# Otimização Aplicada a Processos de Separação de Bens

**Eli Enrico Carnette**

Trabalho de Conclusão de Curso apresentado
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Bacharel em Matemática Aplicada e Computacional

Orientador: Prof. Dr. Julio Michael Stern

São Paulo, 19 de fevereiro de 2013

# Otimização Aplicada a Processos de Separação de Bens

Esta é a versão do trabalho de conclusão de curso
elaborado pelo candidato Eli Enrico Carnette,
submetido e corrigido pela banca avaliadora.

Banca avaliadora:

- Prof. Dr. Julio Michael Stern (orientador) - IME - USP
- Prof. Dr. Henrique von Dreifus - IME – USP
- Profa. Dra. Sônia Regina Leite Garcia - IME – USP

# Agradecimentos

Agradeço aos meus pais, José e Dalva, e minha noiva, Sandra por todo carinho, paciência e apoio dados a mim durante todos meus anos de estudo, principalmente a graduação.

Aos colegas de curso Guilherme Silva Salomão, Leonardo Windlin Cesar, Marco Alexandre Claudino, Pedro Ivo Miguel Avelino e Rafael de Moraes Pontilho, pela companhia e alegrias durante esses bons anos.

Ao professor Julio Michael Stern, pelos cursos ministrados e pelas valiosas orientações, tornando este trabalho possível de ser realizado .

Aos professores Luis Carlos de Castro Santos, Henrique von Dreifus, Manoel Marcilio Sanches, Maria Izabel Ramalho Martins, Paulo José Silva e Silva e Sônia Regina Leite Garcia cujos conhecimentos transmitidos em aula foram muito importantes para o meu crescimento acadêmico.

“A única coisa que nós sabemos é que nós nada sabemos. E isto é o maior vôo do conhecimento humano.”

Lev Tolstoi

# Resumo

Carnette, E.E. **Otimização Aplicada a Processos de Separação de Bens**. 2013. 64 f. Trabalho de Conclusão de Curso - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2013.

Otimização Aplicada a Processos de Separação de Bens trata de métodos que obtém uma divisão justa de bens entre parceiros em litígio.

A primeira parte do trabalho consiste em apresentar os conceitos de Programação Linear (PL) e a ferramenta computacional utilizada para resolver Problemas de Programação Linear (PPL), conhecida como Simplex, bem como diversas de suas variantes. Em seguida são abordadas Técnicas de Partilha (Cake-Cutting Algorithms), que tratam de divisão justa entre pessoas que tem diferentes opiniões sobre o que é *justo*. Neste último tópico é definido o que é *justo* do ponto de vista matemático. A construção desses tópicos segue as obras [Ste07] e [Rob98], unindo os temas e complementando com exemplos computacionais, tanto resultados quanto códigos.

**Palavras-chave:** Otimização Linear, Separação de Bens, Separação Judicial, Programação Linear, Cake-Cutting Algorithms, Modelagem Matemática.

# *Abstract*

*Carnette, E.E. **Applied Optimization to Division of Property Trials**. 2013. 64 p. Technical Report – Institute of Mathematics and Statistics, Universidade de São Paulo, São Paulo, Brazil, 2013.*

*Applied Optimization to Division of Property Trials is a technical report about methods that achieve a fair distribution of goods among litigating partners.*

*The first part of this work presents the concepts of Linear Programming and the computational tool used to solve Linear Programming Problems known as Simplex algorithm, as well as several of its variants. Then Cake-Cutting Algorithms are addressed, concerning fair division between partners that have different opinions on what is fair. The construction of this report follows [Ste07] and [Rob98], complemented with computational examples, codes and simulations.*

***Keywords:*** *Linear Optimization, Asset Division, Property Division, Judicial Separation, Linear Programming, Cake-Cutting Algorithms, Mathematical Modeling.*

# Sumário

# Capítulo 1 - Introdução

Dados históricos do IBGE nos mostram que a quantidade de divórcios concedidos no Brasil vem crescendo [IBGE10]. Em 1984, ano da primeira medição, foram observados 16.348 divórcios, enquanto os dados mais recentes de 2010 nos mostram 89.425 concessões.

Dependendo do regime de separação de bens, o divórcio pode trazer a tona um processo de separação de bens, no qual são colocados em disputa judicial os bens tangíveis do casal e cabe ao juíz fazer a divisão mais justa.

Entretanto, por não existir metodologia padrão para medir o quão justo a separação foi, não necessariamente ambas partes se sentem satisfeitas. Além disso, se fosse possível submeter o mesmo processo a mais de um juiz, provavelmente os resultados finais seriam diferentes.

O intúito deste trabalho é abordar dois temas: Otimização Linear e Técnicas de Partilha (Cake-Cutting Algorithms). O primeiro consiste em apresentar um problema que deve satisfazer uma série de restrições e em como obter, entre diversas soluções, a melhor possível, enquanto o segundo trata de como partilhar itens entre duas ou mais pessoas de forma justa. Neste último, defineremos do ponto de vista matemátco o que é ser justo.

Ao se mesclar os dois temas, é possível fazer a melhor divisão de bens possível, de forma que nenhuma das partes seja injustiçada. Vemos também que este método é capaz de resolver problemas similares, por exemplo, a disputa de herença entre parentes.

## 1.1 Pré-Requisitos

Durante a elaboração deste trabalho serão utilizadas as principais idéias apresentadas em cursos de Programação Linear. Além disso, trabalharemos conceitos de Técnicas de Particionamento. Como este conceito não é geralmente abordado durante os cursos de graduação, teremos um capítulo dedicado ao assunto. Desta forma, temos que tais pré-requisitos são aconselháveis, porém não indispensáveis para o entendimento do conteúdo apresentado.

## 1.2 Organização do Trabalho

Iniciaremos nosso estudo em otimização linear aplicada a processos de separação de bens, relembrando assuntos abordados nos cursos de matemática no ensino superior. No capítulo 2 retomamos os conceitos do Simplex, uma poderosa ferramenta matemática, e diversas de suas variantes. No capítulo seguinte são abordados conceitos usualmente não abordados na graduação, tais como técnicas de divisão justa, e nele também definiremos do ponto de vista matemático o que é uma divisão justa.

Já no quarto capítulo os temas dos dois capítulos anteriores serão unidos e através deles criaremos gradativamente um modelo robusto, capaz de realizar a divisão de bens que estamos procurando.

Por fim teremos as conclusões, seguidas dos códigos computacionais por mim elaborados em R, que permitirão que os interessados simulem os modelos apresentados ao longo do trabalho.

## Capítulo 2 – Otimização Linear – O algoritmo Simplex

Um problema de Otimização Linear consiste em buscar um vetor ótimo que satisfaça uma série de restrições. Estes problemas podem ser, por exemplo, encontrar os melhores caminhos dentro de uma rede de fluxos, planejar a rotação de culturas, determinar quantidades de produtos a serem criados por uma fábrica, ou mesmo planejar o deslocamento de grandes tropas militares.

Todos estes problemas, apesar de serem bem distintos, podem ser modelados na forma de restrições que precisam ser satisfeitas, como por exemplo, a necessidade de produzir no mínimo 10 peças de um tipo, ou plantar milho em não mais do que 50% do terreno. Como em geral essas restrições podem ser satisfeitas de diversas maneiras (afinal, posso plantar milho em 10%, 15%, 22%,...,50% do terreno), passamos então a nos interessar em obter o melhor "custo x benefício".

No exemplo da plantação, se em um determinado período eu plantar 50% de milho e o restante de outras culturas, terei um lucro estimado em X reais, enquanto se eu plantar 25% de milho e 75% de outras culturas, meu lucro estimado será Y reais, que pode ser maior, menor ou até igual a X.

Para esse caso, o problema de Otimização Linear trata de encontrar qual o melhor percentual de plantação de milho e de outras culturas que é necessário ter para determinado período, de forma a maximizar o lucro.

Isso, em termos matemáticos, se traduz por encontrar um vértice $\mathbf{x^*} \in R^n$ que satisfaça um sistema de restrições $\mathbf{Ax = b}$, onde **A** é uma matriz de coeficientes com **m** linhas (quantidade de restrições) e **n** colunas (quantidade de variáveis) e $\mathbf{b} \in R^m$, tal que **x*** é ótimo, ou seja, dado um vetor de coeficientes $\mathbf{c} \in R^n$, representando os custos unitários, $\mathbf{c'x^* \leq c'x}$ para todo **x** satisfazendo as restrições, onde **c**' é o vetor transposto de **c**.

Desta forma, podemos escrever um problema de Otimização Linear como:

$$\textit{Min } c'x$$
$$\textit{Sujeito a: } Ax = b$$

Abordaremos como resolver esse tipo de problema através do algoritmo simplex primal, sua reformulação dual, bem como os casos com restrição de caixa, ou seja, no caso em que $l \leq x \leq u$, onde $l$ e $u$ são vetores do $R^n$ que indicam os limites inferiores e superiores de cada componente de x.

## 2.1 Simplex primal

Em 1949 foi enunciado por George B. Dantzig o método “simplex”, uma ferramenta para resolver problemas do tipo:

$$(P) \quad \begin{aligned} &\textit{Min } c'x \\ &\textit{Sujeito a: } Ax = b \\ &x \geq 0 \end{aligned}$$

Vale notar que restrições do tipo ≥ ou ≤ podem ser facilmente convertidas em restrições de igualdade com a inclusão de uma variável de folga, bem como um problema de maximização pode ser convertido em minimização, pois Max Ω = - Min – Ω.

Também iremos considerar que a matriz A é composta por linhas linearmente independentes (LI). Caso existam linhas dependentes, podemos excluí-las e criar um problema equivalente, que terá o mesmo valor ótimo.

O Simplex é um algoritmo que segue este raciocínio: dada uma solução viável (ou seja, um vetor $x^1$ que respeita $Ax^1 = b$), procuramos outra solução viável, digamos $x^2$, próxima à solução anterior, com melhor valor objetivo, ou seja $Ax^2 = b$ e $c'x^2 < c'x^1$. Se tal solução não existe então paramos, pois temos uma solução ótima em mãos. Caso contrário, procuramos uma “direção” d que podemos seguir que (i) não nos leve para fora do conjunto viável e (ii) melhore o valor objetivo, dado por c’x.

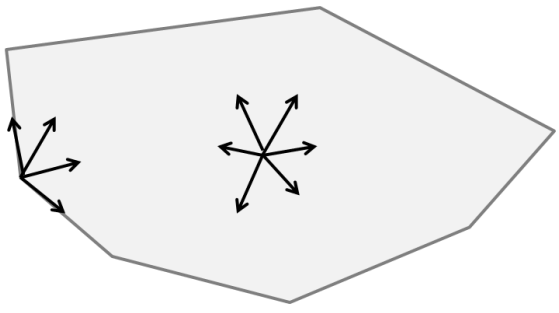

Figura 2.1.1: Exemplo de direções viáveis em diferentes pontos

Como estamos interessados em equações lineares, a obtenção de um mínimo local implica em obtenção de um minimo global, pois nosso conjunto viável é convexo.

Vamos fazer A = [ B R ], onde B e R são submatrizes de A, sendo B uma matriz inversível de tamanho m por m e R de tamanho m por (n-m). Também vamos fazer x = [ $x_B$ $x_R$ ], sendo $x_B$ um vetor de tamanho n e $x_R$ um vetor de tamanho (n-m). Desta forma, podemos reescrever Ax = b como B $x_B$ + R$x_R$ = b ⇔ $x_B$ = $B^{-1}$ (b - R$x_R$).

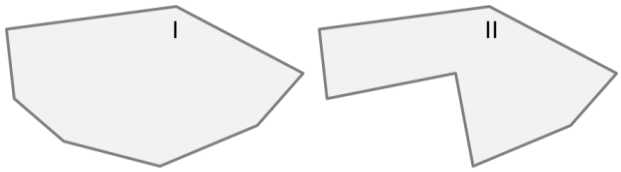


Figura 2.1.2: Exemplo de conjunto convexo (I) e não convexo (II)

B será a matriz básica, assim como $x_B$ será o vetor associado a esta base. R, em contrapartida, é a matriz de variáveis residuais. Fazendo $x_R$=0, x será viável se e somente se $x_B$ ≥ 0. Se x for inviável, significa que precisamos procurar por outra base B. Se x for viável, estamos interessados em sair do ponto $x^1$ e caminhar para o ponto $x^2$ que forneça melhor valor. Isto só pode ser feito aumentando o valor de alguma variável residual até o valor em que alguma variável básica chegue ao seu limite inferior (zero). Neste caso diremos que a variável residual que aumentou seu valor *entra na base*, enquanto a variável básica que chegou ao seu limite inferior *sai da base*.

Para saber quais variáveis podem entrar na base, precisamos primeiro descobrir se existe alguma variável que, se entrar, irá melhorar o valor. Para isso, vamos estudar o como o aumento das variáveis residuais influenciam o valor objetivo:

**c'x =** [ $c_B$ $c_R$ ]' [ $x_B$ $x_R$ ]
= $c'_B x_B$ + $c'_R x_R$
= $c'_B$ $B^{-1}$ (b - R$x_R$) + $c'_R x_R$
= $c'_B$ $B^{-1}$ b – $c'_B$ $B^{-1}$ R$x_R$ + $c'_R x_R$
= **$c'_B$ $B^{-1}$ b + ($c'_R$ – $c'_B$ $B^{-1}$ R) $x_R$**

A quantidade $\bar{c}_j = c_{R_j} - c'_B B^{-1} R_j$ representa o custo por unidade de incremento na variável residual $\mathbf{x_j}$. Estamos, portanto, interessados apenas nas variáveis residuais $\mathbf{x_j}$ com custo unitário negativo. Desta forma um aumento desta variável implica em reduzir a expressão $\boldsymbol{c'x}$.

Suponhamos então que vamos nos mover um passo $\boldsymbol{\Theta > 0}$ em uma direção $\mathbf{d}$ associada a um custo negativo, dessa maneira sairemos do vetor $\mathbf{x}$ para o vetor $\mathbf{x + \Theta d}$. Como este novo vetor precisa ser viável, ele precisa satisfazer a condição **A(x + Θ d) = b**, temos as seguintes equivalências:
**A(x + Θ d) = b** ⇔ Ax + Θ Ad = b ⇔ Θ Ad = 0 ⇔ Ad = 0 ⇔ [ B R ] [ $d_B$ $d_R$ ]' = 0 ⇔
⇔ **B$d_B$ + R$d_R$ = 0**

Como queremos nos mover apenas na j-ésima direção, ou seja, aumentar o valor apenas da j-ésima variável residual, temos que $\mathbf{d_{Rj} = 1}$ e $\mathbf{d_{Ri} = 0}$ para todo i ≠ j. Desta forma, a condição acima se resume a $Bd_B + A_j d_{Rj} = 0 \Leftrightarrow Bd_B + A_j = 0 \Leftrightarrow Bd_B = -A_j \Leftrightarrow \mathbf{d_B = -B^{-1}A_j}$.

