Técnicas computacionais para a implementação eficiente e estável de métodos tipo simplex

Pedro Augusto Munari Junior

Técnicas computacionais para a implementação eficiente e estável de métodos tipo simplex[1]

Pedro Augusto Munari Junior

Orientador: *Prof. Dr. Marcos Nereu Arenales*

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional.

USP - São Carlos
Janeiro de 2009

[1]Este trabalho foi realizado com o apoio da FAPESP

A Jesus Cristo,
dedico e agradeço.

Resumo

Métodos tipo simplex são a base dos principais *softwares* utilizados na resolução de problemas de otimização linear. A implementação computacional direta destes métodos, assim como são descritos na teoria, leva a resultados indesejáveis na resolução de problemas reais de grande porte. Assim, a utilização de técnicas computacionais adequadas é fundamental para uma implementação eficiente e estável. Neste trabalho, as principais técnicas são discutidas, com enfoque naquelas que buscam proporcionar a estabilidade numérica do método: utilização de tolerâncias, estabilização do teste da razão, mudança de escala e representação da matriz básica. Para este último tópico, são apresentadas duas técnicas, a Forma Produto da Inversa e a Decomposição LU. A análise das abordagens é feita baseando-se na resolução dos problemas da biblioteca Netlib.

Palavras-chave: otimização linear; métodos tipo simplex; decomposição LU; mudança de escala; esparsidade.

Abstract

Simplex-type methods are the basis of the main linear optimization solvers. The straightforward implementation of these methods as they are presented in theory yield unexpected results in solving real-life large-scale problems. Hence, it is essencial to use suitable computational techniques for an efficient and stable implementation. In this thesis, we address the main techniques focusing on those which aim for numerical stability of the method: use of tolerances, stable ratio test, scaling and representation of the basis matrix. For the latter topic, we present two techniques, the Product Form of Inverse and the LU decomposition. The Netlib problems are solved using the approaches addressed and the results are analyzed.

Keywords: linear optimization; simplex type methods; LU decomposition; scaling; sparsity.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Métodos tipo simplex são a base dos principais *softwares* utilizados na resolução de problemas de otimização linear. Apesar da complexidade exponencial associada, estes métodos são bastante eficientes na prática. Atualmente, problemas de grande porte, contendo centenas de milhares de restrições e variáveis, podem ser resolvidos em poucos segundos pelos *softwares* disponíveis (Bixby, 2002).

A cada ano, problemas com dimensões cada vez maiores são resolvidos graças aos avanços em tecnologia e ao aprimoramento das implementações computacionais. Baseando-se no software CPLEX (ILOG Inc., 2008), Bixby (2002) relata que, no período de 1992 a 2002, teve-se um ganho de seis ordens de magnitude no tempo computacional para a resolução de problemas de otimização linear. De acordo com o autor, três ordens de magnitude correspondem aos avanços em *hardware* e, os outros três, à evolução dos algoritmos e das técnicas de implementação.

Desde a apresentação do método primal simplex por George B. Dantzig, em 1947, outros métodos utilizando diferentes abordagens foram propostos com o mesmo objetivo, conforme revisado por Todd (2002). Dentre estes, apenas os métodos de pontos interiores, em especial o método primal-dual de barreira logarítmica (Wright, 1997), são atualmente competitivos em relação aos métodos tipo simplex (Bixby, 2002).

As primeiras implementações computacionais do método primal simplex foram realizadas no início da década de 50 e não se mostraram encorajadoras. A tecnologia computacional e as linguagens de programação estavam em estágio bastante prematuro e, além disso, o método era implementado assim como dado na teoria, utilizando a representação explícita da inversa da matriz básica, recalculada a cada iteração. Até mesmo para os computadores atuais, esta abordagem se torna inviável na resolução de problemas de grande porte.

O primeiro salto na eficiência e estabilidade das implementações ocorreu em 1954, com a utilização da Forma Produto da Inversa para a representação da inversa da matriz básica. Com isto, observou-se que a utilização de técnicas adequadas para a implementação do método eram essenciais. Desde então, um grande número de trabalhos tem sido publicado, nos quais implementações eficientes e estáveis de métodos tipo simplex têm sido propostas, envolvendo diversos aspectos como estratégias de *pricing*, heurísticas para o tratamento da degeneração, estruturas de dados adequadas, atualização eficiente da representação da matriz básica, pré-processamento do problema, entre outros.

Nesta dissertação, são apresentadas as principais técnicas para a implementação eficiente e estável de métodos tipo simplex, considerando a resolução de problemas reais

de grande porte. O enfoque principal é dado para as estruturas de dados adequadas e para as técnicas que devem compor o núcleo numérico, que são consideradas a base do *software*, pois o sucesso das demais técnicas é totalmente dependente da eficiência destas. Por exemplo, por melhor que seja a estratégia de *pricing* adotada, o método tipo simplex utilizado será ineficiente se uma técnica adequada de representação da matriz básica não for utilizada.

No Capítulo 2, são introduzidos os fundamentos de otimização linear com o intuito de situar o leitor e definir a notação utilizada nos demais capítulos. Algumas considerações gerais a respeito da implementação de *softwares* de otimização são apresentadas no Capítulo 3, seguidas de uma discussão sobre as técnicas para a implementação eficiente e estável de métodos tipo simplex, sempre se indicando as referências bibliográficas para o leitor interessado em se aprofundar em algum tópico.

As estruturas de dados têm um papel fundamental na implementação de métodos tipo simplex e são descritas no Capítulo 4. Nos capítulos seguintes, é tratado o núcleo numérico da implementação, formado pelas técnicas que buscam manter a estabilidade numérica do método. A utilização de diferentes tolerâncias nas comparações de valores reais, a estabilização do teste da razão e a mudança de escala são descritas no Capítulo 5. No Capítulo 6, são apresentadas duas técnicas para a representação da matriz básica, a Forma Produto da Inversa e Técnica SSLU. Esta última, proposta por Suhl e Suhl (1990, 1993), é reconhecida como o estado-da-arte na representação da matriz básica, sendo utilizada nos principais *softwares* de otimização linar.

Para se verificar a importância das técnicas descritas, são apresentados os resultados de testes computacionais realizados com os problemas da biblioteca Netlib, no Capítulo 7. As conclusões finais e algumas propostas de trabalhos futuros estão no Capítulo 8.

O conteúdo aqui apresentado tem como objetivo levantar as principais propostas da literatura e auxiliar o leitor que pretende implementar algum método tipo simplex. O método primal simplex é utilizado para ilustrar as técnicas, mas a extensão para os outros métodos é direta. A exposição das técnicas do núcleo numérico buscam complementar a literatura, já que os trabalhos disponíveis são bastante sucintos, podendo dificultar a compreensão. Considera-se que o leitor seja familiarizado com os conceitos básicos de otimização linear, a teoria de métodos tipo simplex e a programação de computadores.

Capítulo 2

Fundamentos de otimização linear

Um problema de otimização linear (PL) consiste na minimização ou maximização de uma função linear sujeita a um número finito de restrições lineares. Por convenção, a forma padrão de um PL é dada por

$$\begin{aligned} \text{minimizar} \quad & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{sujeito a} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned} \tag{2.1}$$

sendo $\mathbf{A}$ uma matriz real $m \times n$ com $posto(\mathbf{A}) = m$, $\mathbf{x}$, $\mathbf{c}$, $\mathbf{l}$ e $\mathbf{u}$ vetores n-dimensionais, $\mathbf{b}$ um vetor m-dimensional e $m < n$. A função linear $f(\mathbf{x})$ recebe o nome de *função objetivo* e as equações do sistema linear $\mathbf{A}\mathbf{x} = \mathbf{b}$ são chamadas de restrições do problema. Os vetores $\mathbf{l}$ e $\mathbf{u}$, com $\mathbf{l} \leq \mathbf{u}$, podem ter componentes reais, $-\infty$ e $+\infty$ e correspondem, respectivamente, aos limitantes inferiores e superiores de cada variável do problema. Por esta razão, as variáveis são ditas *canalizadas*.

O conjunto $\mathcal{P} = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$ recebe o nome de *região factível* e todo vetor $\mathbf{x} \in \mathcal{P}$ é chamado de *solução factível*. Se $\mathcal{P} \neq \varnothing$ então o problema (2.1) é *factível*. Caso contrário, tem-se um problema *infactível*, sem solução. Se existir $\mathbf{x}^* \in \mathcal{P}$ tal que $f(\mathbf{x}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \mathcal{P}$, então $\mathbf{x}^*$ é *solução ótima* do problema (2.1). Se para qualquer $\mathbf{x} \in \mathcal{P}$ existir um outro $\bar{\mathbf{x}} \in \mathcal{P}$ tal que $f(\bar{\mathbf{x}}) < f(\mathbf{x})$, então o problema é *ilimitado* e, logo, não possui solução ótima.

Em geral, um problema na forma (2.1) possui diversas *variáveis lógicas*, utilizadas para expressar restrições de desigualdade na forma de equações. Estas variáveis, também conhecidas como variáveis de folga ou excesso, possuem coeficientes nulos em $\mathbf{c}$ e correspondem a colunas unitárias de $\mathbf{A}$, isto é, colunas cujo único elemento não-nulo é igual a 1 ou -1. As colunas associadas às variáveis lógicas são chamadas de *colunas lógicas*. As demais variáveis do problema são chamadas de *variáveis estruturais* e estão associadas às *colunas estruturais* de $\mathbf{A}$. No restante deste trabalho, $\bar{m}$ e $\bar{n}$ representam, respectivamente, o número de variáveis lógicas e o número de variáveis estruturais do problema, sendo $\bar{m} \leq m$ e $\bar{m} + \bar{n} = n$.

Seja $\mathcal{C} = \{1, 2, \ldots, n\}$ o conjunto de índices das variáveis do problema. Considerando a classificação de variáveis como estruturais e lógicas, os índices em $\mathcal{C}$ podem ser particionados em dois subconjuntos ordenados $\mathcal{S}$ e $\mathcal{L}$ formados por índices de variáveis estruturais e lógicas, respectivamente, definidos por:

$$\mathcal{S} = \{1, \ldots, \bar{n}\} \text{ e } \mathcal{L} = \{\bar{n} + 1, \ldots, n\},\ \text{com } |\mathcal{S}| = \bar{n},\ |\mathcal{L}| = \bar{m}. \tag{2.2}$$

Dependendo da forma como o problema é resolvido, pode ser necessária a introdução de *variáveis artificiais*. Para cada variável artificial introduzida, aumenta-se uma coluna unitária na matriz de coeficientes, chamada de *coluna artificial*. O conjunto de índices de variáveis artificiais é denotado por $\mathcal{A}$.

As variáveis de um PL na forma padrão também podem ser classificadas de acordo com seus limitantes, como mostrado na Tabela 2.1.

Limitantes	Tipo da variável x_i	Símbolo
$l_i = u_i$	Fixa	FX
$l_i > -\infty$ e $u_i = +\infty$	Limitada inferiormente	LO
$l_i = -\infty$ e $u_i < +\infty$	Limitada superiormente	UP
$l_i > -\infty$ e $u_i < +\infty$	Limitada	BD
$l_i = -\infty$ e $u_i = 0$	Não-positiva	MI
$l_i = 0$ e $u_i = +\infty$	Não-negativa	PL
$l_i = -\infty$ e $u_i = +\infty$	Livre	FR

Tabela 2.1: Classificação das variáveis de acordo com seus limitantes.

2.1 Formas equivalentes

Todo PL pode ser descrito na forma (2.1), a qual é bastante utilizada na literatura relacionada à implementação de métodos tipo simplex. Entretanto, existem outras formas equivalentes pelas quais um PL pode ser descrito. Algumas permitem a exposição mais simplificada da teoria, enquanto outras têm o intuito de explorar certas particularidades durante a resolução do problema e levar a técnicas mais eficientes.

De modo geral, diversos tipos de restrições podem surgir ao se modelar um PL, como restrições na forma de desigualdade que podem possuir limitantes superiores ou inferiores, ou ambos. Para representar este problema na forma (2.1) é necessário que as restrições de desigualdade sejam convertidas em equações por meio da adição de variáveis lógicas. Para evitar a introdução de novas variáveis ao problema, é possível representá-lo de modo mais conveniente na *forma geral*, dada por:

$$\begin{aligned} \text{minimizar} \quad & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{sujeito a} \quad & \mathbf{L} \leq \mathbf{A}\mathbf{x} \leq \mathbf{U} \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

sendo $\mathbf{L}$ e $\mathbf{U}$ vetores de limitantes que podem ser componentes reais, $-\infty$ e $+\infty$, com $\mathbf{L} \leq \mathbf{U}$. Além disso, é nesta forma que problemas de otimização linear surgem como subproblemas na resolução de problemas de otimização inteira por métodos de enumeração implícita (Wolsey, 1998).

Textos cujo intuito é a introdução da teoria de otimização linear, baseiam-se na *forma padrão com variáveis não-negativas*, dada por:

$$\begin{aligned} \text{minimizar} \quad & f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} \\ \text{sujeito a} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

Esta forma tem como conveniência as variáveis do problema serem limitadas apenas inferiormente e por um limitante nulo. Tal característica resulta em uma notação simplificada que facilita a apresentação da teoria e de métodos tipo simplex, quando comparada à forma padrão (2.1). Entretanto, quanto à implementação computacional, exige que os limitantes sobre as variáveis do problema sejam convertidos em restrições de igualdade e adicionados ao sistema de restrições, resultando no aumento da matriz de coeficientes do problema.

2.2 Dualidade

A todo PL está associado um outro PL, chamado de *problema dual*, e as propriedades envolvendo os dois problemas são extremamente importantes e correspondem aos fundamentos da teoria de otimização linear. Considere um PL na forma (2.1), contendo apenas limitantes finitos sobre suas variáveis. Tem-se o seguinte problema dual associado:

$$\begin{array}{rc} \text{maximizar} & g(\mathbf{y},\mathbf{v},\mathbf{w}) = \mathbf{b}^T\mathbf{y} + \mathbf{l}^T\mathbf{v} - \mathbf{u}^T\mathbf{w} \\ \text{sujeito a} & \mathbf{A}^T\mathbf{y} + \mathbf{v} - \mathbf{w} = \mathbf{c} \\ & \mathbf{v} \geq \mathbf{0},\ \mathbf{w} \geq \mathbf{0} \end{array} \tag{2.3}$$

com $\mathbf{A}$, $\mathbf{c}$, $\mathbf{b}$, $\mathbf{l}$ e $\mathbf{u}$ conforme definidos anteriormente, $\mathbf{y}$ um vetor m-dimensional, $\mathbf{v}$ e $\mathbf{w}$ vetores n-dimensionais. O vetor $(\mathbf{y},\mathbf{v},\mathbf{w})$ é chamado de vetor de variáveis duais.

Neste contexto, o problema (2.1) recebe o nome de *problema primal* e, juntamente com o seu problema dual (2.3), são chamados de *par primal-dual*. Por ser um PL, o problema (2.3) também possui um problema dual associado, dado pelo problema (2.1).

As variáveis do problema dual estão relacionadas com as restrições e os limitantes do problema primal. As variáveis $\mathbf{y}$ estão associadas às restrições $\mathbf{Ax} = \mathbf{b}$, enquanto as variáveis $\mathbf{v}$ e $\mathbf{w}$ estão associadas aos limitantes inferiores $\mathbf{l} \leq \mathbf{x}$ e superiores $\mathbf{x} \leq \mathbf{u}$, respectivamente. Para um PL na forma (2.1) contendo componentes não finitas em $\mathbf{l}$ e $\mathbf{u}$, as variáveis duais associadas são fixadas em zero, isto é,

$$\begin{cases} v_i = 0, & \text{se } l_i = -\infty, \\ w_i = 0, & \text{se } u_i = +\infty, \end{cases} \tag{2.4}$$

para $i = 1,\ldots,n$. A rigor, tais variáveis podem ser removidas do problema dual. Entretanto, essa remoção dificulta a exposição do problema dual e, dessa maneira, assume-se que as variáveis serão fixadas em zero.

A *região factível dual* é dada pelo conjunto $\mathcal{D} = \{(\mathbf{y},\mathbf{v},\mathbf{w}) \in \mathbb{R}^{m+2n} \mid \mathbf{A}^T\mathbf{y}+\mathbf{v}-\mathbf{w} = \mathbf{c},\ \mathbf{v} \geq \mathbf{0},\ \mathbf{w} \geq \mathbf{0}\}$. Todo vetor $(\mathbf{y},\mathbf{v},\mathbf{w}) \in \mathcal{D}$ é chamado de *solução dual factível*. Se existir uma solução $(\mathbf{y}^*,\mathbf{v}^*,\mathbf{w}^*) \in \mathcal{D}$ tal que $g(\mathbf{y}^*,\mathbf{v}^*,\mathbf{w}^*) \geq g(\mathbf{y},\mathbf{v},\mathbf{w}), \forall (\mathbf{y},\mathbf{v},\mathbf{w}) \in \mathcal{D}$, então $(\mathbf{y}^*,\mathbf{v}^*,\mathbf{w}^*)$ é *solução ótima dual*. Os valores das soluções dual factíveis estão relacionados com os valores das soluções factíveis do problema primal, como apresentado no Teorema 2.1.

Teorema 2.1. *Considere o PL descrito em (2.1) e seu dual correspondente dado por (2.3). Supondo $\mathcal{P} \neq \varnothing$ e $\mathcal{D} \neq \varnothing$, tem-se que*

$$g(\mathbf{y},\mathbf{v},\mathbf{w}) \leq f(\mathbf{x}),$$

para quaisquer $\mathbf{x} \in \mathcal{P}$ e $(\mathbf{y},\mathbf{v},\mathbf{w}) \in \mathcal{D}$.

Demonstração: Sejam $\mathbf{x} \in \mathcal{P}$ e $(\mathbf{y}, \mathbf{v}, \mathbf{w}) \in \mathcal{D}$ soluções primal e dual factíveis arbitrárias, respectivamente. A função objetivo do problema dual pode, então, ser reescrita da seguinte maneira:

$$\begin{aligned} g(\mathbf{y}, \mathbf{v}, \mathbf{w}) &= \mathbf{b}^T\mathbf{y} + \mathbf{l}^T\mathbf{v} - \mathbf{u}^T\mathbf{w} \\ &= \mathbf{x}^T\mathbf{A}^T\mathbf{y} + \mathbf{l}^T\mathbf{v} - \mathbf{u}^T\mathbf{w} \\ &= \mathbf{x}^T(\mathbf{c} - \mathbf{v} + \mathbf{w}) + \mathbf{l}^T\mathbf{v} - \mathbf{u}^T\mathbf{w} \\ &= \mathbf{c}^T\mathbf{x} + (\mathbf{l}^T - \mathbf{x}^T)\mathbf{v} + (\mathbf{x}^T - \mathbf{u}^T)\mathbf{w}. \end{aligned}$$

Por $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$, $\mathbf{v} \geq 0$ e $\mathbf{w} \geq 0$ tem-se que $(\mathbf{l}^T - \mathbf{x}^T)\mathbf{v} \leq 0$ e $(\mathbf{x}^T - \mathbf{u}^T)\mathbf{w} \leq 0$ (observe que estes termos existem apenas quando o limitante associado é finito). Logo, $g(\mathbf{y}, \mathbf{v}, \mathbf{w}) \leq \mathbf{c}^T\mathbf{x} = f(\mathbf{x})$. ■

De acordo com o Teorema 2.1, o valor de qualquer solução dual factível é um limitante inferior para o valor da solução ótima do problema primal. Note que se o problema dual é factível porém ilimitado, então o problema primal é, necessariamente, infactível. Caso contrário, haveria pelo menos uma solução primal factível para satisfazer a desigualdade dada pelo teorema. Analogamente, se o problema primal é ilimitado, então o problema dual deve ser infactível.

O problema dual associado ao PL (2.1) também pode ser descrito da seguinte forma:

$$\begin{aligned} \text{maximizar} \quad & g(\mathbf{y}, \mathbf{s}) = \mathbf{b}^T\mathbf{y} + \textstyle\sum_{i=1}^{n} \theta_i(s_i) \\ \text{sujeito a} \quad & \mathbf{A}^T\mathbf{y} + \mathbf{s} = \mathbf{c} \end{aligned} \tag{2.5}$$

com $\mathbf{A}$, $\mathbf{c}$, $\mathbf{b}$, $\mathbf{l}$, $\mathbf{u}$ e $\mathbf{y}$ conforme definidos anteriormente e $\mathbf{s}$ um vetor n-dimensional. Nesse caso, $(\mathbf{y}, \mathbf{s})$ corresponde ao vetor de variáveis duais. A função objetivo deste problema é linear por partes, dada a presença das funções $\theta_i(s_i)$, $i = 1, \ldots, n$, definidas por

$$\theta_i(s_i) = \begin{cases} s_i u_i, & \text{se } s_i < 0, \\ 0, & \text{se } s_i = 0, \\ s_i l_i, & \text{se } s_i > 0. \end{cases} \tag{2.6}$$

A princípio, a linearidade por partes da função objetivo pode parecer um fator complicante. Entretanto, esta característica pode ser explorada em métodos tipo dual simplex, resultando na redução do número de iterações e do tempo computacional (Maros, 2003c; Sousa et al., 2005).

2.3 Soluções básicas

Por hipótese, a matriz de coeficientes satisfaz posto$(\mathbf{A}) = m$. Assim, é possível definir um conjunto contendo m colunas de $\mathbf{A}$ que seja linearmente independente. A esse conjunto dá-se o nome de *base*. Seja $\mathcal{B} \subset \mathcal{C}$ o conjunto ordenado de m índices correspondentes às colunas selecionadas para compor uma base, e $\mathcal{N} \subset \mathcal{C}$ o conjunto ordenado dado por $\mathcal{N} = \mathcal{C} \setminus \mathcal{B}$. Os elementos em $\mathcal{B}$ recebem o nome de *índices básicos*, enquanto aqueles em $\mathcal{N}$ são chamados de *índices não-básicos*. Denota-se por $\mathbf{A}_\mathcal{B}$ a matriz formada por colunas de $\mathbf{A}$ cujos índices estão em $\mathcal{B}$. De forma análoga, tem-se a matriz $\mathbf{A}_\mathcal{N}$.

A definição de uma base induz uma *partição básica* $\mathbf{A} = [\mathbf{B} \mid \mathbf{N}]$, com $\mathbf{B} = \mathbf{A}_\mathcal{B}$ e $\mathbf{N} = \mathbf{A}_\mathcal{N}$, considerando-se uma permutação implícita de colunas da matriz $\mathbf{A}$, sem perda de generalidade. A matriz $\mathbf{B}$ recebe o nome de *matriz básica* e $\mathbf{N}$, de *matriz não-básica*.

Tem-se também a partição $\mathbf{x} = (\mathbf{x}_\mathcal{B}, \mathbf{x}_\mathcal{N})^T$ sobre o vetor de variáveis e $\mathbf{c} = (\mathbf{c}_\mathcal{B}, \mathbf{c}_\mathcal{N})^T$ sobre o vetor de custos da função objetivo. Uma variável x_i é *básica*, ou está na base, se $i \in \mathcal{B}$. Caso contrário, x_i é *não-básica*, ou não está na base.

Definida uma partição básica, o sistema de restrições do problema pode ser escrito da seguinte maneira:

$$\mathbf{B}\mathbf{x}_\mathcal{B} + \mathbf{N}\mathbf{x}_\mathcal{N} = \mathbf{b}.$$

Como $\mathbf{B}$ é não-singular, é possível reescrever essa expressão em função das variáveis não-básicas apenas, obtendo-se a *solução geral* do sistema $\mathbf{A}\mathbf{x} = \mathbf{b}$:

$$\mathbf{x}_\mathcal{B} = \mathbf{B}^{-1}\left(\mathbf{b} - \mathbf{N}\mathbf{x}_\mathcal{N}\right). \tag{2.7}$$

Soluções particulares podem ser obtidas a partir de (2.7) atribuindo-se valores para $\mathbf{x}_\mathcal{N}$, o que resulta na determinação única de $\mathbf{x}_\mathcal{B}$. Uma solução particular de bastante interesse é apresentada na Definição 2.1.

Definição 2.1. (Solução básica primal) *Considere a partição básica* $\mathbf{A} = [\mathbf{B} \mid \mathbf{N}]$ *para o problema (2.1). Seja* $\bar{\mathbf{x}}$ *uma solução particular determinada a partir de (2.7) fixando-se, para cada* $j \in \mathcal{N}$, $\bar{x}_j$ *igual a um de seus limitantes finitos, com* $\bar{x}_j = 0$ *para as variáveis livres. A solução obtida é chamada de* solução básica primal.

Com o intuito de facilitar a exposição, dada uma solução básica primal $\bar{\mathbf{x}}$ o conjunto de índices $\mathcal{N}$ será particionado em três subconjuntos disjuntos $\mathcal{N}^f$, $\mathcal{N}^l$ e $\mathcal{N}^u$ de acordo com os valores atribuídos a $\bar{\mathbf{x}}_\mathcal{N}$:

$$\begin{aligned}
\mathcal{N}^f &= \{j \in \mathcal{N} : \bar{x}_j \text{ é uma variável livre}\},\\
\mathcal{N}^l &= \{j \in \mathcal{N} : \bar{x}_j = l_j > -\infty\},\\
\mathcal{N}^u &= \{j \in \mathcal{N} : \bar{x}_j = u_j < \infty\}.
\end{aligned}$$

Utilizando essa notação, uma solução básica primal é dada por

$$\left\{\begin{array}{lcl}
\bar{\mathbf{x}}_{\mathcal{N}^f} &=& 0,\\
\bar{\mathbf{x}}_{\mathcal{N}^l} &=& \mathbf{l}_{\mathcal{N}^l},\\
\bar{\mathbf{x}}_{\mathcal{N}^u} &=& \mathbf{u}_{\mathcal{N}^u},\\
\bar{\mathbf{x}}_\mathcal{B} &=& \mathbf{B}^{-1}\left(\mathbf{b} - \displaystyle\sum_{j\,\in\,\mathcal{N}^l} l_j\mathbf{a}_j - \sum_{j\,\in\,\mathcal{N}^u} u_j\mathbf{a}_j\right).
\end{array}\right. \tag{2.8}$$

Caso cada componente básica dessa solução satisfaça seu intervalo de factibilidade, tem-se uma *solução básica primal factível*.

Uma solução básica primal é *não-degenerada* se o valor de cada variável básica está no interior de seu intervalo de factibilidade, isto é, $l_i < \bar{x}_i < u_i$ para todo $i \in \mathcal{B}$. Se existir pelo menos uma componente de $\bar{\mathbf{x}}_\mathcal{B}$ igual a um de seus limitantes, então a solução básica primal é *degenerada*. Esta nomenclatura é estendida à base associada.

Considere agora o problema dual dado por (2.3). Substituindo-se a partição básica $\mathbf{A} = [\mathbf{B} \mid \mathbf{N}]$ no sistema de restrições do problema, obtém-se

$$\left[\begin{array}{c}\mathbf{B}^T\mathbf{y}\\ \mathbf{N}^T\mathbf{y}\end{array}\right] + \left[\begin{array}{c}\mathbf{v}_\mathcal{B}\\ \mathbf{v}_\mathcal{N}\end{array}\right] - \left[\begin{array}{c}\mathbf{w}_\mathcal{B}\\ \mathbf{w}_\mathcal{N}\end{array}\right] = \left[\begin{array}{c}\mathbf{c}_\mathcal{B}\\ \mathbf{c}_\mathcal{N}\end{array}\right]. \tag{2.9}$$

Seja $\bar{\mathbf{x}}$ a solução básica primal associada a esta partição, conforme definida em (2.8). Adotando-se a notação $\mathbf{N}_l = \mathbf{A}_{\mathcal{N}^l}$, $\mathbf{N}_u = \mathbf{A}_{\mathcal{N}^u}$ e $\mathbf{N}_f = \mathbf{A}_{\mathcal{N}^f}$, o sistema (2.9) pode ser reescrito como

$$\begin{bmatrix} \mathbf{B}^T\mathbf{y} \\ \mathbf{N}_l^T\mathbf{y} \\ \mathbf{N}_u^T\mathbf{y} \\ \mathbf{N}_f^T\mathbf{y} \end{bmatrix} + \begin{bmatrix} \mathbf{v}_{\mathcal{B}} \\ \mathbf{v}_{\mathcal{N}^l} \\ \mathbf{v}_{\mathcal{N}^u} \\ \mathbf{v}_{\mathcal{N}^f} \end{bmatrix} - \begin{bmatrix} \mathbf{w}_{\mathcal{B}} \\ \mathbf{w}_{\mathcal{N}^l} \\ \mathbf{w}_{\mathcal{N}^u} \\ \mathbf{w}_{\mathcal{N}^f} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_{\mathcal{B}} \\ \mathbf{c}_{\mathcal{N}^l} \\ \mathbf{c}_{\mathcal{N}^u} \\ \mathbf{c}_{\mathcal{N}^f} \end{bmatrix}.$$

Assim, a solução geral deste sistema em função das variáveis $\mathbf{v}_{\mathcal{B}}$, $\mathbf{w}_{\mathcal{B}}$, $\mathbf{w}_{\mathcal{N}^l}$, $\mathbf{v}_{\mathcal{N}^u}$ e $\mathbf{w}_{\mathcal{N}^f}$, é dada por

$$\begin{cases} \mathbf{y}^T &= \mathbf{c}_{\mathcal{B}}^T\mathbf{B}^{-1} - \left(\mathbf{v}_{\mathcal{B}}^T - \mathbf{w}_{\mathcal{B}}^T\right)\mathbf{B}^{-1}, \\ \mathbf{v}_{\mathcal{N}^l}^T &= \left(\mathbf{c}_{\mathcal{N}^l}^T - \mathbf{y}^T\mathbf{N}_l\right) + \mathbf{w}_{\mathcal{N}^l}^T, \\ \mathbf{w}_{\mathcal{N}^u}^T &= -\left(\mathbf{c}_{\mathcal{N}^u}^T - \mathbf{y}^T\mathbf{N}_u\right) + \mathbf{v}_{\mathcal{N}^u}^T, \\ \mathbf{v}_{\mathcal{N}^f}^T &= \left(\mathbf{c}_{\mathcal{N}^f}^T - \mathbf{y}^T\mathbf{N}_f\right) + \mathbf{w}_{\mathcal{N}^f}^T. \end{cases} \tag{2.10}$$

Uma solução particular de bastante interesse é apresentada na Definição 2.2.

Definição 2.2. (Solução básica dual) *Considere a partição básica* $\mathbf{A} = [\mathbf{B} \mid \mathbf{N}]$ *para o problema (2.1). Seja* $\bar{\mathbf{x}}$ *uma solução básica primal conforme definida em (2.8). A solução particular* $(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$ *obtida a partir de (2.10) fixando-se* $(\bar{\mathbf{v}}_{\mathcal{B}}, \bar{\mathbf{w}}_{\mathcal{B}}, \bar{\mathbf{w}}_{\mathcal{N}^l}, \bar{\mathbf{v}}_{\mathcal{N}^u}, \bar{\mathbf{w}}_{\mathcal{N}^f}) = \mathbf{0}$*, é chamada de* solução básica dual.

Para verificar a factibilidade de uma solução básica dual $(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$, deve-se analisar suas componentes não-fixadas. Quando $\bar{\mathbf{v}}_{\mathcal{N}^l} \geq 0$, $\bar{\mathbf{w}}_{\mathcal{N}^u} \geq 0$ e $\bar{\mathbf{v}}_{\mathcal{N}^f} = 0$ tem-se uma solução básica dual *factível*. Se $\bar{\mathbf{v}}_{\mathcal{N}^l} > 0$ e $\bar{\mathbf{w}}_{\mathcal{N}^u} > 0$, então a solução básica dual é *não-degenerada*. Caso contrário, tem-se uma solução básica dual *degenerada*. Novamente, a nomenclatura é estendida à base associada.

Uma base é dita *primal factível* quando a solução básica primal associada é factível. Se a solução básica dual associada for factível, tem-se uma base *dual factível*. Uma base que é tanto primal quanto dual factível é uma *base ótima*, isto é, a solução básica primal associada é uma solução ótima para o problema primal e a solução básica dual associada é uma solução ótima para o problema dual. O Teorema 2.2 formaliza este resultado.

