Tabela de conteúdo

O problema

Otimize numericamente a função:

$$f_{(x,y)} = cos[3\pi(x+y)]cos[3\pi(x-y)] - x^2 - y^2 + 2(x-y) + 2$$

a) Identifique o tipo de extremo.

b) Ache os valores de x e y para esse extremo e o correspondente valor da função nesse ponto.

Simplificando a função objetivo

Partindo de:

$f_{(x,y)} = cos(3\pi x + 3\pi y) \cdot cos(3\pi x - 3\pi y) - x^2 - y^2 + 2(x-y) + 2$

Expandindo o produto dos cossenos da soma e da diferença:

$f_{(x,y)} = cos^2(3\pi x) \cdot cos^2(3\pi y) - sen^2(3\pi x) \cdot sen^2(3\pi y) - x^2 - y^2 + 2(x-y) + 2$

Distribuindo $2(x-y)$:

$f(x,y) = cos^2(3\pi x) \cdot cos^2(3\pi y) - sen^2(3\pi x) \cdot sen^2(3\pi y) - x^2 - y^2 + 2x - 2y + 2$

Vetor gradiente

O vetor gradiente da função é calculado a partir das derivadas de primeira ordem e nos dá a direção de crescimento da função:

$$\nabla f_{(x,y)} = (f_x, f_y)$$

Calculando as derivadas de primeira ordem:

- f_x

$f_x = -6\pi cos(3\pi x)sen(3\pi x)cos^2(3\pi y) - 6\pi cos(3\pi x)sen(3\pi x)sen^2(3\pi y) - 2x - 0 + 2 - 0 + 0$

$f_x = -6\pi cos(3\pi x)sen(3\pi x)cos^2(3\pi y) - 6\pi cos(3\pi x)sen(3\pi x)sen^2(3\pi y) - 2x + 2$

Colocando em evidência $-6\pi cos(3\pi x)sen(3\pi x)$ **:**

$f_x = -6\pi cos(3\pi x)sen(3\pi x) \cdot (cos^2(3\pi y) + sen^2(3\pi y)) - 2x + 2$

Sabendo que $cos^2(\alpha) + sen^2(\alpha) = 1$ **:**

$f_x = -6\pi cos(3\pi x)sen(3\pi x) - 2x + 2$

Sabendo que $2 \cdot cos(\alpha) \cdot sen(\alpha) = sen(2\alpha)$ **:**

$$f_x = -3\pi sen(6\pi x) - 2x + 2$$

- f_y

$f_y = -cos^2(3\pi x)6\pi cos(3\pi y)sen(3\pi y) - sen^2(3\pi x)6\pi cos(3\pi y)sen(3\pi y) - 0 - 2y + 0 - 2 + 0$

$f_y = -6\pi cos(3\pi y)sen(3\pi y)cos^2(3\pi x) - 6\pi cos(3\pi y)sen(3\pi y)sen^2(3\pi x) - 2y - 2$

Colocando novamente em evidência $-6\pi cos(3\pi x)sen(3\pi x)$ **:**

$f_y = -6\pi cos(3\pi y)sen(3\pi y) \cdot (cos^2(3\pi x) + sen^2(3\pi y)) - 2y - 2$

Novamente, sendo $cos^2(\alpha) + sen^2(\alpha) = 1$ **:**

$f_y = -6\pi cos(3\pi y) sen(3\pi y) - 2y - 2$

Novamente, sendo $2 \cdot cos(\alpha) \cdot sen(\alpha) = sen(2\alpha)$:

$$f_y = -3\pi sen(6\pi y) - 2y - 2$$

Chegamos ao vetor gradiente:

$$\nabla f_{(x,y)} = ([-3\pi sen(6\pi x) - 2x + 2], [-3\pi sen(6\pi y) - 2y - 2])$$

Pontos críticos

Os pontos críticos de uma função são pontos em que a **primeira derivada** se iguala a zero, e de acordo com um **teorema de Fermat**[1], todos os mínimos e máximos locais de uma função contínua ocorrem em pontos críticos.

