Numerical Integration

Pawanpreet Kaur (2020PHY1092)
Monu Chaurasiya (2020PHY1102)
Prateek Bhardwaj (2020PHY1110)
S.G.T.B. Khalsa College, University of Delhi, Delhi-110007, India.

Practical Report Submitted to

Dr. Mamta

"3xxxxx - Mathematical Physics III"

Table of Contents

1 Theory

(a) Newtons Cotes Quadrature:

- 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 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 (Using Method of undetermined Coefficients):

As per method of undetermined coefficients,

Let $\int_a^b f(x)\,dx = C_1 f(a) + C_2 f(b)$ (where C_1 and C_2 are undetermined coefficients).

Let the formula be exact for $f(x) = a_o + a_1 x$

$$= \int_a^b (a_o + a_1 x)\,dx$$

$$= a_o[x]_a^b + a_1[\frac{x^2}{2}]_a^b$$

$$= a_o(b-a) + a_1(\frac{b^2-a^2}{2}) \tag{1}$$

Solving RHS, $C_1 f(a) + C_2 f(b)$

$$= C_1(a_o + a_1 a) + C_2(a_o + a_1 b)$$

$$= C_1 a_o + C_1 a a_1 + C_2 a_o + C_2 b a_1 \tag{2}$$

Equating (1) and (2)

$$a_o(b-a) + a_1(\frac{b^2-a^2}{2}) = C_1 a_o + C_1 a a_1 + C_2 a_o + C_2 b a_1$$

$$a_o(b-a)+a_1(\frac{b^2-a^2}{2})=a_o(C_1+C_2)+a_1(aC_1+bC_2)$$

Equating the terms of a_o and a_1

$$C_1+C_2=b-a \tag{3}$$

$$aC_1+bC_2=\frac{b^2-a^2}{2} \tag{4}$$

Solving equations (3) and (4) we get,

$$C_2=\frac{b-a}{2},C_1=\frac{b-a}{2}$$

Putting the values we get,

$$\boxed{\int_a^b f(x)\,dx=\frac{b-a}{2}[f(a)+f(b)]} \tag{5}$$

Simpson$_{\frac{1}{3}}$ (Using Method of undetermined Coefficients):

The Integral $\int_a^b f(x)\,dx$ can be approximated as,

$$\int_a^b f(x)\,dx=W_of(x_o)+W_1f(x_1)+W_2f(x_2) \tag{6}$$

Fixing the function arguments x_o,x_1,x_2 as $x_o=a,x_1=\frac{(a+b)}{2},x_2=b$ equispaced.

The unknown weights W_o,W_1,W_2 are to be determined.

We know Basis for polynomial of degree ≤ 2 are $f(x)=1,x,x^2$

Using Eq.6

$$\int_a^b 1\,dx=W_o+W_1+W_2 \tag{7}$$

$$\int_a^b x\,dx=W_oa+W_1\frac{(a+b)}{2}+W_2b \tag{8}$$

$$\int_a^b x\,dx=W_oa^2+W_1\frac{(a+b)^2}{(2)^2}+W_2b^2 \tag{9}$$

On solving Equations 7,8,9 we get value of W_0,W_1,W_2

$$W_o=\frac{b-a}{6},W_1=\frac{2(b-a)}{3},W_2=\frac{b-a}{6}$$

Putting these values in (6).

We get,

$$\boxed{\int_a^b f(x)\,dx=\frac{b-a}{6}[f(a)+4f(\frac{a+b}{2})+f(b)]} \tag{10}$$

- The Error Term in Trapezoidal is given by $\frac{-1}{12}h^3 f''(\xi)$

The Error Term in $Simpson_{\frac{1}{3}}$ is $\frac{-1}{90}h^5 f''''(\xi)$

The Error Term in $Simpson_{\frac{3}{8}}$ is $\frac{-3}{80}h^5 f''''(\xi)$

Where $h = \frac{b-a}{n}$, 'n' be the number of intervals and in all the above mentioned formulas we can clearly see that Error is inversely proportional to 'n' which means as we increase the 'n', error reduces and so as to get more and more accurate result we need to increase the number of intervals 'n'.

Composite Form for Trapezoidal :

$$\boxed{\int_a^b f(x)\,dx = \sum_{j=1}^{n} \frac{x_j - x_{j-1}}{2}[f(x_{j-1}) + f(x_j)] - \frac{h^2}{12}(b-a)f''(\xi)}$$

Composite Form for $Simpson_{\frac{1}{3}}$:

$$\boxed{\int_a^b f(x)\,dx = \sum_{j=1}^{n} \frac{x_j - x_{j-1}}{6}[f(x_{j-1}) + 4f(\frac{x_{j-1}+x_j}{2}) + f(x_j)] - \frac{h^4}{180}(b-a)f''''(\xi)}$$

Composite Form for $Simpson_{\frac{3}{8}}$:

This rule is more accurate than the standard method, as it uses one more functional value Simpson's 3/8 rule for n intervals (n should be a multiple of 3):

$$\boxed{\int_a^b f(x)\,dx \approx \frac{3h}{8}\sum_{j=1}^{n/3}[f(x_{3j-3}) + 3f(x_{3j-2}) + 2f(x_{3j-1}) + f(x_{3j}))] - \frac{(b-a)}{80}h^4 f''''(\xi)}$$

where $x_j = a + jh$ for j = 0,1,...,n-1,n with $h = \frac{(b-a)}{n}$; in particular, $x_0 = a$ and $x_n = b$.

- **Geometrical Interpretation of Trapezoidal:**

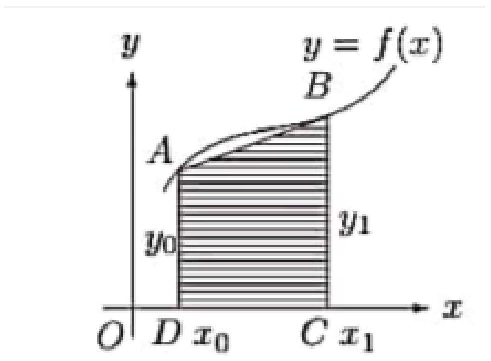


Figure 1: Trapezoidal

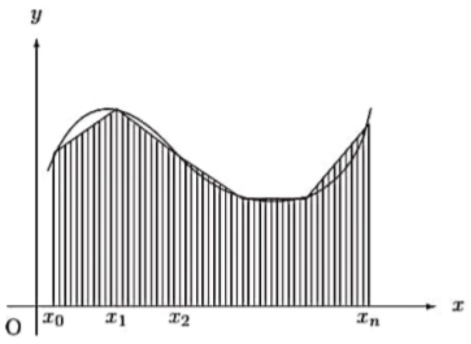


Figure 2: Composite Trapezoidal

In trapezoidal rule,the curve $y = f(x)$ is replaced by the line joining the points $A(x_o, y_o)$ and $B(x_1, y_1)$ (in fig:1).Thus the area bounded by the curve $y = f(x)$, the ordinates $x = x_o, x = x_1$ and the x-axis is then approximately equivalent to the area of the trapezium(ABCD) bounded by the line AB,$x = x_o, x = x_1$ and x-axis.
The Geometrical significance of composite trapezoidal rule is that the curve $y = f(x)$ is replaced by n straight lines joining the points (x_o, y_o) and (x_1, y_1) ; (x_1, y_1) and (x_2, y_2) ; ...,(x_{n-1}, y_{n-1}) and (x_n, y_n).Then the area bounded by the curve $y = f(x)$, the lines $x = x_o, x = x_n$ and the x-axis is then approximately equivalent to the sum of the area of n trapeziums.

- **Geometrical Interpretation of Simpson's Rule:**
  In Simpsons rule, the curve $y = f(x)$ is replaced by the second degree parabola passing

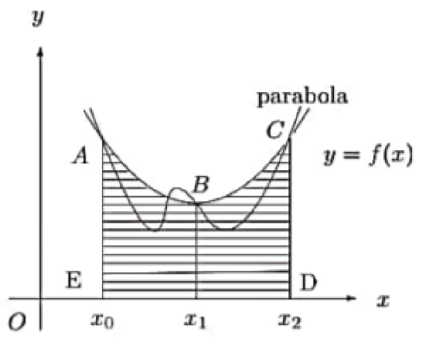


Figure 3: Simpson

through the points $A(x_o, y_o)$,$B(x_1, y_1)$ and $C(x_2, y_2)$. Therefore, the area bounded by the curve $y = f(x)$, the ordinates $x = x_o$,$x = x_2$ and the x-axis is approximated to the area bounded by the parabola ABC, the straight lines $x = x_o$,$x = x_2$ and x-axis i:e, the area of the shaded region ABCDEA.

- **Condition for No. of Intervals:**
  We know that Truncation error decreases as number of intervals increases or h decreases.But there is no point in making 'h' so small that the approximation error becomes much smaller than the rounding error,decreasing h will only be beneficial up to the point at which the truncation and the rounding errors are roughly equal.
  Roundoff Error= machine epsilon times the value of intergral=$\varepsilon I(f)$
  For Trapezoidal:

$$\frac{h^2}{12}[f'(a) - f'(b)] \simeq \varepsilon \int_a^b f(x)\,dx$$

$$h \simeq \left[\frac{12\varepsilon \int_a^b f(x)\,dx}{f'(a) - f'(b)}\right]^{1/2}$$

$$N \simeq \left[\frac{f'(a) - f'(b)}{12 \int_a^b f(x)\,dx}\right]^{1/2} \varepsilon^{-1/2}$$

If $\left[\frac{f'(a)-f'(b)}{12\int_a^b f(x)\,dx}\right]^{1/2} \simeq 1$ and $\varepsilon \simeq 10^{-16}$ for double precision.

So we get $N \simeq 10^8$ for Trapezoidal

For Simpson:

$$\frac{h^4}{180}[f'''(a) - f'''(b)] \simeq \varepsilon \int_a^b f(x)\,dx$$

$$h \simeq \left[\frac{180\varepsilon \int_a^b f(x)\,dx}{f'''(a) - f'''(b)}\right]^{1/4}$$

$$N \simeq \left[\frac{f'''(a) - f'''(b)}{180\int_a^b f(x)\,dx}\right]^{1/4} \varepsilon^{-1/4}$$

If $\left[\frac{f'''(a)-f'''(b)}{180\int_a^b f(x)\,dx}\right]^{1/4} \simeq 1$ and $\varepsilon \simeq 10^{-16}$ for double precision.

So we get $N \simeq 10^4$ for Simpson

- **Error Terms:**

  The error terms in trapezoidal is given by : $-\frac{(b-a)h^2}{12}f''(\xi)$

  So we can clearly see from the error term that in trapezoidal as we get second derivative for a function to be zero then error term becomes identical to zero.

