Projeto 4 - Cálculo

Prof. Dr. Sérgio Garavelli

Rutiele da Silva Oliveira (RA 22353619), Suênia Araújo (RA 22353529) Glaucia Calazans (RA 22354149) e Gerciane Cabral (22352792)

1. **Um fabricante tem um pedido de 2.000 unidades de um pneu de veículo para todos os terrenos que pode ser produzido em duas fábricas. Sejam x_1 e x_2 os números de unidades produzidas nas duas fábricas. A função custo é modelada por**

$$C = 0{,}25x_1^2 + 10x_1 + 0{,}15x_2^2 + 12x_2$$

Encontre o número de unidades que deveriam ser produzidas em cada fabrica para minimizar o custo.

Para resolver este problema, precisamos encontrar os valores de $x1$ e $x2$ que minimizam a função de custo C sujeita à restrição $x1$ + $x2$ =2000.

```
from scipy.optimize import minimize

# Defina a função de custo
# Esta função calcula o valor da função de custo C com base nos valores de x1 e x2
def custo(vars):
    x1, x2 = vars
    return 0.25 * x1**2 + 10 * x1 + 0.15 * x2**2 + 12 * x2

# Defina a restrição
# Esta função define a restrição que a soma de x1 e x2 deve ser 2000
def restricao(vars):
    x1, x2 = vars
    return 2000 - (x1 + x2)

# Chute inicial para x1 e x2
# Fornece valores iniciais de x1 e x2 para o algoritmo de otimização começar
x0 = [1000, 1000]

# Defina as restrições no formato exigido pela função minimize
# 'type': 'eq' indica que é uma restrição de igualdade
# 'fun': restricao refere-se à função de restrição que definimos acima
cons = ({'type': 'eq', 'fun': restricao})

# Chame a função minimize
# minimize tenta encontrar os valores de x1 e x2 que minimizam a função de custo
# sujeita às restrições fornecidas
sol = minimize(custo, x0, constraints=cons)

# Resultados
# Extraímos os valores ótimos de x1 e x2 da solução retornada por minimize
x1_opt, x2_opt = sol.x

# Imprimir os valores ótimos de x1 e x2
print(f"x1 ótimo: {x1_opt}")
print(f"x2 ótimo: {x2_opt}")
```

```
x1 ótimo: 752.499998555153
x2 ótimo: 1247.5000014448472
```

2. **Um fabricante de embalagens de leite (em forma de caixas retangulares) deseja fabricar as embalagens de modo a maximizar o volume do seu conteúdo. No entanto, ele deseja fixar o custo de cada embalagem em 2 reais, sendo que o custo das tampas superior e inferior é 4 reais por unidade de área e o custo das faces laterais é 3 reais por unidade de área. Calcule as dimensões e o volume das embalagens a serem fabricadas.**

Explicações:

A função objective_function define a função objetivo que queremos maximizar, que é o volume da embalagem. Ela recebe as dimensões da embalagem como entrada e retorna o volume calculado a partir dessas dimensões.

A função cost_constraint define a restrição de custo que deve ser respeitada durante a otimização. Ela recebe as dimensões da embalagem e retorna a diferença entre o custo total das tampas e faces laterais e o custo fixado de 2 reais.

initial_guess é uma lista que contém a estimativa inicial das dimensões da embalagem.

bounds é uma tupla que define os limites para as dimensões da embalagem. Neste caso, estamos permitindo dimensões não negativas, então o limite inferior é 0.

