- Cuckoo Search optimization

```
- import numpy as np
-
- # Objective function (to minimize)
- def objective_function(x):
-     return x**2  # Example: f(x) = x^2
-
- # Generate new solutions using Levy flights
- def levy_flight(current_position, alpha=0.01, lambda_param=1.5):
-     u = np.random.normal(0, 1)
-     v = np.random.normal(0, 1)
-     step = u / (abs(v) ** (1 / lambda_param))  # Levy flight step
-     new_position = current_position + alpha * step
-     return new_position
-
- # Initialize parameters
- def initialize_population(n_nests, bounds):
-     return np.random.uniform(bounds[0], bounds[1], n_nests)
-
- # Evaluate fitness of all nests
- def evaluate_fitness(population):
-     return np.array([objective_function(x) for x in population])
-
- # Main Cuckoo Search function
- def cuckoo_search(n_nests=10, pa=0.25, alpha=0.01, bounds=(-10,
  10), max_iter=100):
-     # Step 1: Initialize population (nests)
-     nests = initialize_population(n_nests, bounds)
-     best_solution = None
-     best_fitness = float("inf")
-
-     for iteration in range(max_iter):
-         # Step 2: Generate new solutions using Levy flights
-         new_nests = np.array([levy_flight(x, alpha) for x in
  nests])
-
-         # Step 3: Evaluate fitness of all nests
-         fitness = evaluate_fitness(nests)
-         new_fitness = evaluate_fitness(new_nests)
-
-         # Step 4: Keep the better solutions
-         for i in range(n_nests):
-             if new_fitness[i] < fitness[i]:
-                 nests[i] = new_nests[i]
-                 fitness[i] = new_fitness[i]
-
-         # Step 5: Abandon worst nests and replace with random
  solutions
```

```
-           n_abandon = int(pa * n_nests)  # Fraction of nests to
    abandon
-           worst_indices = fitness.argsort()[-n_abandon:]  # Get
    worst nests
-           nests[worst_indices] = initialize_population(n_abandon,
    bounds)
-
-           # Update the best solution found so far
-           min_index = fitness.argmin()
-           if fitness[min_index] < best_fitness:
-               best_fitness = fitness[min_index]
-               best_solution = nests[min_index]
-
-           # Print progress
-           print(f"Iteration {iteration+1}: Best fitness =
    {best_fitness}")
-
-       return best_solution, best_fitness
-
-   # Run the Cuckoo Search algorithm
-   best_solution, best_fitness = cuckoo_search()
-   print(f"\nBest solution: {best_solution}")
-   print(f"Best fitness: {best_fitness}")
```

Output :

```
Iteration 1: Best fitness = 0.002538440438936984
Iteration 2: Best fitness = 6.0102677484278274e-05
Iteration 3: Best fitness = 6.0102677484278274e-05
Iteration 4: Best fitness = 6.0102677484278274e-05
Iteration 5: Best fitness = 6.0102677484278274e-05
Iteration 6: Best fitness = 8.70216287020563e-06
Iteration 7: Best fitness = 8.70216287020563e-06
Iteration 8: Best fitness = 8.70216287020563e-06
Iteration 9: Best fitness = 8.70216287020563e-06
Iteration 10: Best fitness = 4.776848058463835e-07
Iteration 11: Best fitness = 4.776848058463835e-07
Iteration 12: Best fitness = 4.776848058463835e-07
Iteration 13: Best fitness = 4.776848058463835e-07
Iteration 14: Best fitness = 4.776848058463835e-07
Iteration 15: Best fitness = 4.776848058463835e-07
Iteration 16: Best fitness = 4.776848058463835e-07
Iteration 17: Best fitness = 4.776848058463835e-07
Iteration 18: Best fitness = 4.776848058463835e-07
Iteration 19: Best fitness = 4.776848058463835e-07
Iteration 20: Best fitness = 4.776848058463835e-07
Iteration 21: Best fitness = 4.776848058463835e-07
Iteration 22: Best fitness = 4.776848058463835e-07
Iteration 23: Best fitness = 4.776848058463835e-07
Iteration 24: Best fitness = 4.776848058463835e-07
Iteration 25: Best fitness = 4.776848058463835e-07
Iteration 26: Best fitness = 4.776848058463835e-07
Iteration 27: Best fitness = 4.776848058463835e-07
