Trabalho Final - Implementação do Simplex em C

Otimização 2025.1

Pedro Saito
122149392

Marcos Silva
122133854

Milton Salgado
122169279

Resumo

Este texto foi apresentado como relatório do trabalho final da disciplina de Otimização, oferecida pelo Instituto de Computação da UFRJ no primeiro semestre de 2025. O objetivo deste trabalho é implementar o algoritmo simplex na linguagem de programação C. Um problema de otimização linear será modelado e solucionado com a implementação desenvolvida, e os resultados computacionais serão reportados, incluindo tempo de solução, número de iterações realizadas e valor da função objetivo. O relatório detalhará a arquitetura do código em C, as escolhas de implementação, as instâncias de teste selecionadas e os critérios de avaliação adotados.

1. Introdução

Para compreender as origens do método Simplex, é preciso regressar ao contexto da Segunda Guerra Mundial, quando o cientista matemático George Dantzig, então trabalhando em métodos de planejamento para a Força Aérea dos EUA, passou a formular seus problemas como sistemas de desigualdades lineares. Inspirado pelas matrizes de insumo-produto de Wassily Leontief, Dantzig inicialmente não incluiu nelas uma função objetivo, o que permitia infinitas soluções viáveis e exigia um conjunto de "regras básicas" militares para guiar a escolha de um plano. Somente em meados de 1947, ao perceber que essas regras podiam ser traduzidas em uma única função linear a ser maximizada, o problema tornou-se matematicamente tratável e deu origem ao algoritmo que hoje conhecemos como Simplex.

O método evoluiu gradualmente ao longo de cerca de um ano, tempo em que Dantzig aprimorou sua formulação, incorporou o conceito de função objetivo, baseado em sua tese de doutorado sobre multiplicadores de Lagrange para programas lineares em variáveis contínuas, e comprovou a eficácia prática do esquema de colunas que viria a constituir o cerne do Simplex.

2. Organização

Este documento está estruturado da seguinte forma:

1. Na Seção 3 apresentamos as três instâncias avaliadas (viável, ilimitado e inviável).
2. Nesta mesma seção descrevemos a formulação matemática do problema de transporte e definições teóricas do Simplex.
3. Na Seção 4 detalhamos a implementação em C do método Simplex de duas fases.
4. Na Seção 5 exibimos os resultados computacionais, incluindo os tempos de execução e número de iterações.
5. Por fim, na Seção 6 discutimos conclusões e direções para trabalhos futuros.

3. Problemas

Para avaliar sistematicamente nosso algoritmo de duas fases, selecionamos três instâncias de programação linear, cada uma ilustrando um cenário distinto:

- **Viável e limitado**: O clássico problema de transporte, que possui solução ótima finita.
- **Ilimitado**: Exemplo no qual a função objetivo cresce sem restrições.
- **Inviável**: Problema no qual nenhuma solução satisfaz simultaneamente todas as restrições.

3.1. Problema 1 (Viável e Limitado)

Neste trabalho, escolheu-se como estudo de caso o *Problema de Transporte* aplicado a produtos organizados por categorias, inspirado na situação real enfrentada pela empresa *Protecter & Gamble* (P&G). O objetivo é ilustrar

Enunciado: A Procter & Gamble (P&G) fabrica e comercializa mais de 300 marcas de bens de consumo em todo o mundo. A empresa tem crescido continuamente ao longo de sua longa história, que remonta à década de 1830. Para manter e acelerar esse crescimento, foi realizado um importante estudo de Pesquisa Operacional (OR) com o objetivo de fortalecer a eficácia global da P&G.

Antes do estudo, a cadeia de suprimentos da empresa era composta por centenas de fornecedores, mais de 50 categorias de produtos, mais de 60 fábricas, 15 centros de distribuição e mais de 1.000 zonas de clientes. No entanto, à medida que a empresa avançava rumo à consolidação de marcas globais, a gestão percebeu a necessidade de consolidar fábricas para reduzir os custos de fabricação, melhorar a velocidade de chegada ao mercado e diminuir os investimentos em capital. Por isso, o estudo se concentrou no redesenho do sistema de produção e distribuição da empresa para suas operações na América do Norte. O resultado foi uma redução de quase 20% no número de fábricas na América do Norte, gerando uma economia superior a 200 milhões de dólares em custos antes dos impostos por ano.

