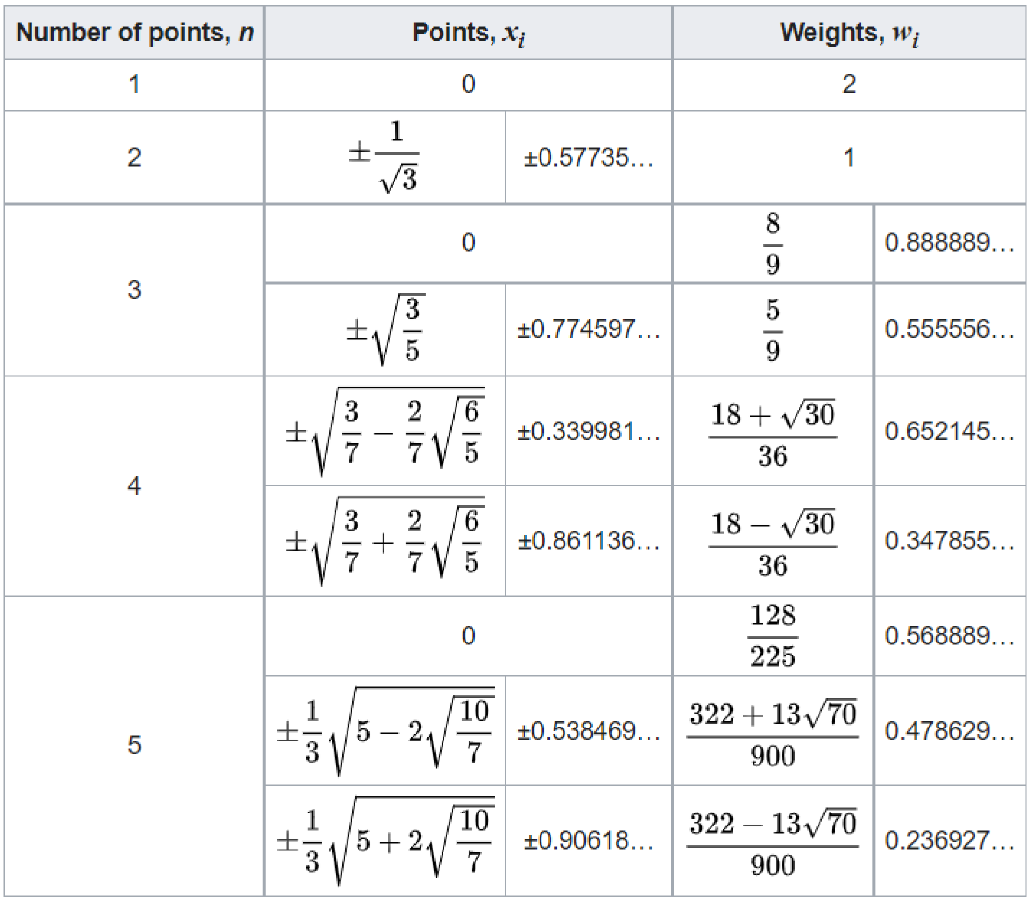
- n is the number of sample points used
- w_i are quadrature weights
- x_i are the roots of the nth Legendre polynomial.

This choice of quadrature weights w_i and quadrature nodes x_i is the unique choice that allows the quadrature rule to integrate degree $2n-1$ polynomials exactly.

For integrating f over $[-1,1]$ with Gauss–Legendre quadrature, the associated orthogonal polynomials are Legendre polynomials, denoted by $P_n(x)$. With the n-th polynomial normalized so that $P_n(1) = 1$, the i-th Gauss node, x_i, is the i-th root of P_n and the weights are given by the formula,

$$w_i = \frac{2}{\left(1-x_i^2\right)\left[P_n'(x_i)\right]^2}$$

Some low-order quadrature rules are tabulated below for integrating over $[-1,1]$



Number of points, n	Points, x_i		Weights, w_i	
1	0		2	
2	$\pm\frac{1}{\sqrt{3}}$	±0.57735…	1	
3	0		$\frac{8}{9}$	0.888889…
	$\pm\sqrt{\frac{3}{5}}$	±0.774597…	$\frac{5}{9}$	0.555556…
4	$\pm\sqrt{\frac{3}{7}-\frac{2}{7}\sqrt{\frac{6}{5}}}$	±0.339981…	$\frac{18+\sqrt{30}}{36}$	0.652145…
	$\pm\sqrt{\frac{3}{7}+\frac{2}{7}\sqrt{\frac{6}{5}}}$	±0.861136…	$\frac{18-\sqrt{30}}{36}$	0.347855…
5	0		$\frac{128}{225}$	0.568889…
	$\pm\frac{1}{3}\sqrt{5-2\sqrt{\frac{10}{7}}}$	±0.538469…	$\frac{322+13\sqrt{70}}{900}$	0.478629…
	$\pm\frac{1}{3}\sqrt{5+2\sqrt{\frac{10}{7}}}$	±0.90618…	$\frac{322-13\sqrt{70}}{900}$	0.236927…

Figure 4

An integral over $[a, b]$ must be changed into an integral over $[-1, 1]$ before applying the Gaussian quadrature rule. This change of interval can be done in the following way:

$$\int_a^b f(x)\,dx = \int_{-1}^{1} f\left(\frac{b-a}{2}\xi + \frac{a+b}{2}\right)\frac{dx}{d\xi}d\xi$$

with $\dfrac{dx}{d\xi} = \dfrac{b-a}{2}$

Applying n point Gaussian quadrature (ξ, w) rule then results in the following approximation:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}\xi_i + \frac{a+b}{2}\right)$$

Q4.Explicitly derive the 2-point quadrature formula for this method.

$$\int_{-1}^{1} f(x)\,dx \approx c_1 f(x_1) + c_2 f(x_2)$$

We know this is exact for polynomials of degree $2n - 1 = 2(2) - 1 = 3$. A basis for polynomials of degree 3 is $[1, x, x^2, x^3]$.

$$\int_{-1}^{1} 1\,dx = 2 = c_1 + c_2 \tag{13}$$

$$\int_{-1}^{1} x\,dx = 0 = c_1 x_1 + c_2 x_2 \tag{14}$$

$$\int_{-1}^{1} x^2\,dx = \frac{2}{3} = c_1{x_1}^2 + c_2{x_2}^2 \tag{15}$$

$$\int_{-1}^{1} x^3\,dx = 0 = c_1{x_1}^3 + c_2{x_2}^3 \tag{16}$$

Solving above 4 equations for c_1, c_2, x_1, x_2, we get

$c_1 = c_2 = 1$, $x_1 = \dfrac{-1}{\sqrt{3}}$, $x_2 = \dfrac{1}{\sqrt{3}}$

Therefore the two point quadrature formula is given by,

$$\int_{-1}^{1} f(x)\,dx = f\left(\frac{-1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right)$$

Q5.Explain n-point composite quadrature formula.