minimize é a função principal do SciPy que encontra o mínimo de uma função sujeita a restrições. Passamos para ela a função objetivo, a estimativa inicial, os limites e a restrição de custo.

```
from scipy.optimize import minimize

# Função a ser minimizada
def objective_function(dimensions):
    x, y, z = dimensions
    return -x * y * z  # Negativo porque queremos maximizar o volume

# Restrição de custo
def cost_constraint(dimensions):
    x, y, z = dimensions
    return 8*x*z + 6*(x*y + y*z) - 2  # Restrição de custo

# Condição inicial para as dimensões
initial_guess = [1, 1, 1]

# Limites para as dimensões (x, y, z)
bounds = ((0, None), (0, None), (0, None))

# Resolver o problema de otimização
result = minimize(objective_function, initial_guess, bounds=bounds, constraints={'type': 'eq', 'fun': cost_constraint})

# Extrair as dimensões e o volume máximo
x_max, y_max, z_max = result.x
volume_max = -result.fun  # O valor máximo é negativo, então invertemos o sinal

# Imprimir os resultados
print("Dimensões da embalagem (metros):")
print(f"Largura (x): {x_max:.2f}")
print(f"Altura (y): {y_max:.2f}")
print(f"Profundidade (z): {z_max:.2f}")
print(f"Volume máximo: {volume_max:.2f} metros cúbicos")
```

```
Dimensões da embalagem (metros):
Largura (x): 0.29
Altura (y): 0.38
Profundidade (z): 0.29
Volume máximo: 0.03 metros cúbicos
```

```python
from scipy.optimize import minimize

# Função a ser minimizada
def objective_function(dimensions):
    x, y, z = dimensions
    return -x * y * z  # Negativo porque queremos maximizar o volume

# Restrição de custo
def cost_constraint(dimensions):
    x, y, z = dimensions
    return 8*x*z + 6*(x*y + y*z) - 2  # Restrição de custo

# Condição inicial para as dimensões
initial_guess = [1, 1, 1]

# Limites para as dimensões (x, y, z)
bounds = ((0, None), (0, None), (0, None))

# Resolver o problema de otimização
result = minimize(objective_function, initial_guess, bounds=bounds, constraints={'type': 'eq', 'fun': cost_constraint})

# Extrair as dimensões e o volume máximo
x_max, y_max, z_max = result.x
volume_max = -result.fun  # O valor máximo é negativo, então invertemos o sinal

# Converter para centímetros e milímetros
x_max_cm, y_max_cm, z_max_cm = [dim * 100 for dim in (x_max, y_max, z_max)]
volume_max_mm3 = volume_max * 1e9  # 1 metro cúbico = 1e9 milímetros cúbicos

print("Dimensões da embalagem (centímetros):")
print(f"Largura (x): {x_max_cm:.2f}")
print(f"Altura (y): {y_max_cm:.2f}")
print(f"Profundidade (z): {z_max_cm:.2f}")
print(f"Volume máximo: {volume_max_mm3:.2f} milímetros cúbicos")
```

```
Dimensões da embalagem (centímetros):
Largura (x): 28.87
Altura (y): 38.49
Profundidade (z): 28.87
Volume máximo: 32075014.90 milímetros cúbicos
```

3. **Suponha que em uma região do espaço o potencial elétrico seja dado por**

$$V(x,y,z) = 5x^2 - 3xy + xyz$$

Determine a taxa de variação do potencial no ponto $P = (3,3,5)$ na direção do vetor $\vec{v} = (1,1,-1)$.

a) **Em que direção o potencial varia mais rapidamente em P?**
b) **E qual é a taxa máxima de variação em P?**

Passos no código:

Calculando o gradiente no ponto P: Calculamos o gradiente no ponto P usando a função grad_V.

Normalizando o vetor: Calculamos o vetor unitário na direção de v.

Calculando a derivada direcional: Calculamos o produto escalar entre o gradiente no ponto e o vetor unitário.

Calculando a taxa máxima de variação: Calculamos o módulo do gradiente.

Direção da máxima variação: Normalizamos o gradiente para obter a direção da máxima variação.

```
import numpy as np

# Definimos a função grad_V para calcular as derivadas parciais do potencial V
def grad_V(x, y, z):
    dV_dx = 10*x - 3*y + y*z  # Derivada parcial em relação a x
    dV_dy = -3*x + x*z       # Derivada parcial em relação a y
    dV_dz = x*y              # Derivada parcial em relação a z
    return np.array([dV_dx, dV_dy, dV_dz])  # Retorna o vetor gradiente

# Ponto P onde vamos calcular as informações
P = np.array([3, 3, 5])

# Vetor v na direção dada
v = np.array([1, 1, -1])

# Calculando o gradiente no ponto P
gradiente_P = grad_V(P[0], P[1], P[2])
print(f"Gradiente no ponto P: {gradiente_P}")

# Normalizando o vetor v para obter o vetor unitário na direção de v
v_unitario = v / np.linalg.norm(v)
print(f"Vetor unitário na direção de v: {v_unitario}")

# Calculando a derivada direcional do potencial na direção de v
# É o produto escalar entre o gradiente e o vetor unitário
derivada_direcional = np.dot(gradiente_P, v_unitario)
print(f"Derivada direcional na direção de v: {derivada_direcional}")

# Calculando a taxa máxima de variação, que é o módulo do gradiente
taxa_max_variacao = np.linalg.norm(gradiente_P)
print(f"Taxa máxima de variação no ponto P: {taxa_max_variacao}")

# A direção em que o potencial varia mais rapidamente é a direção do gradiente
# Normalizamos o gradiente para obter essa direção
direcao_max_variacao = gradiente_P / np.linalg.norm(gradiente_P)
print(f"Direção em que o potencial varia mais rapidamente em P: {direcao_max_variacao}")
```

```
Gradiente no ponto P: [36  6  9]
Vetor unitário na direção de v: [ 0.57735027  0.57735027 -0.57735027]
Derivada direcional na direção de v: 19.052558883257653
Taxa máxima de variação no ponto P: 37.589892258425
Direção em que o potencial varia mais rapidamente em P: [0.95770426 0.15961738 0.23942607]
```