Uma parte essencial do estudo envolveu a formulação e resolução de problemas de transporte para categorias individuais de produtos. Para cada opção em relação às fábricas que deveriam permanecer abertas, e assim por diante, resolver o problema de transporte correspondente a uma categoria de produto mostrava qual seria o custo de distribuição para o envio daquela categoria das fábricas aos centros de distribuição e zonas de clientes.

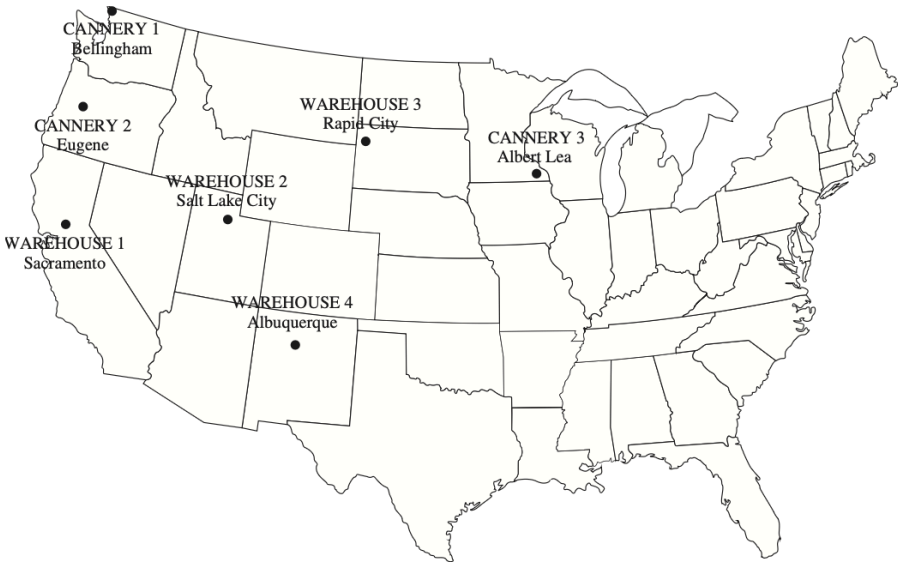

Figura 1: *Localização das fábricas de conservas e armazéns no problema da empresa P & T.*

	Custo de Envio (US$) por Carga de Caminhão				
	Depósito				
Conservas	**1**	**2**	**3**	**4**	**Produção**
1	464	513	654	867	75
2	352	416	690	791	125
3	995	682	388	685	100
Demanda	80	65	70	85	

Tabela 1: *Custos de transporte e demanda para a empresa P & T.*

3.2. Modelagem

A partir dos dados apresentados na Tabela 1, podemos modelar o problema de transporte da empresa *P & T* como um problema clássico de Programação Linear. O objetivo consiste em determinar a quantidade de unidades a serem transportadas de cada fábrica de conservaspara cada armazémde modo a **minimizar o custo total de transporte**, respeitando as capacidades de produção das fábricas e as demandas dos armazéns.

Para isso, define-se:

- x_{ij} : Número de cargas transportadas da conserva i para o depósito j.
- c_{ij} : Custo de transporte por carga entre a conserva i e o depósito j.
- s_i : Capacidade de produção da fábrica i.
- d_j : Demanda do armazém j.

