Trabalho 1

Centro Federal de Educação Tecnológica de Minas Gerais - Campus III

Engenharia de Computação

Computação Evolucionista

Função de Rosenbrock

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. O objetivo da análise da função é comparar os resultados obtidos pelas buscas sequencial e aleatória. 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 abaixo na Figura 1 e apresenta como mínimo global $f(x^*) = 0, em x^* = (1, ..., 1)$. Na Figura 2, temos a função representada num gráfico.

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

Figura 1: Função de Rosenbrock.

Tabela 1: Tabela 1.1 - Busca sequencial no domínio [-5,10]

Melhor ponto	2.1151247353788483	-5.0	1.26061268997721
Pior ponto	10.17893276880821	-5.0	1179712.0668114482

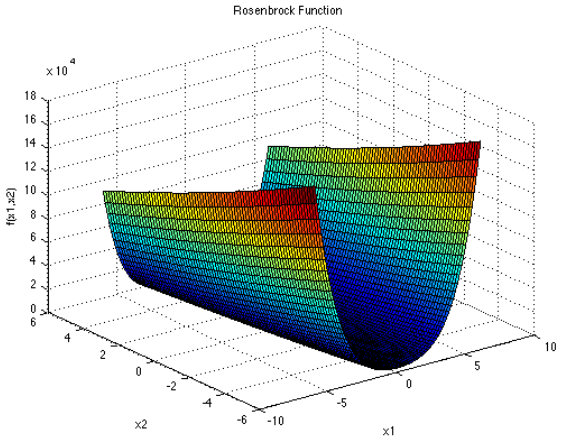

Figura 2: Fonte: SURJANOVIC, BINGHAM.

2 Resultados

Busca Sequencial Para o domínio de [-5,10], os resultados foram organizados na Tabela 1.1 e na Figura 3. Para o domínio de [-2.048, 2.048], os resultados estão na Tabela 1.2 e na Figura 4.

Trabalho 1

Tabela 2: Tabela 1.2 - Busca sequencial no domínio [-2.048, 2.048]

Melhor ponto	0.801591645130932	0.672064752170435	0.12648286067379746
Pior ponto	2.096860574735901	-2.048	4154.779090086123

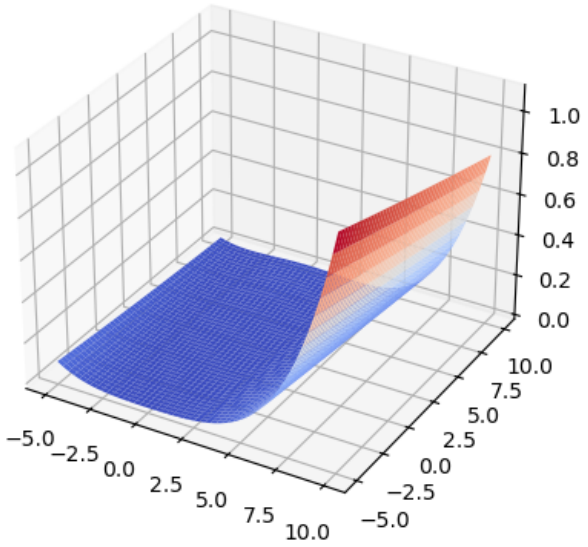

Figura 3: Gráfico da busca sequencial no domínio [-5,10].

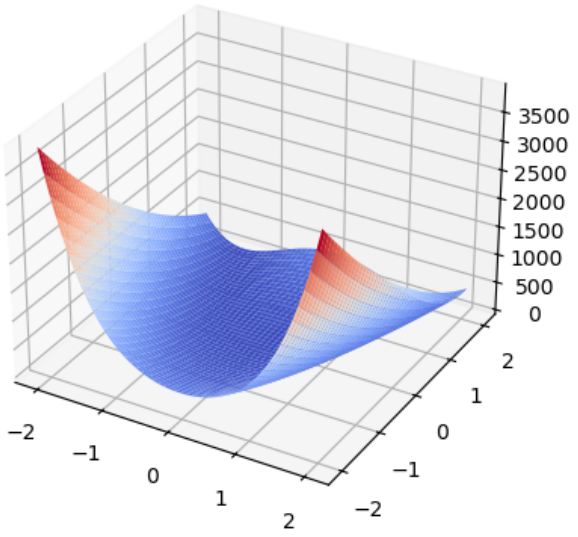

Figura 4: Gráfico da busca sequencial no domínio [-2.048, 2.048].