From the code shown below it can seen that for composite quadrature formula we divide the interval into equal smaller intervals and apply gauss quadrature on each interval. The number of points can be changed according to the desired accuracy or order of polynomial that is to be integrated and this is denoted by n.

2 Pseudo-code

2.1 Pseudo-code for Fixed-tol Trapezoidal method

Algorithm 1 Use Composite Trapezoidal rule to find fixed-tolerance numerical approximation for the given definite integral.

```
procedure MyTrap(f,a,b,max_subs,d)
    Input: f is the integrand, a is the lower limit and b is the upper-limit of integration, max_subs is the number of sub-intervals that the algorithm cannot exceed and d is the number of significant digits required in the numerical approximation of the integral.
    Output: Returns $I_{num}(f)$, the numerical approximation of $I$(f)
                                        ▷ We calculate the fixed-tolerance approximation integral by continually calculating the integral for double the number of subintervals than done in the prior iteration in order to avoid calculation of already calculated values of f(x).

    $m \leftarrow$ A vector of number of subintervals to calculate the integrals for, A geometric progression with commmon ratio 2.
    $I \leftarrow$ A vector that stores the value of the integral obtained after each iteration.
    $X^0 \leftarrow$ A vector of equally spaced nodes in closed interval $[a, b]$ obtained for $m_0$
    $l \leftarrow length(m_0)$
    $n \leftarrow length(X)$
    $I_0 \leftarrow \frac{b-a}{3m_0}\left[f(X_0) + 2\sum_{j=1}^{n-1} f(X_j) + f(X_n)\right]$
    for $k = 1, 2, 3 \ldots l$ do
        $X^k \leftarrow$ A vector of equally spaced nodes in closed interval $[a, b]$ obtained for $m_k$
        $n \leftarrow length(X^k)$
        $I_k = \frac{I_{k-1}}{2} + \frac{b-a}{m_k}\left[\sum_{j=1}^{n-1} f(X_j)\right]$
        if $|I_k - I_{k-1}| \leq 0.5 \times 10^{-d} \times |I_k|$ then
            Return $I_k, m_k$
            EXIT
    Could not reach tolerance.
    Return $I_l, m_l$
    EXIT
```

2.2 Pseudo-code for Fixed-tol Simpson's method

Algorithm 2 Use Composite Simpson rule to find fixed-tolerance numerical approximation for the given definite integral.

```
procedure MySimp(f,a,b,max_subs,d)
    Input: f is the integrand, a is the lower limit and b is the upper-limit of integration, max_subs is the number of sub-intervals that the algorithm cannot exceed and d is the number of significant digits required in the numerical approximation of the integral.
    Output: Returns $I_{num}(f)$, the numerical approximation of $I$(f)

    ▷ We calculate the fixed-tolerance approximation integral by continually calculating the integral for double the number of subintervals than done in the prior iteration in order to avoid calculation of already calculated values of f(x).

    $m \leftarrow$ A vector of number of subintervals to calculate the integrals for, A geometric progression with commmon ratio 2.
    $I \leftarrow$ A vector that stores the value of the integral obtained after each iteration.
    $X \leftarrow$ A vector of equally spaced nodes in closed interval $[a,b]$ obtained for $m_0$
    $l \leftarrow length(m_0)$
    $n \leftarrow length(X)$
    $I_0 \leftarrow \frac{b-a}{3m_0}\left[f(X_0) + 2\sum_{j=1}^{n/2-1} f(X_{2j}) + 4\sum_{j=1}^{n/2} f(X_{2j-1}) + f(X_n)\right]$
    for $k = 1, 2, 3 \ldots l$ do
        $X^k \leftarrow$ A vector of equally spaced nodes in closed interval $[a,b]$ obtained for $m_k$
        $n \leftarrow length(X^k)$
        $I_k = \frac{I_{k-1}}{2} + \frac{b-a}{3m_k}\left[4\sum_{j=1}^{n/2} f(X_{2j-1}) - 2\sum_{j=1}^{n/2-1} f(X_{2j})\right]$
        if $|I_k - I_{k-1}| \leq 0.5 \times 10^{-d} \times |I_k|$ then
            Return $I_k, m_k$
            EXIT
    Could not reach tolerance.
    Return $I_l, m_l$
    EXIT
```

