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

Função de Rosenbrock - Otimização e Buscas

Introdução

O presente 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

A função 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:

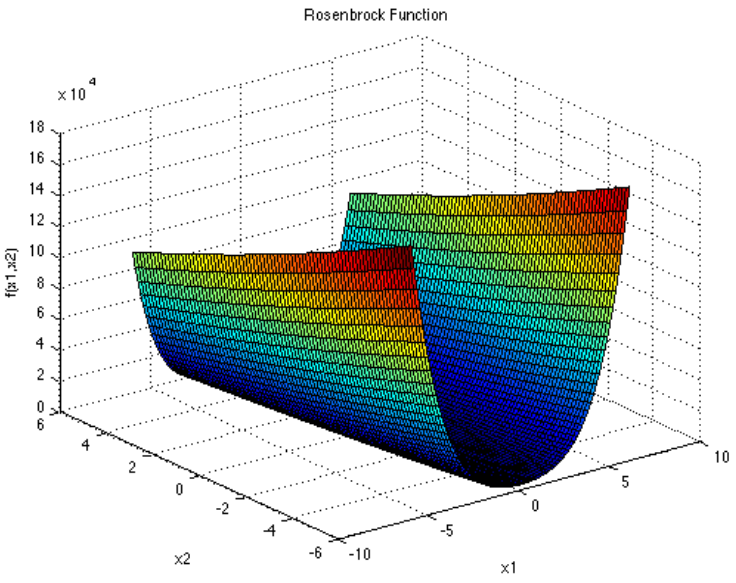
$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

O seu mínimo global é:

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

A função de Rosenbrock pode ser analisada em dois intervalos distintos, $x_i \in [-5, 10]$ e $x_i \in [-2.048, 2.048]$, para todo i = 1, …, d.

O gráfico da função é desta forma:



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.

```
import random
import matplotlib.pyplot as plt
import numpy as np

import plotly.graph_objects as go
#função para iterar com um passo decimal
def drange(start, stop, step):
     r = start
     while r < stop:
         yield r
         r += step
def funcao_objetivo(x):
    return (x-10)**2+1
def funcao_objetivo_seno(x):
    return np.sin(x)
def rosenbrock(args):
    x,y=args
    return (1 - x)**2 + 100*(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.

```
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.

```
def busca_aleatoria(quantidade_iteracoes, limite_inferior,
limite_superior):
    melhor_x=None
    melhor_y=None
    melhor_valor = np.inf
    # melhor_z=None

    melhores_dominios=[]
    # melhores_dominiosZ=[]
    melhores_imagens =[]
    dominio_x=[]
    dominio_y=[]
    # dominioZ=[]
    imagem=[]

    for _ in range(quantidade_iteracoes):
        x=random.uniform(limite_inferior,limite_superior)
        y=random.uniform(limite_inferior,limite_superior)
        valor = rosenbrock((x,y))
        # z=random.uniform(limite_inferior,limite_superior)
        # y= ackley(x,z)
        # y=funcao_objetivo(x)
        if melhor_valor > valor:
            melhor_valor = valor
            melhor_x = x
```

```
            melhor_y = y

        melhores_dominios.append((melhor_x, melhor_y))
        # melhores_dominiosZ.append(melhor_z)
        melhores_imagens.append(melhor_valor)

        dominio_x.append(x)
        dominio_y.append(y)
        # dominioZ.append(z)
        imagem.append(valor)

      #  return  melhor_x,melhor_z,melhor_y,  dominio,dominioZ,  imagem,
melhores_dominios,melhores_dominiosZ, melhores_imagens
    return melhor_x,melhor_y, melhor_valor,dominio_x,dominio_y, imagem,
melhores_dominios, melhores_imagens
```