Busca Sequencial para mais dimensões

Busca Aleatória Para o domínio de [-5,10], os resultados foram organizados na Tabela 3.1 e nas Figuras 5 e 6. Para o domínio de [-2.048, 2.048], os resultados estão na Tabela 3.2 e nas Figuras 7 e 8.

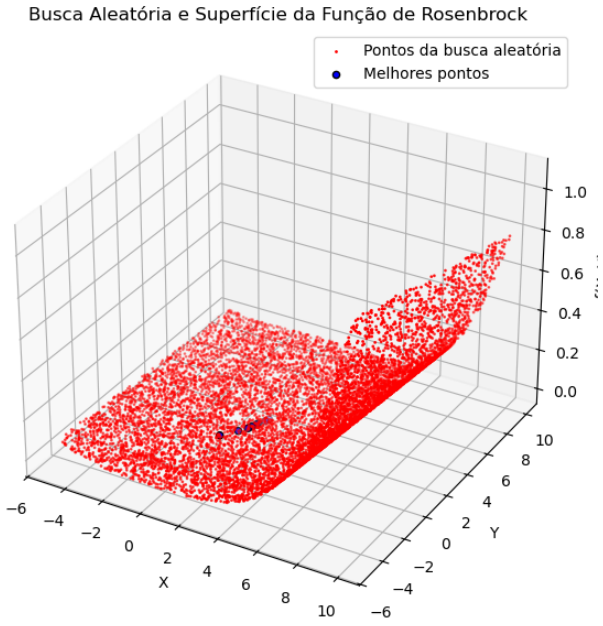

Figura 5: Gráfico da busca aleatória no domínio [-5,10].

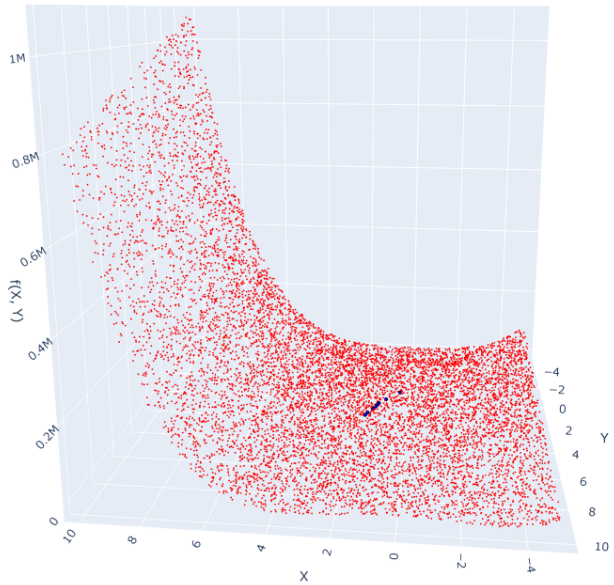

Figura 6: Gráfico da busca aleatória no domínio [-5,10].

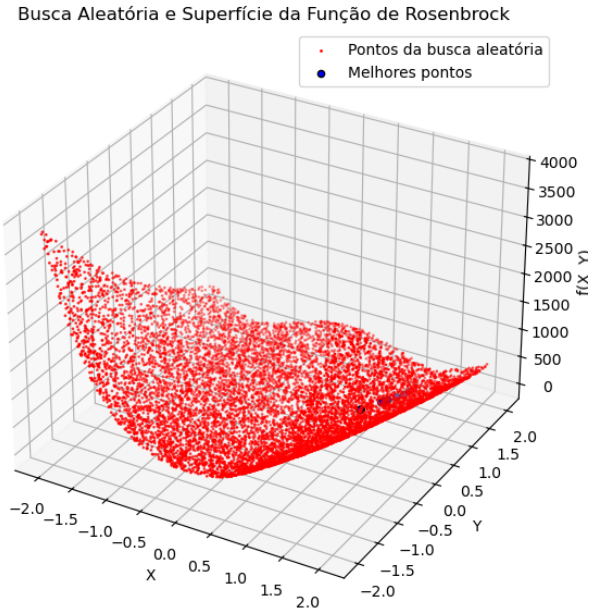

Figura 7: Gráfico da busca aleatória no domínio [-2.048, 2.048].

