# Assignment 3

**Developing Genetic Optimization Algorithm in C with case study**

Pedram Pasandide

Due Date: 29 July 2024

## 1 Introduction

In this assignment, we will develop an optimization algorithm. Before we start talking about what a mathematical optimization is, take a quick look at A Few Optimization Problems to see where we use them.

Optimization problems involve the process of finding the best solution from a set of possible solutions. To understand how we formulate an optimization problem you can take a look at Appendix.

Genetic Algorithm (GA) is one of the well-known optimization algorithms developed many years ago. In a genetic algorithm, a population of individuals evolves over time. The stronger individuals reproduce, and the weaker ones die. The surviving ones can also mutate. With each individual, there is a fitness number that indicates how strong that individual is. For more details, see Genetic Algorithm. A genetic algorithm terminates when a given number of iterations is reached or when a solution is found, that is, an individual with certain fitness. In this project, we'll test the developed genetic algorithm to **minimize**:

1. Griewank Function
2. Levy Function
3. Rastrigin Function
4. Schwefel Function
5. Trid Function
6. Dixon-Price Function
7. Rosenbrock Function
8. Michalewicz Function with m=10
9. Powell Function

10. Styblinski-Tang Function

You can find the formulation of above functions on here. For example Griewank Function is formulated as:

$$f(x) = f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

A 3D plot ($d = 2$ meaning a two-variable function with $x_1$ and $x_2$) of Griewank function is shown the above link as well. Visually, we might be able to see where the minimum has happened (the true answer is $f(x_1, x_2) = 0$ when $x1 = 0$ and $x_2 = 0$). The following C code implements the Griewank function for an input array `d` with `d` dimensions.

```c
#include <math.h>

double griewank(int d, double x[d]) {
 double sum = 0.0;
 double prod = 1.0;
 double result;

 for (int i = 0; i < d; i++) {
  sum += (x[i] * x[i]) / 4000.0;
  prod *= cos(x[i] / sqrt((double)(i + 1)));
 }

 result = sum - prod + 1.0;
 return result;
}
```

For an input array $(x_1, x_2, ..., x_d)$ of `{5.2, 3.4, -65.6, 7.8, -120.2}` (d=5), `griewank()` returns "$f(x) = 5.945752$." First, step of this assignment would be implementing all above functions in C and checking the results to make sure we have the correct implementations. The implementation of all the mentioned functions are inside the file `OF.c`. We will this discuss this file later. You can download all the files from my GitHub or Avenue to Learn.

GA is a popular optimization and search technique inspired by the process of natural selection and evolution. It's used to find approximate solutions (fining the values of $x_1, x_2, ..., x_d$ that minimizes $f(x)$) to complex optimization and search problems. GAs are particularly effective when dealing with large solution spaces, combinatorial optimization problems, and cases where the objective function is non-linear, discontinuous, or has multiple local optima.

Mutation and crossover functions are crucial components of Genetic Algorithms:

**Mutation:** Mutation is a random alteration of an individual's genetic code. It introduces diversity by making small changes in the solution space. For example, in a binary representation, mutation might flip a 0 to a 1 or vice versa. The mutation rate controls the probability of mutation for each gene.

**Crossover (Recombination):** Crossover involves the exchange of genetic material between two parents to create offspring. The way this exchange occurs depends on the specific crossover method used. For instance, in one-point crossover, a random point is selected in the genetic code, and the segments of the parents' genetic code are swapped at that point to create two offspring. In two-point crossover, two random points are selected for the exchange.

**Important:** Read a numerical example applying both crossover and mutation function. Also the Python code `mutation_cross.ipynb` applies the logic of a crossover and mutation function over some data for only one iteration. We'll discuss `mutation_cross.ipynb` during the next lecture.

