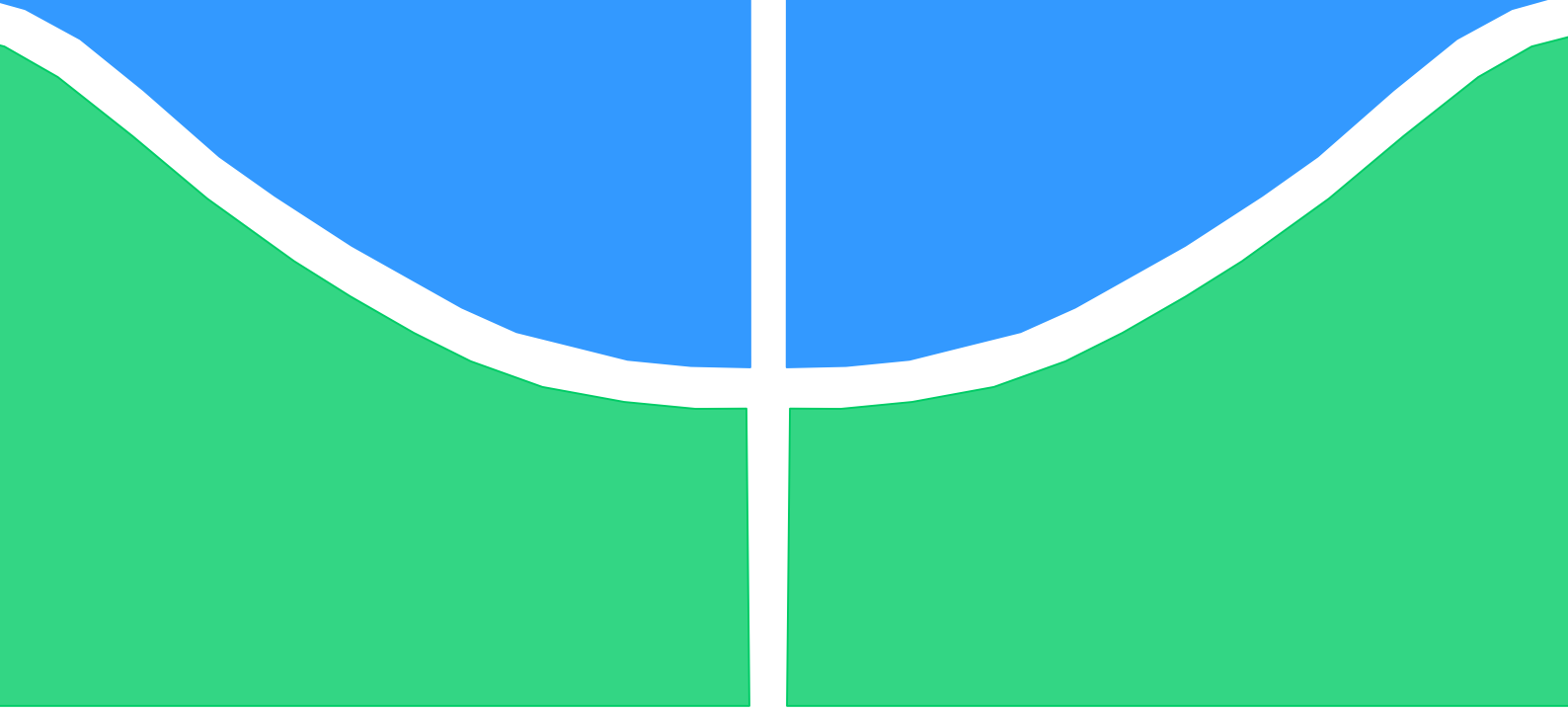

PROJETO DE GRADUAÇÃO

MATRIZES HIERÁRQUICAS E A APROXIMAÇÃO CRUZADA ADAPTATIVA

Por,
Pedro Lucas de A. Morgado

Brasília, 28 de Novembro de 2017.

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Departamento de Engenharia Mecânica

PROJETO DE GRADUAÇÃO

MATRIZES HIERÁRQUICAS E A APROXIMAÇÃO CRUZADA ADAPTATIVA

POR,

Pedro Lucas de A. Morgado

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Mecânico.

Banca Examinadora

Prof. Éder Lima de Albuquerque, UnB/ ENM (Orientador)

Prof. Marcus Vinicius Girão de Morais, UnB/ ENM

Msc. Álvaro Campos Ferreira, UnB/ ENM

Brasília, 28 de Novembro de 2017.

RESUMO

Este trabalho apresenta diferentes ferramentas utilizadas para a implementação de um algoritmo rápido para solucionar problemas de condução de calor bidimensionais, utilizando o Método dos Elementos de Contorno (BEM). O algoritmo rápido inclui a representação da matriz por matrizes hierárquicas, o que envolve a análise de componentes principais (PCA) para a divisão dos dados, a utilização de uma árvore binária e a clusterização hierárquica para organização e agrupamento dos dados, respectivamente, e a aproximação dos blocos de matrizes por matrizes de baixo posto utilizando o método da aproximação cruzada adaptativa (ACA). O algoritmo rápido ainda inclui a resolução do problema matricial $\widetilde{\boldsymbol{H}}x = b$, onde $\widetilde{\boldsymbol{H}}$ é uma matriz Hierárquica, por métodos iterativos. Deste modo, são apresentados e comparados diferentes método iterativos para a solução do problema. É também apresentado uma rotina implementada na linguagem de programação Julia, formado por diversas sub-rotinas, as quais incluem a construção da Matrizes Hierárquica e a aplicação do método ACA para o cálculo de suas componentes. Esse projeto de graduação também inclui uma discussão sobre as limitações e implicações da aplicação do método ACA, assim como inclui uma análise da rotina de programação utilizada. Por fim, é apresentado os resultados obtidos do uso da memória e tempo para a solução de diferentes problemas, em escala, de condução de calor.

Palavras-chaves: ACA. BEM. GMRES. Aproximação Cruzada Adaptativa. Método dos Elementos de Contorno. Matrizes Hierárquicas. Algoritmos rápidos. Análise PCA. Métodos iterativos.

ABSTRACT

This work introduces different tools used to implement a fast algorithm to solve two-dimensional heat conduction problems using the Boundary Element Method (BEM). The fast algorithm represents the matrix using hierarchical matrices, which involve the principal components analysis (PCA) in order to split the data set, the use of a binary tree and a hierarchical clustering to organize and gather the data, respectively, and it approximates the block matrices by low rank matrices utilizing the adaptive cross approximation (ACA). The fast algorithm also includes the solution of the matrix system $\widetilde{\boldsymbol{H}}x = b$, where $\widetilde{\boldsymbol{H}}$ is a data-sparse matrix, by iterative methods. Thus, it introduces and compares different iterative methods to solve the problem. It also presents a routine implemented in the Julia programming language composed by numerous subroutines, which includes the construction of the Hierarchical Matrices and the application of the ACA method in order to obtain its components. This graduation project also includes a discussing about the limitations and implications of using the ACA method, as well as an analysis of the applied routine. Finally, it presents the memory and time usage obtained for solving different (regarding scale) heat conduction problems.

Key words: ACA. BEM. GMRES. Adaptive Cross Approximation. Boundary Element Method. Hierarchical Matrices. Fast algorithms. PCA analysis. Iterative methods.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

CAPÍTULOS 2 e 3

Relacionado a Matrizes	
A	Matriz qualquer.
$\overline{A}(i)$	Aproximação de posto i da matriz A.
$\overline{A}^{*}(i)$	Matriz diferença entre a matriz A e $\overline{A}(i)$.
$A(:,n)$	n-ésima coluna da matriz A**.**
$A(n,:)$	n-ésima linha da matriz A**.**
$\Sigma(\xi)$	Matriz de covariância de ξ.
R^{mxn}	Conjunto das matrizes reais com m linhas e n colunas.
k	Posto da matriz.
$a(i,j)$ *ou* $a_{i,j}$	Elemento da linha i coluna j da matriz A**.**
i^{*}, i^{**} e i^{***}	Linha predefinida por um critério.
j^{*}, j^{**} e j^{***}	Coluna predefinida por um critério.
$p(i)$	Pivô da i-ésima iteração dos métodos ACA, ACA+.
$ps(i)$	Pivô da i-ésima iteração dos métodos ACA, ACA+.
$px(i)$	Pivô auxiliar da i-ésima iteração do método ACA+.
Símbolos Matemáticos	
N	Conjunto dos números naturais.
R	Conjunto dos números reais.
ξ_i	i-ésimo eixo coordenado de maior variância dos dados.
k	Índice que indica a iteração.
n_{int}	Número de iterações.
n_{elem}	Número de elementos.

CAPÍTULO 4 e 9

Relacionados à Geometria		
S	Contorno de um corpo qualquer.	
S_j	Segmento j do contorno.	
Γ_j	Elemento j de contorno que aproxima S_j.	
$\vec{t} = t_x\vec{i} + t_y\vec{j}$	Vetor unitário tangente ao contorno S e suas componentes.	
$\vec{n} = n_x\vec{i} + n_y\vec{j}$	Vetor unitário normal ao contorno S e suas componentes.	
$\vec{r} = r_x\vec{i} + r_y\vec{j}$	Vetor distância radial e suas componentes.	$[m]$
(x_d, y_d)	Coordenada do ponto fonte.	$[m]$
θ_{int}	Ângulo interno do contorno S.	$[rad]$
N_i	Função de forma i.	
ξ	Sistema de coordenadas local.	
L	Comprimento do elemento Γ_j.	$[m]$

Símbolos Físicos		
A	Área	$[m^2]$
k	Condutividade térmica.	$[W/(m \cdot K)]$
$\dot{Q}_i$	Fluxo de calor na direção i.	$[W]$
$\dot{Q}_i\vec{i}$	Vetor fluxo de calor na direção i.	$[W]$
T	Temperatura.	$[K]$
T^*	Solução fundamental para a temperatura.	$[K]$
q	Fluxo de calor.	W/m^2
q^*	Solução fundamental para o fluxo de calor.	W/m^2
$\vec{q}$	Vetor fluxo de calor por unidade de área.	W/m^2
$\dot{q}_g$	Calor gerado por unidade de volume.	W/m^3
$r = \lvert\vec{r}\rvert$	Distância radial.	$[m]$
$\overline{T_j}$ *ou* $\overline{q_j}$	Variáveis T e/ou q de valor conhecido.	
H	Matriz que contém as integrais $\int_{\Gamma_j} N_i q^* d\Gamma$.	
G	Matriz que contém as integrais $\int_{\Gamma_j} N_i T^* d\Gamma$.	

CAPÍTULO 5 e 6

Relacionado a Matrizes	
A	Matriz qualquer.
A'	Matriz A simétrica obtida do produto de $A^T A$.
P	Matriz de pré-condionamento.
I	Matriz Identidade.
M	Matriz que controla o comportamento do erro.
M'	Matriz que controla o comportamento do erro no método Multigrid.
E	Matriz que contém os autovetores (diagonalização da matriz).
Λ	Matriz que contém os em sua diagonal os autovalores (diagonalização da matriz).
D	Matriz que contém a diagonal de A**.**
L	Matriz que contém a parte estritamente triangular superior de A.
R	Matriz de Restrição do Método Multigrid.
T	Matriz de interpolação do Método Multigrid.
$A_{2^n h}$	Matriz resultante da restrição da matriz a n vezes.
K_j	Matriz que contém as bases que geram $\mathbb{K}_j$.
Q	Matriz que contém os vetores da base ortonormal (Decomposição QR).
R	Matriz triangular superior que guarda o processo feito em Gram-Schmidt (Decomposição QR).
V	Matriz que contém os vetores da base ortonormal (Arnoldi).
H	Matriz superior de Hessenberg guarda o processo feito no método de Arnoldi.

H'	Matriz H transformada pelas rotações de Givens.
G	Matriz que realiza a rotação de Givens.
R^{mxn}	Conjunto das matrizes reais com m linhas e n colunas
λ_j	Autovalor j.
ρ	Raio espectral.
Relacionado aos Métodos Iterativos	
k	Índice que indica a iteração.
x_k	Vetor x obtido na k-ésima iteração.
e_k	Erro de x_k na k-ésima iteração.
$e_{2^n h}$	Erro da malha $2^n h$, resultante de n restrições no método Multigrid.
E_h	Erro interpolado para a malha original no método Multigrid.
r_k	Resíduo da k-ésima iteração.
$r_{2^n h}$	Resíduo da malha $2^n h$, resultante de n restrições.
ω	Fator de ponderação do método de Jacobi e método SOR.
h	Espaçamento entre os elementos da malha, também indica o tamanho da malha.
$\mathbb{K}_j$	j-ésimo subespaço de Krylov.
t_j	Vetor da base de K_j ainda não ortonormalizado.
v_j	Vetor que compõe a base já normalizada de K_j (Arnoldi).
$\propto_k$	Tamanho do passo da k-ésima iteração no método CG.
d_k	Direção de procura da k-ésima iteração no método CG.
β_{k+1}	Fator utilizado para obter a nova direção de procura no método CG.
m	Parâmetro de recomeço do método GMRES(m).
n_{int}	Número de iterações.

ABREVIAÇÕES

ACA	Aproximação Cruzada Adaptativa.
ACA+	Aproximação Cruzada Adaptativa Modificada.
CAD	Desenho Auxiliado por Computador.
CAE	Engenharia Auxiliada por Computador.
CAM	Manufatura Auxiliada por Computador.
CG	Gradiente Conjugado.
GMRES	Resíduos Mínimos Generalizados.
MEC	Método dos Elementos de Contorno.
MEF	Método dos Elementos Finitos.
PCA	Análise de Componentes Principais.
SOR	Super-relaxação sucessiva.
SVD	Decomposição em valor singular.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

A evolução tecnológica que começou por volta de cinquenta anos atrás e que continua até os dias atuais possibilitou um grande avanço dos sistemas de computação e de suas capacidades de processamento. Esse avanço viabilizou o desenvolvimento de diferentes métodos computacionais para a solução de uma variada gama de problemas, entre eles problemas de engenharia. A utilização de computadores para auxiliar na solução de problemas de engenharia (CAE) transformou o modo como o estudo de sistemas mecânicos pode ser feito. Atualmente, a análise do comportamento de estruturas de um projeto é em grande parte feita usando computadores, pois os métodos computacionais são muito mais baratos e flexíveis do que os métodos experimentais. Tais métodos numéricos possibilitam facilmente a mudança de parâmetros geométricos e de condições de contorno de um problema. Assim, torna-se não só a análise preliminar, mas também qualquer análise posterior mais rápida e menos onerosa. No entanto, a aplicação direta desses métodos é limitada pela capacidade computacional disponível, o que torna virtualmente impossível a resolução de problemas de larga escala devido à grande complexidade dos algoritmos utilizados, os quais demandam uma quantidade enorme de memória e capacidade de processamento. Tais problemas computacionais comumente envolvem resolver problemas de álgebra linear de grandes dimensões, envolvendo matrizes com quantidades enormes de dados. Assim, com o objetivo de diminuir a complexidade desses algoritmos, tornando possível a resolução desses problemas com utilização de menos recursos computacionais, diferentes métodos vêm sendo desenvolvidos. Métodos os quais, em geral, combinam a utilização de uma representação aproximada da matriz a ser resolvida com a aplicação de um método iterativo para a resolução do sistema.

1.2 OBJETIVO

Esse trabalho tem como objetivo oferecer um método de menor complexidade que os algoritmos tradicionais para a resolução de problemas de álgebra linear que contém matrizes de larga escala advindas de problemas formulados pelo Método dos elementos de contorno (MEC).

O método inclui o tratamento da matriz cheia e não simétrica resultante da aplicação do MEC, o que engloba a representação dessa matriz por matrizes hierárquicas, as quais permitem uma representação esparsa dos dados usando o algoritmo da aproximação cruzada adaptativa (ACA). A resolução do sistema é então feita, utilizando o método dos resíduos mínimos generalizados (GMRES), um método iterativo para resolução de sistemas lineares. Uma vez implantado o algoritmo, será possível à resolução de problemas de maior escala em menor tempo e com menor uso da memória.

Neste trabalho, primeiramente, será demonstrada a base teórica necessária para implementar o método, além da criação de rotinas genéricas para comparação do desempenho do método GMRES com diferentes métodos iterativos para a resolução de sistemas de equações na forma matricial. Na parte final, uma rotina de programação para o método ACA será implementada. Os resultados obtidos, assim como a própria rotina serão analisados de forma a esclarecer e tornar mais visível suas limitações e implicações.

1.3 MÉTODOS NÚMERICOS PARA ANÁLISE EM CAE

A engenharia auxiliada por computador (CAE), envolve todo o uso de computadores em atividades de engenharia, tais como, no design (CAD), análise e manufatura (CAM). Na análise, os dois métodos mais conhecidos e largamente utilizados são o Método dos Elementos Finitos (MEF) e o Método dos Elementos de Contorno (MEC). O MEF é o mais conhecido e também o mais utilizado atualmente, sua formulação e desenvolvimento ocorreu anterior ao MEC, no início da década de 1960, já o segundo ganhou notabilidade somente dez anos mais tarde, sendo utilizado principalmente devido ao seu menor custo computacional quando comparado ao MEF. Embora parcialmente diferentes em suas formulações, ambos os métodos tem a mesma aplicação, eles constituem uma técnica de análise do comportamento de sistemas mecânicos, principalmente de estruturas sobre a ação de cargas externas. Entenda por carga externas qualquer fonte externa que produz uma função de campo não nula (campo de temperatura, campo de tensão, etc.) que descreve o comportamento do sistema, como por exemplo, fluxo de calor, forças de contato, forças de campo e outras condições de contorno. Visto que ambos os métodos tem a mesma função, é normal indagar o porquê da necessidade de diferentes formulações. A resposta é que, para determinadas classes de problemas, o MEF é ineficiente e

trabalhoso, consumindo grande quantidade de tempo, memória de processamento e fornecendo resultados insatisfatórios.

1.3.1 MÉTODO DOS ELEMENTOS FINITOS (MEF)

O método dos elementos finitos é um método de domínio, ou seja, ele é formulado a partir da discretização de todo o corpo em uma série de elementos diferenciais. Esse método foi desenvolvido para resolver equações na forma diferencial. A acurácia e a precisão da solução dependem de como são definidos esses elementos geométricos (constituídos por arestas e nós) e o quão bem eles representam o volume que foi discretizado, ou seja, eles dependem da precisão da malha. O comportamento das arestas e consequentemente a forma do elemento são descritas pelas chamadas funções de forma. Essas funções são utilizadas tanto para descrever o comportamento geométrico do elemento como o comportamento da solução no elemento. Considere, por exemplo, um problema de MEF 2D, dois dos possíveis elementos a de serem utilizados para discretrizar o domínio são mostrados na Figura 1.

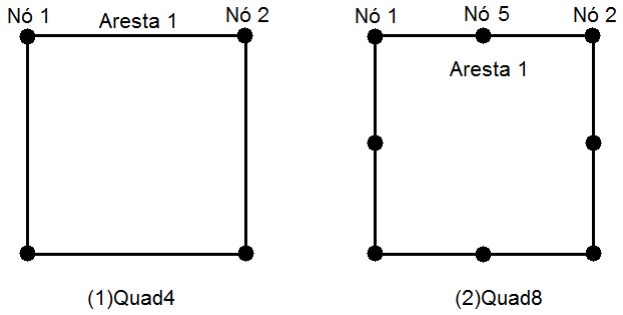

Figura 1 - Elementos quadrilaterais para problemas de MEF planos.

Considere o elemento à esquerda (1), também chamado de QUAD4 (quadrilateral com 4 nós), onde cada aresta é definida por somente dois nós. Assim a maior aproximação de qualquer função $f(x,y)$ possível na aresta 1 é uma aproximação linear, ou seja, obtido uma grandeza (posição ou potencial) nos nós 1 e 2, a mesma grandeza em qualquer ponto da aresta é simplesmente um valor $f(x,y)$ de uma função linear que passa por $f(x_1,y_1)$ e $f(x_2,y_2)$ (Figura 2).

Considere agora o elemento à direita (2), cada aresta é definida por três nós, possibilitando uma aproximação quadrática da forma do elemento. Isto permite que ele represente de melhor forma corpos com curvas suaves e também ofereça uma melhor aproximação da solução no elemento.

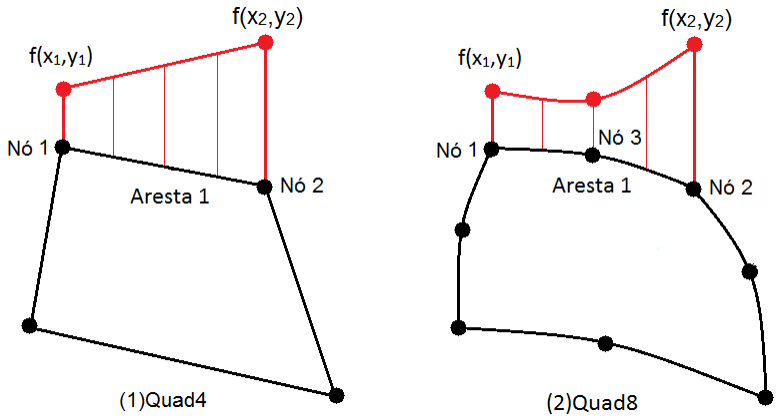

Figura 2 - Aproximação feita por diferentes elementos quadrilaterais.

Embora a análise tenha sido feita somente na aresta 1, ela pode ser estendida sobre todo o elemento, incluindo pontos internos. No caso do elemento (1) a aproximação é continua linear, mas é dada por um plano. No caso do elemento (2) essa é dada por uma superfície quadrática. Mais informações sobre as funções de forma serão discutidas na unidade 9.8.1.

Construída a malha, um sistema de equações lineares é então gerado para calcular o potencial em cada nó. Essas equações são geralmente obtidas a partir da forma fraca da equação diferencial, usando funções peso obtidas a partir de uma função arbitrária que obedece às condições de contorno.

Entre as desvantagens do método dos elementos finitos podemos citar:

- Discretização sobre todo o domínio ocupado pelo corpo, o que torna a geração e a inspeção da malha uma operação laboriosa e demorada.
- Em domínios infinitos é necessário criar um contorno fictício fechado para aplicação do método, o que reduz a acurácia da solução, podendo levar a resultados incorretos.
- Em problemas descritos por funções diferenciais de quarta ordem ou maiores (i.e. equações de placas, equações de cascas), os requerimentos de conformidades, ou seja, o conjunto de requerimentos matemáticos e geométricos que a malha deve satisfazer para que seja possível obter uma solução, pode tornar a formação da malha uma operação que consome grande quantidade de tempo e de recursos computacionais.
- Embora o método tenha grande acurácia em computar a função de campo, sua acurácia cai drasticamente quando tenta obter as derivadas do mesmo, principalmente em regiões de grandes gradientes.

Grande parte das deficiências apontadas acima são resultados da grande complexidade que é a geração de uma malha para a resolução de problemas pelo MEF [1]. Muitas vezes o problema da geração de uma malha confiável é mais trabalhoso e complexo do que a formulação do problema físico a qual a mesma irá resolver.

1.3.2 MÉTODOS DOS ELEMETOS DE CONTORNO (MEC)

O método dos elementos de contorno [2] [3] é um método de fronteira, ou seja, ele é formulado a partir da discretização de apenas o contorno de um corpo. Assim, o problema apresenta dimensão reduzida em um. Dessa forma, problemas de volume tornam de superfície e os de superfície em de linha. O método foi criado para resolver equações diferenciais parciais, as quais são transformadas em equações integrais de contorno.

Diferentemente do MEF, o MEC não utiliza uma função arbitraria como função peso para encontrar a forma fraca das equações diferenciais parciais, mas sim, uma solução analítica que representa os efeitos de uma carga pontual em um ponto de um domínio infinito [4], solução a qual é chamada de solução fundamental. O fato de utilizar uma solução fundamental fornece maior precisão em regiões de grandes gradientes.

As vantagens de utilizar o método dos elementos de contorno são:

- A discretização ocorre somente sobre o contorno do corpo, facilitando a modelagem da malha (Figura 3).

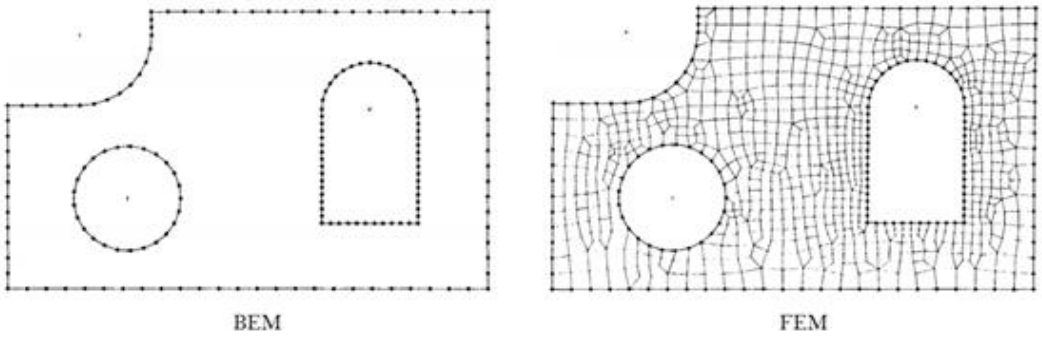

Figura 3 - Comparação entre Discretização do corpo entre MEF (elementos quad4) e MEC (elementos linerares).

- Em domínios infinitos, a condição de contorno no infinito é formulada como exterior ao contorno, sendo que a solução fundamental deve satisfazer algumas condições no infinito.

- O método é efetivo para calcular as derivadas da função de campo, podendo facilmente lidar com cargas concentradas tanto dentro do domínio quanto em seu contorno.
- O método permite analisar a função de campo e suas derivadas em qualquer ponto do corpo, diferentemente do MEF onde a solução é originalmente obtida somente nos pontos nodais e nos demais pontos é interpolada.

O método dos elementos de contorno, no entanto, também apresenta algumas deficiências:

- A equação integral de contorno requer uma solução fundamental, não podendo então ser obtida em problemas no qual ela não existe ou não é conhecida, como por exemplo, problemas descritos por equações diferenciais com coeficientes não constantes. Nestes casos, a utilização do MEC gera não somente integrais de contorno, mas também de domínio, as quais devem ser transformadas em integrais de contorno por técnicas apropriadas.
- A aplicação do método resulta em um sistema de equações lineares no qual a matriz de coeficientes é uma matriz cheia e não simétrica, diferentemente do MEF onde a matriz é esparsa e simétrica (Figura 4). Tal problema é amenizado pela menor dimensão da matriz gerada pelo MEC quando comparado ao MEF.

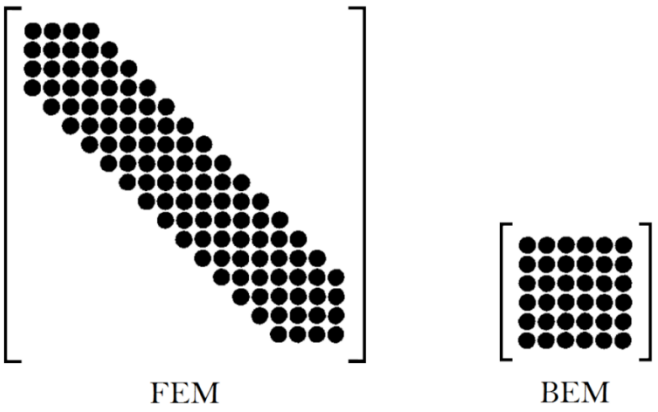

Figura 4 - Representação da distribuição dos coeficientes não nulos na matriz de coeficientes.

2 ALGORITMOS DE BAIXA COMPLEXIDADE

Embora tipicamente o método dos elementos de contorno apresente um menor custo computacional, devido a menor dimensão da matriz gerada, é ainda de interesse desenvolver algoritmos que tornem o processo mais rápido e barato, permitindo então a análise de problemas maiores e mais complexos. Tais algoritmos, chamados de algoritmos rápidos, têm como princípio diminuir a quantidade de operações necessárias para resolver um sistema de equações matriciais, obtendo uma solução aproximada através do uso de algoritmos de complexidade computacional reduzida, ou seja, algoritmos em que o número de operações aumenta pouco com aumento do problema a ser resolvido. Assim, requisitos de tempo e memória serão reduzidos, o que aumenta a capacidade de resolver problemas maiores e mais complexos de forma eficiente.

2.1 NOTAÇÃO DE LANDAU $\boldsymbol{O}(\boldsymbol{N})$

A complexidade de um algoritmo [5] [6] é comumente avaliada utilizando à notação de Landau, ou também chamada de notação grande-O ou notação assintótica. Essa notação é utilizada na análise de algoritmos, mais especificamente na ciência da computação ela é utilizada para descrever o comportamento assintótico de uma função. Basicamente, ela descreve a taxa de crescimento de uma função, também chamada de ordem da função, por isso o nome grande-O.

É importante não confundir desempenho com complexidade. O primeiro está relacionado à quantidade de memória e o tempo de execução efetivamente usado pela máquina, o que obviamente, depende da máquina. Enquanto o segundo está relacionado ao comportamento dos recursos utilizados, ou seja, o que acontece caso o tamanho do problema a ser solucionado mude. Enquanto a complexidade afeta a desempenho, o contrário não ocorre.

A notação de Landau é formalmente definida considerando duas funções, $f(x)$ e $g(x)$.

$$f(x), g(x): \{x \subset \mathbb{N}\} \rightarrow \mathbb{N}$$

$$f(x) = O\big(g(x)\big) \leftrightarrow \exists\, N, C \in \mathbb{R};\ tal\ que$$

$$|\mathrm{f(x)}| \leq \mathrm{C}|\mathrm{g(x)}|;\ \forall\, \mathrm{x} > \mathrm{N}$$

De forma simplificada, isso significa que a taxa de crescimento de $f(x)$ é sempre menor que $|g(x)|$, se aproximando ao ponto que N tende a infinito, por isso o nome notação assintótica. É importante não confundir $f(x)$ o qual é a quantidade exata, com $g(x)$ o qual é um limite superior para a taxa de crescimento.

Esse tipo de notação é utilizado, pois não há interesse no número exato de operações, mas na relação entre o número de operações e a quantidade de dados de entrada. Para exemplificar, considere um algoritmo cuja quantidade de operações necessárias para a realização do mesmo seja dada por $f(n) = 7n^2 - n$, onde n é o número de entradas, assim, a complexidade do algoritmo é $O(N^2)$, observe que $7|n^2 - n| \leq 7|n^2|$ para todo $n > 0$ (definição formal). É importante também observar que $7|n^2 - n| \leq 7|n^2| \leq |n^3|$, mas a complexidade não é $O(N^3)$, pois o interesse está no menor limite superior. Na prática, no entanto, a definição formal não é utilizada, a notação pode ser obtida excluindo qualquer constante multiplicativa e qualquer termo de menor ordem na função que descreve o algoritmo.

Essa notação pode ser usada para analisar o efeito da quantidade de dados de entrada no tempo requerido ou no espaço na memória necessário para executar um algoritmo. O tempo requerido pelo algoritmo é estimado contando o número de operações elementares realizadas pelo algoritmo, onde cada operação elementar demora um intervalo fixo de tempo. Assim o tempo levado para a execução do algoritmo é proporcional à quantidade de operações básicas efetuadas e a complexidade de ambos é, então, a mesma. São operações básicas:

- Uma operação aritmética
- Uma gravação
- Uma leitura
- Uma atribuição
- Um teste de hipótese

A menor complexidade possível em qualquer operação é a complexidade linear $O(N)$, isso, pois para qualquer mapeamento não trivial $f: X_n \to Y_m$ onde n é número de dados de entradas e m é o número de saídas, o menor custo computacional é $O(N)$ onde $N = max\{n, m\}$.

2.2 COMPLEXIDADE EM ALGEBRA LINEAR

Problemas computacionais de grande escala comumente envolvem resolver problemas de álgebra linear de grandes dimensões, envolvendo matrizes com quantidades enormes de dados. Considere agora as operações efetuadas em problemas de álgebra linear, as operações vetoriais possuem complexidade $O(N)$. É importante lembrar que o produto vetorial apresenta complexidade $O(N^2)$, mas tal operação não é utilizada em álgebra linear. Por outro lado, as operações matriciais requerem complexidade $O(N^2)$ e $O(N^3)$.

Para exemplificar a complexidade de uma operação matricial, usaremos a eliminação de Gauss, o método mais conhecido para resolver um sistema de equações lineares na forma matricial. Considere uma matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$, onde no processo de eliminação de Gauss não há pivôs nulos, ou seja, não há troca de linhas e nenhuma das linhas é um vetor nulo. Na eliminação de Gauss, o primeiro elemento a_{11} é definido como o pivô (Figura 5). O pivô é utilizado para eliminar todos os coeficientes abaixo deste, ou seja, todos os coeficientes que pertencem à mesma coluna do pivô, mas que pertencem às linhas abaixo deste.

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,(n-2)} & a_{1,(n-1)} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,(n-2)} & a_{2,(n-1)} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,(n-2)} & a_{3,(n-1)} & a_{3,n} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ a_{(n-2),1} & a_{(n-2),2} & a_{(n-2),3} & \cdots & a_{(n-2),(n-2)} & a_{(n-2),(n-1)} & a_{(n-2),n} \\ a_{(n-1),1} & a_{(n-1),2} & a_{(n-1),3} & \cdots & a_{(n-1),(n-2)} & a_{(n-1),(n-1)} & a_{(n-1),n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,(n-2)} & a_{n,(n-1)} & a_{n,n} \end{bmatrix}$$

Figura 5 - Matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$ com pivô em destaque.

Através de operações básicas entre a linha do pivô e as linhas subsequentes, essas eliminações são feitas e então é obtida uma matriz reduzida $\boldsymbol{R} \in \mathbb{R}^{nxn}$. Uma submatriz $\boldsymbol{B} \in \mathbb{R}^{(n-1)x(n-1)}$ pode então ser definida a partir da matriz $\boldsymbol{R}$ como a matriz que não contém a linha e a coluna do pivô anterior (em cinza na Figura 6).

$\mathbf{R(1)} =$

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	. . . .	$a_{1,(n-2)}$	$a_{1,(n-1)}$	$a_{1,n}$
0	$r_{2,2}$	$r_{2,3}$	. . . .	$r_{2,(n-2)}$	$r_{2,(n-1)}$	$r_{2,n}$
0	$r_{3,2}$	$r_{3,3}$	. . . .	$r_{3,(n-2)}$	$r_{3,(n-1)}$	$r_{3,n}$
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
0	$r_{(n-2),2}$	$r_{(n-2),3}$	. . . .	$r_{(n-2),(n-2)}$	$r_{(n-2),(n-1)}$	$r_{(n-2),n}$
0	$r_{(n-1),2}$	$r_{(n-1),3}$	. . . .	$r_{(n-1),(n-2)}$	$r_{(n-1),(n-1)}$	$r_{(n-1),n}$
0	$r_{n,2}$	$r_{n,3}$	. . . .	$r_{n,(n-2)}$	$r_{n,(n-1)}$	$r_{n,n}$

Figura 6 - Matriz $\boldsymbol{R} \in \mathbb{R}^{nxn}$, resultante da primeira iteração, com pivô em destaque.

Definido a matriz $\boldsymbol{B}$, todo o processo acima é repetido nessa submatriz. E então na submatriz seguinte até que a última submatriz $\boldsymbol{B}$ seja o último elemento da matriz. Observe que essa submatriz muda a cada iteração, tal que $\boldsymbol{B} \in \mathbb{R}^{(n-i+1)x(n-i+1)}$ onde i corresponde a i-ésima iteração. Após todas as iterações, o resultado é uma matriz triangular superior $\boldsymbol{L} \in \mathbb{R}^{nxn}$, onde os elementos da diagonal foram os pivôs do processo de eliminação (em azul Figura 7).

$\mathbf{R(n\text{-}1)} =$

$a_{1,1}$	$a_{1,2}$	$a_{1,3}$	. . . .	$a_{1,(n-2)}$	$a_{1,(n-1)}$	$a_{1,n}$
0	$r_{2,2}$	$r_{2,3}$	. . . .	$r_{2,(n-2)}$	$r_{2,(n-1)}$	$r_{2,n}$
0	$r_{3,2}$	$r_{3,3}$	. . . .	$r_{3,(n-2)}$	$r_{3,(n-1)}$	$r_{3,n}$
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
0	0	0	. . . .	$r_{(n-2),(n-2)}$	$r_{(n-2),(n-1)}$	$r_{(n-2),n}$
0	0	0	. . . .	0	$r_{(n-1),(n-1)}$	$r_{(n-1),n}$
0	0	0	. . . .	0	0	$r_{n,n}$

Figura 7 - Forma escalonada reduzida da matriz $\boldsymbol{A}$, apos o processo de eliminação.

Na primeira iteração são efetuadas $2n(n-1)$ operações. Observe que a primeira linha contém n elementos e que cada elemento é multiplicado por uma mesma constante e então somado ao elemento da linha abaixo, o que contabiliza $2n$ operações. Isso é feito para todas as linhas com exceção da primeira linha, a qual é mantida intacta, assim a operação anterior é repetida $(n-1)$ vezes, totalizando $2n(n-1)$ operações efetuadas na primeira iteração.

Na segunda iteração, a submatriz $\boldsymbol{B}$ tem dimensão $(n-1)x(n-1)$. O processo anterior pode ser repetido nesta obtendo, analogamente a iteração anterior, $2(n-1)(n-2)$ operações. São feitas $n-1$ iterações até a obtenção da matriz final. Assim, o total de operações é dado por:

$$2[\mathrm{n}(\mathrm{n}-1)+(\mathrm{n}-1)(\mathrm{n}-2)+\ldots+1+0] = 2\sum_{i=0}^{n-1}(n-i)(n-i-1) = 2\left[\frac{n^3}{3}-\frac{n}{3}\right]$$

Portanto, a operação de eliminação apresenta complexidade $O(N^3)$ e a necessidade de armazenamento na memória é proporcional à $O(N^2)$ uma vez que todas as entradas de uma matriz nxn devem ser armazenadas.

Deste modo, a grande complexidade computacional associada ao processo de eliminação torna extremamente caro e limitada a resolução de matrizes cheias de grandes dimensões. Devido a essa limitação, novos métodos procuram trabalhar com tipos especiais de matrizes, como por exemplo, matrizes diagonais. No entanto, são poucos os casos em que uma matriz pode ser transformada em uma matriz diagonal de forma vantajosa, isso, pois a matriz que efetua essa transformação geralmente é uma matriz cheia e seu produto deve ser computado. Assim, os tipos de matrizes mais utilizados são as matrizes esparsas, pois a quantidade de entradas não nulas são aproximadamente da ordem de $O(N)$. Assim, tanto o armazenamento quanto as operações de soma e operações matriz-vetor são de complexidade próximas a $O(N)$. O fato da operação $\boldsymbol{A}x$ requerer somente $O(N)$ é a maior razão para a utilização desses métodos. Desta forma, caso o objetivo seja somente realizar o produto matriz-vetor, as matrizes esparsas são ideais. Essa é uma das razões do sucesso do método dos elementos finitos. Na prática, grande parte dos algoritmos que empregam tais métodos atinge complexidade O$(Nlog^qN)$ onde $q>0$, chamada de complexidade quase linear pois, com exceção das matrizes diagonais não há classe de matrizes que permita complexidade $O(N)$.

Objetivando resolver esses sistemas de equações de maneira mais rápida, são utilizados os chamados algoritmos rápidos. Esses usam métodos iterativos para a resolução de sistema de equações lineares, onde cada iteração apresenta complexidade $O(N^2)$. O que ocorre em cada iteração é um produto matriz vetor. Assim, a execução do algoritmo apresenta complexidade O$(n_{int}N^2)$, onde n_{int} é o número total de interações para a obtenção de certa convergência. Deste modo, caso

o número de iterações seja pequeno, o uso de métodos iterativos torna mais rápido o processo de resolução para matrizes de grande ordem. Tais métodos, no entanto, não reduzem os requerimentos de memória, uma vez que toda a matriz precisa ser armazenada para conseguir esse efeito. Então, além da utilização de métodos iterativos são também utilizados métodos para manipulação das matrizes de forma a obter uma representação da matriz de maneira mais esparsa e com blocos de matrizes de menor posto, o que diminui tanto o tempo computacional da operação quanto o requerimento de memória.

3 MATRIZES ESPARSAS

3.1 MATRIZES DE BAIXO POSTO

Existem várias definições diferentes, no entanto equivalentes, sobre o que é o posto de uma matriz. Uma das definições é que o posto k de uma matriz $\boldsymbol{A} \in \mathbb{R}^{mxn}$ é a dimensão $\mathbb{R}$ do espaço vetorial gerado pelo espaço coluna daquela matriz, onde $k \leq mín\{m, n\}$. O posto de uma matriz indica a quantidade de vetores linearmente independentes que formam o espaço coluna e pode ser obtido por eliminação de Gauss, onde o número de pivôs não nulos da matriz reduzida é igual ao posto.

Considere matriz $\boldsymbol{A} \in \mathbb{R}^{3x3}$ abaixo. É fácil perceber que a segunda coluna e a terceira coluna são múltiplas da primeira, assim ambas as colunas são combinações da primeira, logo apenas a primeira coluna é linearmente independente e o posto esperado da matriz $\boldsymbol{A}$ é $k = 1$.

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 8 \end{bmatrix} \xrightarrow{Elimınação\ de\ Gauss} \boldsymbol{R} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$2º\ coluna - 2 \cdot 1ºcoluna$$
$$3º\ coluna - 3 \cdot 1ºcoluna$$

A eliminação de Gauss resulta em uma matriz com apenas um pivô não nulo, confirmando que o posto da matriz $\boldsymbol{A}$ é $k = 1$. As operações entre linhas realizadas no processo de eliminação podem ser guardadas em uma matriz $\boldsymbol{E}$, podendo todo o processo de eliminação ser descrito pelo produto $\boldsymbol{EA}$.

$$\boldsymbol{EA} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \boldsymbol{R}$$

O interessante é que $\boldsymbol{A}$ pode agora ser escrito como $\boldsymbol{A} = \boldsymbol{E^{-1}R}$ também chamado de decomposição $\boldsymbol{LU}$.

$$\boldsymbol{E^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 0 & 1 \end{bmatrix}$$

O produto $\boldsymbol{E^{-1}R}$ pode então ser computado para obter $\boldsymbol{A}$. No entanto, não é preciso efetuar o produto de toda a matriz $\boldsymbol{E^{-1}}$ com toda matriz $\boldsymbol{R}$, mas somente das linhas não nulas de $\boldsymbol{R}$ e das colunas diferentes da matriz identidade de $\boldsymbol{E^{-1}}$.

$$\begin{bmatrix}1 & 0 & 0\\ 2 & 1 & 0\\ 3 & 0 & 1\end{bmatrix}.\begin{bmatrix}1 & 2 & 3\\ 0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}=\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 6\\ 3 & 6 & 8\end{bmatrix}=\begin{bmatrix}1\\ 2\\ 3\end{bmatrix}.\begin{bmatrix}1 & 2 & 3\end{bmatrix}$$

Um melhor modo de entender o porquê de não ser necessário utilizar toda a matriz, é entender o produto entre as duas matrizes como $\sum \boldsymbol{R}(:,n)\boldsymbol{E^{-1}}(n,:)$, ou seja, a soma das matrizes resultantes do produto da coluna n da matriz $\boldsymbol{R}$ com a linha n da matriz $\boldsymbol{E^{-1}}$.

$$\begin{bmatrix}1\\ 2\\ 3\end{bmatrix}.\begin{bmatrix}1 & 2 & 3\end{bmatrix}+\begin{bmatrix}0\\ 1\\ 0\end{bmatrix}.\begin{bmatrix}0 & 0 & 0\end{bmatrix}+\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}.\begin{bmatrix}0 & 0 & 0\end{bmatrix}=\begin{bmatrix}1 & 2 & 3\\ 2 & 4 & 6\\ 3 & 6 & 8\end{bmatrix}$$

Assim uma matriz $\boldsymbol{A} \in \mathbb{R}^{3x3}$ pode ser representada por um produto de dois vetores de dimensão $\mathbb{R}^{3x1}$.

De modo geral, qualquer matriz $\boldsymbol{A} \in \mathbb{R}^{mxn}$ de posto K pode ser representada de forma exata por um produto entre duas matrizes $\boldsymbol{V} \in \mathbb{R}^{mxk}$ e $\boldsymbol{U} \in \mathbb{R}^{kxn}$. No caso anterior, o posto k da matriz $\boldsymbol{A} \in \mathbb{R}^{3x3}$ era 1 e as matrizes $\boldsymbol{V}$ e $\boldsymbol{U}$ vetores $\in \mathbb{R}^{3x1}$. Esse tipo de representação apresenta duas vantagens [7]:

- A quantidade reduzida de dados guardados, pois é necessário somente $k(m+n)$ entradas, ao invés de mn.
- O menor número de operações efetuadas, pois o produto $\boldsymbol{A}x$ se torna $\boldsymbol{UV}x = \boldsymbol{U}(\boldsymbol{V}x)$ e quantidade de operações necessárias para realizar esse produto é $2k(m+n)-m-n$, ao invés de $2mn-m$.

Observe que o dito acima só é vantagem caso o posto k da matriz seja pequeno. Para que ela seja considerada de baixo posto, ela deve satisfazer:

$$k(m+n) < mn$$

No entanto poucas matrizes são globalmente de baixo posto, então é necessária a divisão da matriz em blocos, as chamadas matrizes hierárquicas. Outro problema é que a decomposição $\boldsymbol{LU}$ apresenta complexidade $O(N^3)$, o que torna inviável utilizar esse método para obter as matrizes $\boldsymbol{V} \in \mathbb{R}^{mxk}$ e $\boldsymbol{U} \in \mathbb{R}^{kxn}$. Assim, outros métodos são utilizados, como por exemplo, a aproximação cruzada adaptativa.

3.2 DECOMPOSIÇÃO ESQUELETO

O conjunto de técnicas que possibilitam reduzir de forma eficaz o uso de memória no armazenamento de matrizes de grandes dimensões, reduzindo a quantidade de dados a serem armazenados sem grande perda de acurácia são chamadas de aproximação de baixo posto.

O algoritmo mais conhecido para a aproximação de baixo posto é a decomposição em valores singulares. Dado uma matriz $\boldsymbol{A} \in \mathbb{R}^{mxn}$, ela pode ser decomposta na forma:

$$\boldsymbol{A} = \boldsymbol{USV}^T$$

onde $\boldsymbol{U} \in \mathbb{R}^{mxr}$, $\boldsymbol{S} \in \mathbb{R}^{rxr}$ e $\boldsymbol{V} \in \mathbb{R}^{rxn}$. A matriz $\boldsymbol{S}$ é uma matriz diagonal, cujos valores são chamados de valores singulares. Por convenção, esses valores são listados em ordem decrescente. Em muitos sistemas, o decaimento dos valores singulares ocorre rapidamente, isso é $\frac{s_i}{s_1} \to 0$ quando $i \to r$. Isso permite que a matriz seja aproximada utilizando somente os primeiros valores singulares s_i, ou seja, os maiores. Assim, $\boldsymbol{A}$ pode ser aproximado como:

$$\boldsymbol{A} \cong \widehat{\boldsymbol{U}}\widehat{\boldsymbol{S}}\widehat{\boldsymbol{V}}^T$$

onde $\widehat{\mathbf{U}} \in \mathbb{R}^{mxr^*}$, $\boldsymbol{S} \in \mathbb{R}^{r^*xr^*}$ e $\boldsymbol{V} \in \mathbb{R}^{r^*xn}$, tal que $r^* <<< r$. Embora a decomposição SVD represente a melhor decomposição possível de uma matriz, ela apresenta complexidade $O(N^3)$, o que torna seu uso restrito para problemas de pequenas dimensões. Uma melhor opção então é a decomposição esqueleto.

A decomposição esqueleto [8] diz que uma matriz $\boldsymbol{A} \in \mathbb{R}^{mxn}$ pode ser representada da seguinte forma:

$$\boldsymbol{A} \cong \boldsymbol{C}\widehat{\boldsymbol{A}}^{-1}\boldsymbol{R}$$

onde $\boldsymbol{C} \in \mathbb{R}^{mxk}$, $\widehat{\mathbf{A}} \in \mathbb{R}^{kxk}$ e $\boldsymbol{R} \in \mathbb{R}^{kxn}$ e k é o posto da matriz $\boldsymbol{A}$. A matriz $\boldsymbol{C}$ é formada então por k colunas de $\boldsymbol{A}$, enquanto a matriz $\boldsymbol{R}$ é formada por k linhas de $\boldsymbol{A}$. Por fim, a matriz $\widehat{\mathbf{A}}$ é formada pelos elementos comuns a $\boldsymbol{C}$ e $\boldsymbol{R}$, ou seja, a interseção entre os elementos de $\boldsymbol{C}$ e $\boldsymbol{R}$.

Suponha uma matriz $\boldsymbol{A} \in \mathbb{R}^{5x5}$ de posto $k = 2$, sua decomposição esqueleto é dada pela Figura 8.

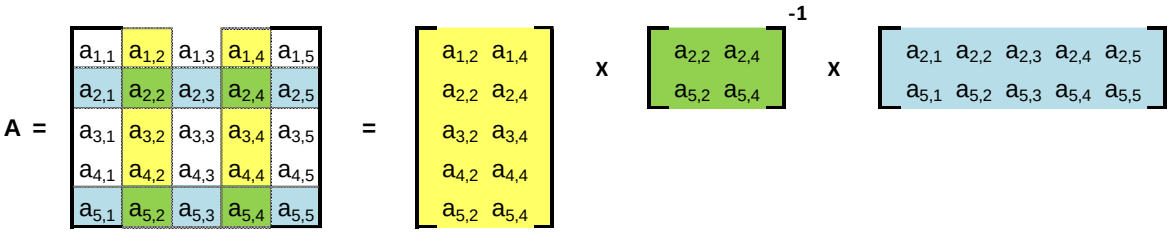

Figura 8 - Decomposição Esqueleto de uma matriz $\boldsymbol{A} \in \mathbb{R}^{5x5}$

O método apresenta complexidade $O(kN^2)$, o que o torna mais viável do que a decomposição SVD para matrizes de baixo posto. O problema a ser resolvido então é saber quais colunas e linhas escolher. Existem diversos algoritmos para efetuar essa escolha, a grande maioria deles é uma variação do método da aproximação cruzada.

3.2.1 APROXIMAÇÃO CRUZADA

Na aproximação cruzada [9] [10] [11] a escolha das linhas e colunas a serem usadas na decomposição esqueleto é feita encontrando na matriz $\boldsymbol{A} \in \mathbb{R}^{mxn}$ o elemento a_{i^*,j^*} de maior valor em módulo. Feito isso, são escolhidos como linha e coluna para a aproximação a linha i^* e a coluna j^* desse elemento. Assim, podem ser definidas as matrizes $\boldsymbol{U} \in \mathbb{R}^{mx1}$ tal que $u_{i,j} = a_{i,j^*}$ e $\boldsymbol{V} \in \mathbb{R}^{1xn}$ tal que $v_{i,j} = \frac{a_{i^*,j}}{a_{i^*,j^*}}$. A aproximação de posto 1 é dada então por $\bar{\boldsymbol{A}} = \boldsymbol{UV}$. Para a obtenção de uma aproximação de posto k é necessário repetir tal processo na matriz diferença $\boldsymbol{A} - \bar{\boldsymbol{A}}$, k vezes. Observe que a cada iteração as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$ crescem em uma ordem, a dimensão dessas matrizes pode então ser dada por $\mathbb{R}^{mxi}$ e $\mathbb{R}^{ixn}$, onde i nesse caso é a i-ésima iteração.

O algoritmo pode ser resumido então da seguinte forma:

1. Achar o pivô $p(i)$ (i^*, j^*) de maior valor em módulo;
2. Definir as matrizes $\boldsymbol{U}$ tal que $u_{i,j} = a_{i,j^*}$ e $\boldsymbol{V}$ tal que $v_{i,j} = \frac{a_{i^*,j}}{p(i)}$;
3. Calcular $\bar{\boldsymbol{A}}(i) = \boldsymbol{UV}$, onde $\boldsymbol{U} \in \mathbb{R}^{mxi}$ e $\mathbf{V} \in \mathbb{R}^{ixn}$;
4. Calcular a matriz diferença $\bar{\boldsymbol{A}}^*(i) = \bar{\boldsymbol{A}}(i-1) - \bar{\boldsymbol{A}}(i)$, onde. $\bar{\boldsymbol{A}}(0) = \boldsymbol{A}$;
5. Repetir processo na matriz diferença até i-ésima iteração.

Considere como exemplo a matriz $\boldsymbol{A} \in \mathbb{R}^{3x3}$, de posto 2, abaixo.

$$\boldsymbol{A} = \begin{bmatrix} 6 & 6 & 4 \\ 4 & 5 & 6 \\ 2 & 4 & 8 \end{bmatrix}$$

O pivô, ou seja, o elemento da matriz que apresenta o maior valor absoluto é o $p(1)(i^*, j^*) = a(3,3) = 8$, agora é possível definir $\boldsymbol{U}$ e $\boldsymbol{V}$**.**

$$\boldsymbol{U} = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}; \boldsymbol{V} = \frac{1}{8}[2 \quad 4 \quad 8] \rightarrow \boldsymbol{V} = [0{,}25 \quad 0{,}5 \quad 1]$$

Definido $\boldsymbol{U}$ e $\boldsymbol{V}$, é possível então calcular $\overline{\boldsymbol{A}}(1) = \boldsymbol{UV}$.

$$\overline{\boldsymbol{A}}(1) = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}[0{,}25 \quad 0{,}5 \quad 1] = \begin{bmatrix} 1 & 2 & 4 \\ 1{,}5 & 3 & 6 \\ 2 & 4 & 8 \end{bmatrix}$$

A matriz $\overline{\boldsymbol{A}}(1)$ é a aproximação de posto 1 da matriz $\boldsymbol{A}$**.** Observe que na aproximação a linha e a coluna do pivô $(i = 3, j = 3)$, são exatas, ou seja, idênticas as da matriz $\boldsymbol{A}$ original. Tal fato irá sempre ocorrer, e resulta de como a matriz $\boldsymbol{U}$ e $\boldsymbol{V}$ foram definidas. Observe que a matriz $\boldsymbol{V}$ foi obtida pela divisão dos elementos da linha do pivô pelo próprio pivô. Assim, quando o produto $\boldsymbol{UV}$ ocorre e o pivô, que e sempre um elemento de $\boldsymbol{U}$, multiplica $\boldsymbol{V}$, a linha original da matriz $\boldsymbol{A}$ é recuperada. No caso da coluna, a matriz $\boldsymbol{V}$ sempre terá um elemento unitário, resultado da divisão do pivô por ele mesmo. Assim, o produto $\boldsymbol{UV}$ sempre contém $\boldsymbol{U}$ que é uma coluna da matriz original.

Continuando o processo é calculado $\overline{\boldsymbol{A}}^*(1) = \overline{\boldsymbol{A}}(0) - \overline{\boldsymbol{A}}(1) = \boldsymbol{A} - \overline{\boldsymbol{A}}(1)$.

$$\overline{\boldsymbol{A}}^*(1) = \boldsymbol{A} - \overline{\boldsymbol{A}}(1) = \begin{bmatrix} 5 & 4 & 0 \\ 2{,}5 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Observe que as a linha e a coluna correspondente ao pivô $(i = 3, j = 3)$ são nulas, o que confirma que são exatas.

Agora o processo é repetido na matriz $\overline{\boldsymbol{A}}^*(1)$. O pivô é $p(2)(i^*, j^*) = a(1,1) = 5$ e então $\boldsymbol{U}$ e $\boldsymbol{V}$ são dados por:

$$\boldsymbol{U} = \begin{bmatrix} 4 & 5 \\ 6 & 2{,}5 \\ 8 & 0 \end{bmatrix}; \boldsymbol{V} = \begin{bmatrix} 0{,}25 & 0{,}5 & 1 \\ 1 & 0{,}8 & 0 \end{bmatrix}$$

Observe que as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$ crescem a cada iteração. Como dito anteriormente, isso ocorre, pois a cada iteração é calculado e adicionado uma coluna e uma linha em cada uma das matrizes, respectivamente. Definido $\boldsymbol{U}$ e $\boldsymbol{V}$, é calculado então $\bar{\boldsymbol{A}}(2) = \boldsymbol{UV}$.

$$\bar{\boldsymbol{A}}(2) = \begin{bmatrix} 4 & 5 \\ 6 & 2{,}5 \\ 8 & 0 \end{bmatrix} \begin{bmatrix} 0{,}25 & 0{,}5 & 1 \\ 1 & 0{,}8 & 0 \end{bmatrix} = \begin{bmatrix} 6 & 6 & 4 \\ 4 & 5 & 6 \\ 2 & 4 & 8 \end{bmatrix} = \begin{bmatrix} 4 & 5 \\ 6 & 2{,}5 \\ 8 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{8} & 0 \\ & \\ 0 & \frac{1}{5} \end{bmatrix} \begin{bmatrix} 2 & 4 & 8 \\ 5 & 4 & 0 \end{bmatrix}$$

Observe que $\bar{\boldsymbol{A}}(2) = \boldsymbol{A}$. Isso ocorre porque a aproximação cruzada consegue representar exatamente qualquer matriz em $i = k$ iterações, onde k é o posto da matriz. O produto $\boldsymbol{UV}$ foi também representado como produto de três matrizes com objetivo de relembrar que a aproximação cruzada é uma decomposição esqueleto que tem forma $\boldsymbol{A} \cong \boldsymbol{C}\widehat{\boldsymbol{A}}^{-1}\boldsymbol{R}$, onde $\boldsymbol{C}$ é a matriz $\boldsymbol{U}$ a qual corresponde ao conjunto das diferentes colunas de cada pivô $p(i)$ da matriz $\bar{\boldsymbol{A}}^{*}(i)$, $\widehat{\boldsymbol{A}}^{-1}$ é a matriz inversa da matriz que contém em sua diagonal os pivôs, e por fim $\boldsymbol{R}$ é a matriz que contém o conjunto das diferentes linhas de cada pivô $p(i)$ da matriz $\bar{\boldsymbol{A}}^{*}(i)$. É interessante notar que, diferente da decomposição SVD, os componentes da matriz diagonal $\widehat{\boldsymbol{A}}^{-1}$ não aparecem de forma decrescente, mas sim da forma contrária, crescente.

O problema da aproximação cruzada é a necessidade do cálculo é armazenamento de toda matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$ para sua aplicação. Isso representa uma complexidade $O(N^2)$ na memória, tornando inviável a sua aplicação na maioria dos problemas de álgebra linear de larga escala.

3.2.2 APROXIMAÇÃO CRUZADA PIVÔTADA

A aproximação cruzada pivôtada utiliza como pivô não o elemento de maior valor absoluto da matriz, mas o de maior valor absoluto de uma linha ou coluna qualquer previamente escolhida. Deste modo, não é preciso calcular e armazenar toda a matriz, mas somente as linhas e colunas utilizadas, assim apresentando O($n_{int}N$) no uso da memória, onde n_{int} é número de iterações, complexidade a qual é quase linear para matrizes grandes de pequeno posto.

A primeira iteração do processo não é muito diferente da aproximação cruzada não pivôtada, mas muda bastante da segunda iteração em diante. Assim, o algoritmo pode ser resumido então da seguinte forma:

- Primeira Iteração

1. Calcular uma linha $\boldsymbol{A}(i,:)$ qualquer.
2. Achar o pivô $p(1)$ (i^*, j^*);
3. Calcular a coluna $\boldsymbol{A}(:, j^*)$ do pivô;
4. Definir $\boldsymbol{U}(:,1)$ = $\boldsymbol{A}(:, j^*)$ e $\boldsymbol{V}(1,:)$ tal que $v_{i,j} = \frac{a_{i,j}}{p(1)} = \frac{1}{p(1)}\boldsymbol{A}(i,:)$
5. Montar as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$
6. Calcular $\overline{\boldsymbol{A}}(i) = \boldsymbol{UV}$, onde $\boldsymbol{U} \in \mathbb{R}^{mxi}$ e $\mathbf{V} \in \mathbb{R}^{ixn}$;

- i-ésima iteração

1. Achar o subpivô (i^{**}, j^{**}) na coluna $\boldsymbol{A}(:, j^*)$, onde $i^{**} \neq i^*$;
2. Calcular a linha $\boldsymbol{A}(i^{**},:)$ do subpivô;
3. Achar o subpivô (i^{***}, j^{***}) na linha $\boldsymbol{A}(i^{**},:)$, onde $j^{***} \neq j^{**}$;
4. Calcular a coluna $\boldsymbol{A}(:, j^{***})$ do pivô;
5. Definir $\boldsymbol{U}(:, i) = \boldsymbol{A}(:, j^{***}) - \overline{\boldsymbol{A}}(i-1)(:, j^{***})$
6. Definir $\boldsymbol{V}'(i,:) = \boldsymbol{A}(i^{**},:) - \overline{\boldsymbol{A}}(i-1)(i^{**},:)$
7. Definir o pivô $p(i)(i^*, j^*)$ como o maior elemento em módulo de $\boldsymbol{V}'(i,:)$
8. Definir $\boldsymbol{V}(i,:) = \frac{\boldsymbol{V}'(i,:)}{\boldsymbol{p}(i)}$
9. Montar as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$
10. Calcular $\overline{\boldsymbol{A}}(i) = \boldsymbol{UV}$, onde $\boldsymbol{U} \in \mathbb{R}^{mxi}$ e $\mathbf{V} \in \mathbb{R}^{ixn}$;
11. Repetir processo.

Considere a mesma matriz $\boldsymbol{A}$ utilizada como exemplo na seção anterior. Suponha que somente a segunda linha tenha sido calculada anteriormente.

$$\boldsymbol{A}(2,:) = [4 \quad 5 \quad 6]$$

O pivô nesse caso é o elemento $p(1)(i^*, j^*) = a(2,3) = 6$. É calculada então a coluna referente aquele elemento.

$$\boldsymbol{A}(:,3) = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}$$

Agora é possível definir as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$.

$$\boldsymbol{U}(:,1) = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix}; \boldsymbol{V} = \frac{1}{6}[4 \quad 5 \quad 6] \rightarrow \boldsymbol{V} = \begin{bmatrix} \frac{4}{6} & \frac{5}{6} & 1 \end{bmatrix}$$

Definido $\boldsymbol{U}$ e $\boldsymbol{V}$, é possível então calcular $\overline{\boldsymbol{A}}(1) = \boldsymbol{UV}$.

$$\overline{\boldsymbol{A}}(1) = \begin{bmatrix} 4 \\ 6 \\ 8 \end{bmatrix} \begin{bmatrix} \frac{4}{6} & \frac{5}{6} & 1 \end{bmatrix} = \begin{bmatrix} \frac{8}{3} & \frac{10}{3} & 4 \\ 4 & 5 & 6 \\ \frac{16}{3} & \frac{20}{3} & 8 \end{bmatrix}$$

Como não se conhece a matriz $\boldsymbol{A}$, não é possível calcular $\overline{\boldsymbol{A}}^{*}(1) = \mathbf{A} - \overline{\boldsymbol{A}}(1)$**.** O que se faz é definir um subpivô como o elemento de maior valor em módulo da coluna calculada. Caso o subpivô coincida com o pivô anterior, é escolhido então o segundo elemento de maior valor em módulo, pois os pivôs não podem ser elementos repetidos. No caso desse exemplo, a coluna calculada foi a coluna $\boldsymbol{A}(:,3)$ o maior elemento dessa coluna é $a(3,3) = 8$, o qual não coincide com o pivô anterior. Escolhido o pivô é calculada a linha referente a ele.

$$\boldsymbol{A}(3,:) = [2 \quad 4 \quad 8]$$

A partir da linha calculada é definido outro subpivô, a escolha segue os mesmos requisitos aplicados ao subpivô anterior. Deste modo, o subpivô é o elemento $a(2,2) = 4$, pois embora $a(3,3) = 8$ seja o elemento de maior valor em módulo, ele coincide como o subpivô anterior. Agora, é calculada a coluna correspondente àquele pivô.

$$\boldsymbol{A}(:,2) = \begin{bmatrix} 6 \\ 5 \\ 4 \end{bmatrix}$$

A nova coluna adicionada a matriz $\boldsymbol{U}$ é definida então, como a diferença entre a nova coluna calculada $\boldsymbol{A}(:,2)$ e a sua correspondente na matriz $\overline{\boldsymbol{A}}(1)$.

$$\boldsymbol{U}(:,2) = \boldsymbol{A}(:,2) - \overline{\boldsymbol{A}}(1)(:,2) = \begin{bmatrix} 6 \\ 5 \\ 4 \end{bmatrix} - \begin{bmatrix} \frac{10}{3} \\ 5 \\ \frac{20}{3} \end{bmatrix} = \begin{bmatrix} \frac{8}{3} \\ 0 \\ -\frac{8}{3} \end{bmatrix}$$

Para definir a nova linha adicionada a matriz $\boldsymbol{V}$, além de tomar a diferença entre a nova linha $\boldsymbol{A}(2,:)$ e a sua correspondentes na matriz $\overline{\boldsymbol{A}}(1)$, é preciso dividir a matriz pelo pivô, o qual é o mesmo elemento do subpivô anterior, ou seja, o elemento $a(2,3)$ mas após ser efetuado a subtração entre as linhas.

$$\boldsymbol{V}'(2,:) = \boldsymbol{A}(3,:) - \overline{\boldsymbol{A}}\,(1)(3,:) = [2 \quad 4 \quad 8] - \left[\frac{16}{3} \quad \frac{20}{3} \quad 8\right] = \left[-\frac{10}{3} \quad -\frac{8}{3} \quad 0\right]$$

$$\boldsymbol{V}(:,2) = -\frac{8}{3}\boldsymbol{V}'(:2) = \begin{bmatrix}\frac{5}{4} & 1 & 0\end{bmatrix}$$

Definido $\boldsymbol{U}$ e $\boldsymbol{V}$, é calculado então $\overline{\boldsymbol{A}}(2) = \mathbf{UV}$.

$$\overline{\boldsymbol{A}}(2) = \begin{bmatrix}4 & \frac{8}{3}\\ 6 & 0\\ 8 & -\frac{8}{3}\end{bmatrix}\begin{bmatrix}\frac{4}{6} & \frac{5}{6} & 1\\ \frac{5}{4} & 1 & 0\end{bmatrix} = \begin{bmatrix}6 & 6 & 4\\ 4 & 5 & 6\\ 2 & 4 & 8\end{bmatrix} = \begin{bmatrix}4 & \frac{8}{3}\\ 6 & 0\\ 8 & -\frac{8}{3}\end{bmatrix}\begin{bmatrix}\frac{1}{6} & 0\\ 0 & -\frac{8}{3}\end{bmatrix}\begin{bmatrix}4 & 5 & 6\\ -\frac{10}{3} & -\frac{8}{3} & 0\end{bmatrix}$$

O resultado obtido é mais uma vez exato, isso, pois como dito anteriormente, a aproximação cruzada, pivôtada ou não, consegue representar exatamente qualquer matriz com posto k em k iterações. A diferença aparece quando a matriz é aproximada por uma de menor posto.

Observe que na aproximação cruzada pivôtada foi necessário calcular somente duas linhas e duas colunas, ao invés de toda a matriz. Embora para o exemplo desenvolvido tal fato não faça grande diferença, para matrizes de maior dimensão e de pequeno posto essa vantagem é significativa.

A grande desvantagem desse método é que para uma matriz qualquer $\boldsymbol{A} \in \mathbb{R}^{nxn}$ não é possível analisar o quão bem a i-ésima iteração representa essa matriz, pois nem o posto e nem a própria matriz é conhecida.

Outra importante observação é que embora ambas as decomposições representem exatamente a matriz $\boldsymbol{A}$, elas resultam em diferentes decomposições esqueleto. De fato, não considerando nenhuma estratégia para escolha dos pivôs, existem $n^2(n-1^2)(n-2^2)\ldots 2^2 1^2$ diferentes decomposições esqueleto possíveis para uma matriz qualquer $\boldsymbol{A} \in \mathbb{R}^{nxn}$. Isso, pois na primeira iteração qualquer entrada da matriz pode ser escolhida como pivô, ou seja, há n^2 possibilidades diferentes. Na segunda iteração pode ser escolhido qualquer elemento da matriz que não pertença à linha ou coluna do pivô anterior, ou seja, há $(n-1)^2$ possibilidades diferentes, e

assim por diante. Para termos uma noção de grandeza, uma matriz $\boldsymbol{A} \in \mathbb{R}^{10x10}$ apresenta $1{,}32x10^{13}$ representações diferentes possíveis. Deste modo, embora a seleção dos pivôs não faça diferença quando se aproxima uma matriz de posto k executando k iterações, o critério de seleção deles é de extrema importância quanto se aproxima a matriz por uma de baixo posto, pois o objetivo é conseguir uma aproximação da matriz com a menor quantidade possível de iterações, ou seja, com menor posto possível, mas que apresente um erro $\|\boldsymbol{A} - \bar{\boldsymbol{A}}(i)\|$ dentro de uma faixa tolerável.