Resultados para 2 dimensões

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.

```
import time
import numpy as np
import pandas as pd

# Parâmetros
quantidade_iteracoes = 60
limite_inferior = -5
limite_superior = 10
num_execucoes = 10

# Listas para armazenar os melhores valores e tempos de cada execução
melhores_valores = []
tempos_execucao = []

for i in range(num_execucoes):
    # Medir o tempo de execução
    inicio_tempo = time.time()

    # Executar a busca aleatória
            x,   y,   melhor_valor,   dominio_x,   dominio_y,   imagem,
melhores_dominios, melhores_imagens = busca_aleatoria(
        quantidade_iteracoes, limite_inferior, limite_superior
```

```
    )

    # Calcular o tempo de execução
    tempo_execucao = time.time() - inicio_tempo

    # Armazenar o melhor valor e o tempo da execução atual
    melhores_valores.append(melhor_valor)
    tempos_execucao.append(tempo_execucao)
      print(f"Execução {i+1}: Melhor valor = {melhor_valor}, Tempo =
{tempo_execucao:.4f} segundos")

# Cálculo das estatísticas dos melhores valores
melhor_resultado = np.min(melhores_valores)
pior_resultado = np.max(melhores_valores)
media_resultado = np.mean(melhores_valores)
mediana_resultado = np.median(melhores_valores)
desvio_padrao_resultado = np.std(melhores_valores)

# Cálculo das estatísticas de tempo
tempo_medio = np.mean(tempos_execucao)
tempo_mediana = np.median(tempos_execucao)
tempo_desvio_padrao = np.std(tempos_execucao)

# Criar DataFrame com os resultados de cada execução
df_execucoes = pd.DataFrame({
    'Execução': list(range(1, num_execucoes + 1)),
    'Melhor Valor': melhores_valores,
    'Tempo de Execução (s)': tempos_execucao
})

# Adicionar uma linha final com as estatísticas agregadas em colunas
separadas
df_estatisticas = pd.DataFrame({
    'Execução': ['Estatísticas'],
    'Melhor Valor': [melhor_resultado],
     'Tempo de Execução (s)': [np.nan],  # Deixar em branco para o tempo
nesta linha
    'Pior Valor': [pior_resultado],
    'Média dos Valores': [media_resultado],
    'Mediana dos Valores': [mediana_resultado],
    'Desvio Padrão dos Valores': [desvio_padrao_resultado],
    'Tempo Médio (s)': [tempo_medio],
    'Tempo Mediana (s)': [tempo_mediana],
```

```
    'Tempo Desvio Padrão (s)': [tempo_desvio_padrao]
})

# Concatenar os DataFrames, alinhando as colunas
df_final = pd.concat([df_execucoes, df_estatisticas],
ignore_index=True)

# Exibir a tabela completa com preenchimento de NaNs
print("\nResultados das Execuções Individuais e Estatísticas
Agregadas:")
print(df_final.fillna('N/A'))
```

Para o domínio de [-5,10], os seguintes resultados foram encontrados:

```
Execução 1: Melhor valor = 24.354362277290704, Tempo = 0.0010 segundos
Execução 2: Melhor valor = 14.854363416635527, Tempo = 0.0000 segundos
Execução 3: Melhor valor = 32.814013923323806, Tempo = 0.0000 segundos
Execução 4: Melhor valor = 11.800025166081133, Tempo = 0.0000 segundos
Execução 5: Melhor valor = 19.654467147579304, Tempo = 0.0000 segundos
Execução 6: Melhor valor = 34.01605676268375, Tempo = 0.0000 segundos
Execução 7: Melhor valor = 24.70922220829673, Tempo = 0.0000 segundos
Execução 8: Melhor valor = 43.061149611280506, Tempo = 0.0000 segundos
Execução 9: Melhor valor = 8.418098977704012, Tempo = 0.0015 segundos
Execução 10: Melhor valor = 20.38677153103563, Tempo = 0.0000 segundos

Resultados das Execuções Individuais e Estatísticas Agregadas:
        Execução  Melhor Valor Tempo de Execução (s) Pior Valor  \
0             1     24.354362              0.000987        N/A
1             2     14.854363                   0.0        N/A
2             3     32.814014                   0.0        N/A
3             4     11.800025                   0.0        N/A
4             5     19.654467                   0.0        N/A
5             6     34.016057                   0.0        N/A
6             7     24.709222                   0.0        N/A
7             8     43.061150                   0.0        N/A
8             9      8.418099               0.00151        N/A
9            10     20.386772                   0.0        N/A
10  Estatísticas      8.418099                   N/A   43.06115

   Média dos Valores Mediana dos Valores Desvio Padrão dos Valores  \
0                N/A                 N/A                       N/A
1                N/A                 N/A                       N/A
2                N/A                 N/A                       N/A
3                N/A                 N/A                       N/A
4                N/A                 N/A                       N/A
5                N/A                 N/A                       N/A
6                N/A                 N/A                       N/A
7                N/A                 N/A                       N/A
8                N/A                 N/A                       N/A
9                N/A                 N/A                       N/A
```

```
10         23.406853           22.370567                 10.225765

   Tempo Médio (s) Tempo Mediana (s) Tempo Desvio Padrão (s)
0              N/A               N/A                     N/A
1              N/A               N/A                     N/A
2              N/A               N/A                     N/A
3              N/A               N/A                     N/A
4              N/A               N/A                     N/A
5              N/A               N/A                     N/A
6              N/A               N/A                     N/A
7              N/A               N/A                     N/A
8              N/A               N/A                     N/A
9              N/A               N/A                     N/A
10         0.00025               0.0                0.000513
```

