CEFET-MG Leopoldina
Curso: Engenharia de Computação
Disciplina: Computação Evolucionista
Nome: Bianca da Silva Pinto

Função de Rosenbrock - Otimização e Buscas

1. Introdução

Este relatório consiste em apresentar os resultados obtidos através das análises de otimização feitas utilizando a função de Rosenbrock. Para estudar a função e gerar os gráficos, usou-se a linguagem de programação Python junto com diversas bibliotecas. Os resultados foram obtidos através de uma busca sequencial e dez buscas aleatórias.

A função de Rosenbrock também é conhecida como função banana ou vale, de n-dimensões e é bem conhecida por ser utilizada em algoritmos de otimização baseados em gradiente. Possui o mínimo global em um vale parabólico, fácil de ser encontrado, porém de difícil convergência. A função é apresentada da seguinte forma:

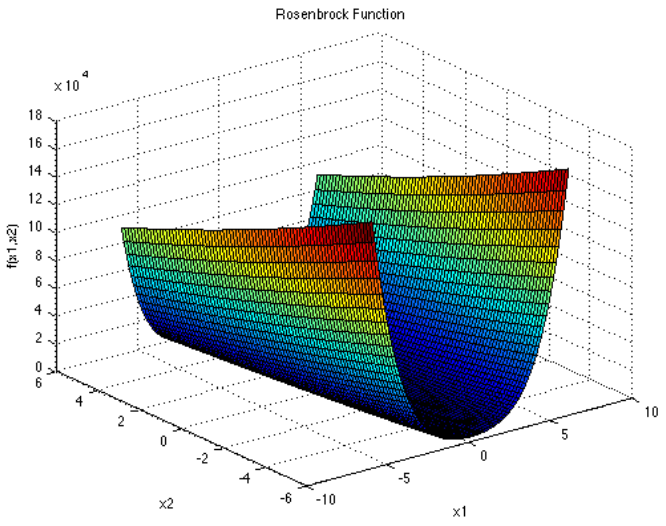
$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

O seu mínimo global é:

$$f(\mathbf{x}^*) = 0, \text{ at } \mathbf{x}^* = (1, \ldots, 1)$$

A função de Rosenbrock (Figura 1) pode ser analisada em dois intervalos distintos no hipercubo, $x_i \in [-5, 10]$ e $x_i \in [-2.048, 2.048]$, para todo i = 1, …, d.

Figura 1 - Função de Rosenbrock



Disponível em: https://www.sfu.ca/~ssurjano/rosen.html. Acesso em: 30 oct 2024.

2. Algoritmos

Os algoritmos usados para o estudo foram fornecidos previamente e era preciso adaptá-los para gerar os resultados buscados. A função de Rosenbrock foi representada em Python em *rosenbrock(args)*, sendo *args* os parâmetros referentes às variáveis das dimensões adotadas. Para gerar os gráficos, adotou-se duas dimensões e por isso, x e y.

Algoritmo 1 - Implementação da função de Rosenbrock

```
import random
import numpy as np
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import pandas as pd
import time

def funcao_rosenbrock(x, y, a=1, b=100):
    """
    Função de Rosenbrock para dois parâmetros.
    """
    return (a - x)**2 + b * (y - x**2)**2
```

A busca sequencial consiste em buscar os melhores valores dentro dos intervalos fornecidos e pelo tamanho do passo. Para duas dimensões, inclui-se x e y para buscar seus melhores valores.

Algoritmo 2 - Implementação da busca sequencial

```
def busca_sequencial(tamanho_passo, limite_inferior, limite_superior):
    melhor_x=None
    melhor_y=None
    melhor_valor = np.inf

    melhores_dominios=[]
    melhores_imagens =[]
    dominio_x=[]
    dominio_y=[]
    imagem=[]

        for x in drange(limite_inferior,limite_superior+tamanho_passo,
tamanho_passo):
                for y in drange(limite_inferior, limite_superior +
tamanho_passo, tamanho_passo):
            valor = rosenbrock((x,y))
            if melhor_valor > valor:
                melhor_valor = valor
                melhor_x=x
```

```
                melhor_y=y

        melhores_dominios.append((melhor_x, melhor_y))
        melhores_imagens.append(melhor_valor)
        dominio_x.append(x)
        dominio_y.append(y)
        imagem.append(valor)

    return melhor_x,melhor_y,melhor_valor,dominio_x,dominio_y, imagem,
melhores_dominios, melhores_imagens
```

Já a busca aleatória procura encontrar os melhores valores de forma aleatória, ou seja, as variáveis adotam um valor qualquer dentro do intervalo (chute inicial) e depois é decidido se aquele valor é melhor do que a encontrada anteriormente.

