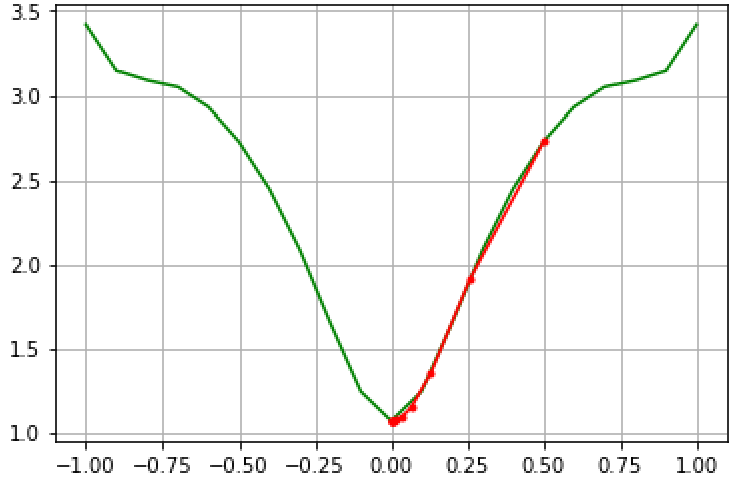
```
#in this project we will be implementing gradient descent algorithm for the given functior
#so we proceed as below

#first we import some important libraries we gonna be using for implementation
import numpy as np
from numpy import asarray,arange
import matplotlib.pyplot as plt
from numpy.random import rand
```

```
#gradient descent implementation of one-dimensional function
#defining the problem function **see attached pdf
def function_problem_def(x):
  return (-20*(np.exp(1)**(-0.125*(x**2))))-(np.exp(1)**(0.5*np.cos(2*np.pi*x)))+20+np.exp

# calculating the derivertive of the given function ***see attached pdf for its derivatiti
def function_derivertive(x):
  return (5*x*(np.exp(1)**(-0.125*(x**2))))

# defing the gradient descent algorithm itself
def gradient_descent_algorithm(function_problem_def, function_derivertive, stopping_val, r
  points, score_val = list(), list() ##this will be storing the values of the outcome
  sol = stopping_val[:, 0] + rand(len(stopping_val)) * (stopping_val[:, 1] - stopping_val[
  for i in range(number_of_iter):
    gradient_value = function_derivertive(sol)  #this calculates the gradient of the giver

    sol = sol - step_size * gradient_value  ##taking a step size during the gradient value
    solution_eval = function_problem_def(sol)

    #we will be appending the obtained soltuions into our list defined earlier
    points.append(sol)
    score_val.append(solution_eval)

    print('>%d reducing minimum point(%s) = %.5f' % (i, sol, solution_eval))


  return [points, score_val]




#initializing the number of iterations
number_of_iter = 11

#initializing the step size for the gradient descent  1e-5
step_size = 0.1

#determing our stopping criterian to be bound on the said domain
stopping_val = asarray([[-1.0, 1.0]])

#running the gradient descent algorithm to get the minimum point
points, score_val = gradient_descent_algorithm(function_problem_def, function_derivertive,

inputs = arange(stopping_val[0,0], stopping_val[0,1]+0.1, 0.1)#increamenting the inputs fr
final_val = function_problem_def(inputs)
```

```
print('The above last array([****]) point represents the obtained minimum point!')
#plotting the graph for a better visualization of the local minimum point to be obtained
plt.plot(inputs, final_val,color='green')
plt.plot(points, score_val, '.-', color='red')
plt.grid()
plt.show()
```

```
>0 reducing minimum point([0.50166869]) = 2.73112
>1 reducing minimum point([0.2586025]) = 1.91142
>2 reducing minimum point([0.13037762]) = 1.35381
>3 reducing minimum point([0.06532718]) = 1.14730
>4 reducing minimum point([0.03268101]) = 1.08946
>5 reducing minimum point([0.01634269]) = 1.07456
>6 reducing minimum point([0.00817162]) = 1.07081
>7 reducing minimum point([0.00408584]) = 1.06987
>8 reducing minimum point([0.00204293]) = 1.06964
>9 reducing minimum point([0.00102146]) = 1.06958
>10 reducing minimum point([0.00051073]) = 1.06957
The above last array([****]) point represents the obtained minimum point!
```




```
#make sure to compare the average outcome to obtain the local minimum point
#as found on the function derivertive, minimum point for this function is at aound the  0.
```

END OF GRADIENT DESCENT ALGORITHM IMPLEMETATION AND EVALUATION ***THANK YOU!!!***

✓ 0s completed at 10:51 PM