# Particle Swarm Optimization (PSO) — A Hands-On Tutorial

Advanced Algorithms

October 31, 2025

**Abstract**

This tutorial introduces Particle Swarm Optimization (PSO) with an emphasis on intuition, step-by-step development, worked numerical examples, and practical implementation guidance. The goal is to help students *understand* why PSO works, how to choose parameters, how to implement it robustly, and where to apply it. Exercises and suggestions for classroom demos are included.

## Contents

# 1 Motivation and Big Picture

Modern engineering and computer science problems often require optimizing complicated functions: tuning neural networks, scheduling, route planning, engineering design. Many such problems are **nonlinear**, **nonconvex**, or **non-differentiable** — classical gradient methods may fail or be expensive.

**Particle Swarm Optimization (PSO)** is a population-based metaheuristic inspired by natural swarm behaviours (bird flocks, fish schools). PSO is:

- Simple to implement,
- Effective for many continuous optimization problems,
- Useful as a baseline and as a building block in hybrids.

This tutorial develops PSO from first principles: the biological metaphor → mathematical model → algorithm → worked examples → practical tips.

# 2 Intuition: The Bird Flock Analogy

Imagine birds searching for food over a large field:

- No single bird knows the exact food location.
- Each bird remembers the best spot it has personally visited.
- Birds also communicate (or observe) the best spot any bird discovered.
- Each bird adjusts its flying direction using its own experience and the group's best information, while keeping some inertia.

Translating this:

- Each *particle* = candidate solution (position in search space).
- Each particle has a *velocity* = how its position changes.
- Each particle stores its personal best position (`pbest`).
- The swarm stores the global best position (`gbest`) or neighborhood best (`lbest`).
- Particles update velocities from three influences: inertia (keep going), cognitive (pull toward personal best), social (pull toward global best).

This mix balances *exploration* (search new regions) and *exploitation* (refine around known good solutions).

# 3 Mathematical Model

We now write the PSO equations used in practice.

Consider a swarm of $N$ particles searching in $D$-dimensional space. For particle $i$ at iteration $t$:

- Position: $\mathbf{x}_i(t) = [x_{i1}(t), \ldots, x_{iD}(t)]$.
- Velocity: $\mathbf{v}_i(t) = [v_{i1}(t), \ldots, v_{iD}(t)]$.
- Personal best position: $\mathbf{p}_i(t)$ (best $\mathbf{x}_i$ seen so far).
- Global best position (across swarm): $\mathbf{g}(t)$.

The standard update rules are:

$$\mathbf{v}_i(t+1) = \omega(t)\,\mathbf{v}_i(t) + c_1\,r_1(t) \odot \big(\mathbf{p}_i(t) - \mathbf{x}_i(t)\big) + c_2\,r_2(t) \odot \big(\mathbf{g}(t) - \mathbf{x}_i(t)\big),$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1).$$

Here:

- $\omega(t)$ is the inertia weight (may change with time).
- $c_1$ (cognitive) and $c_2$ (social) are positive constants.
- $r_1(t)$ and $r_2(t)$ are vectors of independent Uniform$(0, 1)$ random numbers (one per dimension).
- $\odot$ denotes elementwise multiplication.

## Interpretation of terms

- **Inertia term** $\omega\mathbf{v}_i$: maintains momentum (helps exploration).
- **Cognitive term** $c_1 r_1(\mathbf{p}_i - \mathbf{x}_i)$: personal experience — pull towards the particle's best.
- **Social term** $c_2 r_2(\mathbf{g} - \mathbf{x}_i)$: social learning — pull towards swarm best.

# 4 Step-by-Step PSO Algorithm (with Rationale)

Below is the canonical PSO algorithm with explanations of each step.

## Algorithm (high-level)

1. **Initialize:** For each particle $i$, randomly set $\mathbf{x}_i(0)$ inside the search bounds, and set $\mathbf{v}_i(0)$ to small random values. Evaluate fitness $f(\mathbf{x}_i(0))$. Set $\mathbf{p}_i(0) = \mathbf{x}_i(0)$. Set $\mathbf{g}(0)$ to the best $\mathbf{p}_i$.
2. **Repeat** for $t = 0, \ldots, t_{\max} - 1$:
   (a) For each particle $i$:

- Update velocity $\mathbf{v}_i(t+1)$ using the PSO equation.
- Optionally clamp each $v_{id}$ to $[-v_{\max,d}, v_{\max,d}]$.
- Update position $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1)$.
- If $\mathbf{x}_i(t+1)$ is outside bounds, handle it (reflect, clamp, or reinitialize).
- Evaluate $f(\mathbf{x}_i(t+1))$.
- If $f(\mathbf{x}_i(t+1))$ is better than $f(\mathbf{p}_i(t))$, set $\mathbf{p}_i(t+1) = \mathbf{x}_i(t+1)$, else $\mathbf{p}_i(t+1) = \mathbf{p}_i(t)$.

(b) Update global best $\mathbf{g}(t+1)$ as the best of $\{\mathbf{p}_i(t+1)\}$.

(c) Optionally adapt parameters (e.g., decrease inertia $\omega$).

3. **Stop** when maximum iterations or convergence criterion is reached. Return $\mathbf{g}$.

### Why these steps?

- **Random initialization** spreads particles to explore diverse regions.
- **Velocity update** combines exploration (inertia) and exploitation (cognitive/social pulls).
- **Velocity clamping** prevents particles from jumping too far and missing good areas.
- **Position bounds handling** ensures feasible solutions.
- **Parameter adaptation** (e.g., reducing $\omega$) gradually shifts from exploration to exploitation.

## 5 Worked Numerical Example (1-D) — Build it by Hand

A tiny example helps to make the algebra tangible.

### Problem:

Minimize $f(x) = x^2$ over $[-10, 10]$. Use 3 particles and 1D PSO.

### Initial setup:

- Particle 1: $x_1(0) = 5.0, v_1(0) = 0.2$. $p_1 = 5.0$, $f(p_1) = 25$.
- Particle 2: $x_2(0) = -3.0, v_2(0) = -0.1$. $p_2 = -3.0$, $f(p_2) = 9$.
- Particle 3: $x_3(0) = 2.0, v_3(0) = 0.05$. $p_3 = 2.0$, $f(p_3) = 4$.
- Global best $g = 2.0$ (since $f(2) = 4$ is smallest).
- Choose $\omega = 0.7$, $c_1 = c_2 = 1.5$.

### Iteration 1 — particle 1 update (demonstration)

Take particle 1:

$$v_1(1) = 0.7 \cdot 0.2 + 1.5 \cdot r_1(0) \cdot (5.0 - 5.0) + 1.5 \cdot r_2(0) \cdot (2.0 - 5.0).$$

Assume random draws $r_1 = 0.4$, $r_2 = 0.3$. The cognitive term is zero because $p_1 = x_1$. Compute:

$$v_1(1) = 0.14 + 1.5 \cdot 0.3 \cdot (-3.0) = 0.14 - 1.35 = -1.21.$$

Then

$$x_1(1) = 5.0 + (-1.21) = 3.79.$$

Evaluate $f(3.79) = 14.36$ which improves $p_1$ (from 25 to 14.36). So set $p_1 = 3.79$. The global best may now become $p_3 = 2.0$ still (since 4 ¡ 14.36).

### Observation

After just one update, particle 1 moved significantly toward the better region near zero. Repeating this for all particles shows how particles converge toward the minimum.

# 6 Worked Example (2-D, visualizable) — Rosenbrock or simple quadratic

Students should implement PSO for a 2D function like $f(x, y) = x^2 + y^2$ and visualize particle trajectories. This demonstrates swarm motion and convergence in the plane — a powerful classroom demo.

# 7 Practical Implementation Guidelines

Below are pragmatic details and tips for robust PSO implementations.

