LAB 2

Particle Swarm Optimization for Function Optimization

Application:

Optimising a Proportional-Integral-Derivative(PID) controller in a control system

Code:

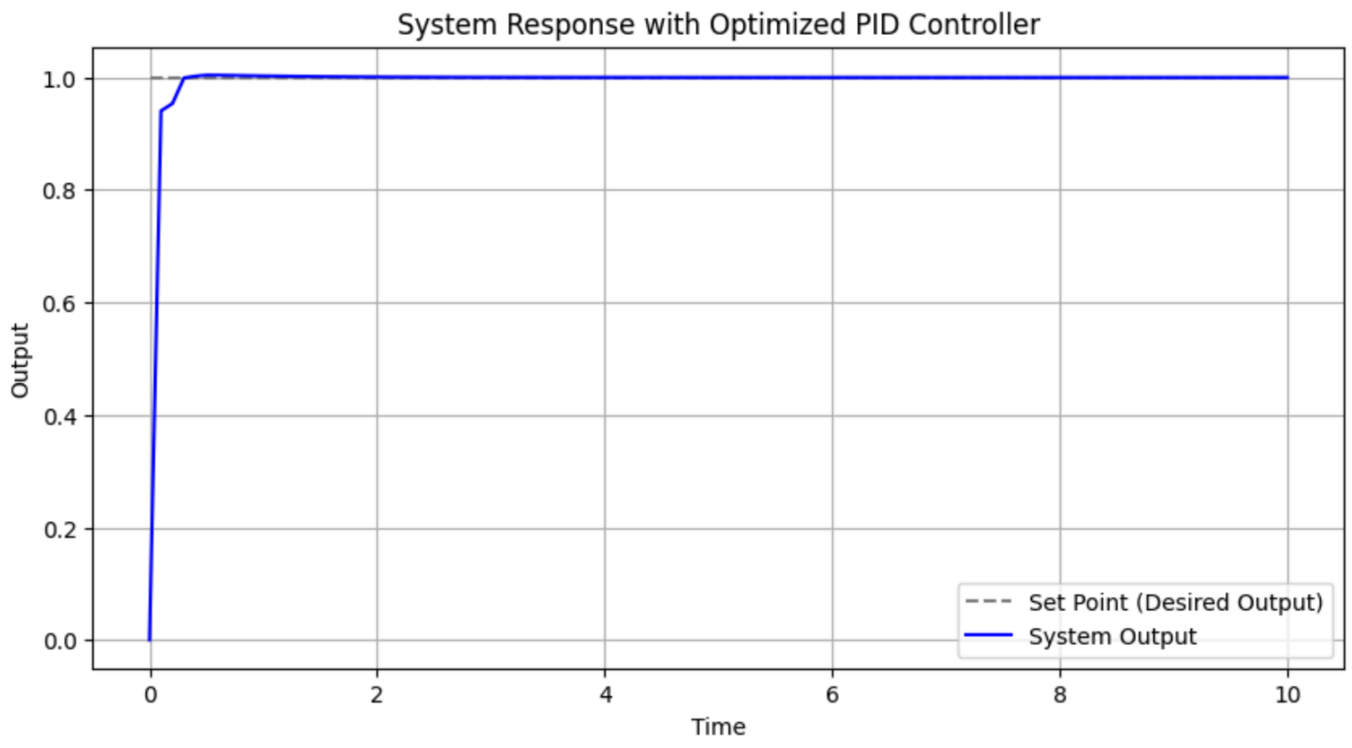
```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Simulation of the system
def system(y, t, u, Kp, Ki, Kd, prev_error, integral_error):
    error = u - y[0]  # Calculate the error
    integral_error += error * (t[1] - t[0])  # Integral of error
    derivative_error = (error - prev_error) / (t[1] - t[0])  # Derivative of error

    # PID control signal
    control_signal = Kp * error + Ki * integral_error + Kd * derivative_error

    # Simple system dynamics (first-order system)
    dydt = -y[0] + control_signal
    return [dydt], error, integral_error

# Fitness function for PSO
def fitness_function(pid_params):
    Kp, Ki, Kd = pid_params
    t = np.linspace(0, 10, 100)  # Time vector
    u = 1  # Desired step input
    y = [0]  # Initial system output
    prev_error = 0
    integral_error = 0
    total_error = 0  # Initialize error accumulator

    for i in range(len(t) - 1):
        dydt, error, integral_error = system(
            y[-1:], [t[i], t[i + 1]], u, Kp, Ki, Kd, prev_error, integral_error
        )
        y.append(y[-1] + dydt[0] * (t[i + 1] - t[i]))
        total_error += abs(error)  # Accumulate the absolute error
        prev_error = error

    return total_error  # Return total error as fitness

# User input for PSO parameters
print("Enter the parameters for Particle Swarm Optimization:")
num_particles = int(input("Number of particles (e.g., 30): "))
num_iterations = int(input("Number of iterations (e.g., 50): "))
inertia_weight = float(input("Inertia weight (e.g., 0.5): "))
cognitive_coeff = float(input("Cognitive coefficient (e.g., 1.5): "))
social_coeff = float(input("Social coefficient (e.g., 1.5): "))
bounds = [0, 10]  # Fixed range for PID parameters
```

```python
# Initialize particles
positions = np.random.uniform(bounds[0], bounds[1], (num_particles, 3))  # PID has 3
parameters
velocities = np.random.uniform(-1, 1, (num_particles, 3))
personal_best_positions = positions.copy()
personal_best_scores = np.array([fitness_function(pos) for pos in positions])
global_best_position = personal_best_positions[np.argmin(personal_best_scores)]
global_best_score = np.min(personal_best_scores)

# PSO Loop
for iteration in range(num_iterations):
    for i in range(num_particles):
        # Update velocities
        r1, r2 = np.random.rand(), np.random.rand()
        velocities[i] = (
            inertia_weight * velocities[i]
            + cognitive_coeff * r1 * (personal_best_positions[i] - positions[i])
            + social_coeff * r2 * (global_best_position - positions[i])
        )
        # Update positions
        positions[i] += velocities[i]
        # Clip positions to bounds
        positions[i] = np.clip(positions[i], bounds[0], bounds[1])

        # Evaluate fitness
        current_fitness = fitness_function(positions[i])
        if current_fitness < personal_best_scores[i]:
            personal_best_scores[i] = current_fitness
            personal_best_positions[i] = positions[i]

    # Update global best
    current_global_best_score = np.min(personal_best_scores)
    if current_global_best_score < global_best_score:
        global_best_score = current_global_best_score
        global_best_position = personal_best_positions[np.argmin(personal_best_scores)]

    print(f"Iteration {iteration + 1}, Best Fitness: {global_best_score:.4f}")

# Output the best solution
print(f"\nOptimized PID parameters (Kp, Ki, Kd): {global_best_position}")
print(f"Best fitness achieved: {global_best_score:.4f}")

# Plot system response with optimized parameters
Kp, Ki, Kd = global_best_position
t = np.linspace(0, 10, 100)
u = 1
y = [0]
prev_error = 0
integral_error = 0

for i in range(len(t) - 1):
    dydt, error, integral_error = system(
        y[-1:], [t[i], t[i + 1]], u, Kp, Ki, Kd, prev_error, integral_error
    )
    y.append(y[-1] + dydt[0] * (t[i + 1] - t[i]))
    prev_error = error
```

```
plt.figure(figsize=(10, 5))
plt.plot(t, [u] * len(t), label="Set Point (Desired Output)", linestyle="--",
color="gray")
plt.plot(t, y, label="System Output", color="blue")
plt.title("System Response with Optimized PID Controller")
plt.xlabel("Time")
plt.ylabel("Output")
plt.legend()
plt.grid(True)
plt.show()
```