  Error term in simpson:$-\frac{(b-a)h^4}{180}f''''(\xi)$

  So error term in simpson become zero for those functions having fourth derivative as zero.

  No we can not keep on increasing the number of intervals in order to bring down the error because as we increase the number of intervals 'h' value decreases.

  Although the truncation error decreases as the number of intervals 'h' decreases , there is no point in making 'h' so small that approximation error becomes much smaller than the rounding error. So we should only decrease h up to the point at which rounding error is almost equal to truncation error.

(b) Legendre Gauss Quadrature:

- **Gauss Quadrature :**

  Gauss quadrature formula is the highest algebraic accuracy of interpolation quadrature formula. By reasonably selecting quadrature nodes and quadrature coefficients of the form of

$$\int_a^b f(x)\,dx \approx \sum_{k=0}^{n} A_k f(x_k)$$

  We can obtain the interpolation quadrature formula with the highest algebraic accuracy; that is,$2n+1$

  The reason why gauss quadrature and newton cotes differ is that in the newton cotes method the integral of a function is approximated by the sum of its functional values at a set of equally spaced points multiplied by some chosen weighing coefficients.However the idea of Gaussian quadrature is to give us the freedom to choose not only weighing coefficients,but also to optimally decide the location of abscissas at which the function is to be evaluated.They will no longer be equally spaced,thus we will have twice the number of degrees of freedom at our disposal.

- **How Gauss quadrature method is linked with a set of orthogonal polynomials**

  If $p(x)$ is polynomial of degree and $2n-1$, $q(x)$ is quotient of degree $n-1$ or less and L_n is n^{th} degree of legendre polynomial

$$p(x) = q(x)L_n(x) + r(x)$$

  On integrating the above equation from -1 to 1,

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

  If $p(x)$ is polynomial of degree $2n-1$ and L_n is is quotient of degree $n-1$ or less. Term I in above equation goes to zero due to orthonormal property of legendre polynomials. Therefore integral of polynomial $p(x)$ becoms equal to integral of remainder $r(x)$.

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

- **For changing the formula $\int_a^b f(x)\,dx$ to $\int_{-1}^1 f(t)dt$**

  The Gauss legendre formula have its weight function as unity so not explicitly displayed in integrand. Gauss legendre quadrature can be applied to integral over any finite range though the Legendre Polynomial $P_l(x)$ on which it is based are only defined and orthogonal over the interval $-1 \leq x \leq 1$. Therefore in order to use their properties, the integral between the limits 'a' and 'b' has to be changed to one between the limits -1 and +1.

  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,

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

- **Derivation of 2-Point Gauss Quadrature Formula**:

  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 . So in the two-point Gauss quadrature rule, the integral is approximated as

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$$

  This is exact for polynomials of degree upto 2n-1=3 (for n=3).
  that is x_0, x_1, x_2, x_2.
  These will give four equations as follows:

$$\int_a^b 1dx = b - a = c_1 + c_2$$

$$\int_a^b xdx = \frac{b^2 - a^2}{2} = c_1 x_1 + c_2 x_2$$

$$\int_a^b x^2 dx = \frac{b^3 - a^3}{3} = c_1 x_1^2 + c_2 x_2^2$$

$$\int_a^b x^3 dx = \frac{b^4 - a^4}{4} = c_1 x_1^3 + c_2 x_2^3$$

  These four simultaneous nonlinear equations can be solved to give a single acceptable solution :

$$c_1 = \frac{b-a}{2}$$
$$c_2 = \frac{b-a}{2}$$
$$x_1 = (\frac{b-a}{2})(\frac{-1}{\sqrt{3}}) + \frac{b+a}{2}$$
$$x_2 = (\frac{b-a}{2})(\frac{1}{\sqrt{3}}) + \frac{b+a}{2}$$

Hence,

$$\boxed{\int_a^b f(x)dx \approx \frac{b-a}{2} f(\frac{b-a}{2}(\frac{-1}{\sqrt{3}}) + \frac{b+a}{2}) + \frac{b-a}{2} f(\frac{b-a}{2}(\frac{1}{\sqrt{3}}) + \frac{b+a}{2})}$$

- **n-point Gauss Quadrature rule:**

  It approximates the integral as ,

$$\int_{-1}^{1} f(x)\,dx \approx C_1 f(x_1) + C_2 f(x_2) + \ldots.. + C_n f(x_n) \tag{11}$$

So we have $2n$ number of unknowns and so as to find their values we assume $(2n-1)^{th}$ degree polynomial.
Equation 11 can also be generalised as,

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

Algorithm 1 Trapezoidal Method

```
function MYTRAP($f, a, b, n_0, key_1 = True, N_{max} = None, key_2 = False, tol = None$)
                                        ▷ f:function, a:initial, b:final, n0:no.of terms
  ▷ key$_1$:boolean true to calculate the intermidiate values by reducing step size and avoiding
  the repeat calculation.
                      ▷ $N_{max}$: optional parameter given by the user to set maximum no. of terms
                                ▷ key$_2$: boolean true if we want the value in the range of tolerance
                                        ▷ tol: optional parameter given for tolerance

    h=(b-a)/n0                                  ▷ calculate the step size using a,b and n0
    S=0.5*(f(a)+f(b))
    for i do in range(1,n0):
        S+= f(a+i*h)            ▷ To calculate the sum of f(function) with increasing step size.
        Integral = S * h
        ▷ To reduce step size to half continuously and calculate the value without repeat
        calculation

        n$_a = [n0]$
        I=[Integral]
        n0=2*n0                                                    ▷ Increase n0=2*n0
        while n0 $\leq N_{max}$ :

        h=(b-a)/n0
        r=0
        for i do in range(1,n0):
            if $i/2! = 0$ :                   ▷ for terms not divisible by 2 calculate the value
            r+=f(a+i*h)
            r=r*h
            I.append((I[-1]/2)+(r))                             ▷ Appending values to list I
            n$_a.append(n0)$

            if key2==True:
            err=abs((I[-1]-I[-2])/I[-1])
            if err $\leq tol$ :
            ▷ If tolerance is given calculate the relative error by taking value from the list I and compare er
```

Algorithm 2 Simpson Method

```
function MYSIMP($f, a, b, n_0, key_1 = True, N_{max} = None, key_2 = False, tol = None$)
                                        ▷ f:function, a:initial, b:final, n0:no.of terms
  ▷ key$_1$:boolean true to calculate the intermidiate values by reducing step size and avoiding
  the repeat calculation.
                      ▷ $N_{max}$: optional parameter given by the user to set maximum no. of terms
                           ▷ key$_2$: boolean true if we want the value in the range of tolerance
                                        ▷ tol: optional parameter given for tolerance

    if $n0/2 == 0$:              ▷ conditioning statement to take even number of intervals.
    pass
    else :
    return Number of intervals must be even
    $S_a = []; T_a = []; I_a = []$
    h=(b-a)/n0
    S = f(a) + f(b)               ▷ for even terms initial sum is f(a) +f(b) and for odd 0
    ▷ using for loop to calculate the sum of f(function) with increasing step size. If term is
    even Sum is calculated as S+2*f(a+i*h) and if it is odd the Sum is calculated as (2 * f(a
    + i*h))/3

    for i do in range(1,n0):
      if i/2 == 0:
      S = S + 2 * f(a + i*h) ▷ For integrated value add 1/3rd sum of even terms with twice
      the sum of odd tems and multiply whole with step size.
      else:    T = T + (2 * f(a + i*h))/3    S=S/3
      Integral =h*(S+2*T)
      while $n_0 \leq N_{max}$:
      h=(b-a)/n0
      T=0
      for i do in range(1,n0):
        if i
        T+=(2*f(a+i*h))/3
        S=$S_a[-1] + T_a[-1]$
        Integral =h*(S+2*T) ▷ If $N_{max}$ reached without achieving required tolerance print
        it and return integrated value and number of terms or if tolerance is achieved
        print it.
        if key2==True: $\rightarrow err = abs((I_a[-1] - I_a[-2])/I_a[-1])$

        if $err \leq tol$:$\rightarrow w = 1$
```

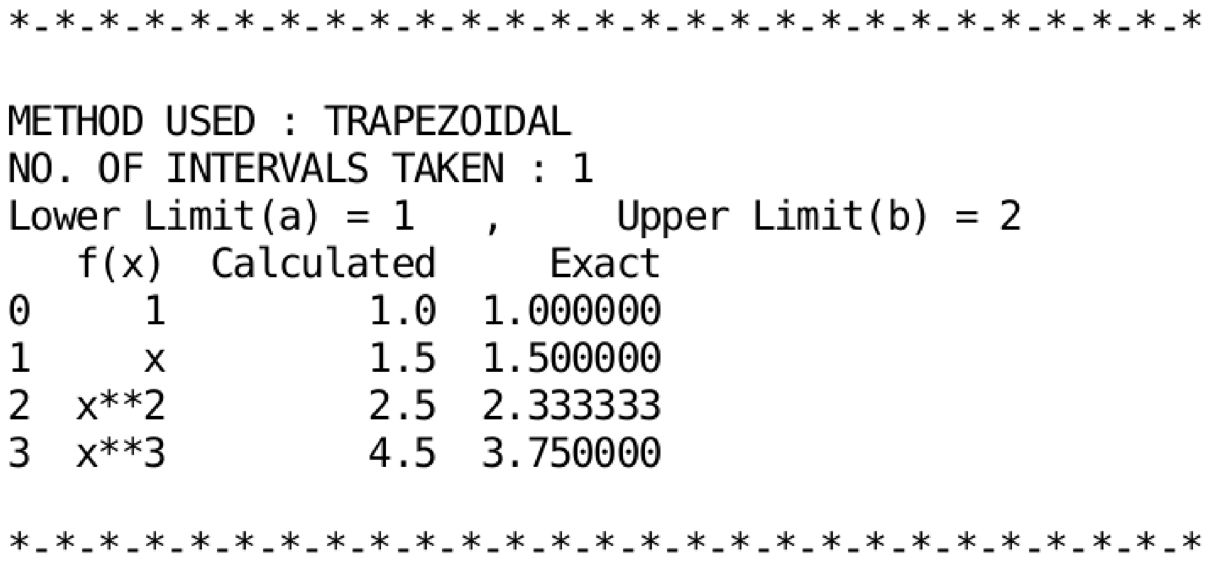
Algorithm 3 Legendre-Gauss Quadrature

```
function MYLEGQUADRATURE($f, a, b, n, m, key = False, tol = None, m_{max} = None$)
                    ▷ f:function, a:initial, b:final, n:degree or weight m:no. of subintervals
                    ▷ key:boolean true if we want to calculate the values by reducing step size
                    ▷ $m_{max}$: optional parameter given by the user to set maximum no. of intevals
                    ▷ tol: optional parameter given for tolerance

    def gs1(f,a,b,n,m0):    ▷ nested function takes the parameters f,a,b,n,m0 and return sum
    x,w = np.polynomial.legendre.leggauss(n) ▷ using inbuilt function which take n as input
    and returns x and y two arrays containing Number of sample points and weights respec-
    tively.
    h=(b-a)/m                                                        ▷ Calculating step size
    s=0
    ▷ using nested for loop to calculate the sum by using weights and sample points
    from inbuilt function and putting weights and sample points in n-point guass legendre
    quadrature formula.

    for i do in range(0,m):
        r=0
        for x do1,w1 in zip(x,w):
            r+=w1*f((((a+(i+1)*h)-(a+i*h))/2)*x1+((a+i*h)+(a+(i+1)*h))/2)
            r= (((a+(i+1)*h)-(a+i*h)) /2 )*r
            s+=r return sum
            m$_a = [m]$
            I=[Integral]
            m=2*m ▷ To reduce step size to half continuously and calculate the value. In-
            crease m=2*m and calculate the value and append it in a list I.
            while $m \leq m_{max}$:
            I.append(gs1(f,a,b,n,m))
            m$_a.append(m)$
            ▷ If tolerance is given calculate the relative error by taking value from the list I and compare er
            err=abs((I[-1]-I[-2])/I[-1])
            if err $\leq tol$ :
            w=1
            break
            else:
            pass
            m=2*m
```