3 Programming

Integration module Myintegration.py containing all methods,

```python
import numpy as np
from scipy.special import roots_legendre

def MyTrap(func,a,b,m=int(1e3),d=None):
    """
    Integrate ‘func‘ from ‘a‘ to ‘b‘ using composite trapezoidal rule. If ‘d‘ is not passed as argument during call then ‘m‘ is the number of uniformly spaced subintervals in the interval ‘[a,b]‘.
    When ‘d‘ is passed during call, the returned integral is accurate to atleast ‘d‘ significant digits, using a fixed relative-tolerance of ‘‘0.5x10**-d‘‘; Also ‘m‘ is now considered to be the upper-limit on the number of subintervals during the calculation of fixed-tolerance integral.

    Parameters
    ----------
    func : function
        A Python function or method to integrate.
    a : float
        Lower limit of integration.
    b : float
        Upper limit of integration.
    m : int, optional
        Number of subintervals for integration and if ‘d‘ is given this gives the maximum number of subintervals to calculate for before aborting.
    d : Integer, optional
        Number of significant digits required in the returned integral.
        Iteration stops when relative error between last two iterates is less than or equal to
        ‘0.5*10**(-d)‘.

    Returns
    -------
    if ‘d‘ is not passed as argument.
    val : float
        Trapezoidal rule's approximation to the integral.

    if ‘d‘ is passed as argument.
    val : float
        Trapezoidal rule's approximation(Fixed-tolerance) to integral to d significant digits.
    m : float
        Number of subintervals used for calculating the integral in the last iteration.
    """
    if d is not None and m!=1:
        max_n = np.floor(np.log2(m))
        m_array = np.logspace(0,max_n,base=2,num = int(max_n+1))
        I = np.zeros(m_array.shape)
        h = (b-a)/m_array
        x = np.linspace(a,b,int(m_array[0]+1))
        y = func(x)
        I[0] = (h[0]/2)*np.sum(y[:-1] + y[1:])
        for i in np.arange(0,m_array.shape[0]):
            x = np.linspace(a,b,int(m_array[i]+1))
            midx  = x[1::2]
            I[i] = (1/2)*I[i-1] + h[i]*(np.sum(func(midx)))
            if np.abs(I[i]-I[i-1]) <= 0.5/10**d*np.abs(I[i]):
                val,last_m =I[i],m_array[i]
                return val,last_m
        print("Could not reach desired accuracy with the given upperlimit on the number of intervals.(m) ")
        val,last_m = I[-1],m_array[-1]
        return val,last_m
    x = np.linspace(a,b,m+1)
    y = func(x)
    val = (b-a)/(2*m)*np.sum(y[:-1]+y[1:])
    return val

def MySimp(func,a,b,m=int(1e3),d=None):
    """
    Integrate ‘func‘ from ‘a‘ to ‘b‘ using composite simpson1/3 rule. If ‘d‘ is not passed as argument during call then ‘m‘ is the number of uniformly spaced subintervals in the interval ‘[a,b]‘.
    When ‘d‘ is passed during call, the returned integral is accurate to atleast ‘d‘ significant digits, using a fixed relative-tolerance of ‘‘0.5x10**-d‘‘; Also ‘m‘ is now considered to be the upper-limit on the number of subintervals during the calculation of fixed-tolerance integral.

    Parameters
    ----------
    func : function
        A Python function or method to integrate.
    a : float
        Lower limit of integration.
    b : float
        Upper limit of integration.
    m : int, optional
        Number of subintervals for integration and if ‘d‘ is given this gives the maximum number of subintervals to calculate for before aborting.
    d : Integer, optional
        Number of significant digits required in the returned integral.
        Iteration stops when relative error between last two iterates is less than or equal to
        ‘0.5*10**(-d)‘.

    Returns
    -------
    if ‘d‘ is not passed as argument.
    val : float
        Simpson1/3 rule’s approximation to the integral.

    if ‘d‘ is passed as argument.
    val : float
        Simpson1/3 rule’s approximation to integral to d significant digits.
    m : float
        Number of subintervals used for calculating the integral in the last iteration.
    """
    if d is not None and m!=1:
        max_n = np.floor(np.log2(m))
        m_array = np.logspace(1,max_n,base=2,num = int(max_n))
        I = np.zeros(m_array.shape)
        h = (b-a)/m_array
        x = np.linspace(a,b,int(m_array[0]+1))
        y = func(x)
        I[0] = (h[0]/3)*np.sum(y[:-1:2] +4*y[1:-1:2] +y[2::2])
        omidy = y[1::2]
        for i in np.arange(1,m_array.shape[0]):
            x = np.linspace(a,b,int(m_array[i]+1))
            midx = x[1::2]
            midy = func(midx)
            I[i] = I[i-1]/2 + h[i]*(4*np.sum(midy) - 2*np.sum(omidy))/3
            if np.abs(I[i]-I[i-1]) <= 0.5/10**d*np.abs(I[i]):
                val,last_m =I[i],m_array[i]
                return val,last_m
            omidy = midy.copy()
        print("Could not reach desired accuracy with the given upperlimit on the number of intervals.(m) ")
        val,last_m = I[-1],m_array[-1]
        return val,last_m
    x = np.linspace(a,b,m+1)
    y = func(x)
    val = (b-a)/(3*m)*np.sum(y[:-1:2]+4*y[1::2]+y[2::2])
    return val

def MyLegQuadrature(func,a,b,n=5,m=100,d=None):
    """
    Integrate ‘func‘ from ‘a‘ to ‘b‘ using composite Gaussian Quadrature Method. If ‘d‘ is not passed as argument during call then ‘m‘ is the number of uniformly spaced subintervals in the interval ‘[a,b]‘.

    When ‘d‘ is passed during call, the returned integral is accurate to atleast ‘d‘ significant digits, using a fixed relative-tolerance of ‘‘0.5x10**-d‘‘; Also ‘m‘ is now considered to be the upper-limit on the number of subintervals during the calculation of fixed-tolerance integral.

    Parameters
    ----------
    func : function
        A Python function or method to integrate.
    a : float
        Lower limit of integration.
    b : float
        Upper limit of integration.
    n : Integer
        No. of points to integrate at.
    m : int, optional
        Number of subintervals for integration and if ‘d‘ is given this gives the maximum number of subintervals to calculate for before aborting.
    d : Integer, optional
        Number of significant digits required in the returned integral.
        Iteration stops when relative error between last two iterates is less than or equal to
        ‘0.5*10**(-d)‘.

    Returns
    -------
    if ‘d‘ is not passed as argument.
    val : float
        Gaussian quadrature approximation to the integral.

    if ‘d‘ is passed as argument.
    val : float
        Gaussian quadrature approximation to integral to d significant digits.
    m : float
        Number of subintervals used for calculating the integral in the last iteration.

    """
    x,w = roots_legendre(n)
    if np.isinf(a) or np.isinf(b):
        raise ValueError("Gaussian quadrature is only available for finite limits.")
    if d is not None and m!=1:
```

```python
        max_n = np.floor(np.log2(m))
        m_array = np.logspace(0,max_n,base=2,num = int(max_n+1))
        I = np.zeros(m_array.shape)
        for i in np.arange(0,m_array.shape[0]):
            I[i] = MyLegQuadrature(func,a,b,n,m_array[i])
            if i == 0 :
                continue
            if np.abs(I[i]-I[i-1]) < 0.5/10**d*np.abs(I[i]):
                val,last_m = I[i],m_array[i]
                return val,last_m
        print("Could not reach desired accuracy with the given upperlimit on the number of intervals.(m) ")
        val,last_m = I[-1],m_array[-1]
        return val,last_m
    I,val = 0,0
    subs = np.linspace(a,b,int(m+1))
    l = len(subs)
    for a_i,b_i in zip(subs[:l-1],subs[1:l]):
        shifted_x = (b_i-a_i)*(x+1)/2 + a_i
        I+= (b_i-a_i)/2 * np.sum(w*func(shifted_x))
    val = I
    return val

MyTrap = np.vectorize(MyTrap)
MySimp = np.vectorize(MySimp)
MyLegQuadrature = np.vectorize(MyLegQuadrature)

if __name__=="__main__":
    #Validation tests integral-assignment_programming(a)
    #For MyTrap
    print(MyTrap(lambda x: x,0,6),6**2/2)
    print(MyTrap(lambda x: x**2,0,6),6**3/3)

    #For MySimp
    print(MySimp(lambda x: x**3,0,6,2),6**4/4)
    print(MySimp(lambda x: x**4,0,6.2),6**5/5)

    #For MyLegQuadrature
    print(MyLegQuadrature(lambda x: x**3,0,6,2,1),6**4/4)
    print(MyLegQuadrature(lambda x: x**4,0,6,2,1),6**5/5)

    pass
```

