INTEGRATION MODULE

SGTB Khalsa College, University of Delhi

Preetpal Singh(2020PHY1140)(20068567043)

Jagjyot Singh(2020PHY1005)(20068567028)

Unique Paper Code: 32221401

Paper Title: Mathematical Physics III

Submitted on: February 8, 2022

B.Sc(H) Physics Sem IV

Submitted to: Dr. Mamta

1 Theory

1.1 Newton Cotes Quadrature

- 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.

1.2 Newton Cotes Quadrature

In numerical analysis, the Newton-Cotes formulas, also called the Newton-Cotes quadrature rules are a group of formulas for numerical integration based on evaluating the integrand at equally spaced points. Newton-Cotes formulas can be useful if the value of the integrand at equally spaced points is given. It is assumed that the value of a function f defined on $[a, b]$ is known at $\boldsymbol{n+1}$ equally spaced points: $a \leq x_0 < x_1 < \cdots < x_n \leq b$.
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. NewtonCotes formulas using $n+1$ points is defined as [1]

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} w_i f(x_i)$$

here,

1. for a closed formula, $x_i = a + ih$, with $h = \frac{b-a}{n}$,
2. for an open formula, $x_i = a + (i+1)h$, with $h = \frac{b-a}{n+2}$. The number h is called step size, w_i are called weights.

2 Method of undetermined coefficients to derive the Trapezoidal, Simpson $_{1/3}$ and Simpson $_{3/8}$ rules for integration.

Let's take

$$\int_a^b f(x)dx = c_1 f(a) + c_2 f(b)$$

Integral of a function is given by multiplying value of function $f(x)$ at point a *i.e.* $f(a)$ by some constant value c_1 and multiplying value of function $f(x)$ at point b *i.e.* $f(b)$ by some constant value c_2 and adding them together.

Set $f(x) = a_0 + a_1 x$ by the formula

$$\int_a^b (a_0 + a_1 x)\, dx = \left[a_0 x + a_1 \frac{x^2}{2}\right]_a^b \Rightarrow \left[\left(a_0 b + \frac{a_1 b^2}{2}\right) - \left(a_0 a + \frac{a_1 a^2}{2}\right)\right]$$

$$= \left[a_0(b-a) + \frac{a_1}{2}\left(b^2 - a^2\right)\right] \tag{1}$$

If we take above formula,

$$\int_a^b f(x)dx \simeq c_1 f(a) + c_2 f(b) \simeq c_1 \left[a_0 + a_1 a\right] + c_2 \left[a_0 + a_1 b\right] \tag{2}$$

Equating (1) and (2)

$$
\begin{aligned}
a_0(b-a)+\frac{a_1}{2}\left(b^2-a^2\right) &\simeq c_1\left[a_0+a_1 a\right]+c_2\left[a_0+a_1 b\right] \\
&\simeq a_0\left[c_1+c_2\right]+a_1\left[c_1 a+c_2 b\right]
\end{aligned}
$$

The given equation only possible when

$$b-a=c_1+c_2 \tag{3}$$

$$\frac{b^2-a^2}{2}=c_1 a+c_2 b \tag{4}$$

multiply equation (3) by (a)

$$a(b-a)=a\left(c_1+c_2\right)$$

$$ab-a^2=ac_1+ac_2 \tag{5}$$

$$(4)-(5)$$

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

$$
\begin{aligned}
\frac{b^2}{2}+\frac{a^2}{2}-ab &= c_2 b-c_2 a \\
\frac{b^2+a^2-2ab}{2} &= c_2(b-a) \\
\frac{(b-a)^2}{2} &= c_2(b-a) \\
c_2 &= \frac{b-a}{2}
\end{aligned}
$$

putting value of c_2 in (3)

$$
\begin{aligned}
b-a &= c_1+\frac{b-a}{2} \\
c_1 &= \frac{b-a}{2}
\end{aligned}
$$

putting values of c_1 & c_2 in eqn(2) gives

$$\int_a^b f(x)dx \simeq c_1 f(a)+c_2 f(b)=\left(\frac{b-a}{2}\right) f(a)+\left(\frac{b-a}{2}\right) f(b) \Rightarrow \left(\frac{b-a}{2}\right)(f(a)+f(b))$$

Method of Undetermined Coefficients to derive Simpson Formula

$$\int_a^b f(x)dx \simeq \omega_0 f\left(x_0\right)+\omega_1 f\left(x_1\right)+\omega_2 f\left(x_2\right)$$

Now, if we fix the function arguments as equispaced points x_0, x_1, x_2 as

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

Using method of undetermined coefficients to calculate unknown weights w_0, w_1, w_2 and basis for given polynomial $P_2(x)$ is $1, x, x^2$ Because of linearity of integral in order to Show that formula is exact for all polymomials of degree ≤ 2, we've to show its exact for all basis of polynomial of degree 2

Let's take

$$\begin{aligned} f(x) &= a_0+a_1 x+a_2 x^2 \\ \int_a^b f(x)dx &= \int_a^b\left(a_0+a_1 x+a_2 x^2\right) dx \\ &= \left|a_0 x+a_1 \frac{x^2}{2}+a_2 \frac{x^3}{3}\right|_a b \\ &= a_0(b-a)+\frac{a_1}{2}\left(b^2-a^2\right)+\frac{a_2}{3}\left(b^3-a^3\right) \end{aligned}$$

Now, at

$$f(a)=a_0+a_1 a+a_2 a^2$$
$$f(b)=a_0+a_1 b+a_2 b^2$$

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

$$\therefore \quad \int_a^b f(x)dx=\omega_0 f(a)+\omega_1 f\left(\tfrac{a+b}{2}\right)+w_2 f(b)$$

$$=\omega_0\left[a_0+a_1 a+a_2 a^2\right]+\omega_1\left[a_0+a_1\left(\frac{a+b}{2}\right)+a_2\left(\frac{a+b}{2}\right)^2\right]+\omega_2\left[a_0+a_1 b+a_2 b^2\right]$$

$$=\left(w_0+w_1+w_2\right) a_0+a_1\left(w_0 a+w_1\left(\frac{a+b}{2}\right)+b\right)+a_2\left(w_0 a^2+w_1\left(\frac{a+b}{2}\right)^2+w_2 b^2\right)$$

By comparing above equations,

$$\begin{aligned} w_0+w_1+w_2 &= b-a \\ w_0 a+w_1\left(\frac{a+b}{2}\right)+b &= \frac{b^2-a^2}{2} \\ w_0 a^2+w_1\left(\frac{a+b}{2}\right)^2+w_2 b^2 &= \frac{b^3-a^3}{3} \end{aligned}$$

Solving these equations simultaneousely

$$\omega_0=\omega_2=\frac{b-a}{6}$$

$$\omega_1=\frac{2}{3}(b-a)$$

$$\begin{aligned} \int_a^b f(x)dx &\simeq \tfrac{b-a}{6} f(a)+\tfrac{2(b-a)}{3} f\left(\tfrac{a+b}{2}\right)+\tfrac{b-a}{6} f(b) \\ &\simeq \frac{b-a}{6}\left[f(a)+4 f\left(\frac{a+b}{2}\right)+f(b)\right] \end{aligned}$$

3 Geometrical interpretation and conditions on number of intervals for each of them

As we know the trapezoidal rule estimates the area the integral represents by replacing the curve with a collection of trapeziums. As the number of trapeziums increases then the accuracy of the approximation should increase as integration represents the area under the curve .

As no. of intervals increase more area would be covered under the curve and hence better ap-

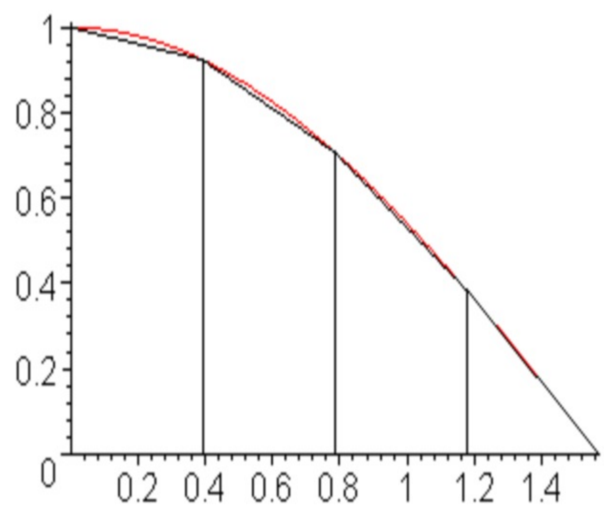


Figure 1: Trapezoidal method for smaller interval

proximation of the integral as shown in figure**??** .Here in the first plot we can see as the no. of intervals increase the area under the curve also increase and hence accuracy increases.In same way we can explain for Simpson rule.

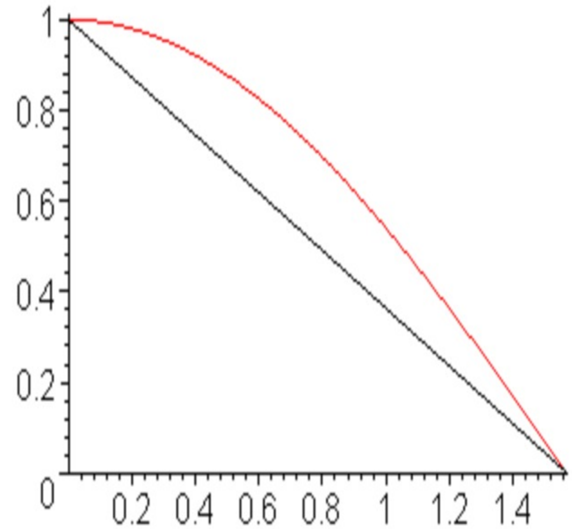


Figure 2: The practical circuit of unknown coil

3.1 Simpson's 1/3 rule

In Simpson's 1/3 rule, the curve $y = f(x)$ is replaced by the second degree parabola passing through the points $A(x_0, y_0), B(x_1, y_1)$ and $C(x_2, y_2)$. Therefore, the area bounded by the curve $y = f(x)$, the ordinates $x = x_0, x = x_2$ and the x-axis is approximated to the area bounded by the parabola ABC, the straight lines $x = x_0, x = x_2$ and x-axis, i.e., the area of the shaded region ABCDEA.

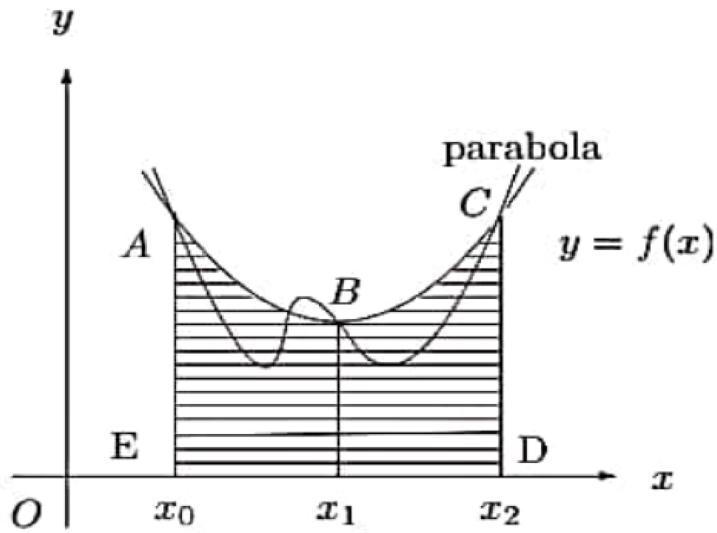


Figure 3: Geometrical interpretation of Simpson's 1/3 rule.

