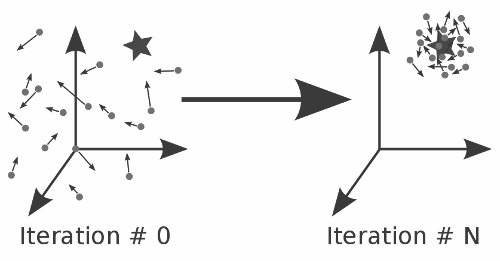
**Particle swarm optimization in python**

# ****PSO compared to Bird flocks:****

PSO can be thought of as a simulation of how a flock of birds searches for food. Imagine a flock of birds flying over a vast terrain in search of food. The birds communicate with one another by making small adjustments to their flight paths, and the entire flock moves in the direction of the food source.

Similarly, in PSO, the particles communicate with each other by adjusting their positions and velocities. The position of each particle represents a potential solution to the optimization problem, and the particles “explore” the search space by flying through it.



The particles are attracted toward the best solution found so far, which is analogous to the birds moving toward the food source. As the particles get closer to the best solution, their movement becomes more coordinated and efficient, just as the birds fly in a more synchronized manner as they get closer to the food.

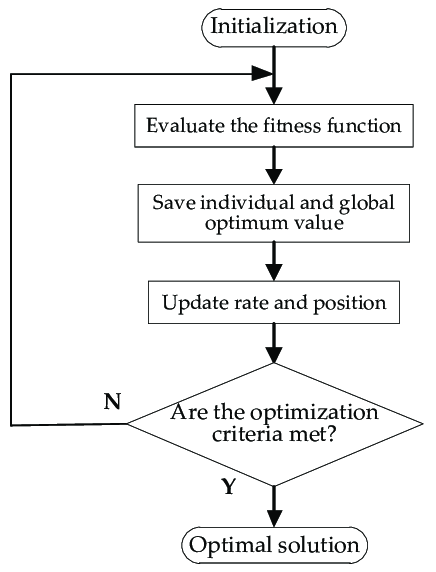
In both cases, the goal is to efficiently navigate a complex landscape in search of an optimal solution. By simulating the behavior of a flock of birds, PSO provides a powerful optimization algorithm that can efficiently search through high-dimensional search spaces to find the best solution.

# ****Introduction of PSO:****

Particle Swarm Optimization (PSO) is a population-based optimization technique that is inspired by the social behavior of birds flocking, fish schooling or insect swarming. It is a heuristic search algorithm that is used to find the optimal solution to a given problem. The algorithm starts by initializing a population of particles, where each particle represents a possible solution to the problem. Each particle moves in the search space, guided by its own experience and the experience of the swarm, trying to find the optimal solution.

The particles in the swarm move through the search space by adjusting their positions and velocities. The velocity of each particle is updated based on its current position, the best position it has ever visited, and the best position visited by the swarm. The new velocity is a weighted sum of three components: the particle’s current velocity, its best position ever visited, and the swarm’s best position ever visited. This new velocity is used to update the particle’s position.

The particles move through the search space iteratively, and each iteration is called a generation or iteration. During each generation, the fitness of each particle is evaluated, which is a measure of how good the particle’s position is in terms of the problem being solved. The fitness value is then used to update the particle’s best position and the swarm’s best position.



The PSO algorithm continues to iterate through the search space until a stopping criterion is met. The stopping criterion can be a maximum number of iterations or reaching a certain fitness value.

In summary, the PSO algorithm is a heuristic search algorithm that iteratively adjusts the positions and velocities of a population of particles to find the optimal solution to a given problem. The algorithm is guided by the experience of each particle and the swarm as a whole. It is a popular optimization algorithm due to its simplicity, efficiency, and effectiveness in solving a wide range of problems.

