CENTRO DE CIÊNCIAS APLICADAS E EDUCAÇÃO

UNIVERSIDADE FEDERAL DA PARAÍBA

Uma Aplicação em Java do Método Simplex

Wellerson Alex Brito da Silva

Rio Tinto, 2020

Wellerson Alex Brito da Silva

Uma Aplicação em Java do Método Simplex

Monografia apresentada ao curso de Licenciatura em Ciência da Computação do Centro de Ciências Aplicadas e Educação, da Universidade Federal da Paraíba, como requisito para a obtenção do grau de Licenciado em Ciência da Computação.

Orientador: Prof. Dr. José Laudelino de M. Neto

Agosto de 2020

Catalogação na publicação
Seção de Catalogação e Classificação

S586a Silva, Wellerson Alex Brito da.
Uma Aplicação em Java do Método Simplex / Wellerson Alex Brito da Silva. - Rio Tinto, 2020.
73 f. : il.

Orientação: José Laudelino de Menezes Neto.
Monografia (Graduação) - UFPB/CCAE.

1. Programação Linear, Método Simplex, Otimização. I. Neto, José Laudelino de Menezes. II. Título.

UFPB/BC

A minha mãe Maria Uberlandia, ao meu pai José Ronaldo, aos meus irmãos, minha esposa Rayssa e a toda minha família que, com muito carinho e apoio, não mediram esforços para que eu chegasse até esta etapa de minha vida.

AGRADECIMENTOS

A Deus, por ter permitido que eu tivesse saúde e determinação para não desanimar durante todos esses anos de estudo. Aos meus pais, irmãos e minha esposa, que me incentivaram nos momentos difíceis e por todo apoio e pela ajuda, que muito contribuíram para a realização deste trabalho. Aos professores, em especial ao meu orientador e todos que fazem parte da banca examinadora, por todos os conselhos, pela ajuda e pela paciência com a qual guiaram o meu aprendizado. Às pessoas com quem convivi ao longo desses anos de curso, que me incentivaram e que certamente tiveram impacto na minha formação acadêmica. À instituição de ensino UFPB CAMPUS-IV, essencial no meu processo de formação profissional, pela dedicação, e por tudo o que aprendi ao longo dos anos do curso.

RESUMO

A programação linear é uma técnica de resolução de problemas focada no uso de modelos matemáticos lineares, utiliza-se do método Simplex para resolver os problemas e obter a solução ótima. Nesta monografia é feito um estudo exploratório, bibliográfico, que tem por objetivo desenvolver uma aplicação, feita em Java, que executa os procedimentos do método Simplex para maximizar problemas de programação linear. O resultado deste estudo se provou de forma satisfatória, pois, com a aplicação dos procedimentos realizados de maneira computacional é possível obter soluções mais confiáveis em um período de tempo menor. A aplicação funciona com algumas limitações para a grande maioria dos problemas que são propostos para esse nível de aprofundamento no tema. Em síntese, reiteramos a importância de trabalhar os conceitos teóricos e práticos de programação linear que faz parte do leque de opções que existem dentro da matemática computacional, com isso, conclui-se que o estudo é importante para auxiliar alunos que demonstrem interesse em aprender técnicas de programação linear e estimular a realização de pesquisas sobre o tema.

Palavras-chave: Programação Linear. Método Simplex. Otimização.

ABSTRACT

Linear programming is a problem solving technique focused on the use of linear mathematical models, using the Simplex method to solve problems and obtain the optimal solution. In this monograph, an exploratory, bibliographic study is made, which aims to develop an application, made in Java, that performs the procedures of the Simplex method to maximize linear programming problems. The result of this study proved to be satisfactory, because with the application of procedures performed in a computational way it is possible to obtain more reliable solutions in a shorter period of time. The application works with some limitations for the vast majority of problems that are proposed for this level of depth in the topic. In summary, we reiterate the importance of working with the theoretical and practical concepts of linear programming that is part of the range of options that exist within computational mathematics, with this, it is concluded that the study is important to assist students who show interest in learning techniques linear programming and encourage research on the topic.

Key-words: Linear Programming. Simplex method. Optimization.

LISTA DE FIGURAS

Figura 1 – Fluxo de procedimentos do Método Simplex..... 26

Figura 2 – Janela para inserir o número de restrições..... 28

Figura 3 – Janela para inserir o número de restrições……..... 29

Figura 4 – Janela para informar a quantidade de colunas…..... 29

Figura 5 – Janela para informar a quantidade de colunas…..... 29

Figura 6 – Janela para inserir os valores da função Z……..... 30

Figura 7 – Janela para informar o termo independente de Z..... 30

Figura 8 – Janela para informar os valores da restrição 1…..... 30

Figura 9 – Janela para informar o termo independente para a restrição 1..... 30

Figura 10 – Resultado retornado pelo programa após chegar na solução ótima.... 31

Figura 11 – Janela de erro, quando o programa entra em Loop..... 33

Figura 12 – Retorno de problema que gera Loop……..... 33

Figura 13 – Janela de erro, quando o problema é ilimitado…..... 34

Figura 14 – Retorno de um problema ilimitado……..... 34

LISTA DE TABELAS

Tabela 1 – Explicação teórica da Forma de Tabela…..... 19

Tabela 2 – Restrições/custos do exemplo……..... 20

Tabela 3 – Inicial……..... 22

Tabela 4 – Fazendo os cálculos da iteração 1……..... 23

Tabela 5 – Resultado da iteração 1……..…..... 23

Tabela 6 – Fazendo os cálculos da iteração 2……..... 24

Tabela 7 – Resultado da iteração 2……..…..... 24

Tabela 8 – Resultado final do método……..…..... 25

Tabela 9 – Exemplo de solução ilimitada……..…..... 27

Tabela 10 – Exemplo de Loop Tableau……..... 32

Tabela 11 – Exemplo de problema ilimitado……..…..... 34

LISTA DE ABREVIATURAS

PL – PROGRAMAÇÃO LINEAR

MSFT – MÉTODO SIMPLEX NA FORMA DE TABELA

SUMÁRIO

1. Introdução.....12

2. Metodologia 15

2.1 Implementações do Método Simplex.....1**5**

3. Teoria do Método Simplex.....17

3.1 Resolução de um Problema Via Método Simplex 19

3.2 Método Simplex na Forma de Tabela.....22

3.3 Constatando o Caso do Problema Ilimitado 26

4. Resultados 28

4.1 Implementação de um Programa que Executa o Método Simplex 28

4.2 Comentários Gerais Sobre o Programa 32

5. Conclusões e trabalhos futuros.....35

6. Referências 37

APÊNDICE - Código Fonte 39

1. Introdução

A Programação Linear (PL) é utilizada para resolver problemas de otimização, muitos problemas que existem no mundo real como: O Problema das Ligas Metálicas, O Problema da Fábrica de Móveis, O Problema do Atleta Indeciso, O Problema de uma Pequena Manufatura, O Problema da Produção de Camisetas, O Problema da Dieta, entre outros, podem ser formulados como problemas de programação linear.

A PL é o processo de minimizar ou maximizar uma função objetivo linear que é o termo para obter a solução ótima e a uma série de restrições lineares que são as variáveis utilizadas no processo, para encontrar a solução. O algoritmo Simplex é o método mais utilizado para a resolução de problemas de programação linear (PLOSKAS; SAMARAS, 2015).

O Método Simplex para programação linear foi criado por George Dantzig em 1947. O sucesso do algoritmo levou a uma vasta gama de especializações e generalizações que têm dominado a pesquisa de operações práticas desde então (DANTZIG; THAPA, 2003), (NEZAMABADI; VAHIDINASAB, 2015), (NASH, 2000), (SOUSA, 2005).

A programação linear é utilizada na solução de problemas como uma técnica de tomada de decisões. Desenvolvida inicialmente para necessidades militares em alocar recursos de maneira efetiva, segundo uma pesquisa realizada em Londres a programação linear é uma das ferramentas mais utilizada para tomada de decisões além do modelo de regressão.

O desenvolvimento da programação linear tem sido classificado entre os mais importantes avanços científicos dos meados do século XX. Seu impacto desde 1950 tem sido extraordinário. Hoje é uma ferramenta padrão que poupou milhares de dólares para muitas empresas ou até mesmo negócios de porte médio em diversos países industrializados ao redor do mundo (HILLIER; LIEBERMAN, 2013).

Os gerentes das organizações, constantemente se deparam, com circunstâncias em que devem tomar decisões, que levam em consideração diversas alternativas conflitantes e concorrentes, diante dessas situações os gerentes podem utilizar a intuição gerencial ou realizar um processo de modelagem da situação (LACHTERMACHER, 2007).

O motivacional para o desenvolvimento deste trabalho foi, desenvolver uma ferramenta computacional para auxiliar na resolução e na correção de problemas de programação linear utilizando o Método Simplex na Forma de Tabela (MSFT) para maximizar uma função objetivo linear.

A Programação Linear possui uma área muito desafiadora que é a matemática computacional, devido a isso foi gerado interesse por parte do autor para a elaboração de um projeto de pesquisa no tema. Sendo importante ressaltar o vínculo pessoal e a sua identificação com as disciplinas de cálculo que são ofertadas no curso de Licenciatura em Ciência da Computação da Universidade Federal da Paraíba e participação em projetos que abordaram o tema.

A solução para os problemas de maximização de uma função objetivo linear utilizando o MSFT, é importante para a área de Programação Linear. A pesquisa promove a difusão do tema, estabelecendo os critérios básicos para a resolução e correção desses problemas. Como o trabalho de pesquisa trata do tema Programação Linear, acreditamos que possa incentivar futuros alunos a se interessarem por explorar o tema.

Neste estudo iremos desenvolver uma aplicação para resolver problemas de otimização priorizando a maximização de funções utilizando o MSFT. Criar um programa utilizando a linguagem Java para resolver e corrigir exercícios de Programação Linear; Explicar a grande demanda de atividades que são realizadas para resolver um exercício de PL para maximizar uma função objetivo linear de forma manual utilizando o método Simplex; e Apontar a importância da utilização da computação para executar as tarefas no processo de otimização para chegar a uma solução ótima.

Em linhas gerais, a programação linear busca, entre as inúmeras tarefas ou atividades, descobrir a melhor distribuição dos recursos a fim de obter um valor ótimo do objetivo desejado. Os problemas de alocação de recursos distinguem-se pela existência de um objetivo explícito por meio de variáveis de decisão e pela existência de restrições para alocar os recursos devido às quantidades disponíveis e a forma de aplicá-los (ANDRADE, 2007).

Este trabalho está organizado em seis seções. A Seção dois apresenta-se a metodologia utilizada, que foi a pesquisa bibliográfica, contendo todo o fluxo da pesquisa e métodos que se construíram para o desenvolvimento do trabalho. Na Seção três, a Teoria do Método Simplex. Na Seção quatro são relatados os resultados deste estudo, mostrando a resolução de um problema utilizando a aplicação criada. Na quinta Seção, apresenta-se a conclusão do trabalho e sugestões para trabalhos futuros. Na sexta Seção são apresentadas as referências utilizadas para o desenvolvimento do trabalho. Por fim, é apresentado no Apêndice o código fonte do programa.

2. Metodologia

O objetivo deste trabalho foi desenvolver uma aplicação para auxiliar e resolver problemas de otimização e verificar se uma resolução está correta, priorizando a maximização de funções utilizando o Método Simplex na Forma de Tabela, inicialmente houve um estudo aprofundado de programação linear e de elaboração de modelos matemáticos, para a resolução de problemas, objeto deste estudo. Obtido o modelo, a próxima etapa foi a sua solução, exposto nesses procedimentos metodológicos.

A pesquisa foi iniciada a partir de leituras realizadas pelo autor com o intuito de se familiarizar com a realidade que a técnica de Programação Linear é abordada nos cursos superiores de Computação e áreas afins. Foram adotados parâmetros para a pesquisa de artigos acadêmicos, realizando o processo de filtragem dos artigos pelas palavras chaves como: “Programação Linear”, “Método Simplex”, “Otimização”, “Maximizar”. A revisão bibliográfica foi construída com base em artigos acadêmicos, dissertações, teses e livros, todos pesquisados pelo Google Acadêmico, e definidos através de leituras feitas pelo próprio autor deste trabalho.

2.1. Implementações do Método Simplex

Neste trabalho foi realizada uma pesquisa sobre implementações simplex já feitas para melhor embasar o desenvolvimento de uma aplicação do Método Simplex na Forma de Tabela. Após ser feito um levantamento foram selecionadas duas aplicações que serão apresentadas a seguir.

A primeira aplicação analisada foi a **Gurobi Optimizer** que é um solucionador de programação matemática que resolve problemas de Programação Linear. É utilizado por várias empresas para obter melhor desempenho, desenvolvimento mais rápido e melhor suporte. Esta aplicação oferece licença anual para algumas de suas ferramentas desde que a instituição seja reconhecida e a licença renovada anualmente, porém existem obstáculos por ser na língua inglesa e pela difícil interação da ferramenta com outras linguagens.

A segunda aplicação estudada foi o grupo **GNU Linear Programming Kit** que é uma biblioteca de rotinas que utilizam algoritmos de pesquisas de operações bem conhecidas para solucionar problemas, o solver simplex integrado as linguagens C e C++, o que gera uma certa dificuldade já que na Paraíba, em especial no Campus IV da UFPB, Java é a linguagem adotada. O solver da CPLEX se torna caro para as instituições de ensino.

3. Teoria do Método Simplex

Os problemas que são considerados simples ou corretamente elaborados de Programação Linear podem ser resolvidos utilizando o Método Simplex, algoritmo introduzido por George Bernard Dantzig. O algoritmo Simplex utiliza uma grande quantidade de cálculos, portanto, nos primeiros períodos de uso, sua resolução era realizada unicamente de forma manual.

Com a chegada do computador, em 1951, a Programação Linear encontrou seu adepto natural e foi se ampliando de uma maneira magnífica. Segundo Murty (1983) métodos clássicos de cálculo ou álgebra não são suficientemente robustos para a resolução de problemas de Programação Linear. Goldbarg e Luna (2000) explicam que o algoritmo simplex soluciona problemas de equações lineares através de uma sequência de passos, otimizando uma função objetivo. Além disso, Cormen (1999) define o simplex como o algoritmo que examina uma sequência de pontos dentro de uma região viável e encontra a solução ótima.

Dada uma função objetivo linear Z e um conjunto de restrições lineares, o método Simplex é uma técnica iterativa que possibilita otimizar a solução da função objetivo em cada etapa. O processo termina quando não é possível continuar otimizando este valor, ou seja, quando se obtém a solução ótima . Com base no valor da função objetivo, em um ponto qualquer, o procedimento consiste em procurar outro ponto que melhore o valor anterior.

A forma padrão do modelo de problema consiste em uma função objetivo linear, sujeita a certos critérios (as restrições lineares):

$$\textit{Função objetivo } Z: \qquad c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

$$\begin{aligned} \textit{sujeito às restrições}: \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \leq b_2 \\ & \cdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \leq b_m \\ & x_1, \ldots, x_n \geq 0 \end{aligned}$$

O modelo deve atender às seguintes condições: O objetivo é maximizar ou minimizar o valor da função objetivo linear Z (por exemplo, aumentar lucros ou reduzir custos) sujeito às restrições lineares impostas.

Restringimos nosso estudo a problemas onde todas as restrições lineares devem conter apenas o sinal de menor ou igual ($\leq$) antes do termo independente e as variáveis (x_i) devem ser positivas ou nulas (condição de não-negatividade) e os termos independentes (b_i) não podem ser negativos, pois existem problemas com outras configurações que podem ser encontrados no livro: Chvatal V.-Linear programming-Freeman (1983). Além disso, iremos priorizar apenas problemas de maximização da função objetivo linear, isso ocorreu devido o nível de aprofundamento e dominância adquirido pelo autor com o tema durante a pesquisa.

