Projeto 2

Arthur Assis Gonçalves
cc22300@g.unicamp.br

Vinícius Dos Santos Andrade
cc22333@g.unicamp.br

TI327 - Tópicos em Inteligência Artificial
Prof. Dr. Guilherme Macedo
Colégio Técnico de Campinas - UNICAMP

Campinas, 19 de Junho de 2024

Sumário

1 Introdução

A otimização linear, é uma técnica matemática utilizada para maximizar ou minimizar uma função linear sujeita a um conjunto de restrições lineares. Este método encontra aplicações em diversas áreas como economia, engenharia, logística e pesquisa operacional, sendo essencial para a tomada de decisões eficientes e racionais.

Apesar de sua extensa aplicação, a otimização possui algumas limitações. Os problemas relacionados à modelagem, solução e interpretação dos resultados podem impactar significativamente a eficiência e a aplicabilidade dos métodos de otimização linear. Entre os desafios mais comuns estão a complexidade computacional de grandes modelos, a dificuldade em lidar com dados imprecisos ou incompletos e as limitações inerentes das suposições de linearidade.

As funções de otimização são descritas de forma genérica por:

$$\begin{aligned} &\text{minimizar } c^T x \\ &\text{sujeito a } Ax = b \\ &x \in \mathbb{R}^+ \end{aligned} \tag{1}$$

Neste relatório, apresentamos a implementação do método Simplex em Python para resolver problemas de otimização linear, seguido da aplicação desse método em vários problemas de teste.

2 Descrição

Nesta seção, apresentaremos os problemas propostos no Projeto 2 da disciplina TI327 - Tópicos em Inteligência Artificial, ministrada pelo Prof. Dr. Guilherme Macedo. Os respectivos códigos utilizados para cada um dos seis problemas serão detalhados na seção 2.2,

2.1 Descrição 1º problema

$$
\begin{aligned}
\text{Minimizar:} \quad & 5x_1 + x_2 \\
\text{Sujeito a:} \quad & \\
& 2x_1 + x_2 \geq 6, \\
& x_1 + x_2 \geq 4, \\
& x_1 + 5x_2 \geq 10, \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

2.2 Descrição 2º problema

$$
\begin{aligned}
\text{Minimizar:} \quad & 2x_1 - 3x_2 \\
\text{Sujeito a:} \quad & \\
& x_1 + 2x_2 \leq 6, \\
& 2x_1 - x_2 \leq 8, \\
& x_1, x_2 \geq 0.
\end{aligned}
$$

2.3 Descrição 3º problema

$$\begin{aligned} \text{Minimizar:} \quad & 15(x_1 + 2x_2) + 11(x_2 - x_3) \\ \text{Sujeito a:} \quad & \\ & 3x_1 \geq x_1 + x_2 + x_3, \\ & 0 \leq x_j \leq 1, \quad j = 1, 2, 3. \end{aligned}$$

2.4 Descrição 4º problema

$$\begin{aligned} \text{Minimizar:} \quad & 10(x_3 + x_4) \\ \text{Sujeito a:} \quad & \\ & \sum_{j=1}^{4} x_j = 400, \\ & x_j - 2x_{j+1} \geq 0, \quad j = 1, 2, 3, \\ & x_j \geq 0, \quad j = 1, 2, 3. \end{aligned}$$

2.5 Descrição 5º problema

$$\begin{aligned} \text{Maximizar:} \quad & -5x_1 + 3(x_1 + x_3) \\ \text{Sujeito a:} \quad & \\ & x_j + 1 \leq x_{j+1}, \quad j = 1, 2, \\ & \sum_{j=1}^{3} x_j = 12, \\ & x_j \geq 0, \quad j = 1, 2, 3. \end{aligned}$$

2.6 Descrição 7º problema

Maximizar: $9x_1 + 5x_2$

Sujeito a:

$$\sin\left(\frac{k}{13}\right)x_1 + \cos\left(\frac{k}{13}\right)x_2 \leq 7, \quad k = 1, \ldots, 13$$

$$x_1, x_2 \geq 0$$

3 Desenvolvimento

3.1 Código 1º problema

```
from scipy.optimize import linprog
import numpy as np

def problema_1():
    c = [5, 1]
    A = [[-2, -1],
         [-1, -1],
         [-1, -5]]
    b = [-6, -4, -10]

    x0_bounds = (0, None)
    x1_bounds = (0, None)

    res = linprog(
        c,
        A_ub=A,
        b_ub=b,
        bounds=[x0_bounds, x1_bounds],
        method='highs',
        options={'disp': False}
    )

    print(f'A solucao otima deste problema foi x* = ({res.x[0]:.0f}, {res.x[1]:.0f}) com f(x*) = {res.fun:.0f}.')

    plot_2d_problema_1(res)
    plot_3d_problema_1(res)

    return res
```

Este código foi desenvolvido para solucionar o primeiro problema proposto na seção 2 2.1. Adotamos essa abordagem para resolver todos os problemas apresentados, alterando apenas nos casos em que o objetivo era a maximização; para esses, os coeficientes da função objetivo foram invertidos.

O código inicia com a definição dos coeficientes da função objetivo que será minimizada, representada pela lista

$$c = [5, 1],$$

onde a função objetivo é dada por

$$f(x) = 5x_1 + x_2.$$

Em seguida, estabelece-se a matriz A, contendo os coeficientes das restrições de desigualdade, onde cada linha corresponde a uma restrição e cada coluna, a uma variável. As restrições definidas são

$$-2x_1 - x_2 \leq -6,$$
$$-x_1 - x_2 \leq -4,$$
$$-x_1 - 5x_2 \leq -10,$$

que podem ser reescritas como

$$2x_1 + x_2 \geq 6,$$
$$x_1 + x_2 \geq 4,$$
$$x_1 + 5x_2 \geq 10,$$

respectivamente.

O vetor

$$b = [-6, -4, -10]$$

contém os termos independentes das restrições de desigualdade. Os limites das variáveis são definidos pelos pares

$$x0\_bounds = (0, None),$$
$$x1\_bounds = (0, None),$$

garantindo que ambas as variáveis sejam não-negativas, ou seja,

$$x_1 \geq 0 \text{ e } x_2 \geq 0.$$

A resolução do problema de programação linear é realizada através da função `linprog`, utilizando o solver 'highs'. Os parâmetros fornecidos incluem os coeficientes da função objetivo, a matriz de restrições, os termos independentes, os limites das variáveis e o método de otimização escolhido. Após a resolução, a solução ótima é apresentada, onde os valores das variáveis x_1 e x_2 são exibidos juntamente com o valor da função objetivo $f(x)$.

