MP-2 LAB ASSIGNMENT-1

Name:Yogesh chad Date:11-03-2025

Roll no:245673989

Course:BSc Physics Hons

Question 1

Determine the depth up to which a spherical homogeneous object of given radius and density will sink into a fluid of given density. **Bisection Method**

Input:

```
import numpy as np

d_s = float(input("Enter density of solid sphere: "))  d_l =
float(input("Enter density of liquid: "))  R =
float(input("Enter radius of solid sphere: "))
x = int(input("Enter the number of significant figures: "))

def f(h):      return (4/3) * d_s * np.pi * R**3 - d_l * np.pi * (R**2) * h + (d_l * np.pi * h**3) / 3

a  = float(input("Enter lower limit: "))
b  = float(input("Enter upper limit: "))

if f(a) * f(b) <= 0:      while round(f(b), x) != round(f(a), x):         mid = (a + b) / 2
if f(mid) > 0:             b = mid         else:            a = mid         print(f'Value of
function at {round(mid, x)} is {round(f(mid), x)}')  else:
    print("Error: Root not found in given range!")
```

Output:

```
Enter density of solid sphere:  1
Enter density of liquid:  2
Enter radius of solid sphere:  1
Enter the number of significant figures:  8
Enter lower limit:  1
Enter upper limit:  2
Value of function at 1.5 is 1.83259571
Value of function at 1.25 is 0.42542401
Value of function at 1.125 is 0.10226539
Value of function at 1.0625 is 0.02505502
```

Value of function at 1.03125 is 0.00619984
Value of function at 1.015625 is 0.00154197
Value of function at 1.0078125 is 0.00038449
Value of function at 1.00390625 is 9.6e-05
Value of function at 1.00195312 is 2.398e-05
Value of function at 1.00097656 is 5.99e-06
Value of function at 1.00048828 is 1.5e-06
Value of function at 1.00024414 is 3.7e-07
Value of function at 1.00012207 is 9e-08
Value of function at 1.00006104 is 2e-08
Value of function at 1.00003052 is 1e-08
Value of function at 1.00001526 is 0.0

Newton Raphson Method

Input:

```
Import numpy as np
Def fun(x, r, rho_b, rho_f):      t = (4 * rho_b * r**3) / rho_f

   Return t + x**3 – 3 * r * x**2
Def fun_prime(x, r):
   Return 3 * x**2 – 6 * r * x
Def newraph_method(a, r, rho_b, rho_f, tol, max_iter):
   A_values = []      for I in range(max_iter):         a_values.append(a)         new_a = a – (fun(a, r, rho_b,
rho_f) / fun_prime(a, r))         if abs(new_a – a) < tol:            a_values.append(new_a)            return
new_a, a_values         a = new_a
   Return a, a_values
R = float(input("Enter radius of ball: "))  rho_b = float(input("Enter density of ball: "))  rho_f =
float(input("Enter density of fluid: "))
Initial_guesses = [r/2, r/3, 2*r/3, r/4, r/5]
Tolerance = 1e-5
Max_iterations = 100
For guess in initial_guesses:      solution, a_values = newraph_method(guess, r, rho_b, rho_f, tolerance,
max_iterations)      if solution > 0:
      Print(f"\nInitial guess: {guess}")
      Print(f"The solution to the equation is approximately {solution:.5f} metres.")         print(f"Number of
iterations: {len(a_values)}")      else:
      Print("\nThere is no valid root for the given values.")
```

Output:

Enter radius of ball: 2
Enter density of ball: 1
Enter density of fluid: 2

Initial guess: 1.0
The solution to the equation is approximately 2.00000 metres.
Number of iterations: 5

Initial guess: 0.6666666666666666
The solution to the equation is approximately 2.00000 metres.
Number of iterations: 6

Initial guess: 1.3333333333333333
The solution to the equation is approximately 2.00000 metres.
Number of iterations: 5

Initial guess: 0.5
The solution to the equation is approximately 2.00000 metres.
Number of iterations: 7

There is no valid root for the given values.

