ReTMAT
REVISTA TOCANTINENSE DE MATEMÁTICA

Revista Tocantinense de Matematica
Vol. 1, n. 1, 2026, p. 1–23
DOI: 10.20873/retmat.uft.v1n1.2026.23236
Recebido: 15 de junho de 2026
Aceito: 22 de junho de 2026

Programação linear utilizando Python na solução do método simplex

Márcio Dias de Lima 1[1,*], Aline Mota de Mesquita Assis 2[1] Ricardo da Silva Santos 3[1], Iasmin Gabriele Sousa de Oliveira 4[1], Jair Alcindo Lobo de Melo 5[2]

Resumo

Este estudo destaca a exploração de conceitos básicos de programação linear por meio da abordagem de resolução de problemas pelo método simplex. Empregando uma metodologia que combina pesquisa bibliográfica com aplicação prática, o método simplex foi utilizado para resolver problemas de programação linear tendo como auxílio a linguagem de programação Python. O objetivo principal foi demonstrar a praticidade e eficiência do método simplex quando implementado em linguagem de programação, como Python, além de proporcionar uma compreensão básica da programação linear. Os resultados obtidos foram satisfatórios, demonstrando que os problemas, quando resolvidos através de linguagem de programação, foram solucionados de forma mais rápida e com menos cálculos, destacando a importância da integração entre teoria e prática na resolução de problemas complexos.

Palavras-chave: Programação Linear; Álgebra Linear; Método Simplex; Python

MSC 2020: 90C05; 90C08; 68N19, 15A06

Abstract

This study highlights the exploration of fundamental concepts of linear programming through the problem-solving approach using the simplex method. Employing a methodology that combines a bibliographic review with practical application, the simplex method was utilized to solve linear programming problems with the aid of the Python programming language. The main objective was to demonstrate the practicality and efficiency of the simplex method when implemented in a programming language such as Python, in addition to providing a basic understanding of linear programming. The results obtained were satisfactory, showing that problems solved using a programming language were resolved more quickly and with fewer calculations, underscoring the importance of integrating theory and practice in solving complex problems.

Keywords: Linear Programming; Linear Algebra; Simplex Method; Python

[1] Instituto Federal de Educação, Ciência e Tecnologia, Goiânia, Goiás, Brasil. E-mail: marcio.lima@ifg.edu.br. ORCID: 0000-0003-2782-386X. [*] Autor correspondente.[1] Instituto Federal de Educação, Ciência e Tecnologia, Goiânia, Goiás, Brasil. E-mail: aline.mesquita@ifg.edu.br. ORCID: 0000-0003-3445-5903.[1] Instituto Federal de Educação, Ciência e Tecnologia, Goiânia, Goiás, Brasil. E-mail: ricardo.santos@ifg.edu.br. ORCID: 0000-0002-8671-0956.[1] Instituto Federal de Educação, Ciência e Tecnologia, Goiânia, Goiás, Brasil. E-mail: iasmingabriele46@gmail.com. ORCID: 0009-0008-9781-0056.[2] Instituto Federal de Educação, Ciência e Tecnologia, Belém, Pará, Brasil. E-mail: jair.melo@ifpa.edu.br. ORCID: 0000-0002-7775-2599.

Sumário

1 Introdução

Ao longo de toda história é possível ver um grande avanço da matemática que, por sua vez, coincide com a evolução dos povos, os quais usavam cálculos no comércio, viagens, expansões territoriais, arquiteturas, navegações e em diversas outras áreas da vida. As sociedades continuam avançando em suas descobertas e estudos, formando, atualmente, um mundo cada vez mais tecnológico, tornando, assim, a resolução de problemas, através de *softwares*, uma maneira mais rápida e menos suscetível a erros.

Esses avanços foram de grande importância para áreas de estudo como a pesquisa operacional (PO), que surgiu durante a segunda guerra mundial (1939-1945), na Inglaterra, através de um grupo de cientistas que tinham o objetivo de planejar estratégias e táticas mais eficientes visando o aproveitamento máximo de materiais escassos, originando um modelo de solução denominado método simplex, que, com o fim da guerra, viria a ser útil para facilitar a tomada de decisões em muitos âmbitos sociais [8]. Hoje em dia, sua aplicação pode ser vista em empresas que visam maximizar o lucro, diminuir os custos e utilizar os materiais de maneira inteligente, sendo esse, um processo de otimização [1], e também pode ser encontrado em problemas de transporte, fluxo máximo, caminho mínimo, teoria dos jogos [8], entre tantas outras aplicações. Diante de diversas possibilidades, foram criados vários modelos e técnicas de solução, variando de acordo com a necessidade do problema.

A programação linear é utilizada em problemas que envolvem apenas funções lineares, uma vez que é um conjunto de técnicas que visa encontrar a solução ótima de um problema dado [18]. Essa abordagem é especialmente útil quando se trata de otimização de recursos em contextos como planejamento financeiro, logística e produção. Para resolver problemas de programação linear de forma eficiente, uma das abordagens comuns é utilizar bibliotecas de otimização, como a biblioteca *pulp* em Python, que oferece ferramentas para modelagem e

resolução de problemas lineares de maneira simplificada. Essas bibliotecas podem implementar métodos como o simplex, entre outros, para encontrar a solução ótima.

De acordo com [7], devem ser seguidos alguns passos para se resolver um problema de programação linear, sendo eles, a formulação do problema, modelo matemático, obtenção de uma solução, teste e a implementação, lembrando que às vezes é preciso fazer a reestruturação do modelo após a fase de teste.

Durante a fase de obtenção de uma solução, são usados os métodos numéricos, tais como o método simplex citado anteriormente, o algoritmo de Gauss-Jordan, algoritmo de Yamnitsky-Levin, que podem ser vistos em [6] e o algoritmo de Pontos Interiores em [15], entre outros.

Desta forma, o objetivo central deste trabalho é apresentar uma aplicação do método simplex utilizando a linguagem de programação Python, por ser uma linguagem de código aberto com sintaxe simples e objetiva, "Python é uma linguagem completa, contando com bibliotecas para acessar bancos de dados, [...] podemos utilizar muitas funções já existentes escrevendo poucas linhas de código"[12, p.22]. Por essas características, Python se torna um facilitador no processo de otimização de alguns problemas.

2 Contexto histórico

Desde a antiguidade, o homem busca soluções ótimas para problemas cotidianos e até mesmo aqueles mais teóricos que não representam fatos concretos da realidade, como Euclides que, em seu terceiro livro, buscava a distância máxima e mínima de um ponto a uma circunferência. Com o passar do tempo, veio o surgimento do cálculo e os conceitos de álgebra que puderam ajudar na resolução de diversos problemas nesse âmbito [5]. Segundo [11, p.5],

> A PL poderia ter sido iniciada em torno de 1758 quando os economistas começaram a descrever sistemas econômicos em termos matemáticos. Também, Walras propôs em 1874 um sofisticado modelo matemático que tinha como parte da sua estrutura coeficientes tecnológicos fixados. O famoso matemático Fourier parece ter sido o primeiro a estudar desigualdades lineares para a Mecânica e para a Teoria das Probabilidades. Ele estava interessado em encontrar o ponto mínimo em um poliedro. Ele sugeriu uma solução por uma descida de vértice em vértice para um mínimo, que é o princípio por trás do método simplex desenvolvido por Dantzig. Este é provavelmente o primeiro exemplo, datado de 1826, de um problema de PL. Mais tarde, em 1911, outro matemático famoso, Poussin, considerou o mesmo problema e propôs uma solução similar.