Por fim, o código invoca funções para a plotagem de gráficos bidimensionais e tridimensionais que ilustram a solução ótima encontrada e retorna o objeto contendo os resultados da otimização, que inclui tanto a solução ótima quanto o valor da função objetivo.

3.2 Código 2º problema

```
def problema_2():
    c = [-2, 3]
    A = [[1, 2],
        [2, -1]]
    b = [6, 8]

    x0_bounds = (0, None)  # x1 >= 0
    x1_bounds = (0, None)  # x2 >= 0

    res = linprog(
        c,
        A_ub=A,
        b_ub=b,
        bounds=[x0_bounds, x1_bounds],
        method='highs',
        options={'disp': False}
    )

    print(f'A solucao otima deste problema e x* = ({res.x[0]:.0f}, {res.x[1]:.0f}) com f(x*) = {-res.fun:.0f}.')

    plot_2d_problema_2(res)
    plot_3d_problema_2(res)

    return res
```

3.3 Código 3º problema

```
def problema_3():
    c = [-15, -41, 11]  # Original: 15x1 + 41x2 - 11x3
    A = [[-2, -1, -1]]
    b = [-3]

    x_bounds = [(0, 1), (0, 1), (0, 1)]

    res = linprog(
        c,
        A_ub=A,
        b_ub=b,
        bounds=x_bounds,
        method='highs',
        options={'disp': False}
    )

     print(f'A solucao otima deste problema e x* = ({res.x[0]:.0f}, {res.x[1]:.0f}, {res.x[2]:.0f}) com f(x*) = {-res.fun:.0f}.')

    plot_2d_problema_3(res)
    plot_3d_problema_3(res)

    return res
```

3.4 Código 4º problema

```
def problema_4():
    c = [0, 0, 10, 10]
    A_ub = [
        [-1, 2, 0, 0],
        [0, -1, 2, 0],
        [0, 0, -1, 2]
    ]
    b_ub = [0, 0, 0]

    A_eq = [[1, 1, 1, 1]]
    b_eq = [400]

    bounds = [(0, None), (0, None), (0, None), (0, None)]


    res = linprog(
        c,
        A_ub=A_ub,
        b_ub=b_ub,
        A_eq=A_eq,
        b_eq=b_eq,
        bounds=bounds,
        method='highs',
        options={'disp': False}
    )

    print(f'A solucao otima deste problema e x* = ({res.x[0]:.0f}, {res.x
[1]:.0f}, {res.x[2]:.0f}, {res.x[3]:.0f}) com f(x*) = {res.fun:.0f}.')

    plot_2d_problema_4(res)
    plot_3d_problema_4(res)

    return res
```

3.5 Código 5º problema

```python
def problema_5():
    c = [2, -3, -3]
    A_ub = [
        [1, -1, 0],
        [0, 1, -1]
    ]
    b_ub = [-1, -1]
    A_eq = [[1, 1, 1]]
    b_eq = [12]

    bounds = [(0, None), (0, None), (0, None)]

    res = linprog(
        c,
        A_ub=A_ub,
        b_ub=b_ub,
        A_eq=A_eq,
        b_eq=b_eq,
        bounds=bounds,
        method='highs',
        options={'disp': False}
    )
    if res.success:
        x_opt = res.x
        f_opt = -res.fun
        print(f"A solucao otima deste problema e x* = ({x_opt[0]:.0f}, {x_opt[1]:.0f}, {x_opt[2]:.0f}) com f(x*) = {f_opt:.2f}.")

        plot_2d_problema_5(res)
        plot_3d_problema_5(res)
    else:
        print("O problema de otimizacao nao encontrou uma solucao viavel.")
    return res
```

3.6 Código 7º problema

```
def problema_7():
    c = [-9, -5]
    A_ub = np.array([[np.sin(k / 13), np.cos(k / 13)] for k in range(1, 14)])
    b_ub = np.array([7] * 13)

    bounds = [(0, None), (0, None)]  # x1 >= 0, x2 >= 0

    res = linprog(
        c,
        A_ub=A_ub,
        b_ub=b_ub,
        bounds=bounds,
        method='highs',
        options={'disp': False}
    )[

    ]
    if res.success:
        plot_2d_problema_7(res)
        plot_3d_problema_7(res)

        print(r'A solucao otima deste problema e $x* = ({:.2f}, {:.2f})$ com $f(x*) = {:.2f}$.'.format(res.x[0], res.x[1], -res.fun))
    else:
        print("Otimizacao falhou. Status:", res.status)
        print("Mensagem:", res.message)
    return res
```

4 Resultados

4.1 Testes Computacionais

Os testes foram realizados em dois dispositivos distintos:

- **Notebook**
  - **CPU:** `Intel Core i5 12500H`
  - **RAM:** `8 GB DDR5`
  - **GPU:** `NVIDIA GeForce RTX 3050 Laptop`
- **Desktop**
  - **CPU:** `AMD Ryzen 3 3300X`
  - **RAM:** `2 x 8 GB DDR4`
  - **GPU:** `NVIDIA GeForce GTX 1660 SUPER`

4.2 Experimento

Para a realização dos experimentos, utilizamos a IDE PyCharm Professional 2024.1.3 (build 241.17890.14) com a versão 3.12.3 do Python. A biblioteca central para a análise foi a SciPy 1.13.1, explorando a função `linprog(method='highs')` para a resolução eficiente dos problemas de programação linear.

4.2.1 Ferramentas e Justificativas

A escolha da função `linprog(method='highs')` se deve à sua capacidade de selecionar automaticamente o solver mais adequado entre o método dual simplex revisado (`'highs-ds'`) e o método de ponto interior (`'highs-ipm'`), ambos da biblioteca **HIGHS**. Essa abordagem híbrida garante robustez, precisão e desempenho na resolução de problemas de programação linear, mesmo em larga escala.

Além disso, a função `linprog` oferece uma interface intuitiva para definir a função objetivo, as restrições e os parâmetros do problema, facilitando a implementação e análise dos resultados.

4.2.2 Objetivos

Com este experimento, buscamos:

1. Resolver de forma eficiente e precisa os problemas de otimização linear propostos.
2. Validar a implementação do método Simplex.
3. Avaliar o desempenho da biblioteca SciPy na resolução de problemas de otimização.

4.2.3 Parâmetros e Resultados

Os parâmetros utilizados na função *linprog* foram:

- `c`: Vetor de coeficientes da função objetivo.
- `A_ub`: Matriz de coeficientes das restrições de desigualdade.
- `b_ub`: Vetor de termos independentes das restrições de desigualdade.
- `bounds`: Tupla definindo os limites das variáveis de decisão.
- `method`: Método de resolução (‘highs‘).
- `options`: Opções adicionais, como exibição de mensagens (‘disp‘).

