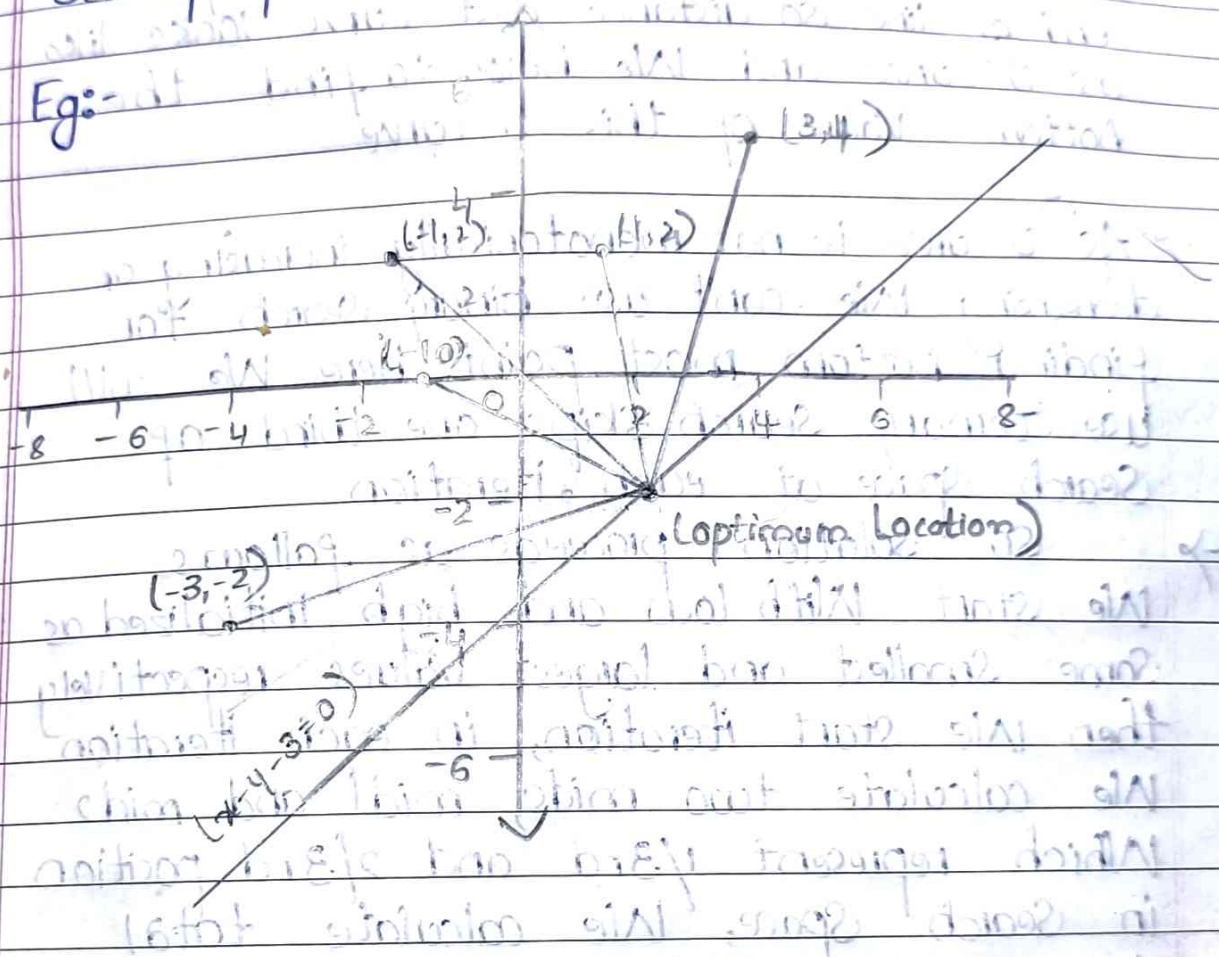
GoodLuck | Page No.
Date

Optimum location of point to minimize total distance

Given a set of points as and a line as $ax+by+c=0$, We need to find a point on given line for which sum of distances from given set of points is minimum

Eg:-



In the above figure optimum location of point of $x-y-3=0$ line is $(2,-1)$, whose total distance with other points is 20.77, which is minimum obtainable total distance.