Considerando $\nabla f_{(x,y)} = 0$, temos o sistema de equações:

$$\begin{cases} -3\pi sen(6\pi x) - 2x + 2 = 0 \\ -3\pi sen(6\pi y) - 2y - 2 = 0 \end{cases}$$

- ### Resolvendo a primeira equação:

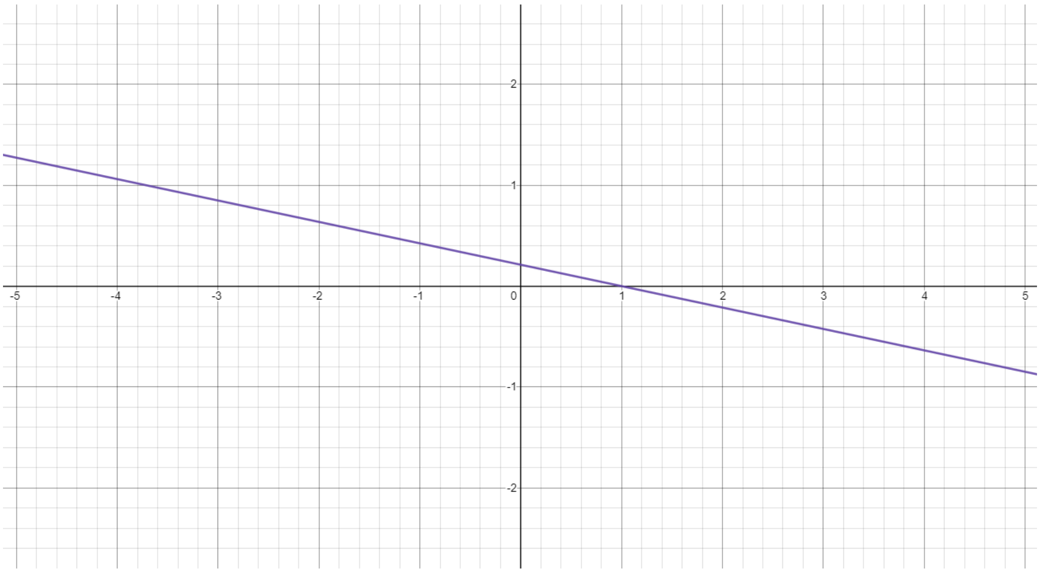
$-3\pi sen(6\pi x) - 2x + 2 = 0$

$2x - 2 = -3\pi sen(6\pi x)$

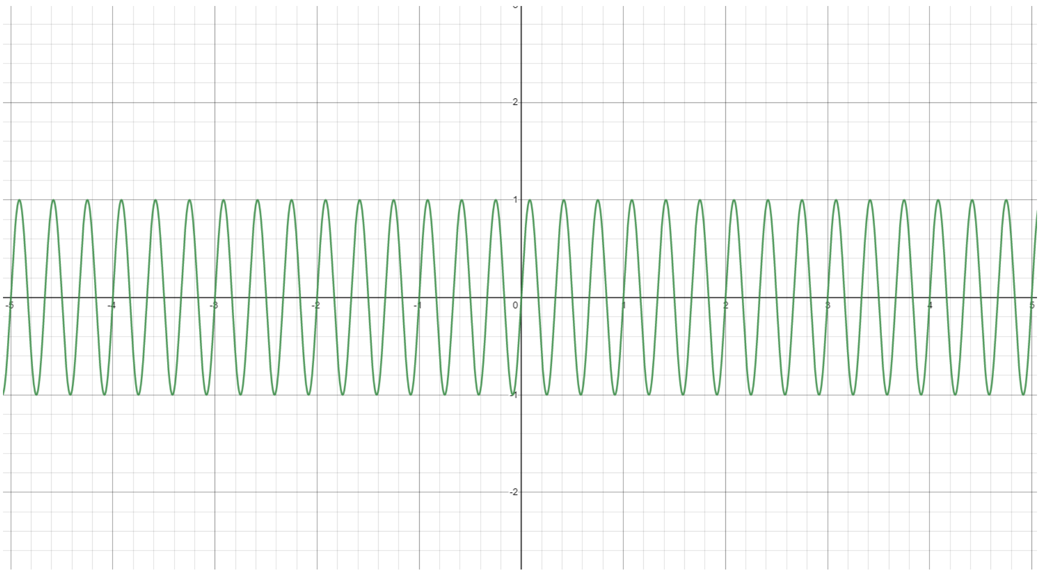
$\frac{(2x-2)}{-3\pi} = sen(6\pi x)$

Fazendo uma análise dos gráficos das funções presentes em cada termo da equação, temos:

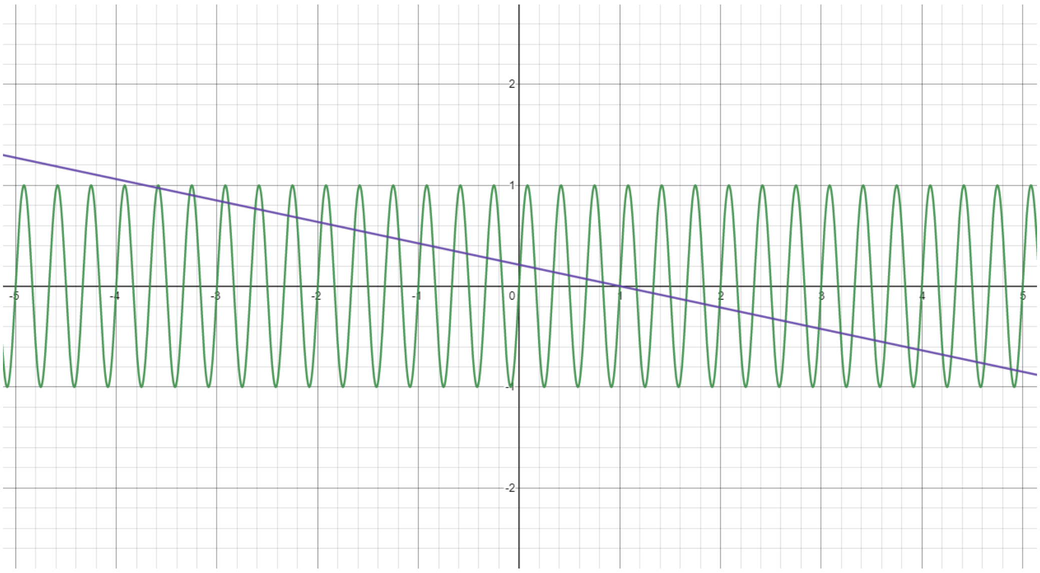
Para $\frac{(2x-2)}{-3\pi}$:



Para $sen(6\pi x)$**:**



Pelas interseções dos gráficos, temos os valores de x **que satisfazem a equação** $\frac{(2x-2)}{-3\pi} = sen(6\pi x)$**:**



$$x = \left\{ \begin{array}{l} -3.595, -3.57, -3.273, -3.226, -2.9473, -2.8847, -2.6202, -2.5452, -2.2923, -2.2064, \\ -1.9639, -1.868, -1.635, -1.53, -1.306, -1.192, -0.977, -0.855, -0.648, -0.517, -0.318, \\ -0.18, 0.011, 0.157, 0.341, 0.494, 0.67, 0.831, 1, 1.169, 1.33, 1.506, 1.659, 1.843, 1.989, 2.18, \\ \quad 2.318, 2.517, 2.648, 2.855, 2.977, 3.192, 3.306, 3.53, 3.635, 3.868, 3.964, 4.206, 4.292, 4.545, \\ \quad 4.62, 4.885, 4.947, 5.226, 5.273, 5.57, 5.595 \end{array} \right\}$$

Resolvendo a segunda equação:

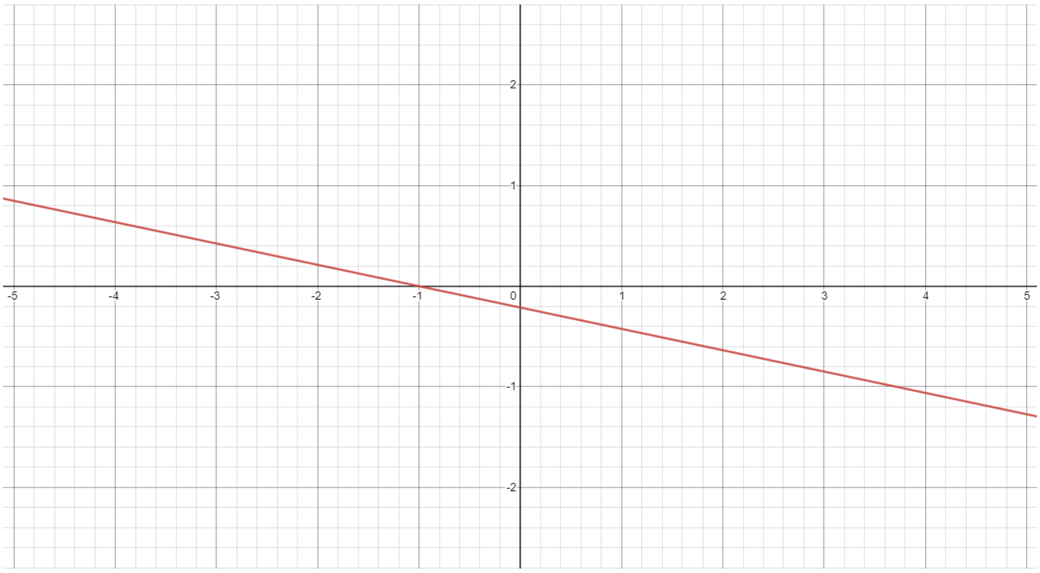
$-3\pi sen(6\pi y) - 2y - 2 = 0$

$2y + 2 = -3\pi sen(6\pi y)$

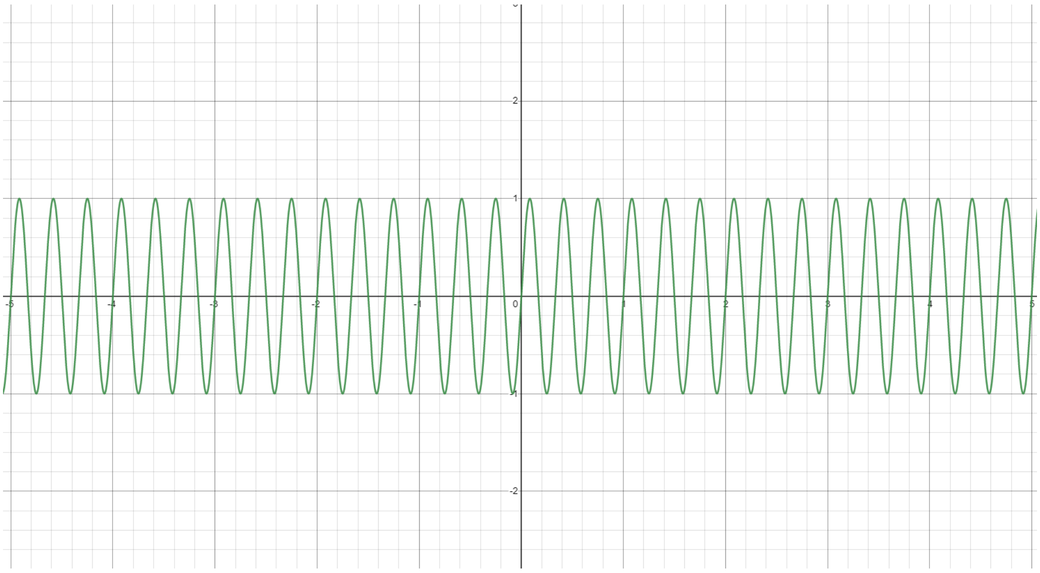
$\frac{(2y+2)}{-3\pi} = sen(6\pi y)$

Fazendo análise gráfica semelhante à anterior:

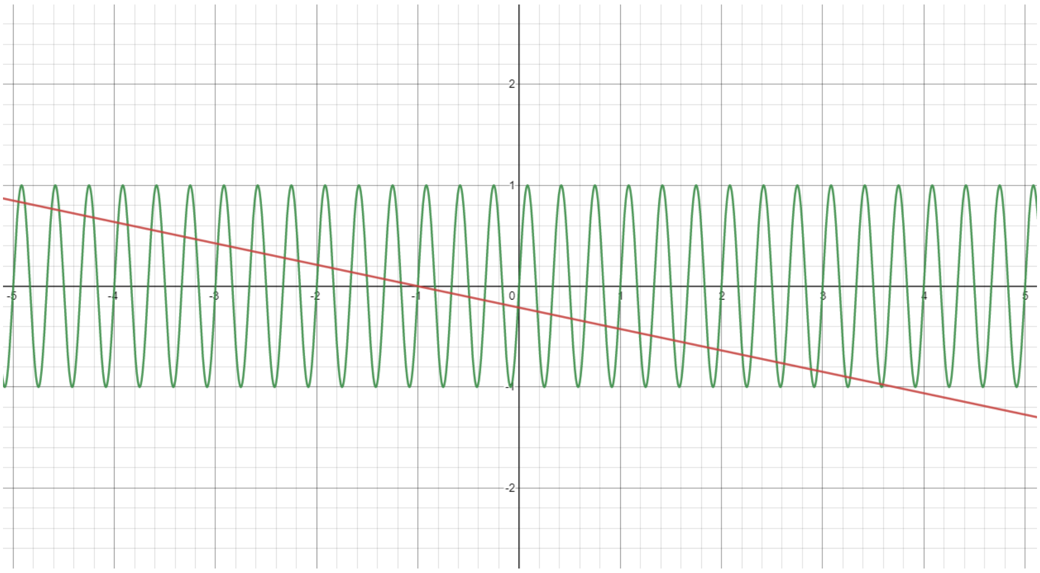
Para $\frac{(2y+2)}{-3\pi}$:



Para $sen(6\pi y)$:



Pelas interseções dos gráficos, temos os valores de y que satisfazem a equação $\frac{(2y+2)}{-3\pi} = sen(6\pi y)$:



$$y = \left\{ \begin{array}{l} -5.595, -5.57, -5.273, -5.226, -4.947, -4.885, -4.62, -4.545, -4.292, -4.206, -3.964, \\ -3.868, -3.635, -3.53, -3.306, -3.192, -2.977, -2.855, -2.648, -2.517, -2.318, -2.18, \\ -1.989, -1.843, -1.659, -1.506, -1.33, -1.169, -1, -0.831, -0.67, -0.494, -0.341, \\ -0.157 - 0.011, 0.18, 0.318, 0.517, 0.648, 0.855, 0.977, 1.192, 1.306, 1.53, 1.635, 1.868, 1.964, \\ \quad 2.206, 2.292, 2.545, 2.62, 2.885, 2.947, 3.226, 3.273, 3.57, 3.595 \end{array} \right\}$$

Matriz Hessiana

A matriz Hessiana $H_{(x,y)}$ é formada pelas seguintes derivadas:

$$H_{(x,y)} = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{bmatrix}$$

A análise do determinante da matriz Hessiana:

$$det\ H_{(x,y)} = f_{xx} \cdot f_{yy} - f_{xy} \cdot f_{yx}$$

nos permite analisar cada ponto crítico da função e determinar se o ponto é um ótimo local ou um ponto de sela[^]:

$$det\ H_{(x,y)} > 0 \Rightarrow f_{xx(x,y)} \begin{cases} > 0 \Rightarrow \text{Mínimo local} \\ < 0 \Rightarrow \text{Máximo local} \end{cases}$$

$$det\ H(x,y) < 0 \Rightarrow \text{Ponto de sela}$$

$$det\ H(x,y) = 0 \Rightarrow \text{Não se pode afirmar}$$

Derivadas da matriz Hessiana:

Partindo de:

$$f_x = -3\pi sen(6\pi x) - 2x + 2$$

$$f_y = -3\pi sen(6\pi y) - 2y - 2$$

Temos:

$$f_{xx} = -18\pi^2 cos(6\pi x) - 2$$

$$f_{yy} = -18\pi^2 cos(6\pi y) - 2$$

$$f_{xy} = 0$$

$$f_{yx} = 0$$

Substituindo em $det\ H_{(x,y)} = f_{xx} \cdot f_{yy} - f_{xy} \cdot f_{yx}$

$$det\ H_{(x,y)} = [-18\pi^2 cos(6\pi x) - 2] \cdot [-18\pi^2 cos(6\pi y) - 2] - 0 \cdot 0$$

$$det\ H_{(x,y)} = [-18\pi^2 cos(6\pi x) - 2] \cdot [-18\pi^2 cos(6\pi y) - 2]$$

Teste dos pontos críticos

Para encontrar os possíveis ótimos locais, calculamos $det\ H_{(x,y)}$ para todos os pontos críticos.