For the analysis of methods and generation of graphs,

```python
from Myintegration import *
import matplotlib.pyplot as plt
import pandas as pd
from scipy.integrate import quadrature
quadrature = np.vectorize(quadrature)

def main():
    #
    # Part(b)
    #
    n = 2*np.arange(1,17)
    h = 1/n
    #f_str = input("Enter the function : ")
    #f = lambda x :eval(f_str)
    f = lambda x : 1/(1+x**2)
    my_pi_simp = 4*MySimp(f,0,1,n)
    my_pi_trap = 4*MyTrap(f,0,1,n)

    err_simp= np.abs(my_pi_simp-np.pi)
    err_trap= np.abs(my_pi_trap-np.pi)

    #fig,(ax1,ax2) = plt.subplots(1,2,1)
    '''
    plt.plot(n,np.arccos(-1)*np.ones(n.shape),label = "")
    plt.plot(n,my_pi_simp,label= "my_pi_simp(n)")
    plt.plot(n,my_pi_trap,label= "my_pi_trap(n)")
    plt.plot()
    plt.show()
    plt.plot(n,err_simp,label= "e_simp(n)")
    plt.plot(n,err_trap,label= "e_trap(n)")
    plt.plot()
    plt.show()
    plt.plot(np.log(h),np.log(err_trap),label= "e_trap(n)")
    plt.plot(np.log(h),np.log(err_simp),label= "e_simp(n)")
    plt.plot()
    plt.legend()
    plt.show()
    '''
    #
    # Part(d)
    #

    '''
    signi_digits = 5
    tab_dat = np.array([MyTrap(f,0,1,int(1e6),signi_digits),MySimp(f,0,1,int(1e6),
   signi_digits)])
    pi_arr =np.pi*np.ones((2,))
    df = pd.DataFrame({"Method":["Trapezoidal","Simpson1/3"],"Pi_calc":4*tab_dat
   [:,0], "n":tab_dat[:,1],"E" : np.abs(4*tab_dat[:,0] - pi_arr)/pi_arr})
    print(df)
    '''
    #
    # Part (e)
    #
    n_points,m_arr = 2**np.arange(1,7),2**np.arange(0,6)
    '''
    nm_mat = np.ones((len(n_points),len(m_arr)))
    for i,n_i in enumerate(n_points):
        nm_mat[i,:] = 4*MyLegQuadrature(f,0,1,n_i,m_arr)
    np.savetxt("pi_quad-1114.dat",nm_mat,delimiter=",",fmt="%.16f")
    '''
    '''
    nm_mat = np.loadtxt("pi_quad-1114.dat",delimiter= ",",dtype=float)
    err_nm_mat = nm_mat - np.arccos(-1)
    print(nm_mat)
    fig,(axs1,axs2) = plt.subplots(1,2)
    axs1.plot(n_points,nm_mat[:,np.where(m_arr==1)[0][0]],label="m=1",marker=".")
    axs1.plot(n_points,nm_mat[:,np.where(m_arr==8)[0][0]],label="m=8",marker=".")
    axs1.plot(n_points,np.arccos(-1)*np.ones(n_points.shape),label = "$\pi = \cos
   ^{-1}(-1)}$")
    axs2.plot(n_points,err_nm_mat[:,np.where(m_arr==1)[0][0]],label="m=1",marker=".")
    axs2.plot(n_points,err_nm_mat[:,np.where(m_arr==8)[0][0]],label="m=8",marker=".")
    axs1.legend();axs2.legend()
    fig1,(ax1,ax2) = plt.subplots(1,2)
    ax1.plot(m_arr,nm_mat[np.where(n_points==2)[0][0]],label="n=2",marker="1")
    ax1.plot(m_arr,nm_mat[np.where(n_points==8)[0][0]],label="n=8",marker="1")
    ax1.plot(m_arr,np.arccos(-1)*np.ones(m_arr.shape),label = "$\pi = \cos^{-1}(-1)}$
   ")
    ax2.plot(m_arr,err_nm_mat[np.where(n_points==2)[0][0]],label="n=2",marker="1")
    ax2.plot(m_arr,err_nm_mat[np.where(n_points==8)[0][0]],label="n=8",marker="1")
    ax1.legend();ax2.legend()

    #plt.legend()
    plt.show()
    '''
    #
    # Part (f)
    #
    #

    n_points = 2**np.arange(1,6)
    tol_arr = np.arange(2,8)
    #myttype =[('pi',float),('m',float)]
    csv_dat = np.zeros((len(n_points),len(tol_arr)*2))
    #fixed_tol_mat = np.ndarray((len(n_points),len(tol_arr),2),dtype=float)
    for i,n_i in enumerate(n_points):
        print(n_i)
        tmp = np.column_stack(MyLegQuadrature(f,0,1,n_i,m=10000,d=tol_arr))
        tmp[:,0] *= 4
        print(tmp)
        #fixed_tol_mat[i,:]= tmp
        csv_dat[i,:] = tmp.flatten()
        #print(quadrature(f,0,1,rtol = tol_arr))
    print(csv_dat)
    np.savetxt("f_data.csv",csv_dat,delimiter= ",",fmt="%.18g")

if __name__ =="__main__":
    plt.style.use("ggplot")
    #import matplotlib
    #matplotlib.use("WebAgg")
    #%matplotlib
    main()
```

4 Discussion

4.1 Validation of Python function for numerical integration

S.No.	Function	Trapezoidal	Simpson	quad 2-point	quad 4-point	Inbuilt
0	x	0.5	0.5	0.5	0.5	0.5
1	x^2	0.5	0.333333333	0.333333333	0.333333333	0.333333333
2	x^3	0.5	0.25	0.25	0.25	0.25
3	x^4	0.5	0.208333333	0.194444444	0.2	0.2
4	x^5	0.5	0.1875	0.152777778	0.166666667	0.166666667
5	x^6	0.5	0.177083333	0.12037037	0.142857143	0.142857143
6	x^7	0.5	0.171875	0.094907407	0.125	0.125
7	x^8	0.5	0.169270833	0.074845679	0.111088435	0.111111111

Table 1: (b)-Values integral calculated by various method using only elementary subintervals

- Note that trapezoidal method for one interval between 0 and 1 calculates the integral accurately for a linear function but not for higher order polynomials
- The function which calculates integral using simpson method can only give accurate result for polynomials of degree 3,but we can see the integral calculated for polynomials having degree greater than 4 are not accurate.