Algoritmo 3 - Implementação da busca aleatória

```
def busca_aleatoria(quantidade_iteracoes, limite_inferior,
limite_superior):
    pontos_x = []
    pontos_y = []
    pontos_valor = []

    melhor_x = None
    melhor_y = None
    melhor_valor = np.inf
    pior_x = None
    pior_y = None
    pior_valor = -np.inf

    for _ in range(quantidade_iteracoes):
        x = random.uniform(limite_inferior, limite_superior)
        y = random.uniform(limite_inferior, limite_superior)
        valor = funcao_rosenbrock(x, y)

        pontos_x.append(x)
        pontos_y.append(y)
        pontos_valor.append(valor)

        # Verifica e atualiza o melhor e pior ponto
        if valor < melhor_valor:
            melhor_valor = valor
            melhor_x = x
            melhor_y = y
```

```
        if valor > pior_valor:
            pior_valor = valor
            pior_x = x
            pior_y = y

        return melhor_x, melhor_y, melhor_valor, pior_x, pior_y,
pior_valor, pontos_x, pontos_y, pontos_valor
```

O código abaixo realiza 10 buscas aleatórias independentes para encontrar os melhores valores, piores, média, mediana e desvio padrão. A cada vez que roda o código inteiro, valores diferentes são encontrados e portanto os resultados sempre serão distintos. Como a função de Rosenbrock apresenta duas formas de restringir o domínio (*limite_inferior* e *limite_superior*), duas tabelas de resultados serão apresentadas.

Algoritmo 4 - Implementação para encontrar melhores e piores valores, além de outros dados estatísticos

```
# Parâmetros da busca aleatória e intervalo de amostragem
quantidade_iteracoes = 1000
limite_inferior = -5
#limite_inferior = -2.048
limite_superior = 10
#limite_superior = 2.048
execucoes = 10

# Armazenamento dos resultados de todas as execuções
resultados = []
todos_pontos_x = []
todos_pontos_y = []
todos_pontos_valor = []
melhores_pontos_x = []
melhores_pontos_y = []
melhores_pontos_valor = []

# Executa a busca aleatória e armazena dados de cada execução
for i in range(execucoes):
    start_time = time.time()
       melhor_x, melhor_y, melhor_valor, pior_x, pior_y, pior_valor,
pontos_x, pontos_y, pontos_valor = busca_aleatoria(
        quantidade_iteracoes, limite_inferior, limite_superior
    )
    exec_time = time.time() - start_time

    todos_pontos_x.extend(pontos_x)
    todos_pontos_y.extend(pontos_y)
```

```
    todos_pontos_valor.extend(pontos_valor)
    melhores_pontos_x.append(melhor_x)
    melhores_pontos_y.append(melhor_y)
    melhores_pontos_valor.append(melhor_valor)

    # Cálculo do desvio padrão para os valores de cada execução
    desvio_padrao = np.std(pontos_valor)

    resultados.append({
        'Execução': i + 1,
        'Melhor X': melhor_x,
        'Melhor Y': melhor_y,
        'Melhor Valor': melhor_valor,
        'Pior X': pior_x,
        'Pior Y': pior_y,
        'Pior Valor': pior_valor,
        'Tempo (s)': exec_time,
        'Desvio Padrão': desvio_padrao
    })

# Cálculo das estatísticas finais
df_resultados = pd.DataFrame(resultados)

# Execução das buscas aleatórias e armazenamento dos resultados
resultados = []

# Cálculo das estatísticas e dos valores mínimos e máximos
estatisticas = {
    "Estatísticas": ["Melhor Valor", "Pior Valor", "Média dos Valores",
"Mediana dos Valores", "Desvio Padrão dos Valores", "Tempo Médio (s)",
"Tempo Total (s)",
                     "Melhor X", "Pior X", "Melhor Y", "Pior Y"],
    "Valores": [
        df_resultados["Melhor Valor"].min(),
        df_resultados["Pior Valor"].max(),
        df_resultados["Melhor Valor"].mean(),
        df_resultados["Melhor Valor"].median(),
        df_resultados["Melhor Valor"].std(),
        df_resultados["Tempo (s)"].mean(),
        df_resultados["Tempo (s)"].sum(),
        df_resultados["Melhor X"].min(),
        df_resultados["Pior X"].max(),
        df_resultados["Melhor Y"].min(),
```

```
        df_resultados["Pior Y"].max()
    ]
}

# Arredondando todos os valores para 6 casas decimais
df_resultados = df_resultados.round({
    'Melhor X': 6,
    'Melhor Y': 6,
    'Melhor Valor': 6,
    'Pior X': 6,
    'Pior Y': 6,
    'Pior Valor': 6,
    'Tempo (s)': 6,
    'Desvio Padrão': 6
})

# Exibe as estatísticas de cada execução
print(df_resultados[['Execução', 'Melhor X', 'Melhor Y', 'Melhor Valor', 'Pior X', 'Pior Y', 'Pior Valor', 'Tempo (s)', 'Desvio Padrão']])

# Tabela final com estatísticas
df_estatisticas = pd.DataFrame(estatisticas)

# Exibindo os resultados
print("\nEstatísticas gerais:")
print(df_estatisticas)
```

