

Universidade de Brasília

Instituto de Física

Davi Valadares Rodrigues Feliciano

Projeto de Trabalho de Conclusão de Curso

**Busca por Extremos de Funções
Multidimensionais Utilizando Algoritmo
Genético**

Brasília

28 de abril de 2022

Davi Valadares Rodrigues Feliciano

**Busca por Extremos de Funções Multidimensionais
Utilizando Algoritmo Genético**

Projeto de Trabalho de Conclusão de Curso
apresentado para obtenção do grau de
Bacharel em Física pelo Instituto de Física
da Universidade de Brasília.

Universidade de Brasília
Instituto de Física

Orientador: Luiz Antonio Ribeiro Junior

Brasília
28 de abril de 2022

Resumo

Nas diversas áreas do conhecimento humano é bem conhecida uma série de problemas de otimização. A solução desses problemas consiste em encontrar em um conjunto de configurações, um subconjunto, ou mesmo um elemento que melhor satisfaça um ou mais vínculos previamente determinados. Uma estratégia famosa por solucionar problemas de tal classe de forma rápida e eficiente consiste no emprego dos algoritmos genéticos. São assim chamados devido à forte inspiração em fenômenos da biologia evolutiva — como mutação, recombinação (ou *crossover*) e seleção — na elaboração de suas etapas de execução. Neste projeto de trabalho é proposto um algoritmo genético para a otimização de funções reais, isto é, para a procura dos pontos nos quais a função é máxima (ou mínima). Sua implementação é feita em Python com o uso da biblioteca NumPy. O algoritmo foi aplicado na otimização de algumas funções de exemplo e foi feita uma discussão acerca dos resultados e da eficácia e performance do programa.

Palavras-chaves: Algoritmo Genético, Otimização Numérica, Python.

Abstract

On the diverse areas of human knowledge, there are a series of well-known optimization problems. The solution of these problems consists in finding, in a collection of configurations, a subset, or even an element that better satisfies one or more constraints previously determined. A famous strategy to solve such problems in a quick and efficient fashion relies on the use of genetic algorithms. They receive such name due to the strong inspiration in phenomena of evolutionary biology—like mutation, recombination (or crossover) and selection—in its execution. In this project, a genetic algorithm is proposed for the optimization of real functions, that is, for the search of the points in which the function is maximum (or minimum). Its implementation is done using the Python programming language, by means of the NumPy library. The code developed was applied in the optimization of some example functions, and a discussion was conducted concerning the results, as well as on the efficiency and performance of the implementation.

Keywords: Genetic Algorithm, Numerical Optimization, Python.

Listas de ilustrações

Figura 1 – Gráfico da função definida na Equação (1.1), com $n = 9$ e $\sigma = 0,4$.	7
Figura 2 – Fluxograma geral de um algoritmo genético.	9
Figura 3 – Diagrama ilustrando—em um caso com $m = 1$ e $l = 8$ —a recombinação entre os indivíduos S_k e S_{k+1} e mutação nos filhos gerados segundo as respectivas máscaras de recombinação e mutação R e M .	12
Figura 4 – (a) Gráfico da função $f_1(x, y)$. (b) Gráfico da função $f_2(x, y)$.	16
Figura 5 – Curvas de nível da função $f_1(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100 ^a geração na otimização da função, com $p_2 = p_3 = 5\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.	18
Figura 6 – Evolução dos valores da função $f_1(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 5\%$.	19
Figura 7 – Curvas de nível da função $f_1(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100 ^a geração na otimização da função, com $p_2 = p_3 = 20\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.	20
Figura 8 – Evolução dos valores da função $f_1(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 20\%$.	21
Figura 9 – Curvas de nível da função $f_2(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100 ^a geração na otimização da função, com $p_2 = p_3 = 5\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.	22
Figura 10 – Evolução dos valores da função $f_2(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 5\%$.	23
Figura 11 – Curvas de nível da função $f_2(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100 ^a geração na otimização da função, com $p_2 = p_3 = 20\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.	24
Figura 12 – Evolução dos valores da função $f_2(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 20\%$.	25

Sumário

1	INTRODUÇÃO	6
1.1	Problemas de Otimização	6
1.2	O Algoritmo Genético	8
1.3	A Implementação do Algoritmo Genético	8
2	METODOLOGIA	10
2.1	Codificação	10
2.2	Seleção	11
2.3	Recombinação e Mutação	12
2.4	Estratégia Elitista	13
3	RESULTADOS PRELIMINARES	15
3.1	Testes em Funções	15
3.1.1	Cosseno Amortecido	15
3.1.2	Gaussianas Próximas	17
3.2	Testes de Performance	17
4	CONCLUSÃO E PERSPECTIVAS	26
5	CRONOGRAMA	27
	REFERÊNCIAS	28

1 Introdução

1.1 Problemas de Otimização

Em matemática e ciência da computação, problemas de otimização consistem naqueles em que a solução é um elemento de um conjunto de candidatos que melhor satisfaz uma determinada série de condições. Um exemplo clássico dessa classe é o problema da mochila (*knapsack problem*, em inglês)⁽¹⁾, no qual procura-se, dentre um conjunto de itens com diversos preços e pesos, o subconjunto que maximiza o valor total. Esse valor é calculado somando-se os preços dos itens postos na mochila.

Entretanto a solução deve respeitar um vínculo: o peso total dos objetos escolhidos não deve exceder o peso máximo suportado pela mochila. Ainda que seja possível encontrar a solução exata desse problema usando algoritmos de programação dinâmica, a complexidade de tais algoritmos é $\mathcal{O}(nw)$, onde n é o número de itens e w é a capacidade da bolsa. Assim, o tempo de execução para muitos itens — que é o caso de interesse, usualmente — pode ser mais longo do que o desejado.

Outro exemplo comum de problema de otimização é o de otimização numérica, do qual constituem solução os pontos de máximo ou mínimo global de uma função $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Tomemos como exemplo simples a função

$$f(x, y) = \cos(n\pi r) \exp\left(-\frac{r^2}{\sigma^2}\right), \quad (1.1)$$

definida no domínio $[-1, 1] \times [-1, 1]$, cujo gráfico se encontra na Figura (1). Analiticamente é fácil determinar o ponto de máximo global em $r = 0$ usando métodos de cálculo para funções de múltiplas variáveis. Contudo, não é sempre que pode-se dispor de tal facilidade. Nos casos em que a função objetivo é de maior complexidade, com domínio contido espaços de maior dimensão; em que a função tem evolução temporal; em que a função possui diversas descontinuidades; ou em que a função depende de variáveis aleatórias, o problema se torna impraticável de resolver de forma analítica. Isto posto, como desenvolver um algoritmo para encontrar a solução?

Uma classe de algoritmos mais simples que se propõem a resolver esse problema são os algoritmos do tipo escalada de morro (*hill climbing*, em inglês). Sugerido pelo nome, o funcionamento desses algoritmos consiste, resumidamente, em sortear um ponto inicial no domínio da função, calcular o gradiente da função no ponto, e seguir na direção resultante por uma distância pré-definida, e repetir esses passos até que o módulo do vetor gradiente seja próximo de zero, respeitando uma tolerância previamente determinada.

