Experiment-3.2

Aim of the Experiment:

Study and implement a Derivative-free Optimization on complex functions.

Theory:

Derivative-free Optimization (DFO) is a branch of optimization techniques that aim to find the minimum or maximum of a function without using derivative information. In traditional optimization methods, derivatives of the objective function (gradients or higher-order derivatives) are used to guide the search for the optimum. However, in many real-world scenarios, obtaining derivatives can be computationally expensive, impractical, or even impossible due to the lack of analytical expressions for the objective function.

Derivative-free optimization techniques are particularly useful in scenarios such as:

a) Black-box Functions: When the objective function is a "black-box," meaning that its mathematical form or structure is unknown, making it impossible to compute derivatives.

b) Noisy or Discontinuous Functions: Functions that are noisy or discontinuous can pose challenges for traditional optimization algorithms that rely on derivative information.

c) Expensive Function Evaluations: In some cases, evaluating the objective function can be computationally expensive, making it impractical to compute derivatives.

Methods of Derivative-free Optimization Techniques:

A) Pattern Search: This method explores the search space by iteratively moving from one point to another based on a set of predefined patterns. At each iteration, the algorithm evaluates the objective function at several points around the current best point and updates the search direction based on the improvement in the function value.

B) Genetic Algorithms: Inspired by the process of natural selection, genetic algorithms maintain a population of candidate solutions and iteratively evolve them by applying selection, crossover, and mutation operators. The fittest individuals in the population, i.e., those with the best objective function values, are more likely to produce offspring in subsequent generations.

C) Simulated Annealing: Simulated annealing is a probabilistic technique that mimics the process of annealing in metallurgy. It starts with an initial solution and iteratively explores the search space by probabilistically accepting worse solutions based on a temperature parameter that gradually decreases over time.

D) Particle Swarm Optimization (PSO): PSO is inspired by the social behavior of bird flocking or fish schooling. It maintains a population of particles (candidate solutions) that move through the search space. Each particle adjusts its position based on its own experience and the experience of its neighbors, aiming to converge towards the optimum.

E) Bayesian Optimization: Bayesian optimization builds a probabilistic surrogate model of the objective function based on the observed function evaluations. It uses this model to decide where to evaluate the function next, balancing exploration (searching in regions with high uncertainty) and exploitation (searching in regions with expected high performance).

Derivative-free optimization techniques are valuable in various fields such as engineering design, finance, machine learning, and computer simulations where the objective functions may be complex, noisy, or expensive to evaluate. However, they typically require more function evaluations compared to derivative-based methods, making them less efficient in certain scenarios.

Code for Experiment :

```
fprintf("Particle Swarm Optimization (PSO) Algorithm: \n")

% Define the Objective function (Rastrigin function)
fun = @(x) sum(x.^2 - 10 * cos(2 * pi * x) + 10, 2);

% Initialization
nDims = 2; % Number of dimensions
lb = -5.12 * ones(1, nDims); % Lower bound for each dimension
ub = 5.12 * ones(1, nDims); % Upper bound for each dimension
nParticles = 20; % Number of particles in the swarm
maxIter = 50; % Maximum number of iterations
fprintf("Number of dimension: %d\n",nDims);
fprintf("Number of particles in Swarm: %d\n",nParticles);
fprintf("Maximum number of Iterations: %d\n\n",maxIter);

% Initialize the positions and velocities of particles
positions = repmat(lb, nParticles, 1) + rand(nParticles, nDims) .* repmat(ub - lb,
            nParticles, 1);
velocities = zeros(nParticles, nDims);
```

```
% Initialize personal best positions and objective function values
pBestPositions = positions;
pBestValues = inf(nParticles, 1);

% Initialize global best position and objective function value
gBestValue = inf;
gBestPosition = zeros(1, nDims);

% Main Optimization loop
for iter = 1:maxIter
	% Evaluate objective function values for each particle
	values = fun(positions);

	% Update personal best positions and values
	updateIndices = values < pBestValues;
	updateIndices = find(updateIndices);
	pBestPositions(updateIndices, :) = positions(updateIndices, :);
	pBestValues(updateIndices) = values(updateIndices);

	% Update global best position and value
	[minValue, minIndex] = min(pBestValues);
	if minValue < gBestValue
		gBestValue = minValue;
		gBestPosition = pBestPositions(minIndex, :);
	end

	% Update velocities and positions
	inertiaWeight = 0.7;
	cognitiveWeight = 1.5;
	socialWeight = 1.5;

	r1 = rand(nParticles, nDims);
	r2 = rand(nParticles, nDims);

	velocities = inertiaWeight * velocities + cognitiveWeight * r1 .* (pBestPositions -
positions) + socialWeight * r2 .* (repmat(gBestPosition, nParticles, 1) - positions);
	positions = positions + velocities;

	% Clip positions to within bounds
	positions = max(positions, lb);
	positions = min(positions, ub);

	% Display iteration information
	disp(['Iteration ', num2str(iter), ', Best Value: ', num2str(gBestValue)]);
end

% Display final result
fprintf('\nOptimal solution:');
disp(gBestPosition);
disp('Optimal function value:');
disp(gBestValue);
```