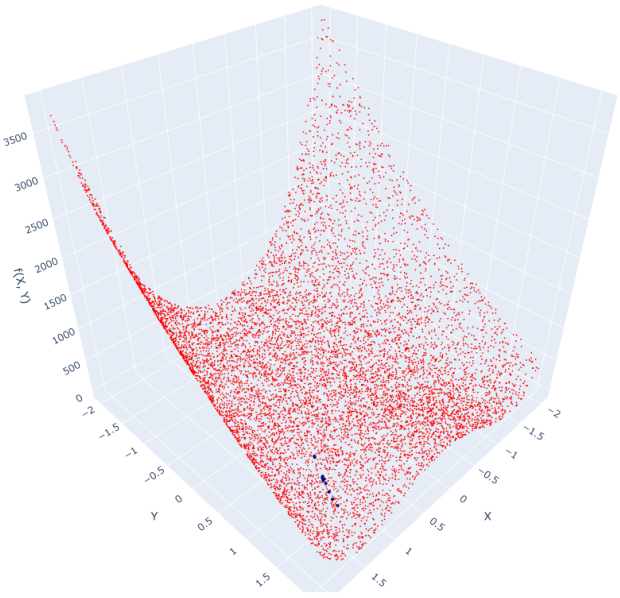

Figura 8: Gráfico da busca aleatória no domínio [-2.048, 2.048].

Busca Aleatória para mais dimensões

3 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.

4 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.

5 Referências

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

6 Anexo 1 - Algoritmos

Algoritmo 1 - Busca Sequencial

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

# Funcao de Rosenbrock para multiplas dimensoes
def rosenbrock(x):
    x = np.array(x)  # Certificar-se de que x e um array NumPy
    return sum(100.0 * (x[1:] - x[:-1]**2.0)**2.0 + (1 - x[:-1])**2.0)

# Funcao para calcular o tamanho do passo em multiplas dimensoes
def calcular_tamanho_passo(num_pontos, limite_inferior, limite_superior,
    dimensao):
    volume_total = (limite_superior - limite_inferior) ** dimensao
```

```
    tamanho_passo = (volume_total / num_pontos) ** (1.0 / dimensao)
    return tamanho_passo

# Funcao de busca sequencial para multiplas dimensoes
def busca_sequencial(dimensao, tamanho_passo, limite_inferior, limite_superior):
    melhor_ponto = None
    melhor_valor = np.inf
    pior_ponto = None
    pior_valor = -np.inf
    melhores_dominios = []
    melhores_imagens = []
    piores_dominios = []
    piores_imagens = []
    imagem = []

    # Gera a lista de valores para cada dimensao com base no tamanho do passo
    ranges = [np.arange(limite_inferior, limite_superior + tamanho_passo, tamanho_passo) for _ in range(dimensao)]

    # Itera por todas as combinacoes de pontos no espaco de busca usando product
    for ponto in itertools.product(*ranges):
        valor = rosenbrock(ponto)
        if valor < melhor_valor:
            melhor_valor = valor
            melhor_ponto = ponto

        if valor > pior_valor:
            pior_valor = valor
            pior_ponto = ponto

        melhores_dominios.append(melhor_ponto)
        melhores_imagens.append(melhor_valor)
        piores_dominios.append(pior_ponto)
        piores_imagens.append(pior_valor)
        imagem.append(valor)

    return melhor_ponto, melhor_valor, pior_valor, pior_ponto, imagem, melhores_dominios, melhores_imagens, piores_dominios, piores_imagens

# Definir os parametros para o dominio [-5, 10] e dimensoes desejadas
limite_inferior = -5
limite_superior = 10
num_pontos = 1000
dimensao = 2  # Aumente para o numero de dimensoes desejado

# Calcular o tamanho do passo para as dimensoes especificadas
tamanho_passo = calcular_tamanho_passo(num_pontos, limite_inferior, limite_superior, dimensao)
```

```
print(f"Tamanho do passo calculado: {tamanho_passo}")

# Executar a busca sequencial
melhor_ponto, melhor_valor, pior_valor, pior_ponto, imagem, melhores_dominios, melhores_imagens, piores_dominios, piores_imagens = busca_sequencial(
    dimensao, tamanho_passo, limite_inferior, limite_superior
)

# Exibir o melhor resultado
print(f'Melhor ponto: {melhor_ponto}, valor = {melhor_valor}')
print(f'Pior ponto: {pior_ponto}, valor = {pior_valor}')

# Plotar o grafico 3D
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

# Gerar pontos para o grafico
x1_v = np.linspace(limite_inferior, limite_superior, 500)
x2_v = np.linspace(limite_inferior, limite_superior, 500)
x1_arr, x2_arr = np.meshgrid(x1_v, x2_v)

# Calcular os valores da funcao Rosenbrock para plotar
f_arr = np.array([[rosenbrock([x1_, x2_]) for x1_ in x1_v] for x2_ in x2_v])

# Plotar a superficie
surf = ax.plot_surface(x1_arr, x2_arr, f_arr, cmap=plt.cm.coolwarm)
plt.show()
```

Algoritmo 2 - Busca Aleatória

```
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):
    return (a - x)**2 + b * (y - x**2)**2

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
```

```
22
23      for _ in range(quantidade_iteracoes):
24          x = random.uniform(limite_inferior, limite_superior)
25          y = random.uniform(limite_inferior, limite_superior)
26          valor = funcao_rosenbrock(x, y)
27
28          pontos_x.append(x)
29          pontos_y.append(y)
30          pontos_valor.append(valor)
31
32          # Verifica e atualiza o melhor e pior ponto
33          if valor < melhor_valor:
34              melhor_valor = valor
35              melhor_x = x
36              melhor_y = y
37
38          if valor > pior_valor:
39              pior_valor = valor
40              pior_x = x
41              pior_y = y
42
43      return melhor_x, melhor_y, melhor_valor, pior_x, pior_y, pior_valor,
        pontos_x, pontos_y, pontos_valor
44
45  # Parametros da busca aleatoria e intervalo de amostragem
46  quantidade_iteracoes = 1000
47  limite_inferior = -5
48  #limite_inferior = -2.048
49  limite_superior = 10
50  #limite_superior = 2.048
51  execucoes = 10
52
53  # Armazenamento dos resultados de todas as execucoes
54  resultados = []
55  todos_pontos_x = []
56  todos_pontos_y = []
57  todos_pontos_valor = []
58  melhores_pontos_x = []
59  melhores_pontos_y = []
60  melhores_pontos_valor = []
61
62  # Executa a busca aleatoria e armazena dados de cada execucao
63  for i in range(execucoes):
64      start_time = time.time()
65      melhor_x, melhor_y, melhor_valor, pior_x, pior_y, pior_valor, pontos_x,
        pontos_y, pontos_valor = busca_aleatoria(
66          quantidade_iteracoes, limite_inferior, limite_superior
67      )
68      exec_time = time.time() - start_time
69
```

```
    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)

# Criacao de uma malha para a superficie da funcao de Rosenbrock
x = np.linspace(limite_inferior, limite_superior, 700)
y = np.linspace(limite_inferior, limite_superior, 700)
x, y = np.meshgrid(x, y)
z = funcao_rosenbrock(x, y)


# Criacao do grafico 3D com a superficie e os pontos da busca aleatoria
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Plota a superficie da funcao de Rosenbrock
# surf = ax.plot_surface(x, y, z, cmap='viridis', alpha=0.6, edgecolor='none')

# Plota todos os pontos encontrados nas 10 execucoes da busca aleatoria
ax.scatter(todos_pontos_x, todos_pontos_y, todos_pontos_valor, color='red', s
    =1, label='Pontos da busca aleatoria')

# Plota os melhores pontos de cada execucao em destaque
ax.scatter(melhores_pontos_x, melhores_pontos_y, melhores_pontos_valor, color
    ='blue', s=20, label='Melhores pontos', edgecolors='black')

# Configuracoes de rotulos, titulo e barra de cores
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("f(X, Y)")
ax.set_title("Busca Aleatoria e Superficie da Funcao de Rosenbrock")
ax.legend()
# fig.colorbar(surf, ax=ax, shrink=0.5, aspect=5)

# Exibe o grafico
plt.show()

# Criacao do grafico de superficie 3D com Plotly
fig = go.Figure()

# Superficie da funcao de Rosenbrock
#fig.add_trace(go.Surface(z=z, x=x, y=y, colorscale='Viridis', opacity=0.7))

# Pontos da busca aleatoria
fig.add_trace(go.Scatter3d(
    x=todos_pontos_x,
    y=todos_pontos_y,
```

```
    z=todos_pontos_valor,
    mode='markers',
    marker=dict(size=1, color='red'),
    name='Pontos da busca aleatoria'
))

# Melhores pontos de cada execucao
fig.add_trace(go.Scatter3d(
    x=melhores_pontos_x,
    y=melhores_pontos_y,
    z=melhores_pontos_valor,
    mode='markers',
    marker=dict(size=2, color='blue', line=dict(width=2, color='black')),
    name='Melhores pontos'
))

# Configuracoes do layout
fig.update_layout(
    title="Busca Aleatoria e Superficie da Funcao de Rosenbrock",
    scene=dict(
        xaxis_title="X",
        yaxis_title="Y",
        zaxis_title="f(X, Y)"
    ),
    width=1500,  # Largura do grafico
    height=1000, # Altura do grafico
)