3. Resultados

Para o domínio de [-5,10], os resultados foram organizados na Tabela 1.1 e 1.2, e para o domínio de [-2.048, 2.048], os resultados estão na Tabela 2.1 e 2.2. Foram calculadas a mediana, a média, o tempo médio e o tempo total para cada domínio.

Tabela 1.1 - Resultados para a busca aleatória no domínio [-5,10]

Execução	Melhor X	Melhor Y	Melhor Valor	Pior X	Pior Y	Pior Valor	Tempo (s)	Desvio Padrão
1	0.632547	0.398304	0.13535	9.980511	-3.121112	.055461.17569	0.001992	224901.621835
2	0.799637	0.700529	0.413583	9.956331	-4.433288	.072585.25736	0.001996	231367.345853
3	0.647665	0.458527	0.276685	9.995593	-3.354682	.066479.01100	0.002007	229692.774824
4	0.037318	-0.000642	0.927171	9.86771	-4.396739	.035760.11171	0.001985	228518.390702
5	0.208805	-0.025816	1.107847	9.963188	-4.367406	.074050.20656	0.001986	221173.857568
6	0.756475	0.660976	0.846467	9.873259	-3.152307	.012789.93026	0.001997	218996.440539
7	0.883501	0.753902	0.084712	9.990654	-4.317121	.084392.65706	0.001993	222467.315789
8	0.997773	1.042951	0.22469	9.984236	-4.300865	1081386.1071	0.001994	234145.460815
9	0.045067	0.035557	1.024294	9.999312	-4.504178	1091905.92375	0.000998	221976.582303
10	0.041914	0.075174	1.456934	9.985611	-3.460545	.064547.06154	0.001994	222321.186853

Tabela 1.2 - Dados destacados da Tabela 1.1 e dados estatísticos

Estatística	Valor
Melhor Valor	0.084712
Pior Valor	1091905.92375
Média dos Valores	0.6497733
Mediana dos Valores	0.630025
Desvio Padrão dos Valores	0.48037551448610377
Tempo Médio (s)	0.0018942
Tempo Total (s)	0.018942
Melhor X	0.037318

Tabela 2.1 - Resultados para a busca aleatória no domínio [-2.048, 2.048]