Secant Method

Input:

```
Def f(h):
   Return 4*R**3*p_r – 3*R*h**2 + h**3

Def secant_method(f, x0, x1,R, tol=1e-4):
   If f(x0) * f(x1) > 0:
      Print("Invalid range f(x0) and f(x1) must have opposite signs.")
      Return None

   While abs(x1 – x0) > tol:
      If abs(f(x1) – f(x0)) < 1e-10:
         Print("Denominator too small, stopping iteration.")
         Return None

      X_new = x1 – f(x1) * (x1 – x0) / (f(x1) – f(x0))
      X0, x1 = x1, x_new

   Print("The sphere will sink till the depth of:",x_new,"cm")

R = float(input("Enter the radius of the sphere(cm):"))
P = float(input("Enter the density of the sphere:"))
P_w=float(input("Enter the density of the medium:"))
P_r=p/p_w
X0=0
X1=2*R

Secant_method(f, x0, x1, R)
```

Output:
Enter the radius of the sphere(cm): 8
Enter the density of the sphere: 1
Enter the density of the medium: 3
The sphere will sink till the depth of: 6.191410296298554 cm

Question 2

Solve transcendental equations like α = tan(α).

Bisection Method for tan(x) = x

Input:

```
import numpy as np

def f(alpha):      return alpha -
np.tan(alpha)

def bisection_method(a, b, tol=1e-5, max_iter=100):
   if f(a) * f(b) >= 0:
      print("Bisection method fails. f(a) and f(b) must have opposite signs.")         return
None      for i in range(max_iter):
      mid = (a + b) / 2
      f_mid = f(mid)         if
abs(f_mid) < tol:
         return mid         if
f(a) * f_mid < 0:
         b = mid
else:
         a = mid
   return (a + b) / 2

a  = -4.0
b  = 10.0

root = bisection_method(a, b)

if root is not None:
   print(f"\nRoot found: {root}")
   print(f"f(root) = {f(root)}")
```

Output:

Root found: -1.5707963267948966
f(root) = 1.6331239353195368e+16 ii)

Newton-Raphson Method import
numpy as np try: def derivative(x):
return (1/(np.cos(x))**2) - 1 def
func(x):
return np.tan(x) - x x =
float(input("Enter limit: "))
iter = 0 while
func(x)!=0: iter+=1
x = x - (func(x)/derivative(x))
print(f"Value of limit after {iter} iteration: {x}") print(f"\nRoot
of tan(x)=x is: {x:.7f}") except: print("Please enter appropriate
values.")

Output:- Enter limit: 0.785 Value of limit after 1 iteration:
0.5704545852916935 Value of limit after 2 iteration:
0.39760588637524774 Value of limit after 3 iteration:
0.27078684177965123 Value of limit after 4 iteration:
0.18230817233485475 Value of limit after 5 iteration:
0.12207994965751587 Value of limit after 6 iteration:
0.08154870436220613 Value of limit after 7 iteration:
0.05441405447662749 Value of limit after 8
iteration: 0.03629036362162512
Value of limit after 9 iteration: 0.024197824908414583
Value of limit after 10 iteration: 0.016133142814688403 Value of limit
after 11 iteration: 0.010755801810954927
Value of limit after 12 iteration: 0.007170645147738028
Value of limit after 13 iteration: 0.004780462872184877
Value of limit after 14 iteration: 0.003186984958983447
Value of limit after 15 iteration: 0.0021246595166175635
Value of limit after 16 iteration: 0.0014164405302937496
Value of limit after 17 iteration: 0.0009442939395035378
Value of limit after 18 iteration: 0.0006295293678598411
Value of limit after 19 iteration: 0.0004196862672665797
Value of limit after 20 iteration: 0.00027979085152528597
Value of limit after 21 iteration: 0.00018652723624164682
Value of limit after 22 iteration: 0.00012435149123797475
Value of limit after 23 iteration: 8.290099514708213e-05
Value of limit after 24 iteration: 5.526732982416914e-05
Value of limit after 25 iteration: 3.6844887821175886e-05
Value of limit after 26 iteration: 2.4563258793384978e-05
Value of limit after 27 iteration: 1.637550672196755e-05
Value of limit after 28 iteration: 1.0917003640427937e-05
Value of limit after 29 iteration: 7.27800150469474e-06
Value of limit after 30 iteration: 4.852009284232332e-06
Value of limit after 31 iteration: 3.2346919147017603e-06
Value of limit after 32 iteration: 2.156470428328954e-06
Value of limit after 33 iteration: 1.4376286788761985e-06
Value of limit after 34 iteration: 9.58434998404343e-07