A combinação de todos os valores de x e y encontrados anteriormente nos dá um total de **3249 pontos críticos**. Devido a essa quantidade elevada, foi implementado um programa em **Python** para testar cada um dos pontos e encontrar os pontos ótimos da função.

O código se encontra disponível [no GitHub]() e também no [final deste documento]().

O algoritmo encontrou os seguintes resultados:

```
Maior valor: 5.00000000000000
Ponto: (1, -1)

Menor valor: -38.0184898871649
Ponto: (-3.57, -5.57)
```

Ótimo global

O algoritmo implementado testa apenas os valores de $f_{(x,y)}$ nos pontos críticos, e como encontramos pontos de máximo e de mínimo, precisamos verificar qual deles é o ótimo global, que significa verificar se o valor da função aumenta ou diminui para além destes valores à medida em que as variáveis crescem **em módulo (pois estão elevadas ao quadrado)**.

Para isso, calculamos o **limite** da função quando x e y tendem ao infinito:

$$\lim_{\to\infty} f_{(x,y)}$$

Sabendo que a função $cos(x)$ só pode assumir valores entre -1 e 1 para qualquer valor de x, podemos eliminá-la da função ao calcular o limite, pois esses valores são insignificantes quando somados ao valor do polinômio $-x^2 - y^2 + 2(x - y) + 2$, para $|x| >> 0, |y| >> 0$ (muito grandes).

Sendo assim, temos:

KaTeX parse error: Undefined control sequence: \rvertx at position 52: …nfty \\\lvert y\rvertx\to \infty \\ y…

Isso nos mostra que $f_{(x,y)}$ **não possui valor mínimo**, pois pode assumir valores infinitamente negativos.

Portanto, concluímos que o máximo local $f_{(1,-1)} = 5$, encontrado pelo algoritmo, é também o **ótimo global**, e com isso finalizamos a otimização com as seguintes constatações:

$$\begin{gathered}\text{Ponto extremo} \Rightarrow (1, -1) \\ \text{Tipo de extremo} \Rightarrow \text{Máximo} \\ f_{(1,-1)} = 5\end{gathered}$$

Código Python

Para executar o código abaixo, favor seguir as intruções presentes no [repositório do GitHub](#).

