Universidade de Brasília

Faculdade UnB Planaltina

Programa de Pós-Graduação em Ciência de Materiais

Wilson Domingos Sidinei Alves Miranda

Algoritmo paralelo para determinação de autovalores de matrizes hermitianas

Brasília

2015

Wilson Domingos Sidinei Alves Miranda

Algoritmo paralelo para determinação de autovalores de matrizes hermitianas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência de Materiais pelo Programa de Pós-Graduação em Ciência de Materiais da Faculdade UnB Planaltina da Universidade de Brasília.

Universidade de Brasília

Faculdade UnB Planaltina

Programa de Pós-Graduação em Ciência de Materiais

Orientador: Bernhard Georg Enders Neto

Brasília

05 de agosto de 2015

Miranda, Wilson Domingos Sidinei Alves

Algoritmo paralelo para determinação de autovalores de matrizes hermitianas/ Wilson Domingos Sidinei Alves Miranda. – Brasília, 05 de agosto de 2015.

84 p.: il. (algumas color.) ; 30 cm.

Orientador: Bernhard Georg Enders Neto

Dissertação (mestrado) – Universidade de Brasília
Faculdade UnB Planaltina
Programa de Pós-Graduação em Ciência de Materiais , 05 de agosto de 2015.

1. Problemas de autovalor. 2. Sequência de Sturm. 3. Computação paralela. 4. Matriz tridiagonal. 5. Matriz simétrica. I. Enders Neto, Bernhard Georg. II. Universidade de Brasília. III. Faculdade UnB Planaltina. IV. Título.

Wilson Domingos Sidinei Alves Miranda

Algoritmo paralelo para determinação de autovalores de matrizes hermitianas

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Ciência de Materiais pelo Programa de Pós-Graduação em Ciência de Materiais da Faculdade UnB Planaltina da Universidade de Brasília.

Trabalho aprovado. Brasília, 05 de agosto de 2015:

Bernhard Georg Enders Neto
Orientador

Fábio Ferreira Monteiro
Membro Externo - IF-UnB

José Eduardo Castilho
Membro Interno

Brasília
05 de agosto de 2015

À minha mãe, que me deu suporte
inicial, e aos meus irmãos Eveline
e Idelbrando, que me apoiaram.

Agradecimentos

Ao meu orientador, Prof. Dr. Bernhard Georg Enders Neto, por suas explicações, sugestões, por sua ajuda durante a redação desta dissertação e por sua amizade.

Aos meus amigos Lindomar, Éder e Charles pela convivência e troca de experiências ao longo do curso.

Ao Prof. Dr. David Lima Azevedo pelo incentivo, novas perspectivas e amizade.

Aos técnicos, Aristides e Jôrive, da secretaria da pós-graduação pelo pronto atendimento quando necessário.

A todos os familiares que contribuíram de alguma forma para a conclusão deste trabalho.

Agradeço à Secretaria de Estado de Educação do Distrito Federal que me deu suporte financeiro através do afastamento remunerado para estudos.

"Os conceitos mais simples
são os mais abstratos."

Wilhelm Ostwald (1853–1932)

Resumo

Um dos principais problemas da álgebra linear computacional é o problema de autovalor, $Au = \lambda u$, onde A é usualmente uma matriz de ordem grande. A maneira mais efetiva de resolver tal problema consiste em reduzir a matriz A para a forma tridiagonal e usar o método da bissecção ou algoritmo QR para encontrar alguns ou todos os autovalores. Este trabalho apresenta uma implementação em paralelo utilizando uma combinação dos métodos da bissecção, secante e Newton-Raphson para a solução de problemas de autovalores de matrizes hermitianas. A implementação é voltada para unidades de processamentos gráficos (GPUs) visando a utilização em computadores que possuam placas gráficas com arquitetura CUDA. Para comprovar a eficiência e aplicabilidade da implementação, comparamos o tempo gasto entre os algoritmos usando a GPU, a CPU e as rotinas DSTEBZ e DSTEVR da biblioteca LAPACK. O problema foi dividido em três fases, tridiagonalização, isolamento e extração, as duas últimas calculadas na GPU. A tridiagonalização via DSYTRD da LAPACK, calculada em CPU, mostrou-se mais eficiente do que a realizada em CUDA via DSYRDB. O uso do método *zeroinNR* na fase de extração em CUDA foi cerca de duas vezes mais rápido que o método da bissecção em CUDA. Então o método híbrido é o mais eficiente para o nosso caso.

Palavras-chaves: Problemas de autovalor. Sequência de Sturm. Computação paralela. Matriz tridiagonal. Matriz simétrica.

Abstract

One of the main problems in computational linear algebra is the eigenvalue problem $Au = \lambda u$, where A is usually a matrix of big order. The most effective way to solve this problem is to reduce the matrix A to tridiagonal form and use the method of bisection or QR algorithm to find some or all of the eigenvalues. This work presents a parallel implementation using a combination of methods bisection, secant and Newton-Raphson for solving the eigenvalues problem for Hermitian matrices. Implementation is focused on graphics processing units (GPUs) aimed at use in computers with graphics cards with CUDA architecture. To prove the efficiency and applicability of the implementation, we compare the time spent between the algorithms using the GPU, the CPU and DSTEBZ and DSTEVR routines from LAPACK library. The problem was divided into three phases, tridiagonalization, isolation and extraction, the last two calculated on the GPU. The tridiagonalization by LAPACK's DSYTRD, calculated on the CPU, proved more efficient than the DSYRDB in CUDA. The use of the method *zeroinNR* on the extraction phase in CUDA was about two times faster than the bisection method in CUDA. So the hybrid method is more efficient for our case.

Keywords: Eigenvalue problems. Sturm's Sequence. Parallel computing. Tridiagonal Matrix. Symmetric Matrix.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
EISPACK	Eigensystem Package
GPU	Graphics Processing Unit
LAPACK	Linear Algebra Package

Sumário

1 Introdução

As evoluções nos campos de pesquisas permitiram até o ano de 2003, produzir CPUs cada vez mais rápidas, sendo que os microprocessadores podem ser capazes de fazer bilhões de operações de ponto flutuante por segundo (GFLOPS) e em *clusters*, pode-se chegar à casa dos cem milhões de GFLOPS de operações por segundo. Apesar disso, em 2003, o ganho de processamento ao se produzir uma CPU mais rápida deixou de ser vantajoso, devido ao alto gasto energético e por causa da dissipação do calor gerado, então a indústria de semicondutores fez a opção por construir microprocessadores com mais núcleos ao invés de aumentar a capacidade de um único núcleo (1).

Com a mudança do número de núcleos em cada processador, a indústria de microprocessadores criou um problema para o desenvolvimento de software, visto que os programas eram processados de forma sequencial em um núcleo, com a introdução de mais núcleos, a forma de confecção dos programas deveria sofrer alterações para utilizar a nova capacidade que os microprocessadores poderiam oferecer, caso contrário os processadores desperdiçariam potência.

Para resolver o problema dos vários núcleos, a indústria de semicondutores, possibilitou que os programas sequenciais fossem executados se movendo entre os múltiplos núcleos, conceito que é conhecido como *multicore trajectory*, e mais tarde a *Intel* criou a tecnologia *hyperthreading*, na qual o microprocessador simula ter mais núcleos, e maximiza a velocidade de execução dos programas sequenciais, em contra-partida, foi pensada também uma forma de fazer os programas rodarem em vários núcleos ao mesmo tempo, conceito que ficou conhecido como *many-core trajectory* (1).

Em 2007, a *NVIDIA*, lançou oficialmente o paradigma de programação CUDA, que é um modelo geral de programação em paralelo para o uso em GPUs, disponível a partir de suas placas GeForce série 8. Estas placas são formadas por várias unidades de processamentos, com memória compartilhada e capazes de executar milhares de *threads* (1).

Uma das principais vantagens em usar as placas gráficas, GPUs, é que as *threads* das GPUs são muito mais leves do que as threads dos processadores, CPUs. Logo o gasto computacional para disparar uma *thread* de GPU é muito menor do que o para disparar uma *thread* de CPU. As tecnologias *many-core*, em especial as unidades de processamento gráfico (GPUs) tem liderado a corrida ao mostrar grande desempenho para resolver cálculos em ponto flutuante (1).

Um ano após o lançamento oficial do CUDA, em 2008, a NVIDIA aparece na

lista dos 500 computadores com maior capacidade de processamento, com o TSUBAME, um supercomputador construído em parceria com o Instituto de Tecnologia de Tóquio, que possuía 170 GPUs, ocupando a trigésima posição no ranque (2).

As GPUs mostram um caminho a ser seguido na computação de alta performance, em relação ao custo e a capacidade de paralelizar dos programas. Em 2009 já existiam GPUs trabalhando a 1 teraflops em precisão simples enquanto os processadores *multicore* estavam na casa de 100 gigaflops em precisão dupla.

1.1 Motivação e relevância

Um dos principais problemas da álgebra linear computacional é o problema de autovalor e autovetor. Dentre as áreas que possuem aplicações cujos autovalores e autovetores precisam ser calculados, podemos citar: vibrações de sistemas mecânicos, macro-economia, física quântica, técnicas de cadeia de Markov, reações químicas, estabilidade em hidrodinâmica, ranqueamento de páginas de internet (3), nanoestruturas (4). Essas aplicações dão origem a problemas com características distintas no que se refere a simetria e ao tipo de problema, padrão ou generalizado.

Desde Jacobi que em 1846 propôs o primeiro algoritmo para cálculo de autovalores e autovetores de problemas auto-adjuntos (5), vários métodos têm sido propostos. Com a evolução dos computadores e utilização de GPUs surgiu a necessidade de novos algoritmos que possam ser utilizados de maneira massivamente paralela.

Em CUDA não há bibliotecas com aceleração vantajosa disponíveis para o cálculo de autovalores em matrizes de ordem muito grande.

1.2 Objetivos e organização

Este trabalho tem como objetivo a implementação de uma rotina para encontrar os autovalores de uma matriz hermitiana usando CUDA. Na solução do problema de autovalor em CUDA, adotou-se a conversão da matriz hermitiana para uma matriz simétrica real com o dobro da ordem.

No capítulo 2 apresentamos os fundamentos matemáticos utilizados nos algoritmos de busca de autovalores, métodos que permitem a tridiagonalização de matrizes e algoritmos para encontrar raízes de funções de maneira geral.

No capítulo 3 apresentamos os algoritmos de busca de autovalores e raízes de funções voltados para matrizes tridiagonais simétricas.

No capítulo 4 apresentamos os resultados e conclusões, e descrevemos possibilidades de trabalhos futuros.

2 Fundamentos teóricos

Neste capítulo apresentamos os fundamentos matemáticos utilizados na busca de autovalores, métodos que permitem a tridiagonalização de matrizes e algoritmos para encontrar raízes de funções reais.

2.1 Método de diferenças finitas

No método de diferenças finitas, as derivadas da equação diferencial original são substituídas por equações de diferenças adequadas. Tais equações de diferenças são escolhidas de acordo com a precisão desejada ou requerida. Nesta seção apresenta-se os principais conceitos relativos à discretização de equações diferenciais por diferenças finitas e se obtêm equações de diferenças discretas, de primeira e segunda ordem, como aproximações dos operadores diferenciais contínuos presentes nas equações diferenciais originais. Uma breve discussão a respeito da predominância circunstancial dos erros de arredondamento sobre os erros de truncamento é apresentada.

2.1.1 Aproximações por diferenças finitas

A solução de uma equação diferencial implica na determinação dos valores da variável dependente em cada ponto do intervalo de interesse. Computacionalmente, pode-se lidar apenas com uma região contínua se for possível determinar uma expressão analítica para a solução da equação diferencial. Nesse caso, o computador pode ser utilizado para calcular a solução em qualquer ponto desejado da região, com o uso da solução analítica. Contudo, no caso de técnicas numéricas de solução, não é possível tratar a região de interesse como contínua, já que, em geral, os métodos numéricos obtêm a solução do problema em pontos preestabelecidos. O processo de transformação do domínio contínuo em domínio discreto é chamado de *discretização*, onde o conjunto de pontos discretos escolhidos para representar o domínio de interesse é chamado de *malha* (6). A Figura 1 mostra uma malha de pontos uniformemente espaçados, o passo da malha é definido como sendo a distância entre dois pontos adjacentes e é dado por $\Delta x = x_{i+1} - x_i$.

Para que seja possível tratar numericamente as equações diferenciais, elas devem ser expressas sob a forma de operações aritméticas que o computador possa executar. Essencialmente, deve-se representar os operadores diferenciais (contínuos) presentes na equação diferencial por expressões algébricas (discretas), ou seja, deve-se discretizar a equação diferencial. Portanto, antes de resolver a equação diferencial

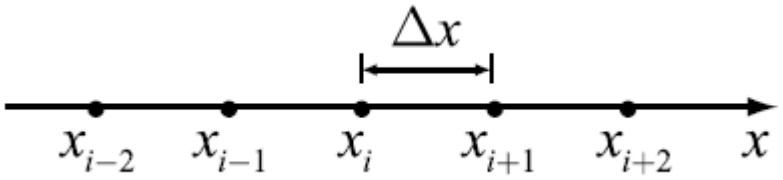


Figura 1 – Malha de pontos uniformemente espaçados.

Fonte: Enders, B.G. (6)

numericamente é preciso encontrar, para os termos que nela aparecem, as respectivas expressões escritas em função dos pontos (finitos) da malha. Essas expressões são denominadas aproximações por diferenças finitas. O resultado final desse processo é uma equação algébrica denominada equação de diferenças finitas. A equação de diferenças finitas é escrita para cada ponto da região discretizada em que se deseja calcular a solução do problema. Resolvendo-se as equações diferenciais, encontra-se a solução aproximada do problema. Tal solução não é exata devido a (*i*) erros inerentes ao processo de discretização das equação, (*ii*) erros de arredondamento nos cálculos feitos pelo computador e (*iii*) erros na aproximação numérica das condições auxiliares (6).

Pode-se obter uma aproximação de diferenças finitas diretamente da definição de derivada de uma função f contínua,

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{2.1}$$

Para tanto, basta que Δx assuma um valor fixo (não-nulo), ao invés de tender a zero, para que o lado direito da Equação (2.1) represente uma aproximação (avançada) de diferenças finitas:

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}. \tag{2.2}$$

Desse modo, utilizando-se dois valores de f separados por uma distância (finita) Δx, a expressão (2.2) representa uma aproximação algébrica para a primeira derivada de f. Essa situação está ilustrada na Figura 2, onde os dois pontos, x e $x + \Delta x$, afastados entre si por uma distância Δx, formam a reta secante cuja declividade serve de aproximação para a derivada da função f no ponto x. Quando a separação Δx diminui, a reta secante se aproxima da reta tangente (derivada real), melhorando assim o valor estimado para a derivada.

Aproximações de diferenças finitas, como mostrada anteriormente, podem ser obtidas de várias formas. As técnicas mais comuns são a expansão em série de Taylor e a interpolação polinomial. O método da expansão em série de Taylor será utilizado na obtenção de aproximações de diferenças finitas de primeira e segunda ordem para as derivadas primeira, segunda e mista de uma função f. A técnica de interpolação

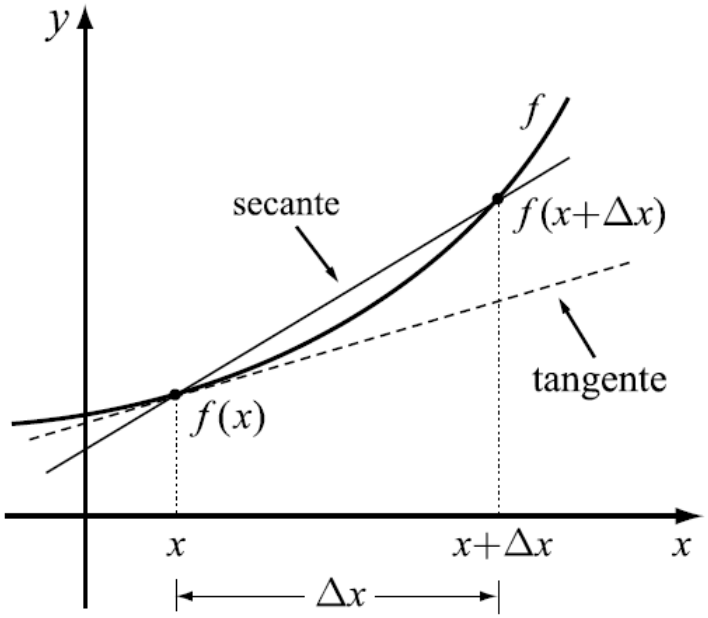


Figura 2 – Aproximação da derivada primeira da função genérica f no ponto x por uma reta secante.

Fonte: Enders, B.G. (6)

é geralmente utilizada no contexto em que se faz necessário o uso de malhas cujo espaçamento não é uniforme (6).

2.1.1.1 Aproximações para a derivada primeira

As aproximações de diferenças finitas têm como base a expansão em série de Taylor de uma função f. Supondo que f seja contínua no intervalo $[a, b]$ de interesse e que possua derivadas até ordem N contínuas nesse intervalo, o teorema de Taylor nos permite escrever, para todo ponto $x \in [a, b]$,

$$f(x) = f(x_0) + (\Delta x)\frac{df}{dx}\bigg|_{x_0} + \frac{(\Delta x)^2}{2!}\frac{d^2f}{dx^2}\bigg|_{x_0} + \frac{(\Delta x)^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x_0} + \cdots + R_N, \tag{2.3}$$

em que $\Delta x = x - x_0$ e R_N é o resto (de Lagrange), definido como

$$R_N = \frac{(\Delta x)^N}{N!}\frac{d^Nf}{dx^N}\bigg|_{\xi}, \ \xi \in [a, b]. \tag{2.4}$$

Para aproximar a derivada primeira de uma função f no ponto x_i vamos expandir $f(x_i + \Delta x)$ em série de Taylor em torno do ponto x_i,

$$f(x_i + \Delta x) = f(x_i) + (\Delta x)\frac{df}{dx}\bigg|_{x_i} + \frac{(\Delta x)^2}{2!}\frac{d^2f}{dx^2}\bigg|_{x_i} + \frac{(\Delta x)^3}{3!}\frac{d^3f}{dx^3}\bigg|_{x_i} + \cdots, \tag{2.5}$$

onde as reticências indicam os termos restantes da série de Taylor até o resto R_N. Após isolar a primeira derivada, pode-se escrever

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} + \left[-\frac{(\Delta x)}{2!}\left.\frac{d^2 f}{dx^2}\right|_{x_i} - \frac{(\Delta x)^2}{3!}\left.\frac{d^3 f}{dx^3}\right|_{x_i} - \cdots \right]. \tag{2.6}$$

Note que, para isolar a primeira derivada, todos os termos da série de Taylor foram divididos pelo espaçamento Δx. Podemos então dizer que a primeira derivada é igual ao quociente

$$\frac{f(x_i + \Delta x) - f(x_i)}{\Delta x}, \tag{2.7}$$

mais o erro local de truncamento, dado por:

$$\left[-\frac{(\Delta x)}{2!}\left.\frac{d^2 f}{dx^2}\right|_{x_i} - \frac{(\Delta x)^2}{3!}\left.\frac{d^3 f}{dx^3}\right|_{x_i} - \cdots \right]. \tag{2.8}$$

O erro local de truncamento aparece naturalmente devido à utilização de um número finito de termos na série de Taylor. Como não pode-se tratar os infinitos termos dessa série na aproximação numérica para a derivada de f, a série foi truncada a partir da derivada de segunda ordem inclusive. O erro local de truncamento fornece uma medida da diferença entre o valor exato da derivada e sua aproximação numérica, indicando também que essa diferença varia linearmente com a redução do espaçamento Δx, isto é, com o refinamento da malha. Assim, para reduzir o erro por quatro, por exemplo, deve-se utilizar um espaçamento $1/4$ do original e portanto, quatro vezes mais pontos na malha (6). Dessa forma, os termos do erro local de truncamento serão representados por $O(\Delta x)$. Deve-se notar que uma expressão do tipo $O(\Delta x)$ só indica como o erro local de truncamento varia com o refinamento da malha, e não o valor do erro.

Pode-se simplificar a notação escrevendo f_i para $f(x_i)$ ou, em geral, $f_{i\pm k}$ para $f(x_i \pm k\Delta x)$. Com isso a Equação (2.6) se torna

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f_{i+1} - f_i}{\Delta x} + O(\Delta x), \tag{2.9}$$

que é uma equação de diferenças finitas que representa uma aproximação de primeira ordem para a primeira derivada de f, utilizando diferença avançada, visto que no cálculo da derivada no ponto x_i foi utilizado um ponto adiante de x_i, no caso, x_{i+1}. A declividade (primeira derivada) de f em x_i é aproximada pela declividade da reta secante formada pelos pontos (x_i, f_i) e (x_{i+1}, f_{i+1}), conforme mostra a Figura 3.

Uma segunda aproximação de diferenças finitas pode ser obtida a partir da expansão de $f(x_i - \Delta x)$ em série de Taylor em torno do ponto x_i:

$$f(x_i - \Delta x) = f(x_i) - (\Delta x)\left.\frac{df}{dx}\right|_{x_i} + \frac{(\Delta x)^2}{2!}\left.\frac{d^2 f}{dx^2}\right|_{x_i} + O(\Delta x)^3. \tag{2.10}$$

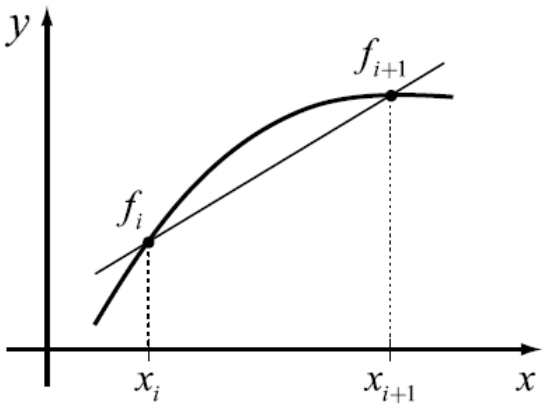


Figura 3 – Pontos utilizados na aproximação para a primeira derivada de f por diferença avançada.

Fonte: Enders, B.G. (6)

Isolando a primeira derivada, têm-se

$$\left.\frac{df}{dx}\right|_{x_i} = \frac{f_i - f_{i-1}}{\Delta x} + O(\Delta x), \tag{2.11}$$

que é outra aproximação de primeira ordem para a primeira derivada de f. Diferentemente da Equação (2.9), na qual utiliza-se um ponto adiante de x_i, a Equação (2.11) utiliza o ponto x_{i-1}, ponto este anterior a x_i. Por essa razão, essa equação de diferenças é chamada de aproximação por diferenças atrasadas (6). A Figura 4 mostra os pontos utilizados nessa aproximação. A declividade da função f no ponto x_i é aproximada pela declividade da reta secante aos pontos (x_{i-1}, f_{i-1}) e (x_i, f_i).

