



CÁLCULO NUMÉRICO COMPUTACIONAL

Sérgio Peters
Julio Felipe Szeremeta

 editora ufsc

COLEÇÃO DIDÁTICA

CÁLCULO NUMÉRICO COMPUTACIONAL

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Reitor

Ubaldo César Balthazar

Vice-Reitora

Alacoque Lorenzini Erdmann

EDITORA DA UFSC

Diretora Executiva

Gleisy R. B. Fachin

Conselho Editorial

Gleisy R. B. Fachin (Presidente)

Adriano Luiz Duarte

Aguinaldo Roberto Pinto

Carlos Luiz Cardoso

Eliete Cibele Cipriano Vaz

Ione Ribeiro Valle

Gestine Cássia Trindade

José Paulo Speck Pereira

Josimari Telino de Lacerda

Katia Jakovljevic Pudla Wagner

Luana Renostro Heinen

Luis Alberto Gómez

Mauri Furlan

Pedro Paulo de Andrade Júnior

Editora da UFSC

Campus Universitário – Trindade

Caixa Postal 476

88040-900 – Florianópolis-SC

Fone: (48) 3721-9408

editora@contato.ufsc.br

www.editora.ufsc.br

Sérgio Peters
Julio Felipe Szeremeta

CÁLCULO NUMÉRICO COMPUTACIONAL

© 2018 Editora da UFSC

Coordenação editorial:

Flavia Vicenzi

Capa:

Paulo Roberto da Silva

Editoração:

Lais Tomaselli Krause

Alicia da Costa Edwirges

Cristiano Tarouco

Revisão:

Heloisa Hübbe de Miranda

Ficha Catalográfica

(Catalogação na fonte pela Biblioteca Universitária da Universidade Federal de Santa Catarina)

P481c Peters, Sérgio

Cálculo numérico computacional [recurso eletrônico] / Sérgio Peters, Julio Felipe Szeremeta. – Dados eletrônicos – Florianópolis : Editora da UFSC, 2018.

527 p. : il., gráfs., tabs. – (Coleção Didática)

Inclui bibliografia.

E-book (PDF)

Disponível em: <<http://editora.ufsc.br/estante-aberta/>>

ISBN 978-85-328-0838-7

1. Matemática aplicada à computação. 2. Computação – Matemática.
3. Informática. I. Szeremeta, Julio Felipe. II. Título. III. Série.

CDU: 519.67

Elaborado por Jonathas Troglio – CRB 1411093



Este livro está sob a licença Creative Commons, que segue o princípio do acesso público à informação. O livro pode ser compartilhado desde que atribuídos os devidos créditos de autoria. Não é permitida nenhuma forma de alteração ou a sua utilização para fins comerciais.

br.creativecommons.org

*Aos meus filhos, Pedro, João, Bia, e à minha mulher, Silvia,
companheiros(as) e incentivadores(as) desta obra.*

Sérgio Peters

*Aos meus filhos, Igor e Yuri, e particularmente à minha esposa,
Mariza, companheira de longa jornada, pela paciência, apoio
e motivação a mim dedicados.*

Julio Felipe Szeremeta

SUMÁRIO

PREFÁCIO	13
APRESENTAÇÃO	15
CAPÍTULO 1	
SISTEMAS DE NUMERAÇÃO E ERROS NUMÉRICOS	19
1.1 REPRESENTAÇÃO DE QUANTIDADES	21
1.2 SISTEMA DE NUMERAÇÃO DECIMAL ($\beta = 10$)	25
1.2.1 Utiliza apenas dez símbolos.....	25
1.2.2 Faz uso do zero	26
1.2.3 Adota o princípio da posicionalidade	26
1.3 SISTEMA BINÁRIO ($\beta = 2$).....	27
1.4 SISTEMA HEXADECIMAL ($\beta = 16$).....	29
1.5 CONVERSÕES ENTRE SISTEMAS DE NUMERAÇÃO	30
1.5.1 Conversão de base β para base 10	30
1.5.2 Conversão de base 10 para base β	30
1.5.3 Conversões diretas entre binário e hexadecimal	33
1.6 REPRESENTAÇÃO DIGITAL DE NÚMEROS	35
1.6.1 Padrão 16 <i>bits</i> original.....	36
1.6.1.1 Menor positivo representável (mp)	38
1.6.1.2 Maior positivo representável (MP).....	38
1.6.1.3 Representação do zero	39
1.6.1.4 Quantidade máxima de elementos representáveis	41
1.6.2 Otimizações da variável de 16 <i>bits</i> , segundo o padrão IEEE 754	43
1.6.2.1 Polarização	44
1.6.2.2 Não armazenamento do primeiro <i>bit</i> não nulo da mantissa	45
1.6.2.3 Flexibilidade na normalização da mantissa	46
1.6.2.4 Identificação da região de <i>overflow</i>	47

1.7	REPRESENTAÇÕES NUMÉRICAS, SEGUNDO O PADRÃO IEEE 754	48
1.7.1	Variável de 32 bits – <i>binary32</i> , tipo <i>float</i> do C, precisão simples	49
1.7.2	Variável de 64 bits – <i>binary64</i> , tipo <i>double</i> do C, precisão dupla	57
1.7.3	Variável de 128 bits – <i>binary128</i> , precisão quádrupla	57
1.8	REPRESENTAÇÃO DE VARIÁVEIS INTEIRAS: PADRÃO ANSI C.....	58
1.8.1	<i>Short int</i>	58
1.9	TIPOS DE ERROS EXISTENTES EM REPRESENTAÇÕES DIGITAIS.....	60
1.9.1	Erros inerentes	60
1.9.2	Erros de arredondamento.....	60
1.9.2.1	Arredondamento manual	61
1.9.2.2	Arredondamento digital	61
1.9.2.2.1	Armazenamento de Racionais ilimitados	62
1.9.2.2.2	Armazenamento de Irracionais	62
1.9.2.2.3	Armazenamento com mudança de base.....	63
1.9.2.2.4	Armazenamento de resultados de operações aritméticas	64
1.9.2.3	Consequências dos erros de arredondamento	68
1.9.2.3.1	Propagação de erros devido à perda de significação.....	68
1.9.2.3.2	Instabilidade numérica nos algoritmos.....	70
1.9.3	Erros de truncamento.....	70
1.9.4	Medida de erros	75
1.10	CONCLUSÕES.....	79
	COMPLEMENTANDO.....	81

CAPÍTULO 2

SOLUÇÃO COMPUTACIONAL DE SISTEMAS

DE EQUAÇÕES LINEARES 89

2.1	SOLUÇÃO ELIMINATIVA DE $A * X = B$	93
2.1.1	Método de eliminação de Gauss	94
2.1.1.1	Pivotação parcial	100
2.1.1.2	Pivotação total	105
2.1.1.3	Quantitativo de soluções de $A * X = B$	110
2.1.2	Método de Gauss-Jordan	116
2.1.3	Método da inversão de matrizes	116
2.1.4	Método de decomposição <i>LU</i> (de Crout)	122
2.1.5	Método de Cholesky	140

2.1.6	Solução de sistemas com a matriz de coeficientes do tipo banda	141
2.1.7	Sistemas lineares mal condicionados	148
2.2	SOLUÇÃO DE SISTEMAS LINEARES POR MÉTODOS ITERATIVOS	154
2.2.1	Método iterativo de Jacobi	156
2.2.2	Método iterativo de Gauss-Seidel	161
2.2.3	Convergência dos métodos iterativos	164
2.2.3.1	Teorema de convergência – critério de Scarborough	164
2.2.4	Aplicação de coeficientes de relaxação	168
2.3	CONCLUSÕES	182
	COMPLEMENTANDO	183

CAPÍTULO 3

SOLUÇÃO DE EQUAÇÕES NÃO LINEARES

A UMA INCÓGNITA

3.1	ISOLAMENTO OU LOCALIZAÇÃO DE SOLUÇÕES DE $f(x) = 0$	190
3.2	REFINAMENTO DA SOLUÇÃO ISOLADA	192
3.2.1	Métodos de quebra	194
3.2.1.1	Método da bisseção (ou bipartição) para raízes reais	194
3.2.1.2	Método da falsa posição	200
3.2.1.3	Método da falsa posição modificado	204
3.2.2	Métodos de linearização	208
3.2.2.1	Método da iteração linear	210
3.2.2.1.1	Teorema de convergência	213
3.2.2.2	Método de Newton	216
3.2.2.3	Método da secante	223
3.2.2.4	Tratamento geral de funções com derivada nula em $x = \alpha$	227
3.2.2.5	Rotinas de funções predefinidas	234
3.2.2.5.1	Recíproco de um número	234
3.2.2.5.2	Raiz quadrada de um número	236
3.3	SOLUÇÃO DE EQUAÇÕES POLINOMIAIS	238
3.3.1	Localização de raízes de equações polinomiais	239
3.3.2	Natureza de raízes de equações polinomiais	247
3.3.3	Cálculo eficiente de valores das funções polinomiais e suas derivadas	250
3.3.4	Método de Newton para $P_n(x) = 0$	254

3.3.4.1	Determinação de raízes de equações polinomiais com multiplicidade	258
3.3.4.2	Purificação de raízes obtidas com redução de grau	274
3.3.5	Determinação de raízes de equações pelo método de Müller	276
3.4	CONCLUSÕES.....	281

CAPÍTULO 4

RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES NÃO LINEARES..... 283

4.1	MÉTODO DE NEWTON	288
4.2	MÉTODO DE NEWTON COM DERIVADAS PARCIAIS NUMÉRICAS.....	296
4.3	MÉTODO DE BROYDEN.....	301
4.4	CONCLUSÕES.....	306

CAPÍTULO 5

APROXIMAÇÃO POLINOMIAL POR INTERPOLAÇÃO, *SPLINE* E CURVAS DE BÉZIER..... 307

5.1	APROXIMAÇÃO POR INTERPOLAÇÃO POLINOMIAL.....	312
5.1.1	Unicidade do interpolador	317
5.1.2	Determinação eficiente do interpolador	318
5.1.2.1	Expressão do interpolador polinomial $P_n(x)$ na base dos polinômios de Lagrange	319
5.1.2.2	Interpolador de Gregory-Newton com diferenças.....	323
5.1.3	Avaliação do erro de truncamento na interpolação	332
5.2	INTERPOLAÇÃO DE FUNÇÕES COM VÁRIAS VARIÁVEIS INDEPENDENTES.....	338
5.3	APROXIMAÇÃO POR INTERPOLAÇÃO <i>SPLINE</i>	342
5.4	TRATAMENTO DO FENÔMENO DE RUNGE	358
5.5	APROXIMAÇÃO DE NÃO FUNÇÕES VIA PARAMETRIZAÇÃO	361
5.6	APROXIMAÇÃO POR CURVAS DE BÉZIER.....	363
5.7	CONCLUSÕES.....	371

CAPÍTULO 6
APROXIMAÇÃO POR SÉRIES E POR FUNÇÕES RACIONAIS 373

6.1 APROXIMAÇÃO DE $y = f(x)$ POR SÉRIES376
 6.1.1 Aproximação por séries de Taylor 377
 6.1.2 Aproximação por séries de Tchebychev 386
6.2 APROXIMAÇÃO RACIONAL DE PADÉ403
6.3 CONCLUSÕES.....420

