Universidade de Brasília

Faculdade UnB Planaltina

Programa de Pós-Graduação em Ciência de Materiais

Lindomar José Rocha

Determinação de autovalores e autovetores de matrizes tridiagonais simétricas usando CUDA

Brasília

2015

Lindomar José Rocha

Determinação de autovalores e autovetores de matrizes tridiagonais simétricas usando CUDA

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

06 de agosto 2015

Rocha, Lindomar José
Determinação de autovalores e autovetores de matrizes tridiagonais simétricas usando CUDA/ Lindomar José Rocha. – Brasília, 06 de agosto 2015-
114 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 , 06 de agosto 2015.

1. Matriz tridiagonal simétrica. 2. Autovalor. 3. Autovetor. 4. Programação paralela. 5. CUDA. 6. Iteração inversa I. Bernhard Georg Enders Neto. II. Universidade de Brasília. III. Faculdade UnB Planaltina. IV. Determinação de autovalores e autovetores de matrizes tridiagonais simétricas usando CUDA

Lindomar José Rocha

Determinação de autovalores e autovetores de matrizes tridiagonais simétricas usando CUDA

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, 06 de agosto 2015:

Bernhard Georg Enders Neto
Orientador

David Lima Azevedo
Professor

Paulo Eduardo de Brito
Professor

Brasília
06 de agosto 2015

Dedico este trabalho a meu pai Nicolau Caetano Rocha (in memorian),
minha mãe Floriana José Rocha (in memorian),
a minha esposa Ivone Aparecida França Rocha
e as minhas filhas Ana Cristiana França Rocha
e Amábile Lúcia França Rocha.

Agradecimentos

Em primeiro lugar agradeço a Deus que me proporcionou a vida é a oportunidade de estar aqui e a toda a minha família pela tolerância e compreensão dispensada a mim nesse período de estudos.

Cabe aqui também agradecer ao professores: Bernhard Georg Enders Neto, meu orientador pela paciência, sabedoria e dedicação, sem falar que se estou aqui hoje o grande responsável foi o professor Bernhard; Paulo Eduardo de Brito por toda a ajuda dispensada e incentivo em momentos de dificuldades; David Lima Azevedo pelo companheirismo e amizade, meu muito obrigado a todos vocês.

Aos amigos Eder Carlos da Silva, Charles de Assis Oliveira Rocha, Wilson Domingos Sidnei Alves Miranda pelo companheirismo e solidariedade dispensados que muito contribuiu para a superação de diversos obstáculos.

Não poderia deixar também de agradecer ao Aristides Alvares Dourado Júnior e Jorivê Sardinha da Costa que atuam na secretaria da pós-graduação, pela assistência prestada e incentivo que me foi dispensado.

Agradeço também a CAPES pelo incentivo financeiro, pois sem o mesmo superar alguns obstáculos seria de grande dificuldade.

"Nunca, jamais desanimeis, embora venham ventos contrários"

Santa Paulina (1865–1942)

Resumo

Diversos ramos do conhecimento humano fazem uso de autovalores e autovetores, dentre eles têm-se Física, Engenharia, Economia, etc. A determinação desses autovalores e autovetores pode ser feita utilizando diversas rotinas computacionais, porém umas mais rápidas que outras nesse senário de ganho de velocidade aparece a opção de se usar a computação paralela de forma mais especifica a CUDA da Nvidia é uma opção que oferece um ganho de velocidade significativo, nesse modelo as rotinas são executadas na GPU onde se tem diversos núcleos de processamento. Dada a tamanha importância dos autovalores e autovetores o objetivo desse trabalho é determinar rotinas que possam efetuar o cálculos dos mesmos com matrizes tridiagonais simétricas reais de maneira mais rápida e segura, através de computação paralela com uso da CUDA. Objetivo esse alcançado através da combinação de alguns métodos numéricos para a obtenção dos autovalores e um alteração no método da iteração inversa utilizado na determinação dos autovetores. Temos feito uso de rotinas LAPACK para comparar com as nossas rotinas desenvolvidas em CUDA. De acordo com os resultados, a rotina desenvolvida em CUDA tem a vantagem clara de velocidade quer na precisão simples ou dupla, quando comparado com o estado da arte das rotinas de CPU a partir da biblioteca LAPACK.

Palavras-chaves: Matriz tridiagonal simétrica. Autovalor. Autovetor. Programação paralela. CUDA. Iteração inversa

Abstract

Severa branches of human knowledge make use of eigenvalues and eigenvectors, among them we have physics, engineering, economics, etc. The determination of these eigenvalues and eigenvectors can be using various computational routines, som faster than others in this speed increase scenario appears the option to use the parallel computing more specifically the Nvidia's CUDA is an option that provides a gain of significant speed, this model the routines are performed on the GPU which has several processing cores. Given the great importance of the eigenvalues and eigenvectors the objective of this study is to determine routines that can perform the same calculations with real symmetric tridiagonal matrices more quickly and safely, through parallel computing with use of CUDA. Objective that achieved by some combination of numerical methods to obtain the eigenvalues and a change in the method of inverse iteration used to determine of the eigenvectors, which was used LAPACK routines to compare with routine developed in CUDA. According to the results of the routine developed in CUDA has marked superiority with single or double precision, in the question speed regarding the routines of LAPACK.

Keywords: Symmetric tridiagonal matrix. Eigenvalue. Eigenvector. Parallel programming. CUDA. Inverse iteration

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CUDA	Compute Unified Device Architecture
MDF	Método das Diferenças Finitas
DNA	Deoxyribonucleic acid
SISD	Single Instruction Stream/Single Data Stream
C	Unidade de controle
P	Central de processamento
M	Memória
SIMD	Single Instruction Stream/Multiple Data Stream
MISD	Multiple Instruction Stream/Single Data Stream
MIMD	Multiple Instruction Stream/Multiple Data Stream
Bit	Binary digi
APIs	Application Programming Interface
OpenMP	Open Multi Processing
ARB	OpenMP Architecture Review Board
MPI	Message Passing Interface
OpenCL	Open Computing Language
CPU	Central Processing Unit
GPU	Graphics Processing Unit
GPGPU	General Purpose Graphics Processing Unit
DRAM	Dynamic Random Access Memory
GDDR	Graphics Double Data Rate
ANSI	American National Standards Institute
SDK	Software Development Kit
MCUDA	Multicore-CUDA

DIMM	Dual Inline Memory Module
LAPACK	Linear Algebra Package
SMs	Streaming Multiprocessors
SPs	Stream Processors

Lista de símbolos

β	Letra grega minúscula Beta
$\|\|$	Sinal de módulo
λ	Letra grega minúscula Lambida
$\|\|\|$	Norma de um vetor
$\langle , \rangle$	Produto interno
$\mathbb{C}$	Números complexos
$\Re$	Números Reais
$=$	Igual
$\neq$	Diferente
$\sum$	Somatório
$\cup$	União
$\varnothing$	Conjunto vazio
$\subset$	Esta contido
$\bigcap$	Intersecção
δ	Letra grega minúscula delta
ϵ	Letra grega minúscula Epsilon
γ	Letra grega minúscula Gama
$\approx$	Aproximadamente igual
∞	Infinito
$\cong$	Congruência
α	Letra grega minúscula Alfa
$\prod$	Produtório
ρ	Letra grega minúscula Rô
ϑ	Letra grega minúscula Úpsilon

Sumário

Introdução

Cálculos de autovalores podem surgir em uma rica variedade de contextos. Um pesquisador em física quântica, por exemplo, pode calcular autovalores para revelar os autoestados de energia eletrônicos de um sistema nanométrico, um engenheiro estrutural pode precisar para construir uma ponte cujas frequências naturais de vibração estão fora da banda de um terremoto típico, autovalores podem também transmitir informações sobre a estabilidade de um mercado para um economista. Um grande número de tais problemas fisicamente significativos pode ser posto como um problema matemático abstrato de encontrar todos os números λ e vetores V diferente de zero que satisfaçam a equação

$$AV = \lambda V$$

O cálculo de autovalores e autovetores, por apresentarem larga aplicação em fenômenos que envolvem matrizes e que necessitam de grande precisão e rapidez computacional um grupo de matrizes em particular que pode ofertar essas qualidades são as matrizes tridiagonais simétricas.

Ao iniciar o estudo da determinação de autovalores e autovetores de matrizes tridiagonais simétricas, antes esta aqui presente uma revisão de literatura iniciando pela definição de matriz, perpassando por algumas definições envolvendo matrizes até chegar a definição de matriz tridiagonal simétrica, bem como uma exposição relacionada a autovalores, autovetores e polinômio característico de uma matriz.

Em seguida se faz presente um estudo acerca do teorema de Sturm, teorema este que se reveste de grande importância no processo de determinação dos autovalores, pois a execução do mesmo produz o número de autovalores. O teorema dos Discos de Gershgorin diz onde se pode encontrar os autovalores, como o próprio nome indica os mesmos estão contidos em discos de raios determinados pelos teorema,onde juntamente com o teorema de Sturm apresenta relevante importância na determinação dos autovalores, pois os mesmos facilitam e diminuem o custo computacional, consequentemente essas vantagem se reflete no cálculo dos autovetores.

Tem-se também uma revisão das etapas de isolamento e extração das raízes (autovalores), esta presente um resumo dos principais métodos numéricos utilizados durante o processo de isolamento e extração das raízes, esta é a parte matemática do processo, agora chegou-se na parte computacional, onde a mesma inicia-se com as definições que envolvem o ato de determinar autovetores de matrizes tridiagonais simétricas e a aplicação dos métodos numéricos juntamente com suas modificações,

modificações estas com a finalidade de ganho de velocidade e consistência no ato de calcular os autovalores e autovetores.

Após as definições e conceitos necessárias chega-se a hora de determinar o modelo de computação e a linguagem a ser utilizada, o modelo escolhido para esse trabalho é de computação paralela juntamente com a linguagem instituída pela Nvidia a lingugem CUDA, que é uma linguagem implementada na placa gráfica da máquina, linguagem esta que oferece um ganho de velocidade nos cálculos realizados.

Agora chegou o momento de se realizar o cálculo dos autovalores e autovetores de matrizes tridiagonais simétricas, onde esse cálculos são realizados através de uma combinação de alguns métodos numéricos, tais como o da bissecção, Newton e Laguerre, isso na determinação dos autovalores quanto ao autovetores o método utilizado é o da iteração inversa, onde se faz presente uma modificação, esta se reverte no uso do método de Thomas no lugar da eliminação gaussiana o que diminui o custo computacional.

1 Fundamentação Teórica

1.1 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 nulos são ditas matrizes banda.

A largura de banda de uma matriz simétrica[1] é definida como sendo a maior distância da diagonal principal até o primeiro elemento não nulo, considerando todas as linhas da matriz (1).

Matematicamente tem-se:

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

e a banda da matriz A é

$$band(A) = \{\{i,j\}|0 < i-j \leqslant \beta(A)\} \tag{1.1}$$

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

Uma matriz tridiagonal genérica é apresentada a seguir:

$$T = \begin{bmatrix} t_{11} & t_{12} & 0 & \dots & 0 \\ t_{21} & t_{22} & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & t_{n-1\ n} \\ 0 & \dots & 0 & t_{n\ n-1} & t_{nn} \end{bmatrix} \tag{1.2}$$

No caso de matrizes tridiagonais, ou seja matrizes onde os elementos não nulos estão nas três diagonais do centro, apresenta um grande vantagem na economia de espaço de armazenamento, pois além 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 abaixo e acima da diagonal principal forem iguais.

[1] Uma matriz quadrada A é dita simétrica se ela for igual a sua transposta, ou seja $A = A^t$

1.2 Autovalores e autovetores

Problemas envolvendo autovalores e autovetores estão presentes em vários campos da Matemática (otimização não linear, sistemas de equações diferenciais), bem como em outros campos, Economia e Biologia (Cadeia de Markov), Engenharia (vibrações de sistemas dinâmicos e de estruturas), Teoria da Informação (crescimento de um rede, processamento de imagens), Física (Mecânica Quântica), estas são apenas algumas áreas que os autovalores e autovetores estão presentes.

Definição 2 *Seja V uma espaço vetorial e $T : V \to V$ um operador linear . Diz-se que um operador linear um escalar λ é um autovalor de T, se existe um vetor não nulo $v \in V$ tal que $T(v) = \lambda v$. Neste caso diz-se que v é um autovetor de T, associado ao autovalor λ*

1.2.1 Autovalores e autovetores de uma matriz

Definição 3 *Seja A uma matriz $n \times n$. Um número real λ é chamado autovalor de A , se existe um vetor não nulo $V = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ tal que*

$$AV = \lambda V \tag{1.3}$$

Um vetor não nulo que satisfaça a Eq. (1.3) *é chamado de autovetor de A.*

De acordo com a definição acima qualquer múltiplo do vetor V pode ser um autovetor , excetuando o vetor zero, pois o mesmo é autovetor de todo e qualquer autovalor.

Uma das formas para determinar os autovalores de uma matriz e subsequentes seus autovetores é através do polinômio característico.

Definição 4 *Chama-se matriz característica de $A_{n\times n}$, a matriz $\lambda I - A$, onde $A = (a_{ik})_1^n$. O determinante da matriz característica*

$$P(\lambda) = det(\lambda I - A) = 0 \tag{1.4}$$

é um polinômio escalar em λ e é chamado de polinômio característico de $A_{n\times n}$.

Para encontrar os autovalores da matriz $A_{n\times n}$ é necessário determinar as raízes do polinômio característico:

$$det(\lambda I - A) = 0$$

Ao passo que encontrar os autovetores associados a cada um dos autovalores constitui-se em determinar os vetores $V \neq 0$ que são soluções do sistema linear homogêneo

$$(\lambda I - A)V = 0$$

1.3 Norma de um vetor

A norma em um espaço vetorial tem função semelhante ao do valor absoluto, ou seja o mesmo oferece a medida de uma distância. Mais exatamente, $\Re^n$ juntamente com uma norma no $\Re^n$, defini um espaço métrico tornando as noções de vizinhança, conjunto aberto, convergência e continuidade mais familiares (2).

Definição 5 *Seja V um espaço com produto interno $\langle , \rangle$. Define-se a norma ou comprimento de um vetor v em relação a este produto interno por por $\|v\| = \sqrt{\langle v, v \rangle}$. Se $\|v\| = 1$, isto é , $\langle v \rangle = 1$, v é chamado vetor unitário. Diz-se também nesse caso que v esta normalizado.*

Observação: Todo vetor não-nulo$v \in V$ pode ser normalizado, basta fazer: $u = \dfrac{v}{\|v\|}$ (3).

Propriedade 1 *Seja V um espaço vetorial com produto interno. Para quaisquer v, w em V e $\alpha \in \Re$.*

i) $\|v\| \geqslant 0$ *e* $\|v\| = 0$ *se, e somente se* $v = 0$.

ii) $\|\alpha v\| = |\alpha|\|v\|$

iii) $|\langle v, w \rangle| \leqslant \|v\|\|w\|$ *(desigualdade de Schwarz)*

iv) $\|v + w\| \leqslant \|v\| + \|w\|$ *(desigualdade triangular)*

1.4 Ortogononalização de Gram-Schmidt

Definição 6 *Seja V um espaço vetorial com produto interno. Diz-se que uma base $\beta = \{v_1, ..., v_n\}$ de V é ortonormal se for ortogonal e cada vetor for unitário, ou seja*

$$\langle v_i, v_j \rangle = \begin{cases} 0 & se \;\; i \neq j \\ 1 & se \;\; i = j \end{cases}$$

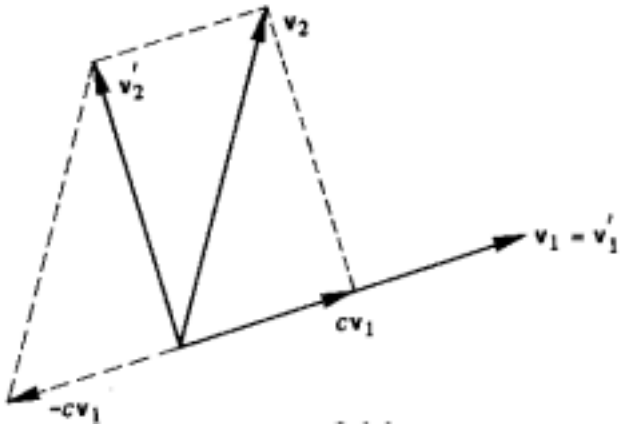
Se tiver uma base ortonormal $\{v_1, ..., v_n\}$, os coeficientes x_i de um vetor $w = x_i v_1 + ... + x_n v_n$ são dados por

$$x_i = \frac{\langle w, v_j \rangle}{\langle v_i, v_i \rangle} = \langle w, v_i \rangle$$

Existe um processo para se conseguir uma base ortonormal, para dar inicio a esse processo parte-se de um base qualquer de um espaço vetorial. A principio faz-se a ortonormalização para um base $\beta = \{v_1, v_2\}$.

Seja $v'_1 = v_1$. Necessita-se descobrir a partir de v_2 um novo vetor v'_2 que seja ortonormal a v'_1, ou seja, $\langle v'_2, v'_1 \rangle = 0$, para tanto toma-se $v'_2 = v_2 - cv'_1$, de maneira que c seja um número fixado de tal forma que $\langle v'_2, v'_1 \rangle = 0$, ou seja, $\langle v_2 - cv'_1, v'_1 \rangle = 0$. Isto indica que $c = \dfrac{\langle v_2, v'_1 \rangle}{\langle v'_1, v'_1 \rangle}$.

Figura 1 – Ortogonalização de vetores



Fonte: (3)

Então

$v'_1 = v_1$

$v'_2 = v_2 - \dfrac{\langle v_2, v'_1 \rangle}{\langle v'_1, v'_1 \rangle} v'_1$

Note que v'_2 foi conseguido de v_2, subtraindo-se do mesmo a projeção do vetor v_2 na direção de v'_1,

$$\frac{\langle v_2, v'_1 \rangle}{\langle v'_1, v'_1 \rangle} . v'_1$$

onde v'_1 e v'_2 são vetores ortogonais diferentes de zero e sua normalização tem a seguinte forma

$$u_1 = \frac{v'_1}{\|v'_1\|} \quad e \quad u_2 = \frac{v'_2}{\|v'_2\|}$$

Obtendo assim uma base $\beta' = \{u_1, u_2\}$ sendo a mesma ortonormal.

A generalização do processo de ortogonalização pode ser realizada para uma base $\{v_1, ..., v_n\}$, utilizando novamente os vetores $v'_1 = v_1$

$v'_2 = v_2 - cv'_1$ onde $c = \dfrac{\langle v_2, v'_1 \rangle}{\langle v'_1, v_1 \rangle}$

Logo, v'_1 é ortogonal a v'_2.

A ideia agora é encontrar um vetor v_3 que seja ortogonal simultaneamente a v'_1 e v'_2. Fazendo $v'_3 = v_3 - mv'_2 - kv'_1$ e calcular os valores de m e k, onde $\langle v'_3, v'_2 \rangle = 0$ e $\langle v'_3, v'_1 \rangle = 0$. Ao desenvolver essas duas situações tem-se:

$\langle v'_3, v'_1 \rangle = 0 \iff \langle v_3 - mv'_2 - kv'_1, v'_1 \rangle = 0$
$\langle v'_3, v'_1 \rangle = 0 \iff \langle v_3, v'_1 \rangle - m\langle v'_2, v'_1 \rangle - k\langle v'_1, v'_1 \rangle = 0$

Do mesmo modo, $\langle v'_2, v'_1 \rangle = 0$, tem-se $\langle v'_3, v'_2 \rangle = 0$ se, e somente se

$$k = \frac{\langle v_3, v'_1 \rangle}{\langle v'_1, v'_1 \rangle}$$

Igualmente, $\langle v'_3, v'_2 \rangle = 0$ se, e somente se

$$m = \frac{\langle v_3, v'_2 \rangle}{\langle v'_2, v'_2 \rangle}$$

Logo,

$$v'_3 = v_3 - \frac{\langle v_3, v'_2 \rangle}{\langle v'_2, v'_2 \rangle} v'_2 - \frac{\langle v_3, v'_1 \rangle}{\langle v'_1, v'_1 \rangle} v'_1$$

Onde v'_3 é conseguido de v_3 subtraindo-se suas projeções sobre v'_1 e v'_2.
Partindo de forma semelhante, obtém-se os vetores $v'_4, ..., v_n$.

Logo, partindo de uma base $\beta = \{v_1, ..., v_n\}$ de um espaço vetorial V, pode-se obter uma base ortonormal $v'_1, ..., v'_n$ dada por

$v'_1 = v_1$
$v'_2 = v_2 - \frac{\langle v_2, v'_1 \rangle}{\langle v'_1, v'_1 \rangle} v'_1$
$v'_3 = v_3 - \frac{\langle v_3, v'_2 \rangle}{\langle v'_2, v'_2 \rangle} v'_2 - \frac{\langle v_3, v'_1 \rangle}{\langle v'_1, v'_1 \rangle} v'_1$
$\vdots$
$v'_n = v_n - \frac{\langle v_n, v'_{n-1} \rangle}{\langle v'_{n-1}, v'_{n-1} \rangle} v'_{n-1} - ... - \frac{\langle v_n, v'_1 \rangle}{\langle v'_1, v'_1 \rangle} v'_1$

O procedimento mostrado acima recebe o nome de processo de ortogonalização de Gram-Schmidt.

Para se ter uma base ortonormal é suficiente normalizar os vetores v'_1, ou seja $u_i = \frac{v'_i}{\|v'_j\|}$, obtêm-se a base $\{u_1, u_2, ..., u_n\}$ de vetores ortonormais (3).

1.5 Sequência e Teorema de Sturm

A sequência de Sturm traz a possibilidade de determinar os limites sucessivos inferiores e superiores das raízes reais de um equação polinomial, ou mesmo de qualquer outra equação, bem como a facilidade de estimativa dos erros de aproximação na determinação das raízes da equação. Isto ocorre se o número de raízes reais da

equação em qualquer intervalo pode ser determinado, e isto pode de fato ser feito utilizando a sequência de Sturm (4).
Que será definida a seguir de acordo com (4).

Definição 7 *Seja a função homogênea $f_0(x) = 0$, onde $f_0(x)$ é diferenciável no intervalo fechado [a,b]. Então as funções contínuas*

$$f_0(x), f_1(x), ..., f_m(x) \tag{1.5}$$

Formam a sequência de Sturm, desde que obedeçam os seguintes critérios:

i) $f_0(x)$ *tem n raízes(onde n é o grau de* $f_0(x)$*) simples no intervalo fechado [a,b];*

ii) $f_m(x) \neq 0$ *em [a, b];*

iii) Se $f_v(\alpha) = 0$*, então* $f_{v-1}(\alpha)f_{v+1}(\alpha) < 0$ *para qualquer raiz* $\alpha \in [a, b]$*;*

iv) Se $f_0(\alpha) = 0$*, então* $f_0'(\alpha)f_1(\alpha) > 0$ *para qualquer raiz* $\alpha \in [a.b]$.

Ainda de acordo com (4) da sequência de Sturm segue o teorema abaixo:

Teorema 1 *(Teorema de Sturm) O números de zeros de $f_0(x)$ em (a,b) é igual a diferença entre o número de variações de sinal em $\{f_0(a), f_1(a), ..., f_m(a)\}$e em $\{f_0(b), f_1(b), ..., f_m(b)\}$ desde que $f_0(a)f_0(b) \neq 0$ e $\{f_0(x), f_1(x), ..., f_m(x)\}$ forme uma sequência de Sturm no intervalo [a,b].*

Prova: O número de variação de sinal pode mudar a medida que x vai de a para b somente por meio de uma função mudando o sinal no intervalo. Por *(ii)* que não pode ser $f_m(x)$. Assume-se que algum $\hat{x} \in (a, b)$, $f_v(\hat{x}) = 0$ para alguns v em $0 < v < m$. Em uma vizinha próximo a x a inscrição deve ser :

x	$f_{v-1}(x)$	$f_v(x)$	$f_{v+1}(x)$
$\hat{x}$ - ε	+	$\pm$	-
$\hat{x}$	+	0	-
$\hat{x}$ + ε	+	$\pm$	-

ou

x	$f_{v-1}(x)$	$f_v(x)$	$f_{v+1}(x)$
$\hat{x}$ - ε	-	$\pm$	+
$\hat{x}$	-	0	+
$\hat{x}$ + ε	-	$\pm$	+

Os sinais na linha $x = \hat{x}$, segue a partir de *(iii)*. Os sinais na primeira e última coluna seguem continuamente para um ϵ suficientemente pequeno.As possibilidades de sinais na coluna do meio são gerais. Mas vê-se nessa tabela que, x passa por $\hat{x}$, não há alterações no número de variações de sinais na sequência de Sturm. Examina-se agora os sinais próximo de um zero $x = \hat{x}$ de $f_0(x)$:

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

ou

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

As colunas de $f_0(x)$ representam os dois casos possíveis para um zero simples. O sinal de $f_1(\hat{x})$, então segue de *(iv)* e a continuidade implica os outros sinais nas últimas colunas. Claramente, existe agora uma diminuição na alteração no número de variação de sinal como x aumenta através de um zero de $f_0(x)$. Quando esses resultados são combinados desse modo resulta no teorema.

É fácil construir uma sequência de Sturm quando $f_0(x) \equiv P_n(x)$ é um polinômio de grau n. Define-se:

$$f_1(x) \equiv f'(x)$$

de modo que *(iv)* é satisfeita com raízes simples. Divide $f_0(x)$ por $f_1(x)$ é chame o resto $-f_2(x)$. Em seguida divide $f_1(x)$ por $f_2(x)$ e chame o resto $-f_3(x)$. Deve-se continuar o processo até terminar [o processo deve continuar até que o polinômio $f_v(x)$ tenha grau menor que seu antecessor, $f_{v-1}(x)$].
O que resulta na seguinte matriz:

$$\begin{array}{rcrcl} f_0(x) & = & q_1(x)f_1(x) & - & f_2(x), \\ f_1(x) & = & q_2(x)f_2(x) & - & f_3(x), \\ & \vdots & & & \\ f_{m-2}(x) & = & q_{m-1}(x)f_{m-1}(x) & - & f_m(x), \\ f_{m-1}(x) & = & q_m(x)f_m(x) & & \end{array} \tag{1.6}$$

Este procedimento é conhecido como o algoritmo de Euclides para determinar o maior fator comum, $f_m(x)$ de $f_0(x)$ e $f_1(x)$. Percebe-se que $f_m(x)$ é um fator de todo $f_v(x), v = 0, 1, ..., m-1$; inversamente, qualquer elemento comum de $f_0(x)$ e $f_1(x)$ deve ser um fator de $f_v(x), v = 2, 3, ..., m$. Assim as múltiplas raízes $f_0(x)$ também são raízes de $f_m(x)$ com multiplicidade reduzida a um. Se $f_m(x)$ não é uma constante, ou seja $f_0(x)$ possui várias raízes, pode-se dividir o $f_v(x)$ por $f_m(x)$ e obter a sequência (1.6) em que $f_0(x)$ tem apenas raízes simples e $f_m(x)$ é uma constante. Segue-se agora que essa sequência reduzida $\{f_0(x), f_1(x), ..., f_m(x)\}$ é uma sequência de Sturm. Somente *(iii)* exige prova: se $f_v(\hat{x}) = 0$ e por (1.6) , neste ponto $f_{v-1}(\hat{x}) = -f_{v+1}(\hat{x})$; mas se $f_{v-1}(\hat{x}) = 0$ então também $f_0(\hat{x}) = f_1(\hat{x}) = 0$ o que é uma contradição. Fórmulas simples para calcular os coeficientes de $f_v(x)$ pode ser obtida de (1.6).

A forma mais usual de se empregar a sequência de Sturm é em bissecções sucessivas, iniciando-se com intervalos escolhidos. Com esse procedimento a cada

nova avaliação da sequência o erro na determinação de uma raiz real e pelo menos metade, dessa maneira esse processo converge com fator assintótico [2]de pelo menos $\frac{1}{2}$. A importância da sequência de Sturm não esta nas propriedades de convergência rápida, mas sim em obter boas aproximações para todas as raízes reais, pois quando a raiz ou raízes forem localizadas, torna-se mais rápido a convergência ao se utilizar um método iterativo, por exemplo o método da falsa posição. Quando uma raiz é conhecida por estar contida em um intervalo *(a,b)*, onde $f(a).f(b) < 0$, calcula-se simplesmente $f\left(\frac{a+b}{2}\right)$. Agora $\frac{(a+b)}{2}$ pode ser uma raiz. Se $\frac{(a+b)}{2}$ não é uma raiz pode-se saber através do sinal $f\left(\frac{a+b}{2}\right)$, então um dos dois subintervalos $\left(a, \frac{(a+b)}{2}\right)$ ou $\left(\frac{(a+b)}{2}, b\right)$, *f(x)* tem um raiz. Pode-se repetir o processo de divisão dos intervalos diversas vezes até onde se sabe que *f(x)* tem uma raiz esse procedimento é chamado de bissecção (5) .