A formulação teórica do problema é dada por:

$$\text{Minimize } f(\mathbf{x}) = \sum_{i=1}^{3}\sum_{j=1}^{4} c_{ij} \cdot x_{ij}$$

$$\text{Sujeito a} \sum_{j=1}^{4} x_{ij} \leq s_i \quad \text{para } i = 1,2,3 \text{ (restrições de oferta)}$$

$$\sum_{i=1}^{3} x_{ij} \geq d_j \quad \text{para } j = 1,2,3,4 \text{ (restrições de demanda)}$$

$$x_{ij} \geq 0 \quad \text{para todos } i, j$$

Por outro lado, a formulação numérica do problema está dada abaixo:

$$\begin{aligned}
\text{Minimize } f(\mathbf{x}) = {} & 464x_{11} + 513x_{12} + 654x_{13} + 867x_{14} + 352x_{21} + 416x_{22} + 690x_{23} + 791x_{24} + 995x_{31} + \\
& 682x_{32} + 388x_{33} + 685x_{34} \\
\text{Sujeito a} \quad & x_{11} + x_{12} + x_{13} + x_{14} = 75 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 125 \\
& x_{31} + x_{32} + x_{33} + x_{34} = 100 \\
& x_{11} + x_{21} + x_{31} = 80 \\
& x_{12} + x_{22} + x_{32} = 65 \\
& x_{13} + x_{23} + x_{33} = 70 \\
& x_{14} + x_{24} + x_{34} = 85 \\
& x_{ij} \geq 0 \quad \text{para todos } i, j
\end{aligned}$$

3.3. Problema 2 (Ilimitado)

Selecionamos como problema com solução ilimitada um exemplo simples como abaixo, para ilustrar o funcionamento do algoritmo para esse caso:

Maximize $f(\mathbf{x}) = x_1 + x_2$
Sujeito a $x_1 - x_2 \leq 1$
$x_1, x_2 \geq 0$

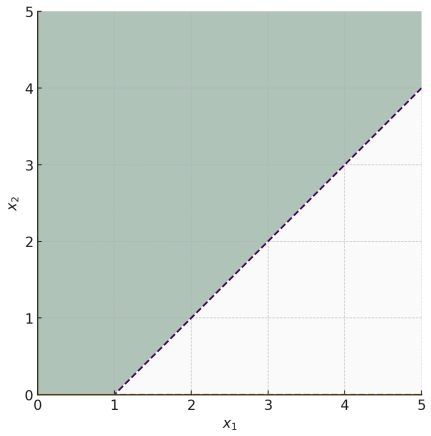

Figura 2: *A ausência de limites superiores para x_1 e x_2 permite que a função objetivo cresça indefinidamente.*

3.4. Problema 3 (Inviável)

Para ilustrar o caso inviável, escolhemos o **Exemplo 13 da Aula 3** dos Slides, cujo enunciado é:

Minimize $f(\mathbf{x}) = x_1 + 2x_2$
Sujeito a $x_1 + x_2 \geq 3$
$2x_1 + x_2 \leq 2$
$x_1, x_2 \geq 0$

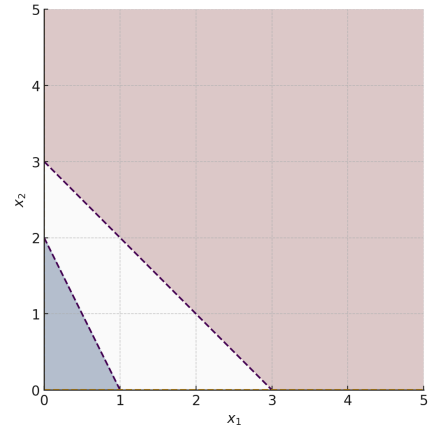

Figura 3: *Problema inviável: Não existe solução que satisfaça ambas restrições.*

3.5. Fundamento Teórico

A seguir, vamos formalizar os conceitos abordados em otimização linear.

Def. Problema de Otimização: Um problema de otimização linear consiste em encontrar, dentre todas as possíveis soluções que satisfazem um conjunto de restrições, aquela que minimiza ou maximiza uma dada função f que denotaremos de **função objetivo**.

- **Minimização**: Busca o valor mínimo de $f(\mathbf{x})$.
- **Maximização**: Busca o valor máximo de $f(\mathbf{x})$.

