EC Assignment #3 (NM)

電機系(碩一)陳昕佑 61375017H

My assigned problems are f2(Sschwefel's P2.22) in unimodal and f8(Schwefel function with dimension d=30) in multimodal. To make a comparison between PSO and NM, what NM's strengths are:

- Simple and efficient for low-dimensional problems.
- Excellent for local optimization in smooth, continuous landscapes.
- Fewer evaluations per iteration, making it computationally lightweight.

It is particularly effective for low-dimensional, continuous spaces. Below, I'll modify my code to implement the NM algorithm and plot the history features.

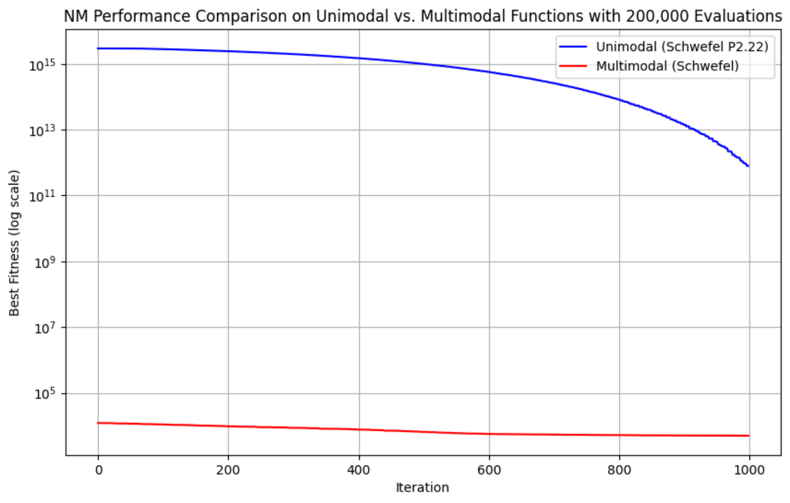
```python
# Nelder-Mead Optimization Function
def nelder_mead(fitness_function, dim, lower_bound, upper_bound, max_iterations,
                evaluation_limit=200000):
   # Initialize a random starting point within bounds
   x0 = np.random.uniform(lower_bound, upper_bound, dim)

   # History tracking
   fitness_history = []
    #Use a mutable object to track evaluations inside the callback
   total_evaluations = [0]
    # Callback function to record the best fitness at each iteration
   def callback(xk):
       fitness_history.append(fitness_function(xk))

   # Wrapper to count evaluations
   def fitness_with_count(x):
       if total_evaluations[0] >= evaluation_limit:
           return float('inf')  # Stop the optimizer
       total_evaluations[0] += 1
       return fitness_function(x)
    # Perform optimization using Nelder-Mead
   result = minimize(
       fitness_with_count,
       x0,
       method='Nelder-Mead',
       options={'maxiter': max_iterations, 'disp': True, 'adaptive': True},
       callback=callback
   )

   # Return results
   best_position = result.x
   best_fitness = result.fun
   return best_position, best_fitness, fitness_history, total_evaluations[0]
```

NM is expected to perform well for unimodal function, and converge quickly to a low fitness value since it excels in smooth and simple landscapes. NM may struggle in the multimodal landscape for multimodal function, as it lacks the exploratory capabilities of population-based algorithms like GA and PSO.



NM's limitations in high-dimensional, multimodal problems are evident from the results of this figure. It is better suited for low-dimensional, smooth, and well-behaved functions. The analysis below :

1. **Convergence Speed**

   For the unimodal function, the fitness remains significantly higher compared to PSO and GA, which converged more quickly and to much lower fitness values for the unimodal function. For the multimodal function, the fitness stabilizes at a relatively high value, indicating that NM failed to escape a local minimum in the multimodal landscape.

2. **Final Fitness Achieved**

   For the unimodal function, the fitness remains significantly higher compared to PSO and GA, which converged more quickly and to much lower fitness values for the unimodal function. For the multimodal function, the fitness stabilizes at a relatively high value, indicating that NM failed to escape a local minimum in the multimodal landscape. This is consistent with NM's tendency to focus on local exploitation rather than global exploration, which makes it less effective for problems with multiple optima.