Para resolver o problema via método Simplex, adicionamos m variáveis de folga $x_{n+1}, x_{n+2}, \ldots, x_{n+m} \geq 0$ para que seja possível resolver o problema utilizando o Método Simplex na Forma de Tabela e transformamos nosso problema na forma:

$$\textit{Função objetivo } Z: \qquad c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

$$\begin{aligned} \textit{sujeito às restrições}: \quad & a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n + x_{1n+m} = b_1 \\ & a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n + x_{2n+m} = b_2 \\ & \cdots \\ & a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n + x_{mn+m} = b_m \\ & x_1, x_2, \ldots, x_n, x_{n+1}, x_{n+2}, x_{n+m} \geq 0 \end{aligned}$$

Este problema é representado pela Tabela 1, abaixo:

Tabela 1: Explicação teórica da Forma de Tabela.

	r								s
	X_1	X_2		X_n	X_{n+1}	X_{n+2}		X_{n+m}	
Linha 1	a_{11}	a_{12}	**...**	a_{1n}	**1**	**0**	**…**	**0**	b_1
Linha 2	a_{21}	a_{22}	**...**	a_{2n}	**0**	**1**	**...**	**0**	b_2
. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
Linha m	a_{m1}	a_{m2}	**...**	a_{mn}	**0**	**0**	**...**	**1**	b_m
Z	c_1	c_2	**...**	c_m	**0**	**0**	**...**	**0**	**0**

A partir desta tabela, otimizamos o problema utilizando o procedimento explicado na próxima seção.

3.1. Resolução de um Problema Via Método Simplex

De forma a elucidar o passo a passo de uma solução para uma problema de PL, selecionamos o exemplo: O Problema da Fábrica de Móveis, dado por (GOLDBARG, M. C.; LUNA, H. P. L.; GOLDBARG, E. F. G. - 1. ed, 2015).

O método Simplex é um algoritmo extremamente eficiente, porém, quando sua aplicação é realizada manualmente se torna de certa forma exaustivamente trabalhoso e cansativo, pois, é necessário realizar diversas iterações com as tarefas de manipulações de variáveis e realização de contas matemáticas.

Iremos explicar este método resolvendo um problema, a seguir é apresentado o passo a passo do MSFT aplicado no Exemplo: O problema da Fábrica de Móveis.

Uma fábrica de móveis dispõe em estoque 250m de tábuas, 600m de pranchas e 500m de painéis de conglomerado. A fábrica oferece uma linha de móveis composta por um modelo de escrivaninha, uma mesa de reunião, um

armário e uma prateleira. Cada tipo de móvel consome quantidades de matéria-prima que seguem as condições descritas na Tabela 2.

A escrivaninha é vendida por R$ 100, a mesa por R$ 80, o armário por R$ 120 e a prateleira por R$ 20. Pede-se exibir um Modelo de Programação Linear que maximize a receita com a venda dos móveis.

Tabela 2: Restrições/custos do exemplo.

Tipo de Insumo componente do móvel	Quantidade de material em metros consumidos por unidade de produto				Disponibilidade do Recurso (metros)
	Escrivaninha	Mesa	Armário	Prateleira	
Tábua	1	1	1	4	250
Prancha	0	1	1	2	600
Painéis	3	2	4	0	500
Valor de Revenda (R$)	100	80	120	20	

Análise do Modelo: o pedido do problema é programar o quantitativo de produção de quatro diferentes tipos de móveis. O objetivo é maximizar a receita. A quantidade dos insumos de cada móvel é limitada. Uma restrição de integridade é introduzida implicitamente.

Modelo de solução

O objetivo é maximizar a receita de uma produção cujos quantitativos relativos são passíveis de controle direto. Encontrar o maior número de quantitativos (ou variáveis de planejamento), os valores de x_i devem ser inteiros, já que os produtos são indivisíveis.

Escolha da variável de decisão

$x_i \equiv$ quantidade em unidades a ser produzida do produto escrivaninha ($i = 1$), mesa ($i = 2$), armário ($i = 3$), prateleira ($i = 4$).

Com as variáveis de decisão escolhidas, devemos expressar a função objetivo como uma função dessas variáveis:

Elaboração da função objetivo

$$Maximizar\ Z = 100x_1 + 80x_2 + 120x_3 + 20x_4$$

Receita bruta em reais em função do número de unidades produzidas de cada tipo de móvel.

Formulação das restrições tecnológicas

a. Restrição associada à disponibilidade de tábuas:

$$x_1 + x_2 + x_3 + 4x_4 \leq 250$$

b. Restrição associada à disponibilidade de pranchas:

$$x_2 + x_3 + 2x_4 \leq 600$$

c. Restrição associada à disponibilidade de painéis:

$$3x_1 + 2x_2 + 4x_3 \leq 500$$

d. Restrições de não negatividade e integralidade

$\{x_1, x_2, x_3, x_4\} \in R$ (conjunto dos números reais)

Problema na forma inicial:

$$\begin{aligned} &maximizar \quad z = 100x_1 + 80x_2 + 120x_3 + 20x_4 \\ &restrito\ a \quad x_1 + x_2 + x_3 + 4x_4 \leq 250 \\ &\qquad\qquad x_2 + x_3 + 2x_4 \leq 600 \end{aligned}$$

$$3x_1 + 2x_2 + 4x_3 \leq 500$$

$$x_1, x_2, x_3, x_4 \geq 0$$

Adicionando variáveis de folga:

No Método Simplex, adicionamos uma variável de folga em cada linha de restrição para transformar as desigualdades em igualdades. Como no exemplo em questão temos três linhas de restrição, vamos adicionar três variáveis de folga x_5, x_6 e x_7, e nosso problema fica com a seguinte configuração.

$$\textit{maximizar} \quad z = 100x_1 + 80x_2 + 120x_3 + 20x_4$$

$$\begin{aligned} \textit{restrito a} \quad & x_1 + x_2 + x_3 + 4x_4 + x_5 = 250 \\ & x_2 + x_3 + 2x_4 + x_6 = 600 \\ & 3x_1 + 2x_2 + 4x_3 + x_7 = 500 \end{aligned}$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0$$

3.2. Método Simplex na Forma de Tabela

Tabela 3: inicial.

	r							**s**
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
Linha 1	**1**	**1**	**1**	**4**	**1**	**0**	**0**	**250**
Linha 2	**0**	**1**	**1**	**2**	**0**	**1**	**0**	**600**
Linha 3	**3**	**2**	**4**	**0**	**0**	**0**	**1**	**500**
Z	**100**	**80**	**120**	**20**	**0**	**0**	**0**	**0**

Iteração 1

1. Escolhemos na linha Z a coluna que tem o maior valor positivo, neste caso será selecionada a coluna X_3 que possui o valor 120.

2. Ao escolhermos a coluna, observamos os valores positivos e escolhemos a linha tal que **s/r** é o menor valor. Fazendo:

{250/1 = 250, 600/1 = 600, 500/4 = 125}, logo devemos escolher a linha X_7 que possui o menor quociente **s/r.**

3. Zeramos o que está acima e abaixo da intersecção linha por coluna.

4. Fazendo os cálculos para zerar a linha X_5 **,** multiplicamos $(X_7 \times (-1/4)) - X_5$, para linha X_6 **,** fazemos a multiplicação dessa form $(X_7 \times (-1/4)) - X_6$ e, por fim na linha **Z** é feito $(X_7 \times (-30)) - Z$.

Tabela 4: Fazendo os cálculos da iteração 1.

	r							s
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
Linha 1	**1**	**1**	**1**	**4**	**1**	**0**	**0**	**250**
Linha 2	**0**	**1**	**1**	**2**	**0**	**1**	**0**	**600**
Linha 3	**3**	**2**	**4**	**0**	**0**	**0**	**1**	**500**
Z	**100**	**80**	**120**	**20**	**0**	**0**	**0**	**0**

Tabela 5: Resultante da Iteração 1.

	r							s
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
Linha 1	**1/4**	**1/2**	**0**	**4**	**1**	**0**	**-1/4**	**125**
Linha 2	**-3/4**	**1/2**	**0**	**2**	**0**	**1**	**-1/4**	**475**
Linha 3	**3**	**2**	**4**	**0**	**0**	**0**	**1**	**500**
Z	**10**	**20**	**0**	**20**	**0**	**0**	**-30**	**-15000**

Iteração 2

1. Neste caso específico acontece um empate entre as colunas X_2 e X_4 então optamos por escolher a coluna referente a variável de menor índice. No caso, escolhemos a coluna X_2.

2. Ao escolhermos a coluna, observamos os valores positivos e escolhemos a linha tal que **s/r** é o menor valor. Fazendo:

{125/(½) = 250, 475/(½) = 950, 500/2 = 250}, percebemos que existe um empate. Neste caso de empate entre linhas, escolhemos a de menor índice, no caso, a Linha 1.

3. Fazendo os cálculos para zerar a linha X_6, multiplicamos $(X_5 \times (-1)) - X_6$, para linha X_7, fazemos a multiplicação dessa forma $(X_5 \times (-4)) - X_7$ e, por fim na linha **Z** é feito $(X_5 \times (-0)) - Z$.

Tabela 6: Fazendo os cálculos da iteração 2.

	r							**s**
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
Linha 1	**1/4**	**1/2**	**0**	**4**	**1**	**0**	**-1/4**	**125**
Linha 2	**-3/4**	**1/2**	**0**	**2**	**0**	**1**	**-1/4**	**475**
Linha 3	**3**	**2**	**4**	**0**	**0**	**0**	**1**	**500**
Z	**10**	**20**	**0**	**20**	**0**	**0**	**-30**	**-15000**

Tabela 7: Resultante da Iteração 2.

	r							s
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
Linha 1	**1/4**	**1/2**	**0**	**4**	**1**	**0**	**-1/4**	**125**
Linha 2	**-1**	**0**	**0**	**-2**	**-1**	**1**	**0**	**350**
Linha 3	**2**	**0**	**4**	**-8**	**-4**	**0**	**2**	**0**
Z	**0**	**0**	**0**	**-140**	**-40**	**0**	**-20**	**-20000**

Chegamos no ótimo, pois todos os valores na linha **Z** são menores ou iguais a zero. Para demonstrar a solução ótima temos que fazer com que todas as colunas X_i que tem o valor menor que zero na linha **Z** e as demais linhas sejam zero e todas as que já possuem o valor igual a zero continuam com seus respectivos valores. Como é visto na Tabela 8:

Tabela 8: Representação final do método.

	r							s
	X_1	X_2	X_3	X_4	X_5	X_6	X_7	
Linha 1	**0**	**1/2**	**0**	**0**	**0**	**0**	**0**	**125**
Linha 2	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**350**
Linha 3	**0**	**0**	**4**	**0**	**0**	**0**	**0**	**0**
Z	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**-20000**

Com isso, podemos entrar a solução ótima que é dada por:

$$Z = 20000$$

$$\tfrac{1}{2}x_2 = 125 \Rightarrow x_2 = 125/\tfrac{1}{2} = 250$$

$$x_6 = 350$$

$$4x_4 = 0 \Rightarrow x_4 = 0$$

Com os valores obtidos pela realização dos procedimentos do MSFT,

utilizando as técnicas de resolução para maximizar, chegamos a conclusão que o mais vantajoso, para se obter mais lucro, é produzir o produto relacionado a $X_2 =$ mesa. Como $X_2 = 250,$ significa que produzir 250 mesas é a forma de ter um melhor retorno financeiro, pois a função objetivo **Z** está vinculada ao lucro. Na Figura 1, podemos ver o fluxo que é processado pelo Método Simplex.

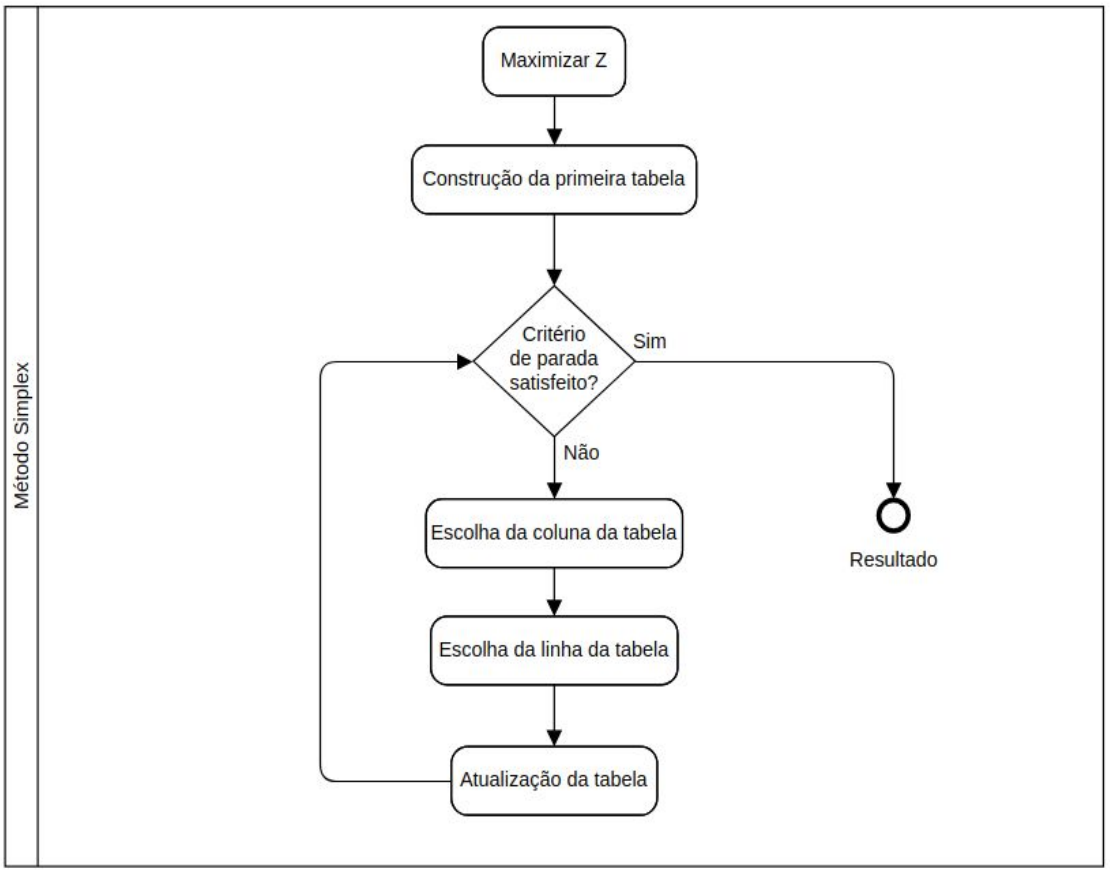

Figura 1: Imagem do autor (criado na ferramenta de modelagem Visual Paradigm Online).

3.3. Constatando o Caso do Problema Ilimitado

A solução ótima é encontrada quando o critério de parada é satisfeito e não existam variáveis com valor positivo, ou seja, quando todos os elementos da linha Z são menores ou iguais a zero, isso significa que a otimização foi alcançada. O valor Z atual é a solução ótima do problema, como é visto na Tabela 7.

Existe também o caso da solução ilimitada que ocorre se toda coluna da variável que escolhemos tiver seus elementos negativos ou nulos como é mostrado na Tabela 9, trata-se de um problema não-limitado. Ou seja, não há valor ótimo concreto para a função objetivo, mas à medida que os valores das variáveis são aumentados, o valor Z também aumenta sem violar qualquer restrição, pode aumentar indefinidamente, o máximo é um valor infinito.

Tabela 9: Exemplo de solução ilimitada.

	r						**s**
	X_1	X_2	X_3	X_4	X_5	X_6	
Linha 1	**2**	**-11**	**-8**	**-43**	**6**	**47**	**0**
Linha 2	**1**	**13**	**-1**	**-51**	**0**	**55**	**0**
Z	**2**	**-2**	**98**	**31**	**0**	**-55**	**0**

Em alguns casos o problema pode ficar em Loop, o que iremos explicar mais detalhadamente com um exemplo no decorrer da próxima seção.

4. Resultados

De forma a analisar o resultado do passo a passo de uma solução para um problema de PL, selecionamos o mesmo exemplo utilizado na seção 3.1: O Problema da Fábrica de Móveis.

4.1. Implementação de um Programa que Executa o Método Simplex

Notamos que um problema de programação linear resolvido pelo método Simplex para maximizar uma função objetivo linear Z demanda uma grande realização de tarefas, desde a criação da tabela inicial e passando pela escolha da coluna e linha até as aplicações de escalonamento e atualização da tabela para poder checar se a solução ótima foi encontrada e caso não tenhamos atingido o ótimo fazemos todo o processo novamente de forma recursiva.