Para o domínio [-2.048,2.048], os seguintes resultados foram encontrados:

```
Execução 1: Melhor valor = 0.12294738824162921, Tempo = 0.0000 segundos
Execução 2: Melhor valor = 1.1748602082881057, Tempo = 0.0000 segundos
Execução 3: Melhor valor = 1.8560791888466237, Tempo = 0.0000 segundos
Execução 4: Melhor valor = 2.331296833475802, Tempo = 0.0000 segundos
Execução 5: Melhor valor = 1.7234005261226213, Tempo = 0.0010 segundos
Execução 6: Melhor valor = 0.1269732432517869, Tempo = 0.0000 segundos
Execução 7: Melhor valor = 0.1573412493876463, Tempo = 0.0000 segundos
Execução 8: Melhor valor = 0.537477647047272, Tempo = 0.0000 segundos
Execução 9: Melhor valor = 0.6185216536059633, Tempo = 0.0000 segundos
Execução 10: Melhor valor = 0.8965965118285681, Tempo = 0.0000 segundos

Resultados das Execuções Individuais e Estatísticas Agregadas:
        Execução  Melhor Valor Tempo de Execução (s) Pior Valor  \
0              1      0.122947                   0.0        N/A
1              2      1.174860                   0.0        N/A
2              3      1.856079                   0.0        N/A
3              4      2.331297                   0.0        N/A
4              5      1.723401              0.000994        N/A
5              6      0.126973                   0.0        N/A
6              7      0.157341                   0.0        N/A
7              8      0.537478                   0.0        N/A
8              9      0.618522                   0.0        N/A
9             10      0.896597                   0.0        N/A
10  Estatísticas      0.122947                   N/A   2.331297

   Média dos Valores Mediana dos Valores Desvio Padrão dos Valores  \
0                N/A                 N/A                       N/A
1                N/A                 N/A                       N/A
2                N/A                 N/A                       N/A
3                N/A                 N/A                       N/A
4                N/A                 N/A                       N/A
5                N/A                 N/A                       N/A
6                N/A                 N/A                       N/A
7                N/A                 N/A                       N/A
8                N/A                 N/A                       N/A
```