3.2.3 APROXIMAÇÃO CRUZADA ADAPTATIVA (ACA)

Adaptação cruzada adaptativa (ACA) apresenta um modo indireto de medir a precisão que uma matriz de baixo posto k obtida pela aproximação cruzada pivôtada representa a matriz original. Assim, o número de iterações executadas pelo processo é uma função da precisão ε desejada para a representação de baixo posto. Uma boa forma é estimar o erro $\|\boldsymbol{A} - \bar{\boldsymbol{A}}(i)\|$.

$$\|\boldsymbol{A} - \bar{\boldsymbol{A}}(i)\| \leq \|\boldsymbol{A} - \bar{\boldsymbol{A}}(i-1)\|$$

$$\|\boldsymbol{A} - \bar{\boldsymbol{A}}(i-1)\| \cong \|\bar{\boldsymbol{A}}(i) - \bar{\boldsymbol{A}}(i-1)\| = \|a_i b_i\|.$$

onde a_i é a coluna do matriz $\boldsymbol{U}$ e b_i é a linha da matriz $\boldsymbol{V}$ calculada na i-ésima iteração.

Assim, o erro relativo obtido na k iteração pode ser estimado utilizando a norma de Frobenius como:

$$\in_{rel} = \frac{\|a_i b_i\|}{\|\bar{\boldsymbol{A}}(i-1)\|}$$

onde a norma de Frobenius é definida como:

$$\|A\| = \left(\sum_{i=0}^{m} \sum_{j=0}^{n} \left(a_{i,j}\right)^2 \right)^{1/2}$$

Considere o exemplo desenvolvido no tópico anterior. A coluna da matriz $\boldsymbol{U}$ calculada na segunda iteração foi $\boldsymbol{U}(:,2)$ e a linha calculada da matriz $\boldsymbol{V}$ foi $\boldsymbol{V}(2,:)$.

Assim temos:

$$a_2 = \boldsymbol{U}(:,2) = \begin{bmatrix} \frac{8}{3} \\ 0 \\ -\frac{8}{3} \end{bmatrix}$$

$$b_2 = \boldsymbol{V}(2,:) = \begin{bmatrix} \frac{5}{4} & 1 & 0 \end{bmatrix}$$

Podemos agora calcular a norma de Frobenius $\|a_k b_k\|$.

$$a_2 b_2 = \begin{bmatrix} \frac{8}{3} \\ 0 \\ -\frac{8}{3} \end{bmatrix} \begin{bmatrix} \frac{5}{4} & 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{10}{3} & \frac{8}{3} & 0 \\ 0 & 0 & 0 \\ -\frac{10}{3} & -\frac{8}{3} & 0 \end{bmatrix}$$

$$\|a_2 b_2\| = \sqrt{2\left(\frac{8}{3}\right)^2 + 2\left(\frac{10}{3}\right)^2} = 6{,}037$$

É preciso agora calcular a norma de Frobenius $\|\overline{\boldsymbol{A}}(i-1)\| = \|\overline{\boldsymbol{A}}(1)\|$.

$$\overline{\boldsymbol{A}}(1) = \begin{bmatrix} \frac{8}{3} & \frac{10}{3} & 4 \\ 4 & 5 & 6 \\ \frac{16}{3} & \frac{20}{3} & 8 \end{bmatrix}$$

$$\|\overline{\boldsymbol{A}}(1)\| = \sqrt{\left(\frac{8}{3}\right)^2 + \left(\frac{10}{3}\right)^2 + 2(4)^2 + (5)^2 + (6)^2 + \left(\frac{16}{3}\right)^2 + \left(\frac{20}{3}\right)^2 + (8)^2} = 15.752$$

É por fim temos que

$$\in_{rel} = \frac{\|a_2 b_2\|}{\|\overline{\boldsymbol{A}}(1)\|} = \frac{6{,}037}{15{,}752} = 0{,}383$$

Para qualquer algoritmo que deseja uma boa representação de baixo posto um erro relativo de 38% é um erro extremamente alto. Assim, é necessária outra iteração. No nosso caso, a próxima iteração resulta na matriz exata, e como consequência, apresenta erro relativo nulo. O alto valor do erro relativo indica dois fatos: o primeiro é que a segunda iteração ainda provoca grandes modificações na matriz $\overline{\boldsymbol{A}}(1)$ de forma que $\overline{\boldsymbol{A}}(2)$ e muito diferente de $\overline{\boldsymbol{A}}(1)$. O segundo fato é que tentar aproximar matrizes

de pequenas dimensões por matrizes de baixo posto é ineficiente, podendo levar a grandes erros. Assim, o método da decomposição esqueleto e o método ACA são eficientes e vantajosos somente para matrizes de grandes dimensões, permitindo uma redução significativa do custo de armazenamento de uma matriz e evitando que seja necessário o cálculo direto da mesma.

O método ACA não altera o algoritmo descrito anteriormente, somente adiciona mais algumas etapas. Observe que ε determina a precisão e deve ser previamente definido.

- i-ésima iteração:

1. Definir $a_i = \boldsymbol{U}(:, i)e\ b_i = \boldsymbol{V}(i,:)$
2. Avaliar $\|a_i b_i\| < \varepsilon \|\overline{\boldsymbol{A}}(i-1)\|$
3. Se verdadeiro parar.
4. Se falso repetir processo.

3.2.4 APROXIMAÇÃO CRUZADA ADAPTATIVA MODIFICADA (ACA+)

O método ACA apresenta uma grande limitação quando utilizado em matrizes esparsas, ou seja, que contém um grande número de valores nulos. Isso, pois suponha que foi calculada previamente somente a primeira linha da matriz e, no entanto todos os elementos pertencentes aquela linhas são nulos. Como então definir um pivô? Qualquer escolha feita pelo algoritmo convergiria a uma aproximação com alto erro, erro o qual pode não ser observável pelo estimador utilizado no método ACA.

O erro tem origem na obtenção da matriz $\boldsymbol{V}$ a qual envolve a divisão pelo pivô, o qual é nulo, ou de forma diferente, mas equivalente, na multiplicação pela matriz $\widehat{\boldsymbol{A}}^{-1}$ na decomposição esqueleto $\boldsymbol{C}\widehat{\boldsymbol{A}}^{-1}\boldsymbol{R}$, pois $\widehat{\boldsymbol{A}}^{-1}$ é a matriz inversa da matriz diagonal que contém os pivôs. No entanto, se um pivô é nulo, o determinante dessa matriz é também nulo e a matriz não é inversível.

Com o objetivo de evitar a ocorrência desse erro, o método ACA+ [12] define uma linha e uma coluna de referência antes de calcular as entradas utilizadas na aproximação. Escolhido uma linha qualquer de referência, a coluna de referência é definida pela menor entrada em valor absoluto da linha de referência. É importante lembrar que o contrário pode também sempre ser feito, ou seja, escolher primeiro uma

coluna e então definir a linha. Essa escolha garante que um bloco não nulo seja intersectado por uma linha ou coluna de referência, o que previne o uso de um pivô nulo.

O algoritmo para a primeira iteração pode ser definido da seguinte forma:

1. Calcular a linha auxiliar $\boldsymbol{A}(i,:)$;
2. Achar o pivô auxiliar $px(1)(i',j')$ de menor valor em módulo;
3. Achar o subpivô $ps(1)(i^*,j^*)$ de maior valor em módulo;
4. Calcular a coluna auxiliar $\boldsymbol{A}(:,j')$;
5. Avaliar $ps(1) > px(1)$;
6. Se verdadeiro:
7. Calcular a coluna $\boldsymbol{A}(:,j^*)$;
8. Achar o pivô $p(1)(i^{**},j^{**})$ na coluna $\boldsymbol{A}(:,j^*)$;
9. Calcular a linha $\boldsymbol{A}(i^{**},:)$;
10. Definir $\boldsymbol{U}(:,1) = \boldsymbol{A}(:,j^*)$ e $\boldsymbol{V}(1,:)$ tal que $v_{i,j} = \frac{a_{i^{**},j}}{p(1)} = \frac{1}{p(1)}\boldsymbol{A}(i^{**},:)$;
11. Se falso:
12. Achar o subpivô $ps(2)(i^{**},j^{**})$ na coluna auxiliar $\boldsymbol{A}(:,j')$;
13. Calcular a linha $\boldsymbol{A}(i^{**},:)$;
14. Achar o pivô $p(1)(i^{***},j^{***})$ na linha $\boldsymbol{A}(i^{**},:)$;
15. Calcular a coluna $\boldsymbol{A}(:,j^{***})$;
16. Fazer $\boldsymbol{U}(:,1) = \boldsymbol{A}(:,j^{***})$ e $\boldsymbol{V}(1,:)$ tal que $v_{i,j} = \frac{a_{i^{***},j}}{p(1)} = \frac{1}{p(1)}\boldsymbol{A}(i^{***},:)$
17. Montar as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$
18. Calcular $\overline{\boldsymbol{A}}\,(1) = \boldsymbol{UV}$, onde $\boldsymbol{U} \in \mathbb{R}^{mx1}$ e $\boldsymbol{V} \in \mathbb{R}^{1xn}$;

Considere a matriz $\boldsymbol{A}$ dada a baixo. Essa matriz apresenta diversos valores nulos, incluindo toda sua primeira linha e quarta coluna.

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 5 & 8 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 6 & 0 & 0 \end{bmatrix}$$

Suponha que somente a primeira linha tenha sido calculada, essa é definida como a linha auxiliar.

$$\boldsymbol{A}(\mathbf{1},:) = [0 \quad 0 \quad 0 \quad 0]$$

O pivô auxiliar é o elemento de menor valor em módulo, neste caso qualquer um dos valores é possível, para esse exemplo considere $px(1)(i', j') = a(1,4) = 0$. Como todos os valores são iguais, o mesmo ocorre para o subpivô. Assim considere $ps(1)(i^*, j^*) = a(1,1) = 0$.

A coluna auxiliar é então calculada como a coluna do pivô auxiliar.

$$\boldsymbol{A}(:,\mathbf{4}) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Em seguida é avaliado a condição $ps(1) > px(1)$. Neste caso a condição é falsa pois $ps(1) = 0 = px(1)$.

A utilização de uma linha e de uma coluna auxiliar tem como objetivo evitar o aparecimento de um pivô nulo. Observe que a condição $ps(1) > px(1)$ só será falsa quando $ps(1) = px(1)$, isso, pois eles são respectivamente o maior é menor valor em módulo na linha auxiliar $\boldsymbol{A}(i,:)$. Assim, $ps(1) = px(1)$ se e somente se todas as entradas da linha auxiliar forem iguais, o que inclui o caso que se quer evitar, onde todas elas são nulas.

No caso em que a condição é falsa e definido um novo subpivô na coluna auxiliar, novamente todos os valores são iguais. Assim, considere $ps(2)(i^{**}, j^{**}) = a(4,4) = 0$. A linha desse novo subpivô é calculada.

$$\boldsymbol{A}(\mathbf{4},:) = [0 \quad 6 \quad 0 \quad 0]$$

Em seguida, é definido um pivô $p(1)(i^{***}, j^{***}) = a(4,2) = 6$ e sua coluna é calculada.

$$\boldsymbol{A}(:,\mathbf{2}) = \begin{bmatrix} 0 \\ 8 \\ 0 \\ 6 \end{bmatrix}$$

Podemos definir então $\boldsymbol{U}(:,\mathbf{1}) = \boldsymbol{A}(:,\mathbf{2})$ e $\boldsymbol{V}(:,\mathbf{1}) = \frac{\mathbf{1}}{\mathbf{6}}\boldsymbol{A}(\mathbf{4},:)$**,** e a aproximação pode ser calculada por $\overline{\boldsymbol{A}}\,(1) = \boldsymbol{UV}$.

Agora, voltando ao início do problema, suponha que somente a terceira linha tenha sido calculada, e não a primeira, nesse caso a linha auxiliar é dada por:

$$\boldsymbol{A}(\mathbf{4},:) = [0 \quad 6 \quad 0 \quad 0]$$

Novamente por existir valores iguais, há diversas opções para a escolha de um pivô auxiliar, considere então $px(1)(i', j') = a(4,4) = 0$. Já o subpivô e dado por $ps(1)(i^*, j^*) = a(4,2) = 6$. Em seguida, é avaliado a condição $ps(1) > px(1)$, neste caso a condição é verdadeira pois $ps(1) = 6 > 0 = px(1)$.

No caso em que a condição seja verdadeira é calculado a coluna do subpivô.

$$\boldsymbol{A}(:,\mathbf{2}) = \begin{bmatrix} 0 \\ 8 \\ 0 \\ 6 \end{bmatrix}$$

É definido então um pivô $p(1)(i^{**}, j^{**}) = a(2,2) = 8$ e sua linha é calculada.

$$\boldsymbol{A}(\mathbf{2},:) = [5 \quad 8 \quad 0 \quad 0]$$

Agora, podemos definir $\boldsymbol{U}(:,\mathbf{1}) = \boldsymbol{A}(:,\mathbf{2})$ e $\boldsymbol{V}(:,\mathbf{1}) = \frac{\mathbf{1}}{\mathbf{8}}\boldsymbol{A}(\mathbf{2},:)$ e a aproximação pode novamente ser calculada por $\overline{\boldsymbol{A}}\,(1) = \boldsymbol{UV}$.

Observe que a utilização de linhas e coluna auxiliares evita o aparecimento de pivôs nulos, o qual é o objetivo do método ACA+.

Em geral, não é necessário calcular novas linhas ou colunas de referência a cada iteração. No caso em que condição $ps(1) > px(1)$ é atendida, será necessária calcular uma nova linha de referência somente quando $\boldsymbol{A}(i,:) = \boldsymbol{A}(i^{**},:)$, o que ocorre quando $ps(1) = p(1)$, ou seja quando o maior elemento da linha auxiliar é também o maior elemento de sua coluna. Nesse caso não é necessária calcular uma nova coluna. No caso $ps(1) = px(1)$, ou seja, quando a condição não e satisfeita, é preciso calcular uma nova linha e coluna auxiliar quando $px(1) = ps(2)$, ou seja, quando $px(1)$ é o maior elemento na coluna auxiliar ou calcular somente uma nova coluna quando

$ps(2)(i^{**}, j^{**}) = p(1)(i^{***}, j^{***})$. Isto ocorre quando o maior elemento em módulo da linha $\boldsymbol{A}(i^{**}, :)$ pertence a coluna auxiliar.

Não há provas da convergência do método ACA+. O método ainda não é capaz de obter uma aproximação precisa para qualquer matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$, mas se prova extremamente eficiente para as matrizes obtidas de problemas analisados pelo MEC. O problema de convergência do ACA+ está em matrizes extremamente esparsas, de forma que se escolhido duas linhas e duas colunas aleatoriamente, elas tenham todos os elementos nulos. Tal fato, no entanto, não é observado em problemas descrito pelo MEC, o qual, embora possa apresentar matrizes com blocos esparsos, não apresentam tal comportamento.

3.3 MATRIZES HIERÁRQUICAS

Matrizes hierárquicas, ou matrizes-*H*, são uma forma de representação esparsa dos dados de matrizes densas. Esse tipo de matriz não é esparsa no sentido de conter grande quantidade de coeficientes nulos, mas no sentido de armazenar os coeficientes de uma matriz densa com apenas uma pequena quantidade de dados. O método consiste em particionar de maneira hierárquica uma dada matriz em blocos de matrizes de menor ordem, de forma que cada bloco possa ser aproximado por uma matriz de menor posto. Qualquer matriz pode ser aproximada por matrizes hierárquicas com certa precisão.

Como essa forma de representação é diferente da representação tradicional, as operações básicas realizadas com matrizes podem ser efetuadas de forma aproximada, com um algoritmo de complexidade inferior quase linear. Essa aproximação possibilita representar matrizes com complexidade linear $O(N)$ ou logarítmico-linear $O(Nlog^{q}N)$, diminuindo a necessidade de armazenamento, os requisitos de memória, e tornando a resolução do problema mais rápida.

A aplicação do método requer um conjunto de três processos diferentes. O primeiro, responsável pelo particionamento da matriz de dados em blocos de matrizes de menor dimensão. O segundo é responsável por organizar essas matrizes de forma hierárquica. E o último, avalia esses blocos e os agrupa ou não de acordo com uma condição de admissibilidade estabelecida. Essa condição de admissibilidade é tal que seja possível construir os maiores blocos de matrizes de baixo posto possíveis.

3.3.1 ANÁLISE DE COMPONENTES PRINCIPAIS (PCA)

A análise de componentes principais ou análise PCA (acrônimo em inglês de Principal Component Analysis) é um método estatístico que permite a transformação de sistema de coordenadas, rotacionando o mesmo na direção de maior variabilidade das características x_j de uma amostra de dados.

Considere o caso bidimensional, onde as características x_j analisadas são o par de coordenadas $x(x_1)$ e $y(x_2)$, ou seja, a posição de cada ponto da nuvem de pontos mostrada na Figura 9.

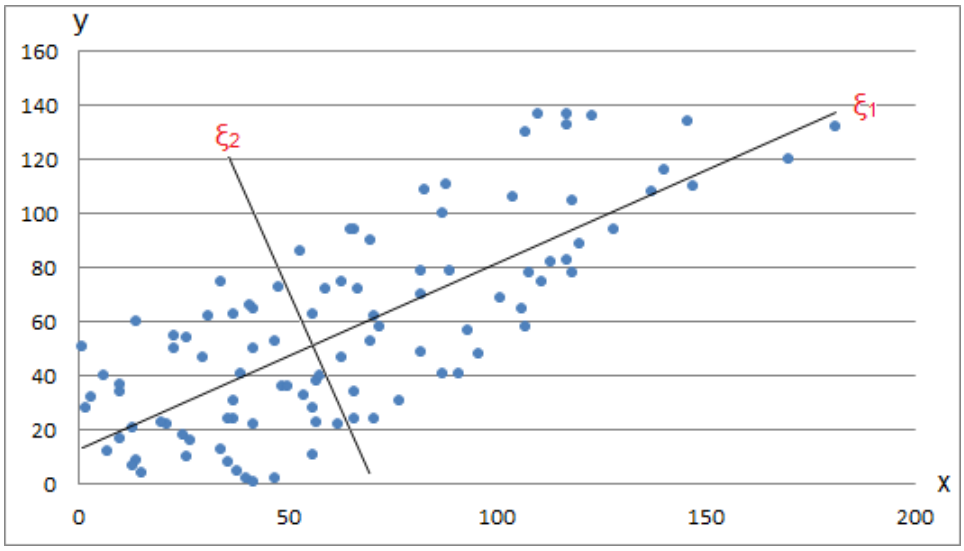

Figura 9 - Nuvem de pontos para análise PCA.

A aplicação do método possibilita a rotação do eixo de coordenadas x e y para o eixo de coordenadas ξ_1 e ξ_2, eixos os quais apresentam a maior variância dos dados. O problema é então encontrar a transformação:

$$\begin{aligned}\xi_1 &= r_{11}x_1 + r_{12}x_2 + \cdots + r_{1(n-1)}x_{(n-1)} + r_{1n}x_n \\ \xi_2 &= r_{21}x_1 + r_{22}x_2 + \cdots + r_{2(n-1)}x_{(n-1)} + r_{2n}x_n \\ &\vdots \\ \xi_m &= r_{m1}x_1 + r_{m2}x_2 + \cdots + r_{m(n-1)}x_{(n-1)} + r_{mn}x_n\end{aligned}$$

ou seja:

$$\xi = \boldsymbol{R}x$$

que maximiza a covariância $\boldsymbol{\Sigma}(\xi)$. No entanto, existe uma identidade para matrizes de covariância que permite que a covariância $\boldsymbol{\Sigma}(\xi)$ seja escrita em função da covariância de x.

$$\boldsymbol{\Sigma}(\xi) = \boldsymbol{\Sigma}(\boldsymbol{R}x) = \boldsymbol{R}\Sigma(x)\boldsymbol{R}^T$$

Assim, o problema é maximizar:

$$\boldsymbol{R\Sigma}(x)\boldsymbol{R}^T$$

restrito a condição:

$$\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$$

Tal condição é uma característica de qualquer matriz de transformação. Observe, por exemplo, a matriz genérica para a rotação no sentido anti-horário de um vetor no espaço bidimensional.

$$\boldsymbol{R} = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}$$

$$\boldsymbol{R}\boldsymbol{R}^T = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix}\begin{bmatrix} cos(\theta) & sen(\theta) \\ -sen(\theta) & cos(\theta) \end{bmatrix}$$

$$\boldsymbol{R}\boldsymbol{R}^T = \begin{bmatrix} cos(\theta)^2 + sen(\theta)^2 & cos(\theta)\,sen(\theta) - cos(\theta)\,sen(\theta) \\ sen(\theta)cos(\theta) - sen(\theta)\,cos(\theta) & sen(\theta)^2 + cos(\theta)^2 \end{bmatrix}$$

$$\boldsymbol{R}\boldsymbol{R}^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Para a resolução do problema de otimização com restrição, é utilizado o método dos multiplicadores de Lagrange. O método diz, resumidamente, que o problema de otimização de uma função $f(x_1, x_2, \ldots, x_n)$, restrita as condições $g_i(x_1, x_2, \ldots, x_n) = c_i$ pode ser substituído pelo problema de otimização descrito pela função de Lagrange:

$$f\left(x_1, x_2, \ldots, x_n, \lambda_1, \lambda_2, \ldots, \lambda_m\right) = f(x, y) - \sum_{i=1}^{m} \lambda_i \left(C_i - g_1(x_1, x_2, \ldots, x_n)\right)$$

onde λ_i são os chamados multiplicadores de Lagrange. Assim, a função de Lagrange para o problema de maximização da covariância é dada por:

$$f = \boldsymbol{R\Sigma}(x)\boldsymbol{R}^T - \lambda(\boldsymbol{I} - \boldsymbol{R}\boldsymbol{R}^T)$$

e então:

$$\nabla(\boldsymbol{R\Sigma}(x)\boldsymbol{R}^T) - \lambda\nabla(\boldsymbol{I} - \boldsymbol{R}\boldsymbol{R}^T) = 0$$

$$\mathbb{I} - \lambda\mathbb{II} = 0;\ \mathbb{I} = \nabla(R\Sigma(x)R^T)\ e\ \mathbb{II} = \lambda\nabla(I - RR^T)$$

Os gradientes das equações $\mathbb{I}$ e $\mathbb{II}$ podem ser obtidos usando a relação mais geral [13]:

$$\nabla(\boldsymbol{BX}+b)^T\boldsymbol{C}(\boldsymbol{DX}+d)=\boldsymbol{B}^T\boldsymbol{C}(\boldsymbol{DX}+d)+\boldsymbol{D}^T\boldsymbol{C}^T(\boldsymbol{BX}+b)$$

onde $\boldsymbol{X}=\boldsymbol{R}$, $\boldsymbol{C}=\boldsymbol{I}$, $\boldsymbol{D}=\boldsymbol{R}^T$ e $b=d=0$.

Para a equação $\mathbb{I}$ é definido $\boldsymbol{B}=\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T$ e então:

$$\nabla(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T\boldsymbol{R})^T\boldsymbol{I}(\boldsymbol{R}^T\boldsymbol{R})=(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T)^T\boldsymbol{I}(\boldsymbol{R}^T\boldsymbol{R})+(\boldsymbol{R}^T)^T\boldsymbol{I}^T(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T\boldsymbol{R})$$

$$\nabla(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T\boldsymbol{R})^T\boldsymbol{I}(\boldsymbol{I})=(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T)^T\boldsymbol{I}(\boldsymbol{I})+\boldsymbol{R}\boldsymbol{I}(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T\boldsymbol{R});\ \boldsymbol{R}^T\boldsymbol{R}=\boldsymbol{I}$$

$$\nabla(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T\boldsymbol{R})^T=(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)^T)^T+(\boldsymbol{R}\boldsymbol{R}^T)\boldsymbol{\Sigma}(x)^T\boldsymbol{R}$$

$$\nabla\boldsymbol{R}\boldsymbol{\Sigma}(x)\boldsymbol{R}^T=\boldsymbol{\Sigma}(x)\boldsymbol{R}+\boldsymbol{\Sigma}(x)^T\boldsymbol{R};\ \boldsymbol{\Sigma}(x)=\boldsymbol{\Sigma}(x)^T$$

$$\mathbb{I}\rightarrow\nabla\boldsymbol{R}\boldsymbol{\Sigma}(x)\boldsymbol{R}^T=2\boldsymbol{\Sigma}(x)\boldsymbol{R}$$

A matriz de covariância é simétrica, pois a covariância entre x_i e x_j é igual à covariância entre x_j e x_i. A matriz transposta de uma matriz simétrica é igual à própria matriz. Assim, a soma da matriz covariância com sua transposta é simplesmente duas vezes a matriz de covariância.

Para a equação $\mathbb{II}$ é definido $\boldsymbol{B}=\boldsymbol{R}^T$**,** e então, procede-se de forma análoga a anterior. Observe que $\boldsymbol{B}$ em $\mathbb{I}$ e em $\mathbb{II}$ se difere somente por $\boldsymbol{\Sigma}(x)^T$. Assim, obtemos:

$$\mathbb{II}\rightarrow\nabla\boldsymbol{R}^T=2\boldsymbol{R}$$

e então:

$$\mathbb{I}-\lambda\,\mathbb{II}=0$$

$$2\boldsymbol{\Sigma}(x)\boldsymbol{R}-2\lambda\boldsymbol{R}=\boldsymbol{0}$$

$$\boldsymbol{\Sigma}(x)\boldsymbol{R}=\lambda\boldsymbol{R}$$

Observe que esse problema é um problema de autovetores e autovalores. Outra conclusão importante pode ser observada multiplicando ambos os lados da equação anterior por $\boldsymbol{R}^T$ a esquerda:

$$\boldsymbol{R}^T\boldsymbol{\Sigma}(x)\boldsymbol{R}=\lambda\boldsymbol{R}^T\boldsymbol{R}$$

$$(\boldsymbol{R}^T\boldsymbol{\Sigma}(x)\boldsymbol{R})^T=\lambda\boldsymbol{I}^T$$

$$\boldsymbol{R}\boldsymbol{\Sigma}(x)\boldsymbol{R}^T=\lambda\boldsymbol{I};\ \boldsymbol{\Sigma}(\xi)=\boldsymbol{R}\Sigma(x)\boldsymbol{R}^T$$

$$\boldsymbol{\Sigma}(\xi)=\lambda\boldsymbol{I}$$

O problema de maximizar a covariância $\boldsymbol{\Sigma}(\xi)$ não somente é um problema de autovetores e autovalores, mas seus autovalores são a própria variância. Assim o maior autovalor está associado à direção de maior variância, e seu autovetor é a nova direção do eixo coordenado ξ_1. O segundo autovalor, está então associado à segunda direção de maior variância é seu autovetor é a nova direção do eixo coordenado ξ_2 e assim por diante, onde o autovalor λ_i está associado à direção de i-ésima maior variância ξ_i.

O algoritmo para a execução da análise de componentes principais pode ser então descrito como:

1. Organizar os dados:
2. Características x_i: $i = 1 \rightarrow n$, com m observações representadas por um vetor coluna.
3. Dispor os vetores colunas em uma única matriz $\boldsymbol{X} \in \mathbb{R}^{mxn}$.
4. Calcular a média de cada característica $\bar{x}_i = \frac{1}{m}\sum_{m=1}^{m} x_{m,i}$;
5. Usar a média para centralizar os dados $\boldsymbol{B} = \boldsymbol{X} - [\bar{x}_1 \quad \dots \quad \bar{x}_n]_{mxn}$;
6. Calcular a matriz de covariância $\boldsymbol{\Sigma} = \frac{1}{m}\boldsymbol{B^T B}$;
7. Calcular autovetores e autovalores da matriz de covariância $\boldsymbol{\Sigma}$.

Quando a média é utilizada para centralizar os dados, os eixos coordenados são transladados para o centroide do conjunto de pontos, o que permite calcular a matriz de covariância sem considerar as médias de cada característica, pois uma vez centralizados os dados, a média é zero.

Após obter os autovalores e autovetores, é possível rotacionar o eixo coordenado obtendo a matriz rotação pelo produto $\boldsymbol{E^T V}$, onde $\boldsymbol{E}$ é a matriz que contém os vetores e_i em suas colunas, ou seja, os vetores unitários nas direções x_i do antigo sistema de coordenadas e $\boldsymbol{V}$ é a matriz que contém os vetores v_i em suas colunas, onde v_i é o autovetor unitário na direção ξ_i.

Para o exemplo bidimensional dado no início desse tópico, a rotação do sistema de coordenas na direção de maior variância resulta na Figura 10.

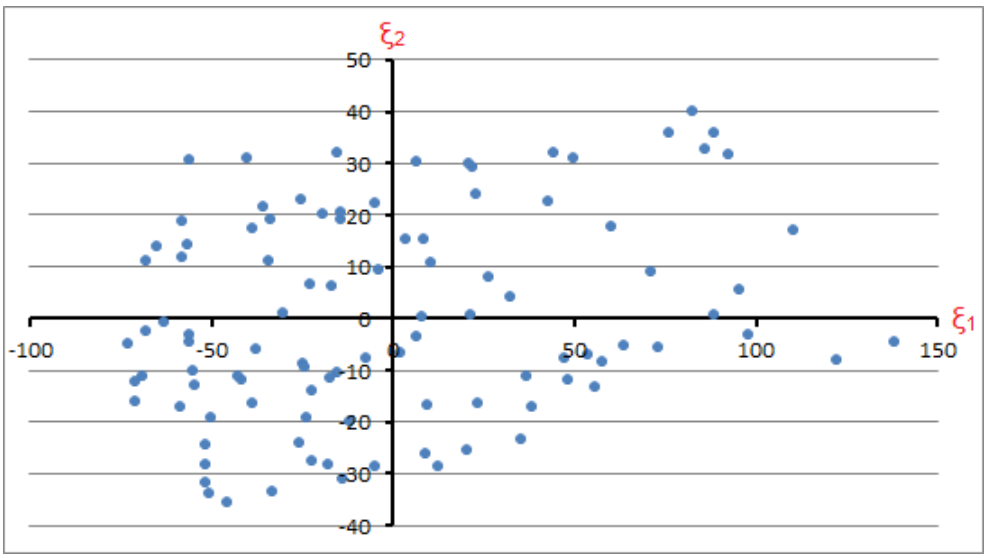

Figura 10 - Novos eixos coordenados após rotação na direção de maior variabilidade.

Na aplicação a qual esse trabalho é destinado, a análise dos componentes principais não tem como objetivo promover uma mudança dos eixos coordenados, mas sim fornecer uma direção para particionar os dados da matriz de forma a se produzir blocos uniformes. Isso, pois obtido o vetor unitário na direção ξ_1 a condição:

$$(x - \bar{x})v(\xi_1) > 0$$

onde x é o vetor posição de um ponto qualquer e $v(\xi_1)$ é o autovetor associado ao autovalor ξ_1, criaria dois blocos distintos a partir da nuvem de pontos inicial. Observe que a condição é nula quando os vetores são perpendiculares. Assim, essa condição dividiria a nuvem de pontos em dois blocos separados pelo eixo ξ_2, o qual é perpendicular a ξ_1. Esse particionamento produz dois blocos quase uniformes. De fato, nessa separação 45% dos pontos estão à direita do eixo ξ_2 e 55% à esquerda. O procedimento pode então ser repetido várias vezes até que se obtenha certa quantidade de blocos, quantidade essa definida pelo mínimo permitido de elementos em cada bloco.

Esse é o primeiro algoritmo necessário no processo de construção de uma matriz hierárquica, o qual é responsável por particionar os dados de uma matriz de forma uniforme.

3.3.2 ÀRVORE BINÁRIA

O agrupamento em árvore é um método utilizado para a estruturação de dados. Essa é uma representação utilizada em dados que apresentam de forma natural uma relação hierárquica. Uma árvore pode ser definida como uma coleção de nós interligados por arestas, onde cada nó, além de armazenar informação, contém o endereço dos nós que foram gerados a partir desse. O primeiro nó da árvore é chamado de raiz, os nós intermediários são chamados de nós internos, e os últimos nós, aqueles que não apresentam subdivisão, são chamados de folhas. Os nós obtidos a partir de outro nó são chamados de filhos e consequentemente o nó a qual deu origem a esses, de pai. Assim, um nó é uma folha somente se o mesmo não tem filhos.

Uma árvore é chamada de árvore binária se cada nó pode no máximo dar origem a dois filhos, mais específico ainda, uma árvore é chamada de árvore binária cheia (Figura 11) quanto todos os nós têm exatamente dois filhos, com exceção das folhas.

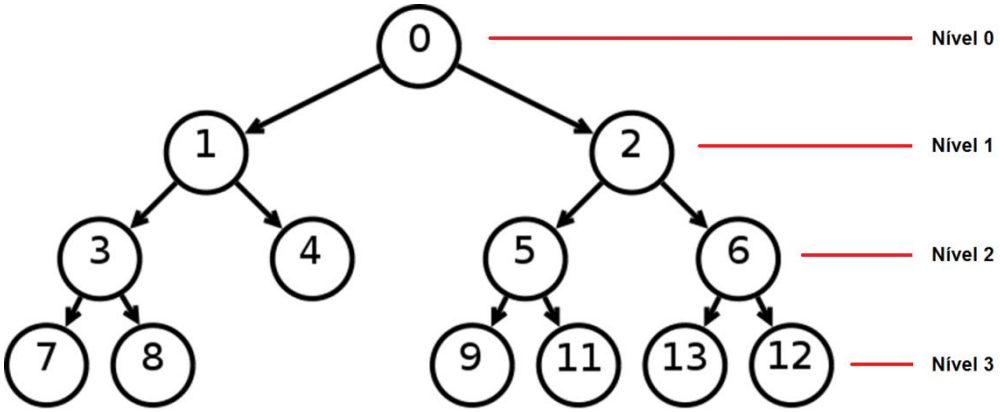

Figura 11 - Árvore binária cheia.

O número máximo de nós em uma árvore binária é dado por 2^L, onde L é o nível da árvore. A partir do nível é possível estimar quantas vezes as folhas de um nível qualquer são menores do que a raiz. Suponha que a quantidade de dados é distribuída uniformemente entre os nós. Assim, no primeiro nível cada nó apresenta metade dos dados. No próximo, os dados em cada nó são divididos na metade novamente e assim consecutivamente, tal que no nível L cada folha apresenta $\left(\frac{1}{2}\right)^L$ dos dados.

A altura h da árvore é definida como o número de arestas no maior caminho entre a raiz é uma folha. O tempo necessário para a realização de uma operação é

proporcional à altura da árvore. Assim, a complexidade de uma árvore é de ordem $O(h)$. A altura mínima de uma árvore é dada por $log_2\left(\frac{n+1}{2}\right)$ é a máxima por $n-1$, onde n é o numero de nós (Figura 12).

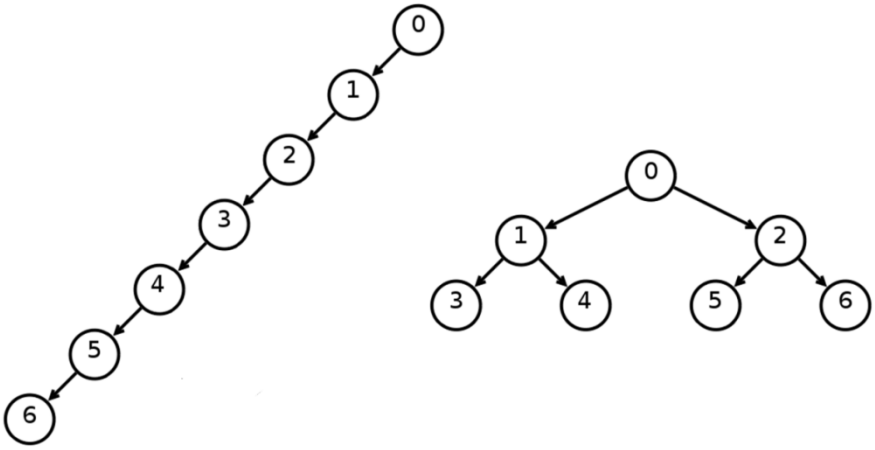

Figura 12 - Diferentes árvores com seis nós; com altura máxima (esquerda) e com altura mínima (direita).

A árvore binária é um processo recursivo, ou seja, cada redução pode ser feita por um mesmo algoritmo. No processo de construção das matrizes hierárquicas, a análise de componentes principais (PCA) é utilizada para gerar um agrupamento com cardinalidade balanceada, ou seja, subdividir os dados em conjuntos similares de forma que cada um contenha aproximadamente a mesma quantidade dados. Isso é feito para se obter uma árvore balanceada com a mínima altura possível. A árvore é chamada de balanceada quando os filhos de um mesmo nó contém aproximadamente a mesma quantidade de dados.

Formalmente uma árvore T_I é uma árvore para o conjunto I se [14]:

- A raiz é definida por I;
- Se $t \in T_I$ têm filhos, t é definido pela união de seus filhos;

$$t = \bigcup_{t' \in filhos(t)} t'$$

- Se $t \in T_I$ têm filhos, seus filhos são conjunto disjuntos.

$$t_1 \neq t_2 \rightarrow t_1 \cap t_2 \emptyset;\ \forall t_1, t_2 \in filhos(t)$$

Tal definição engloba somente um conjunto simples de dados I. Há então outra definição para conjunto que envolve bloco de dados (Figura 13).

Formalmente, uma árvore T_{IxJ} é uma árvore para o bloco formado por T_I e T_J se:

- Cada nó $b \in T_{IxJ}$ é um par $b = (t, s)$, onde $t \in T_I$ e $s \in T_J$;
- A raiz de T_{IxJ} é o par de raízes T_I e T_J;
- O conjunto de $b = (t, s)$ é o produto cartesiano do conjunto $t\ x\ s$;
- Se $b = (t, s) \in T_{IxJ}$ têm filhos, seus filhos são:
    - $\{(t', s) : t' \epsilon\ filhos(t)\}$ se filhos de $(s) = \emptyset$;
    - $\{(t, s') : s' \epsilon\ filhos(t)\}$ se filhos de $(t) = \emptyset$;
    - $\{(t', s') : t' \epsilon\ filhos(t)\ e\ s' \epsilon\ filhos(t)\}$ caso contrário

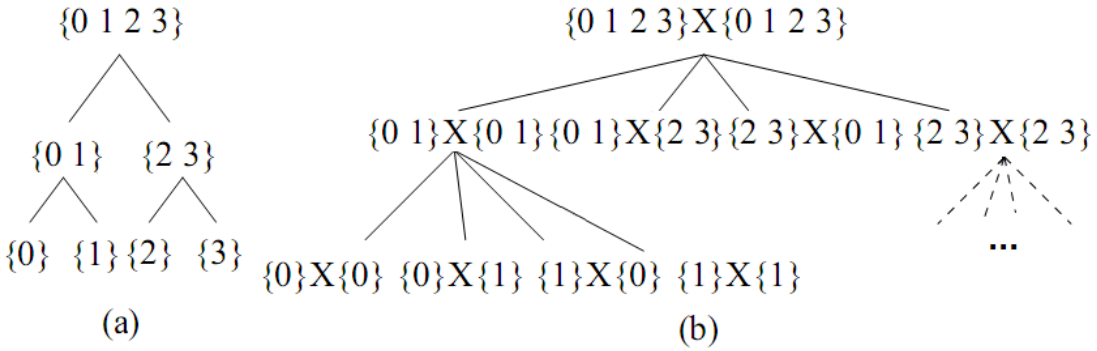

Figura 13 - Árvore T_I para o conjunto $I = \{1, 2, 3, 4\}$ (a) e Árvore T_{IxJ} para o bloco $\{1, 2, 3, 4\} X \{1, 2, 3, 4\}$ (b).

O produto cartesiano $t\ x\ s$ é o conjunto de todos os pares ordenados cujo primeiro termo pertence a t é o segundo a s. Por exemplo, o conjunto de cartas de baralho é o produto cartesiano entre os quatro naipes e os treze elementos (números e Figuras).

3.3.3 CLUSTERIZAÇÃO HIERÁRQUICA

O último algoritmo é responsável então por analisar os dados, agrupando os diferentes nós da árvore de forma eficiente, de maneira a permitir a obtenção dos maiores blocos possíveis que obedecem a uma condição de admissibilidade, a qual garante a qualidade das aproximações a serem calculadas. Assim, o objetivo é construir uma árvore binária que seja adequada para uma aproximação de baixo posto, mas que apresente poucos nós e a menor quantidade de níveis possível.

A condição de admissibilidade para que um bloco possa ser representado por uma matriz de baixo posto apresenta forma geral [14]:

$$\min(diam(\Omega_s, \Omega_t) \leq \mu\, dist(\Omega_s, \Omega_t)$$

onde:

$$diam(\Omega_i) = máx\left\{\left\|x_i - x_j\right\| : x_i, x_j \in \Omega_i\right\}$$

$$dist(\Omega_s, \Omega_t) = mín\left\{\left\|x_i - x_j\right\| : x_i \in \Omega_s \; e \; x_j \in \Omega_t\right\}$$

tal que Ω_s e Ω_t são as fronteiras, ou suportes, dos conjuntos de dados s e t respectivamente. O fator μ controla o número de blocos de matrizes de baixo posto e a precisão da aproximação. Quanto menor μ, menor deve ser o diâmetro do conjunto de dados. Isto resulta em submatrizes menores e em maior quantidade. As submatrizes menores possibilitam uma quantidade menor de aproximações de baixo posto, e consequentemente uma representação mais precisa da matriz. No entanto, essa representação mais precisa devido a um menor μ não é desejável, pois ela é resultado de um menor número de submatrizes aproximadas por matrizes de baixo posto, aumentando a complexidade. Usualmente $\mu = 2$ é considerado um valor ótimo capaz de produzir grandes blocos de matrizes aproximadas por matrizes de baixo posto e apresentar uma representação da matriz original com boa precisão.

Para matrizes hierárquicas, a condição de mínimo é substituída por máximo, obtendo uma condição chamada de condição de admissibilidade forte. Essa mudança penaliza a admissibilidade de blocos muito grandes com blocos pequenos tornando esses inadmissíveis. Como resultado, temos blocos admissíveis mais homogêneo, mas de tamanho reduzido. Além disso, a forma geral da condição é substituída por [15]:

$$\text{máx}\big(diam(B_s, B_t)\big) \leq 2dist(B_t, B_s)$$

onde B_s e B_t são ainda os suportes dos conjuntos de dados s e t respectivamente, mas agora são definidas paralelas aos eixos coordenados, formando uma caixa paralelas aos eixos (Figura 13). Essa substituição permite uma menor complexidade na análise da condição de admissibilidade.

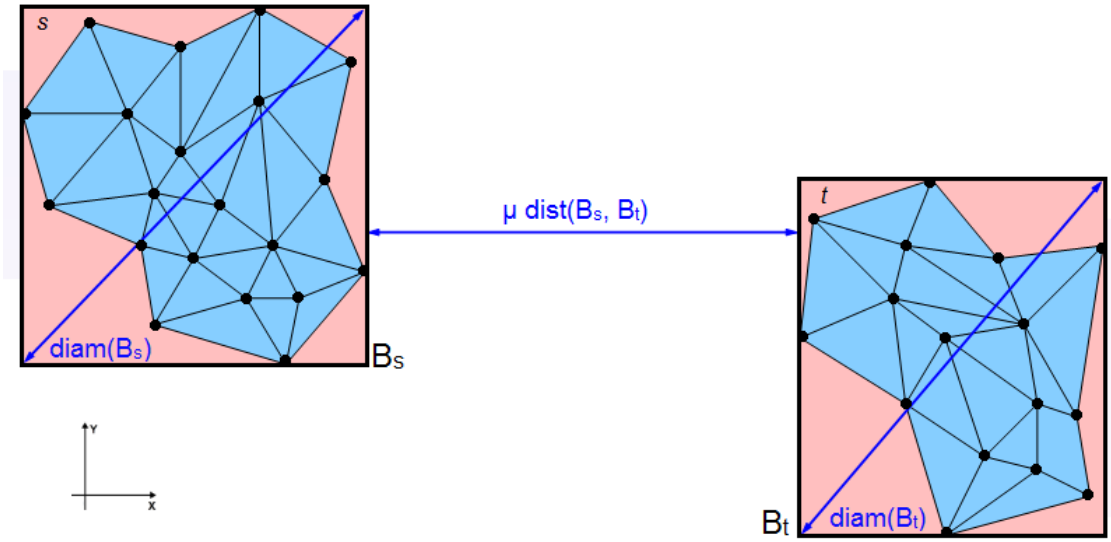

Figura 14 - Condição de admissibilidade

Quando um nó é admissível, ele se torna uma folha, caso contrário, seus filhos são construídos e assim sucessivamente até que se alcance a quantidade mínima permitida de dados em um bloco.

O objetivo do método é permitir a construção de uma árvore binária balanceada para uma malha multidimensional, onde a raiz da árvore são os pontos coordenados de todos os nós da malha.

O propósito do método ACA é aproximar por matrizes de baixo posto o máximo de submatrizes (blocos admissíveis) obtidas por esse particionamento. Como cada submatriz corresponde a uma folha da árvore binária, é desejável obter o menor número possível de folhas, resultando em blocos maiores a serem aproximados. Além disso é desejável que as folhas inadmissíveis, ou seja, as que não atendem a condição de admissibilidade, sejam formadas por pequenos conjunto de dados, pois estes serão representados sem aproximação pela matriz exata.

Deste modo, a construção da árvore é baseada na decomposição do domínio, ou seja, na bissecção dos conjuntos de pontos coordenados. Essa subdivisão recursiva é decidida pelo cumprimento ou não da condição de admissibilidade, a qual requer a construção de caixa de contorno.

Considere como exemplo a árvore T_{IxJ} para o bloco $\{1,2,3,4\}X\{1,2,3,4\}$ apresentada em (a) na Figura 15. Suponha que na primeira divisão os blocos *BP* satisfazem a condição de admissibilidade. Assim, eles se tornam folhas e podem ser aproximados por uma matriz de baixo posto. Os blocos restantes, que não satisfazem

a condição, são repartidos novamente, e mais uma vez seus filhos não satisfazem a condição de admissibilidade. No entanto, eles atingem a quantidade mínima de dados permitida em um bloco. Assim, a matriz hierárquica tem o formato como mostrado em (b) na Figura 15, onde os blocos em azul serão aqueles aproximados por matrizes de baixo posto, enquanto o restante terá representação exata.

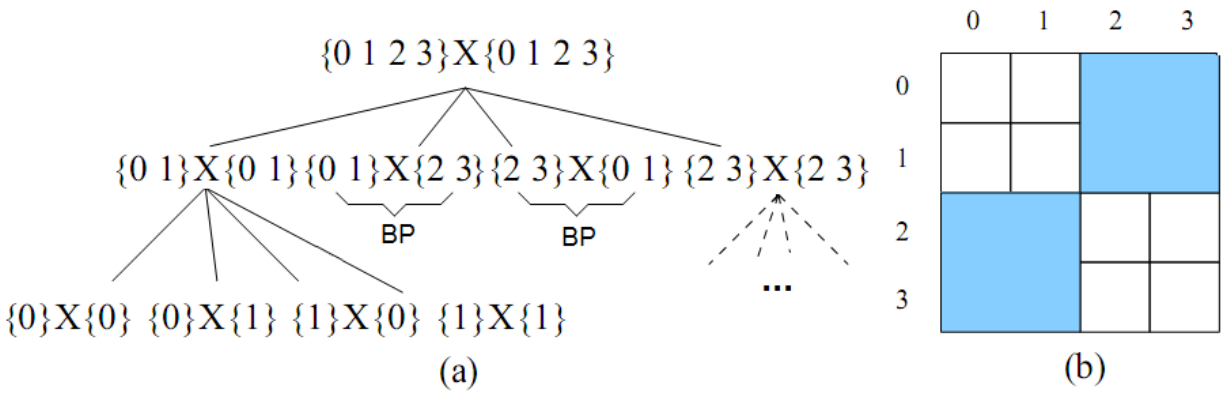

Figura 15 - Árvore T_{IxJ} para o bloco $\{1,2,3,4\}X\{1,2,3,4\}$ (a) é a matriz hierárquica correspondente (b).

4 MÉTODOS ITERATIVOS

Para a resolução de problemas na forma matricial $\boldsymbol{A}x = b$, utilizar o método de eliminação de Gauss nem sempre é vantajoso. Esse é um processo caro e demorado, que apresenta complexidade $O(N^3)$, requerendo grande capacidade de armazenamento e processamento para resolver problemas de larga escala. Nesse contexto, a utilização de métodos iterativos [24], os quais computam o produto matriz-vetor $\boldsymbol{A}x$ em cada iteração, é mais vantajoso por apresentar complexidade O($n_{int}N^2$), onde n_{int} é o número de iterações. Se considerarmos ainda que a matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$ pode ser representada de forma esparsa, com apenas p dados em cada linha, tal que $p <<< n$, a complexidade se torna O($n_{int}pN$), complexidade essa quase linear quando o número de iterações é suficientemente pequeno.

Na utilização de métodos iterativos, existem duas grandes escolhas a serem feitas, a primeira, é a escolha de um bom pré-condicionador e a segunda, a do método a ser utilizado. Um bom pré-condicionador $\boldsymbol{P}$ é uma matriz próxima a $\boldsymbol{A}$, mas que é mais simples de manipular. Entre os diferentes existentes, discutiremos sobre os métodos estacionários, o método Multigrid e os métodos de subespaços de Krylov, o qual inclui o método do gradiente conjugado e o método dos mínimos resíduos generalizados.

4.1 MÉTODOS ESTACIONÁRIOS OU PURAMENTE ITERATIVOS

Os métodos estacionários [25], também chamados de puramente iterativos, computam cada novo vetor x a partir de uma forma recursiva do problema na forma matricial $\boldsymbol{A}x = b$, obtida fazendo:

$$\boldsymbol{A}x = b$$

$$0 = -\boldsymbol{A}x + b$$

$$x - x = -\boldsymbol{A}x + b$$

$$x = x - \boldsymbol{A}x + b$$

$$x = (\boldsymbol{I} - \boldsymbol{A})x + b$$

ao ser escrito na forma recursiva:

$$x_{k+1} = (\boldsymbol{I} - \boldsymbol{A})x_k + b$$

Esse método é chamado de estacionário porque toda iteração é resultado de um mesmo processo. Ele pode ser utilizado em qualquer tipo de matriz, não precisando a matriz ser simétrica ou definida positiva. Entre os métodos estacionários, estão o método de Jacobi é o método de Gauss-Seidel.

O pré-condicionador é uma matriz $\boldsymbol{P}$, que tem como objetivo tornar o processo iterativo mais rápido. A forma exata com o pré-condicionador é dada por:

$$\boldsymbol{P}x = (\boldsymbol{P} - \boldsymbol{A})x + b \tag{4.1}$$

e então, na forma recursiva:

$$\boldsymbol{P}x_{k+1} = (\boldsymbol{P} - \boldsymbol{A})x_k + b \tag{4.2}$$

Observe que caso definirmos $\boldsymbol{P} = \boldsymbol{A}$, o sistema retorna para a forma $\boldsymbol{A}x = b$. Por essa razão a matriz $\boldsymbol{P}$**,** embora parecida com $\boldsymbol{A}$, não pode ser igual a essa. Para que um pré-condicionar seja bem sucedido, ele deve ser escolhido de forma que x_{k+1} possa ser calculado rapidamente, ou seja, que a equação (4.2) possa ser resolvida rapidamente. Além disso, é desejável que erro $e_k = x - x_k$ decaia rapidamente para que uma convergência seja obtida em um pequeno número de iterações. A equação para o erro é obtida subtraindo (4.2) de (4.1), de forma a obtermos:

$$\boldsymbol{P}x - \boldsymbol{P}x_{k+1} = (\boldsymbol{P} - \boldsymbol{A})x + b - [(\boldsymbol{P} - \boldsymbol{A})x_k + b]$$

$$\boldsymbol{P}(x - x_{k+1}) = (\boldsymbol{P} - \boldsymbol{A})(x - x_k)$$

$$(x - x_{k+1}) = \boldsymbol{P}^{-1}(\boldsymbol{P} - \boldsymbol{A})(x - x_k)$$

$$e_{k+1} = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A})e_k$$

$$e_{k+1} = \boldsymbol{M}e_k; \quad \boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A})$$

Para analisarmos a influência da Matriz $\boldsymbol{M}$, suponha um erro inicial dado por e_0, desse modo podemos escrever:

$$e_1 = \boldsymbol{M}e_0$$

$$e_2 = \boldsymbol{M}e_1 = \boldsymbol{M}\,(\boldsymbol{M}e_0) = \boldsymbol{M}^2e_0$$

$$e_3 = \boldsymbol{M}e_2 = \boldsymbol{M}\,(\boldsymbol{M}^2e_0) = \boldsymbol{M}^3e_0$$

e então para a k-ésima iteração:

$$e_k = \boldsymbol{M}^k e_0$$

Dessa forma, o decaimento do erro depende somente de $\boldsymbol{M}$, e consequentemente, a rapidez do processo convergência. Considerando $\boldsymbol{M}$ uma matriz de posto cheio, ou seja, uma matriz $\boldsymbol{M} \in \mathbb{R}^{nxn}$ de posto n. Assim, ela pode ser diagonalizada e escrita na forma $\boldsymbol{M} = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E}$, onde $\boldsymbol{E}$ é a matriz de autovetores de $\boldsymbol{M}$ e $\boldsymbol{\Lambda}$ a matriz diagonal de autovalores. Podemos então, reescrever o erro e_k na forma:

$$e_k = \boldsymbol{M}^k e_0 = (\boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E})^{\boldsymbol{k}} e_0 = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}^k \boldsymbol{E} e_0 \qquad (4.3)$$

onde:

$$\boldsymbol{\Lambda}^{\boldsymbol{k}} = \begin{pmatrix} {\lambda_1}^k & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & {\lambda_n}^k \end{pmatrix}$$

Nessa forma, podemos não somente notar que o decaimento do erro depende inteiramente dos autovalores de $\boldsymbol{M}$, mas também, que só há convergência $e_k \to 0$ quando todos os autovalores de $\boldsymbol{M}$ são menores que a unidade, ou seja, para haver convergência é preciso satisfazer:

$$\left|\lambda_j(\boldsymbol{M})\right| < 1$$

o que implica em:

$${\lambda_j}^k \to 0$$

caso contrário teríamos que:

$${\lambda_j}^k \to \infty$$

e como consequência, não haveria convergência. Outra importante observação, é que a velocidade de convergência depende somente do autovalor de maior valor em módulo, o chamado raio espectral. Deste modo, a condição para convergência requer que o raio espectral $\rho(M) = máx\left|\lambda_j(M)\right| < 1$.

Se não há o uso de um pré-condicionador, temos que $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{A})$ e então é preciso que todos os autovalores de $\boldsymbol{A}$ estejam dentro do intervalo $0 \leq \lambda_j < 2$, para que exista convergência. Deste modo, a primeira função de um bom pré-condicionador é dimensionar a matriz de maneira apropriada.

4.1.1 MATRIZ MODELO

Para a análise de alguns dos métodos iterativos a seguir, utilizaremos recursivamente matrizes com o formato:

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

Essa é uma matriz tridiagonal, que possui somente o valor 2 em sua diagonal principal e -1 nas diagonais logo acima e abaixo dessa. Para entendermos melhor as suas propriedades, mostraremos qual tipo de problema origina essa matriz. Sendo assim, considere um problema qualquer descrito por uma equação diferencial de segunda ordem na forma:

$$-\frac{d^2 f}{dx^2}$$

Esse problema contínuo pode ser discretizado, escrevendo o mesmo na forma de uma equação de diferenças de segunda ordem. Para tal, considere a expansão em série de Taylor, em torno de $h = 0$, da função $f(x + h)$, dada por:

$$f(x + h) = f(x) + f'(x)h + O(h^2)$$

e então:

$$f'(x) = \frac{f(x + h) - f(x)}{h}$$

Essa é a chamada equação de diferenças de primeira ordem pela direita, seu erro é da ordem de $O(h^2)$. Essa mesma expansão, mas agora para $f(x - h)$, nós fornece:

$$f'(x) = \frac{f(x) - f(x - h)}{h}$$

Essa é a chamada equação de diferenças de primeira ordem pela esquerda, seu erro é também da ordem de $O(h^2)$. Somando ambas as expansões, obtemos:

$$f'(x) = \frac{f(x + h) - f(x - h)}{2h}$$

A qual é a equação de diferença de primeira ordem centrada, seu erro, diferentemente das outras, é da ordem de $O(h^3)$, pois o termo do erro de segunda

ordem pela direita é positivo, e o pela esquerda é negativo. Assim, quando somados eles anulam um ao outro.

Para enfim, obtermos a equação de diferenças de segunda, utilizamos mais uma vez a expansão em série de Taylor, em torno de $h = 0$, da função $f(x + h)$. Mas dessa vez, considerando mais um termo:

$$f(x + h) = f(x) + f'(x)h + \frac{f''(x)h^2}{2} + O(h^3)$$

e então:

$$f''(x)h^2 = 2f(x + h) - 2f(x) - 2hf'(x)$$

da equação de diferença de primeira ordem centrada, temos:

$$-2hf'(x) = -f(x + h) + f(x - h)$$

e então reescrevemos a expansão anterior, como:

$$f''(x)h^2 = 2f(x + h) + 2f(x) - f(x + h) + f(x - h)$$

$$f''(x) = \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}$$

Essa é a chamada equação de diferenças de segunda ordem, seu erro é também da ordem de $O(h^3)$. Para diferenças de segunda ordem não há alteração da equação pela esquerda ou direita. O problema contínuo dado pela equação diferencial de segunda ordem foi discretizado pela equação de diferenças de segunda ordem, onde $f(x + h), f(x)\ e\ f(x - h)$ são valores $a_{i+1},\ a_i\ e\ a_{i-1}$ em três pontos consecutivos quaisquer. Podemos agora escrever:

$$-\frac{d^2f}{dx^2} \approx \frac{1}{h^2}(-a_{i+1} + 2\ a_i - a_{i-1}) = \frac{1}{h^2}\begin{bmatrix} & & \cdots & & \\ -1 & 2 & -1 & & 0 \\ & -1 & 2 & -1 & \\ 0 & & -1 & 2 & -1 \\ & & \cdots & & \end{bmatrix}\begin{bmatrix} \vdots \\ a_{i+1} \\ a_i \\ a_{i-1} \\ \vdots \end{bmatrix} = \boldsymbol{A}x$$

Quando conhecemos diretamente a equação que da origem a uma matriz qualquer, podemos calcular os autovetores e autovalores dessa matriz de forma contínua, calculando a chamada autofunção. O problema discreto de autovalores é definido como:

$$\boldsymbol{A}x = \lambda \mathrm{x}$$

Para o problema contínuo dado pela matriz acima, podemos de forma análoga definir:

$$-\frac{d^2f}{dx^2} = \lambda f$$

Cuja solução é $f = Asen(\omega x) + Bcos(\omega x)$, onde a A e B são constantes que dependem da condição de contorno. A primeira propriedade que observamos então, é que os autovalores da matriz $\boldsymbol{A}$ são uma composição das funções seno e cosseno. Para entendermos outra importante propriedade, considere o caso em que $f = sen(\omega x)$, o que ocorre quando a condição de contorno nas extremidades é dada por $f = 0$. Desse modo temos que:

$$-\frac{d^2f}{dx^2} = \omega^2 sen(\omega x) = \lambda f$$

e consequentemente, temos que os autovalores são:

$$\lambda = \omega^2$$

Assim, outra importante propriedade é que quanto maior o autovalor, maior é a frequência do autovetor associado a esse autovalor. Tais conclusões serão importantes posteriormente.

4.1.2 MÉTODO DE JACOBI

O método de Jacobi define o pré-condicionador $\boldsymbol{P}$ como:

$$\boldsymbol{P} = \boldsymbol{D} = diag(\boldsymbol{A})$$

Para entendermos o efeito de escolher $\boldsymbol{D}$ como o pré-condicionador, considere a matriz $\boldsymbol{A}$ dada por:

$$A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix} \tag{4.4}$$

O que corresponde a discretização do problema $-\frac{d^2f}{dx^2}$ para $h = 1$ é condição de contorno nas extremidades de $f = 0$, ou seja, $a_0 = 0$ e $a_{n+1} = 0$.

Os autovalores de $\boldsymbol{A}$ são as raízes da equação característica, obtida por:

$$det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0$$

$$(2-\lambda)^3 - 2(2-\lambda) = 0$$

É possível observar que $\lambda_2(\boldsymbol{A}) = 2$ é um autovalor. Dividindo a equação característica por $(2-\lambda)$, podemos achar os outros autovalores:

$$(2-\ \lambda)^2 - 2 = 0$$

$$2-\ \lambda = \pm\sqrt{2}$$

$$\lambda_1(\boldsymbol{A}) = 2 + \sqrt{2}$$

$$\lambda_3(\boldsymbol{A}) = 2 - \sqrt{2}$$

Para prosseguirmos com a análise é importante entender que os autovalores de $\boldsymbol{A} - c\boldsymbol{I}$, são simplesmente $\lambda(A) - c$, pois:

$$det(\boldsymbol{A} - \lambda(\boldsymbol{A})\boldsymbol{I}) = 0$$

$$det(\boldsymbol{A} + (-c\boldsymbol{I} + c\boldsymbol{I}) - \lambda(\boldsymbol{A})\boldsymbol{I}) = 0$$

$$det\big((\boldsymbol{A} - c\boldsymbol{I}) - (-c\boldsymbol{I} + \lambda(\boldsymbol{A})\boldsymbol{I})\big) = 0$$

$$det\big((\boldsymbol{A} - c\boldsymbol{I}) + (\lambda(\boldsymbol{A}) - c)\boldsymbol{I}\big) = 0$$

e então:

$$\lambda(\boldsymbol{A} - c\boldsymbol{I}) =\ \lambda(\boldsymbol{A}) - c$$

Caso não fosse utilizado nenhum pré-condicionador, teríamos que $\boldsymbol{M} = \boldsymbol{I} - \boldsymbol{A}$ e os autovalores de $\boldsymbol{M}$ seriam dados por:

$$\lambda(\boldsymbol{M}) = \lambda(\boldsymbol{I} - \boldsymbol{A}) =\ 1 -\ \lambda(\boldsymbol{A})$$

É simples notar que o raio espectral é:

$$\rho(\boldsymbol{M}) = máx\big|\lambda_j(\boldsymbol{M})\big| = |1 -\ \lambda_3(\boldsymbol{A})| = 1 + \sqrt{2}$$

Desse modo, não haveria convergência. Utilizando agora o pré-condicionador $\boldsymbol{P}$ dado pela diagonal da matriz $\boldsymbol{A}$, temos que $\boldsymbol{P} = 2\boldsymbol{I}$ e $\boldsymbol{P^{-1}} = \frac{1}{2}\boldsymbol{I}$. Deste modo, temos que $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{P^{-1}A})$ e os autovalores de $\boldsymbol{M}$ são dados por:

$$\lambda(\boldsymbol{M}) = \lambda(\boldsymbol{I} - \boldsymbol{P^{-1}A}) =\ 1 - \frac{1}{2}\lambda(\boldsymbol{A})$$

Assim, com o pré-condicionador, o novo raio espectral é:

$$\rho(\boldsymbol{M}) = máx\left|\lambda_j(\boldsymbol{M})\right| = \left|1 - \frac{1}{2}\lambda_1(\boldsymbol{A})\right| = \left|1 - \frac{1}{2}\lambda_3(\boldsymbol{A})\right| = \frac{\sqrt{2}}{2}$$

e como consequência agora haverá convergência. Observe também que agora o raio espectral é o mesmo tanto para $j = 1$ e $j = n = 3$.

4.1.3 MÉTODO DE JACOBI PONDERADO

A matriz utilizada no exemplo anterior (4.4) é uma matriz do tipo Toeplitz Tridiagonal, ou seja, uma matriz onde somente a diagonal principal e as diagonais logo acima e abaixo dessa são diferentes de zero e cada diagonal é constante. Matrizes desse tipo tem a forma:

$$\boldsymbol{T} = \begin{bmatrix} \delta & \tau & & & & 0 \\ \sigma & \delta & \tau & & & \\ & \sigma & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \tau \\ 0 & & & & \sigma & \delta \end{bmatrix}$$

Esse tipo de matriz tem os autovalores bem conhecidos, dados por [26]:

$$\lambda_j(\boldsymbol{T}) = \ \delta + 2\sqrt{\sigma\delta}\, cos\left(\frac{j\pi}{n+1}\right); \quad j = 1{:}\, n$$

no caso da matriz (4.4), temos:

$$\lambda_j(\boldsymbol{A}) = \ 2 - 2cos\left(\frac{j\pi}{4}\right); \quad j = 1{:}\, 3$$

No método de Jacobi, os autovalores da matriz $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{P^{-1}A})$ foram dados por:

$$\lambda(\boldsymbol{M}) = 1 - \frac{1}{2}\lambda(\boldsymbol{A})$$

é então, substituindo $\lambda(\boldsymbol{A})$:

$$\lambda_j(\boldsymbol{M}) = 1 - \frac{1}{2}\left[2 - 2cos\left(\frac{j\pi}{4}\right)\right]$$

$$\lambda_j(\boldsymbol{M}) = 1 - 1 + cos\left(\frac{j\pi}{4}\right)$$

$$\lambda_j(\boldsymbol{M}) = cos\left(\frac{j\pi}{4}\right)$$

Os autovalores de $\boldsymbol{M}$ são sempre menores que um, ou seja, sempre haverá convergência, além disso, $\left|\lambda_j(\boldsymbol{M})\right|$ é máximo para $j = 1$ e $j = n$, e consequentemente:

$$\rho(\boldsymbol{M}) = cos\left(\frac{\pi}{4}\right) = \left|cos\left(\frac{3\pi}{4}\right)\right| = \frac{\sqrt{2}}{2}$$

como calculado anteriormente. No entanto, quanto maior a matriz, e consequentemente quanto maior n, mais próximo do valor unitário são os autovalores e o raio espectral. Observe que quando $n \to \infty$, temos que:

$$\lambda_1(\boldsymbol{M})_{\lim n\to\infty} = cos\left(\frac{\pi}{n+1}\right) = cos(0) = 1$$

$$\lambda_n(\boldsymbol{M})_{\lim n\to\infty} = cos\left(\frac{n\pi}{n+1}\right) = cos\left(\frac{\pi}{1+\frac{1}{n}}\right) = cos(\pi) = -1$$

Como consequência, quanto maior n mais lenta é a convergência.

Para resolver, em parte, esse problema o método de Jacobi ponderado tenta amortecer, ou seja, diminuir o valor absoluto, dos autovalores associados aos autovetores de alta frequência. Para tal, o pré-condicionador é definido como:

$$\boldsymbol{P} = \frac{1}{\omega}\boldsymbol{P}_{Jacobi}$$

onde ω está no intervalo $0 < \omega < 1$. Com essa mudança, os autovalores de $\boldsymbol{M}$, serão dados por:

$$\lambda(\boldsymbol{M}) = \lambda(\boldsymbol{I} - \boldsymbol{P^{-1}A}) = 1 - \frac{\omega}{2}\lambda(\boldsymbol{A})$$

$$\lambda(\boldsymbol{M}) = 1 - \frac{\omega}{2}\left[2 - 2cos\left(\frac{j\pi}{n+1}\right)\right]$$

Para matrizes do tipo (4.4) temos que $\omega = \frac{2}{3}$ é um valor ótimo:

$$\lambda(\boldsymbol{M}) = 1 - \frac{1}{3}\left[2 - 2cos\left(\frac{j\pi}{n+1}\right)\right]$$

$$\lambda(\boldsymbol{M}) = \frac{1}{3} + \frac{2}{3}cos\left(\frac{j\pi}{n+1}\right)$$

Observe que agora quando $n \to \infty$, temos que:

$$\lambda_1(\boldsymbol{M})_{\lim n\to\infty} = \frac{1}{3} + \frac{2}{3}cos\left(\frac{j\pi}{n+1}\right) = \frac{1}{3} + \frac{2}{3}cos(0) = 1$$

$$\lambda_n(\boldsymbol{M})_{\lim n\to\infty} = \frac{1}{3} + \frac{2}{3}cos\left(\frac{j\pi}{n+1}\right) = \frac{1}{3} + \frac{2}{3}cos(\pi) = -\frac{1}{3}$$

Assim, o método de Jacobi ponderado amortece os autovalores relacionados aos autovetores de maior frequência. Isso torna o erro e_k mais suave, pois de (4.3) o erro pode ser escrito como:

$$e_k = \boldsymbol{E^{-1}\Lambda^k E}e_0$$

onde $\boldsymbol{E}$ são os autovetores de $\boldsymbol{M}$. Deste modo, quando os autovalores relacionados aos autovetores de maior frequência são reduzidos, o efeito desses sobre o erro é menor, e consequentemente o erro se torna mais suave.