CAPÍTULO 7
APROXIMAÇÃO PELA TENDÊNCIA VIA AJUSTE DE CURVAS 421

7.1 MÉTODO DOS MÍNIMOS QUADRADOS PARA AJUSTE
A FUNÇÕES POLINOMIAIS428
7.2 AJUSTE POR MÍNIMOS QUADRADOS A FUNÇÕES
NÃO POLINOMIAIS437
 7.2.1 Determinação direta da ajustadora não polinomial..... 438
 7.2.2 Ajuste por transformações de forma paramétrica em polinomial 443
 7.2.2.1 Ajuste a exponenciais 443
 7.2.2.2 Ajuste a curvas geométricas 446
 7.2.2.3 Ajuste a curvas hiperbólicas 446
 7.2.2.4 Ajuste a sigmoides 446
7.3 CONCLUSÕES.....451

CAPÍTULO 8
INTEGRAÇÃO NUMÉRICA 453

8.1 INTEGRAÇÃO NUMÉRICA POR MÉTODOS
DE NEWTON-COTES458
 8.1.1 Método dos trapézios..... 458
 8.1.2 Método de Simpson 466
8.2 INTEGRAÇÃO NUMÉRICA GAUSSIANA
OU QUADRATURA GAUSSIANA.....471
 8.2.1 Método de Gauss-Legendre..... 473
 8.2.2 Método Gauss-Tchebychev 484
8.3 CONCLUSÕES.....487

CAPÍTULO 9

RESOLUÇÃO NUMÉRICA DE EQUAÇÕES DIFERENCIAIS

ORDINÁRIAS.....489

9.1 MÉTODO DE EULER SIMPLES493

9.2 MÉTODOS DE RUNGE-KUTTA497

9.2.1 Método de Euler aperfeiçoado 498

9.2.2 Método de Runge-Kutta de 4ª ordem 502

9.3 SOLUÇÃO NUMÉRICA DE SISTEMAS DE PVI DE 1ª ORDEM
E DE PVI DE ORDEM SUPERIOR509

9.4 SOLUÇÃO NUMÉRICA DE EDO COM PROBLEMA DE VALOR
NO CONTORNO (PVC)514

9.4.1 Métodos de determinação de uma condição inicial
em função de uma condição de contorno..... 515

9.5 CONCLUSÕES.....521

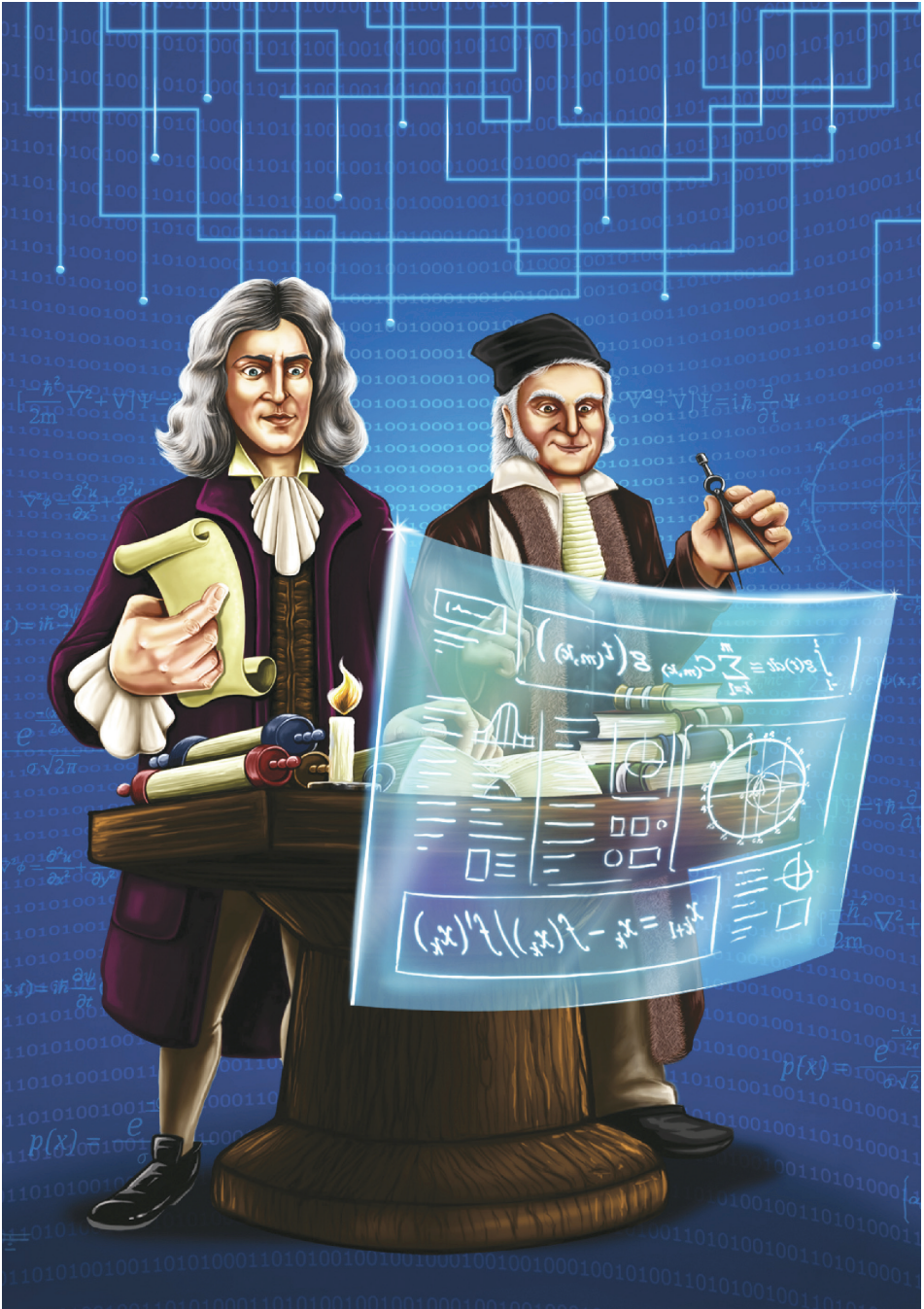
COMPLEMENTANDO...522

REFERÊNCIAS.....525

PREFÁCIO

A arte de modelar matematicamente os fenômenos físicos é uma parte muito importante para todas as áreas da Engenharia, pois com estes modelos podemos estudar a natureza na qual vivemos e também desenvolver os mais diversos equipamentos de uso social, aumentando a qualidade de vida das pessoas. O voo dos pássaros, o suave correr de um córrego, os ventos e as ondas do mar são alguns exemplos de fenômenos que possuem uma representação matemática e podem ser estudados matematicamente. Tais fenômenos são representados por sistemas de equações diferenciais, normalmente parciais e não lineares. Quando estes sistemas de equações diferenciais são muito complexos, como na aerodinâmica de um avião ou de um automóvel, por exemplo, os métodos analíticos não conseguem resolvê-los, e técnicas numéricas devem ser empregadas. Para a aplicação destas técnicas numéricas, e para melhor representar os dados e detalhes do problema, é necessário um ferramental de fundamentos numérico/matemático que devem ser dominados pelo analista/engenheiro. É exatamente do que trata o livro dos professores Sérgio Peters e Julio Felipe Szeremeta, que reúne fundamentos essenciais para o domínio de muitas técnicas numéricas empregadas nos mais diversos ramos da física e da engenharia. Fundamentos dos erros numéricos, solução de equações gerais e polinomiais, solução de sistemas de equações lineares e não lineares, ajuste de curvas genérico, representação de funções em séries e soluções de equações diferenciais ordinárias, são tópicos apresentados neste texto sempre com a preocupação de transmitir ao aluno a importância e o significado do aprendizado desses fundamentos. O livro expõe estas técnicas com a visão das Ciências da Computação e da Engenharia, proporcionada pela formação dos autores, ampliando a bibliografia disponível com um texto didático e bem focado.

Prof. Clovis Raimundo Maliska
*Laboratório de Simulação Numérica em Mecânica
dos Fluidos e Transferência de Calor – SINMEC
Departamento de Engenharia Mecânica – EMC – UFSC*



Isaac Newton e Johann Carl Friedrich Gauss
Ilustração de Claudio José Girardi, 2017.

APRESENTAÇÃO

O **cálculo numérico** é um ramo da matemática aplicada que trata da solução de modelos matemáticos das mais variadas áreas do conhecimento e, diferentemente do **cálculo diferencial e integral**, faz uso de metodologias **construtivas** para a solução desses modelos. Nessas metodologias estão envolvidas apenas operações aritméticas elementares e, normalmente, geram uma sequência de aproximações para a solução exata do modelo. Daí a necessidade de se avaliar quão “distante” a solução aproximada está da solução exata, mesmo quando a desconhecemos. O cálculo numérico é, portanto, não apenas uma metodologia construtiva de solução de modelos, mas também envolve a avaliação da qualidade da solução obtida. Aqui o termo qualidade tem o mesmo significado daquele da fabricação de um determinado produto, isto é, delimitação do grau de confiança a ser depositado no mesmo e dispêndio do menor custo possível para obtê-lo. Como a geração de uma sequência de aproximações para a solução exata é tipicamente repetitiva e envolve somente operações aritméticas básicas, pode ser implementada e executada por computadores. Daí a denominação de **Cálculo numérico computacional**.

Mantendo a praxe, este livro é o resultado da longa experiência de mais de três décadas dos autores lecionando as disciplinas de Cálculo Numérico e Análise Numérica Computacional em cursos da área das ciências exatas, notadamente engenharias e ciências da computação, da Universidade Federal de Santa Catarina – UFSC. Por consequência, foi organizado seguindo os conteúdos tradicionais destas disciplinas.

São princípios norteadores desta obra:

1. Gratuidade e publicidade: versão na forma digital disponível para acesso livre e aberto, incluindo um repositório de Algoritmos, Caderno de Exercícios e de Respostas, nos endereços eletrônicos: <<http://editora.ufsc.br/estante-aberta/>> e <<http://sergiopeters.prof.ufsc.br/livro-calculo-numerico-computacional/>>.

2. Reusabilidade: utiliza *software* livre, com algoritmos escritos em Octave, podendo ser reutilizado pelos estudantes em sala de aula, nos estudos e em futuros locais de trabalho, e é compatível com o *MatLab*[®].
3. Intercambiabilidade: utiliza formato de notação científica com “ponto”, em vez de “vírgula”, para facilitar o processo de copiar, colar e testar exemplos e algoritmos, uma vez que linguagens de programação usam notação com o ponto “.” para separar a parte inteira da parte fracionária dos números, conforme padrão em países de cultura inglesa; e utiliza o símbolo “*” como operador de produto em vez de “x”.
4. Confiabilidade: todos os resultados aproximados apresentam uma medida da sua qualidade, ou seja, é dimensionado o seu erro.
5. Acessibilidade: todos os assuntos e métodos foram abordados e descritos pedagogicamente em linguagem que mantém a preocupação constante com o entendimento pelo leitor.
6. Integralidade: além dos tópicos abordados nos programas das disciplinas de Cálculo Numérico e correlatas, oferecidas nos cursos tradicionais das ciências exatas do país, essa obra traz contribuições de tópicos complementares, inéditos na literatura nacional da área, com destaque para:
 - a) representação digital dos números e avaliação de erros de arredondamento;
 - b) comparativo da eficiência numérica e computacional dos métodos de solução de sistemas de equações lineares abordados;
 - c) metodologia inovadora para determinação das raízes, e respectivas multiplicidades, de equações polinomiais;
 - d) interpolação polinomial bi-dimensional, aproximação por *splines* cúbicas e por curvas de Bézier;
 - e) ajuste generalizado de curvas com análise estatística dos resultados;
 - f) aproximação de funções por séries de Tchebychev e por racionais de Padé;

- g) fundamentação matemática dos métodos de integração numérica por Gauss-Legendre e Gauss-Tchebychev; e
- h) tratamento alternativo para solução numérica de equações diferenciais ordinárias com condições de contorno.