## 2 Programming Genetic Algorithm

### 2.1 Implementation (12 points)

Files you have downloaded are:

- `OF.c` (You can change it to use the other functions but you will not submit it)
- `functions.h` (Do not change anything in it)
- `GA.c` (implement you code here)
- `functions.c` (implement you code here)

- `OF.c`: The file `OF.c` has the implementation of the objective function. The implementation of all above functions are inside this function, and depending on which one you need, you can uncomment the codes and comment the rest. The initial `OF.c` has Levy function and the rest of the functions are comments:

```
#include <math.h>
#include "functions.h"

double Objective_function(int NUM_VARIABLES, double x[NUM_VARIABLES])
```

```
{
  // some comment here
  // // ------------------------------------------------------------------------
  // // levy()
  double w[NUM_VARIABLES];
  for (int i = 0; i < NUM_VARIABLES; i++) {
   w[i] = 1.0 + (x[i] - 1.0) / 4.0;
  }

  double term1 = pow(sin(M_PI * w[0]), 2);
  double term2 = 0.0;
  for (int i = 0; i < NUM_VARIABLES - 1; i++) {
   term2 += pow(w[i] - 1.0, 2) * (1 + 10 * pow(sin(M_PI * w[i] + 1.0), 2));
  }
  double term3 = pow(w[NUM_VARIABLES - 1] - 1.0, 2) * (1 + pow(sin(2 * M_PI
      * w[NUM_VARIABLES - 1]), 2));

  return term1 + term2 + term3;

  // some comment here
}
```

This function returns the product of Levy equation. This means the objective function that will be minimized is Levy function for a set of decision variables `double x[NUM_VARIABLES]` and the dimension of `NUM_VARIABLES`.

- `functions.h`: The file `functions.h` includes the declarations of all functions. **Do NOT change anything** in this file because you will not submit this file.

- `GA.c`: The file `GA.c`, is the main code you have and based on what you need, you can call the functions in `int main(int argc, char *argv[])`. The specifications of the algorithm is given by the user. Let's say if the executable file after compiling the code is named `GA`, I must be able to run the program by the following command in the terminal:

  `./GA <POPULATION_SIZE> <MAX_GENERATIONS> <crossover_rate> <mutate_rate> <stop_criteria>`.

  If you have followed the notes so far, you must know what are the input parameters, except `stop_criteria`. The early stopping criteria in a GA is a mechanism designed to terminate the algorithm prematurely if certain conditions are met before the maximum number of generations (MAX_GENERATIONS) is reached. One common early stopping criterion is based on the difference between the current best fitness and the previous best fitness. This is often referred to as a "convergence criteria" or "stagnation criteria." For example, I must be able to run you code by:

  `./GA 1000 10000 0.5 0.1 1e-16`

where 1000 is population size, 10000 is maximum generation, 0.5 is crossover rate, the mutation rate is 0.1, and `1e-16` is the stopping criteria.

- `functions.c`: The real implementation of all functions are in `functions.c`. This is where you develop the functions you need in `GA.c`. Follow the comments inside `functions.c` and complete the functions.

In general, for a maximum number of iterations, `MAX_GENERATIONS` given by user, the algorithm in the main loop will:

1. Compute the fitness values using `compute_objective_function` function.
2. Then it compute the probability of each set of decision variables in `population`.
3. Picking the stronger set of decision variables to reproduce in `crossover` function based on `crossover_rate`.
4. Applying mutation on the population based on `mutate_rate`.
5. Until the `MAX_GENERATIONS` is reached or stopping criteria is met.

The number of decision variables in the optimization problem is set by `NUM_VARIABLES`, with upper and lower bounds depending on the function and the given bound on here. For example, if the number of decision variable is set to 10 ($d = 10$ or `NUM_VARIABLES = 10`), and the function what we want to perform optimization on it is Levy, then:

```
int NUM_VARIABLES = 10;
double Lbound[] = {-5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0};
double Ubound[] = {+5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0};
```

Every time you execute the code, at the beginning you must print out some specification about you program. Let's say I compile the code for Levy function, and I execute the program by:

`./GA 100 10000 0.5 0.1 1e-16`

After the GA obtains the best solution, you must print out the CPU time, the best solution, and the fitness value (the value of Levy function) for the best solution. This is what I see in the terminal:

```
Genetic Algorithm is initiated.
------------------------------------------------
The number of variables: 2
Lower bounds: [-5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0]
Upper bounds: [+5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0, +5.0]
```

```
Population Size:   100
Max Generations:   10000
Crossover Rate:    0.500000
Mutation Rate:     0.100000
Stopping criteria: 0.0000000000000001

Results
-----------------------------------------------
CPU time: 6.838109 seconds
Best solution found: (1.000, 0.999, 0.998, 0.998, 0.998, 1.000, 1.001, 1.000, 1.000, 0.998)
Best fitness: 0.0000012840986169
```

In this example, after 10000 iterations and about 7 seconds, the obtained optimized decision variables are:

$x1 = 1.000$, $x2 = 0.999$, $x3 = 0.998$, $x4 = 0.998$, $x5 = 0.998$, $x6 = 1.000$, $x7 = 1.001$, $x8 = 1.000$, $x9 = 1.000$, $x10 = 0.998$.

