Assignment 4

Developing Particle Swarm Optimization (PSO) in C with Case Study

Pedram Pasandide

Due Date: 2024, 5 Dec

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, please **take a look at** Appendix.

Particle Swarm Optimization (PSO) is a population-based optimization algorithm inspired by the social behavior of birds flocking or fish schooling. Developed by Kennedy and Eberhart in 1995, PSO mimics how individuals in a group adapt by sharing information to reach an optimal solution in a multi-dimensional search space. In PSO, a "swarm" of particles represents potential solutions to the optimization problem. Each particle adjusts its position based on its own best experience and the best experience of the swarm, gradually converging to an optimal or near-optimal solution. This algorithm is widely used for solving complex optimization problems due to its simplicity, efficiency, and ability to avoid local optima in many cases. In this project, we'll test the developed PSO algorithm to **minimize**:

1. Griewank
2. Levy
3. Rastrigin
4. Rosenbrock
5. Schwefel
6. Dixon-Price
7. Michalewicz with m=10
8. Styblinski-Tang

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 in the above link as well. Visually, we might be able to see where the minimum has happened (the true optimal 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 `x` with `NUM_VARIABLES` (or d) dimensions.

```c
double griewank(int NUM_VARIABLES, double *x) {
 // griewank()
 double sum = 0.0;
 double prod = 1.0;
 double result;

 for (int i = 0; i < NUM_VARIABLES; 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 "$f(x) = 5.945752$." The implementation of all above functions are given in `OF.c` declared in `OF_lib.h`. Please **do NOT change anything in `OF.c` or `OF_lib.h`** as you will not submit them and I will use the original version.

2 Particle Swarm Optimization (PSO) Formulation

In PSO, a population of particles explores the search space, where each particle represents a potential solution. The key components of the algorithm include the particle's position, velocity, personal best position, and the global best position. The algorithm updates each particle's velocity and position iteratively, guided by cognitive and social behaviors. Below is the mathematical formulation with explanations.

Step 1: Initialization

Each particle i has:

- Position: $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]$, where d is the number of **decision variables** (or **dimension**). We also call the array $\mathbf{x}_i$ a set of decision variables. $\mathbf{x}_i$ is the input array to an objective function.
- Velocity: $\mathbf{v}_i = [v_{i1}, v_{i2}, \ldots, v_{id}]$. Each variable x_{ij} in $\mathbf{x}_i$ has a velocity of v_{ij}. $\mathbf{v}_i$ shows which direction with what speed a particle $\mathbf{x}_i$ should move to potentially minimize the objective function.
- Personal best position: $\mathbf{p}_i = [p_{i1}, p_{i2}, \ldots, p_{id}]$. In each iteration, the position $\mathbf{x}_i$ of each particle is updated based on the PSO algorithm, and the objective function is recalculated using the new $\mathbf{x}_i$. However, the updated $\mathbf{x}_i$ does not necessarily yield the minimum objective function. Therefore, for each particle i, the position $\mathbf{x}_i$ that has produced the lowest objective function value so far is stored as the personal best position $\mathbf{p}_i$.
- Fitness value: $f(\mathbf{x}_i)$, which evaluates objective function for a set of decision variables $\mathbf{x}_i$.

The global best position of the swarm is $\mathbf{g} = [g_1, g_2, \ldots, g_d]$, determined by the best fitness among all particles. Each particle has a personal best $\mathbf{p}_i$. Between all $\mathbf{p}_i$s, one results the lowest objective function and it is stored as $\mathbf{g}$. This is the final solution if the maximum iteration or a stopping criteria is met.

Step 2: Velocity Update Equation

The velocity of a particle is updated as:

$$v_{ij}(t+1) = w \cdot v_{ij}(t) + c_1 \cdot r_1 \cdot (p_{ij} - x_{ij}(t)) + c_2 \cdot r_2 \cdot (g_j - x_{ij}(t))$$

where:

- $v_{ij}(t)$: Velocity of particle i in dimension j at iteration t.
- w: Inertia weight, balancing exploration and exploitation. Here, we can assume $w = 0.7$ to simplify the formulation.
- c_1: Cognitive coefficient, scaling the particle's tendency to return to its personal best. You can consider it as a constant value equal to 1.5.

- c_2: Social coefficient, scaling the particle's tendency to follow the global best. You can consider it as a constant value equal to 1.5.
- $r_1, r_2 \in [0, 1]$: Random numbers sampled uniformly for stochasticity.
- p_{ij}: Personal best position of particle i in dimension j.
- g_j: Global best position in dimension j.

Step 3: Position Update Equation

The position of each particle is updated as:

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$$

where:

- $x_{ij}(t+1)$: Updated position of particle i in dimension j.
- $v_{ij}(t+1)$: Updated velocity from the velocity update equation.

Step 4: Objective Function and Best Positions

1. Evaluate the fitness of each particle at its new position: $f(\mathbf{x}_i(t+1))$.
2. Update personal best:

$$\mathbf{p}_i = \begin{cases} \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{p}_i) \\ \mathbf{p}_i & \text{otherwise.} \end{cases}$$

3. Update global best:

$$\mathbf{g} = \arg\min_{\mathbf{p}_i} f(\mathbf{p}_i).$$

Step 5: Termination Condition

The algorithm iterates until:

- A predefined maximum number of iterations ($T_{\max}$) is reached, or

- The fitness value of the global best position is below a predefined threshold. Let's assume this threshold is equal to double precision.

There is a PSO function pseudocode in appendix PSO Pseudo Code. You don't necessarily need to follow exactly the given format. The given pseudocode is just the simplest format possible. To have more efficient results you might need to make minor changes.

3 Programming PSO

3.1 Implementation (18 points)

Implement the PSO logic to minimize the objective functions. Files you have downloaded are:

- `OF.c` and `OF_lib.h` include the objective functions. Please do not change anything here. You will not submit these files.
- `main.c`: to run the test cases. You must complete the code in this file.
- `PSO.c`: implement the PSO logic and necessary functions here.
- `utility.h`: declare all the structures you need, and functions used in `PSO.c` in this file.
- `README.tex`: you can use this Latex format to write your report.

The name of the objective function and other optimization specifications are given by the user. Let's say if the executable file after compiling the code is named `pso`, I must be able to run the program by the following format of command line in the terminal:

```
./pso <ObjectiveFunctionName> <NUM_VARIABLES> <LowerBound> <UpperBound> <NUM_PARTICLES> <MAX_ITERATIONS>
```

For example, for Griewank function with 8 variables (or dimension) with a range of decision variables from -50 (lower bound) to 50 (upper bound), with 500 particles and 1000 iterations, I can run the code with:

```
./pso griewank 8 -50 50 500 1000
```

This will give me:

```
Objective Function: griewank
The number of variables: 8
Lower Bound for all variables: -50.000000
Upper Bound for all variables:  50.000000
```

```
------------------------------------------------
PSO with is initiated:
Number of particles  = 500
Number of iterations = 1000

Results
------------------------------------------------
CPU time: 0.01 seconds
Optimal fitness: 0.032006
Optimal position: 3.1400 -4.4384 -0.0000 6.2706 -0.0000 7.6722 0.0000 -0.0000
```

The above results is showing an optimal solution of $f(\mathbf{x}_i) = 0.032006$ when $x_1 = 3.1400$, $x_2 = -4.4384$, $x_3 = -0.0000$, $x_4 = 6.2706$ $x_5 = -0.0000$, $x_6 = 7.6722$, $x_7 = 0.0000$, $x_8 = 0.0000$. However, it is mentioned here that Griewank function has a minimum $f(\mathbf{x}_i) = 0$ when all the variables in $\mathbf{x}_i$ are equal to zero. In real world optimization problems we don't necessarily know what is the lowest possible fitness value, but here we know Griewank function can result a lower value for a better position or set of decision variables. Let's try again by increasing the particle size:

```
Objective Function: griewank
The number of variables: 8
Lower Bound for all variables: -50.000000
Upper Bound for all variables:  50.000000
------------------------------------------------
PSO with is initiated:
Number of particles  = 5000
Number of iterations = 1000

Results
------------------------------------------------
CPU time: 2.15 seconds
Optimal fitness: 0.007396
Optimal position: -3.1400 -4.4384 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000 -0.0000
```

The fitness value this time is lower (and better). Let's continue increasing the particle size:

```
Objective Function: griewank
The number of variables: 8
Lower Bound for all variables: -50.000000
Upper Bound for all variables:  50.000000
------------------------------------------------
PSO with is initiated:
Number of particles  = 10000
```

```
Number of iterations = 1000

Results
------------------------------------------------
CPU time: 4.00 seconds
Optimal fitness: 0.000000
Optimal position: 0.0000 0.0000 0.0000 0.0000 0.0000 -0.0000 -0.0000 -0.000
```

Run the test cases and complete the following tables to find the optimal solutions. The functions Schwefel, Dixon-Price, Michalewicz, and Styblinski-Tang are more challenging to optimize. To achieve better results for these functions, you may need to enhance the provided pseudocode. Begin by testing the first four functions. If the algorithm performs well for them, progressively improve your PSO implementation to handle the more challenging functions.

Table 1: `NUM_VARIABLES = 10` (or dimension $d = 10$) in **all** functions

Function	Bound		Particles	Iterations	Optimal Fitness	CPU time (Sec)
	Lower	Upper				
Griewank	-600	600				
Levy	-10	10				
Rastrigin	-5.12	5.12				
Rosenbrock	-5	10				
Schwefel	-500	500				
Dixon-Price	-10	10				
Michalewicz	0	π				
Styblinski-Tang	-5	5				

Table 2: `NUM_VARIABLES = 50` (or dimension $d = 50$) in **all** functions

Function	Bound		Particles	Iterations	Optimal Fitness	CPU time (Sec)
	Lower	Upper				
Griewank	-600	600				
Levy	-10	10				
Rastrigin	-5.12	5.12				
Rosenbrock	-5	10				
Schwefel	-500	500				
Dixon-Price	-10	10				
Michalewicz	0	π				
Styblinski-Tang	-5	5				

Table 3: `NUM_VARIABLES = 100` (or dimension $d = 100$) in **all** functions

Function	Bound		Particles	Iterations	Optimal Fitness	CPU time (Sec)
	Lower	Upper				
Griewank	-600	600				
Levy	-10	10				
Rastrigin	-5.12	5.12				
Rosenbrock	-5	10				
Schwefel	-500	500				
Dixon-Price	-10	10				
Michalewicz	0	π				
Styblinski-Tang	-5	5				

3.2 Report and `Makefile` (7 points)

Write a `Makefile` to compile your code. 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 uploaded with other files. Include all above tables in your report. Create a section called "Appendix" in your report and include **ONLY** codes from `PSO.c` and `utility.h` in the appendix.

4 Submission on Avenue to Learn

Please follow the instructions carefully. Implement everything you have learned during this course. Make sure that proper flags are added into the `Makefile`, for example `-O3`.

1. `main.c`
2. `PSO.c`
3. `utility.h`
4. `Makefile`
5. `README.tex`
6. `README.pdf`

5 Appendix

5.1 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.

5.2 What is an Optimization Algorithm?

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, nonlin-

ear 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.

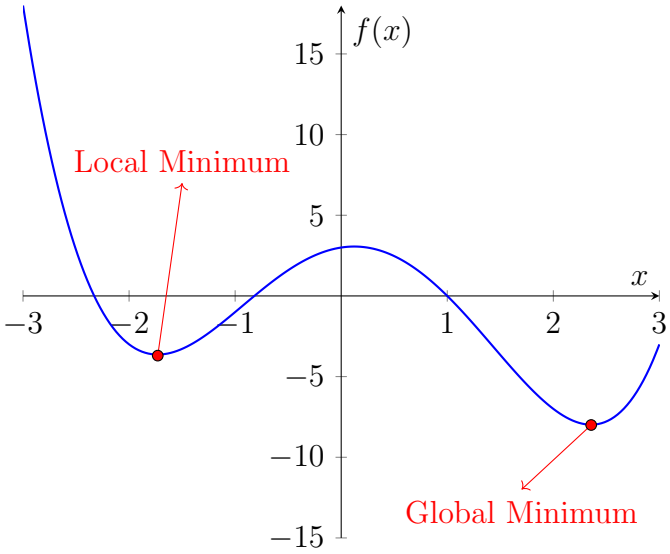

Figure 1: $f(x) = 0.5x^4 - 0.5x^3 - 4x^2 + x + 3$ with local and global minima when $x \in [-3, 3]$

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.

5.3 PSO Pseudo Code

An example of PSO algorithm:

Algorithm 1 Particle Swarm Optimization (PSO)

Initialize constants: $w \leftarrow 0.7$, $c_1 \leftarrow 1.5$, $c_2 \leftarrow 1.5$
For $i \in [1, \text{NUM\_PARTICLES}]$, $j \in [1, \text{NUM\_VARIABLES}]$, allocate memory for:
    $\mathbf{x}[i][j]$, $\mathbf{v}[i][j]$, $\mathbf{p}[i][j]$, $f_{\text{pbest}}[i]$, $\mathbf{g}$
Initialization: $f_{\text{gbest}} \leftarrow \infty$
for $i \leftarrow 1$ to NUM_PARTICLES **do**
    for $j \leftarrow 1$ to NUM_VARIABLES **do**
        $\mathbf{x}[i][j]$, $\mathbf{v}[i][j]$, and $\mathbf{p}[i][j]$
    end for
    $f_{\text{pbest}}[i] \leftarrow$ objective_function(NUM_VARIABLES, $\mathbf{x}[i]$)
    if $f_{\text{pbest}}[i] < f_{\text{gbest}}$ **then**
        $f_{\text{gbest}} \leftarrow f_{\text{pbest}}[i]$
        $\mathbf{g} \leftarrow \mathbf{p}[i]$
    end if
end for
PSO Loop:
for iter $\leftarrow 1$ to MAX_ITERATIONS **do**
    for $i \leftarrow 1$ to NUM_PARTICLES **do**
        for $j \leftarrow 1$ to NUM_VARIABLES **do**
            $r_1 \leftarrow$ random_double$(0, 1)$, $r_2 \leftarrow$ random_double$(0, 1)$
            $\mathbf{v}[i][j] \leftarrow w \cdot \mathbf{v}[i][j] + c_1 \cdot r_1 \cdot (\mathbf{p}[i][j] - \mathbf{x}[i][j]) + c_2 \cdot r_2 \cdot (\mathbf{g}[j] - \mathbf{x}[i][j])$
            $\mathbf{x}[i][j] \leftarrow \mathbf{x}[i][j] + \mathbf{v}[i][j]$
            Clamp $\mathbf{x}[i][j]$ within bounds:
                $\mathbf{x}[i][j] \leftarrow \max(\text{bounds}[j].\text{lowerBound}, \min(\text{bounds}[j].\text{upperBound}, \mathbf{x}[i][j]))$
        end for
        $f \leftarrow$ objective_function(NUM_VARIABLES, $\mathbf{x}[i]$)
        if $f < f_{\text{pbest}}[i]$ **then**
            $f_{\text{pbest}}[i] \leftarrow f$
            $\mathbf{p}[i] \leftarrow \mathbf{x}[i]$
        end if
        if $f < f_{\text{gbest}}$ **then**
            $f_{\text{gbest}} \leftarrow f$
            $\mathbf{g} \leftarrow \mathbf{x}[i]$
        end if
    end for
end for
Copy $\mathbf{g}$ to best_position and Free allocated memory
return f_{gbest}