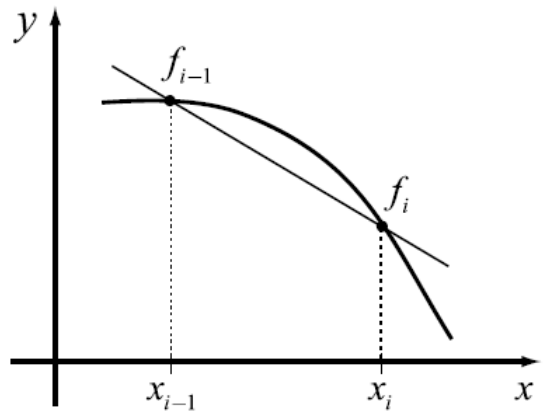


Figura 4 – Pontos utilizados na aproximação para a primeira derivada de f por diferença atrasada.

Fonte: Enders, B.G. (6)

Até agora obteve-se somente aproximações de primeira ordem para a derivada primeira de f, uma aproximação de $O(\Delta x)^2$ ainda para a primeira derivada pode ser

obtida ao subtrair-se as expansões em série de Taylor representadas pelas Equações (2.5) e (2.10):

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2(\Delta x)\frac{df}{dx}\bigg|_{x_i} + O(\Delta x)^3, \tag{2.12}$$

ou, isolando a derivada,

$$\frac{df}{dx}\bigg|_{x_i} = \frac{f_{i+1} - f_{i-1}}{2\Delta x} + O(\Delta x)^2. \tag{2.13}$$

Note que a aproximação dada pela Equação (2.13) utiliza os pontos x_{i-1} e x_{i+1} para o cálculo da primeira derivada de f no ponto central x_i. Por essa razão, ela é denominada aproximação por diferenças centrais. Neste caso, conforme mostra a Figura 5, a derivada de f em x_i é aproximada pela declividade da reta secante que passa pelos pontos (x_{i-1}, f_{i-1}) e (x_{i+1}, f_{i+1}).

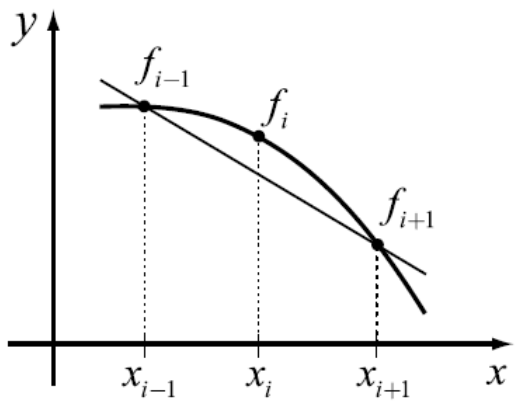


Figura 5 – Pontos utilizados na aproximação segunda ordem para a primeira derivada de f por diferença central.

Fonte: Enders, B.G. (6)

No caso de aproximações de segunda ordem, reduções sucessivas no passo Δx da malha provocam uma redução quadrática no erro da aproximação da primeira derivada de f pela Equação (2.13). Ao dividir o passo por dois, por exemplo, o erro é dividido por quatro, sem precisar de quatro vezes mais pontos, como nas expressões de primeira ordem. Isso é uma propriedade extremamente útil, já que, com menor número de pontos e, portanto, menor esforço computacional, pode-se conseguir uma aproximação melhor que aquelas fornecidas pelas Equações (2.9) e (2.11) (6).

Em resumo, as três fórmulas para a primeira derivada de f deduzidas anterior-

mente, a partir da expansão em série de Taylor, são:

$$\left.\frac{df}{dx}\right|_{x_i} \approx \frac{f_{i+1} - f_i}{\Delta x} \qquad \text{(fórmula avançada)}$$

$$\left.\frac{df}{dx}\right|_{x_i} \approx \frac{f_i - f_{i-1}}{\Delta x} \qquad \text{(fórmula atrasada)}$$

$$\left.\frac{df}{dx}\right|_{x_i} \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x} \qquad \text{(fórmula central).}$$

2.1.1.2 Aproximações para a derivada segunda

Expressões para derivadas de ordem superior podem ser obtidas com o mesmo procedimento com o qual obtivemos as fórmulas para a derivada primeira, isto é, por meio de manipulações adequadas da série de Taylor. Como exemplo, vamos determinar uma aproximação de diferenças centrais de segunda ordem para a segunda derivada de f. Para tal utilizaremos ainda as Equações (2.5) e (2.10). Queremos combiná-las para que a primeira derivada de f seja eliminada, pois estamos interessados na segunda derivada. Por sua vez, as derivadas de ordem superior a dois que permanecerem na expansão farão parte do erro local de trucamento. Assim,

$$f(x_i + \Delta x) - f(x_i - \Delta x) = 2f(x_i) + (\Delta x)^2 \left.\frac{d^2 f}{dx^2}\right|_{x_i} + O(\Delta x)^4. \tag{2.14}$$

Rearranjando os termos, obtém-se

$$\frac{d^2 f}{dx^2}|_{x_i} = \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta x)^2} + O(\Delta x)^2, \tag{2.15}$$

que é a fórmula de diferenças finitas centrais de segunda ordem para derivadas segundas. A Equação (2.15) é a aproximação mais comum encontrada na literatura para derivadas de segunda ordem.

Aproximações de diferenças avançadas e atrasadas de $O(\Delta x)$ para a segunda derivada podem ser obtidas manipulando-se convenientemente as expansões de $f(x_i \pm \Delta x)$ e $f(x_i \pm 2\Delta x)$, nesse caso toma-se os sinais positivos para a aproximação avançada e os negativos para a atrasada. Aproximações avançadas e atrasadas de ordem superior podem também ser obtidas pelas expansões em série de Taylor, simplesmente utilizando mais termos dessa série.

Nos casos em que o problema é dependente do tempo deve-se considerar a expansão em série de Taylor de uma função de duas variáveis independentes, supondo que $f = f(x, t)$, a expansão em torno do ponto x_i fornece:

$$f(x_i + \Delta x, t) = f(x_i, t) + (\Delta x) \left.\frac{\partial f}{\partial x}\right|_{x_i} + \frac{(\Delta x)^2}{2!} \left.\frac{\partial^2 f}{\partial x^2}\right|_{x_i} + \cdots, \tag{2.16}$$

da mesma forma,

$$f(x_i - \Delta x, t) = f(x_i, t) - (\Delta x)\frac{\partial f}{\partial x}\bigg|_{x_i} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 f}{\partial x^2}\bigg|_{x_i} + \cdots . \tag{2.17}$$

Assim, pode-se gerar aproximações de primeira ou segunda ordem para as derivadas parciais, como por exemplo:

$$\frac{\partial f}{\partial x}\bigg|_{x_i} = \frac{f(x_i + \Delta x, t) - f(x_i, t)}{\Delta x} + O(\Delta x) \quad \text{(avançada)} \tag{2.18}$$

$$\frac{\partial f}{\partial x}\bigg|_{x_i} = \frac{f(x_i, t) - f(x_i - \Delta x, t)}{\Delta x} + O(\Delta x) \quad \text{(atrasada)} \tag{2.19}$$

$$\frac{\partial f}{\partial x}\bigg|_{x_i} = \frac{f(x_i + \Delta x, t) - f(x_i - \Delta x, t)}{2\Delta x} + O(\Delta x)^2 \quad \text{(central)} \tag{2.20}$$

$$\frac{\partial^2 f}{\partial x^2}\bigg|_{x_i} = \frac{f(x_i + \Delta x, t) - 2f(x_i, t) + f(x_i - \Delta x, t)}{(\Delta x)^2} + O(\Delta x)^2 \quad \text{(central).} \tag{2.21}$$

É fácil mostrar que existem expressões equivalentes para o tempo, ou seja,

$$\frac{\partial f}{\partial t}\bigg|_{t_i} = \frac{f(x, t_i + \Delta t) - f(x, t_i)}{\Delta t} + O(\Delta t) \quad \text{(avançada)} \tag{2.22}$$

$$\frac{\partial f}{\partial t}\bigg|_{t_i} = \frac{f(x, t_i) - f(x, t_i - \Delta t)}{\Delta t} + O(\Delta t) \quad \text{(atrasada)} \tag{2.23}$$

$$\frac{\partial f}{\partial t}\bigg|_{t_i} = \frac{f(x, t_i + \Delta t) - f(x, t_i - \Delta t)}{2\Delta t} + O(\Delta t)^2 \quad \text{(central)} \tag{2.24}$$

$$\frac{\partial^2 f}{\partial t^2}\bigg|_{t_i} = \frac{f(x, t_i + \Delta t) - 2f(x, t_i) + f(x, t_i - \Delta t)}{(\Delta t)^2} + O(\Delta t)^2 \quad \text{(central).} \tag{2.25}$$

Com as expansões em série de Taylor de $f(x, y)$ em torno do ponto (x_i, y_j) – em que o índice j está associado à coordenada y –, pode-se também determinar expressões envolvendo derivadas parciais mistas com outras variáveis espaciais, isto é, do tipo $\frac{\partial^2 f}{\partial x \partial y}$.

Como estamos interessados agora em obter uma expressão que relacione a variação de f com incrementos em x e y, simultaneamente, deve-se utilizar a expansão em série de Taylor de funções de duas variáveis, dada por:

$$\begin{aligned} f(x_i + \Delta x, y_j + \Delta y) =& f(x_i, y_j) + (\Delta x)\frac{\partial f}{\partial x}\bigg|_{x_i, y_j} + (\Delta y)\frac{\partial f}{\partial y}\bigg|_{x_i, y_j} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 f}{\partial x^2}\bigg|_{x_i, y_j} \\ &+ 2\frac{(\Delta x)(\Delta y)}{2!}\frac{\partial^2 f}{\partial x \partial y}\bigg|_{x_i, y_j} + \frac{(\Delta y)^2}{2!}\frac{\partial^2 f}{\partial y^2}\bigg|_{x_i, y_j} + \cdots . \end{aligned} \tag{2.26}$$

Após algumas manipulações algébricas, a combinação adequada das expansões de $f(x_i \pm \Delta x, y_j - \Delta y)$ e $f(x_i \pm \Delta x, y_j + \Delta y)$ até termos de segunda ordem fornece

$$\left.\frac{\partial^2 f}{\partial x \partial y}\right|_{x_i, y_j} = \frac{f_{i+1,j+1} - f_{i+1,j-1} - f_{i-1,j+1} + f_{i-1,j-1}}{4(\Delta x)(\Delta y)} + O[(\Delta x)^2(\Delta y)^2]. \tag{2.27}$$

2.1.1.3 Erros de arredondamento

Apesar das equações de diferenças finitas obtidas anteriormente sugerirem que a redução do passo Δx melhora a qualidade da aproximação para a derivada, isso só é verdade até certo ponto, na prática a redução do espaçamento da malha é limitada pelos erros de arredondamento. Erros de arredondamento estão presentes na solução numérica de um problema devido à aritmética de precisão finita utilizada pelos computadores. No método de diferenças finitas, se o passo da malha for demasiadamente pequeno, os erros de arredondamento nos cálculos passam a dominar o erro local de truncamento das expressões de diferenças finitas, e o erro relativo percentual efetivamente aumenta (6). Para ilustrar esse fato vamos calcular numericamente a derivada primeira da função $f(x) = xe^x$ no ponto $x = 2$ utilizando a equação de diferenças centrais de segunda ordem, Equação (2.13). Sabe-se que a derivada dessa função é $f'(x) = (x+1)e^x$, vamos utilizar o valor exato de $f'(2)$ para calcular o erro relativo das aproximações realizadas com os diferentes Δx. Conforme podemos verificar na Tabela 1, o erro relativo percentual, dado por

$$E_{rel} = \left|\frac{\text{valor exato} - \text{valor aproximado}}{\text{valor exato}}\right| \cdot 100, \tag{2.28}$$

diminui quadraticamente cada vez que Δx é reduzido pela metade.

Tabela 1 – Erro relativo percentual em função de Δx na aproximação da derivada primeira de $f(x) = xe^x$ no ponto $x = 2$ por diferenças finitas centrais.

Δx	$f'(2)$	Erro Relativo
0.1250000000	22.2634852466	0.4345027228
0.0625000000	22.1912277915	0.1085366179
0.0312500000	22.1731819371	0.0271285905
0.0156250000	22.1686716298	0.0067817999
0.0078125000	22.1675441252	0.0016954283
0.0039062500	22.1672622536	0.0004238557
0.0019531250	22.1671917860	0.0001059638
0.0009765625	22.1671741691	0.0000264910
0.0004882812	22.1671697649	0.0000066227
0.0002441406	22.1671686638	0.0000016557

Fonte: Enders, B.G. (6)

A Figura 6 mostra o gráfico do erro relativo percentual E_{rel} em função do espaçamento Δx, para o cálculo de $f'(2)$ por diferenças centrais de segunda ordem.

Como se pode observar, o erro relativo diminui até que os erros de arredondamento se tornem grandes, comparados com os erros locais de truncamento, e passem a dominar o erro da solução numérica. Além do erro ser oscilatório, seu valor depende não só do espaçamento Δx, mas também do valor da função no ponto. A princípio, não há como evitar problemas desse tipo, mas seu efeito pode ser amenizado se realizarmos as operações aritméticas de ponto flutuante em precisão dupla ou quádrupla (quando disponível) (6). No exemplo da Figura 6, os cálculos realizados com precisão dupla indicam que um espaçamento Δx adequado deve ser da ordem de 10^{-5}. A vantagem de se utilizar maior precisão pode ser percebida com a ajuda da Figura 7, onde efetuamos o mesmo cálculo de $f'(2)$, só que agora utilizando aritmética de ponto flutuante com precisão quádrupla (128 bits). Nesse caso, um valor apropriado para Δx é da ordem de 10^{-11}, com erros relativos percentuais da ordem de 10^{-21}, bem melhor que o valor 10^{-9} obtido com a utilização de precisão dupla nos cálculos.

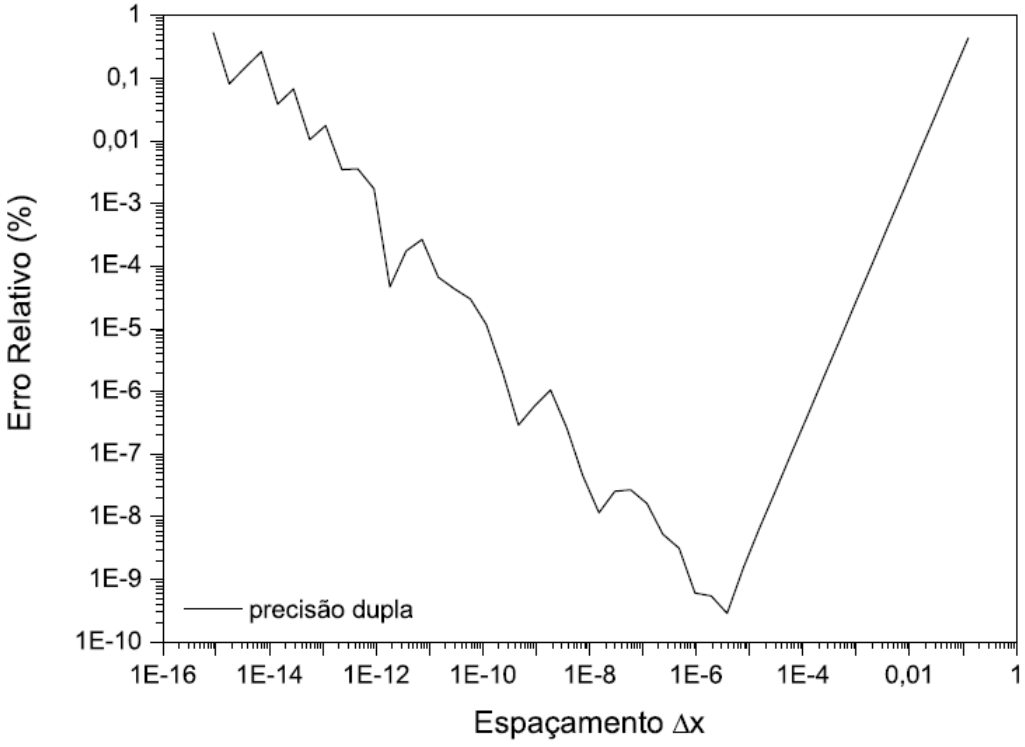


Figura 6 – Erro relativo percentual em função do espaçamento Δx na aproximação da derivada de $f(x) = xe^x$ no ponto $x = 2$ utilizando precisão dupla.

Fonte: Enders, B.G. (6)

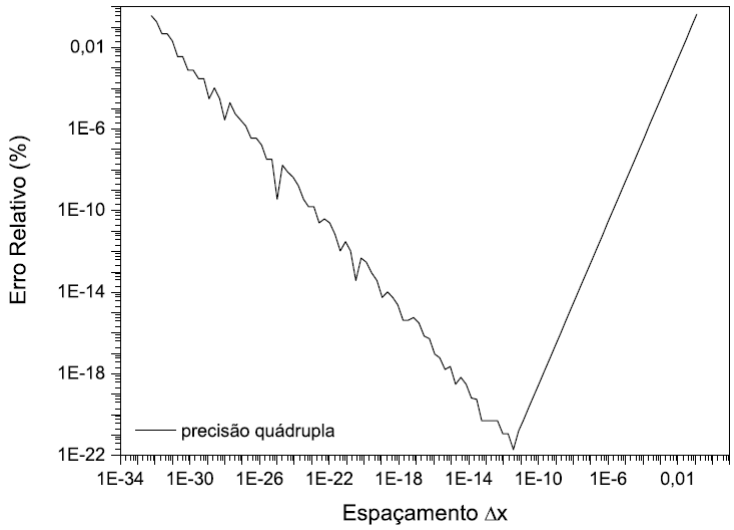


Figura 7 – Erro relativo percentual em função do espaçamento Δx na aproximação da derivada de $f(x) = xe^x$ no ponto $x = 2$ utilizando precisão quádrupla.

Fonte: Enders, B.G. (6)

2.2 Autovalor e autovetor

Definição 2.1 *Seja A uma matriz de ordem n, λ um escalar e u um vetor, $u \neq 0$, λ será um autovalor de A e u um autovetor de A associado a λ se*

$$Au = \lambda u. \tag{2.29}$$

Para determinar os autovalores e autovetores correspondentes de A tem-se que descobrir aqueles que satisfazem a equação $Au = \lambda u$ ou $Au = (\lambda I)u$ ou ainda $(A - \lambda I)u = 0$, onde I é a matriz identidade. Escrevendo explicitamente esta equação, tem-se

$$\begin{bmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.30}$$

Seja

$$B = \begin{bmatrix} a_{11} - \lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \lambda & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \lambda \end{bmatrix}, \tag{2.31}$$

então $B \cdot u = 0$. Se $\det B \neq 0$, sabe-se que o sistema de equações lineares homogêneo indicado em (2.30) tem uma única solução. $u = 0$ sempre é solução de um sistema

homogêneo, então esta única solução seria a nula. Assim, a única maneira de encontrarmos autovetores u (soluções não nulas da equação (2.30)) é ter-se $\det B = 0$, ou seja,

$$\det(A - \lambda I) = 0. \tag{2.32}$$

Impondo esta condição determina-se primeiramente os autovalores λ que satisfazem a equação e depois os autovetores a eles associados. Observa-se que

$$P(\lambda) = \det(A - \lambda I) = \det \begin{bmatrix} a_{11} - \lambda & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} - \lambda \end{bmatrix} \tag{2.33}$$

é um polinômio em λ de grau n, os autovalores procurados são as raízes deste polinômio. $P(\lambda)$ é chamado polinômio característico da matriz A.

2.3 Matriz hermitiana

As matrizes hermitianas pertencem a uma classe de matrizes quadradas de muita utilidade, elas aparecem de maneira natural em diversas aplicações como na solução de equações diferenciais em processamento de imagens. Essas matrizes apresentam outras aplicações de grande importância em primeiro lugar porque o espectro da matriz hermitiana é real. Além disso, as matrizes hermitianas tem um conjunto completo de autovetores ortogonais, por isso as mesmas são diagonalizáveis.

Definição 2.2 *Diz-se que uma matriz $A = (a_{ij})$ complexa de ordem n é hermitiana se $(\overline{A})^T = A$, isto é $a_{ij} = \overline{a}_{ji}$ para todos i,j. Indica-se $A^* = A$ para denotar uma matriz hermitiana.*

Teorema 2.1 *Os autovalores de uma matriz hermitiana são sempre números reais.*

Teorema 2.2 *Os autovetores associados a autovalores distintos de uma matriz simétrica são ortogonais entre si.*

2.4 Matriz tridiagonal

Matrizes esparsas são matrizes que apresentam grande quantidade de elementos nulos, caso contrário são ditas matrizes cheias. Matrizes esparsas que possuem todos os elementos fora da diagonal principal ou das subdiagonais abaixo ou acima da diagonal principal são ditas matrizes banda.

A largura de banda de uma matriz simétrica é definida como sendo a maior distância da diagonal principal até o primeiro elemento não nulo, considerando todas as linhas da matriz.

Matematicamente tem-se:

$$\beta(A) = max\{|i-j||a_{ij} \neq 0\} \tag{2.34}$$

e a banda da matriz A é

$$\text{band}(A) = \{\{i,j\}|0 < i-j \leq \beta(A)\}. \tag{2.35}$$

Definição 2.3 *Uma matriz T é dita tridiagonal se $a_{ij} = 0$, sempre que $|i-j| > 1$.*

Uma matriz tridiagonal genérica é apresentada na equação (2.36).

$$T = \begin{bmatrix} a_{11} & a_{12} & 0 & \dots & 0 \\ a_{21} & a_{22} & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & a_{n-1\ n} \\ 0 & \dots & 0 & a_{n\ n-1} & a_{nn} \end{bmatrix} \tag{2.36}$$

As matrizes tridiagonais apresentam uma grande vantagem no armazenamento, visto que fora das três diagonais os outros elementos são todos iguais a zero, dispensando assim seu armazenamento.

Uma matriz é dita tridiagonal simétrica se obedecer as características anteriores e os elementos das subdiagonais forem iguais.

2.5 Matrizes diagonalizáveis

Definição 2.4 *Uma matriz T $n \times n$ é dita diagonalizável se existirem uma matriz invertível Q e uma matriz diagonal D satisfazendo*

$$Q^{-1}TQ = D. \tag{2.37}$$

Nesse caso diz-se que Q diagonaliza T.

Observações:

- *Se T é diagonalizável, então os vetores colunas da matriz Q que diagonaliza T são autovetores de T e os elementos diagonais de D são os autovalores associados.*
- *Se T é diagonalizável, então T pode ser fatorada em um produto QDQ^{-1}.*

Teorema 2.3 *Uma matriz T $n \times n$ é diagonalizável se e somente se T tem n autovetores linearmente independentes.*

Teorema 2.4 *Se T é uma matriz simétrica $n \times n$, então existe uma matriz ortogonal Q tal que $Q^{-1}TQ = D$, uma matriz diagonal. Os autovalores de T estão sobre a diagonal principal de D. Neste caso, a diagonalização fica Q^TTQ.*

2.6 Matrizes ortogonais

Uma matriz qualquer $Q \in \mathbb{R}^{n\times n}$ é ortogonal se $Q^TQ = I$. Isto significa que, Q é inversível e $Q^{-1} = Q^T$.

Propriedade 2.1 *Dada uma matriz ortogonal $Q \in \mathbb{R}^{n\times n}$ e $x, y \in \mathbb{R}^n$, valem as seguintes propriedades:*

1. $||Qx||_2 = ||x||_2$;
2. $|\det Q| = 1$;
3. $\langle Qx, Qy\rangle = (Qy)^T(Qx) = y^TQ^TQx = y^TIx = y^Tx = \langle x, y\rangle$.

Dos itens 1 e 3 da propriedade acima, podemos concluir que Q preserva o ângulo entre os vetores x e y. Além disso, se P e Q são matrizes ortogonais, então o produto PQ é ortogonal (7).

2.7 Matriz de Householder

Seja v um vetor não nulo, $v \in \mathbb{R}^n$, e H a matriz definida por

$$H = I - \left(\frac{2}{v^Tv}\right) vv^T. \tag{2.38}$$

A matriz H é chamada de matriz de Householder ou reflexão de Householder. O vetor v é chamado de vetor Householder. As matrizes de Householder são simétricas e ortogonais. De fato,

i) $H^T = \left[I - \left(\frac{2}{v^Tv}\right) vv^T\right]^T = \left[I - \left(\frac{2}{v^Tv}\right) vv^T\right] = H$;

ii)
$$\begin{aligned} H^TH \;= HH \;&= I - \left(\frac{4}{v^Tv}\right) vv^T + \left(\frac{2}{v^Tv}\right)\left(\frac{2}{v^Tv}\right)(v^Tv)vv^T = \\ &= I - \left(\frac{4}{v^Tv}\right) vv^T + \left(\frac{4}{v^Tv}\right) vv^T = I. \end{aligned}$$

Ao multiplicar uma matriz de Householder H por um vetor x, refletimos o vetor através do hiperplano perpendicular a v. Assim, para um vetor $x \in \mathbb{R}^n$, teremos que v é da forma $v = x \pm ||x||_2e_1$, onde $e_1 = (1, 0, \cdots, 0)$ (8).

Exemplo 2.1 *Se* $x = (2,2,1)^T$, *temos que* $||x||_2 = 3$. *Admitindo* $v = x + ||x||_2 e_1$, *temos:*

$$v = (5,2,1)^T \Rightarrow v^T v = 30 \; e \; vv^T = \begin{bmatrix} 25 & 10 & 5 \\ 10 & 4 & 2 \\ 5 & 2 & 1 \end{bmatrix}.$$

Logo:

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \frac{1}{15} \begin{bmatrix} 25 & 10 & 5 \\ 10 & 4 & 2 \\ 5 & 2 & 1 \end{bmatrix} \Rightarrow H = \frac{1}{15} \begin{bmatrix} -10 & -10 & -5 \\ -10 & 11 & -2 \\ -5 & -2 & 14 \end{bmatrix},$$

resultando em $Hx = (-3,0,0)^T$.

2.8 Matriz de Givens

Uma matriz (ou rotação) de Givens é uma matriz da forma:

$$G_{ij} = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}, \tag{2.39}$$

onde $c^2 + s^2 = 1$. Naturalmente, podemos considerar $c = \cos\theta$ e $s = \text{sen}\,\theta$, para algum $\theta \in [0, 2\pi]$. Notemos que a intersecção das linhas i e j com as colunas i e j de G_{ij} formam a submatriz

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}. \tag{2.40}$$

Como

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \cdot \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = \begin{bmatrix} c^2 + s^2 & 0 \\ 0 & c^2 + s^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

concluímos que as matrizes G_{ij} são ortogonais. Além disso, por construção, pré-multiplicar vetores por G_{ij} equivale a uma rotação anti-horária de θ radianos em um plano de coordenadas (i, j).

Dado um vetor $x \in \mathbb{R}^n$ não nulo e aplicando G_{ij} a x, teremos que

$$G_{ij}x = \begin{bmatrix} x_1 \\ \vdots \\ cx_i + sx_j \\ \vdots \\ -sx_i + cx_j \\ \vdots \\ x_n \end{bmatrix}. \tag{2.41}$$

Se x_i e x_j não são ambos nulos, e forçando a coordenada j de $G_{ij}x$ ser nula, obtemos o seguinte sistema nas variáveis c e s:

$$\begin{cases} -sx_i + cx_j &= 0 \\ c^2 + s^2 &= 1 \end{cases} \tag{2.42}$$

que tem como solução

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}} \text{ e } s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}}.$$

Então,

$$G_{ij}x = \begin{bmatrix} x_1 \\ \vdots \\ \sqrt{x_i^2 + x_j^2} \\ \vdots \\ 0 \\ \vdots \\ x_n \end{bmatrix}. \tag{2.43}$$

Assim, podemos utilizar rotações de Givens para anular componentes de vetores ou matrizes.

2.9 Método de Jacobi

O método de Jacobi para o problema de autovalor simétrico é inerentemente paralelo (8). Eles trabalham pela realização de uma sequência de atualizações de similaridade ortogonal $A \leftarrow Q^T AQ$ com a propriedade que cada novo A, embora completo, é mais diagonal do que o predecessor. Eventualmente, as entradas fora da diagonal são pequena o bastante para serem declaradas zeros.

A ideia por trás do método de Jacobi é reduzir sistematicamente a quantidade

$$off(A) = \sqrt{\sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} a_{ij}^2}, \tag{2.44}$$

isto é, a norma dos elementos fora da diagonal. As ferramentas para fazer isso são rotações da forma

$$J(p,q,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}, \tag{2.45}$$

que é chamado de rotações de Jacobi.

O passo básico em um procedimento de cálculo de autovalor envolvendo o método de Jacobi é escolher um par de índices (p,q) que satifaz $1 \leq p < q \leq n$, computar um par cosseno-seno (c,s) tal que

$$\begin{bmatrix} b_{pp} & b_{pq} \\ b_{qp} & b_{qq} \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{2.46}$$

é diagonal, e sobrescrevendo A com $B = J^T A J$ onde $J = J(p,q,\theta)$. Observe que matriz B concorda com A exceto nas linhas e colunas p e q. Além disso, desde que a norma de Frobenius seja preservada pela transformação ortogonal encontraremos

$$a_{pp}^2 + a_{qq}^2 + 2a_{pq}^2 = b_{pp}^2 + b_{qq}^2 + 2b_{pq}^2 = b_{pp}^2 + b_{qq}^2$$

e então

$$\begin{aligned} off(B)^2 &= ||B||_F^2 - \sum_{i=1}^{n} b_{ii}^2 \\ &= ||A||^2 - \sum_{i=1}^{n} a_{ii}^2 + (a_{pp}^2 + a_{qq}^2 - b_{pp}^2 - b_{qq}^2) \\ &= off(A)^2 - 2a_{pq}^2. \end{aligned} \tag{2.47}$$

A matriz A se aproxima da forma diagonal a cada passo do método de Jacobi.

Na próxima seção discutiremos como o par de índices (p,q) podem ser escolhidos.

2.9.1 A decomposição 2-por-2 simétrica de Schur

A diagonalização em (2.46) pode ser representada assim

$$0 = b_{pq} = a_{pq}(c^2 - s^2) + (a_{pp} - a_{qq})cs. \tag{2.48}$$

Se $a_{pq} = 0$, então $(c, s) = (1, 0)$. Caso contrário definiremos

$$\tau = \frac{a_{qq} - a_{pp}}{2a_{pq}} \text{ e } t = s/c$$

e concluiremos de (2.48) que $t = \tan(\theta)$ resolve a equação

$$t^2 + 2\tau t - 1 = 0.$$

Por sua vez é importante selecionar a menor das duas raízes,

$$t = -\tau \pm \sqrt{1 + \tau^2},$$

onde c e s podem ser resolvidos da fórmula

$$c = \frac{1}{\sqrt{1 + t^2}} \quad s = tc.$$

Escolher t como a menor das duas raízes implica que $|\theta| \leq \pi/4$ e tem o efeito de minimizar a diferença entre B e A por causa

$$||B - A||_F^2 = 4(1 - c) \sum_{i=1, i \neq p,q}^{n} (a_{ip}^2 + a_{iq}^2) + 2\frac{a_{pq}^2}{c^2}.$$

Algoritmo 2.1: Algoritmo da decomposição de Schur

Dada uma matriz A simétrica $n \times n$ e inteiros p e q que satisfaça $1 \leq p < q \leq n$, esse algoritmo computa um par cosseno-seno (c, s) tal que se $B = J(p, q, \theta)^T A J(p, q, \theta)$ então $b_{pq} = b_{qp} = 0$.

.

Função: $[c, s] =$ **sym.schur2**(A, p, q)

se $A(p, q) \neq 0$ **então**

  $\tau = (A(q, q) - A(p, p))/(2A(p, q))$

  se $\tau \geq 0$ **então**

    $t = 1/(\tau + \sqrt{1 + \tau^2})$;

  fim

  senão

    $t = -1/(-\tau + \sqrt{1 + \tau^2})$;

  fim

  $c = 1/\sqrt{1 + t^2}$

  $s = tc$

fim

senão

  $c = 1$

  $s = 0$

fim

2.9.2 O algoritmo clássico de Jacobi

Como mencionado acima, somente as linhas e colunas p e q são alteradas quando o subproblema (p, q) é resolvido. Uma vez que **sym.schur2** determina a rotação 2-por-2, então a atualização $A \leftarrow J(p, q, \theta)^T AJ(p, q, \theta)$ pode ser implementada em $6n$ *flops* se a simetria for explorada (8).

Como fazer a escolha dos índices p e q? Do ponto de vista da maximização da redução de $off(A)$ em (2.47), faz sentido escolher (p, q) de modo que a_{pq}^2 é máximo. Essa é a base do algoritmo clássico de Jacobi.

Algoritmo 2.2: Algoritmo clássico de Jacobi

Dada uma matriz simétrica $A \in \mathbb{R}^{n \times n}$ e a tolerância $tol > 0$, esse algoritmo sobrescreve A com $V^T AV$, onde V é ortogonal e $off(V^T AV) \leq ||A||_F$.
.
$V = I_n$; $eps = tol||A||_F$
enquanto $off(A) > eps$ **faça**
    Escolha (p, q) então $|a_{pq}| = max_{i \neq j}|a_{ij}|$.
    $(c, s) =$ **sym.schur2**(A, p, q)
    $A = J(p, q, \theta)^T AJ(p, q, \theta)$
    $V = VJ(p, q, \theta)$
fim

2.10 Discos de Gershgorin

O Teorema dos discos de Gershgorin fornece um critério simples para localizar todos os autovalores de uma matriz.

Teorema 2.5 (Discos de Gershgorin) *Seja A uma matriz de ordem n, e d_i, $i = 1, 2, \cdots, n$ os discos cujos centros são os elementos a_{ii} e cujos raios r_i são dados por*

$$r_i = \sum_{j=1, j \neq i}^{n} |a_{ij}|, \quad i = 1, 2, \cdots, n. \tag{2.49}$$

Seja D a união de todos os discos d_i. Então, todos os autovalores A encontram-se contidos em D.

A demonstração pode ser encontrada em (9).

2.11 Métodos numéricos

Para algumas equações existem expressões que facilitam a extração de suas raízes utilizando os coeficientes das mesmas. Porém, para polinômios de grau alto e

funções mais complexas torna-se impossível determinar raízes exatas. O que podemos fazer é encontrar aproximações para essas raízes, existem métodos que conseguem determinar raízes com uma determinada precisão dentro dos limites computacionais de uma máquina (10).

Esses métodos partem de um valor inicial para a raiz e em seguida fazem o refinamento da aproximação da raiz considerada, inicialmente, através de processos iterativos. Esse processo é feito em duas fases (10):

1. **Fase de isolamento**: isolar as raízes, esse processo consiste em obter um intervalo em que as raízes estejam contidas.
2. **Fase de extração**: extrair consiste na escolha de aproximações que melhorem a escolha realizada na fase de isolamento, esse processo é feito sucessivamente até obter uma aproximação para a raiz dentro da precisão ε pré-estabelecida.

Com a grande variedade de métodos numéricos utilizados ou propostos por diversos autores para encontrar raízes, consideramos importante realizar um estudo comparativo geral de alguns deles com a finalidade de determinar qual oferece melhores resultados para um conjunto representativo de matrizes de teste.

Métodos iterativos se caracterizam por executarem sequencialmente as instruções passo a passo, sendo que algumas dessas execuções podem ocorrer de forma cíclica.

Durante a execução quando completa um ciclo temos uma iteração. As iterações fazem usos dos resultados das iterações anteriores e também testes que possibilitam verificar se a precisão exigida foi atingida.

2.11.1 Critério de parada

O critério de parada é o que interrompe o algoritmo iniciado pelos métodos numéricos, onde o mesmo deve avaliar se o x_k na k-ésima iteração está suficientemente próximo da raiz exata. Porém nem sempre o valor exato da raiz é conhecido, logo o processo será interrompido de acordo com um dos critérios seguintes:

1. Avaliação do ponto na função

$$|f(x_k)| \leq \varepsilon;$$

2. Avaliação do tamanho do intervalo

$$|x_k - x_{k-1}| \leq \varepsilon \quad \text{ou} \quad \left| \frac{x_k - x_{k-1}}{x_k} \right| \leq \varepsilon,$$

onde ε denota a precisão desejada. Métodos numéricos são feitos de maneira a satisfazer uma das condições enumeradas acima.

2.11.2 Bissecção

Para enunciar o método da bissecção precisamos enunciar o teorema do valor intermediário, visto que o mesmo serve de base para o método da bissecção.

Teorema 2.6 (Valor intermediário) *Seja* $f : [a,b] \to \mathbb{R}$ *contínua. Se* $f(a) < \gamma < f(b)$ *então existe* $c \in (a,b)$ *tal que* $f(c) = \gamma$.

A demonstração pode ser encontrada em (11).

O método da bissecção é um método de encaixe que consiste na divisão do intervalo inicial $[x_i, x_s]$, onde o mesmo contém uma única raiz, para garantir a convergência. O requisito $f(x_i) \cdot f(x_s) < 0$ diz que dentro do intervalo considerado caso a função seja contínua, a mesma conterá um número ímpar de raízes. A aproximação da raiz é feita através do cálculo do ponto médio deste intervalo

$$x_k = \frac{x_i + x_s}{2}, \quad k = 1, 2, \cdots . \tag{2.50}$$

Após o cálculo do ponto médio temos dois novos intervalos: $[x_i, x_k]$ e $[x_k, x_s]$, o que possui a raiz deve ser escolhido para a próxima iteração,

$$\text{se} \quad f(x_i) \cdot f(x_k) < 0, \quad \text{então} \quad x_s \longleftarrow x_k$$

ou

$$\text{se} \quad f(x_k) \cdot f(x_s) < 0, \quad \text{então} \quad x_i \longleftarrow x_k.$$

O processo de cálculo do ponto médio e a escolha do intervalo para a realização da iteração é repetido até que o critério de parada seja satisfeito, garantindo assim uma aproximação com a precisão desejada.

Se o intervalo da iteração $k+1$ for $[x_i, x_s]$, então o intervalo da iteração $k+2$ será $[x_i, x_{k+1}]$ ou $[x_{k+1}, x_s]$, onde x_{k+1} é definido pela equação (2.50). De onde temos

$$|x_s - x_{k+1}| = |x_{k+1} - x_i| = \frac{1}{2}|x_s - x_i|$$

ou

$$|x^* - x_{k+1}| \approx \frac{1}{2}|x^* - x_k|,$$

onde $x_s = x_k$ (ou $x_i = x_k$) e x_i (ou x_s) $\longrightarrow x^*$, na iteração anterior. Assim,

$$\lim_{k\longrightarrow\infty}\frac{|\varepsilon_{k+1}|}{|\varepsilon_k|} = \lim_{k\longrightarrow\infty}\frac{|x^* - x_{k+1}|}{|x^* - x_k|} = \lim_{k\longrightarrow\infty}\frac{\left(\frac{1}{2}\right)^{k+1}|x_i - x_s|}{\left(\frac{1}{2}\right)^{k}|x_i - x_s|} = \lim_{k\longrightarrow\infty}\frac{\left(\frac{1}{2}\right)^{k+1}}{\left(\frac{1}{2}\right)^{k}}$$

$$= \lim_{k\longrightarrow\infty}\frac{\left(\frac{1}{2}\right)^{k}\cdot\left(\frac{1}{2}\right)^{1}}{\left(\frac{1}{2}\right)^{k}} = \lim_{k\longrightarrow\infty}\left(\frac{1}{2}\right)^{1} = \frac{1}{2}.$$

Sendo $\varepsilon_k = x^* - x_k$, o erro da iteração k. A convergência do método da bissecção é linear. A amplitude do intervalo de iteração k é $2^{-(k-1)}$ vezes a amplitude do intervalo inicial.

O método da bissecção possui uma convergência muita lenta e não depende de $f(x)$, apenas o sinal de $f(x)$ é considerado (12). Se

$$\lim_{k\longrightarrow\infty}\frac{|\varepsilon_{k+1}|}{|\varepsilon_k|^r} = C,$$

onde C é uma constante finita não-nula, pode ocorrer os seguintes casos:

1. Se $r = 1$ e $C < 1$, a taxa de convergência é linear.
2. Se $r > 1$ a taxa de convergência é superlinear.
3. Se $r = 2$ a taxa de convergência é quadrática.

A diferença da convergência linear para a superlinear é que assintoticamente uma sequência que converge linearmente ganha um número constante de dígitos aprimorados por iteração, uma sequência que converge superlinearmente ganha sempre um número maior de dígitos aprimorados em cada iteração. Um método que converge quadraticamente dobra o número de dígitos apurados em cada iteração.

Algoritmo 2.3: Algoritmo da bissecção

Entrada: função real f, $x_i, x_s \in D_f$
$f(x_i)f(x_s) < 0$, tolerância tol
Saída: raiz de f

```
enquanto (x_i − x_s)/2 > tol faça
    x_k ← (x_i + x_s)/2;
    se f(x_k) = 0 então
        pausa
    fim
    se f(x_i)f(x_s) < 0 então
        x_s ← x_k
    senão
        x_i ← x_k
    fim
fim
retorna (x_i + x_s)/2
```

2.11.3 Newton-Raphson

O método de Newton-Raphson pode ser usado para calcular raízes reais ou complexas de uma função. Está entre os métodos mais rápidos de cálculo de raízes e quanto mais próximo estiver de uma raiz mais rapidamente convergirá (12).

Para desenvolver a expressão que determina o método de Newton fazemos uso da expansão de uma função em série de Taylor em torno do ponto x_k. A expressão pode ser escrita como:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + f''(x_k)\frac{(x - x_k)^2}{2!} + f'''(x_k)\frac{(x - x_k)^3}{3!} + \cdots, \quad (2.51)$$

onde x_k é um valor aproximado da raiz de $f(x)$ na iteração k do processo iterativo, e f' e f'' são respectivamente a primeira e a segunda derivada da função.

Seja x_{k+1} a raiz da equação $f(x) = 0$, então a equação (2.51) se reduzirá a:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k) + f''(x_k)\frac{(x_{k+1} - x_k)^2}{2!} + f'''(x_k)\frac{(x_{k+1} - x_k)^3}{3!} + \cdots. \quad (2.52)$$

Utilizando os dois primeiros termos da expansão da série de Taylor do segundo membro da equação (2.52) obtemos a expressão para o método de Newton-Raphson:

$$\begin{aligned} f(x_k) + f'(x_k)(x_{k+1} - x_k) &= 0 \\ f'(x_k)(x_{k+1} - x_k) &= -f(x_k) \\ (x_{k+1} - x_k) &= -\frac{f(x_k)}{f'(x_k)} \end{aligned}$$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},\ k = 1, 2, \cdots . \tag{2.53}$$

A equação (2.53) é a equação iterativa do método de Newton-Raphson. Sendo x_1 a aproximação inicial do processo a equação (2.53) resulta da aproximação de uma reta tangente a $f(x)$ no ponto da iteração vigente, x_k, e calcular a intersecção dessa tangente com o eixo das abscissas. A desvantagem desse método reside na necessidade da primeira derivada de $f(x)$.

Agora verificaremos a ordem de convergência do método de Newton-Raphson.

Proposição 2.1 *Seja $f(x)$ uma função com segunda derivada contínua e suponha que x^* é um ponto tal que $f(x^*) = 0$ e $f'(x^*) \neq 0$. Então, desde que a primeira aproximação à raiz x_1, esteja suficientemente perto de x^*, a sequência $\{x_k\}$ gerada pelo método de Newton converge para x^* e a ordem da convergência é igual a dois.*

Este tipo de convergência depende da aproximação inicial x_1, do valor de f'' em pontos próximos de x^* e dos valores de $f'(x_k)$. Se estes valores forem muitos próximos de zero pode ficar muito lento ou pode até mesmo não convergir para a solução esperada. O teorema 2.7 e o 2.8 fornecem as condições de convergência do referido método (12).

Teorema 2.7 *Seja x_1 a aproximação inicial do método de Newton e seja $\{x_k\}$ a sequência gerada por* (2.53)*. Defini-se o intervalo*

$$I = \left[x_1, x_1 - 2\frac{f(x_1)}{f'(x_1)} \right].$$

Suponha que

$$2\left|\frac{f(x_1)}{f'(x_1)}\right| M \leq |f'(x_1)| \ \ \textit{em que} \ \ M = max_x \in I|f''(x)|.$$

Então $x_k \in I$, para $k = 2, 3, \cdots$ e

$$\lim_{k \longleftarrow \infty} x_k = x^*$$

sendo x^* a única raiz de $f(x) = 0$ em I.