Por fim, é necessário encontrar o tamanho do passo $\boldsymbol{\Theta}^*$ que é possível tomar na direção $\mathbf{d}$ sem violar as restrições das variáveis ($x \geq 0$). Como a direção $\mathbf{d}$ está associada a um custo negativo, é interessante andar o máximo possível nesta direção até o ponto $\mathbf{x} + \boldsymbol{\Theta}^*\mathbf{d}$, tal que $\boldsymbol{\Theta}^* = \mathbf{max}\{\boldsymbol{\Theta} \geq \mathbf{0} \mid (\mathbf{x} + \boldsymbol{\Theta}\,\mathbf{d}) \geq \mathbf{0}\}$.

Para encontrar uma fórmula para $\boldsymbol{\Theta}^*$ é preciso analisar $\mathbf{x} + \boldsymbol{\Theta}\mathbf{d}$, que se torna inviável quando alguma componente se tornar negativa. Existem dois casos:

(1) Se todas componentes de $\mathbf{d}$ forem não negativas (diremos $\mathbf{d} \geq \mathbf{0}$), então para todo valor $\boldsymbol{\Theta} \geq \mathbf{0}$, tem-se que $\mathbf{x} + \boldsymbol{\Theta}\mathbf{d} \geq \mathbf{0}$, que sempre será viável. Neste caso $\boldsymbol{\Theta}^* = \infty$.

(2) Se $\mathbf{d_i} < \mathbf{0}$ para algum $\mathbf{i}$, a desigualdade $\mathbf{x_i} + \boldsymbol{\Theta}\mathbf{d_i} \geq \mathbf{0}$ pode ser reescrita como $\boldsymbol{\Theta} \leq -\frac{\mathbf{x_i}}{\mathbf{d_i}}$. Como esta condição deve ser satisfeita para todo $\mathbf{d_i} < \mathbf{0}$, o maior valor possível para $\boldsymbol{\Theta}$ é dado por:

$$\boldsymbol{\Theta}^* = \mathbf{min}\{-\frac{\mathbf{x_i}}{\mathbf{d_i}} \mid \mathbf{d_i} < \mathbf{0}\}$$

Após calcular $\boldsymbol{\Theta}^*$, calculamos o novo valor objetivo, dado por $\mathbf{c}'(\mathbf{x} + \boldsymbol{\Theta}^*\boldsymbol{d})$. Se $\boldsymbol{\Theta}^* = \infty$, dizemos que temos um problema ilimitado, com valor ótimo igual a $-\infty$.

Seguindo este raciocínio, conseguimos, a partir de um vértice inicial, percorrer arestas de P, até obter o valor ótimo.

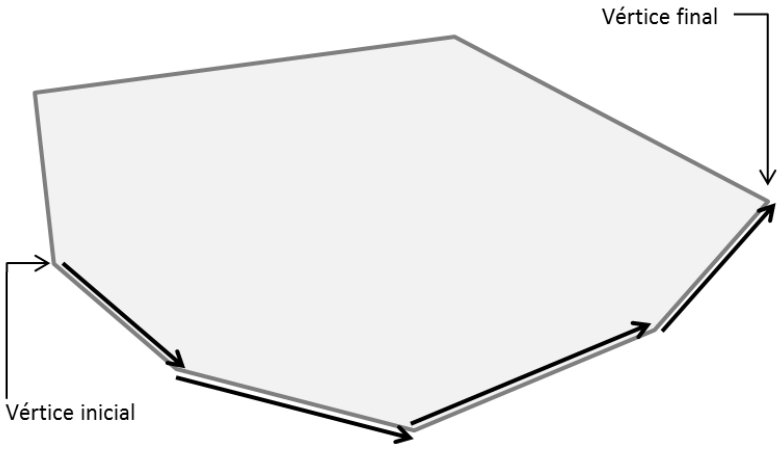


Figura 2.1.3: Exemplo de caminho percorrido do vértice inicial ao final em 4 iterações

Em uma breve análise, percebemos que o cálculo mais trabalhoso é o da inversa da base $B^{-1}$. Para minimizar esse custo computacional, pensou-se em um algorítmo que fizesse a atualização de $B^{-1}$ de maneira inteligente, pois a cada iteração apenas uma coluna de B se modifica (a que está associada à variável que sai da base).

Desta forma o algoritmo simplex revisado pode ser resumido da seguinte forma:

0. Encontre uma base **B**, associada a um vetor **x** viável e calcule sua inversa $\mathbf{B^{-1}}$.

1. Calcule $\bar{c_j} = c_{R_j} - (c'_B B^{-1} A)_j$ e pare no primeiro $\bar{c_j} < 0$. Se não existir custo negativo a solução atual é ótima e o algorítmo acaba. Caso contrário, guarde j.

2. Calcule $\mathbf{u = B^{-1} A_j}$. Se nenhuma componente de u for positiva, o problema é ilimitado, com valor ótimo igual a $-\infty$ e o algorítmo acaba.

3. Calcule $\Theta^* = \min \{ \frac{x_{B(i)}}{u_i} \mid u_i < 0 \}$ e $\mathbf{k = argmin} \{ \frac{x_{B(i)}}{u_i} \mid u_i < 0 \}$. Forme uma nova base substituindo $A_{B(k)}$ por $A_j$. Seja y a nova solução viável, então $y_j = \Theta^*$, $y_{B(i)} = x_{B(i)} - \Theta^* u_i$, para $i \neq k$.

4. Forme a matriz [ $B^{-1}$ u ]. Faça pivotações até transformar essa matriz em [$\bar{B}^{-1}$ $e_k$ ], onde $e_k$ é a k-ésima coluna da identidade. As primeiras m colunas dessa matriz formam a nova inversa de base.

5. Volte ao passo 1.

## 2.2 Simplex dual

A teoria da dualidade trata da relação entre o problema primal e o seu equivalente dual, desvendando o método simplex dual para resolução deste problema.

O início da teoria foi abordar o problema:

$$(P) \quad \begin{array}{l} \textit{Min c'x} \\ \textit{Sujeito a: Ax = b} \\ \textit{x} \geq \textit{0} \end{array}$$

e convertê-lo em:

$$(\bar{P}) \quad \begin{array}{l} \textit{Min c'x + p(b – Ax)} \\ \textit{Sujeito a: x} \geq \textit{0} \end{array}$$

A ideia desta formulação é permitir que as restrições sejam violadas, porém aplicando uma penalização ***p*** caso isso aconteça, dessa forma tornando essa violação desinteressante para o algorítmo (aumentando o valor ótimo do problema). Chamaremos essa formulação de problema relaxado.

Seja g(p) o valor ótimo do problema relaxado como função do vetor p. Por permitirmos a violação de Ax = b, o conjunto viável deste problema é maior, portanto

esperamos que o valor ótimo do problema relaxado não seja maior do que o original, isto é:

$$g(p) = \min \{ c'x + p(b - Ax) \mid x \geq 0 \} \leq c'x^* + p(b - Ax^*) = c'x^* + p(b - b) = c'x^*.$$

Resumindo: **$g(p) \leq c'x^*$**

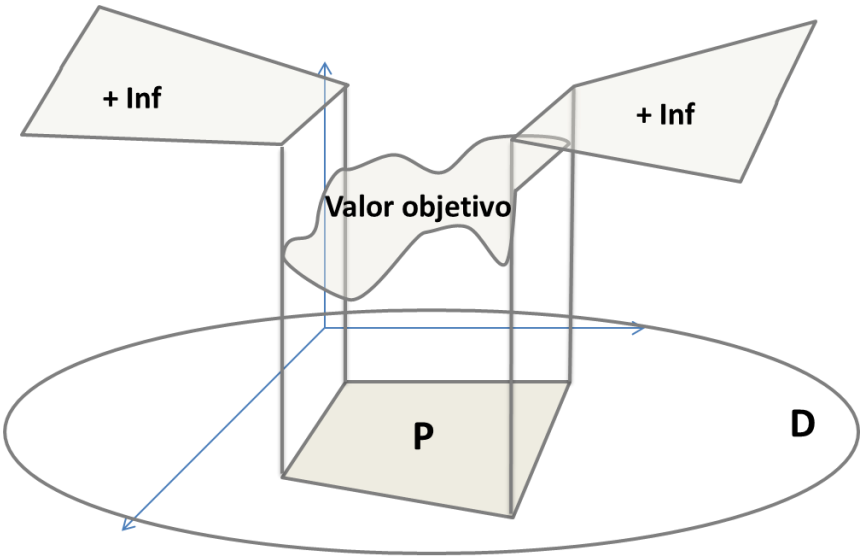


Figura 2.2.1: Violar o primal não é interessante para o dual

Cada p nos leva a um limitante inferior g(p). Desta forma podemos nos conter a encontrar o maior desses limitantes inferiores, ou seja, **max g(p),** e este problema é conhecido como problema dual.

Da definição de g(p) temos que:

$g(p) = \min \{ c'x + p'(b - Ax) \mid x \geq 0 \}$

$= p'b + \min \{ (c' - p'A)x \mid x \geq 0 \}$

sendo que $\min \{ (c' - p'A)x \mid x \geq 0 \} = 0$ se $c' - p'A \geq 0$ e $-\infty$ caso contrário. Desta forma podemos reescrever nosso dual como sendo:

$$(D) \quad \textbf{Max } p'b \quad \textbf{Sujeito a: } p'A \leq c'$$

Construímos assim o problema dual, que é equivalente (ou seja, ou são ambos inviáveis, ou eles têm o mesmo valor ótimo) ao seu primal. Vale notar que partimos do problema com restrições da forma Ax = b e variáveis não negativas. Como desigualdades podem ser convertidas para igualdades com a inclusão de variáveis residuais e variáveis negativas ou livre de sinais podem ser convertidas em variáveis não negativas com a inclusão de novas variáveis, trataremos sem perda de generalidade apenas o dual D.

O teorema forte da dualidade nos diz que se o problema primal P tiver uma solução ótima, então o seu equivalente dual D também a terá e seus valores ótimos serão iguais.

A prova deste teorema decorre do fato de que se P tem uma solução ótima x*, associada a uma base B, então o vetor de custos reduzidos será não negativo, ou seja, $c' - c_B' B^{-1} A \geq 0$. Definimos então uma variável $p = c_B' B^{-1}$, dessa forma p é solução viável de D, pois $c' - p'A \geq 0 \Leftrightarrow p'A \leq c'$. Além disso temos que $p'b = c_B' B^{-1} b$ e $x_B = B^{-1} b$, logo $\mathbf{p'b = c_B' x_B = c'x^*}$.

Em termos práticos, comparando o algorítmo simplex dual com o primal, a maior diferença reside no fato de que no primal primeiro escolhe-se a variável que irá entrar na base, para depois escolher qual irá sair, enquanto o dual faz o inverso: primeiro procura uma variável para sair, para depois escolher quem irá entrar.

Computacionalmente o simplex primal parte de uma solução viável e caminha até um ponto ótimo, mantendo viabilidade. O dual, em contrapartida, começa com uma solução ótima inviável e caminha para um ponto viável, mantendo otimalidade.

Essa abordagem faz com que o simplex dual leve menos iterações (na maioria dos casos) para encontrar uma solução ótima (se existir) em comparação com o seu equivalente primal.

Uma típica iteração do simplex dual consiste em:

0. Encontre uma base **B**, associada a um vetor de custos reduzidos $\bar{c}$ não negativo, calcule sua inversa $\mathbf{B^{-1}}$ e calcule $\mathbf{x_B}$.

1. Se $\mathbf{x_B \geq 0}$, esta solução é ótima. Caso contrário guarde o primeiro **i**, tal que $\mathbf{x_{B(i)} \leq 0}$. A variável $x_i$ irá sair da base.

2. Resolva $\mathbf{w = (B^{-1} e_i)' R}$, onde $e_i$ é a i-ésima coluna da identidade e R é a submatriz A de índices residuais.

3. Seja **J** o conjunto das variáveis não básicas $\mathbf{x_j}$, para as quis $\mathbf{w_j < 0}$. Se J for vazio, o problema é inviável. Caso contrário, calcule $\mathbf{k = argmin \{ \bar{c}_j / w_j \mid j \subset J \}}$. A variável $x_k$ irá entrar na base.

4. Calcule $\mathbf{d = B^{-1} A_k}$.

5. Calcule $\mathbf{\Theta = x_i / w_k}$. Seja y a nova solução viável, então faça $\mathbf{y_k = \Theta}$ e $\mathbf{y_{B(l)} = x_{B(l)} - \Theta * d_l}$, para $\mathbf{l \neq k}$.

6. Faça $\bar{c}_i = - \bar{c}_k / w_k$ e some $\bar{c}_i / w_s$ para todo $\bar{c}_s$ tal que $\mathbf{s \neq i}$.

7. Forme a matriz [ $B^{-1}$ d ]. Faça pivotações até transformar essa matriz em [$B^{-1}$ $e_k$], onde $e_k$ é a k-ésima coluna da identidade. As primeiras m colunas dessa matriz formam a nova inversa de base.

8. Volte ao passo 1.

## 2.3 Simplex primal com restrições de caixa

O simplex primal com restrições de caixa é muito similar ao simplex original,

$$(P) \quad \begin{array}{l} \textbf{\textit{Min c'x}} \\ \textbf{\textit{Sujeito a: Ax = b}} \\ \textbf{\textit{x ≥ 0}} \end{array}$$

porém desta vez estaremos interessados em resolver problemas do tipo

$$(PC) \quad \begin{array}{l} \textbf{\textit{Min c'x}} \\ \textbf{\textit{Sujeito a: Ax = b}} \\ \textbf{\textit{lw ≤ x ≤ up}} \end{array}$$

onde *lw* e *up* são vetores do $R^n$ que indicam os limitantes inferiores e superiores de cada componente de x. Vale notar que P é um caso especial de PC, onde *lw* = 0 e *up* = $\infty$.

Quando todas as componentes de *lw* e *up* forem finitos, dizemos que temos um problema de caixa limitado, caso contrário teremos um problema de caixa ilimitado.

Problemas deste tipo surgem quando nossas variáveis de decisão estão obrigatoriamente contidas em intervalos: por exemplo, o dinheiro disponível para aplicar está entre R$ 100 e R$ 5000, ou precisar plantar no mínimo 10 hectares de milho por questões contratuais, e no máximo 100 hectares, por limitação do espaço disponível.