A função `linprog` retorna um objeto *OptimizeResult* contendo:

- `x`: Valores ótimos das variáveis de decisão.
- `fun`: Valor ótimo da função objetivo.
- `success`: Indicador de sucesso na resolução.
- `status`: Código de status da otimização.
- `message`: Mensagem sobre o resultado.

4.3 Solução 1º Problema

A figura 2 representa graficamente o problema em 3D, assim mostrando a região factível, definida pelas restrições, e a superfície da função objetivo. A solução ótima é destacada como um ponto e conectada ao plano xy por uma linha tracejada.

Dessa forma, é possível verificar que a solução ótima para esse problema é:

$$x^* = (0, 6) \text{ com } f(x^*) = 6$$

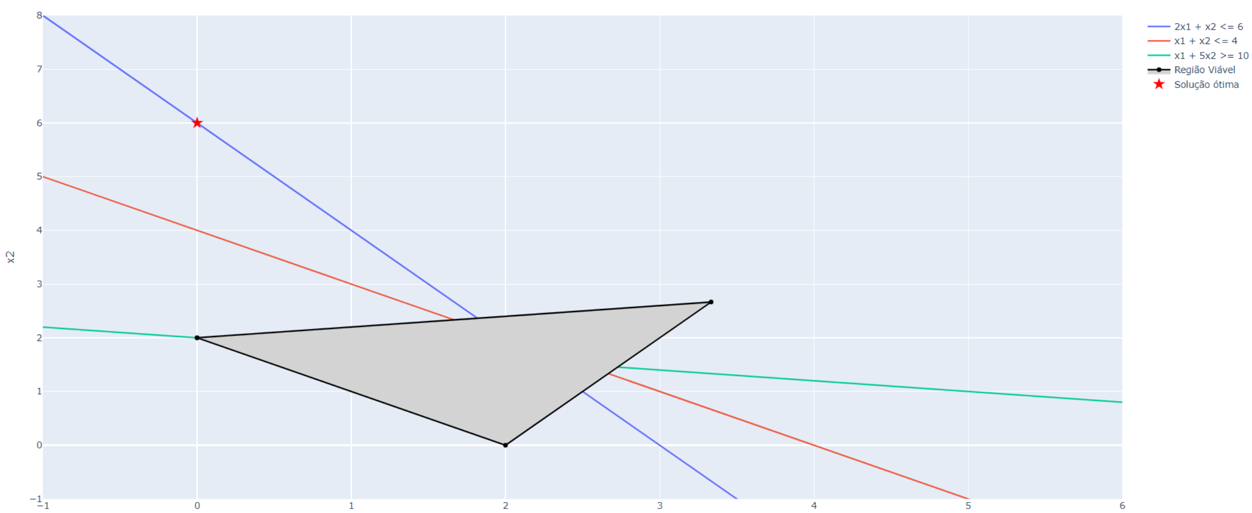

Figura 1: Plot 2D 1º Problema

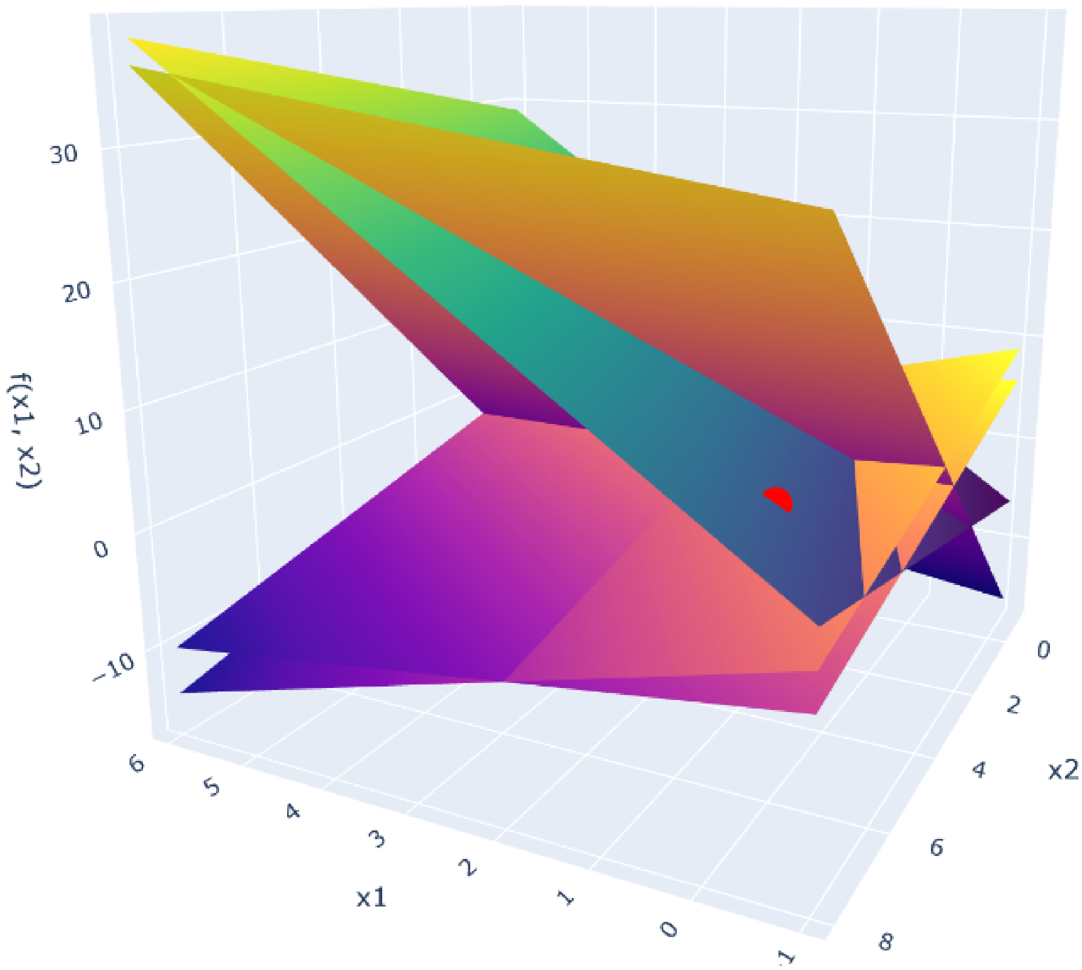

Figura 2: Plot 3D 1º Problema

4.4 Solução 2º Problema

A figura 4 representa graficamente o problema em 3D, dessa forma exibe a região factível do Problema 2, definida pelas restrições, e a superfície da função objetivo. A solução ótima é destacada como um ponto e conectada ao plano xy por uma linha tracejada.