Def. Variáveis de decisão: Vetor de dimensão n, denotado por $\mathbf{x} = (x_1, ..., x_n)$ que descrevem as opções sobre as quais se decide. Em um problema de otimização, cada x_j representa um nível de recurso, quantidade produzida, investimento, dentre outras ...

Def. Função objetivo: Dada por

$$f(\mathbf{x}) = c_1x_1 + c_2x_2 + ... + c_nx_n = \mathbf{c^T x}$$

onde $\mathbf{c} = (c_1, ..., c_n)$ são os coeficientes que quantificam o impacto de cada variável de decisão no valor de f. O objetivo é encontrar $\mathbf{x}^*$ de modo que otimize (minimize ou maximize) $f(\mathbf{x})$.

Def. Restrições: Condições que limitam o conjunto de soluções possíveis. Comumente expressadas por inequações (forma canônica) ou igualdades (forma padrão). Em um problema de otimização linear com n variáveis e m restrições,

- **Igualdades lineares**:

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n = b_i, \quad i = 1, ..., m$$

- **Desigualdades lineares**:

$$a_{i1}x_1 + a_{i2}x_2 + ... + a_{in}x_n \leq b_i \text{ ou } \geq b_i$$

Cada a_{ij} é o coeficiente que relaciona a variável x_j à restrição i, e b_i é o limite disponível ou exigido.

Def. Região viável: Conjunto de todos os vetores $\mathbf{x}$ que satisfazem simultaneamente todas as restrições e as condições de não negatividade. Em programação linear, essa região forma um politopo convexo.

Def. Solução viável: Qualquer ponto, definido pelo vetor $\mathbf{x}$, pertencente à região viável. Nem toda solução viável é ótima, são apenas admissíveis.

Def. Solução Ótima: Solução viável $\mathbf{x}^*$ que alcança o valor ótimo (mínimo ou máximo) da função objetivo:

$$f(\mathbf{x}^*) = \min_{\mathbf{x} \in \text{ viável}} f(\mathbf{x}) \quad \text{ou} \quad \max_{\mathbf{x} \in \text{ viável}} f(\mathbf{x})$$

Def. Variáveis de Folga ou Excedentes e Artificiais: Uma variável de folga ou excedente corresponde a uma variável adicionada a uma restrição de desigualdade para transformá-la em uma restrição de igualdade. Uma restrição de não-negatividade também é adicionada a variável de folga. Em termos gerais:

- **Variáveis de folga**: Para converter inequações $\leq$ ou $\geq$ em igualdades, introduz-se $s_i \geq 0$ tal que $\sum_j a_{ij}x_j + s_i = b_i$.
- **Variáveis Artificiais**: Em certos casos, quando não podemos começar a partir de uma solução básica viável, acrescentamos uma variável artificial a_i para garantir viabilidade inicial, sendo removida no término da Fase I do Simplex no Método de Duas Fases.

Def. Base e Solução Básica Viável: Em um problema de otimização contendo m restrições, escolhe-se um subconjunto de m variáveis para compor a base. Inicialmente, essas variáveis correspondem às variáveis de folga, isto é, sem incluir aquelas presentes na função objetivo.

4. Experiências Computacionais

A implementação e resolução do modelo apresentado na Seção 2 foram realizadas em um computador pessoal com as seguintes configurações de *hardware* e *software*:

Setup	Sistema	CPU	GPU	RAM	Disco
1	macOs	M2 8 núcleos	Int. 10 núcleos	8 GB	460 GB
2	BigLinux	i7 - 14 núcleos	RTX 4050	16 GB	296 GB
3	Ubuntu	i5 - 6 núcleos	GTX 1660	16 GB	467 GB

Setup	Duração - PL 1	Duração - PL 2	Duração - PL 3
1	0.008850447s	0.008607634s	0.008699520s
2	0.003s	0.003s	0.0029s
3	0.00262s	0.002872s	0.002761s

Todos os experimentos foram conduzidos em ambiente local, com uso eficiente de recursos computacionais, respeitando os limites de memória física e de troca. A resolução do modelo foi realizada por meio de implementação própria do algoritmo Simplex em linguagem C, compilado e executado diretamente no sistema descrito acima.