Teorema 2.2. *Dada uma base para problema (2.1), sejam* $\bar{\mathbf{x}}$ *e* $(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$ *a solução básica primal e a solução básica dual associadas a esta base, respectivamente. Se* $\bar{\mathbf{x}}$ *e* $(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$ *são ambas factíveis, então* $\bar{\mathbf{x}}$ *é uma solução ótima do problema (2.1) e* $(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$ *é uma solução ótima do problema (2.3).*

Demonstração: Considerando o Teorema 2.1, resta provar que $f(\bar{\mathbf{x}}) = g(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$. De fato,

$$\begin{aligned} f(\bar{\mathbf{x}}) &= \mathbf{c}_{\mathcal{B}}^T\bar{\mathbf{x}}_{\mathcal{B}} + \mathbf{c}_{\mathcal{N}^f}^T\bar{\mathbf{x}}_{\mathcal{N}^f} + \mathbf{c}_{\mathcal{N}^l}^T\bar{\mathbf{x}}_{\mathcal{N}^l} + \mathbf{c}_{\mathcal{N}^u}^T\bar{\mathbf{x}}_{\mathcal{N}^u} \\ &= \mathbf{c}_{\mathcal{B}}^T\mathbf{B}^{-1}\left(\mathbf{b} - \sum_{j \in \mathcal{N}^l} l_j\mathbf{a}_j - \sum_{j \in \mathcal{N}^u} u_j\mathbf{a}_j\right) + \mathbf{c}_{\mathcal{N}^l}^T\mathbf{l}_{\mathcal{N}^l} + \mathbf{c}_{\mathcal{N}^u}^T\mathbf{u}_{\mathcal{N}^u} \\ &= \mathbf{y}^T\left(\mathbf{b} - \mathbf{N}_l\mathbf{l}_{\mathcal{N}^l} - \mathbf{N}_u\mathbf{u}_{\mathcal{N}^u}\right) + \mathbf{c}_{\mathcal{N}^l}^T\mathbf{l}_{\mathcal{N}^l} + \mathbf{c}_{\mathcal{N}^u}^T\mathbf{u}_{\mathcal{N}^u} \\ &= \mathbf{b}^T\mathbf{y} + \left(\mathbf{c}_{\mathcal{N}^l}^T - \mathbf{y}^T\mathbf{N}_l\right)\mathbf{l}_{\mathcal{N}^l} + \left(\mathbf{c}_{\mathcal{N}^u}^T - \mathbf{y}^T\mathbf{N}_u\right)\mathbf{u}_{\mathcal{N}^u} \\ &= \mathbf{b}^T\mathbf{y} + \mathbf{l}^T\mathbf{v} - \mathbf{u}^T\mathbf{w} \\ &= g(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}}), \end{aligned}$$

com $\mathbf{N}_l = \mathbf{A}_{\mathcal{N}^l}$, $\mathbf{N}_u = \mathbf{A}_{\mathcal{N}^u}$ e $\mathbf{N}_f = \mathbf{A}_{\mathcal{N}^f}$. ■

Observe que em uma solução básica dual $(\bar{\mathbf{y}}, \bar{\mathbf{v}}, \bar{\mathbf{w}})$ tem-se $\bar{v}_j = 0$ ou $\bar{w}_j = 0$, ou ambos, para cada $j \in \mathcal{N}$. Desse modo, pode-se utilizar um único vetor n-dimensional $\bar{\mathbf{s}}$ para a representação dos vetores $\bar{\mathbf{v}}$ e $\bar{\mathbf{w}}$ na solução, definindo-se

$$\begin{bmatrix} \bar{\mathbf{s}}_{\mathcal{B}} \\ \bar{\mathbf{s}}_{\mathcal{N}^l} \\ \bar{\mathbf{s}}_{\mathcal{N}^u} \\ \bar{\mathbf{s}}_{\mathcal{N}^f} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \bar{\mathbf{v}}_{\mathcal{N}^l} \\ -\bar{\mathbf{w}}_{\mathcal{N}^u} \\ \bar{\mathbf{v}}_{\mathcal{N}^f} \end{bmatrix}.$$

Com isso, uma solução básica dual pode ser denotada por $(\bar{\mathbf{y}}, \bar{\mathbf{s}})$, com

$$\begin{cases} \bar{\mathbf{y}}^T &= \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}, \\ \bar{\mathbf{s}}_{\mathcal{N}}^T &= \mathbf{c}_{\mathcal{N}}^T - \bar{\mathbf{y}}^T \mathbf{N}. \end{cases} \tag{2.11}$$

Essa solução dual é factível se satisfaz $\bar{\mathbf{s}}_{\mathcal{N}^l} \geq \mathbf{0}$, $\bar{\mathbf{s}}_{\mathcal{N}^u} \leq \mathbf{0}$ e $\bar{\mathbf{s}}_{\mathcal{N}^f} = \mathbf{0}$.

A teoria relacionando os problemas primal e dual é bastante extensa e sua exposição foge do escopo deste trabalho. Os demais resultados podem ser encontrados em livros-texto de otimização linear como Arenales et al. (2007) e Bazaraa et al. (1990).

O método primal simplex explora o resultado apresentado no Teorema 2.2 para obter uma solução ótima de um PL. De fato, percorre-se bases primal factíveis até obter uma que seja também dual factível. Já o método dual simplex, percorre bases dual factíveis até uma obter uma que seja também primal factível.

Variantes destes métodos recebem o nome de métodos tipo simplex por também se basearem em soluções básicas. Conforme apresentado na literatura, estas variantes podem ser mais eficientes que os métodos convencionais na resolução de certas classes de problemas. Para o leitor interessado, são sugeridos os trabalhos de Paparrizos et al. (2003), Sousa et al. (2005) e Pan (2008).

Na seção a seguir, o método primal simplex é descrito resumidamente, com o intuito de definir a notação para os próximos capítulos. Assume-se que o leitor seja familiarizado com o método.

2.4 Método primal simplex

Considere um PL na forma padrão (2.1). Suponha que o problema seja factível e que uma base primal factível conhecida esteja descrita pelo conjunto de índices $\mathcal{B}$, $\mathcal{N}^f$, $\mathcal{N}^l$ e $\mathcal{N}^u$. Assim, o método primal simplex pode ser iniciado e o primeiro passo é calcular a componente $\mathbf{x}_{\mathcal{B}}$ da solução básica primal factível, conforme descrito em (2.8). Em seguida, o *teste de otimalidade* consiste em verificar se esta solução é ótima, analisando-se a solução básica dual correspondente. Para isto, deve-se calcular a componente $\mathbf{y}$ da solução básica dual, dada por:

$$\mathbf{y}^T = \mathbf{c}_{\mathcal{B}}^T \mathbf{B}^{-1}, \tag{2.12}$$

também chamada de *vetor multiplicador simplex* no contexto do método primal simplex, e o vetor de *custos reduzidos*:

$$\mathbf{s}^T = \begin{bmatrix} \mathbf{s}_{\mathcal{B}}^T \\ \mathbf{s}_{\mathcal{N}}^T \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{c}_{\mathcal{N}}^T - \mathbf{y}^T \mathbf{N} \end{bmatrix}, \tag{2.13}$$

corresponde à componente $\mathbf{s}$ da solução básica dual, conforme apresentado em (2.11).

Para verificar a factibilidade da solução dual, basta que sejam analisadas as componentes não-básicas $\mathbf{s}_{\mathcal{N}}$ dos custos reduzidos. A Proposição 2.1 determina a otimalidade de uma solução básica primal factível de acordo com estes valores.

Proposição 2.1. (Condições de otimalidade) *Seja* $\mathbf{A} = [\mathbf{B} \mid \mathbf{N}]$ *uma partição básica para o PL (2.1), à qual estão associadas a solução básica primal factível* $\mathbf{x}$*, conforme definida em (2.8), e o vetor de custos reduzidos* $\mathbf{s}$*, dado por (2.13). Tem-se que* $\mathbf{x}$ *é uma solução ótima deste problema se as seguintes condições são satisfeitas:*

(i) $s_j \geq 0$ *para todo* $j \in \mathcal{N}^l$*;*

(ii) $s_j \leq 0$ *para todo* $j \in \mathcal{N}^u$*;*

(iii) $s_j = 0$ *para todo* $j \in \mathcal{N}^f$*.*

Demonstração: Por hipótese, $\mathbf{x}$ é uma solução básica primal factível. Para y definido como em (2.12), tem-se que $(\mathbf{y}, \mathbf{s})$ é uma solução básica para o problema dual associado, de acordo com (2.11). Se as condições (i), (ii) e (iii) são satisfeitas, então $(\mathbf{y}, \mathbf{s})$ é uma solução básica dual factível. Logo, pelo Teorema 2.2, $\mathbf{x}$ é uma solução ótima do problema (2.1). ■

Suponha que a solução básica primal factível possua algum índice não-básico $k \in \mathcal{N}^f \cup \mathcal{N}^l \cup \mathcal{N}^u$ que não satisfaça alguma das condições da Proposição 2.1. Neste caso, diz-se que s_k viola as condições de otimalidade. Assim, esta solução não é ótima e uma nova solução capaz de melhorar o valor da função objetivo pode ser determinada, a menos de degeneração primal. A obtenção de uma nova solução básica primal factível $\tilde{\mathbf{x}}$ envolve a determinação de uma direção de busca $\mathbf{d}$ e de um tamanho do passo ε a ser dado nesta direção.

No método primal simplex, a direção de busca recebe o nome de *direção primal simplex*, definida como

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_{\mathcal{B}} \\ \mathbf{d}_{\mathcal{N}} \end{bmatrix} = \begin{bmatrix} -\mathbf{B}^{-1}\mathbf{a}_k \\ \mathbf{e}_r \end{bmatrix} \tag{2.14}$$

sendo $k \in \mathcal{N}$ escolhido de modo que o custo relativo s_k viole as condições de otimalidade. O índice r corresponde à posição de k no conjunto $\mathcal{N}$ e $\mathbf{e}_r$ é a r-ésima coluna da matriz identidade de ordem $n - m$.

Caso exista mais de um índice não-básico que viole as condições de otimalidade, algum critério de seleção deve ser empregado para se determinar k. Essa operação recebe o nome de *pricing*. A estratégia de *pricing* inicialmente proposta para o método primal simplex é chamada de *regra de Dantzig* e consiste em selecionar k de modo que

$$|s_k| = \max\{|s_j| : s_j \text{ viola as condições de otimalidade}, j \in \mathcal{N}^f \cup \mathcal{N}^l \cup \mathcal{N}^u\}, \tag{2.15}$$

isto é, escolhe-se o índice correspondente ao custo reduzido de maior violação das condições de otimalidade. Outras abordagens estão propostas na literatura para a realização dessa escolha (veja a seção 3.4). A utilização de um critério eficiente é bastante importante, já que o número de iterações necessárias para a convergência do método é extremamente dependente do critério utilizado. O cálculo do vetor de custos reduzidos pode ser realizado enquanto se busca pelo índice k, dentro da operação de *pricing*. Além disso, pode-se calcular apenas uma parte do vetor, ao invés de todas as suas componentes.

Determinada a direção primal simplex, é preciso definir o tamanho de passo $\varepsilon > 0$ a ser dado nesta direção. A nova solução básica primal factível $\tilde{\mathbf{x}}$ é definida a partir da solução $\bar{\mathbf{x}}$ por

$$\tilde{\mathbf{x}} = \begin{cases} \bar{\mathbf{x}} + \varepsilon \mathbf{d}, & \text{se } s_k < 0, \\ \bar{\mathbf{x}} - \varepsilon \mathbf{d}, & \text{se } s_k > 0. \end{cases} \tag{2.16}$$

Definindo-se $\varepsilon_q = \min_i\{\varepsilon_i\}$ com

$$\varepsilon_i = \begin{cases} (\bar{x}_{\mathcal{B}_i} - l_{\mathcal{B}_i})/d_{\mathcal{B}_i}, & \text{se } d_{\mathcal{B}_i} > 0 \text{ e } l_{\mathcal{B}_i} > -\infty, \\ (\bar{x}_{\mathcal{B}_i} - u_{\mathcal{B}_i})/d_{\mathcal{B}_i}, & \text{se } d_{\mathcal{B}_i} < 0 \text{ e } u_{\mathcal{B}_i} < \infty, \\ +\infty, & \text{caso contrário.} \end{cases} \tag{2.17}$$

é possível obter a partir da expressão (2.16), a seguinte regra para a determinação do tamanho de passo, conhecida como *teste da razão*:

$$\varepsilon = \min\{u_k - l_k, \varepsilon_q\}. \tag{2.18}$$

Caso não seja possível determinar um valor finito para ε, tem-se que o problema é ilimitado. Quando $\varepsilon_q < +\infty$, o elemento $d_{\mathcal{B}_q}$ é chamado de *pivô*. A implementação computacional do teste da razão deve evitar a seleção de pivôs com valor absoluto muito pequeno. Um pivô próximo de zero pode causar instabilidade numérica e prejudicar a resolução do problema, como discutido no Capítulo 5.

Quando o tamanho de passo obtido é igual à diferença $(u_k - l_k)$ realiza-se uma *troca de limitante* da solução x_k, isto é, a variável passa de um extremo de seu intervalo de factibilidade para o outro. Neste caso, a partição básica não é modificada e apenas os conjuntos $\mathcal{N}^l$ e $\mathcal{N}^u$ são atualizados.

Se $\varepsilon = \varepsilon_q$ tem-se que a variável básica $x_{\mathcal{B}_q}$ se torna igual a um de seus limitantes finitos e a variável não-básica x_k deixa de ser igual a um de seus limitantes (exceto quando $\varepsilon_q = 0$). Esse resultado sugere uma *troca de base*, isto é, $x_{\mathcal{B}_q}$ se torna não-básica e x_k se torna a q-ésima variável básica. Diz-se que $x_{\mathcal{B}_q}$ *sai da base* e x_k *entra na base*. Assim, os conjuntos $\mathcal{B}$ e $\mathcal{N}$ são atualizados, resultando em uma nova base.

Com isso, finaliza-se uma iteração do método primal simplex. O processo iterativo é concluído quando uma solução básica primal factível satisfaz as condições de otimalidade ou quando se detecta que o problema é ilimitado pelo teste da razão. O Algoritmo 1 descreve o método para problemas na forma padrão. O parâmetro de entrada IT_MAX determina o número máximo de iterações que podem ser realizadas. O algoritmo é descrito em alto nível com o intuito de apresentar os passos do método de forma conceitual, sem se prender à implementação computacional. Os detalhes de uma implementação eficiente e estável estão descritos nos próximos capítulos deste trabalho.

No Algoritmo 1 e nos demais algoritmos apresentados nos próximos capítulos desta dissertação, os sinais /* ... */ correspondem aos delimitadores de comentários. A demarcação de início e fim de blocos é feita utilizando-se chaves. A atribuição de valores e a comparação de igualdade são ambas representadas pelo símbolo =, pois ficam bem determinadas pelo contexto.

Algoritmo 1: Método primal simplex para problemas na forma padrão com variáveis canalizadas.

Entrada: matriz $\mathbf{A}$; vetores $\mathbf{b}$, $\mathbf{l}$ e $\mathbf{u}$; Base primal factível dada por $\mathcal{B}$, $\mathcal{N}^f$, $\mathcal{N}^l$ e $\mathcal{N}^u$; IT_MAX.
Saída: Solução ótima $\mathbf{x}$; ou detecta problema ilimitado; ou atinge o máximo de iterações.

```
1  /* Calcular as componentes básicas da solução básica primal factível */
2  x_B = B^{-1} ( b − Σ_{j ∈ N^l} l_j a_j − Σ_{j ∈ N^u} u_j a_j ) ;
3
4  /* Inicia o processo iterativo */
5  Enquanto (IT < IT_MAX) faça
6  {
7     /* Calcular a solução básica dual */
8     y^T = c_B^T B^{-1};
9     s_N^T = c_N^T − y^T N;
10
11    /* Pricing */
12    Obtenha k tal que s_k viole as condições de otimalidade;
13    Se (não foi possível obter k) então PARE; /* Solução ótima encontrada! */
14
15    /* Calcular as componentes básicas da direção primal simplex */
16    d_B = −B^{-1} a_k;
17
18    /* Teste da razão */
19    ε = min{u_k − l_k, ε_q}.
20    Se (ε = +∞) então PARE; /* O problema é ilimitado! */
21
22    /* Atualização da solução básica primal */
23    Se (s_k < 0) então x_B = x_B + ε d_B;
24    Senão x_B = x_B − ε d_B;
25
26    /* Atualização da base */
27    Se (ε = u_k − l_k) então
28    {
29       /* Troca de limitante */
30       Se (k ∈ N^l) então { N^l = N^l \ {k}; N^u = N^u ∪ {k}; }
31       Senão { N^l = N^l ∪ {k}; N^u = N^u \ {k};}
32    }
33    Senão
34    {
35       /* Troca de base */
36       N_r = B_q;
37       B_q = k;
38    }
39    /* Incrementa o contador de iterações */
40    IT = IT + 1;
41 }
```

Capítulo 3

Eficiência e estabilidade em métodos tipo simplex

O desenvolvimento de *softwares* de otimização que sejam capazes de resolver problemas reais de grande porte é bastante trabalhoso. De acordo com Maros e Khaliq (2002), é essencial o conhecimento da teoria de otimização, de resultados relevantes em computação científica, de princípios de engenharia de *software* e do estado-da-arte em tecnologia de *hardware* computacional. A implementação computacional direta de algoritmos sem considerar os resultados relevantes de cada uma dessas áreas leva, inevitavelmente, a resultados indesejáveis, entre eles:

- tempo de processamento elevado para a obtenção da solução procurada;
- uso excessivo de memória para a execução do *software*;
- soluções incorretas ou não-convergência do método devido a erros numéricos;
- resolução apenas de problemas de pequeno porte;

Surge, então, a necessidade de uma abordagem adequada na implementação dos algoritmos teóricos, bem como sua integração em um *software* que seja robusto, eficiente e flexível (Maros e Khaliq, 2002).

Considerando esta discussão, diversas técnicas estão propostas para a implementação de *softwares* de otimização baseados em métodos tipo simplex. A implementação eficiente e estável destes métodos envolve *aspectos conceituais*, entre eles inicialização avançada, pré-processamento do problema, estratégias de *pricing*, tratamento da degeneração; *estruturas de dados* adequadas, para o armazenamento de vetores e matrizes esparsas e de dados internos utilizados pelo método; e *aspectos numéricos*, buscando-se evitar que erros de arredondamento e o mal-condicionamento do problema resultem na instabilidade numérica do método.

Um *software* baseado na implementação de métodos tipo simplex pode ser dividido em quatro núcleos principais, conforme ilustrado na Figura 3.1. As setas são utilizadas para indicar a possibilidade de comunicação entre os núcleos. Para cada núcleo, tem-se a indicação de seus principais componentes.

O núcleo *conceitual* é bastante próximo à teoria, envolvendo a codificação das mesmas operações realizadas na resolução algébrica do problema. Por exemplo, uma estratégia

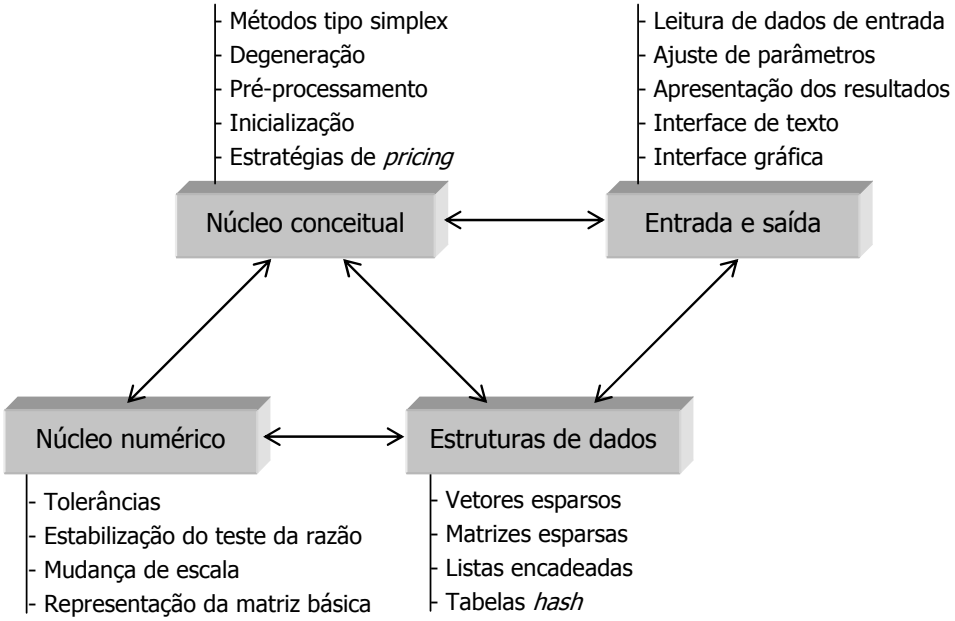

Figura 3.1: Componentes para uma implementação eficiente e estável.

de *pricing* é uma técnica conceitual. As vantagens proporcionadas pela utilização desta técnica não dependem do problema ser resolvido por um computador, ou utilizando-se lápis e papel. O mesmo ocorre para as técnicas de inicialização, tratamento de degeneração e pré-processamento.

Já os outros núcleos, têm uma ligação estreita com a utilização de um computador. O núcleo de *estrutura de dados* deve tratar do armazenamento de diversos tipos de informações, da forma mais eficiente possível. Os dados de entrada, os dados internos utilizados para a execução do método e os dados de saída devem ser representados evitando-se o desperdício de memória e garantindo-se o acesso rápido a estes dados.

O núcleo *numérico* é responsável pelos cálculos envolvendo matrizes e vetores e pelas operações que buscam manter a estabilidade numérica do método implementado. As técnicas utilizadas lidam com a prevenção ou correção dos erros de arredondamento causados pela utilização de uma aritmética de precisão finita. Este núcleo é fundamental para o sucesso do *software*. Se ele falhar, o *software* também falha. Juntamente com as estruturas de dados, os dois núcleos formam a base do *software*.

Para a utilização do *software*, são necessárias as operações do núcleo de *entrada e saída*. A leitura automatizada dos dados de entrada por meio de arquivos é indispensável na resolução de problemas de grande porte. Deve ser permitido ao usuário, o ajuste de parâmetros utilizados na resolução de um determinado problema. Uma interface de texto precisa estar sempre disponível para a utilização manual e automatizada do *software* e uma interface gráfica amigável pode facilitar sua utilização.

Para ilustrar a importância de uma implementação eficiente, Bixby (2002) relata alguns testes envolvendo diferentes versões do *software* comercial CPLEX (ILOG Inc., 2008). A implementação do método dual simplex na versão 1.0 do *software* resolve um problema com 16223 restrições e 28568 variáveis em 1217,4 segundos. Já a versão

7.1, que conta com uma implementação eficiente do mesmo método, resolve o mesmo problema em 22,6 segundos, executando-se no mesmo computador.

Nas seções a seguir, são brevemente discutidas as principais técnicas que devem compor o núcleo conceitual de uma implementação eficiente e estável. Para o leitor interessado em se aprofundar em algum tópico, são sugeridas as referências bibliográficas mais relevantes.

3.1 Tratamento de degeneração

Em um método tipo simplex, uma base degenerada pode ser identificada, dependendo dos valores obtidos para suas variáveis básicas. A degeneração é uma característica do problema e pode levar a certas dificuldades como ciclagem e estagnação.

Na ausência de degeneração, os métodos tipo simplex convergem em um número finito de iterações, obtendo uma solução básica factível ótima ou verificando que tal solução não existe. Entretanto, quando uma solução básica degenerada é encontrada, é possível que o método faça uma troca de base que não altere o valor da função objetivo. Se este processo se estende para as próximas iterações, tem-se o fenômeno conhecido como *estagnação* (na literatura inglesa, *stalling*).

Se após algumas iterações trocando bases degeneradas, uma base já visitada for reconsiderada, forma-se um ciclo de bases, caracterizando o fenômeno de *ciclagem*. Neste caso, se nenhuma regra anti-ciclagem for empregada, o método não converge. Algumas regras anti-ciclagem estão propostas na literatura, como a regra de Bland e a regra lexicográfica (Dantzig e Thapa, 1997), que garantem a convergência teórica do método. Entretanto, estas regras têm importância apenas teórica, pois são computacionalmente ineficientes. Na prática, são utilizados métodos heurísticos, como a perturbação do vetor independente e a expansão de limitantes (Gill et al., 1989; Bixby, 2002), que também ajudam a evitar a estagnação.

3.2 Pré-processamento

Problemas de otimização linear de grande porte são geralmente modelados utilizando-se alguma interface gráfica de modelagem, que permita transformar o modelo gerado em um arquivo contendo os dados do problema. Na maioria das vezes, estes modelos possuem certas redundâncias ou características que podem acarretar comportamentos indesejados durante a resolução, como a instabilidade numérica, e também podem fazer com que o tempo computacional para a obtenção de uma solução seja alto.

Por outro lado, os dados de entrada podem corresponder a um sub-problema definido durante a resolução de um problema mais complexo. Por exemplo, quando um método de enumeração implícita é utilizado na resolução de um problema de otimização inteira (Wolsey, 1998). Em particular, estes métodos têm como característica a resolução de vários problemas de otimização linear que se diferenciam pela introdução de restrições. Estas restrições adicionais podem levar à infactibilidade do problema, à fixação de certas variáveis ou ainda à redefinição de limitantes.

Sendo assim, antes de resolver um problema de otimização linear, deve-se tentar efetuar certas operações sobre o modelo, de modo a obter uma formulação equivalente

para o problema, que permita uma resolução mais eficiente. Esta atividade recebe o nome de *pré-processamento*. Os principais objetivos são reduzir o tamanho do problema, melhorar as características numéricas e detectar se este é ilimitado ou infactível, antes de iniciar a resolução por um método tipo simplex. Após a resolução do problema, as alterações devem ser descartadas para que o resultado seja apresentado de acordo com os dados de entrada.

De acordo com Maros (2003a), a realização de pré-processamento é muito importante em *softwares* que implementam métodos tipo simplex. As técnicas aplicadas consistem de uma série de testes e operações simples envolvendo a análise e o ajuste de limitantes, a remoção de restrições formadas por uma única variável, a substituição de variáveis fixas, entre outros. Outras técnicas estão propostas na literatura (Fourer e Gay, 1993; Mészáros e Suhl, 2003) e podem ser combinadas para a construção de um algoritmo de pré-processamento. Quanto mais técnicas são empregadas, maior o tempo computacional demandado e, assim, deve-se buscar um equilíbrio entre o tempo para a realização do pré-processamento e os benefícios proporcionados.

3.3 Inicialização

Para a resolução de um PL utilizando-se um método tipo simplex, uma base inicial factível deve ser conhecida a priore. A obtenção de uma base factível é chamada de *Inicialização*. O primeiro passo consiste em definir uma *base inicial*, factível ou não, utilizando-se um procedimento para a seleção de colunas que irão compor a matriz básica inicial.

O desempenho de métodos tipo simplex é bastante dependente da base inicial escolhida. Apesar disso, poucos trabalhos na literatura tratam da construção de bases iniciais de alta qualidade. Este fato foi observado por Gould e Reid (1989) e continua sendo realidade. Além disso, os *softwares* de otimização linear mais utilizados possuem procedimentos próprios para a construção de uma base inicial, que utilizam abordagens distintas.

Uma base inicial deve ser formada buscando-se certas características como a proximidade a uma base ótima, não-degeneração, baixo custo computacional para construção, matriz básica esparsa e bem-condicionada. Uma base inicial que satisfaça a todas essas características é de *alta qualidade*. Entretanto, na maioria das vezes, essas características são conflitantes, tornando difícil a construção de uma base de alta qualidade. Os procedimentos propostos na literatura se diferenciam de acordo com qual característica é priorizada.

A maneira mais simples para a construção de uma base inicial é dada pelas bases unitárias. A construção destas bases corresponde à seleção de todas as variáveis lógicas do problema, recorrendo-se a variáveis artificiais quando necessário. Computacionalmente, esta abordagem faz com que as primeiras iterações sejam rápidas e numericamente precisas e o esforço computacional necessário para a construí-las é irrelevante.

Quando uma base unitária é composta apenas por variáveis artificiais, ela recebe o nome de base *artificial*. Além de aumentar o número de variáveis do problema, essa base resulta em um maior número de iterações de um método tipo simplex, por não possuir relação com uma base ótima do problema. Considerando que a matriz de coeficientes do problema tenha posto completo, nenhuma variável artificial permanecerá na base ótima

e, portanto, todas deverão ser removidas da base, sendo necessária uma iteração para cada remoção.

Uma maneira mais inteligente de se construir uma base unitária é aproveitando-se as colunas lógicas da matriz de coeficientes. A base *lógica* possui as mesmas vantagens da base artificial porém amenizando as desvantagens envolvendo o aumento da dimensão do problema e a relação com uma base ótima. Se o problema não possui variáveis lógicas suficientes para se completar uma base lógica, então variáveis artificiais devem ser adicionadas, dando origem a uma base *lógica-mista*.

A inicialização de métodos tipo simplex utilizando bases unitárias era uma escolha padrão para as primeiras implementações destes métodos. Entretanto, a experiência com a resolução de problemas de grande porte mostrou que a utilização de bases avançadas resulta em um melhor desempenho, conforme descrito por Maros e Mitra (1996). Bases *avançadas* são bases iniciais contendo variáveis estruturais do problema. Em geral, tais bases têm mais chance de estar próxima de uma base ótima do que uma base unitária.

As bases iniciais utilizadas pelos *softwares* LPAKO (Lim e Park, 2002) e MOPS (Suhl, 1994) são bases avançadas triangulares que priorizam variáveis lógicas e utilizam variáveis estruturais para completar a base construída. Se diferenciam pela ordem em que as colunas estruturais são escolhidas para compor a matriz básica. Bixby (1992) apresenta uma abordagem semelhante para a construção de uma base avançada, denominada base CPLEX, utilizada como base inicial padrão na versão 2.0 do *software*.

Bases *crash* são bases iniciais avançadas cujo procedimento de construção é baseado na técnica *crash*, apresentada por Carstens (1968). A matriz básica associada é triangular e, assim, não ocorre preenchimento (*fill-in*) na representação de sua inversa, como observado por Maros (2003a). Os *softwares* CPLEX (ILOG) e FortMP (OptiRisk) são exemplos de implementações eficientes de métodos tipo simplex que utilizam bases *crash*, conforme descrito por ILOG Inc. (2008) e Maros e Mitra (1996), respectivamente.

Gould e Reid (1989) propõem um procedimento para a construção de uma base avançada cuja matriz é triangular em blocos, denominado de algoritmo *tearing*. O primeiro passo deste algoritmo é a reordenação das colunas da matriz de coeficientes, com o intuito de obter uma matriz triangular inferior em blocos. Em seguida, os blocos definidos são utilizados para decompor o problema em subproblemas densos que são então resolvidos a fim de se identificar uma base inicial de alta qualidade.

Vale ressaltar que os procedimentos propostos na literatura para a construção de bases avançadas correspondem a procedimentos heurísticos, e não garantem que a base construída seja de alta qualidade. Entretanto, os resultados obtidos com a utilização de bases avançadas são superiores na resolução de problemas de grande porte em geral. Toda implementação eficiente de métodos tipo simplex deve contar com estes procedimentos para a obtenção de uma base inicial.

O segundo passo na Inicialização de um método tipo simplex é verificar a factibilidade da base construída. Caso esta seja factível, então a resolução do problema pode ser iniciada. Caso contrário, dois caminhos são possíveis:

- utiliza-se uma técnica de *duas fases*, caracterizada pela existência de uma fase inicial bem determinada, denominada Fase-I. Nesta fase, um problema auxiliar é resolvido e uma solução básica factível para o problema original é obtida. Em uma segunda etapa, denominada Fase-II, o método simplex é iniciado com a base factível obtida na Fase-I;

- utiliza-se uma técnica de *fase única*, caracterizada pela adição de termos à função objetivo do problema original, que agem como penalizadores de soluções infactíveis. Com isso, a resolução do problema modificado é iniciada sem que se conheça uma base factível. A factibilidade da solução é obtida enquanto se busca, ao mesmo tempo, uma solução ótima do problema.

Em certas técnicas de duas fases, o problema auxiliar definido corresponde a um PL com uma solução básica factível conhecida e, assim, um método tipo simplex pode ser utilizado na Fase-I. Na prática, a utilização de problemas auxiliares cuja função objetivo é linear por partes resultam em abordagens mais eficientes. Neste contexto, Orchard-Hays (1968) apresenta um método para a Fase-I que se baseia na estratégia de métodos tipo simplex e busca a redução da infactibilidade a cada iteração. No método proposto, o autor redefine a escolha de variáveis que devem entrar e sair da base, considerando a linearidade por partes da função objetivo. Maros (1986) propõe uma melhoria para este método que consiste na análise de pontos de não-diferenciação da função objetivo. O progresso realizado em uma única iteração do método proposto pode corresponder ao progresso de várias iterações do método de Orchard-Hays (1968).

Uma técnica de fase única bastante conhecida é chamada de M-grande, caracterizada pela penalização de variáveis artificiais utilizando um escalar M, suficientemente grande para garantir a factibilidade do problema (Bazaraa et al., 1990). Conforme retratado na literatura, a escolha de um valor numérico para M é bastante crítica para a eficiência desta técnica, podendo resultar em um número elevado de iterações, na instabilidade numérica e até mesmo na obtenção de uma solução que seja infactível para o problema original. Buscando evitar estas características, Munari e Arenales (2008) propõem o método M-grande implícito, no qual não se atribui um valor numérico para M. Para isso, M é considerado implicitamente nos cálculos, como sendo um parâmetro suficientemente grande. Com esta modificação, os autores mostram que é possível evitar as desvantagens envolvendo o método M-grande.

3.4 Estratégias de *pricing*

A operação de *pricing* é um assunto em constante pesquisa na área de otimização linear. Conforme apresentado, esta operação consiste em, a cada iteração, selecionar um índice $j \in \mathcal{N}$ tal que s_j viole as condições de otimalidade. Em geral, existe uma grande liberdade na escolha deste índice, resultando em diferentes estratégias de *pricing*. Conceitualmente, a melhor estratégia é dada por aquela que proporcione o menor tempo computacional possível para a resolução do problema. Entretanto, obter essa informação é inviável e as estratégias utilizadas são técnicas heurísticas que buscam a redução do tempo computacional.

Um *pricing* é dito normalizado quando os custos reduzidos calculados são normalizados para evitar que a escala da matriz de coeficientes influencie na comparação dos termos. Nesta categoria, as estratégias mais conhecidas são a DEVEX, proposta por Harris (1973), e a *Steepest Edge*, proposta por Forrest e Goldfarb (1992). Estas estratégias são fundamentais para se garantir a eficiência de métodos tipo simplex, de acordo com Maros (2003a). Bixby (2002) ressalta que a estratégia de Forrest e Goldfarb (1992) é responsável pelo sucesso das implementações do método dual simplex. Essa afirmação é comprovada nos resultados apresentados por Koberstein (2008).