- we know that for a n-point gauss-quadrature rule the integral can be calculated acurately for polynomials upto degree 2n-1,which we can see in row 5 and row 7.

4.2 Estimation of π and related error and convergence of integral

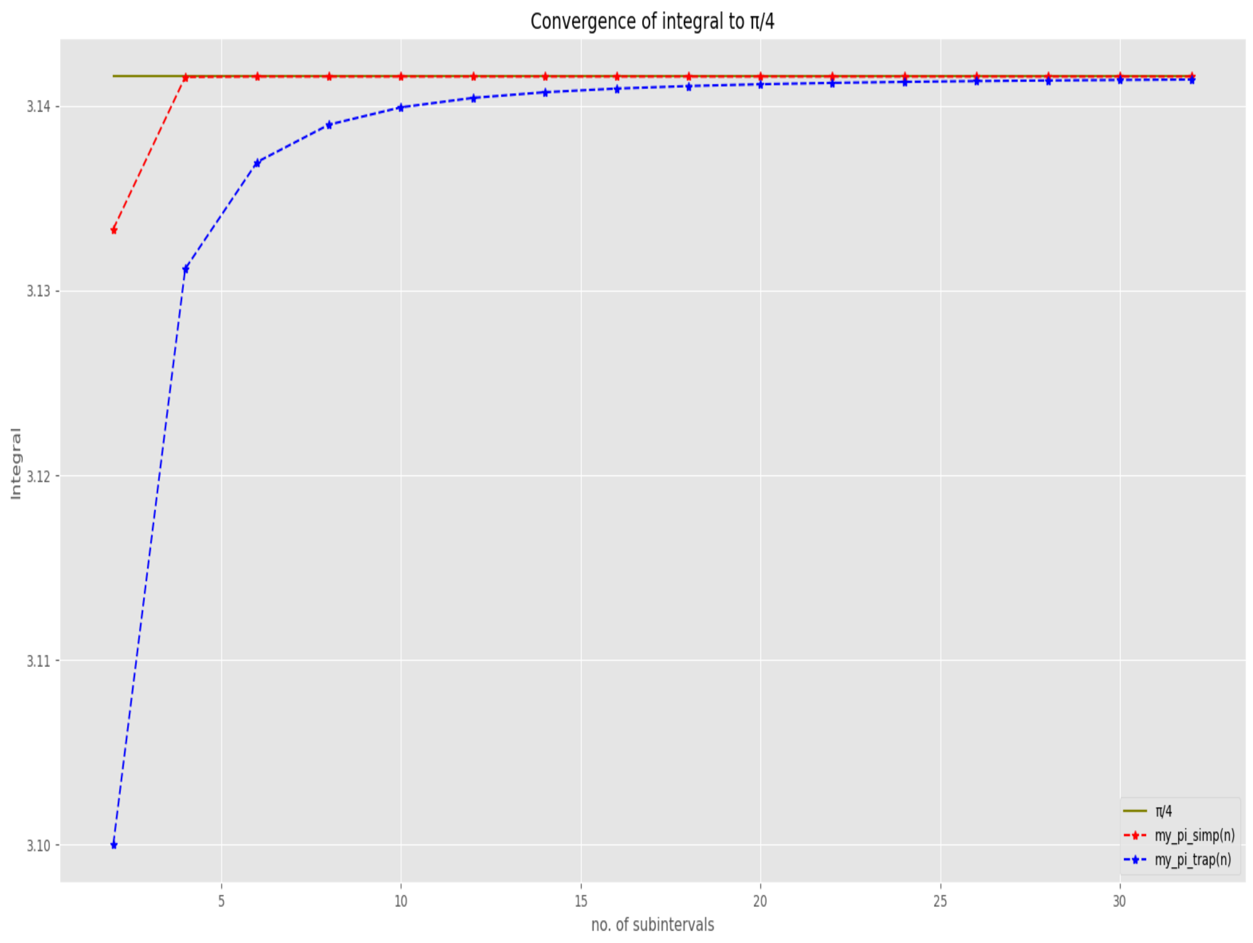


Figure 5

- We can see that as the no. of subintervals increases for the calculation of integral the value of integral approaches true value of π
- Moreover we can see that simpson method has relatively lower error and faster rate of convergence than trapezoidal method