Assim é possível verificar que a solução ótima para o problema é:

$$x^* = (4, 0) \text{ com } f(x^*) = 8$$

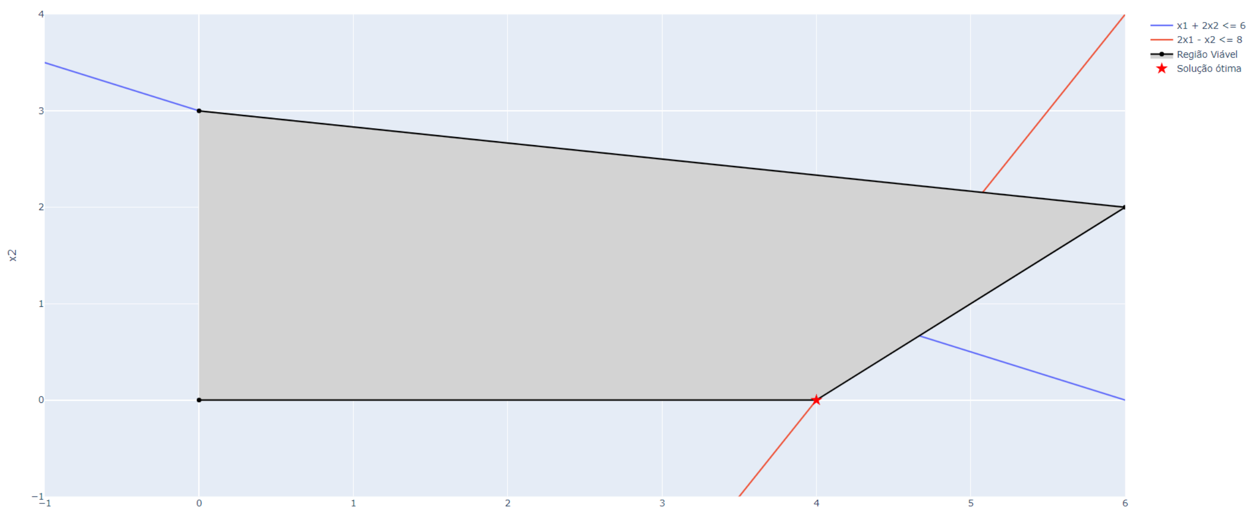

Figura 3: Plot 2D 2º Problema

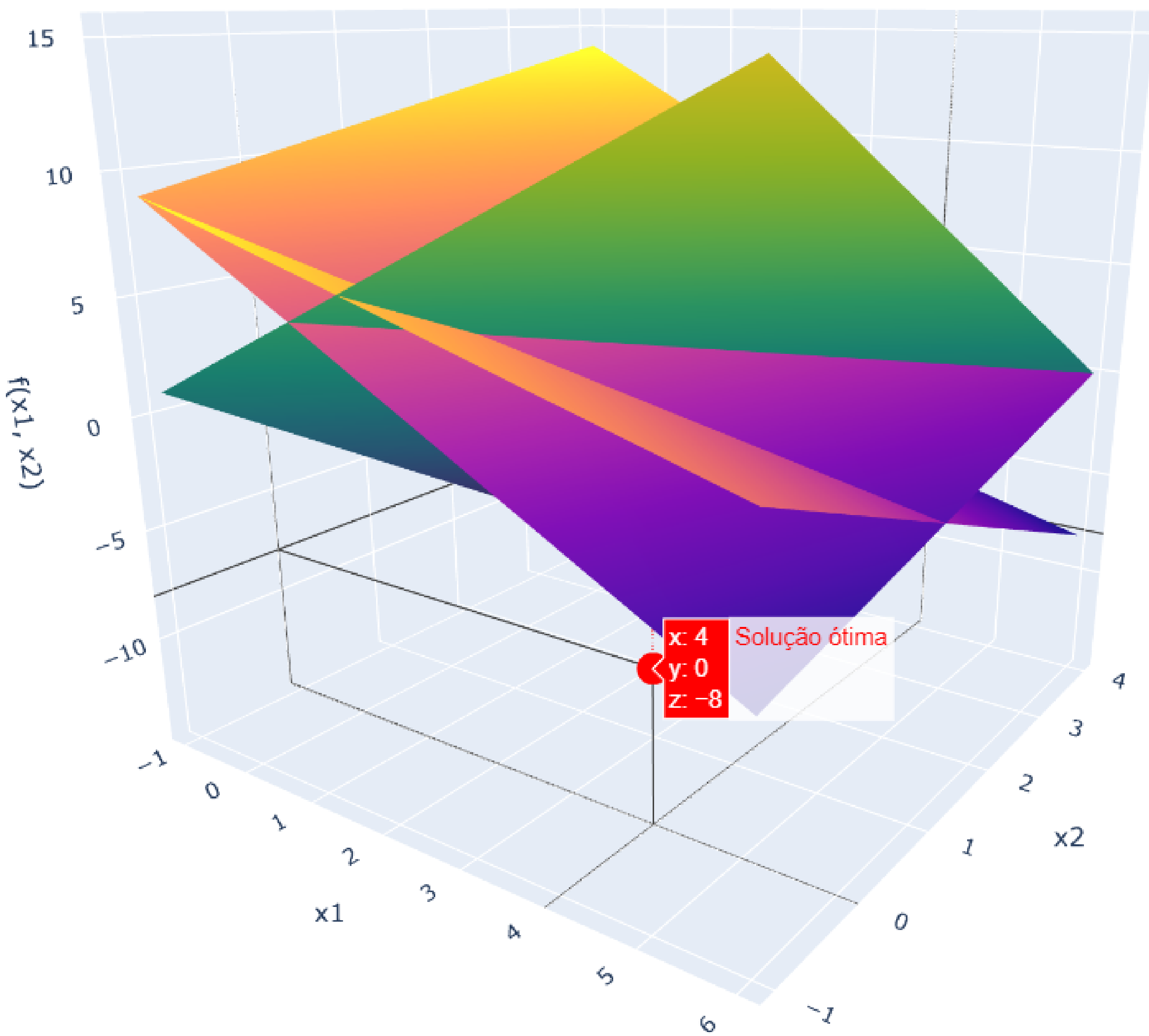

Figura 4: Plot 3D 2º Problema

4.5 Solução 3º Problema

A figura 6 representa graficamente o problema em 3D, assim mostrando a região factível, definida pelas restrições, e a superfície da função objetivo. A solução ótima é destacada como um ponto.