1.6 Teorema dos Discos de Gershgorin

A utilização do Teorema dos Discos de Gershgorin tem sua maior importância no cômputo dos autovalores de uma matriz. A aplicação mais refinada do Teorema dos discos de Gershgorin possibilita a obtenção de conhecimento mais preciso a cerca da localização dos autovetores da matriz, bem como uma melhor estimativa do raio espectral [3] da matriz considerada (4)

Definição 8 *Seja* $A = [a_{ij}] \in \mathbb{C}^{n\times n}$. *Define-se:*

$$R_i = \sum_{j=1, j\neq i}^{n} |a_{ij}| \ , \quad S_i = \sum_{j=1, j\neq i}^{n} |a_{ji}| \quad (i= i,....n)$$

Teorema 2 *(Teorema de Gershgorin) Se* $A \in \mathbb{C}^{n\times n}$ *e* a_{ij} *denota os elementos de A, i,j = 1,...,n é*

$$R_i(A) = \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{1.7}$$

[2] A notação assintótica descreve o crescimento de funções

[3] Diz-se raio espectral de uma matriz A o valor:

$$r(A) \quad = \quad \max_{i=1,...,n} |l_i|$$

Onde $l_1, ..., l_n$ são os autovetores de A

onde $\sum_{j=1,j\neq i}^{n}$ *denota a soma dos elementos da linha i de A, exceto os elementos* a_{ii}*, então todos os autovalores de A encontra-se na união dos n discos de Gershgorin no plano z complexo.*

$$G(A) = \bigcup_{i=1}^{n} \overline{D}_{Ri(A)}(a_{ii}) \tag{1.8}$$

Além disso se um conjunto de discos não tem nenhum ponto em comum com o restante n -m discos, existem exatamente m e somente m autovalores de A nessa região.

Prova: Seja λ um autovalor de A e $x = (x_1, ..., x_n) \neq 0$ um autovalor associado. Seja k um índice tal que:

$$|x_k| \geqslant |x_j| \quad para \;\; j = 1, ...n$$

ou seja x_k é a coordenada de x de maior valor absoluto. Denotando por $(Ax)_k$ a k-ésima representação do vetor $Ax = \lambda x$, tem-se

$$\lambda x_k = (Ax)_k = \sum_{j=1}^{n} a_{kj}x_j$$

que equivale a

$$x_k(\lambda - a_{kk}) = \sum_{j=1,j\neq k}^{n} a_{kj}x_j.$$

Daí

$$|x_k||\lambda - a_{kk}| = \sum_{j=1,j\neq k}^{n} |a_{kj}x_j| = \sum_{j=1,j\neq k}^{n} |a_{kj}||x_j \leqslant |x_k| \sum_{j=1,j\neq k}^{n} |a_{kj}| = |x_k|R_k(A),$$

ou seja

$$|\lambda - a_{kk}| \leqslant R_k(a)$$

Através deste resultado tem-se a prova da parte principal do Teorema de Gershgorin, como não se sabe qual o k é apropriado para cada autovalor λ e um mesmo k pode servir para mais de um autovalor λ o que se pode afirmar é que os autovalores estão na união desses discos.

Para provar a segunda parte é necessário escrever $A = D + B$, onde $D = diag(a_{11}, ..., a_{nn})$ e definir

$$A_t = D + tB$$

para $0 \leqslant t \leqslant 1$. Note que

$$R_i(A_t) = R_i(tB) = tR_i(A).$$

Para simplificar a notação assume-se que a união dos primeiros k discos de *Gershgorin*

$$G_k(A) = \bigcup_{i=1}^{k} \overline{D}_{Ri(A)}(a_{ii})$$

Satisfaz $G_k(A) \cap [G(A)/G_k(A)] = \emptyset$. Tem-se

$$\overline{D}_{Ri(At)}(a_{ii}) = \{z \in \mathbb{C} : |z - a_{ii}| \leqslant R_i(A_t)\} = \{z \in \mathbb{C} : |z - a_{ii}| \leqslant tR_i(A)\} \subset \overline{D}_{Ri(A)}(a_{ii}),$$

Logo

$$G_k(A_t) \subset G_k(A)$$

e

$$G_k(A) \cap [G(A)/G_k(A)] = \emptyset$$

onde $0 \leqslant t \leqslant 1$, porque os autovalores são funções continuas das entradas de uma matriz , o caminho

$$\lambda_i(t) = \lambda_i(A_t)$$

é um caminho continuo que liga $\lambda_i(A_0) = \lambda_i(D) = a_{ii}$ a $\lambda_i(A_1) = \lambda_i(A)$. Seja $1 \leqslant i \leqslant k$. Como $\lambda_i(A_t) \in G_k(A_t) \subset G_k(A)$, concluí-se que para $0 \leqslant t \leqslant 1$ existem k autovalores de A_t em $G_k(A)$; em particular fazendo $t = 1$, obtêm-se que $G_k(A)$ possui pelo menos k autovalores de a. Da mesma forma não pode haver mais que k autovalores de A em $G_k(A)$, pois os n-k autovalores restantes $A_0 = D$ começam fora do conjunto $G_k(A)$ e seguem caminhos contínuos que permanecem fora de $G_k(A)$ (6) (7) .

1.7 Método das diferenças finitas

O Método das Diferenças Finitas (MDF), consiste em encontra soluções para equações diferenciais, através de funções (ou uma aproximação discreta para essa função), onde essa função deve satisfazer certas relações entre suas derivadas que atendam algumas especificações em certas regiões do espaço e/ou tempo, a resolução desse tipo de problema é bastante complicado, sendo praticamente impossível sua resolução de maneira analítica.

A execução do MDF baseia-se na substituição das derivadas por diferenças de aproximações finitas. Isso pode levar a um sistema de grandes proporções, porém o mesmo é um sistema algébrico que tem possibilidade de ser resolvido computacionalmente (8).

A maneira mais simples de aproximação das derivadas de uma função é chamada de fórmula das diferenças finitas, baseada em valores discretos da própria função.

Seja *u(x)* uma função de uma variável e continua, condição essa que proporciona a derivação dessa função diversas vezes, onde cada derivada é limitada e bem definida em um intervalo que contém um ponto de interesse $\overline{x}$.

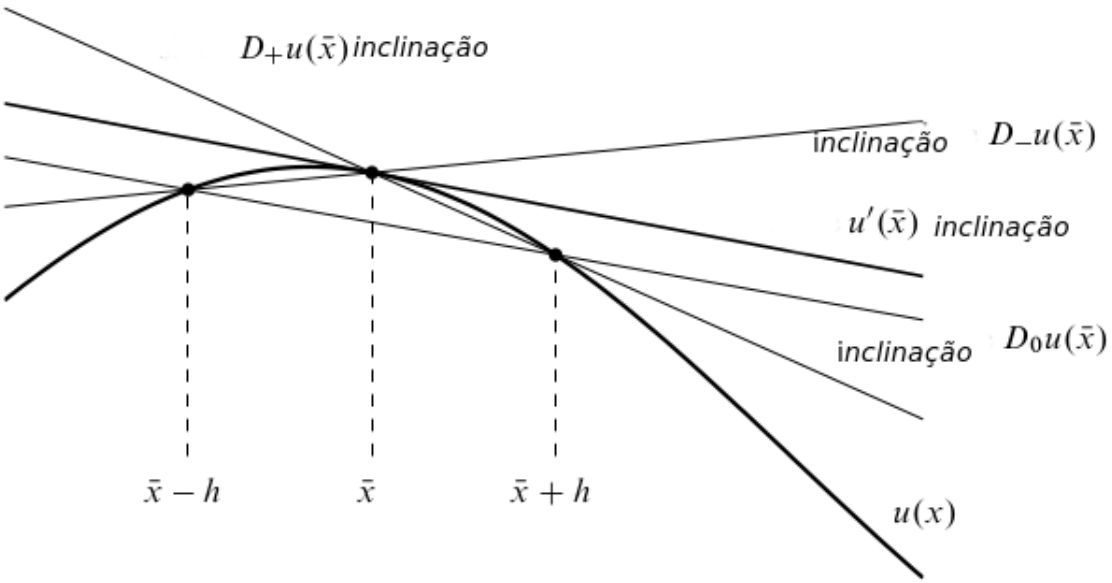
Considere a aproximação $u'(\overline{x})$ é utilizando a aproximação das diferenças finitas, baseado somente nos valores de u e num número finitos de pontos próximos a $\overline{x}$. Uma escolha evidente é utilizar:

$$D_+u(\overline{x}) \equiv \frac{u(\overline{x}+h) - u(\overline{x})}{h} \tag{1.9}$$

para h muito pequeno, o que recai na definição tradicional de derivada quando $h \longrightarrow 0$. Percebe-se que $D_+u(\overline{x})$ é a inclinação da reta de interpolação de u nos pontos $\overline{x}$ e $\overline{x}+h$. Veja Figura 2. A equação (1.9) representa um aproximação somente de um lado para u', onde u é estimado apenas para $x \geqslant \overline{x}$. A aproximação para o outo lado é dada por:

$$D_-u(\overline{x}) \equiv \frac{u(\overline{x}) - (u(\overline{x}) - h)}{h} \tag{1.10}$$

Figura 2 – Diversas aproximações $u'(\overline{x})$, interpretada como a inclinação das retas secantes



Fonte: (8)

Onde cada uma destas fórmulas representa uma aproximação de primeira ordem com precisão de $u'(\overline{x})$, sendo a dimensão do erro de aproximação proporcional a h. Tem-se também a possibilidade de utilizar a aproximação centralizada que é dada por:

$$D_o u(\overline{x}) \equiv \frac{u(\overline{x}+h) - u(\overline{x}-h)}{2h} = \frac{1}{2}(D_+ u(\overline{x}) + D_- u(\overline{x})) \tag{1.11}$$

Esta é a inclinação da reta de interpolação u em $\overline{x} - h$ e $\overline{x} + h$, o que é simplesmente a média das duas aproximações laterais vista a cima. Ao observar a Figura 2 espera-se que a aproximação $D_o u(\overline{x})$ seja um melhor aproximação do que as referentes as laterais. Esta é uma aproximação de segunda ordem e o erro é proporcional a h^2, logo é bem menor do que um erro de uma aproximação de primeira ordem onde h é muito pequeno (8).

1.7.1 Erro de truncamento

A analise do erro no MDF é feita através da expansão da função usando série de Taylor em torno de um ponto $\overline{x}$, por exemplo

$$u(\overline{x}+h) = u(\overline{x}) + hu'(\overline{x}) + \frac{1}{2}h^2 u''(\overline{x}) + \frac{1}{6}h^3 u'''(\overline{x}) + O(h^4) \tag{1.12}$$

$$u(\overline{x}-h) = u(\overline{x}) - hu'(\overline{x}) + \frac{1}{2}h^2 u''(\overline{x}) - \frac{1}{6}h^3 u'''(\overline{x}) + O(h^4) \tag{1.13}$$

As expressões acima tem validade somente se u é suficientemente suave. A notação $"big - oh"$ $O(h^p)$, é utilizada ao se abordar a convergência de métodos numéricos , a seguir uma revisão desta notação (8).

Seja $f(h)$ e $g(h)$ duas funções em h, então diz-se:

$$f(h) = Og((h)) \text{ visto que } h \longrightarrow 0$$

Caso exista uma constante C tal que

$$\left|\frac{f(h)}{g(h)}\right| < C, \text{ Para } h \text{ suficientemente pequeno}$$

ou de maneira similar utilizando limite

$$|f(h)| < C|g(h)|, \text{ para } h \text{ suficientemente pequeno}$$

Notadamente percebe-se que $f(h)$ vai para zero com a mesma velocidade de $g(h)$. Frequentemente $g(h)$ é algum monômio h^q, porém não é necessário. Utilizando (1.12), calcula-se

$$D_+ u(\overline{x}) = \frac{u(\overline{x}+h) - u(\overline{x})}{h} = u'(\overline{x}) + \frac{1}{2}hu''(\overline{x}) + \frac{1}{6}h^2 u'''(\overline{x}) + O(h^3)$$

1.7.2 Derivando aproximação por diferenças finitas

Considere a obtenção de uma aproximação por diferenças finitas para $u'(\overline{x})$, baseado em um dado conjunto de pontos. Isso pode ser realizado utilizando a série de Taylor para obter uma fórmula adequada (8).
Suponha que se queira encontrar uma aproximação unilateral para $u'(\overline{x})$ baseado em $u(\overline{x})$ $u(\overline{x}-h)$ e $u(\overline{x}-2h)$, de maneira que

$$D_2u(\overline{x}) = au(\overline{x}) + bu(\overline{x}-h) + cu(\overline{x}-2h) \tag{1.14}$$

Utiliza-se a série de Taylor para determinar os coeficientes a,b e c com intuito de se obter um precisão melhor. Substituindo (1.13) e a expansão de $u(\overline{x}-2h)$ que é :

$$(\overline{x}-2h) = u(\overline{x}) - 2hu'(\overline{x}) + \frac{1}{2}(2h)^2u''(\overline{x}) - \frac{1}{6}(2h)^3u'''(\overline{x}) + O(h)^4 \tag{1.15}$$

em (1.14), obtém-se:
$D_2u(\overline{x}) = (a+b+c)u(\overline{x}) - (b+2c)hu'(\overline{x}) + \frac{1}{2}(b-4c)h^2u''(\overline{x}) - \frac{1}{6}(b+8c)h^3u''' + ...$
Para que a haja concordância com $u'(\overline{x})$ e o mesmo seja o termo de maior ordem é necessário que:

$$\begin{aligned} a+b+c &= 0 \\ b+2c &= -\frac{1}{h} \\ b+4c &= 0 \end{aligned} \tag{1.16}$$

Resolvendo o sistema linear (1.16), obtêm-se:

$$a = \frac{3}{2h}, \quad b = -\frac{2}{h} \quad \text{e} \quad c = \frac{1}{2h}$$

De maneira que a fórmula obtida é :

$$D_2u(\overline{x}) = \frac{1}{2h}[3u(\overline{x}) - 4u(\overline{x}-h) + u(\overline{x}-2h)] \tag{1.17}$$

1.7.3 Derivada de segunda ordem

A aproximação por derivada de segunda ordem pode ser obtida de forma semelhante. A aproximação modelo para forma centralizada e dada por:

$$\begin{aligned} D_u^2(\overline{x}) &= \frac{1}{h^2}[u(\overline{x}-h) - 2u(\overline{x}) + u(u(\overline{x})+h)] \\ D_u^2(\overline{x}) &= u''(\overline{x}) + \frac{1}{12}h^2u''''(\overline{x}) + O(h^4) \end{aligned} \tag{1.18}$$

Uma vez que esta é um aproximação central, pode-se dispensar os termos ímpares. A obtenção dessa aproximação pode ser encontrada utilizando o método dos coeficientes

indeterminados ou de maneira alternativa calculando a segunda derivada do polinômio interpolado $u(x)$ em $\overline{x} - h$ e $\overline{x} + h$.

Pode-se obter uma aproximação de derivadas de ordem superior por meio de aplicação de diferenças de primeira ordem várias vezes. Como a derivada de segunda ordem origina-se de u', então $D_u^2(\overline{x})$ é um diferença da primeira diferença (8). Assim

$$D^2u(\overline{x}) = D_+D_-u(\overline{x})$$

Onde

$$\begin{aligned} D_+(D_-u(\overline{x})) &= \frac{1}{2}[D_-u(\overline{x}+h) - D_-u(\overline{x})] \\ D_+(D_-u(\overline{x})) &= \frac{1}{h}\left[\left(\frac{u(\overline{x}+h)-u(\overline{x})}{h}\right) - \left(\frac{u(\overline{x}-u(\overline{x}-h)}{h}\right)\right] \\ D_+(D_-u(\overline{x})) &= D^2u(\overline{x}) \end{aligned}$$

Outra alternativa para se obter $D^2u(\overline{x})$ é usar diferença centrada de diferença centrada, utilizando um passo igual a $\frac{h}{2}$ em cada aproximação centrada para a primeira derivada. E definir

$$\hat{D}_0u(x) = \frac{1}{h}\left(u\left(x+\frac{h}{2}\right) - u\left(x-\frac{h}{2}\right)\right)$$

Então encontra-se

$$\hat{D}_0(\hat{D}_0\overline{x}) = \frac{1}{h}\left(\left(\frac{u(\overline{x}+h)-u(\overline{x})}{h}\right) - \left(u(\overline{x} - \frac{u(\overline{x})-u(\overline{x}-h)}{h}\right)\right) = D^2u(\overline{x})$$

1.7.4 Derivadas de ordem superior

Pode-se obter a aproximação por diferenças finitas de derivada de ordem superiores utilizando os processos descritos anteriormente, por exemplo observe as duas aproximação de $u'''(\overline{x})$. A primeira é não centrada e precisão de primeira ordem (8).

$$\begin{aligned} D_+D^2u(\overline{x}) &= \frac{1}{h^3}(u(\overline{x}+h) - 3u(\overline{x}+h) + 3u(\overline{x}) - u(\overline{x}-h) \\ D_+D^2u(\overline{x}) &= u'''(\overline{x}) + \frac{1}{2}hu''''(\overline{x}) + O(h^2) \end{aligned}$$

Outra forma é utilizar aproximação central com precisão de segunda ordem

$$\begin{aligned} D_0D_+D_-u(\overline{x}) &= \frac{1}{2h^3}(u(\overline{x}+2h) - 2u(\overline{x}-h) - u(\overline{x}-2h)) \\ D_0D_+D_-u(\overline{x}) &= u'''(\overline{x}) + \frac{1}{4}h^2u'''''(\overline{x}) + O(h^4) \end{aligned}$$

1.7.5 Forma geral de derivação dos coeficientes

A forma de encontrar diferenças finitas vistas em 1.7.2 pode ser aplicada para calcular os coeficientes de aproximação por diferenças finitas para $u^{(k)}(\overline{x})$ que é a

derivada k-ésima de $u(x)$ analisada em $(\overline{x})$, baseado em um matriz qualquer onde $n \geqslant k+1$ pontos $x_1, ..., x_n$. Usualmente $\overline{x}$ é um dos pontos da matriz, porém nem sempre isso acontece.

Admitindo que $u(x)$ seja pelo menos $n+1$ vezes diferenciável continuamente em um intervalo que contém $\overline{x}$ e todos os pontos da matriz de modo que as seguintes expansões em série de Taylor tem validade. Expansão de Taylor em de u em cada ponto da matriz x_i sobre $u(\overline{x})$ produz

$$u(x_i) = u(\overline{x}) + (x_i - \overline{x})u'(\overline{x}) + ... + \frac{1}{k!}(x_i - \overline{x})^k u^{(k)}\overline{x} + ... \tag{1.19}$$

para $i = 1, ..., n$ procura-se um combinação linear desses valores que combine com $u^{(k)}\overline{x}$ da melhor forma possível.Tem-se

$$c_1 u(x_1) + c_2 u(x_2) + ... + c_n u(x_n) = u^{(k)}\overline{x} + O(h^p) \tag{1.20}$$

onde p é tão grande quando se é possível, aqui h representa a medida da largura da matriz. Se a aproximação por derivação for realizada em uma matriz com espaços de mesma medida de maneira geral o h é a largura média da malha, de forma que $\max_{1 \leqslant i \leqslant n} |x_i - \overline{x}| \leqslant Ch$, onde C é uma contante pequena. De acordo com a seção 1.7.2 pode escolher os c_j coeficientes de forma que

$$\frac{1}{(i-1)!}\sum_{j=1}^{n} c_j (c_j - \overline{x})^{(i-1)} = \begin{cases} 1, & \text{se } i-1 = k, \\ 0, & \text{para qualquer outro valor.} \end{cases} \tag{1.21}$$

Para $i = 1, ..., n$, contando que os pontos x_j são diferentes, esse é um sistema de Vandermonde [4] não singular e possui uma única solução. Se $n \leqslant k$ (poucos pontos na matriz) , logo do lado direito as duas soluções é o vetor zero, porém para $n > k$ os coeficientes possibilitam um aproximação por diferenças finitas adequada.

O vetor do lado direito em 1 tem $k+1$ linha o que dá garantia que a combinação linear tende para k-ésima derivada. Do outro lado 0 (zero), garante que

$$\left(\sum_{j=1}^{n} c_j (x_j - \overline{x})^{(i-1)}\right) u^{(i-1)}(\overline{x})$$

não esta presente na combinação linear proporcionada pela série de Taylor para $i - 1 \neq k$. Para $i - 1 < k$, isso é uma condição necessária para se obter um precisão para a aproximação por diferenças finitas mesmo que seja de primeira ordem. Para $i - 1 > k$ (só é possível se $n > k + 1$), essa condição propicia o cancelamento de termos de ordem superior na expansão e com isso tem-se uma maior precisão de primeira ordem. De maneira geral espera-se que a precisão da aproximação por diferenças finitas seja de pelo menos $p \geqslant n - k$, essa precisão pode ser ainda maior se os termos de ordem superior se cancelarem, o que ocorre com frequência com aproximações centrais (8).

[4] É uma matriz em que os termos de cada linha estão em progressão geométrica.

1.8 Isolamento e extração de raízes

Para Algumas equações existem fórmulas que facilitam a extração de suas raízes utilizando os coeficientes das mesmas, por exemplo as equações polinomiais quadráticas que podem ser resolvidas por meio da fórmula de Bhaskara.
Porém, para polinômios de grau mais alto e funções mais complexas e quase impossível determinar raízes exatas. O que se pode fazer é apenas encontrar aproximações para para essas raízes, porém isso não é uma limitação severa, pois existem métodos que conseguem determinar essas raízes, a menos da limitação da máquina e com um precisão determinada (9).

A ideia desses métodos é partir de um valor inicial para a raiz e em seguida fazer o refinamento da aproximação da raiz inicialmente considerada através de processos iterativos. Esse processo se da em duas fases (9):

i) **Fase I**: Isolar as raízes, esse processo consiste em obter um intervalo em que as raízes estejam contidas.

ii) **Fase II**: Refinar consiste na escolha de aproximações que melhorem a escolha realizada na fase I, esse processo se da sucessivamente até obter uma aproximação para a raiz dentro de uma precisão ε pré-estabelecida.

1.8.1 Isolamento

Nesse estágio faz-se uma análise teórica e gráfica da função considerada.Para que o estagio de refinamento tenha sucesso o isolamento deve ser feito com bastante precisão .
Na análise teórica usa-se o teorema de Bolzano, que será enunciado a seguir, logo após o seguinte teorema de acordo com (10).

Teorema 3 *(Conservação do sinal das funções contínuas) Seja f contínua em c admitindo que f(c)* $\neq 0$*. Existe então um intervalo* $(c-\delta, c+\delta)$ *no qual f tem o mesmo sinal de f(c)*

Prova: Supunha $f(c) > 0$. Devido a continuidade para cada $\varepsilon > 0$ existe um $\delta > 0$ tal que

$$f(c) - \varepsilon < f(x) < f(c) + \varepsilon \;\; sempre \;\; que \;\; c - \delta < x < c + \delta. \tag{1.22}$$

Tomando $\varepsilon = \frac{1}{2} f(c)$ (isto é, ε positivo), substituindo em (1.22) vem

$$\frac{1}{2} f(c) < f(x) < \frac{3}{2} f(c) \;\; sempre \;\; que \;\; c - \delta < x < c + \delta.$$

Portanto $f(x) > 0$ nesse intervalo e isso $f(x)$ e $f(c)$ têm o mesmo sinal. Se $f(c) < 0$ toma-se δ correspondente a $\varepsilon = -\frac{1}{2} f(c)$ chega-se a mesma conclusão.

Teorema 4 *(Bolzano) seja f uma função continua em cada ponto do intervalo fechado [a,b], a qual toma valores f(a) e f(b) de sinais contrários. Então existe pelo menos um c no intervalo aberto (a,b), tal que f(c) = 0.*

Prova: Suponha $f(a) < 0 \;\; e \;\; f(b) > 0$. Podem existir diferentes valores de x entre a e b para os $f(x) = 0$. O problema aqui é determinar um, isto será feito determinando o maior x para o qual $f(x) = 0$. Com essa finalidade designa-se por S o conjunto de todos os pontos x do intervalo *[a,b]* para os quais $f(x) \leqslant 0$. Existe pelo menos um desses pontos em S, porque $f(a) < 0$.Logo S não é vazio . Também S é limitado superiormente, uma vez que todos os elementos de S pertencem ao intervalo $[a, b]$ assim esse tem supremo. Seja sup S=c. Pretende-se provar que *f(c)=0*.

Existem tão somente três hipóteses: $f(c) > 0$, $f(c) < 0$ ou $f(c) = 0$. Se $f(c) > 0$ existe um intervalo $(c - \delta, c + \delta)$ ou $(c - \delta, c]$ se c=b, no qual f é positiva . Logo nenhum ponto de s pode estar a direita de $c - \delta$ e deste modo $c - \delta$ é um limite superior de S. Mas $c - \delta < c$ e c é o menor limite superior de S o que se pode concluir que é impossível a desigualdade $f(c) > 0$. Se $f(c) < 0$ existe um intervalo $(c - \delta, c + \delta)$ ou $[c, c + \delta)$, se $c = a$ no qual f é negativa . Portanto $f(x) < 0$ para algum $x > c$, o que contradiz o fato de que c é supremo de S, logo $f(c) < 0$ é também impossível e a única possibilidade é a hipótese restante $f(c) = 0$. Além disso $a < c < b$. porque $f(a) < 0$ e $f(b) > 0$

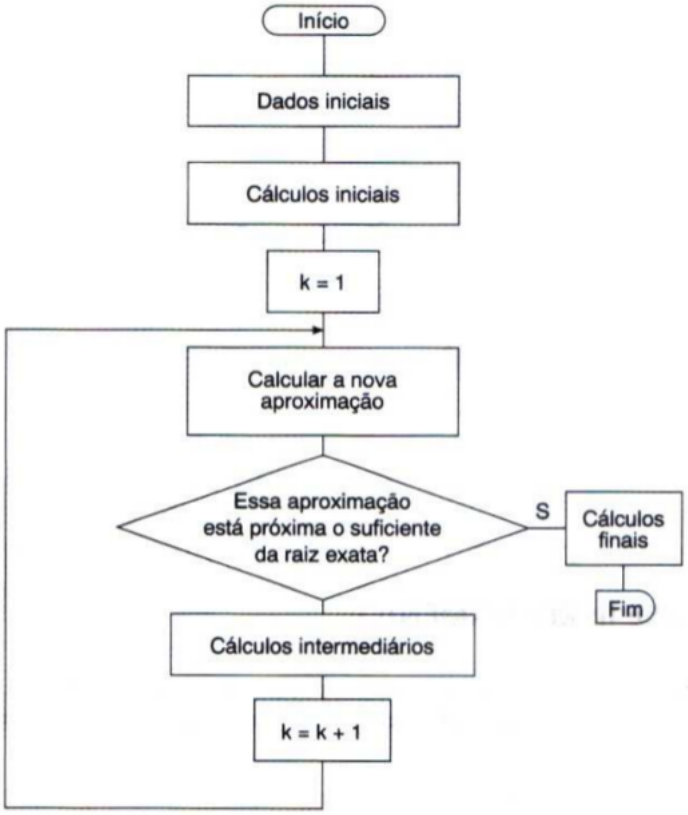
1.8.2 Refinamento

O refinamento das raízes é realizado através de métodos numéricos, existem diversos métodos numéricos, o que os diferencia é a maneira que o refinamento é realizado. Todos eles são métodos iterativos.
Métodos iterativos caracterizam-se por executarem sequencialmente as instruções passo a passo, sendo que algumas dessas execuções podem ocorrer de forma cíclica.

Durante a execução ao se completar um ciclo tem-se ai uma iteração. As iterações fazem usos dos resultados das iterações anteriores é também testes que possibilitam verificar se a precisão exigida foi atingida.

Métodos iterativos para a obtenção de raízes fornecem aproximações para a solução exata, de forma geral esses métodos tem as etapas que são apresentadas no diagrama de fluxo a seguir (10).

Figura 3 – Fluxograma de Métodos Iterativos



Fonte: (11)

1.8.3 Critério de parada

O Critério de parada é o que interrompe a continuidade dos passos iniciados pelos métodos numéricos, onde o mesmo deve avaliar se o x_k na k-ésima iteração esta suficientemente próximo da raiz exata. Porém nem sempre o valor exato da raiz é conhecido, logo o processo será interrompido de acordo com uma das condições a seguintes:

i) Avaliação do ponto na função

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

ii) Avaliação do tamanho do intervalo

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

onde ε denota a precisão desejada. Métodos numéricos são concebidos de maneira a satisfazer uma das condições enumerada acima.
Normalmente em programas de computadores deve-se estipular o número máximo de iterações, pois se isso não for feito o programa pode entrar em *looping* (10).

1.8.4 Métodos numéricos para obtenção de zeros de funções

1.8.4.1 Bissecção

Antes de enunciar o método da bissecção precisa-se enunciar o Teorema do valor intermediário, o qual serve da base para o método da bissecção.