Para visualizar a saída produzida pelo programa, [clique aqui].

```
import sympy as sym
from sympy.abc import x as sym_x
from sympy.abc import y as sym_y
from sympy import oo
from itertools import product

class Problem:
    '''
    This class represents the problem at hand: optimizing the objective function
    f(x,y) = cos[3π(x+y)]cos[3π(x-y)]-x²-y²+2(x-y)+2

    Attributes
    ----------
    objective_function : sympy.Expr()
        a simpy expression which stores the function we want to optimize
    x_set : list[float]
        the list of values for x which compose the critical points
    y_set : list[float]
        the list of values for y which compose the critical points
    critical_points : list[tuple[float, float]]
        list of critical points composed from the values in x_set and y_set
    hessian_list : list[dict[tuple[float, float], float]]
        list of dictionaries with a critical point (x,y) as the key and det H(x,y) as the valu
    local_optima : list[tuple[float, float]]
        list of optimal points for the function, i.e., det H(x,y) > 0
    optima_by_type : dict[str, list[tuple[float, float]]]
        dictionary with a list of local maxima (identified by the key 'maxima') and a list of
        local minima (identified by the key 'minima')
    global_optima : dict[str, dict[(str, tuple[float,float]), (str, float)]]
        dictionary that holds the maximum point with the greatest value, the minimum point
        with the smallest value and their respective values

    Methods
    -------
    get_critical_points(x_set, y_set)
        Combines each value of x with each value of y to produce critical points as tuples
    fxx(x)
        Calculates the second order derivative of the objective function relative to x
        This is sufficient for all second order derivatives in this problem, since its fxx anc
        fyy are the same, and its fxy and fyx are both equal to zero
    det_H(x, y)
        Calculates the determinant of the Hessian matrix for point (x,y)
    get_hessian_list(points)
        Iterates over the list of critical points, calling det_H(x, y) for each of them and
        storing both point and determinant in a dictionary,
        then appends the dictionary to a list
    get_local_optima(hessian_list)
        Checks the sign of each value in the hessian_list and stores its point in a new list i
        the value is positive
    get_optimum_type(local_optima)
        Splits the list of local optimal points into a list of local minima and a list of loca
        maxima
    test_objective(point)
```

```
    Calculates the value of the objective function for the given point
get_global_optima(optima_by_type)
    Iterates over the list of local maxima and stores the one with the greatest value
    then iterates over the list of local minima and stores the one with the smallest value
'''

# cos[3π(x+y)]cos[3π(x-y)]-x²-y²+2(x-y)+2
objective_function = sym.cos(3*sym.pi*(sym_x+sym_y)) # cos[3π(x+y)]
                     *sym.cos(3*sym.pi*(sym_x-sym_y)) # cos[3π(x-y)]
                     -sym_x**2-sym_y**2+2*(sym_x-sym_y)+2 # -x²-y²+2(x-y)+2

x_set = [
-3.595, -3.57,-3.273,-3.226,-2.9473,-2.8847,-2.6202,-2.5452,-2.2923,-2.2064,
-1.9639,-1.868,-1.635,-1.53,-1.306,-1.192,-0.977,-0.855,-0.648,-0.517,-0.318,
0.18,0.011,0.157,0.341,0.494,0.67,0.831,1,1.169,1.33,1.506,1.659,1.843,1.989,2.18,
2.318,2.517,2.648,2.855,2.977,3.192,3.306,3.53,3.635,3.868,3.964,4.206,4.292,4.545,
4.62,4.885,4.947,5.226,5.273,5.57,5.595
]

y_set = [
-5.595,-5.57,-5.273,-5.226,-4.947,-4.885,-4.62,-4.545,-4.292,-4.206,-3.964,
-3.868,-3.635,-3.53,-3.306,-3.192,-2.977,-2.855,-2.648,-2.517,-2.318,-2.18,
-1.989,-1.843,-1.659,-1.506,-1.33,-1.169,-1,-0.831,-0.67,-0.494,-0.341,
-0.157,-0.011,0.18,0.318,0.517,0.648,0.855,0.977,1.192,1.306,1.53,1.635,1.868,1.964,
2.206,2.292,2.545,2.62,2.885,2.947,3.226,3.273,3.57,3.595
]

def get_critical_points(self, x_set, y_set):
    '''
    Returns a list of tuples representing critical points of the objective function

    Parameters
    ----------
    x_set : list[float]
        List of values for the variable x which vanish the first order derivative of the
        objective function
    y_set : list[float]
        List of values for the variable y which vanish the first order derivative of the
        objective function

    Returns
    -------
    critical_points : list[tuple[float,float]]
        List of critical points represented by tuples of floats
    '''

    critical_points = list(product(x_set, y_set))
    return critical_points

def fxx(self, x):
    '''
    Calculates the second order derivative of dependent variable x

    Parameters
    ----------
```

```
        x (float):
            The variable on which the derivative depends

        Returns
        -------
        derivative : float
            Value of the derivative
        '''

        expr = -18*(sym.pi**2)*sym.cos(6*sym.pi*sym_x)-2 # -18π²cos(6πx)-2
        derivative = expr.evalf(subs={sym_x: x})
        return derivative

    def det_H(self, x, y):
        '''
        Calculates the determinant of the Hessian matrix for point (x,y)

        Parameters
        ----------
        x (float):
            x-coordinate of the critical point
        y (float):
            y-coordinate of the critical point

        Returns
        -------
        determinant : float
            Value of the determinant
        '''

        determinant = self.fxx(x)*self.fxx(y)
        return determinant

    def get_hessian_list(self, points):
        '''
        Iterates over the list of critical points, calling det_H(x, y) for each of them and
        storing both point and determinant in a dictionary,