4. **Seja a temperatura de um disco circular de raio 1 dada por**

$$T = y - 2x^2 - y^2$$

a) **Encontre o maior valor de T dentro do disco.**
b) **Encontre o maior valor de T na borda do disco.**

Para encontrar o maior valor da função de temperatura (T(x, y) = y - 2x^2 - y^2) dentro e na borda de um disco de raio 1, analisamos os pontos críticos da função dentro do disco, calculando suas derivadas parciais e resolvendo o sistema resultante. Encontramos o ponto crítico ((0, \frac{1}{2})), onde (T = \frac{1}{4}). Para a borda do disco, parametrizamos a circunferência usando coordenadas polares e reescrevemos (T) em termos de (\theta). Avaliamos essa nova função ao longo de (\theta \in [0, 2\pi]) para encontrar o valor máximo. A combinação dessas análises permite identificar o maior valor de (T) tanto dentro quanto na borda do disco.

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt

# Definindo a função de temperatura T(x, y)
def T(x, y):
    return y - 2*x**2 - y**2

# Definindo as derivadas parciais de T
def grad_T(x, y):
    dT_dx = -4*x
    dT_dy = 1 - 2*y
    return np.array([dT_dx, dT_dy])

# Função objetivo para minimização negativa de T
def objective_func(point):
    x, y = point
    return -T(x, y)

# Constraints para o disco de raio 1
constraints = ({'type': 'ineq', 'fun': lambda point: 1 - np.sqrt(point[0]**2 + point[1]**2)})
```

```
# Initial guess
initial_guess = np.array([0.0, 0.0])

# Encontrando o máximo de T dentro do disco
result_inside = minimize(objective_func, initial_guess, constraints=constraints)

max_T_inside = -result_inside.fun
max_T_inside_point = result_inside.x

# Encontrando o máximo de T na borda do disco
theta = np.linspace(0, 2*np.pi, 1000)
x_borda = np.cos(theta)
y_borda = np.sin(theta)
T_borda = T(x_borda, y_borda)
max_T_borda = np.max(T_borda)
max_T_borda_point = (x_borda[np.argmax(T_borda)], y_borda[np.argmax(T_borda)])

print(f"Maior valor de T dentro do disco: {max_T_inside} no ponto {max_T_inside_point}")
print(f"Maior valor de T na borda do disco: {max_T_borda} no ponto {max_T_borda_point}")

# Plotando o disco e os pontos de máximo
fig, ax = plt.subplots()
disk = plt.Circle((0, 0), 1, color='b', alpha=0.1)
ax.add_artist(disk)
ax.plot(max_T_inside_point[0], max_T_inside_point[1], 'ro', label='Máximo dentro do disco')
ax.plot(max_T_borda_point[0], max_T_borda_point[1], 'go', label='Máximo na borda do disco')
ax.set_xlim([-1.2, 1.2])
ax.set_ylim([-1.2, 1.2])
ax.set_aspect('equal', 'box')
ax.set_title('Distribuição da Temperatura no Disco')
ax.legend()
plt.show()
```

```
Maior valor de T dentro do disco: 0.24999999999999956 no ponto [-1.49011614e-08  5.00000000e-01]
Maior valor de T na borda do disco: -3.7085126796121415e-06 no ponto (-0.001572368047584414, 0.9999987638285974)
```