Iteration 28: Best fitness = 4.776848058463835e-07
```

```
Iteration 29: Best fitness = 4.776848058463835e-07
Iteration 30: Best fitness = 4.776848058463835e-07
Iteration 31: Best fitness = 4.776848058463835e-07
Iteration 32: Best fitness = 4.776848058463835e-07
Iteration 33: Best fitness = 4.776848058463835e-07
Iteration 34: Best fitness = 4.776848058463835e-07
Iteration 35: Best fitness = 4.776848058463835e-07
Iteration 36: Best fitness = 4.776848058463835e-07
Iteration 37: Best fitness = 4.776848058463835e-07
Iteration 38: Best fitness = 4.776848058463835e-07
Iteration 39: Best fitness = 4.776848058463835e-07
Iteration 40: Best fitness = 4.776848058463835e-07
Iteration 41: Best fitness = 4.776848058463835e-07
Iteration 42: Best fitness = 4.776848058463835e-07
Iteration 43: Best fitness = 4.776848058463835e-07
Iteration 44: Best fitness = 4.776848058463835e-07
Iteration 45: Best fitness = 4.776848058463835e-07
Iteration 46: Best fitness = 4.776848058463835e-07
Iteration 47: Best fitness = 4.776848058463835e-07
Iteration 48: Best fitness = 4.776848058463835e-07
Iteration 49: Best fitness = 4.776848058463835e-07
Iteration 50: Best fitness = 4.776848058463835e-07
Iteration 51: Best fitness = 4.776848058463835e-07
Iteration 52: Best fitness = 4.776848058463835e-07
Iteration 53: Best fitness = 4.776848058463835e-07
Iteration 54: Best fitness = 4.776848058463835e-07
Iteration 55: Best fitness = 4.776848058463835e-07
Iteration 56: Best fitness = 4.776848058463835e-07
Iteration 57: Best fitness = 4.776848058463835e-07
Iteration 58: Best fitness = 4.776848058463835e-07
Iteration 59: Best fitness = 4.776848058463835e-07
Iteration 60: Best fitness = 4.776848058463835e-07
Iteration 61: Best fitness = 9.730270617136578e-08
Iteration 62: Best fitness = 9.730270617136578e-08
Iteration 63: Best fitness = 9.730270617136578e-08
Iteration 64: Best fitness = 9.730270617136578e-08
Iteration 65: Best fitness = 9.730270617136578e-08
Iteration 66: Best fitness = 4.496689973467044e-08
Iteration 67: Best fitness = 4.496689973467044e-08
Iteration 68: Best fitness = 4.496689973467044e-08
Iteration 69: Best fitness = 4.496689973467044e-08
Iteration 70: Best fitness = 4.496689973467044e-08
Iteration 71: Best fitness = 4.496689973467044e-08
Iteration 72: Best fitness = 4.496689973467044e-08
Iteration 73: Best fitness = 4.496689973467044e-08
Iteration 74: Best fitness = 4.496689973467044e-08
Iteration 75: Best fitness = 4.496689973467044e-08
Iteration 76: Best fitness = 4.496689973467044e-08
Iteration 77: Best fitness = 4.496689973467044e-08
Iteration 78: Best fitness = 4.496689973467044e-08
Iteration 79: Best fitness = 4.496689973467044e-08
Iteration 80: Best fitness = 1.1273215058429191e-13
Iteration 81: Best fitness = 1.1273215058429191e-13
Iteration 82: Best fitness = 1.1273215058429191e-13
Iteration 83: Best fitness = 1.1273215058429191e-13
Iteration 84: Best fitness = 1.1273215058429191e-13
Iteration 85: Best fitness = 1.1273215058429191e-13
Iteration 86: Best fitness = 1.1273215058429191e-13
```

```
Iteration 87: Best fitness = 1.1273215058429191e-13
Iteration 88: Best fitness = 1.1273215058429191e-13
Iteration 89: Best fitness = 1.1273215058429191e-13
Iteration 90: Best fitness = 1.1273215058429191e-13
Iteration 91: Best fitness = 1.1273215058429191e-13
Iteration 92: Best fitness = 1.1273215058429191e-13
Iteration 93: Best fitness = 1.1273215058429191e-13
Iteration 94: Best fitness = 1.1273215058429191e-13
Iteration 95: Best fitness = 1.1273215058429191e-13
Iteration 96: Best fitness = 1.1273215058429191e-13
Iteration 97: Best fitness = 1.1273215058429191e-13
Iteration 98: Best fitness = 1.1273215058429191e-13
Iteration 99: Best fitness = 1.1273215058429191e-13
Iteration 100: Best fitness = 1.1273215058429191e-13

Best solution: -3.3575608793332684e-07
      Best fitness: 1.1273215058429191e-13
```