4.1. Comparação com Gurobi

Para avaliar o desempenho de nossa implementação, comparamos os resultados obtidos pelo **Simplex em C** com o pacote Python `gurobipy`, executados nas mesmas configurações de hardware que o simplex anterior. Os indicadores de iterações, tempo de otimização e valor objetivo estão resumidos na Tabela.

Método	Iterações	Duração (s)	Valor Ótimo
Simplex em C	20 (FI: 10 e FII: 10)	0.000022	152535.00
Gurobi (Method=0)	4	0.004	152535.00

Como esperado, ambos os métodos retornaram exatamente a mesma alocação de variáveis básicas na solução ótima:

$$x_{1,2} = 20, x_{1,4} = 55, x_{2,1} = 80, x_{2,2} = 45, x_{3,3} = 70, x_{3,4} = 30.$$

5. Código

Nesta seção, descrevemos detalhadamente as principais funções implementadas para a execução do **algoritmo Simplex de duas fases**, responsáveis pelas operações essenciais do método. As funções estão organizadas em módulos conforme suas responsabilidades específicas: inicialização da estrutura **Simplex**, pivoteamento, execução das fases **I** e **II**, remoção de variáveis artificiais e liberação de recursos.

5.1. Estrutura Simplex

A estrutura `Simplex` representa o tableau do método Simplex de duas fases e contém todas as informações necessárias para resolver problemas de programação linear. Contém as seguintes variáveis:

Variável	Descrição
`t`	Tableau unidimensional de tamanho $(m+1) \times \text{col}$.
`m`	Número de restrições do problema.
`n`	Número de variáveis originais do problema.
`col`	Número total de colunas do tableau (incluindo RHS).
`tipo_pl`	Tipo do problema: 0 para minimização, 1 para maximização.
`ilimitado`	Flag indicando se o problema possui solução ilimitada.
`inviavel`	Flag indicando se o problema é inviável.
`basicas`	Vetor com os índices das variáveis básicas em cada linha.
`c_original`	Cópia da função objetivo para uso na Fase II.
`total_variaveis`	Número total de variáveis (excluindo coluna RHS).
`inicio_artificiais`	Índice onde começam as variáveis artificiais no tableau.
`num_artificiais`	Número de variáveis artificiais adicionadas.
`iteracoes_fase1`	Contador de iterações realizadas na Fase I.
`iteracoes_fase2`	Contador de iterações realizadas na Fase II.

5.2. Funções

Nesta seção, descrevemos detalhadamente as principais funções implementadas no simulador, responsáveis por executar as diversas operações essenciais ao funcionamento do escalonador Round-Robin com feedback. As funções estão organizadas em módulos conforme suas responsabilidades específicas, como gerenciamento de processos, interface com o usuário, operação do escalonador e utilitários auxiliares.

Iniciaremos a análise das funções de gerenciamento de processos, enfatizando as relacionadas à criação de processos.

1. `criar_simplex_duas_fases(double **A, double *b, double *c, int *tipo_restricao, int m, int n, int tipo_pl)`
    - Inicializa a estrutura `Simplex` para o método das duas fases, com variáveis auxiliares de acordo com a restrição.
    - Monta o tableau da Fase I e configura a função objetivo.
    - Retorna um ponteiro para a estrutura `Simplex` inicializada.
2. `pivotear(Simplex *s, int linha_pivo, int coluna_pivo)`
    - Executa a operação de **pivoteamento**, normalizando a linha e atualizando as demais.
    - Ajusta a variável básica da linha correspondente.
    - Não possui valor de retorno.
3. `executar_simplex(Simplex *s)`
    - Resolve o tableau atual pelo método Simplex.
    - Utiliza a Regra de Bland e o teste de razão mínima.
    - Retorna o número de iterações.
4. `preparar_fase2(Simplex *s)`
    - Remove variáveis artificiais e restaura função objetivo original.
    - Prepara o tableau para a Fase II.
    - Não possui valor de retorno.