O comportamento dos autovalores de $\boldsymbol{M}$ para a matriz (4.4) pode ser melhor observado na Figura 16, onde os pontos marcados correspondem aos autovalores λ_j para $j = 1{:}3$ da matriz (4.4).

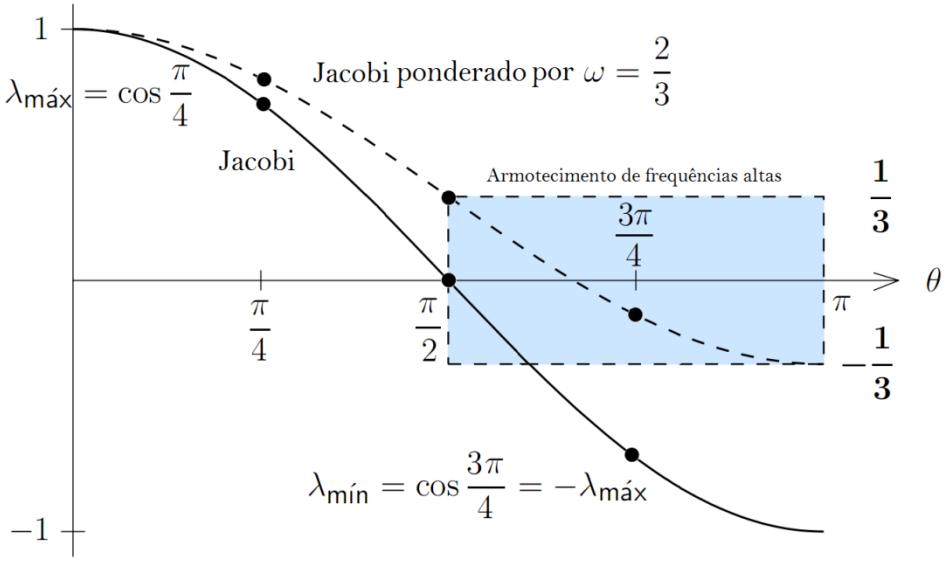

Figura 16 - Comparação entre os autovalores da matriz $\boldsymbol{M}$ no método de Jacobi e Jacobi ponderado.

Embora o método não reduza de forma significativa o raio espectral, ele é essencial para a implementação do método Multigrid, onde é necessário que as altas frequências estejam altamente amortecidas.

4.1.4 MÉTODO DE GAUSS-SEIDEL

O método de Gauss-Seidel utiliza as componentes do vetor x_{k+1} logo que elas são calculadas. Isso reduz a necessidade de armazenamento uma vez que x_{k+1} sobrescreve x_k. O método utiliza como pré-condicionador $\boldsymbol{P}$ a matriz definida por:

$$\boldsymbol{P} = \boldsymbol{D} + \boldsymbol{L} = tril(\boldsymbol{A})$$

onde $\boldsymbol{D}$ **e** $\boldsymbol{L}$ são respectivamente a diagonal (pré-condicionador do método de Jacobi) e a parte estritamente inferior da matriz $\boldsymbol{A}$**.**

Considere como exemplo a matriz genérica $\boldsymbol{A} \in \mathbb{R}^{3x3}$, dada por:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

O pré-condicionador $\boldsymbol{P}$ é a parte triangular inferior de $\boldsymbol{A}$, dada por:

$$\boldsymbol{P} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

e então, temos que de (4.2):

$$\boldsymbol{P}x_{k+1} = (\boldsymbol{P} - \boldsymbol{A})x_k + b$$

$$\begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{k+1}^1 \\ x_{k+1}^2 \\ x_{k+1}^3 \end{bmatrix} = \begin{bmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_k^1 \\ x_k^2 \\ x_k^3 \end{bmatrix} + b$$

Escrevendo na forma de um sistema de equações, obtemos:

$$a_{11}x_{k+1}^1 = a_{12}x_k^2 + a_{13}x_k^3 + b_1$$

$$a_{21}x_{k+1}^1 + a_{22}x_{k+1}^2 = a_{12}x_k^2 + b_2$$

$$a_{31}x_{k+1}^1 + a_{32}x_{k+1}^2 + a_{33}x_{k+1}^3 = b_3$$

Assim, é fácil observar que para o cálculo de x_{k+1}^1, são utilizadas as entradas x_k^2 e x_k^3. Para o cálculo de x_{k+1}^2 são utilizados x_k^2 é o recém-calculado x_{k+1}^1 e finalmente, para o cálculo de x_{k+1}^3 são utilizados x_{k+1}^1 e x_{k+1}^2, ambos calculados nessa iteração.

O método de Gauss-Seidel apresenta duas vantagens, a primeira é que o cálculo de x_{k+1} pode ser feito simplesmente por substituição progressiva. A segunda,

como já dito anteriormente, é que calculada uma entrada de x_{k+1}, essa é utilizada para calcular a próxima entrada. Desse modo, uma iteração de Gauss-Seidel equivale a aproximadamente duas iterações de Jacobi.

4.2 MÉTODO DA SUPER-RELAXAÇÃO SUCESSIVA (SOR)

O método da super-relaxação sucessiva ajusta o pré-condicionador do método de Gauss-Seidel por um fator ω. O novo pré-condicionador é então dado por:

$$\boldsymbol{P} = \boldsymbol{D} + \omega \boldsymbol{L}$$

Para $\omega = 0$ temos o método de Jacobi e para $\omega = 1$ temos o método de Gauss-Seidel.

O parâmetro ω varia no intervalo $0 < \omega < 2$, para valores de $\omega > 2$ não há convergência [27]. Para valores de $\omega > 1$, temos o processo de super-relaxação utilizado para acelerar o processo de convergência, de processos que convergem lentamente. Para valores de $\omega < 1$, temos o processo de sub-relaxação utilizado para estabelecer convergência de processos que tendem a divergir.

Para o caso da super-relaxação, obter um parâmetro de ótimo ω_{opt}, resulta em uma convergência extremamente mais rápida que nos métodos de Jacobi e Gauss-Seidel. Em geral, não é possível obter um ω_{opt} de maneira adiantada, mas para certas classes de matrizes podemos definir [28] [29]:

$$\omega_{opt} \approx \frac{2}{1 + \sqrt{1 - \rho(M)^2}} > 1$$

onde $\rho(M)$ é o raio espectral da Matriz $\boldsymbol{M}$ obtida pela método de Jacobi. Para a matriz tridiagonal do tipo (4.4) essa relação é válida. Assim, para esse tipo de matriz temos:

$$\rho(M) = \cos\left(\frac{\pi}{n+1}\right)$$

e então

$$\omega_{opt} \approx \frac{2}{1 + \sqrt{1 - \left[\cos\left(\frac{\pi}{n+1}\right)\right]^2}} = \frac{2}{1 + \sqrt{1 - 1 + \left[sen\left(\frac{\pi}{n+1}\right)\right]^2}}$$

$$\omega_{opt} \approx \frac{2}{1 + \sqrt{\left[sen\left(\frac{\pi}{n+1}\right)\right]^2}} = \frac{2}{1 + sen\left(\frac{\pi}{n+1}\right)}$$

4.3 MÉTODO MULTIGRID

Os métodos estacionários de Jacobi ponderado e Gauss-Seidel conseguem rapidamente eliminar os componentes de alta frequência do erro e_k, resultando depois de poucas iterações em erros suaves. No entanto, a partir desse ponto o decaimento do erro é extremamente lento.

A ideia principal do método Multigrid [25] [30] é representar o problema por uma malha mais grosseira onde o espaçamento h entre os nós é maior, de forma que o erro suave obtido pelos métodos anteriores não seja mais tão suave e possa ser eliminado rapidamente pelos métodos estacionários. No final do processo, o resultado é interpolado para a malha original.

O método Multigrid sozinho não leva a convergência de x_k, ele somente dimensiona o problema para diferentes malhas. Uma vez dimensionado, o que leva a convergência são as iterações feitas com os métodos estacionários.

Uma vez que o método funciona em conjunto com os métodos estacionários, existem diferentes estratégias, ciclos, de utilização do mesmo (Figura 17). Em cada tipo de malha, resultantes dos diferentes espaçamento entre os nós (h, 2h, 4h e 8h), o problema é resolvido parcialmente por um método iterativo.

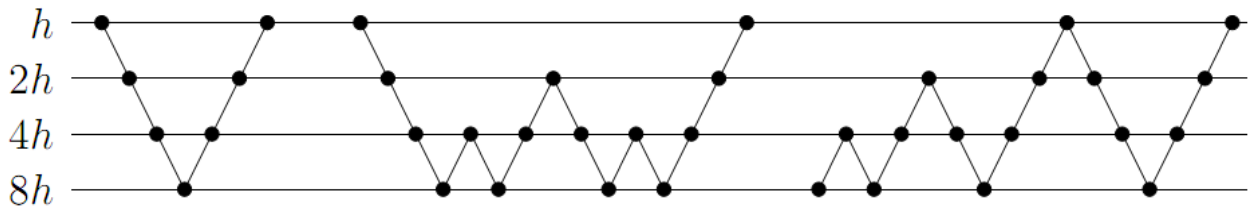

Figura 17 - Ciclo-V, Ciclo-W e Ciclo FMG do método Multigrid.

Considere um problema que após ser discretizado apresenta malha com espaçamento h entre os nós. No caso do ciclo V, é realizado primeiramente algumas iterações utilizando métodos estacionários (ponto no nível h), em seguida a malha é modificada (linha que conecta o ponto no nível h com o ponto no nível 2h) tornando essa mais grosseira, agora com espaçamento 2h entre os nós e consequentemente metade do número de nós. Após a modificação da malha, é realizado novamente algumas iterações utilizando métodos estacionários (ponto no nível 2h). Esse processo é feito repetidamente até atingir a malha mais grosseira possível, nesse caso com espaçamento 8h entre os nós. Após chegar a esse ponto, a malha sofre uma transformação reversa, se tornando mais refinada, até chegar novamente a malha

original, sempre com a realização de iterações com métodos estacionários a cada transformação da malha.

Essa possiblidade de tornar a malha original em uma malha mais grosseira ou mais refinada é o que dá origem aos diversos ciclos e ao nome do método Multigrid.

Enquanto o processo iterativo para a malha original resolve $\boldsymbol{A}x = b$, o processo iterativo para a malha grosseira resolve $\boldsymbol{A}e = r$, onde r é o erro residual. O erro residual, ou resíduo, é o erro na equação $\boldsymbol{A}x$ e não na solução x. Para um x_k obtido depois de k iterações temos que:

$$\boldsymbol{A}x_k + r_k = b$$

$$r_k = b - \boldsymbol{A}x_k \tag{4.5}$$

Se $\boldsymbol{A}x_k = \boldsymbol{A}x$, o resíduo é nulo, então r_k de fato representa o quão longe $\boldsymbol{A}x_k$ está de $\boldsymbol{A}x$, ou seja, o erro da equação. Podemos escrever ainda que:

$$r_k = \boldsymbol{A}x - \boldsymbol{A}x_k$$

$$r_k = \boldsymbol{A}(x - x_k)$$

$$r_k = \boldsymbol{A}e_k \tag{4.6}$$

O ciclo-v (o v minúsculo indica somente uma malha diferente utilizada) do método Multigrid é feito da seguinte forma:

1. n Iterações para $\boldsymbol{A}_h x_h = b_h$
2. Reduzir r_h para a malha grosseira fazendo $r_{2h} = \boldsymbol{R}r_h$
3. Resolver $\boldsymbol{A}_{2h}e_{2h} = r_{2h}$
4. Interpolar novamente para a malha original $E_h = \boldsymbol{T}e_{2h}$
5. n Iterações para $\boldsymbol{A}_h x_h = b_h$; onde $x_h = x_h + E_h$

O primeiro passo é necessário para eliminar as componentes de alta frequência do erro e_k. O subscrito h indica a distância entre os nós, onde h é a distância original e $2h$ é uma malha com o dobro da distância, e consequentemente metade do número de nós. O terceiro passo é resolvido iterativamente.

Para obter uma solução é preciso conhecer $\boldsymbol{R}$, $\boldsymbol{A}_{2h}$ e $\boldsymbol{T}$. Para entender como obter essas matrizes, considere o exemplo unidimensional discretizado pela malha abaixo:

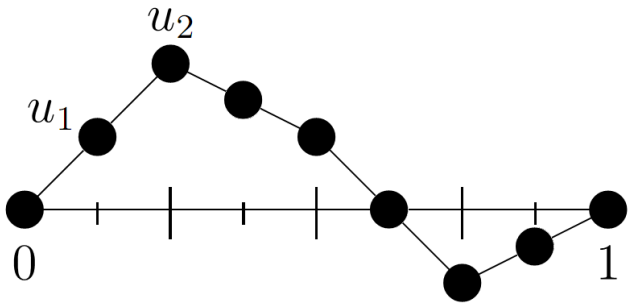

Figura 18 - Malha Original

A malha acima apresenta espaçamento $h = \frac{\Delta x}{n+1} = \frac{1}{8}$ no intervalo $0 \leq x \leq 1$ e condição de contorno de $u = 0$. Os sete valores interiores são os nós u_j. A representação dessa malha de maneira mais esparsa apresenta $2h = \frac{1}{4}$ e consequentemente três nós. Comumente são utilizados os espaçamentos múltiplos de dois tais como $h, 2h, 4h, \ldots, 2^n h$. Desse modo, em cada redução o número de nós é reduzido pela metade, pois a distância entre os nós é dobrada. A representação esparsa da malha acima é dada por:

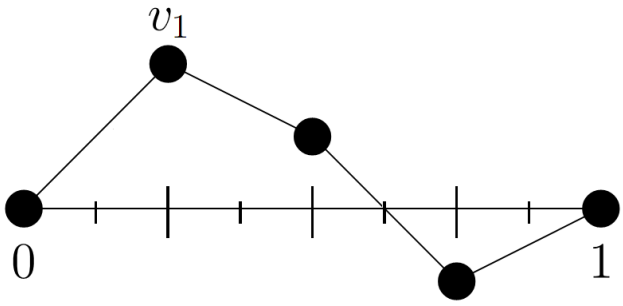

Figura 19 - Representação esparsa da malha dada pela Figura 18.

Observe que u_2, u_4 e u_6 são os mesmos que v_1, v_2 e v_3, esses nós são apenas transferidos de uma malha para a outra. A matriz de interpolação $\boldsymbol{T}$ é responsável por transferir os vetores na malha grosseira novamente para a malha original. Para tal, ela interpola os valores nos nós da malha reduzida, obtendo os valores na malha original. É possível utilizar diferentes tipos de interpolação, mas quando o erro é suave à interpolação linear é simples é eficiente. Para a interpolação linear, definimos:

$$u_{2j} = v_j$$

$$u_{2j+1} = \frac{1}{2}(v_j - v_{j+1})$$

Assim, para descobrir $\boldsymbol{T}$ basta escrever os nós da malha original como uma interpolação linear dos nós da malha grosseira, No caso dessa malha, obtemos:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$x = \boldsymbol{T}u$$

Outras matrizes $\boldsymbol{T}$ são possíveis, mas como dito anteriormente a interpolação linear é simples é eficiente. Isso, somente quando a curva do erro é suave, dai a necessidade de eliminar as componentes de alta frequência do erro e_k.

A segunda matriz que precisamos é a matriz de restrição $\boldsymbol{R}$**,** também chamada de matriz de redução, a qual é responsável por transferir os vetores na malha original para a malha grosseira. Existem diferentes opções para a matriz $\boldsymbol{R}$, como por exemplo, a matriz de injeção que contém somente zeros e uns. Essa matriz define v_j simplesmente copiando os valores de u_{2j} nos mesmos pontos da malha original (Figura 20).

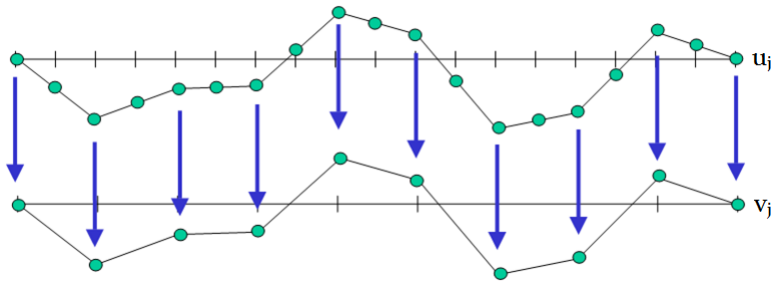

Figura 20 - Elemento v_j criado pela matriz de injeção.

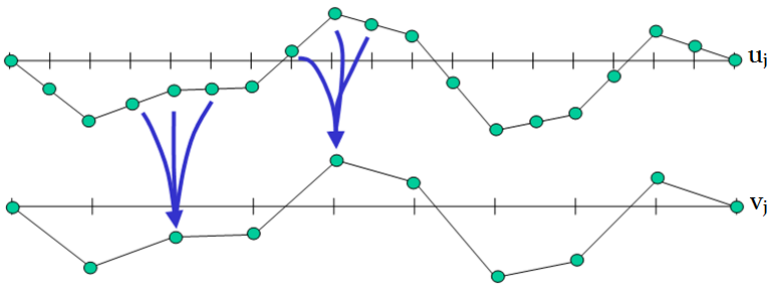

Figura 21 - Elemento v_j criado pela matriz completamente ponderada.

Outra opção é a chamada matriz completamente ponderada, a qual define v_j ponderando os valores de u_{2j-1}, u_{2j} e u_{2j+1} (Figura 21), dada por:

$$\boldsymbol{R} = \frac{1}{2}\boldsymbol{T}^T$$

O fator $\frac{1}{2}$ aparece, pois as soma das ponderações dever ser igual a um. Note que a soma das ponderações corresponde à soma dos elementos da coluna de $\boldsymbol{T}$.

O efeito da matriz $\mathbf{R}$ é de extrema importância, é ela a responsável por tornar o erro suave proveniente da malha original menos suave, resultado esse o qual é desejado. Para ilustrarmos o efeito da transformação provocada por $\boldsymbol{R}$, considere o caso em que $u_j = sen(2j\pi h)$ no intervalo $0 \leq x \leq 1$. Para uma malha unidimensional com sete nós temos que $h = \frac{\Delta x}{n+1} = \frac{1}{8}$ e então $u_j = sen\left(\frac{2j\pi}{8}\right)$. Utilizando a definição para a matriz completamente ponderada temos que:

$$v_m = \frac{1}{4}u_{j-1} + \frac{1}{2}u_j + \frac{1}{4}u_{j+1}$$

$$v_m = \frac{1}{4}sen\left(\frac{2(j-1)\pi}{8}\right) + \frac{1}{2}sen\left(\frac{2j\pi}{8}\right) + \frac{1}{4}sen\left(\frac{2(j+1)\pi}{8}\right)$$

Utilizando a relação do seno da soma de dois arcos, podemos mostrar que:

$$v_m = \left(\frac{2+\sqrt{2}}{4}\right)sen\left(\frac{2j\pi}{8}\right)$$

No entanto para a malha grosseira $m = \frac{1}{2}j$, que é consequência de $h_{esparsa} = 2h = \frac{1}{4}$ na malha grosseira. Assim, temos que:

$$v_m = \left(\frac{2+\sqrt{2}}{4}\right)sen\left(\frac{2m\pi}{4}\right)$$

ou seja, quando o problema é transferido da malha original para a malha grosseira, a frequência de u_j que na malha grosseira é dado v_m dobra, e o comportamento que antes era suave, deixa de ser tão suave. Deste modo, o erro é eliminado mais rapidamente pelos métodos iterativos estacionários. Graficamente podemos observar o comportamento de u_j e v_m na Figura 22:

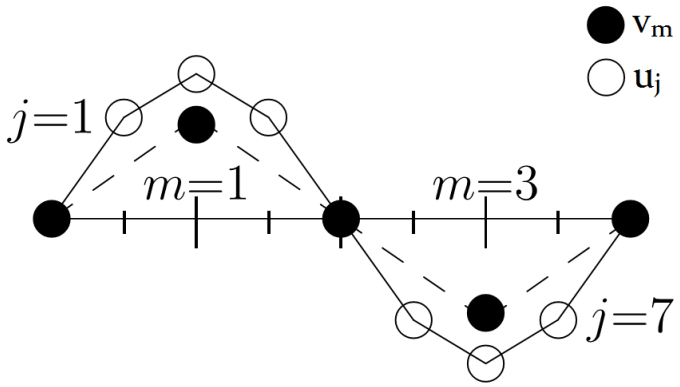

Figura 22 - Nós u_j na malha original e nós v_m na malha grosseira.

A última matriz a ser obtida é a matriz $\boldsymbol{A_{2h}}$. Ela é definida como $\boldsymbol{A_{2h} = RA_hT}$. Para entender o porquê, vamos considerar o processo desde o primeiro procedimento. No primeiro passo, obtemos o resíduo dado por (4.6):

$$r_h = \boldsymbol{A_h} e_h$$

O subscrito h indica, assim como antes, que esse é o resíduo obtido na malha original. No segundo passo r_h é reduzido, e então:

$$r_{2h} = \boldsymbol{R} r_h$$

$$r_{2h} = \boldsymbol{RA}_h e_h$$

No terceiro passo é resolvido o problema para o resíduo:

$$\boldsymbol{A}_{2h} e_{2h} = r_{2h}$$

substituindo r_{2h}, obtemos:

$$e_{2h} = \boldsymbol{A}_{2h}^{-1} r_{2h}$$

$$e_{2h} = \boldsymbol{A}_{2h}^{-1} \boldsymbol{RA}_h e_h$$

No quarto passo o erro é interpolado novamente para a malha original:

$$E_h = \boldsymbol{T} e_{2h}$$

$$E_h = \boldsymbol{TA}_{2h}^{-1} \boldsymbol{RA}_h e_h$$

Vamos supor que o mapeamento tenha sido trivial, tal que o processo tenha mapeado a malha original nela mesma, ou seja, não houve redução da malha. Como resultado, temos que $E_h = e_h$. O que só e verdade se:

$$\boldsymbol{TA}_{2h}^{-1} \boldsymbol{RA}_h = \boldsymbol{I}$$

também temos como consequência que as matrizes $\boldsymbol{R}$ e $\boldsymbol{T}$ são quadradas e de posto cheio, pois elas mapeiam n nós em n nós. Dessa forma, elas possuem inversas e podemos escrever:

$$\boldsymbol{T}\boldsymbol{A}_{2h}^{-1}\boldsymbol{R} = \boldsymbol{A}_h^{-1}$$

$$\boldsymbol{A}_{2h}^{-1} = \boldsymbol{T}^{-1}\boldsymbol{A}_h^{-1}\boldsymbol{R}^{-1}$$

$$\boldsymbol{A}_{2h}^{-1} = (\boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T})^{-1}$$

$$\boldsymbol{A}_{2h} = \boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T}$$

No caso onde a malha não é reduzida as matrizes $\boldsymbol{R}$ e $\boldsymbol{T}$, são as matrizes identidades, pois se não há mudança na malha $\boldsymbol{A}_{2h} = \boldsymbol{A}_h$. No caso geral, $\boldsymbol{A}_{2h}$ é dado pela relação acima e as matrizes $\boldsymbol{R}$ e $\boldsymbol{T}$ são matrizes retangulares.

Pra observarmos o efeito da redução da malha, considere um problema discretizado pela malha da Figura 18, cuja matriz original é dada por:

$$\boldsymbol{A}_h = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

Para a malha reduzida dada pela Figura 19 temos que $\boldsymbol{A}_{2h} = \boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T}$ resulta em:

$$\frac{1}{4}\begin{bmatrix} 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 1 \end{bmatrix}\frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{4h^2}\begin{bmatrix} 0 & 2 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 2 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 2 & 0 \end{bmatrix}\frac{1}{2}\begin{bmatrix} 1 & 0 & 0 \\ 2 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{8h^2}\begin{bmatrix} 4 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 4 \end{bmatrix} = \frac{1}{4h^2}\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} = \frac{1}{(2h)^2}\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$$

Esse resultado é exatamente igual ao esperado para o problema quando esse é discretizado por apenas três nós.

De maneira análoga a matriz $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A})$ que controla o decaimento do erro nos métodos estacionários, é possível definir para o método Multigrid uma matriz $\boldsymbol{M}'$ dada por $\boldsymbol{M}' = \boldsymbol{I} - \boldsymbol{S}$. Onde $\boldsymbol{S}$ é definido por:

$$E_h = \boldsymbol{T}\boldsymbol{A}_{2h}^{-1}\boldsymbol{R}\boldsymbol{A}_{\boldsymbol{h}}e_h$$

$$E_h = \boldsymbol{S}e_h; \quad \boldsymbol{S} = \boldsymbol{T}(\boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T})^{-1}\boldsymbol{R}\boldsymbol{A}_h$$

Uma importante propriedade dessa matriz é que $\boldsymbol{S}^2 = \boldsymbol{S}$:

$$\boldsymbol{S}^2 = \boldsymbol{T}(\boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T})^{-1}\overbrace{\boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T}(\boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T})^{-1}}^{\boldsymbol{I}}\boldsymbol{R}\boldsymbol{A}_h = \boldsymbol{T}(\boldsymbol{R}\boldsymbol{A}_h\boldsymbol{T})^{-1}\boldsymbol{R}\boldsymbol{A}_h$$

Essa propriedade é característica de uma matriz de projeção, pois uma vez projetado um vetor em espaço qualquer, projetar novamente esse vetor nesse espaço não altera o vetor, pois ele já pertence aquele espaço, ou seja, uma vez multiplicado $\boldsymbol{S}$ por um vetor, multiplicar o resultado por $\boldsymbol{S}$ não afeta em nada o resultado anterior. Essa propriedade da matriz $\boldsymbol{S}$ é interessante, e também nos indica o que está acontecendo no processo. O que ocorre, é que quando computamos $x_k + E_h$ no último passo do processo, a parte do erro correspondente à parte projetado na malha grosseira é corrigida. Consequentemente, temos que o erro e_k da solução corresponde somente àquela componente e do erro e_h que foi perdida na projeção (Figura 23), a qual é dada por:

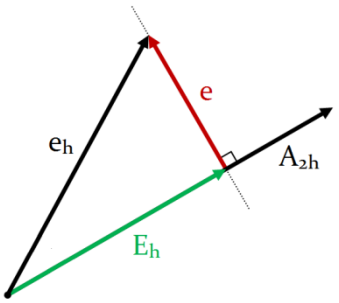

Figura 23 - Representação simplificada da projeção do erro.

$$\begin{aligned} e &= e_h - E_h \\ &= e_h - \boldsymbol{S}e_h \\ &= (\boldsymbol{I} - \boldsymbol{S})e_h \end{aligned}$$

$$e \quad = \boldsymbol{M}' e_h$$

Outra propriedade característica de uma matriz de projeção, a qual é resultado do fato de que $\boldsymbol{S}^2 = \boldsymbol{S}$, é que os autovalores de $\boldsymbol{S}$ são somente 1 ou 0. Consequentemente os de $\boldsymbol{M}'$ são 0 ou 1. Podemos demonstrar isso diagonalizando a matriz $\boldsymbol{S}$, ou seja, escrevendo ela na forma $\boldsymbol{S} = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E}$, onde $\boldsymbol{E}$ é a matriz que contém em suas colunas os autovetores de $\boldsymbol{S}$ e $\boldsymbol{\Lambda}$ a matriz diagonal que contém seus autovalores. Desse modo, temos:

$$\boldsymbol{S} = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E}$$

$$\boldsymbol{S}^2 = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E} = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E}\boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E} = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}^2\boldsymbol{E}$$

no entanto:

$$\boldsymbol{S} = \boldsymbol{S}^2$$

$$\boldsymbol{E}^{-1}\boldsymbol{\Lambda}\boldsymbol{E} = \boldsymbol{E}^{-1}\boldsymbol{\Lambda}^2\boldsymbol{E}$$

o que ocorre somente se:

$$\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^2$$

Assim, $\lambda_j(\boldsymbol{S})$ só pode ser 1 ou 0. Esse fato indica que no método Multigrid parte dos componentes do erro e_h são totalmente eliminados, os quais são aqueles relacionados aos autovalores de $\lambda_j(\boldsymbol{M}') = 0$ e outra parte continua inalterada, os quais são aqueles relacionados aos autovalores de $\lambda_l(\boldsymbol{M}') = 1$.

Essa é a principal razão do sucesso do método Multigrid, em um único processo é capaz de eliminar certos componentes do erro e_h. Desse modo, os diferentes ciclos têm como objetivo eliminar diferentes faixas de componentes do erro. Outra vantagem do processo é que o número de operações efetuadas em cada malha reduzida é também menor, devido à quantidade menor de dados, o que torna vantajosa a redução sucessiva. Para a aplicação do método Multigrid não é estritamente necessário conhecer a malha física que discretiza o problema. Quando a geometria é muito complicada ou quando só é conhecida a matriz, é utilizado o chamado método Multigrid algébrico, o qual imita a ideia de trabalhar com diferentes escalas do problema, mas lidando diretamente com $\boldsymbol{A}x = b$, e não com uma malha.

4.4 SUBESPAÇOS DE KRYLOV

O problema na forma matricial a ser resolvido é dado por $\boldsymbol{A}x = b$. Com o uso do pré-condicionador, o problema toma a forma $\boldsymbol{P}^{-1}\boldsymbol{A}x = \boldsymbol{P}^{-1}\boldsymbol{b}$, o qual pode ser escrito na forma recursiva dada por (4.2):

$$\boldsymbol{P}x_{k+1} = (\boldsymbol{P} - \boldsymbol{A})x_k + b$$

O erro da equação $\boldsymbol{A}x = b$ e dado pelo resíduo, o qual, como dito anteriormente é o erro da equação e não o erro de x_k. Dado pela equação (4.5):

$$r_k = b - \boldsymbol{A}x_k$$

Reescrevendo (4.2), podemos mostrar que:

$$\boldsymbol{P}x_{k+1} = \boldsymbol{P}x_k - \boldsymbol{A}x_k + b$$

$$\boldsymbol{P}x_{k+1} = \boldsymbol{P}x_k + (b - \boldsymbol{A}x_k) \qquad (4.7)$$

Substituindo (4.5) em (4.7), temos:

$$\boldsymbol{P}x_{k+1} = \boldsymbol{P}x_k + r_k$$

$$x_{k+1} = x_k + \boldsymbol{P^{-1}}r_k$$

ou seja, a cada iteração x_k é corrigido por $\boldsymbol{P^{-1}}r_k$.

Considere $\boldsymbol{P} = \boldsymbol{I}$, escolha essa feita somente para facilitar a compreensão do conceito dos subespaços de Krylov [25]. Considere três iterações por métodos estacionários, para um $x_0 = \{0\}$, e consequentemente $r_0 = b$. Na primeira iteração, temos de (4.2) que:

$$x_{k+1} = (\boldsymbol{I} - \boldsymbol{A})x_k + b$$

$$x_1 = (\boldsymbol{I} - \boldsymbol{A})x_0 + b$$

$$x_1 = b$$

para segunda iteração, temos:

$$x_2 = (\boldsymbol{I} - \boldsymbol{A})x_1 + b$$

$$x_1 = (\boldsymbol{I} - \boldsymbol{A})b + b = 2b - \boldsymbol{Ab}$$

e finalmente, para a terceira iteração:

$$x_3 = (\boldsymbol{I} - \boldsymbol{A})x_2 + b$$

$$x_2 = (\boldsymbol{I} - \boldsymbol{A})(2b - \boldsymbol{Ab}) + b = 3b - 3\boldsymbol{A}b + \boldsymbol{A}^2\boldsymbol{b}$$

A ideia de Krylov é a de que x_j pode ser escrito como uma combinação linear de $b, \boldsymbol{A}b, \boldsymbol{A}^2b, \ldots, \boldsymbol{A}^{j-1}b$ (j vetores), como visto nas iterações acima. Se x_j e então uma combinação desses vetores, é possível que exista uma melhor combinação desses vetores que resulte em um x_j mais próximo de $x = \boldsymbol{A}^{-1}b$ do que x_k obtido pelas iterações anteriormente.

A combinação linear das bases $b, \boldsymbol{A}b, \boldsymbol{A}^2b, \ldots, \boldsymbol{A}^{j-1}b$ formam o j-ésimo subespaço $\mathbb{K}_j$ de Krylov, o qual depende somente de $\boldsymbol{A}$ e b. Assim, podemos definir a matriz $\boldsymbol{K}_j$ como a matriz cujo espaço coluna gera o j-ésimo subespaço $\mathbb{K}_j$, ou seja, a matriz em que as colunas são essas bases.

$$\boldsymbol{K}_j = [b \quad \boldsymbol{A}b \quad \boldsymbol{A}^2b \quad \cdots \quad \boldsymbol{A}^{j-1}b]$$

Observe que enquanto $\boldsymbol{K}_j$ é a matriz, $\mathbb{K}_j$ é o subespaço gerado pelo espaço coluna da matriz $\boldsymbol{K}_j$, o qual contém as combinações lineares das colunas de $\boldsymbol{K}_j$.

No caso geral, o coeficiente $\boldsymbol{b}$, na verdade, não coincide com o lado direito da equação $\boldsymbol{A}x = b$, mas sim com resíduo inicial, que no caso de $x_0 = 0$, é dado por $r_0 = b$. Desta forma, para um x_0 qualquer, temos que o resíduo inicial é $r_0 = b - Ax_0$. Portanto, de modo mais geral definimos:

$$\boldsymbol{K}_j = [r_0 \quad \boldsymbol{A}r_0 \quad \boldsymbol{A}^2r_0 \quad \cdots \quad \boldsymbol{A}^{j-1}r_0]$$

A partir dessa ideia, diferentes métodos foram desenvolvidos para solucionar o problema $\boldsymbol{A}x = b$, cada um com um conceito diferente de qual combinação entre as bases é a melhor. Entre esses métodos estão:

- Método do Gradiente Conjugado (CG): A combinação é tal que o resíduo r_j é ortogonal a $\mathbb{K}_j$.
- Método dos Mínimos Resíduos (MINRES e GMRES): A combinação é tal que resíduo r_j tem norma mínima para x_j em $\mathbb{K}_j$.
- Método do Gradiente Biconjugado (BICG): A combinação é tal que o resíduo r_j é ortogonal a um espaço diferente $\mathbb{K}_j(\boldsymbol{A}^T)$.

- Método LQ Simétrico (SYMMLQ): A combinação é tal que o erro e_j tem norma mínima.

Em todos os casos é importante poder computar x_j o mais rápido possível. Isso ocorre principalmente quando é necessária pouca recorrência, ou seja, a utilização de poucos valores obtidos anteriormente.

Nesse trabalho serão discutidos somente os dois primeiros métodos. O método do gradiente conjugado (CG), pois é o que requer menor recorrência, mas que, no entanto sua aplicação é restrita a matrizes do tipo simétrica positiva definida e o Método dos Mínimos Resíduos (GMRES), pois é largamente utilizado para matrizes não simétricas, como é o caso das matrizes obtidas pelo método dos elementos de contorno.

4.4.1 PROJEÇÃO E GRAM-SCHMIDT

A projeção de um vetor b na direção de outro vetor a resulta em um vetor p, o qual é um múltiplo de a, ou seja, $p = xa$ (Figura 24). Quando um vetor é projetado na direção de outro, a componente desse vetor perpendicular à direção de projeção é perdida.

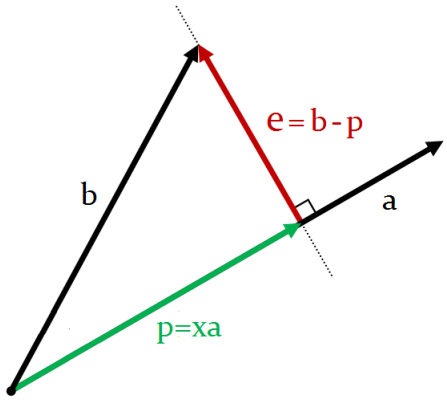

Figura 24 - Projeção do vetor b na direção do vetor a.

No caso da Figura, quando o vetor b é projetado na direção do vetor a a componente e do vetor b, perpendicular a a, é perdida. Podemos escrever o vetor b como a soma das suas duas componentes, a projetada p e a que foi perdida na projeção e, ou seja:

$$b = p + e$$

e então:

$$e = b - p$$

Utilizando o fato da perpendicularidade entre a e e, podemos escrever:

$$a^T e = 0$$

$$a^T (b - p) = 0$$

$$a^T (b - ax) = 0$$

$$a^T b - a^T ax = 0$$

$$x = \frac{a^T b}{a^T a}$$

e finalmente podemos obter a componente projetada p, dada por:

$$p = ax$$

$$p = a\frac{a^T b}{a^T a} = \boldsymbol{P}b; \;\; \boldsymbol{P} = \frac{aa^T}{a^T a}$$

A matriz $\boldsymbol{P}$ é chamada matriz de projeção, ela é responsável por projetar um vetor qualquer na direção do vetor a.

Observe que $\boldsymbol{P}$ é uma matriz simétrica, ou seja, $\boldsymbol{P^T} = \boldsymbol{P}$. Além disso, temos que $\boldsymbol{P^2} = \boldsymbol{P}$.

$$\boldsymbol{P^T} = \left(\frac{aa^T}{a^T a}\right)^T = \frac{aa^T}{a^T a} = \boldsymbol{P}$$

$$\boldsymbol{P^2} = \left(\frac{aa^T}{a^T a}\right)^2 = \frac{aa^T}{a^T a}\frac{aa^T}{a^T a} = \frac{a(a^T a)a^T}{(a^T a)a^T a} = \frac{(a^T a)}{(a^T a)}\frac{aa^T}{a^T a} = \frac{aa^T}{a^T a} = \boldsymbol{P}$$

Essas são propriedades que todas as matrizes de projeção apresentam.

A componente ortogonal de b, dada por e pode ser obtida por:

$$e = b - p$$

$$e = b - a\frac{a^T b}{a^T a}$$

Gram-Schmidt [31] então utilizou desse principio para ortogonalizar a base de um espaço vetorial. Desse modo, dado um conjunto de vetores v_i que formam uma base para um espaço vetorial qualquer, outra base de vetores u_i ortogonais para esse mesmo espaço pode ser obtida é eliminado de cada vetor v_i as suas projeções na direção dos demais vetores da base. Processo dado pela equação:

$$u_i = v_i + \sum_{j=1}^{k-1} u_j p_{ij}\,;\;\; p_{ij} = -\frac{{v_j}^T v_i}{{v_j}^T v_j}$$

O primeiro vetor da base ortonormal é $u_1 = v_1$, o segundo vetor é $u_2 = v_2 - u_1 p_{21}$ o qual é a parte de u_2 ortogonal a u_1, o terceiro vetor é $u_3 = v_3 - u_1 p_{31} - u_2 p_{32}$ o qual é a parte de v_3 ortogonal a u_1 é u_2 e assim por diante. Normalizando os vetores da base obtidos pelo método de Gram-Schmidt, podemos escrever a decomposição:

$$\boldsymbol{A = QR}$$

onde $\boldsymbol{A}$ é a matriz que possui em suas colunas os vetores u_j, $\boldsymbol{Q}$ é a matriz que têm em suas colunas os vetores da base ortonormal $q_{i=}\frac{u_i}{\|u_i\|}$ e $\boldsymbol{R}$ é um matriz triangular superior que guarda a transformação ocorrida durante o processo.

Observe que para v_1:

$$v_1 = u_1$$

$$v_1 = [q_1][\|u_1\|]$$

e para v_2:

$$u_2 = v_2 - u_1 p_{21}$$

$$v_2 = u_2 + u_1 p_{21}$$

$$v_2 = q_2\|u_2\| + q_1\|u_1\|p_{21}$$

$$v_2 = [q_1 \quad q_2]\begin{bmatrix}\|u_1\|p_{21}\\ \|u_2\|\end{bmatrix}$$

e finalmente para v_3:

$$u_3 = v_3 - u_1 p_{31} - u_2 p_{32}$$

$$v_3 = u_3 + u_1 p_{31} + u_2 p_{32}$$

$$v_3 = q_3\|u_3\| + q_1\|u_1\|p_{31} + q_2\|u_2\|p_{32}$$

$$v_3 = [q_1 \quad q_2 \quad q_3]\begin{bmatrix}\|u_1\|p_{31}\\ \|u_2\|p_{32}\\ \|u_3\|\end{bmatrix}$$

juntando em uma única matriz, temos na forma $\boldsymbol{A} = \boldsymbol{QR}$:

$$[v_1 \quad v_2 \quad v_3] = [q_1 \quad q_2 \quad q_3]\begin{bmatrix}\|u_1\| & \|u_1\|p_{21} & \|u_1\|p_{31}\\ 0 & \|u_2\| & \|u_2\|p_{32}\\ 0 & 0 & \|u_3\|\end{bmatrix}$$

4.4.2 ORTOGONALIZAÇÃO DE ARNOLDI

Embora $b, \boldsymbol{A}b, \boldsymbol{A^2}b, \ldots, \boldsymbol{A^{j-1}}b$ formem uma base para o j-ésimo subespaço $\mathbb{K}_j$ de Krylov, essa não é a melhor base para ser utilizada, pois essa não é uma base ortonormal. Assim, a melhor base a ser utilizada é a base $v_1, v_2, \ldots, v_{j-1}$ formada por vetores ortonormais. O método utilizado para a obtenção dos vetores dessa base é o método de Arnoldi [32], onde cada novo vetor v_j é obtido ortonormalizando $t = \boldsymbol{A}v_{j-1}$. O algoritmo do método de Arnoldi é dado abaixo:

1. Definir $v_1 = \frac{b}{\|b\|}$;
2. Para $j = 1{:}\, n-1$
3. $t = Av_j$
4. Para $i = 1{:}\, j$
5. $h_{ij} = {v_i}^T t$
6. $t = t - h_{ij}v_i$
7. Fim do ciclo para.
8. $h_{j+1,j} = \|t\|$
9. $v_{j+1} = \frac{t}{h_{j+1,j}}$
10. Fim do ciclo para.

No primeiro passo há apenas um único vetor na base. Assim, a única preocupação é em normalizar o mesmo.

No terceiro passo, um novo vetor, ainda não ortonormalizando, para a base é calculado. Observe que cada base do j-ésimo subespaço $\mathbb{K}_j$ de Krylov pode ser calculada multiplicando a base anterior por $\boldsymbol{A}$, ou seja, $v_j = \boldsymbol{A}v_{j-1}$.

No quinto passo, a multiplicação ${v_i}^T t$ nos fornece o comprimento, ou norma, da projeção do vetor t na direção do vetor q_i unitário.

No sexto passo, o vetor $h_{ij}v_i$, resultante da projetação do vetor t na direção v_i, é subtraído do vetor t, o que nos fornece a componente do vetor t ortogonal à v_i.

No nono passo, o vetor t já é ortogonal, para ser ortonormal basta dividir ele pela sua norma, o que é feito nesse passo.

O método de Arnoldi é bastante simples e é essencialmente baseado no conceito de Gram-Schmidt para o caso especial dos subespaços $\mathbb{K}_j$ de Krylov. Analogamente a decomposição $\boldsymbol{A} = \boldsymbol{QR}$ em Gram-Schmidt, o método de Arnoldi resulta na decomposição:

$$\boldsymbol{AV} = \boldsymbol{VH}$$

$$\begin{bmatrix} \boldsymbol{A}v_1 & \cdots & \boldsymbol{A}v_n \end{bmatrix} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdot & h_{1n} \\ h_{21} & h_{22} & \cdot & h_{2n} \\ 0 & h_{23} & \vdots & \vdots \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & h_{nn} \end{bmatrix}$$

onde $\boldsymbol{V}$ é a matriz que contém em suas colunas os vetores base v_j. A matriz $\boldsymbol{H}$ é formada por uma matriz triangular superior mais a diagonal logo abaixo da diagonal principal, chamada de matriz superior de Hessenberg. A matriz $\boldsymbol{H}$ apresenta esse formato porque $t_j = \boldsymbol{A}v_{j-1}$ é o vetor t_j não ortogonalizado contêm componentes nos vetores bases ortonormais $v_j, v_{j-1}, \ldots v_1$. Por exemplo, temos que para t_2:

$$t_2 = \boldsymbol{A}v_1 = v_1 h_{11} + v_2 h_{21}$$

o que pode ser escrito como:

$$v_2 = \frac{t_2 - v_1 h_{11}}{h_{21}}$$

O que corresponde ao que o algoritmo descrito anteriormente faz, onde a subtração corresponde ao sexto passo e a divisão ao nono. Da mesma relação para t_2 podemos escrever:

$$\boldsymbol{A}v_1 = v_1 h_{11} + v_2 h_{21} = [v_1 \quad v_2]\begin{bmatrix} h_{11} \\ h_{21} \end{bmatrix}$$

de modo geral, temos que:

$$\boldsymbol{AV}_k = \boldsymbol{V}_{k+1}\boldsymbol{H}$$

Quando $\boldsymbol{A} \in \mathbb{R}^{nxn}$ e $k > n$ todas as linhas da matriz $\boldsymbol{H}$ cujos índices são maiores que n, são nulas, pois a dimensão da matriz $\boldsymbol{A}$ é nxn. Deste modo, qualquer conjunto de n vetores v_k da base e capaz de gerar o espaço de dimensão nxn e qualquer vetor a mais na matriz $\boldsymbol{V}$ é uma combinação linear das bases anteriores, o que resulta em entradas nulas na matriz $\boldsymbol{H}$**.**

Observe que quando $k = n$, temos:

$$\boldsymbol{AV}_n = \boldsymbol{V}_{n+1}\boldsymbol{H}$$

$$\boldsymbol{AV}_n = \boldsymbol{V}_{n+1}\begin{bmatrix} h_{11} & h_{12} & \cdot & h_{1n} \\ h_{21} & h_{22} & \cdot & h_{2n} \\ 0 & h_{23} & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdot & h_{nn} \\ 0 & 0 & & 0 \end{bmatrix}$$

Como a última linha de $\boldsymbol{H}$ é nula, podemos excluir essa e devemos também excluir a última coluna de $\boldsymbol{V}_{n+1}$**,** pois esses são os elementos que multiplicam a última linha de $\boldsymbol{H}$**.** Assim, podemos escrever:

$$\boldsymbol{AV}_n = \boldsymbol{V}_{n+1}\boldsymbol{H}$$

$$\boldsymbol{AV}_n = \boldsymbol{V}_n\boldsymbol{H}$$

$$\boldsymbol{AV} = \boldsymbol{VH}$$

ou seja, a decomposição $\boldsymbol{AV} = \boldsymbol{VH}$ é um o caso particular de $\boldsymbol{AV}_n = \boldsymbol{V}_{n+1}\boldsymbol{H}$ quando a decomposição é completa, ou seja, quando são computados n vetores v_k da base para um matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$.

4.4.3 MÉTODO DO GRADIENTE CONJUGADO (CG)

O método do gradiente conjugado [33] é utilizado para a resolução de matrizes do tipo simétrica positiva definida. Uma matriz $\boldsymbol{A}$ é positiva definida quando sua forma quadrática, dada por $x^T \boldsymbol{A} x$, é maior que zero para qualquer x diferente do vetor nulo. Uma matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$ positiva definida apresenta as seguintes propriedades:

1. $\lambda_i > 0$
2. $\det(\boldsymbol{A}_k) > 0; \quad K = 1, \dots, n$
3. Pivôs da eliminação de Gauss > 0

De certa forma, todas essas propriedades são equivalentes, o que significa que quando uma é atendida, todas as outras também são. Quando uma matriz é positiva definida, o ponto crítico de sua forma quadrática é sempre um ponto de mínimo.

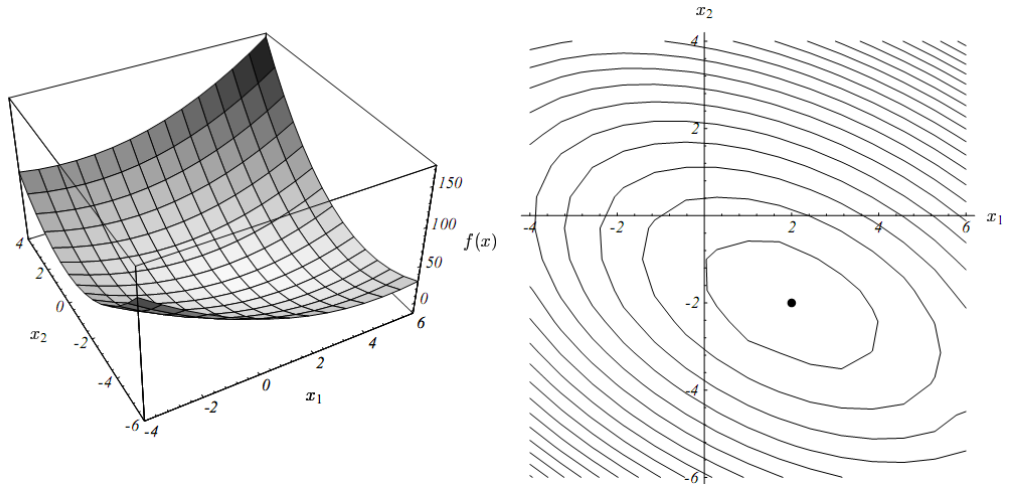

Figura 25 - Gráfico da forma quadrática de uma matriz $\boldsymbol{A} \in \mathbb{R}^{2x2}$ positiva definida (esquerda), e suas linhas de contorno (direita), onde o ponto corresponde ao ponto de mínimo de $f(x)$.

No método do gradiente conjugado, x_k deve ser escolhido tal que o resíduo $r_k = b - \boldsymbol{A} x_k$ seja ortogonal ao espaço $\mathbb{K}_k$, o qual é gerado pelos vetores $v_j, v_{j-1}, \dots v_1$ da base ortogonalizado pelo método de Arnoldi. Observe que o resíduo r_k apresenta o produto $\boldsymbol{A} x_k$. Como consequência o resíduo r_k pertence ao espaço $\mathbb{K}_{k+1}$. Além disso, como definido pelo método, ele deve ser ortogonal a $\mathbb{K}_k$. Desse modo, r_k só pode ser um múltiplo do vetor base v_{k+1}, pois esse pertence à $\mathbb{K}_{k+1}$ e também é ortogonal a $v_j, v_{j-1}, \dots v_1$, consequentemente a todo o espaço $\mathbb{K}_k$. O fato de r_k ser múltiplo de v_{k+1} significa também que cada resíduo é perpendicular ao resíduo anterior, pois:

$$r_k = a v_{k+1}$$

$$r_{k+1} = b v_{k+2}$$

$$r_k^T r_{k+1} = (av_{k+1})^T bv_{k+2} = ab[(v_{k+1})^T bv_{k+2}] = ab(0) = 0$$

onde a e b são constantes quaisquer. Assim, a primeira propriedade do método CG é:

$$r_i{}^T r_k = 0;\ \ para\ i \neq k \tag{4.8}$$

Como r_{k+1} é múltiplo de q_{k+2}, a diferença $r_{k+1} - r_k$ é também ortogonal ao subespaço $\mathbb{K}_k$, pois:

$$r_{k+1} - r_k = bv_{k+2} - av_{k+1}$$

e então:

$$(bv_{k+2} - av_{k+1})^T \boldsymbol{K}_k$$

$$(av_{k+2} - bv_{k+1})^T [v_1 \quad v_2 \quad v_3 \quad \cdots \quad v_k] = 0$$

isso ocorre porque as bases são ortogonais. Por outro lado, temos que $x_{i+1} - x_i$, para $i < k$, certamente pertence à $\mathbb{K}_k$. Assim, temos que:

$$(x_{i+1} - x_i)^T (r_{k+1} - r_k) = 0;\ \ para\ i < k \tag{4.9}$$

Podemos escrever $(r_{k+1} - r_k)$ como:

$$(r_{k+1} - r_k) = (b - \boldsymbol{A}x_{k+1}) - (b - \boldsymbol{A}x_k)$$

$$(r_{k+1} - r_k) = -\boldsymbol{A}(x_{k+1} - x_k) \tag{4.10}$$

e então, substituindo (4.10) em (4.9), temos:

$$(x_{i+1} - x_i)^T [-\boldsymbol{A}(x_{k+1} - x_k)] = 0$$

$$(x_{i+1} - x_i)^T \boldsymbol{A}(x_{k+1} - x_k) = 0;\ \ para\ i < k$$

ou seja, a variação Δx em cada iteração é A-ortogonal a variação Δx na iteração anterior.

Cada iteração do processo começa com um vetor x_k é uma direção de procura d_k para o próximo valor x_{k+1}, e termina com um novo valor x_{k+1} é uma nova direção de procura d_{k+1} para o próximo valor x_{k+2}.

O algoritmo para o método do gradiente conjugado é dado abaixo [33] [34]:

1. Escolher x_0;
2. Computar $r_0 = b - Ax_0$; Definir $d_0 = r_0$
3. Para $k = 0{:}n$
4. Calcular $\propto_k = {r_k^T r_k}/{d_k^T \boldsymbol{A} d_k}$
5. Calcular $x_{k+1} = x_k + \propto_k d_k$
6. Calcular $r_{k+1} = r_k - \propto_k \boldsymbol{A} d_k$
7. Calcular $\beta_{k+1} = {r_{k+1}^T r_{k+1}}/{r_k^T r_k}$
8. Calcular $d_{k+1} = r_k + \beta_{k+1} d_k$
9. Fim ciclo Para

Na quarta etapa do processo é computado o tamanho do passo $\propto_k$, ou seja, o quanto x_k será deslocado na direção d_k para obtermos x_{k+1}. Na quinta etapa, x_{k+1} é computado. Na sexta, é calculado o novo resíduo. A equação para o resíduo foi obtida utilizando a relação (4.10) e a equação para x_{k+1} da etapa anterior. A sétima e oitava etapas correspondem ao método de Gram-Schimdt, é computam a nova direção de procura d_{k+1} A-ortogonal a direção anterior.

O tamanho do passo $\propto_k$ é obtido minimizando a função $E(x) = \frac{1}{2} x^t A x - x^t b$ em função do tamanho do passo. Quando a matriz $\boldsymbol{A}$ é positiva definida à função $E(x)$ tem um ponto de mínimo, e então a função pode ser minimizada. Por isso, no método do gradiente conjugado a matriz $\boldsymbol{A}$ deve ser positiva definida. Primeiramente, tomaremos a derivada em função de x da função $E(x)$, obtendo:

$$E(x) = \frac{1}{2} x^t \boldsymbol{A} x - x^t b$$

$$\frac{dE(x)}{dx} = \frac{1}{2} \boldsymbol{A}^T x + \frac{1}{2} \boldsymbol{A} x - b$$

para uma matriz $\boldsymbol{A}$ simétrica temos que $\boldsymbol{A} = \boldsymbol{A}^T$, e então:

$$\frac{dE(x)}{dx} = \boldsymbol{A}x - b \tag{4.11}$$

O ponto crítico de $E(x)$, o qual é um ponto de mínimo, é obtido quando $\frac{dE(x)}{dx} = 0$. Assim, temos que:

$$\frac{dE(x)}{dx} = \boldsymbol{A}x - b = 0$$

e consequentemente:

$$\boldsymbol{A}x - b = 0$$

$$\boldsymbol{A}x = b$$

ou seja minimizar a função $E(x)$ é o mesmo que resolver o sistema linear $\boldsymbol{A}x = b$.

Da equação (4.11), temos que para um x_{k+1}qualquer que:

$$\frac{dE(x)}{dx_{k+1}} = \boldsymbol{A}x_{k+1} - b = -r_{k+1} \tag{4.12}$$

Na quinta etapa do método do gradiente conjugado, foi definido:

$$x_{k+1} = x_k + \propto_k d_k \tag{4.13}$$

e então, como dito anteriormente, para obtermos $\propto_k$ minimizamos $E(x)$ em função de $\propto_k$. Dessa forma temos:

$$\frac{dE(x)}{d \propto_k} = \left(\frac{dE(x)}{dx_{k+1}}\right)^T \frac{dx_{k+1}}{d \propto_k} = 0$$

De (4.12) e (4.13), temos que:

$$\frac{dE(x)}{d \propto_k} = r_{k+1}^T d_k = 0$$

transpondo o resultado podemos escrever:

$$r_{k+1}^T d_k = d_k^T r_{k+1} = 0$$

Substituindo (4.12) na equação anterior, obtemos:

$$d_k^T r_{k+1} = 0$$

$$d_k^T (b - \boldsymbol{A} x_{k+1}) = 0$$

e então substituindo (4.13) na relação acima, mostramos que:

$$d_k^T (b - \boldsymbol{A} x_{k+1}) = 0$$

$$d_k^T \big(b - \boldsymbol{A}(x_k + \propto_k d_k)\big) = 0$$

$$d_k^T (b - \boldsymbol{A} x_k - \propto_k \boldsymbol{A} d_k) = 0$$

$$d_k^T (b - \boldsymbol{A} x_k) - \propto_k d_k^T \boldsymbol{A} d_k = 0$$

$$d_k^T r_k = \propto_k d_k \boldsymbol{A} d_k$$

$$\propto_k = \frac{d_k^T r_k}{d_k^T \boldsymbol{A} d_k} \tag{4.14}$$

Os dois últimos passos do método correspondem ao processo de Gram-Schmidt, para que a nova direção de procura seja A-ortogonal a direção anterior. Assim, de Gram-Schmidt, podemos escrever:

$$d_k = r_k + \sum_{j=1}^{k-1} d_i \beta_{ik}\,; \tag{4.15}$$

multiplicando por ${r_k}^T$, temos:

$$d_k {r_k}^T = r_k {r_k}^T + \sum_{j=1}^{k-1} \beta_{jk} d_j \, {r_k}^T;$$

$$d_k {r_k}^T = r_k {r_k}^T + 0;$$

transpondo a relação, obtemos:

$${d_k}^T r_k = {r_k}^T r_k$$

O último termo é nulo, pois d_k é obtido por uma combinação dos resíduos $r_{k-1}, r_{k-2}, \ldots, r_0$ e como mostrado em (4.8), cada resíduo é perpendicular ao resíduo

anterior. Assim, r_k é ortogonal a $r_{k-1,}r_{k-2}, \dots, r_0$ é o produto ${r_k}^T r_i = 0$. Podemos então reescrever (4.14) como:

$$\propto_k = \frac{r_k^T r_k}{d_k^T \boldsymbol{A} d_k}$$

Observe que só foi possível encontra um $\propto_k$ que minimiza x_k pois a matriz $\boldsymbol{A}$ é uma matriz positiva definida, portanto a função $E(x)$ apresenta um ponto de mínimo.

Para calcular o termo β_k do último passo, consideramos novamente a relação (4.15).

Transpondo a equação (4.15) e multiplicando em ambos os lados por $\mathbf{A}d_i$, obtemos:

$${d_k}^T \mathbf{A} d_i = {r_i}^T \mathbf{A} d_i + \sum_{j=1}^{k-1} \beta_{jk} {d_j}^T \mathbf{A} d_i; \qquad (4.16)$$

no entanto, da equação (4.9), temos que:

$$(x_{i+1} - x_i)^T (r_{k+1} - r_k) = 0; \quad para\ i < k$$

Da terceira etapa do método do gradiente conjugado temos que:

$$r_{k+1} = r_r - \propto_k \boldsymbol{A} d_k \qquad (4.17)$$

$$r_{k+1} - r_k = - \propto_k \boldsymbol{A} d_k \qquad (4.18)$$

e de (4.13), temos:

$$x_{i+1} - x_i = \propto_k d_i \qquad (4.19)$$

Substituindo (4.18) e (4.19) em (4.9), obtemos:

$$\propto_k d_k^T (- \propto_k \boldsymbol{A} d_i) = 0$$

$$d_k^T \boldsymbol{A} d_i = 0; \quad para\ i < k$$

no entanto, observe que:

$$(d_k^T \boldsymbol{A} d_i)^T = 0$$

$$d_i^T \boldsymbol{A^T} d_K = d_i^T \boldsymbol{A} d_K = 0; \ para\ i < k$$

Pelo fato da matriz $\boldsymbol{A}$ ser simétrica, ou seja, $\boldsymbol{A} = \boldsymbol{A^T}$, temos então que:

$$d_k^T \boldsymbol{A} d_i = 0; \quad para\ i \neq k$$

Podemos agora reescrever a equação (4.16) como:

$$d_k{}^T \mathbf{A} d_i = r_i{}^T \mathbf{A} d_i + \sum_{j=1}^{k-1} \beta_{jk} d_j{}^T \mathbf{A} d_i$$

$$0 = r_i{}^T \mathbf{A} d_k + \beta_{ik} d_k{}^T \mathbf{A} d_k; \quad para\ \ i \neq k$$

$$\beta_{ik} = -\frac{r_i{}^T \mathbf{A} d_k}{d_k{}^T \mathbf{A} d_k}; \quad para\ \ i \neq k \tag{4.20}$$

Multiplicando a equação (4.17) por $r_i{}^T$, obtemos:

$$r_i{}^T r_{k+1} = r_i{}^T r_k - \propto_k r_i{}^T \boldsymbol{A} d_k$$

$$\propto_k r_i{}^T \boldsymbol{A} d_k = r_i{}^T r_k - r_i{}^T r_{k+1}$$

$$r_i{}^T \boldsymbol{A} d_k = \frac{1}{\propto_k} (r_i{}^T r_k - r_i{}^T r_{k+1})$$

no entanto, de (4.8), temos:

$$r_i{}^T r_k = 0; \ \ para\ i \neq k$$

como resultado, temos que:

$$\begin{cases} r_i{}^T \boldsymbol{A} d_k = \dfrac{1}{\propto_i} r_i{}^T r_i & para\ i = k \\ r_i{}^T \boldsymbol{A} d_k = -\dfrac{1}{\propto_{i-1}} r_i{}^T r_i & para\ i = k+1 \\ 0 & caso\ contrário \end{cases}$$

Por fim, de (4.20), podemos escrever:

$$\beta_{i,i-1} = \begin{cases} \dfrac{1}{\propto_{i-1}} \dfrac{r_i{}^T r_i}{d_{i-1}{}^T \mathbf{A} d_{i-1}} & para\ i = k+1 \\ 0 & caso\ contrário \end{cases}$$

$$\beta_{i,i+1} = \frac{d_{i-1}^T \boldsymbol{A} d_{i-1}}{r_{i-1}^T r_{i-1}} \frac{r_i{}^T r_i}{d_{i-1}{}^T \mathbf{A} d_{i-1}}$$

$$\beta_i = \beta_{i,i+1} = \frac{{r_i}^T r_i}{r_{i-1}^T r_{i-1}}$$

Definindo $i = k + 1$, obtemos a fórmula como visto anteriormente:

$$\beta_{k+1} = \frac{{r_{k+1}}^T r_{k+1}}{r_k^T r_k}$$

Somente foi possível encontra uma formula para β_{k+1} que apresenta pequena recorrência, pois a matriz $\boldsymbol{A}$ é uma matriz simétrica.

Outro modo de enxergar essa pequena recorrência é considerar a forma matricial da ortogonalização de Arnoldi, ou seja:

$$\boldsymbol{AV}_k = \boldsymbol{V}_{k+1} \tag{4.21}$$

quando a decomposição é completa, temos:

$$\boldsymbol{AV} = \boldsymbol{VH}$$

$$\boldsymbol{A} = \boldsymbol{VHV}^{-1} = \boldsymbol{VHV}^T$$

Temos que $\boldsymbol{V}^{-1} = \boldsymbol{V}^T$, pois $\boldsymbol{V}$ é uma matriz ortogonal. Tomando a transposta de $\boldsymbol{A}$**,** podemos escrever:

$$\boldsymbol{A}^T = (\boldsymbol{VHV}^T)^T = \boldsymbol{VH}^T\boldsymbol{V}^T$$

no entanto

$$\boldsymbol{A} = \boldsymbol{A}^T$$

$$\boldsymbol{VHV}^T = \boldsymbol{VH}^T\boldsymbol{V}^T$$

$$\boldsymbol{H} = \boldsymbol{H}^T$$

Deste modo, quando $\boldsymbol{A}$ é simétrica a matriz $\boldsymbol{H}$ também é simétrica. Somando agora o fato de que a matriz $\boldsymbol{H}$ é também uma matriz superior de Hessenberg, temos que $\boldsymbol{H}$ só pode ser uma matriz tridiagonal. Deste modo, só é necessário conhecer dois vetores $v_{k-1}\ e\ v_k$ para calcular um vetor v_{k+1} ortogonal não somente a eles, mas a todo o espaço $\mathbb{K}_k$.

Uma importante conclusão pode ser retirada da equação (4.12), dada por:

$$\frac{dE(x)}{dx_{k+1}} = \boldsymbol{A}x_{k+1} - b = -r_{k+1}$$

A direção de maior decaimento da função $E(x)$ é na direção do resíduo. Deste modo, em uma primeira análise, a melhor decisão parece ser definir $d_k = r_k$, pois a direção de maior decaimento é na direção do resíduo. No entanto, a convergência de x_k não dependente somente da escolha de uma boa direção a seguir, mas também do tamanho do passo $\propto_k$ que pode ser dado naquela direção. Quando definimos $d_k = r_k$ temos o chamado método do gradiente ou máximo declive.

Da equação (4.8) sabemos que um resíduo é ortogonal ao anterior e então quando utilizamos $d_k = r_k$ sabemos também que cada direção de procura será ortogonal a anterior. Esse movimento em direções perpendiculares torna o processo iterativo mais lento para o método do gradiente, pois quanto mais próximo da solução x menor é a diferença entre os gradientes e como resultado menor é o tamanho do passo $\propto_k$ a cada iteração, oque torna mais lento a convergência (Figura 26).

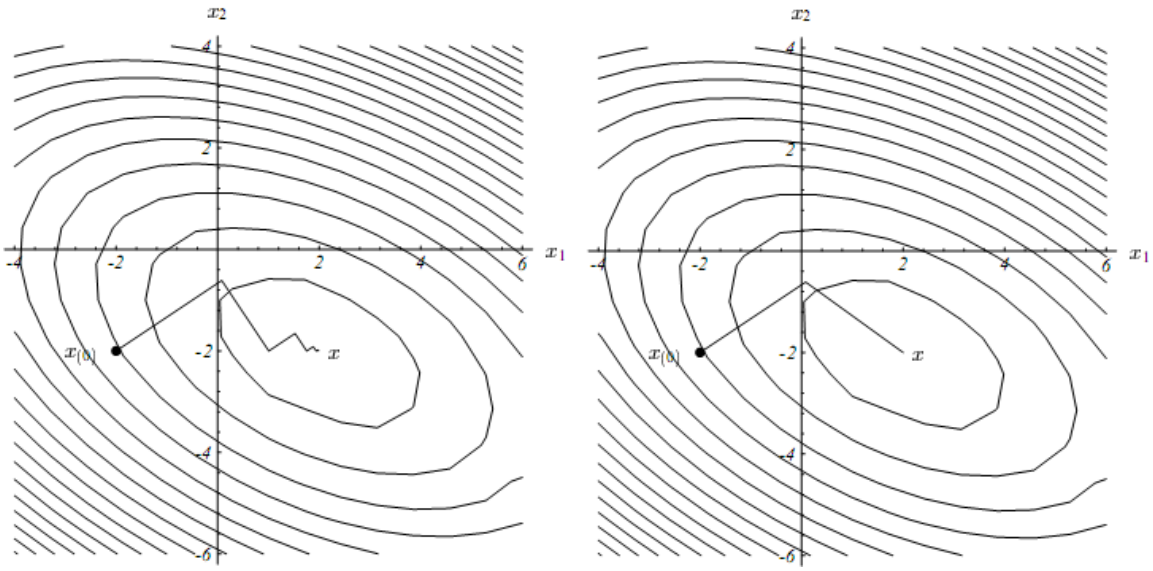

Figura 26 - Comparação entre o método do gradiente (direita) é o método do gradiente conjugado (esquerda). Cada segmento de reta representa uma iteração.

