

Trabalho 1

MEC 2403 - Otimização, Algoritmos e Aplicações na Engenharia
Mecânica

Gustavo Henrique Gomes dos Santos
gustavohgs@gmail.com

Professor: Ivan Menezes



Departamento de Engenharia Mecânica
PUC-RJ Pontifícia Universidade Católica do Rio de Janeiro
maio de 2023

Trabalho 1

MEC 2403 - Otimização, Algoritmos e Aplicações na Engenharia Mecânica

Gustavo Henrique Gomes dos Santos

maio de 2023

1 Introdução

1.1 Objetivos

Esse trabalho tem como objetivo a implementação, em Python, e a realização de análise de convergência, para diferentes funções e pontos iniciais, dos seguintes métodos de otimização:

- Univariate
- Powell
- Steepest Descent
- Fletcher-Reeves
- BFGS
- Newton-Raphson

2 Implementações

A estratégia adotada neste trabalho foi de implementar algoritmos que, dados inputs específicos de cada método, retornam a próxima direção de busca. Para a busca unidirecional na direção especificada por cada método, foram aproveitados e melhorados os códigos do passo constante e da seção áurea utilizados na resolução da Lista-1.

2.1 Pacotes utilizados

As seguintes bibliotecas são necessárias para execução do código final do arquivo principal :

```
import numpy as np
import matplotlib.pyplot as plt
from timeit import default_timer as timer
```

Os métodos de busca unidimensional, assim como os métodos de otimização foram implementados em arquivos distintos (osr_methods.py e line_search_methods.py). Com isso, no arquivo principal também é necessário realizar o import desses algoritmos.

```
import osr_methods as osr
import line_search_methods as lsm
```

Esses arquivos distintos necessitam apenas do pacote numpy. Com isso apenas o seguinte import é necessário nos dois arquivos citados acima.

```
import numpy as np
```

2.2 Busca Unidirecional

Os algoritmos do passo constante e da seção áurea foram implementados em um arquivo denominado line_search_methods.py. O seguinte pacote é necessário nesse arquivo :

```
import numpy as np
```

2.2.1 Passo Constante

Foi implementado um método que recebe como parâmetros de entrada um vetor direção (\vec{dir}), um ponto inicial (\vec{P}_1), a função que se deseja encontrar o mínimo ($f(\vec{P})$), um valor opcional de epsilon da máquina (ϵ default com valor 10^{-8}), um valor opcional de passo ($\Delta\alpha$ default com valor 0.01).

Para definir o sentido da busca, é feita uma comparação entre o valor de $f(\vec{P}_1 - \epsilon \vec{dir})$ e $f(\vec{P}_1 + \epsilon \vec{dir})$. Caso este último valor seja maior do que o primeiro, o sentido de busca considerado é o oposto do vetor direção ($-\vec{dir}$). Caso o primeiro seja o maior valor, o sentido do vetor direção é mantido na busca (\vec{dir}).

Com o passo default de 0.01 ou um qualquer outro passo desejado informado na passagem de parâmetro opcional, o método percorre o sentido de busca definido até encontrar um valor de $f(\vec{P}_1 + \alpha \vec{dir})$ que seja inferior ao valor do próximo passo. Quando essa condição é alcançada, esse último valor de α é definido como mínimo (α_{min}).

Para garantir que a cada incremento de α não tenha sido pulado um mínimo da função, é feito uma comparação entre os valores de $f(\vec{P} - \epsilon \vec{dir})$ e $f(\vec{P})$, onde $\vec{P} = \vec{P}_1 + \alpha \vec{dir}$. Caso o último valor seja superior ao primeiro, o passo é desfeito e o α mínimo é considerado encontrado.

Caso uma das condições abaixo seja atingida, o algoritmo é interrompido e retorna o intervalo $[\alpha, \alpha + \Delta\alpha]$, fazendo os devidos ajustes para adequar os sinais de acordo com sentido de busca.

