

# METODOLOGIAS SIMBÓLICAS PARA A RESOLUÇÃO EFICIENTE DE SISTEMAS LINEARES ESPARSOS COM ESTRUTURA ESTÁTICA

Ricardo Duarte Arantes

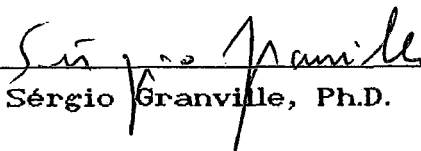
TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:



---

Nelson Maculan Filho, D.Sc.  
(Presidente)



---

Sérgio Granville, Ph.D.



---

Boris Garbati Gorenstin, D.Sc.

ARANTES, RICARDO DUARTE

Metodologias Simbólicas para a Resolução  
Eficiente de Sistemas Lineares Esparsos com  
Estrutura Estática [Rio de Janeiro] 1994

XII, 237 p. 29.7 cm (COPPE/UFRJ, M.Sc., Engenharia  
de Sistemas e Computação, 1994)

Tese - Universidade Federal do Rio de Janeiro,  
COPPE

1. Esparsidade, 2. Otimização, 3. Algoritmos de  
Pontos Interiores, 4. Programação Linear

I. COPPE/UFRJ

II. Título (série).

*A um certo brasileiro de nome Ayrton...*

*A minha mãe*

*Ao amigo Fabiano*

## AGRADECIMENTOS

Esta tese teria sido "meramente concretizada", se contasse apenas com o esforço individual de seu autor.

Assim sendo, boa parte do seu "encanto" jamais acabaria vindo a transpor as meras páginas onde se encontra grafada, se um ingrediente de vital importância não tivesse se mostrado presente durante todos os longos anos em que as técnicas de esparsidade culminaram por ser paulatinamente assimiladas por mim.

Tal "ingrediente" de vital importância foi sem dúvida a interação *humana* com um número considerável de presenças marcantes, e que acabaram por moldar e me levar a reformular boa parte do meu modo de ver e pensar.

Seria portanto por demais extenso citar todos os que de alguma forma contribuíram para que o presente resultado fosse alcançado. O que tenciono nos próximos parágrafos é estabelecer uma aproximação o mais fiel possível das componentes que julgo terem sido as mais relevantes

Agradeço inicialmente portanto a todos os inimigos que por ventura cruzaram pelo meu caminho, pois é sem dúvida graças a resistência as situações embaraçosas pelas quais eu já me vi obrigado a passar, que advém parte da minha garra e disposição de lutar. A esses indivíduos e organizações, ensejo que "continuem do mesmo modo", pois sem dúvida só poderão continuar a me motivar a avançar e afastar de toda a sorte de ineficiências entre tantas outras demonstrações de nítida falta de coerência, que somente os que caminham em sentido retrógrado podem ter como ideal.

Obviamente que a motivação maior desta tese, não foi oriunda de ideais tão pouco nobres como os decorrentes do meu descontentamento com o "errado". A parcela de contribuição desta componente foi mínima, se comparada com a intensidade, clareza e luminosidade emanada por tudo e por todos que julgo tomarem o "certo" como padrão fundamental.



Assim, inicialmente eu venho agradecer a grande receptividade e as valiosas amizades que pude estabelecer no período em que me encontrei como bolsista do Centro de Pesquisas de Engenharia Elétrica (CEPEL), e onde o núcleo desta tese começou a "tomar forma" a partir do final do saudosos ano de 1988 (onde 386's de 20 Mhz eram o "topo da linha", custavam uma "pequena fabula" e andavam numa velocidade comparável a de "verdadeiras carroças", se comparadas com as obtidas por um brasileiro de nome Ayrton, e que acabava de começar a despontar no cenário mundial, com um verdadeiro "foguetete" nas mãos para os padrões da época).

Os meus agradecimentos são portanto a toda a equipe do "antigo DSPT", da qual, me atendo numa ordem puramente alfabética, posso citar o inestimável apoio e atenção recebidas ao longo de todos estes anos, particularmente de: Ayrton, Boris, Cândida, Cordeiro, Diniz, Flávio, Gerson Couto, Granville, Hermínio, Joari, Leslie, Luiz Maurício, Marciano, Mário Veiga, Néelson Martins, Nora, Paulo Alexandre, Pedro, Ronald, Sérgio Henrique, Sérgio Porto, Sílvio e Sueli.

Valiosas trocas de ideias e informações com os pesquisadores: Boris Garbati Gorenstin, Sérgio Porto Romero, Paulo Alexandre Machado, Luiz Antônio Cordeiro, Marciano Morozowski Filho e Sérgio Granville, tornaram extremamente mais fáceis e agradáveis os meus "primeiros passos" na área de resolução de sistemas lineares esparsos. Sem dúvida é desse período inicial que adveio boa parte do meu "gás" em perseverar numa área aparentemente fadada ao esquecimento ou ao insucesso.

O presente trabalho, muito provavelmente não existiria na sua forma atual, caso eu não tivesse tido o "livre acesso" aos códigos de fatoração simbólica desenvolvidos pelos pesquisadores Maurício Resende e Geraldo Veiga, que juntamente com o "restante do time" de Programação Linear via Pontos Interiores, chefiados pelo Ilan Adler na universidade de Berkeley tornaram viável a primeira implementação reconhecidamente bem sucedida de algoritmos capazes de rivalizar com o até então "soberano absoluto" método Simplex.

Foi graças a um "erro de avaliação" entre o desempenho de rotinas simbólicas e convencionais de fatoração, que "tudo começou"... (Esse "erro" foi de fundamental importância para que o meu interesse por ambas as estratégias de solução para o problema fosse definitivamente "despertado" da forma mais imparcial possível. Um raro "batismo" para quem estava começando a dar apenas os seus primeiros passos).

Venho assim, reconhecidamente agradecer a meu orientador Clóvis Caesar Gonzaga, por ter sempre procurado me fornecer as melhores condições de trabalho possíveis, bem como alargado em muito os meus horizontes não apenas na área de Algoritmos de Pontos Interiores, mas numa escala de muito maior relevância pessoal e *humana* para mim. Boa parte do meu caráter como pesquisador, deve-se a uma presença que sem dúvida foi uma das mais marcantes no meu modo de agir e pensar, desde que tive o prazer de conhecê-lo, nos "idos anos" de 1987...

Eu não poderia deixar de agradecer também a todas as amizades pessoais que pude estabelecer ao longo dos anos, e que sem sombra de dúvida contribuíram diretamente para o "recarregamento das minhas energias", afastando-me sempre que possível das atividades técnicas que invariavelmente absorveram a maior parte do meu tempo útil. Assim, expresso meus agradecimentos numa ordem próxima da cronológica/alfabética, particularmente a: Amélia, Betty, Fernando, Renata, Adriana, Marcelo, Valéria, Rogério, Secchin, Marcos Araújo, Lúcia, Luiz Ernesto, Marcelo Avidos, Yuri, Pedro Augusto, Tortorelli, Adriano, Victor, João Batista, Osmani, Annaruma, Garay, José Antônio, Murinho, George Randolph, Erika, Fábio, Delfim e Hsing.

Expresso meus agradecimentos também a alguns dos colegas que tive o prazer de conhecer durante o convívio acadêmico, e que sem dúvida serão sempre lembrados atenciosamente por mim, destacando numa ordem acadêmico/cronológica, particularmente a: Erivaldo, Sandra, Plácido, Fernanda, Marielba, Ingrid, Nicelli, Maurício Nardone, Mariano, Favre, Juliana e Nahri.

Um agradecimento especial, a alguns profissionais "com um pouco mais de experiência e vivência", e que sempre dispensaram valiosa atenção a minha pessoa, destacando-se: Afonso Celso del Nero Gomes, Carlos Gonzaga, Fernando Spinolla, Néelson Maculan Filho e Luiz Costa.

Não posso deixar de expressar também o meu reconhecimento pelo significativo incentivo decorrente de algumas disciplinas por mim cursadas na UFRJ, destacando-se as ministradas por Cláudio Amorim e Lilian Markenzon, nas áreas de arquiteturas paralelas e estruturas de dados.

Gostaria de deixar registrado o meu sincero agradecimento a Armando Castro Filho, que nos idos anos de 1980, veio a me "introduzir com o pé direito" na utilização de computadores de grande porte, e em particular na área de cálculo matricial, (um vício que desde então eu jamais consegui "deixar completamente de lado"...)

Finalmente desejo expressar o inestimável valor e reconhecimento pessoal, pelos trabalhos desenvolvidos por Maurício Resende, Geraldo Veiga, Sergio Pissanetsky e Fred Gustavson na área de resolução de sistemas lineares esparsos, e que em muito influenciaram e contribuíram para sua extensão neste presente trabalho.

Com relação as citações especiais, e para as quais este trabalho é dedicado, cabe mencionar as homenagens que presto a minha mãe e o amigo Fabiano, que estiveram presentes em muitos dos bons e maus momentos pelos quais eu já passei, e um elogio todo especial, a um brasileiro que com muita garra, dedicação, profissionalismo, coragem, disposição, habilidade, sagacidade e tenacidade, defendia as cores de nosso país, levando-as na maioria das vezes ao lugar mais alto do pódio.

Os laços de admiração e simpatia, pelo indivíduo mais capacitado a guiar artefatos de quatro rodas produzidos pelo homem, que o mundo já conheceu, simplesmente não se romperam, com a sua súbita ausência do cenário atual, em pleno auge de sua carreira. Na verdade tornam-se ainda mais fortes e renovados, quando se percebe ao mesmo tempo o valor e a banalidade de muito o que nos rodeia neste mundo. A um grande profissional e batalhador, o meu mais sincero reconhecimento.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

**METODOLOGIAS SIMBÓLICAS PARA A RESOLUÇÃO EFICIENTE DE SISTEMAS LINEARES ESPARSOS COM ESTRUTURA ESTÁTICA**

Ricardo Duarte Arantes

ABRIL, 1994

Orientador: Clovis Caesar Gonzaga

Programa: Engenharia de Sistemas e Computação

Neste trabalho são apresentadas novas metodologias baseadas no uso de pré-processamento simbólico dedicado à resolução eficiente de sistemas lineares esparsos com estrutura estática.

A classe de matrizes e aplicações para as quais os métodos apresentados são voltados engloba as matrizes simétricas definidas positivas e aplicações onde sucessivas refatorações de matrizes com a mesma disposição estrutural de elementos não nulos são efetuadas ao longo do processo.

Nestes casos, mediante a introdução de uma fase auxiliar de pré-processamento simbólico, em que uma análise estrutural de toda a etapa de solução para cada matriz a ser fatorada é efetuada previamente uma única vez, consegue-se (de posse de alguma forma de informação obtida como sub-produto desta fase), elevar-se o desempenho final da fase numérica de solução em que repetidas refatorações tendo por base os novos esquemas propostos são efetivamente realizadas.

Duas formas de metodologias são consideradas neste trabalho: Abordagens híbridas por "janelas de processamento" e abordagens fundamentalmente simbólicas, baseadas nas informações obtidas a partir da árvore de eliminação e do grau de "parentesco" entre as linhas contribuintes em cada etapa do processo de eliminação.

No primeiro caso diversos níveis de janelas são estabelecidos, tendo por base padrões tipicamente encontráveis durante a fatoração de matrizes esparsas, indo desde sub-matrizes diagonais, até o caso completamente denso.

Na segunda abordagem, uma análise mais elaborada de padrões comuns é efetuada, tendo por objetivo a compactação do volume de informação simbólica a ser utilizada definitivamente na fase numérica da solução.

Para tal, um reordenamento dinâmico das contribuições a serem adicionadas em cada etapa do processo é proposto neste trabalho, tomando-se como base a natureza estruturalmente aditiva das contribuições de descendentes de um mesmo nó ancestral.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

**SYMBOLIC METHODOLOGIES FOR THE EFFICIENT SOLUTION OF  
SPARSE LINEAR SYSTEMS WITH STATIC STRUCTURE**

Ricardo Duarte Arantes

APRIL, 1994

Thesis Supervisor: Clovis Caesar Gonzaga

Department: Systems and Computation Engineering

In this work new methodologies based on the use of symbolic pre-processing dedicated to the efficient solution of sparse linear systems with static structure are presented.

The class of matrices and applications for which the presented methods are addressed comprise the symmetric positive-definite matrices and applications where successive refactorizations of matrices with the same structural pattern of non zero elements take place along the process.

In this cases, with the aid of the introduction of an auxiliary phase of symbolic pre-processing, where a structural analysis of the hole solution stage for each matrix to be factored is done previously a single time, one arrives (with the knowledge of some sort of information obtained as a sub-product of this phase), to an increase in the final performance of the numerical solution phase where repeated refactorizations taking for basis the new schemes proposed are effectively done.

Two forms of methodologies are considered in this work: Hybrid approaches by "processing windows" and truly symbolic approaches, based on the information obtained from the elimination tree and the degree of "relationship" between the contributing lines at each stage of the elimination process.

In the first case several levels of windows are established, taking for basis patterns typically obtainable during the factorization of sparse matrices, going from diagonal sub-matrices to completely dense ones.

In the second approach, a more elaborate analysis of common patterns is done, taking for objective the compaction of the symbolic information volume to be definitively used in the numerical phase of the solution.

For that, a dynamic reordering of the contributions to be added at each stage of the process is proposed in this work, taking for basis the structurally additive nature of contributions from descendants from a same ancestral node.

## ÍNDICE

	<i>página</i>	
<b>Capítulo I</b>	<b>Introdução</b>	1
I.1	Conceituação do problema	5
I.2	Restrições de natureza computacional	19
I.3	Evolução histórica dos métodos de esparsidade	35
<b>Capítulo II</b>	<b>Abordagem convencional</b>	41
II.1	Estruturas elementares de armazenamento	42
II.2	Ordenamento visando a redução de <i>fill-in</i>	50
II.3	Obtenção da estrutura da matriz de fatores	58
II.4	Metodologias convencionais para a fase numérica	64
<b>Capítulo III</b>	<b>Abordagens simbólicas</b>	75
III.1	Técnicas visando ao aumento da eficiência	76
III.2	Abordagem simbólica por listas de endereços	94
III.3	Introdução do conceito de <i>supernodes</i>	102
III.4	Abordagem híbrida por janelas	117
<b>Capítulo IV</b>	<b>Nova abordagem proposta</b>	130
IV.1	Motivação computacional e princípios básicos	132
IV.2	Exploração da árvore de eliminação	151
IV.3	Reordenamento ancestral das contribuições	162
IV.4	Formulação completa do método	188
<b>Capítulo V</b>	<b>Resultados Computacionais</b>	210
<b>Capítulo VI</b>	<b>Conclusões</b>	217
<b>Apêndices</b>		
A	Notação e estruturas de dados básicas	219
B	Glossário	223
<b>Referências Bibliográficas</b>		
		228



## Capítulo I

### INTRODUÇÃO

Este trabalho trata sobre técnicas para a resolução eficiente de sistemas lineares esparsos em arquiteturas escalares convencionais via a utilização de métodos diretos para a fatoração da matriz do sistema.

Em meados dos anos 90, poderia-se perguntar o que se esperaria alcançar de novo, em um tema um tanto já explorado na literatura ao longo das últimas duas décadas.

Com o constante desenvolvimento de novas arquiteturas de computadores como as do tipo paralelo ou vetorial, a atenção da literatura, a partir de meados dos anos 80, passou a se voltar quase que exclusivamente para a implementação de métodos de solução esparsa nas novas máquinas de maior desempenho computacional.

A motivação para esta tese veio justamente do fato de haver se deixado de lado uma área que ainda poderia se mostrar fértil, independentemente dos avanços alcançados com a supercomputação.

A idéia básica que motivou este trabalho, foi apresentada pela primeira vez no início dos anos 70 em [G14], e pelo que se pode atribuir a uma fatalidade do destino, acabou fadada ao esquecimento alguns anos depois, simplesmente porque na época não se dispunha de aparatos computacionais suficientemente poderosos, que pudessem viabilizar a plena utilização do método para problemas de grande porte.

Independentemente deste fato, a área de esparsidade continuou inexoravelmente a se desenvolver num ritmo cada vez mais elevado, a ponto de com os aprimoramentos sucessivos pelos quais os métodos tradicionais de solução passaram ao longo dos anos, o seu desempenho nas novas arquiteturas se aproximar ou até mesmo superar a idéia original de Gustavson que havia sido deslumbrada tendo em mente apenas as arquiteturas escalares convencionais da época.

Nesta classe particular de arquiteturas no entanto, o método de Gustavson permanece imbatível por ser teoricamente a abordagem mais livre de *overheads* de implementação possível

Cuma vez que o código produzido é do tipo *loop-free* ou seja linear, sem qualquer laço de desvio, variáveis auxiliares, acessos do tipo indireto à memória, etc...).

Poderia-se portanto perguntar o porque de até hoje não mais se utilizar um método supostamente o mais eficiente para uma certa classe de arquiteturas, e a razão é simplesmente porque para alcançar este elevado desempenho, o método acaba por dispendir um exacerbado volume de espaço de armazenamento na forma de linhas de código "dedicado" gerado.

Tal fato poderia ser amenizado nos dias atuais, em que máquinas com Giga bytes de memória viabilizariam a implementação do método para uma boa parcela dos problemas esparsos de pequeno a médio porte atuais.

Contudo, o maior problema com relação a implementação plena das idéias de Gustavson nas máquinas de hoje (com maior disponibilidade de recursos), é que quase todas se baseiam nas novas arquiteturas de alto desempenho do tipo paralelo ou vetorial, e para as quais o método original de Gustavson sabidamente não é adequado.

Toda esta breve discussão serve para ilustrar no entanto, o dilema clássico "desempenho computacional" *versus* "recursos de armazenamento" dispendidos, e pelo qual a área de solução de sistemas lineares esparsos inexoravelmente é obrigada a passar, cada vez que se introduzem novas arquiteturas ou se formulam novos métodos de solução (o que será um pouco mais detalhado na seção I.2).

O ponto que deu origem a este trabalho foi exatamente a expectativa de poder se encontrar uma nova metodologia para a resolução de sistemas lineares esparsos, que tivesse um desempenho o mais próximo possível de uma abordagem como a de [G14], mas que mantivesse em níveis aceitáveis os requisitos de espaço de armazenamento e de código.

Uma primeira abordagem para este problema teve início em [A8], no qual se chegou ao que se poderia denominar "métodos híbridos" baseados em "janelas de processamento", e que serão apresentados no capítulo III.

O presente trabalho portanto é uma extensão direta de [A8], em que uma nova abordagem ainda mais eficaz para o problema é apresentada.

Esta nova metodologia possui um caráter bem mais

genérico do que as abordagens híbridas propostas anteriormente, e é fundamentada em sólidos conceitos, como a exploração da árvore de caminhos de eliminação, o que será visto em detalhes a partir do capítulo IV onde se apresentam todas as ferramentas conceituais auxiliares utilizadas pelo novo método.

No presente capítulo será conceituado o problema de resolução de sistemas lineares com estrutura estática a ser abordado ao longo de todo o trabalho, bem como as restrições de natureza computacional impostas pelo caráter esparsa do problema.

Uma breve apresentação da evolução histórica dos métodos diretos de solução segue-se ao final deste capítulo, de modo a melhor situar o leitor em face aos crescentes avanços sofridos pela área de esparsidade ao longo das últimas décadas.

No capítulo II são descritas todas as etapas de uma abordagem convencional para a fatoração de matrizes via métodos baseados em "vetores de trabalho expandidos". Entre as fases detalhadas encontram-se a de ordenamento visando a redução de *fill-in's*, a obtenção da estrutura simbólica da matriz de fatores, e as possíveis alternativas para a fase de geração numérica da matriz de fatores resultantes.

As estruturas de dados básicas necessárias para o armazenamento das matrizes esparsas (original e resultante após o processo de fatoração) também são apresentadas neste capítulo e complementarmente no apêndice A.

No capítulo III inicialmente são lançadas as primeiras idéias tendo em vista o aumento da eficiência da fase numérica de solução a ser explorada em detalhes nas seções subsequentes, concentrando-se na apresentação da formulação original de Gustavson [G14] e uma particularização mais econômica (sob o ponto de vista de espaço) desta técnica, voltada para o caso completamente denso.

Neste mesmo capítulo, são introduzidas as abordagens propostas em [A8], como os métodos baseados em "listas simbólicas de endereços", e a metodologia "híbrida por janelas", englobando todas as abordagens anteriores, incluindo também a exploração de "supernodes" (notadamente presentes em problemas típicos de Programação Linear via

Métodos de pontos Interiores como em [A1], [M1]).

No capítulo IV finalmente se apresenta o novo método simbólico proposto, baseado no uso das informações sobre o "parentesco" das diversas contribuições a serem adicionadas a linha corrente da matriz de fatores sendo gerada.

Inicialmente os conceitos fundamentais como a árvore de caminhos de eliminação e a motivação computacional para o novo método são apresentados.

A seguir, uma breve descrição da nova metodologia é introduzida, deixando-se para as seções subseqüentes a formulação mais detalhada das opções de reordenamento dinâmico das contribuições visando a redução do volume de informação a ser codificada para a descrição de toda a seqüência de operações da fase numérica de solução.

No capítulo V são apresentados alguns resultados computacionais preliminares da implementação de algumas das novas técnicas propostas, comparando o seu desempenho com o de implementações consagradas na literatura (tomando como base problemas da NETLIB [G44], extraídos de aplicações na área de Programação Linear via métodos de Pontos Interiores).

Seguem-se as conclusões preliminares do presente trabalho, bem como os apêndices, no qual é formalizada toda a notação de vetores, apontadores e estruturas de dados básicas utilizadas na representação e fatoração de matrizes esparsas ao longo de todo o texto.

## I.1 Conceituação do problema

O problema a ser abordado neste trabalho é o da solução de sistemas de equações algébricas lineares da forma:

$$(I.1) \quad A x = b$$

onde a matriz de coeficientes  $A \in \mathbb{R}^{n \times n}$  é esparsa, simétrica e definida positiva, o vetor solução  $x \in \mathbb{R}^n$  e o vetor lado direito  $b \in \mathbb{R}^n$ .

As abordagens existentes na literatura para a solução de sistemas de equações algébricas lineares dividem-se em dois grandes grupos:

- Métodos diretos
- Métodos iterativos

No presente trabalho, apenas os métodos diretos baseados na fatoração  $U^T D U$  da matriz de coeficientes serão considerados. (Para maiores referências nas abordagens por métodos iterativos, o leitor é reportado à [A1], [B11], [D3], [G38], [G40], [K2], [O3], [O5], [P6], [T15], [V6] e [Y3]).

A solução por meios diretos é baseada em algumas propriedades elementares sobre os sistemas de equações algébricas lineares, podendo-se tomar como referências básicas [G39], [D50], [F1], [D51] e [G5].

Seguem-se portanto algumas das principais propriedades (apenas enumeradas, sem qualquer comprovação formal):

### Propriedade I.1(a)

*A solução de um sistema de equações lineares não se altera quando se multiplica todos os coeficientes de uma dada linha do sistema por uma constante real escalar não nula.*

### Propriedade I.1(b)

*A solução de um sistema de equações lineares não se altera quando se adicionam ou subtraem duas equações (coeficiente a coeficiente), substituindo-se uma delas pela nova equação assim obtida.*

### Propriedade I.1(c)

*A solução de um sistema de equações lineares não se altera quando se permutam duas ou mais linhas de um mesmo sistema entre si.*

**Propriedade I.1(d)**

*A solução de um sistema de equações lineares altera-se apenas a nível de uma permutação de índices das variáveis da solução, quando se permutam duas ou mais colunas associadas ao mesmo sistema entre si.*

**Propriedade I.1(e)**

*Um sistema cuja matriz de coeficientes é real, simétrica e definida positiva, admite apenas uma única solução real.*

**Propriedade I.1(f)**

*Uma matriz simétrica definida positiva admite uma decomposição única na forma  $U^T D U$ , com matrizes  $U \in \mathbb{R}^{n \times n}$  da forma triangular superior com diagonal unitária e  $D \in \mathbb{R}^{n \times n}$  da forma diagonal.*

**Propriedade I.1(g)**

*Uma matriz simétrica definida positiva admite uma decomposição  $U^T D U$  intrinsecamente estável, dispensando deste modo o pivoteamento em valor numérico durante o processo de eliminação de variáveis visando a sua fatoração.*

A essência dos métodos diretos de solução consiste em se chegar a solução do sistema (I.1) mediante sucessivas aplicações de operações de multiplicação das linhas por constantes escalares e adições de duas linhas do mesmo sistema, de modo se a transformar o sistema original numa forma que viabilize a determinação do vetor solução  $x$  (que satisfaz ao sistema original).

A forma que normalmente se adota para tal, é levar a matriz de coeficientes à forma triangular (inferior ou superior), onde uma matriz  $U$  na forma triangular superior (ou  $U^T$  na forma triangular inferior) é assumida como uma matriz que contenha todos os elementos da diagonal unitários, e todos os elementos abaixo da diagonal nulos no caso triangular superior (ou o extremo oposto no caso inferior).

Uma vez transformado o sistema original em um sistema na forma triangular superior (com diagonal unitária):

(I.2)

$$U x = y$$

a solução  $x$  pode ser fácilmente obtida componente a componente, mediante uma etapa de "retro-substituição" de variáveis, pois da última equação de (L2) se obtém  $x_n = y_n$ , e daí para adiante, pode-se substituir o valor de  $x_n$  na equação  $n-1$  e com isso determinar-se  $x_{n-1}$ . De posse do valor destas variáveis, substituindo-as na equação  $n-2$  obtem-se o valor de  $x_{n-2}$ , e assim sucessivamente, até se determinar o valor de  $x_1$ .

O que não irá se demonstrar formalmente neste trabalho reportando-se à [G39], [D50], [G5], é que a matriz  $U$  de (L2) e sua transposta  $U^T$  correspondem as matrizes triangulares com diagonal unitária, que unívocamente determinam a decomposição da matriz  $A$  do sistema original, na forma  $U^T D U$ , com  $D$  uma matriz da forma diagonal.

A luz desta nova interpretação, podem ser enumeradas as seguintes etapas básicas para o processo de solução:

- Determinar matrizes  $U$  e  $D$  que factorem a matriz  $A$  original, na forma  $U^T D U$
- De posse dos factores da decomposição, obter o vetor solução  $x$  mediante solução dos seguintes sub-problemas:

$$(L3) \quad U^T z = b \quad (\text{forward})$$

$$(L4) \quad D y = z$$

$$(L5) \quad U x = y \quad (\text{backward})$$

O presente trabalho irá concentrar-se exclusivamente na etapa de factoração da matriz do sistema, uma vez que as etapas subsequentes de retro-substituição mesmo no caso de sistemas esparsos, não oferecerem maiores dificuldades de natureza computacional a nível de implementação em arquiteturas escalares convencionais.

A idéia básica do método de factoração matricial, conhecido na literatura muitas vezes como método de eliminação Gaussiana [G39] (para o caso mais geral em que a matriz  $A$  é a priori considerada assimétrica), ou método de Cholesky [F1] (no caso de uma factoração da forma  $L L^T$  ao invés de  $U^T D U$  conhecido como eliminação de Gauss

simétrica), é simplesmente ir se levando gradativamente a matriz original  $A$  até a forma  $U$ , mediante repetidas aplicações das propriedades básicas I.1(a) e I.1(b) de modo a se anular os coeficientes desejados.

Para tal existem 3 alternativas possíveis de atualização (com a forma de acesso aos elementos de  $A$  e de geração dos fatores de  $U$  ilustradas nas figuras correspondentes):

#### Alternativa I.1(A) Atualização por colunas

- A cada etapa  $i$  do processo, anular os elementos à esquerda da diagonal, da  $i$ 'ezima linha mediante a subtração de múltiplos escalares das linhas anteriores atualizando-se desta forma a porção acima da diagonal na  $i$ 'ezima coluna, por operações da forma:

para  $i$  de 2 até  $n$   
 para  $k$  de 1 até  $i-1$   
 $u_{k,i} \leftarrow a_{k,i} / a_{k,k}$   
 para  $j$  de  $k+1$  até  $i$   
 $a_{j,i} \leftarrow a_{j,i} - u_{k,i} * a_{k,j}$

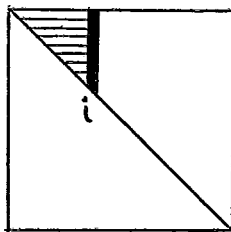


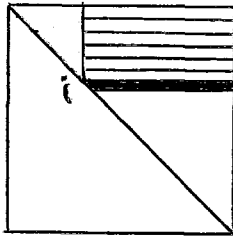
fig I(1) - Atualização por colunas

#### Alternativa I.1(B) Atualização por linhas

- A cada etapa  $i$  do processo, anular os elementos à esquerda da diagonal, na  $i$ 'ezima linha mediante a subtração de múltiplos escalares das linhas anteriores atualizando-se desta forma a porção a direita da diagonal na  $i$ 'ezima linha, por operações da forma:

para  $i$  de 2 até  $n$   
 para  $k$  de 1 até  $i-1$   
 $u_{k,i} \leftarrow a_{k,i} / a_{k,k}$   
 para  $j$  de  $i$  até  $n$   
 $a_{i,j} \leftarrow a_{i,j} - u_{k,i} * a_{k,j}$



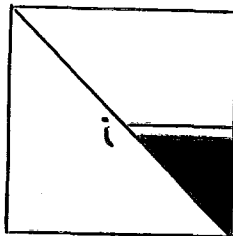
fig I(2) - Atualização por linhas**Alternativa I1(C)**Atualização por submatrizes

- A cada etapa  $k$  do processo, anular os elementos abaixo da diagonal na  $k$ 'ezima coluna mediante a subtração de múltiplos escalares da  $k$ 'ezima linha das demais linhas abaixo desta, atualizando-se desta forma a sub-matriz definida a partir de  $a_{k+1,k+1}$  até  $a_{n,n}$  por operações da forma:

```

para k de 1 até n-1
  para i de k+1 até n
     $u_{k,i} \leftarrow a_{k,i} / a_{k,k}$ 
    para j de i até n
       $a_{i,j} \leftarrow a_{i,j} - u_{k,i} * a_{k,j}$ 

```

fig I(3) - Atualização por submatrizes

Em todos os processos de atualização acima utilizou-se o fato da matriz original ser simétrica, confinando-se as operações apenas sobre os elementos da porção triangular superior que ao final do processo acabará contendo os fatores  $u_{i,j}$  (obtidos simplesmente reescrevendo-se os valores originais  $a_{i,j}$  ao final de cada etapa).

O processo completo de eliminação será visto em detalhes no capítulo II, e por ora omitiu-se detalhes visando uma maior eficiência computacional, como por exemplo a normalização dos elementos diagonais (armazenando-se em seu lugar o inverso dos valores obtidos ao final de cada etapa e que acabarão sendo necessários durante a fase (I.4) de retro-substituições).

Até o momento se apresentaram alternativas e

propriedades gerais, válidas para sistemas lineares cuja matriz de coeficientes seja simétrica e definida positiva, sem se explorar qualquer outro padrão estrutural da mesma.

O presente trabalho se concentra na solução de sistemas cuja matriz de coeficientes além das propriedades anteriores, possui um percentual pequeno de elementos não nulos, se comparado com o número total de coeficientes de uma matriz completamente "cheia" (ou seja, com todos os elementos distintos de zero).

Tais sistemas são denominados "esparços", em oposição aos sistemas tratados como "densos" (onde não se procura explorar qualquer aspecto de eficiência computacional do padrão estrutural de elementos não nulos que porventura existam na matriz original ou na de fatores resultantes).

Seguem-se portanto algumas definições, acrescidas de um breve comentário sobre o "conceito" de esparsidade.

**Definição I.1(1) Densidade de uma matriz**

*Define-se como densidade  $\rho$  de uma matriz de dimensão  $n$ , a razão entre o seu número total de elementos não nulos dividido por  $n^2$  (correspondendo esta última grandeza, ao número total de elementos de uma matriz de mesma dimensão  $n$ ).*

**Definição I.1(2) Matriz Completamente Esparsa**

*Define-se uma matriz como "completamente esparsa" ou "diagonal", quando apenas os elementos da sua diagonal são distintos de zero.*

**Definição I.1(3) Matriz Esparsa**

*Define-se uma matriz como "esparsa", quando a densidade desta é inferior a um certo percentual, aceito como relativamente baixo (na maioria dos casos inferior a 1%, podendo em função da dimensão da matriz, tolerar-se percentuais de densidade mais elevados ou reduzidos).*

**Definição I.1(4) Matriz Quase Densa**

*Define-se uma matriz como "quase densa", quando a densidade desta é superior a um certo percentual, aceito como relativamente elevado (na maioria dos casos superior a 90%, podendo em função da sua dimensão, tolerar-se percentuais de densidade mais reduzidos ou elevados).*

**Definição I.1(5)****Matriz Completamente Densa**

*Define-se uma matriz como "densa", "completamente densa" ou "cheia", quando a densidade desta é de 100% (ou seja todos os seus elementos são distintos de zero).*

Um breve comentário sobre a interpretação das definições acima faz-se necessário, uma vez que o caráter "esparso" ou "denso" de um sistema, é na verdade uma questão que depende intimamente da forma de representação e armazenamento do mesmo, bem como da possível exploração destas características visando a redução de algum outro aspecto de natureza computacional, como por exemplo o tempo de computação ou o número de operações aritméticas efetuadas.

Em princípio uma matriz não pode ser considerada a priori como densa ou esparsa por si só. A forma como ela será tratada e armazenada é que ditará qual das duas características será plenamente explorada.

Assim, é possível ter uma matriz teóricamente considerada "esparsa" (segundo um certo padrão de densidade reduzida), porém armazenada e encarada para fins computacionais, como se todos os seus elementos fossem distintos de zero. Neste caso, o caráter esparso não estará sendo explorado, e para fins computacionais, a matriz em questão acabará sendo tratada como "densa".

Do mesmo modo, uma matriz tida como "quase densa" por algum padrão, pode ser encarada para fins computacionais e de armazenamento, como "completamente densa" ou "esparsa".

Até mesmo o que se poderia considerar a priori como não cogitável, como representar uma matriz "completamente densa" (ou uma porção desta), como "esparsa" pode vir a ocorrer com frequência em certas aplicações como a fatoração de matrizes encontradas na solução de problemas de Programação Linear via Algoritmos de Pontos Interiores, como em [A1].

● *Percebe-se portanto que o conceito de "esparsidade" está intimamente ligado à natureza computacional do processo de eliminação.*

Desta forma, passaremos a considerar apenas abordagens esparsas para o processo de solução, pelo fato da implementação de técnicas para o processamento de matrizes densas em arquiteturas escalares convencionais, ser

relativamente trivial e amplamente estudado na literatura, como em [G39], [F1], [F2], [C16].

O problema da fatoração esparsa está diretamente ligado portanto a dois fatores determinantes do êxito de uma implementação:

- Estrutura de armazenamento

- Metodologia de solução

- *Numa abordagem esparsa, objetiva-se dispender o menor espaço de armazenamento possível, de modo a permitir que a solução de sistemas de porte elevado permaneça viável, em face as limitações de memória "real" da arquitetura onde o método será implementado.*

- *Do mesmo modo, objetiva-se dispender o menor esforço computacional possível, de modo a permitir que a solução de sistemas de porte elevado permaneça viável, em face as limitações de tempo máximo de CPU disponível.*

Percebe-se que ambos os objetivos convergem complementarmente para um mesmo ponto, qual seja o da máxima eficiência computacional, medida a nível dos recursos de armazenamento e tempo de CPU dispendidos.

Fica claro portanto, que uma imensa gama de metodologias e implementações podem ser adotadas para o mesmo problema, privilegiando-se de alguma forma um dos recursos básicos como redução de memória ou tempo de CPU.

É objetivo desta tese, "varrer" de uma forma um tanto quanto sistemática (sem se tornar exaustiva), algum dos "eixos coordenados" definidos pelas duas vertentes de eficiência apresentadas e uma parcela intermediária do espaço de implementações procurando-se um balanceamento de recursos.

A luz desta forma de abordagem, apresentam-se características inerentes aos sistemas esparsos, e passíveis de exploração por algumas das abordagens a serem consideradas neste trabalho.

#### **Característica 1.1[a]**

*Numa abordagem esparsa para a representação de matrizes ou vetores, apenas os seus elementos não nulos precisam ser explicitamente armazenados.*

### Característica I.1b1

*Numa abordagem esparsa para o processo de adição ou multiplicação de matrizes (ou vetores por escalares), apenas as operações sobre os elementos não nulos precisam ser explicitamente efetuadas.*

Um método de eliminação atende aos requisitos eficiência de modo a ser considerado como uma abordagem "típicamente esparsa" para o problema, quando ambas as características I.1a1 e I.1b1 são exploradas concomitantemente.

Tal fato se dá pelo fato do ônus computacional (no caso de alguma destas características não serem integralmente exploradas) ser na maioria das vezes intolerável.

Tais questões, serão objeto de uma análise mais criteriosa na seção I.2, de modo que nos concentraremos a partir de agora, até o final desta seção em justificar o porque da "fama" de um tanto quanto "high-tech" e cercada de "mistério" ser encarada a área de esparsidade, por parte dos "não iniciados", desde os primórdios da solução direta de sistema lineares de grande porte até os dias atuais.

Grande parte desta "fama" advém da maior complexidade computacional dos códigos de eliminação (o que será visto no capítulo II), das diversas alternativas para "balanceamento" dos recursos dispendidos (apresentadas ao longo de todo este trabalho), e da existência de alguns problemas básicos inerentes apenas as abordagens esparsas para o processo de eliminação (a serem analisados em detalhe no capítulo II).

Um dos problemas básicos (e que será amplamente apresentado na seção II.4) é o de se efetuar a soma de 2 vetores contendo alguma forma de representação esparsa para os coeficientes não nulos das linhas a eles associados.

Tal problema corresponde à etapa de combinação linear de 2 linhas de modo a se anular algum dos coeficientes de uma das equações, consistindo na exploração da propriedade básica I.1(b).

Diferentes alternativas, procurando minimizar o overhead adicional de espaço de armazenamento ou de tempo de CPU (com um menor número de operações adicionais de indexação) são possíveis, e serão apresentados no capítulo II.

A forma mais eficiente com que se puder implementar esta

etapa básica do processo de eliminação será na maioria das vezes a determinante para o êxito global do método, visto ser a etapa de combinação linear de vetores esparsos, a etapa mais central do processo, correspondendo ao *loop* mais interno de todas as alternativas I.1(A), I.1(B) ou I.1(C) apresentadas.

Outra característica associada a implementações esparsas para o processo de eliminação (e que será vista em detalhes na seção II.2), advém da seguinte indagação a que se poderia chegar, após uma minuciosa observação da forma pela a qual se processam as combinações lineares de linhas representadas num formato esparsa:

● *Seria possível se reduzir o esforço computacional quer a nível do espaço total de armazenamento ou do número de operações aritméticas efetuadas sobre elementos não nulos, mediante alguma forma de permutação de linhas ou colunas aplicadas sobre a matriz original do sistema ou durante o processo de solução ?*

A resposta a esta indagação é afirmativa na grande maioria dos casos, pois normalmente a disposição estrutural de elementos não nulos de um sistema, na forma em que ele originalmente é formulado, não leva em conta a natureza esparsa do processo de sua solução, o que dá margem a que mediante operações de permutação de linhas/colunas (que pelas propriedades I.1(c) e I.1(d) não alteram a solução do sistema, a menos de uma permutação dos índices das variáveis), se consiga obter uma redução do esforço computacional como do espaço de armazenamento necessário para representação da matriz de fatores resultante (e concomitantemente, do número total de operações aritméticas necessárias para se completar o processo de eliminação como um todo).

Este problema de reordenamento das equações, vem recebendo enorme atenção na literatura desde a sua revolucionária introdução no artigo apresentado por Tinney [T2] no final da década de 60.

O problema de reordenamento está intimamente ligado a um novo conceito, inerente aos métodos de eliminação esparsa, qual seja o da criação de "novos" elementos não nulos ao

longo das etapas do processo de eliminação. Tais elementos receberam um nome especial na literatura, razão pela qual serão apresentados na forma da definição abaixo.

### **Definição I.1(6)**

### **Fill-In**

*Define-se por "elemento de fill-in", ou doravante ao longo do texto simplesmente por "fill-in", a um novo elemento não nulo criado durante o processo de combinação linear esparsa de 2 vetores explorando-se a propriedade I.1(b).*

A luz desta nova definição, percebe-se que métodos de reordenamento que visem uma redução da criação de elementos de *fill-in* são sempre desejáveis, trazendo consigo um benefício "duplo":

- *Com a redução de fill-in's, consegue-se não só uma redução no espaço final de armazenamento, como também uma redução do esforço total de computação a nível de operações aritméticas, uma vez que evitando-se a criação de novos elementos durante o processo, evita-se também a propagação das operações aritméticas sobre os mesmos e que seriam necessárias caso estes elementos passassem a assumir um valor não nulo, a partir de alguma etapa do processo de eliminação.*

Um critério análogo ao da redução de *fill-in's* é o da redução do esforço computacional, ou seja das operações aritméticas efetuadas para a eliminação de um dado coeficiente, escolhendo-se como candidato a sofrer a eliminação a cada etapa, aquele em que o processo de combinações lineares com as demais equações visando eliminá-lo, resultar no menor número de operações sobre elementos não nulos possível.

Este na verdade foi o primeiro critério de reordenamento, originalmente proposto por Markowitz [M3], para o caso de matrizes assimétricas.

No caso simétrico o critério de Markowitz acaba fornecendo o critério mais consagrado na literatura (o *minimum degree*), e introduzido independentemente por Tinney [T2].

Até o presente momento, percebe-se nitidamente a presença de duas fases distintas do processo de eliminação

esparsa: O reordenamento prévio das linhas/colunas da matriz original, e o processamento das combinações lineares entre as linhas do sistema, de modo a se anular alguns dos coeficientes, levando o sistema a forma triangular superior.

Uma complicação de natureza computacional, advém (no caso de um tratamento esparso para a eliminação), em decorrência do efeito que se convencionou chamar por "propagação dos *fill-in's*".

A complicação advém do fato, da criação de *fill-in's* ser um processo cumulativo de uma etapa para outra do processo (ou seja os novos elementos de *fill-in* introduzidos numa etapa, acabarem sendo responsáveis pela criação de novos elementos de *fill-in* nas etapas subseqüentes da eliminação, e assim por diante, até o final do processo).

Em face desta natureza "cumulativa", uma primeira abordagem para o processo de eliminação esparsa pode ser baseada num tratamento "dinâmico" dos *fill-in's* a cada etapa, o que sem dúvida acarreta perdas na eficiência global da implementação, pois as estruturas finais de cada linha da matriz de fatores resultante, só estariam disponíveis ao final do processo, e alguma forma de alocação dinâmica de espaço (como inserção em listas encadeadas), acabaria sendo necessária, contribuindo assim com um *overhead* significativo em termos de tempo de processamento adicional.

Visando contornar esta situação, a partir do início dos anos 80 com os trabalhos de [G3], passou-se a considerar a adoção de uma nova fase auxiliar no processo de eliminação.

Tal fase recebeu o nome de "fatoração simbólica", em contraste com a fase de "fatoração numérica" (onde de fato se processam as operações aritméticas da eliminação), com a única diferença em relação as fatorações numéricas convencionais, de não se processar dinamicamente a criação dos *fill-in's*, assumindo-se que a posição estrutural dos mesmos em cada linha da matriz de fatores resultantes, tenha sido prevista e determinada a priori, em alguma fase anterior, do processo.

Ou seja o que a etapa de fatoração simbólica produz, é na verdade uma "simulação estrutural" do processo de eliminação, retornando a estrutura (simbólica) da matriz de fatores resultante, já incluindo os novos elementos



introduzidos em decorrência da criação de *fill-in's*.

De posse da informação simbólica dos fatores, o processo de combinação linear de vetores armazenados na forma esparsa, pode ser simplificado, e tornado mais eficiente, dispensando qualquer forma de "varredura" ou inserção em listas encadeadas (e que por conter operações de acesso indireto a memória, como notado em [A8], consomem um tempo adicional de processamento quase comparável ao das operações aritméticas em ponto flutuante efetuadas durante a fase numérica do processo de eliminação).

Com a adoção da fase simbólica de pré-processamento, o número de etapas para a implementação da fatoração esparsa, eleva-se para três: o ordenamento, a fatoração simbólica e a fatoração numérica propriamente dita.

O que se poderia perguntar é se "algo mais" ainda poderia ser feito para aumentar a eficiência da fase numérica, visto ser esta a fase a ser empregada mais de uma vez ao longo do processo de solução de qualquer problema global a ser considerado (no qual a solução de sistemas lineares esparsos pode ser encarada apenas como um "sub-problema"), como por exemplo no caso de algoritmos de Pontos Interiores para Programação Linear [A11, IM11], nas áreas de simulação de circuitos e solução de equações diferenciais, entre outras.

A resposta a esta nova indagação também é afirmativa, como o leitor já poderia suspeitar, e será o objetivo desta tese, apresentar algumas das alternativas para se alcançar este aumento na eficiência da fase numérica do processo (nas arquiteturas escalares do tipo convencional).

● *O último ponto que cabe ser mencionado ainda nesta seção, é que os métodos propostos neste trabalho, só se mostram válidos (justificando a sua aplicação), nos casos em que repetidas aplicações do processo de fatoração numérica, com matrizes estruturalmente idênticas de uma aplicação para outra, (diferindo apenas nos valores numéricos dos coeficientes) se mostrem necessárias.*

Nos casos em que a estrutura se altera de uma aplicação para outra, todas as demais fases de reordenamento, fatoração simbólica e de pré processamento visando aumentar a

eficiência da fase numérica, acabariam tendo de ser reafetadas para uma única aplicação da fase numérica, o que sem dúvida seria muito mais oneroso do que se lançar mão de uma fase numérica do tipo convencional, com a criação dinâmica dos *fill-in's*, ou até mesmo de métodos híbridos de reordenamento/fatoração numérica concomitante, que sem dúvida mostram-se os mais adequados e eficientes em aplicações onde o padrão estrutural de elementos não nulos da matriz original sofre alterações a cada aplicação do processo de fatoração.

Percebe-se portanto que uma abordagem bem sucedida para o problema de resolução de sistemas lineares esparsos com estrutura estática requer um conjunto "modular" de etapas a serem aplicadas sobre a matriz original, e que começarão a ser apresentadas em detalhe, a partir do capítulo II.

Antes disso porém, faz-se necessário conceituar os problemas e restrições de natureza computacional, pelas quais qualquer método de solução esparsa deve ser projetado para atender, de modo a "honrar" o status de uma abordagem tipicamente esparsa, visto que em qualquer circunstância, uma abordagem puramente densa para o problema ser de fácil implementação, com custos computacionais na maioria dos casos impraticáveis (especialmente nas arquiteturas escalares convencionais), razão pela qual só se poder contar de fato com os métodos "realmente esparsos" (para a solução direta), ou com os métodos iterativos, cuja eficiência para certas classes de problemas pode ser até mais compensadora, do que a de implementações dos métodos diretos convencionais.

Encerra-se esta seção, com o comentário de que muito ainda se pode fazer no tocante a aprimoração dos métodos diretos de solução, que vem se mostrando até hoje, os mais consistentes para uma vasta classe de problemas encontrados nas aplicações práticas de diversos ramos da engenharia.

Passaremos portanto a analisar as restrições de natureza computacional, e intrínsecas aos sistemas lineares esparsos, independentemente de certa forma da abordagem por métodos de eliminação direta adotada para o processo de solução.

## I.2 Restrições de Natureza Computacional

Esta seção visa dar ao leitor, uma noção da ordem de grandeza dos recursos computacionais dispendidos por abordagens tipicamente esparsas para o processo de eliminação, confrontando-os com abordagens densas para o mesmo problema.

Iniciaremos apresentando os requisitos de memória e o volume de operações aritméticas (inteiras e de ponto flutuante), para abordagens completamente densas, visto serem tais requisitos trivialmente obtidos em função unicamente da dimensão do sistema a se solucionar, sendo desta forma utilizados como base para a comparação com os requisitos das abordagens tipicamente esparsas consideradas.

Serão introduzidas abaixo, algumas notações para a representação do número de elementos não nulos de matrizes e vetores, bem como do número total de operações de ponto flutuante e de aritmética inteira (como as de indexação por exemplo).

- *Em todas as definições, a seguir, assume-se uma matriz  $M$  simétrica, definida positiva e de dimensão  $n$ .*

### Definição I.2(1)

#### $\text{NonzRow}(i,M)$

*Define-se por  $\text{NonzRow}(i,M)$  como sendo o número de elementos não nulos da  $i$ 'ésima linha na porção triangular superior de  $M$  (excluindo-se o elemento pertencente à diagonal).*

### Definição I.2(2)

#### $\text{NonzCol}(i,M)$

*Define-se por  $\text{NonzCol}(i,M)$  como sendo o número de elementos não nulos da  $i$ 'ésima coluna na porção triangular superior de  $M$  (excluindo-se o elemento pertencente à diagonal).*

### Definição I.2(3)

#### $\text{Nflop}(i,M)$

*Define-se por  $\text{Nflop}(i,M)$  como sendo o número de operações de ponto flutuante efetuadas durante a  $i$ 'ésima etapa do processo de eliminação dos elementos de  $M$ .*

**Definição I.2(4)  $N_{\text{indx}}(i,M)$** 

Define-se por  $N_{\text{indx}}(i,M)$  como sendo o número de operações aritméticas inteiras de indexação, efetuadas durante a  $i$ ésima etapa do processo de eliminação dos elementos de  $M$ .

**Definição I.2(5)  $N_{\text{oper}}(i,M)$** 

Define-se por  $N_{\text{oper}}(i,M)$  como sendo o número de operações aritméticas inteiras e de ponto flutuante, efetuadas durante a  $i$ ésima etapa do processo de eliminação dos elementos de  $M$ .

**Definição I.2(6)  $TotNonz(M)$** 

Define-se por  $TotNonz(M)$  como sendo o número total de elementos não nulos da porção triangular superior de  $M$ , incluindo-se os elementos da diagonal.

**Definição I.2(7)  $TotFlop(M)$** 

Define-se por  $TotFlop(M)$  como sendo o número total de operações de ponto flutuante efetuadas durante o processo de eliminação de  $M$ .

**Definição I.2(8)  $TotIndx(M)$** 

Define-se por  $TotIndx(M)$  como sendo o número total de operações aritméticas inteiras de indexação, efetuadas durante o processo de eliminação de  $M$ .

**Definição I.2(9)  $TotOper(M)$** 

Define-se por  $TotOper(M)$  como sendo o número total de operações aritméticas inteiras e de ponto flutuante, efetuadas durante o processo de eliminação de  $M$ .

Complementarmente as definições acima, apresentam-se algumas definições associadas a estrutura de vetores e apontadores utilizados nas representações tipicamente esparsas.

● Em todos os casos, a seguir, é assumido um vetor unidimensional  $v$ , de componentes reais ou inteiras.

**Definição I.2(10)****Len(v)**

Define-se por  $Len(v)$  o comprimento ou dimensão deste vetor, ou seja o número total de posições de memória (reais ou inteiras) alocadas para a sua representação.

**Definição I.2(11)****NonzElem(v)**

Define-se por  $NonzElem(v)$  como sendo o número de seus elementos distintos de zero.

De posse das definições apresentadas nesta seção, passaremos a considerar uma análise do dispêndio computacional de implementações densas para o processo de fatoração, comparando-a posteriormente com abordagens esparsas para o mesmo problema.

Um fato que não será demonstrado neste trabalho, reportando-se à [D6], [G5], [O3] é que o número de operações aritméticas é o mesmo para as 3 alternativas I.1(A), I.1(B) e I.1(C) consideradas na seção anterior, visto serem todos os procedimentos, iterações básicas da forma:

para  $( )$  de  $(\_)$  até  $(\bar{\_})$

para  $[ ]$  de  $[\_]$  até  $[\bar{\_}]$

$$u_{k,i} \leftarrow a_{k,i} / a_{k,k}$$

para  $\{ \}$  de  $\{ \_ \}$  até  $\{ \bar{\_} \}$

$$a_{i,j} \leftarrow a_{i,j} - u_{k,i} * a_{k,j}$$

com  $( )$ ,  $[ ]$ ,  $\{ \}$  denotando alguma permutação dos índices,  $i$ ,  $j$ ,  $k$  e  $(\_)$ ,  $[\_]$ ,  $\{ \_ \}$  bem como  $(\bar{\_})$ ,  $[\bar{\_}]$ ,  $\{ \bar{\_} \}$  uma função de  $n$  e dos índices anteriores.

Consideraremos inicialmente, o número de multiplicações e de subtrações efetuadas em cada uma das alternativas, contabilizando-se apenas o número de uma das operações, por serem efetuadas sempre na forma do par  $a_{i,j} - u_{k,i} * a_{k,j}$ .

Assim temos:

## Alternativa I.1(A)

Multiplicações ou subtrações

$$(I.6) \quad \sum_{i=2}^n \sum_{k=1}^{i-1} \sum_{j=k+1}^i (1) = \sum_{i=2}^n \sum_{k=1}^{i-1} (i - k)$$

$$(I.7) \quad = \sum_{i=2}^n \left( \frac{i(i-1)}{2} \right)$$

$$(I.8) \quad = \frac{n^3}{6} - \frac{n}{6}$$

## Alternativa I.1(B)

Multiplicações ou subtrações

$$(I.9) \quad \sum_{i=2}^n \sum_{k=1}^{i-1} \sum_{j=i}^n (1) = \sum_{i=2}^n \sum_{k=1}^{i-1} (n + 1 - i)$$

$$(I.10) \quad = \sum_{i=2}^n \left( (n + 1 - i)(i - 1) \right)$$

$$= \frac{n^3}{6} - \frac{n}{6}$$

## Alternativa I.1(C)

Multiplicações ou subtrações

$$(I.11) \quad \sum_{k=1}^{n-1} \sum_{i=k+1}^n \sum_{j=i}^n (1) = \sum_{k=1}^{n-1} \sum_{i=k+1}^n (n + 1 - i)$$

$$(I.12) \quad = \sum_{k=1}^{n-1} \left( \frac{(k - n)(k - n - 1)}{2} \right)$$

$$= \frac{n^3}{6} - \frac{n}{6}$$

Contabilizando-se agora as operações de divisão, temos:

## Alternativa I.1(A)

Divisões

$$(I.13) \quad \sum_{i=2}^n \sum_{k=1}^{i-1} (1) = \sum_{i=2}^n (i - 1) = \frac{n^2}{2} - \frac{n}{2}$$

## Alternativa I.1(B)

Divisões

$$(I.14) \quad \sum_{i=2}^n \sum_{k=1}^{i-1} (1) = \sum_{i=2}^n (i - 1) = \frac{n^2}{2} - \frac{n}{2}$$

## Alternativa I.1(C)

Divisões

$$(I.15) \quad \sum_{k=1}^{n-1} \sum_{i=k+1}^n (1) = \sum_{k=1}^{n-1} (n - k) = \frac{n^2}{2} - \frac{n}{2}$$

Desta forma, uma contabilização para o número total de operações do processo de eliminação pode ser expresso por:

<b>Eliminação (Densa)</b>		<u>Operações de ponto flutuante</u>
(L.16)	Multiplicações	$1/6 n^3 - 1/6 n$
(L.17)	Subtrações	$1/6 n^3 - 1/6 n$
(L.18)	Divisões	$1/2 n^2 - 1/2 n$
(L.19)	<u>Total</u>	$1/3 n^3 + 1/2 n^2 - 5/6 n$
		$O(n^3)$ operações

A título de comparação, apresentamos a seguir, o número total de operações dispendidas nas 3 fases processo de retro-substituição:

<b>Retro-Substituição (Densa)</b>		<u>Operações de ponto flutuante</u>
(L.20)	Multiplicações	$n^2 - n$
(L.21)	Subtrações	$n^2 - n$
(L.22)	Divisões	$n$
(L.23)	<u>Total</u>	$2 n^2 - n$
		$O(n^2)$ operações

Percebe-se portanto, que no caso denso, a etapa de eliminação é preponderante sobre a de retro-solução, por pelo menos uma ordem de grandeza, fato este que como será observado mais adiante, nem sempre acontece no caso esparso.

