Prática 2 - Bioinspirados

Rodrigo José Zonzin Esteves

28 de abril de 2025

1 Introdução

Modelos matemáticos são representações abstratas que utilizam linguagem e estruturas matemáticas para descrever fenômenos do mundo real, permitindo sua análise, compreensão e previsão. Eles são amplamente empregados em diversas áreas do conhecimento, como física, biologia, economia, engenharia etc. Equações Diferenciais, Redes Complexas, Autômatos Celulares, Cadeias de Markov e Algoritmos Genéticos são exemplos desses modelos.

Dentro do amplo espectro dos modelos matemáticos, os algoritmos genéticos (AG) destacam-se como uma classe de métodos de otimização e busca inspirados nos mecanismos de seleção natural e evolução biológica. Esses algoritmos utilizam princípios como seleção, recombinação e mutação para iterativamente refinar soluções para problemas, modelando o processo evolutivo de forma computacional.

Dessa forma, o objetivo desse trabalho é implementar um AG de representação real para otimizar uma função sem restrição em seu domínio.

2 Métodos

Para a implementação do AG, foi utilizado uma classe Gene, que contém os atributos **alelos** (um vetor de floats) e **dim_alelos** (dimensão do vetor de alelos). Alguns métodos foram implementados para manipulação da estrutura de dados.

Além disso, uma classe AG reúne os atributos essenciais para a representação computacional do AG: um vetor de Genes, inicializado aleatoriamente, o número de indivíduos, o intervalo de busca, as taxas de mutação etc. O Anexo A apresenta a implementação completa.

O objetivo da prática consiste em otimizar a Função de Ackley, dada pela Equação 1, por meio de um AG com representação real. A Figura 1 apresenta o *plot* da função objetivo, onde o valor ótimo pode ser observado como $f(\vec{0}) = 0$.

$$f(\vec{x}) = -20e^{-0.2\sqrt{\frac{1}{n}\sum x_i^2}} - e^{\frac{1}{n}\sum cos(2\pi x_i)} + 20 + e \tag{1}$$

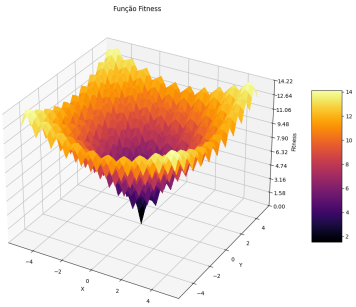

Figura 1: Plot da Equação 1

3 Resultados

A Figura 2 apresenta os resultados obtidos para os seguintes hiperparâmetros em 20 execuções. Para cada execução, um arquivo de texto foi utilizado para salvar o vetor de população, o melhor indivíduo e o fitness associado a ele.

- Tamanho da População: 20
- Dimensão do Gene: 5
- Taxa de Mutação: 0.1
- Intervalo de Busca: $-2 \leq x \leq 2$
- Gerações: 60

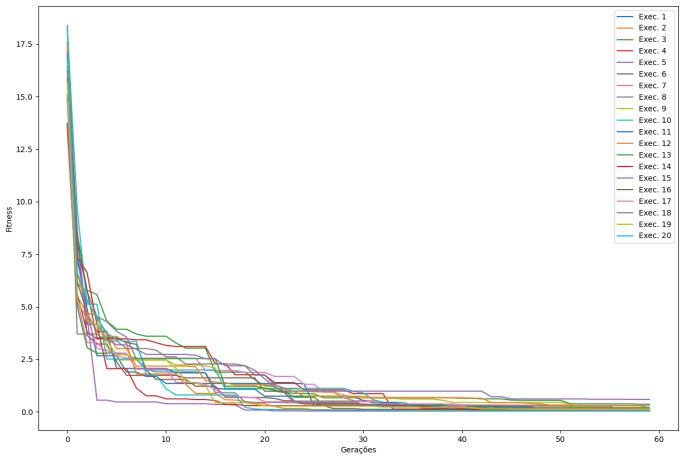

Figura 2: Resultados obtidos

Como se observa, o AG convergiu na maior parte dos testes. Em algumas execuções, no entanto, há indícios de que o AG tenha se estagnado em um mínimo local com imagem entre 0 e 1.5. A consistência dos resultados indica que o AG é suficientemente robusto para a otimização de funções congêneres.

A Figura 3 apresenta o comportamento do fitness ao longo das gerações para diferentes valores de taxas de mutação: 0.001, 0.01, 0.5, 0.1 e 0.2. A taxa de 0.001 não foi capaz de obter o mínimo global, enquanto a taxa de 0.15 obteve o valor ótimo mais rapidamente. De maneira intermediária, a taxa 0.05 foi convergiu, mas mais lentamente.