2 Results and Discussion

2.1 Q3 (b) : Validation the functions MyTrap, MySimp and MyLegQuadrature

The Trapezoidal Method

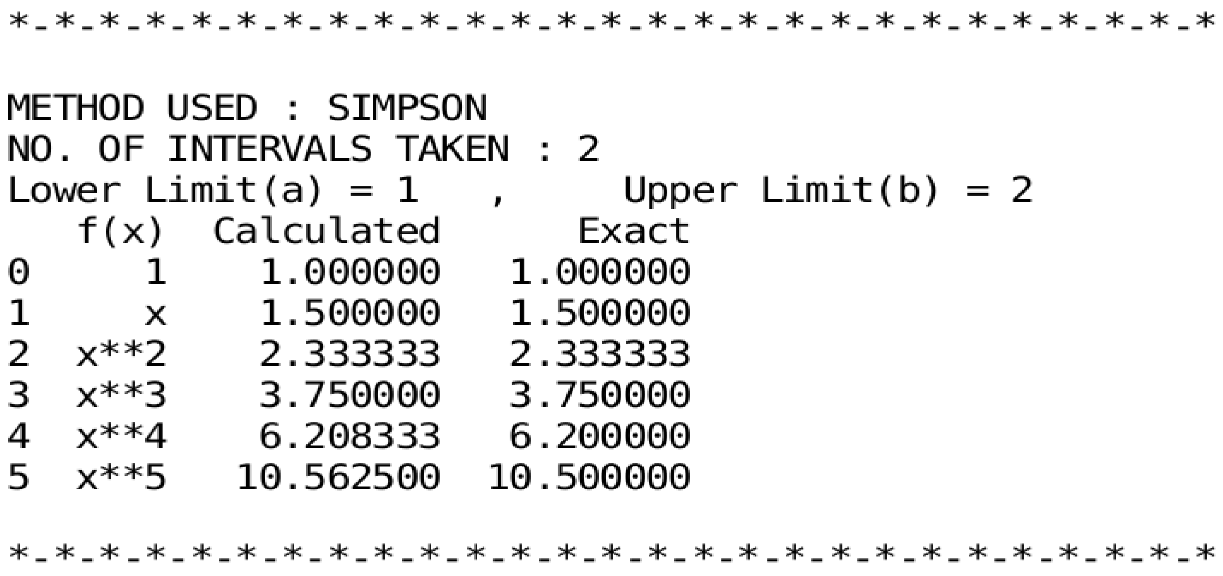


```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

METHOD USED : TRAPEZOIDAL
NO. OF INTERVALS TAKEN : 1
Lower Limit(a) = 1   ,    Upper Limit(b) = 2
   f(x)  Calculated     Exact
0     1         1.0  1.000000
1     x         1.5  1.500000
2  x**2         2.5  2.333333
3  x**3         4.5  3.750000

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 4: *The trapezoidal method with one interval for different functions*

From the above table it can be understood that the trapezoidal method with one interval gives exact result for a linear function but not for a polynomial of order 2 or above.

Simpson 1/3 Method

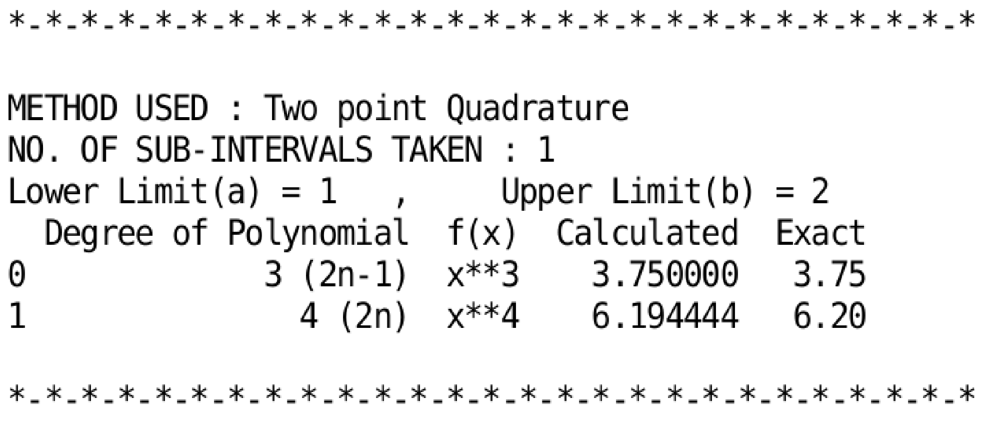


```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

METHOD USED : SIMPSON
NO. OF INTERVALS TAKEN : 2
Lower Limit(a) = 1   ,     Upper Limit(b) = 2
   f(x)  Calculated      Exact
0     1    1.000000   1.000000
1     x    1.500000   1.500000
2  x**2    2.333333   2.333333
3  x**3    3.750000   3.750000
4  x**4    6.208333   6.200000
5  x**5   10.562500  10.500000

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 5: *The simpson 1/3 method with two intervals for different functions*

From the above table it can be understood that the Simpson method with two intervals gives exact result for a polynomial of degree less than or equal to 3 but not for a polynomial of order 4.

Gauss Legendre Method

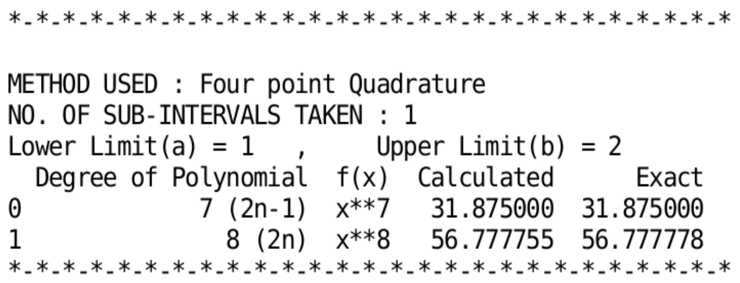


```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

METHOD USED : Two point Quadrature
NO. OF SUB-INTERVALS TAKEN : 1
Lower Limit(a) = 1   ,     Upper Limit(b) = 2
  Degree of Polynomial  f(x)  Calculated  Exact
0             3 (2n-1)  x**3    3.750000   3.75
1               4 (2n)  x**4    6.194444   6.20

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 6: *The the two-point (n = 2) quadrature with only one sub-interval i.e. m = 1*

From the above table it can be understood that the the two-point (n = 2) quadrature with only one sub-interval i.e. m = 1 gives exact result for a polynomial of degree less than or equal to 3

(2n 1) but not for a polynomial of degree 4 (2n).



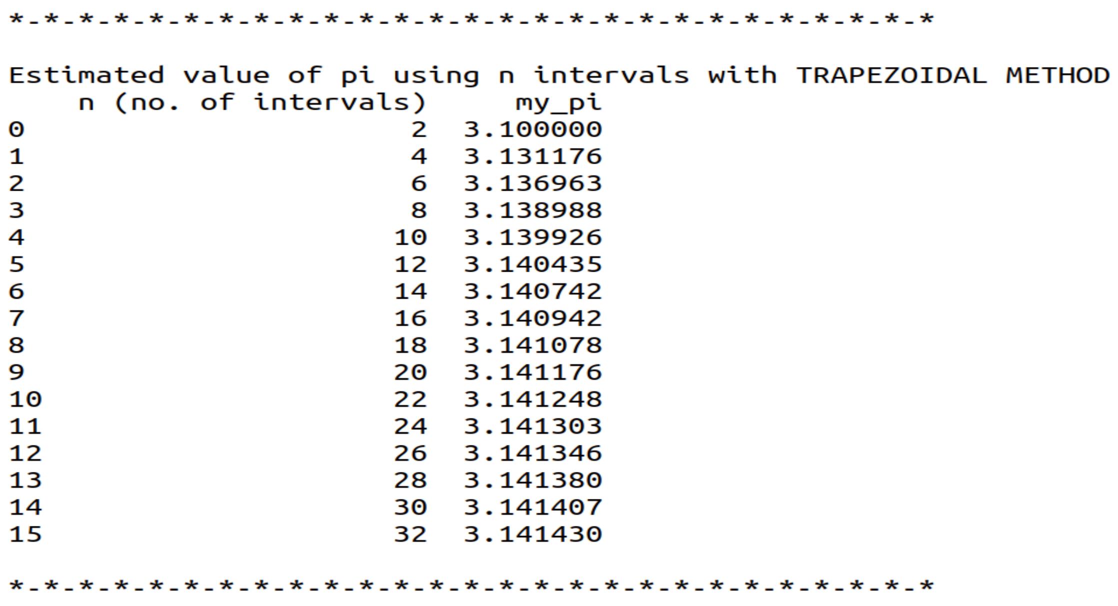
```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

METHOD USED : Four point Quadrature
NO. OF SUB-INTERVALS TAKEN : 1
Lower Limit(a) = 1    ,     Upper Limit(b) = 2
  Degree of Polynomial  f(x)  Calculated      Exact
0             7 (2n-1)  x**7   31.875000  31.875000
1               8 (2n)  x**8   56.777755  56.777778
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 7: *The the four-point (n = 4) quadrature with only one sub-interval i.e. m = 1*

From the above table it can be understood that the the four-point (n = 4) quadrature with only one sub-interval i.e. m = 1 gives exact result for a polynomial of degree less than or equal to 7 (2n 1) but not for a polynomial of degree 8 (2n).

