UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Relatório Final de Iniciação Científica

Bolsa PIBIC/CNPq – cota 2023/2024

INTRODUÇÃO À OTIMIZAÇÃO MULTIOBJETIVO

Orientando: Matheus Queiroz Mota
Orientadora: Kelly Cristina Poldi

Resumo

Este projeto aborda os fundamentos da Otimização Multiobjetivo, desenvolve e aplica métodos computacionais para resolver tais problemas, e aplica essas ferramentas em problemas práticos. Ao longo do projeto, foram implementados algoritmos de métodos exatos, como a Soma Ponderada e o Método do ε-Restrito, utilizando a linguagem Python e a biblioteca PuLP. Esses métodos foram aplicados à problemas reais, como o problema de transporte, o problema de emissão de CO2 em setores produtivos e o problema da dieta. Os resultados obtidos foram analisados com base na aproximação da Fronteira de Pareto gerada pelos algoritmos computacionais implementados, o que nos permite observar soluções eficientes e analisar o comportamento das soluções em termos dos objetivos conflitantes. Além disso, foi desenvolvida uma *interface web* para resumir toda a teoria presente neste trabalho, por meio de exemplos práticos, bem como a explicação dos códigos computacionais.

Palavras-chave: Otimização, Multiobjetivo, Programação Matemática.

1 Introdução

A Programação Multiobjetivo ou Otimização Multiobjetivo lida com problemas de otimização matemática que possuem duas ou mais funções objetivas para serem otimizadas simultaneamente, tais que esses objetivos geralmente são conflitantes, ou seja, ao melhorarmos um deles, pioramos o desempenho dos demais.

Para exemplificar, considere uma empresa de transporte que busca maximizar o lucro de toda a sua frota. Para resolver esse problema, podemos utilizar técnicas de programação linear inteira mista e, com base na solução, obter o maior lucro possível para a empresa. No entanto, suponha que a empresa também deseja minimizar a emissão de dióxido do carbono ou gás carbônico (CO2) de sua frota. Isso caracteriza um problema bi-objetivo, onde a função lucro deve ser maximizada e a função emissão de CO2 deve ser minimizada.

Neste caso, temos um conflito entre as funções objetivas. Para maximizar o lucro da frota de transporte, teríamos que transportar mais produtos, o que resultaria em mais veículos circulando e, consequentemente, um aumento nas emissões de CO2.

1.1 Modelagem de um POM

De forma generalizada, um Problema de Otimização Multiobjetivo (POM) possui um número $p > 1$ de funções objetivos que devem ser maximizadas, sujeito a um conjunto de restrições $\{\mathcal{X} \subseteq \mathbb{R}^n : \mathcal{X} \neq \varnothing\}$. Com isso, tal problema pode ser descrito como:

$$\begin{array}{ll} \max & \mathbf{z} = (f_1(\mathbf{x}), \cdots, f_p(\mathbf{x})) \\ \text{sujeito a} & x \in \mathcal{X} \end{array}$$

em que:

- $x = (x_1, x_2, \cdots, x_n)^T \in \mathbb{R}^n$ é vetor das variáveis de decisão;
- $z_i = f_i(\mathbf{x})$ é a $i-$ésima função objetivo a ser maximizada, $i = 1, \cdots, p$.

Deste modo, um POM busca otimizar (maximizar e/ou minimizar) diversas funções objetivos sujeito a restrições. Assim, não há apenas uma única solução para o problema, mas sim um conjunto ótimo de soluções denominado soluções de Pareto [2].

Vale ressaltar que os conceitos apresentados a seguir serão aplicados a um problema multiobjetivo, cujo objetivo é maximizar todas as funções envolvidas. No entanto, ao lidar com funções que precisam ser minimizadas, pode-se utilizar a seguinte relação:

$$\min f(x) = -\max f(-x) \tag{1}$$

Exemplo: antes de adentrar nos conceitos da Otimização Multiobjetivo, vamos utilizar o seguinte problema para descrever os conceitos:

$$\begin{aligned} &\max f_1 = 25x_1 + 20x_1 \\ &\max f_2 = x_1 + 8x_2 \\ &X: \begin{cases} x_1 + x_2 \leq 50 \\ 2x_1 + x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 220 \\ x_1 \geq 0, x_2 \geq 0 \end{cases} \end{aligned} \tag{2}$$

1.2 Espaço Critério

Em um problema usual de Programação Linear, cada solução factível ($x \in X$) aplicada na função objetivo retorna um valor real. No entanto, em um Problema de Otimização Multiobjetivo (POM), precisamos analisar os valores de p funções objetivo simultaneamente. Isso significa que, em vez de obtermos apenas um único valor real, obtemos um conjunto de pontos em $\mathbb{R}^p$.

Espaço Critério: O Espaço Critério $Z = \{z = f(x) \in \mathbb{R}^p : x \in X\}$ é o espaço formado pela aplicação de cada solução ($x \in X$) em um vetor $z \in \mathbb{R}^p$. Ou seja, para cada $x \in X$, temos um vetor $z = (z_1, z_2, ..., z_p) = f(x) = (f_1(x), f_2(x), ..., f_p(x))$.

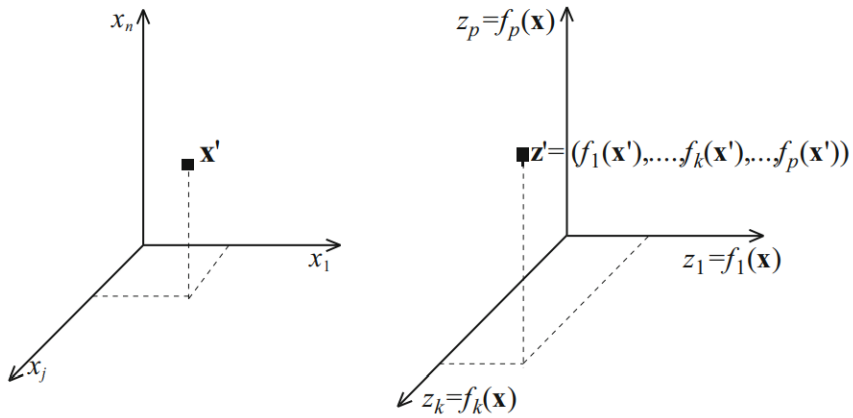

Figura 1: Imagem ilustrativa do Espaço Decisão à esquerda e do Espaço Critério à direita. Retirada de [2].

Espaço Critério do problema [2]:

Para encontrar o Espaço Critério deste problema, devemos obter x_1 e x_2 em função de f_1 e f_2 e substituir nas desigualdades de X:

$$\begin{cases} 25x_1 + 20x_2 = f_1 \\ x_1 + 8x_2 = f_2 \end{cases} \Rightarrow \begin{cases} x_1 = \dfrac{2f_1 - 5f_2}{45} \\ x_2 = \dfrac{25f_2 - z_1}{180} \end{cases}$$

Observe que, para alguns problemas, não é uma tarefa simples calcular x_1, x_2 em função de z_1, z_2. Agora devemos substituir x_1, x_2 nas restrições de X:

$$X: \begin{cases} x_1 + x_2 \leq 50 \\ 2x_1 + x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 220 \\ x_1 \geq 0, x_2 \geq 0 \end{cases} \Rightarrow Z: \begin{cases} 7f_1 + 5f_2 \leq 9000 \\ f_1 - f_2 \leq 960 \\ 11f_1 + 85f_2 \leq 39600 \\ 2f_1 - 5f_2 \geq 0 \\ 25f_2 - f_1 \geq 0 \end{cases} \tag{3}$$

Com isso, obtemos o Espaço Critério para o problema. É evidente que há muito trabalho ao substituir os valores de x_1, x_2 nas desigualdades. Todavia, um *software* que pode facilitar este processo é o Wolfram Mathematica, disponível para alunos da Unicamp.