Claramente conseguimos incluir uma desigualdade deste tipo dentro da matriz A e com a ajuda de 2 variáveis residuais resolver o problema como vimos anteriormente. Desta maneira, se tivermos **n** variáveis e **m** restrições, criamos um problema secundário com **n + 2n = 3n** variáveis e **m + 2n** restrições, aumentando consideravelmente o tamanho do problema. Além disso, para criar um problema auxiliar para este problema secundário, de forma a obter uma base para inicializar o simplex, precisaríamos incluir variáveis artificiais, fazendo com que este problema auxiliar tivesse **3n + m + 2n = 5n + m** variáveis! Esta obviamente não é a melhor abordagem, pois um problema relativamente simples pode acabar consumindo muitos recursos computacionais.

Para resolver este problema, precisamos apenas rever algumas passagens do algorítmo simplex. Tomemos como ponto de partida o algorítmo mencionado anteriormente:

*0. Encontre uma base **B**, associada a um vetor **x** viável e calcule sua inversa $\boldsymbol{B^{-1}}$.*

Continuaremos precisando de uma base inicial viável, ou seja, tal que $lw \leq x_B \leq up$. As variáveis residuais deverão estar ou em seus limitantes inferiores ou superiores para garantir que estamos em um vértice.

*1. Calcule* $\bar{c}_j = c_{Rj} - c_B' B^{-1} A_j$ *e pare no primeiro* $\bar{c}_j < 0$. *Se não existir custo negativo a solução atual é ótima e o algorítmo acaba. Caso contrário, guarde j.*

Após calcular os custos reduzidos, precisamos considerar a posição atual das nossas variáveis residuais. Se uma variável estiver em seu limite inferior, um custo reduzido negativo significa que posso aumentar o valor desta variável, melhorando o valor ótimo do problema. Se estiver em seu limite superior, um custo reduzido positivo significa que posso diminuir o valor desta variável, melhorando o valor ótimo. Sendo assim, escolho o primeiro j, tal que $\bar{c}_j < 0$ e $x_j = lw_j$ ou $\bar{c}_j > 0$ e $x_j = up_j$.

*2. Calcule* $u = B^{-1} A_j$. *Se nenhuma componente de u for positiva, o problema é ilimitado, com valor ótimo igual a* $-\infty$ *e o algorítmo acaba.*

Precisaremos analisar os casos de componentes positivas e negativas de u. Se u = 0 teremos problema ilimitado, o que nunca acontece no caso de termos $lw > -\infty$ e $up < \infty$.

*3. Calcule* $\Theta^* = \min \{ x_{B(i)}/u_i \mid u_i < 0 \}$ *e* $\lambda = \text{argmin} \{ x_{Bi}/u_i \mid u_i < 0 \}$. *Forme uma nova base substituindo* $A_{B(\lambda)}$ *por* $A_j$. *Seja y a nova solução viável, então* $y_j = \Theta^*$, $y_{B(i)} = x_{B(i)} - \Theta^* u_i$, *para* $i \neq \lambda$

O maior passo Θ* que é possível dar deve levar em conta 3 fatores:

i. Nenhuma variável básica pode ficar menor que seu limite inferior

ii. Nenhuma variável básica pode ficar maior que seu limite superior

iii. Se a variável $x_j$ que irá entrar na base estiver em seu limite inferior, então o passo não pode ser maior que $up_j - lw_j$. Caso contrário, segue raciocínio análogo.

Desta forma, nosso cálculo de Θ* é dado por:

Se $x_j$ estiver em seu limite inferior:

$$\Theta^* = \min \{ up_j - lw_j \; ; \; \min \{ (x_{B(i)} - lw_i)/u_i \mid u_i > 0 \} \; ; \; \min \{ (x_{B(i)} - up_i)/u_i \mid u_i < 0 \} \}$$

Se $x_j$ estiver em seu limite superior:

$$\Theta^* = \max \{ lw_j - up_j \; ; \; \max \{ (x_{B(i)} - lw_i)/u_i \mid u_i < 0 \} \; ; \; \max \{ (x_{B(i)} - up_i)/u_i \mid u_i > 0 \} \}$$

Se $x_j = lw_j$ e $\Theta^* = up_j - lw_j$ ou $x_j = up_j$ e $\Theta^* = lw_j - up_j$, significa que o passo é suficiente para levar a j-ésima variável de um limite ao outro, ou seja, ela nem chega a entrar na base. Neste caso apenas atualizamos a variável $x_j$ com seu novo valor $x_j + \Theta^*$.

Caso contrário, guarde o índice k da variável que foi utilizada para encontrar $\Theta^*$ ($x_k$ irá sair da base). Atualizamos a nova solução normalmente.

*4. Forme a matriz [ $B^{-1}$ u ]. Faça pivotações até transformar essa matriz em $[\tilde{B}^{-1}e_k]$, onde $e_k$ é a k-ésima coluna da identidade. As primeiras m colunas dessa matriz formam a nova inversa de base*

O cálculo da atualização da inversa de base não se altera.

Desta forma vemos que utilizando o mesmo algorítmo, porém com algumas ressalvas, conseguimos resolver o problema com restrições de caixa sem precisarmos criar problemas auxiliares com mais variáveis e restrições.

## 2.4 Simplex dual com restrições de caixa

O simplex dual com restrições de caixa segue o mesmo raciocínio dos tópicos anteriormente mencionados.

Desta vez, estamos interessados em:

1) Resolver problemas com restrições de caixa;

2) Utilizar o algorítmo dual, mais eficiente que o primal.

Assim como o algorítmo simplex pôde ser revisto, de forma a permitir a abordagem com restrições de caixa, podemos adaptar nosso algorítmo dual para as restrições de caixa:

*0. Encontre uma base **B**, associada a um vetor de custos reduzidos $\bar{\mathbf{c}}$ não negativo, calcule sua inversa $\mathbf{B^{-1}}$ e calcule $\mathbf{x_B}$.*

A modificação deste passo é que a base inicial deve ser tal que o vetor de custos reduzidos seja ótimo, isto é:

$\bar{c}_j < 0$ se $x_j = up_j$

$\bar{c}_j > 0$se $x_j = lw_j$

*1. Se* $x_B \geq 0$*, esta solução é ótima. Caso contrário guarde o primeiro* $i$*, tal que* $x_{B(i)} \leq 0$*. A variável* $x_i$ *irá sair da base.*

Como estamos trabalhando com problemas de caixa, nossa solução apenas será ótima se $lw_B \leq x_B \leq up_B$. Se sim, paramos o algorítmo, caso contrário escolho o primeiro i, tal que $x_{B(i)} \leq lw_{B(i)}$ ou $x_{B(i)} \geq up_{B(i)}$.

*2. Resolva* $w = (B^{-1} e_i)' R$*, onde* $e_i$ *é a i-ésima coluna da identidade e R é a submatriz A de índices residuais.*

Este passo não sofre alterações.

*3. Seja* $J$ *o conjunto das variáveis não básicas* $x_j$*, para as quais* $w_j < 0$*. Se J for vazio, o problema é inviável. Caso contrário, calcule* $k = \text{argmin} \{ \bar{c}_j / w_j \mid j \subset J \}$*. A variável* $x_k$ *irá entrar na base.*

Como nossas variáveis residuais podem tanto estar em seus limites inferiores como nos superiores, precisamos analisar dois casos:

i) Se $x_i < lw_i$, então seja J o conjunto das variáveis não básicas $x_j$, para as quais:

$w_j < 0$ e $x_j < up_j$ ou

$w_j > 0$ e $x_j > lw_j$

ii) Se $x_i > up_i$, então seja J o conjunto das variáveis não básicas $x_j$, para as quais:

$w_j > 0$ e $x_j < up_j$ ou

$w_j < 0$ e $x_j > lw_j$

Se J for vazio, então o problema é inviável.

Caso contrário, calcule $\mathbf{k = argmin \{ abs(\bar{c}_j / w_j)| j \subset J \}}$. A variável $x_k$ irá entrar na base.

*4. Calcule* $d = B^{-1} A_k$*.*

O cálculo do vetor de direção não se altera.

*5. Calcule* $\Theta = x_i / w_k$*. Seja y a nova solução viável, então faça* $y_k = \Theta$ *e* $y_{B(l)} = x_{B(l)} - \Theta * d_l$*, para* $l \neq k$*.*

Neste passo, o cálculo do passo Θ se altera. Temos duas possibilidades:

i) $\Theta = (x_i - lw_i)/w_k$, se $x_i < lw_i$

ii) $\Theta = (x_i - up_i)/w_k$, se $x_i > up_i$

Então seja y a nova solução viável. Faça $y_k = y_k + \Theta$ e $y_{B(l)} = x_{B(l)} - \Theta * d_l$, para $l \neq k$.

*6. Faça* $\bar{c}_i = -\bar{c}_k / w_k$ *e some* $\bar{c}_i * w_s$ *para todo* $\bar{c}_s$ *tal que* $s \neq i$.

O cálculo da atualização do vetor de custos reduzidos não se altera.

*7. Forme a matriz [* $B^{-1} d$ *]. Faça pivotações até transformar essa matriz em* $[\tilde{B}^{-1} e_k]$, *onde* $e_k$ *é a k-ésima coluna da identidade. As primeiras m colunas dessa matriz formam a nova inversa de base.*

O cálculo da atualização da inversa de base não se altera.

Desta forma vemos que com algumas atualizações no algorítmo anterior, conseguimos utilizar o método simplex dual para resolver problemas sem precisarmos criar problemas auxiliares com mais variáveis e restrições.

## 2.5 Simplex dual com restrições de caixa e variáveis inteiras

Todos algoritmos vistos até agora pressupõem que a solução (x) pertence aos números reais. Entretanto se quisermos uma solução num subconjunto discreto ou misto (parte contínuo, parte discreto) é necessário alterar um pouco o raciocínio.

Dado o problema:

$$(P) \quad \begin{aligned} &\textbf{Min } c'x \\ &\textbf{Sujeito a: } Ax = b \\ &lw \leq x \leq up \\ &x_1 \in N,\ x_{2\ldots n} \in N^{n-1} \end{aligned}$$

o primeiro passo é resolver seu equivalente:

$$(PC) \quad \begin{array}{l} \textbf{\textit{Min c'x}} \\ \textbf{\textit{Sujeito a: Ax = b}} \\ \textbf{\textit{lw} } \leq \textbf{\textit{x}} \leq \textbf{\textit{up}} \\ \textbf{\textit{x}} \in \textbf{\textit{R}}^{\textbf{\textit{n}}} \end{array}$$

que terá como solução $x^* \in R^n$. Digamos que $lw_1 \leq \lfloor x_1^* \rfloor = a_1 < x_1^* < \lceil x_1^* \rceil = a_2 \leq up_1$. Neste caso, para impor $x_1 \in N$, é necessário analisar dois sub-problemas: um no qual iremos criar uma restrição de caixa para $x_1$ da forma $lw_1 \leq x_1 \leq a_1$, outro no qual a restrição em torno de $x_1$ é da forma $a_2 \leq x_1 \leq up_1$.

Como $x_1^*$ era ótimo no problema original, o novo $x_1^*$ nos sub-problemas será igual a $a_1$ ou $a_2$, ambos inteiros. Para selecionar o sub-problema correto, basta ver qual tem o melhor valor ótimo.

Para impor que outras variáveis sejam inteiras, basta realizar este procedimento recursivamente.

Desta forma, vemos que a imposição de um conjunto solução discreto não exige mudança no algorítmo, embora exija que este seja executado recursivamente, ou seja, para um PPL com *n* variáveis, das quais $m \leq n$ necessitam ser discretas, é necessário executar o algorítmo $1 + 2m$ vezes.

Vale notar que este raciocínio também pode ser utilizado no algorítmo simplex primal com restrições de caixa.

## Capítulo 3 – Técnicas de Divisão

Sempre que somos confrontados com a decisão de dividir um pedaço de bolo igualmente entre as pessoas em uma festa, surge a dúvida se de fato a divisão feita foi justa para todos. Será que todos receberam partes iguais ou alguém se sentiu injustiçado?

Questões de "justiça" nos rodeiam constantemente: Pagamos uma quantidade justa de impostos? O preço da gasolina é justo? As leis do nosso país são justas?

Entretanto, mesmo estando presente em nosso cotidiano, "justiça" é algo difícil de se definir, principalmente porque cada pessoa tem a sua impressão do que é algo justo.

Vamos, desde já, definir o que é uma divisão justa no contexto deste trabalho:

**Definição:** Seja X um objeto divisível e $p_1, p_2, ..., p_n$, as n pessoas que irão dividir X. Seja $\mu_i(x) \in [0\%,100\%]$ a medida de satisfação associada a uma parte $x \subset X$ para $p_i$ tal que $\mu_i(\emptyset) = 0\%$, $\mu_i(X) = 100\%$ e se $x \cap y = \emptyset \Longrightarrow \mu_i(x) + \mu_i(y) = \mu_i(x \cup y)$ $\forall$ i = 1, ..., n. Desta forma dizemos que uma partição $X=x_1 \cup x_2 \cup ... \cup x_n$ é justa se $\mu_i(x_i) \geq 1/n$, onde $x_i$ representa a parte de X pertencente a $p_i$

O objetivo deste capítulo, portanto, é abordar algoritmos de divisão e, com o auxílio da definição acima, dizer com quais é possível realizar divisões justas aos participantes envolvidos. Curiosamente, veremos que quando existe divergência entre o que cada uma das partes envolvidas acha justo (isto é, $\mu_i(x_i) \neq \mu_j(x_i)$, $\forall$ i = 1, ..., n, $\forall$ j = 1, ..., n, i ≠ j), é mais fácil criar uma divisão justa. Também veremos que é possível ter $\sum_{i=0}^{n} \mu_i(x_i) > 100\%$.

Este capítulo segue de perto as ideias prsentes em [Rob98].

## 3.1 O método *Cut-and-Choose*

Para iniciar nossa abordagem sobre técnicas de divisão, começaremos com um exemplo simples: a mãe de João e Pedro tem que dividir uma fatia de bolo entre os dois. Por ser a mãe, quer que ambos os pupilos recebam quantidades justas, de forma que nenhum dos irmãos ache que recebeu menos que o outro. De imediato é possível ver que existem métodos mais justos que outros. Para ilustrar, vamos analisar 4 métodos distintos:

| | |
|---|---|
| Método 1 | A mãe corta o bolo em dois pedaços, que ela presume serem iguais, e distribui um para cada filho. |
| Método 2 | A mãe corta o bolo em dois pedaços. João e Pedro tiram “par ou impar” para ver quem escolhe primeiro. O outro pega o pedaço restante. |
| Método 3 | João corta o bolo em duas partes, escolhe uma para si e Pedro pega o pedaço restante |
| Método 4 | João corta o bolo em duas partes que julga serem justas para ambos e Pedro escolhe o pedaço que lhe agrada. João pega o pedaço restante |

Pelo Método 1, a mãe, que é indiferente quanto ao bolo, faz um corte que julga dividí-lo em duas partes iguais, ou seja, cada uma representando 50% do bolo. Entretanto, vamos supor que uma parte do bolo contenha cerejas e João não gosta de cerejas. Desta forma ele avalia que os pedaços que a mãe cortou, em sua visão, representam 30% (parte com cerejas) e 70% (parte sem cerejas) do bolo. Já Pedro gosta de cerejas e, por isso avalia que as partes representam 60% e 40% respectivamente (Fig. 3.1.1). Neste caso, se a mãe der a parte com cerejas a João e sem cerejas a Pedro, nenhum dos dois filhos ficará satisfeito ao final da distribuição. Em compensação, se a mãe fizer distribuir de maneira contrária, ambos sentirão que receberam mais do que tinham direito e, portanto, ficarão felizes. Como este método depende da sorte de a mãe fazer a melhor escolha, vemos que não é um algoritmo confiável.