Manifestamos nossa gratidão aos nossos muitos alunos de graduação, que foram os principais motivadores e que tanto contribuíram para a concretização desta obra, com as suas sugestões de aprimoramento, dificuldades de entendimento, dúvidas e proveitosas discussões.

Agradecemos o incentivo e a contribuição de todos os colegas do Departamento de Informática e Estatística, especialmente aos colegas, professores de Cálculo Numérico, Nelcy Dabrowski de Araújo Mendonça, Ricardo Felipe Custódio, Daniel Santana de Freitas e Juliana Eyng, e aos professores de Estatística, Pedro Alberto Barbeta, André Wust Zibetti e Andreia Zanella, pelas sugestões e apoio no desenvolvimento do tópico relativo à análise estatística dos resultados do ajustamento de curvas.

À professora Lorena Hulse Bittencourt, professora de matemática do ensino fundamental e médio do professor Sérgio Peters, nos colégios Henrique Fontes e Dehon de Tubarão-SC, que abriu oportunidades de experimentar a missão de ensinar, através de aulas particulares aos colegas de colégio.

À professora Verônica Fialka, *in memoriam*, alfabetizadora do professor Julio Felipe Szeremeta, por ter conseguido, mesmo em condições precárias e improváveis, despertar-lhe o interesse e a paixão pela matemática.

Somos gratos ao trabalho de adequação de linguagem, revisão textual, e diagramação inicial, efetuado pelos técnicos Denise Aparecida Bunn, Claudia Leal Estevão e Cláudio José Girardi, que resultou em um texto com linguagem dialógica mais acessível ao público-alvo, melhorando a efetividade do entendimento pelo leitor sobre os assuntos abordados.

Agradecemos à equipe da Editora da UFSC e à professora Gleisy R. B. Fachin, diretora executiva da Editora da UFSC.

SISTEMAS DE NUMERAÇÃO E ERROS NUMÉRICOS

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- compreender os fundamentos dos sistemas de numeração utilizados atualmente;
- efetuar conversões entre diferentes sistemas de numeração;
- realizar representação digital de quantidades reais e inteiras; e
- identificar os tipos e as consequências de erros existentes em representações digitais.

Neste capítulo, vamos apresentar alguns tópicos que fundamentam o cálculo numérico computacional, como as representações de quantidades em computadores, suas respectivas propriedades e os possíveis erros gerados.

1.1 REPRESENTAÇÃO DE QUANTIDADES

As formas de representação de quantidades ou números, chamadas de Sistemas de Numeração, evoluíram e continuam evoluindo no curso da história humana, passando dos sistemas meramente “*associativos*” para os sistemas “simbólicos”, em uso há milênios, nos quais determinado símbolo pode representar vários valores, e, finalmente, para os sistemas “lógicos magnéticos”, atualmente em uso nos computadores.



Livros de alfabetização ilustram os números através de “associações” com os dedos da mão.

Podemos agrupar os sistemas de numeração simbólicos em duas grandes categorias. Na primeira, denominada “aditiva”, cada símbolo tem um valor fixo e o valor total é representado pela junção dos valores específicos de cada símbolo do número, por exemplo, os antigos sistemas romano, grego, egípcio e hebraico. A segunda categoria, que praticamente prevaleceu à primeira e foi desenvolvida pelos astrónomos hindus por volta do século VI, é denominada “posicional”. Nesta categoria, o valor de cada símbolo é relativo e depende da sua posição no número.

A representação simbólica de um número depende da quantidade de símbolos utilizados. Essa quantidade é denominada de **base**, que denotaremos por um número natural β .

A seguir, vamos apresentar exemplos de um sistema aditivo e de sistemas posicionais com suas respectivas bases.

Exemplo de sistema aditivo:

- a) **Sistema romano:** base $\beta = 7$ e símbolos $\{I, V, X, L, C, D, M\}$. Note a inexistência do “zero” e a limitação dos valores representáveis apenas à ordem dos milhares.

Exemplos de três sistemas posicionais:

- a) **Sistema decimal:** base $\beta = 10$ e símbolos $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, denominados de *dígitos ou algarismos*⁹. Note a presença do “zero”.

Dígito: do latim *digitus*, mesmo que dedo; sinal convencional que representa graficamente os números; e cada um dos símbolos que representam os números inteiros positivos menores que a base de um sistema numérico. Fonte: Adaptado de Instituto Antônio Houaiss (2009).



Algarismo: derivação da palavra *algorismi*, versão latina do nome Al-Khwarizmi, matemático árabe que escreveu tratados sobre aritmética e álgebra, entre outros. Fonte: Só Matemática (1998-2016).

- b) **Sistema binário:** base $\beta = 2$ e símbolos $\{0,1\}$, denominados de *bits*, ou *binary digits*. A aritmética binária foi desenvolvida no início do século XVII pelo matemático Leibnitz; em 1814, Boole criou a álgebra booleana; e, em 1935, Shannon a implementou em circuitos.
- c) **Sistema ternário-quântico:** base $\beta = 3$ e símbolos $\{0,1,0^1\}$, em que 0^1 é uma superposição dos dois *bits*, denominados de *qubits*, ou *bit* quântico. Este sistema é usado na moderna computação quântica, que faz uso direto de propriedades da mecânica quântica, ampliando a representação e o processamento.

Denominamos “*notação*” a forma ou o tipo de palavra com que representamos um número X na base β , X_β , usando um sistema posicional.



Neste livro, utilizaremos formato de notação científica com “ponto” em vez de “vírgula”, para facilitar o processo de copiar, colar e testar exemplos e algoritmos, uma vez que as linguagens de programação usam notação com o ponto “.” para separar a parte inteira da parte fracionária dos números, conforme padrão em países de cultura inglesa.

Hoje, usamos quatro tipos de notação:

- a) **Notação convencional** ou **ponto fixo**: nesse caso, representamos um X_β pela “palavra”: $X_\beta = (a_1 a_2 \dots a_k \cdot a_{k+n} \dots a_{k+n+1})_\beta$, em que β é a base, $a_i \in \{0, 1, 2, \dots, \beta - 1\}$, k é o comprimento da parte inteira, e n da parte fracionária do número, com $k, n \in \mathbb{N}$.
- b) **Notação fracionária**: nesse caso, representamos tipicamente um X_β por $X_\beta = (a_1 / a_2)_\beta$, em que $a_2 \neq 0$ e $a_i \in \mathbb{Z}$. Essa representação é limitada a números racionais.
- c) **Notação fatorada**: nesse caso, representamos um X_β por $X_\beta = \sum_{i=1}^k a_i \beta^{k-i} + \sum_{i=k+1}^{n+k} a_i \beta^{k-i}$, em que $a_i \in \{0, 1, 2, \dots, \beta - 1\}$, k é o comprimento da parte inteira, e n da parte fracionária do número, com $k, n \in \mathbb{N}$, quando representado na notação convencional.

Notação em ponto flutuante: nesse caso, representamos um X_β por $X_\beta = (a_1 \cdot a_2 \dots a_t)_\beta \beta^E = (\frac{a_1}{\beta^1} + \frac{a_2}{\beta^2} + \dots + \frac{a_t}{\beta^t}) \beta^E$, em que $a_i \in \{0, 1, 2, \dots, \beta - 1\}$. O grupo de símbolos $(a_1 a_2 \dots a_t)_\beta$ é chamado de mantissa, contém os t *símbolos significativos* da representação e tem comprimento t fixo. Os expoentes E da base β estão situados em um intervalo de números inteiros, $I \leq E \leq S$, e dependem da forma de normalização da mantissa, ou seja, dependem da posição padronizada do ponto (ou vírgula), se antes ou depois do primeiro dígito significativo não nulo. Dessa forma, um sistema de ponto flutuante qualquer pode ser definido pela quádrupla $F(\beta, t, I, S)$.



Símbolos significativos são os não nulos, contados a partir da esquerda até o último não nulo à direita, ou até o último (nulo ou não) indicado (zeros à direita são significativos e zeros à esquerda não o são).

Atualmente, para fins de eficiência computacional, representamos um número na notação em ponto flutuante por: $X_{\beta} = (a_1 \cdot a_2 \dots a_l)_{\beta} \beta^E$ tanto em calculadoras científicas quanto em computadores.

O sistema de representação de quantidades que mais utilizamos é o sistema de numeração decimal.

Ao longo do texto, apresentaremos exemplos com o objetivo de reforçar o conteúdo exposto.

Exemplo 1.1: represente o decimal $X = 309.57$ nas suas notações de ponto fixo, fatorada e ponto flutuante.

Solução:

Considerando a sua notação em ponto fixo:

$$X = (309.57)_{10}$$

Considerando a sua notação fatorada:

$$X = (309.57)_{10} = 3 * 10^2 + 0 * 10^1 + 9 * 10^0 + 5 * 10^{-1} + 7 * 10^{-2}$$

Considerando a sua notação em ponto flutuante:

$$X = (3.0957)_{10} * 10^{+2}$$

Observe que a construção das operações aritméticas deve considerar o valor relativo de cada símbolo numérico como apresentado na sua **notação fatorada**. Assim, quando fazemos qualquer operação aritmética, como a adição, operamos os termos envolvidos considerando a posição do ponto (vírgula) como referência, operando unidade com unidade, dezena com dezena e assim por diante.

Na próxima seção, vamos detalhar os sistemas de numeração e as notações de números mais comuns em computadores.

1.2 SISTEMA DE NUMERAÇÃO DECIMAL ($\beta = 10$)

A aceitação do sistema de numeração decimal como padrão se deve a algumas de suas características, que detalharemos a seguir.

1.2.1 Utiliza apenas dez símbolos

Tais símbolos são representados por 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, conhecidos também por algarismos arábicos ou hindu-arábicos, que são derivados da versão ainda hoje usada no mundo árabe (escrita da direita para a esquerda):

٩, ٨, ٧, ٦, ٥, ٤, ٣, ٢, ١

Historicamente, a simbologia dos dígitos tem diversas hipóteses sobre sua origem e evolução no mundo ocidental, conforme Figura 1.1:

- número de ângulos existentes no desenho de cada algarismo;
- número de traços contidos no desenho de cada algarismo;
- número de pontos de cada algarismo;
- número de diâmetros e arcos de circunferência contidos no desenho de cada algarismo; e
- figuras desenhadas a partir dos traços de um quadrado e suas diagonais.

Figura 1.1 – Origem e evolução dos números hindu-arábicos



1.2.2 Faz uso do zero

O zero foi aceito com certa relutância na origem do sistema decimal, pois era difícil imaginar a quantificação e a representação do nada, do inexistente. Hoje é indispensável, pois é o indicador da ausência de certas potências da base.

Exemplo 1.2: no número decimal

$$(702.4)_{10} = 7 * 10^2 + 0 * 10^1 + 2 * 10^0 + 4 * 10^{-1}$$

a posição correspondente à dezena, ou à potência 10^1 , está ausente.