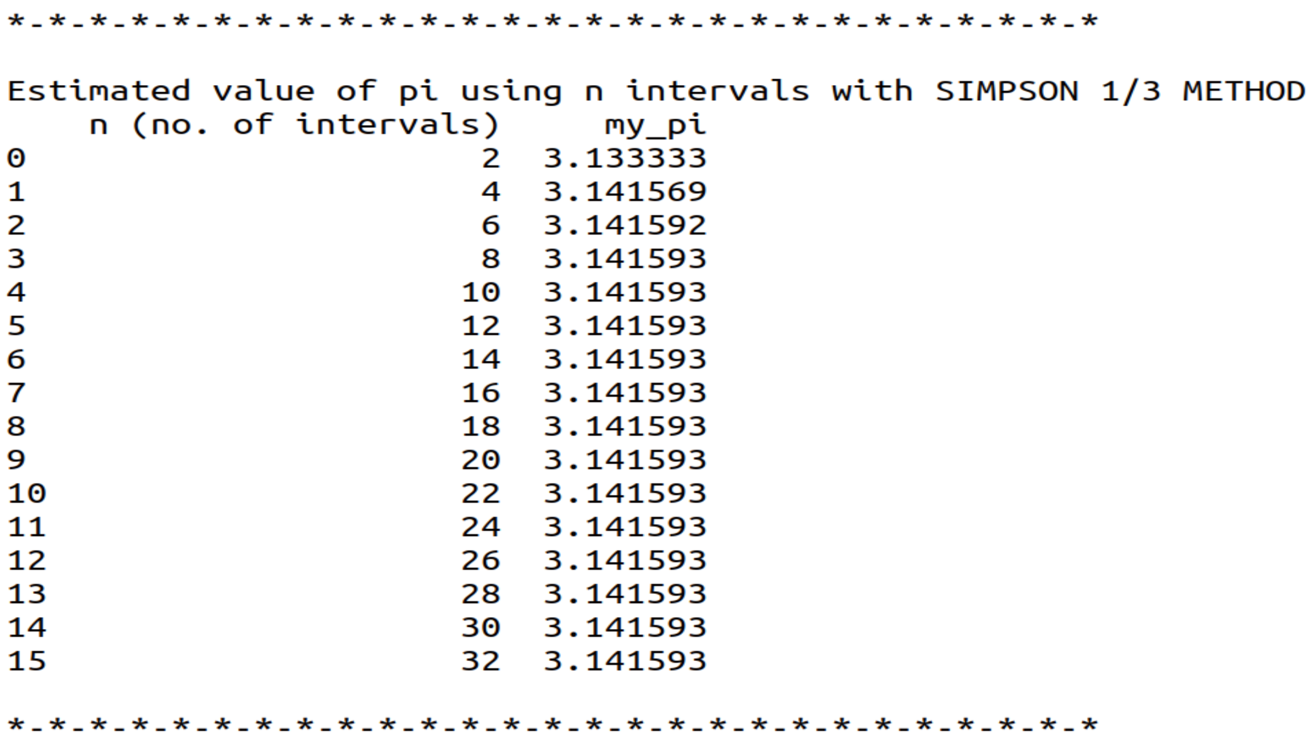
2.2 Q3 (c) : Use of MyTrap and MySimp to estimate the value of π

- **(i)**



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Estimated value of pi using n intervals with TRAPEZOIDAL METHOD
    n (no. of intervals)      my_pi
0                       2   3.100000
1                       4   3.131176
2                       6   3.136963
3                       8   3.138988
4                      10   3.139926
5                      12   3.140435
6                      14   3.140742
7                      16   3.140942
8                      18   3.141078
9                      20   3.141176
10                     22   3.141248
11                     24   3.141303
12                     26   3.141346
13                     28   3.141380
14                     30   3.141407
15                     32   3.141430

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 8: *Estimated value of pi using n intervals with Trapezoidal Method*



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

Estimated value of pi using n intervals with SIMPSON 1/3 METHOD
    n (no. of intervals)     my_pi
0                       2  3.133333
1                       4  3.141569
2                       6  3.141592
3                       8  3.141593
4                      10  3.141593
5                      12  3.141593
6                      14  3.141593
7                      16  3.141593
8                      18  3.141593
9                      20  3.141593
10                     22  3.141593
11                     24  3.141593
12                     26  3.141593
13                     28  3.141593
14                     30  3.141593
15                     32  3.141593

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 9: *Estimated value of pi using n intervals with Simpson 1/3 Method*

- **(ii) Plot of my pi(n) as a function of n for both Trapezoidal Method and Simpson 1/3 Method.**

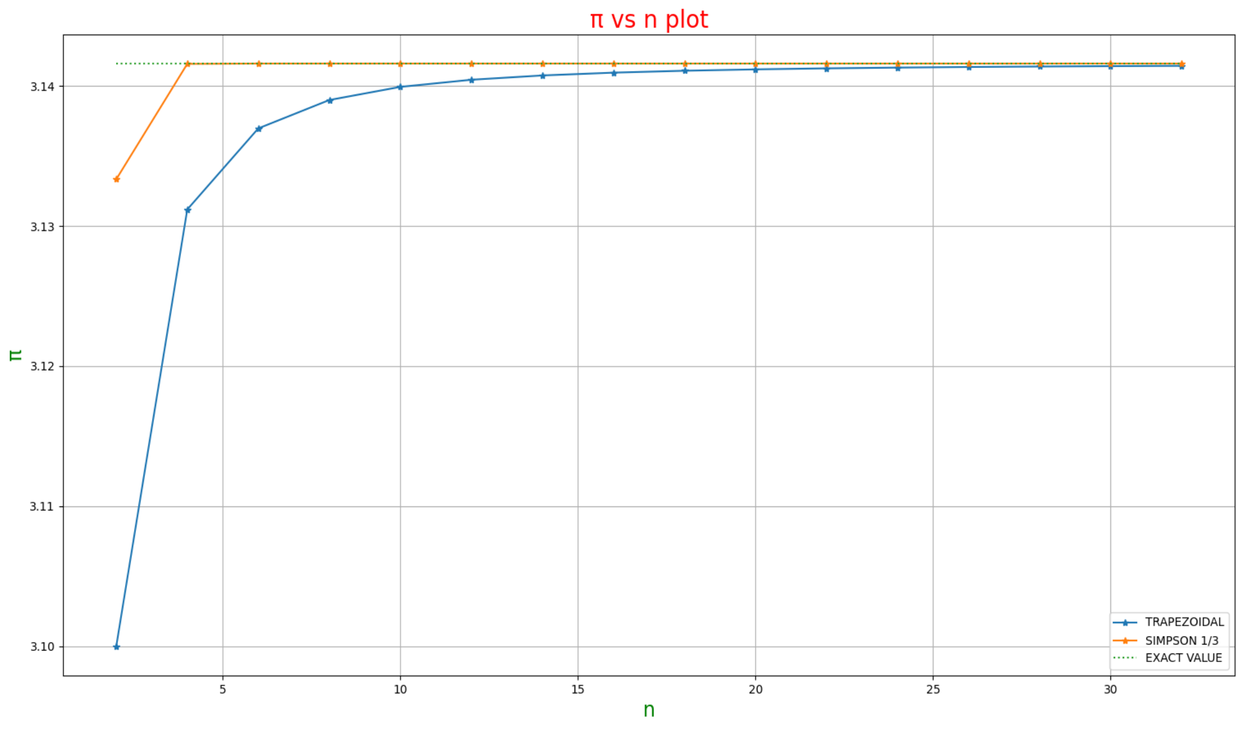
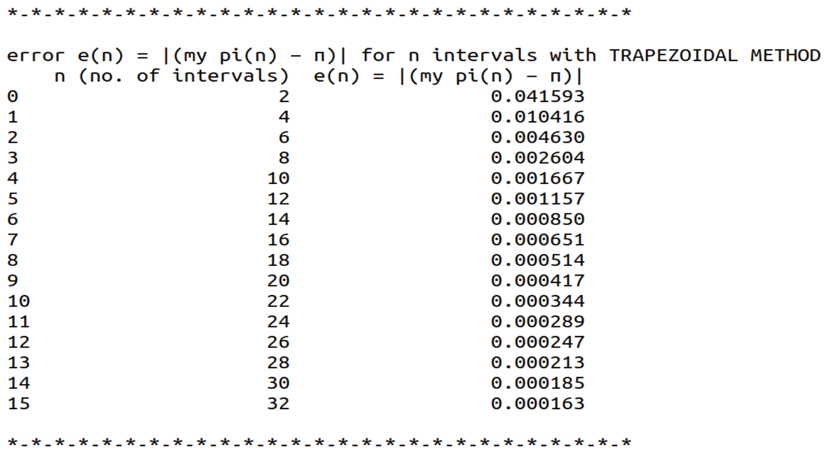


Figure 10: *my pi(n) vs n plot*

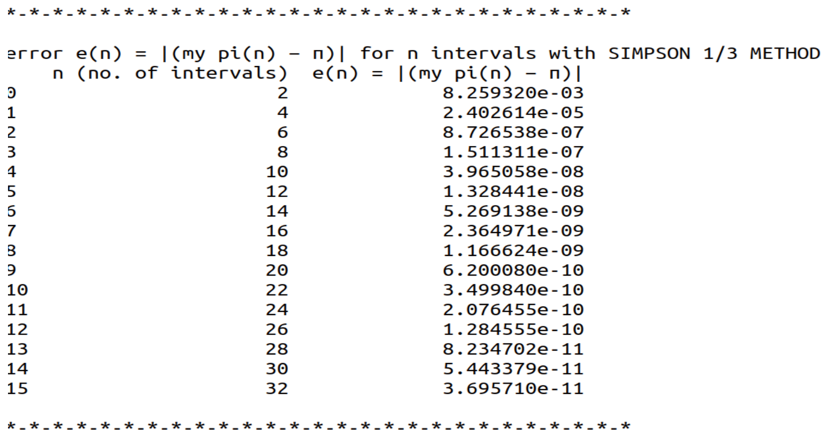
From the graph we can see that the value of pi obtained by trapezoidal and simpsons is approaching towards true value as n increases and we can see that values obtained by trapezoidal is not accurate in comparison to the values obtained by simpson 1/3 method.