Por outro lado, no método do gradiente conjugado, a direção de procura é conjugada, ou seja, dada pela combinação por um fator β_{k+1} do resíduo r_k com a direção anterior. Assim, a nova direção de procura não é ortogonal a anterior. Desse modo o movimento em direções ortogonais é eliminado e a convergência ocorre mais rapidamente (Figura 26). Observe, no entanto que a direção inicial de procura de

ambos os métodos é a mesma, a qual é a direção do resíduo inicial, pois para um primeiro passo essa é a melhor direção para minimizar a função $E(x)$.

Embora o método CG utilize as propriedades dos subespaços de Krylov, em nenhuma de suas etapas ela as computa diretamente, essa é uma das vantagens desse método, pois o fato de não computar $\boldsymbol{K}_j$ torna o algoritmo mais rápido. Esse método é extremamente efetivo, mas limitado a sistemas lineares descritos por matrizes do tipo simétrica positiva definida.

4.4.4 MÉTODO DOS MÍNIMOS RESÍDUOS

Quando a matriz não é simétrica positiva definida a convergência não é garantida para o método do gradiente conjugado, pois o controle sobre as variáveis $\propto_k$ e β_k é perdido. Assim, é necessário utilizar outro método.

O método dos mínimos resíduos [32] escolhe x_k tal que a norma do resíduo r_k seja mínima. Para tal, o vetor x_k é escrito como uma combinação linear dos vetores v_k da base ortonormal obtida pela ortogonalização de Arnoldi para o espaço $\mathbb{K}_k$. Desse modo temos que:

$$x_k = \boldsymbol{V}_k y$$

e consequentemente:

$$y = \boldsymbol{V}_k^T x_k \tag{4.22}$$

O problema linear $\boldsymbol{A}x = b$ a ser resolvido é transformado para o problema $\boldsymbol{A}(x_0 + x_k) = b$, com valor inicial x_0. Desse modo temos que:

$$\boldsymbol{A}(x_0 + x_k) = b$$

$$\boldsymbol{A}x_0 + \boldsymbol{A}x_k = b$$

$$\boldsymbol{A}x_k = b - \boldsymbol{A}x_0$$

$$\boldsymbol{A}x_k = r_0$$

Assim, o problema se torna resolver $\boldsymbol{A}x_k = r_0$, para o resíduo inicial r_0. Multiplicando ambos os lados da equação por $\boldsymbol{V}_{k+1}^T$:

$$\boldsymbol{V}_{k+1}^T \boldsymbol{A} \overbrace{\boldsymbol{V}_k \boldsymbol{V}_k^T}^{\boldsymbol{I}} x_k = \boldsymbol{V}_{k+1}^T r_0$$

Podemos notar de (4.21) que $\boldsymbol{V}_{k+1}^{T}\boldsymbol{A}\boldsymbol{V}_{k}^{T} = \boldsymbol{H}$. Assim, reescrevendo a relação anterior considerando essa igualdade e substituindo (4.22), temos:

$$\boldsymbol{H}y = \boldsymbol{V}_{k+1}^{T}r_0 \tag{4.23}$$

E então, a norma do resíduo é dada por:

$$\|r_k\| = \|\boldsymbol{V}_{k+1}^{T}r_0 - \boldsymbol{H}y\|$$

Para representar x_k como uma combinação linear das bases ortonormais são necessárias os $k+1$ primeiros vetores de $\boldsymbol{V}$, e consequentemente k colunas e $k+1$ linhas de $\boldsymbol{H}$, pois de (4.21) sabemos que $\boldsymbol{V}_{k}^{T}\boldsymbol{H}\boldsymbol{V}_{k+1}^{T} = \boldsymbol{A}$.

$$\boldsymbol{V}\boldsymbol{H} = \begin{bmatrix} v_1 & \cdots & v_{k+1} \end{bmatrix} \begin{bmatrix} h_{11} & h_{12} & \cdot & h_{1k} \\ h_{21} & h_{22} & \cdot & h_{2k} \\ 0 & h_{32} & \vdots & \vdots \\ \cdot & \cdot & & \\ 0 & 0 & \cdot & h_{k+1,k} \end{bmatrix}$$

Desse modo, o problema é então encontrar um y que minimize a função:

$$\|r_k\| = \left\|\boldsymbol{V}_{k+1}^{T}r_0 - \boldsymbol{H}_{k+1,k}y\right\| \tag{4.24}$$

o qual é um problema de mínimos quadrados com $k+1$ equações e k variáveis.

A forma (4.23) da equação $\boldsymbol{A}x_k = r_0$ apresenta grandes vantagens. A primeira é mais obvia é que $\boldsymbol{H}$ é uma matriz de Hessenberg, é consequentemente apresenta uma grande quantidade de entradas nulas. A segunda, é que o produto $\boldsymbol{V}_{k+1}^{T}r_0$ é simplesmente um vetor $f = [\|r_0\|, 0, 0, 0, \dots, 0]^T$, pois o primeiro vetor da base do espaço $\mathbb{K}_k$ é o vetor unitário na direção r_0, e então $r_0 = v_1\|r_0\|$. Portanto, temos que o produto $\boldsymbol{V}_{k+1}^{T}r_0$ é dado por:

$$\boldsymbol{V}_{k+1}^{T}r_0 = \boldsymbol{V}_{k+1}^{T}v_1\|r_0\| = [v_1 \quad v_2 \quad \cdots \quad v_{k+1}]^T[v_1\|r_0\|] = \begin{bmatrix} \|r_0\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|r_0\|e_1; \quad e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.25}$$

pois as bases são ortonormais, ou seja, ${v_i}^T v = 0$ quando $i \neq j$, ou ${v_i}^T v_j = 1$ para $i = j$.

Os métodos mais utilizados então para resolver esse tipo de problema são o método dos resíduos mínimos para uma matriz $\boldsymbol{A}$ simétrica e o método dos resíduos

mínimos generalizados, para o caso de uma matriz $\boldsymbol{A}$ a qualquer. Em ambos os casos é desejável eliminar as entradas abaixo da diagonal da matriz $\boldsymbol{H}$, o que é feito transformando a matriz por rotação de Givens.

4.4.4.1 ROTAÇÃO DE GIVENS

A rotação de Givens [32] [35] é utilizada em análise numérica para eliminar certos coeficientes em matrizes ou vetores, tornando os mesmos nulos. Ela é representada pela matriz:

$$\boldsymbol{G}(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}$$

onde $c = \cos\theta$ e $s = sin\,\theta$. Os elementos não nulos da matriz são:

$$g_{kk} = 1 \quad para\ k \neq i,j$$

$$g_{ii} = c$$

$$g_{jj} = c$$

$$g_{ji} = \begin{cases} -s & se\ j > i \\ s & se\ j < i \end{cases}$$

$$g_{ij} = \begin{cases} s & se\ j > i \\ -s & se\ j < i \end{cases}$$

A matriz $\boldsymbol{G}$ é uma matriz de rotação e representa uma simples rotação de θ graus no plano (i,j) no sentido anti-horário.

Para eliminarmos a entrada a_{ij} da matriz, multiplicamos ela por $\boldsymbol{G}(i,j,\theta)$. É preciso então conhecer θ. Para tal, observe que quando a matriz $\boldsymbol{G}$ multiplica a matriz $\boldsymbol{A}$ pela esquerda, somente as linhas i e j da matriz $\boldsymbol{A}$ são modificadas, além disso, somente o que ocorre na coluna j é de interesse. Desse modo, para o caso onde $j < i$, o problema para θ pode ser resumido na forma:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_{jj} \\ a_{ij} \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

onde r é dado por $r = \sqrt{{a_{jj}}^2 + {a_{ij}}^2}$, o qual é a norma do vetor $[a_{jj} \quad a_{ij}]^T$. Para entendermos a definição de r, considere a analogia com um vetor a de comprimento r no plano xy. Quando esse vetor é rotacionado seu comprimento, ou seja, sua norma, não é afetada, pois somente sua direção sofre alteração. Considere então que esse vetor tenha sido rotacionado de forma que sua direção coincida com um dos eixos coordenados. Deste modo, uma de suas componentes é nula e a outra, para preservar a norma, deve ser r. É exatamente essa rotação que a matriz acima representa.

Para descobrir θ basta resolver o sistema. No entanto, não há interesse em θ diretamente, mas somente no seno e cosseno desse ângulo. Assim temos:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_{jj} \\ a_{ij} \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \longrightarrow \begin{cases} ca_{jj} - sa_{ij} = r \ (\boldsymbol{I}) \\ sa_{jj} + ca_{ij} = 0 \ (\boldsymbol{II}) \end{cases}$$

$$(\boldsymbol{I}) \longrightarrow \frac{r}{r}\left(ca_{jj} - sa_{ij}\right) = r$$

$$rca_{jj} - rsa_{ij} = r^2 = {a_{jj}}^2 + {a_{ij}}^2$$

uma opção então é:

$$c = \frac{a_{jj}}{r} \; s = -\frac{a_{ij}}{r}$$

Podemos checar em $(\boldsymbol{II})$ que essa é, de fato, a solução. Considere como exemplo a matriz abaixo:

$$A = \begin{bmatrix} 1 & 3 & 4 \\ -1 & 2 & 1 \\ 0 & 0 & 5 \end{bmatrix}$$

Queremos eliminar o coeficiente $a(2,1)$ da matriz para que a mesma seja uma matriz diagonal. Para tal, fazemos:

$$\boldsymbol{A' = GA}$$

$$r = \sqrt{{a_{jj}}^2 + {a_{ij}}^2} = \sqrt{{a_{11}}^2 + {a_{21}}^2} = \sqrt{1^2 + (-1)^2} = \sqrt{2}$$

$$c = \frac{a_{jj}}{r} = \frac{a_{11}}{r} = \frac{1}{\sqrt{2}}$$

$$s = -\frac{a_{ij}}{r} = -\frac{(-1)}{\sqrt{2}} = \frac{1}{\sqrt{2}}$$

$$\boldsymbol{A'} = \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 3 & 4 \\ -1 & 2 & 1 \\ 0 & 0 & 5 \end{bmatrix} = \begin{bmatrix} c+s & 3c-2s & 4c-s \\ s-c & 3s+2c & 4s+c \\ 0 & 0 & 5 \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 2 & 1 & 3 \\ 0 & 5 & 5 \\ 0 & 0 & 5\sqrt{2} \end{bmatrix}$$

Esse tipo de eliminação pode ser feita consecutivamente de tal forma que:

$$\boldsymbol{A'} = \boldsymbol{G}_n \ldots \boldsymbol{G}_3\boldsymbol{G}_2\boldsymbol{G}_1\boldsymbol{A}$$

A vantagem da rotação de Givens é que para cada matriz $\boldsymbol{G}$ é necessário computar quatro entradas, todas as outras são de uma matriz identidade. Além disso, por ser uma matriz de rotação ela é uma matriz ortogonal, ou seja, $\boldsymbol{G^T} = \boldsymbol{G}$. Desse modo, é fácil computar $\boldsymbol{A}$ novamente a partir de $\boldsymbol{A'}$:

$$\boldsymbol{A} = {\boldsymbol{G}_1}^T \ldots {\boldsymbol{G}_{n-2}}^T {\boldsymbol{G}_{n-1}}^T {\boldsymbol{G}_n}^T \boldsymbol{A'}$$

Observe que quando utilizamos rotação de Givens para transformar uma matriz $\boldsymbol{A}$ em uma matriz triangular superior $\boldsymbol{A'}$, o processo equivale ao processo de Gram-Schimdt, pois:

$$\boldsymbol{A} = \boldsymbol{QR} = \left({\boldsymbol{G}_1}^T \ldots {\boldsymbol{G}_{n-2}}^T {\boldsymbol{G}_{n-1}}^T {\boldsymbol{G}_n}^T\right)\boldsymbol{A'}$$

$$\boldsymbol{Q} = {\boldsymbol{G}_1}^T \ldots {\boldsymbol{G}_{n-2}}^T {\boldsymbol{G}_{n-1}}^T {\boldsymbol{G}_n}^T \longrightarrow Ortogonal$$

$$\boldsymbol{R} = \boldsymbol{A'} \longrightarrow Triangular\ Superior$$

4.4.4.2 MÉTODO DOS MINIMOS RESÍDUOS GENERALIZADOS (GMRES).

A ideia básica do método dos mínimos resíduos [36] é resolver o problema de mínimos quadrados (4.24) a cada iteração, onde $\boldsymbol{V}^T r_0$ é dado por (4.25).

$$mín_y \left\| \boldsymbol{V}_{k+1}^T r_0 - \boldsymbol{H}_{k+1,k} y \right\|; \quad \boldsymbol{V}_{k+1}^T b = \|r_0\|\, e_{1_{k+1,1}}$$

e então, de modo mais geral:

$$mín_y \left\| \|r_0\|\, e_{1_{k+1,1}} - \boldsymbol{H}_{k+1,k} y \right\|; \quad r_0 = b - \boldsymbol{A}x_0$$

Com o problema escrito nesse formato, o problema de mínimos quadrados da matriz $\boldsymbol{A} \in \mathbb{R}^{NxN}$ é substituído pelo problema equivalente com a matriz de Hessenberg $\boldsymbol{H} \in \mathbb{R}^{(k+1)xk}$. O problema de mínimos quadrados pode ser resolvido por qualquer

algoritmo a sua escolha. No nosso caso utilizaremos mais a frente à decomposição $\boldsymbol{QR}$ obtida pelas sucessivas rotações de Givens da matriz $\boldsymbol{H}$.

O algoritmo básico para o método GMRES pose ser resumido como [32] [34]:

1. Escolher x_0;
2. Computar $r_0 = b - Ax_0$
3. Calcular $v_1 = {}^{r_0}/_{\|r_0\|}$
4. Para $k = 1{:}n$
5. Arnoldi para determinar v_{k+1} é $\boldsymbol{H}_{k+1,k}$
6. Minimizar $\left\| \|r_0\| \, e_{1_{k+1,1}} - \boldsymbol{H}_{k+1,k} y \right\|$
7. Calcular $x_k = \boldsymbol{V}_k y$
8. Fim do ciclo Para
9. $x^* = x_0 + x_k$

O algoritmo acima apresenta, em geral, um bom desempenho, mas também apresenta algumas desvantagens:

- O tamanho das matrizes $\boldsymbol{V}$ e $\boldsymbol{H}$ aumentam a cada iteração, é consequentemente aumentam a capacidade computacional requerida.
- Depois de algumas iterações os vetores v_j deixam de ser exatamente ortogonais devido a erros aritméticas acumulados durante as iterações.

Para corrigir esse erro, o algoritmo do método GMRES é reiniciado a cada m iterações. O método é, então, chamado de GMRES(m). Valores típicos de m estão entre $m \approx 10{\sim}20$. Escolher um valor correto para m, tal que otimize o número de iterações e os requisitos computacionais utilizados é um grande desafio é praticamente um problema aparte [37] [38] [39] [40] [41].

Para o GMRES(m) o algoritmo anterior é levemente modificado [34]:

1. Escolher parâmetro de reinício m
2. Escolher x_0;
3. Para $i = 0{:}n$
4. Computar $r_i = b - Ax_i$
5. Calcular $v_1 = {}^{r_i}/_{\|r_i\|}$
6. Para $k = 1{:}m$
7. Arnoldi para determinar v_{k+1} é $\boldsymbol{H}_{k+1,k}$
8. Minimizar $\left\| \|r_i\| \, e_{1_{k+1,1}} - \boldsymbol{H}_{k+1,k} y \right\|$
9. Calcular $x_k = \boldsymbol{V}_k y$
10. Fim ciclo Para
11. $x^* = x_0 + x_k$
12. Fim do ciclo Para

A escolha de um valor apropriado para m é muito importante, uma má escolha pode levar GMRES(m) a não convergir. O número de iterações n é definido por uma precisão ε previamente definida. A condição é dada por:

$$\varepsilon < \frac{\|\boldsymbol{A}x_k - r_0\|}{\|b\|}$$

Da etapa 11 do processo foi definido $x_k = x^* - x_0$ e também sabemos que o resíduo inicial é dado por $r_0 = b - Ax_0$. Assim, substituindo na condição:

$$\varepsilon < \frac{\|\boldsymbol{A}(x^* - x_0) - b + Ax_0\|}{\|b\|}$$

$$\varepsilon < \frac{\|\boldsymbol{A}x^* - \boldsymbol{A}x_0 - b + \boldsymbol{A}x_0\|}{\|b\|}$$

$$\varepsilon < \frac{\|\boldsymbol{A}x^* - b\|}{\|b\|} = \varepsilon < \frac{\|r^*\|}{\|b\|}$$

Assim, quando mais próximo x^* está da solução exata, menor é o resíduo r^* é mais próximo x^* está de satisfazer a condição para ε.

4.4.4.3 Fatoração QR e mínimos quadrados

Usando Gram-Schmidt, podemos fatorar a matriz $\boldsymbol{A}$ na forma $\boldsymbol{A} = \boldsymbol{QR}$, onde $\boldsymbol{Q}$ é uma matriz ortonormal. Deste modo, o problema de minimizar $\|\boldsymbol{A}x - b\|$ pode ser substituído por:

$$\begin{aligned}\|\boldsymbol{A}x - b\| &= \|\boldsymbol{QR}x - b\| \\ &= (\boldsymbol{QRA}x - b)^{\mathbf{2}} = (\boldsymbol{QR}x - b)^T(\boldsymbol{QR}x - b) \\ &= [(\boldsymbol{QR}x)^T - b^T](\boldsymbol{QR}x - b) = (\boldsymbol{QR}x)^T\boldsymbol{QR}x - 2(\boldsymbol{QR}x)^Tb + b^Tb \\ &= (\boldsymbol{R}x)^T\boldsymbol{Q}^T\boldsymbol{QR}x - 2(\boldsymbol{R}x)^T\boldsymbol{Q}^Tb + b^Tb = (\boldsymbol{R}x)^T\boldsymbol{R}x - 2(\boldsymbol{R}x)^T\boldsymbol{Q}^Tb + b^Tb\end{aligned}$$

somando e subtraindo $(\boldsymbol{Q}^Tb)^T\boldsymbol{Q}^Tb$, podemos escrever:

$$\begin{aligned}\|\boldsymbol{A}x - b\| &= [(\boldsymbol{R}x)^T\boldsymbol{R}x - 2(\boldsymbol{R}x)^T\boldsymbol{Q}^Tb + (\boldsymbol{Q}^Tb)^T\boldsymbol{Q}^Tb] - (\boldsymbol{Q}^Tb)^T\boldsymbol{Q}^Tb + b^Tb \\ &= [(\boldsymbol{R}x)^T - (\boldsymbol{Q}^Tb)^T](\boldsymbol{R}x - \boldsymbol{Q}^Tb) - (\boldsymbol{Q}^Tb)^T\boldsymbol{Q}^Tb + b^Tb \\ &= (\boldsymbol{R}x - \boldsymbol{Q}^Tb)^T(\boldsymbol{R}x - \boldsymbol{Q}^Tb) - (\boldsymbol{Q}^Tb)^T\boldsymbol{Q}^Tb + b^Tb \\ &= (\boldsymbol{R}x - \boldsymbol{Q}^Tb)^2 - (\boldsymbol{Q}^Tb)^2 + b^2 = \|\boldsymbol{R}x - \boldsymbol{Q}^Tb\| - \|\boldsymbol{Q}^Tb\| + \|b\|\end{aligned}$$

Assim, o problema de minimizar $\|\boldsymbol{A}x - b\|$, é transformado no problema de minimizar $\|\boldsymbol{R}x - \boldsymbol{Q}^Tb\|$, pois somente essa componente depende de x. É fácil observar que quando $\boldsymbol{R}$ é uma matriz de posto cheio, a norma $\|\boldsymbol{R}x - \boldsymbol{Q}^Tb\|$ vai ser mínima quando:

$$\boldsymbol{R}x - \boldsymbol{Q}^Tb = 0$$

$$x = \boldsymbol{R}^{-1}\boldsymbol{Q}^Tb$$

Numericamente não há interesse em inverter $\boldsymbol{R}$**,** então resolvemos:

$$\boldsymbol{R}x = \boldsymbol{Q}^Tb \tag{4.26}$$

Observe que $\boldsymbol{R}$ é uma matriz triangular superior, então a equação pode ser resolvida por substituição regressiva.

Analogamente, para o problema de minimização de $\left\| \|r_0\| \, e_{1_{k+1,1}} - \boldsymbol{H}_{k+1,k} y \right\|$, podemos escrever:

$$\left\| \|r_0\| \, e_{1_{k+1,1}} - \boldsymbol{H}_{k+1,k} y \right\| = \left\| -\boldsymbol{H}_{k+1,k} y + \|r_0\| \, e_{1_{k+1,1}} \right\|$$

fazendo uma equivalência com a forma $\|\boldsymbol{A}x - b\|$, observamos que:

$$\left\| -\boldsymbol{H}_{k+1,k} y + \|r_0\| \, e_{1_{k+1,1}} \right\| \sim \|\boldsymbol{A}x - b\| \begin{cases} \boldsymbol{A} \sim -\boldsymbol{H}_{k+1,k} \\ x \sim y \\ b \sim \|r_0\| \, e_{1_{k+1,1}} \end{cases}$$

e então a forma equivalente de (4.26) para o problema $\left\| -\boldsymbol{H}_{k+1,k} y + \|r_0\| \, e_{1_{k+1,1}} \right\|$ e dada por:

$$\boldsymbol{R}y = \boldsymbol{Q}^T \|r_0\| \, e_{1_{k+1,1}}$$

Pela rotação de Givens temos que:

$$\boldsymbol{H}_{k+1,k}{}' = \boldsymbol{G}_n \dots \boldsymbol{G}_3 \boldsymbol{G}_2 \boldsymbol{G}_1 \boldsymbol{H}_{k+1,k}$$

$$\boldsymbol{Q}^T = \boldsymbol{G}_n \dots \boldsymbol{G}_3 \boldsymbol{G}_2 \boldsymbol{G}_1$$

$$\boldsymbol{R} = \boldsymbol{H}_{k+1,k}{}'$$

substituindo, temos que:

$$\boldsymbol{H}_{k+1,k}{}' y = (\boldsymbol{G}_n \dots \boldsymbol{G}_3 \boldsymbol{G}_2 \boldsymbol{G}_1) \|r_0\| \, e_{1_{k+1,1}} \qquad (4.27)$$

A matriz $\boldsymbol{H}_{k+1,k}{}'$ não é uma matriz quadrada, mas sim uma matriz retangular e matrizes retangulares não têm posto cheio. Deste modo, o problema de minimizar (4.27) é transformado em um problema de mínimos quadrados, pois não há solução exata.

O método dos mínimos quadrados tem como objetivo minimizar o resíduo $r = \boldsymbol{A}x - b$ quando a matriz $\boldsymbol{A}$ não é uma matriz de posto cheio. Mais especificamente, quando $\boldsymbol{A}$ possui mais linhas do que colunas, ou seja, mais equações do que incógnitas. Nesse caso, não existe uma solução x para todo $\boldsymbol{A}x = b$ e a solução aproximada mais adequada e aquela que minimiza o resíduo $\boldsymbol{A}x - b$.

Para uma matriz $\boldsymbol{A} \in \mathbb{R}^{nxm}$ de posto n, onde $n > m$, podemos computar a decomposição $\boldsymbol{QR}$, obtendo:

$$\boldsymbol{A}_{nxm} = \boldsymbol{Q}_{nxn}\boldsymbol{R}_{nxm} = \begin{bmatrix}\boldsymbol{Q}^{l}_{nxm} & \boldsymbol{Q}^{r}_{nx(n-m)}\end{bmatrix}\begin{bmatrix}\boldsymbol{R}^{u}_{mxm} \\ \boldsymbol{0}_{(n-m)xm}\end{bmatrix}$$

$$\boldsymbol{A}_{nxm} = \boldsymbol{Q}^{l}_{nxm}\boldsymbol{R}^{u}_{mxm}$$

Como todas as colunas de $\boldsymbol{A}$ são independentes, a matriz $\boldsymbol{Q}$ é uma matriz cheia, pois as colunas de $\boldsymbol{Q}$ são obtidas por uma combinação linear dessas colunas e, por serem independentes, não há combinação das colunas que resulte no vetor nulo. Por outro lado, a matriz de zeros de dimensão $\mathbb{R}^{(n-m)xm}$ aparece devido à dependência linear entre as linhas da matriz $\boldsymbol{A}$, observe que $n - m$ é o número de linhas linearmente dependentes, o qual coincide com o número de linhas nulas na matriz $\boldsymbol{R}$.

A decomposição da matriz $\boldsymbol{A}$ nessa forma apresenta duas vantagens. A primeira é que a matriz $\boldsymbol{Q}^{l}_{nxm}$, embora retangular, ainda satisfaz $\boldsymbol{Q}^{l}_{nxm}{\boldsymbol{Q}^{l}_{nxm}}^{T} = I$ por ser uma matriz ortogonal. A segunda é que $\boldsymbol{R}^{u}_{mxm}$ é uma matriz quadrada de posto cheio e, consequentemente, inversível.

Podemos agora reescrever, o problema $\boldsymbol{A}x = b$, para uma matriz $\boldsymbol{A} \in \mathbb{R}^{nxm}$ de posto n, onde $n > m$, como:

$$\boldsymbol{A}_{nxm}x = b$$

$$\boldsymbol{Q}^{l}_{nxm}\boldsymbol{R}^{u}_{mxm}x = b$$

$$x = {\boldsymbol{R}^{u}_{mxm}}^{-1}{\boldsymbol{Q}^{l}_{nxm}}^{T}b$$

a qual é a solução do problema de mínimo quadrados. Como anteriormente, não há numericamente interesse em inverter $\boldsymbol{R}$, então resolvemos:

$$\boldsymbol{R}^{u}_{mxm}x = {\boldsymbol{Q}^{l}_{nxm}}^{T}b$$

Para o problema (4.27) da matriz ${\boldsymbol{H}_{k+1,k}}'$, temos que somente a última linha é linearmente dependente, na verdade essa é uma linha de zeros, pois seu único elemento $h_{k+1,k}$ foi eliminado pela rotação de Givens. Assim temos que:

$$\boldsymbol{R}^{u}_{mxm} = {\boldsymbol{H}_{k,k}}'$$

Como somente a última linha de ${\boldsymbol{H}_{k+1,k}}'$ é eliminada, temos que para $\boldsymbol{Q}^{l}_{nxm}$ basta eliminar sua última coluna, o que será representado da forma:

$$\boldsymbol{Q}^{l}_{nxm} = (\boldsymbol{G}_n \dots \boldsymbol{G}_3\boldsymbol{G}_2\boldsymbol{G}_1)\|r_0\|\, {e_1}_{k,1}$$

Eliminar a última linha de e_1 equivale a eliminar a última coluna do produto $\boldsymbol{Q}_{nxm}^{l}\, e_1$. Finalmente, a solução para a equação (4.27) é dada por:

$$\boldsymbol{H}_{k,k}{}'y = (\boldsymbol{G}_n \ldots \boldsymbol{G}_3\boldsymbol{G}_2\boldsymbol{G}_1)\|r_0\|\, e_{1_{k,1}}$$

o que é feito por substituição regressiva.

4.4.4.4 MÉTODO GMRES PRÉ-CONDICIONADO

De forma geral, um pré-condicionador $\boldsymbol{P}$ pode ser utilizando tanto multiplicando a direita, quanto à esquerda da matriz $\boldsymbol{A}$ [34]. Até esse momento, foi utilizado somente o pré-condicionador à esquerda. Quanto ele é à esquerda obtemos a equação (4.2), a qual é dada por:

$$\boldsymbol{P}x_{k+1} = (\boldsymbol{P} - \boldsymbol{A})x_k + b$$

Considere a forma exata da equação recursiva dada acima. Multiplicando ela por $\boldsymbol{P}^{-1}$, obtemos:

$$x = (\boldsymbol{I} - \boldsymbol{P}^{-1}\boldsymbol{A})x + \boldsymbol{P}^{-1}b$$

$$x = x - \boldsymbol{P}^{-1}\boldsymbol{A}x + \boldsymbol{P}^{-1}b$$

$$\boldsymbol{P}^{-1}\boldsymbol{A}x = \boldsymbol{P}^{-1}b \qquad (4.28)$$

ou seja, utilizar o pré-condicionador pela esquerda é o mesmo que multiplicar $\boldsymbol{A}$ pela esquerda e embora b também seja multiplicado, isso não ocorre no caso a direita. É fácil observar que para x_0, o resíduo inicial é dado por:

$$\boldsymbol{P}^{-1}\boldsymbol{A}x_0 + r_0 = \boldsymbol{P}^{-1}b$$

$$r_0 = \boldsymbol{P}^{-1}(b - \boldsymbol{A}x_0)$$

e então, a nova matriz $\boldsymbol{K}_j$, cujo espaço coluna gera o j-ésimo subespaço $\mathbb{K}_j$, é dada por:

$$\boldsymbol{K}_j = [r_0 \quad \boldsymbol{P}^{-1}\boldsymbol{A}r_0 \quad (\boldsymbol{P}^{-1}\boldsymbol{A})^2 r_0 \quad \cdots \quad (\boldsymbol{P}^{-1}\boldsymbol{A})^{j-1} r_0]$$

Definido o novo resíduo inicial r_0, esse é utilizado para calcular os novos vetores da base ortogonal da matriz $\boldsymbol{K}_j$. Para resolver o problema de minimização para x_k com o pré-condicionador pela esquerda, podemos utilizar o mesmo algoritmo

descrito anteriormente para o caso sem um pré-condicionador. No entanto, as matrizes $\boldsymbol{Q}$ e $\boldsymbol{H}$ serão diferentes, pois $\boldsymbol{K}_j$ não é o mesmo.

No caso da utilização de um pré-condicionador à direita de x_k, podemos escrever análoga à equação (4.28), a relação:

$$\boldsymbol{A}\overbrace{\boldsymbol{P}^{-1}\boldsymbol{P}}^{\boldsymbol{I}}x = b$$

O problema linear a ser resolvido para a variável x, é substituído por um problema para a variável z, tal que:

$$\boldsymbol{A}\boldsymbol{P}^{-1}z = b;\ z = \boldsymbol{P}x$$

é possível observar que utilizar o pré-condicionador pela direita é o mesmo que multiplicar $\boldsymbol{A}$ pela direita. Para um pré-condicionador à direita, o resíduo inicial é o mesmo do caso sem um pré-condicionador, pois o lado direito da equação não foi alterado. A nova matriz $\boldsymbol{K}_j$ é dada por:

$$\boldsymbol{K}_j = [r_0 \quad \boldsymbol{A}\boldsymbol{P}^{-1}r_0 \quad (\boldsymbol{A}\boldsymbol{P}^{-1})^2 r_0 \quad \cdots \quad (\boldsymbol{A}\boldsymbol{P}^{-1})^{j-1}r_0]$$

O resíduo inicial r_0, é utilizado para calcular os novos vetores da base ortogonal da matriz $\boldsymbol{K}_j$. Para resolver o problema de minimização para x_k com o pré-condicionador pela direita, podemos utilizar o mesmo algoritmo descrito anteriormente para o caso sem um pré-condicionador. Mas é necessário fazer uma única mudança, pois dessa vez o problema é resolvido para z. Assim, na última etapa x^* deve ser definido como:

$$x^* = x_0 + x; \quad x = \boldsymbol{P}^{-1}z$$

$$x^* = x_0 + \boldsymbol{P}^{-1}z; \ \ z = \boldsymbol{V}_n y$$

$$x^* = x_0 + \boldsymbol{P}^{-1}\boldsymbol{V}_n y$$

Uma possível opção a ser utilizada como pré-condicionador, no caso de matrizes aproximadas pelo método ACA, é uma aproximação ACA de menor posto dessa mesma matriz.

5 DESEMPENHO DOS MÉTODOS ITERATIVOS

Nessa seção, o desempenho dos diferentes métodos iterativos para a resolução do problema $\boldsymbol{A}x = b$, discutidos anteriormente, são avaliados. Foram considerados principalmente na avaliação, o tempo de execução e o uso de memória. Além desses, foram também computados o número de iterações até a convergência de x e o comportamento do erro, dado pelo resíduo relativo, definido por:

$$Erro = \frac{\|\boldsymbol{A}x_k - r_0\|}{\|b\|} \tag{5.1}$$

Os diferentes algoritmos foram construídos na linguagem MATLAB (.m) e estão disponíveis nos anexos I a VII. Na comparação dos diferentes métodos, foi utilizado um mesmo problema de condução de calor, avaliado repetidamente com diferentes malhas. O problema é dado por uma placa quadrada de lado $1m$, e condutividade térmica $k = 1\frac{w}{mK}$ isolada nas laterais superior e inferior e com temperatura $0K$ e $1K$ nas laterais, esquerda e direita respectivamente (Figura 27).

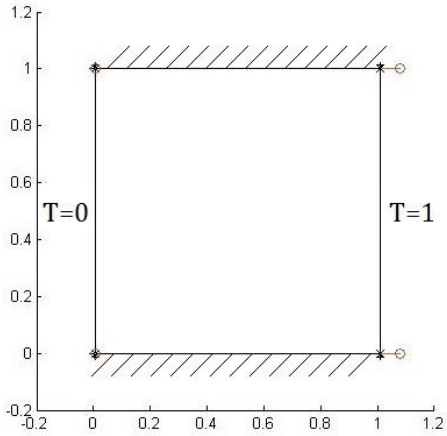

Figura 27 - Problema de condução de calor analisado pelos métodos iterativos.

Esse problema foi escolhido por duas razões principais. Em primeiro lugar, a solução exata desse problema é conhecida, e então, é possível facilmente comparar a solução exata com a obtida pelo método iterativo. Isso foi utilizado para validar a solução obtida pelos algoritmos escritos para os diferentes métodos. Finalmente, por ser uma geometria simples, não é preciso ler o arquivo da malha quando é utilizado o método Multigrid. Uma vez que é conhecido o processo de construção da malha, podemos escrever de forma direta uma fórmula para a matriz de interpolação $\boldsymbol{T}$ e para a matriz de restrição $\boldsymbol{R}$.

A matriz $\boldsymbol{A}$ e o vetor b do problema de condução de calor foram obtidos por um programa que utiliza o método de elementos de contorno para discretrizar o problema. A rotina de programação, não incluído nesse trabalho, utiliza elementos lineares contínuos e foi utilizado para gerar as matrizes $\boldsymbol{G}$ e $\boldsymbol{H}$ e montar a matriz $\boldsymbol{A}$ e o vetor b. Já a resolução do problema $\boldsymbol{A}x = b$, foi feita pelas as rotinas nos anexos I a VII

A solução analítica do problema da Figura 27 é obtida considerando as condições de contorno $T(0) = 0$ e $T(1) = 1$ e a equação de Laplace. Deste modo, temos:

$$\nabla^2 T = \frac{\mathrm{d}^2 T(x)}{dx} = 0$$

O problema é função de x somente, pois a geometria é simétrica e constante em relação à y e a condição de isolamento das paredes permite fluxo de calor somente na direção x. Assim, só há variação da temperatura nessa direção. Agora, Integrando a equação duas vezes, obtemos:

$$x = ax + b$$

e das condições de contorno, mostramos que:

$$a = 1; \quad b = 0$$

e então, a solução e dada por:

$$T(x) = x$$

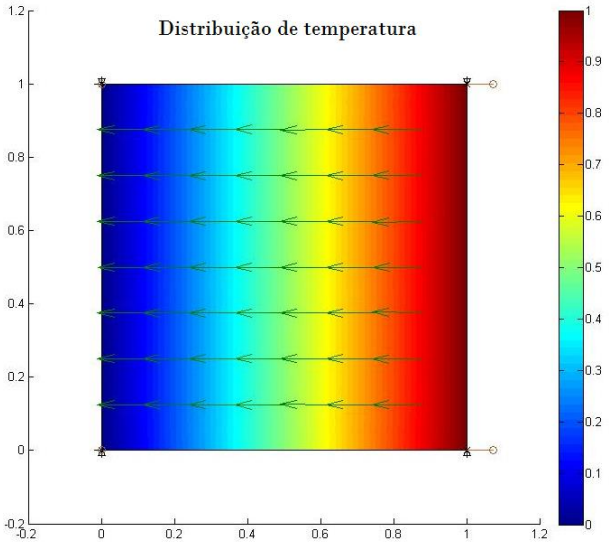

Figura 28- Solução do problema de condução de calor da Figura 27.

5.1 COMPARAÇÃO ENTRE OS MÉTODOS ESTACIONÁRIOS

O problema de condução de calor da Figura 27 foi analisado para onze malhas diferentes, cada uma com uma quantidade diferente de nós.

Tabela 1 - Número de nós das diferentes malhas.

Malha	Número de Nós	Malha	Número de Nós
1	4	6	128
2	8	7	256
3	16	8	512
4	32	9	1024
5	64	10	2048
		11	4096

Essa quantidade de nós foi escolhida devido à formulação adotada no método Multigrid, a sua razão será explicada mais adiante.

5.1.1 O FATOR ω DO MÉTODO DE JACOBI PONDERADO

Para que ocorra uma rápida convergência de x no método de Jacobi ponderado, é importante conhecer o fator ω apropriado, onde $0 < \omega < 1$. Para o problema da Figura 27, o fator ω que minimiza o número de iterações para as diferentes malhas (Tabela 4) foi obtido observando o número de iterações necessárias para atingir um erro máximo permitido, definido por (5.1), de 10^{-6}, variando ω entre $0 < \omega \leq 1$. Assim, o fator ω escolhido foi aquele que resultou no menor número de iterações para convergência. O número de iterações k foi delimitado a um máximo de $k = 1000$.

Tabela 2 - Fator ω que minimiza o número de iterações k no método Jacobi Ponderado e sua faixa de convergência.

Malha	$\boldsymbol{\omega}$	$\boldsymbol{k}$	**Malha**	$\boldsymbol{\omega}$	$\boldsymbol{k}$
1	$0{,}66$	3	*6*	$0{,}19\ (0{,}04 \geq \omega \geq 0{,}20)$	249
2	$0{,}85 - 0{,}86$	25	*7*	$0{,}10\ (0{,}04 \geq \omega \geq 0{,}11)$	489
3	$0{,}69 - 0{,}70\ (\omega \geq 0{,}82)$	41	*8*	$0{,}06^{*}$	867
4	$0{,}49\ (0{,}03 \geq \omega \geq 0{,}55)$	70	*9*	**	$\sim$
5	$0{,}31\ (0{,}04 \geq \omega \geq 0{,}34)$	129	*10*	**	$\sim$
			11	**	$\sim$

Para o intervalo de ω entre parêntesis não há convergência.

* Converge somente para esse valor de ω

* * O método diverge.

Da Tabela 4 é possível observar que quanto maior o número de nós, e consequentemente a matriz, menores são os valores de ω que otimizam o processo iterativo e menor é o intervalo onde há convergência. Para matrizes grandes, o número de iterações é muito elevado e a convergência de x é extremamente dependente do fator ω, o que torna o método de Jacobi ponderado não indicado a matrizes de larga escala. Da Tabela 4, podemos notar também que somente as malhas 1 e 2 não têm um limite superior de ω para convergência. No entanto, quando $\omega = 1$, temos o método de Jacobi. Assim, temos que o método de Jacobi só converge para as malhas 1 e 2.

A Figura 29 mostra o comportamento do número de iterações para diferentes valores de ω para o problema discretizado pela malha 4. Foi considerado um erro máximo permitido de $0{,}015$. Para valores de $\omega \geq 0{,}55$ não há convergência.

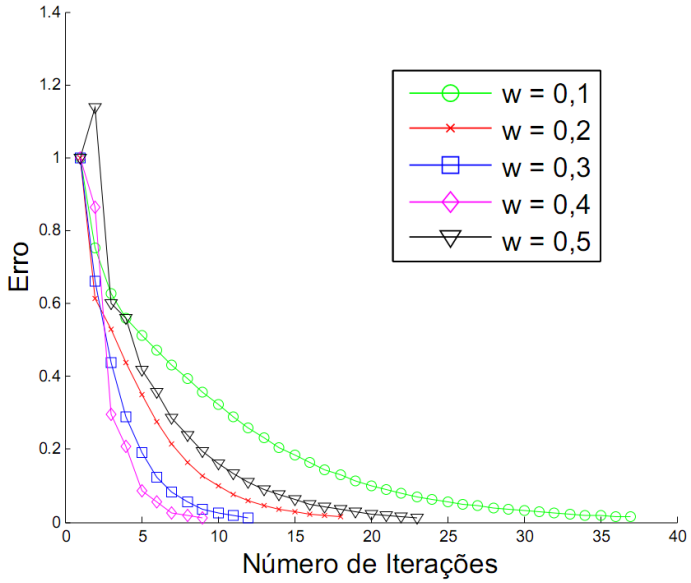

Figura 29 - Número de iterações para a malha 4 para diferentes fatores w no método Jacobi ponderado.

Foi considerado um erro de 0.015, pois o decaimento do erro após certo intervalo de iterações é extremamente lento. Observe pela Figura 29 que para $w = 0{,}5$ o erro foi reduzido de 1 para $0{,}015$ nas primeiras 23 iterações. Agora, é necessário no mínimo mais 47 iterações para reduzir o erro a 10^{-6}, pois o número mínimo de iterações de acordo com a Tabela 4 é 70 para $w = 0{,}49$. Deste modo, se o erro máximo permitido fosse de 10^{-6}, todo o gráfico acima seria deslocado para a esquerda, e não seria possível observar com clareza o efeito de ω sobre o número de iterações.

5.1.2 O FATOR ω DO MÉTODO SUPER-RELAXAÇÃO SUCESSIVA (SOR)

O mesmo feito anteriormente no método de Jacobi ponderado para a obtenção do fator ω que minimiza o número de iterações foi também feito para o método SOR (Tabela 5). Onde ω varia no intervalo entre $1 \leq \omega \leq 2$.

Tabela 3 - Fator ω que minimiza o número de iterações k no método SOR.

Malha	$\boldsymbol{\omega}$	$\boldsymbol{k}$	**Malha**	$\boldsymbol{\omega}$	$\boldsymbol{k}$
1	$1{,}04$	6	*6*	$1{,}05 - 1{,}09$	43
2	$1{,}06 - 1{,}19$	11	*7*	$1{,}01 - 1{,}04$	60
3	$1{,}03 - 1{,}13$	16	*8*	$1{,}00 - 1{,}05$	88
4	$1.00 - 1.09$	22	*9*	$1{,}24 - 1{,}28$	120
5	$1{,}03 - 1{,}10$	31	*10*	$1{,}30 - 1{,}35$	149
			11	$1{,}35 - 1{,}37$	179

Quando a matriz é pequena, não há grandes diferenças do resultado obtido quando comparado com o método de Jacobi ponderado. No entanto, quando a matriz é de larga escala, há diferença significativas no número de iterações. Além do fato de que, no método SOR, a convergência é sempre observada, independentemente do valor de ω.

Quando a matriz é pequena observamos que $\omega \approx 1$, o que equivale ao método de Gauss-Seidel, assim não é necessário utilizar o método SOR quando a matriz é pequena. É possível observar que quanto maior o número de nós, e consequentemente a matriz, maior são os valores de ω que otimizam o processo iterativo. Além disso, a faixa de valores de ω que minimizam o número de iterações é maior do que a observada no caso de Jacobi ponderado.

O comportamento do número de iterações para diferentes valores de ω para o problema discretizado pela malha 4, pode ser observado na Figura 30, onde foi considerado um erro máximo permitido de $0{,}015$.

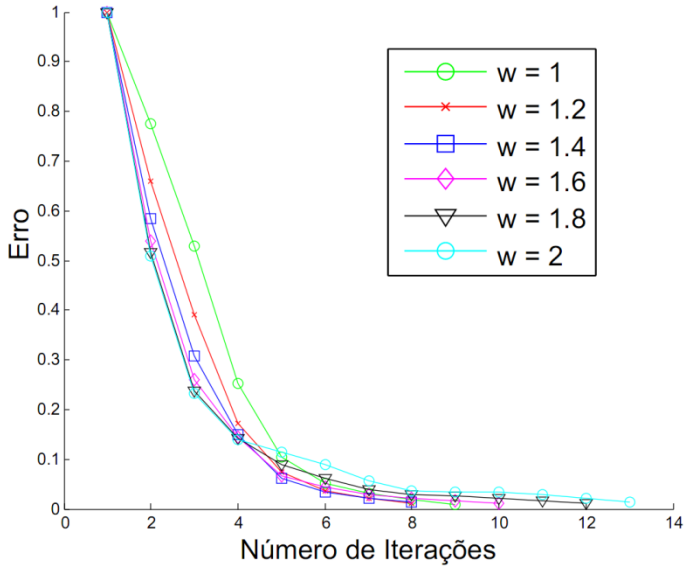

Figura 30 - Número de iterações para a malha 4 para diferentes fatores w no método SOR.

5.1.3 COMPARAÇÃO ENTRE OS MÉTODOS ESTACIONARIOS

Nos algoritmos dos métodos estacionários não há grande diferenças de um método para o outro, a única mudança é o pré-condicionador. Deste modo, apenas uma única rotina de programação (Anexo I) foi escrita para os quatro métodos: Jacobi, Jacobi ponderado, Gauss-Seidel e SOR.

Essa rotina foi utilizada para avaliar a performance dos diferentes métodos em resolver o problema $\boldsymbol{A}x = b$ para as diferentes malhas. Foi contabilizado o tempo de execução, a quantidade de memória utilizada e o número de iterações até a convergência. Nos métodos de Jacobi ponderado e SOR, foram utilizados os valores de ω obtidos nas tabelas 4 e 5, respectivamente.

A avaliação do tempo e da memória requerida pelo algoritmo foi feita utilizando a ferramenta *profile*, essa é uma ferramenta interna do software MATLAB®, a qual permite avaliar o tempo de execução e a memória utilizada por um código (.m) qualquer. Por padrão, a função para rastreamento do uso da memória se encontra desabilitada. Para habilitar a mesma, é necessário inserir na tela de comando: `profile -memory`. A interface da ferramenta *profile* pode ser aberta clicando em seu ícone no menu superior, ou escrevendo na tela de comando: `profile viewer`.

Foi utilizada a versão R2015a do software MATLAB® em um computador com especificações:

- Sistema Operacional: Windows 8.1
- Tipo de Sistema: 64-bit
- Processador: Intel® Core™ i7-4510U CPU @ 2.00GHz 2.60GHz
- Memória Instalada: 16.0 GB (15.9 Utilizável)

O resultado para os diferentes métodos estáticos e malhas, pode ser observado nas tabelas a seguir.

Tabela 4 - Tempo de execução e uso da memória para as Malhas 1 e 2.

	Malha	k	t (s)	Memória (kB)
Jacobi	1	21	0,017	52
Jacobi ponderado		3	0,015	64
Gauss-Seidel		8	0,014	64
SOR		6	0,014	52
Jacobi	2	68	0,014	104
Jacobi ponderado		25	0,015	80
Gauss-Seidel		12	0,015	72
SOR		11	0,014	60

A partir da malha 2, como já dito anteriormente, não é observado convergência de x para o método de Jacobi, pois para qualquer malha, com exceção das duas primeiras, o método rapidamente diverge. Assim, as tabelas a seguir não apresentam mais esse método.

Tabela 5 - Tempo de execução e uso da memória para as Malhas 3 a 8.

	Malha	k	t (s)	Memória (kB)
Jacobi Ponderado	3	41	0,015	92
Gauss-Seidel		17	0,014	88
SOR		16	0,016	64
Jacobi Ponderado	4	70	0,016	124
Gauss-Seidel		22	0,015	116
SOR		22	0,015	136
Jacobi Ponderado	5	129	0,017	136
Gauss-Seidel		32	0,017	124
SOR		31	0,020	156
Jacobi Ponderado	6	249	0,021	204
Gauss-Seidel		46	0,016	144
SOR		43	0,020	136
Jacobi Ponderado	7	489	0,026	520
Gauss-Seidel		62	0,020	120
SOR		60	0,016	140
Jacobi Ponderado	8	867	0,154	8224
Gauss-Seidel		88	0,028	8220
SOR		88	0,027	8220

A partir da malha 8 não é observado convergência de x para o método de Jacobi ponderado. Como anteriormente, para o método de Jacobi, o método rapidamente diverge. Deste modo, a Tabela a seguir não apresenta esse método.

Tabela 6 - Tempo de execução e uso da memória para as Malhas 9 a 11.

	Malha	k	t (s)	Memória (kB)
Gauss-Seidel	**9**	133	0,204	35440
SOR		120	0,187	32856
Gauss-Seidel	**10**	200	1,058	135912
SOR		149	0,896	131284
Gauss-Seidel	**11**	301	6,709	525304
SOR		179	4,685	525268

Para as malhas de 1 a 7 foi observada grande flutuação nos valores do tempo de execução e de uso da memória, provavelmente devido à influência de tarefas secundárias executadas pelo software. Por outro lado, para os problemas maiores, das malhas 8 a 11, foi observada somente uma pequena variação desses parâmetros. Na verdade, em valores absolutos, a variação foi praticamente à mesma, mas como as malhas maiores requerem maior tempo de execução e memória, essa variação de forma relativa é muito menor.

No software MATLAB® não existe uma ferramenta que meça com precisão a memória utilizada para a execução de um programa. Assim, os valores obtidos nas tabelas 6 a 8 e também os das próximas tabelas, não são precisos, mas servem como bons indicadores.

Entre as conclusões que podemos retirar dos dados das Tabelas 6 a 8 está o fato de que para matrizes de larga escala, não é possível utilizar os métodos de Jacobi e Jacobi ponderado, pois os mesmos não apresentam convergência. Podemos observar também, que entre os métodos estacionários, o que apresenta melhor desempenho para matriz grandes é o método SOR. No entanto, há a inconveniência de ser necessário conhecer o valor ω previamente. Deste modo, outra opção é o método de Gauss-Seidel, o qual também apresenta convergência, mas que requer um maior número de iterações e, portanto, um maior tempo de execução. De qualquer forma, quando a matriz a ser resolvida é ainda maior, ambos os métodos se tornam caros devido ao grande número de iterações até a obtenção da convergência. Daí, a

necessidade de métodos diferentes, tais como o método Multigrid, o método do gradiente conjugado e o método dos mínimos resíduos.

5.2 CICLO V E W DO MÉTODO MULTIGRID

O número de nós para cada malha da Tabela 3 foi escolhida com o intuito de facilitar a obtenção da matriz de interpolação $\boldsymbol{T}$ e da matriz de restrição $\boldsymbol{R}$ do método Multigrid. Nesse método, quando são utilizados espaçamentos entre os nós múltiplos de dois, tais como $h, 2h, 4h, \ldots, 2^n h$, o número de nós é reduzido pela metade a cada iteração. Como consequência, qualquer malha com um número ímpar de nós precisa primeiro ser reduzida a uma malha com um número par de nós. Outro problema é que, mesmo para uma malha com um número par de nós, essa depois de reduzida, pode apresentar um número ímpar de nós, como por exemplo, uma malha com 10 nós após reduzida apresenta 5 nós. Além disso, mesmo para malhas com um número par de nós, a geometria da malha resultante da aplicação da matriz de restrição $\boldsymbol{R}$ pode representar a geometria original de forma deficiente (Figura 31).

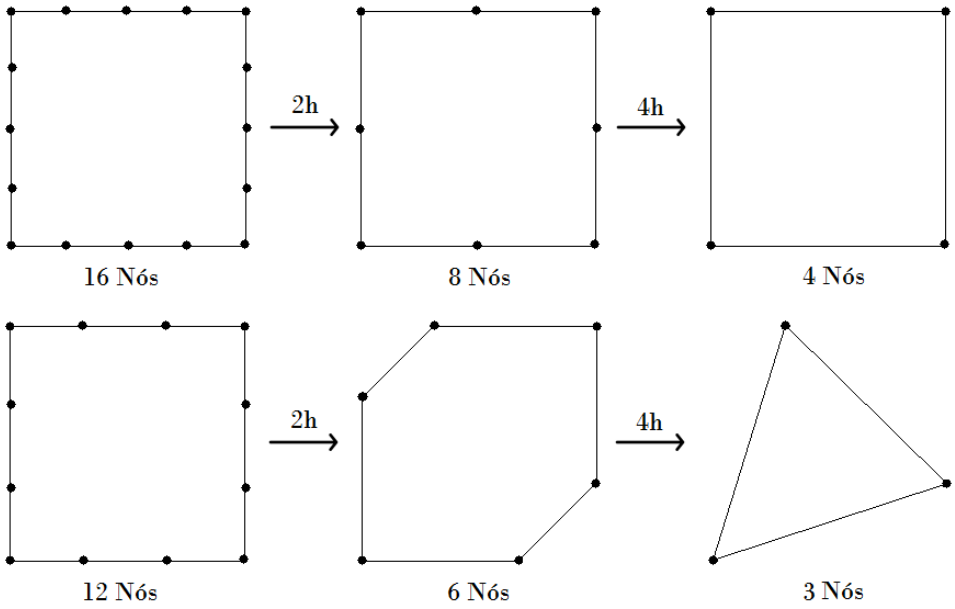

Figura 31 - Efeito da aplicação de uma mesma matriz $\boldsymbol{R}$ a uma mesma geometria discretizada por quantidades diferentes de elementos.

Assim, são necessárias diferentes matrizes $\boldsymbol{T}$ e $\boldsymbol{R}$, dependendo da quantidade de nós e da geometria. No entanto, quando a geometria é quadrada e são escolhidas malhas com 2^n nós, como as da Tabela 3, nenhum desses problemas é observado, pois cada redução é um múltiplo de 2. Deste modo, é possível utilizar somente uma matriz de interpolação $\boldsymbol{T}$ e uma matriz de restrição $\boldsymbol{R}$.

A rotina de programação para a aplicação do método Multigrid (Anexo II) utiliza uma matriz $\boldsymbol{T}$ que representa uma interpolação linear entre os nós e uma matriz $\boldsymbol{R}$ do tipo completamente ponderada. Foram utilizadas cinco iterações para resolver cada matriz $\boldsymbol{A}_{2^k h}$, onde $k = 1,2,\ldots n$, feitas com o método de Gauss-Seidel. O programa computa o ciclo V e W do método do Multigrid e retorna o valor de x e o erro (eq. 5.1) depois do processo. O resultado para o tempo de execução e uso da memória, assim como o erro, pode ser observado na Tabela 9.

Para a obtenção de um erro menor, a primeira opção é, naturalmente, aumentar o número de iterações feito pelo método estacionário logo após cada restrição e interpolação da malha. No entanto, isso deve ser feito com cautela, pois em certos casos pode ser observador um comportamento divergente do erro com o aumento das iterações (Figura 32).

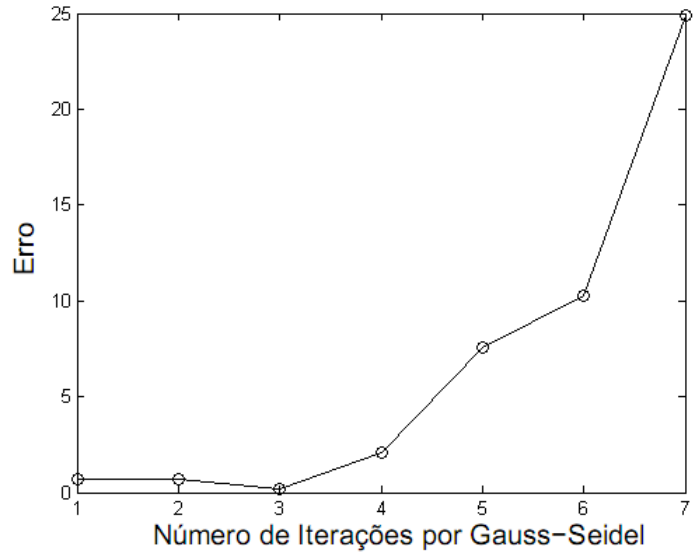

Figura 32 - Comportamento divergente do erro no ciclo V do método Mutigrid para a malha 10.

Tabela 7 - Tempo de execução e uso da memória para o método Multigrid

	Malha	Erro	t (s)	Memória (kB)
Ciclo V	**2**	$9{,}40e-06$	0,019	60
Ciclo W		*	*	*
Ciclo V	**3**	$9{,}88e-05$	0,021	52
Ciclo W		$9{,}88e-05$	0,021	64
Ciclo V	**4**	$4{,}06e-04$	0,012	64
Ciclo W		$4{,}06e-04$	0,014	88
Ciclo V	**5**	$1{,}30e-03$	0,014	80
Ciclo W		$1{,}30e-03$	0,016	92
Ciclo V	**6**	$2{,}60e-03$	0,015	96
Ciclo W		$2{,}90e-03$	0,028	96
Ciclo V	**7**	$6{,}90e-03$	0,012	104
Ciclo W		$4{,}30e-03$	0,063	116
Ciclo V	**8**	$4{,}19e-02$	0,049	5708
Ciclo W		$4{,}90e-03$	0,140	5724
Ciclo V	**9**	0,96	0,170	37544
Ciclo W		$5{,}70e-03$	0,353	42636
Ciclo V	**10**	7,55	0,683	149408
Ciclo W		$7{,}10e-03$	0,514	207436
Ciclo V	**11**	13,33	3,398	621096
Ciclo W		$8{,}70e-03$	5,712	907213

A malha 1 não é apresentada na Tabela 9 pois ela é a menor malha possível, não podendo ser reduzida. Já a malha dois não apresenta o ciclo W, pois essa pode ser reduzida somente uma única vez e, consequentemente, o ciclo W não pode ser implementado. Podemos observar da Tabela 9 que quando a matriz é pequena, não há grande diferença em utilizar um ciclo ou o outro. No entanto, para matrizes maiores o ciclo V diverge e a única opção é o ciclo W, que embora apresente um erro pequeno, requer uma quantidade bem maior de memória e tempo para ser executado. Deste modo, os ciclos V e W do método Multigrid não são a melhor opção para resolver o problema apresentado na Figura 27, quando a matriz é muito grande.

Embora os métodos iterativos de Gauss-Seidel e SOR, requeiram uma grande quantidade de iterações e consequentemente um maior tempo de execução, a memória utilizada por esses métodos para resolver os problemas das malhas 9 a 11 foi bem menor que a observada no ciclo W do método Multigrid. Assim, é preciso encontrar outra opção que apresente convergência para um pequeno número de iterações, reduzindo o tempo de execução, e que utilize menos memória que o processo Multigrid. Cabe aos métodos baseados nos espaços de Krylov essa tarefa.

5.3 COMPARAÇÃO ENTRE OS MÉTODOS CG E GMRES

O método do Gradiente Conjugado foi criado para matrizes quadradas simétricas. Para utilizar o mesmo em problemas que resultam em matrizes não simétricas, como a que surge da aplicação do método de elementos de contorno ao problema da Figura 27, é necessário transformar o problema $\boldsymbol{A}x = b$ para uma matriz $\boldsymbol{A}$ qualquer, em um problema para uma matriz $\boldsymbol{A}'$ simétrica. Isso é feito multiplicando a equação por $\boldsymbol{A}^T$ pela esquerda. Observe que:

$$\boldsymbol{A}'^T = (\boldsymbol{A}^T\boldsymbol{A})^T = \boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{A}'$$

ou seja, $\boldsymbol{A}' = \boldsymbol{A}^T\boldsymbol{A}$ é uma matriz simétrica. O problema a ser resolvido e então dado por:

$$\boldsymbol{A}^T\boldsymbol{A}x = \boldsymbol{A}^T b$$

$$\boldsymbol{A}'x = b'$$

No entanto, essa mudança é extremamente cara quando a matriz $\boldsymbol{A}$ é uma matriz cheia e de larga escala, pois a multiplicação da matriz com ela mesma, como já dito anteriormente, apresenta complexidade $O(N^3)$ o que pode tornar essa

operação inviável, consumindo mais tempo e memória do que a utilizada na execução do próprio método CG.

Por outro lado, para o método dos mínimos resíduos generalizados, não é necessário computar o produto $\boldsymbol{A}^T\boldsymbol{A}$, pois o método foi desenvolvido para qualquer tipo de matriz. No entanto, o método GMRES envolve o cálculo direto das bases dos espaços de Krylov, assim como a decomposição QR da matriz $\boldsymbol{H}$ e também a resolução de um problema de mínimos quadrados. Deste modo, seu algoritmo apresenta naturalmente uma maior complexidade que o algoritmo para o método CG.

O programa para a execução do método CG (Anexo III) apresenta o produto $\boldsymbol{A}^T\boldsymbol{A}$, pois a matriz $\boldsymbol{A}$**,** resultante da aplicação do método de contorno ao problema da Figura 27, é uma matriz assimétrica. O programa para o método GMRES é encontrado no anexo IV. Igualmente ao que foi feito para os métodos estacionários, para o método CG e GMRES foram computados o número de iterações necessárias para atingir um erro máximo permitido, definido por (5.1), de 10^{-6} com o número de iterações k limitado a um máximo de $k = 1000$. O resultado pode ser observado na Tabela 10.

Tabela 8 - Tempo de execução e uso da memória para os métodos CG e GMRES.

	Malha	k	t (s)	Memória (kB)
CG	**1**	1	0,20	*
GMRES		1	0,12	*
CG	**2**	5	0,20	*
GMRES		5	0,13	*
CG	**3**	8	0,21	*
GMRES		9	0,15	*
CG	**4**	16	0,21	*
GMRES		13	0,14	*
CG	**5**	33	0,21	28
GMRES		16	0,14	26
CG	**6**	52	0,24	60
GMRES		22	0,19	32
CG	**7**	69	0,023	516
GMRES		28	0,029	28
CG	**8**	57	0,029	2056
GMRES		34	0,038	52
CG	**9**	40	0,083	8212
GMRES		39	0,118	324
CG	**10**	38	0,388	32776
GMRES		44	0,276	2564
CG	**11**	29	1,968	131272
GMRES		47	0,871	12032

*Não foi obtida leitura de uso da memória.

Na Tabela anterior, a ferramenta *profile* do software MATLAB® não acusou uso de memória para as malhas de 1 a 4, mas como dito anteriormente, os valores obtidos na Tabela 10 não são precisos, mas servem como bons indicadores.

Da Tabela 10, é nítida a influência do produto $\boldsymbol{A^T A}$ para o método CG quando a matriz $\boldsymbol{A}$ tem grandes dimensões, pois para as malhas 8 a 11 há uma grande diferença de tempo de execução e uso da memória entre os dois métodos. Deste modo, métodos que dão origem a matrizes de larga escala não simétricas, como o MEC, devem ter suas matrizes resolvidas pelo método GMRES, pois o produto $\boldsymbol{A^T A}$ inviabiliza a utilização do método CG.

Mesmo computando o produto $\boldsymbol{A^T A}$, o método CG apresentou melhor desempenho tanto em tempo de execução, como em uso da memória, do que os métodos estacionários e os ciclos V e W do método Multigrid. Assim, fica claro, que dentre as opções descritas nesse trabalho, a que apresenta melhor desempenho na resolução de matrizes não simétricas é o método GMRES.

5.3.1.1 GMRES COM PRÉ-CONDICIONADORES E GMRES$(\boldsymbol{m})$

O método GMRES, embora apresente um bom desempenho, ainda apresenta algumas limitações. Como já dito anteriormente, as matrizes $\boldsymbol{V}$ e $\boldsymbol{H}$ aumentam a cada iteração, requerendo a cada iteração, uma quantidade maior de memória. Para corrigir em parte esse problema é utilizado o GMRES(m) (Anexo V), que reinicia o processo, ou seja as matrizes $\boldsymbol{Q}$ e $\boldsymbol{H}$, a cada m iterações.

Além da redução do uso de memória, é possível reduzir também o número de iterações até a convergência, para tal utilizando um pré-condicionador. O uso dessas duas ferramentas pode tornar o método GMRES mais eficiente.

O efeito do método GMRES(m), assim como o efeito do uso de um pré-condicionador para o método GMRES, tanto pela esquerda (Anexo VI), quanto pela direita (Anexo VII), podem ser vistos na Tabela 11. Nesse caso, foi utilizado como parâmetro de recomeço $m = 20$ e o pré-condicionador selecionado é o mesmo utilizado no método de Gauss-Seidel, a parte triangular inferior da matriz $\boldsymbol{A}$.

Tabela 9 - Efeito do método GMRES(m) e do uso de um pré-condicionador

	Malha	k	t (s)	Memória (kB)
GMRES$(\boldsymbol{m})$	9	64	0,148	100
GMRES Pré-esq.		8	0,249	24636
GMRES Pré-dir.		7	0,301	24668
GMRES$(\boldsymbol{m})$	**10**	90	0,465	128
GMRES Pré-esq.		8	1,589	98508
GMRES Pré-dir.		7	1,874	98568
GMRES$(\boldsymbol{m})$	**11**	109	1,807	5156
GMRES Pré-esq.		8	13,558	393940
GMRES Pré-dir.		7	13,633	406304

Quando comparado com o método GMRES (Tabela 10), podemos observar da Tabela 11, que o uso do método GMRES(m) diminui o requerimento do uso de memória. No entanto, devido ao aumento do número de iterações, há um aumento no tempo de execução. Já a utilização de um pré-condicionador no método GMRES diminui consideravelmente o número de iterações até para convergência, mas eleva consideravelmente a memória utilizada. Isso ocorre, pois o espaço de Krylov é modificado e a obtenção dos novos vetores da base envolve o produto da matriz inversa do pré-condicionador $\boldsymbol{P^{-1}}$ com a matriz $\boldsymbol{A}$. Podemos notar também que, embora o pré-condicionador pela direita tenha convergido mais rapidamente, o pela esquerda consumiu menos memória e tempo.

Dos métodos demonstrados nesse trabalho, o método GMRES aparece como o mais indicado para a resolução de problemas advindos da utilização do método dos elementos de contorno. A utilização do método GMRES(m) é ideal para a resolução de matrizes de larga escala quando a capacidade computacional é reduzida, ou o problema extremamente grande. A utilização de um pré-condicionador dever ser feita com cautela, uma opção para não computar a inversa $\boldsymbol{P^{-1}}$ é resolver o produto entre $\boldsymbol{P^{-1}}$ e $\boldsymbol{A}$ iterativamente.

6 IMPLEMENTAÇÃO

6.1 PROBLEMA MODELO

Durante todo esse capítulo um único problema de condução de calor será analisado. Isso é feito com o intuito de facilitar a compreensão e análise do método ACA. O problema é o mesmo dado no capítulo 5, dado por uma placa quadrada de lado $1m$, e condutividade térmica $k = 1\frac{w}{mK}$ isolada nas laterais superior e inferior, com temperatura $0K$ e $1K$ nas laterais, esquerda e direita, respectivamente (Figura 33).

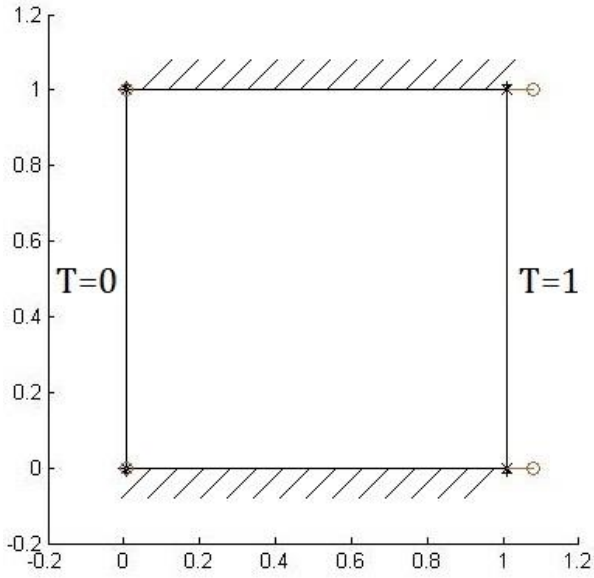

Figura 33 - Problema de condução de calor analisado.

Além das razões já discutidas anteriormente, o problema foi escolhido por decorrência da sua geometria simples, que neste caso, qual facilita a observação dos parâmetros e resultados intermediários obtidos durante a aplicação do método ACA.

