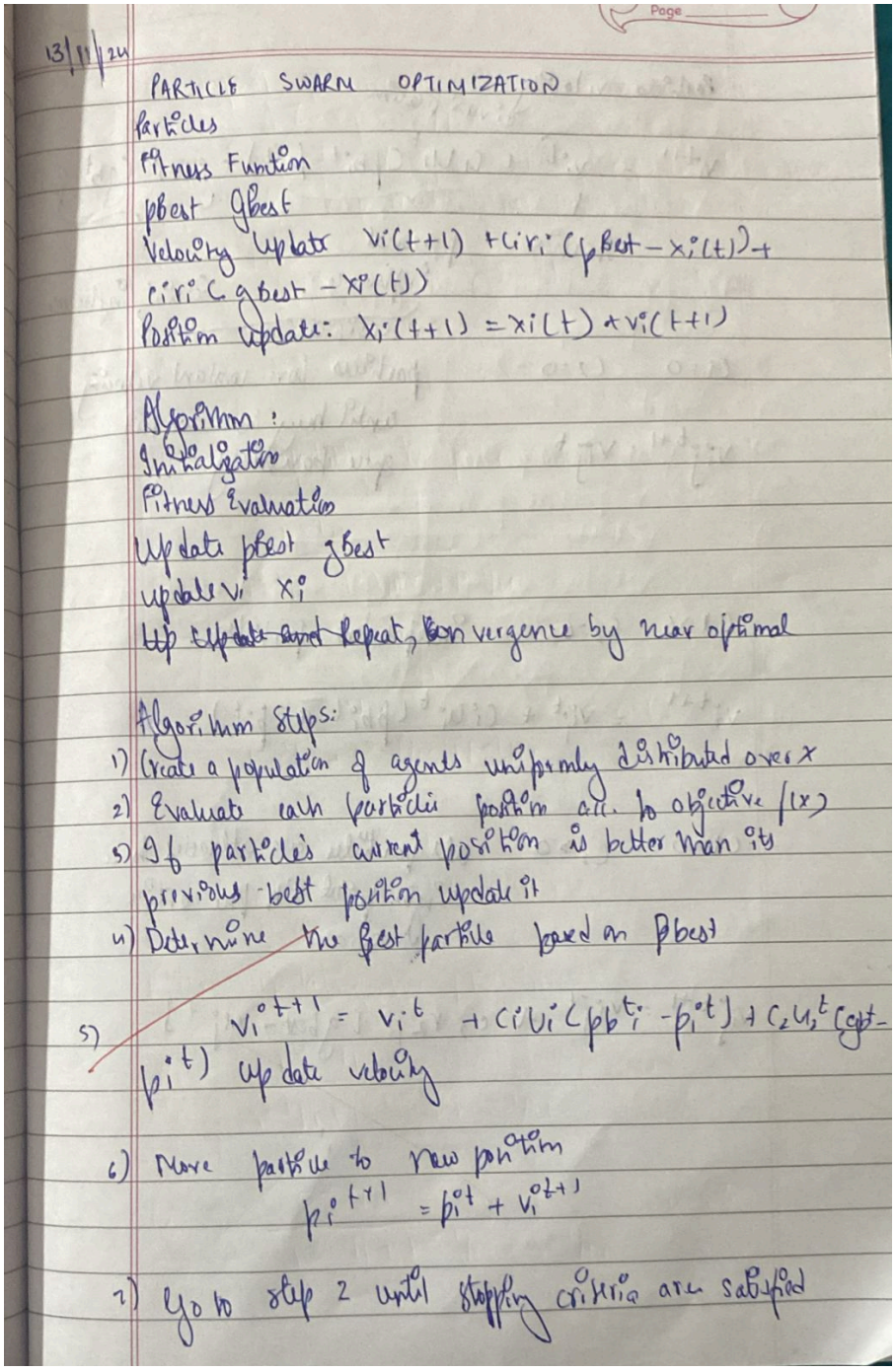
Program 3
Problem statement:Particle Swarm Optimization for Function Optimization

Algorithm:



13/11/24

PARTICLE SWARM OPTIMIZATION

Particles

Fitness Function

pbest gbest

Velocity Update $v_i(t+1) + c_i r_i (p_{best} - x_i(t)) +$
$c_i r_i (g_{best} - x_i(t))$

Position update: $x_i(t+1) = x_i(t) + v_i(t+1)$

Algorithm :

Initialization

Fitness Evaluation

Update pbest gbest

update v_i x_i

~~Update and Repeat~~, convergence by near optimal

Algorithm steps:

1) Create a population of agents uniformly distributed over x
2) Evaluate each particle position acc. to objective f(x)
3) If particle's current position is better than its previous best position update it
4) Determine the best particle based on pbest
5) $v_i^{t+1} = v_i^t + c_i v_i (pb_i^t - p_i^t) + c_2 u_i^t (gb^t - p_i^t)$ update velocity
6) Move particle to new position
$p_i^{t+1} = p_i^t + v_i^{t+1}$
7) Go to step 2 until stopping criteria are satisfied

Particle velocity

diversification

$$v_i^{t+1} = v_i^t + c_1 u_1^t (pb_i^t - p_i^t) + c_2 u_2^t (gb^t - p_i)$$

inertia — personal influence — intensification — social influence

$c_1 = 0 \quad c_2 = 0$ particles have constant velocity until they hit space boundary

$$v_{ij}^{t+1} = v_{ij}^t$$

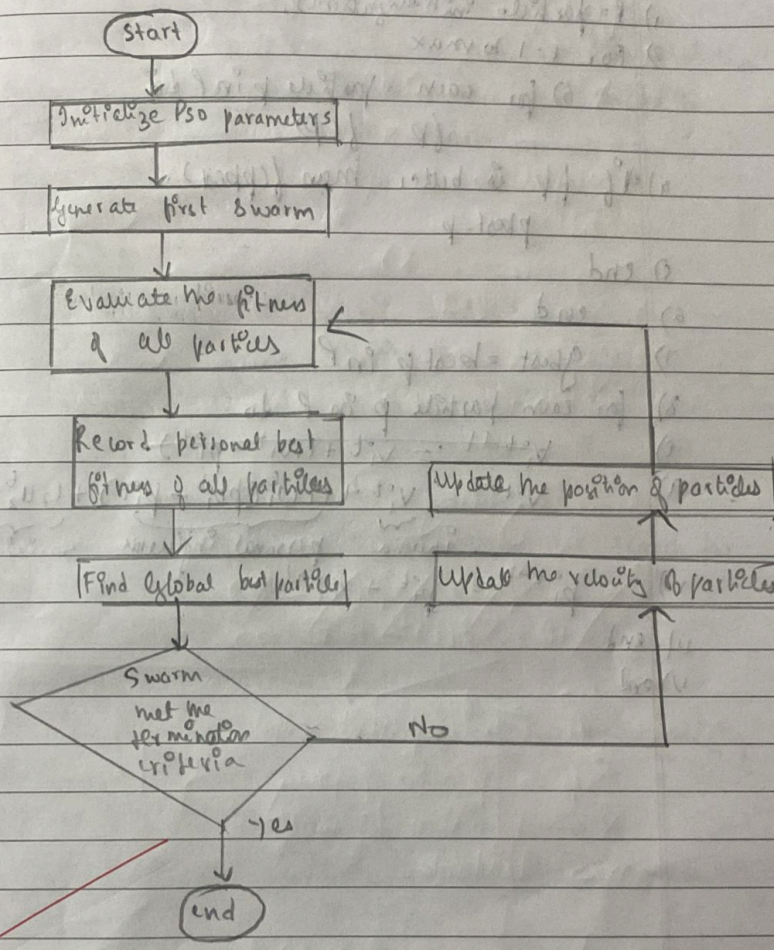
$c_1 > 0 \quad c_2 = 0$ independent particles