- **(iii) error e(n) = $|(mypi(n) - \pi)|$ calculation for each n for both trapezoidal and simpson 1/3 method**



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

error e(n) = |(my pi(n) - π)| for n intervals with TRAPEZOIDAL METHOD
    n (no. of intervals)  e(n) = |(my pi(n) - π)|
0                      2               0.041593
1                      4               0.010416
2                      6               0.004630
3                      8               0.002604
4                     10               0.001667
5                     12               0.001157
6                     14               0.000850
7                     16               0.000651
8                     18               0.000514
9                     20               0.000417
10                    22               0.000344
11                    24               0.000289
12                    26               0.000247
13                    28               0.000213
14                    30               0.000185
15                    32               0.000163

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 11: $|(mypi(n) - \pi)|$ *for each n for TRAPEZOIDAL METHOD*



```
*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*

error e(n) = |(my pi(n) - π)| for n intervals with SIMPSON 1/3 METHOD
    n (no. of intervals)  e(n) = |(my pi(n) - π)|
0                      2           8.259320e-03
1                      4           2.402614e-05
2                      6           8.726538e-07
3                      8           1.511311e-07
4                     10           3.965058e-08
5                     12           1.328441e-08
6                     14           5.269138e-09
7                     16           2.364971e-09
8                     18           1.166624e-09
9                     20           6.200080e-10
10                    22           3.499840e-10
11                    24           2.076455e-10
12                    26           1.284555e-10
13                    28           8.234702e-11
14                    30           5.443379e-11
15                    32           3.695710e-11

*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 12: $|(mypi(n) - \pi)|$ *for each n for SIMPSON 1/3 METHOD*

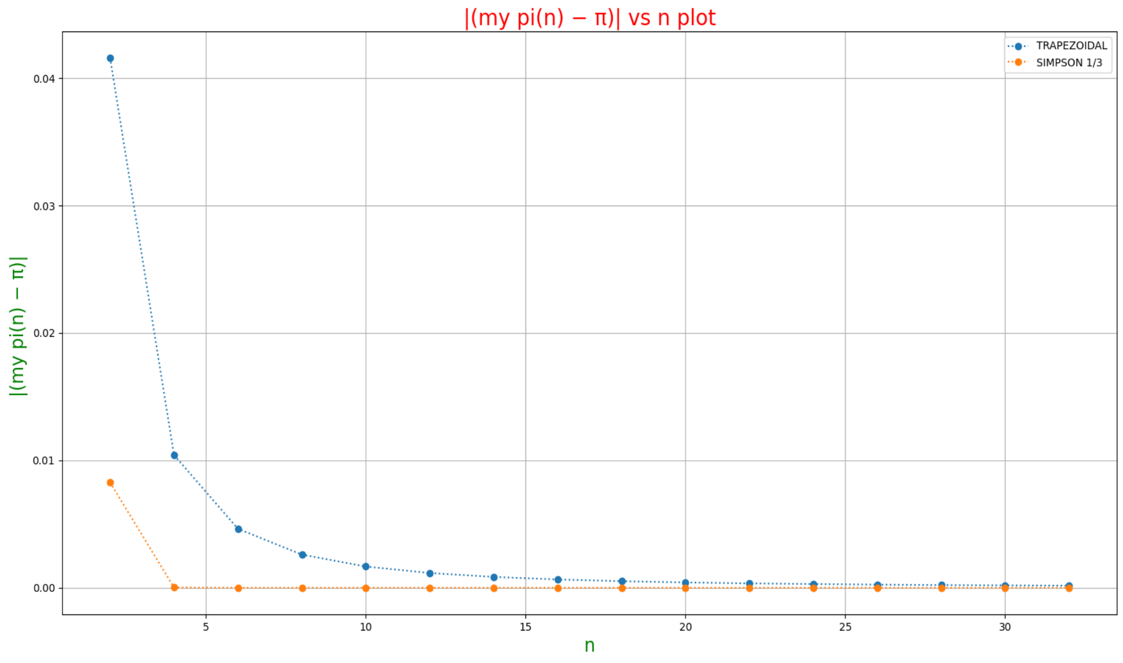


Figure 13: $|(mypi(n) - \pi)|$ *vs n plot for TRAPEZOIDAL METHOD and SIMPSON 1/3 METHOD*

From the above graph we can clearly see the absolute error is more in trapezoidal than simpson method. As n(number of terms) increases the absolute error lines start converging.

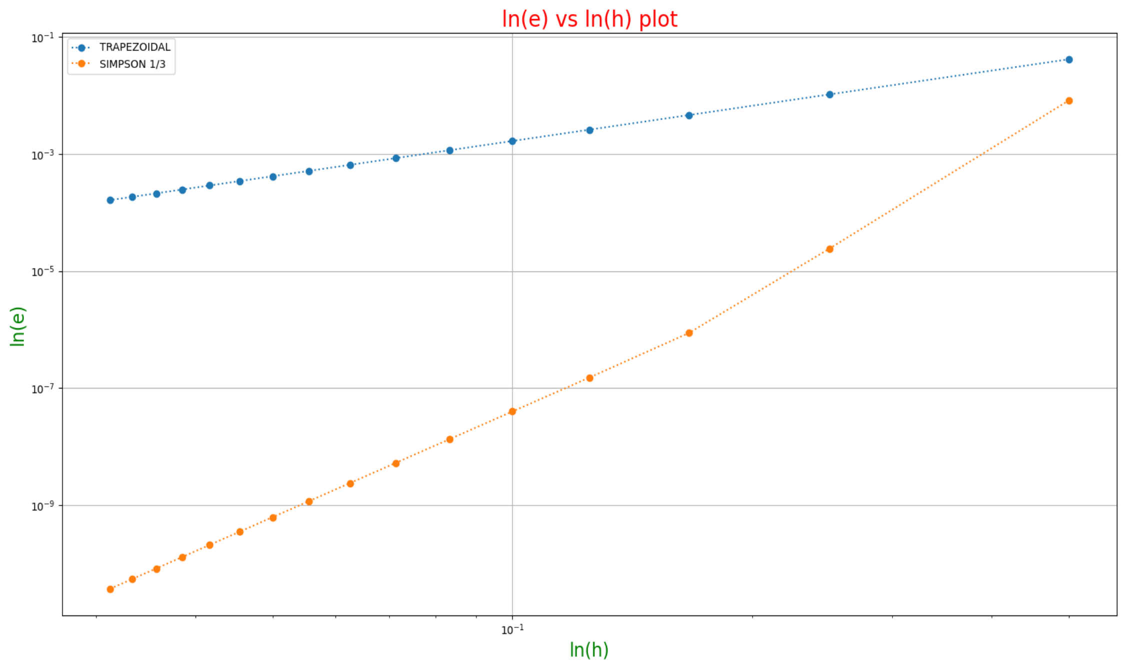
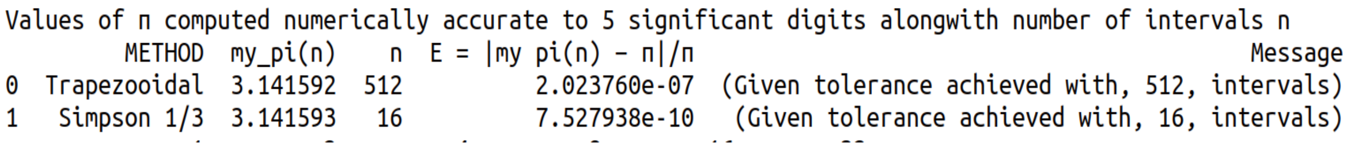


Figure 14: *ln(e) vs ln(h) plot for TRAPEZOIDAL METHOD and SIMPSON 1/3 METHOD*

From the above plot between ln(e) vs ln(h) we can interpret that as the step size shrinks the the integrated value obtained by simpson 1/3 and trapezoidal method is approaching towards the true value. While simpson method is more accurate than trapezoidal we can clearly see from the graph .

2.3 Q3 (d) : Estimation of value of π correct to 5 number of significant digits

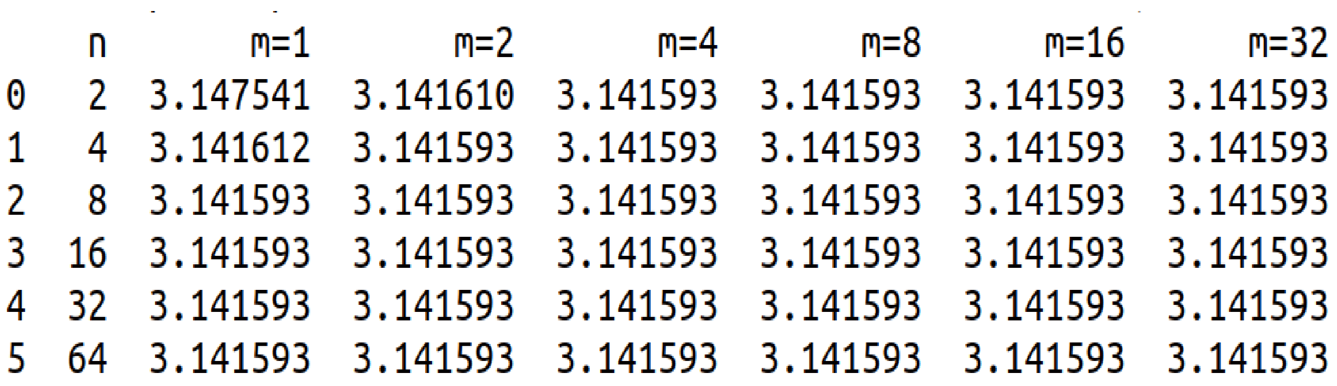


```
Values of π computed numerically accurate to 5 significant digits alongwith number of intervals n
         METHOD  my_pi(n)    n  E = |my pi(n) - π|/π                                     Message
0  Trapezooidal  3.141592  512           2.023760e-07  (Given tolerance achieved with, 512, intervals)
1    Simpson 1/3  3.141593   16           7.527938e-10   (Given tolerance achieved with, 16, intervals)
```

Figure 15: *Estimated value of π*

2.4 Q3 (e) : Legendre Gauss Quadrature

- **(i)**