## 7.1 Parameter choices

- **Population size $N$:** 20–50 for many problems. Smaller for fast prototyping.
- **Inertia $\omega$:** Start 0.9 and reduce to 0.4 linearly; common schedule:

$$\omega(t) = \omega_{\max} - \frac{\omega_{\max} - \omega_{\min}}{T} t.$$

- **Cognitive / Social constants:** $c_1 = c_2 = 1.5$ or 2.0 are standard. Higher $c_2$ favors social learning.
- **Velocity clamp $v_{\max}$:** e.g., 10–20% of the search range per dimension.
- **Iterations $T$:** depends on problem; run until stagnation or $T$ reached.

### 7.2 Velocity clamping

Clamping velocity reduces overshooting. Per dimension:

$$v_{id} \leftarrow \text{sign}(v_{id}) \min(|v_{id}|, v_{\max,d}).$$

### 7.3 Boundary handling

When $x_{id}$ leaves bounds $[L_d, U_d]$:

- *Clamp:* set $x_{id} = L_d$ or $U_d$ and $v_{id} = 0$.
- *Reflect:* $x_{id} = L_d + (L_d - x_{id})$, $v_{id} = -v_{id}$.
- *Wrap-around:* for periodic domains.

Choose based on problem semantics.

### 7.4 Stopping criteria

Common choices:

- Fixed max iterations.
- No improvement in $\mathbf{g}$ for $k$ iterations.
- Fitness threshold (target value reached).

### 7.5 Initialization strategies

- **Uniform random** within bounds is standard.
- **Latin hypercube sampling** for better space coverage when dimensions are large.
- Initialize velocities small: $v_{id} \in [-0.1\,\Delta_d, 0.1\,\Delta_d]$ where $\Delta_d = U_d - L_d$.

## 8 Variants and Extensions

### 8.1 Local best vs Global best

- **Global best (gbest):** all particles use the same $\mathbf{g}$. Faster convergence but risk of premature stagnation.
- **Local best (lbest):** each particle uses the best position found in its neighborhood (ring, random groups). Encourages diversity and avoids premature convergence — better for multimodal problems.

### 8.2 Constriction factor

An alternative update uses a constriction coefficient $\chi$ for stability:

$$v(t+1) = \chi\Big(v(t) + c_1 r_1 (p - x) + c_2 r_2 (g - x)\Big),$$

with $\chi$ chosen based on $c_1 + c_2$ (see Clerc  Kennedy). Helps with convergence control.

### 8.3 Binary and discrete PSO

PSO can be adapted to binary spaces (feature selection) by using a sigmoid on velocity and sampling the position bit:

$$S(v_{id}) = \frac{1}{1 + e^{-v_{id}}}, \quad x_{id} = \begin{cases} 1 & \text{with prob } S(v_{id}) \\ 0 & \text{otherwise} \end{cases}$$

Care is required to interpret velocity meaningfully.

### 8.4 Hybrid PSO

Combine PSO with local search (e.g., hillclimbing or 2-opt for TSP). Often PSO finds good regions; local search refines solutions to high quality.

## 9 Theoretical Remarks (brief)

PSO is heuristic — there is no universal proof of global convergence in finite time. However:

- Under certain parameter choices and stochastic assumptions, particles converge in distribution.
- Empirically PSO is robust for many practical problems.

For B.Tech level, emphasize empirical understanding and parameter sensitivity rather than deep proofs.

## 10 Use Cases and Examples

- **Function optimization:** multimodal functions, hyperparameter tuning.
- **Engineering design:** mechanical design parameter optimization.
- **Neural networks:** PSO for weight initialization or direct weight optimization.
- **Feature selection:** binary PSO selects subsets of features.
- **Control / Robotics:** path planning and PID parameter tuning.