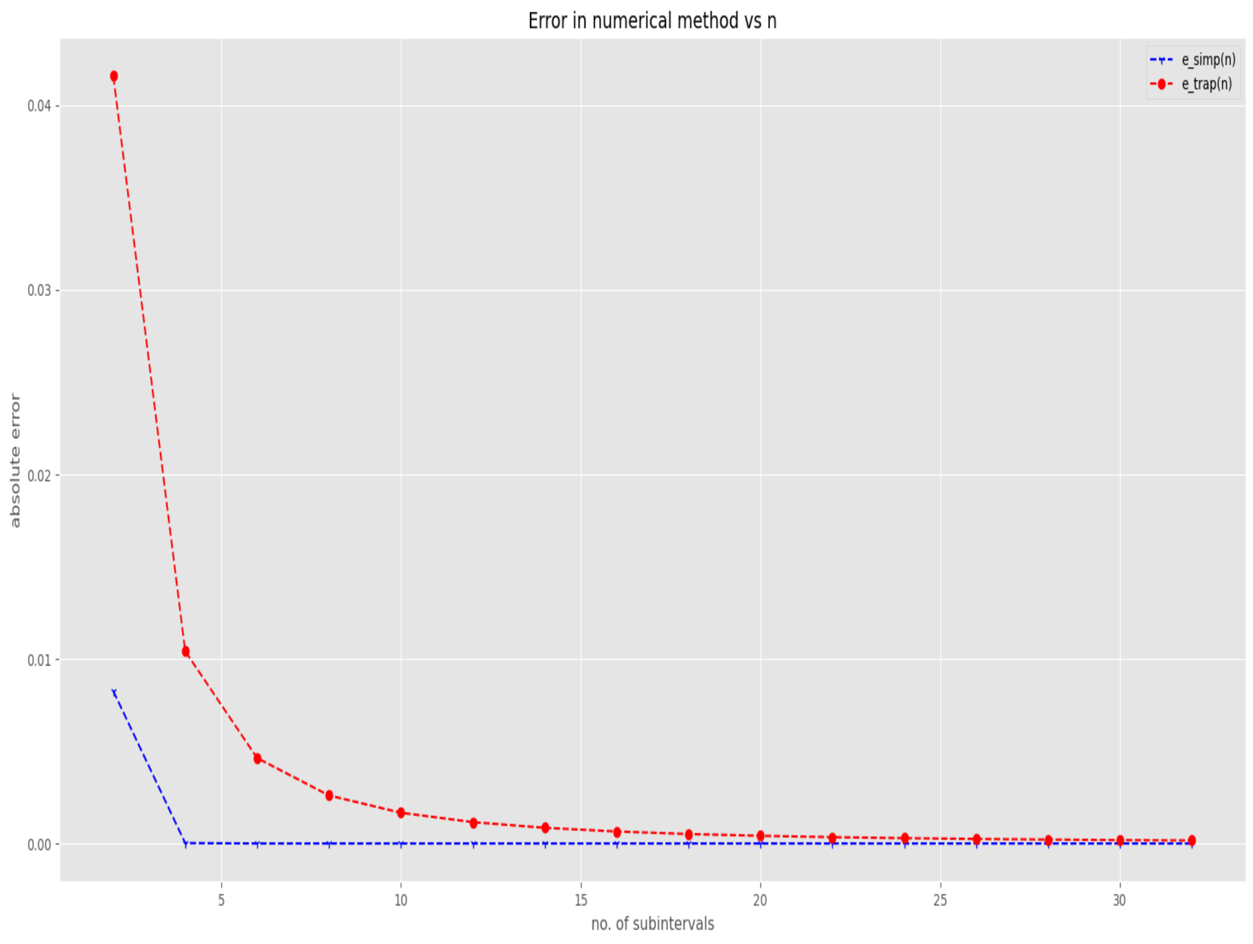


Figure 6

- The error term for trapezoidal method diminishes much more slowly than that of simpson method.

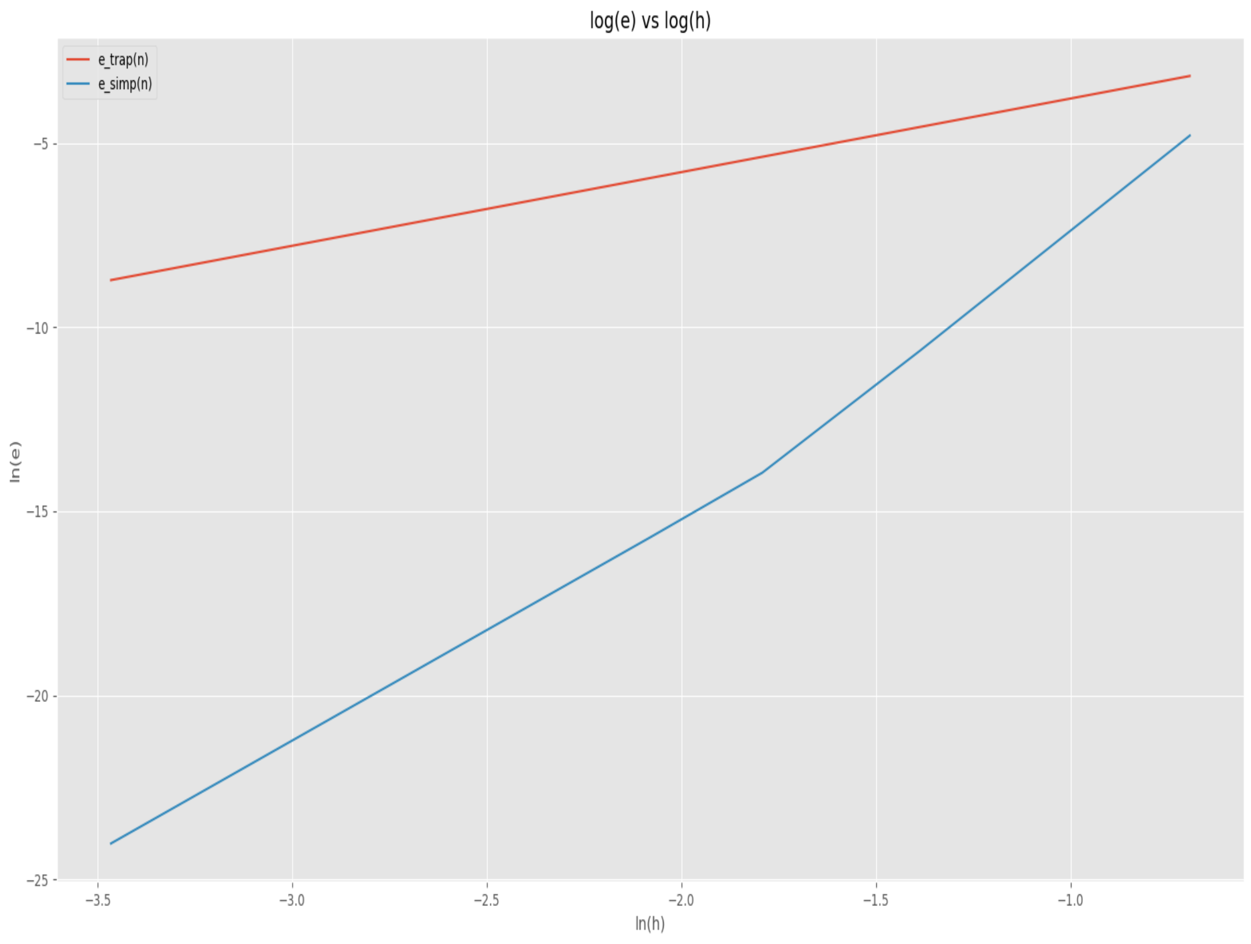


Figure 7

- We can clearly see that slope of the error line of trapezoidal method is less than the slope of the error line of the simpson method,as suggested by theory
- Hence we can conclude that simpson method is better numerical method than trapezoidal method.
- **Note:**
  The slope of error line of simpson method is - 6.652482400960333
  and the slope of error line of trapezoidal method is - 1.9996280792650376

Value of π to correct upto 5 significant digits.

	Method	Pi-calc	n	E
0	Trapezoidal	3.14159011	256	8.10E-07
1	Simpson1/3	3.141592651	16	7.53E-10

Table 2: (d)-Values of π computed numerically accurate to 5 significant digits alongwith number of intervals n.

