Optimization

Neural Networks & Fuzzy Logic (BITS F312)

Ashish Patel

Lectures during April 20 - 28, 2018

Module Contents

- Introduction
- Traditional optimization techniques
  - Steepest Descent Method
- Non-traditional optimization techniques
  - Genetic Algorithm (GA)
  - Particle Swarm Optimization (PSO)

Introduction

Introduction

- Optimization is defined as the method of minimizing or maximizing a function of single or several variables, i.e. finding those values the variables for which the function takes on the minimum or maximum value.

- Point at which the function takes its minimum or maximum value is called a minimizer or maximizer respectively.

Statement of an Optimization Problem

An unconstrained optimization problem can be stated as follows:

Find $\mathbf{X} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix}$ which minimizes $f(\mathbf{X})$

where **X** is an n-dimensional vector called the *design vector*, and $f(\mathbf{X})$ is termed the *objective function*.

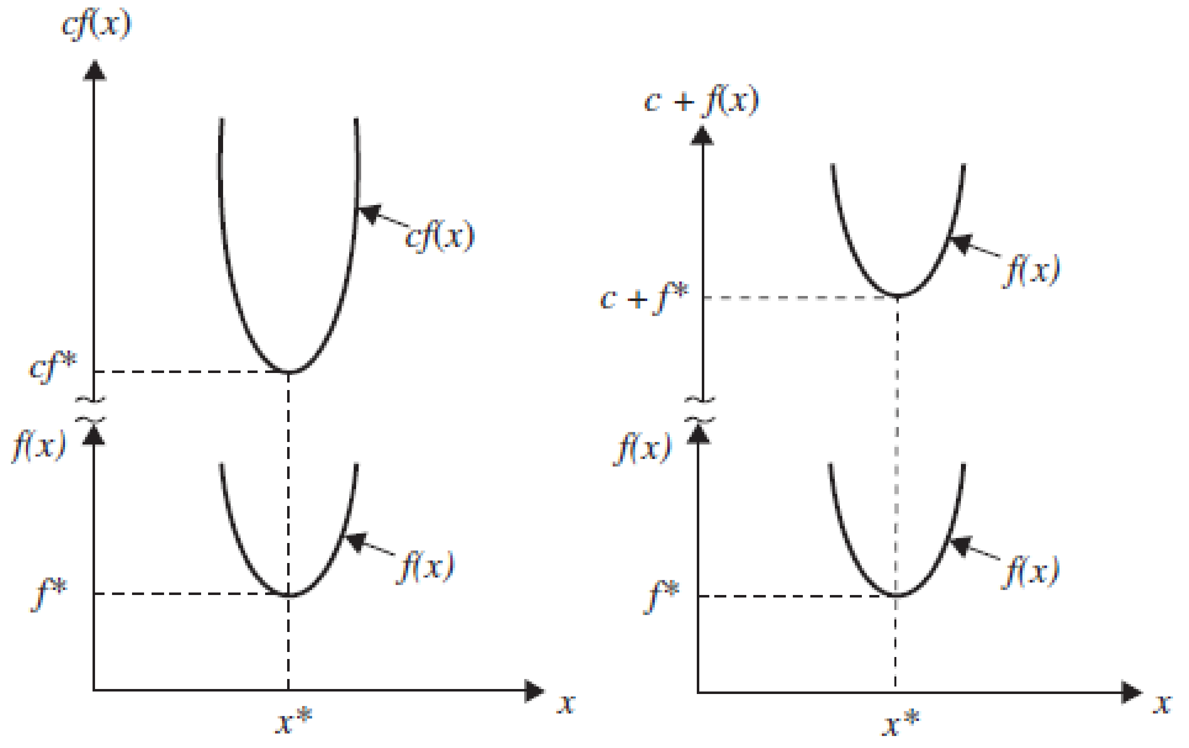
A constrained optimization or mathematical problem can be stated as follows:

Find $\mathbf{X} = \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{Bmatrix}$ which minimizes $f(\mathbf{X})$, subject to the constraints

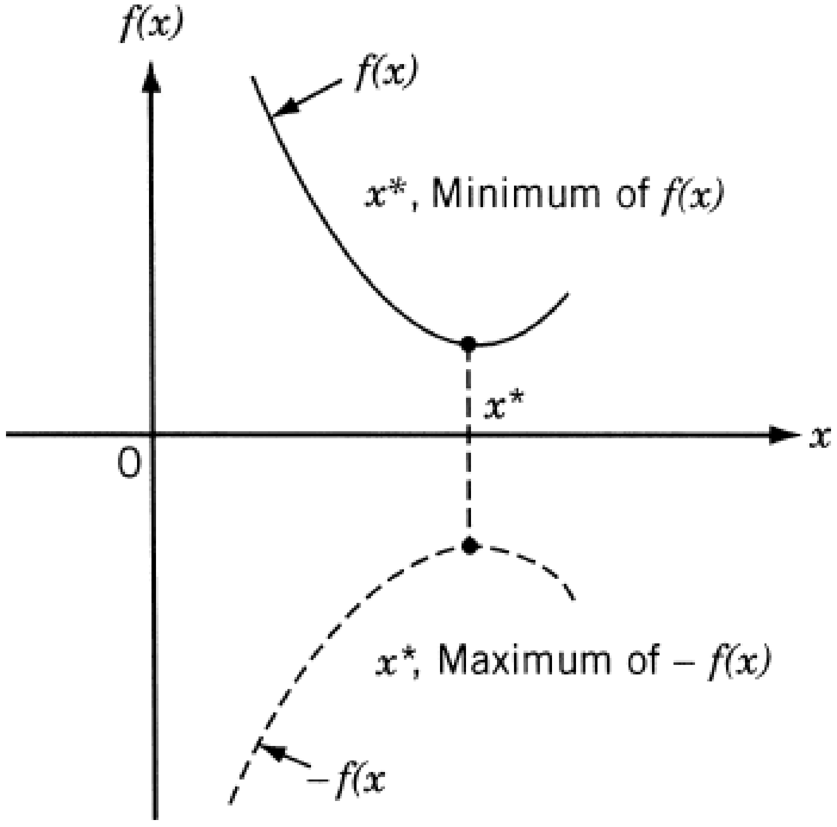
$g_i(\mathbf{X}) \leq 0$, $l_j(\mathbf{X}) = 0$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, p$

where **X** is an n-dimensional vector called the *design vector*, $f(\mathbf{X})$ is termed the *objective function*, and $g_i(\mathbf{X})$ and $l_j(\mathbf{X})$ are known as *inequality* and *equality* constraints respectively.

- Following operations on the objective function will not change the optimum solution *x**, as shown in Fig. 1.2:
  1. Multiplication/division of *f*(*x*) by a positive constant *c*.
  2. Addition/subtraction of a positive constant *c* to/from *f*(*x*).



- Fig. 1.1 shows that if a point *x** corresponds to the minimum value of function *f*(*x*), the same point also corresponds to the maximum value of the negative of the function, −*f*(*x*).



Non-Traditional Optimization Techniques

- Most of the traditional optimization techniques based on gradient methods have the possibility of getting trapped at local minima, depending upon the degree of non-linearity and the initial guess.

- Traditional optimization techniques do not ensure the global optimum.

- Population based non-traditional optimization techniques such as GA, DE, and PSO use natural phenomena and are stochastic in nature.

- Advantages of Population based Global Optimization Techniques
  - Don’t require derivative information
  - Search wide domain of the objective/cost surface
  - Deal with a large number of variables
  - Well suited for parallel computers
  - Optimize variables with extremely complex cost surfaces (they can jump out of a local minimum)
  - Suggest possible other solutions for a given design problem with multiple global solutions which may be marginally inferior to absolute global solution but more suitable for implementation

- Disadvantages of Population based Global Optimization Techniques
  - Population based global optimization techniques are not the best way to solve every problem.
  - For instance, the traditional methods have been tuned to quickly find the solution of a well behaved convex analytical function of only a few variables.
  - For such cases the calculus-based methods outperform the global optimization techniques, quickly finding the minimum while the global optimization technique is still analyzing the costs of the initial population.

Genetic Algorithm

Genetic Algorithm (GA)

- GA was introduced by Holland* in 1975 and popularized by his student Goldberg**

- Philosophically, GA is based on ‘Theory of Evolution’.

- GA adopts the principles of genetics – reproduction, crossover, mutation

- GA allows a population composed of many individuals to evolve under specified selection rules to a state that maximizes the function

* Holland, John (1992). *Adaptation in Natural and Artificial Systems*. Cambridge, MA: MIT Press.
** Goldberg, David (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley Professional.

Steps for Genetic Algorithm (GA)

- Consider a maximization or minimization problem:
  - Maximize Fitness Function $F(x) = f(x)$ or $1/(1+f(x))$ respectively
  - Initial population range is given as $x_i^{(L)} \le x_i \le x_i^{(U)}$ $where\ \ i = 1,2,......,N$
  - For unconstrained optimization, boundary values are not applicable to evolved variables whereas for constrained optimization, evolved variables should follow it strictly.
- Important steps for GA realization
  - Variables Encoding
  - Fitness function Evaluation and Elite Selection
  - Application of GA operators: Reproduction, Crossover, Mutation
  - Check boundary conditions and termination criteria

GA variables

- GA begins by defining a chromosome or an array of variable values to be optimized.

- If the chromosome has N variables (an N-dimensional optimization problem) given by $x_1, x_2, \dots, x_N$, then the chromosome is written as an N element row vector.

  $chromosome = [x_1, x_2, \dots, x_N]$

- A group of ‘n’ chromosomes form a population of ‘n’ members.

Encoding of GA variables

- Coding is the method by which the variables x_i coded into string structures.

- Binary coding is generally used to translate the range of the function variables. This essentially means that a certain number of initial guesses are made within the range of the function variables and these are transformed into a binary format.

- Length of the binary coding is generally chosen with respect to required accuracy.

- For l_i - bit binary encoding, there are 2^{l_i} combinations or codes possible. Accuracy in variable values is given by

$$Accuracy = \frac{x_i^{(U)} - x_i^{(L)}}{2^{l_i}}$$

- Following linear mapping rule is used for the purpose of l_i - bit binary encoding:

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{l_i} - 1}\left[\sum_{j=0}^{l_i - 1}\left(S_j 2^j\right)\right]$$

where $S_j \in \{0, 1\}$

- For 10 - bit binary encoding, upper limit = 6, lower limit = 0

$$Accuracy = \frac{x_i^{(U)} - x_i^{(L)}}{2^{l_i}} = ?$$

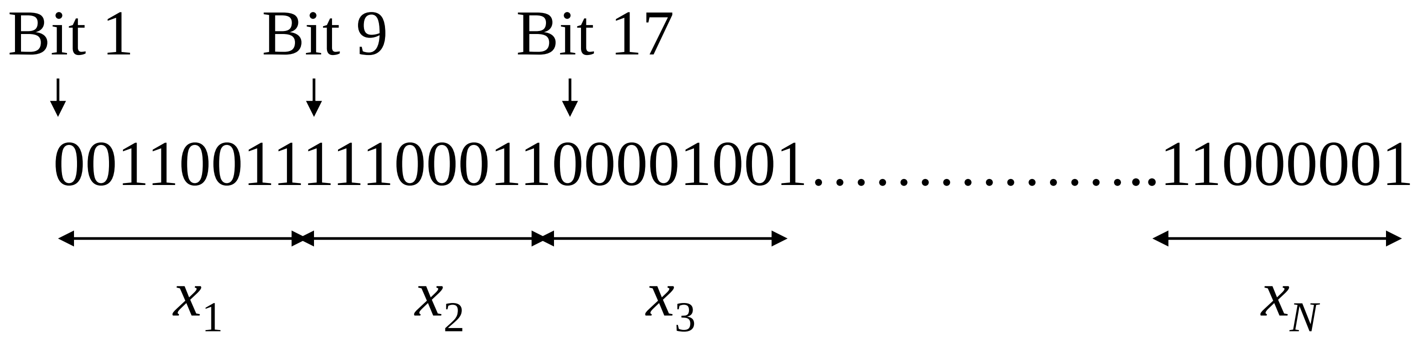
- 0.0059
- For string, 1110111011, what will be decoded and linear values?

$$x_i = x_i^{(L)} + \frac{x_i^{(U)} - x_i^{(L)}}{2^{l_i} - 1} \left[\sum_{j=0}^{l_i - 1} \left(S_j 2^j \right) \right]$$

- Decoded value = 955
- Linear value (x_i) = 5.601

- If we take *l*-bit binary encoding for each of *N* variables of the chromosome, then the chromosome will be represented by continuous bit pattern of $N \times l$ bits.

- Here, starting from left, first *l* bits represent variable x_1, next *l* bits represent variable x_2, and so on. Pictorially it is shown below for a particular chromosome with bit positions mentioned with respect to MSB.



Evaluation of GA Fitness Function

- Fitness function measures probability of a population member (a point) to yield maximum value of function.
- The ‘good points’ or the points which yield larger values for the function have higher probability to continue in the next generation. The ‘bad points’ or the points which yield smaller values for the function have lower probability to continue in the next generation.
- GA maximizes a given function, so it is necessary to transform a minimization problem to a maximization problem. This transformation does not alter the location of the minimum value.

- Depending upon whether initial objective function $f(x)$ needs to be maximized or minimized, the fitness function $F(x)$ is defined as:
  - $F(x) = f(x)$ for maximization problem
  - $F(x) = 1/(1+f(x))$ for minimization problem

- The fitness function value for a particular coded string is known as the string's fitness and it is used to decide whether a particular string carries on to the next generation or not.

- Best performing chromosomes can be guaranteed to move to next generation using Elite Count (*EC*), a non-zero and positive constant.

Reproduction

- Reproduction operation is also known as Selection operation since this operation decides the strings from current population to be selected for Crossover and Mutation operations in order to generate remaining ($n - EC$) members.

- The end result of this operation is the formation of a ‘mating pool’ where strings are selected in a probabilistic manner using following rule:

  Probability of selection into the pool $\propto$ *Fitness value of string*

- Probability of selection of the i^{th} string into the mating pool is

$$p_i = \frac{F_i}{\sum_{j=1}^{n} F_j}$$

where F_i is the fitness of the i^{th} string, n is the population size

- For minimization problem, we needed to maximize its probability of selection. That is the reason behind defining the fitness function as $F(x) = 1/(1+f(x))$.

- Roulette wheel based selection operation has following steps:
  - Using F_i, calculate p_i.
  - Calculate the cumulative probability P_i.
  - Generate n random numbers between 0 and 1.
  - Copy the string that represents the chosen random number in the cumulative probability range into the mating pool.

Crossover

- Crossover operation forms offspring chromosomes for next generation by using parent chromosomes from the mating pool of the current generation. Parent chromosomes chosen from mating pool are random since they resulted from probabilistic roulette wheel selection operation.

- For Single-point crossover operation, a random crossover site, say integer H, is chosen such that $0 \leq H \leq N \times l$, and all the bits to the right of the H^{th} position i.e. $(H+1)^{\text{th}}$ bit onwards in the two parent strings are swapped to get two children strings.

- Single-point crossover at site 11 is illustrated below for 16-bit chromosome:

Crossover site

Parent 1: **00110010111** ¦ **000110**   Offspring 1: **00110010111 011100**

Parent 2: **01110101000** ¦ **011100**   Offspring 2: **01110101000 000110**

- What happens if the crossover sites are $H = 0, N\times l$.

- Crossover operation introduces randomness into the current population to avoid getting trapped in local searches.

- Crossover operation may result in better or worse strings. If few worse off-springs are formed, they will not survive for long since upcoming generations' reproduction will eliminate them.

- What if majority of new off-springs are worse? To avoid such situations, we do not select all strings from the mating pool of current population for crossover.

- If crossover probability is p_c, then we use following number of crossover parents (*CP*) to generate equal numbers of crossover children : *Crossover Count (CC) = round*$(p_c \times (n - EC))$

- Crossover operation can be summarized as follows:
  - Select crossover parents (*CP*) from mating pool resulted by selection operation to generate equal numbers of crossover children: $CC = round(p_c \times (n - EC))$.
  - Select first two pairs of strings and generate a random integer number between 0 to $N \times l$ to decide single-crossover site.
  - Perform single-crossover operation by swapping all the bits to the right of crossover site

Mutation

- Mutation involves making changes in the population members directly, that is, by flipping randomly selected bits in certain strings.

- The aim of mutation is to change the population members by a small amount to promote local searches when the optimum is nearby.

- Mutation is performed by deciding a mutation probability p_m and selecting strings from mating pool on which mutation is to be performed.

- Mutation operation can be summarized as follows:
  - Select mutation parent (*MP*) from mating pool resulted by selection operation to generate mutation child:

    Mutation Count (MC) = (n – *EC* – *CC*).
  - Generate random numbers to decide whether mutation is to be performed on a particular bit of the population member or not.
  - If the random number is greater than mutation probability, do no flip particular bit (no mutation for the bit). Otherwise, flip the particular bit (bit is mutated).

Next Generation Population

- Following is order for next generation population of size = 20, elite count = 2, and crossover probability = 0.8.
  - Elite 1
  - Elite 2
  - Crossover child 1
  - Crossover child 2
  - ...
  - Crossover child 14
  - Mutation child 1
  - ...
  - Mutation child 4

Next Generation Population

- Elite members are updated in each iteration of GA, depending on fitness value of population members.
- In each iteration, first fitness value is calculated and then elite members are elected.
- So elite count (EC) remains same, but elite members change depending on fitness values in that iteration.
- Crossover count (CC) and mutation (MC) count also remain same, but members on which these operations are performed are changed depending on their fitness and selection process.

Termination criteria

- Following termination criteria can be used to Stop GA procedure.
  - Maximum number of Generations, say $G = 100$, is reached.
  - After certain number of generations, say $G = 50$, change in fitness function values for two consecutive generations is less than threshold value, say

  $\{(f_{G=52})-(f_{G=51}) < 10^{-6}$.

Numerical Example for GA

Use unconstrained GA to minimize the objective function:

$f(x_1, x_2) = x_1 - x_2 + 2x_1^2 + 2x_1x_2 + x_2^2$

Following additional information are provided:

(a) Boundary values of variables: $0 \leq x_1, x_2 \leq 6$.

(b) 10-bit binary encoding of variables (x_1, x_2) represent initial population, given in Table 1.

(c) Take elite count = 2, crossover probability = 0.8, mutation probability = 0.05.

(d) Random number in generation 1 for Roulette wheel selection operation, is given in Table 1 on next slide.

(e) From the mating pool, first take required numbers of consecutive strings for crossover operations and then take required number of consecutive strings for mutation operations.

(f) Assume Single-point crossover sites as 11, 9, 1, 12, 10, 17, 5 for respective crossover operations.

(g) Assume following bits have probability less than 0.05: bit 9 for mutation parent 1; bit 15 for mutation parent 2; bit 2 for mutation parent 3: bit 5 for mutation parent 4.

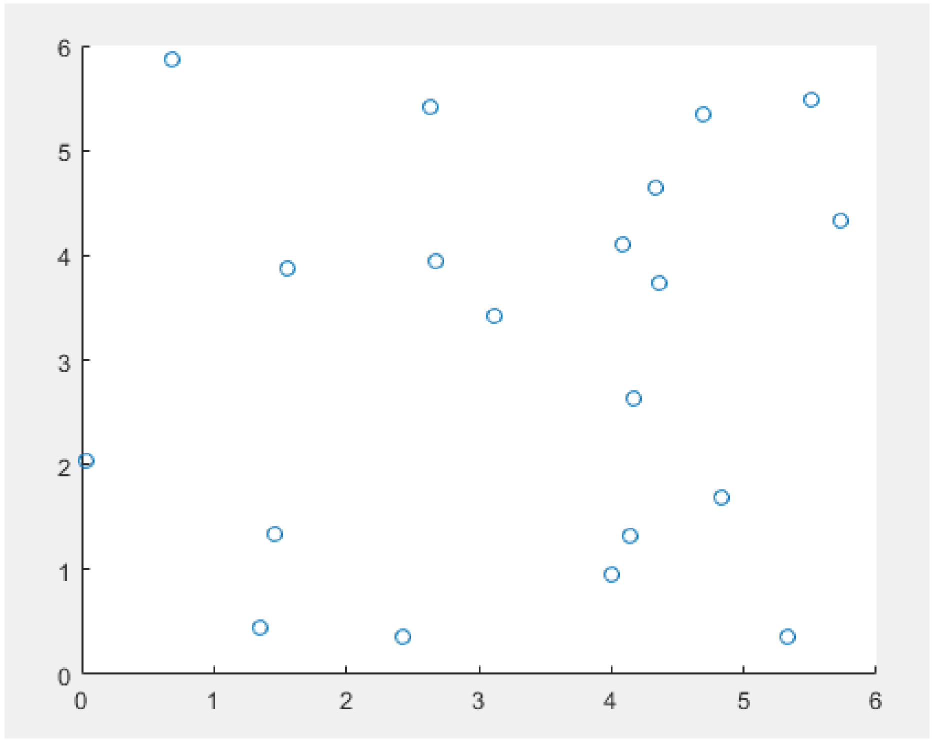
Show calculations for first generation to obtain (i) linearly mapped initial population (ii) fitness function (iii) elite children (iv) mating pool (v) crossover children (vi) mutation children (vii) next generation population.

Table 1

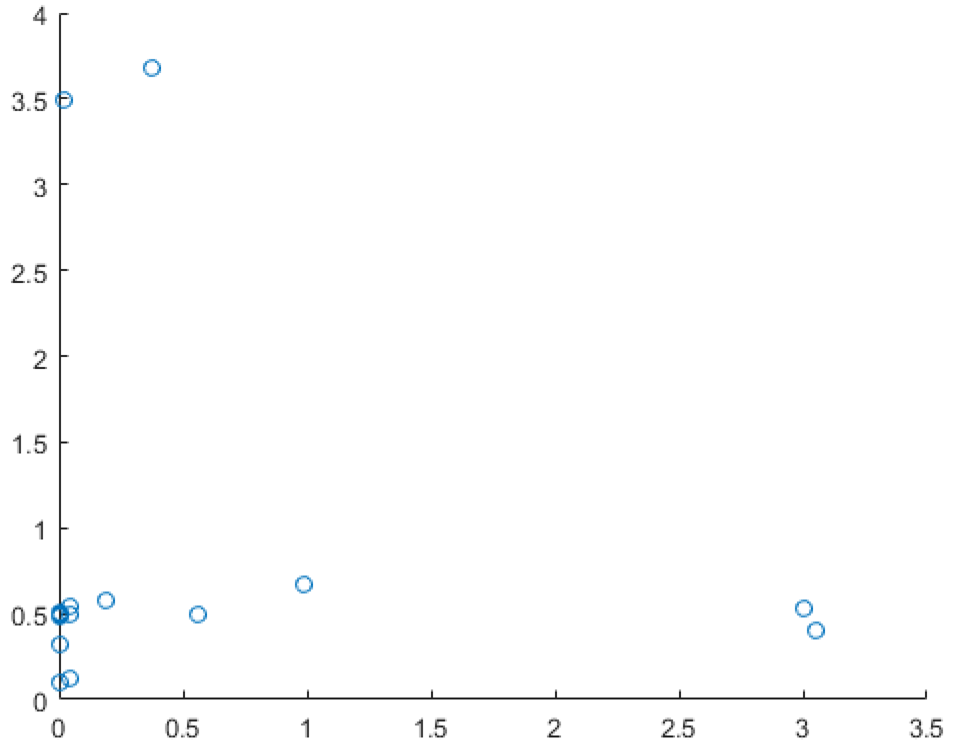
String no	Initial x_1 (binary)	Initial x_2 (binary)	random no. for Roulette wheel selection
0	1110111011	1001010000	0.612
1	1101001101	1111100011	0.441
2	1100001111	0101010011	0.713
3	0110110100	1010100111	0.923
4	1000111111	1111010101	0.245
5	1011010011	1111101011	0.651
6	1111101110	1101001110	0.885
7	1010100100	1011011101	0.775
8	1010000001	1100101111	0.348
9	0000000001	1100011000	0.812
10	1011101100	0001110110	0.745
11	1110000110	0010101010	0.029
12	0010101001	0000001010	0.164
13	1001001011	1111100011	0.786
14	0001111110	1101110111	0.887
15	1010100000	0011100001	0.326
16	0000101010	0100011100	0.734
17	1111001000	1001011000	0.639
18	0010101010	0010101011	0.998
19	1110100000	1010010100	0.008

Solution

After 30 iterations, X = [0.0 0.504], f(X) = -0.25



Initial population



Final population

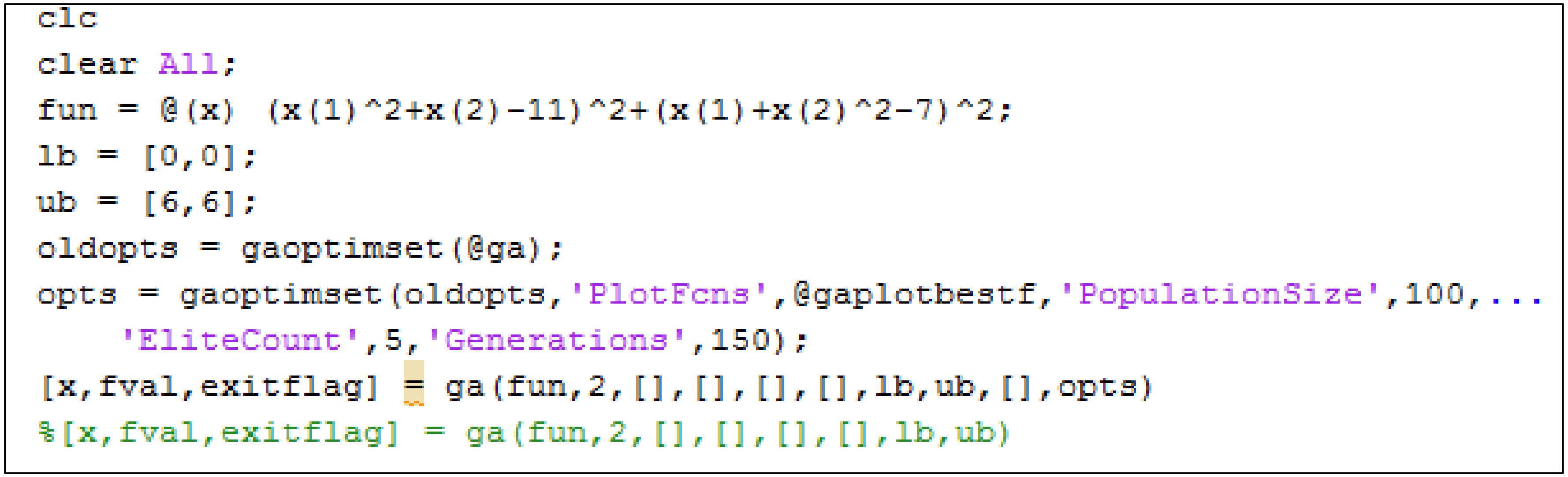
Matlab commands for GA

Problem statement:

Minimize $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$

$0 <= x_1, x_2 <= 6$

Matlab Commands

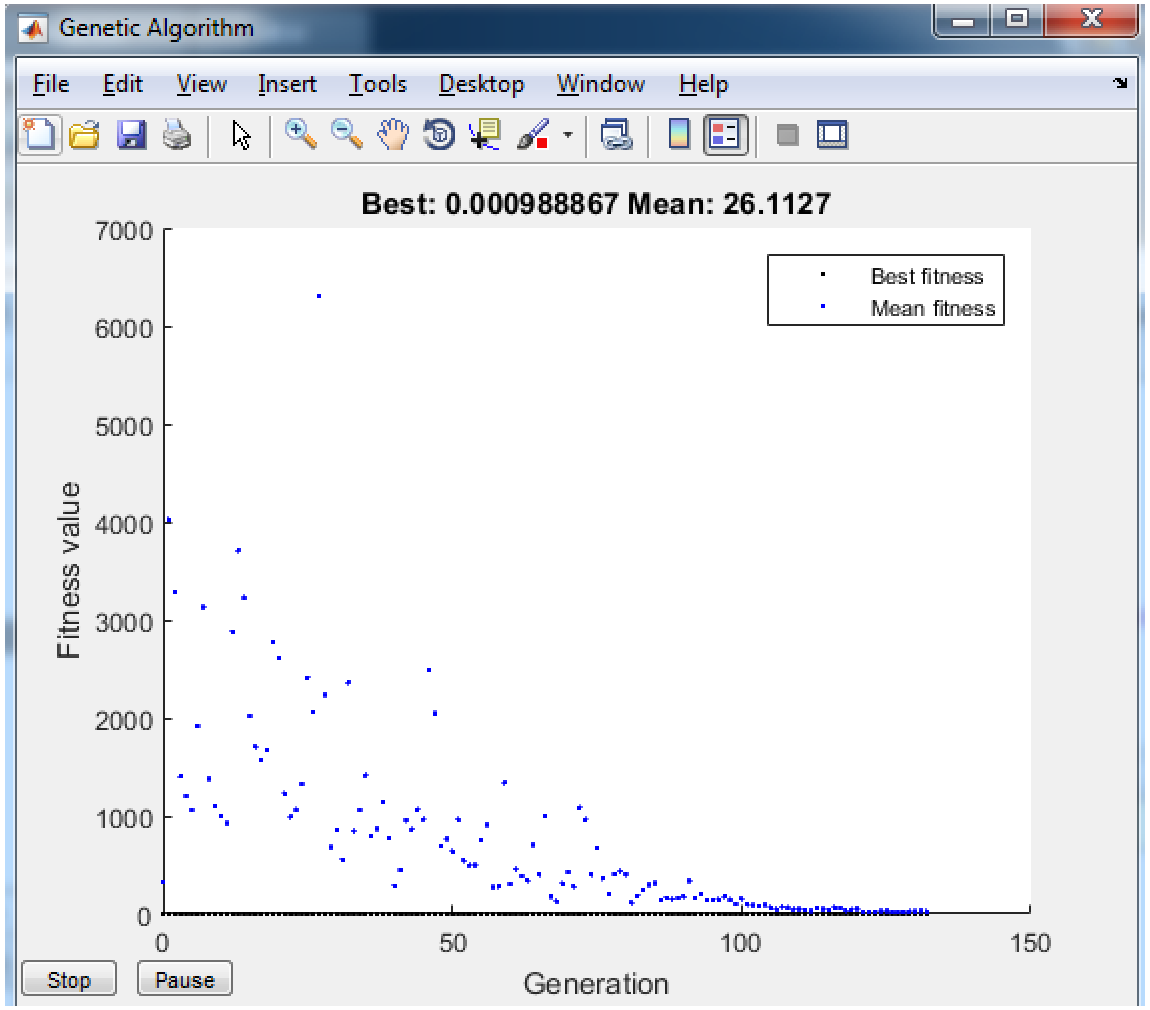


```
clc
clear All;
fun = @(x)  (x(1)^2+x(2)-11)^2+(x(1)+x(2)^2-7)^2;
lb = [0,0];
ub = [6,6];
oldopts = gaoptimset(@ga);
opts = gaoptimset(oldopts,'PlotFcns',@gaplotbestf,'PopulationSize',100,...
    'EliteCount',5,'Generations',150);
[x,fval,exitflag] = ga(fun,2,[],[],[],[],lb,ub,[],opts)
%[x,fval,exitflag] = ga(fun,2,[],[],[],[],lb,ub)
```

Outcomes of Matlab commands

Results:

```
x =

3.0023    1.9917


fval =

9.8887e-04


exitflag =

     1
```



Particle Swarm Optimization

Particle Swarm Optimization

- Particle Swarm Optimization(PSO)
  - Proposed by James Kennedy & Russell Eberhart in 1995
  - Inspired by social behavior of birds and fishes
  - Combines self-experience with social experience
  - Population-based optimization

Concept of PSO

- Uses a number of particles that constitute a swarm moving around in the search space looking for the best solution.

- Each particle in search space adjusts its “flying” according to its own flying experience as well as the flying experience of other particles.

- Swarm: a set of particles (S)
- Particle: a potential solution
  - Position: $\mathbf{x}_i = (x_{i,1}, x_{i,2}, ..., x_{i,n}) \in \Re^n$
  - Velocity: $\mathbf{v}_i = (v_{i,1}, v_{i,2}, ..., v_{i,n}) \in \Re^n$
- Each particle maintains
  - Individual local best position (LBest)
- Swarm maintains its global best (GBest)

PSO Algorithm

- Basic algorithm of PSO
  1. Initialize the swarm form the solution space
  2. Evaluate the fitness of each particle
  3. Update local and global bests
  4. Update velocity, position, and inertia of each particle
  5. Go to Step 2, and repeat until termination condition

- Original velocity update equation

$$\mathbf{v}_i(k+1) = \text{Inertia} + \text{Cognitive} + \text{Social}$$

$$\mathbf{v}_i(k+1) = w_k \times \mathbf{v}_i(k) + c_1 \times random_1() \times (LBest_i - \mathbf{x}_i(k))$$
$$+ c_2 \times random_2() \times (GBest - \mathbf{x}_i(k))$$

  – w_k is Inertia factor; c_1 is Cognitive attraction constant, c_2 is Social attraction constant
  – $random_1()$, $random_2()$: random variable
- Inertia factor for kth generation (k = 0, 1, …., G-1 where G is total number of generations)

$$w_k = w_{\min} + (w_{\max} - w_{\min})\left(1 - \frac{k}{G}\right)$$

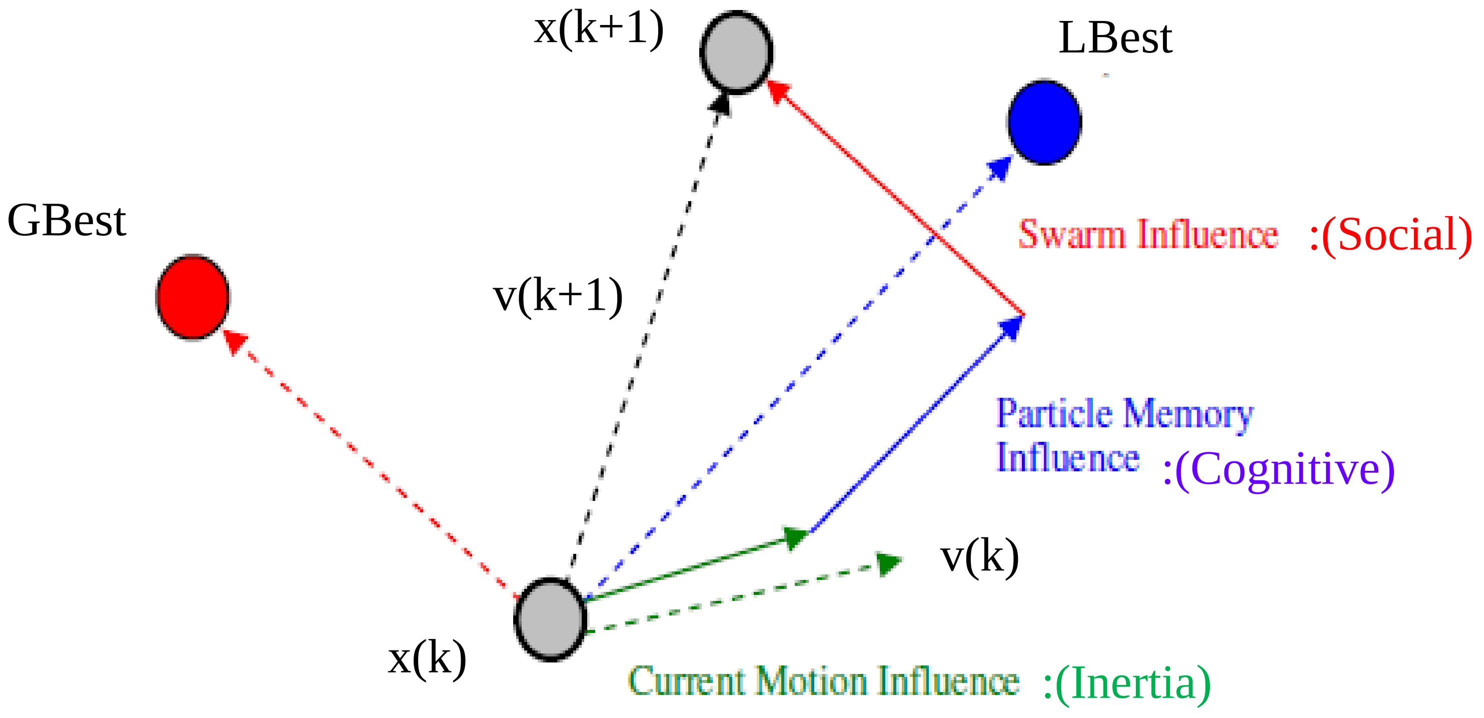
- Position update

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

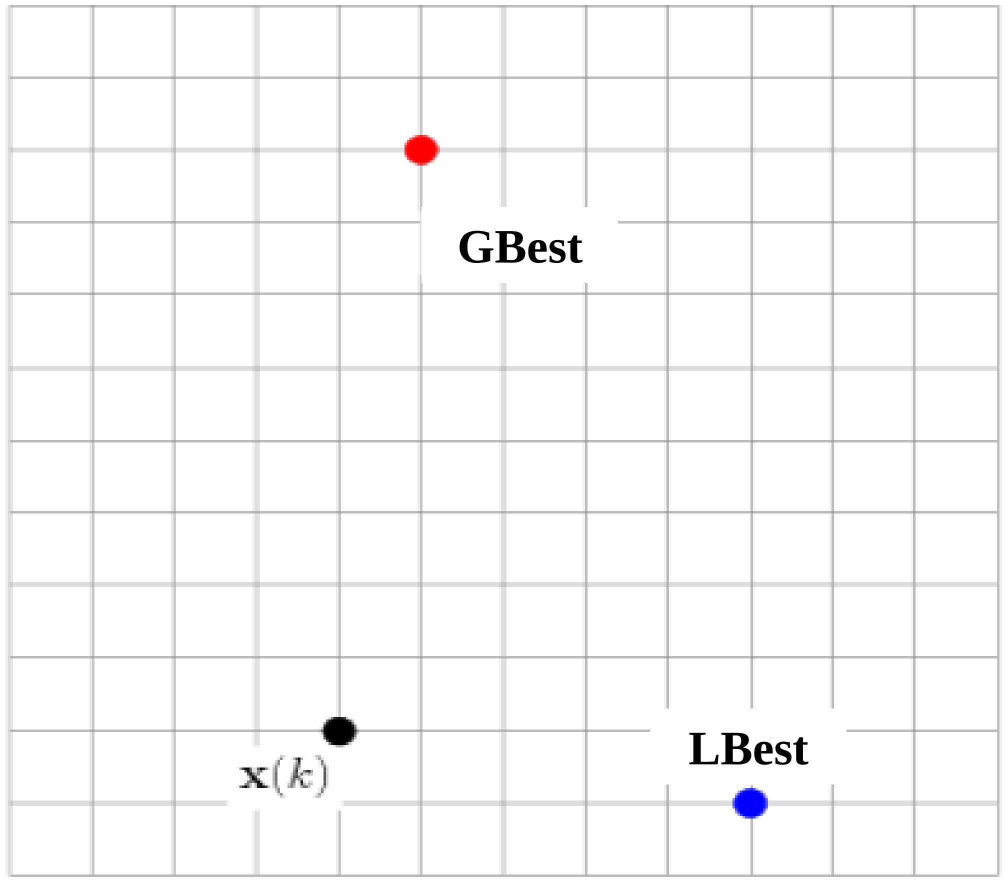
PSO Algorithm Illustration

- Particle's velocity

$$\mathbf{v}_i(k+1) = \text{Inertia} + \text{Cognitive} + \text{Social}$$

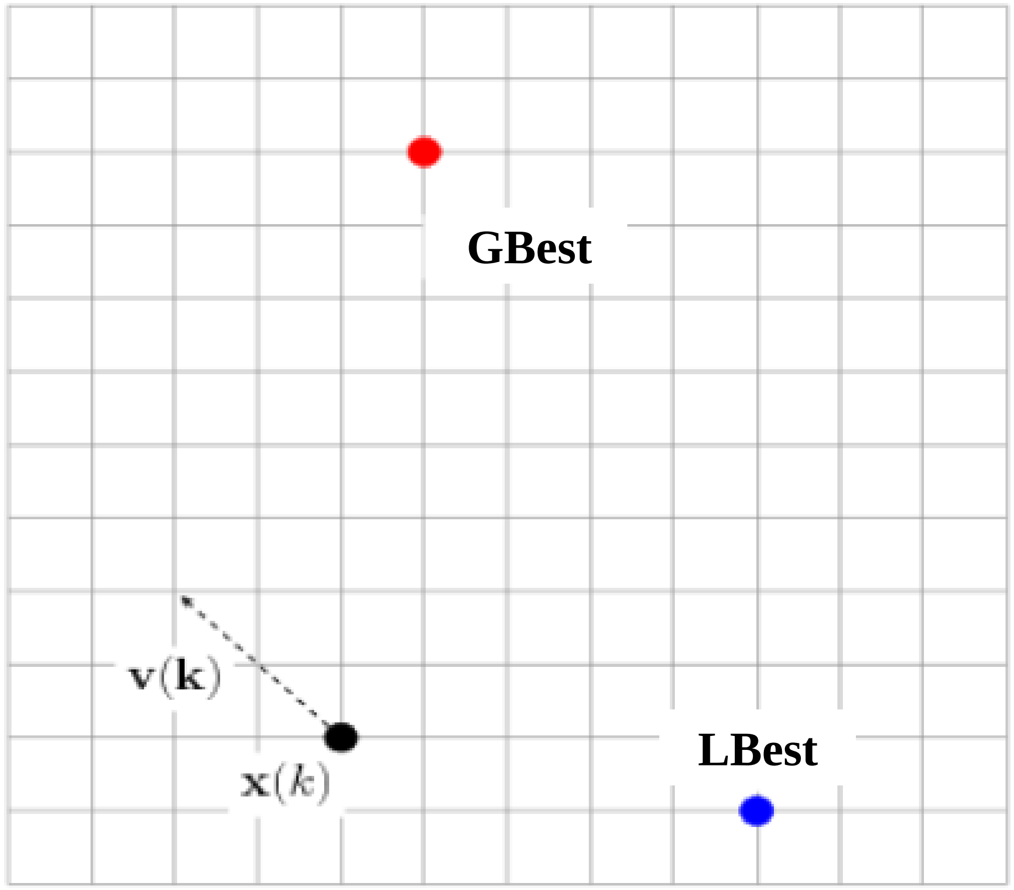

x(k+1)
LBest
GBest
v(k+1)
Swarm Influence :(Social)
Particle Memory Influence :(Cognitive)
v(k)
x(k)
Current Motion Influence :(Inertia)

PSO Update in 2D



- ● $\mathbf{x}(k)$ - Current solution (4, 2)
- ● LBest - Local best solution (9, 1)
- ● GBest-Global best solution (5, 10)

PSO Solution Update in 2D (contd.)



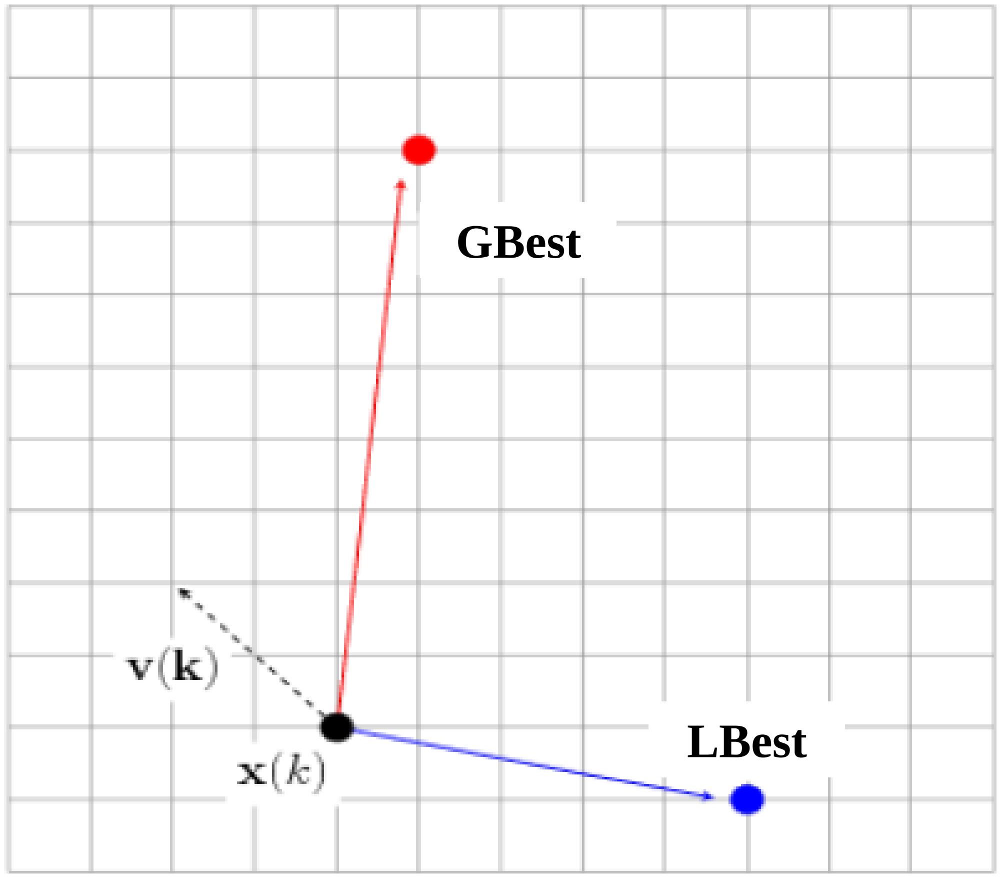
Inertia: **v**(k)=(-2, 2)

● **x**(k) - Current solution (4, 2)

● LBest - Local best solution (9, 1)

● GBest-Global best solution (5, 10)

PSO Solution Update in 2D (contd.)



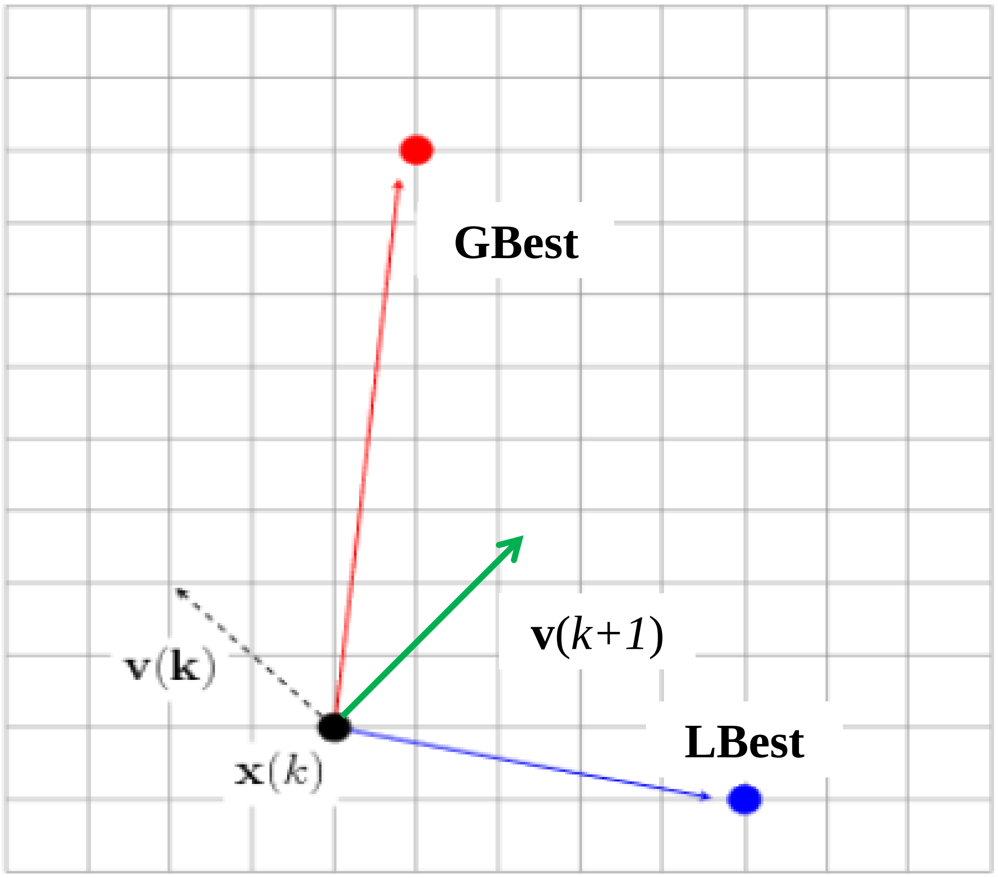
- Inertia: $\mathbf{v}(k)$=(-2,2)
- Cognitive:
  LBest-$\mathbf{x}(k)$=(9,1)-(4,2)=(5,-1)
- Social:
  GBest-$\mathbf{x}(k)$=(5,10)-(4,2)=(1,8)

● $\mathbf{x}(k)$ - Current solution (4, 2)

● LBest - Local best solution (9, 1)

● GBest-Global best solution (5, 10)

PSO Solution Update in 2D (contd.)

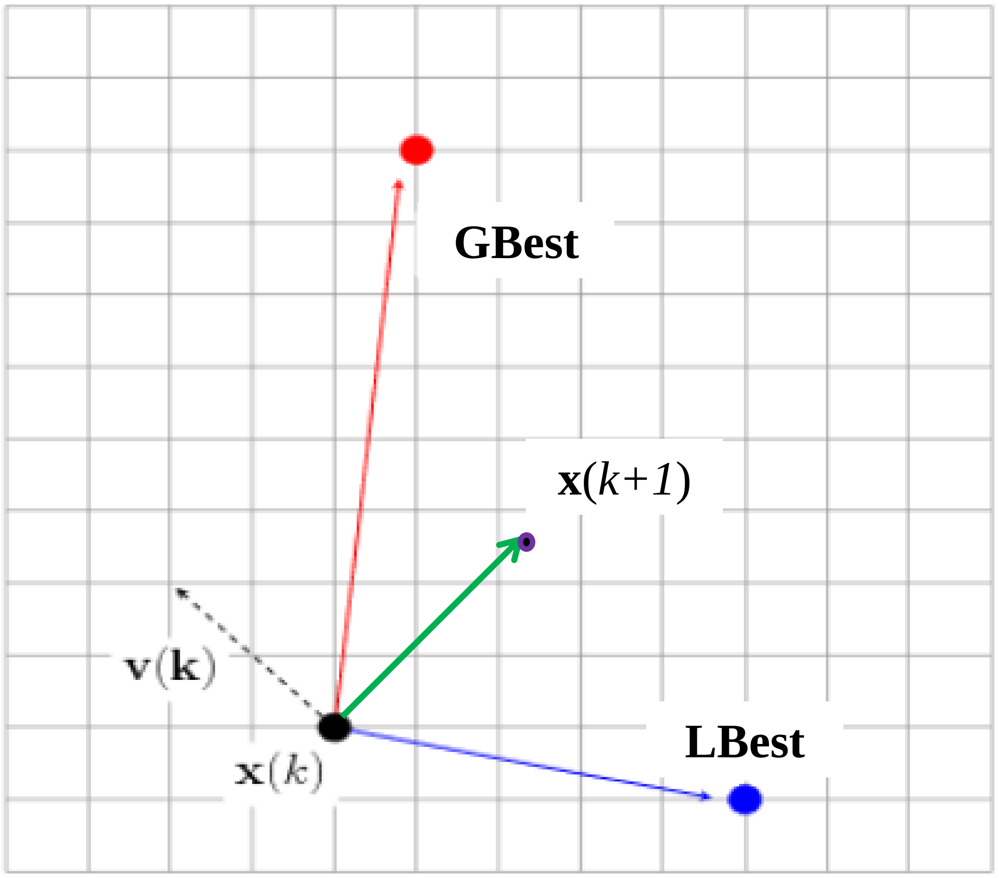


- Inertia: $\mathbf{v}(k)=(-2,2)$
- Cognitive: LBest-$\mathbf{x}(k)=(9,1)-(4,2)=(5,-1)$
- Social: GBest-$\mathbf{x}(k)=(5,10)-(4,2)=(1,8)$

$$\mathbf{v}(k+1) = (-2,2)+0.8*(5,-1)+0.2*(1,8) = (2.2,2.8)$$

●	$\mathbf{x}(k)$ - Current solution (4, 2)
●	LBest - Local best solution (9, 1)
●	GBest-Global best solution (5, 10)

PSO Solution Update in 2D (contd.)



- Inertia: $\mathbf{v}(k)=(-2,2)$
- Cognitive:
  LBest-$\mathbf{x}(k)=(9,1)-(4,2)=(5,-1)$
- Social:
  GBest-$\mathbf{x}(k)=(5,10)-(4,2)=(1,8)$
- $\mathbf{v}(k+1)=(2.2,2.8)$

$$\mathbf{x}(k+1) = \mathbf{x}(k)+\mathbf{v}(k+1) = (4,2)+(2.2,2.8)=(6.2,4.8)$$

● $\mathbf{x}(k)$ - Current solution (4, 2)

🔵 LBest - Local best solution (9, 1)

🔴 GBest-Global best solution (5, 10)

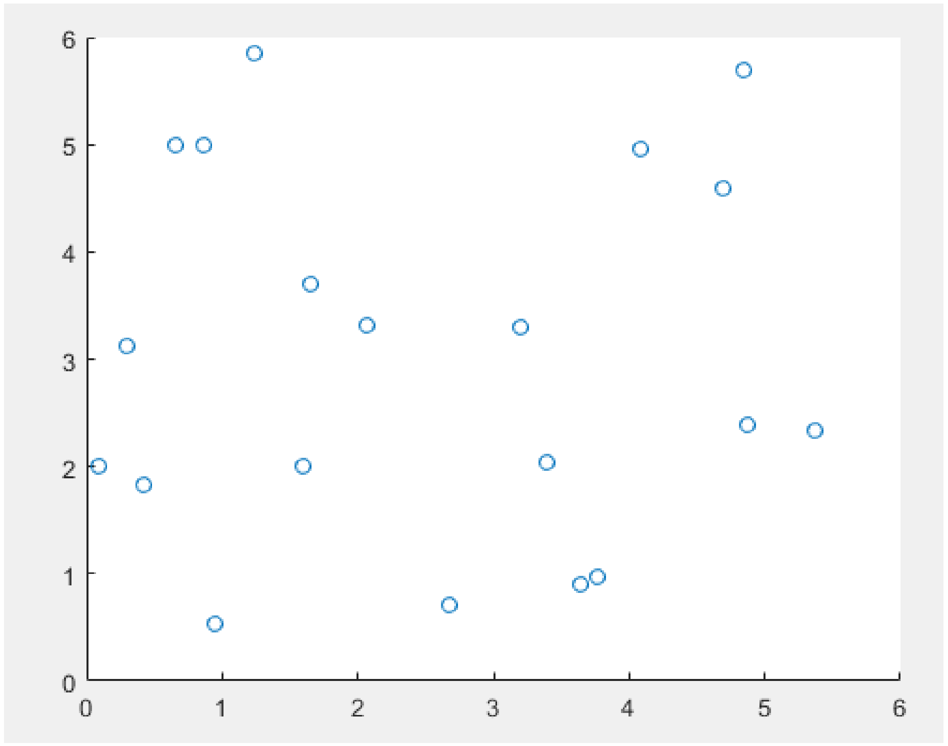
Numerical Example

- Minimize:
- $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2+(x_1 + x_2^2 - 7)^2$
- $0< x_1 , x_2 < 6$
- $c_1 = c_2 = 1.05$
- $w_{\max} = 1.0$, $w_{\min} = 0.4$
- Maximum number of iterations = 30

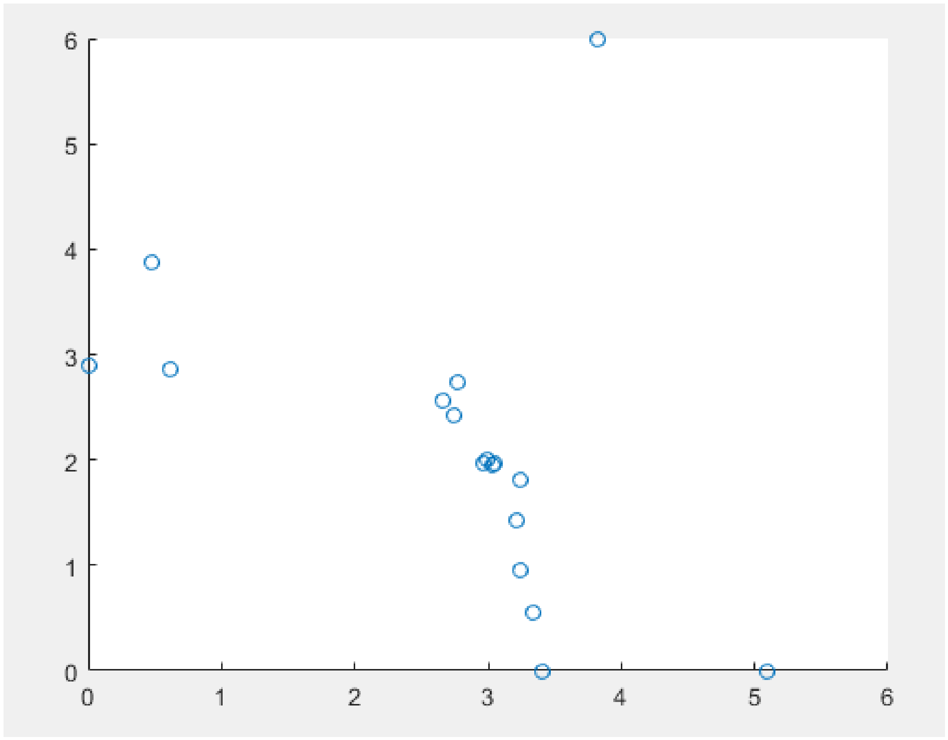
Numerical Example - Solution

- After 30 iterations
- $f(x_1, x_2) = 0.0070$
- $x_1 = 3.0047$, $x_2 = 1.9778$

Initial points



Final points



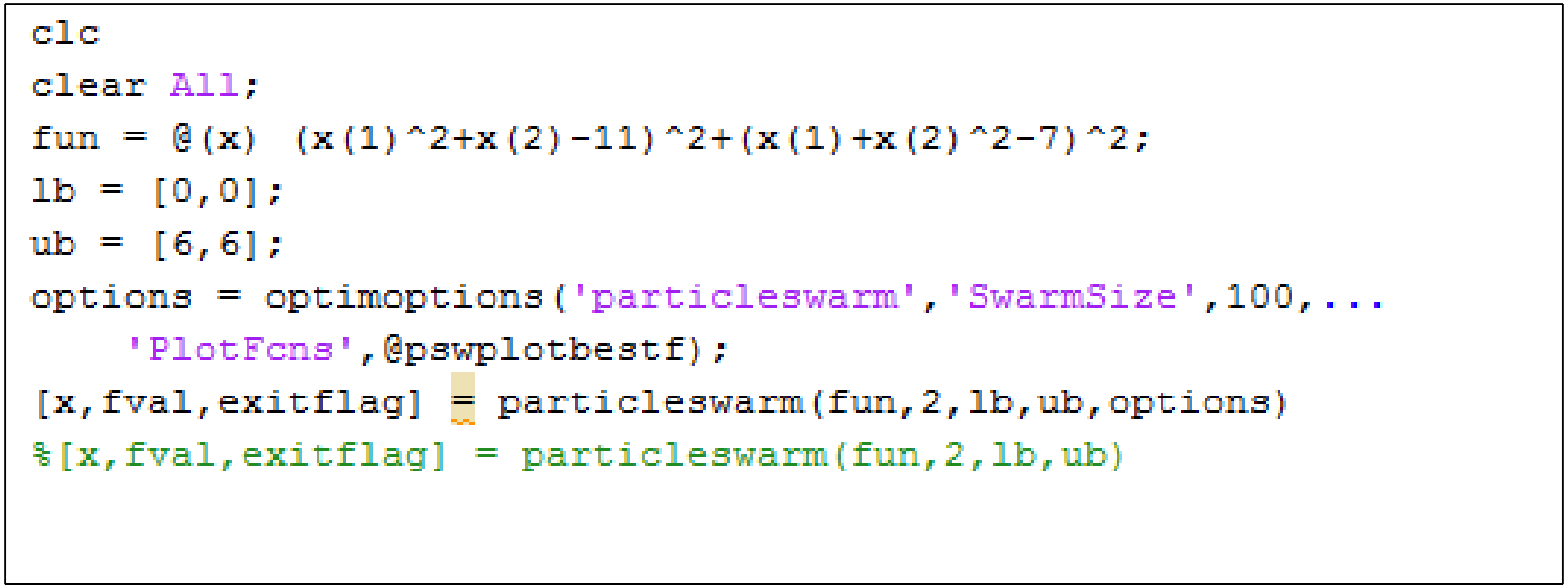
Matlab commands for PSO

Problem statement:

Minimize $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$

$0 <= x_1, x_2 <= 6$

Matlab Commands

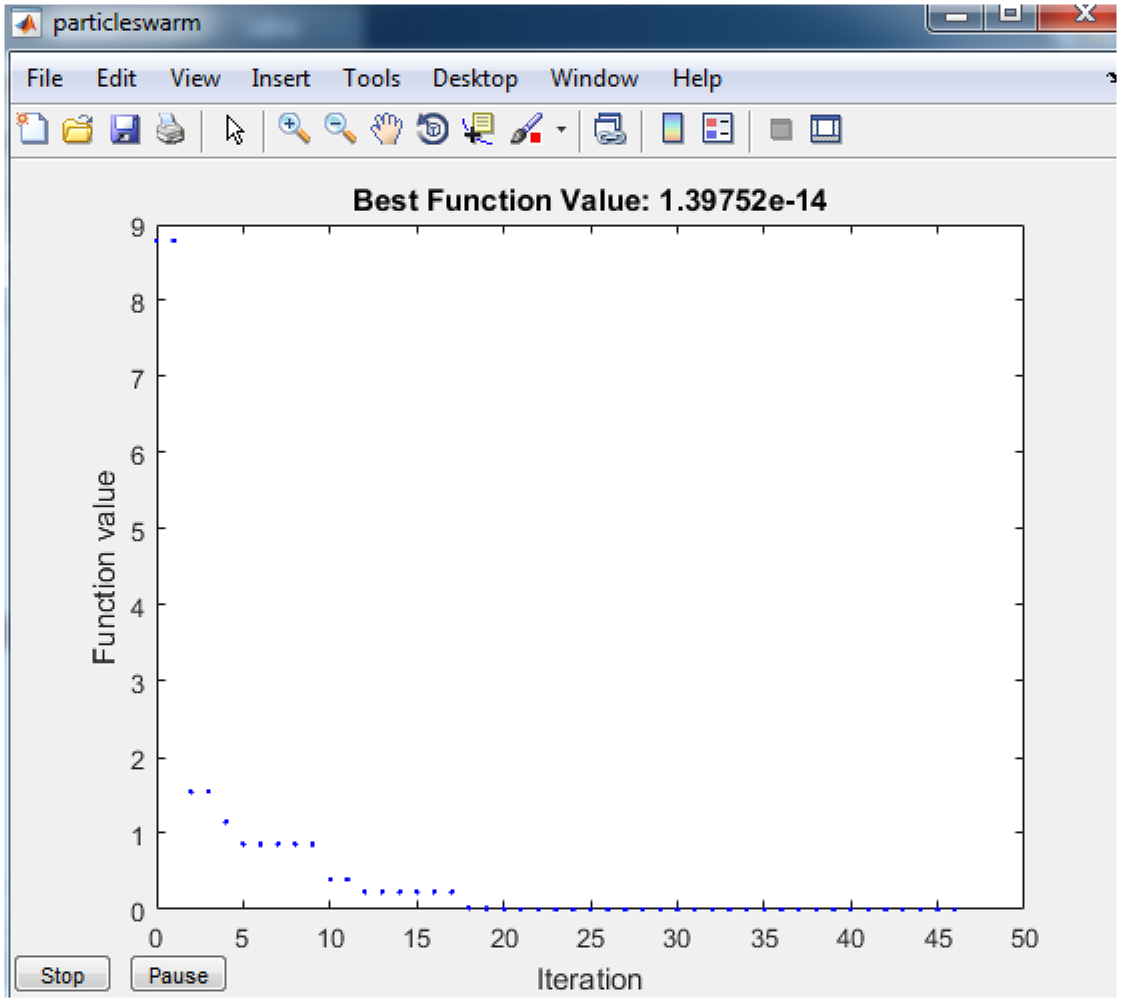


```
clc
clear All;
fun = @(x)  (x(1)^2+x(2)-11)^2+(x(1)+x(2)^2-7)^2;
lb = [0,0];
ub = [6,6];
options = optimoptions('particleswarm','SwarmSize',100,...
    'PlotFcns',@pswplotbestf);
[x,fval,exitflag] = particleswarm(fun,2,lb,ub,options)
%[x,fval,exitflag] = particleswarm(fun,2,lb,ub)
```

Outcomes of Matlab commands

Results:

```
x =

3.0000    2.0000


fval =

1.3975e-14


exitflag =

1
```



Traditional Optimization (Steepest Descent)

Gradient of a Function

- The gradient of a function $f(x_1, x_2, \ldots, x_n)$ is given by

$$\nabla f = \begin{Bmatrix} \partial f / \partial x_1 \\ \partial f / \partial x_2 \\ \vdots \\ \partial f / \partial x_n \end{Bmatrix}$$

- Important properties of gradient:
  - If we move along the gradient direction from any point in n-dimensional space, the function value increases at the fastest rate. Hence the gradient direction i.e. **S** = ∇f is called the *direction of steepest ascent* .
  - If we move against the gradient direction from any point in n-dimensional space, the function value decreases at the fastest rate. Hence the opposite of gradient direction i.e. **S** = $-\nabla f$ is called the *direction of steepest descent* .

- Any method that makes use of the gradient vector can be expected to give the minimum point faster than one that does not make use of the gradient vector.
- All the descent methods make use of the gradient vector, either directly or indirectly, in finding the search directions.
- Direction of steepest ascent/descent is a local property, not a global one.

Fig. 1.5 shows steepest ascent directions.

The gradient vectors ∇f evaluated at points 1, 2, 3, and 4 lie along the directions 11′, 22′, 33′, and 44′, respectively.

The function value increases at the fastest rate in the direction 11′ at point 1, but not at point 2. Similarly, the function value increases at the fastest rate in direction 22′ or 33′ at point 2 or 3, but not at point 3 or 4 respectively.

In other words, the direction of steepest ascent generally varies from point to point, and if we make infinitely small moves along the direction of steepest ascent, the path will be a curved line like the curve 1–2–3–4 in Fig. 1.5.

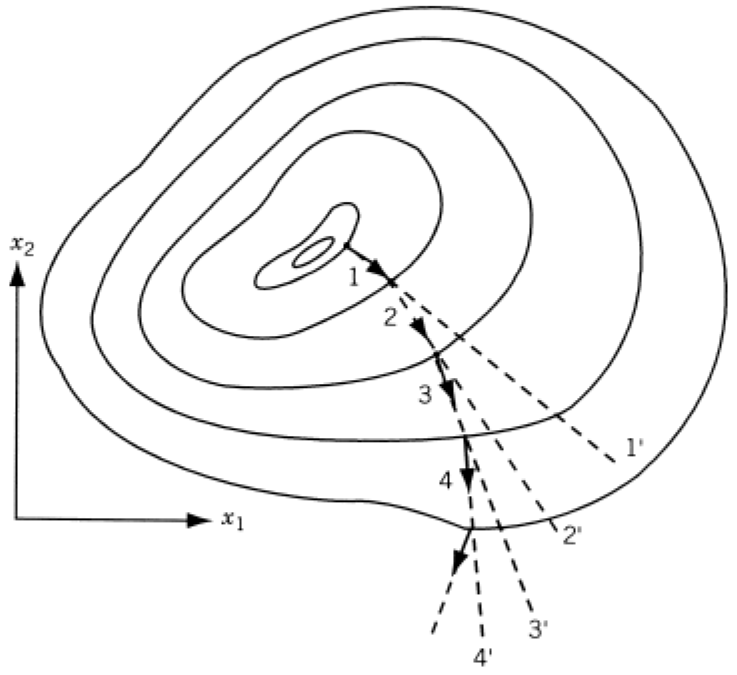


Figure 1.5 Steepest ascent directions

Steepest Descent Method

- Use of the negative of the gradient vector as a direction for minimization was first made by Cauchy in 1847.
- Steps for Steepest Descent method:
    1. Select an initial arbitrary point $\mathbf{X}^{(1)}$. Set the iteration number $i := 1$.
    2. For ith iteration, calculate direction of gradient descent of the function as follows: $\mathbf{S}^{(i)} = -\nabla f(\mathbf{X}^{(i)})$.
    3. Set the next search point as follows: $\mathbf{X}^{(i+1)} = \mathbf{X}^{(i)} + \alpha^{(i)}\,\mathbf{S}^{(i)}$ and determine the optimal step size $\alpha^{(i)}$ (>0) such that $\frac{df\left(\mathbf{X}^{(i+1)}\right)}{d\alpha^{(i)}} = 0$, or $f(\mathbf{X}^{(i+1)})$ is minimum in the direction $\mathbf{S}^{(i)}$.

4. Test the new point for optimality using one of the following criterion:

   (a) When the change in function value in two consecutive iterations is small i.e. $|f(\mathbf{X}^{(\boldsymbol{i}+1)})-f(\mathbf{X}^{(\boldsymbol{i})})| \leq \varepsilon_1$.

   (b) When the components of the gradient ∇f are small i.e. $|\partial f/\partial x_j| \leq \varepsilon_1$ where $j = 1, 2, ..., n$.

   (c) When the change in the design vector in two consecutive iterations is small i.e. $|\mathbf{X}^{(\boldsymbol{i}+1)}-\mathbf{X}^{(\boldsymbol{i})}| \leq \varepsilon_1$.

   If $\mathbf{X}^{(\boldsymbol{i}+1)}$ is optimum, then stop the process. Otherwise, go to Step 5.

5. Set the new iteration number $i := i + 1$ and go to Step 2.

Note: Steps for Steepest Ascent are same as those, except Step 2, for Steepest Descent. Set direction gradient ascent as $\mathbf{S}^{(\boldsymbol{i})} = \nabla f(\mathbf{X}^{(\boldsymbol{i})})$.

Illustrating Effect of Step Size on Gradient Descent

Objective function: $f(x) = x^2$

Initial point: $x = 1$

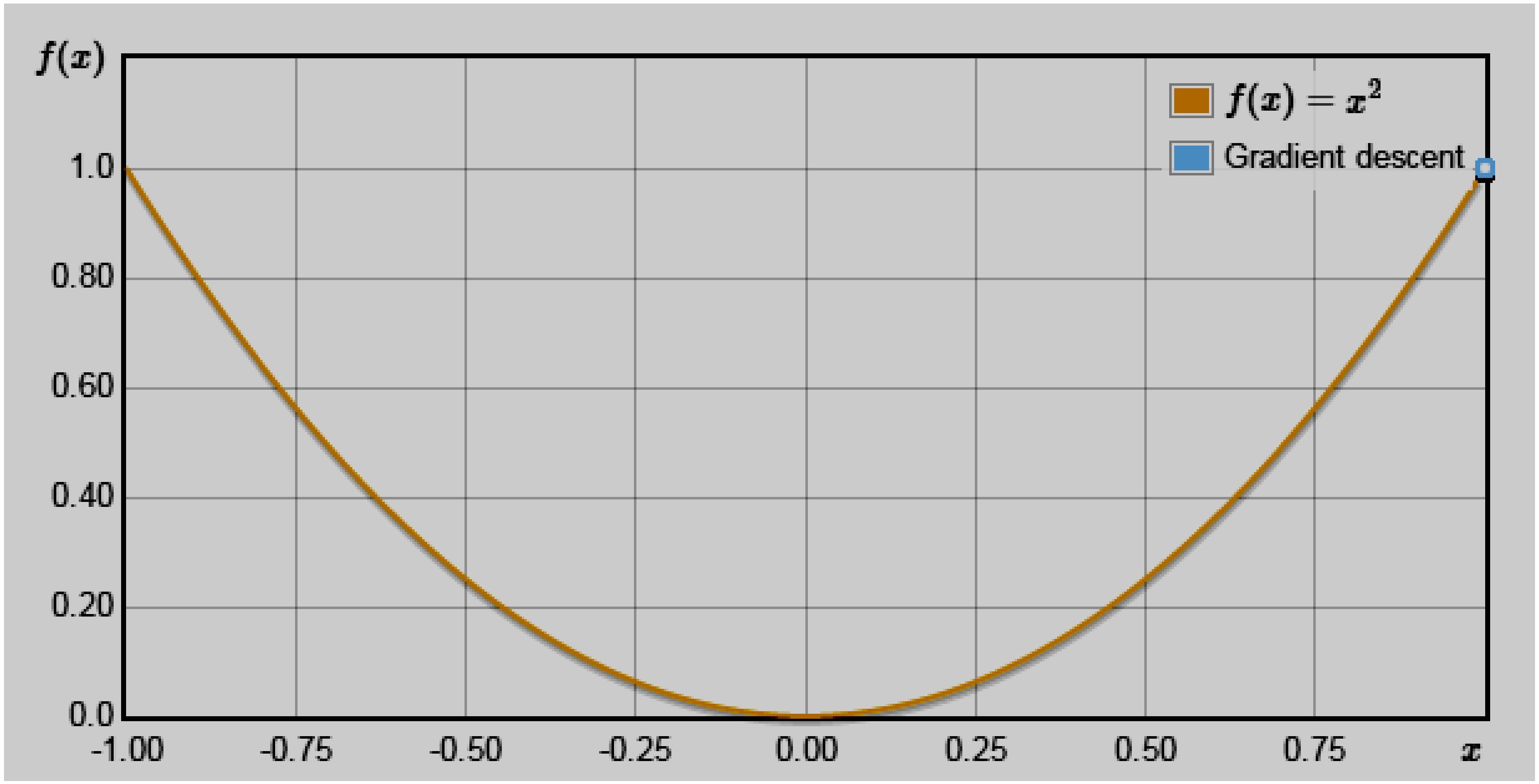
Variable step size: $\alpha = 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$