## 11 Implementation — Pseudocode and Python Skeleton

### Pseudocode

```
Initialize swarm positions x[i] and velocities v[i]
For i in 1..N:
    pbest[i] = x[i]
    pbest_fitness[i] = f(x[i])
```

```
gbest = argmin_i pbest_fitness[i]

for t = 1..T:
    for i = 1..N:
        generate r1, r2 in [0,1]^D
        v[i] = w*v[i] + c1*r1*(pbest[i]-x[i]) + c2*r2*(gbest-x[i])
        clamp v[i] to [-vmax, vmax]
        x[i] = x[i] + v[i]
        enforce bounds on x[i]
        fval = f(x[i])
        if fval < pbest_fitness[i]:
            pbest[i] = x[i]; pbest_fitness[i]=fval
    update gbest as best pbest
    optionally update w
end
return gbest
```

## Python skeleton

```python
import numpy as np

def pso(f, dim, bounds, num_particles=30, max_iter=200):
    # bounds: list of (low, high) per dimension
    x = np.random.uniform(
        [b[0] for b in bounds],
        [b[1] for b in bounds],
        (num_particles, dim)
    )
    v = np.zeros_like(x)
    pbest = x.copy()
    pbest_f = np.array([f(xi) for xi in x])
    gbest = pbest[np.argmin(pbest_f)].copy()

    w_max, w_min = 0.9, 0.4
    c1, c2 = 1.5, 1.5
    vmax = 0.2 * (np.array([b[1] - b[0] for b in bounds]))

    for t in range(max_iter):
        w = w_max - (w_max - w_min) * (t / max_iter)
        for i in range(num_particles):
            r1 = np.random.rand(dim)
            r2 = np.random.rand(dim)
            v[i] = w*v[i] + c1*r1*(pbest[i] - x[i]) + c2*r2*(
               gbest - x[i])
            # clamp velocity
            v[i] = np.clip(v[i], -vmax, vmax)
            x[i] = x[i] + v[i]
            # clamp position
            for d in range(dim):
```

```
                x[i,d] = np.clip(x[i,d], bounds[d][0], bounds[d
                    ][1])
            fval = f(x[i])
            if fval < pbest_f[i]:
                pbest[i] = x[i].copy()
                pbest_f[i] = fval
                if fval < f(gbest):
                    gbest = x[i].copy()
    return gbest
```

Listing 1: Simple PSO skeleton (students can extend)

# 12 Classroom Exercises and Projects

1. **Implement PSO** for $f(x, y) = x^2 + y^2$ and plot particle trajectories (2D).
2. **Parameter study:** Vary $\omega$, $c_1$, $c_2$ and observe convergence speed and solution quality across runs. Report mean and variance.
3. **Compare PSO vs GA:** Solve a benchmark function (Rastrigin, Rosenbrock). Plot convergence curves and discuss strengths/weaknesses.
4. **Hybrid project:** Combine PSO with local search (hillclimbing) for TSP or small combinatorial problem (use discrete encoding).
5. **Binary PSO**: Implement binary PSO for a feature selection task on a small dataset; evaluate classifier accuracy vs number of features.

# 13 Common Pitfalls and Tips

- **Premature convergence:** If swarm converges too fast, increase inertia or use local best topology.
- **Divergence / oscillations:** Reduce $v_{\max}$ or adjust $\omega$ and $c$ constants.
- **Scaling issues:** Normalize variables to similar ranges — differing scales break PSO behavior.
- **Random seed management:** For reproducibility, fix random seeds when benchmarking.

# 14 Summary and Teaching Notes

PSO is intuitively attractive and a useful tool for engineers. For B.Tech students:

- Emphasize the **physical intuition** (inertia, cognition, social).
- Walk through a small numeric example in class (1D or 2D).
- Have students implement a simple PSO and visualize trajectories.
- Use PSO to compare algorithmic design choices (topology, inertia schedule).

# References and Further Reading


- J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of IEEE International Conference on Neural Networks, 1995.
- M. Clerc and J. Kennedy, "The particle swarm—explosion, stability, and convergence in a multidimensional complex space," IEEE Trans. on Evolutionary Computation, 2002.
- R. Eberhart, P. Simpson, and R. Dobbins (eds.), *Computational Intelligence PC Tools*, Academic Press, 1996.
- Online resources: PSO tutorials, TSPLIB, benchmark functions (Rastrigin, Rosenbrock).


**Author's note:** encourage students to play — PSO is best learned by experimenting with different problems, parameters, and visualizations.