Segue-se agora uma contabilização do espaço total de armazenamento da porção triangular superior correspondente a matriz de fatores U, visto ser este compartilhado com a matriz original, reescrevendo-se ao longo de cada etapa os valores definitivos por sobre os de A originais.

<b>Matriz de fatores U (Densa)</b>		<u>Espaço de armazenamento</u>
(L.24)	Porção Triangular	$1/2 n^2 - 1/2 n$
(L.25)	Diagonal	$n$
(L.26)	<u>Total</u>	$1/2 n^2 + 1/2 n$
		$O(n^2)$ posições memória

Passaremos agora a considerar uma análise do volume de recursos computacionais dispendidos por um procedimento tipicamente esparso, concentrando-se na alternativa L1(C), a fim de se estimar tais valores.

Nesta alternativa, percebe-se que o *loop* mais interno, corresponde a "propagação" da estrutura da  $k$ 'ezima linha por toda a porção triangular superior, da sub-matriz definida a partir  $a_{k+1,k+1}$  até  $a_{n,n}$ .

Acontece que por simetria, e por se tratar de um procedimento de eliminação esparsa, apenas alguns coeficientes na  $k$ 'ezima coluna abaixo da diagonal, precisarão ser eliminados (visto já serem nulos os demais coeficientes).

Os elementos que precisarão sofrer eliminação, são aqueles cujas linhas correspondem exatamente às colunas dos elementos distintos de zero na porção à direita da diagonal na  $k$ 'ezima linha, como nos mostra a fig\_1(3).

O número de coeficientes abaixo da diagonal na coluna  $k$ , que sofrerão eliminação, tendo em vista a observação do parágrafo anterior, e seguindo a notação introduzida nesta seção, pode ser expresso por  $NonzRow(k,U)$ .

Como o processo de atualização das linhas subseqüentes a linha corrente  $k$ , só ocorre na região confinada a porção triangular superior da matriz (incluindo-se neste caso os elementos diagonais), segue-se que o número de elementos efetivamente atualizados na etapa  $k$  do processo é dado por  $(NonzRow(k,U)^2 - NonzRow(k,U)) / 2 + NonzRow(k,U)$  com a primeira parcela correspondendo aos elementos da porção triangular e a segunda, aos elementos diagonais atualizados.

Formalmente portanto, podemos expressar:

Caso Esparsa

Multiplicações ou subtrações

$$(I.27) \quad \sum_{k=1}^{n-1} \left[ \frac{NonzRow(k,U)^2 - NonzRow(k,U)}{2} + NonzRow(k,U) \right]$$

$$= \sum_{k=1}^{n-1} \left[ \frac{NonzRow(k,U) ( NonzRow(k,U) + 1 )}{2} \right]$$

Contabilizando-se as operações de divisão, temos:

Caso Esparsa

Divisões

$$(I.28) \quad \sum_{k=1}^{n-1} \left[ NonzRow(k,U) \right]$$



Deste modo, a contabilização total seguindo a notação introduzida, pode ser expressa por:

**Eliminação (Esparsa)**

Operações de ponto flutuante

$$(L.29) \quad N_{flop}(k,U) \quad NonzRow(k,U)^2 + 2 NonzRow(k,U)$$

$$(L.30) \quad TotFlop(U) \quad \sum_{k=1}^{n-1} \left[ NonzRow(k,U)^2 + 2 NonzRow(k,U) \right]$$

Para o processo de retro-substituição esparsa, a título de comparação, apresentam-se os resultados:

**Retro-Substituição (Esparsa)**

Operações de ponto flutuante

$$(L.31) \quad \text{Multiplicações} \quad 2 \sum_{k=1}^{n-1} NonzRow(k,U)$$

$$(L.32) \quad \text{Subtrações} \quad 2 \sum_{k=1}^{n-1} NonzRow(k,U)$$

$$(L.33) \quad \text{Divisões} \quad n$$

$$(L.34) \quad \underline{\text{Total}} \quad \sum_{k=1}^{n-1} \left[ 4 NonzRow(k,U) + 1 \right] + 1$$

Resta agora se considerar o espaço de armazenamento mínimo necessário numa abordagem tipicamente esparsa. (O espaço em uma implementação real, inevitavelmente acaba sendo superior a cota fornecida, e um comentário a respeito no final desta seção se faz necessário, visto que ao longo de todo este trabalho, o que se almejará, serão métodos que se aproximem ao máximo destas cotas teóricas inferiores).

No caso de uma representação esparsa, apenas os elementos não nulos da porção triangular superior (incluindo a diagonal) da matriz original e da matriz fatores resultantes, necessitam ser armazenados.

Como comentado anteriormente, uma vez que se assume que os fatores de  $U$  são reescritos por sobre os valores originais de  $A$ , a contabilização do espaço total, leva em conta apenas a estrutura de elementos não nulos da matriz  $U$ , que pela natureza aditiva do processo de eliminação, seguramente inclui a estrutura da matriz original  $A$  como sub-conjunto.

<b>Matriz de fatores U (Esparsa)</b>	<b><u>Espaço de armazenamento</u></b>
(L35) Porção Triangular	$\sum_{i=1}^n \text{NonzRow}(i,U)$
(L36) Diagonal	$n$
(L37) <u>Total</u>	$\text{TotNonz}(U)$ $= \sum_{i=1}^n \text{NonzRow}(i,U) + n$

De posse de todas as estimativas apresentadas, seguem-se alguns comentários de modo a destacar algumas "filigranas" escondidas entre os valores apresentados, e que se mostram significativas no caso esparsa.

Para tal, vamos mais uma vez reinterpretar o conceito e a definição de "esparsidade" de uma matriz, introduzindo algumas definições complementares:

**Definição I.2(12) Matriz Esparsa**

*Define-se uma matriz como esparsa, caso o número total de seus elementos distintos de zero seja da ordem  $O(n)$ .*

**Definição I.2(13) Matriz Esparsa**

*Define-se uma matriz como esparsa, caso o número de seus elementos distintos de zero em cada linha seja assumido como uma constante  $\tau \in \mathbb{N}$  (pequena em comparação com a dimensão  $n$ ).*

**Definição I.2(14) Matriz Esparsa**

*Define-se uma matriz como esparsa, caso o número total de seus elementos distintos de zero seja da forma  $n^{1+\gamma}$  com  $\gamma \in \mathbb{R}$  e limitado ao intervalo  $0 \leq \gamma < 1$ .*

Em face das novas definições, apresentam-se as estimativas de volume de cálculo e armazenamento, obtidas em função do número de elementos não nulos de cada linha da matriz de fatores.

Inicialmente são exibidos os valores "médios" de  $\text{NonzRow}(i,U)$  para cada uma das definições de matriz esparsa consideradas (assumidos em alguns casos como constantes por simplificação), juntamente com o valor exato para o caso de matrizes densas (a título de comparação).

	$NonzRow(i,U)$	<u>Densidade</u>	<u>Def. Esparsidade</u>
(L38)	$\rho n$	$\rho$	L1(3)
(L39)	$O(1)$	$1/n$	L2(12)
(L40)	$\tau$	$\tau/n$	L2(13)
(L41)	$n^\gamma$	$n^\gamma/n$	L2(14)
(L42)	$n-i$	100%	Matriz densa

De posse destes valores, e substituindo-se em (L29), (L30), (L34) e (L37), obtemos:

	$Nflop(i,U)$	$TotFlop(U)$	<i>Retro-Sub.</i>	<u>Def.</u>
(L43)	$\rho^2 n^2$	$\rho^2 n^3$	$4 \rho^2 n^2$	L1(3)
(L44)	$O(1)$	$O(n)$	$O(n)$	L2(12)
(L45)	$\tau^2$	$\tau^2 n$	$4 \tau n$	L2(13)
(L46)	$n^{2\gamma}$	$n^{1+2\gamma}$	$4 n^{1+\gamma}$	L2(14)
(L47)	$(n-i)(n-i+2)$	$n^3/3 + n^2/2$	$2 n^2 - n$	Densa

	<u>Espaço Armazenamento</u>	<u>Def. Esparsidade</u>
(L48)	$\rho n^2$	L1(3)
(L49)	$O(n)$	L2(12)
(L50)	$\tau n$	L2(13)
(L51)	$n^{1+\gamma}$	L2(14)
(L52)	$n^2/2 + n/2$	Matriz densa

O que se pode observar é que pela maioria dos critérios de esparsidade, o volume de operações da fase de eliminação é praticamente da mesma ordem de grandeza que o da fase de retro-substituição, um fato que não ocorria no caso denso.

Outro item interessante, é que o primeiro critério de esparsidade introduzido neste trabalho L1(3), e que a princípio parecia ser o mais intuitivo de todos, por envolver apenas o conceito de densidade de matrizes, na verdade é um critério questionável, sob o ponto de vista de caracterizar matrizes realmente esparsas. (Justamente por este fato, é que na sua definição formal, não se fixou a princípio, nenhuma cota para a densidade tolerada, deixando-a passível de alteração em função da dimensão do problema).

O que se nota, é que a densidade "tolerável" de uma

matriz "realmente esparsa", deve ser proporcionalmente reduzida para o caso de problemas de dimensão muito elevada, pois de outro modo, o volume total de cálculo e o espaço de armazenamento poderá vir a ser tal, que inviabilize a solução mesmo nos equipamentos de mais grande porte disponíveis nos dias atuais.

Outra observação, é que o volume de cálculo, e o espaço de armazenamento, no caso de matrizes realmente esparsas, ser praticamente da mesma ordem de grandeza, e na maioria dos casos, da ordem apenas de  $n$  (seguramente a cota mais inferior possível em qualquer abordagem esparsa para o problema).

● *Ou seja, em esparsidade, qualquer estrutura de dados ou metodologia de implementação que venha a requerer espaço de armazenamento ou volume de cálculo de ordem superior a  $n$ , estará se distanciando cada vez da máxima eficiência computacional, não explorando portanto plenamente as características esparsas do problema a ser solucionado.*

Isso leva a uma constatação ainda mais "implacável", pelo fato da grande totalidade dos algoritmos desenvolvidos até hoje, nos vários ramos da computação científica, requererem, (mesmo em suas melhores implementações), volume de operações superior a  $O(n)$  em muitos dos casos, implicando em que sequer se possa cogitar em aplicá-los à alguma das etapas do processo de solução.

Como exemplo típico pode-se citar o problema de se ordenar os elementos de um conjunto de vetores, e que será apresentado no capítulo II, em que a solução prática para o mesmo, advém da dupla aplicação de um algoritmo para transposição de matrizes esparsas [A5], e não por repetidas aplicações de um algoritmo de ordenamento como o *QuickSort* [K4] por exemplo.

Outra constatação que se pode chegar a partir do fato do volume total de cálculo para o caso de matrizes significativamente esparsas (como é o caso de algumas das matrizes de potência, em que o número de elementos não nulos por cada linha poder ser para fins práticos, considerado como constante, e da ordem de uns 3 a 4 elementos por exemplo), é que qualquer sorte de *overheads* computacionais adicionais, passam a ser extremamente significativos no desempenho final

do processo de eliminação, visto serem executadas um número muito baixo de operações realmente necessárias para a obtenção da solução do problema, ficando a cargo de operações como acesso indireto a memória, inicialização e incremento de loops, etc... uma boa parcela do tempo efetivo de computação, que poderia ser melhor aproveitado.

Em termos do volume total de operações aritméticas, as 3 alternativas de acesso e geração da matriz de fatores, para o caso genérico, apresentadas na seção I.1 dispendem um volume idêntico de operações.

Porem, é interessante se perguntar, e se observar, se ao longo de cada etapa do processo de eliminação isoladamente, tal volume de cálculo se mantém igualmente distribuído nas 3 alternativas consideradas, ou se ao longo do desenrolar do processo, algumas das estratégias tendem a apresentar alguma redução ou acrescimento ao volume de cálculo para se completar a *i*'ezima etapa da eliminação.

Para tal, será preciso lançar mão novamente, das expressões I.8, I.11 e I.13, cujas expressões dentro de cada somatório, espelham o volume de operações aritméticas efetuado na *i*'ezima (ou *k*'ezima) etapa do processo de eliminação de coeficientes de uma matriz densa de dimensão *n*.

Cabe ressaltar, que no caso da alternativa I.1(C), para fins de padronização, tomou-se a liberdade de expressar em termos da variável *i*, a parcela correspondente ao volume de cálculo, que originalmente estava expressa em termos de *k* no somatório original.

Alternativa I.1(A)                      Volume de cálculo      Etapa *i* (Densa)

$$(I.53) \quad N_{flop}(i,U) = \frac{i(i-1)}{2}$$

Alternativa I.1(B)                      Volume de cálculo      Etapa *i* (Densa)

$$(I.54) \quad N_{flop}(i,U) = (n+1-i)(i-1)$$

Alternativa I.1(C)                      Volume de cálculo      Etapa *i* (Densa)

$$(I.55) \quad N_{flop}(i,U) = \frac{(i-n)(i-n-1)}{2}$$

Uma análise gráfica do comportamento das 3 funções mostra-se interessante, sendo exibida na fig\_I(4) um pouco mais adiante.

O que se percebe é que a alternativa I.1(B) é a mais "balanceada" de todas, em termos de efetuar um volume mais próximo de uma taxa constante, enquanto que as duas outras alternativas, apresentam um comportamento "explosivo" em uma das etapas extremas do processo de eliminação.

Confrontando-se os gráficos, com as fig\_I(1), I(2) e I(3), percebe-se claramente o porque deste comportamento:

Na atualização I.1(A), a medida que o processamento avança, o volume de cálculo tende a aumentar, pois o número de elementos das colunas sendo geradas aumenta, e com ele, o número de outras linhas que necessitarão ser subtraídas de modo a se gerar a coluna corrente.

De modo oposto, na alternativa I.1(C), o tamanho das sub-matrizes vai sendo reduzido, e juntamente com ele, o volume de operações necessárias para a conclusão da *i*'ésima etapa.

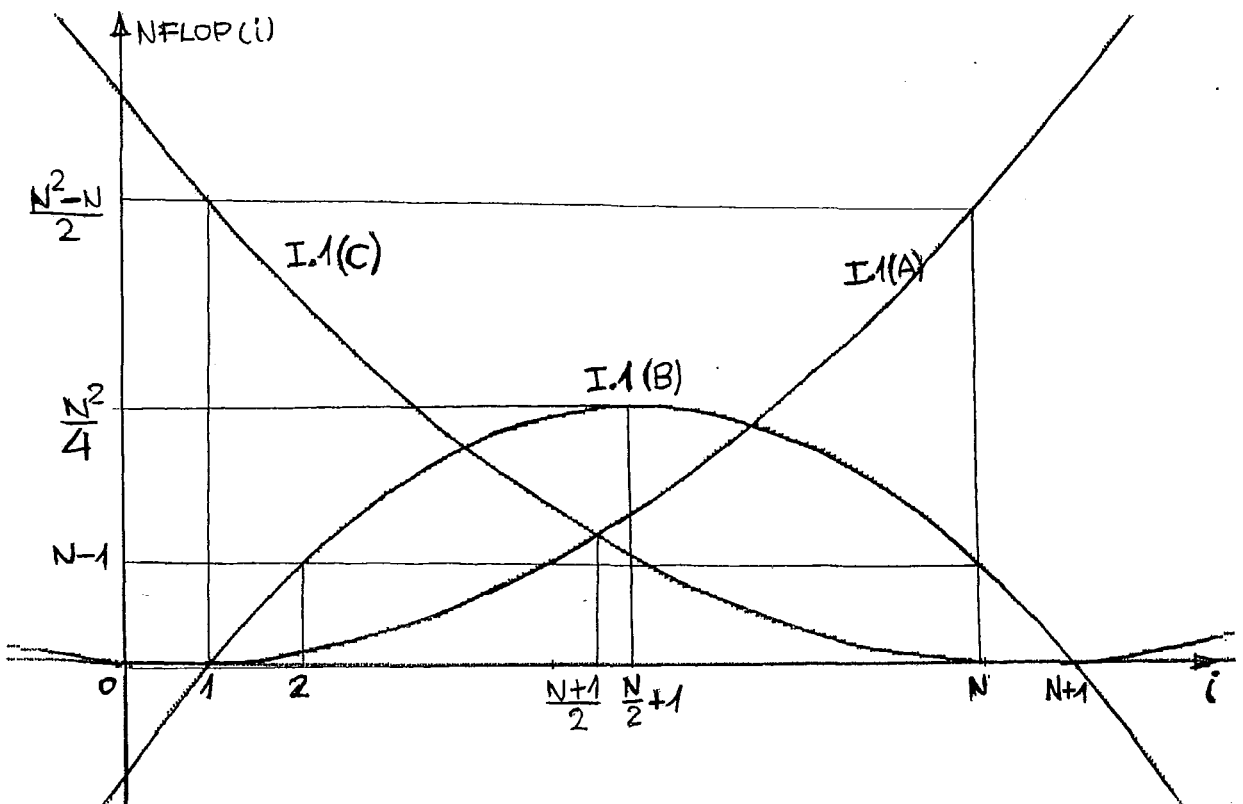
Na alternativa I.1(B), percebe-se que embora a região sendo acessada durante o processo de geração da *i*'ésima linha varie em função de *i*, o produto do número de linhas contribuintes *versus* a largura da contribuição da "área" abrangida tende a se manter próximo de um valor mais ou menos uniforme ao longo de todo o processo.

Um comentário que se lança aqui, mas que só poderá ser melhor apreciado na seção III.4, é que na hipótese de se adotar estratégias "híbridas" para o processo de geração dos elementos (com a aplicação de alternativas distintas para regiões selecionadas na matriz de fatores), o gráfico da fig\_I(4) fornece um critério inicial para se determinar o ponto de "corte" entre as alternativas, visando-se "balancear" ao máximo o volume de cálculo ao longo de todo o processo. (Um exemplo, pode ser tomando-se a alternativa I.1(A) da etapa 1 até  $(n+1)/2$ , aplicando-se a seguir a alternativa I.1(B), para se completar a geração da porção restante das primeiras  $(n+1)/2$  linhas acima da referida linha em questão, e finalmente, lançando-se mão da alternativa I.1(C) para se gerar as demais linhas restantes, até o final do processo).

• No caso esparsa, o comportamento das diferentes alternativas não pode ser modelado exatamente pelas curvas da fig\_I(4), uma vez que a etapa de reordenamento ótimo das equações (visando a redução de fill-in's), invariavelmente tende a levar o acúmulo do esforço computacional para as etapas finais do processo, de modo a se evitar ao máximo a introdução prematura de elementos de fill-in.

Tal comportamento será visto em detalhes, e comentado na seção II.2 onde se apresentam as técnicas mais usualmente empregadas na literatura, visando-se a redução da criação dos novos elementos de fill-in.

O que se percebe facilmente, é que o "ponto de corte" simplesmente acabará sendo "deslocado" para mais adiante, até que o esforço computacional dispendido com a alternativa I.1(A) de geração dos fatores por colunas, comece a apresentar um volume computacional superior ao das demais alternativas.



fig\_I(4) - Volume de cálculo (caso denso)

Em se tratando de "ponto de corte", poderia-se se cogitar um outro critério, em função do comentário anterior de que no caso esparsa (após o reordenamento), o final do processo vir a conter as linhas mais "carregadas" de elementos, decorrentes do processo cumulativo da propagação dos fill-in's, e que a partir de uma certa etapa do processo, podem ser consideradas para fins práticos como "densas" (ou "quase densas" pela definição L1(4)).

Neste caso, valeria a pena se questionar, a partir de que ponto tal "enchimento" das linhas inevitavelmente ocorre, e qual a dimensão da sub-matriz densa restante, em comparação com a dimensão total do problema original.

Tal questão no entanto não é passível de uma solução analítica formal, visto ser o ponto de "enchimento", uma característica um tanto quanto "não determinística", em função da disposição estrutural de cada matriz, e do reordenamento ótimo aplicado.

Na literatura encontram-se alguns trabalhos tratando desta questão IG71, ID261, que no entanto não foram bem sucedidos, ou não tiveram aplicabilidade num caso mais geral, (encontrando-se estimativas apenas para determinadas classes de matrizes em particular).

● *Assim, uma outra pergunta alternativa que se pode fazer, e esta passível de ataque e solução analítica, como veremos logo adiante, é determinar o "ponto de corte" onde o volume de operações dispendido com uma fatoração tipicamente esparsa, se iguala ao volume dispendido na fatoração de uma sub-matriz completamente densa.*

Para a determinação do ponto onde o volume de cálculo se iguala, lançaremos mão dos resultados já apresentados nesta seção, em particular as relações de (L43) até (L47), para cada uma das diferentes definições de esparsidade consideradas.

Denotando-se por  $n'$  a dimensão da matriz densa (que se deseja determinar), e por  $n$  a dimensão da matriz esparsa original, cujo volume de cálculo para a eliminação deverá ser igual ao da matriz densa, chegamos aos seguintes resultados para cada um dos critérios de esparsidade:



	Dimensão $n'$ (densa)	Def. <u>Esparsidade</u>
(L56)	$\rho^{2/3} n$	L1(3)
(L57)	$n^{1/3}$	L2(12)
(L58)	$\tau^{2/3} n^{1/3}$	L2(13)
(L59)	$n^{2\gamma/3} n^{1/3}$	L2(14)

Percebe-se nitidamente uma dependência em termos de  $n^{1/3}$  em praticamente todos os casos, a exceção de (L56) que como já se havia comentado anteriormente, não é um bom critério para medida do grau de esparsidade de uma matriz, a menos que se tome  $\rho$  como alguma função decrescente em termos de  $n$ , como por exemplo  $\rho_n \equiv 1/n$ , caso em que a expressão (L56) acaba recaído em (L57).

Outra dependência que não é explicitamente de  $n^{1/3}$  mas que em certas circunstâncias pode ser considerada como tal, é a da relação (L59), onde o termo  $n^{2\gamma/3}$  pode ser encarado como praticamente "constante" (caso a medida característica  $\gamma$  seja suficientemente próxima de zero, ou relativamente pequena se, comparada a dimensão do problema, para que se possa desprezar a contribuição do termo considerado).

O que se deve notar é que o valor estimado de  $n'$  é na verdade uma cota inferior para o tamanho de uma matriz densa, cujo volume de cálculo no processo de eliminação, se iguale ou supere o esforço total de fatoração de uma matriz "realmente esparsa" (tal que não contenha sub-porções densas de tamanho significativo).

Ou seja na prática, é possível se encontrar matrizes cuja porção densa seja maior do que o valor considerado da ordem de  $n^{1/3}$ , caso este em que seguramente, o processamento denso ao final da matriz, será dominante em termos de esforço computacional, se comparado a porção tipicamente esparsa eliminada nas fases iniciais.

O que o valor  $n^{1/3}$  dita, é na verdade um valor limite máximo que poderá admitir a sub-porção densa de uma matriz, de modo a que o processamento desta porção, não seja o mais preponderante no processo de eliminação como um todo.

Matrizes de potência na maioria das vezes atendem a este requisito, com uma sub-porção densa de algumas poucas linhas ao final do processo, em matrizes com dimensões da ordem de

milhares de linhas.

Nestes casos, a porção densa não é preponderante e o que conta é a eficiência dos códigos voltados para o processamento esparso.

Cabe dizer, que matrizes com características similares as de potência citadas, ou seja, com um número de elementos por linha extremamente baixo (3 a 4 elementos por exemplo), e dimensões elevadas, constituem uma das classes de matrizes em que toda sorte de aprimoramento da eficiência final do código se faz notar com destaque.

Ao longo deste texto, veremos abordagens que alcançam uma maior eficiência, tanto para porções significativamente esparsas, como nas seções III.1, III.2 e IV.4, bem como para porções densas em III.1, III.3 e III.4.

### 1.3 Evolução histórica dos métodos de esparsidade

Esta seção visa apenas a situar o leitor em face à constante evolução das técnicas de esparsidade ao longo das últimas duas décadas, destinando-se especificamente aos interessados em ter uma breve noção do avanço cronológico sofrido pelos métodos de solução, em função da introdução de novas arquiteturas, descoberta de novos algoritmos ou o estabelecimento de técnicas de implementação mais eficazes.

A área de resolução de sistemas esparsos é certamente uma das mais difundidas por praticamente todos os ramos da engenharia, englobando aplicações desde: controle de tráfego aéreo, astrofísica, engenharia química, simulação de circuitos, demografia, modelagem econômica, projeto de reatores nucleares, fluxo de potência ótimo, modelagem estocástica, espalhamento acústico, modelagem de reservatórios de petróleo, solução de equações diferenciais ordinárias e parciais, problemas de Navier-Stokes, oceanografia, problemas estruturais de engenharia civil com modelagem por malhas, elementos finitos, redes de sistemas de potência, problemas estruturais decorrentes do projeto de estruturas da indústria naval, aeroespacial e automobilística, programação matemática (linear e não linear), e resolução de problemas de mínimos quadrados em otimização e estatística, entre outras.

A primeira aplicação das técnicas de esparsidade para a solução de grandes sistemas lineares, remonta à década de 50, se devendo aos esforços de pesquisa nas áreas de Elementos Finitos e Programação Linear. Técnicas de "redução de banda" e o critério de Markowitz [M3] para o pivoteamento pertencem a esta fase em particular.

Para os problemas tipicamente encontrados na área de potência, as técnicas de redução de banda não se mostraram adequadas. A solução de sistemas lineares esparsos nesta época invariavelmente implicava na utilização dos métodos iterativos clássicos, [T15], [V6], [Y3] com uma taxa de convergência na maioria das vezes baixa.

No final da década de 60, após a publicação do

revolucionário artigo de Tinney & Walker [T2], a solução de problemas esparsos mediante o uso de métodos diretos de solução passou a ser uma realidade, graças ao bom desempenho alcançado pela heurística de "menor grau", conhecida como o critério # 2 de Tinney (ou simplesmente T2).

Outros critérios mais onerosos que o segundo proposto por Tinney foram experimentados, como por exemplo o terceiro critério, baseado na minimização local do número de *fill-ins* (eficientemente implementado em [G1]).

O critério T2 (*minimum degree*) consagrou-se no entanto para as aplicações encontradas na área de potência, podendo ser encarado como uma particularização do critério de Markowitz, para o caso de matrizes simétricas. (Com um excelente *survey* da enorme evolução pela qual passou esse critério de ordenamento, apresentado em [G2]).

Paralelamente a evolução das técnicas de ordenamento ótimo para a esparsidade, os trabalhos pioneiros de Tinney [T2] e de Gustavson [G14], [G15], tornaram realidade a solução de grandes sistemas lineares (até então com milhares de equações).

Do início dos anos 70 remontam dois conceitos fundamentais e aplicados desde então ao processamento esparsos, a saber: a utilização de vetores de trabalho [G15] e de esquemas de pré-processamento simbólico [G14]. O primeiro destes esquemas acarretando *overheads* em termos de tempo de execução, enquanto que o segundo invariavelmente introduzindo *overheads* nem sempre toleráveis no espaço de armazenamento (em favor de uma maior eficiência final, a nível de tempos de execução).

Em esparsidade a dicotomia "tempo x espaço" é uma presença constante, e que depende fundamentalmente do tipo de estratégia de solução adotada (métodos diretos, iterativos ou híbridos) e dos recursos disponíveis para a solução (memória, tempo de CPU e tipo de arquitetura). Em muitas situações um ponto de balanceamento ótimo (nem sempre fácil ou possível de ser alcançado) encontra-se numa combinação do melhor entre tais mundos.

Assim, a utilização de esquemas híbridos, aproveitando-se de técnicas densas e esparsas, resultou em abordagens nas quais dois ou mais níveis de processamento são

considerados de acordo com a densidade de cada sub-matriz encontrada durante o processo de solução como em ID11, IA81 e ID351.

Conceitos como os de "tipos de variabilidade" também remontam a esta época em particular, enquanto as idéias de processamento por blocos introduzidas por Hachtel IH11, faziam sua primeira aparição no cenário das técnicas de esparsidade.

No início dos anos 80, implementações esparsas extensivamente testadas como as de Waterloo IG101, IG51, Yale IE21 e Harwell IH21, ID121 tornaram-se realidade, tomando como base na maioria dos casos as estruturas de dados já então padronizadas IE11, ID101, IG201 (fundamentadas no uso de vetores de trabalho ou de acumulação) e critérios de ordenamento como o de Tinney # 2.

Com o surgimento das primeiras arquiteturas vetoriais no início da década de 80 IH111, as estratégias para a solução esparsa eficiente de problemas de grande porte, acabaram tendo de ser completamente reformuladas, visto ser na época, substancialmente inferior o desempenho de códigos baseados em acessos indiretos a memória, nos quais a quase totalidade dos métodos esparsos desenvolvidos até então se baseavam (em face ao bom desempenho obtido pelos códigos voltados para o tratamento de matrizes densas, e que viabilizavam a exploração das características vetoriais das novas arquiteturas).

A partir desta "restrição" computacional, os métodos "Multi-Frontais" ID401, ID231, IL41, (que na verdade podem ser encarados como uma extensão de uma metodologia sólidamente já consagrada na solução dos grandes problemas da área de análise estrutural, como o caso dos métodos "Frontais" II11), acabaram por se firmar definitivamente como a melhor alternativa para implementação nos supercomputadores da época, por alcançar uma maior eficiência vetorial dos códigos de eliminação, mediante a subdivisão deste processo, em etapas cuja exploração de matrizes com características iminentemente densas passassem a ser dominantes sobre as demais etapas do processo.

A necessidade de códigos de fatoração extremamente eficientes continuava a existir em muitas áreas de aplicação,

especialmente naquelas em que apenas poucos elementos do sistema sofriam alterações em valor numérico entre soluções sucessivas de sistemas com matrizes de mesma estrutura (um caso típico na área de otimização).

Destá necessidade surgiram os métodos de "refatoração parcial" baseados em *sparse vectors* [B8], [G27] e [G28], [S8] que continuam a despertar interesse até hoje, especialmente na fase de ordenamento, em arquiteturas do tipo paralelo.

Assim, critérios baseados na minimização da altura da "árvore de caminhos de eliminação" [G27], [G28], passaram a ser considerados como novos "critérios de desempate" para a heurística de "menor grau".

Chegamos ao final da década de 80, e se forem considerados o tamanho dos problemas solucionáveis e o tempo de solução dispendido, comparando-os com os de duas ou três décadas atrás, veremos que o avanço foi gigantesco, tanto do ponto de vista do *hardware*, como do *software*.

Para o final dos anos 80 ainda estavam reservados dois últimos avanços, e que exploram características do *hardware* encontradas nas arquiteturas mais avançadas atualmente disponíveis.

O "redescobrimento" da abordagem via "supernodes" [A11], [H5] (onde blocos de elementos não nulos contíguos são tratados como uma única entidade na estrutura de representação adotada) veio a ser merecedor de um importante prêmio de supercomputação em 1988 [B10] (com uma implementação vetorial/paralela eficiente baseada na exploração deste conceito).

Finalmente uma abordagem importante proposta já no início desta década, é a da representação particionada para a inversa da matriz de fatores [A2], [A6]. A característica inovadora deste método reside no fato de permitir a solução de múltiplos sistemas (com diferentes lados direitos) mediante a utilização de produtos do tipo matriz x vetor (no lugar do processo de retro-substituição adotado na abordagem convencional e que em arquiteturas paralelas não pode ser tão bem explorado quanto as operações do tipo produto matricial).

Paralelamente aos avanços na fase numérica de fatoração e solução, a evolução e aplicação recente das técnicas de "pré-processamento simbólico" [G3], [G4] especialmente em

implementações de Programação Linear baseadas nos Métodos de Pontos Interiores [A1], [M1] além do trabalho pioneiro de Gustavson na geração de códigos *loop-free* [G14] na década de 70, mostram que a busca por códigos cada vez mais eficientes é uma constante na história das técnicas de esparsidade.

Cabe mencionar aqui, o crescente avanço obtido, ao se passar de uma abordagem "clássica" baseada em listas encadeadas e criação "dinâmica" de *fill-in's*, para a abordagem "tradicional" dos dias atuais, baseada em fases distintas de processamento, com a geração prévia da estrutura de fatores resultantes e a adoção de uma representação "supernodal" seqüencial durante a fase numérica de solução.

Recentes conquistas na fase de fatoração simbólica apontam cada vez mais para o uso de estruturas "quase ótimas" [G3] (do ponto de vista da quantidade de informação necessária para se caracterizar uma matriz), baseadas no uso da "árvore de caminhos de eliminação" [L1], chegando-se nos dias atuais a descoberta de "esquemas compactos" de armazenamento [L10] que em alguns casos superam em pelo menos uma ordem de grandeza a economia obtida por um esquema de compressão "tradicional" como o de Shermann [S6], [D6] (utilizado até então com sucesso nas últimas duas décadas).

A preocupação com a minimização de toda a sorte de *overheads* como paginações [L1], utilização eficiente de memórias *cache* [B9], e o uso de políticas eficazes de gerenciamento das informações armazenadas em meio secundário [G11], [L13], [L14], mostra-se cada vez mais determinante no sucesso das novas implementações, bem como a preocupação com a redução do espaço total de armazenamento via o uso de estruturas dinâmicas como em [B1], [B2].

A fatoração esparsa eficiente nas arquiteturas vetoriais ou paralelas existentes [A12], [H5], constitui um dos desafios desta década, servindo de base para a implementação eficaz de uma grande parcela dos métodos de otimização Não Linear em geral [D47], [Z5], [Z6].

Até mesmo a aplicação de antigas técnicas como a de "desenrolamento de *loops*" [D2] encontra terreno fértil na busca por menores *overheads* intrínsecos de processamento.

O problema da fatoração eficiente nas novas arquiteturas vem sendo merecedor por si só de uma atenção redobrada desde

meados da década de 80. Clássicos de análise numérica como os de Golub & Van Loan [G39] dedicam agora capítulos inteiros à fatoração nos diversos tipos de arquitetura existentes como em [D3] e [O3] por exemplo.

Para finalizar esta seção, como muito bem colocou Duff, (um dos mais renomados pesquisadores da área) em [D45], reafirmamos:

- *"O futuro aponta cada vez mais para o constante desenvolvimento e a plena utilização de novas técnicas de esparsidade, sendo portanto extremamente promissor não apenas nos dias atuais, como nas demais gerações do futuro próximo".*



## Capítulo II

## ABORDAGEM CONVENCIONAL

A evolução "tecnológica" da área de esparsidade nos últimos 20 anos foi incrível, como se pode notar na última seção do capítulo anterior.

Tal sucesso advém de sucessivos aprimoramentos introduzidos ao longo das últimas décadas, e que acabaram por constituir uma base extremamente sólida e coesa, para o desenvolvimento e adaptação de novas técnicas, nas mais diversas arquiteturas.

Neste capítulo, concentra-se numa abordagem (para os dias atuais), tida como "convencional", ou seja baseada em métodos de amplo conhecimento e aceitação pela literatura, e voltados para arquiteturas escalares tradicionais, ou seja monoprocessadas, com uma única instrução básica efetuada por vez, e baseadas no modelo original de *Von Neumann*.

Como pode ser notado no capítulo introdutório, a implementação eficiente do problema de resolução de sistemas lineares esparsos por métodos diretos, é de natureza intrinsecamente "modular", onde pelo menos 4 etapas básicas distintas podem ser apontadas:

- Reordenamento visando a redução de *fill-in's*
- Determinação da estrutura da matriz de fatores
- Fatoração numérica propriamente dita
- Retro-substituições triangulares

No presente capítulo serão apresentadas cada uma destas técnicas nas sucessivas seções ao longo do texto.

Um outro aspecto de vital importância em esparsidade, merecedor da atenção inicial deste capítulo, é o das estruturas de dados a serem adotadas para a representação das matrizes e estruturas auxiliares empregadas no processo de solução.

Vale lembrar, que de nada adianta se lançar mão de algoritmos teóricos de eficiência comprovada, se as estruturas de dados adotadas não forem as mais adequadas.

## II.1 Estruturas elementares de armazenamento

Esta seção apresenta algumas das principais estruturas de dados utilizadas no armazenamento de matrizes esparsas, reportando-se o leitor ao apêndice A para uma descrição complementar a esta seção (formalizando-se a notação a ser adotada na representação de matrizes e vetores auxiliares ao longo de todo o texto).

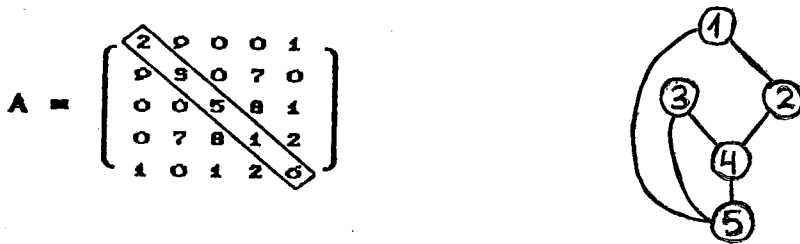
Apenas o caso da representação de matrizes simétricas (definidas positivas), será considerado, visto ser este o escopo de abordagem do presente trabalho. Em todas os casos nesta seção portanto, assume-se que apenas uma porção triangular (superior ou inferior) incluindo-se a diagonal, seja de fato explicitamente armazenada.

Uma propriedade inerente ao processo de eliminação Gaussiana esparsa, e que vem permitindo o contínuo avanço das técnicas de solução por métodos diretos desde [T2], é o fato de por meio de uma permutação adequada de linhas e colunas da matriz original, se conseguir que a matriz de fatores resultante, na maioria das vezes permaneça esparsa.

A densidade da matriz de fatores no entanto, acaba por ser inevitavelmente superior a da matriz original, pelo fato do processo de eliminação intrinsecamente tender a criar novos elementos não nulos (denominados *fill-in's*), em posições que originalmente não eram explicitamente consideradas.

● *A principal razão para se adotar uma estrutura esparsa de armazenamento para a matriz A do sistema original e para a matriz de fatores U associada, está no fato de se armazenarem apenas as informações referentes aos seus elementos não nulos, e que conforme visto na seção I.2, representa uma sensível economia em comparação com uma representação "densa" para o mesmo problema.*

Uma representação que explora plenamente a característica esparsa de uma matriz A, é considerá-la na forma de um grafo  $G_A$  (não orientado) como na fig\_II(1) onde cada aresta unindo os nós  $i$  e  $j$  indica a existência de elementos não nulos nas posições  $a_{ij}$  (e  $a_{ji}$  por simetria).



fig\_II(1) - Grafo associado a uma matriz

Assim em princípio, qualquer das estruturas normalmente utilizadas para a representação de grafos poderia ser adotada para a representação de uma matriz esparsa, como "listas de adjacências", ou de arestas na forma  $\{(i,j,a_{i,j})\}$  por exemplo.

Do ponto de vista de simplicidade, as listas de arestas são consideravelmente mais práticas e fáceis de se manipular do que as listas de adjacências, porém não são as mais adequadas para o tipo básico de operações encontradas durante o processo de fatoração, principalmente durante pesquisas inserções e remoções de elementos (efetuadas com frequência durante a fase inicial de reordenamento).

Assim, por sua "amenidade", este tipo de estrutura normalmente é adotado apenas na interface final com o usuário, trabalhando-se com uma representação interna por adjacências durante todo o restante do processo.

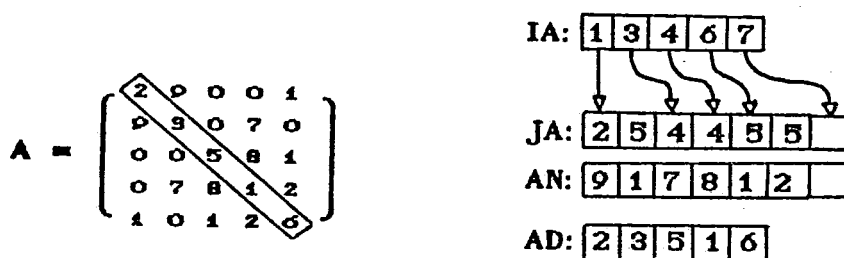
A representação por adjacências por sua vez, pode ser feita usando-se uma implementação por listas "encadeadas" ou por uma representação "seqüencial", detalhada a seguir.

O uso de listas encadeadas, embora mais flexível, não é necessário, quando já se dispõe da estrutura final da matriz de fatores resultantes ao final do processo de eliminação (obtida em alguma etapa prévia, mediante alguma das técnicas a serem apresentadas nas seções posteriores deste capítulo).

Utiliza-se portanto uma representação de adjacências por "listas encadeadas", apenas enquanto ainda não se conhece a estrutura da matriz de fatores resultante, visto que uma vez de posse desta estrutura, pode-se converter a representação "encadeada" numa "seqüencial" equivalente, ganhando-se desta forma em termos de espaço (com a liberação dos apontadores para o próximo elemento da lista) e em eficiência, pois na maioria das arquiteturas de computadores escalares convencionais como notado em [A8], um acesso "seqüencial" é

sempre mais barato (ou de mesmo custo) do que um acesso "randômico" (do tipo indireto à memória) e inevitável numa representação "encadeada" por lista de apontadores.

Um esquema tradicionalmente utilizado para a representação seqüencial por listas de adjacências para uma matriz simétrica (com o armazenamento da diagonal em separado) é apresentado na fig\_II(2) onde IA aponta para a posição inicial das listas de adjacências associadas à cada linha, JA denota a coluna correspondente e AN o valor numérico de cada elemento associado (com o armazenamento da diagonal, feito no vetor denso auxiliar AD).



fig\_II(2) - Representação seqüencial por adjacências

Explorando-se o fato de que a estrutura de elementos não nulos de cada linha da matriz original A é um sub-conjunto da estrutura da linha resultante na matriz de fatores U, (conforme será visto na seção II.3), um esquema seqüencial "segmentado" de armazenamento como o exemplificado na fig\_II(3) pode ser adotado no lugar do esquema "contíguo" apresentado na figura anterior.

A vantagem deste novo esquema como observado em [A8] está em se economizar uma parcela significativa do espaço que seria utilizado, caso se adotasse uma representação em separado para matriz original e para a de fatores resultante.

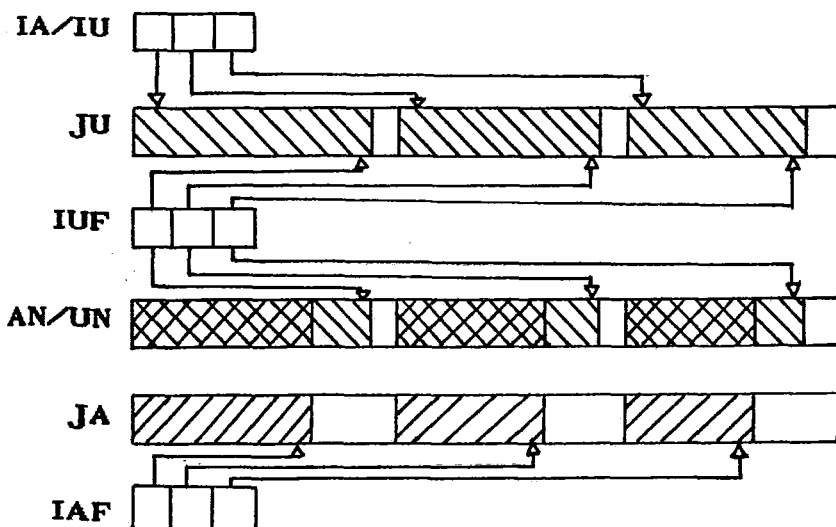
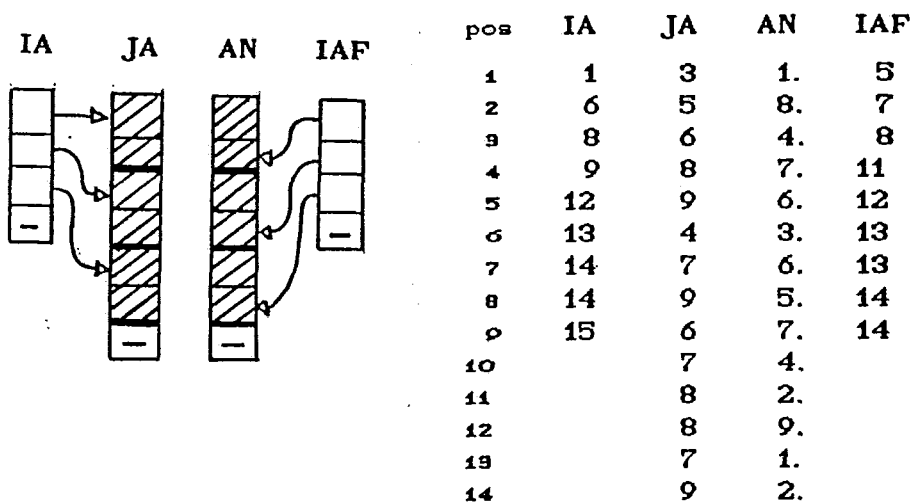
É assumido neste caso que a matriz original acabará irremediavelmente perdida ao final do processo, sendo paulatinamente reescrita pela matriz de fatores sendo gerada.

Tal fato pode ser explorado, pois uma vez gerados os elementos em uma dada linha da matriz de fatores, estes não mais dependerão dos elementos da matriz original, (vindo eventualmente apenas a alterar o valor de elementos em outras linhas subseqüentes, nas demais etapas do processo).

A única diferença com relação a estrutura sequencial "contígua" (anterior) é que se faz necessário agora a utilização de mais um conjunto de apontadores IAF, (pois o final de cada linha não mais se encontra contíguamente armazenado na posição anterior ao início da próxima).

Este *overhead* adicional de espaço é justificável, por ser apenas da ordem de  $n$ , em comparação com o *overhead* da ordem de  $TotNonz(A)$ , que inevitavelmente se teria com uma representação contígua para ambas as matrizes em separado.

$$\begin{array}{c}
 123456789 \\
 A = \begin{pmatrix} 0 & 1 & 84 & 70 \\ 7 & 9 & 6 & 5 \\ 2 & 6 & 742 & 5 \\ 6 & 1 & 9 & 0 \\ 5 & 1 & 8 & 82 \\ 8 & 8 & 8 & 4 \end{pmatrix} \\
 0870054921 \quad 123456789 \\
 0870054921 \quad 123456789
 \end{array}$$



fig\_II(3) - Representação sequencial "segmentada"

Uma outra estrutura auxiliar a ser utilizada com frequência em algumas das etapas apresentadas neste capítulo, é a estrutura de  $U^T$ , ou seja a estrutura de cada linha da transposta da matriz de fatores (correspondendo ao conjunto de índices das linhas contendo elementos não nulos, em cada coluna da matriz de fatores  $U$ ).

Tal estrutura, pode ser facilmente obtida, mediante a aplicação de um algoritmo de transposição esparsa [A5], [G16], apresentado na seção II.3.

Em algumas abordagens tradicionais, a utilização de uma representação explícita de estrutura da transposta de  $U$  é completamente evitada, gerando-se esta informação dinamicamente, durante a fase numérica de processamento, a medida que esta vai se mostrando necessária (como por exemplo para se determinar qual a próxima linha a ser subtraída da linha corrente em cada etapa do processo de eliminação).

No presente trabalho optou-se por lançar mão sempre que possível da representação explícita da matriz  $U^T$  no lugar da geração dinâmica desta informação, por ser simplesmente a forma mais eficiente (a nível de um menor *overhead* em tempo de execução), e a mais didática e flexível de todas, adicionando um *overhead* em espaço adicional, da ordem de  $TotNonz(U)$  posições inteiras de armazenamento, o que para matrizes realmente esparsas, aproxima-se bastante de  $O(n)$ .

Em algumas abordagens, como a da seção IV.3, a utilização de uma representação "implícita" para os elementos em cada linha da matriz transposta, gerando-os sequencialmente "em tempo real" à cada etapa, é simplesmente impossível de se aplicar, visto que para o reordenamento dinâmico de contribuições (a ser apresentado no capítulo IV), é preciso se conhecer toda a estrutura explícita de elementos da *i*-ésima linha da matriz transposta, para se poder levar adiante esta etapa do processo de eliminação.

Assim, como o leitor poderá perceber ao longo de todo este trabalho, sempre que possível, procurou-se privilegiar estruturas que venham a oferecer algum benefício em termos de redução de *overheads* em tempo de execução, em detrimento de estruturas cujo objetivo fosse voltado para um pólo completamente oposto, ou seja propiciando prioritariamente apenas uma redução dos *overheads* de espaço de armazenamento.

● *Assume-se portanto que estruturas com espaço total de armazenamento da ordem de  $n$  ou de  $TotNonz(U)$  sejam toleráveis, ao passo que estruturas da ordem de  $TotOper(U)$  não, por certamente esta segunda grandeza, tornar impraticável a solução de sistemas esparsos de médio a grande porte, na maioria das workstations e micro-computadores, com uma memória real tipicamente da ordem de apenas algumas dezenas de megabytes.*

Algumas estruturas a serem apresentadas, como as "listas simbólicas de endereços" na seção III.2 e as "listas compactas de códigos" na seção IV.4, no pior caso, podem vir a ocupar um espaço próximo ao de  $TotOper(U)$ , o que em princípio, pela filosofia de projeto anteriormente apresentada, poderiam ser sumariamente descartadas como estruturas auxiliares a se empregar no processo de eliminação esparsa.

Porem, o uso "controlado" de tais estruturas, ou seja, com a aplicação destas apenas para certas fases do processamento como um todo, com o confinamento de tais estruturas apenas em regiões onde o crescimento do volume de informação armazenado não é "explosivo", permite que se eleve ainda mais a eficiência final a nível de tempos de CPU, sem contudo se onerar de forma intolerável, o espaço final de armazenamento.

Vale a pena, a partir deste ponto, até o final desta seção, mencionar a título de complementação, outros esquemas consagrados pela literatura, e que podem ser empregados para se reduzir o espaço de armazenamento na representação da matriz de fatores resultante.

O primeiro destes esquemas é o de "compressão de Shermann" adotado com sucesso desde meados da década de 70, e que consiste em se reduzir o tamanho total do vetor  $JU$  (contendo as listas de colunas associadas a cada linha da matriz de fatores), baseando-se na observação empírica de que a estrutura das porções finais das últimas linhas de  $U$  possuem normalmente sub-conjuntos de índices idênticos (e que

portanto acabariam redundantemente armazenados num esquema do tipo "tradicional").

A solução portanto consiste em se lançar mão de mais um vetor auxiliar de apontadores IJU, e que indica a partir de que posição os índices de colunas associadas à uma dada linha na matriz de fatores podem ser obtidos a partir dos índices associados à representação de alguma das linhas prévias.

Com isso, especialmente na sub-porção final da matriz de fatores, consegue-se obter reduções significativas no espaço alocado para o vetor JU (contendo apenas conjuntos realmente distintos de índices de colunas necessários para a caracterização completa de toda a matriz U).

Cabe lembrar que o esquema de "Shermann", só considera o caso em que a coincidência nos índices de colunas ocorre apenas na porção final de cada linha. Se um esquema mais complexo de compressão (como por exemplo os baseados em *supernodes*, a serem vistos na seção III.3) for adotado, a taxa final de compressão pode vir a ser ainda mais elevada.

Na verdade, uma representação por *supernodes* em certo sentido é uma das mais econômicas e eficientes que se pode obter, sendo o principal benefício da sua aplicação não apenas a compressão de espaço, mas sim o aumento da eficiência computacional como um todo, por permitir um melhor aproveitamento da sequencialidade de armazenamento durante a fase numérica do processamento (como será visto no final do próximo capítulo).

As únicas estruturas adicionais capazes de rivalizar com a representação por *supernodes* são as baseadas na "árvore de caminhos de eliminação" [L11] (estrutura esta de vital aplicação em várias etapas do processamento esparsos, e que será vista em maiores detalhes nas seções II.3 e IV.2).

Neste caso, adotando-se "esquemas compactos" de representação, baseados na árvore de eliminação (cujo *overhead* adicional de espaço de armazenamento para sua representação é de apenas  $n$ ), pode-se chegar a ganhos apreciáveis em algumas classes particulares de matrizes como nos mostra [L10].

Finalmente não se deve esquecer de mencionar os esquemas "dinâmicos" via "Bordering" propostos em [B1], [B2] também apoiados na árvore de fatoração, e que simplesmente conseguem



eliminar completamente a necessidade de uma estrutura de representação para as colunas da matriz de fatores resultantes, mas cuja implementação acaba impondo *overheads* em termos de tempo de execução nem sempre compensadores, (a menos que a economia de espaço obtida seja muito superior a conseguida pelos demais meios de compressão usuais).

Passaremos agora a considerar o processo de reordenamento visando a redução de *fill-in's*, e que por consistir na simulação de todo o processo de eliminação, mediante operações sobre o grafo de representação da matriz associada ao sistema, inevitavelmente, acaba por se apoiar em novas estruturas de dados, especificamente voltadas para esta fase em particular, e que serão portanto apresentadas ao longo da próxima seção.

## II.2 Ordenamento visando a redução de fill-in

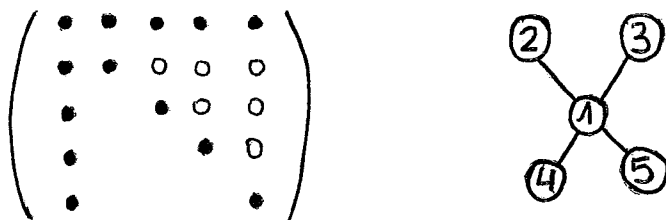
Como já foi comentado em seções anteriores, o processo de eliminação esparsa, tende a introduzir novos elementos não nulos a medida que vão se processando as eliminações a cada etapa. A forma encontrada para se tentar amenizar este esforço computacional extra, se dá mediante o reordenamento prévio das linhas e colunas da matriz original, visando, de alguma forma, à minimização da criação dos elementos de *fill-in*.

No caso de matrizes simétricas, a única forma de permutação de linhas e colunas possível, de modo a se manter a simetria, é efetuar-se permutações simultâneas de pares  $i, j$  de linhas e colunas com mesmo índice associado, o que corresponderia a se tomar apenas elementos pivô situados na diagonal, (permutando-se desta forma uma dada linha  $i$  por  $j$ , bem como a  $i$ 'ezima coluna pela  $j$ 'ezima).

Como foi notado na seção anterior, uma forma conveniente de representação para matrizes esparsas é a baseada na representação por grafos associados à estrutura de seus elementos não nulos, onde cada nó do grafo corresponde ao índice de uma linha (ou coluna) da matriz representada, e cada aresta do mesmo, indica a presença de um elemento não nulo na posição definida pelos índices de linha e coluna associados ao par de nós compreendido pela aresta.

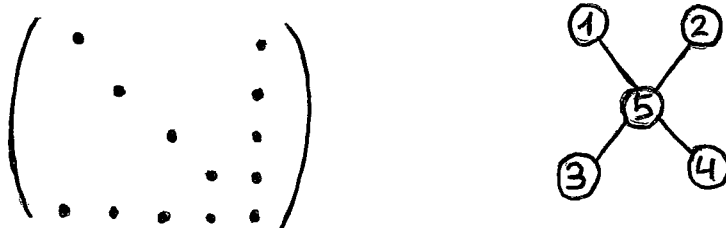
Tomemos como exemplo a matriz da fig\_II(4) e o seu grafo  $G_A$  associado. Se efetuarmos o processo de eliminação Gaussiana na ordem natural em que se encontram as linhas da matriz (seguindo-se a seqüência de nós de 1 à 5), percebe-se que a matriz de fatores resultantes será completamente cheia, visto que ao se eliminar os elementos  $a_{2,1}$ ,  $a_{3,1}$ ,  $a_{4,1}$  e  $a_{5,1}$  (subtraindo-se múltiplos escalares da primeira linha de cada uma das demais), novos elementos acabarão sendo introduzidos nas posições  $a_{2,3}$ ,  $a_{2,4}$ ,  $a_{2,5}$ ,  $a_{3,4}$ ,  $a_{3,5}$  e  $a_{4,5}$ .