1.2.3 Adota o princípio da posicionalidade

No sistema posicional, o valor de cada símbolo é relativo, isto é, depende da sua posição no número.

Exemplo 1.3: nos números decimais

$$(348)_{10} = 3 * 10^2 + 4 * 10^1 + 8 * 10^0$$

$$(574)_{10} = 5 * 10^2 + 7 * 10^1 + 4 * 10^0$$

$$(702.4)_{10} = 7 * 10^2 + 0 * 10^1 + 2 * 10^0 + 4 * 10^{-1}$$

O dígito, ou símbolo, ou algarismo 4 representa 4 dezenas, 4 unidades e 4 décimos, respectivamente. Note que nenhum dígito interfere na posição do outro, eles são inteiramente independentes entre si.

Para facilitar a representação física dos números e otimizar as operações aritméticas em computadores, vamos estabelecer outros sistemas de numeração.

1.3 SISTEMA BINÁRIO ($\beta = 2$)

O sistema binário, como vimos, utiliza apenas dois **símbolos**, 0 e 1, também chamados de *bits*, o **zero** e o **princípio da posicionalidade**. Nesse novo sistema de numeração, a correspondência com o decimal será:

Decimal	0	1	2	3	4	5	6	7	8	9	10	...	19	...
Binário	0	1	10	11	100	101	110	111	1000	1001	1010	...	10011	...

Exemplo 1.4: $(10011)_2 = (1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0)_{10} = (19)_{10}$

Observe no **Exemplo 1.4** que a forma fatorada do número binário já está representada na base decimal.

O sistema binário tem sido amplamente utilizado em computadores, por permitir a otimização de recursos físicos, e apresenta vantagens sobre o sistema decimal como:

- a) **Simplicidade de representação física:** bastam dois estados físicos distintos para representar os *bits* da base:
 1 pode ser representado por – , *on*, ... e
 0 pode ser representado por + , *off*, ...
 Assim, ficou mais fácil construir uma máquina que reconheça dois estados físicos do que dez estados distintos necessários para representar os decimais.
- b) **Facilidade na definição de operadores lógicos para operações aritméticas fundamentais.**

Acompanhe no **Exemplo 1.5** a demonstração da vantagem do sistema binário sobre o decimal em relação à forma de implementação digital das operações aritméticas.

Exemplo 1.5: vamos calcular o número de combinações dos dígitos inteiros x e y necessário para obter a operação de adição.

Em base decimal, temos 100 combinações dos possíveis 10 valores de x com 10 valores de y para definir a função adição entre dois dígitos ou algarismos. Podemos reduzir essas 100 combinações a 19 resultados diferentes, observe:

$0 + 0 = 00$
 \vdots
 $0 + 9 = 09$
 $1 + 0 = 01$
 \vdots
 $1 + 9 = 10$
 \vdots
 \vdots
 $9 + 0 = 09$
 \vdots
 $9 + 9 = 18$

Na base binária, necessitamos de apenas quatro *combinações de dígitos binários*⁹ x e y , com basicamente dois resultados diferentes:

$0 + 0 = 0$
 $0 + 1 = 1$
 $1 + 0 = 1$
 $1 + 1 = 10$

Combinações de dígitos binários

Podemos obter os resultados dessas adições binárias usando expressões booleanas entre x e y :

XOR – também conhecida como **disjunção binária exclusiva**, devolve um *bit* 1 sempre que o número de operandos iguais a 1 é **ímpar**, consequentemente devolve um *bit* 0 sempre que o número de *bits* 1 for zero ou dois, e serve para definir o primeiro dígito (à direita); e

AND – também conhecida como **conjunção binária**, devolve um *bit* 1 somente quando **ambos** os operandos sejam 1 e serve para definir o dígito extra, “vai-um” (à esquerda). Fonte: Patterson e Hennessy (2007).

Observe que, na última operação, temos um *bit* 1 extra transportado para o registro imediatamente à esquerda, com uma operação chamada popularmente de “vai-um” (“*carry out*”).

Porém, uma **desvantagem** do sistema binário em relação ao sistema decimal é a necessidade de **registros longos** para armazenar números representando quantidades relativamente pequenas.

Exemplo 1.6: $(597)_{10} = (1001010101)_2$

Observe que, no **Exemplo 1.6**, foi necessário um registro com capacidade de armazenamento de 10 *bits* para representar a grandeza decimal $(597)_{10}$ de apenas 3 dígitos.

1.4 SISTEMA HEXADECIMAL ($\beta = 16$)

No sistema hexadecimal, que também é posicional, os símbolos representativos são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, em que A, B, C, D, E, F representam as quantidades decimais 10, 11, 12, 13, 14, 15, respectivamente.

Exemplo 1.7:

$$(FA9C)_{16} = 15 * 16^3 + 10 * 16^2 + 9 * 16^1 + 12 * 16^0 = (64156)_{10}$$

$$(FA9C)_{16} = (1111\ 1010\ 1001\ 1100)_2$$

O sistema hexadecimal utiliza um número reduzido de símbolos para representar grandes quantidades, por isso é um sistema de numeração interessante para compactação, visualização e armazenamento de dados, ou seja, usa **registros curtos**. Os registros binários internos de uma variável podem ser convertidos de forma direta para hexadecimal, como veremos na próxima seção.

Exemplo 1.8:

$$\begin{array}{ccccc} (11010110)_2 & = & (D6)_{16} & = & (214)_{10} \\ 8 \text{ bits} & & 2 \text{ símbolos} & & 3 \text{ símbolos} \\ & & \text{hexadecimais} & & \text{decimais} \end{array}$$

1.5 CONVERSÕES ENTRE SISTEMAS DE NUMERAÇÃO

Conversões entre bases são necessárias para possibilitar a comunicação homem-computador e vice-versa, no caso dos sistemas decimal e binário, bem como para a compactação e descompactação das representações, nos casos das conversões binário para hexadecimal e vice-versa.

1.5.1 Conversão de base β para base 10

Obtemos diretamente as conversões de uma base β para base 10 escrevendo o número na sua forma fatorada, com fatores representados na base decimal. Essas conversões são utilizadas pelos computadores para **apresentar os resultados** calculados, para a nossa visualização e o nosso uso.

Confira o desenvolvimento dos **Exemplos 1.9** e **1.10** para entender as conversões entre binário e decimal utilizadas nas diversas formas de interação humano-computador, como os comandos de entrada e saída de dados utilizados nas linguagens de programação.

Exemplo 1.9: converta o binário $X = (101.1)_2$ para decimal.

Solução:

$$X = (101.1)_2 = 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1 * 2^{-1} = (5.5)_{10}$$

1.5.2 Conversão de base 10 para base β

As conversões de base 10 para base $\beta = 2$ são utilizadas por computadores para **armazenar os dados**.

Exemplo 1.10: simule a entrada do número $X = (19.03125)_{10}$ em computador convertendo-o para binário.

Solução:

Como X não é um número inteiro, temos que converter separadamente as suas partes, inteira $(19)_{10}$ e fracionária $(0.03125)_{10}$:

- a) Na parte inteira do número, efetuamos a sua divisão inteira pela base β , sucessivamente, e construímos o número convertido tomando o último quociente e os restos da última divisão até o da primeira.

$$(19)_{10} = (?)_2$$

$$\begin{array}{r} 19 \quad \underline{|} 2 \\ 1 \quad 9 \quad \underline{|} 2 \\ \quad 1 \quad 4 \quad \underline{|} 2 \\ \qquad 0 \quad 2 \quad \underline{|} 2 \\ \qquad \qquad 0 \quad \underline{|} 1 \end{array}$$

$$(19)_{10} = (\underline{1}0011)_2$$

$$\text{Verificação: } (\underline{1}0011)_2 = \underline{1} * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = (19)_{10}$$

- b) Na parte fracionária do número: efetuamos a sua multiplicação inteira pela base β , sucessivamente; construímos o número convertido tomando os inteiros resultantes de cada produto; e multiplicamos novamente a parte fracionária do produto anterior até que o produto final seja um inteiro ou atinja a quantidade limite de símbolos na representação.

$$(0.03125)_{10} = (?)_2$$

$$\begin{array}{r} 0.03125 \quad 0.06250 \quad 0.12500 \quad 0.25000 \quad 0.50000 \\ \quad * 2 \quad \quad * 2 \quad \quad * 2 \quad \quad * 2 \quad \quad * 2 \\ \hline 0.06250 \quad 0.12500 \quad 0.25000 \quad 0.50000 \quad 1.00000 \\ \quad 0 \quad \quad 0 \quad \quad 0 \quad \quad 0 \quad \quad 1 \end{array}$$

$$(0.03125)_{10} = (0.00001)_2$$

Verificação:

$$(0.00001)_2 = 0 * 2^{-1} + 0 * 2^{-2} + 0 * 2^{-3} + 0 * 2^{-4} + 0 * 2^{-5} = (0.03125)_{10}$$

Logo, juntando as duas partes, temos $(19.03125)_{10} = (10011.00001)_2$

A seguir, apresentaremos outro exemplo de conversão decimal binária, cujo resultado tem número de *bits* ilimitado.

Exemplo 1.11: converta $X = (0.1)_{10}$ para binário e armazene-o com 10 *bits* significativos.

Solução:

$$\begin{array}{cccccccccc}
 0.1 & 0.2 & 0.4 & 0.8 & 0.6 & 0.2 & 0.4 & 0.8 & 0.6 & 0.2 \\
 \frac{*2}{0.2} & \frac{*2}{0.4} & \frac{*2}{0.8} & \frac{*2}{1.6} & \frac{*2}{1.2} & \frac{*2}{0.4} & \frac{*2}{0.8} & \frac{*2}{1.6} & \frac{*2}{1.2} & \frac{*2}{0.4} \\
 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0
 \end{array}$$

Assim, $(0.1)_{10} = (0.0\underbrace{0011\underbrace{0011\underbrace{0011\underbrace{0011\dots}}_2}_2)_2$ é exato na base binária somente na sua forma de dízima periódica com infinitos *bits*.

Um número com infinitos *bits* não pode ser armazenado, por isso precisamos desprezar parte dos seus *bits* **menos significativos** por um processo chamado **arredondamento**. Veja o tópico Arredondamento decimal na seção **Complementando...** ao final deste capítulo.

Considerando que o registro binário disponível tenha apenas 10 *bits* para armazenar os dígitos significativos do **Exemplo 1.11**, faremos uma extensão direta do arredondamento decimal usual para o sistema binário da seguinte forma:

$$(0.1)_{10} = 2^{-4} * (\underbrace{1.100110011}_{t=10} \underbrace{0011\dots}_{\text{arredondamento}})_2$$

Observe que a parcela a ser arredondada $(0.\underbrace{0011\dots}_2)_2$ é menor do que $(0.5)_{10} = (0.1000\dots)_2$ e deve ser arredondada para “zero” de modo a gerar o menor erro. Logo,

$$(0.1)_{10} = 2^{-4} * (\underbrace{1.100110011}_{t=10} \underbrace{0011\dots}_2) \cong 2^{-4} * (1.100110011)_2$$

$$(0.1)_{10} \cong 2^{-4} * (1.100110011)_2 \cong (0.0999755859375)_{10}$$

Observe agora que $(0.1)_{10}$ era uma fração decimal exata e a convertemos para uma representação binária ilimitada, também exata enquanto for representada por um número de *bits* infinito. Ou seja, nesse exemplo, note que, na conversão de base, descartamos parte dos *bits* da representação binária, por limitação do número de *bits* representáveis, o que gerou um erro de arredondamento.