# Exibe o grafico
fig.show()
```

Algoritmo 3 - Busca Aleatória para mais dimensões

```
import random
import numpy as np
import time

# Fun  o de Rosenbrock adaptada para m ltiplas dimens es
def funcao_rosenbrock(x):
    x = np.array(x)  # Certificar-se de que x    um array NumPy
    return sum(100.0 * (x[1:] - x[:-1]**2.0)**2.0 + (1 - x[:-1])**2.0)

# Fun  o de busca aleat ria para m ltiplas dimens es
def busca_aleatoria(quantidade_iteracoes, limite_inferior, limite_superior,
    dimensao):
    pontos = []
    valores = []

    melhor_ponto = None
    melhor_valor = np.inf
    pior_ponto = None
```

```
    pior_valor = -np.inf

    for _ in range(quantidade_iteracoes):
        # Gera um ponto aleat rio para cada dimens o
        ponto = [random.uniform(limite_inferior, limite_superior) for _ in
    range(dimensao)]
        valor = funcao_rosenbrock(ponto)

        pontos.append(ponto)
        valores.append(valor)

        # Verifica e atualiza o melhor e pior ponto
        if valor < melhor_valor:
            melhor_valor = valor
            melhor_ponto = ponto

        if valor > pior_valor:
            pior_valor = valor
            pior_ponto = ponto

    return melhor_ponto, melhor_valor, pior_ponto, pior_valor, pontos, valores

# Parametros da busca aleatoria e intervalo de amostragem
quantidade_iteracoes = 1000
limite_inferior = -5
limite_superior = 10
execucoes = 10
dimensao = 3  # Ajuste para o numero desejado de dimensoes

# Armazenamento dos resultados de todas as execucoes
resultados = []
todos_pontos = []
todos_valores = []
melhores_pontos = []
melhores_valores = []
piores_valores = []
tempos_execucao = []

# Executa a busca aleatoria e armazena dados de cada execucao
for i in range(execucoes):
    start_time = time.time()
    melhor_ponto, melhor_valor, pior_ponto, pior_valor, pontos, valores =
    busca_aleatoria(
        quantidade_iteracoes, limite_inferior, limite_superior, dimensao
    )
    exec_time = time.time() - start_time

    todos_pontos.extend(pontos)
    todos_valores.extend(valores)
    melhores_pontos.append(melhor_ponto)
```

```
    melhores_valores.append(melhor_valor)
    piores_valores.append(pior_valor)
    tempos_execucao.append(exec_time)

    # Calculo do desvio padrao para os valores de cada execucao
    desvio_padrao = np.std(valores)

    resultados.append({
        'Execucao': i + 1,
        'Melhor Ponto': melhor_ponto,
        'Melhor Valor': melhor_valor,
        'Pior Ponto': pior_ponto,
        'Pior Valor': pior_valor,
        'Tempo (s)': exec_time,
        'Desvio Padrao': desvio_padrao
    })

# Calculo das estatisticas finais apos todas as execucoes
melhor_valor_global = min(melhores_valores)
pior_valor_global = max(piores_valores)
melhor_ponto_global = melhores_pontos[np.argmin(melhores_valores)]
pior_ponto_global = melhores_pontos[np.argmax(piores_valores)]

media_melhores_valores = np.mean(melhores_valores)
mediana_melhores_valores = np.median(melhores_valores)
media_piores_valores = np.mean(piores_valores)
mediana_piores_valores = np.median(piores_valores)
media_tempo_execucao = np.mean(tempos_execucao)
mediana_tempo_execucao = np.median(tempos_execucao)
desvio_padrao_melhores = np.std(melhores_valores)

# Exibe os resultados das execucoes e as estatisticas finais
for resultado in resultados:
    print(resultado)

print("\nEstatisticas Finais:")
print(f"Melhor Ponto Global: {melhor_ponto_global}")
print(f"Melhor Valor Global: {melhor_valor_global}")
print(f"Pior Ponto Global: {pior_ponto_global}")
print(f"Pior Valor Global: {pior_valor_global}")
print(f"Media dos Melhores Valores: {media_melhores_valores}")
print(f"Mediana dos Melhores Valores: {mediana_melhores_valores}")
print(f"Media dos Piores Valores: {media_piores_valores}")
print(f"Mediana dos Piores Valores: {mediana_piores_valores}")
print(f"Media de Tempo de Execucao: {media_tempo_execucao} s")
print(f"Mediana de Tempo de Execucao: {mediana_tempo_execucao} s")
print(f"Desvio Padrao dos Melhores Valores: {desvio_padrao_melhores}")
```