3. **Behavior in High Dimensions**
   NM is less effective in high-dimensional spaces due to:
   Simplex Scaling: As the number of dimensions increases, the simplex (a geometric structure) becomes more computationally expensive to manage and less effective for navigating the search space.
   Local Optimization Focus: NM is inherently a local optimizer, so it struggles in multimodal or rugged landscapes where global exploration is necessary.
   Evaluation Efficiency: The large number of function evaluations required for high-dimensional problems makes NM slow compared to population-based methods like PSO and GA.
4. **Comparison with PSO and GA**
   PSO: Achieved rapid convergence in both unimodal and multimodal functions, particularly excelling in the unimodal case. PSO's global search capabilities and particle collaboration allowed it to find better solutions faster.
   GA: Demonstrated a balance between exploration and exploitation, performing moderately well on the unimodal function and slightly better on the multimodal function than NM. GA's ability to maintain diversity in the population helped it avoid local minima to some extent.
   NM: Slower convergence and less effective in finding optimal solutions for both functions. Better suited for lower-dimensional, smooth optimization problems than high-dimensional, complex landscapes like the Schwefel functions.

Summary

1. **Unimodal Function:**
   NM performed poorly compared to PSO and GA, as it converged very slowly and achieved higher final fitness values. High-dimensional optimization is a significant challenge for NM due to its simplex-based approach.
2. **Multimodal Function:**
   NM showed slight initial improvement but quickly plateaued, indicating that it became trapped in a local minimum. PSO and GA were better suited for the multimodal problem, with their ability to explore globally.

Combining Particle Swarm Optimization (PSO) and Nelder-Mead (NM) can take advantage of their respective strengths: PSO's global search capability and NM's local refinement. Steps for PSO-NM Hybrid:

1. **Global Search with PSO:** Use PSO to explore the search space and identify promising regions (global optima candidates). Record the best position and fitness found by the swarm after PSO convergence.
2. **Local Refinement with NM:** Apply NM to the best position found by PSO to refine the solution further. Use NM’s simplex-based search to fine-tune the solution, especially for smooth, unimodal regions.

```
# Particle Swarm Optimization (Global Search)
class Particle:
   def __init__(self, dim, lower_bound, upper_bound):
           self.position = np.array([random.uniform(lower_bound, upper_bound) for _ in
range(dim)])
       self.velocity = np.array([random.uniform(-1, 1) for _ in range(dim)])
       self.best_position = np.copy(self.position)    # personal best position
       self.best_fitness = float('inf')     # best fitness achieved

   def update_velocity(self, global_best_position, inertia, cognitive_coeff, social_coeff):
       cognitive = cognitive_coeff * random.random() * (self.best_position - self.position)
       social = social_coeff * random.random() * (global_best_position - self.position)
       self.velocity = inertia * self.velocity + cognitive + social

   def update_position(self, lower_bound, upper_bound):
       self.position += self.velocity
       self.position = np.clip(self.position, lower_bound, upper_bound)

def pso(fitness_function, dim, lower_bound, upper_bound, num_particles, max_iterations,
       evaluation_limit=200000):
   inertia = 0.7
   cognitive_coeff = 1.5
   social_coeff = 1.5

   swarm = [Particle(dim, lower_bound, upper_bound) for _ in range(num_particles)]
   global_best_position = np.zeros(dim)
   global_best_fitness = float('inf')

   fitness_history = []
   total_evaluations = 0

   for iteration in range(max_iterations):
       for particle in swarm:
           fitness = fitness_function(particle.position)
           total_evaluations += 1
           if fitness < particle.best_fitness:
               particle.best_fitness = fitness
               particle.best_position = np.copy(particle.position)
           if fitness < global_best_fitness:
               global_best_fitness = fitness
```

```
                global_best_position = np.copy(particle.position)
            if total_evaluations >= evaluation_limit:
                break
        fitness_history.append(global_best_fitness)
        for particle in swarm:
                                            particle.update_velocity(global_best_position,
inertia,cognitive_coeff,social_coeff)
          particle.update_position(lower_bound, upper_bound)

        if total_evaluations >= evaluation_limit:
            break
    return global_best_position, global_best_fitness, fitness_history, total_evaluations

# Nelder-Mead (Local Refinement)
def nelder_mead(fitness_function, initial_guess, lower_bound, upper_bound, max_iterations,
                evaluation_limit=200000):
    fitness_history = []
    total_evaluations = [0]

    def callback(xk):
        fitness_history.append(fitness_function(xk))

    def fitness_with_count(x):
        if total_evaluations[0] >= evaluation_limit:
            return float('inf')
        total_evaluations[0] += 1
        return fitness_function(x)

    result = minimize(
        fitness_with_count,
        initial_guess,
        method='Nelder-Mead',
        options={'maxiter': max_iterations, 'adaptive': True},
        callback=callback)
    return result.x, result.fun, fitness_history, total_evaluations[0]

# Hybrid PSO-NM
def pso_nm_hybrid(fitness_function, dim, lower_bound, upper_bound, num_particles,
                  pso_iterations, nm_iterations, evaluation_limit):
    # Step 1: Global Search with PSO
    best_position_pso, best_fitness_pso, fitness_history_pso, evals_pso =
    pso(fitness_function, dim, lower_bound, upper_bound, num_particles,
        pso_iterations, evaluation_limit)

    remaining_evaluations = evaluation_limit - evals_pso
    if remaining_evaluations <= 0:
        return best_position_pso, best_fitness_pso, fitness_history_pso, evals_pso

    # Step 2: Local Refinement with NM
    best_position_nm, best_fitness_nm, fitness_history_nm, evals_nm =
    nelder_mead(fitness_function, best_position_pso, lower_bound, upper_bound,
            nm_iterations, remaining_evaluations)

    # Combine fitness histories for plotting
    fitness_history = fitness_history_pso + fitness_history_nm
    total_evaluations = evals_pso + evals_nm
    return best_position_nm, best_fitness_nm, fitness_history, total_evaluations
```