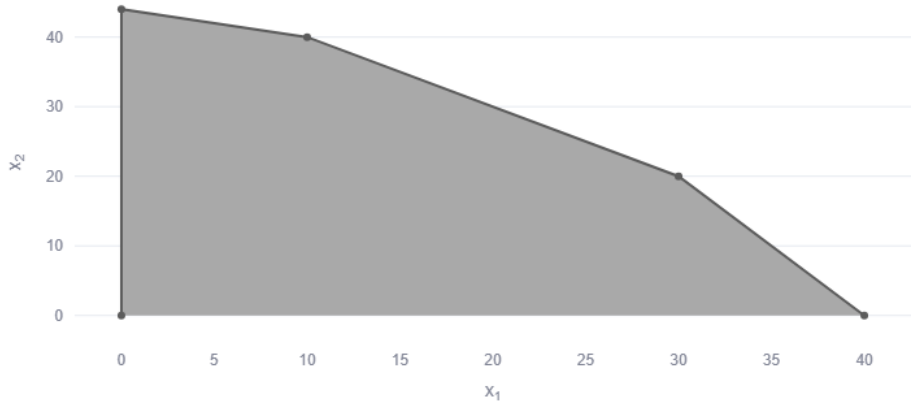

Figura 2: Espaço Decisão do problema [2].

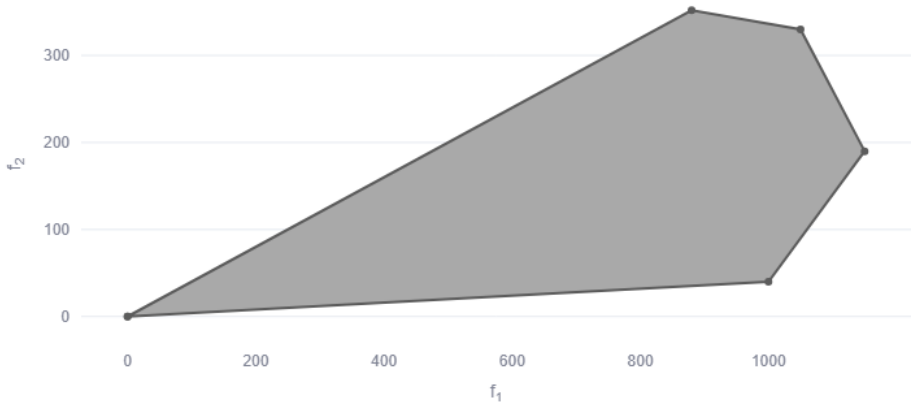

Figura 3: Espaço Critério do problema [2].

1.3 Relações de Dominância

Antes de introduzirmos os conceitos de soluções eficientes para um Problema de Otimização Multiobjetivo, vamos definir a relação de dominância entre duas soluções.

Dominância entre Soluções: considere duas soluções, x^* e $\bar{x}$, pertencentes ao espaço decisão X. A solução x^* domina $\bar{x}$ se as seguintes condições são satisfeitas:

1. $f_j(x^*) \geq f_j(\bar{x})$ para todo $j = 1, \ldots, p$;
2. $f_j(x^*) > f_j(\bar{x})$ para ao menos um j.

Notação: $x^* \preceq \bar{x}$.

Dominância Forte: A solução x^* domina fortemente a solução $\bar{x}$ se a seguinte condição é satisfeita:

$$f_j(x^*) > f_j(\bar{x}), \text{ para } j = 1, ..., p.$$

Dominância Fraca: A solução x^* domina fracamente a solução $\bar{x}$ se as seguintes condições são satisfeitas:

1. $f_j(x^*) \geq f_j(\bar{x})$ para todo $j = 1, \ldots, p$;
2. $f_j(x^*) \neq f_j(\bar{x})$ para ao menos um j;
3. x^* não domina fortemente $\bar{x}$.

1.4 Soluções Eficientes, Pontos não-dominados e Fronteira de Pareto

Um dos principais conceitos de Otimização Multiobjetivo é o de soluções eficientes, pois é por meio deste que vamos compreender quais soluções possuem mais significância para um Problema de Otimização Multiobjetivo.

Solução Eficiente: A solução x^* é dita eficiente se, e somente se, não existe nenhum x no espaço de decisão X, de modo que $x \preceq x^*$.

Ponto não-dominado: O ponto $z(x) = (f_1(x), f_2(x), \ldots, f_p(x)) \in Z$ é dito não dominado se, e somente se, x é uma solução eficiente.

Conjunto Eficiente: O conjunto eficiente X^* é formado por todos os elementos de X que não são dominados por outros elementos de X. Em outras palavras, $X^* = \{x^* \in X : x \not\preceq x^*, \forall x \in X\}$.

Conjunto Eficiente do problema [2]:

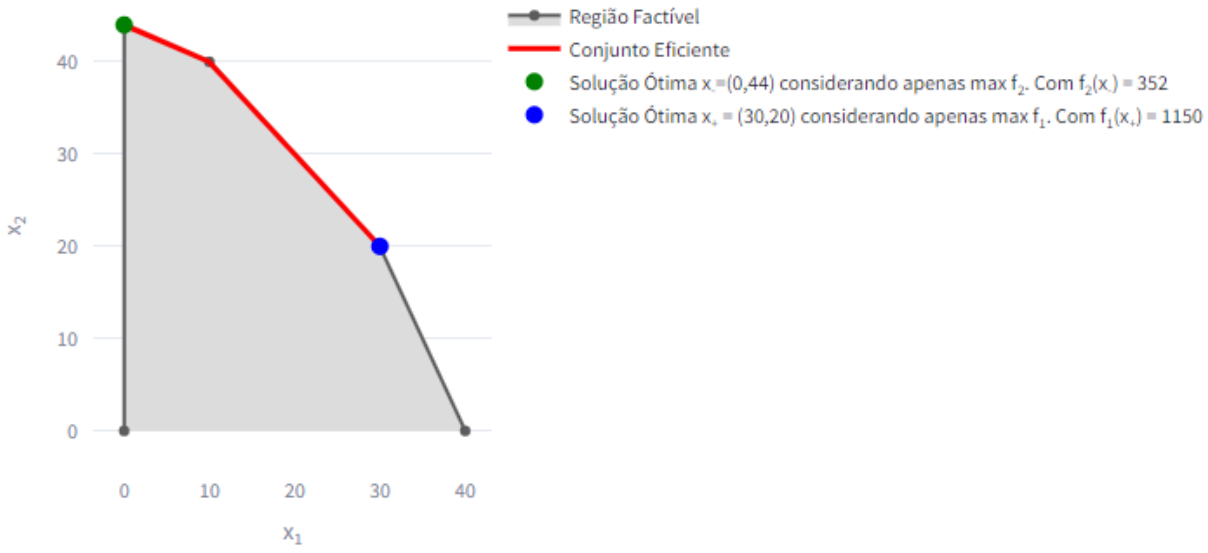

Figura 4: Conjunto Eficiente do problema [2].