Execução	Melhor X	Melhor Y	Melhor Valor	Pior X	Pior Y	Pior Valor	Tempo (s)	Desvio Padrão
1	1.124469	1.258347	0.019194	2.017956	-1.960222	3639.984628	0.000989	627.566202
2	1.17345	1.361779	0.053207	1.995986	-1.999687	3581.3974	0.001995	635.823852
3	0.876166	0.766868	0.015399	1.959678	-1.998766	3410.434569	0.001994	673.589261
4	1.056341	1.094463	0.048946	-2.014254	-1.979018	3652.701	0.002991	636.531861
5	1.036521	1.061224	0.018632	-2.028199	-2.013727	3763.572503	0.000999	670.553311
6	1.016211	1.044097	0.013284	-2.019255	-1.898746	3580.535926	0.000998	586.437787
7	0.866912	0.752875	0.017892	-2.040394	-1.900488	3686.083434	0.001001	695.236865
8	1.063678	1.14483	0.022065	2.012721	-2.016378	3682.387012	0.00101	644.826766
9	1.194747	1.466181	0.188168	-2.040396	-1.974105	3775.912875	0.000997	686.515929
10	1.033615	1.081965	0.019637	-2.015783	-1.915021	3583.223797	0.001996	663.211339

Tabela 2.2 - Dados destacados da Tabela 2.1 e dados estatísticos

Estatística	Valor
Melhor Valor	0.013284
Pior Valor	3775.912875
Média dos Valores	0.0416424
Mediana dos Valores	0.019415500000000002
Desvio Padrão dos Valores	0.05334869756444128
Tempo Médio (s)	0.0014969999999999998
Tempo Total (s)	0.014969999999999999
Melhor X	0.866912

Foram gerados os gráficos para a busca sequencial no domínio [-5,10] (Figura 2) e no domínio [-2.048,2.048] (Figura 3). Os gráficos da busca aleatória (Figuras 4, 5, 6 e 7) foram gerados com os pontos encontrados durante a execução do código.

Figura 2 - Gráfico para a busca sequencial no domínio [-5,10]

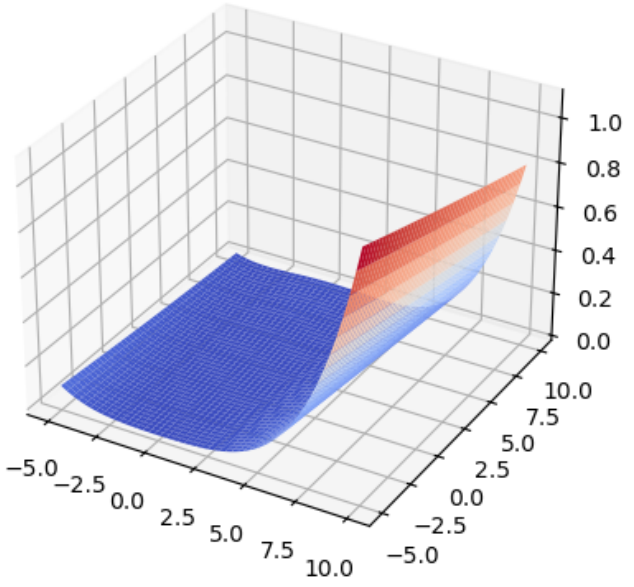


Figura 3 - Gráfico para a busca sequencial no domínio [-2.048, 2.048]

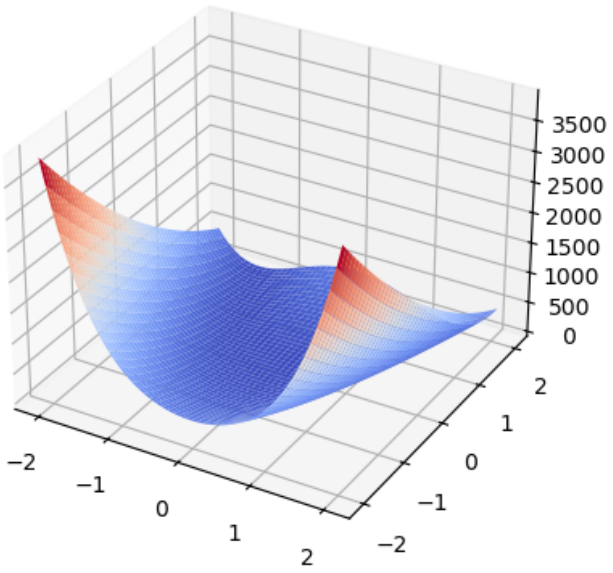


Figura 4 - Gráfico para a busca aleatória no domínio [-5, 10]