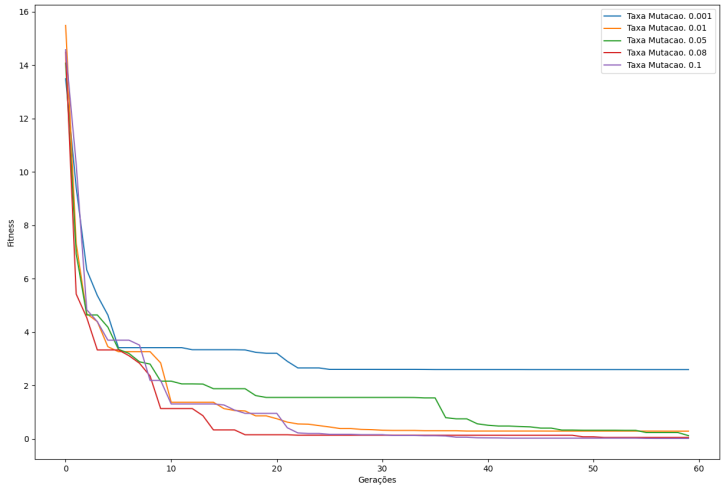

Figura 3: Fitness *vs* Gerações *vs* Taxa de Mutação

Por fim, para uma otimização de parâmetros mais aprofundada, testou-se uma combinação dos seguintes parâmetros:

- taxa de Mutação: 1%, 5%, 10%
- Cruzamento: 60%, 80%, 100%
- População: 25, 50, 100
- Gerações: 25, 50, 100

Os resultados são apresentados nas páginas 9–12, onde a melhor combinação foi dada por $Mutacão = 0.1$, $Cruzamento = 0.6$, $Populacão = 100$, $Geracões = 100$ e $MelhorFitness = 2.57841929540703e - 06$.

Observa-se ainda que o fitness tende a melhorar com o aumento do número de gerações e da população, independentemente da configuração de mutação ou cruzamento. Isso é esperado, pois mais gerações e populações maiores favorecem uma exploração mais abrangente do espaço de busca, permitindo uma convergência mais refinada.

A taxa de mutação de 0.01 apresentou desempenhos consideráveis, particularmente em combinações de maiores populações e gerações. Por exemplo, valores de fitness próximos de 10^{-6} foram alcançados com $pop\_size = 100$ e $geracoes = 50, 100$. Por outro lado, taxas de mutação mais elevadas (0,05 e 0,1) demonstraram comportamentos mistos: enquanto evitaram a estagnação em mínimos locais em alguns casos, em outros impediram a adequada convergência ao ótimo, evidenciado por fitness relativamente elevados mesmo em execuções longas.

Quanto à taxa de cruzamento, a variação entre 0,6, 0,8 e 1,0 influenciou moderadamente o desempenho. Cruzamentos completos ($crossover = 1.0$) não garantiram melhores resultados em todas as circunstâncias; em alguns casos, a combinação de ($crossover = 0.8$) e taxa de mutação baixa (0.01) resultou em valores de fitness mais precisos.

O efeito da população é bastante direto: populações maiores (100 indivíduos) associadas a um número elevado de gerações (100) permitiram alcançar fitness mínimos extremamente baixos, destacando a importância da diversidade genética inicial e da manutenção de variedade durante a evolução.

Em resumo, a configuração que mais favoreceu o algoritmo envolveu baixas taxas de mutação, taxas de cruzamento moderadamente elevadas (0,8) e combinações de populações grandes com um número expressivo de gerações.