Para a solucionar o problema pelo método BEM, o mesmo será discretizado por uma malha com 44 nós, ou seja 11 nós por aresta (Figura 34), essa quantidade foi escolhida pois é pequena o suficiente para possibilitar a observação dos resultados com facilidade e grande o bastante para resultar em um árvore binária e uma quantidade de clusters suficiente para uma melhor análise.

O problema será resolvido utilizando elementos de contorno constantes [17], nesse caso a geometria do problema é aproximada por segmentos de retas com um único nó presente no meio de cada elemento, esse tipo é caracterizado por apresentar temperatura e fluxo constantes ao longo do elemento. Ele foi escolhido pois permite o cálculo das componentes das matrizes $\boldsymbol{G}$ e $\boldsymbol{H}$ de forma analítica, eliminando o uso de

integrais numéricas, o que pode exigir uma grande quantidade de tempo. Isso permite avaliar o método ACA com maior precisão, uma vez que a maior parte do tempo observado na execução da rotina corresponde agora a aplicação do método.

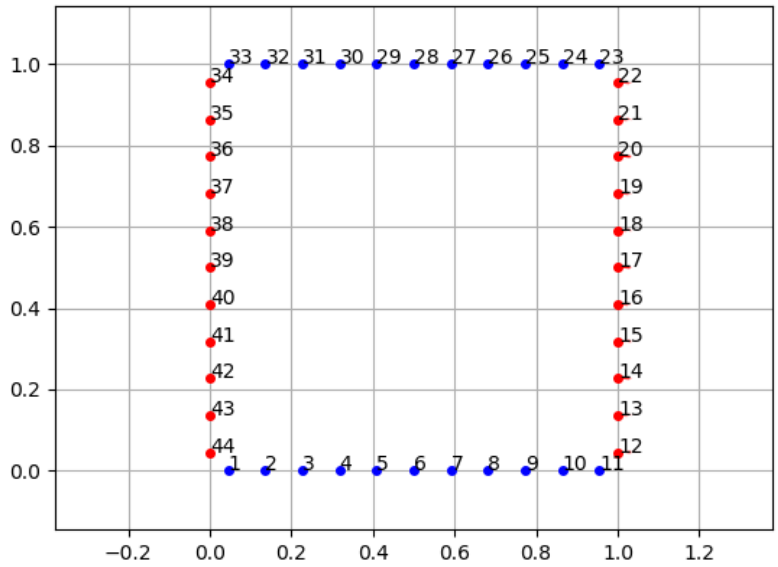

Figura 34 - Nós da malha, onde nos nós vermelhos a temperatura e conhecida e nos nós azuis o fluxo de calor e conhecido.

6.2 LINGUAGEM DE PROGRAMAÇÃO JULIA

Nesse capítulo todos as rotinas são apresentadas na linguagem de programação Julia, escolha feita pois além de sua boa performance, sua grafia se assemelha bastante a observada no Matlab®, linguagem na qual a grande maioria dos trabalhos anteriores foram feitos. Todos os códigos utilizados neste capitulo são um resultado de um esforço coletivos do grupo de mecânica dos sólidos e do Grupo de Dinâmica de Sistemas (GDS) da UnB voltados para o métodos BEM e ACA.

Julia [42] é uma linguagem de programação de alto nível, esse tipo de linguagem apresenta uma forte abstração da linguagem computacional, aproximando essa da linguagem natural, ou seja, humana. Julia é uma linguagem dinâmica de grande performance para simulação numérica, ela providencia um compilador sofisticado, com execução paralela distribuída, o que reduz o tempo necessário de compilação fazendo uso simultâneo de vários recursos computacionais. Além disso, ela apresenta grande acurácia numérica e uma biblioteca matemática extensa, assim como uma crescente número de pacotes externos criados diariamente pela comunidade. Julia também possui uma poderosa interface gráfica baseada em navegadores de internet, o IJulia, que combina capacidade de compilação, criação de textos e grande capacidade gráfica.

Julia apresenta uma boa performance, capaz de se aproximar e em alguns casos se equiparar a linguagem C. Além da capacidade de processar códigos em Python, C e Fortran através da utilização de pacotes específicos. Uma de suas grandes vantagens é ser uma linguagem de programação licenciado pelo MIT (Massachusetts Institute of Technology), o qual trabalha com uma política de linguagem gratuita e código aberto.

A tabela a seguir apresenta o tempo relativo, em relação ao tempo obtido quando executado em C, levados por certas rotinas em diferentes linguagens de programação para serem executadas. É importante saber que essas rotinas não foram implementadas para atingir a performance máxima, elas foram criadas com a intenção de testar a capacidade de compilação de cada linguagem, testando algoritmos específicos em cada uma delas, tais como a análise de matrizes, ordenações, loops numéricos, gerações de números aleatórios e operações matriciais.

Tabela 10 - Tempo de execução relativo, observados em diferentes linguagens de programação [42].

	C gcc 4.8.5	Julia 0.6.0	Fortran gcc 4.8.5	Java 1.8.0_14	Matlab R '2017a'	Python 3.4.6	Mathe-matica 11.1.1
iteration_mandelbrot	1.00	0.63	0.90	0.36	5.01	11.32	5.48
iteration_pi_sum	1.00	0.85	1.00	0.91	0.85	17.11	1.31
recursion_fibonacci	1.00	1.67	0.52	1.53	18.48	89.82	126.56
recursion_quicksort	1.00	0.93	1.30	2.54	3.05	36.74	44.57
parse_integers	1.00	1.36	5.43	3.77	244.33	19.07	14.79
print_to_file	1.00	0.67	3.49	6.08	117.46	1.41	62.95
matrix_statistics	1.00	1.88	1.99	5.97	18.71	16.35	7.86
matrix_multiply	1.00	0.95	1.24	8.66	1.14	1.14	1.16

6.3 MATRIZES HIERÁRQUICAS

O primeiro passo no processo de implementação do método ACA é construir as matrizes hierárquicas, também chamadas matrizes-*H*. Para tal, são necessário três processos distintos: particionar os dados em diferentes blocos, organizar esses blocos de forma hierárquica e por fim, avaliar e agrupar esses blocos de acordo com a condição de admissibilidade.

6.3.1 PARTICIONAMENTO DOS DADOS

No problema utilizado como modelo, os dados a serem particionados correspondem aos nós da malha, esse processo é feito pela rotina a seguir.

Todas as rotinas apresentadas durante esse capitulo são também fornecidas em anexo com maiores comentários.

Rotina 1 (R1) - Função $divnode(\boldsymbol{X}, t)$, responsável por realizar a bipartição dos dados.

```
1  function divnode(X,t)
2      n = length(t)
3      x = X[t,:]
4      c = mean(x,1)
5      covx = cov(x)
6      eig_valx,eig_vecx  = eig(covx)
7      ref = eig_vecx[:,indmax(eig_valx)]
8      attcond = zeros(n)
9          for i=1:n
10             attcond[i] = (x.-c)[i,:]'*ref
11         end
12     x1=t[attcond.>=0]
13     x2=t[attcond.<0]
14     diam = 2*maximum(sqrt.(((x.-c).*(x.-c))[:,1]+((x.-c).*(x.-c))[:,2]))
15     return x1,x2,diam,c
16 end
```

Essa rotina de programação utiliza como dados de entrada a matrizes $\boldsymbol{X}$ com o par coordenado (x, y) de cada nó da malha e um vetor t com os números dos nós a serem divididos. Ela implementa a análise PCA, obtendo os autovalores e autovetores da matriz de covariância (R1, ll.5-7) e repartindo os dados a partir da condição $(x - \bar{x})v(\xi_1) > 0$ (R1, l.9). Os nós que atendem a condição são salvos em $x1$ e os que não atendem em $x2$, formando dois blocos disjuntos de nós (R1, ll.12-13). Ao final é ainda calculado o diâmetro do bloco, antes desse ser dividido, o qual será usado posteriormente para avaliar a condição de admissibilidade dos blocos. Por fim, a rotina retorna os dois novos blocos $x1$ e $x2$, o diâmetro $diam$ e a posição do centro c do bloco, esses dois últimos antes desse ser repartido.

A primeira divisão dos nós do problema modelo (Figura 35) resulta em $x1 = 1{:}\,6, 28{:}\,44$ e $x2 = 7{:}\,27$, ou seja, 23 nós em um bloco e 21 no outro, além de $diam\,(1) = 1.3514$ e $c(1) = (0.5\,, 0.5)$.

A simplicidade do problema torna possível avaliar os resultados obtidos. Observe que qualquer quadrado com canto inferior esquerdo no ponto $(0{,}0)$ tem centro em $\left(\frac{L}{2}, \frac{L}{2}\right)$, onde L é o comprimento de sua aresta, ou seja, um quadrado de lado 1m terá seu centro em $(0.5{,}0.5\,)$, assim como observado em $c(1)$. Para um quadrado, também temos que a maior distância entre dois pontos pertencentes a sua aresta é a diagonal entre dois de seus vértices, que para um quadrado de lado 1m é igual a $\sqrt{2} = 1.4142$. No entanto, o quadrado do problema não apresenta nós em seus

vértices, assim a maior distância entre dois nós deve ser um pouco menor, neste caso $diam\,(1) = 1.3514$. Por último, para uma malha quadrada alinhada com os eixos coordenados é esperado que a direção de maior variabilidade dos dados seja na própria direção dos eixos coordenados. Observe que a divisão que gerou os blocos $x1$ e $x2$ é simplesmente uma linha reta vertical que passa pelo centro da malha. Qualquer nó da malha que passe por essa linha apresenta condição $(x - \bar{x})v(\xi_1) = 0$, isso acontece em dois nós, 6 e 28. Assim, é esperado que $x1$ o qual contém todos os nós tais que a condição e " $>= 0$" tenha dois nós a mais, o que de fato ocorre (Figura 35).

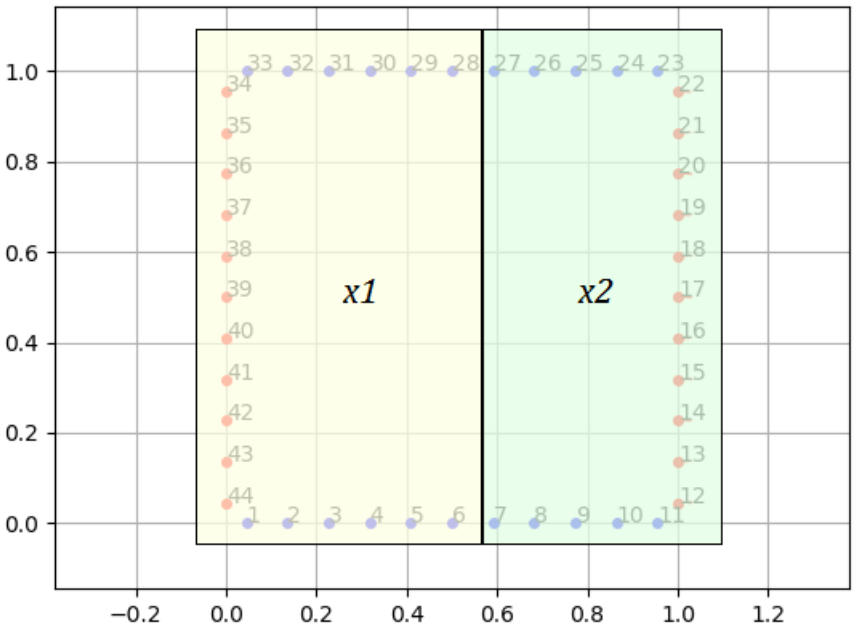

Figura 35 - Resultado da 1ª bipartição dos nós da malha.

6.3.2 ORGANIZAÇÃO HIERÁRQUICA DOS BLOCOS

O processo anterior de particionamento dos nós em diferentes blocos é executado em conjunto com outro algoritmo responsável por organizar esses blocos. Essa organização possibilita montar uma árvore binária, a qual guarda a relação entre esses blocos. Esse processo é executado pela Rotina 2. Observe que essa rotina inclui a função $divnode(\boldsymbol{X}, t)$, pois como dito ambos os processos ocorrem em conjunto.

Rotina 2 (R2) - Parte da função $cluster(\boldsymbol{X}, max\_elem, \eta)$ responsável pela classificação dos blocos.

```
1  function cluster(X, max_elem,η)
2      m,n = size(X)
3      max_clt = ceil(Int,2*m/max_elem)
4      child1 = zeros(1,2*max_clt)
5      child2 = zeros(1,2*max_clt)
6      t = collect(1:m)
7      inode = 1
8      ileaf = 1
9      nodes = Array{Any}(2*max_clt)
10     leaves = Array{Any}(2*max_clt)
11     child = zeros(Int,2*max_clt,2)
12     nodes[1] = t
13     center_row = zeros(2*max_clt,2)
14     diam = zeros(2*max_clt)
15     i = 1
16     while inode >= ileaf
17         t1,t2,d,c = divnode(X,nodes[i])
18         center_row[i,:] = c;
19         diam[i] = d;
20             if length(t1)> max_elem
21                 inode = inode + 1
22                 nodes[inode] = t1
23                 child[i,1] = inode
24             else
25                 leaves[ileaf] = t1
26                 ileaf = ileaf + 1
27                 child1[i] = ileaf
28             end
29             if length(t2) > max_elem
30                 inode = inode + 1
31                 nodes[inode] = t2
32                 child[i,2] = inode
33             else
34                 leaves[ileaf] = t2
35                 ileaf = ileaf + 1
36                 child2[i] = ileaf
37             end
38         i = i + 1
39     end
40 …
```

A Rotina 2 contém parte da função $cluster$. Essa rotina de programação utiliza como dados de entrada a matriz $\boldsymbol{X}$ com o par coordenado (x, y) de cada nó da malha, a variável max_elem que define a quantidade mínima de nós em um bloco e η, coeficiente importante para avaliação da condição de admissibilidade dos blocos.

As linhas 4 a 14 da Rotina 2 têm a função de criar variáveis e alocar matrizes para armazenar certos valores durante a execução da rotina. Na linha 15, $i = 1$ inicia o processo iterativo para o primeiro bloco, ou seja, toda a malha. O bloco é dividido pela função $divnode(\boldsymbol{X}, t)$, os valores de c e $diam$ são armazenados e os blocos $t1$ e $t2$ resultantes da bipartição são avaliados. Se a quantidade de nós no bloco $t1$ for maior do que a quantidade máxima permitida (R2, l.20), o bloco se torna um nó da

árvore (R2, l.22) e será bipartido novamente. Caso contrário (R2, l.24), ele se torna uma folha (R2, l.25) e não será mais bipartido. A linhas 29 a 37 repetem essa avalição para o bloco $t2$. No final, o algoritmo segue para o próximo bloco (R2, l.38) até que o número de folhas seja maior que o de nós (R2, l.16), o que só ocorre quando todos os nós possuem filhos, assim a árvore estará completa.

Os tamanho das matrizes criadas ao início da rotina (em relação ao números de linhas) é dado em função de $max\_clt = ceil(Int, 2 * m/max\_elem)$ (R2, l.3). Esse é um limite superior para o tamanho dessas matrizes. Observe que a condição para a repartição dos blocos (R2, l.20) permite que um bloco com $max_{elem} + 1$, ainda seja dividido, ou seja uma folha terá no mínimo $\frac{max_{elem}+1}{2}$ elementos. Considerando o caso extremo onde cada folha tem o mínimo de elementos possível, podemos dizer que a árvore tem $\frac{2m}{max_{elem}+1}$ folhas, ou ainda de forma conservadora, $\frac{2m}{max_{elem}}$, valor esse que define max_clt. Quando multiplicamos esse valor por 2 temos um limite superior para o número total de nós na árvore (uma árvore binária cheia de n folhas tem sempre $2n - 1$ blocos) e nenhuma matriz que guarde estes blocos (nós e folhas) terá tamanho maior que o dado por esse limite.

Para o problema modelo e $max_{elem} = 10$, podemos extrair os valores:

Tabela 11 - Resultado extraidos da Rotina 2

inode = 6 ileaf = 8	nodes = [[1:44], [1:6,28:44], [7:27], [28:39], [1:6,40:44], [17:27]]	leaves = [[7:16], [28:33], [34:39], [1,40:44], [2:6], [22:27], [17:21]]	child = [2 3; 4 5; 6 0]

A variável inode indica a quantidade de nós na árvore, neste caso a árvore terá 6 nós. Analogamente, a variável $ileaf$ indica a quantidade de folhas na árvore. No entanto, devido ao modo como ela foi definida, a quantidade de folhas é dada por $ileaf - 1$. Neste caso, a árvore terá 7 folhas. Esses valores podem ser verificados observando os vetores $nodes$ e $leaves$ os quais possuem 6 e 7 elementos respectivamente. Esses vetores foram definidos como Array{Any}, ou seja um vetor de qualquer tipo de elemento, neste caso um vetor de vetores. No caso do vetor $nodes$, temos que $nodes[i]$ retorna um vetor com os nós da malha pertencente ao nó i da

árvore. O mesmo é válido para o vetor $leaves$. Este pode ainda ser utilizado para visualizar o resultado final do processo de bipartição dos nós da malha (Figura 36).

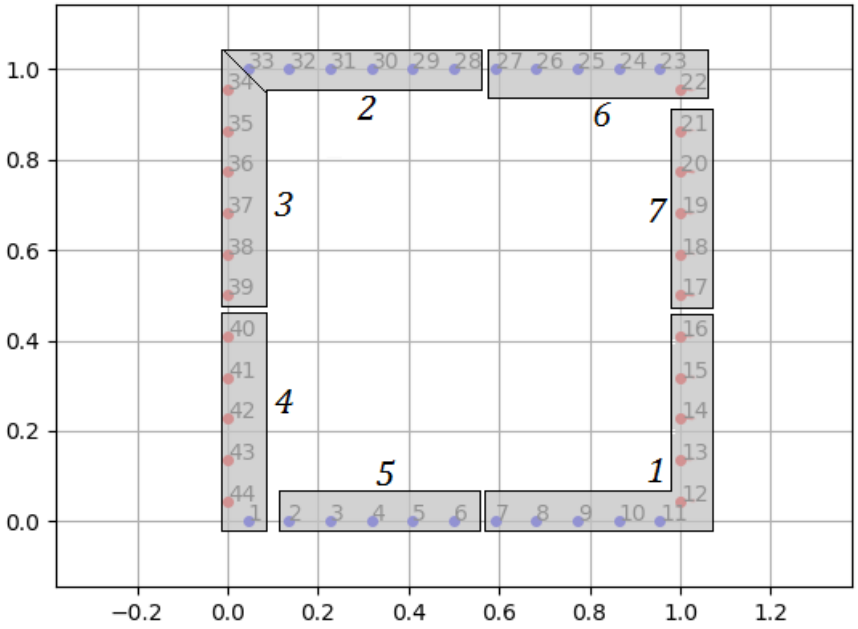

Figura 36 - Resultado final da bipartição do nós da mallha de acordo com vetor $leaves$.

Já a matriz $child$ é definida tal que $child[i]$ contém os filhos do no i. No entanto, até esse ponto a matriz ainda não inclui as folhas, o que é feito no próximo passo.

A rotina anterior define todos os nós e folhas da árvore e também suas relações. A próxima etapa consiste em utilizar esses dados para montar a árvore binária. Isto é feito pela Rotina 3, a qual é parte da função $cluster$.

Rotina 3 (R3) - Parte da função $cluster(\boldsymbol{X}, max\_elem, \eta)$ responsável por montar a árvore binária.

```
1  …
2  Tree = Array{Any}(inode+ileaf-1)
3    for i=1:inode
4      Tree[i] = nodes[i];
5      if child1[i] > 0
6        child[i,1] = child1[i] + inode - 1;
7      end
8      if child2[i] > 0
9        child[i,2] = child2[i] + inode - 1;
10     end
11   end
12   for i=1:ileaf-1
13     Tree[inode+i] = leaves[i];
14     t1,t2,d,c = divnode(X,leaves[i]);
15     center_row[inode+i,:] = c;
16     diam[inode+i] = d;
17     child[inode+i,:] = [0 0];
18   end
19 …
```

A linha 2 da Rotina 3 define o vetor que armazena todos os blocos da árvore e consequentemente o tamanho dessa, valor dado pela soma do número de nós ($inode$) com o número de folhas ($ileaf - 1$). Primeiramente, todos os nós são adicionados à árvore (R3, l.4) e caso aquele nó tenha como filho uma folha, essa folha é adicionada à matriz $child$ (R3, ll.5-10). Depois de adicionados todos os nós a árvore, são adicionadas as folhas (R3, l.13).

As folhas, não sofrem divisão, por isso essas não passaram pela função $divnode$. No entanto, essa função é responsável não só por bipartir os blocos mas por calcular o centro e o diâmetro destes. Com esse intuito, essa função é chamada novamente, mas agora para as folhas, para a obtenção de seus diâmetros e centros (R3, ll.15-17). Para o problema modelo, obtemos:

Tabela 12 - Resultados obtidos pela Rotina 3.

Tree =	child =	Diam =	center_row =
[[1:44], [1:6,28:44], [7:27], [28:39], [1:6,40:44], [17:27], [7:16], [28:33], [34:39]' [1,40:44], [2:6], [22:27], [17:21]]	[2 3; 4 5; 6 7; 8 9; 10 11; 12 13; 0 0; …]	[1.35…, 1.23…, 1.17…, 0.777…, 0.732…, 0.732…, 0.633…, 0.455…, 0.455…, 0.440…, 0.364…, 0.440…, 0.364…, 0.00 …]	[0.5 0.5; 0.142… 0.5; 0.891… 0.5; 0.136… 0.863…; 0.148… 0.103…; 0.896… 0.851…; 0.886… 0.113…; 0.272… 1.0; 0.0 0.727…; 0.007… 0.189…; 0.318… 0.0; 0.810… 0.992…; 1.0 0.681… 0 0 …]

Os dados do vetor $Tree$ e da matriz $child$ são suficientes e necessários para montar a árvore binária do problema dado como modelo. Para estes, temos que $Tree[i]$ retorna os nós da malha pertencentes ao bloco i da árvore e $child[i,:]$ retorna os filhos do nó i. Já o vetor $diam$ e a matriz $center\_row$, essenciais para examinar condição de admissibilidade dos blocos, são definidos tal que $diam[i]$ e $center\_row[i]$ forneçam, respectivamente, o diâmetro e as coordenadas do centro do bloco i.

Na forma gráfica, a árvore do problema é dada pela Figura 37. Observe dos resultado obtidos temos que $Tree[3] = [7{:}\,27]$, $child[3,:] = [6 \quad 7]$, $diam[3] = 1.17$ ... e $center\_row[3] = [0.891$ ... $\quad 0.5]$, o mesmo pode ser observado da Figura 37.

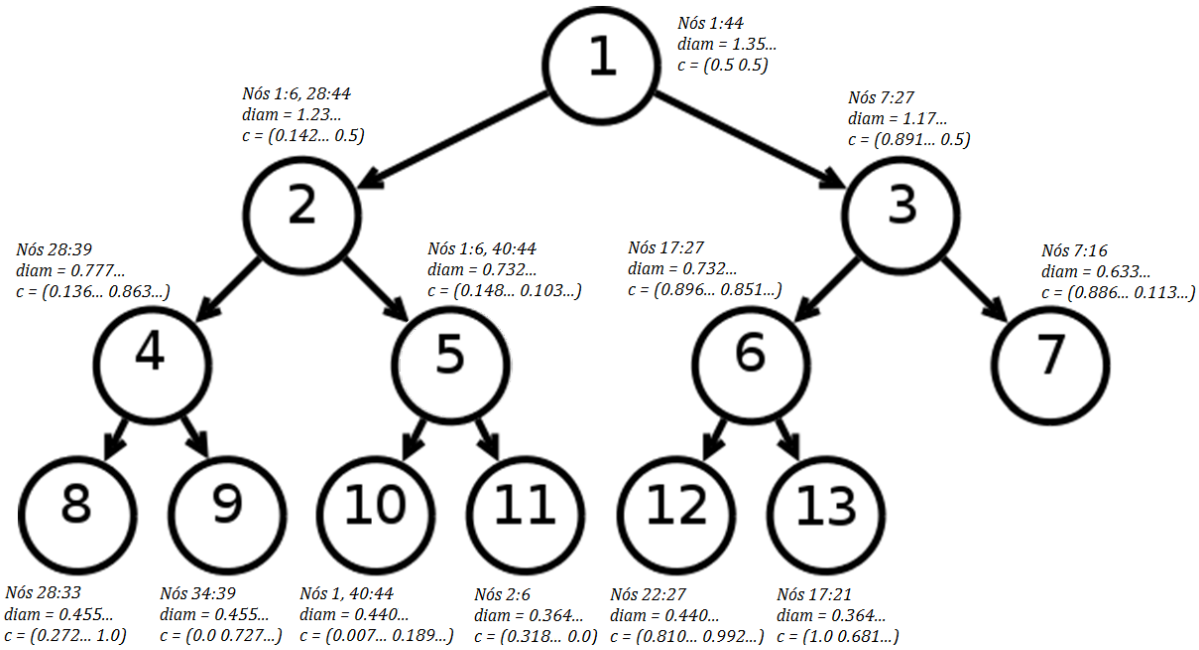

Figura 37 - Árvore binária resultante da aplicação das Rotinas 1 a 3 para o problema modelo.

6.3.3 CONDIÇÃO DE ADMISSIBILIDADE

Após a formação da árvore binária, o próximo passo é avaliar a condição de admissibilidade. Essa condição permite avaliar quais bloco de dados serão aproximada pelo método ACA e quão serão calculados de forma exata.

Antes de avaliarmos essa condição, é importante entender a relação entre a árvore e a condição de admissibilidade com a matriz que descreve o problema. A árvore T_I, obtida pelas rotinas anteriores (Figura 37), engloba somente um conjunto de dados I, os nós da malha. Por outro lado, a matriz que descreve o problema é uma matriz $\boldsymbol{A} \in \mathbb{R}^{nxn}$, onde n é também o número de nós da malha. De forma mais rigorosa, $\boldsymbol{A} \in \mathbb{R}^{mxn}$ é uma matriz de elementos de contorno, onde m representa os pontos fonte e n os pontos campo e cada par coordenado mxn representa a interação de um ponto fonte com um ponto campo. Em alguém momento, cada nó da malha é um ponto campo ou um ponto fonte, o que resulta em uma matriz quadrada $\boldsymbol{A} \in \mathbb{R}^{nxn}$. Deste modo, a árvore T_I representa na verdade uma árvore T_{IxJ}, onde I são os pontos fonte e J os pontos campo. Assim como os índices da matriz $\boldsymbol{A}$ são um produto cartesiano de mxn, os blocos a serem avaliados são também um produto cartesiano de IxJ. A admissibilidade é então avaliada entre um bloco de pontos fonte e um bloco de pontos campo. Enquanto que para os nós 2 e 3 de uma árvore T_I seria analisado simplesmente a admissibilidade entre 2 e 3, para uma arvore T_{IxJ} é avaliado 2 e 2, 2 e 3, 3 e 2 e 3 e 3, onde o primeiro é um bloco de pontos fonte é o segundo de pontos

campo. Só é possível representar uma árvore T_{IxJ} por uma T_I pois, como já mencionado, cada nó da malha é em algum momento um ponto campo ou um ponto fonte. Quando dois blocos são admissíveis, o produto cartesiano de seus nós fornece os índices dos elementos de $\boldsymbol{A}$ que formarão a submatriz a ser aproximada.

A rotina responsável por avaliar a admissibilidade dos blocos, Rotina 4, é composta pela parte final da função $cluster$ e toda a função $blocks$.

Rotina 4 (R4) - Função $blocks\,(Tree, child, allow)$ responsável por avaliar a condição de admissibilidade

```
1  …
2  admiss = zeros(inode+ileaf-1,inode+ileaf-1)
3      for i=1:inode+ileaf-1
4          for j=1:inode+ileaf-1
5        admiss[i,j]=η*norm(center_row[i]-center_row[j],2)-max(diam[i],diam[j])
6          end
7      end
8      allow = admiss.>=0
9      block = blocks(Tree,child,allow)
10     return  Tree,block
11 end
12
13 function blocks(Tree,child,allow)
14     fc1 = [2; 2; 3; 3]
15     fc2 = [2; 3; 2; 3]
16     block = zeros(Int,0,3)
17     c1 = 0
18     while c1 < length(fc1)/2
19         for i=1:2
20             if allow[fc1[c1*2+i],fc2[c1*2+i]]==1
21                 block = vcat(block,[fc1[c1*2+i] fc2[c1*2+i] 1])
22             else
23                 if child[fc1[c1*2+i],1]==0 && child[fc2[c1*2+i],1]==0
24                     block = vcat(block,[fc1[c1*2+i] fc2[c1*2+i] 0])
25                 else
26                     if length(Tree[fc1[c1*2+i]])>=length(Tree[fc2[c1*2+i]])
27                         fc1 = [fc1; child[fc1[c1*2+i],:]]
28                         fc2 = [fc2; fc2[c1*2+i]; fc2[c1*2+i]]
29                     else
30                         fc1 = [fc1; fc1[c1*2+i]; fc1[c1*2+i]]
31                         fc2 = [fc2; child[fc2[c1*2+i],:]]
32                     end
33                 end
34             end
35         end
36         c1 = c1 + 1
37     end
38     return block
39 end
```

A parte inicial da rotina (R4, l.2), cria uma matriz chamada $\boldsymbol{admiss}$, essa guarda o resultado obtido da aplicação da condição de admissibilidade entre os blocos (R4, ll.3-7). Como há n blocos, essa matriz $\in \mathbb{R}^{nxn}$. O resultado é avaliado e uma matriz booleana, com valores de verdadeiro ou falso, $allow \in \mathbb{R}^{nxn}$ é criada para

guardar as informações da admissibilidade entre os blocos. Neste ponto, já são conhecidos os blocos que são e não são admissíveis, mas ainda é preciso organizar essa informação para a aplicação do método ACA, o que é feito pela função $blocks$.

		Nós						Folhas						
		1	2	3	4	5	6	7	8	9	10	11	12	13
Nós	1	false	false	false	false	false	false	false	false	false	false	false	false	false
	2	false	false	true	false	false	true	true	false	false	false	false	true	true
	3	false	true	false	true	true	false	false	true	true	true	false	false	false
	4	false	false	true	false	false	true	true	false	false	false	false	true	true
	5	false	false	true	false	false	true	true	false	false	false	false	true	true
	6	false	true	false	true	true	false	false	true	true	true	true	false	false
Folhas	7	false	true	false	true	true	false	false	true	true	true	true	false	false
	8	false	false	true	false	false	true	true	false	true	true	false	true	true
	9	false	false	true	false	false	true	true	true	false	false	true	true	true
	10	false	false	true	false	false	true	true	true	false	false	true	true	true
	11	false	false	false	false	false	true	true	false	true	true	false	true	true
	12	false	true	false	true	true	false	false	true	true	true	true	false	false
	13	false	true	false	true	true	false	false	true	true	true	true	false	false

Figura 38- Representação da matriz $allow$.

Para o problema modelo e um valor de η = 2.0 a matriz $allow$ obtida é dada pela Figura 38. Analisar essa matriz permite entender melhor a condição de admissibilidade, tornando o processo de avalição dos blocos admissíveis mais eficiente. Primeiro, a admissibilidade de um bloco com ele mesmo (em vermelho na Figura 38) será sempre falsa pois a distância entre esses dois blocos é sempre nula. Segundo, a matriz $allow$ é sempre simétrica, pois se o bloco 2 é admissível com 3, a reciproca também é verdadeira. Terceiro, se um bloco é admissível com outro, seu filhos também são admissíveis com este. Por exemplo se o bloco 2 é admissível com bloco 3, os blocos 4 e 5 também serão também admissível com o bloco 3. Por último, embora tenha sido calculado na matriz $allow$, não se analisa a admissiblidade de um bloco com o bloco 1, pois este representa todos os nós da malha. Considerando essas observações antes do cálculo da matriz $allow$, a quantidade de blocos a serem

analisados pela condição de admissiblidade pode ser reduzida de 169 para 78. Durante o cálculo, essa quantidade sofre ainda maior redução considerando a admissibilidade entre blocos admissíveis e seus filhos.

Parte desses conceitos são aplicados na função $blocks$. Essa função testa a admissibilidade de certos blocos e os salva em uma matriz chamada $block$. Primeiramente, a função define os primeiros blocos a serem testados (R4, ll.14-15). O algoritmo testa a admissibilidade de $fc1(k)$ com $fc2(k)$. Se os blocos são admissíveis, eles são adicionados a linha da matriz juntos com o indicador 1 (admissível) (R4, ll.20-22). Se os blocos não são admissíveis e são folhas eles são adicionados a matriz com o indicador 0 (não admissível) (R4, ll.23-25). Se eles não são admissíveis e nem folhas, seus filhos são adicionados aos conjunto de blocos que devem ser testados (R4, ll.26-31).

Para o problema modelo e um valor de η = 2.0 a função $blocks$ retorna a matriz $block$ dada por:

Tabela 13 - Resultados obtidos pela função $blocks$.

```
Block =
 [2   3  1;
  3   2  1;
  7   7  0;
 12   7  0;
 13   7  0;
  7  12  0;
  7  13  0;
  8   8  0;
  8   9  1;
  9   8  1;
  9   9  0;
  8  10  1;
  8  11  0;
  9  10  0;
```

```
  9  11  1;
 10   8  1;
 11   8  0;
 10   9  0;
 11   9  1;
 10  10  0;
 10  11  1;
 11  10  1;
 11  11  0;
 12  12  0;
 12  13  0;
 13  12  0;
 13  13  0]
```

Essa matriz contém 27 pares de blocos que são ou não admissíveis. Para a obtenção desses 27 pares de blocos a função $blocks$ analisou 50 pares, como esperado, esse número é menor que os 78 máximo previstos. Esses blocos cobrem toda a matriz original do problema e permite mesmo antes de qualquer cálculos conhecer as submatrizes que serão aproximadas ou não (Figura 39).

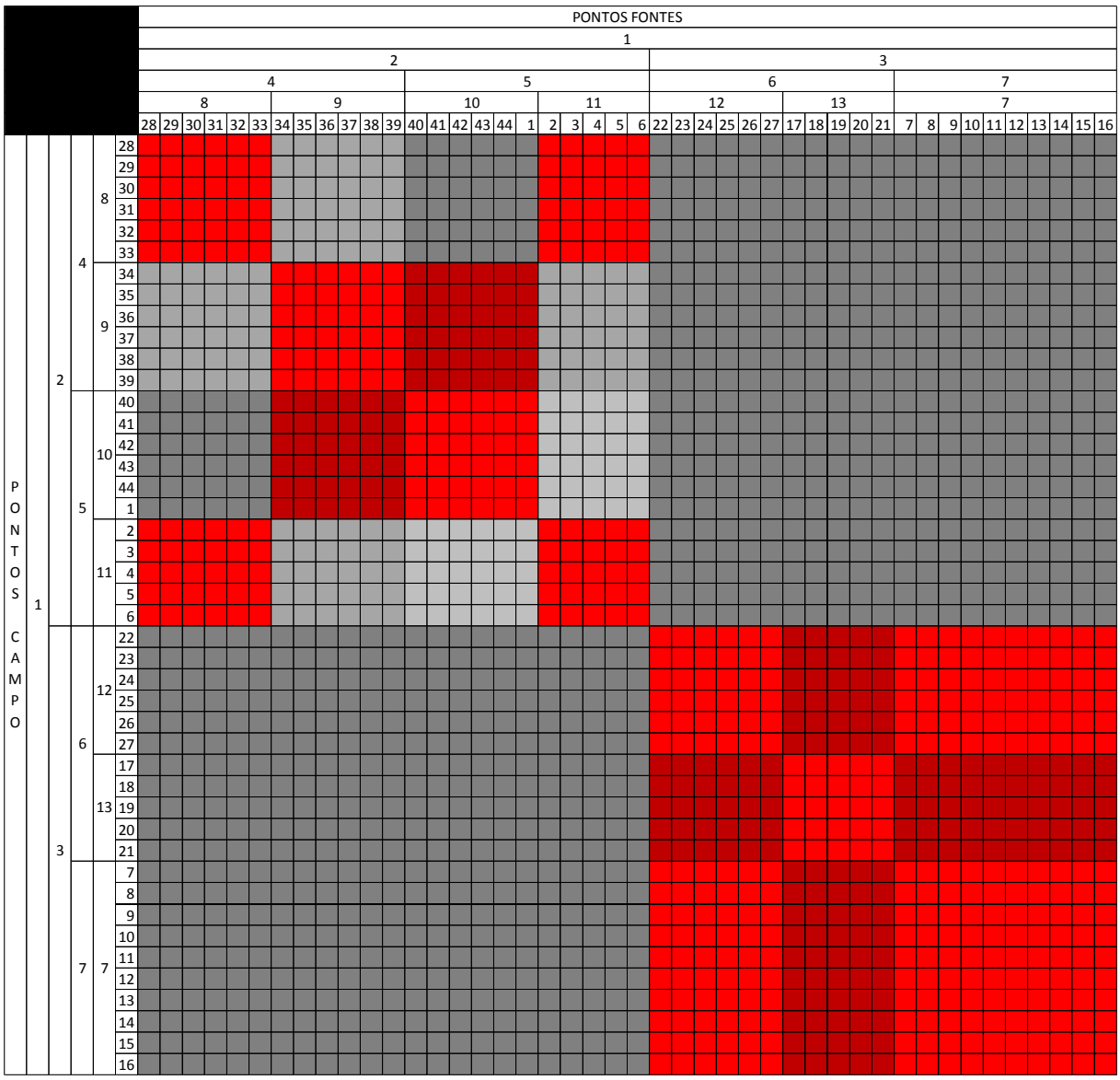

Figura 39 - Submatrizes que serão aproximadas (cinza) e que serão calculadas exatas (vermelho).

Para melhor compreensão de como a montagem da matriz é feita, foi descrita a acima e ao lado esquerdo dessa a árvore binária obtida para o problema para os pontos fonte e para os pontos campo, respectivamente. Da primeira linha da matriz temos $[2 \quad 3 \quad 1]$, ou seja, o nó 2 dos pontos fonte é admissível com o nó 3 dos pontos campo, assim toda área da matriz correspondente a esses nós (área cinza no 3º quadrante) será calculada de forma aproximada pelo método ACA. Conduzindo essa mesma análise para as outras linhas da matriz $block$, resulta na Figura 39.

A parâmetro η é a constante que controla a formação desses blocos, a sua influência na formação das submatrizes e na condição de admissibilidade pode ser observada na Figura 40.

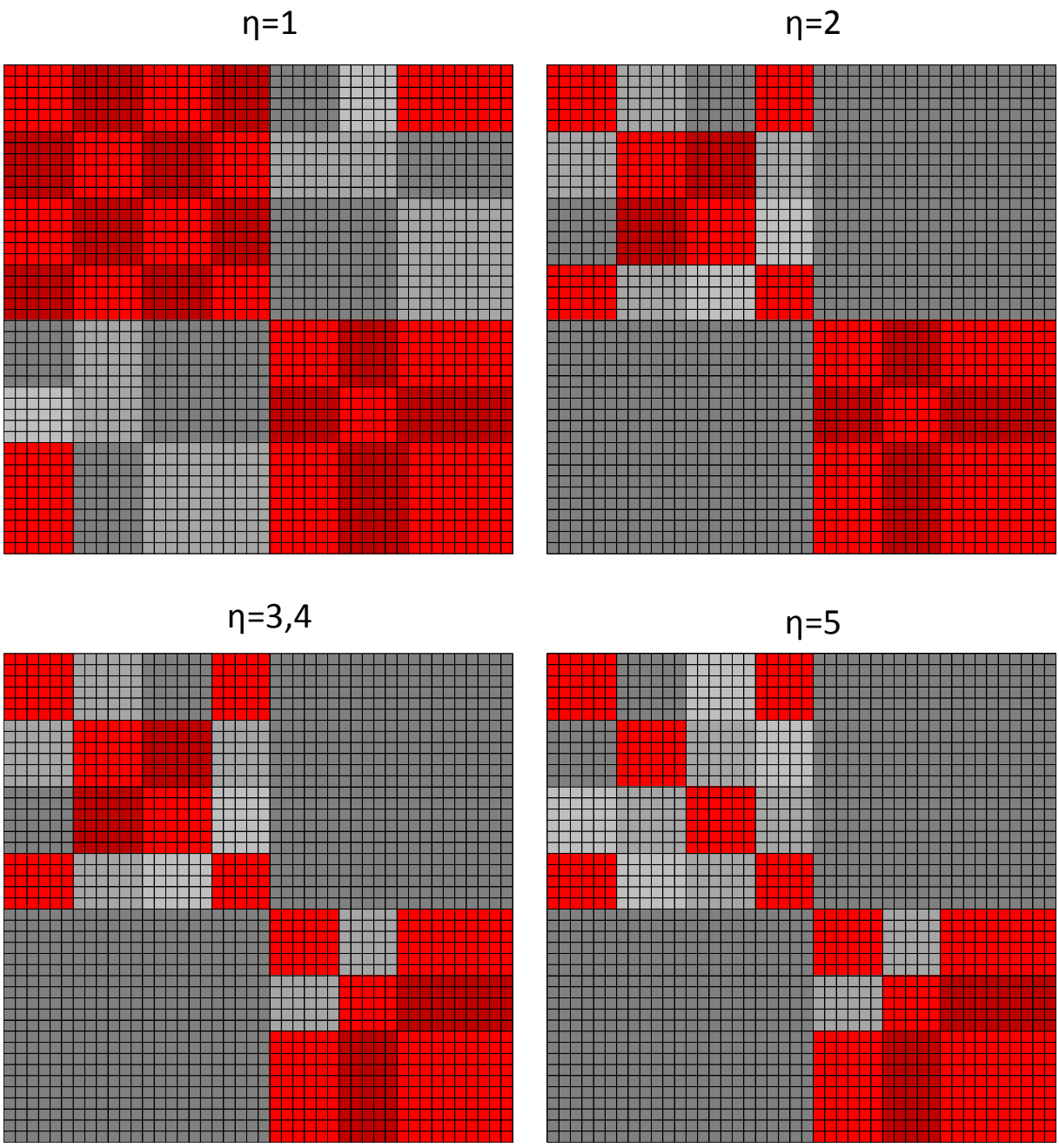

Figura 40 - Submatrizes resultantes para vários valores de η

Podemos observar que quanto maior o valor de η, maior é a quantidade de submatrizes que podem ser representadas por matrizes aproximadas. No entanto, não é desejável aproximar uma quantidade grande de submatrizes pequenas, pois a aproximação de matrizes pequenas pelo método ACA apresenta baixa precisão, o que afeta negativamente a precisão da solução final, algo que é indesejável. Por outro lado, um η muito pequeno resulta não somente em poucas matrizes a serem aproximadas, mas inibe o aparecimento de submatrizes grandes, sendo necessário calcular uma grande quantidade de submatrizes de forma exata e apresentando baixa precisão daquelas aproximadas.

O valor de $\eta = 2$, valor considerado padrão, parece ser a melhor opção pois apresenta uma pequena quantidade de matrizes pequenas quando comparados com valores maiores de η e também submatrizes maiores quando comparado com valores menores de η.

6.4 APROXIMAÇÃO CRUZADA ADAPTATIVA

As submatrizes que formam a matriz hierárquica podem ser calculadas uma vez que se conhece a condição de admissibilidade entre os blocos e a árvore binária que descreve a relação entre elas. O cálculo dessas submatrizes é feito pela função $ACAF\_analitico(Tree, block, fHeG, arg, erro)$. Essa é uma função extensa, que será discutida neste capítulo por etapas. Os dados de entrada dessa função são a árvore binária ($tree$), os blocos a serem calculados e suas condições de admissibilidade ($blocks$), uma função $fHeG$ responsável por calcular os elementos das matrizes $\boldsymbol{H}$ e $\boldsymbol{G}$ do problema de condução de calor, uma série de argumentos necessários para o cálculo desses elementos (arg) e por fim o erro admissível para a aproximação do método ACA.

A primeira parte da função $ACAF\_analitico$ é responsável por calcular os blocos exatos, ou seja, aqueles que não serão aproximados (Rotina 5). Primeiramente, é criada uma matriz chamada $\boldsymbol{Aaca}$ e um vetor b. A matriz tem duas colunas e número de linhas igual ao número de blocos e é responsável por guardar todas as submatrizes calculadas. O vetor é o vetor b do problema $\boldsymbol{A}x = b$ e tem comprimento igual ao número de nós (R5, ll.4 -6).

Rotina 5 (R5) - Parte da função $ACAF\_analitico$ responsável pelo cálculo dos blocos exatos

```
1  function ACAF_analitico(Tree,block,fHeG,arg,erro=1e-5)
2      # arg = [NOS1,NOS_GEO1,tipoCDC,valorCDC,normal,ELEM1,k]
3      #         1      2        3      4        5      6   7
4      n = size(block,1)
5      Aaca = Array{Any}(n,2)
6      b = zeros(size(arg[1],1))
7      for i=1:n
8          b1 = Tree[block[i,1]]
9          b2 = Tree[block[i,2]]
10         if block[i,3]==0
11             Aaca[i,1],B = fHeG(b1,b2,arg)
12             b[b1] = b[b1] + B*arg[4][b2]
13         else
14            . . . (Rotina 6)
15         end
16      end
17     return Aaca,b
18 end
```

Todos os pares de blocos da matriz $blocks$ são avaliados. Os blocos não admissíveis são calculados de forma exata pela função $fHeG$ (R5, ll.10-13). Essa função calcula os elementos das matrizes $\boldsymbol{H}$ e $\boldsymbol{G}$ do problema $[\boldsymbol{H}]\{T\} = [\boldsymbol{G}]\{q\}$, na forma $\boldsymbol{A}x = \boldsymbol{B}b$, onde tanto $\boldsymbol{A}$ quanto $\boldsymbol{B}$ contém elementos de $\boldsymbol{H}$ e $\boldsymbol{G}$. O primeiro associados as variáveis desconhecidas x e o segundo as conhecias b. São utilizados como dados de entrada os nós pertencentes ao par de blocos analisados, $b1$ e $b2$, e certos argumentos necessários para o cálculo. São computados os elementos das matrizes $\boldsymbol{H}$ e $\boldsymbol{G}$, cujos os índices representam o produto cartesiano dos nós $b1$ e $b2$.

Para $i = 1$ (R5, l.7), temos $b1 = 1{:}6, 28{:}44$ (bloco 2), $b2 = 7{:}27$ (bloco 3) e $block[i, 3] = 1$. Assim, a submatriz correspondente a esses blocos é calculada de forma aproximada pelo método ACA. Quando os blocos podem ser aproximados, ou seja, a condição (R5. l10) é falsa, é executada a Rotina 6, iniciando o método.

Rotina 6 (R6) - Início do método ACA+, cálculo da linha auxiliar.

```
1  INDB1=[]
2  INDB2=[]
3  B1 = zeros(0,length(b2))
4  B2 = zeros(length(b1),0)
5  ind1 = trues(length(b1))
6  ind2 = trues(length(b2))
7  indaref = 1
8  aref = 0*ind2
9  for indaref = 1:length(ind1)
10     aref,btemp = fHeG(b1[indaref],b2,arg)
11     aref = aref.'
12     push!(INDB1,indaref)
13     B1 = [B1;btemp]
14     if norm(aref) > 1e-10
15         break
16     end
17     ind1[indaref] = 0
18 end
19 if ind1==falses(ind1)
20     Aaca[i,1] = ind1*0
21     Aaca[i,2] = ind2.'*0
22 else
23 . . . (Rotina 7)
24 end
```

As linhas 1 a 8 da Rotina 6 criam e alocam espaço para variáveis que serão usadas durante a rotina. As linhas seguintes correspondem a uma aplicação conservadora do método ACA+. A linha 9 inicia o processo: a primeira linha é calculada (R6, l.10), se a norma da linha for maior que 1e-10 o processo é finalizado, caso contrário é calculado a próxima linha. O processo somente continuará se a norma

for menor que 1e-10, ou seja, se todos os elementos da linha forem nulos. Neste caso, outra linha é calculada para evitar o aparecimento de um pivô nulo.

No entanto, diferentemente do método ACA+ que calcula somente duas linhas e uma coluna para determinar se a submatriz é uma matriz nula, independentemente de suas dimensões. O método aqui implementado calcula todas as linhas antes de fazer essa determinação. A condição dada na linha 19 só é verdadeira se todas as linhas foram calculadas e todas apresentarem norma menor do que 1e-10. Se verdadeiro, a submatriz é definida como uma matriz nula, produto de $\boldsymbol{U} = 0$ (R6, l.20) e $\boldsymbol{V} = 0$ (R6, l.20-21).

Esse método é mais conservativo por calcular todas as linhas da matriz. Isso requer uma quantidade maior de cálculos, não se aproveitando o comportamento observado nas matrizes advindas do MEC. Para o problema modelo, um elemento da matriz $\boldsymbol{H}$ será nulo quando os pontos fontes e campos analisados pertencerem a uma mesma aresta, com exceção de quando esses coincidem, pois o cálculo desses componentes envolve o produto $\vec{n} \cdot \vec{r}$ onde $\vec{n}$ e o vetor normal ao contorno e $\vec{r}$ o vetor que vai do ponto fonte ao ponto campo. Como esses vetores são ortogonais no caso descrito anteriormente, o produto resultante é nulo. Assim, é possível conhecer os elementos nulos da matriz $\boldsymbol{H}$ (Figura 41).

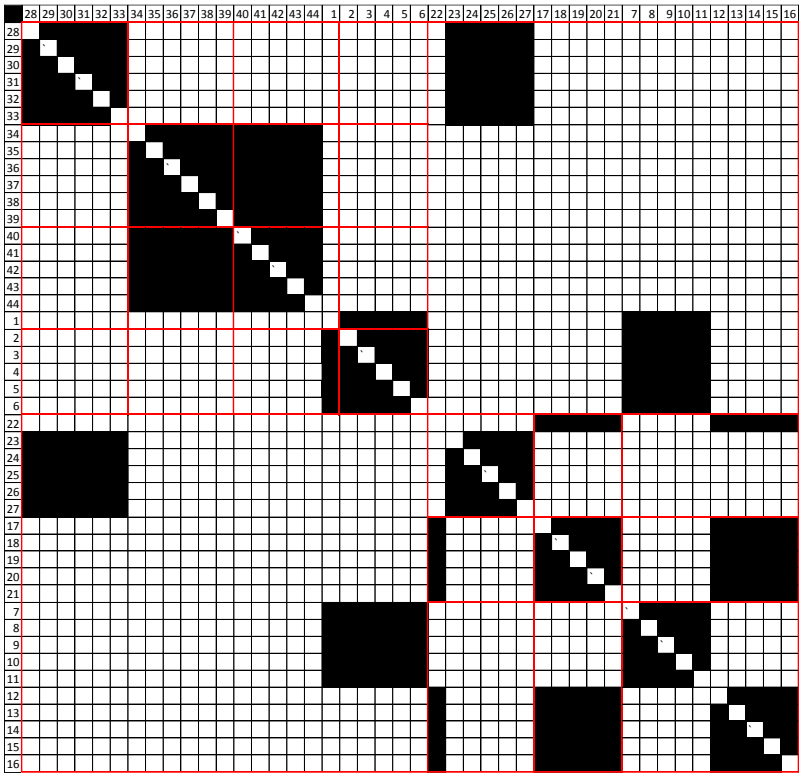

Figura 41 - Elementos nulos (Preto) de cada uma das submatrizes (Vermelho) da matriz $\boldsymbol{H}$.

É possível observar que para qualquer submatriz da matriz $\boldsymbol{H}$, duas linhas e uma coluna seriam o suficiente para classificarmos aquela submatriz como uma matriz nula ou não. Assim, não seria necessário uma aplicação conservadora do método ACA+.

Considerando somente os nós dados por $b1$ e $b2$ anteriores (blocos 2 e 3) e reorganizando os nós de forma crescente (Figura 42) podemos observar os elementos nulos desse bloco, o que é confirmado pelo cálculo da linha auxiliar $aref$.

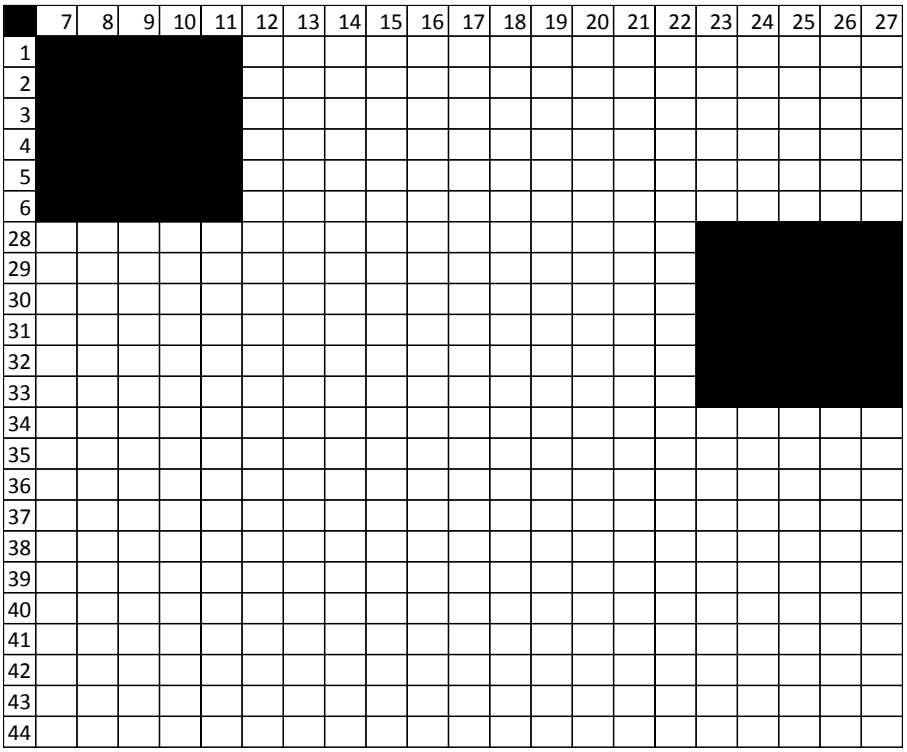

Figura 42 - Indices nulos (Preto) da Submatriz formada pelos blocos 2 e 3.

Da Rotina 6 podemos extrair os valores de $aref$ (Tabela 14), os quais são os elementos calculados da matriz $\boldsymbol{A}$ pertencentes a linha calculada da submatriz. Neste caso, somente a primeira linha foi calculada. Podemos observar da Tabela 14 que os cinco primeiros valores são nulos, o que é confirmado pela Figura 42. Da tabela, também temos os valores calculados da matriz $\boldsymbol{B}$ $(btemp)$, os índices das linhas calculadas $(INDB1)$ e um vetor $ind1$ tal que $ind1[i] = false$ se a linha i tiver sido calculada, ou $true$ caso contrário.

Tabela 14 - Resultados extraidos da Rotina 6.

aref =	btemp =	INDB1 =	ind1 =
[0.0,	[0.0087…,	[1]	[false
0.0,	0.0065…,		true
0.0,	0.0046…,		true
0.0,	0.0029…,		true
0.0,	0.0013…,		true
-0.00065…,	-0.0151…,		true
-0.00052…,	-0.0148…,		true
-0.00026…,	-0.0143…,		true
0.00005…,	-0.0136…,		true
0.00054…,	-0.0128…,		true
0.00108…,	-0.0118…,		true
0.00167…,	-0.0109…,		true
0.00231…,	-0.0100…,		true
0.00297…,	-0.0091…,		true
0.00365…,	-0.0083…,		true
0.00434…,	-0.0075…,		true
0.00792…,	-0.0043…,		true
0.00866…,	-0.0037…,		true
0.00946…,	-0.0030…,		true
0.01029…,	-0.0024…,		true
0.01115…,]	-0.0018…,]		true]

Uma vez definido que a submatriz não é uma matriz nula (R6, l.19) é calculada a coluna auxiliar (Rotina 7).

Rotina 7 (R7) - Método ACA+, cálculo da coluna auxiliar.

```
1   indbref = 1
2   bref = 0*ind1
3   for ii = 1:length(ind2)
4       indbref = indmin(abs(aref[ind2])) # ACA+
5       indbref = indbref + cumsum(ind2.==0)[ind2][indbref]
6       bref,btemp = fHeG(b1,b2[indbref],arg)
7       push!(INDB2,indbref)
8       B2 = [B2 btemp]
9       if norm(bref) > 1e-10
10          break
11      end
12      ind2[indbref] = 0
13  end
14  . . . (Rotina 8)
```

O processo é feito de maneira similar à Rotina 6. A diferença está na escolha do índice da coluna a ser calculada (R7, ll.4-5) que é escolhido como o índice do menor valor absoluto da linha auxiliar anteriormente calculada. Esse índice define o índice da coluna auxiliar.

Analogamente a Rotina 6, podemos extrair da Rotina 7 os valores de $bref$ (Tabela 15), os quais são os elementos calculados da matriz $\boldsymbol{A}$ pertencentes a coluna calculada da submatriz. Podemos observar da Tabela 15 que os seis primeiros valores são nulos, o que é confirmado pela Figura 42. Da tabela, também temos os valores calculados da matriz $\boldsymbol{B}$ $(btemp)$, os índices das linhas calculadas $(INDB1)$, os índices

das colunas calculadas $(INDB2)$ e um vetor $ind2$ tal que $ind2[i] = false$ se a coluna i tiver sido calculada, ou $true$ caso contrário.

Tabela 15 - Resultados extraidos da Rotina 7

bref =	btemp =	INDB1 =	ind2 =
[0.0;	[0.0087…;	[1]	[false
0.0;	0.0114…;		true
0.0;	0.0146…;		true
0.0;	0.0188…;		true
0.0;	0.0248…;		true
0.0;	0.0353…;		true
0.01434…;	0.0000…;		true
0.01399…;	-0.0002…;		true
0.01346…;	-0.0005…;		true
0.01277…;	-0.0009…;		true
0.01198…;	-0.0013…;		true
0.01115…;	-0.0018…;	INDB2 =	true
0.01095…;	-0.0016…;	[1]	true
0.01141…;	-0.0006…;		true
0.01181…;	0.0003…;		true
0.01212…;	0.0014…;		true
0.01225…;	0.0025…;		true
0.01209…;	0.0037…;		true
0.01148…;	0.0047…;		true
0.01025…;	0.0057…;		true
0.00823…;	0.0066…;		true
0.00539…;	0.0072…;		true
0.00188…;]	0.0075…;]		true]

Após o cálculo da linha e coluna auxiliar, é iniciado o cálculo das matrizes de baixo posto $\boldsymbol{U}$ e $\boldsymbol{V}$ (Rotina 8). A matriz resultante do produto dessas, é a matriz aproximada que representa a submatriz. Primeiramente, são definidos dois pivôs (R8, ll.1-2), um para a linha de referência e outro para a coluna. Devido ao modo como o método ACA+ foi implementado, não é necessário comparar o subpivô da linha (maior valor absoluto da linha) com o pivô auxiliar (menor valor absoluto da linha) calculado anteriormente. Em seguida, são criadas e alocados espaço para variáveis que serão utilizadas durante o processo (R8, ll.3-10).

O processo é realizado no máximo $nmin - 1$ vezes (R8, l.11), onde $nmin$ é o menor número entre o número de linhas e colunas da submatriz, pois para uma matriz $\boldsymbol{A} \in \mathbb{R}^{m\,x\,n}$, o número mínimo entre m e n indica o número de linhas ou colunas linearmente independentes, ou seja, o número máximo de linhas necessárias para representar o espaço gerado por aquela matriz de forma exata. Esse valor ainda é subtraído de um, pois uma linha e uma coluna já foram calculados.

O processo começa escolhendo o maior subpivô entre o da linha e o da coluna (R8, l.12). Além desse valor influenciar na precisão da aproximação calculada, também depende dele se o próximo a ser calculado será uma linha ou uma coluna.

Rotina 8 (R8) - Metodo ACA+, cálculo das matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$.

```
1  arefmax = indmax(abs.(aref))
2  brefmax = indmax(abs.(bref))
3  Umax = 0
4  Vmax = 0
5  nmin = min(length(ind1),length(ind2))
6  U = zeros(length(ind1),nmin)
7  V = zeros(nmin,length(ind2))
8  Ap = zeros(length(ind1),length(ind2))
9  norma0 = 0.0
10 cont = 0
11 for cont = 1:nmin-1
12     if abs.(aref[arefmax])>abs.(bref[brefmax])
13         U[:,cont],btemp = fHeG(b1,b2[arefmax],arg)
14         U[:,cont] = U[:,cont] - Ap[:,arefmax]
15         push!(INDB2,arefmax)
16         B2 = [B2 btemp]
17         Umax = indmax(abs.(U[:,cont]))
18         V[cont,:],btemp = fHeG(b1[Umax],b2,arg)
19         V[cont,:] = (V[cont,:][:]-Ap[Umax,:]).'/U[Umax,cont]
20         push!(INDB1,Umax)
21         B1 = [B1;btemp]
22         Vmax = arefmax
23     else
24         V[cont,:],btemp = fHeG(b1[brefmax],b2,arg)
25         V[cont,:] = (V[cont,:][:]-Ap[brefmax,:])
26         push!(INDB1,brefmax)
27         B1 = [B1;btemp]
28         Vmax = indmax(abs.(V[cont,:]))
29         U[:,cont],btemp = fHeG(b1,b2[Vmax],arg)
30         U[:,cont] = (U[:,cont]-Ap[:,Vmax])/V[cont,Vmax]
31         push!(INDB2,Vmax)
32         B2 = [B2 btemp]
33         Umax = brefmax
34     end
35     Ap = U*V
36     norma1 = vecnorm(Ap)
37     if abs.((norma1-norma0)/norma1) < erro
38         break
39     else
40         norma0 = norma1
41     end
42     ind1[Umax] = 0
43     ind2[Vmax] = 0
44     . . . (Rotina 9)
45     arefmax = indmax(abs.(aref))
46     brefmax = indmax(abs.(bref))
47     Aaca[i,1] = U[:,1:cont]
48     Aaca[i,2] = V[1:cont,:]
49 end
50 . . . (Rotina 10)
```

O processo de cálculo das novas linhas e colunas (R8, ll.12-34) é similar aos vistos anteriormente. As diferenças estão no cálculo das matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$, que são calculadas pela diferença entre a linha ou coluna calculada com a sua correspondente na matriz aproximada $\boldsymbol{Ap}$ (R8, ll.14, 19, 25 e 30). Essas ainda devem ser divididas pelo valor do pivô (R8, ll.19 e 30). Somente uma é dividida, a escolha é intercambiável.

A nova matriz aproximada $\boldsymbol{Ap}$ e sua norma são calculadas (R8, ll.35-36). O valor do erro da aproximação é avaliado, se esse for menor que o estipulado o processo para (R8, ll.37-39), não sendo necessário calcular novas linhas ou colunas. Caso contrário, a norma da aproximação anterior é atualizada (R8, l.40), assim como os vetores booleanos que guardam as linhas e colunas já calculadas (R8, l.44). Ao final, são atualizados os pivôs da linha e coluna de referência (linhas 45-46) e a matriz $\boldsymbol{Aaca}$ com os novos valores de $\boldsymbol{U}$ e $\boldsymbol{V}$ (R8, ll.47-48).

Entre cada iteração do processo é preciso avaliar se a linha e/ou coluna calculada coincide com aquelas de referência. Se isso ocorre, é necessário calcular novas para repor aquelas de referência, o que é feito pela Rotina 9. Existem quatro possibilidades: a linha e a coluna coincidem (R9, l.1), somente a linha coincide (R9, l.25), somente a coluna coincide (R9, l.39) ou ambas não coincidem (R9, l.53).

No primeiro caso, uma nova linha (R9, ll.2-12) e uma nova coluna (R9, ll.13-24) são calculadas. A coluna é calculada aleatoriamente, considerando somente aquelas colunas que ainda não foram calculadas (R9, ll.2-4). Seu valor é ajustado pela diferença com a correspondente da matriz $\boldsymbol{Ap}$ (R9, l.5). Em seguida, a coluna é calculada considerando também somente aquelas colunas que não foram calculadas. No entanto, como no início do processo essa é escolhida baseada no índice do menor valor absoluto da linha auxiliar (subpivô).

No segundo caso, somente uma nova linha é calculada (R9, ll.27-38). Primeiramente. A coluna é atualizada, ajustando seu valor pela diferença com a coluna correspondente da matriz $\boldsymbol{Ap}$ (R9, l.26). A linha é calculada considerando somente aquelas linhas que não foram calculadas (R9, ll.27-28), sua escolha é baseada no índice do menor valor absoluto da coluna auxiliar. Seu valor é atualizado pela diferença com a linha correspondente da matriz $\boldsymbol{Ap}$ (R9, l.26).

O terceiro caso (R9, ll.39-52) é similar ao segundo, fazendo para a coluna o que foi feito para a linha.

No último caso, somente se atualiza os valores da linha e coluna de referência.

Rotina 9 - ACA+, cálculo de nova linha e/ou coluna de referência.

```
1  if indaref==Umax && indbref==Vmax
2      for indaref = 1:sum(ind1)
3          indaref = indmax(ind1)
4          aref,btemp = fHeG(b1[indaref],b2,arg)
5          aref = aref-Ap[indaref,:]'
6          push!(INDB1,indaref)
7          B1 = [B1;btemp]
8          if norm(aref) > 1e-10
9              break
10         end
11         ind1[indaref] = 0
12     end
13     for indbref = 1:sum(ind2)
14         indbref = indmin(abs.(aref[ind2]))
15         indbref = indbref + cumsum(ind2.==0)[ind2][indbref]
16         bref,btemp = fHeG(b1,b2[indbref],arg)
17         bref = bref-Ap[:,indbref]
18         push!(INDB2,indbref)
19         B2 = [B2 btemp]
20         if norm(bref) > 1e-10
21             break
22         end
23         ind2[indbref] = 0
24     end
25 elseif indaref==Umax
26     bref = bref-U[:,cont]*V[cont,indbref].'
27     for indaref = 1:sum(ind1)
28         indaref = indmin(abs.(bref[ind1]))
29         indaref = indaref + cumsum(ind1.==0)[ind1][indaref]
30         aref,btemp = fHeG(b1[indaref],b2,arg)
31         aref=aref[:]-Ap[indaref,:][:]
32         push!(INDB1,indaref)
33         B1 = [B1;btemp]
34         if norm(aref) > 1e-10
35             break
36         end
37         ind1[indaref] = 0
38     end
39 elseif indbref==Vmax
40     aref = aref[:]-U[indaref,cont]*V[cont,:]
41     for indbref = 1:sum(ind2)
42         indbref = indmin(abs.(aref[ind2]))
43         indbref = indbref + cumsum(ind2.==0)[ind2][indbref]
44         bref,btemp = fHeG(b1,b2[indbref],arg)
45         bref = bref-Ap[:,indbref]
46         push!(INDB2,indbref)
47         B2 = [B2 btemp]
48         if norm(bref) > 1e-10
49             break
50         end
51         ind2[indbref] = 0
52     end
53 else
54     aref = aref[:] - U[indaref,cont]*V[cont,:]
55     bref = bref - U[:,cont]*V[cont,indbref].'
56 end
```

Neste ponto da rotina, o método ACA está completo para a matriz $\boldsymbol{A}$, resultando na matriz $\boldsymbol{Aaca}$. O método, no entanto, ainda não foi aplicado a matriz $\boldsymbol{B}$, isso é feito pela Rotina 10.

Rotina 10 - ACA aplicado a matriz $\boldsymbol{B}$.

```
1  max1 = ind2sub(size(B1),indmax(abs.(B1[:,INDB2])))
2  maxv = B1[max1[1],INDB2[max1[2]]]
3  Vb = B1[max1[1],:].'
4  Ub = B2[:,max1[2]]/maxv
5  Baca = Ub*Vb
6  B1 = B1 - Baca[INDB1,:]
7  B2 = B2 - Baca[:,INDB2]
8  for i = 1:size(B1,1)-1
9      max1 = ind2sub(size(B1),indmax(abs.(B1[:,INDB2])))
10     maxv = B1[max1[1],INDB2[max1[2]]]
11     if abs.(maxv) < 1e-12
12         break
13     end
14     Vb = [Vb; B1[max1[1],:].']
15     Ub = [Ub B2[:,max1[2]]/maxv]
16     lastUV = Ub[:,end]*Vb[end,:].'
17     B1 = B1 - lastUV[INDB1,:]
18     B2 = B2 - lastUV[:,INDB2]
19 end
20 b[b1] = b[b1] + Ub*Vb*arg[4][b2]
```

Diferentemente do que é feito para a matriz $\boldsymbol{A}$, não é calculada nenhuma nova linha ou coluna para calcular a aproximação da matriz $\boldsymbol{B}$ pois, durante o processo para a matriz $\boldsymbol{A}$, a cada linha ou coluna calculada era também calculada uma para a matriz $\boldsymbol{B}$, as quais foram armazenadas nas matrizes $\boldsymbol{B1}$ e $\boldsymbol{B2}$, respectivamente.