If We take one point on given line at infinite distance then total distance cost will be infinite, now When we move this point on line towards given points the total distance cost starts decreasing and after some time, it Again starts increasing Which reached to infinite on the other infinite end of line so distance cost curve looks like as U-curve and We have to find the bottom Value of this U-curve

→ As U-curve is not monotonically increasing or decreasing We can't use binary search for finding bottom most point, here We will use ternary search skips one third of search space at each iteration

→ So solution proceeds as follows
We start With low and high initialized as some smallest and largest values respectively then We start iteration, in each iteration We calculate two mids, mid1 and mid2 Which represent 1/3rd and 2/3rd position in search space, We calculate total distance of all points With mid1 and mid2 & update low or high by comparing these distance cost, this iteration continues until low and high become approximately equal

GoodLuck | Page No.
Date

* program

```
import math
class optimum_distance:
    class point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

class line:
    def __init(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c
def dist(self, x, y, p):
    return math.sqrt((x - p.x) ** 2 +
                     (y - p.y) ** 2)
def compute(self, p, n, l, x):
    res = 0
    y = -1 * (l.a * x + l.c) / l.b
    for i in range(n):
        res += self.dist(x, y, p[i])
    return res
def find_optimum_cost_until(self, p, n, l):
    low = -1e6
    high = 1e6
    eps = 1e-6 + 1
While ((high - low) > eps):
    mid1 = low + (high - low) / 3
    mid2 = high - (high - low) / 3
```

```
dist1 = self.compute(p, n, l, mid1)
dist2 = self.compute(p, n, l, mid2)
  if (dist1 < dist2):
      high = mid2
  else:
      low = mid1
return self.compute(p, n, l, (low + high)/2)

def find_optimum_cost(self, p, l):
    n = len(p)
    p_arr = [None] * n
  for i in range(n):
      p_obj = self.point(p[i][0], p[i][1])
      p_arr[i] = p_obj
    return self.find_optimum_cost_until(p_arr, n, l)

if __name__ == "__main__":
    obj = optimum_distance()
    l = obj.line(1, -1, -3)
  p = [ [-3, -2], [-1, 0],
        [-1, 2], [1, 2],
        [3, 4] ]
print(obj.find_optimum_cost(p, 1))
```

output

20.7652.

Algorithm:-

Step 1: Define classes

1. point class:

Represents a point in 2D space with coordinates (x, y)

2. Line class:

Represents a line of the form $ax+by+c=0$

Step 2:- Distance calculation

1. Define a function dist(x, y, p) to compute the Euclidean distance between a point (x, y) and another point $p(x, y)$:

$$d = \sqrt{(x-px)^2 + (y-py)^2}$$

Step 3:- Compute Total distance for a given x-coordina-

1. Define compute$(p, n, 1, x)$:

For a given x-coordinate, compute the corresponding y-coordinate using the line equation

$$y = -\left(\frac{a \cdot x + c}{b}\right)$$

- Calculate the sum of distances from all given points to this (x, y)

Step 4:- optimize using Ternary Search

1. Define find_optimum_cost_untill$(p, \infty, 1)$:

- initialize low $= -10^16$, high $= 10^16$ and set a small precision value eps $= 10^{-6}$

Compute two midpoints

$$mid1 = low + \left(\frac{high - low}{3}\right)$$

$$mid2 = high - \left(\frac{high - low}{3}\right)$$

- Calculate total distances

dist1 and dist2 at mid1 and mid2
- If dist1 < dist2, the minimum is in the left segment, update high = mid2
- otherwise the minimum is in the right segment update low = mid1
- continue until high - low

Step 5:- Main Function
1. Define find_optimum_cost(p, l);
- convert input list of points into point objects.
- call
find_optimum_cost_until(p, n, l)
- Return the minimum total distance.