Quando todos os índices não-básicos são considerados para se verificar a violação das condições de otimalidade, o *pricing* é chamado de completo. Se apenas um subconjunto de $\mathcal{N}$ é considerado, tem-se um *pricing* parcial. O *pricing* seccional é um tipo de *pricing* parcial que explora a estrutura do problema. Muitos problemas possuem uma estrutura que é caracterizada pela presença de seções ou *clusters* de variáveis. Em casos assim, a estratégia de *pricing* pode operar de maneira mais eficiente escolhendo candidatos de diferentes seções. Estas técnicas são descritas de forma detalhada por Maros (2003b). Neste mesmo trabalho, o autor propõe uma estratégia geral de *pricing*, que busca escolher a melhor técnica a ser aplicada durante a resolução do problema.

Capítulo 4

Estruturas de dados

Uma questão fundamental na implementação de um método numérico é a maneira como são representados e manipulados os dados de entrada e aqueles gerados durante a resolução do problema. As estruturas de dados precisam ser cuidadosamente projetadas considerando-se certas características dos dados, os tipos de operações realizadas e a forma de acesso a estes dados. O objetivo deste capítulo é apresentar as principais estruturas de dados utilizadas na implementação eficiente de métodos tipo simplex.

Uma característica importante em um PL real de grande porte é a *esparsidade*. Em geral, os dados do problema são descritos por matrizes e vetores esparsos, isto é, apenas uma pequena porcentagem de seus elementos são não-nulos. Na prática, conforme citado por Maros (2003a), existem em torno de 5 a 10 elementos não-nulos, em média, em cada coluna da matriz de coeficientes, independentemente do tamanho do problema. Apenas esses elementos precisam ser armazenados, fazendo com que a esparsidade seja vista como a principal característica em problemas de grande porte, podendo viabilizar a representação e a resolução computacional desta classe de problemas. As estruturas de dados para a representação de vetores e matrizes esparsos de um PL são descritas nas Seções 4.1 e 4.2.

Além da representação de matrizes e vetores, estruturas de dados eficientes também são necessárias para a representação de dados internos utilizados durante a resolução do problema. Estes dados são responsáveis pelas mais variadas informações como os índices de variáveis básicas, o tipo de cada variável do problema, o número de elementos não-nulos em uma dada linha da matriz básica, entre outros.

Como conseqüência, diversas estruturas de dados precisam ser utilizadas, entre elas listas encadeadas simples, listas duplamente encadeadas, pilhas e *heaps*. A descrição dessas estruturas é extensa e pode ser encontrada em livros de estruturas de dados, como Aho et al. (1983). O emprego de cada estrutura depende das características da informação a ser representada e como essa informação é produzida, utilizada e descartada. Por exemplo, os procedimentos para a obtenção de uma representação eficiente para a matriz básica necessitam que contadores de elementos não-nulos em cada coluna e cada linha da matriz sejam mantidos. Em implementações eficientes, são utilizadas listas duplamente encadeadas para a representação desta informação. Maros (2003a) apresenta este tipo de estrutura de forma específica para o contexto descrito.

4.1 Vetores esparsos

Seja $\boldsymbol{\alpha}$ um vetor n-dimensional esparso. Computacionalmente, é possível representá-lo de maneira explícita por meio de um arranjo unidimensional de n posições, em que cada elemento α_i é armazenado na i-iésima posição deste arranjo. Esta forma de representação é chamada de *densa* pois armazena cada elemento do vetor, nulo ou não, deixando de explorar a esparsidade de $\boldsymbol{\alpha}$. A vantagem é que um elemento arbitrário pode ser acessado de forma direta no arranjo, facilitando a utilização e a alteração dos valores do arranjo.

Considere o vetor esparso $\bar{\boldsymbol{\alpha}}$ com 10 elementos, sendo $\bar{\alpha}_2 = -3$, $\bar{\alpha}_7 = 1.1$ e $\bar{\alpha}_8 = 6$ os únicos elementos não-nulos. Na Figura 4.1, tem-se a representação densa deste vetor.

	1	2	3	4	5	6	7	8	9	10
valor:	0	-3	0	0	0	0	1.1	6	0	0

Figura 4.1: Representação densa do vetor $\bar{\boldsymbol{\alpha}}$.

Além do desperdício de memória devido ao armazenamento de elementos nulos, a representação densa faz com que os cálculos com o arranjo envolvam operações aritméticas desnecessárias, ocasionadas pelos elementos nulos. A representação *indexada* busca evitar estas operações, utilizando um arranjo adicional para o armazenamento dos índices de elementos não-nulos. Neste caso, o número de elementos não-nulos nnz precisa ser armazenado, para indicar o número de posições válidas neste segundo arranjo.

O vetor $\bar{\boldsymbol{\alpha}}$ é dado na Figura 4.2 de acordo com a representação indexada. Ao se realizar um cálculo, basta acessar o arranjo índice e obter as posições de elementos não-nulos do arranjo valor. Observe que foi mantido um espaço livre adicional, denotado em cinza na figura, para armazenar o índice de novos elementos não-nulos que possam vir a ser criados. Caso não exista exista esta possibilidade, o espaço adicional pode ser descartado.

	1	2	3							
índice:	2	7	8							
	1	2	3	4	5	6	7	8	9	10
valor:	0	-3	0	0	0	0	1.1	6	0	0

Figura 4.2: Representação indexada do vetor $\bar{\boldsymbol{\alpha}}$.

Uma outra maneira de representar um vetor esparso, chamada de representação *compacta*, consiste em armazenar apenas os elementos não-nulos em um arranjo. Como na representação indexada, um segundo arranjo deve ser usado para o armazenamento dos índices de cada um destes elementos e o número de elementos não-nulos nnz deve ser armazenado. Na Figura 4.3, é mostrada a representação compacta do vetor $\bar{\boldsymbol{\alpha}}$. Note a presença de espaços adicionais livres em ambos os arranjos, denotados em cinza, utilizados para o armazenamento de novos elementos não-nulos que possam vir a ser criados durante os cálculos.

A representação compacta evita o desperdício de memória, porém os cálculos envolvendo dois ou mais vetores nesta representação se tornam mais complexos. Em casos

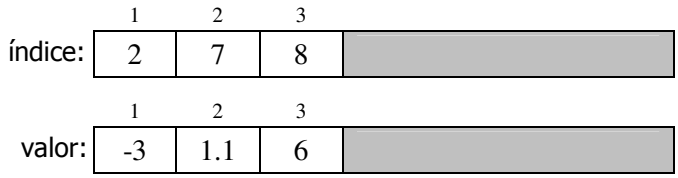

Figura 4.3: Representação compacta do vetor $\bar{\boldsymbol{\alpha}}$.

assim, pode-se *expandir* um dos arranjos, isto é, copiar seus elementos para um arranjo com capacidade para armazená-los juntamente com os elementos nulos. Esta operação é equivalente a converter a representação compacta para a indexada. Os cálculos são realizados considerando a representação indexada e, ao final, é necessário *compactar* o arranjo criado, ou seja, converter a representação indexada para a compacta novamente. Apesar de parecer computacionalmente cara, essa abordagem é bastante eficiente na prática.

4.2 Matrizes esparsas

A representação de matrizes esparsas é semelhante à representação de vetores. Cada coluna da matriz corresponde a um vetor esparso e, assim, pode ser armazenada utilizando uma das representações da seção anterior. Em problemas de grande porte, representar as colunas de forma densa ou indexada é inviável, pois os elementos não-nulos também são armazenados no arranjo. Assim, adota-se a representação compacta das colunas, resultando na *representação compacta por colunas* da matriz.

Seja $\mathbf{A}$ uma matriz esparsa com m linhas, n colunas e nnz elementos reais não-nulos. A representação compacta por colunas desta matriz utiliza quatro arranjos, conforme ilustrado na Figura 4.4. O arranjo valor armazena de forma contígua os elementos não-nulos das colunas $\mathbf{a}_j$, $j = 1, \ldots, n$. Para uma dada coluna $\mathbf{a}_j$, i_{jt} corresponde ao índice da linha de seu t-ésimo elemento armazenado. Para cada elemento no arranjo valor, tem-se o índice de sua linha na mesma posição do arranjo ind_lin. Os dois arranjos devem ter capacidade para, pelo menos, nnz elementos. A posição inicial de uma dada coluna $\mathbf{a}_j$ nestes arranjos é armazenada na j-ésima posição do arranjo ind_col. Nesta mesma posição, porém no arranjo nc, tem-se o número de elementos não-nulos da coluna. Ambos devem ter n posições.

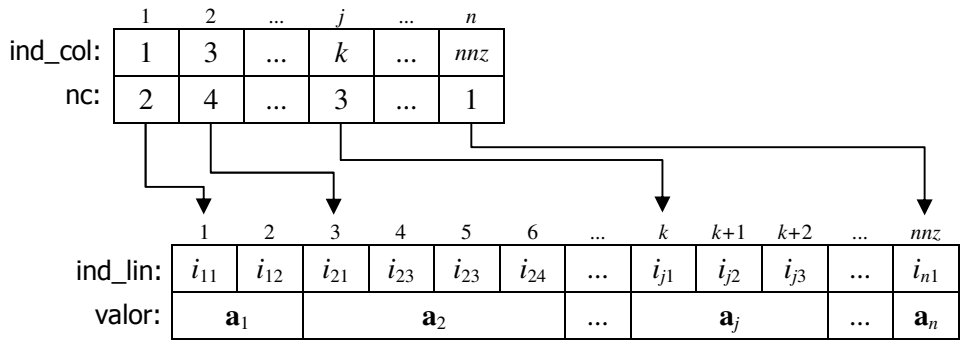

Figura 4.4: Representação compacta por colunas de uma matriz esparsa $\mathbf{A}$.

De forma semelhante, tem-se a *representação compacta por linhas* da matriz $\mathbf{A}$, quando se adota a representação compacta de suas linhas, ao invés de suas colunas. Também são necessários quatro arranjos nesse caso, como ilustrado na Figura 4.5, porém agora a disposição dos elementos é orientada por linhas. O início de cada linha $\mathbf{A}_i$, $i = 1, \ldots, m$, no arranjo valor é dada na i-ésima posição de ind_lin e, na mesma posição do arranjo nr tem-se o número de elementos não-nulos. Os elementos não-nulos de uma dada linha são armazenados de forma contígua no arranjo valor e os índices de suas respectivas colunas são armazenados em ind_col, em posições equivalentes. O índice do t-ésimo elemento armazenado da linha $\mathbf{A}_i$ é denotado por j_{it} na Figura 4.5.

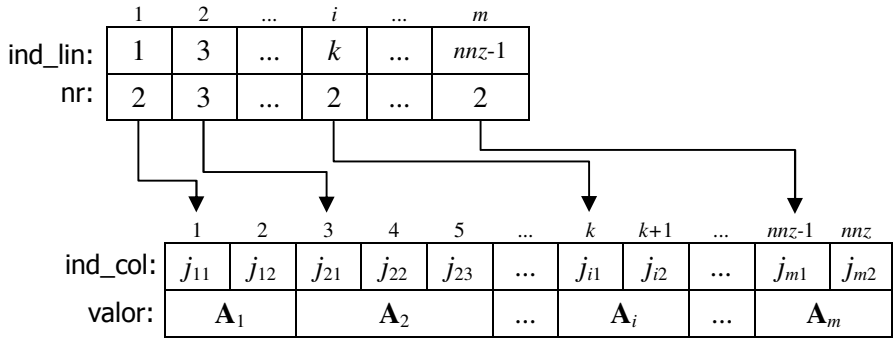

Figura 4.5: Representação compacta por linhas de uma matriz esparsa $\mathbf{A}$.

A forma de acesso aos elementos da matriz é bastante crítica para as representações descritas. Na representação compacta por colunas, os elementos de uma determinada coluna da matriz são facilmente acessados. Para isso, basta obter a posição do primeiro elemento em ind_col e percorrer de forma seqüencial o arranjo valor, de acordo com o número de elementos não-nulos especificado em nc. Por outro lado, se o intuito for acessar os elementos de uma determinada linha da matriz, deve-se percorrer todos os nnz elementos de valor, uma operação computacionalmente cara. De forma semelhante, a representação compacta por linhas favorece o acesso aos elementos de uma mesma linha, enquanto o acesso aos elementos de uma coluna específica é prejudicado.

Assim, a escolha de qual das representações descritas implementar depende de como os dados são utilizados. Por exemplo, no método primal simplex a matriz de coeficientes de um PL a ser resolvido é, em geral, representada na forma compacta por colunas, pois são acessados os elementos de uma determinada coluna durante a resolução. Algumas implementações mantêm as duas representações para a matriz de coeficientes, quando possível, já que certas operações podem exigir o acesso aos elementos de uma mesma linha.

Outro ponto a ser discutido é quanto à necessidade de atualização da matriz. Se os elementos da matriz são modificados após a definição de sua representação e existir a possibilidade de elementos nulos se tornarem não-nulos, devem ser deixados espaços livres nos arranjos para que estes novos elementos possam ser armazenados. Considerando a representação compacta por colunas, são mantidas algumas posições livres nos arranjos ind_lin e valor após cada coluna e uma *região livre* ao final destes vetores. Na Figura 4.6, é ilustrada esta representação, destacando-se as posições livres em cinza.

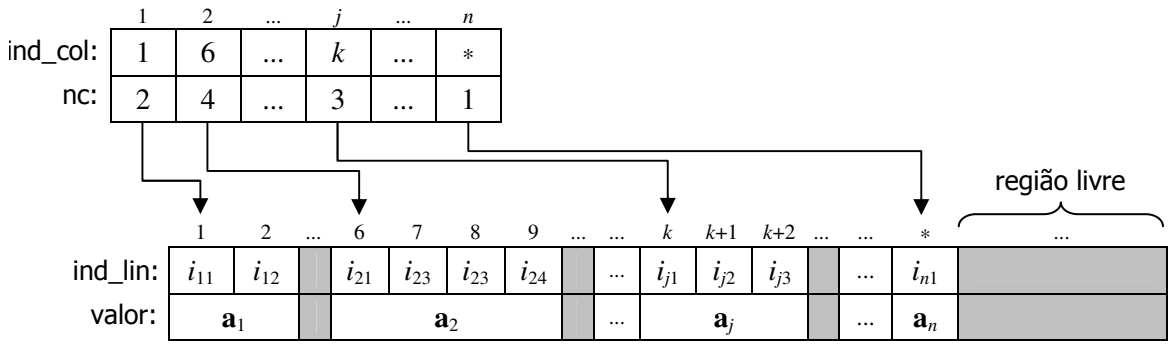

Figura 4.6: Representação compacta por colunas mantendo-se posições livres.

Uma posição livre na representação compacta por colunas pode ser sinalizada definindo-a igual a -1 no arranjo ind_lin, já que as posições ocupadas deste arranjo correspondem a índices de colunas e, portanto, devem ter valor positivo. O objetivo da região livre é o armazenamento dos novos elementos não-nulos quando as posições livres após suas respectivas colunas já estão todas ocupadas. Como os elementos de uma mesma coluna devem ser armazenados de forma contígua, sempre que não houver espaço livre após uma coluna, todos os seus elementos são movidos para a região livre. Quando o espaço da região livre é excedido, deve-se realizar uma coleta de lixo (do inglês, *garbage collection*) para a recuperação de espaços perdidos (Suhl e Suhl, 1990). Toda essa discussão é estendida de forma direta para a representação compacta por linhas.

Capítulo 5

Núcleo numérico

A resolução computacional de um problema está sujeita à introdução de erros. A fase de modelagem, a escolha do método numérico a ser aplicado e a realização de cálculos em uma aritmética de precisão finita são origens de diferentes tipos de erros que podem prejudicar a obtenção de um resultado correto.

Dado um problema real a ser resolvido, a primeira etapa é obter um modelo matemático que possa representá-lo. O modelo elaborado deve representar o problema original da melhor maneira possível, de modo que sua resolução leve, de fato, a uma solução do problema real. Entretanto, a modelagem precisa simplificar certas características do problema real para que o modelo não se torne complexo demais e inviabilize a sua resolução. Em alguns casos, certos valores são definidos de forma aproximada no modelo, pois não é possível obtê-los ou representá-los de forma exata. Além de simplificações na etapa de modelagem, um modelo pode ser originado a partir da resolução de outro problema, cuja solução também está sujeita a erros. As situações descritas podem resultar na introdução de erros cuja origem é chamada de *incerteza dos dados*.

Finalizada a etapa de modelagem do problema, o próximo passo é resolvê-lo. A escolha de um método numérico adequado é essencial. Deve-se conhecer as características do método e, principalmente, o modelo que será resolvido, para decidir se o método é aplicável ou não. Alguns métodos são capazes de obter a solução exata do modelo, a menos de erros originados da pelo uso de aritmética de precisão finita. Outros obtém a solução por meio de aproximações, as quais dão origem a erros de *truncamento*. Os métodos tipo simplex são livres de erros de truncamento.

Os números são representados em um computador por uma quantidade limitada de dígitos. Assim, os cálculos são realizados em uma aritmética de precisão finita e estão sujeitos à introdução de erros de *arredondamento*. Em métodos estáveis estes erros não interferem de forma relevante na obtenção da solução, pois vão se anulando durante a resolução do problema. Porém, quando os erros de arredondamento se acumulam durante a resolução, prejudicando o resultado final, tem-se a *instabilidade numérica*.

Em geral, os métodos tipo simplex são sensíveis aos erros de arredondamento e precisam que técnicas adequadas sejam utilizadas. A instabilidade numérica pode ter conseqüências no resultado final, como a obtenção de uma solução ótima sem precisão numérica e a constatação de infactibilidade em um problema factível, ou durante a resolução do problema, como a definição de uma base singular (ou quase-singular) e a perda de factibilidade.

O núcleo numérico de uma implementação é formado pelas técnicas que buscam manter a estabilidade do método. Uma técnica essencial é a utilização de diferentes tolerâncias nas comparações de valores reais, conforme descrito na Seção 5.1. Na seção 5.2, é apresentada uma modificação para o teste da razão, cujo objetivo é manter a estabilidade numérica do método selecionando-se os índices correspondentes a bons pivôs. A estabilidade nos cálculos envolvendo a matriz básica, ou sua inversa, dependem do número de condição desta matriz. Assim, a mudança de escala tem como objetivo operações simples sobre os dados do problema com o intuito de melhorar o condicionamento das matrizes utilizadas. Este assunto é tratado na Seção 5.3. A parte principal do núcleo numérico é a técnica de representação da matriz básica. Por levar a uma discussão bastante extensa, este assunto é tratado à parte, no Capítulo 6.

5.1 Tolerância numérica

Em um algoritmo, as decisões tomadas são baseadas em testes comparativos que envolvem os valores de variáveis definidos durante a execução. Devido aos erros de arredondamento, o valor atribuído a uma variável após um determinado cálculo pode corresponder a um valor aproximado e este fato deve ser considerado em comparações. Por exemplo, ao se utilizar um método numérico para a resolução de uma determinada equação não-linear $h(x) = 0$, não se deve esperar que a solução obtida $\tilde{x}$ satisfaça a equação de forma exata. De fato, se a implementação do método se basear na comparação de $h(\tilde{x})$ com zero para considerar $\tilde{x}$ uma solução, não há convergência na grande maioria dos casos. O que se pode esperar na prática é que $|h(\tilde{x})| < \delta$, sendo δ um valor próximo de zero, chamado de *tolerância*.

Conforme observado por Oliveira e Stewart (2006), mesmo quando uma variável real a recebe o valor de uma outra variável real b, a comparação a = b pode ser falsa. Isto ocorre pois as variáveis podem ser armazenadas em locais diferentes do computador, que utilizam precisões diferentes para representar uma variável real.

Desta forma, comparações envolvendo variáveis com valores reais devem fazer o uso de tolerâncias. As comparações do tipo a = b devem ser implementadas como |a - b| $< \delta$ e, em geral, diferentes trechos do algoritmo podem adotar diferentes tolerâncias. Isso ocorre pois a tolerância escolhida para a comparação de algumas variáveis pode não ser adequada para a comparação de outras devido à grandeza dos valores.

Em alguns casos, o uso da tolerância deve considerar outros fatores além da expressão de comparação. Para ilustrar, considere o teste da razão do método primal simplex, descrito no Capítulo 2, utilizado para o cálculo do tamanho de passo em uma dada iteração. Para simplificar, suponha que $s_k < 0$ e a nova solução seja dada por $\mathbf{x} + \varepsilon_q \mathbf{d}$. Assim, ε_q deve ser escolhido de modo a garantir a factibilidade das componentes básicas da solução primal, isto é,

$$l_{\mathcal{B}_i} \leq x_{\mathcal{B}_i} + \varepsilon_q d_{\mathcal{B}_i} \leq u_{\mathcal{B}_i}, \quad i = 1, \ldots, m. \tag{5.1}$$

Para isso, é preciso comparar cada componente com seus respectivos limitantes. Porém, observe que a nova solução corresponde a valores calculados que, provavelmente, contêm erros de arredondamento. Além disso, estes erros são acumulados pois a nova solução é calculada a partir da solução anterior. Logo, deve-se utilizar uma tolerância δ_p para

testar a factibilidade da nova solução, substituindo (5.1) por

$$l_{\mathcal{B}_i} - \delta_p \leq x_{\mathcal{B}_i} + \varepsilon_q d_{\mathcal{B}_i} \leq u_{\mathcal{B}_i} + \delta_p, \quad i = 1, \ldots, m.$$

Com isso, ε_q deve ser calculado como o $\min_i\{\varepsilon_i\}$, tal que

$$\varepsilon_i = \begin{cases} (\bar{x}_{\mathcal{B}_i} - l_{\mathcal{B}_i} + \delta_p)/d_{\mathcal{B}_i}, & \text{se } d_{\mathcal{B}_i} > 0 \text{ e } l_{\mathcal{B}_i} > -\infty, \\ (\bar{x}_{\mathcal{B}_i} - u_{\mathcal{B}_i} - \delta_p)/d_{\mathcal{B}_i}, & \text{se } d_{\mathcal{B}_i} < 0 \text{ e } u_{\mathcal{B}_i} < \infty, \\ +\infty, & \text{caso contrário.} \end{cases} \tag{5.2}$$

A implementação computacional considerando esta expressão resulta em maior estabilidade numérica quando comparada à expressão (2.17). Em grande parte dos casos, o método pode não encontrar a solução ótima do problema se estas tolerâncias são ignoradas.

Além da tolerância de factibilidade, outras tolerâncias precisam ser definidas em métodos tipo simplex. Koberstein (2005) sugere a utilização de seis tolerâncias distintas, descritas na Tabela 5.1. As três primeiras tolerâncias são usadas em comparações no *pricing* e no teste de razão. O valor absoluto do pivô selecionado pelo teste da razão não pode ser menor que um valor escolhido entre $[10^{-7}, 10^{-5}]$. Uma variável é considerada igual a zero, quando seu valor absoluto for menor que 10^{-12}. A última tolerância é utilizada no cálculo da representação da matriz básica, para definir elementos muito pequenos iguais a zero. De acordo com o autor, o uso de tolerâncias é a técnica mais importante para se manter a estabilidade numérica de um método tipo simplex.

Tolerância	Utilização
10^{-7}	Teste de factibilidade primal
10^{-9}	Teste de factibilidade primal relativa
10^{-7}	Teste de factibilidade dual
$[10^{-7}, 10^{-5}]$	Valor do pivô
10^{-12}	Comparação com zero
10^{-14}	Teste para anular elemento

Tabela 5.1: Tolerâncias para métodos tipo simplex (Koberstein, 2005).

5.2 Teste da razão de Harris

O teste da razão é uma operação bastante crítica em um método tipo simplex. Além de ser necessária a utilização de uma tolerância de factibilidade, conforme descrito, é essencial que o pivô selecionado não seja próximo de zero. Harris (1973) propõe uma modificação no teste da razão que utiliza a tolerância de factibilidade para ajudar na seleção de um índice q com grandes chances de proporcionar um bom pivô. A técnica é resumida a seguir, no contexto do método primal simplex.

O teste da razão de Harris é realizado em duas fases. A primeira fase consiste em calcular o tamanho de passo $\bar{\varepsilon} = \min_i\{\varepsilon_i\}$ utilizando a tolerância de factibilidade δ_p, assim como em (5.2). Na fase seguinte, calcula-se novamente cada ε_i porém descartando-se a tolerância, isto é, usando a expressão (2.17). Dentre todos os índices i que satisfizerem $\varepsilon_i \leq \bar{\varepsilon}$, deve-se escolher como q o correspondente ao maior pivô em valor absoluto.

Maros (2003a) cita que a proposta de Harris resulta em uma grande melhoria na estabilidade numérica de um método tipo simplex. De acordo com o autor, toda implementação eficiente e estável de um método tipo simplex deve incluir este teste da razão, ou algum outro semelhante.

5.3 Mudança de escala

Os erros numéricos em operações envolvendo matrizes são, em geral, dependentes do condicionamento da matriz. De fato, o número de condição da matriz é utilizado no cálculo de um limitante superior para o erro da solução obtida para um sistema linear. Em sistemas lineares descritos por uma matriz bem-condicionada, o erro relativo é pequeno e não interfere na qualidade do resultado obtido. Entretanto, operações envolvendo matrizes mal-condicionadas podem resultar em soluções totalmente incorretas. A matriz de um sistema linear pode ter suas linhas e colunas multiplicadas por constantes, ou escalares, com o intuito de melhorar o seu condicionamento. Esta técnica mantém inalterado o conjunto de soluções e recebe o nome de *mudança de escala*.

No contexto da otimização linear, a mudança de escala consiste em se definir as matrizes diagonais $\mathbf{R} = diag(r_1, r_2, \ldots, r_m)$ e $\mathbf{S} = diag(s_1, s_2, \ldots, s_n)$ e aplicá-las aos dados do problema, obtendo-se:

$$\begin{aligned} \text{minimizar} \quad & f(\mathbf{x}) = \bar{\mathbf{c}}^t \bar{\mathbf{x}} \\ \text{sujeito a} \quad & \mathbf{RAS}\bar{\mathbf{x}} = \mathbf{Rb} \\ & \bar{\mathbf{l}} \leq \bar{\mathbf{x}} \leq \bar{\mathbf{u}} \end{aligned} \tag{5.3}$$

com $\bar{\mathbf{x}} = \mathbf{S}^{-1}\mathbf{x}$, $\bar{\mathbf{c}} = \mathbf{cS}$, $\bar{\mathbf{l}} = \mathbf{S}^{-1}\mathbf{l}$ e $\bar{\mathbf{u}} = \mathbf{S}^{-1}\mathbf{u}$. Os elementos $r_1, r_2, \ldots, r_m$ e $s_1, s_2, \ldots, s_n$ recebem o nome de fatores de mudança de escala. A matriz $\mathbf{R}$ é responsável pela mudança de escala das restrições do problema, enquanto a matriz $\mathbf{S}$ realiza a mudança de escala das variáveis.

O principal objetivo na mudança de escala de um problema é melhorar as propriedades numéricas de sua formulação. Com isso, a resolução numérica por meio de um método tipo simplex pode se tornar mais estável, evitando que erros de arredondamento prejudiquem a qualidade da solução final. A mudança de escala também pode levar à redução do tempo computacional para a resolução de um problema, como verificado no Capítulo 7.

Maros e Mitra (1996) citam que a mudança de escala da matriz de coeficientes faz com que o intervalo de seus elementos seja reduzido. Além disso, as operações realizadas resultam em um erro relativo muito pequeno e, assim, os elementos podem ser considerados livres de erro numérico após a mudança de escala.

Tomlin (1975) faz um levantamento sobre os principais métodos de mudança de escala e apresenta os resultados obtidos pela utilização desta técnica em problemas de otimização linear. De acordo com o autor, a mudança de escala não deve ser aplicada a um problema bem formulado, podendo prejudicar sua resolução. Porém, Benichou et al. (1977) ressaltam que poucos problemas reais são bem formulados e, dessa maneira, a implementação de um método tipo simplex para a resolução de problemas em geral deve recorrer à mudança de escala.

Os métodos apresentados por Tomlin (1975) são classificados como "ótimos" ou empíricos pelo autor. Nos métodos "ótimos", a obtenção dos fatores de mudança de escala

é feita por meio da resolução de problemas de otimização não-linear, criados a partir dos coeficientes do problema original. Por exemplo, os fatores podem ser calculados de modo que a variância dos expoentes dos novos coeficientes seja a menor possível. Conforme mostrado pelo autor, apesar de ter um custo computacional relativamente alto, a mudança de escala utilizando um método "ótimo" nem sempre melhora o condicionamento da matriz.

Em métodos empíricos, os fatores de mudança de escala são obtidos por meio de operações simples sobre os coeficientes do problema, sempre considerando uma única linha ou coluna para se definir cada fator. Tomlin (1975) descreve três métodos empíricos, sendo eles:

- *Método de equilíbrio.* Os fatores de mudança de escala de linhas são calculados por $r_i = 1/\max_j\{|a_{ij}|\}$, para $i = 1, \ldots, m$, e aplicados sobre cada linha da matriz. Em seguida, os fatores de mudança de escala de colunas são calculados por $s_j = 1/\max_i\{|a_{ij}|\}$, para $j = 1, \ldots, n$, e aplicados sobre cada coluna da matriz.

- *Método de média aritmética.* Sejam nr_i o número de elementos não-nulos na linha i e nc_j o número de elementos não-nulos na coluna j. Para cada linha $i = 1, \ldots, m$ da matriz de coeficientes, calcula-se o fator de mudança de escala dado por $r_i = 1/(\sum_j |a_{ij}|/nr_i)$, que é aplicado sobre os elementos da linha. Em seguida, para cada coluna $j = 1, \ldots, n$, o fator de mudança de escala é calculado por $s_j = 1/(\sum_i |a_{ij}|/nc_j)$ e aplicado sobre os elementos da coluna.

- *Método de média geométrica.* Semelhante aos métodos anteriores, porém os fatores de mudança de escala são calculados utilizando-se a média geométrica dos elementos, isto é, $r_i = 1/(\max_j\{|a_{ij}|\}.\min_j\{|a_{ij}|\})^{1/2}$ e $s_j = 1/(\max_i\{|a_{ij}|\}.\min_i\{|a_{ij}|\})^{1/2}$, para $i = 1, \ldots, m$ e $j = 1, \ldots, n$.

Tomlin ainda ressalta que a combinação dos métodos pode levar a resultados ainda melhores. Baseado em seus testes, o autor recomenda a mudança de escala utilizando o método de média geométrica seguido de equilíbrio, pois essa combinação se mostrou superior às demais. De fato, esta observação é constatada no Capítulo 7.

Benichou et al. (1977) apresenta um método de mudança de escala bastante eficiente baseado na combinação dos métodos empíricos propostos por Tomlin. Este método consiste na aplicação iterativa do método de média geométrica, utilizando como teste de parada a variância dos nnz elementos não-nulos da matriz, dada por

$$\frac{1}{nnz}\left(\sum_{ij} a_{ij}^2 - \frac{\left(\sum_{ij}|a_{ij}|\right)^2}{nnz}\right),$$

verificada após cada mudança de escala pelo método de média geométrica. O método é finalizado se a variância calculada for menor que um determinado parâmetro, sugerido como 10 pelos autores, ou se 4 iterações foram executadas.

Observe que o método de equilíbrio consiste em normalizar as linhas e colunas da matriz considerando a norma infinito. A partir desta idéia, pode-se definir outros dois métodos:

- *Método de norma-1.* Os fatores de mudança de escala de linhas $r_i = 1/\sum_j |a_{ij}|$ são aplicados sobre cada linha da matriz. Em seguida, os fatores de mudança de escala de colunas $s_j = 1/\sum_i |a_{ij}|$, são aplicados sobre cada coluna da matriz.

- *Método de norma-2.* Equivalente ao anterior, porém utilizando os fatores de mudança de escala $r_i = 1/\sqrt{\sum_j a_{ij}^2}$ e $s_j = 1/\sqrt{\sum_i a_{ij}^2}$.

Apesar destes dois métodos não terem sido encontrados na literatura, apresentam bons resultados, como mostrado no Capítulo 7.

A mudança de escala é utilizada na grande maioria de implementações eficientes de métodos tipo simplex. Por exemplo, Kim et al. (2003) citam que o software LPAKO (Lim e Park, 2002) utiliza a mudança de escala por média geométrica seguida de equilíbrio e que, de acordo com os testes realizados, essa combinação resultou em uma melhor estabilidade numérica e na redução de cerca de 20% no tempo computacional para a resolução de problemas. Conforme descrito por Suhl (1994), o software MOPS utiliza um método de mudança de escala iterativo, semelhante ao proposto por Benichou et al. (1977). Bixby (1992) relata que a versão 2.0 do *software* CPLEX utiliza como padrão o método de equilíbrio e que não permite a resolução de um problema sem mudança de escala.

A utilização de métodos "ótimos" não foi observada em trabalhos da literatura descrevendo implementações eficientes de métodos tipo simplex. A razão disto está possivelmente no fato de que tais métodos apresentam resultados semelhantes aos métodos empíricos, porém possuem maior custo computacional para a obtenção dos fatores de mudança de escala. Além disso, apesar do nome atribuído, estes métodos são heurísticas e também podem levar a resultados insatisfatórios como apresentado por Tomlin (1975).