5. `resolver_duas_fases(Simplex *s)`
    - Executa as duas fases do método Simplex.
    - Verifica viabilidade e otimiza a função original.
    - Exibe a solução final.
    - Não possui valor de retorno.
6. `liberar_simplex(Simplex *s)`
    - Libera toda a memória alocada para a estrutura `Simplex`.
    - Não possui valor de retorno.
7. `main(void)`
    - Define um problema de transporte com 7 restrições e 12 variáveis.
    - Coordena a criação, execução e liberação do modelo.
    - Exibe o problema e realiza a medição de tempo.
    - Retorna 1 em todos os casos.

5.3. Resultados

5.3.1. Resultados Computacionais

O problema de otimização linear foi resolvido com sucesso utilizando o algoritmo Simplex de duas fases implementado. A execução do algoritmo demonstrou excelente performance computacional e convergência eficiente para a solução ótima.

5.3.2. Análise do Desempenho Algorítmico

O algoritmo convergiu em **20 iterações totais**, distribuídas igualmente entre as duas fases:

- **Fase I**: 10 iterações para estabelecer viabilidade
- **Fase II**: 10 iterações para otimização

A distribuição das iterações nos informa que o problema possui estrutura bem condicionada, sem degeneração significativa ou dificuldades numéricas que poderiam prolongar a convergência.

5.3.3. Tempo de Execução

O **tempo total de execução foi em média de .000666 segundos**, demonstrando a eficiência computacional da implementação. Este desempenho temporal é excelente considerando:

- Dimensão do problema: 12 variáveis e 7 restrições
- Natureza do algoritmo: método exato que garante solução ótima global
- Complexidade das operações matriciais envolvidas no pivoteamento

5.3.4. Solução Ótima Encontrada

O algoritmo determinou que o **valor ótimo da função objetivo é 152.535**, correspondente ao custo mínimo total do sistema. A solução ótima apresenta a seguinte estrutura:

Variáveis básicas (não-nulas): $x_2 = 20, x_4 = 55, x_5 = 80, x_6 = 45, x_{11} = 70, x_{12} = 30$

Variáveis não-básicas (nulas): $x_1 = x_3 = x_7 = x_8 = x_9 = x_{10} = 0$

5.3.5. Interpretação da Solução

A alocação obtida revela a minimização do custo total sob as capacidades e demandas impostas:

1. **Uso prioritário de rotas de menor custo unitário**: As rotas com custo unitário mais baixo, isto é, $x_5(352)$ e $x_{11}(388)$ são usadas ao máximo $(80 \text{ e } 70)$, esgotando oferta e parte da demanda.

2. **Utiliza parcialmente variáveis de custo intermediário**: x_2, x_4, x_6 e x_{12} complementam a alocação necessária
3. **Evita completamente variáveis de alto custo**: x_7 (690), x_8 (791) e x_9 (995) permanecem nulas.

5.3.6. Verificação de Consistência

A solução satisfaz todas as restrições do problema:

- Restrições de demanda: 75, 125 e 100 unidades atendidas exatamente
- Restrições de oferta: capacidades de 80, 65, 70 e 85 respeitadas
- Condições de não-negatividade: todas as variáveis possuem valores não-negativos

5.3.7. Saída

5.3.7.1. Problema Original

```txt
Resolvendo problema de programacao linear:

Minimize 464x1 + 513x2 + 654x3 + 867x4 + 352x5 + 416x6 + 690x7 + 791x8 + 995x9 +
682x10 + 388x11 + 685x12

sujeito a:
x1 + x2 + x3 + x4 = 75
x5 + x6 + x7 + x8 = 125
x1 + x5 + x9 = 80
x2 + x6 + x10 = 65
x3 + x7 + x11 = 70
x4 + x8 + x12 = 85
x9 + x10 + x11 + x12 = 100
x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12 >= 0

=== EXECUTANDO ALGORITMO SIMPLEX DUAS FASES ===

=== RESULTADOS FINAIS ===

Numero de iteracoes:
  Fase I: 10 iteracoes
  Fase II: 10 iteracoes
  Total: 20 iteracoes

Status: Solucao otima encontrada
Valor otimo da funcao objetivo: 152535.00

Solucao completa:
Variaveis basicas:
  x2 = 20.000000
  x4 = 55.000000
  x5 = 80.000000
  x6 = 45.000000
```

```
  x11 = 70.000000
  x12 = 30.000000

Tempo de execucao: 0.000022 segundos
```