4 Conclusão

A Código completo

```python
import numpy as np
import random
import math
import matplotlib.pyplot as plt
import matplotlib as mpl
import sys


class Gene:
    def __init__(self, alelos):
        self.alelos = alelos
        self.alelos_dim = len(self.alelos)

    def __str__(self):
        return str(self.alelos)

    def shape(self):
        return self.alelos_dim


class AG:
    def __init__(self, pop_size=10, gene_size=40, taxa_mutacao=0.01, intervalo=(-2, 2), minMaxIndiv = (-20, 20)):
        self.pop_size = pop_size
        self.gene_size = gene_size
        self.intervalo = intervalo
        self.minMaxIndiv = minMaxIndiv
        self.pop = self._gerar_populacao()
        self.taxa_mutacao = taxa_mutacao


    def _gerar_populacao(self):
        vmin, vmax = self.minMaxIndiv
        return [Gene([random.uniform(vmin, vmax) for _ in range(self.gene_size)]) for _ in range(self.pop_size)]

    def print_pop(self):
        for gene in self.pop:
            print(gene)

    def fitness(self, gene):
        x = gene.alelos
        n = len(x)
        sum1 = sum([xi ** 2 for xi in x])
        sum2 = sum([math.cos(2 * math.pi * xi) for xi in x])
        parte1 = -20 * math.exp(-0.2 * math.sqrt(sum1 / n))
        parte2 = -math.exp(sum2 / n)
        return parte1 + parte2 + 20 + math.e
```

```
47
48      def roleta(self):
49          valores_fitness = np.array([1 / self.fitness(gene) for gene in
        self.pop])
50          valores_fitness = valores_fitness / np.sum(valores_fitness)
51          return self.pop[np.random.choice(a=range(self.pop_size),
    replace=False, p=valores_fitness)]
52
53      def crossoverBLXAlphaBeta(self, paiX, paiY, alpha=0.75, beta=0.25)
    :
54          filhos = []
55
56          d = [abs(paiX.alelos[i] - paiY.alelos[i]) for i in range(self.
    gene_size)]
57
58          for _ in range(2):
59              alelos_filho = []
60              for i in range(self.gene_size):
61                  if self.fitness(paiX) <= self.fitness(paiY):
62                      x1 = max(paiX.alelos[i] - alpha * d[i], self.
    intervalo[0])
63                      x2 = min(paiY.alelos[i] + beta * d[i], self.
    intervalo[1])
64                  else:
65                      x1 = max(paiY.alelos[i] - beta * d[i], self.
    intervalo[0])
66                      x2 = min(paiX.alelos[i] + alpha * d[i], self.
    intervalo[1])
67                  alelos_filho.append(random.uniform(x1, x2))
68              filhos.append(Gene(alelos_filho))
69
70          return filhos[0], filhos[1]
71
72      def mutacao(self, gene):
73          pmin, pmax = self.intervalo
74          for i in range(len(gene.alelos)):
75              if random.random() < self.taxa_mutacao:
76                  gene.alelos[i] = random.uniform(pmin, pmax)
77          return gene
78
79      def mutacao_suave(self, gene):
80          for i in range(len(gene.alelos)):
81              if random.random() < self.taxa_mutacao:
82                  perturbacao = random.gauss(0, 0.1)
83                  gene.alelos[i] += perturbacao
84
85                  gene.alelos[i] = max(self.intervalo[0], min(self.
    intervalo[1], gene.alelos[i]))
86          return gene
87
88
```

```
    def nova_geracao(self):
        nova_pop = []
        while len(nova_pop) < self.pop_size:
            pai1 = self.roleta()
            pai2 = self.roleta()
            while pai1 == pai2:
                pai2 = self.roleta()

            filho1, filho2 = self.crossoverBLXAlphaBeta(pai1, pai2)

            nova_pop.append(self.mutacao_suave(filho1))

            if len(nova_pop) < self.pop_size:
                nova_pop.append(self.mutacao_suave(filho2))

        self.pop = nova_pop

    def nova_geracao_elitismo(self):
        nova_pop = [self.melhor_individuo()]

        while len(nova_pop) < self.pop_size:
            pai1 = self.roleta()
            pai2 = self.roleta()

            filho1, filho2 = self.crossoverBLXAlphaBeta(pai1, pai2)

            nova_pop.append(self.mutacao(filho1))

            if len(nova_pop) < self.pop_size:
                nova_pop.append(self.mutacao(filho2))

        self.pop = nova_pop

    def melhor_individuo(self):
        return min(self.pop, key=self.fitness)


if __name__ == "__main__":
    #random.seed(42)

    ag = AG(pop_size=30, gene_size=10, taxa_mutacao=0.1, intervalo
    =(-2, 2))
    #gene_0 = Gene(alelos=[0]*6)
    #print('Gene 0:', gene_0)
    #print(f"Fitness(0) = {ag.fitness(gene_0)} ")
    #ag.print_pop()

    fitness_result = []

    geracoes = 100
    for i in range(geracoes):
```

```
139         print(f"\nGera  o {i}")
140         ag.print_pop()
141 
142         melhor = ag.melhor_individuo()
143         print(f"\nMelhor indiv duo: {melhor}")
144         print(f"Fitness: {ag.fitness(melhor)}\n")
145 
146         fitness_result.append(ag.fitness(melhor))
147 
148         ag.nova_geracao_elitismo()
149 
150 
151     plt.figure(figsize=(15, 10))
152     plt.scatter(x=range(geracoes), y=fitness_result)
153 
154     plt.savefig('fitness.png')
```

Listing 1: Implementação do AG Real

Melhor Fitness	Mutação	Cruzamento	População	Gerações
0.4995819462101889	0.01	0.6	25	25
0.14837328929753157	0.01	0.6	25	50
0.042084457104256234	0.01	0.6	25	100
0.008238042712335858	0.01	0.6	50	25
9.975171277876171e-06	0.01	0.6	50	50
0.0001083352294668493 7	0.01	0.6	50	100
0.000995672108239365	0.01	0.6	100	25
0.004319949914247889	0.01	0.6	100	50
0.0003013678300480826 5	0.01	0.6	100	100
0.09139586440326442	0.01	0.8	25	25
0.22239554512224702	0.01	0.8	25	50
0.01986914863357514	0.01	0.8	25	100
0.004051963473298503	0.01	0.8	50	25
0.11037286206869057	0.01	0.8	50	50
0.020590464399245167	0.01	0.8	50	100
0.0030407922917992103	0.01	0.8	100	25
0.0019903655382091934	0.01	0.8	100	50
0.0005773190060982536	0.01	0.8	100	100
0.2064058651301406	0.01	1.0	25	25
0.08308839393834377	0.01	1.0	25	50
0.07852813467089037	0.01	1.0	25	100
0.17831647991896604	0.01	1.0	50	25
0.07307157412632082	0.01	1.0	50	50
0.018762004016001033	0.01	1.0	50	100
0.03717449651673066	0.01	1.0	100	25
3.938768970090933e-06	0.01	1.0	100	50
0.19009053681179156	0.01	1.0	100	100
0.09723823348332461	0.05	0.6	25	25
0.09761710722175154	0.05	0.6	25	50
0.03192822664426176	0.05	0.6	25	100
0.02023222799476132	0.05	0.6	50	25
0.09416246093639602	0.05	0.6	50	50
0.025087009194550536	0.05	0.6	50	100
0.08406138788166162	0.05	0.6	100	25
0.00514066323429363	0.05	0.6	100	50
0.0005782284643101043	0.05	0.6	100	100

0.03479647168612532	0.05	0.8	25	25
0.029537026327976168	0.05	0.8	25	50
0.015011765866070892	0.05	0.8	25	100
0.031381897465430075	0.05	0.8	50	25
0.029741187991540574	0.05	0.8	50	50
0.004414946227495609	0.05	0.8	50	100
0.01780970963622197	0.05	0.8	100	25
0.015544796975938358	0.05	0.8	100	50
0.0015182703671396958	0.05	0.8	100	100
1.217109456344247	0.05	1.0	25	25
0.04318351997217862	0.05	1.0	25	50
0.008686126078619072	0.05	1.0	25	100
0.017828474812799033	0.05	1.0	50	25
0.023824725655949397	0.05	1.0	50	50
0.0419897578759465	0.05	1.0	50	100
0.021342968204721036	0.05	1.0	100	25
0.018366940331525416	0.05	1.0	100	50
0.010571019717762642	0.05	1.0	100	100
0.11457558844530924	0.1	0.6	25	25
0.03586794847405317	0.1	0.6	25	50
0.021615767912031902	0.1	0.6	25	100
0.00302519082825059	0.1	0.6	50	25
0.0019504198376236737	0.1	0.6	50	50
0.013438926495223047	0.1	0.6	50	100
0.08210904009233433	0.1	0.6	100	25
0.02406344955671047	0.1	0.6	100	50
2.5784192954070306e-06	0.1	0.6	100	100
0.005846014697315827	0.1	0.8	25	25
0.02161961673417112	0.1	0.8	25	50
0.049088816148799896	0.1	0.8	25	100
0.024937954326266	0.1	0.8	50	25
0.0454187983412635	0.1	0.8	50	50
0.015810355648842478	0.1	0.8	50	100
0.002392933772061401	0.1	0.8	100	25
0.0001915914072836422	0.1	0.8	100	50
0.0060580713855000745	0.1	0.8	100	100
0.0010319679553494332	0.1	1.0	25	25

0.023518979670300144	0.1	1.0	25	50
0.0014420787758280262	0.1	1.0	25	100
0.0026335418226826057	0.1	1.0	50	25
0.0001376812976556785	0.1	1.0	50	50
0.00030187292813765865	0.1	1.0	50	100
0.00017765486891052618	0.1	1.0	100	25
0.0003511152422146502	0.1	1.0	100	50
7.008326064239512e-05	0.1	1.0	100	100

.