- $f(\vec{P} - \epsilon \vec{dir}) < f(\vec{P})$, com $\vec{P} = \vec{P}_1 + \alpha \vec{dir}$
- $f(\vec{P}_1 + \alpha \vec{dir}) < f(\vec{P}_1 + (\alpha + \Delta\alpha) \vec{dir})$

```
def passo_cte(direcao, P0, f, eps = 1E-8, step = 0.01):
    #line search pelo metodo do passo constante

    #define o sentido correto de busca
    if (f(P0 - eps*(direcao/np.linalg.norm(direcao))) > f(P0 + eps*(direcao/np.linalg.norm(
        direcao)))):
        sentido_busca = direcao.copy()
        flag = 0
    else:
        sentido_busca = -direcao.copy()
        flag = 1

    P = P0.copy()
    P_next = P + step*sentido_busca
    alpha = 0

    while (f(P) > f(P_next)):
        alpha = alpha + step
        P = P0 + alpha*sentido_busca
        P_next = P0 + (alpha+step)*sentido_busca
        if (f(P - eps*(sentido_busca/np.linalg.norm(sentido_busca))) < f(P)):
            alpha = alpha - step
            break

    intervalo = np.array([alpha, alpha + step])

    if(flag == 1):
        intervalo = -intervalo

    #retorna o intervalo de busca = [alpha min, alpha min + step]
    return intervalo
```

2.2.2 Seção Áurea

Implementado um método que recebe como parâmetros de entrada um intervalo de busca ($\overrightarrow{interv} = [\alpha^L, \alpha^U]$), um vetor direção de busca (\vec{dir}), um ponto inicial (\vec{P}_1), a função f ($f(\vec{P})$) e um parâmetro opcional para a tolerância de convergência com valor default de 10^{-5} .

Resumidamente, o método utiliza a razão áurea ($R_a = \frac{\sqrt{5}-1}{2}$) para comparar os valores de $f(\vec{P}_1 + \alpha_E \vec{dir})$ e $f(\vec{P}_1 + \alpha_D \vec{dir})$, onde $\alpha_E = \alpha^L + (1 - R_a)\beta$, $\alpha_D = \alpha^L + R_a\beta$ e $\beta = \alpha^U - \alpha^L$, para determinar em qual trecho, $[\alpha^L, \alpha_D]$ ou $[\alpha_E, \alpha^U]$, o ponto mínimo se encontra. Enquanto a convergência não é alcançada, os valores de $\alpha^L, \alpha^U, \alpha_E$ e α_D vão sendo recalculados e atualizados.

Quando o comprimento do trecho a ser avaliado é inferior à tolerância, o método finaliza e retorna o seguinte valor:

- α_{min} , tal que $\alpha_{min} = \frac{\alpha^L + \alpha^U}{2}$ e α^L , α^U são os extremos do intervalo do último passo do algoritmo, quando a convergência foi obtida

Importante destacar que o algoritmo identifica o sentido de busca e os sinal correto do valor de alpha mínimo através do valor do intervalo passado como parâmetro.

```
def secao_aurea(intervalo, direcao, P0, f, tol=0.00001):
    #line search pelo metodo da secao aurea

    #verifica o sentido da busca
    if(intervalo[1] < 0):
        intervalo = -intervalo
        sentido_busca = -direcao.copy()
        flag = 1
    else:
        sentido_busca = direcao.copy()
        flag = 0

    #atribui os limites superior e inferior da busca a variaveis internas do metodo
    alpha_upper = intervalo[1]
    alpha_lower = intervalo[0]
    beta = alpha_upper - alpha_lower

    #razao aurea
    Ra = (np.sqrt(5)-1)/2

    # define os pontos de analise de f com base na razao aurea
    alpha_e = alpha_lower + (1-Ra)*beta
    alpha_d = alpha_lower + Ra*beta

    #primeira iteracao avalia f nos 2 pontos selecionados pela razao aurea
    f1 = f(P0 + alpha_e*sentido_busca)
    f2 = f(P0 + alpha_d*sentido_busca)

    #loop enquanto a convergencia nao for obtida
    while (beta > tol):
        if (f1 > f2):
            #caso positivo, define novo intervalo variando de alpha_e ate alpha_upper
            # e aproveita os valores anteriores de alpha_d e f2 como novos alpha_e e f1
            alpha_lower = alpha_e
            f1 = f2
            alpha_e = alpha_d

            #calcula novo alpha_d e f2=f(alpha_d)
            beta = alpha_upper - alpha_lower
            #alpha_e = alpha_lower + (1-Ra)*beta
            alpha_d = alpha_lower + Ra*beta
            f2 = f(P0 + alpha_d*sentido_busca)
        else:
            #caso negativo, define novo intervalo variando de alpha_lower ate alpha_d
            # e aproveita os valores anteriores de alpha_e e f1 como novos alpha_d e f2
            alpha_upper = alpha_d
            f2 = f1
            alpha_d = alpha_e

            #calcula novo alpha_e e f1=f(alpha_e)
            beta = alpha_upper - alpha_lower
            alpha_e = alpha_lower + (1-Ra)*beta
            #alpha_d = alpha_lower + Ra*beta
            f1 = f(P0 + alpha_e*sentido_busca)

    # calcula Pmin e alpha min apos convergencia
    alpha_med = (alpha_lower + alpha_upper)/2
    alpha_min = alpha_med

    if (flag == 1):
        alpha_min = -alpha_min

    return alpha_min
```

2.3 Métodos OSR

Os algoritmos dos métodos Univariate, Powell, Steepest Descent, Fletcher-Reeves, BFGS e Newton-Raphson foram implementados em um arquivo denominado osr_methods.py. O seguinte pacote é necessário nesse arquivo

:

```
import numpy as np
```

2.3.1 Univariate

O método univariante alterna entre as direções canônicas. A primeira vez que ele é chamado, retorna a primeira direção canônica. Na segunda vez, a segunda direção canônica, e assim por diante. Quando todas as direções canônicas são utilizadas, reinicia-se pela primeira direção.

A implementação considerou como parâmetros de entrada o número de dimensões e o passo em que a otimização se encontra. Ambos valores são facilmente calculados no código principal que irá chamar os métodos de OSR. O número de dimensões é extraído do ponto inicial e o passo é um valor controlado durante o processo de convergência que irá ser implementado no código principal.

Toda vez que o método é chamado, inicializa-se um vetor com n zeros, sendo n o número de dimensões. O índice do vetor cujo valor será alterado para 1 é calculado através de manipulações em cima do resto da divisão do numero do passo pelo número de dimensões.

```
def univariante(passo, dimens):
    indice = passo%dimens - 1
    if (indice == -1):
        indice = dimens - 1
    ek = np.zeros(dimens)
    ek[indice] = 1

    return ek
```

2.3.2 Powell

```
def powell(P, P1, direcoes, passos, ciclos, dimens):
    indice = passos%(dimens + 1) - 1
    if (indice == -1):
        dir = P - P1
        direcoes[dimens - 1] = dir
    elif (indice == 0):
        ciclos = ciclos + 1
        if (ciclos%(dimens+2) == 0):
            direcoes = np.eye(dimens, dtype=float)
        P1 = P.copy()
        dir = direcoes[indice].copy()
    else:
        dir = direcoes[indice].copy()
        direcoes[indice-1] = dir

    return dir, direcoes, P1, ciclos
```

2.4 Exercício 2

Utilizando os métodos implementados na questão anterior, testar a sua implementação encontrando o ponto mínimo das seguintes funções:

a. Função 1:

$$f(x_1, x_2) = x_1^2 - 3x_1x_2 + 4x_2^2 + x_1 - x_2 \text{ com } \vec{P}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \text{ e } \vec{d} = \begin{bmatrix} -1 \\ -2 \end{bmatrix}$$

b. Função 2 - McCormick:

$$f(x_1, x_2) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1,5x_1 + 2,5x_2 \text{ com } \vec{P}_1 = \begin{bmatrix} -2 \\ 3 \end{bmatrix} \text{ e } \vec{d} = \begin{bmatrix} 1.453 \\ -4.547 \end{bmatrix}$$

c. Função 3 - Himmelblau:

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \text{ com } \vec{P}_1 = \begin{bmatrix} 0 \\ 5 \end{bmatrix} \text{ e } \vec{d} = \begin{bmatrix} 3 \\ 1.5 \end{bmatrix}$$

- Para cada função acima, desenhar(na mesma figura): as curvas de nível e o segmento de reta conectando o ponto inicial ao ponto de mínimo.
- Adotar uma tolerância de 10^{-5} para verificação da convergência numérica.

2.4.1 Função 1

Importação das bibliotecas, da implementação dos métodos de busca unidimensional do exercício anterior e definição da função do exercício:

```
import numpy as np
import matplotlib.pyplot as plt
import linear_search_methods as lsm

def f(P):
    # P = [x1, x2]
    return P[0]**2 - 3*P[0]*P[1] + 4*(P[1]**2) + P[0] - P[1]
```

Cálculo do ponto mínimo usando os 3 métodos do exercício 1:

- Passo constante
- Bisseção
- Seção Áurea

```
#inputs de Ponto inicial e direcao de busca
P1 = np.array([1, 2])
dir = np.array([-1, -2])

#chama o metodo do passo constante
q2_a_pss_ct = lsm.passo_cte(dir.copy(), P1, f)

#funcao lsm.passo_cte entrega o intervalo de busca na segunda posicao do array de retorno
intervalo = q2_a_pss_ct[1]

#resgata o sentido unitario correto da busca unidimensional
sentido_busca = q2_a_pss_ct[2]

#chama o o metodo da bissecao
q2_a_bssc = lsm.bissecao(intervalo.copy(), sentido_busca.copy(), P1, f)

#chama o metodo da secao aurea
q2_a_sc_ar = lsm.secao_aurea(intervalo.copy(), sentido_busca.copy(), P1, f)

print(f'Passo constante : |\u03B1 min| = {intervalo[0]:.10f} e P min = ({q2_a_pss_ct[0][0]:.10f}, {q2_a_pss_ct[0][1]:.10f}) ')
print(f'Bissecao : |\u03B1 min| = {q2_a_bssc[1]:.10f} e P min = ({q2_a_bssc[0][0]:.10f}, {q2_a_bssc[0][1]:.10f}) ')
print(f'Secao Aurea : |\u03B1 min| = {q2_a_sc_ar[1]:.10f} e P min = ({q2_a_sc_ar[0][0]:.10f}, {q2_a_sc_ar[0][1]:.10f}) ')
```

Resultados obtidos (Saída do terminal):

Passo constante : $|\alpha_{min}| = 2.1300000000$ e $P_{min} = (0.0474350416, 0.0948700832)$

Bisseção : $|\alpha_{min}| = 2.1344287109$ e $P_{min} = (0.0454544618, 0.0909089237)$

Seção Áurea : $|\alpha_{min}| = 2.1344280564$ e $P_{min} = (0.0454547546, 0.0909095092)$

A solução para a função $f(x_1, x_2) = x_1^2 - 3x_1x_2 + 4x_2^2 + x_1 - x_2$ então fica :

- Passo constante: $|\alpha_{min}| = 2.1300000000$ e $\overrightarrow{P_{min}} = \begin{bmatrix} 0.0474350416 \\ 0.0948700832 \end{bmatrix}$
- Bisseção: $|\alpha_{min}| = 2.1344287109$ e $\overrightarrow{P_{min}} = \begin{bmatrix} 0.0454544618 \\ 0.0909089237 \end{bmatrix}$
- Seção Áurea: $|\alpha_{min}| = 2.1344280564$ e $\overrightarrow{P_{min}} = \begin{bmatrix} 0.0454547546 \\ 0.0909095092 \end{bmatrix}$