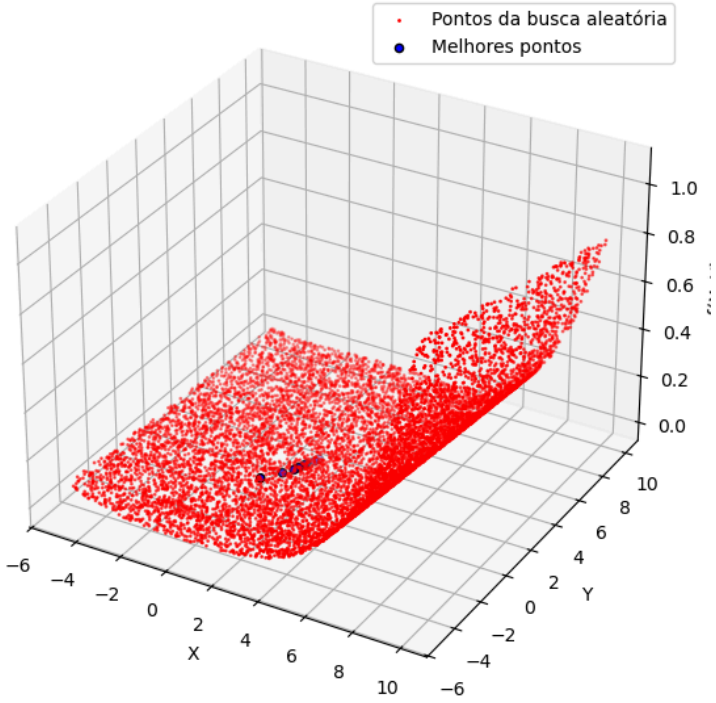


Figura 5 - Gráfico para a busca aleatória no domínio [-2.048, 2.048]

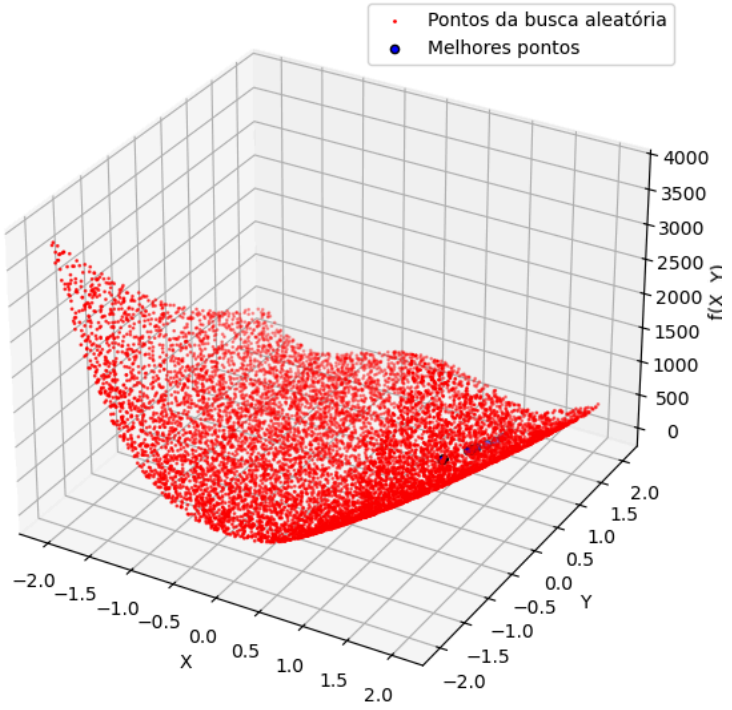


Figura 6 - Gráfico para a busca aleatória (3D) no domínio [-5, 10]