Além desses matemáticos, teve o russo Leonid V. Kantorovich (1912-1986), um economista que voltou seus estudos para a área de programação linear, em 1939 chegou a escrever um livro intitulado Métodos Matemáticos de Organização e Planejamento da Produção, onde resolve um problema de programação linear, contudo seu reconhecimento só se deu 20 anos após sua publicação [16]. O ápice de desenvolvimento da PL se deu durante a segunda Guerra Mundial com o surgimento da Pesquisa Operacional na Inglaterra, onde foi montado um grupo com cientistas, físicos e matemáticos, denominado SCOOP *(Scientific Computation of Optimum Programs)*, em que Dantzig fez seu nome ao criar o método simplex em 1947 [5].

Em termos atuais, [7, p.1] define programação linear (PL) como sendo a “a optimização (minimização ou maximização) de uma função linear, satisfazendo um conjunto de equações e/ou inequações (restrições) igualmente lineares.” E, para [18, p.11], “o termo ‘programação’ significa planejamento e ‘linear’ deixa antever que todas as expressões matemáticas utilizadas são funções lineares”.

Os problemas da programação linear que envolvem duas ou três variáveis podem ser resolvidos manualmente pelo método gráfico, mas para resolver problemas de três até centenas de variáveis, além das inúmeras restrições, são utilizados métodos computacionais, como o método simplex de Dantzig [3]. E a eficiência de um método pode ser avaliada pelo tempo que um algoritmo leva para chegar ao resultado, conforme explica:

> Este cálculo é feito associando-se uma unidade de tempo para cada operação básica que o procedimento executa. Se a dependência do tempo com relação aos dados de entrada for polinomial, o programa é considerado rápido. Se, entretanto, a dependência do tempo for exponencial o programa é considerado lento [9, p.1].

Assim, pode-se dizer que um algoritmo polinomial é aquele que chega ao resultado esperado em tempo hábil, de tal forma que, quanto menos comandos ou iterações um algoritmo tem, mais rápido ele é. Logo, a praticidade de um algoritmo também se torna um fator importante de eficiência. Isso foi importante durante a criação de algoritmos, pois alguns eram exponenciais e, para tanto, foram criados novos métodos de forma a encontrar um algoritmo polinomial e prático.

Em [11], é possível ver uma linha do tempo dos métodos: o primeiro algoritmo foi o simplex, criado por Dantzig em 1947; em 1979 e 1980, Khachiyan fez uso do método dos elipsóides em PL, já que esse método só havia sido utilizado até então para programação convexa, em que as funções não são apenas lineares, entretanto, na década de 70 houve um problema quanto ao uso destes métodos, pois, por um lado, o simplex tinha complexidade exponencial, mas funcionava bem na prática e o método dos elipsóides tinha complexidade polinomial, mas funcionava mal na prática. Foi então que, em 1984, Karmarkar criou o algoritmo de pontos interiores, diminuindo a complexidade em relação ao método de elipsóides. Em 1986, Renegar provou que o uso do método de centros de Huard, escrito em termos da função barreira logarítmica, é polinomial com complexidade igual ao método de Karmarkar e, desde então, foram sendo feitas adaptações e, também, novos métodos para que o uso do algoritmo pudesse resolver problemas com enormes quantidades de variáveis e restrições.

Um exemplo dessas adaptações é a forma híbrida do método simplex com algoritmos de pontos interiores, aplicado em um problema de uma companhia aérea, envolvendo 837 restrições e 12.753.313 variáveis, em que foi utilizado um supercomputador que solucionou o problema em até 5 minutos, conforme descreve [2], mostrando, dessa forma, que com métodos computacionais é possível resolver problemas muito grandes.

3 Formato padrão da programação linear

De acordo com [7], a solução de um Problema de Programação Linear (PPL) segue diversas etapas, sendo a primeira delas a modelagem do problema. Esta fase é marcada pela aplicação da Modelagem Matemática, uma vez que a definição de programação linear envolve a otimização de funções lineares e visa obter as variáveis, as restrições e a função objetivo. Para uma compreensão mais clara, alguns exemplos podem ser encontrados em [13].

Uma vez que a modelagem de um problema novo não é o foco deste estudo e a correspondência entre o sistema real e o modelo formal está propensa a imprecisões, simplificações e lacunas de comunicação, é importante destacar que não existem técnicas precisas que garantam o estabelecimento do modelo de um problema. Para alcançar essa meta, é fundamental empregar a habilidade de análise do problema, conforme é

sugerido em [11].

Matematicamente, podemos definir um PPL conforme [5]:

$$\begin{aligned} &max/min \quad Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n, \\ &sujeito \quad a: \begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n (\leq, =, \geq) b_1, \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n (\leq, =, \geq) b_2, \\ \qquad\qquad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n (\leq, =, \geq) b_m, \\ x_1, x_2, \cdots, x_n \geq 0. \end{cases} \end{aligned}$$

onde Z é a função objetivo, $c_1, c_2, \cdots, c_n$ são os coeficientes de custo, $x_1, x_2, \cdots, x_n$ são as variáveis e todos os a_{mn} são os coeficientes das restrições.

Essa forma também pode ser escrita de maneira matricial, sendo representada na forma:

$$\begin{aligned} &max/min \quad Z = c^T x, \\ &Sujeito \quad a: \begin{cases} Ax (\geq, =, \leq) b, \\ x \geq 0. \end{cases} \end{aligned}$$

onde T é a transposta da matriz,

- $c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$ é o vetor coluna dos coeficientes da função objetivo, com dimensão $n \times 1$. Consequentemente, c^T é um vetor linha com dimensão $1 \times n$;
- $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ é o vetor coluna das variáveis de decisão, com dimensão $n \times 1$ colunas;
- $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$ é a matriz constituída pelos coeficientes das restrição, sendo n a quantidade de colunas, correspondendo à quantidade de variáveis, e m a quantidade de linhas, correspondendo à quantidade de restrições;
- $b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$ é o vetor coluna formado pelos termos independentes das restrições, com dimensão $m \times 1$.

As restrições do tipo $\geq$ podem ser passadas para $\leq$ por meio da multiplicação por (-1), ou seja,

$$a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \geq b_m \Leftrightarrow -a_{m1}x_1 - a_{m2}x_2 - \cdots - a_{mn}x_n \leq -b_m.$$

Logo, observa-se que $min(Z) \Leftrightarrow max(-Z)$, que podem ser escritas na forma matricial e são chamadas de forma canônica da programação linear.

$$\begin{array}{ll} \min & Z = c^T x, \\ \text{sujeito a:} & \begin{cases} Ax \geq b, \\ x \geq 0. \end{cases} \end{array} \qquad \text{ou} \qquad \begin{array}{ll} \max & Z = c^T x, \\ \text{sujeito a:} & \begin{cases} Ax \leq b, \\ x \leq 0. \end{cases} \end{array}$$

Por outro lado, ao transformar as desigualdades das restrições em igualdades, chega-se no formato padrão da PL, ainda podendo ser tanto minimizar quanto maximizar.

$$\begin{array}{ll} min/max & Z = c^T x, \\ sujeito & a : \begin{cases} Ax = b, \\ x \geq 0. \end{cases} \end{array}$$

Essa é uma fase importante para se obter a resolução dos problemas em vários métodos, inclusive o simplex. De maneira básica e prática, para se chegar na forma padrão de uma PL, serão usadas $x_{m+n} \geq 0$ como variáveis de folga, que são variáveis extras, que não estão na função objetivo, e são adicionadas de forma a tornar a função em uma igualdade. Assim, para desigualdades do tipo "$\leq$", elas serão adicionadas, conforme apresentado no sistema de Equações (1):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + x_{m+1} = b_1, \\ a_{21}x_2 + a_{22}x_2 + \ldots + a_{1n}x_n + x_{m+2} = b_2, \\ \qquad\qquad\qquad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + x_{m+n} = b_m. \end{cases} \tag{1}$$

Entretanto, quando se tem desigualdades do tipo " $\geq$", a variável de folga é subtraída.

4 O Método Simplex

Um problema de programação linear busca a otimização de uma certa situação, ou seja, a maximização ou minimização de uma função objetivo, e este resultado encontrado é chamado de solução ótima. Entretanto, durante o processo, são encontradas algumas outras soluções que serão definidas de acordo com a etapa em que cada uma é encontrada e o método de resolução utilizado. Essas soluções alternativas podem ser relevantes para entender o comportamento do problema e a eficácia dos métodos utilizados para sua resolução. Desde sua concepção em 1947, o método simplex tem sido um dos algoritmos mais amplamente empregados em programação linear, devido à sua capacidade de simplificar o tratamento de múltiplas variáveis.

Para compreender este método, será utilizado o Problema 1, que é uma adaptação do Problema apresentado em [13, p. 73], cuja resolução será feita com o método simplex, porém, inicialmente, o problema será resolvido de forma manual, visando descrever os processos e, em seguida, aplicá-lo em uma linguagem de programação, evidenciando como a forma computacional pode ser útil para a resolução de problemas de programação linear, principalmente, através do método simplex.

Problema 1. Uma Indústria fabrica calças e bermudas com as seguintes margens de contribuição por unidade:
- Margem de Contribuição da Calça: R$7,00;
- Margem de Contribuição da Bermuda: R$4,00.

Estes produtos passam respectivamente pelos departamentos de corte e costura, consumindo, em cada um deles, as seguintes horas:

Tabela 1: Limite de horas na produção.

	Departamento de corte	Departamento de costura
Calças	5h	3h
Bermudas	3h	2h

Fonte: Elaborado pelos autores.

A capacidade de produção de cada departamento limita-se a 40 horas para o departamento de corte e 25 horas para o departamento de costura. Os diretores da indústria desejam saber qual seria a quantidade ideal de calças e bermudas que deveriam ser produzidas com o intuito de se obter melhor margem de contribuição para a empresa, de acordo com as restrições existentes em cada um deles.

Para solucionar problemas utilizando o método simplex, é essencial seguir algumas etapas até alcançar um ponto em que tais passos serão reiterados, visando encontrar a solução ótima, conforme aponta [10] e [14], e detalha-se a seguir.

A resolução ocorre somente com o uso de tabelas, onde as informações necessárias para a solução do problema são separadas e organizadas. Nelas são feitas operações entre linhas, criando tabelas equivalentes até chegar à solução ótima. O primeiro passo, então, será colocar o problema na forma padrão, em que são adicionadas as variáveis de folgas a cada restrição (nesse caso: x_3 e x_4), transformando todas as inequações em equações. Logo, temos:

$$\begin{aligned} max \quad & Z = 7x_1 + 4x_2, \\ sujeito \quad a: & \begin{cases} 5x_1 + 3x_2 + x_3 = 40, \\ 3x_1 + 2x_2 + x_4 = 25, \\ x_i \geq 0, i = 1, 2, 3, 4. \end{cases} \end{aligned}$$

A função objetivo será rearranjada para que todos os elementos estejam de um único lado da equação, resultando em $Z - 7x_1 - 4x_2 = 0$ e, em seguida, ficará com as outras equações formando o sistema (S) descrito a seguir:

$$(S): \begin{cases} Z - 7x_1 - 4x_2 = 0, \\ 5x_1 + 3x_2 + x_3 = 40, \\ 3x_1 + 2x_2 + x_4 = 25, \\ x_i \geq 0, i = 1, 2, 3, 4. \end{cases}$$

O próximo passo é encontrar uma solução básica viável, em que serão selecionadas variáveis para serem igualadas a zero, obtendo, assim, valores não nulos para as demais. As variáveis que serão igualadas a zero são chamadas de não básicas, enquanto as que sobrarem serão as variáveis básicas, ou seja, elas vão ter valores não nulos. Para uma solução inicial, as variáveis originais do problema serão designadas como as não básicas. Com base nisso, pode-se criar a Tabela 2:

Tabela 2: Valores iniciais do método simplex.

Linha	Variáveis Básicas	x_1	x_2	x_3	x_4	b
1	x_3	5	3	1	0	40
2	x_4	3	2	0	1	25
3	Z	-7	-4	0	0	0

Fonte: Elaborada pelos autores.

Como $x_1 = x_2 = 0$ por serem variáveis não básicas, teremos, então $x_3 = 40$ e $x_4 = 25$, como pode ser observado na Tabela 2, e aplicando esses valores na função objetivo, tem-se um lucro de $Z = 0$, que não é o máximo esperado, dessa forma, é preciso mudar uma das variáveis básicas para que se possa encontrar novas soluções viáveis até chegar à solução ótima. Isso acontecerá, no caso de maximizar, quando todos os coeficientes da função objetivo forem maiores ou iguais a zero, e no caso de minimizar uma função, quando todos os coeficientes da função objetivo forem menores ou iguais a zero.

Como o problema que está sendo resolvido é do tipo maximizar, então o objetivo é tornar os coeficientes de $Z \geq 0$, para isso é preciso realizar um processo chamado pivotamento, o qual realiza operações entre as linhas, levando a um sistema equivalente, funcionando da seguinte forma:

$$\begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{ij} \\ \vdots \\ a_{mj} \end{pmatrix} \implies \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

Isso significa que um elemento a_{ij} é escolhido para ser o elemento pivô, esse será transformado em 1, por meio de operações elementares, enquanto os demais elementos na mesma linha são ajustados para zero, pelo mesmo processo de operações elementares de linha. Faz-se um processo de busca para saber qual será a linha e coluna pivô e a intersecção entre elas será chamado de elemento pivô. Este processo consiste em escolher qual variável básica vai se tornar não básica e vice-versa. A escolha da coluna pode ser feita de maneira aleatória, mas para chegar no resultado ótimo com mais rapidez, será

analisado qual é o menor coeficiente de Z, que no caso é -7, logo, a coluna x_1 será a coluna pivô escolhida. Já para a linha, será analisada qual o quociente mínimo entre $\frac{b_i}{a_{ij}}$, sendo a_{ij} os coeficientes da coluna pivô, com exceção da função objetivo. Assim, tem-se $(\frac{40}{5}, \frac{25}{3})$ e o mínimo entre eles é 8, logo, nossa linha pivô é a linha 1 e 5 será o elemento pivô. Na Tabela 3 podemos ver como fica esse processo.

Tabela 3: Escolha da linha e coluna pivô.

Linha	Variáveis Básicas	x_1	x_2	x_3	x_4	b
1	x_3	5	3	1	0	40
2	x_4	3	2	0	1	25
3	Z	-7	-4	0	0	0

Fonte: Elaborada pelos autores.

Por meio do pivotamento, foi escolhida a variável x_1 para ser a nova variável básica e x_3 para ser variável não básica. Para criar uma nova tabela, a linha pivô será transformada da seguinte forma: linha pivô atual dividida pelo número pivô. E as outras linhas serão transformadas em: linha atual menos coeficiente da coluna pivô da linha atual multiplicado pela nova linha pivô. Assim, a nova linha pivô vai ser igual a:

Tabela 4: Tabela com a nova linha pivô.

Linha	Variáveis básicas	x_1	x_2	x_3	x_4	b
1	x_1	1	$\frac{3}{5}$	$\frac{1}{5}$	0	8
2	x_4	3	2	0	1	25
3	Z	-7	-4	0	0	0

Fonte: Elaborado pelos autores.

E as linhas 2 e 3 serão transformadas da seguinte forma:

	x_1	x_2	x_3	x_4	b
Linha 2	3	2	0	1	25
3· nova linha pivô	3	$\frac{9}{5}$	$\frac{3}{5}$	0	24
Subtração	0	$\frac{1}{5}$	$-\frac{9}{5}$	1	1

	x_1	x_2	x_3	x_4	b
Linha 3	-7	-4	0	0	0
-7· nova linha pivô	-7	$\frac{-21}{5}$	$\frac{-7}{5}$	0	-56
Subtração	0	$\frac{1}{5}$	$\frac{7}{5}$	0	56

Assim, a nova Tabela fica:

Tabela 5: Valores atualizados do método simplex.

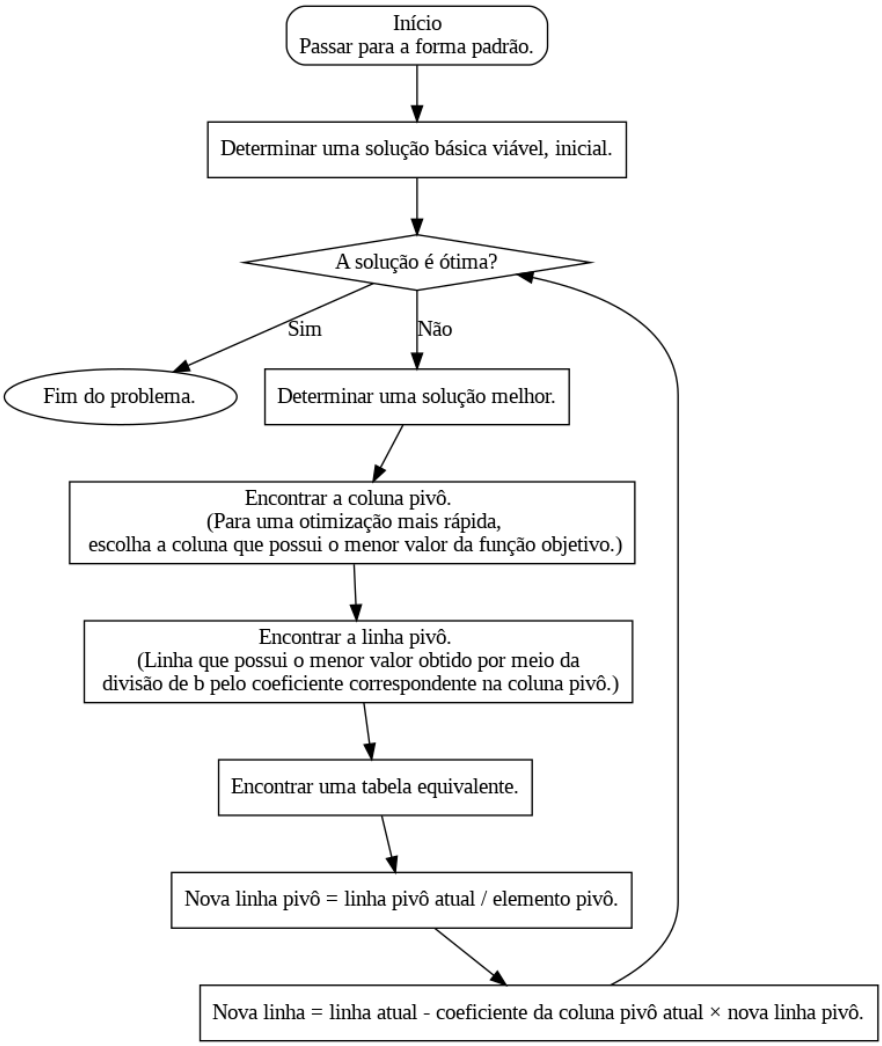
Linha	Variáveis Básicas	x_1	x_2	x_3	x_4	b
1	x_1	1	$\frac{3}{5}$	$\frac{1}{5}$	0	8
2	x_4	0	$\frac{1}{5}$	$-\frac{9}{5}$	1	1
3	Z	0	$\frac{1}{5}$	$\frac{7}{5}$	0	56

Fonte: Elaborado pelos autores.

Como não há outro valor negativo na função objetivo, tem-se que a solução ótima foi obtida, ou seja, quando $x_1 = 8$ e $x_4 = 1$ e $x_2 = x_3 = 0$ o lucro será de 56, que é o lucro máximo. Caso a solução ótima não tivesse sido encontrada, seria preciso repetir todo o processo a partir do pivotamento.

As etapas acima descritas pode ser sintetizadas conforme mostra a Figura 1.

Figura 1: Organograma do método simplex.



Fonte: Adaptado de [10].

5 Aplicação em Python

É comum, em muitos trabalhos, o uso do *Excel* na aplicação do método simplex, como pode ser visto em [8, 13] e [18], entre outros. Isso ocorre devido ao fato de o método simplex envolver muitas tabelas. No entanto, neste trabalho, optou-se por utilizar a linguagem de programação Python, pois, dentre suas diversas bibliotecas, há uma específica para resolver problemas de programação linear, simplificando a resolução e resultando em soluções mais rápidas, uma vez que [17, p.10]:

> Python possui um conjunto de regras de escrita que definem como um algoritmo é interpretado, possuindo um layout visual relativamente organizado e utilizando, com frequência, palavras em Inglês. Python usa indentação como delimitação de blocos.

Desta forma, pretende-se analisar a contribuição da linguagem de programação Python na resolução de problemas de programação linear, o que é, particularmente, relevante, uma vez que problemas em grande escala seriam muito complexos e inviáveis de serem resolvidos manualmente.

A plataforma utilizada foi o *Google Colaboratory* e o código que será descrito funciona com a biblioteca *pulp*, a qual resolve problemas de maximização e minimização. Sendo assim, é essencial realizar sua instalação antes de começar a escrita do código.

Após a instalação da biblioteca, ela será importada diretamente no código. Em seguida, é necessário criar uma instância que representará o problema a ser resolvido (Problema 1). Por ser um problema de maximização, será usado *"LpMaximize"*, conforme sugere [4]. Para o caso de minimização, no lugar de *"LpMaximize"* usa-se *"LpMinimize"*.

A próxima etapa será definir as variáveis do problema, usando *"lowBound"* para colocar um limite inferior nas restrições, que, como pode ser visto no código, o limite é zero. A função objetivo e as restrições devem ser adicionadas na instância que foi criada inicialmente. A aplicação com um problema mais abrangente pode ser exemplificada no seguinte caso, retirado de [10, p.33].

Problema 2. Uma companhia de propaganda deseja planejar uma campanha em 03 diferentes meios: tv, rádio e revistas. Pretende-se alcançar o maior número de clientes possível. Um estudo de mercado resultou nos dados da Tabela 6, sendo os valores válidos para cada veiculação da propaganda. A companhia não quer gastar mais de $800.000 e adicionalmente deseja:

a) No mínimo 2 milhões de mulheres sejam atingidas;
b) Gastar no máximo $500.000 com TV;
c) No mínimo 03 veiculações ocorram no horário normal na TV;
d) No mínimo 02 veiculações ocorram no horário nobre na TV;
e) Número de veiculações no rádio, e nas revistas, devem ficar entre 05 e 10, para cada meio de divulgação.

Tabela 6: Valores dos ganhos com cada veiculação de propaganda.

	TV horário normal	TV horário nobre	Rádio	Revistas
Custo	40.000	75.000	30.000	15.000
Clientes atingidos	400.000	900.000	500.000	200.000
Mulheres atingidas	300.000	400.000	200.000	100.000

Fonte: [10, p. 30].

O objetivo inicial para resolver o problema é formular um modelo de PL adequado à situação, determinando o número de veiculações a serem feitas em cada meio de comunicação, de modo a atingir o máximo possível de clientes. As variáveis de decisão são:

x_1 = Número de exposições em horário normal na TV.
x_2 = Número de exposições em horário nobre na TV.
x_3 = Número de exposições feitas utilizando rádio.
x_4 = Número de exposições feitas utilizando revistas.
Com as informações dadas, chega-se ao seguinte modelo:

$$
\begin{aligned}
max \quad & Z = 400.000x_1 + 900.000x_2 + 500.000x_3 + 200.000x_4, \\
sujeito \quad & a: \begin{cases}
40.000x_1 + 75.000x_2 + 30.000x_3 + 15.000x_4 \leq 800.000, \\
300.000x_1 + 400.000x_2 + 200.000x_3 + 100.000x_4 \geq 2.000.000, \\
40.000x_1 + 75.000x_2 \leq 500.000, \\
x_1 \geq 3, \quad x_2 \geq 2, \quad 5 \leq x_3 \leq 10, \quad 5 \leq x_4 \leq 10, \\
x_1, \quad x_2, \quad x_3, \quad x_4 \geq 0.
\end{cases}
\end{aligned}
$$

Ao aplicar a linguagem de programação Python tem-se o Algoritmo 1:

Algoritmo 1 Resolução do Problema 2 utilizando Programação Linear

1: **Início:** Inicia o programa.
2: **Importar biblioteca:** A biblioteca pulp é importada para resolver problemas de programação linear.
3: **Problema:** Criar um problema de programação linear chamado "propaganda" para maximização.
4: **Variáveis:** Definir as variáveis de decisão x_1, x_2, x_3 e x_4, todas com limite mínimo de zero.
5: **Função objetivo:** Adicionar a função objetivo $(400.000 \times x_1 + 900.000 \times x_2 + 500.000 \times x_3 + 200.000 \times x_4)$ ao problema de maximização.
6: **Restrições:** Adicionar as seguintes restrições ao problema:
$40.000 \times x_1 + 75.000 \times x_2 + 30.000 \times x_3 + 15.000 \times x_4 \leq 800.000$.
$300.000 \times x_1 + 400000 \times x_2 + 200.000 \times x_3 + 100.000 \times x_4 \geq 2.000.000$.
$40000 \times x_1 + 75000 \times x_2 \leq 500.000$.
$x_1 \geq 3$.
$x_2 \geq 2$.
$x_3 \geq 5$.
$x_3 \leq 10$.
$x_4 \geq 5$.
$x_4 \leq 10$.
7: **Resolver:** Resolver o problema de programação linear.
8: **Apresentar a solução:** Exibir o status da solução (ótimo, inviável, não resolvido, etc.)
9: **Apresentar os variáveis** Exibir os valores das variáveis de decisão x_1, x_2, x_3 e x_4 encontrados na solução.
10: **Solução viável:** Exibir o valor da função objetivo no ponto ótimo.
11: **Fim**.

Para utilizar o algoritmo em Python, não é necessário passar para a forma canônica nem para a forma padrão, detalhe que é de suma importância na forma manual. Além disso, ao programar em Python, não é necessário passar por tantos processos e cálculos, de forma que se chega mais rápido ao resultado e sem erros.

A seguir apresenta-se mais uma aplicação (Problema **??**) com um exemplo hipotético.

Problema 3. Uma empresa de energia elétrica opera cinco usinas, identificadas por A, B, C, D e E. As usinas utilizam diferentes fontes de energia e possuem custos operacionais e capacidades máximas de geração distintos, conforme apresentado na Tabela a seguir:

Usina	Fonte	Custo (US$/MWh)	Capacidade (MWh)
A	Hidrelétrica	32	300
B	Solar	38	250
C	Gás natural	24	400
D	Carvão	20	300
E	Biomassa	35	200

A empresa deve atender a uma demanda de 1000 MWh. Além disso, por exigência ambiental, pelo menos 40% da energia fornecida deve ser proveniente de fontes renováveis. Neste problema, são consideradas renováveis as fontes utilizadas pelas usinas A, B e E.

O objetivo é determinar quanto cada usina deve produzir para minimizar o custo operacional total, atendendo à demanda, ao percentual mínimo de energia renovável e

aos limites de capacidade.

Variáveis de decisão.

Sejam

x_i = quantidade de energia produzida pela usina i, em MWh, $i \in \{A, B, C, D, E\}$.

Função objetivo.

O custo operacional total é dado por:

$$\min Z = 32x_A + 38x_B + 24x_C + 20x_D + 35x_E. \tag{2}$$

Restrições.

A produção total deve ser igual à demanda:

$$x_A + x_B + x_C + x_D + x_E = 1000. \tag{3}$$

Como 40% de 1000 MWh correspondem a 400 MWh, a exigência de participação mínima de fontes renováveis é representada por:

$$x_A + x_B + x_E \geq 400. \tag{4}$$

Os limites de capacidade são:

$$x_A \leq 300, \qquad x_B \leq 250, \qquad x_C \leq 400, \tag{5}$$

$$x_D \leq 300, \qquad x_E \leq 200. \tag{6}$$

As condições de não negatividade são:

$$x_A, x_B, x_C, x_D, x_E \geq 0. \tag{7}$$

Esta formulação representa um problema de programação linear que minimiza os custos operacionais, sujeito às restrições de demanda e disponibilidade de cada usina,ver Algoritmo 2. Todos os códigos em linguagem de programação Python estão no Apêndice A desse trabalho.

Algoritmo 2 Resolução do Problema 3 utilizando Programação Linear

Início: Iniciar o programa.
Importar biblioteca: Importar a biblioteca `pulp` para resolver o problema de PL.
Definir usinas: Definir o conjunto de usinas como

$$\texttt{usinas} = [A, B, C, D, E].$$

Definir fontes renováveis: Definir as usinas que utilizam fontes renováveis como

$$\texttt{renovaveis} = [A, B, E].$$

Definir custos: Definir os custos de geração, em dólares por MWh, como

$$\texttt{custos} = [32, 38, 24, 20, 35].$$

Definir capacidades: Definir as capacidades máximas das usinas, em MWh, como

$$\texttt{capacidades} = [300, 250, 400, 300, 200].$$

Definir demanda: Definir a demanda total de energia como

$$\texttt{demanda} = 1000.$$

Definir geração renovável mínima: Calcular a quantidade mínima de energia renovável:

$$\texttt{minimo\_renovavel} = 0{,}40 \times 1000 = 400.$$

Problema: Criar um problema de programação linear denominado `problema_energia`, com objetivo de minimização.
Variáveis: Definir as variáveis contínuas $x_A, \ldots, x_E$, com limite inferior igual a zero.
Função objetivo: Adicionar ao problema a função

$$\min Z = 32x_A + 38x_B + 24x_C + 20x_D + 35x_E.$$

Restrição de demanda: Adicionar a restrição

$$x_A + x_B + x_C + x_D + x_E = 1000.$$

Restrição de fontes renováveis: Adicionar a restrição

$$x_A + x_B + x_E \geq 400.$$

Restrições de capacidade: Adicionar as restrições

$$x_A \leq 300, \quad x_B \leq 250, \quad x_C \leq 400, \quad x_D \leq 300, \quad x_E \leq 200.$$

Resolver: Resolver o problema utilizando o solver CBC utilizando a biblioteca PuLP.
Exibir solução: Apresentar o status da solução obtida.
for cada usina $i \in \{A, B, C, D, E\}$ **do**
    Exibir a quantidade de energia produzida pela usina i.
end for
Geração renovável: Calcular e exibir

$$x_A + x_B + x_E.$$

Custo total: Calcular e exibir o valor ótimo da função objetivo.
Fim: Encerrar o programa.

6 Análise da Eficiência e Rapidez Computacional dos Modelos

A validação dos modelos matemáticos implementados via biblioteca *PuLP* em ambiente *Google Colaboratory* demonstrou que a abordagem por Programação Linear (PL) e Programação Linear Inteira Mista (PLIM) é altamente robusta, tanto sob a ótica da eficiência operacional quanto da rapidez computacional.

Para fundamentar a viabilidade técnica das soluções apresentadas nos Problemas 1, 2 e 3, a análise dos resultados obtidos deve considerar a mecânica do algoritmo *Simplex* e do método *Branch-and-Bound* operados pelo *solver* padrão COIN-OR Branch-and-Cut (CBC).

6.1 Rapidez de Convergência e Complexidade Algorítmica

Embora o método *Simplex* possua uma complexidade de pior caso exponencial, a sua aplicação prática na literatura científica consagra-se por apresentar um tempo de execução que cresce, em média, de forma linear ou polinomial em relação ao número de restrições [10]. Nos três cenários testados, o tempo de CPU requerido para a convergência e obtenção da solução ótima global foi inferior a 0,1 segundos.

Essa rapidez se justifica porque o algoritmo não realiza uma busca exaustiva (força bruta) no espaço de soluções, o que seria computacionalmente inviável em problemas de grande escala. Em vez disso, o *Simplex* adota uma estratégia geométrica iterativa, avaliando exclusivamente os pontos extremos (vértices) do poliedro convexo definido pelas restrições lineares. A transição de uma base factível para outra ocorre apenas se houver melhoria no valor da função objetivo, reduzindo drasticamente o esforço computacional e garantindo a convergência imediata para problemas de pequeno e médio porte.

6.2 Eficiência Operacional e Análise de Sensibilidade

A eficiência da implementação não se limita à velocidade de processamento, mas estende-se à qualidade e utilidade das respostas geradas para a tomada de decisão gerencial:

- **Alocação Ótima de Recursos (Problema 1):** O modelo de planejamento da produção de vestuário demonstrou eficiência máxima ao esgotar completamente as 40 horas do departamento de corte e as 25 horas do departamento de costura. O algoritmo identificou matematicamente o ponto de tangência exato entre as restrições, eliminando qualquer ociosidade oculta no processo fabril.
- **Modularidade e Escalabilidade (Problemas 2 e 3):** A modelagem utilizada para a campanha publicitária (Problema 2) e para o despacho das usinas de energia (Problema 3) mostrou-se altamente escalável. No Problema 3, além dos custos e das capacidades de geração, o modelo incorpora uma exigência mínima de participação de fontes renováveis. A expansão do número de usinas demanda principalmente a atualização dos dicionários de dados e da classificação das fontes de energia, sem alterar a estrutura geral do modelo de programação linear.

Adicionalmente, a obtenção da solução via *PuLP* fornece variáveis duais associadas às restrições (os chamados “preços-sombra”). Essa propriedade matemática enriquece a

discussão econômica, pois permite que os gestores compreendam o impacto marginal de uma unidade adicional de recurso (como um dólar adicional no orçamento publicitário, uma hora adicional disponível na produção ou o aumento da participação mínima exigida de fontes renováveis) sobre o valor global da função objetivo, consolidando a Programação Linear como uma ferramenta indispensável de otimização de sistemas.

7 Conclusão

Em um mundo, onde diariamente aparecem novos problemas que podem ser resolvidos matematicamente, é preciso buscar a melhor forma para calculá-los. Para lidar com problemas grandes, que envolvem várias variáveis, faz-se necessário o auxílio de tecnologias digitais que viabilizem o encontro de soluções de forma eficiente e eficaz. Por isso, este artigo propôs o estudo do método simplex, que possui uma aplicabilidade mais ampla, com a linguagem de programação Python.

Dentre os modos de resolução pelo método simplex, a forma manual envolve muitos cálculos, com a álgebra intrínseca desde a formulação do problema até o resultado ótimo, um exemplo disso é a definição de um problema de programação linear que se trata de um conjunto convexo e as restrições são hiperplanos que delimitam essa forma. Assim, a álgebra desempenha um papel significativo na programação linear e estudos futuros podem destacar a sua importância no contexto do método simplex.

Por meio da utilização da linguagem de programação Python, foi possível resolver computacionalmente os problemas propostos, na qual todos os cálculos são automatizados com o auxílio de uma biblioteca específica para problemas de programação linear. Isso significa que tarefas que antes exigiam muita atenção e cálculos, podem agora ser resolvidas com apenas algumas linhas de código. Além disso, a simplicidade foi um critério de escolha para a linguagem de programação, o que torna as linhas de código compreensíveis e facilita as adaptações necessárias para cada problema. Dessa forma, percebe-se que, com o algoritmo utilizado, tornou-se mais rápida a solução de problemas de programação linear, apresentando o resultado quase que instantaneamente.

Em suma, a aplicação do método simplex, utilizando a linguagem de programação Python, demonstrou-se uma ferramenta valiosa para resolver problemas de programação linear de maneira eficiente. A abordagem manual ressaltou a importância da álgebra linear na formulação e resolução desses problemas, enquanto a abordagem computacional destaca a rapidez e praticidade proporcionadas pela automação dos cálculos. Ao combinar os conhecimentos teóricos da álgebra linear com as facilidades oferecidas pela programação, é possível obter soluções precisas e otimizadas para uma ampla gama de problemas do mundo real. Essa integração entre teoria e prática abre caminho para novas aplicações e avanços na área da otimização, contribuindo para a resolução de desafios complexos em diversas áreas, desde a logística até o planejamento financeiro. Assim, o estudo e aplicação do método simplex e sua implementação computacional representam um importante passo em direção à busca por soluções eficazes e inovadoras para problemas do cotidiano.

Referências

[1] Luara Almeida, Glêndara Martins, and Warley Silva. Otimização de processos utilizando a programação linear. *Enciclopédia Biosfera*, 9(16), 2013.

[2] Robert E Bixby, John W Gregory, Irvin J Lustig, Roy E Marsten, and David F Shanno. Very large-scale linear programming: A case study in combining interior point and simplex methods. *Operations Research*, 40(5):885–897, 1992.

[3] José Luiz Boldrini, Sueli IR Costa, VL Figueredo, and Henry G Wetzler. *Álgebra linear*. Harper & Row, 1980.

[4] Giovanni Castilho Brogiato. Programação linear, linear inteira e os algoritmos simplex e branch and bound: Problemas e aplicações em otimização. Trabalho de conclusão de curso, Universidade Federal de São Carlos, São Carlos, Brazil, 2021.

[5] Emílio Capamba Evaristo. *Programação linear: um manual para o professor*. PhD thesis, Universidade da Beira Interior (Portugal), 2020. Acessado em 20/08/2024.

[6] Paulo Feofiloff. *Algoritmos de Programação Linear Programação Linear Concreto*. Universidade de São Paulo, 2005.

[7] J Júdice, P Martins, M Pascoal, and J Santos. Programação linear. *Coimbra, Portugal: Departamento de Matemática-Universidade de Coimbra*, 2006.

[8] Erico Fagundes Anicet Lisboa. Pesquisa operacional. *Apostila da disciplina. Rio de Janeiro–RJ*, 2002.

[9] Pedro Luiz Aparecido Malagutti and Hipertexto Pitágoras. P versus np, 2002.

[10] Fernando Augusto Silva Marins. Introdução à pesquisa operacional. *São Paulo: Cultura Acadêmica: Universidade Estadual Paulista*, 2011.

[11] Marco Antonio Figueiredo Menezes. Programação linear. *Goiânia: Universidade Católica de Goiás, Departamento de Computação*, 2006.

[12] Nilo Ney Coutinho Menezes. *Introdução à Programação com Python: Algoritmos e Lógica de Programação para Iniciantes*. Novatec, São Paulo, 2010.

[13] Elias Dib Caddah Neto. A importância da programação linear no processo decisório/programming the importance of linear in decision making. *Revista FSA (Centro Universitário Santo Agostinho)*, 1(1):69–80, 2015.

[14] Agnaldo dos Santos Pereira. Método simplex e sua aplicação na resolução de problemas de programação linear em um curso técnico de administração. Dissertação de mestrado profissional em matemática, Universidade Estadual do Maranhão (UEMA), São Luís, Brazil, 2020.

[15] Leizer de Lima Pinto and Marco Antonio Figueiredo Menezes. Implementação de algoritmos simplex e pontos interiores para programação linear. *Revista EVS-Revista de Ciências Ambientais e Saúde*, 35(2):225–246, 2008.

[16] Adília Oliveira Das Neves Rafael. Programação linear e algumas extensões. *Faculdade de Ciências universidade do Porto, Departamento de Matemática*, 2014.

[17] Ana Luisa Soubhia, Elias Teixeira Costa, Flavio Luan Müller Freitas, Laís Brum Menezes, Marcos Alves dos Santos, and Vinícius Maran. *Python 101*. Universidade Federal de Santa Maria (UFSM), Cachoeira do Sul, RS, versão 1.0-2019/2 edition, 2019.

[18] Orisvaldo Gustavo de Sousa. *Aspectos práticos da programação linear*. Universidade Federal de Santa Catarina, 2009.

A Códigos em Python

A.1 Resolução do Problema da Indústria (Calças e Bermudas)

```
# Instala a biblioteca de Programação Linear no Colab
!pip install pulp

import pulp

# 1. Criação do Problema (Objetivo: Maximizar)
problema_1 = pulp.LpProblem("Maximizacao_Margem_Contribuicao", pulp.
    LpMaximize)

# 2. Definição das Variáveis de Decisão
x1 = pulp.LpVariable("Calcas", lowBound=0, cat='Integer')
x2 = pulp.LpVariable("Bermudas", lowBound=0, cat='Integer')

# 3. Função Objetivo: Maximizar a Margem de Contribuição
problema_1 += 7 * x1 + 4 * x2, "Z_Margem_Total"

# 4. Restrições de Capacidade dos Departamentos
problema_1 += 5 * x1 + 3 * x2 <= 40, "Capacidade_Corte"
problema_1 += 3 * x1 + 2 * x2 <= 25, "Capacidade_Costura"

# 5. Resolução do Problema
problema_1.solve()

# 6. Exibição dos Resultados
print(f"Status␣da␣Solução:␣{pulp.LpStatus[problema_1.status]}\n")

if problema_1.status == pulp.LpStatusOptimal:
        print("===␣PLANO␣DE␣PRODUCAO␣===")
        print(f"Quantidade␣de␣Calcas␣(x1)␣␣:␣{int(x1.varValue)}␣
            unidades")
        print(f"Quantidade␣de␣Bermudas␣(x2):␣{int(x2.varValue)}␣
            unidades")
        print("-" * 35)

        # Valor da margem de contribuição máxima alcançada
        margem_maxima = pulp.value(problema_1.objective)
        print(f"Margem␣de␣Contribuicao␣Total␣Maxima␣(Z):␣R$␣{
            margem_maxima:.2f}")

        # Uso real dos recursos para conferencia das restricoes
        horas_corte = 5 * x1.varValue + 3 * x2.varValue
        horas_costura = 3 * x1.varValue + 2 * x2.varValue
```

```
        print("\n=== USO DOS RECURSOS ===")
        print(f"Horas utilizadas no Corte   : {horas_corte}h (Limite: 40h)")
        print(f"Horas utilizadas na Costura: {horas_costura}h (Limite: 25h)")
else:
        print("Nao foi possivel encontrar uma solucao otima.")
```

Algoritmo em linguagem Python 1: Algoritmo Simplex para Fábrica de Calças

A.2 Resolução do Problema da Campanha Publicitária

```
# Instala a biblioteca de Programação Linear no Colab
!pip install pulp

import pulp

# 1. Criação do Problema (Objetivo: Maximizar)
problema = pulp.LpProblem("Campanha_Publicitaria", pulp.LpMaximize)

# 2. Definição das Variáveis de Decisão
x1 = pulp.LpVariable("TV_Normal", lowBound=3, cat='Integer')
x2 = pulp.LpVariable("TV_Nobre", lowBound=2, cat='Integer')
x3 = pulp.LpVariable("Radio", lowBound=5, upBound=10, cat='Integer')
x4 = pulp.LpVariable("Revistas", lowBound=5, upBound=10, cat='Integer')

# 3. Função Objetivo: Maximizar Total de Clientes Atingidos
problema += 400000 * x1 + 900000 * x2 + 500000 * x3 + 200000 * x4, "Z_Total_Clientes"

# 4. Restrições
problema += 40000 * x1 + 75000 * x2 + 30000 * x3 + 15000 * x4 <= 800000, "Orcamento_Total"
problema += 300000 * x1 + 400000 * x2 + 200000 * x3 + 100000 * x4 >= 2000000, "Minimo_Mulheres"
problema += 40000 * x1 + 75000 * x2 <= 500000, "Teto_Gastos_TV"

# 5. Resolução do Problema
problema.solve()

# 6. Exibição dos Resultados
print(f"Status da Solução: {pulp.LpStatus[problema.status]}\n")

if problema.status == pulp.LpStatusOptimal:
        print("=== PLANO DE VEICULACAO IDEAL ===")
        print(f"TV Horário Normal (x1): {int(x1.varValue)} inserções")
        print(f"TV Horário Nobre  (x2): {int(x2.varValue)} inserções")
        print(f"Rádio             (x3): {int(x3.varValue)} inserções")
        print(f"Revistas          (x4): {int(x4.varValue)} inserções")
        print("-" * 35)

        # Cálculo e exibição do valor máximo alcançado
        lucro_total = pulp.value(problema.objective)
```

```
        print(f"Total de clientes atingidos (Z): {int(lucro_total):.0f} pessoas")

        # Verificação de Custos
        custo_total = (40000 * x1.varValue) + (75000 * x2.varValue) + (30000 * x3.varValue) + (15000 * x4.varValue)
        print(f"Custo total da campanha: ${custo_total:,.2f}")
else:
        print("Não foi possível encontrar uma solução ótima para as restrições fornecidas.")
```

Algoritmo em linguagem Python 2: Algoritmo Simplex para Campanha Publicitária

A.3 Resolução do Problema das Usinas de Energia

```
# !pip install pulp

import pulp

# Dados do problema
usinas = ["A", "B", "C", "D", "E"]
renovaveis = ["A", "B", "E"]

fontes = {
        "A": "Hidreletrica",
        "B": "Solar",
        "C": "Gas natural",
        "D": "Carvao",
        "E": "Biomassa"
}

custos = {
        "A": 32,
        "B": 38,
        "C": 24,
        "D": 20,
        "E": 35
}

capacidades = {
        "A": 300,
        "B": 250,
        "C": 400,
        "D": 300,
        "E": 200
}

demanda = 1000
percentual_minimo_renovavel = 0.40
geracao_minima_renovavel = (
demanda * percentual_minimo_renovavel
)

# Criacao do problema: minimizar o custo operacional
problema_energia = pulp.LpProblem(
```

```
"Despacho_Energia_Com_Renovaveis",
pulp.LpMinimize
)

# Variaveis de decisao: producao de cada usina em MWh
producao = pulp.LpVariable.dicts(
"Producao",
usinas,
lowBound=0,
cat="Continuous"
)

# Funcao objetivo
problema_energia += pulp.lpSum(
custos[i] * producao[i]
for i in usinas
), "Custo_Total"

# Restricao de atendimento da demanda
problema_energia += pulp.lpSum(
producao[i]
for i in usinas
) == demanda, "Demanda_Total"

# Restricao de participacao minima de fontes renovaveis
problema_energia += pulp.lpSum(
producao[i]
for i in renovaveis
) >= geracao_minima_renovavel, "Minimo_Renovavel"

# Restricoes de capacidade das usinas
for i in usinas:
problema_energia += (
producao[i] <= capacidades[i],
f"Capacidade_{i}"
)

# Resolucao pelo solver CBC
status = problema_energia.solve(
pulp.PULP_CBC_CMD(msg=False)
)

# Exibicao dos resultados
print(f"Status: {pulp.LpStatus[status]}")

if pulp.LpStatus[status] == "Optimal":
print("\n=== PLANO OTIMO DE GERACAO ===")

for i in usinas:
print(
f"Usina {i} ({fontes[i]}): "
f"{producao[i].value():.2f} MWh"
)

geracao_total = sum(
```

```
producao[i].value()
for i in usinas
)

geracao_renovavel = sum(
producao[i].value()
for i in renovaveis
)

percentual_renovavel = (
100 * geracao_renovavel / geracao_total
)

custo_total = pulp.value(
problema_energia.objective
)

print("-" * 45)
print(f"Geracao total: {geracao_total:.2f} MWh")
print(
f"Geracao renovavel: "
f"{geracao_renovavel:.2f} MWh"
)
print(
f"Participacao renovavel: "
f"{percentual_renovavel:.2f}%"
)
print(f"Custo total: US$ {custo_total:,.2f}")
else:
print("Nao foi encontrada uma solucao otima.")
```

Algoritmo em linguagem Python 3: Algoritmo para o despacho das usinas com participação mínima de fontes renováveis.