Department of Mathematics and Computer Science
University of Southern Denmark, Odense

March 12, 2025
Marco Chiarandini

AI505 – Optimization

Sheet 04, Spring 2025

Solution:
Included.
Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

Exercise 1* Consider the natural evolutionary strategy for an univariate function. Assume the univariate normal distribution as proposal distribution $p(x \mid \theta) = \mathcal{N}(x \mid \mu, \sigma^2)$.

- Derive the update rule for θ
- If after a number of iterations the value of μ becomes equal to x^*, that is, the minimum of f, what will be the update rule for σ^2 and what will be the difficulty encountered by the algorithm?

Solution:
The Gaussian distribution with mean μ and variance σ^2 has probability density function:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

The log-likelihood of a single value sampled from it is:

$$L(x \mid \mu, \sigma^2) = \log p(x \mid \mu, \sigma^2) = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln\sigma^2 - \frac{(x-\mu)^2}{2\sigma^2}$$

For a set of opints $X = \{x_1, \ldots, x_N\}$ draw independently from the distribution we have

$$p(X \mid \mu, \sigma^2) = \prod_{i=1}^{N} p(x_i \mid \mu, \sigma^2)$$

$$L(X \mid \mu, \sigma^2) = \log \prod_{i=1}^{N} p(x_i \mid \mu, \sigma^2) = \sum_{i=1}^{N} \left(-\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln\sigma^2 - \frac{(x_i-\mu)^2}{2\sigma^2} \right)$$

Exercise 2* **(8.4)**

The maximum likelihood estimates are the parameter values that maximize the likelihood of sampling the points $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$.
Derive the maximum likelihood estimate for the cross-entropy method that uses multivariate normal distributions: $\mathcal{N}(\mathbf{x} \mid \mu, \Sigma)$.

Solution:

$$L(\mathbf{x} \mid \mu, \Sigma) = -\frac{1}{2}\ln 2\pi - \frac{1}{2}\ln\Sigma - \frac{(\mathbf{x}-\mu)^2}{2}$$

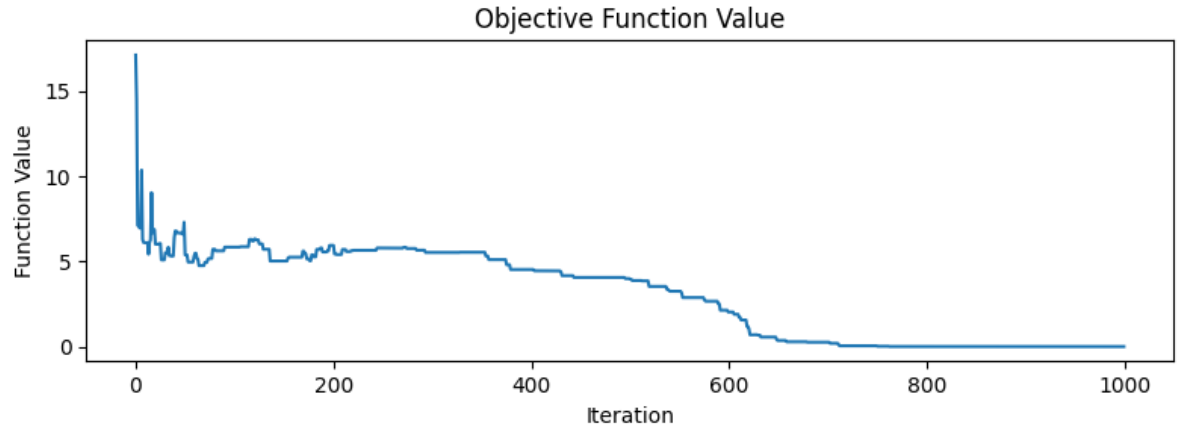
Exercise 3*

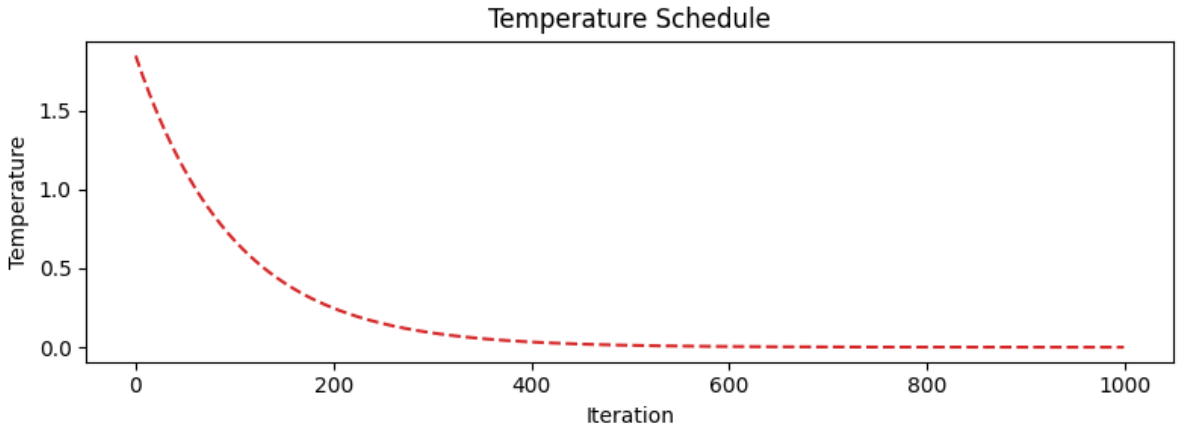
Implement Simulated Annealing. Set the initial temperature such that the initial acceptance ratio is 0.2 and the annealing plan to exponential with coolring rate $\gamma = 0.99$. Apply the algorithm to the Rosenbrock function and plot the value of the function and the temperature throughout the iterations.