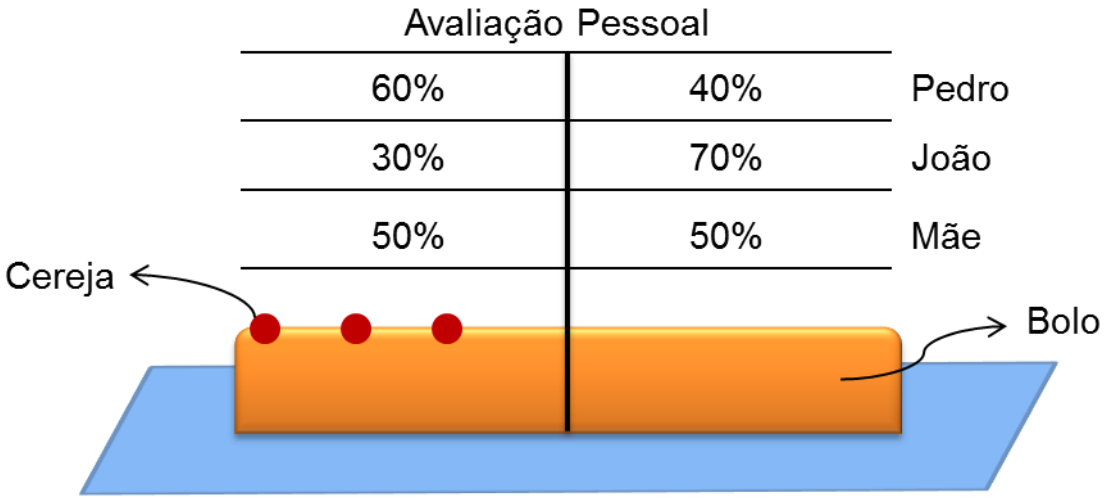


Figura 3.1.1: Representação ilustrativa do bolo

Partindo para o método 2, vemos que também não é confiável. Podemos imaginar o caso em que ambos gostam de cereja e apenas um dos pedaços contem cereja. Logo, o irmão que ganhar o "par-ou-impar" se sentirá justiçado, pois terá a possibilidade de escolher o pedaço com cereja (que avalia valer mais que 50% do bolo), enquanto o outro terá que ficar com a parte sem cereja (que para ele representa menos que 50% do bolo).

Os métodos 3 e 4 partem para outra abordagem, que consiste em deixar participar da divisão apenas quem irá receber os pedaços. A mãe, que é uma participante externa ao processo, ficará de fora. Mesmo assim, vemos que o método 3 não garante uma divisão justa aos dois, pois João pode cortar o bolo em pedaços que representam 90% e 10% e escolher o pedaço maior, enquanto o irmão recebe um pedaço que avalia valer, por exemplo 5%.

Já o método 4 consegue atingir o que chamamos de divisão justa. João cortará o bolo, sabendo que Pedro escolherá o pedaço primeiro. Logo, se cortar em partes "não justas", por exemplo, representando 90% e 10% em sua visão, pode sair prejudicado. Pedro, pode avaliar essas partes, por exemplo, em 80% e 20% e escolher a fatia representando os 80%, que para João representava 90%, logo lhe restando apenas o pedaço correspondente a 10% do bolo. Sendo assim, ao fazer a divisão, irá procurar dividir em partes que, para si, representam 50% e 50%. Desta forma, independentemente de qual fatia Pedro escolher, João terá garantido a si 50% do bolo. Na visão de Pedro, alguma das partes valerá, no mínimo 50% (se duas partes somam 100% e uma vale menos que 50%, então a outra obrigatoriamente vale mais que 50%), desta forma escolhendo a fatia que lhe agrada mais.

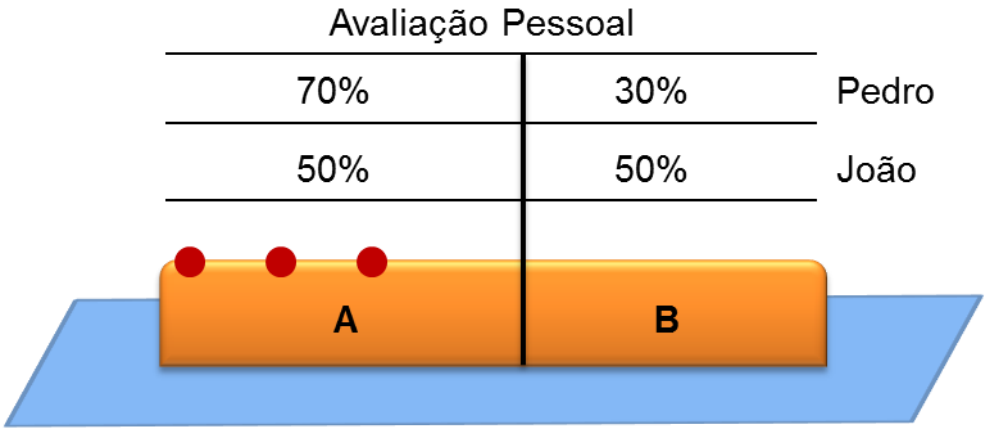


Figura 3.1.2: João corta, Pedro escolhe a fatia A e João fica com a fatia B

O método 4 é chamado na literatura de "*Cut-and-Choose*", ou "corte e escolha" e é justo, pois garante que se os participantes seguirem as regras dadas, ambos terão uma fatia justa do bolo. Caso algum participante decida não seguir as regras, apenas ele poderá sair prejudicado.

Com isso vemos que quando existe diferença de opiniões com relação a quanto vale cada pedaço do bolo, é possível obter uma satisfação total maior que 100%. No exemplo ilustrado acima, teríamos 120% de satisfação total. Entretanto, se olharmos para a figura 3.1.1, vemos que poderíamos ter até 130% de satisfação dependendo de qual fatia ficasse para cada irmão. Ou seja, esse método ainda não garante uma divisão ótima.

## 3.2 O método *Moving Knife*

O método *Moving Knife* – Faca Móvel – é de fácil entendimento e pode ser aplicado para divisão justa de um bolo entre várias pessoas. A ideia consiste em ter uma faca que começa de um dos extremos dos bolos e vai se movimentado lentamente até o outro extremo. Conforme a faca se movimenta, ao primeiro sinal de um dos participantes, a faca pára, corta o bolo, esse participante recebe sua fatia e sai da distribuição, deixando o resto do bolo sendo disputado pelos participantes restantes.

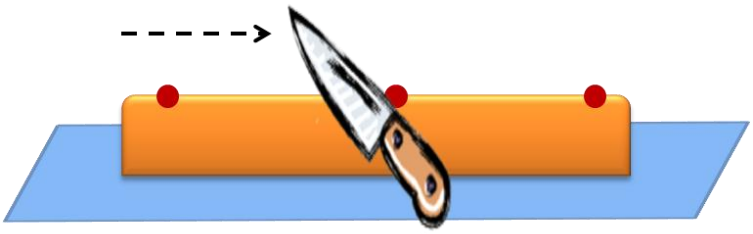

Figura 3.2.1: Método da Faca Móvel

Vamos supor, por exemplo, que estejam João, Pedro e Luiz a dividir o bolo. Conforme a faca se movimenta, enquanto nenhum dos participantes der um sinal, significa que nenhum dos participantes acha que já tem pelo menos 1/3 do bolo passado sob a faca. Vamos supor que João dê o primeiro sinal: isto significa que ele acha que pelo menos 1/3 do bolo já passou sob a faca, enquanto nenhum dos demais participantes acham que tem mais que 1/3. Desta forma João pega seu pedaço, enquanto os demais não se sentirão prejudicados, pois em suas visões o restante do bolo vale mais que 2/3. Novamente a faca prossegue seu caminho e vamos supor que Pedro dê o sinal. Novamente isto significa que a fatia para Pedro representa pelo menos 1/3 do bolo, enquanto para Luiz representa menos que 1/3, logo a fatia restante vale mais que 1/3 para Luiz.

Caso Pedro escolha não dar o sinal quando sabe que já teria uma fatia representando 1/3 do bolo, apenas para tentar conseguir um pedaço maior correspondente, digamos, a 40% em sua visão, corre o risco de Luiz dar o sinal quando a faca estivesse passando por 39% segundo a visão de Pedro. Neste caso Luiz ficaria com pelo menos 1/3 em sua visão, enquanto Pedro teria no máximo 2/3 - 39% = 27% restantes, que não necessáriamente lhe agradará.

Assim como o método Cut-and-Choose, com a Faca Móvel não é possível garantir a melhor solução possível, mas é possível garantir que cada um receberá pelo menos 1/3 do bolo se todos participantes seguirem as regras estipuladas.

Vale notar, entretanto, que este método não é discreto, isto é, não é possível implementá-lo computacionalmente. Isto acontece, pois a faca poderia se movimentar tão lentamente quanto quiséssemos, de forma a nunca chegar ao final do bolo, ou até mesmo não chegar em um ponto em que algum participante desse o primeiro sinal para cortar.

## 3.3 A felicidade do desacordo

Vimos nos tópicos anteriores que existem formas de lidar com divisão justa mesmo quando as pessoas envolvidas tem diferentes perspectivas do que é ser

justo. Agora veremos o porquê de ser possível obter satisfação total maior que 100% apresentando um novo algoritmo.

Suponha que tenhamos um terreno não uniforme a ser dividido igualmente entre os 3 filhos, João, Pedro e Luiz, fruto de uma herança deixada pelo pai. Um lado deste terreno contém a via de acesso, enquanto o lado oposto é banhado por um rio. De forma que todos tenham acesso à via e ao rio, a divisão do terreno se dará em forma de faixas perpendiculares à via de acesso.

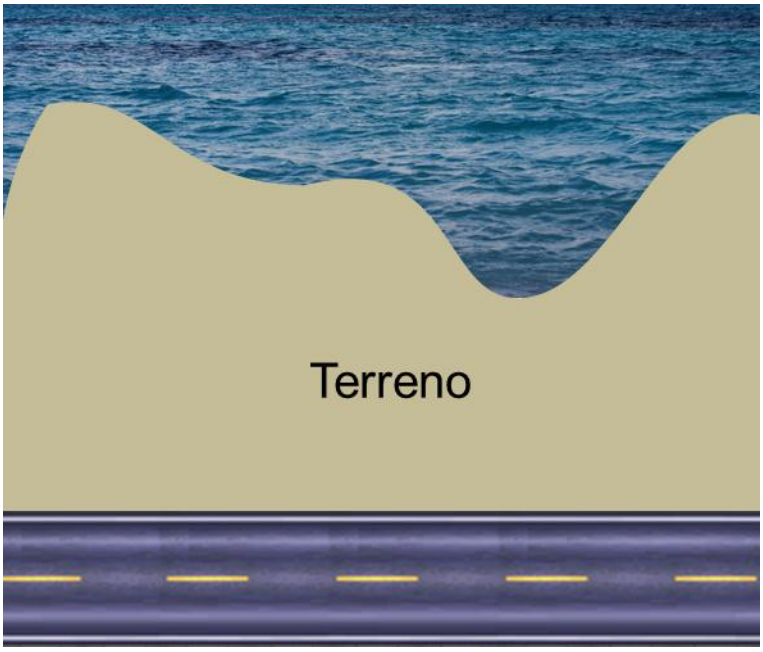


**Figura 3.3.1: Ilustração áerea do terreno no exemplo**

Para realizar a divisão do terreno, cada filho recebe uma copia da ilustração acima e são instruídos a marcarem linhas que representam a divisão do terreno em 3 partes iguais. Desta forma, cada um terá a sua versão do que seria uma divisão justa:


**Figura 3.3.2: Exemplo de divisão dos irmãos**

Sobrepondo as divisões, chegamos ao seguinte esquema:

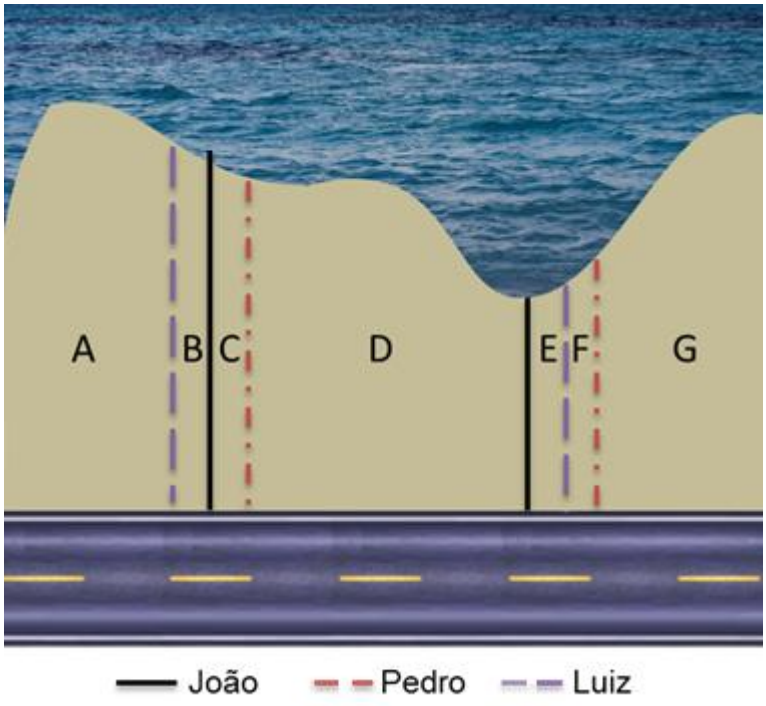


Figura 3.3.2: Esquemas unidos

Utilizando a definição dada no início do capítulo, podemos expressar o esquema acima como:

$\mu_{João}(A \textbf{ U } B) = \mu_{João}(C \textbf{ U } D) = \mu_{João}(E \textbf{ U } F \textbf{ U } G) = 1/3$

$\mu_{Pedro}(A \textbf{ U } B \textbf{ U } C) = \mu_{Pedro}(D \textbf{ U } E \textbf{ U } F) = \mu_{Pedro}(G) = 1/3$

$\mu_{Luiz}(A) = \mu_{Luiz}(B \textbf{ U } C \textbf{ U } D \textbf{ U } E) = \mu_{Luiz}(F \textbf{ U } G) = 1/3$

Sendo assim, a divisão de forma justa poderá ser feita dando-se a parte do terreno A a Luiz, C **U** D a João e G a Pedro (destaque acima), com cada um recebendo 1/3 do terreno, segundo sua visão. Vale notar que as faixas B, E e F ainda estão sem dono. Como $\mu_i(B) > 0$, $\mu_i(E) > 0$ e $\mu_i(F) > 0$, para i = João, Pedro, Luiz, então realizando-se o mesmo procedimento ou simplesmente sorteando-se as faixas restantes entre os filhos, por exemplo B para Luiz, E para João e F para Pedro, já garantiria satisfação total maior que 100%, pois:

$$\mu_{Luiz}(A \textbf{ U } B) + \mu_{João}(C \textbf{ U } D \textbf{ U } E) + \mu_{Pedro}(F \textbf{ U } G) =$$

$$= 100\% + \mu_{Luiz}(B) + \mu_{João}(E) + \mu_{Pedro}(F) > 100\%$$

Vale notar que a satisfação total será igual a 100% se, e somente se, as linhas de divisão de cada um fosse identica às dos outros. Desta forma cada um receberia 1/3 do terreno, segundo sua visão, e não haveriam faixas de terreno sobrando.

