Name : Mudit sand

Roll No. : 203100068

Assignment : 1

```
## Importing the libraries
import math
import numpy as np
import matplotlib.pyplot as plt
```

+ Code + Text

Problem 1 : Trajectory problem

```
## Getting the inputs

l = int(input(" Please enter the endpoint 1 i.e. theata = 0 : "))
u = int(input(" Please enter the endpoint 2 i.e. theata = 45 : "))
R = int(input(" Please enter the Range : "))
V = int(input(" Please enter the velocity : "))
n_max = int(input(" Plese enter the max no. of iteration : "))
tol = float(input(" Plese enter the value of tolerance i.e. .0001 : "))
```

Saved successfully!

```
                                    e. theata = 0 : 1
                                    e. theata = 45 : 44

      Please enter the velocity : 10
      Plese enter the max no. of iteration : 100
      Plese enter the value of tolerance i.e. .0001 : .0001
```

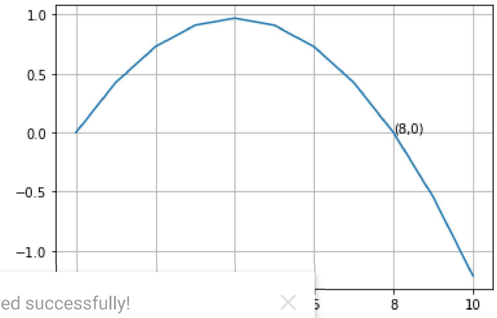
```
## Defining function for trajectory problem
def func(theta,r=R,vel=V):
  fp = (math.tan(theta)*r) - (0.5*9.81*(r/(vel*math.cos(theta)))**2)
  return fp
```

```
## Defining Bisection method for trajectory problem
def bisect(lowerbound,upperbound,max_iter,tolerance):

  a = lowerbound
  b = upperbound
  iter = 1
  Nmax = max_iter
  p = a + (b-a)/2
  fa = func(p)

  while iter < Nmax:
    p = a + (b-a)/2
    fp = func(p)

    if fp == 0 or abs(b-a) < tolerance :
      value = p
      print(" No. of iteration is : " , iter)
      return value
    iter += 1
    if fa* fp > 0:
      a = p
      fa = fp

    if fp * fa <0:
      b = p

  if iter == Nmax :
    return 'No. of iteration exceeded'
```

```
## Calling bisection method for trajectory problem
## Error Handling
range_max = V*V/9.81
if range_max < R:
  print("Invalid Range")
else:
  p= l * math.pi/180
  q= u * math.pi/180
  try:
   result = bisect(p,q,n_max,tol)
   #print(result)
   print(" Result is given below ")
   print(" approx theta is : " , result* 180/math.pi)
  except:
    print(bisect(p,q,n_max,tol))
```

```
      No. of iteration is :  14
      Result is given below
```

```
## Plotting the trajectory for theta = 25.85 for different ranges.
x = np.arange(0,range_max,1)
theta = 25.85 * math.pi/180
plt.plot(func(theta,r=x))
plt.grid()
plt.annotate('(8,0)',(8,0))
plt.show()
```



Saved successfully! ✕

We can verify from the above diagram that for given example of range 8 y(x) == 0 for theta = 25.85

```
### Result for different values of lower and upper bound
print("for a = 2 and b = 45",)
```

Problem 2 : Hanging Chain

```
## Getting the input
x1 = float(input(" Please enter the x 1 : "))
x2 = float(input(" Please enter the x 2 : "))
y1 = float(input(" Please enter y1 : "))
y2 = float(input(" Please enter y2 : "))
L = float(input(" Please enter L : "))
a_init = float(input(" Please enter initial guess : "))
n_max = int(input(" Plese enter the max no. of iteration : "))
tol = float(input(" Plese enter the value of tolerance i.e. .0001 : "))
```

```
      Please enter the x 1 : 0
      Please enter the x 2 : 6
      Please enter y1 : 8
      Please enter y2 : 26
      Please enter L : 20
      Please enter initial guess : .8
      Plese enter the max no. of iteration : 100000
      Plese enter the value of tolerance i.e. .0001 : .0001
```

```
## Writing the function of hanging chain
def func(a,x2=x2,x1=x1,L=L,y2=y2,y1=y1):
    return 2 * a * np.sinh((x2-x1) / (2 * a)) - np.sqrt(L ** 2 - (y2 - y1) ** 2)
```

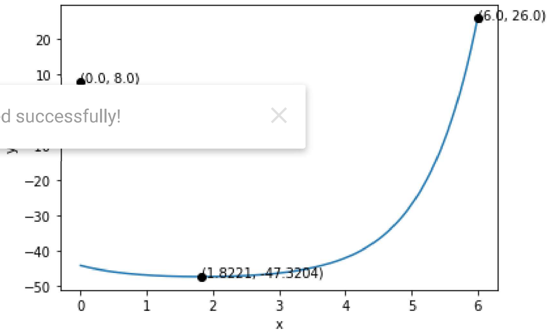
```
from scipy.misc import derivative
## Getting derivative for newtons method
def deri(a):
  return derivative(func,a)
```

```
## Coding Newton's Method
def newton(initialguess, max_iteration, tolerance):
  iter = 1
  while iter < max_iteration:
    a = initialguess - func(initialguess)/deri(initialguess)
    if abs(a-initialguess) <= tolerance:
      print(" No. of Iteration : " , iter)
      print(" value of a is : ", a)
      return a
    iter = iter + 1
    initialguess = a
  if iter == max_iteration:
    print("max iteration reaches")
```

```
a = newton(a_init,n_max,tol)

      No. of Iteration :  1
      value of a is :  0.8000773339691702
```

```
x0 = (x1 + x2) / 2 + (a / 2) * np.log((L - (y2 - y1)) / (L + (y2 - y1)))
y0 = y2 - a * (np.cosh((x2 - x0) / a) - 1)
```

```
## Plotting
xa = np.arange(x1, x2, 0.001)
ya = y0 + a * (np.cosh((xa - x0) / a) - 1)
plt.plot(xa, ya)
x_coor = [x1, x2, x0]
y_coor = [y1, y2, y0]
plt.xlabel('x')
plt.ylabel('y')
plt.plot(x_coor, y_coor, 'o', color='black')
plt.annotate((round(x0,4),round(y0,4)),(x0,y0))
plt.annotate((round(x1,4),round(y1,4)),(x1,y1))
plt.annotate((round(x2,4),round(y2,4)),(x2,y2))
plt.show()
```



Saved successfully! ×

✓ 0s completed at 10:25 PM ● ×

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.