Lab 10: Genetic Algorithms and Differential Evolution

Overview

This lab introduces two population-based optimization methods:

- genetic algorithms (GA);
- differential evolution (DE).

Both methods work with a population of candidate solutions. This is different from classical local search, which usually keeps only one current solution at a time.

This lab is also preparation for Assignment 5. In Assignment 5, you need to optimize CEC 2022 benchmark functions in 20 dimensions. The code in this lab already uses the same basic setting: a solution is a real-valued vector, and the objective is a function that must be minimized.

Goals

By the end of this lab, you should understand:

- what a population-based optimization algorithm is;
- how a real-valued vector represents one candidate solution;
- how a genetic algorithm uses selection, crossover, mutation, and elitism;
- how differential evolution uses vector differences to create new candidates;
- how these methods can be adapted to the CEC 2022 functions from Assignment 5.

Python Source Files

The Python source files in this lab assignment are:

File	Purpose
`continuous_functions.py`	Defines Sphere, Rastrigin, and Ackley functions.
`ga_continuous.py`	Implements a real-valued genetic algorithm.
`de_continuous.py`	Implements differential evolution.
`compare_ga_de.py`	Runs GA and DE on several functions and compares results.

If the files are in the same directory, run them using:

```
python3 ga_continuous.py
python3 de_continuous.py
python3 compare_ga_de.py
```

Common Optimization Setting

In this lab, every solution is a vector:

$$x = (x_1, x_2, \ldots, x_n).$$

For example, in 20 dimensions:

$$x = (x_1, x_2, \ldots, x_{20}).$$

The goal is to minimize an objective function:

$$f(x).$$

A smaller value means a better solution.
The code uses three test functions:

- Sphere;
- Rastrigin;
- Ackley.

The global minimum of these functions is at or near the zero vector:

$$(0, 0, \ldots, 0),$$

where the objective value is approximately 0.

1 Algorithm 1: Genetic Algorithm

A genetic algorithm keeps a population of candidate solutions. Each candidate solution is called an individual or chromosome.

For continuous optimization, an individual is a real-valued vector:

$$[2.1, -0.4, 5.0, \ldots].$$

The algorithm repeats the following steps:

1. Generate an initial random population.
2. Evaluate every individual using the objective function.
3. Select better individuals as parents.
4. Combine parents using crossover.
5. Randomly modify children using mutation.
6. Keep the best solution found so far.

Selection

The code uses tournament selection. Several individuals are chosen randomly, and the best among them becomes a parent:

```
def tournament_selection(population, fitnesses, tournament_size,
    rng):
    candidates = rng.integers(0, len(population), size=
        tournament_size)
    winner = candidates[np.argmin(fitnesses[candidates])]
    return population[winner].copy()
```

Since we minimize, the winner is the candidate with the smallest objective value.

Crossover

The code uses uniform crossover. For each coordinate, the child inherits the coordinate from one of the two parents:

```
def uniform_crossover(parent1, parent2, rng):
    mask = rng.random(parent1.size) < 0.5
    child1 = np.where(mask, parent1, parent2)
    child2 = np.where(mask, parent2, parent1)
    return child1, child2
```

Mutation

Mutation adds small Gaussian noise to some coordinates. After mutation, the vector is clipped to stay inside the bounds:

```
def gaussian_mutation(individual, lower, upper, mutation_rate,
    mutation_sigma, rng):
    mutation_mask = rng.random(individual.size) < mutation_rate
    individual = individual.copy()
    individual[mutation_mask] += rng.normal(0, mutation_sigma,
        size=np.sum(mutation_mask))
    return np.clip(individual, lower, upper)
```

Code to Run

```
python3 ga_continuous.py
```

The script runs the genetic algorithm on the 20-dimensional Rastrigin function.

Questions

After running the code, answer:

a) What is one individual in this implementation?

b) What does tournament selection do?

c) What does uniform crossover do?

d) Why do we clip the mutated vector to the lower and upper bounds?

e) What is the best value found by the algorithm?

Key thing to notice: GA searches using a population. It does not improve only one solution; it repeatedly selects, recombines, and mutates many candidate solutions.

