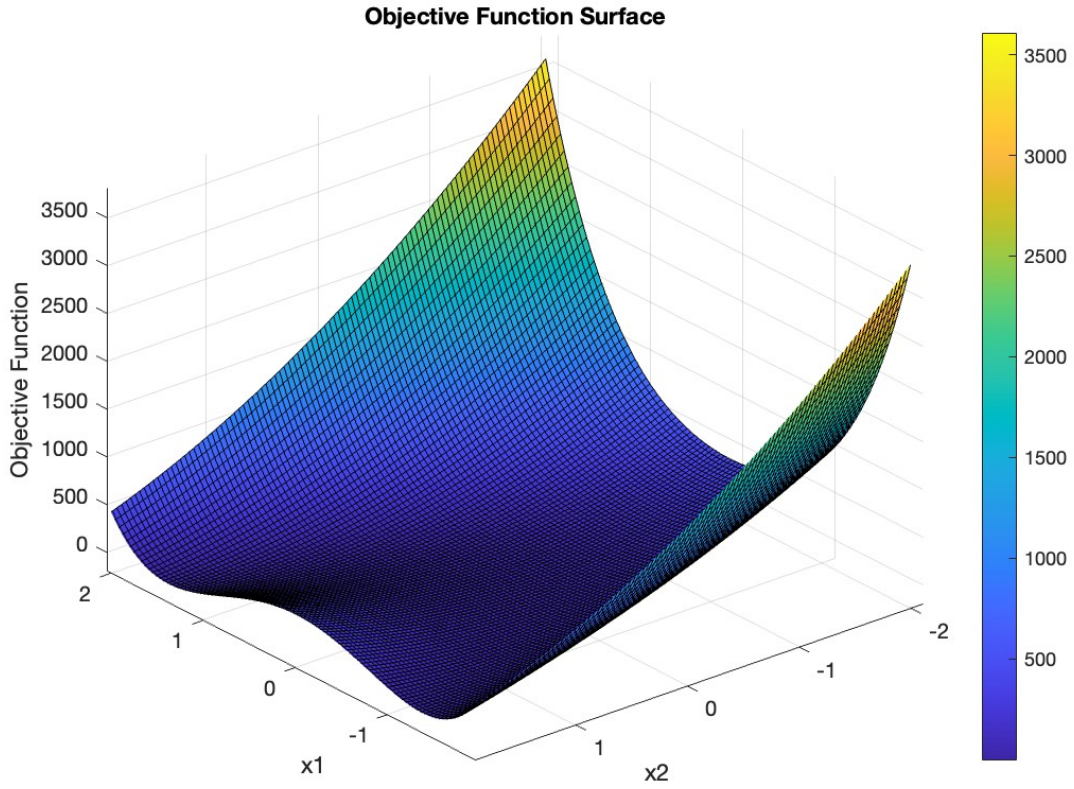
Solution

Step 1: Define the Objective Function

1. Based on the given function, plot the function in three dimensions within the defined domain (-2, 2) using the following steps:

```
% Define Objective Function
ObjectiveFunction = @(x) 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;

% Define the range of the function
x1 = linspace(-2, 2, 100);
x2 = linspace(-2, 2, 100);
[X1, X2] = meshgrid(x1, x2);

% Compute the function value
Y = zeros(size(X1));
for i = 1:numel(X1)
    Y(i) = ObjectiveFunction([X1(i), X2(i)]);
end

% Plot the graphic
figure;
surf(X1, X2, Y);
xlabel('x1');
ylabel('x2');
zlabel('Objective Function');
title('Objective Function Surface');
colorbar;
```

2. Analysis of the Graph



By observing the obtained graph, we can roughly determine the range of function values. The minimum value of the function is in the vicinity of the deep blue region. This observation will help in verifying the correctness of the solution obtained through the genetic algorithm.

Step 2: Genetic Algorithm Parameter Setup

1. Variable Range and Constraints

   In this example, the upper bound of the variables is 2, and the lower bound is -2. There are no constraints.

```
nvars = 2; % Number of variables
LB = [-2 -2]; % Lower bound
UB = [2 2]; % Upper bound
```

2. Set Genetic Algorithm Parameters
   - Use the `gaoptimset` function to create an options structure called `options`.
   - Set the population size `PopulationSize` to 50.
   - Set the number of generations `Generations` to 300.
   - Set the display option `Display` to 'iter', which prints output information at each iteration.

```
options = gaoptimset('PopulationSize',50,'Generations',300,'Display','iter');
```

