A PRACTICAL FILE

Numerical Methods in Chemical Engineering (Python)

(BTCH-18406)

Submitted

for

BACHELOR OF TECHNOLOGY

in

CHEMICAL ENGINEERING



|  |  |
| --- | --- |
| Submitted To | Submitted by |
| **Dr. Vipan K Sohpal** | **XXXXX** |
| Associate Professor & | B. Tech (CHE) |
| Head, Chemical Engineering | Semester:4th Roll No. 20XXXX |

Department of Chemical Engineering & Bio Technology

S.B.S STATE UNIVERSITY GURDASPUR

Jan-May,2022

INDEX

Section-A (Python)

|  |  |  |
| --- | --- | --- |
| Sr. No | Content | Page No |
|  | Solution of a system of linear equations in unknowns by Gaussian elimination. |  |
|  | Gauss-Seidel iterative method to solve a linear system of equations. |  |
|  | Solution of least square curve fitting method. |  |
|  | Solution of nonlinear equation by Newton Raphson method. |  |
|  | Application of Newton's formulae for interpolation. |  |
|  | Application of Lagrange polynomial interpolation formula. |  |
|  | Application of Runge-Kutta formula for ordinary differentiation equation (ODE) |  |
|  | Application of Numerical integration by Trapezoidal rule. |  |
|  | Application of Numerical integration by Simpson's rules. |  |

Section-B (Polymath)

|  |  |  |
| --- | --- | --- |
| Sr. No | Content | Page No |
|  | Solution of a system of linear equations (5\*5 Matrices) |  |
|  | Gauss-Seidel iterative method to solve a linear system of equations. |  |
|  | Solution of least square curve fitting method. |  |
|  | Runge-Kutta formula for ordinary differentiation equation (ODE) |  |

**Bisection Method Algorithm**

1. start

2. Define function f(x)

3. Choose initial guesses x0 and x1 such that f(x0) f(x1) < 0

4. Choose pre-specified tolerable error e.

5. Calculate new approximated root as x2 = (x0 + x1)/2

6. Calculate f(x0) f(x2)

a. if f(x0) f(x2) < 0 then x0 = x0 and x1 = x2

b. if f(x0) f(x2) > 0 then x0 = x2 and x1 = x1

c. if f(x0) f(x2) = 0 then goto (8)

7. if |f(x2) | > e then goto (5) otherwise goto (8)

8. Display x2 as root.

9. Stop

In this python program, x0 and x1 are two initial guesses, e is tolerable error and nonlinear function f(x) is defined using python function definition def f(x):

# Defining Function

def f(x):

return x\*\*3-5\*x-9

# Implementing Bisection Method

def bisection (x0, x1, e):

step = 1

print ('\n\n\*\*\* BISECTION METHOD IMPLEMENTATION \*\*\*')

condition = True

while condition:

x2 = (x0 + x1)/2

print ('Iteration-%d, x2 = %0.6f and f(x2) = %0.6f' % (step, x2, f(x2)))

if f(x0) \* f(x2) < 0:

x1 = x2

else:

x0 = x2

step = step + 1

condition = abs(f(x2)) > e

print ('\nRequired Root is: %0.8f' % x2)

# Input Section

x0 = input ('First Guess: ')

x1 = input ('Second Guess: ')

e = input ('Tolerable Error: ')

# Converting input to float

x0 = float(x0)

x1 = float(x1)

e = float(e)

#Note: You can combine above two section like this

# x0 = float (input ('First Guess: '))

# x1 = float (input ('Second Guess: '))

# e = float (input ('Tolerable Error: '))

# Checking Correctness of initial guess values and bisecting

if f(x0) \* f(x1) > 0.0:

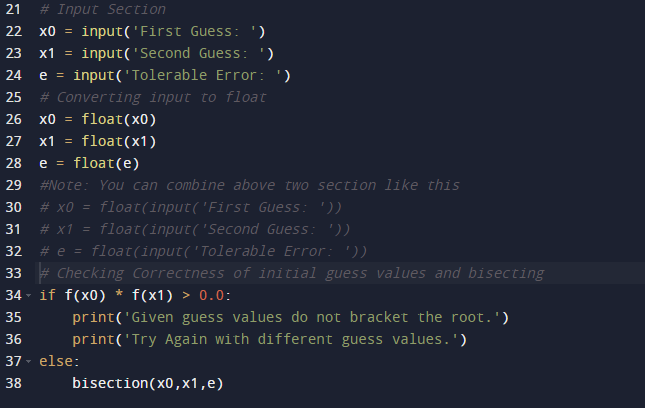
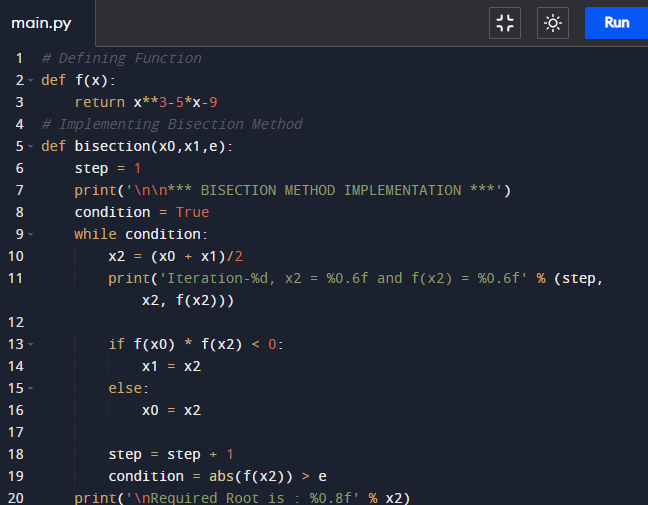
print('Given guess values do not bracket the root.')

print ('Try Again with different guess values.')

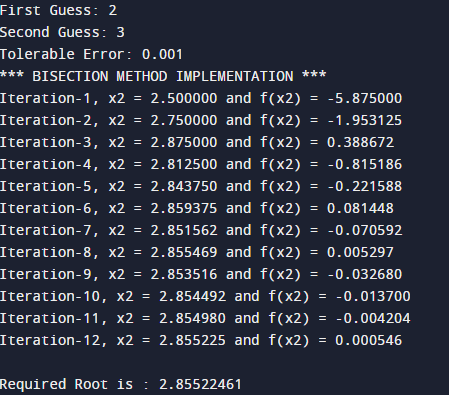
else:

bisection (x0, x1, e)

Execution of Programme



Output:



Algorithm for Newton Raphson Method

An algorithm for Newton Raphson method requires following steps in order to solve any non-linear equation with the help of computational tools:

1. Start

2. Define function as f(x)

3. Define first derivative of f(x) as g(x)

4. Input initial guess (x0), tolerable error (e)

and maximum iteration (N)

5. Initialize iteration counter i = 1

6. If g(x0) = 0 then print "Mathematical Error"

and goto (12) otherwise goto (7)

7. Calculate x1 = x0 - f(x0) / g(x0)

8. Increment iteration counter i = i + 1

9. If i >= N then print "Not Convergent"

and goto (12) otherwise goto (10)

10. If |f(x1) | > e then set x0 = x1

and goto (6) otherwise goto (11)

11. Print root as x1

12. Stop