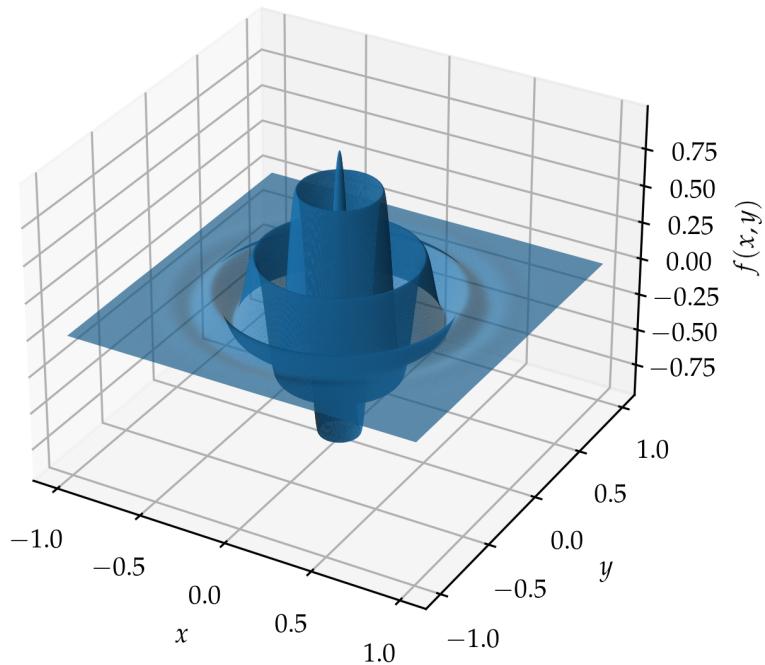


Figura 1 – Gráfico da função definida na Equação (1.1), com $n = 9$ e $\sigma = 0,4$.

O problema com esse tipo de algoritmo é que, quando aplicado em funções como a definida anteriormente, a tendência é de que a solução encontrada seja um máximo local na grande maioria das vezes. No caso da função definida pela Equação (1.1), o máximo local só seria encontrado pelo algoritmo se o ponto inicial fosse tal que $r < 1/n$. Se os pontos iniciais do algoritmo forem gerados aleatoriamente, para o domínio definido, a probabilidade de um ponto pertencer a esta região é $P(n) = \pi/4n^2$. Para $n = 9$, como na Figura (1), $P \approx 1\%$. Ademais, é possível que esse tipo de algoritmo indique erroneamente como solução pontos de sela e planícies em determinados casos, o que o torna ainda menos eficaz para o propósito.

Uma outra categoria de algoritmos que podem resolver ambos os problemas de forma rápida e chegar a uma solução que se aproxima o suficiente da solução exata são os algoritmos genéticos. Tais algoritmos, quando bem implementados, podem chegar a uma solução aproximada para o problema da mochila de forma rápida para valores de n e w ordens de grandeza superiores aos que tornariam praticável o uso da estratégia proposta anteriormente.

No problema de otimização numérica, um algoritmo desse tipo é vantajoso pois é capaz de chegar na solução de forma rápida na maioria dos casos, identificando não

só o máximo global, como também os máximos locais contidos na região de busca. Outra vantagem é que esse método pode ser aplicado sem problemas em espaços de busca com grande número de dimensões, ou em funções não estacionárias, contínuas ou descontínuas. Além disso, como será mostrado posteriormente, o algoritmo é paralelizável, o que pode acelerar a obtenção de uma solução.

1.2 O Algoritmo Genético

Os algoritmos genéticos foram primeiramente desenvolvidos por John H. Holland⁽²⁾⁽³⁾ na década de 60. Levam esse nome pois sua formulação teve como forte influência o processo de evolução natural que fomenta a origem das espécies desde o surgimento da vida em nosso planeta. Em sua execução, é inicializada uma população de indivíduos da mesma espécie*. Cada indivíduo corresponde a um candidato a solução do problema de otimização proposto, e tem sua identidade codificada pelo seu material genético[†]. Então, sob algum critério, é selecionada uma parcela desses indivíduos, a qual dará origem a novos descendentes, cujos cromossomos serão gerados pela recombinação dos cromossomos correspondentes nos pais. Nesse passo é introduzida uma probabilidade de mutação nos genes dos filhos.

Ao final desse processo, descarta-se a parcela não selecionada da população, dando origem a uma nova geração, formada pelos pais seus filhos. Esse processo é repetido iterativamente até que uma condição de parada seja atingida. Essa condição pode ser imposta sobre o número de gerações ou sobre o tempo de execução do programa, por exemplo. Respeitando as diferenças de implementação em cada passo, devido as peculiaridades de cada tipo de problema de otimização proposto, essas etapas são presentes em todo algoritmo genético, e são bem resumidas no fluxograma presente na Figura (2).

1.3 A Implementação do Algoritmo Genético

A linguagem de programação escolhida para a implementação do algoritmo foi Python[‡]. Amplamente utilizada no meio acadêmico, trata-se de uma linguagem de fácil

*Isso significa que seus respectivos códigos genéticos possuem a mesma estrutura: mesmo número de cromossomos com uma mesma quantidade de genes. Assim, pode haver reprodução entre tais indivíduos.

[†]No caso do problema da mochila, por exemplo, uma estrutura natural para o material genético é um cromossomo binário, com n genes, cada um correspondente a um objeto do problema. Por outro lado, no problema de otimização numérica, dois cromossomos seriam necessários, um para codificar cada coordenada em cada dimensão do espaço de busca.

[‡]Versão 3.10.4, com NumPy na versão 1.22.3, e Matplotlib na versão 3.5.1. Código disponível em <https://github.com/davifeliciano/num_opt_ga>.

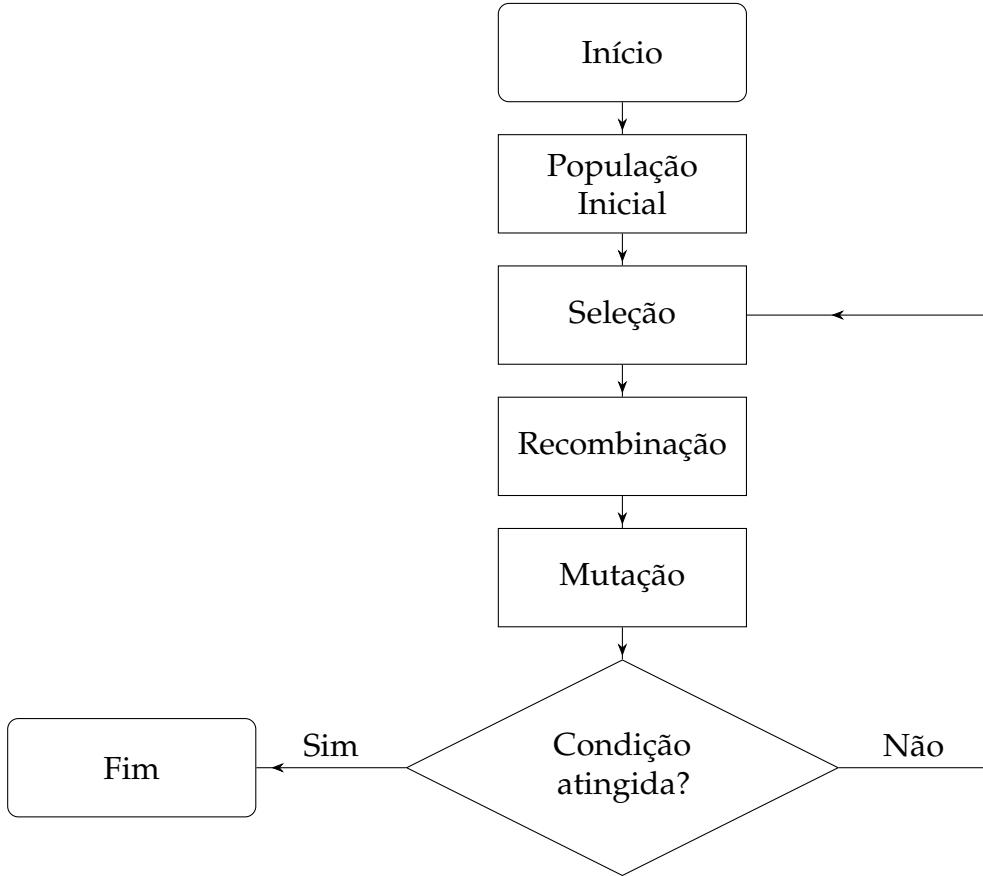


Figura 2 – Fluxograma geral de um algoritmo genético.

compreensão e aprendizado, além de contar com diversas bibliotecas já implementadas com o objetivo de resolver problemas recorrentes.