O processo começa por encontrar o elemento de maior valor em módulo, o pivô, nas linhas das colunas calculadas (R10, ll.1-2). A seguir, é definido $\boldsymbol{U}$ e $\boldsymbol{V}$ a partir da linha e coluna do pivô e a primeira aproximação é calculada (R10, ll.3-5). As linhas e colunas previamente calculadas da matriz $\boldsymbol{B}$ são então atualizadas, subtraindo essas das suas respectivas na matriz aproximada $\boldsymbol{Baca}$ (R10, ll.6-7). O processo se repete (R10, ll.l8-19) até que o pivô seja menor que 1e-12 (R10, ll.11-13) ou todas as linhas da matriz $\boldsymbol{B1}$ forem usadas (R10, l.8). Ao final, é contabilizado a contribuição da matriz $\boldsymbol{B}$ ao vetor b do problema $\boldsymbol{A}x = b$.

Finalmente, após aproximacao de todas as matrizes necessárias o problema é resolvido pelo metodo iterativo GMRES(m), utilizando um pacote externo chamado *KrylovMethods.jl* licenciado pelo MIT, o qual não será tratado neste trabalho.

MÉTODO ACA+ APLICADO NA SUBMATRIZ DOS BLOCOS 2 E 3

Para uma melhor compreensão da rotina que aplica o método ACA, será mostrado nesta seção resultados intermediários obtidos durante o cálculo aproximado da submatriz discutida anteriormente, para os blocos 2 e 3.

Os blocos da primeira linha da matriz $blocks$ ($i = 1$), são dados por $blocks[1,:] = [2 \quad 3 \quad 1]$. Deste modo, $b1$ são os nós pertencentes ao bloco 2 e $b2$ ao blocos 3 (R5, ll.8-9) e a condição $blocks[i,3] == 0$ é falsa (R5, l.10) ou seja, a matriz pode ser aproximada.

O primeiro passo é calcular a linha auxiliar, a rotina começa pela primeira linha (R6, l.9) e irá para a próxima somente se a anterior for nula, ou seja, com norma menor que 1e-10 (R6, ll.14-16). Para o problema modelo, somente a primeira linha é suficiente, o que é indicado por $INDB1$ (Tabela 16), variável que indica as linhas calculadas. É definido então um subpivô, como o elemento de menor valor em módulo da linha auxiliar e é calculado a sua coluna ($indbref$), a coluna auxiliar (R7, ll.4-6).

Em seguida, são definidos dois pivôs (R8, ll.1-2), um na linha auxiliar ($arefmax$) e outro na coluna ($brefmax$) (Figura 43).

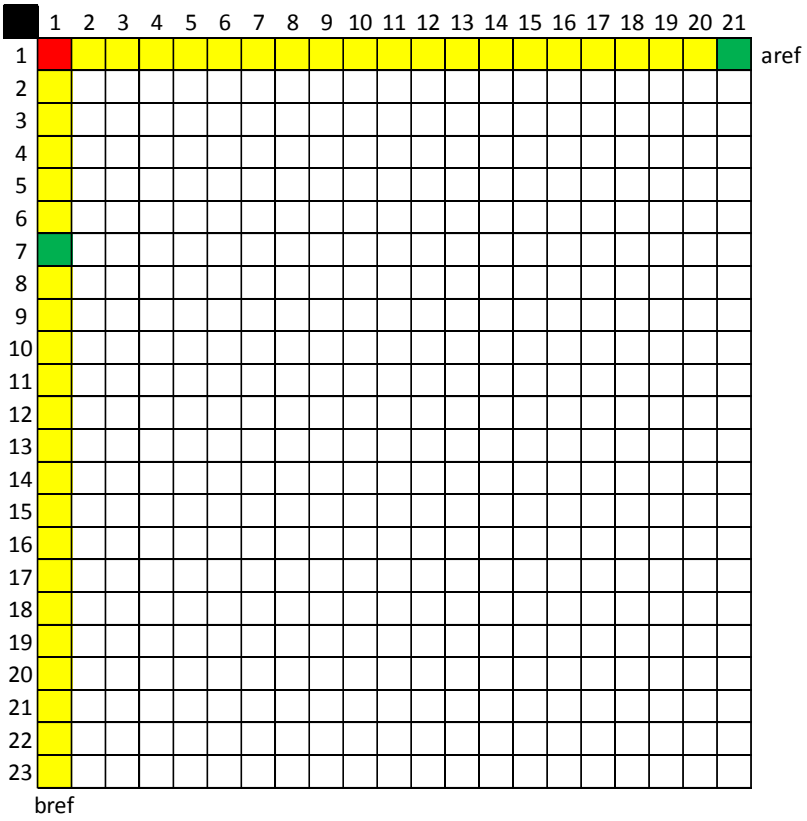

Figura 43 - Linha e coluna auxiliar inicalmente calculadas (amarelo), com subpivô (em vermelho) e pivôs (verde) em destaque.

Tabela 16 - Variáveis importantes calculadas até o resultado obtido na Figura 43.

indaref = 1 indbref = 1 arefmax = 21 brefmax = 7 aref[arefmax] = 0.0112 bref[brefmax] = 0.0143	INDB1 = [1] INDB2 = [1]	ind1 = [true true true . . . true true true]	ind2 = [true true true . . . true true true]

Como o pivô da linha auxiliar é menor que o da coluna auxiliar (R8, l.12), é calculado uma nova linha, definindo $V[1,:]$ (R8, ll. 24-25) nesta linha é encontrado um novo pivô $Vmax$ (R8, l.28) e é calculado sua coluna, definindo $U[:,1]$. Um pivô $Umax$ é também definido para a coluna (R8, ll.29-30).

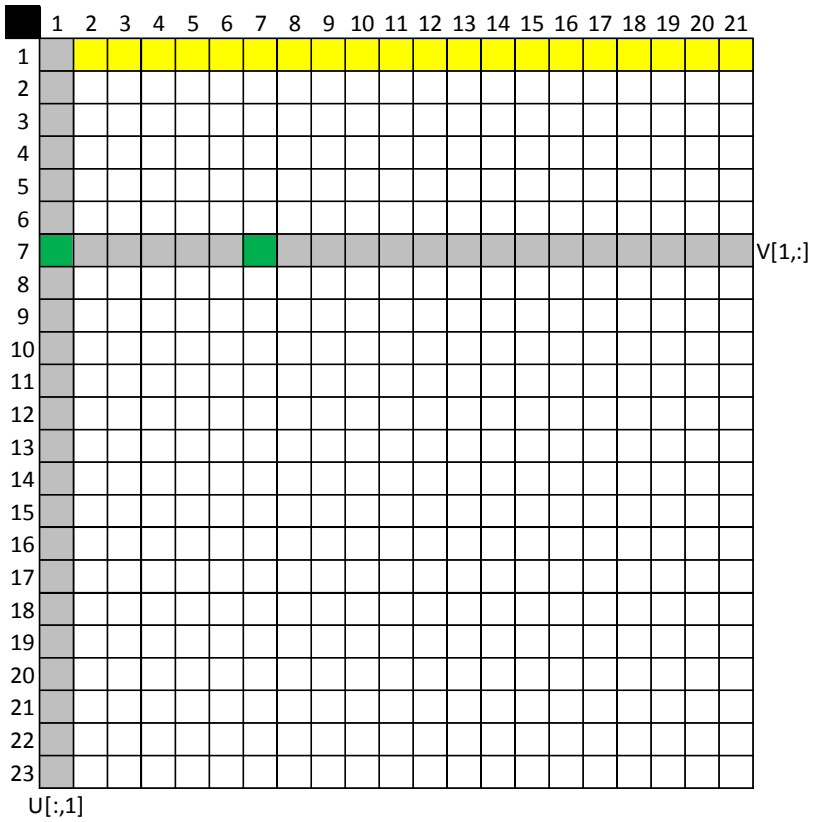

Figura 44 - Linha e coluna calculadas para definir $V[1,:]$ e $U[:,1]$ e pivôs (verde).

Tabela 17 - Variáveis importantes calculadas até o resultado obtido na Figura 44.

*indaref = 1 *indbref = 1 *arefmax = 21 *brefmax = 7 Vmax = 1 Umax = 7 V[Vmax] = 0.0143 *Valores não foram recalculados	INDB1 = [1 7] INDB2 = [1 1]	ind1 = [true true true true true false . . . true]	ind2 = [false true true . . . true true true]

Como o novo pivô $Vmax$ (Tabela 17) coincide com o pivô da coluna auxiliar $indbref$ (R9, l.39) será necessário calcular uma nova coluna auxiliar (R9, ll.41-52).

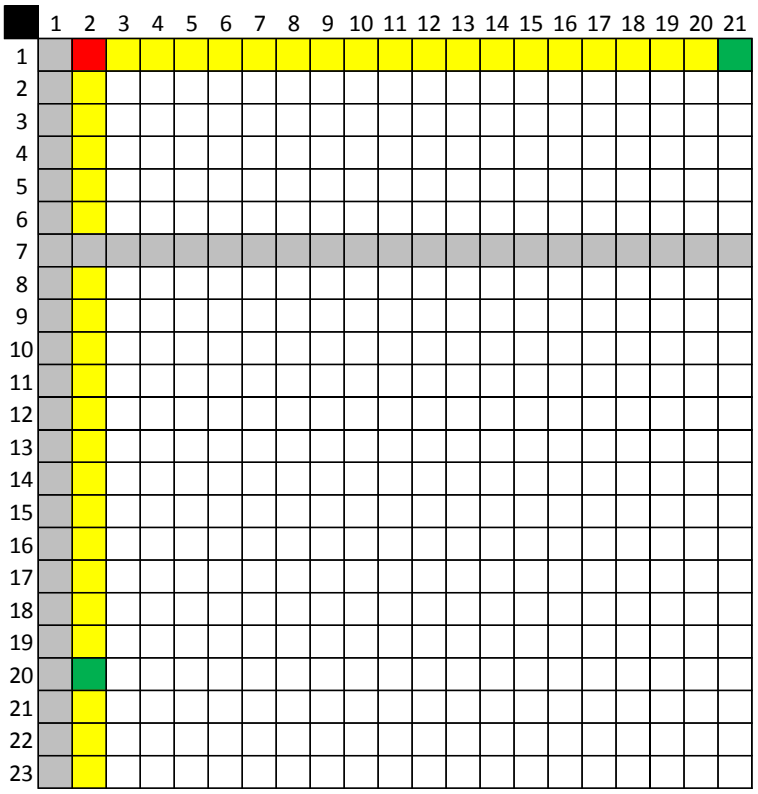

Figura 45 - Nova coluna calculda para definir a coluna auxiliar,), com novos subpivô (em vermelho) e pivôs (verde) em destaque.

Assim como na obtenção das matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$, a linha e coluna de referência são atualizados a cada iteração, subtraindo essas pela sua respectiva na matriz aproximada $\boldsymbol{Ap}$. São atualizados também os seu pivôs (R8, ll.45-46) (Tabela 18).

Tabela 18 - Variáveis importantes calculadas até o resultado obtido na Figura 45.

*indaref = 1 indbref = 2 arefmax = 21 brefmax = 20 *Vmax = 1 *Umax = 7 aref[arefmax] = 0.0112 bref[brefmax] = -0.0018	INDB1 = [1 7] INDB2 = [1 1 2]	ind1 = [true true true . . . true true true]	ind2 = [true true true . . . true true true]

O estado da Figura 45 é o mesmo que o da Figura 43. A partir desse ponto, o processo é repetido até que o erro na aproximação seja menor que o admissível (R8, ll.37-38) ou todas as linhas tenham sido calculadas (R8, l.11). Quando isso ocorre, o algoritmo segue para a próxima linha da matriz $blocks$, analisando os próximos pares de blocos, até que todos tenham sido analisados.

Para os blocos 2 e 3 em análise, o resultado final obtido é dado pela Tabela 19. Os vetores $Ind1$ e $Ind2$ indicam, respectivamente, os índices das linhas e colunas calculadas para determinar as matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$ (não apresenta as utilizadas como referência)**.** As vetores $INDB1$ e $INDB2$ mostram, respectivamente, os índices das linhas e colunas calculadas. Essas variáveis confirmam o observado em $Ind1$ e $Ind2$ e também possibilitam descobrir as linhas e colunas auxiliares utilizadas. Os primeiros valores desses vetores, indicam o índices da primeira linha e coluna auxiliares utilizadas. Se uma linha ou coluna calculada coincidir com uma de referência, ocorre a repetição desse índice nesses vetores. Quando isso ocorre, o algoritmo calcula uma nova linha ou coluna. Assim, o próximo índice após a repetição indica o índice da nova linha ou coluna auxiliar calculada.

Observe que primeiramente foi calculada a linha 1 e a coluna 1, ou seja, a linha e coluna auxiliar (Figura 43) Em seguida é calculada a linha 7 e a coluna 1 novamente, o que indica que uma nova coluna auxiliar deve ser calculada. É calculada então a coluna 2, a nova coluna auxiliar. (Figura 45). Durante o processo para essa matriz, isso ocorre somente mais uma vez, quando a linha 1 é novamente calculada. Assim, é preciso calcular uma nova linha auxiliar, é calculado então a linha 2. Os valores de $indaref$ e $indbref$ da Tabela 19 confirmam os índices da última linha e coluna auxiliar utilizadas.

Tabela 19 - Resultados finais obtidos no cálculo da submatriz formada pelos blocos 2 e 3

indaref = 2 indbref = 2 arefmax = 9 brefmax = 19 Vmax = 13 Umax = 19	INDB1 = [1 7 6 17 12 1 2 21 4 15 9 19]	INDB2 = [1 1 2 21 6 16 17 5 11 20 3 13]	ind1 = [false true true false true false false true false true true false true false false true true true false true true]	ind2 = [false true false true false false true true true true false true true true false true false true true false false]

. Os vetores $Ind1$ e $Ind2$ possibilitam também visualizar as linhas e colunas calculadas da submatriz (Figura 46). Foram calculados 65% dos elementos da matriz. No entanto, a matriz aproximada é salva como um produto das matrizes $\boldsymbol{U}$ e $\boldsymbol{V}$, necessitando que mais elementos sejam guardados, pois para cada elemento em comum entre uma linha e uma coluna calculada da submatriz é preciso guardar dois elementos, um para $\boldsymbol{U}$ e outro para $\boldsymbol{V}$. Assim, em comparação com toda a submatriz, é necessário calcular 82% da quantidade de elementos para representar a matriz.

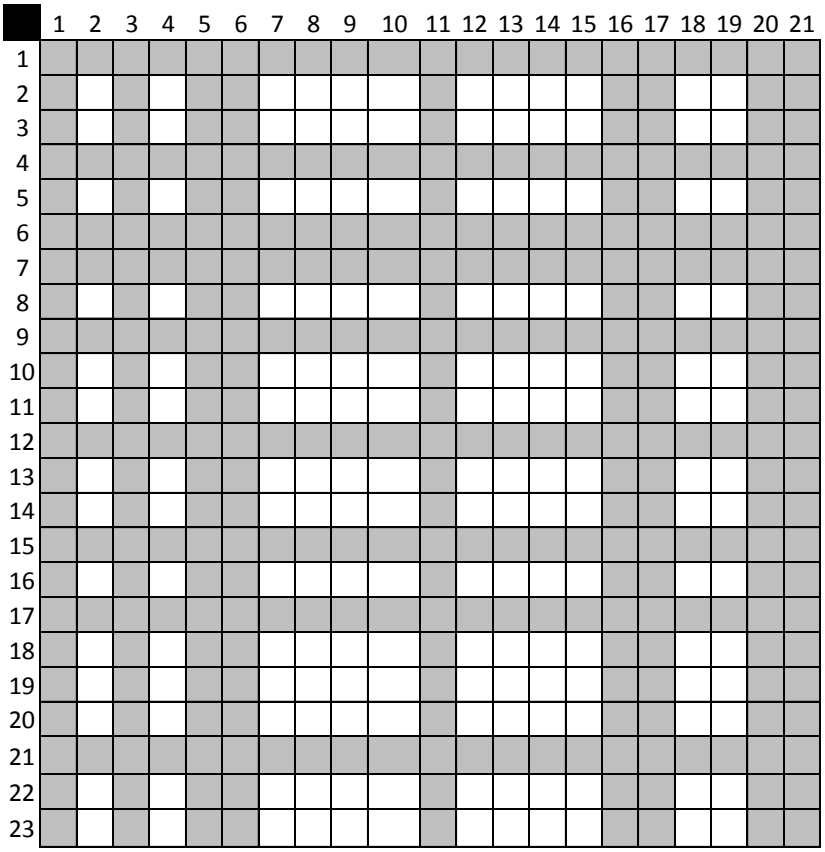

Figura 46 - Elementos calculadas da submatriz (cinza)

No entanto, considerando toda a matriz do problema é necessário calcular 102% do número de elementos para implementar o método ACA+. Isso ocorre, pois embora haja um ganho na representação das submatrizes maiores, existe uma perda na representação das menores que requerem o cálculo de uma quantidade maior de linhas e colunas para apresentar a mesma precisão na aproximação.

O método ACA não é indicado para problemas de pequenas escala, devido a perda na representação de submatrizes pequenas. Esse método requer naturalmente um maior processamento de dados do que a resolução direta do problema e se apoia no ganho existente na representação aproximada de grandes blocos. Assim, se a quantidade desses blocos é escassa, o método se torna ineficiente.

7 ALOCAÇÃO DE MEMÓRIA E TEMPO

Para avaliar o desempenho do algoritmo implementado para a execução do método ACA, foi utilizado diferentes versões do problema dado pela Figura 47. O mesmo problema e resolvido diversas vezes, variando a quantidade de furos e o número de elementos que discretiza seu contorno. É avaliado o tempo de execução e a memória necessária para armazenar a matriz $\boldsymbol{A}$ do problema $\boldsymbol{A}x = b$. Os resultados são comparando com aqueles obtidos pela resolução direta (tradicional) do problema.

Foi utilizada a versão 0.6.0.1 da linguagem Julia em um computador com as mesmas especificações dadas anteriormente (seção 5.1.3).

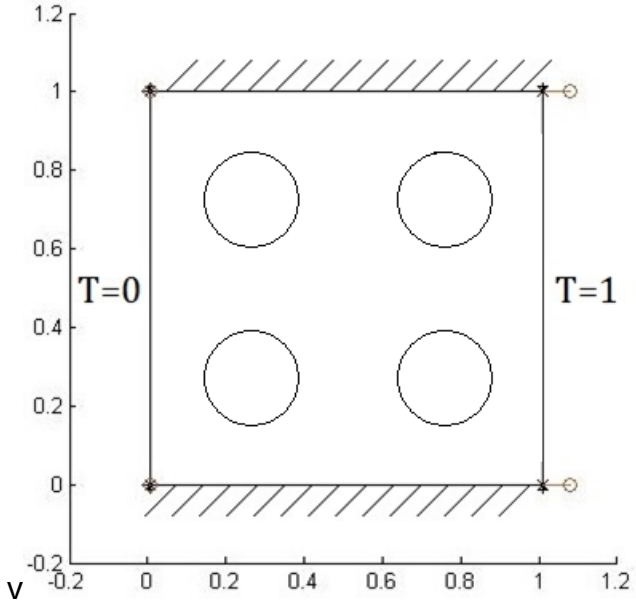

Figura 47 - Problema utilizado para avaliar alocação de memória e tempo.

A geometria do contorno de cada furo é dividido em quatro segmentos, e cada segmento é representado por quatro elementos de contorno. A quantidade de furos e dada por nf^2 e a quantidade de elementos que discretiza cada aresta do quadrado e dada por nf. Assim, o número de nós na malha, o qual é igual ao número de elementos de contorno, é dada por $16nf^2 + 4nf$, contabilizando a contribuição dos furos e das arestas, respectivamente. Os resultados obtidos para valores de nf de 1 a 20 podem ser observados nas Tabelas 20 e 21.

Tabela 20 - Tempo e memória observados na resolução direta.

nf	Quantidade de nós	Tempo de execução [s]	Tamanho da matriz [KB]
1	20	0.09	3.2
2	72	0.09	40
3	156	0.12	190
4	272	0.17	578
5	420	0.28	1378
6	600	0.56	2812
7	812	0.90	5151
8	1056	1.50	8712
9	1332	2.15	13861
10	1640	3.41	21012
11	1980	4.96	30628
12	2352	6.88	43218
13	2756	9.76	59340
14	3192	13.69	79600
15	3660	18.12	104653
16	4160	23.05	135200
17	4692	29.59	171991
18	5256	38.31	215824
19	5852	46.92	267546
20	6480	58.24	328050

Tabela 21 - Tempo e memória observados na resolução pelo método ACA

nf	Quantidade de nós	Tempo de execução [s]	Tamanho da matriz aproximada [KB]	Elementos calculados [%]
1	20	2.29	3,7	107.5
2	72	2.35	36	88.19
3	156	2.45	163	85.51
4	272	2.69	382	65.76
5	420	3.01	800	57.90
6	600	3.94	1598	56.57
7	812	4.20	2143	41.52
8	1056	5.25	3630	41.60
9	1332	7.17	5464	39.39
10	1640	8.98	7397	35.18
11	1980	11.34	12126	39.55
12	2352	16.81	13136	30.37
13	2756	24.19	15341	25.83
14	3192	26.29	18241	22.90
15	3660	42.23	25451	24.31
16	4160	42.08	38368	28.36
17	4692	71.04	39313	22.85
18	5256	55.01	64178	29.73
19	5852	112.21	56654	21.17
20	6480	148.36	68945	21.01

As informações contidas nas Tabelas 20 e 21, estão contidas também nos gráficos das Figuras 48 a 52, os quais facilitam a análise dos resultados.

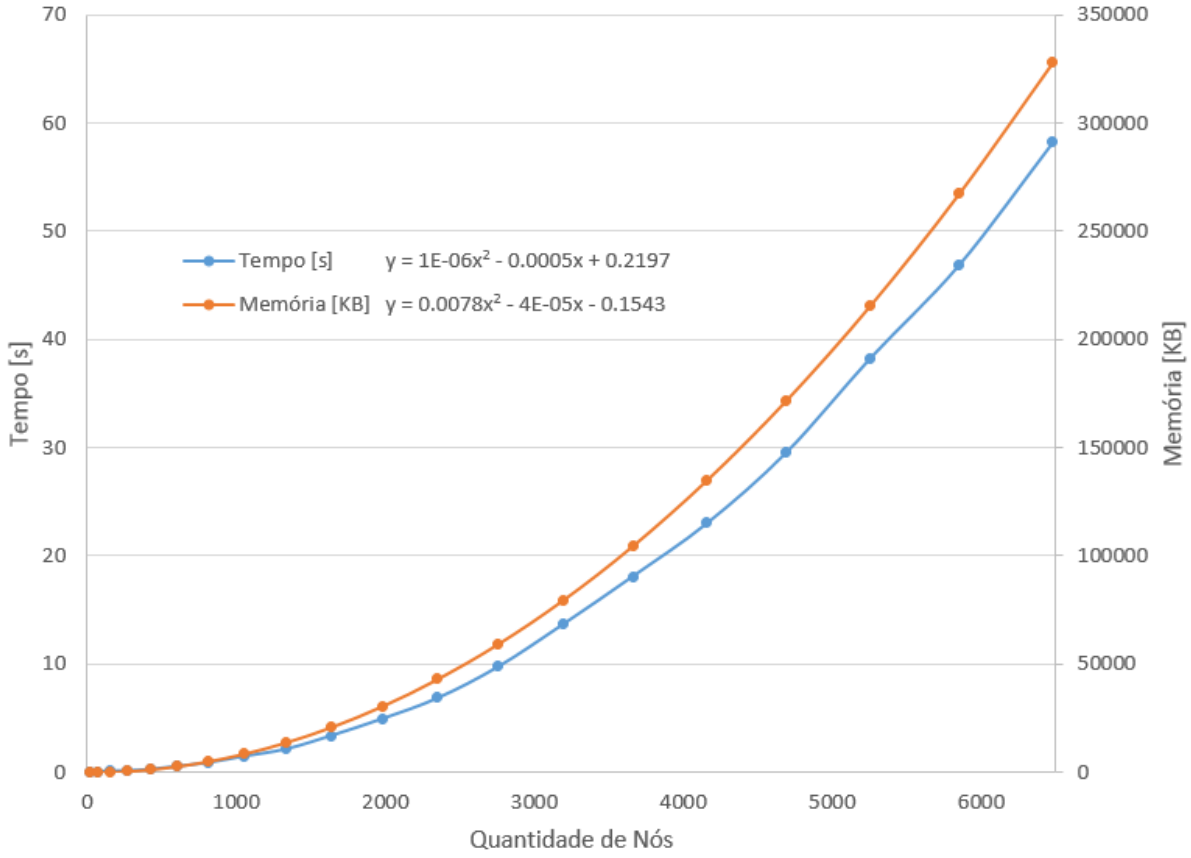

Figura 48 - Tempo e memória observados na resolução direta.

É notavel, da Figura 48, que tanto o crescimento do tempo quanto a utilização da memória na resolução direta do problema apresentam comportamento quadratico. Esse comportamento já era esperado pois cada elemento da matriz requer uma mesma quantidade de memória para ser armazenado e a matriz apresenta n^2 elementos, onde n é o número de nós. Já o tempo de resoluação do problema pelo metodo GMRES é na grande maioria dos casos proporcional ao tamanho da matriz, exceto quando é necessário uma grande quantidade de iterações.

No caso do método ACA (Figura 49), embora a curva inicialmente apresente comportamento quadrático, ela muda rapidamente para as malhas maiores, sendo afetadas fortemente por um comportamento oscilatório. Esse comportamento tem origem principalmente em dois fatores, na condição de admissibilidade e no tamanho mínimo admissível para as folhas da árvore binária. Esses dois fatores juntos controlam a quantidade e tamanho dos blocos a serem aproximados, o que afeta diretamente o tempo de execução do algoritmo.

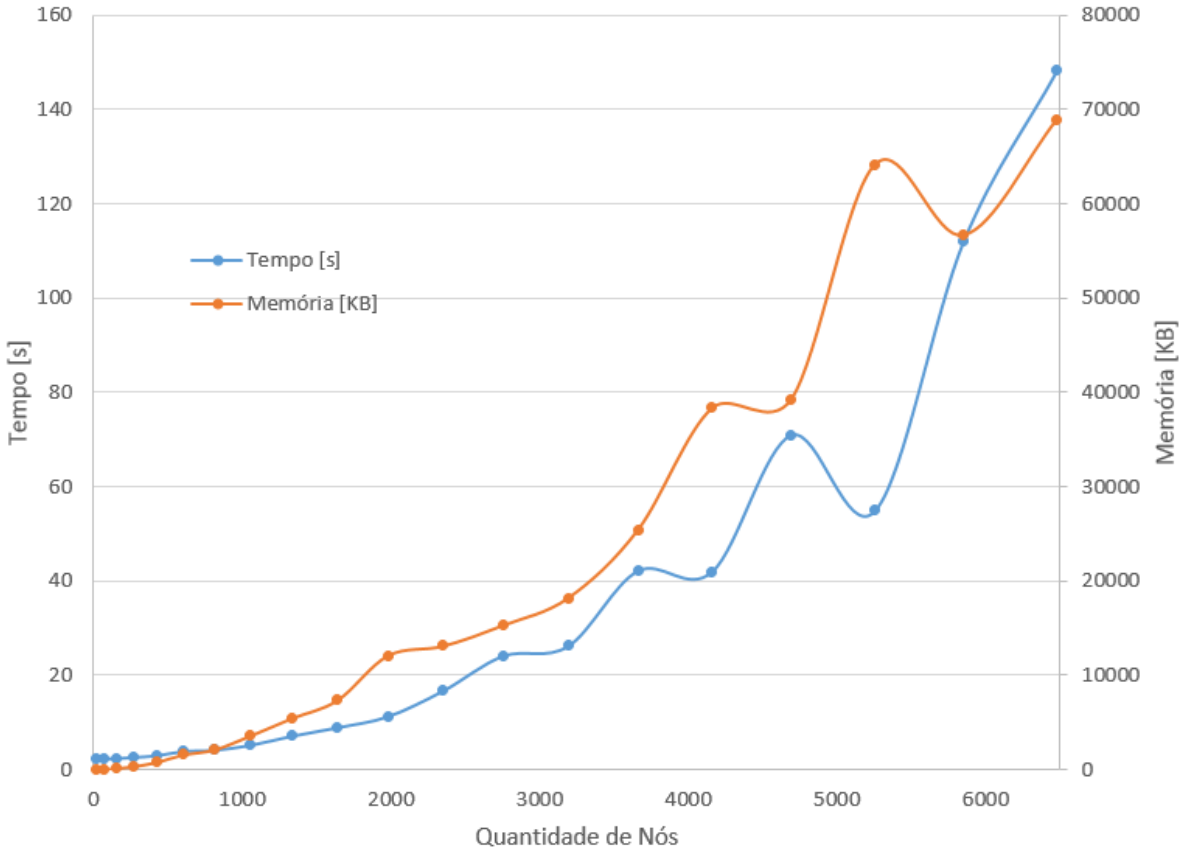

Figura 49 - Tempo e memória observados na resolução pelo método ACA

A comparação para os tempos de execução (Figura 50), não é favorável ao método ACA. O método apresenta tempo sempre maior que o observado na resolução direta, diferença decorrente principalmente das sub-rotinas necessárias para a aplicação do método. No entanto, os resultados obtidos na Figura 50 decorrem da aplicação do método com o cálculo analíticos dos elementos das matrizes $\boldsymbol{H}$ e $\boldsymbol{G}$. Quando esse cálculo é feito numericamente (método necessário para elementos de contorno de maior ordem) é possivel que essa fração de tempo deixe de ser dominante durante a execução da rotina, compensando sua aplicação devido ao cálculo de uma menor quantidade de elementos e a grande economia de memória.

A rotina aplicada para o método ACA nesse trabalho tem função educativa e deve ainda ser otimizada. É possível, por exemplo, aplicar a análise PCA de forma numérica evitando a necessidade do cálculo de autovetores e autovalores. É preciso também implementar a condição que define os blocos de baixo posto para evitar que matrizes sejam representadas na forma aproximada, quando sua representação exata necessitaria de uma menor quantidade de elementos. Isso tornaria resultados como o da quarta coluna da primeira linha da Tabela 21 (107.5%) impossíveis de ocorrer.

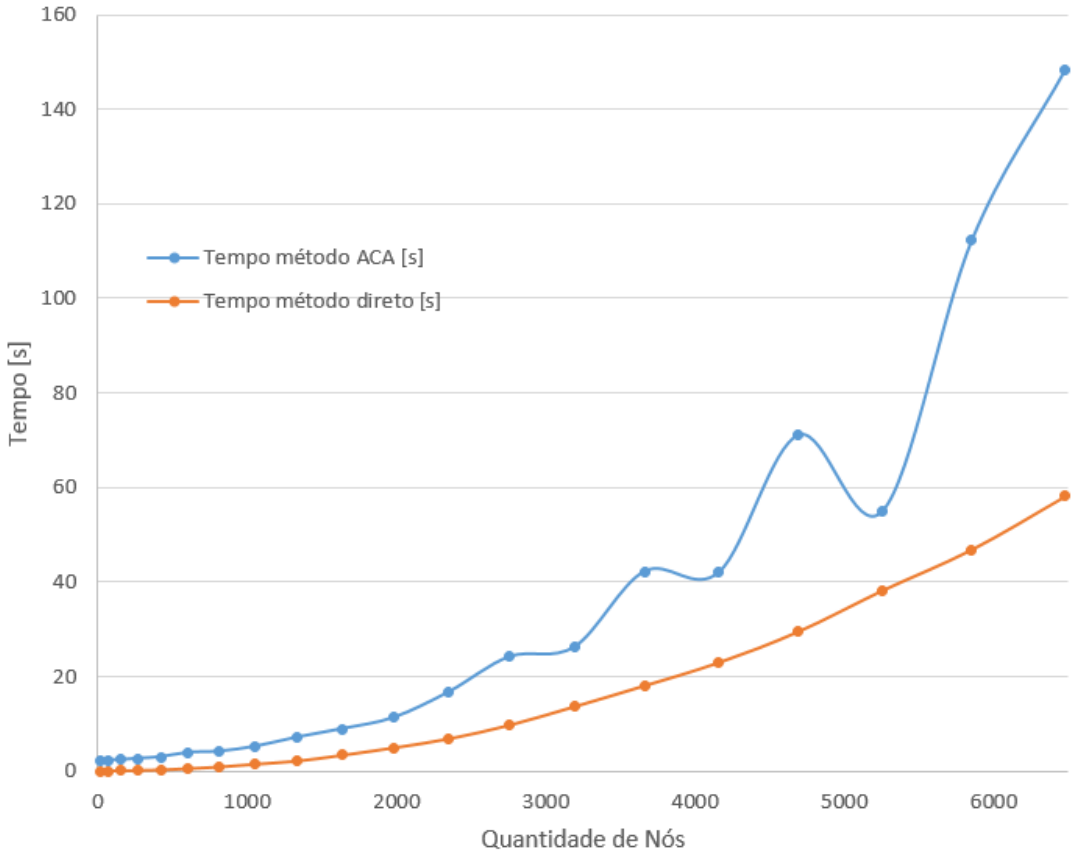

Figura 50 - Comparação dos tempos observados.

Embora o resultado obtido para o tempo de execução não seja favorável, o mesmo não ocorre para o uso da memória, quesito no qual a superioridade do método ACA fica clara. Enquanto a utilização da memória para a resolução direta apresenta comportamento quadrático, a resolução pelo Método ACA apresenta comportamento quase linear (Figura 51). No entanto, é simples observar que esta diferença não justifica a aplicação do método para problemas de pequenas dimensões, onde a diferença não é notável.

Foi avaliado também a quantidade de elementos calculadas para representar a matriz de forma aproximada, em relação a quantidade necessária para a matriz exata (Figura 52). O Valor apresentado para a primeira malha, com 20 nós, é de 107,5%, o que evidência a ineficiência de se aproximar matrizes pequenas. A partir desse ponto, a curva apresenta decaimento exponencial tendendo ao valor de 18,5%, de acordo com a regressão, quando o número de nós vai a infinito. O menor valor observado foi de 16,42% para a malha com 10100 nós ($nf = 25$). Tal valor não é apresentado nas Tabelas 20 e 21, mas foi calculado afim de possibiliar a visualização do comportamento assintota da curva.

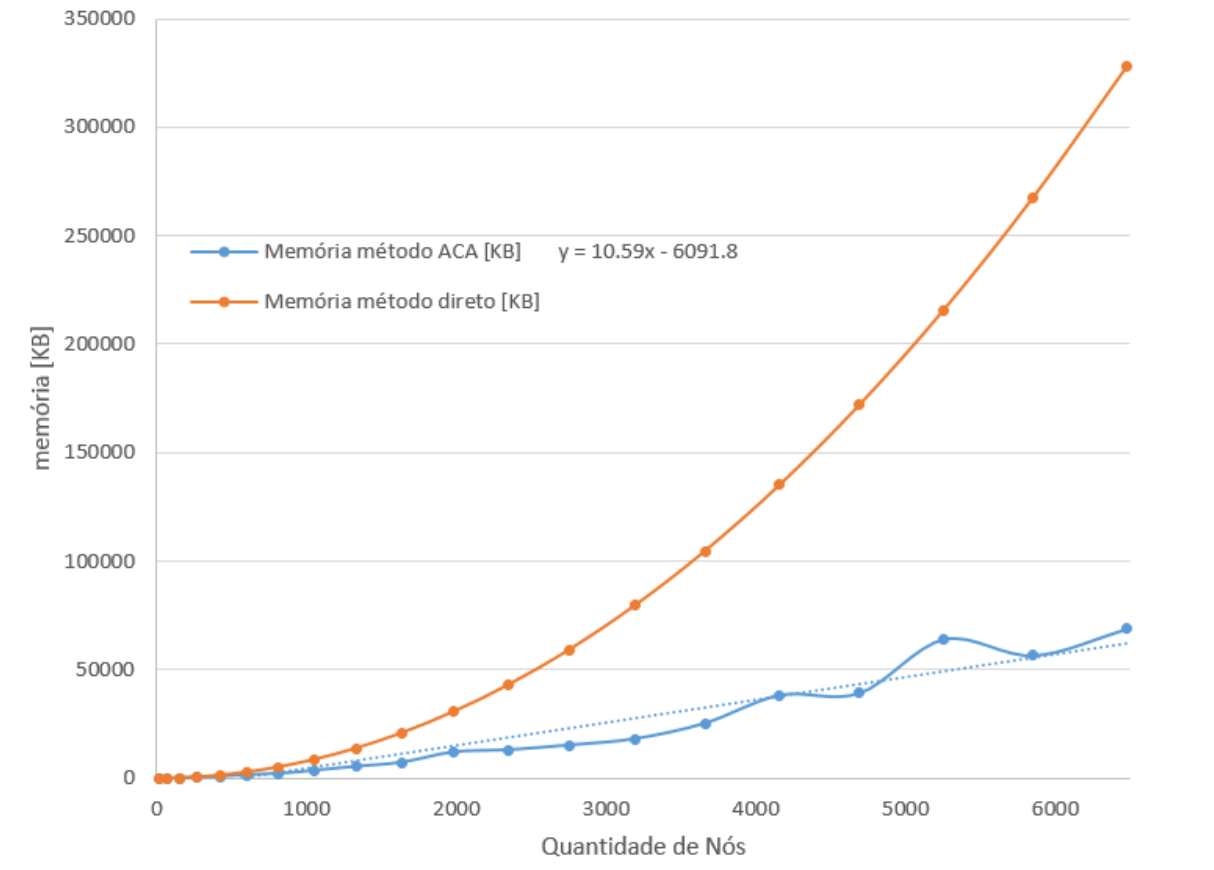

Figura 51 - Comparação no uso de memória observados.

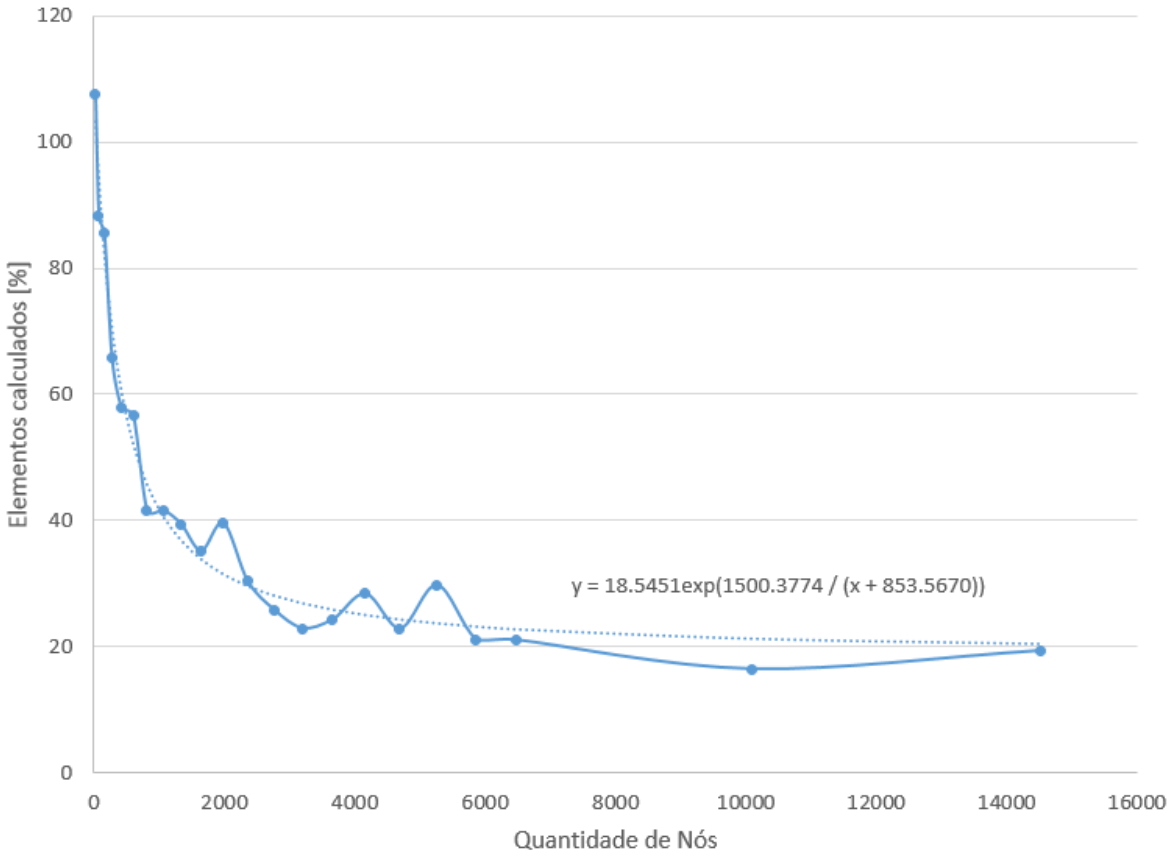

Figura 52 - Quantidade relativa de elementos calculados, quando comparados com a matriz cheia $\boldsymbol{A}$.

Para o problema resolvido, podemos observar da Figura 52 que a aplicação do método ACA permite uma redução de mais de 80% do número de elementos calculados para os problemas maiores o que tem um impacto visível na utilização da memória. No entanto, em relação a tempo de execução o método não foi tão vantajoso. Mas essa condição pode ainda ser melhorada otimizando o algoritmo ou possivelmente revertida quando o problema é de grandes dimensões e a integração dos elementos das matrizes $\boldsymbol{H}$ e $\boldsymbol{G}$ é feita de maneira numérica, pois nesse caso e possível que o tempo dominante durante a execução seja o tempo necessário para o cálculo desses elementos, os quais notavelmente se apresentam em menor quantidade quando utilizado o método ACA.

8 CONCLUSÃO

O método dos elementos de contorno resulta em matrizes cheias e não simétricas. O processo de obtenção, montagem e resolução desse tipo de matriz requer um grande tempo de execução e utilização da memória. A fim de reduzir esses fatores, são utilizados algoritmos rápidos, de complexidade reduzida, que permitem realizar as mesmas operações em tempo menor e com menor necessidade de recursos computacionais.

O primeiro passo é dividir os pontos analisados, no caso, os nós da malha, em agrupamentos menores. Isso é feito utilizando a análise de componentes principais, método que fornece um plano de separação dos dados, gerando agrupamentos menores da maneira mais uniforme possível, com aproximadamente a mesma quantidade de pontos. É preciso guardar a relação entre os blocos, o que é feito utilizando uma árvore binária. O segundo passo é agrupar esses blocos, organizados pela árvore, em blocos maiores. Isso é feito utilizando uma clusterização hierárquica, através de uma condição de admissibilidade para agrupar blocos que apresentem comportamento similar, de forma a obter o maior número de blocos que possam ser aproximados por uma matriz de baixo posto. Uma vez agrupado os dados, é preciso calcular as matrizes que representam cada bloco. O terceiro passo, é calcular a aproximação de baixo posto daquela matriz, o que é efeito utilizando a aproximação cruzada adaptativa. Esses três passos reduzem o tempo de execução e a memória utilizada para calcular e armazenar a matriz, pois não é preciso calcular todas as entradas da matriz. Além disso, os blocos de baixo posto requerem uma menor quantidade de dados armazenados. Esses três passos resultam na representação da matriz por matrizes hierárquicas.

Calculada as matrizes hierárquicas, o último passo é resolver a equação matricial, o que é feito pelos métodos iterativos, no caso o método GMRES, pois esse mostrou ser ideal para as matrizes obtidas nas aplicações do MEC. A representação de uma matriz por matrizes hierárquicas requer uma álgebra adaptada para esse tipo de matriz.

O método ACA se mostra essencial na resolução de problemas de larga escala, possibilitando uma grande redução na memória utilizada e embora resultados

favoráveis em relação ao tempo de execução não tenham ainda sido observados, esses são, sem dúvidas, possíveis.

Essas técnicas reunidas realizam o processo de obtenção, montagem e resolução das matrizes mais rapidamente e com menor utilização da memória, tornando possível resolver problemas de maior complexidade com uma menor utilização de recursos computacionais. Assim, esse conjunto de métodos que compõem os algoritmos rápidos é fundamental para a resolução de problemas de larga escala.

8.1 TRABALHO FUTURO

Embora o título desse trabalho seja matrizes hierárquicas e aproximação cruzada adaptativa, ele reflete pouco todo o conteúdo aqui apresentado. Aqui foi citado todo um conjunto de métodos que possibilitam a implementação de um algoritmo rápido, assim como a teoria básica do método de contorno para o caso de problemas de condução de calor bidimensional. Este trabalho foi feito com o intuito de providenciar uma visão geral e uma base para trabalhos futuros sobre algoritmos rápidos e aproximação cruzada adaptativa.

Os próximos trabalhos do grupo de mecânica dos sólidos da UnB devem ser focados na implementação de um algoritmo otimizado do método ACA, possibilitando um ganho de tempo considerável na execução do mesmo. Esses trabalhos devem também aplicar o método em elementos de contorno de maior ordem, permitindo observar o efeito positivo que o método apresenta também sobre o tempo de execução. Outro ponto importante que deve ser considerado em trabalhos futuros é a influência que certos fatores pré-definidos, como a condição de admissibilidade e número mínimo de elementos nas folhas, têm sobre a eficiência e acurácia do método.

9 REFERÊNCIAS

[1] BELYTSCHKO, T. et al. Element Free Galerkin Methods. International journal for numerical methods in engineering, John Wiley & Sons, Ltd. v. 37, n. 2, p. 229-256, 1994.

[2] WROBEL, L.; ALIABADI, M. The Boundary Element Methods in Engineering. McGraw-Hill College, 2002.

[3] KANE, J. H. Boundary Element Analysis in Engineering Continuum Mechanics. Englewood Cliffs, NJ: Prentice Hall, 1994.

[4] BECKER, A. A. The boundary Element Method in Engineering: A Complete Course. McGraw-Hill Companies, 1992.

[5] KURZ, S.; RAIN, O.; RJASANOW, S. Fast Boundary Element Methods in Computational Electromagnetism. Springer, 2007. p. 249-279.

[6] JIN, J. M. Theory and Computation of Electromagnetic Fields. 2th ed. Section 11.1 - Introduction to fast Algorithms. John Wiley & Sons, Inc. 2015

[7] HACKBUSCH, W.; NOWAK, Z. P. On the Fast Matrix Multiplication in the Boundary Element Method by Panel Clustering. Numerische Mathematik. Springer, v. 54, n. 4, p. 463-491, 1989.

[8] GOREINOV, S. A.; TYRTYSHNIKOV, E. E.; ZAMARASHKIN, N. L. A Theory of Pseudoskeleton Approximations. Linear Algebra and its Applications. Elsevier. v. 261, n. 1, p. 1-21, 1997.

[9] BEBENDORF, M.; GRZHIBOVSKIS, R. Accelerating Galerkin BEM for Linear Elasticity Using Adaptive Cross Approximation. Mathematical Methods in the Applied Sciences. v. 29, p. 1721-1747, 2006.

[10] BEBENDORF, M.; RJASANOW, S. Adaptive Low-rank Approximation of Collocation Matrices. Computing, v. 70, p. 1-24, 2003.

[11] BEBENDORF, M. Approximation of boundary element matrices. Numerishce Mathematik, v. 86, p. 565-589, 2000.

[12] GRASEDYCK, L. Adaptive Recompression of Matrices for BEM. Computing, Springer, v. 74, n. 3, p. 205-223, 2005.

[13] PETERSEN, K. B.; PETERSEN, M. S. The Matrix CookBook, v. of November 15, 2012, p. 11, equation (83).

[14] YANG, F. Construction and Application of Hierarchical Matrix Preconditioners. Section 2.3 - Concept of H-Matrices. University of Iowa. 2008.

[15] BORM, S. Construction of General Hierarchical Matrices for Multi-Dimensional Problems. University of Kiel, Germany.

[16] KATSIKADELIS, J.T. Boundary element: Theory and Application. Chapters 2-3. Elsevier Science Ltd. 2002

[17] ALBURQUERQUE, E. L. Introdução ao Método dos Elementos de Contorno. Universidade de Brasília.

[18] GUENTHER, R. B.; J. W. LEE. Partial Differential Equations of Mathematical Physics and Integral Equations. Section 10.5. New York, NY: Dover Publications, 1996.

[19] MYINT-U, T.; LOKENATH D. Linear Partial Differential Equations for Scientists and Engineers. 4th ed. Section 11. Boston, MA: Birkhauser, 2006.

[20] HUNT, R. E. Lecture Notes. Chapter 2 - Poisson's Equation. p. 29-30. 2002.

[21] TYRTYSHNIKOV, E. Mosaic-skeleton Approximations. Calcolo, Springer, v. 33, n. 1-2, p. 47-57, 1996.

[22] BREBBIA, C. A. Boundary Element Methods in Engineering. Springer New York, NY, USA: 1982.

[23] AZEVEDO, A. F. M. Método dos Elementos Finitos. Faculdade de Engenharia da Universidade de Porto. 1th ed. Capitulo 7 - Funções Interpoladoras. Abril 2003

[24] GREENBAUM, A. Iterative Methods for Solving Linear Systems. Siam, 1997.v. 17.

[25] STRANG, G. Computational Science and Engineering. Chapter 7 - Solving Large Systems. Wellesley-Cambridge Press. 2007

[26] NOSCHESE, S.; PASQUINI, L.; LOTHAR, R. Tridiagonal Toeplitz Matrices: Properties and Novel Applications. Section 1 - Introduction. John Willey & Sons, Ltd. 2006

[27] KAHAN, W. Gauss-Seidel Methods of Solving Large Systems of Linear Equations. Ph.D. thesis. Toronto, Canada. University of Toronto, 1958.

[28] GREENBAUM, A. Iterative Methods for Solving Linear Systems. Frontiers in Applied Mathematics, v. 17. Theorem 10.1.3. SIAM, 1997.

[29] DEMMEL, J. W. Applied Numerical Linear Algebra. Theorem 6.7. SIAM, 1997.

[30] Briggs, W.L. A Multigrid Tutorial. Center of Applied Scientific Computing. University of California

[31] STRANG, G. Linear Algebra and Its Applications. 4th ed. Chapter 3 - Orthogonality. Thomson Learning, Inc. 2006

[32] STAHEL. A. Numerical Methods. Section 2.7 - Iterative Methods For non-symetric Systems. 2007

[33] SHEWCHUK, J. R. An Introduction to the Conjugate Gradient Method without the Agonizing Pain. 1st Edition. Carnegie Mellon University. 1994

[34] SHEN, J.; TANG, T.; WANG, L.L. Spectral Methods: Algorithms, Analysis and Applications. Spring Series in Computational Mathematics 41. Appendix C - Basic Iterative Methods and Preconditioning. 2011

[35] IZADI, M. Hierarchical Matrix Techniques on Massively Parallel Computers. Section 3.1.1.2. Leipzig University. 2012

[36] SAAD, Y.; SCHULTZ, M. H. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetrical Linear Systems. Journal on scientific and statistical computing, SIAM, v. 7, n. 3, p. 856-869, 1986.

[37] JOUBERT, W. On the Convergence Behavior of the Restarted GMRES algorithm for Solving Nonsymmetrical Linear Systems. Numerical linear algebra with applications, Wiley Online Library, v. 1, n. 5, p. 427-447, 1994.

[38] VAN DER VORST, H. A.; VUIK, C. The Super-linear Convergence Behavior of GMRES. Journal of computational and applied mathematics. Elsevier, v. 48, n. 3, p. 327-341, 1993.

[39] EIERMANN, M.; ERNST, O.; SCHNEIDER, O. Analysis of acceleration strategies for restarted minimal residual methods. Journal of computational and applied Mathematics, Elsevier, v. 123, n. 1, p. 261-292, 2000.

[40] EMBREE, M. The Tortoise and the Hare Restart of GMRES. SIAM, v. 45, n. 2, p. 259-266, 2003.

[41] BAKER, A. H.; JESSUP, E. R.; KOLEV, T. V. A Simple Strategy for Varying the Restart Parameter in GMRES(m). Journal of computational and applied mathematics. Elsevier, v. 230, n. 2, p. 751-761, 2009.

[42] Et al. Jeff Bezanson. The Julia Language. https://Julialang.org. Acessado em: 17/10/2017

10 ANEXOS

```
function [x,k,e] = IterativoPuro(A,b)
% Utiliza os diferentes métodos estacionários para resolver o problema
% Ax=b, para uma matriz A qualquer. O programa retorna o vetor x o numero %
% de iterações e o erro. É possível observar o comportamento do erro
% desabilitando "e" e habilitando "e(K)".

clc
close all

% Checa dimensões A(n,m)x = b(n)=========================================

[n,~] = size(A);              % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))    % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A');
    end

% Erro admissível========================================================

erro_adm = 10^-6;              % Em porcentagem de |b|.
kmax = 1000;                   % # máximo de iterações

% Pré-condicionador======================================================

D = diag(diag(A));          % Diagonal de A;
L = tril(A)-D;              % Parte estritamente triangular inferior de A;
w = 0.06;                   % Coeficiente w do método ponderado de Jacobi
wsor = 1.36;                % Coeficiente w do método SOR

P = D;                      % Pré-Condicionador de Jacobi
%P = (1/w)*D;               % Pré-Condicionador de Jacobi Ponderado
%P = L+D;                   % Pré-Condicionador de Gauss
%P = D+wsor*L;              % Pré-Condicionador SOR

% Método Iterativo=======================================================

I = speye(n);                   % Matriz identidade
M = I-P\A;                      % obs.: P\A = inv(P)*A enquanto P/A=inv(A)*P
c = P\b;
k = 1;                % Para a contagem de iterações k (# iterações = k-1)
x = zeros(n,1);                 % x inicial
r0 = b-A*x;                     % Resíduo inicial
%e(k) = norm(r0)/norm(b);       % Erro
 e = norm(r0)/norm(b);
%while e(k) > erro_adm          % Enquanto o erro for maior do que o
% admissível continuar o processo iterativo
     while e > erro_adm
        x = M*x+c;
%         e(k+1) = norm(A*x-r0,'fro')/norm(b); % Calcula erro da k-ésima
% iteração
         e = norm(A*x-r0,'fro')/norm(b);
        k = k+1;                                % Atualiza o contador de
% iterações
            if k == kmax
                display('Não convergiu no numero máximo de iterações
permitido');
                break          % Se não convergir no número máximo permitido
% de iterações, interromper processo.
```

```
            end
%             if e(k) > 10^2
            if e > 10^2
                display('O método diverge');
                break         % Se divergir, interromper processo.
            end
    end
    k = k-1;                  % # de iterações, pois x0 = x(1)
%     hold on
%     plot(e,'-o');           % Plota o comportamento do erro
end
```

```
function [x,e] = Multigrid(A,b)

% Utiliza o método Multigrid juntamente com o método iterativo de
% Gauss-Seidel para a resolução do problema Ax = b, no caso da discretizão
% do contorno de um quadrilátero por 2^(2+level) elementos de mesmo
% comprimento h. O programa retorna a vetor x e o erro.

clc
close all

% Parâmetros=========================================================

kpre = 5;               % # de iterações por método iterativo antes do
Multigrid
kpos = kpre;            % # de iterações por método iterativo depois do
Multigrid
no = length(b);         % Tamanho original da matriz b = # Número de nós
gamma = 2;              % Tipo de ciclo 1-V 2-W

% Checa dimensões A(n,m)x = b(n)=====================================

[n,~] = size(A);              % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))    % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A')
    end

% Checa número de elementos==========================================

global level t
level = log2(no)-2;
    if rem(level,1)~=0 % Se não atender a condição 2^(2+level) elementos
% interromper processo
        error('O número de elementos dever ser = 2^(2+level), onde level é
um número inteiro.');
    elseif level==0    % Se level é 0 não há como reduzir o problema, então
% e resolvido sem redução
        display(' ');
        display('Não há redução da malha. Problema resolvido na malha
original.');
    end

t = 0;                            % Contador do número de espações e
% reduções da malha
%hold on
[x] = grid(A,b,kpre,kpos,gamma); % Chama a função para executar o multigrid
%hold off
e = norm(A*x-b)/norm(b);          % Erro do processo

%====================================================================
end

function [x] = grid(A,b,kpre,kpos,gamma,x0)
global level t

n = length(b);          % Tamanho da matriz b = # Número de nós
```

```
    if nargin<6        % Se o número de entradas na função for menor que 6,
% ou seja, quando não há x0 (1st iteração).
        x = b*0;       % Define como x0 um vetor de zeros do tamanho de b.
    else
        x = x0;        % x inicial. Obs.: Não definimos um x inicial, mas
% ele é preciso na função devido a recursividade.
    end

% Resolvendo Ax=b de forma exata=============================================

ele = n/4;               % # de elementos em cada segmento do contorno.
    if ele==1
        x=A\b;           % Se a matriz é pequena resolver ela de forma
% exata
    else

% Método de Guass-Seidel=====================================================

        P = tril(A);           % Condicionador do Método de Gauss-Seidel
        I = speye(n);          % Matrix Identidade
        M = I-P\A;
        c = P\b;
            for i=1:kpre
                x = M*x+c;     % Gauss-Seidel antes do Multigrid
            end
        r = b-A*x;             % Resíduo após iterações iniciais

% Matrizes T e R da Malha reduzida===========================================

        T = (1/2)*(spdiags(fliplr(spdiags(ones((n/2),1)*[1 2 1],-1:1,...
(n/2),(n/2)+1)),-(n/2):0,n,(n/2))+sparse(n,1,1,n,(n/2)));
        % Cria a matriz de interpolação T
        R = (1/2)*T';          % Cria matriz de restrição R

% Multigrid==================================================================

        rh = R*r;                                          % Restrição do
% resíduo para a malha grosseira
        t = t+1; level = level-1;
        %plot([t-1 t],[level+1 level],'bo-')              % Somente para
% controle e visualização do ciclo
        Eh = rh*0;                                    % Erro inicial na malha
grosseira
            for i = 1:gamma                            % Define tipo de ciclo
                Eh = grid(R*A*T,rh,kpre,kpos,gamma,Eh); % Iteração na malha
% esparsa
            end
        e = T*Eh;                % Interpolação do erro para malha anterior
        t = t+1; level = level+1;
        %plot([t-1 t],[level-1 level],'bo-');             % Somente para
% controle e visualização do ciclo
        x = x+e;                                          % Correção do vetor x
            for i=1:kpos
                x = M*x+c     % Gauss-Seidel após retornar a malha anterior
            end

    end
end
```