Antes de finalizar esta seção, seguem algumas considerações a respeito da implementação computacional. A mudança de escala da matriz de coeficientes é realizada apenas sobre a submatriz $\mathbf{A}_{\mathcal{S}}$, sendo $\mathcal{S}$ o conjunto de índices de variáveis estruturais. Isto é feito pois as colunas lógicas e artificiais são tratadas implicitamente e seus coeficientes são numericamente favoráveis (iguais a 1 em valor absoluto). Os fatores de mudança de escala podem ser armazenados em dois arranjos unidimensionais, um com m posições para os fatores de mudança de escala de linhas e o outro com $\bar{n}$, para os fatores de colunas. A rigor, um fator não é armazenado na forma de razão. Ao invés disso, armazena-se o denominador da razão que o define. Por exemplo, na mudança de escala por equilíbrio, armazena-se o maior elemento em valor absoluto da linha e a mudança de escala é feita dividindo-se cada elemento da linha por esse valor. O armazenamento dos fatores após a mudança de escala se faz necessário pois eles são utilizados após a resolução do problema, já que a mudança de escala de colunas altera as variáveis do problema, as quais devem ser recuperadas no final.

Capítulo 6

Representação da matriz básica

Em um método tipo simplex, as principais operações de uma iteração consistem em calcular os produtos do tipo $\boldsymbol{\alpha} = \mathbf{B}^{-1}\boldsymbol{v}$ e $\boldsymbol{\alpha}^T = \boldsymbol{v}^T\mathbf{B}^{-1}$ ou, de forma equivalente, em resolver os sistemas lineares da forma $\mathbf{B}\boldsymbol{\alpha} = \boldsymbol{v}$ e $\mathbf{B}^T\boldsymbol{\alpha} = \boldsymbol{v}$. Como estas operações são realizadas mais de uma vez por iteração, é essencial se manter uma representação da matriz básica, ou de sua inversa, capaz de contribuir com a eficiência e a estabilidade numérica do método.

Uma representação "ingênua" seria calcular e armazenar de forma explícita a inversa da matriz básica. Apesar de conceitualmente simples e correta, esta técnica se torna computacionalmente inviável na resolução de problemas de grande porte. De fato, todos os elementos de $\mathbf{B}^{-1}$ são armazenados, independente de serem nulos ou não, e a quantidade de memória utilizada depende das dimensões do problema e não do número de elementos não-nulos. Tem-se também que o preenchimento, o número de elementos não-nulos criados, é muito grande na representação explícita da inversa. Isto faz com que mais operações aritméticas sejam realizadas, acarretando na elevação do tempo computacional. Como os elementos criados correspondem ao resultado de cálculos em uma aritmética de precisão finita, pode ocorrer a instabilidade numérica do método. Logo, esta técnica é impraticável e deve-se procurar maneiras mais eficientes de representação.

Técnicas para a representação da matriz básica, ou de sua inversa, são essenciais na implementação eficiente e estável de métodos tipo simplex. Por ser uma submatriz da matriz de coeficientes, tem-se que $\mathbf{B}$ é esparsa e, assim, uma estrutura de dados capaz de explorar esta característica deve ser utilizada. A técnica escolhida deve levar em conta a esparsidade e a estabilidade numérica da representação, e o tempo computacional para obtê-la e atualizá-la deve ser o menor possível.

Atualmente, a técnica de representação mais eficiente em métodos tipo simplex é a decomposição LU e sua atualização conforme propostas por Suhl e Suhl (1990, 1993), denotada aqui por SSLU. A descrição completa da técnica SSLU é dada na Seção 6.2. Esta proposta é reconhecida como o estado-da-arte na resolução de problemas de grande porte, por trabalhos relevantes da literatura (Bixby, 2002; Maros, 2003a).

Uma alternativa para a SSLU é a técnica conhecida como Forma Produto da Inversa (FPI), proposta por Dantzig e Orchard-Hays (1954). Em geral, a FPI produz uma representação menos esparsa e de menor estabilidade numérica que a técnica proposta por Suhl e Suhl. Por outro lado, sua implementação é relativamente simples e pode ser considerada como um primeiro passo na implementação de uma técnica mais elaborada. Os detalhes da FPI são apresentados na Seção 6.1.

6.1 Forma Produto da Inversa

A Forma Produto da Inversa (FPI) foi proposta por Dantzig e Orchard-Hays (1954) ao observarem que, em uma troca de base, a inversa da nova matriz básica pode ser obtida a partir da inversa da matriz básica atual. Para isto, é necessário se pré-multiplicar a inversa da matriz básica atual por uma matriz de transformação elementar, conforme descrito na seqüência.

Uma matriz de transformação elementar é toda matriz que se diferencia da matriz identidade de mesma ordem por uma única linha ou coluna não-trivial. Seja $\mathbf{E}$ uma matriz de transformação elementar de ordem m, com uma coluna não-trivial $\boldsymbol{\eta}$ na posição q. Então,

$$\mathbf{E} = [\mathbf{e}_1, \mathbf{e}_2, \ldots, \boldsymbol{\eta}, \ldots, \mathbf{e}_m] = \begin{bmatrix} 1 & & & \eta_1 & & \\ & 1 & & \eta_2 & & \\ & & \ddots & \vdots & & \\ & & & \eta_q & & \\ & & & \vdots & \ddots & \\ & & & \eta_m & & 1 \end{bmatrix}, \tag{6.1}$$

sendo $\mathbf{e}_i$ a i-ésima coluna da matriz identidade de ordem m, $1 \leq i \leq m$. Note que devido à sua estrutura, uma matriz de transformação elementar pode ser bem representada de forma compacta pelo par $(\boldsymbol{\eta}, q)$.

Seja $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_m]$ a matriz básica de uma dada iteração. Suponha que a variável não-básica x_k tenha sido selecionada para substituir a q-ésima variável básica. Representando-se por $\bar{\mathbf{B}}$ a matriz básica associada à nova base, tem-se

$$\bar{\mathbf{B}} = [\mathbf{b}_1, \ldots, \mathbf{b}_{q-1}, \mathbf{a}_k, \mathbf{b}_{q+1}, \ldots, \mathbf{b}_m].$$

Como $\mathbf{a}_k$ é um vetor m-dimensional, é possível escrevê-lo como a combinação linear das colunas de $\mathbf{B}$, isto é,

$$\mathbf{a}_k = \sum_{i=1}^{m} \beta_i \mathbf{b}_i = \mathbf{B}\boldsymbol{\beta}.$$

Se $\beta_q \neq 0$ então $\mathbf{b}_q$ pode ser isolado na expressão acima obtendo-se

$$\mathbf{b}_q = \frac{1}{\beta_q}\mathbf{a}_k - \sum_{i \neq q} \frac{\beta_i}{\beta_q}\mathbf{b}_i,$$

que é a representação de $\mathbf{b}_q$ como combinação linear das colunas de $\bar{\mathbf{B}}$. Definindo-se o vetor

$$\boldsymbol{\eta} = \left[-\frac{\beta_1}{\beta_q}, \ldots, -\frac{\beta_{q-1}}{\beta_q}, \frac{1}{\beta_q}, -\frac{\beta_{q+1}}{\beta_q}, \ldots, -\frac{\beta_m}{\beta_q}\right]^T,$$

tem-se que $\mathbf{b}_q = \bar{\mathbf{B}}\boldsymbol{\eta}$. Desta forma, multiplicando-se a matriz básica atual pela inversa de $\bar{\mathbf{B}}$, resulta em

$$\begin{aligned} \bar{\mathbf{B}}^{-1}\mathbf{B} &= [\mathbf{e}_1, \ldots, \mathbf{e}_{q-1}, \bar{\mathbf{B}}^{-1}\mathbf{b}_q, \mathbf{e}_{q+1}, \ldots, \mathbf{e}_m] \\ &= [\mathbf{e}_1, \ldots, \mathbf{e}_{q-1}, \boldsymbol{\eta}, \mathbf{e}_{q+1}, \ldots, \mathbf{e}_m] \\ &= \mathbf{E}, \end{aligned}$$

sendo $\mathbf{E}$ uma matriz de transformação elementar cuja coluna não-trivial é dada por $\boldsymbol{\eta}$. Pela expressão obtida,

$$\bar{\mathbf{B}}^{-1} = \mathbf{E}\mathbf{B}^{-1}.$$

Logo, a inversa da nova matriz básica é obtida pela pré-multiplicação da matriz básica atual por uma matriz de transformação elementar.

O fato de uma troca de base poder ser representada por uma matriz de transformação elementar permite a atualização da inversa da matriz básica, evitando-se inverter $\mathbf{B}$ a cada troca de base. Considerando-se que a matriz básica inicial é dada pela matriz identidade de ordem m, tem-se que após p trocas de base a inversa da matriz básica corrente é dada pelo produto

$$\mathbf{B}^{-1} = \mathbf{E}_p\mathbf{E}_{p-1}\ldots\mathbf{E}_1. \tag{6.2}$$

Por esta razão, a técnica recebe o nome de Forma Produto da Inversa. A partir desta expressão, pode-se observar que a inversa da matriz básica pode ser representada de forma compacta, bastando-se armazenar a seguinte seqüência de p pares *elementares*, correspondentes às matrizes $\mathbf{E}_1, \ldots, \mathbf{E}_p$:

$$(\boldsymbol{\eta}^1, q^1), \ldots, (\boldsymbol{\eta}^{p-1}, q^{p-1}), (\boldsymbol{\eta}^p, q^p). \tag{6.3}$$

Os vetores $\boldsymbol{\eta}$ são esparsos e podem ser armazenados utilizando-se a representação compacta, discutida no Capítulo 4. Assim, a estrutura de dados utilizada para o armazenamento dos pares elementares é bastante semelhante àquela usada no armazenamento de matrizes esparsas. Na Figura 6.1, tem-se uma ilustração desta estrutura. Os índices i_{tj} correspondem ao j-ésimo elemento armazenado do vetor $\boldsymbol{\eta}^t$. Note que o primeiro elemento armazenado de cada vetor $\boldsymbol{\eta}^t$ é aquele de índice q^t, isto é, $i_{t1} = q^t$. Desta forma, evita-se a utilização de um vetor adicional para o armazenamento dos índices correspondentes à posição dos vetores em suas respectivas matrizes de transformação elementar.

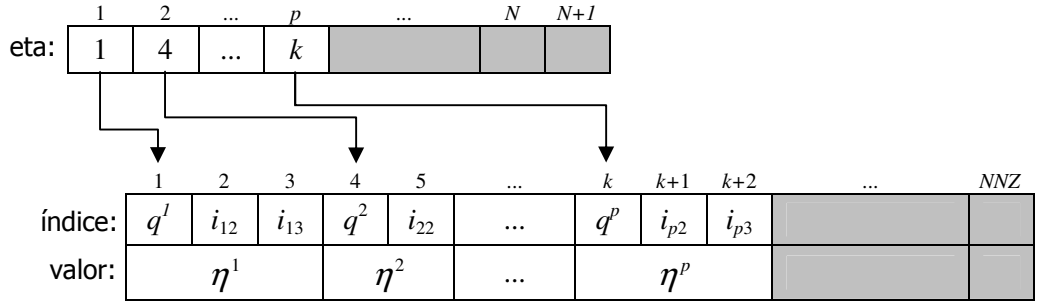

Figura 6.1: Estrutura de dados para o armazenamento dos pares elementares.

Ainda na Figura 6.1, observe que um espaço adicional livre, denotado em cinza, é deixado para o armazenamento de novas matrizes de transformação elementar, que venham a ser criadas durante as trocas de base. O número máximo de vetores é denotado por N e os arranjos índice e valor têm capacidade para NNZ elementos não-nulos. O número de elementos não-nulos em cada vetor $\boldsymbol{\eta}^t$ é obtido pela diferença entre sua posição inicial e a posição inicial do vetor $\boldsymbol{\eta}^{t+1}$, armazenadas no arranjo eta. Por esta razão, este arranjo deve ter capacidade para $N + 1$ posições.

Como já mencionado, uma nova matriz de transformação elementar é incluída na representação da matriz inversa a cada troca de base. Com isso, o espaço adicional reservado para o armazenamento de novos pares $(\boldsymbol{\eta}^i, q^i)$ pode ser excedido. Neste caso, deve-se recorrer ao procedimento de *inversão*, que consiste em descartar a representação corrente e obter uma nova seqüência de matrizes de transformação elementar. Conforme mencionado por alguns autores, esta operação corresponde à *reinversão* da matriz básica e também pode ser utilizada quando detectada a instabilidade numérica nos cálculos de uma dada iteração. A inicialização de métodos tipo simplex também exige que se recorra ao procedimento de inversão para se obter a representação de matriz básica inicial avançada. Na Seção 6.1.2, o procedimento de inversão é descrito conceitualmente e são apresentados os algoritmos para a sua implementação.

Os cálculos envolvendo a inversa da matriz básica utilizam as matrizes de transformação elementar que a representa e, assim, são chamados de *transformações*. Dependendo da ordem em que se utiliza as matrizes de transformação elementar, a transformação pode ser progressiva ou regressiva, conforme apresentado na seção a seguir.

6.1.1 Transformações

Seja $\mathbf{E}$ uma matriz de transformação elementar como descrita em (6.1). Dado um vetor m-dimensional $\boldsymbol{\upsilon}$, o produto $\boldsymbol{\alpha} = \mathbf{E}\boldsymbol{\upsilon}$ é dado por

$$\alpha_i = \begin{cases} \upsilon_i + \eta_i \upsilon_q, & \text{se } i \neq q, \\ \eta_q \upsilon_q, & \text{se } i = q, \end{cases} \tag{6.4}$$

para $i = 1, \ldots, m$. Por esta expressão é possível ver que se $\upsilon_q = 0$, então o resultado deste produto é dado por $\boldsymbol{\alpha} = \boldsymbol{\upsilon}$. Verificar se esta condição é verdadeira antes de se iniciar o cálculo é importante e evita que operações desnecessárias sejam realizadas. Outro ponto a ser ressaltado é que os índices de elementos não-nulos em $\boldsymbol{\alpha}$ correspondem aos índices de elementos não-nulos em $\boldsymbol{\upsilon}$ e $\boldsymbol{\eta}$, isto é, tem-se que $\alpha_i \neq 0$ quando $\eta_i \neq 0$ ou $\upsilon_i \neq 0$, a menos de cancelamento. Assim, o número de elementos não-nulos em $\boldsymbol{\upsilon}$ pode ser muito maior do que em $\boldsymbol{\alpha}$, dependendo do padrão de esparsidade de $\boldsymbol{\eta}$.

Considere a representação da inversa de uma matriz básica por matrizes de transformação elementar conforme (6.2). Um produto do tipo $\boldsymbol{\alpha} = \mathbf{B}^{-1}\boldsymbol{\upsilon}$ é calculado por

$$\boldsymbol{\alpha} = \mathbf{E}_p \mathbf{E}_{p-1} \ldots \mathbf{E}_1 \boldsymbol{\upsilon}, \tag{6.5}$$

iniciando-se com o cálculo de $\boldsymbol{\alpha} = \mathbf{E}_1\boldsymbol{\upsilon}$ e prosseguindo-se com $\boldsymbol{\alpha} = \mathbf{E}_j\boldsymbol{\alpha}$, para $j = 2, \ldots, p$. Cada um destes produtos é realizado de acordo com a expressão (6.4).

Devido à aplicação das matrizes elementares ser em ordem crescente de seus índices, o cálculo em (6.5) recebe o nome de *transformação progressiva*, ou FTRAN (do inglês, *Forward Transformation*). A FTRAN é descrita no Algoritmo 2.

Uma matriz de transformação elementar também pode ser utilizada em um produto do tipo $\boldsymbol{\alpha}^T = \boldsymbol{\upsilon}^T\mathbf{E}$. Neste caso, tem-se, para $i = 1, \ldots, m$,

$$\alpha_i = \begin{cases} \upsilon_i, & \text{se } i \neq q, \\ \displaystyle\sum_{j=1}^{m} \eta_j \upsilon_j, & \text{se } i = q. \end{cases} \tag{6.6}$$

Algoritmo 2: FTRAN (FPI)

Entrada: Vetor $\boldsymbol{v}$; seqüência $(\boldsymbol{\eta}^1, q^1), \ldots, (\boldsymbol{\eta}^{p-1}, q^{p-1}), (\boldsymbol{\eta}^p, q^p)$ que representa $\mathbf{B}^{-1}$; p; δ.
Saída: $\boldsymbol{\alpha} = \mathbf{B}^{-1}\boldsymbol{v}$.

```
1  /* Inicializar o vetor α */
2  α = v;
3
4  /* Pré-multiplicar α por cada matriz de transformação elementar */
5  Para (i = 1 até p, passo 1) faça
6  {
7     /* Obter o par (η^i, q^i) correspondente à matriz de transformação elementar E_i */
8     (η, q) = (η^i, q^i);
9
10    /* Verificar se α_q ≠ 0 */
11    Se (|α_q| > δ) então
12    {
13       aux = α_q;
14       Para cada (η_j ≠ 0, j = 1, ..., m) faça α_j = α_j + aux * η_j;
15       α_q = aux * η_q;
16    }
17 }
```

Neste produto, não é possível se beneficiar de uma componente nula, como em (6.4). Por outro lado, observe que $\boldsymbol{\alpha}$ tem, no máximo, um elemento não-nulo a mais do que $\boldsymbol{v}$, independentemente do padrão de esparsidade de $\boldsymbol{\eta}$.

A expressão (6.6) é utilizada repetidamente na transformação de um vetor $\boldsymbol{v}$ pela inversa da matriz básica, quando representada por uma seqüência de matrizes de transformação elementar. Tem-se que $\boldsymbol{\alpha}^T = \boldsymbol{v}^T\mathbf{B}^{-1}$ é dado pelo produto

$$\boldsymbol{\alpha}^T = \boldsymbol{v}^T\mathbf{E}_p\mathbf{E}_{p-1}\ldots\mathbf{E}_1, \tag{6.7}$$

chamado de transformação regressiva, ou BTRAN (do inglês, *Backward Transformation*), pois o primeiro produto a ser calculado é $\boldsymbol{\alpha}^T = \boldsymbol{v}^T\mathbf{E}_p$, prosseguindo-se com $\boldsymbol{\alpha}^T = \boldsymbol{\alpha}^T\mathbf{E}_j$, para $j = p-1, \ldots, 1$. Um procedimento para o cálculo da expressão (6.7) é dado no Algoritmo 3.

6.1.2 Inversão

Dada uma matriz básica formada pelas colunas descritas em $\mathcal{B}$, o procedimento de inversão consiste em transformá-la na matriz identidade de mesma ordem, utilizando-se um número finito $p \leq m$ de matrizes de transformação elementar. Assim, considerando de forma implícita eventuais permutações de linhas e colunas de $\mathbf{B}$, o procedimento obtém

$$\mathbf{I} = \mathbf{E}_p\mathbf{E}_{p-1}\ldots\mathbf{E}_1\mathbf{B}. \tag{6.8}$$

Logo, a inversa de $\mathbf{B}$ é dada pelo produto destas p matrizes de transformação elementar, conforme (6.2). Na prática, não há necessidade de se obter a matriz identidade de forma explícita como em (6.8). A notação é utilizada apenas para a exposição didática.

Algoritmo 3: BTRAN (FPI)

Entrada: Vetor $\boldsymbol{v}$; seqüência $(\boldsymbol{\eta}^1, q^1), \ldots, (\boldsymbol{\eta}^{p-1}, q^{p-1}), (\boldsymbol{\eta}^p, q^p)$ que representa $\mathbf{B}^{-1}$; p.
Saída: $\boldsymbol{\alpha}^T = \boldsymbol{v}^T \mathbf{B}^{-1}$.

```
1  /* Inicializar o vetor α */
2  α = v;
3
4  /* Pós-multiplicar α por cada matriz de transformação elementar */
5  Para (i = p até 1, passo −1) faça
6  {
7     /* Obter o par (η^i, q^i) correspondente à matriz de transformação elementar E_i */
8     (η, q) = (η^i, q^i);
9     /* Calcular o elemento α_q */
10    soma = 0;
11    Para cada (η_j ≠ 0, j = 1, ..., m) faça soma = soma + α_j * η_j;
12    α_q = soma;
13 }
```

As matrizes elementares na expressão (6.8) são obtidas em p passos de *pivotamento*. Cada passo t consiste em escolher um elemento *pivô* $b_{qk} \neq 0$, que será utilizado para a transformação da coluna $\mathbf{b}_k$ na coluna trivial $\mathbf{e}_q$. Diz-se que a coluna k foi *pivotada* na posição q. Então, a linha q e a coluna k são marcadas como pivotadas para que seus elementos não possam mais ser selecionados como pivôs. A transformação de $\mathbf{b}_k$ é representada na forma matricial por uma matriz de transformação elementar $\mathbf{E}_t$ e equivale a se pré-multiplicar $\mathbf{B}$ por $\mathbf{E}_t$. Com isto, as demais colunas de $\mathbf{B}$ que ainda não foram pivotadas também são modificadas.

Na Figura 6.2, tem-se uma ilustração da matriz básica durante o procedimento de inversão, após t passos de pivotamento. Para facilitar a exposição, considere que as primeiras posições da matriz foram pivotadas. Desta forma, tem-se que suas colunas $\mathbf{b}_1$ a $\mathbf{b}_t$ já foram transformadas em colunas triviais. Quanto às demais colunas, indicadas em cinza na figura, seus elementos podem ter sido modificados e, até mesmo, novos elementos não-nulos podem ter sido criados. A submatriz $\mathbf{S}$ é chamada de *submatriz ativa* e corresponde aos elementos cujos índices ainda não foram pivotados. Apenas estes elementos podem ser selecionados como pivôs nos próximos passos.

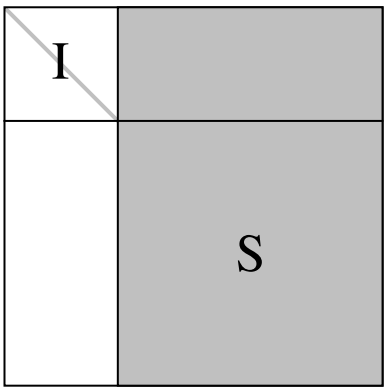

Figura 6.2: Matriz básica após t passos de pivotamento.

Apesar de uma matriz não-singular ter uma inversa única, o mesmo não é verdade para a representação de sua inversa como um produto de matrizes de transformação elementar. Existe uma certa liberdade na escolha dos pivôs e, com isso, diferentes matrizes de transformação elementar podem ser obtidas. Esta liberdade é fundamental para se obter uma implementação eficiente do procedimento de inversão, pois permite a exploração de certas características com o intuito de se reduzir o tempo computacional, garantir a esparsidade das matrizes elementares geradas e manter a estabilidade numérica durante os cálculos. Estas características podem ser analisadas em etapas, como discutido na seqüência. Antes de apresentá-las, são definidos os conceitos iniciais necessários, na seção a seguir.

Notação e conceitos iniciais

A seleção de um elemento pivô b_{qk} deve ser sinalizada para que a linha de índice q e a coluna de índice k não sejam mais consideradas nos próximos passos. Esta sinalização é feita pelos conjuntos $\mathcal{Q}$ de índices de linhas não-pivotadas e $\mathcal{K}$ de índices de colunas não-pivotadas. Inicialmente, define-se $\mathcal{Q} = \mathcal{K} = \{1, 2, \ldots, m\}$ e os índices são removidos de seus respectivos conjuntos conforme são pivotados. Ao final do procedimento, deve-se ter $\mathcal{Q} = \mathcal{K} = \varnothing$. Considerando estes conjuntos, a submatriz ativa $\mathbf{S}$ fica determinada pelos elementos b_{ij}, com $i \in \mathcal{Q}$ e $j \in \mathcal{K}$.

O número de elementos não-nulos em cada linha e coluna da submatriz ativa é utilizado na seleção de um pivô. Assim, são definidos os seguintes contadores:

- nr_i: número de elementos não-nulos na linha i da submatriz ativa, $i \in \mathcal{Q}$,
- nc_j: número de elementos não-nulos na coluna j da submatriz ativa, $j \in \mathcal{K}$.

Os índices de linhas com o mesmo número de elementos não-nulos são agrupados em conjuntos, assim como os índices de colunas, conforme definido a seguir:

$$\mathcal{R}_t = \{i \in \mathcal{Q} : nr_i = t\} \quad \text{e} \quad \mathcal{C}_t = \{j \in \mathcal{K} : nc_j = t\}, \text{ para } t = 1, \ldots, m. \tag{6.9}$$

Um vetor para armazenar a permutação de índices também é necessário. Como uma coluna pode ser pivotada em uma posição diferente da sua posição em $\mathcal{B}$, pode ser necessária uma reordenação dos elementos deste conjunto para deixá-lo de acordo com a representação da matriz básica. Esta reordenação é feita de forma eficiente utilizando-se o vetor de permutações $\boldsymbol{\pi}$, definido inicialmente por $\boldsymbol{\pi} = \mathbf{0}$. Sempre que a coluna de índice $\mathcal{B}_k$ for pivotada na posição q, mantém-se o conjunto $\mathcal{B}$ inalterado e faz-se $\boldsymbol{\pi}_k = q$. Obviamente, a coluna de índice $\mathcal{B}_q$ não poderá ser pivotada na posição q e, assim, $\boldsymbol{\pi}_q$ será modificado no momento oportuno. Ao final do procedimento, tem-se que a posição correta de cada índice $\mathcal{B}_i$ em $\mathcal{B}$ é dada por $\boldsymbol{\pi}_i$ e os elementos podem então ser movidos. A vantagem em se manter o vetor $\boldsymbol{\pi}$ desta maneira está em postergar as permutações para o final do procedimento de inversão.

Os conjuntos e contadores aqui definidos são utilizados tanto para facilitar a descrição do procedimento de inversão quanto para garantir a eficiência em sua implementação. Computacionalmente, eles podem ser representados por arranjos simples, exceto pelos conjuntos descritos na expressão (6.9). Por exemplo, para o conjunto $\mathcal{Q}$, pode-se utilizar

um arranjo de m posições. Assim, a i-ésima posição do arranjo é definida igual a 1 se $i \in \mathcal{Q}$, caso contrário é definida igual a 0.

Uma estrutura de dados para a implementação eficiente dos conjuntos $\mathcal{R}_t$ e $\mathcal{C}_t$, $t = 1, \ldots, m$, é dada por listas estáticas duplamente encadeadas. Esta estrutura permite incluir e remover de forma rápida os elementos destes conjuntos, operações realizadas com bastante freqüência durante os procedimentos descritos. Maros (2003a) descreve de forma detalhada a estrutura e os algoritmos para manipulá-la.

Heurísticas de pivotamento

A utilização dos contadores nr_i e nc_j para a seleção de um pivô, tem o objetivo de reduzir o preenchimento ocorrido durante o procedimento de inversão. Uma técnica de seleção, eficiente quanto à esparsidade, consiste em selecionar o pivô b_{qk} que satisfaça o critério de Markowitz, dado por:

$$(nr_q - 1)(nc_k - 1) = \min_{i,j}\{(nr_i - 1)(nc_j - 1) : b_{ij} \neq 0, i \in \mathcal{Q} \text{ e } j \in \mathcal{K}\}. \tag{6.10}$$

O resultado desta expressão é um limitante superior para o preenchimento em um passo do procedimento de inversão, ocasionado pela escolha do elemento b_{qk} como pivô. Entretanto, o critério de Markowitz calcula apenas o preenchimento local (desconsiderando-se eventuais cancelamentos). O termo preenchimento local é adotado pois a garantia da maior esparsidade possível da representação final exige que um problema de programação dinâmica seja resolvido, já que a seleção de um pivô influencia nos próximos passos e, portanto, nos preenchimentos posteriores. Entretanto, o tempo computacional para a obtenção do menor preenchimento global é alto, tornando-o inviável.

Na prática, nem mesmo o critério (6.10) é usado exatamente como descrito, por dois principais motivos. Em primeiro lugar, é necessário se calcular o produto $(nr_i - 1)(nc_j - 1)$ para cada elemento $b_{ij} \neq 0$ da submatriz ativa, o que é computacionalmente caro, em geral. Estes valores precisariam ser recalculados a cada passo devido à possível criação de novos elementos não-nulos. Desta forma, em implementações deste critério, o cálculo é feito para apenas uma parte dos elementos da submatriz ativa. Por exemplo, escolhe-se os índices de algumas linhas com os menores valores de nr_i e, para os elementos destas linhas calcula-se o produto $(nr_i - 1)(nc_j - 1)$. O menor dos produtos indica qual pivô deve ser selecionado. Apesar de não ser um critério ótimo, a seleção restrita a alguns elementos resulta em um tempo computacional muito menor, sem prejudicar a esparsidade da representação final de forma significativa.

Uma segunda desvantagem do critério (6.10) diz respeito à estabilidade numérica. Se o valor absoluto do elemento b_{qk} que satisfaz o critério de Markowitz for pequeno em relação aos demais elementos da matriz, sua escolha pode levar à potencialização de erros numéricos e, logo, à instabilidade numérica das operações seguintes. Esta dificuldade pode ser contornada utilizando-se um segundo critério para a seleção do pivô b_{qk}, dado por:

$$|b_{qk}| \geq u \max_i |b_{ik}|, \tag{6.11}$$

chamado de *critério de limiar* (do inglês, *threshold criterion*), sendo $u \in [0, 1]$ a *tolerância de limiar*. Este critério evita que elementos relativamente pequenos sejam escolhidos

como pivô, tomando-se o maior elemento em módulo da coluna k como parâmetro. O critério de limiar também pode considerar o maior elemento da linha q como parâmetro:

$$|b_{qk}| \geq u \max_j |b_{qj}|. \tag{6.12}$$

A combinação do critério de limiar com uma aproximação do critério de Markowitz resulta em uma heurística de pivotamento que busca o equilíbrio entre esparsidade e estabilidade numérica na representação final. Neste sentido, Suhl e Suhl (1990) propõem um procedimento eficiente para a seleção de pivô, conforme descrito no Algoritmo 4. Neste algoritmo, o critério de limiar é utilizado nas linhas 17 e 33, seguido de uma combinação com o critério de Markowitz. A função $max\_b(l)$ representa o maior elemento não-nulo em módulo da coluna l, se a expressão (6.11) for utilizada, ou da linha l, se for usada a expressão (6.12). A decisão de qual expressão utilizar é guiada pelo custo computacional para se obter o maior elemento em linhas e colunas.

Descrição do procedimento

O procedimento de inversão deve se basear no conjunto $\mathcal{B}$ de índices básicos. A representação compacta por colunas, descrita no Capítulo 4, deve ser utilizada como estrutura de dados para o armazenamento da submatriz ativa em memória principal. A princípio, esta matriz é igual à matriz básica e apenas as colunas estruturais precisam ser armazenadas. Como a submatriz ativa se modifica durante os cálculos, é necessário se manter espaços adicionais livres, conforme ilustrado na Figura 4.6. Ao final do procedimento, a submatriz ativa deve ser eliminada. O armazenamento das matrizes de transformação elementar é feito utilizando-se a estrutura de dados apresentada na Figura 6.1.

Considerando a discussão apresentada até o momento, o procedimento de inversão é descrito na seqüência, na forma de etapas. As etapas correspondem à análise de certas características da matriz básica que, quando identificadas, contribuem com a esparsidade e estabilidade numérica da representação final e resultam na redução do tempo computacional para a execução do procedimento.

Etapa 1: Seleção de pivôs em colunas lógicas

Uma primeira etapa é identificar as colunas lógicas da matriz básica. Se existirem índices de coluna lógicas em $\mathcal{B}$, então estas colunas devem ser selecionadas antes das demais por não necessitarem de transformação. De fato, as colunas lógicas já são colunas triviais e nenhuma matriz de transformação elementar precisa ser definida. Logo, basta que estas colunas sejam marcadas como pivotadas em suas posições triviais, isto é, a coluna $\mathbf{e}_i$ é pivotada na posição i, independentemente de sua posição em $\mathcal{B}$. Com isso, será necessário menos do que m matrizes elementares para a representação da inversa. Os passos desta etapa são resumidos no Algoritmo 5.

Apenas para facilitar a ilustração da etapa 1, considere que todas as colunas lógicas da matriz básica tenham sido identificadas e, utilizando-se permutações implícitas de linhas e colunas, tenham sido pivotadas nas últimas posições da matriz básica. O resultado, ao final da etapa, é dado na Figura 6.3.

Algoritmo 4: Seleção de elemento pivô combinando-se esparsidade e estabilidade numérica.

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$ e $\mathcal{K}$; contadores nr_i e nc_j, $i \in \mathcal{Q}$ e $j \in \mathcal{K}$; conjuntos $\mathcal{R}_t$ e $\mathcal{C}_t$, $t = 1, \ldots, m$; tolerância u; r_max;
Saída: Par de índices $q \in \mathcal{Q}$ e $k \in \mathcal{K}$ correspondente ao pivô b_{qk} selecionado.

```
1  /* Verificar se existem colunas ou linhas com apenas um elemento não-nulo */
2  Se (C_1 ≠ ∅) então selecione b_qk tal que k ∈ C_1 e PARE;
3  Se (R_1 ≠ ∅) então selecione b_qk tal que q ∈ R_1 e PARE;
4
5  /* Definir valores iniciais */
6  f = m^2;
7  r = 0;
8
9  /* Percorrer os demais conjuntos */
10 Para (t = 2 até m, passo 1) faça
11 {
12    /* Analisar as colunas com t elementos não-nulos */
13    Se (C_t ≠ ∅) então
14       Para cada (j ∈ C_t) faça
15       {
16          /* Selecionar um candidato a pivô */
17          nr_i = min_l{nr_l : |b_lj| ≥ u * max_b(j), l ∈ Q};
18          Se ((nr_i - 1) * (nc_j - 1) < f) então
19          {
20             f = (nr_i - 1) * (nc_j - 1);
21             b_qk = b_ij;
22             Se (f ≤ (t - 1)^2) então PARE;
23          }
24          r = r + 1;
25          Se (r ≥ r_max e f ≤ m^2) então PARE;
26       }
27
28    /* Analisar as linhas com t elementos não-nulos */
29    Se (R_t ≠ ∅) então
30       Para cada (i ∈ R_t) faça
31       {
32          /* Selecionar um candidato a pivô */
33          nc_j = min_l{nc_l : |b_il| ≥ u * max_b(l), l ∈ K};
34          Se ((nr_i - 1) * (nc_j - 1) < f) então
35          {
36             f = (nr_i - 1) * (nc_j - 1);
37             b_qk = b_ij;
38             Se (f ≤ t * (t - 1)) então PARE;
39          }
40          r = r + 1;
41          Se (r ≥ r_max e f ≤ m^2) então PARE;
42       }
43 }
```

Algoritmo 5: Etapa 1 do procedimento de inversão (FPI).