$$v_i^{t+1} = v_{ij}^t + c_1 u_1^t (pb_i^t - p_i^t)$$

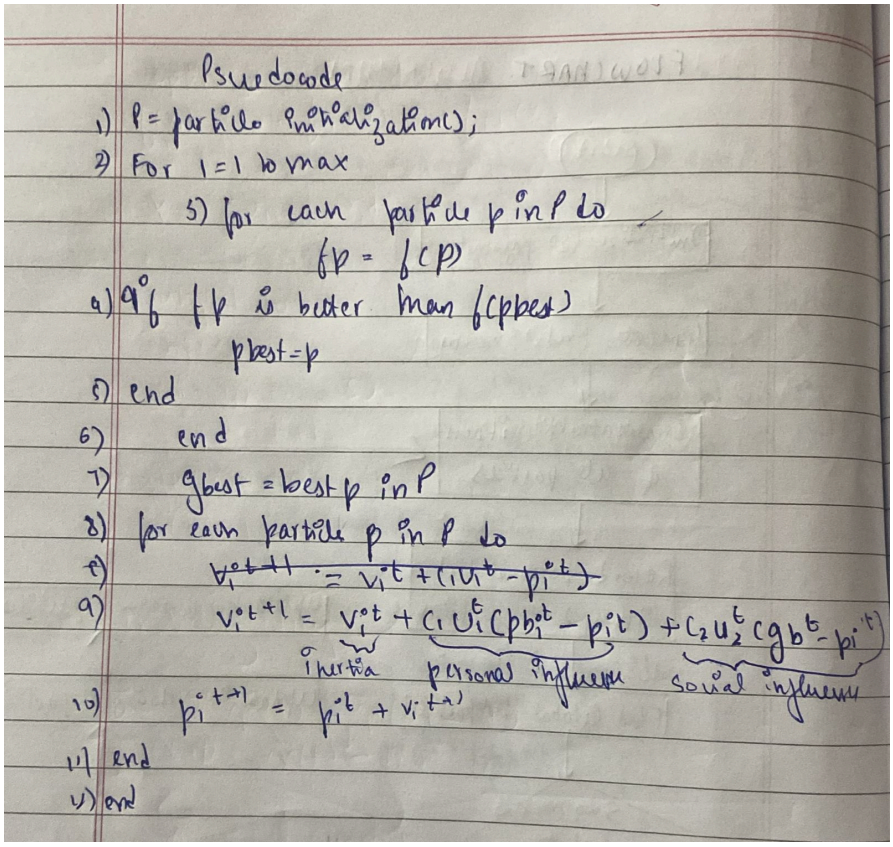
$c_1 = 0 \quad c_2 > 0$ all particles are attracted to a single point in the entire swarm

