CS776 Assignment0 - Hill Descender

Neal Ramaswamy

September 2025

1 Introduction

1.1 Purpose

The purpose of this assignment is to implement a program to find the global minima of the Dejong functions Sphere, Rosenbrock, and Step.

2 Implementation

3 Algorithm Implementation

Initial Solution Generation The initial candidate solution is generated probabilistically from a uniform distribution that matches the range provided.

Neighbor Generation The neighboring solutions are generated through probabilistic perturbations from a normal distribution; a step size parameter is introduced to dictate the amount of variance in the generation.

Acceptance Criteria The algorithm adopts a greedy mindset - as soon as a better solution is found, it adopts that as the new solution and continues.

Termination Criteria The algorithm stops once it reaches the maximum number of iterations parameter.

3.1 Algorithm Psuedocode

The algorithm implemented follows a greedy-style, where it mimics a version of gradient descent where we are unaware of the true function. This is addressed through a normal distribution for generating the next gradient.

1. Determine the fitness of the initial guess

2. For each Iteration

(a) Set step size based on percentage of the way through the iterations we are

(b) Store the fitness of the last guess

(c) Generate a neighboring solution by adding gaussian noise to our previous guess (step size is the st. dev of the gaussian distribution).

(d) if the neighbor has a higher fitness, accept that as our current guess; else reject and keep the current best guess

3. Repeat this process for as many runs as we'd like

4 Function Implementation

Function Validation All functions were validated through test cases on their global minima and range.

4.1 Function Definitions

4.1.1 Sphere Function

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$$

4.1.2 Rosenbrock Function

$$f_2(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

4.1.3 Step Function

$$f_3(\mathbf{x}) = \sum_{i=1}^{n} floor(x_i)$$

5 Experiment

5.1 Design

5.1.1 Sphere Function Data

Runs: 10000
Iterations: 100
Fitness: 0.000000
Best Solution: (2.40172e-05 -1.07062e-05 -2.08052e-05)
Avg. Solution: (0.01940252 0.0141498 0.02825986)
St. Dev Solution (1.851892 1.850866 1.867845)

5.1.2 Rosenbrock Function Data

Runs: 10000
Iterations: 100
Dimensions = 2
Fitness: 0.000000
Best Solution: (0.999972 0.999947)
Avg. Solution: (0.30044456 0.4151165)
St.dev Solution: (0.611533 0.519806)

5.1.3 Rosenbrock Function Data

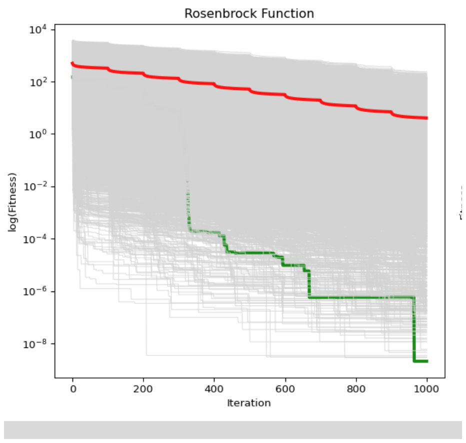
Runs: 10000
Iterations: 100
Dimensions = 5
Fitness: -26.000000
Best Solution: (-4.66352 -5.00248 -4.69556 -4.28807 -4.1959)
Avg. Solution: (-0.03623238 -0.04442598 -0.00975828 -0.00536356 -0.02421492)
St.dev Solution: (2.959615 2.960031 2.922822 2.928475 2.968451)

5.1.4 Table

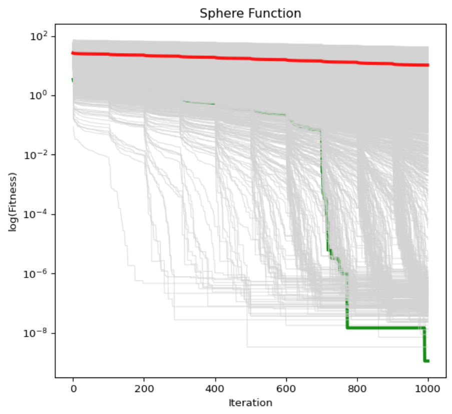
Table 1: Experimental Results for Optimization Functions

Parameter	**Sphere**	**Rosenbrock**	**Step**
Runs	10,000	10,000	10,000
Iterations	100	100	100
Dimensions	3	2	5
Range	[-5.12, 5.12]	[-2.048, 2.048]	[-5.12, 5.12]
Best Fitness	0.000000	0.000000	-26.000000
Best Solution			
x_1	2.40×10^{-5}	0.999972	-4.66352
x_2	-1.07×10^{-5}	0.999947	-5.00248
x_3	-2.08×10^{-5}	-	-4.69556
x_4	-	-	-4.28807
x_5	-	-	-4.1959
Average Solution			
x_1	0.01940252	0.30044456	-0.03623238
x_2	0.0141498	0.4151165	-0.04442598
x_3	0.02825986	-	-0.00975828
x_4	-	-	-0.00536356
x_5	-	-	-0.02421492
Standard Deviation			
x_1	1.851892	0.611533	2.959615
x_2	1.850866	0.519806	2.960031
x_3	1.867845	-	2.922822
x_4	-	-	2.928475
x_5	-	-	2.968451

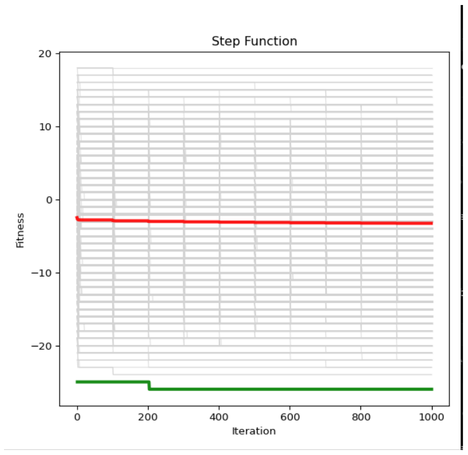
5.1.5 Data: Visualized



(a) Rosenbrock Function



(b) Sphere Function



(c) Step Function

Figure 1: Comparison of different functions

6 Discussion

How did each function perform? Sphere and Rosenbrock hill descent was successful in converging to the global minimum within 100 iterations.
Step hill descent was close, but not successful. Step ended up at a minimum of -26.

Why did certain functions perform better? The algorithm I developed functions is functionally a blinded person stopping randomly around himself to feel which way is lower. Therefore, the algorithm will function optimally on a smooth curve, and poorly in bumpy or flat regions.
Step performed extremely poorly (unable to find global minimmum despite 10000 runs) because it is entirely flat regions. This means the success is dependent on the initial guess being relatively close to global minimum.
Because my algorithm uses a uniform distribution and there are 5 dimensions, the odds of this are extremely low.

What challenges occurred?

1. The implementation was initially unable to find the global minimum effectively, so I created a step size factor adjustment. As a result, step size would be large initially, and decrease by a factor as the number of iterations went up. I hypothesize this prevented the system from bouncing out of minima.

2. The implementation was unable to find the global minimum (or even get close). From the data (Fig 1c.), it was apparent that successful solutions are based largely on luck on where the initial generation occurs, so I increased the number of runs from 30 -¿ 10000.

Improvements In the future, it may be worth exploring concurrent programming. The lack of concurrent programming made it prohibitively expensive to introduce the variance in candidate solution guessing that may have yielded the true global minimum of the step function repeatably even with fewer runs.