Solução Fracamente Eficiente: Uma solução $\hat{x}$ é considerada fracamente eficiente se não houver nenhuma solução $x \in X$ tal que $f_k(x) > f_k(\hat{x})$ para todos os $k = 1, \ldots, p$.

Ponto Fracamente Não-dominado: Um ponto $\hat{z}$ no espaço de critério é considerado fracamente não-dominado se sua imagem inversa for uma solução fracamente eficiente, ou seja, $\hat{z} = f(\hat{x})$.

Exemplo de Relações de Dominância entre Pontos:

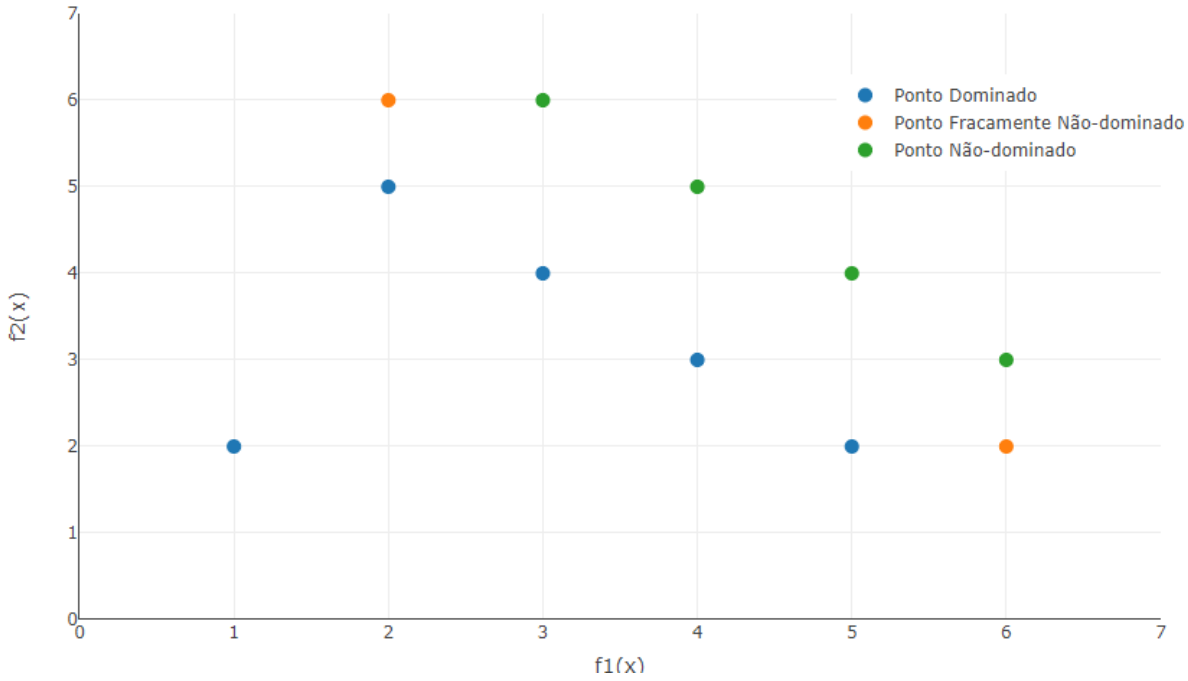

Figura 5: Gráfico com os exemplos de relação de dominância.

Fronteira de Pareto: A fronteira de Pareto, denotada por Z^* é a imagem do conjunto eficiente X^*, ou seja, $Z^* = \{z^* \in \mathbb{R}^p : z^* = f(x^*), \forall x^* \in X^*\}$

A fronteira de Pareto é de extrema importância nos Problemas de Otimização Multiobjetivo, pois fornece as soluções mais adequadas ao problema. No entanto, a decisão final sobre qual solução é a mais apropriada dentro dessa fronteira é uma responsabilidade exclusiva do tomador de decisões. Ele deve considerar cuidadosamente as necessidades e objetivos específicos do problema para fazer a escolha mais adequada.

Fronteira de Pareto do problema [2]: Por meio das definições anteriores e das inequações [3], obtemos que a fronteira de Pareto é dada por:

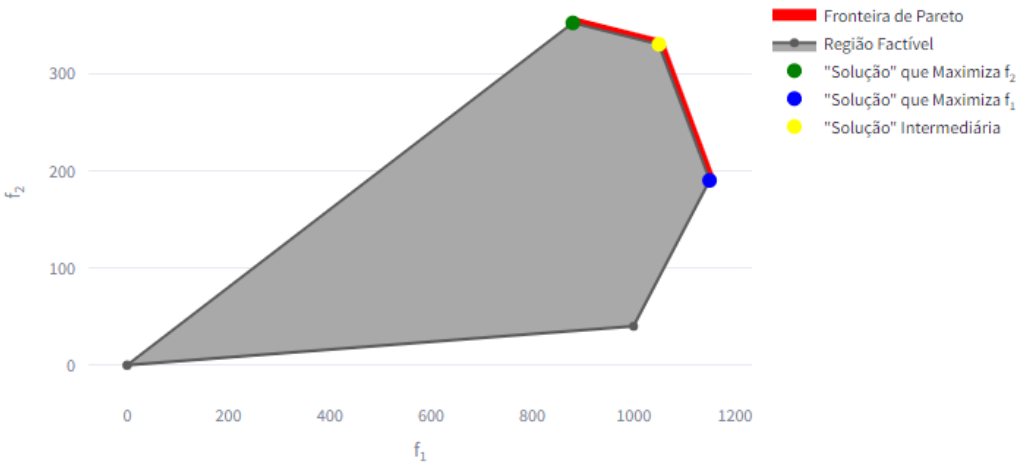

Figura 6: Fronteira de Pareto obtida para o problema [2].

1.5 Métodos Exatos

Na Otimização Multiobjetivo, uma das principais abordagens para obter soluções não dominadas e eficientes é por meio da técnica de escalarização. Essa técnica visa transformar um problema multiobjetivo em um problema de otimização mono-objetivo.

Neste trabalho, são apresentados dois métodos de escalarização:

1. **Método da Soma Ponderada:** nesse método, as funções objetivas são ponderadas por coeficientes específicos. A combinação linear dessas funções resulta em uma única função objetivo, que é então otimizada.

2. **Método do ε-Restrito (ou ε-*Constraint*):** aqui, uma das funções objetivas é otimizada, enquanto as demais são tratadas como restrições. O tomador de decisão específica o limiar que deve ser respeitado nas restrições para cada uma das funções restantes.

1.5.1 Método da Soma Ponderada

Método da Soma Ponderada: Tal método se baseia em resolver um problema mono-objetivo, cuja função objetivo é uma soma ponderada das p funções originais. Com isso, para cada função f_k, associamos um peso $\lambda_k \geq 0$.

Portanto, a solução de um POM por meio do método da Soma Ponderada pode ser obtida por meio da solução do seguinte problema:

$$\begin{aligned} \max z &= \sum_{k=1}^{p} \lambda_k f_k(x) \\ \text{sujeito a: } &\Big\{ x \in X \end{aligned}$$

Teorema 1.5.1 . *Se $\lambda_k > 0$ para todo $k = 1, \ldots, p$, então a solução do problema ponderado é uma solução eficiente para o POM original.*

Demonstração: considere uma solução ótima, $\hat{x} \in X$, para o problema ponderado. Vamos supor que $\hat{x}$ não seja eficiente (i.e, não é solução ótima do POM), ou seja, existe uma solução $\bar{x} \in X$ de modo que $f_k(\bar{x}) \geq f_k(\hat{x})$ para todo $k = 1, \ldots, p$ e $f_i(\bar{x}) > f_i(\hat{x})$ para ao menos um $i \in \{1, \ldots, p\}$. Então,

$$\sum_{k=1}^{p} \lambda_k f_k(\bar{x}) > \sum_{k=1}^{p} \lambda_k f_k(\hat{x}),$$

mas isto contradiz o fato de $\hat{x}$ ser solução ótima do problema ponderado. Portanto, $\hat{x}$ é uma solução eficiente para o POM.

Teorema 1.5.2 *Se o problema ponderado tem uma única solução x^* com $\lambda_k \geq 0$, então x^* é eficiente.*

Demonstração: seja x^* a única solução do problema ponderado. Suponha que x^* não seja eficiente, logo, existe uma outra solução viável $\bar{x} \neq x^*$ de modo que $f_k(\bar{x}) \geq f_k(x^*)$ para todo $k = 1, \ldots, p$ e $f_i(\bar{x}) > f_i(x^*)$ para ao menos um $i \in \{1, \ldots, p\}$. Como $\lambda_k \geq 0$, temos que

$$\sum_{k=1}^{p} \lambda_k f_k(\bar{x}) \geq \sum_{k=1}^{p} \lambda_k f_k(x^*).$$

Como x^* é a solução única, podemos afirmar que

$$\sum_{k=1}^{p} \lambda_k f_k(x^*) > \sum_{k=1}^{p} \lambda_k f_k(\bar{x}),$$

mas isto contradiz a desigualdade imediatamente anterior. Logo, x^* é eficiente.

1.5.2 Método do ε-restrito

Método do ε-restrito: Dado um problema com p funções objetivas $(f_1(x), \ldots, f_p(x))$, selecionamos uma destas funções (digamos $f_i(x)$) para otimizar. Ademais, temos que as outras funções tornam-se parte do conjunto de restrições, de modo que para cada f_k, temos um ε_k associado de modo que $f_k(x) \geq \varepsilon_k$ $(k \neq i)$.

Deste modo, temos que a solução de um (POM) por meio do ε-restrito é obtida por meio da solução do seguinte problema mono-objetivo:

$$\begin{aligned} &\max f_i(x) \\ &\text{sujeito a: } \begin{cases} x \in X \\ f_k(x) \geq e_k, \quad k = 1, ..., i-1, i+1, ..., p. \end{cases} \end{aligned}$$

Teorema 1.5.3 *A solução ótima do problema do método $\varepsilon-$restrito é fracamente eficiente.*

Demonstração: análoga ao do Teorema 1.5.1.

2 Metodologia

Na fase inicial deste projeto, concentramos nossos esforços na construção da base teórica para a Otimização Multiobjetivo, utilizando como principais referências ([1],[2]). Para aprofundar as demonstrações, utilizamos as referências [3] e [4]. Paralelamente, desenvolvemos uma página web em Python, cuja interface foi criada com a biblioteca *Streamlit* ([6]).

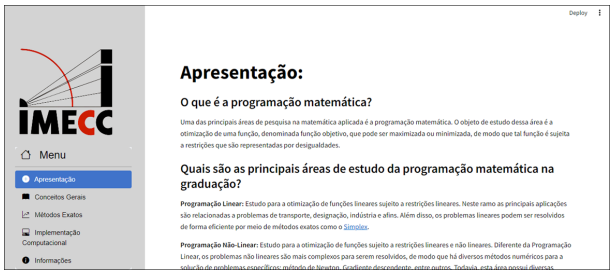

Figura 7: Interface *web* desenvolvida com o *Streamlit*.

A página mostrada na Figura 7 foi estruturada em 6 subpáginas principais, sendo que as duas últimas páginas foram desenvolvidas depois do desenvolvimento do relatório parcial:

- **Apresentação:** Uma breve descrição sobre a otimização matemática e suas diversas aplicações. Além disso, é proposto um problema prático para motivar o leitor a refletir sobre a análise de múltiplos objetivos.
- **Conceitos Gerais:** Introdução aos conceitos fundamentais da otimização multiobjetivo, acompanhada de exemplos que esclarecem esses conceitos.
- **Métodos Exatos:** Descrição detalhada dos principais métodos abordados nesta pesquisa, como o método ε-restrito e a Soma Ponderada, explicando o funcionamento, os tipos de soluções geradas e as vantagens e desvantagens de cada abordagem. Além de evidenciar os teoremas que garantem soluções para cada um dos métodos.
- **Implementação Computacional:** Esta subpágina contém os algoritmos desenvolvidos para cada um dos métodos exatos, e outros dois algoritmos que variam, além de um tutorial que orienta como utilizar tais algoritmos.
- **Aplicações:** Aplicação prática em três exemplos de problemas bi-objetivos, sendo eles o problema da dieta, do transporte e produção;
- **Informações:** Inclui detalhes sobre todas as ferramentas e recursos utilizados durante a pesquisa.

Para a implementação dos métodos Soma Ponderada e ε-restrito em Python, optamos por utilizar a biblioteca PuLP ([5]). A escolha pela PuLP se deu devido à sua facilidade de uso e código aberto. Além disso, a instalação da PuLP pode ser feita diretamente pelo terminal, simplificando a configuração do ambiente de desenvolvimento. A implementação

foi realizada utilizando as funcionalidades da PuLP para modelagem e resolução de problemas de Programação Linear, adaptando os algoritmos conforme a teoria estudada e ajustando as funcionalidades da biblioteca às necessidades específicas de cada método.

Ambos os links do **GitHub** e do **Site Web** podem ser acessados clicando no texto em negrito. É importante mencionar que o site está hospedado nos servidores da biblioteca *Streamlit*. Caso a interface não receba acessos recentes, ela pode entrar em modo de hibernação. No entanto, qualquer usuário com o link pode reativá-la clicando no botão *"Yes, get this app back up!"*.

2.1 Implementação Computacional

Os códigos a seguir foram implementados em Python, com auxílio da biblioteca PuLP ([5]). Como é possível perceber, por meio da teoria apresentada neste trabalho e pelos algoritmos a seguir, estamos trabalhando com problemas de maximização. Caso queiramos executar problemas de minimização, devemos nos recordar que

$$\min f(x) = -\max f(-x)$$

2.1.1 $\varepsilon-$restrito

No código do $\varepsilon-$restrito, temos com entradas as matrizes contendo as desigualdades do tipo "$\leq$", "$\geq$", "$=$"e seus respectivos limitantes dentro dos vetores. Além disso, deve ser informado os coeficientes das funções deverão ser imposta como restrições, seus respectivos limitantes e também os coeficientes da função que deve ser maximizada. Deste modo, o modelo considerado é da forma:

$$\max z_i = f_i(x)$$
$$\text{sujeito a: } \begin{cases} Ax \leq b \\ Gx \geq g \\ Ex = e \\ f_k(x) \geq \varepsilon_k, \quad k = 1, ..., i-1, i+1, ..., p \\ x \geq 0 \end{cases}$$

```
from pulp import LpProblem, LpMaximize, LpVariable, lpDot, LpStatus
import numpy as np
import matplotlib.pyplot as plt


def e_restrito(n=0, A=0, b=0, G=0, g=0, E=0, e=0, coeficientes_fun_pri
    =0, num_funcao_restricao=0, matriz_coeficientes_f_aux=0, epsilons=0):
    model = LpProblem(name="large-problem", sense=LpMaximize)

    # Variaveis de decisao
    x = [LpVariable(name=f"x{i}", lowBound=0) for i in range(n)]

    # Restricoes <=
    if isinstance(A, (list, np.ndarray)) and isinstance(b, (list, np.
    ndarray)):
        for i in range(len(A)):
```

```
            model += (lpDot(A[i], x) <= b[i], f"restricao_<=_{i}")

    # Restricoes >=
    if isinstance(G, (list, np.ndarray)) and isinstance(g, (list, np.
    ndarray)):
        for i in range(len(G)):
            model += (lpDot(G[i], x) >= g[i], f"restricao_>=_{i}")

    # Restricoes ==
    if isinstance(E, (list, np.ndarray)) and isinstance(e, (list, np.
    ndarray)):
        for i in range(len(E)):
            model += (lpDot(E[i], x) == e[i], f"restricao_==_{i}")

    # Funcao Objetivo Principal
    funcao_principal = lpDot(coeficientes_fun_pri, x)
    model += funcao_principal

    # Restricoes epsilon
    if isinstance(matriz_coeficientes_f_aux, (list, np.ndarray)) and
    isinstance(epsilons, (list, np.ndarray)):
        for i in range(num_funcao_restricao):
            model += (lpDot(matriz_coeficientes_f_aux[i], x) >= epsilons
    [i], f"restricao_eps_{i}")

    # Resolvendo o problema
    status = model.solve()

    # Obtencao da solucao
    sol = np.array([x[i].varValue for i in range(n)])
    coef_f_values = np.vstack((coeficientes_fun_pri,
    matriz_coeficientes_f_aux))

    # Calculando os valores das funcoes objetivas
    value_func_matriz = np.zeros(np.shape(coef_f_values)[0])
    for i in range(np.shape(coef_f_values)[0]):
        value_func_matriz[i] = np.dot(coef_f_values[i], sol)

    return sol, status, value_func_matriz
```

Código 1: código do método do $\varepsilon-$restrito.

2.1.2 Soma Ponderada

No código do método da Soma Ponderada, temos como entradas as mesmas matrizes de restrições do método do $\varepsilon-$restrito. A única diferença é que devemos informar os coeficientes de todas as funções objetivos em uma única matriz, e os valores de λ que acompanha cada função (como foi explicado anteriormente, podemos otimizar a escolha dos parâmetros λ para problemas bi-objetivos por meio do Algoritmo [2.1.3]). Deste modo, o modelo considerado é da seguinte forma:

$$\max z = \sum_{k=1}^{p} \lambda_k f_k(x)$$

$$\text{sujeito a: } \begin{cases} Ax \leq b \\ Cx \geq d \\ Ex = e \\ x \geq 0 \end{cases}$$

```python
from pulp import LpProblem, LpMaximize, LpVariable, lpDot, LpStatus
import numpy as np
import matplotlib.pyplot as plt

def soma_ponderada(n, A=0, b=0, C=0, d=0, E=0, e=0, coef_func=0, vetor_lambda=0):
    # Definindo o problema de maximizacao
    model = LpProblem(name="multiobjective-weighted-sum", sense=LpMaximize)

    # Variaveis de decisao
    x = [LpVariable(name=f"x{i}", lowBound=0) for i in range(n)]

    # Restricoes <=
    if isinstance(A, (list, np.ndarray)) and isinstance(b, (list, np.ndarray)):
        for i in range(len(A)):
            model += (lpDot(A[i], x) <= b[i], f"restricao_<=_{i}")

    # Restricoes >=
    if isinstance(C, (list, np.ndarray)) and isinstance(d, (list, np.ndarray)):
        for i in range(len(C)):
            model += (lpDot(C[i], x) >= d[i], f"restricao_>=_{i}")

    # Restricoes ==
    if isinstance(E, (list, np.ndarray)) and isinstance(e, (list, np.ndarray)):
        for i in range(len(E)):
            model += (lpDot(E[i], x) == e[i], f"restricao_==_{i}")

    # Funcao objetivo ponderada
    funcao_principal = lpDot(np.dot(np.transpose(vetor_lambda), coef_func), x)
    model += funcao_principal

    # Resolvendo o problema
    status = model.solve()

    # Obtencao da solucao
    sol = np.array([x[i].varValue for i in range(n)])

    # Calculando os valores das funcoes objetivas
    value_func_aux = np.dot(coef_func, sol)
```

```
39
40      return sol, status, value_func_aux
```

Código 2: código do método da Soma Ponderada.

2.1.3 Variação dos Parâmetros - Soma Ponderada

Para analisar as diferentes soluções do método da soma ponderada, desenvolvemos o código *varrer_lambda*(). Esse código foi criado especificamente para problemas bi-objetivo. Inicialmente, definimos o número de passos percorridos em uma malha uniforme de zero a um. Na primeira iteração, temos $\lambda_1 = 0$, o que implica $\lambda_2 = 1$. Nas iterações subsequentes, o valor de λ_1 é determinado pelo número de divisões da malha uniforme, com $\lambda_2 = 1 - \lambda_1$. Tal abordagem garante pesos distintos e combinações equilibradas para ambos os λ's.

```
1  def varrer_lambda(n_passos):
2      solutions = []
3      valor_f = []
4      lambdas = np.linspace(0,1,n_passos)
5
6      for lambda1 in lambdas:
7          #lambda2 = np.random.uniform(0, lambda1)
8          lambda2 = 1 - lambda1
9          vetor_lambda = np.array([lambda1, lambda2])
10         sol, status,f_values = soma_ponderada(n, A, b, C, d, E, e,
       coef_func, vetor_lambda)
11
12         if status == 1:
13             solutions.append(tuple(np.round(sol, decimals=10)))
14             valor_f.append(tuple(np.round(f_values, decimals=10)))
15
16     # Remover duplicatas
17     solutions_distinct = list(set(solutions))
18     f_values_distintos = list(set(valor_f))
19     return solutions_distinct,f_values_distintos
```

Código 3: código que varia os λ's de problemas bi-objetivo.

2.1.4 Variação dos Parâmetros - $\varepsilon-$restrito

Ademais, o código *percorrer_epsilon*() também é específico para problemas bi-objetivo, porém possui mais parâmetros de entrada. Primeiramente, devemos informar o ε inicial do problema, o ε final e a quantidade de passo que devemos percorrer na malha uniforme entre o valor inicial e final. Diferentemente do algoritmo anterior, não temos um ε "ideal"para todos os tipos de problema. Deste modo, cabe ao tomador de decisões escolher qual seria a tolerância mínima que deve ser respeitada pelo método.

```
1  def percorrer_epsilon(epsilon_range=None):
2      solutions = []
3      valor_f = []
4
5      for epsilon in epsilon_range:
```

```
        epsilons = np.full(num_funcao_restricao, epsilon)
        sol, status,f_val = e_restrito(n, A, b, G, g, E, e,
    coeficientes_fun_pri, num_funcao_restricao, matriz_coeficientes_f_aux
    , epsilons)

        if status == 1:  # 1 indica 'Optimal'
            solutions.append(tuple(np.round(sol, decimals=10)))
            valor_f.append(tuple(np.round(f_val, decimals=10)))
        else:
            pass

    # Remover duplicatas
    solutions_distinct = list(set(solutions))
    valor_f_distinct = list(set(valor_f))
    return solutions_distinct,valor_f_distinct
```

Código 4: código que varia o ε de problemas bi-objetivo.

