MATH529 – Fundamentals of Optimization
Derivative-Free and Direct Search Methods

Marco A. Montes de Oca

Mathematical Sciences, University of Delaware, USA

A review of our assumptions

- Twice continuously differentiable objective functions
- Differentiable constraints

A review of our assumptions

- Twice continuously differentiable objective functions
- Differentiable constraints

What about ...

Estimating derivatives numerically

Since $\frac{df(t)}{dt} = \lim_{h \to 0} \frac{f(t+h) - f(t)}{h}$, we can estimate derivatives at t_0 with

$$\frac{df(t_0)}{dt} \approx \frac{f(t_0 + h) - f(t_0)}{h}$$

with h small. For estimating the gradient of $f : \mathbb{R}^n \to \mathbb{R}$ at t_0, we would need $n + 1$ function evaluations.

The error associated with this calculation is proportional to h, but it cannot be zero in a digital computer. (A reasonable choice for h is $\approx 1\mathrm{e} - 08$.)

But sometimes

- Evaluating f is extremely time-consuming.
- Calculating f at a point may require real-world data (e.g., real experiments).
- f may be calculated with software we don't have access to (i.e., it's a black box).
- f is unpredictable (maybe it has discontinuities).
- f may be noisy.

Derivative-Free Methods

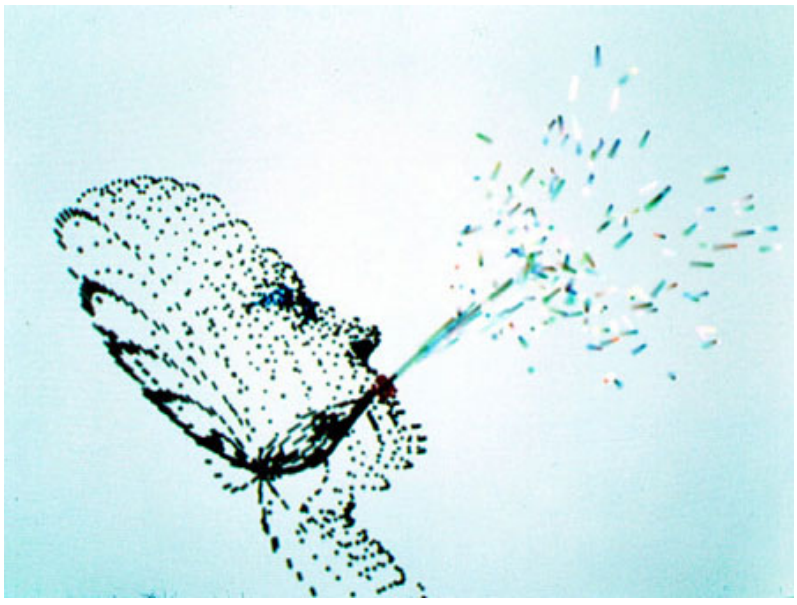
There are two flavors:

- Model-based: Build explicitly, or implictly, a model of f (e.g, a quadratic model by interpolation or regression), then minimize the model and hope for the best.
- Model-free: Make no assumptions about f. (These methods are called **Direct-Search Methods**, or **Black-box Methods**.)

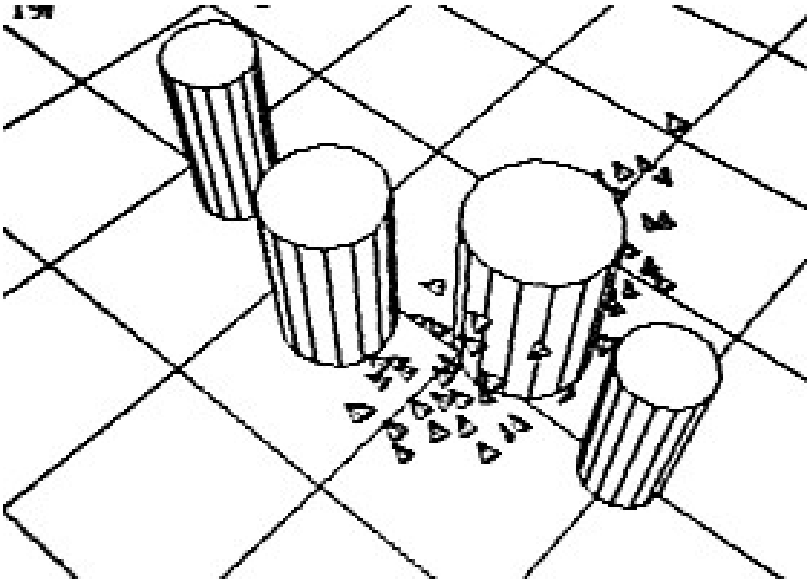
One direct search method I am particulary familiar with:

Particle Swarm Optimization

Particle swarm optimization: Origins



Particle swarm optimization: Origins



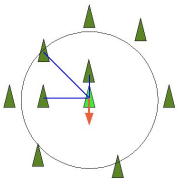
Particle swarm optimization: Origins

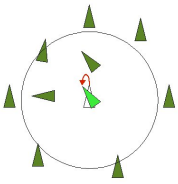
Reynolds proposed a behavioral model in which each agent follows three rules:

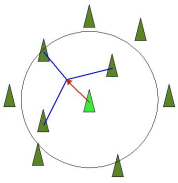
Separation. Each agent tries to move away from its neighbors if they are too close.

Alignment. Each agent steers towards the average heading of its neighbors.

Cohesion. Each agent tries to go towards the average position of its neighbors.



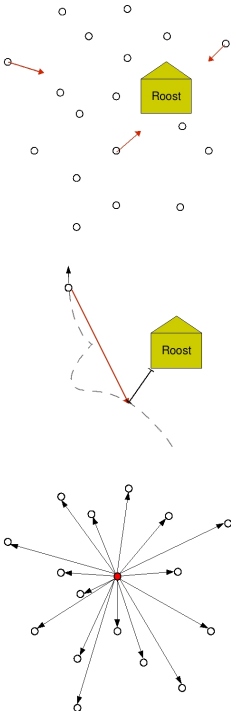




Particle swarm optimization: Origins

Kennedy and Eberhart included a 'roost' in a simplified Reynolds-like simulation so that:

- Each agent was attracted towards the location of the roost.
- Each agent 'remembered' where it was closer to the roost.
- Each agent shared information with its neighbors (originally, all other agents) about its closest location to the roost.



Particle swarm optimization: The idea

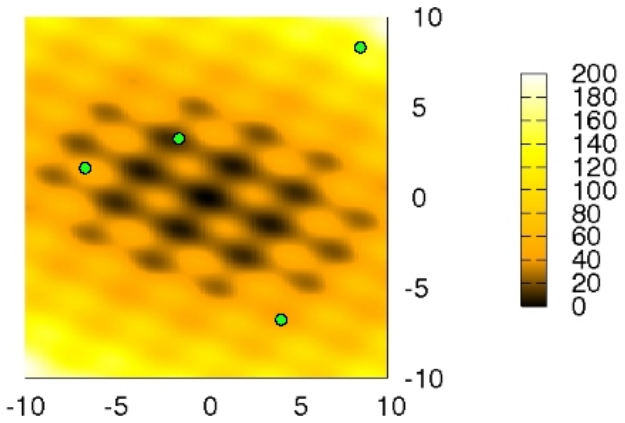
Eventually, all agents 'landed' on the roost.

Particle swarm optimization: The idea

What if the notion of distance to the roost is changed by an objective function to optimize? Will the agents 'land' in the minimum?

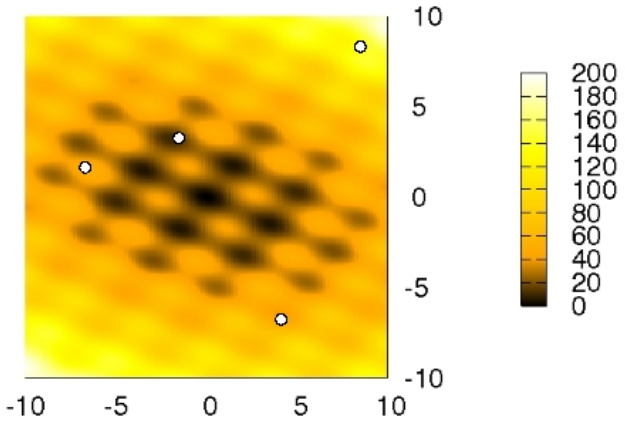
Particle swarm optimization: The basic algorithm

1. Create a 'population' of agents (called *particles*) uniformly distributed over $\mathcal{X}$.



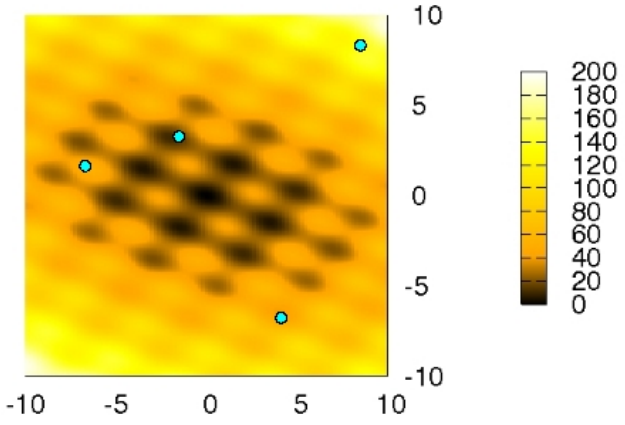
Particle swarm optimization: The basic algorithm

2. Evaluate each particle's position according to the objective function.



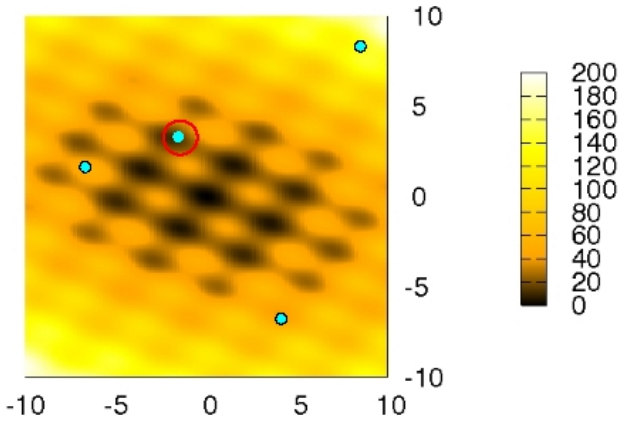
Particle swarm optimization: The basic algorithm

3. If a particle's current position is better than its previous best position, update it.



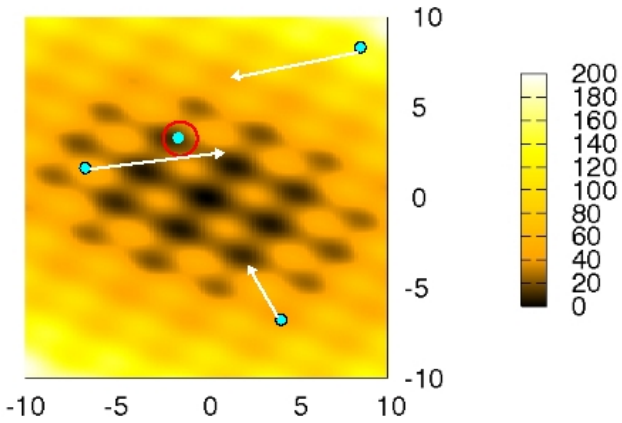
Particle swarm optimization: The basic algorithm

4. Determine the best particle (according to the particle's previous best positions).



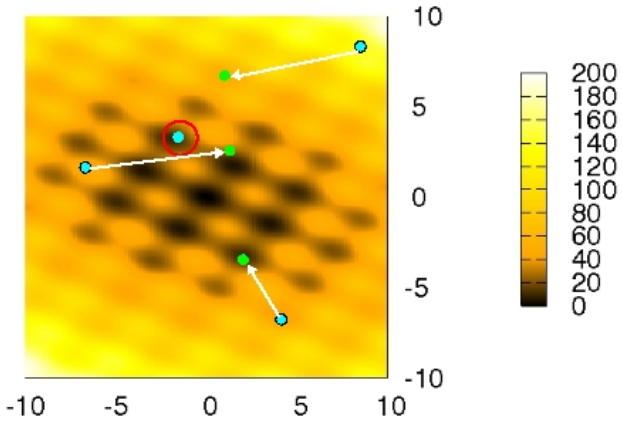
Particle swarm optimization: The basic algorithm

5. Update particles' velocities according to
$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2 \mathbf{U}_2^t(\mathbf{gb}^t - \mathbf{x}_i^t)$.



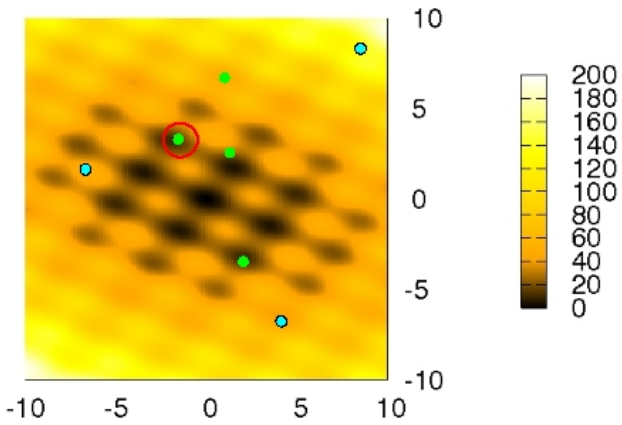
Particle swarm optimization: The basic algorithm

6. Move particles to their new positions according to $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}$.



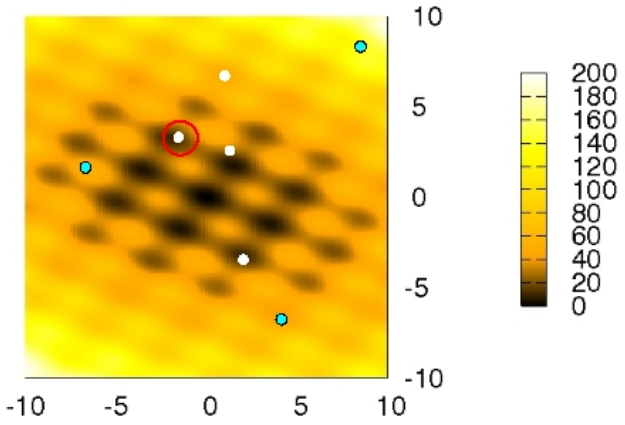
Particle swarm optimization: The basic algorithm

7. Go to step 2 until stopping criteria are satisfied.



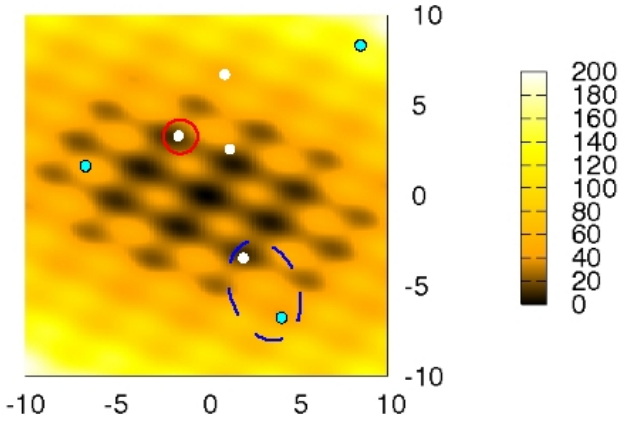
Particle swarm optimization: The basic algorithm

2. Evaluate each particle's position according to the objective function.



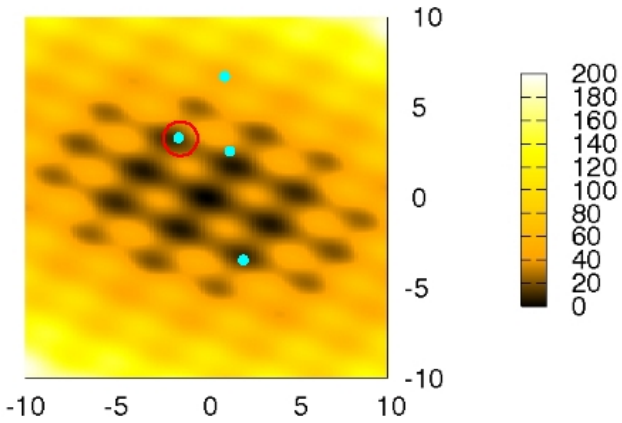
Particle swarm optimization: The basic algorithm

3. If a particle's current position is better than its previous best position, update it.



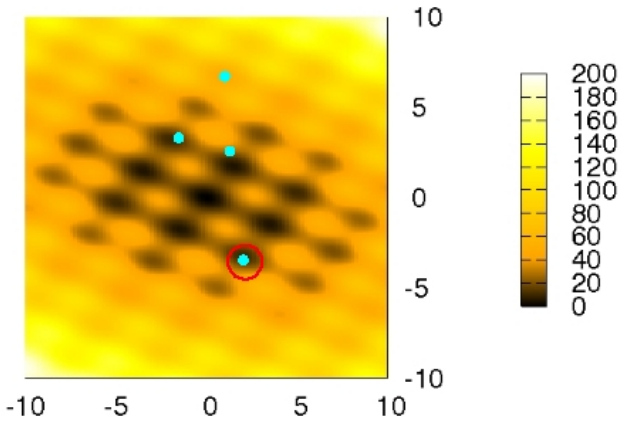
Particle swarm optimization: The basic algorithm

3. If a particle's current position is better than its previous best position, update it.



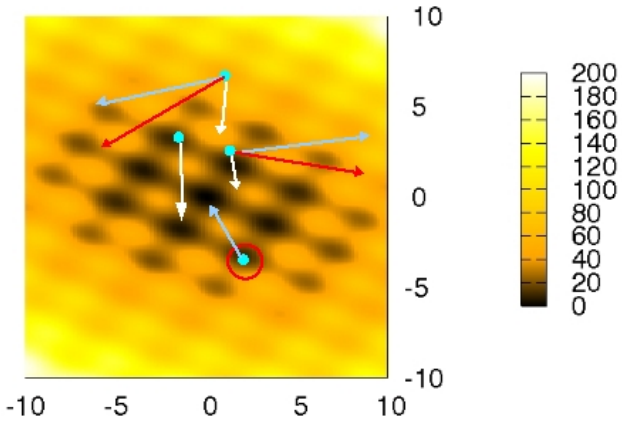
Particle swarm optimization: The basic algorithm

4. Determine the best particle (according to the particle's previous best positions).



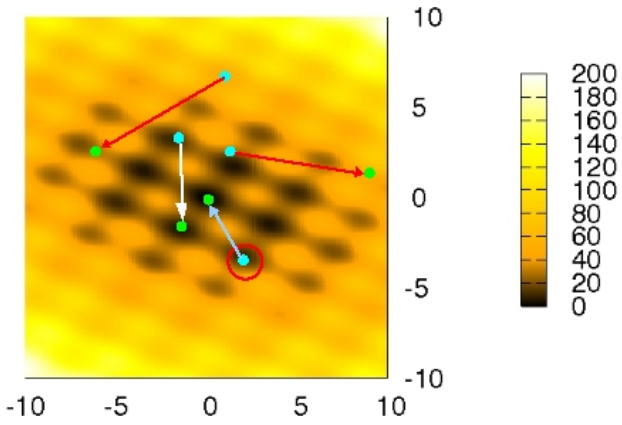
Particle swarm optimization: The basic algorithm

5. Update particles' velocities according to
$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2 \mathbf{U}_2^t(\mathbf{gb}^t - \mathbf{x}_i^t)$.



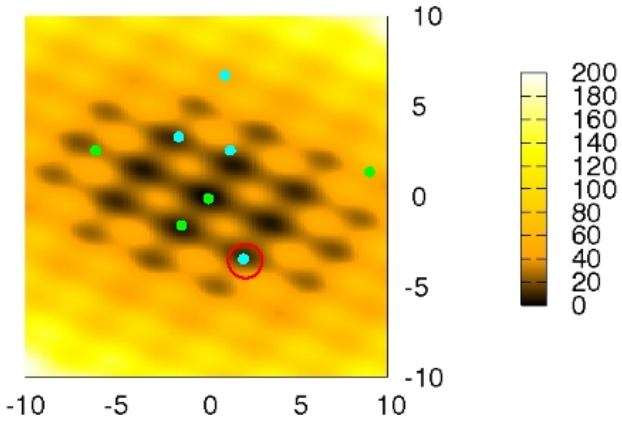
Particle swarm optimization: The basic algorithm

6. Move particles to their new positions according to
$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}$.



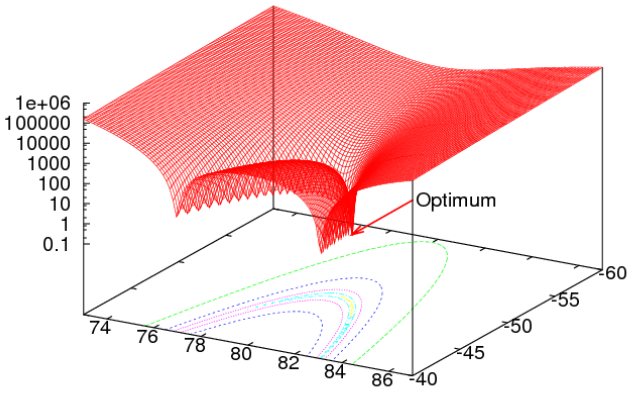
Particle swarm optimization: The basic algorithm

7. Go to step 2 until stopping criteria are satisfied.



Particle swarm optimization: Example

Rosenbrock function



Particle swarm optimization: The basic algorithm in action

Particle swarm optimization: Main modifications and variants

Almost all modifications vary in some way the velocity-update rule:

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2 \mathbf{U}_2^t(\mathbf{gb}^t - \mathbf{x}_i^t)$$

Particle swarm optimization: Main modifications and variants

Almost all modifications vary in some way the velocity-update rule:

$$\mathbf{v}_i^{t+1} = \underbrace{\mathbf{v}_i^t}_{\textit{inertia}} + \varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2 \mathbf{U}_2^t(\mathbf{gb}^t - \mathbf{x}_i^t)$$

Particle swarm optimization: Main modifications and variants

Almost all modifications vary in some way the velocity-update rule:

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \underbrace{\varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t)}_{\textit{personal influence}} + \varphi_2 \mathbf{U}_2^t(\mathbf{gb}^t - \mathbf{x}_i^t)$$

Particle swarm optimization: Main modifications and variants

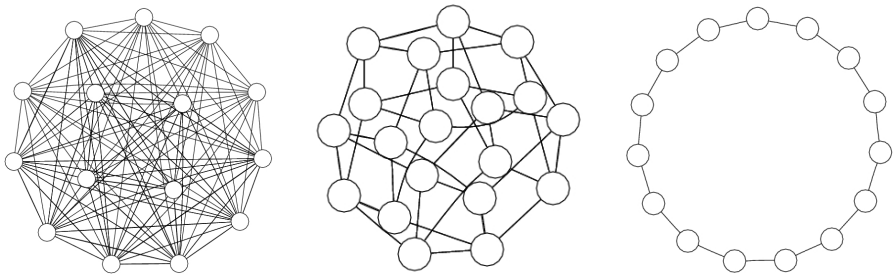
Almost all modifications vary in some way the velocity-update rule:

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \underbrace{\varphi_2 \mathbf{U}_2^t(\mathbf{gb}^t - \mathbf{x}_i^t)}_{\textit{social influence}}$$

Particle swarm optimization: Different population topologies

Every particle i has a neighborhood $N_i \subset \mathcal{P}$, where $\mathcal{P}$ is the set of particles.

$$\mathbf{v}_i^{t+1} = \mathbf{v}_i^t + \varphi_1 \mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2 \mathbf{U}_2^t(\mathbf{lb}_i^t - \mathbf{x}_i^t)$$



The idea was introduced by Kennedy and Eberhart [**?**].

Particle swarm optimization: Inertia weight

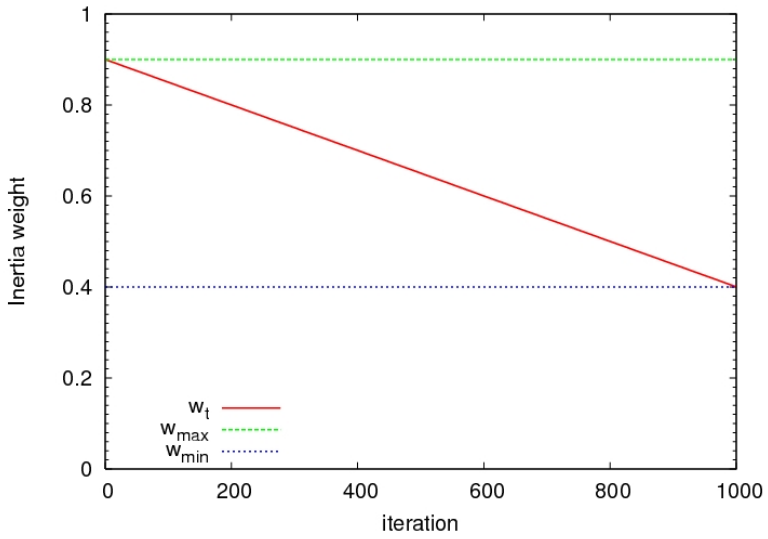
It adds a parameter called *inertia weight* so that the modified rule is:

$$\mathbf{v}_i^{t+1} = w\mathbf{v}_i^t + \varphi_1\mathbf{U}_1^t(\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2\mathbf{U}_2^t(\mathbf{lb}_i^t - \mathbf{x}_i^t)$$

It was proposed by Shi and Eberhart [**?**].

Particle swarm optimization: Time-decreasing inertia weight

The value of the inertia weight is decreased during a run



It was proposed by Shi and Eberhart [**?**].

Particle swarm optimization: Constriction Factor

It is a special case of the inertia weight variant derived from:

$$\mathbf{v}_i^{t+1} = \chi \left[\mathbf{v}_i^t + \varphi_1 \mathbf{U}_1^t (\mathbf{pb}_i^t - \mathbf{x}_i^t) + \varphi_2 \mathbf{U}_2^t (\mathbf{lb}_i^t - \mathbf{x}_i^t) \right] ,$$

where χ is called a "constriction factor" and is fixed.

It has been very influential after its publication by Clerc and Kennedy in 2002 [**?**].

Particle swarm optimization: Parameter selection

Consider a one-particle one-dimensional particle swarm. This particle's velocity-update rule is

$$v^{t+1} = a v^t + b_1 U_1^t (pb^t - x^t) + b_2 U_2^t (gb^t - x^t)$$

Particle swarm optimization: Parameter selection

Additionally, if we make

$$E[U_*^t(0,1)] = \frac{1}{2}\,,$$

$$b = \frac{b_1 + b_2}{2}\,,$$

$$pb^{t+1} = pb^{t+1}\,, gb^{t+1} = gb^t\,,$$

and

$$r = \frac{b_1}{b_1 + b_2} pb^t + \frac{b_2}{b_1 + b_2} gb^t\,.$$

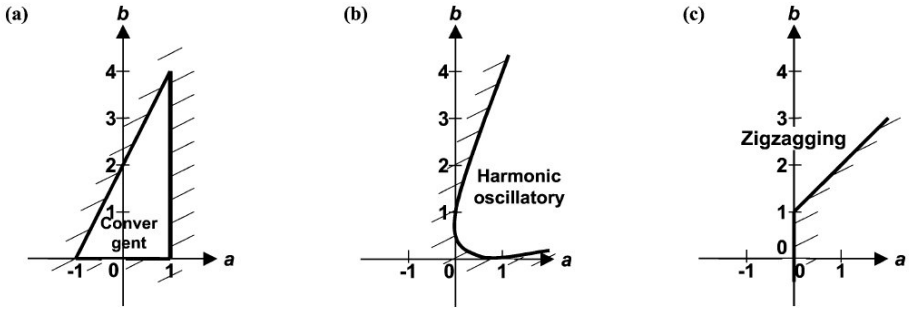
Particle swarm optimization: Parameter selection

Then, we can say that

$$v^{t+1} = av^t + b(r - x^t)\,.$$

Particle swarm optimization: Parameter selection

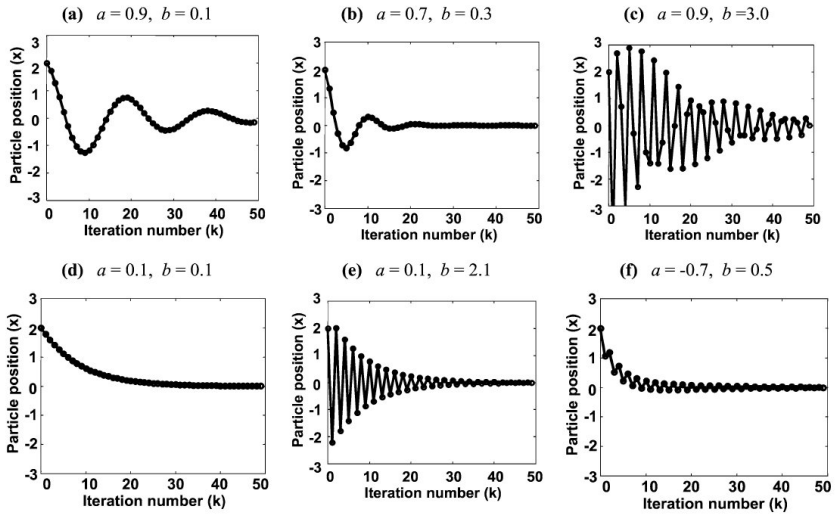
It can be shown that this system will behave in different ways depending on the value of a, b.



Graph taken from Trelea [?].

Particle swarm optimization: Parameter selection

Some examples



Graph taken from Trelea [?].

Particle swarm optimization: Parameter selection

Factors to consider when choosing a particular variant and/or a parameter set:

- The characteristics of the problem ("modality", search ranges, dimension, etc)
- Available search time (wall clock or function evaluations)
- The solution quality threshold for defining a satisfactory solution