# Capítulo 4 – Otimização da Divisão

Vimos no capítulo 2 o Simplex, uma ferramenta capaz de maximizar ou minimizar uma função linear, sujeita a uma série de restrições lineares, e no capítulo 3, técnicas de divisão e a definição do que é uma divisão justa. Agora estamos interessados em unir os temas, ou seja, realizar a melhor divisão possível, ou seja, maximizar a soma da medida de satisfação entre os envolvidos na divisão, sujeito à divisão ser justa, através de um exemplo de separação judicial.

## 4.1 A separação judicial – Uma primeira abordagem

Quando é celebrado um casamento, as partes devem escolher entre um dos Regime de Bens existentes. Os mais conhecidos são a Comunhão Universal de Bens, a Comunhão Parcial de Bens e a Separação Total de Bens, cada um com suas particularidades.

Por exemplo, enquanto a Comunhão Universal de Bens prevê que na hipótese de divórcio todos bens (e dívidas) adquiridos pelos cônjuges, antes ou depois do casamento, integre o patrimônio comum a ser partilhado, a Comunhão Parcial de Bens prevê que seja integrado no patrimônio de partilha apenas o que foi adquirido durante o casamento. Já a Separação Total de Bens prevê que cada cônjuge leve o que está em seu nome e apenas é partilhado aquilo que estiver em nome de ambas partes.

Em todos os casos, quando necessário realizar alguma divisão entre as partes, nos deparamos com um cenário em que cada envolvido pode alegar merecer mais bens do que o outro. Vamos supor que durante o casamento um casal, que agora pretende se divorciar, adquiriu um apartamento, uma casa de praia e dois carros. Um juiz, assim como a mãe do exemplo no capítulo 3, precisa ser isento e garantir que as duas partes recebam parcelas justas. Neste caso poder-se-ia solicitar trazer a valor presente os bens e procurar a divisão de forma mais justa.

Neste exemplo, vamos supor que os valores são os seguintes:

| Bem | Valor Presente |
|---|---|
| Apartamento | R$ 290.000,00 |
| Casa de Praia | R$ 200.000,00 |
| Carro Compacto | R$ 30.000,00 |
| Carro Sedan Luxo | R$ 120.000,00 |
| **Total** | **R$ 640.000,00** |

Desta forma, uma divisão justa poderia ser feita dando-se o Apartamento e o Carro Compacto ao marido e a Casa de Praia e o Carro Sedan Luxo à esposa, cada um recebendo R$ 320.000,00 em bens. O problema desta divisão é que não levou-se em conta a vontade que cada um tinha em receber cada um dos bens: O marido poderia querer o Apartamento e o Sedan, enquanto a esposa preferisse a Casa de Praia e o Compacto. Outro ponto não considerado é que nem sempre será possível dividir em partes iguais, dando exatamente 50% para cada envolvido. Neste caso, como fazer para dividir em partes desiguais e de maneira justa?

Para resolver este problema, o primeiro passo é perguntar a cada envolvido quanto eles acham que cada bem representa em relação ao todo, ao invés de trazer a valor presente ou qualquer outro denominador comum.

Vamos supor que as estimativas são as seguintes:

| Bem | $\mu_{marido}$ | $\mu_{esposa}$ |
|---|---|---|
| Apartamento | 45% | 20% |
| Casa de Praia | 15% | 40% |
| Carro Compacto | 10% | 30% |
| Carro Sedan Luxo | 30% | 10% |
| **Total** | **100%** | **100%** |

Sendo assim, a melhor divisão possível é dando-se o Apartamento e o Sedan ao marido, e a Casa de Praia e o Compacto à esposa, pois assim $\mu_{marido}(apartamento) + \mu_{marido}(sedan) = 45\% + 30\% = 75\% > 50\%$, $\mu_{esposa}(casa) + \mu_{esposa}(compacto) = 40\% + 30\% = 70\% > 50\%$, pois desta forma ambos receberam o que mais queriam.

Este é um caso simples em que é possível resolver simplesmente por inspecção. Mas para um problema mais complexo, que envolvessem mais bens, ou que as medidas de satisfação fossem muito próximas ou mesmo iguais, essa análise manual tornar-se-ia complexa. Por isso vemos a necessidade de utilizar a ferramenta Simplex para resolver qualquer tipo de problema.

## 4.2 O modelo Simplex para separação judicial

O primeiro passo para desenvolver o modelo de resolução do problema pelo método Simplex é escrever o enunciado na forma de um PPL padrão. Em uma primeira abordagem, poderíamos pensar em definir nossa função objetivo como sendo maximizar a satisfação total, sujeito a restrições que definem a quantidade de itens a serem divididos por bem, ou seja:

$$\begin{cases} \max \ \sum_{i=1}^{n}(x_i\,\mu_x^i + \ y_i\,\mu_y^i) \\ \text{Sujeito a: } x_1 + \ y_1 = b_1 \\ \qquad\qquad x_2 + \ y_2 = b_2 \\ \qquad\qquad \ldots \\ \qquad\qquad x_n + \ y_n = b_n \\ \qquad\qquad 0 \le x \le b, x \ \in N^n \\ \qquad\qquad 0 \le y \le b, y \ \in N^n \end{cases} \qquad (1)$$

onde $x_i$ representa a quantidade de itens do i-ésimo bem que a pessoa $A$ recebe, $\mu_x^i$ representa a medida de satisfação da pessoa $A$ referente ao i-ésimo bem. Analogamente, $y_i$ representa a quantidade de itens do i-ésimo bem que a pessoa $B$ recebe, $\mu_y^i$ representa a medida de satisfação da pessoa $B$ referente ao i-ésimo bem, e $b$ é um vetor que define a quantidade de itens de cada bem. Para facilitar o entendimento e sem perda de generalidade, trataremos o vetor $b$ como inteiro e positivo. Caso exista algum bem fracionário, por exemplo, R$ 100,15, basta substituí-lo por sua versão multiplicada por 100, ou seja, 10.015 centavos de real.

Aplicando este método no exemplo do capítulo 3, teríamos:

$$\begin{cases} \max \ 0{,}45.x_1 \ + \ 0{,}15.x_2 + \ 0{,}10.x_3 + \ 0{,}30.x_4 + \\ \qquad + \ 0{,}20.y_1 + \ 0{,}40.y_2 + \ 0{,}30.y_3 + \ 0{,}10.y_4 \\ \text{Sujeito a: } x_1 + \ y_1 = 1 \text{ (1 apartamento)} \\ \qquad\qquad x_2 + \ y_2 = 1 \text{ (1 casa de praia)} \\ \qquad\qquad x_3 + \ y_3 = 1 \text{ (1 carro compacto)} \\ \qquad\qquad x_4 + \ y_4 = 1 \text{ (1 carro sedan)} \\ \qquad\qquad 0 \le x \le 1, x \ \in N^4 \\ \qquad\qquad 0 \le y \le 1, y \ \in N^4 \end{cases} \qquad (2)$$

cuja solução ótima se dá no vértice:

$$\begin{cases} x_1 = x_4 \ = y_2 = y_3 = 1; \\ x_2 = x_3 \ = y_1 = y_4 = 0; \end{cases} \quad (3)$$

justamente como resolvido por inspeção.

Porém este método é falho por não abrager todos os casos possíveis: vamos supor que as medidas $\mu_x$ e $\mu_y$ fossem idênticas (ou seja, ambas partes tivessem as mesmas vontades). Neste caso teríamos inúmeras soluções, como por exemplo:

$$\begin{cases} x_1 = x_2 \ = x_3 = x_4 = 1; \\ y_1 = y_2 \ = y_3 = y_4 = 0; \end{cases} \quad (4)$$

que são ótimas do ponto de vista de otimização, porém não são justas, pois um recebeu tudo e o outro nada.

Para contornar esse problema, podemos exigir que a divisão seja justa criando mais uma restrição:

$$\sum_{i=1}^{n} x_i \, \mu_x^i = \sum_{i=1}^{n} y_i \, \mu_y^i \quad (5)$$

que voltando ao nosso exemplo anterior poderia ser formulada como:

$$0{,}45.x_1 \ + \ 0{,}15.x_2 + \ 0{,}10.x_3 + \ 0{,}30.x_4 = \ 0{,}20.y_1 + \ 0{,}40.y_2 + \ 0{,}30.y_3 + \ 0{,}10.y_4 \quad (6)$$

que implica que a satisfação de ambos deve ser a mesma.

Entretanto o problema com esta restrição não tem solução: vimos que a solução ótima é dada em (3). Desta forma, a restrição apresentada em (6) não é satisfeita, pois:

$$0{,}45 \, . \, 1 \ + 0{,}15.0 + 0{,}10.0 + 0{,}30.1 = \mathbf{0{,}75 \neq 0{,}70} = 0{,}20.0 + 0{,}40.1 + 0{,}30.1 + 0{,}10.0 \quad (7)$$

sendo assim, mais uma vez é necessário repensar a restrição. Com o exposto acima, vemos que o ideal é maximizar a satisfação total, sujeito a termos as satisfações de ambos *próximas*, isto é, $\sum_{i=1}^{n} x_i\, \mu_x^i \cong \sum_{i=1}^{n} y_i\, \mu_y^i$ . Com a inclusão de uma variável de folga, $\delta$, podemos definir nossa nova restrição que garante uma divisão justa:

$$\begin{cases} \sum_{i=1}^{n} x_i\, \mu_x^i - \sum_{i=1}^{n} y_i\, \mu_y^i = \delta \\ \Delta_1 \leq \delta \leq \Delta_2 \end{cases} \qquad (8)$$

Com a impossição dessas duas restrições, nosso modelo passa a aceitar diferença entre a satisfação de ambos, desde que dentro de um limite, no caso entre $\Delta_1$ e $\Delta_2$ .Voltando ao nosso exemplo e com $-\Delta_1 = \Delta_2 = 10\%$ temos:

$$\delta = 0{,}45\,.\,1 + 0{,}15.0 + 0{,}10.0 + 0{,}30.1 - 0{,}20.0 - 0{,}40.1 - 0{,}30.1 - 0{,}10.0 = 5\% \qquad (9)$$
$$-10\% \leq \delta \leq 10\%$$

Como todas restrições foram satisfeitas, conseguimos criar um modelo que faz a melhor divisão possível, garantindo que ambos receberam mais do que acham que mereciam. Apenas um espectador de fora, um juiz no nosso caso, saberia que a diferença entre o que cada um recebeu é da ordem de 5%.

Unindo (1) e (8) e dados $\mu_x$, $\mu_y$, $b$, $\Delta_1$ e $\Delta_2$ chegamos ao modelo completo, que pode ser especificado como:

$$\begin{cases} \max \left(\frac{x' * \mu_x}{b}\right) + \left(\frac{y' * \mu_y}{b}\right) \\ \text{Sujeito a: } x + y = b \\ \left(\frac{x' * \mu_x}{b}\right) - \left(\frac{y' * \mu_y}{b}\right) = \delta \\ x \geq 0, x \in N^n \\ y \geq 0, y \in N^n \\ -\Delta_1 \leq \delta \leq \Delta_2\,, \delta \in R \\ b \in N^n \end{cases} \qquad (10)$$

Vale notar que é necessário realizar a divisão das medidas de satisfação pelas correspondentes componentes do vetor $b$ (a explicação fica como exercício para o leitor).

## 4.3 Usando o Simplex para resolver um exemplo complexo

Digamos que um juiz está responsável por definir a partilha de bens de um casal que construiu um bom patrimônio ao longo dos anos de casado. O primeiro passo é listar os bens e suas respectivas quantidades:

| ID | Bem | Quantidade | Unidade |
|---|---|---|---|
| 1 | Dinheiro | 65000 | R$ |
| 2 | Ação | 22000 | R$ |
| 3 | Apartamento | 1 | un |
| 4 | Terreno | 5000 | m² |
| 5 | Carro | 2 | un |
| 6 | Computador | 3 | un |
| 7 | Objeto de arte | 15 | un |
| 8 | Cachorro | 2 | un |
| 9 | Jóia | 10 | un |
| 10 | Iate | 1 | un |

Porém quando questionados sobre o que acham justo, não têm opinião formada. Sendo assim, o juiz atribui peso de 10% ($\frac{100\%}{10}$) para cada bem e executa o modelo, que pode ser especificado da seguinte forma:

$$
\begin{cases}
\max \frac{0,1}{65000}x_1 + \frac{0,1}{22000}x_2 + \frac{0,1}{1}x_3 + \frac{0,1}{5000}x_4 + \frac{0,1}{2}x_5 + \frac{0,1}{3}x_6 + \frac{0,1}{15}x_7 + \\
\frac{0,1}{2}x_8 + \frac{0,1}{10}x_9 + \frac{0,1}{1}x_{10} + \frac{0,1}{65000}y_1 + \frac{0,1}{22000}y_2 + \frac{0,1}{1}y_3 + \frac{0,1}{5000}y_4 + \\
\frac{0,1}{2}y_5 + \frac{0,1}{3}y_6 + \frac{0,1}{15}y_7 + \frac{0,1}{2}y_8 + \frac{0,1}{10}y_9 + \frac{0,1}{1}y_{10} \\
\\
Sujeito\ a: x_1 + y_1 = 65000 \\
x_2 + y_2 = 22000 \\
x_3 + y_3 = 1 \\
x_4 + y_4 = 5000 \\
x_5 + y_5 = 2 \\
x_6 + y_6 = 3 \\
x_7 + y_7 = 15 \\
x_8 + y_8 = 2 \\
x_9 + y_9 = 10 \\
x_{10} + y_{10} = 1 \\
\\
\frac{0,1}{65000}x_1 + \frac{0,1}{22000}x_2 + \frac{0,1}{1}x_3 + \frac{0,1}{5000}x_4 + \frac{0,1}{2}x_5 + \frac{0,1}{3}x_6 + \frac{0,1}{15}x_7 + \\
+ \frac{0,1}{2}x_8 + \frac{0,1}{10}x_9 + \frac{0,1}{1}x_{10} - \frac{0,1}{65000}y_1 - \frac{0,1}{22000}y_2 - \frac{0,1}{1}y_3 - \frac{0,1}{5000}y_4 \\
- \frac{0,1}{2}y_5 - \frac{0,1}{3}y_6 - \frac{0,1}{15}y_7 - \frac{0,1}{2}y_8 - \frac{0,1}{10}y_9 - \frac{0,1}{1}y_{10} = \delta \\
\\
x \geq 0, y \geq 0, \delta = 0
\end{cases}
\tag{11}
$$

Onde x representa os bens que ficarão com o marido e y com a esposa. Aplicando o algorítmo enunciado em 6.1.5, o resultado deste problema é:

| Exemplo 4.3.1 | Dinheiro | Ação | Apartamento | Terreno | Carro | Computador | Objeto de arte | Cachorro | Jóia | Iate | μ* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| X | 48333 | 0 | 0 | 0 | 1 | 3 | 11 | 2 | 10 | 0 | 52% |
| Y | 16667 | 22000 | 1 | 5000 | 1 | 0 | 4 | 0 | 0 | 1 | 52% |
| Total | 65000 | 22000 | 1 | 5000 | 2 | 3 | 15 | 2 | 10 | 1 | 104% |

*o resultado correto seria 50% para cada, totalizando 100%, porém por arredondamentos nas divisões (por exemplo, 0,1/65000) o resultado acaba sendo ligeiramente alterado.