```
    n       m=1       m=2       m=4       m=8      m=16      m=32
0   2  3.147541  3.141610  3.141593  3.141593  3.141593  3.141593
1   4  3.141612  3.141593  3.141593  3.141593  3.141593  3.141593
2   8  3.141593  3.141593  3.141593  3.141593  3.141593  3.141593
3  16  3.141593  3.141593  3.141593  3.141593  3.141593  3.141593
4  32  3.141593  3.141593  3.141593  3.141593  3.141593  3.141593
5  64  3.141593  3.141593  3.141593  3.141593  3.141593  3.141593
```

Figure 16: *Estimated value of π for different values of n(order of Gauss Legendre) and m (No. of sub intervals)*

- **(ii) Plot of value of pi-quad(n, m) obtained numerically as a function of - (a) n , (b) m**

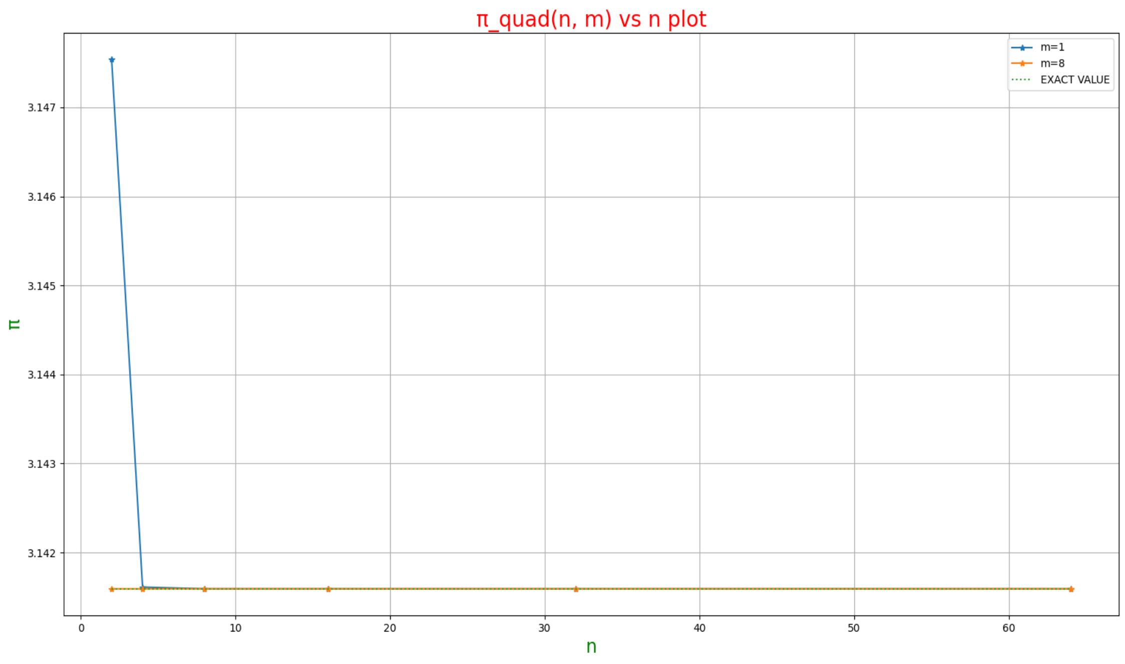


Figure 17: *lot of value of pi-quad(n, m) obtained numerically as a function of n*

From the above plot it can be understood that the error $|pi - quad(n,m) - \pi|$ as a function of n for m=1 and m=8, the absolute error is less for 8 sub intervals as compared to that of 1 sub intervals only. So, increasing subintervals leads to less error in integrated value.

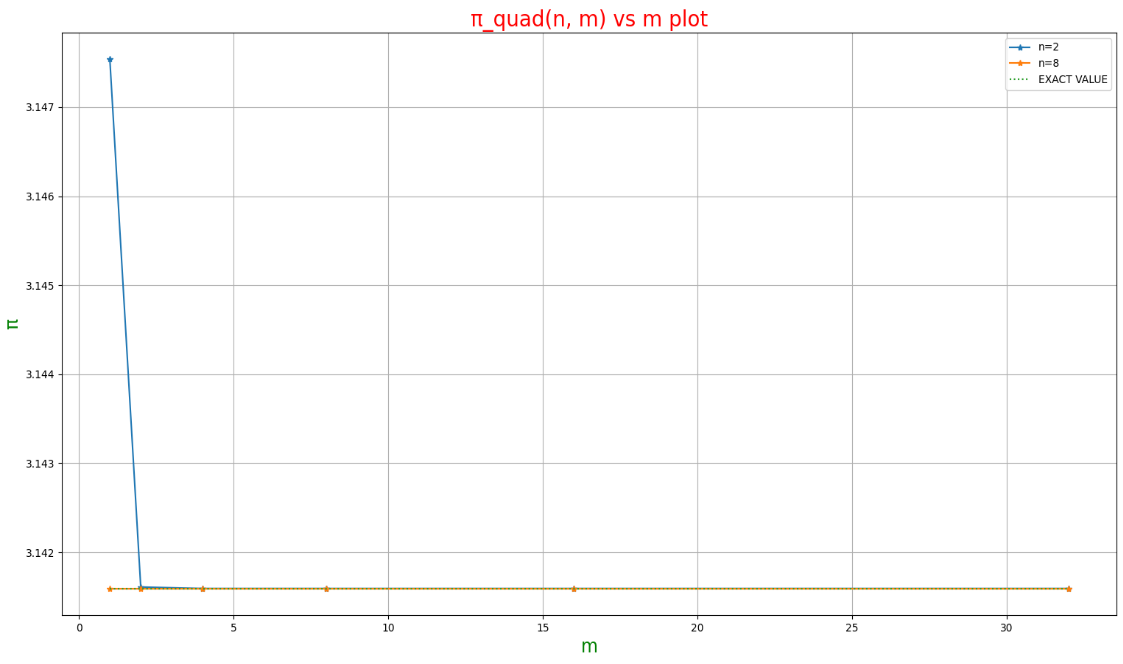


Figure 18: *Plot of value of pi-quad(n, m) obtained numerically as a function of m*

The above plot is between $|pi - quad(n,m) - \pi|$ vs m for n=2 and n=8, we can see that for m<2 (sub-intervals) minimum absolute error is there but as m increases the absolute error line for n=2 and n=8 start converging. So we can say that m>2 will give give us minimum absolute error.

- **(iii) Plot of error e(n, m) = $|(mypi(n) - \pi)|$ as a function of m and n each**

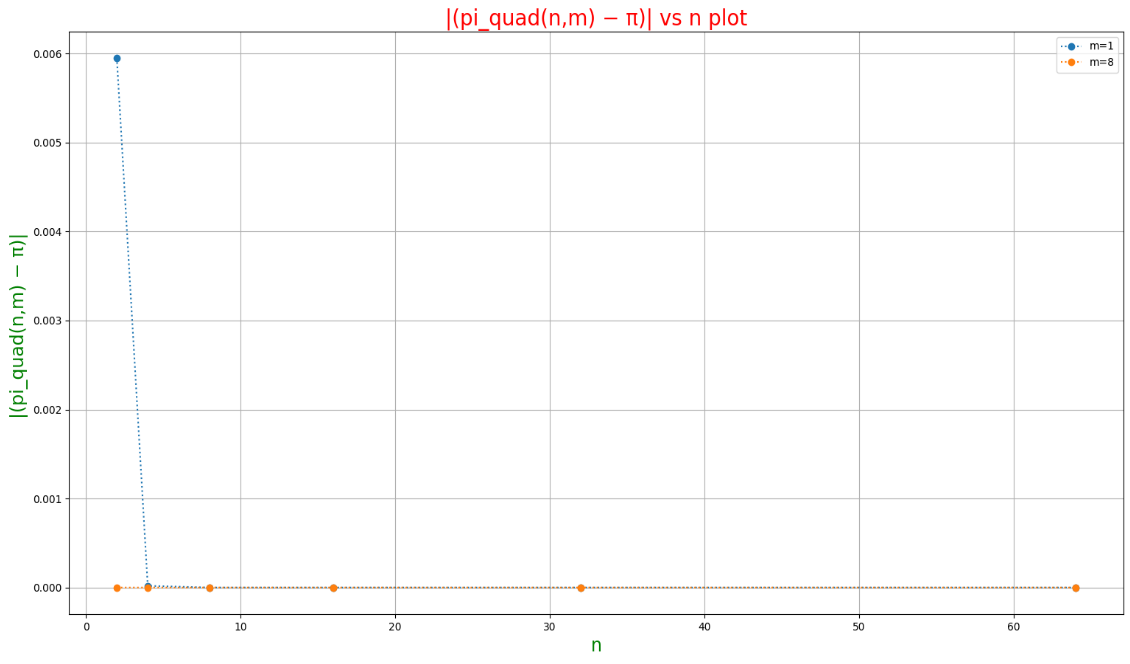


Figure 19: *Plot of error e(n, m) =* $|(mypi(n) - \pi)|$ *as a function of n*

Initial absolute error is less for m=8 as compared to that of m=1.As the value of n increses the error decreases.

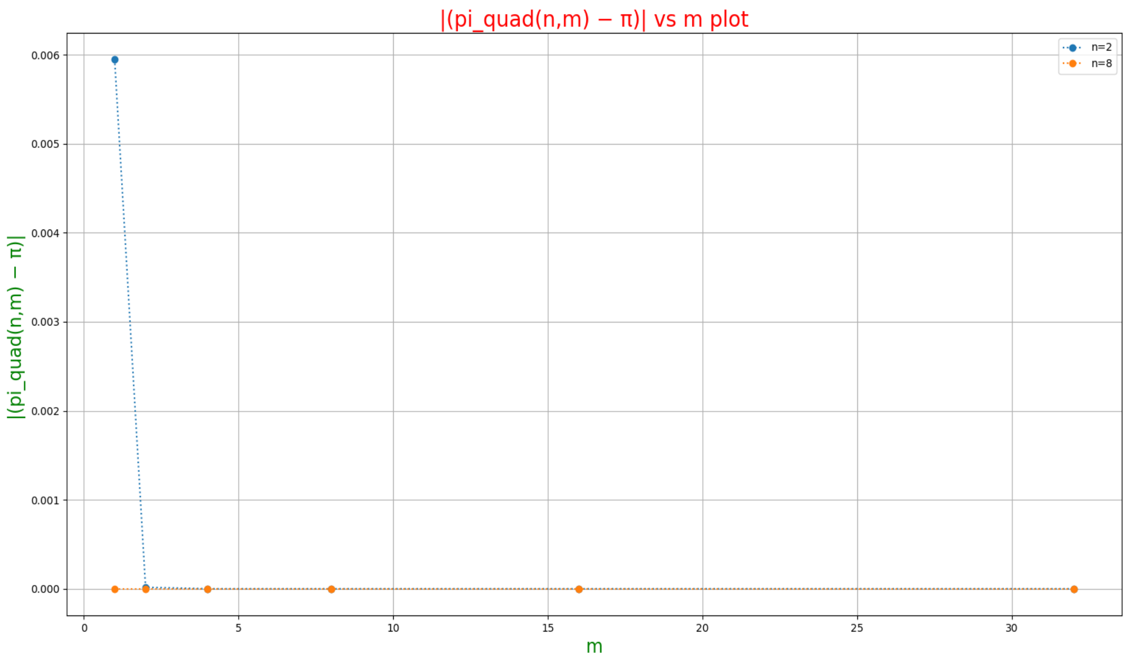
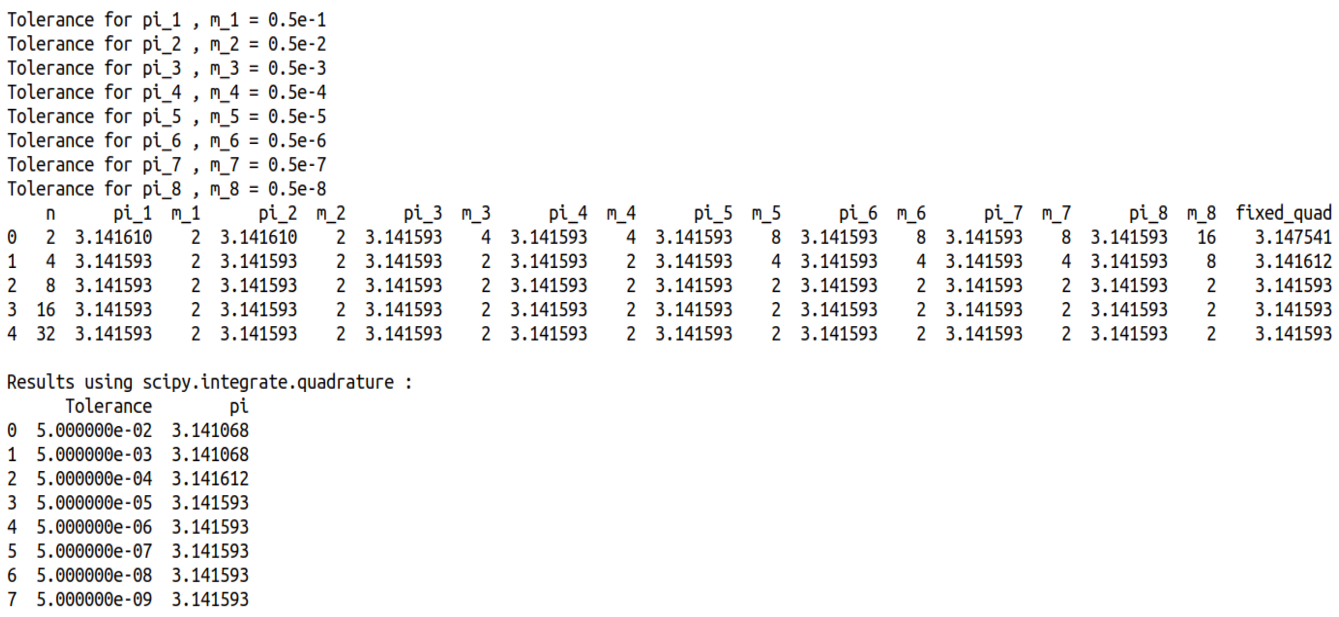


Figure 20: *Plot of error* $e(n,\, m) = |(mypi(n) - \pi)|$ *as a function of* m

2.5 Q3 (f,g) :Estimated value of π correct to certain number of significant digits using Legendre Gauss Quadrature