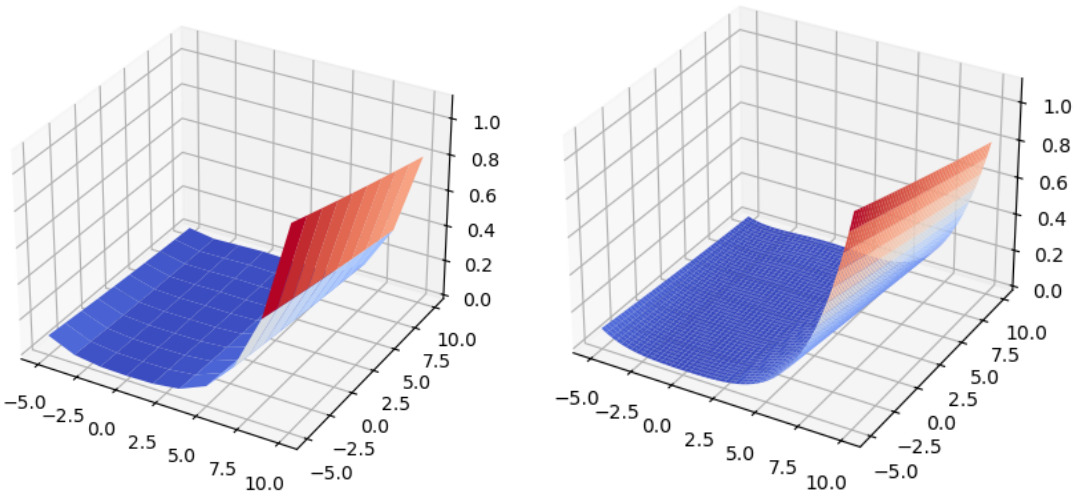
```
9                N/A                  N/A                       N/A
10          0.954549             0.757559                  0.751669
```

```
   Tempo Médio (s) Tempo Mediana (s) Tempo Desvio Padrão (s)
0              N/A                N/A                     N/A
1              N/A                N/A                     N/A
2              N/A                N/A                     N/A
3              N/A                N/A                     N/A
4              N/A                N/A                     N/A
5              N/A                N/A                     N/A
6              N/A                N/A                     N/A
7              N/A                N/A                     N/A
8              N/A                N/A                     N/A
9              N/A                N/A                     N/A
10        0.000099                0.0                0.000298
```

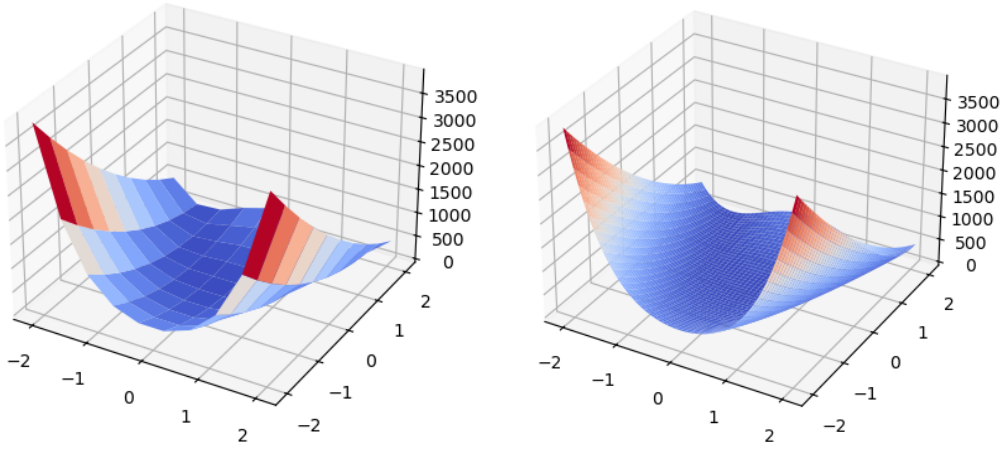
Para a geração dos gráficos, também levou em consideração os domínios distintos. Para fins de visualização, adotou-se 10 pontos e 500 pontos.

```python
#Dados para plot do gráfico 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

x1_v = np.linspace(-2.048,2.048,500)
x2_v = np.linspace(-2.048,2.048,500)
x1_arr, x2_arr = np.meshgrid(x1_v,x2_v)

f_arr = [[rosenbrock([x1_,x2_]) for x1_ in x1_v] for x2_ in x2_v]
f_arr = np.array(f_arr)

# ax=plt.figure().add_subplot(projection='3d')
surf =ax.plot_surface(x1_arr,x2_arr,f_arr,cmap=plt.cm.coolwarm)
```

Domínio [-5,10]:



Domínio [-2.048,2.048]:



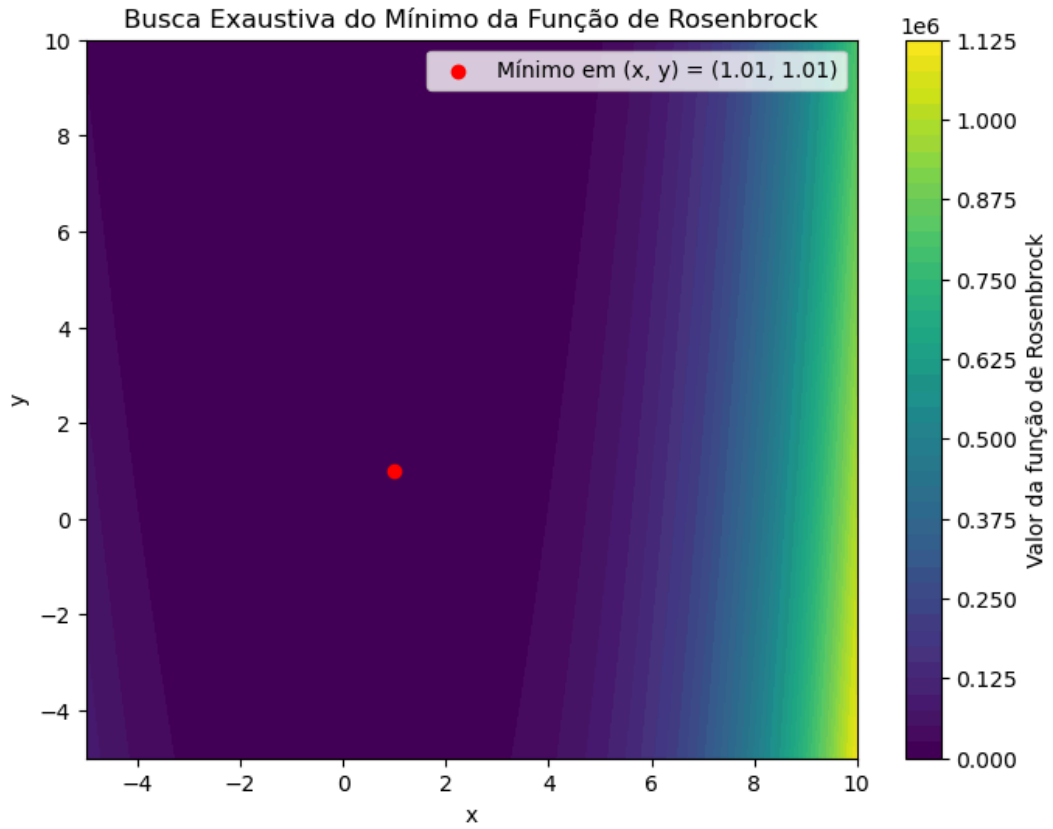
A busca exaustiva encontra o mínimo da função dentro de um determinado intervalo. Para fins de teste, a função de Rosenbrock também foi verificada no código com os dois domínios. Foram gerados gráficos de níveis e lineares (y fixo em 1).

```
def funcao_rosenbrock(x, y, a=1, b=100):
    """
    Função de Rosenbrock.

    Parâmetros:
    x (float): Ponto no eixo x.
    y (float): Ponto no eixo y.
    a (float): Parâmetro da função de Rosenbrock (padrão é 1).
    b (float): Parâmetro da função de Rosenbrock (padrão é 100).

    Retorna:
    float: Valor da função de Rosenbrock para os pontos (x, y).
    """
    return (a - x)**2 + b * (y - x**2)**2


def busca_exaustiva(a, b, passos):
    """
    Realiza uma busca exaustiva para encontrar o mínimo de uma função f
no intervalo [a, b].

    Parâmetros:
    f (callable): Função a ser minimizada.
    a (float): Limite inferior do intervalo.
    b (float): Limite superior do intervalo.
    passos (int): Número de pontos a serem avaliados no intervalo.
```