Perceba nas conversões dos **Exemplos 1.10** e **1.11** que:

$$\text{a) } (0.03125)_{10} = (0.00001)_2$$

$$(0.03125)_{10} = 3125 / 10000 = 1 / 32 = 1/2^5 \text{ (o denominador é uma potência da base 2)}$$

$$\text{b) } (0.1)_{10} = 2^{-4} * \underbrace{(1.100110011)}_{t=10} \underbrace{0011\dots}_2 \cong 2^{-4} * (1.100110011)_2$$

$$(0.1)_{10} = (1 / 10)_2 \text{ (a fração não pode ser expressa por outra equivalente cujo denominador seja uma potência de 2)}$$

Isso se deve ao fato de que um racional $X_\beta = p / q$ será limitado nessa base β se, e somente se, puder ser expresso na forma de $X_\beta = v / \beta^k$.

A seguir, vamos apresentar conversões importantes para entender o mecanismo de visualização de representações de base binária por meio de representações em base hexadecimal.

1.5.3 Conversões diretas entre binário e hexadecimal

Sabemos que cada símbolo hexadecimal corresponde a quatro dígitos binários, pois $16^1 = 2^4$. Por exemplo, $(F)_{16} = (1111)_2$ é armazenado em 4 *bits*. Então, fazemos a conversão direta associando a cada hexadecimal quatro dígitos binários. Ou seja, agrupamos os dígitos binários em grupos de quatro a partir da posição do ponto (vírgula). Caso seja necessário, completamos o grupo de quatro *bits* com zeros não significativos.

Exemplo 1.12: converta $(A1.B)_{16} = (?)_2$

Solução:

Como

$$(A)_{16} = (1010)_2$$

$$(1)_{16} = (0001)_2$$

$$(B)_{16} = (1011)_2$$

teremos:

$$(A1.B)_{16} = (1010\ 0001 \cdot 1011)_2$$

Exemplo 1.13: converta diretamente o registro de 16 *bits* $(1000101001101110)_2$ para hexadecimal.

Solução:

Agrupando os *bits* em grupos de quatro, temos:

$$\left(\underbrace{1000}_8 \underbrace{1010}_A \underbrace{0110}_6 \underbrace{1110}_E \right)_2 = (8A6E)_{16}$$

Exemplo 1.14: demonstre a mesma conversão do **Exemplo 1.13** através da conversão por parcelas decimais.

Solução:

Observe que

$$(1000101001101110)_2 =$$

$$(1 * 2^{15} + 0 * 2^{14} + 0 * 2^{13} + 0 * 2^{12} + 1 * 2^{11} + 0 * 2^{10} + 1 * 2^9 + 0 * 2^8 + 0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0)_{10}$$

fatorando cada grupo de quatro *bits*, em potências de $16 = 2^4$, conforme segue:

$$\begin{aligned}
 &= \underbrace{(1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0)}_8 (2^4)^3 + \underbrace{(1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)}_{10} (2^4)^2 \\
 &+ \underbrace{(0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)}_6 (2^4)^1 + \underbrace{(1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)}_{14} (2^4)^0 \\
 &= 8 \cdot (16)^3 + A \cdot (16)^2 + 6 \cdot (16)^1 + E \cdot (16)^0 = (8A6E)_{16}
 \end{aligned}$$

Nas conversões diretas entre as bases binária e hexadecimal não há perda de dígitos significativos (não há arredondamentos).

1.6 REPRESENTAÇÃO DIGITAL DE NÚMEROS

Conforme vimos na seção 1.1, nos sistemas computacionais, representamos um número na forma de notação em **ponto flutuante**, de maneira a racionalizar o armazenamento digital, pois, se utilizássemos um armazenamento em **ponto fixo** (ou vírgula fixa), seria necessário um número de posições (dígitos) no mínimo igual à variação dos limites dos expoentes.

Por exemplo, para obter a representação de uma calculadora científica comum, com menor positivo $mp = 1.0 \cdot 10^{-99}$ e maior positivo $MP = 9.999999999 \cdot 10^{+99}$, seriam necessárias:

- a) 99 posições para a parte fracionária, entre $1.0 \cdot 10^{-99}$ e 1, pois $1.0 \cdot 10^{-99} = \underbrace{0.000 \dots 00001}_{99}$ tem 99 dígitos depois do ponto (vírgula).
- b) 100 posições para a parte inteira, entre 1 e $9.999999999 \cdot 10^{+99}$, pois $9.999999999 \cdot 10^{+99} = \underbrace{9999999999000 \dots 0000}_{100}$ tem 100 dígitos.
- c) Mais uma posição para o sinal (definido por s), totalizando 200 posições no *display* de uma hipotética calculadora científica “de ponto fixo”, conforme segue:

s		
-----	--	--

100 posições para a
parte inteira

99 posições para a
parte fracionária

Por isso, essas representações de ponto fixo ficaram limitadas às calculadoras para operações básicas.

Por outro lado, uma representação em ponto flutuante (ou vírgula flutuante), como a das calculadoras científicas comuns, funciona com pouco mais de 10 dígitos⁹ e com as posições reservadas ao expoente e aos sinais.

Então, uma representação em ponto flutuante é aquela em que o ponto se desloca entre os dígitos significativos da mantissa, segundo certo padrão de normalização para a sua localização, se antes ou depois do primeiro dígito significativo, conforme vimos na seção 1.1.



Normalmente apresentam 10 dígitos decimais na tela e mais alguns dígitos internos extras, também chamados dígitos de "guarda", usados para ampliar a faixa de abrangência das operações aritméticas.

Exemplo 1.15: represente os números em notação científica ou ponto flutuante e sua forma fatorada:

Solução:

$$\begin{aligned} \text{a) } (+0.0003501)_{10} &= +3.501 \cdot 10^{-4} \\ &= +(3 \cdot 10^0 + 5 \cdot 10^{-1} + 0 \cdot 10^{-2} + 1 \cdot 10^{-3}) \cdot 10^{-4} \end{aligned}$$

$$\begin{aligned} \text{b) } (+101.011)_2 &= (+1.01011)_2 \cdot 2^2 \\ &= +(1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4} + 1 \cdot 2^{-5}) \cdot 2^2 \end{aligned}$$

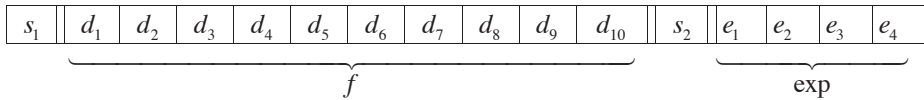
Na próxima seção, vamos exemplificar a representação digital de números em computadores, mostrando os padrões clássicos como registrado na literatura pertinente, a exemplo de Patterson e Hennessy (2007).

1.6.1 Padrão 16 bits original

A representação clássica da variável de 16 bits, utilizada em computadores com arquitetura de 16 bits, não é mais utilizada, mas a detalhamos aqui por motivos didáticos e históricos.

Em um sistema de representação em ponto flutuante e base binária ($\beta = 2$), com $t = 10$ dígitos binários (*bits*) na mantissa e expoentes limitados entre $I = -15$ e $S = +15$, $(15)_{10} = (1111)_2$, simbolicamente temos: $F(\beta, t, I, S) = F(2, 10, -15, +15)_{10}$.

Já para a representação esquemática de dígitos significativos binários, cada *bit* é alocado em um registro, conforme células representadas a seguir:



s_1 = sinal do número (da mantissa f)

f = mantissa com t dígitos significativos

s_2 = sinal do expoente

exp = expoente

Por convenção, temos:

Se $s_1 = 0 \Rightarrow$ número positivo

Se $s_1 = 1 \Rightarrow$ número negativo

$s_2 \Rightarrow$ idem

E, no registro total, temos 16 *bits*:

1 *bit* para o sinal dos dígitos significativos (mantissa), s_1

10 *bits* para armazenar os dígitos significativos da mantissa, f

1 *bit* para o sinal do expoente, s_2

4 *bits* para o módulo do expoente, exp

Nessa representação, os dígitos significativos são armazenados no padrão de normalização com $d_1 \neq 0$ posicionado à direita do ponto. Um número v armazenado nesse registro poderia ser interpretado por:

$$v = (-1)^{s_1} (0.f)_\beta 2^{\pm \text{exp}}$$

ou

$$v = (-1)^{s_1} (0.d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10})_\beta 2^{\pm \text{exp}}$$

Exemplo 1.16: vamos representar $-(101.011)_2$ na variável de 16 *bits* estabelecida anteriormente.

Primeiramente, precisamos normalizar a posição do ponto (vírgula), conforme padrão adotado para 16 *bits*: $-(101.011)_2 = -(0.1010110000)_2 * 2^{+3}$ (com 1º significativo d_1 à direita do ponto). Logo, sinal $s_1 = 1$, mantissa

$f = (1010110000)_2$, convertendo o expoente para binário: $\text{exp} = +(3)_{10} = +(0011)_2$, com sinal $s_2 = 0$, temos então o registro de 16 bits:

1	1	0	1	0	1	1	0	0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

A seguir, apresentaremos os limites da representação de números reais em ponto flutuante no padrão 16 bits apresentado.

1.6.1.1 Menor positivo representável (mp)

0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

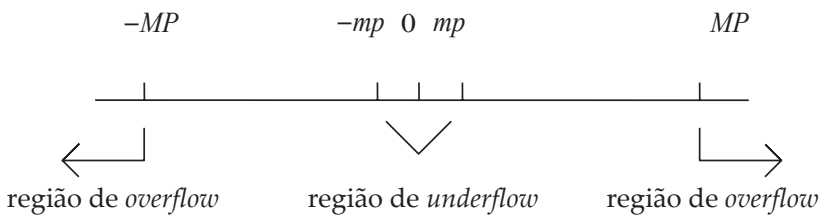
$$mp = +(0.1)_2 * 2^{-15} = (2^{-1} * 2^{-15})_{10} = (2^{-16})_{10} = (0.0000152587890625)_{10}$$


1.6.1.2 Maior positivo representável (MP)

0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$MP = +(0.1111111111)_2 * 2^{15} = (2^{-1} + 2^{-2} + 2^{-3} + \dots + 2^{-10}) * 2^{15} = (32736)_{10} \cong 2^{15}$$

Na reta real, temos a seguinte *representação*⁹ para $F(2, 10, -15, +15)$:



 Os limites de representação dos números negativos são simétricos aos limites dos positivos.

- a) Região de *underflow*: $\{x \in \mathbb{R} / -mp < x < mp\}$, que compreende os números, em módulo, abaixo do mínimo representável e, caso surjam, são arredondados para o mais próximo entre $-mp$, 0 , $+mp$.

- b) Região de *overflow*: $\{x \in \mathbb{R} / x < -MP \text{ e } x > MP\}$, que compreende os números, em módulo, acima do máximo armazenável e não são armazenados (normalmente travam o computador com alguma mensagem de erro).

Exemplo 1.17: se uma operação aritmética gerasse o número $(0.00001527)_{10}$, como poderíamos representá-lo em $F(2, 10, -15, +15)_{10}$?