Value of limit after 35 iteration: 6.389298974763607e-07
Value of limit after 36 iteration: 4.260515359294517e-07
Value of limit after 37 iteration: 2.839342188884958e-07
Value of limit after 38 iteration: 1.893549478399421e-07
Value of limit after 39 iteration: 1.2641836640008865e-07
Value of limit after 40 iteration: 8.502625197273405e-08
Value of limit after 41 iteration: 5.793323162028376e-08
Value of limit after 42 iteration: 3.8065016695153556e-08
Value of limit after 43 iteration: 2.5292592814712707e-08
Value of limit after 44 iteration: 1.5358485352147605e-08
Root of tan(x)=x is: 0.0000000

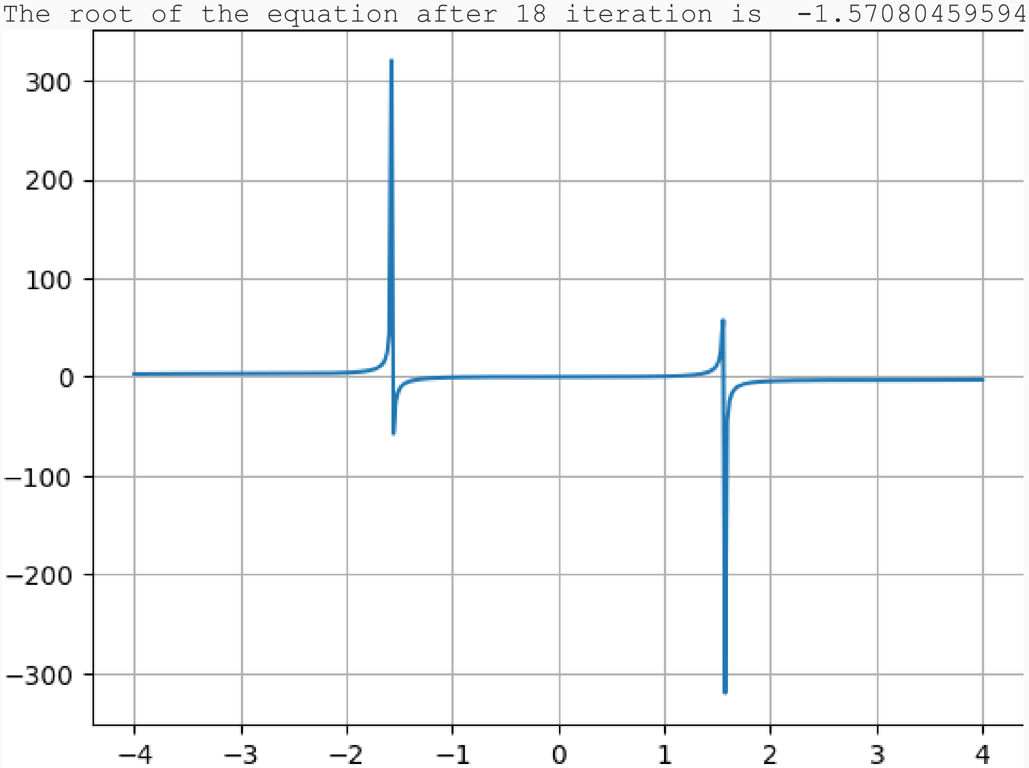
Newton Raphson Method

```
Import numpy as np
Import matplotlib.pyplot as plt
Def f(x):
  Return np.tan(x)-x
Def bisect(f, a, b):
  N=1
  While abs((b – a)/2)> 1e-5 or
f((a+b)/2)==0:
    C = (a + b) / 2
    If f© == 0:
      Print(“The root of the
equation is”,c)
      Break
    Elif f(a) * f(c) < 0:
      B = c
    Else:
      A = c
    Print(“The root of the
equation after”,n,”iteration is
”,c,)
    N+=1
A = float(input(“Enter the
starting point of the initial guess:
“))
B = float(input(“Enter the ending
point of the initial guess: “))
Bisect(f,a,b)
X = np.linspace(-2*b, 2*b, 400)
Y = f(x)
Plt.plot(x, y)
Plt.grid(True)
Plt.show()
```

Output:

Trinket_plot.png

```
Trinket_plot.pngEnter the starting point of the initial guess:  -3
Enter the ending point of the initial guess:  2
The root of the equation after 1 iteration is  -0.5
The root of the equation after 2 iteration is  -1.75
The root of the equation after 3 iteration is  -1.125
The root of the equation after 4 iteration is  -1.4375
The root of the equation after 5 iteration is  -1.59375
The root of the equation after 6 iteration is  -1.515625
The root of the equation after 7 iteration is  -1.5546875
The root of the equation after 8 iteration is  -1.57421875
The root of the equation after 9 iteration is  -1.564453125
The root of the equation after 10 iteration is  -1.5693359375
The root of the equation after 11 iteration is  -1.57177734375
The root of the equation after 12 iteration is  -1.570556640625
The root of the equation after 13 iteration is  -1.5711669921875
The root of the equation after 14 iteration is  -1.57086181640625
The root of the equation after 15 iteration is  -1.570709228515625
The root of the equation after 16 iteration is  -1.5707855224609375
The root of the equation after 17 iteration is  -1.5708236694335938
The root of the equation after 18 iteration is  -1.5708045959472656
```