        then appends the dictionary to a list

        Parameters
        ----------
        points (list[tuple[float,float]]):
            List of critical points for which to calculate det H(x,y)

        Returns
        -------
        hessian_list : list[dict[tuple[float,float],float]]
            List of points with their determinants
        '''

        hessian_list = []
        for point in points:
            hessian_list.append({
                point: self.det_H(point[0],point[1])
            })
```

```
        return hessian_list

    def get_local_optima(self, hessian_list):
        '''
        Checks the sign of each value in the hessian_list and stores its point in a new list i
        the value is positive

        Parameters
        ----------
        hessian_list : list[dict[tuple[float,float],float]]
            List of dictionaries of form: {
                                        (x,y): det_H(x,y)
                                    }

        Returns
        -------
        local_optima : list[tuple[float,float]]
            List of points for which det H is positive
        '''

        local_optima = []
        for point in hessian_list:
            (key, value), = point.items()
            if value > 0:
                local_optima.append(key)
        return local_optima

    def get_optimum_type(self, local_optima):
        '''
        Separates the local optima into a list of local minima and a list of local maxima

        Parameters
        ----------
        local_optima : list[tuple[float,float]]
            List of local optimal points

        Returns
        -------
        optima_by_type : dict[str,list[tuple[float,float]]]
            Dictionary of maxima and minima, identified by keys 'maxima' and 'minima',
            respectively
        '''

        maxima = []
        minima = []
        for point in local_optima:
            if self.fxx(point[0]) > 0:
                minima.append(point)
            elif self.fxx(point[0]) < 0:
                maxima.append(point)
        optima_by_type =  {
            'maxima': maxima,
            'minima': minima
        }
        return optima_by_type
```

```
    def test_objective(self, point):
        '''
        Calculates the value of the objective function for the given point

        Parameters
        ----------
        point : tuple[float,float]
            The point (x,y) for which to calculate the value of f(x,y)

        Returns
        -------
        f_value : float
            The value of f(x,y)
        '''

        f_value = self.objective_function.evalf(subs={sym_x: point[0],sym_y: point[1]})
        return f_value

    def get_global_optima(self, optima_by_type):
        '''
        Iterates over the list of local maxima and stores the one with the greatest value
        then iterates over the list of local minima and stores the one with the smallest value

        Parameters
        ----------
        optima_by_type : dict[str,list[tuple[float,float]]]
            Dictionary of maxima and minima, identified by keys 'maxima' and 'minima',
            respectively

        Returns
        -------
        global_optima : dict[str,dict[(str,tuple[float,float]),(str,float)]]
            Dictionary that holds the maximum point with the greatest value,
                                  the minimum point with the smallest value
                                  and their respective values:
                                    {
                                        'maximum': {
                                            'point': (x,y),
                                            'value': self.test_objective((x,y))
                                        }
                                    }
        '''

        maximum = {
            'point': (0, 0),
            'value': -9999999
        }
        minimum = {
            'point': (0, 0),
            'value': 9999999
        }
        for point in optima_by_type['maxima']:
            point_value = self.test_objective(point)
            if point_value > maximum['value']:
```

```
                maximum['point'] = point
                maximum['value'] = point_value
        for point in optima_by_type['minima']:
            point_value = self.test_objective(point)
            if point_value < minimum['value']:
                minimum['point'] = point
                minimum['value'] = point_value
        global_optima = {
            'maximum': maximum,
            'minimum': minimum
        }
        return global_optima

    def __init__(self):
        self.critical_points = self.get_critical_points(self.x_set, self.y_set)
        self.hessian_list = self.get_hessian_list(self.critical_points)
        self.local_optima = self.get_local_optima(self.hessian_list)
        self.optima_by_type = self.get_optimum_type(self.local_optima)
        self.global_optima = self.get_global_optima(self.optima_by_type)


if __name__ == '__main__':
    problem = Problem()
    max_point = problem.global_optima['maximum']['point']
   , max_value = problem.global_optima['maximum']['value']
    min_point = problem.global_optima['minimum']['point']
   , min_value = problem.global_optima['minimum']['value']
    print(f'Maior is {max_value}')
    print(f'Ponto: {max_point}')
    print('')
    print(f'Menor valor: t {min_value}')
    print(f'Ponto: {min_point}')
```

Saída:

```
Maior valor: 5.00000000000000
Ponto: (1, -1)

Menor valor: -38.0184898871649
Ponto: (-3.57, -5.57)
```

[Retornar](#) para o início do código.

---

1. **Teorema de Fermat:** https://en.wikipedia.org/wiki/Fermat's_theorem_(stationary_points) ↩