3.2 Simpson's 3/8 rule

Simpson's 3/8 rule is similar to Simpson's 1/3 rule, the only difference being that, for the 3/8 rule, the interpolant is a cubic polynomial. Though the 3/8 rule uses one more function value, it is about twice as accurate as the 1/3 rule.

3.3 Error term

3.3.1 In Trapezoidal Method

Let $f(x)$ have 2 continuous derivatives on the interval $a \leq x \leq b$ .Then error term is given by

$$E_n^T(f) = -\frac{h^2\,(b-a)}{12} f''(c_n)$$

for some c_n in the interval [a,b]. Trapezoidal Method give accurate results only upto the order of h^2. So, Even if we keep increasing number of intervals the results will not improve above h^2 order. However, if we take very small intervals then the machine will require high computational power but the results might not be as accurate as expected but the better methods are available which can give better accuracy with less intervals.

3.3.2 In Simpson 1/3 Method

Let $f(x)$ have 2 continuous derivatives on the interval $a \leq x \leq b$ .Then error term is given by

$$E_n^S(f) = -\frac{h^4\,(b-a)}{180} f^{(4)}(c_n)$$

for some c_n in the interval [a,b]. Simpson Method interpolate through parabola and it approximate better than Trapezoidal method. it's gives accurate results only upto the order of h^4. So, Even if we keep increasing number of intervals the results will not improve above h^4 order and it requires high computational power and results might not be as accurate as expected.

3.3.3 In Simpson3/8 Method

Let $f(x)$ have 2 continuous derivatives on the interval $a \le x \le b$ .Then error term is given by

$$E_n^S(f) = -\frac{h^4(b-a)}{80} f^{(4)}(c_n)$$

for some c_n in the interval [a,b]. Simpson 3/8 Method interpolate through cubic polynomial and it approximate better than Trapezoidal and Sympson Method. it's gives accurate results only upto the order of h^4. So, Even if we keep increasing number of intervals the results will not improve above h^4 order and it requires high computational power and results might not be as accurate as expected.

4 Legendre Gauss 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 and corresponding weights to approximate the integral by a weighted sum.

$$I = \int_{-1}^{1} f(x)dx = \sum_{i=0}^{n-1} w_i f(x_i)$$

Comparison of Newton Cotes and Gauss Quadrature Method

Gaussian quadrature is more accurate than the Newton-Cotes quadrature in the following sense:

- In Newton Cotes Methods abscissas are fixed at equally spaced points and weights are computed by approximating the integrands by a polynomial of particular order whereas in Gauss Quadrature method abscissas and weights are computed to give maximum possible accuracy.
- Both Gaussian quadrature and Newton-Cotes quadrature use the similar idea to do the approximation, i.e. they both use the Lagrange interpolation polynomial to approximate the integrand function and integrate the Lagrange interpolation polynomial to approximate the given definite integral.
- When the same number of nodes is used, the algebraic degree of precision of the Gaussian quadrature is higher than that of the Newton-Cotes quadrature.

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 $p(x)$ becomes equal to integral of remainder $r(x)$.

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

Legendre Gauss Quadrature methods for evaluation of integral $\int_{-1}^{1} f(x)\mathrm{d}x$. Transform the formula for the integration $\int_{a}^{b} f(x)\mathrm{d}x$

For N point method,

$$I_n = \sum_{i=1}^{N} w_i f\left(x_i\right)$$

$$I_n = \int_{-1}^{1} f(x)dx = \omega_1 f\left(x_1\right) + \omega_2 f\left(x_2\right) + \cdots + \omega_n f\left(x_n\right)$$

We determine weights and x_i using basis $\left\{1, x, x^2, x^3\right\}$. This method provides accurate results for a polynomial of degree $2n-1$ or less.

If the limits of integration are $x = a$ and $x = b$, it is possible to use a simple linear transformation to bring the limits to the standard $[-1, 1]$. The transformation would be

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

We also have $dx = \frac{b-a}{2}dt$ and hence the required integral is given by

$$I = \int_{a}^{b} f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right)dt$$

With the above proviso we now consider integration of a function over the interval $[-1, 1]$. All the evaluation points are with in the interval i.e. $-1 < x_i < 1$.

2-point Formula

As we know for N point method,

$$I_n = \sum_{i=1}^{N} w_i f(x_i)$$

$$I_n = \int_{-1}^{1} f(x)dx = \omega_1 f(x_1) + \omega_2 f(x_2) + \cdots + \omega_n f(x_n)$$

For 2 point formula,

$$I(f) = \int_{-1}^{1} f(x)dx \simeq w_0 f(x_0) + w_1 f(x_i) = I_2(f)$$

Determining $x_0, x_1, \omega_0, \omega_1$, so that

$$I_2(f) = I(f)$$

for a polynonial if degree less than or equal to 3

$$f(x) = 1, x, x^2, x^3$$
$$I_2(f) = I(f)$$

$$\int_a^b f(x)dx = \int_{-1}^{1} p(t)dt$$
$$x = \left(\frac{b-a}{2}\right)t + \left(\frac{b+a}{2}\right)$$
$$p(t) = f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right)$$
$$p(t) = 1 \quad \int_{-1}^{1} 1 \cdot dt = w_1 p(t_1) + w_2 p(t_2) = w_1 + w_2$$
$$\Rightarrow w_1 + w_2 = 2$$
$$p(t) = t \quad w_1 t_1 + w_2 t_2 = 0$$
$$p(t) = t^2 \quad w_1 t_1^2 + w_2 t_2^2 = \frac{2}{3}$$
$$p(t) = t^3 \quad w_1 t_1^3 + w_2 t_2^3 = 0$$
$$t_1 = -\sqrt{\frac{1}{3}}, \quad t_2 = +\sqrt{\frac{1}{3}}$$
$$w_1 = w_2 = 1$$

Zeros of Legendre polynomial of order 2

$$\int_{-1}^{1} f(t)dt \simeq f\left(-\sqrt{\frac{1}{3}}\right) + f\left(\sqrt{\frac{1}{3}}\right)$$

N-point composite quadrature formula

$$\int a^b f(x)dx = \sum_{j=1}^{m} \int_{xj-1}^{x_j} f(x)dx$$

and

$$\int xj-1^{x_j} f(x)dx = (\frac{x_j - x_{j-1}}{2} \sum k = 0^{n-1} wkf[\frac{(x_j - x_{j-1})t_k}{2} + \frac{(x_j + x_{j-1})}{2}]$$

$$= \frac{h}{2} \sum k = 0^{n-1} w_k f[\frac{ht_k}{2} + \frac{(x_j + xj - 1)}{2}]$$

$$\int a^b f(x)dx = \frac{h}{2} \sum_{j=1}^{m} \sum_{k=0}^{n-1} wkf[\frac{ht_k}{2} + \frac{x_j + x_{j-1}}{2}]$$

Algorithm

Algorithm 1 $MyTrap$

function INPUT(f, $initial\ conditions$, $number\ of\ intervals$) ▷ Here f is function to be integrated

$h = \frac{b-a}{n}$ ▷ Determining step size
for i in range(1, 2 $\times n$) **do** y = f(a+i h) ▷ y at limit points

for j in range(1, len(y)-1 **do**
trp = $h \times (f(a) + f(b))/2$ ▷ calculating integral
end for
return trp =0

Algorithm 2 $MySimp$

function INPUT(f, $initial$,$conditions\ number\ of\ intervals$) Here f is function to be integrated

$h = \frac{b-a}{2\times n}$ ▷ Determining step size
simp = $\frac{h\times(f(a)+f(b))}{3}$
for i in range(1, 2 $\times n$) **do** if i$\overline{\scriptstyle 2=0}$
simp$+ = \frac{2\times h\times f(a+ih)}{3}$
elif $\frac{i}{2}! = 0$
simp = $\frac{4\times h\times f(a+ih)}{3}$ ▷ calculating integral

return trp

Algorithm 3 $QuadratureTol$

function INPUT(f, $initial$,$conditions\ number\ of\ intervals$,$tolerance$) ▷ Here f is function to be integrated

i = 1 ▷ initialise i to 1
$while\ i <= n$: ▷ Calculating Relative Error
$e = \left|\frac{f_n - f_{n-1}}{f_n}\right|$ ▷ Conditions for determining intervals
if $e > 0.5 \times 10^{-tol}$
i = 2 $\times i$ $elif$ e¡=0.5 $\times 10^{-tol}$
break
▷ Break the function when tolerance is reached **return** f

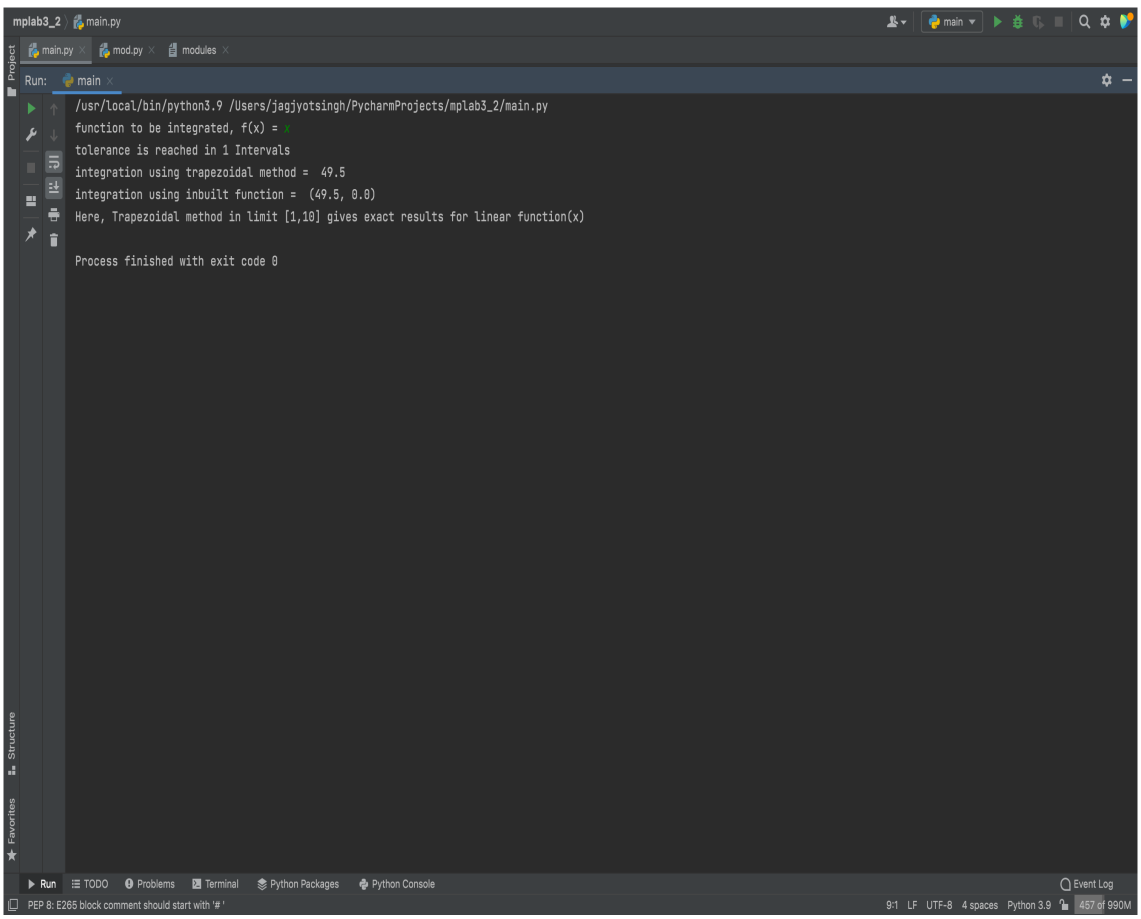
Algorithm 4 n-point Gauss Quadrature method