Teorema 2.8 *Suponha que $f'(x) \neq 0$, que $f''(x)$ não muda de sinal no intervalo $[a, b]$ e que $f(a) \cdot f(b) < 0$. Se*

$$\left|\frac{f(a)}{f'(a)}\right| < (b - a)\ e\ \left|\frac{f(b)}{f'(b)}\right| < (b - a)$$

então o método de Newton-Raphson converge a partir de uma aproximação inicial qualquer, $x_1 \in [a, b]$.

Na realização de cálculos com funções onde as raízes não são simples há a necessidade de se fazer uma alteração na equação iterativa (2.53). Definindo uma nova função

$$u(x) = \frac{f(x)}{f'(x)},$$

que produz a equação iterativa de Newton para o cálculo da raiz simples da função $u(x) = 0$,

$$x_{k+1} = x_k - \frac{u(x_k)}{u'(x_k)} \tag{2.54}$$

ou

$$x_{k+1} = x_k - \frac{f(x_k)f'(x_k)}{(f'(x_k))^2 - f(x_k)f''(x_k)},\ k = 1, 2, \cdots. \tag{2.55}$$

Algoritmo 2.4: Algoritmo sequencial Newton-Raphson

Entrada: número máximo de iterações N, estimava inicial, tolerância, $f(x)$ e $f'(x)$

Saída: a raiz desejada de f

$i \longleftarrow 0$
$x_k \longleftarrow$ estimativa inicial
enquanto $i \leq N$ **faça**
  $x_{k+1} \longleftarrow x_k - \frac{f(x_k)}{f'(x_k)}$;
  se $abs\left(\frac{x_{k+1} - x_k}{x_{k+1}}\right) < \varepsilon$ **então**
    Apresentar x^*
    Finalizar o programa
  fim
  $x_k \longleftarrow x_{k+1}$
  Exibir a mensagem: "Método falhou em N iterações"
fim

2.11.4 Secante

O método da secante realiza o cálculo da raiz de uma função baseado na aproximação da mesma por uma reta nas proximidades da raiz. O ponto de intersecção da reta com o eixo das abscissas é tido como a aproximação da raiz de $f(x) = 0$. Se a aproximação não atingir a precisão desejada da raiz x^* devemos repetir o processo iterativamente até que a mesma seja atingida.

No método de Newton-Raphson é necessário o cálculo da derivada $f'(x)$ e seu valor numérico a cada iteração. Uma maneira de contornar essa situação é substituir a derivada $f'(x)$ pelo quociente das diferenças (12). Do método de Newton-Raphson temos:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

Para a aproximação da derivada temos:

$$f'(x_k) \cong \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}},$$

onde x_k e x_{k-1} são duas aproximações para a raiz. Então a função iteração para o método da secante assume a seguinte forma:

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{\dfrac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}} \\ x_{k+1} &= x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k)-f(x_{k-1})} \\ x_{k+1} &= \frac{x_k f(x_k) - x_k f(x_{k-1}) - x_k f(x_k) + x_{k-1} f(x_k)}{f(x_k)-f(x_{k-1})}. \end{aligned} \tag{2.56}$$

Dessa maneira definimos a função de iteração para o método da secante:

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})},\ k = 2, 3, \cdots . \tag{2.57}$$

Ainda que seja necessário dois pontos para iniciar o processo iterativo, apenas um novo ponto é calculado bem como seu correspondente de acordo com a função considerada, a cada iteração.

Se em uma iteração $f(x_k) \approx f(x_{k-1})$ pode ocorrer situações de *overflow*, quando essa situação ocorrer o uso da seguinte expressão é indicado:

$$\begin{aligned} x_{k+1} &= \frac{(x_k - x_{k-1}) f(x_k)}{f(x_k) - f(x_{k-1})} \\ x_{k+1} &= x_k - \frac{-(x_{k-1} - x_k) f(x_k)}{-[f(x_{k-1}) - f(x_k)]}. \end{aligned}$$

Simplificando os sinais e dividindo o numerador e o denominador por $f(x_{k-1})$ temos:

$$x_{k+1} = x_k - \frac{(x_{k-1} - x_k)\dfrac{f(x_k)}{f(x_{k-1})}}{\left[1 - \dfrac{f(x_k)}{f(x_{k-1})}\right]}, \tag{2.58}$$

utilizamos (2.58) ao invés de (2.57) quando $|f(x_{k-1})|$ for maior que $|f(x_k)|$; senão trocamos os valores de x_k e x_{k-1} assim como os valores da função, antes de usar a expressão (2.58).

A proposição a seguir trata da convergência do método da secante.

Proposição 2.2 *Seja $f(x)$ uma função com segunda derivada contínua e suponha que o ponto x^* é tal que $f(x^*) = 0$ e $f'(x^*) \neq 0$. Então para x_i suficientemente perto de x^* a sequência x_k gerada pelo método da secante converge para x^*, sendo a ordem de convergência de 1.618.*

Ao calcular uma raiz múltipla o método da secante converge linearmente. Porém é necessário acelerar o processo iterativo introduzindo algumas modificações.

Definimos uma nova função

$$u(x) = \frac{f(x)}{f'(x)},$$

onde a equação $u(x) = 0$ tem uma raiz simples para $x = x^*$. Então para o cálculo da raiz de $u(x) = 0$ a equação da secante modificada tem a seguinte forma

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})u(x)}{u(x_k) - u(x_{k-1})} \tag{2.59}$$

em relação a função original $f(x)$ assume a forma

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})f(x_k)f'(x_{k-1})}{f(x_k)f'(x_{k-1}) - f(x_{k-1})f'(x_k)},\ k = 2, 3, \cdots. \tag{2.60}$$

Algoritmo 2.5: Algoritmo sequencial da secante

Entrada: número máximo de iterações N, tolerância, $f(x)$, x_k e x_{k-1}
Saída: a raiz desejada de f

1 $i \longleftarrow 0$
2 **enquanto** $i \leq N$ **faça**
3 $\quad x_{k+1} = \dfrac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$;
4 $\quad$ **se** $abs\left(\dfrac{x_{k+1} - x_k}{x_{k+1}}\right) < \varepsilon$ **então**
5 $\quad\quad$ Apresentar x^{k+1}
6 $\quad\quad$ Finalizar o programa
7 $\quad$ **fim**
8 $\quad x_{k-1} \longleftarrow x_k$
9 $\quad x_k \longleftarrow x_{k+1}$
10 $\quad i \longleftarrow i + 1$
11 $\quad$ Exibir a mensagem: "Método falhou em N iterações"
12 **fim**

2.11.5 Laguerre

Seja o polinômio $p_n(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n = 0$, com raízes $x_1^*, x_2^*, ..., x_n^*$, para uma melhor compreensão escrevemos o polinômio na forma fatorada

$$p_n(x) = (x - x_1^*)(x - x_2^*)...(x - x_n^*). \tag{2.61}$$

A equação (2.61) pode ser simplificada para melhor entendimento, aplicando logaritmo na mesma , obtendo assim

$$\ln|p_n(x)| = ln|x - x_1^*| + \ln|x - x_2^*| + \cdots + \ln|x - x_n^*|$$

$$\ln|p_n(x)| = \sum_{k=1}^{n} \ln|x - x_k^*|. \tag{2.62}$$

Derivando a equação (2.62), temos

$$G = \frac{d(\ln|p_n(x)|)}{dx} = \frac{1}{x - x_1^*} + \frac{1}{x - x_2^*} + \cdots + \frac{1}{x - x_n^*} = \frac{p_n'(x)}{p_n(x)}. \tag{2.63}$$

Calculando a segunda derivada de (2.62) obtemos

$$H = -\frac{d^2(\ln|p_n(x)|)}{dx^2} = \frac{1}{(x - x_1^*)^2} + \frac{1}{(x - x_2^*)^2} + \cdots + \frac{1}{(x - x_n^*)^2}$$

$$H = -\frac{d(\ln|p_n(x)|)}{dx} = \left[\frac{p_n'(x)}{p_n(x)}\right]^2 - \frac{p_n''(x)}{p_n(x)}. \tag{2.64}$$

Na obtenção da estimativa das raízes devemos fazer uso das seguintes representações:

1. A distância entre x_1^* e o valor inicial x_0 será representado por a.
2. A distância entre o valor inicial x_0 e as outras raízes será dado por b.

$$x_0 - x_1 = a \tag{2.65}$$

$$x_0 - x_k = b, \quad k = 2, 3, ..., n. \tag{2.66}$$

Escrevendo (2.63) e (2.64) em função de a e b

$$\frac{1}{a} + \frac{n-1}{b} = G \tag{2.67}$$

$$\frac{1}{a^2} + \frac{n-1}{b^2} = H. \tag{2.68}$$

Isolando b em (2.67) e substituindo em (2.68) obtemos

$$a = \frac{n}{G + \sqrt{(n-1)(nH - G^2)}}. \tag{2.69}$$

Tomando $G = \dfrac{\dfrac{d}{dx}p(x)}{p(x)}$ em H , onde $H = G^2 - \dfrac{\dfrac{d^2}{dx^2}p(x)}{p(x)}$ temos

$$H = \frac{\left(\dfrac{d}{dx}p(x)\right)^2 - p(x)\dfrac{d^2}{dx^2}p(x)}{(p(x))^2}.$$

Desta forma podemos obter a em termos da função $f(x)$ e suas derivadas

$$a = \frac{nf(x)}{\dfrac{d}{dx}f(x) \mp \sqrt{H(x)}} = \frac{np_n(x_k)}{p_n'(x_k) \mp \sqrt{H(x_k)}},$$

onde $H(x) = (n-1)(nH - G^2)$ deste modo $H(x)$ pode ser escrito como

$$H(x) = (n-1)\left((n-1)\left(\frac{d}{dx}f(x)\right)^2 - nf(x)\frac{d^2}{dx^2}f(x)\right),$$

como $x_{k+1} = x_k - a$, então

$$x_{k+1} = x_k - \frac{np_n(x_k)}{p'_n(x_k) \mp \sqrt{H(x_k)}}, \quad k = 1, 2, \cdots, \tag{2.70}$$

que é a equação iterativa do método de Laguerre, onde

$$H(x_k) = (n-1)[(n-1)(p'_n(x_k))^2 - np_n(x_k)p''_n(x_k)].$$

Nessa expressão n é o grau do polinômio. Iniciando o método com x_1 duas sequências de aproximação são geradas, onde essas sequências convergem para as raízes que se encontram mais próximas de x_1, uma com valor inferior a x_1 e a outra superior.

Se usarmos o sinal em (2.70) que origina o menor incremento a adicionar a x_k para se obter x_{k+1}, a sequência gerada converge para a raiz mais próxima do valor inicial. Para equações algébricas com raízes reais, se x_1 estiver à esquerda da menor das raízes ou à direita da maior delas, então o processo iterativo converge respectivamente para a menor ou para a maior das raízes (12).

A convergência do método de Laguerre ocorre de forma global, visto que o mesmo não depende de valores próximos imputados para a primeira iteração.

O método de Laguerre exige a cada iteração o cálculo de $p_n(x)$ e o das derivadas $p'(x)$ e $p''(x)$ tornando-o assim um dos métodos que mais necessita de operações, mas esse importuno é compensado pela convergência rápida, já que sua convergência é de ordem 3 (12).

O método de Laguerre possui vários algoritmos, no Algoritmo 2.6 é apresentado um deles.

Algoritmo 2.6: Algoritmo sequencial Laguerre

Entrada: $n, (a_i : 0 \leq i \leq n), x_0, M, \varepsilon$
Saída: a raiz desejada de f

para $k = 1, 2, ..., M$ **faça**
  $p \longleftarrow x_n$;
  $p' \longleftarrow 0$;
  $p'' \longleftarrow 0$;
  para $j = n-1, n-2, ..., 0$ **faça**
    $p'' \longleftarrow x_0 p'' + p'$;
    $p' \longleftarrow x_0 p' + p$;
    $p \longleftarrow x_0 p + a_j$;
  fim
  $G \longleftarrow \frac{p'}{p}$;
  $H \longleftarrow G^2 - \frac{(2p'')}{p}$;
  $a \longleftarrow G \pm \sqrt{((n-1)(nH - G^2))}$;
  $x_1 \longleftarrow x_0 + \frac{1}{a}$;
  Saída: k, x_1
  se $\left|\frac{x_1 - x_0}{x_1}\right| \leq \varepsilon$ **então**
    Pare;
  fim
  $x_0 \longleftarrow x_1$
fim

3 Autovalores nas matrizes tridiagonais simétricas

Neste capítulo apresentamos os algoritmos de busca de autovalores e raízes de funções voltados para matrizes tridiagonais simétricas.

3.1 Método de Sturm

Este é o método que permite o cálculo do número exato de raízes, em um intervalo real, de uma dada função (13).

Definição 3.1 *Seja* $\{f_i\}_{i=0}^{m}$ *uma sequência de funções contínuas, com* $f_0(x)$ *diferenciável em* $[a,b]$. *Tal sequência é definida como sequência de Sturm se:*

i) $f_0(x)$ *só contém zeros simples;*

ii) $f_m(x)$ *não se anula em* (a,b);

iii) *Se* $f_j(\alpha) = 0$, *então* $f_{j-1}(\alpha)f_{j+1}(\alpha) < 0$, *qualquer que seja* α *em* (a,b);

iv) *Se* $f_0(\alpha) = 0$, *então* $f'(\alpha)f_1(\alpha) > 0$, *qualquer que seja* α *em* (a,b).

Teorema 3.1 (Sturm) *Se* $f_0(a) \cdot f_0(b) \neq 0$, *o número de raízes reais de* $f_0(x)$ *em* (a,b) *é igual a* $V(a) - V(b)$, *onde* $V(x)$ *indica o número de variações de sinal da sequência de Sturm calculada em x (valores nulos não são contados).*

Demonstração: O número de variações de sinal pode mudar quando x percorre o intervalo (a,b), somente quando alguma função muda de sinal neste intervalo.

Por (ii) esta função não pode ser $f_m(x)$.

Assuma que para algum $\hat{x} \in (a,b)$, $f_V\hat{x} = 0$, $0 < v < m$. Assim, em uma vizinhança de $\hat{x}$, temos que os sinais podem ser:

x	$f_{V-1}(x)$	$f_V(x)$	$f_{V+1}(x)$
$\hat{x} - \epsilon$	$+$	$\pm$	$-$
$\hat{x}$	$+$	0	$-$
$\hat{x} + \epsilon$	$+$	$\pm$	$-$

ou

x	$f_{V-1}(x)$	$f_V(x)$	$f_{V+1}(x)$
$\hat{x}-\epsilon$	$-$	$\pm$	$+$
$\hat{x}$	$-$	0	$+$
$\hat{x}+\epsilon$	$-$	$\pm$	$+$

onde:

1. Os sinais da linha $x = \hat{x}$, seguem de (iii);
2. Os sinais das primeiras e últimas colunas segue da continuidade dos elementos da sequência de Sturm para um ϵ pequeno.

Vê-se, através das tabelas acima, que quando x passa por $\hat{x}$, não há mudança no número do variações de sinal na sequência de Sturm.

Examinamos agora os sinais perto de um zero $\hat{x}$ de $f_0(x)$. Podem ocorrer:

x	$f_0(x)$	$f_1(x)$
$\hat{x}-\epsilon$	$+$	$-$
$\hat{x}$	0	$-$
$\hat{x}+\epsilon$	$-$	$-$

a)

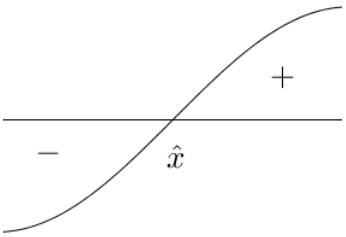
ou

x	$f_0(x)$	$f_1(x)$
$\hat{x}-\epsilon$	$-$	$+$
$\hat{x}$	0	$+$
$\hat{x}+\epsilon$	$+$	$+$

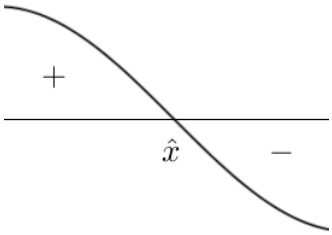
b)

onde:

1. As primeiras colunas representam os dois possíveis casos para um zero simples de $f_0(x)$;
2. O sinal de $f_1(\hat{x})$ segue de (iv) já que se (a), então



e portanto $f_0'(\hat{x}) < 0$, de onde $f_1(\hat{x}) < 0$. E se (b), então



e portanto $f_0'(\hat{x}) > 0$, de onde $f_1(\hat{x}) > 0$;

3. Os sinais dos outros elementos das segundas colunas seguem pela continuidade dos elementos da sequência.

De maneira que agora, como vemos claramente, há um decréscimo de uma mudança no número de variações de sinal quando x passa por um zero de $f_0(x)$.

Portanto, a única maneira de ocorrer mudança de sinal é quando ocorre um zero de $f_0(x)$.

■

É fácil construir a sequência de Sturm quando $f_0(x)$ é um polinômio:

1. Definimos $f_1(x) = f_0'(x)$. Assim (iv) ocorre para zeros simples;
2. Dividimos $f_0(x)$ por $f_1(x)$ e chamamos o resto da divisão de $-f_2(x)$; daí dividimos $f_1(x)$ por $f_2(x)$ e chamamos o resto de $-f_3(x)$, e assim por diante até que determinemos o $M.D.C(f_0(x), f_1(x)) = f_m$.

Podemos escrever então:

$$\begin{aligned} f_0(x) &= P_n(x) \\ f_1(x) &= f_0'(x) \\ f_0(x) &= q_1(x)f_1(x) - f_2(x) \\ &\vdots \\ f_{m-2} &= q_{m-1}(x)f_{m-1} - f_m(x) \\ f_{m-1} &= q_m(x)f_m. \end{aligned} \tag{3.1}$$

Este processo (3.1) é o conhecido algoritmo euclidiano para a determinação do maior divisor comum, f_m, de $f_0(x)$ e $f_1(x)$. Se f_m não é uma constante (isto é, $f_0(x)$ tem raízes múltiplas), então dividimos todos os f_j's anteriores por $f_m(x)$, de onde obtemos uma sequência, como definida, de Sturm, na qual $f_0(x)$ tem somente zeros simples e assim satisfaça (i), e com f_m constante, o que garante (ii).

Note também que se $f_j(\hat{x}) = 0$, então por (3.1), para este ponto, $f_{j-1}(\hat{x}) = -f_{j+1}(\hat{x})$ e, se $f_{j-1}(\hat{x}) = 0$, então $f_0(\hat{x}) = f_1(\hat{x}) = 0$ o que contradiz (i). Logo (iii) é satisfeita.

Portanto, realmente, (3.1) define uma sequência com a qual podemos determinar o número de zeros reais de $P_n(x)$.

Na prática, usamos o Teorema de Sturm sucessivas vezes para subintervalos divididos ao meio. Quando uma raiz é isolada, é mais eficiente empregar uma técnica mais rápida.

Quando é determinado que há uma única raiz em um intervalo (c, d), se esta raiz é simples, temos $f_0(c)f_0(d) < 0$. Podemos dividir este intervalo ao meio e fazer o seguinte teste: $f_0(c)f_0(\frac{c+d}{2}) < 0$, se sim, a raiz procurada está em $(c, \frac{c+d}{2})$. Se não, temos que $f_0(\frac{c+d}{2})f_0(d) < 0$ e a raiz procurada está em $(\frac{c+d}{2}, d)$.

Desta maneira podemos ir dividindo os intervalos de modo que, a cada passo precisamos de menos cálculos do que quando usamos o Teorema de Sturm.

Exemplo 3.1 *Utilizando a sequência de Sturm obtida para* $f(x) = x^3 - 2x - 5$*, construímos o quadro abaixo para calcular* $V(-3), V(-2), V(-1), V(1), V(2)$ *e* $V(3)$*.*

x	f_0	f_1	f_2	f_3
-3	$-$	$+$	$+$	$-$
-2	$-$	$+$	$+$	$-$
-1	$-$	$+$	$+$	$-$
1	$-$	$+$	$+$	$-$
2	$-$	$+$	$+$	$-$
3	$+$	$+$	$+$	$-$

A partir do quadro temos:

- *Para o intervalo* $(-3,-2)$

$$V(-3) - V(-2) = 2 - 2 = 0$$

- *Para o intervalo* $(-2,-1)$

$$V(-2) - V(-1) = 2 - 2 = 0$$

- *Para o intervalo* $(-1,1)$

$$V(-1) - V(1) = 2 - 2 = 0$$

- *Para o intervalo* $(1,2)$

$$V(1) - V(2) = 2 - 2 = 0$$

- *Para o intervalo* $(2,3)$

$$V(2) - V(3) = 2 - 1 = 1$$

Logo $f(x)$ *possui uma raiz no intervalo* $(2,3)$*, que pode ser comprovado pela observação da Figura 8.*

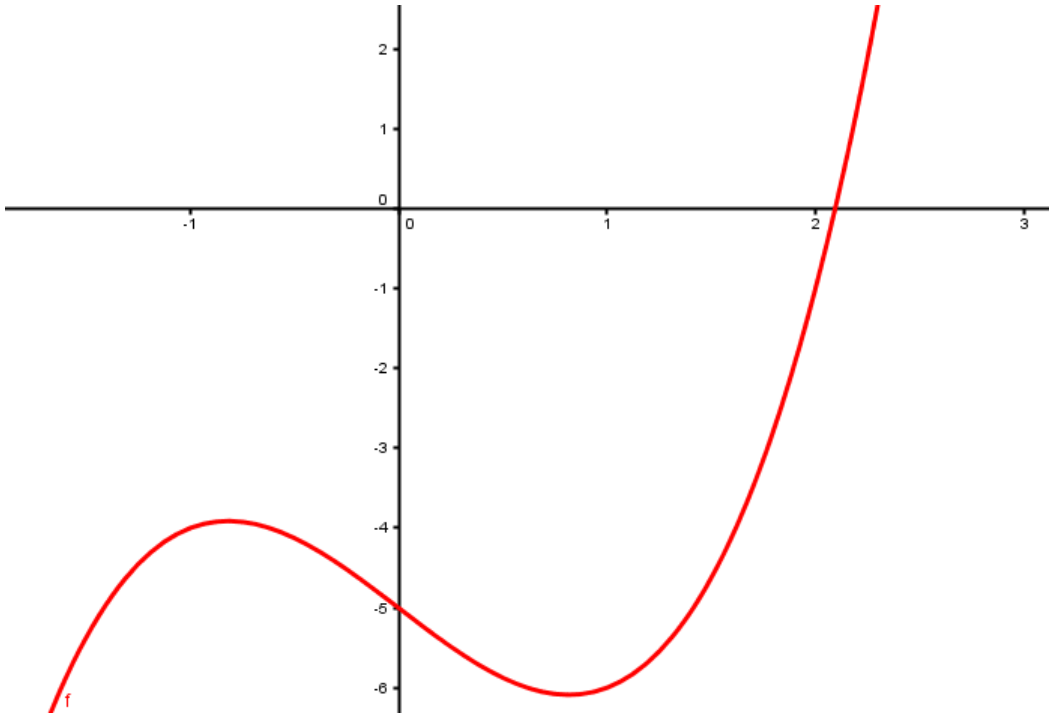


Figura 8 – Gráfico de $f(x) = x^3 - 2x - 5$ no intervalo $(-3,3)$.