Solution:

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

def compute_initial_temperature(objective, bounds):
    dim = len(bounds)

    deltas = []

    for r in range(100): # restarts
        current_solution = np.array([np.random.uniform(b[0], b[1]) for b in bounds])
        current_value = objective(current_solution)

        for i in range(100):
            # Generate a new candidate solution
            new_solution = current_solution + np.random.normal(0, 0.1, size=dim) # Small
                perturbation

            # Ensure the new solution is within bounds
            new_solution = np.clip(new_solution, [b[0] for b in bounds], [b[1] for b in
                bounds])
            new_value = objective(new_solution)

            # store deltas
            delta = new_value - current_value
            deltas.append(delta)

    return -np.mean(deltas)/np.log(0.2)



def simulated_annealing(objective, bounds, initial_temp=1000, cooling_rate=0.99, max_iter
    =1000):
    """
    Simulated Annealing for continuous optimization.

    Parameters:
        objective (function): The objective function to minimize.
        bounds (list of tuples): Bounds for each dimension [(min, max), ...].
        initial_temp (float): Starting temperature.
        cooling_rate (float): Cooling rate (should be between 0 and 1).
        max_iter (int): Maximum number of iterations.

    Returns:
        best_solution (numpy array): The best solution found.
        best_value (float): The corresponding objective function value.
        history (list): Values of the objective function during iterations.
        temp_history (list): Temperature values during iterations.
    """
    # Initialize solution randomly within bounds
    dim = len(bounds)
    current_solution = np.array([np.random.uniform(b[0], b[1]) for b in bounds])
    current_value = objective(current_solution)

    best_solution, best_value = current_solution, current_value

    temp = initial_temp
```

```
    history = []
    temp_history = []

    for i in range(max_iter):
        # Store history
        history.append(current_value)
        temp_history.append(temp)

        # Generate a new candidate solution
        new_solution = current_solution + np.random.normal(0, 0.1, size=dim) # Small
            perturbation

        # Ensure the new solution is within bounds
        new_solution = np.clip(new_solution, [b[0] for b in bounds], [b[1] for b in bounds
            ])
        new_value = objective(new_solution)

        # Acceptance probability
        delta = new_value - current_value
        if delta < 0 or np.exp(-delta / temp) > np.random.rand():
            current_solution, current_value = new_solution, new_value

        # Update best solution found
        if current_value < best_value:
            best_solution, best_value = current_solution, current_value

        # Decrease temperature
        temp *= cooling_rate

    return best_solution, best_value, history, temp_history

# Rosenbrock function
def rosenbrock_function(x):
    return sum(100 * (x[i+1] - x[i]**2)**2 + (1 - x[i])**2 for i in range(len(x) - 1))

bounds = [(-2, 2), (-2, 2)] # 2D problem
initial_temperature = compute_initial_temperature(rosenbrock_function, bounds)
print("Initial temparature:", initial_temperature)
best_sol, best_val, history, temp_history = simulated_annealing(rosenbrock_function,
    bounds, initial_temp = initial_temperature)

# Plot function value and temperature over iterations
fig, axes = plt.subplots(2, 1, figsize=(8, 6))

axes[0].plot(history, color="tab:blue")
axes[0].set_xlabel("Iteration")
axes[0].set_ylabel("Function Value")
axes[0].set_title("Objective Function Value")

axes[1].plot(temp_history, color="tab:red", linestyle="dashed")
axes[1].set_xlabel("Iteration")
axes[1].set_ylabel("Temperature")
axes[1].set_title("Temperature Schedule")

plt.tight_layout()
plt.savefig("simann.png")

print(f"Best solution: {best_sol}")
print(f"Best value: {best_val}")
```


Objective Function Value
Function Value
15
10
5
0
0
200
400
600
800
1000
Iteration


Temperature Schedule
Temperature
1.5
1.0
0.5
0.0
0
200
400
600
800
1000
Iteration