Output:

```
Enter the parameters for Particle Swarm Optimization:
Number of particles (e.g., 30): 30
Number of iterations (e.g., 50): 20
Inertia weight (e.g., 0.5): 0.2
Cognitive coefficient (e.g., 1.5): 1
Social coefficient (e.g., 1.5): 1
Iteration 1, Best Fitness: 1.9928
Iteration 2, Best Fitness: 1.9373
Iteration 3, Best Fitness: 1.6428
Iteration 4, Best Fitness: 1.5630
Iteration 5, Best Fitness: 1.4201
Iteration 6, Best Fitness: 1.3285
Iteration 7, Best Fitness: 1.2876
Iteration 8, Best Fitness: 1.2346
Iteration 9, Best Fitness: 1.2164
Iteration 10, Best Fitness: 1.2009
Iteration 11, Best Fitness: 1.1944
Iteration 12, Best Fitness: 1.1817
Iteration 13, Best Fitness: 1.1783
Iteration 14, Best Fitness: 1.1731
Iteration 15, Best Fitness: 1.1670
Iteration 16, Best Fitness: 1.1645
Iteration 17, Best Fitness: 1.1586
Iteration 18, Best Fitness: 1.1565
Iteration 19, Best Fitness: 1.1540
Iteration 20, Best Fitness: 1.1515

Optimized PID parameters (Kp, Ki, Kd): [7.94470852 9.32540143 0.04290821]
Best fitness achieved: 1.1515
```


System Response with Optimized PID Controller
Output
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
Time
Set Point (Desired Output)
System Output