Result/Output :

Command Window

```
>> SC_Experiment8_ASHISH_23MAI10008
Particle Swarm Optimization (PSO) Algorithm:
Number of dimension: 2
Number of particles in Swarm: 20
Maximum number of Iterations: 50

Iteration 1, Best Value: 13.1889
Iteration 2, Best Value: 12.7243
Iteration 3, Best Value: 12.5209
Iteration 4, Best Value: 12.5209
Iteration 5, Best Value: 12.5209
Iteration 6, Best Value: 10.5875
Iteration 7, Best Value: 10.0094
Iteration 8, Best Value: 10.0094
Iteration 9, Best Value: 10.0094
Iteration 10, Best Value: 10.0094
```

```
Iteration 11, Best Value: 9.8225
Iteration 12, Best Value: 9.8225
Iteration 13, Best Value: 7.1335
Iteration 14, Best Value: 7.1335
Iteration 15, Best Value: 5.5771
Iteration 16, Best Value: 5.5771
Iteration 17, Best Value: 5.5771
Iteration 18, Best Value: 5.5771
Iteration 19, Best Value: 5.5771
Iteration 20, Best Value: 0.6192
Iteration 21, Best Value: 0.6192
Iteration 22, Best Value: 0.6192
Iteration 23, Best Value: 0.6192
Iteration 24, Best Value: 0.6192
Iteration 25, Best Value: 0.6192
Iteration 26, Best Value: 0.6192
Iteration 27, Best Value: 0.6192
Iteration 28, Best Value: 0.6192
Iteration 29, Best Value: 0.6192
Iteration 30, Best Value: 0.6192
Iteration 31, Best Value: 0.5975
Iteration 32, Best Value: 0.5975
Iteration 33, Best Value: 0.5975
Iteration 34, Best Value: 0.5975
Iteration 35, Best Value: 0.5975
```

```
Iteration 36, Best Value: 0.36671
Iteration 37, Best Value: 0.36671
Iteration 38, Best Value: 0.36671
Iteration 39, Best Value: 0.18964
Iteration 40, Best Value: 0.18964
Iteration 41, Best Value: 0.13155
Iteration 42, Best Value: 0.13155
Iteration 43, Best Value: 0.11258
Iteration 44, Best Value: 0.01807
Iteration 45, Best Value: 0.01807
Iteration 46, Best Value: 0.01807
Iteration 47, Best Value: 0.01807
Iteration 48, Best Value: 0.01807
Iteration 49, Best Value: 0.01807
Iteration 50, Best Value: 0.01807

Optimal solution:    0.0076   -0.0058

Optimal function value:
    0.0181

fx >>
```

Learning outcomes :

1. Learnt about the Derivation-free Optimization algorithms.
2. Learnt about Genetic algorithm and Simulated Annealing.
3. Learnt about the concept of Particle Swarm Optimization(PSO).
4. Learnt about how to apply PSO on complex functions.
5. Learnt about the use of Derivation-free Optimization algorithms.