Dessa forma é possível verificar que a solução ótima para esse problema é:

$$x^* = (1, 1, 0) \text{ com } f(x^*) = 56$$

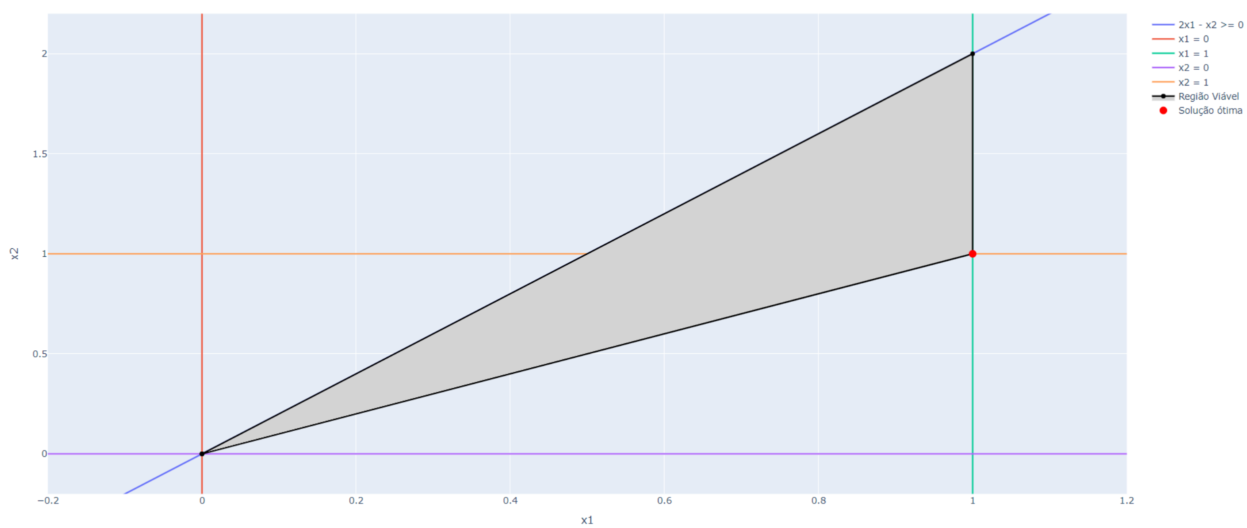

Figura 5: Plot 2D 3º Problema

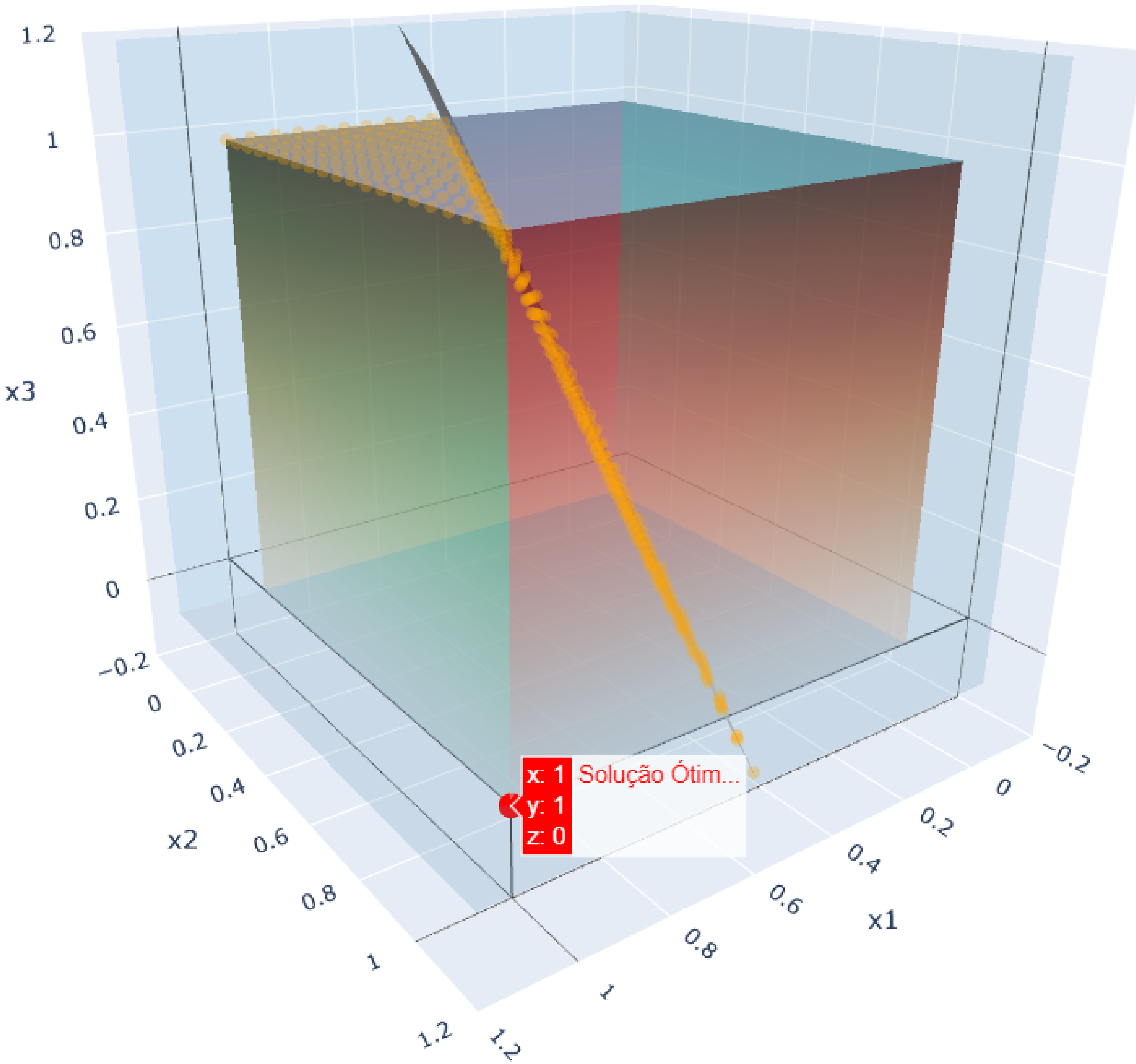

Figura 6: Plot 3D 3º Problema

4.6 Solução 4º Problema

A figura 8 representa graficamente o problema em 3D, assim mostrando a região factível, definida pelas restrições, e a superfície da função objetivo. A solução ótima é destacada como um ponto.