Em seguida o  juiz pergunta aos interessados se esse resultado agrada a ambos. Digamos que o marido esteja mais interessado no dinheiro e no terreno, enquanto a esposa esteja mais interessada nas obras de arte e jóias, neste caso eles não concordarão com essa divisão e serão questionados pelo juiz a darem um parecer sobre quanto acham que cada bem representa com relação ao todo. Vamos supor que o casal então ajuste sua vontade da seguinte forma:

| ID | Bem | Quantidade | Unidade | $\mu_x$ | $\mu_y$ |
|---|---|---|---|---|---|
| 1 | Dinheiro | 65000 | R$ | 20,0% | 10,0% |
| 2 | Ação | 22000 | R$ | 10,0% | 10,0% |
| 3 | Apartamento | 1 | un | 10,0% | 10,0% |
| 4 | Terreno | 5000 | m² | 17,0% | 10,0% |
| 5 | Carro | 2 | un | 10,0% | 5,0% |
| 6 | Computador | 3 | un | 10,0% | 10,0% |
| 7 | Objeto de arte | 15 | un | 2,0% | 12,5% |
| 8 | Cachorro | 2 | un | 10,0% | 10,0% |
| 9 | Jóia | 10 | un | 1,0% | 12,5% |
| 10 | Iate | 1 | un | 10,0% | 10,0% |

Submetendo esses dados ao modelo, o resultado passa a ser:

| Exemplo 4.3.2 | Dinheiro | Ação | Apartamento | Terreno | Carro | Computador | Objeto de arte | Cachorro | Jóia | Iate | μ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| X | 65000 | 19500 | 0 | 5000 | 2 | 0 | 0 | 1 | 0 | 0 | 61% |
| Y | 0 | 2500 | 1 | 0 | 0 | 3 | 15 | 1 | 10 | 1 | 61% |
| Total | 65000 | 22000 | 1 | 5000 | 2 | 3 | 15 | 2 | 10 | 1 | 122% |

Vemos que o modelo designa todo o dinheiro e o terreno ao marido, enquanto a esposa fica com todas obras de arte e jóia, e ambos continuam se sentindo satisfeitos por igual. Entretando vamos supor que os advogados de defesa da esposa argumentem que ela merece receber 15% a mais do que o marido, tanto porque existem provas de que ela contribuiu mais com a renda familiar, além do que ela ficará com a guarda do filho adolescente.

Com esse novo dado, é necessário alterar apenas a restrição apresentada em (8). Devido à esposa merecer ter 15% a mais, o juiz pode colocar uma folga em torno de $\delta$ a seu critério, digamos 15% ± 2%. Desta forma nossa restrição passa a ser:

$$\begin{cases} x'\mu_x - y'\mu_y = \delta \\ -17\% \leq \delta \leq -13\% \end{cases} \quad (12)$$

Substituindo $\delta = 0$ em (11) pela folga apresentada acima, temos um novo resultado:

| Exemplo 4.3.3 | Dinheiro | Ação | Apartamento | Terreno | Carro | Computador | Objeto de arte | Cachorro | Jóia | Iate | $\mu$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| X | 65000 | 16499 | 0 | 5000 | 2 | 0 | 0 | 0 | 0 | 0 | 55% |
| Y | 0 | 5501 | 1 | 0 | 0 | 3 | 15 | 2 | 10 | 1 | 68% |
| Total | 65000 | 22000 | 1 | 5000 | 2 | 3 | 15 | 2 | 10 | 1 | 122% |

$\delta = -13.0004\%$

Este resultado nos mostra que mesmo a esposa merecendo ganhar mais (13 pontos percentuais a mais), a satisfação de ambos ainda superou os 50%, além de dar os itens de mais interesse aos que mais os queriam: dinheiro e terreno ao marido, e obras de arte e jóias à esposa.

## Capítulo 5 – Conclusões

Através do exposto nos capítulos anteriores, vimos como é possível unir duas ferramentas poderosas, Simplex e algorítmos de divisão, para criar um modelo capaz de beneficiar várias pessoas, sejam elas as que estão envolvidas em um processo de separação, quanto aquelas que estão mediando a situação.

Além disso, é possível notar como assuntos que aparentam ser meramente matemáticos, tem plena aplicação no dia-a-dia e são temas que podem ser facilmente entendidos por alunos do ensino superior.

# Capítulo 6 – Códigos

Neste capítulo estão publicados os códigos em R das variantes do Simplex (capítulo 2), bem como parâmetros de entrada utilizados para calcular os exemplos do capítulo 4.

## 6.1 Códigos: Variantes do Simplex

A apresentação dos códigos seguirá o seguinte padrão:

I. Especificação do problema
II. Parâmetros de entrada
III. Parâmetros de saída
IV. Código
V. Exemplo

### 6.1.1 Primal

**I. Especificação do problema:**

Este código é capaz de resolver problemas do tipo:

$$\textbf{\textit{Min } } c'x$$
$$\textbf{\textit{Sujeito a: }} Ax = b$$
$$x \geq 0$$
$$x \in R^n,\ A \in R^{m \times n},\ b \in R^m$$

## II. Parâmetros de entrada:

- A – Matriz com m linhas (restrições) e n colunas (variáveis);
- c – Vetor de tamanho n;
- b – Vetor de tamanho m;
- I – Vetor de tamanho m, de índices das colunas de A, que forma uma base inicial viável primal ($x \geq 0$).

## III. Parâmetros de saída:

- x – Vetor de tamanho n, representando o vértice ótimo;
- vo – Valor Ótimo do problema;

## IV. Código:

```
# Tamanho do problema
n <- length(c)
m <- length(b)

# Gerando a base inicial B e sua inversa B_inversa
B <- matrix(0,m,m)
B[,1:m] <- A[,I[1:m]]
B_inversa <- solve(B)

# Gerando o vetor x inicial
x <- vector(mode="double",length=n)
x[] <- 0
x[I[1:m]] <- (B_inversa %*% b)[1:m]

# Criando variáveis do problema
vo <- c %*% x
c_basico <- vector(mode="double",length=m)
c_barra <- vector(mode="double",length=n)
u <- vector(mode="double",length=m)
d <- vector(mode="double",length=n)

# Rotina principal. Procura solução ótima ou ilimitação do problema
while(TRUE) {

  c_basico[1:m] <- c[I[1:m]]
  c_barra <- c - c_basico %*% B_inversa %*% A
  c_barra[which(abs(c_barra)<1e-10)] <- 0

  # Checa se já temos uma solução ótima
  solucao_otima=FALSE
```

```
  for (j in 1:n) if (c_barra[j] < 0) break; # regra de Bland
  if (j==n && c_barra[n] >= 0) {solucao_otima=TRUE; break;}

  # Checa se temos um problema ilimitado
  u <- B_inversa %*% A[,j]
  for (i in 1:m) if (u[i] > 0) break;
  if (i==m && u[m] <= 0) { vo =-Inf; break;}

  # Procura maior passo possível
  primeira_iteracao = TRUE
  for (i in 1:m) {
    if (u[i] > 0) {
      if (primeira_iteracao == TRUE) {
        t <- x[I[i]]/u[i]
        k <- i
        primeira_iteracao = FALSE
      }
      if (primeira_iteracao == FALSE) {
        t_temp <- x[I[i]]/u[i]
        if (t_temp < t) {
          t <- t_temp
          k <- i
        }
      }
    }
  }

  # Calcula melhor direção possível e descola o vetor x nesta direção
  d[]=0
  d[j] <- 1
  d[I[1:m]] <- -u[1:m]
  x <- x + t*d

  # Atualiza base
  I[k]=j
  B[,k] <- A[,j]
  B_inversa <- solve(B)
}

# Output
if (solucao_otima == TRUE) {
  vo = c %*% x
  print("Valor de x: ")
  print(x)
  print(paste("Valor ótimo: ", vo))
} else {
  print("Problema ilimitado")
  print(paste("Valor ótimo: ", vo))
}
```

**V. Exemplo:**

**Entrada:**

- $A =$
$$\begin{bmatrix} -13 & 18.8 & -83.5 & 14.9 & -68 & -63.2 & 4.7 & 26.6 & -11.6 & -62.3 \\ 79.7 & 72.7 & -21.8 & -79.6 & 65.5 & 50.2 & -72.3 & -57.9 & -94.9 & 19.8 \\ -71.2 & -80.6 & 25.5 & 90.2 & 71 & 24.9 & 100 & -39 & 48.5 & 53.3 \\ -58.2 & -99.6 & -44.4 & -52 & 38.6 & 55.3 & -16.8 & 3.4 & -34.4 & -31.6 \\ 56.7 & -71.8 & 23.1 & -38 & 1.1 & -44.4 & 26.7 & 82.9 & -98.4 & 73.7 \end{bmatrix}$$
- $c = (-7.7 \quad 55.7 \quad -81.6 \quad 91 \quad -69.6 \quad 78.8 \quad -42.5 \quad 77.5 \quad 49.8 \quad 45.1)$
- $b = (3.1 \quad -61.5 \quad -78.3 \quad -83.1 \quad 44.7)$
- $I = (1 \quad 2 \quad 4 \quad 6 \quad 8)$

**Saída** (valores arredondados até a segunda casa decimal)**:**

- $x = (0 \quad 0.65 \quad 0.43 \quad 0 \quad 0 \quad 0.09 \quad 0 \quad 1.35 \quad 0.27 \quad 0)$
- $vo = 126.04$

## 6.1.2 Dual

**I. Especificação do problema:**

Este código é capaz de resolver problemas do tipo:

***Min c'x***
***Sujeito a: Ax = b***
***x ≥ 0***
***x ∈ $R^n$, A ∈ $R^{m \times n}$, b ∈ $R^m$***

**II. Parâmetros de entrada:**

- A – Matriz com m linhas (restrições) e n colunas (variáveis);

- c – Vetor de tamanho n;
- b – Vetor de tamanho m;
- I – Vetor de tamanho m, de índices das colunas de A, que forma uma base inicial viável dual ($\bar{c} \geq 0$).

**III. Parâmetros de saída:**

- x – Vetor de tamanho n, representando o vértice ótimo;
- vo – Valor Ótimo do problema;

**IV. Código:**

```
# Tamanho do problema
n <- length(c)
m <- length(b)

# Gera a base inicial B e sua inversa B_inversa
B <- matrix(0,m,m)
B[,1:m] <- A[,I[1:m]]
B_inversa <- solve(B)

# Calcula o vetor de custos reduzidos c_barra
c_basico <- vector(mode="double",length=m)
c_basico[1:m] <- c[I[1:m]]
c_barra <- vector(mode="double",length=n)
c_barra <- c - c_basico %*% B_inversa %*% A

# Monta tableau
tableau <- matrix(0,m,n)
tableau <- B_inversa %*% A

# Calcula x basico e o valor otimo do problema
x_basico <- B_inversa %*% b
vo <- x_basico %*% c_basico

# Armazena o primeiro indice l, tal que x_basico(l) < 0
l <- which(x_basico<0)[1]

# Rotina principal. Procura solução ótima ou inviabilidade
while(!is.na(l)) {

  tableau[which(abs(tableau)<1e-10)] <- 0
  x_basico[which(abs(x_basico)<1e-10)] <- 0
  c_barra[which(abs(c_barra)<1e-10)] <- 0

  # Percorre a l-esima linha do tableau e procura por j, tal  que j = argmin
```

```
  c_barra/u_l
    primeira_iteracao = TRUE
    j = 0
    for (k in 1:n) {
      if (tableau[l,k] < 0) {
        temp <- c_barra[k]/(-tableau[l,k])
        if (primeira_iteracao) {
          menor = temp
          primeira_iteracao = FALSE
          j = k
        }
        if (!primeira_iteracao && temp < menor) {
          menor = temp
          j = k
        }
      }
    }
    if (j == 0) {vo = +Inf; break;}

    # Realiza pivotações no tableau
    fator = tableau[l,j]
    tableau[l,] <- tableau[l,]/fator
    x_basico[l] <- x_basico[l]/fator
    for (k in 1:m) if(k!=l) {
      fator = tableau[k,j]
      tableau[k,] <- tableau[k,] - tableau[l,]*fator
      x_basico[k] <- x_basico[k] - x_basico[l]*fator
    }
    c_barra <- c_barra - tableau[l,]*c_barra[j]

    # Atualiza indices básicos e variáveis do problema
    I[l] <- j
    c_basico[1:m] <- c[I[1:m]]
    B[,1:m] <- A[,I[1:m]]
    B_inversa <- solve(B)
    l <- which(x_basico<0)[1]
  }

  # Output
  if (vo[1] != +Inf) {
    vo = c %*% x
    print("Valor de x: ")
    print(x)
    print(paste("Valor ótimo: ", vo))
  } else {
    print("Problema inviavel")
  }
```

**V. Exemplo:**

**Entrada:**

- $A =$
$$\begin{bmatrix} -13 & 18.8 & -83.5 & 14.9 & -68 & -63.2 & 4.7 & 26.6 & -11.6 & -62.3 \\ 79.7 & 72.7 & -21.8 & -79.6 & 65.5 & 50.2 & -72.3 & -57.9 & -94.9 & 19.8 \\ -71.2 & -80.6 & 25.5 & 90.2 & 71 & 24.9 & 100 & -39 & 48.5 & 53.3 \\ -58.2 & -99.6 & -44.4 & -52 & 38.6 & 55.3 & -16.8 & 3.4 & -34.4 & -31.6 \\ 56.7 & -71.8 & 23.1 & -38 & 1.1 & -44.4 & 26.7 & 82.9 & -98.4 & 73.7 \end{bmatrix}$$

- $c = (-7.7 \quad 55.7 \quad -81.6 \quad 91 \quad -69.6 \quad 78.8 \quad -42.5 \quad 77.5 \quad 49.8 \quad 45.1)$

- $b = (3.1 \quad -61.5 \quad -78.3 \quad -83.1 \quad 44.7)$

- $I = (2 \quad 4 \quad 7 \quad 8 \quad 9)$

**Saída** (valores arredondados até a segunda casa decimal)**:**

- $x = (0 \quad 0.65 \quad 0.43 \quad 0 \quad 0 \quad 0.09 \quad 0 \quad 1.35 \quad 0.27 \quad 0)$

- $vo = 126.04$

## 6.1.3 Primal com restrições de caixa

**I. Especificação do problema:**

Este código é capaz de resolver problemas do tipo:

***Min c'x***
***Sujeito a: Ax = b***
***lw ≤ x ≤ up***
***x ∈ $R^n$, lw ∈ $R^n$, up ∈ $R^n$, A ∈ $R^{m \times n}$, b ∈ $R^m$***

**II. Parâmetros de entrada:**

- A – Matriz com m linhas (restrições) e n colunas (variáveis);
- c – Vetor de tamanho n;
- b – Vetor de tamanho m;
- I_b – Vetor de tamanho m, de índices das colunas de A, que forma uma base inicial viável primal ($lw \leq x \leq up$);

- I_lw – Vetor de índices que indica quais variáveis estão em seu limite inferior;
- I_up – Vetor de índices que indica quais variáveis estão em seu limite superior;
- lw – Vetor de tamanho n contendo os limites inferiores de x;
- up – Vetor de tamanho n contendo os limites superiores de x..