Devido essa gama de atividades que precisam serem desenvolvidas para se obter a otimização de um problema de maximização, foi criado neste estudo um simples programa para resolver problemas de programação linear onde o objetivo é maximizar uma função objetivo linear Z. O programa, na versão inicial está limitado para resolver problemas com apenas 5 linhas de restrições lineares no máximo, pois a maioria dos problemas não ultrapassa essa quantidade de restrições e esse número pode aumentar em futuras atualizações da aplicação.

Para descrever e explicar o funcionamento do programa serão utilizadas telas da aplicação executando o exemplo da fábrica de móveis. Ao iniciar o programa é aberto uma janela de inserção de dados com a mensagem “Informe o número de restrições do problema” e com uma dica dos possíveis valores para evitar erros nos dados de entrada, como vemos na Figura 2 abaixo.

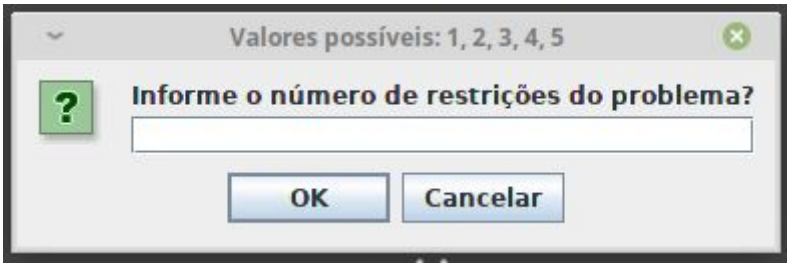

Figura 2: Janela para inserir o número de restrições.

Iremos resolver o problema da Fábrica de Móveis de novo, agora utilizando o programa. Resolvendo, temos que o número de restrições é igual a 3, digitando a entrada como é visto na Figura 3 e depois OK é aberto a próxima janela para informar o número de colunas que são as variáveis principais, incluindo as variáveis de folga e os termos independentes, como é mostrado na Figura 4.

Figura 3:Janela para inserir o número de restrições.

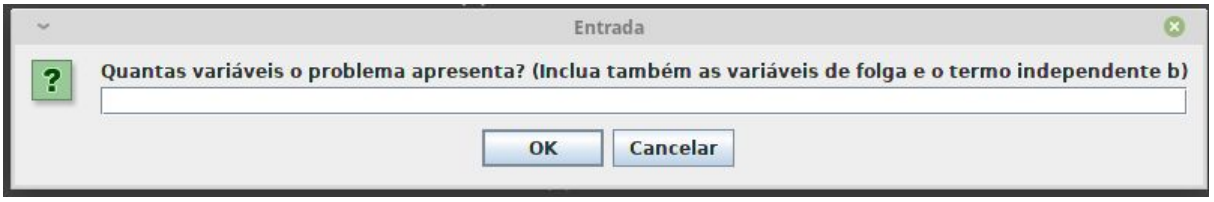

Figura 4: Janela para informar a quantidade de colunas.

Fazendo a contagem de todas as variáveis principais, e também as variáveis de folga e o termo independente temos um total de 8 colunas que por sua vez vai ser o número informado nesta etapa, na Figura 5 inserimos o valor.

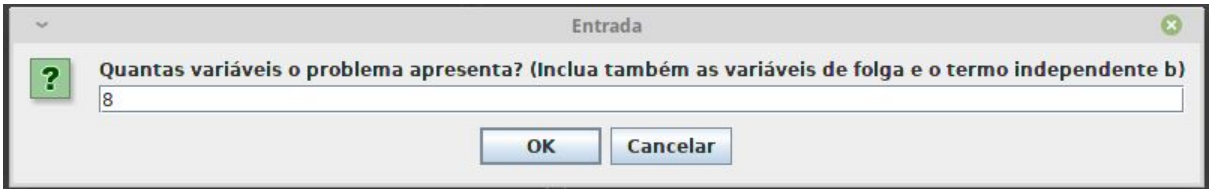

Figura 5: Janela para informar a quantidade de colunas.

Após inserir o número de variáveis e clicar em OK o programa abrirá uma nova janela para que sejam informados os valores das variáveis X_1 até X_7 da função objetivo Z como podemos observar na Figura 6, os valores são informados um de cada vez. Quando chegar o momento de inserir o termo independente aparecerá uma mensagem diferente na janela de entrada como podemos ver na Figura 7.

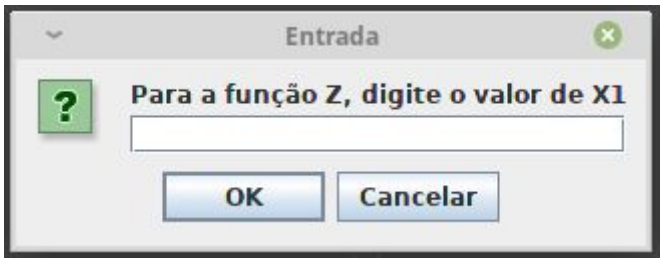

Figura 6: Janela para inserir os valores da função Z.

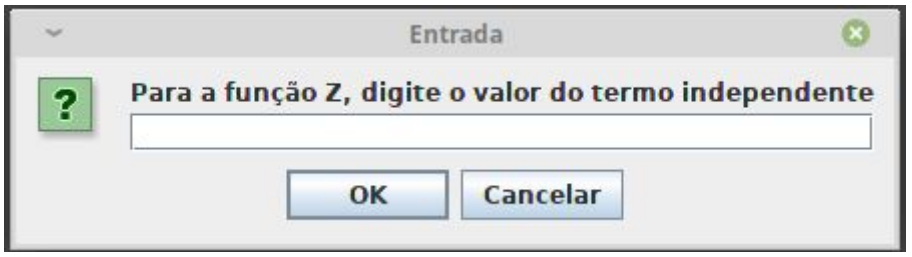

Figura 7: Janela para informar o termo independente de Z.

Feito isso passamos para o momento de informar os valores das restrições que segue o mesmo modelo de entrada para a função Z. São inseridos todos os valores de X_1 até X_7 para a restrição 1, como pode ser visto na Figura 8 e por fim o termo independente b_1 deve ser informado, conforme mostra a Figura 9.

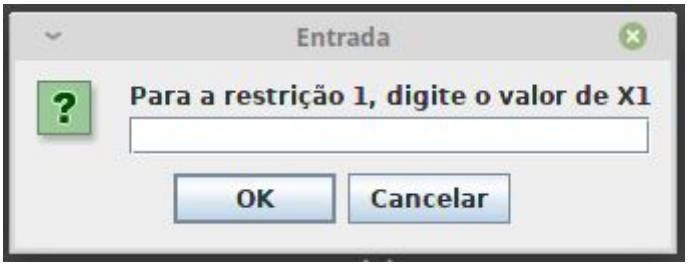

Figura 8: Janela para informar os valores da restrição 1.

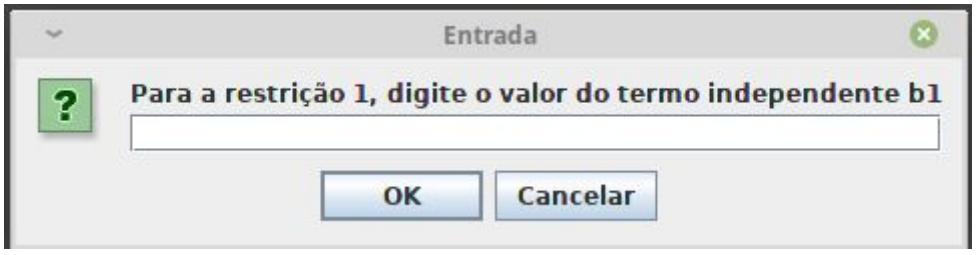

Figura 9: Janela para informar o termo independente da restrição 1.

Para as restrições 2 e 3 é seguido o mesmo modelo de entrada de dados com uma pequena alteração apenas nas mensagens alterando o número da restrição para sua respectiva ordem, por exemplo: para a segunda restrição será

“Para a restrição 2, …” e no termo independente ficará “..., digite o valor do termo independente b_2” e da mesma forma para a terceira restrição, alterando os valores para 3.

O processo de inserção de dados sendo concluído após as etapas anteriores, vem em seguida a execução das fases de resolução do método Simplex e caso não ocorra o caso de um Loop Tableau o resultado será impresso no Console da ferramenta Eclipse. É mostrado a tabela inicial, coluna e linha escolhida, os valores restritivos para cada linha, a tabela resultante para cada iteração e quando a condição de parada é satisfeita é mostrado a solução ótima, como podemos analisar na Figura 10.

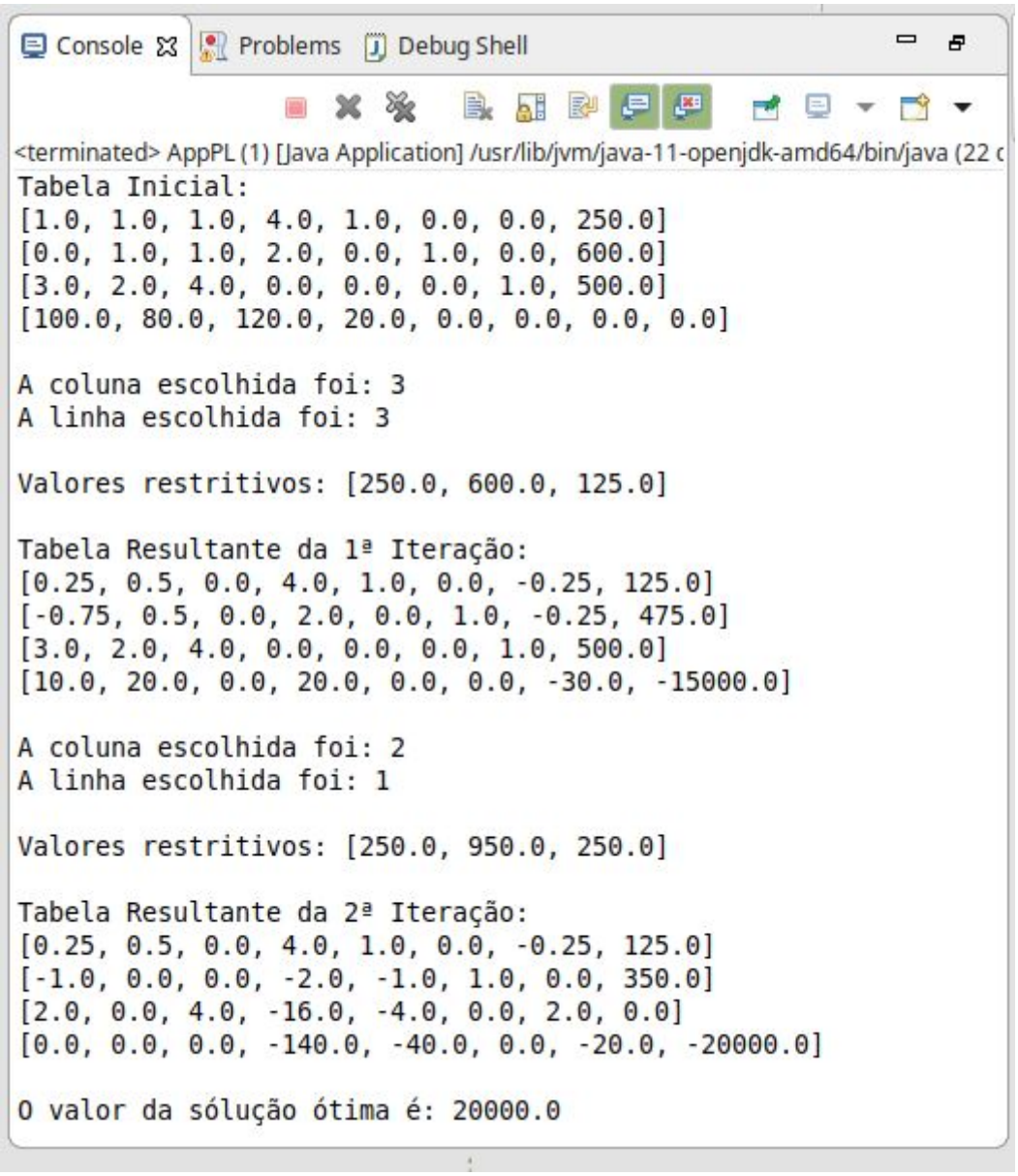

Figura 10: Resultado retornado pelo programa após chegar na solução ótima.

O programa é uma ferramenta facilitadora para correção dos problemas de maximização da área de Programação Linear, pois demonstra detalhadamente todo o processo de criação das tabelas e suas atualizações no decorrer de cada iteração e também fazendo os cálculos de maneira computacional o que diminui consideravelmente as chances de erro.

O código fonte do programa pode ser encontrado no GitHub através do Link: https://github.com/WellersonAlex/ProjetoTCC e também no Apêndice deste trabalho, está disponível para ser usado como auxiliar nas tarefas de maximizar uma função objetivo Z que possua o número de restrições entre 1 e 5, e é de extrema importância que o usuário tome bastante cuidado com a inserção de dados, pois se for informado um valor incorreto, obviamente o programa irá retornar uma resposta diferente da esperada.

4.2. Comentários Gerais Sobre o Programa

Podemos ainda nos deparar com problemas que são ilimitados, onde a função objetivo Z teria valores que crescem infinitamente e ainda casos onde o problema entra em ciclo, ou seja, em um Loop. Em caso de Loop, temos como critério de parada, a comparação da tabela inicial com a tabela resultante de cada iteração, se encontrarmos os mesmos valores nas duas tabelas significa que o problema gerou um Loop, chamado de Loop Tableau.

Como demonstração desse caso (Loop), iremos utilizar um exemplo do livro: Chvatal V.-Linear programming-Freeman (1983), como é visto na Tabela 10 abaixo:

Tabela 10: Exemplo de Loop Tableau.

	X_1	X_2	X_3	X_4	X_5	X_6	
Linha 1	**-2**	**-9**	**1**	**9**	**1**	**0**	**0**
Linha 2	**1/3**	**1**	**-1/3**	**-2**	**0**	**1**	**0**
Z	**2**	**3**	**-1**	**-12**	**0**	**0**	**0**

Após todo o processo de inserção dos valores de entrada, no programa foram inseridos os valores ⅓ e -⅓ em forma de fração, pois, o algoritmo reconhece os termos e faz a divisão internamente dos números fracionários. Com o problema formulado, colocamos o programa para executar, na sequência aparece a janela com a mensagem informando que “O problema entrou em loop, não foi possível resolver”, como é visto na Figura 11.

Em seguida na Figura 12 são mostrados os quadros onde os valores aparecem repetidos em diferentes iterações realizados até que o critério de parada é estabelecido, observamos que os valores são iguais na tabela inicial e na iteração 6, por isso o Loop é gerado.

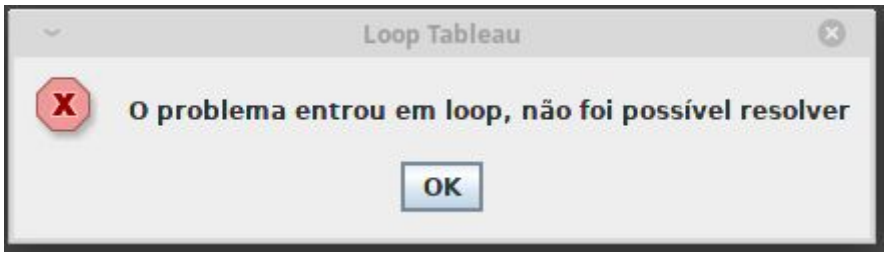

Figura 11: Janela de erro, quando o programa entra em Loop.

```
Os valores são iguais nas duas tabelas, por isso o Loop é gerado

Tabela Inicial:
[-2.0, -9.0, 1.0, 9.0, 1.0, 0.0, 0.0]
[0.3, 1.0, -0.3, -2.0, 0.0, 1.0, 0.0]
[2.0, 3.0, -1.0, -12.0, 0.0, 0.0, 0.0]


Tabela Resultante da 6ª Iteração:
[-2.0, -9.0, 1.0, 9.0, 1.0, 0.0, 0.0]
[0.3, 1.0, -0.3, -2.0, 0.0, 1.0, 0.0]
[2.0, 3.0, -1.0, -12.0, 0.0, 0.0, 0.0]
```

Figura 12: Retorno de problema que gera Loop.