As duas principais bibliotecas utilizadas neste projeto foram NumPy⁽⁴⁾ e Matplotlib⁽⁵⁾. A primeira se trata de uma biblioteca para manipulação numérica e vetorizada de matrizes. É implementada na linguagem C, o que confere performance às operações numéricas, mantendo a facilidade de uso e versatilidade características da linguagem Python. Já a segunda, se trata de uma biblioteca para a criação de gráficos matemáticos, que será de suma importância na ilustração de forma clara dos resultados obtidos.

Os detalhes acerca da codificação escolhida para o material genético dos indivíduos, bem como os métodos utilizados nas etapas de seleção, recombinação e mutação foram propostos em 2006 por Roncaratti, Gargano e Silva⁽⁶⁾, e serão abordados de forma breve no capítulo seguinte.

2 Metodologia

2.1 Codificação

Seja uma população de n indivíduos representada pelo conjunto

$$A = \{A_1, \dots, A_k, \dots, A_n\}, \quad (2.1)$$

onde cada indivíduo tem seu código genético representado por m cromossomos de l bits, de forma que, usando uma notação matricial,

$$A_k = \begin{bmatrix} a_{11}^{(k)} & \cdots & a_{1m}^{(k)} \\ \vdots & \ddots & \vdots \\ a_{l1}^{(k)} & \cdots & a_{lm}^{(k)} \end{bmatrix}, \quad (2.2)$$

onde $a_{ij}^{(k)} \in \{0, 1\}$, para i, j e k inteiros tais que $i \in [0, l]$, $j \in [0, m]$ e $k \in [0, n]$.

Para um problema de otimização numérica de uma função $f : \mathcal{C} \subseteq \mathbb{R}^m \rightarrow \mathbb{R}$, deve existir um mapa injetivo entre A e o conjunto

$$X = \{X_1, \dots, X_k, \dots, X_n\}, \quad (2.3)$$

onde

$$X_k = (x_1^{(k)}, \dots, x_j^{(k)}, \dots, x_m^{(k)}) \quad (2.4)$$

são vetores de \mathbb{R}^n e representam a posição do respectivo indivíduo no espaço de busca. Tal espaço é um subconjunto de \mathcal{C} definido pelo produto cartesiano

$$\mathcal{S} = \bigtimes_{j=1}^m [x_j^{(\min)}, x_j^{(\max)}], \quad (2.5)$$

sendo $x_j^{(\min)}$ e $x_j^{(\max)}$, respectivamente, os valores mínimo e máximo para a coordenada $x_j^{(k)}$.

Como l bits são capazes de representar números inteiros em $[0, 2^l]$, uma forma natural de mapear tal intervalo em $[x_j^{(\min)}, x_j^{(\max)}]$ é

$$x_j^{(k)} = x_j^{(\min)} + \frac{x_j^{(\max)} - x_j^{(\min)}}{2^l - 1} \sum_{i=1}^l a_{ij}^{(k)} 2^{i-1}, \quad (2.6)$$

cuja relação inversa é determinada por

$$\sum_{i=1}^l a_{ij}^{(k)} 2^{i-1} = \left\lfloor \frac{(x_j^{(k)} - x_j^{(\min)})(2^l - 1)}{x_j^{(\max)} - x_j^{(\min)}} \right\rfloor. \quad (2.7)$$

2.2 Seleção

Para o processo de seleção é introduzida uma nova função, denominada função desempenho. Tal função é alguma função $g : \mathbb{R} \rightarrow \mathbb{R}$ a partir da qual se calcula a probabilidade de seleção do indivíduo k

$$P_k = \frac{g(f(X_k))}{\sum_{j=1}^n g(f(X_j))}. \quad (2.8)$$

Existe uma liberdade para a escolha da função desempenho, desde que se respeite a condição a função seja sempre positiva. A ideia é que se escolha uma função desempenho que selecione os melhores indivíduos, mas ainda mantenha uma boa variedade genética na próxima geração. Assim, em toda geração, em média, haverá indivíduos em todo o espaço de busca, evitando que a solução convirja de forma muito rápida*.

Nesse trabalho a função desempenho usada foi uma função linear $g(x) = ax + b$ sendo a e b definidos por

$$a = \begin{cases} \frac{\mu(h-1)}{f_{max}-\mu}, & \text{se } f_{min} > \frac{h\mu-f_{max}}{h-1} \\ \frac{\mu}{\mu-f_{min}}, & \text{caso contrário} \end{cases} \quad (2.9)$$

e

$$b = \begin{cases} \frac{\mu(f_{max}-h\mu)}{f_{max}-\mu}, & \text{se } f_{min} > \frac{h\mu-f_{max}}{h-1} \\ -\frac{\mu f_{min}}{\mu-f_{min}}, & \text{caso contrário} \end{cases}, \quad (2.10)$$

onde

$$f_{min} = \min_{X_k \in X} f(X_k), \quad (2.11)$$

$$f_{max} = \max_{X_k \in X} f(X_k), \quad (2.12)$$

h é um parâmetro real maior que 1 e μ é a média do valor de f sobre a população, dada por

$$\mu = \sum_{k=1}^n \frac{f(X_k)}{n}. \quad (2.13)$$

A seleção dos indivíduos que serão recombinados para formar a geração seguinte é feita por meio de uma roleta simples, permitindo ou não repetições. São feitos $n/2$ sorteios, já que para cada par de indivíduos recombinados, dois novos serão gerados[†]. Desse modo, a geração subsequente será constituída pelos indivíduos pais selecionados e os seus respectivos filhos, seguindo com n indivíduos.

*Retomando a função usada como exemplo no capítulo anterior, definida na Equação (1.1), se escolhermos uma função de desempenho que privilegie de forma desproporcional pontos onde o valor da função é maior. Dependendo da distribuição inicial dos indivíduos, é possível que após algumas gerações, toda a população se concentre em $r = 1/2n$, que é um máximo local.

[†]Por esse motivo, n deve ser um múltiplo de 4.

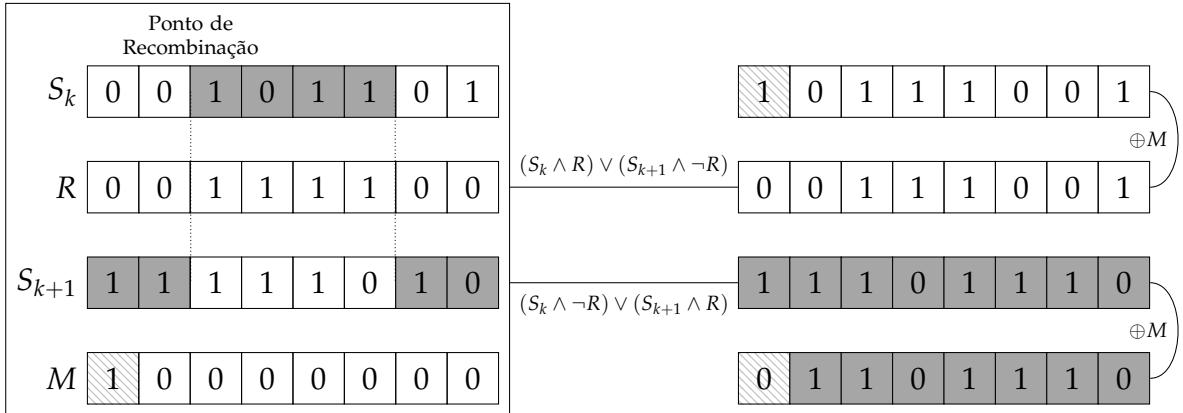


Figura 3 – Diagrama ilustrando — em um caso com $m = 1$ e $l = 8$ — a recombinação entre os indivíduos S_k e S_{k+1} e mutação nos filhos gerados segundo as respectivas máscaras de recombinação e mutação R e M .

