Exercícios de optimização

juliane.marubayashi@gmail.com

December 2019

Os exercícios neste documento são baseados nos dados em sala de aula

1 Exercicio: secção áurea

Utilizando o método da secção áurea, ache o valor mínimo e máximo da equação a seguir no intervalo [-1,0]:

$$f(x) = (2x+1)^2 - 5cos(10x)$$

2 Exercicio: gradiente

Utilizando o método do gradiente encontre o mínimo da função a seguir no com h = 1, x0 = 1 e y0 = 1:

$$f(x,y) = y^2 - 2xy - 6y + 2x^2 + 12$$

3 Exercicio: Quádrica

Utilizando o método da quádrica, encontre o mínimo da equação seguinte considerando x0= 0 e y0= 0:

$$f(x,y) = sin(y) + \frac{y^2}{4} + cos(x) + \frac{x^2}{4} - 1$$

Resposta: x = 0, y = -1,02987, f(x,y) = -0,59207

4 Exercicio: Levemberg Marquardt

Utilizando o método de Levemberg Marquardt, encontre o mínimo da equação a seguir considerando lambda = 1, x0 = 1, y0 = 1:

$$f(x,y) = y^2 - 2xy - 6y + 2x^2 + 12$$

Resposta: x = 2,9692650, y = 5,950269, f(x,y) = -5,998

5 Resposta - exercicio 1 - python

```python
import math as m


def f(x):
    return (2 * x + 1) ** 2 - 5 * m.cos(10 * x)


def aurea_max(x1, x2):
    b = (m.sqrt(5) - 1) / 2
    a = b * b
    for i in range(30):
        x3 = x1 + a * (x2 - x1)
        x4 = x1 + b * (x2 - x1)
        if f(x3) > f(x4):
            x2 = x4
            x4 = x3
        else:
            x1 = x3
            x3 = x4
    return [x1, x2, x3, x4]


def aurea_min(x1, x2):
    b = (m.sqrt(5) - 1) / 2
    a = b * b
    for i in range(30):
        x3 = x1 + a * (x2 - x1)
        x4 = x1 + b * (x2 - x1)
        if f(x3) < f(x4):
            x2 = x4
            x4 = x3
        else:
            x1 = x3
            x3 = x4
    return [x1, x2, x3, x4]


print(aurea_max(-1, 0))
#Expected result:
#[-0.31113734222759837, -0.3111368047370985, -0.311137136924496, -0.311137136924496]
print(aurea_min(-1, 0))
#Expected result:
#[-0.6262978964093815, -0.6262973589188816, -0.6262976911062791, -0.6262976911062791]
```

6 Resposta - exercicio 2 - python

```python
import math as m


def f(x, y):
    return y * y - 2 * x * y - 6 * y + 2 * x * x + 12


def dfx(x, y):
    return 4 * x - 2 * y


def dfy(x, y):
    return 2 * y - 2 * x - 6


def gradiente(xn, yn, h):
    for i in range(30):
        x = xn - h * dfx(xn, yn)
        y = yn - h * dfy(xn, yn)
        if f(x, y) < f(xn, yn):
            h *= 2
            xn = x
            yn = y
        else:
            h /= 2
    return [x, y]


print(gradiente(1, 1, 1))
#Expected result:
#[2.9765625, 5.984375]
```

7 Resposta - exercicio 3 - python

```python
import math as m


def f(x, y):
    return m.sin(y) + y * y / 4 + m.cos(x) + x * x / 4 - 1


def dfx(x, y):
    return x / 2 - m.sin(x)


def dfy(x, y):
    return m.cos(y) + y / 2


def dfxx(x, y):
    return 1 / 2 - m.cos(x)


def dfyy(x, y):
    return 1 / 2 - m.sin(y)


def dfxy(x, y):
    return 0


def dfyx(x, y):
    return 0


def quadratica(xn, yn):
    for i in range(30):
        det = dfyy(xn, yn) * dfxx(xn, yn) - dfyx(xn, yn) * dfxy(xn, yn)
        x = xn - (dfyy(xn, yn) * dfx(xn, yn) - dfyx(xn, yn) * dfx(xn, yn)) / det
        y = yn - (-dfxy(xn, yn) * dfx(xn, yn) + dfxx(xn, yn) * dfy(xn, yn)) / det
        xn = x
        yn = y
    return [x, y]


print(quadratica(0, 0))
```

8 Resposta - exercicio 4 - python

```python
import math as m


def f(x, y):
    return y * y - 2 * x * y - 6 * y + 2 * x * x + 12


def dfx(x, y):
    return 4 * x - 2 * y


def dfy(x, y):
    return 2 * y - 2 * x - 6


def dfxx(x, y):
    return 4


def dfyy(x, y):
    return 2


def dfxy(x, y):
    return -2


def dfyx(x, y):
    return -2


def levemberg(xn, yn, lamb):
    for i in range(20):
        det = dfyy(xn, yn) * dfxx(xn, yn) - dfyx(xn, yn) * dfxy(xn, yn)
        x = xn - (dfyy(xn, yn) * dfx(xn, yn) - dfyx(xn, yn) * dfy(xn, yn)) / det - lamb *
            dfx(xn, yn)
        y = yn - (-dfxy(xn, yn) * dfx(xn, yn) + dfxx(xn, yn) * dfy(xn, yn)) / det - lamb *
            dfy(xn, yn)
        if (x - xn <= 0) and (y - yn <= 0):
            xn = x
            yn = y
        lamb /= 2
    return [x, y]


print(levemberg(1, 1, 1))
#Expected result [2.9999961853027344, 6.000011444091797]
```