Solução:

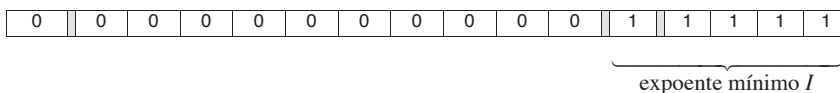
Como esse valor não tem representação binária exata, nós o representaríamos pelo valor discreto mais próximo, no caso $(0.0000152587890625)_{10}$, que é o próprio mp.

1.6.1.3 Representação do zero

A representação do zero é obtida sempre com mantissa nula e o menor expoente representável (I).

Exemplo 1.18: vamos representar o zero em $F(2, 10, -15, +15)$.

Solução:



Devemos lembrar que esse número é o único escrito na forma “não normalizada”, pois sua mantissa é toda zero.

Nos próximos exemplos, veremos o que poderia acontecer em uma operação de adição se o expoente do zero fosse diferente do limite inferior I .

Exemplo 1.19: simule a operação de adição: $0.000135 + 0.0$ em $F(10, 4, -10, +10)$, considerando a representação do zero com expoente nulo ($0.0000 * 10^0$).

Solução:

$$\begin{array}{r}
 0.1350 * 10^{-3} \\
 +0.0000 * 10^0 \\
 \hline
 \end{array}$$

Toda operação de adição/subtração de dois termos é efetuada com os seus dois expoentes iguais ao do maior valor (alinhamento de expoentes):

$$\begin{array}{r} 0.000135 * 10^0 \\ +0.0000 \quad * 10^0 \\ \hline \cong 0.0001 \quad * 10^0 = 0.1000 * 10^{-3} \end{array}$$

Nesse exemplo, ocorre arredondamento com perda dos 2 dígitos significativos excedentes, 3 e 5, que não cabem dentro dos $t = 4$ registros da mantissa. Em calculadoras científicas, esses dígitos excedentes normalmente ficam guardados como dígitos internos de “guarda”.

Exemplo 1.20: simule esta operação de adição: $0.000135 + 0.0$ em $F(10, 4, -10, +10)$, considerando a representação do zero com expoente mínimo $I(0.0000 * 10^{-10})$.

Solução:

$$\begin{array}{r} 0.1350 * 10^{-3} \\ +0.0000 * 10^{-10} \\ \hline \end{array}$$

e alinhamento de expoentes pelo maior valor:

$$\begin{array}{r} 0.1350 * 10^{-3} \\ +0.0000 * 10^{-3} \\ \hline = 0.1350 * 10^{-3} \end{array}$$

No **Exemplo 1.20**, temos um resultado exato, sem arredondamentos, ou seja, o zero representado pelo menor expoente possível retrata corretamente o elemento neutro da operação de adição em meio digital, ou seja, não altera outro número.

A seguir, apresentaremos a quantificação dos elementos representáveis em notação de ponto flutuante.

1.6.1.4 Quantidade máxima de elementos representáveis

Podemos notar que a distribuição de números representáveis em ponto flutuante é padronizada por meio de **tipos de variáveis** e é sempre **discreta** (somente alguns valores são representáveis), enquanto a distribuição de valores da reta Real é **contínua** (qualquer valor é representável).

Exemplo 1.21: represente os dois primeiros números positivos do sistema $F(2, 10, -15, +15)$ de 16 bits.

Solução:

1º positivo -

0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$+ (0.1000000000)_2 * 2^{-15} = (0.0000152587890625)_{10}$$

2º positivo -

0	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$$+ (0.1000000001)_2 * 2^{-15} = (0.0000152885913848876953125)_{10}$$

Observe que existe um intervalo vazio entre esses dois números consecutivos discretos, enquanto na reta Real podem existir infinitos números entre dois números quaisquer. Além disso, a distribuição de números representáveis de $F(\beta, t, I, S)$ não é uniforme em \mathbb{R} , e o número máximo de elementos representáveis é uma combinação das possibilidades de preenchimento de cada registro.

Para cada expoente, existe uma quantidade fixa de números representáveis na mantissa NM , conforme a combinação de possibilidades de seus dígitos, por exemplo, em um sistema genérico de base β , com normalização $d_1 \neq 0$ depois do ponto (vírgula):

- no 1º registro, não podemos ter o número 0 (normalização com $d_1 \neq 0$), logo d_1 pode assumir valores a partir de 1, totalizando $(\beta - 1)$ possibilidades; e
- do 2º registro até o valor da posição t , podemos ter representados qualquer um dos β valores da base, para cada um dos $t - 1$ registros restantes, ou seja, β^{t-1} possibilidades; logo, $NM = (\beta - 1) \beta^{t-1}$.

Para cada mantissa, existe uma quantidade fixa de expoentes representáveis NE , incluindo S , I e o expoente nulo:

$$NE = S - I + 1$$

Dessa forma, o número total de elementos positivos representáveis NP em um sistema de numeração genérico $F(\beta, t, I, S)$ é dado por:

$$NP(\beta, t, I, S) = NM * NE = (S - I + 1)(\beta - 1) \beta^{t-1}$$

Ao dobrar esse número para incluir os negativos e mais um representável para o zero, temos o número total de elementos representáveis NR :

$$NR(\beta, t, I, S) = 2 (S - I + 1)(\beta - 1) \beta^{t-1} + 1$$

Exemplo 1.22: em $F(2, 3, -1, +2)$, temos as seguintes representações possíveis:

a) mantissas possíveis:

0.100
0.101
0.110
0.111

b) expoentes possíveis:

2^{-1}
 2^0
 2^{+1}
 2^{+2}

A combinação de **quatro possibilidades de mantissas** para cada expoente da base ($NM = (\beta - 1) \beta^{t-1} = 4$ para $\beta = 2$ e $t = 3$) com **quatro possibilidades de expoentes** ($NE = S - I + 1 = 4$ para $S = 2$ e $I = -1$) definem o número total de positivos representáveis ($NP(\beta, t, I, S) = NM * NE = 16$).

Do mesmo modo, incluímos as representações dos negativos e do zero, dobrando esse número e somando 1; logo, $NR = 33$.

Exemplo 1.23: em $F(2, 10, -15, +15)$ (binário de 16 *bits* totais), temos: $NP = 2 * (2 - 1) * (15 - (-15) + 1) * 2^{10-1} = 31745$ elementos representáveis, incluindo os positivos, os negativos e o zero.

Exemplo 1.24: em $F(10, 10, -99, +99)$ (calculadora científica comum de 10 decimais, com normalização $d_1 \neq 0$ antes do ponto (vírgula)), temos:

s_1	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	s_2	e_1	e_2
-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	----------	-------	-------	-------

Como d_1 deve ser sempre diferente de zero, essa 1ª posição da mantissa tem nove possibilidades em vez de dez (d_1 está entre 1 e 9), enquanto as demais nove posições têm dez possibilidades de dígitos diferentes (d_i está entre 0 e 9), logo, o número de mantissas diferentes é:

$$NM = (9) * 10 * 10 * 10 * 10 * 10 * 10 * 10 * 10 * 10 = 9 * 10^9$$

$$NM = (\beta - 1) \beta^{t-1} = (10 - 1) * 10^{10-1}$$

Já o número de expoentes depende somente dos seus limites:

$$NE = S + I + 1 = +99 - (-99) + 1$$

Logo, $NR = 2 (\beta - 1) \beta^{t-1} (S - I + 1) + 1 = 3.582 * 10^{12}$ elementos.

A representação binária usada no padrão de 16 *bits* evoluiu juntamente com os computadores e atingiu uma forma mais otimizada de representação, incluindo a **polarização** dos expoentes, mais flexibilidade na normalização da mantissa para permitir maiores faixas de representação, mais *precisão e exatidão*⁹, entre outras.

Dessa evolução, em 1985 surgiu o padrão IEEE 754 (Instituto de Engenheiros Eletricistas e Eletrônicos), que é amplamente utilizado para padronização de variáveis (PATTERSON; HENNESSY, 2007). Em 2008, o padrão IEEE 754 referenciou oficialmente também o padrão otimizado de 16 *bits*, denominando-o de *binary16*.



Confira os conceitos de precisão e exatidão na seção **Complementando...** ao final deste capítulo.

1.6.2 Otimizações da variável de 16 *bits*, segundo o padrão IEEE 754

Otimizações para a variável de 16 *bits* foram adotadas, como *binary16*, com o objetivo de ampliar a sua representação, reduzindo as regiões de *underflow* e *overflow*.

A seguir, apresentaremos o detalhamento dessas otimizações.

1.6.2.1 Polarização

O padrão IEEE 754 adotou a **polarização**, ou excesso, para armazenamento dos expoentes, que é um valor acrescentado (em excesso) a todos os expoentes de um sistema de representação em ponto flutuante, com o objetivo de tornar todos os expoentes positivos, suprimindo o sinal + e ampliando o valor do expoente superior (S). Naturalmente, todas as operações aritméticas devem considerar essa polarização introduzida e, no final, subtraí-la para imprimir os resultados.

Exemplo 1.25: na variável de 16 *bits* $F(2, 10, -15, 15)$, podemos usar uma polarização $p = +(15)_{10} = +(1111)_2$



Observe que os expoentes da variável de 16 *bits* ocupam 5 *bits* (destacados no registro em cinza), uma posição para o sinal do expoente e quatro para o seu módulo. Logo, os expoentes polarizados devem continuar cabendo nos cinco registros binários disponíveis:

$$I + p = -(15)_{10} + p = -(1111)_2 + p = -(1111)_2 + (1111)_2 = (00000)_2$$

$$S + p = +(15)_{10} + p = +(1111)_2 + p = +(1111)_2 + (1111)_2 = (\underline{11110})_2$$

5 *bits*

Como I e S polarizados têm agora o mesmo sinal (+), podemos suprimir s_2 e usar todos os cinco registros binários reservados ao expoente. Assim, os limites dos expoentes polarizados assumem os novos valores:

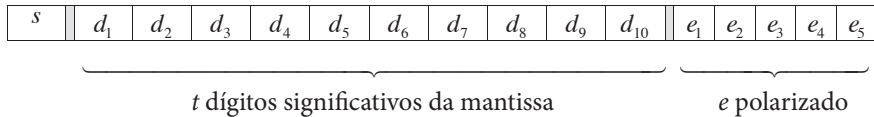
$$I = (00000)_2 = (0)_{10}$$

$$S = (11110)_2 = (30)_{10}$$

Podemos aproveitar melhor os 5 *bits* reservados ao expoente **ampliando** o maior valor possível do **expoente superior** e adotando um novo S como $(11111)_2 = (31)_{10}$. Note que pudemos incluir esse expoente extra porque o expoente zero original (não polarizado) usava duas representações, -0 e $+0$, e uma delas é desnecessária. Agora, o expoente zero polarizado é

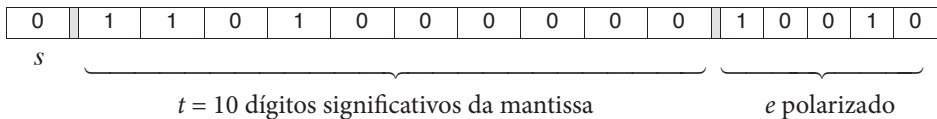
representado apenas pelo seu valor positivo, $+0$, que é representado sem a posição do sinal $+$.

Na forma polarizada dessa variável de 16 *bits*, qualquer número v representado poderá ter esta forma:



Agora s é único, e $v = (-1)^s (0.f)_2 2^{e-15}$.

Exemplo 1.26: no registro binário, a seguir, com exponte polarizado com $p = +(15)_{10}$, qual é o decimal representado?



