ALGORITMO GENÉTICO (GA) EM PYTHON

Editado por: Dr. Arnaldo de Carvalho Junior – Março 2026

1. INTRODUÇÃO

O Algoritmo Genético (*genetic algorithm* - GA) foi proposto por John Holland em 1975. Desde a sua origem, encontrou muitas aplicações interessantes em vários ramos da ciência e da engenharia.

Os Algoritmos Genéticos são rápidos, fáceis de implementar e altamente personalizáveis. GAs são amplamente utilizados em vários campos, como engenharia, finanças, inteligência artificial e problemas de otimização, onde os métodos tradicionais podem ser impraticáveis ou computacionalmente caros.

A capacidade dos GAs de explorar vastos espaços de soluções e encontrar soluções quase ideais os tornam ferramentas valiosas para enfrentar desafios complexos de otimização.

2. GA EM PYTHON

Um código Python para algoritmos genéticos é implementado neste documento. Neste código Python para algoritmos genéticos, você pode implementar o algoritmo genético para seus requisitos específicos com pequenas modificações.

Execute o código do Anexo I utilizando a plataforma Google Colab clicando aqui.

O algoritmo começa com uma população inicial e, por meio de uma série de operações genéticas, como seleção, cruzamento (recombinação) e mutação, novas gerações são criadas.

Os indivíduos mais aptos, aqueles com melhores soluções, têm maior chance de serem selecionados para produzir descendentes, que herdam características de seus pais.

À medida que as gerações avançam, a população tende a evoluir em direção a melhores soluções, à medida que os indivíduos mais aptos dominam. Este processo

continua até que um critério de terminação, como atingir um número máximo de gerações ou alcançar uma solução satisfatória, seja atendido.

Exemplo de saída:

```
Melhor solução: [-3.072, -3.072, -3.072, -3.072]
Melhor aptidão: 37.748736
```

REFERÊNCIAS

Evolucionary Genius. Python code for Genetic Algorithm. Disponível em: https://evolutionarygenius.com/python-code-for-genetic-algorithms/. Acessado em Maio 13, 2024.

Anexo I

Segue o código Python simples e pronto para implementar para algoritmos genéticos.

```
# Algoritmo Genético do zero em Python
# Ref: https://evolutionarygenius.com/python-code-for-genetic-algorithms/
# Adaptado por: Dr. Arnaldo de Carvalho Junior – Março 2026

import random

# Parâmetros do Algoritmo Genético
POPULATION_SIZE = 100
GENERATION_COUNT = 50
CROSSOVER_RATE = 0.8
MUTATION_RATE = 0.1
CHROMOSOME_LENGTH = 4
LOWER_BOUND = -5.12
UPPER_BOUND = 5.12

# Gera um cronossomo randômico
def generate_chromosome():
  return [random.randint(0, 1) for _ in range(CHROMOSOME_LENGTH)]

# Evalia a aptidão de um indivíduo
def evaluate_fitness(chromosome):
  x = decode_chromosome(chromosome)
  #Here you can change the objective function value
  fitness_value = sum([gene**2 for gene in x])
  return fitness_value

# Decodificar cromossomo binário à representação de valor real
def decode_chromosome(chromosome):
  x = []
  for gene in chromosome:
    value = LOWER_BOUND + (UPPER_BOUND - LOWER_BOUND) * int("".join(map(str,
chromosome)), 2) / (2 ** CHROMOSOME_LENGTH - 1)
    x.append(value)
  return x

# Realizar seleção usando a seleção de torneios
def selection(population):
  tournament_size = 5
  selected_parents = []
  for _ in range(len(population)):
    tournament = random.sample(population, tournament_size)
    winner = min(tournament, key=lambda x: x[1])
    selected_parents.append(winner[0])
  return selected_parents
```

```
# Faça um crossover entre dois pais
def crossover(parent1, parent2):
  if random.random() < CROSSOVER_RATE:
    crossover_point = random.randint(1, CHROMOSOME_LENGTH - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2
  else:
    return parent1, parent2

# Faça a mutação em um individuo
def mutate(individual):
  mutated_individual = individual.copy()
  for i in range(CHROMOSOME_LENGTH):
    if random.random() < MUTATION_RATE:
      mutated_individual[i] = 1 - mutated_individual[i]
  return mutated_individual

# Gera uma população inicial
population = [(generate_chromosome(), 0) for _ in range(POPULATION_SIZE)]

# Laço principal do algorítmo genético
for _ in range(GENERATION_COUNT):
  # Avalie a aptidão de cada indivíduo na população
  population = [(chromosome, evaluate_fitness(chromosome)) for chromosome, _ in
population]

  # Seleciona pais para reprodução
  parents = selection(population)

  # Criar filhos através do crossover e mutação
  offspring = []
  for i in range(0, POPULATION_SIZE, 2):
    parent1 = parents[i]
    parent2 = parents[i + 1]
    child1, child2 = crossover(parent1, parent2)
    child1 = mutate(child1)
    child2 = mutate(child2)
    offspring.extend([child1, child2])

  # Substitua a população velha com os filhos
  population = [(chromosome, 0) for chromosome in offspring]

# Selecione o melhor indivídulo como a solução
best_individual = min(population, key=lambda x: x[1])[0]
decoded_solution = decode_chromosome(best_individual)
fitness_value = evaluate_fitness(best_individual)
# Imprimir a solução
print("Melhor solução:", decoded_solution)
print("Melhor aptidão:", fitness_value)
```