Lista de Exercícios 1

MEC 2403 - Otimização e Algoritmos para Engenhria 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

abril de 2023

Lista de Exercícios 1

MEC 2403 - Otimização e Algoritmos para Engenhria Mecânica

Gustavo Henrique Gomes dos Santos

abril de 2023

1 Objetivo

Esse relatório tem como objetivo apresentar a resolução, utilizando a linguagem de programação Python, da Lista de Exercícios 1 da disciplina MEC 2403 - Otimização e Algoritmos para Engenhria Mecânica, com prazo de entrega em 04 de abril de 2023.

2 Exercícios

2.1 Exercício 1

Implementar os seguintes métodos para o cálculo do ponto mínimo de funções de uma única variável:

1. Passo constante (com $\Delta\alpha = 0.01$)
2. Bisseção (usando o Passo Constante para obtenção do intervalo de busca)
3. Seção Áurea (usando o Passo Constante para obtenção do intervalo de busca)

2.1.1 Passo Constante

Foi implementado um método que recebe como parâmetros de entrada um vetor direção, um ponto inicial, a função que se deseja encontrar o mínimo, um valor opcional de passo ($\Delta\alpha$ default com valor 0.01) e mais um valor opcional de módulo ou distância de busca (com valor default de 1000).

A partir do vetor direção, um método para cálculo do vetor unitário é chamado. Para definir o sentido da busca, é feita uma comparação entre o valor de $f(\overrightarrow{P_1} - \epsilon\,\hat{dir})$ e $f(\overrightarrow{P_1} + \epsilon\,\hat{dir})$, onde ϵ foi definido como 10^{-8}. Caso este último valor seja maior do que o primeiro, o sentido de busca considerado é o oposto do vetor direção. Caso o primeiro seja o maior valor, o sentido do vetor direção é mantido na busca.

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 $f(\overrightarrow{P_1} + \alpha\,\hat{sentido})$ que seja inferior ao valor do próximo passo. Quando essa condição é alcançada, esse último valor de α é definido como mínimo (α_{min}).

O método é então interrompido, retornando as seguinte informações :

1. $\overrightarrow{P_{min}}$, tal que $\overrightarrow{P_{min}} = \overrightarrow{P_1} + \alpha\,\hat{sentido}$;
2. $\overrightarrow{interv} = [\alpha_{min}, \alpha_{min} + \Delta\alpha]$;
3. $\hat{sentido}$, tal que $\hat{sentido} = \hat{dir}$ ou $\hat{sentido} = -\hat{dir}$, onde $\hat{dir}$ é o vetor unitário na direção de busca informada na chamada do método.

Importante destacar que, como o vetor sentido de busca unitário é calculado nesse método e o Pmin é uma das saídas do próprio método, os valores de α sempre são tratados como valores positivos, deixando o vetor sentido "cuidar"do correto tratamento do sinal das operações.

```
import numpy as np
import math

def dirUnit(direcao):
  #calcula o vetor unitario a partir do vetor direcao informado
  return direcao/np.linalg.norm(direcao)

  def passo_cte(direcao, P1, f, step = 0.01, modulo_direcao = 1000):
```

```
    #linear search pelo metodo do passo constante

    #calcula o vetor unitario na direcao de busca solicitada
    direcao_unitaria = dirUnit(direcao)

    #epsilon
    eps=0.00000001

    #define o sentido unitario correto de busca
    if (f(P1 - eps*direcao_unitaria) > f(P1 + eps*direcao_unitaria)):
        sentido_busca = direcao_unitaria
    else:
        sentido_busca = -direcao_unitaria

    #iteracao em alpha variando de 0 ate o modulo informado da direcao(opcional)
    #ou ate 1000 (default)
    for alpha in np.arange(0, modulo_direcao, step):
        # atualiza os valores de Pmin e alpha min com os valores de cada step
        Pmin = P1 + alpha*sentido_busca
        alpha_min = alpha

        #verifica se a funcao fica ascendente no proximo step
        #caso positivo, a busca unidimensional termina e os valores de Pmin
        #e alpha min ja estao guardados
        if (f(Pmin) < f(P1 + (alpha+step)*sentido_busca)):
            break

        #retorna o Pmin,
        #o intervalo de busca = [alpha min, alpha min + step]
        #e o vetor unitario sentido
    return Pmin, np.array([alpha_min, alpha_min+step]), sentido_busca
```

2.1.2 Bisseção

Implementado um método recursivo que recebe como parâmetros de entrada um intervalo de busca ($\overrightarrow{interv} = [\alpha^L, \alpha^U]$), um vetor unitário no sentido correto de busca, um ponto inicial, a função f e um parâmetro opcional para a tolerância de convergência com valor default de 0.00001.