**III. Parâmetros de saída:**

- x – Vetor de tamanho n, representando o vértice ótimo;
- vo – Valor Ótimo do problema;

**IV. Código:**

```
# Tamanho do problema
n <- length(c)
m <- length(b)

I_res <- c(I_lw,I_up)
Controle_base <- matrix(0,1,n)
Controle_base[I_b] <- "B"
Controle_base[I_lw] <- "L"
Controle_base[I_up] <- "U"

B <- matrix(A[,I_b],m,m)
B_inversa <- solve(B)

R <- A[,I_res]
x_res <- matrix(NA,1,(n-m))[1,]
for (i in 1:(n-m)) x_res[i] <- if (Controle_base[I_res[i]] == "U") up[I_res[i]] else if
(Controle_base[I_res[i]] == "L") lw[I_res[i]] else 0
x_basico <- B_inversa%*%(b - R%*%x_res)

x <- vector(mode="double",length=n)
x[I_b] <- x_basico
x[I_res] <- x_res

vo <- c %*% x
flip <- FALSE
ilimitado <- FALSE

# Rotina principal. Procura solução ótima ou ilimitação do problema
while(TRUE) {
  B_inversa [which(abs(B_inversa) < 1e-10)] <- 0
  c_barra <- c - c[I_b] %*% B_inversa %*% A
```

```
  c_barra[which(abs(c_barra) < 1e-10)] <- 0

  # verifica se a solução atual é ótima, caso negativo guarda
  # índice q, que representa variável que irá entrar na base
  solucao_otima=FALSE
  q = 0
  menor_c = 0
  maior_c = 0
  for (i in 1:n) {
    if (Controle_base[i] == "L" && c_barra[i] < 0) {
      if (c_barra[i] < menor_c) {menor_c <- c_barra[i]; q = i}
    } else if (Controle_base[i] == "U" && c_barra[i] > 0) {
      if (c_barra[i] > maior_c) {maior_c <- c_barra[i]; q = i}
    } else if (Controle_base[i] == "Z" && c_barra[i] != 0) {
      if (c_barra[i] < menor_c) {menor_c <- c_barra[i]; q = i}
      else if (c_barra[i] > maior_c) {maior_c <- c_barra[i]; q = i}
    }
  }
  if (q == 0) {solucao_otima=TRUE; break;}

  alpha <- B_inversa %*% A[,q]
  lambda <- matrix(c(lw[I_b],lw[q]),1,m+1)[1,]
  mu <- matrix(c(up[I_b],up[q]),1,m+1)[1,]

  if (Controle_base[q] != "U") {
    minimo_1_aux <- matrix(Inf,1,m)[1,]
    for (i in 1:m) minimo_1_aux[i] <- if (alpha[i] > 0) (x[I_b[i]] - lambda[i])/alpha[i]
else if (alpha[i] < 0) (x[I_b[i]] - mu[i])/alpha[i] else minimo_1_aux[i]

    minimo_1 <- min(minimo_1_aux)
    k <- which(minimo_1 == minimo_1_aux)[1]
    minimo_2 <- mu[m+1] - lambda[m+1]

    theta = minimo_1
    if (minimo_2 < theta) theta <- minimo_2
    ilimitado <- theta == Inf
    if (!ilimitado && theta == minimo_1) {
      leave_lower <- alpha[k] > 0
      leave_upper <- alpha[k] < 0
    } else {flip <- !ilimitado && theta == minimo_2}

  } else if (Controle_base[q] != "L") {
    maximo_1_aux <- matrix(-Inf,1,m)[1,]
    for (i in 1:m) maximo_1_aux[i] <- if (alpha[i] < 0) (x[I_b[i]] - lambda[i])/alpha[i]
else if (alpha[i] > 0) (x[I_b[i]] - mu[i])/alpha[i] else maximo_1_aux[i]

    maximo_1 <- max(maximo_1_aux)
    k <- which(maximo_1 == maximo_1_aux)[1]
    maximo_2 <- lambda[m+1] - mu[m+1]
```

```
    theta = maximo_1
    if (maximo_2 > theta) theta <- maximo_2
    ilimitado <- theta == -Inf
    if (!ilimitado && theta == maximo_1) {
      leave_lower <- alpha[k] < 0
      leave_upper <- alpha[k] > 0
    } else {flip <- !ilimitado && theta == maximo_2}
  }

  if (ilimitado) break
  if (flip) {
    if (Controle_base[q] == "L") {
      Controle_base[q] <- "U"
      x[q] <- up[q]
    } else {
      Controle_base[q] <- "L"
      x[q] <- lw[q]
    }
    x_basico <- x_basico - (theta * alpha)
    x[I_b] <- x_basico
    flip <- FALSE

  } else {
    saida <- I_b[k]
    I_b[k] <- q
    Controle_base[q] <- "B"
    if (leave_lower) Controle_base[saida] <- "L"
    if (leave_upper) Controle_base[saida] <- "U"

    I_res <- sort(setdiff(1:n,I_b))
    B <- A[,I_b]
    B_inversa <- solve(B)
    R <- A[,I_res]
    for (i in 1:(n-m)) x_res[i] <- if (Controle_base[I_res[i]] == "U") up[I_res[i]] else
if (Controle_base[I_res[i]] == "L") lw[I_res[i]] else 0
    x_basico <- B_inversa%*%(b - R%*%x_res)
    x[I_b] <- x_basico
    x[I_res] <- x_res
  }
  vo <- vo + theta*c_barra[q]
}

# Output
if (!ilimitado) {
  vo = c %*% x
  print("Valor de x: ")
  print(x)
  print(paste("Valor ótimo: ", vo))
} else {
  print("Problema ilimitado");  print("Valor ótimo: -Inf"); }
```

**V. Exemplo:**

**Entrada:**

- $A =$
$$\begin{bmatrix} -13 & 18.8 & -83.5 & 14.9 & -68 & -63.2 & 4.7 & 26.6 & -11.6 & -62.3 \\ 79.7 & 72.7 & -21.8 & -79.6 & 65.5 & 50.2 & -72.3 & -57.9 & -94.9 & 19.8 \\ -71.2 & -80.6 & 25.5 & 90.2 & 71 & 24.9 & 100 & -39 & 48.5 & 53.3 \\ -58.2 & -99.6 & -44.4 & -52 & 38.6 & 55.3 & -16.8 & 3.4 & -34.4 & -31.6 \\ 56.7 & -71.8 & 23.1 & -38 & 1.1 & -44.4 & 26.7 & 82.9 & -98.4 & 73.7 \end{bmatrix}$$
- $c = (-7.7 \quad 55.7 \quad -81.6 \quad 91 \quad -69.6 \quad 78.8 \quad -42.5 \quad 77.5 \quad 49.8 \quad 45.1)$
- $b = (3.1 \quad -61.5 \quad -78.3 \quad -83.1 \quad 44.7)$
- $I\_b = (1 \quad 2 \quad 3 \quad 4 \quad 5)$
- $I\_lw = (6 \quad 7 \quad 8)$
- $I\_up = (9 \quad 10)$
- $lw = (-1000 \quad -1000 \quad -1000 \quad -1000 \quad -1000)$
- $up = (1000 \quad 1000 \quad 1000 \quad 1000 \quad 1000)$

**Saída** (valores arredondados até a segunda casa decimal)**:**

- $x = (-182.68 \quad -75.55 \quad 1000 \quad -1000 \quad 138.96 \quad -1000 \quad 905.13 \quad -1000 \quad -322.75 \quad -1000)$
- $vo = -441014.10$

## 6.1.4 Dual com restrições de caixa

**I. Especificação do problema:**

Este código é capaz de resolver problemas do tipo:

$$\textit{Min } c'x$$
$$\textit{Sujeito a: } Ax = b$$
$$lw \leq x \leq up$$
$$x \in R^n,\ lw \in R^n,\ up \in R^n,\ A \in R^{m \times n},\ b \in R^m$$

## II. Parâmetros de entrada:

- A – Matriz com m linhas (restrições) e n colunas (variáveis);
- c – Vetor de tamanho n;
- b – Vetor de tamanho m;
- I_b – Vetor de tamanho m, de índices das colunas de A, que forma uma base inicial;
- lw – Vetor de tamanho n contendo os limites inferiores de x;
- up – Vetor de tamanho n contendo os limites superiores de x.

## III. Parâmetros de saída:

- x – Vetor de tamanho n, representando o vértice ótimo;
- vo – Valor Ótimo do problema;

## IV. Código:

```
# Tamanho do problema
n <- length(c)
m <- length(b)

B <- matrix(A[,I_b],m,m)
B_inversa <- solve(B)

c_barra <- vector(mode="double",length=n+m)
c_barra <- (c - c[I_b] %*% B_inversa %*% A)[1,]
c_barra[which(abs(c_barra) < 1e-10)]  <- 0

I_lw <- which(c_barra<0)
I_up <- which(c_barra>0)

I_res <- c(I_lw,I_up)
Controle_base <- matrix(0,1,n)
Controle_base[I_b] <- "B"
Controle_base[I_lw] <- "L"
Controle_base[I_up] <- "U"

R <- A[,I_res]
```

```
x_res <- matrix(NA,1,(n-m))[1,]
for (i in 1:(n-m)) x_res[i] <- if (Controle_base[I_res[i]] == "U") up[I_res[i]] else if
(Controle_base[I_res[i]] == "L") lw[I_res[i]] else 0
x_basico <- B_inversa%*%(b - R%*%x_res)

x <- vector(mode="double",length=n)
x[I_b] <- x_basico
x[I_res] <- x_res

ilimitado <- FALSE
# Rotina principal. Procura solução ótima ou inviabilidade
while(TRUE) {
  B_inversa[which(abs(B_inversa) < 1e-10)]  <- 0
  x[which(abs(x) < 1e-10)]  <- 0
  c_barra[which(abs(c_barra) < 1e-10)]  <- 0

  solucao_otima=FALSE
  q <- I_b[x[I_b] < lw[I_b] | x[I_b] > up[I_b]]
  if (is.na(q[1])) {solucao_otima=TRUE; break;}
  q <- which(max(abs(x[q])) == abs(x))

  w <- matrix(0,1,n)[1,]
  w[I_b] <- 0
  w[I_res] <- (B_inversa[which(I_b == q),]%*%A[,I_res])[1,]

  if (x[q] < lw[q]) {
    J <- I_res[w[I_res] < 0 & x[I_res] < up[I_res]]
    J <- c(J,I_res[w[I_res] > 0 & x[I_res] > lw[I_res]])
  } else  {
    J <- I_res[w[I_res] > 0 & x[I_res] < up[I_res]]
    J <- c(J,I_res[w[I_res] < 0 & x[I_res] > lw[I_res]])
  }
  if (is.na(J[1])) {ilimitado <- TRUE; break;}

  minimo <- +Inf
  j <- 0
  for (l in 1:length(J)) {
    if (w[J[l]] != 0 && abs(c_barra[J[l]]/w[J[l]]) < minimo) {
      minimo <- abs(c_barra[J[l]]/w[J[l]])
      j <- J[l]
    }
  }

  d <- B_inversa%*%A[,j]

  if (x[q] < lw[q]) {
    t <- (x[q] - lw[q])/w[j]
  } else  {
    t <- (x[q] - up[q])/w[j]
  }
```

```
  x[j] <- x[j] + t
  x[I_b] <- x[I_b] - t*d

  c_barra[q] <- -c_barra[j]/w[j]
  c_barra <- c_barra + c_barra[q]*w

  I_b[which(I_b == q)] <- j
  I_res[which(I_res == j)] <- q

  I_b <- sort(I_b)
  I_res <- sort(I_res)

  B_inversa <- solve(A[,I_b])
}

# Output
if (!ilimitado) {
  vo = -c %*% x
  print("Valor de x: ")
  print(x)
  print(paste("Valor ótimo: ", vo))
} else {
  print("Problema inviável");  print("Valor ótimo: Inf"); }
```

## V. Exemplo:

**Entrada:**

- $A =$
$$\begin{bmatrix} -13 & 18.8 & -83.5 & 14.9 & -68 & -63.2 & 4.7 & 26.6 & -11.6 & -62.3 \\ 79.7 & 72.7 & -21.8 & -79.6 & 65.5 & 50.2 & -72.3 & -57.9 & -94.9 & 19.8 \\ -71.2 & -80.6 & 25.5 & 90.2 & 71 & 24.9 & 100 & -39 & 48.5 & 53.3 \\ -58.2 & -99.6 & -44.4 & -52 & 38.6 & 55.3 & -16.8 & 3.4 & -34.4 & -31.6 \\ 56.7 & -71.8 & 23.1 & -38 & 1.1 & -44.4 & 26.7 & 82.9 & -98.4 & 73.7 \end{bmatrix}$$

- $c = (-7.7 \quad 55.7 \quad -81.6 \quad 91 \quad -69.6 \quad 78.8 \quad -42.5 \quad 77.5 \quad 49.8 \quad 45.1)$

- $b = (3.1 \quad -61.5 \quad -78.3 \quad -83.1 \quad 44.7)$

- $I\_b = (1 \quad 2 \quad 3 \quad 4 \quad 5)$

- $lw = (-1000 \quad -1000 \quad -1000 \quad -1000 \quad -1000)$

- $up = (1000 \quad 1000 \quad 1000 \quad 1000 \quad 1000)$

**Saída** (valores arredondados até a segunda casa decimal)**:**

- $x = (-182.68 \quad -75.55 \quad 1000 \quad -1000 \quad 138.96$
  $-1000 \quad 905.13 \quad -1000 \quad -322.75 \quad -1000)$

- $vo = -441014.10$

## 6.1.5 Dual com restrições de caixa e variáveis inteiras

### I. Especificação do problema:

Este código é capaz de resolver problemas do tipo:

$$\begin{aligned} &\textbf{\textit{Min } } c'x \\ &\textbf{\textit{Sujeito a: }} Ax = b \\ &lw \leq x \leq up \\ &x_1 \in N^q,\ x_2 \in R^w,\ q+w=n \\ &lw \in R^n,\ up \in R^n,\ A \in R^{m \times n},\ b \in R^m \end{aligned}$$

### II. Parâmetros de entrada:

- A – Matriz com m linhas (restrições) e n colunas (variáveis);
- c – Vetor de tamanho n;
- b – Vetor de tamanho m;
- I_b – Vetor de tamanho m, de índices das colunas de A, que forma uma base inicial;
- lw – Vetor de tamanho n contendo os limites inferiores de x;
- up – Vetor de tamanho n contendo os limites superiores de x;
- inteiras – Vetor contendo índices das variáveis que devem ser inteiras.

