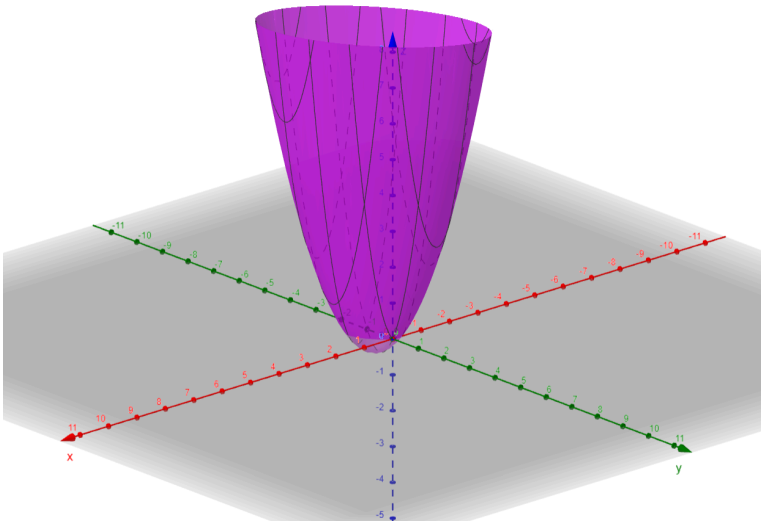
MiniProjeto2 – Otimização pelo vetor gradiente
Grupo 2: Hector, Giovanni e Pedro Garcia

TAREFA 1

a)



b)

$$f(x,y) = 3x^2 + 3xy + 2y^2 + x + 2y$$

$$6x + 3y + 0 + 1 + 0$$
$$0 + 3x + 4y + 0 + 2$$

$$\vec{\nabla} f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (6x + 3y + 1, 3x + 4y + 2)$$

c)

```python
import numpy as np

def f(x, y):
    return 3*x**2 + 3*x*y + 2*y**2 + x + 2*y

def gradiente(x, y):
    df_dx = 6*x + 3*y + 1
    df_dy = 3*x + 4*y + 2
```

```
    return np.array([df_dx, df_dy])

def gradiente_descendente(alpha, x0=0, y0=0, tol=1e-5, max_iter=1000000):
    x, y = x0, y0
    for i in range(max_iter):
        grad = gradiente(x, y)
        norma = np.linalg.norm(grad)

        if norma < tol:
            break

        x = x - alpha * grad[0]
        y = y - alpha * grad[1]

    return x, y, f(x, y), i + 1, norma

valores_alpha = [0.1, 0.15, 0.2, 0.3, 0.5]
for alpha in valores_alpha:
    x, y, f_val, it, norma = gradiente_descendente(alpha)
    if np.isnan(x) or np.isinf(x):
        print(f"α = {alpha:.2f} → DIVERGIU")
    else:
        print(f"α = {alpha:.2f} | x = {x:.6f}, y = {y:.6f}, f =
{f_val:.6f}, "f"iterações = {it}, ||grad|| = {norma:.2e}")
```

d)

Resultados obtidos:

α = 0.10 | x = 0.133330, y = -0.599996, f = -0.533333, iterações = 58, ||grad|| = 9.76e-06
α = 0.15 | x = 0.133330, y = -0.599996, f = -0.533333, iterações = 37, ||grad|| = 9.42e-06
α = 0.20 | x = 0.133331, y = -0.599999, f = -0.533333, iterações = 28, ||grad|| = 9.49e-06
α = 0.30 → DIVERGIU
α = 0.50 → DIVERGIU

Observa-se que, conforme o valor do passo α aumenta, o método do gradiente descendente (mede a intensidade de variação e é usada como critério de parada) converge mais rapidamente até certo limite. Desse modo, o número de iterações diminui - pode-se chegar mais rápido ao resultado

Para α = 0.10, 0.15 e 0.20, o algoritmo converge para o mesmo ponto mínimo aproximado (x ≈ 0.1333, y ≈ -0.6000), com valores de f ≈ -0.5333 e norma do gradiente se aproximando de zero. Porém, ao aumentar o passo para α = 0.30 e α = 0.50, o método diverge, ou seja, ultrapassa o ponto de mínimo procurado, com os valores de x e y indo para infinito.