Resultados do Gurobi

```
Restricted license - for non-production use only - expires 2026-11-23
Set parameter Method to value 0
Gurobi Optimizer version 12.0.2 build v12.0.2rc0 (mac64[arm] - Darwin 24.5.0 24F74)

CPU model: Apple M2
Thread count: 8 physical cores, 8 logical processors, using up to 8 threads

Non-default parameters:
Method  0

Optimize a model with 7 rows, 12 columns and 24 nonzeros
Model fingerprint: 0xc923d8aa
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [4e+02, 1e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+01, 1e+02]
Presolve time: 0.00s
Presolved: 7 rows, 12 columns, 24 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    2.3621500e+05   1.150000e+02   6.001905e+06      0s
       4    1.5253500e+05   0.000000e+00   0.000000e+00      0s

Solved in 4 iterations and 0.00 seconds (0.00 work units)
Optimal objective  1.525350000e+05
Tempo de otimização: 0.004 s
Obj: 152535.00
x[1,2] = 20.00
x[1,4] = 55.00
x[2,1] = 80.00
x[2,2] = 45.00
x[3,3] = 70.00
x[3,4] = 30.00
```

5.3.7.2. Problema Ilimitado

```
Resolvendo problema de programacao linear:                    txt

Maximize 1x1 + 1x2
```

```
sujeito a:
x1 -x2 <= 1
x1, x2 >= 0

=== EXECUTANDO ALGORITMO SIMPLEX DUAS FASES ===

=== RESULTADOS FINAIS ===

Numero de iteracoes:
  Fase I: 0 iteracoes
  Fase II: 2 iteracoes
  Total: 2 iteracoes

Status: Solucao ilimitada

Tempo de execucao: 0.000008 segundos
```

5.3.7.3. Problema Inviável

txt

```
Resolvendo problema de programacao linear:

Minimize 1x1 + 2x2

sujeito a:
x1 + x2 >= 3
2*x1 + x2 <= 2
x1, x2 >= 0

=== EXECUTANDO ALGORITMO SIMPLEX DUAS FASES ===

RESULTADO: Problema inviavel (valor da Fase I = -1.000000)

Tempo de execucao: 0.000007 segundos
```

Bibliografia

1. Trick, M.: Linear Programming, (1998).
2. Camm, J.D., Chorman, T.E., Dill, F.A., Evans, J.R., Sweeney, D.J., Wegryn, G.W.: Blending OR/MS, Judgment, and GIS: Restructuring P & G's Supply Chain. Interfaces. 27, 128–142 (1997).
3. Hillier, F.S., Lieberman, G.J.: Introduction to Operations Research. McGraw-Hill, New York (2009).
4. Jardim, M.H.C.H.: Slides da disciplina de Otimização da UFRJ, (2025).

6. Conclusão

Os resultados computacionais mostram que a implementação do método Simplex de duas fases é ao mesmo tempo robusta e eficiente, garantindo a obtenção da solução ótima com tempo de processamento reduzido. A rápida convergência e a qualidade das soluções obtidas atestam tanto a correção do algoritmo quanto a adequação da abordagem metodológica a este tipo de problema.

Para tornar a ferramenta ainda mais versátil, o código será organizado no futuro em módulos independentes, de modo que novas instâncias de problemas possam ser inseridas de forma simples e rápida, sem a necessidade de modificações na lógica central do algoritmo como está sendo feito no momento. Além disso, será adicionada a possibilidade de leitura da PL diretamente da entrada padrão (`STDIN`) ou de um arquivo externo, simplificando o processo de teste e validação.