Dessa forma é possível verificar que a solução ótima para esse problema é:

$$x^* = (400, 0, 0, 0) \text{ com } f(x^*) = 0$$

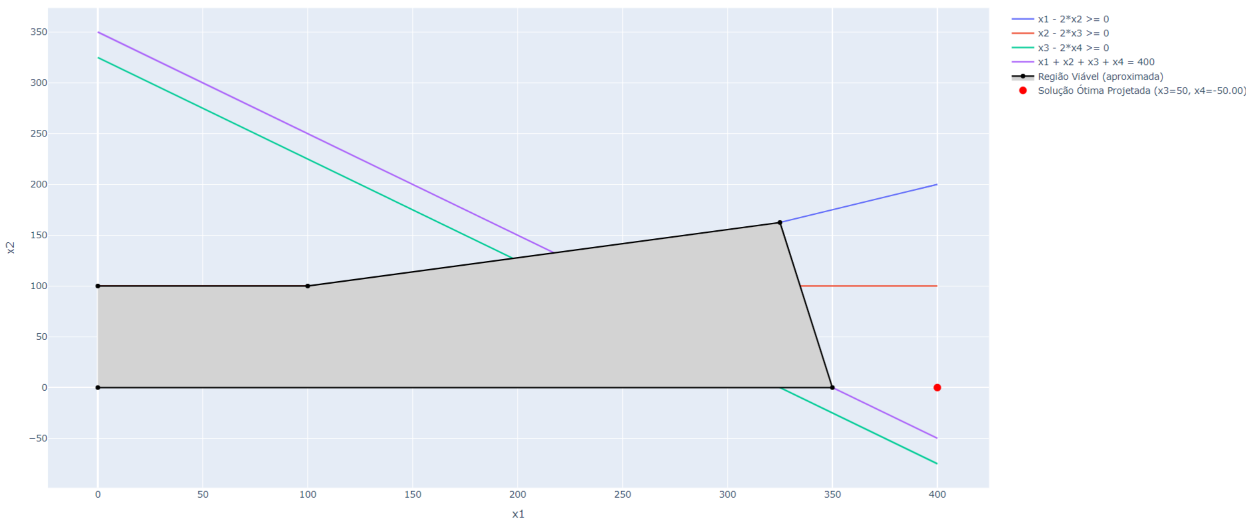

Figura 7: Plot 2D 4º Problema

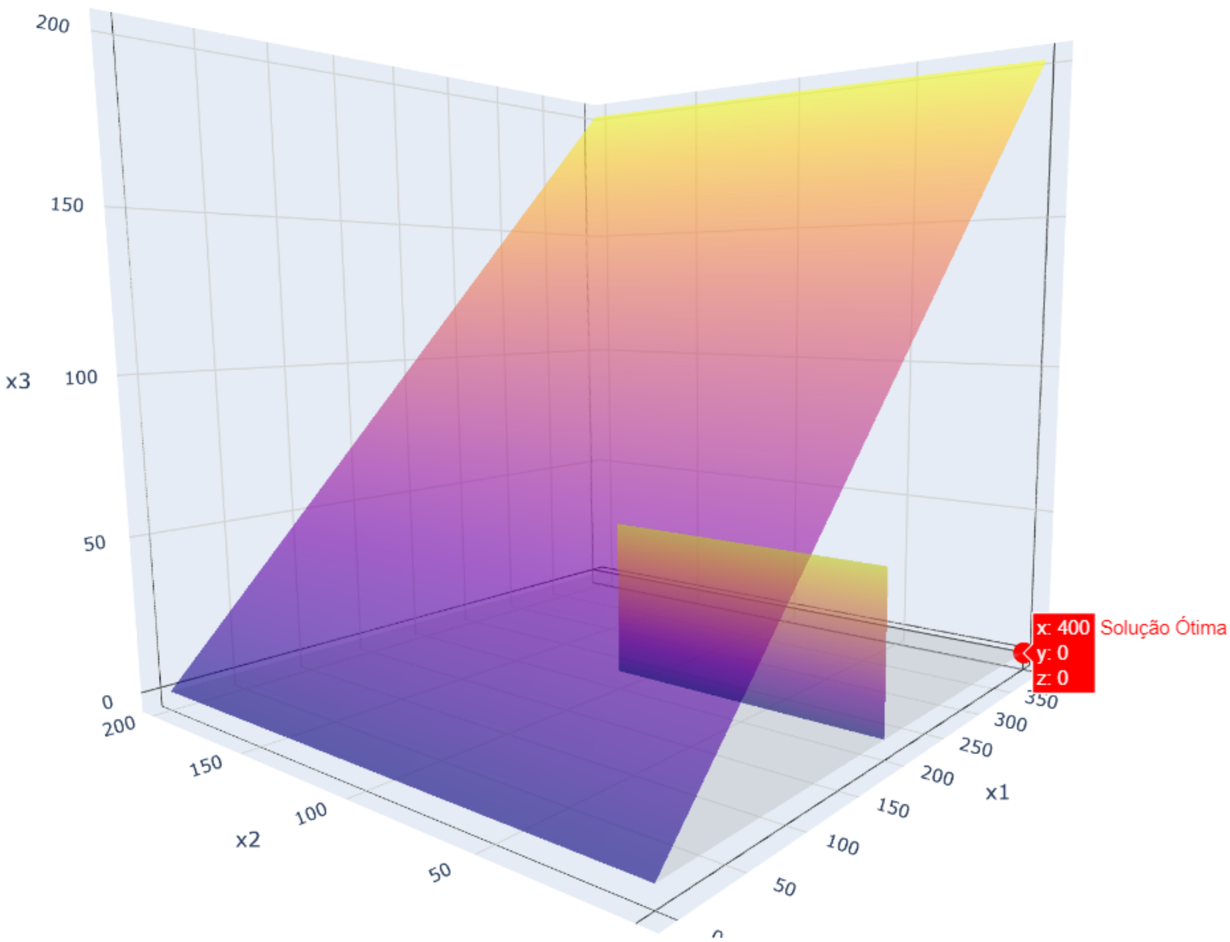

Figura 8: Plot 3D 4º Problema

4.7 Solução 5º Problema

A figura 12 representa graficamente o problema em 3D, assim mostrando a região factível, definida pelas restrições, e a superfície da função objetivo. A solução ótima é destacada como um ponto.