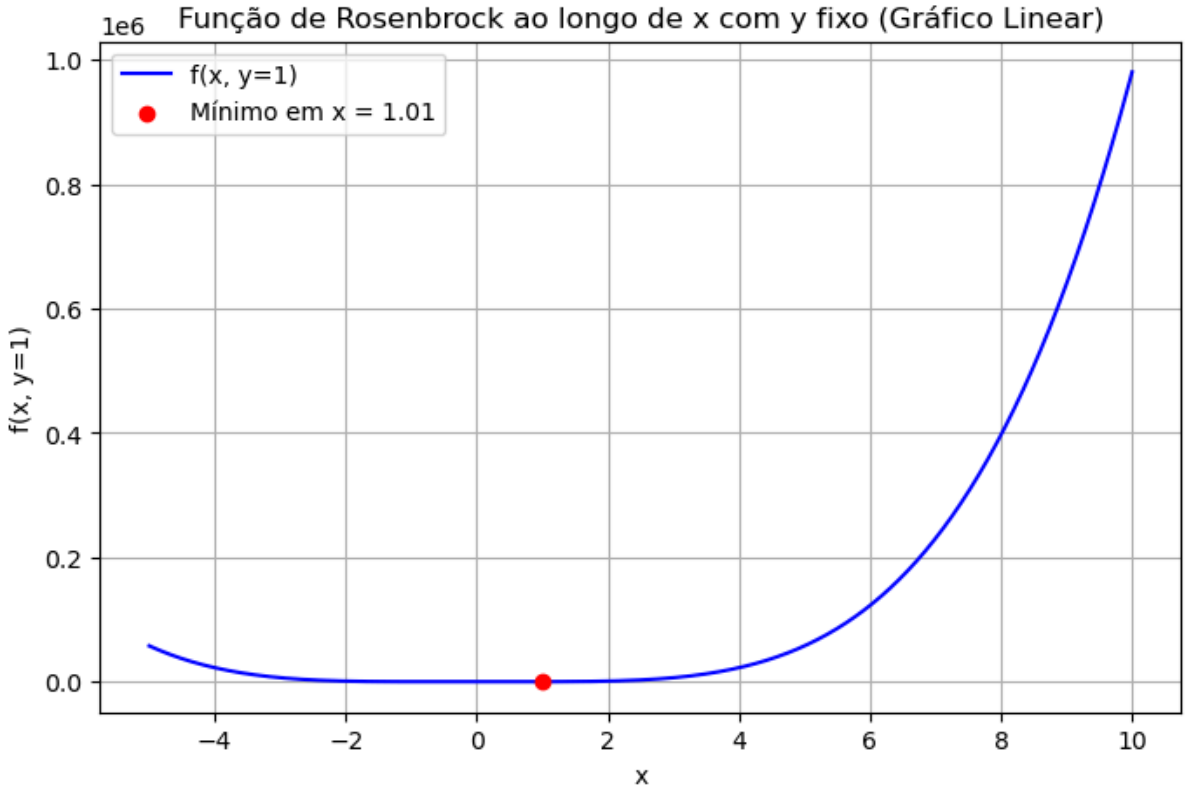
```
    Retorna:
    float: Ponto onde a função f tem o menor valor.
    float: Valor mínimo da função.
    """
     # Cria um vetor de pontos uniformemente espaçados no intervalo [a,
b]
    x = np.linspace(a, b, passos)
    y = np.linspace(a, b, passos)
    x_grid, y_grid = np.meshgrid(x,y)

    # Avalia a função em cada ponto
    valores = funcao_rosenbrock(x_grid,y_grid)

    # Encontra o índice do valor mínimo
    idx_min = np.unravel_index(np.argmin(valores), valores.shape)
    ponto_minimo_x = x_grid[idx_min]
    ponto_minimo_y = y_grid[idx_min]
    valor_minimo = valores[idx_min]

    # Retorna o ponto de mínimo e o valor mínimo
       return  (ponto_minimo_x,  ponto_minimo_y),  valor_minimo,  x_grid,
y_grid, valores

# Exemplo de uso
# Definindo uma função exemplo f(x) = (x - 2) ** 2

# Intervalo de busca [a, b] e número de passos
limite_inferior = -5
limite_superior = 10
passos = 1000

# Encontrando o mínimo
ponto_minimo, valor_minimo, x_grid, y_grid, valores = busca_exaustiva(
    limite_inferior, limite_superior, passos
)


# Plotando a função e o ponto de mínimo
plt.figure(figsize=(8, 6))
plt.contourf(x_grid, y_grid, valores, levels=50, cmap='viridis')
plt.colorbar(label='Valor da função de Rosenbrock')
```

```
plt.scatter(*ponto_minimo, color='red', label=f'Mínimo em (x, y) =
({ponto_minimo[0]:.2f}, {ponto_minimo[1]:.2f})')
plt.title('Busca Exaustiva do Mínimo da Função de Rosenbrock')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

print(f'O mínimo da função ocorre em (x, y) = {ponto_minimo}')
print(f'O valor mínimo da função é f(x, y) = {valor_minimo}')

# Criação dos pontos no eixo x e avaliação da função com y fixo
x = np.linspace(limite_inferior, limite_superior, passos)
valores = funcao_rosenbrock(x, y=1)  # Mantendo y fixo em 1

# Encontrando o índice do valor mínimo
idx_min = np.argmin(valores)
ponto_minimo_x = x[idx_min]
valor_minimo = valores[idx_min]

# Plotando o gráfico linear
plt.figure(figsize=(8, 5))
plt.plot(x, valores, label='f(x, y=1)', color='blue')
plt.scatter(ponto_minimo_x, valor_minimo, color='red', label=f'Mínimo
em x = {ponto_minimo_x:.2f}', zorder=5)
plt.title('Função de Rosenbrock ao longo de x com y fixo (Gráfico
Linear)')
plt.xlabel('x')
plt.ylabel('f(x, y=1)')
plt.grid(True)
plt.legend()
plt.show()

print(f'O mínimo da função ocorre em x = {ponto_minimo_x}')
print(f'O valor mínimo da função é f(x, y=1) = {valor_minimo}')
```