Quando nos deparamos com um problema ilimitado, fazemos o seguinte critério de parada: ao escolher a coluna, se todos os valores de r forem ≤ 0

(menor ou igual que 0), significa que o problema é ilimitado. As iterações são encerradas pois não existe linha válida para ser escolhida.

Utilizamos um exemplo do livro: Chvatal V.-Linear programming-Freeman (1983) de um problema que é ilimitado, como é visto na Tabela 11 abaixo:

Tabela 11: Exemplo de problema ilimitado.

	X_1	X_2	X_3	X_4	X_5	
Linha 1	**1**	**-1**	**1**	**0**	**0**	**-1**
Linha 2	**-1**	**-1**	**0**	**1**	**0**	**-3**
Linha 3	**2**	**-1**	**0**	**0**	**1**	**2**
Z	**3**	**1**	**0**	**0**	**0**	**0**

Após inserirmos os valores e executar o programa, é retornado que o problema é ilimitado como podemos ver na Figura 13 que mostra através de uma janela a mensagem “O problema é ilimitado” e na Figura 14 mostra a última iteração realizada e a escolha da coluna 2, observamos que nesta coluna os valores das restrições são negativos, logo não teria nenhuma linha para ser escolhida.

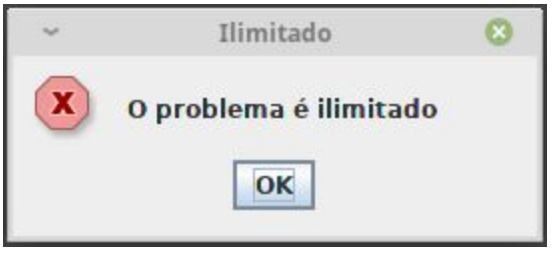

Figura 13: Janela de erro, quando o problema é ilimitado.

```
Tabela Resultante da 1ª Iteração:
[0.0, -0.5, 1.0, 0.0, -0.5, -2.0]
[0.0, -1.5, 0.0, 1.0, 0.5, -2.0]
[2.0, -1.0, 0.0, 0.0, 1.0, 2.0]
[0.0, 2.5, 0.0, 0.0, -1.5, -3.0]

A coluna escolhida foi: 2
```

Figura 14: Retorno de um problema ilimitado.

5. Conclusões e trabalhos futuros

O trabalho desenvolvido apresenta uma importante aplicação da Programação Linear que conduzem suas teorias a otimização de modelos matemáticos, respeitando as restrições, é possível encontrar soluções ótimas, seguindo os passos e as regras de aplicação do Método Simplex. A PL permite, ainda, uma infinidade de análises, dependendo apenas da forma de que as variáveis são controladas, podemos assim ter um melhor retorno e produzir ao máximo com os recursos disponíveis.

Existem matérias para o ensino de Programação Linear que são em inglês e por muitas vezes acompanhados de tutoriais de difícil compreensão para os iniciantes em programação e até mesmo para iniciados na área. A importância didática deste trabalho é por estar em português, o que facilita no entendimento teórico do assunto e a aplicação que serve de auxílio durante o processo de aprendizagem.

Assim, o objetivo principal deste trabalho foi desenvolver uma aplicação para solucionar questões de Programação Linear para buscar valores ótimos no contexto de maximização. O resultado pode ser considerado bom, pois de maneira geral a aplicação funciona corretamente para o que lhe foi sugerida, de forma razoável atende aos critérios de resolução de problemas com até 5 (cinco) linhas de restrições.

Este estudo, abre espaço para o auxílio à Programação Linear e pode ser referência para novas pesquisas nessa área. Como ainda está sendo implementado e revisado, os resultados reais do uso da aplicação que utiliza os procedimentos do Método Simplex ainda limitados, porém os resultados nos primeiros testes já indicam um grande avanço na resolução de questões bem elaboradas, o que mostra que o método aplicado pode ser uma boa ferramenta de auxílio na resolução e correção de tarefas e modelos de PL.

Como recomendação para trabalhos futuros, seria implementar, para os problemas que entram em Loop, uma forma de fugir do Loop e encontrar uma

solução viável. Aplicar a forma para resolver problemas de minimização. Expandir a quantidade de restrições, além de 5 linhas. Solucionar problemas onde os termos independentes b_i podem ser negativos. A restrição de menor ou igual, pode ser também de maior ou igual.

6. Referências

ANDRADE, E. L. **Introdução à pesquisa operacional**: métodos e modelos para análise de decisões. 4 ed. Rio de Janeiro: LTC, 2009.

CORMEN, Thomas H. **Introduction to algorithm**. 23. ed. MIT: McGraw-Hill, 1999.

Chvátal, V. **Linear Programming**, W. H. Freeman and Company, 1999.

DANTZIG, G. B.; THAPA, M. N. **Linear Programming: 2: Theory and Extensions**. Nova York: [s.n.]. Edição: Softcover reprint of hardcover 1st ed. 2003.

Garfinkel, R. S. & Nemhauser, G. L. **Integer Programming**, John Wiley & Sons, 1972.

GNU Linear Programming Kit. **IBM Developer**, 2006. Disponível em: <https://www.ibm.com/developerworks/br/library/l-glpk1/index.html>. Acesso em: 03, de julho de 2020.

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização Combinatória e Programação Linear.** 2 Ed. Editora Campus, 2000.

GOLDBARG, M. C.; LUNA, H. P. L.; GOLDBARG, E. F. G. **Programação linear e fluxos em redes**. - 1. ed. - Rio de Janeiro : Elsevier, 2015.

Gurobi Optimizer. **GUROBI OPTIMIZATION**. Disponível em: <https://www.gurobi.com/products/gurobi-optimizer/>. Acesso em: 03, de julho de 2020.

HILLIER, F. S. LIEBERMAN, G. J. **Introdução à pesquisa operacional**. 9 ed. Porto Alegre : McGraw-Hill, 2013.

História da Pesquisa Operacional. **PHPSimplex**, 2006. Disponível em: <http://www.phpsimplex.com/pt/historia.htm>. Acesso em: 15, de março de 2020.

LACHTERMACHER, G. **Pesquisa operacional na tomada de decisões**. 4 ed. Rio de Janeiro: Campus, 2009.

MURTY, Katta G. **Linear programing. Stanford: John Wiley & Sons**, 1983.

NASH, J. C. THE (DANTZIG) SIMFLEX METHOD FOR LINEAR PROGRAMM. **Computing in Science & Engineering**, v. 2, n. 1, p. 29–31, 2000.

NEZAMABADI, H.; VAHIDINASAB, V. Two stage decision making of technical virtual power plants in electricity market via Nash-SFE equilibrium. **2015 3rd International Istanbul Smart Grid Congress and Fair, ICSG 2015,** 2015.

PLOSKAS, N.; SAMARAS, N. Efficient GPU-based implementations of simplex type algorithms. **APPLIED MATHEMATICS AND COMPUTATION**, v. 250, p. 552–570, 2015.

Programação Linear. **Google Play**, 2017. Disponível em: <https://play.google.com/store/apps/details?id=com.programacao.linear&hl=en_US> . Acesso em: 15, de março de 2020.

SOUSA, R. S. Métodos tipo dual simplex para problemas de otimização linear canalizados *.**Tese doutorado Instituto de Ciências Matemáticas e de Computação - ICMC-USP**, p.168, 2005.

APÊNDICE - Código Fonte

Classe: AppPL

```
package br.ufpb.dcx.tcc.mainpl;
import javax.swing.JOptionPane;
import br.ufpb.dcx.tcc.pl.FachadaPL;
import br.ufpb.dcx.tcc.pl.TipoProblemaPL;
/**
 *
 * Classe concreta que permite executar a aplicação de Programação Linear.
 *
 * @author Wellerson Alex
 *
 */
public class AppPL {
	/** Executa toda a aplicação recebendo os dados de entrada*/
	public static void main(String[] args) {
		/** Propriedade fachada*/
		FachadaPL fachada;
		/** Propriedade valor incorreto para evitar erros de entrada*/
		boolean valorIncorreto = true;
		/** Propriedade tipo de problema*/
		int tipoProblema;
		/** Recebe o tipo de problema*/
		String aux = JOptionPane.showInputDialog(null, "Informe o número de restrições do
problema?", "Valores possíveis: 1, 2, 3, 4, 5", JOptionPane.QUESTION_MESSAGE);
		tipoProblema = Integer.parseInt(aux);
		/** Corrige os dados de entrada caso venha incorreto*/
		while(valorIncorreto) {
			if(tipoProblema > 0 && tipoProblema < 6) {
				valorIncorreto = false;
			}else {
				JOptionPane.showMessageDialog(null, "Apenas probelmas com o
número de restrições entre 1 e 5",  "O programa não funciona para "+aux+" rstrições",
JOptionPane.ERROR_MESSAGE);
				aux = JOptionPane.showInputDialog(null, "Informe o número de
restrições do problema?", "Valores possíveis: 1, 2, 3, 4, 5", JOptionPane.QUESTION_MESSAGE);
				tipoProblema = Integer.parseInt(aux);
			}
		}
		/** Direciona o tipo de problema para sua determinada classe*/
		if (tipoProblema == 1) {
			fachada = new FachadaPL(TipoProblemaPL.Maximizar1R);
		} else if (tipoProblema == 2) {
			fachada = new FachadaPL(TipoProblemaPL.Maximizar2R);
		} else if (tipoProblema == 3) {
			fachada = new FachadaPL(TipoProblemaPL.Maximizar3R);
		} else if (tipoProblema == 4) {
			fachada = new FachadaPL(TipoProblemaPL.Maximizar4R);
		} else{
			fachada = new FachadaPL(TipoProblemaPL.Maximizar5R);
		}
		fachada.entradaDados();
	}
}
```

Classe: FacahadaPL

```
package br.ufpb.dcx.tcc.pl;
/**
 *
 * Classe concreta que apenas chama outras classes, é uma fachada.
 *
 * @author Wellerson Alex
 *
 */
public class FachadaPL {
	/** Propriedade probelma */
	private ProgramacaoLinear problema = new ProgramacaoLinear();
	/** Escolhe o tipo de maximização de acordo com o tipo de problema */
	public FachadaPL(TipoProblemaPL tipoProblema) {
		/** Probelmas com até uma restrição */
		if(tipoProblema.equals(problema.maximizar1R())) {
			problema = new Maximizar1R();
		}
		/** Probelmas com até duas restrições */
		else if(tipoProblema.equals(problema.maximizar2R())) {
			problema = new Maximizar2R();
		}
		/** Probelmas com até três restrições */
		else if(tipoProblema.equals(problema.maximizar3R())) {
			problema = new Maximizar3R();
		}
		/** Probelmas com até quatro restrições */
		else if(tipoProblema.equals(problema.maximizar4R())) {
			problema = new Maximizar4R();
		}
		/** Probelmas com até 5 restrições */
		else {
			problema = new Maximizar5R();
		}
	}
	/**
	 *
	 * Recebe todos os dados de entrada: função objetivo, restrições, variáveis,
	  variáveis de folga e termos independentes.
	 *
	 * @return Solução ótima para Z, todas as iterações feitas na tabela e
	  os valores mais restritivos para cada escolha de linha em cada iteração.
	 *
	 */
	public void entradaDados() {
		problema.entradaDados();
	}
}
```