4.3 Gauss-Legendre quadrature

n	m=1	2	4	8	16	32
2	3.1475409836065573	3.1416098930310254	3.1415927610583028	3.1415926552715474	3.1415926536160739	3.1415926535902039
4	3.1416119052458056	3.1415926996011878	3.1415926535891021	3.1415926535897922	3.1415926535897931	3.1415926535897936
8	3.1415926535191190	3.1415926535897967	3.1415926535897940	3.1415926535897931	3.1415926535897931	3.1415926535897940
16	3.1415926535897927	3.1415926535897931	3.1415926535897927	3.1415926535897931	3.1415926535897931	3.1415926535897931
32	3.1415926535897931	3.1415926535897931	3.1415926535897936	3.1415926535897931	3.1415926535897931	3.1415926535897931
64	3.1415926535897927	3.1415926535897931	3.1415926535897927	3.1415926535897922	3.1415926535897927	3.1415926535897931

Table 3: (e)-nm_matrix

- As we increase the no. of subintervals used to calculate the integral for a particular n-point formula the,integral approaches the known value of π.
- Further if we choose to keep the same no. of subintervals and change the method used i.e, increase n,then also the accuracy increases.

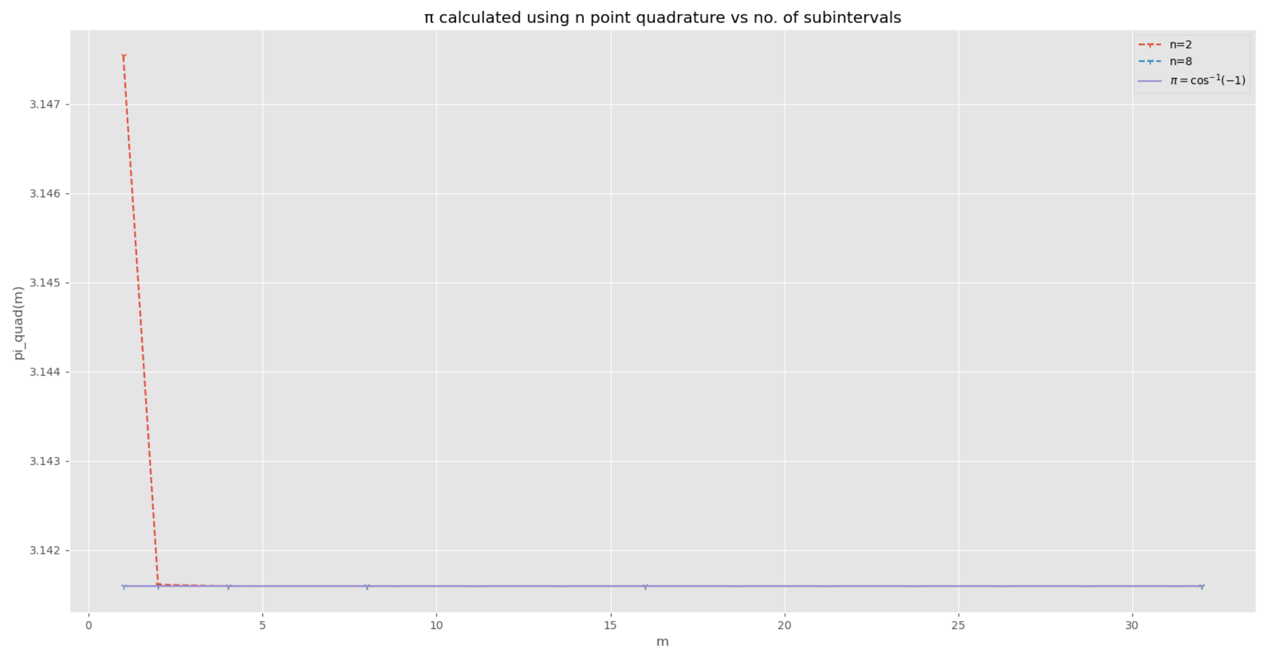


Figure 8

- Note that the integral calculated by the 8-point gauss quadrature is more accurate than 2-point quadrature for same no. of subintervals

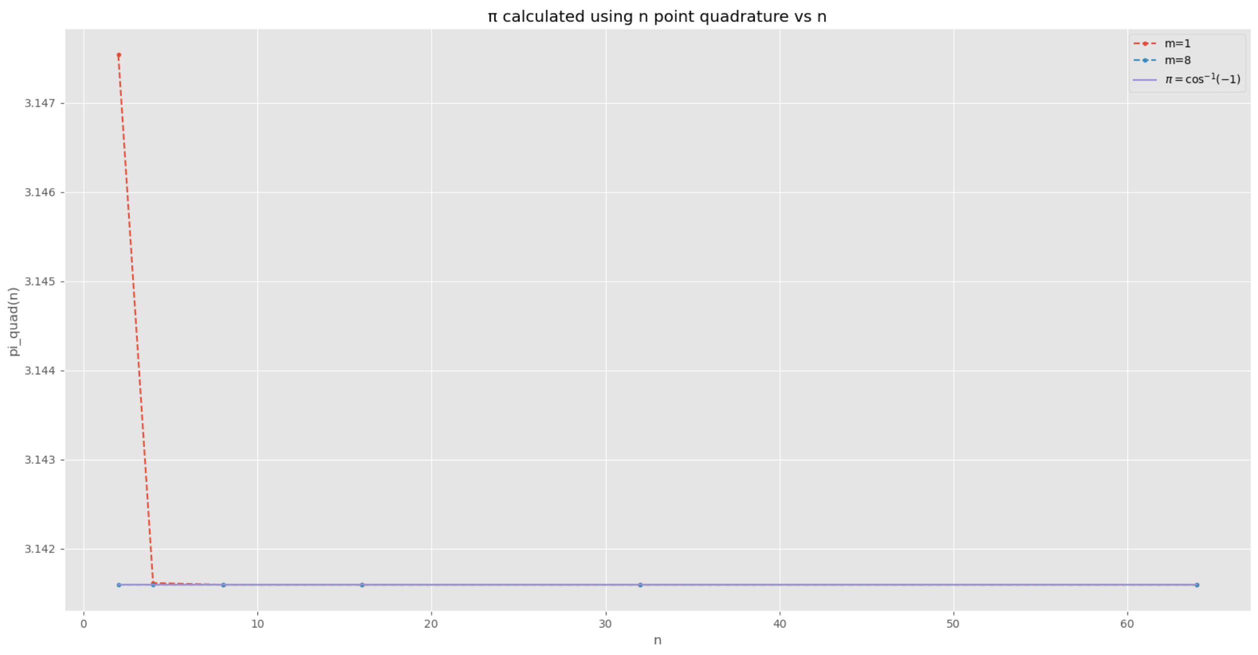


Figure 9

- Similarly here for greater number of subintervals and same n-point formula used.Then the integral calculated using greater no. of subintervals has higher accuracy

