Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Análise da Influência de Algoritmos de Reordenação de Matrizes Esparsas no Desempenho do Método CCCG(η)

Fernanda Cristina do Carmo

Belo Horizonte

Agosto de 2005

Fernanda Cristina do Carmo

Análise da Influência de Algoritmos de Reordenação de Matrizes Esparsas no Desempenho do Método CCCG(η)

Orientador: José Monteiro da Mata
Co-Orientador: Frederico Ferreira Campos, filho

Dissertação apresentada ao curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

Agosto de 2005

Agradecimentos

Gostaria de agradecer a todos que contribuíram de forma direta ou indireta para a realização deste trabalho:

Ao professor Frederico Ferreira Campos, pela orientação, dedicação e incentivo.

Ao professor José Monteiro da Mata e às professoras Silvana Bocanegra e Kíssia Carvalho, pela contribuição no desenvolvimento desta dissertação.

A minha família, pela força.

Aos amigos, pelo apoio.

A Deus.

Resumo

Este trabalho consiste em analisar a influência de algoritmos de reordenação de matrizes esparsas no desempenho do método Cholesky controlado gradiente conjugado - CCCG(η). Este método tem se mostrado muito eficiente na solução de sistemas lineares de alta ordem com matriz simétrica e definida positiva.

São estudados algoritmos mais simples como o contagem de colunas e o Cuthill-McKee reverso, além de algoritmos mais sofisticados como o mínimo grau aproximado. Alguns resultados numéricos com o efeito das reordenações no preenchimento e no número de iterações do método são apresentados, mostrando experimentalmente que certos algoritmos, como o mínimo grau aproximado, podem trazer benefícios.

Estes benefícios consistem em uma aceleração da convergência do método e em uma redução da quantidade de armazenamento utilizado. Enfim, foi estabelecida a situação onde uma reordenação deve ser utilizada para melhorar o desempenho do CCCG(η).

Abstract

In this work we investigate the influence of reordering algorithms on the performance of controlled Cholesky conjugate gradient method - CCCG(η). This method has been proved to be efficient in solution of high linear systems with positive definite coefficient matrix.

It has been considered simple algorithms like column count and reverse Cuthill-McKee and more sophisticated algorithms like approximate minimum degree. Some numerical results on the effect of orderings on the fill-in and the iteration number have been presented. It is shown experimentally that certain reorderings like aproximated minimum degree can be very beneficial.

The benefits consist of a faster convergence of the method and a lower storage requirements. Finally, the situation where a reordering can improve the CCCG(η) was estabilished.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Em áreas da engenharia e da matemática aplicada, tais como otimização, redes neurais, sistemas eletromagnéticos, simulação de processos físicos, previsão de tempo e monitoramento de dados sísmicos, a maioria dos problemas envolve a solução de sistemas lineares. Esses sistemas lineares geralmente possuem matrizes de coeficientes de alta ordem ($\geq 10^4$) [TB97] e esparsas. A etapa de solução desses sistemas demanda a maior parte do processamento, aumentando a necessidade de técnicas computacionais cada vez mais eficientes.

Em relação às estruturas físicas, por exemplo, a indústria está confiando cada vez mais nestas simulações computacionais. As estruturas a serem analisadas pelo computador são discretizadas por meio de malhas complexas. Quanto melhor for a resolução destas malhas, maior é a precisão da simulação numérica correspondente. Entretanto, aumentando-se esta resolução, o tamanho do sistema linear correspondente a ser resolvido numericamente também aumenta. Problemas com milhões de variáveis estão sendo construídos hoje em dia e o tamanho destas malhas irá crescer substancialmente em um futuro próximo. Desta forma, o grande obstáculo no desempenho de simulações numéricas de grande escala é dado pela velocidade na qual sistemas de equações lineares são resolvidos.

A Figura 1.1 apresenta o resultado de um estudo feito pelo *Research Institute for Advanced Computer Science* da NASA [RIA] envolvendo a modelagem bi-dimensional do fluxo sobre a asa de um avião com aerofólio. Nesta simulação, os pontos críticos são representados por uma maior quantidade de informação, aumentando a resolução nestes locais. Variáveis como pressão hidrodinâmica e componentes de velocidade resultam em um sistema de equações diferenciais parciais (não lineares). Cada passo de uma iteração não linear requer a solução de um sistema de equações lineares. Como ambos, conectividade e localização geométrica

dos pontos da malha são conhecidos, é possível construir o grafo.

Este exemplo possui 4253 pontos (tamanho do sistema) e 28831 conexões. A estrutura da matriz de coeficientes correspondente a esta malha pode ser vista na Figura 1.2. Esta matriz é altamente esparsa, ou seja, possui a maioria dos elementos iguais a zero (densidade igual a 0,0016). Esta correspondência entre grafos e matrizes esparsas será vista com mais detalhes na seção 4.3.

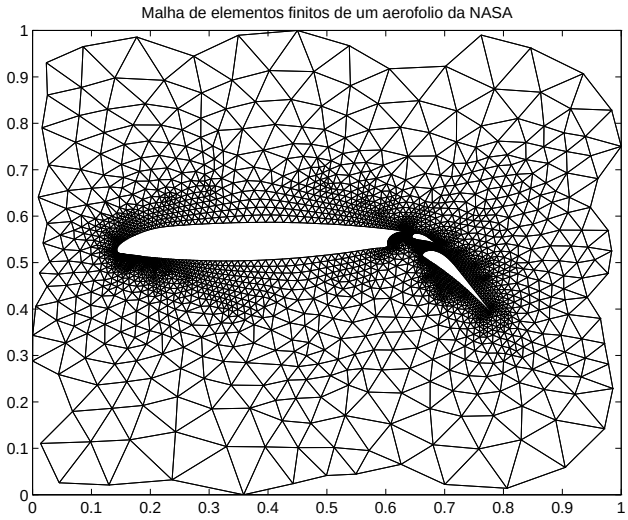


Figura 1.1: Malha de elementos finitos de um aerofólio de um avião.

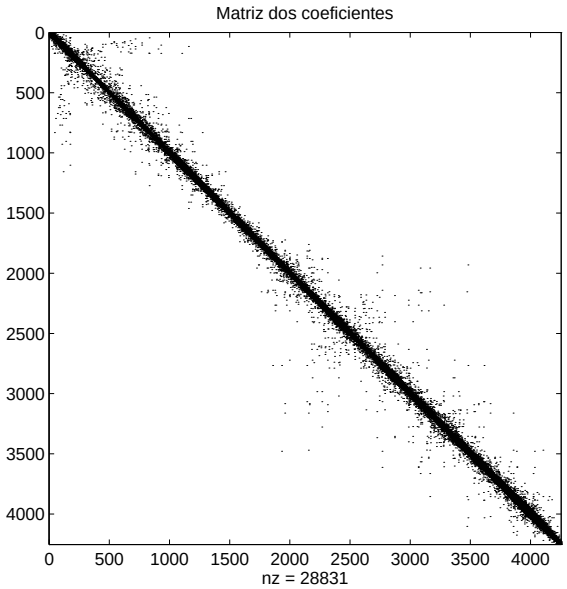


Figura 1.2: Estrutura da matriz de coeficientes.

Existe um grande número de métodos numéricos para solução de sistemas de equações

lineares. A escolha entre eles depende do tamanho e da estrutura da matriz de coeficientes do sistema a ser resolvido. Eles podem ser divididos em duas classes: métodos diretos e iterativos. Os métodos diretos são aqueles em que a solução do sistema é obtida, teoricamente, com um número finito de operações aritméticas. Já na segunda classe, a solução exata é obtida somente com um número infinito de operações, mas na prática existem critérios de parada que tornam a solução aproximada satisfatória.

Os sistemas que são abordados neste trabalho possuem matrizes de coeficientes simétrica e definida positiva. Desta forma, os algoritmos apropriados para este tipo de sistema são a fatoração de Cholesky (abordagem direta) e o gradiente conjugado (abordagem iterativa). Porém, os métodos diretos perdem sua eficiência à medida que cresce o número de equações, inviabilizando a fatoração de Cholesky já que os problemas a serem resolvidos são de grande porte.

O método do gradiente conjugado [HS52] tem se firmado como o método ideal para solução de sistemas lineares de alta ordem com matriz simétrica e definida positiva. No entanto, suas potencialidades são evidenciadas por meio de uma operação de pré-processamento da matriz chamada de precondicionamento, tornando-o bastante eficiente diante de uma grande variedade de problemas. O método Cholesky controlado gradiente conjugado - CCCG(η), proposto por Campos [Cam95], utiliza um tipo de precondicionamento que controla o preenchimento da matriz, sendo muito eficiente para a solução de grandes sistemas esparsos.

Um procedimento adicional pode ser feito na matriz de entrada para aumentar a eficiência do CCCG(η) - a reordenação. Os algoritmos de reordenação modificam a estrutura da matriz esparsa, ou seja, a distribuição dos elementos não nulos. Com esta modificação, o espaço de armazenamento utilizado durante a execução do algoritmo tende a diminuir, o que pode diminuir também o tempo de execução.

O objetivo deste trabalho é analisar a influência de algoritmos de reordenação de matrizes esparsas na solução de sistemas lineares pelo método CCCG(η). Serão estudados algoritmos mais simples como contagem de colunas e Cuthill-McKee reverso [CM69] e algoritmos mais sofisticados como o mínimo grau e suas variações [ADD96]. Desta forma, pretende-se estabelecer em qual situação um determinado algoritmo é o mais indicado para tornar o CCCG(η) mais eficiente.

A influência da reordenação na convergência de métodos iterativos precondicionados tem sido discutida por um grande número de autores. Vários artigos tratam de problemas simétricos [CT95, DFT92, DM89, Ort91]. Neste contexto, Duff e Meurant [DM89]

fizeram um estudo bastante detalhado sobre o efeito da reordenação no método do gradiente conjugado precondicionado, ou seja, no caso de sistemas com matriz de coeficientes simétrica e definida positiva. Eles testaram 17 diferentes tipos de reordenação, considerando um precondicionador sem preenchimento ou com apenas um nível de preenchimento.

Baseado em extensos experimentos, eles concluíram que o número de iterações requeridas para a convergência usando reordenação é normalmente o mesmo, sendo às vezes maior que a ordenação original. Como o Cuthill-McKee reverso apresentou bons resultados, ele foi considerado como uma boa escolha. Uma observação importante vista em [DM89] é que o número de iterações do gradiente conjugado não está relacionado ao preenchimento descartado na fatoração incompleta (como foi conjecturado por Simon [Sim85]), mas sim à norma residual da matriz.

Em [BSD99], Benzi *et al.* observaram que os benefícios da reordenação dependem de quão próximo a matriz está de ser simétrica, diagonalmente dominante e do tipo de precondicionador utilizado. Eles concluiram que, para matrizes aproximadamente simétricas, a reordenação não tem efeito positivo na convergência de métodos do subespaço de Krylov precondicionados, concordando com Duff e Meurant. Por outro lado, para matrizes não simétricas, eles observaram que reordenações podem de fato fazer uma grande diferença. Permutações que parecem não ter efeito para casos quase simétricos, melhoram freqüentemente a robustez e o desempenho de iterações precondicionadas nos casos não simétricos. Outro fato observado em [BSD99] é que, para alguns casos, a reordenação da matriz de coeficientes antes da fatoração incompleta pode ter o efeito de produzir fatores triangulares estáveis, além de precondicionadores mais efetivos. Geralmente a norma residual não contém informação suficiente para descrever em um modelo quantitativo o comportamento de fatorações incompletas. Em particular, não é possível estabelecer comparações entre fatorações incompletas baseadas apenas na norma. Isto não é surpresa, considerando que para problemas não simétricos não é conhecida uma maneira de predizer a taxa de convergência de métodos iterativos.

Ainda sobre matrizes não simétricas, alguns autores concluíram que as reordenações não são recomendadas para o uso com métodos iterativos precondicionados [DFT92, Lan89, Sim85]. Simon [Sim85] usou o mínimo grau e o *Nested Dissection* em conjunção com a fatoração incompleta para um tipo de problema específico (simulação em reservatório de óleo) e não encontrou nenhuma melhoria considerável sobre a ordenação original. Conclusões similares foram obtidas por Langtangen [Lan89], que aplicou o mínimo grau com a fatoração incompleta em problemas também específicos (formulação Petrov-Galerkin para equações de convecção-difusão). Porém, deve ser mencionado que os problemas considerados por estes autores não exibiram qualquer tipo de instabilidade de fatores triangulares incompletos,

sendo a maioria considerada fácil de resolver nos padrões de hoje. Dutto [Dut93], resolvendo equações diferenciais também específicas (equações de Navier-Stokes), foi a primeira a observar que o mínimo grau e outras reordenações podem ter um efeito positivo na convergência do GMRES [Saa96] com a fatoração incompleta. Isto fica mais consistente com os experimentos reportados por Benzi *et al.* em [BSD99].

Voltando ao contexto de matrizes simétricas, Clift e Tang [CT95] testaram problemas de equações diferenciais parciais originadas de simulações em reservatórios de óleo. Eles verificaram que o Cuthill-McKee reverso e algumas variações funcionaram satisfatoriamente para problemas simétricos e anisotrópicos. Essas variações foram desenvolvidas considerando os coeficientes do grafo além da sua estrutura (reordenações ponderadas). Experimentos numéricos reportados em [BF84] indicaram que um caso especial da reordenação Cuthill-McKee usado com a fatoração incompleta em certos problemas não simétricos definidos em malhas retangulares pode ser superior a ordenação original, mas esta observação não recebeu a atenção que merecia. Talvez isto pode ter sido causado pelo uso de uma terminologia particular ao campo de simulação em reservatórios. Um pequeno parágrafo mencionando que estas reordenações podem ser úteis para métodos iterativos precondicionados pode ser encontrado em [Saa96] e observações similares são encontradas pela literatura. Desta forma, ainda é possível encontrar alguns artigos que consideram as reordenações sem efeito, ou até são ruins, em métodos iterativos precondicionados.

Em [BT99], Bridson e Tang verificaram que algumas reordenações que melhoram a convergência de métodos do subespaço de Krylov podem ter dificuldades particularmente com sistemas originados de funções anisotrópicas. Concordando com [CT95], eles observaram a necessidade de utilizar reordenações ponderadas para este caso, considerando os valores numéricos da matriz. Eles concluíram que a informação estrutural de uma matriz não é o suficiente para observar os benefícios na convergência, já que a anisotropia pode ter um efeito significativo no desempenho, seja em relação ao tempo de precondicionamento ou ao tempo de solução. Enfim, eles finalizaram informando a necessidade de heurísticas ponderadas mais sofisticadas para um maior progresso.

Em outro artigo, Bridson e Tang [BT00b] apresentaram uma análise do efeito de reordenações para fatorização incompleta em termos da estrutura de esparsidade da matriz. Eles observaram que o Cuthill-McKee reverso normalmente apresenta um bom resultado e o mínimo grau torna-se muito competitivo quando algum preenchimento é permitido. Em [BT00a], Benzi e Tuma consideraram a influência de reordenações no desempenho de um precondicionador que utiliza a matriz aproximadamente inversa. Através de seus experimentos, eles observaram que o mínimo grau e o *Nested Dissection* foram muito benéficos. Estes benefícios consistem na redução do tempo e espaço de armazenamento requeridos para

a construção do precondicionador, além da convergência mais rápida em muitos casos de interesse prático. Estes resultados são compatíveis com os encontrados neste trabalho.

Finalmente, tem-se o caso de alguns artigos que consideram outros tipos de reordenações, tais como aquelas motivadas pela computação paralela [EA89, Ort91]. Tais reordenações podem ser úteis na prática, mas elas são fortemente específicas ao tipo do problema e à arquitetura do computador utilizado.

Este trabalho é constituído por 7 capítulos. No que se segue, tem-se a descrição dos sistemas lineares com matriz de coeficientes simétrica e definida positiva, assim como os métodos mais utilizados para sua solução: a fatoração de Cholesky e o gradiente conjugado. No capítulo 3 é apresentado o método utilizado neste projeto para solução de sistemas lineares de grande porte, o Cholesky controlado gradiente conjugado, que é baseado nos dois métodos apresentados anteriormente. O capítulo 4 é uma revisão sobre matrizes esparsas, suas estruturas de armazenamento e os principais algoritmos que utilizam este tipo de estrutura. Já no capítulo 5 são apresentados alguns algoritmos para reordenação de matrizes esparsas que podem melhorar a convergência do CCCG(η): contagem de colunas, Cuthill-McKee reverso e mínimo grau. Os experimentos numéricos são descritos no capítulo 6, onde foram utilizadas matrizes simétricas esparsas disponíveis em bibliotecas digitais. No capítulo 7, tem-se as conclusões e, finalizando, as referências bibliográficas.

Capítulo 2

Sistemas de equações lineares

Neste capítulo serão vistos alguns algoritmos para solução de sistemas de equações lineares. Estes algoritmos podem ser divididos em duas classes: métodos diretos e métodos iterativos. Na primeira classe, a solução exata do sistema é obtida, teoricamente, com um número finito de operações aritméticas. Destacam-se os métodos de eliminação tais como eliminação de Gauss, fatoração LU e fatoração de Cholesky [Cam01].

Já na segunda classe, a solução exata é obtida somente com um número infinito de operações. Partindo de um valor inicial qualquer, o algoritmo converge para o resultado até atingir uma precisão pré-definida. Exemplos clássicos são os métodos iterativos estacionários como Jacobi, Gauss-Seidel e SOR. Exemplos mais sofisticados são dados pelos métodos do subespaço de Krylov [TB97, Saa96] tais como gradiente conjugado, GMRES e Lanczos.

Os métodos iterativos, apesar de dependerem de condições de convergência e apresentarem algumas restrições quanto ao uso, são os métodos mais indicados para resolver sistemas de grande porte. Isto porque os métodos diretos perdem sua eficiência à medida que cresce o número de equações.

Os sistemas utilizados neste trabalho possuem matrizes de coeficientes simétrica e definida positiva, assunto da primeira seção. Desta forma, neste capítulo serão vistos os algoritmos apropriados para este tipo de sistema: a fatoração de Cholesky e o gradiente conjugado.

2.1 Sistema com matriz simétrica definida positiva

Uma parte substancial dos problemas envolvendo sistemas de equações lineares na ciência e engenharia tem a matriz de coeficientes simétrica e definida positiva. Seja um sistema de equações lineares da forma $Ax = b$, onde $A \in \mathbb{R}^{n \times n}$ é uma matriz simétrica definida positiva se

$$x^T Ax > 0 \ , \ \forall x \in \mathbb{R}^n \text{ e } x \neq 0.$$

Desta forma, tem-se que:

- Se uma matriz $A \in \mathbb{R}^{n \times n}$ for simétrica, então possui n autovalores reais e n autovetores ortonormais;
- Uma matriz simétrica é definida positiva se, e somente se, seus autovalores são reais e positivos;
- Uma matriz simétrica é definida positiva se, e somente se, cada uma de suas submatrizes possui determinante positivo;
- Uma matriz simétrica é definida positiva se, e somente se, o processo de eliminação de Gauss pode ser realizado sem permutações de linhas ou colunas e possui todos os elementos pivôs positivos.

As matrizes definidas positivas, quando decompostas, apresentam uma grande estabilidade numérica. O método de Cholesky aplicado a uma matriz simétrica e definida positiva não necessita de pivotação (troca de linhas e/ou colunas) para manter a estabilidade, o que não acontece com matrizes indefinidas.

Outra vantagem do uso deste tipo de matriz está na facilidade de tratamento do problema da reordenação (seção 5). Já que elas podem ser facilmente modeladas como grafos não direcionados (seção 4.3), é possível aumentar o escopo de manipulação com algoritmos e técnicas já existentes para o tratamento de grafos. Além disto, a matriz reordenada é também simétrica e definida positiva para qualquer regra de permutação de linhas ou colunas, permitindo a execução da reordenação sem se preocupar com a estabilidade e antes mesmo que a fatoração numérica comece de fato.

2.2 Método de Cholesky

Para um sistema $Ax = b$, quando $A \in \mathbb{R}^{n\times n}$ for uma matriz simétrica e definida positiva, ela pode ser decomposta em:

$$A = LL^T,$$

onde L é uma matriz triangular inferior.

A decomposição $A = LL^T$ é conhecida como decomposição de Cholesky e sua existência é garantida pelo teorema (2.1) [GL96].

Teorema 2.1. *Se $A \in \mathbb{R}^{n\times n}$ for uma matriz simétrica e definida positiva, então existe uma única matriz triangular $L \in \mathbb{R}^{n\times n}$ com elementos da diagonal positivos tal que $A = LL^T$.*

Para resolver o sistema $Ax = b$, usa-se a matriz na forma decomposta

$$Ax = b \Rightarrow (LL^T)x = b\,.$$

Resolvendo

$$Ly = b \;\text{ e }\; L^Tx = y\,,$$

tem-se que a solução y do sistema triangular inferior é obtida pelas substituições sucessivas (seção 4.2.2) e a solução x do sistema triangular superior pelas substituições retroativas (seção 4.2.3) .

A prova da decomposição de Cholesky é construtiva. Entretanto, uma maneira simples para obter o fator L pode ser feita pela comparação dos elementos da equação $A = LL^T$:

$$\begin{array}{llll}
a_{11} = l_{11}l_{11}, & & & \\
a_{21} = l_{21}l_{11}, & a_{22} = l_{21}l_{21} + l_{22}l_{22}, & & \\
\vdots & \vdots & \ldots & \\
a_{n1} = l_{n1}l_{11}, & a_{n2} = l_{n1}l_{21} + l_{n2}l_{22}, & \ldots & a_{nn} = l_{n1}l_{n1} + l_{n2}l_{n2} + \ldots + l_{nn}l_{nn}.
\end{array}$$

Sendo L uma matriz triangular inferior, é possível rearranjar as equações acima da seguinte forma:

$$\begin{aligned}
l_{jj} &= \sqrt{a_{jj} - \sum_{k=1}^{j-1} {l_{jk}}^2}\,, \\
l_{ij} &= \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik}l_{jk}}{l_{jj}}\,, \text{ para } i > j.
\end{aligned} \tag{2.1}$$

Uma maneira alternativa para resolver o sistema $Ax = b$ pode ser feita decompondo a matriz A em:

$$A = LDL^T,$$

onde L é uma matriz triangular inferior unitária ($l_{ii} = 1, \forall i$) e D é uma matriz diagonal. Desta forma, evita-se o cálculo de raízes quadradas.

Para resolver o sistema $Ax = b$, usa-se a matriz na forma decomposta:

$$Ax = b \Rightarrow (LDL^T)x = b\,.$$

Fazendo

$$Ly = b, \quad Dz = y \ \text{ e } \ L^T x = z,$$

tem-se que a solução y do sistema é obtida pelas substituições sucessivas e a solução x pelas substituições retroativas. As matrizes L e D são calculadas por:

$$\begin{aligned} d_j &= a_{jj} - \sum_{k=1}^{j-1} {l_{jk}}^2 d_k\,, \\ l_{ij} &= \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_k l_{jk}}{d_j}\,, \text{ para } i > j. \end{aligned}$$

Conforme visto na seção anterior, as matrizes definidas positivas possuem alta estabilidade numérica, dispensando o uso de estratégias de pivotação. Já para matrizes indefinidas, o uso de estratégias de pivotação pode ser necessário. Além disto, sabe-se que se a matriz é definida positiva então a decomposição de Cholesky existe. Desta forma, torna-se um atrativo aplicar esta decomposição para determinar se a matriz possui ou não esta propriedade.

Algoritmo de Cholesky

A Figura 2.1 apresenta um algoritmo para fatorar uma matriz simétrica e definida positiva $A = LL^T$ utilizando a decomposição de Cholesky [Cam01]. Os parâmetros de entrada são a ordem n e a matriz A, e os parâmetros de saída são a matriz triangular inferior L, o determinante Det e a condição de erro Erro. Se Erro = 0, significa que a decomposição foi realizada e portanto, a matriz é definida positiva, se Erro = 1, a decomposição não foi realizada porque a matriz não é definida positiva.

A Figura 2.2 apresenta um algoritmo para fatorar uma matriz simétrica em $A = LDL^T$. Este algoritmo pode ser utilizado para fatorar matrizes indefinidas. Os parâmetros de entrada

são a ordem n e a matriz A, e os parâmetros de saída são a matriz triangular inferior L e a matriz D na diagonal de L.

A Figura 2.3 mostra um algoritmo para resolver o sistema $Ax = b$ utilizando a decomposição LDL^T. Os parâmetros de entrada são a ordem n, matriz L, contendo D na diagonal e o vetor de termos independentes b, e o parâmetro de saída é a solução x do sistema.

Algoritmo Cholesky
{ **Objetivo:** Fazer a decomposição LL^T de uma matriz A simétrica e definida positiva }
parâmetros de entrada n, A { ordem e matriz a ser decomposta }
parâmetros de saída L, Det, $Erro$ { fator, determinante e condição de erro }
  $Det \leftarrow 1$
  para $j \leftarrow 1$ **até** n **faça**
    $Soma \leftarrow 0$
    para $k \leftarrow 1$ **até** $j - 1$ **faça**
      $Soma \leftarrow Soma + L(j,k)^2$
    fim para
    $t \leftarrow A(j,j) - Soma$; $Det \leftarrow Det * t$
    $Erro \leftarrow t \leq 0$ { variável lógica: se verdadeiro tem erro e se falso não há erro }
    se $Erro$ **então**
      escreva "a matriz não é definida positiva"; **abandone**
    senão
      $L(j,j) \leftarrow$ raiz(t); $r \leftarrow 1/L(j,j)$
    fim se
    para $i \leftarrow j + 1$ **até** n **faça**
      $Soma \leftarrow 0$
      para $k \leftarrow 1$ **até** $j - 1$ **faça**
        $Soma \leftarrow Soma + L(i,k) * L(j,k)$
      fim para
      $L(i,j) \leftarrow (A(i,j) - Soma) * r$
    fim para
  fim para
fim algoritmo

Figura 2.1: Decomposição de Cholesky de uma matriz A simétrica e definida positiva.

Algoritmo LDL^T
{ **Objetivo:** Fazer a decomposição LDL^T de uma matriz A simétrica }
parâmetros de entrada $n,\ A$ { ordem e matriz a ser decomposta }
parâmetros de saída L { fator L contendo D na diagonal }

```
para j ← 1 até n faça
    para i ← j até n faça
        Soma1 ← 0
        Soma2 ← 0
        para k ← 1 até j − 1 faça
            Soma1 ← Soma1 + L(j, k)^2 * d(k)
            Soma2 ← Soma2 + L(i, k) * L(j, k) * d(k)
        fim para
        d(j) ← A(j, j) − Soma1
        L(i, j) ← (A(i, j) − Soma2)/d(j)
        L(j, j) ← d(j)
    fim para
fim para
```

fim algoritmo

Figura 2.2: Decomposição LDL^T de uma matriz A simétrica.

Algoritmo Sub_Sucessivas_Retroativas
{ **Objetivo:** Resolver os sistemas $Ly = b$, $Dz = y$ e $L^T x = z$ }
parâmetros de entrada $n,\ L,\ b$
{ ordem, matriz L contendo D na diagonal, vetor de termos independentes }
parâmetros de saída x { solução do sistema }
{ Substituições Sucessivas }
$y(1) \leftarrow b(1)$
$z(1) \leftarrow y(1)/L(1,1)$
para $i \leftarrow 2$ **até** n **faça**
$soma \leftarrow 0$
para $j \leftarrow 1$ **até** $i - 1$ **faça**
$soma \leftarrow soma + L(i,j) * y(j)$
fim para
$y(i) \leftarrow b(i) - soma$
$z(i) \leftarrow y(i)/L(i,i)$
fim para
{ Substituições Retroativas }
$x(n) \leftarrow y(n)$
para $i \leftarrow n - 1$ **até** 1 **passo** -1 **faça**
$soma \leftarrow 0$
para $j \leftarrow i + 1$ **até** n **faça**
$soma \leftarrow soma + L(j,i) * x(j)$
fim para
$x(i) \leftarrow z(i) - soma$
fim para
fim algoritmo

Figura 2.3: Solução do sistema utilizando a decomposição LDL^T.

2.3 Método do gradiente conjugado

O método do gradiente conjugado (CG: *conjugate gradient*) pertence a um conjunto de métodos de otimização do qual fazem parte o método do gradiente e o método das direções conjugadas [ML86], entre outros. Eles são modelados como uma minimização irrestrita, ou seja:

$$\begin{cases} min \;\; f(x) \\ x \in \mathbb{R} \end{cases}$$

onde $f(x)$ é uma função convexa não linear, caracterizando um problema de programação não linear (PPNL).

Na minimização de uma função convexa, o ponto de mínimo é aquele onde o vetor gradiente se anula, ou seja:

$$\nabla f(x) = 0 \, .$$

Para resolver um sistema linear da forma $Ax = b$ usando minimização, é necessário encontrar uma função cujo gradiente seja:

$$\nabla f(x) = Ax - b \, ,$$

desta forma, tem-se como resultado:

$$\begin{aligned} \nabla f(x) &= 0 \, , \\ Ax - b &= 0 \, , \\ Ax &= b \, , \end{aligned}$$

que é a solução do sistema linear.

Por definição [She94], a função utilizada para o problema com matriz A simétrica e definida positiva é:

$$f(x) = \frac{1}{2} x^T A x - b^T x \, . \tag{2.2}$$

A Figura 2.4 mostra um exemplo da forma quadrática desta função e de suas curvas de nível para as matrizes:

$$A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix} \;\; e \;\; b = \begin{bmatrix} 2 \\ -8 \end{bmatrix} . \tag{2.3}$$

Derivando parcialmente a função:

$$\nabla f(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b \, .$$

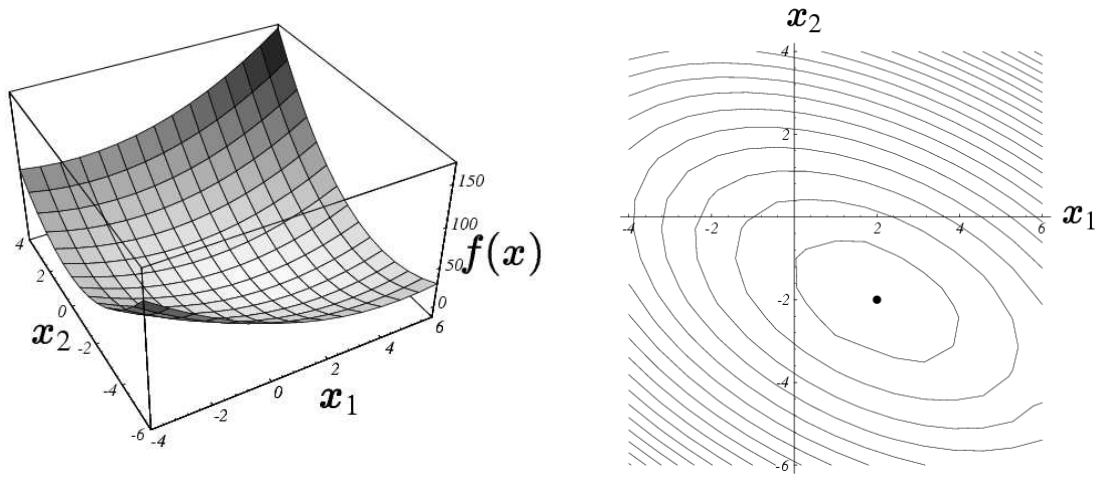


Figura 2.4: Forma quadrática e suas curvas de nível.

Como a matriz A é simétrica, o gradiente fica de acordo com o esperado:

$$\nabla f(x) = Ax - b\,.$$

Assim, é possível modelar a solução de um sistema lineares como uma minimização de (2.2).

Definido o problema não linear, o método de minimização poderá ser descrito. Antes do gradiente conjugado, serão vistos os métodos do gradiente e das direções conjugadas, já que eles fornecem toda uma base teórica para o CG. Existem várias formas de se descrever estes métodos. Será apresentada aqui uma abordagem intuitiva e com boas interpretações geométricas, totalmente baseada em [She94].

2.3.1 Método do gradiente

Este método, também chamado de máximo declive (*steepest descent*), é um dos mais conhecidos para minimização de uma função de várias variáveis. Além de sua simplicidade e facilidade de análise, ele é importante por servir de base para outros métodos mais eficientes resultantes da modificação no seu algoritmo.

De um modo geral, os algoritmos de minimização irrestrita partem de um ponto inicial x_0 qualquer e, a cada iteração i, movem um passo α_i em uma direção d_i buscando atingir um ponto estacionário que satisfaça a condição de parada. O método do gradiente caminha na direção oposta ao vetor gradiente para convergir para o ponto de mínimo:

$$-\nabla f(x_i) = -Ax_i + b.$$

Pode-se ver que esta direção é a mesma do resíduo da solução na iteração i:

$$\begin{aligned} r_i &= b - Ax_i \,, \\ r_i &= -\nabla f(x_i) \,. \end{aligned}$$

Assim, a cada iteração deste método, a solução é dada por:

$$x_{i+1} = x_i + \alpha_i r_i \,.$$

Para que o tamanho do passo α_i em direção ao resíduo seja definido, é necessário minimizar a função criada para interseção da quadrática com a direção de busca $-\nabla f(x_i)$ no ponto x_{i+1}. Na Figura 2.5, tem-se o caminhamento realizado pelo exemplo da Figura 2.4 partindo do ponto $[-2, -2]^T$.

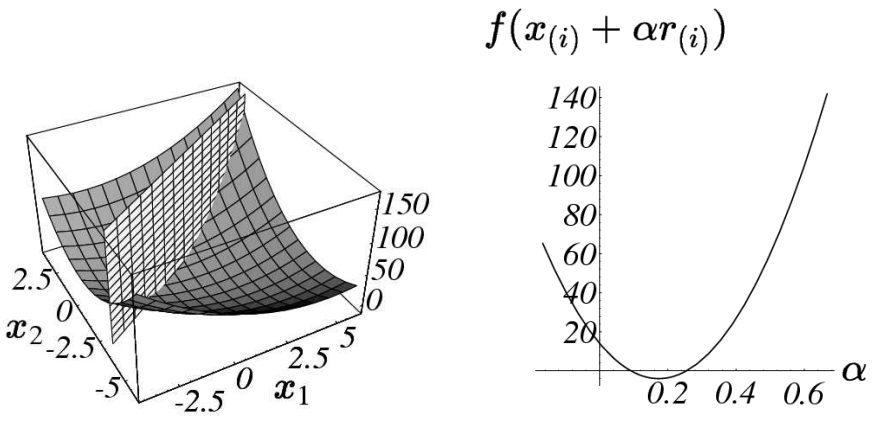


Figura 2.5: Interseção das superfícies.

Sabe-se que o ponto de mínimo é o ponto onde a derivada é nula, ou seja:

$$\begin{aligned}
\frac{\partial f(x_{i+1})}{\partial \alpha} &= 0, \\
\frac{\partial f(x_{i+1})^T}{\partial x} \cdot \frac{\partial f(x_i + \alpha_i r_i)}{\partial \alpha} &= 0, \\
r_{i+1}^T r_i &= 0, \\
(b - Ax_{i+1})^T r_i &= 0, \\
(b - A(x_i + \alpha_i r_i))^T r_i &= 0, \\
(b - Ax_i)^T r_i - \alpha_i (Ar_i)^T r_i &= 0, \\
(b - Ax_i)^T r_i &= \alpha_i (Ar_i)^T r_i, \\
r_i^T r_i &= \alpha_i r_i^T A r_i, \\
\alpha_i &= \frac{r_i^T r_i}{r_i^T A r_i}.
\end{aligned}$$

Desta forma, enquanto uma determinada condição de parada não for satisfeita, este método executa as seguintes operações:

$$\begin{aligned}
r_i &= b - Ax_i \,, \\
\alpha_i &= \frac{r_i^T r_i}{r_i^T A r_i} \,, \\
x_{i+1} &= x_i + \alpha_i r_i \,.
\end{aligned}$$

2.3.2 Método das direções conjugadas

O método do gradiente é bastante simples em termos computacionais, mas pode ter uma convergência lenta. Ele utiliza pouca informação, somente o cálculo do gradiente, usando direções de busca ortogonais, já que os resíduos são ortogonais entre si.

O método das direções conjugadas minimiza a função quadrática de ordem n em no máximo n iterações. Para isto, ele utiliza direções de busca conjugadas e não mais ortogonais. Por definição, dada uma matriz A simétrica, dois vetores d_i e d_j são conjugados se:

$$d_i^T A d_j = 0 \,.$$

Sendo $A_{n \times n}$, esta definição pode ser estendida ao subespaço $\mathbb{R}^n$, onde existem n vetores conjugados:

$$d_i^T A d_j = 0, \text{ para } 0 < i, j < n, i \neq j.$$

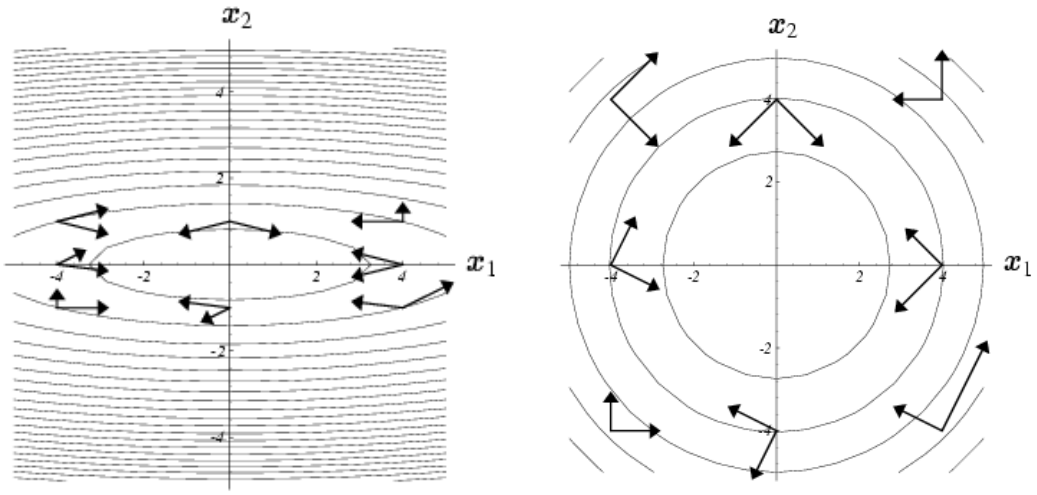


Figura 2.6: Direções conjugadas × direções ortogonais.

Graficamente, as direções conjugadas são projeções das direções ortogonais quando as curvas de nível são elípticas (Figura 2.6).

Para gerar a direção conjugada, é usado o processo de conjugação de Gram-Schmidt [She94]. Este método se baseia em um conjunto de vetores $u_0, u_1, ..., u_{n-1}$ linearmente independentes para gerar as direções conjugadas da seguinte forma:

$$\beta_{ik} = -\frac{u_i^T A d_k}{d_k^T A d_k}, \text{ para } i > k, \tag{2.4}$$

$$d_i = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_k. \tag{2.5}$$

O processo de encontrar o tamanho do passo na direção escolhida é o mesmo usado no método do gradiente. Assim, é feita uma minimização da função criada pela interseção da quadrática com a direção de busca no ponto x_{i+1}:

$$
\begin{aligned}
\frac{\partial f(x_{i+1})}{\partial \alpha} &= 0, \\
\frac{\partial f(x_{i+1})^T}{\partial x} \cdot \frac{\partial f(x_i + \alpha_i d_i)}{\partial \alpha} &= 0, \\
r_{i+1}^T d_i &= 0, \\
(b - Ax_{i+1})^T d_i &= 0, \\
(b - A(x_i + \alpha_i d_i))^T d_i &= 0, \\
(b - Ax_i)^T d_i - \alpha_i (Ad_i)^T d_i &= 0, \\
(b - Ax_i)^T d_i &= \alpha_i (Ad_i)^T d_i, \\
r_i^T d_i &= \alpha_i (Ad_i)^T d_i, \\
r_i^T d_i &= \alpha_i d_i^T A d_i, \\
d_i^T r_i &= \alpha_i d_i^T A d_i, \\
\alpha_i &= \frac{d_i^T r_i}{d_i^T A d_i}.
\end{aligned} \tag{2.6}
$$

Como resultado, gerando um conjunto de i direções conjugadas e efetuando uma minimização em cada direção d_i a partir de x_0, é possível atingir o ponto de mínimo da quadrática caminhando da seguinte maneira:

$$x_{i+1} = x_i + \alpha_i d_i \, . \tag{2.7}$$

A dificuldade em usar Gram-Schmidt está na necessidade de armazenar todas as $i - 1$ direções já geradas para construir a próxima. Assim, $O(n^3)$ operações são requeridas para se gerar o conjunto total, onde n é a ordem do sistema. Por isto, o método das direções conjugadas não é muito eficiente para minimização de funções.

2.3.3 Método do gradiente conjugado

O método do gradiente conjugado foi proposto por Hestenes e Stiefel em 1952 [HS52]. É um dos métodos mais eficientes para solução de sistemas lineares de grande porte com matriz de coeficientes esparsa, simétrica e definida positiva. Assim como o método das direções conjugadas, ele caminha para a solução efetuando uma minimização em cada direção d_i ($x_{i+1} = x_i + \alpha_i d_i$).

A cada iteração, este método gera uma direção de busca formada pela combinação linear do resíduo e de direções conjugadas. Porém, estas direções conjugadas são construídas pela

conjugação dos resíduos, o que simplifica o processo de conjugação de Gram-Schmidt devido às seguintes propriedades:

$$\begin{cases} r_i^T r_j = 0, & \text{para } i \neq j \ , \\ d_i^T r_j = 0, & \text{para } i < j \ . \end{cases} \tag{2.8}$$

Além disto, tomando-se o produto interno de (2.5) por r_j:

$$d_i^T r_j = u_i^T r_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T r_j \ ,$$

como, por (2.4), $i > k$ e fazendo $i = j$, por (2.8) tem-se:

$$d_i^T r_i = u_i^T r_i \ . \tag{2.9}$$

Desta forma, não é mais necessário guardar todas as direções conjugadas já geradas, sendo isto um dos fatores que torna o CG tão eficiente em relação ao método das direções conjugadas. Para demonstrar esta proposição, basta reescrever o resíduo multiplicando cada termo de (2.7) por -A e adicionando b:

$$\begin{aligned} x_{j+1} &= x_j + \alpha_j d_j \ , \\ -Ax_{j+1} + b &= (-Ax_j - \alpha_j A d_j) + b \ , \\ r_{j+1} &= r_j - \alpha_j A d_j \ . \end{aligned}$$

Multiplicando ambos os lados por r_i^T, tem-se:

$$\begin{aligned} r_i^T r_{j+1} &= r_i^T r_j - \alpha_j r_i^T A d_j \ , \\ r_i^T A d_j &= \frac{r_i^T r_j - r_i^T r_{j+1}}{\alpha_j} \ , \end{aligned}$$

por (2.8):

$$r_i^T A d_j = \begin{cases} \dfrac{r_i^T r_i}{\alpha_i} \ , & \text{para } i = j, \\ \dfrac{-r_i^T r_i}{\alpha_{i-1}}, & \text{para } i = j+1, \\ 0 \ , & \text{nos demais casos.} \end{cases}$$

Em (2.4), considerando $u_i = r_i$ (resíduo linearmente independente), tem-se:

$$\begin{aligned} \beta_{ij} &= -\frac{u_i^T A d_j}{d_j^T A d_j} \ , \ \text{para } i > j, \\ \beta_{ij} &= \begin{cases} \dfrac{-r_i^T r_i}{\alpha_{i-1} d_{i-1}^T A d_{i-1}} \ , & \text{para } i = j+1, \\ 0 \ , & \text{para } i > j+1. \end{cases} \end{aligned}$$

Com isto, muitos termos de β_{ij} se anulam para $i > j+1$, sendo necessário armazenar apenas os valores da última iteração. Simplificando $\beta_i = \beta_{i,i-1}$ e usando (2.6), tem-se:

$$\begin{aligned} \beta_i &= -\frac{r_i^T r_i}{d_{i-1}^T r_{i-1}}, \quad \text{para } i = j+1, \\ \beta_i &= -\frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}, \quad \text{por (2.9)}. \end{aligned}$$

Assim, a direção de busca será dada por:

$$\begin{aligned} r_{i+1} &= r_i - \alpha_i A d_i \,, \\ \beta_{i+1} &= -\frac{r_{i+1}^T r_{i+1}}{r_i^T r_i} \,, \\ d_{i+1} &= r_{i+1} + \beta_{i+1} d_i \,. \end{aligned}$$

Em relação ao tamanho do passo, dado que $u_i = r_i$, por (2.6) e (2.9), tem-se:

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i} \,.$$

Algoritmo do gradiente conjugado

A Figura 2.7 apresenta um algoritmo para calcular a solução de um sistema $Ax = b$ pelo método do gradiente conjugado. Os parâmetros de entrada são matriz A, o vetor de termos independentes b, uma solução inicial x^0, o número máximo de iterações Kmax e a tolerância toler. Os parâmetros de saída são a solução do sistema x e a condição de erro Erro para verificar se houve convergência. Se Erro $= 0$, significa que houve convergência e se Erro $= 1$ que não houve.

Algoritmo *CG*
{ **Objetivo:** Resolver o sistema $Ax = b$ pelo método gradiente conjugado }
parâmetros de entrada $A,\ b,\ x_0,\ kmax,\ toler$
{ matriz dos coeficientes, vetor de termos independentes, solução inicial }
{ número máximo de iterações, tolerância máxima admitida }
parâmetros de saída $x,\ Erro$ { solução do sistema, condição de erro }
$r_0 \leftarrow b - A * x_0$
$k \leftarrow 0;\ d_0 \leftarrow 0;\ \rho_0 \leftarrow 1$
repita
$k \leftarrow k + 1$
$\rho_k \leftarrow (r_{k-1})^T r_{k-1}$
$\beta_k \leftarrow \rho_k / \rho_{k-1}$
$d_k \leftarrow r_{k-1} + \beta_k d_{k-1}$
$\alpha_k \leftarrow \rho_k / (d_k)^T A d_k$
$r_k \leftarrow r_{k-1} - \alpha_k A d_k$
$x_k \leftarrow x_{k-1} + \alpha_k d_k$ { caminhamento }
escreva $k,\ x_k,\ r_k$
se $\|r_k\| / \|r_0\| < toler$ **ou** $k = kmax$
então interrompa
fim se
fim repita
$Erro \leftarrow \|r_k\| / \|r_0\| \geq toler$ { variável lógica }
fim algoritmo

Figura 2.7: Algoritmo do gradiente conjugado.

Capítulo 3

Cholesky controlado gradiente conjugado

Já se tem conhecimento e experiência com os fundamentos teóricos relativos ao uso de técnicas para o aumento de eficiência do gradiente conjugado. O precondicionamento é um procedimento que pode ser aplicado a métodos iterativos para melhorar suas características de convergência, principalmente, em problemas que possuem a matriz dos coeficientes com autovalores muito dispersos. Precondicionar um sistema linear é fazer com que a matriz dos coeficientes apresente as condições desejadas para que o método de solução aplicado seja eficiente.

Neste capítulo serão vistos alguns precondicionadores para o gradiente conjugado, tais como o escalonamento diagonal, a fatoração de Cholesky incompleta e a fatoração de Cholesky controlada. Este último serve como base para o método de solução de sistemas utilizado neste trabalho, o Cholesky controlado gradiente conjugado [CR95b].

3.1 Precondicionadores para o gradiente conjugado

A convergência do método do gradiente conjugado depende do formato das curvas de nível da função (2.2) a ser minimizada, ou seja, dos autovalores da matriz de coeficientes. Autovalores de mesma magnitude tornam a superfície esférica, acelerando a convergência. Quando estas curvas de nível são elípticas, a convergência pode se tornar lenta (caminhamento em zigue-zague), tornando-se necessária a utilização de técnicas para melhorar esta convergência. A utilização de precondicionadores é determinante para a eficiência do CG,

pois tendem a remover os autovalores pequenos e/ou grandes e melhorar a sua distribuição de tal modo que eles fiquem próximos da unidade ou em grupos de mesmo valor.

Uma forma simples de realizar o precondicionamento é fazer uma pré-multiplicação da inversa de uma matriz auxiliar M, denominada precondicionadora, a um sistema de equações lineares $Ax = b$, da seguinte maneira:

$$M^{-1}Ax = M^{-1}b\,.$$

Para que este precondicionamento seja eficiente, suas operações devem ser de baixo custo, a matriz precondicionadora M deve ser facilmente construída, ter esparsidade próxima de A e os autovalores de $M^{-1}A$ devem estar próximos a unidade ou agrupados.

Um sistema pode ser precondicionado implicitamente se apenas o vetor resíduo é modificado ao resolvê-lo, ou explicitamente, se as modificações ocorrem na matriz dos coeficientes. Neste trabalho nos restringiremos a sistemas precondicionados explicitamente.

Seja $Ax = b$ um sistema de equações lineares, onde A é simétrica e definida positiva. Este sistema pode ser precondicionado por:

$$(M^{-1}AM^{-T})y = M^{-1}b\,,$$

onde $y = M^T x$. Para resolvê-lo, será utilizado o algoritmo da Figura 3.1.

Existem várias formas para construir um precondicionador M, e uma boa escolha pode ter um efeito destacável na convergência do método. A seguir, serão vistos alguns precondicionadores utilizados para resolver sistemas com o método do gradiente conjugado.

3.1.1 Escalonamento diagonal

Um dos precondicionadores mais simples utilizados para o gradiente conjugado é o escalonamento diagonal (*diagonal scaling*). Em 1955, Forsythe e Strauss [FS55] propuseram uma maneira muito simples para precondicionar um sistema de equações lineares utilizando uma matriz diagonal D:

$$(DAD)(D^{-1}x) = Db\,.$$

Para matrizes simétricas, quando $D = \mathrm{diag}(A)$, o sistema pode ser precondicionado por:

$$(D^{-1/2}AD^{-1/2})y = D^{-1/2}b\,,$$

onde $(D^{-1/2}AD^{-1/2})$ é uma matriz com diagonal unitária e $y = D^{1/2}x$.

Algoritmo Precond_CG
{ **Objetivo:** Resolver o sistema de equações lineares $Ax = b$ pelo método }
{ gradiente conjugado precondicionado explicitamente }
parâmetros de entrada $A,\ M,\ b,\ y_0,\ kmax,\ toler$
{ matriz dos coeficientes, matriz precondicionadora, vetor de termos independentes, }
{ solução inicial, número máximo de iterações, tolerância máxima admitida }
parâmetros de saída $x,\ \ Erro$ { solução do sistema, condição de erro }
$r_0 \leftarrow M^{-1}(b - AM^{-T}y_0)$
$k \leftarrow 0;\ d_0 \leftarrow 0;\ \rho_0 \leftarrow 1$
repita
$k \leftarrow k + 1$
$\rho_k \leftarrow (r_{k-1})^T r_{k-1}$
$\beta_k \leftarrow \rho_k/\rho_{k-1}$
$d_k \leftarrow r_{k-1} + \beta_k d_{k-1}$
$\alpha_k \leftarrow \rho_k/((d_k)^T M^{-1} A M^{-T} d_k)$
$y_k \leftarrow y_{k-1} + \alpha_k d_k$
$r_k \leftarrow r_{k-1} - \alpha_k M^{-1} A M^{-T} d_k$
escreva $k,\ y_k,\ r_k$
se $\|r_k\|/\|r_0\| < toler$ **ou** $k = kmax$
então interrompa
fim se
fim repita
$x_k \leftarrow M^{-T} y_k$
$Erro \leftarrow \|r_k\|/\|r_0\| \geq toler$ { variável lógica }
fim algoritmo

Figura 3.1: Algoritmo do gradiente conjugado precondicionado explicitamente.

3.1.2 Fatoração incompleta de Cholesky

Da mesma forma que métodos iterativos podem ser usados para refinar a solução obtida por métodos diretos, também os métodos diretos podem ser utilizados para acelerar a convergência de métodos iterativos. Um exemplo disto é o uso da fatoração de Cholesky como base para a construção de precondicionadores para o método do gradiente conjugado.

A fatoração incompleta de Cholesky (ICF: *incomplete Cholesky factorization*), é uma das estratégias mais importantes de obter precondicionadores para sistemas de equações lineares com matrizes de grande porte, esparsas, simétricas e definidas positivas. O uso da fatoração

de Cholesky para resolver estes sistemas pode não ser possível, por limitações de memória. Em geral, o fator L, obtido da decomposição de Cholesky, será mais denso que a matriz dos coeficientes, devido ao preenchimento (*fill-in*) ocorrido durante a fatoração. Uma das formas de se evitar este problema é o uso da fatoração incompleta, ou seja, aplica-se a decomposição de Cholesky ignorando alguns elementos para reduzir o preenchimento.

Seja $Ax = b$ um sistema de equações lineares. Uma forma fácil e eficiente para obter um precondicionador M para este sistema é determinar uma matriz triangular inferior $\tilde{L}$ que tenha os elementos não nulos nas mesmas posições dos elementos não nulos de A, tal que, $M = \tilde{L}\tilde{L}^T$. Desta forma, elimina-se alguns dos elementos que causam preenchimento, fazendo com que o espaço requerido para armazenar $\tilde{L}$ seja o mesmo da matriz original A.

Em 1968, Stone [Sto68] propôs um procedimento implícito avançado (SIP: *strongly implicit procedure*) que usa a iteração:

$$Mx_{k+1} = Mx_k + \sigma_i r_i \, ,$$

onde:
$M = A + R$, sendo R a matriz resto,
$r_i = b - Ax_i$,
σ_i é uma parâmetro de iteração.

Ao se decompor a matriz dos coeficientes modificada em $M = \tilde{L}\tilde{L}^T$, onde $\tilde{L}$ têm elementos não nulos somente nas posições correspondentes aos elementos não nulos de A, a matriz M pode ser utilizada como precondicionador para o sistema $Ax = b$.

Foram desenvolvidos vários precondicionadores baseados em fatorações incompletas. Em 1977, Meijerink e van der Vorst [MdV77] propuseram uma classe de matrizes precondicionadoras com níveis de preenchimento denominadas decomposição LU incompleta. Munksgaard [Mun80] propôs um esquema de retirada por tolerância, no qual, elimina-se somente os elementos de $\tilde{L}$ que são menores que um valor pré-estabelecido. Jones e Plassmann [JP95] propuseram um precondicionador (IICD: *improved incomplete Cholesky decomposition*) onde o fator $\tilde{L}$ possui um número fixo de elementos não nulos por linha ou coluna. Este número é igual à quantidade de elementos não nulos de cada linha ou coluna de A, porém os elementos escolhidos são os maiores em módulo.

3.1.3 Fatoração controlada de Cholesky

A fatoração controlada de Cholesky (CCF: *controlled Cholesky factorization*), foi proposta por Campos [CR95b] em 1995 também como uma variação da fatoração incompleta

de Cholesky. O objetivo é construir uma matriz precondicionadora que possua os autovalores agrupados e próximos à unidade, de forma a acelerar a convergência do método gradiente conjugado.

Seja a matriz $A \in \mathbb{R}^{n\times n}$ fatorada em:

$$A = LL^T = \tilde{L}\tilde{L}^T + R, \tag{3.1}$$

onde:
L = fator obtido pela fatoração completa de Cholesky,
$\tilde{L}$ = fator obtido pela fatoração incompleta de Cholesky,
R = matriz resto.

Usando $\tilde{L}$ como uma matriz precondicionadora para A, obtém-se:

$$\tilde{L}^{-1}A\tilde{L}^{-T} = \tilde{L}^{-1}LL^T\tilde{L}^{-T} = (\tilde{L}^{-1}L)(\tilde{L}^{-1}L)^T.$$

Definindo, $E = L - \tilde{L}$ e substituindo na igualdade acima, tem-se:

$$\tilde{L}^{-1}A\tilde{L}^{-T} = (I + \tilde{L}^{-1}E)(I + \tilde{L}^{-1}E)^T.$$

Observe que, se $\tilde{L} \to L$ então $E \to 0$ e $\tilde{L}^{-1}A\tilde{L}^{-T} \to I$.

Duff e Meurant [DM89] mostraram que o número de iterações gastas pelo método gradiente conjugado está diretamente relacionado com a norma de R. De acordo com (3.1), R pode ser expresso por:

$$\begin{aligned}
R &= LL^T - \tilde{L}\tilde{L}^T, \\
R &= LL^T - L\tilde{L}^T + L\tilde{L}^T - \tilde{L}\tilde{L}^T, \\
R &= L(L^T - \tilde{L}^T) + (L - \tilde{L})\tilde{L}^T, \\
R &= L(L - \tilde{L})^T + (L - \tilde{L})\tilde{L}^T, \\
R &= LE^T + E\tilde{L}^T.
\end{aligned}$$

O precondicionador CCF foi construído baseado na minimização da norma de Frobenius de E, visto que, quando $\|E\| \to 0 \Rightarrow \|R\| \to 0$.

Considere o seguinte problema:

$$\min \|E\|^2_F = \sum_{j=1}^{n} c_j = \sum_{i=1}^{n} |l_{ij} - \tilde{l_{ij}}|^2 . \tag{3.2}$$

Dividindo c_j em dois somatórios, tem-se:

$$\sum_{j=1}^{n} c_j = \sum_{k=1}^{m_j+\eta} |l_{i_kj} - \tilde{l}_{i_kj}|^2 + \sum_{k=m_j+\eta+1}^{n} |l_{i_kj}|^2 ,$$

onde:
m_j = número de elementos não nulos abaixo da diagonal na j-ésima coluna da matriz A;
η = número extra de elementos não nulos permitido por coluna;
n = ordem da matriz.

O problema (3.2) pode ser resolvido, baseando-se na seguinte heurística:

- Aumentando o fator η, permitindo maior preenchimento. Desta forma, c_j decresce, pois a primeira soma contém mais elementos. Isto é o mesmo que ocorre quando níveis de preenchimento são usados nas fatorações incompletas.
- Escolhendo os maiores $m_j + \eta$ elementos de $\tilde{L}$, em valor absoluto, para η fixo. Desta forma os maiores elementos estão na primeira soma e os menores na segunda, produzindo um fator $\tilde{L}$ ótimo, para uma determinada quantidade de armazenamento. Este esquema de minimização é semelhante ao esquema de retirada por tolerância.

Algoritmo da fatoração controlada de Cholesky

A Figura 3.2 é uma versão aprimorada do algoritmo CCF dado em [CB98]. Para cada coluna j em $\tilde{L}$, tem-se apenas $m_j + \eta$ elementos não nulos. Inicialmente, a coluna j é calculada e armazenada, depois selecionam-se os k maiores elementos em valor absoluto e descarta-se o restante. Os parâmetros de entrada para o algoritmo são a matriz A, a ordem da matriz n e o fator para controle de preenchimento η. Tem-se como parâmetro de saída, a matriz precondicionadora L.

Algoritmo CCF
{ **Objetivo:** Computar a fatoração controlada de Cholesky $\tilde{L}D\tilde{L}^T$ }
parâmetros de entrada $n,\ nza,\ V,\ \eta,\ \epsilon_m$
{ ordem, número de elementos não nulos, matriz a ser fatorada, }
{ controle de preenchimento e precisão da máquina }
parâmetros de saída $L,\ Erro$
{ matriz $\tilde{L}$ com diagonal contendo a diagonal de D e condição de erro }
$R \leftarrow$ diagonal(V)
$V \leftarrow R^{-1/2}\,VR^{-1/2}$
$\sigma \leftarrow 0\ ;\ \ t \leftarrow 0$
repita
Erro $\leftarrow$ falso
para $j \leftarrow 1$ **até** n **faça**
para $k \leftarrow 1$ **até** $j-1$ **faça** $\ t(k) \leftarrow L(j,k) * L(k,k)$ **fim para**
para $i \leftarrow j+1$ **até** n **faça** $\ w(i) \leftarrow V(i,j) - \sum_{k=1}^{j-1} L(i,k) * t(k)$ **fim para**
$m_j \leftarrow$ número de elementos não nulos na coluna j de V
se $\eta \geq 0$ **então**
$M \leftarrow max(m_j + \eta, 0)$
senão
$M \leftarrow max\left(m_j * \left(\frac{n*\eta}{nza-n} + 1\right), 0\right)$
fim se
$w \leftarrow$ selecionar M maiores elementos de w, em módulo, (eliminar demais)
$L(j,j) \leftarrow 1 - \sum_{k=1}^{j-1} L(j,k) * t(k) + \sigma$
se $L(j,j) \leq \epsilon_m$ **então**
$\sigma \leftarrow 5 \cdot 10^{-4} * 2^t\ ;\ \ t \leftarrow t+1$
Erro $\leftarrow$ verdadeiro
`interrompa para-faça (j)`
fim se
para $k \leftarrow 1$ **até** M **faça**
$i(k) \leftarrow$ índice da k-ésima linha selecionada de w
$L(i(k),j) \leftarrow w(i(k))/L(j,j)$
fim para
fim para
até não Erro **ou** $t > 15$
fim algoritmo

Figura 3.2: Fatoração Controlada de Cholesky.

3.2 CCCG(η)

O gradiente conjugado precondicionado pela fatoração controlada de Cholesky resulta no método de solução de sistemas utilizado neste projeto, o Cholesky controlado gradiente conjugado (CCCG: *controlled Cholesky conjugate gradient*).

As principais vantagens do uso do precondicionador CCF podem ser descritas como:

1. Escolha dos elementos por valor

   O padrão de esparsidade da matriz original não é considerado pelo CCF. As posições dos elementos não nulos escolhidas pelo precondicionador podem ou não cobrir aquelas da matriz original, já que a seleção é feita por valor e não por posição. O CCF dá bons precondicionadores utilizando pouca memória em comparação com outros que conservam o padrão de esparsidade da matriz original.

2. Generalização do IICD (*improved incomplete Cholesky decomposition*)

   Jones e Plassmann [JP95] desenvolveram o precondicionador IICD que permite um número fixo de elementos não nulos em cada linha ou coluna do precondicionador. Este número é dado pela quantidade destes elementos na matriz original A. Porém, somente os maiores elementos em magnitude são mantidos e o padrão de esparsidade de A é ignorado. Desta forma, pode-se considerar o CCF como uma generalização do método IICD, já que é permitido preenchimento.

3. Utilização do incremento exponencial

   Manteuffel [Man80] provou que se A for simétrica definida positiva, então há uma constante $\sigma > 0$ tal que exista uma fatoração incompleta de $A + \sigma I$. Assim, um valor é acrescido aos elementos da diagonal durante a construção do precondicionador, obtendo uma menor perturbação das diagonais e fazendo com que A permaneça definida positiva. No caso do CCF, este valor é dado por um incremento exponencial igual a $\sigma_i = 5.10^{-4} \times 2^i$.

4. Precondicionador versátil

   O CCF produz uma matriz precondicionadora M versátil para $(M^{-1}AM^{-T})y = M^{-1}b$. Quando $\eta = -n$, é feito apenas o escalonamento diagonal. Já quando $\eta = n$, é feita a fatoração completa de Cholesky. Desta forma, η pode ser escolhido no intervalo $[-n, n]$ tal que M necessite de mais ou menos memória que A. Em outras palavras, preenchimento ($\eta > 0$) e retirada ($\eta < 0$) são permitidos pelo CCF.

5. Armazenamento previsível

   Seja n a ordem da matriz A e m o número de seus elementos não nulos. Considerando o número de bytes presentes em um inteiro e em um real iguais a 4 e 8, respectivamente, a quantidade de memória utilizada pelo IICD(0) é $20m + 4n + 4$ e pelo CCF(η) é $28m + (16\eta + 8)n + 8$. Embora o CCF(η) utilize mais memória que o IICD(0), ele possui uma estrutura de dados mais versátil que permite que o número de elementos não nulos de $\tilde{L}$ seja variável.

Na implementação do método de solução CCCG(η) construído a partir deste precondicionador, destacam-se alguns tópicos que serão descritos nas seções a seguir.

Verificação do precondicionador

O método CCCG(η) pode ter a opção de verificação do precondicionador. Dependendo do valor de η escolhido, o precondicionador criado pode não ser de boa qualidade, ficando com autovalores muito grandes. Daí a importância da verificação da qualidade do precondicionador.

Os resíduos de CG podem ser escritos em termos do resíduo inicial $r_k = \phi_k(A)r_0$, onde $\phi_k(A)$ é o erro polinomial de CG com grau k, tal que $\phi_k(0) = 1$,

$$\phi_k(x) = \prod_{j=1}^{k} \frac{\tau_j - x}{\tau_j} \,. \tag{3.3}$$

Este polinômio é expresso em termos de suas raízes τ_j (valores de Ritz) que são os autovalores de uma matriz simétrica tridiagonal T de ordem k calculada a partir dos valores α_j e β_j para k passos de CG [CR95a]:

$$T_{1,1} = \frac{1}{\alpha_1}, \quad T_{j,j} = \left(\frac{1}{\alpha_j} + \frac{\beta_j}{\alpha_{j-1}}\right),$$
$$T_{j,j-1} = \; T_{j-1,j} = \; -\frac{\sqrt{\beta_j}}{\alpha_{j-1}}, \quad 1 < j < k.$$

Os autovalores τ_j de T podem ser calculados pelo método eficiente e robusto QL com incremento (*shift*) implícito [DMW68]. A presença de autovalores grandes na matriz precondicionadora pode ser verificada por (3.3). Inicialmente η é escolhido e CCF(η) é calculado. Após a execução de algumas iterações, o maior valor de Ritz τ_{max} é estimado. Se ele for pequeno o processo continua, senão o precondicionador pode ser descartado desde que autovalores grandes podem tornar a convergência lenta. Assim, outro precondicionador é obtido

usando um η maior. Desta forma, os valores de Ritz produzidos pelo CG podem ser usados durante o processo para verificar o precondicionador.

Comparação com outros métodos

Campos e Birkett [CB98] fizeram uma comparação entre os métodos ICCG(0) (*incomplete Cholesky conjugate gradient* [MdV77]), IICCG(0) (precondicionador IICD [JP95]) e CCCG(0), todos eles sem preenchimento e usando incremento da diagonal para que as matrizes permaneçam definidas positivas durante as iterações. Em relação ao tempo, o precondicionador CCF é às vezes o mais lento devido a maior complexidade de sua estrutura de dados, que permite preenchimento e retirada. Porém, ele converge mais rápido porque produz matrizes mais bem condicionadas, o que pode ser comprovado pelo número e tempo das iterações.

Eles também compararam o seu desempenho com a sub-rotina MA31, que é uma implementação do método ICCG usando o esquema já mencionado de retirada por tolerância. Os resultados mostraram que o CCCG(η) faz uma melhor seleção de elementos, pois converge para os sistemas nos quais a MA31 falha, mesmo usando menos memória. A MA31 é da Harwell Subroutine Library e está implementada em outros pacotes matemáticos como o da NAG [Ltd90].

Finalmente, eles verificaram que o CCCG(η) é muito eficiente para solução de equações diferenciais parabólicas dependentes do tempo. Ao se permitir preenchimento no fator do precondicionador, o CCCG(η) foi até 14 vezes mais rápido do que a fatoração incompleta sem preenchimento.

Campos e Rollett [CR95a] analisaram o desempenho do CG por meio da distribuição dos autovalores da matriz dos coeficientes precondicionada por quatro diferentes esquemas. Eles também mostraram [CR95b] que, quando os fatores são obtidos sem preenchimento, a fatoração de Cholesky controlada é um precondicionador melhor que o ICCG [MdV77].

Um precondicionador adaptativo para o CG

Para algumas classes de problemas, é necessário resolver vários sistemas lineares cuja matriz de coeficientes é a mesma ou aproximadamente a mesma. Um exemplo é a solução de equações diferenciais parabólicas dependentes do tempo. Nesta situação, é possível estimar

um η ótimo (η_{otm}) para o tempo total t mínimo:

$$t = mp + si, \tag{3.4}$$

onde:
m = número de sistemas necessários para determinar η_{otm},
s = número total de sistemas a serem resolvidos,
p = tempo de precondicionamento,
i = tempo da iteração.

O mesmo precondicionador pode ser mantido para resolver os $s - m$ sistemas restantes tão logo η_{otm} seja encontrado. Tipicamente, $m \approx 10$ e se $s \approx m$, então $t = s(p + i)$ precisa ser minimizado. Por outro lado, se $s \gg m$, então $t = si$ precisa ser minimizado.

Este processo pode ser descrito variando-se η até que o melhor valor seja obtido. Em uma primeira instância, o intervalo $[a, c] \subset [\eta_{min}, \eta_{max}]$ que contém η_{otm} é obtido dobrando-se sucessivamente os intervalos até que um aumento do tempo necessário para o CCCG(η) resolver o sistema seja detectado. Para isto, o parâmetro η_{min} pode ser iniciado com $-n$ e η_{max} pode ser iniciado com o máximo armazenamento permitido. A partir daí, $\eta_{otm} \in [a, c]$ é obtido utilizando-se a minimização de Brent [Bre73].

Capítulo 4

Matrizes esparsas

Uma matriz é dita esparsa quando a maioria de seus elementos são iguais a zero, ou seja, ela possui relativamente poucos elementos não nulos. Sistemas lineares que surgem na solução de problemas reais possuem, em sua maioria, matrizes esparsas e de ordem elevada. Nestes casos, é proibitivo armazenar toda a matriz por questão de memória do computador e o que se faz na prática é guardar somente os elementos não nulos.

4.1 Estrutura de dados para matrizes esparsas

Existem vários esquemas para armazenar as matrizes esparsas. Para exemplificar, seja a pequena matriz de ordem 10 mostrada na Figura 4.1. Quando a matriz for simétrica então apenas os elementos não nulos localizados da diagonal para baixo (ou para cima) devem ser armazenados.

As seções seguintes mostram três tipos de estruturas de dados para o armazenamento de matrizes esparsas: esquema de coordenadas, coluna esparsa compactada e linha esparsa compactada.

4.1.1 Esquema de coordenadas

O modo mais simples de armazenar uma matriz esparsa é por meio do esquema de coordenadas ou tripleto. Ele consiste em um conjunto de três vetores, um especificando o índice da linha do elemento (`IndLin`), outro o índice da coluna (`IndCol`) e o terceiro o valor do elemento (`A`). O comprimento dos três vetores é igual ao número de elementos não nulos

da matriz. No caso de matriz simétrica, o comprimento é dado pelo número de elementos abaixo da diagonal. A Figura 4.2 apresenta o esquema de coordenadas para a matriz da Figura 4.1, de ordem $n = 10$ e com $m = 30$ elementos não nulos.

A maior dificuldade com o esquema de coordenadas encontra-se na inconveniência ao se ter acesso a estrutura de dados por linha ou por coluna, já que os vetores de linha e coluna não precisam estar, necessariamente, ordenados.

4.1.2 Coluna esparsa compactada

Pode-se observar na Figura 4.2 uma repetição desnecessária dos índices das colunas no vetor `IndCol` no formato de coordenadas. Isto pode ser evitado com o uso do esquema de coluna esparsa compactada (CSC: *Compressed Sparse Column*), que é considerado um formato padrão para matriz esparsa [DGL92]. Neste formato, o vetor de índice das colunas (`IndCol`) de tamanho m é substituído por um outro vetor (`PCol`) de tamanho $n+1$, contendo as localizações da primeira entrada de cada coluna da matriz nos vetores `IndLin` e `A`. A posição $n+1$ de `PCol` contém o valor $m+1$. A Figura 4.3 mostra o formato coluna esparsa compactada para a matriz da Figura 4.1, de ordem $n = 10$ e com $m = 30$ elementos não nulos.

Usando variáveis inteiras com i bytes e reais com r bytes, o número total de bytes gastos

i\j	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
1	5.0						−4.0			
2		2.4		2.1				1.5		
3			8.5		−3.0					3.5
4	1.1			1.9			6.0			
5			−2.0		5.7			4.0		
6	−3.0		6.0			10.3				
7		1.4				2.0	15.0		4.8	
8								5.8		7.7
9	3.6			2.2		3.3			14.2	
10					−3.2		3.8			6.1

Figura 4.1: Matriz esparsa não simétrica.

i	IndLin	IndCol	A
1	1	1	5.0
2	4	1	1.1
3	6	1	−3.0
4	9	1	3.6
5	2	2	2.4
6	7	2	1.4
7	3	3	8.5
8	5	3	−2.0
9	6	3	6.0
10	2	4	2.1

i	IndLin	IndCol	A
11	4	4	1.9
12	9	4	2.2
13	3	5	−3.0
14	5	5	5.7
15	10	5	−3.2
16	6	6	10.3
17	7	6	2.0
18	9	6	3.3
19	1	7	−4.0
20	4	7	6.0

i	IndLin	IndCol	A
21	7	7	15.0
22	10	7	3.8
23	2	8	1.5
24	5	8	4.0
25	8	8	5.8
26	7	9	4.8
27	9	9	14.2
28	3	10	3.5
29	8	10	7.7
30	10	10	6.1

Figura 4.2: Esquema de coordenadas ou tripleto.

pelo formato de coluna esparsa compactada para armazenar uma matriz A de ordem n e com m elementos não nulos é $B_A = (m + n + 1)i + mr$.

4.1.3 Linha esparsa compactada

Se em vez de indexar os elementos coluna por coluna for feito por linha, tem-se o esquema de linha esparsa compactada (CSR: *Compressed Sparse Row*). Ele consiste de três vetores, um contendo os índices das colunas da matriz (`IndCol`) de tamanho m, um segundo com os

i	PCol	IndLin	A
1	1	1	5.0
2	5	4	1.1
3	7	6	−3.0
4	10	9	3.6
5	13	2	2.4
6	16	7	1.4
7	19	3	8.5
8	23	5	−2.0
9	26	6	6.0
10	28	2	2.1

i	PCol	IndLin	A
11	31	4	1.9
12		9	2.2
13		3	−3.0
14		5	5.7
15		10	−3.2
16		6	10.3
17		7	2.0
18		9	3.3
19		1	−4.0
20		4	6.0

i	IndLin	A
21	7	15.0
22	10	3.8
23	2	1.5
24	5	4.0
25	8	5.8
26	7	4.8
27	9	14.2
28	3	3.5
29	8	7.7
30	10	6.1

Figura 4.3: Esquema de coluna esparsa compactada.

valores dos elementos (`A`) também de tamanho m e um terceiro (`PLin`) de tamanho $n+1$ com as localizações da primeira entrada de cada linha da matriz nos vetores `IndCol` e `A`. A última posição de `PLin` contém o valor $m+1$. é apresentado na Figura 4.4 o formato linha esparsa compactada para a matriz da Figura 4.1, de ordem $n = 10$ e com $m = 30$ elementos não nulos.

i	PLin	IndCol	A
1	1	1	5.0
2	3	7	−4.0
3	6	2	2.4
4	9	4	2.1
5	12	8	1.5
6	15	3	8.5
7	18	5	−3.0
8	22	10	3.5
9	24	1	1.1
10	28	4	1.9

i	PLin	IndCol	A
11	31	7	6.0
12		3	−2.0
13		5	5.7
14		8	4.0
15		1	−3.0
16		3	6.0
17		6	10.3
18		2	1.4
19		6	2.0
20		7	15.0

i	IndCol	A
21	9	4.8
22	8	5.8
23	10	7.7
24	1	3.6
25	4	2.2
26	6	3.3
27	9	14.2
28	5	−3.2
29	7	3.8
30	10	6.1

Figura 4.4: Esquema de linha esparsa compactada.

4.2 Algoritmos para operações com matrizes esparsas

A operação de maior custo computacional utilizada pelos métodos iterativos do subespaço de Krylov é a multiplicação matriz-vetor. Esta seção mostra um algoritmo para realizar esta multiplicação utilizando a estrutura de coluna esparsa compactada, a qual está sendo usada pelo algoritmo CCCG(η). Em seguida, são mostrados dois algoritmos para resolver sistemas lineares que usam a mesma estrutura de dados da multiplicação matriz-vetor: substituição sucessiva e retroativa.

4.2.1 Multiplicação matriz-vetor

A Figura 4.5 mostra um algoritmo para multiplicar uma matriz esparsa por um vetor, com a matriz armazenada no esquema de coluna esparsa compactada.

```
Algoritmo Matriz_vetor_coluna
{ Objetivo: Multiplicar uma matriz armazenada no esquema }
{  de coluna esparsa compactada por um vetor }
parâmetros de entrada n, IndLin, PCol, A, v
   { ordem, índice das linhas, apontador das colunas, valores dos elementos, }
   { vetor a ser multiplicado }
parâmetros de saída w  { vetor produto de matriz-vetor }
   para i ← 1 até n faça
      w(i) ← 0
   fim para
   para j ← 1 até n faça
      para k ← PCol(j) até PCol(j + 1) − 1 faça
         i ← IndLin(k)
         w(i) ← w(i) + A(k) ∗ v(j)
      fim para
   fim para
fim algoritmo
```

Figura 4.5: Produto matriz-vetor para matriz no esquema de coluna esparsa compactada.

4.2.2 Substituição sucessiva

A Figura 4.6 mostra um algoritmo para realizar a substituição sucessiva realizando a multiplicação matriz-vetor e utilizando o esquema de coluna esparsa compactada. Ele resolve o sistema linear inferior da forma $Lx = c$.

Este mesmo algoritmo está descrito na Figura 2.3, mas com a a estrutura de armazenamento densa.

4.2.3 Substituição retroativa

A Figura 4.7 mostra um algoritmo para realizar a substituição retroativa realizando a multiplicação matriz-vetor e utilizando o esquema de coluna esparsa compactada. Da mesma forma, este algoritmo está descrito na Figura 2.3 com a estrutura de armazenamento densa. Ele resolve o sistema linear superior da forma $Ux = d$.

Algoritmo Substituição_Sucessiva
{ **Objetivo:** Resolver o sistema triangular inferior $Lx = c$ }
{ pelas substituições sucessivas }
parâmetros de entrada $n,\ IndLin,\ PCol,\ L,\ c$
{ ordem, matriz triangular inferior e vetor independente. }
parâmetros de saída x { solução do sistema triangular inferior }
para $i \leftarrow 1$ **até** n **faça**
$x(i) \leftarrow c(i)$
fim para
para $i \leftarrow 1$ **até** n **faça**
para $j \leftarrow PCol(i) + 1$ **até** $PCol(i + 1) - 1$ **faça**
$k \leftarrow IndLin(j)$
$x(k) \leftarrow x(k) - L(j) * x(i)$
fim para
$x(i) \leftarrow x(i)\ /\ L(PCol(i))$
fim para
fim algoritmo

Figura 4.6: Substituição sucessiva.

Algoritmo Substituição_Retroativa
{ **Objetivo:** Resolver o sistema triangular superior $Ux = d$ }
{ pelas substituições retroativas }
parâmetros de entrada $n,\ IndLin,\ PCol,\ U,\ d$
{ ordem, matriz triangular superior e vetor independente. }
parâmetros de saída x { solução do sistema triangular superior }
para $i \leftarrow 1$ **até** n **faça**
$x(i) \leftarrow d(i)$
fim para
para $i \leftarrow n - 1$ **até** 1 **faça**
$soma \leftarrow 0$
para $j \leftarrow PCol(i) + 1$ **até** $PCol(i + 1) - 1$ **faça**
$soma \leftarrow soma + U(j) * x(IndLin(j))$
fim para
$x(i) \leftarrow d(i)\ /\ L(PCol(i)) - soma$
fim para
fim algoritmo

Figura 4.7: Substituição retroativa.

4.3 Grafos e matrizes esparsas

A teoria dos grafos foi identificada como uma poderosa ferramenta para a computação de matrizes esparsas quando Seymour Parter [Par61] usou grafos não direcionados para modelar a eliminação Gaussiana há mais de quarenta anos atrás. Grafos podem ser usados para modelar matrizes simétricas, não simétricas, fatorações etc. Além de tornar mais fácil a compreensão e análise de algoritmos para matrizes esparsas, eles também aumentam o escopo de manipulação destas matrizes usando algoritmos e técnicas já existentes.

Um grafo é, fundamentalmente, um modo de representar uma relação binária entre objetos. Para o propósito deste trabalho, considere um grafo $G = (V, E)$ como um conjunto de vértices $V = \{v_1, v_2, \dots\}$ (ou nós) e um conjunto de arestas $E = \{e_1, e_2, \dots\}$. Estas arestas são representadas por pares não ordenados, por exemplo, $e_1 = (v_1, v_2)$.

Um conhecimento básico de teoria dos grafos é importante para o entendimento do trabalho. Conceitos mais específicos serão introduzidos quando necessários. Um resumo dos principais conceitos é dado por:

- Grau do vértice: número de arestas incidentes no vértice.
- Vértices adjacentes: dois vértices v_1 e v_2 são adjacentes quando possuem uma aresta entre eles, ou seja, $e_1 = (v_1, v_2)$.
- Caminho: seqüência de arestas disjuntas $(x_1, x_2) \rightarrow (x_2, x_3) \rightarrow \cdots \rightarrow (x_{k-1}, x_k)$.
- Ciclo: caminho com mesmo vértice inicial e final.
- Grafo conectado (conexo): possui caminho entre qualquer par de vértices.
- Subgrafo completo (clique): cada vértice do subgrafo possui aresta incidente em todos os outros vértices do subgrafo.
- Árvore: grafo conectado sem ciclos.

Assim como um grafo, uma matriz também descreve uma relação binária entre variáveis através de seus elementos não nulos. Uma matriz simétrica $A_{n\times n}$ pode ser modelada como um grafo $G(V, E)$, onde os n vértices em V representam as dimensões da matriz e existe uma aresta conectando os vértices i e j se $a_{ij} \neq 0$.

A Figura 4.8 mostra uma matriz simétrica de ordem 10 e um grafo não direcionado que representa esta matriz. Como na linha 1 existem os elementos não nulos $a_{1,1}$ e $a_{1,2}$,

conclui-se que nó 1 está conectado ao nó 2. Da mesma forma, como existem os elementos $a_{2,1}, a_{2,2}, a_{2,3}, a_{2,5}$ e $a_{2,6}$, o nó 2 está conectado aos nós 1, 3, 5 e 6. O mesmo raciocínio pode ser feito usando os elementos $a_{1,2}, a_{2,2}, a_{3,2}, a_{5,2}$ e $a_{6,2}$ pertencentes a coluna 2 de A.

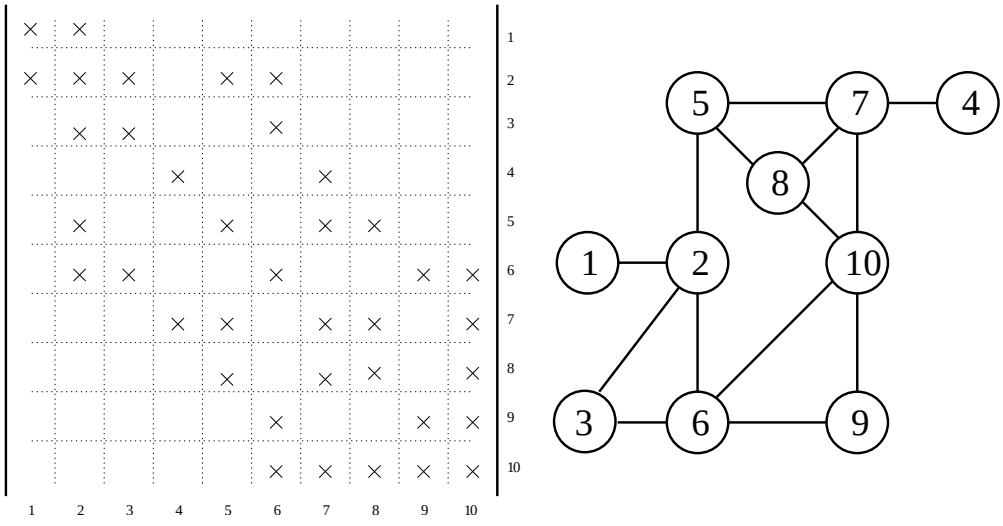
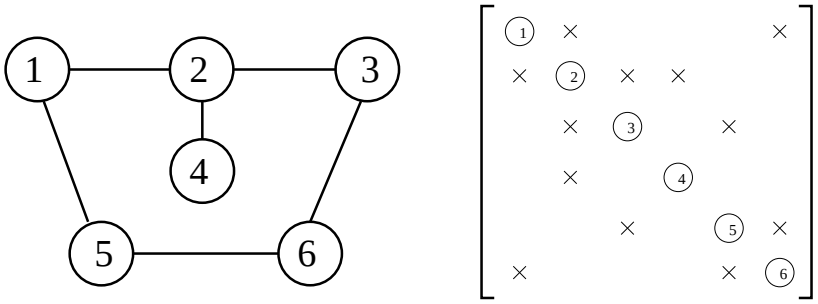


Figura 4.8: Representação de uma matriz simétrica por um grafo.

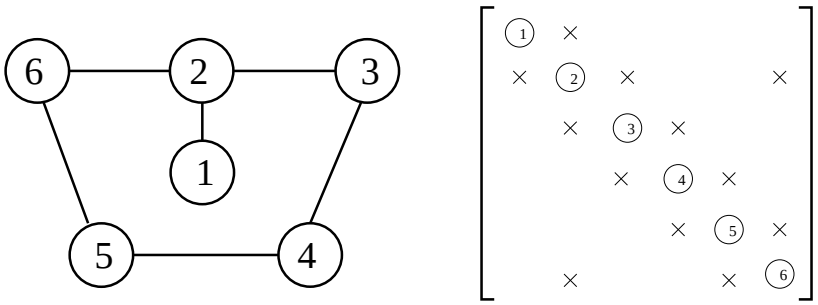
A reordenação de uma matriz a partir de um vetor é assunto principal do capítulo 5. Como o uso de grafos pode facilitar o estudo de matrizes esparsas, esta reordenação de matrizes será vista em termos da reordenação dos nós de um grafo.

Na Figura 4.9(a) é possível ver a ordenação do grafo $G(V, E)$. Uma ordenação é simplesmente o mapeamento de um vetor $\alpha = \{1, 2, 3, 4, 5, 6\}$ aos vértices do grafo, onde os elementos deste vetor identificam a linha e coluna da matriz. Assim, o nó 1 e seus relacionamentos são representados na linha e na coluna 1 da matriz, o nó 2 na linha e na coluna 2, e assim por diante.

Por outro lado, uma reordenação deste grafo é uma mudança do mapeamento inicial, ou seja, aplica-se um vetor $\beta = \{6, 2, 3, 1, 4, 5\}$ aos vértices, por exemplo. O resultado desta reordenação β no grafo 4.9(a) pode ser visto na Figura 4.9(b), onde uma nova matriz é formada. Para construir esta nova matriz, os nós do grafo são renumerados e esta nova numeração vai representar cada linha e coluna da matriz. Desta forma, o nó 4 e seus relacionamentos são representados na linha e na coluna 1 da matriz, já que o identificador 1 está na quarta posição do vetor. Os nós 2 e 3 continuam representados na mesma linha e coluna, porém seus relacionamentos mudam de posição de acordo com as mudanças dos nós. O nó 5 e seus relacionamentos são representados na linha e na coluna 4 da matriz, já que o identificador 4 está na quinta posição do vetor. O nó 6 ocupa a posição 5 da matriz e o nó 1 está na posição 6.



(a) Grafo ordenado por $\alpha = \{1, 2, 3, 4, 5, 6\}$ e sua matriz correspondente.



(b) Reordenação do grafo (a) por $\beta = \{6, 2, 3, 1, 5, 4\}$ e matriz resultante.

Figura 4.9: Reordenação de um grafo.

Capítulo 5

Algoritmos para reordenação de matrizes esparsas

Neste capítulo serão apresentadas algumas técnicas para reordenação de matrizes esparsas simétricas que representam as matrizes dos coeficientes de sistemas lineares de grande porte. A principal motivação para o uso destas técnicas está na possibilidade da redução de memória durante o processo de decomposição de Cholesky (seção 2.2) utilizado na solução destes sistemas pelo CCCG(η).

Para exemplificar, considere o sistema $Ax = b$:

$$\begin{bmatrix} 4 & 1 & 2 & 0,5 & 2 \\ 1 & 0,5 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 0,5 & 0 & 0 & 0,625 & 0 \\ 2 & 0 & 0 & 0 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 7 \\ 3 \\ 7 \\ -4 \\ -4 \end{bmatrix}.$$

De acordo com (2.1), o fator de Cholesky da matriz de coeficientes A é dado por:

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0,5 & 0,5 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 \\ 0,25 & -0,25 & -0,5 & 0,5 & 0 \\ 1 & -1 & -2 & -3 & 1 \end{bmatrix}.$$

O exemplo acima ilustra o fato mais importante da aplicação do método de Cholesky a uma matriz esparsa A: a matriz decomposta normalmente sofre um preenchimento, ou seja, L possui elementos não nulos em posições onde havia zeros em A.

Mudando as posições das variáveis x_i de acordo com a regra $\tilde{x_i} \rightarrow x_{6-i}$, $i = 1, \ldots, 5$ e rearranjado as equações de forma que o último se torne o primeiro, o penúltimo se torne o segundo e assim sucessivamente, obtém-se o seguinte sistema de equações:

$$\begin{bmatrix} 16 & 0 & 0 & 0 & 2 \\ 0 & 0,625 & 0 & 0 & 0,5 \\ 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0,5 & 1 \\ 2 & 0,5 & 2 & 1 & 4 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \\ \tilde{x}_5 \end{bmatrix} = \begin{bmatrix} -4 \\ -4 \\ 7 \\ 3 \\ 7 \end{bmatrix}.$$

Esta mudança das posições das variáveis e a reordenação das equações equivale a uma permutação das linhas e colunas de A, a qual também é aplicada a x e a b. Esta permutação é realizada pela multiplicação de uma matriz P construída a partir de uma matriz identidade da mesma ordem do sistema e com as posições das linhas e colunas trocadas de acordo com a regra de permutação. Neste caso, temos o vetor de ordenação igual a $(5, 4, 3, 2, 1)$, resultando na seguinte matriz P:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

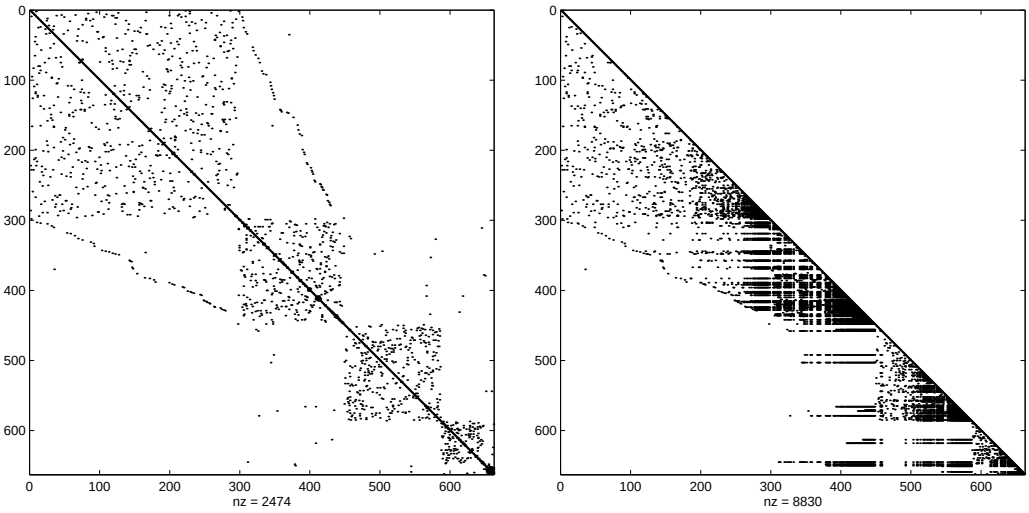
Este novo sistema $PAP^T(Px) = Pb$, onde $P^T = P^{-1}$, é referido como $\tilde{A}\tilde{x} = \tilde{b}$. Usando o método de Cholesky neste novo sistema, a fatoração de $\tilde{A}$ em $\tilde{L}\tilde{L}^T$ retorna:

$$\tilde{L} = \begin{bmatrix} 4 & 0 & 0 & 0 & 0 \\ 0,5 & 0,791 & 0 & 0 & 0 \\ 0 & 0 & 1,73 & 0 & 0 \\ 0 & 0 & 0 & 0,707 & 0 \\ 0,5 & 0,632 & 1,15 & 1,41 & 1,29 \end{bmatrix}.$$

A solução encontrada $\tilde{x}$ é, simplesmente, uma forma rearranjada de x. É necessário mudar a ordem dos elementos dessa solução conforme a matriz de permutação para se chegar a solução desejada x. O ponto crucial é que essa reordenação de equações e variáveis produziu um fator triangular $\tilde{L}$ tão esparso quanto a matriz triangular inferior de A. Embora seja difícil na prática alcançar esta esparsidade, uma reordenação de linhas e colunas pode levar a uma enorme redução no preenchimento, reduzindo o custo de execução e armazenamento. O estudo de algoritmos que realizam este processo de reordenação é o assunto principal deste capítulo. Serão vistos três algoritmos: contagem de colunas, Cuthill-McKee reverso e mínimo grau.

A matriz apresentada ilustra as características básicas de uma eliminação e o efeito da reordenação. Para enfatizar melhor estes pontos, considere um exemplo maior (662 × 662), onde o padrão de esparsidade está mostrado na Figura 5.1(a). Esta matriz é conhecida como *662_bus* (série psadmit) e pertence à coleção Harwell-Boeing [DGL92], onde existem várias séries de matrizes disponíveis para testes. Pela fatoração desta matriz em LL^T, obtém-se a estrutura de L em 5.1(b), onde percebe-se a grande quantidade de preenchimento obtido. Enquanto A possui 1568 elementos não nulos na parte triangular inferior (2474 no total), L possui 8830.

A Figura 5.2(a) mostra a estrutura da permutação simétrica A'. Esta matriz foi obtida de A usando-se o método Cuthill-McKee reverso que concentra os elementos próximos da diagonal. Seu fator de Cholesky L', exibido na Figura 5.2(b), possui 6595 elementos não nulos. Já a Figura 5.3(a) mostra a matriz A'' que foi obtida também de A utilizando-se o método do mínimo grau. Este método tem o objetivo de diminuir o preenchimento criado durante a fatoração de Cholesky, o que pode ser facilmente comprovado pelo seu fator L'' (Figura 5.3(b)) que possui apenas 2551 elementos não nulos.

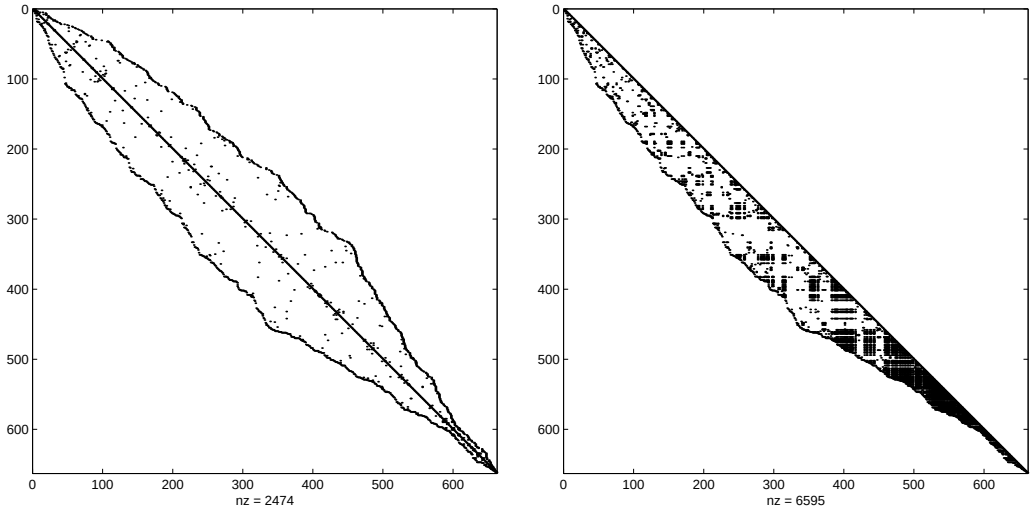


a) Matriz A b) Fator L de Cholesky

Figura 5.1: Preenchimento causado pela fatoração da matriz *662_bus*.

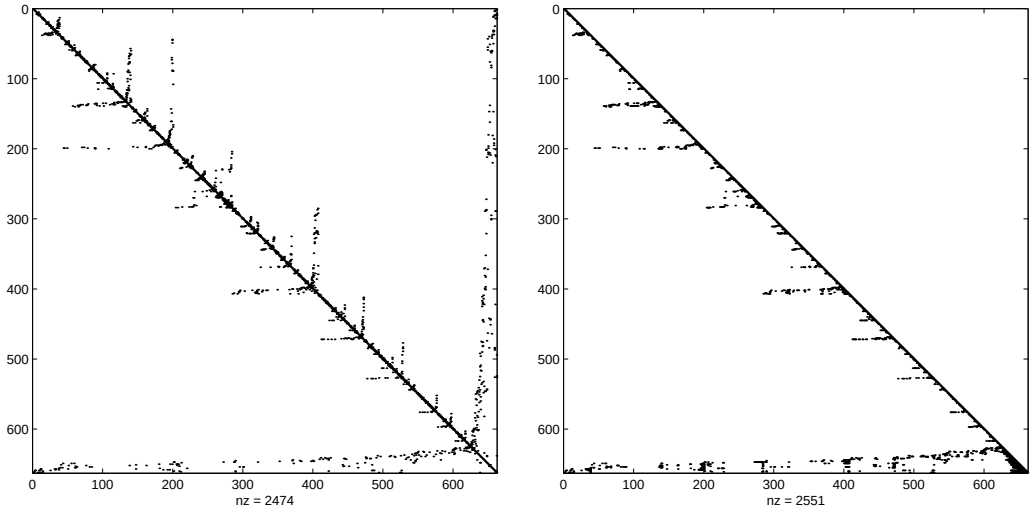
5.1 Contagem de colunas

O método de contagem de colunas (*column count*) é baseado no número de elementos não nulos de cada coluna da matriz. Ele simplesmente move linhas e colunas com o maior número de elementos não nulos para as últimas posições da matriz. É um bom método de



a) Matriz A' b) Fator L' de Cholesky

Figura 5.2: Fatoração da matriz *662_bus* reordenada por Cuthill-McKee reverso.



a) Matriz A'' b) Fator L'' de Cholesky

Figura 5.3: Fatoração da matriz *662_bus* reordenada por mínimo grau.

reordenação para matrizes com estruturas muito irregulares, especialmente se há uma grande variação na quantidade destes elementos.

Neste método são realizadas as seguintes etapas:

1. Contagem de elementos de cada coluna e armazenamento em um vetor;
2. Ordenação deste vetor;
3. Permutação das linhas e colunas da matriz com base neste vetor criado.

A ordenação é feita através do algoritmo Quicksort não recursivo da Figura 5.4.

```
Algoritmo Quicksort_não_recursivo
parâmetros de entrada A, N { Vetor a ser ordenado, ordem do vetor. }
  K ← 1;   StackL(1) ← 1;   StackR(K) ← N;
  enquanto (K <> 0)
    L ← StackL(K);   R ← StackR(K);   K ← K − 1;
    enquanto (L < R)
      I ← L;   J ← R;   X ← A((L + R)/2);
      enquanto (I <= J)
        enquanto (A(I) < V)   I ← I + 1;   fim enquanto
        enquanto (X < A(J))   J ← J − 1;   fim enquanto
        se I <= J então
          W ← A(I);   A(I) ← A(J);   A(J) ← W;   I ← I + 1;   J ← J − 1;
        fim se
      fim enquanto
      se (J − L) < (R − I) então
        se I < R então
          K ← K + 1;   StackL(K) ← I;   StackR(K) ← R;
        fim se
        R ← J
      senão
        se L < J então
          K ← K + 1;   StackL(K) ← L;   StackR(K) ← J;
        fim se
        L ← I
      fim se
    fim enquanto
  fim enquanto
fim algoritmo
```

Figura 5.4: Quicksort não recursivo.

Desta forma, o método de contagem de colunas pode levar a fatores mais esparsos. Não é o melhor método de reordenação existente, mas é rápido e realiza um bom trabalho. O algoritmo utilizado para a ordenação do vetor é o quicksort [CLR90] e a complexidade do contagem de colunas é dada por este algoritmo de ordenação, ou seja, $O(n \log_2 n)$.

5.2 Cuthill-McKee reverso

Em 1969, Cuthill e McKee [CM69] propuseram um algoritmo de reordenação, cujo objetivo principal é reduzir a largura de banda (*bandwidth*) de uma matriz simétrica. A largura de banda β de uma matriz A é dada pela maior distância de um elemento não nulo à diagonal principal, ou seja:

$$\beta(A) = \max_{a_{ij} \neq 0} |i - j|.$$

Desta forma, os elementos são movidos para perto da diagonal, melhorando a localidade de acesso aos dados dentro das matrizes. Como esta propriedade é mantida no fator de Cholesky correspondente, a reordenação tende a reduzir o preenchimento. Isto torna o algoritmo muito usado na prática. Não é garantido que se encontre a menor largura de banda possível, mas isso normalmente acontece.

O Cuthill-McKee executa uma minimização local de cada largura de banda β_i correspondente a cada linha i para reduzir a largura de banda da matriz. Isto sugere que o algoritmo possa ser usado para reduzir o *profile* da matriz, que é dado por:

$$profile(A) = \sum_{i=1}^{n} \beta_i(A).$$

George [Geo71] propôs em 1971 que a ordenação obtida pelo reverso da Cuthill-McKee, freqüentemente, supera a original em termos de redução do *profile*, embora mantenha a mesma largura de banda. Esse algoritmo é conhecido como Cuthill-McKee reverso (RCM: *reverse Cuthill-McKee*).

Considerando um grafo conectado, o problema de reduzir a largura de banda é convertido em encontrar uma renumeração dos vértices do grafo tal que a diferença entre os índices seja mínima. Seja n o número total de vértices representando a dimensão da matriz A. Uma aresta conecta os nós i e j quando A_{ij} é não nulo. Assim, temos o seguinte algoritmo:

1. Vértice inicial : Determina o vértice inicial r, atribuindo seu valor a x_1;

2. Parte principal : Para $i = 1, \ldots, n$, encontra todos os vizinhos não numerados do vértice x_i, numerando-os em ordem crescente do grau;

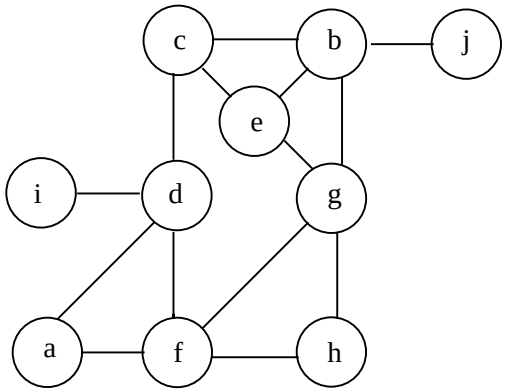
3. Ordenação reversa : A ordenação reversa é dada por $y_1, \ldots, y_n$ onde $y_i = x_{n-i+1}$ para $i = 1, \ldots, n$.

Quando o grafo é desconexo, é possível aplicar o algoritmo acima a cada componente conexo. Para um dado nó inicial, a parte principal do RCM (Figura 5.5) é relativamente simples de ser implementado tendo como base o algoritmo de busca em largura (*Breadth-first search* [CLR90]).

```
Algoritmo RCM (Parte Principal)
{ Objetivo: Visitar vértices alcançáveis a partir da raiz V. }
parâmetros de entrada  V  { Vértice raiz. }
   Fila ← V; { Enfileira }
   Visitou(V) ← verdadeiro;
   enquanto FilaVazia(Fila) = falso faça
      V ← Fila; { Desenfileira }
      Numera vértice V;
      { Visita os vértices adjacentes a W em ordem crescente do grau }
      para cada vértice W adjacente a V em ordem crescente de grau faça
         se Visitou(W) = falso então
            Fila ← W; { Enfileira }
            Visitou(W) ← verdadeiro;
         fim se
      fim para
   fim enquanto
fim algoritmo
```

Figura 5.5: Caminhamento do RCM baseado na busca em largura.

No grafo da Figura 5.6(a), suponha que o nó g seja o nó inicial. A Tabela 5.6(b) mostra como seria a numeração dos nós dada pelo algoritmo RCM. A ordenação resultante é dada pela Figura 5.7, onde os nós do grafo original 5.6(a) estão numerados em ordem contrária ao caminhamento. O nó inicial g está identificado pelo número 10 e último nó visitado (i) está identificado pelo número 1. A matriz resultante da ordenação também pode ser vista nesta figura, onde tem-se um *profile* de tamanho 22.



a) Grafo original.

i	nó x_i	Vizinhos não numerados em ordem crescente de grau
1	g	h,e,b,f
2	h	-
3	e	c
4	b	j
5	f	a,d
6	c	-
7	j	-
8	a	-
9	d	i
10	i	-

b) Numeração dos vértices a partir de g.

Figura 5.6: Caminhamento do algoritmo RCM.

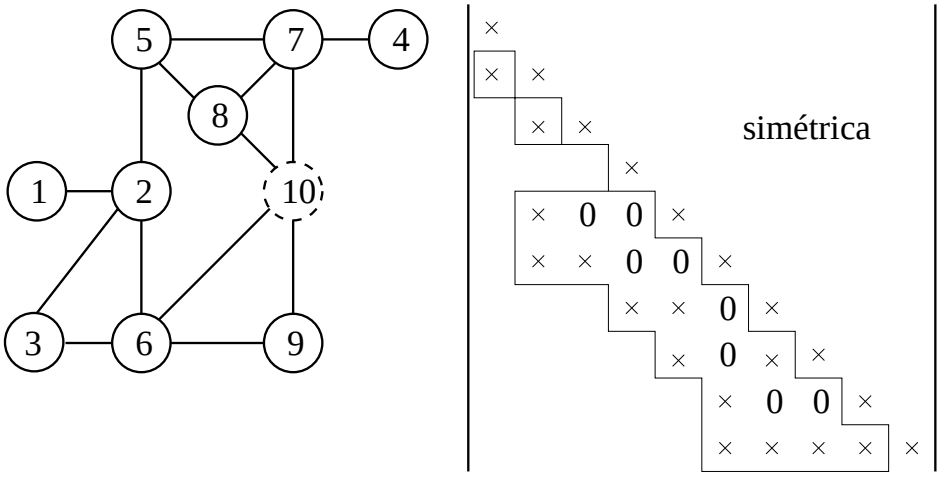


Figura 5.7: Ordenação do grafo 5.6(a) a partir de 5.6(b) e matriz resultante.

A qualidade da ordenação do algoritmo depende, essencialmente, da escolha do nó inicial. No grafo da Figura 5.6(a), se o nó a fosse escolhido como nó inicial, o resultado seria dado pela Figura 5.8. O nó inicial a está identificado pelo número 10 e último nó visitado (j) possui número 1. Nesta mesma figura, é possível ver a matriz resultante, a qual possui um *profile* de tamanho 18.

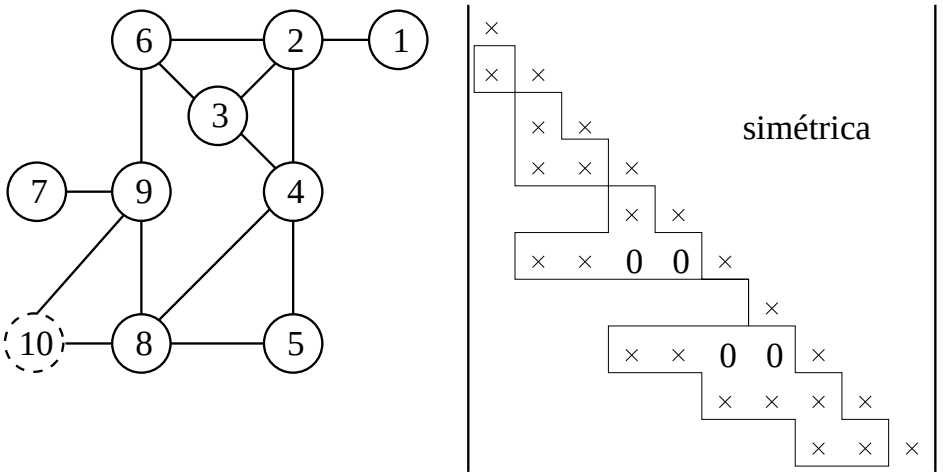


Figura 5.8: Ordenação do grafo 5.6(a) com nó inicial a e matriz resultante.

Nó pseudo-periférico

Um estudo feito por George e Liu [GL79] propõe um algoritmo para encontrar o vértice inicial do algoritmo. O objetivo é encontrar um par de nós que estão o máximo possível (ou próximo disto) distantes um do outro.

Um caminho de tamanho k de um nó x_0 para o nó x_k é um conjunto ordenado de vértices distintos $(x_0, x_1, \ldots, x_k)$, onde x_i pertence ao conjunto de nós adjacentes a x_{i+1}, ou seja, $x_i \in adj(x_{i+1})$ para $i = 0, 1, \ldots, k-1$. A distância $d(x, y)$ entre dois nós x e y no grafo conectado $G = (X, E)$ é, simplesmente, o tamanho do menor caminho entre os nós x e y. Por definição, a excentricidade ℓ de um nó x é dada pela sua maior distância, ou seja:

$$\ell(x) = max\{d(x, y) \mid y \in X\}.$$

O diâmetro δ de um grafo G é dado pela maior excentricidade alcançada por um nó qualquer pertencente a este grafo:

$$\begin{aligned} \delta(G) &= max\{\ell(x) \mid x \in X\}, \\ \delta(G) &= max\{d(x, y) \mid x, y \in X\}. \end{aligned}$$

Um nó $x \in X$ é conhecido como periférico se sua excentricidade for igual ao diâmetro do grafo, ou seja, se $\ell(x) = \delta(G)$. A Figura 5.9 mostra um grafo com 8 nós e diâmetro 5. Os nós x_2, x_5 e x_7 são nós periféricos, já que possuem a maior excentricidade deste grafo.

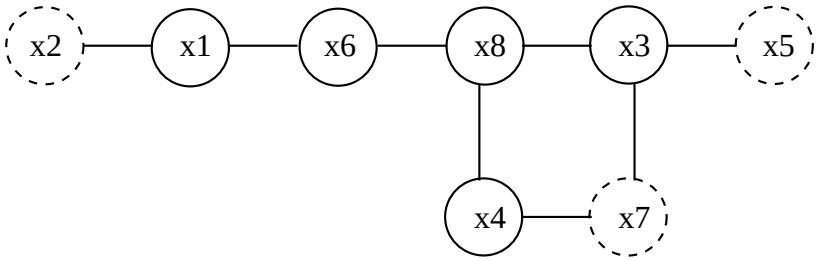


Figura 5.9: Nós periféricos.

Desta forma, o objetivo é descrever uma heurística eficiente para encontrar nós de alta excentricidade. O algoritmo não garante encontrar um nó periférico, ou mesmo próximo disto. Porém, os nós encontrados possuem alta excentricidade, sendo bons vérticies iniciais para o RCM. Estes vértices são conhecidos como nós pseudo-periféricos.

Uma modelagem de grafos muito útil na descrição deste algoritmo é conhecida como estrutura de nível (*rooted level structure*). Dado um nó $x \in X$, a estrutura de nível de x é o particionamento $\mathcal{L}(x)$ de X satisfazendo:

$$\mathcal{L}(x) = \{L_0(x), L_1(x), \ldots, L_{\ell(x)}(x)\},$$

onde

$$L_0(x) = \{x\}, L_1(x) = adj(L_0(x)),$$

e

$$L_i(x) = adj(L_{i-1}(x)) - L_{i-2}(x), i = 2, 3, \ldots, \ell(x).$$

O particionamento $\mathcal{L}(x)$ possui um tamanho igual a excentricidade $\ell(x)$ e sua largura $w(x)$ é definida por:

$$w(x) = max\{|L_i(x)| \mid i = 0, 1, \ldots, \ell(x)\}.$$

A estrutura de nível do nó x_6 da Figura 5.9 é vista na Figura 5.10. O particionamento dos nós é dado por $L_0(x_6) = \{x_6\}, L_1(x_6) = \{x_1, x_8\}, L_2(x_6) = \{x_2, x_3, x_4\}, L_3(x_6) = \{x_5, x_7\}$. Desta forma, tem-se que $\ell(x_6) = 3$ e $w(x_6) = 3$.

O algoritmo 5.11 descreve a busca pelo nó pseudo-periférico.

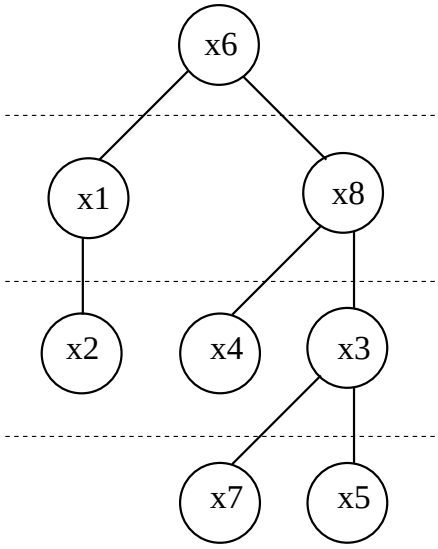


Figura 5.10: Estrutura de nível.

Algoritmo Nó_pseudo_periférico
{ **Objetivo:** Encontrar um nó pseudo periférico de um grafo. }
parâmetros de entrada G(X) { Grafo. }
{ Inicialização }
Escolher um nó qualquer r em X;
{ Geração da estrutura de nível }
Construir a estrutura de nível de r:
$\mathcal{L}(r) = \{L_0(r), L_1(r), \ldots, L_{\ell(r)}(r)\}$;
repita
Escolher um nó x em $L_{\ell(r)}(r)$ de mínimo grau;
{ Nova geração da estrutura de nível }
Construir a estrutura de nível de x;
se $\ell(x) > \ell(r)$ **então**
$r \leftarrow x$;
senão
interrompa
fim se
fim repita
{ Finalização }
O nó x é o nó pseudo-periférico;
fim algoritmo

Figura 5.11: Busca pelo nó pseudo-periférico.

A Figura 5.12 mostra um exemplo da execução do algoritmo RCM em um grafo a partir de três nós iniciais diferentes. O nó raiz (inicial) está identificado pela letra r e os nós pertencentes ao nível i estão identificados pelo inteiro i. No primeiro caso, o algoritmo alcança apenas dois níveis de profundidade. Nos dois últimos grafos, o nó raíz escolhido é pseudo-periférico e o algoritmo alcança quatro níveis de profundidade.

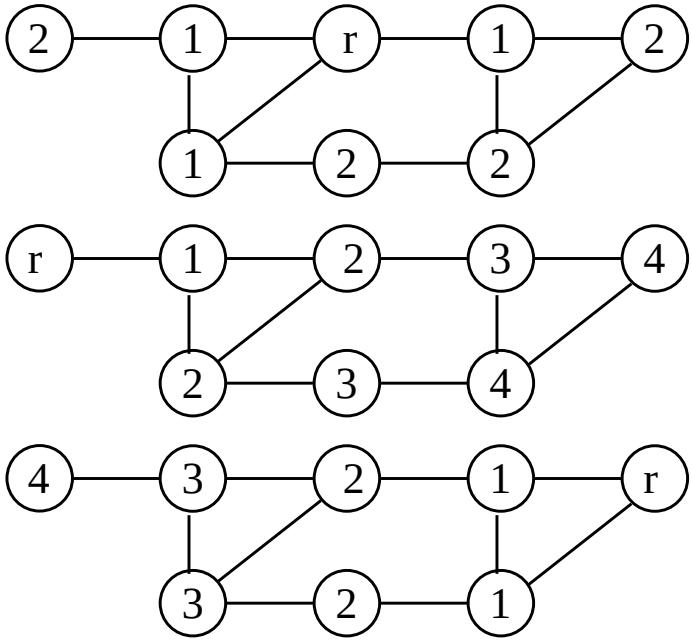


Figura 5.12: Aplicação de algoritmo.

A complexidade do RCM é dada pelo produto entre o grau n do vértice de maior grau do grafo e o número total de arestas $|E|$, ou seja, $O(n|E|)$.

5.3 Mínimo grau

O algoritmo mais usado para reordenação de matrizes esparsas é o mínimo grau [GL80, GL81]. Esta heurística tem o objetivo de encontrar uma ordenação para a matriz tal que ela sofra o menor preenchimento quando for fatorada pela decomposição de Cholesky. O algoritmo é muito sofisticado e poderoso, levando a fatores de solução normalmente mais esparsos que outras ordenações eficientes como Cuthill-Mckee reverso.

Usando uma abordagem mais simples, pode-se modelar a matriz simétrica como um grafo não direcionado, conhecido como grafo de eliminação G. O algoritmo básico pode ser convenientemente escrito em termos do grafo de eliminação da Figura 5.13.

Algoritmo Mínimo_grau
parâmetros de entrada A { Matriz simétrica. }
{ Inicialização }
Construir o grafo de eliminação G_0 correspondente a matriz A;
para $k = 1, 2, \ldots$ **até** $G_k = \{\ \}$ **faça**
Escolher um vértice v_k de G_k com o menor grau;
Eliminar v_k de G_k para formar o novo grafo de eliminação G_{k+1};
fim para
fim algoritmo

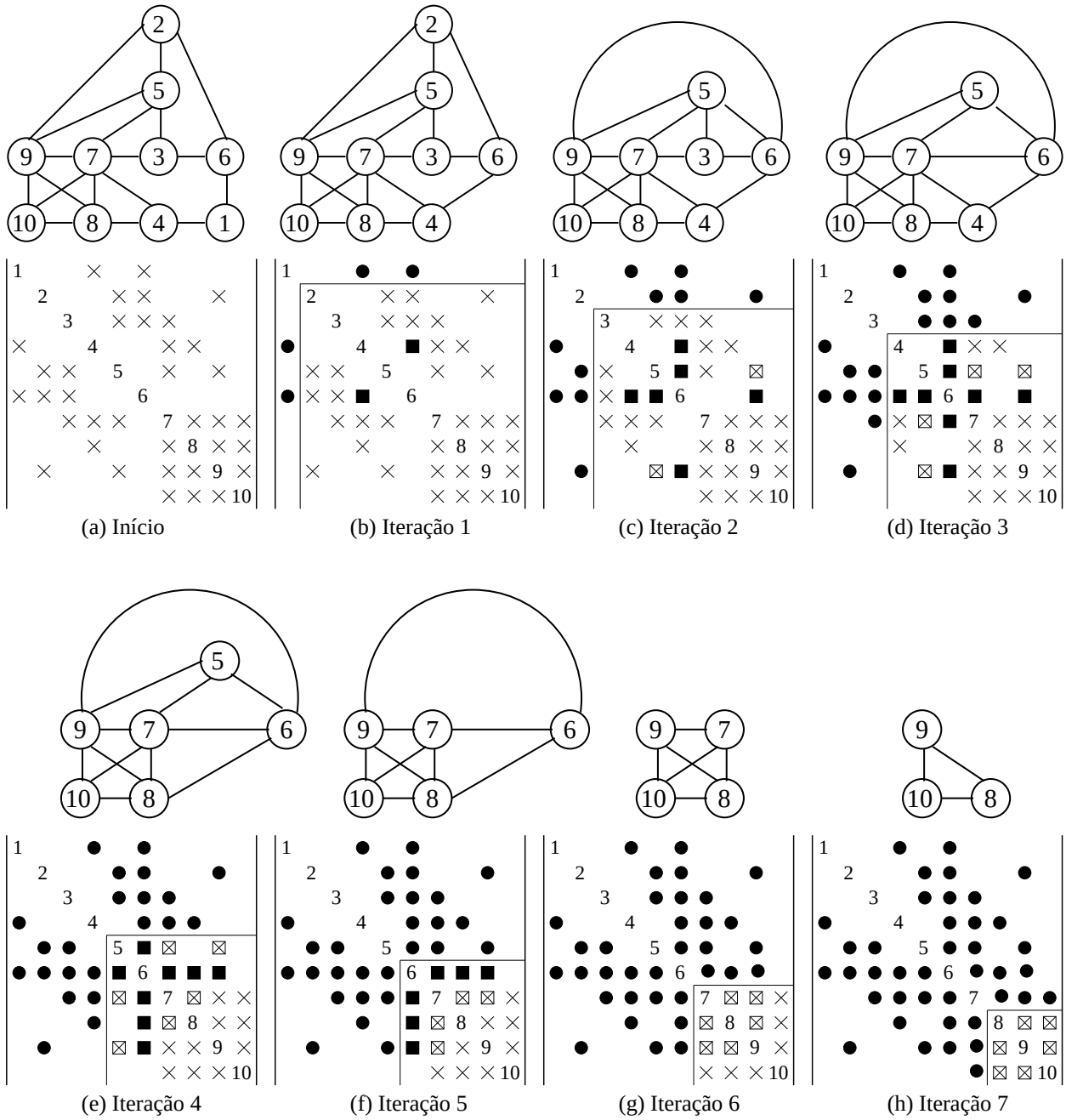
Figura 5.13: Mínimo grau.

A cada passo k, o vértice com o menor grau é escolhido no grafo de eliminação corrente G_k. Uma eliminação $G(v)$ é obtida de um grafo G pela remoção do vértice v e de todas as suas arestas incidentes. Porém, novas arestas são adicionadas transformando todos os vértices adjacentes a v em um clique, ou seja, cria-se uma aresta entre cada par de vértices adjacentes a v. A ordenação resultante é uma seqüência de vértices $v_0, v_1, \ldots$ eliminados pelo algoritmo.

A Figura 5.14 mostra o relacionamento entre o algoritmo mínimo grau (MD: *minimum degree*) e o preenchimento da matriz. Nesta figura pode-se observar o grafo de eliminação e a matriz correspondente durante as iterações do algoritmo. A parte (a) corresponde ao início do algoritmo, a parte (b) se refere a eliminação do nó 1 (iteração 1), a parte (c) corresponde a eliminação do nó 2 (iteração 2) e assim sucessivamente.

Nas matrizes desta figura, os elementos das diagonais são numerados de acordo com o nó correspondente no grafo. Suas posições não nulas, que correspondem às arestas do grafo, são representadas por $\times$. À medida que os nós são eliminados, as matrizes L e L^T são formadas com as posições não nulas representadas por $\bullet$. Desta forma, em (b) tem-se a primeira linha de L^T e a primeira coluna de L.

O preenchimento (novas arestas) criado no decorrer da fatoração são representados por $\blacksquare$. Assim, a matriz resultante da eliminação do nó 1 sofreu preenchimento entre os nós 4 e 6. As posições originalmente não nulas, onde uma nova aresta seria inserida durante a eliminação, são representados por $\boxtimes$ (posições que não precisaram sofrer preenchimento).



(a) Início (b) Iteração 1 (c) Iteração 2 (d) Iteração 3

(e) Iteração 4 (f) Iteração 5 (g) Iteração 6 (h) Iteração 7

Figura 5.14: Grafo de eliminação e matriz correspondente.

A seleção de um nó de mínimo grau a ser eliminado implica que o clique formado seja pequeno. O sucesso desta heurística pode ser atribuída a esta propriedade de formar o menor clique possível como resultado da eliminação. Muitas técnicas têm sido incorporadas a este algoritmo visando melhorar a eficiência. Na seção 5.3.1 serão apresentadas algumas destas melhorias.

5.3.1 Propriedades do algoritmo

Diversas técnicas têm sido incorporadas ao algoritmo básico do MD para melhorar seu desempenho. Depois de uma eliminação, o grafo é modificado e os graus dos nós remanescentes têm que ser recalculados para que o vértice de mínimo grau seja escolhido novamente. Como a operação mais cara do algoritmo consiste na atualização dos graus, estas técnicas visam uma redução deste número de atualização dos graus.

Eliminação em massa

Uma técnica simples e muito utilizada para reduzir a quantidade de atualização de graus é conhecida como eliminação em massa (*mass elimination*). Esta técnica usa o conceito de *supernode* ou nós indistingüíveis para eliminar um subconjunto de nós na mesma iteração, o que pode agilizar o algoritmo.

No processo de eliminação, dois nós x e y se tornam indistingüíveis quando satisfazem a igualdade:

$$adj(y) \cup y = adj(x) \cup x,$$

onde $adj(y)$ representa o conjunto de nós adjacentes a y. Desta forma, ter-se-ia um *supernode* formado por x e y que seria eliminado de uma vez. O número de nós indistingüíveis contidos em um *supernode* é conhecido como o peso do *supernode*.

Grau externo

No esquema convencional, o grau de um nó é calculado pelo número de nós adjacentes no grafo de eliminação corrente, ou seja, o grau exato. Desde que a técnica de eliminação em massa esteja sendo usada, o tamanho do clique formado por um nó y eliminado com seus indistingüíveis pode ser diferente do tamanho do clique formado se y fosse eliminado sozinho (grau exato).

Diante disto, a técnica do grau externo é utilizada para substituir o grau exato. O grau externo de um nó y é o número de nós vizinhos que não sejam indistingüíveis a ele, ou seja, o grau externo ao *supernode* contendo y. O grafo da Figura 5.15 exemplifica este conceito.

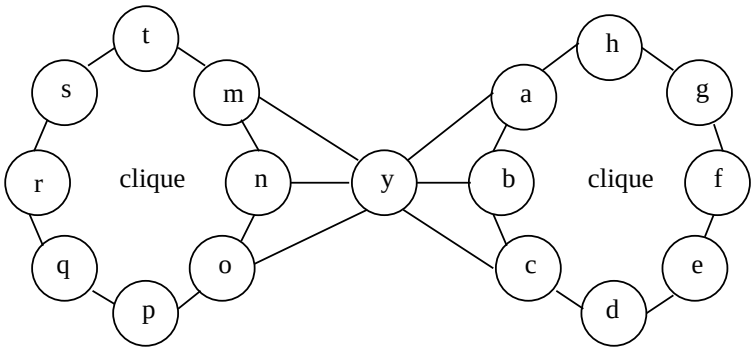


Figura 5.15: Grau externo.

O mínimo grau corrente é 6, e o nó que possui este grau é y. A eliminação deste nó irá criar um clique com os nós $\{a, b, c, m, n, o\}$. Assim, estes nós ficariam com o grau exato de tamanho 10 e os próximos a serem eliminados seriam $\{d, e, f, g, h\}$ ou $\{p, q, r, s, t\}$, todos de grau 7.

Pela Tabela 5.1 é possível comparar o grau exato ao grau externo. Como y não possui vizinhos indistingüíveis, seu grau externo é o próprio grau exato. Já o nó a é indistingüível de b e c, fazendo que seu grau externo seja o grau exato diminuído de 2. Por outro lado, o nó d é indistingüível de e, f, g e h, resultando num grau externo ainda menor $(7 - 4 = 3)$.

Nós	Grau exato	Grau externo
y	6	6
$\{a, b, c\}, \{m, n, o\}$	8	6
$\{d, e, f, g, h\}, \{p, q, r, s, t\}$	7	3

Tabela 5.1: Comparação entre grau exato e grau externo.

A eliminação em massa do *supernode* $\{d, e, f, g, h\}$ prioritariamente, resulta em um clique $\{a, b, c\}$ de tamanho 3 e, conseqüentemente, um preenchimento bem menor, o que justifica o uso desta técnica.

Eliminação múltipla

Outra técnica utilizada por alguns algoritmos para reduzir a quantidade de atualização de graus é a eliminação múltipla. Esta técnica elimina múltiplos *supernodes* com o corrente mínimo grau antes que uma completa atualização dos graus seja feita. O algoritmo escolhe o conjunto maximal de vértices independentes, ou seja, um conjunto de vértices não adjacentes que seja maximal para ser eliminado em uma iteração.

É importante destacar que esta modificação pode não resultar em uma mesma ordenação provida pelo algoritmo original do mínimo grau. Pelo grafo da Figura 5.16, pode-se ver que o mínimo grau é 2 e os nós com este grau são $\{a, b, d, f, g\}$. Considere a eliminação do nó a. O nó b é indistingüível de a, então a e b fazem parte de um *supernode* e são eliminados em massa. O menor grau corrente passou a ser 1 e somente c possui este grau. Desta forma, a seqüência de eliminação é mostrada na Tabela 5.2.

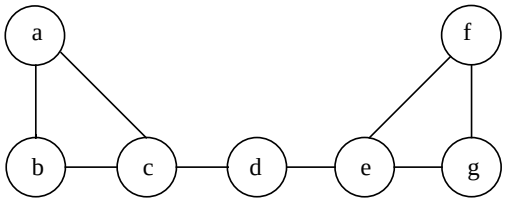


Figura 5.16: Grafo de eliminação.

Iteração	Nós eliminados
1	$\{a, b\}$
2	c
3	d
4	$\{e, f, g\}$

Tabela 5.2: Seqüência de eliminação usando eliminação simples.

Por outro lado, se for aplicada a eliminação múltipla, os nós $\{a, b\}$, d e $\{f, g\}$ de grau 2 serão eliminados na mesma iteração. A nova seqüência de eliminação é vista na Tabela 5.3, onde foram gastas apenas 2 iterações e os graus dos nós sofreram menos atualizações.

Iteração	Nós eliminados
1	$\{a, b\}, d, \{f, g\}$
2	c, e

Tabela 5.3: Seqüência de eliminação usando eliminação múltipla.

Conjunto alcançável

Uma modelagem muito utilizada para representar um conjunto adjacente a um nó pertencente a um grafo de eliminação é conhecida como conjunto alcançável (*reachable set*). A partir de um subconjunto S de nós de um grafo, com $x \notin S$, o nó x é alcançável por y através de S se existe um caminho $(y, v_1, \ldots, v_k, x)$ de y a x tal que $v_i \in S$ para $i = 1, \ldots, k$ e $k > 0$. Se k fosse zero, o conjunto alcançável seria equivalente ao conjunto de vértices adjacentes $adj(y)$.

O conjunto alcançável de y através de S é definido pelo operador *alcance* (*reach*):

$$alcance(y, S) = \{x \notin S | s \text{ é alcançável por } y \text{ através de } S\}.$$

A Figura 5.17 exemplifica esta notação. Supondo o conjunto $S = \{s1, s2, s3, s4\}$, o conjunto alcançável de y é dado por $alcance(y, S) = \{a, b, c\}$, já que existem os caminhos $(y, s2, s4, a)$, (y, b) e $(y, s1, c)$.

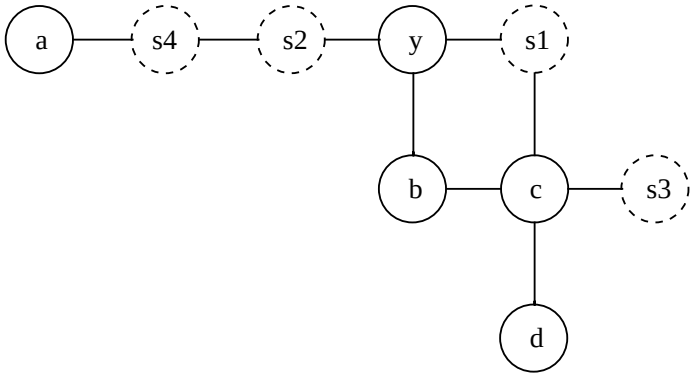


Figura 5.17: Conjunto alcançável.

Para compreender a relação existente entre conjunto alcançável e a modelagem de uma eliminação, considere o grafo numerado da Figura 5.18. Esta numeração representa a seqüência de nós a serem eliminados.

Definindo-se $S_i = \{x_1, \ldots, x_{i-1}\}$ como o conjunto de nós já eliminados na iteração i, não

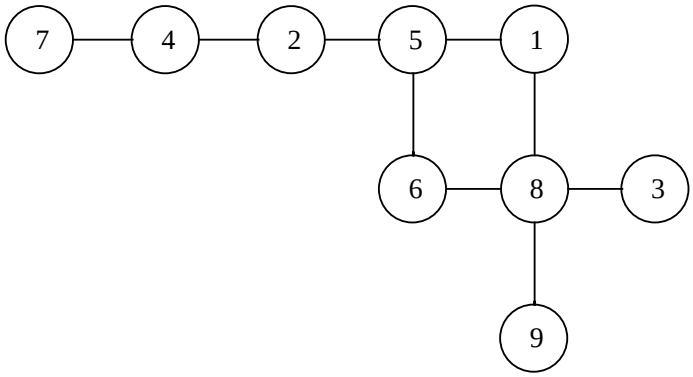


Figura 5.18: Grafo numerado.

é difícil ver pela definição de conjunto alcançável que:

$$\begin{aligned}
alcance(x_1, S_0) &= \{x_5, x_8\} \\
alcance(x_2, S_1) &= \{x_4, x_5\} \\
alcance(x_3, S_2) &= \{x_8\} \\
alcance(x_4, S_3) &= \{x_5, x_7\} \\
alcance(x_5, S_4) &= \{x_6, x_7, x_8\} \\
alcance(x_6, S_5) &= \{x_7, x_8\} \\
alcance(x_7, S_6) &= \{x_8\} \\
alcance(x_8, S_7) &= \{x_9\} \\
alcance(x_9, S_8) &= \phi.
\end{aligned}$$

Desta forma, é possível definir o conjunto de nós adjacentes a um nó y pertencente a um grafo de eliminação $G_i = (X_i, E_i)$ como:

$$alcance(y, \{x_1, \ldots, x_i\}),$$

onde o operador *alcance* é aplicado ao grafo original G_0.

Absorção de elementos

Os nós eliminados de um grafo, também conhecidos como elementos, são utilizados no cálculo do conjunto alcançável de um nó. Por exemplo, considerando-se o grafo da Figura 5.18 após a eliminação dos nós $\{x_1, x_2, x_3, x_4, x_5\}$, seu grafo de eliminação correspondente é dado pela Figura 5.19.

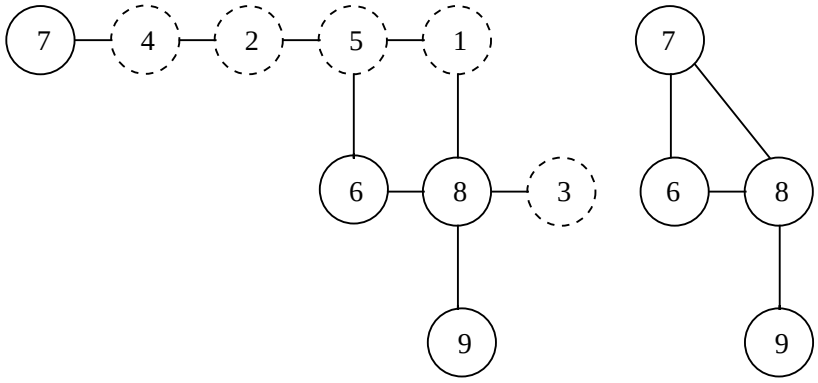


Figura 5.19: Resultado da eliminação dos nós $\{x_1, x_2, x_3, x_4, x_5\}$.

Sendo $S = \{x_1, x_2, x_3, x_4, x_5\}$, pode-se ver que $x_6 \in alcance(x_7, S)$, já que existe o caminho $(x_7, x_4, x_2, x_5, x_6)$. Da mesma forma, $x_8 \in alcance(x_7, S)$ devido ao caminho $(x_7, x_4, x_5, x_1, x_8)$. O tamanho dos caminhos é 4 e 5, respectivamente. Assim, duas observações podem ser feitas:

- A quantidade de trabalho para gerar os conjuntos alcançáveis pode ser reduzida se o tamanho dos caminhos para os nós não eliminados for pequeno.

- Se estes caminhos forem diminuídos ao extremo, ter-se-á a própria modelagem do grafo de eliminação onde não faz sentido o uso de conjunto alcançável.

Por estes motivos, uma técnica conhecida como absorção de elementos (*element absorption*) realiza a união entre os nós eliminados que sejam conectados para formar um único nó, fazendo com que o tamanho do conjunto alcançável fique pequeno e mais fácil de ser calculado. Por exemplo, o novo grafo que representa a Figura 5.19 seria dado pela Figura 5.20.

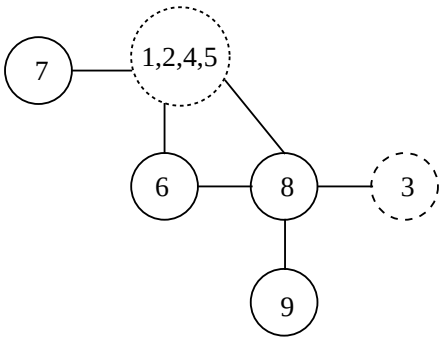


Figura 5.20: Absorção de elementos.

Por conveniência, o algoritmo usa $x_5 = \{x_1, x_2, x_4, x_5\}$ e $x_3 = \{x_3\}$ para denotar estes componentes conectados em S. Com este novo grafo, os novos caminhos são (x_7, x_5, x_6) e (x_7, x_5, x_8), que possuem tamanho igual a 2. Assim, um novo tipo de grafo pode ser usado para modelar a eliminação de nós. Neste grafo, que será apresentado na próxima seção, cada clique formado durante a eliminação é representado por um elemento composto pela absorção dos nós eliminados e não mais por arestas explícitas. Esta representação tem um grande impacto no gerenciamento de dados e na quantidade de memória utilizada pelo algoritmo.

Grafo quociente

O modelo de eliminação implementado pelos algoritmos envolve o conceito de grafo quociente (ou grafo de elementos). O uso deste tipo de grafo introduz uma forma mais compacta e elegante de representação da seqüência de mudanças no grafo de eliminação.

Um grafo quociente $\mathcal{G}$ consiste de dois tipo de nós: *snodes* (ou *supernodes*) e *enodes* (ou elementos). Inicialmente, ele é idêntico ao grafo de eliminação G_0 e consiste apenas de *snodes*. Quando um *snode* é eliminado, ele não é removido do grafo quociente, mas se torna um *enode* (*supernode* eliminado). O conjunto de *snodes* adjacentes a um *snode* r é representado por $sadj(r)$ e o conjunto de *enodes* adjacentes a r é representado por $eadj(r)$. Assim, no grafo quociente, tem-se:

$$adj_{\mathcal{G}}(r) = sadj(r) \cup eadj(r).$$

Usando esta nova notação, o conjunto alcançável de um *snode* r é a união dos $sadj(r)$ e dos *snodes* que podem ser alcançados através de caminhos consistindo apenas de seus *enodes* adjacentes, o que corresponde aos vizinhos no grafo de eliminação G:

$$alcance_{\mathcal{G}}(r) = sadj(r) \cup (\cup_{e \in eadj(r)} sadj(e)).$$

Conseqüentemente, para determinar o próximo vértice a ser eliminado no algoritmo, o tamanho dos conjuntos alcançáveis de todos os *snodes* candidatos deve ser calculado. Para que isto seja feito com uma maior eficiência, a técnica de absorção de elementos é usada para que o caminho composto apenas por *enodes* fique pequeno.

A eliminação de r pode causar mudanças nos conjuntos de adjacências de outros *snodes*:

- Se dois *snodes* tornam-se indistingüíveis, eles também são absorvidos em um único *snode*.

- Se dois *snodes* adjacentes r e s têm um mesmo *enode* como vizinho, então a aresta que une os dois pode ser retirada do grafo, já que existe uma redundância (ambos são alcançáveis através do *enode*).

Este novo modelo representa o processo de eliminação como uma seqüência de grafos quociente. Na Figura 5.21 é possível ver esta seqüência. O algoritmo inicia com o grafo quociente original $\mathcal{G}_0$, onde o conjunto alcançável de cada *snode* candidato a eliminação deve ser calculado. Como o *snode* 1 possui o menor conjunto alcançável, deverá ser eliminado.

Com a eliminação do *snode* 1, o novo grafo quociente $\mathcal{G}_1$ é formado. Neste grafo, o nó eliminado se torna um *enode*. Novamente, o conjunto alcançável dos candidatos a eliminação é calculado e o *snode* 2 é o próximo escolhido.

Com a eliminação do *snode* 2 no grafo $\mathcal{G}_2$, a técnica de absorção de elementos é aplicada ao *enode* 1, o qual é absorvido pelo novo *enode* 2. Os *snodes* 5 e 6 se tornam indistingüíveis, formando um único *supernode*. Além disto, a aresta redundante que conectava os *snodes* $\{5, 6\}$ ao 4 é retirada do grafo, já que ambos possuem o mesmo *enode* 2 como vizinho. Novamente é feita a escolha do próximo nó a ser eliminado e o *snode* 3 é o de menor conjunto alcançável.

Finalmente, no último grafo quociente $\mathcal{G}_3$ temos a eliminação do *snode* 3. Um novo *supernode* formado por 4, 5 e 6 é formado e torna-se o único candidato a eliminação.

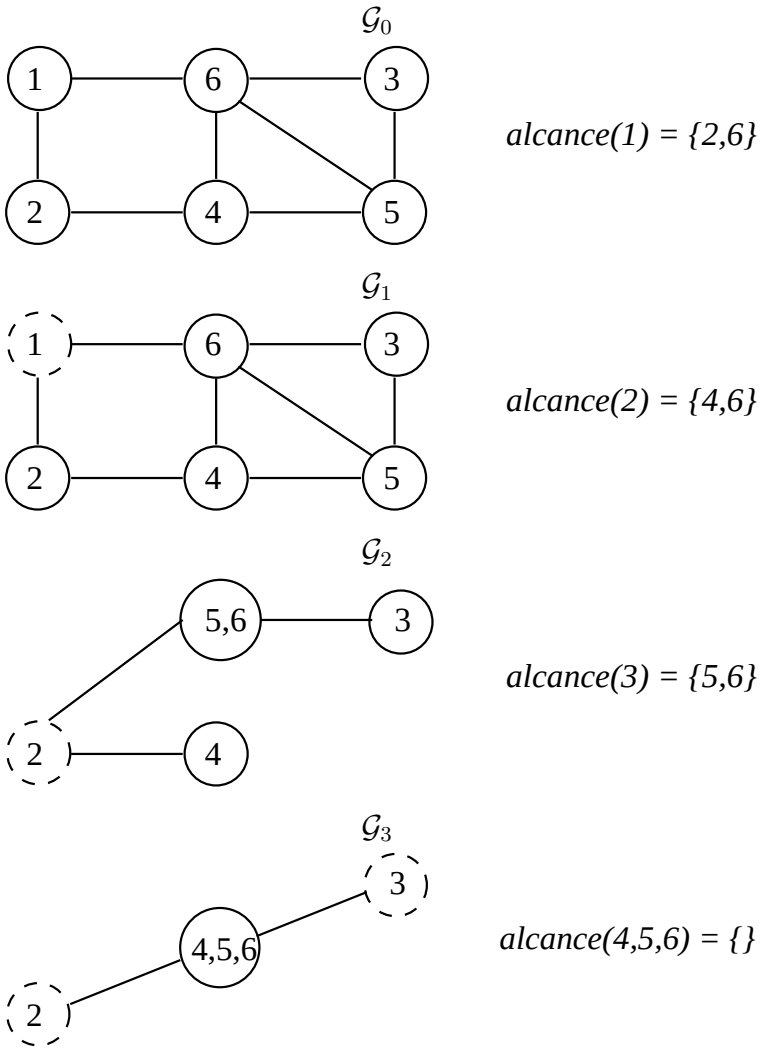


Figura 5.21: Eliminação por meio de grafos quociente.

5.3.2 Mínimo grau original

O algoritmo original do MD usando grafo quociente será apresentado nesta seção. Ele usa algumas das técnicas vistas na seção anterior: eliminação em massa, grau externo, conjunto alcançável e absorção de elementos.

Após a eliminação do nó u_k na iteração k, os graus dos nós vizinhos ao nó eliminado precisam ser atualizados. No grafo quociente, isto corresponde ao $alcance_{\mathcal{G}_{k-1}}(u_k)$, ou seja, torna-se necessário calcular o conjunto alcançável do *snode* a ser eliminado. Após a eliminação, os *snodes* do conjunto alcançável verificam seus próprios conjuntos alcançáveis para calcular seus novos graus.

A cada etapa k, quando o *snode* u_k é eliminado em $\mathcal{G}_{k-1}$, a atualização dos graus é realizada da seguinte forma:

- O *enodes* adjacentes a u_k são absorvidos por este nó formando um único *enode*;
- O novo conjunto alcançável de u_k é calculado adicionando-se os *snodes* adjacentes aos *enodes* absorvidos por ele;
- Para cada *snode* r no conjunto alcançável de u_k, cada um dos seus *snodes* vizinhos são contados exatamente uma vez;
- Os *snodes* adjacentes aos *enodes* vizinhos a r também são contados exatamente uma vez.

O algoritmo pode ser descrito na Figura 5.22. O número de arestas examinadas durante a execução do algoritmo é dada pela seguinte expressão [HEKP01]:

$$O\left(\sum_{k=1}^{n_p}\left(|sadj(u_k)| + \sum_{e\in eadj(u_k)}|sadj(e)| + \sum_{r\in alcance(u_k)}\left(|eadj(r)| + \sum_{e\in eadj(r)}|sadj(e)|\right)\right)\right),$$

onde n_p é o número de *supernodes* eliminados ($\leq$ número de nós do grafo) e u_k corresponde ao *supernode* eliminado na iteração k.

Nesta análise de complexidade, cada termo da expressão corresponde às partes 1a, 1b, 2a e 2b do algoritmo. Seja n o número de nós do grafo e m o número de arestas. As adjacências de todos os nós do grafo é $O(m)$ e a soma dos *snodes* adjacentes aos *enodes* examinados a cada iteração também é $O(m)$. O conjunto alcançável é limitado por $O(n)$ e o número de arestas examinadas por este *alcance* é $O(m)$. Com isto, a ordem de complexidade do algoritmo é $O(n(m+nm)) = O(n^2m)$.

Algoritmo MD
{ **Objetivo:** Realizar a ordenação por mínimo grau }
$\mathcal{G}_0 = G$;
Calcular *supernodes* iniciais e seus *pesos*;
Calcular graus iniciais;
$visita \leftarrow 0;\ k \leftarrow 0;\ t \leftarrow 0;$
enquanto existem *snodes* em $\mathcal{G}_k$ **faça**
$k \leftarrow k + 1$;
Escolher *snode* u_k de mínimo grau;
Substituir *snode* u_k pelo *enode* u_k;
{ 1. Encontrar conjunto alcançável de u_k }
$t \leftarrow t + 1;\ alcance \leftarrow \{\}$;
{ 1a. Incluir *snodes* adjacentes a u_k no conjunto alcançável }
para cada *snode* $r \in sadj(u_k)$ **faça**
$visita(r) \leftarrow t;\ alcance \leftarrow alcance \cup r$;
{ 1b. Incluir *snodes* vizinhos aos *enodes* adjacentes a u_k no conjunto alcançável }
para cada *enode* $e \in eadj(u_k)$ **faça**
para cada *snode* $r \in sadj(e)$ com $visita(r) < t$ **faça**
$visita(r) \leftarrow t;\ alcance \leftarrow alcance \cup r$;
u_k absorve e formando um único *enode*;
Calcular novos *supernodes*;
Atualizar grafo quociente formando $\mathcal{G}_k$;
{ 2. Atualizar graus dos *snodes* pertencentes ao conjunto alcançável de u_k }
para cada *snode* $r \in alcance$ **faça**
$t \leftarrow t + 1;\ visita(r) \leftarrow t;\ grau(r) \leftarrow 0;$
{ 2a. Examinar *snodes* adjacentes a r }
para cada *snode* $s \in sadj(r)$ **faça**
$visita(s) \leftarrow t;\ grau(r) \leftarrow grau(r) + peso(s)$;
{ 2b. Examinar *enodes* adjacentes a r e os *snodes* vizinhos a estes *enodes* }
para cada *enode* $e \in eadj(r)$ **faça**
para cada *snode* $s \in sadj(e)$ com $visita(s) < t$ **faça**
$visita(s) \leftarrow t;\ grau(r) \leftarrow grau(r) + peso(s)$;
$n_p \leftarrow k$;

Figura 5.22: Algoritmo do mínimo grau.

5.3.3 Mínimo grau múltiplo

O algoritmo mínimo grau múltiplo (MMD: *multiple minimum degree*), foi originalmente proposto por Liu [Liu85]. Além das técnicas utilizadas pelo MD, ele utiliza a eliminação múltipla para aumentar a eficiência.

A cada passo i do algoritmo, um conjunto independente K_i de vértices de mínimo grau é encontrado. Eles são eliminados e os vértices adjacentes a eles são marcados como vértices cujos graus precisam ser atualizados. No grafo quociente, estes vértices que precisam ser atualizados consistem da união dos conjuntos alcançáveis de todos os *snodes* em K_i. Se estes conjuntos alcançáveis tiverem *snodes* em comum, serão feitas menos atualizações que no algoritmo original MD. Os graus de todos os vértices marcados são atualizados apenas depois da eliminação. Porém, se todos os conjuntos independentes tiverem tamanho um, o trabalho gasto pelo MMD é igual ao MD.

Como são usados graus externos para *snodes*, a eliminação de um *snode* do conjunto independente K pode tornar o grau externo de um *snode* fora de K menor que o grau externo dos *snodes* restantes de K. Neste caso, uma pequena perturbação na ordenação poderá ocorrer, como foi discutido anteriormente nas tabelas 5.2 e 5.3. Entretanto, na prática, a qualidade do MMD é melhor que a do MD em relação ao preenchimento.

O algoritmo está representado pela Figura 5.23. Novamente, o número de arestas examinadas durante a execução do algoritmo é dada pela seguinte expressão [HEKP01]:

$$O\left(\sum_{k=1}^{n_h}\left(\sum_{k\in K_i}|sadj(k)| + \sum_{e\in eadj(K_i)}|sadj(e)| + \sum_{r\in alcance(K_i)}\left(|sadj(r)| + \sum_{e\in eadj(r)}|sadj(e)|\right)\right)\right),$$

onde n_h é o número de conjuntos independentes de *supernodes* eliminados ($\leq$ número de nós do grafo) e k corresponde a um dos *supernodes* contidos no conjunto independente K_i eliminado na iteração i.

Cada termo da expressão acima corresponde as partes 1a, 1b, 3a e 3b do algoritmo. A análise é similar a do agoritmo MD. No máximo $O(n)$ *snodes* podem estar no conjunto alcançável, fazendo com que a ordem de complexidade do algoritmo também seja $O(n(m + nm)) = O(n^2m)$ para o pior caso.

Algoritmo MMD
{ **Objetivo:** Realizar a ordenação por mínimo grau múltiplo }
$\mathcal{G}_0 = G$;
Calcular *supernodes* iniciais e seus *pesos*;
Calcular graus iniciais;
$visita \leftarrow 0;\ i \leftarrow 0;\ t \leftarrow 0;$
enquanto existem *snodes* em $\mathcal{G}_i$ **faça**
$i \leftarrow i + 1$;
Escolher o conjunto independente de *snodes* de mínimo grau K_i;
$t \leftarrow t + 1;\ alcance \leftarrow \{\}$;
{ 1. Eliminar *snodes* em K_i }
{ *alcance* é a união dos conjuntos alcançáveis destes *snodes* }
para cada *snode* $k \in K_i$ **faça**
$visita(k) \leftarrow \infty$;
para cada *snode* $r \in sadj(k)$ **faça** { 1a }
$visita(r) \leftarrow t;\ alcance \leftarrow alcance \cup r$;
para cada *enode* $e \in eadj(k)$ **faça** { 1b }
para cada *snode* $r \in sadj(e)$ com $visita(r) < t$ **faça**
$visita(r) \leftarrow t;\ alcance \leftarrow alcance \cup r$;
{ 2. Atualizar grafo quociente após a eliminação dos *snodes* em K_i }
para cada *snode* $k \in K_i$ **faça**
Substituir *snode* k pelo *enode* k;
k absorve $eadj(k)$ formando um único *enode*;
Calcular novos *supernodes*;
Formar $\mathcal{G}_i$;
{ 3. Atualizar graus dos *snodes* pertencentes ao conjunto alcançável }
para cada *snode* $r \in alcance$ **faça**
$t \leftarrow t + 1;\ visita(r) \leftarrow t;\ grau(r) \leftarrow 0;$
para cada *snode* $s \in sadj(r)$ **faça** { 3a }
$visita(s) \leftarrow t;\ grau(r) \leftarrow grau(r) + peso(s)$;
para cada *enode* $e \in eadj(r)$ **faça** { 3b }
para cada *snode* $s \in sadj(e)$ com $visita(s) < t$ **faça**
$visita(s) \leftarrow t;\ grau(r) \leftarrow grau(r) + peso(s)$;
$n_h \leftarrow i$;

Figura 5.23: Algoritmo do mínimo grau múltiplo.

5.3.4 Mínimo grau aproximado

Tal como o MD, o algoritmo do mínimo grau aproximado (AMD: *approximate minimum degree*) [ADD96] não utiliza a eliminação múltipla do MMD. A idéia do AMD é calcular um limite superior para os graus dos nós a serem atualizados a um custo mais baixo do que o cálculo do grau externo correto. Este limite superior é definido como grau aproximado *agrau* e é usado na escolha do *snode* a ser eliminado.

Sabe-se que o peso de um *snode* consiste no número de nós indistingüíveis que compõem este *supernode*. Pode-se definir o peso de um *enode* e como a soma dos pesos dos *snodes* adjacentes a ele no grafo quociente corrente, ou seja:

$$peso(e) = \sum_{s \in sadj(e)} peso(s).$$

Seja r um *snode* cujo grau precisa ser atualizado. O grau de r não pode ser maior que a soma dos pesos de todos os *snodes* e *enodes* adjacentes a ele no grafo quociente corrente. Porém, o conjunto dos *snodes* adjacentes aos *enodes* de $eadj(r)$ pode se sobrepor, fazendo com que o limite se torne muito acima do real e causando um *gap*. Este *gap* pode ser reduzido calculando-se um valor *dif(e)* associado com cada *enode* para remover algumas destas sobreposições nos conjuntos adjacentes.

Seja u_k o nó a ser eliminado no passo k. Após absorver todos os seus *enodes* vizinhos, seu novo peso torna-se a soma dos pesos de todos os *snodes* $r \in alcance_{\mathcal{G}_{k-1}}(u_k)$:

$$peso(u_k) = \sum_{r \in alcance_{\mathcal{G}_{k-1}}(u_k)} peso(r).$$

Como cada *snode* r no conjunto alcançável acima é um vizinho do *enode* u_k em $\mathcal{G}_k$, o valor $peso(u_k)$ será adicionado ao grau aproximado de r. Assim, para todos os *enodes* $e \in eadj(r)$ onde $e \neq u_k$, incluímos no $peso(e)$ apenas a contribuição dos pesos dos *snodes* disjuntos daqueles que estão no conjunto alcançável.

A função *dif(e)* para *enodes* $e \in eadj(alcance_{\mathcal{G}_{k-1}}(u_k))$ no grafo quociente $\mathcal{G}_k$ pode ser definida por:

$$dif(e) = \begin{cases} peso(e) & \text{se } e = u_k, \\ peso(e) - \sum_{r \in (alcance_{\mathcal{G}_{k-1}}(u_k) \cap sadj(e))} peso(r) & \text{se } e \neq u_k. \end{cases}$$

O grau aproximado de $r \in alcance_{\mathcal{G}_{k-1}}(u_k)$ é calculado da seguinte forma:

$$agrau(r) = peso(u_k) + \sum_{s \in sadj(r)} peso(s) + \sum_{e \in eadj(alcance(u_k))} dif(e).$$

O algoritmo é apresentado na Figura 5.24. Devido a grande dificuldade em se encontrar interseção entre conjuntos, a eliminação múltipla normalmente não é implementada no AMD. Assim, o número total de iterações do algoritmo é a seguinte [HEKP01]:

$$O\left(\sum_{k=1}^{n_p}\left(|adj(u_k)| + \sum_{e\in eadj(u_k)}|sadj(e)| + \sum_{r\in alcance(u_k)}|eadj(r)|\right)\right),$$

onde n_p é o número de *supernodes* eliminados ($\leq$ número de nós do grafo) e u_k corresponde ao *supernode* eliminado na iteração k.

Na expressão acima, o segundo e o terceiro termos juntos possuem uma complexidade da ordem $O(m)$. Assim, a complexidade do algoritmo AMD é $O(n(m+m)) = O(nm)$, que é menor que a complexidade dos algoritmos MD e MMD ($O(n^2m)$). Por isto, o AMD foi escolhido para reordenação das matrizes de testes neste trabalho.

Algoritmo AMD

$\mathcal{G}_0 = G$;
Calcular *supernodes* iniciais e seus *pesos*;
para cada *snode* $r \in \mathcal{G}_0$ **faça**
  $grau(r) \leftarrow 0$;
  para cada *snode* $s \in sadj(r)$ **faça**
    $grau(r) \leftarrow grau(r) + peso(s)$;
$visita \leftarrow 0;\ k \leftarrow 0;\ t \leftarrow 0;$
enquanto existem *snodes* em $\mathcal{G}_k$ **faça**
  $k \leftarrow k + 1$;
  Escolher *snode* de mínimo grau aproximado u_k;
  $kpeso \leftarrow peso(u_k)$;
  Substituir *snode* u_k pelo *enode* u_k;
  $t \leftarrow t + 1;\ alcance \leftarrow \{\};\ peso(u_k) \leftarrow 0;$
  { 1a. Incluir *snodes* adjacentes a u_k no conjunto alcançável }
  para cada *snode* $r \in sadj(u_k)$ **faça**
    $visita(r) \leftarrow t;\ alcance \leftarrow alcance \cup r;$
    $peso(u_k) \leftarrow peso(u_k) + peso(r)$;
    $sgrau(r) \leftarrow sgrau(r) - kpeso$;
  { 1b. Incluir *snodes* vizinhos aos *enodes* adjacentes a u_k no conjunto alcançável }
  para cada *enode* $e \in eadj(u_k)$ **faça**
    para cada *snode* $r \in sadj(e)$ com $visita(r) < t$ **faça**
      $visita(r) \leftarrow t;\ alcance \leftarrow alcance \cup r;$
      $peso(u_k) \leftarrow peso(u_k) + peso(r)$;
    u_k absorve e formando um único *enode*;
  Calcular novos *supernodes*; Atualizar $\mathcal{G}_k$; $k \leftarrow k + 1$;
  { 2a. Calcular *dif(e)* para *enodes* adjacentes aos *snodes* do conjunto alcançável }
  para cada *snode* $r \in alcance$ **faça**
    para cada *enode* $e \in eadj(r)$, sendo $e \neq u_k$ **faça**
      se $visita(e) < t$ **então**
        $dif(e) \leftarrow peso(e) - peso(r);\ visita(e) \leftarrow t;$
      senão $dif(e) \leftarrow dif(e) - peso(r)$;
  { 2b. Calcular grau aproximado dos *snodes* pertencentes ao conjunto alcançável }
  para cada *snode* $r \in alcance$ **faça**
    $agrau(r) \leftarrow grau(r) + peso(u_k) - peso(r)$;
    para cada *enode* $e \in eadj(r)$, sendo $e \neq u_k$ **faça**
      $agrau(r) \leftarrow agrau(r) + dif(e)$;
$n_p \leftarrow k$;

Figura 5.24: Algoritmo de mínimo grau aproximado.

Capítulo 6

Influência da reordenação no CCCG(η)

Este capítulo tem como finalidade analisar a influência da reordenação de matrizes esparsas no desempenho método CCCG(η). O efeito de reordenações sobre métodos iterativos precondicionados tem sido discutido por muitos artigos. Normalmente, a primeira escolha baseia-se em algoritmos que são especializados na redução do preenchimento criado pelo precondicionador. Entretanto, como pode ser visto em [DM89], os algoritmos que reduzem a largura de banda da matriz também podem acelerar a convergência da solução, apesar de demandarem um maior espaço de armazenamento. Desta forma, pretende-se verificar situações onde um determinado tipo de reordenação torna o CCCG(η) mais eficiente.

6.1 Descrição da implementação

O sucesso de algoritmos para computação de matrizes esparsas depende crucialmente da qualidade de sua implementação. Por este motivo, estes algoritmos foram analisados detalhadamente nos capítulos anteriores. Toda a implementação foi feita usando a linguagem FORTRAN-77 [Ltd90], já que o objetivo do trabalho é incorporar os algoritmos de reordenação ao código original do CCCG(η) escrito em FORTRAN e disponibilizado pelo autor [Cam95]. Como esta linguagem foi projetada para cálculos matemáticos (suporta números complexos), torna-se muito eficiente no contexto de algoritmos numéricos. Além disto, existem muitos códigos disponíveis escritos em FORTRAN e muitas bibliotecas digitais. Com este ganho em eficiência, foi possível realizar testes com matrizes de grande porte.

O algoritmo de contagem de colunas, o qual utiliza o número de elementos não nulos de cada coluna para realizar a ordenação, foi o mais simples de ser implementado. Como a estrutura de dados utilizada pelo CCCG(η) é a coluna esparsa compactada (seção 4.1.2), a maior dificuldade encontrada na implementação do algoritmo foi contar os elementos da parte triangular superior da matriz. Estes elementos não são armazenados pela estrutura e foi necessário descobrir a informação da sua coluna através da sua simetria com os elementos da matriz triangular inferior para fazer a contagem.

Em relação ao mínimo grau aproximado, foi utilizado o código `AMDBAR` que se encontra disponível no repositório da NETLIB [DADR]. Esta implementação é muito conhecida pela sua eficiência. Foi necessário fazer uma adaptação entre este código e o CCCG(η), já que suas estruturas de armazenamento eram completamente diferentes. Enquanto o CCCG(η) utiliza a coluna esparsa compactada, o `AMDBAR` utiliza uma estrutura que armazena todos os elementos não nulos da matriz, com exceção da diagonal. Devido ao tamanho das matrizes de testes, esta montagem da nova estrutura a partir do formato CCCG foi feita buscando minimizar ao máximo a quantidade de memória necessária.

Quanto ao Cuthill-McKee reverso, o Instituto de Pesquisa *European Synchrotron Radiation Facility* [PD] disponibiliza o código `CUTCHO` que usa a mesma heurística estudada na seção 5.2. Da mesma forma que o mínimo grau, foi necessário montar a estrutura utilizada pelo `CUTCHO` a partir do CCCG, já que todos os elementos não nulos da matriz são utilizados no caminhamento proposto no Cuthill-McKee reverso.

Finalmente, a partir do vetor de ordenação resultante destes três algoritmos, foi necessário criar uma versão especial do *Quicksort* não recursivo para reordenar a matriz, tal como foi visto na seção 4.3. A dificuldade em se reordenar os elementos na estrutura de coluna esparsa compactada está na mudança de posição entre os elementos da parte triangular inferior para a parte triangular superior da matriz e vice-versa. O *Quicksort* teve que ser adaptado para prever a posição exata onde cada elemento deveria ficar, armazenando somente a parte inferior da nova matriz no final da reordenação.

6.2 Descrição das matrizes de testes

Os testes foram realizados utilizando o repositório de matrizes esparsas disponibilizado pela Universidade da Flórida [Dav]. Ele é composto por um grande conjunto de matrizes esparsas com origem em vários tipos de problemas tais como engenharia de estruturas, estatística etc. Compreende, aproximadamente, 1322 matrizes. A menor possui ordem 5 com

19 elementos não nulos e a maior possui ordem 5,1 milhões com 99,2 milhões de elementos.

Dentre as coleções disponíveis, a Harwell-Boeing [DGL92] é a mais utilizada, apesar de que novas coleções maiores estão começando a ser difundidas. A Harwell-Boeing contém matrizes de vários tipos (simétricas, não simétricas, retangulares, reais, complexas), mas apenas matrizes simétricas definidas positivas foram escolhidas.

Na Tabela 6.1 encontram-se as séries de matrizes utilizadas nos testes (formato Harwell-Boeing).

Série	Aplicação
Bcsstruc1	*Dynamic analyses in structural engineering*
Bcsstruc2	*Static analyses in structural engineering*
Bcsstruc3	*Dynamic analyses in structural engineering*
Lanpro	*Linear equations in structural engineering*
Psadmit	*Power system networks*
Boeing	*Structural engineering matrices*
Cunningham	*Finite element matrices*
GHS_psdef	*Positive definite matrices*
Mulvey	*Financial portfolio optimization*
Nasa	*Matrices from NASA - structural engineering problems*
ND	*ND problem set*
Oberwolfach	*Oberwolfach model reduction benchmark collection*

Tabela 6.1: Séries de matrizes de testes.

Entre as matrizes testadas, algumas foram selecionadas e suas características principais encontram-se na Tabela 6.2. Nesta tabela, o tamanho da matriz se refere ao número de elementos não nulos da parte triangular inferior. Na Figura 6.1 tem-se o padrão de esparsidade de algumas matrizes.

Matriz	Ordem	Tamanho	Fonte
nos3	960	8402	Lanpro
nos6	675	1965	Lanpro
nos7	729	2673	Lanpro
494_bus	494	1080	Psadmit
1138_bus	1138	2596	Psadmit
bcsstk06	420	4140	Bcsstruc1
bcsstk08	1074	7017	Bcsstruc1
bcsstk10	1086	11578	Bcsstruc1
bcsstk15	3948	60882	Bcsstruc2
bcsstk16	4884	147631	Bcsstruc2
minsurfo	40806	122214	GHS_psdef
wathen120	36441	301101	GHS_psdef
finan512	74752	335872	Mulvey
nasasrb	54870	1366097	Nasa
gyro_k	17361	519260	Oberwolfach

Tabela 6.2: Propriedades de algumas matrizes de testes.

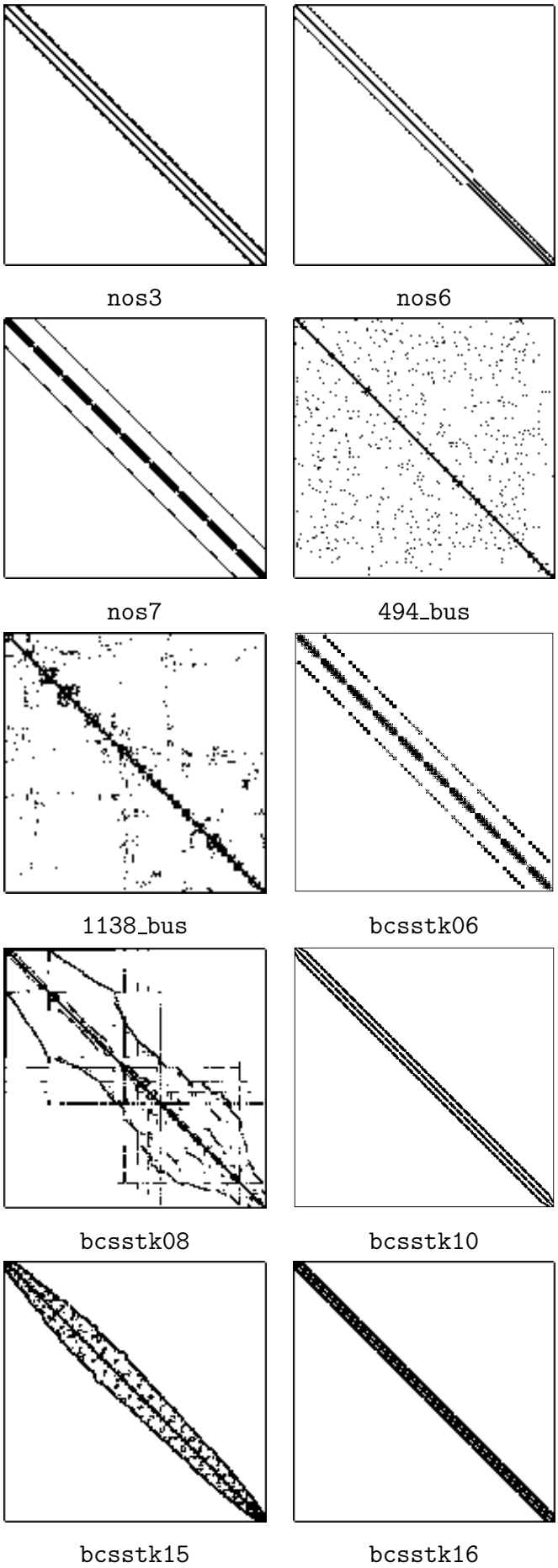


Figura 6.1: Padrão de esparsidade de algumas matrizes.

6.3 Análise de L e $L^{-1}AL^{-T}$

Foram realizados alguns testes preliminares com o propósito de mostrar como o preenchimento do fator L utilizado pelo precondicionador varia com a reordenação. Além disto, também foi feita uma análise sobre o efeito da reordenação na distribuição dos autovalores (autoespectro) da matriz precondicionada $L^{-1}AL^{-T}$. É interessante verificar como o comportamento das características de convergência de algumas matrizes precondicionadas é modificado com a aplicação de algoritmos de reordenação.

Estes testes de simulação foram realizados no MATLAB [Mat00], onde uma versão da fatoração controlada de Cholesky e do gradiente conjugado foi desenvolvida. Além disto, o MATLAB possui as rotinas próprias dos algoritmos de reordenação: *symamd* (mínimo grau aproximado), *symrcm* (Cuthill-McKee reverso) e *colperm* (contagem de colunas). Como o código destas heurísticas não está disponível, é possível que haja alguma diferença em relação aos resultados obtidos pelo código em FORTRAN.

6.3.1 Estrutura de L

O nível de preenchimento da fatoração controlada de Cholesky pode ser modificada pelo parâmetro η, levando a diferentes padrões de esparsidade do fator L criado. É interessante comparar este padrão de esparsidade de uma matriz com a ordenação original e após o uso de algum algoritmo de reordenação.

Na Figura 6.1 tem-se o padrão de esparsidade das matrizes `1138_bus` e `bcsstk06`. A primeira matriz possui estrutura bastante irregular, com os elementos não nulos bem espalhados, e a segunda possui uma estrutura conhecida como banda, onde os elementos se concentram próximos da diagonal. A variação do fator de preenchimento L pode ser vista nas Figuras 6.2, 6.3, 6.4 e 6.5 para a matriz `1138_bus` e nas Figuras 6.6, 6.7, 6.8 e 6.9 para a matriz `bcsstk06`. Cada figura contém quatro matrizes que representam os níveis de preenchimento controlado pelo parâmetro η que são 0, 5, 10 e 20. É fácil perceber como a reordenação da matriz leva a fatores mais esparsos.

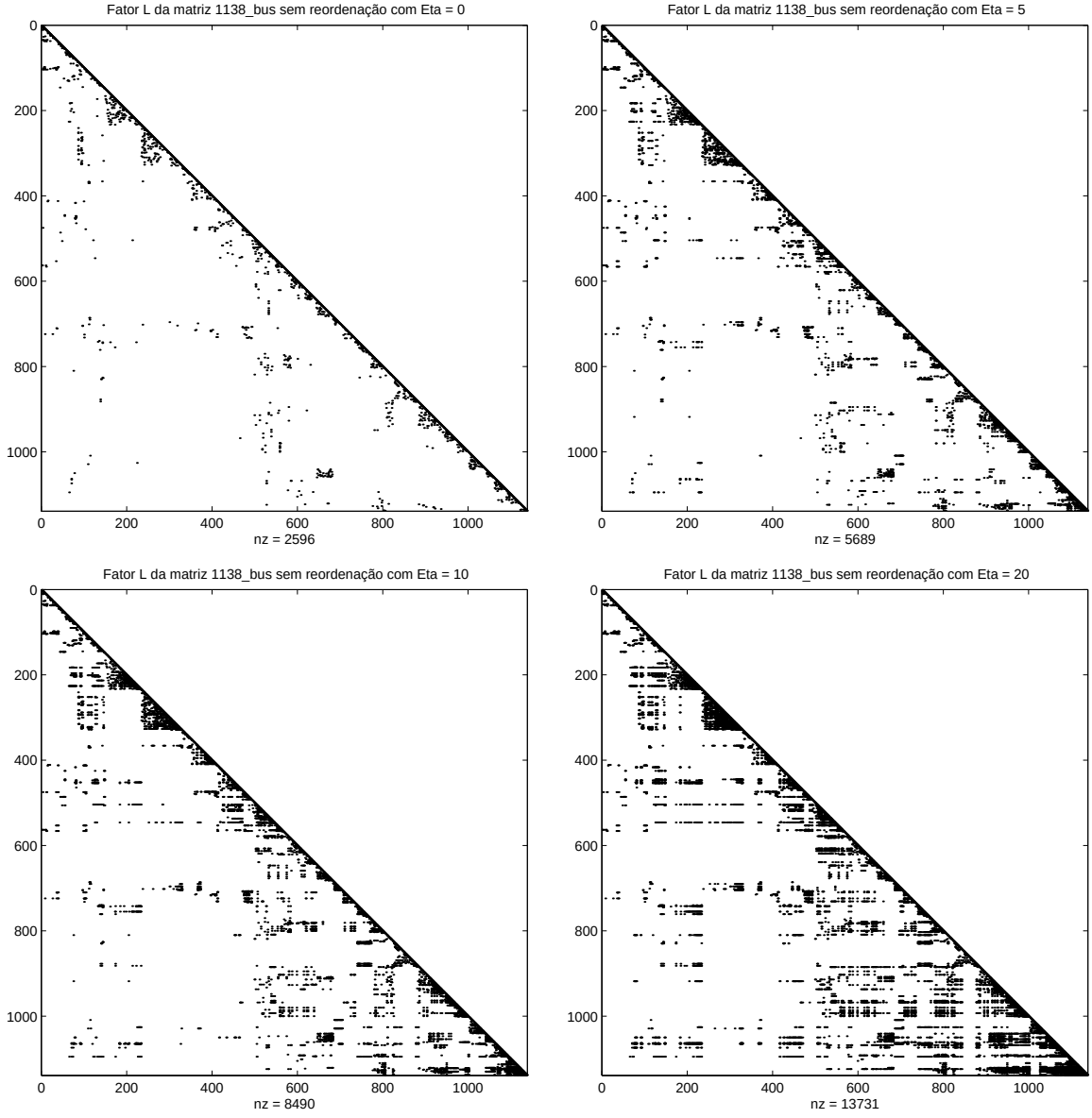


Figura 6.2: Fator L da matriz `1138_bus` sem reordenação.

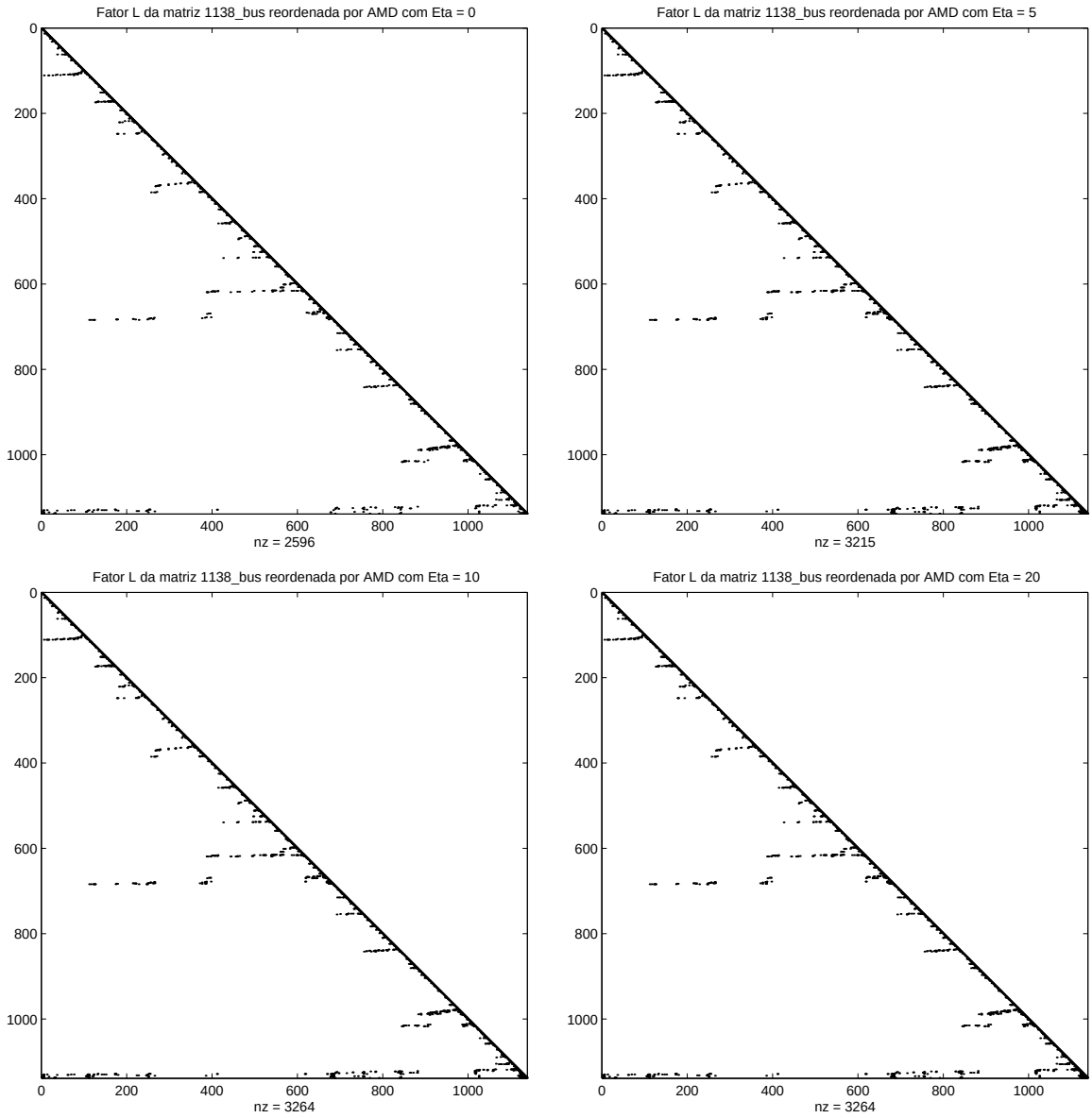


Figura 6.3: Fator L da matriz `1138_bus` reordenada pelo mínimo grau aproximado.

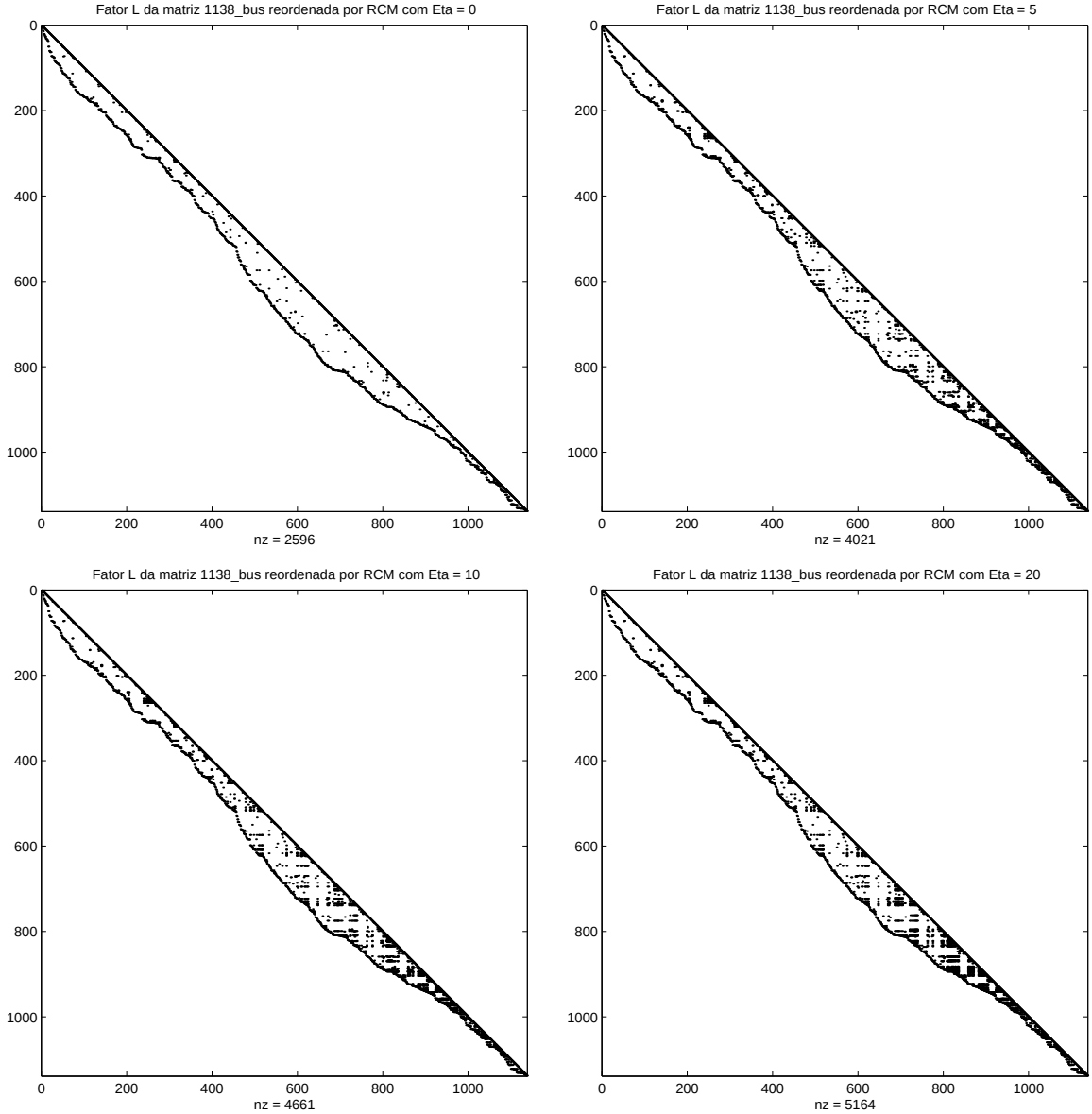


Figura 6.4: Fator L da matriz `1138_bus` reordenada pelo Cuthill-McKee reverso.

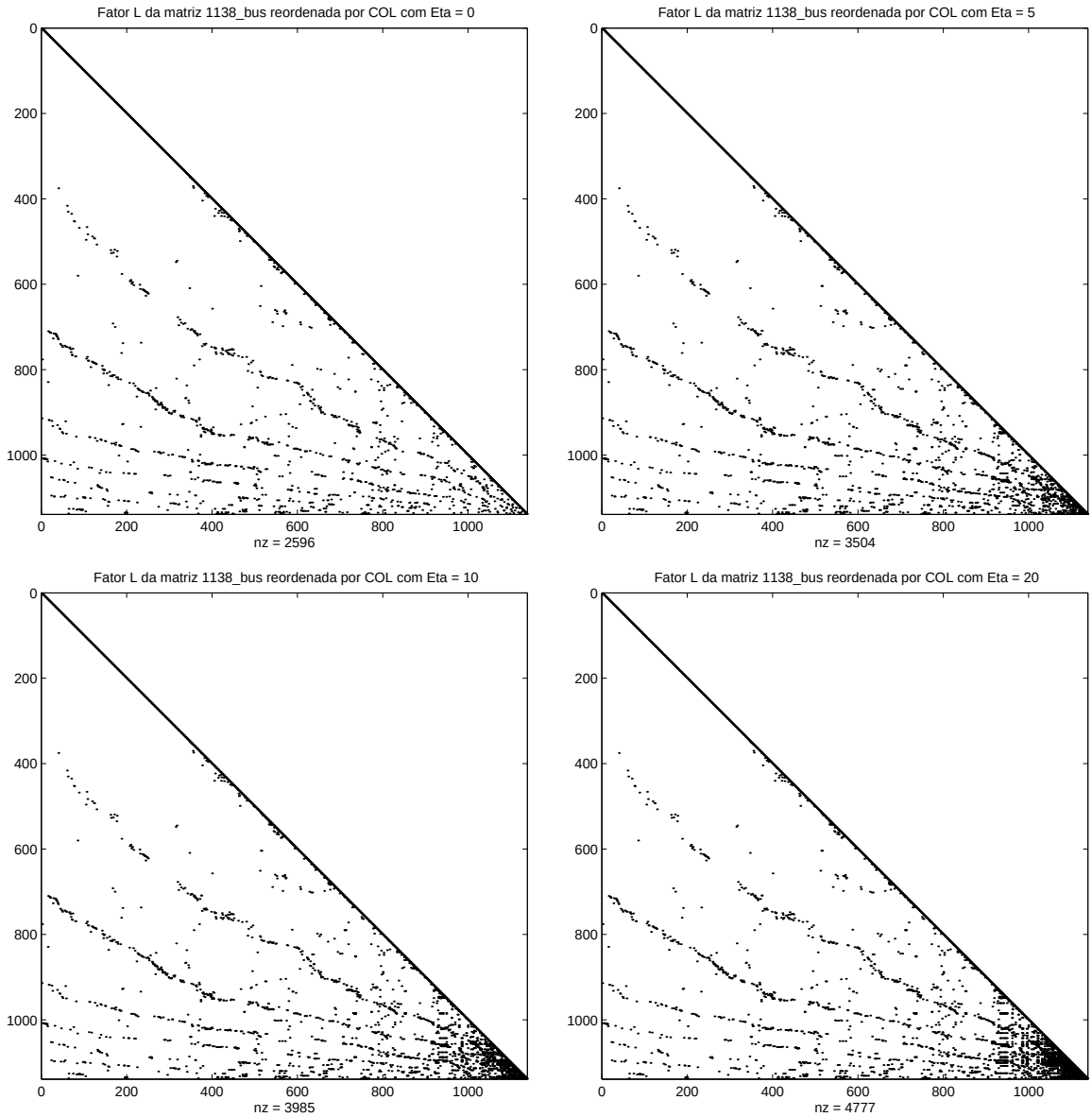


Figura 6.5: Fator L da matriz `1138_bus` reordenada pelo contagem de colunas.

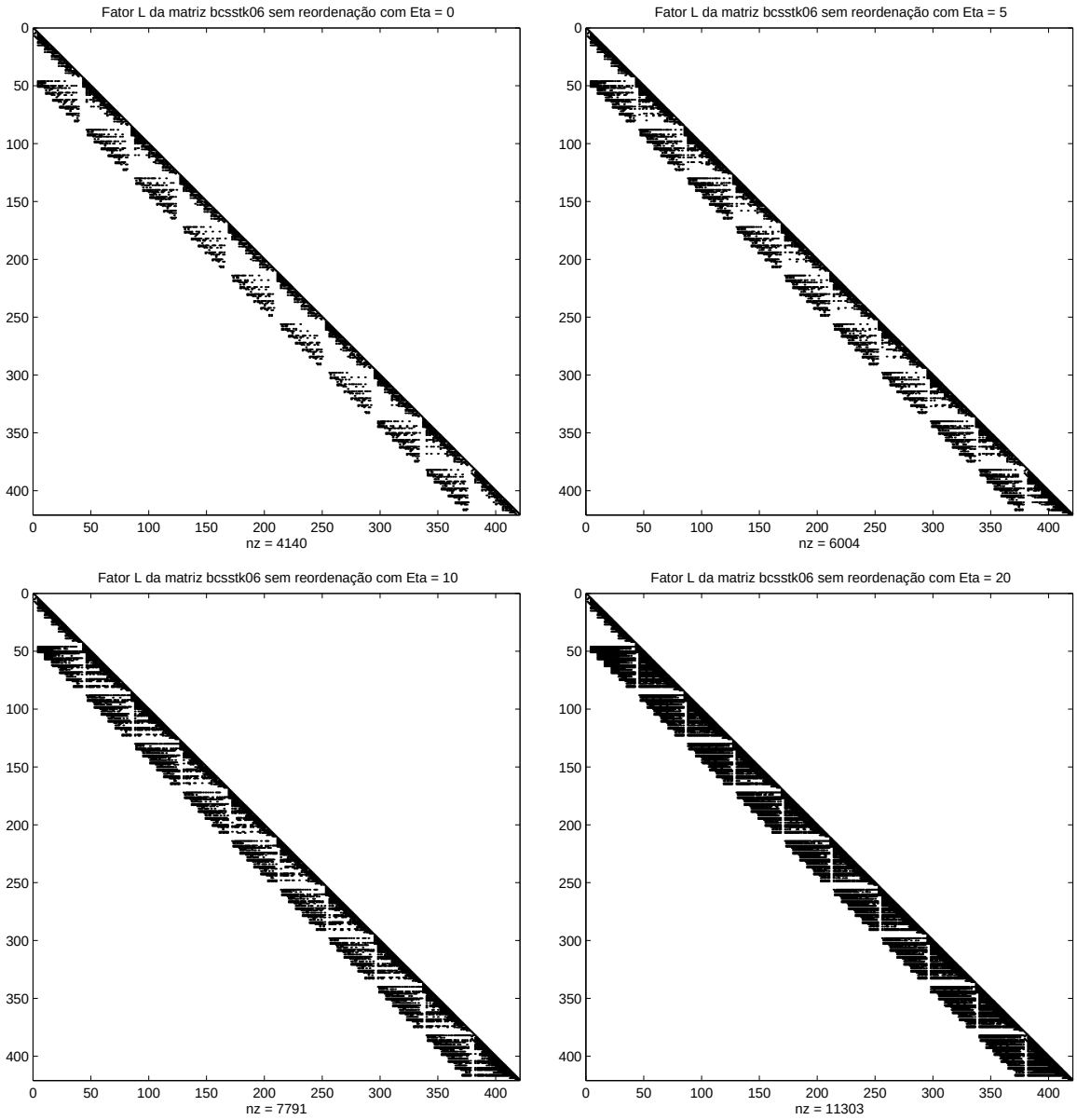


Figura 6.6: Fator L da matriz `bcsstk06` sem reordenação.

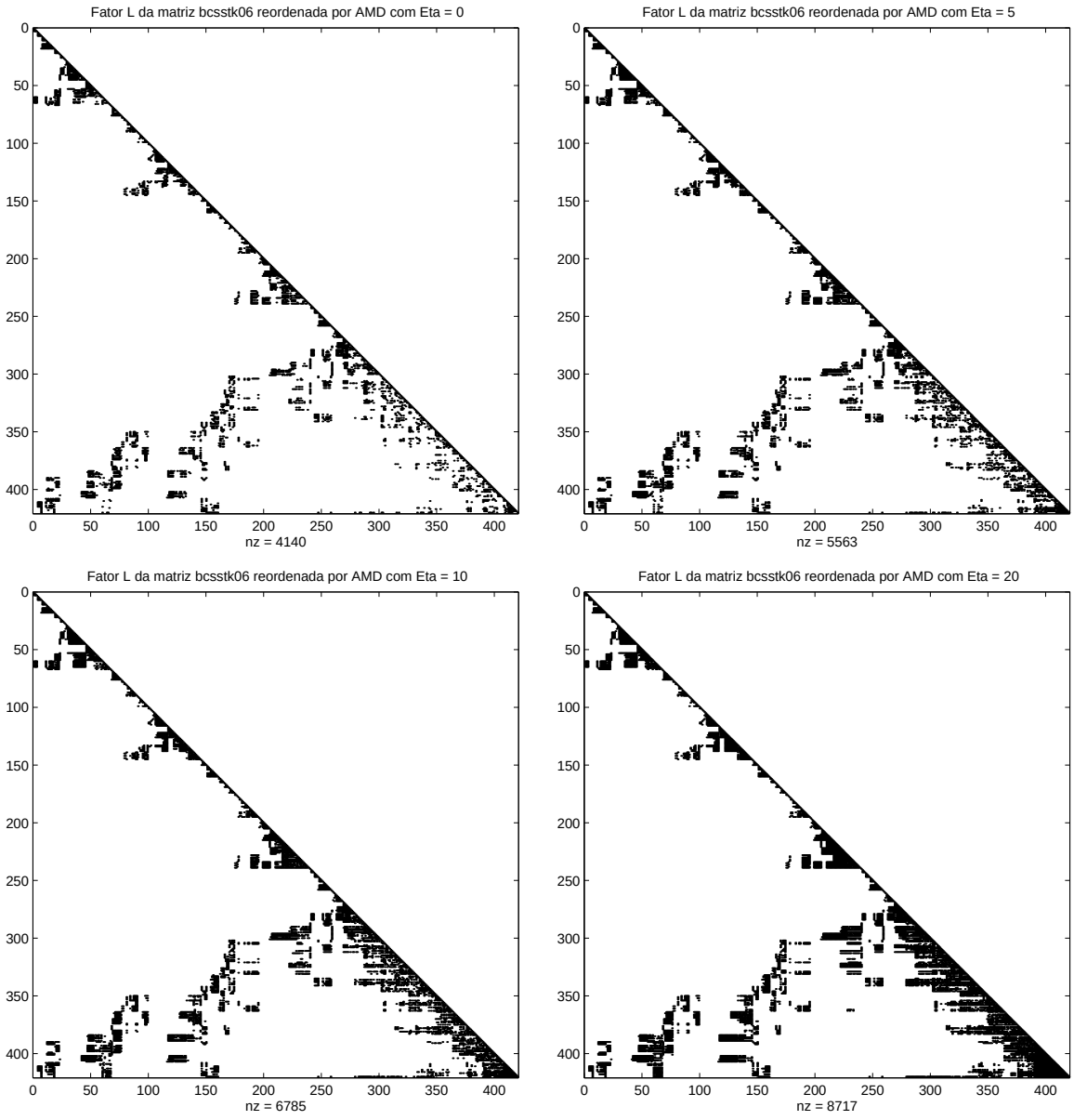


Figura 6.7: Fator L da matriz `bcsstk06` reordenada pelo mínimo grau aproximado.

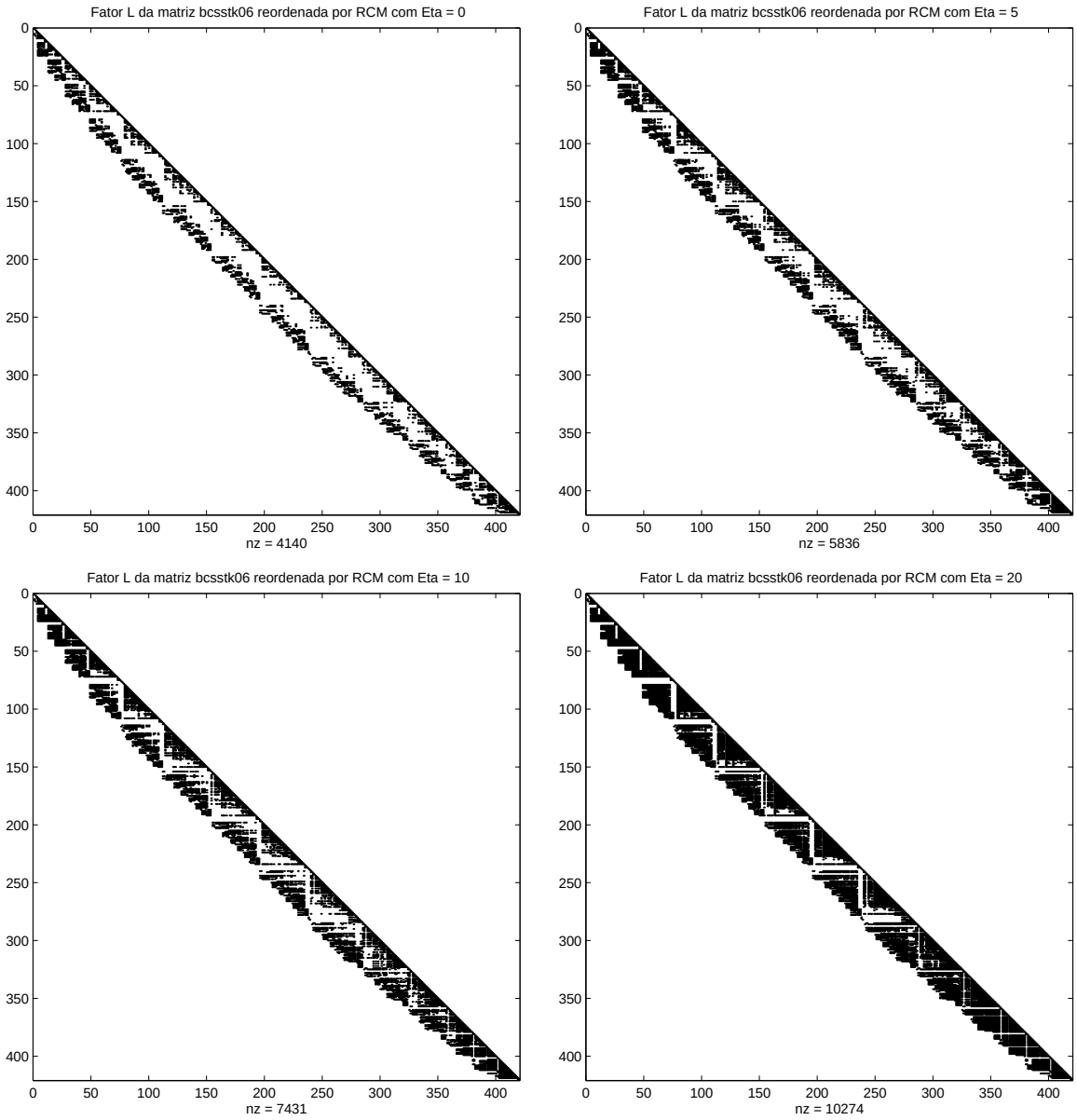


Figura 6.8: Fator L da matriz `bcsstk06` reordenada pelo Cuthill-McKee reverso.

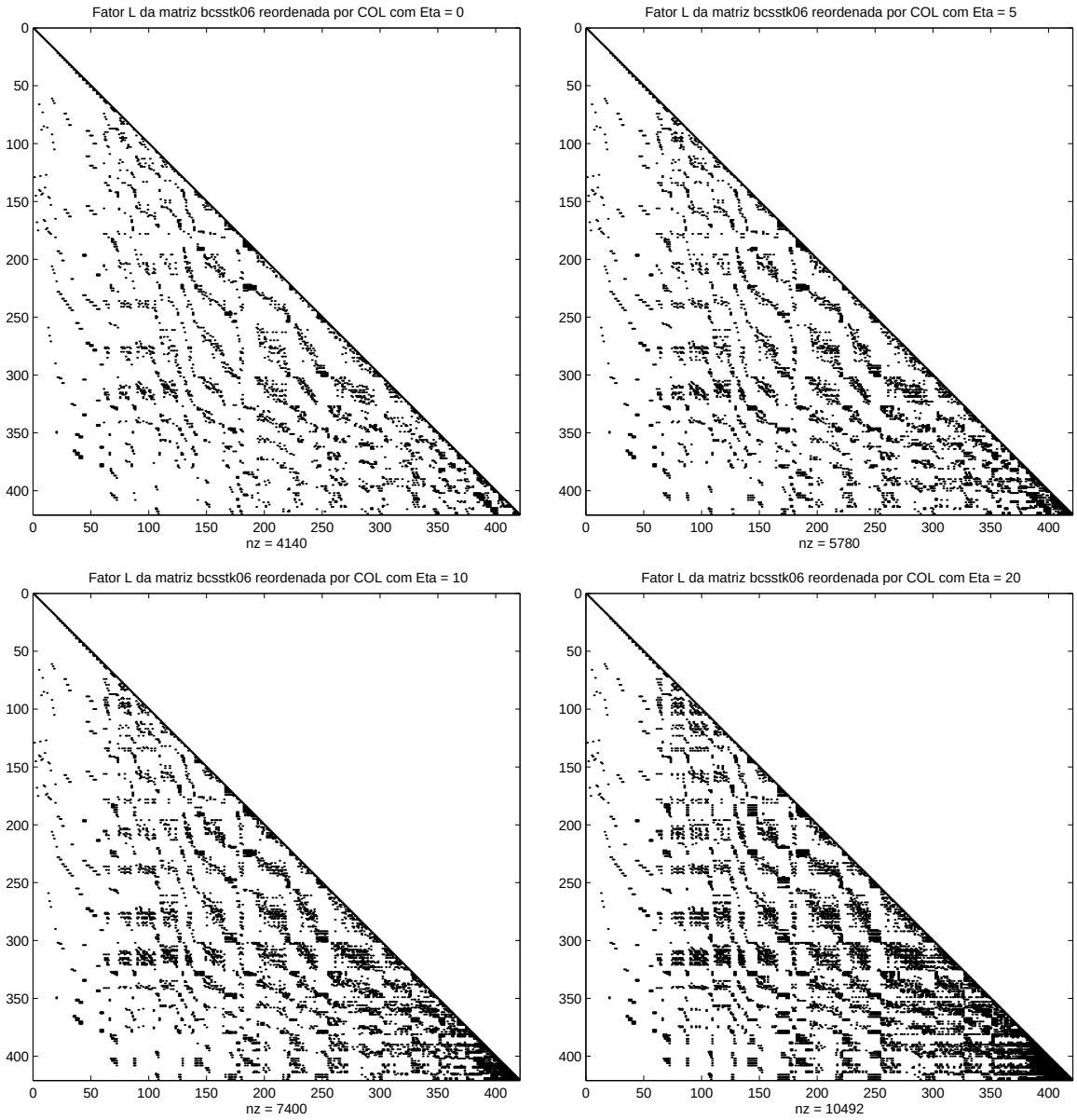


Figura 6.9: Fator L da matriz `bcsstk06` reordenada pelo contagem de colunas.

A Tabela 6.3 mostra o número de elementos não nulos do fator L das matrizes `1138_bus` e `bcsstk06` para os 4 níveis de preenchimento testados (η = 0, 5, 10 e 20) e para as reordenações utilizadas: original, mínimo grau aproximado (AMD), Cuthill-McKee reverso (RCM) e contagem de colunas (COL). Todos os algoritmos levaram a fatores mais esparsos, sendo o mínimo grau aproximado o que mais se destacou, já que seu objetivo é exatamente produzir fatores mais esparsos (seção 5.3).

η	Matriz `1138_bus`				Matriz `bcsstk06`			
	original	AMD	RCM	COL	original	AMD	RCM	COL
0	2596	2596	2596	2596	4140	4140	4140	4140
5	5689	3215	4021	3504	6004	5563	5836	5780
10	8490	3264	4661	3985	7791	6785	7431	7400
20	13731	3264	5164	4777	11303	8717	10274	10492

Tabela 6.3: Número de elementos do fator L obtido pelo MATLAB.

6.3.2 Autoespectro de $L^{-1}AL^{-T}$

A distribuição dos autovalores de uma matriz precondicionada determina a convergência do gradiente conjugado. Quanto mais próximos estes autovalores estiverem de 1, mais rápida será a convergência do método. Assim, foram realizados alguns testes para verificar o comportamento das características de convergência de algumas matrizes precondicionadas diante da aplicação de algoritmos de reordenação.

Sejam as matrizes `1138_bus` e `bcsstk06` da Figura 6.1. A variação do seu autoespectro pode ser vista nas Figuras 6.10, 6.11, 6.12 e 6.13 para a matriz `1138_bus` e 6.14, 6.15, 6.16 e 6.17 para a `bcsstk06`. O eixo x representa os autovalores em escala log_{10}, ou seja, o valor 0 representa um autovalor igual a $10^0 = 1$, o valor -0,5 representa o autovalor igual a $10^{-0,5} = 0,3162$, variando entre 0,0001 (10^{-4}) e 1000 (10^3). Já o eixo y representa a quantidade de autovalores que contém o valor determinado no eixo x. Obviamente, as matrizes possuem um total de autovalores igual à sua ordem.

O autoespectro da matriz `1138_bus` sem reordenação está representada na Figura 6.10. Neste caso, para $\eta = 0$, há uma maior dispersão dos autovalores, que variam entre 10^{-4} e 10^0. Para η = 5, 10 e 20, tem-se autovalores espalhados entre 10^{-3} e 10^0. Porém, para $\eta = 20$, esses autovalores estão concentrados mais próximos de 1. Após a reordenação pelo mínimo grau aproximado, tem-se o autoespectro na Figura 6.11. Para $\eta = 5$ já é possível perceber uma melhoria em relação à distribuição dos autovalores, que variam entre 10^{-2} e 10^0. Nos

demais níveis de preenchimento, houve uma concentração entre $10^{-0,5}$ e 10^{0}, acelerando a convergência do CG. No caso do Cuthill-McKee reverso (Figura 6.12), o comportamento deste algoritmo foi parecido com o do mínimo grau aproximado, variando de 10^{-2} a 10^{0} para $\eta = 5$ e concentrando próximo de 1 com o preenchimento. Na Figura 6.13, tem-se o autoespectro após a reordenadação pelo contagem de colunas. Este algoritmo teve a pior distribuição, variando de 10^{-4} a 10^{0} sem preenchimento, de 10^{-3} a 10^{0} com $\eta = 5$, de 10^{-2} a 10^{0} com $\eta = 10$ e de 10^{-1} a 10^{0} com $\eta = 20$.

Outro exemplo é dado pela matriz `bcsstk06`, na Figura 6.14. Para $\eta = 0$ e 5, há uma variação dos autovalores entre 10^{-2} e 10^{0}, sendo a maioria concentrada em $10^{-0,5}$. Para $\eta = 10$, tem-se autovalores espalhados entre 10^{-1} e $10^{0,5}$, com a maioria ainda concentrada em $10^{-0,5}$. Já para $\eta = 20$, houve uma melhora da distribuição e os autovalores ficaram concentrados mais próximos de 1. Após a reordenação pelo mínimo grau aproximado, o autoespectro desta matriz pode ser visto na Figura 6.15. É visível como este algoritmo piorou a distribuição dos autovalores, os quais permaneceram espalhados em todos os níveis de preenchimento, com uma grande concentração em $10^{-0,5}$. O Cuthill-McKee reverso (Figura 6.16) teve uma distribuição ruim para $\eta = 0$ variando de 10^{-2} a 10^{1}. Porém, para um maior nível de preenchimento, a distribuição melhorou e os autovalores se concentraram entre $10^{-0,5}$ e 1. Pela Figura 6.17, pode-se ver que o contagem de colunas novamente teve a pior distribuição de todos. Os autovalores ficaram dispersos para todos os níveis de preenchimento e se concentraram em $10^{-0,5}$.

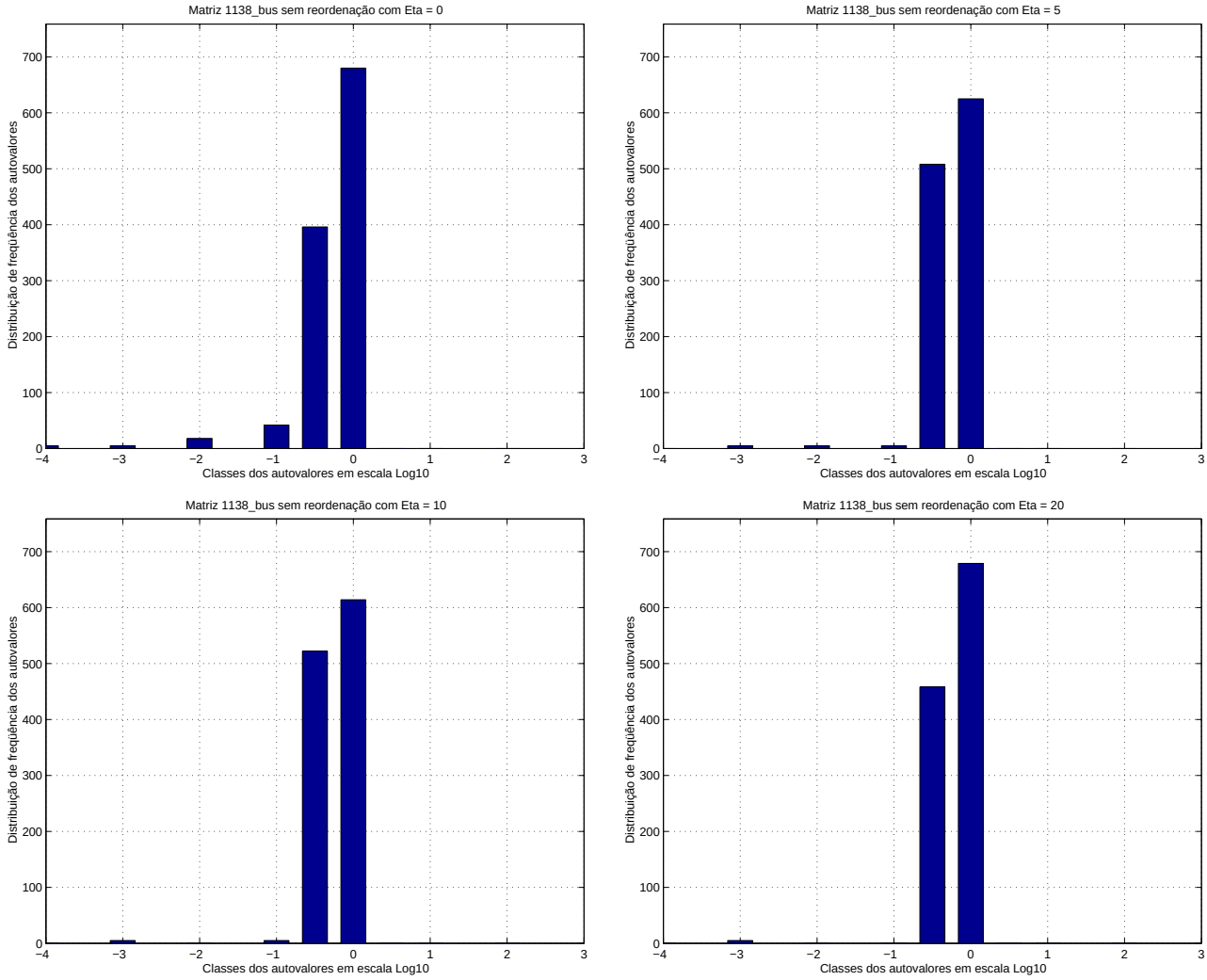


Figura 6.10: Autoespectro da matriz `1138_bus` sem reordenação.

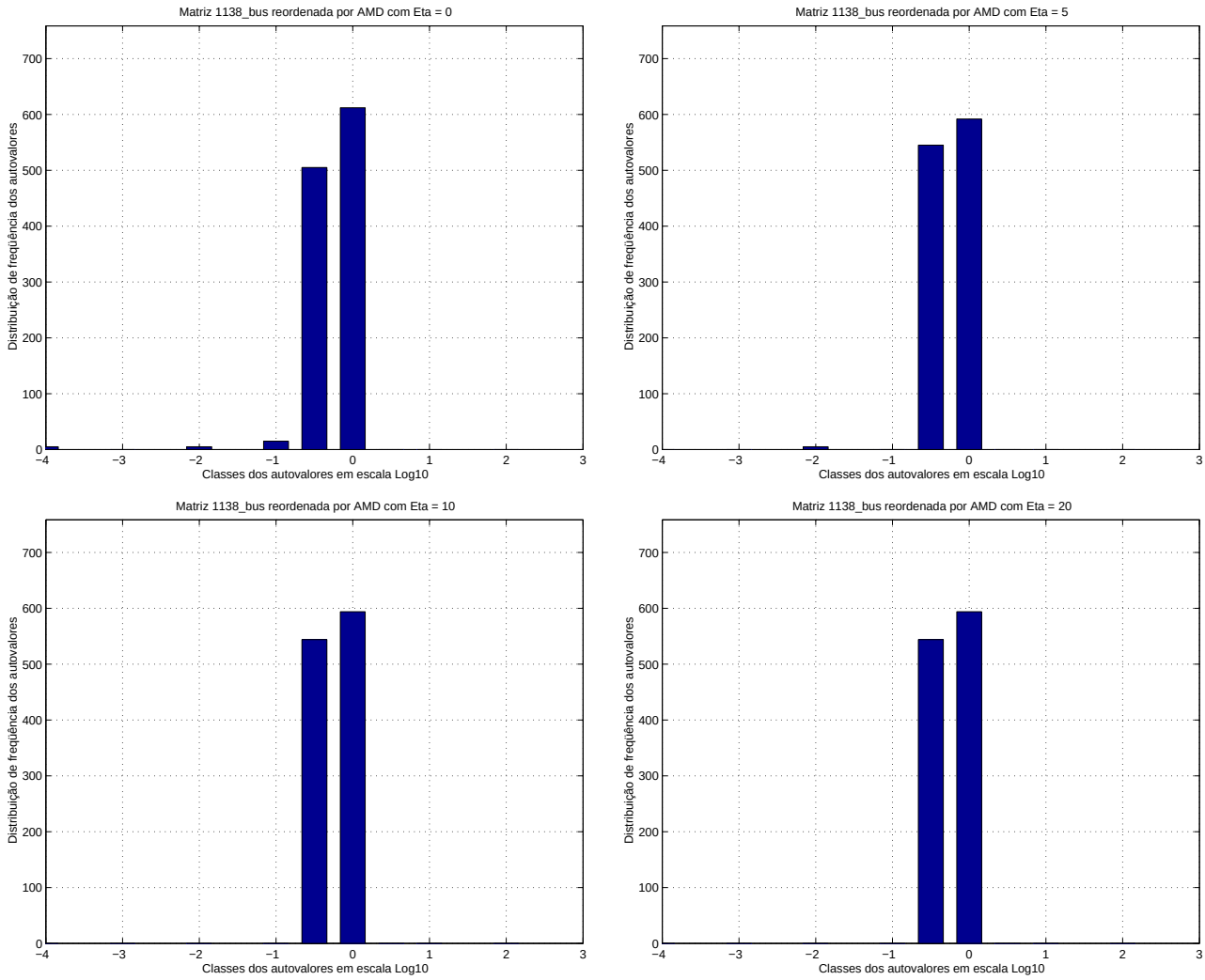


Figura 6.11: Autoespectro da matriz `1138_bus` reordenada pelo mínimo grau aproximado.

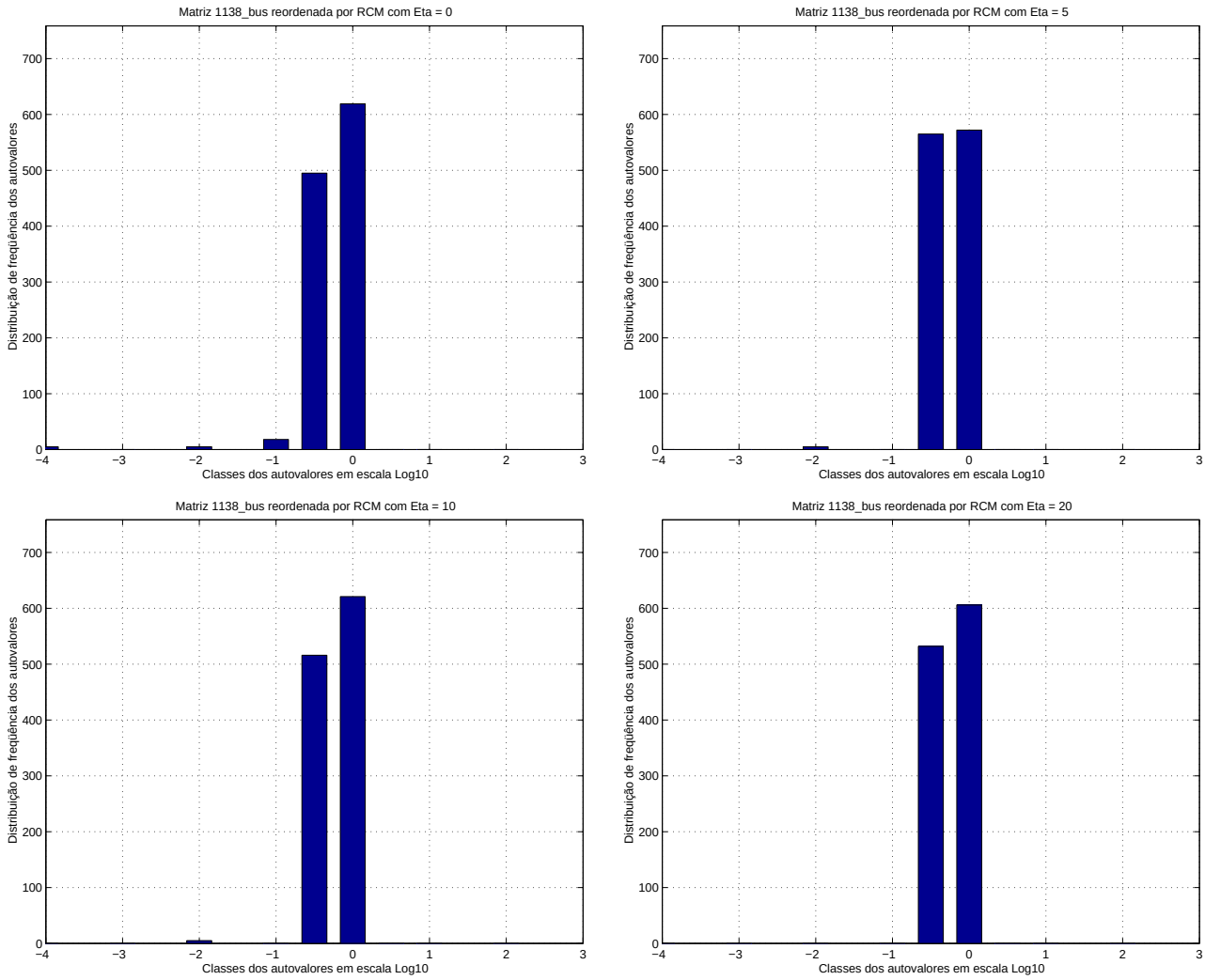


Figura 6.12: Autoespectro da matriz `1138_bus` reordenada pelo Cuthill-McKee reverso.

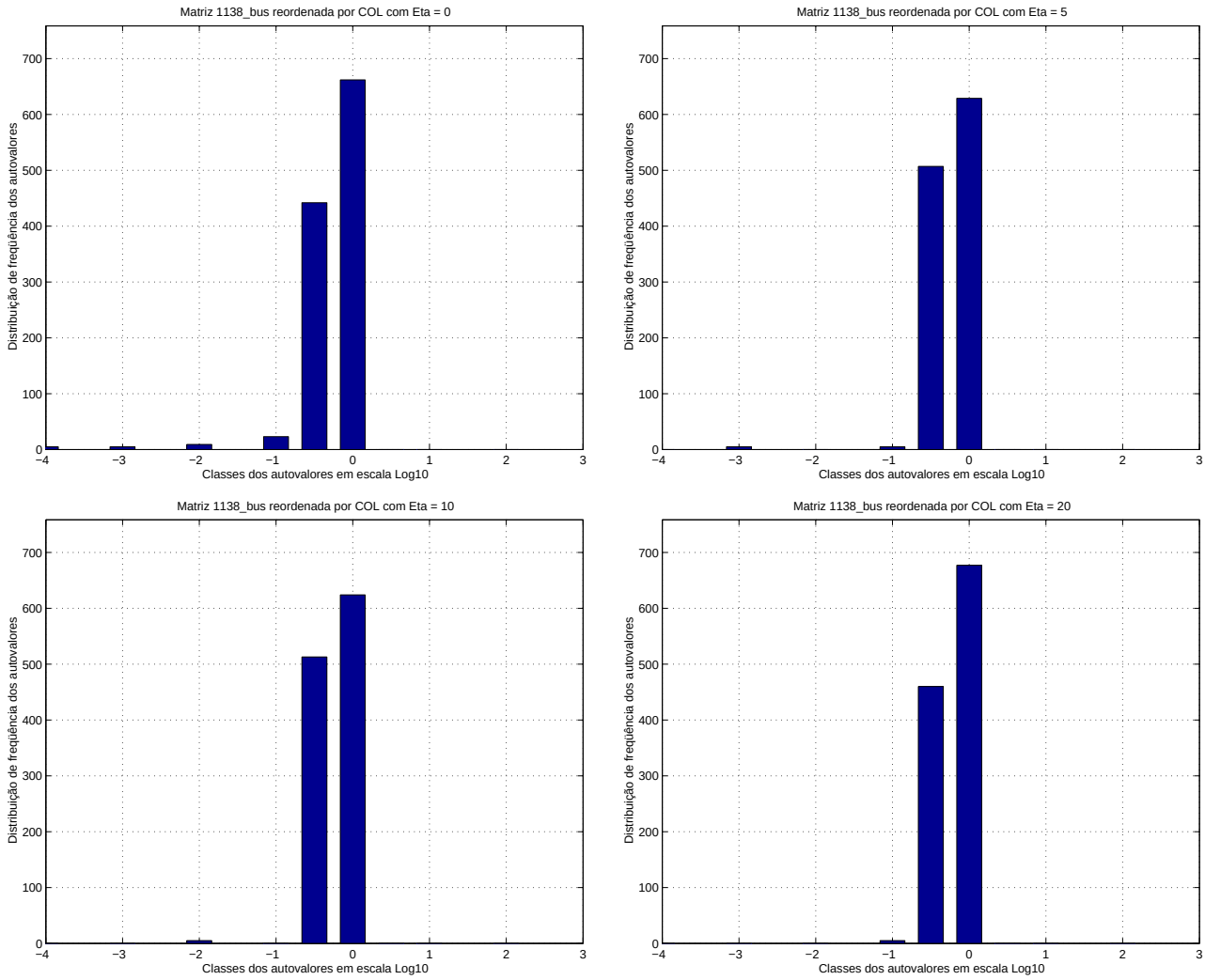


Figura 6.13: Autoespectro da matriz `1138_bus` reordenada pelo contagem de colunas.

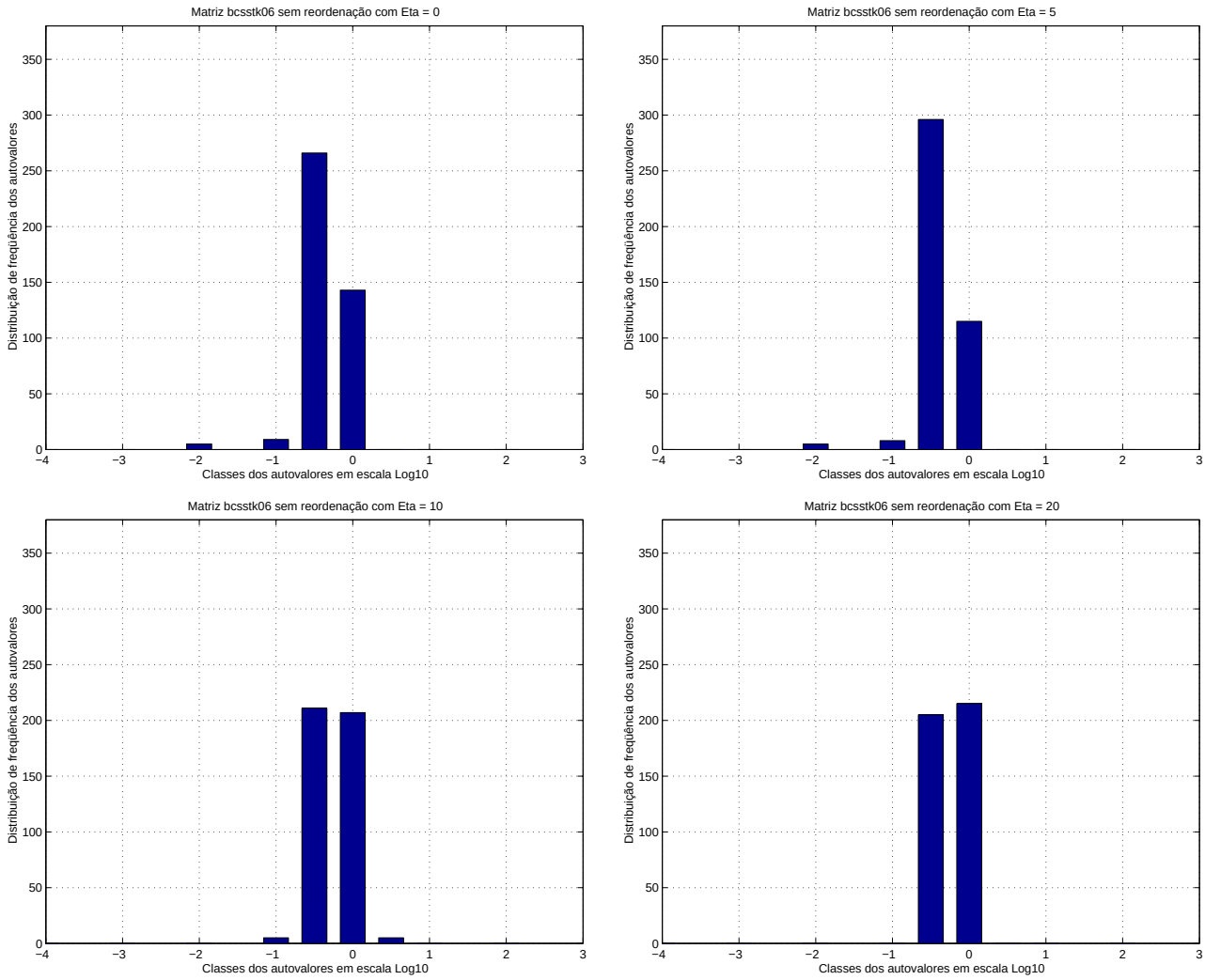


Figura 6.14: Fator L da matriz `bcsstk06` sem reordenação.

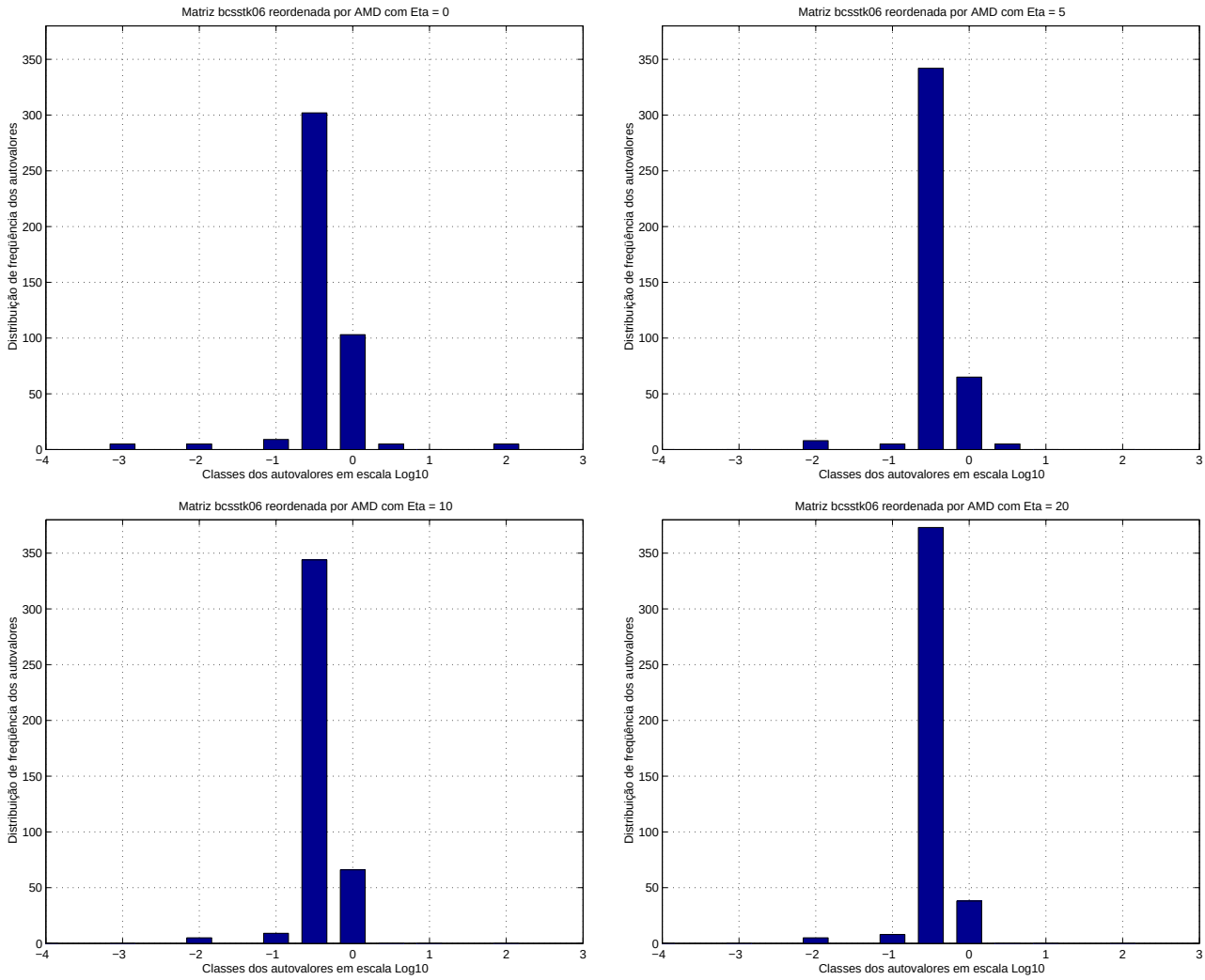


Figura 6.15: Autoespectro da matriz `bcsstk06` reordenada pelo mínimo grau aproximado.

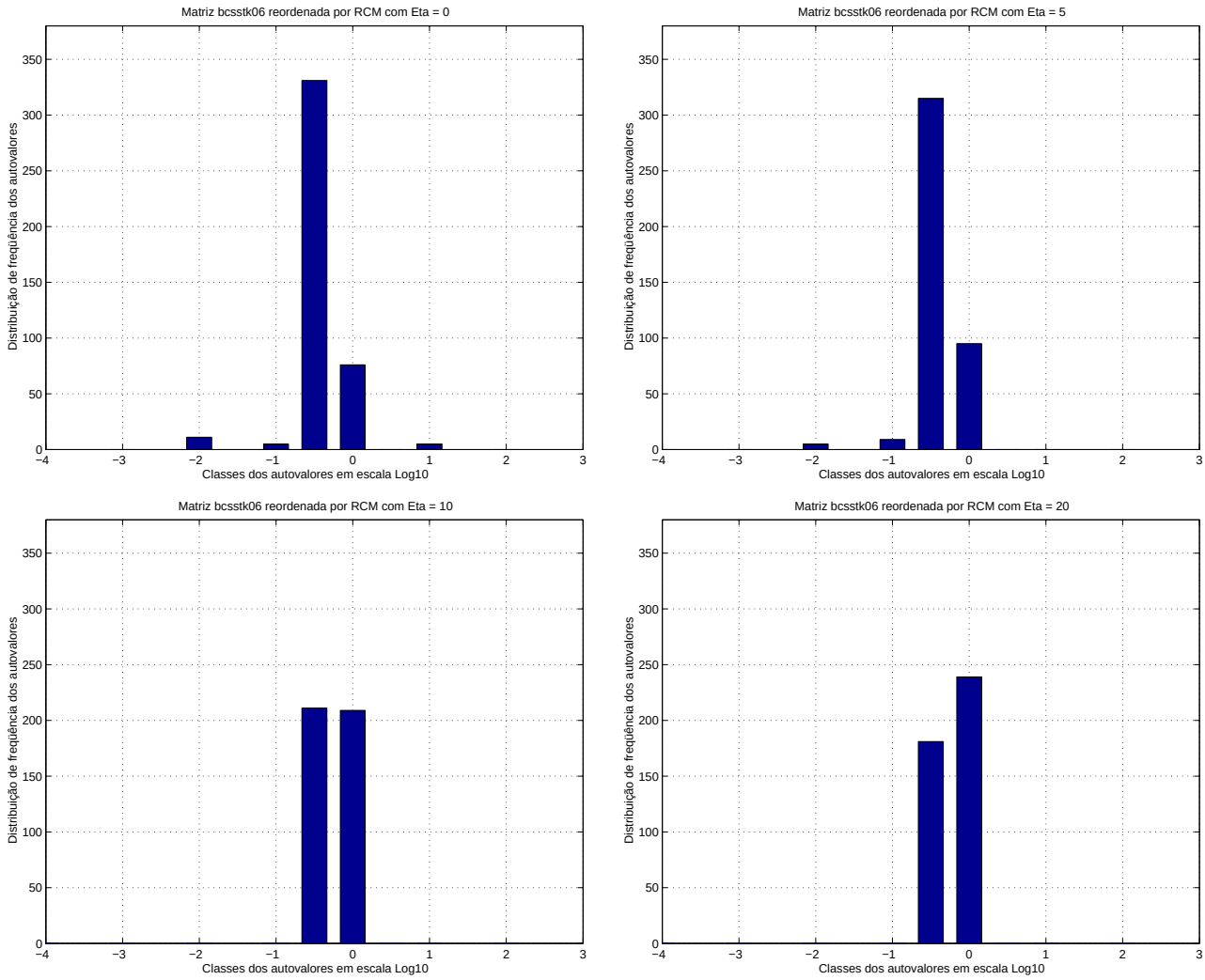


Figura 6.16: Autoespectro da matriz `bcsstk06` reordenada pelo Cuthill-McKee reverso.

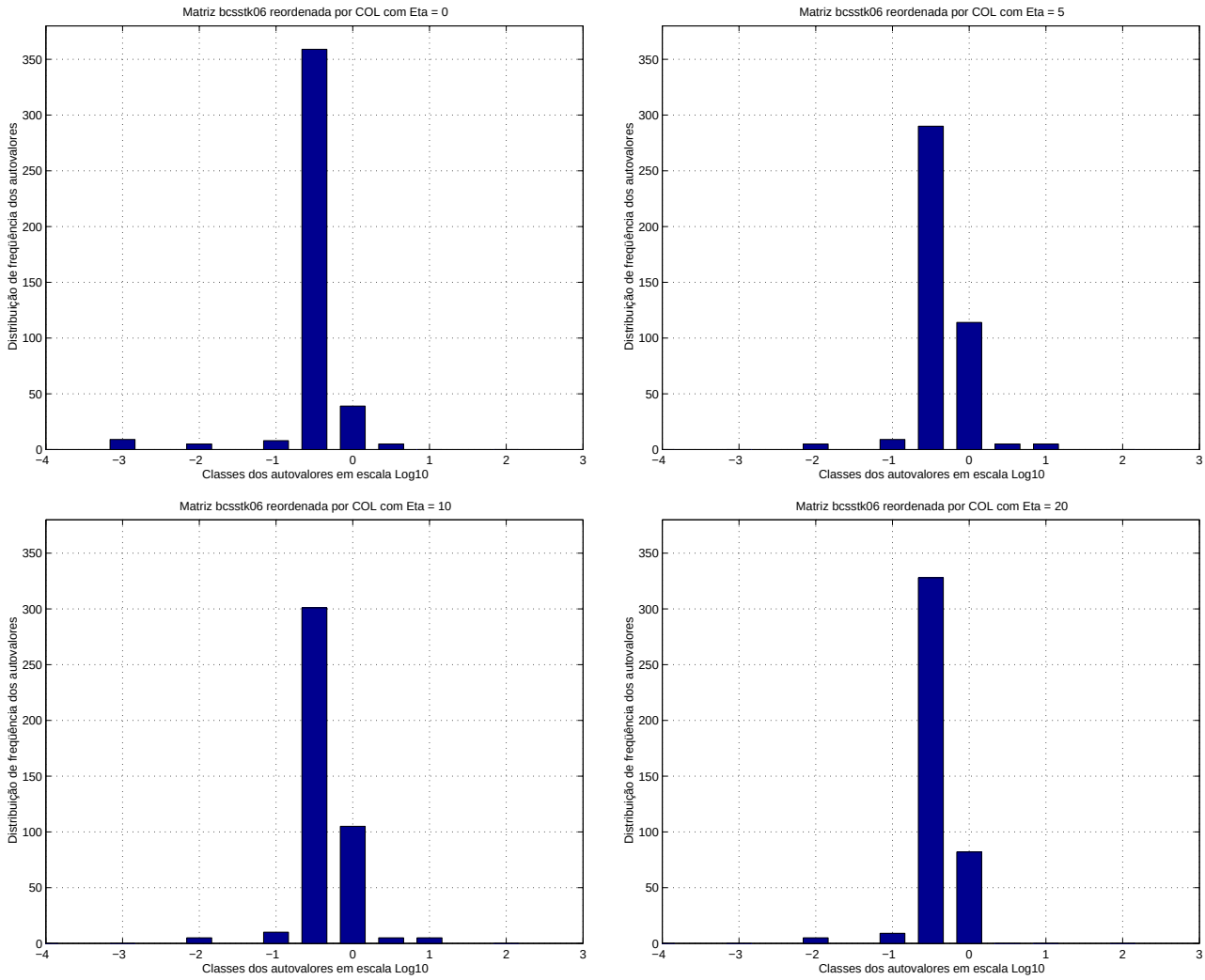


Figura 6.17: Autoespectro da matriz `bcsstk06` reordenada pelo contagem de colunas.

A Tabela 6.4 mostra o número de iterações gastas na convergência do gradiente conjugado precondicionado para os 4 níveis de preenchimento testados (η = 0, 5, 10 e 20) e para a reordenações utilizadas nas matrizes `1138_bus` e `bcsstk06`. Para a matriz `1138_bus`, a qual possui estrutura irregular, todos os algoritmos de reordenação aceleraram a convergência do método, sendo que o mínimo grau aproximado precisou de menos iterações ao ser permitido algum preenchimento. No caso da matriz banda `bcsstk06`, apenas o Cuthill-McKee reverso acelerou a convergência. Este resultado está de acordo com o autoespectro destas matrizes analisado anteriormente, onde a distribuição dos autovalores melhora a convergência do método. É interessante ver na Figura 6.11 como o mínimo grau aproximado se destacou em relação aos demais, caso oposto acontecendo com o contagem de colunas na Figura 6.17.

η	Matriz `1138_bus`				Matriz `bcsstk06`			
	original	AMD	RCM	COL	original	AMD	RCM	COL
0	131	75	70	95	40	72	50	89
5	31	9	11	16	29	46	26	61
10	19	1	7	10	19	32	15	50
20	12	1	3	6	13	24	9	32

Tabela 6.4: Número de iterações gastas pelo *CG* precondicionado, obtido pelo MATLAB.

6.4 Resultados numéricos

O principal critério utilizado para comparação entre os métodos de reordenação é o número de iterações gastas na convergência do CCCG. Porém, deve haver um equilíbrio entre o número de iterações, o preenchimento do fator L e o tempo gasto para realizar a convergência do método.

Os experimentos numéricos usando o G77 foram realizados utilizando o sistema operacional Suse Linux 9.2, processador Intel(R) Xeon(TM) de 3,06 GHz e memória de 1,5 Gb. Os resultados descritos aqui são uma seleção representativa de um grande número de experimentos.

Um observação importante é que mesmo a matriz A sendo simétrica definida positiva, a fatoração incompleta pode fazer com que o fator L não tenha elementos positivos na diagonal. Esse problema é resolvido incrementando-se a diagonal de A por um fator constante. Porém, isto pode causar desestabilização no gradiente conjugado.

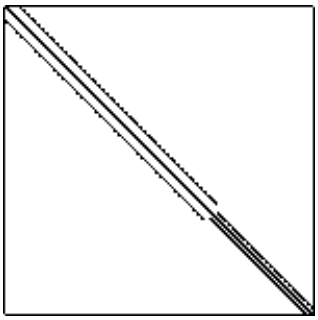
6.4.1 Matriz A com estrutura irregular

Algumas matrizes com estruturas semelhantes foram escolhidas para que seja feita uma comparação entre o preenchimento e a quantidade de iterações necessárias para a convergência do método. Matrizes com estrutura irregular não possuem nenhum tipo de organização no seu padrão de esparsidade, como pode ser visto na Figura 6.1 pelas matrizes `nos6`, `494_bus`, `1138_bus`, `bcsstk08` e `bcsstk15`. O preenchimento criado pela fatoração e o número de iterações gastas na solução destas matrizes serão analisados a seguir.

De acordo com a Figura 6.18, o número de iterações da matriz `nos6` reordenada pelo AMD diminui consideravelmente. Da mesma forma, para a matriz `494_bus` (Figura 6.19), o AMD teve um desempenho destacável na convergência, sendo que o RCM também trouxe melhorias.

O comportamento da matriz `1138_bus`, representado na Figura 6.20, foi parecido com o da `494_bus`, onde o AMD teve o melhor desempenho em relação aos outros métodos. Para a matriz `bcsstk08`, novamente o AMD trouxe grandes melhorias ao ser permitido preenchimento, o que pode ser visto na Figura 6.21. O COL desestabilizou a matriz precondicionada para $\eta = 0$, o que foi representado pelo símbolo (*).

No caso da matriz `bcsstk15` (Figura 6.22), tanto o AMD como o COL desestabilizaram a matriz precondicionada para $\eta = 0$ e 5. Desta forma, nenhum método melhorou a convergência do CCCG(η).



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	1965	1965	1965	1965	26	47	32	36
5	5074	3735	4932	4735	9	18	13	15
10	8135	5087	7548	7368	7	8	8	9
15	10896	5978	9912	9845	5	4	6	6
20	12935	6392	12041	12186	2	3	5	5
30	16904	6453	15875	16634	1	1	3	3
40	16904	6453	18797	20855	1	1	2	2
50	16904	6453	21338	24533	1	1	2	2

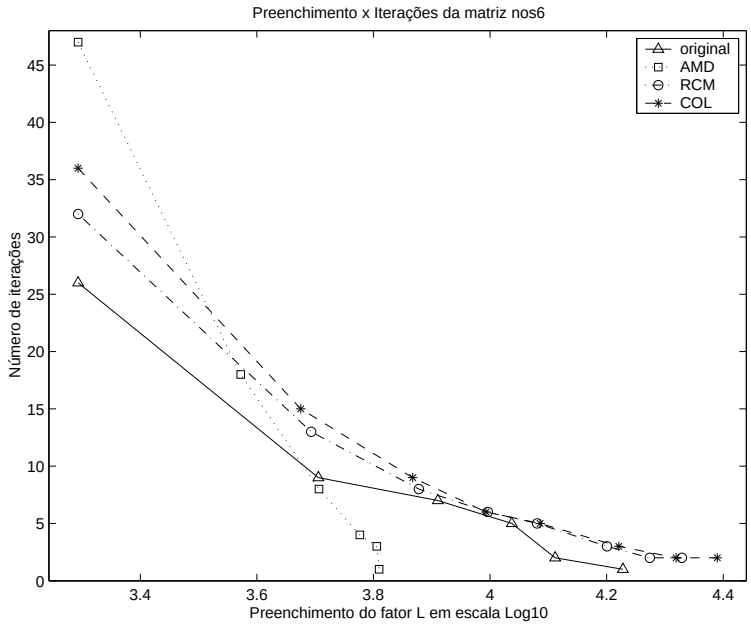
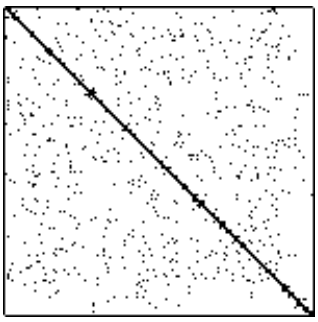


Figura 6.18: Preenchimento e número de iterações da matriz `nos6`.



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	1080	1080	1080	1080	105	48	94	129
5	2405	1394	2211	2713	19	9	20	30
10	3312	1414	3095	4232	12	1	13	19
15	3997	1414	3839	5562	10	1	10	14
20	4475	1414	4440	6801	9	1	8	11
30	5363	1414	5449	9059	6	1	7	9
40	5999	1414	6322	11177	5	1	5	7
50	6511	1414	7058	13271	4	1	3	6

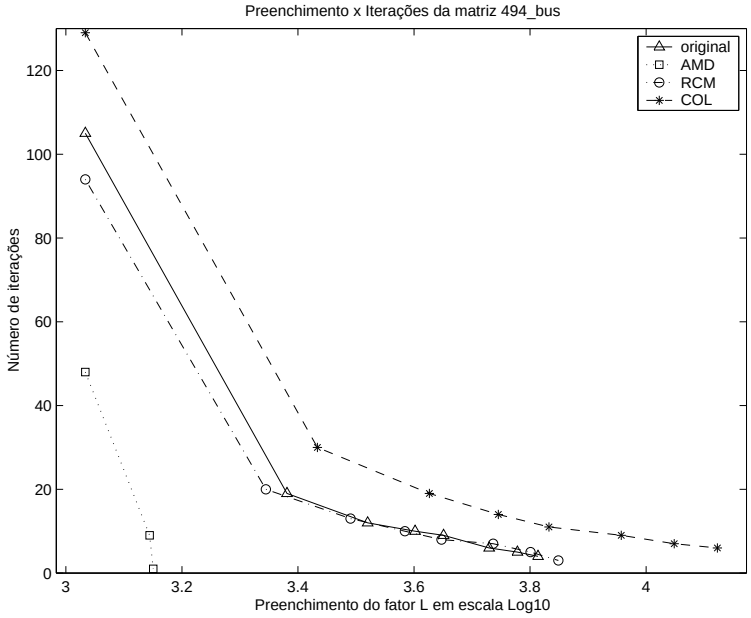
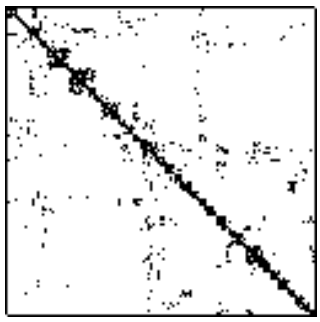


Figura 6.19: Preenchimento e número de iterações da matriz `494_bus`.



η	Elementos não nulos				Número de iterações			
	original	AMD	RCM	COL	original	AMD	RCM	COL
0	2596	2596	2596	2596	140	75	149	201
5	5688	3225	5531	6641	32	10	35	51
10	8490	3274	8080	10461	20	1	21	30
15	11149	3274	10306	14136	15	1	15	23
20	13731	3274	12296	17631	13	1	12	18
30	18603	3274	15940	24018	11	1	9	12
40	23236	3274	19325	29575	10	1	9	9
50	27579	3274	22360	34550	9	1	7	8

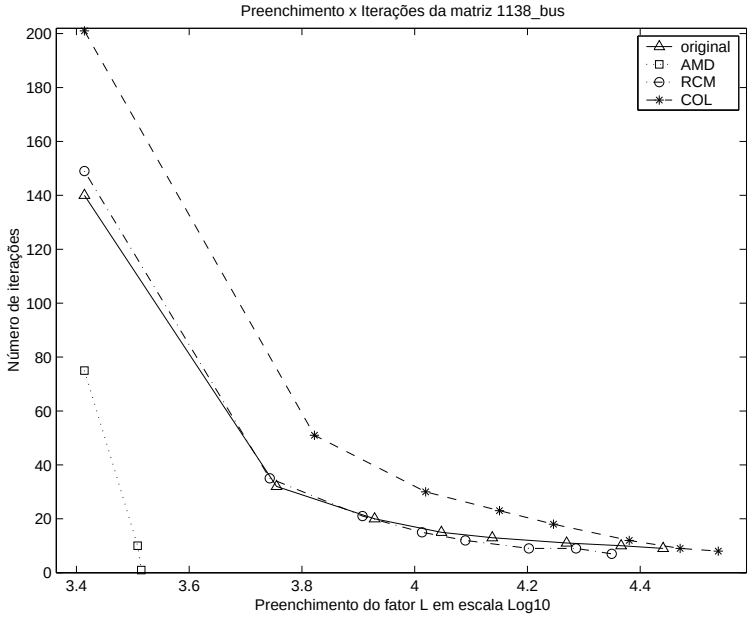
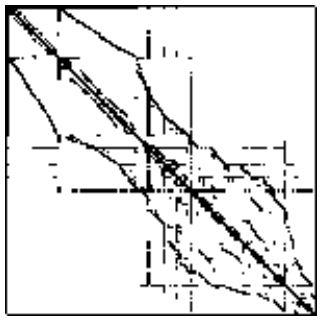


Figura 6.20: Preenchimento e número de iterações da matriz `1138_bus`.



η	Elementos não nulos				Número de iterações			
	original	AMD	RCM	COL	original	AMD	RCM	COL
0	7017	7017	7017	7017	23	29	27	30(*)
5	12143	10053	11564	12269	14	9	17	15
10	17316	12455	15992	17470	12	7	16	13
15	22454	14593	20392	22654	12	6	15	12
20	27533	16574	24711	27804	11	5	14	11
30	37626	20101	33130	38008	10	4	13	10
40	47829	23030	41423	48127	9	3	12	9
50	57784	25504	49543	58312	8	3	11	9

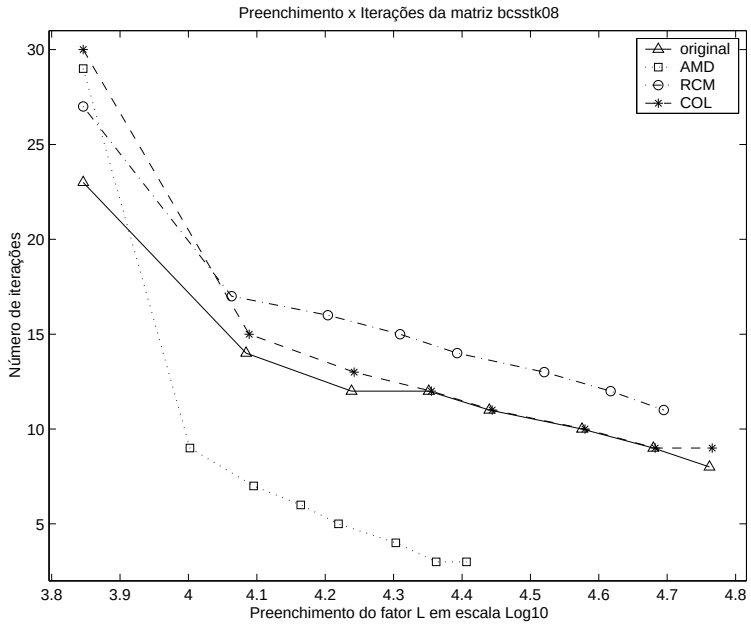
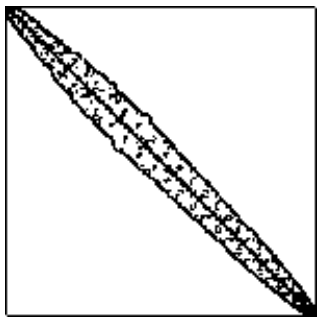


Figura 6.21: Preenchimento e número de iterações da matriz `bcsstk08`.



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	60882	60882	60882	60882	68	123(*)	114	127(*)
5	80330	77580	79964	80163	46	85(*)	65	80(*)
10	99696	93482	98971	99464	37	58	50	54
15	118913	108765	117892	118742	31	45	42	48
20	138205	123670	136632	137886	27	39	36	42
30	176405	152721	173527	175905	22	30	28	35
40	214222	180825	210073	213732	19	25	23	30
50	251869	207942	245899	251318	16	21	20	29

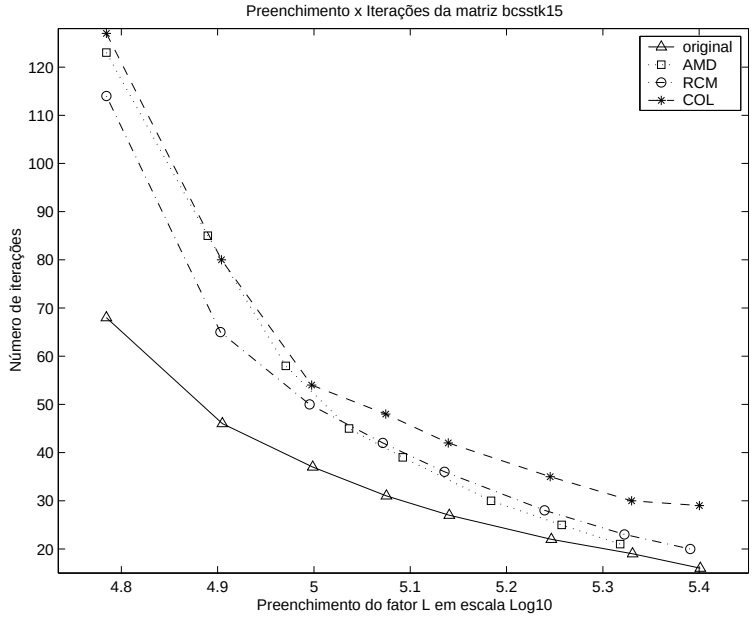


Figura 6.22: Preenchimento e número de iterações da matriz `bcsstk15`.

Estes resultados indicam que a convergência do CCCG(η) é melhorada com o uso do mínimo grau aproximado para matrizes com estrutura irregular.

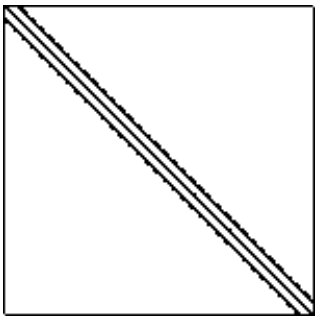
6.4.2 Matriz A com estrutura banda

Matrizes com estrutura banda possuem uma organização onde seus elementos encontram-se agrupados próximos a diagonal. Neste caso, serão apresentados os resultados obtidos pela execução do CCCG(η) para as matrizes `nos3`, `nos7`, `bcsstk06`, `bcsstk10` e `bcsstk16` (Figura 6.1).

Os resultados da matriz `nos3` estão representados na Figura 6.23. As reordenações não apresentaram melhorias significativas na convergência. No caso da matriz `nos7` (Figura 6.24), o AMD apresentou melhorias na convergência quando foi permitido um alto nível de preenchimento.

Na Figura 6.25, observa-se que todos os métodos desestabilizaram a matriz `bcsstk06`, o que impossibilita a comparação do número de iterações. Mesmo assim, é possível ver como a inclinação da curva do AMD é maior, mostrando como ele converge mais rápido com altos níveis de preenchimento.

No caso da matriz `bcsstk10` (Figura 6.26), a ordenação original foi melhor, sendo que todos os métodos desestabilizaram a matriz precondicionada, impossibilitando a comparação do número de iterações. Para a matriz `bcsstk16`, representada na Figura 6.27, novamente a ordenação original foi melhor.



η	Elementos não nulos				Número de iterações			
	original	AMD	RCM	COL	original	AMD	RCM	COL
0	8402	8402	8402	8402	43	63	58	69
5	13146	12018	12998	12756	30	40	40	42
10	17833	15231	17448	16984	21	30	31	32
15	22470	18056	21722	21075	17	24	25	27
20	27062	20565	25785	24993	15	19	22	23
30	36120	24572	33546	32665	10	13	17	18
40	40061	27365	40167	39994	1	10	14	15
50	40061	29164	45933	47038	1	7	12	13

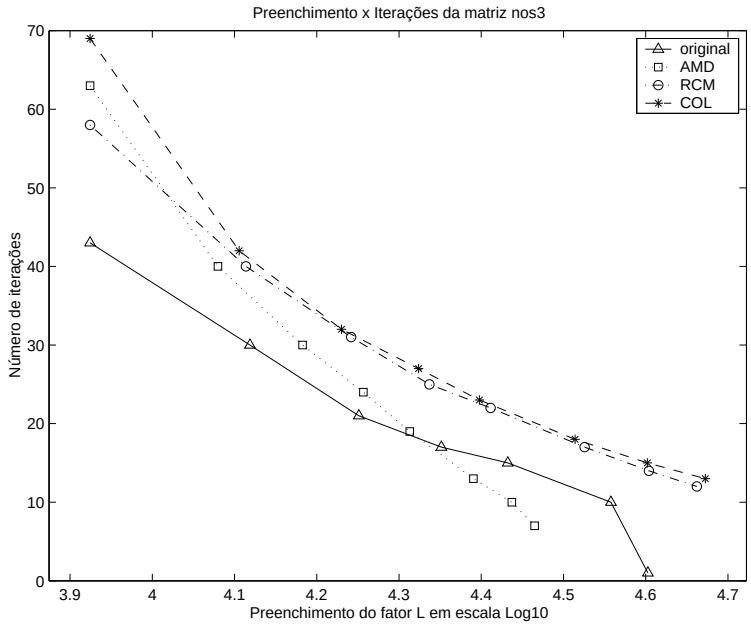
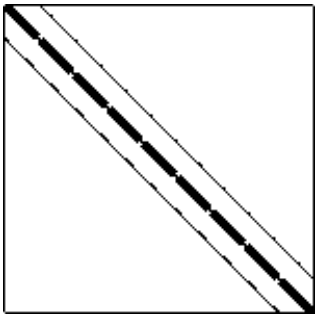


Figura 6.23: Preenchimento e número de iterações da matriz `nos3`.



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	2673	2673	2673	2673	27	45	35	37
5	6264	4462	5965	5952	15	29	21	23
10	9851	6233	9175	9197	11	17	16	17
15	13308	7808	12233	12406	10	12	12	13
20	16689	9213	15396	15529	9	10	10	12
30	23300	11580	21234	21740	9	8	8	10
40	29963	13544	26792	27953	8	6	7	8
50	36693	15154	32092	33839	6	5	6	8

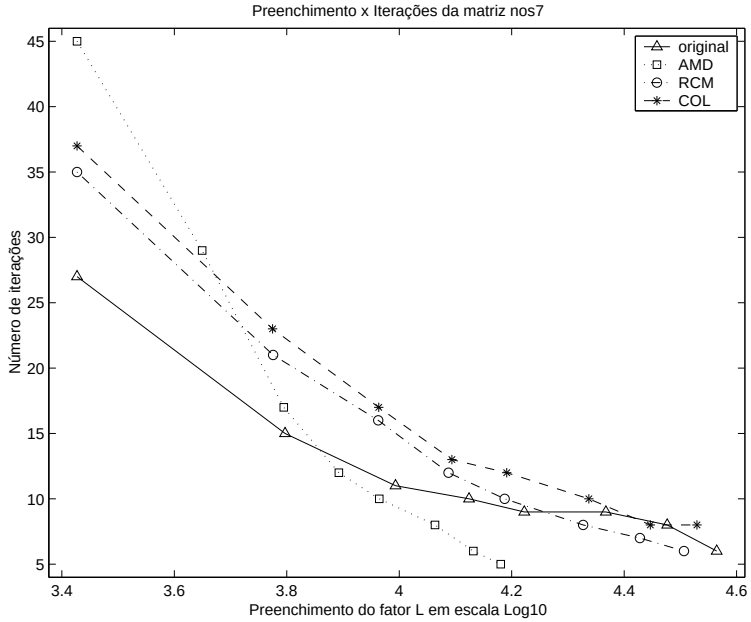
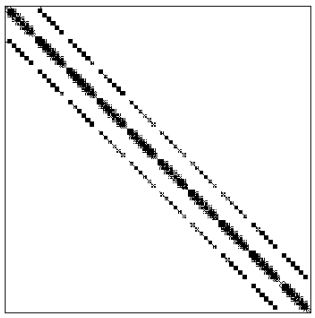


Figura 6.24: Preenchimento e número de iterações da matriz `nos7`.



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	4140	4140	4140	4140	39(*)	72(*)	58(*)	81(*)
5	6004	5557	6016	6046	27(*)	56(*)	31(*)	54(*)
10	7791	6764	7813	7936	19	39(*)	19	31
15	9587	7825	9576	9804	16	18	23(*)	24
20	11303	8757	11275	11638	14	13	21(*)	23
30	13849	10218	14560	15173	6	8	8	20
40	14282	11025	17646	18653	1	4	5	24(*)
50	14282	11154	20522	22051	1	1	4	23(*)

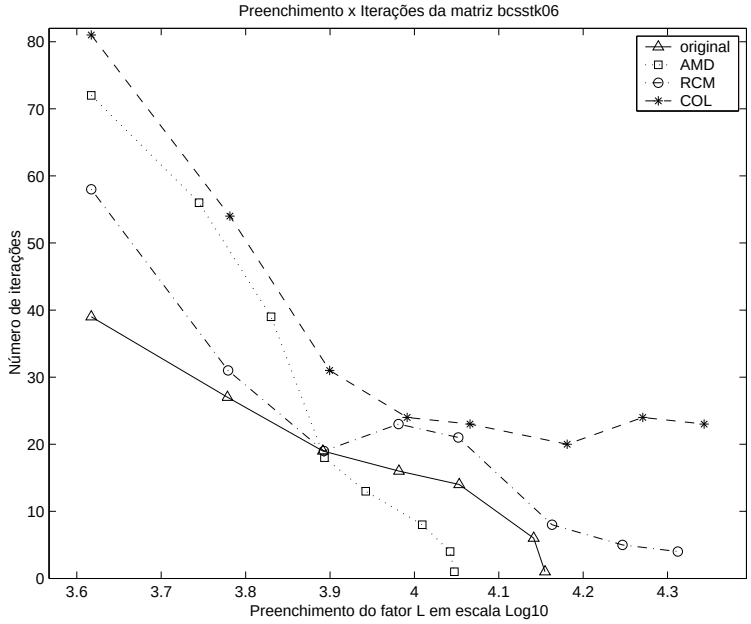
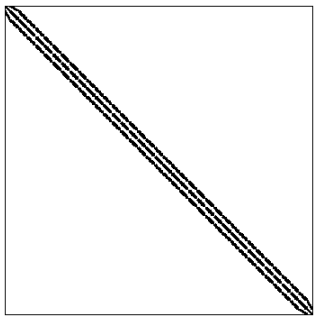


Figura 6.25: Preenchimento e número de iterações da matriz `bcsstk06`.



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	11578	11578	11578	11578	62(*)	140(*)	110(*)	201(*)
5	16914	15576	16814	16708	29	74(*)	81(*)	129(*)
10	20350	18944	21599	21723	9	36(*)	72(*)	98(*)
15	20771	21397	25149	26591	1	23(*)	54(*)	65(*)
20	20771	22915	27650	31344	1	13	29(*)	48(*)
30	20771	23789	30715	40460	1	1	7	26(*)
40	20771	23789	31910	49177	1	1	2	13
50	20771	23789	31967	57441	1	1	1	9

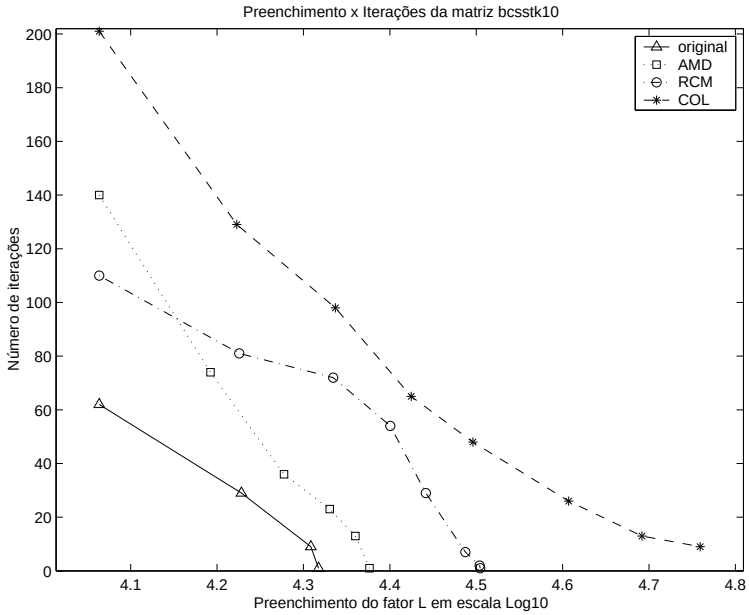
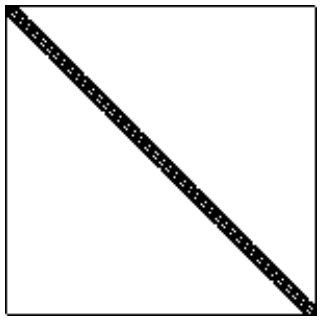


Figura 6.26: Preenchimento e número de iterações da matriz `bcsstk10`.



	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	147631	147631	147631	147631	32	48	42	53
5	171615	167285	171394	171243	29	41	35	50
10	195569	186633	195097	194893	25	35	30	42
15	214222	205332	218715	218521	22	32	27	41
20	243317	223550	242207	242055	20	29	24	35
30	290909	258640	289023	289100	16	24	20	30
40	338347	292489	335507	335849	14	21	17	26
50	385615	325742	381804	382333	12	18	14	24

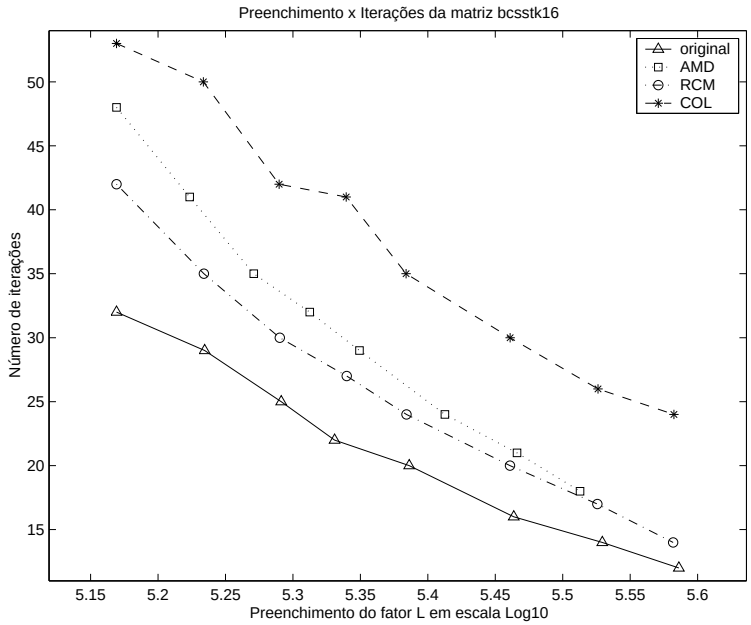


Figura 6.27: Preenchimento e número de iterações da matriz `bcsstk16`.

Nestes casos, a reordenação não trouxe melhorias na convergência do CCCG(η), indicando que para matriz banda a reordenação não deve ser utilizada.

6.4.3 Matriz A com estrutura desconhecida

Os testes agora foram realizados com as matrizes `minsurfo`, `wathen120`, `finan512`, `nasasbr` e `gyro_k` as quais possuem alta ordem. Porém, sua estrutura de esparsidade não nos é conhecida, podendo ser tanto banda como irregular.

No caso da matriz `minsurfo`, observada na Figura 6.28, o AMD diminuiu consideravelmente o número de elementos do fator L, tornando-se competitivo quando altos níveis de preenchimento são permitidos. O comportamento da matriz `wathen120`, visto na Figura 6.29, foi semelhante ao da anterior, sem melhorias no número de iterações.

Na Figura 6.30, pode-se ver que o desempenho do AMD para a matriz `finan512` foi destacável, tanto em relação ao número de iterações quanto ao preenchimento. Para a matriz `nasasbr` (Figura 6.31), mesmo com a desestabilização em todos os casos, o desempenho do AMD para esta matriz foi muito bom. Novamente, mesmo ocorrendo desestabilização na matriz `gyro_k`, representada na Figura 6.32, o AMD conseguiu diminuir o número de iterações e o preenchimento.

	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	122214	122214	122214	122214	30	52	34	39
5	325400	224372	321037	298210	10	23	16	18
10	527689	316154	516923	473547	7	12	11	12
15	728168	384165	709045	634998	5	9	9	9
20	927977	442781	896383	794499	4	7	7	8
30	1323253	539889	1259585	1100825	3	5	6	6
40	1717398	613520	1609681	1405377	3	4	5	5
50	2107483	674767	1948138	1707895	2	4	4	4

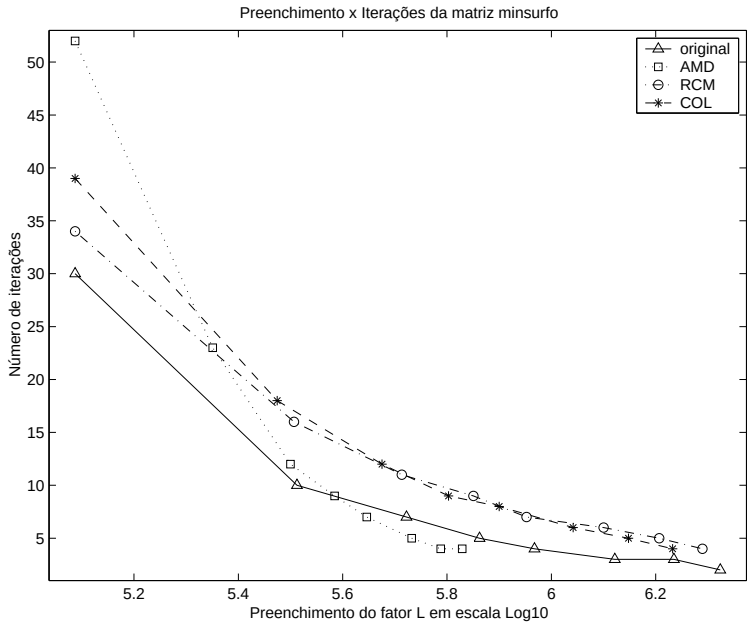


Figura 6.28: Preenchimento e número de iterações da matriz `minsurfo`.

	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	301101	301101	301101	301101	14	23	26	24
5	483176	434026	465277	477096	7	14	16	24
10	664607	553057	618074	652191	5	9	11	16
15	845867	655326	767451	825104	4	8	9	13
20	1026621	749152	912374	997197	3	6	7	11
30	1387004	903507	1200938	1341094	2	5	5	9
40	1746059	1018002	1488063	1679181	2	3	5	8
50	2103418	1115111	1774033	2016994	2	3	4	7

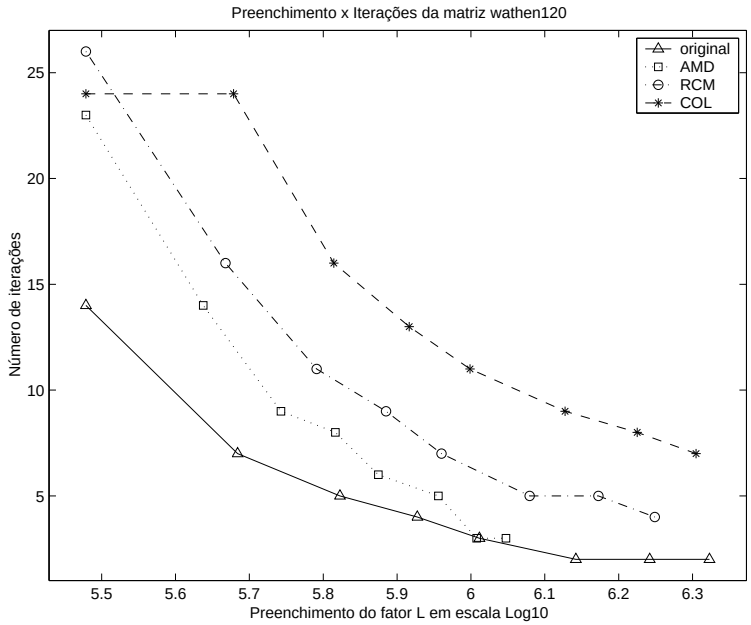


Figura 6.29: Preenchimento e número de iterações da matriz `wathen120`.

	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	335872	335872	335872	335872	11	9	12	12
5	675928	506944	633855	681895	9	6	9	9
10	1004663	618017	918538	1019177	8	5	8	8
15	1311016	704166	1196557	1368323	7	4	8	8
20	1564860	776957	1469228	1724649	7	4	7	7
30	2034732	903887	2006097	2395616	6	3	6	6
40	2504244	1013933	2540339	2998525	5	3	6	5
50	2977224	1120759	3077690	3599139	5	2	5	5

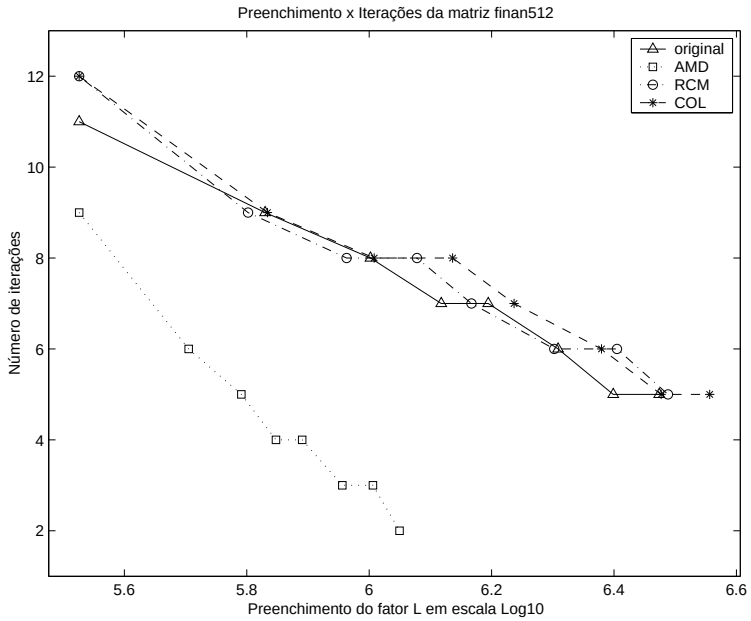


Figura 6.30: Preenchimento e número de iterações da matriz `finan512`.

η	Elementos não nulos				Número de iterações			
	original	AMD	RCM	COL	original	AMD	RCM	COL
0	1366097	1366097	1366097	1366097	2565(*)	3373(*)	2817(*)	3401(*)
5	1640293	1596084	1637596	1634230	2340(*)	2850(*)	2138(*)	2995(*)
10	1914451	1821289	1908524	1902689	2254(*)	3176(*)	1778(*)	2437(*)
15	2188529	2042879	2179227	2170959	2202(*)	2157(*)	1366(*)	2040(*)
20	2462479	2256207	2449213	2437683	1987(*)	1897(*)	1407(*)	1683(*)
30	3009949	2666707	2987195	2968805	1804(*)	1776(*)	818(*)	1610(*)
40	3556853	3059443	3522582	3497599	1685(*)	1171(*)	596(*)	1386(*)
50	4103064	3430931	4055316	4024581	1641(*)	1048(*)	558(*)	1269(*)

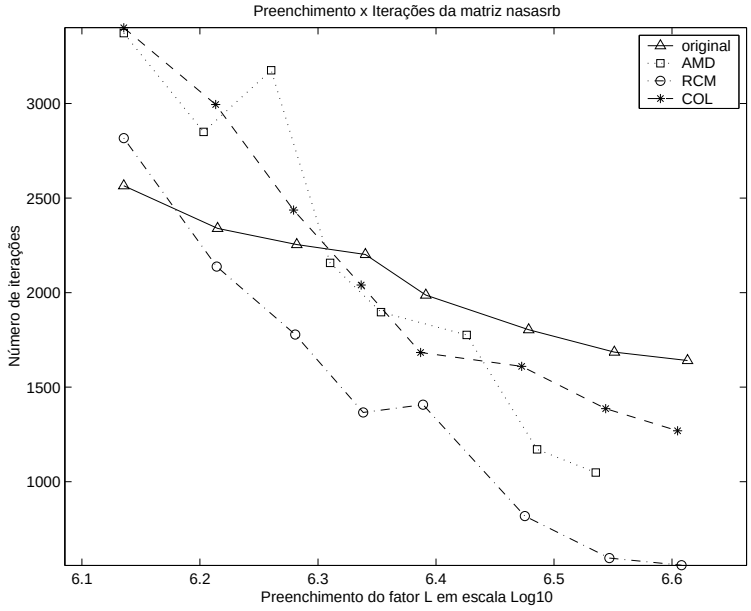


Figura 6.31: Preenchimento e número de iterações da matriz `nasasbr`.

	Elementos não nulos				Número de iterações			
η	original	AMD	RCM	COL	original	AMD	RCM	COL
0	519260	519260	519260	519260	1843(*)	1091(*)	1718(*)	1736(*)
5	604017	583302	599770	605473	1356(*)	929(*)	1278(*)	1390(*)
10	687277	644186	680211	691658	1235(*)	687(*)	1044(*)	1253(*)
15	769995	698694	758996	777927	1170(*)	539(*)	953(*)	1045(*)
20	852084	748221	836817	863826	1033(*)	461(*)	783(*)	934(*)
30	1015228	833624	991001	1035483	867(*)	350(*)	770(*)	822(*)
40	1175904	903667	1142523	1206620	708(*)	330(*)	542(*)	704(*)
50	1344656	964306	1294234	1377053	643(*)	242(*)	503(*)	672(*)

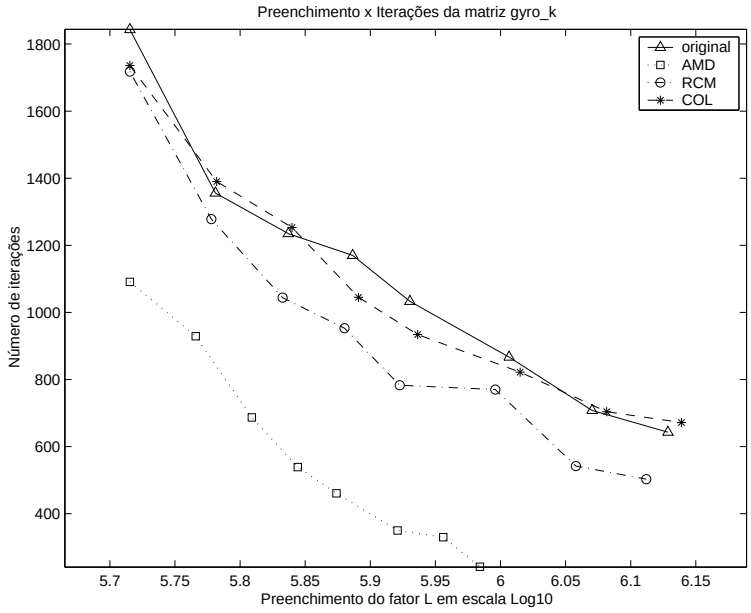


Figura 6.32: Preenchimento e número de iterações da matriz `gyro_k`.

6.4.4 Tempo de execução

Duff e Meurant [DM89] não informaram o tempo de solução de seus experimentos, justificando que ele foi quase diretamente proporcional ao número de iterações. Porém, podemos comprovar aqui que para alguns casos, mesmo aumentando o número de iterações, é possível diminuir consideravelmente o tempo de solução usando a reordenação.

Foram medidos 4 tempos para os valores de η iguais a 0, 5, 10, 15, 20, 30, 40 e 50:

- Reordenação (REO): inclui o tempo gasto pelo algoritmo de reordenação e o pela reordenação da estrutura de dados do CCCG(η) a partir do vetor de permutação criado, além da permutação do próprio vetor de solução.
- Precondicionamento (PRE): inclui o tempo do precondicionador CCF(η) e do escalonamento diagonal.
- Iterações (ITE): tempo gasto pelas iterações do gradiente conjugado.
- Total (TOT): valor total do tempo de execução.

Sobre o tempo de CPU, vale ressaltar que o compilador G77 possui uma rotina para computar este tempo com uma precisão de apenas 0.01 de um segundo de CPU, sendo importante considerar esta margem de erro nos resultados obtidos.

Conforme visto nos resultados numéricos da matriz `minsurfo` na Tabela 6.33, o número de iterações não diminuiu com a reordenação. Porém, com o aumento do parâmetro η, o AMD conseguiu diminuir o tempo consideravelmente devido ao seu baixo preenchimento. Para a matriz `finan512`, os resultados da Tabela 6.34 mostram que o AMD diminuiu o número de iterações e o tempo de execução consideravelmente devido ao seu baixo preenchimento.

η	REO	PRE	ITE	TOT
0	0.00	0.04	0.28	0.32
5	0.00	0.15	0.15	0.30
10	0.00	0.31	0.15	0.46
15	0.00	0.50	0.15	0.65
20	0.00	0.73	0.15	0.88
30	0.00	1.29	0.17	1.45
40	0.00	1.95	0.22	2.17
50	0.00	2.73	0.20	2.94

a) Ordenação original

η	REO	PRE	ITE	TOT
0	0.14	0.05	0.54	0.73
5	0.14	0.12	0.31	0.57
10	0.14	0.20	0.20	0.54
15	0.14	0.27	0.17	0.59
20	0.14	0.34	0.15	0.64
30	0.14	0.47	0.13	0.75
40	0.14	0.59	0.13	0.86
50	0.14	0.69	0.14	0.97

b) Mínimo grau aproximado

η	REO	PRE	ITE	TOT
0	0.06	0.05	0.36	0.48
5	0.06	0.25	0.26	0.57
10	0.06	0.52	0.24	0.82
15	0.06	0.85	0.25	1.16
20	0.06	1.23	0.24	1.54
30	0.06	2.18	0.28	2.52
40	0.06	3.35	0.32	3.72
50	0.06	4.69	0.32	5.07

c) Cuthill-McKee reverso

η	REO	PRE	ITE	TOT
0	0.05	0.04	0.40	0.49
5	0.05	0.15	0.27	0.47
10	0.05	0.31	0.24	0.60
15	0.05	0.52	0.22	0.79
20	0.05	0.76	0.24	1.05
30	0.05	1.39	0.25	1.68
40	0.05	2.21	0.27	2.53
50	0.05	3.18	0.27	3.51

d) Contagem de colunas

Figura 6.33: Tempo de execução em segundos da matriz `minsurfo`.

η	REO	PRE	ITE	TOT
0	0.00	0.27	0.29	0.57
5	0.00	0.61	0.33	0.93
10	0.00	1.08	0.36	1.44
15	0.00	1.62	0.38	2.01
20	0.00	2.20	0.43	2.63
30	0.00	3.33	0.46	3.79
40	0.00	4.35	0.50	4.85
50	0.00	5.17	0.58	5.74

a) Ordenação original

η	REO	PRE	ITE	TOT
0	0.36	0.19	0.27	0.82
5	0.36	0.32	0.23	0.90
10	0.36	0.45	0.22	1.02
15	0.36	0.59	0.20	1.14
20	0.36	0.73	0.21	1.30
30	0.36	1.06	0.18	1.61
40	0.36	1.40	0.20	1.95
50	0.36	1.77	0.17	2.29

b) Mínimo grau aproximado

η	REO	PRE	ITE	TOT
0	0.18	0.34	0.44	0.95
5	0.18	0.74	0.42	1.33
10	0.18	1.26	0.44	1.88
15	0.18	1.93	0.50	2.60
20	0.18	2.72	0.51	3.40
30	0.18	4.70	0.56	5.43
40	0.18	7.12	0.68	7.98
50	0.18	9.98	0.69	10.84

c) Cuthill-McKee reverso

η	REO	PRE	ITE	TOT
0	0.14	0.38	0.49	1.00
5	0.14	0.82	0.48	1.44
10	0.14	1.42	0.52	2.08
15	0.14	2.20	0.60	2.94
20	0.14	3.13	0.62	3.89
30	0.14	5.42	0.68	6.25
40	0.14	8.36	0.72	9.22
50	0.14	11.92	0.83	12.89

d) Contagem de colunas

Figura 6.34: Tempo de execução em segundos da matriz `finan512`.

As matrizes `minsurfo` e `finan512` permaneceram estáveis durante a solução, ou seja, mantiveram a condição de ser definida positiva. Porém, a Figura 6.35 mostra a matriz `nasasbr` onde ocorreu problema com a diagonal. Neste caso, foi inserida uma coluna σ que mostra o número de vezes que a matriz L teve que ser recalculada, já que não possui diagonal positiva. A desestabilização da matriz precondicionada prejudica o tempo de execução dos métodos. Mesmo assim, o RCM foi muito eficiente, já que também diminui o preenchimento da matriz.

η	σ	REO	PRE	ITE	TOT
0	7	0.00	2.28	158.13	160.42
5	7	0.00	2.98	159.65	162.64
10	7	0.00	3.83	170.39	174.21
15	7	0.00	4.74	183.85	188.59
20	6	0.00	5.61	183.21	188.82
30	6	0.00	7.92	191.40	199.33
40	6	0.00	10.63	200.67	211.30
50	6	0.00	13.67	218.12	231.80

a) Ordenação original

η	σ	REO	PRE	ITE	TOT
0	8	0.80	6.60	266.88	274.28
5	7	0.79	5.81	242.46	249.07
10	6	0.80	5.49	288.13	294.41
15	6	0.80	7.05	208.32	216.17
20	6	0.80	9.39	194.11	204.30
30	5	0.79	10.46	202.57	213.83
40	5	0.80	13.65	145.54	159.98
50	5	0.80	19.32	139.47	159.58

b) Mínimo grau aproximado

η	σ	REO	PRE	ITE	TOT
0	7	0.58	2.80	174.88	178.26
5	6	0.57	3.61	146.88	151.06
10	6	0.58	6.53	134.86	141.97
15	5	0.57	8.09	114.03	122.70
20	4	0.57	5.98	128.80	135.36
30	4	0.58	23.35	86.49	110.42
40	3	0.58	23.49	70.72	94.78
50	3	0.57	45.84	73.99	120.41

c) Cuthill-McKee reverso

η	σ	REO	PRE	ITE	TOT
0	8	0.59	23.84	320.95	345.38
5	7	0.59	26.13	305.51	332.24
10	6	0.59	27.02	267.19	294.80
15	6	0.59	32.73	239.27	272.59
20	6	0.59	44.64	211.33	256.56
30	6	0.59	65.28	227.29	293.16
40	5	0.59	66.80	217.43	284.83
50	5	0.59	85.93	217.99	304.51

d) Contagem de colunas

Figura 6.35: Tempo de execução em segundos da matriz `nasasbr`.

Estes experimentos mostraram que a reordenação de matrizes é benéfica em relação ao tempo de solução do sistema linear. Mesmo sendo o mínimo grau aproximado o método com maior tempo de reordenação, este atraso é compensado pelo precondicionamento que se torna mais rápido, já que L possui menos elementos. Seguindo este mesmo raciocínio, o tempo gasto nas iterações do gradiente conjugado também é reduzido devido à diminuição da quantidade de elementos da matriz precondicionada. Da mesma forma, como o Cuthill-McKee reverso consegue diminuir o preenchimento da matriz, ele também pode reduzir o tempo de execução em muitos casos.

Capítulo 7

Conclusões

A influência da reordenação na convergência de métodos iterativos precondicionados tem sido considerada por um grande número de autores. O benefício da reordenação sobre a matriz de coeficientes depende em parte da estrutura e da estabilidade da matriz, além do tipo de fatoração incompleta utilizada no precondicionador.

Esta dissertação deu seqüência a uma monografia de conclusão de curso realizada por Carmo e Campos [CC02]. Utilizando o MATLAB, eles observaram que o uso das técnicas de reordenação reduziu o custo de armazenamento das matrizes e o número de iterações necessárias para se chegar à solução do sistema. Seus experimentos revelaram que o Cuthill-McKee reverso apresentou resultados melhores que o mínimo grau múltiplo, o qual mostrou um desempenho melhor quando algum preenchimento era permitido no precondicionador.

Com a utilização do FORTRAN neste trabalho, as condições de testes foram melhoradas, tornando possível a solução de sistemas de maior ordem e em tempo menor do que em MATLAB. Foram feitos experimentos com matriz simétrica definida positiva, usando um precondicionador para o gradiente conjugado baseado na fatoração controlada de Cholesky. Tentou-se estabelecer em qual situação uma determinada reordenação é mais indicada para obter a solução de modo mais eficiente.

Este estudo focaliza apenas a estrutura de esparsidade da matriz, já que os métodos de reordenação não consideram os seus valores numéricos. Desta forma, observou-se que uma redução muito grande no número de iterações pode ser obtida pela reordenação de matrizes com estrutura irregular. Por outro lado, se esta matriz tiver estrutura banda ou próxima de banda, pouco ganho é obtido.

O mínimo grau aproximado foi freqüentemente melhor que as outras reordenações tes-

tadas e muitas vezes melhor que a ordenação original. Como seu propósito é preservar a esparsidade durante a fatoração, torna-se muito útil quando a fatoração sofre um preenchimento extremamente alto. Isto acontece muito com matrizes de estrutura irregular. Nestas condições, este método deve ser recomendado.

A qualidade do precondicionador após a reordenação pelo Cuthill-McKee reverso depende da escolha do nó inicial e da ordenação dos nós dentro de um mesmo nível da estrutura utilizada pelo algoritmo. Ainda não está claro como as diferentes estratégias utilizadas nesta heurística afetam a fatoração. É possível também que algumas ordenações dentro de uma mesmo nível produzam uma seqüência ruim de pivôs e piorem a qualidade da fatoração.

É interessante destacar que nos casos onde o mínimo grau aproximado reduziu o número de iterações, o Cuthill-McKee reverso teve um comportamento parecido com o da ordenação original, não se destacando em relação a convergência.

Apesar de não ser o foco deste trabalho, observou-se que a reordenação não traz melhorias em relação à estabilidade da matriz. Em praticamente todos os casos onde a matriz original desestabilizou-se durante a solução, ela permaneceu desestabilidada após a reordenação.

Em geral, o comportamento do contagem de colunas foi o pior de todos tanto em relação ao número de iterações quanto ao preenchimento. Assim, este método não é recomendado.

Em relação ao tempo de execução, praticamente todas as matrizes que diminuíram o preenchimento também diminuíram este tempo. Isto pode acontecer mesmo em casos onde tenha ocorrido um aumento no número de iterações. O mínimo grau aproximado teve uma redução destacável no tempo de execução do CCCG(η).

Assim sendo, baseado nos experimentos numéricos, conclui-se que muito ganho pode ser obtido pela reordenação de matrizes, particularmente, quando o mínimo grau aproximado é utilizado em matrizes com estrutura irregular.

Considerando possíveis trabalhos futuros, seria interessante testar outros tipos de reordenação com o CCCG(η). Na literatura existem muitas outras reordenações que produzem resultados interessantes, tais como a *Nested Dissection* e a *Red-Black* [DM89]. Além disto, existem também novos métodos que consideram os valores numéricos da matriz, os quais estão tendo resultados promissores [CT95].

Finalmente, existem algumas questões que demandam uma investigação futura. Conforme mencionado, seria importante verificar o efeito que a escolha do nó inicial e a ordenação dentro de um mesmo nível do Cuthill-McKee reverso provocam no desempenho do

CCCG(η). Outra questão seria verificar como matrizes com origem em um mesmo tipo de problema são afetadas pela reordenação. Esta análise iria complementar a verificação da estrutura da matriz, conforme foi realizado neste trabalho.

Referências Bibliográficas

[ADD96] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[BF84] G. A. Behie and P. A. Forsyth. Incomplete factorization methods for fully implicit simulation of enhanced oil recovery. *SIAM Journal on Scientific and Statistical Computing*, 5(3):543–561, 1984.

[Bre73] R. P. Brent. *Algorithms for minimization without derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[BSD99] M. Benzi, D. B. Szyld, and A. Van Duin. Orderings for incomplete factorization preconditioning of nonsymmetric problems. *SIAM Journal on Scientific Computing*, 20(5):1652–1670, 1999.

[BT99] R. Bridson and W. P. Tang. Ordering, anisotropy, and factored sparse approximate inverses. *SIAM Journal on Scientific Computing*, 21(3):867–882, 1999.

[BT00a] M. Benzi and M. Tuma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM Journal on Scientific Computing*, 21(5):1851–1868, 2000.

[BT00b] R. Bridson and W. P. Tang. A structural diagnosis of some IC orderings. *SIAM Journal on Scientific Computing*, 22(5):1527–1532, 2000.

[Cam95] F. F. Campos. *Analysis of conjugate gradients - type methods for solving linear equations*. PhD thesis, Oxford University Computing Laboratory, 1995.

[Cam01] F. F. Campos. *Algoritmos numéricos*. LTC Editora, Rio de Janeiro, 2001.

[CB98] F. F. Campos and N. R. C. Birkett. An efficient solver for multi-rigth hand side linear systems based on the CCCG(η) method with applications to implicit time-dependent partial differential equations. *SIAM Journal on Scientific Computing*, 19(1):126–138, 1998.

[CC02] F. C. Carmo and F. F. Campos. Algoritmos para reordenação de matrizes esparsas. Technical Report RT001/2002, Departamento de Ciência da Computação, UFMG, Belo Horizonte, 2002.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to algorithms*. The MIT Press, 1990.

[CM69] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 24th National Conference ACM*, pages 157–172, New York, 1969.

[CR95a] F. F. Campos and J. S. Rollet. Analysis of preconditioners for conjugate gradients througth distribution of eigenvalues. *International Journal of Computer Mathematics*, 58:135–158, 1995.

[CR95b] F. F. Campos and J. S. Rollet. Controlled Cholesky factorisation for preconditioning the conjugate gradient method. Technical Report NA95/05, Oxford University Computing Laboratory, Oxford, 1995.

[CT95] S. S. Clift and W. P. Tang. Weighted graph based ordering techniques for preconditioned conjugate gradient methods. *BIT*, 35(1):30–47, 1995.

[DADR] T. A. Davis, P. R. Amestoy, I. S. Duff, and J. K. Reid. Netlib Repository. Available at < http://www.netlib.org/linalg/amd/amdbar.f >. Acess on March 06, 2005.

[Dav] T. A. Davis. University of Florida sparse matrix collection. Available at < http://www.cise.ufl.edu/research/sparse/matrices/ >. Acess on March 04, 2005.

[DER86] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices*. Oxford University Press, Oxford, 1986.

[DFT92] E. F. D'Azevedo, P. A. Forsyth, and W. P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM Journal on Matrix Analysis and Applications*, 13(3):994–961, 1992.

[DGL92] I. S. Duff, R. G. Grimes, and J. G. Lewis. User's guide for the Harwell-Boeing sparse matrix collection (release I). Technical Report TR/PA/92/86, CERFACS, Toulouse, France, 1992.

[DM89] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradients. *BIT*, 29(4):635–657, 1989.

[DMW68] A. Dubrulle, R. S. Martin, and J. H. Wilkinson. The implicit QL algorithm. *Numerische Mathematik*, 12:377–383, 1968.

[Dut93] L. C. Dutto. The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Engineering*, 36:457–497, 1993.

[EA89] H. Elman and E. Agron. Ordering techniques for the preconditioned conjugate gradient method on parallel computers. *Computer Physics Communications*, 53:253–269, 1989.

[FS55] G. Forsythe and E. Straus. On best conditioned matrices. In *Proceedings of the Amer. Math. Soc.*, volume 6, pages 340–345, 1955.

[Geo71] J.A. George. *Computer implementation of the finite element method*. PhD thesis, Department of Computer Science, Stanford University, 1971.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness*. Freeman, New York, 1979.

[GL79] A. George and J. W. H. Liu. An implementation of a pseudoperipheral node finder. *ACM Transactions on Mathematical Software*, 5(3):284–295, 1979.

[GL80] A. George and J. Liu. A fast implementation of the minimum degree algorithm using quotient graphs. *ACM Transactions on Mathematical Software*, 6(3):337–358, 1980.

[GL81] A. George and J. Liu. *Computer solution of large sparse positive definite systems*. Prentice Hall, New Jersey, 1981.

[GL89] A. George and J. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[GL96] G. H. Golub and C. F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, 1996.

[HEKP01] P. Heggernes, S. C. Eisenstat, G. Kumfert, and A. Pothen. Complexity of the minimum degree algorithm. Technical Report 2001-42, NASA Langley Research Center, Hampton, 2001.

[HS52] M. R. Hestenes and E. Steifel. Methods of conjugate gradients for solving linear systems. *Journal of Research of National Bureau of Standarts*, 49:409–436, 1952.

[JP95] M. T. Jones and P. E. Plassmann. An improved incomplete Cholesky factorization. *ACM Transactions on Mathematical Software*, 21(1):5–17, 1995.

[Lan89] H. P. Langtangen. Conjugate gradient methods and ILU preconditioning of non-symmetric matrix systems with arbitrary sparsity patterns. *International Journal for Numerical Methods in Fluids*, 9:213–223, 1989.

[Liu85] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, 1985.

[Ltd90] NAG Ltd. The NAG fortran library manual – Mark 14, 1990.

[Man80] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34:473–497, 1980.

[Mat00] The MathWorks, Inc., Natick, Massachusetts. *MATLAB reference guide*, sixth edition, 2000.

[MdV77] J. A. Meijerink and H. A. Van der Vorst. An iterative solution method for linear system of which the coefficient matrix is a symmetric M-matrix. *Math. Comp*, 31:148–162, 1977.

[ML86] G. R. Mateus and H. P. L. Luna. *Programação não linear*. Editora Gráfica Formato, 1986.

[Mun80] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients. *ACM Transactions on Mathematical Software*, 6(2):206–219, 1980.

[NR97] Esmond G. Ng and P. Raghavan. Performance of greedy ordering heuristics for sparse Cholesky factorization. Technical Report CS-97-XX, University of Tennessee, Knoxville, TN, 1997.

[Ort91] J. M. Ortega. Orderings for conjugate gradient preconditionings. *SIAM Journal on Optimization*, 1(4):565–582, 1991.

[Par61] S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3:119–130, 1961.

[PD] E. Pohl and J. Dietel. European Synchrotron Radiation Facility library. Available at < http://www.esrf.fr/computing/expg/libraries/numeric/fortra77/kap04/cutcho.htm >. Acess on March 06, 2005.

[RIA] Research Institute for Advanced Computer Science. Available at < http://www.riacs.edu/ >. Acess on August 03, 2005.

[Saa96] Y. Saad. *Iterative methods for sparse linear systems*. PWS Publishing Company, Minnesota, 1996.

[She94] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, Carnegie Mellon University, Pittsburgh, PA, 1994.

[Sim85] H. D. Simon. Incomplete LU preconditioners for conjugate-gradient-type iterative methods. In *Proceedings of the 1985 SPE Reservoir Simulation Symposium*, pages 387–396, Texas, 1985.

[Sto68] H. L. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM Journal on Numerical Analysis*, 5(3):530–558, 1968.

[TB97] L. N. Trefethen and D. Bau. *Numerical linear algebra*. SIAM, Philadelphia, 1997.

[ZS01] D. Zundel and W. Schönauer. A fast "parallelized"single pass bandwidth optimizer for sparse matrices. In *Proceedings of International Conference on Numerical Algorithms*, pages 1–5, Marrakesh, Morocco, 2001.