Classe: Maximizar1R

```
package br.ufpb.dcx.tcc.pl;
import java.text.DecimalFormat;
import java.util.Arrays;
import javax.swing.JOptionPane;
/**
 *
```

```
 * Classe concreta para maximizar uma função obejetivo Z, com uma restrição
 *
 * @author Wellerson Alex
 *
 */
public class Maximizar1R extends ProgramacaoLinear {
	/** Propriedade de contagem*/
	private int cont;
	/** Propriedade que guarda os valores iniciais de Z */
	private double[] guardarInicialZ;
	/** Propriedade que guarda os valores iniciais da restrição 1 */
	private double[] guardarInicialR1;
	/**
	 * Encotra a solução ótima para a função Z de forma recursiva.
	 *
	 *@return Tabela inicial, tabelas atualizadas e solução ótima.
	 */
	public double[] max1R(double[] Z, double[] r1) {
		if (cont == 0) {
			guardarInicialR1 = new double[r1.length];
			guardarInicialZ = new double[Z.length];
			guardarInicialR1 = r1;
			guardarInicialZ = Z;
		}
		verificarLoopTableau(Z, r1, guardarInicialR1, guardarInicialZ);
		if (chegouNoOtimo(Z)) {
			return Z;
		}
		mostrarQuadroInicial(Z, r1);
		cont += 1;
		int posicaoColuna = colunaMaiorValorPositivo(Z);
		double valorRestritivo1 = linhaMaisRestrtitiva(r1, posicaoColuna - 1);
		double[] valoresRestritivos = inserir(valorRestritivo1);
		int posicaoLinha = linhaDoMenorValorRestritivoPositivo(valoresRestritivos);
		double[] linhaFixa = linhaEscolhida(posicaoLinha, r1);
		double interseccao = linhaFixa[posicaoColuna - 1];
		double valorZerarColuna1 = calcularValorEscalonamento(interseccao,
r1[posicaoColuna - 1]);
		double valorZerarColunaZ = calcularValorEscalonamento(interseccao,
Z[posicaoColuna - 1]);
		r1 = zerarColuna(linhaFixa, r1, valorZerarColuna1);
		Z = zerarColuna(linhaFixa, Z, valorZerarColunaZ);
		System.out.println();
		System.out.println("A coluna escolhida foi: " + posicaoColuna);
		if(posicaoLinha == 0) {
			JOptionPane.showMessageDialog(null, "O problema é ilimitado", "Ilimitado",
JOptionPane.ERROR_MESSAGE);
			throw new IllegalArgumentException("Problema Ilimitado");
		}
		System.out.println("A linha escolhida foi: " + posicaoLinha);
		System.out.println();
		System.out.println("Valores restritivos: " + Arrays.toString(valoresRestritivos));
		mostrarIteracao(Z, r1);
		return max1R(Z, r1);
	}
	/** Mostrar cada iteração */
	private void mostrarIteracao(double[] Z, double[] r1) {
		System.out.println();
```

```
                System.out.println("Tabela Resultante da " + cont + "ª Iteração:");
                double[] printR1 = formatarDecimaisUmaCasa(r1);
                double[] printZ = formatarDecimaisUmaCasa(Z);
                System.out.println(Arrays.toString(printR1));
                System.out.println(Arrays.toString(printZ));
        }
        /** Verifica se entrou em Loop */
        private void verificarLoopTableau(double[] Z, double[] r1, double[] guardarInicialR1, double[]
guardarInicialZ) {
                if (cont > 0) {
                        boolean testeR1 = testeLoop(guardarInicialR1, r1);
                        boolean testeZ = testeLoop(guardarInicialZ, Z);
                        if (testeR1 && testeZ) {
                                JOptionPane.showMessageDialog(null, "O problema entrou em loop,
não foi possível resolver", "Loop Tableau", JOptionPane.ERROR_MESSAGE);
                                System.out.println();
                                System.out.println("Os valores são iguais nas duas tabelas, por isso
o Loop é gerado");
                                System.out.println();
                                mostrarQuadroInicialLoop(Z, r1);
                                mostrarIteracao(Z, r1);
                                throw new IllegalArgumentException("Loop Tableau");
                        }
                }
        }
        /** Exibe tabela inicial quando há Loop */
        private void mostrarQuadroInicialLoop(double[] Z, double[] r1) {
                System.out.println("Tabela Inicial:");
                r1 = formatarDecimaisUmaCasa(r1);
                Z = formatarDecimaisUmaCasa(Z);
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(Z));
                System.out.println();
        }
        /** Testa se as tabelas são iguais */
        private boolean testeLoop(double[] inicial, double[] r) {
                double[] r1Formatado = formatarDecimaisUmaCasa(r);
                double[] inicialFormatodo = formatarDecimaisUmaCasa(inicial);
                int cont = 0;
                for (int i = 0; i < r1Formatado.length; i++) {
                        if (inicialFormatodo[i] == r1Formatado[i]) {
                                cont += 1;
                        }
                }
                if (cont == r.length) {
                        return true;
                } else {
                        return false;
                }
        }
        /** Formatar para uma casa decimal */
        private double[] formatarDecimaisUmaCasa(double[] vetor) {
                double[] novo = new double[vetor.length];
                DecimalFormat umaCasa = new DecimalFormat("0.0");
                for (int i = 0; i < vetor.length; i++) {
                        String v = umaCasa.format(vetor[i]);
                        String[] part1 = v.split("[,]");
                        String v1 = part1[0] + "." + part1[1];
```

```
                double valor1 = Double.parseDouble(v1);
                novo[i] = valor1;
        }
        return novo;
}
/** Verifica se a solução foi encontrada*/
public boolean chegouNoOtimo(double[] max) {
        for (int i = 0; i < max.length; i++) {
                if (max[i] > 0) {
                        return false;
                }
        }
        return true;
}
/** Exibe tabela inicial*/
private void mostrarQuadroInicial(double[] Z, double[] r1) {
        if (cont == 0) {
                System.out.println("Tabela Inicial:");
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(Z));
        }
}
/** Escolhe a coluna da tabela*/
public int colunaMaiorValorPositivo(double[] v) {
        double maior = Integer.MIN_VALUE;
        int coluna = 0;
        for (int i = 0; i < v.length; i++) {

                if (v[i] > maior && v[i] > 0) {
                        maior = v[i];
                        coluna = i + 1;
                }
        }
        return coluna;
}
/** Esolhe a linha da tabela*/
public double linhaMaisRestrtitiva(double[] v, int i) {
        double valor = 0;
        if (v[(int) i] >= 0) {
                double s = v[v.length - 1];
                double r = v[(int) i];
                if (r == 0) {
                        return -1;
                } else {
                        valor = s / r;
                        return valor;
                }
        } else {
                return -1;
        }
}
public double[] inserir(double valor1) {
        double vo1 = formatarDecimais(valor1);
        double[] v = new double[1];
        v[0] = vo1;
        return v;
}
private double formatarDecimais(double valor) {
```

```
		DecimalFormat umaCasa = new DecimalFormat("0.0000000000");
		String v = umaCasa.format(valor);
		String[] part1 = v.split("[,]");
		String v1 = part1[0] + "." + part1[1];
		double valor1 = Double.parseDouble(v1);
		return valor1;
	}
	public int linhaDoMenorValorRestritivoPositivo(double[] v) {
		double menor = Integer.MAX_VALUE;
		int linha = 0;
		for (int i = 0; i < v.length; i++) {
			if (v[i] < menor && v[i] >= 0) {
				menor = v[i];
				linha = i + 1;
			}
		}
		return linha;
	}
	public double[] linhaEscolhida(int linha, double[] r1) {
		return r1;
	}
	/** Definir valor para zerar coluna*/
	private double calcularValorEscalonamento(double interseccao, double d) {
		double vo1 = formatarDecimais(interseccao);
		double valor;
		valor = (d / vo1);
		valor = (valor * -1);
		double val1 = formatarDecimais(valor);
		return val1;
	}
	/** Realiza escalonamento para atualização da tabela*/
	public double[] zerarColuna(double[] vr, double[] vz, double valor) {
		double[] novoV = new double[vz.length];
		int cont = 0;
		for (int i = 0; i < vr.length; i++) {
			if (vr[i] == vz[i]) {
				cont += 1;
			}
		}
		atualizarValores(vr, vz, valor, novoV);
		int quantP = vr.length;
		if (cont == quantP) {
			return vr;
		} else {
			return novoV;
		}
	}
	private void atualizarValores(double[] vr, double[] vz, double valor, double[] novoV) {
		for (int i = 0; i < vr.length; i++) {
			double vIndice = ((vr[i] * valor) + vz[i]);
			double vo1 = formatarDecimais(vIndice);
			novoV[i] = vo1;
		}
	}
	/** Recebe dados da tabela*/
	public void entradaDados() {
		Maximizar1R problema = new Maximizar1R();
		int tamanho;
```

```
		String aux;
		int cont = 0;
		aux = JOptionPane.showInputDialog(null,"Quantas variáveis o problema apresenta?
(Inclua também as variáveis de folga e o termo independente b)");
		tamanho = Integer.parseInt(aux);
		double[] Z = valoresDeZ(tamanho, cont);
		double[] restricao1 = valoresR1(tamanho);
		double[] maxZ = problema.max1R(Z, restricao1);
		double solucaoOtima = maxZ[maxZ.length - 1] * -1;
		System.out.println();
		System.out.println("O valor da sólução ótima é: " + solucaoOtima);
	}
	/** Recebe valores para a primeira restrição*/
	private double[] valoresR1(int tamanho) {
		int posicao = 0;
		String aux;
		int cont = 0;
		double[] restricao1 = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null,"Para a restrição 1", "digite
o valor do termo independente b1");
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao1, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a restrição 1, digite o
valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao1, i);
			}
		}
		return restricao1;
	}
	/** Recebe dados da função objetivo Z*/
	private double[] valoresDeZ(int tamanho, int cont) {
		int posicao = 0;
		String aux;
		double[] Z = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor do termo independente");
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
			}
		}
		return Z;
	}
	/** Trata caso com valor fracionário */
	private int posicaoBarra(String aux, int posicao) {
		posicao = aux.indexOf("/");
```

```
			return posicao;
		}
		/** Realiza operações com fração */
		private void verificarFracao(String aux, int posicao, double[] Z, int i) {
			if (posicao > 0) {
				double numerador = Double.parseDouble(aux.substring(0, posicao));
				double denominador = Double.parseDouble(aux.substring(posicao + 1,
aux.length()));
				Z[i] = (numerador / denominador);
			} else {
				Z[i] = Double.parseDouble(aux);
			}
		}
}
```

Classe: Maximizar2R

```
package br.ufpb.dcx.tcc.pl;
import java.text.DecimalFormat;
import java.util.Arrays;
import javax.swing.JOptionPane;
/**
 *
 * Classe concreta para maximizar uma função obejetivo Z, com duas restrição
 *
 * @author Wellerson Alex
 *
 */
public class Maximizar2R extends ProgramacaoLinear {
		/** Propriedade de contagem */
		private int cont;
		/** Propriedade que guarda os valores iniciais de Z */
		private double[] guardarInicialZ;
		/** Propriedade que guarda os valores iniciais da restrição 1 */
		private double[] guardarInicialR1;
		/** Propriedade que guarda os valores iniciais da restrição 2 */
		private double[] guardarInicialR2;
		/**
		 * Encotra a solução ótima para a função Z de forma recursiva.
		 *
		 * @return Tabela inicial, tabelas atualizadas e solução ótima.
		 */
		public double[] max2R(double[] Z, double[] r1, double[] r2) {
			if (cont == 0) {
				guardarInicialR1 = new double[r1.length];
				guardarInicialR2 = new double[r2.length];
				guardarInicialZ = new double[Z.length];
				guardarInicialR1 = r1;
				guardarInicialR2 = r2;
				guardarInicialZ = Z;
			}
			verificarLoopTableau(Z, r1, r2, guardarInicialR1, guardarInicialR2, guardarInicialZ);
			if (chegouNoOtimo(Z)) {
				return Z;
			}
			mostrarQuadroInicial(Z, r1, r2);
			cont += 1;
			int posicaoColuna = colunaMaiorValorPositivo(Z);
			double valorRestritivo1 = linhaMaisRestrtitiva(r1, posicaoColuna - 1);
```

```
                double valorRestritivo2 = linhaMaisRestrtitiva(r2, posicaoColuna - 1);
                double[] valoresRestritivos = inserir(valorRestritivo1, valorRestritivo2);
                int posicaoLinha = linhaDoMenorValorRestritivoPositivo(valoresRestritivos);
                double[] linhaFixa = linhaEscolhida(posicaoLinha, r1, r2);
                double interseccao = linhaFixa[posicaoColuna - 1];
                double valorZerarColuna1 = calcularValorEscalonamento(interseccao,
r1[posicaoColuna - 1]);
                double valorZerarColuna2 = calcularValorEscalonamento(interseccao,
r2[posicaoColuna - 1]);
                double valorZerarColunaZ = calcularValorEscalonamento(interseccao,
Z[posicaoColuna - 1]);
                r1 = zerarColuna(linhaFixa, r1, valorZerarColuna1);
                r2 = zerarColuna(linhaFixa, r2, valorZerarColuna2);
                Z = zerarColuna(linhaFixa, Z, valorZerarColunaZ);
                System.out.println();
                System.out.println("A coluna escolhida foi: " + posicaoColuna);
                if(posicaoLinha == 0) {
                        JOptionPane.showMessageDialog(null, "O problema é ilimitado", "Ilimitado",
JOptionPane.ERROR_MESSAGE);
                        throw new IllegalArgumentException("Problema Ilimitado");
                }
                System.out.println("A linha escolhida foi: " + posicaoLinha);
                System.out.println();
                System.out.println("Valores restritivos: " + Arrays.toString(valoresRestritivos));
                mostrarIteracao(Z, r1, r2);
                return max2R(Z, r1, r2);
        }
        /** Verifica se entrou em Loop */
        private void verificarLoopTableau(double[] Z, double[] r1, double[] r2, double[]
guardarInicialR1, double[] guardarInicialR2, double[] guardarInicialZ) {
                if (cont > 0) {
                        boolean testeR1 = testeLoop(guardarInicialR1, r1);
                        boolean testeR2 = testeLoop(guardarInicialR2, r2);
                        boolean testeZ = testeLoop(guardarInicialZ, Z);
                        if (testeR1 && testeR2 && testeZ) {
                                JOptionPane.showMessageDialog(null, "O problema entrou em loop,
não foi possível resolver", "Loop Tableau", JOptionPane.ERROR_MESSAGE);
                                System.out.println();
                                System.out.println("Os valores são iguais nas duas tabelas, por isso
o Loop é gerado");
                                System.out.println();
                                mostrarQuadroInicialLoop(Z, r1, r2);
                                mostrarIteracao(Z, r1, r2);
                                throw new IllegalArgumentException("Loop Tableau");
                        }
                }
        }
        /** Exibe tabela inicial quando há Loop */
        private void mostrarQuadroInicialLoop(double[] Z, double[] r1, double[] r2) {
                System.out.println("Tabela Inicial:");
                r1 = formatarDecimaisUmaCasa(r1);
                r2 = formatarDecimaisUmaCasa(r2);
                Z = formatarDecimaisUmaCasa(Z);
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(r2));
                System.out.println(Arrays.toString(Z));
                System.out.println();
        }
```

```
/** Testa se as tabelas são iguais */
private boolean testeLoop(double[] inicial, double[] r) {
	double[] r1Formatado = formatarDecimaisUmaCasa(r);
	double[] inicialFormatodo = formatarDecimaisUmaCasa(inicial);
	int cont = 0;
	for (int i = 0; i < r1Formatado.length; i++) {
		if (inicialFormatodo[i] == r1Formatado[i]) {
			cont += 1;
		}
	}
	if (cont == r.length) {
		return true;
	} else {
		return false;
	}
}
/** Mostrar cada iteração */
private void mostrarIteracao(double[] Z, double[] r1, double[] r2) {
	System.out.println();
	System.out.println("Tabela Resultante da " + cont + "ª Iteração:");
	double[] printR1 = formatarDecimaisUmaCasa(r1);
	double[] printR2 = formatarDecimaisUmaCasa(r2);
	double[] printZ = formatarDecimaisUmaCasa(Z);
	System.out.println(Arrays.toString(printR1));
	System.out.println(Arrays.toString(printR2));
	System.out.println(Arrays.toString(printZ));
}
/** Formatar para uma casa decimal */
private double[] formatarDecimaisUmaCasa(double[] vetor) {
	double[] novo = new double[vetor.length];
	DecimalFormat umaCasa = new DecimalFormat("0.0");
	for (int i = 0; i < vetor.length; i++) {
		String v = umaCasa.format(vetor[i]);
		String[] part1 = v.split("[,]");
		String v1 = part1[0] + "." + part1[1];
		double valor1 = Double.parseDouble(v1);
		novo[i] = valor1;
	}
	return novo;
}
/** Verifica se a solução foi encontrada */
public boolean chegouNoOtimo(double[] max) {
	for (int i = 0; i < max.length; i++) {
		if (max[i] > 0) {
			return false;
		}
	}
	return true;
}
/** Exibe tabela inicial */
private void mostrarQuadroInicial(double[] Z, double[] r1, double[] r2) {
	if (cont == 0) {
		System.out.println("Tabela Inicial:");
		r1 = formatarDecimaisUmaCasa(r1);
		r2 = formatarDecimaisUmaCasa(r2);
		Z = formatarDecimaisUmaCasa(Z);
		System.out.println(Arrays.toString(r1));
		System.out.println(Arrays.toString(r2));
```

```
                System.out.println(Arrays.toString(Z));
        }
}
/** Escolhe a coluna da tabela */
public int colunaMaiorValorPositivo(double[] v) {
        double maior = Integer.MIN_VALUE;
        int coluna = 0;
        for (int i = 0; i < v.length; i++) {
                if (v[i] > maior && v[i] > 0) {
                        maior = v[i];
                        coluna = i + 1;
                }
        }
        return coluna;
}
/** Esolhe a linha da tabela */
public double linhaMaisRestrtitiva(double[] v, int i) {
        double valor = 0;
        if (v[(int) i] >= 0) {
                double s = v[v.length - 1];
                double r = v[(int) i];
                if (r == 0) {
                        return -1;
                } else {
                        valor = s / r;
                        return valor;
                }
        } else {
                return -1;
        }
}
public double[] inserir(double valor1, double valor2) {
        double vo1 = formatarDecimais(valor1);
        double vo2 = formatarDecimais(valor2);
        double[] v = new double[2];
        v[0] = vo1;
        v[1] = vo2;
        return v;
}
private double formatarDecimais(double valor) {
        DecimalFormat ateDezCasas = new DecimalFormat("0.0000000000");
        String v = ateDezCasas.format(valor);
        String[] part1 = v.split("[,]");
        String v1 = part1[0] + "." + part1[1];
        double valor1 = Double.parseDouble(v1);
        return valor1;
}
public int linhaDoMenorValorRestritivoPositivo(double[] v) {
        double menor = Integer.MAX_VALUE;
        int linha = 0;
        for (int i = 0; i < v.length; i++) {
                if (v[i] < menor && v[i] >= 0) {
                        menor = v[i];
                        linha = i + 1;
                }
        }
        return linha;
}
```

```
	public double[] linhaEscolhida(int linha, double[] r1, double[] r2) {
		if (linha == 1) {
			return r1;
		} else {
			return r2;
		}
	}
	/** Definir valor para zerar coluna */
	private double calcularValorEscalonamento(double interseccao, double d) {
		double vo1 = formatarDecimais(interseccao);
		double valor;
		valor = (d / vo1);
		valor = (valor * -1);
		double val1 = formatarDecimais(valor);
		return val1;
	}
	/** Realiza escalonamento para atualização da tabela */
	public double[] zerarColuna(double[] vr, double[] vz, double valor) {
		double[] novoV = new double[vz.length];
		int cont = 0;
		for (int i = 0; i < vr.length; i++) {
			if (vr[i] == vz[i]) {
				cont += 1;
			}
		}
		atualizarValores(vr, vz, valor, novoV);
		int quantP = vr.length;
		if (cont == quantP) {
			return vr;
		} else {
			return novoV;
		}
	}
	private void atualizarValores(double[] vr, double[] vz, double valor, double[] novoV) {
		for (int i = 0; i < vr.length; i++) {
			double vIndice = ((vr[i] * valor) + vz[i]);
			double vo1 = formatarDecimais(vIndice);
			novoV[i] = vo1;
		}
	}
	/** Recebe dados da tabela */
	public void entradaDados() {
		Maximizar2R problema = new Maximizar2R();
		int tamanho;
		String aux;
		int cont = 0;
		aux = JOptionPane.showInputDialog(null,
				"Quantas variáveis o problema apresenta? (Inclua também as
variáveis de folga e o termo independente b)");
		tamanho = Integer.parseInt(aux);
		double[] Z = valoresDeZ(tamanho, cont);
		double[] restricao1 = valoresR1(tamanho);
		double[] restricao2 = valoresR2(tamanho);
		double[] maxZ = problema.max2R(Z, restricao1, restricao2);
		double solucaoOtima = maxZ[maxZ.length - 1] * -1;
		System.out.println();
		System.out.println("O valor da sólução ótima é: " + solucaoOtima);
	}
```

```
	/** Recebe valores para a primeira restrição */
	private double[] valoresR1(int tamanho) {
		int restricao = 1;
		double[] restricao1 = adicionarVariaveis(tamanho, restricao);
		return restricao1;
	}
	/** Recebe valores para a segunda restrição */
	private double[] valoresR2(int tamanho) {
		int restricao = 2;
		double[] restricao2 = adicionarVariaveis(tamanho, restricao);
		return restricao2;
	}
	/** Adiciona os valores */
	private double[] adicionarVariaveis(int tamanho, int restri) {
		int posicao = 0;
		String aux;
		int cont = 0;
		double[] restricao = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null, "Para a restrição " + restri
+ ", digite o valor do termo independente b" + restri);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a restrição " + restri
+ ", digite o valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao, i);
			}
		}
		return restricao;
	}
	/** Recebe dados da função objetivo Z */
	private double[] valoresDeZ(int tamanho, int cont) {
		String aux;
		int posicao = 0;
		double[] Z = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor do termo independente");
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
			}
		}
		return Z;
	}
	/** Trata caso com valor fracionário */
	private int posicaoBarra(String aux, int posicao) {
		posicao = aux.indexOf("/");
```

```
                return posicao;
        }
        /** Realiza operações com fração */
        private void verificarFracao(String aux, int posicao, double[] Z, int i) {
                if (posicao > 0) {
                        double numerador = Double.parseDouble(aux.substring(0, posicao));
                        double denominador = Double.parseDouble(aux.substring(posicao + 1,
aux.length()));
                        Z[i] = (numerador / denominador);
                } else {
                        Z[i] = Double.parseDouble(aux);
                }
        }
}
```