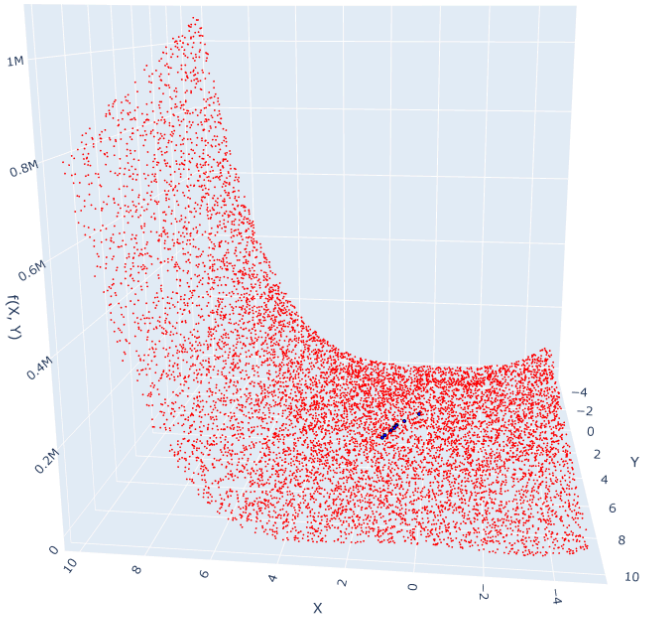
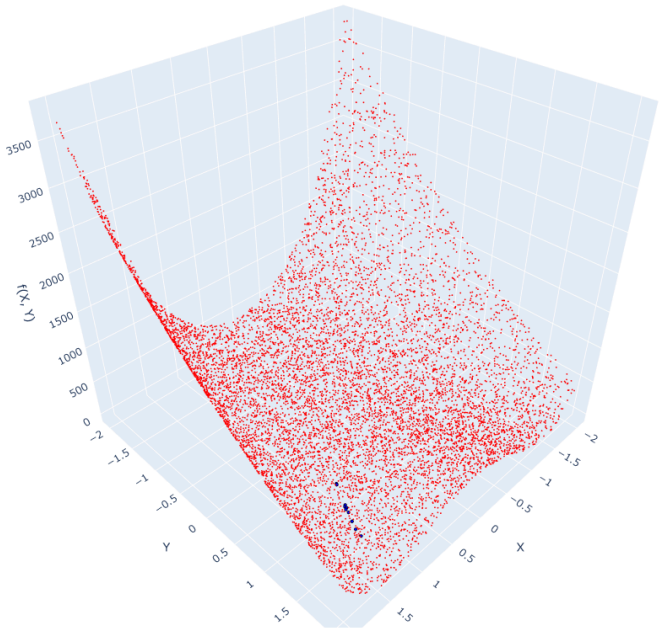


Figura 7 - Gráfico para a busca aleatória (3D) no domínio [-2.048, 2.048]



4. Discussão dos Resultados

Os valores do desvio padrão e da média foram muito maiores ao utilizar um domínio maior do que o menor. Pelo fato de que os valores de x e y podem oscilar entre os extremos de um domínio, na busca aleatória é possível encontrar valores ainda piores no domínio [-5, 10]. Apesar disso, mesmo quando executamos o mesmo código inúmeras vezes, consegue encontrar um melhor valor, que tende a ser mais próximo de 0.

5. Conclusão

Apesar da função de Rosenbrock poder ser avaliada em dois domínios distintos, os resultados melhores foram encontrados no domínio mais restrito. Como são menos valores a serem analisados, fica mais fácil encontrar os pontos dos extremos e o valor do mínimo ser próximo do esperado. Os resultados ficam nitidamente visíveis quando os gráficos e os dados estatísticos são comparados. Para o domínio mais restrito, os valores da média, mediana e desvio padrão foram menores e mais satisfatórios.

Seria possível obter mais resultados pela análise da função com mais dimensões e testar a sua forma modificada de Picheny, com d = 4.

6. Referências

Rosenbrock Function. Disponível em: https://www.sfu.ca/~ssurjano/rosen.html. Acesso em: 30 oct 2024.