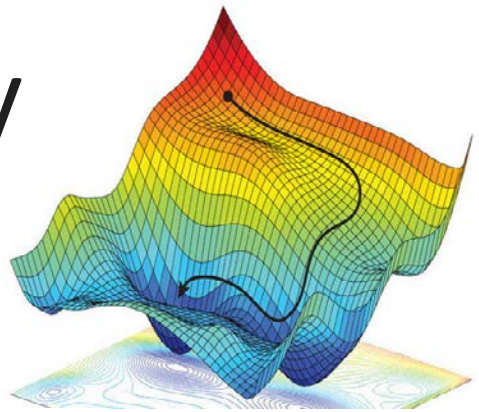
# Optimization Concepts, Difficult cases and Evolutionary Algorithms



**Asst. Prof. L. Özlem AKKAN**

# Optimization Concepts, Difficult cases and Evolutionary Algorithms

**Asst. Prof. L. Özlem AKKAN**

# Engineering Design: Analysis, Modeling, and Optimization

# Motivation for Optimization

# What is optimization?

In practice, doing something as well as possible within practical constraints is very satisfactory.

Optimization provides us with the means to make things happen/work in the best possible practical way.

Brute-force or manual optimization is done by trial and error, and using past experience, which for most practical problems:

- Will lead to highly sub-optimal solutions, since only few trials can be performed in a limited time, and/or
- Is too time consuming to be practically feasible.

This is where **quantitative optimization** comes in:

- Uses mathematical strategies to provide an efficient and systematic way to optimize.
- Using the capabilities of modern day computing, these mathematical strategies become all the more powerful in implementation.

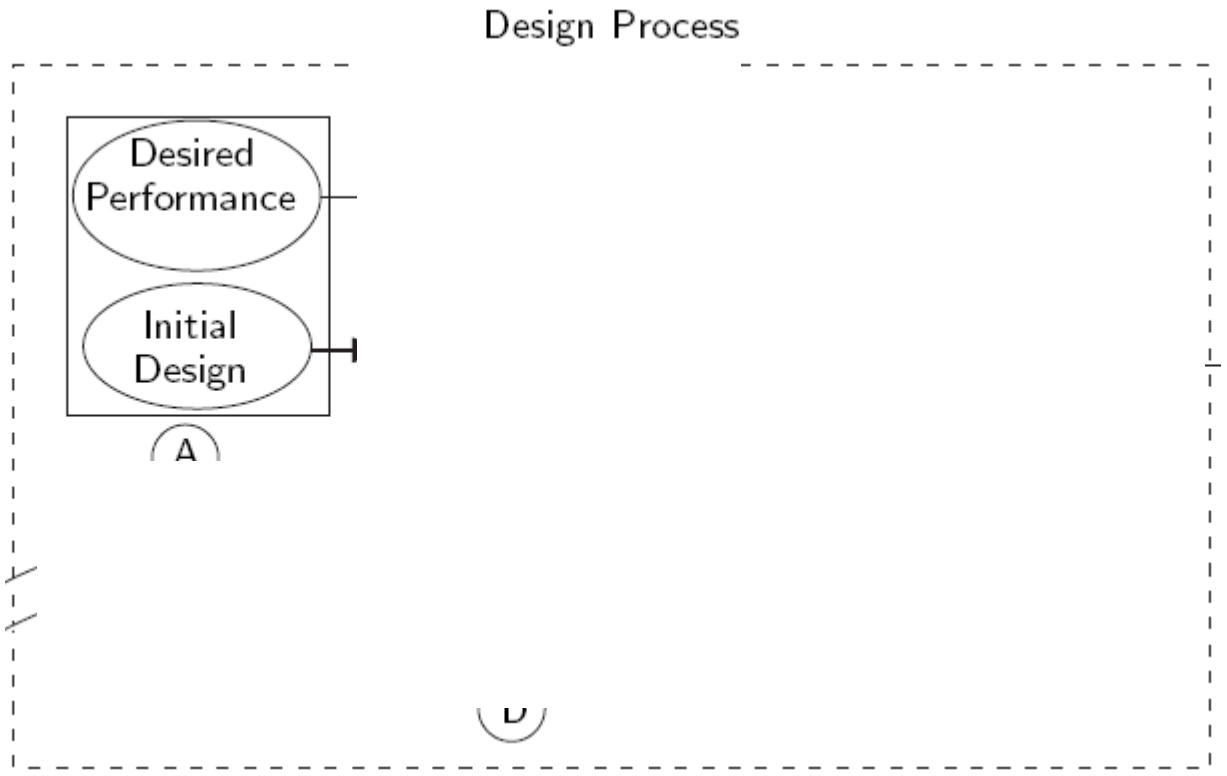
# Traditional vs. Mathematical Optimization process

Box A shows two inputs – 1. the dream design and 2. the initial design.

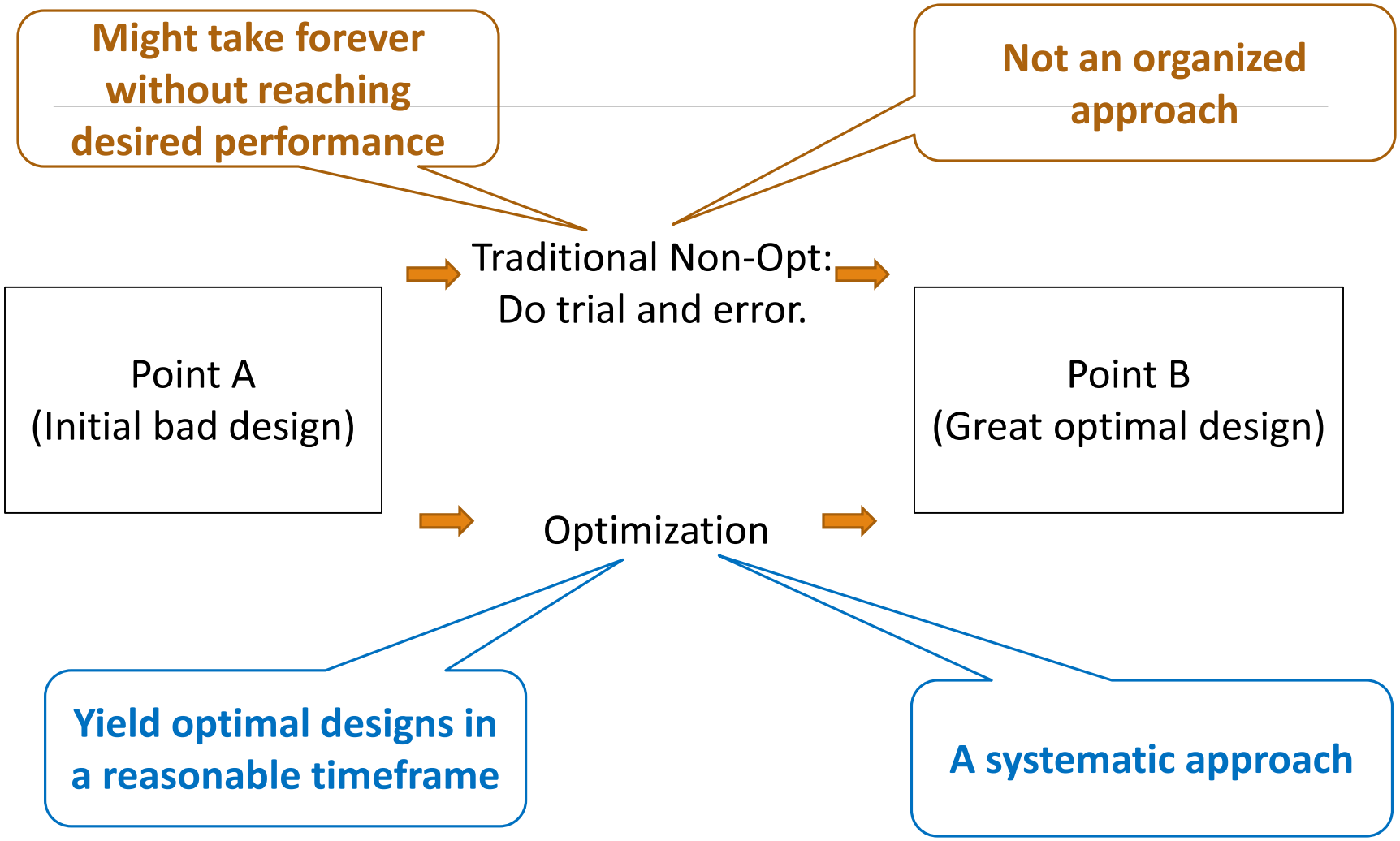
Box B shows the analysis phase.

Box D is where the design is improved in a very systematic way.

Box E shows manual optimization by a human being (trial & error/intuitive)



# Traditional vs. Mathematical Optimization



# Why Should We Study Optimization?

**Undergraduates**: To acquire the ability to optimize designs yourself, and feel comfortable and confident with the results

- As a student, in the classroom setting (e.g., Capstone Design project);
- As an engineer (post graduation), in the industry setting.

**Graduate Students**: To be able to use optimization to find better ways to proceed with your experiments, system modeling, or designs, in the course of your ongoing and future research.

**Industry Personnel**: To acquire the ability to leverage the immense potential of optimization in different real-life projects.

- Learn software tools to readily apply optimization in your projects;
- Acquire knowledge to be able to critically verify the optimal designs
- Acquire knowledge to be able to identify the challenges in optimizing a system, and to know where to look for the solutions.

# Analysis, Design, Optimization and Modeling

Analysis

Optimization

Design

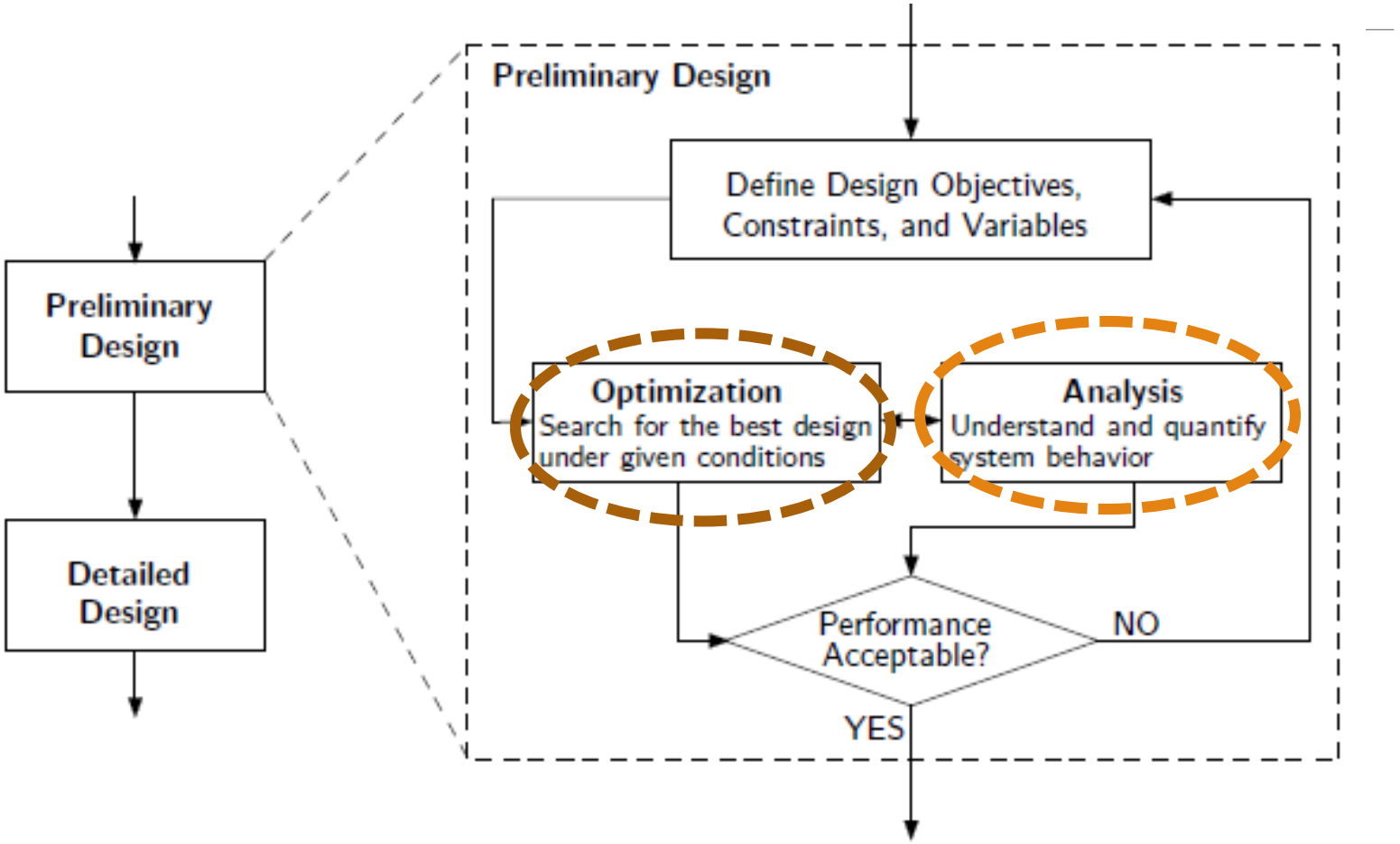
- *Analysis* and *optimization* are two activities integral to the process of design
  - These can be considered to be steps within the process of design
  - We will primarily focus on these two activities in the context of engineering or systems design

Modeling

- modeling system behavior
- modeling an optimization problem
- These involve two distinct lines of expertise

# Important Components of Systems Design

***Analysis*** and ***Optimization*** are two core components of a systems design process.



# What Is Analysis?

Engineering Analysis can be defined as:

*The application of scientific principles and processes to reveal the properties and the state of a system, and also understand the underlying physics driving the system behavior.*

Analysis generally demands disciplinary knowledge pertinent to the system or mechanism being analyzed.

Practical systems involve multiple disciplines, e.g., designing an aircraft requires structural, aerodynamic, and control analyses.

If disciplinary understanding has reached certain level of maturity, mathematical analysis tools might be readily available.

On the other hand, in the case of mechanisms or phenomena that are not yet well understood, in-depth and fundamental analysis might be required thereby demanding the involvement of a disciplinary expert.

# Optimization

***Mathematical optimization is the process of maximizing and/or minimizing one or more objectives without violating specified design constraints, by regulating a set of variable parameters that influence the objectives and the design constraints.***

The three types of quantities in optimization:

1. **Objectives**: The quantities that you would like to improve – *e.g., fuel efficiency of an aircraft (to be maximized), or manufacturing cost of the aircraft (to be minimized)*.
2. **Constraints**: The quantities or criteria that your design needs to satisfy – *e.g., the cargo/payload capacity of an aircraft*.
3. **Variables**: The quantities that you can directly change to improve the design, where the values of the objectives and constraints are regulated by these quantities – *e.g., the dimensions and the material of the aircraft wing*.

***Objective functions** and **constraint functions** are often together called **criteria functions**, since one can be converted into another.*

# Modeling for analysis

In the context of design, analysis and optimization are related to each other through **modeling**.

**Modeling**: *A process by which an engineer or a scientist translates the actual physical system/phenomena under study into a set of mathematical equations or operations*.

Mathematically, modeling can be represented as: $\boldsymbol{P = f(X)}$, where, ***P*** the quantity of interest, which is expressed as a function of a vector of design variables, ***X***.

Mathematical models may not be a single analytical function.

# Modeling the Optimization Problem

The success of optimization depends both on:

- the capabilities of the optimization method/algorithm;
- the effectiveness of the **optimization formulation**.

Modeling the optimization or **problem formulation** essentially involves developing a clear definition of the design variables, design objectives, and design constraints.

Problem formulation is also strongly correlated with the choice of optimization algorithms – e.g.

> During problem formulation, One can convert equality constraints into inequality constraints using a tolerance value, in order to leverage powerful algorithms that perform well in the absence of equality constraints.

# Modeling for Analysis and Optimization

It is important to ensure that **optimization problem formulation** is coherent with the **system behavior model**.

For example, the input output exchange (between analysis and optimization) demanded by the optimization formulation should be satisfied by the capabilities of the models used thereof.

The choice of analysis models also affects the choice of optimization algorithm, and vice versa. For example:

- If you choose an algorithm that generally requires a large number of system evaluations, then a fast model of the system behavior is needed.
- If the system behavior model is inherently highly nonlinear, you will need to formulate the optimization problem such that a nonlinear optimization algorithm can be used to solve the problem.

# Multi-objective Optimization

# Multi-objective Problems

Many real-life design problems contains more than one design objective (to be maximized or minimized).

- Example: Car design: Maximize Fuel Efficiency and Minimize Cost.

Generally, in practical problems where multiple objectives need to be considered, the objectives tend to be conflicting in nature

- Example: More fuel efficient cars (e.g., hybrid cars or cars with regenerative braking) tend to be more expensive than less efficient counterparts).

*Multi-objective optimization is a methodical way to solve problems involving* ***multiple design objectives simultaneously***.

# Multi-objective Problem Statement

$$\min_{x} \quad [\mu_1(x) \quad \mu_2(x) \quad \dots \quad \mu_n(x)]^T$$

Multiple **objective functions**

subject to

$$g(x) \leq 0$$

Vector of **inequality constraints**

$$h(x) = 0$$

Vector of **equality constraints**

$$x_l \leq x \leq x_u$$

**Side constraints**

When *n* = 2, we call it a bi-objective problem.

# Concept of Pareto Solutions

One of the interesting features of multi-objective optimization is that the solution to the problem is generally **not unique**.

A set of solutions called **Pareto Optimal Solutions** or the **best tradeoff solutions** form the complete solution set of the optimization problem.

**Example**

Lets consider an unconstrained problem involving two design objectives, $\mu_1$ and $\mu_2$, which are functions of a single design variable $x$.

We are interested in minimizing both design objectives simultaneously.

# Obtaining Pareto Solutions

Many optimization algorithms (including most conventional or classical algorithms) are suited to solve single objective problems.

One way to solve a multi-objective problem, by leveraging typical single-objective algorithms is to combine all the objectives into a single **Aggregate Objective Function**.

If the single objective is optimized, a Pareto solution is obtained.

# Aggregate Objective Function

Definition: An **Aggregate Objective Function** (**AOF**) is a combined function of all the design objectives.

The AOF typically contains weight parameters to be selected by the designer. These parameters generally reflect the relative importance of each design objective.

The type of final solution that you obtain will be directly dependent on the type of AOF that you use.

A general guideline is to use an AOF that allows a designer to impose his/her design objective preferences in an unambiguous manner.

# Aggregate Objective Function Formulations

**Some popular AOF formulations:**

1. The weighted sum method
2. Compromise programming
3. Goal programming
4. Physical programming

# Goal Programming

Instead of minimizing design objectives, we might wish to reach a given target value or goal for each objective.

For example, the stress design objective ($\mu_1$) is required to be as close to 1,500 MPa as possible, while the value of deflection ($\mu_2$) is required to be as close to 2 inches as possible.

Goal programming AOF can be formulated using the compromise programming concept, where *n* is an even integer (often 2), e.g.:

$$J(x) = w_1 (\mu_1 - 1,500)^n + w_2 (\mu_2 - 2)^n$$

The smallest theoretical value of *J* is zero, when $\mu_1$ = 1500 and $\mu_2$ = 2. In most real-life problems, this point will not be achievable.

By choosing an appropriate set of weights ($w_1$ and $w_2$) for the design objectives, we can obtain a Pareto solution that reflects the best trade-offs.

# Goal Programming

A normalized version of the goal programming formulation

$$J(x) = w_1 \left( \frac{\mu_1 - \overline{\mu}_{1g}}{\overline{\mu}_{1b} - \overline{\mu}_{1g}} \right)^n + w_2 \left( \frac{\mu_2 - \overline{\mu}_{2g}}{\overline{\mu}_{2b} - \overline{\mu}_{2g}} \right)^n$$

$\mu_{1g}$ and $\mu_{2g}$ are the goal values of the objectives $\mu_1$ and $\mu_2$, respectively. $\mu_{1b}$ and $\mu_{2b}$ are reference bad values of the objectives $\mu_1$ and $\mu_2$, respectively.

We set $\mu_{1g}$ = 1, 500 and $\mu_{2g}$ = 2; and we might have $\mu_{1b}$ = 8, 000 and $\mu_{2b}$ = 10.

Such a normalized formulation will have more stable numerical behavior, especially when the objectives have different orders of magnitude.

Conceptually, these formulations can be readily extended to cases of multiple objectives.

# Global Optimization & Discrete Optimization

# Global Optimization

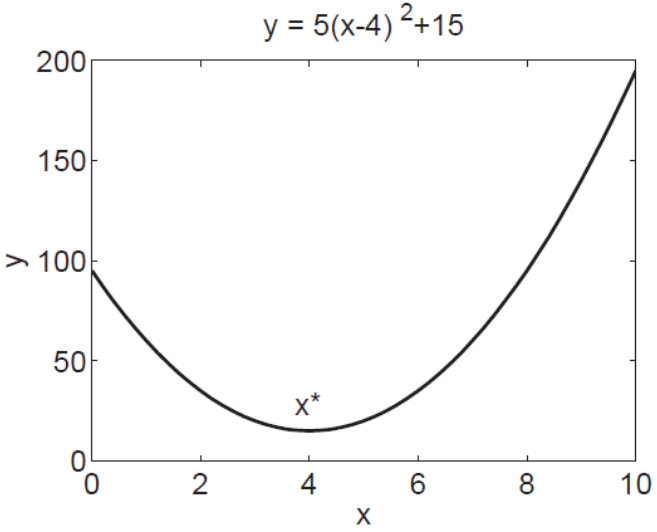
- The ***objective of global optimization*** is to find the best **global solution** in the possible or known presence of multiple local optima.

- Global optimization seeks a global solution to an unconstrained or constrained optimization problem.

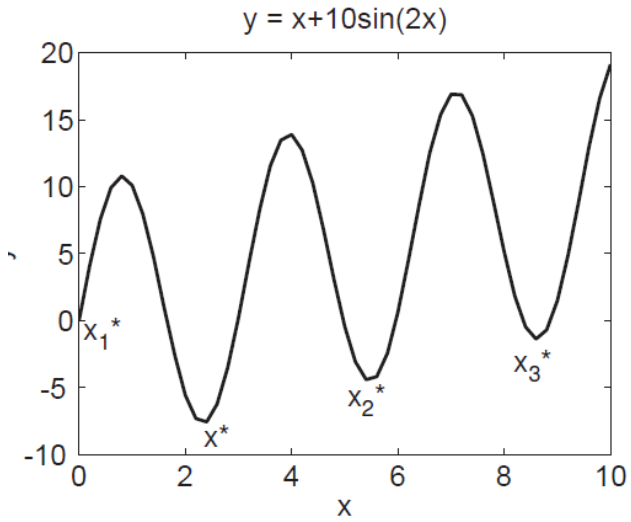
- Global search techniques are often essential for many applications, e.g., advanced engineering design, data analysis, financial planning, process control, risk management, and scientific modeling.

# Practical Issues in Global Optimization

- In ***unimodal*** optimization problem, objective function only has one local minimum, which is also its global minimum.
- In ***multimodal*** optimization problem, objective functions can have global minima and local minima.
- The objective of global optimization is to find ***global optima***.




**Unimodal Function**




**Multimodal Function**

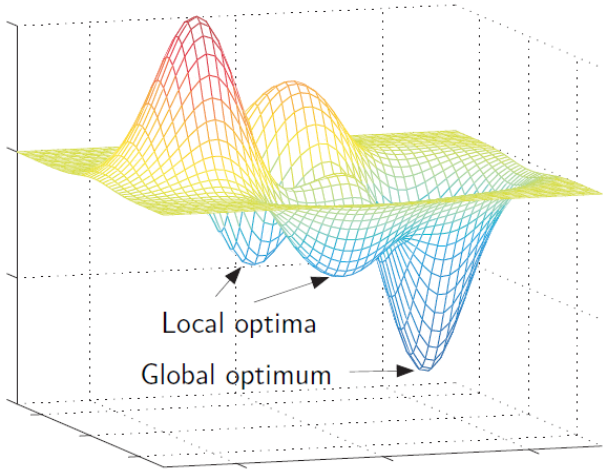
# Practical Issues in Global Optimization

- In a unimodal objective function :

The global minimum can be found by using the **gradient-based optimization methods**.

- In a multimodal objective function :

Gradient-based optimization methods will simply yield one of the local minima, which depends on which starting point was used. Finding the global optimum is **not guaranteed** when using gradient based algorithms.



# Practical Issues in Global Optimization

**In practical global optimization problems**

- The derivatives of objective functions (at some points or in certain intervals) may not be available, or may be expensive to compute.

- It is often even challenging to determine whether a highly nonlinear function is unimodal or multimodal before starting the optimization.

**Then**

Gradient-based optimization methods may not be appropriate for such global optimization problems. **Derivative-free methods**, such as **evolutionary algorithms**, may be more appropriate in these cases.

# Discrete Optimization

- In **continuous optimization** problems, the design variables were assumed to be continuous, i.e., they can assume any real values within given ranges.
- In many practical engineering problems, the acceptable values of the design variables do not form a continuous set; such problems are referred to as ***discrete optimization problems***.

**Example:**

- The number of rivets required in a riveted joint has to be an integer (such as 1, 2, 3).

# Evolutionary Algorithms

# Role of Genetic Algorithms in Global Optimization

- Evolutionary algorithms are **population-based** optimization algorithms, inspired by the principles of nature revolution.
- The **advantage** of evolutionary algorithms is that they do not make limiting assumptions about the underlying objective functions.
  - The objective functions are treated as black-box functions.
  - The definition of objective functions does not require significant insight into the structure of the design space.
- ***Evolutionary algorithms*** are a popular choice for discrete optimization because of their ability to work directly with discrete search spaces.

# Role of Genetic Algorithms in Global Optimization

- Evolutionary algorithms are based on the principles of biological evolution described in **Darwin's theory**.

- His theory was summarized as follows:

1. **Variation**: There is variation between individuals in a population.

2. **Competition**: Resources are limited. In such an environment, there will be a struggle for survival among individuals.

3. **Offspring**: Species have great fertility. They make more offspring than can grow to adulthood.

4. **Genetics**: Organisms pass genetic traits to their offspring.

5. **Natural selection**: Those individuals with the most beneficial traits are more likely to survive and reproduce.

# Role of Genetic Algorithms in Global Optimization

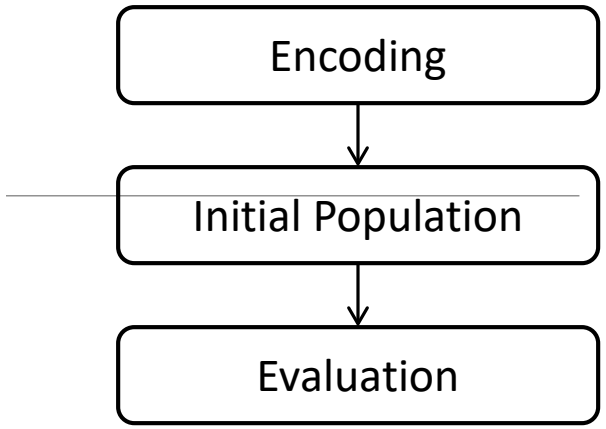
- Genetic Algorithms (GAs) are **population-based metaheuristic** optimization algorithms.
- The GA was first conceived by **J.H. Holland** in 1975.
- Genetic algorithms have been used to solve a wide range of problems, which involve continuous and discrete variables.
- Use of GAs is popular for optimization of laminate composite structures, multiobjective optimization, structural and design problems.

# Role of Genetic Algorithms in Global Optimization

- The GA uses a population of solutions, whose **individuals** are represented in forms of chromosome. The individuals in the population go through a process of **simulated evolution** to obtain the global optimum.
- The GA repeatedly modifies a set of solutions or individuals in the course of its entire run. At each iteration, the genetic algorithm selects individuals from the current population to be **parents** based on certain criteria.
- The GA uses parents to produce the next generation of individuals, called **children**. Over successive generations, the population evolves toward an optimal solution

# Role of Genetic Algorithms in Global Optimization

***The procedure of the genetic algorithm***



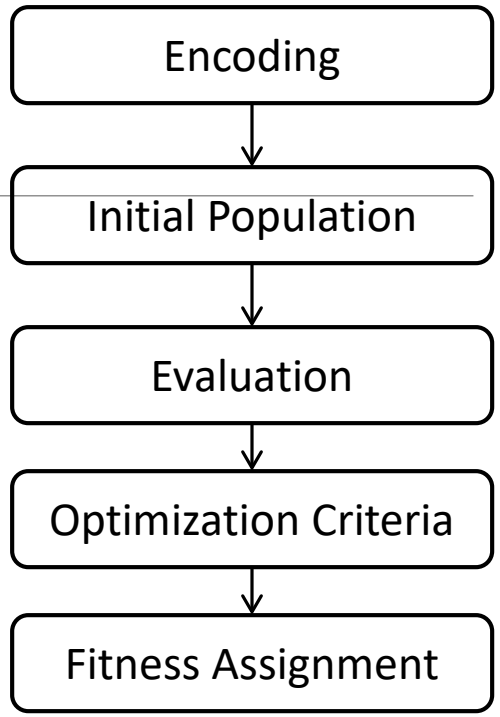

***Encoding****:* Encoding is a way to represent individuals in evolutionary algorithms. Typically, individuals are coded as a finite fixed length **string**. This string is also known in the literature as a chromosome.

***Initial population*****:** The algorithm begins by generating individuals in the design space. Prior to population initialization, the designer must choose the number of individuals in each population and the number of bits in the encoding.

***Evaluation***: Computation of the objective values for the individual solutions.

# Role of Genetic Algorithms in Global Optimization

**The procedure of the genetic algorithm**




***Optimization criteria:*** The stopping criteria of the algorithm. Examples of stopping criteria include number of generations, time limit, and function tolerance.

***Fitness assignment:*** There are several choices of fitness assignment. In rank-based fitness assignment, the individuals are sorted according to their objective values. It creates an order among the individuals.

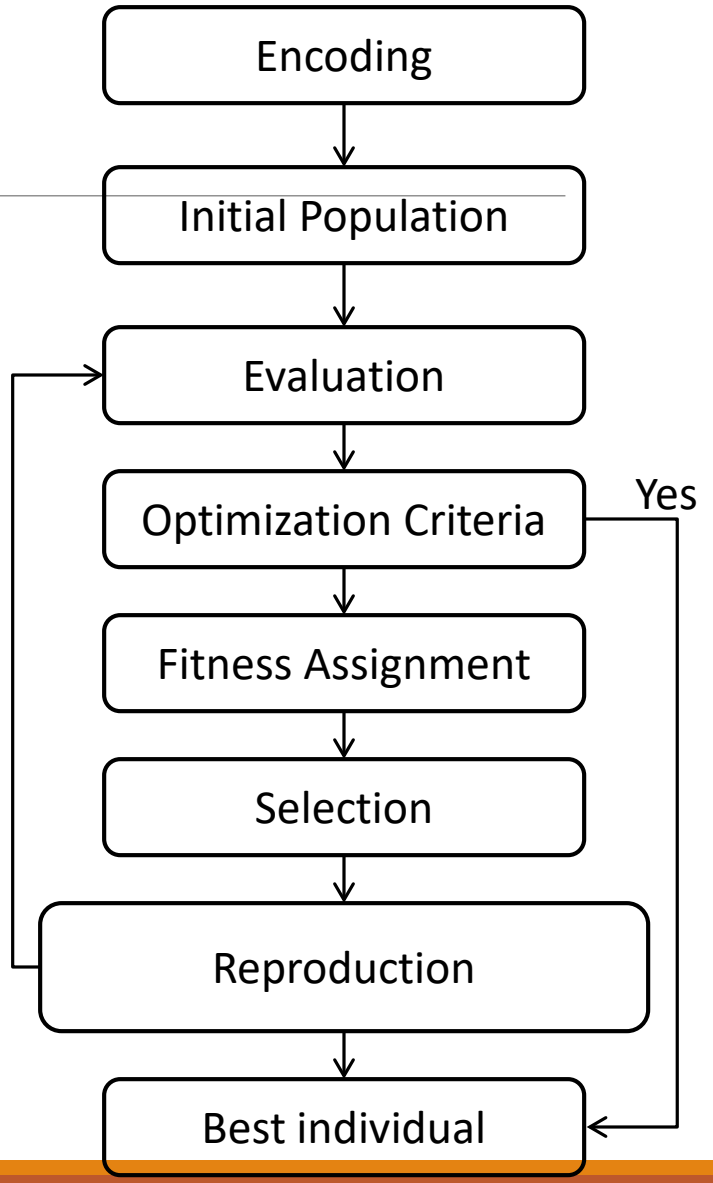
# Role of Genetic Algorithms in Global Optimization

## The procedure of the genetic algorithm

**Selection:** A selection criteria filters out the candidate solutions with bad fitness and retains those with acceptable fitness to enter the reproduction process with a higher probability.

**Reproduction:** A new generation in the genetic algorithm is created by reproduction from the previous generation. Three mechanisms (elitist, crossover, and mutation) are mainly used to create a new generation.

**Best Individual:** The global optimum satisfying the optimization criteria.

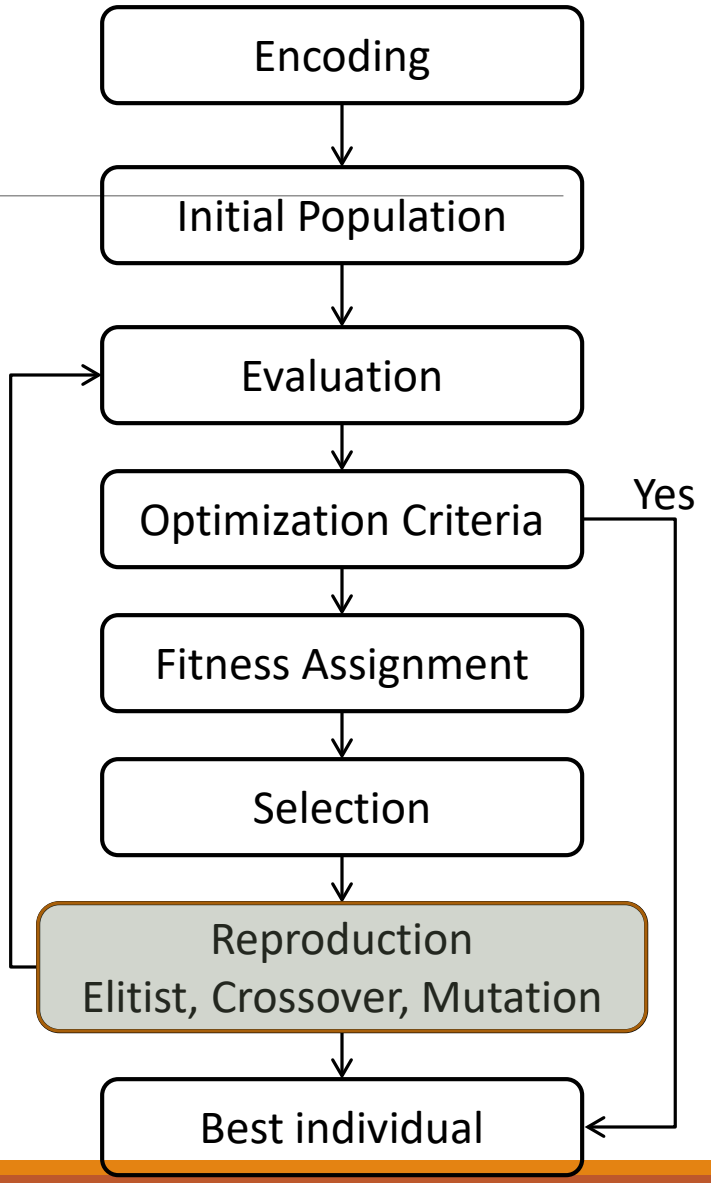


# Role of Genetic Algorithms in Global Optimization

## The procedure of the genetic algorithm

**Reproduction:**

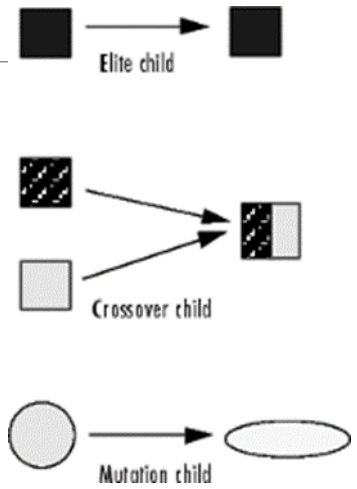
- **Elitist:** The individuals with the best fitness values in the current generation are guaranteed to survive in the next generation.
- **Crossover**: In this technique, a part of the encoded string of one individual is exchanged with the corresponding string part of another individual.
- **Mutation**: Mutated child solution is generated from a single parent by randomly reversing some bits from 0 to 1, or vice versa.



# Role of Genetic Algorithms in Global Optimization

## The procedure of the genetic algorithm

**Reproduction:**




*Example for Crossover :*

There are two individuals, 10101 and 11001. Exchange the first two bits of the two individuals. The offspring of the two individuals are 11101 and 10001.

*Example for Mutation:*

There is one individual, 10101. Through mutation, the 2nd bit and the 5th bit of 10101 are reversed. The new offspring is 11100.

# Genetic Algorithms

A GA implementation involves the following tasks:

1. **Encoding:**
   - Encoding is a method that represent individuals in evolutionary algorithms.
   - Typically, individuals are coded as a fixed length string, e.g., a binary number (0 or 1).
   - This string is also known as a chromosome.

Example:

Use a string length of 5 to code a number in binary, e.g., 10001. This string can be decoded into a base 10 decimal number as shown below.

$$1\times2^4+0\times2^3+0\times2^2+0\times2^1+1\times2^0 = 16+1 = 17$$

# Genetic Algorithms

A GA implementation:

**2. Initial population:**

- The algorithm begins by generating a random initial population.
- Important choices such as number of individuals in each population, number of bits in the encoding must be made.
- These choices govern the performance of the GA.

# Genetic Algorithms

A GA implementation:

**Example:**

- Assume six individuals in the population for the present example.
- Each individual is randomly generated by a series of five fair coin flips, heads=1 and tails=0.
- Note that five coin flips are needed because we chose to encode each individual using a 5-bit binary string.

# Genetic Algorithms

A GA implementation:

- The initial population thus generated is 10101, 11001, 01001, 11101, 10111, and 10000.
- In the decimal system, the initial population is the following set of numbers {21, 25, 9, 29, 23, 16} .
- Now evaluate the fitness function value (f the individuals in the initial population. T value set is {441, 625, 81, 841, 529, 256}

*Initial Population* →

| | Initial random population | decimal equivalent | function value |
|---|---|---|---|
| 1 | 1 0 1 0 1 | 21 | 441 |
| 2 | 1 1 0 0 1 | 25 | 625 |
| 3 | 0 1 0 0 1 | **9** | **81** |
| 4 | 1 1 1 0 1 | 29 | 841 |
| 5 | 1 0 1 1 1 | 23 | 529 |
| 6 | 1 0 0 0 0 | 16 | 256 |

# Genetic Algorithms

A GA implementation:

**3. Reproduction:**

- A new generation, called child, in the genetic algorithm is created by reproduction from the previous generation, called the parent.
- The notion of "survival of the fittest" is usually used in genetic algorithms.
- There are three main mechanisms used to create a new generation.
  - Elitist
  - Crossover
  - Mutation

# Genetic Algorithms

A GA implementation:

**3. Reproduction:**

**Elitist :**

In this approach, the individuals with the best fitness values in the current generation are guaranteed to survive

**Example:**

- ✓Out of the following six individuals in the parent population:
- ✓{10101, 11001, **01001**,11101,10111}
- ✓The third individual,01001, had the lowest function value for the current minimization problem.
- ✓Such an individual is considered elite, and will become part of the next generation.

| | Initial random population | decimal equivalent | function value |
|---|---|---|---|
| 1 | 1 0 1 0 1 | 21 | 441 |
| 2 | 1 1 0 0 1 | 25 | 625 |
| 3 | 0 1 0 0 1 | **9** | **81** |
| 4 | 1 1 1 0 1 | 29 | 841 |
| 5 | 1 0 1 1 1 | 23 | 529 |
| 6 | 1 0 0 0 0 | 16 | 256 |

→ **Elite**

# Genetic Algorithms

A GA implementation:

**3. Reproduction:**

## Crossover:

In this technique, some bits of the encoded string of one parent individual are exchanged with the corresponding bits of another parent individual.

**Example:**

- ✓ Assume that the elite individual, 01001, is part of the next generation.
- ✓ **First**, choose which individual is crossed over with which individual, e.g.,
  - individual 1 with individual 2,
  - or individual 1 with individual 3,
  - or individual 1 with individual 4.
  - This choice is usually made randomly;
  - Assume that individual 1, 10101, is crossed over with individual 2, 11001; and individual 4, 11101, is crossed over with individual 5, 10111.

# Genetic Algorithms

A GA implementation:

**3. Reproduction:**

## Crossover:

**The final choice to be made is the positions of the bits**: exchange the first three bits, or the last three bits, etc. For this example, the first three bits will be exchanged for individuals 1 and 2, and bit 2 for individuals 4 and 5 .

Reproduction

| Initial random population (parent) | Reproduction option (random choice) | New population (child) | decimal equivalent | function value |
|---|---|---|---|---|
| 1 0 1 0 1 | *crossover with 2 -- bits 1, 2, 3* | 1 1 0 0 1 | 25 | 625 |
| 1 1 0 0 1 | *crossover with 1 -- bits 1, 2, 3* | 1 0 1 0 1 | 21 | 441 |
| **0 1 0 0 1** | **elite** | 0 1 0 0 1 | 9 | 81 |
| 1 1 1 0 1 | *crossover with 5 -- bit 2* | 1 0 1 0 1 | 21 | 441 |
| 1 0 1 1 1 | *crossover with 4 -- bit 2* | 1 1 1 1 1 | 31 | 961 |
| 1 0 0 0 0 | *Mutation (bits 1, 3)* | 0 0 1 0 0 | **4** | **16** |

# Genetic Algorithms

A GA implementation:

**3. Reproduction:**

## Mutation:

Unlike crossovers (which require two parents), mutation children are generated from a single parent by randomly reversing some bits from 0 to 1, or vice versa.

In most GA implementations, a probability value for a mutation to occur is assumed.

**Example:**

Make the random choice that individual 6 goes through a mutation on bits 1 and 3.

The bits are reversed for these two positions, leading to a new child.

Reproduction

| Initial random population (parent) | Reproduction option (random choice) | New population (child) | decimal equivalent | function value |
|---|---|---|---|---|
| 1 0 1 0 1 | *crossover with 2 -- bits 1, 2, 3* | 1 1 0 0 1 | 25 | 625 |
| 1 1 0 0 1 | *crossover with 1 -- bits 1, 2, 3* | 1 0 1 0 1 | 21 | 441 |
| **0 1 0 0 1** | **elite** | 0 1 0 0 1 | 9 | 81 |
| 1 1 1 0 1 | *crossover with 5 -- bit 2* | 1 0 1 0 1 | 21 | 441 |
| 1 0 1 1 1 | *crossover with 4 -- bit 2* | 1 1 1 1 1 | 31 | 961 |
| 1 0 0 0 0 | *Mutation (bits 1, 3)* | 0 0 1 0 0 | **4** | **16** |

# Genetic Algorithms

A GA implementation:

**4.** The function values of the new population thus generated are computed.

- Using a combination of the above reproduction options, the algorithm proceeds further, until a desired stopping criterion is achieved.
- Examples of stopping criteria include number of generations, time limit, and function tolerance.

**Example:**

The best individual in the child generation shows a decrease in the function value when compared to the parent generation.

| Initial random population (parent) | Reproduction option (random choice) | New population (child) | decimal equivalent | function value |
|---|---|---|---|---|
| 1 0 1 0 1 | *crossover with 2 -- bits 1, 2, 3* | 1 1 0 0 1 | 25 | 625 |
| 1 1 0 0 1 | *crossover with 1 -- bits 1, 2, 3* | 1 0 1 0 1 | 21 | 441 |
| **0 1 0 0 1** | **elite** | 0 1 0 0 1 | 9 | 81 |
| 1 1 1 0 1 | *crossover with 5 -- bit 2* | 1 0 1 0 1 | 21 | 441 |
| 1 0 1 1 1 | *crossover with 4 -- bit 2* | 1 1 1 1 1 | 31 | 961 |
| 1 0 0 0 0 | *Mutation (bits 1, 3)* | 0 0 1 0 0 | **4** | **16** |

# Types

Evolutionary algorithms imitate living beings or adopt natural selection processes to develop powerful computational algorithms.

Some of the popular techniques that fall under the umbrella of evolutionary algorithms are :

- Genetic Algorithms (GA)
- Simulated Annealing (SA)
- Ant Colony Optimization (ACO)
- Particle Swarm Optimization (PSO)
- Tabu Search (TS)

# Reference

*Book, Optimization In Practice With Matlab® For Engineering Students And Professionals,* ***Achille Messac*** *, Cambridge University Press, 2015*

# Thanks for listening

*Any Question?*