Solução:

Temos que $v = (-1)^s (0.f)_2 2^{e-15}$

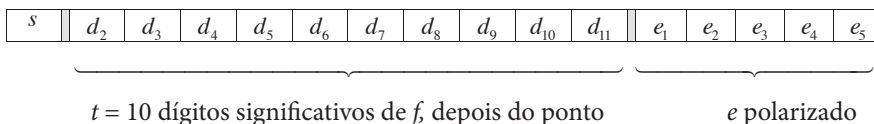
$$s = 0, f = 110100000 \text{ e } e = (10010)_2 = (18)_{10}$$

$$v = (-1)^0 (0.110100000)_2 2^{18-15} = +(110.1)_2 = (6.5)_{10}$$

1.6.2.2 Não armazenamento do primeiro *bit* não nulo da mantissa

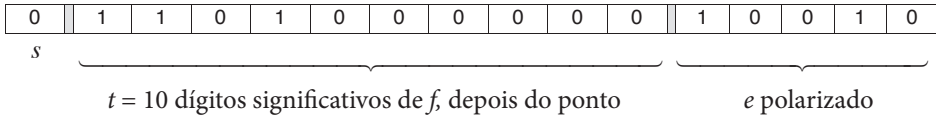
Outra otimização adotada pelo padrão IEEE 754 foi o **não armazenamento do primeiro *bit* da mantissa** $d_1 \neq 0$, que é sempre **unitário**. Nessa otimização, usamos uma representação implícita do primeiro *bit* unitário e liberamos uma posição binária para armazenamento de um novo *bit* significativo. Logo, o número total de dígitos binários significativos é ampliado para $1 + t$, a mantissa f representada passa a conter os *bits* de d_2 à d_{t+1} e a posição normalizada do ponto (vírgula) passa a ser à direita de $d_1 = 1$, ou seja:

$$v = (-1)^s (1.f)_2 2^{e-15}$$



Observe que $d_1 = 1$ está implícito antes do ponto e não aparece nesse registro.

Exemplo 1.28: no registro binário, a seguir, com polarização e com primeiro *bit* implícito, qual é o decimal representado?



Solução:

Temos que $v = (-1)^s (1.f)_2 2^{e-15}$

$$s = 0, f = 110100000 \text{ e } e = (10010)_2 = (18)_{10}$$

$$v = (-1)^0 (1.110100000)_2 * 2^{18-15} = +(1110.1)_2 = (14.5)_{10}$$

Observe que, com os mesmos *bits* representados no **Exemplo 1.26**, podemos agora armazenar números com mais dígitos significativos totais.

Logo, o valor do *MP* foi ampliado para:



$$MP = (1.111111111)_2 * 2^{30-15} = (2^0 + 2^{-1} + 2^{-2} + 2^{-3} + \dots + 2^{-10}) * 2^{15} = (65504)_{10} \cong (2^{16})$$

(*MP* era $(32736)_{10}$ conforme vimos na seção 1.6.1.2)

1.6.2.3 Flexibilidade na normalização da mantissa

O padrão IEEE 754 também adotou a **flexibilidade na normalização da mantissa** para ampliar a faixa de abrangência de números pequenos, como para o primeiro número positivo *mp*, reduzindo a região de *underflow*.

Como vimos anteriormente, o *mp* era



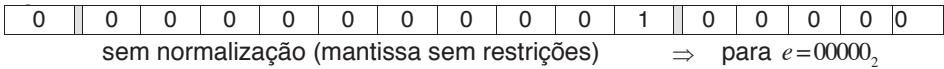
$$mp = +(0.1000000000)_2 * 2^{-15} = (2^{-16}) = (0.0000152587890625)_{10}$$

Com a otimização, que reposicionou a normalização do $d_1 \neq 0$ colocando o ponto à sua direita, deixando $d_1 = 1$ implícito, o novo valor de *mp* seria:



$mp = +(1.0000000000)_2 * 2^{0-15} = (2^{-15})_{10}$, que seria maior ainda do que o anterior.

Essa normalização da mantissa foi flexibilizada especificamente para o caso de representação de números com o menor expoente polarizado possível, ou seja, para $e = (00000)_2$, a **normalização** com $d_1 = 1$ implícito foi **eliminada**. Nesse caso, a mantissa f pode assumir qualquer valor, mas o **valor mínimo do expoente**, não polarizado, deve ser $e = -14_{10}$, e o novo mp se torna:



Logo,

$mp = +(0.0000000001)_2 * 2^{-14} = (2^{-24})_{10} = 5.960464478 * 10^{-8}$
 (mp era $(2^{-16})_{10} = 1.52587890625 * 10^{-5}$, conforme vimos na seção 1.6.1.1)

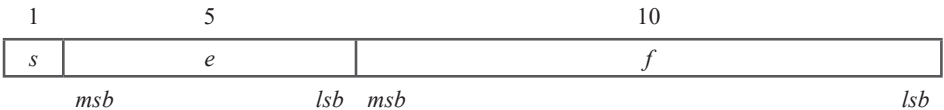
Isso ocorre para manter uma continuidade na representação entre:

- 📍 **Menor representável** com " $e = 1$ ", na faixa $0 < e < 31$, dado por $v = (-1)^s 2^{(1-15)}(1.0000000000)_2$, usando a normalização; e
- Maior representável** com " $e = 0$ ", dado por $v = (-1)^s 2^{(-14)}(0.1111111111)_2$, sem normalização.

1.6.2.4 Identificação da região de *overflow*

Por último, uma convenção adicional foi adotada no padrão IEEE 754 para **identificar a região de *overflow***: a todo número gerado dentro dessa região, atribuímos expoente igual ao seu limite superior: $S = (31)_{10}$.

O novo padrão IEEE 754 otimizado para 16 *bits* (2 *bytes*) ficou nesta forma final:



Em que:

$s = 0 \Rightarrow v$ positivo e $s = 1 \Rightarrow v$ negativo

e = expoente polarizado

f = mantissa fracionária, a partir do segundo *bit* significativo (o primeiro *bit* significativo será sempre unitário e não armazenado neste registro)

polarização = $(15)_{10} = (01111)_2$

msb = *bit* mais significativo

lsb = *bit* menos significativo, de cada e e f

Um número v armazenado nesse registro é interpretado conforme o valor de seu expoente polarizado e da seguinte forma:

Se $0 < e < 31$, então $v = (-1)^s 2^{e-15} (1.f)_2$

Se $e = 0$ e $f \neq 0$, então $v = (-1)^s 2^{-14} (0.f)_2$

Se $e = 0$ e $f = 0$, então $v = (-1)^s 2^{-14} (0.0)_2 = zero$

Se $e = 31 = 2^5 - 1$, então v pertence à região de *overflow*

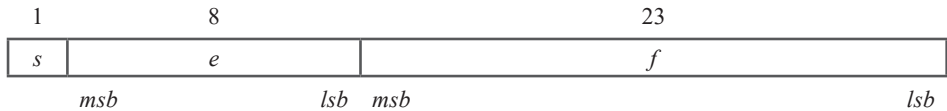
Esse padrão *binary16* com 16 *bits* praticamente não é mais utilizado, pois atualmente temos custos de memória muito reduzidos e a arquitetura comum nos computadores evoluiu para 64 *bits*, o que nos permite usar variáveis de 64 *bits* com a mesma velocidade de acesso de variáveis de 32 ou 16 *bits*.

1.7 REPRESENTAÇÕES NUMÉRICAS, SEGUNDO O PADRÃO IEEE 754

O padrão IEEE 754 de 1985 é amplamente utilizado em linguagens de programação comerciais, e o apresentaremos na sua forma esquemática para representação na forma de variáveis reais.

1.7.1 Variável de 32 bits – *binary32*, tipo *float* do C, precisão simples

No padrão de 4 *bytes* (1 *byte* = 8 *bits*) ou 32 *bits* (precisão de 7 a 8 dígitos decimais significativos equivalentes), podemos representar um número Real v por um registro de 32 *bits*:



que é análogo ao padrão *binary16*, apenas com faixa de valores mais ampliada, como polarização $(127)_{10} = (01111111)_2$, em que um número v armazenado em *binary32* é interpretado conforme o valor de seu expoente polarizado e .

Dessa forma:

Se $0 < e < 255$, então $v = (-1)^s 2^{e-127} (1.f)_2$;

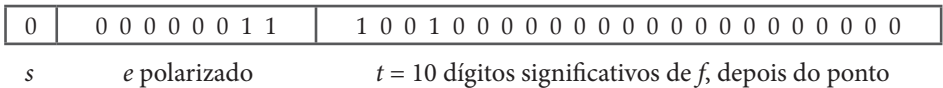
Se $e = 0$ e $f \neq 0$, então $v = (-1)^s 2^{-126} (0.f)_2$;

Se $e = 0$ e $f = 0$, então $v = (-1)^s 2^{-126} (0.0)_2 = \text{zero}$; e

Se $e = 255 = 2^8 - 1$, então v pertence à região de *overflow*.

A apresentação desses registros binários em computadores digitais normalmente é feita em *bytes*. Nesse padrão, temos 4 *bytes*, e cada *byte* é composto de 2 registros hexadecimais (8 *bits* para cada *byte*).

Exemplo 1.28: defina o decimal armazenado em



Solução:

Observe que

$s = 0 \Rightarrow v$ positivo

$e = (00000011)_2 = 1 * 2^1 + 1 * 2^0 = 3_{10}$

$f = (100100000000000000000000)_2$

como $e = 3_{10}$, logo $0 < e < 255$, então $v = (-1)^s 2^{e-127} (1.f)_2$

Assim, o número v armazenado é:

$$v = (-1)^0 2^{3-127} (1.100100000000000000000000)_2$$

$$v = (+) 2^{-124} (1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 0 * 2^{-3} + 1 * 2^{-4} + 0 * 2^{-5} + \dots)_{10}$$

$$v = (+) 4.70197740328915 * 10^{-38} (1 + 0.5 + 0 + 0 + 0.0625 + 0 \dots)_{10}$$

$$v = +7.346839693 * 10^{-38}$$

Exemplo 1.29: represente $(0.1)_{10}$ na base binária no padrão IEEE de 32 *bits*.

Solução:

Para tal, temos duas formas de conversão possíveis.

Primeira forma de conversão: podemos converter $(0.1)_{10}$ diretamente para binário, conforme efetuamos no **Exemplo 1.11**:

$$(0.1)_{10} = (0.00011001100110011001100110011\dots)_2$$

e adequar o resultado diretamente ao padrão 32 *bits*, obtendo s, e, f .

Primeiramente, normalizamos a mantissa, colocando o 1º dígito significativo (mais significativo), unitário, antes do ponto (vírgula), na forma $(1.f)_2$, para obter o expoente não polarizado original ($e-127$), supondo que e esteja na faixa: $0 < e < 255$ em que $v = (-1)^s 2^{e-127} (1.f)_2$:

Assim,

$$(0.10)_{10} = 2^{-4} (1.100110011001100110011001100110011\dots)_2$$

O expoente original, -4 , não polarizado, deve ser equivalente ao expoente $(e - 127)$ da fórmula $v = (-1)^s 2^{e-127} (1.f)_2$, também não polarizado, logo: $(e - 127) = -4$

Então, o expoente e polarizado é o expoente original -4 somado à polarização $+127$:

$$e = -4 + 127 = 123$$

Como $e = 123$ está no *intervalo* $0 < e < 255$, então confirmamos a representação do número $(0.1)_{10}$ pela fórmula prevista $v = (-1)^s 2^{e-127} (1.f)_2$



Caso e estivesse fora da faixa $0 < e < 255$, teríamos que adequar a representação a outro formato do mesmo padrão IEEE 754, com $e = 0$ ou com $e = 255$.