2.3 Recombinação e Mutação

Seja o conjunto de indivíduos selecionados

$$S = \{S_1, \dots, S_k, \dots, S_n\}. \quad (2.14)$$

Estes indivíduos passarão, em pares adjacentes[‡], pelo processo de recombinação e mutação, para dar origem a geração seguinte da população. Esse processo é iniciado com o sorteio de duas posições em cada cromossomo, para cada um desses pares. Essas posições são chamadas de pontos de recombinação.

Em seguida, o material genético dos indivíduos filhos será obtido pela união das partes complementares de cada par de cromossomos dos pais e, por fim, são introduzidas mutações nos genes, com uma probabilidade predefinida.

Matematicamente, podemos realizar esse processo para cada par da seguinte forma. Primeiro, devemos sortear os pontos de recombinação. Esse processo é feito por meio da geração de m pares de números naturais aleatórios distintos $(\alpha_j^{(k)}, \beta_j^{(k)})$ tais que $0 \leq \alpha_j^{(k)} < \beta_j^{(k)}$ e $\alpha_j^{(k)} \leq \beta_j^{(k)} < l$ onde $j = 0, 1, \dots, m$ e $k = 0, 2, 4, \dots, n/2 - 2$. Depois, é montada uma matriz R denominada máscara de recombinação, definida por

$$r_{ij}^{(k)} = \begin{cases} 0, & \text{se } i \leq \alpha_j^{(k)} \text{ ou } i > \beta_j^{(k)} \\ 1, & \text{caso contrário} \end{cases}, \quad (2.15)$$

bem como uma máscara de mutação M , tal que

$$m_{ij}^{(k)} = \begin{cases} 1, & \text{se houver mutação} \\ 0, & \text{caso contrário} \end{cases}. \quad (2.16)$$

[‡]O par (S_1, S_2) dá origem a dois indivíduos. O par (S_3, S_4) forma outros dois, e assim segue até que $n/2$ novos indivíduos sejam gerados.

Finalmente, o conjunto dos indivíduos resultantes da recombinação da seleção no conjunto definido pela Equação (2.14) será

$$S' = \{S'_1, \dots, S'_k, \dots, S'_{n/2}\} \quad (2.17)$$

tal que

$$s'_{ij}^{(k)} = (s_{ij}^{(k)} \wedge r_{ij}^{(k)}) \vee (s_{ij}^{(k+1)} \wedge \neg r_{ij}^{(k)}) \oplus m_{ij}^{(k)} \quad (2.18)$$

e

$$s'_{ij}^{(k+1)} = (s_{ij}^{(k)} \wedge \neg r_{ij}^{(k)}) \vee (s_{ij}^{(k+1)} \wedge r_{ij}^{(k)}) \oplus m_{ij}^{(k)}, \quad (2.19)$$

onde \neg , \wedge , \vee e \oplus são os operadores lógicos de negação, conjunção, disjunção e disjunção exclusiva, respectivamente.

Ao final desse processo—resumido no diagrama presente na Figura (3)—teremos uma nova população, sobre a qual poderemos ordenar os indivíduos segundo a função desempenho, verificar as características dos primeiros, e, caso necessário, repetir o algoritmo desde a etapa de seleção.

2.4 Estratégia Elitista

A fim de garantir a reprodução das características dos melhores indivíduos na geração seguinte, uma estratégia elitista pode ser tomada. Seja e o melhor indivíduo[§] do conjunto S definido na Equação (2.14). A ideia dessa abordagem é introduzir—antes do processo se recombinação—cópias de S_e na população[¶] com o objetivo de preservar seu material genético, assim como recombina-lo com o de outros indivíduos, na esperança de obter soluções ainda melhores para o problema.

Para isso, substituímos o conjunto S por

$$S = \left\{ \underbrace{S_e, \dots, S_e}_{e_1 \text{ cópias}}, \underbrace{S_e, \dots, S_e}_{e_2 \text{ cópias}}, \underbrace{S_{e_1+e_2+1}, \dots, S_{n/2}}_{e_3 \text{ cópias dentre os restantes}} \right\}, \quad (2.20)$$

onde os primeiros $e_1 + e_2$ indivíduos foram trocados por S_e ^{||}, e, dentre os indivíduos restantes, são feitas e_3 substituições similares em posições aleatórias.

Ao final desse processo, prossegue-se para a etapa de recombinação, com o detalhe de que os primeiros e_1 indivíduos terão probabilidade de mutação $p_1 = 0$, os e_2 indivíduos seguintes e o restante da população terão probabilidades de mutação não nulas p_2 e p_3 . Como consequência, S_e será parte da população seguinte, uma vez que a recombinação entre um indivíduo e seu clone dá origem a dois outros clones, na

[§]Aquele cuja solução correspondente X_e satisfaz $g(f(X_e)) = \max_{X_k \in X} g(f(X_k))$.

[¶]O conjunto dessas cópias é chamado de elite da população.

^{||}Como a recombinação ocorre em pares, e_1 e e_2 devem ser pares.

ausência de mutações. Ademais, cópias mutadas também estarão presentes, bem como cruzamentos de S_e com e_3 indivíduos distintos.

3 Resultados Preliminares

Nesse capítulo são exibidos os resultados de alguns testes em funções reais em \mathbb{R}^2 . Em cada teste, 8 populações com 1000 indivíduos com cromossomos de 32 bits são evoluídas por 100 gerações. O espaço de busca considerado foi $[-1, 1] \times [-1, 1]$, o valor usado para o parâmetro da função desempenho foi $h = 2$, as configurações para a elite foram $e_1 = 4$, $e_2 = 6$ e $e_3 = 10$ e as probabilidades de mutação p_2 e p_3 escolhidas foram $p_2 = 5\%$ e $p_3 = 5\%$.

Ao final do processo, foram gerados gráficos com as curvas de nível de cada função e com as posições de todos os integrantes de todas as populações. Os 8 melhores indivíduos são marcados de forma distinta, afim de atestar se o algoritmo foi capaz ou não de encontrar a solução real do problema. Um segundo gráfico com os valores das funções dos 200 melhores indivíduos de cada população no decorrer das gerações foi gerado, a fim de mostrar, em caso de convergência, sua rapidez.

Ambos os gráficos foram feitos novamente com probabilidades de mutação $p_2 = p_3 = 20\%$ para demonstrar o papel da mutação no decorrer do algoritmo, e averiguar seu impacto na diversidade genética dos indivíduos. Por fim, é feita uma breve discussão sobre a performance da implementação do algoritmo e sua complexidade de tempo de execução.