O intervalo de busca e o vetor unitário no sentido da busca podem ser livremente fornecidos ou então estimados pelo método do passo constante. Resumidamente, o método compara os valores de $f(\overrightarrow{P_1} + (\alpha_{med} - \epsilon)\,\hat{sentido})$ e $f(\overrightarrow{P_1} + (\alpha_{med} + \epsilon)\,\hat{sentido})$, onde $\alpha_{med} = \frac{\alpha^L + \alpha^U}{2}$ e $\epsilon = 10^{-8}$, para determinar em qual metade do intervalo o ponto mínimo se encontra, descartando a outra metade. Após identifcar a metade correta, o algoritmo chama recursivamente o método da Bisseção com um novo intervalo de busca.

Isso ocorre até que a convergência seja alcançada e assim o ponto mínimo determinado. O método finaliza e retorna os seguintes valores :

1. $\overrightarrow{P_{min}}$, tal que $\overrightarrow{P_{min}} = \overrightarrow{P_1} + \alpha_{min}\,\hat{sentido}$
2. 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 algortimo, quando a convergência foi obtida

```
  def bissecao(intervalo, sentido_busca, P1, f, tol=0.00001):
    #linear search pelo metodo da bissecao
    #funcao estruturada de forma recursiva

    #epsilon
    eps = 0.00000001

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

    #atualiza o valor de alpha min
    #(igual ao ultimo alpha med) e  Pmin em cada chamada do metodo
    alpha_med = (alpha_lower + alpha_upper)/2
    alpha_min = alpha_med
    Pmin = P1 + alpha_min*sentido_busca

    #condicao de convergencia
    if (alpha_upper - alpha_lower) <= tol:
```

```
            #caso positivo, a busca termina
            #e os valores de Pmin e alpha min ja estao guardados
            return Pmin, alpha_min
        else:
            #caso negativo, verifica se o lado a esquerda
            #ou a direita do alpha med deve ser descartado
            # e chama, recursivamente, o metodo da bissecao com o intervalo restante
            f1 = f(P1 + (alpha_med - eps)*sentido_busca)
            f2 = f(P1 + (alpha_med + eps)*sentido_busca)

            #verifica se o valor de f a esquerda
            #e maior do que o valor de f a direita do alpha med
            if (f1 > f2):
                #caso positivo, define novo intervalo de busca
                #como sendo do alpha med atual ate o alpha upper atual
                alpha_lower = alpha_med
                intervalo[0] = alpha_lower

                #chama novamente o metodo com o novo intervalo de busca
                return  bissecao(intervalo, sentido_busca, P1, f, tol)
            else:
                #caso negativo, define novo intervalo
                #de busca como sendo do alpha lower atual ate o alpha med atual
                alpha_upper = alpha_med
                intervalo[1] = alpha_upper
                #chama novamente o metodo com o novo intervalo de busca
                return  bissecao(intervalo, sentido_busca, P1, f, tol)
```

2.1.3 Seção Áurea

Implementado um método que recebe como parâmetros de entrada um intervalo de busca ($\overrightarrow{interv} = [\alpha^L, \alpha^U]$), um vetor unitário no sentido correto de busca, um ponto inicial, a função f e um parâmetro opcional para a tolerância de convergência com valor default de 0.00001.