Dessa forma é possível verificar que a solução ótima para esse problema é:

$$x^* = (0, 1, 11) \text{ com } f(x^*) = 36$$

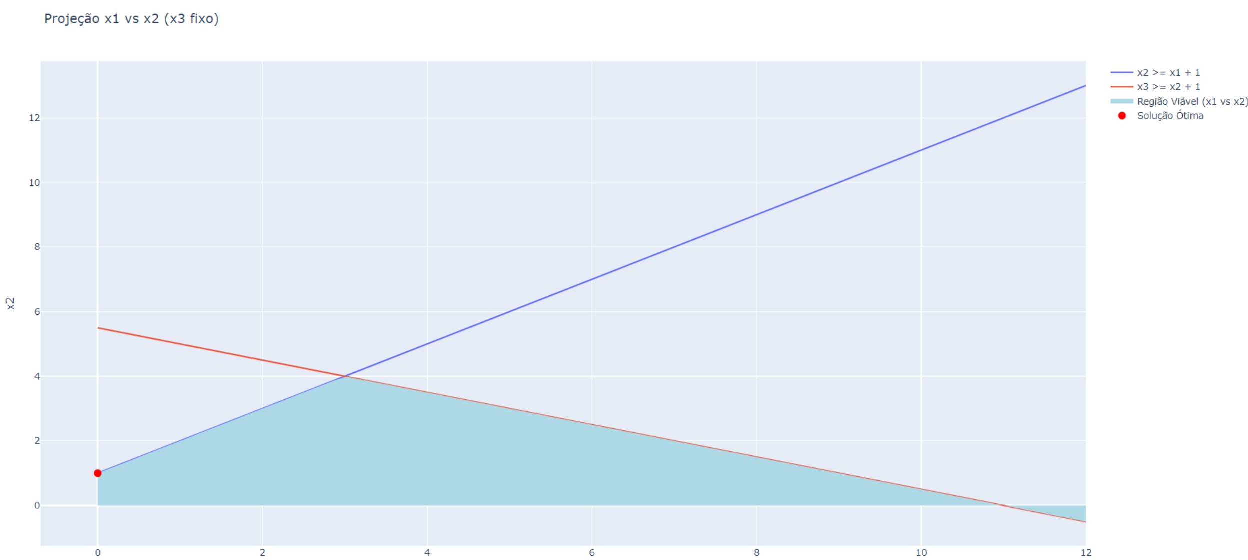

Figura 9: 5º Problema 2D: x1 vs x2 (x3 fixo)

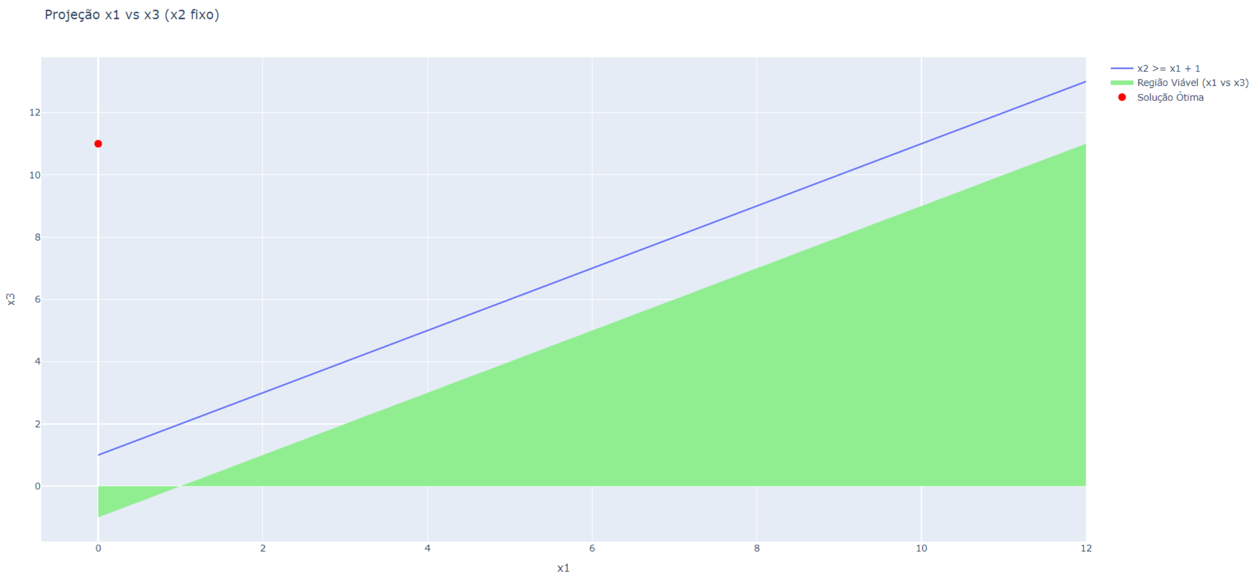

Figura 10: 5º Problema 2D: x1 vs x3 (x2 fixo)

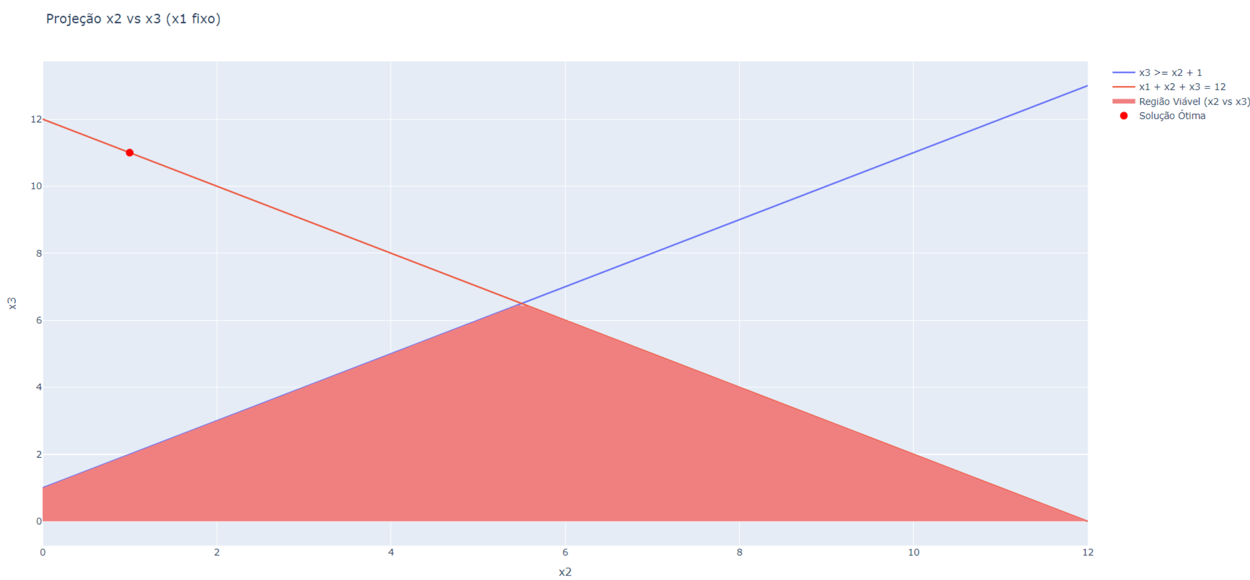

Figura 11: 5º Problema: x2 vs x3 (x1 fixo)