TAREFA 2

Mínimo 1 (y>0): x ≈ 1.611320, y ≈ 1.283454, g ≈ 3.022240329860, iterações ≈ 138
Mínimo 2 (y<0): x ≈ 1.611320, y ≈ −1.283454, g ≈ 3.022240329860, iterações ≈ 138

Apenas a condição inicial foi alterada, mantivemos x0 = 1 e mudamos o sinal de y0.
- Para o primeiro mínimo: (x0, y0) = (1.0, +1.0)
- Para o segundo mínimo: (x0, y0) = (1.0, −1.0)

Se y0 = 0, o método pode estacionar em um ponto de sela (x ≈ 1.31136, y ≈ 0), portanto usamos y0 ≠ 0.

α = 0.10 | x = 1.311353, y = 0.000000, g = 3.330621, iterações = 47, ||grad|| = 8.57e-06
α = 0.15 | x = 1.311353, y = 0.000000, g = 3.330621, iterações = 29, ||grad|| = 9.23e-06
α = 0.20 | x = 1.311353, y = 0.000000, g = 3.330621, iterações = 20, ||grad|| = 9.05e-06
α = 0.30 | x = 1.311355, y = 0.000000, g = 3.330621, iterações = 11, ||grad|| = 3.06e-06
α = 0.50 | x = 1.311357, y = 0.000000, g = 3.330621, iterações = 10, ||grad|| = 2.16e-06

Conclusão: aumentar α reduz o número de iterações (convergência mais rápida) até certo limite. Para todas as simulações acima, iniciando com y0 = 0, o método converge para o mesmo ponto de sela, apesar da norma do gradiente tender a zero.
Se y0 = 0 → ponto de sela
Se y0 > 0 ou y0 < 0 → convergência para um dos dois mínimos globais simétricos.

TAREFA 3

Máximo 1 (0,0): x=0.000003, y=0.000005, h=6.000007, iterações=3
Máximo 2 (2,3): x=1.907478, y=2.901478, h=3.584081, iterações=16

Só foi necessário alterar o gradiente de descendente para ascendente.
Antes:

```
x = x - α * grad[0]
y = y - α * grad[1]
```