Como todos os métodos entregam respostas de P_{min} muito parecidas, com diferenças relativamente pequenas, para a etapa de desenhar as curvas de nível e o segmento de reta conectando P_1 ao P_{min} tomei a liberdade de plotar apenas o resultado da Seção Áurea.

```

#Escolhido o Pmin gerado pelo metodo da secao aurea para representar graficamente
Pmin = q2_a_sc_ar[0]

x1 = np.linspace(-7.5, 7.5, 100)
x2 = np.linspace(-7.5, 7.5, 100)
X1, X2 = np.meshgrid(x1, x2)
x3 = f([X1, X2])
niveis = plt.contour(X1, X2, x3, [0, 3, 10, 25, 40, 60, 100, 150], colors='black')
plt.clabel(niveis, inline=1, fontsize=10)
plt.annotate('', xy=Pmin, xytext=P1,
             arrowprops=dict(width=1, color='green', headwidth=10, headlength=10, shrink=
                             0.05), fontsize='10')
plt.annotate(f'({P1[0]}, {P1[1]})', xy=P1, xytext=(5,-10), textcoords='offset points', color=
             'green')
plt.annotate(f'$P_{\{min\}}$=({round(Pmin[0], 7)}, {round(Pmin[1], 7)})', xy=Pmin, xytext=(0.
             03,0.95), textcoords='axes fraction', color='
             green')
plt.plot(P1[0], P1[1], marker="o", markersize=7, markeredgecolor="green", markerfacecolor="
             green")
plt.plot(Pmin[0], Pmin[1], marker="o", markersize=7, markeredgecolor="green",
             markerfacecolor="green")

plt.xlabel('$x_1$', fontsize='16')
plt.ylabel('$x_2$', fontsize='16')
plt.grid(linestyle='--')
plt.title("$f(x_1, x_2)$ - Metodo da Secao Aurea", fontsize='16')
plt.savefig("A_solution.pdf", format="pdf")
plt.show()

```

Figura 1: Função 1 - Seção Áurea

2.4.2 Função 2: McCormick

Importação das bibliotecas, da implementação dos métodos de busca unidimensional do exercício anterior e definição da função do exercício:

```

import numpy as np
import matplotlib.pyplot as plt
import linear_search_methods as lsm

def mcCormick(P):
    # P = [x1, x2]
    return np.sin(P[0] + P[1]) + (P[0] - P[1])**2 - 1.5*P[0] + 2.5*P[1]

```

Cálculo do ponto mínimo usando os 3 métodos do exercício 1:

- Passo constante
- Bissecção
- Seção Áurea

```

#inputs de Ponto inicial e direcao de busca
P1 = np.array([-2, 3])
dir = np.array([1.453, -4.547])

#chama o metodo do passo constante
q2_b_pss_ct = lsm.passo_cte(dir.copy(), P1, mcCormick)

#funcao lsm.passo_cte entrega o intervalo de busca na segunda posicao do array de retorno
intervalo = q2_b_pss_ct[1]

#resgata o sentido unitario correto da busca unidimensional
sentido_busca = q2_b_pss_ct[2]

#chama o o metodo da bissecao
q2_b_bssc = lsm.bissecao(intervalo.copy(), sentido_busca.copy(), P1, mcCormick)

#chama o metodo da secao aurea
q2_b_sc_ar = lsm.secao_aurea(intervalo.copy(), sentido_busca.copy(), P1, mcCormick)

```

```

print(f'Passo constante : |\u03B1| min = {intervalo[0]:.10f} e P min = ({q2_b_pss_ct[0][0]:.10f}, {q2_b_pss_ct[0][1]:.10f}) ')
print(f'Bissecao      : |\u03B1| min = {q2_b_bssc[1]:.10f} e P min = ({q2_b_bssc[0][0]:.10f}, {q2_b_bssc[0][1]:.10f}) ')
print(f'Secao Aurea   : |\u03B1| min = {q2_b_sc_ar[1]:.10f} e P min = ({q2_b_sc_ar[0][0]:.10f}, {q2_b_sc_ar[0][1]:.10f}) ')

```

Resultados obtidos (Saída do terminal):

Passo constante : $|\alpha_{min}| = 4.7700000000$ e $P_{min} = (-0.5480690486, -1.5436545327)$

Bisseção : $|\alpha_{min}| = 4.7735791016$ e $P_{min} = (-0.5469796129, -1.5470637991)$