Tal fato, nos leva a pensar se esta seria a melhor situação possível para a fatoração de uma matriz tão simples como a apresentada.



fig\_II(4) - Matriz original e seu grafo associado

Assim, se passarmos a considerar permutações com o objetivo de se reduzir o número de novos elementos criados na matriz de fatores, veremos que com uma simples permutação como a apresentada na fig\_II(5), a matriz de fatores resultantes não só possui consideravelmente menos elementos do que a apresentada no caso anterior, como neste exemplo em particular, nenhum elemento novo acabou por ser introduzido no processo, visto que a eliminação dos elementos  $a_{5,1}$ ,  $a_{5,2}$ ,  $a_{5,3}$  e  $a_{5,4}$  contribui apenas no elemento  $a_{5,5}$  e que já pertencia à representação da matriz original.



fig\_II(5) - Matriz reordenada e seu grafo associado

Infelizmente a escolha "ótima" da seqüência de permutações a serem efetuadas ao longo do processo é um problema "NP-Completo" [Y1], de modo que o que se pode esperar como solução prática, são métodos "heurísticos" que tenham como princípio básico a redução do número de novos elementos introduzidos (e conseqüentemente do esforço computacional envolvido).

● Assim o que os métodos heurísticos de reordenamento buscam é algum tipo de minimização do número de elementos de fill-in introduzidos ao longo do processo de eliminação.

Na literatura encontram-se muitos critérios, quase todos baseados no esquema #2 proposto por Tinney [T2] no final da década de 60, e que ficou conhecido como critério de permutação segundo o "menor grau" (*minimum degree*) [G2].

Critérios extremamente mais simples do que este, como o critério #1, que leva em conta apenas o número de elementos não nulos na matriz original (e determina a seqüência de permutações segundo um ordenamento crescente destes valores), não chegam a competir com o de "menor grau" em termos de percentuais de redução do número de *fill-in's*.

Critérios mais complexos como o de #3 (conhecido como o de "menor fill-in local") embora em alguns casos conseguindo reduções significativas, na maioria das vezes é muito mais oneroso do que o critério #2, e portanto não justifica o seu uso, a não ser em casos onde múltiplas soluções de um sistema com a mesma estrutura sejam requeridas (onde o custo adicional de um reordenamento mais elaborado acabaria "diluido", caso o aumento de eficiência nas demais fases do processo, em face de uma maior redução do número de *fill-in's*, se mostrasse acentuado).

O critério de "menor grau" consiste portanto em se efetuar uma "simulação" de todo o processo de eliminação (levando-se em conta apenas as informações estruturais da matriz original e da matriz de fatores sendo gerada, não efetuando desta forma, qualquer operação de ponto flutuante).

O processo como um todo consiste em ir se tomando a cada etapa, como candidato a pivô (removendo-se do grafo associado), o nó que possuir o menor "grau" (ou seja o menor número de conexões com os demais nós), correspondendo no grafo  $G_A^i$  associado à *i*'ésima etapa da simulação, a remoção do nó com o menor número de arestas ligadas ao mesmo.

No processo de remoção de um dado nó do grafo associado, introduzem-se eventualmente novas ligações entre os elementos originalmente ligados ao nó em questão, correspondendo tal etapa da simulação, à criação dos novos elementos de *fill-in*.

E' natural que a cada etapa, se possa eventualmente encontrar nós com o "mesmo grau" e que portanto poderão ser

indistintamente tomados como "candidatos".

Nestes casos muitas heurísticas foram tentadas como "critério de desempate" durante toda a década de 70, e a conclusão obtida (derivada de um longo período de experimentação) foi de que nenhum dos critérios testados em particular, se sobressaiu sobre os demais.

● *Assim, por simplicidade o critério de desempate mais usualmente adotado é tomar o primeiro nó encontrado dentre o conjunto de nós com o menor grau.*

Em todas as abordagens para o problema de "ordenamento ótimo", faz-se uso de muitas das propriedades e ferramentas desenvolvidas na área de Teoria dos Grafos, uma vez que o processo de fatoração esparsa pode ser visto como uma seqüência de eliminações de arestas a partir do grafo  $G_A^0 \equiv G_A$  associado a matriz original e que a cada etapa  $i$  do processo, vai sendo transformado num grafo intermediário  $G_A^i$ , até se chegar a  $G_A^n$  contendo o último nó do processo de eliminação.

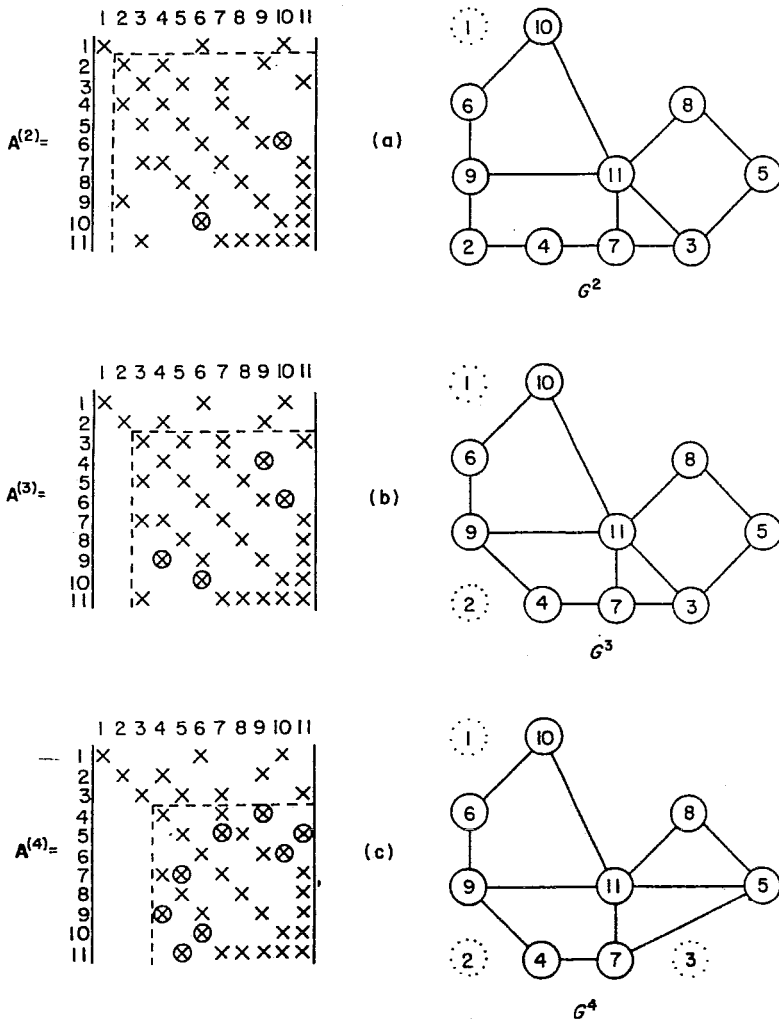
O aparecimento de *fill-in's* por exemplo se dá quando durante o processo de eliminação das arestas associadas a um dado nó, criam-se ligações entre todos os nós ligados ao nó original (e que originalmente não existiam no início de cada etapa do processo), como pode ser notado no exemplo da *fig\_II(6)*.

Neste exemplo, considera-se o processo de eliminação a partir da segunda etapa, (com a estrutura de novos elementos de *fill-in* representada por círculos ao redor dos elementos  $x$  associados).

Percebe-se que com a remoção do nó 2, cria-se uma ligação entre os nós 4 e 9, correspondendo ao elemento de *fill-in* assim introduzido.

Com a eliminação do nó 3, criam-se ligações entre os elementos 5 e 7, e entre 5 e 11, visto serem estes os nós anteriormente conectados ao nó removido.

O processo se estenderia neste caso até a décima etapa, (não representada na figura por analogia com as etapas iniciais ilustradas), onde o grafo associado, conteria apenas o último nó a ser eliminado (o de número 11 em particular)



fig\_II(6) - Grafos associados as etapas da eliminação

Em termos de estruturas de dados, é aconselhável que a fase de reordenamento, se baseie no uso de estruturas do tipo "encadeado" pois como se pode perceber, as informações sendo geradas se alteram dinamicamente com a evolução do processo.

Utiliza-se na maioria das vezes uma representação por listas (encadeadas) de adjacências para cada nó do grafo associado à matriz original, e que vai sendo progressivamente transformado a cada etapa.

Representações alternativas mais eficientes, consagradas na literatura, se baseiam no uso de "Reachable Sets", e no conceito de "eliminação em massa" de nós (explorando-se o conceito de *supernodes*) e não serão consideradas neste trabalho, reportando-se o leitor à IG21.

Contrariamente ao caso de uma representação por "reachable sets", numa implementação convencional (onde o processo de eliminação é simulado mediante apenas operações de união/remoção nos conjuntos de índices associados as linhas envolvidas em cada etapa), o espaço total de armazenamento a ser ocupado pela matriz de fatores, só poderá ser conhecido ao final do processo.

Um ponto benéfico a favor das implementações convencionais (simulando-se a fase de eliminação via operações elementares sobre conjuntos) é o fato destas fornecerem "gratuitamente" a estrutura da matriz de fatores resultante (eliminando assim, a necessidade de uma fase auxiliar de "fatoração simbólica" a fim de se determinar a posteriori esta estrutura, como será visto na próxima seção).

Nas implementações por "reachable sets" ao contrário, não se dispõe de uma estrutura de fatores "pronta" ao final do processo, devendo-se neste caso lançar mão de uma fase auxiliar de processamento dedicada a esta finalidade.

As vantagens em termos de economia e previsibilidade de espaço inerentes a este tipo de implementação no entanto, aliadas a grande eficiência com que se consegue atualmente efetuar a fase "simbólica" subsequente, tornam este tipo de representação o mais aconselhável (especialmente quando mais de uma fatoração de matrizes com a mesma estrutura vier a se mostrar necessária).

Consideraremos no entanto apenas as alternativas baseadas numa representação convencional, por razões de simplicidade didática.

A nível de pseudo-código, são apresentados a seguir, 3 alternativas de ordenação, baseadas nos critérios # 1, 2 e 3 propostos originalmente por Tinney [T2].

Em todas elas é assumido que uma representação para o grafo original,  $G_A$  associado a matriz a ser fatorada seja fornecida como entrada, obtendo-se como resultado em todas as alternativas, o vetor NORDER de índices das linhas a serem eliminadas (ditando a seqüência a ser efetivamente adotada durante a fase numérica de solução).

## Alternativa II(#1)

$i \leftarrow 1$

- 1: Dentre o conjunto de todos os nós não eliminados, escolha o nó  $j$  com o menor de todos os graus. (Caso todos os nós tenham sido eliminados termine o processo).
- 2: Faça NORDER [ $i$ ]  $\leftarrow j$ ,  $i \leftarrow i + 1$
- 3: Marque o nó  $j$  como "já eliminado"
- 4: Volte ao passo 1:

## Alternativa II(#2)

$i \leftarrow 1$

- 1: Dentre o conjunto de todos os nós não eliminados, escolha o nó  $j$  com o menor de todos os graus. (Caso todos os nós tenham sido eliminados termine o processo).
- 2: Faça NORDER [ $i$ ]  $\leftarrow j$ ,  $i \leftarrow i + 1$
- 3: Marque o nó  $j$  como "já eliminado" e processe todas as alterações no grafo associado correspondentes à eliminação do nó em questão (ligando-se todos os nós conectados ao  $j$  entre si, criando-se desta forma a estrutura da linha resultante na matriz de fatores associada à  $j$ )
- 4: Volte ao passo 1:

## Alternativa II(#3)

$i \leftarrow 1$

- 1: Dentre o conjunto de todos os nós não eliminados, escolha o nó  $j$  tal que a sua eliminação venha a resultar na criação do menor número possível de elementos de *fill-in* durante a atual etapa do processo (simulando-se a eliminação para cada um dos nós candidatos dessa etapa). (Caso todos os nós tenham sido eliminados termine o processo).
- 2: Faça NORDER [ $i$ ]  $\leftarrow j$ ,  $i \leftarrow i + 1$
- 3: Marque o nó  $j$  como "já eliminado" e processe todas as alterações no grafo associado correspondentes à eliminação do nó em questão (ligando-se todos os nós conectados ao  $j$  entre si, criando-se desta forma a estrutura da linha resultante na matriz de fatores associada à  $j$ )
- 4: Volte ao passo 1:



Uma vez obtido o "vetor de ordenamento" *NORDER*, nos casos em que múltiplas refatorações de matrizes com a mesma estrutura venham a ser realizadas, pode-se efetuar uma permutação física das linhas da matriz original, tomando como ordem a seqüência ditada pelo vetor de ordenamento, ou seja, a primeira linha/coluna da matriz permutada corresponderá à *NORDER(1)* da matriz original. A segunda corresponderá à *NORDER(2)*, e assim sucessivamente até a *n*'ésima linha/coluna.

Deste ponto, (antes da fatoração numérica propriamente dita), pode se passar à fase de fatoração simbólica nos casos em que a estrutura resultante da matriz de fatores não houver sido fornecida como subproduto do ordenamento (como por exemplo caso o critério #1 tenha sido utilizado, ou uma implementação por "reachable sets" tenha sido adotada para o critério #2), o que passaremos a considerar na seção a seguir.

### II.3 Obtenção da estrutura da matriz de fatores

Em certos procedimentos de ordenação visando a redução de *fill-in*, a estrutura final da matriz de fatores  $U$  não pode ser fornecida como um sub-produto do ordenamento.

Nestes casos, uma etapa auxiliar de determinação da posição estrutural dos fatores (conhecida na literatura como fase de "fatoração simbólica"), mostra-se adequada, uma vez que de posse desta informação, o processo de fatoração numérica propriamente dito, pode ser implementado de forma mais eficiente (como veremos na próxima seção).

A "fatoração simbólica", assim denominada pela literatura, e abordada nesta seção, não deve ser confundida com os procedimentos também "simbólicos", apresentados nos capítulos subsequentes deste trabalho, e que objetivam um aprimoramento complementar da eficiência da fase numérica dos métodos de fatoração considerados.

Na verdade a "fatoração simbólica" usual, pode ser encarada como um sub-conjunto das técnicas "simbólicas" mais gerais, visando a minimização dos *overheads* em termos de execução, e que particularmente atinge este objetivo, mediante a obtenção da estrutura "estática" da matriz de fatores incluindo-se os *fill-in's* gerados, (dispensando desta forma a determinação dinâmica da posição estrutural destes elementos durante a fase numérica da eliminação).

As demais técnicas simbólicas a serem apresentadas, complementam esta fase, determinando-se informações simbólicas adicionais, como os endereços (posições de memória) de cada um dos elementos acessados durante cada etapa da eliminação, ou listas compactas de códigos, cujos procedimentos de "interpretação" e execução das operações codificadas, podem ser mais eficientemente implementados do que as abordagens numéricas convencionais.

Ou seja, tais metodologias simbólicas a serem apresentadas nos capítulos III e IV, constituem o que se optou convencionar por uma fase complementar de pré-processamento denominada "codificação simbólica" pelo autor deste trabalho.

● É apenas na fase de "fatoração simbólica" propriamente dita, que se concentra esta seção, cujo único objetivo é o da determinação da estrutura de fatores resultantes ao final da eliminação (incluindo-se os elementos de fill-in gerados).

A solução para este problema básico a ser enfrentado, consiste em se fazer uma concatenação (controlada) dos conjuntos de índices de colunas associadas de todas as linhas a serem subtraídas da linha corrente, (com o controle se dando de modo a evitar operações de concatenação desnecessárias como veremos a seguir).

Tomemos como exemplo a matriz da fig\_II(7), e consideremos a sétima etapa do processo, correspondendo à eliminação dos elementos à esquerda da diagonal na linha corrente #7.

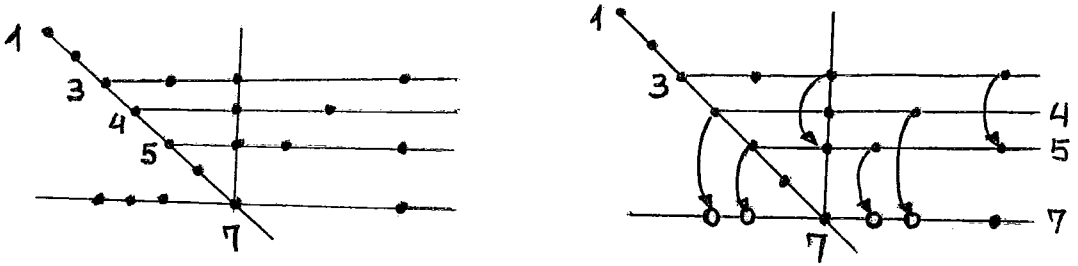
Inicialmente temos que subtrair a linha #3, da linha corrente #7. Ao se efetuar tal subtração, cria-se um novo elemento em  $a_{75}$  (e que posteriormente obrigará também a subtração da linha #5 da linha corrente). Continuando o processo vemos que a linha #4 também necessita ser subtraída da corrente (de modo a eliminar o elemento  $a_{74}$ ). Finalmente em face da criação do novo elemento  $a_{75}$  a linha #5 como notado, acabará por ser subtraída da #7.

Assim uma implementação "imediatista" seria simplesmente concatenar as porções a direita da coluna #7 das linhas #3, #4, #5 e #7, cujas listas de índices de colunas à direita da sétima coluna, (incluindo-a), são dadas pelos conjuntos listados a seguir:

linha #3:	colunas:	7	11	
linha #4:	colunas:	7	9	
linha #5:	colunas:	7	8	11
linha #7:	colunas:	7	11	

Estas listas de índices terão de sofrer uma operação de "concatenação", o que acabará resultando na estrutura final de fatores da linha #7 e que será:

linha #7:	colunas:	7	11	9	8
-----------	----------	---	----	---	---



fig\_II(7) - Concatenação de índices

Uma propriedade da eliminação de Gauss (descoberta por Rose [R8] em meados da década de 70) permite que o número de linhas cujos conjuntos de índices devem ser efetivamente concatenados, possa ser reduzido.

Simplesmente se observou que todas as linhas candidatas a concatenação que possuírem algum elemento não nulo na porção compreendida entre à direita de seu elemento diagonal e à esquerda da coluna corrente (#7 no exemplo em particular), não precisam ser consideradas durante o processo de concatenação (para a formação da linha corrente).

Tal fato se dá pois tomando como base o exemplo em particular, no qual a linha #3 não precisa ser incluída durante o processo de concatenação da linha #7, pois uma vez que possui o elemento  $a_{35}$  por simetria acabará por possuir o elemento  $a_{53}$  em alguma etapa mais adiante, e na eliminação deste elemento (durante a quinta etapa do processo), a porção da estrutura de índices a direita da coluna #5 na linha #3, acabará por ser concatenada ao conjunto de fatores resultantes da linha #5 (a ser gerada durante o processamento da quinta etapa).

Assim, quando da concatenação das linhas para a obtenção da estrutura definitiva da linha #7, ao se adicionar o conjunto de índices associados a linha #5, se estará implicitamente adicionando também o conjunto de índices da linha #3 (uma vez que sua estrutura por construção, já deverá ter sido concatenada à da linha #5 na quinta etapa do processo).

Assim, no exemplo em particular, o conjunto de linhas a serem efetivamente concatenadas para a obtenção da estrutura final da linha #7 é dado por:

linha #4:	colunas:	7	9	
linha #5:	colunas:	7	8	11
linha #7:	colunas:	7	11	

Desta forma, ao se efetuar a operação de concatenação na ordem apresentada acima, a nova estrutura resultante para a linha #7 acabará sendo:

linha #7:	colunas:	7	9	8	11
-----------	----------	---	---	---	----

Um detalhe a respeito das operações de concatenação descritas nos parágrafos anteriores é que todas podem ser feitas de forma desordenada, pois durante a fase puramente "simbólica" do processo de fatoração (quando o único objetivo é o de determinar-se a estrutura final da matriz de fatores resultantes) não é necessário que as listas de colunas associadas a cada linha estejam ordenadas de modo crescente.

Um outro item que passou a ser merecedor de destaque nas implementações modernas do processo de fatoração simbólica é a possibilidade de se abortar imediatamente o processo, tão logo se encontre a partir de um certo ponto, uma linha da matriz de fatores resultantes com todas as posições ocupadas, o que significará que a partir deste ponto a submatriz de fatores remanescentes a se eliminar será completamente densa, e portanto sua estrutura poderá ser facilmente determinada, sem ter de se recorrer à operações de concatenação (e que seriam bem mais onerosas durante o final do processo).

De posse de uma notação básica de conjuntos associados as diversas estruturas envolvidas no processo de fatoração simbólica (a serem apresentadas abaixo), incluímos a título de complementação, um pseudo-código para a fase simbólica descrita nesta seção.

Partindo-se das definições:

**Definição II.3(1)** *Estrutura de elementos por linha*

$$Struct(M_{i,*}) := \left\{ k > i \mid m_{i,k} \neq 0 \right\}$$

**Definição II.3(2)** *Estrutura de elementos por coluna*

$$\text{Struct}(M_{*,j}) := \left\{ k < j \mid m_{k,j} \neq 0 \right\}$$

**Definição II.3(3)** *Primeiro elemento não nulo*

$$p(i) := \begin{cases} i & \text{caso } \text{Struct}(U_{i,*}) = \emptyset \\ \min \{ j \in \text{Struct}(U_{i,*}) \} & \text{caso} \\ & \text{contrário} \end{cases}$$

E da propriedade:

**Propriedade II.3(a)** *Concatenação selecionada*

$$\text{Struct}(U_{i,*}) := \text{Struct}(A_{i,*}) \cup \left[ \bigcup_{j < i} \{ \text{Struct}(U_{j,*}) \mid p(j) = i \} \right] - \{ i \}$$

Pode-se apresentar o procedimento:

**Procedimento II.3(A)** Fatoração Simbólica

```

para i de 1 até n faça
    Ri ← ∅
para i de 1 até n faça
    S ← Struct(Ai,*)
    para j ∈ Ri faça
        S ← S ∪ Struct(Uj,*) - { i }
    Struct(Ui,*) ← S
    se Struct(Ui,*) ≠ ∅ então
        p(i) ← min { j ∈ Struct(Ui,*) }
        Rp(i) ← Rp(i) ∪ { i }

```

A propriedade II.3(b) é justamente a descoberta por Rose [R81], visando a redução do número de linhas a serem concatenadas e exemplificada nesta seção.

Cabe notar, que para a aplicação do algoritmo de fatoração simbólica apresentado, é assumido que uma permutação física (simétrica) nas linhas (e colunas) da matriz original seguindo a ordem ditada pelo vetor de ordenamento NORDER (apresentado na seção anterior), tenha sido previamente executada antes de se iniciar o processo.

O algoritmo pode ser melhorado, caso se inclua um teste adicional logo após a condição se  $Struct(U_{i,*}) \neq \emptyset$  então, verificando-se se a cardinalidade de  $Struct(U_{i,*})$  é igual a  $n-i$ , caso em que a estrutura da linha resultante seria completamente "cheia", e poderia-se interromper o algoritmo a partir deste ponto, gerando-se trivialmente a estrutura das linhas densas da sub-matriz restante, conforme foi comentado.

Uma vez obtida a matriz de fatores pelo processo de fatoração simbólica descrito, faz-se necessário uma etapa complementar de ordenação dos índices de colunas em cada uma das linhas de  $U$  geradas, visto serem as concatenações realizadas no algoritmo apresentado, efetuadas de forma desordenada, de modo a se reduzir o esforço computacional durante o processamento.

A solução para o problema de ordenação dos  $n$  conjuntos de índices associados a cada uma das linhas, se dá mediante a dupla aplicação de um algoritmo para a transposição de matrizes esparsas, e bastante divulgado na literatura [A5].

A essência do método é na primeira passagem do algoritmo, com a transposição da estrutura de  $U$  por linhas, se conseguir montar a estrutura da transposta com os elementos ordenados em cada coluna. Na segunda passagem, com a transposição de volta à estrutura por linhas, se conseguir a montagem da mesma de forma ordenada (por linhas).

Tal algoritmo por sua simplicidade, não será exibido aqui, reportando-se o leitor a [P1], [A8], para uma apresentação mais detalhada.

## II.4 Metodologias convencionais para a fase numérica

Nesta seção finalmente irá se apresentar os métodos de fatoração numérica tidos como "convencionais" por se utilizarem de técnicas já estabelecidas e de amplo conhecimento na literatura ao longo das duas últimas décadas.

O problema básico a ser abordado inicialmente é o da combinação linear de vetores armazenados em algum formato esparsos, correspondendo ao *loop* mais interno do processo de eliminação.

As duas alternativas de eliminação apresentadas, baseiam-se nas versões com geração dos fatores por linhas, e por colunas, apresentadas no capítulo I.

Finalmente o processo de retro-substituição é apresentado apenas a nível de complementação, visto não pertencer ao escopo de abordagem do presente trabalho.

Um dos "problemas básicos" de uma abordagem esparsa para a solução de sistemas lineares, consiste se implementar de forma eficiente as operações de combinação linear de linhas anteriores com a linha corrente.

O que se requer é uma forma de se adicionar vetores esparsos e que se encontram armazenados de alguma forma compacta (contiguamente no caso de uma representação sequencial ou sequencial segmentada como visto na seção II.1) ou na forma encadeada.

O problema da adição de vetores esparsos pode ser solucionado de três formas distintas, o que passaremos a discutir a seguir (supondo-se o caso em que os conjuntos de índices associados à cada linha já se encontram ordenados de modo crescente segundo colunas).

A primeira solução consiste em se varrer "controladamente" os dois vetores de índices, tomando-se o cuidado de só se efetuar uma operação de soma quando o índice (da coluna) associado à ambos os vetores for coincidente.

Neste caso um pseudo-código é apresentado na alternativa II.4(A).



● *Em todos os casos, assume-se que o vetor V a ser adicionado a U, possua como estrutura de elementos não nulos, um sub-conjunto da estrutura de U, (ou seja a estrutura de U engloba toda a estrutura de vetores a serem adicionados ao mesmo, tendo sido determinada em alguma fase anterior do processamento).*

Nos procedimentos abaixo, V denota um dos vetores esparsos a serem adicionados ao vetor básico U (também esparsos), ambos armazenados de forma seqüencial segmentada, com *iu*, *iuf*, *iv*, *ivf* denotando os ponteiros para a posição de início e fim de cada representação, e com JU e JV indicando os conjuntos de colunas associadas aos vetores U e V.

#### Alternativa II.4(A)

#### Varredura controlada

```

      k ← iu
      para j de iv até ivf faça
α:      se JV(j) = JU(k) então
          U(k) ← U(k) + V(j)
          k ← k + 1
          senão
              k ← k + 1
          vá para α
      fim se
      fim para j
  
```

Outra forma de se implementar a adição de 2 vetores esparsos (armazenados de modo seqüencial segmentado), e que vem sendo utilizada até hoje, desde [G15], consiste em primeiro se descompactar os elementos do vetor base U, em uma estrutura auxiliar de trabalho W de dimensão n e a seguir adicionar-se os elementos dos vetores V nas posições correspondentes de W, compactando-se finalmente todos os elementos diferentes de zero acumulados temporariamente em W, nas posições ocupadas pelos elementos originais de U.

#### Alternativa II.4(B)

#### Vetor de trabalho expandido

```

      para j de iu até iuf faça
          W(JU(j)) ← U(j)
      para j de iv até ivf faça
          W(JU(j)) ← W(JU(j)) + V(j)
      para j de iu até iuf faça
          U(j) ← W(JU(j))
  
```

Neste caso, incorre-se nos *overheads* de ter-se que carregar e descarregar o vetor auxiliar de trabalho  $W$  com o conteúdo do vetor básico original  $U$ . No caso em que apenas um vetor  $V$  é adicionado ao vetor  $U$ , este *overhead* pode vir a ser por demais oneroso, por envolver dois *loops* adicionais com quase a mesma ordem de operações que as do *loop* central.

Quando o número de vetores  $V$  a serem adicionados em  $U$  supera um determinado valor, o *overhead* original passa a ser amenizado durante o processo.

● O método descrito acima, é o que foi adotado em praticamente todas as implementações esparsas desenvolvidas a partir dos anos 80 e consagradas na literatura.

A terceira forma de solução, adotada em [A1] e [A8], requer o uso de uma lista auxiliar, contendo diretamente as posições de memória na representação compacta de  $U$  correspondente aos elementos de  $V$  a serem subtraídos. No caso da adição de um único vetor  $V$ , esta é a forma mais eficiente de se efetuar a adição de vetores armazenados de forma esparsa.

Quando o número de vetores a serem adicionados se torna expressivo, este passa a ser o mais oneroso de todos os métodos em termos de espaço. Assim o seu uso durante todo o processo de fatoração torna-se praticamente inviável do ponto de vista do dispêndio de recursos exigidos. A sua utilização "controlada" no entanto (durante a fase inicial do processo de fatoração) quando o grau de densidade das linhas ainda é consideravelmente baixo, representa uma das opções que foram empregadas com sucesso em implementações recentes, especialmente na área de algoritmos de Pontos Interiores para Programação Linear [A1].

#### Alternativa IL4(C)

#### Listas simbólicas de endereços

```
nadr ← 1
para j de iv até ivf faça
    UCLSTADD(nadr) ← UCLSTADD(nadr) + V(j)
    nadr ← nadr + 1
fim para
```

Onde LSTADD denota a "lista de endereços" contendo as posições de memória dos elementos de  $U$ , a sofrerem a adição dos elementos correspondentes nas colunas associadas de  $V$ .

Com isto encerra-se a apresentação das alternativas para se efetuar a adição de dois ou mais vetores esparsos.

Cabe lembrar, que em todos os casos foi assumido que o vetor básico  $U$  já se encontrava representado com a estrutura resultante que o mesmo teria após o processo de adição dos múltiplos vetores  $V$ . Assim, as posições correspondentes aos elementos de *fill-in* do vetor  $U$ , devem inicialmente conter explicitamente um valor nulo.

A seguir, detalharemos os procedimentos completos relativos a fase "numérica" da fatoração propriamente dita, (e que podem ser aplicados indefinidamente para se fatorar matrizes com a mesma estrutura).

A técnica utilizada para se efetuar a soma de vetores esparsos é a baseada na Alternativa II.4(B), via o uso de um vetor auxiliar de trabalho de dimensão  $n$  (e que é adotada como padrão por todos os pacotes comercialmente disponíveis como YSMP, SPARSPAK e HARWELL).

Como já detalhado anteriormente, a idéia básica consiste em se expandir a linha corrente de  $U$  (sofrendo o processo de eliminação) num vetor auxiliar  $W$ , e a seguir efetuar a subtração de todos os demais vetores esparsos (das linhas necessárias para a eliminação da linha corrente) acumulando-se os valores nas posições expandidas correspondentes no vetor auxiliar.

Ao final do processo de eliminação da linha corrente, o vetor auxiliar é então compactado novamente na estrutura sequencial contígua de armazenamento adotada para a matriz de fatores.

Quando a linha assim gerada vier a ser necessária durante o processo de eliminação de outra linha, os seus coeficientes serão novamente subtraídos de forma expandida, no vetor auxiliar de trabalho  $W$ .

O Procedimento II.4(1) consiste justamente num processo de eliminação por linhas (assumindo-se que a matriz original e a triangular superior de fatores resultante, estejam armazenadas de forma sequencial segmentada por linhas).

A estrutura de dados utilizada para a representação da matriz de fatores  $U$  é baseada nos vetores  $IU$ ,  $IUF$ ,  $JU$ ,  $UN$ , contendo respectivamente ponteiros para as posições inicial e final de cada linha de  $U$ , bem como as colunas e os valores numéricos associados aos elementos não nulos excluindo-se a diagonal (armazenada separadamente em  $DI$ , e contendo o inverso destes valores ao final do processo).

É assumido também que inicialmente a matriz original esteja mapeada sobre o mesmo espaço de armazenamento utilizado para a matriz de fatores, com as posições relativas aos elementos de *fill-in* explicitamente "zeradas".

Durante o processo de eliminação, faz-se necessário também a representação estrutural da transposta de  $U$  (correspondendo a matriz triangular inferior de fatores  $U^T$ ). Esta informação, quando o espaço de armazenamento disponível é limitado, pode ser gerada dinamicamente durante o processo como em  $IP11$ . Por razões de simplicidade e eficiência (em termos de tempo de execução), sempre que possível a informação explícita da estrutura da transposta será adotada neste trabalho.

Assim neste caso,  $IUT$ ,  $IUTF$  e  $JUT$ , análogamente à estrutura adotada para a matriz  $U$ , denotam os ponteiros para as posições inicial e final, bem como para os índices das colunas (a esquerda da diagonal) de cada linha da matriz  $U^T$ .

A outra alternativa para a geração da matriz de fatores apresentada no Procedimento  $II.4(2)$ , consiste na geração dos fatores por colunas, adotando-se a mesma representação sequencial segmentada por linhas para as matrizes original e de fatores, e utilizando-se o vetor auxiliar de trabalho  $W$  (inicialmente "zerado"), como um vetor de contribuições a serem adicionadas na representação original de cada coluna de  $U$  sendo gerada.

**Procedimento IL4(1)      Geração dos fatores por "linhas"**

*(para cada linha corrente "i" à eliminar)*

**para i de 1 até n faça**

*(inicialização do apontador de posições iniciais)*  
 $IUP(i) \leftarrow IU(i)$

*(inicialização do valor da diagonal)*  
 $piv \leftarrow DI(i)$

*(expansão da linha corrente "i" no vetor de trabalho)*

**para j de IU(i) até IUF(i) faça**

$W(JU(j)) \leftarrow UN(j)$

**fim para j**

*(para cada linha "l" a ser subtraída da corrente)*

**para k de IUT(i) até IUTF(i) faça**

*(linha a ser subtraída de "i")*  
 $l \leftarrow JUT(k)$

*(posição inicial em "l")*  
 $iuc \leftarrow IUP(l)$

*(próxima posição inicial)*  
 $IUP(l) \leftarrow IUP(l) + 1$

*(fator multiplicativo)*  
 $um \leftarrow UN(iuc) * DI(l)$

*(atualização da diagonal)*  
 $piv \leftarrow piv - UN(iuc) * um$

*(normalização)*  
 $UN(iuc) \leftarrow um$

*(subtração da linha "l" da linha corrente "i")*

-----  
**para j de iuc + 1 até IUF(l) faça**

$W(JU(j)) \leftarrow W(JU(j)) - UN(j) * um$

-----  
**fim para j**

**fim para k**

*(inversão do elemento diagonal)*  
 $DI(i) \leftarrow 1 / piv$

*(compactação definitiva da linha corrente "i")*

**para j de IU(i) até IUF(i) faça**

$UN(j) \leftarrow W(JU(j))$

**fim para j**

**fim para i**

Alguns comentários sobre o procedimento apresentado e as estruturas auxiliares empregadas, fazem-se necessários.

O vetor IUP até então ainda não introduzido no contexto da eliminação esparsa, é um dos ingredientes fundamentais e presente em quase todos os métodos a serem abordados ao longo do texto.

Assim, uma apresentação mais detalhada da sua finalidade, é um item merecedor de destaque.

Reportando-se o leitor a seção II.3, quando se apresenta um exemplo de concatenação de índices de colunas, gerando-se a estrutura da sétima linha (no exemplo considerado), percebe-se que as linhas a serem efetivamente subtraídas da #7 são as de #3, #4 e #5.

Em todas elas, no método de eliminação por linhas, conforme a fig\_I(2), apenas a porção "acima da linha corrente" das demais linhas envolvidas, precisa ser explicitamente subtraída da linha base em questão.

Deste modo, no caso particular da linha #3, apenas a porção à direita da coluna #7 (incluindo-a), precisará ser efetivamente subtraída da porção à direita da diagonal na linha base #7 sendo gerada.

A finalidade do vetor IUP é na verdade se manter um apontador dinâmico para a posição inicial de cada linha, a ser efetivamente usada durante a subtração (da linha associada), de alguma outra linha base corrente, mais adiante ao longo do processo.

Deste modo, IUP é inicializado para as posições iniciais (contendo o primeiro elemento não nulo à direita da diagonal) de cada linha, e sucessivamente incrementado, a medida que uma dada linha em questão é usada no processo de contribuição sobre uma outra linha base corrente, visto que uma vez contribuindo com elementos a partir da posição IUP(i), na próxima vez em que a linha i vier a se mostrar necessária, a parcela a ser efetivamente utilizada para a contribuição, começará a partir de IUP(i)+1, e isso explica em linhas gerais, o porque da presença desta estrutura auxiliar (de dimensão n), e a razão para sua "dinamicidade".

Outra estrutura que se mostra conveniente comentar, é a da transposta de  $U$ , visto ser utilizada para a determinação de quais linhas necessitarão ser subtraídas da linha corrente.

Em face a simetria da matriz original, e por consequência, da matriz de fatores resultantes, o processo de eliminação (simétrica) efetua apenas operações sobre os elementos armazenados na porção triangular superior.

Porém, a estrutura de elementos não nulos que se busca anular em cada etapa do processo, se encontra à esquerda da diagonal da linha base, a cada etapa fundamental  $i$  da eliminação. Tal estrutura, corresponde por simetria, a estrutura da porção da  $i$ ésima coluna de  $U$ , acima da diagonal, e que corresponde portanto à da representação por linhas de  $U^T$ , utilizada ao longo deste trabalho.

Deste modo  $JUT(k)$  denota o índice das linhas a serem de fato subtraídas da linha base corrente a cada etapa.

Esta informação, como foi comentado, poderia ser determinada dinamicamente durante o processo, porém, pelas razões já apresentadas, optou-se por adota-la explicitamente, mediante a utilização da matriz  $U^T$ .

A última "filigrana", no procedimento II.4(1) é a normalização dos elementos de cada linha, e o armazenamento do inverso da diagonal, no lugar do seu valor direto.

A "normalização" em questão, corresponde na verdade à divisão de todos os fatores de cada linha, pelo elemento diagonal correspondente, de modo a que a matriz  $U$  assim obtida, satisfaça a condição de possuir a diagonal unitária.

A "filigrana" como se comentou, é que a normalização vai sendo paulatinamente efetuada, a medida que se avança o ponteiro  $IUP(l)$ , e que ao final de todo o processo de eliminação, terá varrido toda a estrutura de elementos não nulos de cada linha  $l$  da matriz de fatores.

O armazenamento de  $1/u_{ii}$  no lugar de  $u_{ii}$  se deve ao fato de operações de divisão serem sempre mais onerosas do que as de multiplicação, e com o armazenamento do valor inverso, as novas divisões poderem ser efetuadas mediante multiplicações.

**Procedimento II.4(2)      Geração dos fatores por "colunas"**

*( "zeragem" inicial do vetor de trabalho)*

para i de 1 até n faça

$W(i) \leftarrow 0$

fim para i

*(para cada linha corrente "i" à eliminar)*

para i de 1 até n faça

*(inicialização do apontador de posições finais)*

$IUP(i) \leftarrow IU(i)$

*(inicialização do valor da diagonal)*

$piv \leftarrow DI(i)$

*(para cada linha "l" a ser subtraída da corrente)*

    para k de IUT(i) até IUTF(i) faça

*(linha a ser subtraída de "i")*

$l \leftarrow JUT(k)$

*(posição final na linha "l")*

$iuc \leftarrow IUP(l)$

*(próxima posição final)*

$IUP(l) \leftarrow IUP(l) + 1$

*(fator multiplicativo)*

$um \leftarrow UN(iuc) - W(l)$

*( "zeragem" do vetor de trabalho)*

$W(l) \leftarrow 0$

*(normalização)*

$UN(iuc) \leftarrow um * DI(l)$

*(atualização da diagonal)*

$piv \leftarrow piv - UN(iuc) * um$

*(adição das "contribuições" relativas a "l")*

        para j de IUC(l) até iuc - 1 faça

$W(JUC(j)) \leftarrow W(JUC(j)) + UN(j) * um$

        fim para j

    fim para k

*(inversão do elemento diagonal)*

$DI(i) \leftarrow 1 / piv$

fim para i



No procedimento II.4(2), cabe notar que o papel de IUP é revertido, com relação ao procedimento anterior. Neste caso, IUP denota agora a posição "final" de cada linha à contribuir sobre a linha base corrente, visto que no método de eliminação com geração dos fatores por colunas, conforme a fig\_I(1), a porção efetivamente subtraída vai da diagonal da linha contribuinte, até a coluna associada à linha base corrente.

Outro comentário, é que o vetor de trabalho  $W$  neste caso é utilizado como um vetor de "contribuições", acumulando as parcelas a serem posteriormente subtraídas da linha corrente, e desta forma, precisa estar "zerado" ao início de cada etapa fundamental do processo.

A vantagem deste novo procedimento está no fato de se evitar o processo de descompactação/compactação da linha corrente, indispensável no procedimento anterior. O preço pago por esta economia se reflete em uma operação aritmética de soma em ponto flutuante adicional para cada elemento da matriz de fatores resultante (em " $um \leftarrow UN(iuc) - W(U)$ ") mas que mesmo assim se mostra compensador, pois o *overhead* em decorrência da inicialização e incremento dos *loops* de compactação e descompactação no caso anterior, normalmente superam o custo da operação de ponto flutuante adicional deste processo.

A única grande desvantagem deste procedimento se dá com relação a impossibilidade de se explorar de forma eficiente algumas características encontradas com frequência na matriz de fatores, como a presença de *supernodes* ou de sub-porções densas (e que normalmente são determinantes em termos do volume de cálculo dispendido nestas porções em comparação com o restante do processo de eliminação).

Assim, um procedimento de fatoração ideal, muitas vezes é uma combinação *híbrida* entre dois ou mais procedimentos (como será visto na seção III.4), adotando-se a geração por coluna nas etapas iniciais do processo, e revertendo-se para a geração por linhas no final (ao se aproximar da sub-porção densa, ou quando o percentual de *supernodes* se mostrar significativo por exemplo).

Complementando a abordagem numérica convencional, apresentamos os procedimentos relativos à fase de retro-substituição.

**Procedimento II.4(3)**

Retro-Substituição

*(forward e diagonal)*

```

para i de 1 até n faça
    X(i) ← B(i)

para i de 1 até n-1 faça
    xi ← X(i)
    para j de IU(i) até IUF(i) faça
        X(JU(j)) ← X(JU(j)) - xi * UN(j)
    X(i) ← xi * DI(i)
X(n) ← X(n) * DI(n)

```

*(backward)*

```

para i de n-1 de volta até 1 faça
    xi ← X(i)
    para j de IUF(i) de volta até IU(i) faça
        xi ← xi - X(JU(j)) * UN(j)
    X(i) ← xi

```

Para finalizar este capítulo, é conveniente se listar toda a seqüência de procedimentos e fases envolvidas para a obtenção da solução de sistemas lineares esparsos.

*Uma única vez (para cada matriz estruturalmente distinta)*

- Ordenamento visando à redução de *fill-in's*
- Permutação simétrica segundo o ordenamento "ótimo"
- Fatoração Simbólica (determinando a estrutura de **U**)
- Ordenação das colunas dos fatores em cada linha

*Repetidas vezes (para matrizes numericamente distintas)*

- Fatoração Numérica (com geração por linhas ou colunas)
- Retro-Substituições (forward, diagonal e backward)

## Capítulo III

## ABORDAGENS SIMBÓLICAS

Neste capítulo serão apresentadas novas técnicas visando a um aumento da eficiência computacional da fase de fatoração numérica.

Para tal, uma nova fase de processamento executada uma única vez, antes da fase de fatoração numérica propriamente dita, produz como resultado, informações ou códigos, que quando devidamente interpretados, possibilitam a redução dos *overheads* intrínsecos de uma abordagem numérica convencional, (como descompactações/compactações do vetor de trabalho auxiliar expandido, e ineficiências a nível de acesso as posições de memória, como operações de acesso *indireto* desnecessárias).

Estas novas abordagens, receberam a denominação de simbólicas (pelo autor deste trabalho), em função da idéia básica original, (datada do início dos anos 70, e introduzida por Gustavson [G14]), se referir a uma metodologia "simbólica" para a resolução de sistemas lineares esparsos.

O termo simbólico, nas abordagens deste capítulo, como já foi comentado, em nada tem a ver com o processo de "fatoração simbólica" apresentado na seção II.3, cujo único objetivo é o da determinação da estrutura da matriz de fatores resultantes, (incluindo-se as novas posições introduzidas pelos elementos de *fill-in*).

Tal procedimento trabalha apenas com informações do tipo estrutural, ao passo que os procedimentos simbólicos apresentados neste capítulo, se dedicam exclusivamente à fase numérica do processo, assumindo portanto que a fase de "fatoração simbólica", como definida na literatura, tenha sido previamente executada.

Sob o ponto de vista mais geral, os procedimentos simbólicos apresentados nesta seção, englobam a fatoração simbólica "tradicional", como um sub-conjunto, pois é possível a construção de uma fase de pré-processamento simbólico inicial, que produza além da estrutura de fatores resultantes, estruturas auxiliares visando um aumento ainda maior da eficiência da fase numérica *per si*.

Estes serão portanto os procedimentos considerados neste capítulo, (e até o final deste trabalho).

### III.1 Técnicas visando ao aumento da eficiência

Ao se observar os códigos de fatoração numérica, como os apresentados nos procedimentos II.4(1) e II.4(2), percebe-se que apesar de toda sorte de estruturas auxiliares e demais "sutilezas" empregadas, a codificação final dos mesmos é "simples", no sentido em que um código ocupando apenas uma página de texto, ser capaz de descrever completamente a fase numérica da eliminação de matrizes esparsas.

● *O que o leitor poderia se perguntar obviamente é se seria possível melhorar ainda mais o desempenho de tais métodos.*

A resposta, como já se deve estar pressentindo, é afirmativa mais uma vez, e que por razões históricas, completamente distantes da abordagem um tanto quanto "destilada" apresentada nos procedimentos da seção II.4, merecem um certo destaque como forma de introdução a esta seção, em particular.

O principal fato, é que na época em que Gustavson introduziu sua técnica de "geração simbólica de códigos dedicados à resolução de sistemas esparsos", os códigos "gerais" de esparsidade disponíveis na literatura, (e distribuídos na forma de subrotinas pelos grandes centros acadêmicos de pesquisa), eram deveras mais "complexos" do que os procedimentos apresentados no segundo capítulo deste trabalho.

Assim, qualquer esforço adicional, no sentido de se elevar o desempenho da fase numérica, eram (e continuam até hoje) válidos, pois em certas aplicações, centenas ou mais vezes, fatorações de matrizes com a mesma disposição estrutural de elementos não nulos acabam sendo necessárias, como por exemplo na área de simulação de circuitos elétricos, ou solução de equações diferenciais parciais.

Voltemos portanto ao início dos anos 70, procurando

reinterpretar o trabalho de Gustavson, a luz dos conhecimentos atuais.

A idéia (na época brilhante) proposta por Gustavson e sua equipe, foi simplesmente produzir como resultado final, para cada tipo estrutural de sistema linear a se resolver, um código de máquina (assembler) ou em FORTRAN, que simplesmente efetuasse todas as operações aritméticas do processo de eliminação.

A grande vantagem do código assim gerado, era que o mesmo era livre de quaisquer formas de *loops* ou acessos indiretos a estruturas auxiliares, como vetores de trabalho expandidos, etc ...

Ou seja, o código produzido pela abordagem de Gustavson era completamente livre de *overheads*, e o que se efetuavam eram apenas as operações aritméticas necessárias, diretamente sobre as posições de memória correspondentes aos elementos de **U** (contendo originalmente os elementos da matriz original, com as posições relativas aos *fill-in's*, inicialmente "zeradas").

A luz das técnicas apresentadas na seção II.4, percebe-se que o que Gustavson introduziu na época, foi uma quarta alternativa de se efetuar a soma de vetores esparsos, simplesmente "desenrolando-se" completamente o *loop* da alternativa II.4(3) (o que será visto mais adiante).

A forma proposta para a solução do problema, na verdade se aproxima bastante da forma como se apresentaria o processo de eliminação Gaussiana, passo a passo, efetuando-se todos os cálculos no "quadro-negro" (sem qualquer auxílio de aparatos típicos de uma abordagem computacional, como registradores intermediários, indexadores, ou apontadores).

Segue-se portanto um exemplo de código em FORTRAN, extraído da referência original [G14], e apresentado na fig\_III(1) a seguir.

Na versão original de Gustavson, se lançava mão de vetores auxiliares C e Y, simplesmente para se evitar que os valores originais de A e B do sistema original, fossem reescritos, (o que poderia ser evitado sem esta imposição).

```

//SOLVE EXFC FORTRANG,PARAM='MAP,DECK,LIS1'
//SYSTIN DD *
      DIMENSION A( 12),B( 6),C( 18),Y( 6),X( 6)
      5 READ (5,10) (A(I),I=1, 12)
      10 FORMAT (1PSE14,7)
      WRITE (6,20) (A(I),I=1, 12)
      20 FORMAT (1X1P10E13,6)
      READ (5,10) (B(I),I=1, 6)
      WRITE (6,20) (B(I),I=1, 6)
      C( 1)=A( 1)
      C( 2)=A( 2)
      C( 3)=A( 3)
      Y( 1)=B( 1)/C( 1)
      C( 4)=A( 4)/C( 1)
      C( 5)=A( 5)/C( 1)
      C( 6)=A( 6)
      C( 7)=A( 7)
      Y( 2)=[-C( 2)*Y( 1)+B( 2)]/C( 6)
      C( 8)=-C( 2)*C( 4)/C( 6)
      C( 9)=-C( 2)*C( 5)/C( 6)
      C(10)=A( 8)
      C(11)=-C( 4)*C( 3)
      C(12)=-C( 8)*C( 7)
      Y( 3)=B( 3)/C( 10)
      C(13)=A( 9)
      C(14)=A(10)
      Y( 4)=[-C( 3)*Y( 1)-C( 11)*Y( 3)+B( 4)]/C( 13)
      C(15)=-C( 3)*C( 5)/C( 13)
      C(16)=-C( 9)*C( 7)+A( 11)
      C(17)=-C(15)*C( 16)
      Y( 5)=[-C( 7)*Y( 2)-C( 12)*Y( 3)+B( 5)]/C( 16)
      C(18)=A(12)
      Y( 6)=[-C(14)*Y( 4)-C( 17)*Y( 5)+B( 6)]/C( 18)
      X( 6)=Y( 6)
      X( 5)=Y( 5)
      X( 4)=-C( 15)*X( 5)+Y( 4)
      X( 3)=Y( 3)
      X( 2)=-C( 8)*X( 3)-C( 9)*X( 5)+Y( 2)
      X( 1)=-C( 6)*X( 3)-C( 5)*X( 5)+Y( 1)
      WRITE (6,20) (C(I),I=1, 18)
      WRITE (6,20) (Y(I),I=1, 6)
      WRITE (6,20) (X(I),I=1, 6)
      GO TO 5
      END

```

fig\_III(1) - Código "loop-free" à la Gustavson

No exemplo apresentado em questão, assumiu-se que a matriz original A estivesse representada sequencialmente por linhas, no vetor A, bem como o vetor lado direito b de forma densa em B, com a solução do sistema de 6 equações (do exemplo em particular), assim formulado, no vetor X.

Um dos grandes problemas com a abordagem de Gustavson, é a necessidade de se ter que lançar mão de uma fase de "geração automática de código", pois o código apresentado na fig\_III(1) por exemplo, foi produzido "automaticamente" por um dos módulos encarregados da geração do código FORTRAN, no pacote original desenvolvido na época, e tal código aplica-se apenas à solução de sistemas com a característica estrutural do problema para o qual o mesmo foi gerado.

O que se tentou na época, de forma a amenizar esta imposição, foi ao invés de se gerar um código de alto nível, gerar-se diretamente um código de máquina ou Assembler, e cuja fase de "carregamento" em memória e *linkedição* com as demais rotinas da aplicação principal, pudesse ser feito em tempo hábil menor.

O problema maior da abordagem original de Gustavson é que para matrizes esparsas de dimensão elevada, mesmo com um

grau de densidade reduzido, o espaço total ocupado pelo código gerado passa a ser impraticável para a maioria dos computadores de pequeno a médio portes atuais.

Na época em que a idéia original de Gustavson foi apresentada, tais requisitos eram ainda mais restringentes, pois apenas em máquinas do mais elevado porte, se poderia cogitar em aplicar o método para a solução de problemas com uma dimensão da ordem de milhares de variáveis.

Deste modo, uma idéia que se mostrava excepcional para matrizes de pequeno a médio porte, era completamente inviável na época, por simples falta de recursos computacionais de porte à altura.

Não é necessário dizer, que a idéia acabou rapidamente caindo no esquecimento, sendo contudo citada até hoje, em referências conceitadas como [D6].

Tal era o panorama da resolução de sistemas (por métodos diretos), de meados da década de 70 até meados da década seguinte (período em que as estratégias convencionais, apresentadas no segundo capítulo, floresceram), e quando uma "nova variável" acabou por viabilizar o "reaquecimento" das idéias propostas por Gustavson.

O fato em questão, foi a descoberta de um "novo método polinomial" para a resolução de problemas de Programação Linear, proposto na época, por um ainda não tão conhecido, Narendra Karmarkar [K1].

O que o método tinha de "novo" do ponto de vista computacional, em relação as abordagens até então, baseadas no método Simplex [D49], ou no método dos Elipsóides proposto por Kachian (e que se havia mostrado inaplicável do ponto de vista computacional de modo a competir com outras abordagens), era que o novo método proposto por [K1], segundo o seu próprio autor, se mostrava competitivo computacionalmente com relação a um método sólidamente consagrado ao longo das últimas 4 décadas, como era o caso do método Simplex.

O fato que de certa forma despertou novamente o interesse de pesquisadores, por melhores estratégias de resolução de sistemas esparsos, se dava porque todas as variantes do método original proposto por [K1], se baseavam (até então), na resolução de sistemas esparsos (com matrizes

simétricas definidas positivas), sendo tal etapa a de maior peso computacional em cada iteração básica dos algoritmos de Pontos Interiores utilizados para a resolução do problema de Programação Linear.

● *Ou seja com a introdução de melhoramentos nas rotinas de solução de sistemas esparsos, se conseguiria um aprimoramento proporcional da eficiência de algoritmos como os de IK11, IA11 e IO11.*

De um grupo de pesquisadores em Berkeley IA11, surgiu portanto a idéia de se tentar um aprimoramento da eficiência, estendendo-se a idéia original de Gustavson, numa estratégia de forma "interpretada" para o problema, e que deste modo deixava de lado as questões relativas a "geração automática" de códigos dedicados, como na abordagem original de IG141.

Foi a partir desta idéia, (que os pesquisadores de Berkeley passaram a denominar por "listas simbólicas de endereços" em IA11), que se deu origem este trabalho atual.

Cabe notar, que a abordagem de IA11, assim como a original de Gustavson, partilhavam do mesmo problema de crescimento "explosivo" do espaço de armazenamento, e que nas porções próximas ao final do processo de eliminação (onde sub-matrizes "quase densas" começam a se fazer notar), tal ordem de crescimento é ditada por um termo em função de  $n^3$  (como foi visto ao final da seção I.2).

A abordagem por "listas simbólicas de endereços" se mostrou portanto "passível a críticas" (como em IG11), por exibir um dispêndio de memória, um tanto quanto exacerbado, mesmo para os padrões atuais, (em máquinas com centenas de megabytes de memória, como a utilizada pelo grupo de Berkeley), sem contudo apresentar uma sensível redução dos tempos computacionais.

Visando de certa forma "controlar" o crescimento exacerbado da lista de endereços, foi proposta uma primeira alternativa como "ponto de corte", e que no trabalho original IA11, se baseava em levar adiante o processo por listas simbólicas de endereços, até o surgimento de uma porção "quase densa", na sub-matriz restante (ainda por eliminar), quando a partir de então, técnicas "convencionais", dedicadas as demais fases passavam a ser aplicadas.



O ponto de onde este trabalho, retoma e concentra a atenção, é portanto no aprimoramento das idéias tanto de [A1], como [G14].

A primeira das técnicas a ser "extendida", nesta seção, é a de códigos simbólicos *loop free*, para uma classe extremamente particular de matrizes, e onde qualquer melhoria na eficiência computacional se faz presente, qual seja, nas matrizes "completamente densas".

A abordagem por listas simbólicas de endereços será o objeto de estudo e maior formalização em uma seção posterior, de modo que nos concentraremos a partir de agora até o final desta seção, em apresentar o que se poderia chamar um "paliativo" eficaz e "implementável" das idéias de Gustavson para o caso *denso*.

O que vem imediatamente à mente quando se pensa em uma abordagem no estilo de Gustavson para o caso denso, é no tamanho do código gerado, e que sem qualquer forma de "controle" ou de desvios condicionais, acabaria por dispendir um volume da ordem de  $n^3$  linhas de código ...

O último paragrafo contém uma resposta para esta indagação, pois percebe-se que a maior "rigidez" de um código ao estilo de Gustavson, é não permitir qualquer forma de desvio condicional, (além de não se utilizar variáveis temporárias, indexadores ou apontadores de qualquer natureza).

Dentre as características citadas acima, a que menos afetaria o desempenho final caso fosse "relaxada", seria a utilização "controlada" de desvios condicionais para trechos "recorrentes" do código.

O tipo de desvio assim considerado seria da forma de um "GOTO computado" (em FORTRAN), "case" (em PASCAL), ou "switch" (em C/C<sup>++</sup>), visto que a introdução de *loops* controlados explicitamente por índices (como "DO" ou "for" nas diversas linguagens), acabaria por eliminar uma das características básicas da idéia de Gustavson, qual seja, a de procedimentos da forma "*loop free*".

A pergunta que se pode fazer, e que acaba conduzindo a forma de abordagem proposta para se solucionar o problema é simplesmente:

• *Seria possível se aproveitar trechos de código loop free, utilizados para eliminar matrizes densas de uma dada dimensão, para matrizes também densas, porém de dimensão inferior ?*

Ou seja, o que se questiona é se o processo de eliminação (por alguma das alternativas apresentadas na seção I.1) possui alguma recorrência em termos de operações e acessos a posições de memória, que possam ser desta forma aproveitados em trechos "comuns" dentro código gerado para a eliminação de uma matriz densa com dimensão superior a da nova matriz em que se espera poder aproveitar o mesmo código.

Observando-se atentamente as 3 alternativas, percebe-se que pela forma intrínseca de evolução da alternativa I.1(C), (gerando-se a matriz de fatores, por atualizações na forma de "sub-matrizes"), que esta seria a mais propensa a se considerar numa primeira tentativa de se enquadrar a característica procurada, à uma nova metodologia de solução.

Na verdade é possível se aplicar a idéia de recorrência de código, para as 3 alternativas apresentadas, na seção I.1, porém, no presente trabalho, apenas a alternativa I.1(C) será considerada, por ser a de mais fácil assimilação, e a mais adequada para as sub-porções densas encontradas ao final do processo de eliminação.

O que se pode perceber na forma de acesso e geração de fatores da alternativa I.1(C) é que a cada etapa, se atualizam sub-matrizes de dimensão inferior a da etapa anterior.

Aí encontra-se a chave para uma implementação de idéias similares as de Gustavson, mas que explore as características recorrentes do código gerado, e desta forma venha conduzir a redução dos requisitos finais de tamanho de código à um valor aceitável.

A idéia básica, consiste em se observar que um código para a atualização por sub-matrizes, pode ser decomposto em 3 fases (em cada etapa básica do processo como um todo):

- Montar-se um vetor auxiliar contendo os fatores multiplicativos da coluna abaixo do elemento diagonal corrente, (a serem multiplicados pelos elementos normalizados à direita da diagonal na linha base, durante o processo de atualização de cada um dos elementos da sub-matriz restante)
- Processar-se a atualização da sub-matriz definida pelos elementos da porção triangular superior abaixo da linha corrente.
- Normalizar-se os elementos da linha base corrente

Com isso, o que se percebe é que o código gerado para a atualização da sub-matriz restante a cada etapa, é uma porção "recorrente", do código mais geral, pois para cada valor decrescente de  $n$ , as sub-matrizes restantes a cada etapa, são idênticas (em termos estruturais), as sub-matrizes atualizadas nas etapas anteriores.

O que se torna necessário portanto, são apenas alguns "entry points" de modo a se definir o ponto de início do processamento de cada uma das sub-matrizes de dimensões inferiores, dentro do código geral para a eliminação de uma matriz de dimensão  $n$ .

O que se consegue desta forma, é uma redução considerável do tamanho de código, pois passa-se de um total da ordem de  $n^3$  linhas, para  $n^2$  (e que corresponde ao esforço computacional para a atualização de uma única etapa, com uma sub-matriz de dimensão  $n$ ).

● Ou seja, torna-se possível, mediante a introdução de alguns overheads de processamento, (a serem comentados mais adiante), a construção de um código "simbólico", onde para o caso particular de matrizes densas, o espaço total de armazenamento é proporcional ao número de elementos e não ao número de operações efetuadas (como no caso da abordagem original [G14] para o caso esparsa mais geral).

É conveniente portanto se ilustrar o fato, com um pequeno exemplo, para matrizes densas de dimensão 4, onde apresenta-se inicialmente um código ao estilo de Gustavson, e a seguir uma sequência de aprimoramentos ao mesmo, baseada nas novas idéias propostas.

Assume-se neste caso, a mesma representação adotada para matrizes esparsas (armazenadas simetricamente de forma contígua por linhas), como nas demais abordagens anteriores, e ilustrada (com os elementos abaixo da diagonal explicitados), a fim de se facilitar a apresentação a seguir.

Representação de uma matriz densa de dimensão 4

DI(1)	UN(1)	UN(2)	UN(3)
UN(1)	DI(2)	UN(4)	UN(5)
UN(2)	UN(4)	DI(3)	UN(6)
UN(3)	UN(5)	UN(6)	DI(4)

**Codificação III.1(a)**

Loop free à la Gustavson

```

DI(2) = DI(2) - UN(1) * UN(1) / DI(1)
UN(4) = UN(4) - UN(1) * UN(2) / DI(1)
UN(5) = UN(5) - UN(1) * UN(3) / DI(1)

DI(3) = DI(3) - UN(2) * UN(2) / DI(1)
UN(6) = UN(6) - UN(2) * UN(3) / DI(1)

DI(4) = DI(4) - UN(3) * UN(3) / DI(1)

UN(1) = UN(1) / DI(1)
UN(2) = UN(2) / DI(1)
UN(3) = UN(3) / DI(1)

DI(1) = 1 / DI(1)

DI(3) = DI(3) - UN(4) * UN(4) / DI(2)
UN(6) = UN(6) - UN(4) * UN(5) / DI(2)

DI(4) = DI(4) - UN(5) * UN(5) / DI(2)

UN(4) = UN(4) / DI(2)
UN(5) = UN(5) / DI(2)

DI(2) = 1 / DI(2)

DI(4) = DI(4) - UN(6) * UN(6) / DI(3)
UN(6) = UN(6) / DI(3)

DI(3) = 1 / DI(3)

DI(4) = 1 / DI(4)

```

Nesta codificação à princípio não se percebe nenhuma recorrência "explícita" no código associado.

A estratégia para solução, está em se lançar mão de um vetor auxiliar  $W$ , de dimensão  $n$  e que a cada etapa passará a conter uma cópia dos valores numéricos da linha corrente associada, e que serão utilizados para se atualizar a sub-matriz restante (ao invés de se utilizar as posições de memória originais dos elementos da linha corrente).

Neste caso, a fim de facilitar a compreensão, tomando-se a liberdade de representar a seqüência de vetores  $W^k$  (com o índice  $k$  associado à dimensão da sub-matriz corrente), dispondo-as estruturalmente nas posições correspondentes na matriz original ao longo de cada etapa, teremos:

Representação da seqüência de vetores  $W$

DI(1)	$W^4(1)$	$W^4(2)$	$W^4(3)$
$W^4(1)$	DI(2)	$W^3(2)$	$W^3(3)$
$W^4(2)$	$W^3(2)$	DI(3)	$W^2(3)$
$W^4(3)$	$W^3(3)$	$W^2(3)$	DI(4)

No código associado a seguir, tomou-se a liberdade de introduzir-se "labels" em determinados pontos, e que como veremos mais adiante (no próximo aprimoramento do código), constituem uma das peças fundamentais que viabilizam a exploração das recorrências contidas intrinsecamente no processo de eliminação por sub-matrizes, para o caso denso.

Percebe-se também que na verdade não é necessário uma "seqüência" de vetores  $W^k$  distintos, mas apenas um único vetor  $W$ , reescrito ao longo de cada etapa do processo, necessita ser empregado para a fatoração de toda a matriz.

O único ponto em que a princípio não se consegue evitar nesta primeira abordagem, é ter que se lançar mão de trechos aparentemente "repetitivos", de inicialização do vetor  $W$ , e de normalização dos elementos ao final da atualização por submatrizes, e que podem ser contornados como veremos um pouco mais adiante.

## Codificação III.1(b)

Loop free usando vetor auxiliar W

$$\begin{aligned}
 & W^4(1) = UN(1) \\
 & W^4(2) = UN(2) \\
 & W^4(3) = UN(3) \\
 \alpha^4: & \quad DI(2) = DI(2) - W^4(1) * W^4(1) / DI(1) \\
 & \quad UN(4) = UN(4) - W^4(1) * W^4(2) / DI(1) \\
 & \quad UN(5) = UN(5) - W^4(1) * W^4(3) / DI(1) \\
 \beta^4: & \quad DI(3) = DI(3) - W^4(2) * W^4(2) / DI(1) \\
 & \quad UN(6) = UN(6) - W^4(2) * W^4(3) / DI(1) \\
 \gamma^4: & \quad DI(4) = DI(4) - W^4(3) * W^4(3) / DI(1) \\
 & \quad UN(1) = UN(1) / DI(1) \\
 & \quad UN(2) = UN(2) / DI(1) \\
 & \quad UN(3) = UN(3) / DI(1) \\
 & \quad DI(1) = 1 / DI(1) \\
 & \\
 & \quad W^3(2) = UN(4) \\
 & \quad W^3(3) = UN(5) \\
 \beta^3: & \quad DI(3) = DI(3) - W^3(2) * W^3(2) / DI(2) \\
 & \quad UN(6) = UN(6) - W^3(2) * W^3(3) / DI(2) \\
 \gamma^3: & \quad DI(4) = DI(4) - W^3(3) * W^3(3) / DI(2) \\
 & \quad UN(4) = UN(4) / DI(2) \\
 & \quad UN(5) = UN(5) / DI(2) \\
 & \quad DI(2) = 1 / DI(2) \\
 & \\
 & \quad W^2(3) = UN(6) \\
 \gamma^2: & \quad DI(4) = DI(4) - W^2(3) * W^2(3) / DI(3) \\
 & \quad UN(6) = UN(6) / DI(3) \\
 & \quad DI(3) = 1 / DI(3) \\
 & \\
 & \quad DI(4) = 1 / DI(4)
 \end{aligned}$$

O que se percebe no código acima, tendo em vista os comentários já apresentados, é que à cada atualização de sub-matrizes de dimensão  $k$  menor, as operações de atualização dos elementos são na verdade as mesmas já "codificadas" durante a atualização de uma etapa anterior (a menos da divisão por um elemento diagonal distinto), como por exemplo as operações compreendidas de  $\beta^3$  até  $\gamma^3$  e que correspondem as operações indo desde  $\beta^4$  até  $\gamma^4$ .

O que se percebe portanto, é que apenas um único trecho de código, correspondente a atualização da sub-matriz de tamanho máximo considerado, precisa ser escrito, com os demais sub-trechos, executados mediante desvios apropriados.

## Codificação III.1(c)

Recorrente usando vetor auxiliar W

```

      k = 4
      W(1) = UN(1)
      W(2) = UN(2)
      W(3) = UN(3)
      GOTO  $\alpha$ 
 $\delta^4$ :  UN(1) = UN(1) / DI(1)
        UN(2) = UN(2) / DI(1)
        UN(3) = UN(3) / DI(1)
        DI(1) = 1 / DI(1)

      k = 3
      W(2) = UN(4)
      W(3) = UN(5)
      GOTO  $\beta$ 
 $\delta^3$ :  UN(4) = UN(4) / DI(2)
        UN(5) = UN(5) / DI(2)
        DI(2) = 1 / DI(2)

      k = 2
      W(3) = UN(6)
      GOTO  $\gamma$ 
 $\delta^2$ :  UN(6) = UN(6) / DI(3)
        DI(3) = 1 / DI(3)

      k = 1
 $\delta^1$ :  DI(4) = 1 / DI(4)

      STOP

 $\alpha$ :   DI(2) = DI(2) - W(1) * W(1) / DI(4-k+1)
        UN(4) = UN(4) - W(1) * W(2) / DI(4-k+1)
        UN(5) = UN(5) - W(1) * W(3) / DI(4-k+1)
 $\beta$ :   DI(3) = DI(3) - W(2) * W(2) / DI(4-k+1)
        UN(6) = UN(6) - W(2) * W(3) / DI(4-k+1)
 $\gamma$ :  DI(4) = DI(4) - W(3) * W(3) / DI(4-k+1)

      GOTO ( $\delta^1, \delta^2, \delta^3, \delta^4$ ) k

```

O que se pode melhorar no código acima, são dois fatos: Evitar-se as divisões por  $DI(4-k+1)$  no trecho recorrente, simplesmente lançando mão de um segundo vetor auxiliar V, contendo os valores "normalizados" dos elementos associados à

linha base corrente sendo utilizada para a atualização, e tentar de algum modo, eliminar a codificação "redundante" dos trechos de inicialização e normalização dos elementos, efetuada até então "explicitamente" (elemento a elemento).

A solução para o que foi comentado acima é apresentada a seguir, com algumas considerações finais sobre o código em questão, encerrando-se assim esta seção em que se objetivou a apresentação de técnicas visando o aumento da eficiência dos códigos de eliminação esparsa, e no qual, (como pode-se perceber ao longo do primeiro capítulo), se inclui o tratamento eficiente de matrizes densas, como um de seus casos particulares.

### Codificação III.1(d)

### Recorrente com normalização em V

```

SUBROUTINE DNSFAT (m, UN, DI)
PARAMETER (n = 4)
DIMENSION UN(*), DI(*), W(n), V(n)
j = ((n ** 2 - 3 * n) - (m ** 2 - 3 * m)) / 2
k = m

```

---

```

δ:      DI(n-k+1) = 1 / DI(n-k+1)
        DO ω i = (n-k+1), (n-1)
            W(i) = UN(i+j)
            UN(i+j) = UN(i+j) * DI(n-k+1)
            V(i) = UN(i+j)

```

```

ω:      CONTINUE
        j = j + (k-2)
        GOTO (ε, γ, β, α) k

```

```

ε:      RETURN

```

---

```

α:      DI(2) = DI(2) - W(1) * V(1)
        UN(4) = UN(4) - W(1) * V(2)
        UN(5) = UN(5) - W(1) * V(3)
β:      DI(3) = DI(3) - W(2) * V(2)
        UN(6) = UN(6) - W(2) * V(3)
γ:      DI(4) = DI(4) - W(3) * V(3)

```

---

```

k = k - 1
GOTO δ
END

```



No código acima, especificamente escrito para o processamento de matrizes de dimensão até 4, (e apresentado na forma de uma subrotina),  $m$  é um parâmetro fornecido, indicando a dimensão da sub-matriz a ser fatorada, com UN e DI contendo originalmente os valores de A e retornando ao final do processo, os fatores da matriz U.

O código em questão, mediante a utilização da expressão analítica para o valor inicial de  $j$  (cuja dedução não será apresentada), permite que se inicialize o processo a partir de qualquer sub-matriz de dimensão igual ou inferior a  $n$ .

O processo de eliminação da matriz densa de dimensão  $m$  termina, quando  $k$  assume o valor unitário no final do ciclo de repetições efetuadas.

A utilização do procedimento apresentado acima para matrizes de dimensão inferior à máxima  $n$  (para o qual o código foi projetado), merece um pequeno comentário, visto se lançar mão neste caso, de um artifício presente na maioria das linguagens de programação, qual seja a utilização de mecanismos como os de *EQUIVALENCE*, ou a passagem de endereços como parâmetro (no lugar de passagens por valor).

Tomemos como exemplo uma matriz de dimensão 3, a ser mapeada de forma se permitir a utilização do procedimento III.1(d), apresentando-se a disposição estrutural de seus elementos como abaixo:

DI(1)	UN(1)	UN(2)	UN(3)
	$a_{11}$	$a_{12}$	$a_{13}$
	DI(2)	UN(4)	UN(5)
		$a_{22}$	$a_{23}$
		DI(3)	UN(6)
			$a_{33}$
			DI(4)

Percebe-se que o que muda, é a posição inicial do primeiro elemento efetivamente utilizado tanto em DI, como em UN, visto que a parcela desde DI(1) até UN(3) no exemplo em questão não será acessada em nenhum momento durante o processo de solução.

O que se precisa portanto é na passagem de parâmetros, durante a chamada da subrotina DNSFAT, se especificar um

endereço "fictício" para o início do vetor UN (passado como parâmetro no programa principal), de tal sorte que a primeira posição efetivamente utilizada em UN (vista de dentro da subrotina), corresponda à UN(4).

Da mesma forma, o endereço inicial passado como parâmetro para o vetor DI, deve ser tal que faça corresponder a posição relativa ao primeiro elemento diagonal (da matriz 3x3 considerada), à posição DI(2) dentro do código de fatoração.

Assumindo-se que no programa principal, os vetores associados a UN e DI sejam denotados por U e D, a solução para o problema de mapeamento em questão se resume em se chamar:

$$ip_U = 1 - ((n ** 2 - n) - (m ** 2 - m)) / 2$$

$$ip_D = m - n + 1$$

CALL DNSFAT (m, U(ip\_U), D(ip\_D))

Vale ressaltar, que numa linguagem como FORTRAN, a passagem de um elemento de vetor como parâmetro, como no caso de U(ip\_U) ou D(ip\_D), efetivamente passa o endereço da posição correspondente ao elemento, que por sua vez, dentro da subrotina, acabará sendo interpretado como a posição do primeiro elemento do vetor associado.

O mapeamento a ser efetuado corresponde portanto à:

$$\begin{array}{cccc} U(ip_U) & \dots & U(1) & \dots & U((m^2-m)/2) \\ UN(1) & \dots & UN(2-ip_U) & \dots & UN((n^2-n)/2) \end{array}$$

E para o exemplo (com m = 3 e n = 4) em questão:

$$\begin{array}{cccc} U(-2) & \dots & U(1) & \dots & U(3) \\ UN(1) & \dots & UN(4) & \dots & UN(6) \end{array}$$

Com o mapeamento efetuado para o vetor DI sendo:

$$\begin{array}{cccc} D(0) & \dots & D(1) & \dots & D(3) \\ DI(1) & \dots & DI(2) & \dots & DI(4) \end{array}$$

Alguns fatos com relação ao tamanho de código gerado, se

fazem necessários notar, especialmente com relação a fase de retro-substituições, que também pode ser implementada utilizando-se técnicas similares as apresentadas nesta seção.

No caso da retro-substituição, o espaço final ocupado pelo código é da mesma ordem que o de uma abordagem como III.1(d), ou seja da ordem de  $(n^2 - n)/2$  linhas de código em linguagens de alto nível, como FORTRAN, PASCAL, ou C.

Outro comentário que se pode fazer, é que no caso da retro-substituição "esparsa", embora o código gerado acabe sendo particular e dedicado, (apenas para matrizes com a mesma disposição estrutural de elementos não nulos que a da matriz utilizada para a geração), o espaço final de código gerado, é de certa forma "tolerável", pois no pior caso acabaria sendo da ordem de  $n^2$  linhas de código caso a matriz de fatores resultantes fosse densa, e que mesmo assim, ainda é proporcional ao número total de elementos, e ao espaço utilizado para o seu armazenamento.

Assumindo-se uma densidade suficientemente reduzida para a matriz de fatores, percebe-se que a implementação de abordagens simbólicas para a fase de retro-substituições, (e que não serão abordadas neste trabalho), é uma área em que ainda se pode cogitar em estender algumas das metodologias apresentadas (para a fase de eliminação), com real chance de se obter bons resultados.

Uma potencial área seria por exemplo, a exploração de características do tipo "supernodal", e que também se fazem notar nesta fase do processamento em particular.

Retornando a etapa de eliminação, o que se apresenta a seguir, é o que se poderia chamar de uma extensão ainda "mais implementável" das idéias de Gustavson, mas que de certa forma se afasta um pouco do objetivo original da técnica, qual seja a de códigos puramente *loop free*.

O aprimoramento proposto tendo em vista uma redução ainda maior do número total de linhas de código, é considerar-se a introdução de *loops* indexados, para se efetuar as operações de atualização de cada uma das linhas das sub-matrizes restantes.

Assim, ao invés de se efetuar cada uma destas operações elemento a elemento, como na abordagem III.1(d), o que se pode utilizar no lugar do trecho de código correspondente as

linhas indo de  $\alpha$  até  $\gamma$  no procedimento anterior, é substituí-las por:

**Codificação III.1(e)**

Recorrente com operações indexadas

---

$\alpha$ :             $DI(2) = DI(2) - W(1) * V(1)$   
                   DO  $\alpha'$  p = 4, 5  
 $\alpha'$ :             $UN(p) = UN(p) - W(1) * V(p-2)$

$\beta$ :              $DI(3) = DI(3) - W(2) * V(2)$   
                   DO  $\beta'$  p = 6, 6  
 $\beta'$ :              $UN(p) = UN(p) - W(2) * V(p-3)$

$\gamma$ :             $DI(4) = DI(4) - W(3) * V(3)$

---

Mantendo-se o restante da codificação de DNSFAT exatamente como da forma apresentada em III.1(d).

● *O que esta nova abordagem traz, é o benefício de uma redução ainda maior no espaço de código, e que com esta estratégia, passa a ter da ordem de apenas  $O(n)$  linhas de código, (ou mais especificamente  $3n$  linhas caso a implementação seja feita em alguma linguagem de alto nível).*

O que se percebe portanto, é que esta nova abordagem possui como vantagem um espaço de código extremamente reduzido, (comparado com uma abordagem original como III.1(a)), e em certas classes de arquiteturas, como as do tipo vetorial, por exemplo, seria inclusive mais eficientemente implementada, por se basear em operações indexadas (por meio de loops), do que em acessos puramente escalares a memória.

● *O fato é que existe portanto uma liberdade na escolha de como se implementar esta porção de código considerada, tendo se apresentado aqui, os dois polos cardeais, com acessos puramente escalares, ou indexados.*

O leitor mais atento, se reparar com um pouco mais de detalhe, perceberá que praticamente se retornou a um esquema de codificação usual para o processo de eliminação de matrizes densas, baseado em codificações por 3 níveis de loops aninhados como na alternativa I.1(C) por exemplo.

Na verdade, os resultados apresentados nesta seção,

poderiam ter sido obtidos mediante um caminho inverso, qual seja, desenrolando-se gradativamente cada um dos *loops* envolvidos na etapa de fatoração para o caso de matrizes densas.

As técnicas de "desenrolamento" de *loops* não serão consideradas neste trabalho em particular, visto terem sido amplamente apresentadas em [A8], e serem de ampla divulgação na literatura em geral, como em [D2].

O que se pode perguntar ao final desta seção, em que se considerou apenas o caso da aplicação de técnicas simbólicas (ao estilo de Gustavson), voltadas exclusivamente para matrizes densas, é se algo similar poderia ser "cogitado" para o caso esparso mais geral ...

Este será o tema central do próximo capítulo, e deixamos em suspenso até lá, quaisquer comentários ou referências a técnicas baseadas nos conceitos apresentados nesta seção.

O que nos concentraremos até o final deste capítulo, é em formalizar e apresentar extensões das técnicas convencionais e das baseadas em "listas simbólicas de endereços" [A1], [A8], procurando de certa forma, atingir o mesmo objetivo final, qual seja, o da redução dos *overheads* em termos de execução da fase numérica do processo de eliminação.

### III.2 Abordagem simbólica por listas de endereços

Nesta seção serão apresentadas extensões dos procedimentos IL.4(1) e IL.4(2), baseados na geração dos fatores por linhas e colunas como nas alternativas I.1(B) e I.1(A) apresentadas desde o início deste trabalho.

A técnica de eliminação Gaussiana via o uso de "listas simbólicas de endereços", passou a ser considerada na literatura, a partir do trabalho de [A1] na implementação de Algoritmos de Pontos Interiores para Programação Linear.

Anteriormente, trabalhos como os de [D1], citados por [D6], e baseados em alternativas "interpretadas" para o processo de fatoração, já haviam sido cogitados, sem contudo se chegar a uma padronização ou ampla aceitação de tais métodos, pelo simples fato de que a eficiência com que se conseguiu levar as estratégias tidas como "convencionais" (por não se basearem nas informações simbólicas), simplesmente terem alcançado um desempenho comparativamente próximo ao das abordagens híbridas e interpretadas de outrora.

A extensão do trabalho de [A1] se dará em duas etapas: Nesta seção, se apresentarão métodos baseados em listas simbólicas de endereços, e na próxima, métodos que exploram características "*supernodais*" (e que podem ser encarados como uma extensão do conceito de processamento por listas simbólicas), pelo que o autor deste trabalho convencionou denominar listas simbólicas "compactas", em distinção as listas simbólicas de endereços tradicionais, abordadas nesta seção.

A base para a construção de procedimentos baseados em ferramentas como as listas simbólicas de endereços, já foi lançada na seção II.4, onde se considerou as 3 alternativas para a adição esparsa de vetores.

A alternativa IL.4(C), se utiliza do ingrediente básico das implementações simbólicas a serem consideradas nesta seção, qual seja o conceito de "listas simbólicas de endereços", e que passaremos a definir mais formalmente a seguir.

**Definição III.2(1)                    Listas Simbólicas de endereços**

*Define-se por listas simbólicas de endereços, a seqüência global de índices (para todas as etapas do processo de eliminação) de todas as posições de memórias no armazenamento de cada porção da matriz de fatores sendo gerada, a serem efetivamente acessadas durante a contribuição de elementos de porções anteriores, visando a sua geração.*

Esta definição é propositalmente ambígua, no sentido de não se especificar "qual porção" efetivamente sendo gerada, o que dá margem portanto a duas novas definições:

**Definição III.2(2)                    Listas Simbólicas por "linhas"**

*Define-se por listas simbólicas de endereços com geração por linhas, a seqüência global de índices (para todas as etapas do processo de eliminação) de todas as posições de memórias no armazenamento de cada linha da matriz de fatores sendo gerada, a serem efetivamente acessadas durante a contribuição de elementos de linhas anteriores, visando a sua geração.*

**Definição III.2(3)                    Listas Simbólicas por "colunas"**

*Define-se por listas simbólicas de endereços com geração por colunas, a seqüência global de índices (para todas as etapas do processo de eliminação) de todas as posições de memórias no armazenamento de cada coluna da matriz de fatores sendo gerada, a serem efetivamente acessadas durante a contribuição de elementos de linhas anteriores, visando a sua geração.*

O que se pode notar das definições acima, é que o processo de contribuição no segundo caso considerado, continua a ser efetuado por linhas, embora a geração dos fatores seja feita por colunas.

Este é um dos detalhes da "mecânica" da alternativa I.1(A), até então ainda não comentado ou explorado neste trabalho, e que abordaremos um pouco mais adiante.

A idéia básica que pode ser aplicada ao procedimento convencional II.4(1) baseado no uso do vetores de trabalho

expandidos, é se tentar de alguma forma suprimir esta estrutura auxiliar, e se efetuar as operações de subtração das contribuições dos elementos de linhas anteriores, diretamente sobre as posições de memória da linha corrente.

Esta por exemplo é a base da alternativa IL4(C) para se efetuar o processo de soma de 2 vetores U e V no formato esparsos, com o vetor V estando associado a qualquer uma das linhas anteriores a contribuírem na linha base corrente, e o vetor U, correspondendo justamente a representação desta linha base sendo gerada.

Todas as demais etapas do procedimento IL4(1) permanecem inalteradas. O que efetivamente se muda é simplesmente suprimir os *loops* de compactação e descompactação no vetor de trabalho W, e no lugar do *loop* mais interno baseado em acessos indiretos a este vetor, substituí-lo por um *loop* como o da alternativa IL4(C).

Alguns comentários sobre este novo procedimento a ser apresentado a seguir, se fazem notar.

A lista simbólica de endereços necessita conter apenas os endereços das posições correspondentes em UN na linha base corrente, porque simplesmente os elementos da linha contribuinte "l" a serem subtraídos da linha "i", estão na verdade armazenados de forma sequencial, e a única posição realmente "não determinística" (a não ser em casos especiais, que serão vistos na próxima seção), é a corresponde à posição do elemento na linha "i" relativa ao elemento da linha "l", subtraído.

Outro detalhe a se considerar, é que a lista simbólica assim definida nesta seção, abrange todo o processo de fatoração, razão pela qual, o incremento para a próxima posição da lista, efetuado mediante  $nadr = nadr + 1$  ser levado adiante no *loop* mais intermediário de todo processo, o que evidencia que o tamanho das listas simbólicas de endereços (por linhas ou colunas), perfazem um comprimento da ordem do número total de operações de subtração (ou adição) efetuadas em todo o processo de eliminação esparsa.



**Procedimento III.2(1)      Geração linhas (Listas Simbólicas)**

*(posição inicial na lista de endereços)*  
 nadr  $\leftarrow$  1

*(para cada linha corrente "i" à eliminar)*

**para i de 1 até n faça**

*(inicialização do apontador de posições iniciais)*  
 IUP(i)  $\leftarrow$  IU(i)

*(inicialização do valor da diagonal)*  
 piv  $\leftarrow$  DI(i)

*(para cada linha "l" a ser subtraída da corrente)*

**para k de IUT(i) até IUTF(i) faça**

*(linha a ser subtraída de "i")*  
 l  $\leftarrow$  JUT(k)

*(posição inicial em "l")*  
 iuc  $\leftarrow$  IUP(l)

*(próxima posição inicial)*  
 IUP(l)  $\leftarrow$  IUP(l) + 1

*(fator multiplicativo)*  
 um  $\leftarrow$  UN(iuc) \* DI(l)

*(atualização da diagonal)*  
 piv  $\leftarrow$  piv - UN(iuc) \* um

*(normalização)*  
 UN(iuc)  $\leftarrow$  um

*(subtração da linha "l" da linha corrente "i")*

-----  
**para j de iuc + 1 até IUF(l) faça**

UNCLSTADD(nadr)  $\leftarrow$  UNCLSTADD(nadr) - UN(j) \* um  
 nadr  $\leftarrow$  nadr + 1

-----

**fim para j**

**fim para k**

*(inversão do elemento diagonal)*  
 DI(i)  $\leftarrow$  1 / piv

**fim para i**

A grande vantagem do procedimento apresentado, em relação a forma convencional baseada em vetores de trabalho, é que dispensa-se as descompactações/compactações como já comentado, mas principalmente, porque o *loop* mais interno do novo procedimento acessa "diretamente" os valores da linha base corrente sendo gerada, sem precisar recorrer a qualquer forma de transferência de valores para estruturas auxiliares,

como ocorre com o uso de vetores de trabalho expandidos por exemplo.

Passaremos a considerar agora um procedimento análogo ao anterior, só que com a geração dos fatores por colunas, (como na abordagem II.4(2)).

Uma diferença no entanto se faz com relação ao procedimento convencional, pois neste primeiro caso, um processo de "acumulação de contribuições" é efetuado em cada etapa do processo, adicionando-se cada uma das contribuições a serem posteriormente aplicadas à linha corrente.

Já no procedimento simbólico a ser apresentado a seguir, a estratégia de implementação adotada se baseia na técnica de "subtração dos elementos" de cada linha anterior diretamente dos elementos na linha base corrente.

As diferenças entre o procedimento III.2(2) e o II.4(2) são poucas, como o leitor poderá notar, e que leva a uma observação deveras interessante:

● *Em arquiteturas "não paginadas" a eficiência computacional do procedimento convencional II.4(2) se aproxima muito da que se pode obter com o procedimento simbólico III.2(2) apresentado nesta seção.*

A razão para este tão "eloquente" comentário, se dá pelo simples fato de os únicos *overheads* existentes no procedimento convencional, serem as operações adicionais:

$$\begin{aligned} & \dots \leftarrow \dots - W(i) \\ & W(i) \leftarrow 0 \end{aligned}$$

E que só são efetuadas uma única vez, por cada elemento não nulo da matriz de fatores.

A menos de casos extremamente esparsos, com linhas com menos de 3 elementos não nulos por linha, onde o número de operações de ponto flutuante de cada etapa  $i$  do processo seja portanto da ordem de poucas operações (3 ou 4 por exemplo), o *overhead* adicional da subtração de  $W(i)$  e a subsequente "zeragem" desta posição, não chegam a contribuir de forma expressiva sobre o tempo total de computação da fase de eliminação como um todo.

Percebe-se claramente que o *loop* mais interno de ambos os procedimentos é similar, com a única diferença se dando

pelo fato de que no procedimento convencional, o vetor de trabalho  $W$  ser acessado, enquanto que no procedimento simbólico, uma posição de memória de  $UN$  ser diretamente acessada.

Dai a inclusão no comentário da "salvaguarda" com relação a arquiteturas "não paginadas", pois nas máquinas onde o acesso à dados da memória se baseie numa estrutura como a de páginas, (sendo portanto compartilhado com outros processos rodando na máquina, de modo que apenas uma parcela do espaço de memória abrangido pelo código sendo executado, esteja disponível a cada momento fisicamente na memória principal), o procedimento convencional pode vir a acarretar um maior *overhead*, especialmente nas etapas iniciais do processo, pois na abordagem simbólica, os únicos elementos acessados são os relativos a digonal (DI) e a estrutura de fatores (UN), além de todos os demais vetores auxiliares comuns a ambos os códigos.

Desta observação, percebe-se que nas arquiteturas "não paginadas" não é compensador aplicar-se a técnica de geração dos fatores por colunas, mediante procedimentos baseados em "listas simbólicas de endereços".

Tal constatação será reanalisada na seção III.4, quando por ocasião oportuna, serão apresentadas alternativas e "critérios de corte" adequados para a seleção de cada um dos procedimentos abordados neste trabalho até então.

Os procedimentos simbólicos apresentados nesta seção poderiam ser ligeiramente aprimorados, aproximando-os da forma original (com geração por sub-matrizes) proposta em [A1], mas que por razões didáticas e de analogia com os procedimentos convencionais já apresentados no capítulo II, optou-se por sua apresentação seguindo uma notação mais coerente e uniforme ao longo de todo o texto.

Um melhoramento que pode ser introduzido é a remoção dos vetores auxiliares IUT, IUTF, e IUP, armazenando-se estas informações diretamente em LSTADD, o que acabaria por tornar este vetor o que se conceituará no capítulo IV como "lista de códigos" (por conter uma "codificação" de natureza híbrida, a ser reinterpretada durante a fase de solução).

Procedimentos baseados em "listas de códigos" mais eficientes estão por ser apresentados no próximo capítulo, de modo que não se concentrará esforços em se estender as técnicas simbólicas apresentadas nesta seção, no sentido de aprimorar sua eficiência a nível de menores tempos de execução.

**Procedimento III.2(2)      Geração colunas (Listas Simbólicas)**

*(posição inicial na lista de endereços)*

nadr  $\leftarrow$  1

*(para cada linha corrente "i" à eliminar)*

para i de 1 até n faça

*(inicialização do apontador de posições finais)*

IUP(i)  $\leftarrow$  IU(i)

*(inicialização do valor da diagonal)*

piv  $\leftarrow$  DI(i)

*(para cada linha "l" a ser subtraída da corrente)*

para k de IUT(i) até IUTF(i) faça

*(linha a ser subtraída de "i")*

l  $\leftarrow$  JUT(k)

*(posição final em "l")*

iuc  $\leftarrow$  IUP(l)

*(próxima posição final)*

IUP(l)  $\leftarrow$  IUP(l) + 1

*(fator multiplicativo)*

um  $\leftarrow$  UN(iuc)

*(normalização)*

UN(iuc)  $\leftarrow$  um \* DI(l)

*(atualização da diagonal)*

piv  $\leftarrow$  piv - UN(iuc) \* um

*(subtração da linha "l" da linha corrente "i")*

-----  
para j de IU(l) até iuc - 1 faça

UNCLSTADD(nadr))  $\leftarrow$  UNCLSTADD(nadr)) - UN(j) \* um

nadr  $\leftarrow$  nadr + 1  
-----

fim para j

fim para k

*(inversão do elemento diagonal)*

DI(i)  $\leftarrow$  1 / piv

fim para i

Uma indagação é lançada neste ponto, sendo objeto de estudo das próximas seções deste trabalho:

- *Seria possível de algum modo se "reduzir" (ou compactar) o tamanho dispendido na codificação simbólica necessária para a fatoração de uma matriz esparsa ?*

A resposta afirmativa a tal questão pode vir por duas abordagens distintas, mas que convergem para um único ponto: O conceito de *supernodes*, a ser apresentado na próxima seção.

### III.3 Introdução do conceito de supernodes

Como comentado na seção anterior, os caminhos para se chegar a uma abordagem via a utilização do conceito de *supernodes* pode se dar por alternativas completamente distintas.

A que será adotada nesta seção em particular, será mediante uma extensão do conceito de "listas simbólicas de endereços", mediante a "compactação" de algumas das informações armazenadas, especialmente na fase final do processo de fatoração, onde o crescimento do tamanho da lista se faz notar de forma mais acentuada quando da presença de linhas com estrutura "quase densa".

A idéia básica de "compactação" de informações não é nova em esparsidade, haja visto a seção I.3 onde se mencionou algumas das freqüentes tentativas de se "melhorar o que já está muito próximo do ótimo", em que as técnicas de compactação sempre se mostraram como presenças constantes.

Curiosamente, uma das primeiras técnicas de compactação adotadas amplamente pela literatura desde meados dos anos 70, serve de base para que se possa introduzir um conceito que sómente uma década e meia mais tarde, veio a ser definitivamente estendido e explorado de forma ainda mais eficiente como em [A11].

A compactação referida anteriormente é a do tipo "*Sherman*", apresentada na seção II.1, e que visa exclusivamente a redução do espaço de armazenamento, em detrimento de uma maior ineficiência no processo de eliminação (em termos de tempos de execução).

Ou seja, na época o que se pensava era em se conseguir uma redução significativa do espaço de armazenamento reservado para a matriz de fatores, em face das restrições de natureza puramente computacionais daquele período em questão, mais tipicamente "falta de memória" do que de tempo de CPU, Conde naturalmente poderia se deixar um "job" rodando madrugada adentro, ou ir se tomar um café enquanto se aguardava os resultados, ao passo que simplesmente não se conseguia "ganhar" espaços de memória adicionais simplesmente vindos "do nada", ou esperando-se ad eternum).

A idéia de *Sherman* era boa, e atendia as restrições da época, além de se continuar a utiliza-la até hoje, em abordagens, cujo fator espaço de armazenamento seja mais crítico do que a eficiência computacional em termos de tempo de CPU.

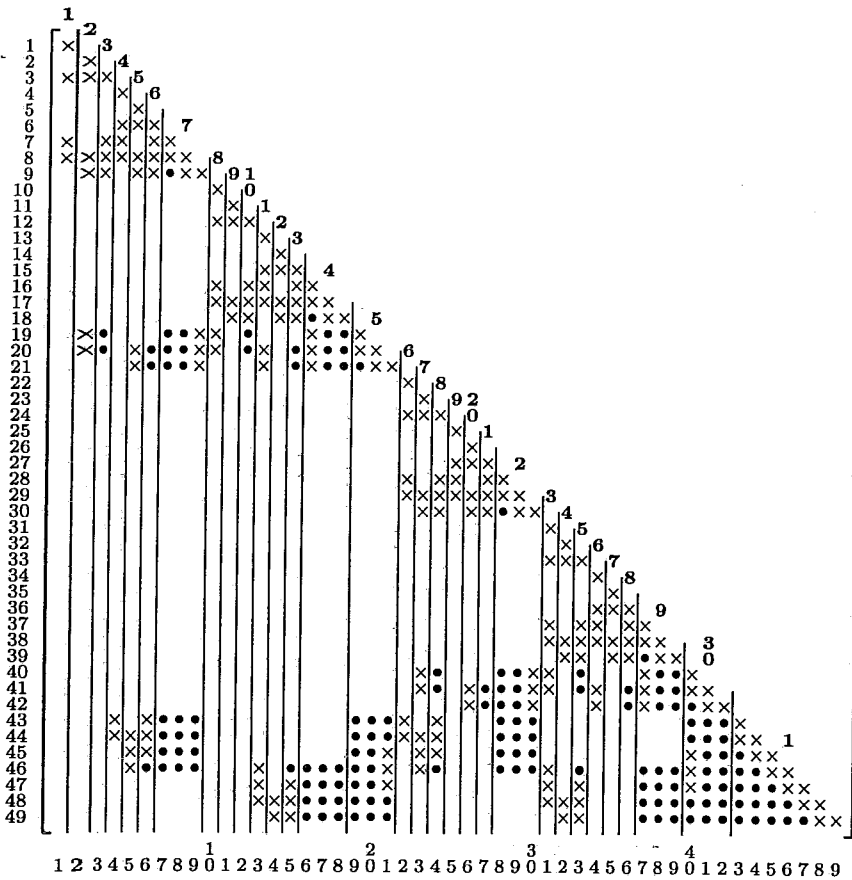
● *A pergunta que se pode fazer no entanto, é se seria possível estender a idéia de Sherman de modo a se alcançar não apenas uma compressão de espaço, mas concomitantemente um aumento da eficiência medida a nível de menores tempos de CPU.*

Deixando que o leitor realmente se convença de que mais uma resposta afirmativa a mais uma indagação levantada como esta, não tenha sido "provida unicamente dos céus", passaremos a analisar a essência por trás da compressão de *Shermann*, e que fornece a base para um dos conceitos mais "sólidos" e eficazes das abordagens esparsas dos dias atuais, (até então mencionado diversas vezes nas seções anteriores deste trabalho, mas cuja apresentação "formal" requer um pouco mais de atenção e detalhe).

Básicamente o que a compressão de *Sherman* se utiliza, é de uma noção de "recorrência" de natureza puramente "estrutural" e que pode ser observada inicialmente de forma "empírica" (como no exemplo apresentado na fig\_III(2), em que contrariando a notação adotada ao longo do texto, se lançou mão de uma representação por colunas da matriz  $U^T$  que simetricamente corresponde a uma representação por linhas de  $U$ ).

O que se percebe é que determinados grupos de linhas de  $U$  (colunas de  $U^T$ ) possuem estrutura praticamente idêntica, e que seriam portanto candidatas a se explorar num esquema de compressão como o de *Sherman*.

A questão é, de que modo tal "estrutura comum" entre as várias linhas de um mesmo grupo poderiam ser exploradas para se aumentar a eficiência do processo de fatoração como um todo.



fig\_III(2) - Presença de "supernodes"

O que a "similaridade" estrutural permite é na verdade o que se pode apresentar como a quinta alternativa de adição de vetores esparsos:

**Alternativa III.3(A)      Adição de vetores "supernodais"**

$k \leftarrow iu$

para  $j$  de  $iv$  até  $ivf$  faça

$U(k) \leftarrow U(k) + V(j)$

$k \leftarrow k + 1$

O que se assume na alternativa acima, é que a partir da posição inicial do vetor  $V$ , exista uma "coincidência estrutural perfeita" entre as colunas contendo elementos não nulos de ambos os vetores, o que permite que o processo de adição seja bastante simplificado, (se mostrando um dos mais eficientes possíveis para o caso esparsos, e só perdendo para técnicas ao estilo *loop free* como as apresentadas na seção III.1).

A alternativa III.3(A) pode ser vista como uma



simplificação da alternativa II.4(A) baseada na "varredura controlada" de ambos os vetores, no caso em que a priori se sabe que a coincidência de índices de colunas contendo elementos não nulos entre ambos os vetores é "perfeita", dispensando portanto os testes comparativos da alternativa por varredura tradicional.

● *O que a princípio não seria tão evidente se questionar, é se uma técnica baseada na alternativa III.3(A) apresentada nesta seção, poderia ser utilizada para estender uma alternativa por "listas simbólicas de endereços", como a II.4(C) por exemplo.*

A resposta já se supõe, será afirmativa, porém o ponto de fundamental importância, é se perceber detalhadamente de que modo tal pode ser efetuado, pois aí encontra-se uma das chaves para se chegar a nova metodologia proposta no capítulo IV e a compreensão do conceito de *supernodes* abordado nesta seção.

Visto que pelas definições III.2(1) à III.2(3), listas simbólicas de endereços no seu estrito senso, contém apenas informações relativas a endereços, (ou seja indicadores de posições de memória a serem acessadas), não se poderia levar adiante a idéia de "compactação" das informações, sem que tal requisito com relação a "apenas um único tipo de informação armazenada por lista" fosse relaxado.

O que se define agora é o que se convencionará chamar por "lista simbólica de códigos", cuja aplicação é bem mais geral e ampla do que a das listas simbólicas de endereços, (incluindo-as como um caso particular).

#### **Definição III.3(1)**

#### **Listas simbólicas de códigos**

*Define-se por "listas simbólicas de códigos" (voltadas para o processo de fatoraçoão esparsa), como listas de informações, de algum modo "codificadas" de tal sorte que mediante algum processo de decodificação futura, se possa reproduzir alguma seqüência de operações do processo de geração de fatores em alguma implementação "dedicada" baseada em métodos diretos de eliminação.*

A definição acima, é abrangente o bastante, para incluir

várias formas de codificações simbólicas apresentadas neste trabalho, mas de certa forma restrigente no sentido de limitar o contexto das listas simbólicas de códigos exclusivamente ao processo de fatoração esparsa, (tema central deste trabalho).

Listas simbólicas de informação codificada, podem ser utilizadas também no processo de retro-substituição esparsa, ou até mesmo na implementação de métodos iterativos, e que não serão considerados na presente abordagem.

Restringindo-se ao contexto da eliminação esparsa, vejamos de que modo se pode a partir da alternativa III.3(A), se chegar a uma lista simbólica de "códigos", com uma propriedade particular das mais desejáveis possíveis: ser "compacta".

O que se pode introduzir, em face do reconhecimento das notadas características de "similaridade" e seqüencialidade presentes em grupos de linhas com um mesmo padrão estrutural, (e que hesitamos ainda chamar por *supernodes*), é que neste caso em particular (no qual a alternativa III.3(A) pode ser empregada), se poderia na verdade de alguma forma especificar como uma das informações simbólicas adicionais (armazenadas em uma lista de códigos), a indicação da presença de um grupo de linhas para as quais o processo de adição de vetores pode ser efetuado de forma mais eficiente como em III.3(A).

Ou seja o que se considera aqui é a adoção de uma lista simbólica de "códigos", tendo como base uma lista simbólica de "endereços", na qual se toma a liberdade de se introduzir novos elementos em posições adequadas da mesma, de modo a se indicar uma forma especial de processamento, como os da forma III.3(A), como um dos casos especiais.

Uma lista assim construída é "híbrida" por construção, pois não contém apenas endereços, (visto que seria redundante se armazenar os endereços das posições de memória acessadas num processamento para o caso "supernodal", onde a seqüencialidade das posições pode ser trivialmente explorada mediante *loops* como os apresentados nesta seção).

É justamente a seqüencialidade das posições acessadas num processamento dedicado para os "supernodes" que se deseja explorar, de sorte que as informações relativas à endereços de memória necessitam ser incluídas apenas para os demais

casos gerais de adição de vetores de natureza estruturalmente distinta.

Como partindo-se apenas da posição inicial na linha base corrente, e do número de posições contíguas a se operar, ser informação mais do que suficiente para se caracterizar toda a seqüência de operações relativas a subtração de uma linha por outra para o caso *supernodal*, a estratégia de codificação a ser apresentada é justamente a exploração desta característica.

Segue-se portanto uma primeira abordagem em que se explora fundamentalmente a seqüencialidade de acessos, de modo a se utilizar esta técnica "controladamente" para todas as linhas do processo de eliminação.

A explicação é simples: mesmo linhas que em princípio possuam padrões estruturais distintos, acabam contendo coincidências nos índices das colunas associadas, visto que a linha base por natureza do processo de eliminação, obrigatoriamente contém a estrutura das linhas anteriores dela subtraídas.

O que ocorre no caso de uma coincidência apenas "parcial" de índices (em oposição a coincidência "perfeita" no caso *supernodal*) é que algumas posições na linha base (correspondentes a elementos inexistentes na linha anterior sendo subtraída), necessitam ser ignoradas (simplesmente saltando-as), de modo a se atingir novamente um elemento na linha base correspondente a um elemento de alguma linha anterior a ser subtraída.

Percebe-se portanto que apenas mediante a especificação de 2 informações básicas: quantos elementos processar de forma *supernodal* (contígua), e quantos elementos "saltar" na linha base corrente até que a próxima coincidência estrutural de índices ocorra, são mais do que suficientes para se caracterizar todas as operações a serem efetuadas ao longo da eliminação.

É este primeiro procedimento, baseado numa lista de "códigos", (com as características consideradas até então), que será apresentado a seguir:

**Procedimento IIL3(1)      Lista de códigos (supernodais)**

```

ncod = 0
DO  $\alpha$  i = 1, n
    IUP(i) = IU(i)
    piv = DI(i)
    DO  $\gamma$  k = IUT(i), IUTF(i)
        l = JUT(k)
        iuc = IUP(l)

        IUP(l) = IUP(l) + 1
        um = UN(iuc) * DI(l)
        piv = piv - UN(iuc) * um
        UN(iuc) = um

        ij = IU(i)
        jl = iuc + 1
        iucf = iuf(l)
 $\gamma\gamma$         IF (jl .GT. iucf) GOTO  $\gamma$ 
            ncod = ncod + 1
            icod = LSTCOD(ncod)
            IF (icod .GT. 0) GOTO  $\beta\beta$ 
 $\gamma\gamma\gamma$         jf = jl - icod
            DO  $\beta$  j = jl, jf
                UN(ij) = UN(ij) - UN(j) * um
                ij = ij + 1
 $\beta$             CONTINUE
                jl = jf + 1
            GOTO  $\gamma\gamma$ 
 $\beta\beta$         ij = ij + icod
            ncod = ncod + 1
            icod = LSTCOD(ncod)
        GOTO  $\gamma\gamma\gamma$ 
 $\gamma$         CONTINUE
    DI(i) = 1 / piv
 $\alpha$     CONTINUE

```

No procedimento acima, um valor negativo para elementos de LSTCOD indica (em valor absoluto) o número de posições contíguas a serem processados de forma *supernodal* no loop  $\beta$  do trecho de código em FORTRAN listado.

Já um valor positivo em LSTCOD, indica na verdade um "incremento" a se dar na posição da linha base corrente de modo a se buscar a próxima coincidência estrutural de colunas, onde novamente um processamento da forma *supernodal* no *loop*  $\beta$  é efetuado.

Percebe-se que este código é baseado inteiramente numa única forma de processamento para o *loop* mais interno, e que se mostra a mais eficiente de todas para o caso esparso (especialmente o *supernodal*), pois elimina acessos indiretos à memória presentes nas abordagens até então apresentadas, onde acessos da forma UN(LSTADD(nadr)) ou W(JUCJ)) tinham de ser empregados por justamente o padrão de acesso a estes vetores ser "não determinístico" (a princípio).

No caso em que a presença de linhas com padrões estruturais idênticos se faz notar com maior destaque sobre as demais, (de tal modo que a maior parcela do esforço computacional dispendido esteja concentrado no processamento destas linhas), a forma de abordagem proposta acima se mostra inteiramente válida, pois explora nitidamente uma característica que viabiliza um aumento da eficiência computacional em ambos os sentidos: economia do espaço de armazenamento (com a compactação obtida na especificação de "lotes" de elementos a processar de forma contígua), e redução dos *overheads* em tempos de execução, visto que não se precisa lançar mão de estruturas auxiliares, acessando-se diretamente os elementos sendo gerados na linha base, e o que é certamente ainda melhor, e determinante direto do aumento final desta eficiência, com o acesso direto aos elementos da linha corrente se processando de forma sequencial (contígua).

Cabe notar porém, que no caso onde não ocorra predominância de processamentos tipicamente da forma *supernodal*, o procedimento acima não é o mais adequado, e para tal. o que será exibido a seguir, é mais recomendado.

**Procedimento IIL3(2)      Lista de códigos (híbrida)**

```

ncod = 0
nadr = 0

nm = -n
nm2 = nm * 2

DO  $\alpha$  i = 1, n
    IUP(i) = IU(i)
    piv = DI(i)
    iuj = IU(i) - 1
    DO  $\gamma$  k = IUT(i), IUTF(i)
        l = JUT(k)
        iuc = IUP(l)

        IUP(l) = IUP(l) + 1
        um = UN(iuc) * DI(l)

        umn = - um

        piv = piv + UN(iuc) * umn
        UN(iuc) = um

        ij = iuj
        jl = iuc + 1
        iucf = IUF(l)
 $\gamma\gamma$  IF (jl .GT. iucf) GOTO  $\gamma$ 
            ncod = ncod + 1
            icod = LSTCOD(ncod)
            IF (icod .LE. 0) GOTO  $\beta$ 
            DO  $\delta$  j = jl, icod
                nadr = nadr + 1
                UNCLSTADD(nadr) =
                    UNCLSTADD(nadr) + UN(j) * umn
 $\delta$  CONTINUE
                ij = LSTADD(nadr)
                jl = icod + 1
            GOTO  $\gamma\gamma$ 
 $\beta$  IF (icod .LT. nm) GOTO  $\beta\beta$ 
        jf = jl - icod
        DO  $\varepsilon$  j = jl, jf
            ij = ij + 1
            UN(ij) = UN(ij) + UN(j) * umn
 $\varepsilon$  CONTINUE
            jl = jf + 1
        GOTO  $\gamma\gamma$ 

```

(continua na próxima página)

(continuação da página anterior)

```

ββ          IF (icod .LT. nm2) GOTO βββ
            jf = jl - (icod - nm)
            DO δδ j = jl, jf
                nadr = nadr + 1
                UN(LSTADD(nadr)) = UN(j) * umn
δδ          CONTINUE
            ij = LSTADD(nadr)
            jl = jf + 1
            GOTO γγ
βββ         jf = jl - (icod - nm2)
            DO εε j = jl, jf
                ij = ij + 1
                UN(ij) = UN(j) * umn
εε          CONTINUE
            jl = jf + 1
            GOTO γγ
γ           CONTINUE
            DI(i) = 1 / piv
α          CONTINUE

```

Este novo procedimento é baseado na utilização "híbrida" de listas de códigos e endereços, e é propositalmente bem mais complexo do que o procedimento anterior, por incluir fases de processamento dedicado até então ainda não consideradas neste trabalho.

Um dos tratamentos dedicados da porção de código apresentada acima é o voltado para a primeira vez em que se efetua uma operação de subtração sobre um elemento de *fill-in* ainda não acessado, ou seja quando o valor associado na estrutura de fatores resultantes ainda é inicialmente "nulo".

Neste caso, percebe-se que a operação de subtração é desnecessária, pois o valor a ser armazenado na posição correspondente em UN será  $0 - UN(j) * um \equiv -UN(j) * um$  ou simplesmente  $UN(j) * umn$  como considerado nos *loops* δδ e εε, e cuja a razão para se tomar um valor de sinal contrário à *um* (como *umn*) ser devido a desta forma se eliminar uma operação adicional de "inversão de sinal", que acabaria sendo efetuada em todos os acessos aos elementos da linha base gerada (o que em algumas arquiteturas é quase tão custoso quanto uma

operação de soma ou subtração em ponto flutuante).

A nova característica da abordagem "híbrida" assim proposta, é se poder explorar de forma eficiente, ambos os casos típicos de se encontrar durante as etapas de eliminação esparsa: linhas com padrão estrutural idêntico, ou linhas cujo padrão estrutural seja tal, que os acessos as posições correspondentes na linha base gerada sejam consideravelmente "dispersos" ao longo da mesma.

Para cada um destes casos considerados, se adota uma estratégia de ataque (dedicada), daí o código em questão, lançar mão tanto de listas simbólicas de "endereços", como de listas simbólicas de "códigos".

O tratamento mediante "listas simbólicas de endereços" se dá no trecho associado ao *loop*  $\delta$  do procedimento, ao passo que um tratamento da forma *supernodal* por "listas de códigos" (como no procedimento III.3(1) apresentado nesta seção), é efetuado nos trechos associados ao *loop*  $\epsilon$ .

As porções de código relativas aos *loops*  $\delta\delta$  e  $\epsilon\epsilon$ , correspondem a aplicação de ambas as abordagens consideradas para o acesso e geração dos elementos da linha base, porém incluindo um tratamento diferenciado e particular, apenas para a primeira vez em que um novo elemento de *fill-in* é gerado.

Uma única sutileza de codificação faz-se notar no procedimento acima, e que é o fato de que no caso de uma informação da forma *supernodal* na lista de códigos, o valor máximo do número de operações "sequenciais contíguas" a se efetuar na subtração de uma dada linha, ser sempre inferior a dimensão  $n$ . Por esta razão, códigos com valor superior a  $n$  na lista de códigos LSTCOD serem desta forma utilizados para denotar operações com tratamento diferenciado para a primeira operação sobre os elementos de *fill-in*, e valores inferiores a  $n$ , utilizados num tratamento da forma *supernodal* como já se apresentou nesta seção.

O tratamento diferenciado para a primeira operação sobre elementos de *fill-in* não é de uma necessidade ou obrigatoriedade "constante" para todas as classes de matrizes. Em muitas delas, tal processamento dedicado pode ser perfeitamente ignorado, tratando-se as operações de uma mesma forma, como nas demais abordagens usuais.



Em matrizes muito esparsas no entanto, o tratamento diferenciado para os *fill-in's* pode se mostrar um fator significativo na redução dos *overheads* de computação a nível de tempos de execução.

Percebe-se portanto, que com o uso de alternativas diferenciadas para os diversos padrões estruturais passíveis de se encontrar durante a etapa de eliminação, consegue-se um procedimento ainda mais "ajustado" ao processamento eficiente de matrizes esparsas, com as mais distintas características, indo desde os casos "super-esparsos", até os quase-densos ou completamente densos.

O que se pode ainda argumentar com relação a procedimentos híbridos puramente simbólicos, é que por serem "simbólicos", ou seja conterem alguma informação detalhada ao nível de cada uma das operações e acessos efetuados no *loop* mais interno do processo, o volume de codificação final, possa ainda exibir uma característica "explosiva" (por ser de algum modo, proporcional ao número de operações de ponto flutuante realizadas ao longo de toda a eliminação).

Em vista destas observações, é que na abordagem da próxima seção, consideram-se procedimentos "híbridos", porém incluindo-se neste caso, prioritariamente procedimentos baseados nas estratégias convencionais apresentadas no capítulo II.

Antes de passarmos a nova seção, mostra-se necessário porém, apresentar e definir formalmente o conceito "tão aguardado" ao longo do texto e desta seção em particular: o de "*supernodes*".

### Definição III.3(2)

### Supernodes

Define-se como um "*supernode*", um conjunto de linhas contíguas  $j, j+1, \dots, j+t$  (na estrutura de fatores  $U$  resultantes do processo de eliminação), tais que  $Struct(U_{k,*}) = Struct(U_{k+1,*}) \cup \{k+1\}$  para  $j \leq k \leq j+t-1$

Onde  $Struct(M_{i,*})$  denota a estrutura de índices de colunas associadas a cada linha  $i$  da representação da matriz de fatores  $U$  resultante, conforme apresentado na seção II.3.

Uma definição complementar a de "*supernodes*" mostra-se

adequada, e se refere a um processamento da forma "supernodal":

**Definição III.3(3)                      Processamento "Supernodal"**

*Define-se como um processo de eliminação da forma "Supernodal", como uma etapa particular do processo de geração de fatores, exclusivamente associada a geração de cada uma das linhas de um dado "supernode", (mediante combinações escalares de todas as linhas anteriores pertencentes ao referido "supernode" em questão).*

O que se percebe pela definição acima, é que o que os procedimentos apresentados nesta seção buscaram até então, foi "enxergar" o processo de eliminação segundo uma ótica relativamente "miope", com respeito a exploração das propriedades "supernodais" como um todo.

O que se tratou até então, foram estratégias mais bem ajustadas para o processamento isolado de cada uma das linhas de fatores sendo geradas, uma a uma, em função de cada etapa básica *i* do processo de eliminação.

O conceito de *supernodes* é bem mais abrangente, pois envolve uma das bases para a seção posterior, onde se explora o processamento por "grupos contíguos" de linhas (de forma dedicada e diferenciada das demais).

O que abordagens explorando-se plenamente *supernodes*, juntamente com as estratégias da próxima seção permitem, é que se processem conjuntos de linhas de uma só forma, o que implica numa redução ainda maior de *overheads*, pois simplesmente em cada procedimento dedicado, sabe-se de antemão, o tipo de estrutura característica a ser eficientemente explorado.

Para se finalizar esta seção, o que se pode comentar, é que o conceito de *supernodes* é bem mais amplo, do que a sua definição "puramente estrutural" aparentemente revela.

O que se encontra de mais forte, por trás deste conceito é a natureza com que se pode implementar os "processamentos na forma supernodal" definidos em III.3(3).

O que se percebe desta definição, como uma característica inerente exclusivamente aos "*supernodes*" é que este processo de eliminação, pode ser feito de forma

inteiramente seqüencial e contígua, o que viabiliza uma eficiência de implementação muito próxima da máxima teórica possível, e que pode ser "quase" obtida no caso de um processamento exclusivamente denso, (como por exemplo nos mostra a seção III.1).

O grande "conceito" por trás da noção de *supernodes*, é que dentro de seu "escopo" (ou seja, das linhas que o constituem), o processo de implementação esparsa para a eliminação, poder ser efetuado de forma "completamente densa", uma vez que como todas as linhas da matriz de fatores são assumidas estarem representadas "seqüencialmente" de forma contígua (ou segmentada), todos os elementos em que se efetuarão operações, (ao longo da eliminação de cada uma das linhas do *supernode*), podem ser vistos como pertencendo a uma "matriz densa" (incluindo apenas os elementos não nulos de cada linha), e que existe apenas "ficticiamente", pois na verdade o carater "denso" da mesma, só se manifesta pela forma como se optou por armazenar as linhas em questão: numa forma seqüencial contígua, contendo apenas os elementos distintos de zero.

O que se acabou de comentar, dá margem a algumas novas colocações e possíveis "reinterpretações" do conceito de *esparsidade*, o que nos leva a algumas definições complementares:

**Definição III.3(4)                    Matriz "realmente" esparsa**

*Define-se uma matriz de fatores (resultante de um processo de eliminação Gaussiana) como "realmente" esparsa, quando a seqüência de acesso aos elementos de cada uma de suas linhas base sendo geradas, não puder ser efetuada segundo um padrão "determinístico" de forma contígua.*

**Definição III.3(5)                    Matriz "aparentemente" esparsa**

*Define-se uma matriz de fatores (resultante de um processo de eliminação Gaussiana) como "aparentemente" esparsa ou "supernodal", quando toda a seqüência de acesso aos elementos de cada uma de suas linhas base sendo geradas, puder ser efetuada segundo um padrão "determinístico" de forma contígua.*

**Definição III.3(6)****Grau de esparsidade**

*Define-se o "grau" de esparsidade de uma matriz de fatores (resultante de um processo de eliminação Gaussiana), como sendo o número de seus fatores não nulos (excluindo-se a diagonal) que não pertençam a algum "supernode", dividido pela grandeza formada pelo número de seus "supernodes" acrescida do número total de fatores da referida matriz.*

- Pelas definições acima, percebe-se por exemplo, que uma matriz de fatores "completamente densa", como havíamos definido até então, pode ser encarada como uma matriz "aparentemente" esparsa, por ser apenas um caso particular de matrizes "Supernodais" em que o número de *supernodes* é unitário.
- Percebe-se também, que o "grau" de esparsidade de uma matriz de fatores (como definido acima), é unitário para o caso de matrizes "realmente" esparsas, e nulo, para as matrizes "aparentemente" esparsas, o que nos leva a crer que matrizes de fatores "densas" são na verdade um mero caso particular de matrizes esparsas, e que na verdade o que se precisaria ensinar nos cursos de álgebra linear computacional e análise numérica, é o conceito de Esparsidade, por ser bem mais amplo, coerente, e englobar todos os demais "sub-casos" da forma mais igualitária possível.
- Em termos da "quantidade" de informação (necessária para se caracterizar todo o processo de eliminação), percebe-se que nada separa uma matriz "completamente densa" de uma "completamente diagonal" por exemplo, pois em ambos os casos o que se precisa especificar é apenas uma única informação: a sua dimensão  $n$ .

### III.4 Abordagem híbrida por janelas

Pelo que já se pode perceber ao longo deste trabalho, uma das chaves para uma implementação esparsa eficiente do processo de eliminação, está no "reconhecimento" de todas características estruturais da matriz sendo fatorada, e na sua plena "exploração", ou seja escolhendo a forma de tratamento mais adequada para cada uma das características encontradas.

Como se comentou no início do capítulo II, ("de nada adianta se escolher um bom algoritmo sem uma estrutura de dados adequada"), o contrário também se aplica, ou seja, de nada adianta uma boa estrutura de dados, sem os algoritmos adequados para a sua plena exploração.

Assim, nesta seção nos concentraremos em apresentar varias alternativas do ponto de vista "algorítmico" (a nível de implementação), e que serão aplicados porém a uma estrutura de dados básica "constante", (ou seja a representação seqüencial por linhas da matriz de fatores  $U$ , e adotada ao longo de todo o texto).

O que se pode notar, observando padrões estruturais típicos da disposição de fatores de matrizes esparsas das mais variadas áreas e aplicações, (como as apresentadas no apêndice final), é que pelo menos 4 padrões se destacam dos demais, e serão portanto observados com um pouco mais de atenção.

São tais padrões, numa ordem mais ou menos típica de se encontrar, nas diversas linhas dos fatores (em maior ou menor grau dependendo de cada matriz):

- Completamente Diagonal (ou seja com linhas na representação de  $U^T$  contendo apenas a diagonal)
- Típicamente Esparso (com um número aceitável de elementos por cada linha de  $U$ , e sem algum padrão "determinístico" reconhecível à priori)
- Supernodal (ou "quase denso" nas porções finais da matriz de fatores)
- Completamente Denso (como a última sub-matriz)

Assim, o que se mostra como uma alternativa bastante atrativa, é tentar se adequar procedimentos específicos para cada uma das porções da matriz de fatores consideradas.

Em alguns casos, como no padrão completamente Diagonal, o que se precisa fazer é muito pouco, (apenas se inverter o elemento da diagonal, e que desta forma só acabará sendo utilizado novamente no processo de retro substituição).

Cabe lembrar que o padrão Diagonal, pode vir a se repetir ou se fazer notar, em porções ou grupos distintos de linhas um pouco mais avante, (na maior parte das vezes, notadamente nas linhas iniciais, e até uma dimensão próxima à metade da dimensão original).

Outros padrões como o completamente Denso por exemplo, só podem ocorrer na porção final, (visto que deste ponto em diante, a sub-matriz restante torna-se completamente densa).

Os padrões mais típicos de se encontrar nas demais porções, são os que já se dispõe de ferramentas para trata-los adequadamente, qual seja, o uso de abordagens simbólicas ou convencionais para a porção tipicamente Esparsa, e o uso das técnicas *supernodais*, para as porções com esta característica, onde se pode destacar por exemplo a porção "quase densa" que na maioria das vezes antecede a porção completamente densa, nos casos em que esta chega a se manifestar notadamente.

A maior "liberdade de escolha", recai sobre a porção tipicamente Esparsa, pois como já se pode comentar, é nesta porção que se concentra o maior grau de "caoticidade" a princípio não determinístico do processo, no sentido de não se poder prever um padrão típico de acessos na linha base, (ao contrário do que ocorre no caso *supernodal* ou *denso* por exemplo).

Esta porção Esparsa é a porção mais difícil de ser "domada", e a que na hipótese de pouca ou nenhuma predominância das porções *supernodais* ou completamente *densas*, acaba se mostrando a determinante da eficiência final dos códigos de fatoração numérica.

- Numa abordagem convencional, incorre-se em *overheads* como por exemplo as etapas de descompactação/compactação do vetor de trabalho.
- Numa abordagem simbólica por "listas de endereços", percebe-se que a porção tipicamente esparsa é a única que não pode ser devidamente "compactada", dando margem assim a um crescimento indesejável do tamanho final da lista.
- *Ou seja, de alguma forma acaba por se incorrer em algum overhead no tratamento da porção "tipicamente esparsa", onde a "aleatoriedade" da disposição estrutural dos elementos da matriz de fatores se faz notar maciçamente e de um modo que poderia-se qualificar na melhor das hipóteses como "intratável" ...*

Bem, esta é a visão mais otimista, que se apresentou até então, e não convém comentar o que o leitor a esta altura já pode estar a cogitar ser um "novo aprimoramento" de tudo o que foi feito, conseguindo-se dessa forma, de algum modo "domar" a fase tipicamente Esparsa (ou ao menos coloca-la sob eixos bem mais tratáveis do que já se havia pensado até então).

O capítulo IV revela pois, mais alguns dos "mistérios" que tornam a área de esparsidade (na visão deste autor) como uma das mais privilegiadas em termos de riqueza de problemas e de possíveis abordagens para a sua solução (num moto-perpétuo aparentemente inesgotável ao menos até hoje).

Voltando a realidade do que já foi visto e apresentado neste trabalho, percebe-se que com o que se dispõe de ferramentas, é possível melhorar o desempenho final tanto a nível de economia de recursos de armazenamento, como aumento da eficiência a nível de tempos de execução.

Esta seção visa portanto, agrupar todas as abordagens até então apresentadas, em um "*framework*" comum, e que por ser dedicado à exploração das características de cada matriz a ser fatorada, possibilita que os requisitos computacionais dispendidos sejam portanto os mais ajustados possíveis ao problema sendo tratado.

O grau de liberdade que se oferece não é muito, pois como se percebe, apenas a porção tipicamente Esparsa tende a

se apresentar como um verdadeiro desafio à se determinar qual a melhor estratégia de solução.

O que irá se considerar no caso Esparsos são 4 possíveis abordagens, já apresentadas nas seções II.4 e III.2, qual sejam: a geração dos fatores (por linhas ou colunas), mediante as técnicas convencionais (via o uso de vetores de trabalho), ou mediante a utilização de listas simbólicas de endereços.

Como já havia se comentado na seção III.2, a utilização de um procedimento baseado em "listas simbólicas" com geração por colunas como o III.2(2), não se mostra adequado, pelo fato do dispêndio em termos de espaço de armazenamento para a informação simbólica, não ser compensador em termos do desempenho final obtido, (em comparação com um procedimento tradicional como o II.4(2) e que pode ser utilizado em lugar do procedimento simbólico, sem perda sensível da eficiência).

Deste modo descartando-se este método, ficamos com a escolha de por qual das outras 3 alternativas, se iniciar o processo de eliminação.

Como já se havia comentado ao final dos capítulos I e II, e na seção III.2 deste, percebe-se que o método de geração por colunas é o mais indicado para a fase inicial do processo de fatoração, visto que o volume de cálculo em função de cada etapa básica  $i$  do processo ser uma função crescente com o avançar deste índice, (ao passo que numa abordagem por sub-matrizes se dar exatamente o oposto por exemplo).

Outro fato que pesa significativamente a favor da escolha do método de geração por colunas, é que de ambas as alternativas "convencionais" mediante o uso de vetores de trabalho expandidos, ser este o método com o menor *overhead*, (como se comentou na seção onde foi apresentado formalmente).

Assim, parece inegável que excetuando-se os casos Diagonais, que porventura possam existir no início do processo de eliminação, se tome uma alternativa de geração por colunas nas etapas iniciais do processo.

A partir de algum "ponto de corte", obviamente alguma



outra estratégia pode começar a ser explorada, e uma solução lógica, tendo-se em mente a maior eficiência computacional possível (a nível de tempos de execução), mantendo-se no entanto os "pés no chão" (com relação ao espaço disponível de armazenamento), seria considerar a aplicação da geração dos fatores complementares por linha, via o uso de "listas simbólicas de endereços", (até um limite aceitável para o tamanho da lista, em função da memória disponível), e apenas na eventualidade das limitações de memória terem sido alcançadas antes da conclusão de toda a etapa de fatoração, passar-se a complementar o restante das eliminações via a estratégia convencional por vetores de trabalho expandidos até se notar a presença de outra característica dominante (como o caso das sub-matrizes quase densas ou completamente densas), quando obviamente a solução mais apropriada certamente não é mais a utilização das técnicas convencionais por vetores de trabalho, mas sim por exemplo procedimentos simbólicos ao estilo *loop free* como os apresentados na seção III.1), ou estratégias como as apresentadas em [A8] por exemplo.

Percebe-se que com esta estratégia "modular" de ataque e sub-divisão da etapa de eliminação (como formalizada em [A8]), é uma solução sensata para o problema, se for considerado o objetivo básico a ser alcançado neste trabalho, ou seja o aumento da eficiência computacional da fase numérica, e que porventura poderá vir a ser usada repetidamente no futuro, por um número tal de vezes, que justifique o maior cuidado em se elaborar de forma mais otimizada todas etapas a serem seguidas durante esta fase.

O que fica faltando portanto, é se "conceituar" melhor a estratégia de solução proposta, e formaliza-la na forma de um procedimento, como veremos um pouco mais adiante.

O conceito novo a ser introduzido (e apresentado em [A8] na forma como veremos nesta seção, tendo sido porem proposto em níveis menos detalhados na literatura como em [D1]), é o de "janelas de processamento".

**Definição III.4(1)****Janela de processamento**

*Define-se por "janela de processamento" (para processos de eliminação Gaussiana esparsa), como sendo uma região delimitada por linhas e colunas contíguas (da matriz a ser fatorada) e que virão a ser tratados de forma unificada por uma mesma estratégia de implementação, diferenciada do restante das regiões (pertencentes as demais janelas de processamento), durante a fase de geração dos fatores associados à esta dada janela em particular.*

Esta definição é um pouco mais ampla do que a considerada em [A8], (onde as regiões consideradas eram apenas na forma de conjuntos contíguos de linhas, abrangendo-se neste caso toda a extensão de colunas das linhas de fatores  $U$  consideradas).

Naquela abordagem, apenas, os índices delimitadores das linhas inicial e final, mostravam-se necessários, para se caracterizar as regiões sendo tratadas de uma mesma forma.

Com a definição um pouco mais ampla apresentada acima, permite-se estender o conceito de janelas de processamento, de modo a englobar também casos particulares de destacada importância como os do tipo *supernodal* (na época ainda não cogitados pelo autor).

O que se precisa basicamente são das informações:

- linha inicial da matriz  $U$   
(a ser parcialmente gerada)
- linha final da matriz  $U$   
(a ser parcialmente gerada)
- coluna inicial em  $U^T$   
(associada as linhas a serem subtraídas de  $U$ )
- coluna final em  $U^T$   
(associada as linhas a serem subtraídas de  $U$ )

O que em princípio pode se mostrar "inquietante" é o fato de se utilizar uma delimitação para as colunas associadas em  $U^T$  (e não de  $U$ ).

Recordando-se que as colunas correspondentes à elementos não nulos numa dada linha de  $U^T$  determinam exatamente o conjunto de linhas anteriores a linha base em  $U$ , que efetivamente necessitarão ser subtraídas desta, percebe-se que com a delimitação de um intervalo de colunas em  $U^T$ , o processamento de uma dada "janela", poderá não ser "completo", no sentido de se efetuarem todas as subtrações de linhas anteriores necessárias para a geração completa de suas linhas base.

Dai ter se incluído o comentário relativo a geração apenas "parcial" das linhas de fatores  $U$ .

Como veremos um pouco mais adiante, esta metodologia não impõe restrições, no caso da geração dos fatores por linhas.

Numa geração de fatores por colunas, não se permite tal liberdade, pois o *loop* intermediário do procedimento II.4(2), só pode ser executado numa única ordem (crescente de índices).

A razão para tal pode parecer um tanto "obscura" para o leitor, de modo que se irá apenas comentá-la, no intuito de documentar o fato mencionado acima.

Na geração dos fatores por colunas, não se pode mudar a ordem com que se escolhe as linhas a serem subtraídas, porque simplesmente a porção sendo operada no vetor de contribuições  $W$ , vai sendo progressivamente transferida definitivamente para o armazenamento do fator de  $U$  correspondente. Deste modo, com uma subtração em uma ordem não crescente de índices de colunas, estaria se deixando para trás por exemplo, possíveis *fill-in's* que porventura teriam sido criados no processamento de uma linha não subtraída em virtude da alteração da ordem tomada para o *loop* intermediário.

De posse deste fato, e assumindo-se que na geração por linhas tais restrições não se encontram presentes (o que não será comprovado formalmente neste trabalho), pode-se começar a esboçar a forma que deverá ter um procedimento para uma eliminação seguindo uma metodologia "híbrida por janelas".

Para algumas das alternativas consideradas, apenas uma de ambas as informações relativas as colunas de  $U^T$  precisa ser efetivamente especificada, assumindo-se que a informação complementar seja dada pela coluna inicial ou final da região

considerada em  $U^T$ .

Estes são detalhes que podem ser usados para se minimizar o número de vetores e informações auxiliares, (mas que não serão considerados nesta abordagem), visando apenas facilitar a exposição e compreensão da técnica de eliminação por "janelas" proposta nesta seção.

De posse das informações relevantes enumeradas inicialmente, apresentamos o seguinte pseudo-código para o processamento "híbrido por janelas" como um todo:

**Procedimento III.4(1)      Eliminação Híbrida por "janelas"**

**para** *iwnd* de 1 até *nwnd* **faça**

*ibeg* ← BEGLIN(*iwnd*)

*iend* ← ENDLIN(*iwnd*)

*kbeg* ← BEGCOL(*iwnd*)

*kend* ← ENDCOL(*iwnd*)

**caso** WNDTYP(*iwnd*) **seja**

        1: **execute** DIAGONAL (*ibeg*, *iend*, *kbeg*, *kend*)

        2: **execute** WORKCOL (*ibeg*, *iend*, *kbeg*, *kend*)

        3: **execute** SIMBROW (*ibeg*, *iend*, *kbeg*, *kend*)

        4: **execute** WORKROW (*ibeg*, *iend*, *kbeg*, *kend*)

        5: **execute** SUPERNODAL (*ibeg*, *iend*, *kbeg*, *kend*)

        6: **execute** DENSE (*ibeg*, *iend*, *kbeg*, *kend*)

**fim caso** WNDTYP(*iwnd*)

**fim para** *iwnd*

No procedimento acima, assume-se como estrutura principal o vetor WNDTYP, que denota o tipo de janela de processamento a cada etapa do processo.

Os apontadores *ibeg*, *iend*, e *kbeg*, *kend* apontam em cada janela *iwnd* associada, para as posições inicial e final do grupo de linhas de  $U$  a serem processadas e parcialmente geradas, bem como as posições das colunas em  $U^T$  a serem consideradas como indicadoras do conjunto de linhas anteriores a serem subtraídas de cada uma das linhas base da janela sendo parcialmente gerada.

Os casos considerados foram particularmente os seguintes (correspondendo a ordem com que foram enumerados no

procedimento em questão): "completamente diagonal", convencional com geração por linhas, simbólico via listas de endereços com geração por linhas, convencional mediante vetores de trabalho e geração por linha, supernodal e completamente denso.

Percebe-se portanto, que com repetidas aplicações de alguma das formas de processamento consideradas, consegue-se levar adiante todo o processo de eliminação, com algumas porções podendo eventualmente ser geradas apenas parcialmente, numa primeira aplicação isolada de alguma das alternativas, e a seguir complementadas, em aplicações futuras de outras alternativas mais apropriadas.

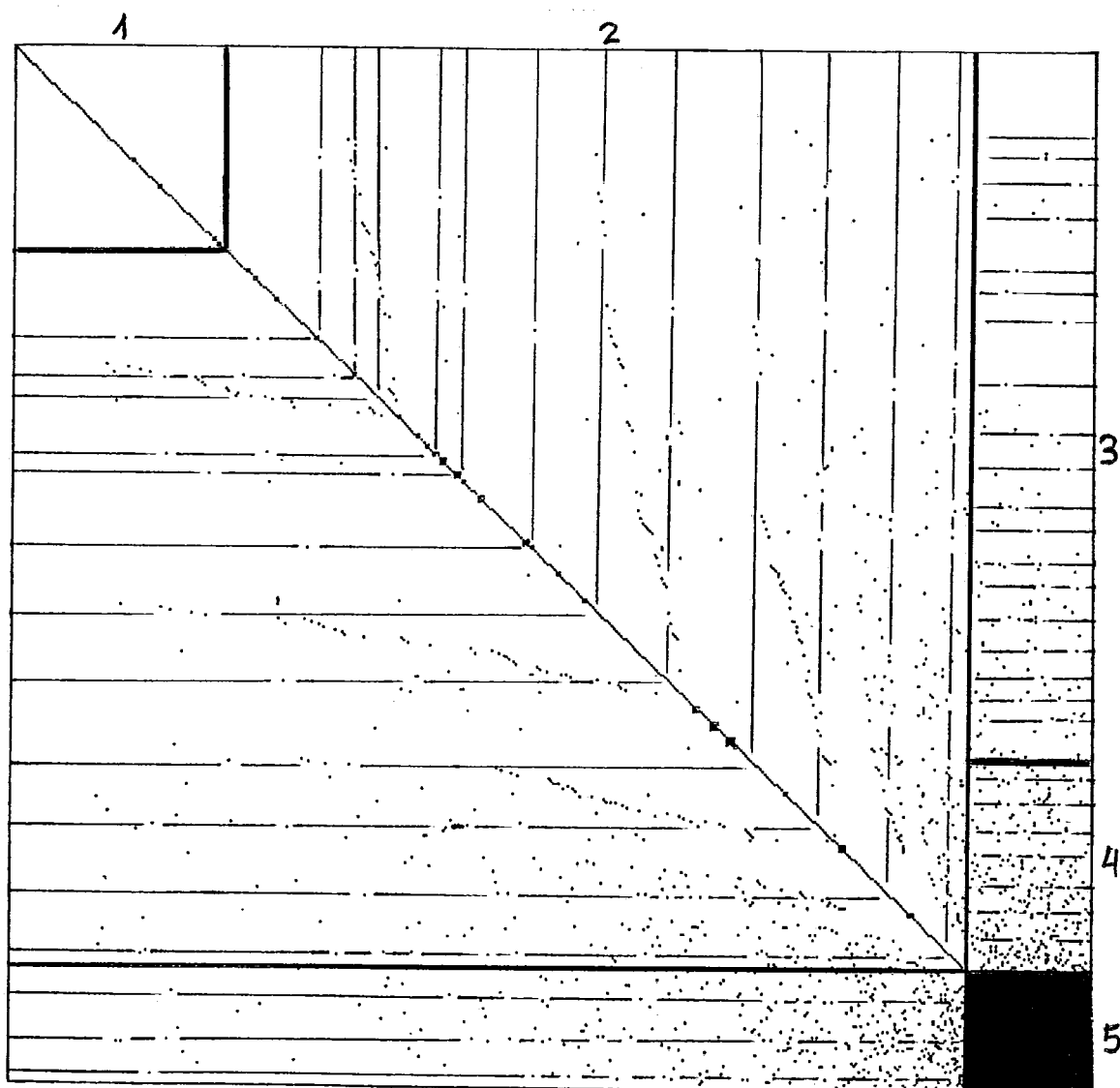
Como exemplo do que foi apresentado, consideremos as figuras fig\_III(3) e fig\_III(4), onde janelas voltadas para procedimentos convencionais com geração de fatores por colunas, simbólicos por listas de endereço (ou convencional) com geração por linhas, e completamente denso foram considerados para uma mesma matriz, (a título de exemplificação).

No primeiro caso, tratou-se inicialmente da porção completamente diagonal, e a seguir, adotou-se uma estratégia de geração por colunas até se chegar a linha correspondente à porção completamente densa, a partir da qual, uma geração por sub-matrizes na forma densa passou a ser empregada.

Como neste caso a geração de fatores por colunas inicialmente efetuada se deu apenas até a coluna correspondente ao início da porção densa, as porções restantes de cada uma destas linhas iniciais, necessitou ser gerada, e o procedimento adotado neste caso, foi a geração por linhas (inicialmente pela forma simbólica por listas de endereços, e a seguir pela forma convencional).

Percebe-se que no exemplo em particular, o número de "janelas" empregado foi de 5 alternativas distintas, e que assumindo-se que o índice do final da porção completamente diagonal seja dado por *idiag*, o do final do processamento simbólico por *isimb* e o do início da porção densa por *idens*, teremos as seguintes informações:

<i>iwnd</i>	<i>WNTYP</i>	<i>ibeg</i>	<i>iend</i>	<i>kbeg</i>	<i>kend</i>
1	1	1	<i>idiag</i>	1	<i>idiag</i>
2	2	<i>idiag+1</i>	<i>idens-1</i>	1	<i>idens-1</i>
3	3	1	<i>isimb</i>	1	<i>idens-1</i>
4	4	<i>isimb+1</i>	<i>idens-1</i>	1	<i>idens-1</i>
5	6	<i>idens</i>	<i>n</i>	<i>idens</i>	<i>n</i>

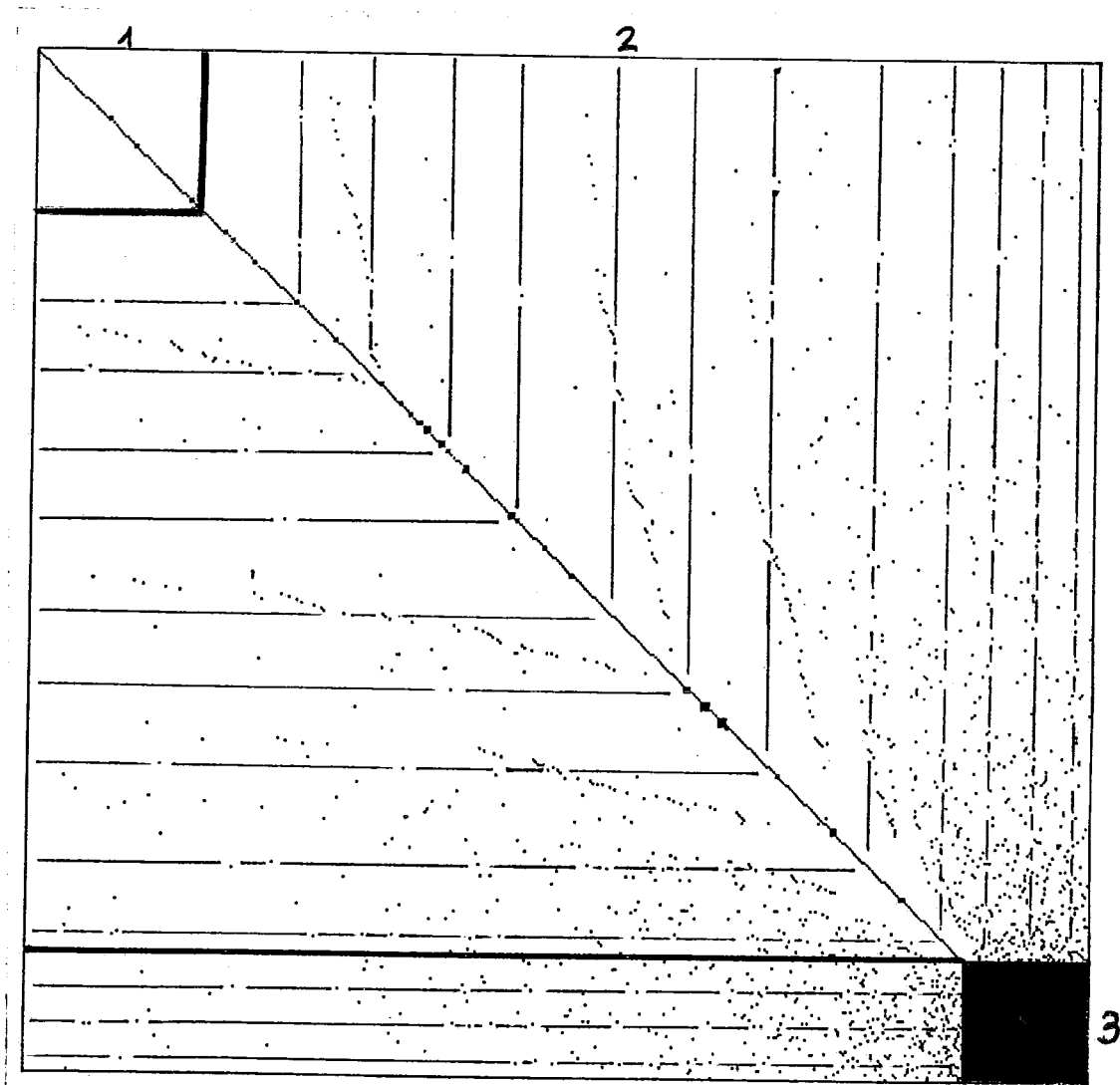


fig\_III(3) - Eliminação híbrida (1)

No segundo caso, apenas 3 etapas são utilizadas para se fatorar a mesma matriz do exemplo anterior.

Inicialmente considera-se a porção completamente

diagonal, passando-se a seguir a efetuar a geração por colunas, completando-se o processo com a eliminação da sub-matriz densa restante de forma completamente densa.



fig\_III(4) - Eliminação híbrida (2)

Segue-se a título de complementação, a seqüência de informações utilizadas neste caso, (assumindo-se a mesma notação para as porções indicadas no exemplo anterior).

iwnd	WNTYP	ibeg	iend	kbeg	kend
1	1	1	<i>idiag</i>	1	<i>idiag</i>
2	2	<i>idiag+1</i>	<i>n</i>	1	<i>idens-1</i>
3	6	<i>idens</i>	<i>n</i>	<i>idens</i>	<i>n</i>

O que fica faltando se apresentar mais formalmente, são "critérios de corte" (no estilo do que foi comentado na seção 1.2), de modo a se poder melhor ajustar o processamento, visando além de uma elevação na eficiência computacional a nível de tempos de execução, uma preservação sempre que possível do espaço total reservado para armazenamento (dentro de limites toleráveis).

Um dos critérios mais expressivos, como se pode intuitivamente notar, é considerar a geração por colunas até a coluna correspondente à da porção densa que porventura exista na matriz de fatores, e a partir daí continuar a geração por colunas "controladamente" até a última coluna, gerando-se neste caso porém, apenas os fatores acima da porção densa, deixando para uma abordagem completamente densa, a tarefa de completar o processamento da sub-matriz densa restante.

O segundo critério merecedor de destaque, é caso se opte por uma geração por colunas apenas até a coluna anterior a da porção densa, e a partir deste ponto considerar a geração por linhas para se complementar as porções restantes acima da porção densa, neste caso, tomar como critério de corte para a escolha de qual das estratégias de geração por linhas considerar, a quantidade total de memória disponível, adotando-se uma geração simbólica por listas de endereços até certo ponto, e a partir do ponto em que o tamanho total da lista se tornar inaceitável, complementar-se o processo, com uma geração convencional mediante vetores de trabalho, para as linhas restantes, deixando apenas a sub-porção densa que porventura exista, para ser abordada de forma completamente densa.

O que o leitor neste ponto, com o espírito já bem mais aguçado por tantos meandros e detalhes da geração esparsa de fatores, poderia se questionar, é porque se considerar a geração por linhas afinal, visto que uma estratégia como a adotada no segundo exemplo, com a geração por colunas (que notadamente tem desempenho melhor que uma geração convencional por linhas mediante vetores de trabalho), adotando-se esta estratégia em toda a porção excluindo-se a sub-matriz densa, e lançando-se mão exclusivamente de uma estratégia densa para esta porção final; não poderia ser



sempre a adotada em todos os casos, dispensando-se a sub-divisão em demais janelas baseadas na geração por linhas por exemplo, e cujo desempenho final a princípio não parece apontar para uma redução significativa de *overheads* em comparação com a geração por colunas, que só não se mostra eficiente nos casos *supernodais* e *densos* de um modo geral.

A resposta a esta indagação só poderá ser dada no próximo capítulo, pois o novo e "último" método a ser apresentado, baseia-se exclusivamente na geração por linhas, e definitivamente é mais compensador do que uma estratégia de geração por colunas (pelo menos em algumas classes de arquiteturas escalares convencionais, como as do tipo CISC por exemplo).

O novo método forçosamente se baseia numa geração por linhas, pelo simples fato já comentado anteriormente nesta seção, de que a geração por colunas ser bem mais "rígida" do que a geração por linhas, por simplesmente não permitir flexibilidade alguma no tocante a uma possível alteração na ordem com que as linhas contribuintes venham a ser efetivamente subtraídas de cada linha base corrente.

● *É justamente esta flexibilidade de reordenamento "dinâmico" da ordem das contribuições, a característica básica a ser explorada pela nova metodologia proposta.*

## Capítulo IV

## NOVA ABORDAGEM PROPOSTA

Neste capítulo apresenta-se uma nova metodologia de eliminação, com geração dos fatores  $U$  por linhas, e que reúne algumas das características mais desejáveis para um código de fatoração esparsa: ser eficiente em termos de menores *overheads* em tempo de execução, e ao mesmo tempo "tratável" do ponto de vista do espaço final de armazenamento e código utilizado.

Na primeira seção, apresenta-se a motivação computacional básica para o método, e uma primeira abordagem para o problema.

Fica claro a partir deste ponto, que "algo mais" ainda pode ser feito. Porém para se alcançar este novo patamar, torna-se necessário a utilização de um importante conceito em esparsidade, até então não explorado neste trabalho.

Em vista disso, na seção subsequente, introduz-se e formaliza-se com um pouco mais de detalhe algumas noções e propriedades da "árvore de eliminação".

De posse deste novo ferramental, viabiliza-se a construção de uma "variante" ainda mais bem sucedida, do método original apresentado na primeira seção.

Uma formulação completa do método, englobando ambas as alternativas consideradas no presente estudo, é apresentada finalmente, concluindo este capítulo em questão.

O novo método a ser apresentado, pode ser encarado como uma generalização das alternativas propostas na seção III.1, adequando-as para o caso esparsa geral, ou como uma extensão da metodologia de Gustavson [G14], mantendo-se no entanto o tamanho de código "constante" para qualquer estrutura de matrizes esparsas a serem fatoradas.

O método pode ser visto também como um aprimoramento das alternativas baseadas na utilização de listas simbólicas de endereços (apresentadas na seção III.2), em que a informação básica armazenada é constituída na verdade de "códigos", capazes de caracterizar completamente as operações de combinação linear entre múltiplos vetores, ao nível "elementar" num estilo similar ao de Gustavson, e cujo

processo de "interpretação" pode ser implementado de forma eficiente em praticamente todas as linguagens de programação de alto nível, comercialmente disponíveis.

Por se basear na especificação ao nível "elementar" das operações a serem efetuadas durante a eliminação (da mesma forma como as estratégias baseadas na utilização de listas simbólicas de endereços), poderia se esperar um crescimento "explosivo" da lista de códigos utilizada para a caracterização de todo o processo.

Embora em um "pior caso", tal comportamento seja teóricamente possível, na prática, o crescimento se dá numa taxa bem mais "controlada", mantendo-se em níveis aceitáveis o espaço total de armazenamento, que é na maioria das vezes, proporcional apenas ao número de fatores não nulos presentes na matriz resultante após o processo de eliminação, (em oposição a um volume proporcional ao número de operações de ponto flutuante efetuadas em todo o processo, como no caso das alternativas da seção III.2).

Finalmente, pelo fato de uma das alternativas do método proposto, viabilizar uma "compactação" eficiente das informações necessárias para a caracterização de múltiplas operações elementares de combinação linear de vetores; características da forma *supernodal* (como as apresentadas na seção III.3) podem ser eficientemente exploradas pelo método, sem perda alguma de sua generalidade para o caso "realmente" esparsos (como definido no final da seção III.3).

É justamente esta flexibilidade de permitir uma "compactação" de informações, mesmo em face a presença de características "realmente" esparsas, ou seja, com um padrão de acesso "aparentemente" não determinístico aos fatores de cada linha base sendo gerada, é que confere ao método um caráter de certa forma especial, merecendo portanto uma atenção mais cuidadosa em abordagens futuras, especialmente na fase de reordenamento visando a redução de *fill-in's*, e na aplicação do método à novas arquiteturas do tipo paralelo.

#### IV.1 Motivação computacional e princípios básicos

A motivação para o novo método proposto, partiu originalmente de questões de natureza puramente "semânticas", a nível de linguagens de programação, com um questionamento lançado pelo pesquisador Hermínio J.C.Pinto ao final do período em que o presente autor encontrava-se como bolsista do Centro de Pesquisas em Engenharia Elétrica (CEPEL).

O questionamento feito na época foi com relação a se haveria alguma "vantagem" em se programar rotinas de fatoração esparsa na linguagem C por exemplo, em comparação com implementações equivalentes numa linguagem como FORTRAN.

Na época, a resposta dada pelo presente autor, foi de que "aparentemente não", ou seja, muito pouco se poderia esperar de ganhos reais em implementações em C, visto que os compiladores FORTRAN otimizantes disponíveis no mercado, conseguem na grande maioria das vezes, produzir códigos de máquina muito próximos ou tão otimizados quanto os obtidos com os melhores compiladores C disponíveis.

A pergunta feita ao final de 1990, permaneceu desde então, pois em todas alternativas consideradas pelo autor até o início de 1993, nenhuma das rotinas codificadas, com exceção das baseadas na utilização de "listas simbólicas" de endereços, possuía alguma característica que pudesse ser mais bem explorada em uma particular linguagem como no caso de C, (e que por se tratar de uma das linguagens de alto nível, que mais se aproxima de uma "linguagem de máquina", se poderia esperar um melhor aproveitamento de pequenas nuances de implementação, impossíveis de serem tratadas em outras linguagens).

O único problema com a abordagem por "listas simbólicas de endereços" é que elas podem se tornar demasiadamente extensas, mesmo com a utilização de recursos como o "truncamento" ao se chegar a porção "quase densa", ou a compactação mediante a exploração de características *supernodais*.

Esta deficiência intrínseca, somada ao questionamento original a respeito de alguma forma de codificação em C que pudesse ser mais bem implementada e explorada do que em FORTRAN, deu portanto origem ao novo método, e que responde

finalmente a questão levantada.

Em pelo menos uma forma de abordagem (como a que será apresentada no presente capítulo), uma implementação em C mostra-se mais eficiente do que em FORTRAN, pelo fato de se basear em acessos "diretos" a posições de memória via o uso de apontadores (*pointers*), e que em FORTRAN necessitam ser "simulados" mediante o uso de "vetores de índices".

A motivação computacional básica portanto, foi encontrar alguma forma de codificação, que permitisse uma melhor implementação, baseada no uso de acessos diretos à posições de memória via o uso de *apontadores*, e que de algum modo, não "herdasse" os inconvenientes das abordagens por "listas simbólicas de endereços".

O primeiro passo neste sentido, foi notar que uma outra "variante" do método de eliminação esparsa, conhecido na literatura como método de Crout (para o caso assimétrico em geral) [D50], [G39], [F1], e até então não explorada ou considerada em nenhuma das abordagens implementadas pelo presente autor, poderia ser aplicada e com sucesso, numa abordagem esparsa baseada no uso de apontadores para posições de memória (correspondentes a cada um dos elementos da matriz de fatores sendo acessados ou gerados).

Segue-se portanto, uma apresentação desta variante para o caso simétrico, e que passaremos a olhar um pouco mais atentamente.

O método de Crout, nada mais é do que uma outra forma de implementação do método tradicional de geração dos fatores por linhas, como a alternativa L1(B), simplesmente revertendo-se a ordem dos 2 *loops* mais internos.

Ou seja, ao invés de processar a subtração de todos os elementos de cada uma das linhas contribuintes na linha base corrente  $i$  "linha à linha", o que se faz é acumular-se as contribuições de todos os elementos de uma mesma coluna de uma só vez, subtraindo-se tal contribuição do elemento base (da coluna correspondente) na linha  $i$ , (e que a partir deste ponto passa a conter o valor definitivo do fator gerado), passando-se ao processamento das acumulações de elementos da coluna seguinte, até terem sido gerados todos os fatores da linha base.

O método de Crout é baseado portanto em uma geração dos

fatores por linhas ao nível de cada etapa básica  $i$ , porem gerando-se tais fatores "coluna a coluna" durante o processamento da eliminação relativa a cada etapa básica do processo como um todo.

Segue-se portanto, um pseudo-código ao estilo das alternativas apresentadas na seção I.1, assumindo-se inicialmente um processamento na forma completamente *densa*.

**Procedimento IV.1(1)      Metodo de Crout "simetrico" (denso)**

- A cada etapa  $i$  do processo, anular os elementos à esquerda da diagonal, na  $i$ 'ezima linha mediante a subtração de múltiplos escalares de elementos em uma mesma coluna das linhas anteriores, atualizando-se desta forma a porção à direita da diagonal na  $i$ 'ezima linha (coluna a coluna), por operações da forma:

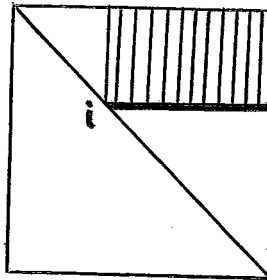
para  $i$  de 2 até  $n$

para  $k$  de 1 até  $i-1$

$$u_{k,i} \leftarrow a_{k,i} / a_{k,k}$$

para  $j$  de  $i$  até  $n$

$$a_{i,j} \leftarrow a_{i,j} - \sum_{k=1}^{i-1} u_{k,i} * a_{k,j}$$



fig\_IV(1) - Atualização simetrica por Crout

Gráficamente, tal processo é ilustrado na fig\_IV(1) acima, onde se percebe que a única diferença com relação a alternativa I.1(B) é a ordem com que se processam todas as subtrações em uma dada etapa básica do processo, não se alterando porém a região de fatores anteriores acessada, nem a porção gerada ao final de cada etapa (correspondendo a linha base  $i$ ), o que nos mostra que o método de Crout, continua a ser uma metodologia de geração dos fatores "por

linhas", (independentemente da ordem como tais fatores venham a ser gerados em cada etapa básica).

Uma vez compreendida a "mecânica" do método de Crout para o caso *denso*, o que seria de se perguntar é porque então este método quase nunca foi empregado para o caso *esparso* mais geral.

A resposta é que para se implementar o método de Crout no caso *esparso*, apenas as estruturas de dados elementares disponíveis em função do armazenamento seqüencial por linhas, não são suficientes, pois como se pode recordar dos comentários feitos ao final da seção III.3, o processamento "realmente" *esparso* (como foi assim definido), implica em acessos à posições de "difícil previsibilidade" na linha base sendo gerada, em função da natureza originalmente aleatória, da estrutura da matriz A do problema original.

Ou seja torna-se necessário, lançar mão de novas estruturas "dinâmicas" ou "simbólicas", que possibilitem a implementação da etapa correspondente ao somatório no *loop* mais interno do procedimento IV.1(1) apresentado.

A aparente dificuldade de implementação do somatório em questão, se dá pelo fato da estrutura básica de armazenamento dos elementos de cada linha de U, permanecer a mesma adota ao longo de todo este trabalho, (ou seja na forma seqüencial por linhas), ao passo que o somatório em questão, deve ser efetuado "coluna a coluna", o que se mostra inteiramente trivial de se implementar numa abordagem *densa*, mas que requer um pouco mais de atenção para o caso *esparso*, o que passaremos a considerar a seguir.

Vale ressaltar, que foi exatamente deste ponto, visando a contornar as dificuldades intrínsecas da abordagem via Crout para o caso *esparso*, e ainda tendo em mente o objetivo de por algum modo se obter uma nova metodologia baseada em acessos diretos a memória (via o uso de apontadores), que se divisou uma solução de fácil implementação, atendendo à ambos os requisitos.

A idéia consiste em se dispor de um "vetor" de apontadores  $p_k$ , cada um para uma dada posição de memória correspondente a representação de cada uma das linhas  $k$  a

serem consideradas na etapa de acumulação do somatório.

Inicialmente, tais apontadores  $p_k$  devem apontar para a posição correspondente a coluna  $i$  em cada uma das linhas  $k$  associadas.

Esta informação, recordando-se o que foi apresentado na seção II.4, é trivialmente obtida a cada etapa básica  $i$  do processo, mediante o uso do vetor de apontadores IUP, (utilizado em muitas das abordagens apresentadas).

Uma vez inicializados os apontadores  $p_k$ , procede-se a uma "normalização" dos elementos na coluna  $i$ , de cada uma das linhas  $k$  consideradas, (correspondendo ao segundo *loop* no procedimento IV.1(1) apresentado nesta seção).

A fase seguinte é a que requer maior atenção, pois é onde irá se processar o acúmulo de cada uma das contribuições "coluna à coluna" para cada um dos índices  $j$ , (correspondendo apenas a operações sobre os elementos não nulos em cada uma das linhas  $k$  envolvidas), e que posteriormente serão subtraídas do elemento  $a_{ij}$ , atualizando-se desta forma definitivamente o seu valor numérico (permitindo que a partir deste ponto, o novo fator assim gerado, venha a ser utilizado em etapas básicas  $i$  mais adiante do processo, durante o acúmulo de contribuições para a geração de novos fatores).

A grande dificuldade de se implementar estas operações de forma eficiente, é porque a princípio a forma "dinâmica" apresentada no procedimento IV.1(2) a seguir, parece ser a única viável para se atacar o problema.

As ineficiências inerentes a este procedimento em particular, descartariam a sua utilização de modo a competir com as demais estratégias de eliminação consideradas ao longo deste trabalho.

O principal problema é simplesmente o *loop* mais interno (na variável  $k$ ), cuja varredura "controlada" para cada coluna "c" sendo acumulada acaba vindo a recair numa forma similar à alternativa II.4(A), e que se baseia em testes condicionais para se determinar uma coincidência no índice de colunas, de modo a se efetuar a simples operação de acumulação associada.



## Procedimento IV.1(2)

Acumulação "dinâmica"

```

para i de 1 até n faça
  IUP(i) ← IUC(i)
  piv ← DIC(i)
  para k de IUT(i) até IUTF(i) faça
    l ← JUT(k)
    pl ← IUP(l)
    pfl ← IUF(l)
    IUP(l) ← IUP(l) + 1
    uml ← UN(pl) * DIC(l)
    piv ← piv - UN(pl) * uml
    UN(pl) ← uml
    pl ← (pl + 1)
  fim para k
  para j de IUC(i) até IUF(i) faça
    c ← JUC(j)
    s ← 0
    para k de IUT(i) até IUTF(i) faça
      l ← JUT(k)
      se (JU(pl) = c) e (pl ≤ pfl) então
        s ← s + uml * UN(pl)
        pl ← (pl + 1)
      fim se
    fim para k
    UN(j) ← UN(j) - s
  fim para j
  DIC(i) ← 1 / piv
fim para i

```

Algo portanto precisaria ser incorporado ao procedimento apresentado acima, de modo a se eliminar as ineficiências básicas das operações de varredura e acumulação efetuadas.

É neste ponto que as estratégias apresentadas na seção III.1, voltam a se mostrar atrativas, como veremos a seguir.

A idéia chave para uma implementação eficiente, é de algum modo trocar a varredura "dinâmica", por um processo

baseado em informações "estáticas", (por que não dizer "simbólicas"), cuja decodificação possa ser feita de forma mais eficiente do que a estratégia apresentada até então.

Percebe-se pois que mediante uma fase auxiliar executada uma única vez antes da fase numérica, cujo o objetivo seja o de coletar e produzir informações ou códigos adicionais voltados ao processamento das acumulações efetuadas na etapa mais interna do processo de eliminação, estará se caminhando na direção correta, no sentido de se viabilizar uma implementação "dedicada" da fase numérica, mais eficiente do que o procedimento IV.1(2).

A idéia principal a ser observada e explorada, é que a acumulação na forma considerada, mostra-se de "difícil implementação", simplesmente porque a estrutura de elementos não nulos em cada coluna sendo acumulada, aparentemente não é "previsível", pois nem todas as linhas envolvidas na eliminação da linha base  $i$  do processo, contém elementos não nulos em todas as colunas "c" correspondentes aos fatores não nulos da linha base (e que acabarão definitivamente gerados ao final da etapa básica associada).

Ou seja, onde alguma informação de natureza "simbólica" pode vir a ser útil, numa implementação de uma estratégia do tipo Crout para o caso *esparso*, não é na determinação das posições de memória a serem acessadas na linha base, como nas demais estratégias simbólicas apresentadas até então, mas sim em alguma "caracterização" da estrutura de elementos não nulos presentes em cada coluna a ser acumulada, durante a fase de geração dos elementos da linha básica corrente.

A informação simbólica mais simples que se pode obter, caracterizando completamente o fato acima, é lançar mão de uma simples representação binária do padrão de elementos não nulos presentes em cada uma das colunas "c" a serem consideradas durante a fase de geração da linha básica.

Detalhando-se um pouco mais o processo, tomemos como exemplo o seguinte padrão estrutural de linhas a serem subtraídas da linha corrente  $i$ , na sexta etapa do processo de eliminação, apresentado na figura a seguir.

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
(1)	*					*	●			●	
(2)		*									*
(3)			*			*				●	●
(4)				*			*				*
(5)					*	*				●	
(6)	○		○		○	*	*		*	*	*

$a_{6,6}$

fig\_IV(2) - Padrão estrutural de linhas contribuintes

No exemplo acima, apenas as linhas 1, 3 e 5 necessitam ser subtraídas da linha 6, e os únicos elementos não nulos que efetivamente contribuem sobre a linha 6 são os representados por "●" e "\*" visto que os elementos representados por "\*" pertencem a linhas que não serão subtraídas da linha base (pois nas posições à esquerda da diagonal da linha base, estes elementos já serem nulos pela própria natureza estrutural da matriz considerada no exemplo em particular).

Assim convém lançar mão de uma representação compacta da disposição estrutural apenas das linhas e colunas contendo elementos não nulos, realmente envolvidas no processo, e apresentadas na figura a seguir.

	(6)	(7)	(9)	(10)	(11)
(1)	*	●		●	
(3)	*			●	●
(5)	*			●	
(6)	*	*	*	*	*

$a_{6,6}$

fig\_IV(3) - Estrutura compacta das contribuições

Por esta representação, percebe-se claramente que o que fica faltando para se especificar de forma eficiente uma informação que caracterize as operações de acumulação em cada coluna envolvida, é tomar-se como código associado a cada

coluna "c" da estrutura de fatores da linha base, uma representação binária do padrão de elementos não nulos em cada uma das colunas desta representação compacta.

Ou seja, para o exemplo em questão, e assumindo-se que a codificação binária ocorra no sentido da linha de menor índice à contribuir sobre a linha base (no caso a de #1 em particular) em direção à última linha contribuinte (no caso a de #5), teríamos a seguinte representação binária, excluindo-se as contribuições na coluna do elemento diagonal (a serem tratadas de forma mais eficiente em separado):

	(7)	(9)	(10)	(11)
(1)	1	0	1	0
(3)	0	0	1	1
(5)	0	0	1	0
	001	000	111	010
LSTCOD:	1	0	7	2

fig\_IV(4) - Codificação binária

O que fica faltando se especificar é como se aproveitar esta codificação binária associada a cada coluna, de modo a se tornar o procedimento de acumulação das contribuições em cada coluna mais eficiente do que no procedimento IV.1(2).

A forma a ser considerada, se aproxima das técnicas simbólicas ao estilo de Gustavson (apresentadas em III.1), uma vez que a operação básica a ser efetuada de posse de uma codificação binária associada, é na verdade um produto escalar de vetores, onde cada um dos termos a serem adicionados (correspondendo ao produto das componentes destes vetores) são da forma  $um_l * UN(p_l)$ .