Pelo domínio [-5,10], encontrou-se os seguintes resultados:

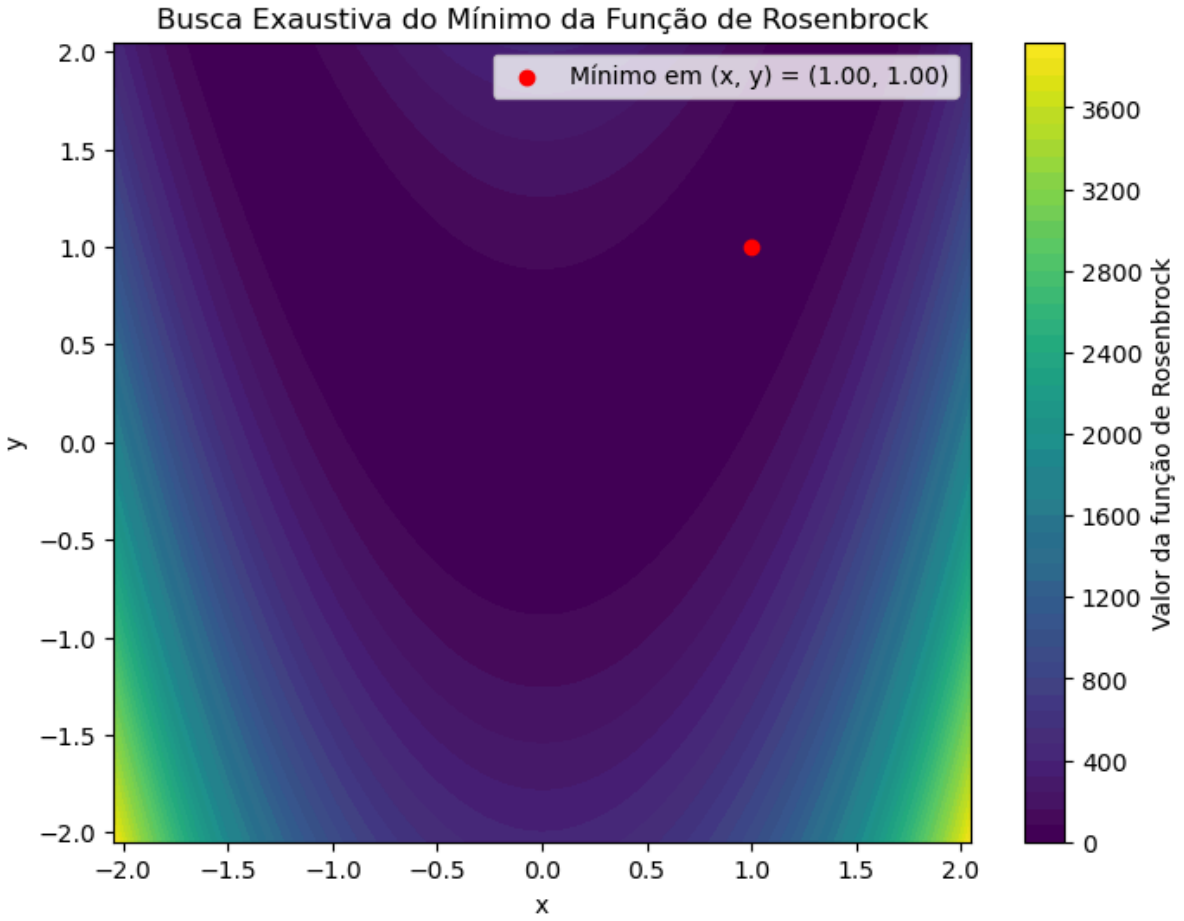


O mínimo da função ocorre em (x, y) = (1.0060060060060056, 1.0060060060060056)
O valor mínimo da função é f(x, y) = 0.003686742901893119