Seção Áurea : $|\alpha_{min}| = 4.7735766069$ e $P_{min} = (-0.5469803722, -1.5470614229)$

A solução para a função $f(x_1, x_2) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1,5x_1 + 2,5x_2$ então fica :

- Passo constante: $|\alpha_{min}| = 4.7700000000$ e $\overrightarrow{P_{min}} = \begin{bmatrix} -0.5480690486 \\ -1.5436545327 \end{bmatrix}$
- Bisseção: $|\alpha_{min}| = 4.7735791016$ e $\overrightarrow{P_{min}} = \begin{bmatrix} -0.5469796129 \\ -1.5470637991 \end{bmatrix}$
- Seção Áurea: $|\alpha_{min}| = 4.7735766069$ e $\overrightarrow{P_{min}} = \begin{bmatrix} -0.5469803722 \\ -1.5470614229 \end{bmatrix}$

Como todos os métodos entregam respostas de P_{min} muito parecidas, com diferenças relativamente pequenas, para a etapa de desenhar as curvas de nível e o segmento de reta conectando P_1 ao P_{min} tomei a liberdade de plotar apenas o resultado do Passo Constante.

```

#Escolhido o Pmin gerado pelo metodo do passo constante para representar graficamente
Pmin = q2_b_pss_ct[0]

x1 = np.linspace(-7.5, 7.5, 100)
x2 = np.linspace(-7.5, 7.5, 100)
X1, X2 = np.meshgrid(x1, x2)
x3 = mcCormick([X1, X2])
niveis = plt.contour(X1, X2, x3, [0, 5, 15, 40, 60], colors='black')
plt.clabel(niveis, inline=1, fontsize=10)
plt.annotate('', xy=Pmin, xytext=P1,
              arrowprops=dict(width=1, color='green', headwidth=10, headlength=10, shrink=
                              0.05), fontsize='10')
plt.annotate(f'({P1[0]}, {P1[1]})', xy=P1, xytext=(10,0), textcoords='offset points', color=
              'green')
plt.annotate(f'$P_{\{min\}}$=({round(Pmin[0], 7)}, {round(Pmin[1], 7)})', xy=Pmin, xytext=(0.
              03, 0.95), textcoords='axes fraction', color=
              'green')
plt.plot(P1[0], P1[1], marker="o", markersize=7, markeredgecolor="green", markerfacecolor="
              green")
plt.plot(Pmin[0], Pmin[1], marker="o", markersize=7, markeredgecolor="green",
              markerfacecolor="green")

plt.xlabel('$x_1$', fontsize='16')
plt.ylabel('$x_2$', fontsize='16')
plt.grid(linestyle='--')
plt.title("$mcCormick(x_1, x_2)$ - Metodo do Passo Constante", fontsize='16')
plt.savefig("B_solution.pdf", format="pdf")
plt.show()

```

Figura 2: McCormick - Passo Constante

2.4.3 Função 3: Himmelblau

Importação das bibliotecas, da implementação dos métodos de busca unidimensional do exercício anterior e definição da função do exercício:

```

import numpy as np
import matplotlib.pyplot as plt
import linear_search_methods as lsm

```



```
def himmelblau(P):
    # P = [x1, x2]
    return (P[0]**2 + P[1] - 11)**2 + (P[0] + P[1]**2 - 7)**2
```

Cálculo do ponto mínimo usando os 3 métodos do exercício 1:

- Passo constante
- Bissecção
- Seção Áurea

```
#inputs de Ponto inicial e direcao de busca
P1 = np.array([0, 5])
dir = np.array([3, 1.5])

#chama o metodo do passo constante
q2_c_pss_ct = lsm.passo_cte(dir, P1, himmelblau)

#funcao lsm.passo_cte entrega o intervalo de busca na segunda posicao do array de retorno
intervalo = q2_c_pss_ct[1]

#resgata o sentido unitario correto da busca unidimensional
sentido_busca = q2_c_pss_ct[2]

#chama o o metodo da bissecao
q2_c_bssc = lsm.bissecao(intervalo.copy(), sentido_busca.copy(), P1, himmelblau)

#chama o metodo da secao aurea
q2_c_sc_ar = lsm.secao_aurea(intervalo.copy(), sentido_busca.copy(), P1, himmelblau)

print(f'Passo constante : |\u03B1 min| = {intervalo[0]:.10f} e P min = ({q2_c_pss_ct[0][0]:.10f}, {q2_c_pss_ct[0][1]:.10f}) ')
print(f'Bissecao : |\u03B1 min| = {q2_c_bssc[1]:.10f} e P min = ({q2_c_bssc[0][0]:.10f}, {q2_c_bssc[0][1]:.10f}) ')
print(f'Seção Aurea : |\u03B1 min| = {q2_c_sc_ar[1]:.10f} e P min = ({q2_c_sc_ar[0][0]:.10f}, {q2_c_sc_ar[0][1]:.10f}) ')
```

Resultados obtidos (Saída do terminal):

Passo constante : $|\alpha_{min}| = 3.3900000000$ e $P_{min} = (-3.0321081775, 3.4839459113)$

Bissecção : $|\alpha_{min}| = 3.3921630859$ e $P_{min} = (-3.0340429004, 3.4829785498)$

Seção Áurea : $|\alpha_{min}| = 3.3921633455$ e $P_{min} = (-3.0340431325, 3.4829784337)$

A solução para a função $f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$ então fica :

- Passo constante: $|\alpha_{min}| = 3.3900000000$ e $\overrightarrow{P_{min}} = \begin{bmatrix} -3.0321081775 \\ 3.4839459113 \end{bmatrix}$
- Bissecção: $|\alpha_{min}| = 3.3921630859$ e $\overrightarrow{P_{min}} = \begin{bmatrix} -3.0340429004 \\ 3.4829785498 \end{bmatrix}$
- Seção Áurea: $|\alpha_{min}| = 3.3921633455$ e $\overrightarrow{P_{min}} = \begin{bmatrix} -3.0340431325 \\ 3.4829784337 \end{bmatrix}$

Como todos os métodos entregam respostas de P_{min} muito parecidas, com diferenças relativamente pequenas, para a etapa de desenhar as curvas de nível e o segmento de reta conectando P_1 ao P_{min} tomei a liberdade de plotar apenas o resultado da Bissecção.

```
#Escolhido o Pmin gerado pelo metodo da bissecao para representar graficamente
Pmin = q2_c_bssc[0]

x1 = np.linspace(-7.5, 7.5, 1000)
x2 = np.linspace(-7.5, 7.5, 1000)
X1, X2 = np.meshgrid(x1, x2)
x3 = himmelblau([X1, X2])
niveis = plt.contour(X1, X2, x3, [10,50,100,200, 350, 1000], colors='black')
plt.clabel(niveis, inline=1, fontsize=10)
plt.annotate(' ', xy=Pmin, xytext=P1,
```

```

        arrowprops=dict(width=1, color='green', headwidth=10, headlength=10, shrink=
                                0.05), fontsize='10')
plt.annotate(f'$P_1$=({P1[0]}, {P1[1]})', xy=P1, xytext=(0.03,0.95), textcoords='axes
                                fraction', color='green')
plt.annotate(f'$P_{\{min\}}$=({round(Pmin[0], 7)}, {round(Pmin[1], 7)})', xy=Pmin, xytext=(0.
                                55,0.95), textcoords='axes fraction', color='
                                green')
plt.plot(P1[0], P1[1], marker="o", markersize=7, markeredgecolor="green", markerfacecolor="
                                green")
plt.plot(Pmin[0], Pmin[1], marker="o", markersize=7, markeredgecolor="green",
                                markerfacecolor="green")

plt.xlabel('$x_1$', fontsize='16')
plt.ylabel('$x_2$', fontsize='16')
plt.grid(linestyle='--')
plt.title("$himmelblau(x_1, x_2)$ - Metodo da Bissecão", fontsize='16')
plt.savefig("C_solution.pdf", format="pdf")
plt.show()

```

Figura 3: Himmelblau - Bissecção