■ Solution of Equations Using Bisection and Newton-Ramphson method.

Aim :- To implement the bisection method and the Newton Ramphson method in python and use them to find the roots of at least four mathematical equations. Analyse the convergence and compare the speed and reliability of both method.

Theory :-

The bisection method is a bracketing root finding method based on the intermediate value theorem. If $f(a) \cdot f(b) < 0$ then root lies in the interval $[a,b]$. At each step interval is halved and the subinterval containing the root is chosen for the next iteration.

Convergence :- Slow

eg :- $x^3 - 4x - 9 = 0$

$f(0) = -9 \quad f(1) = -12 \quad f(2) = -9 \quad f(3) = 6$

$\therefore f(2) \cdot f(3) < 0 \quad f(2.5) = -3.375$

$\therefore (2,3) \longrightarrow \frac{2+3}{2} = \underline{2.5}$

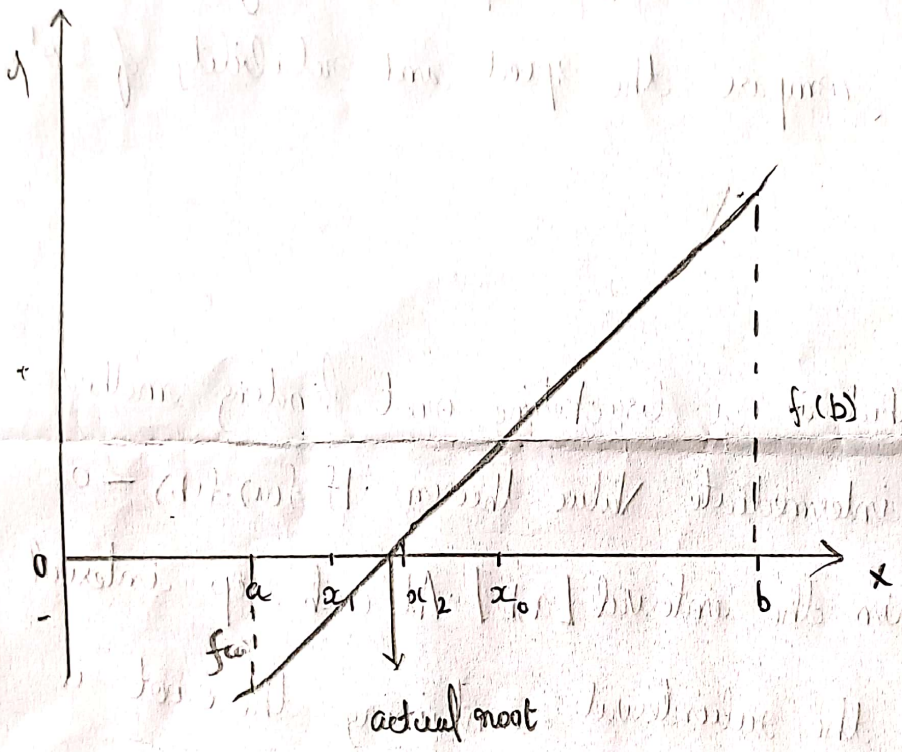
If it is -ve value replace @

New interval (2.5, 3)

$x_2 = \frac{2.5+3}{2} = 2.75$

$x = 2.75 = f(2.75) = 0.7968$ - +ve value replace interval b

Interval = $\therefore (2.5, 2.75) = \frac{a+b}{2} = \frac{2.5+2.75}{2} = x_3 = 2.7065$



Newton Rampson method:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

eg:- $x^3 - 2x - 5 = 0$

$f'(x_n) = 3x^2 - 2$

$f(0) = -5 < 0 \quad f(1) = -6 < 0 \quad f(2) = -1 < 0 \quad f(3) = 16 > 0$

$\therefore x_0 = 2$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{f(2)}{f'(2)} = 2.1$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 2.1 - \frac{f(2.1)}{f'(2.1)} = 2.0946$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 2.0946 - \frac{f(2.0946)}{f'(2.0946)} = 2.0946 \text{ -root}$$

Theory - Let x_0 be an approximate root of the Given equation $f(x) = 0$ which may be algebric or transcendental. Let $x_1 = x_0 + h$ be the exact Value or better approximation of the corresponding root, h being a small Quantity. Then $f(x_0 + h) = 0$

Expanding it by Tylors theorem. We get

$$f(x_0 + h) = f(x_0) + h f'(x_0) + h^2 f'(x_0)/2! + \cdots = 0$$

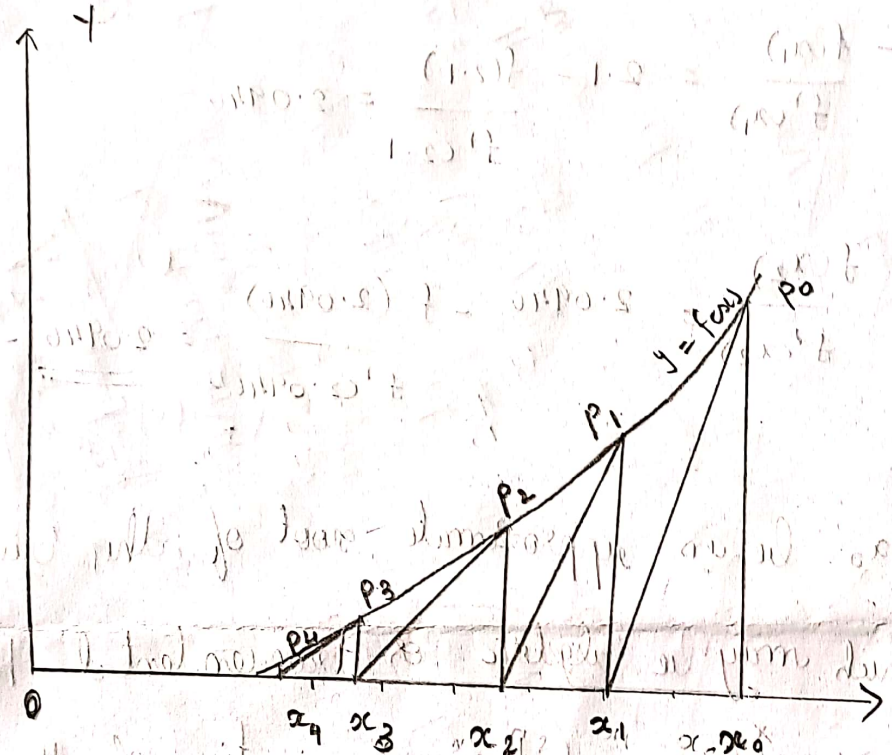
Since h is small we can neglect h^2, and higher degree terms

$$h = \frac{-f(x_0)}{f'(x_0)}$$

$$x_1 = x_0 + h$$

$$= x_0 - \frac{f(x_0)}{f'(x_0)}$$

The Graph shows the convergence of root from approximate to actual value.



Convergence :- Fast (Quadratic)

Pitfall :- May diverge if $f'(x_n) = 0$

Algorithm :-

① Bisection method

① Define the function

② Enter limit 'a'

③ Enter limit 'b'

④ If f(a). f(b) ≥ 0 Print "Invalid Interval" and stop

⑤ While absolute difference of a and b >= 0.000001 do step 7 to 9

⑥ Compute c = a+b/2

⑦ Check if f(c)* f(a) > 0 then assign the value of x to a

⑧ Else assign value of c to b

⑨ Print the approximate root c

② Newton Rampheon method

① Define the function.

③ Enter the approximation root x_0

④ Enter the delta variation in x

⑤ Start a while loop to do steps from 6-13 with infinite no of iteration

⑥ Compute $y = f(x_0)$

⑦ Compute $y_2 = f(x_0 + dx/2)$

⑧ Compute $y_1 = f(x_0 - dx/2)$

⑨ Compute $dy = y_2 - y_1$

⑩ Compute y prime = dy/dx

⑪ Compute the 1st approx root $x = x_0 - y/\text{prime}$

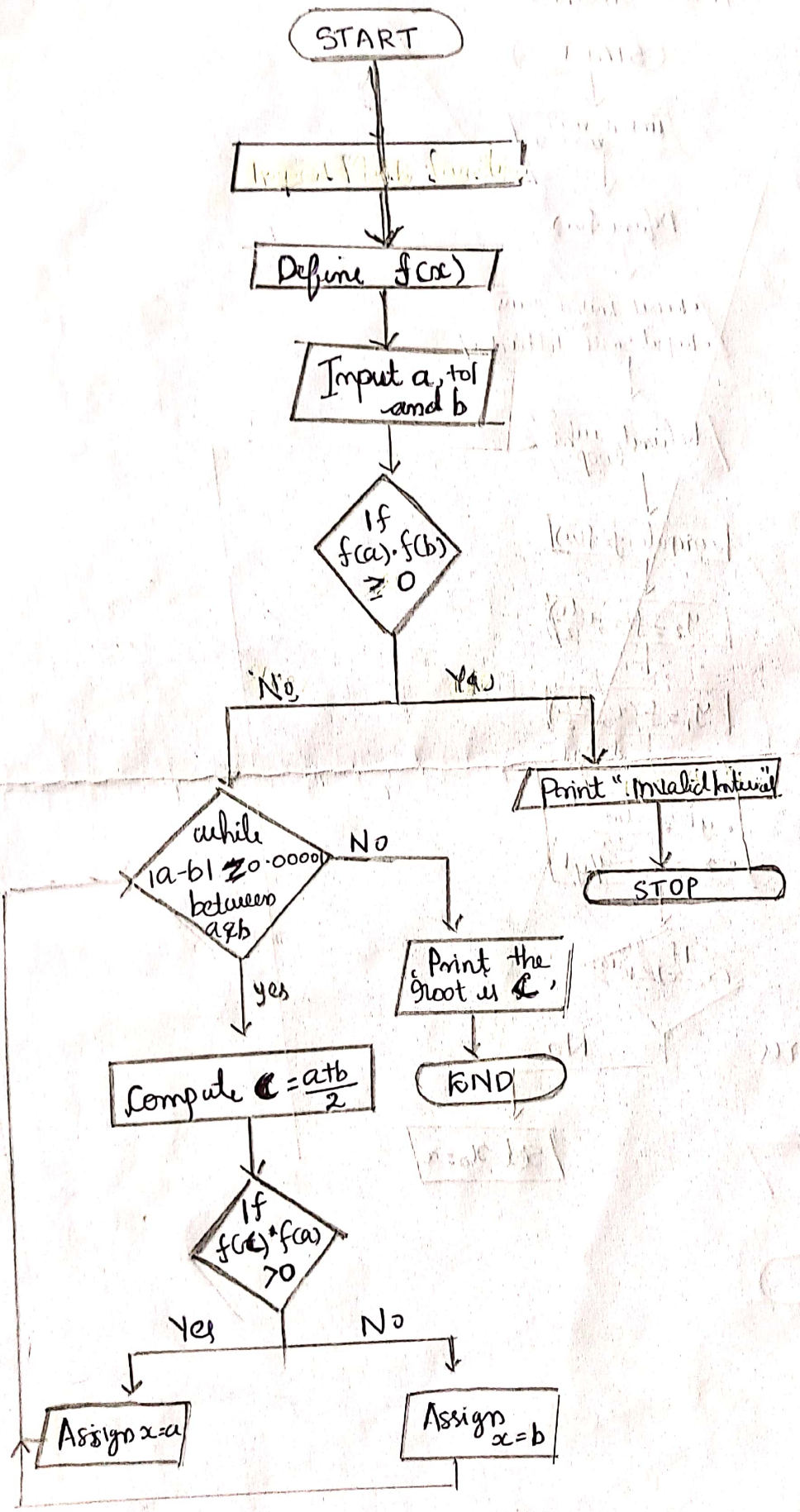
⑫ If $abs(x_0 - x) < 0.000000001$ the already value go to step 12

⑬ else replace value of x_0 by x go to step 5

⑭ print last solution, x

FLOW CHART :-

① Bisection method


START
Define f(x)
Input a, tol and b
If f(a).f(b) ≥ 0
No
Yes
Print "Invalid Initial"
STOP
while |a-b| ≥ 0.0000 between a&b
No
yes
Print the Root is c
END
Compute c = a+b/2
If f(c)*f(a) >0
Yes
No
Assign x=a
Assign x=b

Bisection Method

```
def f(x):
    return ( )

def bisection (a,b,tol):
  if f(a) * f(b) ≥ 0 :
    Print ("Bisection method fails. f(a) and f(b) must have
                opposite signs")
    return None

  While (b-a)/2.0 > tol :
    c = (a+b)/2.0
    if f(c) == 0 :
      Break
    elif f(a) * f(c) < 0 :
      b = c
    else:
      a = c
  return c

a = float (input ("Enter the lower bound a:" )
b = float ( input ("Enter the upper bound b:")
tol = 0.000 001
root = bisection (a, b, tol)
if root is not None :
```

```
from Ipython.display import display
display (f "Approximate root: {root}")
```

Newton Raphson method

```
def f(x):
    return ( )
def df(x):
    return ( )
def newton_rampheon (x0, tol, max_iter):
    for i in range (max_iter):
        f_x0 = f(x0)
        df_x0 = df(x0)
        if df_x0 == 0:
            Print ("zero derivative. No solution found")
            return None
        x1 = x0 - f_x0/df_x0
        if abs(x1-x0) < tol:
            return x1
        x0 = x1
    Print ("Exceeded maximum iteration")
    return None
x0 = float (input ("Enter the x0 value"))
```

```
tol = float(input("Enter the tol"))
max_iter = int(input("Enter the maximum iteration"))
root = newton_raphson(x0, tol, max_iter)
if root is not None:
    from Ipython display import display
    display(f"Approximate root: {root}")
```

Newton-Raphson Method

Algorithm

Step 1 : Start

Step 2 : Define function $f(x)$ and it derivative $df(x)$.

Step 3 : input initial guess x0, tolerance tol, and maximum iteration max_iter

Step 4 : For i from 0 to max_iter - 1 :

- compute $f(x0)$
- compute $df(x0)$
- If $df(x0) == 0$:
  - Print "zero derivative. No solution Found"
  - Return None
- compute next approximation

$$x_1 = x_0 - \frac{f(x_0)}{df(x_0)}$$

- If $|x1 - x0| < tol$:
  - Return x1 as root

~~Else~~

Step 5:

Else set x0 = x1 and repeat.

Step 5: If loop completh without returning:

- Print "Exceeded maxium iterations"
- Return None

Step 6: End

Flow Chart