O intervalo de busca e o vetor unitário no sentido da busca podem ser livremente fornecidos ou então estimados pelo método do passo constante. Resumidamente, o método utiliza a razão áurea ($R_a = \frac{\sqrt{5}-1}{2}$) para comparar os valores de $f(\overrightarrow{P_1} + \alpha_E\, \hat{sentido})$ e $f(\overrightarrow{P_1} + \alpha_D\, \hat{sentido})$, 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 os seguintes valores:

1. $\overrightarrow{P_{min}}$, tal que $\overrightarrow{P_{min}} = \overrightarrow{P_1} + \alpha_{min}\, \hat{sentido}$
2. 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 algortimo, quando a convergência foi obtida

```
def secao_aurea(intervalo, sentido_busca, P1, f, tol=0.00001):
    #linear search pelo metodo da secao aurea

    #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 = (math.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(P1 + alpha_e*sentido_busca)
    f2 = f(P1 + 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(P1 + 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(P1 + alpha_e*sentido_busca)

    # calcula Pmin e alpha min apos convergencia
    alpha_med = (alpha_lower + alpha_upper)/2
    alpha_min = alpha_med
    Pmin = P1 + alpha_min*sentido_busca

    return Pmin, alpha_min
```

2.2 Exercício 2

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

1. Função 1:
$$f(x_1, x_2) = x_1^2 - 3x_1x_2 + 4x_2^2 + x_1 - x_2 \;\; com\; \overrightarrow{P_1} = \begin{Bmatrix} 1 \\ 2 \end{Bmatrix} \;\; e \;\; \overrightarrow{d} = \begin{Bmatrix} -1 \\ -2 \end{Bmatrix}$$

2. Função 2 - McCormick:
$$f(x_1, x_2) = \sin(x_1 + x_2) + (x_1 - x_2)^2 - 1.5x_1 + 2.5x_2 \;\; com\; \overrightarrow{P_1} = \begin{Bmatrix} -2 \\ 3 \end{Bmatrix} \;\; e \;\; \overrightarrow{d} = \begin{Bmatrix} 1.453 \\ -4.547 \end{Bmatrix}$$

3. Função 3 - Himmelblau:
$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \;\; com\; \overrightarrow{P_1} = \begin{Bmatrix} 0 \\ 5 \end{Bmatrix} \;\; e \;\; \overrightarrow{d} = \begin{Bmatrix} 3 \\ 1.5 \end{Bmatrix}$$

- 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.

2.2.1 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()
```

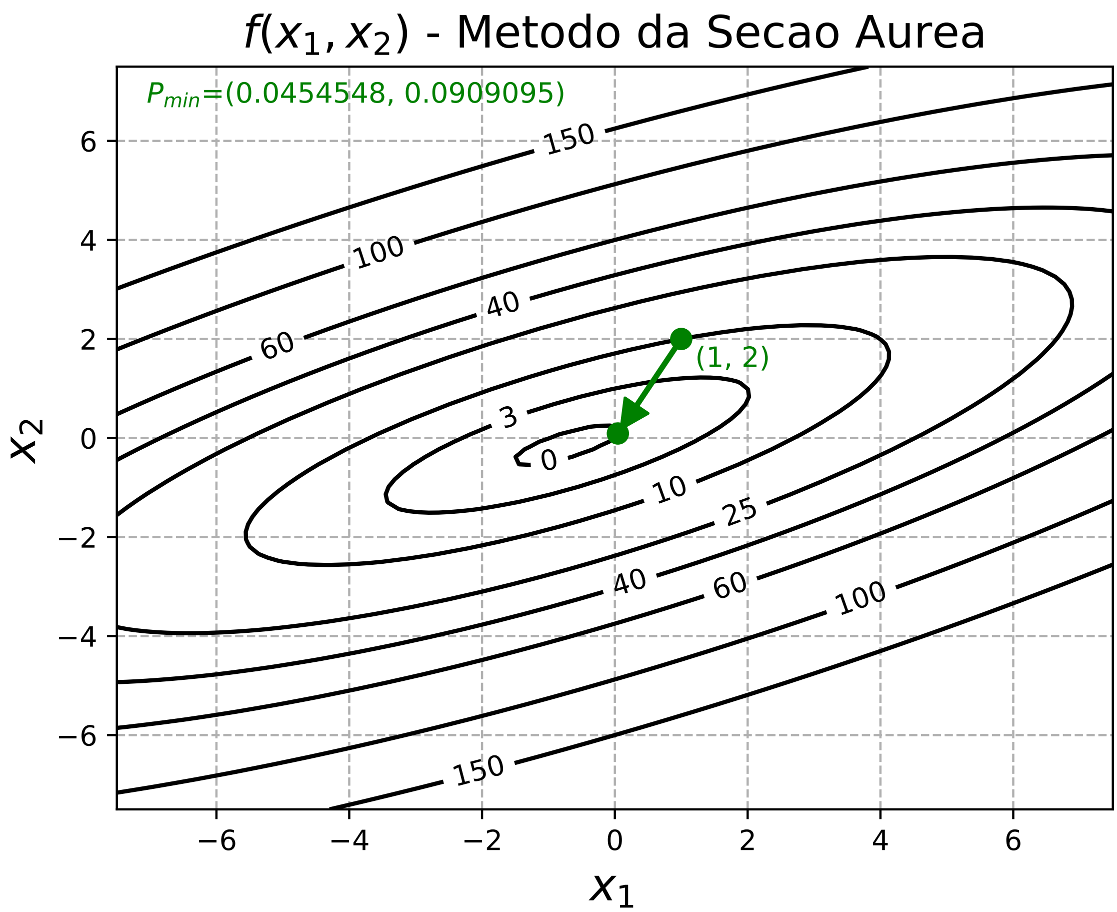

Figura 1: Função 1 - Seção Áurea

2.2.2 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
- Bisseção
- Seção Áurea

