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

Yousha Munowar

McMaster University

MECHTRON 2MP3

Pedram Pasandide

December 6th 2024

**Test Cases:**

Table 1: 10 variables/dimensions in all functions

| Function | Bound | | Particle | Iterations | Optimal Fitness | CPU time (sec) |
|---|---|---|---|---|---|---|
| | Lower | Upper | | | | |
| Griewank | -600 | 600 | 4800 | 4800 | 0.0000 | 1.5345 |
| Levy | -10 | 10 | 500 | 2000 | 0.0000 | 0.0154 |
| Rastirign | -5.12 | 5.12 | 2000 | 500 | 0.0000 | 0.3541 |
| Rosenbrock | -5 | 10 | 30000 | 1000 | 0.0000 | 10.267 |
| Sxhwefel | -500 | 500 | 40000 | 1500 | 0.0000 | 40.596 |
| Dixon-Price | -10 | 10 | 500 | 2000 | 1.0645 | 1.348 |
| Michalewicz | 0 | $\pi$ | 25000 | 8000 | -40.6542 | 5.486 |
| Styblinski-Tang | -5 | 5 | 10000 | 200 | -235.4572 | 2.796 |

Table 2: 50 variables/dimensions in all functions

| Function | Bound | | Particle | Iterations | Optimal Fitness | CPU time (sec) |
|---|---|---|---|---|---|---|
| | Lower | Upper | | | | |
| Griewank | -600 | 600 | 40000 | 6000 | 0.0000 | 1.056 |
| Levy | -10 | 10 | 20000 | 2000 | 0.0000 | 3.048 |
| Rastirign | -5.12 | 5.12 | 800000 | 20000 | 0.0000 | 3078.26 |
| Rosenbrock | -5 | 10 | 100000 | 20000 | 0.0000 | 2075.046 |
| Sxhwefel | -500 | 500 | 50000 | 20000 | 4.4521 | 4976.54 |
| Dixon-Price | -10 | 10 | 30000 | 2000 | 0.0000 | 10.563 |
| Michalewicz | 0 | $\pi$ | 375000 | 20000 | -50.4839 | 1.432 |
| Styblinski-Tang | -5 | 5 | 100000 | 20000 | -1076.464 | 7.183 |

Table 3: 100 variables/dimensions in all functions

| Function | Bound | | Particle | Iterations | Optimal Fitness | CPU time (sec) |
|---|---|---|---|---|---|---|
| | Lower | Upper | | | | |
| Griewank | -600 | 600 | 3000 | 2000 | 0.0000 | 3.483 |
| Levy | -10 | 10 | 250000 | 20000 | 0.0000 | 74.532 |
| Rastirign | -5.12 | 5.12 | 600000 | 20000 | 45.3134 | 3642.412 |
| Rosenbrock | -5 | 10 | 600000 | 20000 | 55.1024 | 2513.10 |
| Sxhwefel | -500 | 500 | 650000 | 20000 | 160.8243 | 7108.553 |
| Dixon-Price | -10 | 10 | 35000 | 4000 | 0.0000 | 170.483 |
| Michalewicz | 0 | $\pi$ | 350000 | 16000 | -50.4545 | 8.183 |
| Styblinski-Tang | -5 | 5 | 240000 | 20000 | -4043.1014 | 18.486 |

**Compiling and Running**

To compile and run this program, the following can be done in the terminal:

1. Download all the files and ensure they are all in the same directory
2. Once done, go to the terminal and simply type “make”, this should compile all the files
3. Once that is done, you can run the program using the following format
   a. ./ pso (Function Name) (Dimensions) (Lower Bound) (Upper Bound) (Number of Particles) (Max Iterations)
   b. For example
      i. ./pso griewank 8 −600 600 1000 2000

**Appendix**

Utility.h

```
#ifndef UTIL_H
#define UTIL_H

typedef double (*FitnessFunction)(int, double*);

typedef struct {
double lower_bound;
double upper_bound;
} VariableBounds;

typedef struct {
double *position; // Current position
double *velocity; // Current velocity
double *personal_best; // Best position found so far
double fitness; // Current fitness value
double best_fitness; // Personal best fitness value
} Particle;

double random_in_range(double min, double max);
void setup_particles(Particle *swarm, int num_particles, int dimensions,
VariableBounds *bounds, FitnessFunction fitness_func,
double *global_best_position, double *global_best_fitness);
void update_particles(Particle *swarm, int num_particles, int dimensions,
VariableBounds *bounds, FitnessFunction fitness_func,
        double *global_best_position, double *global_best_fitness,
        double inertia_weight, double cognitive_coeff, double social_coeff);
        void free_particles(Particle *swarm, int num_particles);
        double particle_swarm_optimize(FitnessFunction fitness_func, int dimensions,
VariableBounds *bounds,
        int num_particles, int max_iterations, double *best_position);

        #endif
```

PSO.c

```
        #include <stdlib.h>
        #include <math.h>
        #include <float.h>
        #include <stdio.h>
        #include "utility.h"

        // Generate a random value within a range
        double random_in_range(double min, double max) {
        return min + (max - min) * ((double)rand() / RAND_MAX);
        }

        // Initialize the particle swarm with random positions and velocities
        void setup_particles(Particle *swarm, int num_particles, int dimensions,
        VariableBounds *bounds, FitnessFunction fitness_func,
        double *global_best_position, double *global_best_fitness) {
        int i = 0; // Using while loop for initialization
        while (i < num_particles) {
        swarm[i].position = malloc(dimensions * sizeof(double));
        swarm[i].velocity = malloc(dimensions * sizeof(double));
        swarm[i].personal_best = malloc(dimensions * sizeof(double));
        swarm[i].best_fitness = DBL_MAX;

        for (int j = 0; j < dimensions; j++) {
        swarm[i].position[j] = random_in_range(bounds[j].lower_bound, bounds[j].upper_bound);
        swarm[i].velocity[j] = random_in_range(-1.0, 1.0);
        swarm[i].personal_best[j] = swarm[i].position[j];
        }

        swarm[i].fitness = fitness_func(dimensions, swarm[i].position);

        if (swarm[i].fitness < *global_best_fitness) {
        *global_best_fitness = swarm[i].fitness;
        for (int j = 0; j < dimensions; j++) {
```

```
        global_best_position[j] = swarm[i].position[j];
        }
        }

        i++; // Increment in while loop
        }
        }

        // Update the velocities and positions of the particles
        void update_particles(Particle *swarm, int num_particles, int dimensions,
        VariableBounds *bounds, FitnessFunction fitness_func,
        double *global_best_position, double *global_best_fitness,
        double inertia_weight, double cognitive_coeff, double social_coeff) {
        for (int i = 0; i < num_particles; i++) {
        for (int j = 0; j < dimensions; j++) {
        double rand_cognitive = random_in_range(0.0, 1.0);
        double rand_social = random_in_range(0.0, 1.0);

        swarm[i].velocity[j] = inertia_weight * swarm[i].velocity[j]
        + cognitive_coeff * rand_cognitive * (swarm[i].personal_best[j] - swarm[i].position[j])
        + social_coeff * rand_social * (global_best_position[j] - swarm[i].position[j]);

        // Clamp velocity to a maximum limit
        double max_velocity = (bounds[j].upper_bound - bounds[j].lower_bound) * 0.2;
        if (swarm[i].velocity[j] > max_velocity) swarm[i].velocity[j] = max_velocity;
        if (swarm[i].velocity[j] < -max_velocity) swarm[i].velocity[j] = -max_velocity;

        // Update position
        swarm[i].position[j] += swarm[i].velocity[j];

        // Handle boundaries with reflection
        if (swarm[i].position[j] < bounds[j].lower_bound) {
        swarm[i].position[j] = bounds[j].lower_bound + (bounds[j].lower_bound -
swarm[i].position[j]);
        swarm[i].velocity[j] *= -1;
        }
        if (swarm[i].position[j] > bounds[j].upper_bound) {
        swarm[i].position[j] = bounds[j].upper_bound - (swarm[i].position[j] -
bounds[j].upper_bound);
        swarm[i].velocity[j] *= -1;
        }
        }

        // Update fitness
```

```
    swarm[i].fitness = fitness_func(dimensions, swarm[i].position);

    // Update personal best
    if (swarm[i].fitness < swarm[i].best_fitness) {
    swarm[i].best_fitness = swarm[i].fitness;
    for (int j = 0; j < dimensions; j++) {
    swarm[i].personal_best[j] = swarm[i].position[j];
    }
    }

    // Update global best
    if (swarm[i].fitness < *global_best_fitness) {
    *global_best_fitness = swarm[i].fitness;
    for (int j = 0; j < dimensions; j++) {
    global_best_position[j] = swarm[i].position[j];
    }
    }
    }
    }

    // Free dynamically allocated memory for the swarm
    void free_particles(Particle *swarm, int num_particles) {
    for (int i = 0; i < num_particles; i++) {
    free(swarm[i].position);
    free(swarm[i].velocity);
    free(swarm[i].personal_best);
    }
    }

    // Perform Particle Swarm Optimization
    double particle_swarm_optimize(FitnessFunction fitness_func, int dimensions,
VariableBounds *bounds,
    int num_particles, int max_iterations, double *best_position) {
    Particle *swarm = malloc(num_particles * sizeof(Particle));
    double global_best_fitness = DBL_MAX;
    double *global_best_position = malloc(dimensions * sizeof(double));

    setup_particles(swarm, num_particles, dimensions, bounds, fitness_func,
    global_best_position, &global_best_fitness);

    double inertia = 0.7, cognitive = 1.5, social = 1.5;
    int patience = 50, no_improvement = 0, iteration = 0;

    while (iteration < max_iterations) {
```

```
update_particles(swarm, num_particles, dimensions, bounds, fitness_func,
global_best_position, &global_best_fitness, inertia, cognitive, social);

if (global_best_fitness < 1e-13) {
no_improvement++;
if (no_improvement >= patience) break;
} else {
no_improvement = 0;
}
iteration++;
}

for (int i = 0; i < dimensions; i++) {
best_position[i] = global_best_position[i];
}

free_particles(swarm, num_particles);
free(swarm);
free(global_best_position);

return global_best_fitness;
}
```

## References

Chatgpt. (2022, November 30). https://chatgpt.com/

*For additional information on APA Style formatting, please consult the APA Style Manual, 7th Edition.*