Pseudo code:

Input:

N ← Population size

P ← Number of antibodies to select

C ← Clone factor (number of clones per antibody)

MutationRate ← Probability of mutation

MaxIterations ← Maximum number of iterations

Output:

Best antibody (solution) found

Begin

1. Initialize a population of N random antibodies

2. For each antibody, evaluate its fitness using the fitness function

3. For iteration = 1 to MaxIterations do:

a. Select the top P antibodies with highest fitness

b. Clone each selected antibody C times to form the clone population

c. For each clone in the clone population:

i. Mutate the clone with probability MutationRate

d. Evaluate the fitness of all mutated clones

e. Combine the original population and the mutated clones

f. Select the best N antibodies from the combined set to form the new population

4. End For

5. Return the antibody with the highest fitness as the best solution

End

**1. Import Required Library**

python

CopyEdit

import numpy as np

* numpy is imported as np because we are dealing with arrays and numerical operations (e.g., random numbers, mutations, fitness evaluation).

**2. Define Problem-Specific Parameters**

python

CopyEdit

POP\_SIZE = 10

CLONE\_FACTOR = 3

MUTATION\_RATE = 0.2

ITERATIONS = 50

| **Parameter** | **Meaning** |
| --- | --- |
| **POP\_SIZE** | Number of antibodies (solutions) initially generated. Here, 10 antibodies. |
| **CLONE\_FACTOR** | How many copies (clones) each selected antibody should produce. |
| **MUTATION\_RATE** | Probability that each gene (element) in a clone will mutate. |
| **ITERATIONS** | How many times the optimization (selection, cloning, mutation) repeats. |

**3. Fitness Function (Affinity)**

python

CopyEdit

def fitness\_function(x):

return -np.sum((x - 0.5) \*\* 2)

* This function measures **how good** a candidate antibody is.
* **Goal:** Minimize the distance from 0.5 in all dimensions.
* x is an antibody (an array of real numbers).
* We calculate the sum of squares of the difference from 0.5.
* We **negate** the value because in CLONALG, we **maximize affinity** (higher fitness = better antibody).

**4. Initialize Population**

python

CopyEdit

def initialize\_population(size, dim=5):

return np.random.rand(size, dim)

* Generates size number of antibodies, each with dim dimensions (default 5).
* Each dimension’s value is random between 0 and 1.

**5. Select Best Antibodies**

python

CopyEdit

def select\_best(population, fitness, num\_selected):

indices = np.argsort(fitness)[-num\_selected:]

return population[indices], fitness[indices]

* **Selects top antibodies** based on highest fitness.
* np.argsort(fitness) sorts fitness values in ascending order.
* [-num\_selected:] picks the best (highest fitness) antibodies.

**6. Clone Selected Antibodies**

python

CopyEdit

def clone\_population(selected, clone\_factor):

clones = np.repeat(selected, clone\_factor, axis=0)

return clones

* **Creates multiple copies** of each selected antibody.
* np.repeat() duplicates each antibody clone\_factor times.
* Cloning helps focus more on **promising solutions**.

**7. Mutate the Clones**

python

CopyEdit

def mutate\_population(clones, mutation\_rate):

mutations = (np.random.rand(\*clones.shape) < mutation\_rate) \* np.random.uniform(-0.1, 0.1, clones.shape)

return np.clip(clones + mutations, 0, 1)

* For each gene in each clone:
  + Randomly decide whether to mutate based on mutation\_rate.
  + If yes, apply a small change between -0.1 and 0.1.
* np.clip() keeps all gene values between [0, 1].

**8. Main Clonal Selection Algorithm**

python

CopyEdit

def clonal\_selection\_algorithm():

* This function **runs the entire algorithm**.

**Inside the Main Function:**

**Step 1: Initialization**

python

CopyEdit

population = initialize\_population(POP\_SIZE)

* Create initial random population.

**Step 2: Iterate Over Generations**

python

CopyEdit

for iteration in range(ITERATIONS):

* Run the optimization for a set number of iterations.

**Step 3: Evaluate Fitness**

python

CopyEdit

fitness = np.array([fitness\_function(ind) for ind in population])

* Calculate fitness for each antibody.

**Step 4: Selection**

python

CopyEdit

selected, selected\_fitness = select\_best(population, fitness, POP\_SIZE // 2)

* Select the best 50% (top 5 if POP\_SIZE=10) antibodies.

**Step 5: Cloning**

python

CopyEdit

clones = clone\_population(selected, CLONE\_FACTOR)

* Clone each selected antibody multiple times.

**Step 6: Hypermutation**

python

CopyEdit

mutated\_clones = mutate\_population(clones, MUTATION\_RATE)

* Mutate the clones.

**Step 7: Create New Population**

python

CopyEdit

new\_population = np.vstack((population, mutated\_clones))

new\_fitness = np.array([fitness\_function(ind) for ind in new\_population])

* Combine old population and mutated clones into one.
* Recalculate fitness.

**Step 8: Survival of the Fittest**

python

CopyEdit

population, \_ = select\_best(new\_population, new\_fitness, POP\_SIZE)

* Keep only the best POP\_SIZE antibodies for the next generation.

**Step 9: Track Progress**

python

CopyEdit

best\_fitness = max(new\_fitness)

print(f"Iteration {iteration + 1}: Best Fitness = {best\_fitness:.5f}")

* Print best fitness at each generation to observe improvement.

**Step 10: Return Final Best Solution**

python

CopyEdit

best\_solution = population[np.argmax([fitness\_function(ind) for ind in population])]

return best\_solution

* From the final population, pick the antibody with the highest fitness.

**9. Run the Algorithm**

python

CopyEdit

best\_solution = clonal\_selection\_algorithm()

print("Best Solution Found:", best\_solution)

* Finally, run the whole process and print the best solution found after all iterations.

**📊 Overall Flowchart:**

mathematica

CopyEdit

Initialize random population

↓

Evaluate fitness

↓

Select top antibodies (best fitness)

↓

Clone selected

↓

Hypermutate clones

↓

Combine original + mutated clones

↓

Select best individuals (next generation)

↓

Repeat for ITERATIONS

↓

Return best antibody found