Teorema 5 *(do valor intermediário) Se f for contínua em um intervalo [a,b] e se γ for um número real compreendido entre f(a) e f(b) então existirá pelo menos um c em [a,b],tal que* $f(c) = \gamma$

Prova. Suponha $f(a) < \gamma < f(b)$ considere a função

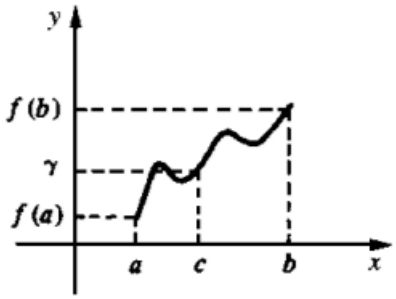
$$g(x) = f(x) - \gamma, \; x \text{ em } [a,b].$$

Como f é contínua em $[a,b]$ g também o é; tem-se ainda

$$g(a) = f(a) - \gamma < 0 \text{ e } g(b) = f(b) - \gamma > 0$$

Pelo teorema de Bolzano existe c em $[a,b]$ tal que $g(c) = 0$, ou seja $f(c) = \gamma$

Figura 4 – Teorema do Valor Intemediário



Fonte: (12)

O método da Bissecção que é um método de encaixe consiste na divisão do intervalo inicial $[x_i, x_s]$ (x_i limite inferior e x_s limite superior) onde o mesmo contém uma única raiz, para garantir a convergência.
O requisito a seguir

$$f(x_i)f(x_s) < 0$$

diz que dentro do intervalo considerado e se a função for contínua conterá um número ímpar de raízes. A aproximação da raiz é feita através do cálculo do ponto médio deste intervalo.

$$x_k = \frac{x_i + x_s}{2}, \;\; k = 1, 2, ... \tag{1.23}$$

Após o cálculo do ponto médio tem-se dois novos intervalos que são: $[x_i, x_k]$ e $[x_k, x_s]$ o que contiver a raiz e que deve ser escolhido para a iteração próxima é,

$$[x_i, x_k] \;\; se \;\; f(x_i)f(x_k) < 0 \text{ e faz-se } x_s \longleftarrow x_k$$

ou

$$[x_k, x_s] \;\; se \;\; f(x_k)f(x_s) < 0 \text{ e faz-se } x_i \longleftarrow x_k$$

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 estabelecido seja satisfeito, garantido assim uma aproximação com uma certa precisão.

Se o intervalo da iteração $k + 1$ for $[x_i, x_s]$, então o intervalo da interação $k + 2$ será $[x_i, x_{k+1}]$ ou $[x_{k+1}, x_s]$, onde x_{k+1} é definido por (1.23). Donde se tira que

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

ou

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

Onde na iteração anterior $x_s = x_k$ (ou $x_i = x_k$) e x_i (ou x_s) $\longrightarrow x^*$. 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 tem uma convergência muita lenta é não depende de *f(x)*, apenas o sinal de f é considerado (13).
Observação:

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

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

i) Se $r = 1$ e $C < 1$, a taxa de convergência é **linear**

ii) Se $r > 1$ a taxa de convergência é **superlinear**

iii) Se $r = 2$ a taxa de convergência é **quadrática**

O que distingue a convergência linear da superlinear é que assintoticamente uma sequência que converge linearmente ganha um número constante de dígitos aprimorados por iteração ao passo que uma sequência que converge superlinearmente ganha sempre um numero maior de dígitos aprimorados em cada iteração. Um método que converge quadraticamente dobra o número de dígitos apurados em cada iteração.

Algoritmo 1.1: Algoritmo sequencial da Bissecção

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

```
1  enquanto (x_i − x_s)/2 > tol faça
2      x_k ← (x_i + x_s)/2;
3      se f(x_k) = 0 então
4          pausa
5      fim
6      se f(x_i)f(x_s) < 0 então
7          x_s ← x_k
8      senão
9          x_i ← x_k
10     fim
11 fim
12 retorna (x_i + x_s)/2
```

1.8.4.2 Newton-Raphson

O método de Newton-Raphson poder ser usado para calcular raízes reais ou complexa de $f(x) = 0$. É um dos mais rápidos métodos de cálculo de raiz e quanto mais perto estiver da raiz mais rápido será a convergência (13).
Para desenvolver a expressão que determina o método de Newton faz-se uso da expansão de uma função em série de Taylor em torno do ponto x_k. Ela é escrita como:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + f''(x_k)\frac{(x - x_k)^2}{2!} + f'''(x_k)\frac{(x - x_k)^3}{3!} + ..., \quad (1.24)$$

onde x_k é um valor aproximado da raiz de $f(x)$ na iteração k do processo iterativo, onde f' é 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 (1.24) reduz-se a:

$$0 = f(x_k) + f'(x_k)(x_{k+1} - x_k) + f''(x_k)\frac{(x_{k+1} - x_k)^2}{2!} + f'''(x_k)\frac{(x_{k+1} - x_k)^3}{3!} + ..., \quad (1.25)$$

Utilizando os dois primeiros termos da expansão da série de Taylor do segundo membro da equação (3.4) obtém-se a expressão para o método Newton-Raphson:

$$f(x_k) + f'(x_k)(x_{k+1} - x_k) = 0$$

$$f'(x_k)(x_{k+1} - x_k) = -f(x_k)$$
$$(x_{k+1} - x_k) = -\frac{f(x_k)}{f'(x_k)}$$
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)},\ k = 1, 2, ... \tag{1.26}$$

Que é a equação iterativa do método Newton-Raphson. Sendo x_1 a aproximação inicial do processo a equação (1.26) resulta da aproximação de uma reta tangente a $f(x)$ no ponto da iteração vigente, x_k e calcular a interssecção dessa tangente com o eixo da abcissas (raiz da equação) . A desvantagem desse método reside no fato de calcular analiticamente a primeira derivada de $f(x)$.

Agora passamos a verificar a ordem de convergência do método de Newton-Raphson.

Proposição 1 *Seja $f(x)$ uma função com segunda derivada continua 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.*

Considerando a expansão em série de Taylor da função $f(x)$ em relação a x_k,

$$f(x^*) = f(x_k) + (x^* - x_k)f'(x_k) + \frac{1}{2}(x^* - x_k)^2 f''(\xi),\ \ \xi \in [x_k, x^*]$$

e dividindo-se por $f'(x_k)$ e como $f(x^*) = 0$ tem-se

$$\frac{f(x_k)}{f'(x_k)} + (x^* - x_k) + \frac{1}{2}(x^* - x_k)^2 \frac{f''(\xi)}{f'(x_k)} = 0$$
$$\frac{f(x_k)}{f'(x_k)} + (x^* - x_k) = -\frac{1}{2}(x^* - x_k)^2 \frac{f''(\xi)}{f'(x_k)}$$
$$\frac{f(x_k)}{f'(x_k)} + x^* - x_k = -\frac{1}{2}(x^* - x_k)^2 \frac{f''(\xi)}{f'(x_k)}$$
$$x^* - \left[x_k - \frac{f(x_k)}{f'(x_k)}\right] = -\frac{1}{2}(x^* - x_k)^2 \frac{f''(\xi)}{f'(x_k)}$$

De acordo com a equação (1.26) tem-se

$$x^* - x_{k+1} = -\frac{1}{2}(x^* - x_k)^2 \frac{f''(\xi)}{f'(x_k)} \tag{1.27}$$

Se $\varepsilon_k = x^* - x_k$ for o erro da iteração k (1.27) transforma-se em:

$$\varepsilon_{k+1} = -\frac{1}{2}\frac{f''(\xi)}{f'(x_k)}\varepsilon_k^2 \tag{1.28}$$

e quando $x_k \longleftarrow x^*$ também $\xi \approx x^*$ substituindo em (1.28) e transpondo ε_k^2 para o primeiro membro tem-se:

$$\frac{\varepsilon_{k+1}}{\varepsilon_k^2} = -\frac{1}{2}\frac{f''(x^*)}{f'(x^*)}$$

Calculando o limite abaixo

$$\lim_{k\longleftarrow\infty} \frac{|\varepsilon_{k+1}|}{\varepsilon_k^2} = \frac{1}{2}\frac{f''(x^*)}{f'(x^*)}$$

O que mostra que o método Newton-Raphson tem convergência quadrática.

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 o método pode ficar muito lento ou até mesmo não convergir para a solução esperada. Os dois teoremas abaixo fornece as condições de convergência do referido método (13).

Teorema 6 *Seja x_1 a aproximação inicial do método de Newton e seja $\{x_k\}$ a sequência gerada por* (1.26)*. Define-se o intervalo*

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

Suponha que

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

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

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

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

Teorema 7 *Suponha que $f'(x) \neq 0$, que $f''(x)$ não muda de sinal no intervalo [a,b] e que $f(a)f(b) < 0$*

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, $x_1 \in [a, b]$.

Para utilizar o método de Newton- Raphson no cálculo de raízes de funções complexas a aproximação inicial deve ser um valor complexo e os cálculos serão realizados em aritmética complexa.

Ao realizar cálculos com funções onde as raízes não são simples há a necessidade de se fazer uma alteração na equação iterativa (1.26). Definição da nova função

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

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

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

ou

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

Algoritmo 1.2: Algoritmo sequencial Newton-Raphson

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

Saída: a raiz desejada de f

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

1.8.4.3 Secante

O método da secante realiza o cálculo da raiz da função $f(x) = 0$ e baseia-se na aproximação de $f(x)$ por uma reta nas proximidades da raiz. O ponto de interssecção da reta com o eixo das abcissas é tido como a aproximação da raiz de $f(x) = 0$. Se a aproximação não atingir a precisão desejada da raiz x^* o processo deve ser repetido iterativamente até que a mesma seja atingida. Um inconveniente existente no método de Newton-Raphson é 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 (13). Do método de Newton-Raphson tem-se:

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

Para a aproximação da derivada tem-se:

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

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:

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

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})} \tag{1.29}$$

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

Dessa maneira define-se a função de iteração para o método da secante:

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})},\ k = 2, 3, ... \tag{1.30}$$

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

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

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

$$x_{k+1} = x_k - \frac{-(x_{k-1} - x_k) f(x_k)}{-[f(x_{k-1}) - f(x_k)]}$$

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

$$x_{k+1} = x_k - \frac{(x_{k-1} - x_k)\frac{f(x_k)}{f(x_{k-1})}}{\left[1 - \frac{f(x_k)}{f(x_{k-1})}\right]}, \tag{1.31}$$

Utiliza-se (1.31) ao invés de (1.30) quando $|f(x_{k-1})|$ for maior que $|f(x_k)|$; senão troca-se os valores de x_k e x_{k-1} assim como os valores da função, antes de usar a expressão (1.31).

A seguir ver-se a como se dá a convergência do método da secante;

Proposição 2 *Seja $f(x)$ uma função com segunda derivada continua 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.*

Utilizando a fórmula interpoladora de Newton baseada em diferenças divididas tem-se:

$$f(x) = f(x_k) + (x - x_k)[x_{k-1}, x_k] + \frac{1}{2}(x - x_{k-1})(x - x_k)f''(\xi) \tag{1.32}$$

onde $[x_{k-1}, x_k]$ é a diferença dividida de primeira ordem definida por:

$$[x_{k-1}, x_k] = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

e ξ pertence ao intervalo definido por x, x_{k-1} e x_k . Substituindo x por x^* em (1.32)

$$f(x^*) = f(x_k) + (x^* - x_k)[x_{k-1}, x_k] + \frac{1}{2}(x^* - x_{k-1})(x^* - x_k)f''(\xi) \tag{1.33}$$

A equação(1.29) agora escrita como

$$x_{k+1}f(x_k) - f(x_{k-1}) = x_k f(x_k) - f(x_{k-1}) - (x_k - x_{k-1})f(x_k)$$

dividindo os dois lados por $x_k - x_{k-1}$ tem-se

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

$$x_{k+1}[x_{k-1}, x_k] = x_k[x_{k-1}, x_k] - f(x_k), \tag{1.34}$$

Somando (1.32) com (1.33) obtém-se:

$$(x^* - x_{k+1})[x_{k-1}, x_k] = -\frac{1}{2}(x^* - x_{k-1})(x^* - x_k)f''(\xi)$$

De acordo com o teorema 5, tem-se:

$$[x_{k-1}, x_k] = f'(\eta) \;\; \eta \in [x_{k-1}, x_k]$$

e considerando o erro da iteração k como sendo $\varepsilon_k = x^* - x_k$ tem-se a relação a seguir

$$\varepsilon_{k+1} = -\frac{f''(\xi)}{2f'(\eta)}\varepsilon_{k-1}\varepsilon_k. \tag{1.35}$$

Quando $k \longrightarrow \infty$ tem-se $\xi \approx x^*$ e $\eta \approx x^*$ e (1.35) toma a seguinte forma

$$|\varepsilon_{k+1}| \approx K|\varepsilon_{k-1}||\varepsilon_k| \tag{1.36}$$

onde $K \neq 0$ é uma constante. Considerando $|\varepsilon_{k+1}| \approx C|\varepsilon_k|^p$, $|\varepsilon_k| \approx C|\varepsilon_{k-1}|^p$ e substituindo em (1.36) chega-se em

$$C|\varepsilon_k|^p = K\left(\frac{|\varepsilon_k|}{C}\right)^{\frac{1}{p}}|\varepsilon_k|$$

$$C|\varepsilon_k|^p = KC^{-\frac{1}{p}}|\varepsilon_k|^{\frac{1}{p}}|\varepsilon_k|.$$

de

$$C = KC^{-\frac{1}{p}} \Rightarrow K = \frac{C}{(C)^{-\frac{1}{p}}} \Rightarrow K = C^{1+\frac{1}{p}}$$

e

$$|\varepsilon_k|^p = |\varepsilon_k|^{\frac{1}{p}}|\varepsilon_k| \Rightarrow |\varepsilon_k|^p = |\varepsilon_k|^{\frac{1}{p}+1}$$

logo p = $\frac{1}{p} + 1$ e $K = C^p$
A raiz positiva da equação acima é 1.618 de onde se conclui que a ordem de convergência do método da secante é 1.618 uma vez que

$$|\varepsilon_{k+1}| \approx (K)^{\frac{1}{p}}|\varepsilon_k|^p$$

ou

$$\lim_{k \longleftarrow \infty} \frac{|\varepsilon_{k+1}|}{|\varepsilon_k|^p} = K^{\frac{1}{p}}$$

com p= 1.618
Esta convergência é dita convergência superlinear.

O método da secante pode ser utilizado para calcular raízes complexas, para que isso ocorra basta introduzir a aritmética complexa nos cálculos e as aproximações iniciais sejam números complexos.

Ao proceder o cálculo de uma raiz múltipla o método da secante converge linearmente. Porém é necessário acelerar o processo iterativo introduzindo algumas modificações. Definindo uma nova função

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

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

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})u(x)}{u(x_k) - u(x_{k-1})} \tag{1.37}$$

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

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})f(x_k)f'(x_{k-1})}{f(x_k)f'(x_{k-1}) - f(x_{k-1})f'(x_k)},\ k = 2, 3, ... \tag{1.38}$$

Algoritmo 1.3: 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 ⟵ 0
2  enquanto i ⩽ N faça
3      x_{k+1} = (x_{k-1} f(x_k) − x_k f(x_{k-1})) / (f(x_k) − f(x_{k-1}));
4      se abs((x_{k+1} − x_k) / x_{k+1}) < ε então
5          Apresentar x*
6          Finalizar o programa
7      fim
8      x_{k-1} ⟵ x_k
9      x_k ⟵ x_{k+1}
10     i ⟵ i + 1
11     Exibir a mensagem:"Método falhou em N iterações"
12 fim
```

1.8.4.4 Falsa Posição

O método da falsa posição assim como o da bissecção é um método de encaixe. A escolha das duas aproximações iniciais devem levar em consideração o seguinte:

$$f(x_i)f(x_s) < 0$$

O intervalo definido pelas aproximações $[x_i, x_s]$deve conter apenas a raiz procurada. A equação a seguir gera a sequencia $\{x_k\}$ de aproximação a raiz é descrita abaixo, que a equação da reta $r(x)$ que passa pelos pontos $(x_i, f(x_i))$ e $(x_s, f(x_s))$,

$$\frac{x - x_i}{r(x) - f(x_i)} = \frac{x_s - x_i}{f(x_s) - f(x_i)},$$

como $r(x) = 0$

$$\frac{x - x_i}{-f(x_i)} = \frac{x_s - x_i}{f(x_s) - f(x_i)},$$

ou

$$x_k = x_i - f(x_i)\frac{x_s - x_i}{f(x_s) - f(x_i)}, k = 1, 2, ... \tag{1.39}$$

que corresponde a aproximação da raiz de $f(x)$ por uma reta que passa pelos pontos $(x_i, f(x_i))$ e $(x_s, f(x_s))$ e calcular a raiz da equação, ou seja o ponto de intersecção da reta com o eixo da abscissas.

Como a próxima iteração será realizada no intervalo que contém a raiz procurada, dos subintervalos $[x_i, x_k]$ e $[x_k, x_s]$ escolhe-se:

$$[x_i, x_k] \; se \; f(x_i)f(x_k) < 0 \text{ e faz-se } x_s \longleftarrow x_k$$

ou

$$[x_k, x_s] \; se \; f(x_k)f(x_s) < 0 \text{ e faz-se } x_i \longleftarrow x_k$$

Assim como o método da bissecção o método da falsa posição sempre converge, exige-se que a função seja continua. A ordem de convergência é a mesma da secante, ou seja 1.618, porém em certas situações a ordem de convergência pode ser linear.Quando a função é convexa no intervalo inicial $[x_i, x_s]$, o ponto inicial x_i permanece durante as iterações e se

$$\varepsilon_k = x^* - x_k$$

for o erro da aproximação x_k alcançado na k-ésima iteração da relação (1.36)

$$|\varepsilon_{k+1}| \approx K|\varepsilon_k||\varepsilon_{k-1}| \text{ tem-se com } |\varepsilon_{k-1}| \approx |\varepsilon_i|,$$

$$|\varepsilon_{k+1}| \approx K|\varepsilon_k||\varepsilon_i| = K_1|\varepsilon_i|$$

Com $K \neq 0$ e $K_1 = K|\varepsilon_i|$, constantes e

$$\lim_{k \longleftarrow \infty} \frac{|\varepsilon_{k+1}|}{|\varepsilon_k|} = K_1$$

o que diz que a ordem de convergência é igual a um. O método é robusto no começo da iterações principalmente se estiver longe da solução, porém torna-se muito lento ao se aproximar da solução (13).

Algoritmo 1.4: Algoritmo sequencial da Falsa Posição

Entrada: x_i e x_s, f e ε
Saída: a raiz desejada de f

se $f(x_i)f(x_s) < 0$ **então**
  $x_k = x_i - f(x_i)\dfrac{x_s - x_i}{f(x_s) - f(x_i)}$
  enquanto $|f(x)| > \varepsilon$ *e* $|x_s - x_i| > \varepsilon$ **faça**
    $x_{k+1} = \dfrac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$;
    se $f(x_i)f(x_s) < 0$ **então**
      $x_i \longleftarrow x_k$
      senão
        $x_s \longleftarrow x_k$
        $x_k = x_i - f(x_i)\dfrac{x_s - x_i}{f(x_s) - f(x_i)}$
      fim
    fim
  fim
  senão
    Escreva ("Não existem raízes no intervalo dado")
  fim
fim

1.8.4.5 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 escreve-se o polinômio na forma fatorada

$$P_n(x) = (x - x_1^*)(x - x_2^*)...(x - x_n^*) \tag{1.40}$$

A equação (1.40) pode ser simplificada para melhor clareza, aplicando logaritmo na mesma , obtendo assim

$$\begin{aligned} ln|P_n(x)| &= ln|x - x_1^*| + ln|x - x_2^*| + ... + ln|x - x_n^*| \\ ln|P_n(x)| &= \sum_{k=1}^{n} ln|x - x_k^*| \end{aligned} \tag{1.41}$$

Derivando (1.41), tem-se

$$G = \frac{d\ ln|P_n(x)|}{dx} = \frac{1}{x - x_1^*} + \frac{1}{x - x_2^*} + ... + \frac{1}{x - x_n^*} = \frac{P_n'(x)}{P_n(x)} \tag{1.42}$$

Calculando a segunda derivada de (1.41) obtêm-se

$$\begin{aligned} H = -\frac{d\ ln|P_n(x)|}{dx} &= \frac{1}{(x-x_1^*)^2} + \frac{1}{(x-x_2^*)^2} + ... + \frac{1}{(x-x_n^*)} \\ H = -\frac{d\ ln|P_n(x)|}{dx} &= \left[\frac{P_n'(x)}{P_n(x)}\right]^2 - \frac{P_n''(x)}{P_n(x)} \end{aligned} \tag{1.43}$$

Na obtenção da estimativa das raízes deve-se fazer as seguintes proposições:

i) A distância entre x_1^* e o valor inicial x_0 será representado por a

ii) A distância entre o valor inicial x_0 e e as outras raízes será dado por b

$$\begin{aligned} x_0 - x_1 &= a \\ x_0 - x_k &= b, \qquad k = 2, 3, ..., n \end{aligned} \tag{1.44}$$

Escrevendo (1.42) e (1.43) em função de a e b

$$\frac{1}{a} + \frac{n-1}{b} = G \tag{1.45}$$

$$\frac{1}{a^2} + \frac{n-1}{b^2} = H \tag{1.46}$$

Isolando b em (1.45) e substituído em (1.46) obtêm-se

$$a = \frac{n}{G + \sqrt{(n-1)(nH - G^2)}} \tag{1.47}$$

Tomando (3.21) como $G = \dfrac{\frac{d}{dx}p(x)}{p(x)}$ em H , onde $H = G^2 - \dfrac{\frac{d^2}{dx^2}p(x)}{p(x)}$ tem-se

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

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

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

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

$$x_{k+1} = x_k - \frac{nf(x_k)}{p'_n(x_k) \mp \sqrt{H(x_k)}}, \quad k = 1, 2, ... \tag{1.48}$$

que é a equação iterativa do método de Laguerre. Onde

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

Tem-se nessa expressão n o grau do polinômio. Sendo iniciado 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 a x_1

Ao usar o sinal em (1.48) que da origem ao menor incremento e adiciona-lo a x_k para se obter x_{k+1} a sequência gerada converge para a raiz mais próximo do valor inicial. Para equações algébricas onde todas suas raízes são reais e uma aproximação inicial x_1 que pode ser a menor ou maior das raízes $x_0 < x_1^*$ $(x_0 > x_n^*)$, então se $x_k < x_{k+1} < x_1^*$ $(x_k > x_{k+1} > x_n^*)$, logo o processo iterativo converge reciprocamente para a menor ou maior das raízes.

A convergência do método de Laguerre ocorre de forma global, pois o mesmo não depende de valores próximos imputados para a primeira iteração. Para raízes complexas não é verdade que a convergência do método é para qualquer valor inicial. 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, porém esse importuno é compensado pela convergência rápida, pois sua convergência é de ordem 3 (13).

O método de Laguerre possui vários algoritmos esse é um deles:

Algoritmo 1.5: Algoritmo sequencial Laguerre

Entrada: $n, (a_i : 0 \leqslant i \leqslant n), x_0, M, \varepsilon$
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 $\left|\frac{x_1 - x_0}{x_1}\right| \leqslant \varepsilon$ **então**
    Pare;
  fim
  $x_0 \longleftarrow x_1$
fim

2 Autovalores e Autovetores de matrizes tridiagonais simétricas

2.1 Métodos para determinação dos autovalores

Existem diversos métodos para a determinação dos autovalores e autovetores de matrizes tridiagonais simétricas, não se fará aqui tratamento ou comparação entre todos os métodos existentes, porém se faz necessário uma revisão dos principais métodos existentes juntamente com seus algoritmos especificamente para matrizes tridiagonais.

Ao se resolver problemas de autovalores que envolvam matrizes tridiagonais encontra-se basicamente três métodos: o método interativo de Francis , os métodos que adotam estrategias do gênero dividir para conquistar e o bissecção/multissecção (14).

Em (15) os autores fizeram uma comparação entre métodos que utilizados na determinação de autovalores e autovetores tanto em caso de matrizes simétricas, tridiagonais ou densas , onde os critérios observados para comparação foram: exatidão, a susceptibilidade ao transbordamento, velocidade de execução e necessidade de espaço para armazenamento.

Acerca da comparação entre os métodos os autores distingue o caso sequencial do paralelo no que diz respeito a matrizes tridiagonais densa e os requisitos do usuário quando se deseja calcular todos autovalores e autovetores ou parte dos mesmos. As conclusões a que os autores chegaram foram as seguintes:

i) O método da Bissecção/Multissecção é o mais exato dos três métodos analisados garantindo que a exatidão dos resultados só depende das limitações determinada pela arquitetura e dos dados de entrada e não pelo algoritmo.

ii) O método da Bissecção/Multissecção é o único método que permite determinar uma parte do espectro (conjunto dos autovalores de uma matriz), mesmo que em certos casos o referido método pode ser mais lento que o os métodos dividir para conquistar e o de Francis, especialmente se desejar determinar uma parte significativa dos autovalores e autovetores.

iii) Se a finalidade for determinar apenas os autovalores, nessa situação o método de Francis é o mais indicado, porém se precisar determinar os autovetores também ai o mais indicado é o dividir para conquistar.

iv) Quanto a paralelização somente os métodos Bissecção/Multissecção e o dividir para conquistar admitem uma paralelização eficiente, porém se desejar determinar uma grande quantidade de autovalores e autovetores o segundo é o mais indicado.

v) Em relação a armazenamento no caso de matrizes tridiagonais todos os métodos ocupam um espaço da ordem de $O(n)$ no diz respeito ao cálculo de autovalores, porém se a finalidade é calcular também os autovetores o método de Francis e o da Bissecção/Multissecção, permite que se use um espaço proporcional ao número k de autovetores com custo $O(nk)$, enquanto que o dividir para conquistar tem um custo de $O(n^2)$

Após essas considerações os autores recomendam que a importância maior deve estar na precisão dos resultados do que no desempenho do algoritmo e os mesmos ainda dizem: a rotina escolhida deve ser sempre a mais precisa e que não exija um espaço de armazenamento deliberadamente grande. Neste caso se os dados utilizados compreendem uma matriz tridiagonal o método a ser escolhido é o Bissecção/Multissecção.

2.2 Método da bisssecção e a sequência de Sturm

O uso do método da bissecção em matrizes tridiagonais simétricas se fundamenta no uso de uma qualidade própria dessa especie de matriz, que é a oportunidade de obter o valor do polinômio característico com um custo muito pequeno (14).

No entanto ao se referir a obtenção do valor do polinômio característico em uma matriz tridiagonal simétrica o mesmo não será obtido através de seus coeficientes, mas sim diretamente dos elementos da matriz.

A seguir a representação de uma matriz tridiagonal simétrica $T \in \Re^{nxn}$

$$T_n = \begin{bmatrix} a_1 & b_1 & & & \mathbf{0} \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \mathbf{0} & \\ & & \mathbf{0} & \mathbf{0} & b_{n-1} \\ \mathbf{0} & & & b_{n-1} & a_n \end{bmatrix} \tag{2.1}$$

E um valor real λ, assim o polinômio caraterístico de T em λ, toma a seguinte forma:

$$P_n(\lambda) = det(T - \lambda I) \tag{2.2}$$

Fazendo uso da sequência de Sturm que uma recorrência de segunda ordem que facilita o cálculo das raízes do polinômio característico em qualquer ponto, onde a

mesma é definida a seguir:

$$\begin{aligned} P_0(\lambda) &= 1 \\ P_1(\lambda) &= a_1 - \lambda \\ P_i(\lambda) &= (a_i - \lambda)P_{i-1}(\lambda) - b_{i-1}^2 P_{i-2}(\lambda) \;\; i = 2, 3, L, n \end{aligned} \tag{2.3}$$

Definindo T_i como uma submatriz de T de tamanho ixi a sequencia de Sturm se fundamenta no cálculo recursivo do polinômio característico das submatrizes, assim sendo para calcular o polinômio característico da matriz T, basta calcular o polinômio das submatrizes.

A utilização da sequência de Sturm associada ao método da bissecção tem sua justificativa em uma propriedade que se relaciona com os autovalores da matriz T, juntamente com a propriedade que diz que λ é um autovalor de T se $P_n(\lambda) = 0$

Antes de enunciar esse teorema faz-se necessário conhecer dois outros resultados que estão listados a seguir:

Propriedade 2 *Propriedade do estrito entrelaçado*
Seja A_{r-1} e A_r as submatrizes principais de tamanho $(r-1)$ e r de uma matriz simétrica A $\in \Re^{nxn}$. Vale o seguinte:

$$\mu_1 < \mu_2 < L < \mu_{r-1}$$

que são os autovalores de A_{r-1} e

$$\lambda_1 \leqslant \mu_1 \leqslant \lambda_2 \leqslant \mu_2 \leqslant L \leqslant \lambda_{r-1} \leqslant \mu_{r-1} \leqslant \lambda_r$$

A demostração pode ser vista em (5) A propriedade citada acima diz que os autovalores de uma matriz A servem como separadores para uma matriz de ordem imediatamente inferior, porém se for considerado $\mu_0 = -\infty$ e $\mu_r = +\infty$ a reciproca da propriedade se cumpre, ou seja os autovalores de uma matriz de tamanho $r-1$ servem de separadores, essa separação nem sempre é rigorosa, porém para o caso de matrizes tridiagonais simétricas irredutíveis essa separação funciona rigorosamente. Uma matriz tridiagonal simétrica é dita irredutível quando todos os elementos de sua subdiagonal bem como da sua superdiagonal são diferentes de zero, isto é, $b_i \neq 0 (i = 1, 2, ..., n-1)$. A condição de que a matriz seja irredutível não afeta a generalidade do resultado apresentado a seguir, visto que no caso de não existir elementos iguais a zero na diagonal pode-se dividir a matriz em blocos triangulares irredutíveis aos quais se pode aplicar a propriedade. (14)

Propriedade 3 *Propriedade estrita do entrelaçado Seja $T \in \Re^{nxn}$ uma matriz tridiagonal simétrica irredutivel e seja T_{r-1} e T_r duas de submatrizes cujos autovalores são respectivamente:*

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

e

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

De onde se tem:

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

A demonstração pode ser vista em (5).

A propriedade 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 um μ qualquer a partir de Sturm $i = r$ de modo que b_r seja diferente de zero,$P_{r-2}(\mu) = 0$, dando continuidade para $i = r - 1$, chega-se que $P_{r-3}(\mu) = 0$, seguindo adiante até $r = 0$ o que contradiz a definição que diz que $P_0(\mu) = 1$ (14).

Propriedade 4 *Propriedade da sequência de Sturm*
Seja $T \in \Re^{nxn}$ uma matriz tridiagonal simétrica irredutivel e λ um número real, então o número de trocas de sinais na sequência de Sturm

$$\{P_0(\lambda), P_1(\lambda), L, P_n(\lambda)\} \tag{2.4}$$

é igual ao número de autovalores de T que são menores que λ. Nessa propriedade faz-se uso da convenção de que $P_r(\lambda)$ possui sinal oposto de $P_{r-1}(\lambda)$, quando $P_r(\lambda) = 0$ para que tenha validade para qualquer valor de λ

A demonstração pode ser encontrada em (5)

Aqui faz-se necessário definir um função $neg_n(\lambda)$, onde esta função retornará o número de trocas de sinais da sequência de (2.4) em um ponto λ.

Em (16), onde pela primeira vez o autor percebeu que poderia utilizar a sequência de Sturm para calcular os autovalores de uma matriz tridiagonal simétrica. Assim como definido em (2.3), o cálculo da sequência de Sturm em um ponto qualquer a e a função associada $neg_n(\lambda)$, possibilita a divisão do espectro da matriz T bem como decidir quantos e quais autovalores da matriz são menores que a, quantos e quais são menores que o ponto de divisão.
Se ao invés de um ponto a aplicar a ideia em dois ponto α e β, pode-se determinar quais autovalores estão contidos no intervalo $[\alpha, \beta)$. De maneira concreta se define $na = neg_n(\alpha)$ e $nb = neg_b(\beta)$, onde os autovalores da matriz T que estão contidos no referido intervalos são:

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

Se o intervalo citado anteriormente for dividido de acordo com um ponto μ, existe a possibilidade de se especificar a posição de cada um dos autovalores citados no

intervalo anterior sem a necessidade do cálculo de $nc = neg_n(\mu)$ continuando o raciocínio anterior indubitavelmente a aplicação de um esquema de bissecção traz possibilidade de calcular o autovalor λ_i, onde o mesmo esta contido no intervalo inicial, com a precisão limitada apenas pelo número de iteração executadas pela bissecção, da arquitetura utilizada e dos dados do problema.

Ao observar a recorrência (2.3) que possibilita a realização do cálculo da sequência de Sturm, porém quando n se torna muito grande pode aparecer graves problemas de transbordamento. Esse problema foi identificado por W.Brth, R.S. Martin e J. H. Wilknson em (17), onde propuseram o uso de um sequência de Sturm modificada, listada a seguir:

$$q_i(\lambda) = \frac{P_i(\lambda)}{P_{i-1}(\lambda)} i = 1, 2, L, n \tag{2.5}$$

Utilizando (2.3) percebe-se que para calcular a função anterior, pode-se fazer uso da recorrência a seguir que é de primeira ordem:

$$\begin{aligned} q_0(\lambda) &= 1 \\ q_1(\lambda) &= a_1 - \lambda \\ q_i(\lambda) &= (a_i - \lambda) - \frac{b_{i-1}^2}{q_{i-1}(\lambda)} \quad i = 2, 3, L, n \end{aligned} \tag{2.6}$$

Ao fazer uso de quocientes sucessivos na sequência original de Sturm praticamente se elimina o problema de transbordamento ao desenvolver o cálculo da sequência.

Pode-se também notar que o número de trocas de sinais entre os sucessivos elementos pertencentes a uma sequência de Sturm denotada por $neg_n(\lambda)$ concorda com o número de sinais negativos apresentados em (2.5) (14).

A sequência de Sturm modificada diferencia-se da primeira pela introdução de um quociente o pode possibilitar um divisão por zero, durante a execução de (2.6). Alguns autores propuseram soluções para esse problema.
Em (18) os autores propuseram a substituição dos termo $q_{i-1}(\lambda)$ por uma variante desse termo onde a mesma pode ser vista abaixo:
$\tilde{q}_{i-1}(\lambda) = q_{i-1}(\lambda) \pm \varepsilon$,
Onde ε é um número muito pequeno, onde se soma $q_{i-1}(\lambda) \geqslant 0$ ou subtrai em outros casos. Com esta modificação evita-se a possibilidade de ocorrer uma divisão por zero durante a execução da recorrência (2.6).

Outro autor De Ros, citado por (14) propôs uma solução semelhante a anterior que consiste em trocar $q_{i-1}(\lambda)$ por um valor ε muito pequeno quando a função assumir valor zero durante a execução da recorrência. Esta modificação equivale a trocar o termo a_{i-1} da matriz T pelo termo $a_{i-1+}\varepsilon$, esse processo envolve uma pertubação mesmo que pequena em um elemento da matriz T.

Uma outra solução foi proposta pelos autores Li e Zeng, que fundamenta-se na aplicação das seguintes modificações se a função $q_i(\lambda)$ for igual a zero:

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

Neste caso se $a_i = \lambda$ a transformação realizada é a substituição de a_1 por $a_1 + b_1^2\varepsilon^2$, ao passo que quando $q_i(\lambda) = 0$ com $i > 1$, a correção deverá ser feita substituindo b_{i-1} por $b_{i-1}(1-\varepsilon^2)^{\frac{1}{2}}$ (14).

Em qualquer situação a opção por uma das modificações citadas acima e sua consequente aplicação na recorrência (2.4), isso acarretará pertubações mesmo que muito pequenas nos dados do problema ou seja nos elementos da matriz (14).

2.3 O método da bisecção e sua estabilidade

Uma característica essencial do problema de calcular o autovalor de matrizes simétricas é o bom condicionamento, isto é, variações pequenas nos dados do problema provocam alterações menores nos resultados. Essa característica aparece no teorema a seguir (14):

Teorema 8 *Estabilidade de autovalores de matrizes simétricas*
Seja $A \in \Re^{nxn}$ uma matriz simétrica. Seja $\tilde{A} = A + E$, com E uma matriz de pertubação simétrica e seja $\lambda_1 \leqslant \lambda_2 \leqslant L \leqslant \lambda_n$ os autovalores de A e $\tilde{\lambda}_1 \leqslant \tilde{\lambda}_2 \leqslant L \leqslant \tilde{\lambda}_n$ os de $\tilde{\lambda}$, então

$$|\tilde{\lambda}_i - \lambda_i| \leqslant \|E\|_2 \ \ i = 1, 2, L, n$$

A demonstração pode ser vista em (19).

Portanto ao demonstrar a estabilidade do método da bissecção pode se garantir que mediante a pequeno erros apresentados nos dados, os autovalores calculados estão bem próximos dos autovetores da matriz considerada.

Em (5), existe uma demonstração que envolve o método da bissecção puro com sequência de Sturm (2.3) que fica evidenciado a grande estabilidade. De maneira objetiva demonstra-se que o cálculo do determinante

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

Que calcula o valor exato do polinômio característico de uma matriz perturbada $T + \delta T$, onde
$|\delta a_i| \leqslant 3.01\varepsilon(|a_i| + |\lambda|)$
$|\delta b_i| \leqslant 1.51\varepsilon|b_i|$

onde ε é um número muito pequeno e partindo de
$-\|T\|_\infty \leqslant \lambda_i \leqslant +\|T\|_\infty$
Usando o teorema da monotocidade de Weyl tem-se:
$|\tilde{\lambda}_i - \lambda_i| \leqslant \|\delta T\|_\infty = max\{|\delta b_{i-1}| + |\delta a_i|\} \leqslant 3.01\varepsilon(|a_i| + |\lambda|) + 3.02\varepsilon|b_i| \cong$
$\cong 3.02\varepsilon(|a_i| + |b_i| + |\lambda|)$.

2.4 Algoritmo elementar do método da bissecção

Algoritmos que tem como base o método da bissecção/multissecção possui privilégio sobre outros métodos tais como o iterativo de Francis e o baseados em dividir para conquistar, por apresentar grande flexibilidade. Ao fazer uso desse algoritmo pode calcular todos os autovalores de uma matriz ou parte deles de maneira seletiva, o método da bissecção atende perfeitamente a esses requisitos (14). O método da bissecção pode ser utilizado para responder as seguintes condições:

i) Calcular um autovalor qualquer λ_i de acordo com o indíce i num conjunto ordenado

$$\lambda_1 < \lambda_2 < ... < \lambda_n$$

ii) Calcular todos os autovalores de uma matriz contidos em um intervalo real delimitado por (a,b).

iii) Calcular todos os autovalores de uma matriz onde seus índices podem ser encontrados entre os valores inteiros {i,j}.

iv) Calcular todos os autovalores de uma matriz.

Dada uma matriz tridiagonal $T \in \Re^{nxn}$ e um valor λ qualquer real que não seja autovalor da mesma. Existe um algoritmo que permite o cálculo da função $neg_n(\lambda)$. Em primeiro lugar ver-se a como resolver o problema proposto no item i em seguida expandir o método para compreender como resolver o item iii e os itens ii e iv podem ser resolvidos com uma variação do item iii (14).

2.5 Cálculo de um único autovalor

Em uma primeira aproximação através do uso da função $neg_n(\lambda)$ tem como base a aplicação dessa função em um ponto a que determina a posição de um autovalor qualquer em relação ao ponto a em uma reta real.Ao se aplicar a função em dois pontos quaisquer α e β, obtém-se os valores naturais na e nb, então o número de autovalores

de T estão contidos no intervalo $[\alpha, \beta)$ e dado por $nb - na$. Suponha que se possa ordenar os autovalores da matriz T da maneira a seguir:

$$\lambda_1 < \lambda_2 < ... < \lambda_n$$

e deseja-se calcular um valor qualquer λ_i de T

A aplicação do método da bissecção nesta situação fundamenta-se em iniciar em um intervalo (α, β) onde esteja contido o autovalor procurado e utilizar a função $neg_n(\lambda)$ para determinar a posição de λ_i nesse intervalo com a precisão requerida.

A primeira ação a ser feita é determinar o intervalo (α, β)que contém o autovalor a ser encontrado, ou seja:

i) $neg_n(\alpha) < i$

ii) $neg_n(\beta) \geqslant i$

Na obtenção de α e β corretos pode-se utilizar o método de tentativa e erro, testando vários pontos que obedeçam as prerrogativas anteriores, ou pode-se fazer uso do teorema de Gershgorin associado a matriz T (14).

Teorema 9 *Teorema dos discos de Gershgorin Seja uma matriz $A \in \mathbb{C}^{nxn}$, então se $X^{-1}AX = D + F$ com $D = diag\ (d_1, d_2, ..., d_n)$ e seja F uma matriz com elementos iguais a zero na diagonal, então*

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

onde $D_i = \left\{ z \in \mathfrak{C}^{nxn} : |z - d_i| \leqslant \sum_{j=1}^{n} |f_{ij}| \right\}$ *i = 1, 2, L, n*

que são os discos de Gershgorin para a matriz A.

Este teorema afirma que cada autovalor da matriz A esta contido dentro de um dos discos de Gershgorin, porém o mesmo não garante que os discos são separados ou que contenha um único autovalor. Quando a matriz A $\in \Re^{nxn}$ pertença a classe das matrizes simétricas, os discos de Gershgorin passam a ser intervalos pertencentes a reta real, sendo definido da seguinte forma:

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

, Uma maneira de garantir a existência de um autovalor qualquer dentro de um intervalo é usar o superior e inferior de todos os discos de Gershgorin. Utilizando-se dessa estratégia assegura-se que o intervalo obtido contenha todos os autovalores da matriz, consequentemente o autovalor procurado (14).

Ao determinar o intervalo inicial no caso de matrizes tridiagonais simétricas os extremos desse intervalo toma a seguinte forma:

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

Após a definição do intervalo inicial (α, β), que contém o autovalor λ_i a ser determinado, procede-se a aplicação do método da bissecção a partir do intervalo inicial. Usa-se a função $neg_n(\lambda)$ para definir qual dos subintervalos resultantes da aplicação da bissecção, deve-se 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 tamanho do intervalo que contém cada um dos autovalores procurados são divididos por dois a cada iteração executada pelo método da bisseção, portanto apos k iterações tem-se a delimitação do autovetor dentro de um intervalo de amplitude $\frac{(\alpha - \beta)}{2^k}$, onde α e β são os limites do intervalo inicial (14).

Algoritmo 2.1: 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
```

Através da aplicação desse algoritmo pode-se obter qualquer autovalor da matriz T, com garantia da convergência do método da bissecção para o autovalor que se deseja obter, porém esse método apresenta o inconveniente de ser muito lento sua convergência para o autovalor que se quer encontrar. Seria interessante encontrar uma maneira de acelerar a convergência do algoritmo acima, isso pode ser feito combinando o algoritmo simples da bissecção com outra técnica de busca que possui uma velocidade de convergência maior.

Para-se utilizar esse novo método observa-se a sequência (2.3), combinada com a matriz T, em um ponto λ. Usando essa sequência define-se um função nova $q_n(\lambda)$ que é determinada exatamente pelo último valor no ponto λ, isto é dado um número real λ qualquer tem-se um algoritmo para obter a sequência (2.3) neste ponto, pode-se

conseguir o valor da função $q_n(\lambda)$ (14).

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

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

Tomando como base (2.4) λ será autovalor de T se e somente se $det(LDL^t)$ onde L é uma matriz unidade triangular inferior, ou seja λ é um autovalor de T se e somente se $det(D) = 0$
então tem-se que:

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

Se usar a função

$$P_m(\lambda) = det(T_m - \lambda I_m) \qquad m = 1, 2, ..., n$$

Com T_m sendo a principal submatriz de T com dimensão nxn, então

$$P_m(\lambda) = \prod_{i=1}^{m} q_i(\lambda) \qquad m = 1, 2, L, ..., n$$

Logo

$$q_n(\lambda) = \frac{P_n(\lambda)}{P_{n-1}(\lambda)} \tag{2.7}$$

Assim qualquer valor de λ que torne a função $q_{(\lambda)}$ nula será um autovalor da matriz T.Dessa forma o problema de determinar os autovalores da matriz tridiagonal simétrica T consiste em encontrar as raízes da função $q_n(\lambda)$.

Retornando ao caso de determinar o autovalor λ_i da matriz T, suponha que de alguma maneira tenha-se determinado o intervalo (α, β) que pode ou não conter um autovalor da matriz T. O autovalor λ_i poderá ser a única raíz da função $q_n(\lambda)$ neste intervalo. Dessa forma pode-se usar uma técnica de busca de raízes com convergência mais rápida do a bissecção simples na determinação de qualquer autovetor da matriz T.

Ainda se tem o problema da determinação do intervalo (α, β) que contenha o autovalor λ_i da matriz T. Doravante esse problema será denominado isolamento do autovalor da matriz T. Após o isolamento necessita-se fazer a extração dos autovalores da matriz T, doravante denominado extração do autovetor (14).

2.5.1 Critério de parada do algoritmo

Suponha que ao executar o método da bissecção tem-se um sequência de aproximação $\left\{\tilde{\lambda}_i\right\}_{k=1}$ cada vez mais próximo do autovalor λ_i. De acordo com 2.3 pode-se definir o seguinte critério de parada para o algoritmo da bisssecção:

$$|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}| \leqslant max\left\{\delta, |\tilde{\lambda}_i^{(k)}|\varepsilon\right\} \tag{2.8}$$

onde ε representa o arrendondamento da unidade da máquina e δ é um cota de erro absoluto definida como:

$$\delta = 2.5\varepsilon max\left\{|b_i| + |b_{i+1}|\right\}$$

Assim considera-se que um autovalor esta convergindo quando a diferença entre duas aproximações seguidas for menor que uma cota pré-estabelecida tanto para um erro absoluto δ, ou para o erro relativo a ultima aproximação obtida $|\tilde{\lambda}_i^{(k)}|\varepsilon$. Outros autores como Li e Zeng, fazem uso de outro critério de parada semelhante ao anterior, porém com menor número de restrições. Esses autores comprovaram na prática que esse critério nunca falha, desde que obedeça (2.8) (14).

$$\frac{|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}|^2}{|\tilde{\lambda}_i^{(k-1)} - \tilde{\lambda}_i^{(k-2)}|} \leqslant 2.5\varepsilon max\left\{\ |b_i|, |b_{i-1}|\right\} + |\tilde{\lambda}_i^{(k)}|\varepsilon$$

2.5.2 Isolamento do autovalor

O problema agora é isolar o autovalor λ_i da matriz T, o mesmo pode ser traduzido como: Dada uma matriz tridiagonal simétrica $T \in \Re^{nxn}$, tomando-se dois números reais positivos sendo eles ε, que representa o arrendondamento da unidade da máquina e δ que representa uma cota de erro. Tomando-se ainda um intervalo real (α_0, β_0), onde $\alpha_0 < \beta_0$ de maneira que esse intervalo contenha o autovalor λ_i da matriz T ao mesmo tempo este intervalo poderá conter outros autovalores da matriz T, os números reais, α e β devem ser calculados da seguinte maneira $\alpha_0 \leqslant \alpha < \beta \leqslant \beta_0$ que contenha somente o autovalor λ_i da matriz T ou a seguinte desigualdade seja satisfeita:

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

A resolução inicialmente parte de um intervalo (α_0, β_0) que contém o autovalor procurado juntamente com os demais, após a inicialização tem-se um outro intervalo isolado (α, β), intervalo este que contém apenas λ_i, para que isso ocorra esse intervalo isolado deve obedecer o seguinte critério:

$$neg_n = (\alpha) = i - 1 \ \ e \ \ neg_n(\beta) = i \tag{2.9}$$

Tem-se aqui um segundo critério para a solução do problema do isolamento. De acordo com este se for delimitado um intervalo que contenha apenas o autovalor procurado e

o intervalo tenha tamanho mínimo, para-se o algoritmo, pois ai existe um *cluster* de autovalores (14).

Definição 9 *Cluster de autovalores*
cluster de autovalores é definido como sendo $\lambda_i < ... < \lambda_j, \quad j > i$ *que é um grupo de autovalores consecutivos contidos em um intervalo de tamanho menor que um dada cota* ε.

Considera-se que vários autovalores formam um *cluster* de autovalores se os mesmos são aritmeticamente diferentes, porém numericamente indistinguíveis. Mas a definição anterior coloca um condição menos restritiva e exige somente que sua proximidade seja menor que uma cota ε muito pequena.
Nesta hipótese considera-se 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

$$|\beta - \alpha| \leqslant max\{\delta, max\{|\alpha|, |\beta|\}\varepsilon\}$$

Como o *cluster* de autovalores contido no intervalo (α, β) obedecerá a seguinte sequência $\lambda_i < \lambda_{i+1} < ... < \lambda_j$.

Ao encontrar um *cluster* de autovalores finaliza-se a aproximação dos mesmos nesse ponto. Não se avança para o próximo passo da extração, sem que a aproximação de todos os autovalores tenha alcançado a precisão desejada.

O isolamento que tem por fundamento a aplicação sucessiva do método da bissecção partindo de um intervalo inicial até seja cumprida as condições citada acima (14).

Algoritmo 2.2: Algoritmo isolamento dos autovalores

```
1  Programa isolar (a, b, δ, ε)
2  Calcular um intervalo (a_0, b_0) tal que
     neg_n(a_0) < i e neg_n(b_0) ⩾ i
3  a = a_0
4  b = b_0
5  repita
6      c = (a + b)/2;
7      se neg_n < i então
8          a=c;
9          senão
10             b=c
11         fim
12     fim
13 até neg_n(a) = i − 1  e  neg_n(b) =
     i  ou  (|b − a| ⩽ max{δ, max{|a|, |b|}ε});
```

2.5.3 Extração dos autovalores

Nesta etapa parte-se de um intervalo isolado que contém a aproximação do autovalor λ_i , onde essa aproximação deve ser feita de acordo com a precisão estabelecida. O problema da extração dos autovalores pode ser enunciado da seguinte forma:

Dada um matriz simétrica tridiagonal $T \in \Re^{(nxn)}$, tomando-se dois números reais positivos sendo eles ε, que representa o arrendondamento da unidade da máquina e δ que representa uma cota de erro. Tomando-se ainda um intervalo real (α, β) com $\alpha < \beta$, intervalo este que contém um só autovalor λ_i da matriz T. Agora calcula-se uma sequência de aproximação para o autovalor λ_i da matriz T da seguinte forma: $\left\{ \tilde{\lambda}_i^{(l)} \right\}_{l=1}^{k}$ tal que $|\tilde{\lambda}_i^{(k)} - \tilde{\lambda}_i^{(k-1)}| \leqslant max \left\{ \delta, |\tilde{\lambda}_i^{(k)}| \varepsilon \right\}$ Para proceder a extração do autovalor isolado utiliza-se um algoritmo de busca com uma velocidade de convergência superior a do método da bissecção pura é a possibilidade de aceleração desse método apareceu em (5) que propõe uma aproximação feita utilizando o método de Newton que possui convergência quadrática ou método iterativo de La Guerre que possui convergência cúbica.

Existe diversos métodos a seguir faz-se um estudo de alguns deles que permitem a determinação de autovalores de matrizes tridiagonais simétricas.

2.5.3.1 Método da secante

O método da secante consiste em fazer a aproximação da raiz da função $q_n(\lambda)$ em (2.7), procurando uma intersecção da reta secante com com o eixo real da função em dois pontos $y_1 = q_n(x_1)$ e $y_2 = q_n(x_2)$, levando-se em consideração que a raiz esta contida em um intervalo (x_1, x_2).

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

No caso especifico da função $q_n(\lambda)$ em (2.7), sabe-se que a mesma é contínua em uma porção de intervalos definidos pelas raízes de $q_{n-1}(\lambda)$, porém essa mesma função apresenta uma gama de pontos de descontinuidade nos autovalores das submatrizes principal T_{n-1} de T, $\mu_1 < \mu_2 < ... < \mu_{n-1}$.

Ao se executar o método da secante existe a necessidade de se obedecer uma condição a mais além da (2.9) para que no intervalo (α, β), não esteja contido nem um dos pontos μ_i. Para que isso ocorra além de (2.9) é necessário satisfazer a seguinte condição:

$$q_n(\alpha) > 0 \quad e \quad q_n(\beta) < 0 \tag{2.10}$$

Definição 10 *Autovalor defeituoso*
Seja $A \in \Re^{nxn}$ 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 a seguir.

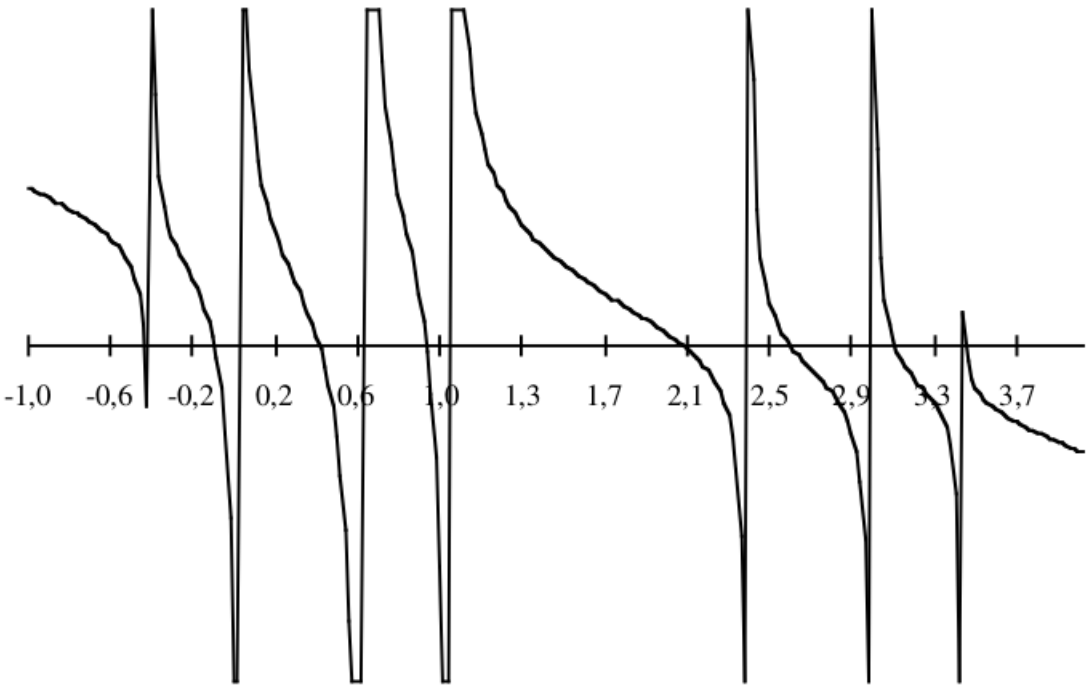
Teorema 10 *Teorema do isolamento estrito*
Sejam dois números reais α e β, que são números reais não defeituoso em relação a matriz T e que o intervalo (α, β) tenha um único autovalor (com multiplicidade 1) em T. Suponha que a α e β não sejam autovalores da matriz T. Então o intervalo (α, β) não contém autovalores de T_{n-1}, se e somente se, $q_n(\alpha) > 0$ e $q_n(\beta) < 0$.

Este teorema possibilita especificar aproximadamente a configuração da função com respeito aos autovalores das matrizes T e T_{n-1}. Para iniciar denota-se os autovalores da matriz T_{n-1} como $\mu_1, \mu_2, ..., \mu_{n-1}$ e os autovalores da matriz T por $\lambda_1, \lambda_2, ..., \lambda_n$. pelo teorema do entrelaçamento estrito tem-se:

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

Dessa forma os autovalores da matriz T, exceto o primeiro e o último estão contidos em intervalos, onde as bordas dos mesmos são autovalores da submatriz T_{n-1}. Sabe-se que a função $q_n(\lambda)$ é zero para um λ_i e terá valores positivos e negativos nestes pontos, é sabido também que não esta definida para os pontos μ_i, pois os mesmos tornam zero o denominador da função. De posse dessas informações pode-se esboçar o gráfico dessa função (14).

Figura 5 – Gráfico da função $q_n(\lambda)$



Fonte:(14)

Ao observar o gráfico percebe-se que os autovalores dessa matriz coincidem com as intersecções da função $q_n(\lambda)$ com o eixo das abscissas que a matariz tem pontos de descontinuidade nos autovalores de uma matriz de ordem 7.

Para assegurar o funcionamento de (2.10) e evitar uma sucessão de etapa do método da bissecção durante o isolamento, pode utilizar o algoritmo a seguir:

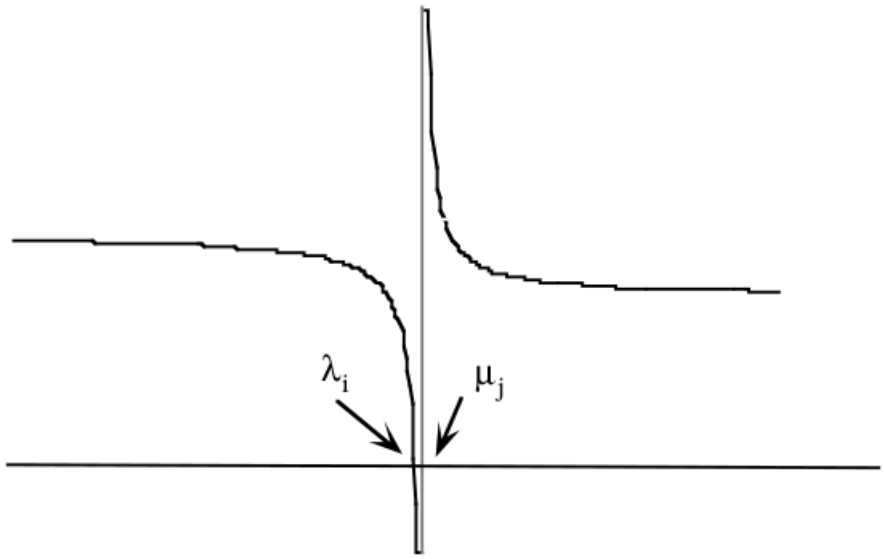
Algoritmo 2.3: Algoritmo do isolamento estrito

```
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 um cota estabelecida, nessa situação pode ocorrer que não se pode distinguir um autovalor da matriz T de um 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 diz-se que tem um autovalor oculto (5).

Graficamente esse fenômeno ocorre quando um autovalor λ_i de T esta 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.

Figura 6 – Autovalor oculto



Fonte:(14)

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

A extração se fundamenta nos requisitos (2.9) e (2.10), tendo a certeza que o intervalo isolado doravante chamado de (x_1, x_2) esteja entre os pontos μ_{i-1} e μ_i. De acordo com essa 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

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

Essa equação determina uma reta com coeficiente angular m definida como

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

E o ponto de intersecção com a secante e 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 de o valor x for maior 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 autovetor, obedecendo o limite de uma determinada cota. Se em alguma das iterações do método a aproximação alcançada estiver fora do intervalo que contém o autovalor procurado, escolhe como aproximação o ponto médio e dá-se prosseguimento ao método de extração. Por outro lado se a quantidade de iterações é maior

que um cota estabelecida, deve-se parar o método para evitar um *loop* infinito (14).

Algoritmo 2.4: Algoritmo método da secante

```
1  programa extrasec (x_1, x_2, y_1, y_2, δ, ε, x)
2  isola-extra(x_1, x_2, δ, ε)
3  repita
4      prevx = x
5      m = (y_1 − y_2)/(x_1 − x_2)
6      x = x_1 − (1/m) ∗ y_1
7      calcular y = f(x) ;
8      se y ∗ y_1 > 0 então
9          prevy = y_1
10         y_1 = y
11         x_1 = x
12     senão
13         prevy = y_2
14         y_2 = y
15         x_2 = x
16     fim
17 até |x − prevx| < max{δ, |x|ε};
```

2.5.3.2 Método de pegasus

O método de pegasus é resultado de uma modificação aplicada no método da falsa posição, porém com velocidade de convergência maior que o método da falsa posição. O sistema de execução desse método é semelhante ao da secante no que diz respeito as iterações e aproximações, porém no mesmo existe um processo de aceleração na busca da raiz procurada. A finalidade da introdução dessa modificação é evitar que a convergência seja lenta em torno de um ponto procurado quando este se encontra ao lado do ponto de corte. Para evita essa situação, quando o ponto de corte x achar-se mais de uma vez consecutiva do mesmo lado do autovalor, introduz-se uma mudança na fórmula da reta tangente de maneira que o ponto de corte tenha um valor muito maior que o autovalor procurado. Por exemplo se o valor x está duas vezes consecutivas a esquerda do autovalor, logo o valor da função do lado direito passará a ser

$$y_m^2 = \frac{y_2^{(k)} y_1^{(k)}}{y_1^{(k)} + y_2^{(k+1)}}$$

Onde os subscritos representam o extremo do intervalo que contém o valor da função e os expoentes indicam a quantidade de iterações realizadas pelo método de aproximação. Diante do exposto acima o algoritmo do método segue abaixo:

Algoritmo 2.5: Algoritmo método da pegasus

```
1  programa extrae-peg (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     fim
13     se (Se tiver repetido mais de uma vez
       consecutivamente esta condição) então
14         y2 = (y2 ∗ prevy)/(prevy + y1)
15     senão
16         prevy = y2
17         y2 = y
18         x2 = x
19     fim
20     se (Se tiver repetido mais de uma vez
       consecutivamente esta condição) então
21         y1 = (y1 ∗ prevy)/(prevy + y2)
22     fim
23 até |x − prevx| < max{δ, |x|ε};
```

O algoritmo acima tem como fator de entrada o intervalo isolado (x_1, x_2) no qual o autovalor se encontra bem como o valor da função $q_n(\lambda)$ nos extremos desse intervalo y_1, y_2 e as cotas de erro δ e ε, ao passo que o resultado da execução do algoritmo é o autovalor x aproximado (14).

2.5.3.3 Método de Newton

Esse método tem como fundamento a expressão a seguir que calcula as aproximações sucessivas da raiz procurada.

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

Especificamente quando se procura as raízes do polinômio característico, então $f(x) = p_n(x)$. Assim sendo 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 (2.3) obtêm-se a sequência a seguir:

$$\begin{aligned} p'_0(\lambda) &= 0 \\ p'_1(\lambda) &= -1 \\ p'_i(\lambda) &= (a_i - \lambda)p'_{i-1}(\lambda) - b^2_{i-1}p'_{i-2}(\lambda) \end{aligned} \tag{2.11}$$

A aplicação de (2.11) pode acarretar diversos problemas de transbordamento. A solução encontrada na literatura é calcular alternativamente a sequência a baixo:
$N_i = \dfrac{p'_i(\lambda)}{p_i(\lambda)} \qquad i$= 1,2, L, n.

Depois de calcar N_i sua inversa será a correção do método de Newton para obter a aproximação seguinte para a raiz. Calculando uma sequência para (2.11) obtêm-se:

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

Onde:

$$\begin{aligned} N_0 &= 0 \\ N_1 &= \frac{-1}{a_i - \lambda} \\ N_i &= \frac{1}{q_i(\lambda)}\left[(a_i - \lambda)N_{i-1} - 1 - \frac{b^2_{i-1}}{q_{i-1}(\lambda)}N_{i-2}\right] \end{aligned} \tag{2.12}$$

Em (20) há a proposição de uma formula diferente para calcular N_n que tem um custo menor. Esta fórmula esta baseada no seguinte:

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

Onde a relação a seguir é verificada:

$$N_i = R_i + N_{i-1} \qquad i = 1, 2, L, n. \tag{2.13}$$

Utilizando a definição de $q_i(\lambda)$ tem-se:

$p_i = q_i p_{i-1}$
Derivando a igualdade acima obtêm-se:

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

Dividindo-se por p_i chega-se a :

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

Derivando (2.6) obtêm-se:

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

Dividindo por $q_i(\lambda)$obtêm-se:

$$\begin{aligned} R_0 &= 0 \\ R_1 &= \frac{-1}{q_1(\lambda)} \\ R_i &= \frac{1}{q_i(\lambda)} \left(-1 + \frac{b^2_{i-1}}{q_{i-1}(\lambda)} R_{i-1} \right) \quad i = 2, 3, L, n. \end{aligned} \tag{2.14}$$

O uso da sequência (2.14), juntamento com (2.13) possibilita o cálculo de N_n, usando um sequência de primeira ordem ao invés de uma de segunda ordem como em (2.12). O uso de uma sequência de primeira ordem reduz o custo computacional, pois elimina-se o cálculo de um produto no método de Newton (14). A seguir o algoritmo que realiza esse cálculo.

Algoritmo 2.6: Algoritmo zeroinNR

```
1  zeroinNR (λ, a, b, n)
2  q = a_1 − λ
3  R = −1/q
4  N = R
5  para i = 2 até n faça
6  |  m = b²_{i−1}/q
7  |  q = (a_i − λ) − m
8  |  R = (m R − 1)/q
9  |  N = N + R
10 fim
11 λ = λ − 1/N
```

O algoritmo acima permite realizar cálculo de Nn e da função $q_n(\lambda)$. Por outro lado pode-se determinar $neg_n(\lambda)$ através da contagem de ocorrências negativas que aparecem durante a execução do algoritmo. A função $neg_n(\lambda)$ possibilita que se saiba 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 escolhe-se o ponto ponto de partida para a iteração de Newton, o ponto médio do intervalo escolhido, ou seja inicia-se com a aplicação de uma etapa do método da bissecção, isto esta presente no algoritmo a seguir (14).

Algoritmo 2.7: Algoritmo método de Newton

```
1  programa extrae-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);
```

2.5.3.4 Método de Laguerre

Esse método tem sua indicação em (5) como uma boa alternativa durante o período de extração do método da bissecção.

O método iterativo de Laguerre se fundamenta no cálculo de uma nova aproximação das raízes do polinômio característico, através da seguinte iteração:

$$L_{\pm}(x) = x + \frac{n}{\left(-\frac{p_n'(x)}{p_n(x)}\right) \pm \sqrt{(n-1)\left[(n-1)\left(-\frac{p_n'(x)}{p_n(x)}\right)^2 - n\left(\frac{p_n''(x)}{p_n(x)}\right)\right]}} \tag{2.15}$$

A iteração de Laguerre converge globalmente para raízes reais e simples e sua convergência é de ordem três ao se aproximar dos zeros. O teorema a seguir colabora para validar essas informações.

Teorema 11 *Convergência da iteração de Laguerre*
Seja $T \in \Re^{(nxn)}$ uma matriz tridiagonal simétrica irredutível. E sejam

$$\lambda_1 < \lambda_2 < ... < \lambda_n$$

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,...,n tem-se que:

i) $\lambda_i < L_-(x) < x < L_+(x) < \lambda_{i+1}$

ii) Existem duas contantes c_- *e* c_+ *de maneira que:*
$|L_+(x) - \lambda_{i+1}| \leqslant c_+|x - \lambda_{i+1}|^3$ *se estiver próximo de* λ_{i+1}

$|L_-(x) - \lambda_i| \leqslant c_-|x - \lambda_i|^3$ *se estiver próximo de* λ_i

A demonstração pode ser encontrada em (5). Esse teorema diz que a iteração de Laguerre aplicada tem como resultando duas sequências

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

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 esta representada em (2.15) e necessário calcular as funções $p_n(\lambda)$, $p'_n(\lambda)$ e $p''_n(\lambda)$ ou seja

$$\frac{p'_n(\lambda)}{p_n(\lambda)} \quad e \quad \frac{p''_n(\lambda)}{p_n(\lambda)}$$

Em (2.12) definiu-se a sequência para calcular

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

A seguir a representação de um sequência que possibilita o cálculo de

$$S_i = \frac{p''_i(\lambda)}{p_i(\lambda)}\ i = 1, 2, L, n.$$

Ao derivar (2.11), obtêm-se

$$\begin{aligned} p''_0(\lambda) &= 0 \\ p''_1(\lambda) &= 0 \\ p''_i(\lambda) &= (a_i - \lambda)p''_{i-1}(\lambda) - 2p'_{i-1}(\lambda) - b_{i-1}^2 p''_{i-2}(\lambda)\ i = 1, 3, L, n. \end{aligned} \tag{2.16}$$

Partindo de (2.11) utilizando raciocino semelhante a (2.12) obtêm-se

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

Assim sendo

$$\begin{aligned} S_0 &= 0 \\ S_1 &= 0 \\ S_i &= \frac{1}{q_i(\lambda)} = \left[(a_i - \lambda) S_{i-1} - 2Ni - 1 - \frac{b_{i-1}^2}{q_{i-1(\lambda)}} S_{i-1} \right] \quad i = 2, 3, L, n. \end{aligned} \tag{2.17}$$

Fazendo uso de (2.6), (2.12) e (2.17), implementa-se o algoritmo para calcular as iterações do método de Laguerre (14).

Algoritmo 2.8: Algoritmo 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 anteriormente 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 é o seguinte:

Algoritmo 2.9: Algoritmo método da Laguerre

```
1  programa extrae-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 o que foi dito sobre as duas formas do método da bissecção modificado, isto é no isolamento e extração o que produz é 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 2.10: Algoritmo método da bissecção modificada

1 $programacalcular - av(i, \lambda_i)$
2 calcular o intervalo inicial (a, b) contendo λ_1 utilizando os discos de Gershgorin.
3 /* Fase de isolamento */
4 isolar $(a, b, \varepsilon, \delta)$
5 /* Fase de extração */
6 extrair $(a, b, \varepsilon, \delta, \lambda_1)$

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

2.6 Método de Thomas

O método de Thomas é uma variante da eliminação gaussiana que lida com matrizes esparsas, a opção fundamental tem seu foco em matrizes tridiagonais. Esse algoritmo facilita a otimização do armazenamento e possui uma complexidade computacional de $O(n)$ (21).

Por ser um método direto o algoritmo de Thomas esta dividido em duas fases, a primeira constitui-se na normalização dos elementos da diagonal principal que recebe o nome de $forward$, onde os elementos da subdiagonal inferior são eliminados, na segunda fase o sistema é resolvido por retrosubstituição (22).

A segunda fase do algoritmo de Thomas tem um velocidade maior que a primeira, pois possui um número menor de operações aritméticas, porém a complexidade é de $O(n)$ para as duas fases. Para sistemas tridiagonais esse algoritmo exige um número de operações proporcional a n que é menor que $\frac{1}{3}n^3$, que é a complexidade exigida pela eliminação gaussiana (23).

Nesse método a matriz do coeficientes é guardada em três vetores, a, b e c, onde é guardado respectivamente diagonal inferior, principal e a superior, dessa forma o sistema pode ser visto da seguinte forma:

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & & \\ & a_3 & b_3 & c_3 & & & \\ & & & \ddots & & & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} & \\ & & & & a_n & b_n & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_{n-1} \\ r_n \end{bmatrix} \tag{2.18}$$

Não é verdade que na diagonal principal os b_j variam $j = 1, ..., n$, assim como também não é verdade que na subdiagonal abaixo os elementos a_j variam de 2 a n, bem como os elementos c_j da subdiagonal superior não tem a variação $1 \leqslant j \leqslant n - 1$. A alternativa é

escrever (2.18) da seguinte forma :

$$\begin{array}{rcrcrcrcrcl}
b_1x_1 & + & c_1x_2 & & & & & & & = & r_1 \\
a_2x_1 & + & b_2x_2 & + & c_2x_3 & & & & & = & r_2 \\
 & & a_3x_2 & + & b_3x_3 & + & c_3x_4 & & & = & r_3 \\
 & & & & & & & & & \vdots & \\
 & & & & & & a_{n-1}x_{n-2} & + & b_{n-1}x_{n-1} + c_{n-1}x_n & = & r_{n-1} \\
 & & & & & & & & a_nx_{n-1} + b_nx_n & = & r_n
\end{array} \tag{2.19}$$

O modo de resolver esse sistema de equações é combinado as equações de maneira a eliminar algumas incógnitas. Deve-se eliminar x_1 na segunda equação, multiplicando a primeira equação por $\frac{a_2}{b_1}$ e subtraindo a segunda, em seguida substitui-se a segunda equação pelo resultado da subtração tendo assim um novo sistema.

$$\begin{array}{rcrcrcrcrcl}
b_1x_1 & + & c_1x_2 & & & & & & & = & r_1 \\
 & & (b_2 - \frac{a_2}{b_1})x_2 & + & c_2x_3 & & & & & = & r_2 - \frac{a_2}{b_1}r_1 \\
 & & a_3x_2 & + & b_3x_3 & + & c_3x_4 & & & = & r_3 \\
 & & & & & & & & & \vdots & \\
 & & & & & & a_{n-1}x_{n-2} & + & b_{n-1}x_{n-1} + c_{n-1}x_n & = & r_{n-1} \\
 & & & & & & & & a_nx_{n-1} + b_nx_n & = & r_n
\end{array} \tag{2.20}$$

Para efeito de simplificação de notação introduz-se a seguinte mudança:

$$\beta_1 = b_1, \qquad \rho_1 = r_1 \tag{2.21}$$

e

$$\beta_2 = b_2 - \frac{a_2}{\beta_1}c_1, \qquad \rho_2 = r_2 - \frac{a_2}{\beta_1}\rho_1 \tag{2.22}$$

Substituindo (2.21) e (2.22) em (2.20) tem-se:

$$\begin{array}{rcrcrcrcrcl}
\beta_1x_1 & + & c_1x_2 & & & & & & & = & \rho_1 \\
 & & \beta_2x_2 & + & c_2x_3 & & & & & = & \rho_2 \\
 & & a_3x_2 & + & b_3x_3 & + & c_3x_4 & & & = & r_3 \\
 & & & & & & & & & \vdots & \\
 & & & & & & a_{n-1}x_{n-2} & + & b_{n-1}x_{n-1} + c_{n-1}x_n & = & r_{n-1} \\
 & & & & & & & & a_nx_{n-1} + b_nx_n & = & r_n
\end{array} \tag{2.23}$$

Elimina-se agora x_2, multiplica-se a segunda equação por $\frac{a_3}{\beta_2}$ e subtraindo da terceira, obtêm-se:

$$\begin{array}{rcrcrcrcrcl}
\beta_1x_1 & + & c_1x_2 & & & & & & & = & \rho_1 \\
 & & \beta_2x_2 & + & c_2x_3 & & & & & = & \rho_2 \\
 & & & & \beta_3x_3 & + & c_3x_4 & & & = & \rho_3 \\
 & & & & & & & & & \vdots & \\
 & & & & & & a_{n-1}x_{n-2} & + & b_{n-1}x_{n-1} + c_{n-1}x_n & = & r_{n-1} \\
 & & & & & & & & a_nx_{n-1} + b_nx_n & = & r_n
\end{array} \tag{2.24}$$

Procedendo as seguintes mudanças têm-se:

$$\beta_3 = b_3 - \frac{a_3}{b_2}c_2, \qquad \rho_3 = r_3 - \frac{a_3}{b_2}\rho_2 \tag{2.25}$$

Ao observar as transformações anteriores nota-se um padrão. Após $n-1$ passos chega-se ao seguinte conjunto de equação:

$$\begin{array}{rcrcrcrcl}
\beta_1 x_1 & + & c_1 x_2 & & & & & = & \rho_1 \\
 & & \beta_2 x_2 & + & c_2 x_3 & & & = & \rho_2 \\
 & & & & \beta_3 x_3 & + & c_3 x_4 & = & \rho_3 \\
 & & & & & & & \vdots & \\
 & & & & & & \beta_{n-1} x_{n-1} + c_{n-1} x_n & = & \rho_{n-1} \\
 & & & & & & \beta_n x_n & = & \rho_n
\end{array} \tag{2.26}$$

Definindo:

$$\beta_j = b_j - \frac{a_j}{\beta_{j-1}}c_{j-1} \quad e \quad \rho_j = r_j - \frac{a_j}{\rho_{j-1}}c_{j-1}, \quad j = 2, ..., n. \tag{2.27}$$

A partir de agora esse conjunto de equações pode ser resolvido por retroalimentação. Partindo da última equação obtêm-se:

$$x_n = \frac{\rho_n}{\beta_n} \tag{2.28}$$

Substituindo (2.28) na segunda equação do conjunto de equações tem-se:

$$x_{n-1} = \frac{(\rho_{n-1} - c_{n-1}x_n)}{\beta_{n-1}} \tag{2.29}$$

e assim por diante. Assim como as equações anteriores tem o mesmo padrão de solução, pode se obter um resultado geral:

$$x_{n-j} = \frac{(\rho_{n-j} - c_{n-j}x_{n-j+1})}{\beta_{n-j}}, \quad j = 1, ..., n-1 \tag{2.30}$$

que é a solução do problema original (24).

Algoritmo 2.11: Algoritmo do método de Thomas

(a, b, c, x, r)
se $b_1 = 0$ **então**
    Pare a resolução zero na diagonal
senão
    $\beta_1 = b_1$
    $\rho_1 = r_1$
    para $2 \leqslant j \leqslant n$ **faça**
        $\beta_j = \dfrac{b_j - a_j * c_{j-1}}{\beta_{j-1}}$
        $\rho_j = \dfrac{r_j - a_j * \rho_{j-1}}{\beta_{j-1}}$
        se $\beta_j = 0$ **então**
            Pare a resolução zero na diagonal
        fim
    fim
    //agora a retroalimentação
    $x_n = \dfrac{\rho_n}{\beta_n}$
    para $1 \leqslant j \leqslant n-1$ **faça**
        $x_{n-j} = \dfrac{(\rho_{n-j} - c_{n-j} * x_{n-j+1})}{\beta_{n-j}}$
    fim
fim

2.7 Determinação de autovetores através do método da iteração inversa

Considere uma matriz simétrica $A \in \Re^{nxn}$ e adotando um boa aproximação para $\vartheta \in \Re$ para o autovalor $\lambda \in \Re$ da matriz A e uma aproximação qualquer $v^{(0)} \in \Re^n_*$, $\|v^{(0)}\|_2 = 1$, para o autovetor associado $v \in \Re^n_*$, $\|v\|_2 = 1$. Ficando subentendido que $\vartheta \neq \lambda$, onde ϑ não é um autovalor da matriz A de maneira que a matriz $A - \vartheta I$, onde a mesma não é singular. O método da iteração inversa define a seguinte sequencia de vetores $v^{(k)}$, com $k = 0, 1, ...$, da seguinte maneira: dado $v^{(k)} \in \Re^n_*$ procura-se $w^{(k)} \in \Re^n_*$ e a seguir $v^{(k+1)} \in \Re^n_*$ de

$$\begin{array}{rcl} (A - \vartheta I)w^{(k)} & = & v^{(k)}, \\ v^{(k+1)} & = & c_k w^{(k)}, \end{array} \tag{2.31}$$

onde $c_k = \dfrac{1}{\sqrt{w^{(k)t} w^{(k)}}} = \dfrac{1}{\|w^{(k)}\|_2}$, então tem-se $\|v^{(k)}\|_2 = 1, \;\; k = 0, 1, 2, ...$

Teorema 12 *Suponha que $A \in \Re_{sym}^{(nxn)}$. A sequência de vetores $(v^{(k)})$ em $\Re_*^n$ definida no processo da iteração inversa em* (2.31) *converge para o autovetor $v \in \Re_*^n$ normalizado correspondente ao autovalor $\lambda \in \Re$ que esta mais próximo de $\vartheta \in \Re$, desde que λ seja um vetor simples e o vetor $v^{(0)} \in \Re_*^n$ não seja ortogonal ao vetor v.*

A demonstração pode ser vista em (25).

Se a aproximação de ϑ esta dentro da margem de arredondamento de λ_s e os autovalores são esparsos a sequência $(v^{(k)})$ converge rapidamente, normalmente um par de iterações é suficiente.

Na demonstração do teorema (12) conclui-se que $\alpha_s = 0$, ou seja, o vetor inicial $v^{(0)}$ é ortogonal ao autovetor procurado. Porém esse resultado não implica que a iteração (2.31) irá parar; por causa do erro de arredondamento é introduzido um múltiplo do vetor $x^{(s)}$ na expansão de $v^{(0)}$ em termo de $x^{(j)}$ para $j = 1, 2, ..., n$, logo o autovetor procurado será encontrado com um número reduzido de iterações. Esta é um propriedade de grande utilidade para o método, pois na prática não é possível comprovar se $v^{(0)}$ é ou não ortogonal ao vetor v, pois o autovetor v não é conhecido.

Existirá problema quando um autovalor for múltiplo de um outro, ou quando dois autovalores são muitos próximos: quando ocorrer a primeira situação $\dfrac{|\lambda_s - \vartheta|}{|\lambda_j - \vartheta|} = 1$ para $j \neq s$ que é a finalização do teorema (12); na segunda situação $\dfrac{|\lambda_s - \vartheta|}{|\lambda_j - \vartheta|} \approx 1$ para $j \neq s$ a convergência será bastante lenta.

A determinação de $w^{(k)}$ por (2.31) deve ser feita através da solução de um sistema linear cuja matriz é $A - \vartheta I$. De maneira geral a resolução de um sistema pode apresentar algum perigo devido a erros de arredondamento, para resolver esse problema é introduzido um múltiplo do autovetor dominante o que é exatamente o que se necessita.

Existem duas formas de se implementar a iteração inversa uma delas é utilizar a matriz original $A \in \Re^{(nxn)}$ o que leva a implementação de (2.31). A outra forma é trocar a matriz original por uma matriz tridiagonal $T \in \Re^{(nxn)}$ que pode ser obtida por exemplo através do método de Householder, esse processo acelera o cálculo dos autovalores e assim obter o autovetor correspondente de maneira mais rápida.

A utilização da iteração inversa com a matriz original $A \in \Re^{(nxn)}$ requer o uso da fatoração LU na decomposição de A seguida por uma ou mais retrosubstituição o que requer $\dfrac{1}{3}n^3$ multiplicações. Ao passo que o mesmo processo, porém utilizando matriz tridiagonal juntamente com o método de Thomas requer apenas n multiplicações (25).

2.7.1 Perturbação

Às vezes se faz necessário ter uma avaliação de quanto aos autovalores e autovetores de uma matriz são perturbados por mudanças nos elementos. Essas perturbações podem aparecer quando os elementos da matriz são obtidos, por aferições físicas inexatas ou por aproximações de diferenças finitas de equações diferenciais.

Teorema 13 *Seja* $M \in \Re_{sym}^{(nxn)}$*, com autovalores* λ_i *e correspondentes autovetores ortonormais* v_i, $i = 1, 2, ..., n$ *e supondo que* $u \neq 0$ *e* $w \in \Re$ *e* μ *um número real tal que*

$$(M - \mu I)u = w \tag{2.32}$$

Logo pelo menos um autovalor λ_j *de M satisfaz*

$$|\lambda_j - \mu| \leqslant \frac{\|w\|_2}{\|u\|_2}$$

Teorema 14 *(caso simétrico) Suponha que* $A, E \in \Re_{sym}^{(nxn)}$ *e* $B = A - E$ *. Suponha ainda que os autovalores de A são dados por* λ_j *para j= 1, 2, ..., n e* μ *um autovalor de B. Logo pelo menos um autovalor* λ_j *satisfaz*

$$|\lambda_j - \mu| \leqslant \|E\|_2$$

A demostração dos teoremas (13) e (14) pode ser vista em (25).

A combinação dos resultados dos teoremas (13) e (14) mostra que no caso da matrize simétrica A pequenas alterações nos seus elementos podem ocasionar alterações nos autovalores de A, além disso os mesmos fornecem condições de saber até onde os autovalores de A foram perturbados.

A seguir um algoritmo da iteração inversa.

Algoritmo 2.12: Algoritmo da iteração inversa

Entrada: $T \in \Re^{(nxn)}$:matriz tridiagonal simétrica, $\tilde{\lambda}_1, ..., \tilde{\lambda}_m \in \Re$: autovalores aproximados de T

Saída: $q_1, ..., q_m \in \Re^n$:autovetores de T

para $j = 1$ *até* m **faça**
  gerar $v^{(0)_j}$ números aleatórios
  $T - \tilde{\lambda_j}\mathbf{I} = P_j L_j U_j$
  $i = 0$ **repita**
    $i = i + 1$
    normalize $v^{(i-1_j)} = q_j$
    calcule $v_j^{(i)} : P_j L_j U_j v_j^{(i)} = v_j^{(i-1)}$
    se $|\tilde{\lambda}_j - \tilde{\lambda}_{j-1}| \leqslant 10^{-3}\|T\|$ **então**
      reortogonalize $v_j^{(i)}$ versus $q_j, ..., q_{j-1}$
      e gere q_j
    senão
      $j_1 = j$, $q_j = v_j^{(i)}$
    fim
  até *convergência*
  normalize q_j;
fim

3 Computação Paralela

Existe uma continua e grande demanda pelo poder computacional dos sistemas operacionais o que nos dias de hoje é possível de se obter. Algumas áreas requerem grande poder de computação incluindo computação cientifica e problemas de engenharia.Problemas que envolvem essas áreas exigem uma grande quantidade cálculos que se repetem sobre dados amplos e que necessitam de resultados validos.

Os cálculos devem ser executados dentro de um prazo razoável, por exemplo cálculos de engenharia e simulação devem ser executados e armazenados em segundos ou mesmo em minutos se possível. Os sistemas que envolvem situações que necessitam de mais velocidade no seus cálculos tornam-se cada vez mais complexos por esse motivo exige-se cada vez mais tempo para simulá-los.

Existem problemas que possuem tempo especifico para a realização de seus cálculos, a previsão do tempo é um deles, problema este que os se cálculos demorarem a serem executados corre-se o risco da previsão não surtir efeito pois a situação climática pode ter alterado. A previsão climática e a estrutura do DNA são problemas que oferecem grandes desafios, considera-se grande desafio problemas que não podem ser resolvidos utilizando os computadores existentes dentro de um prazo razoável.

Os computadores normais possuem um único processador para executar as tarefas que lhe são imputadas por algum *software*. Uma forma de se ganhar mais velocidade e assim resolver problema em espaço de tempo menor foi a de se ter vários processadores em uma mesma maquina ou seja o uso de multiprocessadores outra alternativa é o usos de diversos computadores que trabalham em um único problema.Em ambos os casos o problema é divido em diversas partes, onde cada uma dessas partes é executada por um processador separado em paralelo. Programas que escrevem para essa forma de se calcular é conhecido como programação paralela. A plataforma de computação paralela pode ser um sistema de computadores especificamente projetado com vários processadores ou diversos computadores interligados de alguma forma. O uso da computação paralela deve prever um acréscimo considerável no desempenho.

A ideia de que p processadores/computador poderia fornecer até p vezes a velocidade de um único processador/computador não importando a velocidade do processador, a expectativa de que o problema seja resolvido em $\frac{1}{p-ésimo}$ do tempo é uma situação ideal o que na prática é dificilmente alcançada. Nem todos os problemas podem ser divididos em partes exatamente independentes havendo a necessidade de interação entre as partes quer seja para transferência de dados ou sincronização. No entanto melhorias consideráveis podem ser alcançadas dependendo do problema e da

quantidade de paralelismo adotado no problema (26).

3.1 Taxonomia de Flynn

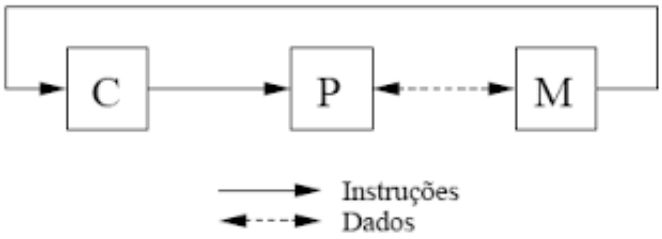
A classificação de arquitetura mais conhecida é de longe a Taxonomia de Flynn, classificação esta que se refere ao fluxo de entrada de dados e o numero de fluxo de instruções das máquinas.Um fluxo de instrução corresponde a uma sequencia de instruções realizadas (num processador) sobre um fluxo de dados os quais as instruções estão relacionadas. Alicerçado na possível unicidade multiplicidade fluxo de dados e instruções as arquiteturas de computadores podem ser divididas em quatro classes :

3.1.1 *Single Instruction Stream/Single Data Stream* -Fluxo único de instruções/-Fluxo único de dados (SISD)

Este modelo refere-se a identificação mais simples, pois o equipamento é considerado sequencial, ou seja executa uma única instrução a cada vez enviado.

Nos computadores e *mainframes* antigos, mesmo com alguns apresentando unidades múltiplas de processamento, não havia qualquer especie de dependência ou relação entre os dados e o fluxo de instrução.

Figura 7 – Arquitetura de máquina SISD



Fonte: (27)

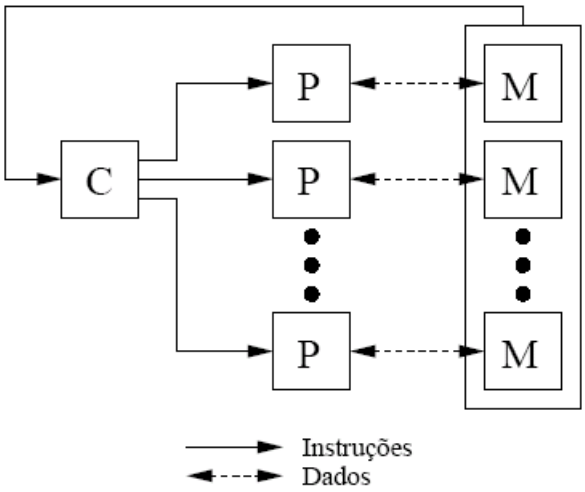
Onde na figura acima:
C: unidade de controle
P: Central de processamento
M: Memória

3.1.2 *Single Instruction Stream/Multiple Data Stream*- Fluxo único de instruções/Fluxo múltiplo de dados (SIMD)

Este modelo consiste em uma única instrução sendo executada sobre múltiplos dados, tem-se nesse processo múltiplos processadores (escravos) sob o controle de

uma única unidade (mestre). Supercomputadores vetoriais fundamenta-se na arquitetura SIMD, onde nos mesmos tem-se a possibilidade de paralelização de instruções melhorando assim o tempo de processamento de dados.Este tipo de máquinas são utilizados no processamento de matrizes e melhoramento de imagens.

Figura 8 – Arquitetura de máquina SIMD

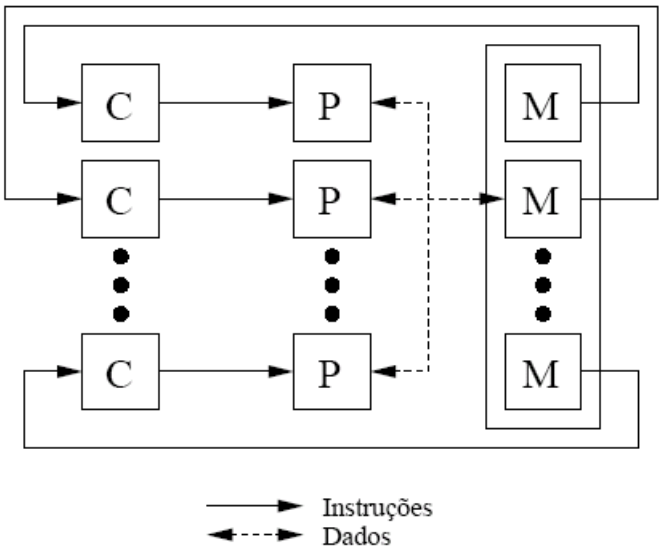


Fonte: (27)

3.1.3 *Multiple Instruction Stream/Single Data Stream* - Fluxo múltiplo de instruções/Fluxo único de dados (MISD)

Neste tipo de arquitetura tem-se múltiplos processadores realizando diferentes instruções em um único conjunto de dados. Usualmente não se encontra máquinas que se enquadrem nesta categoria.

Figura 9 – Arquitetura de máquina MISD

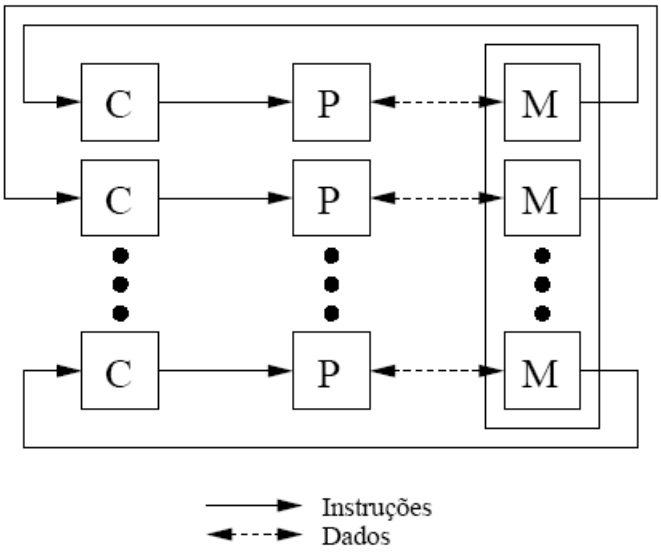


Fonte: (27)

3.1.4 *Multiple Instruction Stream/Multiple Data Stream* - Fluxo múltiplo de instruções/Fluxo múltiplo de dados (MIMD)

Esta arquitetura engloba a maior parte dos processadores paralelos. Arquiteturas estas que possuem como característica a execução de vários fluxos de instrução ao mesmo tempo. Essa aptidão deve-se ao fato de que as tarefas realizadas nesse modelo são executadas por vários processadores que trabalham de forma coopertiva ou concorrentes. Nesta categoria estão grande parte dos computadores paralelos.

Figura 10 – Arquitetura de máquina MIND



Fonte: (27)

Esta categoria divide-se em duas outras:

i) Memória compartilhada: Espaço do endereçamento de memória compartilhado.

ii) Memória local ou individual: Espaço de endereçamento próprio.

3.2 Modelos de Paralelismo

Encontra-se vários tipos de paralelismos em níveis diferentes de hierarquia. Estes níveis podem ser classificados de quatro formas (28).

i) Nível de *bits*: Esta espécie de paralelismo deu-se por intermédio do aumento da palavra utilizada pelos processadores sofreu sucessivos aumentos de 4 a 64 *bits* (28).

ii) Nível de instrução: Basicamente um *software* de computador é um conjunto de fluxo de instrução que é executado por um processador, estas instruções podem

ser reordenadas e combinadas em conjuntos que logo serão executados em paralelo sem haver mudanças no *software*, este processo é chamado de instrução (28).

iii) Nível de dado: Tem como base os laços de repetição, onde a distribuição dos dados é feita em diferentes nós computacionais para que sejam processados em paralelo (29).

iv) Nível de tarefa: Sua característica é a realização de cálculos em um mesmo conjunto de dados ou em conjuntos de dados diferentes, através de um *software* paralelo. O paralelismo de tarefa normalmente não possui a possibilidade de escalonar [1] o problema de acordo com seu tamanho (29).

v) Nível de *Thread*: Este tipo de paralelismo abrange a execução em paralelo de um conjunto de tarefas independente uma das outras, esse processo recebe a denominação de *thread* (30). Para que as *Thread* sejam eficientes as mesmas devem ser ao máximo independentes umas das outras, isto é não deve existir pontos em comum entre elas, pois isso exigiria sincronia entre as *threads* o que pode acarretar *race condition* (quando dois ou mais processos lê ou escreve em um dado compartilhado) (30).

3.3 APIs de Programação Paralela

As APIs (*Application Programming Interface*) em português Interface de Programação de Aplicativos para computação paralela, algumas dessas APIs e suas principais características são apresentadas a seguir.

3.3.1 OpenMP

OpenMP *(Open Multi Processing)*: É uma API de programação de *software* em memória compartilhada, onde a mesma é baseada no esforço anterior o que facilita o compartilhamento de memória na programação paralela, a OpenMP não foi oficialmente sancionada, mas sim surgiu de um acordo entre os membros da ARB *(OpenMP Architecture Review Board)* que era um grupo de fornecedores que tinham o mesmo interesse em uma API portátil fácil de usar e com tratamento eficiente para programação paralela.OpenMp não é uma nova linguagem, mas sim uma notação que pode ser utilizada em um programa sequencial em Fortran, C, C++, ou pra descrever a maneira como o exercício será compartilhado entre os itens a serem executados nos diferentes processadores e como será o acesso aos dados compartilhados (31).

[1] Escalonar vem de escalabilidade ou seja a aptidão para manusear uma parte crescente de trabalho de maneira igual, ou esta organizado para crescer

3.3.2 MPI

MPI *(Message Passing Interface)* é usado na comunicação de dados entre processadores sem a necessidade da memória global. O fundamento para essa estrutura é que cada processador tem uma memória própria que é utilizada na comunicação entre os processadores existes, fazendo uso de mensagens. A não utilização de uma grande memória global e bem como da sincronização, dá a MPI uma grande vantagem sobre o sistema de memória compartilhada. Durante a execução de um programa o mesmo é dividido ao mesmo tempo em diversos processos que são executados em processadores distintos. Caso o número de processos seja maior que a quantidade de processadores, mais de um processo será executado em um mesmo processador através do tempo compartilhado. Processos sendo executado em um mesmo processador utiliza os canais internos para trocarem mensagens entre si, processos sendo rodados em diferentes processadores utiliza canais externos para trocarem mensagens.Dados trocados entre os processadores não são compartilhados, por isso o uso do esquema de troca de mensagem, onde essa troca é efetuada através de copias. Um beneficio dessa maneira de troca é a eliminação por exemplo de farol para efetuar a sincronização o que resulta na melhoria da performance, além dessa vantagem o sistema por passagem de mensagem oferta elasticidade na acomodação de vários processadores, além de ser facilmente escalável (32).

3.3.3 OpenCL

OpenCL *(Open Computing Language)* é um arcabouço da industria de computadores para programação com arranjo entre CPU *(Central Processing Unit)* e GPU *(Graphics Processing Unit)* e demais processadores. Os sistemas heterogêneos vieram a ser uma classe considerável de plataformas e o OpenCL é o primeiro padrão da industria a lidar diretamente com suas necessidades. Com o OpenCL pode-se escrever um único *software* que pode ser executado em grande quantidade de plataformas tais como: celulares, *notebooks* e nós de supercomputadores (33).

O padrão OpenCL pode ser definido em quatro partes, ou modelos que podem ser especificadas como a seguir (34):

i) Modelo Plataforma: Modelo este onde tem-se um processador de coordenação *(host)* com capacidade para coordenar um ou mais processadores que executam o código OpenCL. Ele delimita um padrão de *hardware* abstrato utilizado por programadores para escrever as funções OpenCL.

ii) Modelo de Execução: Delimita como a plataforma OpenCL esta configurada no *host* e de que maneira os *kernels* são executados na maquina. Isso inclui a formação de um cenário OpenCL no *host* ofertando aparatos que possibilitem a

interação hospedeiro-dispositivo e assim definindo um modelo de concorrência que pode ser utilizado para a execução de *kernels* em dispositivos.

iii) Modelo de Memória: Esclarece a categoria de memória abstrata utilizada pelo *kernel* independente da arquitetura da memória real implícita. Este modelo é bastante semelhante hierarquicamente ao modelo de memória da das GPUS dos dias de hoje, porém esse motivo não acarreta limitações por parte de outros aceleradores.

iv) Modelo de Programação: Define como o modelo de concorrência é apresentado no *hardware*

3.3.4 CUDA ™

Nos últimos anos u uma grande empresa fabricante de *hardware* especialmente para jogos vem revolucionando o mercado computacional essa empresa é a chamada NVIDIA. Com a inserção da linguagem de programação CUDA ™ *(Compute Unified Device Architecture)*, doravante denotada apenas por CUDA, a partir desse momento co-processadores gráficos muito poderosos poderiam ser usados por programadores C para a realização de trabalhos computacionais onerosos. A partir de quando esses dispositivos passaram a ser incorporados desde computadores de usuários domésticos a supercomputadores, tudo mudou.

Uma substancial mudança ocorrida na industria de *software* de computadores foi o deslocamento do eixo de programação do sequencial para o paralelo, nesta transformação a CUDA tem representado avanços significativos. A GPU *(Graphics Processing Unit)* tem sua concepção pautada na execução de operações gráficas de alta velocidades que por sua natureza são paralelas.
Ao utilizar a CUDA não requer conhecimento gráfico algum não há também a necessidade de ser um programador de jogos. CUDA faz que se olhe para a GPU como um outro dispositivo programável qualquer (35).

CUDA oferece várias APIs para programação onde os níveis das mesmas são decrescentes (36):

i) API de dados paralelos utilizado em C++

ii) API de tempo de execução que pode ser utilizada em C e em C++

iii) API do controlador que também pode ser utilizado em C e em C++

No desenvolvimento de aplicativos, CUDA pode ser utilizado uma única APi ou uma combinação delas, bem como pode ser empregado diversos tipos de linguagens de alto nível, Python, Java, Fortran, dentre outras.

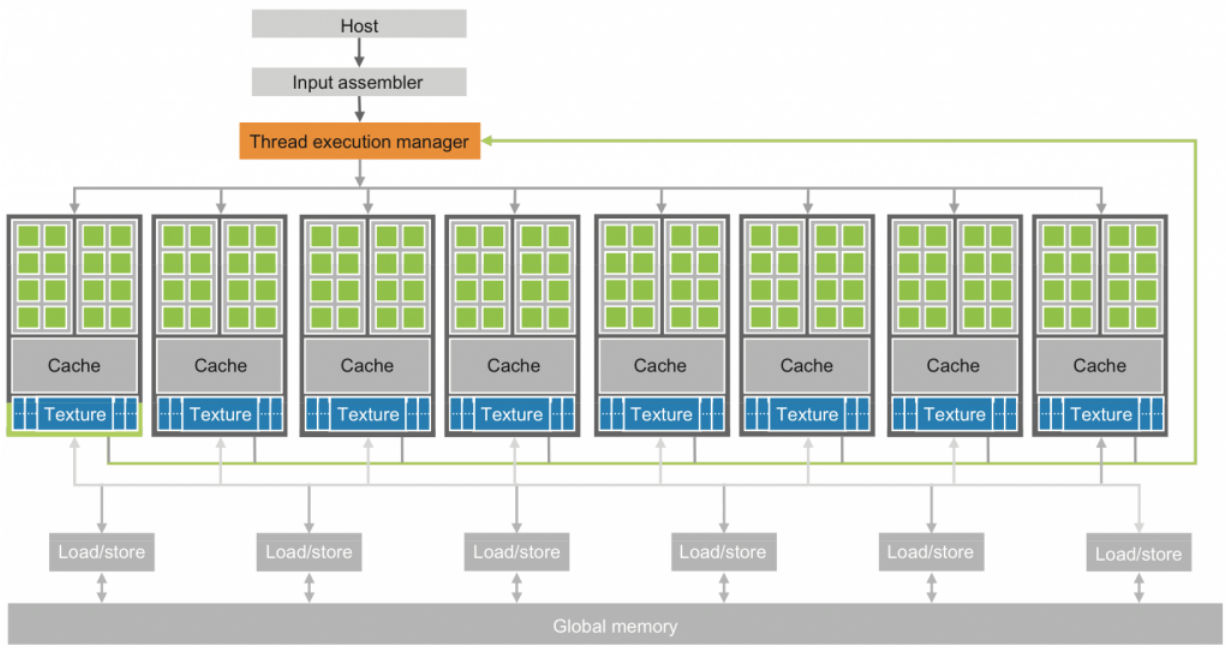
Qual API usar, essa decisão esta reacionado ao grau de controle que o programador deseja exercer sobre a GPU. As linguagens de alto nível, podem auxiliar o programador e em algumas situações decidir por ele, de maneira geral a utilização de uma linguagem de alto nível tem apresentado disposição para em proporcionar um alto desenvolvimento computacional, além de dar grande generalidade e conveniência. Ao fazer uso de uma linguagem de alto nível o código fonte pode ser desenvolvido com mais rapidez e facilidade e o mesmo pode ser de mais fácil leitura e sustentável. Outra característica do desenvolvimento com esse tipo de linguagem é a capacidade de adaptação do código a um novo *hardware*. Uma desvantagem no uso de linguagem de alto nível e que ela pode apartar alguns recursos do *hardware* desenvolvedor e deixar amostra somente um subconjunto de recursos desse *hardware*.

O uso de um linguagem de alto nível como C++ por exemplo em algumas situações pode se tornar muito pesado ou com um grande número de detalhes. Programadores da área cientifica de maneira peculiar podem sentir que as estruturas de repetição de sintaxe simples pode se perder no C++.As vezes ao iniciar a programação pode-se escolher uma linguagem de mais alto nível e depois com o decorrer do tempo optar por uma de nível mais baixo, quando se visualiza a possibilidade de um esforço adicional onde irá proporcionar maior desempenho ou mesmo utilizar uma capacidade de nível mais baixo necessária para dar apoio a escolha feita. O tempo de execução CUDA, foi projetado especialmente para que os desenvolvedores obtenham acesso a todos os recursos presentes na GPGPU *(General Purpose Graphics Processing Unit)*, com a adição de algumas sintaxes simples, elegantes e poderosas na linguem C, o resultado do tempo de execução CUDA, pode ser um código limpo e de fácil leitura, além disso o mesmo pode ser extremamente eficiente. Um outro fator importante da utilização de um controlador de interface de nível mais baixo é o controle das filas de espera e transferência de dados que se torna mais preciso.

Os desenvolvedores CUDA não estão presos a uma só linguagem, o que ocorria até versão 3.2 da CUDA no ano de 2010 quando houve a liberação, as versões mais novas permitem aos desenvolvedores utilizarem uma das três APis nos seus aplicativos, dessa forma o desenvolvedor pode iniciar seu aplicativo em uma APi de alto nível e depois passar a usar uma outra de mais baixo nível a fim de aproveitar uma funcionalidade do tempo de execução ou do *hardware* (36).

3.3.4.1 Arquitetura de uma GPU CUDA

Figura 11 – Arquitetura de uma GPU CUDA



Fonte: (37)

A Figura 11 representa como é a arquitetura de uma GPU CUDA, bem como sua organização onde seus multiprocessadores de transmissão altamente segmentados (SMs) são dispostos em forma de matriz. De acordo com a Figura 11 dois SMs juntos configuram um bloco de construção, porém o número desses blocos pode sofrer variações de uma geração GPUs CUDA para outra. Outra característica presente na Figura 11 a cerca dos SMs é que cada possuem um número de processadores de fluxo (SPs) que dividem a lógica do controle acionário e a *cache* de instrução.

Nos dias atuais em cada placa gráfica estão presentes 4 *gigabyte* de DRAM *(Dynamic Random Access Memory)* GDDR *(Graphics Double Data Rate)* denominada memória global. A diferença entre as DRAMs GDDRs e a CPUs *(Central Processing Unit)* da placa mãe, é porque as mesmas são utilizadas como memória *frame buffer* que é usada nos gráficos. Para propósitos que envolvam atividades gráficas com imagem de vídeo e textura tridimensional a memória as guarda para renderização em 3D, porém para a computação elas atuam como uma grande largura de banda fora do *chip*, se bem que com uma latência [2] maior do que a memória especifica do sistema. Ao desenvolver aplicações de cunho massivamente paralela uma memória com uma largura de banda mais alta gera também uma latência maior.

Com o lançamento da Geforce 8 ou G80 que é a oitava geração de placa Geforce da Nvdia houve a introdução da arquitetura CUDA com uma largura de 86,4 GB/s de

[2] indica a quantidade de pulsos de *clock* que a memória leva para retornar um dado solicitado

memória além de uma banda de 8 GB/s para comunicação com a CPU.
Um *software* CUDA pode passar dados a partir da memória do sistema a uma velocidade de 4 GB/s e ao mesmo tempo transferir dado de volta para a memória do sistema a uma velocidade de 4GB/s , totalizando 8 GB/s de velocidade de transferência.A largura da banda de comunicação tem tamanho bem menor do que a largura de banda da memória a principio isso pode parecer uma limitação a largura de banda do PCI Express ™ é semelhante a largura de banda do barramento da CPU, dessa forma o que parecia ser uma limitação no principio necessariamente não o é.

O tamanho da memória da GPU cresce a cada geração os *softwares* por sua vez tem a tendencia de manter seus dados na memória global e usara o PCI Express, somente quando houver necessidade de comunicação com a memória da CPU e quando houver obrigatoriedade de acessar uma biblioteca que esta acessível somente na CPU. A largura da banda de comunicação devera crescer futuramente assim como cresce a largura de banda da memória da CPU (37).

3.3.4.2 Computação utilizando a placa gráfica

Durante o desenvolvimento da arquitetura da GPU Tesla ™, notou-se que a mesma tinha um grande potencial se ela pudesse ser vista e utilizada como um processador pelos desenvolvedores. O tratamento adotado pela NVIDIA foi de uma abordagem onde os desenvolvedores anunciam abertamente a dimensão dos dados paralelos em sua carga de trabalho. Na geração gráfica DirectX ™ 10 a NVIDIA começou a trabalhar com uma grande eficiência no que tange a ponto flutuante bem como com processadores inteiros o que que abre a possibilidade de se realizar várias cargas de trabalho o que de certa forma suportaria a *pipeline* gráfica.

Durante o desenvolvimento do da GPU Tesla seus projetistas avançaram mais um pouco. Processadores *shader* agora podem ser programados, com memória e cache de instrução bem como um controle sequencial de instrução. Como o ganho de performance através da adição de *hardware* era de esperar um aumento nos preços, porém isso não aconteceu, pois os diversos processadores de *shader* existentes compartilham sua cache lógica sequencial de instrução.Esse tipo de desenho tem um bom desempenho para aplicações gráficas, porque o programa *shader* são aplicados em um número muito grande de *pixels* ou vértices. A NVIDIA acrescentou o beneficio de leitura e escrita na memória e a faculdade de endereçamento do *byte* aleatório para utilização da linguagem de programação C compilada.

Para aplicações que não sejam gráficas a GPU Tesla ™ inseriu um padrão de programação paralela mais geral de uma categoria de seções paralelas com sincronização de barreira e as operações atômicas de enviar e fiscalizar as atividades de computação altamente paralela.

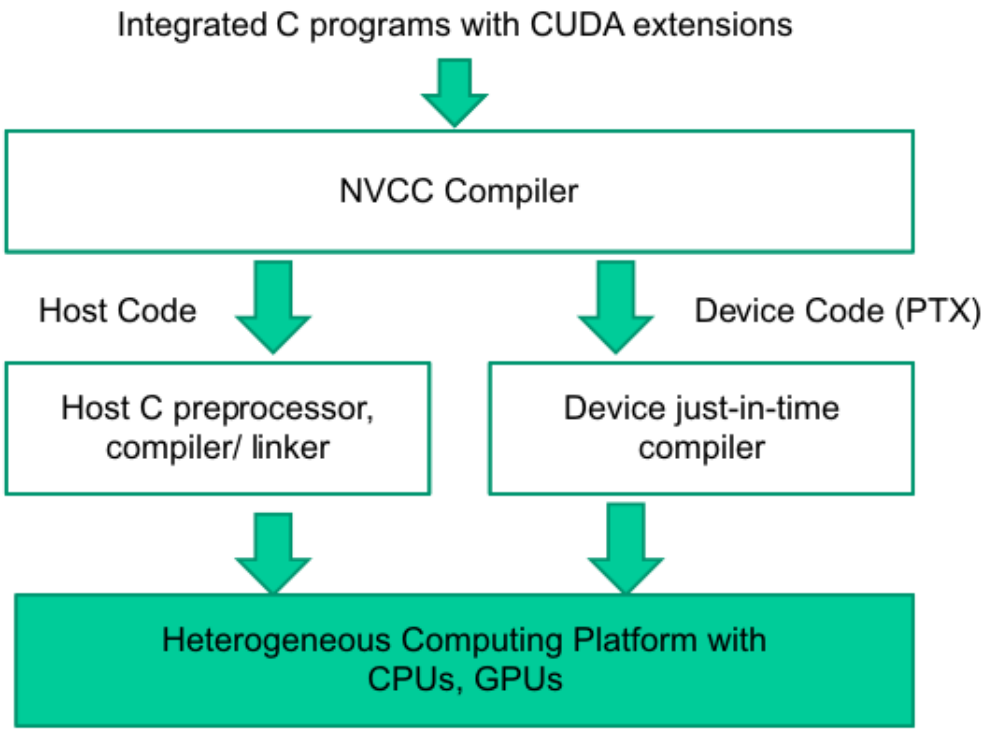
A NVIDIA elaborou o CUDA C/C++ compilador, bibliotecas de tempo de execução e *softwares* o que possibilitou que os desenvolvedores pudessem acessar de maneira simples o novo modelo de programação paralela (37).

3.3.4.3 Organização de um *software* CUDA

A organização de um *software* CUDA representa a simultaneidade entre um hospedeiro (*host*) e uma ou mais GPUs existentes no computador, onde os programas são executados em etapas e as mesmas podem ocorrer tanto no *host* como na GPU. Os *software* CUDA podem ser compostos por uma combinação entre o *host* e a GPU. A principio todo código em C é também um código CUDA que roda no *host*, onde o paralelismo é encontrado em pequenas quantidades ou inexiste ao passo que a parte do código que roda na GPU tem por sua vez uma gama grande de paralelismo, onde os mesmos são diferenciados por conter palavras chaves que pertencem tipicamente ao CUDA.

Quando se adiciona ao código funções da GPU e palavras chaves da CUDA ela não é mais compilado em um compilador C tradicional, esse código necessita de um compilador que capte e identifique as adições efetuadas no código. A NVIDIA criou um compilador para fazer essa função de leitura e compreensão de códigos com características do CUDA o mesmo tem o nome de NVCC NVIDIA C *Compiler*

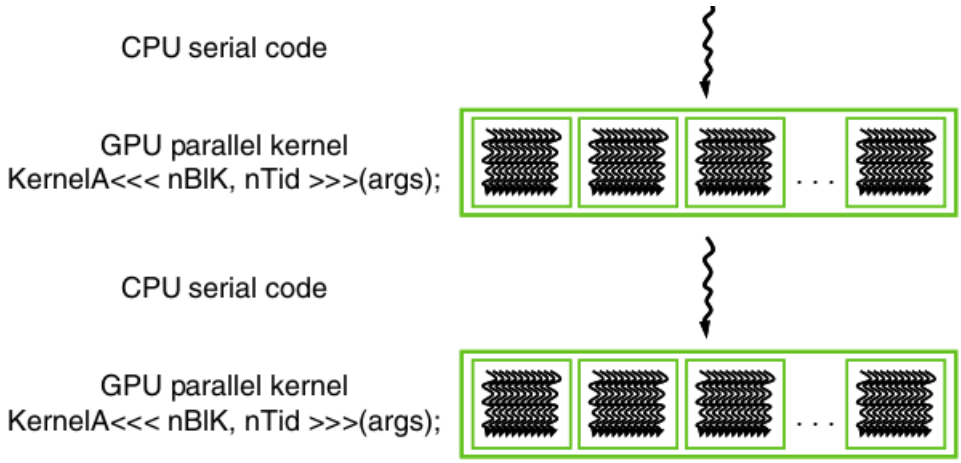
Figura 12 – Visão geral do processo de compilação de um programa CUDA



Fonte: (37)

A Figura 12 exibe como é processado um programa CUDA, onde as palavras chaves da CUDA é utilizada para distinguir partes do código que ira rodar na GPU da parte que ira rodar no *host*. A parte do código que irá rodar no *host* é escrito e compilado com um compilar padrão de C na CPU, enquanto que a parte que roda na GPU é escrita utilizando ANSI C estendido, onde palavras chaves são usadas para marcar as funções com dados paralelos, que recebem o nome de *kernel*. Em situações onde não se tem a presença de um placa gráfica CUDA, pode-se ainda usar um recurso de emulação utilizando o pacote de desenvolvimento de *software* (SDK) CUDA ou uma ferramental de nome MCUDA.

Figura 13 – Execução de um *software* CUDA



Fonte: (37)

A execução de um *software* CUDA é demostrado na Figura 13, onde a execução tem inicio no *rost*. Ao ser chamada uma função *kernel* a mesma executa um número grande de *threads* no dispositivo. Todas *threads* formados pela execução de um *kernel* recebem o nome de grade. A Figura 13 demonstra a execução de duas grades de *threads*. Ao terminar realização de um item de *kernel* a grade correspondente tem fim é a execução tem continuidade no *rost* até que outro *kernel* tenha inicio. A Figura 13 ilustra um exemplo onde a execução da CPU e GPU não se sobrepõe.

Uma *threads* é um aspecto simplificado de como um processador executa um *software* nos computadores atuais. Uma *threads* é composta pelo código do *software* em um ponto especifico que esta sendo executado, os valores da variáveis e as estruturas de dados. Ao ser executada uma *threads* é realizada sequencialmente de acordo com o interesse do desenvolvedor.

Threads vem sendo usadas na programação sequencial da CPU há muitos anos. Ao desenvolver um programa se seu desenvolvedor tiver o intuito de utilizar programação paralela em seu programa o mesmo terá que fazer uso de várias *threads* e diversas bibliotecas e ao mesmo tempo gerenciá-las e ainda utlizar linguagens especiais.

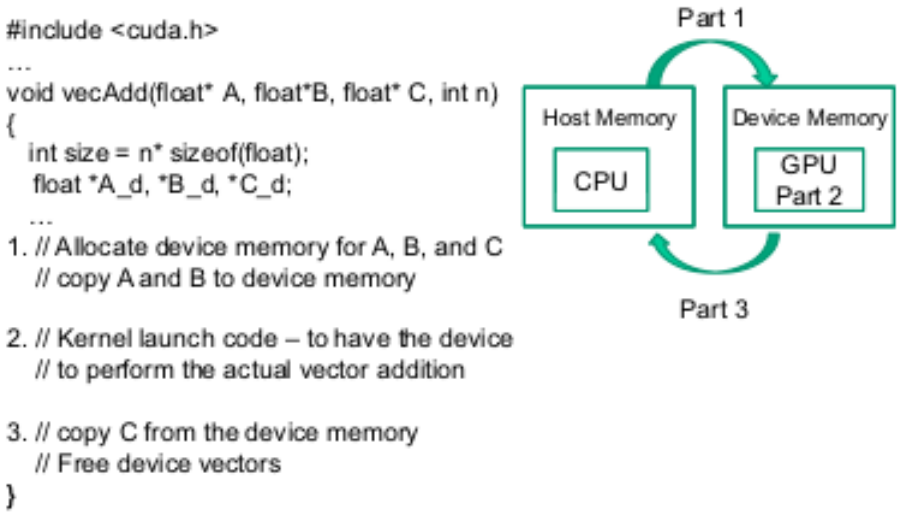
Em CUDA a execução das *threads* é de forma sequencial, um *software* CUDA tem sua execução em paralelo quando há inicio de funções do *kernel*, o que leva a criação de várias *threads* que fazem o processamento das diferentes partes dos dados paralelos.

O disparo de um *kernel* geralmente produz um grande número de *threads* que exploram o paralelismo de dados (37).

Em CUDA a memória do *host* e das *devices* estão em locais diferentes, em virtude disso as *devices* são *hardware* que possuem memorias próprias de acesso aleatório. Ao executar um *kernel* em uma *device* o desenvolvedor precisará alocar memoria global na *devices* e transladar dados da memória do *host* para a memória do *device* alocada.

3.3.4.4 Dispositivo de memória global e transferência de dados

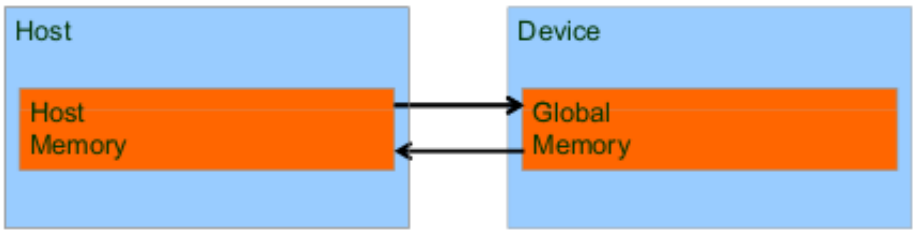
Figura 14 – Transferência de dados entre dispositivos



Fonte: (37)

Esta operação de transferência esta representada na Figura 14 e corresponde a a parte 1 da mesma. Da mesma forma o desenvolvedor depois da execução no *device* precisa transferir os dados para a memória do *host* e em seguida liberar a memoria do *device*, pois a mesma já não é mais necessária. Esta operação corresponde a parte 3 da Figura 14, a CUDA oferece um sistema que execução (APi) que realiza essa operação pelo desenvolvedor.

Figura 15 – Transferência de dados entre dispositivos



Fonte: (37)

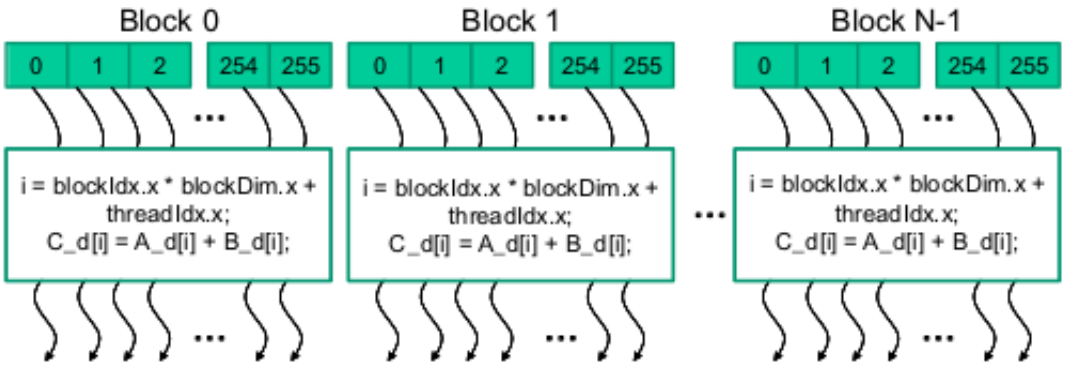
A Figura 15 representa o modelo de memoria do *host* e do *device* CUDA , isso serve para que os desenvolvedores pensem sobre alocação de memoria no *device* e a transferência de dados entre o *host* e o *device*.

A CUDA oferece um mecanismo de gerência de dados na memoria do *device*, através do sistema de execução.

3.3.4.5 Funções do *kernel* e *threading*

Em CUDA uma função do *kernel* caracteriza o código a ser utilizado por todas as *threads* durante um estágio paralelo. Como todas as *threads* rodam o mesmo *script*, a linguagem CUDA é um ocorrência da forma de programar conhecida como SPMD *-(Single-Program, Multiple-Data)*, uma forma de programação comum para computação massivamente paralela, esta arquitetura não é a mesma SIMD *(single instruction, multiple data)* da Taxonimia de Flynn, pois enquanto na arquitetura SIMD todos os processadores efetuam a instruções iguais a qualquer instante na arquitetura SPMD os processadores paralelos realizam o mesmo programa em lugares diferentes dos dados e esse processamento não precisa ser realizado com a mesma orientação e nem ao mesmo tempo. Quando um código *runtime* dispara um *kernel* o sistema de execução CUDA produz uma grade de *threads*, onde os mesmos tem um organização hierárquica de dois níveis. As grades possuem uma organização em forma de uma matriz bloco de *arrays*.

Figura 16 – *Threads* em uma grade executam o mesmo *script*



Fonte: (37)

Todos os blocos de uma grade possuem tamanho idêntico, cada um deles pode conter até 1024 *threads* a Figura 16 apresenta um exemplo onde cada um dos blocos é composto por por 256 *threads*. O número de *threads* em cada bloco de *thread* é enumerado pelo código do *host* quando um *kernek* é disparado. O mesmo *kernek* pode ser inicializado com diferentes números de *threads* em lugares distintos do código do *host*.

Para uma dada grade de *threads* o numero de *threads* que um bloco contém é dado pela variável *blockDim*. Na figura 16 a variável *blockDim.x* tem valor 256, usualmente o tamanho de um bloco de *threads* é múltiplo de 32, por questões de eficiência do *hardware*. Em um bloco cada *threads* tem um valor *threadIdx* exclusivo, essa característica permite que cada *thread* agregue seus valores *threadIdx* e *blockIdx* com a finalidade de gerar um índice global único, onde esse índice é utilizado pela própria *thread* e pela grade.

Ao disparar um *kernel* com um grande número de blocos, tem-se a possibilidade de realizar cálculo com vetores maiores. Quando disparado um *kernel* com n ou mais *thread* consegue-se o processamento de n vetores.

Figura 17 – Função de *kernel* para soma de dois vetores

```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition
__global__
void vecAddKernel(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C[i] = A[i] + B[i];
}
```

Fonte: (37)

A Figura 17 apresenta uma função de *kernel* que realiza a operação de adição entre dois vetores, a sintaxe utilizada é C ANSI *(American National Standards Institute)* com alguma extensões diferentes. A principio tem-se uma palavra chave CUDA que é __global__ que esta presente, logo após a declaração *vecAddKernel* () é uma palavra chave que mostra que a mesma é um núcleo essa função pode ser solicitada por uma aplicação do hospedeiro, para criar uma grade de *threads*.

Figura 18 – Palavras-chaves para declaração de função em CUDA C

	Executed on the:	Only callable from the:
`__device__ float DeviceFunc()`	device	device
`__global__ void KernelFunc()`	device	host
`__host__ float HostFunc()`	host	host

Fonte: (37)

Usualmente se expande CUDA para a linguagem C com o uso de três palavras-chaves que funcionam como habilitador na declaração da função.

A Figura 18 destaca as três palavras chaves que são usualmente empregadas em CUDA C. A palavra chave __global__ , salienta que a função declarada é uma função que pertence a um *kernel* CUDA. A função __global__ tem sua realização no *device* e sua chamada é feira pelo *host*, com a finalidade de gerar uma grade de *threads* no próprio *device*. A palavara-chave __device__ designa que a função declarada é uma função que pertence a *device* CUDA. A função *device* é efetivada em uma *device* CUDA e sua chamada só pode ser realizada se a mesma for proveniente de uma aplicação *kernel*, ou ainda de outra função *device*. A palavra-chave __host__ que a função que esta sendo declarada é uma função do *host* isso significa que a mesma é uma função C habitual que é executada no próprio *host* e sua chamada poderá ser realizada somente por uma função do *host*. Habitualmente as funções presentes em um código CUDA pertencem ao *host* se não contiverem uma palavra-chave CUDA em sua declaração, isso tem coerência, pois diversas aplicações CUDA são provenientes de ambientes que são executados em CPUs. O desenvolvedor estaria adicionando funções do *kernel* e do *device* na realização do processo de portabilidade. As funções primitivas mantém-se como funções do *host*. Dispondo todas as funções primitivas como funções do *host* o desenvolvedor tera seu trabalho reduzido, pois o mesmo não terá a obrigação de modificar as declarações das funções primitivas.

Existe a possibilidade de se utilizar simultaneamente __host__ e __device__ em uma declaração de função, ao se utilizar dessa prerrogativa o sistema de compilação é

notificado que deverá produzir duas versões da mesma função, onde uma deverá ser efetuada no *host* e sua chamada será realizada somente por uma função do *host*, ao passo que a outra versão é executada no *device* e sua chamada é realizada somente por uma função do *device* ou uma função pertencente ao *kernel*. Esse procedimento dá base para que o código seja recopilado para uso em um outro *device*.

A Figura 16 apresenta uma outra extensão notável em ANSI C que é *threadIdx.x*,juntamente com as palavras-chaves *blockIdx.x* e *blockDim.x* que são os índices da *threads*, ainda de acordo com esquema apresentado na Figura 16 todo o conjunto de segmentos executam o mesmo *software* do *kernel*. Faz-se necessário a existência de um procedimento que as diferencie entre si e a partir dai direcione cada uma para a parte do código, para o qual foram organizadas para agir. Essas palavras-chaves são variáveis que identificam funções preestabelecidas que permite que a *threads* tenha acesso aos registros de *hardware* em modo *runtime* o que oferece as coordenadas que identificam uma *threads* em particular.

Threads distintas possuem valores também distintos para as variáveis *threadIdx.x, blockIdx.x, e blockDim.x*, doravante uma *threads* será denotada por $threads_{threadIdx.x,threadIdx.x}$, onde o .x poderá ser um y e z.

As *threads* CUDA possuem uma organização de forma hiparquia em dois neveis a saber: uma grade que é formada por um ou mais blocos, ao passo que os blocos são formados por uma ou mais *threads*. As *threads* de um mesmo bloco partilham o mesmo índice de bloco que poder ser acessado através da variável *blockIdx.x* em um *kernel*, cada *thread* por sua vez é acessado por intermédio da variável *threadIdx* em um *kernel*. Para um desenvolvedor CUDA as variáveis *blockIdx e threadIdx* aparecem embutidas na inicialização das variáveis, onde as mesmas podem ser acessadas dentro de uma função *kernel*. No momento em que uma *thread* executa uma função pertencente ao *kernel* as indicações ás variáveis *blockIdx e threadIdx* devolvem as coordenadas da *thread*.

Os critérios de organização da realização do disparo de um *kernel* delimitam as dimensões de uma grade bem como as dimensões de cada um dos blocos, onde essas dimensões estão acessíveis como variáveis pré-definidas internamente *gridDim e blockDim* em aplicações do *kernel*.

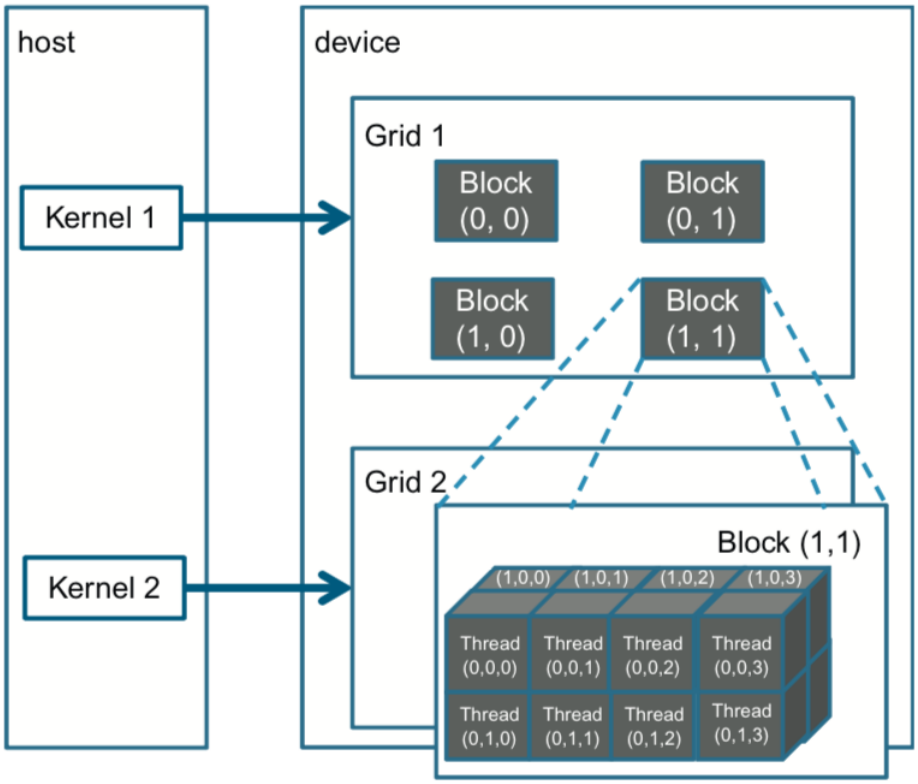
Usualmente uma grade é constituída por uma matriz em três dimensões de blocos de *threads*. Cabe ao desenvolvedor decidir quantas dimensões irá usar, onde as dimensões não usadas são definidas para 1. A sistematização correta de uma grade é definida pelos critérios de conformação e execução durante o disparo de um *kernel*. O primeiro critério de conformação e execução mostra as dimensões da grade de acordo com o número de blocos. O segundo critério mostra as dimensões de cada bloco de acordo com o número de *threads*. Esses critérios são de categoria DIM 3, que é um *struct* em C (registro que é um pacote de variáveis de diversos tipos), que possuem

tês divisões, onde as mesmas são números inteiros sem sinal, X,Y,Z, que são as três dimensões.

Em uma grade e bloco com uma ou duas dimensões, os campos de dimensões não utilizado podem ser definidos com valor 1, aumentando assim a compreensão. Por praticidade em CUDA C existe um atalho para o disparo de um *kernel* que possuem grades e blocos de uma dimensão, onde no lugar de DIM3 pode -se fazer uso de expressões aritmética para tipificar as configurações de grades e blocos de uma dimensão, ao realizar esse procedimento o compilador CUDA C transforma as expressões aritméticas na dimensão X e assume que as dimensões Y e Z são iguais a 1.

A grade pode ser maior que seus blocos ou o contrário também pode ocorrer.

Figura 19 – Organização multidimensional de uma grade CUDA



Fonte: (37)

A Figura 19 exibe um modelo de uma grade de duas dimensões. Cada um dos blocos da Figura 19 tem marcação $(blockIdx.y, blockIdx.x)$. Ao observar a figura nota-se que o bloco (1,0) possui $blockIdx.y = 1$ e $blockIdx.x = 0$. As etiquetas estão organizadas de forma que a maior dimensão vem em primeiro lugar, ou seja, em ordem crescente, nos parâmetros de configuração essa ordem ocorre de maneira inversa, isto é, em ordem decrescente. Isso inverteu a ordem de marcação das *threads* o que facilita a ilustração do mapeamento das mesmas em índices coordenados de dados de acesso aos *arrays* multidimensionais.

Cada *threadId* divide-se em outros três campos a saber:a coordenada x $threadId.x$, a coordenada y, $threadIdx.y$ e a coordenada z, $threadIdx.z$. A Figura 19, mostra a organização de tópicos em um bloco, neste exemplo cada bloco esta sistematizado em 4x2x2 *arrays* de *threads*. Como todos os blocos de uma *thread*, tem a mesma dimensão, logo na Figura 19, tem a representação de apenas um deles, na mesma figura esta presente a expansão do bloco (1, 1) para que se possa visualizar suas 16 *threads*. Para (1, 0,2) a *thread* tem-se $threadIdx.z = 1$, $threadIdx.y = 0$ e $threadIdx.x = 2$. A Figura 19, ilustra 4 blocos de 16 *threads* cada um, totalizando 64 *threads* na grade. A Figura 19 utiliza um números pequenos para facilitar a compreensão, normalmente CUDA contém milhares de milhões de *threads*.

4 Resultados e conclusões

4.1 Resultados

Os resultados foram obtidos mediante a utilização de uma máquina que possui a seguinte configuração:

i) Dois processadores Intel(R) Xeon(R) CPU E5645 2.40GHz, com 12 núcleos cada

ii) 96 GB ram

iii) Quatro placas gráficas NVIDIA Corporation GF110GL [Tesla C2075], com 448 CUDA *cores* cada uma das placas.

No cenário que envolve o cálculo de autovetores e autovalores pode-se encontrar diversas bibliotecas numéricas que apresentam a implementação de vários métodos nas mais variadas linguagens de programação. A biblioteca utilizada no desenvolvimento dessa trabalho foi a LAPACK *(linear álgebra package)*.

As rotinas selecionadas para serem rodadas na CPU e comparadas com a rotina implementada em CUDA foram as seguintes:

i) sstemr (precisão simples) e dstemr (precisão dupla) rotina esta que calcula os autovalores e autovetores de uma matriz tridiagonal simétrica utilizando o método QR. Através dessa rotina pode-se calcular o espectro inteiramente ou parcialmente através da delimitação de um intervalo.

ii) sstevx (precisão simples) e dstevx (precisão dupla) essa rotina obtêm os autovalores através do método da bisseção e os autovetores são calculados pala iteração inversa, onde a matriz utilizada é uma matriz tridiagonal simétrica real.

A Tabela 1 apresenta os tempos de execução das rotinas em precisão simples, onde sstemr e sstevx foram implementadas na linguagem C e a rotina CUDA como o próprio nome sugere foi implementada en CUDA. Quanto ao local de execução as rotinas sstemr e sstevx foram executadas na CPU, ao passo que a rotina CUDA foi executa na GPU exclusivamente. Onde a ordem da matriz é dado por (N*1024), implementações foram realizadas a cada 3, ou seja N= 1,3,...,15 ($1 \leqslant N \leqslant 15$). Esse limitação é motivada pela máquina onde as rotinas foram rodadas, pois como as matrizes são (nxn) na GPU só se executa em *double* até 15*1024 e por uma questão de tempo todas foram implementadas até esse limite em *float* poderia ir até 22*1024.

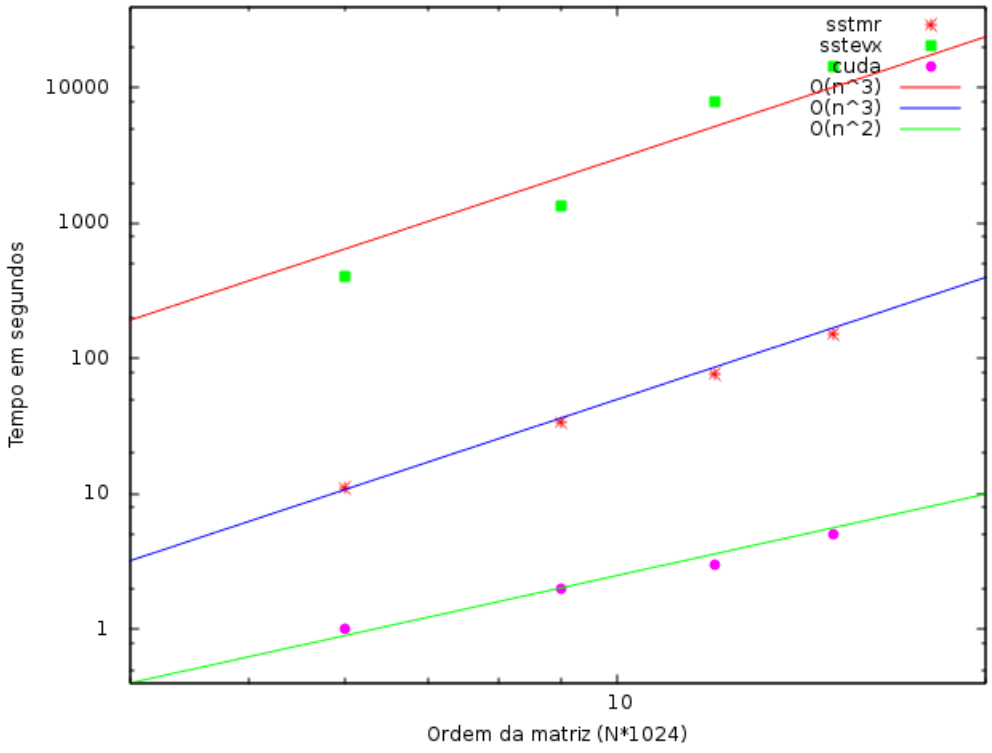
Ao observar os dados contidos na Tabela 1, pode-se notar que a rotina implementada em CUDA possui tempos inferiores as outras duas rotinas, ressaltando a diferença de tempo entre a CUDA e a sstevx que apresenta diferença bastante significativa.

Tabela 1 – Tempos de execução em segundos das três rotinas com precisão simples

N x 1024	SSTEMR	SSTEVX	CUDA
1	0,137	0,484	0,198
3	1,795	48,094	0,406
6	11,037	411,567	1,033
9	34,554	1340,621	2,041
12	77,171	8007,885	3,794
15	125,780	14471,786	5,966

Observação: os eixos coordenados dos gráficos da Figura (20) até a Figura (27), estão em escala logarítmica.

Figura 20 – Tempos de execução em segundos das três rotinas com precisão simples



Os dados apresentados na Tabela 1 estão aqui plotados na Figura 20, onde fica evidenciado a grande redução de tempo na execução da rotina CUDA, também nota-se que a medida que a ordem da matriz aumenta a velocidade da relativa entre a CUDA e as demais rotinas também aumenta, ou seja para matrizes de grande ordem a CUDA

tem significativo ganho de velocidade em detrimento das demais, lembrando que esses resultados são para matrizes tridiagonais simétricas .

A seguir pode-se observar dois gráficos representam os tempos de execução entres as rotinas citadas nos mesmos.

Figura 21 – Tempos de execução em segundos da rotina sstemr e CUDA com precisão simples

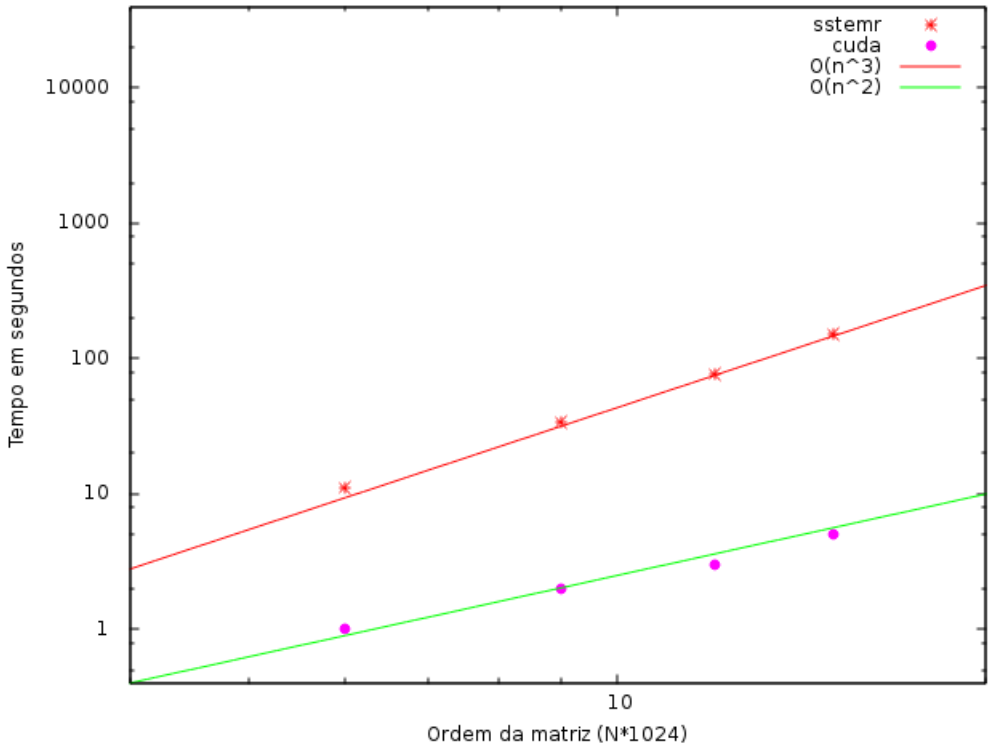
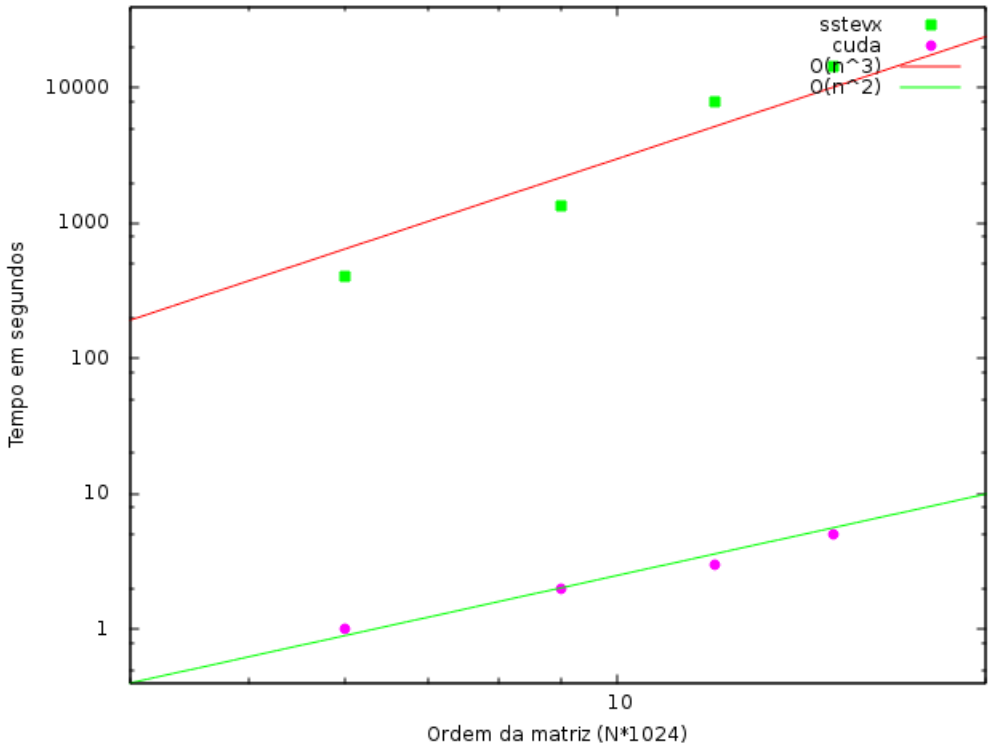


Figura 22 – Tempos de execução em segundos da rotina sstevx e CUDA com precisão simples



Nas Figuras 21 e 22 pode-se observar melhor a diferença de tempo de execução entre uma rotina e outra, o que fornece uma compreensão razoável dessa diferença , pois na Figura 20, estão presente o gráfico das três rotinas simultaneamente.

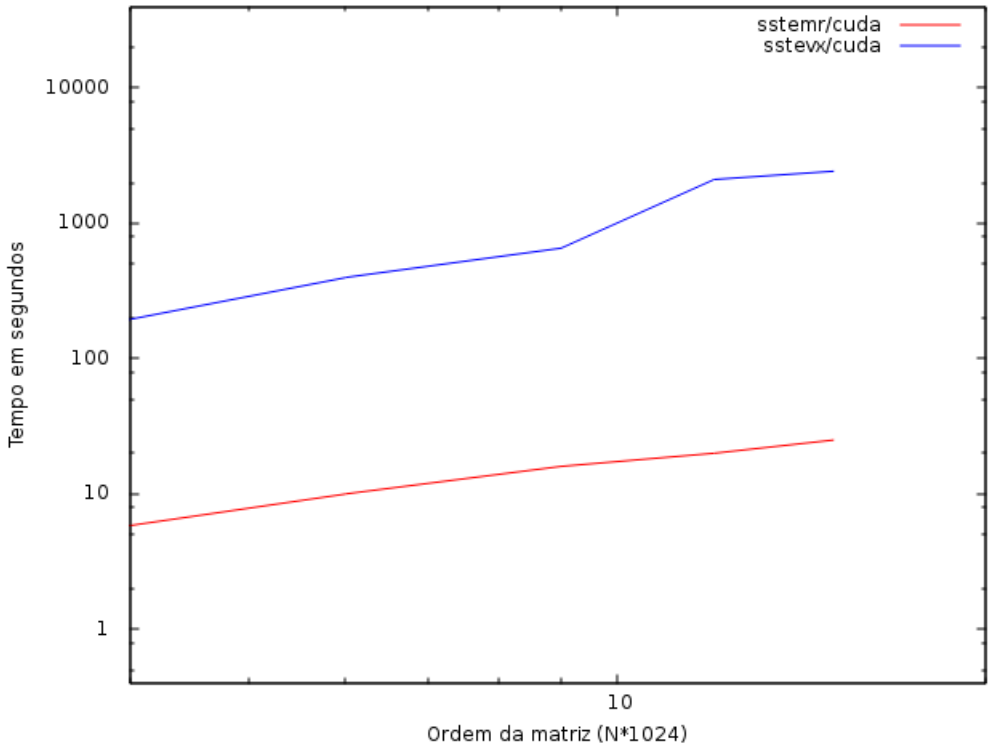
Também foi efetuado o cálculo da aceleração relativa entre CUDA e as demais rotinas que esta representado na Tabela

Tabela 2 – Aceleração relativa entre sstemr/CUDA e sstevx/CUDA com precisão simples

N x 1024	SSTEMR/CUDA	SSTEVX/CUDA
1	0,692	2,444
3	4,421	118,458
6	10,685	398,419
9	16,930	656,845
12	20,340	2110,670
15	25,608	2425,710

As aceleração representadas na Tabela 2 podem serem vistas no gráfico da Figura 23

Figura 23 – Aceleração relativa entre sstemr/CUDA e sstevx/CUDA com precisão simples



Ao observar a Figura 23 e a Tabela 2, pode-se perceber a grande aceleração da CUDA em detrimento da sstevx (bissecção e iteração inversa), e uma aceleração não tão acentuada em relação a rotina sstemr (QR), mas num patamar significativo.

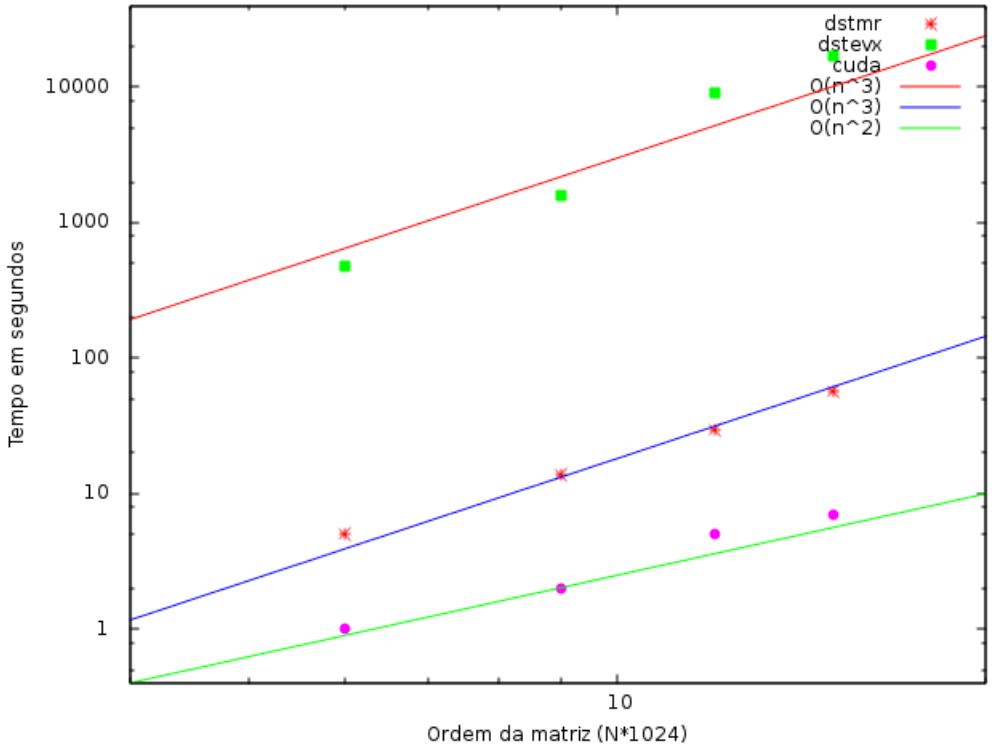
Para precisão dupla as rotinas são dstemr e dstvx com *script* idêntico ao da precisão simples, exceto no que tange a precisão que em *double*. A Tabela 3 contém os tempos de execução dessas rotinas

Tabela 3 – Tempos de execução em segundos das três rotinas com precisão dupla

N x 1024	DSTEMR	DSTEVX	CUDA
1	0,133	0,936	0,197
3	1,125	64,419	0,490
6	5,611	485,899	1,383
9	14,763	1575,248	2,992
12	30,908	9195,814	5,405
15	57,198	17084,183	7,706

Novamente aqui pode-se observar a superioridade da rotina CUDA em detrimento das demais rotinas. Esses resultados podem ser observados no gráfico da Figura 24

Figura 24 – Tempos de execução em segundos das três rotinas com precisão dupla



Tem-se a seguir os gráficos dos tempos das rotinas dstemr e CUDA, dstevx e CUDA, para uma melhor visualização e compreensão das diferenças de tempos entre essas duas rotinas e CUDA.

Figura 25 – Tempos de execução em segundos da rotina dstemr e CUDA com precisão dupla

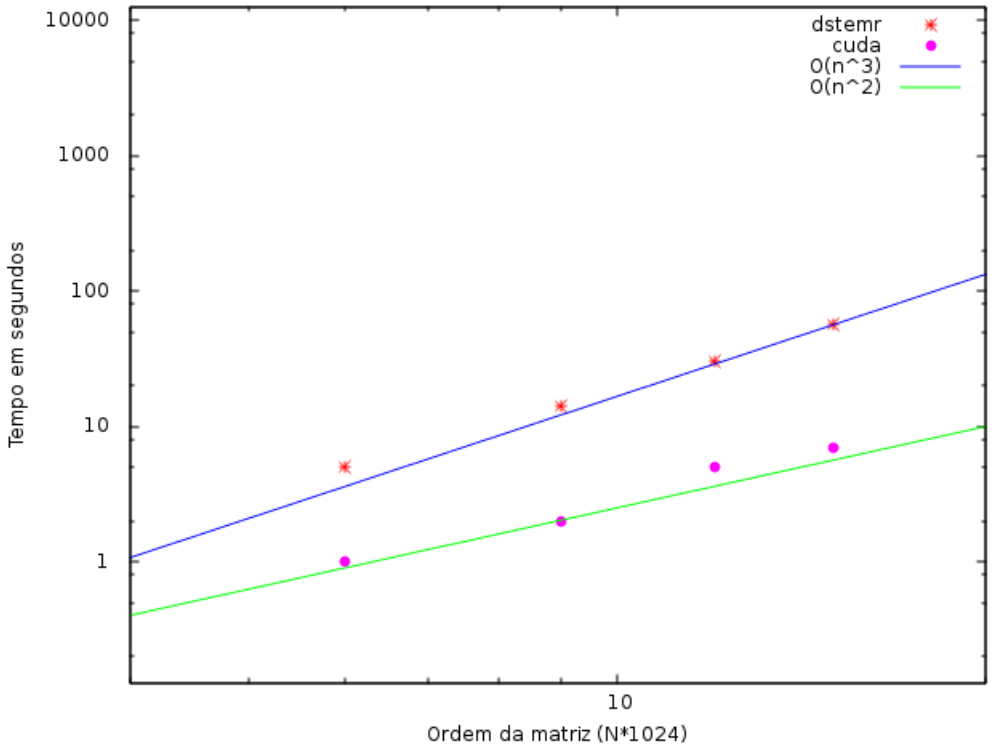
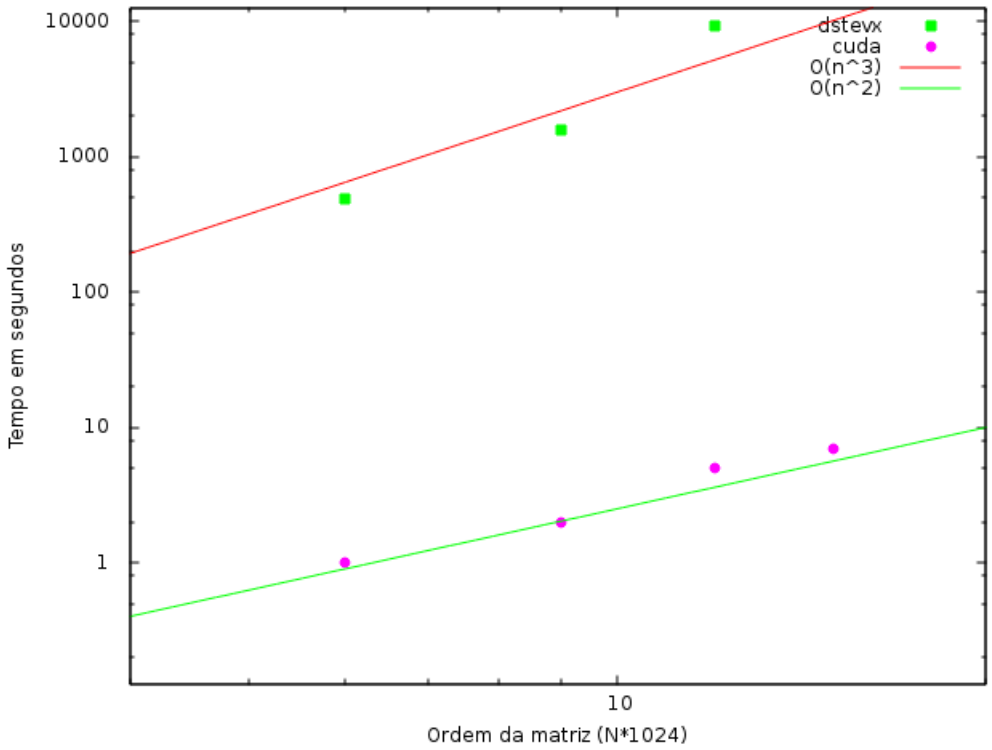


Figura 26 – Tempos de execução em segundos da rotina dstevx e CUDA com precisão dupla



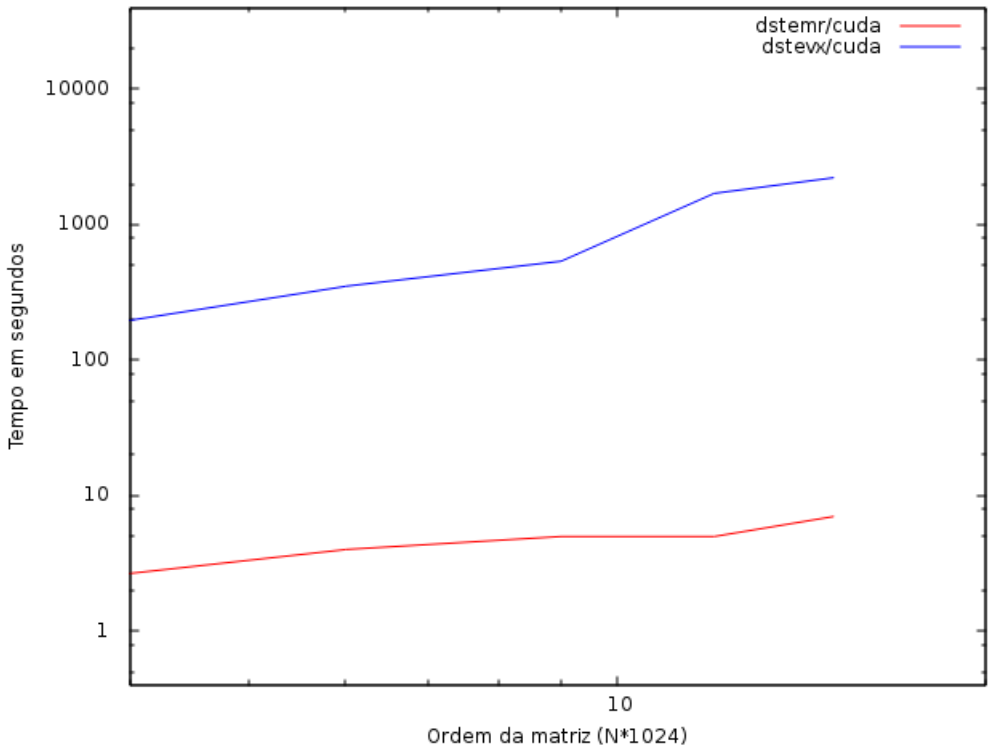
A Tabela 4 apresenta a aceleração relativa entre as rotinas dstemr, dstevx e CUDA com precisão dupla.

Tabela 4 – Aceleração relativa entre dstemr, dstevx e CUDA com precisão dupla

N	DSTEMR/CUDA	DSTEVX/CUDA
1	0,675	5,751
3	2,296	131,467
6	4,057	351,337
9	5,040	537,811
12	5,718	1701,353
15	7,420	2216,997

A aceleração relativa entre dstemr, dstevx e CUDA com precisão dupla pode ser vista no gráfico da figura

Figura 27 – Aceleração da rotina dstevx/CUDA e dstemr/CUDA com precisão dupla



Em ambos os casos tanto para precisão simples como para precisão dupla a aceleração da CUDA e bem maior em relação as rotinas sstevx e dstevx que utilizam a bissecção e iteração inversa para a determinação dos autovalores e autovetores do que nas rotinas sstmer e dstmer que fazem uso do QR, um outro fator que pode ser atribuído ao fato que a rotina CUDA tem um desempenho superior é a substituição na interação inversa da eliminação gaussiana pelo método de Thomas que tem custo computacional menor.

Uma questão relativa ao uso da CUDA de bastante relevância é o balanceamento de cargas nos SMs *(Streaming Multiprocessors)* para cada *kernel*, a seguir estão representados os gráficos do balanceamento relativo as etapas de extração e isolamento do autovalor e determinação do autovetor referente a aplicação da rotina CUDA em matriz tridiagonal simétrica real.

Figura 28 – Utilização da GPU na fase de isolamento do autovalor

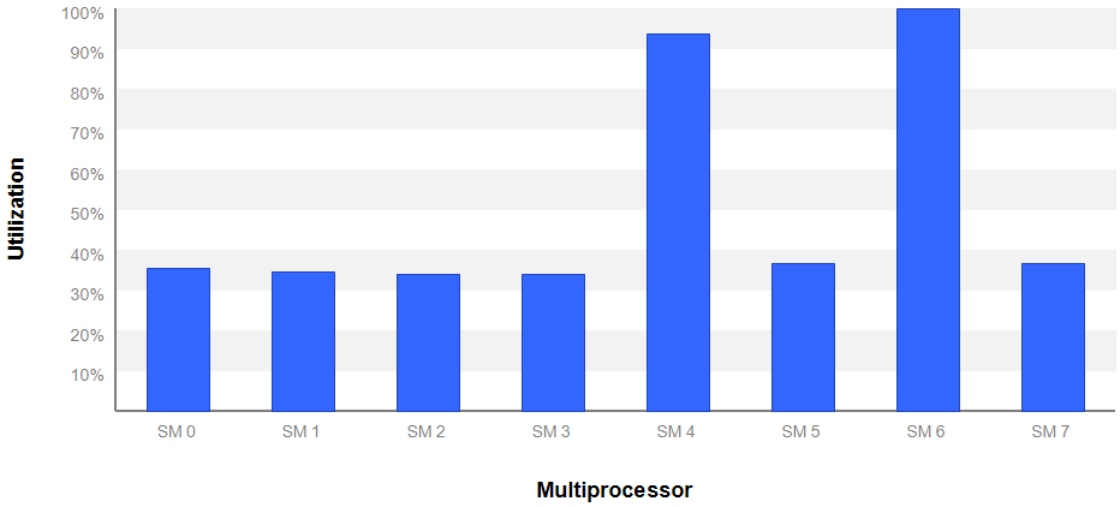


Figura 29 – Utilização da GPU na fase de extração do autovalor

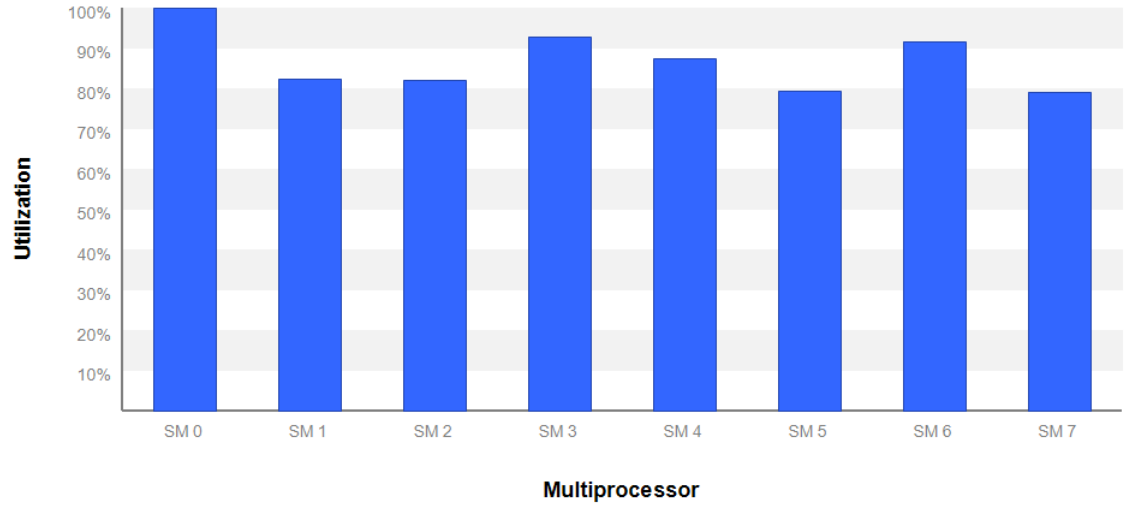
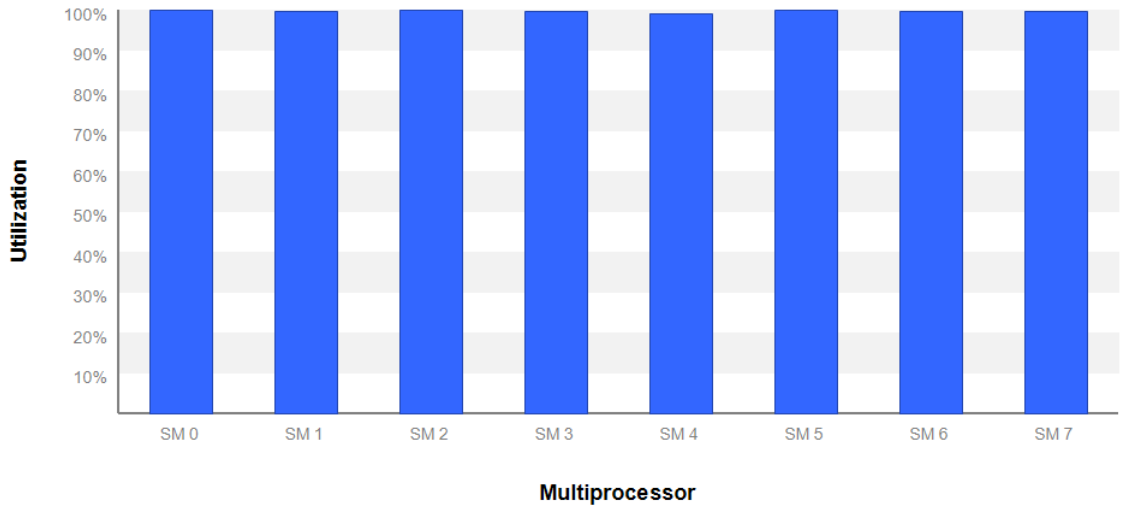


Figura 30 – Utilização da GPU no cálculo do autovetor



De acordo com a Figura 30 que representa a execução do *kernel* da obtenção dos autovetores os SMs estão 100 % ocupados o que é o ideal na execução de um *kernel*, isso implica que o balanceamento de carga é perfeito, consequentemente tem-se uma performance melhor do algoritmo. Ao passo que na fase de isolamento e extração do autovalor Figura 28 e Figura 29 esse balanceamento não foi o ideal de maneira mais acentuada na fase de isolamento . Um motivo para que haja esse diferença na distribuição de carga entre os SMs durante a fase de isolamento depende do espectro da matriz, ou seja, como os autovalores estão distribuídos ao longo do intervalo de Gershgorin, o balanceamento de cargas entre os SMs pode ficar demasiadamente prejudicado, isso pode ocorrer quando uma matriz apresenta muitos *clusters* de autovalores em uma determinada região. Problema este que pode ser contornado com um algoritmo de isolamento mais eficiente para estes casos. Uma melhora nesse balanceamento de cargas pode levar a um ganho de performance ainda maior.

4.2 Conclusão

Para atingir o objetivo desse trabalho que foi a redução de tempo no cálculo de autovalores e consequentemente dos autovetores de matrizes tridiagonais simétricas utilizando computação paralela, mas especificamente a CUDA da Nvdia, percorreu-se uma longa trilha iniciando com levantamento de conteúdos referentes ao assunto na fundamentação teórica. Durante essa fase inicial de estudos esta presente os resultados do teorema de Sturm e Gershgorin que foi de grande importância na obtenção dos resultados alcançados.

Após levantamento dos conteúdos referentes ao tema procedeu-se um estudo matemático das fases pelas quais se passa ao resolver problemas que envolvem o cálculo de autovalores e autovetores, além dos diversos métodos numéricos existentes tais como: Bissecção, Newton, Secante dentre outros, bem como seus algoritmos.

Procedeu-se também um estudo de cada um dos métodos numéricos acerca das suas propriedades computacionais e a aplicação dos teoremas de Sturm e Gershgorin nos algoritmos juntamente com os métodos numéricos, durante esse estudo também evidenciou-se dos tópicos de suma importância que são a função $neg(\lambda)$ que possibilita o cálculo do número de sinais na sequência de Sturm e o algoritmo do *zeroin* que calcula autovalores numa combinação do métodos da bissecção com o método da secante, onde todos esses conceitos e aplicação são direcionados para matrizes tridiagonais simétricas.

A programação paralela também teve seu espaço no arcabouço desse trabalho, onde esta presente um estudo sobre as arquiteturas de maquinas paralelas de acordo com a taxinomia de Flynn e os principais modelos de programação paralela tais como MPI, OpenMP, OpenCL e a CUDA que o modelo utilizado nesse trabalho por ser o modelo escolhido o mesmo possui uma sessão mais longa que os demais detalhando suas principais propriedades e aplicabilidade.

Finalmente chega-se aos resultados, onde os mesmos comprovam que o objetivo proposto foi alcançado através de escolhas adequadas e modificações realizadas uma dessas modificações feita foi a substituição da eliminação gaussiana na rotina da iteração inversa pelo método de Thomas possibilitando assim um ganho de velocidade significativo, já que o custo computacional do método de Thomas é menor do que a eliminação gaussiana.

Quanto a trabalhos futuros pode-se procurar formas de uma melhor distribuição de cargas do *kernel* para cálculo dos autovalores em CUDA. Um fator limitante no cálculo dos autovetores é o custo do armazenamento dos dados, assim poderia se pensar em uma forma mais otimizada que permitisse realizar a determinação tanto dos autovalores bem como dos autovetores de matrizes de maior dimensão.

Referências

1 COLEMAN, T. F. *Large Sparse Numerical Optimization*. [S.l.: s.n.], 1984. v. 165. (Lecture Notes in Computer Science, v. 165).

2 GOLUB, G. G.; LOAN, C. F. V. *Matrix Computations*. 3rd. ed. [S.l.]: The Johns Hopkins University Press, 2013.

3 BOLDRINI, J. *Algebra linear*. [S.l.]: HARBRA, 1986. ISBN 9788529402024.

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

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

6 BIEZUNER, R. J. *Notas de Aula Álgebra Linear Numérica*. Belo Horizonte: Universidade Federal de Minas Gerais, 2009.

7 LANCASTER, P.; TISMENETSKY, M. *The Theory of Matrices*. Second. [S.l.: s.n.], 1985.

8 LEVEQUE, R. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems (Classics in Applied Mathematics)*. University of Washington Seattle, Washington: SIAM, Society for Industrial and Applied Mathematics, 2007. ISBN 0898716292, 9780898716290.

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

10 APOSTOL, T. M. *Calculus Vol. 1*. Rio de janeiro: Editora Reverté, 1979.

11 RUGGIERO M. A . G . & LOPES, V. . L. . R. *Cálculo numérico : aspectos teóricos e computacionais*. São Paulo: Makron, 1997.

12 GUIDORIZZI, H. L. *Um Curso de Cálculo, vol. 1*. Rio de Janeiro: Livros Técnicos e Científicos, 2001.

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

14 CONTELLES, J. M. B. *Algoritmos Paralelos para el Cálculo de los Valores Propios de Matrices Estructuradas*. 425 f. Tese (Sistemas Informáticos y Computación) — Universidad Politécnica de Valencia, Valencia-Espanha, 1996.

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

16 GIVENS, W. J. *Numerical Computation of the Characteristic Values of a Real Symmetric Matrix*. inst-ORNL:adr, 1954.

17 BARTH, W.; MARTIN, R. S.; WILKINSON, J. H. Calculation of the eigenvalues of a symmetric tridiagonal matrix by the method of bisection. *Numerische Mathematik*, V9, n. 5, p. 386–393, apr 1967.

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

19 WATKINS, D. S. *Fundamentals of Matrix Computations; 3nd ed.* Newark, NJ: Wiley, 2010.

20 RALHA, R. Parallel solution of the symmetric tridiagonal eigenvalue problem on a transputer network. *SEMNI*, n. 93, p. 1026–1033, 1993.

21 FLETCHER, C. *Computational techniques for fluid dynamics*. [S.l.]: Springer, 1988. (Springer series in computational physics). ISBN 9780387194660.

22 HIRSCH, C. (Ed.). *Numerical Computation of Internal And External Flows: Fundamentals of Numerical Discretization*. New York, NY, USA: John Wiley & Sons, Inc., 1988. ISBN 0-471-91762-1.

23 CANAL, A. P. *Paralelização de Métodos de Resolução de Sistemas Lineares Esparsos com o DECK em um Cluster de PCs*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2000.

24 DEVRIES, P. L. *A First Course in Computational Physics*. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1993. ISBN 0471548693.

25 SÜLI, E.; MAYERS, D. F. *An Introduction to Numerical Analysis*. 1st. ed. [S.l.]: Cambridge University Press, 2003. ISBN 978-0-521-00794-8.

26 WILKINSON, B.; ALLEN, C. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. [S.l.]: Pearson/Prentice Hall. (An Alan R. Apt book). ISBN 9780131405639.

27 ARQUITETURAS Paralelas: Site. 2015. Disponível em: <http://www-usr.inf.ufsm.br/~sandro/elc139/tarefa1.php>. Acesso em: 27 mar. 2015.

28 CULLER, D. E.; GUPTA, A.; SINGH, J. P. *Parallel Computer Architecture: A Hardware/Software Approach*. 1st. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. ISBN 1558603433.

29 SPOLAVORE, A. D. *Computação de Alto Desempenho Aplicada a Modelagem Numérica de Fenômenos Atmosféricos*. Dissertação (Mestrado) — Universidade Federal do Rio Grande (FURG), 2014.

30 TANENBAUM, A. *Sistemas operacionais modernos*. [S.l.]: Prentice-Hall do Brasil, 2010. ISBN 9788576052371.

31 CHAPMAN, B. et al. *Using OpenMP : portable shared memory parallel programming*. Cambridge, Mass., London: MIT Press, 2008. ISBN 978-0-262-53302-7.

32 EL-REWINI, H.; ABD-EL-BARR, M. *Advanced Computer Architecture and Parallel Processing (Wiley Series on Parallel and Distributed Computing)*. [S.l.]: Wiley-Interscience, 2005. ISBN 0471467405.

33 MUNSHI, A. et al. *OpenCL Programming Guide*. 1. ed. [S.l.]: Addison-Wesley Professional. ISBN 0321749642, 9780321749642.

34 GASTER, B. et al. *Heterogeneous Computing with OpenCL*. 1. ed. [S.l.: s.n.]. ISBN 0123877660, 9780123877666.

35 COOK, S. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. [S.l.]: Elsevier Science, 2012. (Applications of GPU Computing Series). ISBN 9780124159884.

36 FARBER, R. *CUDA Application Design and Development*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN 9780123884268, 9780123884329.

37 KIRK, D. B.; HWU, W.-m. W. *Programming Massively Parallel Processors: A Hands-on Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2013. ISBN 0123814723, 9780123814722.