# ****Steps involved in PSO:****

The Particle Swarm Optimization (PSO) algorithm typically involves the following steps:

1. Initialization: Define the problem domain and initialize the population of particles (candidate solutions) randomly within the domain.
2. Evaluation: Evaluate the fitness of each particle (based on its position).
3. Update the personal best: Update the personal best position of each particle based on its fitness value.
4. Update the global best: Update the global best position based on the best position of all particles.
5. Update velocity and position: Update the velocity and position of each particle based on its current position and velocity, and the personal and global best positions.
6. Termination: Repeat steps 2–5 for a fixed number of iterations or until a stopping criterion is met (e.g. maximum number of iterations reached, target fitness value achieved, etc.).

In each iteration, the velocity of a particle is updated based on its current velocity, its personal best position, and the global best position. The position of a particle is updated based on its current position and the updated velocity. The global best position is updated if a particle has a better personal best position than the current global best position.

The PSO algorithm seeks to optimize a fitness function by adjusting the positions of the particles in the swarm. By iterating through these steps, PSO aims to find the optimal solution within the problem domain.

# Figure 1.The PSO algorithm.

# ****Implementation in python****

Here is an example of using PSO to optimize the Rastrigin function, which is a popular test function in optimization. The Rastrigin function has many local minima, making it a challenging optimization problem.

**Example PSO**

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# Define the Rastrigin function

def rastrigin(x):

n = len(x)

return 10\*n + sum([xi\*\*2 - 10\*np.cos(2\*np.pi\*xi) for xi in x])

# Define the PSO algorithm

def pso(cost\_func, dim=2, num\_particles=30, max\_iter=100, w=0.5, c1=1, c2=2):

# Initialize particles and velocities

particles = np.random.uniform(-5.12, 5.12, (num\_particles, dim))

velocities = np.zeros((num\_particles, dim))

# Initialize the best positions and fitness values

best\_positions = np.copy(particles)

best\_fitness = np.array([cost\_func(p) for p in particles])

swarm\_best\_position = best\_positions[np.argmin(best\_fitness)]

swarm\_best\_fitness = np.min(best\_fitness)

# Iterate through the specified number of iterations, updating the velocity and position of each particle at each iteration

for i in range(max\_iter):

# Update velocities

r1 = np.random.uniform(0, 1, (num\_particles, dim))

r2 = np.random.uniform(0, 1, (num\_particles, dim))

velocities = w \* velocities + c1 \* r1 \* (best\_positions - particles) + c2 \* r2 \* (swarm\_best\_position - particles)

# Update positions

particles += velocities

# Evaluate fitness of each particle

fitness\_values = np.array([cost\_func(p) for p in particles])

# Update best positions and fitness values

improved\_indices = np.where(fitness\_values < best\_fitness)

best\_positions[improved\_indices] = particles[improved\_indices]

best\_fitness[improved\_indices] = fitness\_values[improved\_indices]

if np.min(fitness\_values) < swarm\_best\_fitness:

swarm\_best\_position = particles[np.argmin(fitness\_values)]

swarm\_best\_fitness = np.min(fitness\_values)

# Return the best solution found by the PSO algorithm

return swarm\_best\_position, swarm\_best\_fitness

# Define the dimensions of the problem

dim = 2

# Run the PSO algorithm on the Rastrigin function

solution, fitness = pso(rastrigin, dim=dim)

# Print the solution and fitness value

print('Solution:', solution)

print('Fitness:', fitness)

# Create a meshgrid for visualization

x = np.linspace(-5.12, 5.12, 100)

y = np.linspace(-5.12, 5.12, 100)

X, Y = np.meshgrid(x, y)

Z = rastrigin([X, Y])

# Create a 3D plot of the Rastrigin function

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

ax.plot\_surface(X, Y, Z, cmap='viridis')

ax.set\_xlabel('x')

ax.set\_ylabel('y')

ax.set\_zlabel('z')

# Plot the solution found by the PSO algorithm

ax.scatter(solution[0], solution[1], fitness, color='red')

plt.show()