### III. Parâmetros de saída:

- x – Vetor de tamanho n, representando o vértice ótimo;
- vo – Valor Ótimo do problema;

### IV. Código:

```
### Definição de funções ###

# Rotina principal. Procura solução ótima ou inviabilidade
pl_solve <- function(A,b,c,I_b,I_res,x,lw,up) {
  n <- length(c)
  m <- length(b)
  B <- matrix(A[,I_b],m,m)
  B_inversa <- solve(B)
  x_basico <- matrix(x[I_b],1,m)[1,]
  x_res <- matrix(x[I_res],1,(n-m))[1,]
  c_barra <- vector(mode="double",length=n+m)
  ilimitado <- FALSE
  c_barra <- (c - c[I_b] %*% B_inversa %*% A)[1,]
  for (i in 1:n) if (abs(c_barra[i]) < 1.0e-10) c_barra[i]=0

  while(TRUE) {
    for (i in 1:m) for (j in 1:m) if (abs(B_inversa[i,j]) < 1.0e-10) B_inversa[i,j]=0
    for (i in 1:n) if (abs(x[i]) < 1.0e-10) x[i]=0

    solucao_otima=FALSE
    q <- I_b[x[I_b] < lw[I_b] | x[I_b] > up[I_b]]
    if (is.na(q[1])) {solucao_otima=TRUE; break;}
    q <- which(max(abs(x[q])) == abs(x))

    w <- matrix(0,1,n)[1,]
    w[I_b] <- 0
    w[I_res] <- (B_inversa[which(I_b == q),]%*%A[,I_res])[1,]

    if (x[q] < lw[q]) {
      J <- I_res[w[I_res] < 0 & x[I_res] < up[I_res]]
      J <- c(J,I_res[w[I_res] > 0 & x[I_res] > lw[I_res]])
    } else  {
      J <- I_res[w[I_res] > 0 & x[I_res] < up[I_res]]
      J <- c(J,I_res[w[I_res] < 0 & x[I_res] > lw[I_res]])
    }

    if (is.na(J[1])) {ilimitado <- TRUE; break;}

    minimo <- +Inf
    j <- 0
    for (l in 1:length(J)) {
      if (w[J[l]] != 0 && abs(c_barra[J[l]]/w[J[l]]) < minimo) {
        minimo <- abs(c_barra[J[l]]/w[J[l]])
        j <- J[l]
      }
    }

    d <- B_inversa%*%A[,j]
```

```
      if (x[q] < lw[q]) {
        t <- (x[q] - lw[q])/w[j]
      } else  {
        t <- (x[q] - up[q])/w[j]
      }

      x[j] <- x[j] + t
      x[I_b] <- x[I_b] - t*d

      c_barra[q] <- -c_barra[j]/w[j]
      c_barra <- c_barra + c_barra[q]*w

      I_b[which(I_b == q)] <- j
      I_res[which(I_res == j)] <- q

      I_b <- sort(I_b)
      I_res <- sort(I_res)

      B_inversa <- solve(A[,I_b])
    }

    if (ilimitado) {
      Inf
    } else {
      pl <- matrix(data=0,nrow=2,ncol=n)
      pl[1,] <- x
      y <- matrix(data=0,nrow=1,ncol=n)[1,]
      y[I_res] <- -1
      pl[2,] <- y
      pl
    }
  }

  # Função recursiva que cria dois subproblemas a cada chamada,
  # procurando qual deles contém a melhor solução
  ramifica <- function(x,ind,lw,up,inteiras) {
    if(!is.na((which(ehInteiro(x[inteiras])==FALSE))[1])) {
      var <- (which(ehInteiro(x[inteiras])==FALSE))[1]

      I_b <- which(ind==0)
      I_res <- which(ind!=0)

      lw_aux <- lw
      up_aux <- up

      lw_aux[var] <- lw[var]
      up_aux[var] <- floor(x[var])
      x1 <- pl_solve(A,b,c,I_b,I_res,x,lw_aux,up_aux)
      if (x1[1] != Inf) {
        lw_aux[var] <- x1[1,var]
```

```
      up_aux[var] <- x1[1,var]
      vl_1 <- ramifica(x1[1,],x1[2,],lw_aux,up_aux,inteiras)
    } else {
      vl_1 <- Inf
    }

    lw_aux[var] <- ceiling(x[var])
    up_aux[var] <- up[var]
    x2 <- pl_solve(A,b,c,I_b,I_res,x,lw_aux,up_aux)
    if (x2[1] != Inf) {
      lw_aux[var] <- x2[1,var]
      up_aux[var] <- x2[1,var]
      vl_2 <- ramifica(x2[1,],x2[2,],lw_aux,up_aux,inteiras)
    } else {
      vl_2 <- Inf
    }

    if(vl_1[1] != Inf) {prob1 <- (-c%*%vl_1)[1,1]} else {prob1=Inf}
    if(vl_2[1] != Inf) {prob2 <- (-c%*%vl_2)[1,1]} else {prob2=Inf}

    if (prob1 < prob2) {
      vl_1
    } else {
      vl_2
    }
  } else {
    x
  }
}


# Função para checar se o numero é inteiro
ehInteiro <- function(x, tol = .Machine$double.eps^0.5)  abs(x - round(x)) < tol

### Fim definição de funções ###

## Início do script ##

n <- length(c)
m <- length(b)

B <- matrix(A[,I_b],m,m)
B_inversa <- solve(B)

c_barra <- vector(mode="double",length=n+m)
c_barra <- (c - c[I_b] %*% B_inversa %*% A)[1,]
c_barra[which(abs(c_barra) < 1e-10)] <- 0

I_lw <- which(c_barra<0)
I_up <- which(c_barra>0)
```

```
  I_res <- 0
  for (i in 1:n) {
    if(is.na(which(I_b==i)[1])) I_res <- if(I_res[1] == 0) i else c(I_res,i)
  }

  R <- matrix(A[,I_lw],m,length(I_lw))
  xr <- lw[I_lw]
  S <- matrix(A[,I_up],m,length(I_up))
  xs <- up[I_up]

  x_basico <- (B_inversa %*% (b - R %*% xr - S %*% xs))[,1]

  x <- vector(mode="double",length=n)
  x[I_b] <- x_basico
  x[I_lw] <- xr
  x[I_up] <- xs

  # Chama rotina principal – resultado é um vetor sem restrições de variáveis
  inteiras
  x <- pl_solve(A,b,c,I_b,I_res,x,lw,up)

  # Chama função recursive que irá restringir que algumas variáveis sejam
  inteiras
  y <- ramifica(x[1,],x[2,],lw,up,inteiras)

  # Output
  if (y[1] != Inf) {
    vo = c %*% y
    print("Valor de x: ")
    print(y)
    print(paste("Valor ótimo: ", vo))
  } else {
print("Problema inviável");}
```

## V. Exemplo:

**Entrada:**

- $A =$
$$\begin{bmatrix} -13 & 18.8 & -83.5 & 14.9 & -68 & -63.2 & 4.7 & 26.6 & -11.6 & -62.3 \\ 79.7 & 72.7 & -21.8 & -79.6 & 65.5 & 50.2 & -72.3 & -57.9 & -94.9 & 19.8 \\ -71.2 & -80.6 & 25.5 & 90.2 & 71 & 24.9 & 100 & -39 & 48.5 & 53.3 \\ -58.2 & -99.6 & -44.4 & -52 & 38.6 & 55.3 & -16.8 & 3.4 & -34.4 & -31.6 \\ 56.7 & -71.8 & 23.1 & -38 & 1.1 & -44.4 & 26.7 & 82.9 & -98.4 & 73.7 \end{bmatrix}$$

- $c = (-7.7 \quad 55.7 \quad -81.6 \quad 91 \quad -69.6 \quad 78.8 \quad -42.5 \quad 77.5 \quad 49.8 \quad 45.1)$

- $b = (3.1 \quad -61.5 \quad -78.3 \quad -83.1 \quad 44.7)$
- $I\_b = (1 \quad 2 \quad 3 \quad 4 \quad 5)$
- $lw = (-1000 \quad -1000 \quad -1000 \quad -1000 \quad -1000)$
- $up = (1000 \quad 1000 \quad 1000 \quad 1000 \quad 1000)$
- $inteiras = (1 \quad 2 \quad 3 \quad 4)$

**Saída** (valores arredondados até a segunda casa decimal)**:**

- $x = (-183 \quad -75 \quad 1000 \quad -1000 \quad 138.87$
  $-999.35 \quad 905.60 \quad -999.43 \quad -323.02 \quad -1000)$
- $vo = -440913.10$

## 6.2 Parâmetros de entrada

Os parâmetros de entrada enunciados nos tópicos a seguir devem ser carregados na memória do R e na sequência utilizar o código enunciado em 6.1.5.

## 6.2.1 Exemplo 4.3.1

- c <- c(-0.000002, -0.000005, -0.1, -0.00002, -0.05, -0.033333, -0.006667, -0.05, -0.01, -0.1, -0.000002, -0.000005, -0.1, -0.00002, -0.05, -0.033333, -0.006667, -0.05, -0.01, -0.1, 0)
- b <- c(65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 0)
- lw <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.001)
- up <- c(65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 0.001)
- I_b <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
- Inteiras <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21)
- A <- matrix(NA,11,21)

- A[1,] <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- A[2,] <- c(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- A[3,] <- c(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
- A[4,] <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
- A[5,] <- c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
- A[6,] <- c(0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
- A[7,] <- c(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
- A[8,] <- c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
- A[9,] <- c(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)
- A[10,] <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
- A[11,] <- c(0.000002, 0.000005, 0.1, 0.00002, 0.05, 0.033333, 0.006667, 0.05, 0.01, 0.1, -0.000002, -0.000005, -0.1, -0.00002, -0.05, -0.033333, -0.006667, -0.05, -0.01, -0.1, -1)

## 6.2.1 Exemplo 4.3.2

- c <- c(-0.000003, -0.000005, -0.1, -0.000034, -0.05, -0.033333, -0.001333, -0.05, -0.001, -0.1, -0.000002, -0.000005, -0.1, -0.00002, -0.025, -0.033333, -0.008333, -0.05, -0.0125, -0.1, 0)
- b <- c(65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 0)
- lw <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.00001)
- up <- c(65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 0.00001)
- I_b <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
- Inteiras <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
- A <- matrix(NA,11,21)
- A[1,] <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- A[2,] <- c(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- A[3,] <- c(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
- A[4,] <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
- A[5,] <- c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
- A[6,] <- c(0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
- A[7,] <- c(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
- A[8,] <- c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
- A[9,] <- c(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)
- A[10,] <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
- A[11,] <- c(0.000003, 0.000005, 0.1, 0.000034, 0.05, 0.033333, 0.001333, 0.05, 0.001, 0.1, -0.000002, -0.000005, -0.1, -0.00002, -0.025, -0.033333, -0.008333, -0.05, -0.0125, -0.1, -1)

## 6.2.1 Exemplo 4.3.3

- c <- c(-0.000003, -0.000005, -0.1, -0.000034, -0.05, -0.033333, -0.001333, -0.05, -0.001, -0.1, -0.000002, -0.000005, -0.1, -0.00002, -0.025, -0.033333, -0.008333, -0.05, -0.0125, -0.1, 0)
- b <- c(65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 0)
- lw <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -0.17)
- up <- c(65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, 65000, 22000, 1, 5000, 2, 3, 15, 2, 10, 1, -0.13)
- I_b <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)
- Inteiras <- c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20)
- A <- matrix(NA,11,21)
- A[1,] <- c(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- A[2,] <- c(0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0)
- A[3,] <- c(0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0)
- A[4,] <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0)
- A[5,] <- c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0)
- A[6,] <- c(0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0)
- A[7,] <- c(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0)
- A[8,] <- c(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
- A[9,] <- c(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0)
- A[10,] <- c(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)
- A[11,] <- c(0.000003, 0.000005, 0.1, 0.000034, 0.05, 0.033333, 0.001333, 0.05, 0.001, 0.1, -0.000002, -0.000005, -0.1, -0.00002, -0.025, -0.033333, -0.008333, -0.05, -0.0125, -0.1, -1)

## Referências Bibliográficas


[Ste07] - Stern, J.M., et al – Otimização e Processos Estocásticos Aplicados à Economia e Finanças, 1996-2008.

[Rob98] - Robertson, J., Webb, W. – Cake-Cutting Algorithms, Be Fair If You Can, 1998.

[Dan63] - Dantzig, G.B. – Linear Programming and Extensions, 1963.

[Ber97] - Bertsimas, D. and Tsitsiklis, J.N. – Introduction to Linear Optimization, 1997.

[Chv83] - Chvátal, V. – Linear Programming, 1983.

[Baz77] - Bazaraa, M.S. and Jarvis J.J – Linear Programming and Network Flows, 1977.

[IBGE10] - Instituto Brasileiro de Geografia e Estatística (IBGE) – Estatísticas do Registro Civíl, vol. 37, 2010.