2 Algorithm 2: Differential Evolution

Differential Evolution is also a population-based algorithm, but it creates new candidates in a different way.

The central DE move is:

$$\text{mutant} = a + F(b - c),$$

where a, b, and c are different individuals from the population.

The vector difference $b - c$ gives a direction and step size. The parameter F controls how large this step is.

Mutation and Crossover in DE

In the code, for each current individual, we pick three other individuals:

```
a_idx, b_idx, c_idx = rng.choice(candidates, size=3, replace=False
    )
a = population[a_idx]
b = population[b_idx]
c = population[c_idx]

mutant = a + F * (b - c)
mutant = np.clip(mutant, lower, upper)
```

Then we combine the mutant with the current individual using crossover:

```
crossover_mask = rng.random(dimensions) < CR
forced_dimension = rng.integers(0, dimensions)
crossover_mask[forced_dimension] = True

trial = np.where(crossover_mask, mutant, population[i])
```

The parameter CR is the crossover rate. It controls how many coordinates are copied from the mutant vector.

Selection in DE

DE uses a simple replacement rule:

```
if trial_value <= fitnesses[i]:
    population[i] = trial
    fitnesses[i] = trial_value
```

So the trial vector replaces the current vector only if it is at least as good.

Code to Run

```
python3 de_continuous.py
```

The script runs differential evolution on the 20-dimensional Rastrigin function.

Questions

After running the code, answer:

a) What does the parameter F control?

b) What does the parameter CR control?

c) Why do we need three different vectors a, b, and c?

d) Why is the mutant vector clipped to the bounds?

e) What is the best value found by the algorithm?

Key thing to notice: DE creates new solutions by using differences between existing solutions. The population itself gives the algorithm information about useful search directions.

3 Comparison Experiment

Run:

```
python3 compare_ga_de.py
```

This script compares GA and DE on Sphere, Rastrigin, and Ackley in 20 dimensions. Fill in the following table using your output:

Function	GA best value	DE best value
Sphere		
Rastrigin		
Ackley		

Questions

Answer:

a) Which method performs better on Sphere?

b) Which method performs better on Rastrigin?

c) Which method performs better on Ackley?

d) Are the results the same if you change the random seed?

e) What happens if you increase the number of generations?

4 Parameter Experiment

Differential evolution is sensitive to the parameters F and CR.

Try at least three combinations:

F	CR	Best value
0.3	0.2	
0.7	0.9	
0.9	0.2	

For the genetic algorithm, try changing:

- mutation rate;
- mutation sigma;
- population size;
- number of generations.

5 Connection to Assignment 5

Assignment 5 uses the CEC 2022 functions from the `opfunu` package. The optimization idea is the same as in this lab.

In this lab:

```
objective = FUNCTIONS["rastrigin"]
lower, upper = bounds_for("rastrigin", dimensions)
result = differential_evolution(objective, lower, upper)
```

For Assignment 5, the same pattern becomes:

```
import numpy as np
from opfunu.cec_based import cec2022
from de_continuous import differential_evolution

func = cec2022.F12022(ndim=20)
objective = lambda x: func.evaluate(x)
lower = np.array(func.lb)
upper = np.array(func.ub)

result = differential_evolution(objective, lower, upper)
print(result["value"])
print(result["solution"])
```

For Assignment 5, you must run all 12 functions:

```
functions = [
    cec2022.F12022, cec2022.F22022, cec2022.F32022,
    cec2022.F42022, cec2022.F52022, cec2022.F62022,
    cec2022.F72022, cec2022.F82022, cec2022.F92022,
    cec2022.F102022, cec2022.F112022, cec2022.F122022
]
```

Important: Assignment 5 does not allow you to use already built optimizers from other packages. You may use benchmark functions from `opfunu`, but the optimization algorithm itself should be your own implementation.

Final Summary

Method	Main idea	Important parameters
GA	Select, recombine, and mutate individuals	population size, mutation rate, mutation sigma
DE	Create candidates using vector differences	population size, F, CR

The main lesson is:

> Population-based algorithms search with many candidate solutions at once. This makes them useful for difficult continuous optimization problems, especially when the function has many local minima.