```
Tolerance for pi_1 , m_1 = 0.5e-1
Tolerance for pi_2 , m_2 = 0.5e-2
Tolerance for pi_3 , m_3 = 0.5e-3
Tolerance for pi_4 , m_4 = 0.5e-4
Tolerance for pi_5 , m_5 = 0.5e-5
Tolerance for pi_6 , m_6 = 0.5e-6
Tolerance for pi_7 , m_7 = 0.5e-7
Tolerance for pi_8 , m_8 = 0.5e-8
    n      pi_1  m_1      pi_2  m_2      pi_3  m_3      pi_4  m_4      pi_5  m_5      pi_6  m_6      pi_7  m_7      pi_8  m_8  fixed_quad
0   2  3.141610    2  3.141610    2  3.141593    4  3.141593    4  3.141593    8  3.141593    8  3.141593    8  3.141593   16    3.147541
1   4  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    4  3.141593    4  3.141593    4  3.141593    8    3.141612
2   8  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2    3.141593
3  16  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2    3.141593
4  32  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2  3.141593    2    3.141593

Results using scipy.integrate.quadrature :
      Tolerance        pi
0  5.000000e-02  3.141068
1  5.000000e-03  3.141068
2  5.000000e-04  3.141612
3  5.000000e-05  3.141593
4  5.000000e-06  3.141593
5  5.000000e-07  3.141593
6  5.000000e-08  3.141593
7  5.000000e-09  3.141593
```

Figure 21: *Estimated value of π upto given tolerance and the results using inbuilt functions*

A Programs

```python
import numpy as np

'''
My Function for integrating a function using trapezoidal method that works for both     fixed number of intervals and fixed tolerance.
Input Parameters -  function (f) , a - lower limit , b - upper limit , n0 - Number of panels , key1(Bool)= False(for finding Integral for only 1 value of n(intervals)) True(for finding integral for more than 1 value of n) , N_max=maximum value of n , key2(Bool)= True (for tolerance) , tol=tolerance
'''
def MyTrap(f,a,b,n0,key1=True,N_max=None,key2=False,tol=None):
    w=0
    h=(b-a)/n0          #step size
    S=0.5*(f(a)+f(b))
    for i in range(1,n0):
        S+= f(a+i*h)
    Integral = S * h
    if key1== True:
        pass
    else:
        return Integral      #returning the integral if key is set to false
    n_a=[n0]        #creating array for values of n(intervals)
    I=[Integral]        #creating array to store values of integral
    n0=2*n0          #doubling subintervals
    while n0<=N_max:
        h=(b-a)/n0
        r=0
        for i in range(1,n0):        #calculating the value of function at certain points to avoid repeated calculations
              if i%2 != 0:
                 r+=f(a+i*h)
        r=r*h
        I.append((I[-1]/2)+(r))
        n_a.append(n0)
        if key2==True:
            err=abs((I[-1]-I[-2])/I[-1])  #calculation of relative error
            if err<=tol:
                w=1
                break
            else:
                pass
        else:
            pass
        n0=2*n0
    if key2==True:    #printing the message if key2 is true
        if w==0:
            s=("N_max reached without achieving required tolerance")
        elif w==1:
             s="Given tolerance achieved with",n_a[-1],"intervals"
        return I[-1],n_a[-1],s        #returning integral,number of intervals and message
    else:
        return I[-1],n_a[-1]          #returning integral,number of intervals

'''
My Function for integrating a function using simpson 1/3 method that works for both     fixed number of intervals and fixed tolerance.
Input Parameters -  function (f) , a - lower limit , b - upper limit , n0 - Number of panels , key1(Bool)= False(for finding Integral for only 1 value of n(intervals)) True(for finding integral for more than 1 value of n) , N_max=maximum value of n , key2(Bool)= True (for tolerance) , tol=tolerance
'''

def MySimp(f,a,b,n0,key1=True,N_max=None,key2=False,tol=None):
    if n0%2 ==0:
        pass
    else :
        return "Number of intervals must be even"     #this works for even number of sub intervals
    w=0
    S_a=[];T_a=[];I_a=[]
    h=(b-a)/n0      #step size
    S = f(a) + f(b)
    T=0
    for i in range(1,n0):
        if i%2 == 0:
            S = S + 2 * f(a + i*h)
        else:
            T = T + (2 * f(a + i*h))/3
    S=S/3
    Integral =h*(S+2*T)
    if key1== True:
        pass
    else:
        return Integral   #returning the integral if key is set to false
    S_a.append(S);T_a.append(T);I_a.append(Integral);n_a=[n0]         #creating array for values of n(intervals),Integral
    n0=2*n0  #doubling subintervals
    while n0<=N_max:
        h=(b-a)/n0
        T=0
        for i in range(1,n0):      #calculating the value of function at certain points to avoid repeated calculations
            if i%2 != 0:
                T+=(2*f(a+i*h))/3

        S=S_a[-1]+T_a[-1]
        Integral =h*(S+2*T)
        S_a.append(S);T_a.append(T);I_a.append(Integral);n_a.append(n0)
        if key2==True:
            err=abs((I_a[-1]-I_a[-2])/I_a[-1])          #calculation of relative error
            if err<=tol:
                w=1
                break
            else:
                pass
        else:
            pass
        n0=2*n0
    if key2==True:      #printing the message if key2 is true
        if w==0:
            s=("N_max reached without achieving required tolerance")
        elif w==1:
             s="Given tolerance achieved with",n_a[-1],"intervals"
        return I_a[-1],n_a[-1],s     #returning integral,number of intervals and message
    else:
        return I_a[-1],n_a[-1]      #returning integral,number of intervals

'''
My Function for integrating a function using Legendre Gauss method that works for any order and any sub intervals and tolerance is optional parameter
Input Parameters -  function (f) , a - lower limit , b - upper limit ,  n - order of gauss legendre , m - Number of sub intervals , key(Bool)= False (for finding Integral for only a certain value of m and n ) True(for finding integral for more than 1 value of n,m upto  certain tolerance) , m_max=maximum value of m  , tol=tolerance
'''


def MyLegQuadrature(f,a,b,n,m,key=False,tol=None,m_max=None):
  def gs1(f,a,b,n,m0):           #subfunction which returns the value of integral for specific n and m
      x,w = np.polynomial.legendre.leggauss(n)
      h=(b-a)/m
      s=0
      for i in range(0,m):
          r=0
          for x1,w1 in zip(x,w):
               r+=w1*f((((a+(i+1)*h)-(a+i*h))/2)*x1+((a+i*h)+(a+(i+1)*h))/2)
          r= (((a+(i+1)*h)-(a+i*h)) /2 )*r
          s+=r
      return s
  Integral=gs1(f,a,b,n,m)
  if key==True:
     pass
  else:

     return Integral     #returning thr value of integral if key is false
  w=0
```

```python
  m_a=[m]
  I=[Integral]
  m=2*m
  while m<=m_max:
    I.append(gs1(f,a,b,n,m))
    m_a.append(m)

    err=abs((I[-1]-I[-2])/I[-1])
    if err<=tol:
        w=1
        break
    else:
        pass

    m=2*m
  if w==0:
      s=("m_max reached without achieving required tolerance")
  elif w==1:
      s="Given tolerance achieved with",m_a[-1],"sub-intervals"
  return [I[-1],m_a[-1],s]        #returns integral,number of
   subintervals and message
```

Source Code 1: Python Program

```python
#Name= Monu Chaurasiya
#College Roll No. = 2020 PHY1102
#University Roll No. = 20068567035

from MyIntegration import MySimp
from MyIntegration import MyTrap
from MyIntegration import MyLegQuadrature
import pandas as pd


# (i)
# The trapezoidal method

f_x=["1","x","x**2","x**3"]
Calc=[]
Exact=[1,1.5,2.33333333,3.75]
a=1
b=2
n=1

f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MyTrap(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MyTrap(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MyTrap(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MyTrap(f,a,b,n,False))
```

```python
data={"f(x)":f_x,"Calculated":Calc,"Exact":Exact}

print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
print("METHOD USED : TRAPEZOIDAL")
print("NO. OF INTERVALS TAKEN : 1")
print("Lower Limit(a) = 1    ,     Upper Limit(b) = 2 ")
print(pd.DataFrame(data))

print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()


# (ii)
# The simpson method
f_x=["1","x","x**2","x**3","x**4","x**5"]
Calc=[]
Exact=[1,1.5,2.33333333,3.75,6.2,10.5]


f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
a=1
b=2
n=2

Calc.append(MySimp(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MySimp(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MySimp(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MySimp(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MySimp(f,a,b,n,False))
f=eval("lambda x:"+input("Enter the value of the FUNCTION F(x): ")) #
    Defining the function to be used for evaluation
Calc.append(MySimp(f,a,b,n,False))
data={"f(x)":f_x,"Calculated":Calc,"Exact":Exact}
```

Source Code 2: Python Program

```python
from MyIntegration import MySimp
from MyIntegration import MyTrap
from MyIntegration import MyLegQuadrature
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy import integrate
```

```

#(c)
#(i)
#Trapezoidal,Simpson_1/3
f=lambda x : 1/(1+x**2)

n_a=np.arange(1,17)
n_a=2*n_a
a=0
b=1
I_n_tr=[]
my_pi_tr=[]
I_n_si=[]
my_pi_si=[]

for n in n_a:
    I_n_tr.append(MyTrap(f,a,b,n,False))
    my_pi_tr.append(4*MyTrap(f,a,b,n,False))
    I_n_si.append(MySimp(f,a,b,n,False))
    my_pi_si.append(4*MySimp(f,a,b,n,False))

print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()

print("Estimated value of pi using n intervals with TRAPEZOIDAL METHOD
    ")
data={"n (no. of intervals)":n_a,"my_pi":my_pi_tr}
print(pd.DataFrame(data))

print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()

print("Estimated value of pi using n intervals with SIMPSON 1/3 METHOD
    ")
data={"n (no. of intervals)":n_a,"my_pi":my_pi_si}
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
pi=np.array([math.pi]*len(n_a))

#(ii)
plt.plot(n_a,my_pi_tr,label="TRAPEZOIDAL",marker="*")
plt.plot(n_a,my_pi_si,label="SIMPSON 1/3",marker="*")
plt.plot(n_a,pi,label="EXACT VALUE",linestyle='dotted')
plt.xlabel("n",c="green",fontsize=17)
plt.ylabel("\u03C0",c="green",fontsize=17)
plt.title("\u03C0 vs n plot",c="red",fontsize=20)
plt.legend()
plt.grid()
plt.show()

#(iii)
```

```python
e_t=[];e_s=[]
for x,y in zip(my_pi_tr,my_pi_si):
    e_t.append(abs(x-math.pi))
    e_s.append(abs(y-math.pi))

print("error e(n) = |(my pi(n)       )| for n intervals with TRAPEZOIDAL METHOD")
data={"n (no. of intervals)":n_a,"e(n) = |(my pi(n)       )|":e_t}
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
print("error e(n) = |(my pi(n)       )| for n intervals with SIMPSON 1/3 METHOD")
data={"n (no. of intervals)":n_a,"e(n) = |(my pi(n)       )|":e_s}
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()

plt.plot(n_a,e_t,label="TRAPEZOIDAL",marker="o",linestyle='dotted')
plt.plot(n_a,e_s,label="SIMPSON 1/3",marker="o",linestyle='dotted')
plt.xlabel("n",c="green",fontsize=17)
plt.ylabel("|(my pi(n)     \u03C0)| ",c="green",fontsize=17)
plt.title("|(my pi(n)     \u03C0)| vs n plot",c="red",fontsize=20)
plt.legend()
plt.grid()
plt.show()

h_a=[]
for n in n_a:
    h_a.append((b-a)/n)

plt.plot(h_a,e_t,label="TRAPEZOIDAL",marker="o",linestyle='dotted')
plt.plot(h_a,e_s,label="SIMPSON 1/3",marker="o",linestyle='dotted')
plt.xlabel("ln(h)",c="green",fontsize=17)
plt.ylabel("ln(e)",c="green",fontsize=17)
plt.title("ln(e) vs ln(h) plot",c="red",fontsize=20)
plt.legend()
plt.grid()
plt.xscale("log")
plt.yscale("log")
plt.show()



# f,a,b,n0,key1=True,N_max=None,key2=False,tol=None

#(d)
met=["Trapezooidal","Simpson 1/3"]
v1=MyTrap(f,a,b,1,key1=True,N_max=10000,key2=True,tol=0.1e-5)
v2=MySimp(f,a,b,2,key1=True,N_max=10000,key2=True,tol=0.1e-5)
pi_array = [4*v1[0],4*v2[0]]
n_a=[v1[1],v2[1]]
m_a=[v1[2],v2[2]]
```

```

e_a=[]
for x in pi_array:
    e_a.append(abs(x-math.pi)/math.pi)

data={"METHOD":met,"my_pi(n)":pi_array,"n":n_a,"E = |my pi(n)     \
   u03C0|/\u03C0":e_a,"Message":m_a}
print("Values of \u03C0 computed numerically accurate to 5 significant
     digits alongwith number of intervals n")
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
#(e)
f=lambda x : 1/(1+x**2)
a=0
b=1

#(i)
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(4*MyLegQuadrature(f,a,b,n,m,key=False,tol=None,m_max=
    None))
    g.append(s)
d=np.array(g).reshape(6,6)

np.savetxt('pi quad-1102a.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(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
#(ii)
plt.plot(n_a,g[0],label="m=1",marker="*")
plt.plot(n_a,g[3],label="m=8",marker="*")
plt.plot(n_a,pi,label="EXACT VALUE",linestyle='dotted')
plt.xlabel("n",c="green",fontsize=17)
plt.ylabel("\u03C0",c="green",fontsize=17)
plt.title("\u03C0_quad(n, m) vs n plot",c="red",fontsize=20)
plt.legend()
plt.grid()
plt.show()


y1=[];y2=[]
for i in range(6):
    y1.append(d[i][0])
    y2.append(d[i][2])
```

```python


plt.plot(m_a,y1,label="n=2",marker="*")
plt.plot(m_a,y2,label="n=8",marker="*")
plt.plot(m_a,pi,label="EXACT VALUE",linestyle='dotted')
plt.xlabel("m",c="green",fontsize=17)
plt.ylabel("\u03C0",c="green",fontsize=17)
plt.title("\u03C0_quad(n, m) vs m plot",c="red",fontsize=20)
plt.legend()
plt.grid()
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.plot(n_a,e1,label="m=1",marker="o",linestyle='dotted')
plt.plot(n_a,e2,label="m=8",marker="o",linestyle='dotted')
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()
plt.show()

plt.plot(m_a,e3,label="n=2",marker="o",linestyle='dotted')
plt.plot(m_a,e4,label="n=8",marker="o",linestyle='dotted')
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()
plt.show()

#(f)
f=lambda x : 1/(1+x**2)
a=0
b=1

n_a=[2, 4,8,16,32]


S1=[];S2=[];S3=[];S4=[];S5=[];S6=[];S7=[];S8=[]
s1=[];s2=[];s3=[];s4=[];s5=[];s6=[];s7=[];s8=[]
for n in n_a:
    o1=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-1,m_max=1000)
    o2=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-2,m_max=1000)
    o3=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-3,m_max=1000)
    o4=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-4,m_max=1000)
    o5=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-5,m_max=1000)
    o6=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-6,m_max=1000)
    o7=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-7,m_max=1000)
    o8=MyLegQuadrature(f,a,b,n,1,key=True,tol=0.5e-8,m_max=1000)
    s8.append(o8[1]);s7.append(o7[1])
    s6.append(o6[1]);s5.append(o5[1])
    s4.append(o4[1]);s3.append(o3[1])
    s2.append(o2[1]);s1.append(o1[1])
    S8.append(4*o8[0]);S7.append(4*o7[0])
    S6.append(4*o6[0]);S5.append(4*o5[0])
    S4.append(4*o4[0]);S3.append(4*o3[0])
    S2.append(4*o2[0]);S1.append(4*o1[0])


q1=[]
for n in n_a:
    q1.append(4*integrate.fixed_quad(f,a,b,n=n)[0])

print("Tolerance for pi_1 , m_1 = 0.5e-1")
print("Tolerance for pi_2 , m_2 = 0.5e-2")
print("Tolerance for pi_3 , m_3 = 0.5e-3")
print("Tolerance for pi_4 , m_4 = 0.5e-4")
print("Tolerance for pi_5 , m_5 = 0.5e-5")
print("Tolerance for pi_6 , m_6 = 0.5e-6")
print("Tolerance for pi_7 , m_7 = 0.5e-7")
print("Tolerance for pi_8 , m_8 = 0.5e-8")
data={"n":n_a,"pi_1":S1,"m_1":s1,"pi_2":S2,"m_2":s2,"pi_3":S3,"m_3":s3,"pi_4":S4,"m_4":s4,"pi_5":S5,"m_5":s5,"pi_6":S6,"m_6":s6,"pi_7":S7,"m_7":s7,"pi_8":S8,"m_8":s8,"fixed_quad":q1}
print(pd.DataFrame(data))

#(g)
to=[0.5e-1,0.5e-2,0.5e-3,0.5e-4,0.5e-5,0.5e-6,0.5e-7,0.5e-8]
q2=[]
for tol in to:
    q2.append(4*integrate.quadrature(f, a, b,rtol=tol,maxiter=64, vec_func=True, miniter=2)[0])

print()
print("Results using scipy.integrate.quadrature :")
data={"Tolerance":to,"pi":q2}
print(pd.DataFrame(data))
print()
print("*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*")
print()
```

Source Code 3: Python Program

B Contribution of team mates