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$ e $\mathcal{K}$; vetor de permutações $\boldsymbol{\pi}$.
Saída: Matriz $\mathbf{B}$ com as colunas lógicas pivotadas em suas posições triviais.

```
1  Para cada (coluna lógica b_k) faça
2  {
3      Defina q tal que e_q = b_k;
4
5      /* Definir um novo pivô */
6      π_k = q;
7      Q = Q \ {q};
8      K = K \ {k};
9  }
```

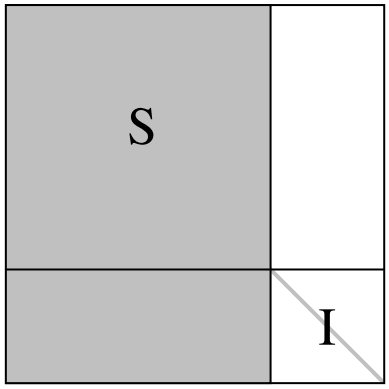

Figura 6.3: Matriz básica após a permutação de colunas lógicas.

Etapa 2: Seleção de pivôs em linhas com um único elemento não-nulo

Uma característica importante da matriz básica é que, devido a sua esparsidade, grande parte de suas linhas e colunas possuem apenas um único elemento não-nulo. Identificá-las é essencial para a eficiência do procedimento de inversão.

Considere uma coluna $\mathbf{b}_k$, $k \in \mathcal{K}$, cujo elemento b_{qk} satisfaça $nr_q = 1$, $q \in \mathcal{Q}$. Se t é o número de matrizes de transformação elementar definidas até o momento, então a transformação desta coluna em $\mathbf{e}_q$ é feita utilizando-se a matriz elementar $\mathbf{E}_{t+1}$, cuja q-ésima coluna é dada pela coluna não-trivial $\boldsymbol{\eta}^{t+1}$, definida por

$$\eta_i^{t+1} = \begin{cases} 1/b_{qk}, & \text{se } i = q, \\ -b_{ik}/b_{qk}, & \text{caso contrário.} \end{cases} \tag{6.13}$$

para $i = 1, \ldots, m$. A pré-multiplicação da matriz básica por $\mathbf{E}_{t+1}$ modifica apenas a coluna $\mathbf{b}_k$, como pode ser observado na expressão (6.4), já que as demais colunas da submatriz ativa possuem o q-ésimo elemento nulo. Assim, tem-se que $\mathbf{b}_k = \mathbf{a}_{\mathcal{B}_k}$ na expressão (6.13), para todo k selecionado durante esta etapa.

Para cada pivô b_{qk} selecionado, adiciona-se uma nova matriz de transformação elementar à representação, atualiza-se os conjuntos $\mathcal{Q}$ e $\mathcal{K}$ e define-se π_k. Os contadores nr_i devem ser decrementados, para cada $i \in \mathcal{Q}$ que satisfaça $b_{ik} \neq 0$. O Algoritmo 6 apresenta os passos desta segunda etapa. A atualização dos contadores nr_i (linha 17) pode fazer com que o conjunto $\mathcal{R}_1$, definido na expressão (6.9), seja modificado a cada

pivô selecionado. Assim, uma linha i com $nr_i > 1$ ao início da etapa, será selecionada caso nr_i seja decrementado até 1.

Algoritmo 6: Etapa 2 do procedimento de inversão (FPI).

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$, $\mathcal{K}$ e $\mathcal{R}_1$; vetor de permutações $\boldsymbol{\pi}$.
Saída: Matriz $\mathbf{B}$ após o pivotamento em linhas com um único elemento em $\mathbf{S}$.

```
1  Para cada (q ∈ R_1) faça
2  {
3      /* Obter a posição do único elemento não-nulo */
4      Defina k tal que b_qk ≠ 0, k ∈ K;
5
6      /* Definir um novo pivô */
7      π_k = q;
8      Q = Q \ {q};
9      K = K \ {k};
10
11     /* Definir a matriz de transformação elementar */
12     η = 0;
13     η_q = 1/b_qk;
14     Para cada (b_ik ≠ 0,  i = 1,...,m,  i ≠ q) faça
15     {
16         η_i = −b_ik/b_qk;
17         Se (i ∈ Q) então nr_i = nr_i − 1;
18     }
19     Inclua o par (η, q) na representação da inversa;
20 }
```

Na Figura 6.4, é ilustrada a matriz básica após a realização da etapa 2, considerando-se permutações implícitas de linhas e colunas para facilitar a exposição. Os elementos da submatriz ativa permanecem inalterados após a transformação das colunas pivotadas durante a etapa.

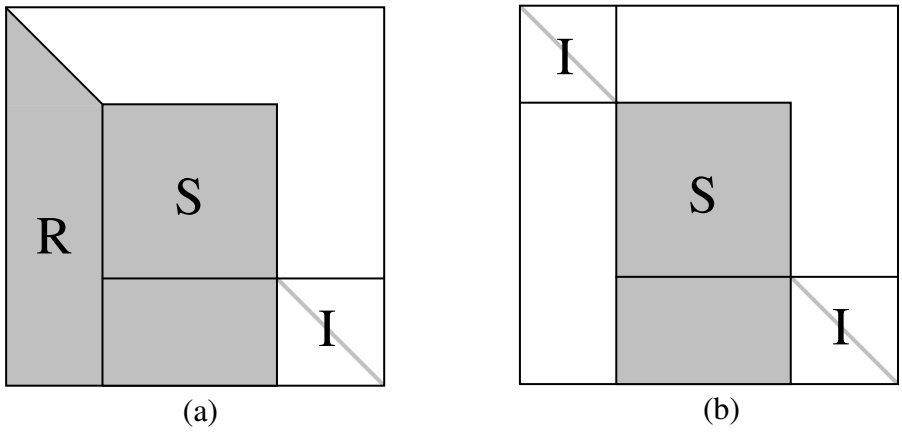

Figura 6.4: (a) Permutação de linhas contendo um único elemento não-nulo na submatriz ativa; (b) Transformação das colunas na partição $\mathbf{R}$ de (a).

Etapa 3: Seleção de pivôs em colunas com um único elemento não-nulo

Colunas com um único elemento não-nulo também devem ser identificadas. A etapa 3 consiste na seleção de pivôs b_{qk} cujas colunas satisfaçam $nc_k = 1$, $k \in \mathcal{K}$. Diferentemente da etapa anterior, realiza-se apenas a sinalização dos pivôs selecionados. As matrizes de transformação elementar correspondentes são criadas apenas no final do procedimento de inversão, com o intuito de evitar a transformação das demais colunas.

Conceitualmente, postergar a criação destas matrizes é equivalente a permutar as colunas com um único elemento não-nulo para as últimas posições da submatriz ativa, de modo que os elementos que devem ser escolhidos como pivôs fiquem sobre a diagonal principal. O resultado desta permutação é apresentado na Figura 6.5. Observe que se os elementos de **S** são selecionados como pivôs antes que os elementos de **C**, então as matrizes de transformação elementar criadas não modificam **C**. Após a transformação das colunas de **S**, as colunas de **C** podem ser transformadas sem influenciar nas demais.

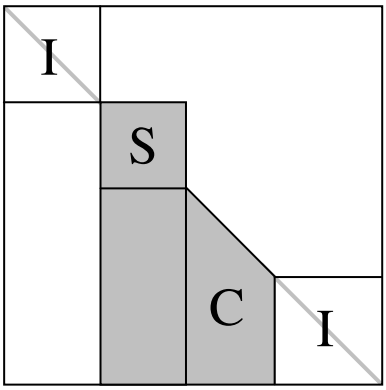

Figura 6.5: Permutação de colunas com um único elemento não nulo.

A implementação desta etapa exige a utilização de um *buffer* para o armazenamento dos índices de colunas que precisam ser transformadas ao final do procedimento. A leitura dos índices para a criação das matrizes de transformação elementar deve ser realizada na ordem inversa do armazenamento. Considerando a discussão apresentada, a etapa 3 é dada no Algoritmo 7.

Etapa 4: Seleção de pivôs no núcleo

Após a realização das etapas anteriores, a submatriz ativa recebe o nome de *núcleo*. Esta denominação é utilizada por ser nesta submatriz que a inversão é realizada de fato. A etapa 4 consiste em transformar as colunas de **S** em colunas triviais. Agora, as matrizes de transformação elementar definidas modificam as demais colunas ainda não pivotadas e, assim, a seleção de pivô deve evitar o preenchimento. Para isto, será usado o critério de seleção descrito no Algoritmo 4, baseando-se na expressão (6.11) para a definição da função $max\_b(l)$. Esta escolha é feita pois a submatriz ativa é armazenada na forma compacta por colunas, e os maiores elementos em módulo podem ser armazenados nas primeiras posições de suas colunas.

Suponha que b_{qk} seja um elemento do núcleo, selecionado como pivô pelo Algoritmo 4. A transformação de $\mathbf{b}_k$ em $\mathbf{e}_q$ é dada pela pré-multiplicação de $\mathbf{B}$ por uma matriz de transformação elementar $\mathbf{E}_{t+1}$, cuja coluna não-trivial é definida na expressão (6.13). Agora, tem-se que as demais colunas ainda não pivotadas também são transformadas

Algoritmo 7: Etapa 3 do procedimento de inversão (FPI).

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$, $\mathcal{K}$ e $\mathcal{C}_1$; contadores nc_j; vetor de permutações $\boldsymbol{\pi}$; *buffer*.
Saída: Matriz $\mathbf{B}$ após a sinalização de colunas com um único elemento em $\mathbf{S}$.

```
1  Para cada (k ∈ C_1) faça
2  {
3     /* Obter a posição do único elemento não-nulo */
4     Defina q tal que b_qk ≠ 0, q ∈ Q;
5
6     /* Definir um novo pivô */
7     π_k = q;
8     Q = Q \ {q};
9     K = K \ {k};
10
11    /* Postergar a criação da matriz de transformação elementar */
12    Armazene o índice k em buffer;
13
14    /* Atualizar os contadores de elementos não-nulos */
15    Para cada (b_qj ≠ 0, j ∈ K) faça nc_j = nc_j − 1;
16 }
```

por $\mathbf{E}_{t+1}$, caso tenham o q-ésimo elemento não-nulo. Desta forma, diferentemente da etapa 2, é bem provável que $\mathbf{b}_k \neq \mathbf{a}_{\mathcal{B}_k}$. Os contadores nr_i e nc_j devem ser atualizados de acordo com os elementos não-nulos da linha q e coluna k. O Algoritmo 8 descreve a quarta etapa.

Finalização

A etapa final consiste em verificar se todas as colunas descritas em $\mathcal{B}$ foram pivotadas. Se $\mathcal{Q} = \varnothing$, então o procedimento de inversão obteve sucesso na definição das matrizes de transformação elementar que representam a inversa da matriz básica. Resta agora criar as matrizes cujos índices estão armazenados em *buffer*. Antes de finalizar o procedimento, deve-se reordenar os índices em $\mathcal{B}$, caso necessário, de acordo com o vetor de permutações $\boldsymbol{\pi}$. A ordem correta de um índice alocado na posição i de $\mathcal{B}$ é dada por π_i.

Quando $\mathcal{Q} \neq \varnothing$ tem-se que a inversão da matriz básica não foi possível e pode-se concluir que o conjunto de índices $\mathcal{B}$ corresponde a uma matriz singular ou quase-singular. Para evitar a falha do procedimento, podem ser atribuídas colunas lógicas ou artificiais $\mathbf{e}_i$, para cada índice $i \in \mathcal{Q}$, e atualizar-se o conjunto $\mathcal{B}$ de acordo com esta redefinição. Entretanto, isto pode fazer com que a nova solução seja infactível ou, então, resultar na piora do valor da função objetivo.

O procedimento de inversão é dado no Algoritmo 9, considerando cada uma das etapas descritas. A realização destas etapas é extremamente importante para eficiência do procedimento. Em geral, grande parte da matriz básica é composta por colunas lógicas e a existência de linhas e colunas com um único elemento não-nulo também é significativa. O processamento das colunas conforme descrito, evita que transformações desnecessárias sejam realizadas, contribuindo com a esparsidade e com a estabilidade numérica da representação final. O critério de seleção de pivôs apresentado no Algoritmo 4 também contribui de maneira significativa com este objetivo.

Algoritmo 8: Etapa 4 do procedimento de inversão (FPI).

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$ e $\mathcal{K}$; vetor de permutações $\boldsymbol{\pi}$; contadores nr_i e nc_j.
Saída: Matriz $\mathbf{B}$ após o pivotamento no núcleo.

```
1  Enquanto (Q ≠ ∅) faça
2  {
3     Selecione o pivô b_qk de acordo com o Algoritmo 4;
4     Se não foi possível selecionar um pivô então PARE!
5
6     /* Definir um novo pivô */
7     π_k = q;
8     Q = Q \ {q};
9     K = K \ {k};
10
11    /* Definir a matriz de transformação elementar */
12    η = 0;
13    η_q = 1/b_qk;
14    Para cada (b_ik ≠ 0,  i = 1,...,m,  i ≠ q) faça
15    {
16       η_i = −b_ik/b_qk;
17       Se (i ∈ Q) então nr_i = nr_i − 1;
18    }
19    Inclua o par (η, q) na representação da inversa;
20
21    /* Atualizar as demais colunas da submatriz ativa */
22    Para cada (b_qj ≠ 0, j ∈ K) faça
23    {
24       nc_j = nc_j − 1;
25       aux = b_qj;
26       Para cada (η_l ≠ 0, l = 1,...,m) faça
27       {
28          Se (b_lj = 0) então
29          {
30             /* Preenchimento: um novo elemento não-nulo será criado */
31             nc_j = nc_j + 1;
32             nr_l = nr_l + 1;
33          }
34          b_lj = b_lj + aux * η_l;
35       }
36       b_qj = aux * η_q;
37    }
38 }
```

Algoritmo 9: Procedimento de inversão da FPI.

Entrada: Conjunto de índices $\mathcal{B}$.
Saída: Representação de $\mathbf{B}^{-1}$ por uma seqüência de matrizes de transformação elementar.

```
1  /* Inicialização */
2  Q = K = {1, ..., m};
3  S = B;
4  π = 0;
5  buffer = ∅;
6
7  /* Selecionar as colunas lógicas */
8  Algoritmo 5;
9
10 /* Definir os contadores de elementos não-nulos */
11 Para cada (i ∈ Q) calcule nr_i e defina o conjunto R_i;
12 Para cada (j ∈ K) calcule nc_j e defina o conjunto C_j;
13
14 /* Realizar as demais etapas */
15 Algoritmo 6;
16 Algoritmo 7;
17 Algoritmo 8;
18
19 /* Verificar se a matriz básica é singular ou quase-singular */
20 Se (Q ≠ ∅) então PARE;
21
22 /* Transformar as colunas em buffer */
23 Enquanto (buffer ≠ ∅) faça
24 {
25    Defina k igual ao último índice em buffer;
26    /* Definir a matriz de transformação elementar */
27    η = 0;
28    η_q = 1/b_qk;
29    Para cada (b_ik ≠ 0,  i = 1, ..., m,  i ≠ q) faça η_i = −b_ik/b_qk;
30    Inclua o par (η, q) na representação da inversa;
31    Remova k do buffer;
32 }
33
34 /* Reordenar os índices em B */
35 Mova o índice B_j para a posição π_j de B, j = 1, ..., m;
```

6.2 Decomposição e atualização LU

Dada uma matriz não-singular $\mathbf{B}$, a decomposição LU consiste em representá-la como um produto de duas matrizes triangulares $\mathbf{L}$ e $\mathbf{U}$. A matriz $\mathbf{U}$ é triangular superior e a matriz $\mathbf{L}$ é triangular inferior. Por convenção, todos os elementos na diagonal principal de $\mathbf{L}$ devem ser iguais a 1.

O procedimento de obtenção destes fatores é equivalente à eliminação de Gauss aplicada na resolução de um sistema linear cuja matriz de coeficientes é dada por $\mathbf{B}$. A matriz $\mathbf{U}$ é exatamente a matriz triangular superior obtida ao final da eliminação, enquanto a matriz $\mathbf{L}$ corresponde à representação matricial das operações elementares realizadas na transformação de $\mathbf{B}$. Entretanto, a decomposição LU apresenta uma vantagem prática sobre a eliminação de Gauss. A resolução de sistemas lineares distintos mas com a mesma matriz de coeficientes é feita em menor complexidade computacional. De fato, basta se decompor a matriz de coeficientes uma única vez e, para cada sistema do tipo $\mathbf{B}\boldsymbol{\alpha} = \boldsymbol{v}$, resolver $\mathbf{LU}\boldsymbol{\alpha} = \boldsymbol{v}$ em dois passos:

$$\begin{array}{ll} (i) & \text{Resolva } \mathbf{L}\boldsymbol{\beta} = \boldsymbol{v}; \\ (ii) & \text{Resolva } \mathbf{U}\boldsymbol{\alpha} = \boldsymbol{\beta}. \end{array} \tag{6.14}$$

A rigor, para que uma matriz não-singular $\mathbf{B}$ arbitrária seja decomposta em fatores triangulares $\mathbf{L}$ e $\mathbf{U}$, pode ser necessária a utilização de matrizes de permutações $\mathbf{P}$ e $\mathbf{Q}$, tais que,

$$\mathbf{PBQ} = \mathbf{LU}.$$

A matriz de permutação $\mathbf{P}$ corresponde à permutação de linhas, enquanto a permutação de colunas é determinada pela matriz $\mathbf{Q}$.

Na prática, ao invés de se permutar as linhas e colunas de $\mathbf{B}$, realiza-se as permutações sobre os fatores triangulares. Assim, as matrizes obtidas podem não estar em uma forma triangular explícita, isto é, pode ser necessária a utilização das matrizes de permutações para colocá-los na forma triangular. Denotando-se por $\tilde{\mathbf{L}}$ e $\tilde{\mathbf{U}}$ as matrizes tais que $\mathbf{B} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}$, então os fatores triangulares podem ser obtidos, fazendo-se:

$$\mathbf{L} = \mathbf{P}\tilde{\mathbf{L}}\mathbf{P}^{-1} \quad \text{e} \quad \mathbf{U} = \mathbf{P}\tilde{\mathbf{U}}\mathbf{Q}. \tag{6.15}$$

Para diferenciar, $\tilde{\mathbf{L}}$ e $\tilde{\mathbf{U}}$ são chamados de fatores triangulares *permutados*. $\tilde{\mathbf{L}}$ é uma matriz triangular inferior permutada e $\tilde{\mathbf{U}}$, uma matriz triangular superior permutada.

No contexto dos métodos tipo simplex, a decomposição LU é utilizada para a representação da matriz básica. Com isto, reduz-se o custo computacional para a resolução dos sistemas lineares de uma dada iteração, já que envolvem a matriz básica. Entretanto, apenas esta redução não justifica o seu uso, pois a matriz básica é modificada a cada troca de base e o custo para o cálculo dos fatores triangulares para cada nova base se torna impraticável. O ponto principal que torna esta técnica viável na prática é a atualização dos fatores a cada troca de base.

A representação da matriz básica pela decomposição LU foi introduzida em métodos tipo simplex por Markowitz. Entretanto, em sua proposta, o autor ainda utilizava a FPI para a atualização da representação a cada troca de base. Foram Bartels e Golub (1969) os primeiros a proporem uma atualização dos fatores triangulares descartando a necessidade da FPI para representar as trocas de base. Apesar de proporcionar maior estabilidade numérica, o método proposto para a atualização era pouco eficiente. Anos

depois, Forrest e Tomlin (1972) apresentaram uma abordagem eficiente para o procedimento de atualização, cujos testes mostraram que, para problemas de grande porte, o preenchimento era menor em comparação à atualização pela FPI.

Atualmente, a técnica de maior eficiência para a representação da matriz básica pela decomposição LU e sua atualização é a proposta por Suhl e Suhl (1990, 1993). Esta técnica, aqui denominada SSLU, resulta em uma maior estabilidade numérica e menor preenchimento com relação à FPI, porém sua implementação é mais trabalhosa. A técnica SSLU é descrita nas seções a seguir.

6.2.1 Decomposição SSLU

A decomposição SSLU consiste em transformar a matriz básica $\mathbf{B}$ em uma matriz triangular superior permutada $\tilde{\mathbf{U}}$, utilizando-se um número $p < m$ de matrizes de transformação elementar $\mathbf{L}_1, \mathbf{L}_2, \ldots, \mathbf{L}_p$. O produto destas p matrizes é igual a uma matriz triangular inferior permutada que corresponde à inversa da matriz $\tilde{\mathbf{L}}$. Assim,

$$\mathbf{L}_p\mathbf{L}_{p-1}\ldots\mathbf{L}_1\mathbf{B} = \tilde{\mathbf{L}}^{-1}\mathbf{B} = \tilde{\mathbf{U}} \quad \Rightarrow \quad \mathbf{B} = \tilde{\mathbf{L}}\tilde{\mathbf{U}}. \tag{6.16}$$

Os fatores triangulares $\mathbf{L}$ e $\mathbf{U}$ podem ser obtidos a partir dos fatores $\tilde{\mathbf{L}}$ e $\tilde{\mathbf{U}}$ como definido na expressão (6.15), utilizando-se as matrizes de permutação $\mathbf{P}$ e $\mathbf{Q}$. Para isto, as matrizes de permutação precisam ser definidas durante o procedimento de decomposição e mantidas após sua finalização. Elas são importantes na resolução dos sistemas lineares envolvendo os fatores triangulares e na atualização da representação a cada troca de base.

Como a posição das colunas básicas é definida por $\mathcal{B}$, as permutações determinadas por $\mathbf{Q}$ podem ser usadas para se reordenar os índices deste conjunto, ao final do procedimento de decomposição. Assim, $\mathbf{Q}$ pode ser descartada e apenas $\mathbf{P}$ precisa ser mantida. De fato, multiplicando-se $\tilde{\mathbf{L}}^{-1}\mathbf{B} = \tilde{\mathbf{U}}$ à direita pela matriz $\mathbf{QP}$, tem-se

$$\begin{aligned} \tilde{\mathbf{L}}^{-1}\mathbf{B}(\mathbf{QP}) &= \tilde{\mathbf{U}}(\mathbf{QP}) \\ &= \mathbf{P}^{-1}\mathbf{U}\mathbf{Q}^{-1}(\mathbf{QP}) \\ &= \mathbf{P}^{-1}\mathbf{UP}. \end{aligned}$$

Redefinindo-se $\mathbf{B} := \mathbf{BQP}$ e $\tilde{\mathbf{U}} := \tilde{\mathbf{U}}\mathbf{QP}$ tem-se novamente $\tilde{\mathbf{L}}^{-1}\mathbf{B} = \tilde{\mathbf{U}}$ mas, agora,

$$\mathbf{U} = \mathbf{P}\tilde{\mathbf{U}}\mathbf{P}^{-1}. \tag{6.17}$$

Assim, ao se utilizar as matrizes $\mathbf{Q}$ e $\mathbf{P}$ para a reordenação das colunas da matriz básica e do fator $\tilde{\mathbf{U}}$, pode-se descartar $\mathbf{Q}$. Após esta reordenação de $\tilde{\mathbf{U}}$, todo elemento pivô passa a ficar sobre sua diagonal principal. Esta característica é bastante favorável nas operações de transformação e na atualização da representação, como apresentado adiante. Diz-se que o fator triangular $\mathbf{U}$ é obtido a partir de $\tilde{\mathbf{U}}$ por meio de permutações simétricas, já que $\mathbf{P}^{-1} = \mathbf{P}^T$.

A seguir, é descrito o procedimento para se realizar a decomposição SSLU de uma matriz básica, partindo-se do conjunto de índices básicos $\mathcal{B}$. Este procedimento é semelhante ao procedimento de inversão definido para a FPI. A notação utilizada é a mesma definida na Seção 6.1.2.

Descrição do procedimento

Em geral, o procedimento de decomposição é realizado em $m-1$ passos caracterizados pela seleção de um elemento pivô a cada passo. O pivô b_{qk} selecionado em um passo t é utilizado para anular os demais elementos de sua coluna que estão na submatriz ativa. Esta operação é chamada de *eliminação*. Conseqüentemente, todos os elementos da submatriz ativa também são transformados. Cada elemento b_{ij}, com $i \in \mathcal{Q} \setminus \{q\}$ e $j \in \mathcal{K} \setminus \{k\}$, é redefinido como:

$$b_{ij} - \frac{b_{ik}}{b_{qk}} b_{qj}. \tag{6.18}$$

Pela expressão, pode-se ver que os elementos da coluna k da submatriz ativa são anulados, exceto o elemento pivô. Esta atualização pode ser representada na forma matricial pela matriz de transformação elementar $\mathbf{L}_t$, cuja coluna não-trivial é dada por:

$$\eta_i^t = \begin{cases} 1, & \text{se } i = q, \\ -b_{ik}/b_{qk}, & \text{se } i \in \mathcal{Q} \setminus \{q\}, \\ 0, & \text{caso contrário,} \end{cases} \tag{6.19}$$

para $i = 1, \ldots, m$. A atualização definida em (6.18) é equivalente a se pré-multiplicar a matriz básica por $\mathbf{L}_t$. Ao final de $m-1$ passos, tem-se que a matriz básica é transformada em uma matriz triangular superior permutada, como indicado em (6.16).

Durante o procedimento de decomposição, a matriz de permutações de linhas $\mathbf{P}$ é representada pelo vetor $\boldsymbol{\rho}$. O vetor $\boldsymbol{\pi}$ representa o produto $\mathbf{QP}$, que corresponde a uma matriz de permutações de colunas, conforme discutido no início da seção. Ao se escolher o pivô b_{qk} no passo t, sinaliza-se que a coluna k foi pivotada na posição q por $\pi_k = q$. A linha q pivotada deve corresponder à linha t dos fatores triangulares e, assim, define-se $\rho_q = t$. Estes vetores são importantes pois permitem que os índices originais sejam utilizados durante o procedimento de decomposição, evitando a tradução de índices. Ao final do procedimento, $\boldsymbol{\pi}$ deve ser utilizado para a reordenação dos índices de $\mathcal{B}$ e apenas $\boldsymbol{\rho}$ precisa ser mantido.

Computacionalmente, a matriz básica deve ser armazenada utilizando-se a representação compacta por linhas (veja o Capítulo 4) antes de se iniciar o procedimento. Como seus elementos podem ser modificados durante a decomposição, a estrutura deve manter espaços livres adicionais, conforme ilustrado na Figura 4.6. Em uma dada iteração do procedimento, os elementos em linhas e colunas pivotadas compõem parte da matriz $\tilde{\mathbf{U}}$, enquanto os demais compõem a submatriz ativa. Ao final do procedimento, os elementos na estrutura representam a matriz $\tilde{\mathbf{U}}$ e, portanto, esta não pode ser descartada. Na Figura 6.6, tem-se a estrutura de dados descrita, considerando que o procedimento de decomposição foi finalizado. Note que as matrizes de transformação elementar são armazenadas na mesma estrutura, iniciando-se da última posição.

Os procedimentos de decomposição e atualização SSLU e as operações de transformação (veja Seção 6.2.3) podem ser bastante eficientes quando a matriz básica (e, conseqüentemente, a matriz $\tilde{\mathbf{U}}$) é também armazenada utilizando-se a representação compacta por colunas, conforme observado por Koberstein (2005). Apesar de parecer redundante, esta abordagem permite que os elementos sejam acessados por linhas ou colunas, evitando a realização de buscas. O autor ainda sugere que apontadores sejam utilizados entre os elementos das duas estruturas, facilitando o acesso aos dados. O uso adicional de memória é justificado pela redução do tempo computacional.

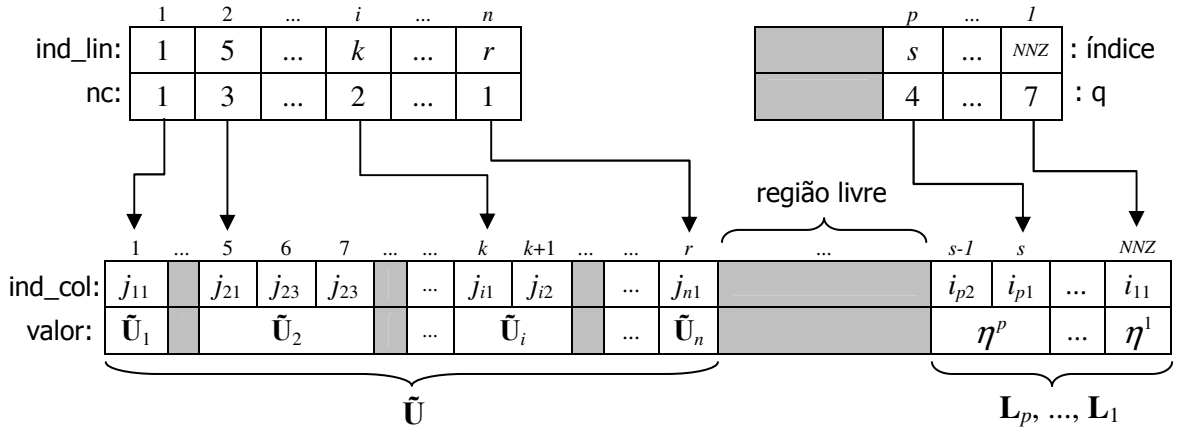

Figura 6.6: Estrutura de dados para a decomposição SSLU.

Para garantir a eficiência do procedimento de decomposição é essencial dividi-lo em etapas, as quais permitem explorar a esparsidade da matriz básica com o intuito de proporcionar o menor preenchimento possível, manter a estabilidade numérica dos cálculos e reduzir o tempo computacional para a execução do procedimento. Estas etapas são descritas na seqüência. Ao invés de um contador de iterações, a descrição é feita baseando-se no contador *nps* que representa o número de pivôs selecionados.

Etapa 1: Seleção de pivôs em colunas lógicas

Assim como no procedimento de inversão da FPI, a primeira etapa da decomposição SSLU consiste em pivotar as colunas lógicas antes das demais colunas básicas. Para isto, basta que estas colunas sejam identificadas e permutadas de acordo com os índices do elemento pivô, sem a necessidade de transformações do tipo (6.18). Os passos desta etapa são apresentados no Algoritmo 10. Antes de iniciá-lo, o valor do contador *nps* deve ser definido igual a 0.

Algoritmo 10: Etapa 1 do procedimento de decomposição SSLU.

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$ e $\mathcal{K}$; vetores $\boldsymbol{\pi}$, $\boldsymbol{\rho}$; contador *nps*.
Saída: Matriz $\mathbf{B}$ com todas as colunas lógicas pivotadas.

```
1  Para cada (coluna lógica b_k) faça
2  {
3      Defina q tal que e_q = b_k;
4
5      /* Definir um novo pivô */
6      nps = nps+1;
7      π_k = q;
8      ρ_q = nps;
9      Q = Q \ {q};
10     K = K \ {k};
11 }
```

Etapa 2: Seleção de pivôs em colunas com um único elemento não-nulo

Quando as colunas com um único elemento não-nulo na submatriz ativa são identificadas e pivotadas antes das demais colunas, evitam-se as atualizações do tipo (6.18). Cada coluna identificada é pivotada na posição correspondente ao seu único elemento não-nulo. As únicas operações realizadas nesta etapa são definir os vetores de permutações $\boldsymbol{\rho}$ e $\boldsymbol{\pi}$ de acordo com os índices dos pivôs selecionados e atualizar os conjuntos $\mathcal{Q}$ e $\mathcal{K}$ que determinam a submatriz ativa. A definição das permutações de linhas em $\boldsymbol{\rho}$ deve considerar que cada pivô é atribuído à primeira posição livre sobre a diagonal de $\mathbf{U}$. Os passos para a realização da etapa 2 são descritos no Algoritmo 11.

Algoritmo 11: Etapa 2 do procedimento de decomposição SSLU.

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$, $\mathcal{K}$ e $\mathcal{C}_1$; vetores $\boldsymbol{\pi}$, $\boldsymbol{\rho}$; contadores nc_j e *nps*.
Saída: Matriz $\mathbf{B}$ após o pivotamento em colunas com um único elemento em $\mathbf{S}$.

```
1  Para cada (k ∈ C_1) faça
2  {
3      /* Obter a posição do único elemento não-nulo */
4      Defina q tal que b_qk ≠ 0, q ∈ Q;
5
6      /* Definir um novo pivô */
7      nps = nps+1;
8      π_k = q;
9      ρ_q = nps;
10     Q = Q \ {q};
11     K = K \ {k};
12
13     /* Atualizar os contadores de elementos não-nulos */
14     Para cada (b_qj ≠ 0, j ∈ K) faça nc_j = nc_j − 1;
15 }
```

Uma ilustração de como deve ficar a matriz básica após a realização da etapa 2 é dada na Figura 6.7, considerando que as permutações de linhas e colunas foram aplicadas para facilitar a exposição. Os elementos em $\mathbf{U}'$ correspondem às posições pivotadas nas etapas 1 e 2. A submatriz ativa $\mathbf{S}$ é reduzida sem que seus elementos sejam atualizados e a decomposição SSLU deve continuar sobre os elementos desta submatriz.