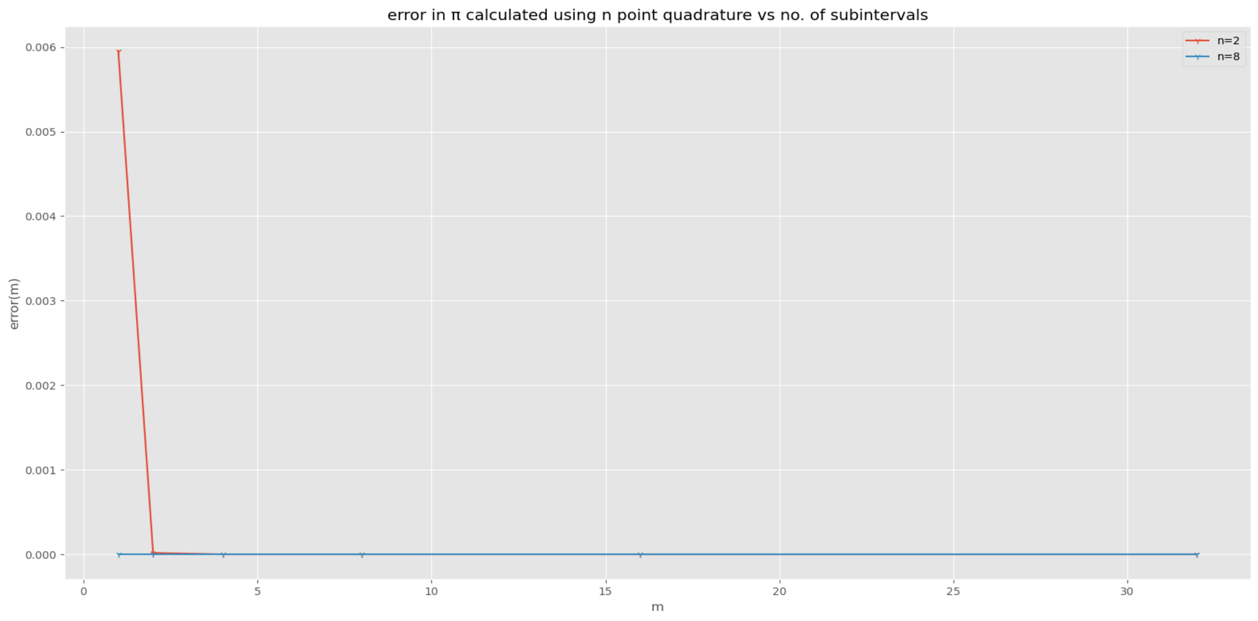


Figure 10

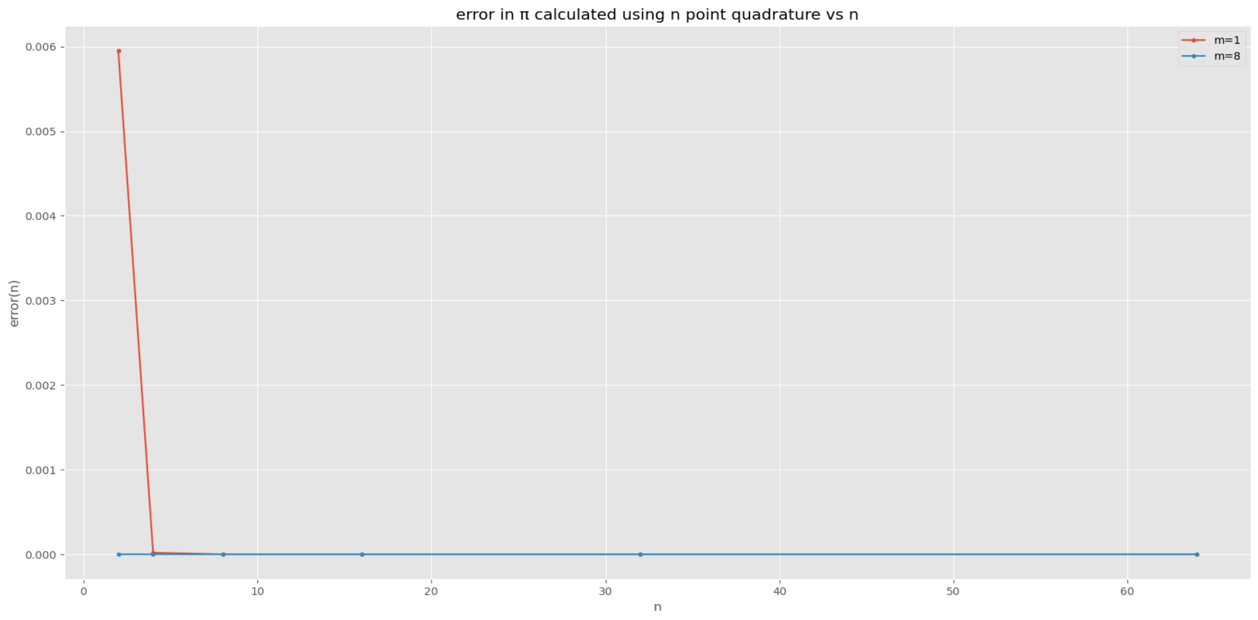


Figure 11

- When we only change the n-point formula and keep the no. of subintervals constant,then the method which is using higher no, of subintervals will diminish its error faster
- When we only increase the no. of subintervals and keep the n-point method same,then the method which is using higher endpoint formula will diminish its error faster

4.4 Value of π calculated upto given tolerence

	tol=0.5×10^{-2}		0.5×10^{-3}		0.5×10^{-4}		0.5×10^{-5}		0.5×10^{-6}		0.5×10^{-7}	
n	π	m	π	m	π	m	π	m	π	m	π	m
2	3.14160989303102545	2	3.14159276105830276	4	3.14159276105830276	4	3.14159265527154741	8	3.14159265527154741	8	3.14159265527154741	8
4	3.14159269960118781	2	3.14159269960118781	2	3.14159269960118781	2	3.14159265358910211	4	3.14159265358910211	4	3.14159265358910211	4
8	3.14159265358979667	2	3.14159265358979667	2	3.14159265358979667	2	3.14159265358979667	2	3.14159265358979667	2	3.14159265358979667	2
16	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2
32	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2	3.14159265358979312	2

Table 4: Data showing the variation of the approximate value of pie, calculated using MyLegQuadrature method, with change in tolerance and number of points used for Gauss-Legendre Quadrature.