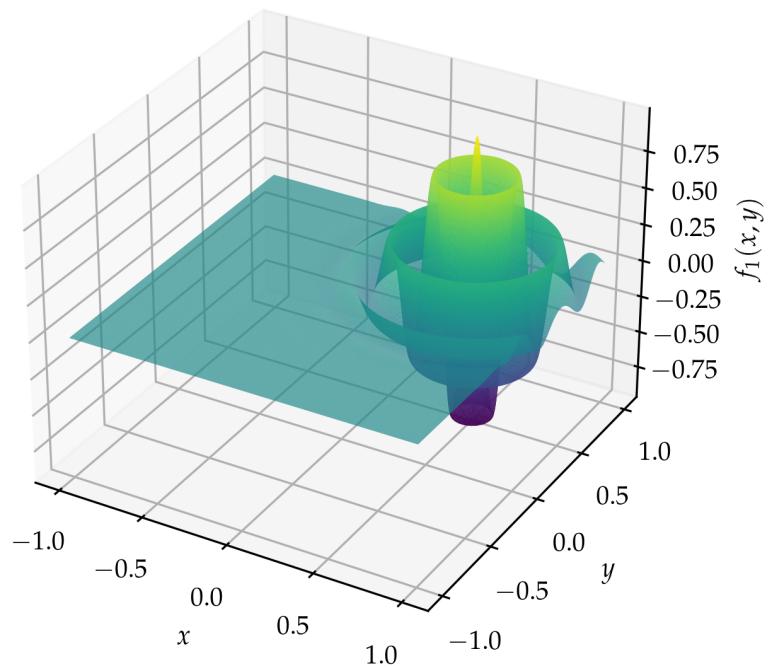
3.1 Testes em Funções

3.1.1 Cosseno Amortecido

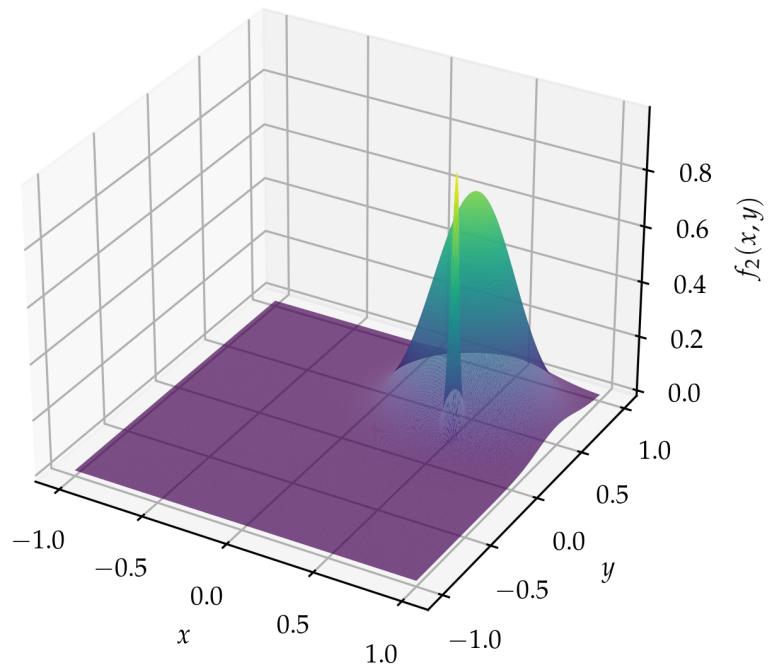
A primeira função testada foi

$$f_1(x, y) = \cos(9\pi r) \exp\left\{-\frac{r^2}{(0,4)^2}\right\}, \text{ com} \\ r = \sqrt{(x - 0,5)^2 + (y - 0,5)^2} \quad (3.1)$$

e os resultados obtidos estão dispostos nas Figuras (5), (6), (7) e (8).



(a)



(b)

Figura 4 – (a) Gráfico da função $f_1(x, y)$. (b) Gráfico da função $f_2(x, y)$.

3.1.2 Gaussianas Próximas

A segunda função testada foi

$$f_2(x, y) = 0,8 \exp \left\{ -\frac{r_1^2}{(0,3)^2} \right\} + 0,88 \exp \left\{ -\frac{r_2^2}{(0,03)^2} \right\}, \text{ onde}$$

$$r_1 = \sqrt{(x - 0,5)^2 + (y - 0,5)^2} \text{ e}$$

$$r_2 = \sqrt{(x - 0,6)^2 + (y - 0,1)^2},$$
(3.2)

com os resultados obtidos ilustrados nas Figuras (9), (10), (11) e (12).

3.2 Testes de Performance

Em testes de tempo de execução o programa escalou bem, sendo sua complexidade temporal $\mathcal{O}(n)$ para o tamanho de população, e $\mathcal{O}(g)$ para número de gerações decorridas. Nas figuras exibidas nesse capítulo, o tempo médio decorrido na evolução das populações de 1000 indivíduos foi de 15 segundos* enquanto que o tamanho da população necessário para que o processo tomasse mais de uma hora foi superior a $n = 10^5$, para o mesmo número de gerações.

Isso mostra que a implementação é escalável para grandes valores de n , o que pode ser desejável em algumas aplicações. É posto a prova também a performance da biblioteca NumPy, cujas funções e estruturas de dados implementadas se mostraram altamente otimizadas.

*Tempo obtido usando uma máquina com processador Intel i7-8550U com frequência máxima de 4GHz. É importante frisar aqui as evoluções das 8 populações ocorreram em 8 processos em paralelo. Assim, é possível que, para uma única população, os tempos de execução sejam ligeiramente menores.

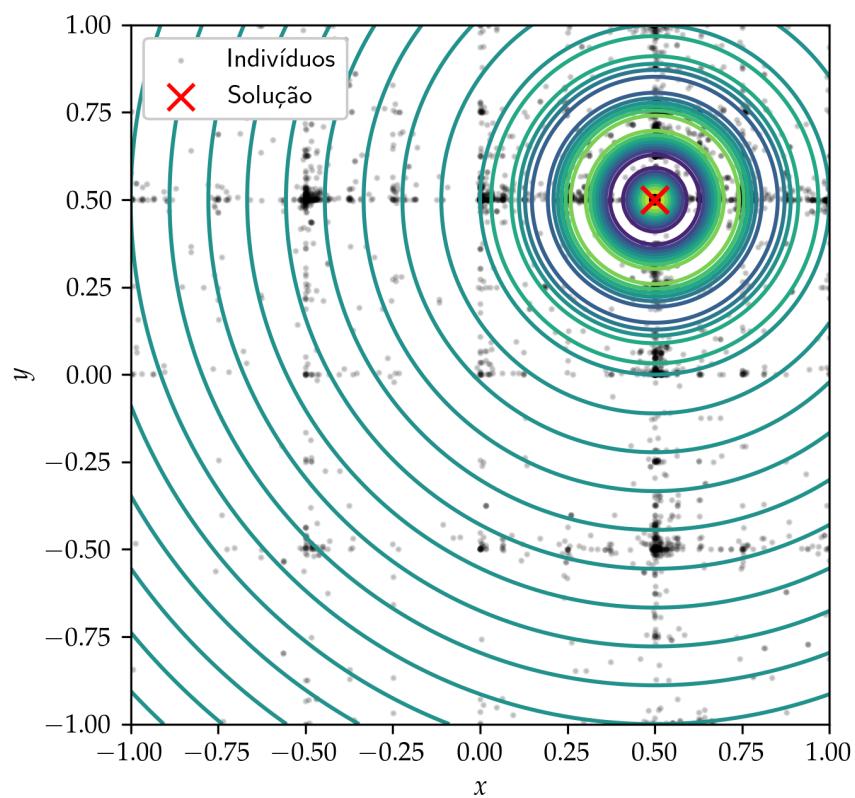


Figura 5 – Curvas de nível da função $f_1(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100^a geração na otimização da função, com $p_2 = p_3 = 5\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.

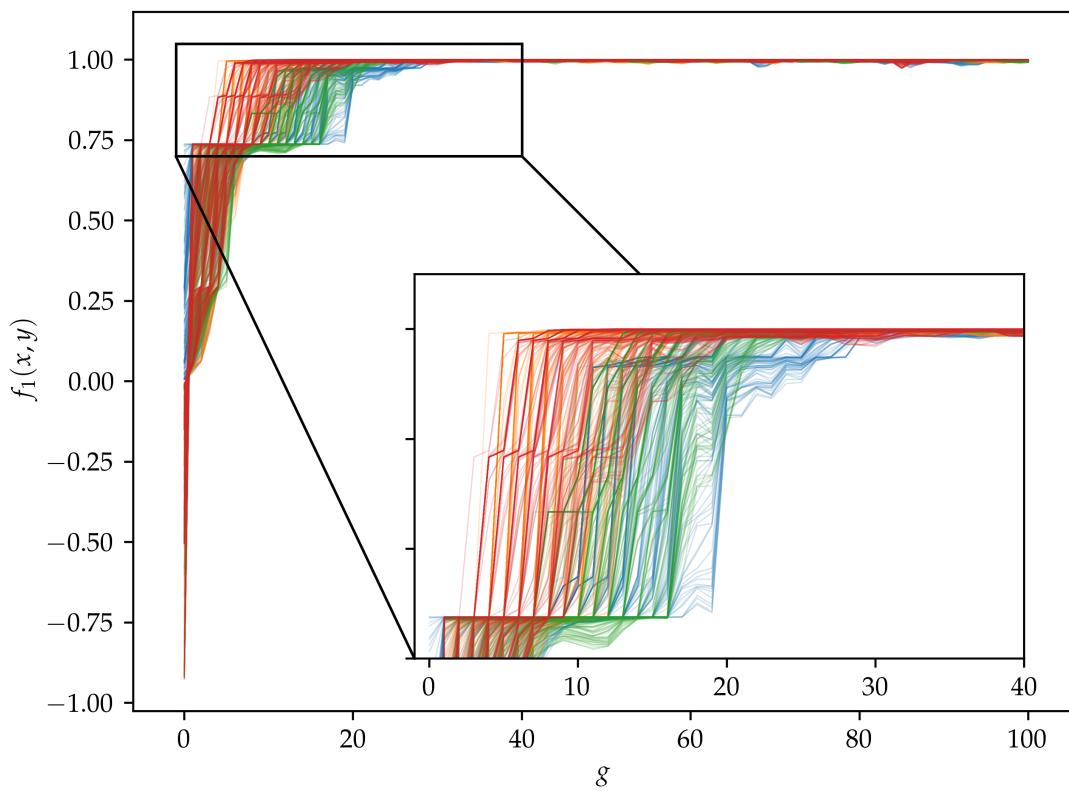


Figura 6 – Evolução dos valores da função $f_1(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 5\%$.

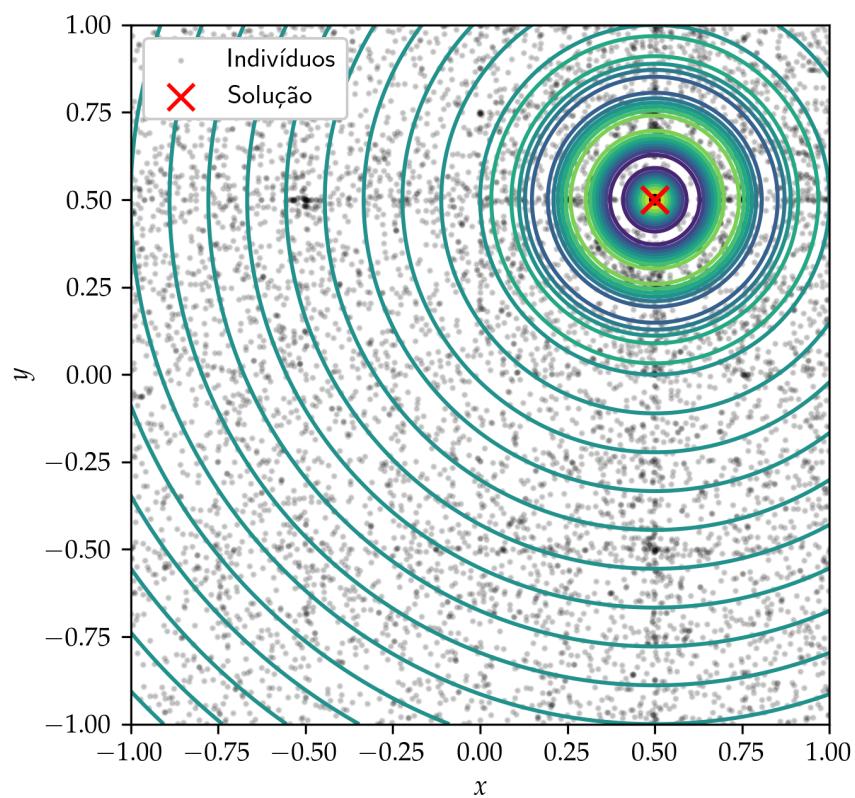


Figura 7 – Curvas de nível da função $f_1(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100^a geração na otimização da função, com $p_2 = p_3 = 20\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.

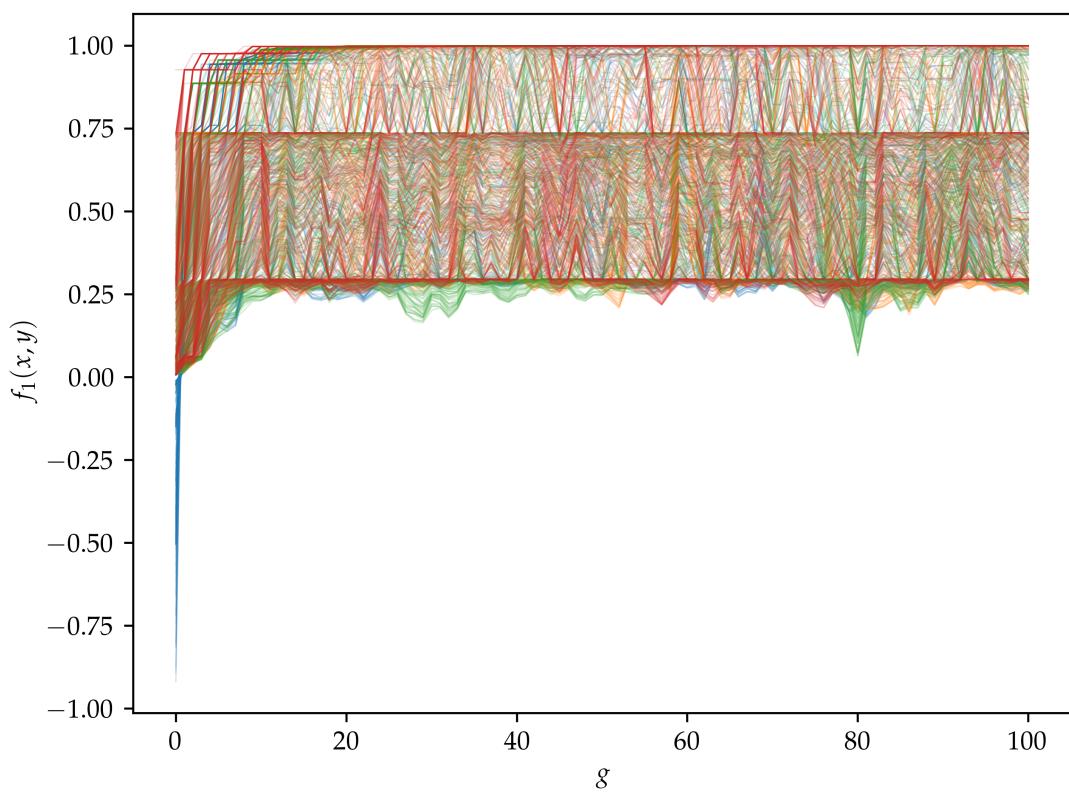


Figura 8 – Evolução dos valores da função $f_1(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 20\%$.

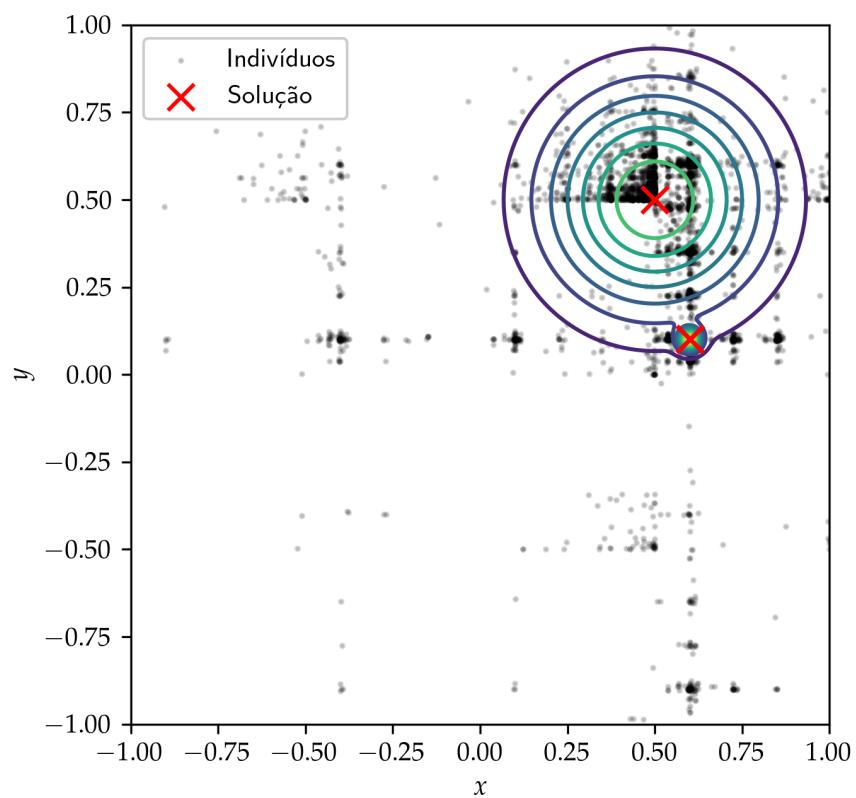


Figura 9 – Curvas de nível da função $f_2(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100^a geração na otimização da função, com $p_2 = p_3 = 5\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.

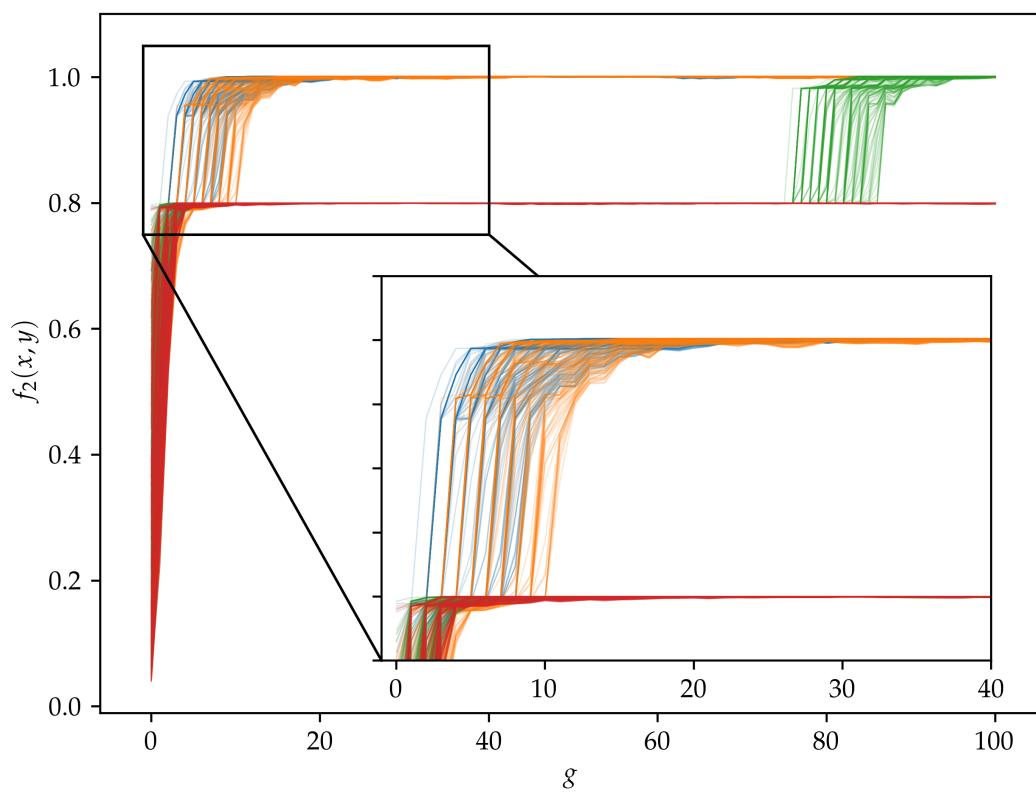


Figura 10 – Evolução dos valores da função $f_2(x, y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 5\%$.

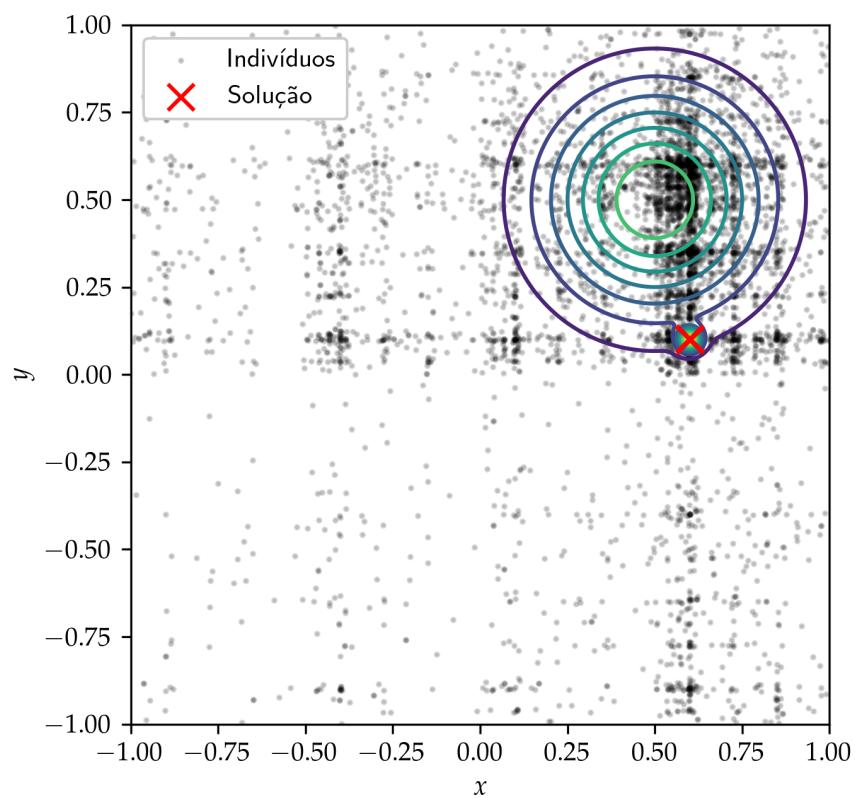


Figura 11 – Curvas de nível da função $f_2(x, y)$. Os pontos em preto indicam as posições dos indivíduos de 8 populações em sua 100^a geração na otimização da função, com $p_2 = p_3 = 20\%$. Marcado com um \times vermelho estão os melhores indivíduos de cada população.

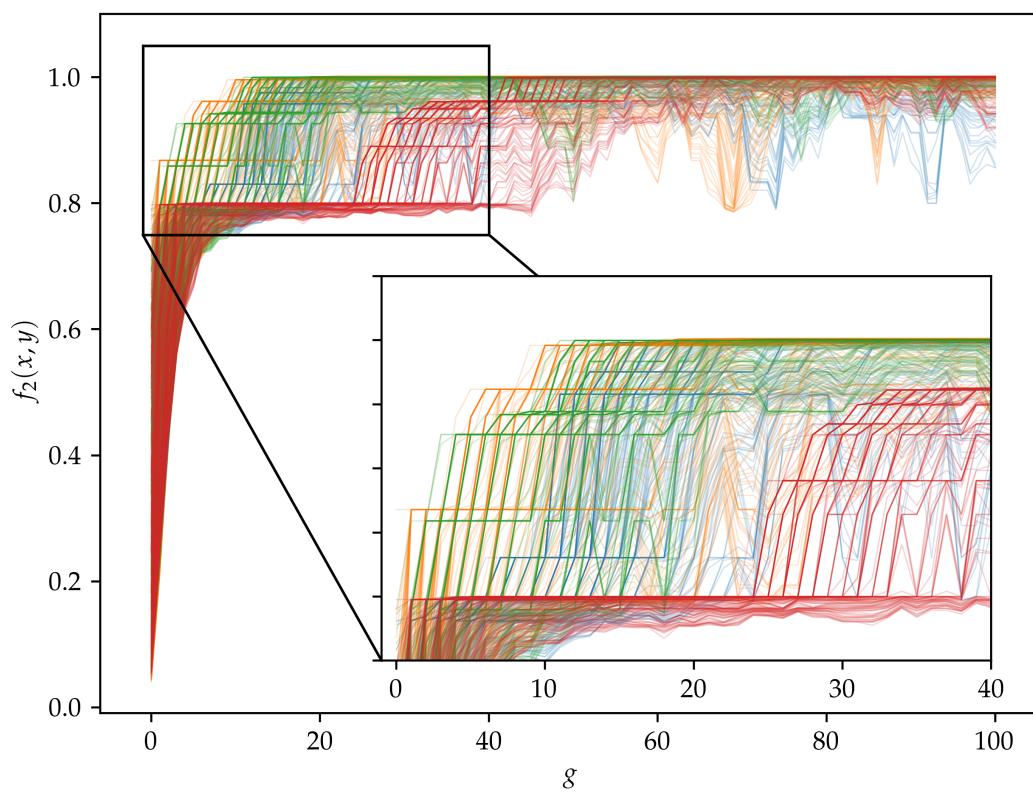


Figura 12 – Evolução dos valores da função $f_2(x,y)$ para os melhores 200 indivíduos de cada população, diferenciadas por cor, em termos da geração g , com $p_2 = p_3 = 20\%$.

4 Conclusão e Perspectivas

Um algoritmo do tipo genético se mostrou eficaz na solução de problemas de otimização numérica, encontrando o máximo global com poucas iterações e mantendo uma varia, sendo uma estratégia muito superior a qualquer método do tipo escalada de morro.

Como pode ser observado, em todos os casos, os melhores indivíduos foram localizados no máximo global da função, bastando a escolha correta dos parâmetros para que a solução desejada fosse encontrada com precisão. Não obstante, uma variedade genética proporcional a probabilidade de mutação foi mantida, dada uma escolha correta de p_2 e p_3 para cada problema. Nos casos em que $p_2 = 20\%$, os indivíduos, ao final do processo, se encontravam espalhados por todo o espaço de busca.

Vale ressaltar porém que o valor necessário de p_2 e p_3 para que uma população tenha a distribuição desejada depende da função a ser otimizada, como pode ser visto comparando as Figuras (7) e (8) com as Figuras (11) e (12). Assim, a influência dos parâmetros p_2 , p_3 , e_1 , e_2 e e_3 no comportamento da população deve ser estudada em cada caso, afim de extrair do algoritmo o resultado desejado.

Outra vantagem do algoritmo é que os máximos locais também puderam ser encontrados. Isso pode ser observado especialmente na Figura (8), onde visivelmente há uma concentração de indivíduos em $f_1(x, y) \approx 0,74$ e $f_1(x, y) \approx 0,29$, que correspondem aos dois primeiros máximos locais. Algo similar pode ser observado na segunda função, nas primeiras gerações da Figura (12), com alguma concentração da população em vermelho e em laranja em $f_1(x, y) \approx 0,8$.

Claro que, mesmo nos casos em que não é possível inferir o máximo local da evolução de uma parcela da população — como ocorre na Figura (6) — uma análise estatística feita sobre os valores das funções desempenho dos indivíduos e suas respectivas posições ainda seria capaz de acusá-lo.

Em suma, o algoritmo desenvolvido produziu resultados favoráveis nos testes apresentados, e suas capacidades e limitações serão postos a prova em outros testes subsequentes e nas futuras etapas do trabalho, onde será feita sua aplicação no estudo acerca de algum sistema físico.

5 Cronograma

Neste capítulo são apresentadas as atividades realizadas até o momento da confecção deste trabalho, bem como as perspectivas para o próximo período.

- Período 2021/02 — Nesse período foi feita a leitura da bibliografia citada neste texto, e então foi implementada a primeira versão do algoritmo. Posteriormente, foi feita uma série de testes com diferentes funções, dentre as quais estão as citadas nesse trabalho. Finalmente, foi confeccionado este texto.
- Período 2022/01 — No período subsequente, planeja-se escolher um problema físico que envolva a otimização de funções, realizar uma consulta sobre referências bibliográficas relacionadas, fazer a leitura dos dois textos encontrados e aplicar o programa na solução do problema. Por fim, o Trabalho de Conclusão de Curso será elaborado acerca dos resultados obtidos e a defesa dessa tese será apresentada.

Referências

- 1 Wikipedia contributors. *Knapsack problem* — Wikipedia, The Free Encyclopedia. 2021. <https://en.wikipedia.org/w/index.php?title=Knapsack_problem&oldid=1060217808>. [Online; accessed 20-April-2022].
- 2 HOLLAND, J. H. Genetic algorithms. *Scientific American*, JSTOR, v. 267, n. 1, p. 66–73, 1992. Disponível em: <<http://papers.cumincad.org/data/works/att/7e68.content.pdf>>.
- 3 Wikipedia contributors. *John Henry Holland* — Wikipedia, The Free Encyclopedia. 2021. <https://en.wikipedia.org/w/index.php?title=John_Henry_Holland&oldid=1041115860>. [Online; accessed 20-April-2022].
- 4 HARRIS, C. R. et al. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, set. 2020. Disponível em: <<https://doi.org/10.1038/s41586-020-2649-2>>.
- 5 HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007.
- 6 RONCARATTI, L. F.; GARGANO, R.; SILVA, G. M. e. A genetic algorithm to build diatomic potentials. *Journal of Molecular Structure: THEOCHEM*, Elsevier, v. 769, n. 1-3, p. 47–51, 2006.
- 7 CHARBONNEAU, P. et al. An introduction to genetic algorithms for numerical optimization. *Boulder, Colorado*, 2002.