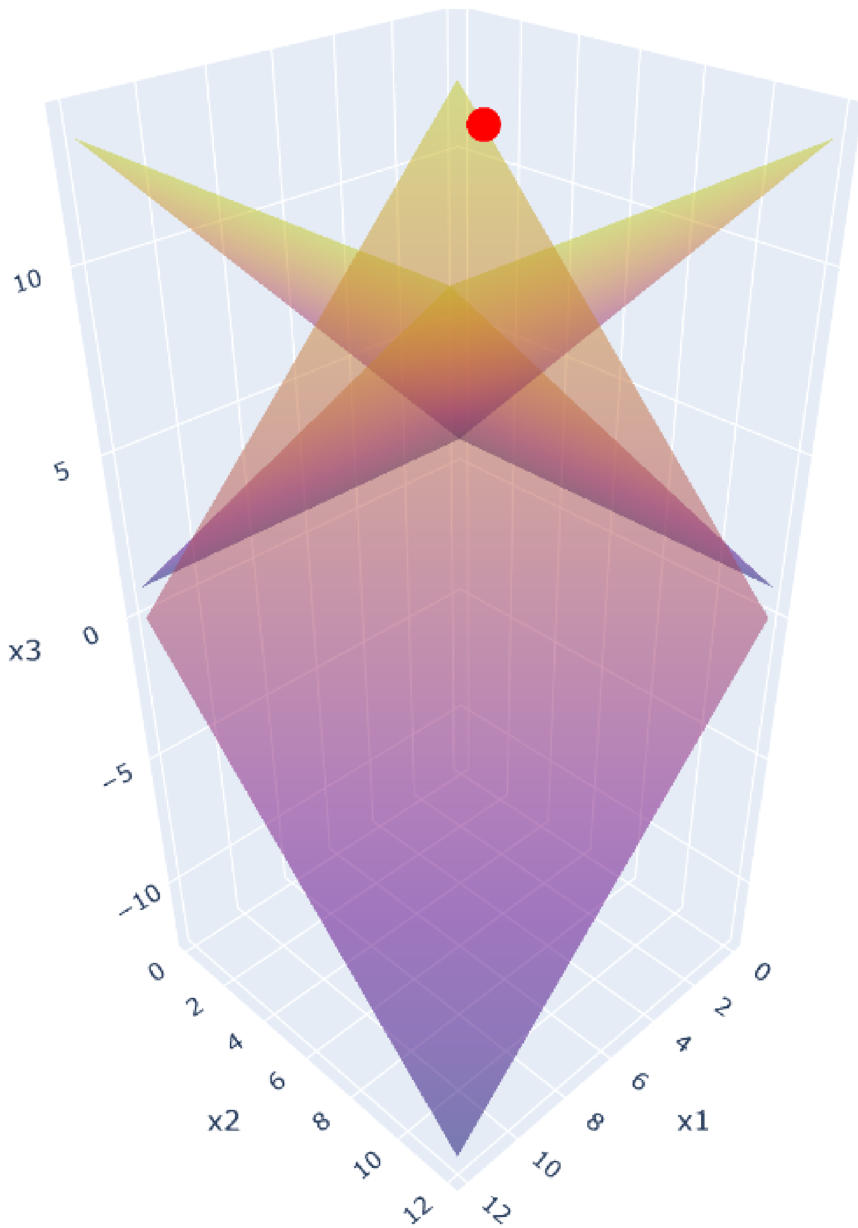

Figura 12: Plot 3D

4.8 Solução 7º Problema

imagens/imagens_corretas/2D/problema_7_2D.png

Figura 13: Plot 7

5 Referências Bibliográficas

[1] Huangfu, Q., Galabova, I., Feldmeier, M., and Hall, J. A. J. “HiGHS - high performance software for linear optimization.” https://highs.dev/

[2] Huangfu, Q. and Hall, J. A. J. “Parallelizing the dual revised simplex method.” Mathematical Programming Computation, 10 (1), 119-142, 2018. DOI: 10.1007/s12532-017-0130-5

[3] Harris, Paula MJ. “Pivot selection methods of the Devex LP code.” Mathematical programming 5.1 (1973): 1-28.

[4] Goldfarb, Donald, and John Ker Reid. “A practicable steepest-edge simplex algorithm.” Mathematical Programming 12.1 (1977): 361-371.

[5] Virtanen, P. et al., 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17(3), pp.261-272.

[6] SciPy documentation. https://docs.scipy.org/doc/scipy/reference/. Acessado em [data de acesso].

[7] Dantzig, George B. Linear programming and extensions. Rand Corporation Research Study. Princeton Univ. Press, Princeton, NJ, 1963.

[8] Hillier, S.H. and Lieberman, G.J. “Introduction to Mathematical Programming”. McGraw-Hill, 1995. Chapter 4.

[9] Bland, Robert G. "New finite pivoting rules for the simplex method."Mathematics of Operations Research 2 (1977): 103-107.

[10] Andersen, Erling D., and Knud D. Andersen. “The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm.” High performance optimization. Springer US, 2000. 197-232.

[11] Andersen, Erling D. "Finding all linearly dependent rows in large-scale linear programming." Optimization Methods and Software 6.3 (1995): 219-227.

[12] Freund, Robert M. "Primal-Dual Interior-Point Methods for Linear Programming based on Newton's Method." Unpublished Course Notes, March 2004. Accessed February 25, 2017. https://ocw.mit.edu/courses/sloan-school-of-management/15-084j-nonlinear-programming-spring-2004/lecture-notes/lec14_int_pt_mthd.pdf

[13] Fourer, Robert. "Solving Linear Programs by Interior-Point Methods." Unpublished Course Notes, August 26, 2005. Accessed February 25, 2017. http://www.4er.org/CourseNotes/Book

[14] Andersen, Erling D., and Knud D. Andersen. "Presolving in linear programming." Mathematical Programming 71.2 (1995): 221-245.

[15] Bertsimas, Dimitris, and J. Tsitsiklis. "Introduction to linear programming." Athena Scientific 1 (1997): 997.

[16] Andersen, Erling D., et al. Implementation of interior point methods for large scale linear programming. HEC/Universite de Geneve, 1996.