Anexo III

```
function [x,k,e] = ConjGrad(A,b)
% Utiliza o método do Gradiente Conjugado para resolver o problema Ax = b.
% Retorna o vetor x o numero de iterações e o erro. E possível plotar o
% comportamento do erro desabilitando "e" e habilitando "e(k)"
% Obs.: O método é originalmente desenvolvido para matrizes A simétricas.

clc
close all

% Para Matriz A Não Simétrica================================================

% b = A'*b;
% A = A'*A;

% Checa dimensões A(n,m)x = b(n) ============================================

[n,~] = size(A);                % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))      % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A');
    end

% Erro admissivél ===========================================================

erro_adm = 10^-6;             % Em porcentagem de |b|.
kmax = n;                     % # máximo de iterações

%Gradiente Conjugado ========================================================

x = zeros(length(b),1);         % x inicial.
r0 = b-A*x;                     % Resíduo Inicial.
r = r0;
d = r;                          % Direção de Procura Inicial.
rr = r'*r;
k = 1;                    % Para a contagem de iterações k(# iterações = k-1)
%e(k) = norm(r)/norm(b);         % Erro inicial
e = norm(r)/norm(b);
%    while e(k) > erro_adm
    while e > erro_adm
        Ad = A*d;
        alfa = rr/(d'*Ad);      % Calcula o passo alfa.
        x = x + alfa*d;         % Calcula o novo x.
        r = r - alfa*Ad;        % Calcula o novo resíduo.
        rrold = rr;             % Calcula o produto r'(k-1)r(k-1.
        rr = r'*r;              % Calcula o produto r'(k)r(k).
        beta = rr/(rrold);      % Calcula novo fator beta.
        d = r + beta*d;         % Calcula nova direção de procura.
%        e(k+1) = norm(A*x-r0,'fro')/norm(b); % Calcula erro.
        e = norm(A*x-r0,'fro')/norm(b);
        k = k+1;                % Atualiza contador.
            if k == kmax
                break           % Se não convergir no número máximo
% permitido de iterações, interromper processo.
            end
    end
k = k-1; % # de iterações, pois x0 =x(1)
%plot(e,'-o');
end
```

```
function [x,k,e] = GMRES(A,b)
% Utiliza o método dos Resíduos Mínimos Generalizados para resolver o
% problema Ax = b. O programa retorna o vetor x o numero de iterações e o
% erro. O comportamento do erro pode ser observado desabilitando "e" e
% habilitando "e(k)".

clc
close all

% Checa dimensões A(n,m)x = b(n) ==========================================

[n,~] = size(A);                % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))      % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A');
    end

% Erro admissível ==========================================================

erro_adm = 10^-6;               % Em porcentagem de |b|.
kmax = n;                       % # máximo de iterações

% Valores Iniciais =========================================================

x0 = zeros(length(b),1);        % x inicial.
r0 = b-A*x0;                    % Resíduo Inicial.
k = 1;                  % Para a contagem de iterações k (# iterações = k-1)
%e(k) = norm(r0)/norm(b);        % Erro inicial
e = norm(r0)/norm(b);

% Ortogonalização de Arnoldi ===============================================

nr0 = norm(r0);                     % Norma do resíduo Inicial.
V(:,1) = r0/nr0;                    % Primeiro vetor ortonormal da base.
%    while e(k)> erro_adm
    while e > erro_adm
        t = A*V(:,k);               % Computa novo vetor da base t ainda
% não ortonormalizado.
            for i = 1:k
            H(i,k) = V(:,i)'*t;     % Calcula a componente do vetor t na
% direção do vetor v(i)(suas projeções).
            t = t - H(i,k)*V(:,i);  % Subtrai as projeções para tornar t
% ortogonal.
            end
        H(k+1,k) = norm(t);         % Calcula norma de t. H =(k+1)xk
        V(:,k+1) = t/H(k+1,k);      % Normaliza t         V = nxk+1

% Rotação de Givens ========================================================

        [m,~]=size(H);
        G = speye(m,m);  % Define G inicialmente como uma matriz identidade
        R = H;
            for i = 2:m
                r = sqrt(R(i,i-1)^2+R(i-1,i-1)^2);  % Calcula o "raio".
                c = R(i-1,i-1)/r;                   % Calcula o cosseno da
% matriz de rotação de Givens.
                s = - R(i,i-1)/r;                   % Calcula o seno da
% matriz de rotação de Givens.
```

```
                G(i,i) = c;                             % Define o valor dos
% elementos da matriz que são diferentes da Identidade;
                G(i-1,i-1) = c;
                G(i,i-1) = s;           % Seno para o caso onde j(i) > i(i-1)
                G(i-1,i) = -s;
                    if i<m
                        G = [speye(m,m),G];             % Estoca as matrizes
% Gn,Gn-1,...,G1 em uma única matriz na forma G = [Gn Gn-1 ... G1].
                        R = G(:,m+1:2*m)*R;             % Computa nova matriz Q
% para eliminar próximo índice.
                    else
                        G = G;                          % Evita que seja
% estocado no formato G = [I Gn Gn-1 ... G1]. Q = Gn = (k+1)x(k+1)
                        R = G(:,1:m)*R;    % Computa a matriz R. R ~ (k+1)xk
                    end
            end

% Mínimos quadrados ======================================================

        e1 = zeros(k+1,1);
        e1(1,1) = nr0;
        Qb = e1;                                        % Qb ~ (k+1)x1
            for i=1:m-1
            Qb = G(:,m*(m-1-i)+1:m*(m-i))*Qb;
            end
        y = R(1:k,:)\Qb(1:k,:);                         % y ~ (k,1)
        x = x0 + V(:,1:k)*y;                            % Calcula novo x
        r = b-A*x;                                      % Novo resíduo
%        e(k+1) = norm(r)/norm(b);                       % Novo erro;
        e = norm(r)/norm(b);
        k = k+1;                                        % Atualiza contagem;
            if k == kmax
                break         % Se não convergir no número máximo permitido
% de iterações, interromper processo.
            end
    end
%plot(e,'-o');                % Plota o comportamento do erro
k = k-1;                      % Numero de iterações, pois x0 = x(1)
end
```

```
function [x,k,alfa,e] = GMRESm(A,b)
% Utiliza o método dos Resíduos Mínimos Generalizados com recomeço a cada m
% iterações para resolver o problema Ax = b. O programa retorna o vetor x o
% numero de iterações, numero de recomeços e o erro. O comportamento do
% erro pode ser observado desabilitando "e" e habilitando "e(k)".

clc
close all

% Checa dimensões A(n,m)x = b(n) ==========================================

[n,~] = size(A);                % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))      % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A');
    end

% Erro admissível ==========================================================

global erro_adm kmax rec
erro_adm = 10^-6;                 % Em porcentagem de |b|.
kmax = n;                         % # máximo de iterações
rec = 20;                         % Número de Iterações até recomeço

% Valores Iniciais =========================================================

x0 = zeros(length(b),1);        % x inicial.
k = 1;                  % Para a contagem de iterações k (# iterações = k-1)
alfa = 1;     % Para contagem de número de recomeços (# recomeços = alfa-1)
r0 = b-A*x0;                    % Resíduo Inicial
%e(k) = norm(r0)/norm(b);        % Erro inicial
e = norm(r0)/norm(b);
[x,k,e,alfa] = RESm(A,b,x0,r0,alfa,e,k);
k =k-1;                         % Numero de iterações, pois x0 = x(1)
alfa = alfa-1;                  % Numero de recomeços, pois alfa = 1 para as
% iterações iniciais, quando ainda não houve recomeço.
end

function [x,k,e,alfa] = RESm(A,b,x0,r0,alfa,e,k)
global erro_adm kmax rec

% Ortogonalização de Arnoldi ===============================================
nr0 = norm(r0);                     % Norma do resíduo Inicial.
V(:,1) = r0/nr0;                    % Primeiro vetor ortonormal da base.
    for j = 1:rec
        t = A*V(:,j);               % Computa novo vetor da base t ainda
% não ortonormalizado.
            for i = 1:j
            H(i,j) = V(:,i)'*t;     % Calcula a componente do vetor t na
% direção do vetor v(i) (suas projeções).
            t = t - H(i,j)*V(:,i);  % Subtrai as projeções para tornar t
% ortogonal.
            end
        H(j+1,j) = norm(t);         % Calcula norma de t. H =(k+1)xk
        V(:,j+1) = t/H(j+1,j);      % Normaliza t         V = nxk+1

% Rotação de Givens ========================================================
```

```
        [m,~]=size(H);
        G = speye(m,m); % Define G inicialmente como uma matriz identidade.
        R = H;
            for i = 2:m
                r = sqrt(R(i,i-1)^2+R(i-1,i-1)^2);  % Calcula o "raio".
                c = R(i-1,i-1)/r;                   % Calcula o cosseno da
% matriz de rotação de Givens.
                s = - R(i,i-1)/r;                   % Calcula o seno da
% matriz de rotação de Givens.
                G(i,i) = c;                         % Define o valor dos
% elementos da matriz que são diferentes da Identidade;
                G(i-1,i-1) = c;
                G(i,i-1) = s;         % Seno para o caso onde j(i) > i(i-1)
                G(i-1,i) = -s;
                    if i<m
                        G = [speye(m,m),G];         % Estoca as matrizes
% Gn,Gn-1,...,G1 em uma única matriz na forma G = [Gn Gn-1 ... G1].
                        R = G(:,m+1:2*m)*R;         % Computa nova matriz Q
% para eliminar próximo elemento.
                    else
                        G = G;                      % Evita que seja
% estocado no formato G = [I Gn Gn-1 ... G1]. Q = Gn = (k+1)x(k+1)
                        R = G(:,1:m)      % Computa a matriz R. R = (k+1)xk
                    end
            end
% Mínimos quadrados ======================================================
        e1 = zeros(j+1,1);
        e1(1,1) = nr0;
        Qb = e1;                                    % Qb = (k+1)x1
            for i=1:m-1
            Qb = G(:,m*(m-1-i)+1:m*(m-i))*Qb;
            end
        y = R(1:j,:)\Qb(1:j,:);                     % y = (k,1)
        x = x0 + V(:,1:j)*y;                        % Calcula novo x
        r = b-A*x;                                  % Novo resíduo
        k = ((alfa-1)*rec)+j;             % Atualiza contagem de iterações;
%        e(k) = norm(r)/norm(b);                     % Novo erro;
        e = norm(r)/norm(b);
%            if e(k) < erro_adm
            if e < erro_adm
                break         % Se convergir, interromper processo.
            end
            if k == kmax
                break         % Se não convergir no número máximo permitido
% de iterações, interromper processo.
            end
    end
        if k == kmax          % Se não convergir no número máximo permitido
% de iterações, interromper processo.
            display('Não convergiu no número máximo de iterações
permitidas');
            return            % Retorna a função anterior GMRESm
        end
%        if e(k) > erro_adm
        if e > erro_adm       % Se erro e a maior que o admissível, repetir
% processo
            alfa = alfa+1;                % Atualiza contagem de recomeços
            [x,k,e,alfa] = RESm(A,b,x,r,alfa,e,k);   % Se não convergir
% repetir o processo.
        end
end
```

```
function [x,k,e] = GMRESPreCondEsq(A,b)
% Utiliza o método dos Resíduos Mínimos Generalizados com pré-condicionador
% à esquerda para resolver o problema Ax = b. Os pré-condicionadores
% utilizados são aqueles dos métodos estacionários.  O programa retorna o
% vetor x o numero de iterações e o erro. O comportamento do erro pode ser
% observado desabilitando "e" e habilitando "e(k)".

clc
close all

% Checa dimensões A(n,m)x = b(n) ==========================================

[n,~] = size(A);                % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))      % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A');
    end

% Erro admissível ==========================================================

erro_adm = .015;                % Em porcentagem de |b|.
kmax = n;                       % # máximo de iterações

% Pré-condicionador ========================================================
D = diag(diag(A));              % Diagonal de A;
L = tril(A)-D;                  % Parte estritamente triangular inferior de A
w = 4/15;                       % Coeficiente w do método ponderado de Jacobi
wsor = 1.5;                     % Coeficiente w do método SOR

%P = D;                          % Pré-Condicionador de Jacobi
%P = (1/w)*D;                    % Pré-Condicionador de Jacobi Ponderado
P = L+D;                        % Pré-Condicionador de Gauss
%P = D+wsor*L;                   % Pré-Condicionador SOR

% Valores Iniciais =========================================================

x0 = zeros(length(b),1);        % x inicial.
r0 = P\(b-A*x0);                % Resíduo Inicial.
k = 1;                  % Para a contagem de iterações k (# iterações = k-1)
invPb = P\b;
%e(k) = norm(r0)/norm(b);        % Erro inicial
e = norm(r0)/norm(b);

% Ortogonalização de Arnoldi ===============================================

nr0 = norm(r0);                     % Norma do resíduo Inicial.
V(:,1) = r0/nr0;                    % Primeiro vetor ortonormal da base.
%    while e(k)> erro_adm
    while e > erro_adm
        t = (P\A)*V(:,k);           % Computa novo vetor da base t ainda
% não ortonormalizado.
            for i = 1:k
            H(i,k) = V(:,i)'*t;     % Calcula a componente do vetor t na
% direção do vetor v(i)(suas projeçoes).
            t = t - H(i,k)*V(:,i);  % Subtrai as projeções para tornar t
% ortogonal.
            end
        H(k+1,k) = norm(t);         % Calcula norma de t. H =(k+1)xk
```

```
        V(:,k+1) = t/H(k+1,k);      % Normaliza t         V = nxk+1

% Rotação de Givens ======================================================

        [m,~]=size(H);
        G = speye(m,m);                                 % Define G inicialmente
% como uma matriz identidade.
        R = H;
            for i = 2:m
                r = sqrt(R(i,i-1)^2+R(i-1,i-1)^2);  % Calcula o "raio".
                c = R(i-1,i-1)/r;                       % Calcula o cosseno da
% matriz de rotação de Givens.
                s = - R(i,i-1)/r;                       % Calcula o seno da
% matriz de rotação de Givens.
                G(i,i) = c;                             % Define o valor dos
% elementos da matriz que são diferentes da Identidade;
                G(i-1,i-1) = c;
                G(i,i-1) = s;                           % Seno para o caso onde
% j(i) > i(i-1).
                G(i-1,i) = -s;
                    if i<m
                        G = [speye(m,m),G];         % Estoca as matrizes
% Gn,Gn-1,...,G1 em uma única matriz na forma G = [Gn Gn-1 ... G1].
                        R = G(:,m+1:2*m)*R;         % Computa nova matriz Q
% para eliminar próximo índice.
                    else
                        G = G;                          % Evita que seja
% estocado no formato G = [I Gn Gn-1 ... G1]. Q = Gn = (k+1)x(k+1)
                        R = G(:,1:m)*R;    % Computa a matriz R. R = (k+1)xk
                    end
            end
% Mínimos quadrados ======================================================

        e1 = zeros(k+1,1);
        e1(1,1) = nr0;
        Qb = e1;                                        % Qb = (k+1)x1
            for i=1:m-1
            Qb = G(:,m*(m-1-i)+1:m*(m-i))*Qb;
            end
        y = R(1:k,:)\Qb(1:k,:);                         % y = (k,1)
        x = x0 + V(:,1:k)*y;                            % Calcula novo x
        r = P\(b-A*x);                                  % Novo resíduo
%         e(k+1) = norm(r)/norm(b);                      % Novo erro;
        e = norm(r)/norm(b);
        k = k+1;                                        % Atualiza contagem;
            if k == kmax
                display('Não convergiu no número máximo de iterações
permitidas');
                break         % Se não convergir no número máximo permitido
% de iterações, interromper processo.
            end
    end
k = k-1;          % Numero de iterações, pois x0=x(1)
%plot(e,'-o')
end
```

```
function [x,k,e] = GMRESPreCondDir(A,b)
% Utiliza o método dos Resíduos Mínimos Generalizados com pré-condicionador
% a direita para resolver o problema Ax = b. Os pré-condicionadores
% utilizados são aqueles dos métodos estacionários.  O programa retorna o
% vetor x o numero de iterações e o erro. O comportamento do erro pode ser
% observado desabilitando "e" e habilitando "e(k)".

clc
close all

% Checa dimensões A(n,m)x = b(n) ========================================

[n,~] = size(A);               % n = # de linhas; m = # de colunas.

    if any([n 1]~=size(b))     % Checa se # de linhas de A = # linhas de b.
        error('b deve ter o mesmo número de linhas que a matriz A');
    end

% Erro admissível ========================================================

erro_adm = .015;               % Em porcentagem de |b|.
kmax = n;                      % # máximo de iterações

% Pré-condicionadores ====================================================

D = diag(diag(A));             % Diagonal de A;
L = tril(A)-D;                 % Parte estritamente triangular inferior de A
w = 4/15;                      % Coeficiente w do método ponderado de Jacobi
wsor = 1.5;                    % Coeficiente w do método SOR

%P = D;                         % Pré-Condicionador de Jacobi
%P = (1/w)*D;                   % Pré-Condicionador de Jacobi Ponderado
P = L+D;                       % Pré-Condicionador de Gauss
%P = D+wsor*L;                  % Pré-Condicionador SOR

% Valores Iniciais =======================================================

x0 = zeros(length(b),1);       % x inicial.
r0 = b-A*x0  ;                 % Resíduo Inicial.
k = 1;                  % Para a contagem de iterações k (# iterações = k-1)
%e(k) = norm(r0)/norm(b);       % Erro inicial
e = norm(r0)/norm(b);

% Ortogonalização de Arnoldi =============================================

nr0 = norm(r0);                    % Norma do resíduo Inicial.
V(:,1) = r0/nr0;                   % Primeiro vetor ortonormal da base.
%    while e(k)> erro_adm
     while e > erro_adm
        t = (A/P)*V(:,k);              % Computa novo vetor da base t
%ainda não ortonormalizado.
            for i = 1:k
            H(i,k) = V(:,i)'*t;     % Calcula a componente do vetor t na
% direção do vetor v(i) (suas projeções).

            t = t - H(i,k)*V(:,i);  % Subtrai as projeções para tornar t
% ortogonal.
```

```
            end
        H(k+1,k) = norm(t);          % Calcula norma de t. H =(k+1)xk
        V(:,k+1) = t/H(k+1,k);       % Normaliza t          V = nxk+1

% Rotação de Givens ======================================================

        [m,~]=size(H);
        G = speye(m,m);                                  % Define G inicialmente
% como uma matriz identidade.
        R = H;
            for i = 2:m
                r = sqrt(R(i,i-1)^2+R(i-1,i-1)^2);  % Calcula o "raio".
                c = R(i-1,i-1)/r;                        % Calcula o cosseno da
% matriz de rotação de Givens.
                s = - R(i,i-1)/r;                        % Calcula o seno da
% matriz de rotação de Givens.
                G(i,i) = c;                              % Define o valor dos
% elementos da matriz que são diferentes da Identidade;
                G(i-1,i-1) = c;
                G(i,i-1) = s;                            % Seno para o caso onde
% j(i) > i(i-1).
                G(i-1,i) = -s;
                    if i<m
                        G = [speye(m,m),G];          % Estoca as matrizes
% Gn,Gn-1,...,G1 em uma única matriz na forma G = [Gn Gn-1 ... G1].
                        R = G(:,m+1:2*m)*R;          % Computa nova matriz Q
% para eliminar próximo índice.
                    else
                        G = G;                           % Evita que seja
% estocado no formato G = [I Gn Gn-1 ... G1]. Q = Gn = (k+1)x(k+1)
                        R = G(:,1:m)*R;    % Computa a matriz R. R ~ (k+1)xk
                    end
            end
% Mínimos quadrados ======================================================

        e1 = zeros(k+1,1);
        e1(1,1) = nr0;
        Qb = e1;                                         % Qb ~ (k+1)x1
            for i=1:m-1
            Qb = G(:,m*(m-1-i)+1:m*(m-i))*Qb;
            end
        y = R(1:k,:)\Qb(1:k,:);                          % y ~ (k,1)
        x = x0 + P\V(:,1:k)*y;                           % Calcula novo x
        r = b-A*x;                                       % Novo resíduo
%        e(k+1) = norm(r)/norm(b);                        % Novo erro;
        e = norm(r)/norm(b);
        k = k+1;                                         % Atualiza contagem;
            if k == kmax
                display('Não convergiu no número máximo de iterações
permitidas');
                break         % Se não convergir no número máximo permitido
% de iterações, interromper processo.
            end
    end
k = k-1;
%plot(e,'-o')
End
```

Anexo VIII

```
function divnode(X,t)
    # Realiza divisao binária dos nós da malha;
    # X = matrix que contêm as coordenadas dos nós;{t,2}
    # t = vetor com os números dos nós; [t]
    n = length(t)    # Quantidade de nós
    x = X[t,:] # Matrix com as coordenadas dos nós que pertencem ao bloco a
    ser dividido; {t,2}
    c = mean(x,1)    # Vetor com as coordenadas do centro geometrico do
    conjunto de nós;{1,2}
    # mean(x,1) = média ao longo da 1 dimensão da matrix (1dim = linhas).
    covx = cov(x)                        # Calcula matrix de covariancia de x
    eig_valx,eig_vecx  = eig(covx)
    # Calcula autovalores e autovetores da matriz de covariancia de x
    ref = eig_vecx[:,indmax(eig_valx)]
    # Define como referencia o autovetor relacionado ao maior autovalor
    # Direcao desse autovetor e a direcao de maior variabilidade dos dados
    attcond = zeros(n)
        for i=1:n
            attcond[i] = (x.-c)[i,:]'*ref
            # Condicao que divide os nos em dois blocos diferentes.
        end
    x1=t[attcond.>=0]          # Bloco tal que a condicao e >= 0
    x2=t[attcond.<0]           # Bloco tal que a condicao e < 0
    diam = 2*maximum(sqrt.(((x.-c).*(x.-c))[:,1]+((x.-c).*(x.-c))[:,2]))
    # Calcula diametro do conjunto de dados, centralizando eles;
    dia = 2*norma do ponto mais distante do centro
    return x1,x2,diam,c
end

function cluster(X, max_elem,η = 1.0)
    # X = Coordenadas (x,y) dos nós
    # max_elem = Define máximo de nós em cada folha, tal que:
    max_elem/2 =< nós em cada folha < max_elem
    m,n = size(X)
    # Tamanho da matriz contendo as coordernadas de cada nó {m,2}
    max_clt = ceil(Int,2*m/max_elem)  # Define limite superior para tamanho
   (nº de linhas) das matrizes e vetores utilizados
    child1 = zeros(1,2*max_clt)
    child2 = zeros(1,2*max_clt)
    t = collect(1:m)                # Nós
    inode = 1                       # Começa a contagem de nós da árvore
    ileaf = 1                       # Começa a contagem de folhas da árvore
    nodes = Array{Any}(2*max_clt)  # Aloca um vetor de vetores para guardar
    os nós da malha pertencentes a cada nó da árvore
    # Nodes[i] = vetor com os nós da malha pertencentes ao nó i da árvore
    leaves = Array{Any}(2*max_clt) # Aloca um vetor para guardar as folhas
    child = zeros(Int,2*max_clt,2)
    # Aloca uma matriz para guardar os filhos de cada nó.
    # Child[i,:] = filhos do nó i
    nodes[1] = t
    # O 1º nó da árvore (raiz) contem todos os nós da malha.
    center_row = zeros(2*max_clt,2)  # Aloca um vetor para guardar o centro
    geometrico de cada bloco de nós da malha.
    diam = zeros(2*max_clt)
   # Aloca um vetor para guardar o diametro de cada bloco de nós da malha.
    i = 1                           # Começa contagem de nós
    while inode >= ileaf
    # Enquanto o quantidade de nós for maior que a de folhas.
    # Observe que a condição só não vai ser satisfeita quando a árvore
    estiver completa.
```

```
        t1,t2,d,c = divnode(X,nodes[i])
        # Executa a rotina que divide os nós da malha.
        center_row[i,:] = c;    # Salva centro geometrico do nó i da árvore
        diam[i] = d;                    # Salva diametro do nó i da árvore
            if length(t1)> max_elem      # Se a quantidade de nós em t1 for
            maior que max_elem, definir como nó
                inode = inode + 1            # Chama proximo nó
                nodes[inode] = t1            # Define t1 como um nó
                child[i,1] = inode           # Define t1 como filho do nó i
            else                             # Se a quantidade de nós for
            menor que max_elem, definir como folha
                leaves[ileaf] = t1           # Define t1 como uma folha
                ileaf = ileaf + 1            # Chama proxima folha
                child1[i] = ileaf            # Define t1 como folha do nó i
            end
            # Realiza o mesmo para t2-------------------------------------
-
            if length(t2) > max_elem
                inode = inode + 1
                nodes[inode] = t2
                child[i,2] = inode
            else
                leaves[ileaf] = t2
                ileaf = ileaf + 1
                child2[i] = ileaf
            end
            # -------------------------------------------------------------
-
        i = i + 1
    end

    Tree = Array{Any}(inode+ileaf-1)    # Define tamanho da árvore
    for i=1:inode                       # Para todos os nós
        Tree[i] = nodes[i]              # Insere os nós na árvore
        if child1[i] > 0                # Se aquele nó tem folhas
            child[i,1] = child1[i] + inode - 1
            # Adiciona as folhas pares na matriz child
        end
        if child2[i] > 0                # Se aquele nó tem folhas
            child[i,2] = child2[i] + inode - 1
            # Adiciona as folhas impares na matriz child
        end
    end

    for i=1:ileaf-1     # Para todos as folhas
        Tree[inode+i] = leaves[i]   # Insere as folhas na árvore
        t1,t2,d,c = divnode(X,leaves[i]) # Calcula o diam e c das folhas
        # Havia sido calculado somente os dos bloco que foram divididos, ou
        seja, os nós.
        center_row[inode+i,:] = c
        # Adicona o c das folhas na matrixc center_row
        diam[inode+i] = d
        # Adicona diam das folhas na matrix diam
        child[inode+i,:] = [0 0]
       # Completa a matrix child com par [0,0], pois folhas nao tem filhos
    end

    admiss = zeros(inode+ileaf-1,inode+ileaf-1)
    # Cria matriz para alocar o resultado da aplicadao da condicao de
    admissiblidade entre os blocos
    for i=1:inode+ileaf-1     # Para todos os nós da malha
```

```
        for j=1:inode+ileaf-1
            admiss[i,j] = η*norm(center_row[i]-center_row[j],2) - max(diam[
i],diam[j])
            # Condicao de adimissiblidade, para satisfazer deve ser > 0
        end
    end
    allow = admiss.>=0; # Salva blocos onde a condicao e satisfeita
    block = blocks(Tree,child,allow)
    # Funcao que retorna os blocos admissiveis
    return  Tree,block
end

function blocks(Tree,child,allow)
    fc1 = [2; 2; 3; 3]   # Primeiros Blocos a serem avaliados
    fc2 = [2; 3; 2; 3]   # Primeiros Blocos a serem avaliados
    # fc1(1) e fc(2) formam blocos a serem analisados -> (22, 23, 32, 33)
    block = zeros(Int,0,3)
    # Matrix que aloca os blocos admissiveis [:,1:2] e se atende a condicao
    de admissiblidade [:,3]
    c1 = 0;     # Contador
    while c1 < length(fc1)/2
        for i=1:2
            if allow[fc1[c1*2+i],fc2[c1*2+i]]==1
            # Se blocos são admissiveis
                block = vcat(block,[fc1[c1*2+i] fc2[c1*2+i] 1])
                # Adicionar blocos e identificador 1 (admissivel) a proxima
                linha matrix block
            else # Se blocos não são admissiveis
                if child[fc1[c1*2+i],1]==0 && child[fc2[c1*2+i],1]==0
                # Se ambos os blocos não tem filhos, ou seja, se ambos sao
                folhas
                    block = vcat(block,[fc1[c1*2+i] fc2[c1*2+i] 0])
                    # Adicionar blocos e identificador 0 (não admissivel) a
                    proxima linha matrix block
                else
                    if length(Tree[fc1[c1*2+i]])>=length(Tree[fc2[c1*2+i]])
                    # Se a quantidade de elementos no bloco Tree[fc1[...]]]
                    e >=  Tree[fc2[...]]
                        fc1 = [fc1; child[fc1[c1*2+i],:]]
                        # Adiciona filhos a fc1[...]
                        fc2 = [fc2; fc2[c1*2+i]; fc2[c1*2+i]]
                        # Repete elemento de fc2[...]
                    else
                        fc1 = [fc1; fc1[c1*2+i]; fc1[c1*2+i]]
                        # Repete elemento de fc1[...]
                        fc2 = [fc2; child[fc2[c1*2+i],:]]
                        # Adiciona filhos a fc2[...]
                    end
                end
            end
        end
        c1 = c1 + 1  # Atualiza contador
    end
    return block
    #Matriz que contem os blocos analisados e sua condicao de admiiblidade
end
```

Anexo IX

```
function ACAF_analitico(Tree,block,fHeG,arg,erro=1e-5)
    # arg = [NOS1,NOS_GEO1,tipoCDC,valorCDC,normal,ELEM1,k]
    #         1      2        3      4        5     6   7
    n = size(block,1)               # Quantidade de Submatrizes
    Aaca = Array{Any}(n,2)
    # Cria vetor{Any} que armazena submatrizes [N° de submatrizes x 2]
    b = zeros(size(arg[1],1))
    # Cria matriz b, Ax=b, de zeros [N° de nos x 1]
    for i=1:n                       # Para cada Submatriz
        b1 = Tree[block[i,1]]
        # Nós I da malha que formam a submatriz (Pontos Fonte) (linhas)
        b2 = Tree[block[i,2]]
        # Nós J da malha que formam a submatriz (Pontos Campo) (Colunas)
        # Submatriz = Produto cartesiano I x J
        if block[i,3]==0            # Se esses blocos não são admissiveis
            Aaca[i,1],B = fHeG(b1,b2,arg)
            # Salva na linha i da 1° coluna da matriz Aaca a matriz A e
            salva a matriz B
            b[b1] = b[b1] + B*arg[4][b2]
            # Contribuicao para o valor de G*q dos nos que formam b2
        else                    # Caso contrario (Se blocos são admissiveis)
            INDB1=[]
            # Vetor para armazenar indice das linhas calculadas
            INDB2=[]
            # Vetor para armazenar indice das colunas calculadas
            B1 = zeros(0,length(b2))
            # Aloca vetor para salvar as linhas calculadas da matriz G
            B2 = zeros(length(b1),0)
            # Aloca vetor para salvar as colunas calculadas da matriz G
            ind1 = trues(length(b1))
            # Aloca vetor para salvar se linha foi calculada ou não
            ind2 = trues(length(b2))
            # Aloca vetor para salvar se coluna foi calculada ou não
            indaref = 1
            # Contador para o numero de linhas (Pontos fonte)
            aref = 0*ind2
            # Vetor que a armazena a linha calculada da matriz H
            for indaref = 1:length(ind1) # Calcula linha Auxiliar
            # Para todos os Pontos fonte (para todas as linhas da matriz)
                aref,btemp = fHeG(b1[indaref],b2,arg)
                # Calcula uma linha da matriz A (aref) e uma linha da
                matriz B (btemp)
                aref = aref.'           # Salva o vetor como linha
                push!(INDB1,indaref)
                # Armazena o indice da linha calculada em INDB1
                B1 = [B1;btemp]
                # Adiciona a B1 a linha calculada da matriz G
                if norm(aref) > 1e-10
                # Se a norma da linha caculada for maior que 1e-10
                    break               # Não calcular mais linhas
                end
                ind1[indaref] = 0           # Torna true = 1, em falso = 0
                # Indica que a linha [indaref] foi calculada de forma exata
            end
            if ind1==falses(ind1)
            # Se todas as linhas foram calculadas, norma de todas elas
            e < 1e-10 = Matriz nula
                Aaca[i,1] = ind1*0
                # Salva na linha i da 1° coluna da matriz Aaca uma coluna
                de zeros
```

```
    Aaca[i,2] = ind2.'*0
    # Salva na linha i da 2° coluna da matriz Aaca uma linha
    de zeros
else                # Se nem todas as linhas foram calculadas
    indbref = 1
    # Contador para o numero de colunas (Pontos campo)
    bref = 0*ind1
    # Vetor que a armazena a coluna calculada da matriz H
    for ii = 1:length(ind2)      # Calcula coluna Auxiliar
 # Para todos os pontos campo (para todas as colunas da matriz)
        indbref = indmin(abs.(aref[ind2])) # ACA+
        indbref = indbref + cumsum(ind2.==0)[ind2][indbref]
        # Indice de menor valor em modulo, de uma coluna ainda
        nao calculada, da ultima linha calculada
        bref,btemp = fHeG(b1,b2[indbref],arg)
        # Calcula uma coluna da matriz A (bref) e uma coluna da
        matriz B (btemp)
        push!(INDB2,indbref) # Armazena o indice das colunas
        calculadas em INDB2
        B2 = [B2 btemp]      # Adiciona a B2 a coluna calculada
        da matriz G
        if norm(bref) > 1e-10# Se a norma da linha caculada for
        maior que 1e-10
            break            # Não calcular mais colunas
        end
        ind2[indbref] = 0   # Torna true = 1, em falso = 0
  # Indica que a coluna [indaref] foi calculada de forma exata
    end
    arefmax = indmax(abs.(aref))
    # Indice do maior em modulo da ultima linha calculada
    brefmax = indmax(abs.(bref))
    # Indice do maior em modulo da ultima coluna calculada
    Umax = 0          # Quarda indice maximo coluna (pivo)
    Vmax = 0          # Quarda indice maximo linha (pivo)
    nmin = min(length(ind1),length(ind2))
    # Minimo entre o numero de linhas e colunas da matriz
    U = zeros(length(ind1),nmin)            # Aloca matrix U
    V = zeros(nmin,length(ind2))            # Aloca matrix V
    Ap = zeros(length(ind1),length(ind2))   # Aloca matrix Ap
    norma0 = 0.0                            # Norma Inicial
    cont = 0                                # Define contador
    for cont = 1:nmin-1
    # Ja foi calculado 1 linha e coluna da matrix, são
    necessarios agora no maximo nmin -1
        if abs.(aref[arefmax])>abs.(bref[brefmax])
        # Se o maior valor em modulo (pivo) da linha e maior
        que o da coluna
            U[:,cont],btemp = fHeG(b1,b2[arefmax],arg)
            # Calcula coluna do pivo A[:,i]
            U[:,cont] = U[:,cont] - Ap[:,arefmax]
            # U = A[:,i] - Ap[;,i]  Computa matriz diferenca
            push!(INDB2,arefmax)
            # Armazena o indice da coluna calculada em INDB2
            B2 = [B2 btemp]
            # Adiciona a B2 a coluna calculada da matriz G
            Umax = indmax(abs.(U[:,cont]))
            # Encontra pivo na coluna calculada
            V[cont,:],btemp = fHeG(b1[Umax],b2,arg)
            # Calcula linha do pivo
           V[cont,:] = (V[cont,:][:]-Ap[Umax,:]).'/U[Umax,cont]
            # Define V = (1/pivo)(V-Ap)
```

```
                push!(INDB1,Umax)
                # Armazena o indice da linha calculada em INDB1
                B1 = [B1;btemp]
                # Adiciona a B1 a linha calculada da matriz G
                Vmax = arefmax              # Atualiza arefmax
            else
            # Se o maior valor em modulo(pivo) da coluna e maior
            que o da linha
                V[cont,:],btemp = fHeG(b1[brefmax],b2,arg)
                # Calcula linha do pivo
                V[cont,:] = (V[cont,:][:]-Ap[brefmax,:])
                # Define V = A[i,:] - Ap[i,:]
                Computa matriz diferenca
                push!(INDB1,brefmax)
                # Armazena o indice da linha calculada em INDB1
                B1 = [B1;btemp]
                # Adiciona a B1 a linha calculada da matriz G
                Vmax = indmax(abs.(V[cont,:]))
                # Encontra pivo na linha calculada
                U[:,cont],btemp = fHeG(b1,b2[Vmax],arg)
                # Calcula coluna do pivo
                U[:,cont] = (U[:,cont]-Ap[:,Vmax])/V[cont,Vmax]
                # Define U = (1/pivo) (A[:,i] - Ap[:,i])
                push!(INDB2,Vmax)
                # Armazena o indice da coluna calculada em INDB2
                B2 = [B2 btemp]
                # Adiciona a B2 a coluna calculada da matriz G
                Umax = brefmax              # Atualiza Umax
            end
            Ap = U*V # Calcula Aproximacao
            norma1 = vecnorm(Ap)
            # Calcula norma da matriz aproximada
            if abs.((norma1-norma0)/norma1) < erro
            # Se o erro e menor que o permitido
                # norma1 -norma0 = A(i)-A(i-1)= a(i)b(i)
                break                   # Parar
            else
                norma0 = norma1 # Atualiza norma
            end
            ind1[Umax] = 0
          # Indica que a coluna [Umax] foi calculada de forma exata
            ind2[Vmax] = 0
         # Indica que a linha [Vmax] foi calculada de forma exata
            if indaref==Umax && indbref==Vmax
            # Se os pivos são os mesmos que os de referencia
                    for indaref = 1:sum(ind1)
                # De 1 ate a nº de colunas que não foram calculadas
                        indaref = indmax(ind1)
                        aref,btemp = fHeG(b1[indaref],b2,arg)
# Calcula uma linha da matriz A (aref) e uma linha da matriz B (btemp)
                        aref = aref-Ap[indaref,:]'
                        push!(INDB1,indaref)
                    # Armazena o indice da linha calculada em INDB1
                        B1 = [B1;btemp]
                      # Adiciona a B1 a linha calculada da matriz B
                        if norm(aref) > 1e-10
                 # Se a norma da linha caculada for maior que 1e-10
                            break               # Parar
                        end
                        ind1[indaref] = 0
                  # Indica que a linha foi calculada de forma exata
                    end
```

```
            for indbref = 1:sum(ind2)
        # De 1 ate a nº de colunas que não foram calculadas
                indbref = indmin(abs.(aref[ind2]))
        indbref = indbref + cumsum(ind2.==0)[ind2][indbref]
                bref,btemp = fHeG(b1,b2[indbref],arg)
                # Calcula uma linha da matriz A (aref) e
                uma linha da matriz B (btemp)
                bref = bref-Ap[:,indbref]
                push!(INDB2,indbref)
           # Armazena o indice da coluna calculada em INDB2
                B2 = [B2 btemp]
             # Adiciona a B2 a coluna calculada da matriz G
                if norm(bref) > 1e-10
        # Se a norma da coluna caculada for maior que 1e-10
                    break               # Parar
                end
                ind2[indbref] = 0
         # Indica que a coluna foi calculada de forma exata
            end
      elseif indaref==Umax
    # Se somente o pivo Umax ser o mesmo que os de referencia
            bref = bref-U[:,cont]*V[cont,indbref].'
            for indaref = 1:sum(ind1)
        # De 1 ate a nº de colunas que não foram calculadas
                indaref = indmin(abs.(bref[ind1]))
        indaref = indaref + cumsum(ind1.==0)[ind1][indaref]
                aref,btemp = fHeG(b1[indaref],b2,arg)
                # Calcula uma linha da matriz A (aref) e
                uma linha da matriz B (btemp)
                aref=aref[:]-Ap[indaref,:][:]
                push!(INDB1,indaref)
            # Armazena o indice da linha calculada em INDB1
                B1 = [B1;btemp]
              # Adiciona a B1 a linha calculada da matriz G
                if norm(aref) > 1e-10
        # Se a norma da coluna caculada for maior que 1e-10
                    break               # Parar
                end
                ind1[indaref] = 0
         # Indica que a coluna foi calculada de forma exata
            end
      elseif indbref==Vmax
    # Se somente o pivo Vmax ser o mesmo que os de referencia
            aref = aref[:]-U[indaref,cont]*V[cont,:]
            # Atualiza linha de referencia
            for indbref = 1:sum(ind2)
        # De 1 ate a nº de colunas que não foram calculadas
                indbref = indmin(abs.(aref[ind2]))
        indbref = indbref + cumsum(ind2.==0)[ind2][indbref]
         # Indice de menor valor em modulo, de uma coluna
           ainda nao calculada, da ultima linha calculada
                bref,btemp = fHeG(b1,b2[indbref],arg)
                # Calcula uma linha da matriz A (aref) e
                uma linha da matriz B (btemp)
                bref = bref-Ap[:,indbref]
                # Nova coluna de referencia
                push!(INDB2,indbref)
           # Armazena o indice da coluna calculada em INDB2
                B2 = [B2 btemp]
             # Adiciona a B2 a coluna calculada da matriz G
                if norm(bref) > 1e-10
        # Se a norma da coluna caculada for maior que 1e-10
```

```
                        break                    # Parar
                    end
                    ind2[indbref] = 0
                # Indica que a coluna foi calculada de forma exata
                end
            else
                aref = aref[:] - U[indaref,cont]*V[cont,:]
                # Atualiza linha de referencia
                bref = bref - U[:,cont]*V[cont,indbref].'
                # Atualiza linha de referencia
            end
            arefmax = indmax(abs.(aref))
            # Indice da coluna de referencia
            brefmax = indmax(abs.(bref))
            # Indice da linha de referencia
            Aaca[i,1] = U[:,1:cont]    # Armazena os valores de U,
            para o bloco, na 1° coluna da matrix Aaca
            Aaca[i,2] = V[1:cont,:]    # Armazena os valores de V,
            para o bloco, na 2° coluna damatrix Aaca
        end
        # Todos os blocos já foram calculados
        max1 = ind2sub(size(B1),indmax(abs.(B1[:,INDB2])))
        # Retorna indices do maior elemento de B1[:,INDB2]
        # B1 guarda as linhas calculadas da matriz G, INDB1 contem
        os indices das linhas
        # B1[:,INDB2] contem as linhas das colunas que foram
        calculadas, dadas por INDB2
        maxv = B1[max1[1],INDB2[max1[2]]]
        # Recupera o valor desse elemento
        Vb = B1[max1[1],:].'
        # Define Vb como a linha desse elemento
        Ub = B2[:,max1[2]]/maxv
        # Define Ub como a coluna desse elemento
        Baca = Ub*Vb
        # Calcula a primeira matriz aproximacao
        B1 = B1 - Baca[INDB1,:]
        # Atualiza o valor de B1 subtraindo da aproximacao Gaca
        # B1 contem somente as linhas que foram calculadas
        anteriormente INDB1,
      # e precido pegar as mesmas linhas da matriz aproximada Gaca
        B2 = B2 - Baca[:,INDB2]
        # Atualiza valor de B2 subtraindo da aproximacao Gaca
        for i = 1:size(B1,1)-1
# Para a quantidade de linha em B1, menos uma que ja foi calculada
            # Repete  processo
            max1 = ind2sub(size(B1),indmax(abs.(B1[:,INDB2])))
            # Acha indices do novo pivo
            maxv = B1[max1[1],INDB2[max1[2]]]
            # Definie o pivo
            if abs.(maxv) < 1e-12
            # Se o valor do pivo for menor que 1e-12
                break                # Para processo
            end
            Vb = [Vb; B1[max1[1],:].']
            # Adiciona nova linha a Vb
            Ub = [Ub B2[:,max1[2]]/maxv]
            # Adiciona nova coluna a Ub
            lastUV = Ub[:,end]*Vb[end,:].'
            # Calcula ultima contrinuicao a aproximacao
            # Baca = Vb1*Ub1 + Vb2*Ub2 + ...
            # B - Baca = B - Vb1*Ub1 - Vb2*Ub2 + ...
            B1 = B1 - lastUV[INDB1,:]
```

```
                    # Atualiza o valor de B1 subtraindo da aproximacao Gaca
                    B2 = B2 - lastUV[:,INDB2]
                    # Atualiza valor de B2 subtraindo da aproximacao Gaca
                end
                b[b1] = b[b1] + Ub*Vb*arg[4][b2]
                # Contribuicao para o valor de B*b dos nos que formam b2
                # arg[4] valor da CDC
            end
        end
    end
    return Aaca,b
end
```

11 ANEXO - ELEMENTOS DE CONTORNO EM TRANSFERÊNCIA DE CALOR

Nesse capítulo será discutido o problema de condução de calor bidimensional [16] [17]. Primeiramente a equação de Laplace, a qual é a equação governante do problema, será deduzida. Em seguida, será calculada a solução fundamental da equação de Laplace, a qual é importante para a formulação do MEC. A equação integral de contorno é então obtida e discretizada. Os elementos utilizados na discretização, ou seja, na formulação da malha, serão os elementos lineares contínuos, os quais possibilitam obter solução analítica para algumas das integrais que descrevem o problema.

11.1 EQUAÇÃO DE LAPLACE

O problema de condução de calor é governado pela lei de Fourier, essa relaciona o gradiente de temperatura com a taxa do fluxo de calor.

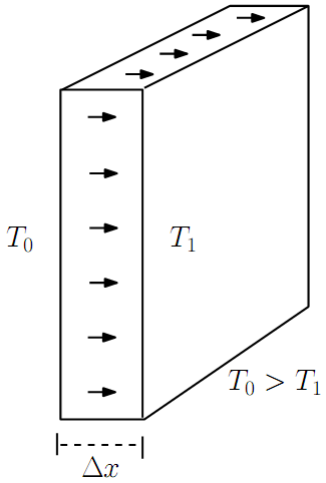

Figura 53 - Fluxo de calor unidirecional através em uma placa retangular.

No caso particular da condução de calor unidirecional (Figura 53), a taxa do fluxo de calor na direção x que passa por um corpo de seção transversal A é definida como:

$$\dot{Q}_x = -kA\frac{(T_1 - T_0)}{\Delta x}$$

onde k é a condutividade térmica do material, A é a área da seção transversal da placa e T_1 e T_0 são, respectivamente, as temperaturas na face fria e na face quente da placa.

Observe que a definição de quente e de frio é relativa e nesse contexto indica somente que $T_0 > T_1$.

Como dito anteriormente $\dot{Q}_x$ é a taxa, ou seja, a variação temporal, do fluxo de calor, nesse caso na direção x dada por:

$$\dot{Q}_x = \frac{\partial Q_x}{\partial t}$$

Considerando agora um pequeno elemento da placa, de largura e área infinitesimal, podemos escrever a equação anterior na forma:

$$\dot{Q}_x = -kA \lim_{\Delta x \to 0} \frac{(T_1 - T_0)}{\Delta x}$$

$$\dot{Q}_x = -kdA \frac{\partial T}{\partial x} \tag{9.1}$$

Essa equação pode ser generalizada para o caso tridimensional, nesse caso é interessante considerar a taxa de condução de calor por unidade de área, tal que:

$$\dot{\vec{q}} = \frac{1}{dA}\left(\dot{Q}_x\vec{\imath} + \dot{Q}_y\vec{\jmath} + \dot{Q}_z\vec{k}\right) = \frac{1}{dA}\left[-kdA\left(\frac{\partial T}{\partial x}\vec{\imath} + \frac{\partial T}{\partial y}\vec{\jmath} + \frac{\partial T}{\partial z}\vec{k}\right)\right]$$

$$\dot{\vec{q}} = -k\left(\frac{\partial T}{\partial x}\vec{\imath} + \frac{\partial T}{\partial y}\vec{\jmath} + \frac{\partial T}{\partial z}\vec{k}\right)$$

$$\dot{\vec{q}} = -k\vec{\nabla}T \tag{9.2}$$

onde $\vec{\nabla}$ é o operador gradiente.

Em sistemas onde não há trabalho mecânico, a 1ª lei na termodinâmica garante que:

$$\sum_{i=1}^{n} \dot{Q}_i = \dot{U} \longrightarrow \dot{Q}_E + \dot{Q}_G = \dot{Q}_S + \dot{U}$$

onde $\dot{Q}_E$, $\dot{Q}_G$ e $\dot{Q}_S$ são, respectivamente, o calor que entra, o que é gerado é o que sai do sistema e $\dot{U}$ é a variação da energia interna, nesse caso energia térmica. Para o caso unidirecional, a 1ª lei na termodinâmica pode ser reescrita como:

$$\dot{Q}_x + \dot{q}_g dV = \left(\dot{Q}_x + \frac{\partial \dot{Q}}{\partial x}dx\right) + c_p \rho V \dot{T}$$

onde $\dot{q}_g$é o calor gerado por unidade de volume, c_p o calor especifico e ρ a densidade do material. Considerando o caso estacionário, o que significa que $\dot{T} = 0$, a equação anterior pode ser reescrita, isolando o termo de geração interna de calor, como:

$$\dot{q}_g dV = \frac{\partial \dot{Q}}{\partial x} dx \tag{9.3}$$

Substituindo (9.1) em (9.3) obtemos:

$$\dot{q}_g dV = \frac{\partial}{\partial x}\left(-kdA\frac{\partial T}{\partial x}\right) dx$$

Considerando k constante ao longo do material e expandindo o termo dV, podemos reescrever como:

$$\dot{q}_g dxdydz = -k\frac{\partial}{\partial x}\left(dA\frac{\partial T}{\partial x}\right) dx$$

$$\dot{q}_g dydz = -k\frac{\partial}{\partial x}\left(dA\frac{\partial T}{\partial x}\right)$$

no entanto $dydz = dA$, e então temos:

$$\dot{q}_g dA = -kdA\frac{\partial^2 T}{\partial^2 x}$$

$$\dot{q}_g = -k\frac{\partial^2 T}{\partial^2 x}$$

A relação anterior pode ser estendida para as três dimensões, obtendo então:

$$\frac{\partial^2 T}{\partial^2 x} + \frac{\partial^2 T}{\partial^2 y} + \frac{\partial^2 T}{\partial^2 z} = -\frac{\dot{q}_g}{k}$$

$$\nabla^2 T = -\frac{\dot{q}_g}{k}$$

onde ∇^2 é o operador laplaciano. Essa equação é conhecida como equação de Fourier para condução de calor e quando não há geração interna de calor, ela é dada por:

$$\nabla^2 T = 0$$

chamada de equação de Laplace.

11.2 DELTA DE DIRAC

Em diversos problemas de engenharia é comum considerar a aplicação de cargas externas de forma concentrada em um ponto do domínio. No entanto, a noção de carga concentrada é relativa e depende da distância entre o local de aplicação da carga externa e o ponto onde se analisa a perturbação gerada por essa. Embora na realidade a ideia de carga concentrada seja algo abstrato, matematicamente ela é muito útil.

É importante lembrar que nesse contexto carga externa é qualquer fonte externa que produz uma função de campo não nula (campo de temperatura, campo de tensão, etc.) que descreve o comportamento do sistema, como por exemplo, fluxo de calor, forças de contato, forças de campo e outras condições de contorno.

Para definir matematicamente uma carga concentrada considere uma função degrau, também conhecida como função pulso, centrada em d e de comprimento a. A função degrau é definida de tal forma que:

$$F(x,d,a) = \begin{cases} 0 & x < d - \dfrac{a}{2} \\ \dfrac{1}{a} & d - \dfrac{a}{2} < x < d + \dfrac{a}{2} \\ 0 & x > d + \dfrac{a}{2} \end{cases}$$

Assim a função degrau é sempre nula no intervalo $d - \frac{a}{2} > x > d + \frac{a}{2}$ e é igual a $F(x,d,a) = \frac{1}{a}$ quando $d - \frac{a}{2} < x < d + \frac{a}{2}$ (Figura 54). Observe também que:

$$\int_{-\infty}^{+\infty} F(x,d,a) = \int_{d-\frac{a}{2}}^{-\infty} F(x,d,a)dx + \int_{d+\frac{a}{2}}^{d-\frac{a}{2}} F(x,d,a)dx + \int_{+\infty}^{d+\frac{a}{2}} F(x,d,a)dx$$

$$= 0 + \int_{d-\frac{a}{2}}^{d-\frac{a}{2}} \frac{1}{a}dx + 0$$

$$\int_{-\infty}^{+\infty} F(x,d,a) = \frac{1}{a}\int_{d-\frac{a}{2}}^{d-\frac{a}{2}} dx = 1$$

ou seja, a integral da função degrau é o valor unitário se o intervalo de integração contém o intervalo $d - \frac{a}{2} < x < d + \frac{a}{2}$, caso contrário a integral é nula.

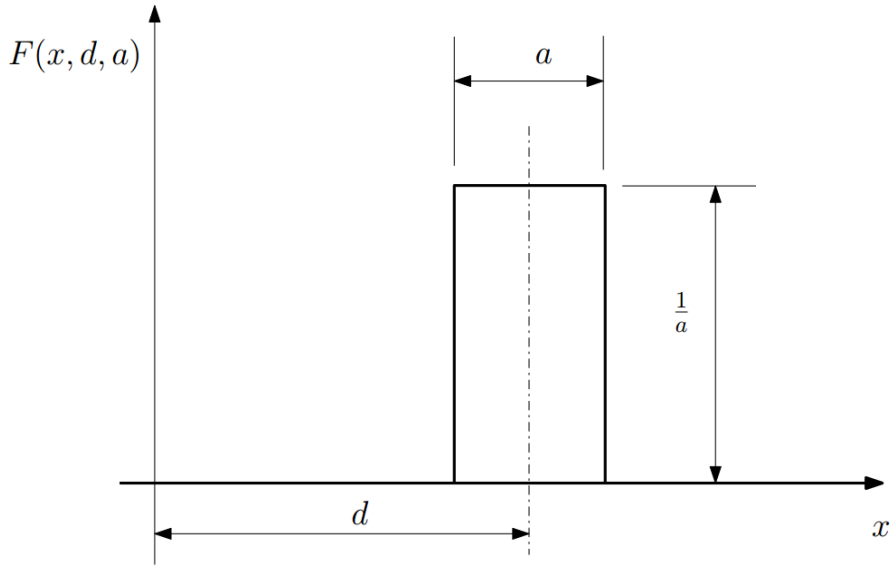

Figura 54 - Função degrau centrada em d com largura a.

Definido à função pulso, a função delta de Dirac é o limite da função pulso quando seu comprimento $a \to 0$, observe que quando isso acontece a função adquire valor não nulo somente quando $x = d$ e seu valor $F(x, d, a) = \frac{1}{a} \to \infty$. Assim, definimos a função delta de Dirac como [18] [19]:

$$\delta(x - d) = \lim_{a \to 0} F\,(x, d, a)$$

$$\delta(x - d) = \begin{cases} \infty & se\ x = d \\ 0 & se\ x \neq d \end{cases}$$

Análogo à função degrau, na integração da função delta de Dirac obtemos:

$$\int_a^b \delta(x - d) = \begin{cases} 0 & se\ d < a \\ 1 & se\ a \leq d \leq b \\ 0 & se\ d > b \end{cases}$$

Outra importante propriedade relacionada à integral é:

$$\int_a^b g(x)\delta(x - d) = \begin{cases} 0 & se\ d < a \\ g(d) & se\ a \leq d \leq b \\ 0 & se\ d > b \end{cases}$$

O delta de Dirac pode ser definido em qualquer dimensão. O modo mais simples para definir um delta de Dirac multidimensional é representar o mesmo como um produto entre deltas de Dirac de menor dimensão.

No caso bidimensional, o delta de Dirac e dado por:

$$\delta(x-d) = \delta(x-x_d)\delta(y-y_d)$$

Observe que $\delta(x-d)$ só não será nulo se $x = x_d$ e $y = y_d$. Nesse caso a integral sobre o domínio A não é mais uma integral simples mais uma integral dupla, de forma que:

$$\iint_A \delta(x-d) = \begin{cases} 1 & se\ d_1\ e\ d_2 \in A \\ 0 & caso\ contrário \end{cases} \tag{9.4}$$

11.3 TEOREMA DE GAUSS

O teorema de Gauss, ou teorema da Divergência, relaciona o fluxo vetorial através das superfícies de um corpo com o comportamento do campo vetorial dentro do corpo. Matematicamente ele permite representar uma equação integral de volume por equações integrais sobre superfícies.

Considere uma função $f(x,y)$ contínua sobre uma área A do domínio. A integral sobre a área A da derivada parcial em x dessa função é dada por:

$$\iint_A \frac{\partial f(x,y)}{\partial x} dA = \iint_A \frac{\partial f(x,y)}{\partial x} dxdy = \int_{y_1}^{y_2} \left(\int_{x_1}^{x_2} \frac{\partial f(x,y)}{\partial x} dx \right) dy$$

resolvendo a integral entre parênteses, podemos reescrever da seguinte forma:

$$\iint_A \frac{\partial f(x,y)}{\partial x} dxdy = \int_{y_1}^{y_2} \left[f\left(x_2(y)\right) - f\left(x_1(y)\right) \right] dy \tag{9.5}$$

Observe que os elementos diferenciais dx e dy podem ser escritos em função de um elemento diferencial dS do contorno S do domínio de integração A (Figura 55).

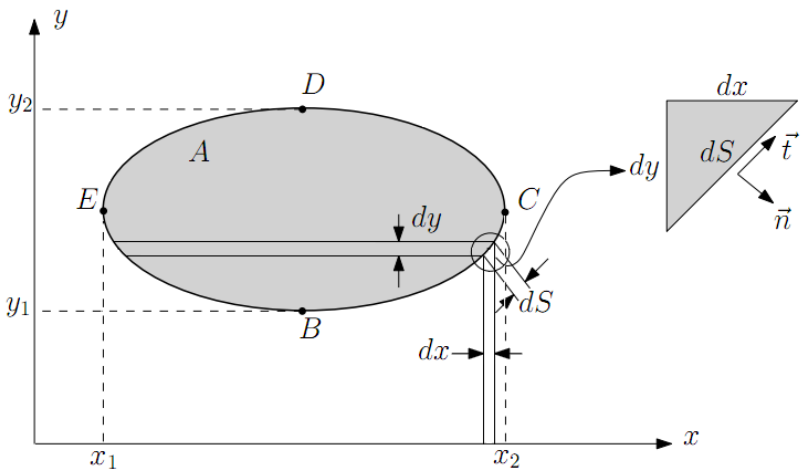

Figura 55 - Elemento diferencial de contorno ds do domínio de integração A.

Da Figura podemos observar que:

$$\vec{t} = \frac{dx}{dS}\vec{\imath} + \frac{dy}{dS}\vec{\jmath} = t_x\vec{\imath} + t_y\vec{\jmath}$$

$$\vec{n} = \frac{dy}{dS}\vec{\imath} - \frac{dx}{dS}\vec{\jmath} = n_x\vec{\imath} + n_y\vec{\jmath} \tag{9.6}$$

deste modo é possível escrever:

$$n_x = \frac{dy}{dS}$$

$$dy = n_x dS \tag{9.7}$$

$$n_y = -\frac{dx}{dS}$$

$$dx = -n_y dS$$

assim, substituindo a equação (9.7) na equação (9.5), obtemos:

$$\int_{y_1}^{y_2} [f(x_2(y)) - f(x_1(y))]\, dy = \int_{y_1}^{y_2} f(x_2(y)) n_x dS - \int_{y_1}^{y_2} f(x_1(y)) n_x dS$$

Observe também na Figura 55 que o trajeto correspondente à integração de y_1 a y_2 corresponde aos arcos BCD ou BED. Assim, podemos reescrever a equação como:

$$\int_{y_1}^{y_2} [f(x_2(y)) - f(x_1(y))]\, dy = \int_{BCD} f(x_2(y)) n_x dS - \int_{BED} f(x_1(y)) n_x dS$$

Inverter o sentido de integração BED para DEB inverte também o sinal da integração e permite que a integral agora possa ser escrita como uma única integral ao longo do contorno BCDEB, o qual é o contorno fechado S do domínio de integração A.

$$\begin{aligned}\int_{y_1}^{y_2}\left[f\left(x_2(y)\right)-f\left(x_1(y)\right)\right]dy &= \int_{BCD} f(s)n_x dS + \int_{DEB} f(s)n_x dS \\ &= \int_{BCD} f(s)n_x dS + \int_{DEB} f(s)n_x dS \\ &= \int_{BCDEB} f(s)n_x dS = \oint_S f(s)n_x dS\end{aligned}$$

Substituindo o resultado em (9.5), obtemos:

$$\iint_A \frac{\partial f(x,y)}{\partial x}\,dA = \oint_s f(s)n_x dS \tag{9.8}$$

É possível demonstrar, de forma análoga, que:

$$\iint_A \frac{\partial f(x,y)}{\partial y}\,dA = \oint_s f(s)n_y dS \tag{9.9}$$

Somando somente o lado esquerdo das equações (9.8) e (9.9), obtemos:

$$\iint_A \frac{\partial f(x,y)}{\partial x}\,dA + \iint_A \frac{\partial f(x,y)}{\partial y}\,dA = \iint_A \left(\frac{\partial f(x,y)}{\partial x} + \frac{\partial f(x,y)}{\partial y}\right)dA = \iint_A \nabla \cdot f\; dA$$

Somando agora somente o lado direito das equações (9.8) e (9.9), obtemos:

$$\oint_S f(s)n_x dS + \oint_S f(s)n_y dS = \oint_S \left(f(s)n_x + f(s)n_y\right)dS = \oint_S (f \cdot \vec{n})dS$$

E finalmente juntando o lado esquerdo e direito novamente, temos:

$$\iint_A \nabla \cdot f\; dA = \oint_S (f \cdot \vec{n})dS$$

o qual é o teorema de Gauss para o caso bidimensional, onde $\nabla \cdot f$ é o operador divergente aplicado em f e $f \cdot \vec{n}$ é o produto escalar de f com o vetor unitário normal a ao contorno de integração s, observe que por convenção o vetor normal a uma superfície fechada aponta sempre externamente a essa.

O teorema de Gauss pode ser aplicado a qualquer dimensão, no espaço tridimensional o mesmo é dado por:

$$\iiint_V \nabla \cdot f \, dV = \oiint_S (f \cdot n) dS$$

onde V é o volume de integração e S são as superfícies que delimitam esse volume.

11.4 SOLUÇÃO FUNDAMENTAL

A solução fundamental é a base de qualquer formulação baseada no MEC. Essa é a solução analítica representando os efeitos de uma carga pontual em um ponto de um domínio infinito. No caso da condução e calor, a solução fundamental representa o campo de temperatura em um domínio infinito gerado por uma fonte de calor concentrada em um ponto qualquer desse domínio. A solução é obtida matematicamente resolvendo a equação de Fourier, para um termo de geração de calor definido pelo delta de Dirac.

$$\nabla^2 T^* = -\frac{\delta(x-d)}{k}$$

Considere k constante e a origem do sistema de coordenadas no ponto fonte, ou seja, no ponto onde é aplicada a fonte de calor. Desse modo, temos que $\delta(x-d) = \delta(x-0) = \delta(x)$, e então o problema apresente simetria polar. Podemos então definir uma função G tal que:

$$-k\nabla^2 T^* = \nabla^2 G = \delta(x)$$

$$-kT^* = G \tag{9.10}$$

Desta forma, a questão a ser resolvida é encontrar uma função $G(r)$ qualquer, onde r é a distância radial, tal que $\nabla^2 G = \delta(x)$. Observe que esse delta de Dirac é bidimensional, pois ele é definido sobre um domínio plano.

Integrando a função $\delta(x)$ no domínio definido por um círculo de raio R, obtemos:

$$1 = \iint_0^R \delta(x) \, dA = \iint_0^R \nabla^2 G \, dA = \iint_0^R \nabla \cdot (\nabla G) \, dA \tag{9.11}$$

A integral tem valor unitário, devido à propriedade (9.4) do delta de Dirac, pois ela contém o ponto $d = (0,0)$. Aplicando o teorema de Green na integral anterior, obtemos:

$$\iint_0^R \nabla \cdot (\nabla G)\, dA = \oint_R \nabla G \cdot n\, dS$$

Como o problema apresenta simetria polar, ou seja, G é função somente de r, o produto escalar $\nabla G \cdot n$ é simplesmente a derivada na direção radial [19] [20].

$$\nabla G \cdot n = \left(\frac{\partial G}{\partial r}\vec{r} + \frac{\partial G}{\partial \theta}\vec{\theta}\right) \cdot \left(\vec{r} + \vec{\theta}\right)$$

$$\frac{\partial G}{\partial \theta} = 0$$

$$\nabla G \cdot n = \left(\frac{\partial G}{\partial r}\vec{r} + 0\,\vec{\theta}\right) \cdot \left(\vec{r} + \vec{\theta}\right) = \frac{\partial G}{\partial r}$$

deste modo, temos que:

$$\iint_0^R \nabla \cdot (\nabla G)\, dA = \oint_R \frac{\partial G}{\partial r}\, dS \tag{9.12}$$

e então:

$$\oint_R \frac{\partial G}{\partial r}\, dS \;=\; \oint_R \frac{\partial G}{\partial r}\, r d\theta \;=\; R\frac{\partial G(R)}{\partial r} \int_0^{2\pi} d\theta \;=\; 2\pi R \frac{\partial G(R)}{\partial r} \tag{9.13}$$

Na integral acima, R é um raio qualquer, ou seja, a integral tem limite superior indefinido. Então, podemos substituir R pela variável r. Assim, substituindo (9.13) em (9.12) obtemos:

$$\iint_0^r \nabla \cdot (\nabla G)\, dV = \; 2\pi r \frac{\partial G}{\partial r} \tag{9.14}$$

Substituindo (9.14) em (9.11), temos:

$$1 = 2\pi r \frac{\partial G}{\partial r}$$

$$\frac{\partial G}{\partial r} = \frac{1}{2\pi}\frac{1}{r}$$

e finalmente integrando, podemos mostrar que:

$$G = \frac{1}{2\pi}\int \frac{1}{r}\,dr = \frac{1}{2\pi}\ln r + C \tag{9.15}$$

A solução fundamental para o problema bidimensional de condução de calor e dada então substituindo (9.15) em (9.10):

$$-kT^* = G = \frac{1}{2\pi}\ln r + C$$

$$T^* = -\frac{1}{2\pi k}\ln r + C$$

A solução final é dada considerando a condição de contorno $T^* \to 0$ quando $r \to \infty$, o que resulta em $C = 0$. Além disso, transladando a origem do sistema de coordenadas para uma posição x_0 qualquer, obtemos então:

$$T^* = -\frac{1}{2\pi k}\ln|x - x_d|$$

ou

$$T^* = -\frac{1}{2\pi k}\ln r \tag{9.16}$$

onde $r = |x - x_d|$ é a distancia radial, tal que x_d é a posição do ponto onde há a geração de calor, chamado de ponto fonte, e x a posição do ponto onde se quer determinar a temperatura, chamado de ponto campo (Figura 56).

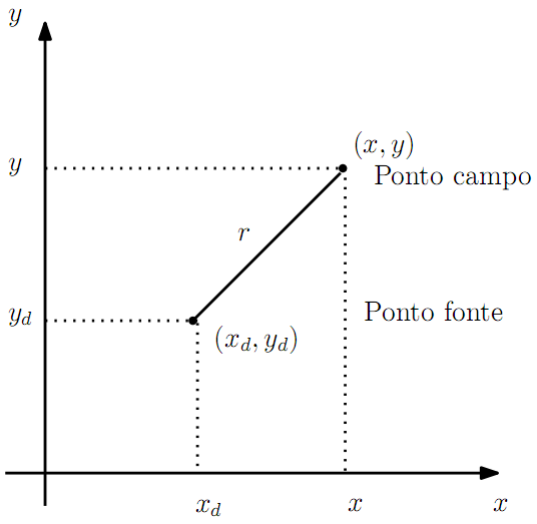

Figura 56 - Ponto fonte e ponto campo

A suavidade assintótica da solução fundamental, a qual é logarítmica, é condição suficiente para existência de uma aproximação de baixo posto da matriz que descreve o problema [8] [10] [11] [21].

11.4.1 SOLUÇÃO FUNDAMENTAL PARA O FLUXO DE CALOR NO CONTORNO

A taxa do fluxo de calor q que passa pelo contorno por unidades de área é dada por:

$$q = \dot{\vec{q}} \cdot \vec{n} \tag{9.17}$$

Substituindo (9.2) em (9.17), obtemos:

$$q = -k\left(\vec{\nabla} T \cdot \vec{n}\right) = -k\left(\frac{\partial T}{\partial n}\right)$$

$$q^* = -k\left(\frac{\partial T^*}{\partial n}\right) \tag{9.18}$$

onde $\frac{\partial T^*}{\partial n}$ é a derivada direcional da solução fundamental na direção do vetor $\vec{n}$ normal ao contorno. Agora, substituindo (9.16) em (9.18), temos:

$$q^* = -k\frac{\partial}{\partial n}\left(-\frac{1}{2\pi k}\ln r\right)$$

$$q^* = \frac{1}{2\pi}\left[\frac{\partial}{\partial x}(\ln r)n_x + \frac{\partial}{\partial y}(\ln r)n_y\right] \tag{9.19}$$

Calculando primeiramente o termo $\frac{\partial}{\partial x}(\ln r)$, temos que:

$$\frac{\partial}{\partial x}(\ln r) = \frac{\partial}{\partial r}(\ln r)\frac{\partial r}{\partial x} = \frac{1}{r}\frac{\partial r}{\partial x}$$

$$r = [(x - x_d)^2 + (y - y_d)^2]^{\frac{1}{2}}$$

$$\frac{\partial r}{\partial x} = \frac{(x - x_0)}{[(x - x_d)^2 + (y - y_d)^2]^{\frac{1}{2}}} = \frac{(x - x_0)}{r}$$

e então, obtemos que:

$$\frac{\partial}{\partial x}(\ln r) = \frac{1}{r}\frac{\partial r}{\partial x} = \frac{1}{r}\frac{(x - x_d)}{r} = \frac{(x - x_d)}{r^2} \tag{9.20}$$

de forma análoga, o cálculo do termo $\frac{\partial}{\partial y}(\ln r)$ resulta em:

$$\frac{\partial}{\partial y}(\ln r) = \frac{(y - y_d)}{r^2} \tag{9.21}$$

substituindo (9.20) e (9.21) em (9.19) obtemos a solução fundamental para o fluxo de calor no contorno, dada por:

$$q^* = \frac{1}{2\pi r^2}\left[(x - x_d)n_x + (y - y_d)n_y\right] \tag{9.22}$$

11.5 EQUAÇÃO INTEGRAL DE CONTORNO A PARTIR DO MÉTODO DOS RESIDUOS PONDERADOS

Considere novamente a equação de Laplace dada por:

$$\nabla^2 T = 0$$

O objetivo agora é obter a equação integral de contorno para a equação de Laplace, a qual será posteriormente discretizada pelos elementos de contorno. Para tal, considere uma função peso $w(x, y)$, e então defina que a integral sobre o domínio A do produto da equação de Laplace com a função peso w é zero .

$$\iint_A (\nabla^2 T) w \, dA = 0 \tag{9.23}$$

Esse é um método para resolução de equações diferenciais chamado de métodos dos resíduos ponderadas.

Expandindo os termos de (9.23), temos:

$$\iint_A \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) w \, dA = 0$$

$$\iint_A \frac{\partial^2 T}{\partial x^2} w \, dA + \iint_A \frac{\partial^2 T}{\partial y^2} w \, dA = 0 \tag{9.24}$$

A parcela a esquerda da equação (9.24), referente à x, pode ser escrita integrando sobre o domínio A a derivada em x da função $f = \frac{\partial T}{\partial x} w$, ou seja:

$$\frac{\partial}{\partial x}\left(\frac{\partial T}{\partial x} w\right) = \frac{\partial^2 T}{\partial x^2} w + \frac{\partial T}{\partial x}\frac{\partial w}{\partial x}$$

Observe que a derivada foi obtida utilizando a regra do produto para derivadas. Agora, integrando ambos os lados, obtemos:

$$\iint_A \frac{\partial}{\partial x}\left(\frac{\partial T}{\partial x}w\right)\,dA = \iint_A \frac{\partial^2 T}{\partial x^2}w\,dA + \iint_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x}\,dA \tag{9.25}$$

Observe que o primeiro termo do lado direito de (9.25) é a parcela a esquerda da equação (9.24). No entanto, essa representação não é de interesse ainda, pois não apresenta integral de contorno. Assim, aplicando o teorema de Green somente do lado esquerda da equação (9.25), obtemos:

$$\iint_A \frac{\partial}{\partial x}\left(\frac{\partial T}{\partial x}w\right)\,dA = \oint_s \left(\frac{\partial T}{\partial x}w\vec{\imath}\right)\cdot n\,dS$$

$$\oint_s \left(\frac{\partial T}{\partial x}w\vec{\imath}\right)\cdot n\,dS = \oint_s \left[\left(\frac{\partial T}{\partial x}w\vec{\imath}\right)\left(n_x\vec{\imath} + n_y\vec{\jmath}\right)\right]dS = \oint_s \frac{\partial T}{\partial x}wn_x\,dS$$

ou seja,

$$\iint_A \frac{\partial}{\partial x}\left(\frac{\partial T}{\partial x}w\right)\,dA = \oint_s \frac{\partial T}{\partial x}wn_x\,dS \tag{9.26}$$

e então substituindo (9.26) em (9.25) e reorganizando os termos, podemos mostrar que:

$$\iint_A \frac{\partial^2 T}{\partial x^2}w\,dA = \oint_s \frac{\partial T}{\partial x}wn_x\,dS - \iint_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x}\,dA \tag{9.27}$$

De maneira análoga podemos mostrar que:

$$\iint_A \frac{\partial^2 T}{\partial y^2}w\,dA = \oint_s \frac{\partial T}{\partial y}wn_y\,dS - \iint_A \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}\,dA \tag{9.28}$$

substituindo (9.27) e (9.28) em (9.24), temos que:

$$\begin{aligned}\iint_A \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)w\,dA &= \iint_A \frac{\partial^2 T}{\partial x^2}w\,dA + \iint_A \frac{\partial^2 T}{\partial y^2}w\,dA \\ &= \oint_s \frac{\partial T}{\partial x}wn_x\,dS - \iint_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x}dA + \oint_s \frac{\partial T}{\partial y}wn_y\,dS - \iint_A \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}\,dA\end{aligned}$$

$$= \oint_s \left(\frac{\partial T}{\partial x} w n_x + \frac{\partial T}{\partial y} w n_y\right) dS - \iint_A \left(\frac{\partial T}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}\right) dA$$

e então:

$$\iint_A \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) w\, dA = \oint_s \frac{\partial T}{\partial n} w\, dS - \iint_A \left(\frac{\partial T}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}\right) dA \qquad (9.29)$$

Analogamente ao que foi feito anteriormente, a parcela a direita da equação (9.29), referente à x, pode ser escrita integrando sobre o domínio A a derivada em x da função $f = T\frac{\partial w}{\partial x}$, ou seja:

$$\frac{\partial}{\partial x}\left(T\frac{\partial w}{\partial x}\right) = \frac{\partial T}{\partial x}\frac{\partial w}{\partial x} + T\frac{\partial^2 w}{\partial x^2}$$

Mais uma vez, a derivada foi obtida utilizando a regra do produto para derivadas. Agora, integrando ambos os lados, obtemos:

$$\iint_A \frac{\partial}{\partial x}\left(T\frac{\partial w}{\partial x}\right) dA = \iint_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x} dA + \iint_A T\frac{\partial^2 w}{\partial x^2}\, dA \qquad (9.30)$$

O termo no meio da equação (9.30) é a parcela referente à x do lado direito da equação (9.29). Do mesmo modo que anteriormente, essa representação não é de interesse ainda, pois não apresenta integral de contorno. Assim, aplicando o teorema de Green somente do lado esquerda da equação (9.30), obtemos:

$$\iint_A \frac{\partial}{\partial x}\left(T\frac{\partial w}{\partial x}\right) dA = \oint_s \left(T\frac{\partial w}{\partial x}\vec{\imath}\right) \cdot n\, dS$$

$$\oint_s \left(T\frac{\partial w}{\partial x}\vec{\imath}\right) \cdot n\, dS = \oint_s \left[\left(T\frac{\partial w}{\partial x}\vec{\imath}\right)\left(n_x\vec{\imath} + n_y\vec{\jmath}\right)\right] dS = \oint_s T\frac{\partial w}{\partial x} n_x\, dS$$

ou seja:

$$\iint_A \frac{\partial}{\partial x}\left(T\frac{\partial w}{\partial x}\right) dA = \oint_s T\frac{\partial w}{\partial x} n_x\, dS \qquad (9.31)$$

e então substituindo (9.31) em (9.30) e reorganizando os termos, podemos mostrar que:

$$\iint_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x}\, dA = \oint_s T\frac{\partial w}{\partial x}n_x\, dS - \iint_A T\frac{\partial^2 w}{\partial x^2}\, dA \tag{9.32}$$

De maneira análoga podemos mostrar que:

$$\iint_A \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}\, dA = \oint_s T\frac{\partial w}{\partial y}n_y\, dS - \iint_A T\frac{\partial^2 w}{\partial y^2}\, dA \tag{9.33}$$

e então, de (9.33) e (9.32), temos que:

$$\begin{aligned}\iint_A \left(\frac{\partial T}{\partial x}\frac{\partial w}{\partial x} + \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}\right) dA &= \iint_A \frac{\partial T}{\partial x}\frac{\partial w}{\partial x}dA + \frac{\partial T}{\partial y}\frac{\partial w}{\partial y}dA \\ &= \oint_s T\frac{\partial w}{\partial x}n_x\, dS - \iint_A T\frac{\partial^2 w}{\partial x^2}\, dA + \oint_s T\frac{\partial w}{\partial y}n_y\, dS \\ &\quad - \iint_A T\frac{\partial^2 w}{\partial y^2}\, dA \\ &= \oint_s T\left(\frac{\partial w}{\partial x}n_x + \frac{\partial w}{\partial y}n_y\right) dS - \iint_A T\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right) dA \\ &= \oint_s T\frac{\partial w}{\partial n}dS - \iint_A \mathrm{T}\nabla^2 w\, dA \end{aligned} \tag{9.34}$$

Finalmente, substituindo (9.34) em (9.29), obtemos a equação integral de contorno dada por:

$$\iint_A \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) w\, dA = \oint_s \frac{\partial T}{\partial n}w\, dS - \oint_s T\frac{\partial w}{\partial n}dS + \iint_A \mathrm{T}\nabla^2 w\, dA = 0 \tag{9.35}$$

Observe que ainda existe um termo que apresenta integração sobre o domínio A. Para solucionar esse problema é preciso definir uma função peso $w(x, y)$ que elimine essa integral. Uma possibilidade e também a escolha mais apropriada é a própria solução fundamental [22]. Observe que se:

$$\nabla^2 w = \nabla^2 T^* = -\frac{\delta(x-d)}{k}$$

$$\iint_A \mathrm{T}\nabla^2 w \, dA = \iint_A \mathrm{T}\left(-\frac{\delta(x-d)}{k}\right) dA = -\frac{1}{k}\iint_A \mathrm{T}\delta(x-d)\, dA \qquad (9.36)$$

Considerando que o ponto fonte está no interior do domínio de integração A podemos utilizar a seguinte propriedade do delta de Dirac:

$$\iint_A \mathrm{f(x,y)}\delta(x-d)\, dA = f(d) = f(x_d, y_d)$$

Então, aplicando essa propriedade em (9.36), temos que:

$$\iint_A \mathrm{T}\nabla^2 w \, dA = -\frac{1}{k}\iint_A \mathrm{T}\delta(x-d)\, dA = -\frac{T(x_d, y_d)}{k} \qquad (9.37)$$

Observe que $\nabla^2 w = \nabla^2 T^*$ implica em $w = T^*$. Assim, substituindo (9.37) em (9.35), obtemos:

$$\oint_s \frac{\partial T}{\partial n} T^* \, dS - \oint_s T \frac{\partial T^*}{\partial n} dS - \frac{T(x_d, y_d)}{k} = 0$$

Multiplicando por k , e rearranjando os termos, temos:

$$T(x_d, y_d) = -\oint_s Tk\left(\frac{\partial T^*}{\partial n}\right) dS + \oint_s k\left(\frac{\partial T}{\partial n}\right) T^* \, dS \qquad (9.38)$$

No entanto:

$$q = -k\left(\frac{\partial T}{\partial n}\right) \qquad (9.39)$$

e então, finalmente, substituindo (9.39) em (9.38) obtemos a chamada equação integral de contorno, para o caso onde o ponto fonte se encontra dentro do domínio, dada por:

$$T(x_d, y_d) = \oint_s T q^* dS - \oint_s q T^* \, dS$$

Caso o ponto fonte se encontre no contorno do domínio, uma pequena modificação pode ser feita nesse contorno de forma a tornar o ponto fonte interno ao domínio (Figura 57).