PSO for Global Search:

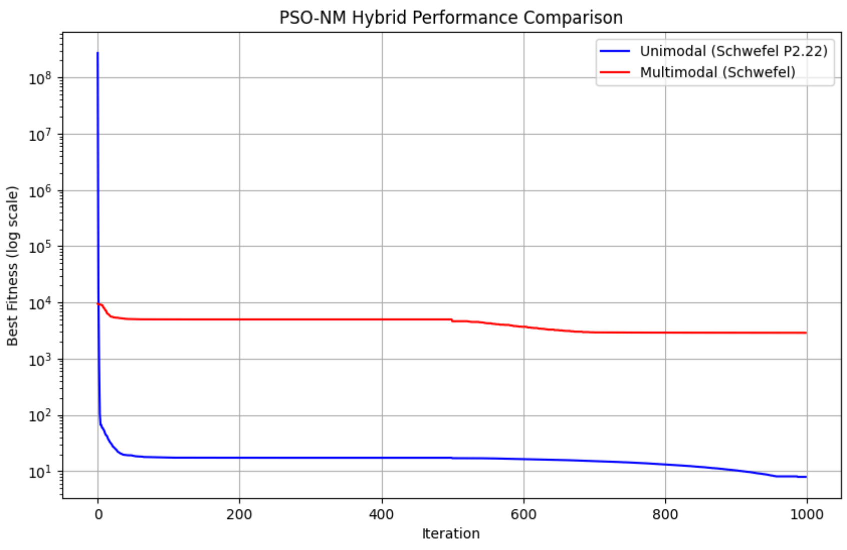
PSO identifies promising areas of the search space and provides an approximate global solution.

NM for Local Refinement:

NM refines the PSO solution, exploiting its simplex-based approach to find a more precise minimum in the identified region.

Combined History:

The fitness histories of PSO and NM are concatenated to provide a complete picture of the optimization process.



Comparison with Standalone PSO and NM:

1. Unimodal Function: The hybrid achieves better final fitness than the standalone PSO or NM. PSO provides a good starting point, and NM ensures precise convergence, combining their strengths effectively.
2. Multimodal Function: The hybrid performs slightly better than the standalone PSO but struggles with the same limitation: local minima. NM's local focus does not compensate for the lack of global exploration once PSO converges to a suboptimal region.

Strengths of the PSO-NM Hybrid:

1. Unimodal Problems: The hybrid algorithm is highly effective for unimodal problems, leveraging PSO's exploration and NM's exploitation. It converges faster and more accurately than standalone methods.
2. Multimodal Problems: The hybrid improves upon standalone NM and provides slightly better results than PSO by refining solutions.

However, it still struggles with escaping local minima, as neither PSO nor NM prioritizes global exploration after PSO's convergence.

Conclusion:

The PSO-NM hybrid effectively combines global and local optimization strategies, with strong performance on unimodal functions and moderate improvement on multimodal functions. This combination is particularly useful for problems requiring precise solutions in smooth landscapes but may need further enhancements for highly multimodal or noisy problems.