Classe: Maximizar3R

```
package br.ufpb.dcx.tcc.pl;
import java.text.DecimalFormat;
import java.util.Arrays;
import javax.swing.JOptionPane;
/**
 *
 * Classe concreta para maximizar uma função obejetivo Z, com três restrição
 *
 * @author Wellerson Alex
 *
 */
public class Maximizar3R extends ProgramacaoLinear {
        /** Propriedade de contagem*/
        private int cont;
        /** Propriedade que guarda os valores iniciais de Z */
        private double[] guardarInicialZ;
        /** Propriedade que guarda os valores iniciais da restrição 1 */
        private double[] guardarInicialR1;
        /** Propriedade que guarda os valores iniciais da restrição 2 */
        private double[] guardarInicialR2;
        /** Propriedade que guarda os valores iniciais da restrição 3 */
        private double[] guardarInicialR3;
        /**
         * Encotra a solução ótima para a função Z de forma recursiva.
         *
         *@return Tabela inicial, tabelas atualizadas e solução ótima.
         */
        public double[] max3R(double[] Z, double[] r1, double[] r2, double[] r3) {
                if (cont == 0) {
                        guardarInicialR1 = new double[r1.length];
                        guardarInicialR2 = new double[r2.length];
                        guardarInicialR3 = new double[r3.length];
                        guardarInicialZ = new double[Z.length];
                        guardarInicialR1 = r1;
                        guardarInicialR2 = r2;
                        guardarInicialR3 = r3;
                        guardarInicialZ = Z;
                }
                verificarLoopTableau(Z, r1, r2, r3, guardarInicialR1, guardarInicialR2,
guardarInicialR3, guardarInicialZ);
                if (chegouNoOtimo(Z)) {
                        return Z;
```

```
                }
                mostrarQuadroInicial(Z, r1, r2, r3);
                cont += 1;
                int posicaoColuna = colunaMaiorValorPositivo(Z);
                double valorRestritivo1 = linhaMaisRestrtitiva(r1, posicaoColuna - 1);
                double valorRestritivo2 = linhaMaisRestrtitiva(r2, posicaoColuna - 1);
                double valorRestritivo3 = linhaMaisRestrtitiva(r3, posicaoColuna - 1);
                double[] valoresRestritivos = inserir(valorRestritivo1, valorRestritivo2,
valorRestritivo3);
                int posicaoLinha = linhaDoMenorValorRestritivoPositivo(valoresRestritivos);
                double[] linhaFixa = linhaEscolhida(posicaoLinha, r1, r2, r3);
                double interseccao = linhaFixa[posicaoColuna - 1];
                double valorZerarColuna1 = calcularValorEscalonamento(interseccao,
r1[posicaoColuna - 1]);
                double valorZerarColuna2 = calcularValorEscalonamento(interseccao,
r2[posicaoColuna - 1]);
                double valorZerarColuna3 = calcularValorEscalonamento(interseccao,
r3[posicaoColuna - 1]);
                double valorZerarColunaZ = calcularValorEscalonamento(interseccao,
Z[posicaoColuna - 1]);
                r1 = zerarColuna(linhaFixa, r1, valorZerarColuna1);
                r2 = zerarColuna(linhaFixa, r2, valorZerarColuna2);
                r3 = zerarColuna(linhaFixa, r3, valorZerarColuna3);
                Z = zerarColuna(linhaFixa, Z, valorZerarColunaZ);
                System.out.println();
                System.out.println("A coluna escolhida foi: " + posicaoColuna);
                if(posicaoLinha == 0) {
                        JOptionPane.showMessageDialog(null, "O problema é ilimitado", "Ilimitado",
JOptionPane.ERROR_MESSAGE);
                        throw new IllegalArgumentException("Problema Ilimitado");
                }
                System.out.println("A linha escolhida foi: " + posicaoLinha);
                System.out.println();
                System.out.println("Valores restritivos: " + Arrays.toString(valoresRestritivos));
                mostrarIteracao(Z, r1, r2, r3);
                return max3R(Z, r1, r2, r3);
        }
        /** Verifica se entrou em Loop */
        private void verificarLoopTableau(double[] Z, double[] r1, double[] r2, double[] r3, double[]
guardarInicialR1, double[] guardarInicialR2, double[] guardarInicialR3, double[] guardarInicialZ) {
                if (cont > 0) {
                        boolean testeR1 = testeLoop(guardarInicialR1, r1);
                        boolean testeR2 = testeLoop(guardarInicialR2, r2);
                        boolean testeR3 = testeLoop(guardarInicialR3, r3);
                        boolean testeZ = testeLoop(guardarInicialZ, Z);
                        if (testeR1 && testeR2 && testeR3 && testeZ) {
                                JOptionPane.showMessageDialog(null, "O problema entrou em loop,
não foi possível resolver", "Loop Tableau", JOptionPane.ERROR_MESSAGE);
                                System.out.println();
                                System.out.println("Os valores são iguais nas duas tabelas, por isso
o Loop é gerado");
                                System.out.println();
                                mostrarQuadroInicialLoop(Z, r1, r2, r3);
                                mostrarIteracao(Z, r1, r2, r3);
                                throw new IllegalArgumentException("Loop Tableau");
                        }
                }
        }
```

```
/** Mostrar cada iteração */
private void mostrarIteracao(double[] Z, double[] r1, double[] r2, double [] r3) {
	System.out.println();
	System.out.println("Tabela Resultante da " + cont + "ª Iteração:");
	double[] printR1 = formatarDecimaisUmaCasa(r1);
	double[] printR2 = formatarDecimaisUmaCasa(r2);
	double[] printR3 = formatarDecimaisUmaCasa(r3);
	double[] printZ = formatarDecimaisUmaCasa(Z);
	System.out.println(Arrays.toString(printR1));
	System.out.println(Arrays.toString(printR2));
	System.out.println(Arrays.toString(printR3));
	System.out.println(Arrays.toString(printZ));
}
/** Exibe tabela inicial quando há Loop */
private void mostrarQuadroInicialLoop(double[] Z, double[] r1, double[] r2, double [] r3) {
	System.out.println("Tabela Inicial:");
	r1 = formatarDecimaisUmaCasa(r1);
	r2 = formatarDecimaisUmaCasa(r2);
	r3 = formatarDecimaisUmaCasa(r3);
	Z = formatarDecimaisUmaCasa(Z);
	System.out.println(Arrays.toString(r1));
	System.out.println(Arrays.toString(r2));
	System.out.println(Arrays.toString(r3));
	System.out.println(Arrays.toString(Z));
	System.out.println();
}
/** Testa se as tabelas são iguais */
private boolean testeLoop(double[] inicial, double[] r) {
	double[] r1Formatado = formatarDecimaisUmaCasa(r);
	double[] inicialFormatodo = formatarDecimaisUmaCasa(inicial);
	int cont = 0;
	for (int i = 0; i < r1Formatado.length; i++) {
		if (inicialFormatodo[i] == r1Formatado[i]) {
			cont += 1;
		}
	}
	if (cont == r.length) {
		return true;
	} else {
		return false;
	}
}
/** Formatar para uma casa decimal */
private double[] formatarDecimaisUmaCasa(double[] vetor) {
	double[] novo = new double[vetor.length];
	DecimalFormat umaCasa = new DecimalFormat("0.0");
	for (int i = 0; i < vetor.length; i++) {
		String v = umaCasa.format(vetor[i]);
		String[] part1 = v.split("[,]");
		String v1 = part1[0] + "." + part1[1];
		double valor1 = Double.parseDouble(v1);
		novo[i] = valor1;
	}
	return novo;
}
/** Verifica se a solução foi encontrada*/
public boolean chegouNoOtimo(double[] max) {
	for (int i = 0; i < max.length; i++) {
```

```
                if (max[i] > 0) {
                        return false;
                }
        }
        return true;
}
/** Exibe tabela inicial*/
private void mostrarQuadroInicial(double[] Z, double[] r1, double[] r2, double[] r3) {
        if (cont == 0) {
                System.out.println("Tabela Inicial:");
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(r2));
                System.out.println(Arrays.toString(r3));
                System.out.println(Arrays.toString(Z));
        }
}
/** Escolhe a coluna da tabela*/
public int colunaMaiorValorPositivo(double[] v) {
        double maior = Integer.MIN_VALUE;
        int coluna = 0;
        for (int i = 0; i < v.length; i++) {
                if (v[i] > maior && v[i] > 0) {
                        maior = v[i];
                        coluna = i + 1;
                }
        }
        return coluna;
}
/** Esolhe a linha da tabela*/
public double linhaMaisRestrtitiva(double[] v, int i) {
        double valor = 0;
        if (v[(int) i] >= 0) {
                double s = v[v.length - 1];
                double r = v[(int) i];
                if (r == 0) {
                        return -1;
                } else {
                        valor = s / r;
                        return valor;
                }
        } else {
                return -1;
        }
}
public double[] inserir(double valor1, double valor2, double valor3) {
        double vo1 = formatarDecimais(valor1);
        double vo2 = formatarDecimais(valor2);
        double vo3 = formatarDecimais(valor3);
        double[] v = new double[3];
        v[0] = vo1;
        v[1] = vo2;
        v[2] = vo3;
        return v;
}
private double formatarDecimais(double valor) {
        DecimalFormat umaCasa = new DecimalFormat("0.0000000000");
        String v = umaCasa.format(valor);
        String[] part1 = v.split("[,]");
```

```
		String v1 = part1[0] + "." + part1[1];
		double valor1 = Double.parseDouble(v1);
		return valor1;
	}
	public int linhaDoMenorValorRestritivoPositivo(double[] v) {
		double menor = Integer.MAX_VALUE;
		int linha = 0;
		for (int i = 0; i < v.length; i++) {
			if (v[i] < menor && v[i] >= 0) {
				menor = v[i];
				linha = i + 1;
			}
		}
		return linha;
	}
	public double[] linhaEscolhida(int linha, double[] r1, double[] r2, double[] r3) {
		if (linha == 1) {
			return r1;
		} else if (linha == 2) {
			return r2;
		} else {
			return r3;
		}
	}
	/** Definir valor para zerar coluna*/
	private double calcularValorEscalonamento(double interseccao, double d) {
		double vo1 = formatarDecimais(interseccao);
		double valor;
		valor = (d / vo1);
		valor = (valor * -1);
		double val1 = formatarDecimais(valor);
		return val1;
	}
	/** Realiza escalonamento para atualização da tabela*/
	public double[] zerarColuna(double[] vr, double[] vz, double valor) {
		double[] novoV = new double[vz.length];
		int cont = 0;
		for (int i = 0; i < vr.length; i++) {
			if (vr[i] == vz[i]) {
				cont += 1;
			}
		}
		atualizarValores(vr, vz, valor, novoV);
		int quantP = vr.length;
		if (cont == quantP) {
			return vr;
		} else {
			return novoV;
		}
	}
	private void atualizarValores(double[] vr, double[] vz, double valor, double[] novoV) {
		for (int i = 0; i < vr.length; i++) {
			double vIndice = ((vr[i] * valor) + vz[i]);
			double vo1 = formatarDecimais(vIndice);
			novoV[i] = vo1;
		}
	}
	/** Recebe dados da tabela*/
```

```
	public void entradaDados() {
		Maximizar3R problema = new Maximizar3R();
		String aux;
		int cont = 0;
		aux = JOptionPane.showInputDialog(null, "Quantas variáveis o problema apresenta?
(Inclua também as variáveis de folga e o termo independente b)");
		int tamanho = Integer.parseInt(aux);
		double[] Z = valoresDeZ(tamanho, cont);
		double[] restricao1 = valoresR1(tamanho);
		double[] restricao2 = valoresR2(tamanho);
		double[] restricao3 = valoresR3(tamanho);
		double[] maxZ = problema.max3R(Z, restricao1, restricao2, restricao3);
		double solucaoOtima = maxZ[maxZ.length - 1] * -1;
		System.out.println();
		System.out.println("O valor da sólução ótima é: " + solucaoOtima);
	}
	/** Recebe valores para a primeira restrição*/
	private double[] valoresR1(int tamanho) {
		int restricao = 1;
		double[] restricao1 = adicionarVariaveis(tamanho, restricao);
		return restricao1;
	}
	/** Recebe valores para a segunda restrição*/
	private double[] valoresR2(int tamanho) {
		int restricao = 2;
		double[] restricao2 = adicionarVariaveis(tamanho, restricao);
		return restricao2;
	}
	/** Recebe valores para a terceira restrição*/
	private double[] valoresR3(int tamanho) {
		int restricao = 3;
		double[] restricao3 = adicionarVariaveis(tamanho, restricao);
		return restricao3;
	}
	/** Adiciona os valores*/
	private double[] adicionarVariaveis(int tamanho, int restri) {
		int posicao = 0;
		String aux;
		int cont = 0;
		double[] restricao = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null,"Para a restrição "+ restri +
", digite o valor do termo independente b"+restri);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a restrição "+
restri+", digite o valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao, i);
			}
		}
		return restricao;
	}
	/** Recebe dados da função objetivo Z*/
	private double[] valoresDeZ(int tamanho, int cont) {
```

```
		String aux;
		int posicao = 0;
		double[] Z = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor do termo independente");
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
			}
		}
		return Z;
	}
	/** Trata caso com valor fracionário */
	private int posicaoBarra(String aux, int posicao) {
		posicao = aux.indexOf("/");
		return posicao;
	}
	/** Realiza operações com fração */
	private void verificarFracao(String aux, int posicao, double[] Z, int i) {
		if (posicao > 0) {
			double numerador = Double.parseDouble(aux.substring(0, posicao));
			double denominador = Double.parseDouble(aux.substring(posicao + 1,
aux.length()));
			Z[i] = (numerador / denominador);
		} else {
			Z[i] = Double.parseDouble(aux);
		}
	}
}
```