3 Aplicações e Análise de Resultados

Nesta seção, serão modelados quatro problemas bi-objetivos para explorar o comportamento da curva característica, que representa o conflito entre as soluções [4], considerando diferentes combinações de funções objetivas (min-min, max-min, min-max, max-max). Além disso, será avaliado o desempenho de cada um dos métodos exatos implementados.

3.1 Problema do Transporte

Na cidade de Campinas, uma empresa possui dois armazéns (Armazéns 1 e 2) responsáveis pelo transporte de produtos alimentícios para três mercados locais (Mercados A, B e C). Um dos objetivos da empresa é minimizar os custos de transporte até os mercados. No entanto, visando emitir a quantidade miníma de CO2 por toda sua frota, a empresa estimou a produção de CO2, em média por valor unitário de produto transportado, de cada armazém para cada cidade. Note que neste exemplo, o decisor busca um equilíbrio entre a eficiência logística e questão ambiental.

Variáveis de decisão:

x_{1A} : quantidade de mercadorias transportadas do Armazém 1 para o Mercado A
x_{1B} : quantidade de mercadorias transportadas do Armazém 1 para o Mercado B
x_{1C} : quantidade de mercadorias transportadas do Armazém 1 para o Mercado C
x_{2A} : quantidade de mercadorias transportadas do Armazém 2 para o Mercado A
x_{2B} : quantidade de mercadorias transportadas do Armazém 2 para o Mercado B
x_{2C} : quantidade de mercadorias transportadas do Armazém 2 para o Mercado C

Funções objetivo: neste problema, a função custo é representada por $z_1(x)$ e a função que representa a quantidade de CO2 é $z_2(x)$. Os coeficientes correspondentes são dados por:

$$\min\ z_1(x) = 2x_{1A} + 4x_{1B} + 5x_{1C} + 3x_{2A} + 1x_{2B} + 2x_{2C} \quad \text{(Custo de transporte em R\$}$$
$$\min\ z_2(x) = 9x_{1A} + 4x_{1B} + x_{1C} + 2x_{2A} + 5x_{2B} + 8x_{2C} \quad \text{(Emissão de CO2)}$$

Restrições:

- Capacidade de armazenamento: cada armazém possui uma capacidade máxima de produtos que podem ser estocados, ou seja, a quantidade de mercadorias que vai do armazém para todas as cidades não deve exceder a capacidade do armazém. Os respectivos valores de capacidade são dados a seguir:

$$x_{1A} + x_{1B} + x_{1C} \leq 100 \quad \text{(Capacidade do Armazém 1)}$$
$$x_{2A} + x_{2B} + x_{2C} \leq 150 \quad \text{(Capacidade do Armazém 2)}$$

- Demanda: cada mercado possui uma demanda mínima de produtos que deve ser respeitada, ou seja, a produção de cada armazém para a cidade deve ser maior ou igual a essa demanda mínima. Vale ressaltar que, nas restrições de demanda, estamos trabalhando com desigualdades do tipo "$\geq$", pois restrições do tipo "$=$"podem ocasionar infactibilidade. As restrições são dadas a seguir:

$$x_{1A} + x_{2A} \geq 80 \quad \text{(Demanda do Mercado A)}$$
$$x_{1B} + x_{2B} \geq 70 \quad \text{(Demanda do Mercado B)}$$
$$x_{1C} + x_{2C} \geq 100 \quad \text{(Demanda do Mercado C)}$$

Solução: no método do ε-restrito, variamos o valor de ε no intervalo $[-500, 500]$ em mil passos, por meio do algoritmo *percorrer_epsilon*(). Considerando a função custo como restrição, ou seja, a função $z_1(x) \geq \varepsilon$, e aplicando o método do ε-restrito, obtivemos a mesma solução do problema ponderado, testado no algoritmo *varrer_lambda*() com mil passos. Os resultados obtidos são apresentados a seguir:

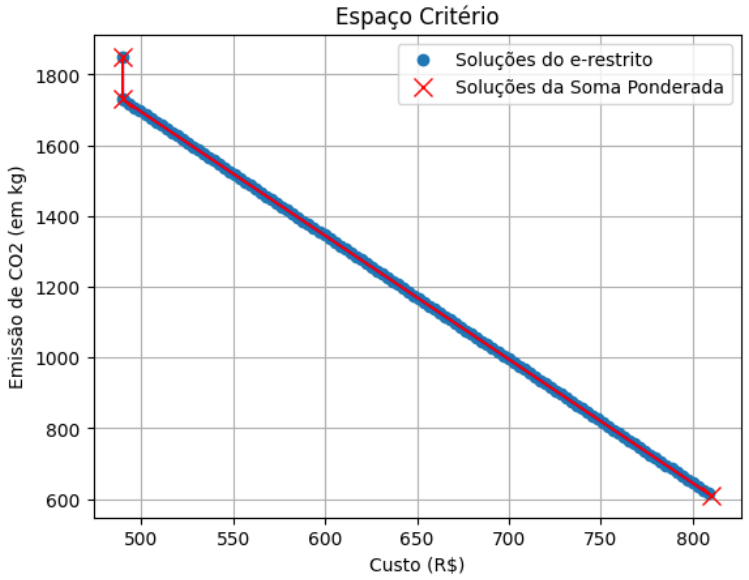

Figura 8: Gráfico da fronteira de Pareto do exemplo do transporte, para os métodos exatos.

Neste exemplo, como é possível observar na Figura [8], ao diminuir o custo de transporte, aumentamos a quantidade de CO2 emitida. No entanto, note que, no canto superior esquerdo da Figura [8], uma das soluções obtidas pelo método da soma ponderada, $s_1(x) = (490; 1850)$, possui o mesmo custo que outra solução obtida pelo mesmo método, $s_2(x) = (490; 1730)$, mas com uma emissão de CO2 superior. Conforme as Definições [1.4] e [1.4], concluímos que a solução $s_1(x)$ é fracamente eficiente (dado que este é um problema com duas minimizações).

3.2 Emissão de CO2 e Lucro em Setores de Produção

Uma empresa produz seis produtos P_1, P_2, P_3, P_4, P_5 e P_6. Cada produto gera um certo lucro e emite uma quantidade específica de CO2. O objetivo é encontrar a quantidade de cada produto a ser fabricado para maximizar o lucro e minimizar as emissões de CO2.

Variáveis de decisão:

x_1 : quantidade de P_1 a ser produzida
x_2 : quantidade de P_2 a ser produzida
x_3 : quantidade de P_3 a ser produzida
x_4 : quantidade de P_4 a ser produzida
x_5 : quantidade de P_5 a ser produzida
x_6 : quantidade de P_6 a ser produzida

Funções objetivo: neste problema, a função lucro é representada por $z_1(x)$ e a função de emissões de CO2 é representada por $z_2(x)$. As funções objetivo são dadas por:

$$\max z_1(x) = 15x_1 + 20x_2 + 25x_3 + 30x_4 + 22x_5 + 18x_6 \quad \text{(Lucro em R\$)}$$
$$\min z_2(x) = 12x_1 + 10x_2 + 18x_3 + 22x_4 + 14x_5 + 16x_6 \quad \text{(Emissão de CO2 em kg)}$$

Restrições:

- Orçamento: O custo total de produção não deve exceder o orçamento disponível:

$$50x_1 + 70x_2 + 60x_3 + 80x_4 + 65x_5 + 55x_6 \leq 15,000$$

- Horas de trabalho: o total de horas de trabalho necessárias para a produção não deve exceder as horas disponíveis:

$$2x_1 + 3x_2 + 1x_3 + 4x_4 + 2.5x_5 + 1.5x_6 \leq 2,000$$

- Demanda mínima: cada produto deve atender a uma demanda mínima de mercado:

$$x_i \geq 10 \quad \text{para } i = 1, \ldots, 6$$

Modelo Matemático Multiobjetivo:

$$\begin{aligned}
\max\ & z_1(x) = 15x_1 + 20x_2 + 25x_3 + 30x_4 + 22x_5 + 18x_6 \\
\min\ & z_2(x) = 12x_1 + 10x_2 + 18x_3 + 22x_4 + 14x_5 + 16x_6 \\
\text{s.a:}\ & \begin{cases}
50x_1 + 70x_2 + 60x_3 + 80x_4 + 65x_5 + 55x_6 \leq 15,000 \\
2x_1 + 3x_2 + 1x_3 + 4x_4 + 2.5x_5 + 1.5x_6 \leq 2,000 \\
x_i \geq 10 \quad \text{para } i = 1, \ldots, 6 \\
x_i \geq 0 \quad \text{para } i = 1, \ldots, 6
\end{cases}
\end{aligned}$$

Solução: por meio dos métodos do $\varepsilon-$restrito e soma ponderada, obtemos a seguinte fronteira de Pareto:

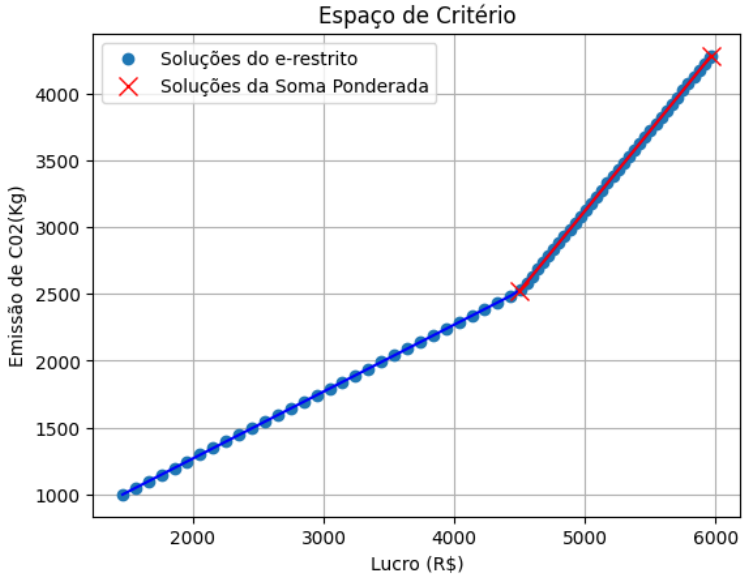

Figura 9: Gráfico da fronteira de Pareto do exemplo de lucro × CO2.

Pela Figura [9], temos que ao elevar o seu lucro, a empresa acaba por elevar a emissão de CO2 advinda de seus produtos. Tal resultado é evidente, tendo em vista que para aumentar seu lucro, tal empresa necessita produzir mais de seus produtos, que por sua vez aumentam a emissão de CO2.

3.3 Problema da Dieta

O objetivo deste modelo é obter uma dieta que minimize o custo diário de alimentos e maximize a quantidade de proteínas ingeridas. Para resolver esse problema, selecionamos uma lista de alimentos (Tabela 1), suas informações nutricionais e a dose diária recomendada de nutrientes. Os valores apresentados na tabela correspondem a 100g de cada alimento, enquanto o custo médio por 100g foi obtido a partir de preços da internet. A quantidade de calorias, sódio e gordura foi ajustada para considerar outros ingredientes envolvidos no preparo desses alimentos.

Com base nas informações, elaboramos o seguinte modelo:

Alimento / Nutrientes	Arroz (x_1)	Feijão (x_2)	Frango (x_3)	Coxão Mole (x_4)	Queijo (x_5)	Costela (x_6)
Carboidrato (g)	28	14	0	0	5	0
Proteína (g)	3	5	32	32	15	14
Gordura (g)	2	2	8	12	8	13
Calorias (Kcal)	216	164	456	516	176	670
Fibra (g)	2	8	0	0	0	0
Sódio (mg)	166	167	205	209	236	228
Custo (R$)	0,8	0,9	2,7	4,0	6,5	2,8

Tabela 1: Valores nutricionais dos alimentos.

$$\min\ f_1(x) = 0.8x_1 + 0.9x_2 + 2.7x_3 + 4x_4 + 6.5x_5 + 2.8x_6 \quad \text{(Custo)}$$
$$\max\ f_2(x) = 3x_1 + 5x_2 + 32x_3 + 32x_4 + 15x_5 + 14x_6 \quad \text{(Proteína)}$$
$$\text{s.a.} \quad \begin{cases} 1800 \leq 216x_1 + 164x_2 + 456x_3 + 516x_4 + 176x_5 + 670x_6 \leq 2500 \quad \text{(Kcal)} \\ 100 \leq 28x_1 + 14x_2 + 0x_3 + 0x_4 + 5x_5 + 0x_6 \leq 150 \quad \text{(Carboidrato)} \\ 25 \leq 2x_1 + 8x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 \leq 36 \quad \text{(Fibra)} \\ 1300 \leq 166x_1 + 167x_2 + 205x_3 + 209x_4 + 236x_5 + 228x_6 \leq 2300 \quad \text{(Sódio)} \\ 22 \leq 2x_1 + 2x_2 + 8x_3 + 12x_4 + 8x_5 + 13x_6 \leq 29 \quad \text{(Gordura Saturada)} \\ x_i \geq 10 \quad \text{para } i = 1, \ldots, 6 \end{cases}$$

Ao adaptarmos tal modelo ao método da soma ponderada e ao ε-restrito, considerando a função de proteínas como uma restrição, obtivemos os seguintes resultados:

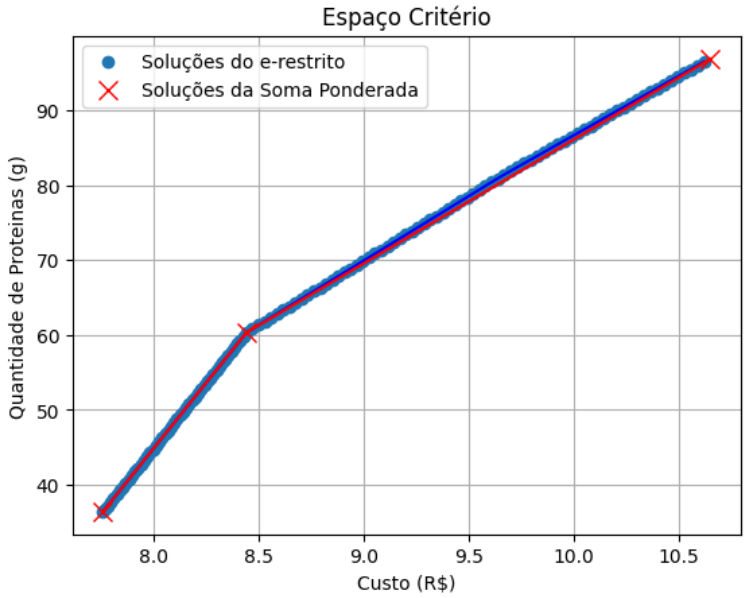

Figura 10: Gráfico da fronteira de Pareto do exemplo da dieta.