3.2 Matriz hermitiana para a forma real simétrica

O análogo complexo de uma matriz real simétrica é a matriz hermitiana. Transformações de Jacobi podem ser usadas para encontrar os autovalores e autovetores, tal como redução de Householder a forma tridiagonal seguido por uma iteração QL.

Uma alternativa, é converter o problema hermitiano para um problema real simétrico: Se $C = A + iB$ é uma matriz hermitiana, então o problema de autovalor complexo $n \times n$ (14)

$$(A + iB) \cdot (u + iv) = \lambda(u + iv) \tag{3.2}$$

é equivalente ao $2n \times 2n$ problema real

$$\begin{bmatrix} A & -B \\ B & A \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} u \\ v \end{bmatrix}. \tag{3.3}$$

Observe que a matriz $2n \times 2n$ em (3.2) é simétrica: $A^T = A$ e $B^T = -B$ se C é hermitiana.

Correspondendo a dado autovalor λ, o vetor

$$\begin{bmatrix} -v \\ u \end{bmatrix} \tag{3.4}$$

é também um autovetor, como você pode verificar por escrito as duas equações matriciais implícita por (3.3). Assim, se $\lambda_1, \lambda_2, \cdots, \lambda_n$ são os autovalores de C, então os $2n$ autovalores do problema aumentado (3.3) são $\lambda_1, \lambda_1, \lambda_2, \lambda_2, \cdots, \lambda_n, \lambda_n$; cada, em outras palavras, é repetido duas vezes. Os autovetores são pares da forma $u + iv$ e $i(u + iv)$; isto é, eles são o mesmo até uma fase não essencial. Assim, vamos resolver o problema aumentado (3.3), e escolher um autovalor e autovetor de cada par. Estes dão os autovalores e autovetores da matriz original C.

Trabalhando com a matriz aumentada requer um fator duas vezes maior de armazenamento do que o armazenamento da matriz complexa original. Em princípio, um algoritmo complexo é também duas vezes mais eficiente em tempo de computacional do que a solução do problema aumentado.

3.3 Armazenamento de matrizes

Matrizes simétricas, hermitianas ou triangulares podem ser armazenadas de forma mais compacta, se o triângulo relevante for armazenado por colunas em um *array* de uma dimensão (15).

Observe a matriz A, triangular superior,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{bmatrix}, \tag{3.5}$$

seria armazenada em um *array* da seguinte forma

$$a_{11}\,\underbrace{a_{12}a_{22}}\,\underbrace{a_{13}a_{23}a_{33}}\,\underbrace{a_{14}a_{24}a_{34}a_{44}}, \tag{3.6}$$

caso a matriz A fosse triangular inferior

$$A = \begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}, \tag{3.7}$$

o armazenamento no *array* ficaria assim

$$\underbrace{a_{11}a_{21}a_{31}a_{41}}\,\underbrace{a_{22}a_{32}a_{42}}\,\underbrace{a_{33}a_{43}}\,a_{44}. \tag{3.8}$$

Note que para matrizes reais ou complexas simétricas, armazenar a matriz triangular superior por colunas é equivalente à armazenar a matriz triangular inferior por linhas; armazenar a matriz triangular inferior por colunas é equivalente a armazenar a matriz triangular superior por linhas. Para matrizes hermitianas complexas, armazenar a triangular superior por colunas é equivalente à armazenar o conjugado da triangular inferior por linhas; armazenar a triangular inferior por colunas é equivalente à armazenar o conjugado da triangular superior por linhas (15).

3.4 Métodos para cálculo de autovalores

Há diversos métodos para a determinação de autovalores de matrizes tridiagonais simétricas, faremos uma revisão dos principais métodos numéricos utilizados em conjunto com seus algoritmos voltados para matrizes tridiagonais.

Na resolução de problemas de autovalores que envolvam matrizes tridiagonais simétricas temos basicamente três métodos: o método interativo QR e suas diversas variantes, os métodos que adotam estratégias do tipo dividir e conquistar e os métodos bissecção/multissecção (16).

Em (17) os autores definem critérios para comparação entre os métodos utilizados na determinação de autovalores em casos de matrizes simétricas, tridiagonais ou

densas, onde os critérios observados para comparação foram: precisão, a susceptibilidade ao *overflow*, a velocidade de execução e necessidade de espaço de armazenamento.

Em relação as condições de comparação entre os métodos numéricos, os autores distinguem o caso sequencial do paralelo em relação as matrizes tridiagonais densa e os requisitos do usuário quando se deseja calcular todos autovalores ou parte dos mesmos. As principais conclusões a que os autores chegaram foram as seguintes:

- O método da bissecção/multissecção é o mais preciso dos três métodos analisados, garantindo que a precisão dos resultados só depende das limitações que arquitetura determina e dos dados de entrada, e não depende do algoritmo.
- O único método que permite determinar uma parte do espectro é o método da bissecção/multisecção, apesar de ser mais lento em alguns casos que os outros métodos, especificamente se desejar determinar uma parte significativa dos autovalores.
- Para determinar apenas os autovalores, o método de QR é o mais indicado.
- Em relação a paralelização somente os métodos bissecção/multissecção e o dividir e conquistar admitem uma paralelização eficiente, caso seja necessário determinar uma grande quantidade de autovalores o método divide e conquistar é o mais indicado.
- Em relação ao armazenamento no caso de matrizes tridiagonais, todos os métodos utilizam um espaço O(n), em relação ao cálculo dos autovalores.

Levando em conta as considerações anteriores os autores recomendam os algoritmos que tenham maior precisão nos resultados do que os que tenham melhor desempenho, além disso os mesmos ainda dizem: a rotina utilizada deve ser sempre a mais precisa e que não exija um espaço grande de armazenamento. No caso de matrizes tridiagonais o método a ser escolhido é o método da bissecção/multissecção.

3.5 Bissecção e Sequência de Sturm

A utilização do método da bissecção em matrizes tridiagonais simétricas é baseado em uma característica desse tipo de matriz, que é a obtenção do polinômio característico com um custo muito reduzido (16).

Na matriz tridiagonal simétrica o valor do polinômio característico não será obtido através de seus coeficientes, o mesmo será obtido a partir dos elementos da matriz.

Dada uma matriz tridiagonal simétrica $T \in \mathbb{R}^{n\times n}$

$$T_n = \begin{bmatrix} a_1 & b_1 & 0 & \cdots & 0 \\ b_1 & a_2 & b_2 & 0 & \vdots \\ 0 & b_2 & a_3 & \ddots & 0 \\ \vdots & 0 & \ddots & \ddots & b_{n-1} \\ 0 & \cdots & 0 & b_{n-1} & a_n \end{bmatrix} \tag{3.9}$$

e um número real λ, defineremos o polinômio caraterístico de T em λ, como

$$p_n(\lambda) = \det(T - \lambda I) \tag{3.10}$$

Utilizaremos a sequência de Sturm para facilitar o cálculo das raízes do polinômio característico em qualquer ponto, a sequência é definida assim:

$$\begin{aligned} p_0(\lambda) &= 1 \\ p_1(\lambda) &= a_1 - \lambda \\ p_i(\lambda) &= (a_i - \lambda)p_{i-1}(\lambda) - b_{i-1}^2 p_{i-2}(\lambda) \;\; i = 2, 3, \cdots, n. \end{aligned} \tag{3.11}$$

Seja T_i uma submatriz de T de tamanho $i \times i$, a sequência de Sturm baseia-se no cálculo recursivo do polinômio característico destas submatrizes, dessa forma para calcular o polinômio característico da matriz T, é suficiente calcular o polinômio característico das submatrizes.

O uso das sequências de Sturm com o método da bissecção se baseia em uma propriedade que as relaciona com os autovalores da matriz T, em conjunto com a propriedade que nos diz que λ é um autovalor de T se $p_n(\lambda) = 0$.

Antes de enunciar essa propriedade precisamos conhecer dois teoremas que relacionam os autovalores das distintas submatrizes de T (16).

Teorema 3.2 (Propriedade do entrelaçamento não estrito) *Seja A_{r-1} e A_r as submatrizes principais de tamanho $(r-1)$ e r de uma matriz simétrica $A \in \mathbb{R}^{n\times n}$. Se*

$$\mu_1 \leq \mu_2 \leq \cdots \leq \mu_{r-1}$$

são os autovalores de A_{r-1} e

$$\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_r$$

são os autovalores de A_r, então

$$\lambda_1 \leq \mu_1 \leq \lambda_2 \leq \mu_2 \leq \cdots \leq \lambda_{r-1} \leq \mu_{r-1} \leq \lambda_r.$$

A demostração pode ser encontrada em (18, 19).

A propriedade citada no teorema anterior nos diz que os autovalores de uma submatriz principal de A servem como separadores para uma matriz de ordem imediatamente inferior. Mas se consideramos $\mu_0 = -\infty$ e $\mu_r = +\infty$ a recíproca da propriedade se cumpre, ou seja, os autovalores de uma matriz de tamanho $r-1$ servem de separadores dos autovalores de uma submatriz de tamanho r, essa separação nem sempre é estrita, no caso de matrizes tridiagonais simétricas irredutíveis essa separação funciona rigorosamente.

Uma matriz tridiagonal simétrica é irredutível quando todos os elementos de sua subdiagonal, e também da sua superdiagonal são diferentes de zero, isto é, $b_i \neq 0$ $(i = 1, 2, \cdots, n-1)$.

A condição de que a matriz seja irredutível não afeta a generalidade do resultado apresentado a seguir, no caso de existir algum elemento não-nulo na diagonal, pode-se dividir a matriz em blocos triangulares irredutíveis nos quais podem se aplicar a propriedade (16).

Teorema 3.3 (Propriedade do entrelaçamento estrito) *Seja $T \in \mathbb{R}^{n\times n}$ uma matriz tridiagonal simétrica irredutível e seja T_{r-1} e T_r duas de suas submatrizes principais cujos autovalores são respectivamente:*

$$\{\mu_1, \mu_2, \cdots, \mu_{r-1}\}$$

e

$$\{\lambda_1, \lambda_2, \cdots, \lambda_r\},$$

então se:

$$\lambda_1 < \mu_1 < \lambda_2 < \mu_2 < \cdots < \lambda_{r-1} < \mu_{r-1} < \lambda_r.$$

A demonstração pode ser encontrada em (18, 8).

A propriedade do teorema anterior pode ser verificada a partir da definição da sequência de Sturm. Suponha que por redução ao absurdo que $p_{r-1}(\mu) = 0$ e $p_r(\mu) = 0$ para algum μ, a partir de (3.11) com $i = r$ de modo que b_r seja diferente de zero, $p_{r-2}(\mu) = 0$. Continuando o raciocínio para $i = r-1$, chegamos que $p_{r-3}(\mu) = 0$ e seguindo adiante até $r = 0$, concluímos que $p_0(\mu) = 0$, o que contradiz a definição de $p_0(\mu) = 1$ (16).

Teorema 3.4 (Propriedade da sequência de Sturm) *Seja $T \in \mathbb{R}^{n\times n}$ uma matriz tridiagonal simétrica irredutível, e λ um número real, então o número de trocas de sinais na sequência de Sturm*

$$\{p_0(\lambda), p_1(\lambda), \cdots, p_n(\lambda)\} \tag{3.12}$$

é igual ao número de autovalores de T que são menores que λ. Nessa propriedade usaremos a convenção de que $p_r(\lambda)$ possui sinal oposto a $p_{r-1}(\lambda)$, se $P_r(\lambda) = 0$, para que seja válida para qualquer valor de λ.

A demonstração pode ser encontrada em (18).

Definiremos uma função $neg_n(\lambda)$, que será utilizada em diversos raciocínios posteriores, esta função retorna o número de trocas de sinais da sequência de Sturm (3.12) em um ponto λ.

Foi Wallace Givens (20) o primeiro autor que percebeu a possibilidade de utilizar a sequência de Sturm para calcular os autovalores de uma matriz tridiagonal simétrica. Como foi definido na equação (3.11), o cálculo da sequência de Sturm em um ponto qualquer α e a função associada $neg_n(\lambda)$, torna possível a divisão do espectro da matriz T bem como definir quantos e quais autovalores da matriz são menores que α, e quantos e quais são menores que este ponto de divisão.

Se aplicarmos o raciocínio do parágrafo anterior a dois pontos α e β, podemos determinar quais autovalores estão contidos no intervalo $[\alpha, \beta)$. Se definirmos $na = neg_n(\alpha)$ e $nb = neg_b(\beta)$, onde os autovalores da matriz T que estão contidos no referido intervalo são:

$$\lambda_{na+1} < \lambda_{na+2} < \cdots < \lambda_{nb}.$$

Se o intervalo $[\alpha, \beta)$ for dividido em um ponto μ, podemos definir a posição de cada um dos autovalores anteriores sem a necessidade do cálculo de $nc = neg_n(\mu)$. É óbvio que a aplicação de um esquema de bissecção como continuação do raciocício anterior, nos permite calcular o autovalor λ_i contido no intervalo incial, com a precisão limitada apenas pelo número de iterações executadas pela bissecção, precisão da arquitetura utilizada e dos dados do problema.

A partir da recorrência (3.11) que possibilita calcular a sequência de Sturm, é óbvio que quando n for muito grande podemos ter graves problemas de *overflow*. Esse problema foi detectado por Barth, Martin e Wilkinson (21) que propuseram o uso de uma sequência de Sturm modificada para resolver este problema, listada a seguir:

$$q_i(\lambda) = \frac{p_i(\lambda)}{p_{i-1}(\lambda)} \quad i = 1, 2, \cdots, n. \tag{3.13}$$

Da Equação (3.11) percebe-se que para calcular a função anterior, podemos usar a recorrência a seguir de primeira ordem:

$$\begin{aligned} q_0(\lambda) &= 1 \\ q_1(\lambda) &= a_1 - \lambda \\ q_i(\lambda) &= (a_i - \lambda) - \frac{b_{i-1}^2}{q_{i-1}(\lambda)} \quad i = 2, 3, \cdots, n. \end{aligned} \tag{3.14}$$

Usando os quocientes sucessivos na sequência original de Sturm praticamente se elimina o problema de *overflow* que surgia ao calculá-la.

Nota-se também que o número de trocas de sinais entre os sucessivos elementos de uma sequência de Sturm denotada por $neg_n(\lambda)$ concorda com o número de sinais negativos apresentados em (3.13) (16).

Na versão modificada da sequência de Sturm pode surgir um problema de divisão por zero, durante a execução da Equação (3.14). Diversos autores propuseram soluções para esse problema.

Os autores Basermann e Weidner (22) propuseram a substituição do termo $q_{i-1}(\lambda)$ por uma versão modificada desse termo da seguinte forma:

$$\tilde{q}_{i-1}(\lambda) = q_{i-1}(\lambda) \pm \varepsilon,$$

onde ε é um número muito pequeno (depende da precisão do computador), que se soma a $q_{i-1}(\lambda) \geq 0$ ou subtrai em outros casos. Com esta pequena modificação se evita a possibilidade de ocorrer uma divisão por zero durante a execução da recorrência (3.14) (16).

O autor De Ros (16) propôs uma solução muito semelhante que consiste em trocar $q_{i-1}(\lambda)$ por um valor ε muito pequeno quando a função assumir o valor zero durante a execução da recorrência. Esta modificação equivale a trocar o termo a_{i-1} da matriz T pelo termo $a_{i-1} + \varepsilon$, o processo envolve uma pequena perturbação em um elemento da matriz T (16).

Outra solução foi proposta pelos autores Li e Zeng (23), que consiste na aplicação das seguintes modificações se a função $q_i(\lambda)$ for nula:

$$\begin{aligned} &\text{Se} \quad q_i(\lambda) = 0 \;\; (a_i = \lambda) \quad \text{então} \quad q_i(\lambda) = b_1^2 \varepsilon^2. \\ &\text{Se} \quad q_i(\lambda) = 0 \;\; (i > 1) \quad \text{então} \quad q_i(\lambda) = \frac{b_{i-1}^2}{q_{i-1}(\lambda)} \varepsilon^2. \end{aligned}$$

Se $a_i = \lambda$ a modificação realizada é a substituição de a_1 por $a_1 + b_1^2\varepsilon^2$. Se $q_i(\lambda) = 0$ com $i > 1$, a correção deverá ser feita substituindo b_{i-1} por $b_{i-1}(1 - \varepsilon^2)^{1/2}$ (16).

A adoção de uma destas técnicas para modificar a recorrência (3.12), acarretará perturbações muito pequenas nos dados do problema, ou seja, nos elementos da matriz. Levando em conta o bom condicionamento do problema de cálculo dos autovalores de matrizes simétricas e o uso do método da bissecção, as modificações da recorrência não afetam em um grau apreciável a precisão dos resultados obtidos (16).

3.6 Estabilidade

Uma propriedade do problema de calcular os autovalores de matrizes simétricas é o bom condicionamento, isto é, variações pequenas nos dados do problema causam modificações pequenas nos resultados. Essa característica é representada no teorema abaixo (16).

Teorema 3.5 (Estabilidade de autovalores de matrizes simétricas) *Seja $A \in \mathbb{R}^{n\times n}$ uma matriz simétrica. Seja $\tilde{A} = A + E$, com E uma matriz de pertubação simétrica, e sejam $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ os autovalores de A e $\tilde{\lambda_1} \leq \tilde{\lambda_2} \leq \cdots \leq \tilde{\lambda_n}$ os de $\tilde{A}$, então*

$$|\tilde{\lambda_i} - \lambda_i| \leq \|E\|_2 \;\; i = 1, 2, \cdots, n.$$

A demonstração pode ser vista em (7).

Com a demonstração da estabilidade do método da bissecção temos a garantia que mediante a pequeno erros apresentados nos dados, os autovalores calculados estão bem próximos dos autovalores exatos da matriz considerada.

Em (18) se demonstra a grande estabilidade do método da bissecção puro utilizando sequência de Sturm (3.11). De maneira objetiva se demonstra que o cálculo do determinante

$$p_n(\lambda) = \det(T - \lambda I)$$

nos retorna o valor exato do polinômio característico de uma matriz perturbada $T + \delta T$, onde

$$|\delta a_i| \leq 3,01\varepsilon(|a_i| + |\lambda|)$$
$$|\delta b_i| \leq 1.51\varepsilon|b_i|$$

sendo ε um número muito pequeno. Se

$$-\|T\|_\infty \leq \lambda_i \leq +\|T\|_\infty$$

pelo o teorema da monotocidade de Weyl (19) temos

$$|\tilde{\lambda_i} - \lambda_i| \leq \|\delta T\|_\infty = max\{|\delta b_{i-1}| + |\delta a_i|\} \leq 3,01\varepsilon(|a_i| + |\lambda|) + 3,02\varepsilon|b_i| \cong$$
$$\cong 3,02\varepsilon(|a_i| + |b_i| + |\lambda|).$$

3.7 Algoritmo básico do método da bissecção

Algoritmos baseados nos métodos da bissecção/multissecção possuem privilégios sobre outros métodos, como o iterativo QR e os métodos do tipo dividir e conquistar, por apresentar grande flexibilidade. De fato, ao fazer uso desse algoritmo podemos calcular uma parte dos autovalores de uma matriz ou todos de maneira seletiva (16). O método da bissecção pode ser utilizado para responder as seguintes condições abaixo.

1. Calcular um autovalor qualquer λ_i a partir do indíce i num conjunto ordenado
$$\lambda_1 < \lambda_2 < \cdots < \lambda_n.$$
2. Calcular todos os autovalores de uma matriz contidos em um intervalo real delimitado por (a, b).
3. Calcular todos os autovalores de uma matriz onde seus índices podem ser encontrados entre os valores inteiros $\{i, j\}$.
4. Calcular todos os autovalores de uma matriz.

Dada uma matriz tridiagonal simétrica $T \in \mathbb{R}^{n\times n}$ e um número real λ que não seja autovalor da mesma. Temos um algoritmo que permite calcular a função $neg_n(\lambda)$. Em primeiro lugar veremos como resolver o problema proposto no item 1), em seguida expandiremos o método para compreender como resolver o item 3). É óbvio que os itens 2) e 4) podem ser resolvidos com uma variação do item 3) (16).

3.8 Busca de um único autovalor

A aproximação através do uso da função $neg_n(\lambda)$ se baseia na aplicação dessa função em um ponto α que determina a posição de um autovalor qualquer em relação ao ponto α da reta real. Se aplicarmos a função em dois pontos quaisquer α e β, teremos os valores naturais na e nb, então o número de autovalores de T, que estão no intervalo $[\alpha, \beta)$ é dado por $nb - na$.

Suponha que os autovalores da matriz T estejam ordenados da maneira abaixo.
$$\lambda_1 < \lambda_2 < ... < \lambda_n$$
e desejamos calcular um valor qualquer λ_i de T.

A aplicação do método da bissecção consiste nesse caso em iniciar em um intervalo (α, β) onde esteja contido o autovalor que se deseja calcular, e utilizar a função $neg_n(\lambda)$ para delimitar a posição de λ_i nesse intervalo com a precisão definida.

Primeiro necessitamos determinar o intervalo (α, β) que contém o autovalor λ_i a ser calculado, ou seja,

1. $neg_n(\alpha) < i$;
2. $neg_n(\beta) \geq i$.

Os pontos α e β podem ser obtidos por tentativa e erro, testando vários pontos que obedeçam as condições anteriores ou podemos fazer uso do teorema de Gershgorin associado a matriz T (16).

Teorema 3.6 (Discos de Gershgorin) *Seja uma matriz $A \in \mathbb{C}^{n\times n}$, então se $X^{-1}AX = D + F$ com $D = diag(d_1, d_2, \cdots, d_n)$ e F uma matriz com elementos diagonais nulos, então*

$$\lambda(A) \subset \bigcup_{i=1}^{n} D_i$$

onde

$$D_i = \left\{ z \in \mathbb{C}^{n\times n} : |z - d_i| \leq \sum_{j=1}^{n} |f_{ij}| \right\} \quad i = 1, 2, \cdots, n$$

são os discos de Gershgorin de A.

A demonstração pode ser encontrada em (8). Este teorema mostra que cada autovalor da matriz A está contido dentro de um dos discos de Gershgorin, mas não garante que os discos são disjuntos ou que contenham um único autovalor. Para a matriz $A \in \mathbb{R}^{n\times n}$, caso seja uma matriz simétrica, os discos de Gershgorin passam a ser intervalos da reta real, sendo definido o $i-$ésimo intervalo da seguinte forma

$$\left[d_i - \sum_{j=1}^{n} |f_{ij}|, \;\; d_i + \sum_{j=1}^{n} |f_{ij}| \right].$$

Uma maneira de assegurar a existência de um autovalor qualquer dentro de um intervalo é usar os limites superior e inferior de todos os discos de Gershgorin. Dessa forma assegura-se que o intervalo obtido contenha todos os autovalores da matriz e consequentemente o autovalor procurado (16).

Determinando o intervalo inicial desta forma no caso de matrizes tridiagonais simétricas, os extremos desse intervalo tomam a seguinte forma

$$\alpha = \min_{1\leq i\leq n-2} \{a_i - (|b_i| + |b_{i+1}|)\}$$

$$\beta = \max_{1\leq i\leq n-2} \{a_i - (|b_i| + |b_{i+1}|)\}.$$

Definido o intervalo inicial (α, β), que contém o autovalor λ_i a ser determinado, iniciaremos a aplicação do método da bissecção a partir do intervalo inicial. Usamos a função $neg_n(\lambda)$ para definir qual dos subintervalos resultantes da aplicação da bissecção, se deve escolher para a próxima iteração, subintervalo este que deverá conter o autovalor procurado. Esta repetição da bissecção se dará até que o autovalor procurado seja encontrado com a precisão desejada.

O comprimento do intervalo que contém cada um dos autovalores procurados é dividido por dois a cada iteração executada pelo método da bisseção, então após k

iterações temos a delimitação do autovalor dentro de um intervalo de comprimento $(\alpha - \beta)/2^k$, onde α e β são os limites do intervalo inicial (16).

Algoritmo 3.1: Algoritmo simples da bissecção

```
1  Programa bissecção (i, λ_i)
2  Calcular um intervalo (a, b) tal que
   neg_n(a) < i e neg_n(b) ≥ i
3  repita
4      c = (a + b)/2;
5      se neg_n < i então
6          a=c;
7          senão
8              b=c
9          fim
10     fim
11 até |b − a| < cota;
12 λ_i = (a + b)/2
```

Aplicando diretamente esse algoritmo obtemos qualquer autovalor da matriz T, com garantia da convergência do método da bissecção para o autovalor que se desejado. Mas esse método apresenta o inconveniente de ter uma convergência lenta para o autovalor buscado. Seria interessante obter uma maneira de acelerar a convergência do Algoritmo 3.1, isso pode ser feito combinando o algoritmo simples da bissecção com outra técnica para busca de raízes que possua uma velocidade de convergência maior.

Para utilizar esse novo método voltaremos a sequência (3.11), combinada com a matriz T, em um ponto λ. Usamos essa sequência para definir uma função nova $q_n(\lambda)$ que é dada exatamente pelo último valor no ponto λ, isto é, dado um número real λ qualquer temos um algoritmo para obter a sequência (3.11) neste ponto, podemos conseguir o valor da função $q_n(\lambda)$ (16).

De acordo com a definição de autovalor, um número real λ é autovalor da matriz T, se e somente se

$$\det(T - \lambda I) = 0.$$

Tendo como base $(A - \lambda I) = LDL^t$, λ será autovalor de T se e somente se

$$\det(LDL^t) = 0.$$

Dado que L é uma matriz unidade triangular inferior, ou seja, λ é um autovalor de T se e somente se

$$det(D) = 0$$

e portanto se

$$\prod_{i=1}^{n} q_i(\lambda) = 0.$$

Se usarmos a função

$$p_m(\lambda) = \det(T_m - \lambda I_m) \quad m = 1, 2, \cdots, n$$

com T_m sendo a submatriz principal de T com dimensão $m \times m$, então

$$p_m(\lambda) = \prod_{i=1}^{m} q_i(\lambda) \quad m = 1, 2, \cdots, n$$

e portanto

$$q_n(\lambda) = \frac{p_n(\lambda)}{p_{n-1}(\lambda)}. \tag{3.15}$$

Logo qualquer valor de λ que anule a função $q_n(\lambda)$ será autovalor da matriz T, ou seja, o problema de determinar os autovalores da matriz tridiagonal simétrica T coincide com as raízes da função $q_n(\lambda)$.

Voltando ao caso de determinar o autovalor λ_i de T, suponha que de alguma forma tenhamos determinado o intervalo (α, β) que pode ou não ter um autovalor da matriz T. O autovalor λ_i poderá ser a única raiz da função $q_n(\lambda)$ neste intervalo. Nesse caso podemos usar uma técnica de extração de raízes com convergência mais rápida que a bissecção simples na determinação de qualquer autovalor da matriz T.

Entretanto o problema da determinação do intervalo (α, β) que contenha o autovalor λ_i da matriz T. A partir desse ponto o problema será denominado isolamento do autovalor da matriz T. Após o isolamento iremos fazer a extração dos autovalores da matriz T, que será denominado extração do autovalor (16).

3.8.1 Critério de parada no algoritmo da bissecção

Suponha que ao utilizarmos o método da bissecção define-se uma sequência de aproximações $\left\{\tilde{\lambda}_i\right\}_{k=1}$ cada vez mais próximo do autovalor λ_i. Com o estudo da seção 3.6 pode-se definir o seguinte critério de parada no algoritmo da bissecção:

$$|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}| \leq max\left\{\delta, |\tilde{\lambda}_i^{(k)}|\varepsilon\right\}, \tag{3.16}$$

onde ε representa a precisão da máquina e δ é uma cota de erro absoluta definida como:

$$\delta = 2,5\varepsilon max\left\{|b_i| + |b_{i+1}|\right\}.$$

Dessa forma, consideramos que um autovalor tenha convergido quando a diferença entre duas aproximações seguidas for menor que uma cota definida, tanto para um erro absoluto δ, ou para o erro relativo da última aproximação obtida $|\tilde{\lambda}_i^{(k)}|\varepsilon$.

Outros autores como Li e Zeng (23) fazem uso de outro critério de parada similar menos restritivo. Estes autores comprovaram na prática que o seguinte critério

nunca falha, desde que obedeça (3.16) (16):

$$\frac{|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}|^2}{|\tilde{\lambda}_i^{(k-1)} - \tilde{\lambda}_i^{(k-2)}|} \leq 2,5\varepsilon max\{\ |b_i|, |b_{i-1}|\} + |\tilde{\lambda}_i^{(k)}|\varepsilon.$$

3.8.2 Isolamento do autovalor

Iremos nos preocupar com o problema de isolar o autovalor λ_i da matriz T. Este problema pode ser expresso da forma na definição abaixo.

Definição 3.2 (Problema de isolamento) *Dada uma matriz tridiagonal simétrica $T \in \mathbb{R}^{n\times n}$, dados dois números reais positivos ε representando a precisão da máquina e δ uma cota de erro, e dados dois números reais $\alpha_0 < \beta_0$ tal que o intervalo (α_0, β_0) contém o autovalor λ_i de T (e possivelmente outros), calcular dois números reais α e β tal que $\alpha_0 \leq \alpha < \beta \leq \beta_0$ contenha somente o autovalor λ_i de T, ou seja, se cumpra a seguinte desigualdade: $|\beta - \alpha| \leq max\{\delta, max\{|\alpha|, |\beta|\}\varepsilon\}$ (16).*

A resolução deste problema nos leva, a partir de um intervalo inicial (α_0, β_0) que contém o autovalor junto com outros, a outro intervalo isolado (α, β) que somente se encontra λ_i. Para que isso ocorra esse intervalo isolado deve cumprir o seguinte critério:

$$neg_n = (\alpha) = i - 1 \ \text{ e } \ neg_n(\beta) = i. \tag{3.17}$$

Na realidade temos estabelecido um segundo critério para resolver o problema de isolamento. Segundo este, se não podemos delimitar um intervalo que contém somente o autovalor procurado e o intervalo alcançou a cota mínima, pararemos o algoritmo e vamos considerar que foi encontrado um *cluster* de autovalores.

Definição 3.3 (*Cluster* de autovalores) *Definiremos um cluster de autovalores $\lambda_i < \cdots < \lambda_j$, $j > i$, como um grupo de autovalores consecutivos que se encontram em um intervalo de comprimento menor que uma cota dada Δ (16).*

Pode-se considerar que diversos autovalores formam um *cluster*, se são aritmeticamente distintos, mas numericamente indistinguíveis (16). A definição anterior impõe uma condição menos restritiva e exige somente que sua proximidade seja menor que uma cota muito pequena Δ.

Neste caso consideramos que o intervalo (α, β) tem um *cluster* de autovalores se

$$neg_n(\alpha) = i - 1, \quad neg_n(\beta) = j \quad e \quad j > i + 1$$

mas

$$|\beta - \alpha| \leq max\left\{\delta, max\left\{|\alpha|, |\beta|\right\}\varepsilon\right\}.$$

Como o *cluster* de autovalores contido no intervalo (α, β) estará formado por $\lambda_i < \lambda_{i+1} < ... < \lambda_j$.

No caso em que encontrarmos um *cluster* finalizaremos a aproximação dos autovalores que se formam neste ponto, isto é, não continuaremos com a fase de extração ao considerar que todos os autovalores incluídos no *cluster* tenham sidos aproximados com a precisão desejada.

No desenvolvimento da fase de isolamento, esta se baseará na aplicação de sucessivas bisecções a partir do intervalo inicial, desde que cumpra uma das condições citadas anteriormente.

Algoritmo 3.2: Algoritmo do isolamento

Entrada: $(a, b, \delta, \varepsilon)$, calcular intervalo (a_0, b_0) tal que $neg_n(a_0) < i$ e $neg_n(b_0) \geq i$, onde $a = a_0$ e $b = b_0$.

Saída: o intervalo (a, b)

```
1 enquanto (neg_n(a) = i − 1 e neg_n(b) = i)
  ou (|b − a| ≤ max{δ, max{|a|, |b|}ε}) faça
2     c = (a + b)/2;
3     se neg_n(c) < i então
4         a=c
5     senão
6         b = c
7     fim
8 fim
```

3.8.3 Extração dos autovalores

Na fase de extração das raízes partiremos de uma intervalo isolado e aproximaremos o autovalor λ_i até assegurar que o mesmo tenha uma dada precisão. O problema de extração pode ser expresso da forma apresentada na definição seguinte.

Definição 3.4 (Problema de extração) *Dada uma matriz tridiagonal simétrica $T \in \mathbb{R}^{n \times n}$, dados dois números reais positivos ε representando a precisão da máquina e δ uma cota de erro, e dados dois números reais $\alpha < \beta$ tal que o intervalo (α, β) contém somente o autovalor λ_i de T, calcular uma sucessão de aproximações ao autovalor λ_i de T da forma $\{\tilde{\lambda}_i^{(l)}\}_{l=1}^{k}$ tal que $|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}| \leq max\{\delta, |\tilde{\lambda}_i^{(k)}|\varepsilon\}$ (16).*

Na extração do autovalor no intervalo isolado usaremos um algoritmo de busca de raízes que convirja mais rápido que o método da bissecção pura. A possibilidade de acelerar o método de biseccção foi observada por Wilkinson (18), que propôs a utilização de outros métodos como o método iterativo de Newton com convergência quadrática e o método iterativo de Laguerre com convergência cúbica.

Após Wilkinson (18) diversos autores propuseram métodos diferentes de aproximação de raízes durante a fase de extração. Parlett (19) sugeriu o uso do método da secante ou alguma varição do mesmo, Pereyra e Scherer (24) propuseram uma combinação adequada do método da bisecção, o método da secante e o de Newton, chegando a conclusão de que o método da secante é globalmente mais eficiente, mas que poderia se obter substanciais benefícios se o substituirmos pelo método de Newton nas proximidades do autovalor. Lo, Philippe e Sameh (25) propuseram como alternativas ao método da bissecção, o método de Newton e o *zeroin* (combinação do método da secante e da bissecção). Baker (26) estudou três possíveis técnicas: a interpolação cúbica polinomial e a iteração de Newton de primeira e segunda ordem, chegando a conclusão que o método de Newton sempre oferece melhores resultados. Finalmente, Ralha (27) propôs uma modificação do método de Newton que o autor denominou *zeroinNR*.

Levando em conta a variedade de métodos utilizados ou propostos por diversos autores, consideramos importante realizar um estudo comparativo de alguns deles com a finalidade de determinar qual oferece melhores resultados para um conjunto representativo de matrizes de prova. Em relação a esse respeito temos implementado e comparado alguns dos métodos utilizados por outros autores.

Recordaremos alguns métodos aplicados na determinação de autovalores de matrizes tridiagonais simétricas.

3.8.3.1 Método da secante

O método da secante se baseia em fazer a aproximação da raiz da função $q_n(\lambda)$(3.15), procurando a intersecção da reta secante com o eixo real da função em dois pontos $y_1 = q_n(x_1)$ e $y_2 = q_n(x_2)$, levando em conta que a raiz está contida em um intervalo (x_1, x_2).

Para garantir que a função converge para a raiz procurada após várias iterações com o método da secante é necessário que a função seja contínua no intervalo de busca.

No caso da função $q_n(\lambda)$ (3.15), temos que a mesma é contínua em uma porção de intervalos definidos pelas raízes de $p_{n-1}(\lambda)$. Portanto a função apresenta uma série de pontos de descontinuidade nos autovalores da submatriz principal T_{n-1} de T, $\mu_1 < \mu_2 < \cdots < \mu_{n-1}$.

Ao executar o método da secante existe a necessidade de obedecer uma condição adicional a (3.17) para que no intervalo (α, β), não esteja contido nem um dos pontos μ_i. Para que isto ocorra além de (3.17) é necessário que satisfaça a condição seguinte:

$$q_n(\alpha) > 0 \quad e \quad q_n(\beta) < 0. \tag{3.18}$$

Definição 3.5 (Autovalor defeituoso) *Seja $A \in \mathbb{R}^{n\times n}$ uma matriz simétrica. Diz-se que um número real λ não é defeituoso em relação a A, se e somente se, não é autovalor de nenhuma das submatrizes principais de A, $A_k(k = 1, ..., n-1)$. Logo λ é um autovalor defeituoso se for autovalor de qualquer uma das submatrizes principais de A.*

Estas condições são formalizadas no teorema abaixo.

Teorema 3.7 (Isolamento estrito) *Suponha que α e β são dois números reais não defeituosos em relação a matriz T, e que (α, β) contém exatamente um autovalor (com multiplicidade um) de T. Suponha que nem α nem β sejam autovalores de T. Então (α, β) não contém autovalores de T_{n-1} se e somente se $q_n(\alpha) > 0$ e $q_n(\beta) < 0$.*

A demonstração pode ser encontrada em (28).

Este teorema nos possibilita definir aproximadamente a forma da função em relação aos autovalores das matrizes T e T_{n-1}. Denotamos os autovalores da matriz T_{n-1} como $\mu_1, \mu_2, \cdots, \mu_{n-1}$ e os autovalores da matriz T por $\lambda_1, \lambda_2, \cdots, \lambda_n$. Pelo teorema do entrelaçamento estrito temos

$$\lambda_1 < \mu_1 < \lambda_2 < \mu_2 < \cdots < \lambda_{n-1} < \mu_{n-1} < \lambda_n.$$

Assim os autovalores da matriz T, exceto o primeiro e o último, estão contidos em intervalos, cujos extremos dos mesmos são autovalores de T_{n-1}. A função $q_n(\lambda)$ é zero para um λ_i e que passa de um valor positivo a negativo nestes pontos. Sabemos também que não está definida nos pontos μ_i, visto que estes anulam o denominador da função.

Para assegurar o funcionamento de (3.18) e evitar uma sucessão de etapa do método da bissecção durante o isolamento, podemos utilizar o Algoritmo 3.3.

Algoritmo 3.3: Algoritmo do isolamento estrito

```
1  programa isolaestr (a, b, δ, ε)
2  Dado um intervalo (a, b) isolado
   (neg_n(a) = i − 1  e  neg_n(b) = i)
   ya = q_n(a)
3  yb = q_n(b)
4  enquanto [(ya < 0  ou  yb >
   0)  e  (|b − a| > max{δ, max(|a|, |b|)ε})]
                                          faça
5                                           c = (a + b)/2
6                                           y = q_(c)
7                                           nc = neg_n(c);
8                                           se nc = i − 1 então
9                                             a = c
10                                            ya = y
11                                          senão
12                                            b = c
13                                            yb = y
14                                          fim
15                                        fim
16                                        se |b − a| ≤ {δ, max(|a|, ||)ε} então
17                                          λ_i = (a + b)/2
18                                        fim
```

A rotina de isolamento pode terminar quando a amplitude do intervalo que contém o autovalor for menor que uma cota definida, nessa situação pode ocorrer que não seja possível distinguir um autovalor da matriz T de um autovalor da matriz T_{n-1}, mesmo que se tenha isolado um autovalor dos outros autovalores da matriz, estes autovalores encontram-se tão perto de um autovalor da submatriz T_{n-1} que numericamente não se pode diferenciá-los. Quando essa situação ocorrer temos um autovalor oculto (18, 19).

Observando a Figura 9, esse fenômeno ocorre quando um autovalor λ_i de T está muito perto de um ponto de descontinuidade μ_j da função $q_n(\lambda)$. Quando esse fenômeno ocorre a função $q_n(\lambda)$ tende a não ser zero nas proximidades da raiz procurada de T, ainda que a função $q_n(\lambda)$ possa tender a zero a função $q_{n-1}(\lambda)$ também o fará, de modo que a raiz de $q_n(\lambda)$ será ocultada pela raiz de $q_{n-1}(\lambda)$, daí o nome autovalor oculto.

O fenômeno dos autovalores ocultos é muito comum em matrizes tridiagonais simétricas, mesmo quando se gera os dados aleatoriamente.

A extração é baseada nos requisitos (3.17) e (3.18), tendo a certeza que o intervalo está isolado, o denominaremos (x_1, x_2), o mesmo se encontra entre os pontos μ_{i-1} e μ_i. De acordo com essas exigências e que o método da secante segue no processo de encontrar a intersecção da função nos pontos $y_1 = q_n(x_1)$ e $y_2 = q_n(x_2)$. Sendo

$$(y - y_1) = m(x - x_1)$$

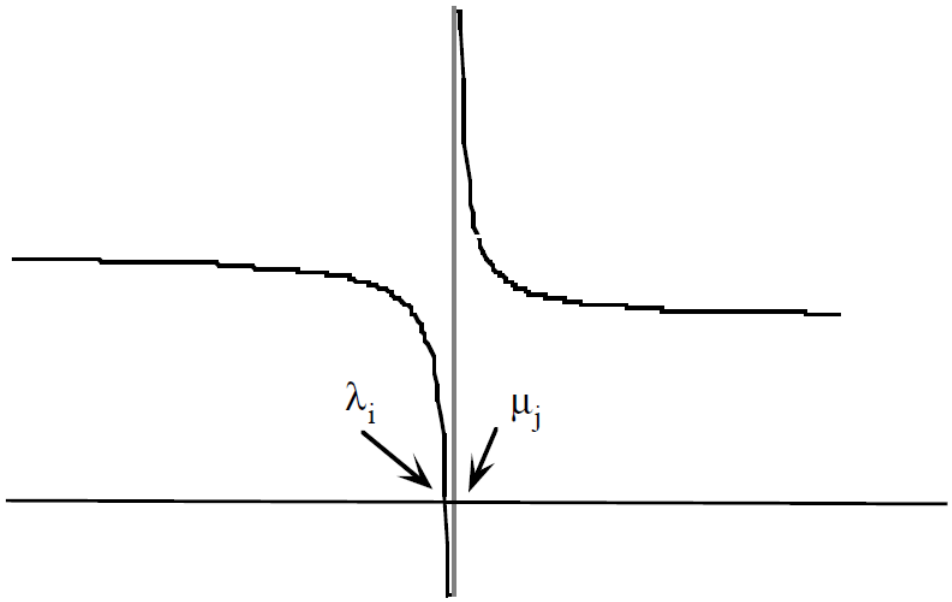


Figura 9 – Autovalor oculto.

Fonte: CONTELLES, J. M. B.(16)

a equação que define a reta secante com coeficiente angular m definido como

$$m = \frac{y - y_1}{x - x_1}$$

o ponto de intersecção da secante é dado por

$$x = x_1 - \frac{1}{m} y_1.$$

Se o valor de x for menor que o autovalor a determinar o mesmo assumirá o valor do extremo inferior do intervalo, ou seja, x_1, se ocorrer do valor x seja maior do que o autovalor a determinar o mesmo assumirá o valor do extremo superior do intervalo, ou seja, x_2. Esse processo de determinação do ponto de intersecção da reta secante e da redefinição do intervalo que contém o autovalor se repetirá até que esteja mais próximo do autovalor, obedecendo o limite da cota determinada.

Se em alguma das iterações do método a aproximação alcançada estiver fora do intervalo que contém o autovalor procurado, escolhemos como aproximação o ponto médio e damos prosseguimento ao método de extração. Por outro lado se a quantidade de iterações é maior que a cota definida, devemos parar o método para evitar um *loop* infinito (16).

Algoritmo 3.4: Algoritmo do método da secante

```
1  programa extrasec (x1, x2, y1, y2, δ, ε, x)
2  isola-extra(x1, x2, δ, ε)
3  repita
4      prevx = x
5      m = (y1 − y2)/(x1 − x2)
6      x = x1 − (1/m) ∗ y1
7      calcular y = f(x) ;
8      se y ∗ y1 > 0 então
9          prevy = y1
10         y1 = y
11         x1 = x
12     senão
13         prevy = y2
14         y2 = y
15         x2 = x
16     fim
17 até |x − prevx| < max{δ, |x|ε};
```

3.8.3.2 Método de Newton

Esse método se baseia na Equação (3.19) que faz o cálculo das aproximações sucessivas da raiz procurada.

$$x^{(k+1)} = x^{(k)} - \frac{f\left(x^{(k)}\right)}{f'\left(x^{(k)}\right)} \tag{3.19}$$

Especificamente quando se procura as raízes do polinômio característico, então $f(x) = p_n(x)$. A aplicação do método de Newton inclui, além do uso da função $p_n(x)$ o uso da sequência de Sturm, sendo necessário também calcular a função $p'_n(x)$. Derivando (3.11) obtemos a sequência a seguir:

$$\begin{aligned} p'_0(\lambda) &= 0 \\ p'_1(\lambda) &= -1 \\ p'_i(\lambda) &= (a_i - \lambda)p'_{i-1}(\lambda) - b^2_{i-1}p'_{i-2}(\lambda). \end{aligned} \tag{3.20}$$

A aplicação de (3.20) pode gerar diversos problemas de *overflow*. A solução encontrada na literatura é calcular alternativamente a sequência abaixo:

$$N_i = \frac{p'_i(\lambda)}{p_i(\lambda)} \quad i = 1, 2, \cdots, n.$$

Depois de calcular N_i sua inversa será a correção do método de Newton que permite obter a aproximação seguinte para a raiz. Calculando uma sequência para

(3.20) obtemos:

$$\begin{aligned} N_i &= (\alpha_i - \lambda)\frac{p'_{i-1}(\lambda)}{p_i(\lambda)} - \frac{p_{i-1}(\lambda)}{p_i(\lambda)} - b_{i-1}^2\frac{p'_{i-2}(\lambda)}{p_i(\lambda)} = \\ &= (\alpha_i - \lambda)\frac{p'_{i-1}(\lambda)p_i(\lambda)}{p_{i-1}(\lambda)p_i(\lambda)} - \frac{p_{i-1}(\lambda)}{p_i(\lambda)} - b_{i-1}^2\frac{p'_{i-2}(\lambda)p_{i-2}(\lambda)p_{i-1}(\lambda)}{p_{i-2}(\lambda)p_{i-1}(\lambda)p_i(\lambda)} = \\ &= \frac{1}{q_i(\lambda)}\left[(\alpha_i - \lambda)N_{i-1} - 1 - \frac{b_{i-1}^2}{q_{i-1}(\lambda)}N_{i-2}\right]. \end{aligned}$$

Então

$$\begin{aligned} &N_0 = 0 \\ &N_1 = \frac{-1}{\alpha_i - \lambda} \\ &N_i = \frac{1}{q_i(\lambda)}\left[(\alpha_i - \lambda)N_{i-1} - 1 - \frac{b_{i-1}^2}{q_{i-1}(\lambda)}N_{i-2}\right] \quad i = 2, 3, \cdots, n. \end{aligned} \tag{3.21}$$

Ralha (27) propõe uma fórmula diferente para calcular N_n que tem um custo menor. Esta fórmula se baseia na seguinte função:

$$R_i = \frac{q'_i(\lambda)}{q_i(\lambda)}.$$

Observe que a relação a seguir é verificada:

$$N_i = R_i + N_{i-1} \qquad i = 1, 2, \cdots, n. \tag{3.22}$$

Da definição de $q_i(\lambda)$ temos:

$$p_i = q_i p_{i-1}$$

derivando a igualdade obtemos

$$p'_i = q'_i p_{i-1} + q_i p'_{i-1}$$

e dividindo por p_i chegamos à

$$\frac{p'_i}{p_i} = \frac{q'_i}{q_1} + \frac{p'_{i-1}}{p_{i-1}}.$$

Derivando (3.14) obtemos

$$\begin{aligned} &q'_0(\lambda) = 0 \\ &q'_1(\lambda) = -1 \\ &q'_i(\lambda) = -1 + \frac{b_{i-1}^2}{q_{i-1}^2(\lambda)}q'_{i-1}(\lambda) \quad i = 2, 3, \cdots, n \end{aligned}$$

e dividindo por $q_i(\lambda)$

$$\begin{aligned} R_0 &= 0 \\ R_1 &= \frac{-1}{q_1(\lambda)} \\ R_i &= \frac{1}{q_i(\lambda)}\left(-1 + \frac{b_{i-1}^2}{q_{i-1}(\lambda)} R_{i-1}\right) \quad i = 2, 3, \cdots, n. \end{aligned} \tag{3.23}$$

A utilização da sequência (3.23), junto com (3.22), possibilita o cálculo de N_n, usando uma sequência de primeira ordem ao invés de uma de segunda ordem como em (3.21). O uso de uma sequência de primeira ordem reduz o custo computacional, pois elimina o cálculo de um produto no método de Newton (16). O Algoritmo 3.5 realiza esse cálculo.

Algoritmo 3.5: Algoritmo zeroinNR

```
1  zeroinNR (λ, a, b, n)
2  q = a_1 − λ
3  R = −1/q
4  N = R
5  para i = 2, n faça
6  |  m = b_{i−1}^2 / q
7  |  q = (a_i − λ) − m
8  |  R = (mR − 1)/q
9  |  N = N + R
10 fim
11 λ = λ − 1/N
```

O Algoritmo 3.5 permite realizar cálculo de N_n e da função $q_n(\lambda)$. Por outro lado podemos determinar $neg_n(\lambda)$ através da contagem de ocorrências negativas que aparecem durante a execução do algoritmo. A função $neg_n(\lambda)$ possibilita que saibamos sempre em qual subintervalo resultante das iterações se encontra o autovalor procurado, além disso permite estabelecer um mecanismo de correção quando as iterações se aproximam do intervalo onde se encontra o autovalor procurado.

Nesta situação escolhemos o ponto de partida para a iteração de Newton como o ponto médio do intervalo escolhido, ou seja, iniciamos o algoritmo com a aplicação de uma etapa do método da bissecção, isto está presente no Algoritmo 3.6 (16).

Algoritmo 3.6: Algoritmo do método de Newton

```
1  programa extrai-newt
2  (x_1, x_2, y_1, y_2, δ, ε, x)
3  repita
4      prevx = x
5      zeroinNR(x, q, n)
6      x = x − 1/n
7      se (x < x_1) ou (x > x_2) então
8          (x_1 + x_2)/2
9      fim
10     se neg_n(x) > i então
11         x_2 = x
12     senão
13         x_1 = x
14     fim
15 até |x − prevx| < max(δ, |x|ε);
```

3.8.3.3 Método de Laguerre

Esse método foi sugerido em (18) como uma boa alternativa durante a fase de extração do método da bissecção.

O método iterativo de Laguerre se baseia no cálculo de uma nova aproximação das raízes do polinômio característico, utilizando:

$$L_{\pm}(x) = x + \frac{n}{\left(-\frac{p_n'(x)}{p_n(x)}\right) \pm \sqrt{(n-1)\left[(n-1)\left(-\frac{p_n'(x)}{p_n(x)}\right)^2 - n\left(\frac{p_n''(x)}{p_n(x)}\right)\right]}}. \tag{3.24}$$

A iteração (3.24) converge globalmente para raízes reais e simples, sua convergência é de ordem três próxima dos zeros. O teorema a seguir apresenta formalmente as afirmações anteriores.

Teorema 3.8 (Convergência da iteração de Laguerre) *Seja $T \in \mathbb{R}^{n\times n}$ uma matriz tridiagonal simétrica irredutível. E sejam*

$$\lambda_1 < \lambda_2 < \cdots < \lambda_n$$

que são as raízes do polinômio característico $p_n(\lambda) = \det(T - \lambda I)$. Sejam $\lambda_0 = -\infty$, então para $x \in (\lambda_i, \lambda_{i+1})$, $i = 0, 1, \cdots, n$ temos que:

1. $\lambda_i < L_-(x) < x < L_+(x) < \lambda_{i+1}$
2. *Existem duas contantes c_- e c_+ tal que*

$$|L_+(x) - \lambda_{i+1}| \leq c_+|x - \lambda_{i+1}|^3 \text{ se estiver próximo de } \lambda_{i+1}$$

$$|L_-(x) - \lambda_i| \leq c_-|x - \lambda_i|^3 \text{ se estiver próximo de } \lambda_i.$$

A demonstração pode ser encontrada em (18). Esse teorema mostra que a iteração de Laguerre aplicada tem como resultando duas sequências

$$x_+^{(k)} = L_+\left(x_+^{(k-1)}\right) = L_+^k(x) = L_+\left(L_+\left(\cdots L_+(x)\cdots\right)\right)$$
$$x_-^{(k)} = L_-\left(x_-^{(k-1)}\right) = L_-^k(x) = L_-\left(L_-\left(\cdots L_-(x)\cdots\right)\right)$$
$$\lambda_i \longleftarrow \cdots (x)_-^{(2)} < x_-^{(1)} < x < x_+^{(1)} < x_+^{(2)} < \cdots \longrightarrow \lambda_{i+1}$$

que convergem monotonicamente para duas raízes consecutivas do polinômio característico λ_i e λ_{i+1}.

Para aplicar a iteração de Laguerre da maneira que ela está definida em (3.24) é necessário calcular as funções $p_n(\lambda)$, $p_n'(\lambda)$ e $p_n''(\lambda)$, ou seja,

$$\frac{p_n'(\lambda)}{p_n(\lambda)} \quad \text{e} \quad \frac{p_n''(\lambda)}{p_n(\lambda)}.$$

Em (3.21) definimos a sequência para calcular

$$N_i = \frac{p_i'(\lambda)}{p_i(\lambda)} \quad i = 1, 2, \cdots, n.$$

A seguir apresentamos uma sequência que possibilita o cálculo de

$$S_i = \frac{p_i''(\lambda)}{p_i(\lambda)} \quad i = 1, 2, \cdots, n.$$

Ao derivar (3.20), obtemos

$$p_0''(\lambda) = 0$$
$$p_1''(\lambda) = 0$$
$$p_i''(\lambda) = (\alpha_i - \lambda)p_{i-1}''(\lambda) - 2p_{i-1}'(\lambda) - b_{i-1}^2 p_{i-2}''(\lambda) \quad i = 1, 3, \cdots, n. \tag{3.25}$$

Partindo de (3.20) e utilizando raciocínio semelhante a (3.21) obtemos

$$S_i = \frac{1}{q_i(\lambda)} = \left[(\alpha_i - \lambda)S_{i-1} - 2Ni - 1 - \frac{b_{i-1}^2}{q_{i-1(\lambda)}}S_{i-1}\right].$$

Logo

$$S_0 = 0$$
$$S_1 = 0$$
$$S_i = \frac{1}{q_i(\lambda)} = \left[(\alpha_i - \lambda)S_{i-1} - 2Ni - 1 - \frac{b_{i-1}^2}{q_{i-1(\lambda)}}S_{i-1}\right] \quad i = 2, 3, \cdots, n. \tag{3.26}$$

Fazendo uso de (3.14), (3.21) e (3.26), implementamos o algoritmo para calcular as iterações do método de Laguerre (16).

Algoritmo 3.7: Algoritmo da iteração de Laguerre

```
1  Programa zeroinLag (λ, q, N, S)
2  q_1 = a_1 − λ
3  N_0 = 0
4  N_1 = −1/q
5  S_0 = 0
6  S_1 = 0
7  para i = 2 faça
8      n
9      m = b_{i−1}^2 / q_{i−1}
10     dif = (a_1 − λ)
11     q_i = dif − m
12     N_i = (dif ∗ N_{i_1} m ∗ N_{i−2}) ∗ coc
13     S_i =
       (dif ∗ S_{i−1} − 2 ∗ N_{i−1} − m ∗ S_{i−2}) ∗ coc
14 fim
```

Levando em consideração o que foi visto e partindo de um intervalo (x_1, x_2), onde se deve isolar o autovalor de T. O algoritmo utilizado para sua aproximação é o de Laguerre combinado com o da bissecção, que é apresentado no Algoritmo 3.8.

Algoritmo 3.8: Algoritmo do método da Laguerre

```
1  programa extrai-lag (x_1, x_2, y_1, y_2, δ, ε, x)
2  x^(0)=x
3  repita
4      zeroinLang(x^(k−1), q, N, S)
5      se neg_n(x^(k−1)) < i então
6          x^(k) = L_+ x^(k−1)
7      senão
8          x^(k) = L_− x^(k−1)
9      fim
10 até |x^(k) − x^(k−1)| < max(δ, |x^(k)|ε);
```

Após tudo o que foi dito sobre as duas formas do método da bissecção modificado, isto é no isolamento e extração, o que produz uma versão algorítmica, que tem por finalidade aproximar o $i-$ésimo autovalor de uma matriz tridiagonal simétrica T_n combinando o método básico da bissecção com uma técnica de busca de raízes.

Algoritmo 3.9: Algoritmo do método da bissecção modificada

```
1  programa calcula−av(i, λ_i)
2  calcular o intervalo inicial (a, b) contendo
   λ_1 utilizando os discos de Gershgorin.
3  /* Fase de isolamento */
4  isolar (a, b, ε, δ)
5  /* Fase de extração */
6  extrair (a, b, ε, δ, λ_1)
```

Este algoritmo extrai os autovalores, utilizando os procedimentos apresentados anteriormente.

3.9 Algoritmo principal

O algoritmo desenvolvido seguirá as etapas apresentadas na Figura 10, a partir da etapa de isolamento o algoritmo é executado em CUDA.

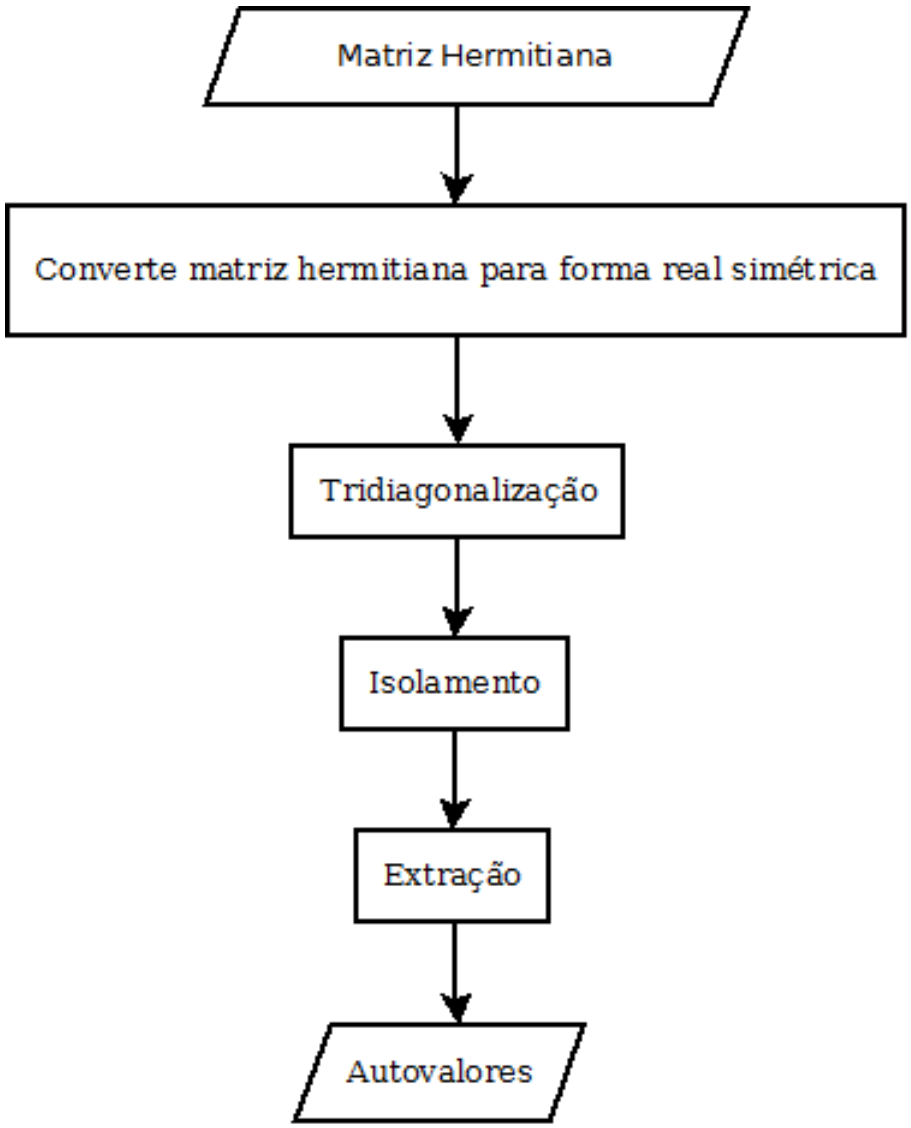


Figura 10 – Fluxograma do algoritmo desenvolvido.

4 Resultados e conclusões

A máquina utilizada na coleta dos resultados é dotada de dois processadores Intel® Xeon® CPU E5645 2,40 GHz, de 12 núcleos cada, com 96 GB de memória ram e quatro placas gráficas NVIDIA Corporation GF110GL [Tesla C2075].

Em várias linguagens de programação há diversas bibliotecas numéricas que implementam o método da bissecção para o cálculo de autovalores e tridiagonalização de matrizes reais. A LAPACK e a EISPACK (29), são duas destas bibliotecas, a primeira, possui códigos em C e FORTRAN, e a segunda, somente em FORTRAN.

Utilizaremos a rotina DSYTRD da LAPACK para tridiagonalizar as matrizes em todos os casos, no sequencial, paralelo e em CUDA. Na seção 4.3 faremos uma comparação entre a tridiagonalização da DSYTRD da LAPACK com a DSYRDB da biblioteca CULA, que roda em CUDA. A rotina DSYTRD é a mais rápida para tridiagonalização da LAPACK e a DSYRDB é a mais rápida em CUDA.

Neste trabalho, escolhemos para comparação com a nossa rotina desenvolvida em CUDA, a rotina DSTEBZ e a DSYEVR da biblioteca LAPACK. A rotina DSTEBZ realiza o cálculo de autovalores usando o método da bisseccão, a rotina DSYEVR também realiza o cálculo de autovalores, mas tem a opção de calcular os autovetores também. A rotina DSTEBZ é a mais rápida da LAPACK que utiliza somente bissecção para o cálculo dos autovalores e a DSYEVR é a mais rápida para cálculo de autovalores da LAPACK.

4.1 Resultados obtidos com execução sequencial em CPU

Na Tabela 2, temos os tempos, em segundos, gastos para o cálculo dos autovalores de uma matriz hermitiana de ordem $1024 \cdot N/2$, a matriz é passada para a forma real simétrica antes de iniciar o processo de tridiagonalização, adquirindo a seguinte ordem $1024 \cdot N$, onde N é um fator de multiplicação, ou seja, a matriz sempre será múltipla de 1024, em precisão dupla, na rotina DSTEBZ e na rotina DSYEVR. Lembrando que a matriz será passada para a forma tridiagonal utilizando a rotina DSYTRD.

A Figura 11 mostra graficamente as diferenças de tempo na execução sequencial das rotinas na CPU. Podemos observar que para matrizes de ordem acima de $N = 30$ torna-se vantajoso utilizar a rotina DSYEVR no lugar da DSTEBZ.

Tabela 2 – Tempos de execução sequencial da tridiagonalização com método da bissecção na rotina DSTEBZ e DSYEVR.

N	DSTEBZ	DSYEVR
1	0,309	0,539
5	30,621	59.469
10	243,780	513,590
15	1403,991	1471,015
20	3362,724	3585,848
25	6557,276	6489,172
30	9841,628	10048,512
35	17499,355	14734,972
40	24006,768	20383,722
45	30018,600	26519,035

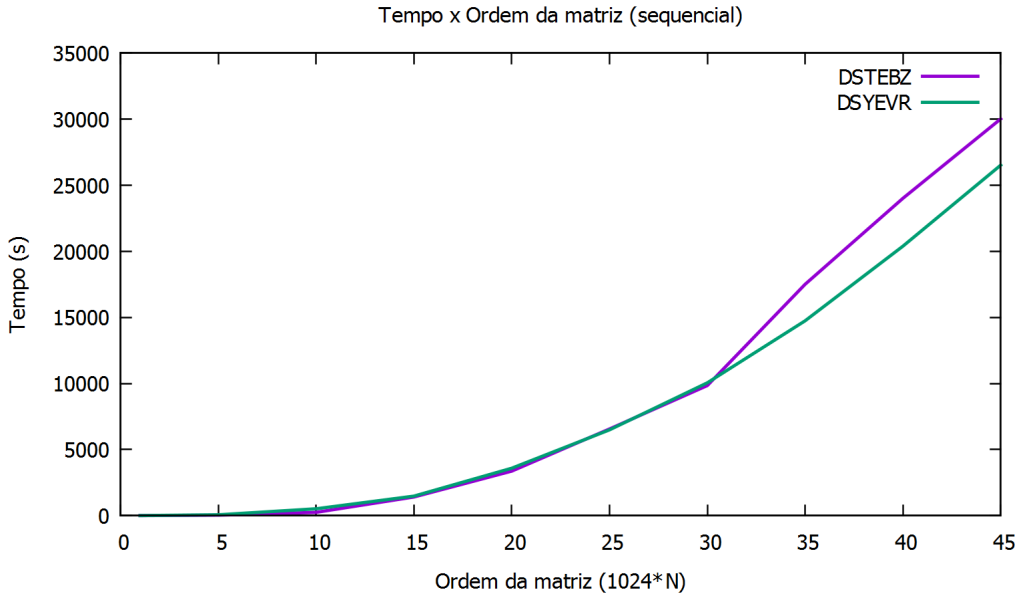


Figura 11 – Gráfico do tempo de execução sequencial versus ordem da matriz.

A Tabela 6 mostra o tempo de tridiagonalização da matriz utilizada como teste rodando de forma sequencial, nota-se que o tempo de execução das rotinas da Tabela 2 é praticamente o tempo gasto somente na tridiagonalização da matriz.

4.2 Resultados obtidos com execução paralela em CPU

Na Tabela 3, temos os tempos obtidos com execução paralela em CPU, foram utilizados os 24 núcleos disponíveis, para fazer a mesma execução da seção 4.1.

Na Figura 12 pode-se observar a diminuição do tempo de execução das rotinas e que os dois gráficos quase se sobrepõem por causa da execução paralela na CPU.

Tabela 3 – Tempos de execução em paralelo com 24 núcleos em CPU da tridiagonalização com método da bissecção em DSTEBZ e DSYEVR.

N	DSTEBZ	DSYEVR
1	0,204	0,208
5	10,897	10,516
10	76,966	76,769
15	259,15	259,333
20	601,893	606,550
25	1167,107	1219,832
30	2091,119	2094,824
35	3355,920	3365,907
40	5090,152	5207,846
45	7331,883	7322,800

Apesar disso, observa-se que o ganho não é proporcional ao número de núcleos que dividiram os processos entre si.

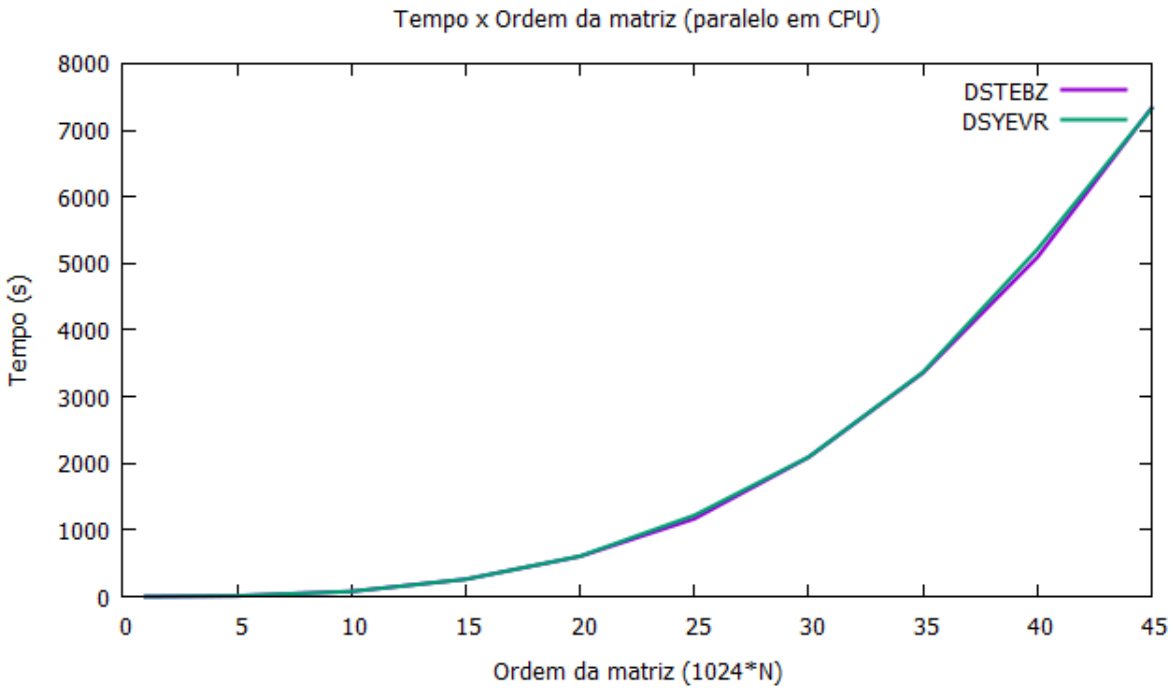


Figura 12 – Gráfico do tempo de execução em paralelo com 24 núcleos em CPU versus ordem da matriz.

Na Tabela 6 podemos ver os tempos da tridiagonalização da matriz executada de forma paralela na CPU. Caracterizando que a tridiagonalização é o gargalo do método utilizado nas rotinas para o cálculo dos autovalores.

4.3 Resultados obtidos com execução em CUDA

Na Tabela 4, temos os tempos de execução da fase de extração em CUDA dos autovalores usando o método da bissecção e o método *zeroinNR* e a aceleração obtida.

Tabela 4 – Tempos de execução da extração em CUDA.

N	Bissecção	*zeroinNR*	Aceleração
1	0,023	0,013	1,769
5	0,159	0,084	1,893
10	0,457	0,258	1,771
15	1,080	0,503	2,147
20	1,705	0,849	2,008
25	2,594	1,208	2,147
30	3,662	1,741	2,103
35	4,973	2,282	2,179
40	6,413	2,894	2,216
45	7,982	3,688	2,164
50	9,692	4,491	2,158
55	11,688	5,320	2,197
60	13,783	6,151	2,241
65	15,794	7,475	2,113
70	18,475	8,331	2,218
75	21,171	9,699	2,183
80	23,926	11,048	2,166
85	26,953	12,048	2,237
90	30,018	13,778	2,179
95	33,292	14,991	2,221
100	36,722	16,444	2,233

Com os dados da Tabela 4 e a Figura 13 podemos observar que na fase de extração o método *zeroinNR* é mais rápido do que o método da bissecção em CUDA, principalmente para matrizes de ordem alta.

A partir dos dados da Tabela 6 e da Figura 15 podemos observar que a rotina DSYTRD paralela em CPU é a mais rápida entre as três formas de tridiagonalização, mostrando que das rotinas disponíveis para tridiagonalização a da LAPACK ainda é mais rápida do que a disponível em CUDA. Observando a Tabela 5 e a Figura 14 pode-se ver que a melhor forma de resolver o problema de autovalor proposto é utilizar um método híbrido, com a DSYTRD paralela em CPU para tridiagonalização, e fazer o isolamento e extração em CUDA.

Observando os tempos da Tabela 7 comprovamos que a tridiagonalização das matrizes é o gargalo do método. O tempo de isolamento e extração é praticamente irrelevante frente ao tempo de tridiagonalização como pode ser visto na Figura 16.

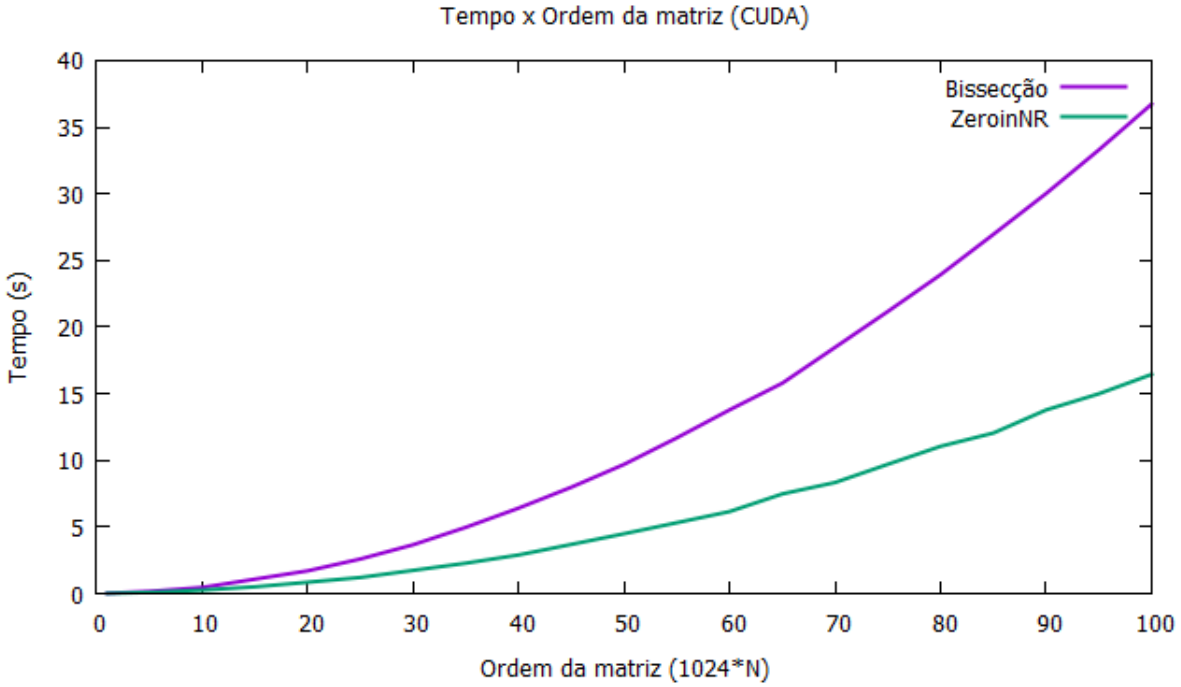


Figura 13 – Gráfico do tempo de execução da fase de extração em CUDA versus ordem da matriz.

Tabela 5 – Tempos em segundos do cálculo dos autovalores execução com 1 núcleo em CPU, execução em CUDA, execução híbrida (tridiagonalização paralela em CPU com isolamento e extração em CUDA), aceleração em relação a execução com 1 núcleo em CPU e em relação a execução em CUDA.

N	CPU-SINGLE	CUDA	Híbrido	acel.SINGLE	acel.CUDA
1	0,309	0,894	0,196	1,576	4,561
5	30,621	35,493	10,854	2,821	3,270
10	243,780	260,571	78,133	3,120	3,335
15	1403,991	869,402	259,459	5,411	3,351
20	3362,724	2217,537	549,394	6,121	4.036
25	6557,276	4104,422	1170,852	5,600	3,506

4.4 Trabalhos futuros

A partir dos resultados apresentados verificamos que o gargalo do algoritmo apresentado é a tridiagonalização, então uma possibilidade de trabalho futuro seria o desenvolvimento de uma rotina em CUDA para tridiagonalização das matrizes simétricas que seja mais eficiente do que a DSYRDB.

Outra possibilidade de trabalho futuro seria o desenvolvimento de uma rotina em CUDA usando o método de Jacobi, que também resolveria o problema da tridiagonalização.

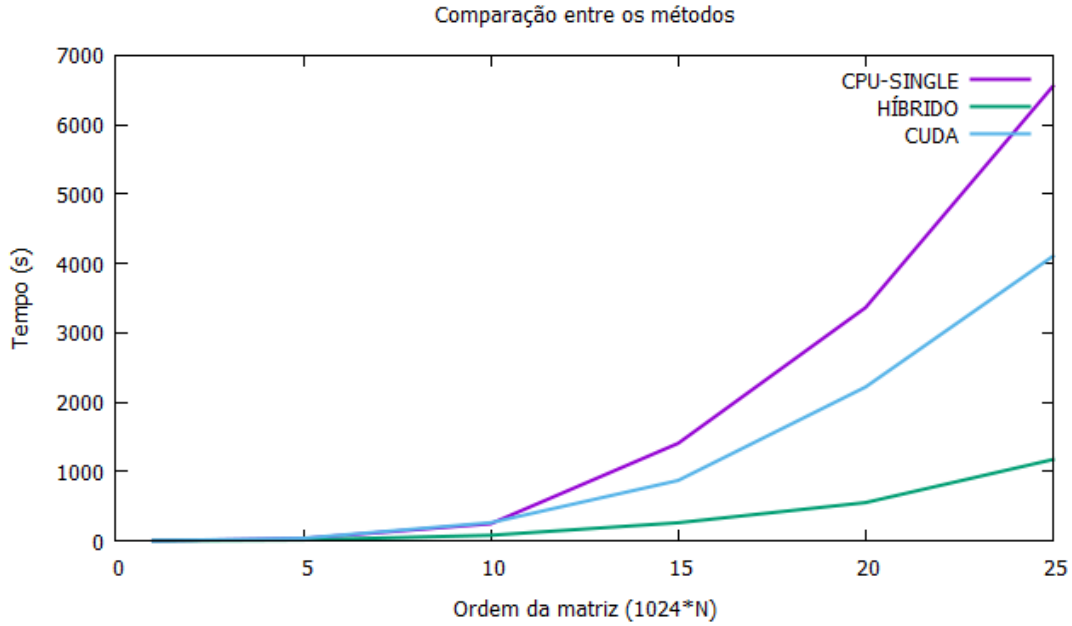


Figura 14 – Comparação entre os métodos.

Tabela 6 – Tempos de execução da tridiagonalização com 1 núcleo em CPU da DSYTRD, de execução paralela com 24 núcleos em CPU da DSYTRD, de execução em CUDA da DSYRDB.

N	DSYTRD-1	DSYTRD-24	DSYRDB
1	0,165	0,298	0,863
5	10,473	29,928	35,112
10	76,752	236,521	259,190
15	256,698	816,402	866,641
20	545,474	2089,544	2213,617
25	1164,467	5744,562	4098,037

Tabela 7 – Tempos de execução paralela com 24 núcleos em CPU da tridiagonalização com isolamento e extração usando ZeroinNR em CUDA.

N	Tridiagonalização	Isolamento	Extração
1	0,165	0,018	0,013
5	10,473	0,297	0,084
10	76,752	1,123	0,258
15	256,698	2,258	0,503
20	545,474	3,071	0,849
25	1164,467	5,177	1,208
30	2064,485	6,767	1,741
35	3328,220	9,142	2,282
40	5018,841	11,620	2,894
45	7298,747	14,044	3,688

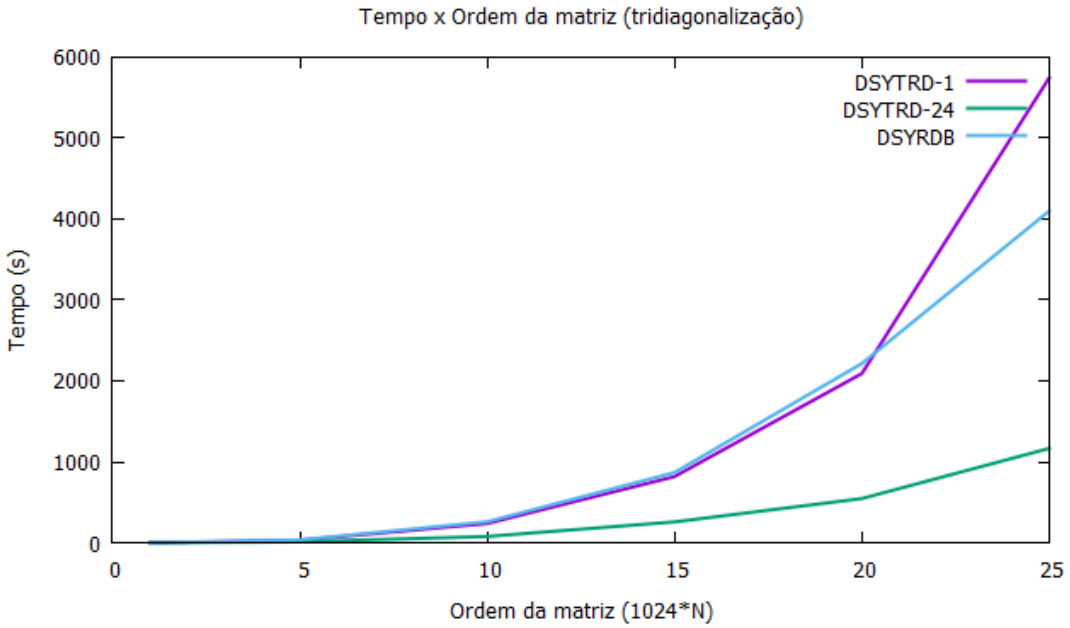


Figura 15 – Gráfico da tridiagonalização em CPU e em CUDA.

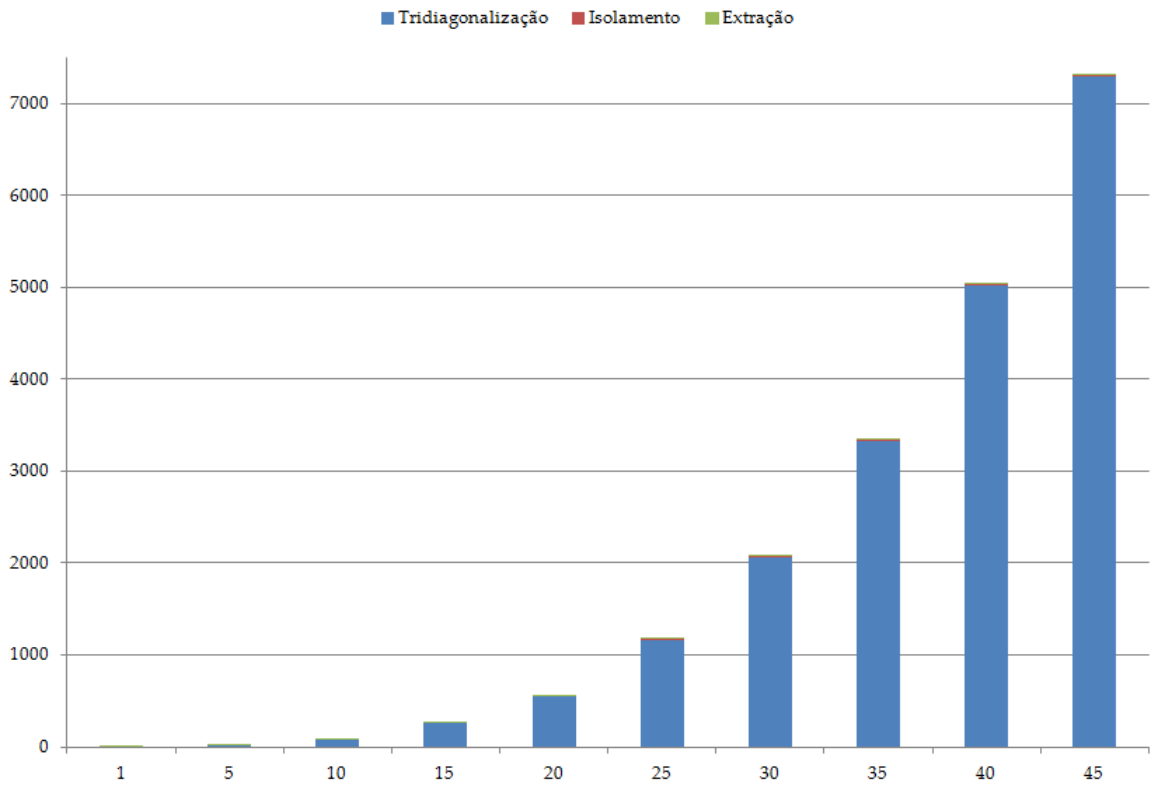


Figura 16 – Gráfico de colunas do tempo de execução do algoritmo desenvolvido.

Referências

1 KIRK, D. B.; HWU, W. W. *Programming massively parallel processors: a hands-on approach*. Burlington: Morgan Kaufmann, 2010.

2 TOP500. 2008. Top 500 list. Disponível em: <http://www.top500.org/list/2008/11/>. Acesso em: 04 jul. 2015.

3 GOLUB, G. H.; GREIF, C. An arnoldi-type agorithm for computing page rank. *BIT Numerical Mathematics*, Springer, v. 46, n. 4, p. 759–771, 2006.

4 VöMEL, C. et al. State-of-the-art eigensolvers for electronic structure calculations of large scale nano-systems. *Journal of Computational Physics*, Elsevier, v. 227, n. 15, p. 7113–7124, 2008.

5 JACOBI, C. G. J. Über ein leichtes verfahren die in der theorie der säcularstörungen vorkommenden gleichungen numerisch aufzulösen. *Journal fur die reine und angewandte Mathematik*, p. 51–94, 1846.

6 ENDERS, B. G. *Efeito de Campos Elétricos, Dopagens Não-Abruptas e Interfaces Graduais na Estrutura Eletrônica de Poços Quânticos de GaAs/AlGaAs e GaN/AlGaN*. Tese (Doutorado) — Universidade de Brasília, 2007.

7 WATKINS, D. S. *Fundamentals of Matrix Computations*. 2nd. ed. New York: John Wiley & Sons, INC., 2002.

8 GOLUB, G. G.; LOAN, C. F. V. *Matrix Computations*. 3rd. ed. Baltimore: The Johns Hopkins University Press, 1996.

9 ISAACSON, E.; KELLER, H. B. *Analysis of Numerical Methods*. New York: Dover, 1994. (Reprint of 1966 edition).

10 PULINO, P. *Algebra Linear e suas Aplicacoes: Notas de Aula*. Campinas: Universidade Estadual de Campinas, 2012.

11 LIMA, E. L. *Análise Real, vol. 1. Funções de Uma Variável*. 9. ed. Rio de Janeiro: IMPA, 2007.

12 FERNANDES, E. M. G. P. *Computação Numérica*. Braga-Portugal: Universidade do Minho, 1997.

13 STURM, J. F. C. Mémoire sur la résolution des équations numériques. p. 271–318, 1829.

14 PRESS, W. H. et al. *Numerical Recipes in C*. 2nd. ed. New York: Cambridge University Press, 1992.

15 PACKED Storage. 1999. Disponível em: <http://www.netlib.org/lapack/lug/node123.html>. Acesso em: 04 jul. 2015.

16 CONTELLES, J. M. B. *Algoritmos Paralelos para el Cálculo de los Valores Propios de Matrices Estructuradas*. Tese (Doutorado) — Universidade Politécnica de Valência, 1996.

17 DEMMEL, J. et al. *Guidelines for the Design of Symmetric Eigenroutines, SVD, and Iterative Refinement and Condition Estimation for Linear Systems*. [S.l.], 1988.

18 WILKINSON, J. H. *The Algebraic Eigenvalue Problem*. Oxford, England: Oxford University Press, 1965.

19 PARLETT, B. N. *The Symmetric Eigenvalue Problem*. New Jersey: Prentice-Hall, Inc., 1980.

20 GIVENS, W. J. *Numerical Computation of the Characteristic Values of a Real Symmetric Matrix*. Oak Ridge, 1954.

21 BARTH, W.; MARTIN, R. S.; WILKINSON, J. H. Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection. *Numerische Mathematik*, v. 9, p. 386–393, 1967.

22 BASERMANN, A.; WEIDNER, P. A parallel algorithm for determining all eigenvalues of large real symmetric tridiagonal matrices. *Parallel Computing*, Elsevier, v. 18, n. 10, p. 1129–1141, 1992.

23 LI, T. Y.; ZENG, Z. The laguerre iteration in solving the symmetric tridiagonal eigenproblem, revisited. *SIAM Journal on Scientific Computing*, Society for Industrial and Applied Mathematics, v. 15, n. 5, p. 1145–1173, 1994.

24 PEREYRA, V.; SCHERER, G. Eigenvalues of symmetric tridiagonal matrices: A fast, accurate and reliable algorithm. *J. Inst. Maths Applics*, Academic Press Inc. (London) Limited, v. 12, n. 2, p. 209–222, 1973.

25 LO, S. S.; PHILIPPE, B.; SAMEH, A. A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem. *SIAM Journal on Scientific and Statistical Computing*, Society for Industrial and Applied Mathematics, v. 8, n. 2, p. s155–s165, 1987.

26 BAKER, G. Accelerated bisection techniques for tri- and quindiagonal matrices. *International Journal for Numerical Methods in Engineering*, John Wiley and Sons, v. 35, n. 1, p. 203–218, 1992.

27 RALHA, R. Parallel solution of the symmetric tridiagonal eigenvalue problem on a transputer network. *Proceedings of the Second Congress of Numerical Methods in Engineering*, Spanish Society of Numerical Methods in Engineering, 1993.

28 TRENCH, W. F. Numerical solution of the eigenvalue problem for hermitian toeplitz matrices. *SIAM Journal on Matrix Analysis and Applications*, Society for Industrial and Applied Mathematics, v. 10, n. 2, p. 135–146, 1989.

29 NAT, L. A. *EISPACK. Eigensystem package*. Illinois: [s.n.], 1972.