```python
#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: $|\alpha_{min}| = 4.77357910169$ 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.

```python
#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()
```

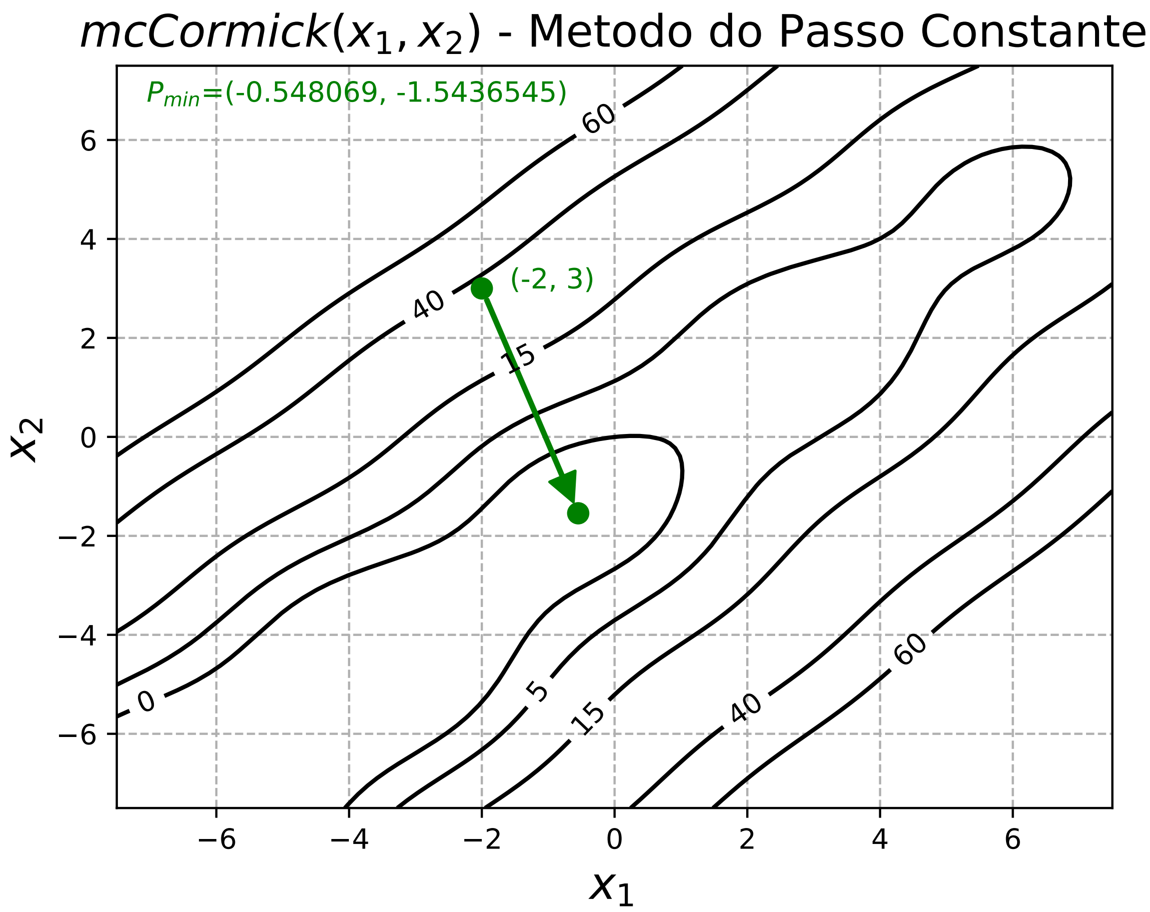

Figura 2: McCormick - Passo Constante

2.2.3 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
- Bisseção
- 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)
```