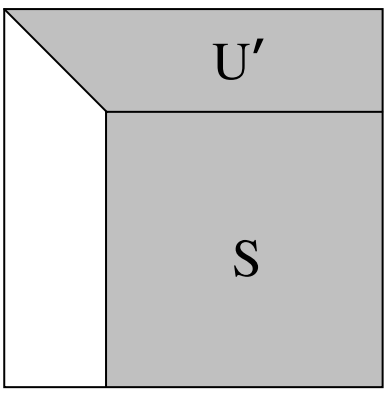

Figura 6.7: Matriz básica permutada após a realização da etapa 2.

Etapa 3: Seleção de pivôs em linhas com um único elemento não-nulo

Após a realização da etapa 2, a seleção de pivôs em linhas com um único elemento não-nulo na submatriz ativa, também evita a atualização das demais colunas desta submatriz. Porém, neste caso, é necessária a criação de uma matriz de transformação elementar que elimine os demais elementos da coluna pivotada que estão na submatriz ativa. Como a linha pivotada tem um único elemento não-nulo, a atualização envolve apenas a coluna correspondente.

A etapa 3 é ilustrada na Figura 6.8, considerando que a matriz básica foi permutada de acordo com os vetores de permutações definidos até o momento. Observe que esta etapa consiste em identificar as colunas que formam um fator triangular inferior inicial $\mathbf{L}'$, cujos elementos são eliminados para a obtenção do fator triangular superior. Os elementos do fator triangular inicial $\mathbf{U}'$ e da submatriz ativa $\mathbf{S}$ não são modificados pela eliminação dos elementos em $\mathbf{L}'$.

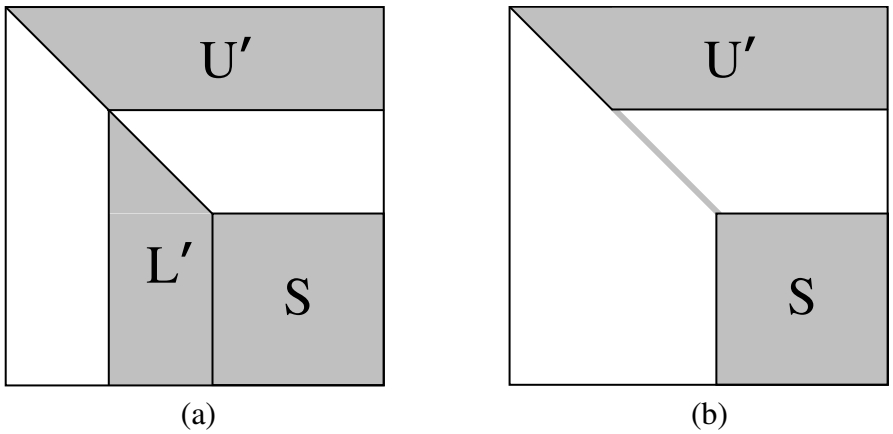

Figura 6.8: (a) Fatores triangulares iniciais definidos apenas com a permutação de linhas e colunas sobre a matriz básica. (b) Matriz básica ao final da etapa 3, após a eliminação dos elementos em $\mathbf{L}'$.

Na prática, a eliminação dos elementos de uma coluna pivotada é feita imediatamente após a sua seleção. A cada pivô b_{qk} selecionado, uma matriz de transformação elementar é criada e adicionada à representação da inversa de $\tilde{\mathbf{L}}$. A matriz criada é responsável por anular os elementos de $\mathbf{b}_k$ que estão na submatriz ativa. Assim, basta que estes elementos sejam removidos da estrutura que representa a submatriz.

As operações realizadas na etapa 3 são descritas no Algoritmo 12. Note que, na linha 18 deste algoritmo, a definição de um elemento como nulo é feita em alto nível. A rigor, como a implementação deve usar uma estrutura de dados que trata a esparsidade da matriz básica, a operação de anular um elemento é equivalente a remover um valor do arranjo que representa os elementos não-nulos. Da mesma forma, a inicialização da coluna não-trivial na linha 14, tem caráter apenas conceitual.

Etapa 4: Seleção de pivôs no núcleo

A submatriz ativa resultante após a realização das etapas anteriores recebe o nome de *núcleo*. A eliminação de elementos de uma coluna do núcleo envolve a atualização das demais colunas desta submatriz e, portanto, tem-se a introdução de erros de arredondamento e a possibilidade de criação de novos elementos não-nulos. Como existe

Algoritmo 12: Etapa 3 do procedimento de decomposição SSLU.

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$, $\mathcal{K}$ e $\mathcal{R}_1$; vetores $\boldsymbol{\pi}$, $\boldsymbol{\rho}$; contadores nr_i e *nps*.
Saída:Matriz $\mathbf{B}$ após o pivotamento em linhas com um único elemento em $\mathbf{S}$.

```
1  Para cada (q ∈ R_1) faça
2  {
3      /* Obter a posição do único elemento não-nulo */
4      Defina k tal que b_qk ≠ 0, k ∈ K;
5
6      /* Definir um novo pivô */
7      nps = nps+1;
8      π_k = q;
9      ρ_q = nps;
10     Q = Q \ {q};
11     K = K \ {k};
12
13     /* Eliminação da coluna */
14     η = 0;
15     Para cada (b_ik ≠ 0, i ∈ Q) faça
16     {
17         η_i = −b_ik/b_qk;
18         b_ik = 0;
19         nr_i = nr_i − 1;
20     }
21     Inclua o par (η, q) na representação de L̃^{-1}.
22 }
```

uma certa liberdade na escolha dos elementos pivôs, deve-se utilizar uma heurística de pivotamento que busque manter a esparsidade e a estabilidade numérica. De acordo com Suhl e Suhl (1990), uma forma eficiente para a seleção de pivôs é dada no Algoritmo 4, baseando-se na expressão (6.12) para a definição da função $max\_b(l)$. Esta escolha é feita pois a matriz básica é armazenada na forma compacta por linhas, e os maiores elementos em módulo podem ser armazenados nas primeiras posições de suas linhas.

Suponha que o pivô b_{qk} tenha sido selecionado utilizando-se o Algoritmo 4. A eliminação dos elementos de $\mathbf{b}_k$ que estão no núcleo é equivalente a pré-multiplicar a matriz básica por uma matriz de transformação elementar, formada a partir de $\mathbf{b}_k$, e adicioná-la à representação de $\tilde{\mathbf{L}}^{-1}$. As demais colunas do núcleo que contenham um elemento não-nulo na linha q também são modificadas de acordo com a expressão (6.18). Estes passos são descritos no Algoritmo 13.

Finalização

Finalizada a etapa 4, é preciso verificar se todas as colunas da matriz básica foram pivotadas. Se o número de pivôs selecionados for igual a m, então o procedimento de decomposição SSLU obteve os fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$ com sucesso. Ao final do procedimento, o fator $\tilde{\mathbf{U}}$ corresponde à matriz básica após as operações de eliminação das etapas 3 e 4. $\tilde{\mathbf{L}}^{-1}$ é representada pelas matrizes de transformação elementar formadas nas operações de eliminação. O procedimento de decomposição SSLU é resumido no Algoritmo 14, considerando-se as etapas descritas.

As permutações definidas pelos vetores $\boldsymbol{\rho}$ e $\boldsymbol{\pi}$ são necessárias na resolução de sistemas lineares utilizando os fatores triangulares permutados e, portanto, precisam ser mantidas após a finalização do procedimento. Conforme mencionado no início desta seção, $\boldsymbol{\pi}$ pode ser descartada se os índices das variáveis básicas forem reordenados em $\mathcal{B}$. Cada índice na posição j de $\mathcal{B}$ deve ser movido para a posição π_j, $j = 1, \ldots, m$. Esta reordenação deve ser repassada para o fator $\tilde{\mathbf{U}}$ e faz com que todos os elementos pivôs fiquem sobre sua diagonal.

6.2.2 Atualização SSLU

Na técnica SSLU, o procedimento de atualização tem como objetivo recuperar a forma triangular superior da matriz $\mathbf{U}$ após uma troca de base, através da atualização dos fatores $\tilde{\mathbf{L}}$ e $\tilde{\mathbf{U}}$. Como a troca de base é executada em grande parte das iterações de um método tipo simplex, o tempo computacional necessário para a atualização deve ser o menor possível. Também, é muito importante que os fatores se mantenham esparsos e contribuam com a estabilidade numérica dos cálculos. A atualização SSLU busca combinar de forma adequada estas três características: baixo tempo computacional, esparsidade e estabilidade numérica.

Considere a matriz básica $\mathbf{B}$ decomposta nos fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$ pelo procedimento de decomposição SSLU. Suponha que, após uma troca de base, a nova matriz básica seja dada por $\bar{\mathbf{B}} = [\mathbf{b}_1, \ldots, \mathbf{b}_{q-1}, \mathbf{a}_k, \mathbf{b}_{q+1}, \ldots, \mathbf{b}_m]$. Multiplicando-a à esquerda por $\tilde{\mathbf{L}}^{-1}$:

$$\begin{aligned} \tilde{\mathbf{L}}^{-1}\bar{\mathbf{B}} &= \left[\tilde{\mathbf{L}}^{-1}\mathbf{b}_1, \ldots, \tilde{\mathbf{L}}^{-1}\mathbf{b}_{q-1}, \tilde{\mathbf{L}}^{-1}\mathbf{a}_k, \tilde{\mathbf{L}}^{-1}\mathbf{b}_{q+1}, \ldots, \tilde{\mathbf{L}}^{-1}\mathbf{b}_m\right] \\ &= \left[\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_{q-1}, \mathbf{g}, \tilde{\mathbf{u}}_{q+1}, \ldots, \tilde{\mathbf{u}}_m\right]. \end{aligned} \tag{6.20}$$

Algoritmo 13: Etapa 4 do procedimento de decomposição SSLU.

Entrada: Matriz $\mathbf{B}$; conjuntos $\mathcal{Q}$ e $\mathcal{K}$; vetores $\boldsymbol{\pi}$, $\boldsymbol{\rho}$; contadores nr_i, nc_j e *nps*.
Saída: Matriz $\mathbf{B}$ após o pivotamento no núcleo.

```
1  Enquanto (nps < m) faça
2  {
3     Selecione o pivô b_qk de acordo com o Algoritmo 4;
4     Se não foi possível selecionar um pivô então PARE!
5
6     /* Definir um novo pivô */
7     nps = nps+1;
8     π_k = q;
9     ρ_q = nps;
10    Q = Q \ {q};
11    K = K \ {k};
12
13    /* Eliminação da coluna */
14    η = 0;
15    Para cada (b_ik ≠ 0, i ∈ Q) faça
16    {
17       /* Definir a coluna não-trivial e atualizar o núcleo */
18       η_i = −b_ik/b_qk;
19       b_ik = 0;
20       nr_i = nr_i − 1;
21
22       /* Transformar os demais elementos da linha i */
23       Para cada (b_qj ≠ 0, j ∈ K) faça
24       {
25          Se (b_ij = 0) então
26          {
27             /* Preenchimento: um novo elemento não-nulo será criado */
28             nc_j = nc_j + 1;
29             nr_i = nr_i + 1;
30          }
31          /* Atualizar o elemento de acordo com a expressão (6.18) */
32          b_ij = b_ij + η_i * b_ij;
33       }
34    }
35    Inclua o par (η, q) na representação de L̃^{-1}.
36 }
```

Algoritmo 14: Procedimento de decomposição SSLU.

Entrada: Conjunto $\mathcal{B}$.
Saída: Fatores triangulares permutados $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$ e vetor de permutações $\boldsymbol{\rho}$.

```
1  /* Inicialização */
2  Q = K = {1, ..., m};
3  S = B;
4  ρ = π = 0;
5  nps = 0;
6
7  /* Selecionar as colunas lógicas */
8  Algoritmo 10;
9
10 /* Definir os contadores de elementos não-nulos */
11 Para cada (i ∈ Q) calcule nr_i e defina o conjunto R_i;
12 Para cada (j ∈ K) calcule nc_j e defina o conjunto C_j;
13
14 /* Realizar as demais etapas */
15 Algoritmo 11;
16 Algoritmo 12;
17 Algoritmo 13;
18
19 /* Verificar se a matriz básica é singular ou quase-singular */
20 Se (nps < m) então PARE;
21
22 /* Realizar as permutações de colunas de acordo com π */
23 Mova o índice B_j para a posição π_j de B, j = 1, ..., m;
24 Mova a coluna ũ_j para a posição π_j de Ũ, j = 1, ..., m;
```

Logo, para que os fatores triangulares permutados representem a nova matriz básica, basta substituir a q-ésima coluna de $\tilde{\mathbf{U}}$ pela coluna $\mathbf{g} = \tilde{\mathbf{L}}^{-1}\mathbf{a}_k$, chamada de *coluna espeto*. Na Figura 6.9, é possível entender o motivo desta denominação, ao se observar uma ilustração do fator triangular $\mathbf{U}$ após a atualização de $\tilde{\mathbf{U}}$. Como indicado na figura, a coluna espeto substitui a t-ésima coluna do fator triangular, sendo $t = \rho_q$. Assim, a atualização de $\tilde{\mathbf{U}}$ pode fazer com que o fator triangular $\mathbf{U}$ correspondente deixe de ser triangular superior e, então, sua forma original deve ser recuperada.

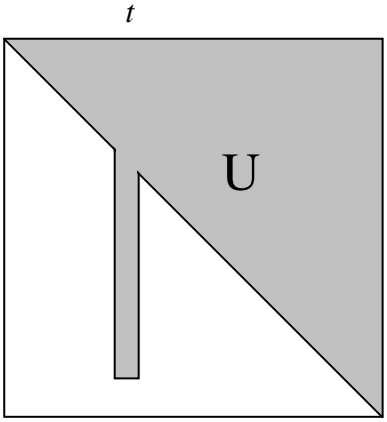

Figura 6.9: Matriz $\mathbf{U}$ após a atualização de $\tilde{\mathbf{U}}$.

Uma primeira idéia para se recuperar a forma triangular de $\mathbf{U}$ seria, como no procedimento de decomposição, utilizar uma matriz de transformação elementar cuja coluna não-trivial fosse definida a partir de $\mathbf{g}$, de acordo com a expressão (6.19). Assim, a matriz utilizada poderia ser incluída na representação de $\tilde{\mathbf{L}}^{-1}$ e a atualização estaria finalizada. Entretanto, esta abordagem resultaria na atualização das demais colunas, podendo prejudicar ainda mais a forma triangular de $\mathbf{U}$.

A transformação de $\mathbf{g}$ deve ser realizada de forma cuidadosa, buscando a menor alteração possível dos fatores triangulares permutados. Neste sentido, a atualização SSLU realiza permutações simétricas de linhas e colunas e utiliza uma matriz de transformação elementar caracterizada por uma *linha* não-trivial, diferentemente do procedimento de decomposição. Como resultado, tem-se uma atualização eficiente cujo objetivo é um bom compromisso entre esparsidade e estabilidade numérica. O procedimento é descrito a seguir.

Descrição do procedimento

Suponha que a q-ésima coluna de $\tilde{\mathbf{U}}$ tenha sido substituída pela coluna espeto $\mathbf{g}$. Para facilitar a exposição, a descrição do procedimento de atualização SSLU é feita baseando-se no fator triangular $\mathbf{U}$. Neste contexto, a coluna espeto ocupa a posição $t = \rho_q$.

Seja l a posição do último elemento não-nulo da coluna $\mathbf{u}_t$, conforme ilustrado na Figura 6.10(a). Suponha que $l > t$ pois, caso contrário, a forma triangular de $\mathbf{U}$ não teria sido prejudicada. Em vez de eliminar elementos não-nulos da coluna espeto, o procedimento de atualização SSLU usa outra estratégia. São eliminados os elementos não-nulos da t-ésima *linha* de $\mathbf{U}$, que estiverem nas posições $t+1, t+2, \ldots, l$ desta linha.

Na Figura 6.10(b), tem-se uma ilustração da matriz **U** após esta eliminação. Conforme indicado nesta figura, o próximo passo é mover a coluna espeto da posição t para a posição l. Para isto, as colunas $t+1, t+2, \ldots, l$ precisam ser deslocadas uma posição para a esquerda, a fim de deixar a l-ésima posição livre. O resultado destas permutações de colunas é apresentado na Figura 6.10(c). A matriz **U** passa a ter a forma de uma matriz *Hessenberg* superior. Basta, agora, deslocar as linhas $t+1, t+2, \ldots, l$ uma posição para cima e mover a linha t para a posição l. As permutações de linhas fazem com que a forma triangular da matriz seja recuperada, conforme ilustrado na Figura 6.10(d).

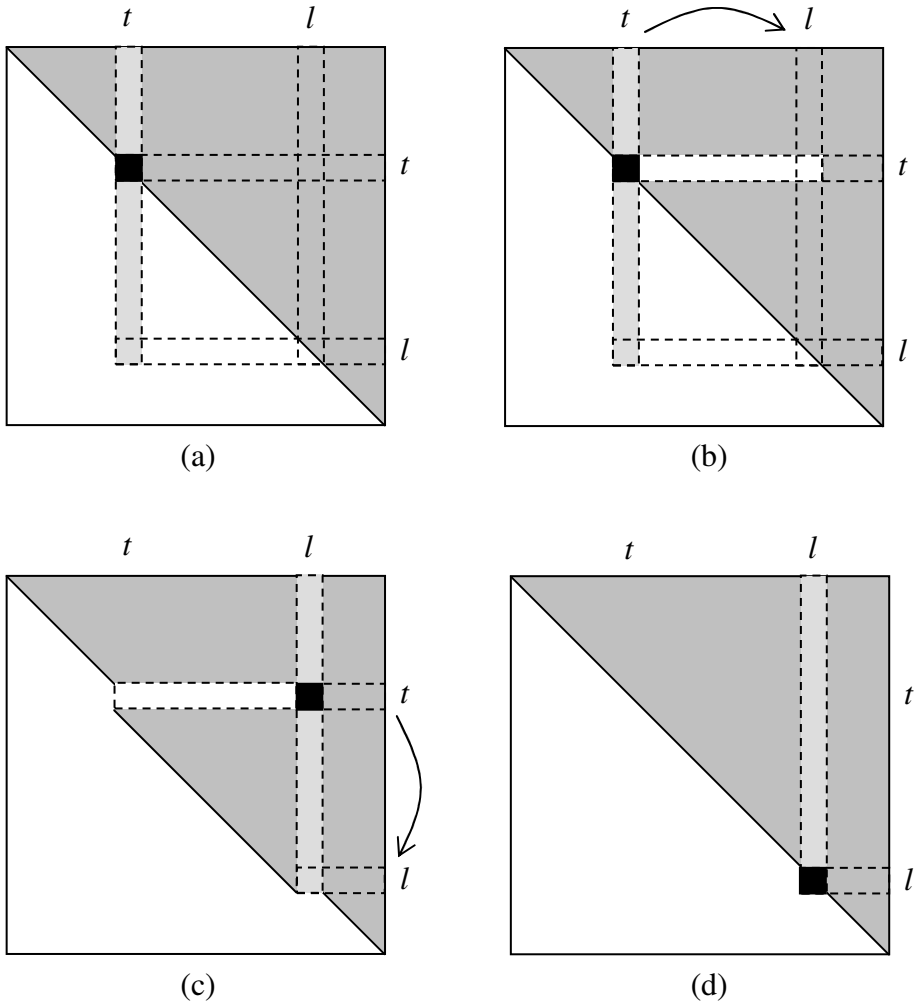

Figura 6.10: Matriz **U** na atualização SSLU; (a) A coluna espeto substitui a coluna t e a posição l é identificada; (b) Os elementos $u_{t,t+1}, \ldots, u_{t,l}$ são eliminados; (c) Permutações de colunas realizadas para que a coluna t ocupe a posição l; (d) Permutações de linhas realizadas para que a linha t ocupe a posição l e a forma triangular é recuperada.

A eliminação de elementos não-nulos da t-ésima linha de **U** é feita de forma iterativa, considerando cada posição $t+1, t+2, \ldots, l$ desta linha. Para um dado índice k, o elemento u_{tk} é eliminado utilizando-se o elemento u_{kk}, que está sobre a diagonal principal de **U**. Esta operação pode modificar os demais elementos da linha $\mathbf{u}_t$ e, até mesmo, fazer com que ocorra preenchimento nas posições $k+1, \ldots, m$ da linha. Entretanto, os

elementos não-nulos criados até a posição l também são eliminados no processo iterativo. Após a eliminação, é criada uma matriz de transformação elementar que corresponde às operações realizadas. Esta matriz é formada por uma *linha* não-trivial.

Computacionalmente, o procedimento de atualização SSLU, que acabou de ser descrito no contexto do fator triangular $\mathbf{U}$, é realizado sobre $\tilde{\mathbf{U}}$. Por esta razão, é necessária a utilização do vetor $\boldsymbol{\rho}$ que representa a matriz de permutações $\mathbf{P}$. Para aumentar a eficiência do procedimento, pode-se manter um outro vetor de permutações que representa $\mathbf{P}^{-1}$, denotado por $\tilde{\boldsymbol{\rho}}$. Este vetor pode ser criado durante o procedimento de decomposição, devido sua simetria com relação ao vetor $\boldsymbol{\rho}$.

Após a atualização de $\tilde{\mathbf{U}}$, a atualização de $\tilde{\mathbf{L}}^{-1}$ consiste em incluir na sua representação a matriz de transformação elementar correspondente à eliminação realizada. Supondo que existam p matrizes de transformação elementar representando $\tilde{\mathbf{L}}^{-1}$, a matriz a ser incluída é dada por $\mathbf{L}_{p+1}$ e se diferencia da matriz identidade por uma única linha não-trivial $\bar{\boldsymbol{\eta}}$, em sua q-ésima posição, dada por:

$$\bar{\eta}_j = \begin{cases} -\tilde{u}_{qj}/\tilde{u}_{jj}, & \text{se } \rho_j \in \{t+1, t+2, \ldots, l\}, \\ 1, & \text{se } \rho_j = t, \\ 0, & \text{caso contrário,} \end{cases} \tag{6.21}$$

para $j = 1, \ldots, m$. Baseando-se na discussão apresentada, o procedimento de atualização SSLU é dado no Algoritmo 15.

6.2.3 Transformações

Quando a técnica SSLU é escolhida para a representação da matriz básica, os cálculos envolvendo esta matriz são realizados utilizando-se os fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$. Como $\tilde{\mathbf{L}}^{-1}$ é representada como um produto de matrizes de transformação elementar estes cálculos são chamados de transformações. Apesar de receberem o mesmo nome no contexto da FPI, as transformações são definidas de forma diferente na técnica SSLU.

Considere que $\tilde{\mathbf{L}}^{-1}$ seja representada por $p+t$ matrizes de transformação elementar, sendo que p foram definidas no procedimento de decomposição e t no procedimento de atualização. As matrizes de transformação elementar criadas no procedimento de decomposição são caracterizadas por colunas não-triviais, enquanto aquelas criadas no procedimento de atualização são formadas por linhas não-triviais. Assim, os cálculos envolvendo estas matrizes são realizados de forma diferente e precisam ser separados.

O cálculo da solução $\boldsymbol{\alpha}$ de um sistema linear do tipo $\mathbf{B}\boldsymbol{\alpha} = \boldsymbol{\upsilon}$ é chamado de transformação progressiva, ou FTRAN (do inglês, *Forward Transformation*), e deve ser realizado em três estágios:

(i) Calcule $\boldsymbol{\upsilon} = \mathbf{L}_p \mathbf{L}_{p-1} \ldots \mathbf{L}_1 \boldsymbol{\upsilon}$;

(ii) Calcule $\boldsymbol{\upsilon} = \mathbf{L}_{p+t} \mathbf{L}_{p+t-1} \ldots \mathbf{L}_{p+1} \boldsymbol{\upsilon}$;

(iii) Resolva $\tilde{\mathbf{U}}\boldsymbol{\alpha} = \boldsymbol{\upsilon}$.

Estes estágios são detalhados no Algoritmo 16. Para a resolução do sistema linear dado no estágio (iii) é necessário que $\tilde{\mathbf{U}}$ seja considerada na forma triangular superior, utilizando-se os vetores $\boldsymbol{\rho}$ e $\tilde{\boldsymbol{\rho}}$. Conceitualmente, o sistema pode ser escrito como $\mathbf{U}\tilde{\boldsymbol{\alpha}} =$

Algoritmo 15: Procedimento de atualização SSLU.

Entrada: Fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$; vetores $\boldsymbol{\rho}$ e $\tilde{\boldsymbol{\rho}}$; coluna espeto $\mathbf{g}$; índice q.
Saída: Fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$ atualizados; vetores $\boldsymbol{\rho}$ e $\tilde{\boldsymbol{\rho}}$ atualizados.

1 /* Substituir a q-ésima coluna de $\tilde{\mathbf{U}}$ por $\mathbf{g}$ */
2 Para cada ($\tilde{u}_{iq} \neq 0$, $i = 1, ..., m$) faça $\tilde{u}_{iq} = 0$;
3 Para cada ($g_i \neq 0$, $i = 1, ..., m$) faça $\tilde{u}_{iq} = g_i$;
4
5 /* Definir os índices t e l */
6 $t = \rho_q$;
7 $l = \max\{\rho_i : \tilde{u}_{iq} \neq 0, i = 1, \ldots, m\}$;
8
9 /* Eliminar os elementos $u_{t,t+1}, \ldots, u_{t,l}$ */
10 $\bar{\boldsymbol{\eta}} = \mathbf{0}$;
11 Para ($k = t + 1$ até l, passo 1) faça
12 {
13 $j = \tilde{\rho}_k$;
14 $\bar{\eta}_j = -\tilde{u}_{qj}/\tilde{u}_{jj}$;
15 $\tilde{u}_{qj} = 0$;
16 Para cada ($\tilde{u}_{jl} \neq 0$, $l = 1, ..., m$) faça $\tilde{u}_{ql} = \tilde{u}_{ql} + \bar{\eta}_j \tilde{u}_{jl}$;
17 }
18 Inclua o par $(\bar{\boldsymbol{\eta}}, q)$ na representação de $\tilde{\mathbf{L}}^{-1}$;
19
20 /* Realizar as permutações de linhas e colunas */
21 Para ($k = t + 1$ até l, passo 1) faça
22 {
23 $j = \tilde{\rho}_k$;
24 $\rho_j = k - 1$;
25 $\tilde{\rho}_{k-1} = j$;
26 }
27 $\tilde{\rho}_l = q$;
28 $\rho_q = l$;

$\tilde{\boldsymbol{v}}$, refletindo-se as permutações em $\boldsymbol{\alpha}$ e $\boldsymbol{v}$. Assim, a solução é dada por

$$\tilde{\alpha}_i = \frac{1}{u_{ii}} \left(\tilde{v}_i - \sum_{j=i+1}^{m} u_{ij} \tilde{\alpha}_j \right), \quad i = m, m-1, \ldots, 1.$$

Pode-se notar nesta expressão que, a partir do momento em que $\tilde{\alpha}_j$ é calculado, seu valor é utilizado na multiplicação dos elementos de apenas uma coluna de $\mathbf{U}$. Desta forma, se $\tilde{\alpha}_j = 0$ então a coluna $\mathbf{u}_j$ não interfere no cálculo da solução e pode ser ignorada.

Em sistemas lineares do tipo $\mathbf{B}^T \boldsymbol{\alpha} = \boldsymbol{v}$, a solução $\boldsymbol{\alpha}$ é obtida pela transformação regressiva, ou BTRAN (do inglês, *Backward Transformation*), dada por:

$$\begin{aligned}
&(i) \quad \text{Resolva } \tilde{\mathbf{U}}^T \boldsymbol{\alpha} = \boldsymbol{v}; \\
&(ii) \quad \text{Calcule } \boldsymbol{\alpha} = \mathbf{L}_{p+1}^T \ldots \mathbf{L}_{p+t-1}^T \mathbf{L}_{p+1}^T \boldsymbol{\alpha}; \\
&(iii) \quad \text{Calcule } \boldsymbol{\alpha} = \mathbf{L}_1^T \ldots \mathbf{L}_{p-1}^T \mathbf{L}_p^T \boldsymbol{\alpha}.
\end{aligned}$$

O detalhamento dos estágios é apresentado no Algoritmo 17. O primeiro estágio consiste em resolver um sistema linear cuja matriz de coeficientes deve ser triangular inferior e, assim, os vetores de permutação $\boldsymbol{\rho}$ e $\tilde{\boldsymbol{\rho}}$ precisam ser usados. Como mencionado na FTRAN, as componentes nulas de $\boldsymbol{\alpha}$ devem ser identificadas. Considerando as permutações, o sistema linear pode ser visto como $\mathbf{U}^T \tilde{\boldsymbol{\alpha}} = \tilde{\boldsymbol{v}}$, cuja solução é dada por

$$\tilde{\alpha}_j = \frac{1}{u_{jj}} \left(\tilde{v}_j - \sum_{i=1}^{j-1} u_{ij} \tilde{\alpha}_i \right), \quad j = 1, 2, \ldots, m.$$

Logo, se $\tilde{\alpha}_i = 0$ então a i-ésima linha de $\mathbf{U}$ pode ser ignorada nos cálculos subseqüentes.

A descrição das transformações finaliza a exposição da técnica SSLU. É recomendado ao leitor, consultar os trabalhos originais sobre a decomposição SSLU (Suhl e Suhl, 1990) e sua atualização (Suhl e Suhl, 1993). Koberstein (2005) também apresenta os fundamentos da técnica SSLU e detalha sua implementação junto ao software MOPS.

Algoritmo 16: FTRAN (SSLU)

Entrada: vetor $\boldsymbol{v}$; fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$; contadores p e t; tolerância δ.
Saída: Solução $\boldsymbol{\alpha}$ do sistema linear $\mathbf{B}\boldsymbol{\alpha} = \boldsymbol{v}$.

```
1  /* Inicializar o vetor α */
2  α = v;
3
4  /* (i) Pré-multiplicar α pelas matrizes L_1, ..., L_p */
5  Para (i = 1 até p, passo 1) faça
6  {
7      /* Obter o par (η^i, q^i) correspondente à matriz de transformação elementar L_i */
8      (η, q) = (η^i, q^i);
9
10     /* Verificar se α_q ≠ 0 */
11     Se (|α_q| > δ) então
12     {
13         aux = α_q;
14         Para cada (η_j ≠ 0, j = 1, ..., m) faça α_j = α_j + aux * η_j;
15         α_q = aux * η_q;
16     }
17 }
18
19 /* (ii) Pré-multiplicar α pelas matrizes L_{p+1}, ..., L_{p+t} */
20 Para (i = p + 1 até p + t, passo 1) faça
21 {
22     /* Obter o par (η̄^i, q^i) correspondente à matriz de transformação elementar L_i */
23     (η̄, q) = (η̄^i, q^i);
24
25     /* Calcular o elemento α_q */
26     soma = 0;
27     Para cada (η_j ≠ 0, j = 1, ..., m) faça soma = soma + α_j * η_j;
28     α_q = soma;
29 }
30
31 /* (iii) Resolver o sistema Ũα = L̃^{-1} v */
32 Para (k = n até 1, passo −1) faça
33 {
34     j = ρ̃_k;
35     α_j = α_j / ũ_jj;
36
37     /* Verificar se α_j ≠ 0 */
38     Se (|α_j| > δ) então
39     {
40         aux = −α_j;
41         Para cada (ũ_ij ≠ 0, i = 1, ..., m, i ≠ j) faça α_i = α_i + aux * ũ_ij;
42     }
43 }
44
```

Algoritmo 17: BTRAN (SSLU)

Entrada: vetor $\boldsymbol{v}$; fatores $\tilde{\mathbf{L}}^{-1}$ e $\tilde{\mathbf{U}}$; contadores p e t; tolerância δ.
Saída: Solução $\boldsymbol{\alpha}$ do sistema linear $\mathbf{B}^T\boldsymbol{\alpha} = \boldsymbol{v}$.

```
1  /* Inicializar o vetor α */
2  α = v;
3
4  /* (i) Resolver o sistema Ũ^T α = v */
5  Para (k = 1 até n, passo 1) faça
6  {
7      i = ρ̃_k;
8      α_i = α_i / ũ_ii;
9
10     /* Verificar se α_i ≠ 0 */
11     Se (|α_i| > δ) então
12     {
13         aux = −α_i;
14         Para cada (ũ_ij ≠ 0, j = 1,...,m, j ≠ i) faça α_j = α_j + aux * ũ_ij;
15     }
16 }
17
18 /* (ii) Pré-multiplicar α pelas matrizes L_{p+1},...,L_{p+t} */
19 Para (i = p + t até p + 1, passo −1) faça
20 {
21     /* Obter o par (η̄^i, q^i) correspondente à matriz de transformação elementar L_i */
22     (η̄, q) = (η̄^i, q^i);
23
24     /* Calcular o elemento α_q */
25     soma = 0;
26     Para cada (η_j ≠ 0, j = 1,...,m) faça soma = soma + α_j * η_j;
27     α_q = soma;
28 }
29
30 /* (iii) Pré-multiplicar α pelas matrizes L_1,...,L_p */
31 Para (i = p até 1, passo −1) faça
32 {
33     /* Obter o par (η^i, q^i) correspondente à matriz de transformação elementar L_i */
34     (η, q) = (η^i, q^i);
35
36     /* Verificar se α_q ≠ 0 */
37     Se (|α_q| > δ) então
38     {
39         aux = α_q;
40         Para cada (η_j ≠ 0, j = 1,...,m) faça α_j = α_j + aux * η_j;
41         α_q = aux * η_q;
42     }
43 }
44
```

Capítulo 7

Resultados e discussões

As técnicas apresentadas neste trabalho buscam contribuir com a eficiência e a estabilidade numérica de métodos tipo simplex. Para verificar o impacto ao utilizá-las na resolução de problemas de grande porte, elas foram implementadas e testadas e os resultados obtidos são apresentados e analisados neste capítulo.

