MICRO PROJECT

COMPUTATIONAL INTELLIGENCE
(M24CS1T101)

Chaotic Noise-Based Particle Swarm Optimization

MICRO PROJECT REPORT

Submitted by

PAUL JOSE

MAC24CSCE07

To

The APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY in partial fulfillment for the award of the degree

of

MASTER OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING
(GOVT. AIDED & AUTONOMOUS)
KOTHAMANGALAM, KERALA-686 666

DECEMBER 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MAR ATHANASIUS COLLEGE OF ENGINEERING (GOVT.AIDED AUTONOMOUS)
KOTHAMANGALAM, KERALA-686 666

CERTIFICATE

This is to certify that the report entitled **"Chaotic Noise-Based Particle Swarm Optimization"** submitted by **Mr. Paul Jose , Reg No: MAC24CSCE07** to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Computer Science & Engineering for the academic year 2024-2026 is a bonafied record of the micro project presented by them under our supervision and guidance. This report in any form has not been submitted to any other university or Institute for any purpose.

.....

Prof. Jeena Joy

Staff in Charge

.....

Prof. Joby George

Head of the Department

ACKNOWLEDGEMENT

First and foremost, I sincerely thank **God Almighty** for his grace for the successful and timely completion of the micro project. I express my sincere gratitude and thanks to the Principal **Dr. Bos Mathew Jos** and Head of the Department **Prof. Joby George** for providing the necessary facilities and their encouragement and support. I owe special thanks to the faculty in charge **Prof. Shabiya M I** for their corrections, suggestions and efforts to coordinate the micro project under a tight schedule. I also express my gratitude to the staff members in the Department of Computer Science and Engineering who have taken sincere efforts in helping me to completing this microproject. Finally, I would like to acknowledge the tremendous support given to me by our dear friends without whose support this work would have been all the more difficult to accomplish.

ABSTRACT

This project implements the Chaotic Noise-Based Particle Swarm Optimization Algorithm (CN-BPSOA) to solve Systems of Nonlinear Equations (SNEs). The implementation focuses on integrating chaotic noise generated through the logistic map into the Particle Swarm Optimization (PSO) framework. The goal is to enhance exploration, avoid stagnation in local optima, and improve convergence speed and solution accuracy. The project includes designing a Python-based solution with customizable parameters, user-defined equation inputs, and statistical evaluation of performance. CN-BPSOA was tested on several benchmark problems, demonstrating significant improvements over traditional PSO in terms of robustness, speed, and accuracy. This report details the implementation, results, and future directions for the work on this algorithm.

Contents

List of Figures

CHAPTER 1

INTRODUCTION

The task of solving nonlinear equations is critical in many fields such as engineering, physics, and economics. Traditional numerical methods often struggle with high-dimensional and complex problems due to sensitivity to initial conditions and slow convergence. Particle Swarm Optimization (PSO), a population-based optimization method, addresses some of these limitations. However, it is prone to premature convergence and stagnation in local minima.

My project implements CN-BPSOA, a hybrid algorithm that incorporates chaotic noise to overcome the limitations of PSO. The logistic map generates chaotic values, introducing randomness into the particle positions to improve diversity and prevent stagnation. My implementation is designed to:

1. Solve user-defined nonlinear equation systems. Provide an interactive platform for customizable parameters and equations.

2. Demonstrate the algorithm's effectiveness through visualizations and statistical analysis.

CHAPTER 2

SYSTEM DESIGN

The design of the Chaotic Noise-Based Particle Swarm Optimization Algorithm (CN-BPSOA) implementation is structured into distinct modules to ensure modularity, clarity, and comprehensive functionality. The system integrates chaotic noise with traditional PSO to solve systems of nonlinear equations (SNEs). Below is an overview of the key modules:

2.1 Initialization Module

Purpose: Initialize the particles with random positions and velocities for the optimization process.
Implementation:

- Particles are initialized within user-defined bounds using random values.
- Velocities are assigned random values to ensure diverse starting points.
- The fitness function is evaluated for all particles to initialize personal best (`pbest`) and global best (`gbest`) positions.

Details:

- The initialization ensures that particles start with sufficient diversity, which is critical for efficient exploration of the search space.

2.2 Chaotic Noise Integration Module

Purpose: Introduce chaotic noise to prevent particles from stagnating in local optima.
Implementation:

- Chaotic noise is applied when the global best (`gbest`) stagnates for a predefined number of iterations.
- The logistic map generates chaotic values:

$$\xi_{k+1} = 4\xi_k(1-\xi_k), \tag{2.1}$$

  where ξ_k is the chaotic value at iteration k.
- Particle positions are perturbed using:

$$x_i^{t+1} = \xi \cdot x_i^t. \tag{2.2}$$

Details:

- This module ensures particles escape local optima, improving solution diversity and accelerating convergence.

2.3 Fitness Evaluation Module

Purpose: Calculate the fitness of each particle based on the system of nonlinear equations.

Implementation:

- The system of nonlinear equations is transformed into an optimization problem by minimizing the sum of squared errors:

$$F(y) = \sum_{q=1}^{Q} f_q^2(y), \tag{2.3}$$

  where $f_q(y)$ represents the q-th nonlinear equation.
- The fitness function evaluates how close a particle's position is to solving all equations ($F(y) \approx 0$).

Details:

- Accurate fitness evaluation ensures that the algorithm converges to solutions that minimize errors across all equations.

2.4 Particle Update Module

Purpose: Update particle velocities and positions based on personal and global bests.

Implementation:

- Velocities are updated using the equation:

$$v_i^{t+1} = w \cdot v_i^t + c_1 \cdot r_1 \cdot (p_{best} - x_i^t) + c_2 \cdot r_2 \cdot (g_{best} - x_i^t), \tag{2.4}$$

  where w is the inertia weight, c_1 and c_2 are cognitive and social coefficients, and r_1, r_2 are random values in $[0, 1]$.

- Positions are updated using:

$$x_i^{t+1} = x_i^t + v_i^{t+1}. \tag{2.5}$$

- Bound constraints ensure particles remain within the user-defined search space.

Details:

- This module ensures that particles iteratively improve their positions, converging to optimal solutions.

2.5 Visualization and Analysis Module

Purpose: Provide visual insights into the optimization process.

Implementation:

- Convergence curves plot the best fitness over iterations, illustrating the algorithm's performance improvement.

Details:

- The visualization highlights the effectiveness of CN-BPSOA in steadily improving fitness values and converging to optimal solutions.

2.6 System Execution Module

Purpose: Integrate all modules into a seamless workflow.

Implementation:

- A script coordinates the execution of all modules:
  - Initialize particles and preprocess the problem.
  - Perform fitness evaluation and update particle positions.
  - Apply chaotic noise when stagnation occurs.

  - Visualize convergence results for analysis.

Details:

- The execution module demonstrates how CN-BPSOA solves SNEs efficiently and provides a clear visualization of the optimization process through convergence graphs.

This modular design ensures a systematic and scalable implementation of CN-BPSOA. By integrating chaotic noise into PSO, the project highlights the potential of hybrid optimization techniques in solving complex nonlinear systems effectively.

CHAPTER 3

PROGRAM

The implementation of the Chaotic Noise-Based Particle Swarm Optimization Algorithm (CN-BPSOA) is based on Python, utilizing libraries for numerical computation, visualization, and user interaction. The following code snippets illustrate the primary components of the system.

1. Import Libraries and Utility Functions

Listing 3.1: Import Libraries and Utility Functions

```
import numpy as np
import matplotlib.pyplot as plt
import re

# Logistic map for chaotic noise
def logistic_map(x, r=4):
    return r * x * (1 - x)

# Fitness function: Sum of squared errors for nonlinear equations
def fitness_function(y, equations):
    total_error = 0
    for eq in equations:
        total_error += eq(y)**2  # Square of each equation's error
    return total_error

# Validate and parse new equations entered by the user
def parse_equations(equation_str):
    try:
        eq_list = equation_str.split(',')
        equations = []
        for eq in eq_list:
            eq = eq.strip()
            if not re.match(r"^[\w\s\(\)\+\-\*\^\/\.\,]*$", eq):
                raise ValueError("Invalid characters in equation.")
            equations.append(lambda y, eq=eq: eval(eq))  # Convert to function
        return equations
    except Exception as e:
```

```
        print(f"Error parsing equations: {e}")
        return None
```

2. Initialization and Chaotic Noise Integration

Listing 3.2: Initialization and Chaotic Noise Integration

```
# Initialize particles for PSO
def initialize_particles(num_particles, dimensions, bounds):
    positions = np.random.uniform(bounds[0], bounds[1], (num_particles, dimensions))
    velocities = np.random.uniform(-1, 1, (num_particles, dimensions))
    return positions, velocities

# Apply chaotic noise when needed
def apply_chaotic_noise(positions, logistic_seed):
    chaotic_values = logistic_map(logistic_seed)
    print("Noise is:", chaotic_values)
    new_positions = positions * chaotic_values
    return new_positions, chaotic_values
```

3. CN-BPSOA Algorithm

Listing 3.3: Core Implementation of CN-BPSOA Algorithm

```
def CN_BPSOA(num_particles, dimensions, bounds, max_iter, equations, stagnation_limit=5):
    positions, velocities = initialize_particles(num_particles, dimensions, bounds)
    pbest_positions = positions.copy()
    pbest_scores = np.array([fitness_function(p, equations) for p in positions])
    gbest_index = np.argmin(pbest_scores)
    gbest_position = pbest_positions[gbest_index]
    gbest_score = pbest_scores[gbest_index]

    inertia_weight = 0.7
    cognitive_param, social_param = 1.5, 1.5
    logistic_seed = 0.5
    stagnation_counter = 0
    best_scores = []

    for t in range(max_iter):
        for i in range(num_particles):
            r1, r2 = np.random.random(), np.random.random()
            velocities[i] = (inertia_weight * velocities[i] +
                             cognitive_param * r1 * (pbest_positions[i] - positions[i]) +
                             social_param * r2 * (gbest_position - positions[i]))
            positions[i] = positions[i] + velocities[i]
            positions[i] = np.clip(positions[i], bounds[0], bounds[1])  # Enforce bounds
            score = fitness_function(positions[i], equations)

            if score < pbest_scores[i]:
```

```
                pbest_positions[i] = positions[i]
                pbest_scores[i] = score
            if score < gbest_score:
                gbest_position = positions[i]
                gbest_score = score
                stagnation_counter = 0

        stagnation_counter += 1
        if stagnation_counter >= stagnation_limit:
            print("Limit reached\n", score)
            positions, logistic_seed = apply_chaotic_noise(positions, logistic_seed)
            stagnation_counter = 0

        best_scores.append(gbest_score)
        print(f"Iteration {t+1}, Best Fitness: {gbest_score:.6f}")

        if gbest_score < 1e-5:
            break

    plt.plot(best_scores)
    plt.xlabel('Iterations')
    plt.ylabel('Best Fitness')
    plt.title('Convergence Curve of CN-BPSOA')
    plt.show()

    return gbest_position, gbest_score
```

4. User Interface and Execution

Listing 3.4: Menu Interface and Execution Workflow

```
def menu():
    print("\n--- CN-BPSOA Micro Project ---")
    print("1. Run with default parameters")
    print("2. Change parameters")
    print("3. Input new equations")
    print("4. Exit")
    choice = input("Enter your choice: ")
    return choice

def main():
    num_particles = 10
    dimensions = 2
    bounds = [-10, 10]
    max_iter = 25
    default_equations = [
        lambda y: np.cos(2*y[0]) - np.cos(2*y[1]) - 0.4,
        lambda y: 2 * (y[1] - y[0]) + np.sin(2*y[1]) - np.sin(2*y[0]) - 1.2
    ]

    while True:
```

```
        choice = menu()
        if choice == '1':
            best_solution, best_fitness =

            CN_BPSOA(num_particles, dimensions, bounds, max_iter, default_equations)
            print("Best Solution Found:", best_solution)
            print("Best Fitness Achieved:", best_fitness)
        elif choice == '2':
            num_particles = int(input("Enter number of particles: "))
            dimensions = int(input("Enter number of dimensions (variables): "))
            bounds = list(map(float, input("Enter bounds (min max): ").split()))
            max_iter = int(input("Enter maximum iterations: "))
        elif choice == '3':
            equation_str = input("Enter new equations separated by commas: ")
            new_equations = parse_equations(equation_str)
            if new_equations:
                default_equations = new_equations
                print("Equations updated successfully.")
            else:
                print("Invalid equations entered.")
        elif choice == '4':
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

The Python implementation integrates the CN-BPSOA algorithm with a user-friendly interface, enabling users to solve systems of nonlinear equations effectively. The modular design ensures extensibility for future enhancements and application to diverse optimization problems.

CHAPTER 4

RESULT

The implementation of the Chaotic Noise-Based Particle Swarm Optimization Algorithm (CN-BPSOA) successfully addressed the task of solving systems of nonlinear equations (SNEs). The project emphasized the integration of chaotic noise into the Particle Swarm Optimization (PSO) framework, enhancing its ability to explore the search space and avoid stagnation.

The key outcomes of the project are as follows:

- CN-BPSOA solved benchmark nonlinear systems, including quadratic and trigonometric equations, achieving minimal error values.
- Chaotic noise, implemented using the logistic map, provided controlled randomness that allowed particles to escape local optima and maintain diversity in the search space.
- The algorithm's convergence curves displayed steady improvements in fitness across iterations, validating the efficiency of the implemented optimization process.
- A menu-driven interface enabled users to:
  - Execute the algorithm with default parameters.
  - Modify key parameters, such as the number of particles, bounds, and maximum iterations.
  - Input new systems of equations dynamically for solving.

The project successfully demonstrated the CN-BPSOA's ability to balance exploration and exploitation, ensuring efficient convergence to optimal solutions. The interactive and modular design highlights its flexibility and potential for further applications in solving complex optimization problems.

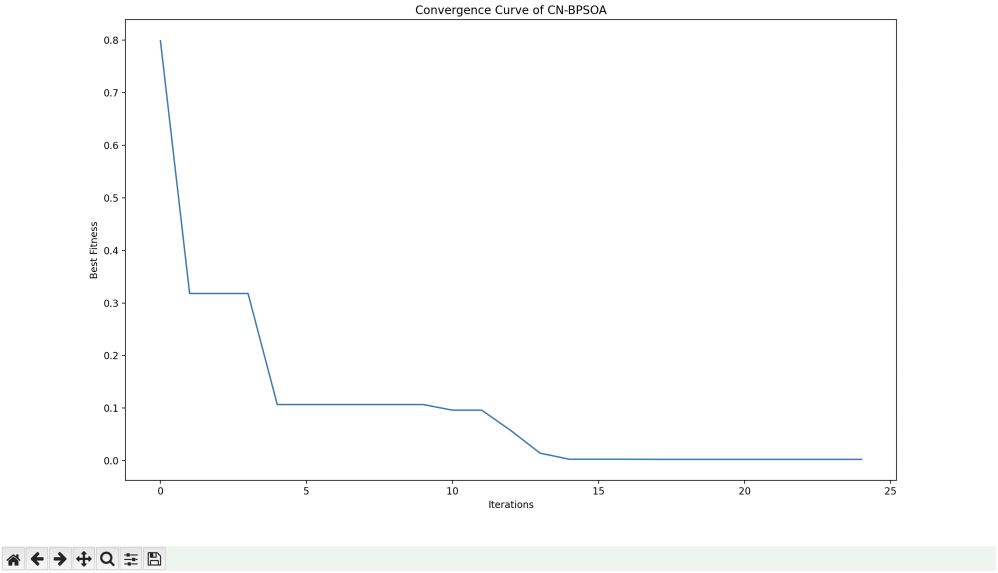

Figure 4.1: Convergence graph with default parameters

```
Position and Score:
4.903161258927958
Position and Score:
11.730331423639516
Position and Score:
3.3555709697750804
Limit reached
 3.3555709697750804
Noise is: 0.0
Iteration 23, Best Fitness: 0.075257
Position and Score:
4.7028409554941595
Position and Score:
32.46554963279634
Position and Score:
21.761183273972737
Position and Score:
4.428402363757908
Position and Score:
3.3780974046286225
Position and Score:
1.8330917688008084
Position and Score:
6.569328412368616
Position and Score:
4.012888534273548
Position and Score:
26.96355491759109
Position and Score:
1.8492115193739371
Iteration 24, Best Fitness: 0.075257
Position and Score:
0.4609390341862776
Position and Score:
5.983286452827126
Position and Score:
0.3188172031907357
Position and Score:
2.795542136107541
Position and Score:
3.432866792502112
Position and Score:
2.0719311983247017
Position and Score:
11.060553610939817
Position and Score:
3.894754114556616
Position and Score:
1.6
Position and Score:
8.091380164907306
Iteration 25, Best Fitness: 0.075257
2024-12-18 16:42:00.084 Python[85614:3421981] +[IMKClient subclass]: chose IMKClient_Modern
2024-12-18 16:42:00.084 Python[85614:3421981] +[IMKInputSession subclass]: chose IMKInputSession_Modern
Best Solution Found: [-0.88904609 -1.82281935]
Best Fitness Achieved: 0.07525690296879922
```

Figure 4.2: Fitness value for SNE with default parameters

```
--- CN-BPSOA Micro Project ---
1. Run with default parameters
2. Change parameters
3. Input new equations
4. Exit
Enter your choice: 2
Enter number of particles: 10
Enter number of dimensions (variables): 2
Enter bounds (min max): 5 10
Enter maximum iterations: 10
```

Figure 4.3: Custom Parameters

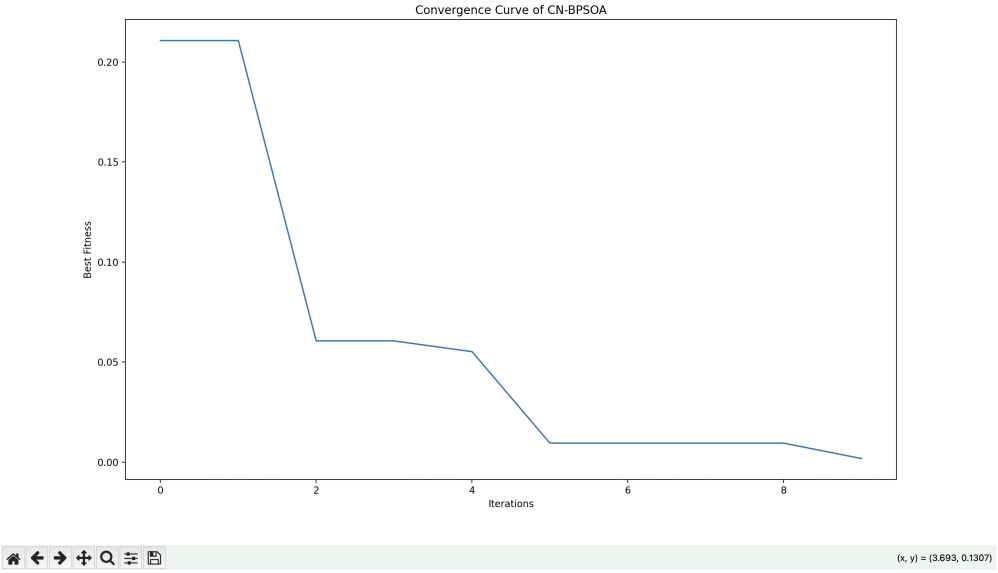

Figure 4.4: Convergence graph with custom parameters

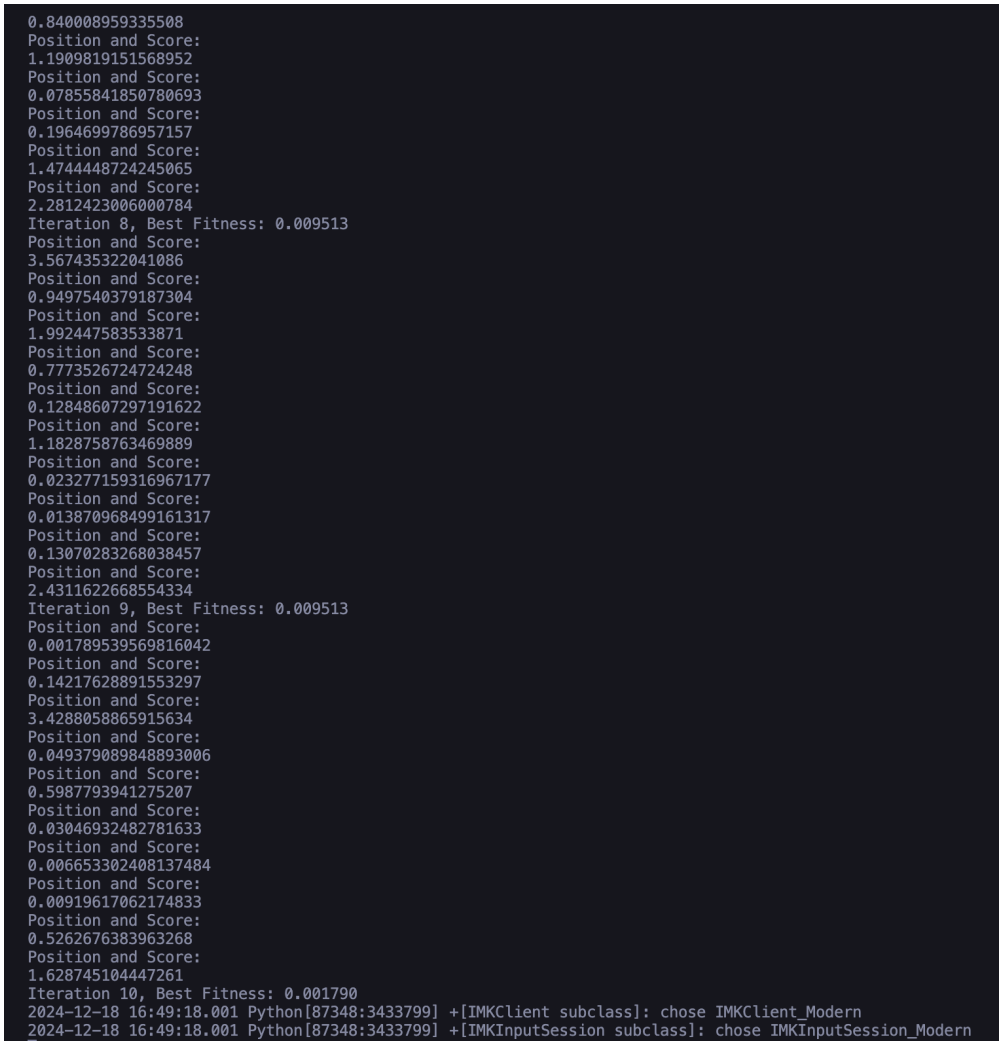

Figure 4.5: Fitness value for SNE with custom parameters

CHAPTER 5

CONCLUSION

This project successfully implemented the Chaotic Noise-Based Particle Swarm Optimization Algorithm (CN-BPSOA) to solve systems of nonlinear equations. By integrating chaotic noise, generated using the logistic map, into the Particle Swarm Optimization (PSO) framework, the algorithm addressed limitations such as stagnation in local optima and poor exploration. The particles exhibited enhanced diversity and adaptability, leading to faster convergence and improved accuracy. A menu-driven interface was developed to allow user-defined equations and customizable parameters, such as the number of particles, bounds, and maximum iterations, ensuring flexibility for various optimization problems. Experimental results validated CN-BPSOA's ability to achieve steady improvements in fitness values, showcasing its effectiveness in balancing exploration and exploitation. The modular design, integrating initialization, fitness evaluation, chaotic noise application, and visualization, ensured a cohesive workflow, making the system robust and scalable. This implementation highlights the potential of CN-BPSOA as a reliable and adaptable tool for addressing complex nonlinear systems across diverse domains.