Depois:

```
x = x + α * grad[0]
y = y + α * grad[1]
```

TAREFA DESAFIO

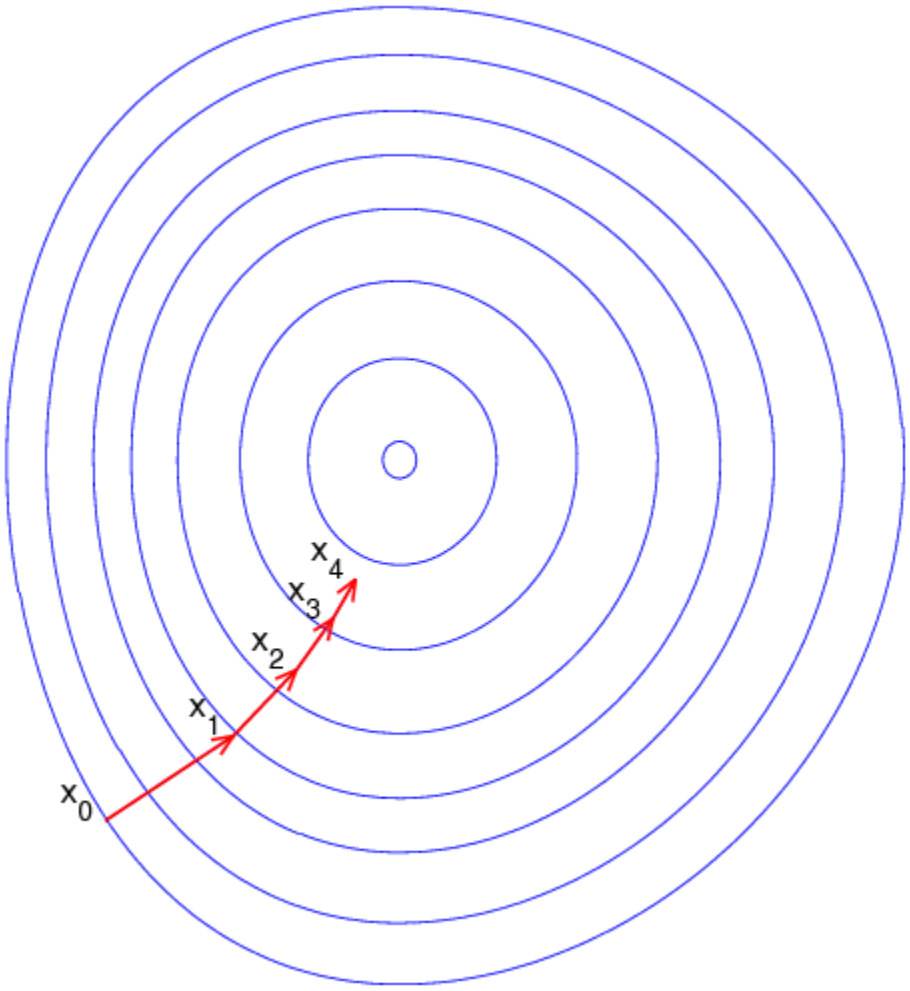
```
import numpy as np

def f(x, y):
    return 3*x**2 + 3*x*y + 2*y**2 + x + 2*y

def grad(x, y):
    return np.array([6*x + 3*y + 1, 3*x + 4*y + 2], float)

H = np.array([[6.0, 3.0],
              [3.0, 4.0]])

def gradiente_descendente_passo_variavel(x0=0.0, y0=0.0, tol=1e-5,
max_iter=100000):
    x, y = float(x0), float(y0)
    it = 0
    g = grad(x, y)

    while np.linalg.norm(g, 2) > tol and it < max_iter:
        g = grad(x, y)
        Hg = H @ g # matriz hessiana x vetor gradiente
        alpha = float(g @ g) / float(g @ Hg)
        x -= alpha * g[0]
        y -= alpha * g[1]
        it += 1

    return x, y, f(x, y), it, np.linalg.norm(grad(x, y), 2), alpha

xv, yv, fv, itv, normv, last_alpha = gradiente_descendente_passo_variavel()
print(f"[Passo variável] x={xv:.6f}, y={yv:.6f}, f={fv:.6f},
iterações={itv}, ||grad||={normv:.2e}, último α={last_alpha:.6f}")
```

[Passo variável] x=0.133333, y=-0.599998, f=-0.533333, iterações=22, ||grad||=5.82e-06, último α=0.312500

Em nossa pesquisa, descobrimos que existem diferentes métodos para calcular passo variável. O método utilizado no código foi Hessian matrix:

A Matriz Hessiana H informa ao algoritmo qual é o "formato" do vale. O algoritmo então usa essa informação para calcular o passo perfeitamente ajustado: pequeno em vales estreitos e grande em vales abertos.

A figura a seguir faz uma boa demonstração disso:



Com passo variável (o tamanho do passo por iteração é definido de forma dinâmica, e não fixa, a partir do cálculo dos gradientes) por line search exato a partir de (0,0), o método atingiu o mesmo mínimo (2/15, −3/5) em ≈22 iterações, reduzindo o número de iterações em relação aos melhores passos fixos testados (28 iterações em α=0.20). Para funções quadráticas com Hessiana conhecida, o passo é fechado e estável, evitando divergências.