The fitness value (the objective function which in this case was Levy function) at this given point is equal to 0.0000012840986169 which is the most minimum value that GA optimizer could achieve. Fill out the following tables, include the table in your report.

Table 1: `NUM_VARIABLES = 10`, Crossover Rate = 0.5, Mutation Rate = **0.05** for **Levy** function

| Pop Size | Max Gen | Best Fitness | CPU time (Sec) |
|---|---|---|---|
| 10 | 100 | | |
| 100 | 100 | | |
| 1000 | 100 | | |
| 10000 | 100 | | |
| 1000 | 1000 | | |
| 1000 | 10000 | | |
| 1000 | 100000 | | |
| 1000 | 1000000 | | |

Table 2: `NUM_VARIABLES = 10`, Crossover Rate = 0.5, Mutation Rate = **0.2** for **Levy** function

| Pop Size | Max Gen | Best Solution | CPU time (Sec) |
|---|---|---|---|
| 10 | 100 | | |
| 100 | 100 | | |
| 1000 | 100 | | |
| 10000 | 100 | | |
| 1000 | 1000 | | |
| 1000 | 10000 | | |
| 1000 | 100000 | | |
| 1000 | 1000000 | | |

Table 3: `NUM_VARIABLES = 50`, Crossover Rate = 0.5, Mutation Rate = **0.2** for **Levy** function

| Pop Size | Max Gen | Best Solution | CPU time (Sec) |
|---|---|---|---|
| 10 | 100 | | |
| 100 | 100 | | |
| 1000 | 100 | | |
| 10000 | 100 | | |
| 1000 | 1000 | | |
| 1000 | 10000 | | |
| 1000 | 100000 | | |
| 1000 | 1000000 | | |

Conduct a case study to optimize other functions and fill out the following table as well. Don't forget to change `Lbound[]` and `Ubound[]` when you switch to another function in `OF.c`.

Table 4: `NUM_VARIABLES = 10`, Crossover Rate = CR, Mutation Rate = MR for **all** function

| Function | Pop Size | Max Gen | CR | MR | Best Solution | CPU time (Sec) |
|---|---|---|---|---|---|---|
| Griewank | | | | | | |
| Levy | | | | | | |
| Rastrigin | | | | | | |
| Schwefel | | | | | | |
| Trid | | | | | | |
| Dixon-Price | | | | | | |
| Michalewicz | | | | | | |
| Powell | | | | | | |
| Styblinski-Tang | | | | | | |

### 2.2 Report and `Makefile` (3 points)

Provide a `Makefile` to compile your code, and in the report with details tells us how your `Makefile` works. Your report **must** be in `README.tex` file. Other formats will not be accepted. Create the `README.pdf` file from `README.tex` and submit both formats. You can use LaTex file format that I have sent you. Create a section called "Appendix" in your report and include all your codes in the appendix.

### 2.3 Only the fastest program! - Bonus (+2 points)

In addition to speed, your code must also attain a satisfactory level of accuracy. This situation mirrors the workings of a genetic algorithm, where only the fittest survive! Similarly, in our context, resembling a Genetic Algorithm, the student who produces the swiftest code (at least 20% faster than their peers) will be awarded a +2 bonus.

### 2.4 Future directions

This section is not mandatory and it doesn't have any bonus. I leave this section to students who want to go deeper in this field. Create your own GA shared library on Python, using the C source codes you have developed in this assignment. Compare your algorithm with other global optimizers like Differential Evolution and Dual Annealing available on SciPy. The Python code `DE_DA_optimizer.ipynb` uses Differential Evolution and Dual Annealing optimization algorithms to minimize the same functions we had in this assignment. Create a good report and add this project with all case studies to your GitHub.

## 3 Submission on Avenue to Learn

- `GA.c`, `function.c`, and `Makefile`
- `README.tex` and `README.pdf`

# 4 Appendix

## 4.1 A Few Optimization Problems

Here are a few examples of optimization problems from different fields. The part **Optimization Method** for each example is just based on my general knowledge and they might not necessary work always!

1. **Chemical Engineering - Refinery Processes:**
   - **Objective:** Optimize the production of Liquefied Petroleum Gas (LPG), Aromatics, or Hydrogen while minimizing operating costs.
   - **Decision Variables:** Flow rates, temperatures, pressures, and compositions of various process streams, as well as catalyst selection.
   - **Constraints:** Temperature and pressure limits within specific equipment, material and energy balances, and purity requirements for the products.
   - **Optimization Method:** Nonlinear programming techniques, such as Sequential Quadratic Programming (SQP) or Genetic Algorithms, can be used to solve complex refinery optimization problems.
2. **Aerospace:**
   - **Objective:** Aircraft Wing Design for Maximum Efficiency
   - **Decision Variables:** Wing shape parameters (e.g., wing sweep, aspect ratio, airfoil shape), materials, and structural design.
   - **Constraints:** Weight constraints, stability, and strength requirements, as well as aerodynamic performance specifications.
   - **Optimization Method:** Computational fluid dynamics (CFD) simulations combined with multi-objective optimization algorithms, like multi-objective genetic algorithms, to find the trade-off between lift, drag, and structural weight.
3. **Civil Engineering - Traffic Signal Timing:**
   - **Objective:** Optimize traffic signal timings to minimize congestion and reduce travel time.
   - **Decision Variables:** Signal phase durations, cycle lengths, and offset timings for a network of traffic signals.
   - **Constraints:** Traffic flow capacity, safety constraints, and legal requirements.

- **Optimization Method:** Dynamic programming or heuristic algorithms like traffic signal control optimization to minimize traffic congestion.

4. **Mechanical Engineering - HVAC System Design:**
   - **Objective:** Optimize the design of a heating, ventilation, and air conditioning (HVAC) system to minimize energy consumption while maintaining indoor comfort.
   - **Decision Variables:** Equipment selection, duct sizing, control strategies, and insulation materials.
   - **Constraints:** Indoor temperature and humidity requirements, building occupancy, and budget constraints.
   - **Optimization Method:** Genetic algorithms or simulated annealing for multi-objective HVAC system design.
5. **Environmental Engineering - Water Treatment Plant Operation:**
   - **Objective:** Optimize the operation of a water treatment plant to minimize chemical usage and energy consumption.
   - **Decision Variables:** Flow rates, chemical dosages, treatment processes, and equipment settings.
   - **Constraints:** Water quality standards, equipment capacity, and safety regulations.
   - **Optimization Method:** Linear programming or mixed-integer programming for efficient operation of water treatment plants.
6. **Electrical Engineering - Power Grid Optimization:**
   - **Objective:** Optimize the operation of an electrical power grid to ensure reliable and efficient electricity distribution while minimizing operational costs.
   - **Decision Variables:** Power generation levels from various sources (e.g., coal, natural gas, renewables), routing of power through transmission lines, and voltage control settings.
   - **Constraints:** Power demand at different locations, line capacity limits, voltage constraints, and environmental regulations.
   - **Optimization Method:** Linear programming or mixed-integer linear programming can be used to optimize power generation and distribution in real-time, taking into account changing demand and constraints. Advanced methods may involve optimal power flow (OPF) and security-constrained optimal power flow (SCOPF) models for large-scale power grids.

These examples demonstrate the diversity of optimization problems in various engineering fields, as well as the wide range of decision variables and constraints involved in real-world applications. In each case, the objective is to find the best possible configuration or solution while considering the specific requirements and limitations of the domain.

## 4.2 What an Optimization Algorithm Is!

Optimization problems involve the process of finding the best solution from a set of possible solutions. These problems can be broadly described as follows:

**Objective Function:** Optimization problems typically begin with the definition of an objective function, which is a mathematical expression or a simulation that quantifies what you want to maximize (e.g., profit, efficiency) or minimize (e.g., cost, error). This function depends on one or more decision variables.

