Particle Swarm Optimization (1)

Winter 2011

Learning from Creatures

Swarm Intelligence

- Swarm of creatures could achieve things that no individual could
  - No central control
  - **Cooperation** and **self-organization**
  - Simple, and yet powerful rule for individual

Introduction

- First described in 1995 by James Kennedy and Russell Eberhart
- Implement the underlying rules that enable large numbers of organisms (birds, fishes, herds) to move synchronously, often changing direction suddenly, scattering and regrouping
- Attractiveness: simplicity, implement in little code; few parameters to adjust

PSO Theory (1)

- Each particle iteratively moves across the search space
- Attracted to the position (location) of the best fitness (evaluation of the objective function) historically achieved by the particle itself (local best; pBest) and by the best among the neighbors of the particle (global best; gBest).

PSO Theory (2)

- In essence, each particle continuously focuses and refocuses the effort of its search according to both local and global best.
- This behavior mimics the cultural adaptation of a biological agent in a swarm: it evaluates its own position based on certain fitness criteria, compares with others, and imitates the best in the entire swarm

Basic PSO Flow (1)

- Initialize a population of particles in the **search space**
  - Provide each particle with a random location and a random velocity within the N-dimensional space
- Evaluate each particle's fitness, based on the **objective value**
- If the fitness is better than the particle's best experience (pBest), save the location vector for the particle as pBest

Basic PSO Flow (2)

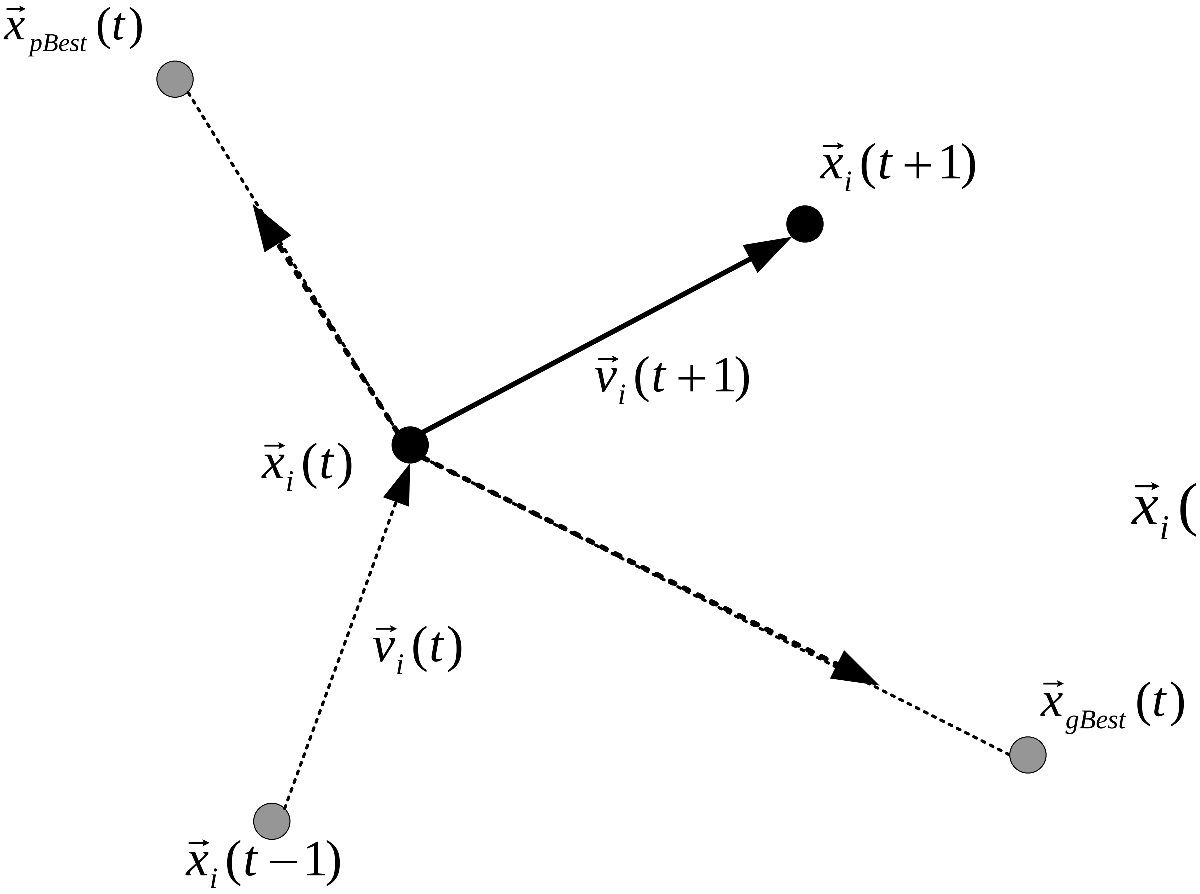
- If the fitness is better than the best in the entire population (gBest), save the location vector for the particle as gBest
- Update the particle's velocity and location based on pBest and gBest

Update the Velocity

$$\vec{v}_i(t+1) = w \times \vec{v}_i(t) + \underline{r_1 c_1 (\vec{x}_{pBest} - \vec{x}_i(t))} + \underline{r_2 c_2 (\vec{x}_{gBest} - \vec{x}_i(t))}$$

- w is an inertia weight to control influence of the previous velocity; usually between 0 and 1
- r_1 and r_2 are two random numbers uniformly distributed in the range of (0,1);
- c_1 and c_2 are two acceleration constants; usually between 1~4.

Update the Location



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

Further Discussion

- PSO models the "learning from own experience (cognition)" behavior of individual particles whereas reflects the "social interaction" phenomenon among particles.
- These two terms are stochastically weighted by r_1c_1 and r_2c_2. The randomness provides the possibility to reach the global optimal solution without being trapped at local optimal solutions (exploration).

PSO in Pseudocode (1)

```
FOR each particle
    Initialize particle randomly
WHILE maximum iteration or convergence criteria is not met
    FOR each particle
        Calculate fitness value $F(i,t)$ corresponded to location $X(i,t)$
          % $i,t$: particle $i$ at iteration $t$
        IF $F(i,t)$ is better than pbest
            pBest = $F(i,t)$
            pBestLoc = $X(i,t)$        → $\vec{x}_{pbest}$
        END
    END
```

PSO in Pseudocode (2)

```
    Choose the particle $i*$ with the best fitness
        gBest = F($i*$,$t$)
        gBestLoc = X($i*$,$t$)          →  $\vec{x}_{gbest}$
    FOR each particle
        Calculate particle velocity
        Update particle location
    END
END WHILE
Return gBest
```

Numerical Example

- Maximize $f(x,y) = x^{0.5} - 1/y; 0 \leqq (x,y) \leqq 5$
- We will use this example to illustrate a PSO algorithm

Numerical Example (1)

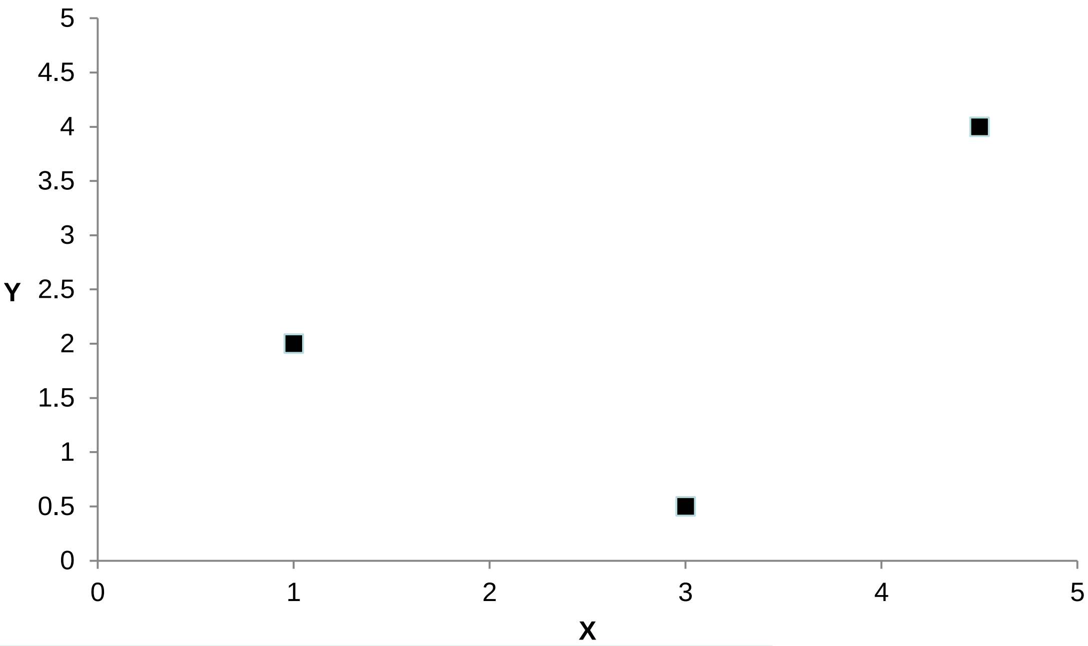
- Randomly initialize locations and velocities

Location

x	y
1	2
4.5	4
3	0.5

Velocity

x	y
0.83	-0.51
-0.67	0.94
0.45	0.89



Assume the initial velocity is randomly generated within range (-1,1)

Numerical Example (2)

Iteration 1: Calculate Fitness

Location

x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

Numerical Example (3)

Iteration 1: Update pBest and pBestLoc

This is the first iteration, so pBest will be the current fitness

pBest

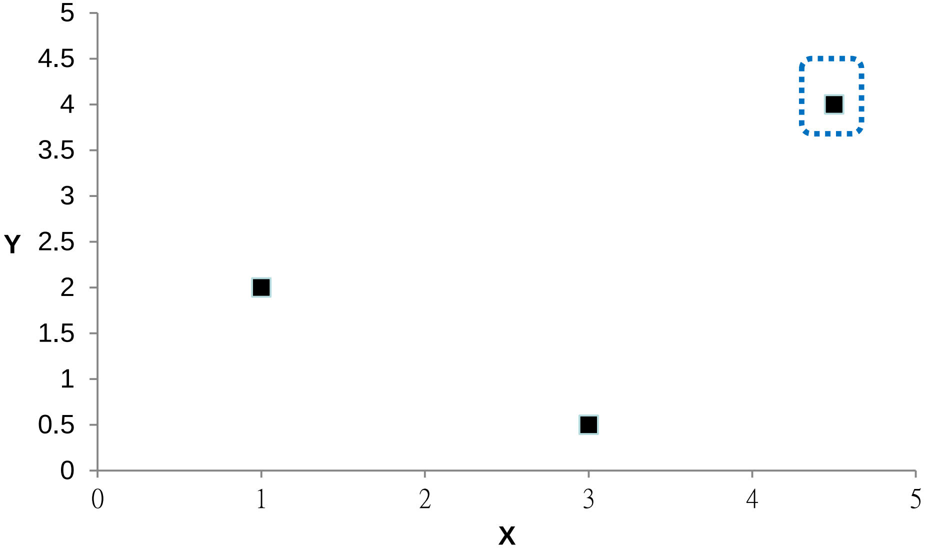
x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

Numerical Example (4)

Iteration 1: Determine gBest and gBestLoc

gBest

x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268



Numerical Example (5)

Iteration 1: Calculate velocities

Velocity

x	y		
3.49	1.01	--->	$\overset{w}{1}\times\overset{\textit{previous velocity}}{(0.83,-0.51)} + r_1\times\overset{c_1}{2}\times(0,0) + \overset{r_2}{0.38}\times\overset{c_2}{2}\times[(4.5,4)-(1,2)]$
-0.67	0.94	--->	global best; $1\times(-0.67,0.94)$
1.86	4.18	--->	$1\times(0.45,0.89) + r_1\times2\times(0,0) + 0.47\times2\times[(4.5,4)-(3,0.5)]$

Location

x	y
1	2
4.5	4
3	0.5

Velocity

x	y
0.83	-0.51
-0.67	0.94
0.45	0.89

pBest

x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

gBest

x	y	Fitness
4.5	4	1.871

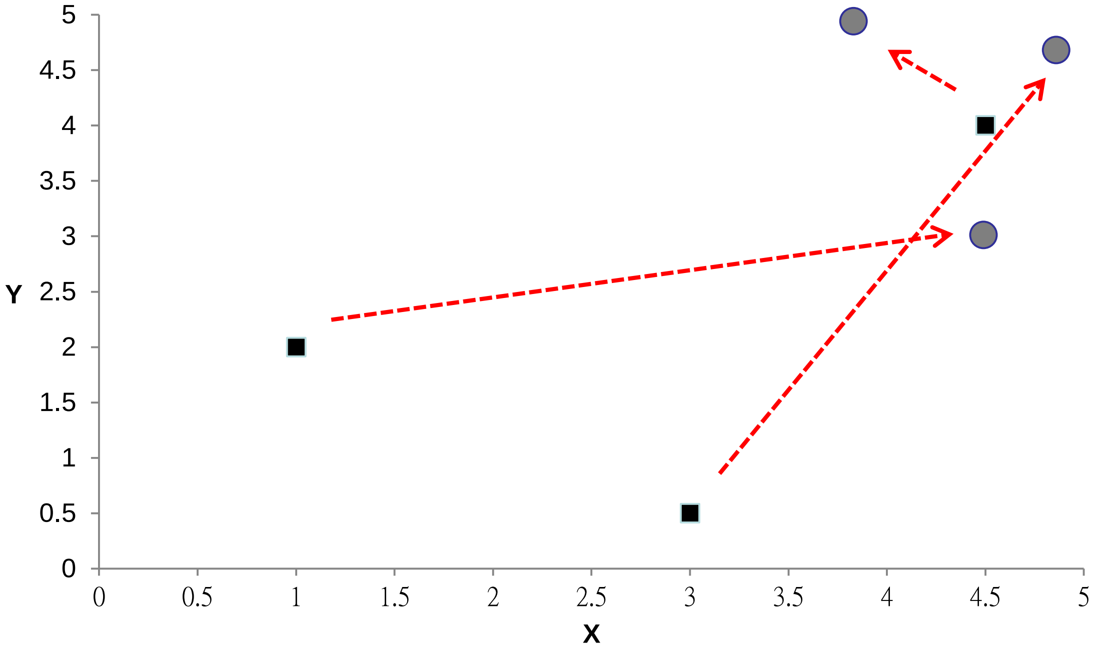
Numerical Example (6)

Iteration 1: Calculate locations

New Location			Location			Velocity	
x	**y**		**x**	**y**		**x**	**y**
4.49	3.01	=	1	2	+	3.49	1.01
3.83	4.94		4.5	4		-0.67	0.94
4.86	4.68		3	0.5		1.86	4.18

Numerical Example (7)

New Location			Location			Velocity	
x	y		x	y		x	y
4.49	3.01		1	2		3.49	1.01
3.83	4.94	=	4.5	4	+	-0.67	0.94
4.86	4.68		3	0.5		1.86	4.18



Numerical Example (8)

Iteration 2: Calculate fitness

Location

x	y	Fitness
4.49	3.01	1.787
3.83	4.94	1.755
4.86	4.68	1.991

Numerical Example (9)

Iteration 2: Update pBest and pBestLoc

Original pBest

x	y	Fitness
1	2	0.500
4.5	4	1.871
3	0.5	-0.268

Location

x	y	Fitness
4.49	3.01	1.787
3.83	4.94	1.755
4.86	4.68	1.991

New pBest

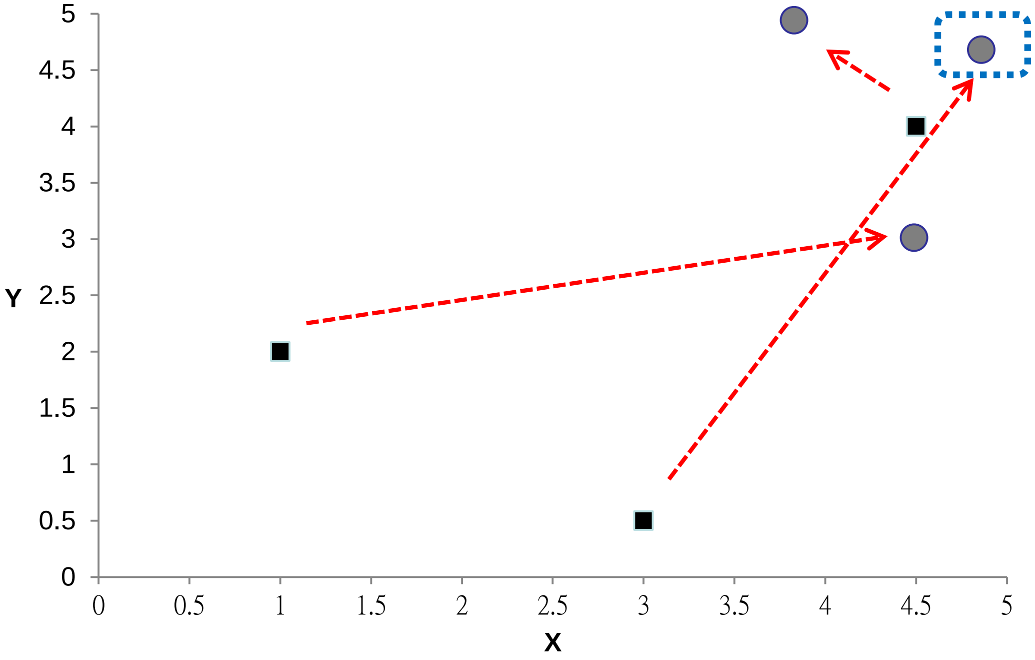
x	y	Fitness
4.49	3.01	1.787
4.5	4	1.871
4.86	4.68	1.991

Numerical Example (10)

Iteration 2: Determine gBest and gBestLoc

gBest

x	y	Fitness
4.49	3.01	1.787
4.5	4	1.871
4.86	4.68	1.991



Constrained Velocity and Location

- If the velocity would lead the particle to move beyond boundaries, what should we do?
- First, the velocity may be limited
- Second, whether the location is allowed to move outside the boundary needs special caution

Damping Limit for Velocity

- To prevent undesired oscillations (i.e., the velocity may expand wider and wider until approaching infinity), the velocity is often dampened by an upper limit $v^{\max}$

$$v(i,t) = \frac{v(i,t)}{\left|v(i,t)\right|} v^{\max} \quad \text{if } \left|v(i,t)\right| > v^{\max}$$

$v^{\max}$ is smaller than the domain of the search space

Treatment for Boundary Violation

- Do not allow any particle to move outside, infeasible solutions
- Force particle to stick to the boundary; repair the location if necessary
- There are some other alternatives; we will address them next time
- Let’s continue iteration 2 of the numerical example to repair the location

Numerical Example (11)

Recap

Velocity

x	y
3.49	1.01
-0.67	0.94
1.86	4.18

Location

x	y	Fitness
4.49	3.01	1.787
3.83	4.94	1.755
4.86	4.68	1.991

pBest

x	y	Fitness
4.49	3.01	1.787
4.5	4	1.871
4.86	4.68	1.991

gBest

x	y	Fitness
4.86	4.68	1.991

Numerical Example (12)

Iteration 2: Calculate velocities

Velocity

$$\vec{v}_i(t+1) = w \times \vec{v}_i(t) + r_1 c_1 (\vec{x}_{pBest} - \vec{x}_i(t)) + r_2 c_2 (\vec{x}_{gBest} - \vec{x}_i(t))$$

x	y		
4.038	3.482	--->	1×(3.49,1.01) + 0.17×2×[(4.49,3.01)-(4.49,3.01)] + 0.74×2×[(4.86,4.68)-(4.49,3.01)]
0.151	0.286	--->	1×(-0.67,0.94) + 0.29×2×[(4.5,4)-(3.83,4.94)] + 0.21×2×[(4.86,4.68)-(3.83,4.94)]
1.860	4.180	--->	1×(1.86,4.18) + 0.68×2×[(4.86,4.68)-(4.86,4.68)] + 0.11×2×[(4.86,4.68)-(4.86,4.68)]

Numerical Example (13)

Iteration 2: Velocity damping; e.g., $v^{\max} = 2$

Velocity		Damped Velocity	
x	y	x	y
4.038	3.482	2	2
0.151	0.286	0.151	0.286
1.860	4.180	1.860	2

Numerical Example (14)

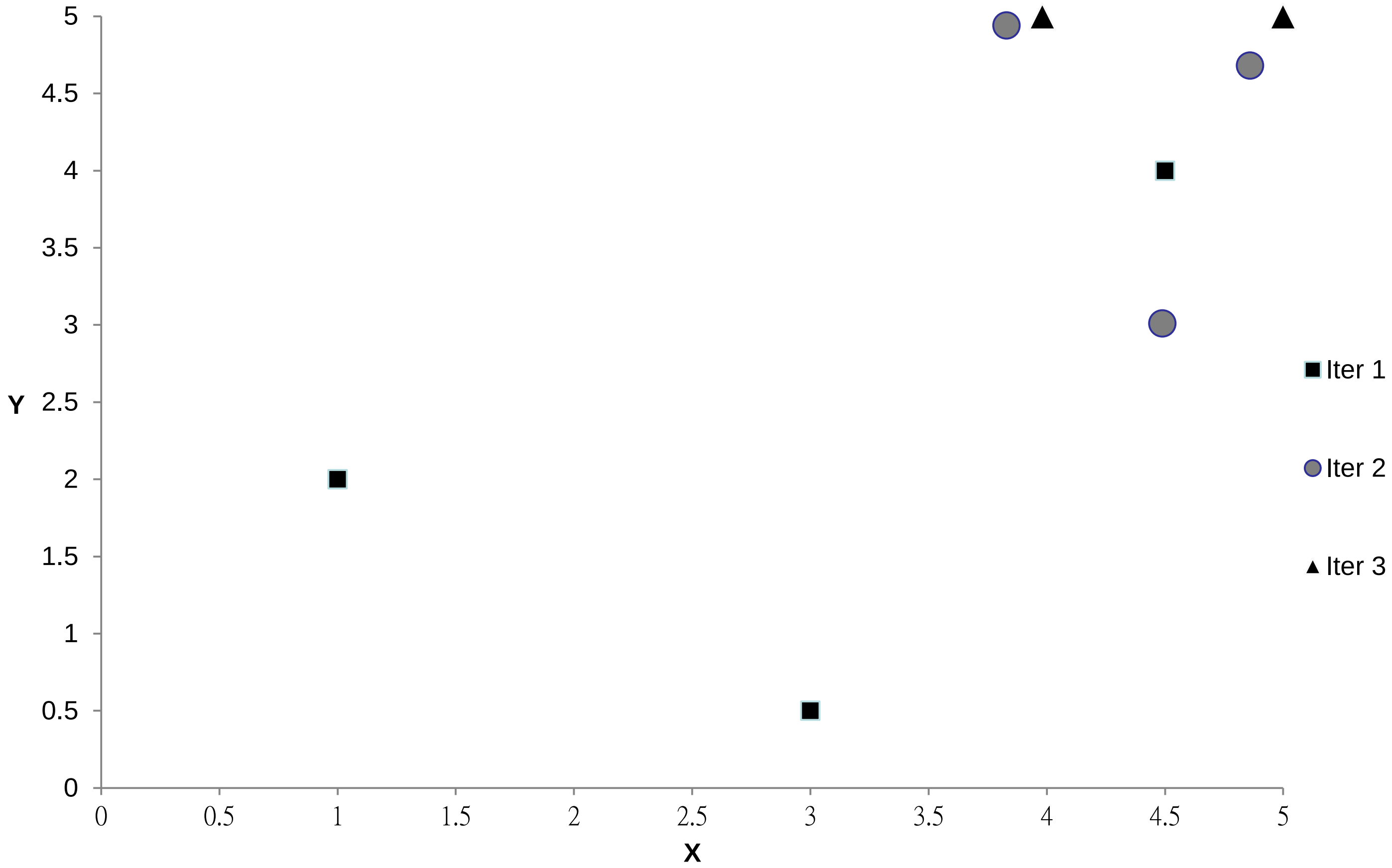
Iteration 2: Repair locations, e.g., ≦5

New Location			Location			Velocity	
x	y		x	y		x	y
6.490	5.01	=	4.49	3.01	+	2	2
3.981	5.226		3.83	4.94		0.151	0.286
6.720	6.680		4.86	4.68		1.860	2

Repaired Location

x	y
5.000	5.000
3.981	5.000
5.000	5.000

Numerical Example (15): End of Iteration 2



Still need velocity limit?

- If we will constrain the location after all, why do we need to set the damping limit for velocity?
- Because it will influence the velocity in the subsequent iterations