Para a realização de testes, foi implementado um *software* em linguagem C, baseando-se no método primal simplex. As principais características do *software* são as seguintes:

- Estruturas de dados para o armazenamento de vetores e matrizes esparsas;
- Inicialização pela técnica de duas fases, com as bases artificial e lógica-mista;
- Tratamento de degeneração por meio de perturbações;
- Estratégia de *pricing*: regra de Dantzig;
- Teste da razão: simples com tolerância e teste de Harris;
- Utilização de diferentes tolerâncias e métodos de mudança de escala;
- Técnicas FPI e SSLU para a representação da matriz básica.

As tolerâncias utilizadas na implementação são apresentadas na Tabela 7.1. Foram implementados 9 métodos de mudança de escala, dados na Tabela 7.2. Para a utilização do *software*, parâmetros de entrada podem ser definidos de modo a escolher o tipo de inicialização, a técnica de representação da matriz básica, o tipo de teste da razão e se a mudança de escala deve ser utilizada, indicando-se qual o método.

Os parâmetros definidos como padrão são os seguintes:

- Base inicial lógica-mista;
- Teste da razão simples com tolerância;
- Mudança de escala pelo método de média geométrica seguido de equilíbrio;
- Técnica SSLU.

Em casos de estagnação, o teste da razão de Harris é utilizado automaticamente. A entrada de dados deve ser feita no formato MPS, padrão na descrição de problemas de otimização linear disponíveis em bibliotecas públicas.

Para a compilação do *software* foi utilizado o mingw32 3.4.2, versão do compilador gcc para sistemas operacionais Microsoft Windows 32 bits. Todos os testes foram executados em um único computador, com processador Intel Core 2 Duo 2.39 MHz, 2 GB de RAM e sistema operacional Microsoft Windows XP.

Tolerância	Utilização
10^{-6}	Verificar a solução da Fase-I
10^{-8}	Valor absoluto do pivô
10^{-2}	Valor relativo do pivô
10^{-10}	Teste de factibilidade primal
10^{-9}	Teste de factibilidade dual
10^{-10}	Verificar degeneração
10^{-10}	Comparação com zero
10^{-14}	Teste para anular elemento

Tabela 7.1: Tolerâncias utilizadas no *software* implementado.

Método	Símbolo
Sem mudança de escala	S
Equilíbrio	EQ
Média aritmética	MA
Média geométrica	MG
Benichou et al. (1977)	BE
Norma 1	N1
Norma 2	N2
Média aritmética seguido de equilíbrio	MAE
Média geométrica seguido de equilíbrio	MGE
Média geométrica seguido de norma 2	MG2

Tabela 7.2: Métodos de mudança de escala que podem ser utilizados.

7.1 Problemas da biblioteca Netlib

Ao se implementar um *software* numérico, os testes computacionais são essenciais para a sua validação. Um gerador aleatório de problemas de teste pode ser utilizado, cujos dados sigam alguma distribuição de probabilidade. Entretanto, os problemas gerados podem não retratar certas características importantes, que surgem em problemas reais, como instabilidade numérica, degeneração, quase-infactibilidade, entre outras. Além disso, a comparação entre diferentes métodos e implementações pode ser prejudicada, pois os problemas de teste acabam não sendo os mesmos.

Estas desvantagens podem ser resolvidas se bibliotecas públicas de problemas de teste forem utilizadas. A biblioteca Netlib[1] é a coleção de problemas mais utilizada para o teste de métodos tipo simplex. Esta biblioteca é formada por problemas que modelam aplicações reais, conforme descrito por Gay (1997), os quais foram selecionados devido à dificuldade em resolvê-los. Em geral, são caracterizados pela esparsidade de seus dados e por causarem a instabilidade numérica de métodos tipo simplex. A maioria dos problemas foram disponibilizados no final da década de 1980 e eram considerados de grande porte na época. Alguns não estão mais nesta categoria, porém continuam sendo importantes do ponto de vista numérico.

As características da matriz de coeficientes dos problemas da Netlib são dadas nas Tabelas 7.3 e 7.4. Na Tabela 7.5, são destacados os problemas de Kennington, um subdiretório da biblioteca. Na primeira coluna das tabelas é indicado o nome do problema. Para cada problema, tem-se o número de linhas, colunas e elementos não-nulos e a densidade da sua matriz de coeficientes, nas colunas 2, 3, 4 e 5, respectivamente. O valor ótimo exato é apresentado na última coluna, com precisão de 11 dígitos, de acordo com Koch (2004). Para a obtenção destes valores, o autor utilizou o código perPlex (http://www.zib.de/koch/perplex), que realiza os cálculos utilizando uma aritmética racional e, portanto, sem erros de arredondamento.

A densidade da matriz de coeficientes de um problema é calculada pela expressão:

$$\frac{\text{número de elementos não-nulos}}{(\text{número de linhas}) * (\text{número de colunas})},$$

e apresentada na forma de porcentagem nas tabelas. Note como são baixas as densidades, retratando o caráter esparso dos problemas da Netlib. A densidade de grande parte dos problemas não chega a 1% e, em geral, os maiores problemas são os mais esparsos. Dois problemas, FIT1D e FIT2D, têm a densidade alta e podem não ser considerados esparsos. Entretanto, são mantidos na biblioteca por corresponderem à formulação dual dos problemas FIT1P e FIT2P, respectivamente.

Os problemas da Netlib são definidos no formato MPS, o qual permite a definição de restrições com limitantes inferiores e superiores. Apenas as variáveis e colunas estruturais são descritas e os problemas são dados na forma geral (veja o Capítulo 2). Antes de se iniciar a resolução de um problema, as restrições na forma de inequações precisam ser transformadas em equações, para colocá-lo na forma padrão. Para isto, são adicionadas variáveis lógicas a cada restrição, e os limitantes são repassados para estas variáveis, da seguinte maneira:

$$\begin{array}{rc} \text{minimizar} & f(\mathbf{x}_{\mathcal{S}}) = \mathbf{c}_{\mathcal{S}}^T \mathbf{x}_{\mathcal{S}} \\ \text{sujeito a} & \mathbf{L} \leq \mathbf{A}\mathbf{x}_{\mathcal{S}} \leq \mathbf{U} \\ & \mathbf{l} \leq \mathbf{x}_{\mathcal{S}} \leq \mathbf{u} \end{array} \quad \Leftrightarrow \quad \begin{array}{rc} \text{minimizar} & f(\mathbf{x}) = \mathbf{c}_{\mathcal{S}}^T \mathbf{x}_{\mathcal{S}} + \mathbf{0}\mathbf{x}_{\mathcal{L}} \\ \text{sujeito a} & \mathbf{A}\mathbf{x}_{\mathcal{S}} + \mathbf{x}_{\mathcal{L}} = \mathbf{U} \\ & \mathbf{l} \leq \mathbf{x}_{\mathcal{S}} \leq \mathbf{u} \\ & \mathbf{0} \leq \mathbf{x}_{\mathcal{L}} \leq (\mathbf{U} - \mathbf{L}) \end{array}$$

[1] http://www.netlib.org/lp/data

Problema	Linhas	Colunas	Não-nulos	Densidade (%)	Valor ótimo
25FV47	821	1571	10400	0,806	5,5018458883E+03
80BAU3B	2262	9799	21002	0,095	9,8722419241E+05
ADLITTLE	56	97	383	7,051	2,2549496316E+05
AFIRO	27	32	83	9,606	-4,6475314286E+02
AGG	488	163	2410	3,030	-3,5991767287E+07
AGG2	516	302	4284	2,749	-2,0239252356E+07
AGG3	516	302	4300	2,759	1,0312115935E+07
BANDM	305	472	2494	1,732	-1,5862801845E+02
BEACONFD	173	262	3375	7,446	3,3592485807E+04
BLEND	74	83	491	7,994	-3,0812149846E+01
BNL1	643	1175	5121	0,678	1,9776295615E+03
BNL2	2324	3489	13999	0,173	1,8112365404E+03
BOEING1	350	384	3485	2,593	-3,3521356751E+02
BOEING2	166	143	1196	5,038	-3,1501872802E+02
BORE3D	233	315	1429	1,947	1,3730803942E+03
BRANDY	220	249	2148	3,921	1,5185098965E+03
CAPRI	271	353	1767	1,847	2,6900129138E+03
CYCLE	1903	2857	20720	0,381	-5,2263930249E+00
CZPROB	929	3523	10669	0,326	2,1851966989E+06
D2Q06C	2171	5167	32417	0,289	1,2278423615E+05
D6CUBE	415	6184	37704	1,469	3,1549166667E+02
DEGEN2	444	534	3978	1,678	-1,4351780000E+03
DEGEN3	1503	1818	24646	0,902	-9,8729400000E+02
DFL001	6071	12230	35632	0,048	1,1266396047E+07
E226	223	282	2578	4,099	-1,8751929066E+01
ETAMACRO	400	688	2409	0,875	-7,5571523337E+02
FFFFF800	524	854	6227	1,392	5,5567956482E+05
FINNIS	497	614	2310	0,757	1,7279096547E+05
FIT1D	24	1026	13404	54,435	-9,1463780924E+03
FIT1P	627	1677	9868	0,938	9,1463780924E+03
FIT2D	25	10500	129018	49,150	-6,8464293294E+04
FIT2P	3000	13525	50284	0,124	6,8464293232E+04
FORPLAN	161	421	4563	6,732	-6,6421896127E+02
GANGES	1309	1681	6912	0,314	-1,0958573613E+05
GFRD-PNC	616	1092	2377	0,353	6,9022359995E+06
GREENBEA	2392	5405	30877	0,239	-7,2555248130E+07
GREENBEB	2392	5405	30877	0,239	-4,3022602612E+06
GROW15	300	645	5620	2,904	-1,0687094129E+08
GROW22	440	946	8252	1,983	-1,6083433648E+08
GROW7	140	301	2612	6,198	-4,7787811815E+07
ISRAEL	174	142	2269	9,183	-8,9664482186E+05
KB2	43	41	286	16,222	-1,7499001299E+03
LOTFI	153	308	1078	2,288	-2,5264706062E+01
MAROS-R7	3136	9408	144848	0,491	1,4971851665E+06
MAROS	846	1443	9614	0,788	-5,8063743701E+04
MODSZK1	687	1620	3168	0,285	3,2061972906E+02
NESM	662	2923	13288	0,687	1,4076036488E+07
PEROLD	625	1376	6018	0,700	-9,3807552782E+03
PILOT.JA	940	1988	14698	0,787	-6,1131364656E+03
PILOT	1441	3652	43167	0,820	-5,5748972928E+02
PILOT.WE	722	2789	9126	0,453	-2,7201075328E+06
PILOT4	410	1000	5141	1,254	-2,5811392589E+03
PILOT87	2030	4883	73152	0,738	3,0171034733E+02
PILOTNOV	975	2172	13057	0,617	-4,4972761882E+03
RECIPE	91	180	663	4,048	-2,6661600000E+02
SC105	105	103	280	2,589	-5,2202061212E+01
SC205	205	203	551	1,324	-5,2202061212E+01
SC50A	50	48	130	5,417	-6,4575077059E+01
SC50B	50	48	118	4,917	-7,0000000000E+01

Tabela 7.3: Descrição dos problemas da biblioteca Netlib. (Parte 1)

Problema	Linhas	Colunas	Não-nulos	Densidade (%)	Valor ótimo
SCAGR25	471	500	1554	0,660	-1,4753433061E+07
SCAGR7	129	140	420	2,326	-2,3313898243E+06
SCFXM1	330	457	2589	1,717	1,8416759028E+04
SCFXM2	660	914	5183	0,859	3,6660261565E+04
SCFXM3	990	1371	7777	0,573	5,4901254550E+04
SCORPION	388	358	1426	1,027	1,8781248227E+03
SCRS8	490	1169	3182	0,556	9,0429695380E+02
SCSD1	77	760	2388	4,081	8,6666666743E+00
SCSD6	147	1350	4316	2,175	5,0500000077E+01
SCSD8	397	2750	8584	0,786	9,0499999993E+02
SCTAP1	300	480	1692	1,175	1,4122500000E+03
SCTAP2	1090	1880	6714	0,328	1,7248071429E+03
SCTAP3	1480	2480	8874	0,242	1,4240000000E+03
SEBA	515	1028	4352	0,822	1,5711600000E+04
SHARE1B	117	225	1151	4,372	-7,6589318579E+04
SHARE2B	96	79	694	9,151	-4,1573224074E+02
SHELL	536	1775	3556	0,374	1,2088253460E+09
SHIP04L	402	2118	6332	0,744	1,7933245380E+06
SHIP04S	402	1458	4352	0,743	1,7987147004E+06
SHIP08L	778	4283	12802	0,384	1,9090552114E+06
SHIP08S	778	2387	7114	0,383	1,9200982105E+06
SHIP12L	1151	5427	16170	0,259	1,4701879193E+06
SHIP12S	1151	2763	8178	0,257	1,4892361344E+06
SIERRA	1227	2036	7302	0,292	1,5394362184E+07
STAIR	356	467	3856	2,319	-2,5126695119E+02
STANDATA	359	1075	3031	0,785	1,2576995000E+03
STANDMPS	467	1075	3679	0,733	1,4060175000E+03
STOCFOR1	117	111	447	3,442	-4,1131976219E+04
STOCFOR2	2157	2031	8343	0,190	-3,9024408538E+04
TRUSS	1000	8806	27836	0,316	4,5881584719E+05
TUFF	333	587	4520	2,312	2,9214776509E-01
VTP.BASE	198	203	908	2,259	1,2983146246E+05
WOOD1P	244	2594	70215	11,094	1,4429024116E+00
WOODW	1098	8405	37474	0,406	1,3044763331E+00

Tabela 7.4: Descrição dos problemas da biblioteca Netlib. (Parte 2)

Problema	Linhas	Colunas	Não-nulos	Densidade (%)	Valor ótimo
CRE-A	3516	4067	14987	0,105	2,3595407061E+07
CRE-B	9648	72447	256095	0,037	2,3129639886E+07
CRE-C	3068	3678	13244	0,117	2,5275116141E+07
CRE-D	8926	69980	242646	0,039	2,4454969765E+07
KEN-07	2426	3602	8404	0,096	-6,7952044338E+08
KEN-11	14694	21349	49058	0,016	-6,9723822625E+09
KEN-13	28632	42659	97246	0,008	-1,0257394789E+10
KEN-18	105127	154699	358171	0,002	-5,2217025287E+10
OSA-07	1118	23949	143694	0,537	5,3572251730E+05
OSA-14	2337	52460	314760	0,257	1,1064628447E+06
OSA-30	4350	100024	600138	0,138	2,1421398732E+06
OSA-60	10280	232966	1397793	0,058	4,0440725032E+06
PDS-02	2953	7535	16390	0,074	2,8857862010E+10
PDS-06	9881	28655	62524	0,022	2,7761037600E+10
PDS-10	16558	48763	106436	0,013	2,6727094976E+10
PDS-20	33874	105728	230200	0,006	2,3821658640E+10

Tabela 7.5: Descrição dos problemas de Kennington.

7.2 Testes realizados

O *software* implementado foi aplicado na resolução dos problemas da Netlib, utilizando-se os parâmetros padrões descritos. Nas Tabelas 7.6 e 7.7, são apresentados o número de iterações e o tempo total de execução do método primal simplex, considerando Fase-I e Fase-II. A unidade de tempo é milissegundos. Para todos os problemas apresentados, o valor ótimo foi obtido com precisão de 11 casas decimais, exatamente como dado nas Tabelas 7.3, 7.4 e 7.5.

O *sofware* falhou na resolução de três problemas, D6CUBE, DFL001 e KEN-18, devido à instabilidade numérica. Por esta razão, eles não se encontram nas tabelas. Vale ressaltar que estes problemas podem ser resolvidos se outra combinação de parâmetros for utilizada. Por exemplo, a solução ótima do problema D6CUBE é obtida em 11859 ms, 21168 iterações, se o teste da razão de Harris for utilizado e nenhuma mudança de escala for aplicada ao problema.

Definir um conjunto de parâmetros e tolerâncias que resulte na obtenção da solução ótima de todos os problemas é bastante complicado. Mesmo os *softwares* mais eficientes podem falhar na resolução de alguns problemas da Netlib, quando seus parâmetros padrões são utilizados. Por exemplo, em testes realizados por Koch (2004), a versão 8.0 do software CPLEX (ILOG Inc., 2008), considerando seus parâmetros padrões, não foi capaz de resolver os problemas ETAMACRO, D2Q06 e SCSD6. Para que as soluções ótimas corretas destes problemas fossem obtidas, o autor precisou redefinir certas tolerâncias e ajustar os parâmetros de forma particular, como a utilização de uma mudança de escala "agressiva" (expressão usada pelo autor, embora não explicitada) e a desativação do pré-processamento.

Comparação com outro *software*

Para comparar a eficiência do *software* implementado, os problemas da Tabela 7.6, com tempo de execução maior do que zero, foram resolvidos pelo *software* lp_solve, versão 5.5.0.5. Para os testes, seus parâmetros foram definidos iguais aos parâmetros padrões do *software* implementado.

O *software* lp_solve é escrito em linguagem C e baseia-se na implementação dos métodos primal e dual simplex. Um método *branch-and-bound* também está disponível e pode ser utilizado na resolução de problemas de otimização inteira. O desenvolvimento do *software* foi iniciado por Michel Berkelaar e Jeroen Dirks e, atualmente, é mantido por Kjell Eikland e Peter Notebaert. Juergen Ebert contribuiu com o desenvolvimento de uma interface gráfica bastante amigável que permite desde o ajuste de cada parâmetro de resolução até a análise pós-otimização. É um *software* livre, de código aberto, e pode ser obtido em http://lpsolve.sourceforge.net/.

Na Tabela 7.8, é dado o tempo de execução do método primal simplex para cada problema resolvido pelo lp_solve, somando-se Fase-I e Fase-II. Foram omitidos os problemas DEGEN2 e FIT1P, pois não foram resolvidos corretamente pelo lp_solve com os parâmetros definidos. Para facilitar a comparação, as razões entre os tempos de execução são apresentadas na Tabela 7.8, sendo

$$\text{Razão} = \frac{\text{tempo lp\_solve}}{\text{tempo } \textit{software} \text{ implementado}}.$$

Problema	IT	Tempo (ms)	Problema	IT	Tempo (ms)
25FV47	4955	824	PILOT.JA	8437	1659
80BAU3B	11053	3598	PILOT	14535	10565
ADLITTLE	134	0	PILOT.WE	13754	2267
AFIRO	26	0	PILOT4	1493	139
AGG	133	21	PILOT87	24497	42535
AGG2	233	23	PILOTNOV	2504	437
AGG3	242	25	RECIPE	85	0
BANDM	800	48	SC105	107	0
BEACONFD	165	0	SC205	255	0
BLEND	103	0	SC50A	50	0
BNL1	3028	288	SC50B	58	0
BNL2	9486	2621	SCAGR25	718	41
BOEING1	854	46	SCAGR7	163	0
BOEING2	228	0	SCFXM1	422	28
BORE3D	272	20	SCFXM2	967	86
BRANDY	369	30	SCFXM3	1415	171
CAPRI	527	31	SCORPION	384	26
CYCLE	4136	998	SCRS8	1019	76
CZPROB	2031	257	SCSD1	334	25
D2Q06C	29889	14350	SCSD6	726	55
DEGEN2	2124	183	SCSD8	3195	403
DEGEN3	11228	3636	SCTAP1	304	23
E226	615	40	SCTAP2	1129	134
ETAMACRO	965	60	SCTAP3	1476	220
FFFFF800	760	60	SEBA	1487	89
FINNIS	914	54	SHARE1B	335	21
FIT1D	989	99	SHARE2B	141	0
FIT1P	1906	188	SHELL	749	58
FIT2D	11862	11032	SHIP04L	640	57
FIT2P	25125	15562	SHIP04S	526	38
FORPLAN	353	29	SHIP08L	1300	171
GANGES	1690	176	SHIP08S	995	96
GFRD-PNC	1162	74	SHIP12L	1675	290
GREENBEA	13223	5608	SHIP12S	1301	148
GREENBEB	10510	4383	SIERRA	1630	181
GROW15	364	33	STAIR	785	73
GROW22	526	54	STANDATA	283	27
GROW7	168	0	STANDMPS	506	36
ISRAEL	161	0	STOCFOR1	93	0
KB2	74	0	STOCFOR2	2855	532
LOTFI	195	10	TRUSS	13850	5725
MAROS-R7	4339	4384	TUFF	1078	75
MAROS	2197	287	VTP.BASE	383	25
MODSZK1	2254	211	WOOD1P	427	160
NESM	3998	536	WOODW	2295	754
PEROLD	7305	953			

Tabela 7.6: Resultado da resolução dos problemas da Netlib.

Problema	IT	Tempo (ms)	Problema	IT	Tempo (ms)
CRE-A	6203	2007	OSA-14	1775	3571
CRE-B	49887	157671	OSA-30	3538	15549
CRE-C	6569	1905	OSA-60	6789	70016
CRE-D	37414	109298	PDS-02	2991	777
KEN-07	4039	812	PDS-06	46783	58567
KEN-11	30840	39617	PDS-10	201847	551998
KEN-13	174278	647876	PDS-20	901065	8201798
OSA-07	799	703			

Tabela 7.7: Resultado da resolução dos problemas de Kennington.

Problema	Tempo (ms)	Razão	Problema	Tempo (ms)	Razão
25FV47	1583	1,92	PILOT	8442	0,80
80BAU3B	6590	1,83	PILOT.WE	2554	1,13
AGG	10	0,48	PILOT4	210	1,51
AGG2	21	0,91	PILOT87	23203	0,55
AGG3	20	0,80	PILOTNOV	340	0,78
BANDM	40	0,83	SCAGR25	60	1,46
BNL1	310	1,08	SCFXM1	30	1,07
BNL2	2224	0,85	SCFXM2	111	1,29
BOEING1	71	1,54	SCFXM3	230	1,35
BORE3D	10	0,50	SCORPION	20	0,77
BRANDY	30	1,00	SCRS8	81	1,07
CAPRI	30	0,97	SCSD1	20	0,80
CYCLE	220	0,22	SCSD6	90	1,64
CZPROB	731	2,84	SCSD8	382	0,95
D2Q06C	29632	2,06	SCTAP1	30	1,30
DEGEN3	22182	6,10	SCTAP2	200	1,49
E226	30	0,75	SCTAP3	343	1,56
ETAMACRO	60	1,00	SEBA	50	0,56
FFFFF800	90	1,50	SHARE1B	19	0,90
FINNIS	60	1,11	SHELL	100	1,72
FIT1D	110	1,11	SHIP04L	90	1,58
FIT2D	9664	0,88	SHIP04S	61	1,61
FIT2P	81606	5,24	SHIP08L	251	1,47
FORPLAN	20	0,69	SHIP08S	130	1,35
GANGES	240	1,36	SHIP12L	482	1,66
GFRD-PNC	100	1,35	SHIP12S	240	1,62
GREENBEA	8422	1,50	SIERRA	191	1,06
GREENBEB	5948	1,36	STAIR	80	1,10
GROW15	80	2,42	STANDATA	10	0,37
GROW22	170	3,15	STANDMPS	60	1,67
LOTFI	10	1,00	STOCFOR2	791	1,49
MAROS-R7	5088	1,16	TRUSS	6369	1,11
MAROS	43842	*	TUFF	70	0,93
MODSZK1	531	2,52	VTP.BASE	10	0,40
NESM	1132	2,11	WOOD1P	341	2,13
PEROLD	1051	1,10	WOODW	1122	1,49
PILOT.JA	1202	0,72	**Média**: 1,38; **Desvio padrão**: 0,92		

Tabela 7.8: Resultado da resolução dos problemas da Netlib pelo *software* lp_solve.

Observe que o desempenho dos *softwares* é semelhante, já que a maioria das razões está em torno de 1, com média aritmética 1,38 e desvio padrão 0,92. A razão calculada para o problema MAROS é igual 152,76, um valor bastante destoante dos demais e, por isto, foi desconsiderada no cálculo da média.

Métodos de mudança de escala

Com o intuito de verificar como a mudança de escala influencia na resolução de problemas de otimização linear, os problemas das Tabelas 7.6 e 7.7 foram novamente resolvidos pelo *software* implementado, utilizando-se cada um dos métodos de mudança de escala apresentados na Tabela 7.2. Os demais parâmetros foram mantidos conforme a definição padrão.

Os resultados obtidos estão resumidos na Tabela 7.9, de acordo com cada método de mudança de escala. O tempo de execução do método primal simplex para a resolução de cada problema foi calculado e dividido pelo tempo de execução utilizando-se o método

de média geométrica seguido de equilíbrio (MGE), apresentado nas Tabelas 7.6 e 7.7. Para as razões obtidas, calculou-se a média aritmética e o desvio padrão, os quais estão apresentados na Tabela 7.9. O tempo total para a resolução de todos os problemas também é dado. Para alguns métodos, nem todos os problemas foram resolvidos com sucesso.

	Netlib				Kennington			
Método	**Falhas**	**Média**	**Desvio**	**Tempo total**	**Falhas**	**Média**	**Desvio**	**Tempo total**
S	4	2,18	4,70	451476	2	1,02	0,08	10194936
EQ	1	1,20	1,13	158410	0	1,09	0,14	10490575
MA	1	1,08	0,45	184378	0	0,98	0,11	10455020
MG	4	1,73	3,33	216748	0	1,04	0,15	10534575
BE	0	1,15	2,01	140839	0	1,00	0,01	10463080
N1	3	1,25	1,43	159541	0	0,78	0,35	3121495
N2	2	1,15	0,94	137214	0	0,72	0,33	3461634
MAE	4	1,15	1,18	131393	0	1,02	0,11	10441944
MGE	0	1,00	0,00	137708	0	1,00	0,00	10461283
MG2	3	0,96	0,58	112445	0	0,75	0,32	3562840

Tabela 7.9: Resolução dos problemas com diferentes métodos de mudança de escala.

Como pode ser observado na Tabela 7.9, a não utilização da mudança de escala (S) resulta em um tempo de execução médio maior que o dobro do tempo de execução utilizando-se o método de média geométrica seguido de equilíbrio (MGE). Essa medida é verificada, também, ao se comparar o tempo total de execução para as duas técnicas. Além disso, seis problemas não foram resolvidos de forma correta, devido à instabilidade numérica. Este resultado reforça que a mudança de escala é, em geral, necessária para a eficiência e estabilidade de métodos tipo simplex.

Os demais métodos tiveram um desempenho parecido com a utilização do método de média geométrica seguido de equilíbrio, inclusive os métodos de norma 1 (N1) e norma 2 (N2), propostos no Capítulo 5. Algumas combinações mostraram-se mais eficazes que os métodos originais. Por exemplo, o método de média geométrica (MG) tem um desempenho médio 70% inferior se não for seguido do método de equilíbrio, considerando-se os problemas da Netlib. A combinação entre o método de média geométrica e o método de norma 2 (MG2) merece destaque, pois proporcionou um desempenho muito bom tanto para os problemas da Netlib quanto para os de Kennington.

Para auxiliar na compreensão dos resultados obtidos, um segundo teste foi realizado. Para cada problema das Tabelas 7.6 e 7.7, calculou-se os elementos com o menor e o maior valor em módulo, dentre todos os elementos não-nulos da matriz de coeficientes, após a realização da mudança de escala. Também foram calculados a média e o desvio padrão, considerando todos os elementos não-nulos da matriz, em módulo. Os mesmos cálculos foram realizados para a matriz de coeficientes sem a mudança de escala. Nas Tabelas 7.10, 7.11 e 7.12 são dados os valores obtidos sem a mudança de escala (S) e utilizando-se o método de média geométrica seguido de equilíbrio (MGE).

Pelas tabelas, é possível verificar que o intervalo de coeficientes não-nulos dos problemas é bastante amplo, em especial para os problemas PEROLD, PILOT.JA, PILOT.WE, PILOT4 e PILOTNOV. Por esta razão, a mudança de escala produz um bom desempenho na resolução dos problemas. Ao se aplicar a mudança de escala pelo método de média geométrica seguido de equilíbrio, os coeficientes se tornam menores ou iguais a 1, em módulo, reduzindo bastante a diferença entre eles.

	S				MGE			
	Menor	**Maior**	**Média**	**Desvio**	**Menor**	**Maior**	**Média**	**Desvio**
25FV47	0,000200	238,950	5,294	14,359	0,002144	1	0,522	0,337
80BAU3B	0,000220	104,740	1,122	4,570	0,010679	1	0,795	0,289
ADLITTLE	0,001200	64,300	1,955	6,891	0,010989	1	0,540	0,379
AFIRO	0,107000	2,429	1,006	0,522	0,327109	1	0,827	0,231
AGG	0,000020	424,000	2,165	22,255	0,002366	1	0,582	0,358
AGG2	0,000020	424,000	2,229	20,322	0,002366	1	0,534	0,351
AGG3	0,000020	424,000	2,221	19,919	0,002366	1	0,533	0,351
BANDM	0,001000	200,000	6,963	21,491	0,001362	1	0,479	0,370
BEACONFD	0,001200	500,000	5,727	18,775	0,002695	1	0,289	0,319
BLEND	0,003000	66,000	2,555	6,002	0,014489	1	0,577	0,343
BNL1	0,001100	78,000	1,931	5,749	0,022248	1	0,631	0,330
BNL2	0,000600	78,000	1,662	5,191	0,014565	1	0,696	0,315
BOEING1	0,011320	3102,585	56,383	198,147	0,004584	1	0,536	0,352
BOEING2	0,010000	3000,000	17,849	122,489	0,002008	1	0,487	0,360
BORE3D	0,000100	1426,904	8,596	56,693	0,000568	1	0,458	0,406
BRANDY	0,000800	203,700	5,091	16,500	0,002282	1	0,462	0,348
CAPRI	0,000090	217,745	4,574	14,543	0,000148	1	0,561	0,356
CYCLE	0,000010	910,618	18,881	62,212	0,000048	1	0,478	0,360
CZPROB	0,001570	137,000	2,057	12,101	0,062335	1	0,815	0,206
D2Q06C	0,000200	2322,700	11,767	47,298	0,000172	1	0,526	0,354
DEGEN2	1,000000	1,000	1,000	0,000	1,000000	1	1,000	0,000
DEGEN3	1,000000	1,000	1,000	0,000	1,000000	1	1,000	0,000
E226	0,000260	1486,200	14,486	67,406	0,002945	1	0,418	0,343
ETAMACRO	0,019000	2000,000	26,348	182,386	0,020000	1	0,690	0,347
FFFFF800	0,008000	108984,560	87,669	2770,271	0,002934	1	0,510	0,367
FINNIS	0,000461	32,000	1,856	4,115	0,004325	1	0,666	0,362
FIT1D	0,010000	1890,000	46,110	113,585	0,004108	1	0,528	0,294
FIT1P	0,010000	1890,000	38,926	106,440	0,002489	1	0,397	0,370
FIT2D	0,050000	2564,000	17,436	55,017	0,002002	1	0,532	0,329
FIT2P	0,050000	2564,000	13,100	48,042	0,001815	1	0,502	0,397
FORPLAN	0,007390	2800,000	5,171	112,149	0,027827	1	0,541	0,318
GANGES	0,001400	1,000	0,512	0,457	0,059715	1	0,586	0,386
GFRD-PNC	1,000000	1095,200	244,086	388,355	0,033655	1	0,773	0,383
GREENBEA	0,000060	100,000	0,857	1,160	0,000967	1	0,511	0,337
GREENBEB	0,000060	100,000	0,857	1,160	0,000967	1	0,511	0,337
GROW15	0,000006	1,000	0,174	0,346	0,000046	1	0,189	0,366
GROW22	0,000006	1,000	0,175	0,347	0,000046	1	0,190	0,367
GROW7	0,000006	1,000	0,171	0,343	0,000046	1	0,186	0,364
ISRAEL	0,001000	1600,000	124,573	306,181	0,001370	1	0,341	0,326
KB2	0,170000	113,000	40,365	44,081	0,007595	1	0,602	0,368
LOTFI	0,019200	1000,000	24,784	90,551	0,020573	1	0,526	0,422
MAROS-R7	0,001672	1,000	0,110	0,247	0,001672	1	0,111	0,247
MAROS	0,000100	16838,400	265,313	1515,309	0,000369	1	0,526	0,351
MODSZK1	0,000740	1,190	0,848	0,335	0,027203	1	0,749	0,325
NESM	0,001000	33,333	1,867	6,039	0,017710	1	0,651	0,344
PEROLD	0,000053	23614,629	438,308	1982,129	0,000204	1	0,490	0,416
PILOT.JA	0,000002	5851141,000	9523,788	194676,720	0,000056	1	0,409	0,392
PILOT	0,000001	145,600	0,705	3,804	0,000093	1	0,312	0,362
PILOT.WE	0,000143	47950,938	135,388	1263,407	0,000264	1	0,600	0,379
PILOT4	0,000037	27843,988	211,949	1495,500	0,000061	1	0,352	0,407
PILOT87	0,000001	1000,000	1,897	34,632	0,000039	1	0,246	0,327
PILOTNOV	0,000002	5851141,000	10324,338	206523,971	0,000023	1	0,433	0,405
RECIPE	0,120000	145,000	29,329	40,091	0,025263	1	0,707	0,338
SC105	0,100000	2,000	1,096	0,303	0,125000	1	0,897	0,183
SC205	0,100000	2,000	1,100	0,287	0,125000	1	0,903	0,172
SC50A	0,100000	2,000	1,088	0,337	0,125000	1	0,883	0,205
SC50B	0,300000	3,000	1,201	0,710	0,467138	1	0,908	0,136

Tabela 7.10: Coeficientes dos problemas da Netlib sem mudança de escala (S) e utilizando-se o método MGE. (Parte 1)

	S				MGE			
	Menor	**Maior**	**Média**	**Desvio**	**Menor**	**Maior**	**Média**	**Desvio**
SCAGR25	0,200000	9,320	1,028	1,086	0,146490	1	0,711	0,315
SCAGR7	0,200000	9,320	1,023	1,049	0,146490	1	0,721	0,311
SCFXM1	0,000500	130,000	8,813	24,581	0,002432	1	0,490	0,370
SCFXM2	0,000500	130,000	8,805	24,568	0,002432	1	0,489	0,371
SCFXM3	0,000500	130,000	8,803	24,564	0,002432	1	0,489	0,371
SCORPION	0,010000	1,000	0,695	0,370	0,012816	1	0,676	0,347
SCRS8	0,001000	388,765	21,826	54,400	0,023405	1	0,773	0,292
SCSD1	0,242536	1,000	0,750	0,226	0,288675	1	0,818	0,209
SCSD6	0,242536	1,000	0,752	0,227	0,353553	1	0,817	0,200
SCSD8	0,242536	1,000	0,762	0,239	0,500000	1	0,851	0,180
SCTAP1	1,000000	80,000	10,072	13,364	0,117622	1	0,780	0,285
SCTAP2	1,000000	80,000	10,853	13,673	0,117622	1	0,778	0,283
SCTAP3	1,000000	80,000	9,873	12,482	0,117622	1	0,791	0,280
SEBA	1,000000	156,000	3,866	11,045	0,110675	1	0,688	0,246
SHARE1B	0,100000	1322,230	76,445	172,106	0,020989	1	0,578	0,327
SHARE2B	0,010000	103,000	34,416	41,459	0,034213	1	0,626	0,295
SHELL	1,000000	1,000	1,000	0,000	1,000000	1	1,000	0,000
SHIP04L	0,013888	4,706	1,004	0,149	0,687141	1	0,980	0,054
SHIP04S	0,013888	4,706	1,005	0,180	0,687141	1	0,979	0,057
SHIP08L	0,011211	5,000	1,003	0,146	0,583780	1	0,987	0,042
SHIP08S	0,011211	5,000	1,006	0,196	0,583780	1	0,985	0,046
SHIP12L	0,006250	1,600	0,996	0,069	0,558581	1	0,975	0,068
SHIP12S	0,006250	1,600	0,992	0,097	0,558581	1	0,974	0,071
SIERRA	1,000000	100000,000	904,853	9464,656	0,003162	1	0,787	0,406
STAIR	0,000010	9,853	0,435	0,798	0,000059	1	0,265	0,373
STANDATA	1,000000	300,000	7,646	21,473	0,208751	1	0,764	0,266
STANDMPS	1,000000	300,000	6,476	19,654	0,208751	1	0,757	0,265
STOCFOR1	0,062580	336,600	52,442	104,058	0,062580	1	0,679	0,380
STOCFOR2	0,202680	336,600	56,025	106,550	0,139875	1	0,736	0,330
TRUSS	0,447200	1,000	0,767	0,209	0,500000	1	0,880	0,214
TUFF	0,000010	10000,000	32,179	259,834	0,000010	1	0,480	0,352
VTP.BASE	0,133330	4000,000	588,824	1311,430	0,014770	1	0,679	0,371
WOOD1P	0,000030	1000,000	71,720	214,840	0,000000	1	0,082	0,209
WOODW	0,010000	1000,000	359,112	424,866	0,003405	1	0,450	0,378

Tabela 7.11: Coeficientes dos problemas da Netlib sem mudança de escala (S) e utilizando-se o método MGE. (Parte 2)

	S				MGE			
	Menor	**Maior**	**Média**	**Desvio**	**Menor**	**Maior**	**Média**	**Desvio**
CRE-A	0,6000	71,000	7,431	14,897	0,055	1	0,642	0,353
CRE-B	0,6000	71,000	1,534	4,356	0,055	1	0,657	0,362
CRE-C	0,5000	71,000	7,146	14,855	0,049	1	0,633	0,353
CRE-D	0,5000	71,000	1,471	4,156	0,045	1	0,661	0,376
KEN-07	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000
KEN-11	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000
KEN-13	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000
OSA-07	0,2925	13,075	2,841	2,231	0,176	1	0,728	0,250
OSA-14	0,2925	13,075	2,823	2,214	0,176	1	0,721	0,253
OSA-30	0,1572	13,190	2,831	2,232	0,120	1	0,680	0,287
OSA-60	0,3502	13,206	2,848	2,238	0,170	1	0,693	0,254
PDS-02	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000
PDS-06	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000
PDS-10	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000
PDS-20	1,0000	1,000	1,000	0,000	1,000	1	1,000	0,000

Tabela 7.12: Coeficientes dos problemas de Kennington sem mudança de escala (S) e utilizando-se o método MGE.

Nos problemas de Kennington, os coeficientes não-nulos são mais próximos. Um fato interessante é que, para metade dos problemas, a matriz de coeficientes é formada por elementos -1 e $+1$. Estas razões ajudam a compreender porque, neste caso, o desempenho sem a mudança de escala (S) é semelhante àquele utilizando-se o método de média geométrica seguido de equilíbrio (MGE), como mostrado na Tabela 7.9.

Na Tabela 7.13, são apresentados cálculos semelhantes para 50 problemas da Netlib, considerando outros dois métodos de mudança de escala: média geométrica (MG) e média geométrica seguido de norma 2 (MG2). Mais uma vez, os valores calculados ajudam a compreender a Tabela 7.9. Apesar de ter reduzido a dispersão dos coeficientes não-nulos em valor absoluto, o método MG ainda os manteve relativamente distantes. Já o método MG2, resultou em coeficientes mais próximos, característica semelhante ao método de média geométrica seguido de equilíbrio. A média dos coeficientes foi bem pequena para a maioria dos problemas, indicando que os coeficientes ficaram mais próximos de zero.

Analisando-se os resultados apresentados nas tabelas, pode-se verificar que, em geral, quanto mais próximos entre si estão os coeficientes em valor absoluto de um problema, menor o tempo computacional para resolvê-lo. Vale ressaltar que, no caso do método MG2, como os coeficientes se tornaram próximos de zero, as tolerâncias definidas como padrão (veja Tabela 7.1) podem ter se tornado inadequadas, o que justificaria a não resolução de 3 problemas.

Técnicas de representação da matriz básica

Para a comparação entre as técnicas PFI e SSLU, os problemas das Tabelas 7.6 e 7.7 foram resolvidos utilizando-as. Nos testes com a FPI, foram calculados o número de vezes que o procedimento de inversão foi invocado (#I), a média do preenchimento ocorrido nestas inversões (I) e a média do preenchimento ocorrido após 50 atualizações da representação (AF). De forma semelhante, na resolução utilizando a SSLU, foram calculados a quantidade de decomposições realizadas (#D), a média do preenchimento nesta decomposições (D) e a média do preenchimento após 50 atualizações SSLU (AS). Os tempos de execução do método primal simplex, Fase-I e Fase-II, foram calculados em ambas as técnicas. A média das razões entre os tempos de execução para a técnica FPI sobre os tempos para a SSLU é igual a 1,76, com desvio padrão igual a 2,54.

Na Tabela 7.14, são apresentados os resultados obtidos para os 50 problemas com maior razão I/D, considerando apenas a Fase-II. A Fase-I possui um grande número de variáveis artificiais na base, o que poderia prejudicar a análise dos resultados. Observe como o preenchimento utilizando-se a FPI é grande quando comparado ao preenchimento utilizando-se a SSLU. Para o problema FIT2P, a representação da inversa da matriz básica tem, em média, cerca de 5000 elementos não-nulos a mais que a própria matriz, após o procedimento de inversão. Este preenchimento é 858,21 vezes maior que o preenchimento médio obtido no procedimento de decomposição SSLU. Além destes elementos, são criados, em média, cerca de 138130 elementos não-nulos nas 50 atualizações da FPI após a inversão, um preenchimento igual a quase três vezes o número de elementos não-nulos na matriz de coeficientes do problema.

Devido ao alto preenchimento, a resolução utilizando-se a FPI exige maior quantidade de memória, sendo inviável na resolução de problemas de grande porte. De fato, nos testes realizados com esta técnica, o *software* implementado não foi capaz de resolver

	MG				MG2			
	Menor	**Maior**	**Média**	**Desvio**	**Menor**	**Maior**	**Média**	**Desvio**
25FV47	0,028818	34,701	1,638	1,770	0,000104	1,000	0,301	0,246
80BAU3B	0,057857	17,284	1,188	0,943	0,000238	1,000	0,624	0,266
ADLITTLE	0,089973	11,114	1,517	1,702	0,000747	1,000	0,410	0,292
AFIRO	0,571934	1,748	1,129	0,347	0,072871	1,000	0,584	0,212
AGG	0,039687	25,197	1,759	2,234	0,000116	1,000	0,210	0,154
AGG2	0,039687	25,197	1,691	2,354	0,000294	1,000	0,203	0,171
AGG3	0,039687	25,197	1,685	2,337	0,000294	1,000	0,203	0,171
BANDM	0,028813	34,707	1,996	3,189	0,00013	1,000	0,307	0,308
BEACONFD	0,040897	24,451	2,104	3,421	0,00037	1,000	0,139	0,242
BLEND	0,08405	11,898	1,634	2,225	0,000802	1,000	0,326	0,251
BNL1	0,108284	9,235	1,258	1,231	0,001338	1,000	0,415	0,239
BNL2	0,093064	10,745	1,228	1,208	0,000523	1,000	0,440	0,236
BOEING1	0,067707	14,770	1,752	2,306	0,000019	1,000	0,244	0,226
BOEING2	0,033387	29,951	2,465	3,353	0,000189	1,000	0,254	0,235
BORE3D	0,021068	47,465	2,313	5,258	0	1,000	0,340	0,324
BRANDY	0,029442	33,965	2,036	2,657	0,000029	1,000	0,226	0,255
CAPRI	0,010014	99,860	1,789	6,162	0	1,000	0,347	0,282
CYCLE	0,006838	146,249	2,293	7,242	0,000001	1,000	0,289	0,233
CZPROB	0,134982	7,408	1,076	0,416	0,026249	1,000	0,550	0,166
D2Q06C	0,012028	83,139	1,739	2,900	0,000004	1,000	0,297	0,267
DEGEN2	1	1,000	1,000	0,000	0,213201	1,000	0,354	0,094
DEGEN3	1	1,000	1,000	0,000	0,142857	1,000	0,250	0,106
E226	0,046985	21,283	1,707	2,302	0,000053	1,000	0,231	0,236
ETAMACRO	0,141421	7,071	1,512	1,309	0,000398	1,000	0,427	0,321
FFFFF800	0,036133	27,676	1,745	2,423	0,000009	1,000	0,276	0,247
FINNIS	0,060461	16,540	1,902	2,712	0,000687	1,000	0,432	0,281
FIT1D	0,048726	20,523	1,574	1,533	0,00051	0,675	0,209	0,181
FIT1P	0,017837	56,062	1,706	2,796	0,000089	1,000	0,197	0,362
FIT2D	0,029719	33,649	1,626	1,427	0,00026	0,655	0,218	0,184
FIT2P	0,027581	36,257	1,194	1,384	0,000048	1,000	0,279	0,437
FORPLAN	0,122276	8,178	1,337	1,173	0,003127	1,000	0,226	0,203
GANGES	0,173373	5,768	0,954	1,001	0,006452	1,000	0,378	0,316
GFRD-PNC	0,173831	5,753	2,262	2,215	0,000913	1,000	0,599	0,318
GREENBEA	0,021755	45,966	1,428	2,031	0,00001	1,000	0,344	0,239
GREENBEB	0,021755	45,966	1,428	2,031	0,00001	1,000	0,344	0,239
GROW15	0,005517	181,251	6,461	22,164	0,000008	1,000	0,157	0,300
GROW22	0,005517	181,251	6,455	22,152	0,000008	1,000	0,157	0,300
GROW7	0,005517	181,251	6,483	22,209	0,000008	1,000	0,156	0,302
ISRAEL	0,02489	40,176	2,278	3,218	0,000057	1,000	0,147	0,203
KB2	0,077873	12,841	2,353	2,804	0,002809	1,000	0,310	0,217
LOTFI	0,079736	12,541	1,942	2,282	0,000706	1,000	0,375	0,381
MAROS-R7	0,04089	24,456	1,621	3,749	0,001428	1,000	0,100	0,234
MAROS	0,015258	65,539	1,619	2,736	0,00002	1,000	0,308	0,236
MODSZK1	0,119268	8,384	1,153	0,930	0,00036	1,000	0,641	0,317
NESM	0,074008	13,512	1,306	1,894	0,00003	1,000	0,258	0,392
PEROLD	0,005172	193,365	3,862	11,912	0,00001	1,000	0,356	0,320
PILOT.JA	0,003427	291,763	3,145	8,767	0,000001	1,000	0,243	0,276
PILOT	0,00386	259,067	3,147	8,935	0	1,000	0,180	0,229
PILOT.WE	0,005175	193,233	2,414	8,992	0,00001	1,000	0,441	0,333
PILOT4	0,002089	478,724	4,707	20,663	0,000002	1,000	0,275	0,345
PILOT87	0,004657	214,732	2,504	6,073	0,000003	1,000	0,138	0,219
PILOTNOV	0,003427	291,763	3,908	12,566	0,000001	1,000	0,278	0,299
RECIPE	0,115751	8,639	1,453	1,750	0,005864	1,000	0,439	0,281
SC105	0,353553	2,828	1,022	0,227	0,083045	0,894	0,591	0,136
SC205	0,353553	2,828	1,017	0,191	0,083045	1,000	0,593	0,130

Tabela 7.13: Coeficientes de 50 problemas da Netlib utilizando-se mudança de escala pelos métodos MG e MG2.

	FPI			SSLU			Razões	
Problema	**#I**	**I**	**AF**	**#D**	**D**	**AS**	**I/D**	**AF/AS**
FIT2P	484	5081,12	138130,27	403	5,92	1052,54	858,21	131,24
FIT1P	25	493,68	29796,96	25	0,92	1016,32	536,61	29,32
SIERRA	14	122,79	1041,29	11	0,82	261,55	150,07	3,98
AGG2	3	102,33	3018,33	3	1,33	800,67	76,75	3,77
BOEING2	1	147,00	4095,00	2	3,00	349,50	49,00	11,72
AGG	1	470,00	6728,00	1	15,00	475,00	31,33	14,16
CAPRI	1	2160,00	8174,00	1	83,00	950,00	26,02	8,60
STOCFOR2	20	7731,00	27995,90	20	297,90	842,05	25,95	33,25
PILOTNOV	16	33593,19	33026,31	15	1381,73	2070,93	24,31	15,95
WOODW	23	3767,91	5655,65	23	159,43	1866,91	23,63	3,03
STANDATA	1	78,00	1199,00	1	4,00	330,00	19,50	3,63
FFFFF800	3	442,00	2736,33	2	23,50	661,50	18,81	4,14
PILOT.JA	105	32081,26	31884,06	95	1930,46	1566,86	16,62	20,35
D2Q06C	483	39148,59	51624,25	537	2374,74	1686,96	16,49	30,60
SCFXM2	4	1969,75	4788,00	5	119,60	714,00	16,47	6,71
BNL2	96	7380,78	27104,96	86	500,59	1045,03	14,74	25,94
MAROS	25	4783,40	14782,32	21	342,24	1041,19	13,98	14,20
SCFXM3	7	2483,00	4560,43	7	188,00	716,71	13,21	6,36
PILOT4	16	8213,69	14503,75	15	637,13	2036,00	12,89	7,12
NESM	41	2867,24	3997,37	38	224,66	770,29	12,76	5,19
SCAGR25	3	877,00	5415,00	3	71,67	1050,33	12,24	5,16
ISRAEL	3	250,00	6995,00	3	20,67	1282,33	12,10	5,45
CYCLE	12	5940,58	13003,50	15	531,33	1281,93	11,18	10,14
SCTAP1	2	108,00	1081,00	2	10,00	458,00	10,80	2,36
BNL1	5	1985,80	9343,00	9	185,11	785,33	10,73	11,90
SCFXM1	2	816,50	3051,00	2	78,00	549,00	10,47	5,56
25FV47	69	10016,54	24285,81	62	1055,15	1662,35	9,49	14,61
GREENBEA	187	9910,65	27792,97	155	1047,37	1513,43	9,46	18,36
TRUSS	175	20758,49	27588,66	174	2254,99	1815,85	9,21	15,19
E226	9	825,44	4315,67	8	90,50	758,50	9,12	5,69
MODSZK1	20	3324,20	20399,30	20	368,70	1538,40	9,02	13,26
PILOT	109	166365,85	55021,59	136	19030,10	3884,54	8,74	14,16
GREENBEB	104	12902,43	26722,49	115	1480,07	1414,94	8,72	18,89
GANGES	2	727,00	9404,50	2	83,50	882,50	8,71	10,66
ETAMACRO	7	906,29	5390,29	7	104,86	1002,29	8,64	5,38
TUFF	3	694,00	5313,33	4	81,00	675,75	8,57	7,86
SCRS8	7	1587,29	4820,00	8	187,38	987,13	8,47	4,88
SCORPION	1	946,00	1886,00	1	115,00	1174,00	8,23	1,61
BOEING1	10	403,40	2313,30	9	49,44	525,44	8,16	4,40
FINNIS	9	265,44	3130,89	7	32,57	756,57	8,15	4,14
BANDM	2	2916,50	10320,50	2	358,00	1255,00	8,15	8,22
PEROLD	81	13549,83	20437,25	68	1675,97	1528,79	8,08	13,37
LOTFI	1	32,00	1322,00	1	4,00	324,00	8,00	4,08
PILOT87	335	353431,64	83565,10	335	46332,17	5892,15	7,63	14,18
PILOT.WE	238	13470,85	27918,55	214	1771,51	1558,23	7,60	17,92
ADLITTLE	2	95,00	1033,00	2	13,00	562,50	7,31	1,84
GROW15	1	8794,00	2551,00	1	1269,00	671,00	6,93	3,80
80BAU3B	112	457,89	3610,39	108	66,93	689,28	6,84	5,24
BRANDY	2	1674,00	7014,00	3	250,67	1078,67	6,68	6,50
STAIR	6	13002,17	19096,00	6	1981,17	2963,67	6,56	6,44
DEGEN3	106	15756,12	20038,77	118	2407,88	2048,41	6,54	9,78
GROW22	1	9920,00	3726,00	1	1523,00	715,00	6,51	5,21
SHARE1B	2	480,50	4158,50	2	93,00	696,00	5,17	5,97
FORPLAN	1	1497,00	4989,00	1	303,00	910,00	4,94	5,48
SCSD8	47	2747,45	7611,09	55	557,36	987,62	4,93	7,71

Tabela 7.14: Preenchimento na representação da matriz básica.

corretamente 5 problemas de Kennington, sendo eles KEN-11, KEN-13, KEN-18, PDS-10 e PDS-20, devido a erros numéricos ou por exceder o espaço de memória alocado durante o procedimento de inversão.

Um último teste foi realizado, com o intuito de verificar como a mudança de escala interfere no preenchimento dado na decomposição e atualização SSLU. Os problemas da Netlib foram resolvidos de duas formas: sem a mudança de escala (S) e, em seguida, utilizando-se o método de média geométrica seguido de norma 2 (MG2). De forma semelhante à comparação do preenchimento entre FPI e SSLU (Tabela 7.14), calculou-se o número de elementos não-nulos a mais do que na matriz básica, após a decomposição (D) e após 50 atualizações da representação (A).

Na Tabela 7.15, são apresentados os resultados obtidos para os 50 problemas com as maiores razões dadas pelo preenchimento após a decomposição sem mudança de escala sobre aquele considerando o método MG2. De acordo com os dados, a mudança de escala contribui com a redução do preenchimento e, conseqüentemente, a resolução dos problemas é realizada em menor tempo computacional, como mostrado na Tabela 7.9. De fato, devido à melhor característica numérica do problema após a mudança, os critérios usados na seleção de elementos pivôs, durante o procedimento de decomposição, conseguem garantir maior esparsidade para a representação.

	SSLU + S			SSLU + MG2			Razões	
Problema	**#D**	**D**	**A**	**#D**	**D**	**A**	**D(S/MG2)**	**A(S/MG2)**
FIT1P	31	35,55	845,84	24	0,33	1027,42	106,65	0,82
FIT1D	50	33,74	208,04	30	0,90	142,50	37,49	1,46
FIT2P	514	36,68	1030,13	431	2,32	952,11	15,81	1,08
DEGEN3	118	2407,88	2048,41	2000	194,92	903,01	12,35	2,27
BOEING2	2	17,00	495,00	2	1,50	275,00	11,33	1,80
SCTAP3	13	31,54	407,92	10	3,00	205,80	10,51	1,98
SCTAP1	1	50,00	553,00	2	5,50	495,00	9,09	1,12
FIT2D	546	67,99	235,18	455	8,32	145,80	8,17	1,61
CYCLE	179	1661,63	1374,51	14	206,79	1166,71	8,04	1,18
SCTAP2	12	2,50	353,17	6	0,33	252,00	7,50	1,40
ISRAEL	6	40,17	1055,83	5	9,20	868,80	4,37	1,22
DEGEN2	12	575,17	1362,33	220	173,73	949,70	3,31	1,43
LOTFI	1	18,00	618,00	1	6,00	364,00	3,00	1,70
TUFF	8	144,50	770,38	1	71,00	1086,00	2,04	0,71
25FV47	148	1622,91	1458,18	52	833,27	1484,15	1,95	0,98
ADLITTLE	1	19,00	966,00	2	10,00	458,50	1,90	2,11
KB2	1	69,00	915,00	1	41,00	566,00	1,68	1,62
SCFXM1	4	90,00	601,00	3	54,33	631,33	1,66	0,95
SCFXM3	12	391,58	782,50	9	241,22	676,89	1,62	1,16
SCSD6	7	89,71	683,86	8	59,75	704,25	1,50	0,97
GREENBEA	250	1283,14	1797,24	111	857,23	1361,68	1,50	1,32
FFFFF800	3	18,33	282,00	3	12,33	187,67	1,49	1,50
SCORPION	1	101,00	914,00	1	68,00	1598,00	1,49	0,57
80BAU3B	102	69,09	540,31	86	47,23	530,72	1,46	1,02
CAPRI	2	107,50	756,50	2	75,00	358,50	1,43	2,11
BRANDY	1	394,00	1253,00	2	276,50	1367,00	1,42	0,92
SCFXM2	7	226,43	756,86	6	165,83	733,50	1,37	1,03
SCSD8	19	397,84	1001,21	18	292,72	765,50	1,36	1,31
SHARE1B	2	115,50	835,00	2	85,50	515,00	1,35	1,62
FINNIS	10	16,70	419,70	8	12,38	410,88	1,35	1,02
E226	11	105,27	703,27	11	78,82	713,00	1,34	0,99
BOEING1	16	68,75	457,88	11	52,09	559,00	1,32	0,82
BNL1	13	216,62	759,69	12	174,67	696,75	1,24	1,09
SCAGR25	4	83,50	1362,00	3	68,00	1102,33	1,23	1,24
CZPROB	17	5,18	1059,12	14	4,29	2097,07	1,21	0,51
TRUSS	246	2213,59	1938,39	221	1848,49	1810,42	1,20	1,07
GREENBEB	174	1527,98	1576,51	73	1291,10	1559,25	1,18	1,01
BNL2	109	472,27	953,41	87	415,37	1001,01	1,14	0,95
ETAMACRO	7	72,57	805,43	7	64,57	591,14	1,12	1,36
STAIR	6	1997,17	3112,00	6	1874,33	2973,17	1,07	1,05
PILOT	356	16617,47	3815,71	225	15780,47	3376,21	1,05	1,13
PILOT87	843	42023,21	5141,97	470	40151,89	5538,43	1,05	0,93
GROW22	5	1708,40	2368,40	2	1647,50	781,00	1,04	3,03
PEROLD	132	1628,08	1374,22	105	1593,50	1427,79	1,02	0,96
AGG	1	15,00	426,00	1	15,00	597,00	1,00	0,71
SHIP04S	3	1,33	232,00	3	1,33	158,00	1,00	1,47
BANDM	5	279,20	1106,60	3	281,00	1331,33	0,99	0,83
SC205	2	263,50	2195,00	2	276,50	2021,00	0,95	1,09
PILOT.WE	230	1572,22	1427,99	140	1671,20	1313,84	0,94	1,09
SCRS8	8	141,25	1001,88	10	150,30	895,90	0,94	1,12
SIERRA	11	6,64	245,55	8	7,13	340,88	0,93	0,72
NESM	99	191,94	676,26	76	207,91	737,76	0,92	0,92
MODSZK1	28	187,29	790,79	28	207,75	786,64	0,90	1,01
GROW15	3	1003,33	1115,00	2	1206,50	494,00	0,83	2,26
GANGES	4	36,25	501,25	3	54,33	822,67	0,67	0,61

Tabela 7.15: Influência da mudança de escala no preenchimento (SSLU).

Capítulo 8

Conclusões e trabalhos futuros

A implementação de métodos tipo simplex para a resolução de problemas reais de grande porte é bastante complexa e envolve os conceitos de diversas áreas, entre elas otimização linear, análise numérica, estruturas de dados e programação de computadores. As técnicas disponíveis na literatura são essenciais para a implementação eficiente e estável destes métodos.

As estruturas de dados utilizadas devem considerar a esparsidade dos dados do problema, buscando a eficiência do armazenamento e da utilização desta informação. O núcleo numérico deve possuir técnicas para a representação da matriz básica e para manter a estabilidade numérica do método, utilizando-se diferentes tolerâncias, estabilização do teste da razão e mudança de escala.

De acordo com os testes realizados, a técnica SSLU mostrou-se bastante eficiente quando comparada à FPI. Em geral, o preenchimento ocorrido no procedimento de decomposição SSLU e durante as atualizações da representação é menor que na FPI e, como conseqüência, o tempo computacional para a resolução dos problemas é reduzido. Além disso, a estabilidade numérica proporcionada pela SSLU é maior, diminuindo o número de falhas na resolução de problemas. Apesar de ser trabalhosa a implementação da técnica SSLU, os resultados obtidos são compensadores.

A mudança de escala também pode ser considerada uma técnica bastante eficaz. Ao melhorar as propriedades numéricas do problema, o preenchimento na representação da matriz básica foi reduzido e o tempo de resolução decaiu. Entretanto, deve-se escolher um método de mudança de escala adequado, como o de média geométrica seguido de equilíbrio ou seguido de norma 2. Estes métodos têm como característica reduzir a diferença entre o valor absoluto dos elementos não-nulos da matriz de coeficientes de um problema, tornando-os mais próximos e dentro do intervalo $[0, 1]$.

A eficiência de métodos tipo simplex também depende de outras técnicas, conforme descrito no Capítulo 3. As técnicas do núcleo numérico precisam ser combinadas com aquelas do núcleo conceitual, de modo a se reduzir ainda mais o número de iterações e o tempo computacional na resolução de problemas. Assim, como proposta para trabalhos futuros tem-se o enfoque nas técnicas do núcleo conceitual, em especial para as estratégias de *pricing* e para as técnicas de pré-processamento, por contribuírem bastante com a eficiência dos métodos tipo simplex.

Referências Bibliográficas

Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. *Data structures and algorithms.* Addison Wesley, 1983.

Arenales, M. N.; Armentano, V. A.; Morabito, R.; Yanasse, H. *Pesquisa operacional.* Editora Campus, 2007.

Bartels, R. H.; Golub, G. H. The simplex method of linear programming using LU decomposition. *Communications of the ACM*, v. 12, n. 5, p. 266–268, 1969.

Bazaraa, M. S.; Jarvis, J. J.; Sherali, H. D. *Linear programming and network flows.* 2 ed. John Wiley & Sons Inc., 1990.

Benichou, M.; Gauthier, J. M.; Hentges, G.; Ribiere, G. The efficient solution of large-scale linear programming problems - some algorithmic techniques and computational results. *Mathematical programming*, v. 13, p. 280–322, 1977.

Bixby, R. E. Implementing the simplex method: The initial basis. *ORSA Journal on Computing*, v. 4, n. 3, p. 267–284, 1992.

Bixby, R. E. Solving real-world linear programs: A decade and more of progress. *Operations Research*, v. 50, n. 1, p. 3–15, 2002.

Carstens, D. M. Crashing techniques. In: Orchard-Hays, W., ed. *Advanced linear-programming computing techniques*, McGraw-Hill Book Company, p. 131–139, 1968.

Dantzig, G. B.; Orchard-Hays, W. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, v. 8, n. 46, p. 64–67, 1954.

Dantzig, G. B.; Thapa, M. N. *Linear programming*, v. 1. Springer Series in Operations Research, 1997.

Forrest, J. J.; Goldfarb, D. Steepest-edge simplex algorithms for linear programming. *Mathematical programming*, v. 57, p. 341–374, 1992.

Forrest, J. J. H.; Tomlin, J. A. Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical programming*, v. 2, p. 263–278, 1972.

Fourer, R.; Gay, D. M. *Experience with a primal presolve algorithm.* Relatório Técnico, AT&T Bell Laboratories, 1993.

GAY, D. M. Eletronic mail distribution of linear programming test problems. *Lucent Bell Laboratories*, p. 1–13, 1997.

GILL, P. E.; MURRAY, W.; SAUNDERS, M. A.; WRIGHT, M. H. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical programming*, v. 45, p. 437–474, 1989.

GOULD, N. I. M.; REID, J. K. New crash procedures for large systems of linear constraints. *Mathematical programming*, v. 45, p. 475–501, 1989.

HARRIS, P. Pivot selection method of the devex LP code. *Mathematical programming*, v. 5, p. 1–28, 1973.

ILOG INC. Solver CPLEX. `http://www.ilog.com/products/cplex/`, 2008.

KIM, W.; LIM, S.; DOH, S.; PARK, S.; AHN, J. Numerical aspects in developing LP softwares, LPAKO and LPABO. *Journal of Computational and Applied Mathematics*, v. 152, p. 217–228, 2003.

KOBERSTEIN, A. *The dual simplex method, techniques for a fast and stable implementation.* Tese de Doutoramento, University of Paderborn II, Berlin, Germany, 2005.

KOBERSTEIN, A. Progress in the dual simplex algorithm for solving large scale lp problems: techniques for a fast and stable implementation. *Computational Optimization and Applications*, v. 41, p. 185–204, 2008.

KOCH, T. The final netlib-LP results. *Operations Research Letters*, v. 32, p. 138–142, 2004.

LIM, S.; PARK, S. LPAKO: A simplex-based linear programming program. *Optimization methods and software*, v. 17, p. 717–745, 2002.

MAROS, I. A general phase-1 method in linear programming. *European Journal of Operational Research*, v. 23, p. 64–77, 1986.

MAROS, I. *Computational techniques of the simplex method.* Kluwer Academic Publishers, 2003a.

MAROS, I. A general pricing scheme for the simplex method. *Annals of operations research*, v. 124, p. 193–203, 2003b.

MAROS, I. A generalized dual phase-2 simplex algorithm. *European Journal of Operational Research*, v. 149, p. 1–16, 2003c.

MAROS, I.; KHALIQ, M. H. Advances in design and implementation of optimization software. *European Journal of Operational Research*, v. 40, p. 322–337, 2002.

MAROS, I.; MITRA, G. Simplex algorithms. In: BEASLEY, J. E., ed. *Advances in Linear and Integer Programming*, cap. 1, Oxford Science Publication, p. 1–41, 1996.

MÉSZÁROS, C.; SUHL, U. H. Advanced preprocessing techniques for linear and quadratic programming. *OR Spectrum*, v. 25, p. 575–595, 2003.

MUNARI, P. A.; ARENALES, M. N. Uma abordagem implícita para o método M-grande. In: *Anais do XL Simpósio Brasileiro de Pesquisa Operacional*, SOBRAPO, 2008, p. 2193–2204.

OLIVEIRA, S.; STEWART, D. *Writing scientific software.* Cambridge University Press, 2006.

ORCHARD-HAYS, W. *Advanced linear programming computing techniques.* McGraw-Hill, 1968.

PAN, P.-Q. A primal deficient-basis simplex algorithm for linear programming. *Applied Mathematics and Computation*, v. 196, p. 898–912, 2008.

PAPARRIZOS, K.; SAMARAS, N.; STEPHANIDES, G. An efficient simplex type algorithm for sparse and dense linear programs. *European Journal of Operational Research*, v. 148, p. 323–334, 2003.

SOUSA, R. S.; SILVA, C. T. L.; ARENALES, M. N. Métodos do tipo dual simplex para problemas de otimização linear canalizados. *Pesquisa Operacional*, v. 3, p. 349–382, 2005.

SUHL, L. M.; SUHL, U. H. A fast LU update for linear programming. *Annals of Operations Research*, v. 43, p. 33–47, 1993.

SUHL, U. H. MOPS - Mathematical OPtimization System. *European Journal of Operational Research*, v. 72, p. 312–322, 1994.

SUHL, U. H.; SUHL, L. M. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, v. 2, n. 4, p. 325–335, 1990.

TODD, M. J. The many facets of linear programming. *Mathematical programming*, v. 91, p. 417–436, 2002.

TOMLIN, J. A. On scaling linear programming problems. *Mathematical Programming Study*, v. 4, p. 146–166, 1975.

WOLSEY, L. A. *Integer programming.* Wiley-Interscience, 1998.

WRIGHT, S. J. *Primal-dual interior-point methods.* SIAM, Philadelphia, 1997.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo