Capítulo 7

Tratamento numérico de matrizes esparsas

7.1 Considerações iniciais

O capítulo anterior permitiu verificar que os modelos híbrido-mistos de elementos finitos conduzem, regra geral, à obtenção de sistemas governativos de grandes dimensões, mas caracterizados por um elevado índice de esparsidade. Este facto é ainda mais evidente quanto se utilizam séries de Walsh na aproximação dos campos estático e cinemático.

Para assegurar a eficiência numérica dos modelos apresentados, torna-se imprescindível a utilização de algoritmos especialmente desenvolvidos para permitir um tratamento eficaz deste tipo de matrizes. Para explorar ao máximo a esparsidade das matrizes e em simultâneo garantir um bom desempenho do processo de cálculo, estes algoritmos devem respeitar as seguintes regras básicas [57, 59, 165]:

- Apenas se devem armazenar os coeficientes não-nulos presentes nas matrizes;
- Só se devem efectuar operações envolvendo coeficientes não-nulos, sendo de evitar todos os cálculos redundantes envolvendo elementos nulos;
- Deve-se tentar preservar a esparsidade das matrizes, tanto quanto o permitam os métodos de cálculo utilizados.

Os algoritmos que permitem trabalhar directamente com matrizes esparsas são mais complexos e pesados que os algoritmos utilizados no tratamento de matrizes cheias ou de matrizes armazenadas em *banda* ou em *perfil*. Esta complexidade resulta não só da necessidade de se evitarem sequências de operações redundantes, mas também do tipo de armazenamento utilizado, onde o acesso à informação é efectuado de uma

forma *indirecta*. A aplicação e a programação de tais algoritmos deve ser efectuada de uma forma cuidada para que se possa garantir a exploração eficaz do facto de existir apenas um número muito pequeno de coeficientes não-nulos envolvidos nos cálculos. É necessário evitar, por exemplo, a execução de *buscas* de valores envolvendo sequências de testes a todos os elementos armazenados.

São neste capítulo referidos alguns dos aspectos essenciais envolvidos no armazenamento e manipulação de matrizes esparsas. São apresentadas primeiro algumas das formas de armazenamento mais utilizadas, sendo discutidas as vantagens e inconvenientes associados à utilização de cada uma delas. São apresentados em apêndice alguns dos algoritmos que permitem efectuar de uma forma eficaz algumas operações simples envolvendo matrizes esparsas. No desenvolvimento desses algoritmos é de capital importância a sequência através da qual se utilizam os coeficientes das matrizes envolvidas no cálculo, pois a eficiência com que tais coeficientes são localizados é fortemente condicionada pela estrutura de dados utilizada no armazenamento, sendo muito usual existir uma sequência de acesso preferencial.

Quando se efectua uma análise elástica linear com os modelos híbrido-mistos de elementos finitos, é a solução dos sistemas de equações que condiciona a eficiência de todo o processo de cálculo. Será mesmo na maioria das situações responsável por larga percentagem do tempo total de CPU dispendido em toda a análise. Também quando se realiza uma análise elastoplástica, uma parte substancial do esforço de cálculo é concentrado na resolução dos sistemas de equações lineares que resultam da expansão em série dos sistemas governativos não-lineares. Compreende-se desta forma a necessidade de se utilizarem algoritmos de resolução suficientemente eficazes para evitar a degradação do desempenho do modelo. Grande parte deste capítulo é dedicado à apresentação e discussão dos métodos de resolução de sistemas esparsos.

São utilizados tanto *métodos directos* quanto *métodos iterativos*. Os métodos directos são os mais utilizados na resolução dos sistemas de equações associados à aplicação do método dos elementos finitos. São várias as razões que podem ser apresentadas para explicar este facto. Os métodos directos apresentam a vantagem de ser possível garantir *a priori* que a solução é obtida ao fim da execução de um número finito (e conhecido à partida) de operações. Permite ainda o reaproveitamento da factorização da matriz dos coeficientes para efectuar a solução do mesmo sistema com outros termos independentes. Permitem, por outro lado, obter tempos de resolução inferiores aos obtidos com métodos iterativos para os exemplos de pequena e média dimensão.

É quando a dimensão e a esparsidade dos sistemas governativos aumenta que os métodos iterativos começam a ser competitivos. Embora a convergência do processo envolva um número indeterminado de operações e seja necessário discutir qual o critério de paragem mais adequado, a sua utilização começa a ser atractiva pela enorme economia que podem proporcionar em termos de memória envolvida no armazenamento dos coeficientes. Tipicamente, a aplicação de um método iterativo implica apenas o armazenamento da matriz dos coeficientes na sua forma inicial e

de um pequeno número de vectores de dimensão igual à dimensão do sistema. Por vezes é ainda necessário armazenar a *matriz de pré-condicionamento*, utilizada para *acelerar* o processo de convergência.

Quando se utilizam métodos directos, para além da matriz inicial, é necessário armazenar a matriz factorizada. Durante o processo de eliminação, passam a ser diferentes de zero alguns coeficientes que eram nulos à partida. É este fenómeno, designado geralmente por *fill-in*, que condiciona drasticamente a memória que se torna necessário disponibilizar e também o volume da cálculos a efectuar. Para sistemas de grandes dimensões, este fenómeno pode degradar significativamente a eficiência dos algoritmos directos ou mesmo impossibilitar a sua utilização.

Não é só a redução da necessidade de armazenamento que torna apetecível a utilização de algoritmos iterativos. Também se consegue assegurar em muitas circunstâncias melhores tempos de execução. Deve chamar-se no entanto a atenção para o facto da aplicação dos algoritmos iterativos na sua forma *pura* não ser em geral muito eficiente. Para que tais métodos sejam competitivos, é necessário utilizar formas para acelerar a convergência, recorrendo-se a *matrizes de pré-condicionamento*. Surgem os chamados *métodos mistos*, nos quais se utiliza um método directo na construção do pré-condicionador e um método iterativo na resolução do sistema de equações propriamente dito.

Este capítulo termina com a discussão de alguns resultados numéricos e a apresentação de algumas conclusões, onde se procura comparar os desempenhos dos diferentes algoritmos testados.

7.2 Armazenamento de matrizes esparsas

Nas formulações clássicas de elementos finitos, as matrizes de rigidez são geralmente armazeadas em *semi-banda* ou em *perfil* [103]. No caso dos modelos híbrido-mistos aqui apresentados, a utilização de tais estruturas de armazenamento revela-se completamente inadequada. Basta observar a distribuição dos coeficientes não-nulos da matriz representada na figura 6.8 para perceber de imediato que a adopção de qualquer uma daquelas duas estruturas de dados conduz inevitavelmente ao armazenamento de um elevado número de coeficientes nulos.

Antes de se apresentarem as estruturas que permitem armazenar apenas os coeficientes não-nulos realmente existentes nas matrizes esparsas, convém verificar que os vectores também podem ser armazenados numa *forma compacta*, através da utilização de dois vectores com uma dimensão igual ao número de elementos diferentes de zero existentes. No primeiro vector armazenam-se os valores dos coeficientes, enquanto que no segundo se registam as posições ocupadas por cada um deles no vector inicial.

O armazenamento de vectores na forma compacta só faz sentido quando pelo menos metade dos seus elementos são nulos. Caso contrário, o volume de informação a guardar será maior na forma compacta do que na forma inicial. Por outro lado, deve ter-se em conta que é superior o tempo de acesso a um qualquer elemento do vector armazenado na forma compacta, pois o *endereçamento* é efectuado de uma forma *indirecta*. Tendo em conta este facto, e para que a forma compacta conduza também a uma redução dos tempos de execução dos cálculos em que tal operador se encontre envolvido, é necessário que o vector apresente um índice de esparsidade elevado, da ordem dos 50%-75% [59] para casos envolvendo processamento sequencial.

$$\mathbf{B} = \begin{bmatrix} 3 & 0 & 0 & -1 & 0 & 0 & 7 & 0 \\ 0 & 5 & 1 & 0 & 0 & 5 & 2 & 0 \\ 0 & 1 & 3 & 2 & 0 & 0 & 9 & 0 \\ -1 & 0 & 2 & 4 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 5 & 0 & 1 & 0 & 3 & 0 & 4 \\ 7 & 2 & 9 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 3 & 0 & 4 & 1 & 8 \end{bmatrix} . \tag{7.1}$$

Seja uma matriz esparsa com n_{nz} coeficientes não-nulos. A forma mais simples de efectuar o armazenamento de tal matriz consiste na utilização de três vectores de dimensão n_{nz}. No primeiro, **VAL**, armazenam-se os valores dos coeficientes não-nulos existentes. Os índices correspondentes às linhas e colunas de cada um daqueles coeficientes são armazenados nos dois restantes vectores, **IRN** e **ICN**, respectivamente. A matriz **B** definida em (7.1) pode ser representada na forma:

$$\mathbf{VAL} = \begin{bmatrix} 4 & 2 & -1 & 8 & 1 & 3 & 2 & 7 & 3 & 1 & 1 & 1 & 3 & 5 & 5 & 4 & 3 & 9 & 2 \end{bmatrix}$$

$$\mathbf{IRN} = \begin{bmatrix} 4 & 5 & 1 & 8 & 2 & 1 & 2 & 1 & 6 & 7 & 7 & 4 & 3 & 2 & 2 & 6 & 4 & 3 & 3 \end{bmatrix}$$

$$\mathbf{ICN} = \begin{bmatrix} 4 & 5 & 4 & 8 & 3 & 1 & 7 & 7 & 6 & 8 & 7 & 6 & 3 & 2 & 6 & 8 & 8 & 7 & 4 \end{bmatrix}$$

Esta estrutura de dados é muitas vezes denominada *esquema coordenado* (*coordinate scheme*) de armazenamento. Quando a matriz é simétrica, guardam-se apenas os valores localizados no triângulo superior (ou inferior). Verifica-se que não é necessário armazenar os valores por ordem crescente da linha e/ou coluna a eles associada.

Embora seja muito simples, esta estrutura de armazenamento apresenta uma grande limitação: torna bastante pesado e moroso o processo de obtenção de todos os coeficientes situados numa mesma linha ou coluna. A sua utilização é perfeitamente possível quando se pretende multiplicar a matriz por um vector cheio, situação que é típica na aplicação de algoritmos iterativos na resolução de sistemas de equações. Quando se pretende utilizar um método directo, que envolve sequências de operações com linhas (ou colunas), é fortemente desaconselhada a utilização desta estrutura.

Registe-se no entanto que, para simplificar a definição dos dados, a grande maioria dos programas comerciais existentes no mercado utilizam esta estrutura para a definição, por parte do utilizador, das matrizes esparsas envolvidas nos cálculos. Os programas procedem depois à conversão da forma de armazenamento, sendo criadas internamente, e sem qualquer intervenção por parte do utilizador, as estruturas de dados mais convenientes e eficazes para o tipo de operações a efectuar.

Uma das estruturas de armazenamento mais utilizadas foi inicialmente proposta por Gustavson [93] e é geralmente conhecida na literatura como *Sparse Row-Wise Format*. Tal como o seu nome indica, consiste em definir a matriz como sendo constituída por uma lista de vectores correspondentes a cada uma das linhas. Estas são definidas de uma *forma compacta* e devem ser armazenadas sequencialmente.

Desta forma, a representação de matrizes esparsas é conseguida à custa da definição de um conjunto de três vectores. No primeiro, o vector **AN**, os valores dos coeficientes não-nulos são armazenados por linhas. No segundo, **JA**, define-se a coluna associada a cada coeficiente listado em **AN**. Finalmente, define-se um vector de apontadores, **IA**, que permite identificar onde se inicia, em **AN** e **JA**, a definição dos valores referentes a cada uma das linhas.

Se N_E for a dimensão da matriz, então **IA** tem N_E coeficientes, enquanto que **AN** e **JA** são vectores com dimensão n_{nz}. É necessário ainda definir na posição $N_E + 1$ uma entrada adicional em **IA** por forma a ser possível identificar a primeira posição vazia em **AN** e **JA**. A matriz **B** pode ser armazenada na forma:

$$\mathbf{AN} = \left[\begin{array}{ccc|cccc|ccc|ccc|c|cc|c} 3 & -1 & 7 & 5 & 1 & 5 & 2 & 3 & 2 & 9 & 4 & 1 & 3 & 2 & 3 & 4 & 1 & 1 & 8 \end{array} \right]$$

$$\mathbf{JA} = \left[\begin{array}{ccc|cccc|ccc|ccc|c|cc|c} 1 & 4 & 7 & 2 & 3 & 6 & 7 & 3 & 4 & 7 & 4 & 6 & 8 & 5 & 6 & 8 & 7 & 8 & 8 \end{array} \right]$$

$$\mathbf{IA} = \left[\begin{array}{cccccccccc} 1 & 4 & 8 & 11 & 14 & 15 & 17 & 19 & 20 \end{array} \right]$$

Neste processo as linhas têm que ser armazenadas por ordem. No entanto, e dentro de uma mesma linha, não é obrigatório que os coeficientes sejam guardados por ordem crescente do valor da coluna a eles associada. Esta representação *não-ordenada* é em geral bastante conveniente. O resultado da maioria das operações matriciais é obtido nesta forma e seria computacionalmente oneroso ordenar a representação resultante. Por outro lado, verifica-se que os algoritmos de manipulação de matrizes esparsas não obrigam, com algumas raras excepções [165], a que as representações sejam ordenadas.

De acordo com a definição dos apontadores listados no vector **IA**, a parte da linha l contida no triângulo superior da matriz é descrita pelas posições $\mathbf{IA}(l)$ até $\mathbf{IA}(l+1)-1$ de **AN** e **JA**, excepto quando $\mathbf{IA}(l+1) = \mathbf{IA}(l)$, o que significa que nessa linha não existe nenhum coeficiente com valor diferente de zero.

Muitas vezes os elementos da diagonal principal são armazenados em separado, num outro vector de dimensão N_E, aqui designado por **AD**. A representação da matriz **B** toma neste caso a forma:

$$\mathbf{AN} = \begin{bmatrix} -1 & 7 & 1 & 5 & 2 & 2 & 9 & 1 & 3 & 4 & 1 \end{bmatrix}$$

$$\mathbf{JA} = \begin{bmatrix} 4 & 7 & 3 & 6 & 7 & 4 & 7 & 6 & 8 & 8 & 8 \end{bmatrix}$$

$$\mathbf{IA} = \begin{bmatrix} 1 & 3 & 6 & 8 & 10 & 10 & 11 & 12 & 12 \end{bmatrix}$$

$$\mathbf{AD} = \begin{bmatrix} 3 & 5 & 3 & 4 & 2 & 3 & 1 & 8 \end{bmatrix}$$

Esta estrutura de armazenamento, designada muitas vezes por RR(D)U [165], permite identificar com facilidade todos os elementos pertencentes a uma dada linha. No entanto, a sua utilização não está isenta de dificuldades. A mais marcante surge quando se pretende inserir uma nova entrada. Esta é uma situação típica dos processos de factorização, quando um múltiplo de uma linha é adicionado a uma outra. Há então a necessidade de se *criar* espaço para permitir o armazenamento dos novos coeficientes não-nulos. Não é difícil verificar que esta estrutura de armazenamento, ao impor que as linhas se encontrem *arrumadas* sequencialmente e sem qualquer *folga* entre elas, não é muito flexível quando se trata de resolver este problema. A prática usual [59] consiste então em *desperdiçar* temporariamente o espaço inicialmente ocupado pela linha à qual se pretendem adicionar novas entradas e *duplicar* a informação, listando no fim da estrutura a nova *definição* da linha em causa. É preciso também alterar o valor do apontador correspondente em **IA**, uma vez que a *definição* daquela linha se inicia agora numa outra posição dos vectores **JA** e **AN**. Quando o espaço *desperdiçado* no interior da estrutura começa a ser excessivo, devem recolocar-se as linhas nas suas posições iniciais.

A adopção desta estratégia implica que os coeficientes não-nulos de uma dada linha, l, deixem de ser representados pelas posições compreendidas entre $\mathbf{IA}(l)$ e $\mathbf{IA}(l+1)-1$. Apenas se pode garantir que a definição da linha se inicia em $\mathbf{IA}(l)$. Torna-se necessário definir um vector auxiliar, também de dimensão N_E, que tem como finalidade registar o número de coeficientes não-nulos existentes de facto em cada linha. A introdução de um pequeno artifício permite, no entanto, evitar a utilização deste vector adicional; consiste em inverter o sinal do valor em **JA** que corresponde ao último elemento de uma dada linha. Torna-se então possível identificar de uma forma automática a posição do último coeficiente pertencente a cada uma das linhas da matriz.

A utilização das *linked lists* [59] permite obter uma estrutura de armazenamento muito flexível e eficaz e que permite o acesso automático a todos os elementos de uma mesma linha sem que para isso se tenham que armazenar por ordem todas as linhas da matriz. Permite ainda adicionar ou mesmo remover entradas de uma forma simples, sem que para tal seja necessário utilizar os procedimentos descritos nos parágrafos anteriores.

O armazenamento de matrizes passa neste caso pela utilização de quatro vectores. Dois deles correspondem aos vectores **VAL** e **ICN** e permitem efectuar o armazenamento do valor de cada um dos coeficientes não-nulos e a coluna a que cada um deles

pertence. O terceiro vector, com dimensão N_E e desigado aqui por **IRNS**, tem como finalidade listar, na posição l, a localização em **VAL** e **ICN** do primeiro coeficiente não-nulo existente na linha l da matriz. Finalmente, o vector de apontadores **LINK** permite identificar a localização do próximo elemento pertencente à mesma linha. Um valor nulo em **LINK**, que tem como dimensão n_{nz}, significa que a definição da linha em causa já terminou. A matriz **B** pode ser apresentada na forma:

$$\mathbf{IRNS} = \left[\begin{array}{cccccccc} 6 & 14 & 13 & 1 & 2 & 9 & 11 & 4 \end{array}\right]$$

$$\mathbf{VAL} = \left[\begin{array}{cccccccccccccccccc} 4 & 2 & -1 & 8 & 1 & 3 & 2 & 7 & 3 & 1 & 1 & 1 & 3 & 5 & 5 & 4 & 3 & 9 & 2 \end{array}\right]$$

$$\mathbf{ICN} = \left[\begin{array}{cccccccccccccccccc} 4 & 5 & 4 & 8 & 3 & 1 & 7 & 7 & 6 & 8 & 7 & 6 & 3 & 2 & 6 & 8 & 8 & 7 & 4 \end{array}\right]$$

$$\mathbf{LINK} = \left[\begin{array}{cccccccccccccccccc} 12 & 0 & 8 & 0 & 15 & 3 & 0 & 0 & 16 & 0 & 10 & 17 & 19 & 5 & 7 & 0 & 0 & 0 & 18 \end{array}\right]$$

Se se pretender adicionar uma nova entrada numa qualquer linha da matriz, as alterações a efectuar na estrutura de dados são mínimas. A título de exemplo, considere-se que se pretende adicionar um novo coeficiente não-nulo, $B(2,4) = -2$, à matriz definida em (7.1). Para o fazer, há apenas que considerar $VAL(20) = -2$, $ICN(20) = 4$ e efectuar uma pequena troca de índices no vector **LINK**, que passa agora a ser dado por:

$$\mathbf{LINK} = \left[\begin{array}{cccccccccccccccccc} 12 & 0 & 8 & 0 & 20 & 3 & 0 & 0 & 16 & 0 & 10 & 17 & 19 & 5 & 7 & 0 & 0 & 0 & 18 & 15 \end{array}\right]$$

7.3 Operações envolvendo matrizes esparsas

Muitas das operações envolvendo matrizes esparsas podem ser transformadas numa sequência conveniente de operações envolvendo vectores. No apêndice E apresenta-se um conjunto de algoritmos que permitem realizar as seguintes operações básicas: produto interno de vectores, combinações lineares, produtos matriz por vector e produtos de matriz por matriz. Esses algoritmos seguem de perto o disposto em [59].

7.4 Métodos directos na resolução de sistemas lineares

Considere-se o sistema de equações lineares escrito na forma,

$$\mathbf{A}\,\mathbf{x} = \mathbf{b}\ , \tag{7.2}$$

onde **A** é uma matriz simétrica e onde **x** e **b** representam o vector solução e os termos independentes, respectivamente. É possível demonstrar [164] que a matriz **A**, sempre que seja não-singular, pode ser expressa na forma,

$$\mathbf{A} = \mathbf{U}^t\mathbf{D}\,\mathbf{U}\ , \tag{7.3}$$

onde **U** é uma matriz triangular superior com diagonal unitária e **D** é uma matriz diagonal.

A factorização da matriz dos coeficientes pode ser efectuada através da aplicação de um processo de *eliminação* de Gauss, ou de uma *decomposição* de Choleski caso **A** seja uma matriz positiva definida. Sendo esparsa a matriz inicial, também o serão os seus factores. No entanto, e devido ao *fill-in*, o índice de esparsidade associado às matrizes factores pode ser significativamente inferior ao índice de esparsidade da matriz inicial.

O *fill-in* condiciona de uma forma muito marcante a eficiência numérica dos métodos directos. Tendo em atenção que este fenómeno depende directamente da distribuição dos coeficientes não-nulos em **A**, a sua minimização passa pela realização de uma *reordenação* da matriz dos coeficientes antes da aplicação do algoritmo de factorização. Para ilustrar a importância que esta reordenação pode ter, repare-se no caso da matriz $\mathbf{A}_1$, apresentada em (7.4). Não é difícil verificar que quando se aplica um processo de eliminação de Gauss se obtém uma matriz cheia. Já quando se factoriza a matriz $\mathbf{A}_2$, obtida directamente de $\mathbf{A}_1$ através da permutação da primeira e última linha (e coluna), não surge nenhuma nova entrada; o *fill-in* é nulo. É importante ter em conta que esta é uma situação extrema. No entanto, a aplicação de uma reordenação criteriosa conduz, regra geral, a uma melhoria muito substancial do desempenho do método directo de resolução.

$$\mathbf{A}_1 = \begin{bmatrix} 4 & 1 & 2 & 3 & 1 & 2 \\ 1 & 2 & 0 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 & 0 \\ 3 & 0 & 0 & 6 & 0 & 0 \\ 1 & 0 & 0 & 0 & 4 & 0 \\ 2 & 0 & 0 & 0 & 0 & 5 \end{bmatrix} ; \mathbf{A}_2 = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 & 0 & 2 \\ 0 & 0 & 0 & 6 & 0 & 3 \\ 0 & 0 & 0 & 0 & 4 & 1 \\ 2 & 1 & 2 & 3 & 1 & 4 \end{bmatrix} \tag{7.4}$$

Existem na literatura [59, 88] diferentes algoritmos que permitem efectuar uma *reordenação simétrica* da matriz dos coeficientes com o intuito de minimizar o *fill-in*. Infelizmente, o ganho associado a cada um daqueles métodos depende fortemente do tipo de matriz em causa. Não se pode prever à partida qual o que conduz a melhores desempenhos.

O desenvolvimento destes métodos de reordenação baseia-se muitas vezes na aplicação de um critério muito simples, proposto inicialmente por Markowitz [128]: o *pivot* a escolher em cada etapa do processo de eliminação deve ser o elemento a_{ij} que, pertencendo à sub-matriz ainda não factorizada, permita obter um valor mínimo para o produto entre o número de entradas existentes na linha i e o número de coeficientes não-nulos existentes na coluna j.

Os algoritmos de resolução directa de sistemas esparsos costumam ser subdivididos em três fases distintas [59, 87, 165]: *factorização simbólica*, *factorização numérica*, e *obtenção da solução*.

Na primeira etapa de cálculo, conhecida vulgarmente por *factorização simbólica*, apenas se lida com a *distribuição* das entradas na matriz dos coeficientes, não sendo sequer necessário conhecer o seu valor numérico. É nesta fase que se aplicam os algoritmos de reordenação, se obtém a distribuição dos coeficientes não-nulos da matriz factorizada e é *preparada* a estrutura de dados que vai permitir efectuar o seu armazenamento.

O cálculo dos valores dos elementos das matrizes $\mathbf{D}$ e $\mathbf{U}$ é efectuado na *factorização numérica*. O processo de eliminação que conduz à sua determinação segue a sequência de *pivotagem* definida na fase anterior. É importante ter em conta que quando a matriz do sistema é positiva definida, a *estabilidade numérica* do processo é garantida à partida. Pelo contrário, quando a matriz é *indefinida*, podem surgir problemas durante a fase de factorização. Nestes casos, pode acontecer que num dado instante o *pivot* a utilizar nos cálculos seguintes tenha um valor muito pequeno, ou mesmo nulo. Para evitar problemas de instabilidade numérica ou mesmo a interrupção anormal do processo de resolução, deve alterar-se a sequência de *pivots* estabelecida durante a factorização simbólica.

Uma vez factorizada a matriz dos coeficientes, a solução do problema é obtida atavés de uma *substituição directa* e uma *retrosubstituição*. Tendo em conta a decomposição apresentada em (7.3), a solução do sistema de equações pode ser então obtida a partir da aplicação da seguinte sequência de operações:

$$\mathbf{U}^t\mathbf{w} = \mathbf{b}\ ;\ \mathbf{D}\,\mathbf{y} = \mathbf{w}\ ;\ \mathbf{U}\mathbf{x} = \mathbf{y}\ .$$

A separação do algoritmo de resolução em três fases bem distintas permite uma fácil e eficiente reutilização dos resultados parcelares. Assim, quando se pretendem resolver vários sistemas de equações com termos independentes diferentes mas com a mesma matriz dos coeficientes, apenas há que aplicar repetidas vezes a terceira etapa de cálculo, aproveitando-se sempre a informação gerada pelas duas primeiras. Outra situação que por vezes surge é a existência de conjuntos de sistemas de equações com matrizes que, embora possuam elementos com valores numéricos diferentes, partilham a mesma distribuição de coeficientes não-nulos. Nestes casos é possível reutilizar a informação gerada pela factorização simbólica.

Para a resolução directa dos sistemas de equações, foi implementado neste trabalho um algoritmo apresentado por Pissanetzky [165], seguindo uma ideia original de Gustavson [93]. Este algoritmo baseia-se na aplicação de um processo de *eliminação por linhas*. Esta variante do método de Gauss permite obter um método de resolução mais eficiente, uma vez que o acesso aos coeficientes é efectuado segundo a sua sequência natural de armazenamento.

O algoritmo de Pissanetzky não prevê a reordenação da matriz dos coeficientes; os *pivots* são seleccionados de entre os elementos da diagonal principal de uma forma sequencial. Esta é a principal limitação deste algoritmo. Como não é prevista qualquer técnica de minimização de *fill-in*, é de esperar uma forte degradação do seu desempenho, sobretudo à medida que o número de entradas armazenadas vai

crescendo. Para contornar esta dificuldade e permitir a utilização de um método directo mais sofisticado e eficaz, foi utilizado um *package* comercial, o MA47 [96], da *Harwell Subroutine Library*.

Este programa prevê especificamente a resolução de sistemas de equações cuja matriz dos coeficientes é *indefinida*, e utiliza uma técnica de *resolução multifrontal* desenvolvida por Duff e Reid [58]. A reordenação da matriz dos coeficientes é realizada à custa da aplicação de uma variante local do critério de Markowitz, denominada por algoritmo do *grau mínimo* (*minimum degree algorithm*) [59].

De acordo com o trabalho de Bunch e Parlett [26], a sequência de pivotagem pré-estabelecida pode ser alterada durante a factorização numérica por forma a evitar-se a utilização de um *pivot* com um valor absoluto muito pequeno (ou mesmo nulo). Quando tal não é possível evitar, é substituida a utilização de 2 pivots por um bloco de pivotagem de 2×2.

7.5 Métodos iterativos na resolução de sistemas lineares

7.5.1 Método dos gradientes conjugados

O método dos gradientes conjugados foi apresentado em 1952 por Hestenes e Stiefel [98]. Foi no entanto apenas muitos anos depois que foi utilizado pela primeira vez como um método iterativo para a resolução de sistemas lineares esparsos [174]. Desde então, o algoritmo dos gradientes conjugados tem conhecido uma difusão e aceitação cada vez maiores, sendo considerado hoje em dia como um dos métodos iterativos mais versáteis e competitivos.

Pode demonstrar-se [164] que em *aritmética exacta* e sempre que a matriz dos coeficientes é *positiva definida*, o algoritmo dos gradientes conjugados permite obter a solução exacta do sistema (7.2) ao fim de um máximo de N iterações, onde N representa a dimensão da matriz $\mathbf{A}$.

Seja $\mathbf{x}_i$ uma solução aproximada do sistema (7.2). A aplicação do algoritmo dos gradientes conjugados, a partir de agora denotado por *CG (conjugate gradients)*, permite calcular uma nova aproximação com base na igualdade:

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \, \mathbf{p}_i \ . \tag{7.5}$$

Em (7.5), os vectores $\mathbf{p}_i$ definem as chamadas *direcções de busca* e α_i é o parâmetro que controla a dimensão do passo a efectuar.

As direcções de busca formam um sistema de vectores $\mathbf{A}$-*conjugados*,

$$\mathbf{p}_i^t \mathbf{A} \, \mathbf{p}_j = 0 \ , para \ i \neq j \ . \tag{7.6}$$

Em cada passo do processo iterativo os vectores $\mathbf{p}_i$ podem ser determinados a partir da igualdade,

$$\mathbf{p}_{i+1} = \mathbf{r}_i + \beta_i \, \mathbf{p}_i \ ,$$

onde o vector *resíduo* $\mathbf{r}_i$ corresponde a

$$\mathbf{r}_i = \mathbf{b} - \mathbf{A}\,\mathbf{x}_i \ , \tag{7.7}$$

e o valor do coeficiente β_i é determinado por forma a que se satisfaça a condição (7.6).

É possível demonstrar [164] que os *resíduos* formam um sistema de vectores ortogonais, verificando-se:

$$\mathbf{r}_i^t \, \mathbf{r}_j = 0, \ , para \ i \neq j \ .$$

O algoritmo dos gradientes conjugados garante ainda que em cada iteração é minimizada a função $E_i = \mathbf{r}_i^t \mathbf{A}^{-1} \mathbf{r}_i$. Esta quantidade pode ser interpretada como representando a *norma quadrática* do vector resíduo, $\mathbf{r}_i$.

O número de iterações envolvido na resolução de (7.2) é desconhecido à partida e depende fortemente do *condicionamento* da matriz dos coeficientes. Este pode ser quantificado com recurso ao chamado *número de condição* de $\mathbf{A}$, definido através da igualdade

$$\kappa(\mathbf{A}) = \|\mathbf{A}\|_2 \left\|\mathbf{A}^{-1}\right\|_2 = \frac{\lambda_n}{\lambda_1} \ ,$$

onde λ_n e λ_1 representam os valores absolutos do maior e do menor *valor próprio* da matriz dos coeficientes, respectivamente. Quanto maior for o valor de $\kappa(\mathbf{A})$, pior será o condicionamento da matriz do sistema e maior será o número de iterações necessárias para garantir a convergência do processo.

Ao contrário do que se verifica com os algoritmos de resolução directa de sistemas, a quantidade de operações requeridas pela aplicação do método dos gradientes conjugados depende fundamentalmente do valor numérico dos coeficientes existentes na matriz do sistema, sendo muito pouco sensível à sua distribuição. Desta forma, a resolução iterativa de dois sistemas envolvendo matrizes que possuam exactamente a mesma distribuição de elementos diferentes de zero, mas com outros valores numéricos, pode exigir volumes de cálculo substancialmente diferentes.

Existem diferentes processos que possibilitam *acelerar* [146] a convergência do algoritmo dos gradientes conjugados. Um dos métodos mais utilizadas e eficazes consiste em tentar diminuir o *número de condicionamento* da matriz através da aplicação de uma técnica de *pré-condicionamento*. Esta consiste em *substituir* implicitamente o sistema inicial (7.2) pelo sistema equivalente,

$$\mathbf{M}^{-1}\mathbf{A}\,\mathbf{x} = \mathbf{M}^{-1}\,\mathbf{b} \ ,$$

onde $\mathbf{M}$ representa a chamada *matriz de pré-condicionamento*. A eficiência do algoritmo dos gradientes conjugados aplicado agora ao sistema (7.2) é função do

valor de $\kappa(\mathbf{M}^{-1}\mathbf{A})$. Para assegurar uma efectiva redução do número de iterações envolvida na convergência, a escolha da matriz $\mathbf{M}$ deve ser tal que permita obter um valor tão pequeno quanto possível para o número de condição da matriz modificada, $\mathbf{M}^{-1}\mathbf{A}$. Se no limite se considerar $\mathbf{M} = \mathbf{A}$, então $\kappa(\mathbf{M}^{-1}\mathbf{A}) = \kappa(\mathbf{A}^{-1}\mathbf{A}) = \kappa(\mathbf{I}) = 1$, o que faz com que seja necessária apenas uma iteração do algoritmo pré-condicionado para obter a solução do sistema. É evidente que não faz sentido utilizar este pré-condicionador.

São três as condições que devem ser respeitadas quando se pretende obter uma matriz de pré-condicionamento eficaz:

1. O cálculo dos elementos da matriz $\mathbf{M}$ deve ser tão simples quanto possível;
2. A resolução dos sistemas $\mathbf{M}\,\mathbf{z} = \mathbf{r}$ deve poder ser efectuada de uma forma simples e muito eficaz;
3. A matriz $\mathbf{M}$ deve assemelhar-se tanto quanto possível à matriz $\mathbf{A}$.

A selecção da matriz de pré-condicionamento é dificultada pelo conflito existente entre a última das condições, que tem como finalidade assegurar uma redução efectiva do número de iterações e as duas restantes, que exprimem a necessidade de garantir a eficiência global de todo o processo de cálculo.

São de seguida apresentadas as matrizes de pré-condicionamento utilizadas nos algoritmos implementados e testados ao longo deste trabalho. Tratam-se, todos eles de *pré-condicionadores globais*, o que significa que são obtidos considerando a matriz dos coeficientes como um todo. Embora não sejam aqui exploradas, existem outras técnicas que permitem obter as matrizes de condicionamento ao nível elementar, o *pré-condicionamento elementar*, ou ao nível de conjuntos de elementos, o chamado *pré-condicionamento por sub-domínios*. Estes métodos exploram o facto da matriz global ser obtida através da reunião dos diferentes sistemas governativos provenientes de cada elemento ou sub-domínio considerado. A aplicação dos algoritmos iterativos permite que o sistema global nunca seja obtido explicitamente, permitindo desta forma que todos os cálculos possam ser efectuados directamente com base na informação respeitante a cada elemento/sub-domínio.

O método de pré-condicionamento global mais simples consiste em considerar

$$\mathbf{M} = diag(\mathbf{A}) \ .$$

A vantagem do *pré-condicionamento diagonal (CG_D)* reside na facilidade e rapidez com que são determinados os elementos de $\mathbf{M}$ e na eficiência com que podem ser resolvidos os sistemas da forma $\mathbf{M}\,\mathbf{z} = \mathbf{r}$. No entanto, este método não é o que permite obter a melhor taxa de aceleração do processo de convergência.

Uma técnica de pré-condicionamento mais interessante é a que se baseia na aplicação do algoritmo *CG_SSOR* (*symmetric successive over relaxation*). A matriz de

condicionamento correspondente é definida através da igualdade:

$$\mathbf{M} = \frac{1}{w(2-w)}(\mathbf{D} + w\mathbf{L})\mathbf{D}^{-1}(\mathbf{D} + w\mathbf{L}^t) \ ,$$

onde se considera a decomposição $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{L}^t$. A aplicação deste método requer a utilização do parâmetro de relaxação w, que pode tomar um qualquer valor pertencente ao intervalo $]0, 2[$. Para cada situação particular, existe um valor óptimo para o valor de w, valor esse que permite minimizar o número de iterações a efectuar. Infelizmente, a determinação *a priori* de tal valor óptimo não é em geral possível.

O método de pré-condicionamento mais utilizado baseia-se na realização de uma *factorização incompleta* da matriz dos coeficientes. A ideia subjacente a este algoritmo consiste em obter a matriz $\mathbf{M}$ através da factorização de **A,** mas em que se ignoram alguns ou mesmo todos os coeficientes não-nulos que surgem durante o processo de eliminação como resultado do *fill-in*. Quando se ignora por completo o *fill-in*, a matriz de pré-condicionamento pode ser escrita na forma

$$\mathbf{M} = \mathbf{U}^t\mathbf{D}\mathbf{U} \ ,$$

onde $\mathbf{U}$ apresenta exactamente a mesma esparsidade e a mesma distribuição de coeficientes não-nulos que o triângulo superior da matriz $\mathbf{A}$.

Apresenta-se agora o algoritmo que permite aplicar o método dos gradientes conjugados na resolução iterativa de sistemas de equações lineares.

Inicializar

$$\begin{aligned}
&\mathbf{x}_0 = \overline{\mathbf{x}} \\
&\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0 \\
&\mathbf{z}_0 = \mathbf{M}^{-1}\mathbf{r}_0 \\
&\mathbf{p}_0 = \mathbf{z}_0
\end{aligned}$$

Para cada $i = 0, 1, \ldots$

$$\begin{aligned}
&\mathbf{u}_i = \mathbf{A}\,\mathbf{p}_i \\
&\alpha_i = \frac{\mathbf{r}_i^t\,\mathbf{z}_i}{\mathbf{p}_i^t\,\mathbf{u}_i} \\
&\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i\,\mathbf{p}_i \\
&\mathbf{r}_{i+1} = \mathbf{r}_i - \alpha_i\,\mathbf{u}_i \\
&\varepsilon = \frac{\|\mathbf{r}_{i+1}\|}{\|\mathbf{b}\|} \qquad \text{Se } \varepsilon < tol, \text{ então acabar} \\
&\mathbf{z}_{i+1} = \mathbf{M}^{-1}\mathbf{r}_{i+1} \\
&\beta_i = \frac{\mathbf{r}_{i+1}^t\,\mathbf{z}_{i+1}}{\mathbf{r}_i^t\,\mathbf{z}_i} \\
&\mathbf{p}_{i+1} = \mathbf{z}_{i+1} + \beta_i\,\mathbf{p}_i
\end{aligned}$$

fim do passo i

No que diz respeito ao volume de cálculos a efectuar em cada passo do processo iterativo, as operações condicionantes consistem na realização da multiplicação

matriz/vector, $\mathbf{A}\,\mathbf{p}_i$ e na resolução do sistema $\mathbf{M}\,\mathbf{z}_{i+1} = \mathbf{r}_{i+1}$. Para além destas, e sem contar com algumas operações envolvendo escalares e actualização de vectores, realizam-se três produtos internos envolvendo vectores de dimensão N.

O algoritmo apresentado permite confirmar que a quantidade de memória necessária é substancialmente inferior à que é requerida pela aplicação de um qualquer método directo. Para além dos coeficientes não-nulos da matriz inicial e dos elementos da matriz de pré-condicionamento (ou os seus factores), há apenas que armazenar 5 vectores de dimensão N, para além de um número reduzido de escalares.

O critério de paragem aqui seleccionado utiliza directamente o valor da norma do resíduo, $\|\,\mathbf{r}_i\|$. Embora tal não seja aqui considerado, existe a possibilidade de basear tal critério no valor dos coeficientes α_i e β_i [125].

Convém relembrar que a convergência do algoritmo dos gradientes conjugados é garantida de um ponto de vista teórico apenas quando a matriz do sistema é positiva definida. Quando tal não sucede, a utilização do método continua a ser possível, mas a sua aplicação requer alguns cuidados; o valor do produto triplo $\mathbf{p}_i^t\mathbf{A}\mathbf{p}_i$ pode ser nulo, mesmo que os vectores $\mathbf{p}_i$ contenham elementos não-nulos. Esta situação pode provocar o *cancelamento* imediato do processo de convergência. Desde já se adianta no entanto que nos testes efectuados envolvendo a aplicação do algoritmo dos gradientes conjugados na resolução dos sistemas indefinidos que resultam da aplicação dos modelos apresentados, foi sempre garantida a convergência, tendo sido possível evitar a situação anteriormente descrita.

Outro dos problemas que pode resultar do facto de se utilizarem gradientes conjugados na resolução de sistemas indefinidos tem a ver com o *arrastar* do processo de convergência. Para tentar contornar este problema, devem utilizar-se pré-condicionadores eficazes ou então utilizar outros algoritmos iterativos, tais como o método de Lanczos ou o algoritmo *MRES*.

7.5.2 Método de Lanczos

O algoritmo de Lanczos surgiu inicialmente como um método iterativo para a resolução de problemas de valores e vectores próprios. Mais tarde, a sua utilização foi alargada ao âmbito da resolução de sistemas de equações lineares. A sua utilização neste campo permite resolver sistemas com múltiplos termos independentes de uma forma mais eficiente do que o método dos gradientes conjugados. Em certos casos permite também o tratamento mais eficaz de problemas em que a matriz dos coeficientes é indefinida [146].

O algoritmo de Lanczos pode ser descrito como sendo o processo de construção de um conjunto de vectores ortonormais, $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_i$, através da aplicação do processo de ortonormalização de Gram-Schmidt [103] ao conjunto de vectores que formam o subespaço de Krylov. Estes últimos são definidos pela sequência $\mathbf{r}_0, \mathbf{A}\,\mathbf{r}_0, \ldots, \mathbf{A}^{i-1}\mathbf{r}_0$,

na qual $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$.

A equação fundamental do método de Lanczos é aquela que permite obter a sequência de vectores ortonormais e que pode ser escrita na forma,

$$\beta_{i+1}\,\mathbf{q}_{i+1} = \widehat{\mathbf{q}}_{i+1} = \mathbf{A}\,\mathbf{q}_i - \alpha_i\,\mathbf{q}_i - \beta_i\,\mathbf{q}_{i-1}\ , \tag{7.8}$$

onde

$$\alpha_i = \mathbf{q}_i^t\,\mathbf{A}\,\mathbf{q}_i\ ;\ \beta_i = \|\,\mathbf{q}_i\|\ .$$

Uma das propriedades básicas deste algoritmo consiste no facto da projecção da matriz $\mathbf{A}$ no sub-espaço definido pelos vectores $\mathbf{q}_i$ ser uma matriz tridiagonal. Pode definir-se:

$$\mathbf{T}_i = \mathbf{Q}_i^t\,\mathbf{A}\,\mathbf{Q}_i = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & 0 \\ \beta_2 & \alpha_2 & \ddots & 0 \\ 0 & \ddots & \ddots & \beta_i \\ 0 & 0 & \beta_i & \alpha_i \end{bmatrix}\ ,$$

onde $\mathbf{Q}$ é uma matriz cujas colunas correspondem aos diferentes vectores $\mathbf{q}_i$,

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \cdots & \mathbf{q}_i \end{bmatrix}\ .$$

As relações existentes entre as grandezas envolvidas na aplicação do algoritmo de Lanczos podem ser resumidas nas seguintes equações:

$$\mathbf{Q}_i^t\mathbf{Q}_i = \mathbf{I}\ ,$$

$$\mathbf{A}\,\mathbf{Q}_i - \mathbf{Q}_i\,\mathbf{T} = \widehat{\mathbf{q}}_{i+1}\mathbf{e}_i^t\ ,$$

$$\mathbf{Q}_i^t\mathbf{r}_i = \mathbf{0}\ ,$$

onde $\mathbf{e}_i$ representa a coluna i da matriz de identidade, $\mathbf{I}$.

Em cada passo do processo iterativo, a solução aproximada, $\mathbf{x}_i$, pode ser obtida a partir da igualdade:

$$\mathbf{x}_i = \mathbf{Q}_i\mathbf{y}_i\ , \tag{7.9}$$

onde $\mathbf{y}_i$ representa a solução do sistema (7.2), *projectado* no espaço de Krylov definido pelos vectores $\mathbf{q}_i$;

$$\mathbf{Q}_i^t\,\mathbf{A}\,\mathbf{Q}_i\mathbf{y}_i = \mathbf{Q}_i^t\,\mathbf{b}\ , \tag{7.10}$$

Em cada iteração é possível obter o valor da norma do resíduo, $\|\,\mathbf{r}\|$, sem que para tal seja necessário efectuar o seu cálculo explícito através da aplicação da definição (7.7). Por outro lado, o algoritmo de Lanczos não necessita que seja calculada a aproximação da solução $\mathbf{x}_i$ em cada passo. Desta forma, durante o processo iterativo devem ser armazenados apenas os valores dos coeficientes presentes na matriz $\mathbf{T}$ e os vectores ortogonais $\mathbf{q}_i$. Para minimizar o volume de memória necessária e como

a aplicação da definição básica (7.8) requer apenas o conhecimento dos vectores calculados nas duas iterações anteriores, os restantes podem e devem ser armazenados em ficheiro.

No final do processo iterativo recuperam-se todos os vectores $\mathbf{q}_i$ previamente armazenados e resolve-se o sistema (7.10). Sendo $\mathbf{T}_i$ uma matriz tridiagonal, a resolução de tal sistema pode ser efectuada de uma forma muito eficiente [169]. Finalmente, a solução de (7.2) pode ser obtida através da igualdade (7.9).

Pode encontrar-se escrito em [139] o algoritmo que permite aplicar o método de Lanczos na resolução de sistemas lineares. No entanto, esse algoritmo pode ser modificado por forma a permitir que este método possa ser descrito como um algoritmo iterativo similar ao dos gradientes conjugados. A forma não pré-condicionada desse algoritmo, proposta por Paige e Saunders [143], encontra-se apresentada no apêndice E.

Os métodos de Lanczos e dos gradientes conjugados encontram-se intimamente relacionados. É mesmo possível obter o algoritmo dos gradientes conjugados tendo como ponto de partida o método de Lanczos [139]. Em aritmética exacta, ambos produzem soluções aproximadas idênticas em cada passo do processo iterativo [146]. É também possível relacionar directamente o valor dos coeficientes α_i e β_i presentes no método dos gradientes conjugados com os elementos constituintes da matriz tridiagonal $\mathbf{T}_i$ [146].

As diferenças entre estes dois algoritmos surgem naturalmente quando os cálculos são efectuados com *precisão finita*. Quando a matriz dos coeficientes do sistema a resolver é indefinida, o método de Lanczos permite regra geral obter a solução com um número inferior de iterações.

7.5.3 Método *MRES*

O algoritmo *MRES*, conhecido habitualmente na literatura como *method of minimum residuals*, foi desenvolvido para permitir especificamente a resolução iterativa de sistemas de equações indefinidos.

O algoritmo é muito semelhante ao dos gradientes conjugados, vindo apenas alterada a forma através da qual são calculados os coeficientes α_i e β_i. Se se garantir a minimização de $E_{i^*} = \mathbf{r}_i^t\,\mathbf{r}_i$ obtém-se, sem pré-condicionamento:

$$\alpha_i = \frac{\mathbf{r}_i^t \mathbf{A}\,\mathbf{r}_i}{\mathbf{u}_i^t\,\mathbf{u}_i}\ ;$$

$$\beta_i = \frac{\mathbf{r}_{i+1}^t \mathbf{A}\,\mathbf{r}_{i+1}}{\mathbf{r}_i^t \mathbf{A}\,\mathbf{r}_i}\ ;$$

Em relação aos gradientes conjugados, este algoritmo apresenta à partida a desvantagem de exigir em cada iteração a realização de duas multiplicações de matriz por

vector.

7.6 Exemplos de aplicação

Para ilustrar a aplicação dos diferentes algoritmos aqui apresentados e para aferir a eficiência e o mérito relativo de cada um deles, utiliza-se de novo o exemplo definido no capítulo anterior. Na análise da consola quadrada representada na figura 6.5 apenas se utilizam, para este efeito, aproximações baseadas na utilização de séries de Walsh, por serem as que conduzem a sistemas governativos de maiores dimensões. Utilizam-se nas análises as três malhas de elementos finitos representadas na figura 6.6, sendo sempre utilizado o modelo híbrido-misto de equilíbrio.

7.6.1 Métodos directos

Forma não-condensada do sistema governativo

Considere-se a malha A representada na figura 6.6. Na tabela 7.1 apresenta-se o tempo (em segundos) gasto na resolução directa da forma não-condensada do sistema governativo quando se utiliza o algoritmo apresentado por Pissanetzky. Indica-se ainda, e para cada valor do parâmetro p, o número de graus de liberdade envolvidos na discretização, n_{gl}, o número de coeficientes não-nulos existentes na matriz do sistema, $n_{nz}(A)$, e ainda o número de coeficientes não-nulos existentes na matriz que resulta da aplicação do processo de eliminação, $n_{nz}(U)$.

p	2	3	4	5	6
n_{gl}	74	266	986	3770	14714
$n_{nz}(A)$	160	768	3840	20992	128000
$n_{nz}(U)$	398	2732	22552	231216	2837088
$n_{nz}(U)/n_{nz}(A)$	2.48	3.56	5.87	11.01	22.16
$T(s)$	0.01	0.02	0.26	9.68	427.91

Tabela 7.1: *Solução directa do sistema não condensado; malha A.*

Ressalta da análise da tabela 7.1 o rápido crescimento do tempo de resolução à medida que aumenta o valor de p. Este facto pode ser explicado atendendo ao elevado *fill-in* que ocorre durante o processo de eliminação. Basta observar a diferença existente entre o número de coeficientes não-nulos na matriz inicial e na matriz factorizada para se perceber o porquê de tão grande degradação existente em termos de tempo de cálculo.

A mesma tabela permite verificar que o peso relativo do *fill-in* vai aumentando com o valor de p. O quociente $n_{nz}(U)/n_{nz}(A)$ toma valores sempre crescentes para os

sucessivos valores do parâmetro p. Este facto faz prever que o *fill-in* condicione ou mesmo impossibilite a utilização deste algoritmo quando se utilizam nas discretização valores de p a partir de um certo limite.

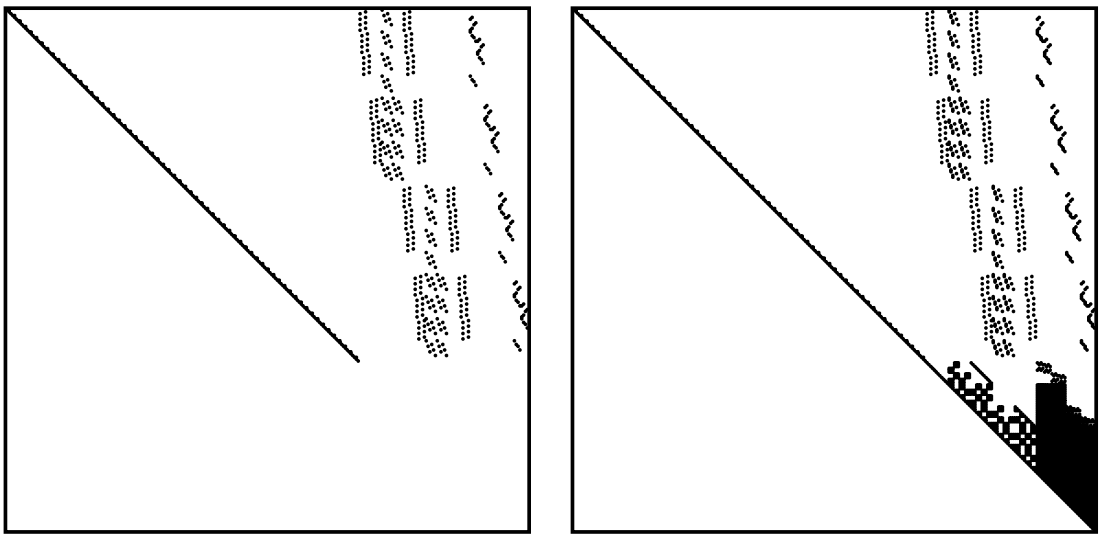

Figura 7.1: *Ilustração da ocorrência de fill-in.*

A ocorrência de *fill-in* é ilustrada na figura 7.1, onde se representa a distribuição dos coeficientes não-nulos, antes e depois da execução do processo de eliminação de Gauss. A matriz dos coeficientes assim representada corresponde à discretização com $p = 3$.

Apresentam-se, na tabela 7.2, os tempos de resolução obtidos quando se utilizam as rotinas MA47 na resolução dos sistemas a que se refere a tabela 7.1. É notória a diminuição do tempo de cálculo para os casos com valores de p iguais a 5 e a 6. Esta significativa melhoria do tempo de cálculo resulta fundamentalmente da aplicação de um algoritmo de minimização de *fill-in*. Repare-se no entanto que os tempos de resolução para os restantes valores do parâmetro p são ligeiramente superiores aos que foram obtidos com a utilização do algoritmo de Pissanetsky. Isto acontece porque para além do *fill-in* não ser condicionante em qualquer um destes casos (ver tabela 7.1), o tempo dispendido na aplicação do algoritmo de reordenação da matriz dos coeficientes representa uma parcela significativa do tempo total de resolução, que é nestes casos extremamente reduzido, sempre inferior a um segundo.

p	2	3	4	5	6
n_{gl}	74	266	986	3770	14714
$T(s)$	0.01	0.05	0.39	4.95	119.75

Tabela 7.2: *Solução directa do sistema não-condensado com MA47; malha A.*

Nas tabelas 7.3 e 7.4 apresentam-se os tempos de resolução dos sistemas governativos

associados à utilização das malhas B e C, respectivamente.

p	2	3	4	5	6
n_{gl}	284	1036	3884	14956	58604
$T(s)$	0.06	0.34	2.37	25.65	568.98

Tabela 7.3: *Solução directa do sistema não-condensado com MA47; malha B.*

p	2	3	4	5	6
n_{gl}	1112	4088	15416	59576	-
$T(s)$	0.28	1.78	13.56	138.90	-

Tabela 7.4: *Solução directa do sistema não-condensado com MA47; malha C.*

Estas tabelas permitem confirmar a existência de um salto acentuado nos tempos de cálculo sempre que se passa de discretizações com $p = 5$ para discretizações com $p = 6$. Verifica-se por outro lado que para discretizações envolvendo aproximadamente o mesmo número de graus de liberdade, a que está associada a um tempo de resolução mais baixo é aquela a que corresponde o menor valor do parâmetro p. Tomem-se, como exemplo, as discretizações que possuem cerca de 15000 graus de liberdade. A tabela 7.5 permite identificar para cada uma delas a malha, o valor de p e o tempo de resolução correspondente.

malha	A	B	C
n_{gl}	14714	14956	15416
p	6	5	4
$T(s)$	119.75	25.65	13.56

Tabela 7.5: *Tempos de resolução associados a discretizações com aproximadamente o mesmo número de graus de liberdade.*

Estes factos podem condicionar a estratégia a seguir quando, utilizando métodos directos de resolução do sistema governativo, se pretende refinar a solução obtida a partir de uma dada discretização. O procedimento p- adaptativo pode deixar de ser eficiente, de um ponto de vista estritamente numérico, a partir de um determinado valor de p. Será então mais vantajoso utilizar um procedimento h- hierárquico, sendo no entanto importante estabelecer um ponto de equilíbrio; se a utilização de um valor de p muito elevado faz degradar significativamente a eficiência do cálculo, a consideração de um número exagerado de elementos na malha faz perder algumas das vantagens inerentes à utilização de modelos híbrido-mistos de elementos finitos.

Forma condensada do sistema governativo

Apresentam-se nas tabelas 7.6, 7.7 e 7.8 os tempos de resolução (sempre em segundos) obtidos quando se utiliza um método directo na solução da forma condensada

do sistema governativo, (3.57). O tempo total de cálculo, $T_{tot}(s)$, pode ser dividido em várias parcelas. A primeira, T_{pc}, diz respeito ao tempo necessário para efectuar a factorização da matriz

$$\left[\begin{array}{cc} \mathbf{F} & \mathbf{A}_v \\ \mathbf{A}_v^t & \mathbf{0} \end{array} \right] . \tag{7.11}$$

A segunda parcela, T_{aka}, engloba o tempo dispendido na execução do produto triplo definido em (3.58) e na obtenção dos termos independentes presentes em (3.59). O cálculo deste produto triplo nunca passa pela obtenção explícita da inversa da matriz (7.11). Se tal fosse feito, rapidamente se excederia a capacidade de armazenamento, uma vez que a inversa de uma matriz esparsa é, regra geral, uma matriz cheia. O cálculo da coluna j de

$$\left[\begin{array}{cc} \mathbf{F} & \mathbf{A}_v \\ \mathbf{A}_v^t & \mathbf{0} \end{array} \right]^{-1} \left[\begin{array}{c} \mathbf{A}_\gamma \\ \mathbf{0} \end{array} \right] , \tag{7.12}$$

$\mathbf{c}_j^{kA}$, é efectuado directamente através da resolução do sistema de equações

$$\left[\begin{array}{cc} \mathbf{F} & \mathbf{A}_v \\ \mathbf{A}_v^t & \mathbf{0} \end{array} \right] \mathbf{c}_j^{kA} = col_j \left\{ \left[\begin{array}{c} \mathbf{A}_\gamma \\ \mathbf{0} \end{array} \right] \right\} , \tag{7.13}$$

em que se utiliza a factorização de (7.11) previamente obtida. Em (7.13), $col_j\,\{\}$ denota a coluna j da matriz definida como argumento dessa função.

Finalmente, a terceira parcela, T_{sol}, traduz o tempo necessário para efectuar o espalhamento das diferentes equações elementares e resolver o sistema governativo global.

Na factorização da matriz (7.11) utiliza-se o algoritmo apresentado por Pissanetzky [165]. Esta escolha pode parecer estranha se se tiver em conta os resultados apresentados e discutidos na secção anterior. De facto, se se tivesse que escolher o algoritmo a utilizar nesta fase do cálculo apenas com base na eficiência com que a factorização é realizada, então o escolhido seria o código MA47. Verifica-se no entanto que o algoritmo de Pissanetzky permite efectuar as operações de substituição e retrosubstituição de uma forma mais eficiente que as rotinas MA47. Como no cálculo de cada uma das colunas de (7.12) se torna necessário repetir estas mesmas operações, resulta que o tempo ganho nesta fase permite compensar o maior tempo de factorização. Na resolução do sistema de equações global utiliza-se apenas o *package* MA47.

p	2	3	4	5	6
N	18	42	90	186	378
n_{gl}	74	266	986	3770	14714
$T_{pc}(s)$	< 0.01	< 0.01	0.02	0.41	19.28
$T_{aka}(s)$	0.01	0.06	0.79	12.88	258.0
$T_{sol}(s)$	< 0.01	< 0.01	0.03	0.16	1.22
$T_{tot}(s)$	$\simeq$0.01	$\simeq$0.06	0.84	13.45	278.05

Tabela 7.6: *Resolução directa da forma condensada do sistema; malha A.*

Quando existem na malha grupos de elementos com forma, dimensões e propriedades idênticas, é de esperar que se torne vantajoso utilizar a forma condensada (3.57). Torna-se então possível optimizar o processo de cálculo, uma vez que todos os elementos pertencentes a um dado grupo partilham a mesma matriz de rigidez elementar, definida em (3.58). Deste ponto de vista, a situação ideal corresponde à utilização de uma malha constituída por elementos todos idênticos.

A utilização da forma condensada permite por outro lado atenuar o impacto negativo do *fill-in*. Quando se factoriza a matriz (7.11), definida para um dado *elemento tipo*, o número de coeficientes não nulos que surgem durante o processo de eliminação é muito inferior ao que se obteria se se considerasse a matriz do sistema governativo global escrito na forma não condensada. Esta diminuição do *fill-in* será tanto mais significativa quanto menor for o número de grupos de elementos utilizados.

A análise da tabela 7.6 permite verificar que a resolução directa da forma condensada do sistema governativo não é compensadora quando existe apenas um elemento na malha de elementos finitos utilizada na análise. Para comprovar esta afirmação, basta comparar os tempos totais acima listados com os que estão associados à utilização do programa MA47 na resolução directa da forma não-condensada do sistema governativo (ver tabela 7.2). Isto acontece porque o que condiciona a eficiência dos algoritmos de resolução directa de sistemas esparsos é o número de coeficientes não-nulos e a sua *distribuição* na matriz dos coeficientes. Em consequência, e tendo em conta que quando apenas existe um elemento não há a possibilidade de se acelerar o processo de cálculo através da reutilização de valores e operadores já obtidos, a forma condensada está associada a tempos de execução superiores, ainda que a matriz dos coeficientes seja nestes casos de muito menores dimensões.

p	2	3	4	5	6
N	60	140	300	620	1260
n_{gl}	284	1036	3884	14956	58604
$T_{sol}(s)$	0.01	0.04	0.21	1.12	8.64
$T_{tot}(s)$	0.02	0.10	1.02	14.41	285.92

Tabela 7.7: *Resolução directa da forma condensada do sistema; malha B.*

p	2	3	4	5	6
N	216	504	1080	2232	4536
n_{gl}	1112	4088	15416	59576	233912
$T_{sol}(s)$	0.08	0.27	1.33	9.19	73.79
$T_{tot}(s)$	0.09	0.33	2.14	22.48	351.07

Tabela 7.8: *Resolução directa da forma condensada do sistema; malha C.*

Nas tabelas 7.7 e 7.8 apresentam-se os tempos de resolução correspondentes à utilização das malhas B e C. Apenas se listam os tempos totais e os tempos associados à solução do sistema governativo global, $T_{tot}(s)$ e $T_{sol}(s)$, respectivamente. O tempo

necessário para realizar as operações de factorização, $T_{pc}(s)$, é semelhante ao que foi obtido no caso da malha A, uma vez que os elementos existentes nas malhas B e C são todos iguais. Idênticos são também os tempos envolvidos no cálculo do produto triplo definido em (3.58). Da análise destas duas tabelas verifica-se que em qualquer uma das situações se registam tempos de cálculo inferiores aos que tinham sido obtidos aquando da resolução directa da forma não-condensada do sistema governativo (comparar com os valores listados nas tabelas 7.3 e 7.4). Tal como seria de esperar, observa-se que a redução do tempo de cálculo é tão mais significativa quanto maior for o número de elementos iguais. Esta afirmação é ilustrada através do gráfico de barras apresentado na figura 7.2, no qual se comparam os tempos de execução associados à resolução directa das formas não-condensada e condensada dos sistemas governativos globais.

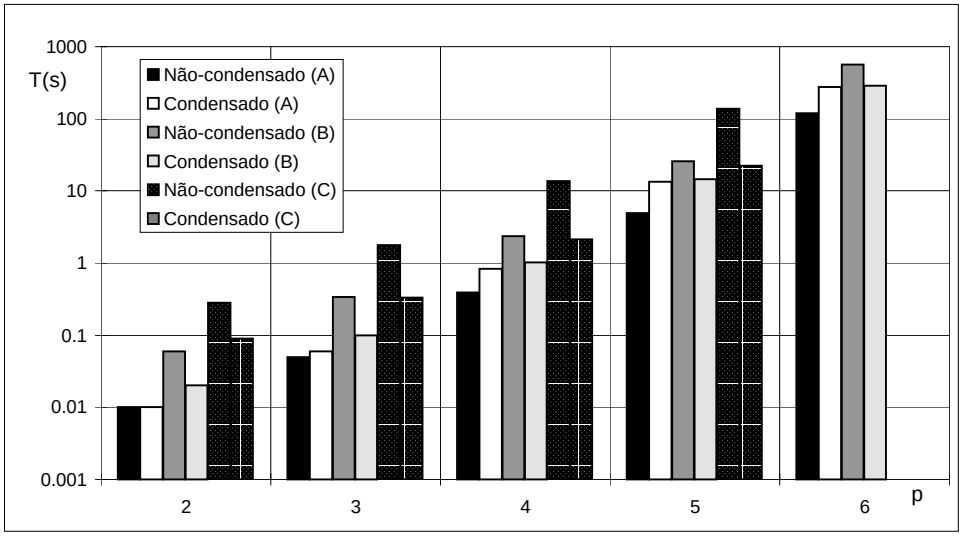

Figura 7.2: *Comparação da resolução directa das formas condensada e não-condensada do sistema governativo.*

Embora tal assunto não seja abordado neste trabalho, é interessante notar a facilidade com que o algoritmo apresentado nesta secção pode ser *paralelizado*. Cada processador pode efectuar, de forma completamente independente, o estabelecimento das equações elementares para cada grupo de elementos com propriedades idênticas. Uma vez reunidas as contribuições de cada elemento e resolvido o sistema governativo global, as operações de pós-processamento podem ser realizadas em paralelo, uma vez que as aproximações dos campos de tensões e de deslocamentos no domínio são completamente independentes de elemento para elemento.

7.6.2 Métodos iterativos

Pretende-se agora aferir a eficiência numérica dos algoritmos iterativos acima descritos. Numa primeira etapa são comparados os desempenhos dos algoritmos baseados

no método dos gradientes conjugados na resolução da forma não condensada do sistema governativo. Analisa-se depois o que sucede quando se normaliza a matriz dos coeficientes do sistema de equações por forma a que os elementos da diagonal passem a apresentar um valor unitário. Numa terceira fase é comparada a eficiência do método dos gradientes conjugados com a do método de Lanczos e a do algoritmo *MRES*. Para finalizar, resolve-se iterativamente a forma condensada do sistema governativo escrito na forma (3.54).

Solução da forma não-condensada do sistema governativo

Encontram-se listados na tabela 7.9 os valores correspondentes ao tempo de resolução dos sistemas de equações que se obtêm quando se utiliza a malha A da figura 6.6 na análise da consola quadrada definida na figura 6.5. Indica-se também para cada caso o número de iterações, N_{iter}, necessário para assegurar a convergência do processo iterativo.

Na coluna *CG* são apresentados os valores que correspondem à utilização do algoritmo dos gradientes conjugados sem qualquer pré-condicionamento. Nas restantes colunas são listados os resultados obtidos com a utilização das diferentes formas de condicionamento atrás descritas.

	p	2	3	4	5	6
	N	74	266	986	3770	14714
CG	N_{iter}	77	220	505	1128	2619
	$T(s)$	0.01	0.07	0.79	10.23	168.08
CG_D	N_{iter}	73	205	443	1000	2267
	$T(s)$	0.01	0.09	0.89	10.86	162.22
CG_SSOR	N_{iter}	46	137	326	710	1786
	$T(s)$	0.01	0.12	1.23	14.19	222.56
CG_IF	N_{iter}	34	57	88	139	229
	$T_{fac}(s)$	0.01	0.01	0.01	0.09	0.75
	$T_{sol}(s)$	0.01	0.03	0.28	2.39	25.02
	$T_{tot}(s)$	0.01	0.03	0.29	2.48	25.77

Tabela 7.9: *Resolução iterativa dos sistemas governativos; malha A.*

No algoritmo *CG_D* a matriz de pré-condicionamento é a matriz diagonal construída com os elementos diagonais da matriz dos coeficientes do sistema governativo (3.53). Sempre que estes sejam nulos, toma-se um valor unitário na posição correspondente da matriz de pré-condicionamento.

Para se conseguir estimar o valor do parâmetro de relaxação a utilizar na aplicação do algoritmo *CG_SSOR*, pode ser efectuado um estudo paramétrico que consiste em testar, para uma dado conjunto de discretizações, uma gama completa de valores de w e seleccionar aquele que conduz à convergência num menor número de iterações.

Apresentam-se na figura 7.3 os gráficos que relacionam, para as quatro discretizações definidas por $p = 2, 3, 4$ e 5, o valor de N_{iter} com o valor do parâmetro w. Verifica-se que de caso para caso o valor mínimo para o número de iterações corresponde a valores ligeiramente diferentes do parâmetro de relaxação. No entanto, observa-se que tal valor mínimo se encontra sempre na vizinhança do ponto $w = 1$, pelo que é esse o valor adoptado nas análises efectuadas.

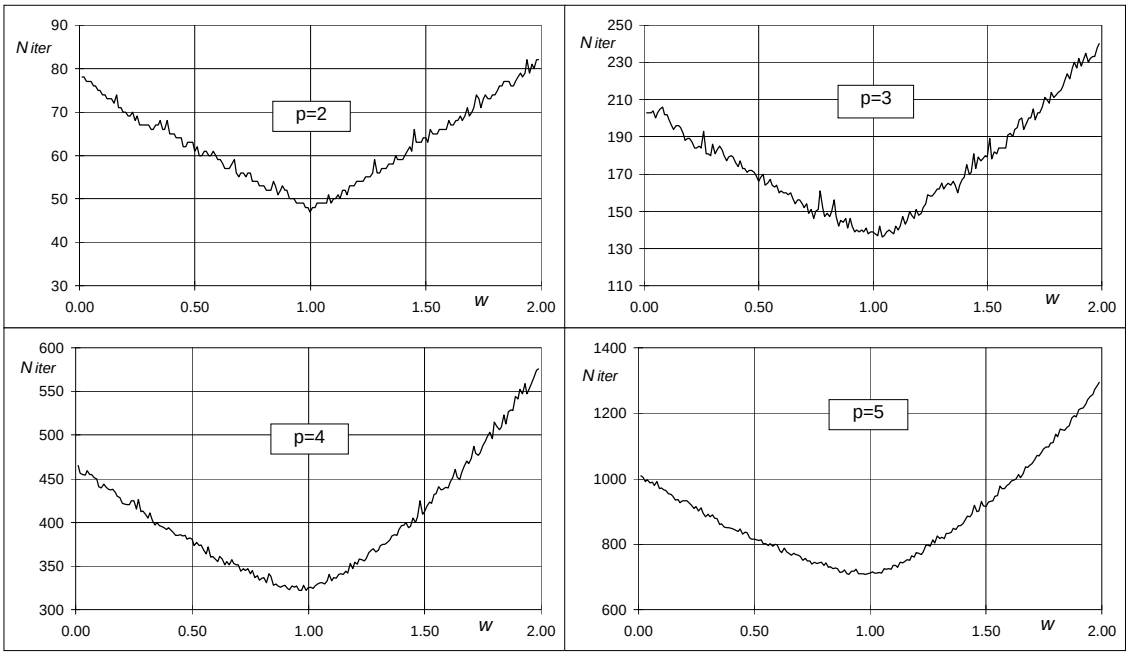

Figura 7.3: *Estimativa do valor óptimo para o parâmetro w.*

Os tempos de cálculo associados à aplicação do algoritmo *CG_IF*, podem ser decompostos em duas parcelas. A primeira, $T_{fac}(s)$, corresponde ao tempo dispendido no processo de factorização incompleta da matriz dos coeficientes. A segunda parcela, $T_{sol}(s)$ dá conta do tempo gasto na aplicação do algoritmo dos gradientes conjugados.

Na factorização incompleta que aqui é efectuada, e com a excepção dos elementos da diagonal principal, são ignorados todos os coeficientes não-nulos resultantes do fenómeno de *fill-in*. No cálculo da matriz de pré-condicionamento é utilizado o algoritmo de Pissanetzky. Como se pretende ignorar o *fill-in*, a etapa correspondente à *factorização simbólica* não é efectuada. Como se pode confirmar pela análise da tabela 7.9, a factorização incompleta é realizada sempre de uma forma bastante rápida.

O critério de convergência utilizado nestes exemplos foi sempre o mesmo. Considera-se que o processo iterativo deve terminar quando se verifica a desigualdade:

$$\frac{\| \mathbf{r} \|}{\| \mathbf{b} \|} < 10^{-6} \ .$$

Como solução inicial considera-se sempre $\mathbf{x}_0 = \mathbf{b}$. Esta não é geralmente a solução adoptada na generalidade das aplicações. No entanto é a que conduz nestas situações ao melhor desempenho desses algoritmos, traduzido pelo menor número de iterações efectuadas.

A análise dos valores apresentados na tabela 7.9 permite tecer alguns comentários. O método mais eficiente, quer se comparem tempos de execução ou número de iterações efectuadas, é o que utiliza a factorização incompleta na obtenção da matriz de pré-condicionamento.

Se se comparar apenas o número de iterações, verifica-se que existe em qualquer um dos casos considerados uma diminuição gradual e consistente à medida que se utilizam pré-condicionadores mais sofisticados. Este facto é confirmado pelo gráficos de barras apresentado na figura 7.4 onde se representa, para os mesmos casos de teste, o número de iterações necessárias para assegurar a convergência de cada um dos algoritmos em análise.

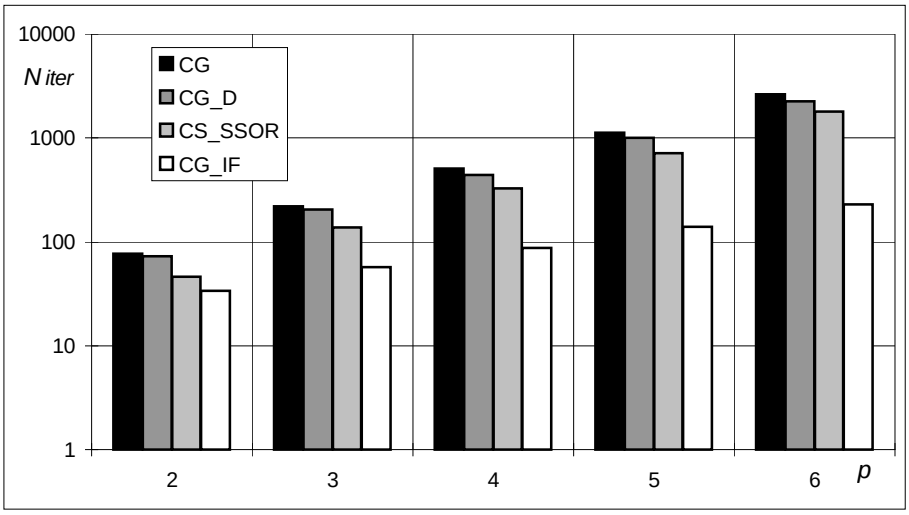

Figura 7.4: *Comparação do número de iterações necessário para garantir a convergência.*

Se se compararem os tempos de execução, verifica-se que nem sempre uma diminuição do número de iterações corresponde a uma diminuição do tempo dispendido nos cálculos. Observa-se desta forma que os algoritmos *CG_D* e *CG_SSOR*, embora necessitem de menos iterações para alcançar a solução do problema que o algoritmo *CG*, apresentam tempos de execução semelhantes ou mesmo superiores aos obtidos por este último. Isto acontece porque o número de operações a efectuar em cada passo do processo iterativo é superior sempre que se considera pré-condicionamento. Para que haja uma melhoria efectiva em termos de tempo global de solução, torna-se necessário que a redução do número de iterações seja tal que permita compensar o acréscimo de tempo dispendido em cada passo.

Analisando os valores da tabela 7.9, observa-se que os tempos de solução associados ao algoritmo *CG_D* são sempre ligeiramente superiores aos obtidos utilizando o algoritmo *CG*. Apenas para o caso em que $p = 6$ se obtém um tempo de solução menor. Já o método *CG_SSOR* conduz sempre a tempos de execução superiores aos obtidos com os algoritmos *CG* e *CG_D*. Estas afirmações podem ser confirmadas pela análise do gráfico apresentado na figura 7.5, onde se traça a evolução dos tempos de execução para todos os casos testados.

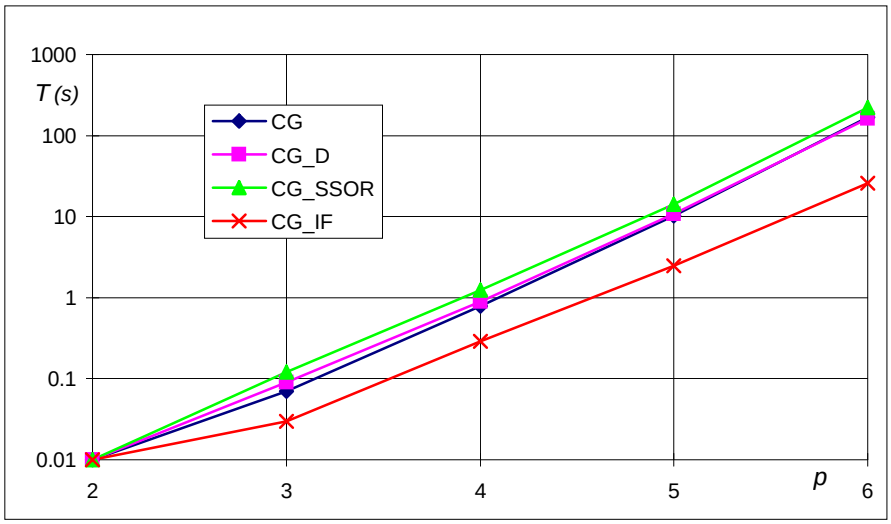

Figura 7.5: Comparação dos tempos de solução dos algoritmos iterativos.

Para os casos testados, verifica-se que foi possível assegurar a convergência com um número de iterações sempre inferior à dimensão do sistema, com a excepção de um único caso (algoritmo *CG* com $p = 2$). É por outro lado importante observar que o valor do quociente N_{iter}/N vai diminuindo de uma forma significativa à medida que se aumenta o valor do parâmetro p.

Não sendo positiva definida a matriz dos coeficientes, é de esperar que a convergência do método iterativo seja *irregular*. Este facto é ilustrado pelos gráficos da figura 7.6, nos quais se representa a evolução da norma do resíduo, $\|\mathbf{r}\|$, ao longo do processo iterativo, na análise da consola quadrada considerando $p = 3$. É bem visível a forte oscilação presente ao longo de todo o processo de cálculo. Já o algoritmo *CG_IF* conduz a uma convergência ligeiramente mais *suave*, ainda que não isenta de perturbações.

É importante agora comparar os tempos de execução obtidos com o algoritmo *CG_IF* e os tempos obtidos com o método directo de resolução, baseado na utilização do *package MA47*. A análise comparativa das tabelas 7.9 e 7.2 permite verificar que o método iterativo conduz sempre a tempos de execução inferiores. Observa-se que a redução do tempo de cálculo verificada é tanto maior quanto maior é o valor de p. Para a discretização que utiliza $p = 6$, o método directo necessita de 119.75 segundos

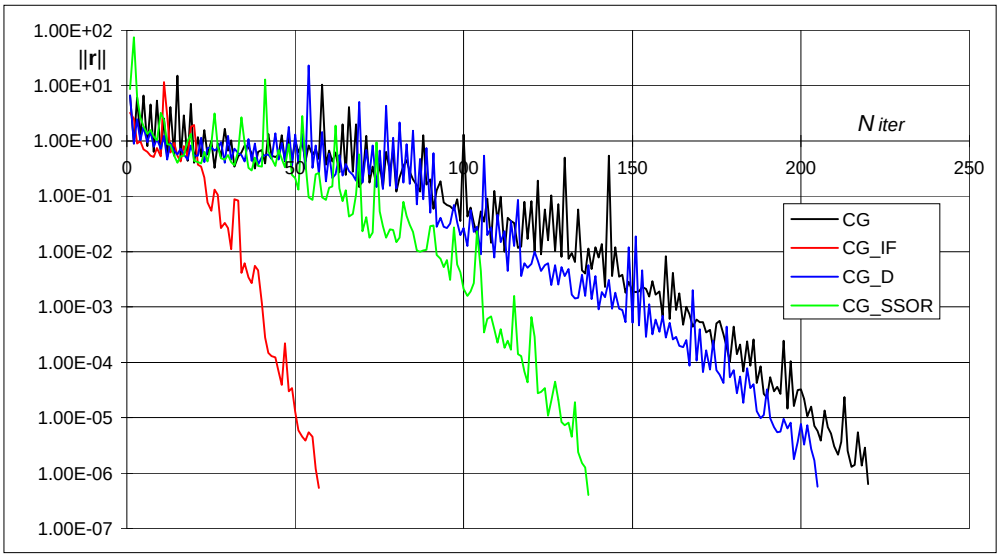

Figura 7.6: *Evolução do valor da norma do resíduo; gradientes conjugados.*

para fornecer a solução, enquanto que o algoritmo CG_IF o consegue fazer em 25.77 segundos. Este facto está associado ao aumento do tempo de cálculo que se verifica no método directo em consequência da existência do fenómeno de *fill-in*.

As tabelas 7.10, e 7.11 contêm o mesmo tipo de informação que a tabela 7.9, mas agora para as discretizações envolvendo a utilização das malhas B e C, respectivamente. No caso da malha C, não foi possível obter o valor do tempo de execução e o número de iterações associados à utilização dos algoritmos *CG*, *CG_D* e *CG_SSOR* para a discretização com p=6.

	p	2	3	4	5	6
	N	284	1036	3884	14956	58604
CG	N_{iter}	231	498	1062	2197	4441
	$T(s)$	0.08	0.72	7.62	84.97	1162.88
CG_D	N_{iter}	203	432	904	1838	3738
	$T(s)$	0.10	0.85	8.31	84.96	1089.02
CG_SSOR	N_{iter}	142	312	658	1325	2724
	$T(s)$	0.11	1.08	10.86	111.69	1371.87
CG_IF	N_{iter}	77	105	160	243	372
	$T_{fac}(s)$	0.01	0.01	0.06	0.38	3.25
	$T_{sol}(s)$	0.05	0.30	2.23	17.95	167.47
	$T_{tot}(s)$	0.05	0.31	2.29	18.33	170.72

Tabela 7.10: *Resolução iterativa dos sistemas governativos; malha B.*

A análise destas tabelas permite confirmar os comentários atrás efectuados. Observa-se por outro lado que os tempos de execução obtidos com o algoritmo *CG_IF*

	p	2	3	4	5	6
	N	1112	4088	15416	59576	233912
CG	N_{iter}	475	962	1901	4628	-
	$T(s)$	0.64	6.21	56.62	741.10	-
CG_D	N_{iter}	419	826	1627	3268	-
	$T(s)$	0.78	6.99	61.12	620.83	-
CG_SSOR	N_{iter}	297	591	1182	2370	-
	$T(s)$	0.98	8.86	79.71	797.07	-
CG_IF	N_{iter}	154	206	302	461	695
	$T_{fac}(s)$	0.01	0.05	0.26	1.60	17.48
	$T_{sol}(s)$	0.41	2.70	17.75	139.49	5163.42
	$T_{tot}(s)$	0.42	2.75	18.01	141.09	5180.90

Tabela 7.11: *Resolução iterativa dos sistemas governativos; malha C.*

são muito competitivos quando comparados com os tempos obtidos com o programa *MA47*. No caso da malha B, os tempos de execução obtidos para *CG_IF* são sempre inferiores aos que foram obtidos pelo aplicação daquele método directo. Já no caso da malha C, tal vantagem não é tão visível. Aí os tempos de execução são aproximadamente iguais, com excepção da discretização com $p = 6$, a qual não foi possível resolver através da utilização das rotinas *MA47*.

No caso das malhas B e C, o método directo que permite obter tempos de solução inferiores é o que se baseia na resolução da forma condensada do sistema governativo. Importa comparar estes tempos com os que foram obtidos com a utilização do algoritmo *CG_IF*. Verifica-se que o método directo é mais eficaz por permitir a minimização do volume de cálculos a efectuar, uma vez que foi considerado na malha apenas um tipo de macro-elemento.

Solução do sistema governativo *normalizado*

Uma das formas existentes para melhorar o *número de condição* da matriz dos coeficientes consiste em *normalizar* o sistema por forma a garantir que os elementos da diagonal principal passem a ter valor unitário [103]. Na tabela 7.12 encontram-se listados os tempos de execução e o número de iterações associados à aplicação dos algoritmos *CG*, *CG_SSOR* e *CG_IF* na resolução dos sistemas desta forma escalados e que resultam da análise da consola quadrada com a malha A.

A análise comparativa das tabelas 7.9 e 7.12 permite registar uma ligeira diminuição do número de iterações e do tempo de execução, sempre que a aplicação dos algoritmos *CG* e *CG_SSOR* é precedida pela *normalização* do sistema de equações. Deixa neste caso de fazer sentido falar em pré-condicionamento diagonal, uma vez que os elementos da diagonal principal da matriz do sistema têm todos eles valores unitários.

		CG		*CG_SSOR*		*CG_IF*			
p	N	N_{iter}	$T(s)$	N_{iter}	$T(s)$	N_{iter}	$T_{fac}(s)$	$T_{sol}(s)$	$T_{tot}(s)$
2	74	72	0.01	43	0.02	34	< 0.01	0.01	0.01
3	266	207	0.07	133	0.09	57	< 0.01	0.03	0.03
4	986	451	0.73	312	0.95	88	0.02	0.28	0.30
5	3770	995	9.13	683	11.33	139	0.09	2.40	2.49
6	14714	2271	147.43	1723	191.30	229	0.76	24.93	25.69

Tabela 7.12: *Resolução iterativa dos sistemas normalizados; malha A.*

Interessante é verificar que a normalização do sistema governativo em nada altera as características de convergência do algoritmo *CG_IF*, que continua a ser o mais eficiente, uma vez que a influência do valor dos elementos da diagonal é *filtrada* durante o processo de factorização incompleta.

Estudo comparativo de algoritmos alternativos

Os resultados até aqui apresentados permitem concluir que quando se utilizam algoritmos iterativos na resolução dos sistemas governativos resultantes da aplicação das formulações híbridas-mistas de elementos finitos não se coloca o problema da não convergência ou da convergência demasiadamente lenta do processo iterativo. No entanto, verifica-se que a convergência é bastante irregular, tal como vem ilustrado na figura 7.6.

Interessante é então verificar o que sucede quando se utiliza o método de Lanczos e o algoritmo *MRES*. De um ponto de vista meramente teórico, estes algoritmos são menos sensíveis ao facto da matriz do sistema não ser uma matriz positiva definida. Comparam-se, na tabela 7.13, os tempos de execução e o número de iterações envolvidos na convergência destes algoritmos alternativos. Listam-se apenas os valores obtidos para a malha A. Em qualquer um dos casos é efectuada a normalização do sistema de equações antes de se aplicar o algoritmo de resolução.

		CG		*Lanczos*		*MRES*	
p	N	N_{iter}	$T(s)$	N_{iter}	$T(s)$	N_{iter}	$T(s)$
2	74	72	0.01	72	0.01	72	0.02
3	266	207	0.07	203	0.11	202	0.11
4	986	451	0.73	439	0.96	417	1.23
5	3770	995	9.13	985	11.28	966	17.09
6	14714	2271	147.43	2248	165.90	2110	271.40

Tabela 7.13: *Utilização de algoritmos alternativos.*

Se se comparar o algoritmo *CG* com o método de Lanczos, verifica-se que este último permite obter uma muito ligeira redução no número de iterações. No entanto, e como

cada passo do método de Lanczos envolve um volume de cálculos superior ao que é efectuado em cada iteração do algoritmo dos gradientes conjugados, o tempo de execução total é também ligeiramente superior. A muito pequena diferença registada em termos de número de iterações era já esperada porque em aritmética exacta os gradientes conjugados e o método de Lanczos apresentam exactamente a mesma taxa de convergência, sendo a solução obtida precisamente no mesmo número de iterações.

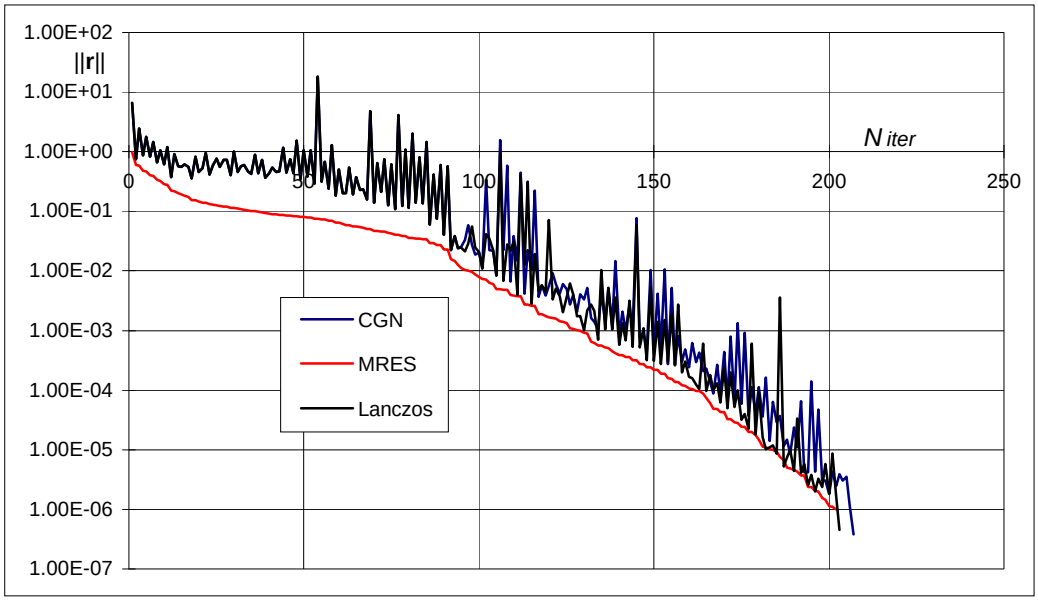

Figura 7.7: *Evolução do valor da norma do resíduo; CG, Lanczos e MRES.*

Na figura 7.7 apresentam-se os gráficos que, para a discretização com $p = 3$, representam a evolução da norma do resíduo ao longo do processo iterativo associado a cada um dos três algoritmos testados. Observa-se de imediato que a convergência do método de Lanczos é caracterizada pelo mesmo comportamento *irregular* já antes notado para o caso dos gradientes conjugados.

De entre estes três algoritmos, é o *MRES* que permite obter a solução com o menor número de iterações. Verifica-se também que a convergência é muito mais *suave* que no caso dos dois outros algoritmos. Basta observar o gráfico correspondente apresentado na figura 7.7, para verificar que a convergência do *MRES* é quase *monotónica*. No entanto, e como a aplicação deste algoritmo envolve duas multiplicações *matriz/vector* em cada passo do processo iterativo, o tempo de resolução total é significativamente superior ao obtido com os outros métodos.

Solução da forma condensada do sistema governativo

Aplica-se agora o método dos gradientes conjugados na solução do sistema governativo escrito na forma condensada apresentada em (3.54).

Uma vez que a matriz $\mathbf{F}$ é diagonal por blocos, a sua inversão é imediata. De acordo com a equação (6.16), é mesmo possível, sob certas condições, calcular directamente e de uma forma muito eficiente os elementos constituintes da matriz $\mathbf{F}^{-1}$. Refira-se por outro lado que nunca é efectuado o cálculo explícito da matriz

$$\mathbf{S} = \begin{bmatrix} \mathbf{A}_v^t & \mathbf{0} \\ -\mathbf{A}_\gamma^t & \mathbf{0} \end{bmatrix} \mathbf{F}^{-1} \begin{bmatrix} \mathbf{A}_v & -\mathbf{A}_\gamma \\ \mathbf{0} & \mathbf{0} \end{bmatrix} ,$$

a qual se costuma designar na literatura por *complemento de Schur* [146]. Quando na aplicação do algoritmo dos gradientes conjugados se pretende efectuar a multiplicação de $\mathbf{S}$ por um dado vector, o cálculo é decomposto numa sequência de três produtos de matriz por vector. Pode escrever-se:

$$\mathbf{y} = \mathbf{S}\mathbf{x} = \begin{bmatrix} \mathbf{A}_v^t & \mathbf{0} \\ -\mathbf{A}_\gamma^t & \mathbf{0} \end{bmatrix} \left\{ \mathbf{F}^{-1} \left(\begin{bmatrix} \mathbf{A}_v & -\mathbf{A}_\gamma \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \mathbf{x} \right) \right\} .$$

Na tabela 7.14 apresentam-se tempos de cálculo e número de iterações efectuadas na resolução iterativa dos sistemas definidos para cada uma das discretizações acima definidas. Os resultados dizem respeito à aplicação do algoritmo dos gradientes conjugados sem a utilização de qualquer pré-condicionamento.

	Malha A			*Malha B*			*Malha C*		
p	N	N_{iter}	$T(s)$	N	N_{iter}	$T(s)$	N	N_{iter}	$T(s)$
2	26	32	< 0.01	92	78	0.03	344	174	0.25
3	74	78	0.03	268	175	0.27	1016	321	2.49
4	218	175	0.32	812	354	3.14	3128	592	21.21
5	698	390	4.28	2668	696	31.73	10424	1160	223.81
6	2426	821	61.07	9452	1311	409.73	37304	2245	2769.09

Tabela 7.14: *Resolução iterativa da forma condensada do sistema de equações sem pré-condicionamento.*

O facto de nunca se calcularem explicitamente os elementos constituintes do complemento de Schur $\mathbf{S}$, faz com que seja agora mais difícil obter um pré-condicionador computacionalmente eficiente. É aqui utilizado um método de pré-condicionamento que pode ser considerado como *quase-diagonal*, pois em vez de se utilizar $\mathbf{M} = \text{diag}(\mathbf{S})$, como seria natural, se considera:

$$\mathbf{M} = diag(\begin{bmatrix} \mathbf{A}_v^t & \mathbf{0} \\ -\mathbf{A}_\gamma^t & \mathbf{0} \end{bmatrix} diag(\mathbf{F}^{-1}) \begin{bmatrix} \mathbf{A}_v & -\mathbf{A}_\gamma \\ \mathbf{0} & \mathbf{0} \end{bmatrix}) .$$

A montagem deste pré-condicionador pode ser efectuada de uma forma bastante rápida e eficiente. Para além disso, a análise da tabela 7.15 permite comprovar

que a sua utilização conduz a uma melhoria muito significativa no que diz respeito à eficiência numérica do processo de cálculo. Observa-se também que de todos os algoritmos iterativos aqui testados é este que conduz à obtenção de tempos de execução inferiores.

	Malha A			*Malha B*			*Malha C*		
p	N	N_{iter}	$T(s)$	N	N_{iter}	$T(s)$	N	N_{iter}	$T(s)$
2	26	25	< 0.01	92	58	0.03	344	115	0.16
3	74	46	0.02	268	80	0.12	1016	166	1.21
4	218	71	0.13	812	125	1.07	3128	226	7.96
5	698	102	1.08	2668	181	8.13	10424	333	61.47
6	2426	154	11.23	9452	253	76.62	37304	487	600.67

Tabela 7.15: *Resolução iterativa da forma condensada do sistema de equações com pré-condicionamento.*

No gráfico da figura 7.8 representa-se a evolução do valor da norma do vector resíduo ao longo do processo iterativo utilizado na resolução do sistema resultante da análise da consola quadrada com a malha A e $p = 3$. A curva *Schur_2* diz respeito à utilização do pré-condicionamento *quase-diagonal*, enquanto que *Schur_1* representa a curva obtida com a aplicação dos gradientes conjugados sem o recurso a qualquer método de pré-condicionamento.

A análise da figura 7.8 permite observar que o processo de convergência é consideravelmente mais *suave* do que no caso da solução iterativa da forma não-condensada do sistema governativo global.

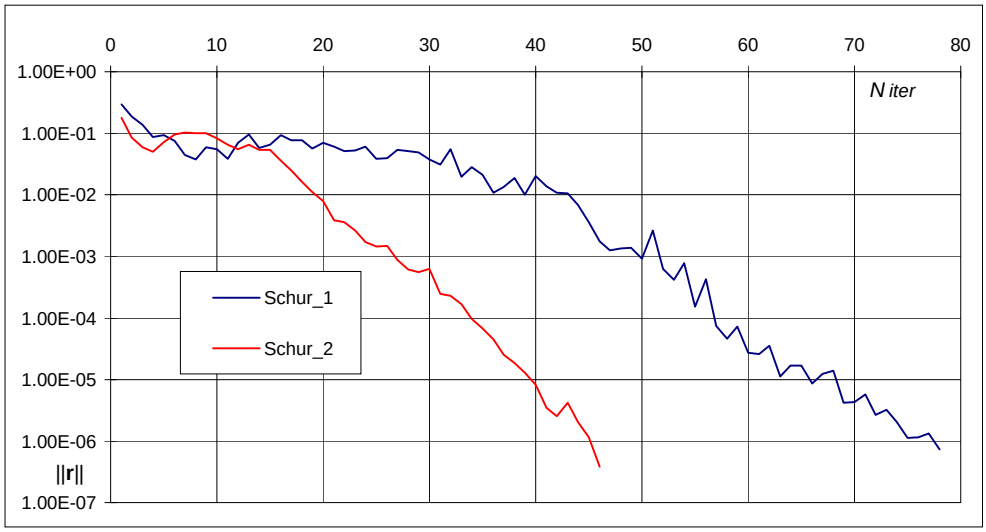

Figura 7.8: *Evolução do valor da norma do resíduo; complemento de Schur.*

Importa agora verificar que os tempos de solução associados a este último algoritmo

são sempre inferiores aos tempos correspondentes obtidos com o código *MA47*. Para o confirmar, basta comparar os valores apresentados nas tabelas 7.15, 7.2, 7.3 e 7.4.

Verifica-se por outro lado que estes tempos de execução são ainda bastante competitivos mesmo quando comparados com os tempos obtidos com o algoritmo baseado na aplicação de métodos directos na resolução da forma condensada do sistema de equações. Comparando a tabela 7.15 com as tabelas 7.7 e 7.8 conclui-se que apenas para o caso em que se consideram 16 elementos iguais é que os gradientes conjugados associados ao pré-condicionamento *quase-diagonal* deixam de ser o método de resolução mais eficiente.

7.7 Comentários finais

Os resultados que aqui foram apresentados permitem confirmar a ideia de que nem sempre é fácil decidir sobre qual o método a utilizar quando se pretende efectuar a resolução de sistemas de equações. Se os métodos directos são atractivos por garantirem a obtenção da solução ao fim de um número finito de operações e permitirem a resolução eficaz de sistemas com múltiplos termos independentes, os métodos iterativos apresentam como grande aliciante a sua simplicidade e a total ausência de fenómenos de *fill-in* e dos problemas a este associados. A decisão ainda se torna mais difícil quando os tempos de execução são da mesma ordem de grandeza, como por vezes sucede.

Os diferentes testes efectuados permitem no entanto extrair algumas indicações, pelo menos no âmbito da resolução dos sistemas de equações que surgem quando da aplicação dos modelos híbrido-mistos de elementos finitos em estudo. Para qualquer outro tipo de sistemas, as conclusões podem resultar substancialmente diferentes.

A análise dos resultados apresentados permite verificar que quando se considera uma malha com apenas um elemento, são os métodos iterativos os que permitem obter a solução com um menor esforço computacional, expresso quer em termos de memória necessária, quer em termos de tempo efectivo de cálculo. De entre os diferentes algoritmos iterativos, provou ser mais eficiente aquele que aplica os gradientes conjugados em conjunto com o pré-condicionamento *quase-diagonal* na resolução da forma condensada (3.54) da matriz do sistema.

Quando se utilizam na análise malhas com vários elementos finitos com a mesma forma e com as mesmas características, são os métodos directos que permitem efectuar a análise de uma forma mais rápida, muito embora sejam mais condicionantes no que diz respeito à quantidade de memória necessária. Também aqui os métodos mais eficientes são aqueles que trabalham com a forma condensada do sistema governativo elementar (3.57).

A vantagem dos métodos directos nesta última situação reside na possibilidade de

se reutilizar a factorização efectuada para cada *elemento tipo* e limitar o *fill-in*. Quando existem na malha muitos elementos com características diferentes, esta vantagem dilui-se consideravelmente. Embora tal não se possa concluir a partir dos exemplos apresentados, é então de prever que os métodos iterativos voltem a ser os mais eficazes.

É importante salientar que os testes e comentários aqui efectuados dizem respeito apenas à resolução de problemas em regime elástico. Em regime elastoplástico efectua-se, em cada passo do processo incremental, a resolução repetida do mesmo sistema de equações onde apenas variam os termos independentes. Para além disto, a possibilidade de se utilizar a pré-condensação definida em (3.82), faz com que sejam utilizados neste trabalho métodos directos de análise para a resolução dos problemas não-lineares.

Finalmente, é importante não esquecer que todos os métodos aqui discutidos foram implementados e testados num ambiente de *processamento sequencial*. Ganhos substanciais podem resultar da *reestruturação* destes algoritmos e sua posterior implementação em ambientes onde possa ser possível o *processamento paralelo*. Dada a sua simplificade e flexibilidade, são os métodos iterativos os que possibilitam uma adaptação mais fácil a este tipo de processamento e são também aqueles que previsivelmente mais beneficiarão da implementação em arquitecturas vectoriais e/ou paralelas.