**Decision Variables:** Decision variables are the parameters or variables that you can adjust or control in the problem. The objective function is often expressed in terms of these variables. The goal is to find the values of these variables that optimize the objective function. Let's say if $x1$ and $x2$ are decision variables, the objective function $OF$ can be defined as **an example** like:

$$OF = \frac{exp(x_1/x_2) + sin(x_2)}{x_1^2 + \sqrt{x_2}}$$

**Constraints:** Many real-world optimization problems come with constraints, which are conditions or limitations that the solution must satisfy. Constraints can be of various types, such as equality constraints (e.g., $x1 + x2 = 10$) and inequality constraints (e.g., $x1 >= 0, x2 <= 5$). The solution must adhere to these constraints while optimizing the objective function.

**Optimization Direction:** Depending on the problem, you may be seeking to either maximize or minimize the objective function. These are known as maximization and minimization problems, respectively.

**Feasible Region:** The feasible region is the set of all possible solutions that satisfy the given constraints. It represents the space in which you can search for the optimal solution.

**Optimal Solution:** The optimal solution is the set of values for the decision variables that both satisfy the constraints and either maximize or minimize the objective function. In maximization problems, this is the highest attainable value; in minimization problems, it is the lowest attainable value.

**Search and Optimization Methods:** Solving optimization problems often involves using various mathematical and computational methods. These can include linear programming, nonlinear programming, dynamic programming, genetic algorithms, gradient descent, and many others. The choice of method depends on the problem's characteristics, such as linearity, convexity, and the number of variables and constraints.

**Sensitivity Analysis:** In some cases, it's essential to understand how sensitive the optimal solution is to changes in the problem's parameters. Sensitivity analysis helps assess how robust or fragile the solution is to variations in input data.

**Local and Global:** Local optimization algorithms focus on finding the best solution within a limited region of the solution space, often starting from an initial guess and making iterative improvements based on local information. These methods can be efficient but may get stuck in local optima and fail to find the global best solution. In contrast, global optimization algorithms aim to find the overall best solution by exploring the entire solution space, often using techniques to escape local optima and balance exploration with exploitation. Genetic algorithms, which mimic the process of natural selection, fit into the category of global optimization algorithms.

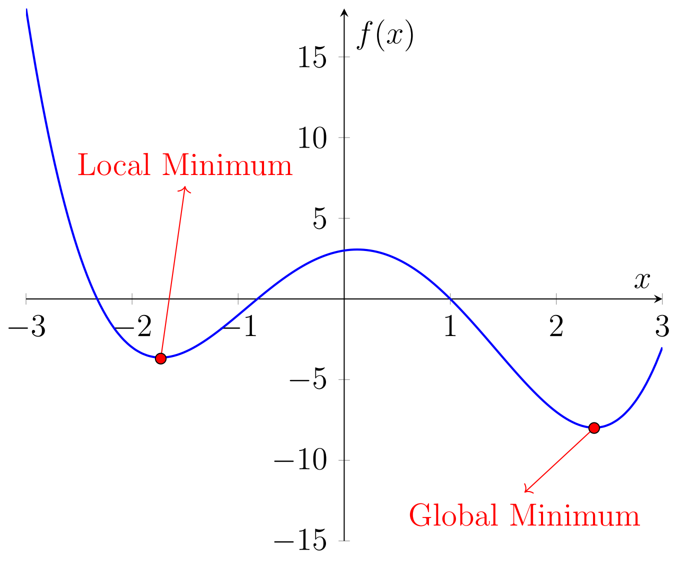

Figure 1: $f(x) = 0.5x^4 - 0.5x^3 - 4x^2 + x + 3$ with local and global minima when $x \in [-3, 3]$

**Multi-Objective Optimization:** In some scenarios, there may be multiple conflicting objectives that need to be considered simultaneously. Multi-objective optimization aims to find a set of solutions that represent trade-offs between these objectives, known as the Pareto front.

Optimization problems are ubiquitous in various fields, including engineering, economics, operations research, logistics, machine learning, and more. They play a crucial role in decision-making, resource allocation, and improving system performance. The choice of method and the complexity of solving an optimization problem depend on the specific problem's characteristics and the available computational resources.