Desta forma codificando-se todas as possíveis combinações de operações de acumulação da forma descrita, para um exemplo particular de vetores com até 3 componentes, ficaríamos com trechos de código como os apresentados a seguir, (assumindo-se que  $s$  esteja originalmente zerado).

**Codificação IV.1(a)      Produto Escalar via Codificação binária**

0:            **retorne ao ponto de chamada**  
1:             $s \leftarrow s + um_1 * UN(p_1)$   
              **retorne ao ponto de chamada**  
2:             $s \leftarrow s + um_2 * UN(p_2)$   
              **retorne ao ponto de chamada**  
3:             $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2)$   
              **retorne ao ponto de chamada**  
4:             $s \leftarrow s + um_3 * UN(p_3)$   
              **retorne ao ponto de chamada**  
5:             $s \leftarrow s + um_1 * UN(p_1) + um_3 * UN(p_3)$   
              **retorne ao ponto de chamada**  
6:             $s \leftarrow s + um_2 * UN(p_2) + um_3 * UN(p_3)$   
              **retorne ao ponto de chamada**  
7:             $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
              **retorne ao ponto de chamada**

Percebe-se que com a introdução dos trechos de código acima, o processo de "decodificação" da informação binária relativa as linhas a serem efetivamente utilizadas no acúmulo do somatório, passa a ser efetuado de uma forma bem mais eficiente do que no procedimento IV.1(2) onde a acumulação era efetuada baseando-se apenas nas informações dinâmicas determinadas em tempo de execução.

Uma vez que a codificação dos produtos escalares correspondendo ao acúmulo das contribuições em cada coluna é certamente a etapa básica de uma metodologia ao estilo Crout, percebe-se que de posse do esquema proposto em IV.1(a), pode-se passar a uma nova especificação (completa) do método de eliminação, e que será apresentada a seguir.

Neste novo procedimento, lança-se mão de um artifício com relação aos índices das linhas contribuintes a participarem do processo de acumulação.

Ao invés de se tomar os valores absolutos de  $l$ , toma-se na verdade um índice "reduzido"  $r$ , correspondendo a posição de cada uma das linhas  $l$  envolvidas, porém utilizando-se a representação "compacta" apresentada na fig\_IV(3).

Assim, no exemplo em questão, teríamos os índices  $r$

assumindo os valores 1, 2 e 3, correspondendo as linhas 1, 3 e 5 respectivamente.

Outro ponto a se notar, é que os apontadores  $p_r$  são automaticamente incrementados após cada operação de acúmulo na forma de produto escalar, o que não havia sido explicitado na codificação IV.1(a) porque nesta alternativa em questão, ainda não se havia considerado a dinâmica do processo de eliminação via Crout, em que uma vez contribuindo no acúmulo de uma determinada coluna, a próxima contribuição só voltará a ocorrer com o próximo elemento não nulo da mesma linha, razão pela qual os apontadores para a posição corrente em cada uma das linhas, serem deslocados para a posição subsequente após o acúmulo em uma dada coluna.

No procedimento em questão, lança-se mão de uma "lista de códigos" armazenados no vetor LSTCOD, e que basicamente contém a codificação binária associada a cada coluna a ser acumulada durante o processo de geração da  $i$ 'ésima linha da matriz de fatores.

O problema mais importante a ser considerado com um pouco mais de detalhe portanto, é o caso em que o número de linhas a contribuírem numa dada etapa do processo, superar o número máximo de linhas previsto para a representação binária.

Nestes casos, novas "varreduras" de todas as posições da linha base corrente, mostram-se necessárias, processando-se o acúmulo das novas contribuições, até que tenham sido consideradas todas as linhas contribuintes.

A solução mais prática para se contornar esta questão, é lançar mão de códigos auxiliares indicando que o processamento das acumulações em cada coluna ainda não foi concluído. No procedimento em questão, utilizou-se uma lista de códigos auxiliar LSTKEND indicando os valores finais de  $k$  necessários para se levar avante o processo de acumulação em etapas voltadas ao tratamento de no máximo 3 linhas simultaneamente (no caso particular da presente codificação considerada e apresentada).

**Procedimento IV.1(3)      Acumulação via codificação binária**

```

ncod ← 0
nk ← 0
para i de 1 até n faça
    IUP(i) ← IUC(i)
    piv ← DI(i)
    kbeg ← IUT(i)
    kf ← IUTF(i)
α:    nk ← nk + 1
    kend ← LSTKEND(nk)
    r ← 0
    para k de kbeg até kend faça
        r ← r + 1
        l ← JUT(k)
        pr ← IUP(l)
        IUP(l) ← IUP(l) + 1
        umr ← UN(pr) * DI(l)
        piv ← piv - UN(pr) * umr
        UN(pr) ← umr
        pr ← (pr + 1)
    fim para k
    para j de IUC(i) até IUF(i) faça
        s ← 0
        ncod ← ncod + 1
        vá para (0, 1, 2, 3, 4, 5, 6, 7)
            em função de LSTCOD(ncod)
β:    UN(j) ← UN(j) - s
    fim para j
    kbeg ← kend + 1
    se kend ≠ kf vá para α
    DI(i) ← 1 / piv
fim para i
termine procedimento
0:    vá para β
1:    s ← s + um1 * UN(p1)
    p1 ← (p1 + 1)
    vá para β

```

(continua na próxima página)

(continuação da página anterior)

- 2:  $s \leftarrow s + um_2 * UN(p_2)$   
 $p_2 \leftarrow (p_2 + 1)$   
 vá para  $\beta$
- 3:  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 vá para  $\beta$
- 4:  $s \leftarrow s + um_3 * UN(p_3)$   
 $p_3 \leftarrow (p_3 + 1)$   
 vá para  $\beta$
- 5:  $s \leftarrow s + um_1 * UN(p_1) + um_3 * UN(p_3)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 vá para  $\beta$
- 6:  $s \leftarrow s + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 vá para  $\beta$
- 7:  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 vá para  $\beta$

Nos casos em que apenas uma única "varredura" é necessária para se gerar os elementos da linha base, o valor armazenado na lista de códigos LSTKEND é na verdade idêntico a IUTF(i).

Nestes casos, pode-se notar também, que o tamanho da lista LSTCOD é na verdade idêntico ao número de fatores não nulos da matriz obtida ao final da eliminação.

Ou seja, assumindo-se a hipótese de cada linha na representação da transposta de  $U$  possuir até 3 elementos não nulos (no caso da codificação exemplificada em IV.1(3) em particular), percebe-se que o total de informações simbólicas adicionais necessárias para se caracterizar completamente o processo, é dado por  $n + Nonz(U)$ , o que sem dúvida mostra-se dentro de um limite tolerável em termos de espaço adicional de armazenamento.



Nos casos em que mais do que o número máximo de componentes consideradas na codificação dos produtos escalares, necessitarem ser subtraídas em cada coluna, o espaço total de armazenamento passa a ter um comportamento crescente da ordem de  $\#varreduras * (n + Nonz(U))$ , onde  $\#varreduras$  indica o número máximo de varreduras necessárias para se completar cada etapa básica  $i$  de geração dos fatores em cada uma das linhas base.

Naturalmente esta é uma estimativa máxima, do pior caso, pois apenas na geração de fatores de linhas base que necessitem de mais do que uma varredura, precisará se lançar mão deste recurso inevitável na abordagem considerada até então.

Percebe-se pois, que em alguns casos como na presença predominante de estruturas do tipo *supernodal* por exemplo, o número de varreduras acabaria se tornando impraticável.

Algo mais precisa ser acrescentado à abordagem apresentada até então, para torna-la de fato competitiva com as demais alternativas apresentadas neste trabalho.

A extensão do método se dará ao longo das próximas seções deste capítulo, e nos concentraremos no final desta seção, em complementar a exposição da abordagem considerada, apresentando-se o processo de geração da lista de "codificações" binárias, o que não havia sido considerado até o presente momento.

Se olharmos mais detalhadamente o procedimento IV.1(2) baseado na atualização "dinâmica" das contribuições em cada coluna, perceberemos que o mesmo contém a base para uma geração das codificações binárias necessárias para a caracterização das operações a serem efetuadas, visto que durante o processo de varredura "dinâmica" dos elementos em cada coluna, pode-se ir gradualmente construindo os índices binários associados, como no procedimento apresentado a seguir, e onde  $n_{max}$  denota o número máximo de componentes a serem consideradas de cada vez, nas operações de acumulação (o que corresponderia a 3 no exemplo adotado nesta seção).

**Procedimento IV.1(4)      Geração da codificação binária**

```

ncod ← 0
nk ← 0
para i de 1 até n faça
    IUP(i) ← IU(i)
    kbeg ← IUT(i)
    kf ← IUTF(i)
α:    kend ← min (kbeg + (nmax - 1), kf)
    nk ← nk + 1
    LSTKEND(nk) ← kend
    para k de kbeg até kend faça
        l ← JUT(k)
        pl ← IUP(l)
        pfl ← IUF(l)
        IUP(l) ← IUP(l) + 1
        pl ← (pl + 1)
    fim para k
    para j de IU(i) até IUF(i) faça
        c ← JU(j)
        icod ← 0
        b ← 1
        para k de IUT(i) até IUTF(i) faça
            l ← JUT(k)
            se (JU(pl) = c) e (pl ≤ pfl) então
                icod ← icod + b
                b ← b * 2
                pl ← (pl + 1)
            fim se
        fim para k
        ncod ← ncod + 1
        LSTCOD(ncod) ← icod
    fim para j
    kbeg ← kend + 1
    se kend ≠ kf vá para α
fim para i

```

No procedimento em questão, adotou-se a representação original dos índices  $l$  associados a cada uma das linhas envolvidas, visto não ser necessário utilizar-se uma representação compacta baseada nos índices  $r$  como a adotada no procedimento IV.1(3).

Os mesmos comentários observados com relação ao espaço de armazenamento da lista de códigos e de posições finais do índice  $k$ , em função do número de varreduras necessárias também é aplicável ao procedimento de geração da codificação apresentado, porem o fato mais importante a se observar, é na verdade o comportamento em termos do tempo de execução deste algoritmo, o que merece um pouco de atenção, por não se mostrar tão eficiente quanto o processo de decodificação das informações.

Pelo fato de se efetuar uma varredura "dinâmica" em cada uma das colunas da linha de fatores sendo gerada, mesmo supondo-se o caso ideal onde apenas uma varredura mostra-se necessária por etapa, percebe-se que o volume de operações é ditado por  $Nonz(U_{i*}^T) * Nonz(U_{i*})$  o que sem dúvida é maior que o número de operações de ponto flutuante que seriam necessárias para se implementar a mesma etapa.

Tal fato ocorre, porque muitas das comparações efetuadas durante o processo de varredura, resultam em operações que na verdade não seriam efetuadas durante a eliminação numérica, por não haver a coincidência do índice da coluna associada na presente linha anterior sendo considerada durante a varredura.

Percebe-se portanto que algo precisa ser feito no sentido de se contornar tais *overheads* indesejáveis.

A solução para este problema, vem de uma estratégia que poderia passar de início "desapercebida", ou de todo não cogitável a princípio, qual seja: recorrer aos métodos de eliminação por linha "tradicionais" como os apresentados na seção II.4 por exemplo.

Ou seja, a fase de geração da codificação simbólica para um método do tipo Crout simétrico, pode ser mais bem implementada, se for baseada na estratégia de sua variante "rival" em termos de eficiência na fase numérica, e considerada em muitas das abordagens neste trabalho.

Portanto, deixando de lado questões de outra natureza, passaremos a considerar uma nova implementação da fase de geração dos códigos binários, tendo por base o procedimento II.4(1).

**Procedimento IV.1(5)      Codificação binária eficiente**

```

ncod ← 0
nk ← 0
para i de 1 até n faça
    IUP(i) ← IUC(i)
    para j de IUC(i) até IUF(i) faça
        IWCOD(JUC(j)) ← 0
    fim para j
    kbeg ← IUT(i)
    kf ← IUTF(i)
α:   kend ← min (kbeg + (nmax - 1), kf)
    nk ← nk + 1
    LSTKEND(nk) ← kend
    b ← 1
    para k de kbeg até kend faça
        l ← JUT(k)
        IUP(l) ← IUP(l) + 1
        para j de IUP(l) até IUF(l) faça
            c ← JUC(j)
            IWCOD(c) ← IWCOD(c) + b
        fim para j
        b ← b * 2
    fim para k
    para j de IUC(i) até IUF(i) faça
        ncod ← ncod + 1
        LSTCOD(ncod) ← IWCOD(JUC(j))
    fim para j
    kbeg ← kend + 1
    se kend ≠ kf vá para α
fim para i

```

Neste procedimento, no lugar do vetor de trabalho expandido  $W$  presente na abordagem de fatoração numérica, lança-se mão de um vetor de trabalho inteiro expandido IWCOD, que irá acumular cada uma das codificações binárias associadas a todos os fatores não nulos da linha base a ser gerada.

A diferença com relação ao procedimento de geração da codificação anterior é que neste caso, cada linha contribuinte é acessada apenas nas posições correspondentes aos seus elementos não nulos, (e não em  $\text{Nonz}(U_{i,*})$  que seriam consideradas na alternativa baseada na varredura dinâmica de cada coluna).

Percebe-se portanto, que o volume de operações do novo procedimento de geração de códigos apresentado, é da mesma ordem que o número de operações de flutuante que seriam efetuadas ao longo de todo o processo, numa fase de fatoração numérica.

● *Ou seja, para se determinar a informação simbólica adicional na forma da codificação binária considerada nesta seção, gasta-se da ordem de uma fatoração numérica adicional em termos de overhead medidos a nível de tempos de CPU.*

Cabe notar porém, que esta fase simbólica de codificação, só precisa ser efetuada uma única vez, para matrizes com a mesma disposição estrutural que porventura venham a ser fatoradas numericamente pelo procedimento IV.1(3) por exemplo, que é justamente o caso para onde as abordagens presentes neste trabalho são voltadas.

O que precisa ser melhorado nos procedimentos considerados nesta seção, é o tratamento especial, de características capazes de viabilizar uma maior "compactação" de informações, como no caso da presença de estruturas do tipo *supernodal*, e onde se viu na seção III.3 que a redução obtida nestes casos pode ser bastante compensadora.

Um comentário de natureza um tanto distinta das efetuadas ao longo deste trabalho, é com relação a uma maior estabilidade numérica nas implementações de alternativas baseadas no método de acumulação por produtos escalares, como se considerou nesta seção.

Vale a pena notar, que uma abordagem via o método de

Crout, oferece um maior benefício em termos de melhor estabilidade numérica para o processo, pois as operações de acumulação das combinações lineares entre os diversos vetores subtraídos do vetor base, podem ser feitas em "precisão dupla" (nas linguagens que contam com a opção de tal recurso), o que significa que somente o valor final do somatório de todas as contribuições sobre um dado elemento, precisarão ser "truncados" e armazenados no espaço de armazenamento reservado ao fator correspondente.

Ou seja as operações intermediárias podem ser efetuadas com maior precisão do que nas demais abordagens consideradas nos capítulos anteriores deste trabalho.

## IV.2 Exploração da árvore de eliminação

Nesta seção será apresentada uma estrutura auxiliar de fundamental importância em vários aspectos relacionados ao processamento de matrizes esparsas simétricas (definidas positivas).

Tal estrutura recebeu o nome de *árvore de eliminação* na literatura [L11], (tendo sido originalmente apresentada na forma como é conhecida atualmente por [S1]).

Suas aplicações incluem a determinação de quais linhas necessitarão ser novamente eliminadas, caso apenas algumas linhas da matriz original venham a sofrer alteração em valor numérico de uma aplicação para outra do método de eliminação.

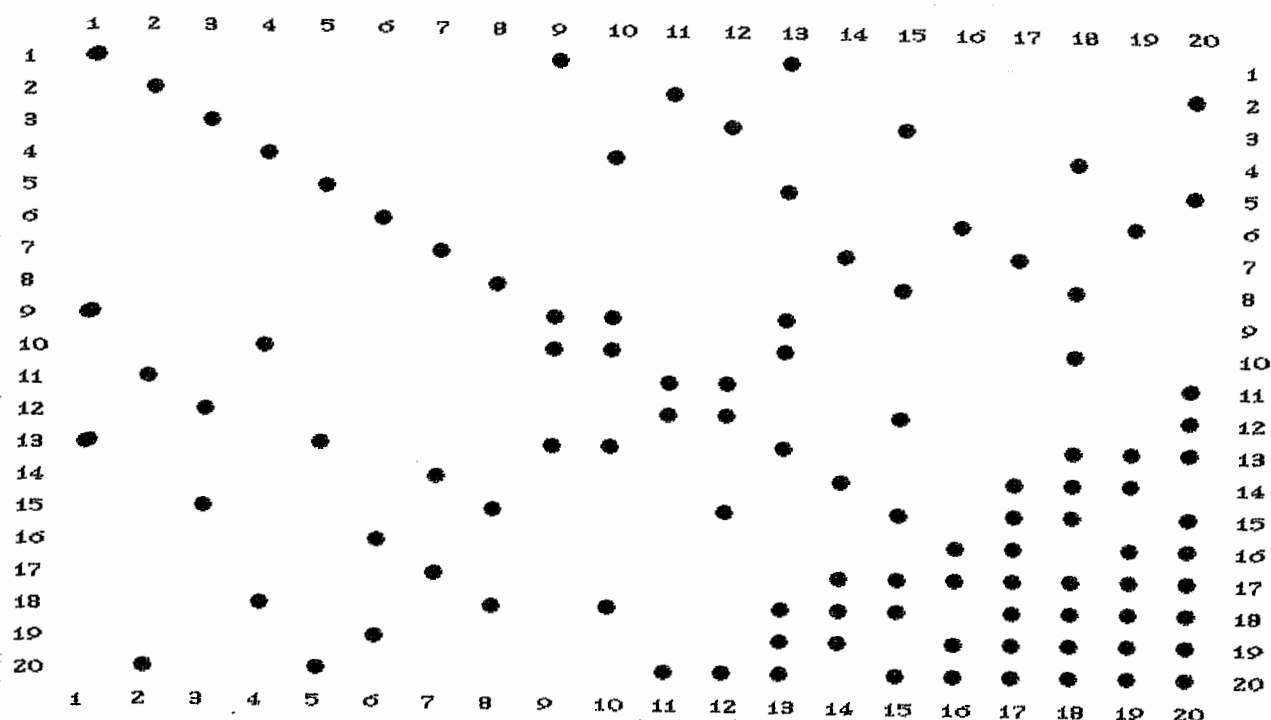
Métodos explorando este fato receberam o nome de métodos de refatoração parcial baseados em *sparse vectors* [B8], [G27], [G28], pelo fato de se explorar de forma eficiente a estrutura de esparsidade do vetor lado direito e efetuar-se apenas as operações indispensáveis sobre os elementos alterados na matriz de fatores.

Em processamento paralelo, a "árvore de eliminação" é utilizada como base para se determinar o mapeamento das linhas do sistema original, entre os vários processadores.

No contexto do presente trabalho, a árvore de eliminação será utilizada para se aumentar a eficiência de métodos baseados na estratégia de Crout, no sentido de se obter uma codificação binária mais "compacta" de modo a caracterizar as operações de acumulação na forma de produtos escalares, quando o número de componentes envolvidos for elevado em comparação com o valor máximo permissível pela codificação adotada.

Básicamente o que a *árvore de eliminação* determina, são relações de dependência entre linhas da matriz de fatores  $U$  resultante.

Para ilustrar este conceito, e introduzir a noção da *árvore* associada, convém lançar mão de uma matriz de fatores (de dimensão  $n = 20$  no caso em particular) como exemplo, e apresentada na fig\_IV(5).



fig\_IV(5) - Matriz de fatores de dimensão 20

A informação que se busca especificar com a árvore de eliminação, é a da dependência entre linhas do sistema, de modo a se saber quais linhas são "influenciadas" e afetadas por outras, de modo a se poder determinar que linhas podem ser eliminadas antes ou independentemente de outras.

A informação na forma do conjunto de todas as linhas subsequentes que uma dada linha afeta não é possível de se representar na forma de uma árvore, visto que cada linha pode afetar um número qualquer de outras, e no sentido oposto, uma dada linha pode depender de eliminação de várias outras linhas prévias.

O que se busca portanto é uma estrutura compacta (como uma árvore por exemplo), que de algum modo possa caracterizar as relações de dependência entre as diversas linhas.

Para tal, a informação considerada será apenas a da primeira linha (em valor crescente dos índices) que dependa diretamente de uma dada linha (cuja relação de dependência com relação as demais esteja sendo procurada).

Ou seja, no caso da matriz do exemplo em questão



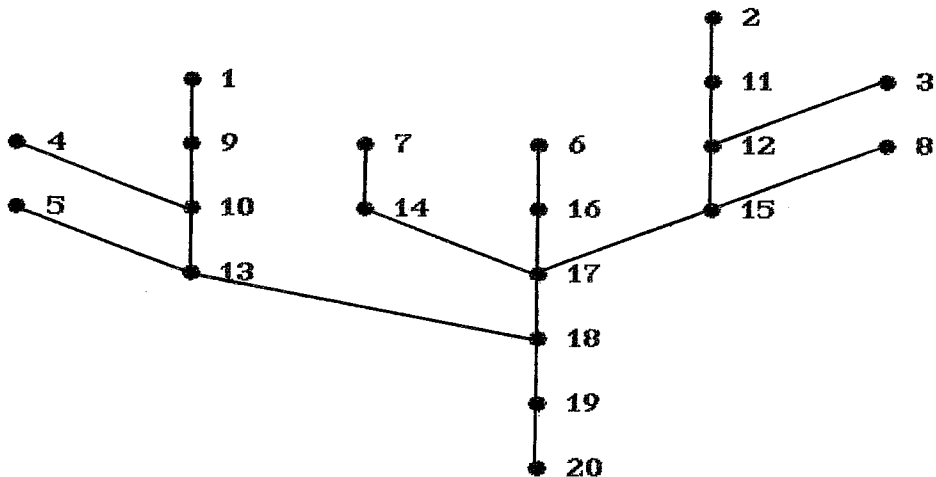
(apresentado na fig\_IV(5)), e tomando-se por exemplo a sua primeira linha, o que se busca é a informação da primeira linha a depender explicitamente da primeira.

Percebe-se pelo exemplo, que a linha 9, é a primeira a conter um elemento não nulo na primeira coluna, de modo que será a primeira linha a depender diretamente da linha 1.

Para o caso da segunda linha, percebe-se que a primeira linha a depender diretamente desta é a 11.

No caso da terceira linha, a primeira dependência desta, ocorre na linha 12.

Levando-se adiante este processo, e estabelecendo-se uma ligação entre cada linha "ancestral" e sua primeira "descendente" direta (a ser afetada por sua ancestral), ficaríamos com a seguinte "árvore" associada à matriz de fatores de dimensão 20 (apresentada na fig\_IV(5)).



fig\_IV(6) - Árvore de eliminação associada

As informações que se pode extrair desta estrutura são de grande valia como veremos nesta seção e na próxima.

O primeiro fato a se observar é que os nós ancestrais possuem sempre índice inferior ao de seus descendentes (pela própria forma como a árvore foi definida).

O segundo é que pela natureza "estruturalmente aditiva" do processo de eliminação, com cada linha anterior contribuindo sobre a estrutura de elementos não nulos da

linha base "sucessora", a árvore assim apresentada, permite que se determine facilmente o conjunto de linhas que dependem de uma dada linha original, simplesmente se percorrendo a árvore a partir da linha ancestral básica, em direção a raiz.

Todos os nós da árvore (correspondentes a linhas da matriz de fatores) encontrados no caminho intermediário entre um nó ancestral e o nó raiz da árvore em questão, correspondem ao conjunto de linhas que serão afetadas por qualquer alteração na linha básica ancestral.

Em termos de uma representação suficiente para caracterizar completamente a árvore, e permitir que se caminhe com facilidade sobre a mesma (das folhas em direção a raiz), a única estrutura adicional necessária é um vetor PARENT, indicando o sucessor direto de um dado nó na árvore.

No caso da matriz exemplo apresentada, o vetor PARENT assume os valores:

	1	2	3	4	5	6	7	8	9	10
PARENT: (	9,	11,	12,	10,	13,	16,	14,	15,	10,	13,
	11	12	13	14	15	16	17	18	19	20
	12,	15,	18,	17,	17,	17	18	19	20,	0 )

Onde se definiu  $PARENT(n) := 0$  por convenção, e mais especificamente:

para  $i$  de 1 até  $n-1$

$PARENT(i) := JUCIUC(i)$

(Assumindo-se que a estrutura de colunas dos fatores em JU esteja devidamente ordenada de forma crescente, e que IU aponte para a posição do primeiro elemento não nulo em cada linha de U).

Mais formalmente:

$PARENT(i) := \min \{ j \mid u_{ji} \neq 0, j > i \}$

ou

$PARENT(i) := \min \{ j \mid u_{ij} \neq 0, j > i \}$

Apenas a informação sobre a coluna do primeiro elemento não nulo de cada linha da matriz de fatores é necessária, visto que a informação da linha abaixo da diagonal correspondente ao primeiro elemento não nulo em cada coluna de  $U^T$  é idêntica a da primeira coluna contendo um elemento não nulo em cada uma das linhas de  $U$ .

Um caminharmento sobre a árvore no sentido das folhas para a raiz, é trivialmente implementado partindo-se de um dado nó  $i$  mediante:

Percurso em direção a raiz

```

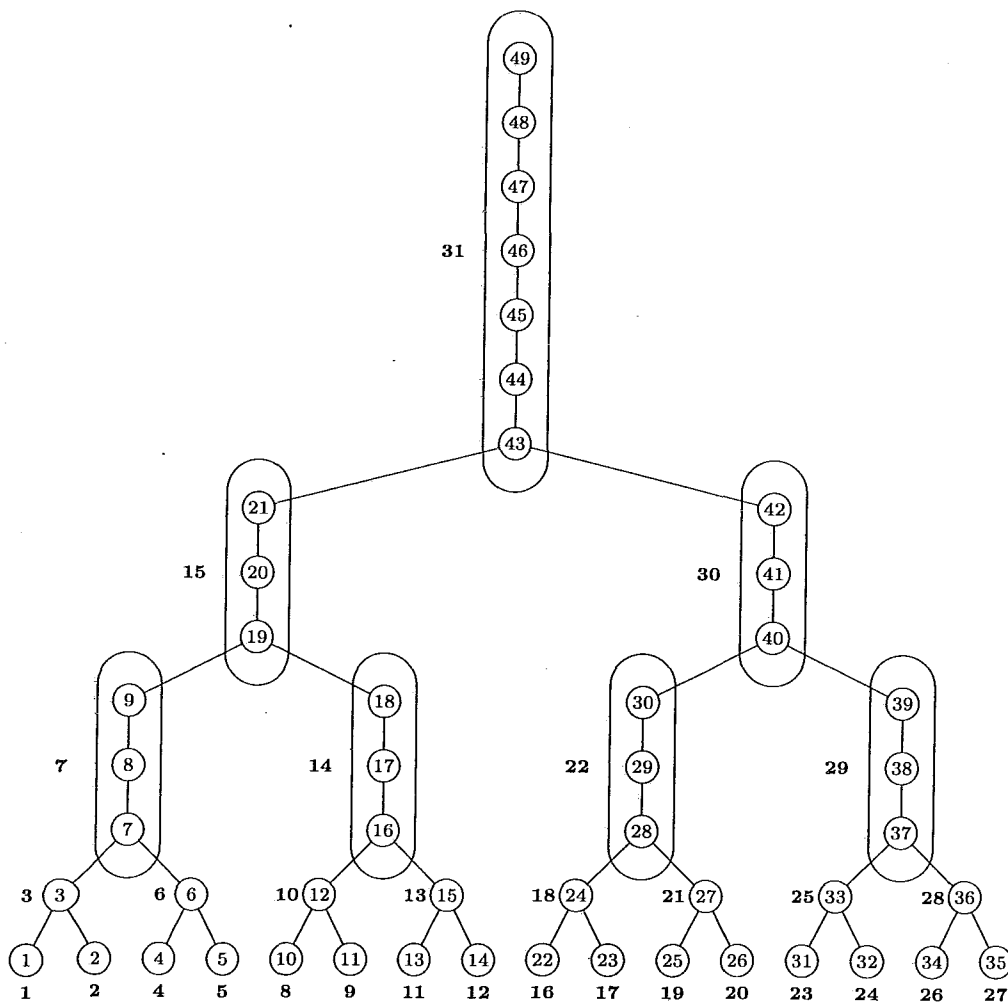
marque o nó  $i$  como visitado
(e inclua-o no caminho)
enquanto  $i \neq n$  faça
     $i \leftarrow \text{PARENT}(i)$ 
    marque o nó  $i$  como visitado
    (e inclua-o no caminho)
fim enquanto
  
```

Algumas outras características da árvore de eliminação são interessantes de se apresentar, visto permitirem uma melhor compreensão do conceito de *supernodes* (apresentado na seção III.3).

Apresenta-se na fig\_IV(7) a seguir, a árvore de eliminação correspondente a matriz exemplo considerada na fig\_III(2).

O que se percebe, é que a estrutura de *supernodes* corresponde a de nós  $j$  consecutivos na árvore de eliminação, tais que  $\text{PARENT}(i) = j$  somente para  $i = j-1$  (com exceção do primeiro nó  $j$ ). Ou seja nós consecutivos, e que só tenham um único predecessor direto (a menos do primeiro nó constituinte de cada *supernode*).

Tanto na fig\_III(2), quanto na fig\_IV(7), os índices em **negrito** correspondem à numeração tomando-se por base a estrutura de *supernodes*, e que se encontram envolvidos por ovais, destacando-os dos demais.



fig\_IV(7) - Árvore associada ao exemplo da fig III(2)

O que se percebe da relação de dependência de estruturas do tipo *supernodal* é que estas são intrinsecamente "seqüenciais" por natureza, onde a "seqüencialidade" considerada não se dá apenas a nível dos acessos a posições de memória (como foi visto na seção III.3), mas sim em termos de uma menor liberdade (ou grau de paralelismo) com o qual se poderia tentar eliminar algumas linhas independentemente de outras.

Em um mesmo *supernode*, tal independência não existe, pois cada linha afeta indistintamente todas as suas sucessoras no *supernode*, e que por sua vez, dependem de todas as demais linhas do mesmo *supernode*.

Ou seja num caso da presença de *supernodes*, o grau de liberdade que se teria na ordem das operações, seria o mesmo que no caso de uma matriz completamente densa.

Um outro fato importante é que a *árvore de eliminação*, não caracteriza a dependência entre linhas da matriz de fatores a um nível mais baixo ou elementar de "granularidade" a nível de cada uma das operações de ponto flutuante efetuadas, mas sim num nível mais elevado.

Deste modo os comentários sobre a "inflexibilidade" de se explorar algum paralelismo no caso da presença de estruturas *supernodais*, é na verdade "miope" sob uma ótica mais elementar ao nível das operações básicas a cada etapa.

Este tipo de paralelismo, (a nível de cada uma das operações básicas efetuadas sobre os elementos), pode ser explorado, em implementações eficientes, simplesmente liberando-se a porção já atualizada da linha base, de modo a se permitir a sua contribuição em outras linhas, (que de outra forma, ficariam a aguardar que toda a linha base viesse a ser gerada para poder prosseguir).

● *A principal característica da árvore de eliminação, se dá no sentido de exibir plenamente o grau de "parentesco" inerente à matriz de fatores em particular, e mediante um percurso "controlado" na estrutura de PARENT, se permitir determinar a estrutura final de uma linha sucessora de cada linha inicial na árvore.*

É justamente esta propriedade "intrínseca" em função da natureza estruturalmente aditiva processo de eliminação como um todo, que será explorada no final desta seção e na abordagem proposta na próxima seção.

Para tal, tomemos como exemplo um caminho partindo-se da linha 2 em direção a raiz, e observemos como se comporta o padrão estrutural de todas as linhas envolvidas durante o percurso.

A seqüência de linhas a serem visitadas é:

2    11    12    15    17    18    19    20

Assim, dispondo-se a estrutura de cada uma destas linhas teremos o seguinte padrão (considerando-se apenas a porção triangular superior de cada uma excluindo-se a digonal):

	11	12	13	14	15	16	17	18	19	20
2	●									●
11		●								●
12					●					●
15							●	●		●
17								●	●	●
18									●	●
19										●

fig\_IV(8) - Matriz reduzida de linhas sucessoras

O que fica evidente a partir desta observação, é que o padrão estrutural da coluna 20 nas linhas envolvidas é de uma "previsibilidade" total, pois uma vez que a segunda linha possui o elemento  $u_{2,20}$  não nulo, todas as demais linhas que venham a depender direta ou indiretamente da linha 2, acabarão por conter a posição correspondente a coluna 20 não nula.

É esta a natureza "estruturalmente aditiva" do processo a que se havia referido, pois nota-se que o parágrafo anterior contém uma "entrelinha" um tanto quanto sutil, e de vital importância para a compreensão da mecânica de todo o processo de "amalgamento estrutural" pelo qual vão passando as sucessivas linhas de fatores sendo geradas.

Observa-se que se comentou: "quaisquer linhas que venham a depender direta ou indiretamente da linha 2 em questão".

A dependência direta é facilmente compreensível, pois se uma dada linha base necessita ter a linha 2 subtraída de si, obviamente que todo o padrão estrutural da linha subtraída será "incorporado" ao da linha base.

No exemplo em questão, as linhas 11 e 20 são exemplos típicos do que acabou de se comentar, visto ambas possuírem elementos não nulos na segunda coluna à esquerda de suas respectivas diagonais, indicando "explicitamente" portanto, que a segunda linha necessariamente terá de ser subtraída

de cada uma destas linhas.

O que não é de todo "aparente", é a segunda observação com relação a dependência *indireta* da linha 2 em questão.

Nestes casos, o que se pode notar, é que como a linha 2 teve o seu padrão estrutural incorporado ao da linha 11, todas as linhas que dependerem diretamente da linha 11, (como a 12 por exemplo), terão naturalmente o padrão estrutural da linha diretamente predecessora (11 no caso em particular), bem como o padrão de todas as demais linhas ancestrais na árvore de eliminação.

Esta observação é contundente, pois explica entre outras coisas, uma razão para o "enchimento" da porção da sub-matriz de fatores, pois nestes casos a "herança" estrutural passa a vir de praticamente todas as "famílias" de linhas antecessoras, de modo que como resultado deste processo estruturalmente aditivo (e irreversível), a probabilidade de uma propagação "explosiva" dos padrões das linhas finais é muito maior do que nas linhas iniciais do processo, onde apenas os seus sucessores mais próximos passam a herdar o o padrão estrutural da linha ancestral.

Esta propriedade de "propagação estrutural aditiva", explica também o aparecimento de *supernodes* por exemplo, uma vez que o padrão estrutural de uma linha sucessora na melhor das hipóteses é mantido inalterado, caso este padrão seja idêntico ao da linha ancestral contribuinte, que é justamente o que ocorre no caso *supernodal* onde como já se observou na *fig\_IV(7)*, a estrutura de *supernodes* é determinada por nós consecutivos, e que só possuam um ancestral *direto* (excluindo-se deste caso o nó mais ancestral do *supernode*), correspondendo neste novo exemplo considerado aos nós 18, 19 e 20 por exemplo.

O que a natureza de propagação aditiva da estrutura das linhas de fatores traz é um conceito intrínseco muito poderoso, e que passaremos a explorar plenamente na próxima seção.

Simplesmente, (embora com a dimensão do exemplo apresentado, não dê para o leitor se convencer tão facilmente das constatações apresentadas a seguir), a propriedade que mais uma vez permite que de algum modo se reformule o conceito de *esparsidade* e de "aparente aleatoriedade" ou não

determinismo no padrão de acesso aos elementos da linha base, é que num grupo restrito as linhas descendentes de um mesmo nó ancestral, o padrão de contribuições é deveras mais previsível e determinístico do que se poderia imaginar até então.

● *A constatação é que a partir da presença de um elemento não nulo em uma dada coluna de uma das linhas contribuintes consideradas, a partir deste ponto, este padrão estrutural não nulo passará a se propagar em todas as linhas subseqüentes (e descendentes diretas ou indiretas na árvore de eliminação).*

Ou seja o que se constata, e que para os que alguma vez já foram apresentados ao jargão utilizado com frequência nas aplicações de engenharia civil, ligadas a eliminação esparsa, é um conceito análogo ao método de "envelopes", e que observaremos a seguir, e que diz simplesmente que se pode adotar uma caracterização para alguma porção da matriz de fatores, simplesmente se especificando a primeira linha a conter um elemento não nulo em cada uma das colunas.

A propriedade observada com relação a natureza aditiva das contribuições permite que este fato seja plenamente explorado, porém não na matriz de fatores original, mas sim numa estrutura "reduzida", contendo apenas as linhas descendentes em uma mesma "família" (ou seja descendentes de um mesmo nó ancestral).

O que a propriedade acima permite, é que se caracterize cada uma das colunas da matriz "reduzida", mediante apenas um índice, justamente o da primeira linha a partir da qual a coluna em questão tornará-se completamente densa deste ponto até a última linha contribuinte na estrutura reduzida.

Esta é uma propriedade de extrema valia, pois além de se conseguir "domar" o que "aparentemente" parecia dominado por uma natureza "caótica", se consegue o maior benefício que se esteve a almejar desde a apresentação dos métodos da seção IV.1, ou seja alguma forma de "compactação" das informações necessárias para se caracterizar o processo de acúmulo das contribuições nos somatórios correspondentes aos produtos escalares geradores de cada coluna.



Com o resultado observado nesta seção, percebe-se que com um mesmo tamanho de palavra usado para uma codificação anteriormente binária, se pode na verdade representar muito mais informação.

Para tal, assumindo-se que se disponha de uma palavra de codificação com  $n_{\max}$  bits, no caso binário o que se consegue tratar são acumulações com no máximo  $n_{\max}$  elementos, visto que todas as combinações possíveis entre tais elementos deve ser levada em consideração, por justamente não se saber a priori qual padrão será necessário.

No caso da exploração da árvore de eliminação, a quantidade de informação é reduzida logaritmicamente, pois o único padrão estrutural que se precisa considerar, é o de porções completamente densas em cada coluna, a partir de uma certa linha. Ou seja nestes casos, basta se especificar a posição relativa a primeira contribuição não nula em cada coluna da matriz reduzida, que daí para adiante, sabe-se de antemão toda a seqüência de operações a ser efetuada para o acúmulo das contribuições relativas ao produto escalar gerador desta coluna.

Simplesmente, neste caso, tomando-se a mesma palavra de codificação com  $n_{\max}$  bits, percebe-se que se consegue univocamente especificar completamente o processo de acumulação com até  $2^{n_{\max}}$  elementos, ou seja um sensível acréscimo em relação a uma codificação binária como a que se havia considerado até então.

A codificação apresentada nesta seção, passará a ser denominada codificação "ancestral" de modo a se fazer distinção com a forma binária.

O que se passará a considerar na próxima seção, é simplesmente se aplicar este novo conceito numa abordagem completamente inédita do que já foi apresentado até então.

### IV.3 Reordenamento ancestral das contribuições

Nesta seção se apresenta um novo método de eliminação do tipo Crout simétrico, baseado porém nas informações de origem ancestral disponíveis a partir da árvore de eliminação apresentada na seção anterior.

Com o conhecimento das informações de descendência ou ancestralidade obtidas com a árvore, pode-se lançar mão de uma codificação realmente "compacta", visto que a estrutura de linhas contribuintes descendentes de um nó ancestral comum, possuem um padrão estrutural "aditivo", o que permite que numa representação "reduzida" (como a apresentada na seção anterior), levando em conta apenas as linhas contribuintes de uma mesma "família", se possa adotar uma caracterização na forma de "envelope", ou seja, especificando-se apenas a posição reduzida inicial da linha contendo a primeira contribuição não nula em cada coluna da linha base sendo gerada.

Percebe-se que a chave para se explorar tal forma de codificação, é lançando mão de uma estratégia que será denominada "reordenamento ancestral", e que consiste simplesmente em se reordenar as colunas da representação da transposta de  $U$ , (correspondendo aos índices das linhas contribuintes no processo de eliminação da linha base), de tal sorte que a nova ordem venha a ser ditada em função do grau de parentesco de cada uma das linhas envolvidas.

Com esta etapa de reordenamento, que será detalhada um pouco mais adiante, percebe-se que o que fica faltando para se adotar o novo esquema de codificação, e incorporá-lo numa estratégia como a IV.1(3) é simplesmente se efetuar múltiplas varreduras na linha base, com cada uma associada a uma dada "família" de contribuições (contendo um padrão estrutural aditivo comum), repetindo-se o processo para o número de famílias "distintas" envolvidas no processo de eliminação da linha base.

Óbvio, em face dos mesmos argumentos apresentados na primeira seção deste capítulo, (quando da exposição do método de codificação binária), poderia ser questionado se com a introdução de múltiplas varreduras, o desempenho final, especialmente a nível de espaço de armazenamento, acabaria

por ser de certa forma degradado, tornando a nova estratégia impraticável, ou no mínimo não competitiva, com as demais apresentadas até então.

Neste ponto cabe uma observação "empírica", (e que confere a nova abordagem um caráter merecedor de uma investigação mais detalhada), de que o número de famílias "distintas" (correspondendo ao número de varreduras necessárias), é na verdade pequeno em comparação com o número total de linhas contribuintes.

Um "pior" caso, só ocorreria, quando todas as linhas contribuintes fossem provenientes de ramos completamente distintos e independentes da árvore.

Tal situação pode ocorrer, porém na prática nunca ao longo de todo o processo de eliminação, pois na maioria dos casos, a largura máxima da árvore por mais elevada que seja, representa apenas uma fração do número total de nós, e para que sempre houvessem nós provenientes de ramos distintos a cada etapa, seria preciso que a árvore viesse a ter altura próxima da unitária, com praticamente todas as linhas independentes uma das outras.

Deixemos por ora estas questões relativas aos possíveis "piores casos", e que serão levadas a tona oportunamente mais adiante, visto que o que aparentemente constitui o "pior caso", também oferece indícios de que pode ser de certa forma "contornado".

Assumindo-se portanto que um número suficiente de varreduras, correspondendo a cada uma das famílias distintas de contribuições a serem acumuladas, venha a ser tolerado no processo de solução, o que fica faltando é se especificar como proceder a este "reordenamento ancestral" em questão.

Para tal, consideremos como exemplo, a mesma matriz da fig\_IV(5), e cuja árvore de eliminação foi apresentada na fig\_IV(6).

Vamos analisar o processo de reordenamento de duas linhas base para fins de exemplificação.

Inicialmente tomemos a linha 18, e vejamos como proceder a uma reordenação dos índices JUT associados a esta linha.

Originalmente o vetor JUT contém a informação de todas as linhas contribuintes, ordenadas crescentemente, (como se obervou na seção II.3, após a fase de fatoração simbólica).

Para o exemplo em questão:

linha 18 (original)

**JUT:**           4       8       10       13       14       15       17

A idéia consiste em se ir percorrendo a árvore a partir das linhas de menor índice presentes em **JUT**, visto serem estas as candidatas mais potencialmente indicadas a serem "ancestrais" de outras linhas presentes.

Neste caso, a primeira linha considerada seria a 4, que por ser uma "folha" da árvore, seguramente é uma linha ancestral.

Assim, inicia-se o procedimento, incluindo-se o índice da linha 4 numa estrutura auxiliar, e que ao final do processo acabará contendo a nova ordenação desejada.

Ou seja, inicialmente teríamos:

**JUTORD:**   4

A partir deste ponto, o que se deve fazer, é percorrer a árvore de eliminação a partir do nó 4, em direção a raiz, e ir adicionando-se possíveis outras linhas presentes no vetor original **JUT** e encontradas como descendentes da linha 4 no exemplo em particular.

Neste caso, levando-se o processo adiante, encontraríamos as linhas 10 e 13 durante o percurso, de tal forma que o vetor **JUTORD** ao final desta etapa, acabaria contendo:

**JUTORD:**   4       10       13

Como todos os descendentes da linha 4 foram incluídos, ao final desta etapa de percurso na árvore, terá se estabelecido a primeira família de contribuições, que por construção, possuem uma disposição estrutural da forma aditiva, permitindo que uma codificação na forma de "envelopes" possa ser seguramente empregada.

A etapa seguinte é continuar a varrer o vetor **JUT**

original, em busca de novas linhas ainda não consideradas, e que portanto pertencerão a famílias distintas da primeira encontrada (e a esta altura dada por completamente encerrada).

O processo retoma ao ponto em que uma vez encontrada uma nova linha em **JUT**, inclui-se a mesma no vetor **JUTORD** com alguma indicação de que esta pertence a uma nova família a ser explorada.

Percorrendo-se a árvore a partir deste ponto, (no caso a partir da linha 8) se obteriam as descendentes desta, repetindo-se todo o processo apresentado nos paragrafos anteriores.

Para o exemplo em questão, lançando-se mão do artifício de se representar as linhas de cada uma das famílias mediante o uso de um separador como | teríamos após o segundo percurso efetuado:

**JUTORD:** 4 10 13 | 8 15 17 |

A última linha presente em **JUT** e ainda não incluída no novo vetor ordenado é a de número 14, de sorte que o vetor **JUTORD** final, associado a linha 18, passaria a ser:

*linha 18 (ordenada ancestralmente)*

**JUTORD:** 4 10 13 | 8 15 17 | 14 |

Apenas a título de realçar a mecânica do processo, apresentam-se resultados similares que seriam obtidos no caso de se considerar um reordenamento para a linha 20.

*linha 20*

**JUT:** 2 5 11 12 13 15 16 17 18 19  
**JUTORD:** 2 11 12 15 17 18 19 | 5 13 | 16 |

Um ponto que vale a pena ressaltar, é que o "reordenamento ancestral" não é "único", no sentido de que outras escolhas para os ancestrais iniciais, acabarão determinando "famílias" com um número distinto de componentes dos obtidos com a estratégia proposta originalmente.

Ou seja, no exemplo da linha 20 em questão, reportando-se a árvore de eliminação na fig\_IV(6), percebe-se por exemplo que o nó 16, embora não sendo uma das folhas, é um candidato potencial a "patriarca" de uma família contendo a si próprio e os nós 17, 18 e 19 como descendentes.

Se observarmos o ordenamento original apresentado em JUTORD, veremos que o nó 16 só acabou pertencendo a uma última família isolada, simplesmente porque os nós anteriores já haviam sido incluídos em outras famílias.

Uma outra alternativa para o ordenamento seria por exemplo:

linha 20 (ordenamento alternativo)

JUTORD: 16 17 18 19 | 2 11 12 15 | 5 13 |

O que a estratégia originalmente apresentada garante é que por construção, se estará especificando uma família inteira ao se concluir um percurso na árvore.

A ordem com que se pode especificar as famílias oferece mais liberdade do que a particular estratégia considerada, portanto.

O problema da "escolha ótima" da ordem de especificação das famílias, é ainda um problema em aberto, e não será considerado formalmente ou em maior detalhe neste trabalho, sendo objeto portanto de investigações futuras.

O que se percebe no entanto é que esta característica ao invés de oferecer "restrições", na verdade viabiliza o oposto, pois confere flexibilidade o bastante para se poder melhor escolher em que família incluir uma dada linha, com critérios de desempate baseados por exemplo em alguma maior "similaridade" da natureza estrutural, que venha possibilitar uma compactação ainda maior da codificação necessária para se caracterizar o processo de acúmulo das contribuições.

Uma observação que se pode fazer, com respeito a escolha de algumas "famílias" em particular, se dá por exemplo na presença de características *supernodais*.

Neste caso, percebe-se claramente que o *supernode* inteiro é a família mais potencialmente indicada a se considerar.

A estratégia de construção do vetor **JUTORD** apresentada nesta seção, atende a esta observação, pois como *supernodes* são necessariamente nós contíguos na estrutura da árvore (como se observou na seção anterior), o primeiro percurso descendente em direção a raiz que passar por algum dos integrantes de um *supernode*, certamente passará por todos os demais integrantes, de modo que todos os nós do *supernode* acabarão sendo alocados na mesma família.

O único critério com certa liberdade de escolha, é saber quais outros nós antecessores do *supernode* incluir na família associada ao mesmo.

No exemplo em questão, percebe-se que os nós 18, 19 e 20, constituem um *supernode*. A questão onde se oferece liberdade (em função da estratégia de construção do vetor **JUTORD** a ser empregada), é saber se é mais vantajoso incluir-se este *supernode* na família de descendentes dos nós 2, 16 ou 5.

Passaremos a partir deste ponto, a considerar uma formalização maior do procedimento de ordenação proposto originalmente, tendo em vista que por construção, a escolha dos índices de menor valor como candidatos a patriarcas iniciais, viabiliza uma solução para o problema, e que a princípio não se mostra inferior as demais possibilidades de reordenamento não exploradas neste trabalho.

Um ponto que merece ser destacado com relação ao processo de percurso na busca por descendentes em direção a raiz, é que o mesmo não precisa ser efetuado até se atingir o nó  $n$  (raiz), pois o que se busca a cada etapa básica  $i$  do processo de eliminação, é o reordenamento ancestral das linhas contribuintes na linha base  $i$ , e que portanto necessariamente devem possuir índices inferiores a  $i$ .

Deste modo, o processo de percurso pode ser "truncado", a partir do ponto em que se alcançar o nó  $i$  correspondente a etapa básica do processo sendo considerada.

Na verdade como veremos ao longo da especificação completa das estruturas auxiliares e etapas a serem efetuadas no processo de obtenção do vetor **JUTORD**, o "truncamento" pode ser ainda mais estendido, pois se percebe que uma vez visitando um dos nós, todos os seus descendentes subsequentes acabarão sendo visitados.

Assim, em um novo percurso pela árvore, ao se encontrar um nó já visitado (durante a especificação de uma outra família), pode-se truncar o processo de busca por descendentes para a nova família, pois os nós subseqüentes, por construção, já terão sido incluídos em alguma das famílias anteriores.

Esta possibilidade de truncamento, mostra que um algoritmo completo de reordenamento, não deve exibir um mal comportamento na forma de um número intolerável de etapas para se alcançar uma solução final.

Sem o truncamento, certamente este fato não ocorreria, razão pela qual a primeira estrutura auxiliar a se considerar na implementação do algoritmo de reordenação, é lançar mão de um vetor VISITED, de dimensão  $n$ , indicando os nós já visitados durante algum percurso.

Outra estrutura fundamental é um vetor auxiliar expandido CONTRIB, indicando as linhas que efetivamente contribuem sobre a linha base corrente.

Este vetor será utilizado para uma rápida determinação se cada um dos nós sucessores visitados na árvore pertencem de fato a estrutura da transposta de  $U$  na linha base (ou seja, se o nó em questão corresponde a uma linha contribuinte ou não).

Assumindo-se o vetor CONTRIB inicialmente como falso para todas as suas posições, inicializa-se com o valor verdadeiro as posições correspondentes as linhas associadas aos valores de JUT da linha base corrente.

Do mesmo modo, inicializa-se o vetor VISITED como falso para todas as suas  $n$  posições.

Na verdade a inicialização para todas as  $n$  posições como falso tanto para CONTRIB como para VISITED, só precisarão ser efetuadas uma única vez durante todo o processo de eliminação, visto que o final de cada reordenamento ancestral na  $i$ 'ezima etapa do processo, os valores porventura modificados nestes vetores, acabarão sendo restaurados para seus valores originais (falsos).



Se tal não ocorresse, teríamos um algoritmo de natureza apenas "teórica", pois caso fosse necessário reinicializar todas as  $n$  posições a cada etapa  $i$  do processo, acabaria se efetuando  $n^2$  operações apenas de inicialização, enquanto que o leitor deve se recordar da seção I.2, que uma estimativa para o volume de operações efetuadas em um processamento tipicamente esparsa deve ser da ordem  $O(n)$  para se mostrar atraente ou competitivo.

Visando auxiliar o processo de restauração do vetor VISITED ao final do reordenamento, torna-se necessário a introdução de mais um vetor auxiliar LVIS também de dimensão  $n$ , e que originalmente não necessita ser inicializado. Tal vetor, será utilizado para se armazenar os índices de linhas visitadas que não pertençam a estrutura de linhas contribuintes da linha base a ser eliminada.

O procedimento produz como resultado um vetor JUT reordenado "ancestralmente", e vetores IFAM e KFAM especificando o número de famílias a serem utilizadas na geração de cada linha base  $i$ , e as posições em JUT do início de cada família.

O vetor JUTORD de dimensão  $n$  utilizado, contém apenas uma cópia local reordenada do elementos de JUT a cada etapa básica, e que são reescritos de volta ao vetor original ao se completar o reordenamento da etapa associada.

Uma vez atualizado o vetor JUT, as posições utilizadas em CONTRIB e VISITED são restauradas para seus valores originais (como falso), para serem utilizadas no reordenamento da próxima etapa básica.

No procedimento a ser apresentado, assume-se também que o vetor PARENT indicando a topologia da árvore de eliminação e apresentado na seção anterior, esteja também disponível, de modo a permitir um fácil deslocamento em direção a raiz.

**Procedimento IV.3(1)****Reordenamento "ancestral"**

para j de 1 até n faça  
 CONTRIB(j) ← falso  
 VISITED(j) ← falso

IFAM(1) ← 1  
 KFAM(1) ← 1

para i de 1 até n faça

para k de IUT(i) até IUTF(i) faça  
 CONTRIB(JUT(k)) ← verdadeiro

k ← IUT(i)  
 kf ← IUTF(i)

l ← JUT(k)  
 lf ← JUT(kf)

VISITED(PARENT(lf)) ← verdadeiro  
 VISITED(i) ← verdadeiro

nl ← 0  
 nvis ← 0  
 nfam ← 0

$\alpha$ : VISITED(l) ← verdadeiro  
 nl ← nl + 1  
 JUTORD(nl) ← l

$\beta$ : l ← PARENT(l)  
 se VISITED(l) vá para  $\gamma$   
 se CONTRIB(l) vá para  $\alpha$   
 VISITED(l) ← verdadeiro  
 nvis ← nvis + 1  
 LVIS(nvis) ← l

vá para  $\beta$

$\gamma$ : nfam ← nfam + 1  
 KFAM(IFAM(i) + nfam) ← IUT(i) + nl

$\delta$ : k ← k + 1  
 se k > kf vá para  $\varepsilon$   
 l ← JUT(k)  
 se VISITED(l) vá para  $\delta$   
 vá para  $\alpha$

$\varepsilon$ : j ← 1  
 para k de IUT(i) até IUTF(i) faça  
 CONTRIB(JUT(k)) ← falso  
 VISITED(JUT(k)) ← falso  
 JUT(k) ← JUTORD(j)  
 j ← j + 1

VISITED(PARENT(lf)) ← falso  
 VISITED(i) ← falso

para j de 1 até nvis faça  
 VISITED(LVIS(j)) ← falso

IFAM(i+1) ← IFAM(i) + nfam

fim para i

Uma sutileza de codificação foi empregada no procedimento apresentado, visto desta forma eliminar-se comparações desnecessárias, e que seriam efetuadas durante o percurso pela árvore.

Além de se indicar o ponto de "truncamento" no percurso a partir do nó correspondente a etapa básica *i* como havia sido sugerido, percebe-se que na verdade o percurso só precisa ser efetuado até se encontrar um nó correspondente a linha contribuinte de maior índice presente a cada etapa básica.

A primeira solução seria portanto marcar-se o nó correspondente a `JUT(IUTF(i))` como já visitado, abreviando-se desta forma o processo de percurso.

Porém neste caso, seria necessário um tratamento especial para este nó, visto que uma vez marcado como visitado, o mesmo não seria incluído em nenhuma das famílias determinadas pelo processo.

A solução definitiva e que foi empregada no procedimento IV.3(1) é a de se marcar o descendente deste último nó como já visitado, o que se dá mediante a instrução `VISITED(PARENT(lf)) ← verdadeiro` no pseudo-código apresentado.

O vetor LVIS mostra-se necessário, pois ao final do processo, é necessário restaurar-se as posições marcadas como visitadas, pois as únicas posições em que seguramente se pode saber de antemão terem sido visitadas, são as correspondentes aos nós contribuintes sobre a linha básica de cada etapa.

Como outros nós intermediários durante o percurso pela árvore acabarão visitados, o vetor LVIS é utilizado como uma forma de armazenar esta informação de linhas "não contribuintes" visitadas, e que de outro modo seria irremediavelmente perdida.

Uma solução alternativa (eliminando o uso do vetor LVIS) porém não adotada por poder induzir a um número adicional de visitas a nós da árvore, seria simplesmente evitar a marcação como "visitados" dos nós que não pertençam a estrutura de contribuintes da da linha básica de cada etapa.

Neste caso, incorreria-se no *overhead* de se caminhar novamente por trechos já visitados, e que por não conterem elementos contribuintes, acabariam podendo ser revisitados

desnecessariamente em novos percursos.

O uso do vetor LVIS foi adotado portanto, visto ser este um vetor de dimensão no máximo  $n$ , e a possibilidade de economia de esforço computacional com a rotulação de todos os nós visitados ser muito mais compensadora.

O último comentário sobre o código acima, se dá com relação a atualização dos vetores IFAM e KFAM.

O primeiro vetor, indica a posição inicial a cada etapa básica  $i$ , dos apontadores KFAM a serem utilizados para se determinar cada uma das famílias presentes nesta etapa.

Ou seja, IFAM é um vetor de apontadores para posições em KFAM.

O vetor KFAM por sua vez, aponta para as posições iniciais de cada família tendo como endereço básico as posições ocupadas pela representação da estrutura reordenada de contribuições no vetor JUT.

Ou seja, KFAM é um vetor de apontadores para posições em JUT.

Inicialmente ambos os vetores apontam para uma posição inicial unitária, e a medida que cada família vai sendo determinada, o apontador KFAM vai sendo atualizado, tomando como base um endereço relativo a  $IUT(i)$ , e que indica a posição inicial de armazenamento em JUT correspondente a  $i$ 'ezima etapa básica.

A variável  $n_i$  contém o número total de linhas já incluídas no vetor reordenado JUTORD, e ao final do processo,  $n_i$  corresponde ao número de elementos não nulos da estrutura de JUT original, associada à linha básica. Ou seja ao final de cada etapa de ordenamento ancestral  $n_i = IUTF(i) - IUT(i) + 1$ . Deste modo, a posição apontada pelo vetor KFAM relativa a última família detectada, corresponde à:

$$\begin{aligned} KFAM(IFAM(i) + n_{fam}) &\leftarrow IUT(i) + n_i \\ &= IUT(i) + (IUTF(i) - IUT(i) + 1) \\ &= IUTF(i) + 1 \end{aligned}$$

Para que este fato possa ser explorado, assume-se portanto que a representação da transposta de  $U$  esteja na forma sequencial "contígua", com  $IUTF(i) + 1$  correspondendo a  $IUT(i + 1)$ . Caso contrário, uma linha adicional de código correspondendo à  $KFAM(IFAM(i)) \leftarrow IUT(i)$  deveria ser adicionada logo após a instrução  $n_{fam} \leftarrow 0$  imediatamente antes

do ponto  $\alpha$ .

O trechos rotulados no programa, correspondem as diversas etapas já consideradas, com  $\alpha$ : perfazendo a inclusão de um novo nó na posição ordenada em JUTORD,  $\beta$ : correspondendo ao *loop* básico de percurso pela árvore, com desvios para  $\gamma$ : caso se tenha detectado um indicador de fim de percurso, ou adicionando-se o nó em uma nova posição de JUTORD (caso o mesmo pertença a estrutura de nós contribuintes da linha base).

Ao se chegar ao trecho  $\gamma$ : significa que uma família foi completamente explorada na árvore, de modo que os apontadores correspondentes são incrementados, e passa-se a procurar um novo elemento na estrutura de linhas contribuintes, e que ainda não tenha sido visitado (ou seja não pertença a nenhuma das famílias já detectadas até então). Neste caso retorna-se ao ponto  $\alpha$ : na hipótese de se encontrar um elemento ainda não incluído, caso contrário, desvia-se para o trecho final do código em  $\varepsilon$ : quando o vetor JUT é reescrito com os novos valores reordenados, e as posições utilizadas nos vetores CONTRIB e VISITED são restauradas para seus valores originais.

Antes de se iniciar uma nova etapa, o vetor IFAM é atualizado, indicando a posição de início da representação de famílias da próxima linha básica a ser considerada, retornando-se ao trecho inicial do código, com o avançar de uma nova etapa  $i$ .

Um fato merecedor de destaque e de uma análise futura, é se determinar uma estimativa do volume médio de operações efetuadas pelo algoritmo apresentado, bem como a identificação de seu pior caso.

Pela forma como se procedeu ao truncamento de percursos na árvore no entanto, estima-se que o volume médio de operações não seja elevado, e seja proporcional a  $Nonz(U)$  ao final da contabilização de todas as etapas do processo.

Outra solução não adotada neste trabalho, porém que se mostra bastante promissora, é ao invés de se adotar uma árvore de eliminação tradicional, adotar-se uma representação *supernodal* para a mesma, com cada grupo de nós correspondentes a um mesmo *supernode*, indicados apenas por um único nó na árvore.

Neste caso, o tamanho médio dos percursos pode ser significativamente reduzido, o que pode vir a justificar a adoção desta nova estrutura, e ser merecedor de um estudo mais aprofundado futuro.

Passaremos a partir deste ponto a considerar a especificação de um novo método de eliminação do tipo Crout, baseado nas idéias de codificação e reordenamento "ancestral" apresentadas nestas duas últimas seções.

Se olharmos o procedimento IV.1(3) mais detalhadamente, a luz das novas estratégias de codificação e reordenamento propostas, veremos que na verdade muito pouco necessita ser modificado com relação ao procedimento original, (voltado para uma codificação da forma "binária").

Na verdade a porção de código que necessariamente tem de ser alterada, é a porção final, correspondendo a cada uma das operações de produto escalar associadas aos códigos.

Ou seja, um mesmo código binário, que antes era usado para se caracterizar operações apenas sobre um determinado conjunto de elementos em particular, agora passará a ser usado para se especificar um grupo contíguo de elementos a serem considerados para efeito do somatório, em função da codificação por "envelopes" utilizada para se caracterizar o processo de acumulação na nova abordagem.

No caso de uma codificação ancestral, como por construção, sabe-se que se uma dada linha  $r$  na representação reduzida, contiver um elemento não nulo numa dada coluna, todas as demais linhas abaixo desta na representação reduzida terão um elemento não nulo na mesma coluna, o que se precisa "codificar" (a nível de programa), são instruções da forma:

$$(4.1) \quad s \leftarrow s + \sum_{r=1}^{cod} u_r * UN(p_r)$$

Para cada um dos valores de  $cod$  possíveis dentro do tamanho de palavra adotado para a codificação.

Ou seja no caso de  $n_{max}$  bits de representação, deveremos ter  $2^{n_{max}}$  trechos de código correspondentes a "explicitação" dos somatórios em (4.1) elemento a elemento.

Na verdade uma forma alternativa de implementação destas operações, será apresentada após a formalização do novo procedimento de eliminação sendo considerado.

O que se percebe com relação a nova estratégia, é que os valores de *kend* especificados em uma lista de códigos auxiliar no procedimento via codificação binária, devem ser simplesmente adaptados de modo a se considerar "famílias" inteiras em cada varredura, com apenas uma família sendo especificada de cada vez.

A hipótese de que a codificação na forma de envelope com apenas um único código não seja suficiente para caracterizar uma família inteira, é possível de ocorrer, porém de forma bem remota, pois no caso de uma codificação com 8 bits por exemplo, precisaríamos ter linhas base contendo mais do que 256 linhas contribuintes em uma mesma família.

Nestes casos, a única solução, é lançar mão de uma nova operação de varredura, dedicada a complementar os acúmulos em todas as colunas da linha base, referentes aos restantes dos descendentes que uma dada família contribuinte possa vir a conter.

Soluções como se tentar incluir nestes casos, outras famílias menos extensas e cujo padrão estrutural se aproxime do da família excedente, de tal forma que uma codificação por envelopes ainda possa ser utilizada para se caracterizar as operações, é uma das opções que se pode considerar, e sem dúvidas merecedoras de atenção no futuro.

Para um caso bem típico de estrutura encontrado com frequência, como o caso *supernodal* por exemplo esta solução pode ser trivialmente aplicada.

A melhor indicação no caso remoto de se vir a precisar de mais de um código por envelopes para se caracterizar as operações em uma família, é considerar as linhas restantes (e ainda não processadas), como as últimas linhas na representação reduzida da próxima codificação.

Este comentário em função do que se mencionou com relação a características da forma *supernodal* por exemplo, se deve a constatação empírica, de que linhas contendo padrões de supernodes, especialmente ao final do processo, possuem um padrão estrutural "quase denso", de forma que muito provavelmente tal padrão deve englobar o de linhas com menor densidade, provenientes de outros ramos da árvore.

● O fato é que a codificação por envelopes não necessariamente precisa ser feita na forma "ancestral", podendo existir casos de coincidência estrutural de forma tal que linhas de famílias distintas possam ser caracterizadas em apenas um único código.

Ou seja, o que se levanta como hipótese merecedora de investigação, é que além de um reordenamento ancestral, se lance mão de alguma outra estratégia de reconhecimento de padrões, de tal sorte a aproveitar características passíveis de serem exploradas e codificadas na forma por envelope.

Esta portanto é mais uma "flexibilidade" intrínseca da metodologia proposta, como já se havia comentado nesta seção, quando se levantou a hipótese de uma especificação alternativa para os integrantes de cada família, percebendo-se que a princípio não se conhece ainda um critério para especificação de um ordenamento "ótimo" das contribuições.

O que seguramente se garante com a forma de construção apresentada na metodologia proposta, é que se consegue adotar uma representação por envelopes com um determinado número de códigos, caso se lance mão de um ordenamento do tipo ancestral. Uma representação com menos códigos, em certos casos é possível, e este é um assunto para futuras investigações como já se comentou.

Na prática, este comportamento de pior caso quase nunca se verifica, pois nos casos em que seguramente as famílias poderiam vir a se tornar realmente "extensas", como no caso da presença de *supernodes*, formas mais eficientes de tratamento ao problema podem ser empregadas, como as apresentadas na seção III.3 por exemplo.

A principal deficiência de uma codificação por reordenamento "ancestral" da forma como estamos propondo neste trabalho, na verdade tem a ver com um possível número excessivo de varreduras, em função de um número elevado de famílias distintas.

Ou seja não são os casos de "famílias numerosas", que se tem que procurar tratar de forma mais eficiente, mas sim o de



"numerosas famílias" não tão extensas, e que no final poderiam levar o algoritmo a efetuar um número desnecessário de varreduras por toda a linha base.

Deixando de lado o aspecto que já se comentou, de que mesmo o "pior caso", qual seja o de cada linha vir a depender de um número elevado de contribuições provenientes de ramos distintos da árvore, pode vir a ser contornado (o que se comentará mais adiante), concentraremos agora em tentar de alguma forma aprimorar o procedimento IV.1(3), de modo a tornar menos "onerosas" as operações de varredura.

A principal ineficiência que se percebe, é que o *loop* mais interno em  $j$ , é na verdade efetuado para todo o padrão estrutural da linha base sendo gerada.

Ou seja em muitos dos casos, correspondendo a posições na linha base que não venham a sofrer contribuições de nenhuma das linhas anteriores, o que se acabará executando é uma simples operação de desvio para a posição 0: no código (sem acúmulo algum), o que sem dúvida se mostra um *overhead* desnecessário.

A alternativa para se contornar esta questão, já foi apresentada na seção III.2, ou seja, lançando-se mão de listas simbólicas de "endereços", apontando diretamente para as posições na linha base a serem acessadas.

Nos casos em que apenas uma varredura sobre a linha base é suficiente para se caracterizar todas as operações de acúmulo por produtos escalares, a nova informação simbólica introduzida no vetor que passaremos a chamar LSTJADR, será da ordem de  $\text{Nonz}(U)$ , o que sem dúvida é bem inferior à  $\text{TotOper}(U)$  como no caso da abordagem por listas de endereços tradicionais.

Nos casos em que mais de uma varredura se mostra necessário, o tamanho de LSTJADR crescerá proporcionalmente ao número máximo de famílias distintas consideradas.

Esta é portanto uma questão meramente de escolha entre um *tradeoff* em espaço por um em tempos de execução e que será comentada após a listagem do novo procedimento.

**Procedimento IV.3(2)      Acumulação via reordenamento ancestral**

```

ncod ← 0
nk ← 0
nadr ← 1
njadr ← 0
para i de 1 até n faça
  IUP(i) ← IU(i)
  piv ← DI(i)
  kbeg ← IUT(i)
  kf ← IUTF(i)
α: nk ← nk + 1
  kend ← LSTKEND(nk)
  r ← kend - kbeg + 2
  para k de kbeg até kend faça
    r ← r - 1
    l ← JUT(k)
    pr ← IUP(l)
    IUP(l) ← IUP(l) + 1
    umr ← UN(pr) * DI(l)
    piv ← piv - UN(pr) * umr
    UN(pr) ← umr
    pr ← (pr + 1)
  fim para k
  njadr ← njadr + 1
  nadrf ← LSTJADR(njadr)
  para jadr de nadr até nadrf faça
    s ← 0
    ncod ← ncod + 1
    vá para (0, 1, 2, 3, 4, 5, 6, 7)
      em função de LSTCOD(ncod)
β: UN(LSTADD(jadr)) ← UN(LSTADD(jadr)) - s
  fim para jadr
  nadr ← nadrf + 1
  kbeg ← kend + 1
  se kend ≠ kf vá para α
  DI(i) ← 1 / piv
fim para i
termine procedimento

```

(continua na próxima página)

(continuação da página anterior)

- 0:        **vá para  $\beta$**
- 1:         $s \leftarrow s + um_1 * UN(p_1)$   
 $p_1 \leftarrow (p_1 + 1)$   
**vá para  $\beta$**
- 2:         $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
**vá para  $\beta$**
- 3:         $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
**vá para  $\beta$**
- 4:         $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $\quad + um_4 * UN(p_4)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 $p_4 \leftarrow (p_4 + 1)$   
**vá para  $\beta$**
- ...
- 7:         $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $\quad + um_4 * UN(p_4) + um_5 * UN(p_5) + um_6 * UN(p_6)$   
 $\quad + um_7 * UN(p_7)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 $p_4 \leftarrow (p_4 + 1)$   
 $p_5 \leftarrow (p_5 + 1)$   
 $p_6 \leftarrow (p_6 + 1)$   
 $p_7 \leftarrow (p_7 + 1)$   
**vá para  $\beta$**

Caso se adote a estratégia original apresentada no procedimento IV.1(3), estará se incorrendo em um *overhead* em termos de tempos de execução.

Caso se opte pela nova estratégia apresentada, se estará incorrendo em uma penalidade em espaço final de armazenamento

da ordem de  $\text{Nonz}(U) * \# \text{maximo de famílias}$ .

Percebe-se que no procedimento apresentado, as únicas diferenças com relação ao anterior, se dão na forma de codificação dos produtos escalares, e na inclusão do tratamento baseado em listas simbólicas de endereços, para se efetuar apenas os acessos na linha base correspondentes a operações de acumulação a cada varredura sendo considerada.

Este novo procedimento permite na verdade, uma redução no número de acessos na linha base, maior do que se poderia a princípio imaginar.

Tal fato se dá porque na hipótese de múltiplas varreduras, correspondendo cada uma as operações de acúmulo relativas a famílias distintas de linhas contribuintes, a informação simbólica por listas de endereços para o processamento de cada família, poder vir a ser diferente de família para família.

Ou seja, no caso de famílias com padrão estrutural completamente distinto entre si, e que contenham apenas uma pequena porção de elementos não nulos, comparados com o número total de elementos da linha base, a abordagem por listas de endereços (para os acessos relativos a cada família) é a mais "ajustada", pois se efetuará as operações de decodificação apenas para as colunas realmente necessárias ao processamento das contribuições de cada família em particular.

Desta forma, percebe-se que um leque de estratégias de implementação é possível, tendo como eixos fundamentais, os requisitos computacionais básicos como tempo de CPU e memória.

● *A principal vantagem dos procedimentos simbólicos via codificação ancestral (ou binária) apresentados neste capítulo, sobre estratégias no estilo de Gustavson, como as apresentadas na seção III.1, é que o tamanho final de "código" (a nível de programa implementado em alguma linguagem) é constante e único para qualquer classe de matrizes.*

Assim, percebe-se que as alternativas apresentadas neste capítulo, oferecem além de um desempenho muito próximo ao das

estratégias simbólicas realmente *loop free*, a vantagem de se precisar compilar a rotina de eliminação esparsa uma única vez, sem ter que gerá-la ou ajustá-la para cada nova matriz estruturalmente distinta em que se venha a requerer a solução de um sistema linear.

Ou seja, embora os códigos de máquina ou em qualquer linguagem de alto nível, que implementem os procedimentos IV.1(3) e IV.3(2), venham a ser ainda um tanto "indigestos", os mesmos só precisam ser compilados uma única vez, de sorte a poderem ser incorporados a alguma biblioteca de rotinas de resolução esparsa.

A "adaptação" as características estruturais de cada matriz, se dá mediante o uso das listas de "códigos", e cujo processo de decodificação pode ser implementado de forma bastante eficiente como se mostrou nos procedimentos baseados tanto na codificação binária, quanto na ancestral.

Uma pequena "variante" possível num esquema de codificação por envelopes, é apresentada a seguir, e onde mais uma vez se faz uma opção de balanço entre eficiência a nível de tempos de execução ou outro recurso computacional, como no caso particular a extensão do trecho de código necessário para a implementação das operações de acúmulo por produto escalar.

A chave para uma redução, advém do fato de que para um dado índice (correspondente a um código), os trechos de códigos (programa) que precisam ser executados para cada valor do índice associado, serem na verdade "recorrentes" pois a implementação do processo de acumulação para o maior valor possível para o índice associado, na verdade contém todos os demais casos para valores inferiores do índice de codificação.

Recordando-se das técnicas apresentadas na seção III.1, o que se precisa apenas é lançar mão de "entry points" para cada porção associada da implementação do somatório para o número máximo de parcelas.

Desta forma, se apresenta a forma alternativa para a acumulação de produtos escalares, em estratégias baseadas no reordenamento ancestral (como a IV.3(2)), e onde codificações por "envelope" se mostram mais do que adequadas.

**Codificação IV.3(a) Alternativa para acúmulo (por envelopes)**

```

7:      s ← s + um7 * UN(p7)
        p7 ← (p7 + 1)
6:      s ← s + um6 * UN(p6)
        p6 ← (p6 + 1)
5:      s ← s + um5 * UN(p5)
        p5 ← (p5 + 1)
4:      s ← s + um4 * UN(p4)
        p4 ← (p4 + 1)
3:      s ← s + um3 * UN(p3)
        p3 ← (p3 + 1)
2:      s ← s + um2 * UN(p2)
        p2 ← (p2 + 1)
1:      s ← s + um1 * UN(p1)
        p1 ← (p1 + 1)
0:      retorne ao ponto de chamada

```

A única ineficiência da codificação acima é que determinados compiladores não conseguem gerar código tão otimizado quanto no procedimento IV.3(2), simplesmente por acabarem explicitamente armazenando as parcelas intermediárias de volta a posição de memória alocada para a variável  $s$ , enquanto que tais valores parciais de  $s$  na verdade poderiam ser mantidos em um registrador físico da máquina onde o código viesse a ser executado.

Em linguagem de máquina ou Assembler, tais inconveniências podem ser facilmente contornadas, e o único problema com relação a geração de código otimizado para uma implementação de alto nível da codificação apresentada acima, é que a maioria dos compiladores, não consegue ser hábil o bastante, para reconhecer que os valores parciais de  $s$  jamais precisarão ser armazenados temporariamente ou de volta, a posição de memória correspondente a variável em questão.

Assim, a codificação IV.3(a) fica como um diretriz passível de exploração, em implementações de mais baixo nível das operações básicas de acúmulo de produtos escalares.

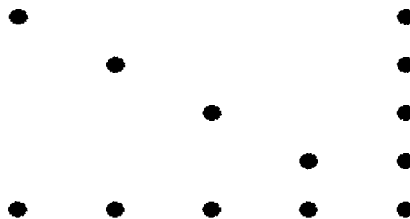
Para finalizar esta seção, voltaremos a um ponto de certa forma intrigante com relação ao desempenho das

estratégias propostas, em face a um possível pior caso, capaz de por si só, demolir todos os esforços dispendidos nesta seção, com o intuito de aprimorar um método de eliminação tipo Crout, baseado em uma codificação ancestral.

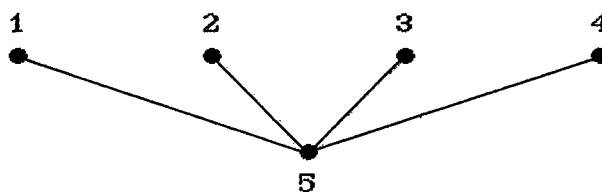
O "pior caso", excetuando-se o de famílias com um número por demais extensivo de componentes, (e com um número significativo destes contribuindo sobre a linha base a ser eliminada), é justamente o oposto, ou seja um número extensivo de famílias (na maior parte das vezes com poucos componentes, e um número pequeno de descendentes em cada uma contribuindo sobre a linha base).

Neste caso, o número de varreduras, sem dúvida se mostraria intolerável, caso tivéssemos que efetuar da ordem  $O(n)$  varreduras pelas colunas da linha base, correspondendo cada uma a uma família proveniente de um ramo distinto da árvore de eliminação.

O mais curioso, tomando-se como exemplo um pior caso para uma matriz de dimensão 5 como a apresentada na fig\_IV(9) (e que já havia sido considerada na seção II.2 em particular), é que o que se esperaria ser um caso "intratável" para uma codificação por envelopes, mostra-se na verdade um dos mais "amenos" possíveis.



fig\_IV(9) - Exemplo de um pior caso



fig\_IV(10) - Árvore de eliminação associada

Neste caso, percebe-se que na quinta etapa do processo de eliminação, uma codificação da forma puramente "ancestral", acabaria resultando em 4 códigos distintos para se efetuar as contribuições sobre o elemento  $a_{5,5}$ , com cada um dos códigos correspondendo a cada uma das contribuições associadas as linhas de 1 a 4, pertencentes a famílias distintas na árvore de eliminação.

Como já se comentou anteriormente nesta seção, existe muita flexibilidade na forma de se codificar operações de contribuição via a técnica por "envelopes", onde se precisa especificar apenas a posição (na representação reduzida) da primeira linha a contribuir na coluna sendo considerada para efeito do acúmulo do somatório, pois a partir desta informação, sabe-se por construção (e definição do termo "envelope"), que todas as demais linhas consideradas "dentro deste envelope" (poderia-se dizer), possuem o mesmo padrão estrutural não nulo.

A codificação mediante o reordenamento "ancestral" é uma das alternativas possíveis, e que por construção garante a exploração deste tipo de padrão.

Percebe-se na verdade agora, que outras estratégias de reagrupamento ou codificação também são possíveis, e o que se mostra de certa forma tão intrigante como promissor, é que tomando-se o exemplo considerado em particular, percebe-se que por justamente possuir o maior número de contribuições em ramos distintos (e independentes) da árvore, o padrão estrutural da quinta coluna acabou por atender plenamente aos requisitos para uma compactação por envelopes. Ou seja, com todas as contribuições de ramos distintos da árvore, vindo a ocorrer em uma mesma coluna, e "contíguamente" para todas as linhas envolvidas.

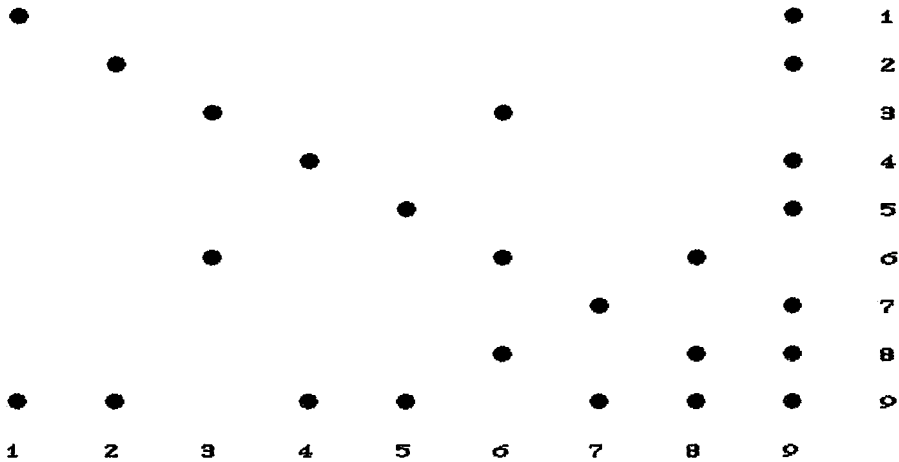
O que se percebe portanto, é que neste caso em particular, a codificação necessária, seria apenas de um único código, contendo o valor 1, e que espelharia na verdade uma característica "agregada" de contribuições provenientes de ramos distintos da árvore.

O que a codificação "ancestral" faz, é produzir apenas codificações "isoladas" para cada família, sem se levar em conta possíveis padrões estruturais comum das demais.

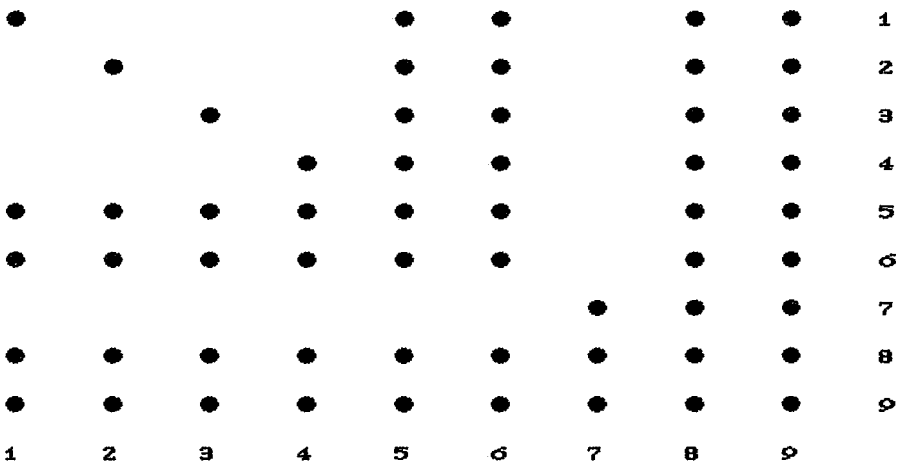
O primeiro exemplo apresentado, serve para ilustrar



alguns outros pontos, e para tanto, consideremos duas outras "variantes" de piores casos, para matrizes com dimensão um pouco mais elevada e apresentadas nas figuras abaixo.



fig\_IV(11) - Segundo exemplo



fig\_IV(12) - Terceiro exemplo

Na matriz do segundo exemplo, percebe-se claramente que a estratégia de codificação por envelopes não pode ser aplicada de uma só vez, para todas as contribuições na nona coluna.

Porém conjuntos contíguos de linhas, correspondentes a ramos distintos da árvore, ainda podem ser "amalgamados" em uma única codificação, como no caso das linhas ( 1, 2 ), ( 4, 5 ), e ( 7, 8 ).

O que se percebe é que alguma forma de análise da sequência de contribuições a nível das colunas presentes em

cada linha base da representação de  $U^T$  pode vir a permitir uma compactação melhor do que a de reordenamento "ancestral" aplicada indiscriminadamente.

A estratégia de "construção" do vetor JUTORD como apresentada nesta seção, permite que tais seqüencialidades a nível de colunas possam vir a ser analisadas e exploradas, bastando simplesmente antes de se considerar uma nova família para um novo nó a ser acrescentado, se verificar se o padrão estrutural da linha associada, se assemelha ao de alguma outra família já estabelecida, ou em particular ao da família anteriormente definida (caso em que colunas seqüencialmente contíguas na representação de  $U^T$  acabariam sendo enquadradas por exemplo).

A matriz do terceiro exemplo, assemelha-se muito a uma forma tipicamente encontrada em matrizes da área de elementos finitos e cálculo estrutural em geral, onde normalmente se aplica um reordenamento visando a redução de *fill-in's*, mediante heurísticas como as de *Nested Dissection*, e não o método de ordenamento segundo o "menor grau" (como apresentado na seção II.2).

O que se percebe neste exemplo é exatamente o que se observou no primeiro exemplo, ou seja, que embora o número de famílias envolvidas na eliminação de uma linha base como a 6 ou a 8, seja elevado, a estrutura das contribuições de linhas provenientes de ramos distintos da árvore pode vir a se assemelhar bastante (sendo idêntica no caso particular em questão).

● *Este fato mostra que uma outra possibilidade de melhoria do esquema de codificação é possível, tomando-se como base uma variável até então ainda não considerada: A estratégia de reordenamento ótimo visando a redução de fill-in's aplicada originalmente sobre a matriz, nas quais um possível critério de desempate poderia ser justamente um ordenamento que resultasse num padrão típico como o da figura IV(12), ou qualquer outro que vier a permitir uma maior compactação das informações na forma de envelope necessárias para se caracterizar o processo de eliminação como um todo.*

Tais esquemas não serão considerados neste trabalho, mas

percebe-se que o potencial de extensão e aplicação das diretrizes básicas aqui apresentadas se mostra bastante promissor.

Ao final desta seção, percebe-se que "muito" ainda pode ser feito na área de resolução de sistemas esparsos como um todo, e em particular na área de compactação de informação e procedimentos simbólicos, mesmo para a já "esquecida" e restrita classe de arquiteturas escalares de computadores atuais, e que na opinião dos que lideram as pesquisas em esparsidade hoje em dia, parecem fadadas ao total esquecimento, como meras peças de decoração nos futuros museus.

Na verdade mostrou-se que esta "antiguidade" não (é ainda tão pré-histórica assim), e nem anula o status ou o caráter no mínimo instigante a que a área de esparsidade como um todo vem impondo aos seus pesquisadores ao longo de todos estes anos, e nas mais variadas gamas de arquiteturas.

#### IV.4 Formulação completa do método

Nesta seção final, se procurará reunir todos os conceitos apresentados neste capítulo, em uma abordagem unificada, no estilo da que se apresentou na seção III.4, considerando-se naquele caso, uma estratégia híbrida baseada em janelas de processamento, e que continua válida e aplicável os métodos de eliminação do tipo Crout simétricos, propostos nesta seção.

O que se precisa inicialmente unificar, são as formas de codificação a serem adotadas, e em que circunstâncias aplica-las em detrimento das demais.

Pelo que se pode observar na seção anterior, quando se apresentou o procedimento IV.3(2), a estrutura básica do algoritmo de decodificação e processamento das contribuições, permaneceu praticamente a mesma, se comparada com a do algoritmo IV.1(3), baseado num esquema de codificação binária, só diferindo a parte puramente "combinacional" da codificação, onde a implementação das operações de acúmulo por produto escalar, certamente é distinta da empregada para se processar códigos na forma de "envelope" adotados nos esquemas baseados no reordenamento ancestral.

O que se propõe como estratégia de solução, é se considerar procedimentos "híbridos" em que ambas as formas de codificação possam vir a ser exploradas, assim como uma terceira forma de processamento tipicamente encontrável, e merecedora de atenção dedicada por si só, como no caso de estruturas da forma *supernodal*.

O que fica faltando portanto é o estabelecimento de critérios, ou "pontos de corte", entre cada uma das 3 estratégias consideradas.

Percebe-se que intuitivamente, alguns critérios podem de antemão ser estabelecidos, em face do que já foi exposto ao longo do texto.

● Um critério simples porém eficaz, é simplesmente se considerar um esquema de codificação binária, para o processamento do acúmulo de contribuições numa dada etapa básica do processo, caso o número de linhas contribuintes seja inferior ao valor  $n_{\max}$  (correspondendo ao número de bits da codificação utilizada).

Este é um caso típico de se encontrar no processamento de matrizes extremamente esparsas, como algumas matrizes de potência, e que contenham por exemplo menos do que 8 elementos não nulos por cada linha de  $U^T$  (assumindo-se uma codificação de 1 byte por código).

Para estes casos uma codificação binária é mais do que suficiente, e um reordenamento ancestral mostraria-se completamente desnecessário, por simplesmente se oferecer o risco de se introduzir novas varreduras desnecessárias na linha base, além do tempo de pré-processamento inicial perdido com o reordenamento (que de certo modo ainda poderia ser tolerado, por ser efetuado apenas uma única vez).

Assim, como nestes casos, uma codificação alternativa por outro esquema não se mostra mais vantajosa sob o ponto de vista de compactação de informações, o uso de uma codificação binária, é mais do que apropriado.

Em casos de extrema esparsidade como os considerados, para a aplicação incondicional da codificação binária, nota-se que apenas uma varredura na linha base se mostra necessária, e o método como um todo, se apresenta como uma alternativa bem mais eficaz do que os métodos convencionais apresentados na seção II.4.

● Um segundo caso a se considerar de forma tão incondicional quanto os de extrema esparsidade, é justamente quando se detecta a presença do oposto, ou seja, *supernodes*, e cujo padrão estrutural viabiliza a utilização de esquemas dedicados e ainda mais compactos do que uma representação por envelopes, pois basta se especificar a linha inicial a ser considerada (na forma "reduzida"), que deste ponto em diante, toda a seqüência de acúmulos para todas as colunas constituintes da linha base podem ser efetuada, pois o padrão estrutural de todas as colunas é exatamente o mesmo, (e simplesmente denso).

Deste modo, no caso de linhas base que venham a conter um número maior do que  $n_{\max}$  linhas contribuintes, o primeiro passo a se fazer, é se detectar todos os conjuntos distintos de *supernodes* presentes, e excluí-los da forma seguinte de processamento a ser considerada, tratando cada um dos *supernodes* separadamente de forma dedicada.

● A terceira estratégia, via reordenamento ancestral, passa a ser considerada apenas nos casos em que mediante a combinação dos 2 esquemas anteriores, ainda restarem linhas contribuintes a se considerar.

Ou seja mesmo no caso de linhas em que originalmente o número de elementos não nulos na estrutura de  $U^T$  é superior a  $n_{\max}$ , pode ocorrer, de não se precisar lançar mão de um esquema de codificação ancestral, bastando que nestes casos, o número de linhas contribuintes remanescentes, que não pertençam a nenhum *supernode*, seja inferior a  $n_{\max}$ , onde um esquema binário voltaria a se mostrar atrativo.

Percebe-se portanto, que o esquema de codificação binária, pode ser utilizado para se tratar de casos excedentes, quando o número de elementos for inferior ao máximo permitido por esta codificação.

É justamente nos casos em que estratégias de reordenamento ancestral se mostram "inevitáveis", que se pode pensar em adotar um esquema de codificação binária como um esquema "complementar", no sentido de por exemplo se processar as "sobras" de diversas famílias com um número demasiadamente extenso de elementos (superior a  $2^{n_{\max}}$ ) que venham a se apresentar em algum caso mais "espúrio".

Na prática com o tratamento dos nós de característica *supernodal* de forma dedicada e isolada dos restantes, o número de linhas contribuintes que normalmente necessitam ser codificadas por algum dos outros 2 critérios, tende a ser bastante reduzido, e a hipótese de diversas famílias conterem ainda assim um número expressivo de linhas contribuintes mostra-se realmente como espúria.

Diversas estratégias complementares, na tentativa de se compactar ainda mais as informações do tipo envelope que possam ser encontradas, como as que foram apresentadas no final da seção anterior, podem ser empregadas como auxiliares, num esquema de reordenamento ancestral.

Particularmente se deve notar a observação de que um dos piores padrões típicos de se encontrar, como a dispersão das famílias por um número elevado de ramos distintos, na verdade tende a fornecer padrões de similaridade tal que uma codificação por envelopes possa ainda assim ser adotada,

simplesmente porque a dispersão por muitos ramos distintos, tende a forçar a um "achatamento" da largura da faixa de colunas contribuintes como se pode observar num exemplo como o da fig\_IV(12).

No tocante a codificação por envelopes, a questão da melhor estratégia ainda permanece em aberto no momento, com um variado leque de possibilidades a serem exploradas.

O reordenamento ancestral é sómente a técnica mais conservadora e segura de se tratar a questão.

O que fica faltando para se apresentar um novo procedimento unificado é portanto de algum modo se "incorporar" ao mesmo, as regras e critérios de escolha apresentadas informalmente acima.

Na verdade o que se considerou até o presente momento, foram estratégias "complementares" a serem utilizadas no tratamento das diversas contribuições a serem efetuadas em cada etapa básica.

Ou seja, o que se precisa especificar para um processamento eficaz o mais ajustado possível, é um "particionamento" da estrutura de  $U^T$ , similar a utilizada para o reordenamento ancestral, só que desta vez, dividindo-se o conjunto de linhas contribuintes em 3 grupos distintos, a serem processados via codificação binária, ancestral ou *supernodal*.

De posse dos índices das linhas destinadas a cada uma destas formas de processamento, se pode lançar mão de um esquema muito similar ao dos procedimentos IV.3(2) ou IV.1(3) por exemplo, com a única diferença de agora se incorporar ambas as estratégias de acúmulo por produtos escalares, num só procedimento, e com rótulos correspondentes as porções de código associadas, obviamente distintos entre si.

Ou seja, não é preciso se perder nenhuma quantidade de informação por palavra de codificação para se poder diferenciar os códigos, pois na verdade um tratamento em porções dedicadas à cada uma das estratégias estará sendo efetuado.

O que isso quer dizer, é que os pontos de "retorno" nos acúmulos dos somatórios das estratégias IV.1(3) e IV.3(2), continuarão a ser os mesmos, ou seja, para uma porção correspondente no grupo de instruções dedicadas a cada uma

das duas estratégias consideradas.

Assim, o novo procedimento a ser apresentado, deve apresentar ao menos uma porção do *loop* mais interno "replicada" de forma a tratar os casos por codificação binária, por envelopes e por *supernodes*, e para onde cada porção associada de acúmulo das contribuições deve retornar após o processamento de cada código.

Se não se adotasse este esquema de separação inicial por categoria de processamento, e não se replicasse certos trechos do programa, acabaria necessário introduzir novos códigos adicionais (um por cada linha contribuinte a cada etapa básica), para se poder determinar a qual grupo de processamento em particular a mesma pertence.

Percebe-se que com esta forma de abordagem, conforme se comentou no início desta seção, se estará bem próximo de estratégias híbridas como as por "janelas de processamento", consideradas no terceiro capítulo, e cujo objetivo final é uma adaptação a mais ajustada possível, aos padrões característicos de cada matriz.

*No caso das estratégias consideradas nesta seção, se tem na verdade o que se poderia chamar por "categorias de codificação" no lugar de "janelas de processamento".*

A diferença entre as abordagens, se dá ao nível da granularidade em que se considera o reconhecimento de características: Uma granularidade elevada no caso do tratamento por janelas, reconhecendo-se os padrões básicos mais proeminentes, e uma granularidade intermediária, no caso das abordagens por "categorias de codificação" apresentadas nesta seção.

Um nível ainda mais baixo de granularidade é por exemplo o empregado nas estratégias simbólicas ao estilo de Gustavson, apresentadas em III.1.

O que se percebe é que diferentes níveis de abordagens e granularidades podem ser considerados caso se deseje levar sempre avante a meta proposta neste trabalho, qual seja, a de se obter estratégias o mais eficiente possíveis, mantendo uma penalidade adicional de espaço dentro de limites toleráveis, sem se lançar mão de estratégias como a originalmente proposta por Gustavson, e que só se mostrou



inviável em face ao seu exorbitante requisito em termos de espaço de código adicional.

O que na verdade não se cogitou até o momento, foi a utilização "conjunta" de abordagens por "janelas de processamento" e "categorias de codificação", explorando o que ambas apresentam de melhor, ou seja, a flexibilidade de adaptação e escolha dentre várias alternativas possíveis.

Um esquema de utilização conjunta de algumas destas técnicas é bem simples e intuitivo, justificando portanto a sua apresentação.

Simplesmente nota-se que certas fases do processo de eliminação, com particular atenção o tratamento *supernodal* e o completamente denso (que na verdade pode ser visto e implementado como um mero caso particular de processamento *supernodal*), podem e devem ser considerados de forma isolada, caso se esteja pensando em elevar a eficiência final dos códigos de eliminação.

A estratégia mais intuitiva a se seguir, e que reúne praticamente quase todas as formas de processamento propostas e apresentadas, é simplesmente se considerar apenas 4 classes de janelas numa primeira análise das características de cada matriz a se processar, considerando-se apenas os casos completamente diagonais, os por geração dos fatores por linhas, os por processamento *supernodal* e o caso completamente denso.

Estas 4 classes, englobam todas as demais consideradas na seção III.4, e desta forma se mostram as mais adequadas a se tomar como "eixos" coordenados.

As classes diagonal, *supernodal*, e completamente densa, podem ser imediatamente reconhecidas, e tratadas de forma dedicada pelos métodos apresentados ao longo deste trabalho, o que deixa todas as demais porções restantes para serem abordadas via os esquemas tipo Crout simétrico propostos neste capítulo.

O que se passa a considerar num nível seguinte de granularidade, são as 3 "categorias de codificação" (como se denominou nesta seção).

O que o leitor com justificada razão pode se perguntar, é porque incluir um tratamento *supernodal* tanto no nível de mais alta granularidade, como num nível intermediário.

A explicação advém de um detalhe até então ainda não comentado, de que na verdade existem duas formas de processamento *supernodal* passíveis de exploração.

A explorada num nível de granularidade mais alta, diz respeito apenas a combinação escalar das linhas correspondentes a um mesmo *supernode*, de modo a se cancelar os seus elementos correspondentes na porção triangular inferior da matriz de fatores.

Para melhor compreensão deste fato, tomemos um caso particular de *supernode* como a porção completamente densa ao final de uma matriz como a da fig\_III(3) por exemplo.

O que a abordagem por "janelas de processamento" considera, é que a porção densa dos fatores será na verdade gerada e processada em duas etapas. A primeira delas, correspondendo a subtração de todas as linhas anteriores nas janelas 3 e 4 da porção densa, e que corresponderia a eliminação dos elementos da faixa destacada à esquerda desta porção. Nesta fase o método empregado continua a ser o de geração por linhas, só que gerando-se de forma apenas parcial os elementos correspondentes de cada linha na porção densa. Na etapa seguinte é que se processa uma eliminação por "sub-matrizes", e esta sim, de forma completamente *densa*, pois só leva em conta o cancelamento dos elementos a esquerda da diagonal contidos na região densa. Ou seja na segunda etapa, as contribuições podem ser processadas de forma *supernodal* ou completamente densa em particular, pois correspondem apenas as contribuições de linhas cujo padrão estrutural é o mesmo.

Este processamento relatado é o que se aplica num nível mais elevado de granularidade, como numa abordagem por janelas.

O processamento da forma *supernodal* presente numa granularidade intermediária como a das "categorias de codificação", se dá por uma razão bem distinta, pois na verdade assumindo-se que uma abordagem por janelas inicialmente tenha sido adotada, o processamento restrito das linhas de cada *supernode* acabaria sendo efetuado de forma

dedicada já nesta fase, e o que realmente sobra para se considerar na granularidade a seguir, é o caso das contribuições de *supernodes* em linhas base que não pertençam a *supernode* algum.

Nestes casos, a presença de *supernodes* dentro do conjunto de linhas contribuintes sobre uma dada linha base, pode ser explorada a um nível menos eficiente do que na granularidade mais alta (quando todas as linhas de um mesmo *supernode* são processadas em várias etapas básicas consecutivas e de uma mesma forma).

O que se pode fazer quando se detecta a presença de *supernodes* na estrutura de linhas contribuintes, é simplesmente agrupar os seus componentes e processar as atualizações relativas a cada uma das colunas da linha base, utilizando-se para tal, um mesmo e único código, no lugar de um código por coluna que seria indispensável nas demais formas de codificação.

A eficiência que se ganha portanto é em compactação de informação, e não em aumento do desempenho a nível de menores tempos de execução (como por exemplo quando se trata todo o conjunto de operações relativas apenas as linhas de um mesmo *supernode*).

Percebe-se que numa abordagem por *supernodes* numa granularidade à nível intermediário, pode-se lançar mão para o processo de acúmulo das contribuições, da mesma estratégia empregada para o tratamento por "envelopes", pois a única diferença com relação a esta abordagem, é que no caso por envelopes mais geral, os índices reduzidos iniciais, associados a cada linha, podem ser distintos, enquanto que na presença de *supernodes*, tais índices se mantêm constantes por todas as colunas da linha base envolvidas.

De posse de todos as diretivas e comentários apresentados, passa-se portanto a especificação de um procedimento híbrido por "categorias de codificação", e que pode ser adotado como o esquema preferencial de geração dos fatores por linhas, durante a etapa de divisão do problema por janelas de processamento.

**Procedimento IV.4(1)    Eliminação via Categorias de Codificação**

ncod  $\leftarrow$  0  
 nk  $\leftarrow$  0  
 nkf  $\leftarrow$  0  
 nadr  $\leftarrow$  1  
 njadr  $\leftarrow$  0

**para** i de 1 até n **faça**

IUP(i)  $\leftarrow$  IU(i)  
 piv  $\leftarrow$  DI(i)  
 kbeg  $\leftarrow$  IUT(i)  
 nkf  $\leftarrow$  nkf + 1  
 kf  $\leftarrow$  LSTKF(nkf)

**caso** CATCOD(i) **seja**

- 1: **execute**  
     CODBIN;
- 2: **execute**  
     CODENV;
- 3: **execute**  
     CODBIN, nkf  $\leftarrow$  nkf + 1, kf  $\leftarrow$  LSTKF(nkf),  
     CODENV;
- 4: **execute**  
     CODSUP;
- 5: **execute**  
     CODBIN, nkf  $\leftarrow$  nkf + 1, kf  $\leftarrow$  LSTKF(nkf),  
     CODSUP;
- 6: **execute**  
     CODENV, nkf  $\leftarrow$  nkf + 1, kf  $\leftarrow$  LSTKF(nkf),  
     CODSUP;
- 7: **execute**  
     CODBIN, nkf  $\leftarrow$  nkf + 1, kf  $\leftarrow$  LSTKF(nkf),  
     CODENV, nkf  $\leftarrow$  nkf + 1, kf  $\leftarrow$  LSTKF(nkf),  
     CODSUP;

**fim caso** CATCOD(i)

DI(i)  $\leftarrow$  1 / piv

**fim para** i

**termine procedimento principal**

(continua na próxima página)

(continuação da página anterior)

Processamento via Codificação Binária**CODBIN:**

```

 $\alpha$ :      nk  $\leftarrow$  nk + 1
           kend  $\leftarrow$  LSTKEND(nk)
           r  $\leftarrow$  0
           para k de kbeg até kend faça
               r  $\leftarrow$  r + 1
               l  $\leftarrow$  JUT(k)
                $p_r \leftarrow$  IUP(l)
               IUP(l)  $\leftarrow$  IUP(l) + 1
                $um_r \leftarrow$  UN( $p_r$ ) * DI(l)
               piv  $\leftarrow$  piv - UN( $p_r$ ) *  $um_r$ 
               UN( $p_r$ )  $\leftarrow$   $um_r$ 
                $p_r \leftarrow$  ( $p_r$  + 1)
           fim para k
           njadr  $\leftarrow$  njadr + 1
           nadrf  $\leftarrow$  LSTJADR(njadr)
           para jadr de nadr até nadrf faça
               s  $\leftarrow$  0
               ncod  $\leftarrow$  ncod + 1
               vá para ( $B_0, B_1, B_2, B_3, B_4, B_5, B_6, B_7$ )
                   em função de LSTCOD(ncod)
 $\beta$ :      UNCLSTADD(jadr))  $\leftarrow$  UNCLSTADD(jadr)) - s
           fim para jadr
           nadr  $\leftarrow$  nadrf + 1
           kbeg  $\leftarrow$  kend + 1
           se kend  $\neq$  kf vá para  $\alpha$ 

           retorne ao ponto de chamada

```

(continua na próxima página)

(continuação da página anterior)

Processamento via Codificação por Envelopes**CODENV:**

```

 $\alpha'$ :      nk  $\leftarrow$  nk + 1
              kend  $\leftarrow$  LSTKEND(nk)
              r  $\leftarrow$  kend - kbeg + 2
              para k de kbeg até kend faça
                  r  $\leftarrow$  r - 1
                  l  $\leftarrow$  JUT(k)
                   $p_r \leftarrow$  IUP(l)
                  IUP(l)  $\leftarrow$  IUP(l) + 1
                   $um_r \leftarrow$  UN( $p_r$ ) * DI(l)
                  piv  $\leftarrow$  piv - UN( $p_r$ ) *  $um_r$ 
                  UN( $p_r$ )  $\leftarrow$   $um_r$ 
                   $p_r \leftarrow$  ( $p_r$  + 1)
              fim para k
              njadr  $\leftarrow$  njadr + 1
              nadrf  $\leftarrow$  LSTJADR(njadr)
              para jadr de nadr até nadrf faça
                  s  $\leftarrow$  0
                  ncod  $\leftarrow$  ncod + 1
                  vá para ( $E_0, E_1, E_2, E_3, E_4, E_5, E_6, E_7$ )
                      em função de LSTCOD(ncod)
 $\beta'$ :      UNCLSTADD(jadr))  $\leftarrow$  UNCLSTADD(jadr)) - s
              fim para jadr
              nadr  $\leftarrow$  nadrf + 1
              kbeg  $\leftarrow$  kend + 1
              se kend  $\neq$  kf vá para  $\alpha'$ 

              retorne ao ponto de chamada

```

(continua na próxima página)

(continuação da página anterior)

Processamento via Codificação Supernodal

**CODSUP:**

```

 $\alpha''$ :      nk  $\leftarrow$  nk + 1
              kend  $\leftarrow$  LSTKEND(nk)
              r  $\leftarrow$  kend - kbeg + 2
              para k de kbeg até kend faça
                  r  $\leftarrow$  r - 1
                  l  $\leftarrow$  JUT(k)
                   $p_r \leftarrow$  IUP(l)
                  IUP(l)  $\leftarrow$  IUP(l) + 1
                   $um_r \leftarrow$  UN( $p_r$ ) * DI(l)
                  piv  $\leftarrow$  piv - UN( $p_r$ ) *  $um_r$ 
                  UN( $p_r$ )  $\leftarrow$   $um_r$ 
                   $p_r \leftarrow$  ( $p_r$  + 1)
              fim para k
              ncod  $\leftarrow$  ncod + 1
              icod  $\leftarrow$  LSTCOD(ncod)
              para j de IUC(i) até IUF(i) faça
                  s  $\leftarrow$  0
                  vá para ( $S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7$ )
                      em função de icod
 $\beta''$ :      UN(j)  $\leftarrow$  UN(j) - s
              fim para j
              kbeg  $\leftarrow$  kend + 1
              se kend  $\neq$  kf vá para  $\alpha''$ 

              retorne ao ponto de chamada

```

(continua na próxima página)

(continuação da página anterior)

Acúmulo por decodificação Binária $B_0$ : vá para  $\beta$  $B_1$ :  $s \leftarrow s + um_1 * UN(p_1)$  $p_1 \leftarrow (p_1 + 1)$ vá para  $\beta$  $B_2$ :  $s \leftarrow s + um_2 * UN(p_2)$  $p_2 \leftarrow (p_2 + 1)$ vá para  $\beta$  $B_3$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2)$  $p_1 \leftarrow (p_1 + 1)$  $p_2 \leftarrow (p_2 + 1)$ vá para  $\beta$  $B_4$ :  $s \leftarrow s + um_3 * UN(p_3)$  $p_3 \leftarrow (p_3 + 1)$ vá para  $\beta$  $B_5$ :  $s \leftarrow s + um_1 * UN(p_1) + um_3 * UN(p_3)$  $p_1 \leftarrow (p_1 + 1)$  $p_3 \leftarrow (p_3 + 1)$ vá para  $\beta$  $B_6$ :  $s \leftarrow s + um_2 * UN(p_2) + um_3 * UN(p_3)$  $p_2 \leftarrow (p_2 + 1)$  $p_3 \leftarrow (p_3 + 1)$ vá para  $\beta$  $B_7$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$  $p_1 \leftarrow (p_1 + 1)$  $p_2 \leftarrow (p_2 + 1)$  $p_3 \leftarrow (p_3 + 1)$ vá para  $\beta$ 

(continua na próxima página)



(continuação da página anterior)

Acúmulo por decodificação via Envelopes

- $E_0$ : vá para  $\beta'$
- $E_1$ :  $s \leftarrow s + um_1 * UN(p_1)$   
 $p_1 \leftarrow (p_1 + 1)$   
vá para  $\beta'$
- $E_2$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
vá para  $\beta'$
- $E_3$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
vá para  $\beta'$
- $E_4$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $+ um_4 * UN(p_4)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 $p_4 \leftarrow (p_4 + 1)$   
vá para  $\beta'$
- ...
- $E_7$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $+ um_4 * UN(p_4) + um_5 * UN(p_5) + um_6 * UN(p_6)$   
 $+ um_7 * UN(p_7)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 $p_4 \leftarrow (p_4 + 1)$   
 $p_5 \leftarrow (p_5 + 1)$   
 $p_6 \leftarrow (p_6 + 1)$   
 $p_7 \leftarrow (p_7 + 1)$   
vá para  $\beta'$

(continua na próxima página)

(continuação da página anterior)

Acúmulo por decodificação Supernodal

$S_0$ : vá para  $\beta''$   
 $S_1$ :  $s \leftarrow s + um_1 * UN(p_1)$   
 $p_1 \leftarrow (p_1 + 1)$   
vá para  $\beta''$   
 $S_2$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
vá para  $\beta''$   
 $S_3$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
vá para  $\beta''$   
 $S_4$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $\quad + um_4 * UN(p_4)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 $p_4 \leftarrow (p_4 + 1)$   
vá para  $\beta''$   
...  
 $S_7$ :  $s \leftarrow s + um_1 * UN(p_1) + um_2 * UN(p_2) + um_3 * UN(p_3)$   
 $\quad + um_4 * UN(p_4) + um_5 * UN(p_5) + um_6 * UN(p_6)$   
 $\quad + um_7 * UN(p_7)$   
 $p_1 \leftarrow (p_1 + 1)$   
 $p_2 \leftarrow (p_2 + 1)$   
 $p_3 \leftarrow (p_3 + 1)$   
 $p_4 \leftarrow (p_4 + 1)$   
 $p_5 \leftarrow (p_5 + 1)$   
 $p_6 \leftarrow (p_6 + 1)$   
 $p_7 \leftarrow (p_7 + 1)$   
vá para  $\beta''$

fim de todos os procedimentos

O procedimento listado nas páginas anteriores, se mostra aparentemente "complexo", porém cada uma de suas etapas básicas já foi devidamente analisada em separado, nas diversas seções deste trabalho.

Como se havia comentado, a "extensão" do código, deve-se a "replicação" de uma parcela considerável de trechos praticamente idênticos, a menos de alguns pequenos detalhes, comentados a seguir.

O primeiro deles, é de que lança-se mão de mais um vetor auxiliar LSTKF, e que indica para cada etapa básica  $i$ , as posições finais no vetor JUT, de cada uma das formas de codificação adotadas para geração da linha base correspondente.

Ou seja, assume-se que ao se iniciar o procedimento IV.4(1), o vetor JUT tenha sido ordenado não apenas de forma ancestral, mas também por categoria de processamento, com as posições iniciais correspondendo ao grupo de linhas contribuintes a serem processadas mediante uma codificação da forma binária, seguidas pelos grupos de famílias a serem processadas mediante uma codificação ancestral por envelopes, e finalmente seguido dos diversos grupos de linhas correspondentes a *supernodes*.

Deste modo, mediante um outro vetor auxiliar CATCOD, indicando as categorias de codificação a serem empregadas em cada linha, pode se processar cada grupo de linhas de forma dedicada, simplesmente varrendo-se o vetor JUT, por etapas, (delimitadas pelas posições kbeg e kend).