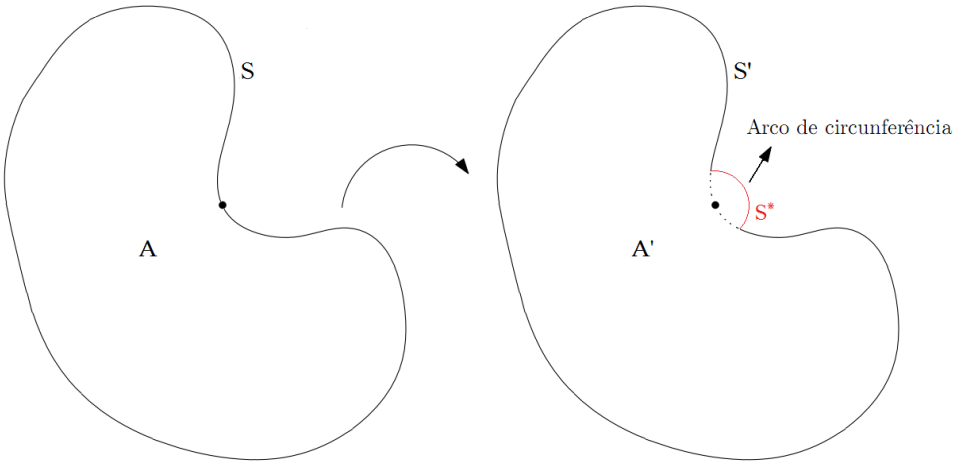

Figura 57 - Contorno modificado para incluir ponto fonte.

Assim, a equação integral de contorno pode ser escrita da seguinte forma:

$$T(x_d, y_d) = \oint_{s'} Tq^* dS - \oint_{s'} qT^* \, dS$$

$$T(x_d, y_d) = \oint_{s} Tq^* dS - \oint_{s} qT^* \, dS + \left(\oint_{s^*} Tq^* dS - \oint_{s^*} qT^* \, dS \right) \qquad (9.40)$$

Como já visto anteriormente a solução fundamental para o fluxo de calor é dada pela eq. (9.22):

$$q^* = \frac{1}{2\pi r^2} \left[(x - x_d) n_x + (y - y_d) n_y \right]$$

Substituindo (9.22) no primeiro termo dentro do parêntesis de (9.40), obtemos:

$$\oint_{s^*} Tq^* dS = \oint_{s^*} T \frac{1}{2\pi r^2} \left[(x - x_0) n_x + (y - y_0) n_y \right] dS \qquad (9.41)$$

No contorno s^* o vetor normal unitário ao contorno $\vec{n}$ é igual ao vetor unitário na direção $\vec{r}$, pois o contorno é um arco de circunferência, o qual é centrado no ponto fonte de coordenadas (x_d, y_d), desse modo:

$$\vec{r} = (x - x_d)\vec{\imath} + (y - y_d)\vec{\jmath}$$

$$\vec{n} = \frac{\vec{r}}{\|r\|} = \frac{1}{r} \left[(x - x_d)\vec{\imath} + (y - y_d)\vec{\jmath} \right]$$

Definindo $r_x = (x - x_d)$ e $r_y = (y - y_d)$, temos:

$$\vec{n} = \frac{\vec{r}}{\|r\|} = \frac{r_x}{r}\vec{\imath} + \frac{r_y}{r}\vec{\jmath} = n_x\vec{\imath} + n_y\vec{\jmath}$$

ou seja:

$$n_x = \frac{r_x}{r};\ n_y = \frac{r_y}{r} \tag{9.42}$$

Substituindo (9.42) em (9.41) e utilizando coordenadas polares; podemos mostra que:

$$\begin{aligned}
\oint_{s^*} Tq^* dS &= \oint_{\theta_1}^{\theta_2} T\frac{1}{2\pi r^2}\left[r_x\frac{r_x}{r} + r_y\frac{r_y}{r}\right] r\, d\theta \\
&= \oint_{\theta_1}^{\theta_2} \frac{T}{2\pi r^2}\left[\frac{{r_x}^2 + {r_y}^2}{r}\right] r\, d\theta \\
&= \oint_{\theta_1}^{\theta_2} \frac{T}{2\pi}\left[\frac{{r_x}^2 + {r_y}^2}{r^2}\right] d\theta \\
&= \oint_{\theta_1}^{\theta_2} \frac{T}{2\pi}[\vec{n}\cdot\vec{n}]\, d\theta\ ;\ \vec{n}\cdot\vec{n} = 1 \\
\oint_{s^*} Tq^* dS &= \oint_{\theta_1}^{\theta_2} \frac{T}{2\pi} d\theta
\end{aligned} \tag{9.43}$$

Observe que o vetor $\vec{n}$ é um vetor unitário, e o produto escalar desse vetor com ele mesmo tem sempre valor unitário. Se considerarmos r suficientemente pequeno, ou seja, quando $r \to 0$, podemos considerar que a temperatura T é constante e igual a temperatura no ponto fonte $T(x_d, y_d)$, deste modo podemos escrever:

$$\oint_{s^*} Tq^* dS = \oint_{\theta_1}^{\theta_2} \frac{T}{2\pi} d\theta = \frac{T(x_d, y_d)}{2\pi}\oint_{\theta_1}^{\theta_2} d\theta$$

$$\oint_{s^*} Tq^* dS = \frac{T(x_d, y_d)(\theta_2 - \theta_1)}{2\pi} \tag{9.44}$$

O mesmo tipo de análise deve agora ser feito para o segundo termo dentro do parêntesis da eq. (9.40). Temos que a solução fundamental para a distribuição de temperatura é dada pela eq. (9.16):

$$T^* = -\frac{1}{2\pi k}\ln r$$

Substituindo (9.16) no segundo termo dentro do parêntesis da eq. (9.40) e escrevendo em coordenadas polares, podemos então fazer:

$$\begin{aligned}\oint_{s^*} qT^* dS &= \oint_{s^*} q\left(-\frac{1}{2\pi k}\ln r\right) dS \\ &= -\frac{1}{2\pi k}\oint_{s^*} q(\ln r)\, r\, d\theta \\ &= -\frac{r\ln r}{2\pi k}\oint_{s^*} q\, d\theta\end{aligned}$$

Novamente, considerando r suficientemente pequeno, ou seja, quando $r \to 0$, temos que o termo $r\ln r \to 0$. Assim, independente da integral do lado direito da equação, o lado esquerdo será sempre igual à zero, ou seja:

$$\oint_{s^*} qT^* dS = 0 \qquad (9.45)$$

Substituindo as equações (9.44) e (9.45) na eq. (9.40), obtemos:

$$T(x_d, y_d) = \oint_s Tq^* dS - \oint_s qT^* dS + \frac{T(x_d, y_d)(\theta_2 - \theta_1)}{2\pi}$$

$$T(x_d, y_d)\left[1 - \frac{(\theta_2 - \theta_1)}{2\pi}\right] = \oint_s Tq^* dS - \oint_s qT^* dS \qquad (9.46)$$

Considerando somente o lado direito da equação, temos:

$$T(x_d, y_d)\left[1 - \frac{(\theta_2 - \theta_1)}{2\pi}\right] = T(x_d, y_d)\left[\frac{2\pi - (\theta_2 - \theta_1)}{2\pi}\right] = \frac{\theta_{int}}{2\pi}T(x_d, y_d) \qquad (9.47)$$

onde θ_{int} é o ângulo replementar de $(\theta_1 + \theta_2)$, ou seja, que juntamente com θ_1e θ_2 somam em 360º (Figura 58).

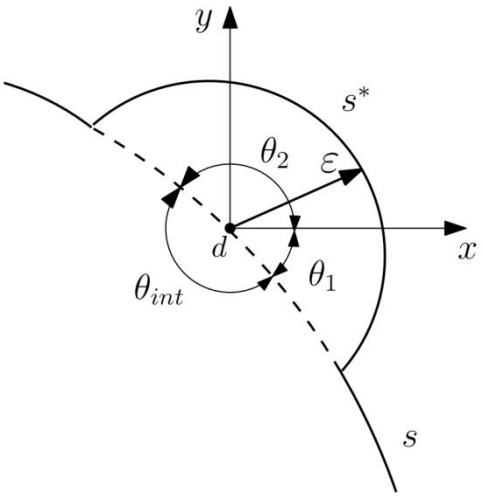

Figura 58 - Ângulo interno θ_{int}

Substituindo a eq.(9.47) em (9.46) obtemos a equação integral de contorno para o caso onde o ponto fonte pertence ao contorno.

$$\frac{\theta_{int}}{2\pi}T(x_d, y_d) = \oint_s Tq^* dS - \oint_s qT^* dS$$

Finalmente, para o último caso onde a fonte não pertence ao domínio, temos que pela propriedade (9.4) do delta de Dirac que:

$$\oint_s Tq^* dS - \oint_s qT^* dS = 0$$

Deste modo, podemos definir a forma geral da equação integral de contorno:

$$cT(x_d, y_d) = \oint_s Tq^* dS - \oint_s qT^* dS \qquad (9.48)$$

onde:

$$c = \begin{cases} 1 & se\ (x_d, y_d) \in ao\ dominio\ A \\ \dfrac{\theta_{int}}{2\pi} & se\ (x_d, y_d) \in ao\ contorno\ S \\ 0 & se\ (x_d, y_d) \notin ao\ dominio\ A\ ou\ contorno\ S \end{cases}$$

Quando o ponto fonte se encontra em um seguimento suave do contorno, ou seja, quando o mesmo não está em um vértice, temos que:

$$\frac{\theta_{int}}{2\pi} = \frac{\pi}{2\pi} = \frac{1}{2}$$

11.6 DISCRETIZACAO DA EQUACAO INTEGRAL DE CONTORNO

A formulação de problemas de engenharia pelo MEC transformam as equações diferenciais que descrevem esse problema em equações integrais de contorno. Deste modo, não é necessário a discretização de todo o domínio A, mas somente do contorno S. Assim, a discretização da equação integral de contorno é feita discretizando somente o contorno S em diferentes segmentos S_j (Figura 59), os quais serão futuramente aproximados por elementos de contorno:

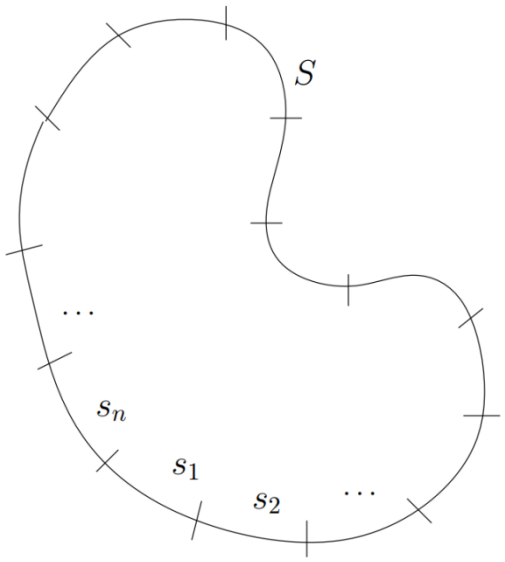

Figura 59 - Contorno S discretizado em diferentes seguimentos S_n.

De modo que:

$$S = S_1 + S_2 + \cdots + S_n$$

e então a integral de contorno (9.48) toma a forma:

$$cT(x_d, y_d) = \sum_{j=1}^{n} \int_{S_j} Tq^* dS - \sum_{j=1}^{n} \int_{S_j} qT^* dS$$

11.7 CÁLCULO DA TEMPERATURA E DO FLUXO EM PONTOS INTERNOS

Como dito anteriormente na introdução sobre MEC, o método permite analisar a função de campo e suas derivadas em qualquer ponto do corpo. No caso da condução de calor é possível obter a temperatura e o fluxo de calor em qualquer ponto interno do domínio.

Uma vez calculados os valores de T e q no contorno é possível determinar o valor de T em um ponto (x, y) qualquer interno ao domínio fazendo desse ponto o ponto fonte, ou seja, $(x, y) = (x_d, y_d)$, e então utilizando a equação integral de contorno (9.48), onde $c = 1$, uma vez que o ponto pertence ao domínio, dada por:

$$T(x,y) = \oint_s Tq^* dS - \oint_s qT^* dS$$

Observe que, uma vez obtido T e q, as integrais dependem somente das soluções fundamentais, as quais, por sua vez, dependem somente da posição do ponto fonte uma vez que o contorno é o mesmo, mas a posição do ponto fonte é a posição do ponto interno onde se quer calcular a solução, a qual também é conhecida. Assim, é possível efetuar a integração e obter a temperatura nesse ponto.

No caso do fluxo de calor, para um ponto interno qualquer também é utilizado o mesmo artificio de fazer desse ponto o ponto fonte, mas nesse caso é necessário derivar a equação (9.48) em relação às coordenadas do ponto fonte, pois de (9.2) temos que:

$$\dot{\vec{q}} = -k\vec{\nabla}T$$

para a coordenada x_d, temos:

$$\frac{\partial T(x,y)}{\partial x_d} = \frac{\partial}{\partial x_d}\left(\oint_s Tq^* dS - \oint_s qT^* dS\right)$$

$$\frac{\partial T(x,y)}{\partial x_d} = \left(\oint_s T\frac{\partial q^*}{\partial x_d} dS - \oint_s q\frac{\partial T^*}{\partial x_d} dS\right) \qquad (9.49)$$

Para calcular o primeiro termo dentro do parêntesis de (9.49), é preciso considerar a solução fundamental para o fluxo de calor, dada por (9.22) e também lembrar que $r_x = (x - x_d)$ e $r_y = (y - y_d)$, desse modo, temos que:

$$\frac{\partial q^*}{\partial x_d} = \frac{\partial}{\partial x_d}\left[\frac{1}{2\pi r^2}\left(r_x n_x + r_y n_y\right)\right] = \frac{1}{2\pi}\frac{\partial}{\partial x_d}\left[\frac{1}{r^2}\left(r_x n_x + r_y n_y\right)\right]$$

e então, pela regra do produto para derivadas:

$$\frac{\partial q^*}{\partial x_d} = \frac{1}{2\pi}\left[\left(r_x n_x + r_y n_y\right)\frac{\partial}{\partial x_d}\left(\frac{1}{r^2}\right) + \frac{1}{r^2}\frac{\partial}{\partial x_d}\left(r_x n_x + r_y n_y\right)\right]$$

$$\frac{\partial q^*}{\partial x_d} = \frac{1}{2\pi}\left[\left(r_x n_x + r_y n_y\right)\boldsymbol{I} + \frac{1}{r^2}\boldsymbol{II}\right]; \boldsymbol{I} = \frac{\partial}{\partial x_d}\left(\frac{1}{r^2}\right) \; e \; \boldsymbol{II} = \frac{\partial}{\partial x_d}\left(r_x n_x + r_y n_y\right) \tag{9.50}$$

Resolvendo ***I*** pela regra da cadeia, onde $r = \left({r_x}^2 + {r_y}^2\right)^{\frac{1}{2}}$, obtemos:

$$\frac{\partial}{\partial x_d}\left(\frac{1}{r^2}\right) = \frac{\partial}{\partial x_d}\left({r_x}^2 + {r_y}^2\right)^{-1} = -\left({r_x}^2 + {r_y}^2\right)^{-2} 2r_x(-1)$$

$$\frac{\partial}{\partial x_d}\left(\frac{1}{r^2}\right) = \frac{2r_x}{r^4} \tag{9.51}$$

Resolvendo ***II*** pela regra do produto, podemos mostrar que:

$$\frac{\partial}{\partial x_d}\left(r_x n_x + r_y n_y\right) = n_x \frac{\partial}{\partial x_d}(r_x) + r_x \frac{\partial}{\partial x_d}(n_x) + n_y \frac{\partial}{\partial x_d}\left(r_y\right) + r_y \frac{\partial}{\partial x_d}\left(n_y\right)$$

$$\frac{\partial}{\partial x_d}\left(r_x n_x + r_y n_y\right) = n_x \frac{\partial}{\partial x_d}(r_x) + 0 + 0 + 0$$

$$\frac{\partial}{\partial x_d}\left(r_x n_x + r_y n_y\right) = -n_x \tag{9.52}$$

Os três últimos termos da derivada são nulos pois o vetor n normal ao contorno não depende da posição do ponto fonte, sendo então n_x e n_y constantes em um ponto qualquer do contorno e também porque r_y não é função de x_d.

Substituindo então (9.51) e (9.52) em (9.50) obtemos:

$$\frac{\partial q^*}{\partial x_d} = \frac{1}{2\pi}\left[\left(r_x n_x + r_y n_y\right)\frac{2r_x}{r^4} + \frac{1}{r^2}(-n_x)\right]$$

$$= \frac{1}{2\pi r^4}\left(2r_x r_x n_x + 2r_x r_y n_y - r^2 n_x\right)$$

$$= \frac{1}{2\pi r^4}\left(2{r_x}^2 n_x + 2r_x r_y n_y - r^2 n_x\right)$$

$$= \frac{1}{2\pi r^4}\left[2{r_x}^2 n_x + 2r_x r_y n_y - \left({r_x}^2 + {r_y}^2\right)n_x\right]$$

$$= \frac{1}{2\pi r^4}\left(2{r_x}^2 n_x + 2r_x r_y n_y - {r_x}^2 n_x - {r_y}^2 n_x\right)$$

$$= \frac{1}{2\pi r^4}\left({r_x}^2 n_x - {r_y}^2 n_x + 2r_x r_y n_y\right)$$

$$\frac{\partial q^*}{\partial x_d} = \frac{1}{2\pi r^4}\left[n_x\left({r_x}^2 - {r_y}^2\right) + 2r_x r_y n_y\right]$$

de forma análoga é possível mostrar que:

$$\frac{\partial q^*}{\partial y_d} = \frac{1}{2\pi r^4}\left[n_y\left(-{r_x}^2 + {r_y}^2\right) + 2r_x r_y n_y\right]$$

Agora, calculando o segundo termo dentro do parêntesis de (9.49), considerando a solução fundamental para o campo de temperatura dada por (9.16), temos que:

$$\frac{\partial T^*}{\partial x_d} = \frac{\partial}{\partial x_d}\left(-\frac{1}{2\pi k}\ln r\right) = -\frac{1}{2\pi k}\frac{\partial}{\partial x_d}(\ln r)$$

a derivada $\frac{\partial}{\partial x}(\ln r)$ é dada por (9.20), dessa forma obtemos:

$$\frac{\partial}{\partial x}(\ln r) = -\frac{\partial}{\partial x_d}(\ln r) = \frac{(x - x_d)}{r^2}$$

$$\frac{\partial T^*}{\partial x_d} = -\frac{1}{2\pi k}\left[-\frac{(x - x_d)}{r^2}\right] = \frac{r_x}{2\pi k r^2}$$

de forma análoga é possível mostrar que:

$$\frac{\partial T^*}{\partial y_d} = \frac{r_y}{2\pi k r^2}$$

Agora que já são conhecidas as derivadas das equações fundamentais é possível calcular o fluxo de calor para um ponto interno qualquer.

11.8 ELEMENTOS DE CONTORNO

Uma vez discretizado o contorno S a solução da equação integral de contorno depende do tipo de elemento de contorno definido para descrever os diferentes segmentos S_j.

Os elementos de contorno são elementos matemáticos criados por funções chamadas funções de forma que somente aproximam S_j, ou seja, em geral elas não são uma representação exata de S_j.

Elementos de maior simplicidade, como elementos constantes e lineares possibilitam encontrar em alguns casos uma solução exata para a integral. No entanto, a representação do contorno pode ser pobre para formas de geometria complexa, o que induz um certo nível de erro na solução. Por outro lado, elementos mais complexos representam de forma mais fiel o contorno, mas devem ser integrados numericamente, requerendo um algoritmo com maior complexidade e apresentando certo nível de erro devido a integral aproximada.

Esse trabalho tem foco no desenvolvimento de um algoritmo que resolva de forma eficiente as matrizes resultantes da aplicação do MEC. Assim, não há interesse no tipo de elemento que originou essa matriz. Por isso discutiremos somente sobre os elementos de contorno lineares contínuos.

11.8.1 FUNÇÕES DE FORMA

Funções de forma ou funções interpoladoras [23], são funções que servem de base para a representação de outras funções. Assim como qualquer vetor em um espaço vetorial pode ser representado como uma combinação linear dos vetores bases, qualquer função continua no espaço função, pode ser representada com uma combinação linear de funções bases.

As funções de forma N_i apresentam as seguintes propriedades fundamentais:

- Assumem valor unitário para $x = x_i$;
- Assumem valor nulo para os nós restantes.

O método mais utilizado para se obter uma função base de forma polinomial é a fórmula de interpolação de Lagrange, dada por:

$$N_i = \prod_{\substack{k=1 \\ (k \neq i)}}^{n} \frac{(x - x_k)}{(x_i - x_k)}$$

Observe que N_i é função somente de x.

Quando a função de forma é do tipo polinomial, é desejável manter o menor grau de polinômio possível. Deste modo, para se representar de forma exata um polinômio de grau j são necessários somente $n = j + 1$ pontos.

Considere por exemplo à função linear $y = 3x + 2$, sendo uma função linear são necessários dois pontos, $x_1 = 1$ e $x_2 = 2$. Assim, temos:

$$N_2 = \frac{(x - x_1)}{(x_2 - x_1)} = \frac{(x - 1)}{(2 - 1)} = x - 1$$

$$N_1 = \frac{(x - x_2)}{(x_1 - x_2)} = \frac{(x - 2)}{(1 - 2)} = -x + 2$$

observe que $N_2(x_1) = 0$, $N_2(x_2) = 1$, $N_1(x_1) = 1$ e $N_1(x_2) = 0$, satisfazendo as propriedades fundamentais da função de forma. De fato, as funções de forma lineares N_1 e N_2 apresentam o comportamento dado pela Figura 60, onde é possível notar facilmente que elas satisfazem as propriedades fundamentais.

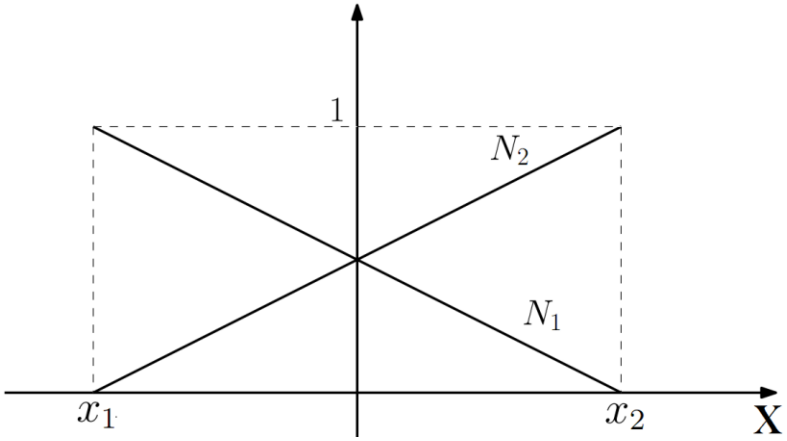

Figura 60 - Funções de forma lineares N_1 e N_2.

Se N_1 e N_2 formam uma base para o espaço função das funções lineares em $\mathbb{R}^2$, então $y = 3x + 2$ é uma combinação linear de N_1 e N_2, observe que para x temos que:

$$\begin{aligned} x &= N_1 x_1 + N_2 x_2 \\ &= (-x + 2)1 + (x - 1)2 \\ &= x + 2 + 2x - 2 \\ x &= x \end{aligned}$$

embora isso indique que as funções de forma funcionam, esse resultado não é interessante pois não sabemos se as funções podem de fato representar y.

Análogo ao que foi feito para x, podemos obter y conhecendo $y_1 = y(x_1) = 5$ e $y_2 = y(x_2) = 8$, e então de fato podemos mostrar que:

$$\begin{aligned} y &= N_1 y_1 + N_2 y_2 \\ &= (-x+2)5 + (x-1)8 \\ &= -5x + 10 + 8x - 8 \\ y &= 3x + 2 \end{aligned}$$

Desse modo, N_1 e N_2 formam de fato uma base para funções lineares em $\mathbb{R}^2$, ou seja, para qualquer polinômio de 1º grau no espaço $\mathbb{R}^2$.

Considere agora outro exemplo, a função quadrática $y = 4x^2 - 3x + 2$. Sendo uma função quadrática, são necessários três pontos, $x_1 = 1$ e $x_2 = 2$ $x_2 = 3$. Assim, temos:

$$N_3 = \frac{(x-x_1)}{(x_3-x_1)}\frac{(x-x_2)}{(x_3-x_2)} = \frac{(x-1)}{(3-1)}\frac{(x-2)}{(3-2)} = \frac{(x-1)(x-2)}{2}$$

$$N_2 = \frac{(x-x_1)}{(x_2-x_1)}\frac{(x-x_3)}{(x_2-x_3)} = \frac{(x-1)}{(2-1)}\frac{(x-3)}{(2-3)} = (x-1)(-x+3)$$

$$N_1 = \frac{(x-x_2)}{(x_1-x_2)}\frac{(x-x_3)}{(x_1-x_3)} = \frac{(x-2)}{(1-2)}\frac{(x-3)}{(1-3)} = \frac{(x-2)(x-3)}{2}$$

Observe que N_1, N_2 e N_3 novamente satisfazem as propriedades fundamentais das funções de forma. Seus comportamentos podem ser mais bem observados pela Figura 61, onde é facilmente notado que elas satisfazem as propriedades fundamentais.

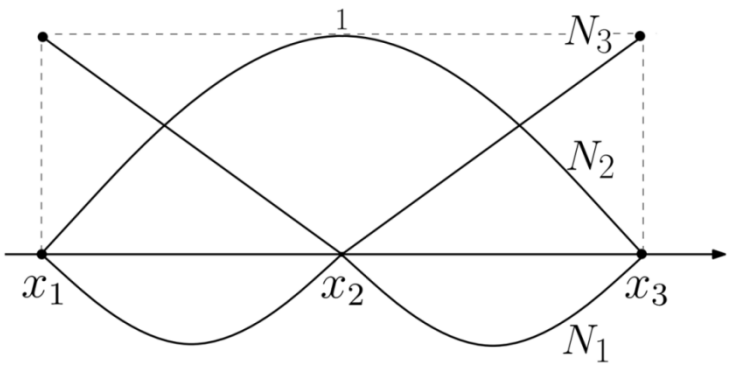

Figura 61 - Funções de forma quadráticas N_1, N_2 e N_3

Sabendo que $y_1 = y(x_1) = 3$, $y_2 = y(x_2) = 12$ e $y_3 = y(x_3) = 29$. Podemos escrever então:

$$
\begin{aligned}
y &= N_1 y_1 + N_2 y_2 + N_3 y_3 \\
&= \left[\frac{(x-2)(x-3)}{2}\right] 3 + [(x-1)(-x+3)]12 + \left[\frac{(x-1)(x-2)}{2}\right] 29 \\
&= \left(\frac{1}{2}x^2 - \frac{5}{2}x + 3\right) 3 + (-x^2 + 4x - 3)12 + \left(\frac{1}{2}x^2 - \frac{3}{2}x + 1\right) 29 \\
&= \frac{3}{2}x^2 - \frac{15}{2}x + 9 - 12x^2 + 48x - 36 + \frac{29}{2}x^2 - \frac{87}{2}x + 29 \\
&= \left(\frac{3}{2} - 12 + \frac{29}{2}\right)x^2 + \left(-\frac{15}{2} + 48 - \frac{87}{2}\right)x + (9 - 36 + 29) \\
y &= 4x^2 - 3x + 2
\end{aligned}
$$

Assim, N_1, N_2 e N_3 formam de fato uma base para funções quadráticas em $\mathbb{R}^2$, ou seja, para qualquer polinômio de 2º grau no espaço $\mathbb{R}^2$.

Existem vários outros tipos de funções de forma, mesmo dentre as do tipo polinomial, como por exemplo, as funções de interpolação Hermitiana (Figura 62). Esse tipo de interpolação não somente atende aos valores nodais, mas também atende as derivadas das funções nos nós. Esse tipo de formulação é principalmente utilizado em problemas considerando rotação, como por exemplo, em problemas envolvendo vigas.

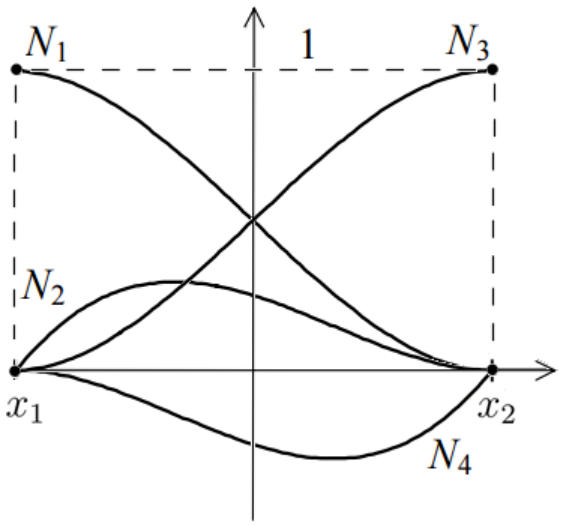

Figura 62 - Funções de forma Hermitiana N_1, N_2, N_3 e N_4 para elemento definido por dois nós.

No entanto devido às condições para atender as derivadas além dos valores nodais, são necessárias quatro funções de forma para aproximar um elemento definido por dois nós.

Outras importantes propriedades das funções de forma, as quais são consequências das propriedades fundamentais, são:

$$\sum_{k=1}^{n} N_K = 1$$

e consequentemente:

$$\sum_{k=1}^{n} xN_K = x$$

Para o exemplo linear que foi dado, temos que:

$$\sum_{k=1}^{n} N_K = N_1 + N_2 = (-x + 2) + (x - 1) = 1$$

$$\sum_{k=1}^{n} xN_K = xN_1 + xN_2 = x(N_1 + N_2) = x$$

Para o exemplo quadrático, temos:

$$\begin{aligned}\sum_{k=1}^{n} N_K = N_1 + N_2 + N_3 \quad &= \left(\frac{1}{2}x^2 - \frac{5}{2}x + 3\right) + (-x^2 + 4x - 3) + \left(\frac{1}{2}x^2 - \frac{3}{2}x + 1\right) \\ &= \left(\frac{1}{2} - 1 + \frac{1}{2}\right)x^2 + \left(-\frac{5}{2} + 4 - \frac{3}{2}\right)x + (3 - 3 + 1) \\ &= 0x^2 + 0x + 1 = 1\end{aligned}$$

Essas propriedades servem como um teste para validar funções de forma, ou para encontra a última função de forma quando já se conhece as outras.

11.8.2 ELEMENTOS DE CONTORNO LINEARES CONTÍNUOS

A discretização do contorno utilizando elementos lineares contínuos aproxima o segmento S_j do contorno por uma função linear, ou seja, um polinômio de 1º grau. Para se definir uma interpolação linear, são necessários dois pontos, nós, do

segmento S_j. Cada elemento de contorno Γ_j é definido de forma que cada nó se encontre em uma das extremidades do elemento. Dessa forma, dois elementos diferentes podem ser definidos com somente três pontos, onde um deles é compartilhado entre ambos os elementos (Figura 63).

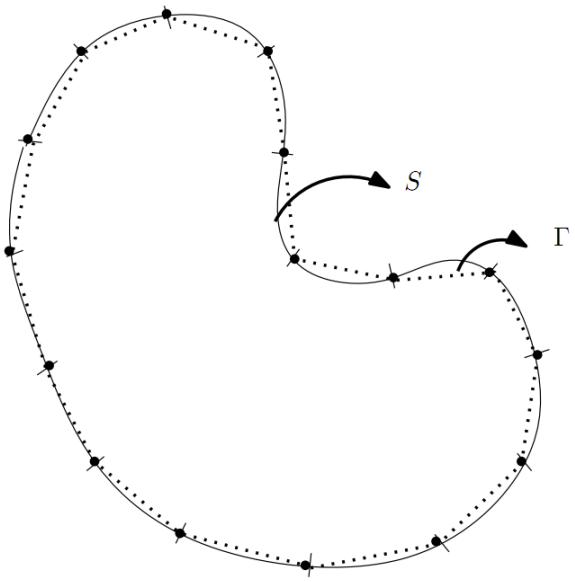

Figura 63 - Contorno S aproximado pelo contorno Γ formado por elementos lineares contínuos Γ_j.

Essa formulação é uma formulação isoparamétrica, ou seja, onde as mesmas funções de forma que são utilizadas para aproximar o contorno, são utilizadas para se aproximar as quantidades físicas, no caso da condução de calor, a temperatura e o fluxo de calor.

Quando o contorno S é aproximado por elementos de contorno Γ_j, a equação discretizada da integral de contorno é dada por:

$$cT(x_d, y_d) = \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} Tq^* d\Gamma - \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} qT^* d\Gamma \qquad (9.53)$$

onde n_{elem} é a quantidade de elementos de contorno utilizados para discretrizar o contorno S. Observe que agora não somente as coordenadas (x, y) de um ponto do contorno variam linearmente ao longo de um elemento Γ_j mas também as quantidades físicas T e q variam linearmente ao longo do elemento.

Assim, podemos não somente escrever:

$$x = N_1 x_1 + N_2 x_2 \tag{9.54}$$

$$y = N_1 y_1 + N_2 y_2$$

mas também:

$$T = N_1 T_1 + N_2 T_2 \tag{9.55}$$

$$q = N_1 q_1 + N_2 q_2 \tag{9.56}$$

onde T_1 e T_2 são as temperaturas no nó 1 de coordenada (x_1, y_1) e no nó 2 de coordenada (x_2, y_2), q_1 e q_2 são os fluxos de calor nesses nós e N_1 e N_2 são duas funções de forma lineares.

Podemos escrever T e q na forma matricial, obtendo:

$$T = [N_1 \quad N_2] \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}$$

$$q = [N_1 \quad N_2] \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}$$

Agora, é possível escrever a equação integral discretizada (9.53) na forma:

$$cT(x_d, y_d) = \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} [N_1 \quad N_2]_j \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_j q^* d\Gamma - \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} T^* [N_1 \quad N_2]_j \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_j d\Gamma$$

Como T_1, T_2, q_1 e q_2 são valores nodais e portanto constantes, podemos reescrever a equação acima na forma:

$$cT(x_d, y_d) = \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} [N_1 \quad N_2]_j q^* d\Gamma \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_j - \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} T^* [N_1 \quad N_2]_j d\Gamma \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_j$$

e então:

$$cT(x_d, y_d) = \sum_{j=1}^{n_{elem}} [h_1 \quad h_2]_j \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_j - [g_1 \quad g_2]_j \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_j \tag{9.57}$$

onde:

$$h_1 = \int_{\Gamma_j} N_1 q^* d\Gamma\,; \quad h_2 = \int_{\Gamma_j} N_2 q^* d\Gamma\,; \quad g_1 = \int_{\Gamma_j} N_1 T^* d\Gamma; \quad g_2 = \int_{\Gamma_j} N_2 T^* d\Gamma$$

Considere como exemplo o problema plano de condução de calor dado por uma placa retangular isolada nas partes superior e inferior e que apresenta temperatura $T = 0$ na lateral esquerda é $T = 1$ na lateral direita. O contorno da placa retangular foi discretizada em quatro elementos lineares contínuos, onde cada nó corresponde a um vértice e cada elemento a uma das arestas. (Figura 64)

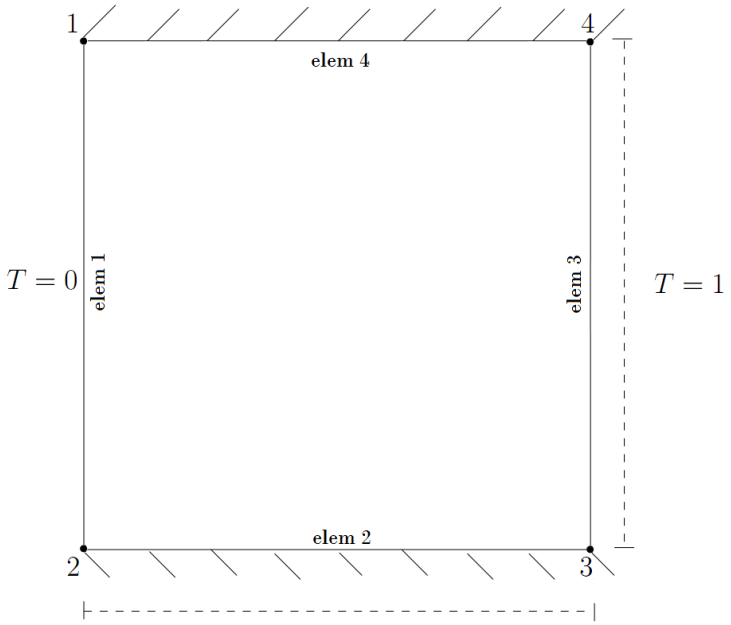

Figura 64 - Problema de condução de calor ao longo de uma placa retangular.

Os elementos são definidos pelos nós:

Tabela 22 - Definição dos elementos.

Elemento	Nó *i*	Nó *j*
elem 1	1	2
elem 2	2	3
elem 3	3	4
elem 4	4	1

e as variáveis em cada nó são:

Tabela 23 - Variáveis nos nós.

Nó	Variáveis Conhecidas		Variáveis Desconhecidas
1	$\overline{T_1}$	$\overline{q_1^a}$	q_1^d
2	$\overline{T_2}$	$\overline{q_2^d}$	q_2^a
3	$\overline{T_3}$	$\overline{q_3^a}$	q_3^d
4	$\overline{T_4}$	$\overline{q_4^d}$	q_4^a

Daqui em diante será utilizada uma barra horizontal sobre as variáveis conhecidas, com o único propósito de ser possível identificar essas quando escrito juntamente com as variáveis desconhecidas.

Observe que a temperatura é sempre continua nos nós, mas o fluxo de calor não, pois o fluxo de calor antes do nó q_j^a não é necessariamente igual ao fluxo de calor depois do nó q_j^d. Isso fica claro nesse exemplo. Observe que o nó 1 se encontra no vértice entre um lado isolado e outro lado com temperatura $T = 1$, no lado isolado, ou seja, antes do nó temos que $\overline{q_1^a} = 0$, pois uma superfície isolada não permite o fluxo de calor. Isso não acontece para o lado não isolado com temperatura $T = 1$, o que se encontra depois do nó, onde é esperado que $q_1^d \neq 0$. Nesse exemplo, isso ocorre em todos os nós, pois todos se encontram entre um lado isolado e outro não isolado.

Considerando o ponto fonte nó 1, podemos escrever a equação integral discretizada (9.57) como:

$$cT_1 = [h_1 \quad h_2]_1 \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_1 + [h_1 \quad h_2]_2 \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_2 + [h_1 \quad h_2]_3 \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_3 + [h_1 \quad h_2]_3 \begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_3$$

$$-[g_1 \quad g_2]_1 \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_1 - [g_1 \quad g_2]_2 \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_2 - [g_1 \quad g_2]_3 \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_3 - [g_1 \quad g_2]_4 \begin{bmatrix} q_1 \\ q_2 \end{bmatrix}_4$$

Essa notação, no entanto, não considera o número global de cada nó. Para entender melhor, observe que:

$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix}_3 = \begin{bmatrix} \overline{T_3} \\ \overline{T_4} \end{bmatrix}$$

ou seja, a temperatura T_1 e T_2 do elemento 3 são na verdade as temperaturas globais $\overline{T_3}$ *e* $\overline{T_4}$. Assim, escrevendo a equação integral discretizada na forma global obtemos:

$$c\overline{T_1} = [h_{11} \quad h_{21}]\begin{bmatrix}\overline{T_1}\\ \overline{T_2}\end{bmatrix} + [h_{12} \quad h_{22}]\begin{bmatrix}\overline{T_2}\\ \overline{T_3}\end{bmatrix} + [h_{13} \quad h_{23}]\begin{bmatrix}\overline{T_3}\\ \overline{T_4}\end{bmatrix} + [h_{14} \quad h_{24}]\begin{bmatrix}\overline{T_4}\\ \overline{T_1}\end{bmatrix}$$

$$-[g_{11} \quad g_{21}]\begin{bmatrix}q_1^d\\ q_2^a\end{bmatrix} - [g_{12} \quad g_{22}]\begin{bmatrix}\overline{q_2^d}\\ \overline{q_3^a}\end{bmatrix} - [g_{13} \quad g_{23}]\begin{bmatrix}q_3^d\\ q_4^a\end{bmatrix} - [g_{14} \quad g_{24}]\begin{bmatrix}\overline{q_4^d}\\ \overline{q_1^a}\end{bmatrix}$$

Reescrevendo essa equação, de forma a somar os elemento comuns a uma mesma temperatura T_j, temos:

$$c\overline{T_1} = (h_{11} + h_{24})\overline{T_1} + (h_{21} + h_{12})\overline{T_2} + (h_{22} + h_{13})\overline{T_3} + (h_{23} + h_{14})\overline{T_4}$$

$$-g_{11}q_1^d - g_{21}q_2^a - g_{12}\overline{q_2^d} - g_{22}\overline{q_3^a} - g_{13}q_3^d - g_{23}q_4^a - g_{14}\overline{q_4^d} - g_{24}\overline{q_1^a}$$

E então, em função dos índices das matrizes $\boldsymbol{G}$ e $\boldsymbol{H}$, podemos escrever:

$$c\overline{T_1} = H_{11}\overline{T_1} + H_{12}\overline{T_2} + H_{13}\overline{T_3} + H_{14}\overline{T_4}$$

$$-G_{11}q_1^d - G_{12}q_2^a - G_{13}\overline{q_2^d} - G_{14}\overline{q_3^a} - G_{15}q_3^d - G_{16}q_4^a - G_{17}\overline{q_4^d} - G_{18}\overline{q_1^a}$$

Definindo:

$$\begin{cases} H_{ij} & se\ i \neq j \\ -cT_i + H_{ij} & se\ i = j \end{cases}$$

podemos reescrever como:

$$H_{11}\overline{T_1} + H_{12}\overline{T_2} + H_{13}\overline{T_3} + H_{14}\overline{T_4} =$$

$$= G_{11}q_1^d + G_{12}q_2^a + G_{13}\overline{q_2^d} + G_{14}\overline{q_3^a} + G_{15}q_3^d + G_{16}q_4^a + G_{17}\overline{q_4^d} + G_{18}\overline{q_1^a}$$

e então, na forma matricial, temos:

$$[H_{11} \quad H_{12} \quad H_{13} \quad H_{14}]\begin{bmatrix}\overline{T_1}\\ \overline{T_2}\\ \overline{T_3}\\ \overline{T_4}\end{bmatrix} = [G_{11} \quad G_{12} \quad \cdots \quad G_{17} \quad G_{18}]\begin{bmatrix}q_1^d\\ q_2^a\\ \vdots\\ \overline{q_4^d}\\ \overline{q_1^a}\end{bmatrix}$$

Nesse momento há somente uma equação e quatro variáveis desconhecidas. Para obter o sistema completo de equações é necessário encontrar a equação integral

discretizada para o ponto fonte localizado em todos os nós possíveis, ou seja, colocando o ponto fonte em cada nó. No nosso caso, foi obtida uma única equação considerando o ponto fonte no nó 1. Deste modo, para obter as outras três equações necessárias, é preciso colocar o ponto fonte nos três nós restantes, resultando na equação matricial completa dada por:

$$\begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{21} & H_{22} & H_{23} & H_{24} \\ H_{31} & H_{32} & H_{33} & H_{34} \\ H_{41} & H_{42} & H_{43} & H_{44} \end{bmatrix} \begin{bmatrix} \overline{T_1} \\ \overline{T_2} \\ \overline{T_3} \\ \overline{T_4} \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} & \cdots & G_{17} & G_{18} \\ G_{21} & G_{22} & \cdots & G_{27} & G_{28} \\ G_{31} & G_{32} & \cdots & G_{87} & G_{48} \\ G_{41} & G_{42} & \cdots & G_{47} & G_{48} \end{bmatrix} \begin{bmatrix} q_1^d \\ q_2^a \\ \vdots \\ \overline{q_4^d} \\ \overline{q_1^a} \end{bmatrix}$$

$$\boldsymbol{H}T = \boldsymbol{G}q$$

Há ainda uma última etapa necessária. É preciso separar as variáveis conhecidas das desconhecidas. Para tal, as matrizes são manipulas de forma que as variáveis desconhecidas fiquem a esquerda e as conhecidas a direita da igualdade, obtendo então:

$$\begin{bmatrix} G_{11} & G_{12} & G_{15} & G_{16} \\ G_{21} & G_{22} & G_{25} & G_{26} \\ G_{31} & G_{32} & G_{35} & G_{36} \\ G_{41} & G_{42} & G_{45} & G_{46} \end{bmatrix} \begin{bmatrix} q_1^d \\ q_2^a \\ q_3^d \\ q_4^a \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & -G_{13} & -G_{14} & H_{13} & H_{14} & -G_{17} & -G_{18} \\ H_{21} & H_{22} & -G_{23} & -G_{24} & H_{23} & H_{24} & -G_{27} & -G_{28} \\ H_{31} & H_{32} & -G_{33} & -G_{34} & H_{33} & H_{34} & -G_{37} & -G_{38} \\ H_{41} & H_{42} & -G_{43} & -G_{44} & H_{43} & H_{44} & -G_{47} & -G_{48} \end{bmatrix} \begin{bmatrix} \overline{T_1} \\ \overline{T_2} \\ \overline{q_2^d} \\ \overline{q_3^a} \\ \overline{T_3} \\ \overline{T_4} \\ \overline{q_4^d} \\ \overline{q_1^a} \end{bmatrix}$$

$$\boldsymbol{A}x = b$$

Agora o sistema de equações que está escrito na forma linear $\boldsymbol{A}x = b$ pode ser resolvido. Observe que a matriz $\boldsymbol{A}$ é uma matriz quadrada $\in \mathbb{R}^{4x4}$. De forma geral a matriz $\boldsymbol{A}$ é sempre uma matriz quadrada $\in \mathbb{R}^{nxn}$ onde n é o numero de nós, o qual coincide com o número de elementos para um contorno fechado. Já o vetor b é o produto de uma matriz $\in \mathbb{R}^{4x8}$ com o vetor contendo as variáveis conhecidas. De forma geral essa é uma matriz $\in \mathbb{R}^{nx2n}$, isso porque para cada nó existem dois fluxos de calor que podem ser diferentes, um a direita é outro a esquerda do nó.

11.8.3 INTEGRAÇÃO DAS MATRIZES $[\boldsymbol{H}]$ E $[\boldsymbol{G}]$ QUANDO O PONTO FONTE NÃO PERTENCE AO ELEMENTO

Quando o ponto fonte não pertence ao elemento Γ_j as integrações existentes nos termos das matrizes $\boldsymbol{H}$ e $\boldsymbol{G}$ não possuem uma solução analítica simples e em geral são feitas numericamente.

Considerando a equação integral de contorno para um nó i e então substituindo (9.55) e (9.56) em (9.53), obtemos:

$$cT^{(i)}(x_d, y_d) = \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} (N_1 T_1 + N_2 T_2) q^* d\Gamma - \sum_{j=1}^{n_{elem}} \int_{\Gamma_j} (N_1 q_1 + N_2 q_2) T^* d\Gamma$$

$$= \sum_{j=1}^{n_{elem}} T_1 \int_{\Gamma_j} N_1 q^* d\Gamma + \sum_{j=1}^{n_{elem}} T_2 \int_{\Gamma_j} N_2 q^* d\Gamma - \sum_{j=1}^{n_{elem}} q_1 \int_{\Gamma_j} N_1 T^* d\Gamma - \sum_{j=1}^{n_{elem}} q_2 \int_{\Gamma_j} N_2 T^* d\Gamma$$

$$= \sum_{j=1}^{n_{elem}} T_1 \int_{\Gamma_j} h_1 d\Gamma + \sum_{j=1}^{n_{elem}} T_2 \int_{\Gamma_j} h_2 d\Gamma - \sum_{j=1}^{n_{elem}} q_1 \int_{\Gamma_j} g_1 d\Gamma - \sum_{j=1}^{n_{elem}} q_2 \int_{\Gamma_j} g_2 d\Gamma$$

11.8.3.1 MATRIZ $[\boldsymbol{G}]$

Para a matriz $\boldsymbol{G}$**,** é preciso integrar:

$$\int_{\Gamma_j} g_i d\Gamma = \int_{x_1}^{x_2} N_i T^* d\Gamma$$

Considerando a solução fundamental dada por (9.16), temos:

$$\int_{\Gamma_j} g_i d\Gamma = \frac{-1}{2\pi K} \int_{\Gamma_j} N_i \ln r \, d\Gamma \tag{9.58}$$

onde:

$$r = [(x - x_d)^2 + (y - y_d)^2]^{\frac{1}{2}}$$

Para realizar a integração, é necessário transformar as coordenadas, parametrizando as variáveis, pois embora as variáveis da equação sejam x e y, a integração é em relação à $d\Gamma$ o qual depende de ambos, dx e dy. Além disso, é numericamente interessante transformar o sistema de coordenadas global de forma a tornar o intervalo de integração, no novo sistema local, sempre o mesmo. Considere o elemento Γ_j na Figura 65, definido no sistema de coordenadas global entre

(x_1, y_1) *e* (x_2, y_2), queremos então mapear ele no sistema local de coordenada ξ no intervalo (-1,1).

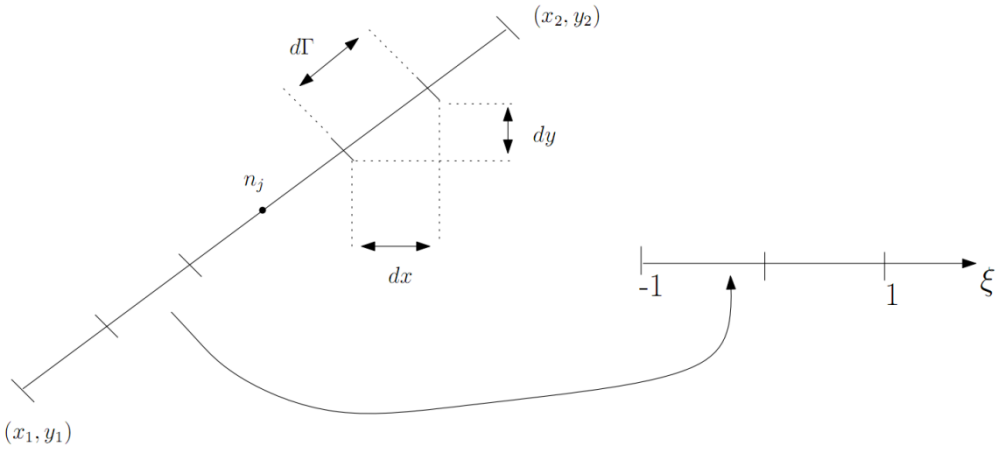

Figura 65 - Mapeamento do sistema de coordenadas global no sistema local.

Observe que x é dado por (9.54):

$$x = N_1(x)x_1 + N_2(x)x_2;\ x_2 \geq x \geq x_1$$

o que queremos é então encontrar $x(\xi)$ tal que:

$$x = N_1(x(\xi))x_1 + N_2(x(\xi))x_2;\ 1 \geq \xi \geq -1$$

Podemos encontrar $x(\xi)$ usando a interpolação de Lagrange, onde $\xi_1 = -1$ e $\xi_2 = 1$, dessa forma obtemos:

$$\mathrm{N}_i = \prod_{\substack{k=1 \\ (k \neq i)}}^{2} \frac{(\xi - \xi_k)}{(\xi_i - \xi_k)} = \begin{cases} \mathrm{N}_2(\xi) = \dfrac{(\xi + 1)}{2} \\ \mathrm{N}_1(\xi) = -\dfrac{(\xi - 1)}{2} \end{cases}$$

e então:

$$x(\xi) = \mathrm{N}_1(\xi)x_1 + \mathrm{N}_2(\xi)x_2$$

$$x(\xi) = -\frac{(\xi - 1)}{2}x_1 + \frac{(\xi + 1)}{2}x_2$$

$$x = \frac{1}{2}[(x_2 - x_1)\xi + (x_2 + x_1)] \qquad (9.59)$$

$$\frac{dx}{d\xi} = \frac{x_2 - x_1}{2}$$

de forma análoga obtemos para y:

$$y = \frac{1}{2}[(y_2 - y_1)\xi + (y_2 + y_1)]$$

$$\frac{dy}{d\xi} = \frac{y_2 - y_1}{2}$$

Da Figura 65, podemos obter que:

$$d\Gamma = \left(dx^2 + dy^2\right)^{\frac{1}{2}}$$

Dividindo ambos os lados da igualdade por $d\xi$, temos:

$$\frac{d\Gamma}{d\xi} = \left[\left(\frac{dx}{d\xi}\right)^2 + \left(\frac{dy}{d\xi}\right)^2\right]^{\frac{1}{2}} = \left[\left(\frac{x_2 - x_1}{2}\right)^2 + \left(\frac{y_2 - y_1}{2}\right)^2\right]^{\frac{1}{2}}$$

$$\left[\left(\frac{x_2 - x_1}{2}\right)^2 + \left(\frac{y_2 - y_1}{2}\right)^2\right]^{\frac{1}{2}} = \frac{1}{2}[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{\frac{1}{2}};$$

$$\frac{d\Gamma}{d\xi} = \frac{1}{2}L;\ L = \frac{1}{2}[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{\frac{1}{2}}$$

onde L é o comprimento do elemento de contorno Γ_j. Reescrevendo (9.58), obtemos:

$$\int_{\Gamma_j} g_i d\Gamma = \frac{-1}{2\pi K}\int_{\Gamma_j} N_i \ln r\, d\Gamma = \frac{-1}{2\pi K}\int_{-1}^{1} N_i \ln r \frac{d\Gamma}{d\xi} d\xi$$

$$\int_{\Gamma_j} g_i d\Gamma = \frac{-L}{4\pi K}\int_{-1}^{1} N_i \ln r\, d\xi$$

Equação que pode ser integrada uma vez que todas as variáveis estão em função da variável ξ.

11.8.3.2 MATRIZ $[\boldsymbol{H}]$

Para a matriz $\boldsymbol{H}$ é preciso integrar:

$$\int_{\Gamma_j} h_i d\Gamma = \int_{x_1}^{x_2} N_i q^* d\Gamma$$

Considerando a solução fundamental para o fluxo de calor dado por (9.22), temos:

$$\int_{\Gamma_j} g_i d\Gamma = \frac{1}{2\pi}\int_{\Gamma_j} \frac{1}{r^2} N_i\left(r_x n_x + r_y n_y\right) d\Gamma$$

Utilizando a mesma parametrização empregada anteriormente para a matriz $\boldsymbol{G}$, obtemos:

$$\int_{\Gamma_j} g_i d\Gamma = \frac{L}{2\pi}\int_{-1}^{1} \frac{1}{r^2} N_i\left(r_x n_x + r_y n_y\right) \frac{d\Gamma}{d\xi} d\xi$$

$$\int_{\Gamma_j} g_i d\Gamma = \frac{L}{4\pi}\int_{-1}^{1} N_i \frac{\left(r_x n_x + r_y n_y\right)}{r^2} d\xi$$

Como o elemento Γ_j é linear, as componentes do vetor $\vec{n}$ normal a esse contorno são constantes e dadas por:

$$n_x = -\frac{dy}{dS} = -\frac{\Delta y}{\Delta S} = -\frac{(y_2 - y_1)}{L}$$

$$n_x = \frac{dx}{dS} = \frac{\Delta x}{\Delta S} = \frac{(x_2 - x_1)}{L}$$

A integração agora já pode ser efetuada uma vez que todas as variáveis estão em função da variável ξ.

11.8.4 INTEGRAÇÃO DAS MATRIZES $[\boldsymbol{H}]$ E $[\boldsymbol{G}]$ QUANDO O PONTO FONTE PERTENCE AO ELEMENTO

Quando a formulação do problema do MEC é feita utilizando elementos lineares ou constantes, é possível obter simples solução analítica para as integrais quando o ponto fonte pertence ao elemento.

11.8.4.1 MATRIZ $[\boldsymbol{G}]$

Para a matriz $\boldsymbol{G}$ é preciso integrar:

$$\int_{\Gamma_j} g_i d\Gamma = \int_{\Gamma_j} N_i T^* d\Gamma$$

Considerando a solução fundamental dada por (9.16), temos novamente a equação (9.58) dada por:

$$\int_{\Gamma_j} g_i d\Gamma = \frac{-1}{2\pi K} \int_{\Gamma_j} N_i \, ln\, r \, d\Gamma$$

Utilizando a parametrização (9.59) podemos escrever a coordenada x_d do ponto fonte como:

$$x_d = \frac{1}{2}\left[(x_2 - x_1)\xi_d + (x_2 + x_1)\right]$$

e então temos que r_x pode ser escrito como:

$$r_x = x - x_d = \frac{1}{2}[(x_2 - x_1)\xi + (x_2 + x_1)] - \frac{1}{2}\left[(x_2 - x_1)\xi_d + (x_2 + x_1)\right]$$

$$r_x = \frac{1}{2}(x_2 - x_1)\left(\xi - \xi_d\right)$$

Analogamente, temos que r_y pode ser escrito da forma:

$$r_y = \frac{1}{2}(y_2 - y_1)\left(\xi - \xi_d\right)$$

E então r pode ser reescrito como:

$$r = \left[{r_x}^2 + {r_y}^2\right]^{\frac{1}{2}} = r = \sqrt{\left[\frac{1}{2}(x_2 - x_1)\left(\xi - \xi_d\right)\right]^2 + \left[\frac{1}{2}(y_2 - y_1)\left(\xi - \xi_d\right)\right]^2}$$

$$= \frac{1}{2}\left(\xi - \xi_d\right)[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{\frac{1}{2}}$$

$$r = \frac{L\left(\xi - \xi_d\right)}{2}; L = [(x_2 - x_1)^2 + (y_2 - y_1)^2]^{\frac{1}{2}} \qquad (9.60)$$

Considerando a parametrização das variáveis e então substituindo (9.60) em (9.58), obtemos:

$$\int_{\Gamma_j} g_i d\Gamma = \frac{-1}{2\pi K} \int_{-1}^{1} N_i \ln\left[\frac{L(\xi - \xi_d)}{2}\right] \frac{d\Gamma}{d\xi} d\xi$$

$$\int_{\Gamma_j} g_i d\Gamma = \frac{-L}{4\pi K} \int_{-1}^{1} N_i \ln\left[\frac{L(\xi - \xi_d)}{2}\right] d\xi \qquad (9.61)$$

É importante saber que $N_i(x) = N_i\big(x(\xi)\big) = N_i(\xi)$. Considere, por exemplo, $N_1(x)$. Assim, temos:

$$N_1(x) = \frac{x - x_2}{x_1 - x_2} \tag{9.62}$$

Substituindo (9.59) em (9.62), mostramos que essa afirmação é verdadeira:

$$\begin{aligned} N_1\big(x(\xi)\big) &= \frac{\frac{1}{2}[(x_2 - x_1)\xi + (x_2 + x_1)] - x_2}{x_1 - x_2} \\ &= \frac{1}{2}\frac{(x_2 - x_1)\xi + x_2 + x_1 - 2x_2}{x_1 - x_2} = \frac{1}{2}\frac{(x_2 - x_1)\xi - x_2 + x_1}{x_1 - x_2} \\ &= -\frac{1}{2}\frac{(x_1 - x_2)\xi + x_2 - x_1}{x_1 - x_2} = -\frac{1}{2}\frac{(x_1 - x_2)\xi - (x_1 - x_2)}{x_1 - x_2} \\ N_1\big(x(\xi)\big) &= -\frac{\xi - 1}{2} = N_1(\xi) \end{aligned} \tag{9.63}$$

Substituindo (9.63) em (9.61), podemos calcular a integral para $g_i = g_1$:

$$\begin{aligned} \int_{\Gamma_j} g_1 d\Gamma = &= \frac{-L}{4\pi K}\int_{-1}^{1}\left(-\frac{\xi - 1}{2}\right) ln\left[\frac{L(\xi - \xi_d)}{2}\right] d\xi \\ &= \frac{L}{8\pi K}\int_{-1}^{1}(\xi - 1)\, ln\left[\frac{L(\xi - \xi_d)}{2}\right] d\xi \\ &= \frac{L}{8\pi K}\int_{-1}^{1}(\xi - 1)\, ln\left[\frac{(\xi - \xi_d)}{2}\right] d\xi + \int_{-1}^{1}(\xi - 1)\, ln\, L\, d\xi \end{aligned}$$

Observe que caso o ponto fonte esteja no nó 2, ou seja $\xi_2 = 1$, a integral é nula pois $(\xi - 1) = 0$. No caso geral, o que ocorre é que $N_i\big(\xi_j\big) = 0$ para $i \neq j$ e deste modo a integral para g_i só não é nula caso o ponto fonte esteja no nó i.

Agora, resolvendo primeiramente a integral à direita, temos:

$$\int_{-1}^{1}(\xi - 1) ln\, L\, d\xi = ln\, L\int_{-1}^{1}(\xi - 1) d\xi = ln\, L\left(\frac{\xi^2}{2} - \xi\right)\Big|_{-1}^{1} = 2\, ln\, L$$

e então:

$$\int_{\Gamma_j} g_1 d\Gamma = \frac{L}{8\pi K}\left[\int_{-1}^{1} ln\left[\frac{(\xi - \xi_d)}{2}\right](\xi - 1)\, d\xi + 2\, ln\, L\right] \tag{9.64}$$

Definindo a mudança de variável, dada por:

$$\eta = \frac{(\xi - \xi_d)}{2}$$

ou seja:

$$\xi = 2\eta + \xi_d;\ \frac{d\xi}{d\eta} = 2$$

Assim, o novo intervalo de integração e dado por:

$$\eta(\xi = -1) = \frac{(-1 - \xi_d)}{2}$$

$$\eta(\xi = 1) = \frac{(1 - \xi_d)}{2}$$

Observe que ξ_d pode ser somente 1 ou -1, pois o ponto fonte pertence a um dos nós do elemento, os quais estão na extremidade desse onde $\xi = 1$ ou $\xi = -1$. Deste modo, temos que o intervalo de integração é $(-1{,}0)$ para $\xi_d = 1$ ou $(0{,}1)$ para $\xi_d = -1$. No entanto, como dito anteriormente, para a integral de g_1 não resultar em valor nulo é preciso considerar o ponto fonte no nó 1 e consequentemente a integração para $\xi_d = \ \xi_1 = -1$ no intervalo $(0{,}1)$.

Assim, utilizando a mudança de variável a integral (9.64) pode ser escrito como:

$$\begin{aligned}
\int_{\Gamma_j} g_1 d\Gamma \ &= \frac{L}{8\pi K}\left[\int_0^1 ln\,\eta(2\eta - 1 - 1)\frac{d\xi}{d\eta} d\eta + 2\, ln\, L\right] \\
&= \frac{L}{8\pi K}\left[\int_0^1 ln\,\eta\, 2(\eta - 1)\, 2d\eta + 2\, ln\, L\right] \\
&= \frac{L}{8\pi K}\left[4\left(\int_0^1 ln\,\eta(\eta - 1)\, d\eta\right) + 2\, ln\, L\right] \\
&= \frac{L}{8\pi K}\left[4\left(\int_0^1 \eta ln\,\eta\, d\eta - \int_0^1 ln\,\eta\, d\eta\right) + 2\, ln\, L\right]
\end{aligned}$$

O termo entre parêntesis resulta em:

$$\int_0^1 \eta ln\,\eta\, d\eta - \int_0^1 ln\,\eta\, d\eta = \frac{3}{4}$$

e finalmente temos que quando o ponto fonte está no nó 1 a integral é dada por:

$$\int_{\Gamma_j} g_1 d\Gamma = \frac{L}{8\pi K}\left[4\left(\frac{3}{4}\right) + 2\,ln\,L\right]$$

e então:

$$\int_{\Gamma_j} g_1 d\Gamma = \frac{L}{2\pi K}\left(\frac{3}{2} + ln\,L\right)$$

$$\int_{\Gamma_j} g_2 d\Gamma = 0$$

De maneira análoga, é possível mostrar que, caso o ponto fonte esteja no nó 2, a integral é dada por:

$$\int_{\Gamma_j} g_1 d\Gamma = 0$$

$$\int_{\Gamma_j} g_2 d\Gamma = \frac{L}{2\pi K}\left(\frac{3}{2} + ln\,L\right)$$

11.8.4.1 MATRIZ $[\boldsymbol{H}]$

Primeiramente é necessário lembrar que:

$$\begin{cases} H_{ij} & se\ i \neq j \\ -cT_i + H_{ij} & se\ i = j \end{cases}$$

Desse modo, a integral se torna:

$$\int_{\Gamma_j} g_i d\Gamma = -c + \int_{\Gamma_j} g_i d\Gamma$$

Quando o contorno é aproximado por elementos de contorno lineares contínuos, o vetor $\vec{n}$, normal a um elemento Γ_j do contorno, é constante. Nesse elemento, e qualquer vetor formado por dois pontos desse elemento é perpendicular

a esse vetor normal. Assim, considerando o ponto fonte em uma região suave do contorno, ou seja, $c = \frac{1}{2}$, temos que:

$$\int_{\Gamma_j} g_i d\Gamma = -\frac{1}{2} + \frac{1}{2\pi}\int_{\Gamma_j} \frac{1}{r^2} N_i \left(r_x n_x + r_y n_y\right) d\Gamma$$

$$\int_{\Gamma_j} g_i d\Gamma = -\frac{1}{2} + \frac{1}{2\pi}\int_{\Gamma_j} \frac{1}{r^2} N_i (\vec{r} \cdot \vec{n}) d\Gamma$$

No entanto $\vec{r}$ é definido por um nó pertence ao elemento Γ_j e o ponto fonte, o qual também pertence ao elemento, é perpendicular ao vetor normal $\vec{n}$. Assim, temos que $\vec{r} \cdot \vec{n} = 0$ e a integral e dada por:

$$\int_{\Gamma_j} g_i d\Gamma = -\frac{1}{2}$$