Step 3: Invoke the Genetic Algorithm

- Use the `ga` function to invoke the genetic algorithm and store the results in the variables `x` and `fval`.
- `ObjectiveFunction` is the objective function.
- `nvars` is the number of variables.
- `LB` and `UB` are the lower and upper bounds of the variables.
- `options` is the set of options for the genetic algorithm.
- [] is used for constraints, as there are no constraints in this function.

```
[x, fval] = ga(ObjectiveFunction, nvars, [], [], [], [], LB, UB, [], options);
```

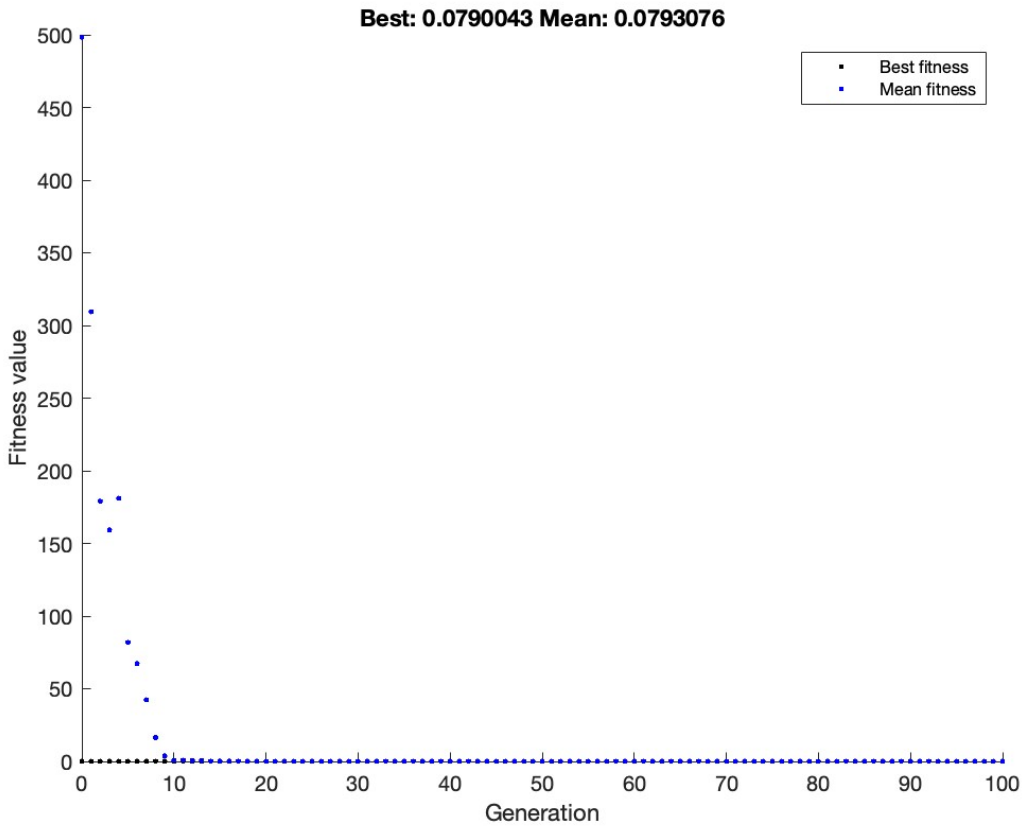
Step 4: Compute and Analyze the Results

1. Running the `GeneticAlgorithm` code, the output after 100 iterations is as follows:

```
>> GeneticAlgorithm

Single objective optimization:
2 Variable(s)

Options:
CreationFcn:       @gacreationuniform
CrossoverFcn:      @crossoverscattered
SelectionFcn:      @selectionstochunif
MutationFcn:       @mutationadaptfeasible

                            Best       Max        Stall
Generation  Func-count        f(x)     Constraint  Generations
    1           2500    0.00370485            0      0
    2           4950    0.00206078            0      0
    3           7400    0.00148908            0      0
    4           9850   0.000966505            0      0
    5          13005   0.000370361            0      0
Optimization terminated: average change in the fitness value less than options.FunctionTolerance
 and constraint violation is less than options.ConstraintTolerance.
```

   As the number of iterations increases, the optimal value of the function gradually approaches zero, and the corresponding input values converge to (1, 1).

2. By running the `OptimizeLive` live file:



By setting the relevant parameters, various images of the genetic algorithm can be generated. The following graph shows the change in fitness value with increasing iterations.

After 10 iterations, the fitness value starts to approach 0. With the increasing number of iterations, it gradually approaches the optimal solution, which is the minimum value of the function.