Com esta estratégia, o vetor CATCOD contém uma informação binária de 3 bits, e que pode ser facilmente decodificada, como nos mostra o procedimento principal, com o primeiro bit indicando a presença de linhas a serem processadas mediante codificação binária, o segundo bit indicando a presença de linhas a serem tratadas por envelope, e o último bit da codificação de CATCOD( $i$ ), indicando a presença de tratamento *supernodal* para alguma das linhas contribuintes da etapa básica  $i$ .

Os procedimentos via codificação e acúmulo na forma binária, bem como os baseados na codificação e acúmulo ancestral por envelopes não foram modificados em relação aos

respectivos procedimentos apresentados nas seções onde foram originalmente descritos.

A única exceção, como se havia comentado, é que os rótulos de desvio utilizados, agora necessitam ser explicitamente distintos dos utilizados na apresentação original de cada procedimento, pelo fato de duas classes distintas de códigos estarem presentes na implementação híbrida.

O único procedimento que passa a ser incorporado, é o do processamento *supernodal*, e que difere ligeiramente dos demais.

Uma das diferenças é que não se precisa utilizar uma lista simbólica de endereços para se determinar a posição correspondente na linha básica, uma vez que toda a sua estrutura de colunas contendo elementos não nulos acabará sendo operada.

Com isso, o acesso aos elementos da linha base, ficam ainda mais eficientes, pois troca-se o acesso indireto por listas simbólicas, por um acesso sequencial contíguo, na variável  $j$ , varrendo toda a extensão da linha base.

O outro fato a se comentar, é que não se precisa mais acessar o vetor LSTCOD para cada uma das colunas da linha base, (de modo a buscar a informação do índice relativo na representação reduzida, correspondendo a primeira linha contendo uma contribuição distinta de zero), visto que tal informação é a mesma e constante ao longo de todo o processo de acúmulo das contribuições via *supernodes*.

Assim essa informação é acessada apenas uma única vez, (fora do *loop* em  $j$ , correspondendo à varredura dos elementos da linha base corrente).

O que ficaria faltando se apresentar, seria todo o procedimento de geração da codificação por categorias, e cujo resultado seria o de fornecer justamente as estruturas auxiliares para o procedimento IV.4(1).

A extensão e o grau de detalhes de tal procedimento são tais, que levaria a uma extensão demasiada da exposição de técnicas que já foram apresentadas em seções isoladas deste

trabalho.

Desta forma, optou-se por finalizar esta seção, com alguns comentários gerais sobre a nova estratégia proposta, e a apresentação de alguns esquemas alternativos que podem vir a ser incorporados no futuro.

A primeira observação, é de que estratégias ainda mais elaboradas de codificação e implementação do acúmulo dos produtos escalares podem ser consideradas, especialmente na presença de características do tipo *supernodal*, como por exemplo processar-se 2 linhas base simultaneamente ao invés de uma única linha base por cada etapa.

Nestes casos, uma codificação a se considerar poderia ser da forma apresentada na codificação IV.4(4), em que se condidera o caso de duas acumulações simultâneas para cada coluna.

A grande vantagem de um esquema desta forma, está na melhor exploração de recursos como memória cache ou utilização de registradores (em arquiteturas com um número expressivo de registradores de ponto flutuante, como as do tipo RISC por exemplo).

Tal fato ocorre, pois apenas os fatores multiplicadores *um* são distintos no processamento de uma linha base para outra, com os acessos a posições de memória  $UN(p_l)$  sendo comuns.

Assim em máquinas dispondo de um número suficiente de registradores, os fatores multiplicativos *um* seriam armazenados em cada registrador, e os acessos as posições  $UN(p_l)$  só seriam efetuados uma única vez.

Esta exploração da "localidade" dos acessos a posições de memória sem dúvida é merecedora de atenção nas novas arquiteturas, visto quase todas sem exceção, contarem com mecanismos eficientes de aproveitamento das informações localmente acessadas em tempos de execução.

## Codificação IV.4(4)

Acumulação "múltipla"

$$7: \quad \begin{aligned} s1 &\leftarrow s1 + um_7^1 * UN(p_7) \\ s2 &\leftarrow s2 + um_7^2 * UN(p_7) \\ p_7 &\leftarrow (p_7 + 1) \end{aligned}$$

$$6: \quad \begin{aligned} s1 &\leftarrow s1 + um_6^1 * UN(p_6) \\ s2 &\leftarrow s2 + um_6^2 * UN(p_6) \\ p_6 &\leftarrow (p_6 + 1) \end{aligned}$$

$$5: \quad \begin{aligned} s1 &\leftarrow s1 + um_5^1 * UN(p_5) \\ s2 &\leftarrow s2 + um_5^2 * UN(p_5) \\ p_5 &\leftarrow (p_5 + 1) \end{aligned}$$

$$4: \quad \begin{aligned} s1 &\leftarrow s1 + um_4^1 * UN(p_4) \\ s2 &\leftarrow s2 + um_4^2 * UN(p_4) \\ p_4 &\leftarrow (p_4 + 1) \end{aligned}$$

$$3: \quad \begin{aligned} s1 &\leftarrow s1 + um_3^1 * UN(p_3) \\ s2 &\leftarrow s2 + um_3^2 * UN(p_3) \\ p_3 &\leftarrow (p_3 + 1) \end{aligned}$$

$$2: \quad \begin{aligned} s1 &\leftarrow s1 + um_2^1 * UN(p_2) \\ s2 &\leftarrow s2 + um_2^2 * UN(p_2) \\ p_2 &\leftarrow (p_2 + 1) \end{aligned}$$

$$1: \quad \begin{aligned} s1 &\leftarrow s1 + um_1^1 * UN(p_1) \\ s2 &\leftarrow s2 + um_1^2 * UN(p_1) \\ p_1 &\leftarrow (p_1 + 1) \end{aligned}$$

0: **retorne ao ponto de chamada**

Um outro comentário que se pode fazer com relação ao tipo básico de acesso efetuado pelas estratégias tipo Crout propostas neste capítulo, é que por se basearem no acúmulo de produtos escalares, além das vantagens de maior estabilidade numérica em função da possível utilização de precisão dupla, como já se havia comentado, o que se oferece é justamente uma melhor exploração do uso de registradores pois os acúmulos de parcelas intermediárias do somatório, não precisam ser armazenados explicitamente de volta a posição de memória correspondente a variável  $s$  (responsável pelo acúmulo),

podendo se efetuar todo o processo de somas com os resultados parciais armazenados apenas em registradores.

Esta é uma vantagem que só pode ser compartilhada com os métodos ao estilo Gustavson, onde a estratégia originalmente apresentada era justamente um método de Crout simbólico, justamente em função das vantagens supra mencionadas.

Nas demais estratégias convencionais, via o uso de vetores de trabalho expandido, incorre-se em *overheads* não apenas no processo de descompactação/compactação, mas fundamentalmente no nível do *loop* mais interno, pois os valores acumulados no vetor expandido  $W$  necessitam ser reescritos de volta a memória a cada varredura da linha base (correspondendo as operações de subtração de uma linha contribuinte anterior), simplesmente porque na forma de processamento das contribuições linha a linha, os valores temporariamente obtidos nas colunas de  $W$  não serem definitivos.

Numa abordagem por Crout em que apenas uma varredura se mostra necessária, os valores definitivos de cada elemento não nulo da linha básica, podem ser definitivamente escritos a medida em que se vão processando as colunas associadas.

Neste caso, apenas um acesso final de modo a se escrever o valor definitivo mostra-se realmente necessário.

● *O que vale se comentar também, é que embora um dos aparentes overheads da abordagem via Crout apresentada neste trabalho, qual seja um possível número excessivo de varreduras na linha base corrente, na verdade se encontra presente em tão ou maior grau, nas abordagens convencionais via o uso de vetores de trabalho, pois simplesmente nestes casos, se efetuam necessariamente múltiplas varreduras, cada uma associada a uma dada linha contribuinte em particular.*

Ou seja, numa abordagem por Crout, em que o número de varreduras seja próximo de um valor constante  $\tau$  (a cada etapa básica), a estratégia de utilização de listas simbólicas de endereços para se acessar apenas as colunas necessárias na linha base, torna a implementação do método de Crout seguramente mais eficiente do que uma implementação

tradicional via o uso de vetores de trabalho.

O preço que se terá de pagar em termos de espaço adicional com a lista de endereços, será alto, porém com uma constante de proporcionalidade ditada pelo número médio de varreduras  $\tau$  multiplicado pelo número total de elementos não nulos da matriz de fatores.

Nos casos em que duas ou três varreduras mostram-se suficientes, percebe-se que este *overhead* ainda se encontra dentro de um limite tolerável, tendo em vista um melhor desempenho em tempos de execução.

Nos casos em que o número de varreduras torna-se maior, uma estratégia não apresentada formalmente neste trabalho, é simplesmente se considerar uma variante da forma de varredura tradicional como a apresentada no procedimento IV.1(3) (e que indiscriminadamente passa por todos os elementos da linha base), simplesmente incorporando ao *loop* mais interno em cada varredura um par reduzido de índices, correspondentes a delimitação da máxima disposição estrutural de colunas a serem consideradas em cada etapa básica em questão.

Ou seja, restringir-se a varredura, apenas ao conjunto de elementos não nulos na linha base, compreendidos dentro do maior padrão de contribuição a ser encontrado em cada varredura.

Tais estratégias, conferem portanto ainda mais flexibilidade para uma implementação que ao mesmo tempo atenda aos objetivos de tempo x memória, de forma um pouco mais balanceada que as demais.

As vantagens de estratégias tipo Crout, são portanto:

- Evitar o acesso desnecessário a posições temporárias de memória, escrevendo-se apenas o valor definitivo de cada novo fator gerado, permitindo uma eficiente exploração da localidade de acessos, e a plena utilização de registradores.
- Perfazer um número significativamente menor de varreduras e inicializações de *loops* do que as estratégias convencionais mediante vetores de trabalho expandido.



Uma última observação a se fazer nesta seção, é de que ao menos uma variante do método de reordenamento ancestral, já vinha sendo utilizada a longa data na literatura, sem que se percebesse tratar-se apenas de um caso muito particular de reordenamento ancestral.

Simplesmente ao se lançar mão de um tratamento especial por *supernodes*, o que se está implicitamente considerando, é uma forma trivial de reordenamento, na qual as linhas contribuintes relativas ao *supernode* são naturalmente deixadas para o final do processo de eliminação a cada etapa.

Tais linhas pertencentes a *supernodes* são por construção as últimas linhas consecutivas na estrutura de  $U^T$ , razão pela qual nunca se havia pensado em reordená-las até então, visto já se encontrarem numa ordem julgada a princípio a mais indicada, (ou seja a ordem sequencial crescente dos índices associados a cada linha da transposta de  $U$ ).

● *Uma outra observação ainda mais esclarecedora, é que num caso ainda mais particular de estrutura supernodal, como o de matrizes banda, e onde originalmente tiveram origem os métodos de armazenamento "por envelopes", já se fazia o mesmo tipo de reordenamento ancestral implícito, (sem se perceber), simplesmente pelo fato da ordem natural das contribuições ser a mais propícia a se adotar num esquema por envelopes.*

Na verdade percebe-se agora, que a representação por envelopes não foi "feita" apenas para matrizes banda, mas sim para *qualquer* matriz, pois seguindo uma ordem ancestral de contribuições na representação reduzida das linhas em cada família, a estrutura final é da mesma forma que a encontrada nas matrizes de cálculo estrutural nas últimas décadas, (com matrizes tipo banda sendo apenas o caso particular mais simples e reconhecido de aplicação *implícita* desta técnica).

O que se nota com este trabalho, é que se abre um novo leque de opções em função de outros reordenamentos de contribuições que porventura venham a ser idealizados no futuro, bastando que deseje desvendar novos segredos, que por ora ainda soariam como enigmas encrustados numa entre tantas outras conchas de uma praia deserta, mas inteiramente aberta e exposta ao poder da curiosidade humana.

## Capítulo V

## RESULTADOS COMPUTACIONAIS

Neste capítulo são apresentados resultados computacionais "preliminares", com o objetivo de se comparar o desempenho de algumas das metodologias apresentadas ao longo do texto, em uma arquitetura escalar convencional tipicamente encontrável nos dias atuais.

Para tal se utilizou de uma configuração IBM-PC compatível, com processador INTEL 80486/DX-2 com clock interno de 66 Mhz, barramento ISA em 11 Mhz, 8 Mb de memória principal em Zero Wait-State (com tempo de acesso de 70 Nano-segundos), 256 Kb de memória cache em Zero Wait-State (com tempo de acesso de 20 Nano-segundos e opção de leitura em 2-1-1-1 ciclos), BIOS AMI Versão 1992 e chipset OPTi modelo OP495SLC.

O ambiente de programação contou com o MS-DOS 6.2, gerenciador de memória QEMM 7.01, cache de disco NCACHE 2.0, compressor de dados STACKER 3.0, e compilador FORTRAN Lahey (F77L-EM/32) Versão 4.01 com DOS-Extender Ergo (OS386) Versão 2.1.05.

As opções de compilação utilizadas são apresentadas abaixo, a título de complementação.

F77L-EM/32 - FORTRAN 77, Version 4.01;  
(C) Copyright 1988 through 1991; Lahey Computer Systems, Inc.

OPTION	DESCRIPTION	OPTION	DESCRIPTION
/n0	- Standard FORTRAN 77 IMPLICIT	/nL	- No Line-number table
/n7	- Optimize inter-statement	/nP	- No constant arg Protection
/nA2	- No allocatable array checking	/nQ	- No Quirky situations
/nB	- No Bounds checking	/R	- Remember local variables
/nC	- Ignore nonstandard usage	/nS	- No SOLD file created
/nC1	- INTEGER constants 4 bytes	/nT	- INTEGER*4, LOGICAL*4 default
/D	- DIRECT files without headers	/V	- VAX Fortran interpretation
/nH	- No Hardcopy source listing	/W	- Display Warning messages
/nI	- No Interface checking	/nX	- No Xref listing
/nK	- Generate 80x87 code	/Z1	- Production Optimizations

Os problemas teste considerados foram extraídos de aplicações reais, do conjunto de problemas da NETLIB [G44], [D54], voltados para a área de Programação Linear.

Segue-se portanto na fig\_V(1), uma tabela descritiva com os nomes e dimensões dos problemas (já transformados na forma simétrica, definida positiva, mediante o produto  $A A^T$ ), bem como o número de fatores triangulares (inferiores), após a etapa de fatoração simbólica.

Problem	Rows	$\text{nonz}((PA)^T PA)$	$\text{nonz}(L)$
<i>Afro</i>	27	90	107
<i>ADLittle</i>	56	377	404
<i>Scagr7</i>	129	606	734
<i>Sc205</i>	205	654	1182
<i>Share2b</i>	96	871	1026
<i>Share1b</i>	117	967	1425
<i>Scorpion</i>	388	1915	2324
<i>Scagr25</i>	471	2370	2948
<i>ScTap1</i>	300	1686	2667
<i>BrandY</i>	220	2190	2850
<i>Scsd1</i>	77	1133	1392
<i>Israel</i>	174	3545	3707
<i>BandM</i>	305	2929	4114
<i>Scfxm1</i>	330	3143	4963
<i>E226</i>	223	2683	3416
<i>Scrs8</i>	490	1953	5134
<i>Beaconfd</i>	173	1720	1727
<i>Scsd6</i>	147	2099	2545
<i>Ship04s</i>	402	2827	3134
<i>Scfxm2</i>	660	6306	9791
<i>Ship04l</i>	402	4147	4384
<i>Ship08s</i>	778	3552	4112
<i>ScTap2</i>	1090	6595	14870
<i>Scfxm3</i>	990	9469	14619
<i>Ship12s</i>	1151	4233	5063
<i>Scsd8</i>	397	4280	5879
<i>ScTap3</i>	1480	8866	19469
<i>CzProb</i>	929	6265	6655
<i>Ship08l</i>	778	6772	7128
<i>Ship12l</i>	1151	8959	9501

fig\_V(1) Problemas Teste da NETLIB

Inicialmente apresenta-se na fig\_V(2), uma tabela com os tempos de execução para algumas fases complementares de pré-processamento inicial, como o reordenamento ótimo, a fatoração simbólica, a transposição estrutural visando reordenar a estrutura de fatores por colunas em cada linha, e o tempo gasto no procedimento de reordenamento ancestral das colunas em JUTORD.

Tempo de computacao em Segundos

PROBLEM	OPTIMAL ORDER	SYMBOLIC FACTOR	SYMMETRIC TRANSP	ANCESTRAL REORD
afiro	9.94E-03	1.70E-04	2.20E-04	2.20E-04
adlittle	1.75E-02	4.40E-04	7.60E-04	9.80E-04
sc205	2.25E-02	1.65E-03	2.20E-03	3.05E-03
share1b	4.77E-02	1.95E-03	3.00E-03	3.30E-03
scorpion	6.10E-02	3.30E-03	5.50E-03	5.50E-03
sctapl	6.78E-02	3.30E-03	4.95E-03	7.40E-03
scagr25	5.88E-02	4.30E-03	6.60E-03	1.10E-02
israel	1.74E-00	1.70E-02	2.70E-02	2.20E-02
brandy	2.07E-01	3.80E-03	7.10E-03	7.70E-03
bandm	2.41E-01	6.10E-03	9.90E-03	1.16E-02
scfxm1	2.01E-01	6.60E-03	1.10E-02	1.10E-02
e226	1.96E-01	4.40E-03	7.80E-03	8.80E-03
scrs8	2.55E-01	7.60E-03	1.32E-02	1.86E-02
beaconfd	1.93E-01	3.55E-03	5.75E-03	6.35E-03
scsd6	9.23E-02	2.80E-03	4.90E-03	6.60E-03
ship04s	1.49E-01	4.40E-03	7.60E-03	8.80E-03
czprob	9.89E-01	6.00E-03	2.20E-02	2.20E-02

fig\_V(2) Fases complementares de pré-processamento

A seguir, apresenta-se na fig\_V(3), o tempo de fatoração, expresso em segundos, para algumas alternativas consideradas neste trabalho, especificamente para o caso de sub-matrizes "densas".

Tempo de Fatoracao em Segundos (Matrizes Densas)

n	5	10	20	30	40	50	80
NUMFAT	7.72E-05	3.66E-04	2.06E-03	6.10E-03	1.35E-02	2.54E-02	9.89E-02
LSTFAT	4.28E-05	2.38E-04	1.66E-03	5.63E-03	1.25E-02	2.57E-02	1.56E-01
ROWFAT	5.38E-05	2.76E-04	1.71E-03	5.32E-03	1.19E-02	2.32E-02	9.51E-02
COLFAT	4.44E-05	2.33E-04	1.50E-03	4.75E-03	1.12E-02	2.14E-02	8.74E-02
RECFAT	3.54E-05	1.69E-04	1.02E-03	3.53E-03	7.53E-03	1.41E-02	5.61E-02
GUSFAT	2.52E-05	1.37E-04	9.67E-04	3.28E-03	-- ND --	-- ND --	-- ND --

(\*) -- ND -- = Nao Disponivel (codigo gerado excedeu capacidade do compilador)

fig\_V(3) Fatoração de matrizes densas

A rotina NUMFAT, corresponde ao procedimento convencional por linhas, com geração dinâmica da informação da transposta de U, apresentada originalmente em IP11.

LSTFAT, corresponde a um procedimento simbólico por "listas de endereços", com geração por linhas, conforme apresentado na seção III.2.

ROWFAT e COLFAT, correspondem aos procedimentos convencionais via o uso de vetores de trabalho, com geração dos fatores por linhas/colunas, como apresentado na seção II.4.

RECFAT corresponde ao procedimento exclusivamente voltado para sub-matrizes densas, baseado na exploração da "recursividade" do código no estilo *loop free* apresentado na seção III.1.

Finalmente GUSFAT corresponde a um procedimento típico no estilo de Gustavson IG141, com um código FORTRAN proporcional ao número total de operações aritméticas (e que no caso denso considerado em particular, é governado por um crescimento da ordem de  $n^3$  linhas de código).

Cabe ressaltar, que na abordagem puramente no estilo de Gustavson, o tamanho do código gerado, já para matrizes de dimensão superior a 30, superou os limites do particular compilador FORTRAN utilizado, razão pela qual, só se dispõe de medidas de tempo de execução até o máximo valor tolerado de  $n$  para esta forma de abordagem.

Apresenta-se a seguir, na fig\_V(4), os tempos de fatoração para um grupo de problemas extraídos eletronicamente da NETLIB, em que os procedimentos convencionais via o uso de vetores de trabalho expandido, com geração por linhas/colunas, são comparados com duas implementações do método de codificação binária (apresentado na seção IV.1). O procedimento BINFAT se baseia numa varredura completa com códigos associados a todos os elementos de cada linha base, ao passo que ADRFAT se baseia na utilização concomitante de uma lista simbólica de endereços, especificando apenas os elementos necessários.

## Tempos de execucao em segundos

PROBLEM	COLFAT	ROWFAT	BINFAT	ADRFAT
afiro	3.40E-04	3.30E-04	4.50E-04	3.90E-04
adlittle	2.00E-03	2.30E-03	2.20E-03	2.10E-03
sc205	4.95E-03	5.75E-03	6.35E-03	5.75E-03
share1b	1.15E-02	1.29E-02	9.05E-03	8.50E-03
scorpion	1.32E-02	1.43E-02	1.49E-02	1.43E-02
sctapl	1.70E-02	1.86E-02	1.82E-02	1.70E-02
scagr25	1.38E-02	1.54E-02	1.77E-02	1.77E-02
israel	6.80E-01	6.42E-01	1.75E-01	1.76E-01
brandy	5.17E-02	5.39E-02	3.19E-02	3.08E-02
bandm	5.16E-02	5.38E-02	4.40E-02	4.18E-02
scfxm1	5.50E-02	5.92E-02	4.62E-02	4.40E-02
e226	4.40E-02	4.62E-02	3.52E-02	3.30E-02
scrs8	1.04E-01	1.09E-01	6.70E-02	6.24E-02
beaconfd	3.63E-02	3.68E-02	2.26E-02	2.14E-02
scsd6	2.37E-02	2.53E-02	2.31E-02	2.25E-02
ship04s	2.75E-02	2.75E-02	2.26E-02	2.10E-02
czprob	8.02E-02	7.90E-02	5.28E-02	4.38E-02

fig\_V(4) Fatoração de matrizes extraídas da NETLIB

Complementando os resultados apresentados nesta seção, apresenta-se na fig\_V(5), uma medida do tempo gasto com fases auxiliares do processo de resolução de sistemas, como a fase de retro-substituições implementada no procedimento SOLSYS, bem como duas estatísticas de importância para a comparação do dispêndio total dos métodos baseados em formas de codificação, como os do capítulo IV.

Neste caso, apresenta-se em CODBIN e CODENV, uma medida do tempo gasto unicamente com os procedimentos de codificação binária, e por envelopes, para o mesmo conjunto de problemas teste.

Dos resultados apresentados, percebe-se que como foi comentado em seções anteriores, o tempo de codificação na prática se mostrou comparável ao tempo de uma fatoração numérica adicional, o que considerando-se o escopo de aplicação das novas rotinas propostas, em aplicações onde o número de refatorações numéricas deve ser significativo, nos mostra que as abordagens apresentadas neste trabalho, se

encontram no "caminho correto", no sentido de se poder alcançar maiores reduções dos *overheads* computacionais.

Tempos de execucao em segundos

PROBLEM	CODBIN	CODENV	SOLSYS
afiro	1.60E-04	2.70E-04	2.20E-04
adlittle	1.20E-03	1.76E-03	7.80E-04
sc205	3.00E-03	4.65E-03	3.00E-03
share1b	6.55E-03	9.35E-03	2.75E-03
scorpion	8.20E-03	1.10E-02	5.50E-03
sctapl	9.90E-03	1.70E-02	5.50E-03
scagr25	9.30E-03	1.37E-02	7.20E-03
israel	2.53E-01	4.78E-01	2.80E-02
brandy	2.80E-02	3.73E-02	7.70E-03
bandm	2.84E-02	5.18E-02	9.80E-03
scfxm1	3.06E-02	5.40E-02	1.10E-02
e226	2.42E-02	3.64E-02	6.60E-03
scrs8	5.38E-02	1.84E-01	1.86E-02
beaconfd	1.97E-02	3.24E-02	6.60E-03
scsd6	1.37E-02	1.98E-02	5.50E-03
ship04s	1.75E-02	3.90E-02	8.20E-03
czprob	4.18E-02	1.48E-01	2.54E-02

fig\_V(5) Medidas complementares de tempo

Cabe lembrar, que os procedimentos basedos nas formas de codificação via o método de Crout, foram implementados apenas em FORTRAN, sem se procurar qualquer tentativa de se otimizar manualmente o código de alto nível empregado para a processo de solução.

Conforme se notou na seção IV.1, a comparação mais expressiva, seria implementando-se tais métodos na linguagem C, por suas nítidas vantagens sobre FORTRAN, no tocante ao acesso a "listas simbólicas de endereços", o que nesta segunda linguagem, necessita ser "simulado", mediante o uso de vetores de índices.

Para concluir esta breve seção, onde se apresentou resultados ainda "preliminares", sem qualquer intenção de espelhar uma estimativa a mais próxima do verdadeiro potencial das novas abordagens, pode-se notar que para problemas de pequena dimensão, o uso das técnicas de codificação não se mostrou tão vantajoso assim (pelo menos

para a classe particular de problemas), muito se devendo ao fato de que os procedimentos de codificação não terem sido "otimizados" no sentido de uma máxima compactação da informação.

Para problemas de maior porte, especialmente aqueles em que características "supernodais" se fazem notar maciçamente, como no caso do problema "quase denso" ISRAEL, os métodos de codificação apresentaram nítida vantagem sobre as estratégias convencionais de solução.

Como comentado no final do último capítulo, o método de reordenamento ancestral e as estratégias de codificação, permitem uma maior flexibilidade que as demais técnicas já exploradas na literatura, simplesmente por permitir que novos critérios de ordenamento ótimo (visando a redução de fill-in's), possam ser introduzidos, tomando-se como um critério de desempate, a minimização das informações simbólicas necessárias para se caracterizar completamente o processo de fatoração.

É neste caminho que deve avançar a presente pesquisa, com uma possível extensão a novas classes de arquiteturas, como as do tipo RISC superescalar, ou num espectro ainda maior de arquiteturas do tipo paralelo.

Para arquiteturas escalares convencionais, muito ainda pode ser feito, num terreno onde a princípio se julgava completamente "exaurido" de possibilidades.



## Capítulo VI

## CONCLUSÕES

Em face do elevado número de alternativas e estratégias de solução apresentadas neste trabalho, conclusões definitivas mostram-se seguramente difíceis e fadadas a não espelhar toda a realidade.

O que se apresenta portanto, é um conjunto de diretivas que se supõe sejam as melhores a serem seguidas, no sentido de se aprimorar ainda mais os métodos de resolução esparsa de sistemas lineares com estrutura estática.

- Alternativas simbólicas, apesar de todo o grau de maior sofisticação e dispêndio de recursos básicos como espaço adicional de armazenamento, mostram-se viáveis e implementáveis, tendo-se em vista maiores ganhos em termos de tempo de CPU, em detrimento à um maior consumo dos demais recursos básicos.
- Dentre as alternativas "tradicionais" já consagradas à longa data pela literatura, a geração de fatores por colunas, com um armazenamento seqüencial por linhas da matriz original e de fatores como no procedimento II.4(2), mostra-se a mais indicada, especialmente em matrizes com poucos elementos não nulos, onde seguramente características da forma *supernodal* não se mostram significativas.
- A aplicação de mais de uma estratégia de eliminação, como nas abordagens híbridas por janelas consideradas neste trabalho, apontam para um caminho no sentido de se melhor ajustar o processo de eliminação numérica, as características fundamentais de cada classe de matrizes consideradas.
- Estratégias especificamente voltadas para porções completamente densas podem ser empregadas com sucesso, visto em muitos dos casos, ser esta uma porção significativa do tempo total de CPU dispendido.

- Do mesmo modo, técnicas baseadas na exploração de *supernodes* mostram-se as mais adequadas a se adotar, quando na presença significativa de tais padrões estruturais, (encontrados com frequência, nas mais diversas classes de matrizes e aplicações da esparsidade).
- Métodos do tipo Crout simbólicos como os apresentados neste trabalho, são merecedores de mais atenção no futuro, visto conterem um amplo espectro de possibilidades ainda por explorar.
- O reordenamento ancestral apresentado neste trabalho, viabiliza o reconhecimento de padrões compactos de codificação e de eficiente decodificação até então ainda não considerados pela literatura.
- O reordenamento ancestral é apenas a forma mais conservadora de se enfrentar o problema de codificação por envelopes, por ser seguramente (por construção e pela natureza "ancestralmente aditiva" do processo de eliminação) uma estratégia em que tal forma de codificação pode ser empregada.
- Esquemas híbridos baseados na utilização das alternativas de codificação apresentadas neste trabalho, parecem apontar na direção mais propícia para implementações eficientes e "ajustadas" sob-medida para cada matriz e padrão característico encontrado.
- A divisão do processo de eliminação numérica em duas fases a níveis distintos de granularidade, como a abordagem por "janelas de processamento" (aplicada inicialmente), e a abordagem por "estratégias de codificação", aplicada posteriormente, mostra-se entre todas as opções apresentadas, a estratégia mais balanceada, e com maior flexibilidade sobre as demais.

## Apêndice A

## NOTAÇÃO E ESTRUTURAS DE DADOS BÁSICAS

Neste apêndice são apresentadas as estruturas de dados básicas, utilizadas para a representação de matrizes esparsas, bem como algumas das principais estruturas auxiliares empregadas no processo de eliminação, de vários procedimentos abordados neste trabalho.

Assume-se que o sistema esparso de equações lineares a ser resolvido seja da forma:

$$A x = b$$

Com:

$A \in \mathbb{R}^n \times \mathbb{R}^n$  simétrica (definida positiva)

$b \in \mathbb{R}^n$  denso

$x \in \mathbb{R}^n$  denso

Em todos os métodos considerados, a solução deste problema se dá mediante a fatoração triangular da matriz original  $A$  na forma:

$$U^T D U$$

Com:

$U^T \in \mathbb{R}^n \times \mathbb{R}^n$  triangular inferior (diagonal unitária)

$D \in \mathbb{R}^n \times \mathbb{R}^n$  diagonal

$U \in \mathbb{R}^n \times \mathbb{R}^n$  triangular superior (diagonal unitária)

De posse das matrizes  $D$  e  $U$ , obtidas na etapa de fatoração, a solução do sistema original se dá mediante uma fase complementar de retro-substituições triangulares na forma:

$$\begin{aligned} U^T z &= b && \text{(forward)} \\ D y &= z && \text{(diagonal)} \\ U x &= y && \text{(backward)} \end{aligned}$$

Assume-se durante todo o trabalho, que apenas a representação da matriz de fatores  $U$  seja utilizada para fins

de armazenamento, com os elementos originais de  $A$  mapeados diretamente nas posições correspondentes na matriz  $U$  (com as demais posições em  $U$ , associadas aos elementos de *fill-in* inicialmente zeradas).

Os valores numéricos de  $A$ , desta forma são irremediavelmente perdidos ao final da etapa de eliminação, uma vez que os fatores de  $D$  e  $U$  associados à cada uma das linhas (geradas com o decorrer do processo), vão sendo reescritos por sobre os valores originalmente armazenados.

A representação esparsa adota para a matriz original  $A$  e de fatores  $U$ , se baseia num armazenamento sequencial segmentado por linhas excluindo-se os elementos diagonais, conforme apresentado na seção II.1, mediante a utilização dos seguintes vetores:

#### Matriz $U$

*dimensão*

$n$	<b>IU</b>	Apontadores para a posição inicial de armazenamento de cada linha.
$n$	<b>IUF</b>	Apontadores para a posição final de armazenamento de cada linha.
$\text{Nonz } U$	<b>JU</b>	Colunas associadas aos elementos não nulos de cada linha.
$\text{Nonz } U$	<b>UN</b>	Valores numéricos associados aos elementos não nulos.

Originalmente contendo os valores de  $A$ , com as demais posições dos *fill-in's* zeradas, e posteriormente os valores de  $U$  obtidos ao final da eliminação.

A representação adotada para a diagonal da matriz original  $A$  e de fatores  $D$ , se baseia num armazenamento denso mediante o seguinte vetor:

**Matriz D***dimensão*

n	<b>DI</b>	Contendo originalmente os valores diagonais de A, e posteriormente o inverso dos valores de D ao final da eliminação.
---	-----------	---

Algumas estruturas auxiliares adotadas ao longo de todo o texto nos diversos procedimentos apresentados são:

**Estrutura de  $U^T$** *dimensão*

n	<b>IUT</b>	Apontadores para a posição inicial de armazenamento de cada linha de $U^T$ .
---	------------	--

n	<b>IUTF</b>	Apontadores para a posição final de armazenamento de cada linha de $U^T$ .
---	-------------	--

Nonz U	<b>JUT</b>	Colunas associadas as posições não nulas de cada linha de $U^T$ .
--------	------------	---

Indicando cada uma das linhas anteriores a serem subtraídas da linha base corrente em cada etapa  $i$  do processo de fatoração.

**Apontadores Dinâmicos de Posição Inicial***dimensão*

n	<b>IUP</b>	Apontadores dinâmicos para a posição inicial em cada linha a ser utilizada durante a eliminação das linhas subseqüentes.
---	------------	--

Atualizados a cada etapa básica  $i$  do processo de fatoração.

**Lista Simbólica de Endereços***dimensão*

Noper U	<b>LSTADD</b>	Lista simbólica dos endereços associados as posições de memória dos elementos em cada linha base de U a serem acessados ao longo de todo o processo de eliminação.
---------	---------------	--

### Lista Simbólica de Códigos

**LSTCOD** Lista de códigos, que de algum modo caracterizem uma seqüência de operações de ponto flutuante a serem efetuadas durante o processo de eliminação.

Sendo posteriormente "interpretada" durante alguma fase de decodificação, em procedimentos especialmente construídos de modo a se aproveitar das informações codificadas.

### Vetor de Trabalho Expandido

*dimensão*

**n**      **W**      Vetor auxiliar expandido, contendo temporariamente os valores numéricos na forma descompactada dos elementos de cada linha base do processo.

Utilizado para se efetuar a acumulação das contribuições das demais linhas sobre a linha base corrente.

### Sucessores na Árvore de Eliminação

*dimensão*

**n**      **PARENT**      Estrutura de sucessores na árvore de eliminação associada a matriz de fatores.

Indicando a primeira linha a depender necessariamente da eliminação prévia da linha associada para poder ser eliminada.

## Apêndice B

## GLOSSÁRIO

<b>Acesso Contíguo</b>	Acesso a posições consecutivas de memória.
<b>Acesso Linear</b>	O mesmo que acesso contíguo ou sequencial
<b>Acesso Indireto</b>	Acesso de forma duplamente indexada aos elementos não nulos de uma dada linha.
<b>Acesso Sequencial</b>	Acesso de forma linear contígua aos elementos não nulos de uma dada linha.
<b>Algoritmo</b>	Conjunto de instruções a serem seguidas de modo a se solucionar algum problema.
<b>Alternativa</b>	O mesmo que procedimento ou algoritmo
<b>Árvore de Eliminação</b>	Estrutura determinante do nível de parentesco e precedência entre as linhas da matriz de fatores.
<b>Associados</b>	O mesmo que fatores ou elementos associados
<b>Atualização</b>	O mesmo que eliminação
<b>Base</b>	O mesmo que etapa ou linha corrente
<b>Código</b>	Informação básica, suficiente para caracterizar uma operação ou conjunto de operações.

<b>Código</b>	Versão implementada a nível de um programa ou procedimento.
<b>Corrente</b>	O mesmo que base
<b>Densidade</b>	Medida do percentual de elementos não nulos de uma matriz.
<b>Elementos</b>	Coefficientes não nulos de uma matriz.
<b>Elementos Associados</b>	O mesmo que fatores associados
<b>Endereço</b>	Posição de memória associada a um dado elemento na representação esparsa de uma matriz.
<b>Eliminação</b>	Processo de cancelamento de elementos de modo a se levar a matriz original do sistema à forma triangular.
<b>Esparsidade</b>	Característica estrutural de uma matriz, normalmente explorada na presença de poucos elementos não nulos a serem armazenados ou operados posteriormente.
<b>Estrutural</b>	Informação a nível apenas da disposição espacial dos elementos não nulos de uma matriz.
<b>Etapa</b>	Fase fundamental do processo de eliminação, correspondendo ao <i>loop</i> mais externo na variável <i>i</i> , onde são gerados os elementos da <i>i</i> 'ezima linha ou coluna da matriz de fatores.
<b>Fase Numérica</b>	O mesmo que fatoração numérica
<b>Fatoração</b>	O mesmo que eliminação



<b>Fatoração Numérica</b>	Fase correspondente a geração dos valores numéricos dos elementos da matriz de fatores.
<b>Fatoração Simbólica</b>	Fase correspondente a determinação da posição estrutural dos fatores de $U$ .
<b>Fatores Associados</b>	Elementos na linha base, correspondentes as mesmas colunas de elementos de outras linhas a serem subtraídas destes.
<b>Fill-In</b>	Elemento não nulo introduzido ao longo do processo de eliminação, em decorrência da combinação escalar de linhas prévias com a linha base de cada etapa.
<b>i</b>	Variável básica do processo de eliminação, associada ao <i>loop</i> mais externo, indicando a linha ou coluna da matriz de fatores a ser gerada a cada etapa.
<b>j</b>	Variável associada ao <i>loop</i> mais interno do processo de eliminação, correspondendo a posição dos elementos não nulos de cada uma das linhas anteriores sendo subtraídas da linha base corrente.
<b>k</b>	Variável associada ao <i>loop</i> intermediário do processo de eliminação, varrendo a estrutura da matriz $U^T$ , de modo a se determinar quais linhas anteriores subtrair da linha base corrente.
<b>Linear</b>	O mesmo que acesso seqüencial ou contíguo
<b>Linha Base</b>	O mesmo que linha corrente

<b>Linha Corrente</b>	Linha correspondente a <i>i</i> 'ezima etapa do processo de eliminação a ser operada de modo a se obterem os fatores associados.
<b>Método</b>	O mesmo que algoritmo
<b>Numérica</b>	O mesmo que fase ou fatoração numérica
<b>Operação</b>	Normalmente uma seqüência de operações de ponto flutuante da forma $a \pm b * c$ .
<b>Overhead</b>	Dispêndio adicional de recursos computacionais, advindo de alguma ineficiência a nível de implementação.
<b>Procedimento</b>	O mesmo que algoritmo
<b>Processo</b>	Fatoração da matriz $A$ na forma $U^T D U$ .
<b>Programa</b>	Conjunto de instruções a serem efetuadas sobre estruturas de dados.  Codificação em alguma linguagem de alto nível de uma seqüência de instruções visando a implementação de um algoritmo.
<b>Representação</b>	O mesmo que estrutura de armazenamento
<b>Segmentada</b>	Estrutura de armazenamento na qual os elementos não nulos das linhas de uma matriz são armazenados em posições contíguas em cada linha, com as posições inicial e final de linhas consecutivas não necessariamente contíguas.
<b>Seqüencial</b>	O mesmo que contíguo
<b>Seqüencial Segmentada</b>	O mesmo que estrutura segmentada

**Simbólica**

Informação auxiliar com objetivo de tornar mais eficiente a fase numérica da fatoração.

## REFERÊNCIAS BIBLIOGRÁFICAS

- IA11 Adler, Karmarkar, Resende & Veiga "Data Structures and Programming Techniques for the Implementation of Karmarkar Algorithm for Linear Programming", ORSA Journal on Computing, Vol 1 (2), pp 84-106, 1989.
- IA21 Alvarado, Tinney & Enns "Sparse Matrix Inverse Factors", Manuscript 88 SM 728-8, IEEE Summer Power Meeting, Portland, (a ser publicado em IEEE Trans. on Power Systems), 1988.
- IA51 Alvarado, F.L. "A Note on Sorting Sparse Matrices", Proceedings IEEE, Vol 67 (9), pp 1362-1363, 1979.
- IA61 Alvarado, F.L., Yu, D.C & Betancourt, R. "Ordering Schemes for Partitioned Sparse Inverses", SIAM Symposium on Sparse Matrices, Salishan Lodge, Oregon, 1989.
- IA81 Arantes, R.D. "Uma Abordagem Híbrida via Processamento Simbólico, para a Resolução Eficiente de Sistemas Lineares Esparsos com Estrutura Estática, Especialmente Aplicável à Algoritmos de Pontos Interiores para Programação Linear" Relatório Técnico 102/91 CEPEL, 1990
- IA111 Ashcraft, C. & Grimes, R. "The Influence of Relaxed Supernode Partitions on the Multifrontal Method", ACM Trans. on Math. Software, Vol 15 (4), pp 291-309, 1989.
- IA121 Ashcraft, C. et al. "Progress in Sparse Matrix Methods for Large Linear Systems on Vector Supercomputers", Int. J. of Supercomputer Appl., Vol 1 (4), pp 10-30, 1987.

- IB11 Bank, R.E. & Smith, R.K. "General Sparse Elimination Requires No Permanent Integer Storage", SIAM J. Sci. Stat. Comput., Vol 8 (4), pp 574-584, 1987.
- IB21 Bank, R.E. & Rose, D.J. "On the Complexity of Sparse Gaussian Elimination via Bordering", SIAM J. Sci. and Stat. Comput., Vol 11 (1), pp 145-160, 1990.
- IB81 Betancourt, R. "An Efficient Heuristic Ordering Algorithm for Partial Matrix Refactorization", IEEE Trans. on Power Systems, Vol 3 (3), pp 1181-1187, 1988.
- IB91 Benner, R., Montry, G. & Weigand, G. "Concurrent Multifrontal Methods: Shared Memory, Cache, and Frontwidth Issues", Int. J. Supercomputer Appl., Vol 1 (3), pp 26-44, 1987.
- IB101 Browne, Dongarra, Karp, Kennedy, & Kuck "Special Report: 1988 Gordon Bell Prize", IEEE Software, Vol 6, pp 78-85, 1989.
- IB111 Barret, R. et al. "Templates for the solution of Linear Systems: Building blocks for iterative methods", SIAM Publications, 1993.
- IC11 Carvalho, M.L. "On the Minimization of Work Needed to Factor a Symmetric Positive Definite Matrix", Manuscript ORC 87-14, Dept. of Industrial Eng. and Operations Research, University of California, Berkeley, 1987.
- IC161 Carnahan, B., Luther, H.A. & Wilkes, J.O. "Applied Numerical Methods", John Wiley & Sons, 1969.
- ID11 Dembart, B. & Erisman, A.M. "Hybrid Sparse Matrix Methods", IEEE Trans. Circuit Theory, Vol CT-20, pp 641-649, 1973.

- ID21 Dongarra, J.J. & Hinds, A.R. "Unrolling Loops in FORTRAN", Software Practice and Experience, Vol 9, pp 219-229, 1979.
- ID31 Dongarra, J.J. et al. "Solving Linear Systems on Vector and Shared Memory Computers", SIAM Publications, 1991.
- ID61 Duff, Erisman & Reid "Direct Methods for Sparse Matrices", Clarendon Press, Oxford, 1986.
- ID101 Duff, I.S. "Data Structures, Algorithms and Software for Sparse Matrices", Technical Report 84-1846, Harwell Laboratory, 1984.
- ID121 Duff, I.S. "MA27 - A set of Fortran subroutines for sparse symmetric linear equations", Report R10533, HMSO, AERE Harwell, 1982.
- ID231 Duff, I.S., Gould, N., Lescrenier, M. & Reid, J.K. "The Multifrontal Method in a Parallel Environment", Computer Science and Systems Division, Harwell Laboratory, Report Number CSS-211, Oxon, England, 1987.
- ID261 Duff, I.S. "On the Number of Nonzeros Added when Gaussian Elimination is Performed on Sparse Random Matrices", Math. of Computation, Vol 28 (125), pp 219-230, 1974.
- ID351 Duff, I.S. "Full Matrix Techniques in Sparse Gaussian Elimination", AERE Harwell, Tech. Report CSS 114, 1981.
- ID401 Duff, I.S. & Reid, J.K. "The Multifrontal Solution of Unsymmetric Sets of Linear Equations", SIAM J. Sci. and Stat. Comput., Vol 5 (3), pp 633-641, 1984.

- ID451 Duff, I.S. "A Sparse Future", em "Sparse Matrices and their uses", I.S. Duff (ed), Academic Press, pp 1-29, 1981.
- ID471 Dembo, R.S. "Solving Box-Constrained Quadratic Programming Problems on a Vector Processor", Tech. Report, Dept. of Computer Science, University of Toronto, Canada, 1987.
- ID491 Dantzig, G.B. "Linear Programming and Extensions", Princeton University Press, 1962.
- ID501 Durand, E. "Solutions Numériques des Équations Algébriques, Tome II: Systèmes de Plusieurs Équations", Masson & Cie, 1972.
- ID511 Demidovich, B.P. & Maron, I.A. "Computational Mathematics", Mir Publishers, 1976.
- ID541 Dongarra, J. & Grosse, E. "Distribution of Mathematical Software via Electronic Mail", Comm. ACM, Vol 30 (5), pp 403-407, 1987.
- ID551 Dongarra, J. "NA-NET Is Up and Running at Oak Ridge National Laboratory", SIAM News, Vol 24 (2), pp 22, 1991.
- [E1] Eisenstat, S.C. et al. "Algorithms and Data Structures for Sparse Symmetric Gaussian Elimination", SIAM J. Sci. Stat. Comput., Vol 2 (2), pp 225-237, 1981.
- [E2] Eisenstat, S., Gursky, M., Schultz, M. & Sherman, A. "YALE Sparse Matrix Package I: The Symmetric Codes", Int. J. for Numerical Methods in Engineering, Vol 18, pp 1145-1151, 1982.
- [F1] Forsythe, G.E. & Moler, C.B. "Computer Solution of Linear Algebraic Systems", Prentice-Hall, 1967.

- IF2] Forsythe, G.E., Malcolm, M.A. & Moler, C.B. "Computer Methods for Mathematical Computations", Prentice-Hall, 1977.
- IG1] Gay, D.M. "Massive Memory Buys Little Speed for Complete In-Core Sparse Cholesky Factorizations on Some Scalar Computers", Lin. Alg. Appl., Vol 152, pp 291-314, 1991.
- IG2] George, J.A. & Liu, J.W. "The Evolution of the Minimum Degree Ordering Algorithm", SIAM Review, Vol 31 (1), pp 1-19, 1989.
- IG3] George, A. & Liu, J.W. "An Optimal Algorithm for Symbolic Factorization of Symmetric Matrices", SIAM J. Comput., Vol 9 (3), pp 583-593, 1980.
- IG4] George, A. & Ng, E. "Symbolic Factorization for Sparse Gaussian Elimination with Partial Pivoting", SIAM J. Sci. Stat. Comput., Vol 8 (6), pp 877-898, 1987.
- IG5] George, A. & Liu, J.W.H. "Computer Solution of Large Positive Definite Systems", Prentice-Hall, 1981.
- IG7] George, A. & Liu, W.H. "A Note on Fill for Sparse Matrices", SIAM J. Numerical Analysis, Vol 12 (3), pp 452-455, 1975.
- IG10] George, A. & Liu, J.W.H. "The Design of a User Interface for a Sparse Matrix Package", ACM Trans. Math. Software, Vol 5 (2), pp 139-162, 1979.
- IG11] George, A. & Rashwan, H. "Auxiliary Storage Methods for Solving Finite Element Systems", SIAM J. Sci. and Stat. Comput., Vol 6 (4), pp 882-910, 1985.
- IG14] Gustavson, F.G. et al. "Symbolic Generation of an Optimal Crout Algorithm for Sparse Systems of Linear Equations", Journal ACM, Vol 17 (1), pp 87-109, 1970.



- IG151 Gustavson, F.G. "Some Basic Techniques for Solving Sparse Systems of Linear Equations" em "Sparse Matrices and Their Applications", D.J. Rose & R.A. Willoughby (ed.), Plenum Press, New York, pp 41-52, 1972.
- IG161 Gustavson, F.G. "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition", ACM Trans. on Math. Software, Vol 4 (3), pp 250-269, 1978.
- IG201 Gentleman, W.M. & George, A. "Sparse Matrix Software", em "Sparse Matrix Computations", J.R. Bunch & D.J. Rose (ed.), Academic Press, Inc., New York, 1976.
- IG271 Gomez, A. & Franquelo, L.G. "Node Ordering Algorithms for Sparse Vector Method Improvement", IEEE Trans. on Power Systems, Vol 3 (1), pp 73-79, 1988.
- IG281 Gomez, A. & Franquelo, L.G. "An Efficient Ordering Algorithm to Improve Sparse Vector Methods", IEEE Trans. on Power Systems, Vol 3 (4), pp 1538-1544, 1988.
- IG391 Golub, G.H. & Van Loan, C.F. "Matrix Computations", (Second edition), Johns Hopkins University Press, Baltimore, 1989.
- IG401 Golub, G.H. & O'Leary, D.P. "Some history of the Conjugate Gradient and Lanczos algorithms: 1948-1976", SIAM Review, Vol 31 (1), pp 50-102, 1989.
- IG441 Gay, D.M. "Electronic Mail Distribution of Linear Programming Test Problems", Math. Programming Society Comitee on Algorithms Newsletter, no. 13, 1985.

- IG451 Grosse, E. "Netlib News: Greatings", SIAM News, Vol 23 (6), pp 14, 1990.
- IG461 Grosse, E. "Netlib News: Searching for Files", SIAM News, Vol 25 (4), pp 10, 1992.
- IH11 Hachtel, G.D., Brayton, R.K. & Gustavson, F.G. "The Sparse Tableau Approach to Network Analysis and Design", IEEE Trans. on Circuit Theory, Vol CT-18 (1), pp 101-113, 1971.
- IH21 Hachtel, G.D. "Vector and matrix variability type in sparse matrix algorithms", em "Sparse Matrices and their Applications", Rose, D. & Willoughby, R. (ed.), Plenum Press, pp 53-66, 1972.
- IH51 Heath, M.T., Ng, E. & Peyton, B.W. "Parallel Algorithms for Sparse Linear Systems", SIAM Review, Vol 33 (3), pp 420-460, 1991.
- II11 Irons, B.M. "A frontal solution program for finite element analysis", Int. J. Numer. Meth. in Eng., Vol 2, pp 5-32, 1970.
- IJ11 Jung, H.W., Marsten, R.E. & Saltzman, M.J. "Numerical Factorization Methods for Interior Point Algorithms", ORSA J. Computing, Vol 6 (1), 1994.
- IK11 Karmarkar, N. "A New Polynomial Time Algorithm for Linear Programming", Combinatorica, Vol 4, pp 373-395, 1984.
- IK21 Karmarkar, N. & Ramakrishnan, K. "Implementation and Computational Results of the Karmarkar Algorithm for Linear Programming, Using an Iterative Method for Computing Projections", Tech. Report, AT&T, Bell Labs., New Jersey, 1988.
- IK41 Knuth, D.E. "The Art of Computer Programming: Vol 3, Sorting and Searching", Addison-Wesley, 1973.

- IL11 Liu, J.W.H. "A Note on Sparse Factorization in a Paging Environment", SIAM J. Sci. Stat. Comput., Vol 8 (6), pp 1085-1888, 1987.
- IL41 Liu, J.W.H. "The Multifrontal Method for Sparse Matrix Solution: Theory and Practice", SIAM Review, Vol 34 (1), pp 82-109, 1992.
- IL101 Liu, J.W.H. "A Compact Row Storage Scheme for Cholesky Factors Using Elimination Trees", ACM Trans. on Math. Software, Vol 12 (2), pp 127-148, 1986.
- IL111 Liu, J.W.H. "The Role of Elimination Trees in Sparse Factorizations", SIAM J. Matrix Anal. and Appl., Vol 11 (1), pp 134-172, 1990.
- IL131 Liu, J.W.H. "An Adaptive General Sparse Out-of-Core Cholesky Factorization Scheme", SIAM J. Sci. and Stat. Comput., Vol 8 (4), pp 585-599, 1987.
- IL141 Liu, J.W.H. "On the Storage Required in the Out-of-Core Multifrontal Method for Sparse Factorization", ACM Trans. on Math. Software, Vol 12 (3), pp 249-264, 1986.
- IL301 Lustig, I.J. "The Influence of Computer Language on Computational Comparisons: An Example from Network Optimization", ORSA J. Computing, Vol 2 (2), pp 152-161, 1990.
- IL311 Lustig, I.J., Marsten, R.E. & Shanno, D.F. "Interior Point Methods for Linear Programming: Computational State of the Art", ORSA J. Computing, Vol 6 (1), pp 1-14, 1994.
- IL321 Lustig, I.J., Marsten, R.E. & Shanno, D.F. "The Last Word on Interior Point Methods for Linear Programming - For Now", ORSA J. Computing, Vol 6 (1), pp 35-36, 1994.

- IM11 Marsten, R.E. "*User's Manual for the Research Version of OB1<sup>®</sup>*", Software Manual, School of Ind. and Systems Eng., Georgia Institute of Technology, Atlanta, 1990.
- IM31 Markowitz, H.M. "*The elimination form of the inverse and its application to linear programming*", Management Science, Vol 3, pp 255-269, 1957.
- IO11 Ogbuobiri, E.C., Tinney, W.F. & Walker, J.W. "*Sparsity-Directed Decomposition for Gaussian Elimination on Matrices*", IEEE Trans. on Power Apparatus and Systems, Vol PAS-89 (1), pp 141-155, 1970.
- IO31 Ortega, J.M. "Introduction to Parallel and Vector Solution of Linear Systems", Plenum Press, New York, 1988.
- IO51 Ortega, J.M. & Rheinboldt, W.C. "Iterative Solution of Nonlinear Equations in Several Variables", Academic Press, 1970.
- IP11 Pissanetsky, S. "Sparse Matrix Technology", Academic Press, Inc., London, 1984.
- IP61 Paige, G.C. & Saunders, M.A. "*LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares*", ACM Trans. on Math. Software, Vol 8 (1), pp 43-71, 1982.
- IR81 Rose, D., Tarjan, R. & Lueker, G. "*Algorithm Aspects of Vertex Elimination on Graphs*", SIAM J. Computing, Vol 5 (2), pp 266-283, 1976.
- IS11 Schreiber, R. "*A New Implementation of Sparse Gaussian Elimination*", ACM Trans. Math. Software, Vol 8 (3), pp 256-276, 1982.

- [S6] Sherman, A.H. "On the Efficient Solution of Sparse Systems of Linear and Non-Linear Equations", Report No. 46, Dept. of CS, Yale University, 1975.
- [S8] Sherman, M. & Brandwajn, V. "Partial Matrix Refactorization" IEEE Trans on Power Systems, Vol PWR5-1 (1), pp 193-200, 1986.
- [T1] Terry, L.A. e Pereira, M.V.F. "Tratamento de Sistemas Lineares Esparsos", Relatório Interno Projeto 7167, CEPEL, 1980.
- [T2] Tinney, W. & Walker, J. "Direct Solution of Sparse Network Equations by Optimally Ordered Triangular Factorization", Proc. IEEE, Vol 55, pp 1801-1809, 1967.
- [T15] Traub, J.F. "Iterative Methods for the Solution of Equations", Prentice-Hall, 1964.
- [V6] Varga, R.S. "Matrix iterative analysis", Prentice-Hall, 1962.
- [Y1] Yannakakis, M. "Computing the Minimum Fill-In is NP-Complete", SIAM J. Algebraic and Discrete Methods, Vol 2 (1), 1981.
- [Y3] Young, D.M. "Iterative Solution of Large Linear Systems", Academic Press, 1971.
- [Z5] Zenios, S.A. & Mulvey, J.M. "Vectorization and Multitasking of Nonlinear Network Programming Algorithms", Mathematical Programming, Vol 42, pp 449-470, 1988.
- [Z6] Zenios, S.A. "Parallel Computing: The Introduction of Novel Computer Architectures Greatly Alters Technical Landscape", OR/MS Today, pp 44-49, Agosto, 1992.