Dimensions, Distance and Optimization

Natural language and ARtificial intelligence Group

Warning

This is meant to be helpful, fun and a bit silly.
This is not meant to make you feel patronized.
Don't take it like that.

Dimensions,
Not what you think

“ Dimension of a space is the minimum number of coordinates needed to describe a point in that space. ”

0-D

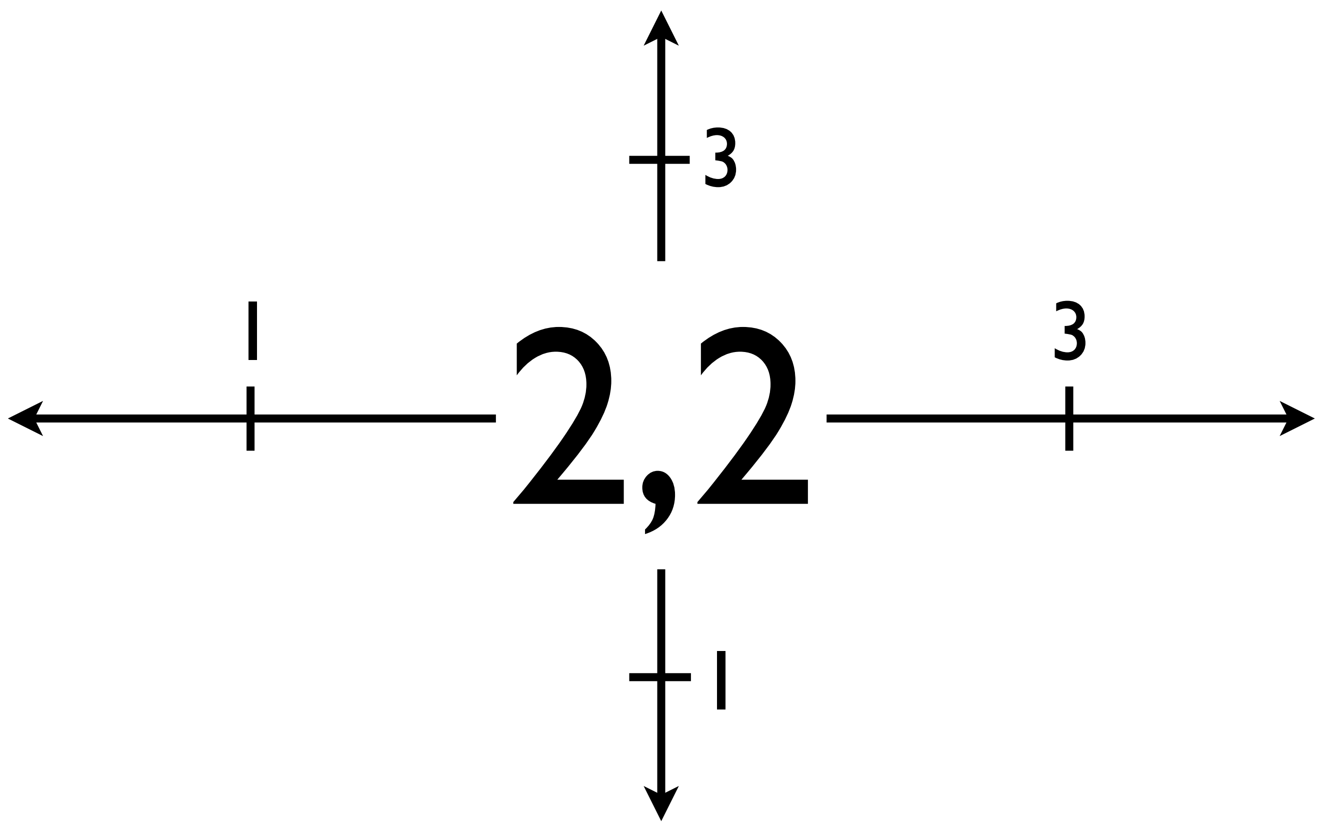
2

Dimensionless number, a number without a parent space
Famous example: Reynolds Number

1-D

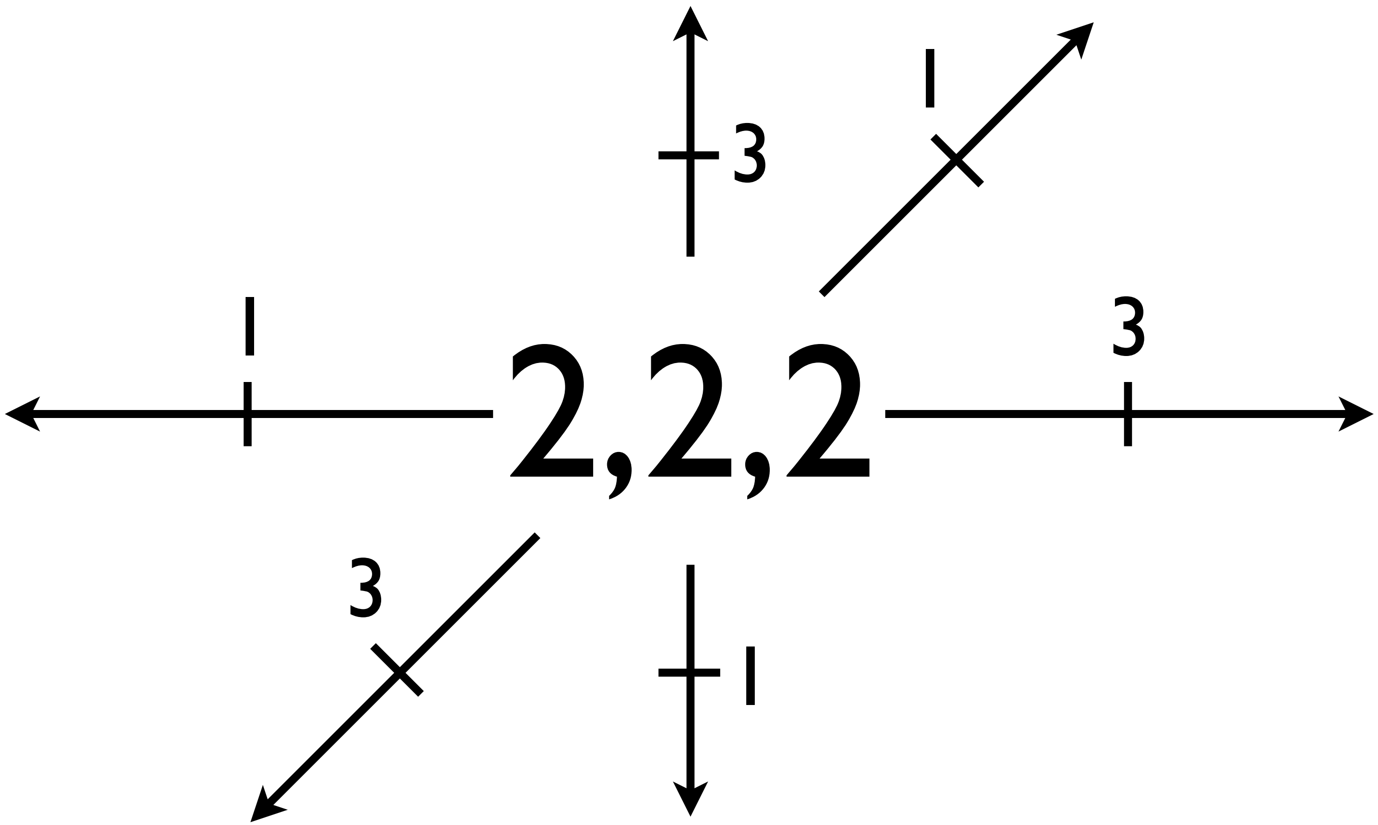
Line, a space that takes 1 coordinate to define a point

2-D



Plane, a space that takes 2 coordinates to define a point

3-D



Volume, a space that takes 3 coordinates to define a point

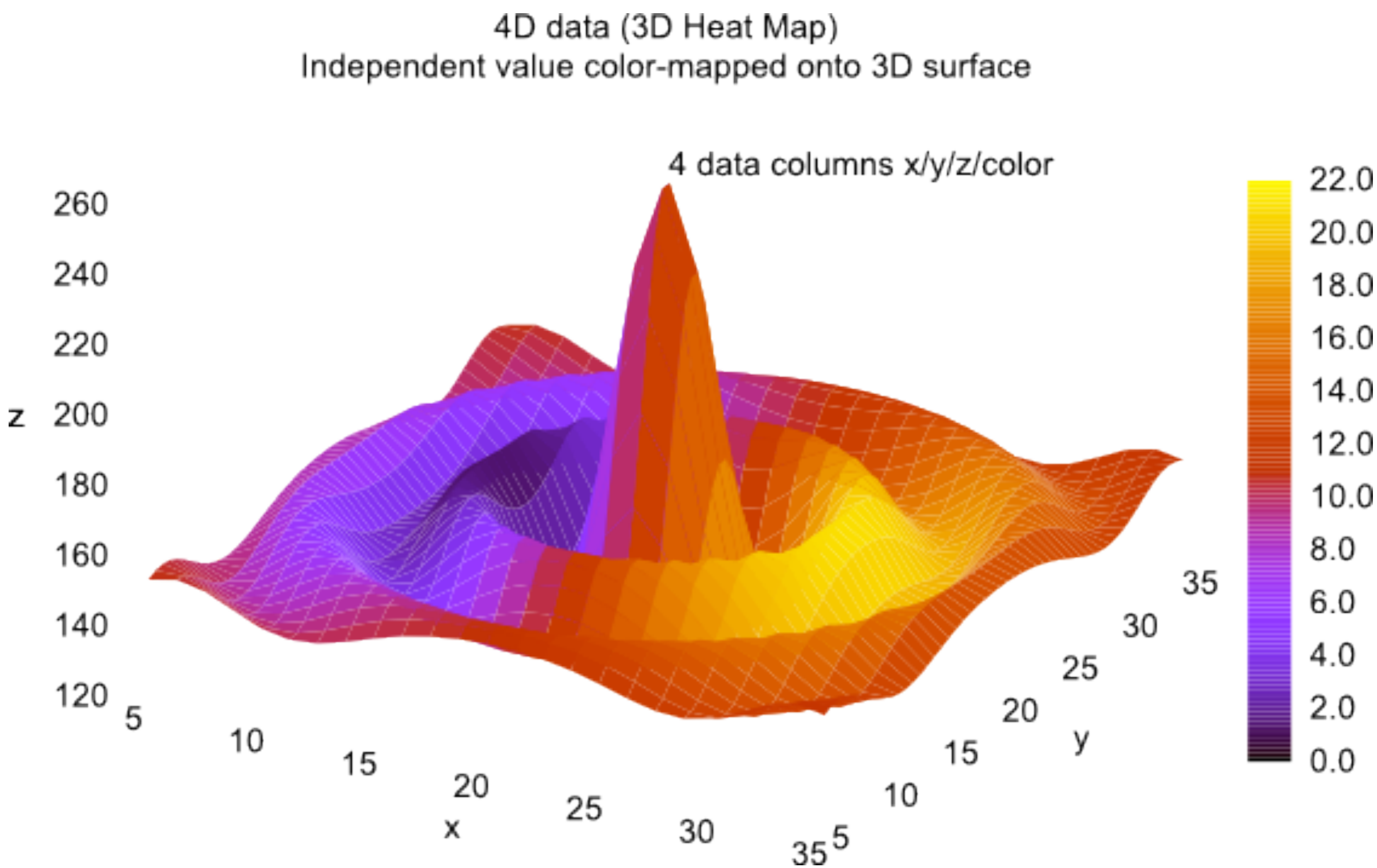
4-D

THE 4TH DIMENSION IS NOT TIME

It is the dimension after 3 before 5

Spaces larger than 3 dimensions are generally referred to as "Hyper-Volume" or "N-Dimensional Space" or "N-Space"

Visualizing 4-D



Visualizing N-D

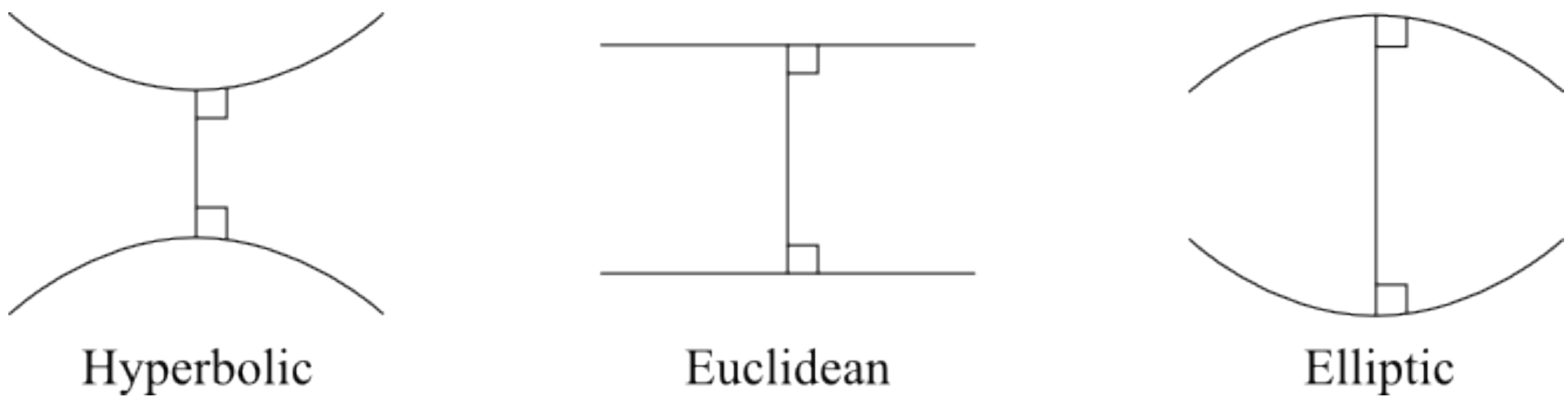
- Last graph we had was plotting a space using (x, y, z, color)
- We could use (x, y, z, r) for 4-D
- (x, y, z, r, g) for 5-D
- (x, y, z, r, g, b) for 6-D
- Even (x, y, z, r, g, b, a) for 7-D
- Beyond that you need to be creative

Shapes of Spaces

- There are many different shapes of space
- These spaces can be enumerated by fucking with Euclid's Fifth Postulate

"If a line bisects another line at an angle less than 90 degrees the lines **will** intersect at some point"

Various Space Shapes



All of these lines are parallel (even if they don't look it)

Assumptions for NARG

- Assume that all space is always Euclidean (It’s hard to play in spaces other than Euclidean, so we won’t)
- Euclidean Distance means we are taking the distance between two points on a space that has Euclidean shape
(i.e. Euclid's 5th Postulate holds true)

Distance Metrics

- Distance metrics are one of the best ways of measuring similarity
- Similarity is good to know for most AI and ML applications
    - Optimization Algorithms
    - Reinforcement Learning
    - Etc.

My Favorite Distances

- Hamming Distance - Number of swaps
- Euclidean Distance - Normal distance
- Manhattan Distance - Number of “blocks”

Hamming Distance

Kitten → Kitteh

Hamming Distance: 1

11101 → 10011

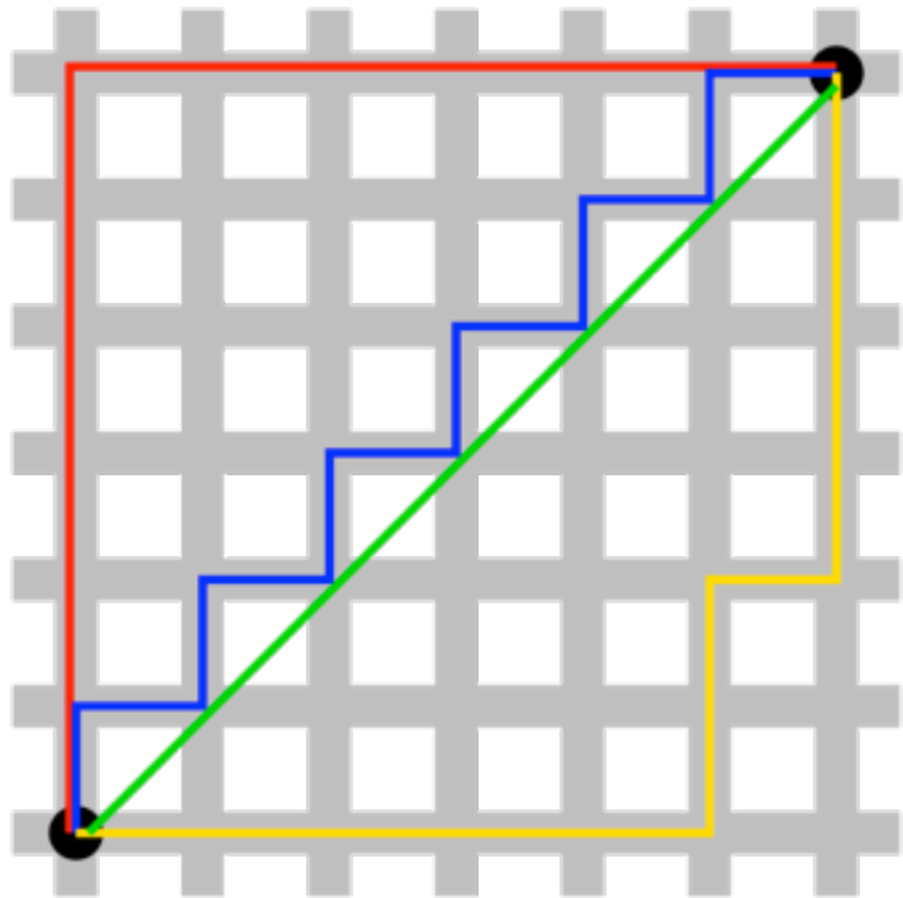
Hamming Distance: 3

The number of "swaps" to get from one thing to the target

Euclidean Distance

Distance between two points (generalizable to n-space)

Manhattan Distance



Not the green one

Distance between two points in city blocks (also generalizable to n-space)

Problem Spaces with Examples

- Our dataset contains the following features: Eye color, Hair color, Gender, Height, Weight
- Since we have 5 features our space is 5-D
- A problem space is the set containing all possible combinations of these features
- A feature vector is a single set of features, i.e. [Blue, Blonde, Woman, Average, Thick]

Fitness Landscapes

- Fitness landscapes are also known as "Error Spaces"
- Add N dimensions to the feature vector to express the amount of error that feature vector has
- The slopes on a fitness landscape is what gradient searches use to find the maxima

Optimization Algorithms

- Optimization algorithms try and find some optimal configuration of a feature vector
- An optimal feature vector depends on the application, examples include:
  - City order in Traveling Salesman Problems
  - Neural Net topology
  - My ideal date given a pool of applicants

Particle Swarm Optimization

A More Mundane Use of PSO Algorithms

- We want to find the feature vector (5, 8) in our search space
- Fitness function is determined by the Euclidean Distance from one particle to the target

PSO Algorithm Overview

- Initalize the particles
  - Set starting location, velocity and fitness
- While stop criteria hasn't been met
  - Check each particles fitness saving the best ever and the best at this time
  - Flock particles toward the best at this time and the best ever

PSO Initalization

```
function init_particles(num, dim, rand)
    local particles = {}
    for i=1, num do
        local particle = {}
        particle.p = {}
        particle.v = {}
        for j=1, dim do
            if rand then
                tinsert(particle.p, random())
                tinsert(particle.v, random())
            else
                tinsert(particle.p, 0)
                tinsert(particle.v, 0)
            end
        end
        particle.f = 999999
        tinsert(particles, particle)
    end
    return particles
end
```

PSO Runtime Loop

```
function run_pso (param, fitness_func)

    local particles = init_particles(num, dim, random)

    local pbest = particles[1]
    local gbest = particles[1]

    while iterations > 0 and pbest.f > success do
        pbest, gbest = calc_fitness(particles, fitness_func, pbest)
        update_particles(particles, pbest, gbest, pphi, gphi)
        print_particles(particles)
        iterations = iterations - 1
    end

    return pbest
end
```

PSO Calc Fitness

```lua
function calc_fitness (particles, fitness_func, pbest)
    local gbest = particles[1]

    for i=1, getn(particles) do
        particles[i].f = fitness_func(particles[i])

        if particles[i].f < pbest.f then
            pbest = {
                p = {},
                v = {},
            }
            for j=1, getn(particles[i].p) do
                tinsert(pbest.p, particles[i].p[j])
                tinsert(pbest.v, particles[i].v[j])
            end
            pbest.f = particles[i].f
        end

        if particles[i].f < gbest.f then
            gbest = particles[i]
        end
    end

    return pbest, gbest
end
```

PSO Update Particle

```
function update_particles (particles, pbest, gbest, pphi, gphi)
    for i=1, getn(particles) do
        particles[i].v =
            update_velocity(particles[i], pbest, gbest, pphi, gphi)
        particles[i].p =
            update_position(particles[i])
    end
end
```

PSO Update Position and Velocity

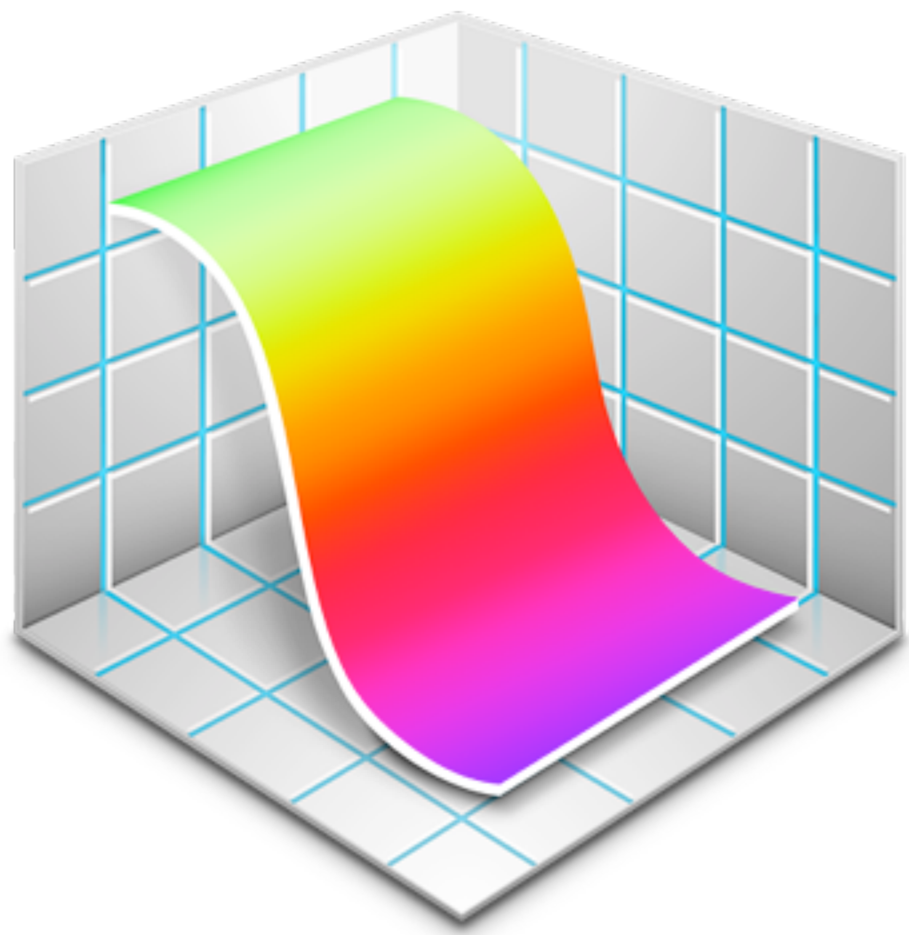
```
function update_position (particle)
    local position = {}
    for i=1, getn(particle.p) do
        tinsert(position, particle.p[i] + particle.v[i])
    end
    return position
end

function update_velocity (particle, pbest, gbest, pphi, gphi)
    local velocity = {}
    for i=1, getn(particle.v) do
        local prand = pphi*random()
        local grand = gphi*random()
        local pdiff = pbest.p[i] - particle.p[i]
        local gdiff = gbest.p[i] - particle.p[i]
        local vel = particle.v[i] + (prand * pdiff) + (grand * gdiff)
        tinsert(velocity, vel)
    end
    return velocity
end
```

PSO Fitness Function

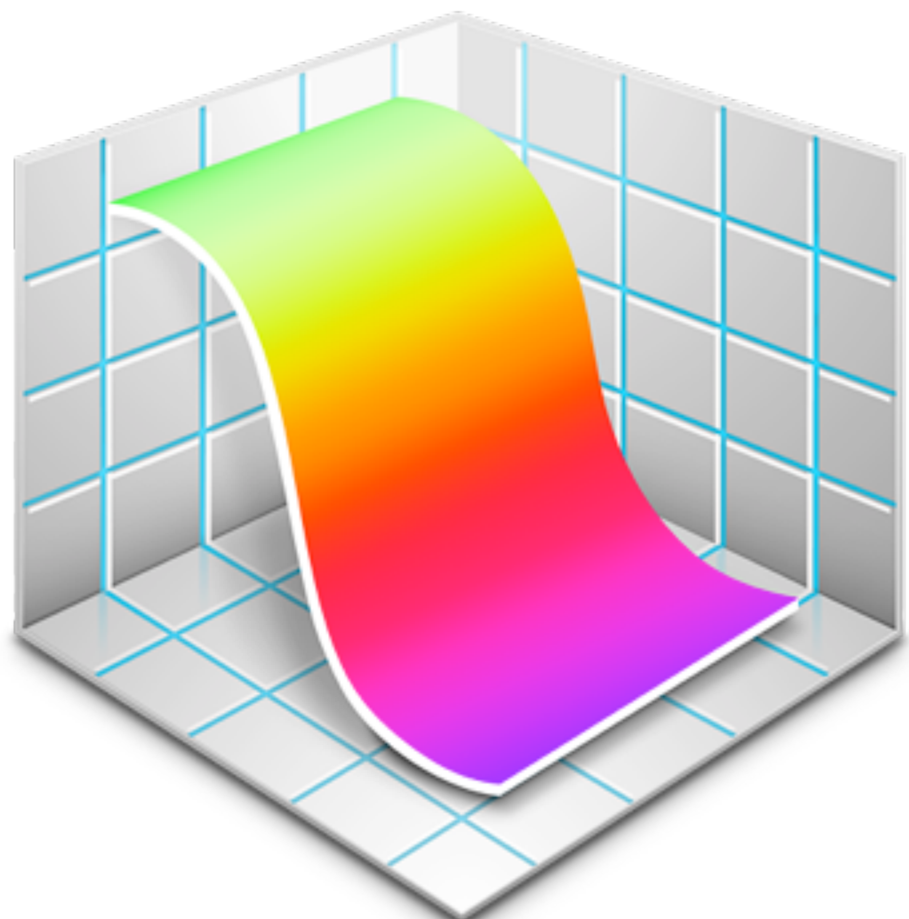
```lua
fitness_func = function (particle)
    local point = {5, 8}
    local fitness = 0.0
    for i=1, getn(point) do
        fitness = fitness + (point[i] - particle.p[i])^2
    end
    return sqrt(fitness)
end
```

Note the n-space flair

PSO Problem Space

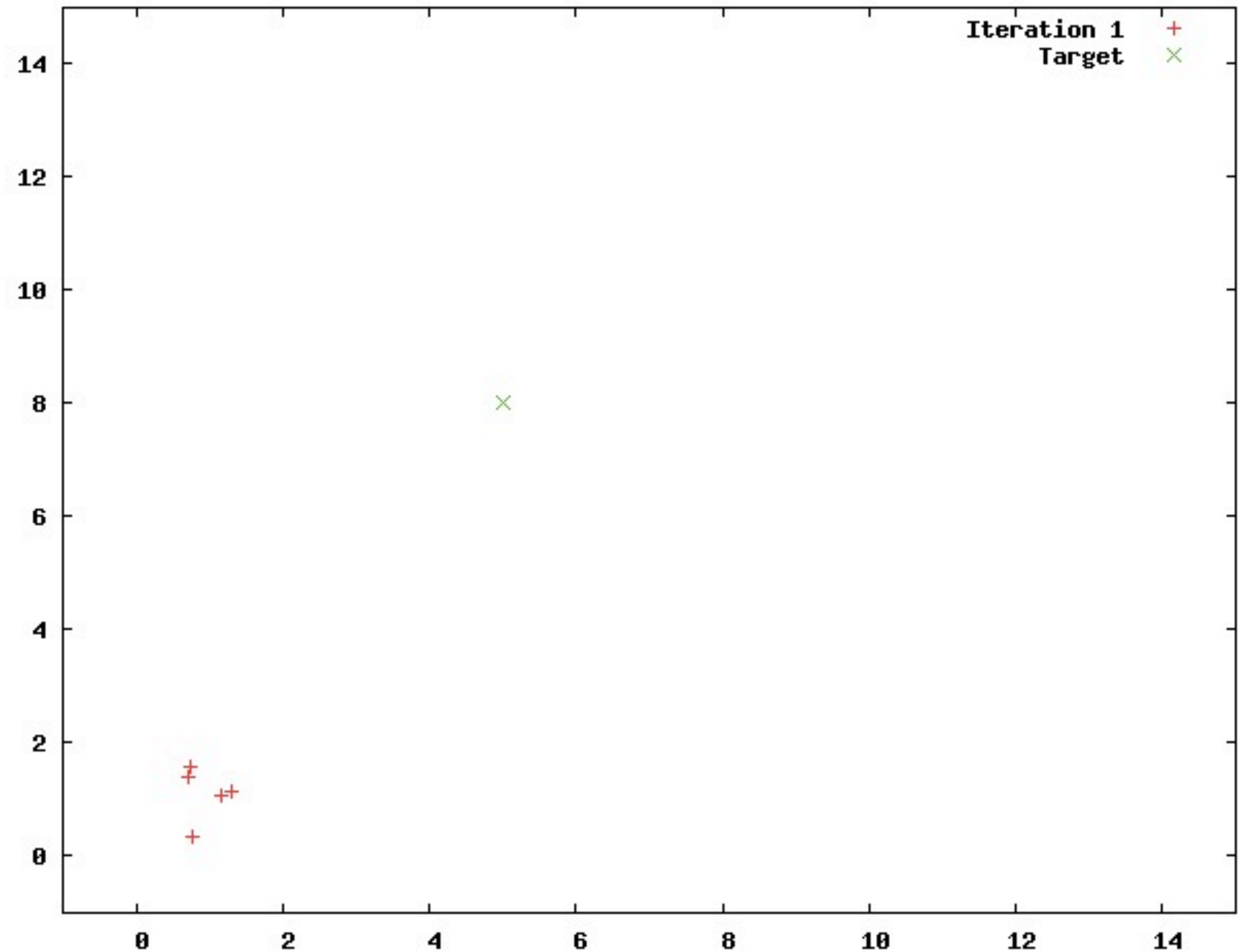


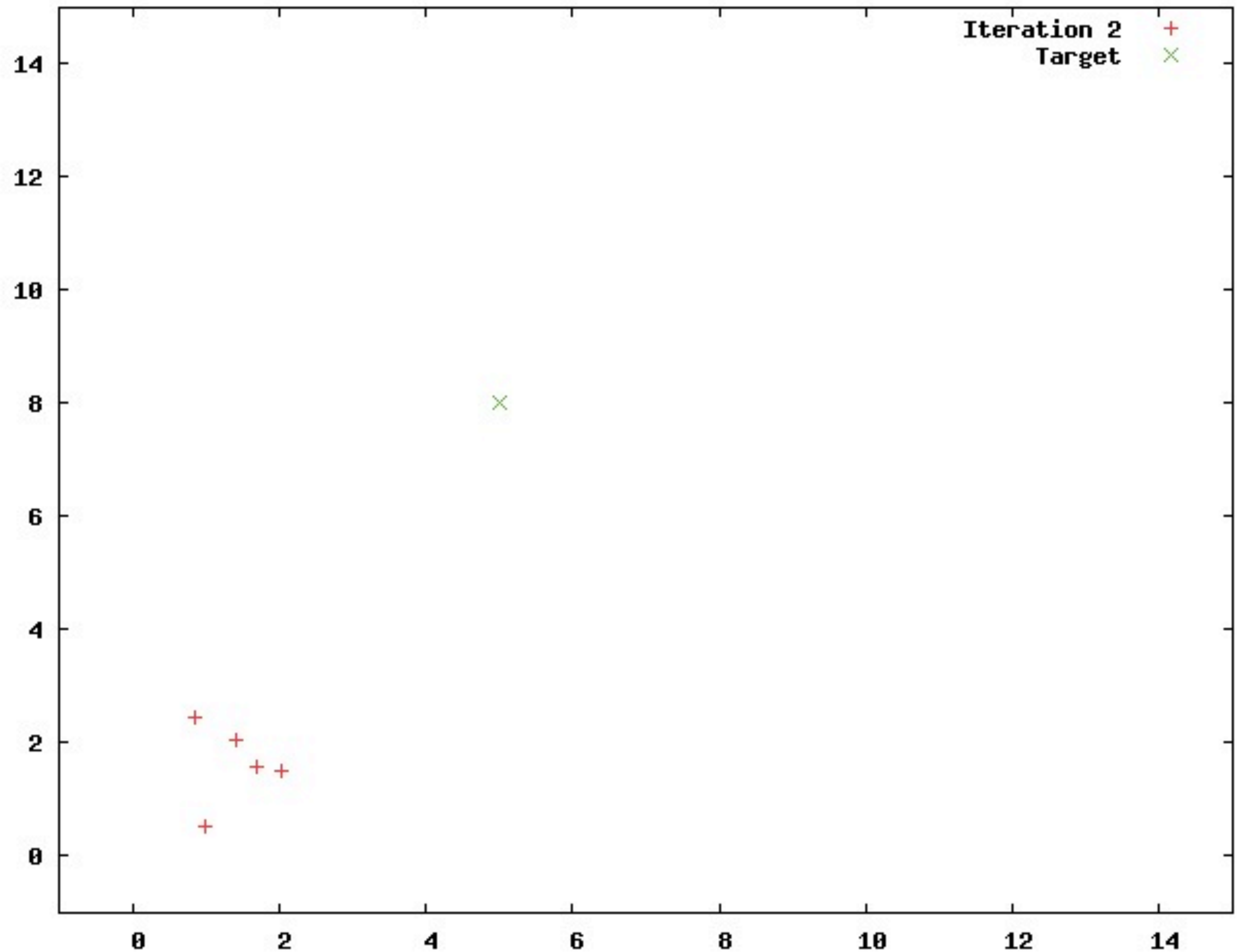
PSO Fitness Landscape



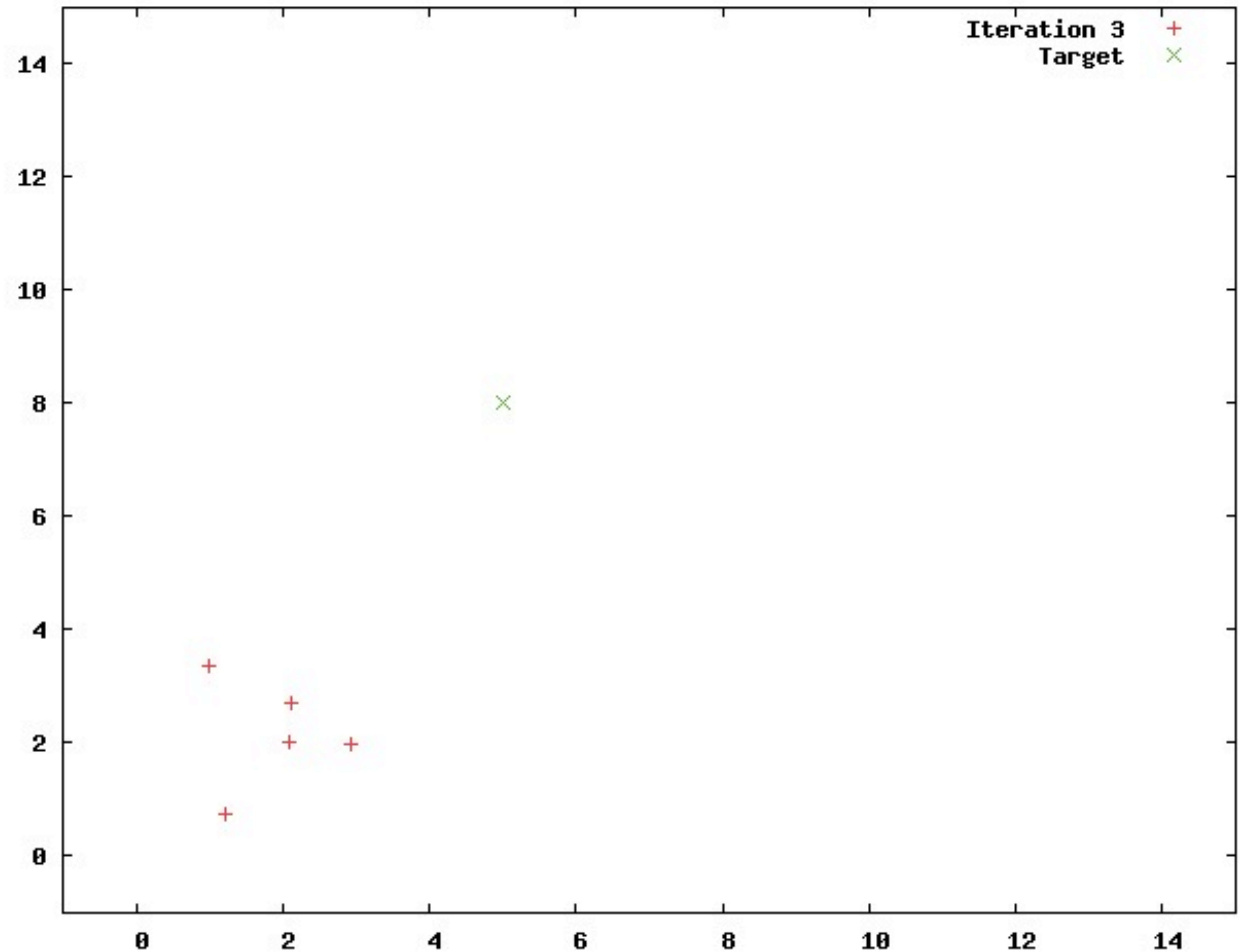
PSO Results

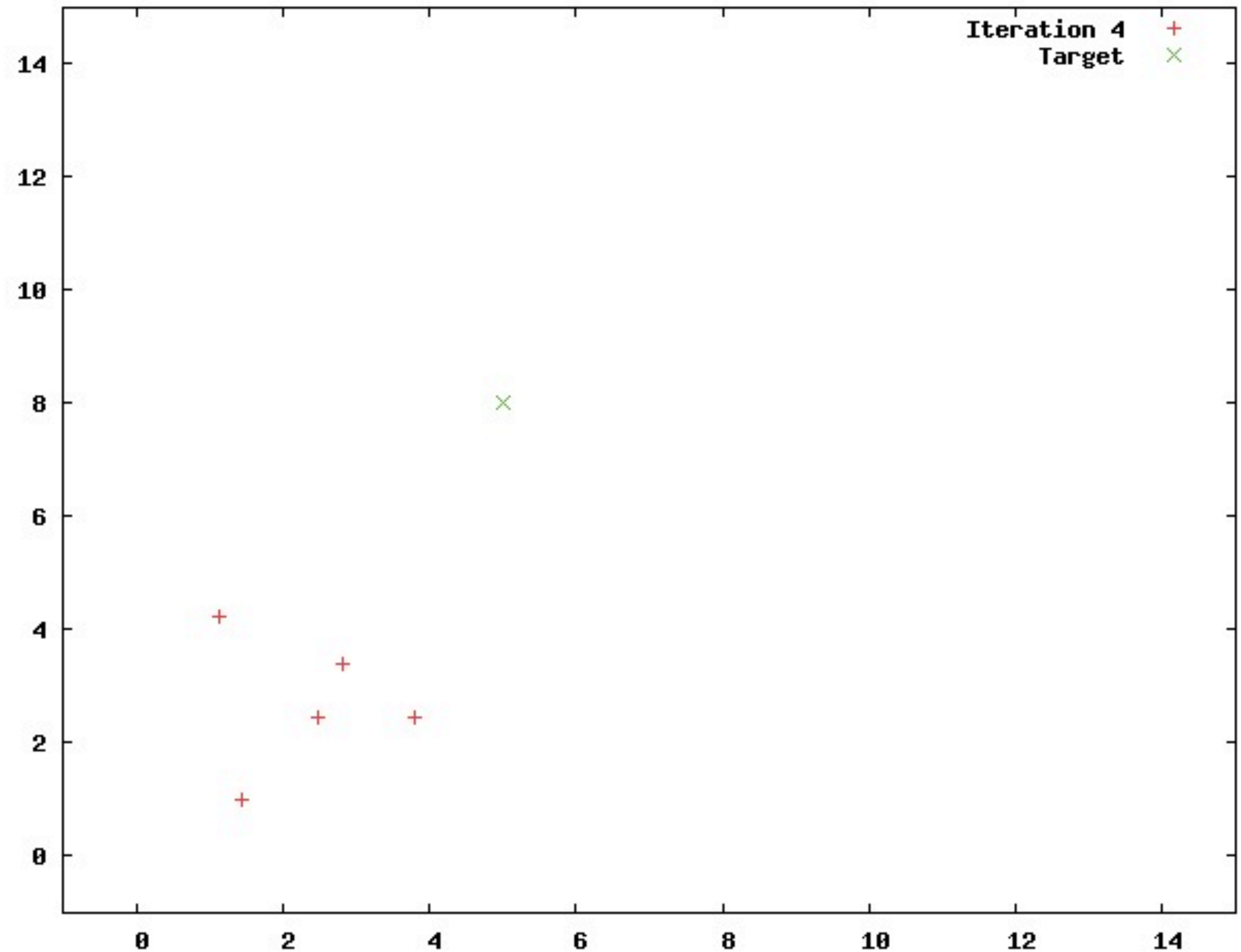
- The graphs will show the positions of 5 particles trying to find the feature vector: (5, 8)
- Last graph will have the best particle ever found
- Note particles search the problem space to find the maxima (in this case point (5, 8))

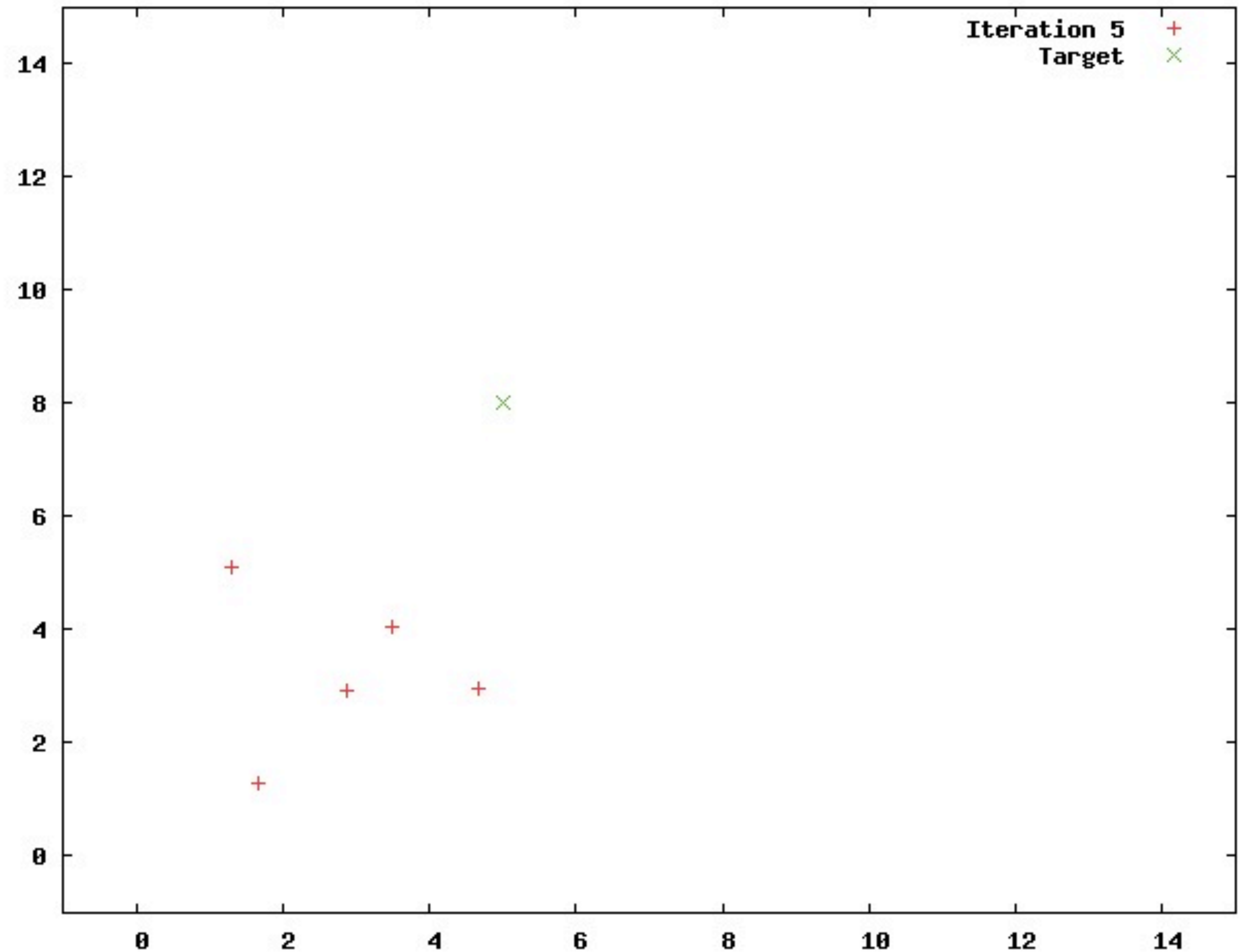

Iteration 1
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

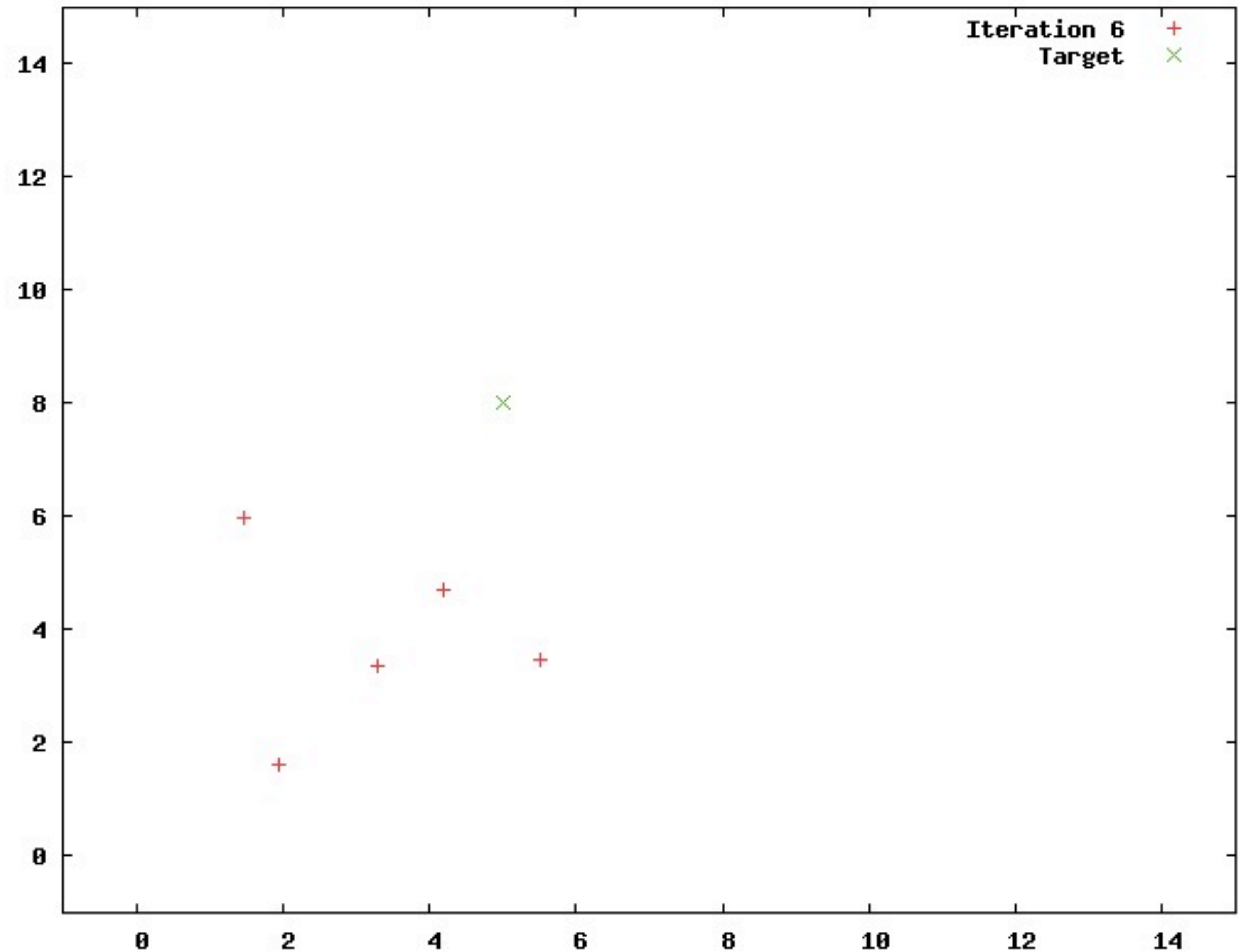


Iteration 2
Target
0
2
4
6
8
10
12
14

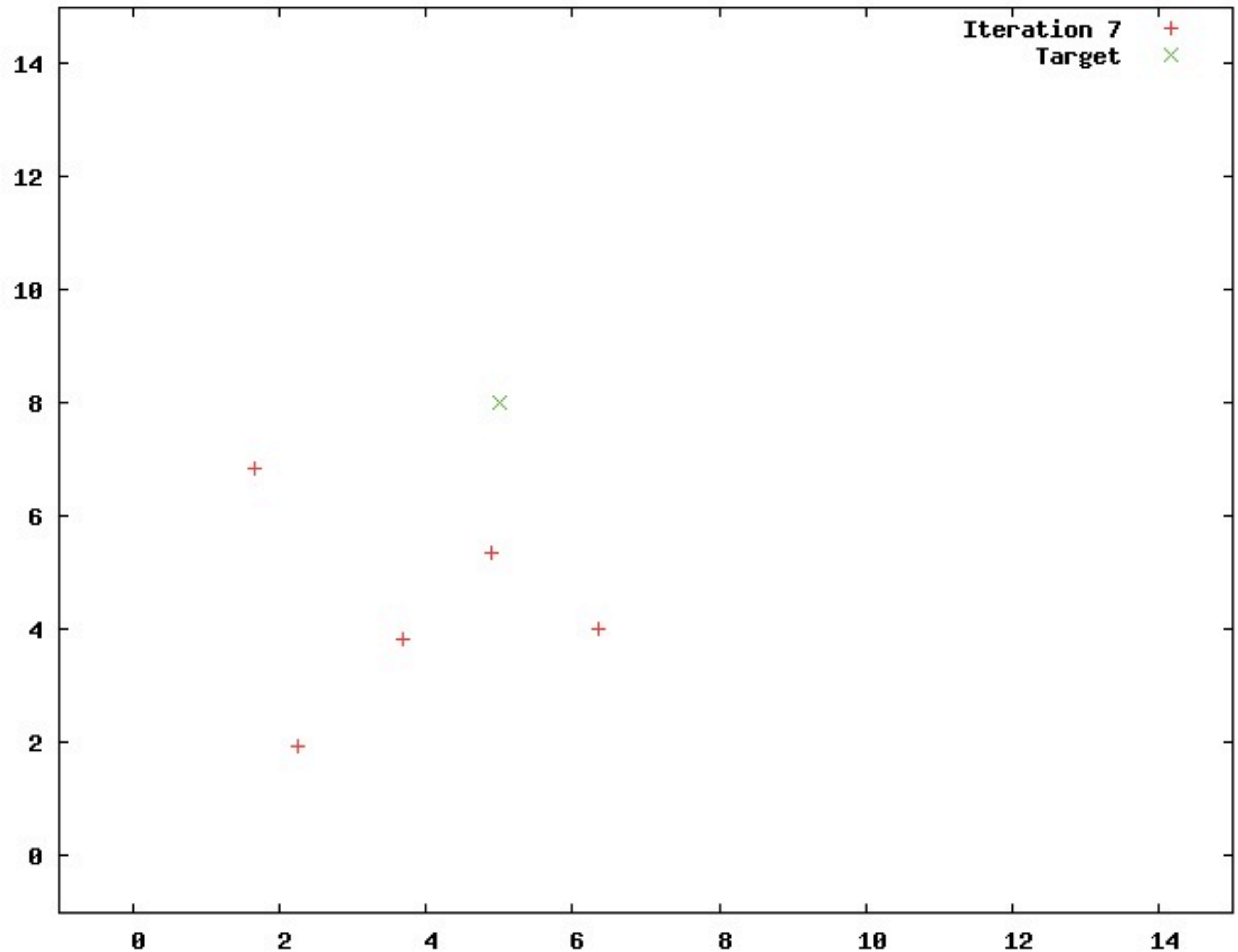

Iteration 3
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

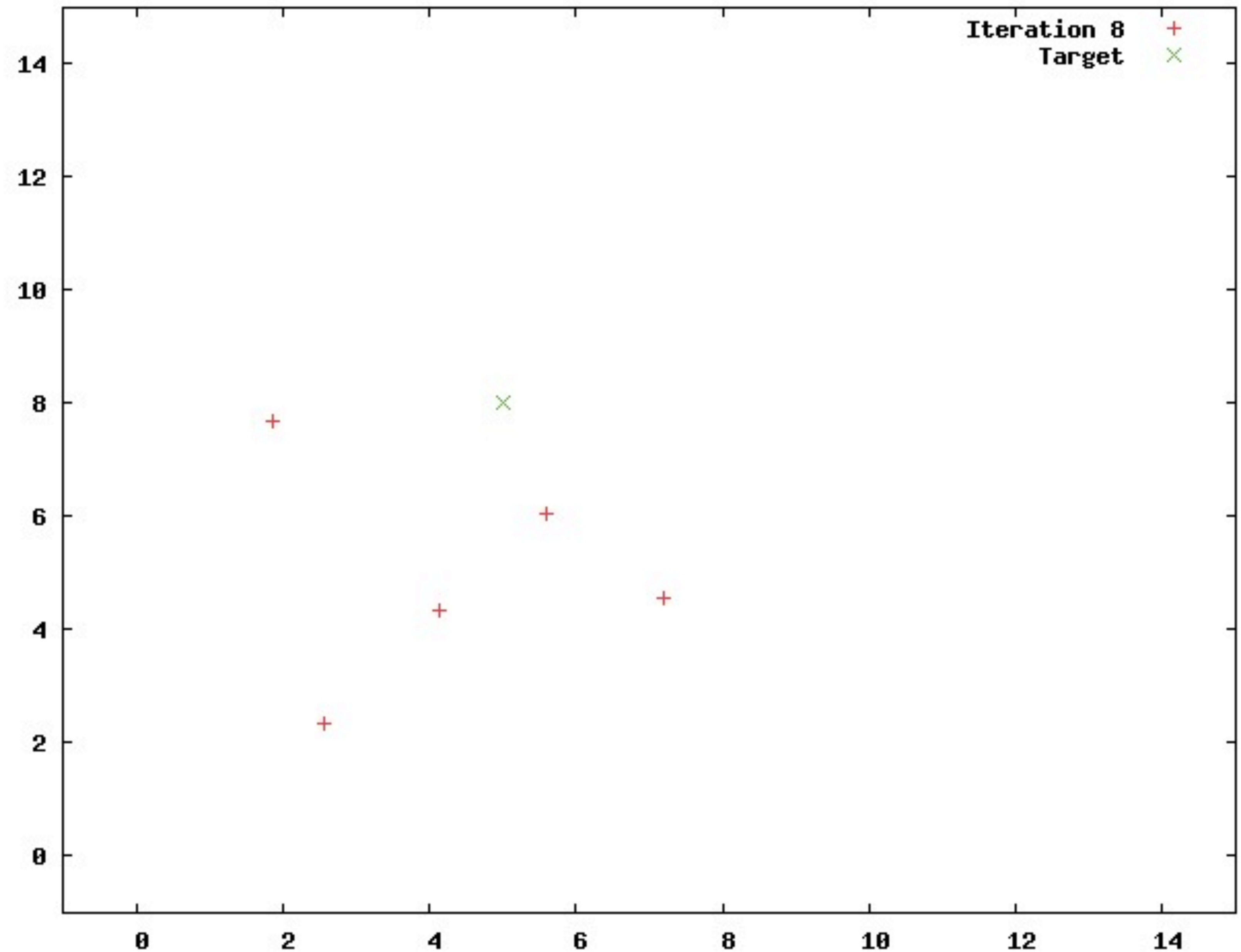

Iteration 4
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

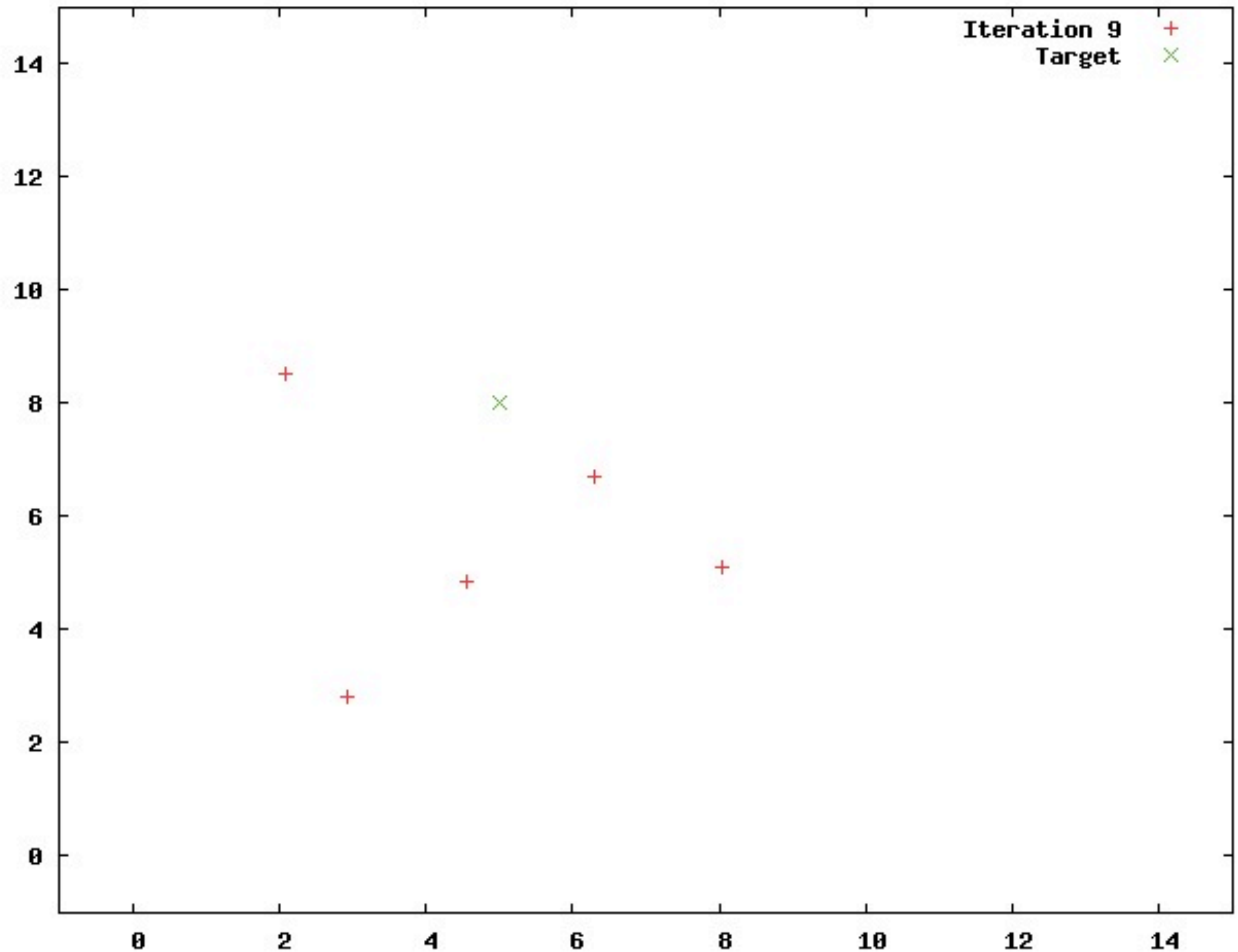

Iteration 5
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

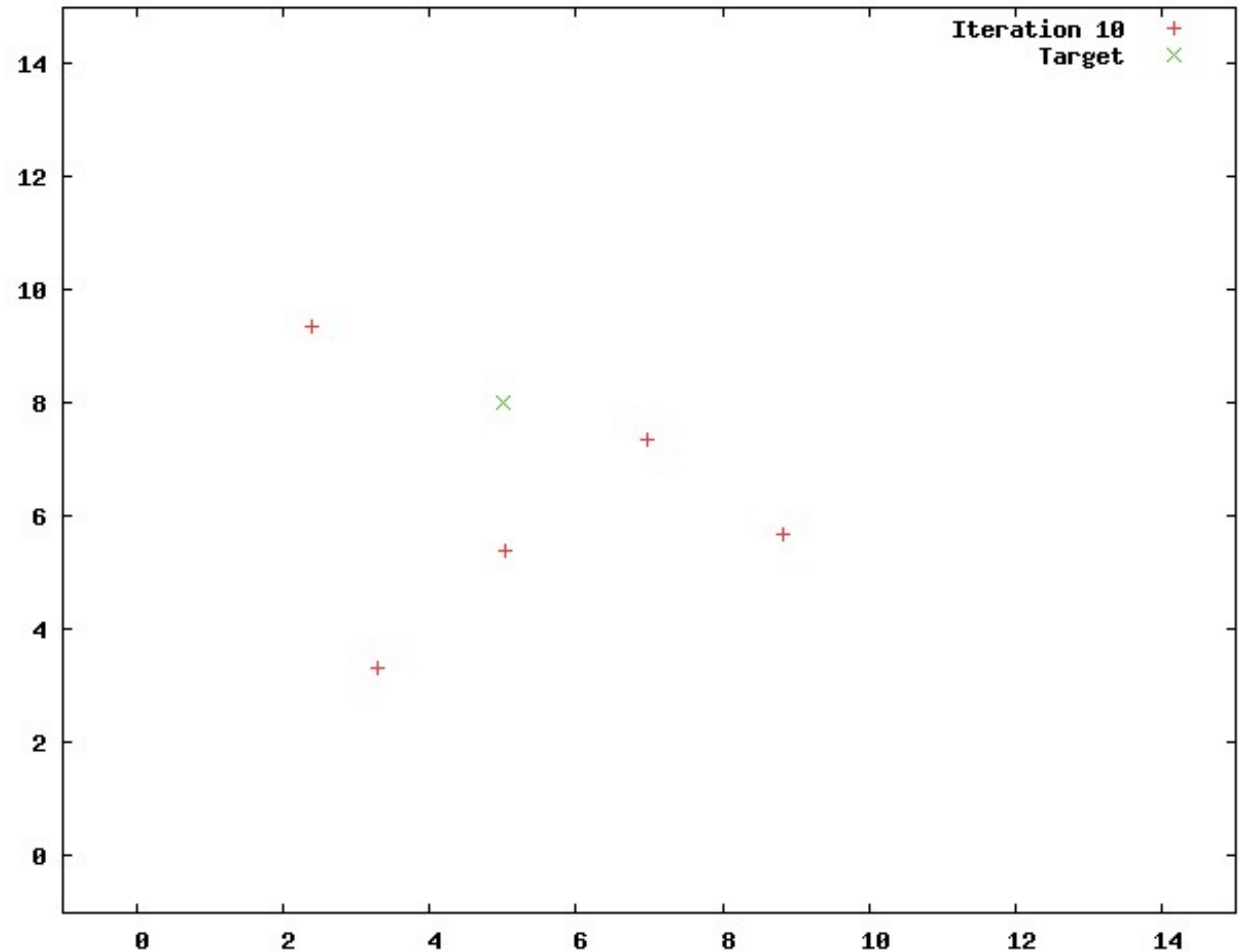


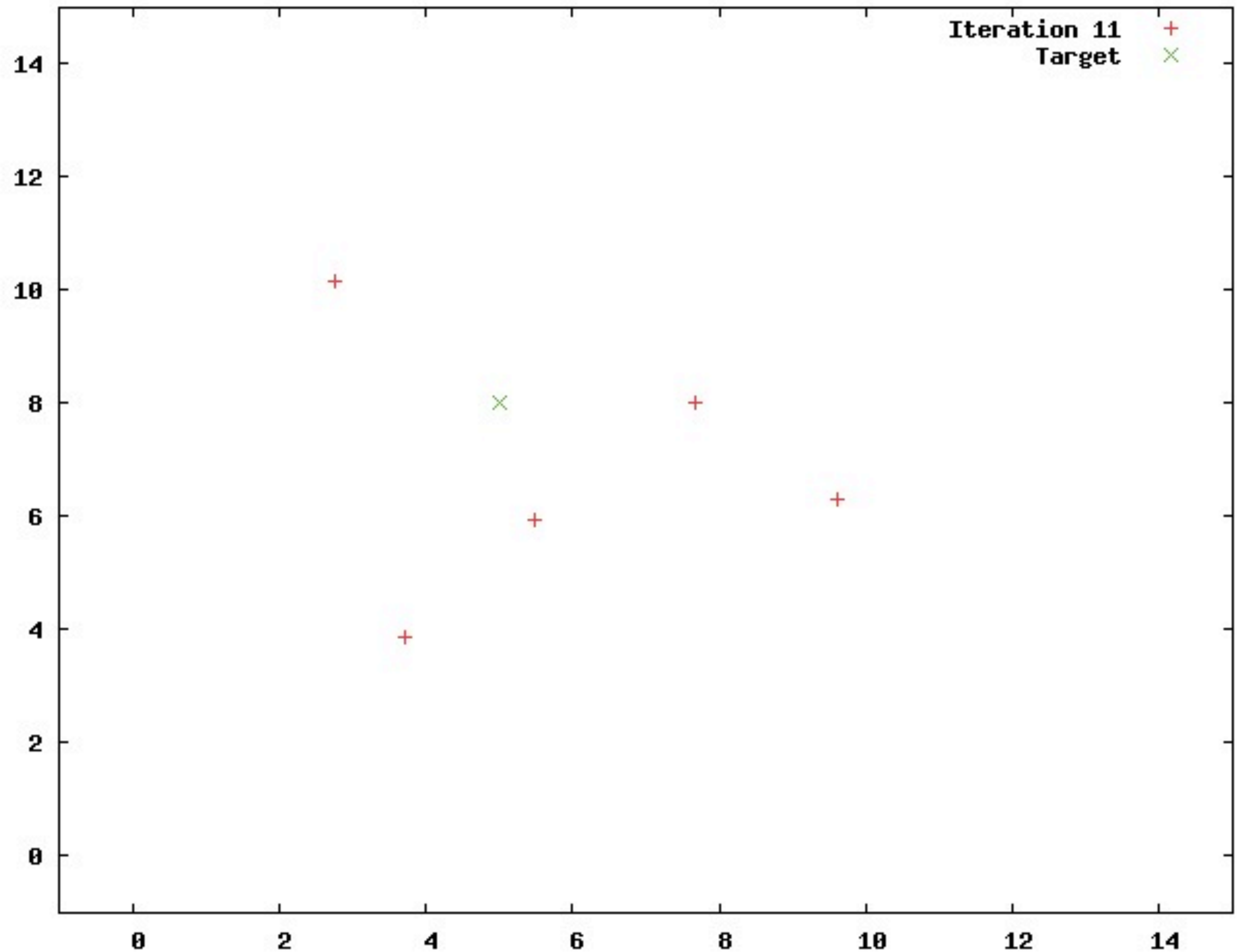
Iteration 6
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14


Iteration 7
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

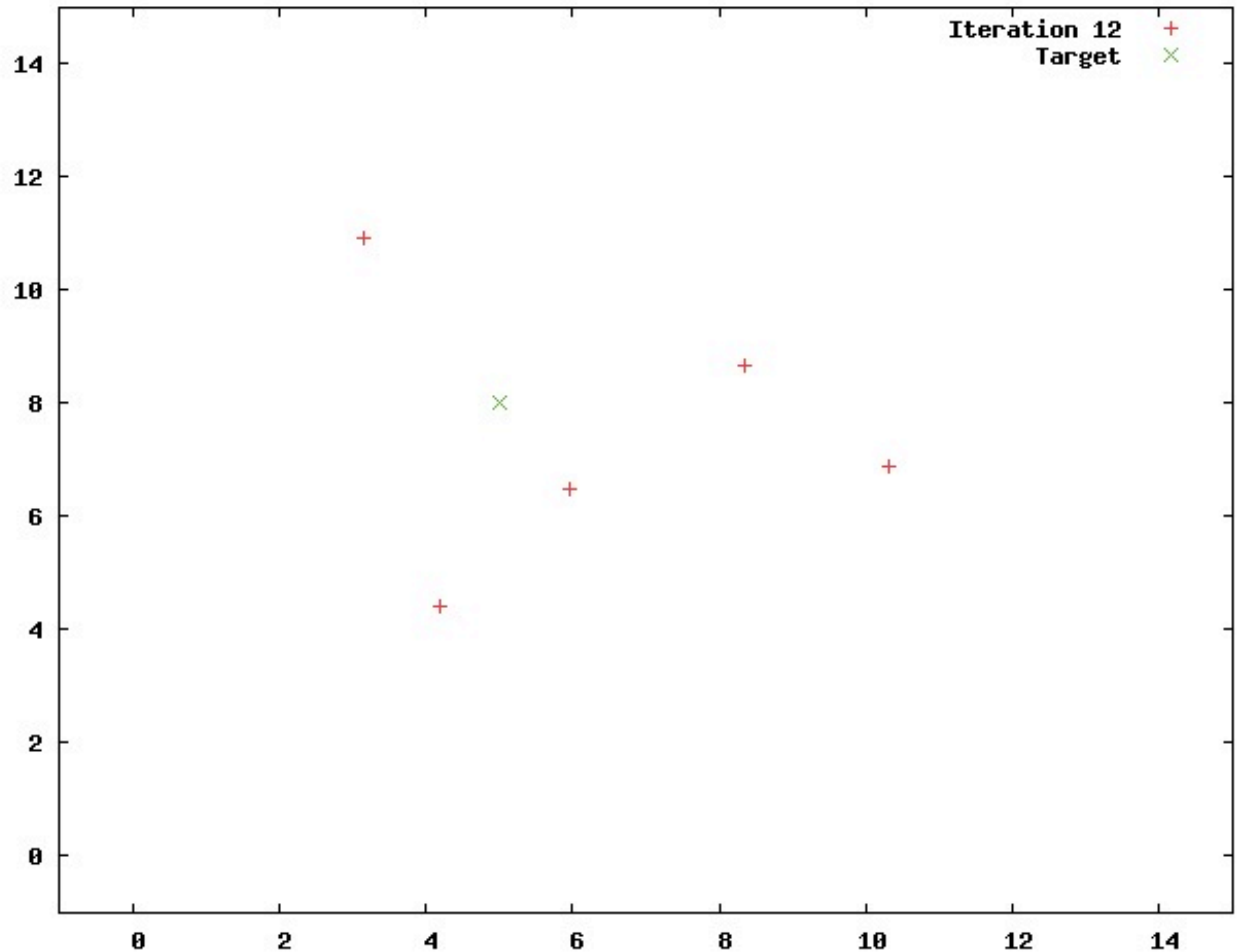

Iteration 8
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

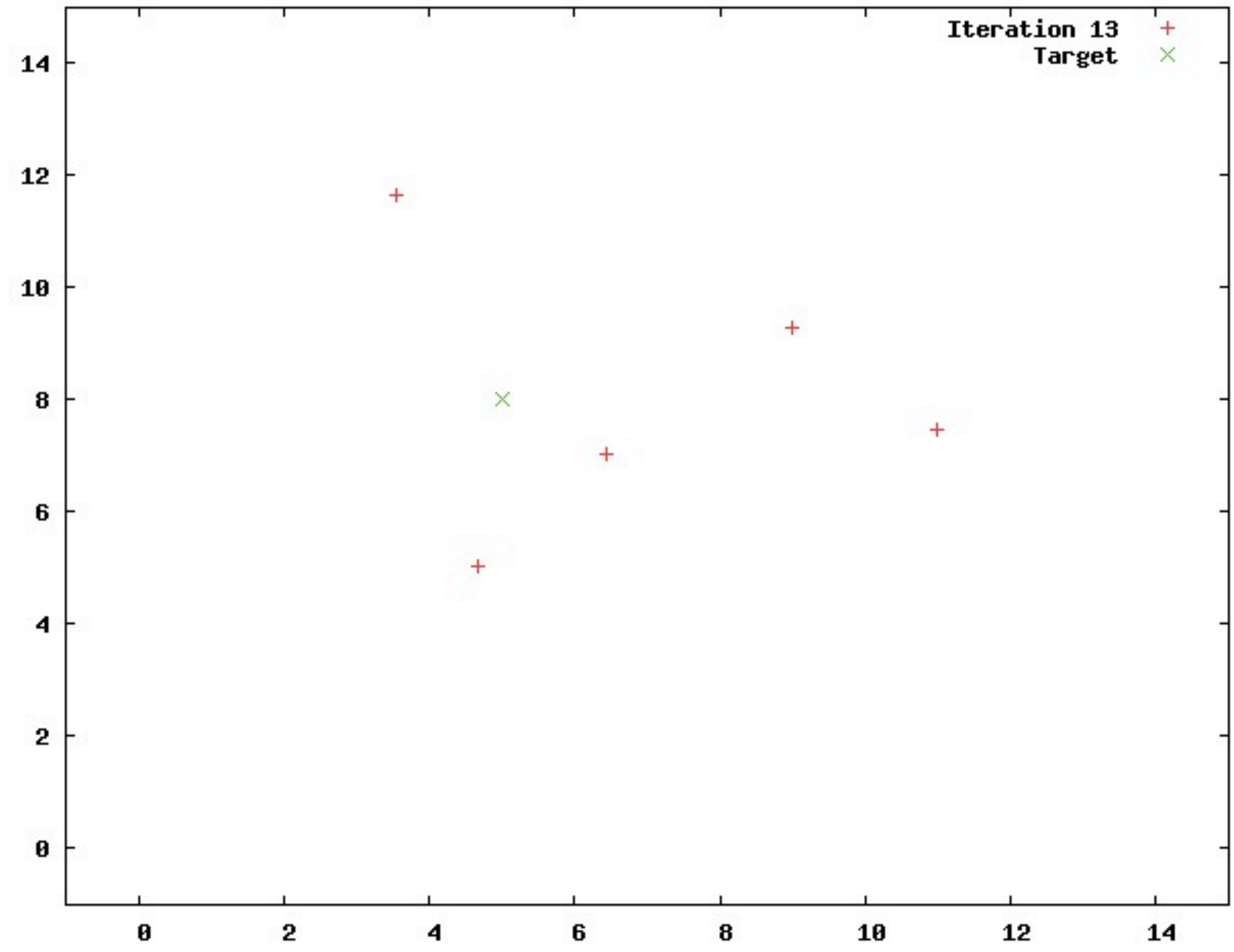

Iteration 9
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14


Iteration 10
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

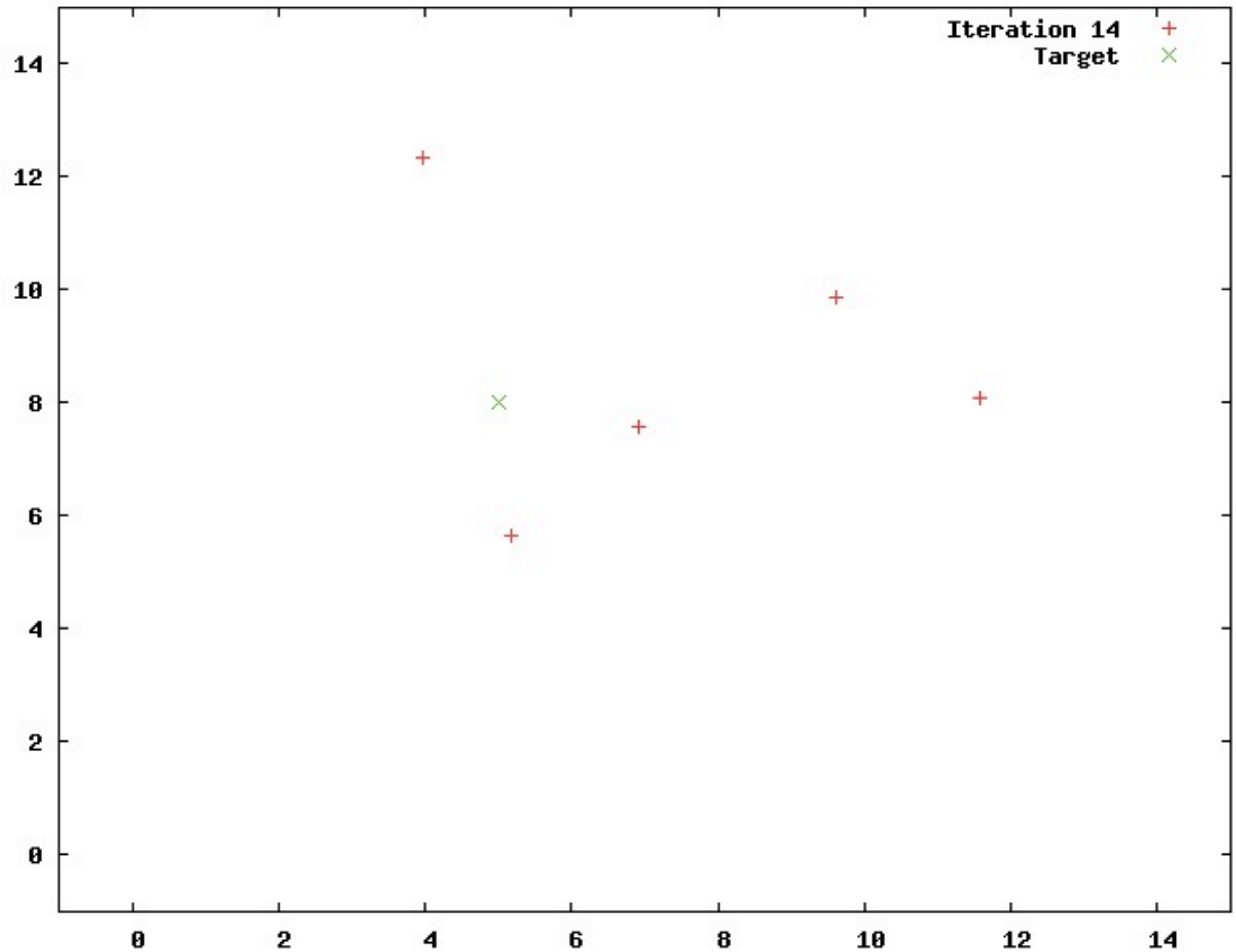


Iteration 11
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14

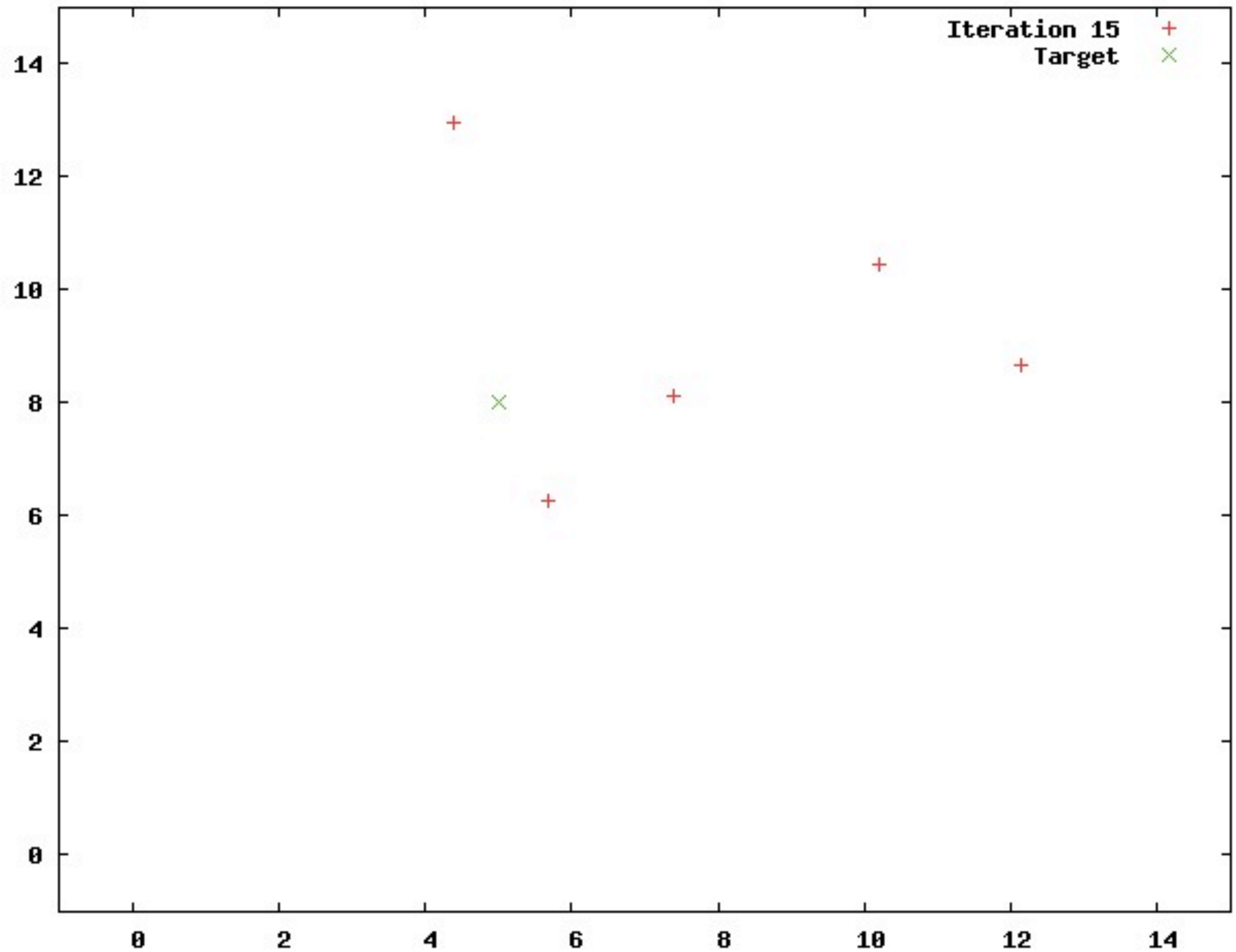

Iteration 12
Target
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14



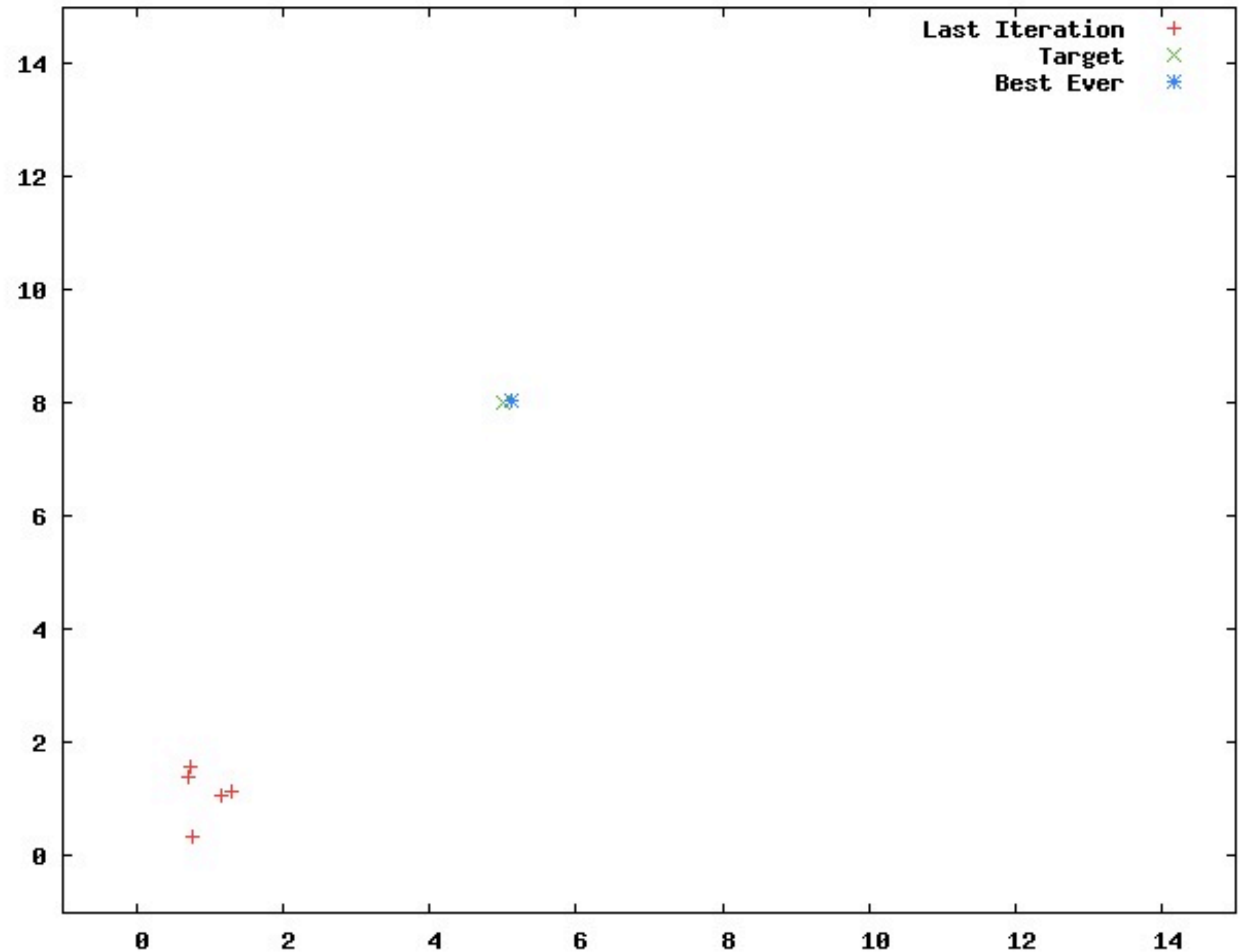
Iteration 13
Target
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14



Iteration 14
Target
0
2
4
6
8
10
12
14


Iteration 15
Target
0
2
4
6
8
10
12
14

85 Iterations Later...


Last Iteration
Target
Best Ever
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14

PSO and NN: Stochastic Training

- Change the vector of numbers to represent the weights of all of the connections in a neural network
- Change the fitness function to how well the net performs on training data
- Use Hamming Distance to calculate the distance from actual to target outputs

Questions?

?