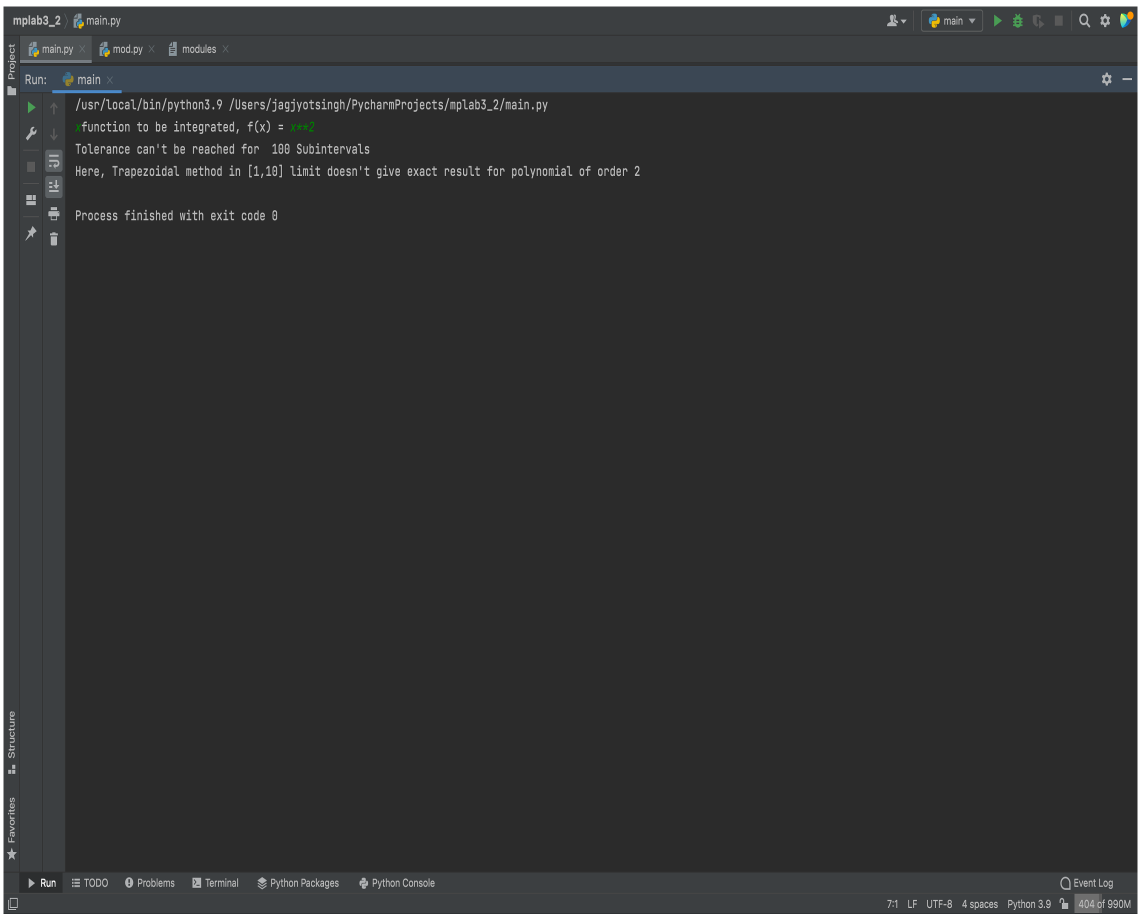
$h = (b - a)/m$ ▷ Determining step size using limits of integration and total number of subintervals
legendre zeroes, w = p roots for n point ▷ Extracting legendre polynomial roots corresponding weights ▷ method using module p roots from library scipy
$sum, x = 0, [a]$ ▷ initializing integration and interval
for i in range(0, n) **do**
    for j in range(1, m+1 **do**
        $x = a + hi$ ▷ incrementing intervals
        $sum+ = (h/2)wf(0.5 * h * legzer + 0.5 * (x[i] + x[i - 1]))$ ▷ n point composite formula
    end for
end for
    return sum =0

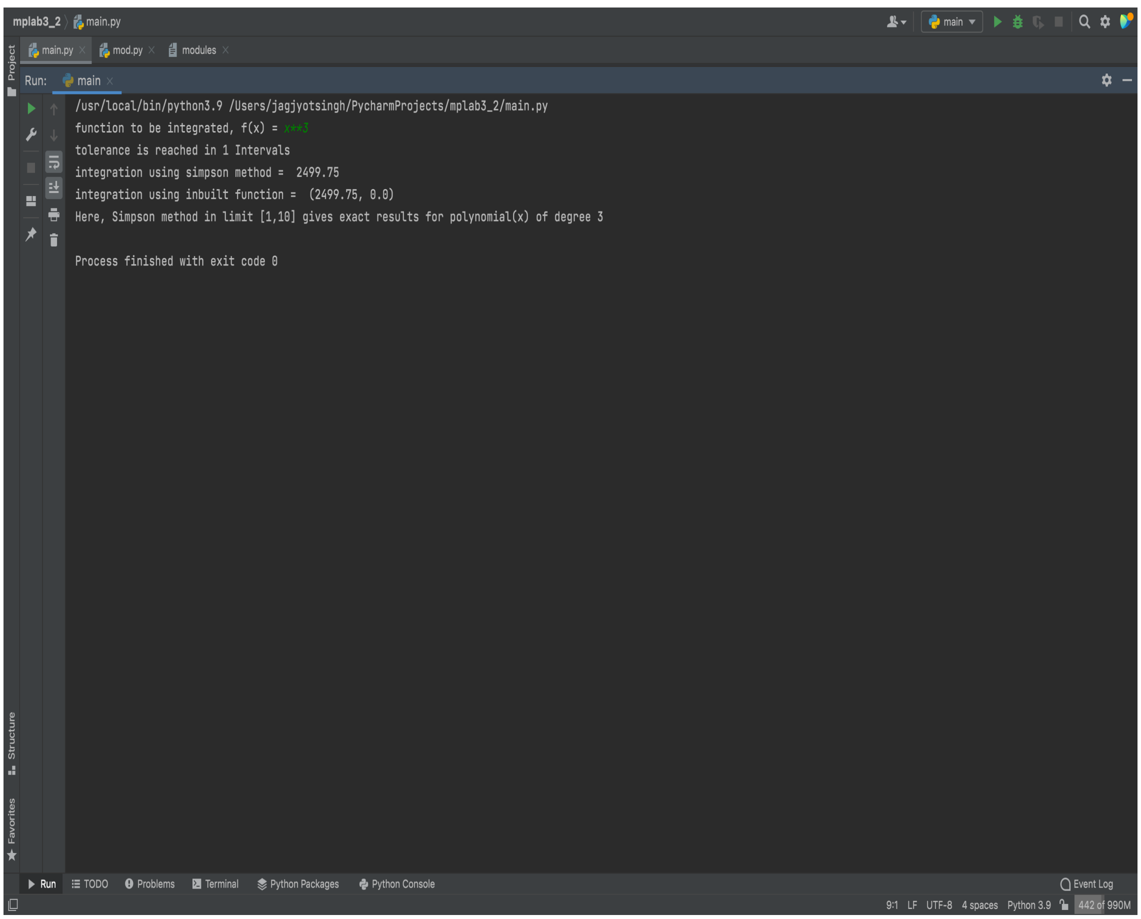
Algorithm 5 n-point Gauss Quadrature method tolerance

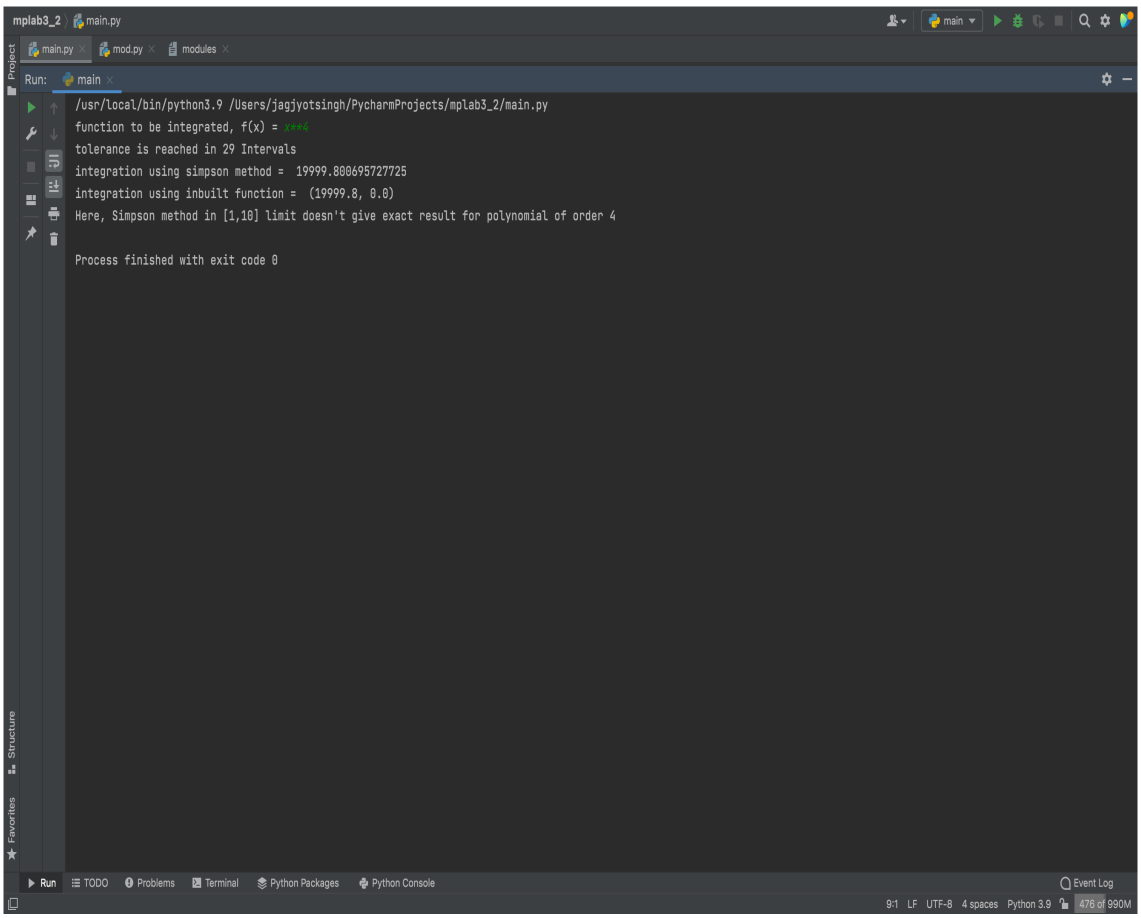
$i = 1$ ▷ initializing intervals
While true ▷ to run the code till tolerance is satisfied
$error = |gaussquad(i) - gaussquad(i - 1)|/gaussquad(i)$ ▷ calculating relative error
if error > tolerance ▷ loop will increment value of i in mutiples of 2
i = i*2
elif error ≤ tolerance ▷ when tolerance is reached loop will break
tolerance is reached in i subintervals
integration using n point method(composite) = gaussquad
break

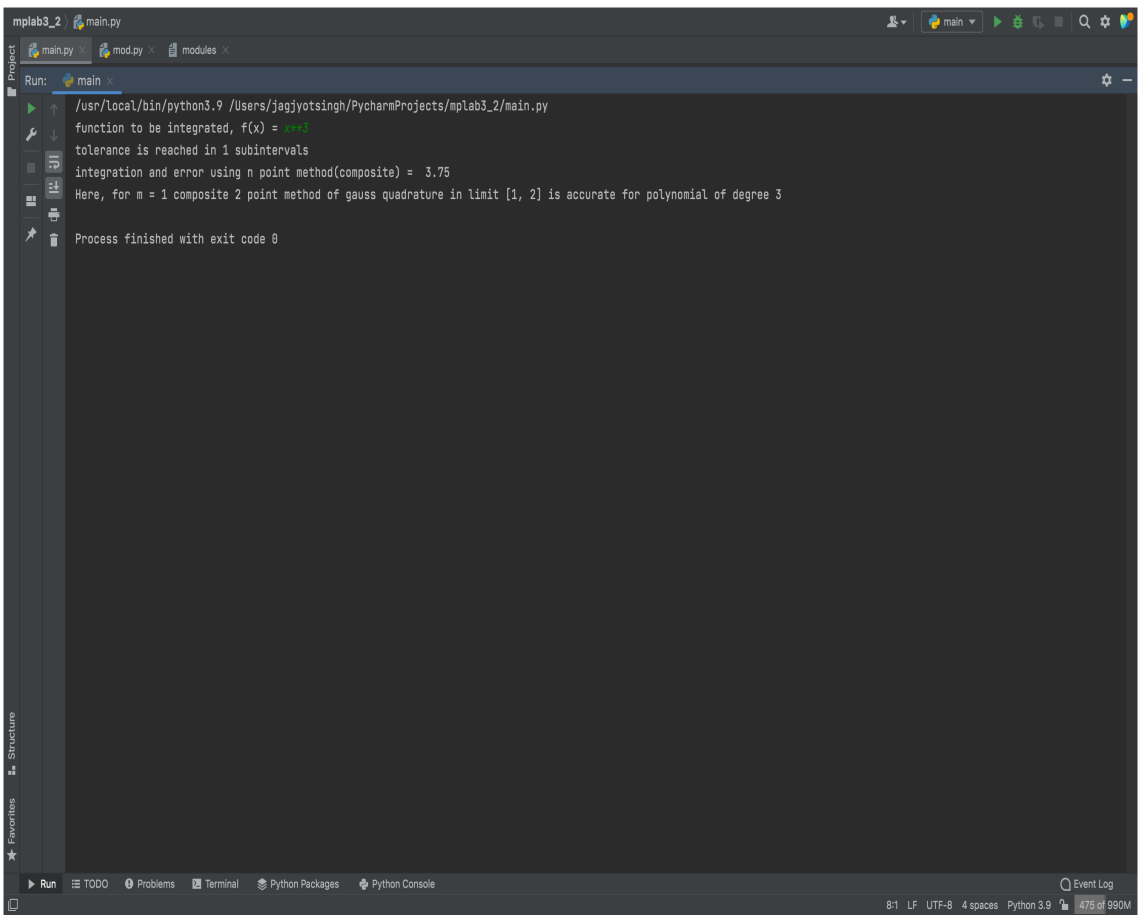
4.1 Programming

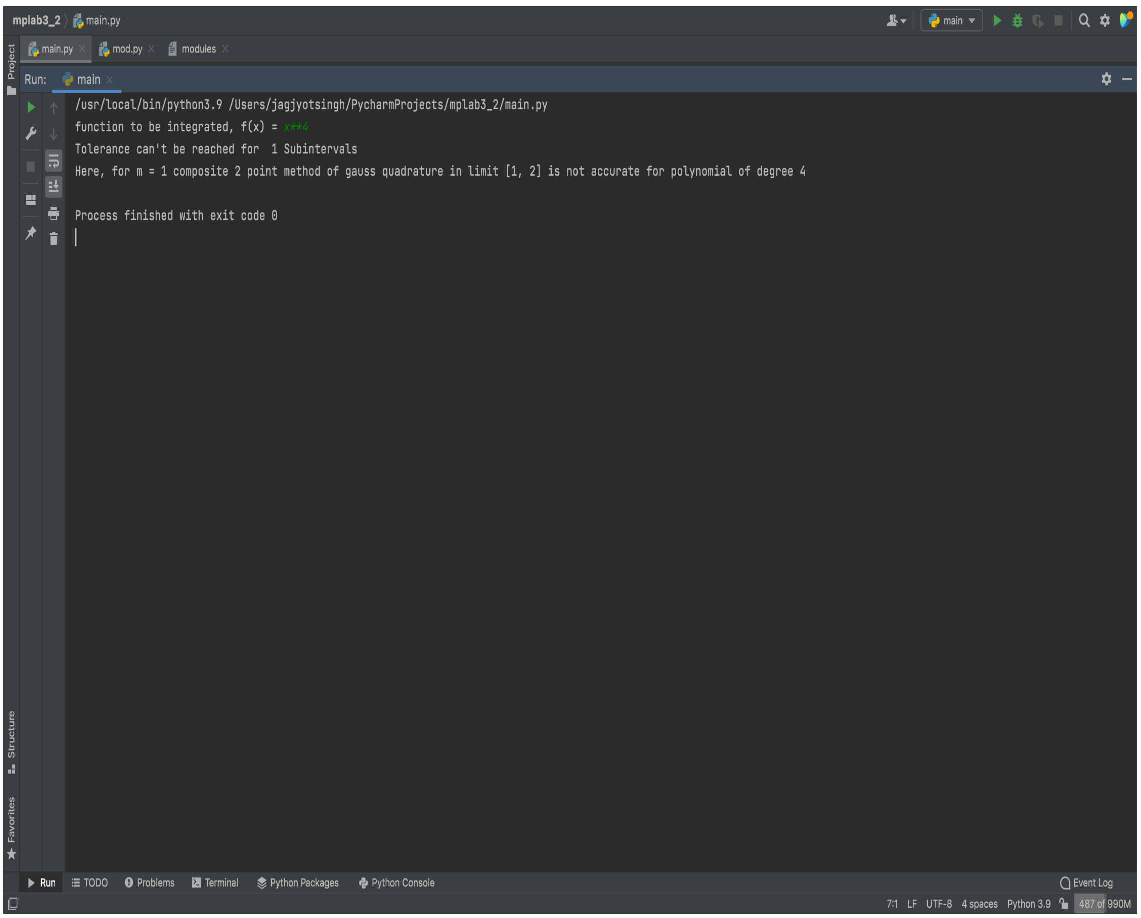

mplab3_2 main.py
main.py
mod.py
modules
Run:
main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x
tolerance is reached in 1 Intervals
integration using trapezoidal method = 49.5
integration using inbuilt function = (49.5, 0.0)
Here, Trapezoidal method in limit [1,10] gives exact results for linear function(x)
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
PEP 8: E265 block comment should start with '# '
9:1 LF UTF-8 4 spaces Python 3.9
457 of 990M

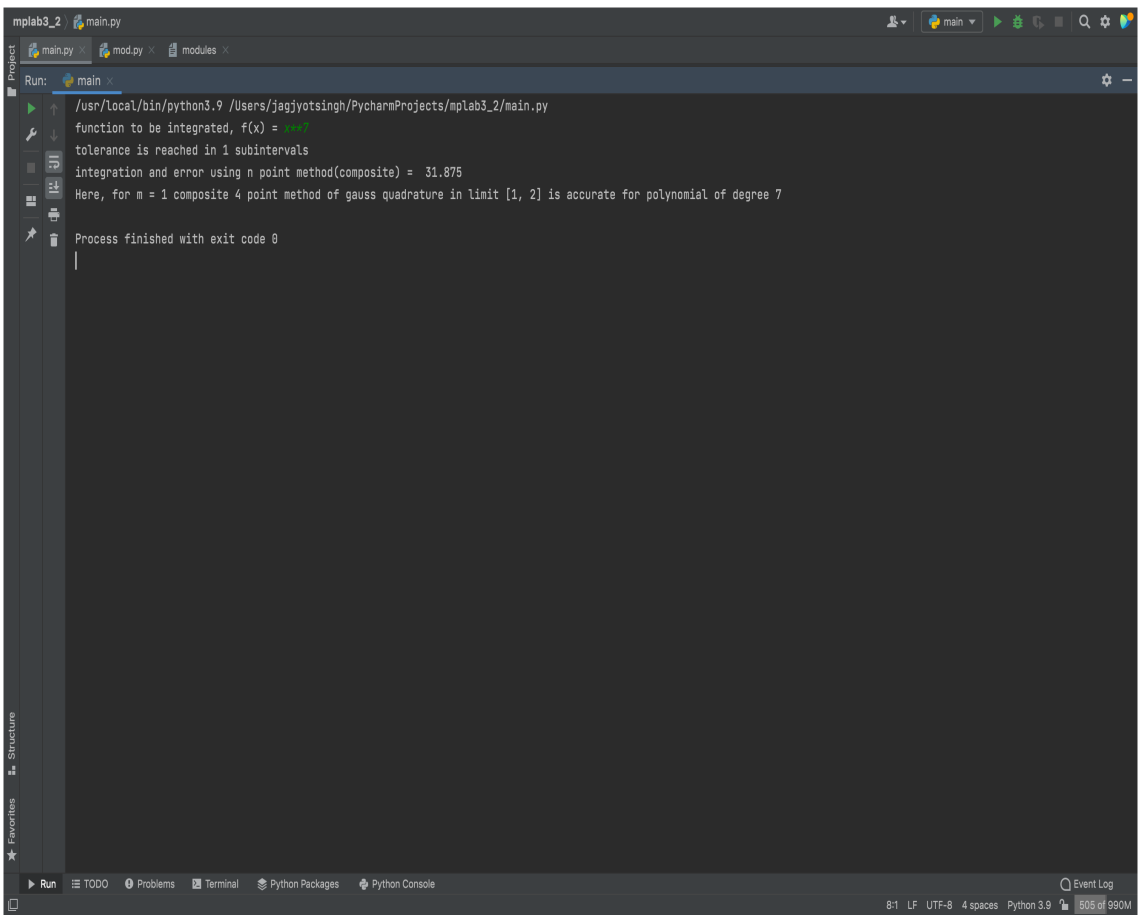

mplab3_2 main.py
main.py
mod.py
modules
Run: main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
xfunction to be integrated, f(x) = x**2
Tolerance can't be reached for 100 Subintervals
Here, Trapezoidal method in [1,10] limit doesn't give exact result for polynomial of order 2
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
7:1 LF UTF-8 4 spaces Python 3.9 404 of 990M