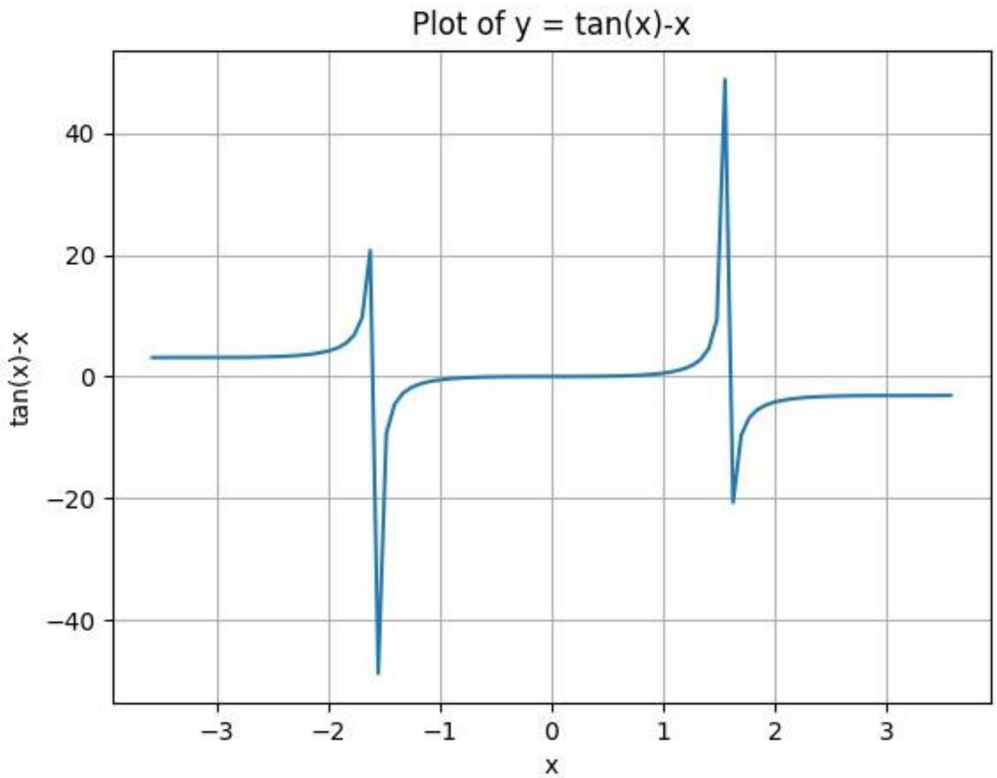


trinket_plot.png

Secant Method

import numpy as np def
fun(a):
return np.tan(a)-a def
secant_met(fun,a,b,tol,max_iter): for _ in
range(max_iter): f_b=fun(b)
f_a=fun(a)
if abs(f_b-f_a)<tol: print("The difference between the functional values is
too small") return None c=b-(f_b*(b-a))/(f_b-f_a) if abs(c-b)<tol:
return c
a,b=b,c
print("Maximum iterations exceeded.")
return None x0=3.14 x1=3.14*6
tolerance=0.000001 max_iterations=100
root=secant_met(fun, x0, x1, tolerance, max_iterations)
if root is not None:
print(f"The root of the equation tan(x)=x is approximately {root:6f}") else:
print("The Secant Method did not converge.") Output:
The root of the equation tan(x)=x is approximately 0.000000

iv) Graph import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(-np.pi/2 - 2, np.pi/2 + 2, 100) y =
np.tan(x) - x plt.plot(x, y)
plt.xlabel('x') plt.ylabel('tan(x)-x')
plt.title('Plot of y = tan(x)-x')
plt.grid() plt.show()
Output:-



Question 3

To approximate the nth root of a number up to a given number of significant digits.

Bisection Method

```
Import numpy as np
Import matplotlib.pyplot as plt
Def f(x):
  Return np.tan(x)-x
Def bisect(f, a, b):
  N=1
  While abs((b – a)/2)> 1e-5 or f((a+b)/2)==0:
    C = (a + b) / 2
    If f© == 0:
      Print("The root of the equation is",c)
      Break
    Elif f(a) * f(c) < 0:
      B = c
    Else:
      A = c
    Print("The root of the equation after",n,"iteration is ",c,)
    N+=1
A = float(input("Enter the starting point of the initial guess: "))
B = float(input("Enter the ending point of the initial guess: "))
Bisect(f,a,b)
```

```
X = np.linspace(-2*b, 2*b, 400)
Y = f(x)
Plt.plot(x, y)
Plt.grid(True)
Plt.show()
```

Output:

```
Enter the starting point of the initial guess:  4.1
Enter the ending point of the initial guess:  3.6
The root of the equation after 1 iteration is  3.8499999999999996
The root of the equation after 2 iteration is  3.9749999999999996
The root of the equation after 3 iteration is  4.0375
The root of the equation after 4 iteration is  4.00625
The root of the equation after 5 iteration is  3.9906249999999996
The root of the equation after 6 iteration is  3.9984374999999996
The root of the equation after 7 iteration is  4.00234375
The root of the equation after 8 iteration is  4.000390625
The root of the equation after 9 iteration is  3.9994140624999996
The root of the equation after 10 iteration is  3.9999023437499996
The root of the equation after 11 iteration is  4.000146484375
The root of the equation after 12 iteration is  4.0000244140625
The root of the equation after 13 iteration is  3.9999633789062496
The root of the equation after 14 iteration is  3.9999938964843746
The root of the equation after 15 iteration is  4.000009155273437
Trinket_plot.png
Trinket_plot.png
```

Newton Raphson Method

```
def nth_root(a, n, tol, max_iter):
x = a / n  for i in
range(max_iter):
f_x = x**n - a  der_x =
n*x**(n-1)  new_x = x-
(f_x/der_x)  if abs(x-new_x)
< tol:
return new_x  x =
new_x  return x
x = int(input("Enter an integer: "))
n = int(input(f"Enter n to find nth root of {x}: "))  Root =
nth_root(x, n, 0.00001, 100)  print(f'The required root of the
number is : {Root:4f} ')  Output:
Enter an integer: 64
Enter n to find nth root of 64: 3
The required root of the number is : 4.000000
```

Secant Method def fun(x,a,n): return x**n-a

```
def secant_met(fun,x0,x1,a,tol,max_iter):  for _ in
range(max_iter):
f_x1=fun(x1,a,n)
 f_x0=fun(x0,a,n)
if  abs(f_x1-f_x0)<tol:  print("The difference between the functional values is
too small")  return None  x2=x1-(f_x1*(x1-x0))/(f_x1-f_x0)  if abs(x2-x1)<tol:
return x2
x0,x1=x1,x2  print("Maximum iterations
exceeded.")  return None
x = int(input("Enter an integer: "))
 n = int(input(f"Enter n to find nth root of {x}: "))  x0 = x/n  x1 =
x/(n-1)  tolerance = 0.00001  max_iterations = 100
root=secant_met(fun,x0,x1,x,tolerance,max_iterations)  if root is
not None:
print(f"The {n}th root of {x} is approximately {root:.5f}")  else:
print("The Secant Method did not converge.")
```

Output:-

```
Enter an integer: 12
 Enter n to find nth root of 120:5
 The 7th root of 128 is approximately 2.605
```

iv) **Graph** import numpy as np

```
import matplotlib.pyplot as plt

def nth_root(A, n):     tol =
0.00001     x = A / n  # Initial
guess
   iterations = [x]  # List to store iteration values

   while True:        x_new = ((n - 1) * x + A / (x ** (n -
1))) / n
      iterations.append(x_new)

      if abs(x_new - x) < tol:  # Convergence condition           break
      x = x_new

   return iterations

x = int(input("Enter an integer: "))
n = int(input(f"Enter n to find nth root of {x}: "))

iterations = nth_root(x, n)

plt.plot(range(len(iterations)), iterations, marker='o', linestyle='-', label=f'{n}th Root Approximation')
plt.axhline(y=x**(1/n), color='r', linestyle='--', label='Actual Root') plt.xlabel('Iteration')
plt.ylabel('Approximation')
```

```
plt.title(f'Convergence of {n}th Root Approximation for {x}')
plt.legend() plt.grid()
plt.show()
```

Output: