Optimization

Neural Networks & Fuzzy Logic

(BITS F312)

Introduction

- The method to find the 'best' solution out of several feasible solutions
- Express the criterion to judge the goodness of a solution as a function of factors which influence the chosen criterion
- Point at which the function takes its minimum or maximum value is called a minimizer or maximizer respectively

Mathematically,

find $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ to (decision/design vector/variables)

minimize/ maximize $f(\underline{x})$ (objective(cost/fitness) function)

subject to

$g_i(\underline{x}) = 0 \;\; i = 1,2,\dots m$ (equality constraints)

$h_j(\underline{x}) \leq 0 \;\; j = 1,2,\dots p$ (inequality constraints)

and

$L_i \leq x_i \leq U_i \quad i = 1,2,\dots n$ (bounds)

Various Types of Optimization Problems:

1) Unconstrained and Constrained

2) Single and Multiobjective

3) Static and Dynamic

4) Linear Programming and Nonlinear Programming

5) Integer Programming and Real Valued Programming

Traditional & Nontraditional/Metaheuristic Techniques

- Based on pure mathematics
- Yields precise solutions
- Suitable for problems which are easy to model mathematically
- May not be suitable to solve complex real-world problems
- They complement each other

- Maximization problems can be converted to minimization problems by modifying the objective function as

$$-f(\underline{x})$$

$$\frac{1}{f(\underline{x})} \quad for \;\; f(\underline{x}) \neq 0$$

$$\frac{1}{1+f(\underline{x})} \quad for \;\; f(\underline{x}) \geq 0$$

$$\frac{1}{1+\{f(\underline{x})\}^2} \quad \text{etc.}$$

- Following operations on the objective function will not change the optimum solution *x**

 1. Multiplication/division of *f*(*x*) by a positive constant *c*.
 2. Addition/subtraction of a positive constant *c* to/from *f*(*x*).

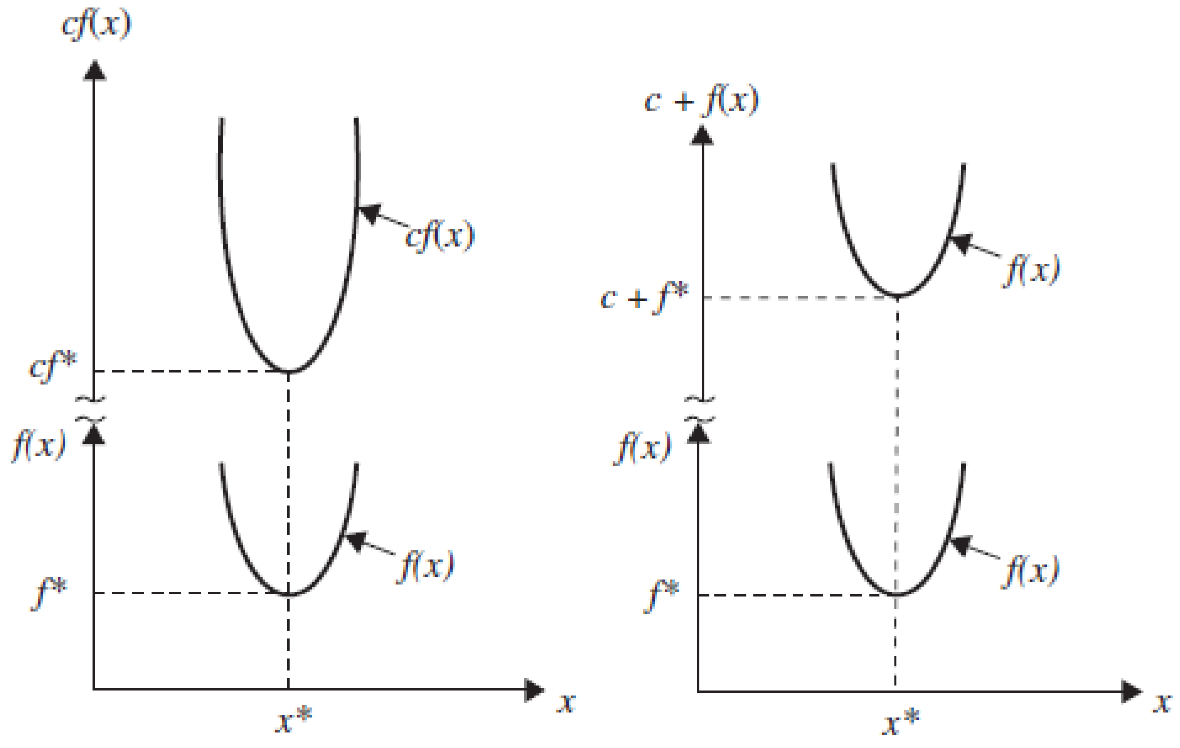


Figure 1.2 Optimum solution of $cf(x)$ or $c + f(x)$ same as that of $f(x)$.

Applications

- Optimization problems arise in almost all fields where numerical information is processed (science, engineering, mathematics, economics, commerce, etc.)

- Relevant wherever technical or managerial decision making is involved or a trade off is involved

- Engineering design applications usually have constraints since the variables can not take arbitrary values and usually we want the design to be best in some sense like minimum cost or maximum efficiency etc.

For example, while designing a bridge, an engineer will be interested in minimizing the cost, while maintaining a certain minimum strength for the structure.

- Control Engg. example: some popular performance indices
- Estimation Theory (E.g. Kalman Filter)

- Many other problems can be cast as optimization problems

(E.g. root finding; solution of overdetermined/underdetermined linear systems of equations etc.)

A Bit of History

- Originated in Greece during Euclid (~300 BC)
- Classical analytical methods date back to the era of Cauchy, Bernoulli, Euler and Lagrange (also the Calculus of Variations)
- The beginnings of the subject of **operations research** can be traced to the early period of World War II
- In WW-II, the allied forces faced the problem of allocating limited resources such as fighter airplanes, warships and submarines for deployment to various targets and destinations. Gave birth to linear programming technique.

- During WW-II, Alan Turing used some heuristic search to break the German Enigma codes
- The second wave came in 1970’s partly due to rapid growth in computing power
- Various nontraditional techniques to solve engineering optimization problems started to emerge from 1990’s
- These techniques are mostly inspired from nature or from how some biological systems function

Analytical Method: Unconstrained Case

If $f(\underline{x})$ has an extremum at $\underline{x}^*$ then

$$\left.\frac{\partial f}{\partial x_1}\right|_{\underline{x}^*} = \left.\frac{\partial f}{\partial x_2}\right|_{\underline{x}^*} = \cdots \left.\frac{\partial f}{\partial x_n}\right|_{\underline{x}^*} = 0$$

and

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\,\partial x_n} \\ \frac{\partial^2 f}{\partial x_2\,\partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\,\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\,\partial x_1} & \frac{\partial^2 f}{\partial x_n\,\partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} . = \text{ PD or ND } \quad \text{(minimum/maximum)}$$

Saddle point if neither PD nor ND

Analytical Method: Constrained Case (Equality Constraints)

Lagrange Multiplier Method:

minimize $f(\underline{x})$

subject to $g_j(\underline{x}) = 0 \;\; j = 1,2, \ldots m$

Lagrangian, $\mathrm{L}(\underline{x}\,, \underline{\lambda}) = f(\underline{x}) + \lambda_1 g_1(\underline{x}) + \lambda_2 g_2(\underline{x}) + \cdots + \lambda_m g_m(\underline{x})$

$$\frac{\partial L}{\partial x_i} = 0 \quad i = 1, 2, \ldots\ n$$

$$\frac{\partial L}{\partial \lambda_j} = 0 \quad j = 1, 2, \ldots m$$

Physical Meaning of Lagrange Multipliers:

a measure of how rigid the constraints are

$$df^* = \lambda^* \; db$$

Inequality Constraints (Kuhn-Tucker Conditions)

$$g_j(\underline{x}) \leq 0 \qquad j = 1,2,\ldots m$$

$$g_j(\underline{x}) + y_j^2 = 0 \qquad j = 1,2,\ldots m$$

introducing nonnegative slack variables

Checking Second Order Conditions are not as important!

An Example: Matrix Pseudoinverse

$$2x_1 + 3x_2 + x_3 = 4$$

$$A\,\underline{x} = \underline{b}$$

$$\underline{x} = (A^T A)^{-1} A^T\ \underline{b} \quad \text{or} \quad A^T (A\,A^T)^{-1}\,\underline{b}$$

minimize $x_1^2 + x_2^2 + x_3^2$

subject to $2x_1 + 3x_2 + x_3 - 4 = 0$

Ans.: $\underline{x}^* = \begin{bmatrix} 0.57 \\ 0.86 \\ 0.29 \end{bmatrix}$

Numerical Method: Unconstrained Problem

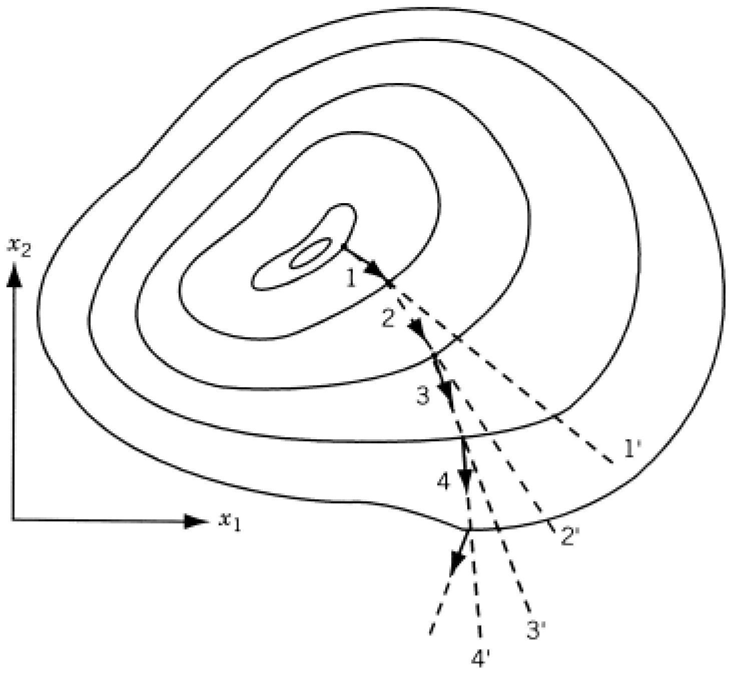
- Steepest Descent Method (Gradient based method)

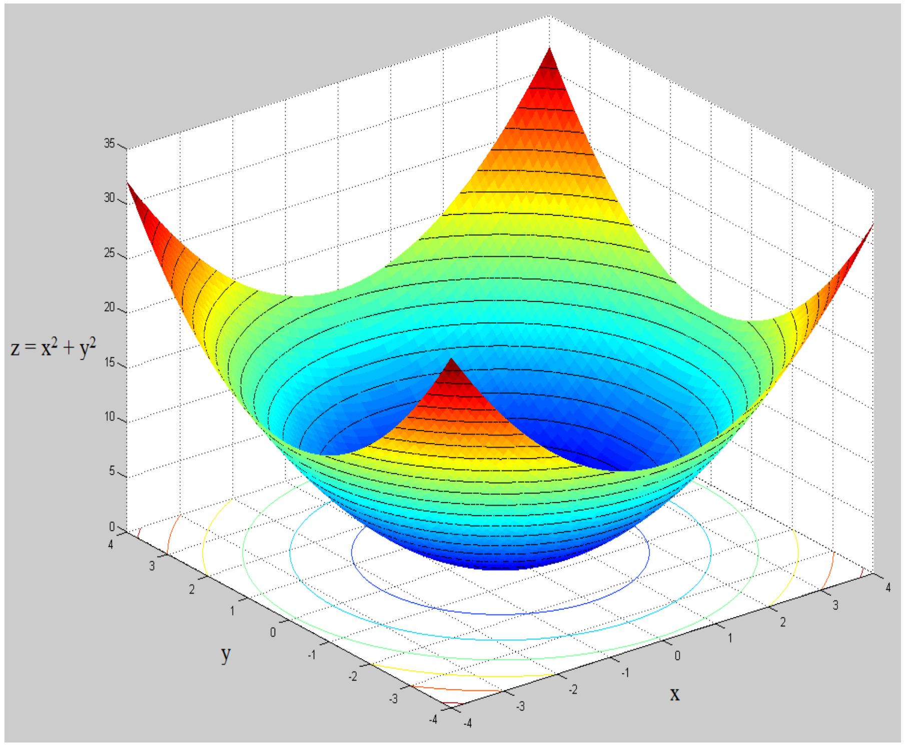
- Cauchy (1847)

$$\text{minimize} \quad f\left(\underline{x}\right) \quad where\ \underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- At any point on the objective function, the function decreases at the maximum rate along its negative gradient

- Move along this direction iteratively in small steps





Steepest ascent directions

- Steps for Steepest Descent method:

1. Select an initial arbitrary point $\underline{x}(1)$
2. For ith iteration, calculate the search direction as

$$\underline{S}(i) = -\nabla f\,(\underline{x}(i)).$$

3. Set the next search point as

$$\underline{x}(i+1) = \underline{x}(i) + \lambda_i\, \underline{S}(i) \qquad (\lambda_i > 0)$$

The step size λ_i can be optimized by solving $\frac{d}{d\lambda_i}\left(f\,(\underline{x}(i) + \lambda_i \underline{S}(i))\right) = 0$

4. Test the new point for optimality using some stopping criterion.

If $\underline{x}(i)$ is optimum, then stop the process

Otherwise, iterate

- Stopping Criteria

  i) change in the decision variable becomes small

$$\left|x_j(i+1) - x_j(i)\right| \leq \varepsilon \qquad for\ \ j = 1,2,\ldots n$$

  ii) normalized change in the objective function is small

$$\left|\frac{f(i+1)-f(i)}{f(i)}\right| \leq \varepsilon$$

  iii) gradient vector (slope) becomes small

$$\left|\frac{\partial f}{\partial x_j}(i)\right| \leq \varepsilon \qquad for\ \ j = 1,2,\ldots n$$

An Example:

minimize $f = -10 \cos x_1 \cos x_2 \; e^{-\left\{\frac{(x_1-1)^2}{4} + (x_2-2)^2\right\}}$

$-5.0 \le x_1 \le 5.0\,; \quad -5.0 \le x_2 \le 5.0$

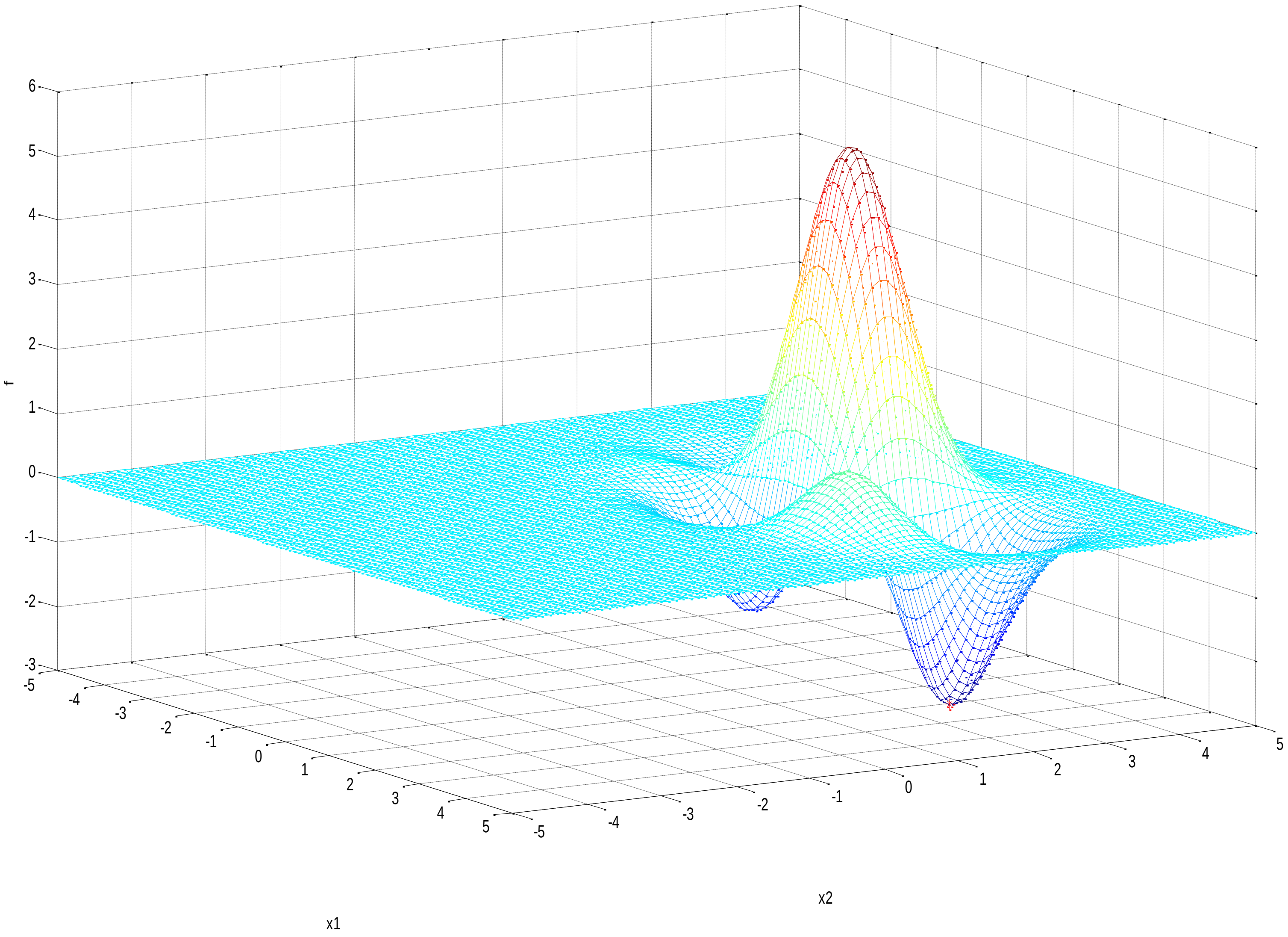
$\underline{x}(0) = \begin{bmatrix} 2 \\ 1.5 \end{bmatrix}$

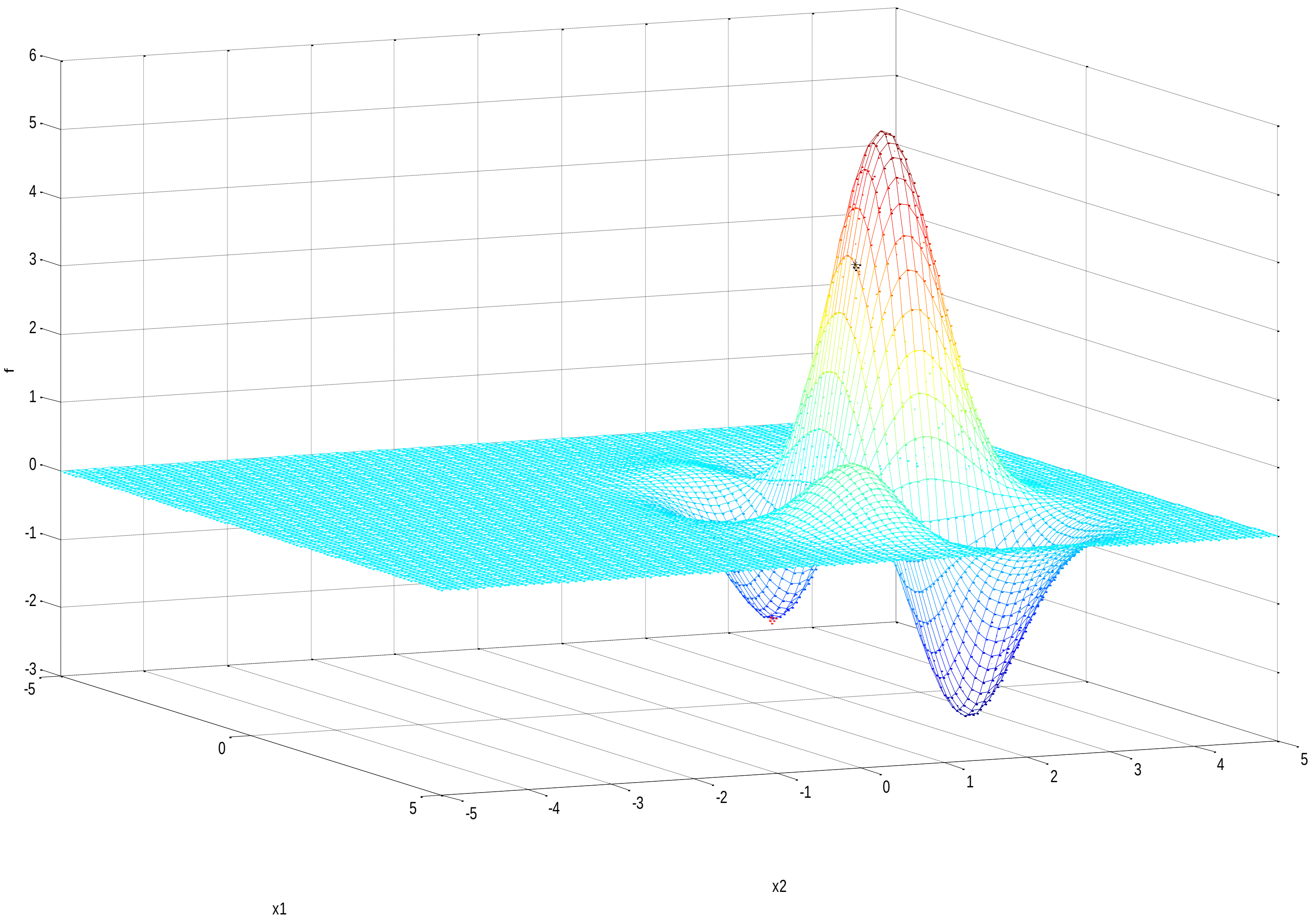
$\underline{x}^* = \begin{bmatrix} 2.49 \\ 2.43 \end{bmatrix}$

$f^* = -2.8733$

$\lambda = 0.05$

$i = 15$


f
6
5
4
3
2
1
0
-1
-2
-3
-5
-4
-3
-2
-1
0
1
2
3
4
5
x1
-5
-4
-3
-2
-1
0
1
2
3
4
5
x2



$$\underline{x}(0) = \begin{bmatrix} 0.5 \\ 2.0 \end{bmatrix} \; ; \quad \underline{x}^* = \begin{bmatrix} 0.34 \\ 1.08 \end{bmatrix} \; ; \quad f^* = -1.7093; \quad i = 19$$

- **Limitations**

    i) cannot be used for discontinuous objective functions

    ii) local minima

    iii) dependent on initial guess

    iv) slow terminal convergence

Numerical Example:

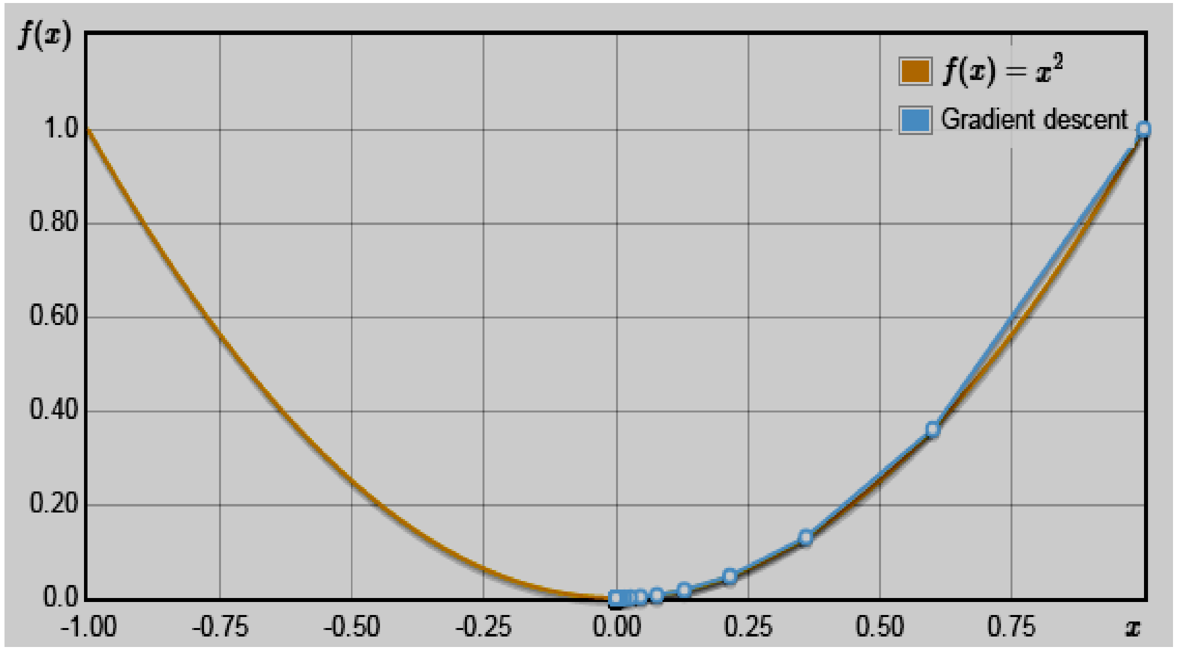
Minimize $f(x_1, x_2) = x_1^2 + 2x_2^2 + x_1x_2$ by steepest descent method. Show the calculations for one iteration considering the starting point to be (2, 2).

- **Effect of Step Size**

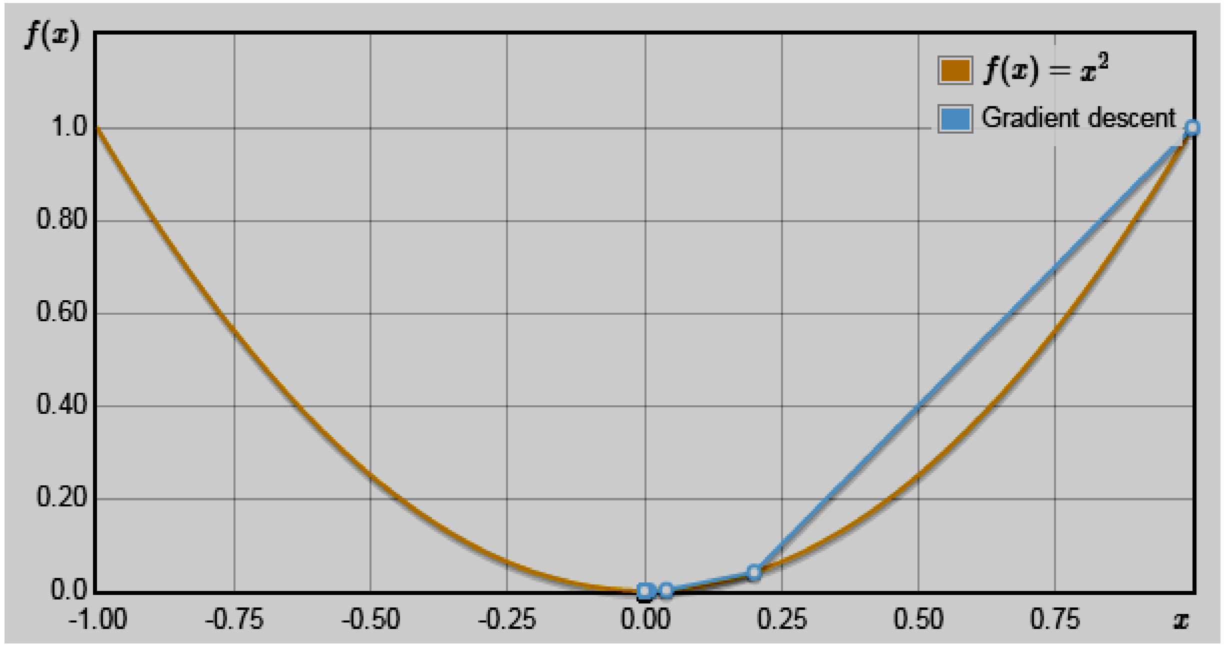
  Objective function: $f(x) = x^2$

  Initial point: $x = 1$

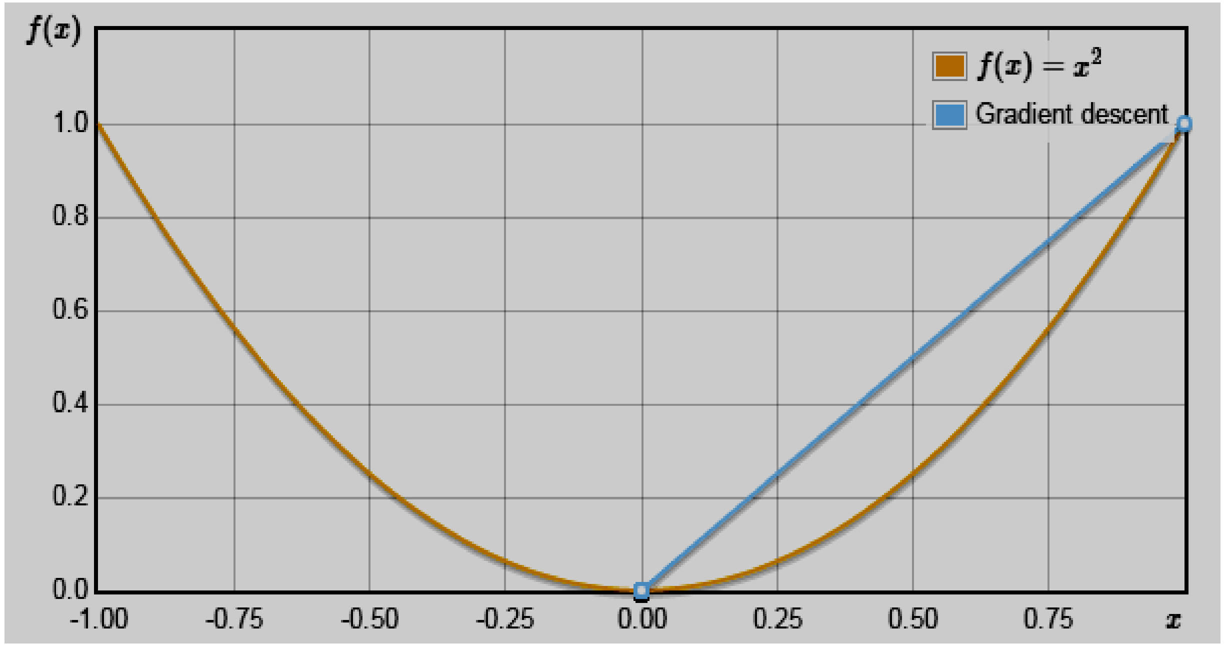
  Variable step size (λ = 0.2, 0.4, 0.5, 0.8, 1)



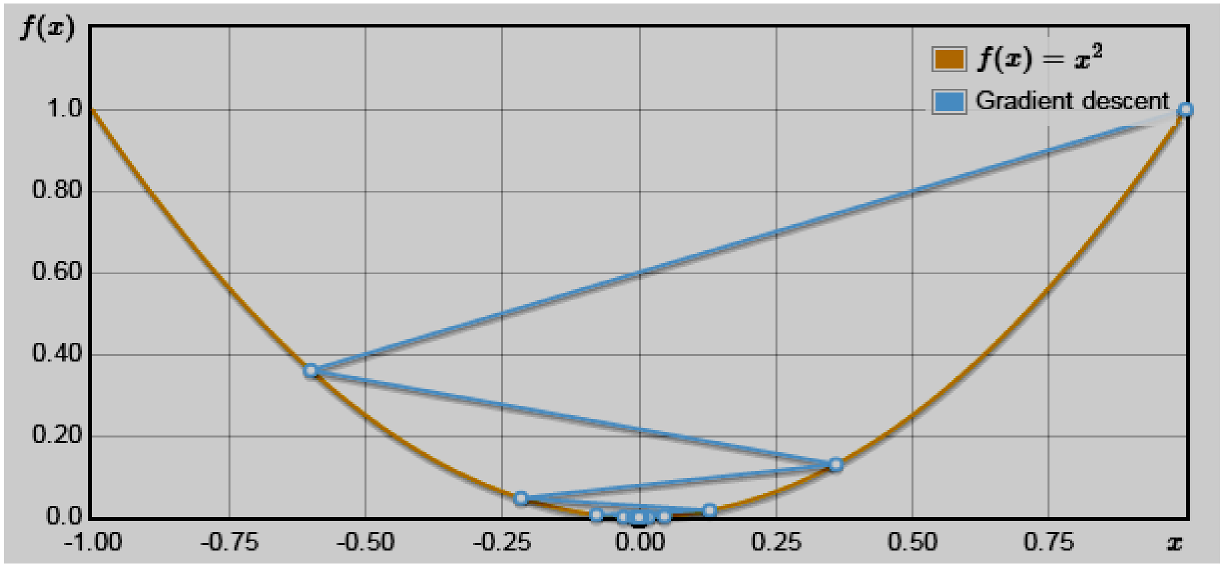
$\lambda = 0.2$



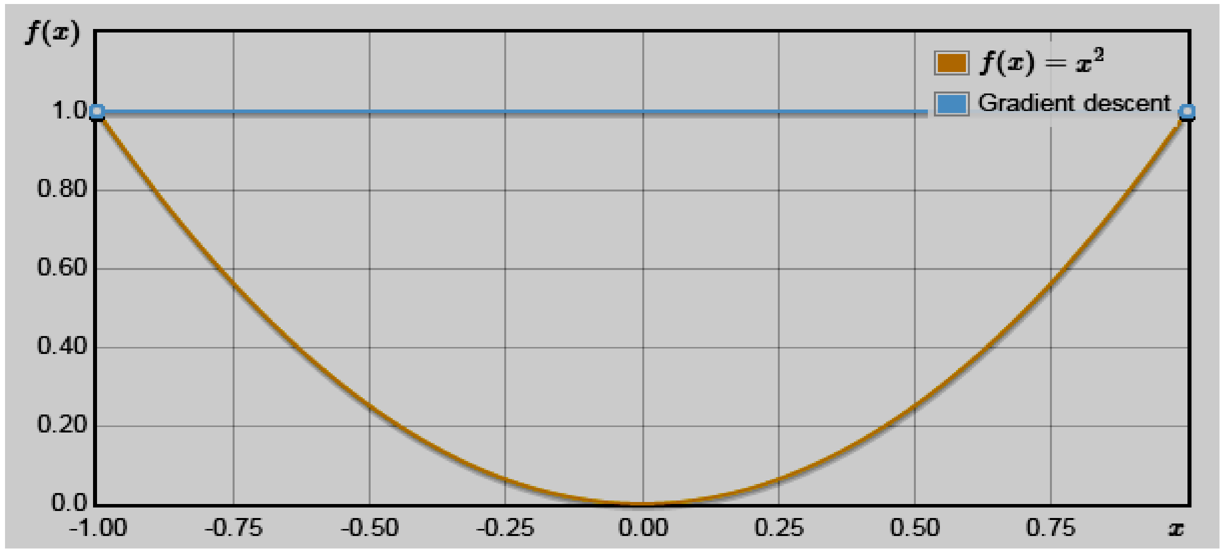
$\lambda = 0.4$



$\lambda = 0.5$
(optimal)



$\lambda = 0.8$



$\lambda = 1.0$

- Adaptive Step Size

- **Momentum Method**

  - ✓ $\Delta \underline{x}(i) = \lambda\, \underline{S}(i)$: without momentum

    $\Delta \underline{x}(i) = \lambda\, \underline{S}(i) + \alpha\, \Delta \underline{x}(i-1)\,; \quad 0 < \alpha < 1$

  - ✓ Faster convergence
  - ✓ If gradients have same signs in consecutive iterations then acceleration; else deceleration

Numerical Method: Unconstrained Problem

➢ Random Walk Method (Gradient free method)

- minimize $f(\underline{x})$ $where\ \underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$

- Randomly guess an initial solution and choose a large step size (>1)

- From this starting point, choose a random direction and check if a step is taken in that direction then the objective function decreases or not

- If yes, take that step. i.e. $\underline{x}(i+1) = \underline{x}(i) + \lambda\, \underline{u}(i)$
  If not, choose another direction and check. Keep checking for a maximum N number of times

- After N times if no better solution is found, halve the step size and start all over again

- Continue till the termination criterion is satisfied

How to generate the random direction?

✓ Generate ‘n’ uniformly distributed random numbers $(r_1, r_2, \ldots r_n)$ between -1 and 1.

✓ Discard if $(r_1^2 + r_2^2 + \ \ldots + r_n^2)^{1/2} > 1$ (to avoid bias)

✓ Then compute the unit vector as $\underline{u} = \frac{1}{(r_1^2 + r_2^2 + \ldots + r_n^2)^{1/2}} \begin{bmatrix} r_1 \\ \vdots \\ r_n \end{bmatrix}$

- **Comparison between Random Walk and Steepest Descent**

  i) steepest descent has much faster convergence

  ii) random walk is slightly more likely to find the global minimum

  iii) random walk can be applied if the objective function is discontinuous

Numerical Example:

Minimize $f(x_1, x_2) = x_1^2 + 2x_2^2 + x_1x_2$
Show the calculations for one iteration considering the starting point to be (2, 2); two random numbers 0.3 and -0.2; and a step size of 1.0.

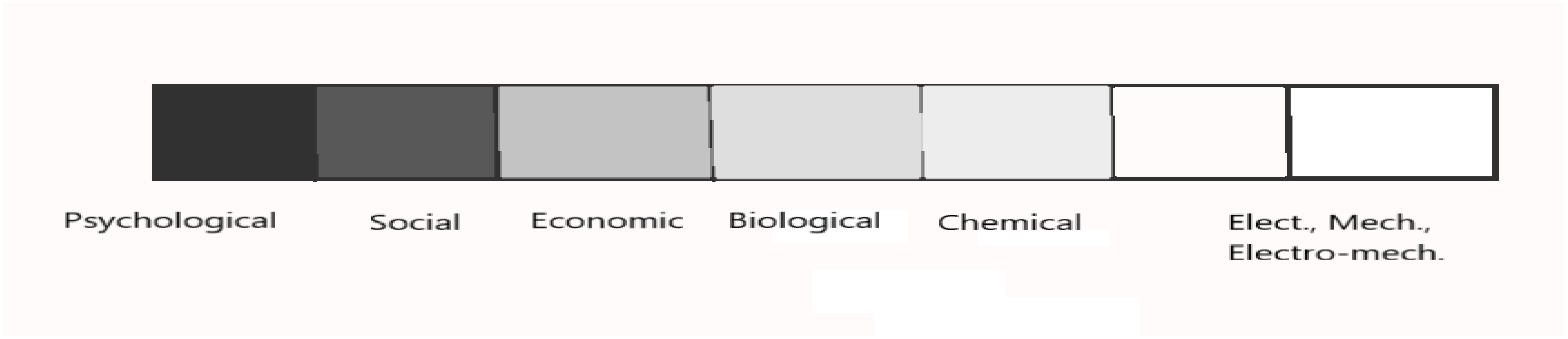
Nontraditional Optimization

Neural Networks & Fuzzy Logic

(BITS F312)

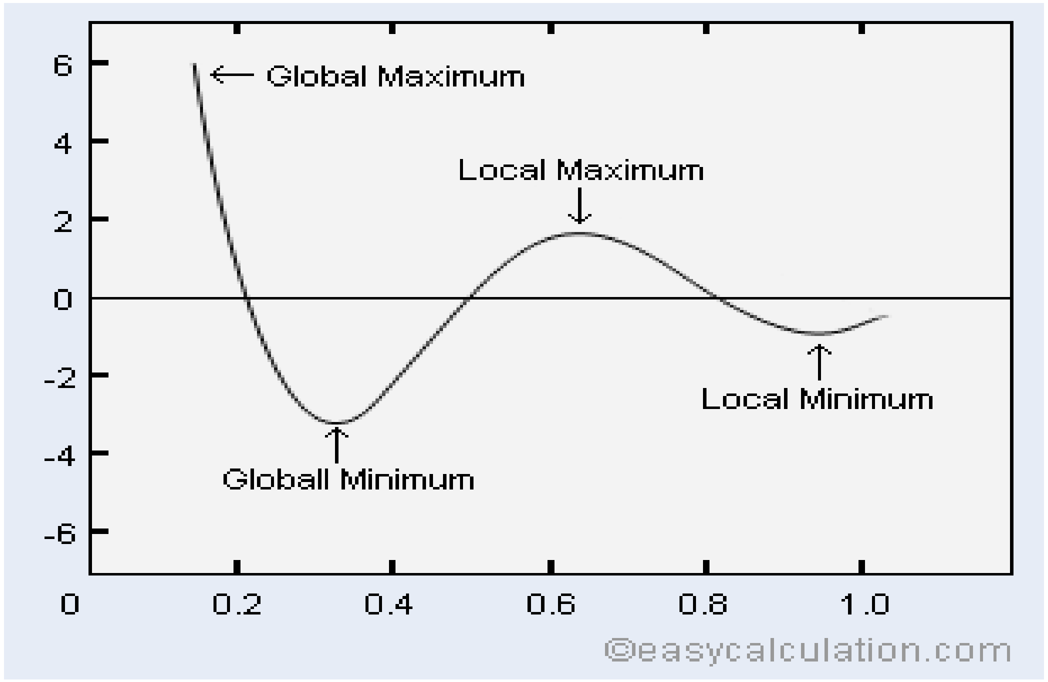
Traditional & Nontraditional/Metaheuristic Techniques

- Based on pure mathematics
- Yields precise solutions
- Suitable for problems which are easy to model mathematically



- May not be suitable to solve many complex real-world problems
- They complement each other

- To find a global minimum, users normally try a heuristic approach where several local minima are found by repeated trials with different starting values or by using different techniques
- The smallest of all known local minima is then assumed to be global minima
- Not very reliable



- **Features of Nontraditional Techniques**
  - Do not require derivative information (derivative free methods)
  - Search wide domains of the objective/cost surface
  - Can efficiently deal with a large number of variables
  - Optimizes variables with extremely complex cost surfaces
  - Suggests possible other sub-optimal solutions which may be more suitable for implementation
  - Highly flexible and versatile
  - Computationally expensive (more suitable for off-line applications)
  - Intuitive (no strong mathematical ground; black box nature) and inspired from some natural/biological process
  - Stochastic in nature (relying heavily on random number generation)
  - Large number of hyper parameters

- **Some Nontraditional Optimization Techniques**
  - GA, ES
  - Differential Evolution (DE)
  - Simulated Annealing (SA)
  - Particle Swarm Optimization (PSO)
  - Ant Colony Optimization (ACO)
  - Artificial Bee Colony (ABC)
  - Artificial Immune System (AIS)
  - Gray Wolf Algorithm (GWA)
  - Firefly Algorithm (FA)
  - Bacterial Foraging Optimization (BFO)
  - Harmony Search (HS)   and the list goes on and on …

Genetic Algorithm (GA)

- Introduced by John Holland in 1975 (University of Michigan) and popularized by his student David Goldberg
- Philosophically, GA is based on Darwin's theory of survival of the fittest
- Basic idea is to start from a randomly chosen set of probable solutions and then allow intermixing of these candidate solutions over many generations and gradually evolve towards the optimal solution

Steps for Binary Coded GA

- In principle, suitable for maximization problems
- Let us first consider the unconstrained optimization problem
- find $\underline{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ to

maximize $f(\underline{x})$

where $x_i^{min} \leq x_i \leq x_i^{max} \quad i = 1,2,\ldots n$

i.e. $\underline{x} \in \Omega \subset \mathcal{R}^n$

Initialization

Encoding

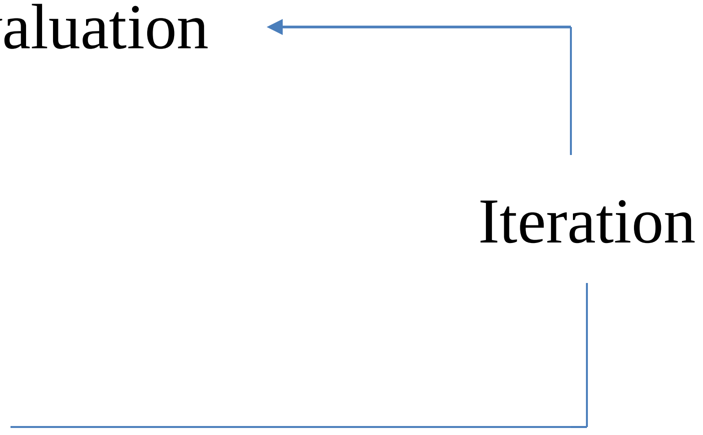
Fitness Evaluation

Selection

Crossover

Mutation

Decoding



i) Initialization

- A population size N is chosen and then N random points are picked from the phenotype space (search space) Ω
- N is even and usually a few 10’s to a few 100’s (not too high!)

ii) Encoding

- N points from the phenotype space are mapped to N corresponding points in the genotype space
- Each variable x_i is represented by a binary substring of length l_i
- Such binary substrings are concatenated for all $i=1,2,\ldots n$
- Performed for all N points

- l_i depends on the desired accuracy level of x_i
- $\epsilon_i = \frac{x_i^{max} - x_i^{min}}{2^{l_i} - 1}$
- $l_i = ceiling\,[\,log_2\left(1 + \frac{x_i^{max} - x_i^{min}}{\epsilon_i}\right)]$
- Mapping from the range $\left[x_i^{min}, x_i^{max}\right]$ to $[0,\ 2^{l_i} - 1]$ is done as follows:

$$x_i' = round\left\{\frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}}\left(2^{l_i} - 1\right)\right\} \text{ or } x_i = x_i^{min} + \epsilon_i' \,.\, x_i'$$

- x_i' is then converted to binary substring D_i

Numerical Example:

In an optimization problem we have two design variables varying in ranges $x_1 \in [-2,\ 2]$ and $x_2 \in [1, 12]$.

Represent the point $\underline{x} = \begin{bmatrix} -0.85 \\ 4.21 \end{bmatrix}$ as a chromosome. Each variable must have an accuracy level of at least 0.2.

Ans.: 01001 010010

iii) Decoding and Fitness Evaluation

- Next the fitness values at all the N points in the genotype space are computed
- Extract the decimal integer x_i' from the GA string and then map back to the phenotype space as

$$x_i = x_i^{min} + \epsilon_i' \,.\, x_i'$$

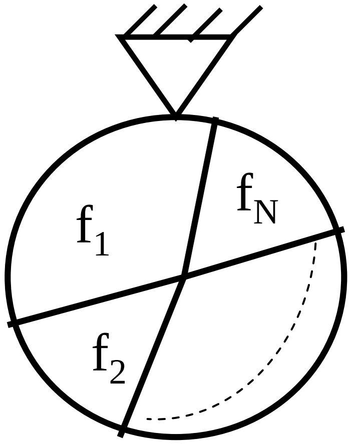
- Compute the objective function value once all x_i (i=1, 2, … n) are obtained for a particular chromosome
- This is usually the fitness value of that chromosome

iv) Selection

A mating pool (size N) is selected from the population

(a) Proportionate selection/Roulette-Wheel selection

- Probability of getting selected in a mating pool $\propto$ fitness
- Implemented with the help of a Roulette-Wheel

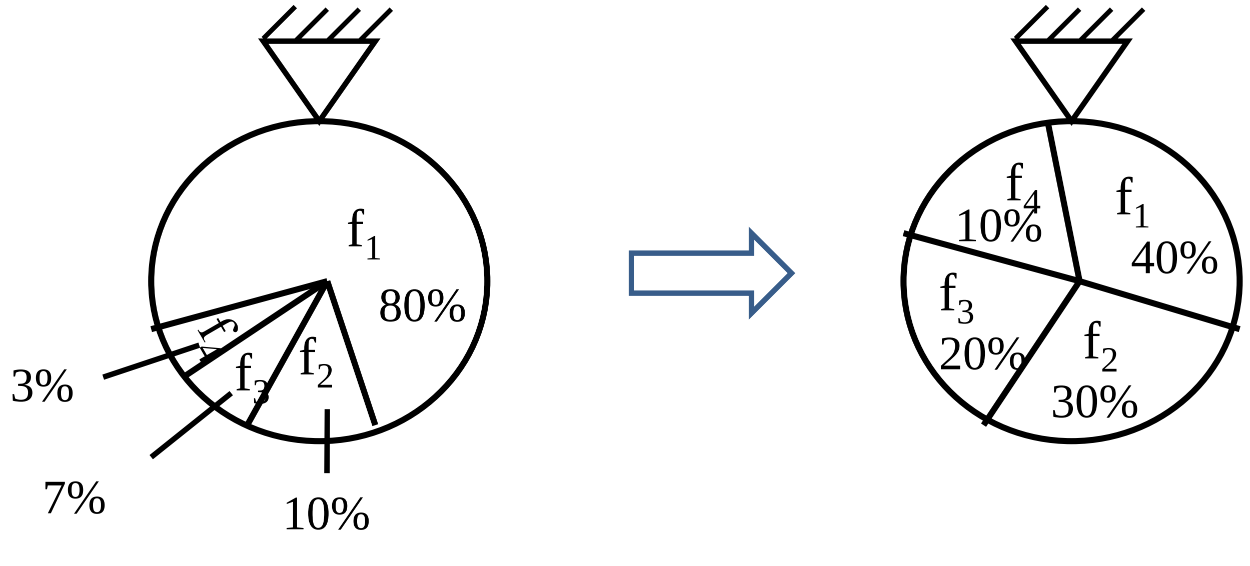


$$P_i = \frac{f_i}{\sum_{i=1}^{N} f_i}$$

- Balancing Selection Pressure and Population Diversity

(b) Ranking Selection

- Least fit string is given a rank of 1, next one rank 2 etc.
- Thus the fittest string gets the highest rank
- Then Roulette wheel selection is performed based on the ranks (rather than on the fitness)



Proportionate selection

Ranking selection

(c) Tournament Selection

- Small number (2-3) of chromosomes are randomly picked from the population and the fittest is put in the mating pool
- All the candidates are returned to the original population
- The process is repeated N times

v) Crossover

- Crossover operation forms children chromosomes for next generation by using parent chromosomes from the mating pool of the current generation
- N/2 mating pairs are selected at random
- With some crossover probability (p_c near to 1.0), the parent chromosomes swap some portions of themselves to produce two children chromosomes
- In single-point crossover a crossover site is chosen at random and the portions on the right side of the crossover site are interchanged

- **Single-point Crossover**

$$\left.\begin{array}{c|c}
0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 & 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \\
0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0 & 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1
\end{array}\right\}\ \text{Parents}$$

$$\Downarrow$$

$$\left.\begin{array}{c|c}
0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1 & 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1 \\
0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0 & 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1
\end{array}\right\}\ \text{Children}$$

- **Two-point Crossover**

$$\left.\begin{array}{c|c|c}
1\ 0\ 1\ 0\ 1\ 1 & 1\ 0\ 0\ 1\ 1 & 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0 \\
0\ 1\ 0\ 0\ 1\ 0 & 1\ 1\ 1\ 0\ 1 & 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1
\end{array}\right\}\ \text{Parents}$$

$$\Downarrow$$

$$\left.\begin{array}{c|c|c}
1\ 0\ 1\ 0\ 1\ 1 & 1\ 1\ 1\ 0\ 1 & 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0 \\
0\ 1\ 0\ 0\ 1\ 0 & 1\ 0\ 0\ 1\ 1 & 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1
\end{array}\right\}\ \text{Children}$$

- **Uniform Crossover**

  ✓ At each bit position a coin is tossed to decide whether there will be interchange of the bits.

  ✓ If ‘head’ appears there will be a swapping else not.

$$\left.\begin{array}{l} 1\ \overset{\checkmark}{0}\ 1\ \overset{\checkmark}{1}\ \overset{\checkmark}{0}\ 0\ 0\ \overset{\checkmark}{0}\ \overset{\checkmark}{1}\ 1\ 1\ \overset{\checkmark}{0}\ 0\ 1\ \overset{\checkmark}{1}\ 0\ 1\ \overset{\checkmark}{1}\ 1\ \overset{\checkmark}{0} \\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0 \end{array}\right\} \text{Parents}$$

Let us assume that 2-nd, 4-th, 5-th, 8-th, 9-th,12-th, 18-th and 20-th bit positions are selected for swapping

$$\left.\begin{array}{l} 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 0\ 0 \end{array}\right\} \text{Children}$$

Found to perform better for large search spaces

- Crossover operation introduces randomness into the current population to help avoid getting trapped in local minimum

- Crossover operation may result in better or worse strings. If a few worse offsprings are formed, they will not survive for long since upcoming generations are likely to eliminate them

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

vi) Mutation

- After crossover, a small fraction (decided by a small mutation probability p_m) of the N members are made to undergo Mutation
- Each bit of the selected chromosomes are flipped with the same mutation probability p_m (bit-wise mutation)
- A mutation site is chosen at random and the corresponding bit is flipped (single point mutation)

- Helps a great deal to avoid getting stuck in local minima
- Too low p_m $\longrightarrow$ local minima problem

  Too high p_m $\longrightarrow$ too much randomization

- GA relies more on crossover, whereas ES puts more weightage on mutation

Termination criteria

- Maximum number of Generations, say G = 100
- At least n_2 generations are complete and at least over the last n_1 generations, the best solution is not changing by more than a small ε

➢ Best solution of each generation is saved

Elitism

– Objective is to speed up convergence

– A few best chromosomes of the present generation (Elite Count (EC) = 2, say) are directly copied to the next generation

Numerical Example

maximize $f(x) = -x^2 + 5x + 200$; $0.0 \le x \le 15.0$

Initial population is chosen as

1001

1110

0011

0100

1000

0010

Random numbers generated for Roulette wheel selection are 0.96, 0.83, 0.47, 0.61, 0.12, 0.35. Compute the next generation and their fitness values. Assume fixed crossover site of two; mating pairs as (1-4), (2-5) and (3-6); $P_c = 1.0$; $P_m = 0.0$ and EC=0.

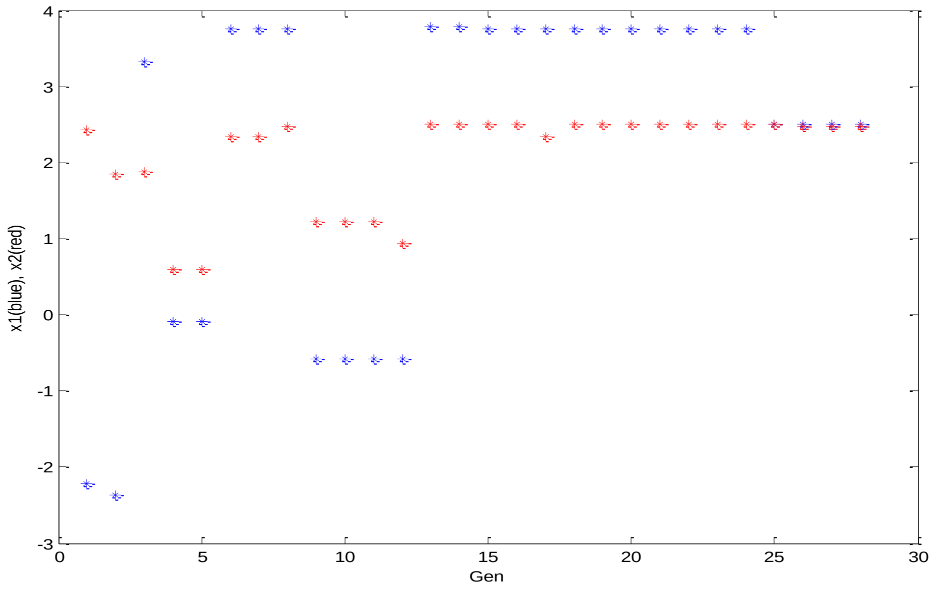
Example:

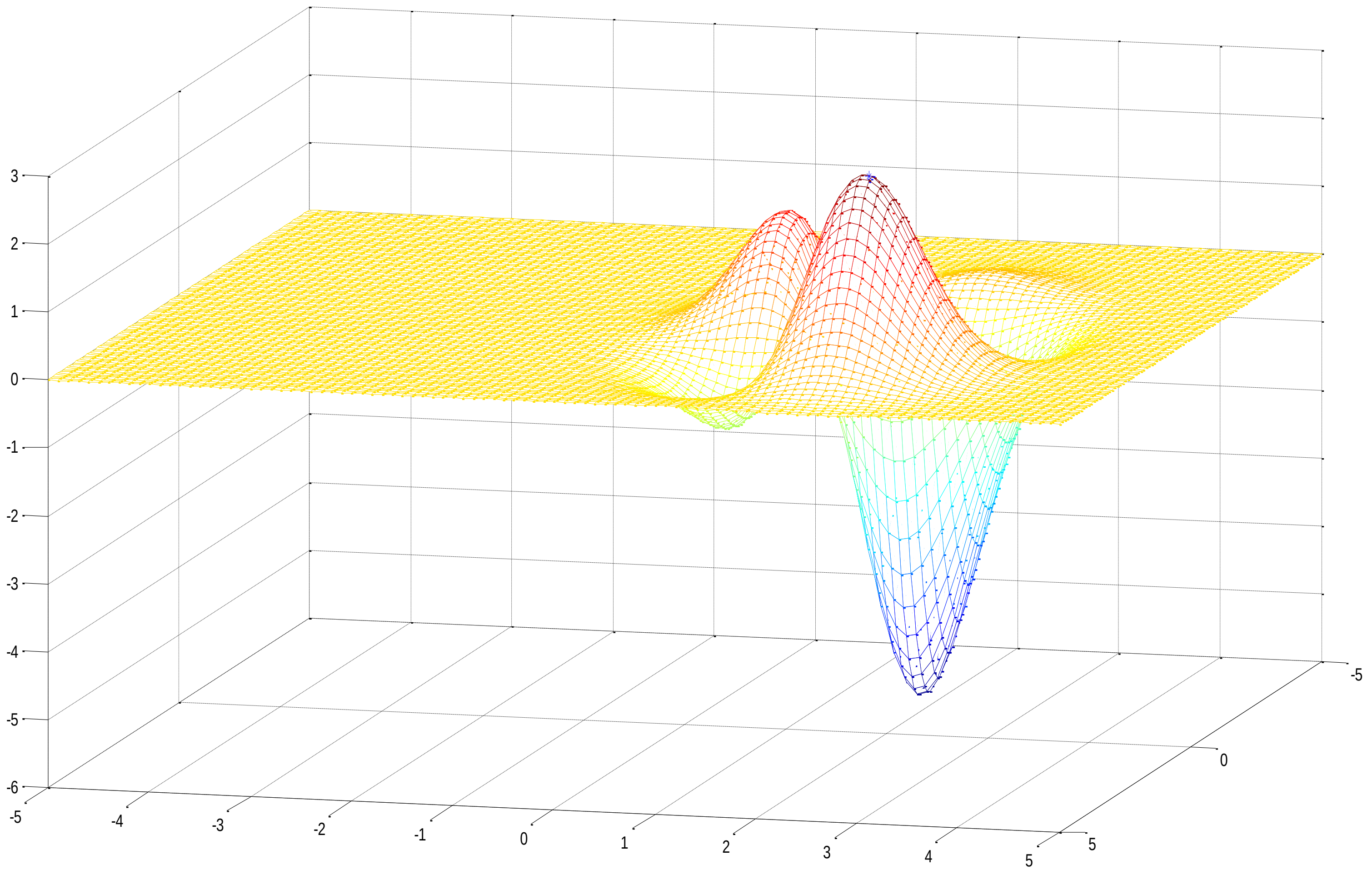
$$f = -10 \cos x_1 \cos x_2 \; e^{-\left\{\frac{(x_1-1)^2}{4} + (x_2-2)^2\right\}}$$

$$-5.0 \le x_1 \le 5.0 \, ; \quad 0.0 \le x_2 \le 5.0$$

$$\underline{x}^* = \begin{bmatrix} 2.51 \; (2.49) \\ 2.48 \; (2.43) \end{bmatrix} \qquad f^* = -2.8613 \; (-2.8733)$$

- N=20 (40)
- Pc=0.9
- Pm=0.09
- EC=0
- Chromosome length= 10+10
- Uniform crossover




3
2
1
0
-1
-2
-3
-4
-5
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
-5
0
5

Schema Theorem of Binary Coded GA

- Proposed by Prof. John Holland

- An attempt to give GA a mathematical foundation

Let us consider a population of binary-strings created at random

```
1 1 0 0 1 1
0 1 0 1 0 0
¦ ¦ ¦ ¦ ¦ ¦
1 1 0 1 1 1
1 1 0 0 0 0
```

Let us assume the following two schemata (templates):

H_1: * 1 0 * * *

H_2: * 1 0 * 0 0

(* could be either 1 or 0)

- Order of schema O(H):

  No. of fixed positions (bits) present in a schema.

  For example: $O(H_1) = 2$; $O(H_2) = 4$

- Defining length of schema $\delta(H)$:

  Distance between the first and last fixed positions in a string.

  For example: $\delta(H_1) = 3-2 = 1$; $\delta(H_2) = 6-2 = 4$

- **Effect of Selection:**

Let $m(H, t)$ = No. of strings belonging to schema H at tth Gen.

$m(H, t+1)$ = No. of strings belonging to schema H at (t+1)th Gen.

$f(H)$ = Schema fitness or Avg. fitness of the strings represented by schema H at t-th Gen.

$\bar{f}$ = Avg. fitness of the entire population at t-th Gen.

$$E\,[m(H, t+1)] = m(H, t)\,\frac{f(H)}{\bar{f}}$$

- **Effect of Crossover** (Single-point):

Let P_c= Probability of crossover

L = String length

A schema is destroyed if crossover site falls within the defining length

Probability of destruction = $p_c \frac{\delta(H)}{L-1}$

Probability of survival = $1 - p_c \frac{\delta(H)}{L-1}$

- **Effect of Mutation** (Bit–wise Mutation):

To protect a schema, mutation should not occur at the fixed bits

Let p_m : probability of mutation

probability of destruction = p_m

probability of survival = $1- p_m$

Probability of survival considering all the fixed bits in a schema

$$p_s = (1- p_m)\,(1- p_m) \ldots\ldots\ldots\ldots O(H)$$

$$= (1- p_m)^{O(H)}$$

$$= 1- O(H)\, p_m \qquad \text{as } p_m \ll 1$$

Considering the contributions of all three operators,

$$\mathrm{E}\left[m(H,t+1)\right] = m(H,t)\,\frac{f(H)}{\overline{f}}\left[1 - p_c\,\frac{\delta(H)}{L-1} - O(H)\,p_m\right]$$

<u>Building-Block Hypothesis:</u>

The schemata having low order, short defining length and fitness considerably more than average fitness of the population will receive more and more representations in future generations

Limitations of Binary Coded GA

- Unable to yield any arbitrary precision in the solution → Real Coded GA

- Hamming Cliff problem → creates an artificial hindrance to the gradual search of a GA → Gray Coded GA

$$\begin{tikzcd}
14: & 0\,1\,1\,1\,0 \arrow[d, bend left, "\text{1 change}"] \\
15: & 0\,1\,1\,1\,1 \arrow[d, bend left, "\text{5 changes}"] \\
16: & 1\,0\,0\,0\,0
\end{tikzcd}$$

Constraints Handling in GA (also applicable to PSO)

Optimize $f(\underline{x})$

Subject to

$$g_j(\underline{x}) \le 0, \; j = 1,2,\ldots,m$$

$$h_k(\underline{x}) = 0, \; k = 1,2,\ldots,p$$

$$\underline{x} = [x_1 \; x_2 \; \ldots \; x_n]^T$$

$$\underline{x}_{\min} \le \underline{x} \le \underline{x}_{\max}$$

Let $m+p = q$

Functional constraints

$$\Phi_k(\underline{x}), \; k = 1,2,\ldots,q$$

Penalty Function Approach

Fitness function of ith solution

$$F_i(\underline{x}) = f_i(\underline{x}) \pm P_i \quad \text{(+ for minimization problems)}$$

where P_i indicates penalty

$$P_i = C \sum_{k=1}^{q} \left\{ \varphi_{ik}(X) \right\}^2$$

C indicates penalty coefficient

Static Penalty

Fitness of i-th solution

$$F_i(X) = f_i(X) + \sum_{k=1}^{q} C_{k,r} \left\{ \varphi_{ik}(X) \right\}^2$$

where $C_{k,r}$: rth level violation of kth constraint

(amount of violation is divided into various pre-defined levels)

Dynamic Penalty

Fitness $$F_i(X) = f_i(X) + (C.t)^{\alpha} \sum_{k=1}^{q} \left|\varphi_{ik}(X)\right|^{\beta}$$

where C, α, β are the user-defined constants

t = number of generations

✓ Penalty increasing with generation number (pressurizing GA)

Adaptive Penalty

Fitness $$F_i(X) = f_i(X) + \lambda(t) \sum_{k=1}^{q} \left\{\phi_{ik}(X)\right\}^2$$

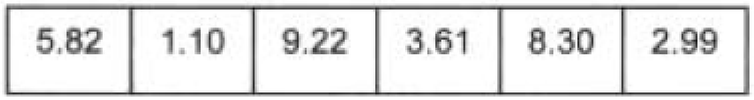
where t : number of generations

$$\lambda(t+1) = \begin{cases} \frac{1}{\beta_1}.\lambda(t), & \text{if best solns. of last } N_f \text{ GEN were feasible} \\ \beta_2.\lambda(t), & \text{if infeasible} \end{cases}$$

if neither, λ(t+1)= λ(t) (where $\beta_1 \neq \beta_2$ and β_1, β_2 >1)

Real Coded GA:

Chromosome:



5.82	1.10	9.22	3.61	8.30	2.99

- ✓ Selection: Same as Binary Coded GA

- ✓ Crossover: Single point

  Linear Crossover

$$\left.\begin{array}{l} Ch1 = 0.5*Pr1+0.5*Pr2 \\ Ch2 = 1.5*Pr1-0.5*Pr2 \\ Ch3 = -0.5*Pr1+1.5*Pr2 \end{array}\right\} \text{Best Two}$$

- ✓ Mutation: Replace a value with a random value in the range of the corresponding variable

- Mutation probability in RCGA is more than that in BCGA

Let m be the string length for a variable $x1$ in BCGA

Probability that this variable survives mutation is $(1 - P_m)^m$

$$\approx (1 - mP_m)$$

Hence, $(1 - P_m^R) = (1 - mP_m^B)$

$$P_m^R = m\, P_m^B$$

Numerical Example: maximize $f(x_1, x_2) = - x_1^2 - 2x_2^2 - x_1x_2$

Assume the mating pool to be (4,0); (-2,2); (3,1); (3,1)

Considering mating pairs as 1-3 and 2-4, obtain the next generation by single point crossover and linear crossover. Assume Pc=1.0.

Particle Swarm Optimization

- James Kennedy & Russell Eberhart in 1995
- Inspired from social behavior of birds and fish (also humans)
- Population-based optimization
- Complex tasks are better performed in a group
- Combines self-experience with social experience

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 ‘flight’ 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 has access to

  - Individual previous best position (P_{best})

  - Swarm's best or global best position (G_{best})

PSO Algorithm

1. Select swarm size (a few 10's) and initialize the positions of the particles from the solution space (velocities may be zero or random)
2. Evaluate the fitness of each particle
3. Update personal and global bests
4. Update velocity, position, and inertia of each particle
5. Go to Step 2, and repeat until termination condition

   (termination condition: all points have nearly converged)

- Velocity update equation:

$$\mathbf{v}_i(t+1) = w.\mathbf{v}_i(t) + c_1.r_1.(\mathbf{P}_{best_i}(t) - \mathbf{x}_i(t))$$
$$+c_2.r_2.(\mathbf{G}_{best}(t) - \mathbf{x}_i(t))$$

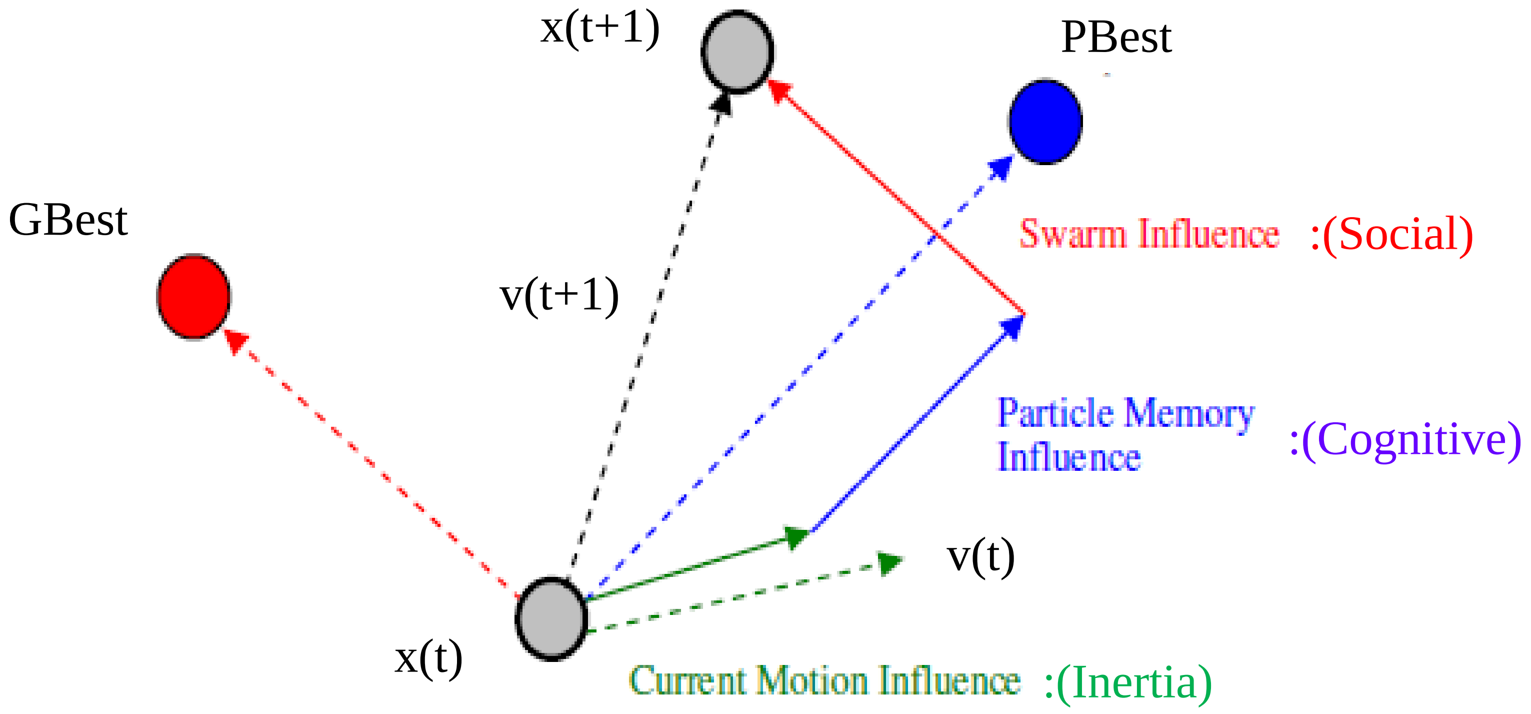
  - w is inertia weight
  - c_1 is cognitive attraction constant (may be chosen as 2.0)
  - c_2 is social attraction constant (may be chosen as 2.0)
  - r_1, r_2 are random numbers in [0, 1]

- Position update equation:

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

- Particle's velocity

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



- *w* (<1.0) is introduced to reduce momentum
- *w* is gradually reduced while nearing the solution

  (say, from 0.9 to 0.4)
- *r1* and *r2* may be vectors also

- Sometimes $\mathbf{G}_{\text{best}}$ is replaced with $\mathbf{L}_{\text{best}}$

  - Each particle tries to emulate the best in the neighbourhood

  - Various topologies proposed to define the neighbourhood

  - More exploration and less exploitation

    (hence less chance of premature convergence, but slower)

  - Neighbourhood may be static or dynamic

- Sometimes velocity is hard limited to $\pm V_{max}$ to prevent the particle to diverge

- Sometimes position may also be hard limited to the predefined search space

- Some analysis regarding stability is available in Dan Simon

$$\mathbf{v}_i(t+1) = K[\mathbf{v}_i(t) + \varphi_1(\mathbf{P}_{best_i}(t) - \mathbf{x}_i(t)) + \varphi_2(\mathbf{G}_{best}(t) - \mathbf{x}_i(t))]$$

Some Improvements:

- Making Personal and Social Experiences complementary to each other

$$\mathbf{v}_i(t+1) = r_2.\mathbf{v}_i(t) + (1-r_2).c_1.r_1.(\mathbf{P}_{best_i}(t) - \mathbf{x}_i(t)) \\ + (1-r_2)c_2.(1-r_1).(\mathbf{G}_{best}(t) - \mathbf{x}_i(t))$$

- Learning from mistakes (moving away from the worst solutions)

$$\mathbf{v}_i(t+1) = w.\mathbf{v}_i(t) + c_1.r_1.(\mathbf{P}_{best_i}(t) - \mathbf{x}_i(t)) + c_2.r_2.(\mathbf{G}_{best}(t) - \mathbf{x}_i(t)) \\ - c_3.r_3.(\mathbf{P}_{worst_i}(t) - \mathbf{x}_i(t)) - c_4.r_4.(\mathbf{G}_{worst}(t) - \mathbf{x}_i(t))$$

- **Introducing a Craziness Term:**
  - To increase diversity
  - Incorporates fish/birds taking sudden turns

$$\mathbf{v}_i(t) = \mathbf{v}_i(t) + P(r_4).\mathrm{sgn}(r_4).\mathbf{v}_i^{\,craziness};$$

$$\mathbf{v}_i^{\,craziness} \in [\mathbf{v}_i^{\min}, \mathbf{v}_i^{\max}]$$

$$sgn(r_4) = \begin{cases} +1, & r_4 \geq 0.5 \\ -1, & r_4 < 0.5 \end{cases}$$

$$\mathrm{P}(r_4) = \begin{cases} 1, & r_4 \leq P_{cr} \\ 0, & r_4 > P_{cr} \end{cases}$$

GA vs. PSO

- GA is a good global optimizer but a poor local optimizer whereas PSO is good at both
- PSO is much faster than GA

Numerical Example:

maximize $f(x) = -x^2 + 5x + 200; \quad -5 \le x \le 5$

Assume N=4, initial positions $[4.7 \quad 2.1 \quad -4.3 \quad 3.4\,]^T$, initial velocity to be zero, w=c1=c2=1.0, r1=0.33, r2=0.18

Compute the positions of the particles after one iteration.

Further Resources:

- Engineering Optimization by S.S. Rao
- Soft Computing by D.K. Pratihar
- Evolutionary Optimization Algorithms by Dan Simon