$$v_{ij}^{t+1} = v_{ij}^t + c_2 r_{2j}^t [gbest - x_{ij}^t]$$

FLOWCHART



8/4/24



Code:

```
import numpy as np
def objective_function(x):
    return np.sum(x**2)
class ParticleSwarmOptimization:
    def __init__(self, objective_function, dim, num_particles, max_iter, w=0.5, c1=1, c2=2):
        self.objective_function = objective_function
        self.dim = dim
        self.num_particles = num_particles
        self.max_iter = max_iter
        self.w = w
        self.c1 = c1
        self.c2 = c2
        self.positions = np.random.uniform(-5, 5, (num_particles, dim))
        self.velocities = np.random.uniform(-1, 1, (num_particles, dim))
        self.personal_best_positions = np.copy(self.positions)
        self.personal_best_scores = np.array([objective_function(pos) for pos in self.positions])
        self.global_best_position =
self.personal_best_positions[np.argmin(self.personal_best_scores)]
        self.global_best_score = np.min(self.personal_best_scores)
    def update_velocities(self):
        r1 = np.random.rand(self.num_particles, self.dim)
        r2 = np.random.rand(self.num_particles, self.dim)
```

```
        cognitive_component = self.c1 * r1 * (self.personal_best_positions - self.positions)
        social_component = self.c2 * r2 * (self.global_best_position - self.positions)
        self.velocities = self.w * self.velocities + cognitive_component + social_component
    def update_positions(self):
        self.positions = self.positions + self.velocities
        self.positions = np.clip(self.positions, -5, 5)
    def evaluate_particles(self):
        scores = np.array([self.objective_function(pos) for pos in self.positions])
        better_mask = scores < self.personal_best_scores
        self.personal_best_positions[better_mask] = self.positions[better_mask]
        self.personal_best_scores[better_mask] = scores[better_mask]
        best_particle = np.argmin(self.personal_best_scores)
        if self.personal_best_scores[best_particle] < self.global_best_score:
            self.global_best_position = self.personal_best_positions[best_particle]
            self.global_best_score = self.personal_best_scores[best_particle]
    def run(self):
        for i in range(self.max_iter):
            self.update_velocities()
            self.update_positions()
            self.evaluate_particles()
            print(f"Iteration {i+1}/{self.max_iter}: Global Best Score = {self.global_best_score}")
        return self.global_best_position, self.global_best_score
dim = 2
num_particles = 30
max_iter = 10
w = 0.5
c1 = 1
c2 = 2
pso = ParticleSwarmOptimization(objective_function, dim, num_particles, max_iter, w, c1, c2)
best_position, best_score = pso.run()
print(f"\nBest Position: {best_position}")
print(f"Best Score: {best_score}")
```

```python
import numpy as np
def objective_function(x):
    return np.sum(x**2)
class ParticleSwarmOptimization:
    def __init__(self, objective_function, dim, num_particles, max_iter, w=0.5, c1=1, c2=2):
        self.objective_function = objective_function
        self.dim = dim
        self.num_particles = num_particles
        self.max_iter = max_iter
        self.w = w
        self.c1 = c1
        self.c2 = c2
        self.positions = np.random.uniform(-5, 5, (num_particles, dim))
        self.velocities = np.random.uniform(-1, 1, (num_particles, dim))
        self.personal_best_positions = np.copy(self.positions)
        self.personal_best_scores = np.array([objective_function(pos) for pos in self.positions])
        self.global_best_position = self.personal_best_positions[np.argmin(self.personal_best_scores)]
        self.global_best_score = np.min(self.personal_best_scores)
    def update_velocities(self):
        r1 = np.random.rand(self.num_particles, self.dim)
        r2 = np.random.rand(self.num_particles, self.dim)
        cognitive_component = self.c1 * r1 * (self.personal_best_positions - self.positions)
        social_component = self.c2 * r2 * (self.global_best_position - self.positions)
        self.velocities = self.w * self.velocities + cognitive_component + social_component
    def update_positions(self):
        self.positions = self.positions + self.velocities
        self.positions = np.clip(self.positions, -5, 5)
    def evaluate_particles(self):
        scores = np.array([self.objective_function(pos) for pos in self.positions])
        better_mask = scores < self.personal_best_scores
        self.personal_best_positions[better_mask] = self.positions[better_mask]
        self.personal_best_scores[better_mask] = scores[better_mask]
        best_particle = np.argmin(self.personal_best_scores)
        if self.personal_best_scores[best_particle] < self.global_best_score:
            self.global_best_position = self.personal_best_positions[best_particle]
            self.global_best_score = self.personal_best_scores[best_particle]
    def run(self):
        for i in range(self.max_iter):
            self.update_velocities()
            self.update_positions()
            self.evaluate_particles()
            print(f"Iteration {i+1}/{self.max_iter}: Global Best Score = {self.global_best_score}")
        return self.global_best_position, self.global_best_score
dim = 2
num_particles = 30
max_iter = 10
w = 0.5
c1 = 1
c2 = 2
pso = ParticleSwarmOptimization(objective_function, dim, num_particles, max_iter, w, c1, c2)
best_position, best_score = pso.run()
print(f"\nBest Position: {best_position}")
print(f"Best Score: {best_score}")
```

Output:

```
Iteration 1/10: Global Best Score = 0.02723138518053337
Iteration 2/10: Global Best Score = 0.02723138518053337
Iteration 3/10: Global Best Score = 0.02723138518053337
Iteration 4/10: Global Best Score = 0.02723138518053337
Iteration 5/10: Global Best Score = 0.000857224508156451
Iteration 6/10: Global Best Score = 0.000857224508156451
Iteration 7/10: Global Best Score = 0.000857224508156451
Iteration 8/10: Global Best Score = 0.000857224508156451
Iteration 9/10: Global Best Score = 5.556248703274748e-05
Iteration 10/10: Global Best Score = 5.556248703274748e-05

Best Position: [-0.00724152 -0.00176718]
Best Score: 5.556248703274748e-05
```