Classe: Maximizar4R

```
package br.ufpb.dcx.tcc.pl;
import java.text.DecimalFormat;
import java.util.Arrays;
import javax.swing.JOptionPane;
/**
 *
 * Classe concreta para maximizar uma função obejetivo Z, com quatro restrição
 *
 * @author Wellerson Alex
 *
 */
public class Maximizar4R extends ProgramacaoLinear {
	/** Propriedade de contagem*/
	private int cont;
	/** Propriedade que guarda os valores iniciais de Z */
	private double[] guardarInicialZ;
	/** Propriedade que guarda os valores iniciais da restrição 1 */
	private double[] guardarInicialR1;
	/** Propriedade que guarda os valores iniciais da restrição 2 */
	private double[] guardarInicialR2;
```

```
	/** Propriedade que guarda os valores iniciais da restrição 3 */
	private double[] guardarInicialR3;
	/** Propriedade que guarda os valores iniciais da restrição 4 */
	private double[] guardarInicialR4;
	/**
	 * Encotra a solução ótima para a função Z de forma recursiva.
	 *
	 *@return Tabela inicial, tabelas atualizadas e solução ótima.
	 */
	public double[] max4R(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4) {
		if (cont == 0) {
			guardarInicialR1 = new double[r1.length];
			guardarInicialR2 = new double[r2.length];
			guardarInicialR3 = new double[r3.length];
			guardarInicialR4 = new double[r4.length];
			guardarInicialZ = new double[Z.length];
			guardarInicialR1 = r1;
			guardarInicialR2 = r2;
			guardarInicialR3 = r3;
			guardarInicialR4 = r4;
			guardarInicialZ = Z;
		}
		verificarLoopTableau(Z, r1, r2, r3, r4, guardarInicialR1,
guardarInicialR2,guardarInicialR3, guardarInicialR4, guardarInicialZ);
		if (chegouNoOtimo(Z)) {
			return Z;
		}
		mostrarQuadroInicial(Z, r1, r2, r3, r4);
		cont += 1;
		int posicaoColuna = colunaMaiorValorPositivo(Z);
		double valorRestritivo1 = linhaMaisRestrtitiva(r1, posicaoColuna - 1);
		double valorRestritivo2 = linhaMaisRestrtitiva(r2, posicaoColuna - 1);
		double valorRestritivo3 = linhaMaisRestrtitiva(r3, posicaoColuna - 1);
		double valorRestritivo4 = linhaMaisRestrtitiva(r4, posicaoColuna - 1);
		double[] valoresRestritivos = inserir(valorRestritivo1, valorRestritivo2,
valorRestritivo3, valorRestritivo4);
		int posicaoLinha = linhaDoMenorValorRestritivoPositivo(valoresRestritivos);
		double[] linhaFixa = linhaEscolhida(posicaoLinha, r1, r2, r3, r4);
		double interseccao = linhaFixa[posicaoColuna - 1];
		double valorZerarColuna1 = calcularValorEscalonamento(interseccao,
r1[posicaoColuna - 1]);
		double valorZerarColuna2 = calcularValorEscalonamento(interseccao,
r2[posicaoColuna - 1]);
		double valorZerarColuna3 = calcularValorEscalonamento(interseccao,
r3[posicaoColuna - 1]);
		double valorZerarColuna4 = calcularValorEscalonamento(interseccao,
r4[posicaoColuna - 1]);
		double valorZerarColunaZ = calcularValorEscalonamento(interseccao,
Z[posicaoColuna - 1]);
		r1 = zerarColuna(linhaFixa, r1, valorZerarColuna1);
		r2 = zerarColuna(linhaFixa, r2, valorZerarColuna2);
		r3 = zerarColuna(linhaFixa, r3, valorZerarColuna3);
		r4 = zerarColuna(linhaFixa, r4, valorZerarColuna4);
		Z = zerarColuna(linhaFixa, Z, valorZerarColunaZ);
		System.out.println();
		System.out.println("A coluna escolhida foi: " + posicaoColuna);
		if(posicaoLinha == 0) {
```

```
                        JOptionPane.showMessageDialog(null, "O problema é ilimitado", "Ilimitado",
JOptionPane.ERROR_MESSAGE);
                        throw new IllegalArgumentException("Problema Ilimitado");
                }
                System.out.println("A linha escolhida foi: " + posicaoLinha);
                System.out.println();
                System.out.println("Valores restritivos: " + Arrays.toString(valoresRestritivos));
                mostrarIteracao(Z, r1, r2, r3, r4);
                return max4R(Z, r1, r2, r3, r4);
        }
        /** Verifica se entrou em Loop */
        private void verificarLoopTableau(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4,
double[] guardarInicialR1, double[] guardarInicialR2, double[] guardarInicialR3, double[]
guardarInicialR4, double[] guardarInicialZ) {
                if (cont > 0) {
                        boolean testeR1 = testeLoop(guardarInicialR1, r1);
                        boolean testeR2 = testeLoop(guardarInicialR2, r2);
                        boolean testeR3 = testeLoop(guardarInicialR2, r3);
                        boolean testeR4 = testeLoop(guardarInicialR2, r4);
                        boolean testeZ = testeLoop(guardarInicialZ, Z);
                        if (testeR1 && testeR2 && testeR3 && testeR4 && testeZ) {
                                JOptionPane.showMessageDialog(null, "O problema entrou em loop,
não foi possível resolver", "Loop Tableau", JOptionPane.ERROR_MESSAGE);
                                System.out.println();
                                System.out.println("Os valores são iguais nas duas tabelas, por isso
o Loop é gerado");
                                System.out.println();
                                mostrarQuadroInicialLoop(Z, r1, r2, r3, r4);
                                mostrarIteracao(Z, r1, r2, r3, r4);
                                throw new IllegalArgumentException("Loop Tableau");
                        }
                }
        }
        /** Exibe tabela inicial quando há Loop */
        private void mostrarQuadroInicialLoop(double[] Z, double[] r1, double[] r2, double[] r3,
double[] r4) {
                System.out.println("Tabela Inicial:");
                r1 = formatarDecimaisUmaCasa(r1);
                r2 = formatarDecimaisUmaCasa(r2);
                r3 = formatarDecimaisUmaCasa(r3);
                r4 = formatarDecimaisUmaCasa(r4);
                Z = formatarDecimaisUmaCasa(Z);
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(r2));
                System.out.println(Arrays.toString(r3));
                System.out.println(Arrays.toString(r4));
                System.out.println(Arrays.toString(Z));
                System.out.println();
        }
        /** Testa se as tabelas são iguais */
        private boolean testeLoop(double[] inicial, double[] r) {
                double[] r1Formatado = formatarDecimaisUmaCasa(r);
                double[] inicialFormatodo = formatarDecimaisUmaCasa(inicial);
                int cont = 0;
                for (int i = 0; i < r1Formatado.length; i++) {
                        if (inicialFormatodo[i] == r1Formatado[i]) {
                                cont += 1;
                        }
```

```
		}
		if (cont == r.length) {
			return true;
		} else {
			return false;
		}
	}
	/** Mostrar cada iteração */
	private void mostrarIteracao(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4) {
		System.out.println();
		System.out.println("Tabela Resultante da " + cont + "ª Iteração:");
		System.out.println(Arrays.toString(r1));
		System.out.println(Arrays.toString(r2));
		System.out.println(Arrays.toString(r3));
		System.out.println(Arrays.toString(r4));
		System.out.println(Arrays.toString(Z));
	}
	/** Formatar para uma casa decimal */
	private double[] formatarDecimaisUmaCasa(double[] vetor) {
		double[] novo = new double[vetor.length];
		DecimalFormat umaCasa = new DecimalFormat("0.0");
		for (int i = 0; i < vetor.length; i++) {
			String v = umaCasa.format(vetor[i]);
			String[] part1 = v.split("[,]");
			String v1 = part1[0] + "." + part1[1];
			double valor1 = Double.parseDouble(v1);
			novo[i] = valor1;
		}
		return novo;
	}
	/** Verifica se a solução foi encontrada*/
	public boolean chegouNoOtimo(double[] max) {
		for (int i = 0; i < max.length; i++) {
			if (max[i] > 0) {
				return false;
			}
		}
		return true;
	}
	/** Exibe tabela inicial*/
	private void mostrarQuadroInicial(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4) {
		if (cont == 0) {
			System.out.println("Tabela Inicial:");
			System.out.println(Arrays.toString(r1));
			System.out.println(Arrays.toString(r2));
			System.out.println(Arrays.toString(r3));
			System.out.println(Arrays.toString(r4));
			System.out.println(Arrays.toString(Z));
		}
	}
	/** Escolhe a coluna da tabela*/
	public int colunaMaiorValorPositivo(double[] v) {
		double maior = Integer.MIN_VALUE;
		int coluna = 0;
		for (int i = 0; i < v.length; i++) {
			if (v[i] > maior && v[i] > 0) {
				maior = v[i];
				coluna = i + 1;
```

```
                }
        }
        return coluna;
}
/** Esolhe a linha da tabela*/
public double linhaMaisRestrtitiva(double[] v, int i) {
        double valor = 0;
        if (v[(int) i] >= 0) {
                double s = v[v.length - 1];
                double r = v[(int) i];
                if (r == 0) {
                        return -1;
                } else {
                        valor = s / r;
                        return valor;
                }
        } else {
                return -1;
        }
}
public double[] inserir(double valor1, double valor2, double valor3, double valor4) {
        double vo1 = formatarDecimais(valor1);
        double vo2 = formatarDecimais(valor2);
        double vo3 = formatarDecimais(valor3);
        double vo4 = formatarDecimais(valor4);
        double[] v = new double[4];
        v[0] = vo1;
        v[1] = vo2;
        v[2] = vo3;
        v[3] = vo4;
        return v;
}
private double formatarDecimais(double valor) {
        DecimalFormat umaCasa = new DecimalFormat("0.0000000000");
        String v = umaCasa.format(valor);
        String[] part1 = v.split("[,]");
        String v1 = part1[0] + "." + part1[1];
        double valor1 = Double.parseDouble(v1);
        return valor1;
}
public int linhaDoMenorValorRestritivoPositivo(double[] v) {
        double menor = Integer.MAX_VALUE;
        int linha = 0;
        for (int i = 0; i < v.length; i++) {
                if (v[i] < menor && v[i] >= 0) {
                        menor = v[i];
                        linha = i + 1;
                }
        }
        return linha;
}
public double[] linhaEscolhida(int linha, double[] r1, double[] r2, double[] r3, double[] r4) {
        if (linha == 1) {
                return r1;
        } else if (linha == 2) {
                return r2;
        } else if (linha == 3){
                return r3;
```

```
                }else {
                        return r4;
                }
        }
        /** Definir valor para zerar coluna*/
        private double calcularValorEscalonamento(double interseccao, double d) {
                double vo1 = formatarDecimais(interseccao);
                double valor;
                valor = (d / vo1);
                valor = (valor * -1);
                double val1 = formatarDecimais(valor);
                return val1;
        }
        /** Realiza escalonamento para atualização da tabela*/
        public double[] zerarColuna(double[] vr, double[] vz, double valor) {
                double[] novoV = new double[vz.length];
                int cont = 0;
                for (int i = 0; i < vr.length; i++) {
                        if (vr[i] == vz[i]) {
                                cont += 1;
                        }
                }
                atualizarValores(vr, vz, valor, novoV);
                int quantP = vr.length;
                if (cont == quantP) {
                        return vr;
                } else {
                        return novoV;
                }
        }
        private void atualizarValores(double[] vr, double[] vz, double valor, double[] novoV) {
                for (int i = 0; i < vr.length; i++) {
                        double vIndice = ((vr[i] * valor) + vz[i]);
                        double vo1 = formatarDecimais(vIndice);
                        novoV[i] = vo1;
                }
        }
        /** Recebe dados da tabela*/
        public void entradaDados() {
                Maximizar4R problema = new Maximizar4R();
                int tamanho;
                String aux;
                int cont = 0;
                aux = JOptionPane.showInputDialog(null, "Quantas variáveis o problema apresenta?
(Inclua também as variáveis de folga e o termo independente b)");
                tamanho = Integer.parseInt(aux);
                double[] Z = valoresDeZ(tamanho, cont);
                double[] restricao1 = valoresR1(tamanho);
                double[] restricao2 = valoresR2(tamanho);
                double[] restricao3 = valoresR3(tamanho);
                double[] restricao4 = valoresR4(tamanho);
                double[] maxZ = problema.max4R(Z, restricao1, restricao2, restricao3, restricao4);
                double solucaoOtima = maxZ[maxZ.length - 1] * -1;
                System.out.println();
                System.out.println("O valor da sólução ótima é: " + solucaoOtima);
        }
        /** Recebe valores para a primeira restrição*/
        private double[] valoresR1(int tamanho) {
```

```
		int restricao = 1;
		double[] restricao1 = adicionarVariaveis(tamanho, restricao);
		return restricao1;
	}
	/** Recebe valores para a segunda restrição*/
	private double[] valoresR2(int tamanho) {
		int restricao = 2;
		double[] restricao2 = adicionarVariaveis(tamanho, restricao);
		return restricao2;
	}
	/** Recebe valores para a terceira restrição*/
	private double[] valoresR3(int tamanho) {
		int restricao = 3;
		double[] restricao3 = adicionarVariaveis(tamanho, restricao);
		return restricao3;
	}
	/** Recebe valores para a quarta restrição*/
	private double[] valoresR4(int tamanho) {
		int restricao = 4;
		double[] restricao4 = adicionarVariaveis(tamanho, restricao);
		return restricao4;
	}
	/** Adiciona os valores*/
	private double[] adicionarVariaveis(int tamanho, int restri) {
		String aux;
		int cont = 0;
		double[] restricao = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null,"Para a restrição " + restri +
", digite o valor do termo independente b"+restri);
				restricao[i] = Double.parseDouble(aux);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a restrição" + restri +
", digite o valor de X" + cont);
				restricao[i] = Double.parseDouble(aux);
			}
		}
		return restricao;
	}
	/** Recebe dados da função objetivo Z*/
	private double[] valoresDeZ(int tamanho, int cont) {
		String aux;
		double[] Z = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor do termo independente");
				Z[i] = Double.parseDouble(aux);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor de X" + cont);
				Z[i] = Double.parseDouble(aux);
			}
		}
		return Z;
```

```
	}
}
```

Classe: Maximizar5R

```
package br.ufpb.dcx.tcc.pl;
import java.text.DecimalFormat;
import java.util.Arrays;
import javax.swing.JOptionPane;
/**
 *
 * Classe concreta para maximizar uma função obejetivo Z, com cinto restrição
 *
 * @author Wellerson Alex
 *
 */
public class Maximizar5R extends ProgramacaoLinear {
	/** Propriedade de contagem*/
	private int cont;
	/** Propriedade que guarda os valores iniciais de Z */
	private double[] guardarInicialZ;
	/** Propriedade que guarda os valores iniciais da restrição 1 */
	private double[] guardarInicialR1;
	/** Propriedade que guarda os valores iniciais da restrição 2 */
	private double[] guardarInicialR2;
	/** Propriedade que guarda os valores iniciais da restrição 3 */
	private double[] guardarInicialR3;
	/** Propriedade que guarda os valores iniciais da restrição 4 */
	private double[] guardarInicialR4;
	/** Propriedade que guarda os valores iniciais da restrição 5 */
	private double[] guardarInicialR5;
	/**
	 * Encotra a solução ótima para a função Z de forma recursiva.
	 *
	 *@return Tabela inicial, tabelas atualizadas e solução ótima.
	 */
	public double[] max5R(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4, double[] r5)
{
		if (cont == 0) {
			guardarInicialR1 = new double[r1.length];
			guardarInicialR2 = new double[r2.length];
			guardarInicialR3 = new double[r3.length];
			guardarInicialR4 = new double[r4.length];
			guardarInicialR5 = new double[r5.length];
			guardarInicialZ = new double[Z.length];
			guardarInicialR1 = r1;
			guardarInicialR2 = r2;
			guardarInicialR3 = r3;
			guardarInicialR4 = r4;
			guardarInicialR5 = r5;
			guardarInicialZ = Z;
		}
		verificarLoopTableau(Z, r1, r2, r3, r4, r5, guardarInicialR1,
guardarInicialR2,guardarInicialR3, guardarInicialR4, guardarInicialR5, guardarInicialZ);
		if (chegouNoOtimo(Z)) {
			return Z;
		}
		mostrarQuadroInicial(Z, r1, r2, r3, r4, r5);
		cont += 1;
```

```
                int posicaoColuna = colunaMaiorValorPositivo(Z);
                double valorRestritivo1 = linhaMaisRestrtitiva(r1, posicaoColuna - 1);
                double valorRestritivo2 = linhaMaisRestrtitiva(r2, posicaoColuna - 1);
                double valorRestritivo3 = linhaMaisRestrtitiva(r3, posicaoColuna - 1);
                double valorRestritivo4 = linhaMaisRestrtitiva(r4, posicaoColuna - 1);
                double valorRestritivo5 = linhaMaisRestrtitiva(r5, posicaoColuna - 1);
                double[] valoresRestritivos = inserir(valorRestritivo1, valorRestritivo2,
valorRestritivo3, valorRestritivo4, valorRestritivo5);
                int posicaoLinha = linhaDoMenorValorRestritivoPositivo(valoresRestritivos);
                double[] linhaFixa = linhaEscolhida(posicaoLinha, r1, r2, r3, r4, r5);
                double interseccao = linhaFixa[posicaoColuna - 1];
                double valorZerarColuna1 = calcularValorEscalonamento(interseccao,
r1[posicaoColuna - 1]);
                double valorZerarColuna2 = calcularValorEscalonamento(interseccao,
r2[posicaoColuna - 1]);
                double valorZerarColuna3 = calcularValorEscalonamento(interseccao,
r3[posicaoColuna - 1]);
                double valorZerarColuna4 = calcularValorEscalonamento(interseccao,
r4[posicaoColuna - 1]);
                double valorZerarColuna5 = calcularValorEscalonamento(interseccao,
r5[posicaoColuna - 1]);
                double valorZerarColunaZ = calcularValorEscalonamento(interseccao,
Z[posicaoColuna - 1]);
                r1 = zerarColuna(linhaFixa, r1, valorZerarColuna1);
                r2 = zerarColuna(linhaFixa, r2, valorZerarColuna2);
                r3 = zerarColuna(linhaFixa, r3, valorZerarColuna3);
                r4 = zerarColuna(linhaFixa, r4, valorZerarColuna4);
                r5 = zerarColuna(linhaFixa, r5, valorZerarColuna5);
                Z = zerarColuna(linhaFixa, Z, valorZerarColunaZ);
                System.out.println();
                System.out.println("A coluna escolhida foi: " + posicaoColuna);
                if(posicaoLinha == 0) {
                        JOptionPane.showMessageDialog(null, "O problema é ilimitado", "Ilimitado",
JOptionPane.ERROR_MESSAGE);
                        throw new IllegalArgumentException("Problema Ilimitado");
                }
                System.out.println("A linha escolhida foi: " + posicaoLinha);
                System.out.println();
                System.out.println("Valores restritivos: " + Arrays.toString(valoresRestritivos));
                mostrarIteracao(Z, r1, r2, r3, r4, r5);
                return max5R(Z, r1, r2, r3, r4, r5);
        }
        /** Verifica se entrou em Loop */
        private void verificarLoopTableau(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4,
double[] r5, double[] guardarInicialR1, double[] guardarInicialR2, double[] guardarInicialR3, double[]
guardarInicialR4, double[] guardarInicialR5, double[] guardarInicialZ) {
                if (cont > 0) {
                        boolean testeR1 = testeLoop(guardarInicialR1, r1);
                        boolean testeR2 = testeLoop(guardarInicialR2, r2);
                        boolean testeR3 = testeLoop(guardarInicialR2, r3);
                        boolean testeR4 = testeLoop(guardarInicialR2, r4);
                        boolean testeR5 = testeLoop(guardarInicialR2, r5);
                        boolean testeZ = testeLoop(guardarInicialZ, Z);
                        if (testeR1 && testeR2 && testeR3 && testeR4 && testeR5 && testeZ) {
                                JOptionPane.showMessageDialog(null, "O problema entrou em loop,
não foi possível resolver", "Loop Tableau", JOptionPane.ERROR_MESSAGE);
                                System.out.println();
```

```
                                System.out.println("Os valores são iguais nas duas tabelas, por isso
o Loop é gerado");
                                System.out.println();
                                mostrarQuadroInicialLoop(Z, r1, r2, r3, r4, r5);
                                mostrarIteracao(Z, r1, r2, r3, r4, r5);
                                throw new IllegalArgumentException("Loop Tableau");
                        }
                }
        }
        /** Exibe tabela inicial quando há Loop */
        private void mostrarQuadroInicialLoop(double[] Z, double[] r1, double[] r2, double[] r3,
double[] r4, double[] r5) {
                System.out.println("Tabela Inicial:");
                r1 = formatarDecimaisUmaCasa(r1);
                r2 = formatarDecimaisUmaCasa(r2);
                r3 = formatarDecimaisUmaCasa(r3);
                r4 = formatarDecimaisUmaCasa(r4);
                r5 = formatarDecimaisUmaCasa(r5);
                Z = formatarDecimaisUmaCasa(Z);
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(r2));
                System.out.println(Arrays.toString(r3));
                System.out.println(Arrays.toString(r4));
                System.out.println(Arrays.toString(r5));
                System.out.println(Arrays.toString(Z));
                System.out.println();
        }
        /** Testa se as tabelas são iguais */
        private boolean testeLoop(double[] inicial, double[] r) {
                double[] r1Formatado = formatarDecimaisUmaCasa(r);
                double[] inicialFormatodo = formatarDecimaisUmaCasa(inicial);
                int cont = 0;
                for (int i = 0; i < r1Formatado.length; i++) {
                        if (inicialFormatodo[i] == r1Formatado[i]) {
                                cont += 1;
                        }
                }
                if (cont == r.length) {
                        return true;
                } else {
                        return false;
                }
        }
        /** Mostrar cada iteração */
        private void mostrarIteracao(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4,
double[] r5) {
                System.out.println();
                System.out.println("Tabela Resultante da " + cont + "ª Iteração:");
                System.out.println(Arrays.toString(r1));
                System.out.println(Arrays.toString(r2));
                System.out.println(Arrays.toString(r3));
                System.out.println(Arrays.toString(r4));
                System.out.println(Arrays.toString(r5));
                System.out.println(Arrays.toString(Z));
        }
        /** Formatar para uma casa decimal */
        private double[] formatarDecimaisUmaCasa(double[] vetor) {
                double[] novo = new double[vetor.length];
```

```
		DecimalFormat umaCasa = new DecimalFormat("0.0");
		for (int i = 0; i < vetor.length; i++) {
			String v = umaCasa.format(vetor[i]);
			String[] part1 = v.split("[,]");
			String v1 = part1[0] + "." + part1[1];
			double valor1 = Double.parseDouble(v1);
			novo[i] = valor1;
		}
		return novo;
	}
	/** Verifica se a solução foi encontrada*/
	public boolean chegouNoOtimo(double[] max) {
		for (int i = 0; i < max.length; i++) {
			if (max[i] > 0) {
				return false;
			}
		}
		return true;
	}
	/** Exibe tabela inicial*/
	private void mostrarQuadroInicial(double[] Z, double[] r1, double[] r2, double[] r3, double[] r4,
double[] r5) {
		if (cont == 0) {
			System.out.println("Tabela Inicial:");
			System.out.println(Arrays.toString(r1));
			System.out.println(Arrays.toString(r2));
			System.out.println(Arrays.toString(r3));
			System.out.println(Arrays.toString(r4));
			System.out.println(Arrays.toString(r5));
			System.out.println(Arrays.toString(Z));
		}
	}
	/** Escolhe a coluna da tabela*/
	public int colunaMaiorValorPositivo(double[] v) {
		double maior = Integer.MIN_VALUE;
		int coluna = 0;
		for (int i = 0; i < v.length; i++) {
			if (v[i] > maior && v[i] > 0) {
				maior = v[i];
				coluna = i + 1;
			}
		}
		return coluna;
	}
	/** Esolhe a linha da tabela*/
	public double linhaMaisRestrtitiva(double[] v, int i) {
		double valor = 0;
		if (v[(int) i] >= 0) {
			double s = v[v.length - 1];
			double r = v[(int) i];
			if (r == 0) {
				return -1;
			} else {
				valor = s / r;
				return valor;
			}
		} else {
			return -1;
```

```
		}
	}
	public double[] inserir(double valor1, double valor2, double valor3, double valor4, double
valor5) {
		double vo1 = formatarDecimais(valor1);
		double vo2 = formatarDecimais(valor2);
		double vo3 = formatarDecimais(valor3);
		double vo4 = formatarDecimais(valor4);
		double vo5 = formatarDecimais(valor5);
		double[] v = new double[5];
		v[0] = vo1;
		v[1] = vo2;
		v[2] = vo3;
		v[3] = vo4;
		v[4] = vo5;
		return v;
	}
	private double formatarDecimais(double valor) {
		DecimalFormat umaCasa = new DecimalFormat("0.00000000000");
		String v = umaCasa.format(valor);
		String[] part1 = v.split("[,]");
		String v1 = part1[0] + "." + part1[1];
		double valor1 = Double.parseDouble(v1);
		return valor1;
	}
	public int linhaDoMenorValorRestritivoPositivo(double[] v) {
		double menor = Integer.MAX_VALUE;
		int linha = 0;
		for (int i = 0; i < v.length; i++) {
			if (v[i] < menor && v[i] >= 0) {
				menor = v[i];
				linha = i + 1;
			}
		}
		return linha;
	}
	public double[] linhaEscolhida(int linha, double[] r1, double[] r2, double[] r3, double[] r4,
double[] r5) {
		if (linha == 1) {
			return r1;
		} else if (linha == 2) {
			return r2;
		} else if (linha == 3){
			return r3;
		}else if(linha == 4){
			return r4;
		}else {
			return r5;
		}
	}
	/** Definir valor para zerar coluna*/
	private double calcularValorEscalonamento(double interseccao, double d) {
		double vo1 = formatarDecimais(interseccao);
		double valor;
		valor = (d / vo1);
		valor = (valor * -1);
		double val1 = formatarDecimais(valor);
		return val1;
```

```
	}
	/** Realiza escalonamento para atualização da tabela*/
	public double[] zerarColuna(double[] vr, double[] vz, double valor) {
		double[] novoV = new double[vz.length];
		int cont = 0;
		for (int i = 0; i < vr.length; i++) {
			if (vr[i] == vz[i]) {
				cont += 1;
			}
		}
		atualizarValores(vr, vz, valor, novoV);
		int quantP = vr.length;
		if (cont == quantP) {
			return vr;
		} else {
			return novoV;
		}
	}
	private void atualizarValores(double[] vr, double[] vz, double valor, double[] novoV) {
		for (int i = 0; i < vr.length; i++) {
			double vIndice = ((vr[i] * valor) + vz[i]);
			double vo1 = formatarDecimais(vIndice);
			novoV[i] = vo1;
		}
	}
	/** Recebe dados da tabela*/
	public void entradaDados() {
		Maximizar5R problema = new Maximizar5R();
		int tamanho;
		String aux;
		int cont = 0;
		aux = JOptionPane.showInputDialog(null, "Quantas variáveis o problema apresenta?
(Inclua também as variáveis de folga e o termo independente b)");
		tamanho = Integer.parseInt(aux);
		double[] Z = valoresDeZ(tamanho, cont);
		double[] restricao1 = valoresR1(tamanho);
		double[] restricao2 = valoresR2(tamanho);
		double[] restricao3 = valoresR3(tamanho);
		double[] restricao4 = valoresR4(tamanho);
		double[] restricao5 = valoresR5(tamanho);
		double[] maxZ = problema.max5R(Z, restricao1, restricao2, restricao3, restricao4,
restricao5);
		double solucaoOtima = maxZ[maxZ.length - 1] * -1;
		System.out.println();
		System.out.println("O valor da sólução ótima é: " + solucaoOtima);
	}
	/** Recebe valores para a primeira restrição*/
	private double[] valoresR1(int tamanho) {
		int restricao = 1;
		double[] restricao1 = adicionarVariaveis(tamanho, restricao);
		return restricao1;
	}
	/** Recebe valores para a segunda restrição*/
	private double[] valoresR2(int tamanho) {
		int restricao = 2;
		double[] restricao2 = adicionarVariaveis(tamanho, restricao);
		return restricao2;
	}
```

```
	/** Recebe valores para a terceira restrição*/
	private double[] valoresR3(int tamanho) {
		int restricao = 3;
		double[] restricao3 = adicionarVariaveis(tamanho, restricao);
		return restricao3;
	}
	/** Recebe valores para a quarta restrição*/
	private double[] valoresR4(int tamanho) {
		int restricao = 4;
		double[] restricao4 = adicionarVariaveis(tamanho, restricao);
		return restricao4;
	}
	/** Recebe valores para a quinta restrição*/
	private double[] valoresR5(int tamanho) {
		int restricao = 5;
		double[] restricao5 = adicionarVariaveis(tamanho, restricao);
		return restricao5;
	}
	/** Adiciona os valores*/
	private double[] adicionarVariaveis(int tamanho, int restri) {
		String aux;
		int posicao = 0;
		int cont = 0;
		double[] restricao = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null,"Para a restrição " + restri +
", digite o valor do termo independente b"+restri);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao, i);
			} else {
				aux = JOptionPane.showInputDialog(null, "Para a restrição " + restri
+ ", digite o valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, restricao, i);
			}
		}
		return restricao;
	}
	/** Recebe dados da função objetivo Z*/
	private double[] valoresDeZ(int tamanho, int cont) {
		int posicao = 0;
		String aux;
		double[] Z = new double[tamanho];
		for (int i = 0; i < tamanho; i++) {
			cont += 1;
			if (cont == tamanho) {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor do termo independente");
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);

			} else {
				aux = JOptionPane.showInputDialog(null, "Para a função Z, digite o
valor de X" + cont);
				posicao = posicaoBarra(aux, posicao);
				verificarFracao(aux, posicao, Z, i);
```

```
                        }
                }
                return Z;
        }
        /** Trata caso com valor fracionário */
        private int posicaoBarra(String aux, int posicao) {
                posicao = aux.indexOf("/");
                return posicao;
        }
        /** Realiza operações com fração */
        private void verificarFracao(String aux, int posicao, double[] Z, int i) {
                if (posicao > 0) {
                        double numerador = Double.parseDouble(aux.substring(0, posicao));
                        double denominador = Double.parseDouble(aux.substring(posicao + 1,
aux.length()));
                        Z[i] = (numerador / denominador);
                } else {
                        Z[i] = Double.parseDouble(aux);
                }
        }
}
```

Classe: ProgramacaoLinear

```
package br.ufpb.dcx.tcc.pl;
/**
 *
 * Classe concreta Programação Linear, define o tipo de problema
   que será maximizado
 *
 * @author Wellerson Alex
 *
 */
public class ProgramacaoLinear {
        /** Problema com uma restrição*/
        public TipoProblemaPL maximizar1R() {
                return TipoProblemaPL.Maximizar1R;
        }
        /** Problema com duas restrições*/
        public TipoProblemaPL maximizar2R() {
                return TipoProblemaPL.Maximizar2R;
        }
        /** Problema com três restrições*/
        public TipoProblemaPL maximizar3R() {
                return TipoProblemaPL.Maximizar3R;
        }
        /** Problema com quatro restrições*/
        public TipoProblemaPL maximizar4R() {
                return TipoProblemaPL.Maximizar4R;
        }
        /** Problema com cinco restrições*/
        public TipoProblemaPL maximizar5R() {
                return TipoProblemaPL.Maximizar5R;
        }
        /** Cada classe implementa seu prórpio método desse tipo*/
        public void entradaDados() {
        }
}
```

Classe: TipoDeProblemaPL

```
package br.ufpb.dcx.tcc.pl;
/**
 *
 * Classe concreta de Enumeração, onde define os possíveis tipos de problemas de maximização.
 *
 * @author Wellerson Alex
 *
 */
public enum TipoProblemaPL {
	Maximizar1R, Maximizar2R, Maximizar3R, Maximizar4R, Maximizar5R
}
```