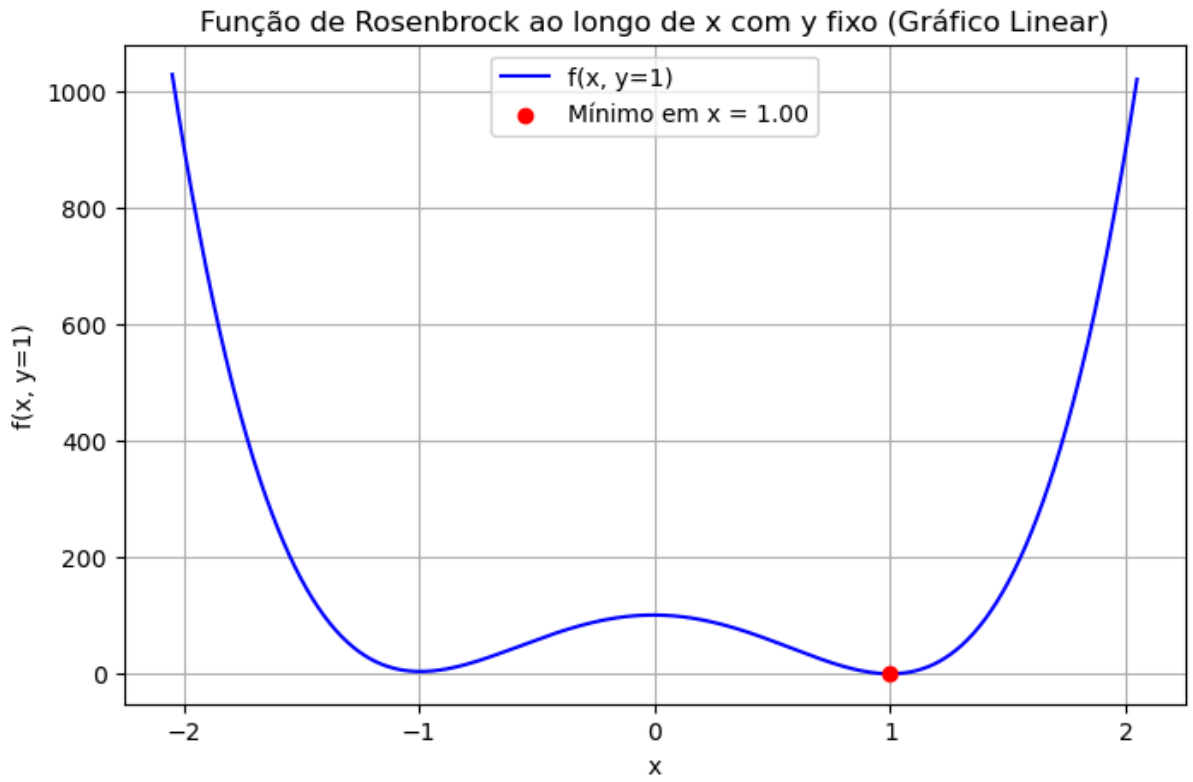


```
O mínimo da função ocorre em x = 1.0060060060060056
O valor mínimo da função é f(x, y=1) = 0.014551705204778649
```

Para o domínio [-2.048,2.048], os seguintes resultados foram encontrados:




```
O mínimo da função ocorre em (x, y) = (0.9983743743743743,
0.9983743743743743)
O valor mínimo da função é f(x, y) = 0.00026605002977602083
```




```
O mínimo da função ocorre em x = 0.9983743743743743
O valor mínimo da função é f(x, y=1) = 0.0010579884374510775
```

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.

Referências

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