O resultado anterior nos mostra que, ao aumentar a quantidade de proteínas diárias,

inevitavelmente elevamos o custo da refeição. Este resultado é corroborado pelas quantidades de alimentos selecionados pelo modelo em cada par de soluções, presentes na Tabela [2].

Na primeira solução, onde o custo é mínimo, há uma penalização devido à redução do custo, resultando na não seleção de alimentos ricos em proteínas. Foram selecionadas apenas 71g de costela, a carne com o mais baixo teor proteico, além de grandes quantidades de arroz e feijão, que possuem um custo menor. Esses alimentos, embora menos proteicos, contêm nutrientes suficientes para satisfazer as outras restrições nutricionais.

Por outro lado, na solução em que obtivemos o maior valor de proteínas diárias, o modelo optou por reduzir a quantidade de arroz e feijão consumidos e aumentar a quantidade de frango (aproximadamente 240g), alimento com maior valor proteico entre todos os outros alimentos. Essa escolha reflete a necessidade de incluir alimentos mais caros, mas com maior conteúdo proteico, para atingir o objetivo de maximizar a ingestão de proteínas.

Ademais, a última solução da tabela é uma solução intermediária que busca equilibrar o custo e a quantidade proteica. Como podemos ver, a quantidade de frango selecionada foi de 116g, enquanto a de arroz foi de 432g.

Alimento / **(Custo, Proteínas)**	**Arroz** (x_1)	**Feijão** (x_2)	**Frango** (x_3)	**Coxão Mole** (x_4)	**Queijo** (x_5)	**Costela** (x_6)
(7,76; 36,40)	3,88	2,96	0,00	0,00	0,00	0,71
(10,62; 96,60)	2,30	2,55	2,40	0,00	0,00	0,00
(8,44; 60,30)	4,32	2,07	1,16	0,00	0,00	0,00

Tabela 2: Tabela com os valores das funções objetivo e soluções factíveis.

3.4 Problema da Dieta - Maximização de Proteínas e Carboidratos

Considere o mesmo problema da dieta do exemplo anterior. porém, iremos desconsiderar a função custo e a substituir pela restrição de carboidratos, isto é, desejamos maximizar a quantidade de carboidratos ingerida.

Deste modo, temos o seguinte modelo:

$$\max f_1(x) = 28x_1 + 14x_2 + 0x_3 + 0x_4 + 5x_5 + 0x_6 \quad \text{(Carboidrato)}$$
$$\max f_2(x) = 3x_1 + 5x_2 + 32x_3 + 32x_4 + 15x_5 + 14x_6 \quad \text{(Proteína)}$$
$$\text{s.a.} \begin{cases} 1800 \leq 216x_1 + 164x_2 + 456x_3 + 516x_4 + 176x_5 + 670x_6 \leq 2500 \quad \text{(Kcal)} \\ 25 \leq 2x_1 + 8x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 \leq 36 \quad \text{(Fibra)} \\ 1300 \leq 166x_1 + 167x_2 + 205x_3 + 209x_4 + 236x_5 + 228x_6 \leq 2300 \quad \text{(Sódio)} \\ 22 \leq 2x_1 + 2x_2 + 8x_3 + 12x_4 + 8x_5 + 13x_6 \leq 29 \quad \text{(Gordura Saturada)} \\ x_i \geq 10 \quad \text{para } i = 1, \ldots, 6 \end{cases}$$

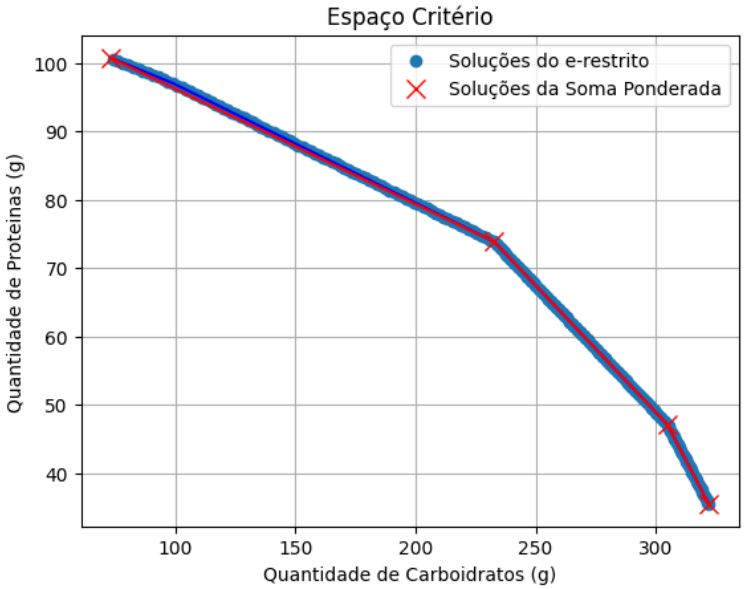

Figura 11: Espaço Critério do problema da dieta 2.

4 Conclusão

Neste projeto de Iniciação Científica, exploramos os fundamentos teóricos e práticos da Otimização Multiobjetivo, com ênfase em métodos exatos, como a Soma Ponderada e o Método do ε-Restrito e em problemas contendo apenas variáveis contínuas. Desenvolvemos e implementamos algoritmos em Python, utilizando a biblioteca PuLP, que foram aplicados a problemas reais, como o transporte de mercadorias, produção e dieta.

Os resultados mostram a relevância da Otimização Multiobjetivo na tomada de decisões, onde é necessário equilibrar múltiplos critérios conflitantes. A análise da aproximação da Fronteira de Pareto revelou que, para cada problema abordado, existe um conjunto de soluções eficientes que permitem ao tomador de decisões escolher a alternativa que melhor atende às suas necessidades específicas, considerando as restrições e objetivos do problema.

Além disso, a implementação prática dos algoritmos demonstrou a aplicabilidade das teorias estudadas em situações reais, reforçando a importância da modelagem matemática e computacional na solução de problemas otimizados em diferentes áreas. Deste modo, este trabalho buscou contribuir para o estudo e aplicações de técnicas de otimização em cenários multiobjetivo, abrindo caminho para futuras pesquisas e aprimoramentos nas metodologias utilizadas.

5 Agradecimentos

Agradecimento especial à UNICAMP/PIBIC/CNPq pelo financiamento desta pesquisa.

Referências

[1] A. Aliano Filho. *Introdução à otimização multiobjetivo: aspectos teóricos e computacionais focando nos métodos exatos*. São Carlos: [Minicurso de Difusão]. ICMC/USP, 2023. URL: http://paginapessoal.utfpr.edu.br/angeloaliano/curso-de-otimizacao-multiobjetivo/SBPO_parte_2.pdf/at_download/file (acesso em 20/02/2024).

[2] C. H. Antunes, M. J. Alves e J. N. Clímaco. *Multiobjective linear and integer programming*. Switzerland: Springer, 2016.

[3] V. Chankong e Y. Y. Haimes. *Multiobjective decision making: Theory and methodology*. New York: Elsevier Science Publishing Co, 1983.

[4] M. Ehrsgott. *Multicriteria optimization*. Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, 2005.

[5] Python Optimization Tools (PyOpt). *PuLP: LP Modelling in Python*. URL: https://github.com/coin-or/pulp (acesso em 19/08/2024).

[6] *Streamlit Documentation*. URL: https://docs.streamlit.io/library/api-reference (acesso em 19/08/2024).