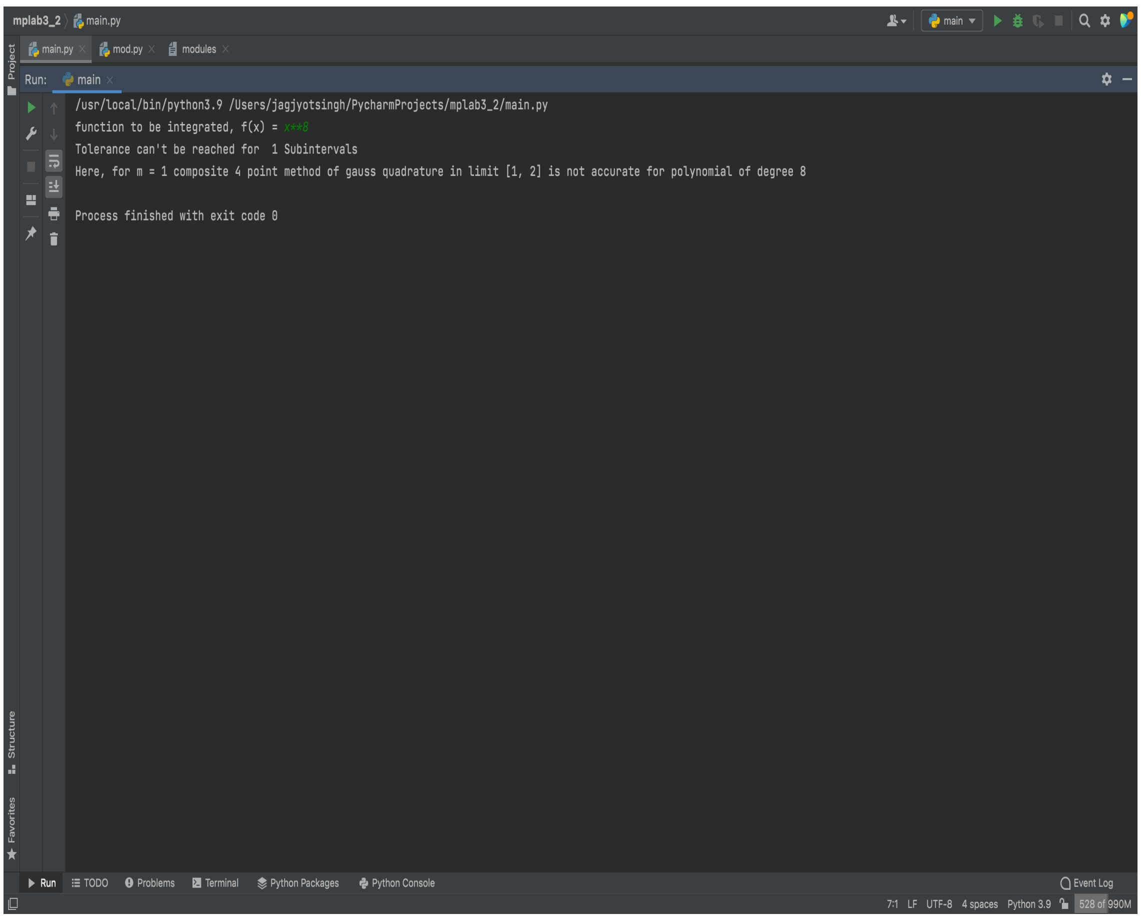

mplab3_2 main.py
main.py
mod.py
modules
Run: main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x**3
tolerance is reached in 1 Intervals
integration using simpson method = 2499.75
integration using inbuilt function = (2499.75, 0.0)
Here, Simpson method in limit [1,10] gives exact results for polynomial(x) of degree 3
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log


mplab3_2 main.py
main.py
mod.py
modules
Run: main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x**4
tolerance is reached in 29 Intervals
integration using simpson method = 19999.800695727725
integration using inbuilt function = (19999.8, 0.0)
Here, Simpson method in [1,10] limit doesn't give exact result for polynomial of order 4
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
9:1 LF UTF-8 4 spaces Python 3.9 476 of 990M

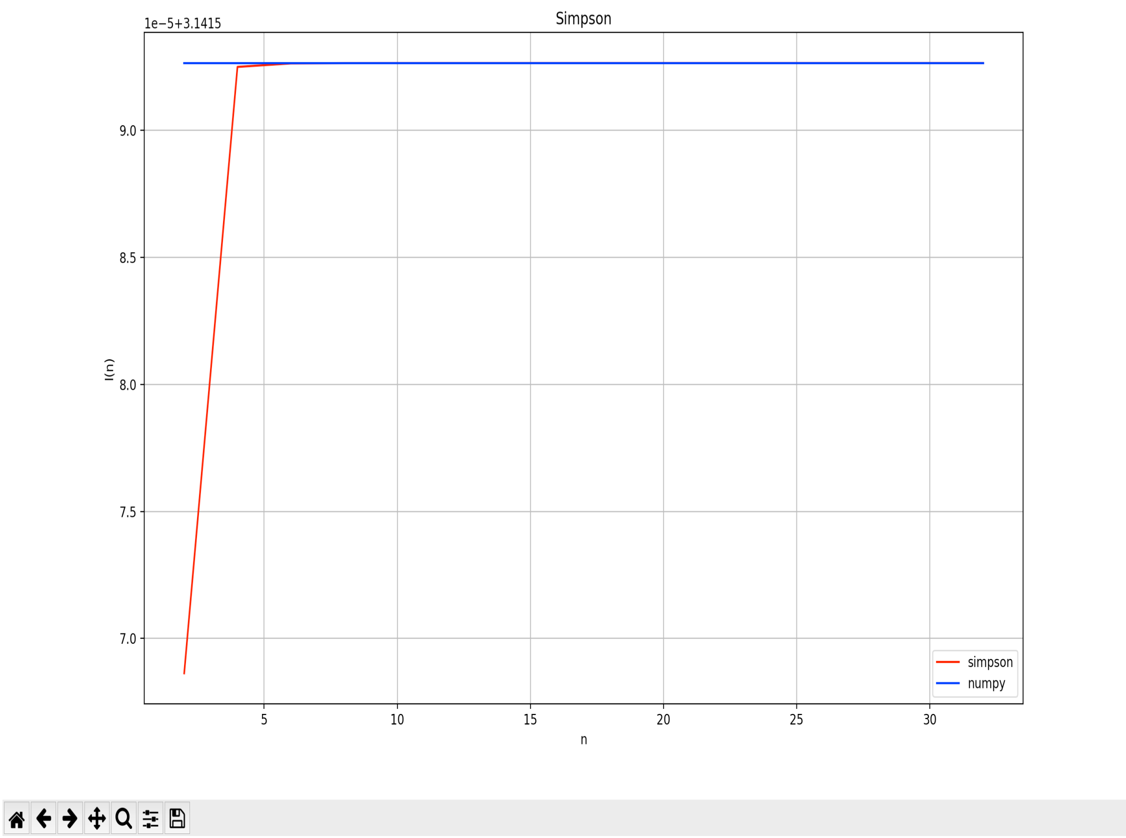

mplab3_2 main.py
main.py
mod.py
modules
Run: main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x**3
tolerance is reached in 1 subintervals
integration and error using n point method(composite) = 3.75
Here, for m = 1 composite 2 point method of gauss quadrature in limit [1, 2] is accurate for polynomial of degree 3
Process finished with exit code 0


mplab3_2
main.py
main.py
mod.py
modules
Run:
main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x**4
Tolerance can't be reached for 1 Subintervals
Here, for m = 1 composite 2 point method of gauss quadrature in limit [1, 2] is not accurate for polynomial of degree 4
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
7:1 LF UTF-8 4 spaces Python 3.9
487 of 990M

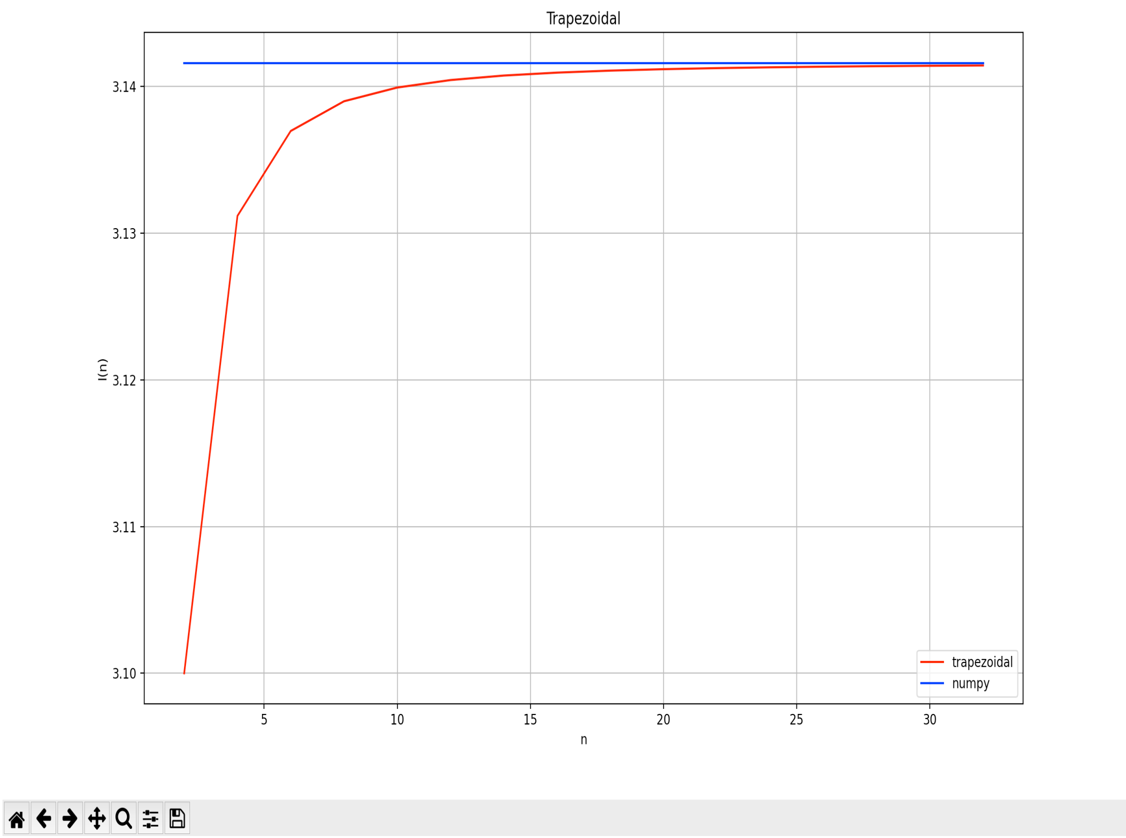

mplab3_2 main.py
main.py
mod.py
modules
Run: main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x**7
tolerance is reached in 1 subintervals
integration and error using n point method(composite) = 31.875
Here, for m = 1 composite 4 point method of gauss quadrature in limit [1, 2] is accurate for polynomial of degree 7
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
8:1 LF UTF-8 4 spaces Python 3.9 505 of 990M

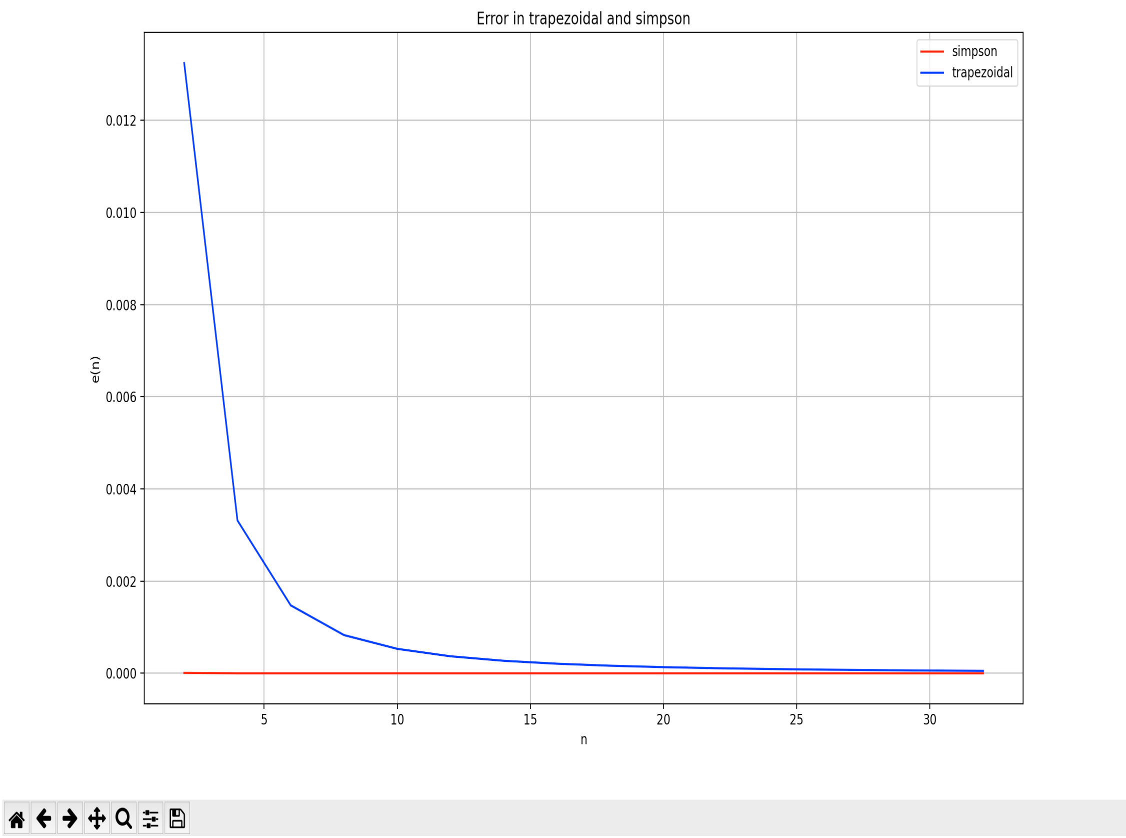


mplab3_2
main.py
main.py
mod.py
modules
Run:
main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = x**8
Tolerance can't be reached for 1 Subintervals
Here, for m = 1 composite 4 point method of gauss quadrature in limit [1, 2] is not accurate for polynomial of degree 8
Process finished with exit code 0
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log

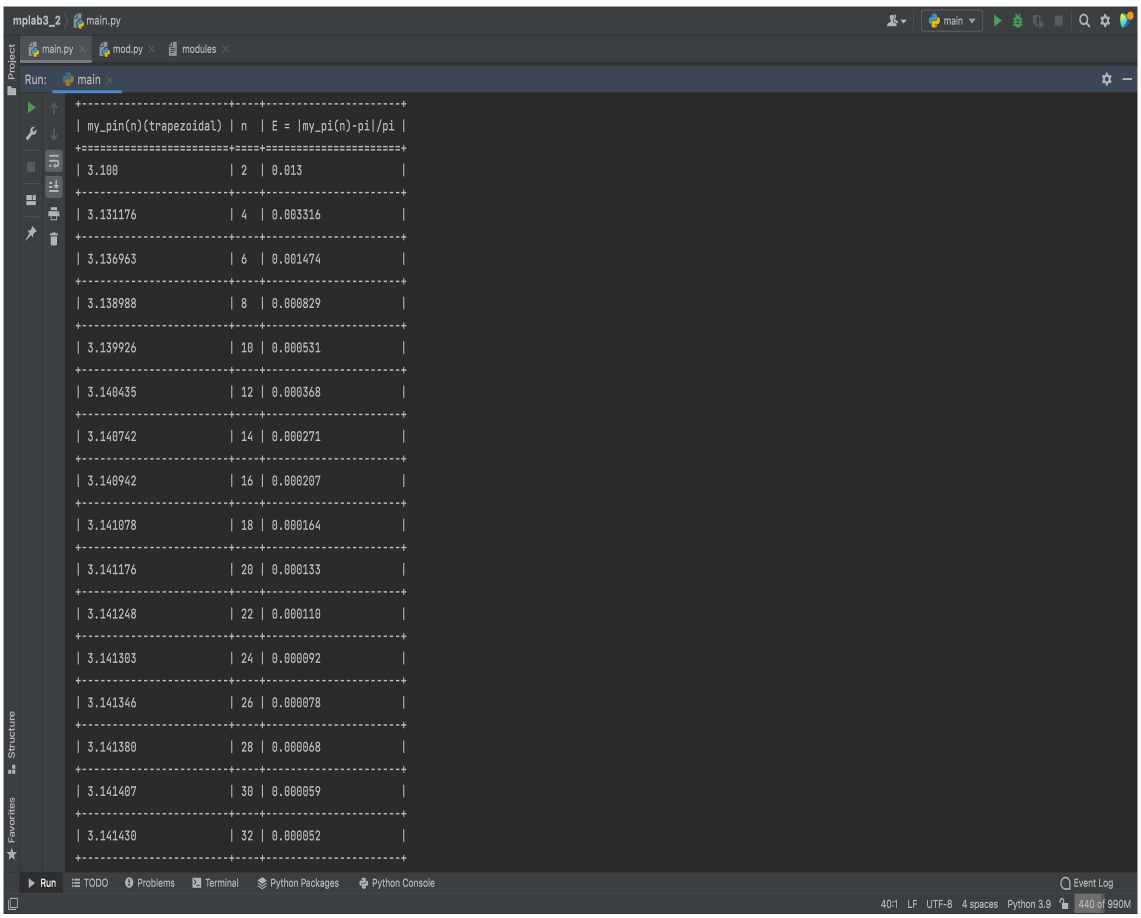


Simpson
1e−5+3.1415
9.0
8.5
8.0
7.5
7.0
I(n)
5
10
15
20
25
30
n
simpson
numpy

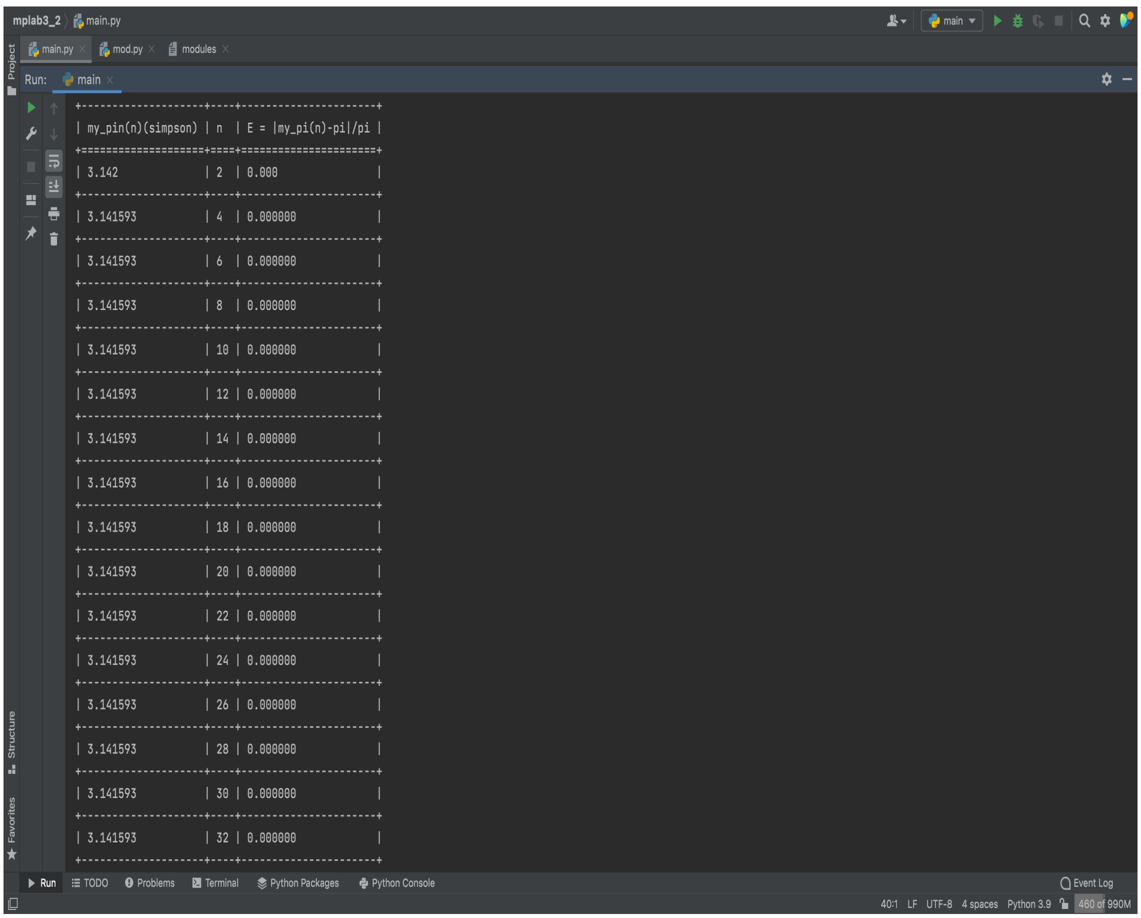

Trapezoidal
3.14
3.13
3.12
3.11
3.10
I(n)
5
10
15
20
25
30
n
trapezoidal
numpy

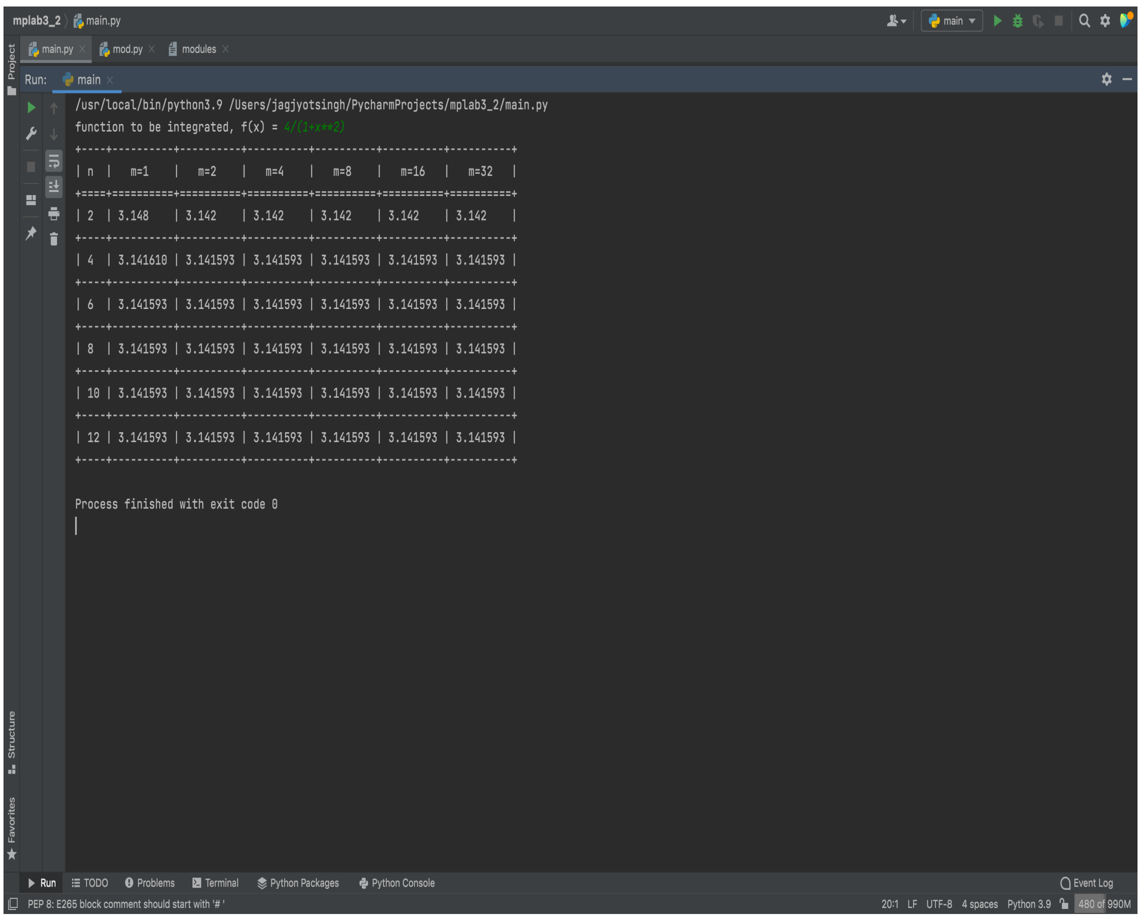


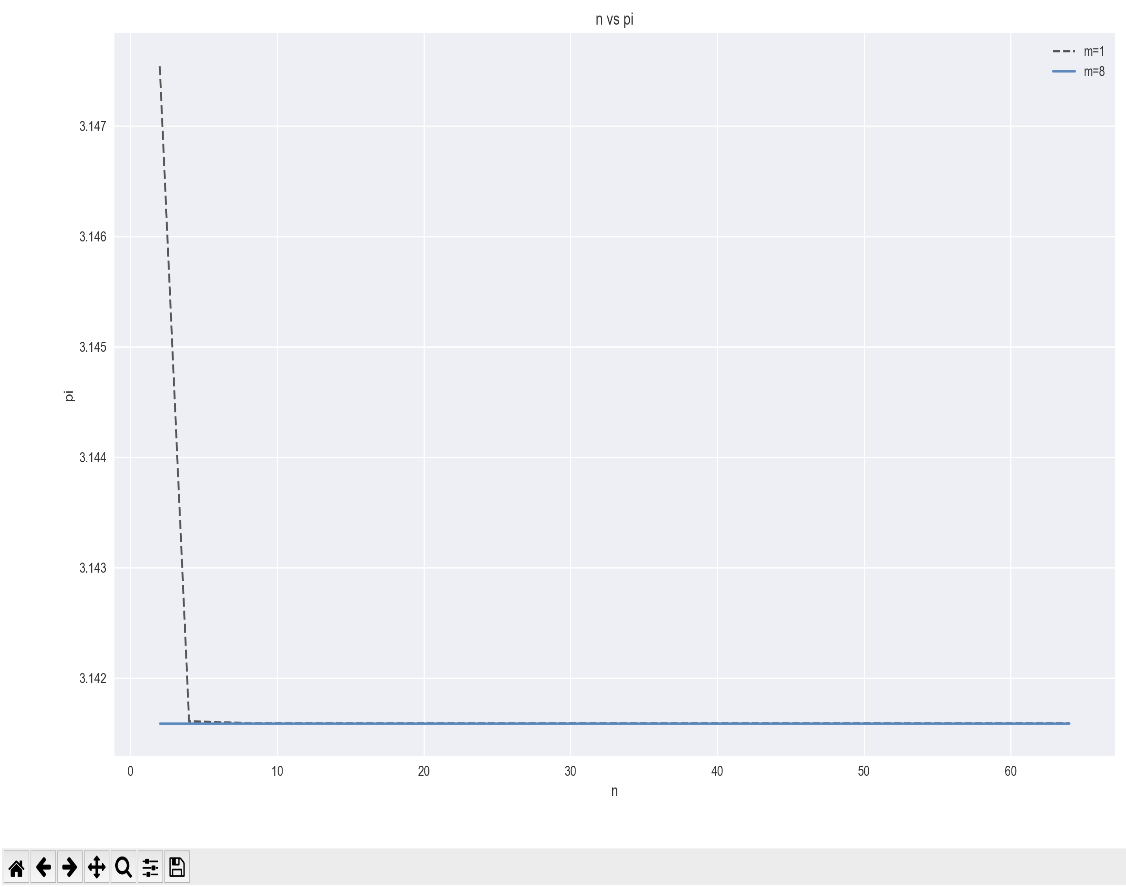
Error in trapezoidal and simpson
simpson
trapezoidal
0.012
0.010
0.008
0.006
0.004
0.002
0.000
e(n)
5
10
15
20
25
30
n

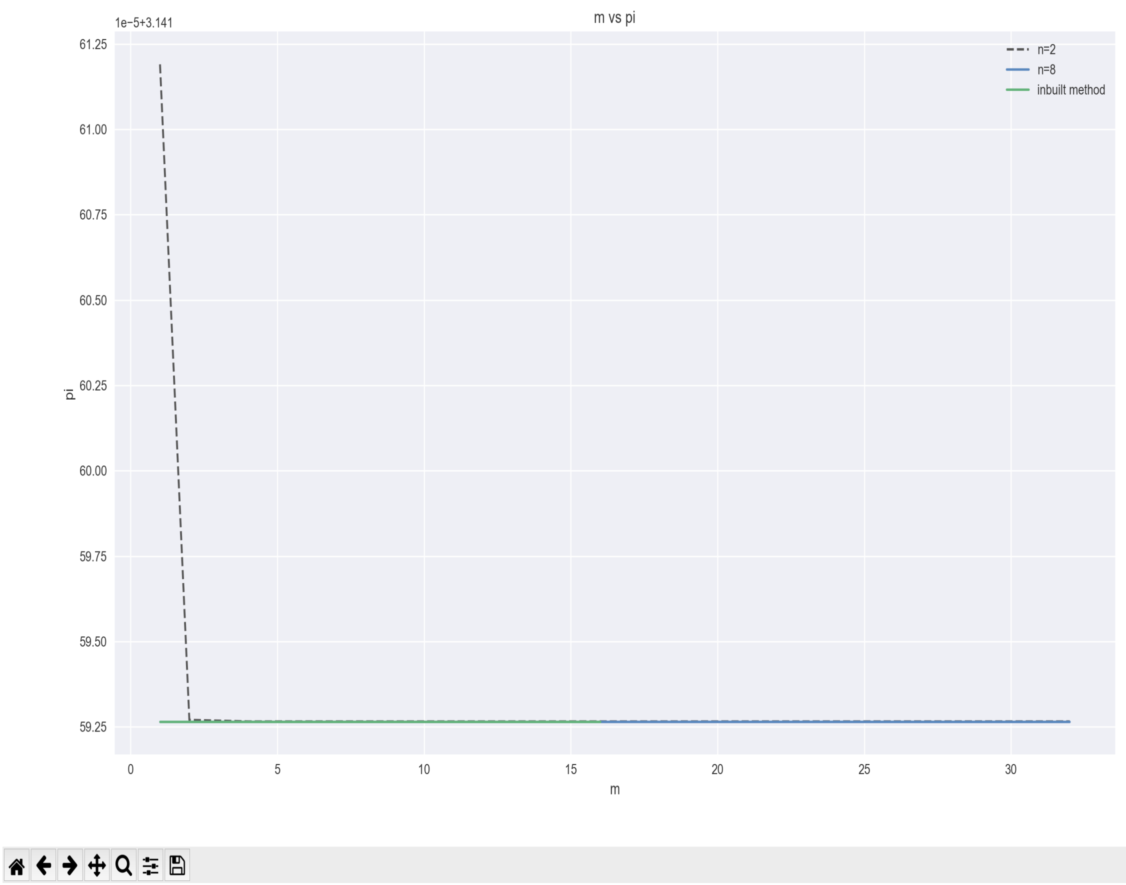


my_pin(n)(trapezoidal)	n	E = |my_pi(n)-pi|/pi
3.100	2	0.013
3.131176	4	0.003316
3.136963	6	0.001474
3.138988	8	0.000829
3.139926	10	0.000531
3.140435	12	0.000368
3.140742	14	0.000271
3.140942	16	0.000207
3.141078	18	0.000164
3.141176	20	0.000133
3.141248	22	0.000110
3.141303	24	0.000092
3.141346	26	0.000078
3.141380	28	0.000068
3.141407	30	0.000059
3.141430	32	0.000052


mplab3_2 main.py
main.py
mod.py
modules
Run: main
my_pin(n)(simpson)	n	E =	my_pi(n)-pi	/pi
3.142	2	0.000
3.141593	4	0.000000
3.141593	6	0.000000
3.141593	8	0.000000
3.141593	10	0.000000
3.141593	12	0.000000
3.141593	14	0.000000
3.141593	16	0.000000
3.141593	18	0.000000
3.141593	20	0.000000
3.141593	22	0.000000
3.141593	24	0.000000
3.141593	26	0.000000
3.141593	28	0.000000
3.141593	30	0.000000
3.141593	32	0.000000
Run
TODO
Problems
Terminal
Python Packages
Python Console
Event Log
40:1 LF UTF-8 4 spaces Python 3.9 460 of 990M


mplab3_2 main.py
main.py mod.py modules
Run: main
/usr/local/bin/python3.9 /Users/jagjyotsingh/PycharmProjects/mplab3_2/main.py
function to be integrated, f(x) = 4/(1+x**2)
n	m=1	m=2	m=4	m=8	m=16	m=32
2	3.148	3.142	3.142	3.142	3.142	3.142
4	3.141610	3.141593	3.141593	3.141593	3.141593	3.141593
6	3.141593	3.141593	3.141593	3.141593	3.141593	3.141593
8	3.141593	3.141593	3.141593	3.141593	3.141593	3.141593
10	3.141593	3.141593	3.141593	3.141593	3.141593	3.141593
12	3.141593	3.141593	3.141593	3.141593	3.141593	3.141593
Process finished with exit code 0
Run TODO Problems Terminal Python Packages Python Console Event Log
PEP 8: E265 block comment should start with '# '
20:1 LF UTF-8 4 spaces Python 3.9 480 of 990M


n vs pi
m=1
m=8
3.147
3.146
3.145
3.144
3.143
3.142
pi
0
10
20
30
40
50
60
n


m vs pi
1e−5+3.141
61.25
61.00
60.75
60.50
60.25
60.00
59.75
59.50
59.25
pi
0
5
10
15
20
25
30
m
n=2
n=8
inbuilt method

5 Python program

```python
from sympy import *
from scipy.special.orthogonal import p_roots
from scipy import integrate
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import texttable as tt
tab = tt.Texttable()

x = symbols('x')
f = eval("lambda x:" + input("function to be integrated, f(x) = "))
#d = int(input("Enter the number of significant figures upto which results should be accurate = "))


def My_Trap(f_, a, b, n):
    y = []
    h = (b - a) / n
    for i in range(n + 1):
        y.append(f(a + i * h))  # y at limit points
    trp = h * (f(a) + f(b)) / 2
    for j in range(1, len(y) - 1):
        trp = trp + h * (y[j])
    return (trp)


def My_Simp(f_, a, b, n):
    h = (b - a) / (2 * n)
    simp = h * (f(a) + f(b)) / 3
    for i in range(1, 2 * n):
        if (i % 2 == 0):
            simp = simp + 2 * h * f(a + i * h) / 3
        elif (i % 2 == 1):
            simp = simp + 4 * h * f(a + i * h) / 3

    return (simp)


def MyTrap_tol(f_, a, b, n, d):
    i = 1
    err = []
    while i <= n:
        e = abs(f_("x", a, b, i) - f_("x", a, b, i + 1)) / f_("x", a, b, i + 1)
        err.append(e)
        if e > 0.5 * 10 ** -d:
            i = i + 1
            if i > n:
                print("Tolerance can't be reached for ", n, "Subintervals")
        elif e <= 0.5 * 10 ** -d:
            print("tolerance is reached in", i, "Intervals")
            print("integration using trapezoidal method = ", f_("x", a, b, i))
            print("integration using inbuilt function = ", integrate.quadrature(f, a, b))
            break


def MySimp_tol(f_, a, b, n, d):
    i = 1
    err = []
    while i <= n:
        e = abs(f_("x", a, b, i) - f_("x", a, b, i + 1)) / f_("x", a, b, i + 1)
        err.append(e)
        if e > 0.5 * 10 ** -d:
            i = i + 1
            if i > n:
```

```python
                print("Tolerance can't be reached for ", n, "intervals")
        elif e <= 0.5 * 10 ** -d:
            print("tolerance is reached in", i, "Intervals")
            print("integration using simpson method = ", f_("x", a, b, i))
            print("integration using inbuilt function = ", integrate.quadrature(f, a, b))
            break


def MyLegQuadrature(fs, a, b, n, m):  # Gauss legendre Quadrature ftion
    h = (b - a) / m
    e = []
    [leg_zer, w] = p_roots(n)
    leg_zer.tolist()
    w.tolist()
    sum_ = 0
    x_ = [a]
    s = []
    for k in range(0, n):
        for i in range(1, m + 1):
            x_.append(a + h * i)
            sum_ += (h / 2) * w[k] * f(0.5 * h * leg_zer[k] + 0.5 * (x_[i] + x_[i - 1]))
            s.append(sum_)
    return sum_


def MyLegQuadrature_tol(fs, a, b, n, m, d):
    i = 1
    err = []
    while i <= m:
        e = abs(MyLegQuadrature(fs, a, b, n, i) - MyLegQuadrature(fs, a, b, n, i + 1)) / MyLegQuadrature(fs, a, b, n,
                                  i + 1)
        err.append(e)
        if e > 0.5 * 10 ** -d:
            i = i * 2
            if i > m:
                print("Tolerance can't be reached for ", m, "Subintervals")
        elif e <= 0.5 * 10 ** -d:
            print("tolerance is reached in", i, "subintervals")
            print("integration and error using n point method(composite) = ", MyLegQuadrature(fs, a, b, n, i))
            # print("integration using inbuilt ftion = ", integrate.quadrature(f, a, b))
            break
            return MyLegQuadrature(fs, a, b, n, i)


def MyLegQuadrature_tol_1(fs, a, b, n, d):
    i = 1
    p = 0
    err = []
    while True:
        e = abs(MyLegQuadrature(fs, a, b, n, i) - MyLegQuadrature(fs, a, b, n, i + 1)) / MyLegQuadrature(fs, a, b, n,
                                  i + 1)
        err.append(e)
        if e > 0.5 * 10 ** -d:
            i = i * 2
        elif e <= 0.5 * 10 ** -d:
            p = MyLegQuadrature(fs, a, b, n, i)
            break
    return p, i, d
```

```python
from Numerical_Methods import My_Trap,My_Simp,MyLegQuadrature, MyLegQuadrature_tol,
```

```python
    MySimp_tol,MyTrap_tol,MyLegQuadrature_tol_1
import numpy as np
import math
import matplotlib.pyplot as plt
import texttable as tt
tab = tt.Texttable()
# 3 b i
#MyTrap_tol(My_Trap, 1, 10, 100, 8)
#print("Here, Trapezoidal method in limit [1,10] gives exact results for linear
    function(x)")


#MyTrap_tol(My_Trap, 1, 10, 100, 8)
#print("Here, Trapezoidal method in [1,10] limit doesn't give exact result for
    polynomial of order 2")

# 3 b ii

#MySimp_tol(My_Simp, 1, 10, 100, 8)
#print("Here, Simpson method in limit [1,10] gives exact results for polynomial(x)
    of degree 3")


#MySimp_tol(My_Simp, 1, 10, 100, 8)
#print("Here, Simpson method in [1,10] limit doesn't give exact result for
    polynomial of order 4")


# 3 b iii

#MyLegQuadrature_tol("x", 1, 2, 2, 1, 8)
#print("Here, for m = 1 composite 2 point method of gauss quadrature in limit [1,
    2] is accurate for polynomial of degree 3")


#MyLegQuadrature_tol("x", 1, 2, 2, 1, 8)
#print("Here, for m = 1 composite 2 point method of gauss quadrature in limit [1,
    2] is not accurate for polynomial of degree 4")


#MyLegQuadrature_tol("x", 1, 2, 4, 1, 8)
#print("Here, for m = 1 composite 4 point method of gauss quadrature in limit [1,
    2] is accurate for polynomial of degree 7")

#MyLegQuadrature_tol("x", 1, 2, 4, 1, 8)
#print("Here, for m = 1 composite 4 point method of gauss quadrature in limit [1,
    2] is not accurate for polynomial of degree 8")


# 3 c
#Trapezoidal method table
m = np.arange(1,17,1)
n = 2*m
h = (1-0)/n
my_pi_1 = []
my_pi_2 = []
const = []
err_1, err_2 = [], []
for i in n:
    p = My_Simp("x", 0, 1, i)
    q = My_Trap("x", 0, 1, i)
    const.append(np.pi)
    e_1 = abs(p - np.pi)/np.pi
    e_2 = abs(q-np.pi)/np.pi
    err_1.append(e_1)
    err_2.append(e_2)
    my_pi_1.append(p)
```

```
    my_pi_2.append(q)
my_pi = [my_pi_1, my_pi_2]
#print(my_pi)
#plt.plot(n, err_1)
#plt.plot(np.log(h), np.log(err_1), c='r')
#plt.plot(np.log(h), np.log(err_2), c='b')
#plt.plot(n, err_2, c='b')
#plt.plot(n, const, c='b')
plt.grid(True)
plt.xlabel("log(h)")
plt.ylabel('log(error)')
plt.title("log plot for error vs step size")
plt.legend(["simpson", "trapezoidal"])
#plt.show()

# 3 d
'''headings_1 = ["my_pin(n)(simpson)","n","E = |my_pi(n)-pi|/pi"]
tab.header(headings_1)
for row in zip(my_pi_1, n, err_1):
    tab.add_row(row)
    tab.set_max_width(0)
    tab.set_precision(6)
s = tab.draw()
print(s)
tab.reset()'''
#mysimpson method table
'''headings_2 = ["my_pin(n)(trapezoidal)","n","E = |my_pi(n)-pi|/pi"]
tab.header(headings_2)
for row in zip(my_pi_2,n,err_2):
    tab.add_row(row)
    tab.set_max_width(0)
    tab.set_precision(6)
s = tab.draw()
print(s)
tab.reset()'''

# 3 e
n_ = [2, 4, 8, 16, 32, 64]
m_ = [1, 2, 4, 8, 16, 32]
matrix_pi_quad = []
e = []
'''for i in n_:
    for j in m_:
        u = MyLegQuadrature("x", 0, 1, i, j)
        matrix_pi_quad.append(u)
        error = abs(u - np.pi)/np.pi
        e.append(error)'''
for i in m_:
    for j in n_:
        u = MyLegQuadrature("x", 0, 1, i, j)
        matrix_pi_quad.append(u)
        error = abs(u - np.pi)/np.pi
        e.append(error)

t = np.reshape(matrix_pi_quad,(len(m_), len(n_)))
t_ = np.reshape(e,(len(m_), len(n_)))
#print(t)
#print(t_)

headings_3 = ["n","m=1","m=2","m=4","m=8","m=16","m=32"]
tab.header(headings_3)
for row in zip(n,t[0],t[1],t[2],t[3],t[4],t[5]):
    tab.add_row(row)
    tab.set_max_width(0)
    tab.set_precision(6)
s = tab.draw()
#print(s)
```

```
tab.reset()

# 3 e ii
'''plt.plot(n_, t_[0])
plt.plot(n_, t_[3])
plt.style.use('seaborn')
plt.grid(True)
plt.xlabel("n")
plt.ylabel('e(n)')
plt.title("e vs n")
plt.legend(["m = 1", "m = 8"])
plt.show()'''

plt.plot(m_, t_[0])
plt.plot(m_, t_[2])
plt.style.use('seaborn')
plt.grid(True)
plt.xlabel("m")
plt.ylabel('e(m)')
plt.title("e vs m")
plt.legend(["n = 2", "n = 8"])
plt.show()


'''plt.style.use('seaborn')
fig1, ax1 = plt.subplots()
ax1.plot(n_, t[0], color='#444444',
         linestyle='--', label='m=1')
ax1.plot(n_,t[3], label='m=8')
ax1.set_title('n vs pi')
ax1.set_xlabel('n')
ax1.set_ylabel('pi')
ax1.legend()
plt.tight_layout()'''
#plt.show()


#@np.vectorize
'''def const_value(m):
    return np.pi
for i in m:
    # cons_pi = []
    cons_pi = const_value(m)
plt.style.use('seaborn')
fig1, ax1 = plt.subplots()
ax1.plot(m_, t[1], color='#444444',
         linestyle='--', label='n=2')
ax1.plot(m_,t[3], label='n=8')
ax1.plot(m,cons_pi,label = "inbuilt method")
ax1.set_title('m vs pi')
ax1.set_xlabel('m')
ax1.set_ylabel('pi')
ax1.legend()
plt.tight_layout()
plt.show()'''


n_ = [2, 4, 8, 16, 32]
d = [1, 2, 3, 4, 5, 6, 7, 8]
matrix_pi_quad = []
for i in d:
    for j in n:
        u = MyLegQuadrature_tol_1("x", 0, 1, j, i)
        t = matrix_pi_quad.append(u)



```

```
g = np.reshape(t, (len(n_), len(d)))
print(g)
print(matrix_pi_quad)
```