```python
#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'Secao 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)
Bisseção : $|\alpha\ min|$ = 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 $\overrightarrow{P_{min}} = \begin{Bmatrix} -3.0321081775 \\ 3.4839459113 \end{Bmatrix}$
- Bisseção: $|\alpha_{min}| = 3.3921630859$ 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 Bisseção.

```python
#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 Bissecao", fontsize='16')
plt.savefig("C_solution.pdf", format="pdf")
plt.show()
```

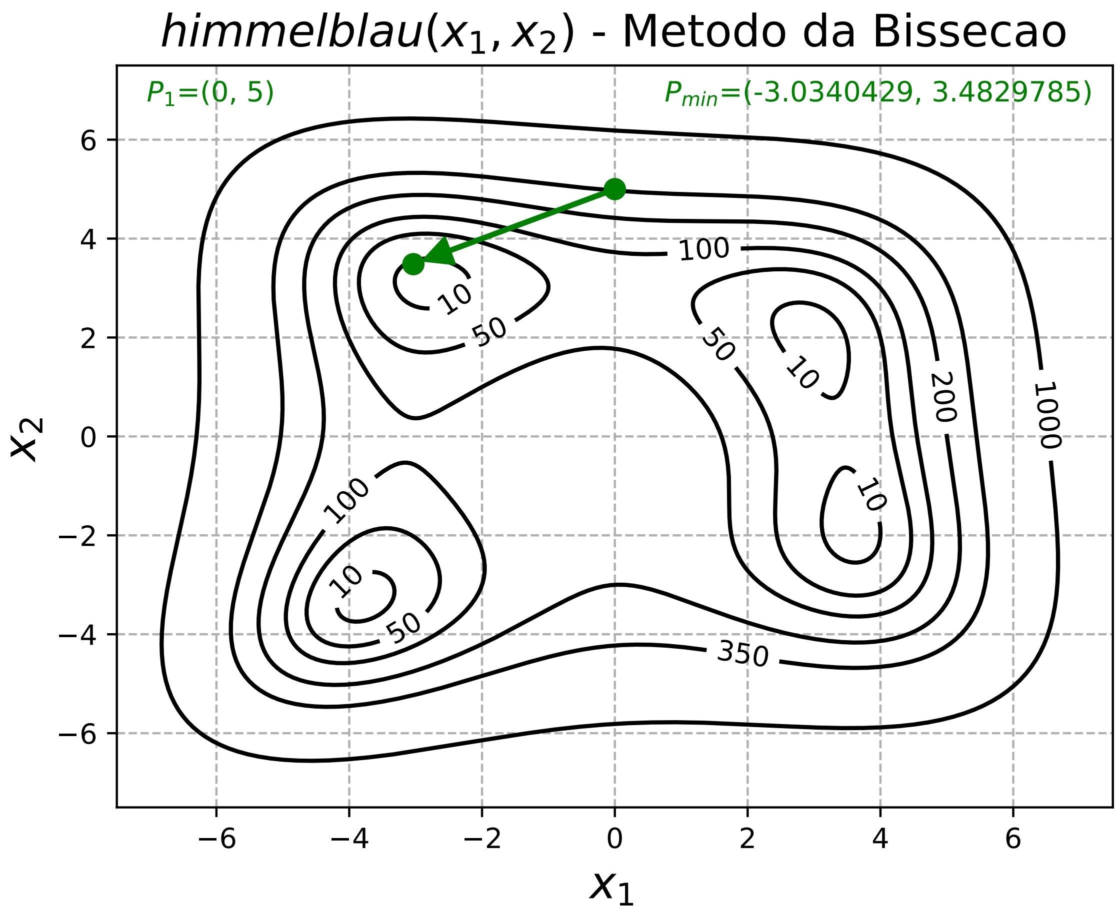

Figura 3: Himmelblau - Bisseção