Optimal step size: $\alpha = 0.5$

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 0$

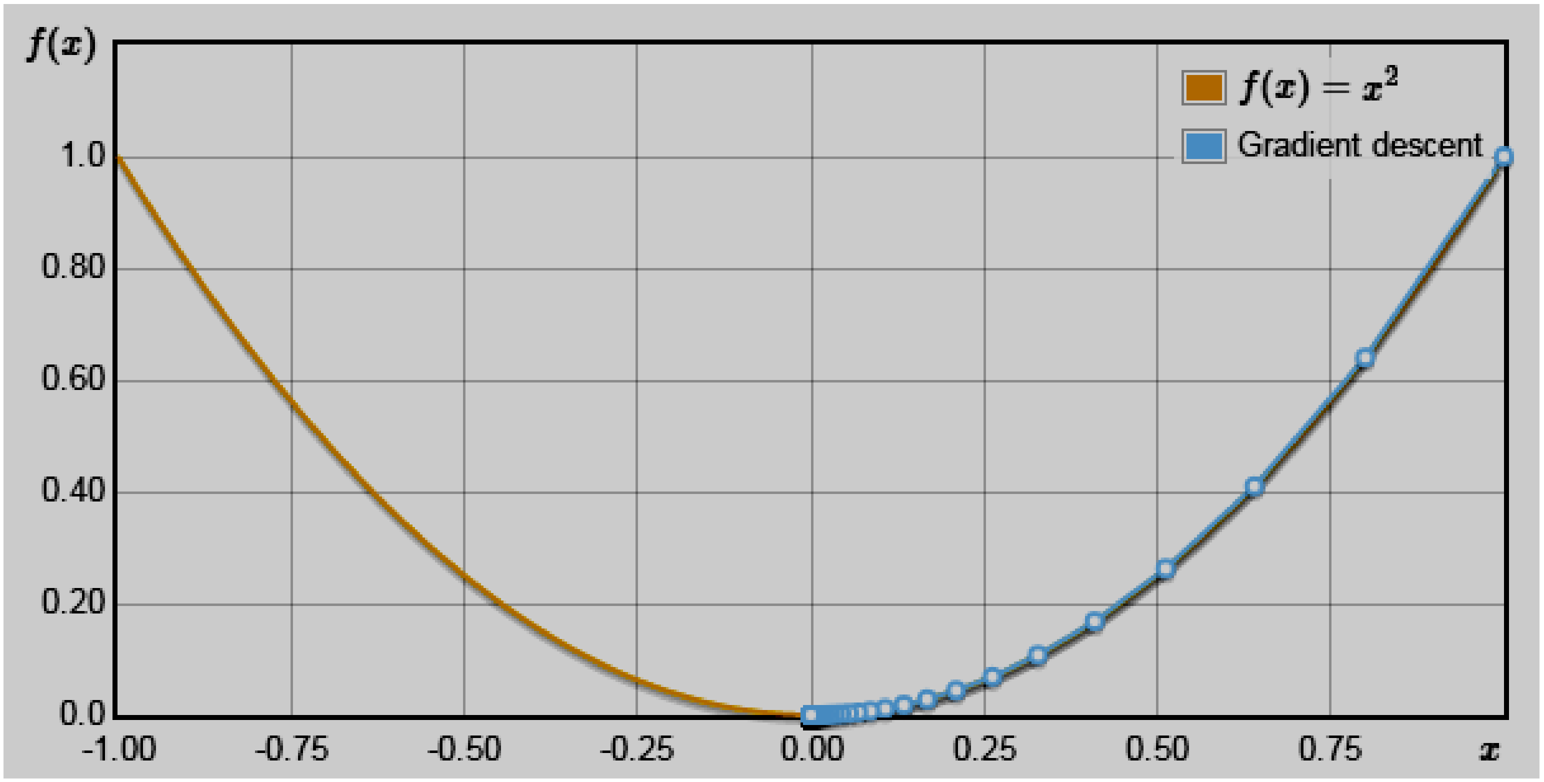


Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 0.1$

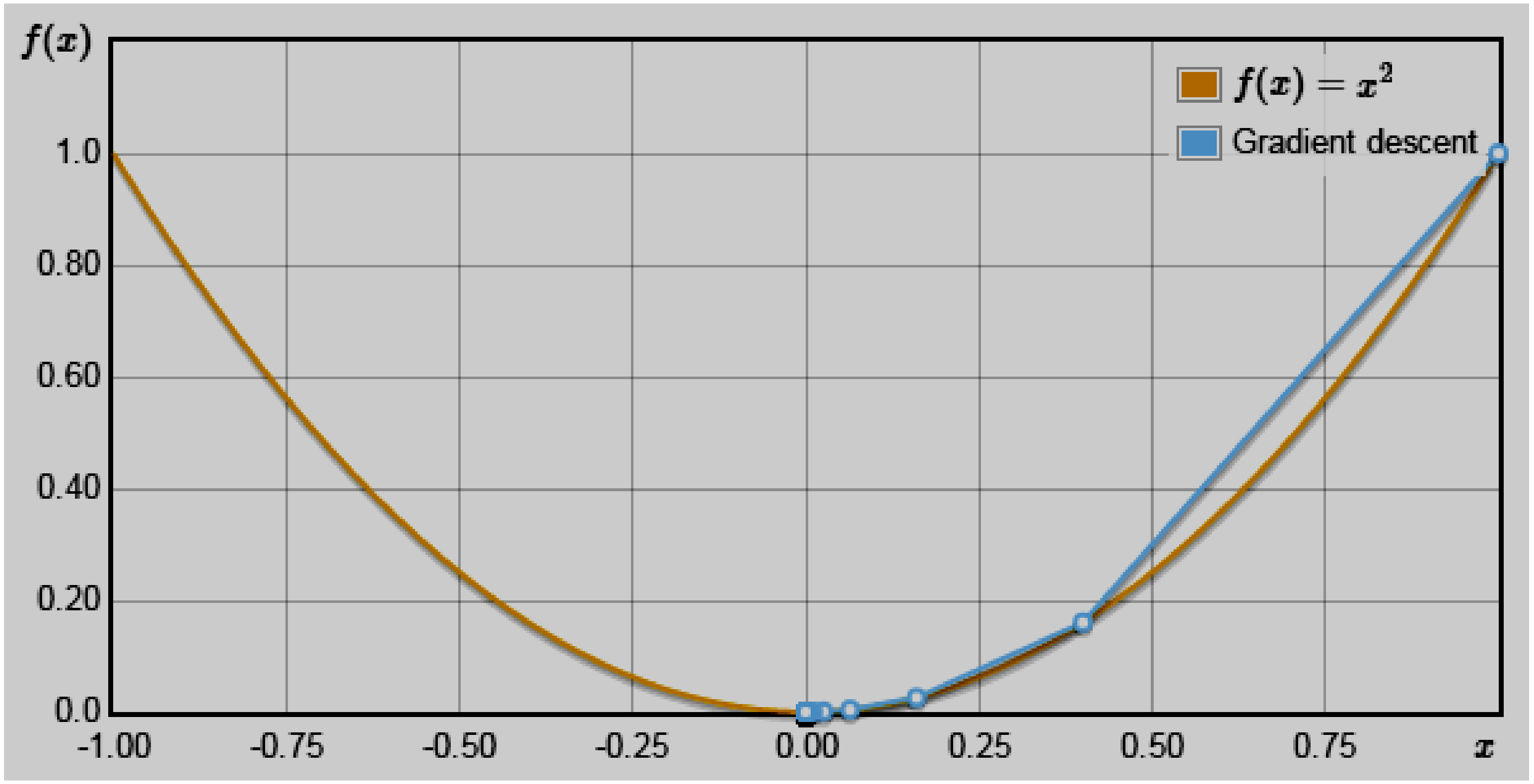


Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 0.3$

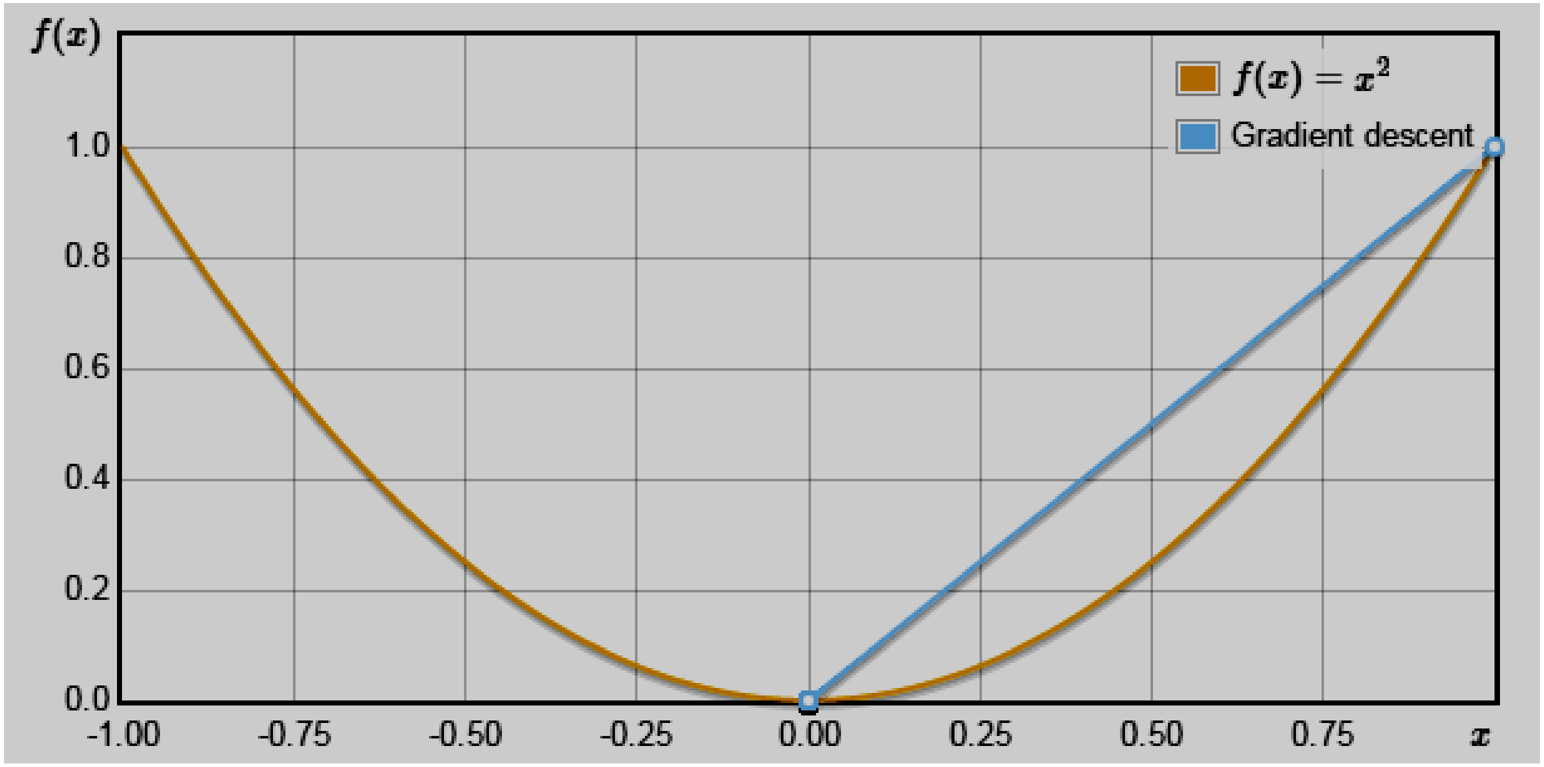


Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 0.5$ (optimal)

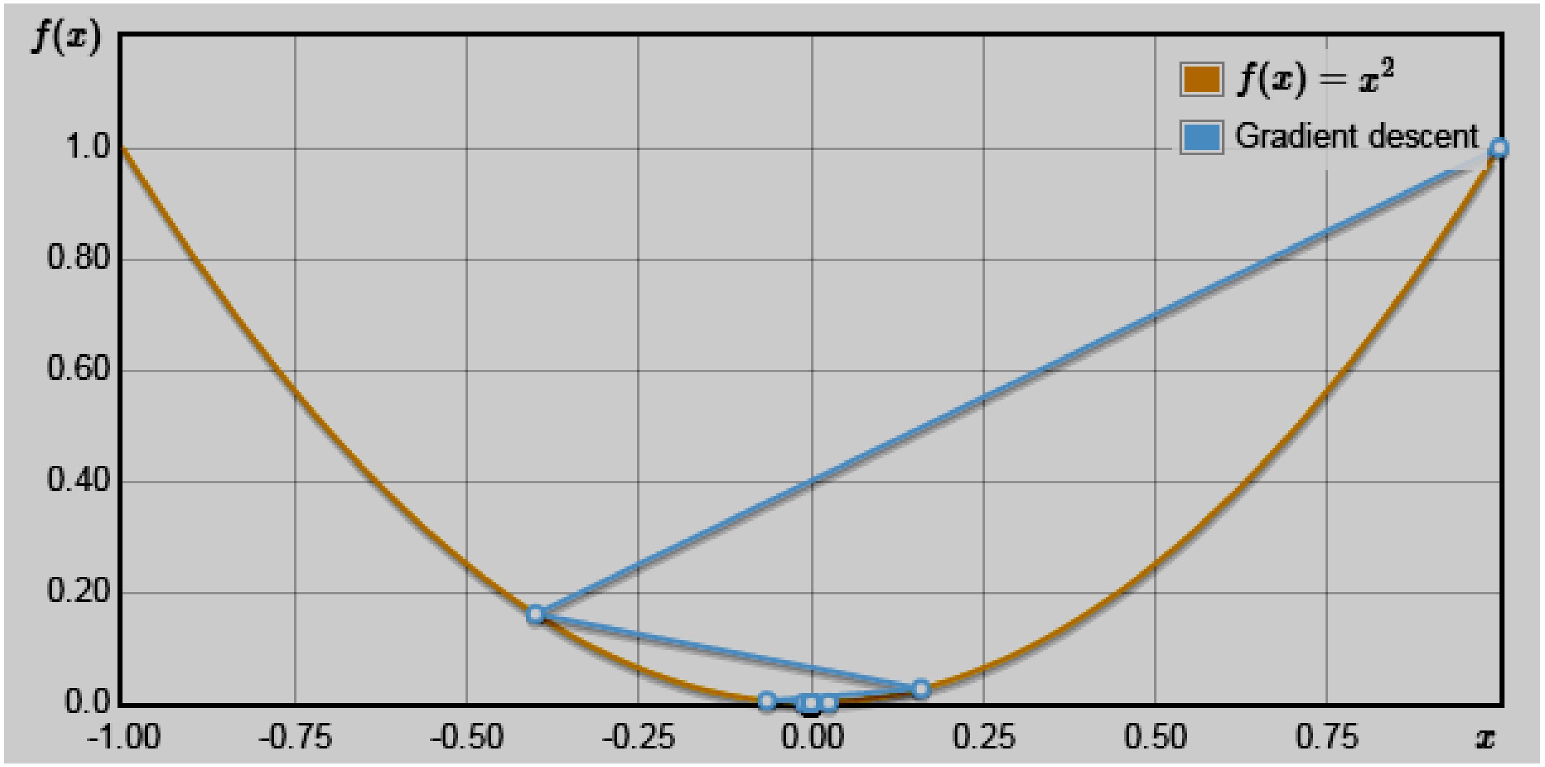


Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 0.7$

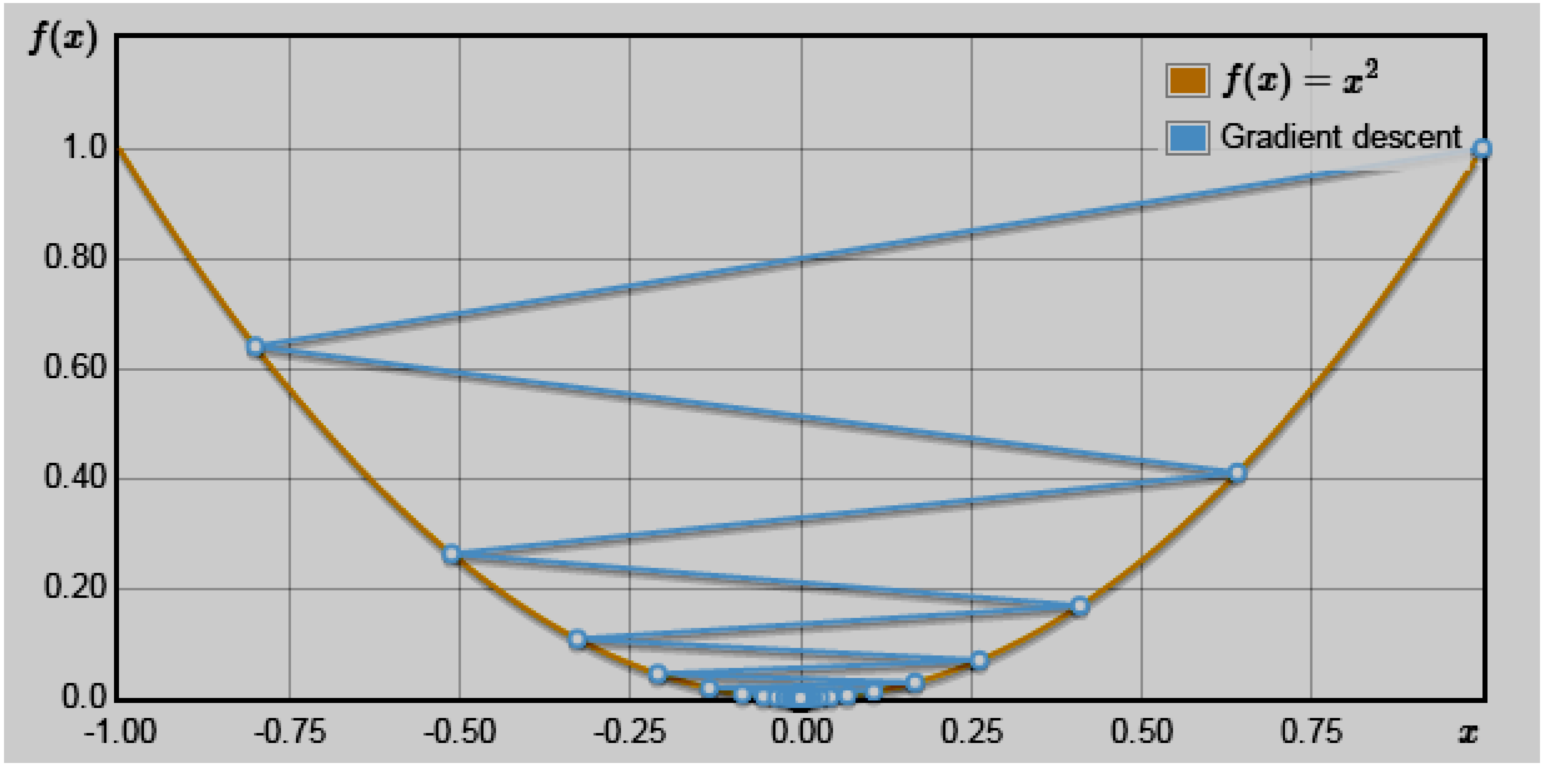


Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 0.9$

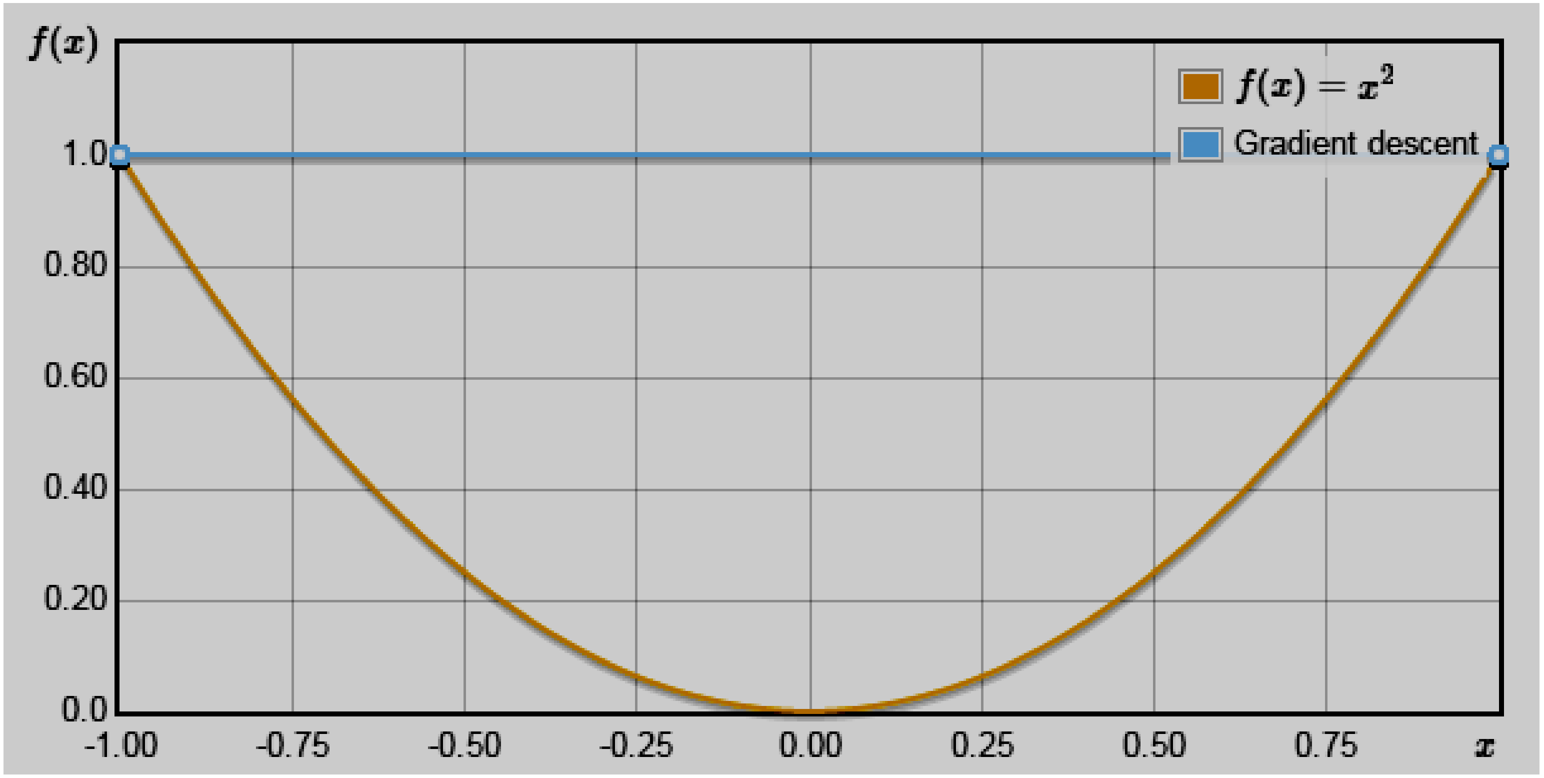


Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Objective function: $f(x) = x^2$

Initial point: $x = 1$

Step size: $\alpha = 1$



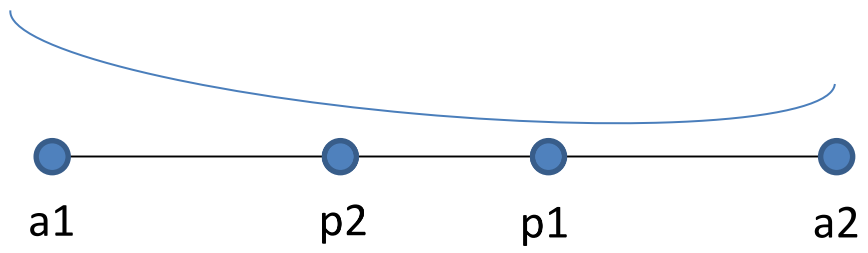
Courtesy: http://www.onmyphd.com/?p=gradient.descent&ckattempt=1

Step 3 of Steepest Descent Method

- Perform unidirectional search to find optimum $\alpha^{(i)}$ such that $f(\mathbf{X}^{(i+1)})$ or $f(\mathbf{X}^{(i)} + \alpha^{(i)}\,\mathbf{S}^{(i)})$ is minimum
- Unidirectional search methods:
  1. Bisection method
  2. Fibonacci Search method
  3. Golden Section Search method

     (Fast and less computationally intensive search method)

Golden Section Search Method



- Steps:
  - 1. Let search space (range of $\alpha^{(i)}$) be (a1,a2)
  - 2. p1 = a1+0.618*(a2-a1), p2 = a2-0.618*(a2-a1)
  - 3. $X_1^{i+1} = X^i + p1 \times S^i$, $X_2^{i+1} = X^i + p2 \times S^i$
  - 4. Calculate $f(X_1^{i+1})$ *and* $f(X_2^{i+1})$
  - 5. If $f(X_1^{i+1}) < f(X_2^{i+1})$ then let new search space be (p2,a2) else let new search space be (a1,p1)
  - 6. If a1~a2 < ϵ_2 then stop, α^i = (a1+a2)/2 else go to step 2

Steepest descent converges to one of local minima as shown in Fig. 1.7. Convergence to local minima depends on initial point as well as step size

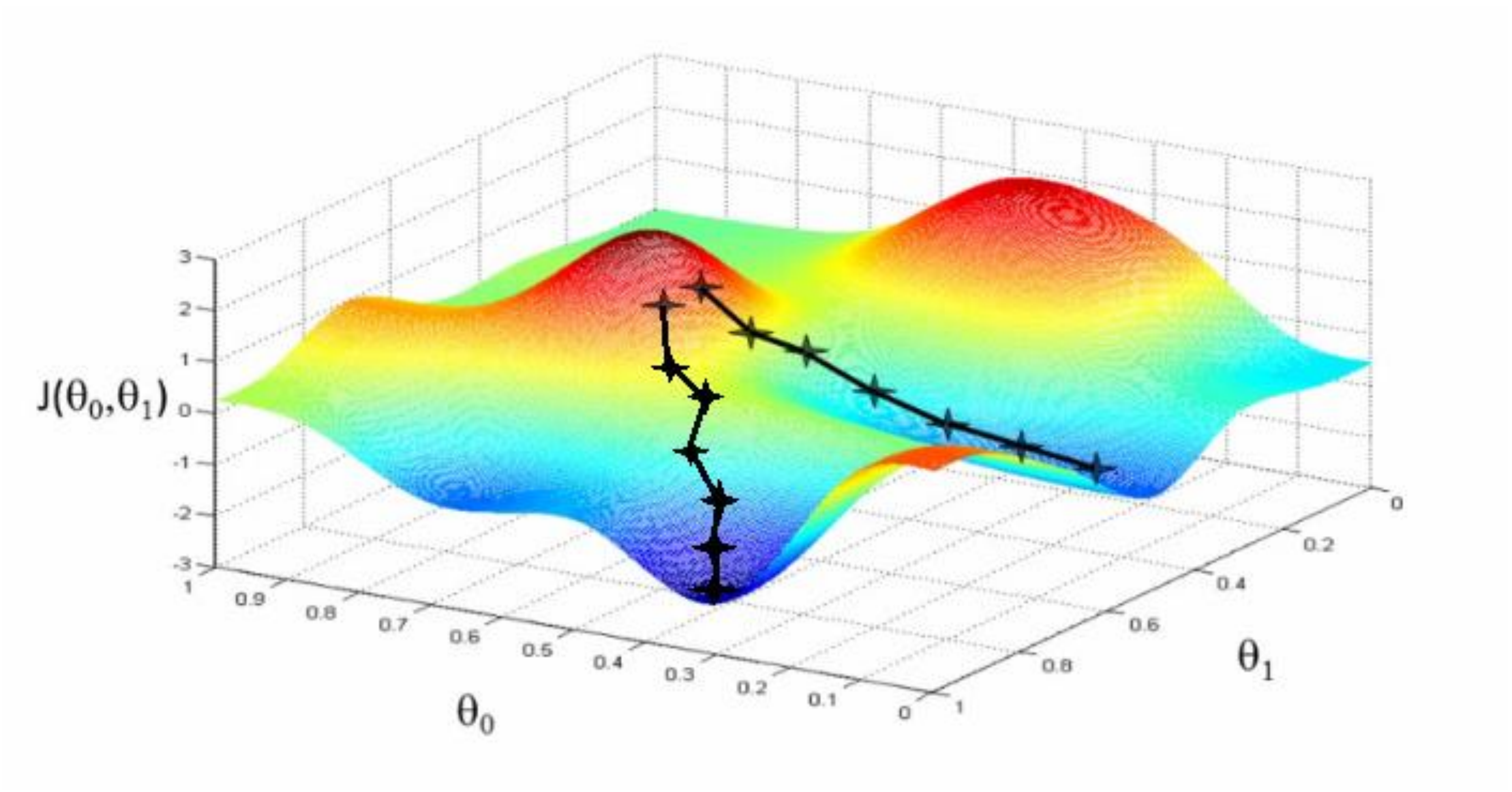


Figure 1.7 Surface with multiple local minima

Numerical Example for Steepest Descent Method

Find the optimum by the method of steepest descent for an objective function :

$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2+(x_1 + x_2^2 - 7)^2$.

The starting point of the search has the co-ordinate (0, 0).

$\epsilon_1 = \epsilon_2 = 0.001$

Iteration 1

- i = 0
- $X^0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
- Step 2

Let h = 0.001

$$\frac{\partial f}{\partial x_1} = \frac{f(x1+h, x2) - f(x1-h, x2)}{2h} = \text{-14}$$

$$\frac{\partial f}{\partial x_2} = \frac{f(x1, x2+h) - f(x1, x2-h)}{2h} = \text{-22}$$

$$S^0 = \begin{bmatrix} 14 \\ 22 \end{bmatrix}$$

Iteration 1

- Step 3: Find optimum α^0 such that $f(\mathbf{X^1})$ or $f(\mathbf{X^0} + \alpha^0\, \mathbf{S^0})$ is minimum using Golden Section Method
- Iteration 1 of GSM
  - 1. Range of α^0 be (0,1)
  - 2. p1 = 0.618, p2 = 0.382
  - 3. $X_1^1 = \begin{bmatrix} 8.652 \\ 13.596 \end{bmatrix}$ and $X_2^1 = \begin{bmatrix} 5.348 \\ 8.404 \end{bmatrix}$
  - 4. $f(X_1^1)$= 40782.4 *and* $f(X_2^1)$ = 5433.8
  - 5. $f(X_1^1) > f(X_2^1)$, so new search space is (0,0.618)
  - 6. convergence criterion not met, go to iteration 2 of GSM
- After many (~15) such iterations α^0 = 0.127
- $X^1 = \begin{bmatrix} 1.783 \\ 2.801 \end{bmatrix}$
- Step4: convergence criterion not met, so go to next iteration

Solution

Iteration	**S**	$\boldsymbol{\alpha}$	**X**	**f(X)**
1	$[14\ 22]^T$	0.127	$[1.783\ 2.801]^T$	32.126
2	$[30.542\ -19.428]^T$	0.040	$[2.999\ 2.028]^T$	0.0126
3	$[-0.471\ -0.936]^T$	0.017	$[2.991\ 2.012]^T$	0.00329
4	$[0.442\ -0.217]^T$	0.020	$[3.000\ 2.007]^T$	0.00088
5	$[-0.115\ -0.244]^T$	0.017	$[2.998\ 2.003]^T$	0.00024
6	$[0.117\ -0.059]^T$	0.020	$[3.000\ 2.002]^T$	0.00006
7	$[-0.034\ -0.066]^T$	0.017	$[2.999\ 2.001]^T$	0.00002