Precisamos ainda definir s e f :

- a) o sinal de $+(0.1)_{10}$ é positivo, logo $s = 0$, então:

$$+(0.1)_{10} = (-1)^0 2^{(123-127)} (1.1001100110011001100110011001100...)_{2^2}$$

- b) a mantissa f , composta dos *bits* depois do 1º *bit* unitário, tem infinitos dígitos significativos (é uma dízima periódica). Por isso, precisamos limitá-la aos 23 *bits* do registro padrão IEEE de 32 *bits* totais. Essa limitação de registros segue os mesmos princípios do arredondamento decimal. Logo, precisamos arredondar a parcela do número binário a partir do 24º *bit*, depois do ponto (vírgula), e gerar o menor erro possível:

$$+(0.1)_{10} = (-1)^0 2^{(123-127)} \underbrace{(1.10011001100110011001100)}_{\text{arredondamento}} \underbrace{11001100...}_{\text{arredondamento}}_{2^2}$$

Então, a parcela que temos de eliminar, $(0.11001100...)_{2^2} * 2^{-23}$, deve ser avaliada segundo os critérios de arredondamento, ou seja, podemos aproximá-la para $(0.)_{2^2} * 2^{-23}$ ou para $(1.)_{2^2} * 2^{-23}$. Como devemos escolher o valor aproximado com menor erro e verificamos que $(0.11001100...)_{2^2} > (0.5)_{10}$, então fazemos o arredondamento $(0.11001100...)_{2^2} \cong (1.)_{2^2}$.

Observe que:

- i) toda fração binária de mais de um *bit* significativo que comece por 1 é maior que $(0.5)_{10}$, e que comece por 0 é menor que $(0.5)_{10}$; e
- ii) se a fração binária for de um “único” *bit* significativo e “unitário”, então a fração binária será equivalente a $(0.5)_{10}$ e deve seguir o critério do *bit* anterior “par ou ímpar”, conforme previsto no processo de arredondamento (*vide* seção **Complementando...**).

Logo, pelos mesmos critérios de minimização de erros, o nosso número fica arredondado para cima, somando 1 no seu último *bit*:

$$+(0.1)_{10} \cong (-1)^0 2^{(123-127)} (1.10011001100110011001100)_{2}^{+1}$$

$$+(0.1)_{10} \cong (-1)^0 2^{(123-127)} (1.1001100110011001100110\underline{1})_{2}$$

Também devemos converter o valor do expoente e para binário:

$(e - 127) = -4 \Rightarrow e = (123)_{10} = (01111011)_2$ (e deve ser completado com “zero(s)” à esquerda para ficar com 8 bits).

Em resumo:

$$s = 0$$

$$e = (123)_{10} = (01111011)_2$$

$$f = (10011001100110011001101)_2$$

0	0	1	1	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

s

e

f

Segunda forma de conversão: alternativamente, podemos converter $+(0.1)_{10}$ diretamente para a fórmula do padrão 32 bits, supondo inicialmente que e esteja na faixa $0 < e < 255$:

$$+(0.1)_{10} = (-1)^s 2^{e-127} (1.f)_2$$

Assim, temos uma equação com três incógnitas, s , e , f , sendo $s = 0$, para números positivos; e e, f , incógnitas que precisamos definir, mas e é um inteiro, então considerando o valor mínimo de f , ou seja, $f = 0$, para poder estimar o valor de e , temos:

$$+(0.1)_{10} = (-1)^0 2^{e-127} (1.0)_2$$

Nesse caso, *determinamos um valor de e maior do que o verdadeiro*, mas e terá a parte inteira correta:

$$(0.1)_{10} = 2^{e-127}$$

$$e = (123.6781\dots)_{10}$$

$f = 0$ representa uma mantissa menor do que a verdadeira, pois $1 \leq (1.f)_2 < 2$.

Logo, tomamos o menor inteiro $e = (123)_{10}$ (“eliminamos” a sua parte fracionária). Verificamos que a suposição inicial para o expoente e na faixa $0 < e < 255$ é válida, ou teríamos que tentar armazenar o nosso número em outra fórmula, com $e = (0)_{10}$ ou com $e = (255)_{10}$.

Agora, determinamos o valor de f , com os valores de s e e obtidos anteriormente:

$$+(0.1)_{10} = (-1)^0 2^{123 - 127} (1.f)_2$$

$$(1.f)_2 = (0.1)_{10} / 2^{123 - 127}$$

$$(1.f)_2 = (1.6)_{10}, \text{ em que } (1.f)_2 \text{ deve ser sempre um número entre 1 e 2}$$

Por fim, convertendo $(1.6)_{10}$ para binário, temos:

$$+(1.6)_{10} = (\underbrace{1.10011001100110011001100}_{\text{Arredondamento}} \underbrace{11001100\dots}_2)_2$$

Com a parcela a ser arredondada $(0.11001100\dots)_2 > (0.5)_{10}$, aproximamos para $(1.)_2$ e a somamos no último *bit*:

+1

$$+(1.6)_{10} \cong (1.10011001100110011001100)_2$$

$$+(1.6)_{10} \cong (1.1001100110011001100110\underline{1})_2$$

Gerando os mesmos resultados da primeira forma de conversão:

0	0	1	1	1	1	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Exemplo 1.30: converta os 32 *bits* do **Exemplo 1.29** do padrão IEEE para o número decimal equivalente.

Solução:

Interpretando os 32 *bits*, podemos converter o registro para decimal, conforme segue:

$$s = 0$$

$$e = (123)_{10} = (01111011)_2$$

$$f = (10011001100110011001101)_2$$

Logo, se $0 < e < 255$, então:

$$v = (-1)^0 2^{123 - 127} (1.10011001100110011001101)_2$$

$$v = (+) 2^{-4} (1 * 2^0 + 1 * 2^{-1} + 0 + 0 + 1 * 2^{-4} + 1 * 2^{-5} + 0 + 0 + 1 * 2^{-8} + 1 * 2^{-9} + 1 * 2^{-12} + 1 * 2^{-13} + 1 * 2^{-16} + 1 * 2^{-17} + 1 * 2^{-20} + 1 * 2^{-21} + 1 * 2^{-23})_2$$

$$v = 0.10000000\underline{1490116} \text{ (com 16 decimais significativos).}$$

Os dígitos sublinhados representam os arredondamentos gerados na representação do número decimal original 0.1 exato para sua representação em binária, no padrão IEEE de 32 *bits*.

Exemplo 1.31: calcule o *erro* de arredondamento cometido na conversão do decimal 0.1 para a representação binária padrão IEEE de 32 *bits* do **Exemplo 1.29**.

Solução:

O *erro* é sempre calculado por meio de uma comparação entre o valor aproximado obtido VA e o seu valor exato VE (quando disponível). Nesse caso, queremos o erro de:

$VA = 0.10000000\underline{1490116}$ (valor armazenado em binário, com erro de arredondamento) e temos $VE = 0.1000000000000000$ (valor exato, disponível na representação decimal).

1.7.2 Variável de 64 bits – *binary64*, tipo *double* do C, precisão dupla

Nesse padrão 8 *bytes* ou 64 *bits* (precisão de 16 a 17 dígitos decimais significativos equivalentes), podemos representar um número Real v por um registro de 64 *bits*:



Em que a polarização $p = (1023)_{10} = (011111111111)_2$.

Um número v armazenado no registro 64 *bits* é interpretado da seguinte forma:

Se $0 < e < 2047$, então $v = (-1)^s 2^{e-1023} (1.f)_2$

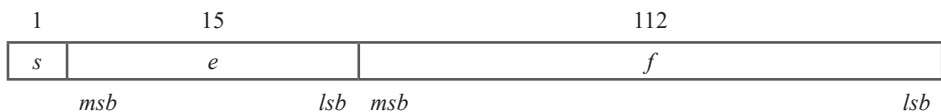
Se $e = 0$ e $f \neq 0$, então $v = (-1)^s 2^{-1022} (0.f)_2$

Se $e = 0$ e $f = 0$, então $v = (-1)^s 2^{-1022} (0.0)_2 = \text{zero}$

Se $e = 2047 = 2^{11} - 1$, então v pertence à região de *overflow*

1.7.3 Variável de 128 bits – *binary128*, precisão quádrupla

No padrão 16 *bytes*, ou 128 *bits* (precisão de 34 a 35 dígitos decimais significativos equivalentes), podemos representar um número Real v por um registro de 128 *bits*:



Em que a polarização $p = (16383)_{10} = (0111111111111111)_2$.

Um número v armazenado no registro 128 *bits* é interpretado da seguinte forma:

Se $0 < e < 32767$, então $v = (-1)^s 2^{e-16383} (1.f)_2$

Se $e = 0$ e $f \neq 0$, então $v = (-1)^s 2^{-16382} (0.f)_2$

Se $e = 0$ e $f = 0$, então $v = (-1)^s 2^{-16382} (0.0)_2 = \text{zero}$

Se $e = 32767 = 2^{15} - 1$, então v pertence à região de *overflow*

1.8 REPRESENTAÇÃO DE VARIÁVEIS INTEIRAS: PADRÃO ANSI C

Os formatos de representação de variáveis do tipo inteiras podem seguir diversos padrões e tamanhos. Vamos apresentar um dos tipos para mostrar como a representação de inteiros também pode gerar erros numéricos.

1.8.1 *Short int*

Short int são tipos inteiros limitados à faixa entre -32768 e $+32767$, correspondendo ao armazenamento como 2 bytes (16 bits), na qual os inteiros “negativos” são armazenados em forma de complemento de dois.

Exemplo 1.35: defina os limites da variável *short int*.

Solução:


$$\begin{array}{lclclcl}
 \text{Zero}^{\circ} & 0_{10} & = & (0000\ 0000\ 0000\ 0000)_2 & = & (0000)_{16} \\
 \text{Maior positivo} & +32767_{10} & = & (0111\ 1111\ 1111\ 1111)_2 & = & (7FFF)_{16} \\
 \text{Menor negativo} & -32768_{10} & = & -(1000\ 0000\ 0000\ 0000)_2 & &
 \end{array}$$



Observe que o 1ª bit “zero”, indica um número positivo.

Observe que esse menor negativo não será armazenado com o sinal “-” e sim por complemento de dois. Todos os inteiros negativos *short int* são representados por complemento de dois, que é o seu complemento de um (diferença de cada dígito do número negativo em relação a um) somado à unidade. Então, o menor negativo será armazenado como:

$$\begin{array}{r}
 \text{Menor negativo } -32768_{10} \quad -(1000\ 0000\ 0000\ 0000)_2 \\
 = \\
 \quad \quad \quad (0111\ 1111\ 1111\ 1111)_2 \rightarrow \text{complemento de um} \\
 \quad \quad \quad \quad \quad \quad +1 \rightarrow \text{soma 1} \\
 \hline
 \quad \quad \quad (1000\ 0000\ 0000\ 0000)_2 \rightarrow \text{complemento de dois}
 \end{array}$$

 Observe que o 1º bit é “um”, o que indica um número negativo (representação sem sinal).

Logo, o menor negativo é representado por:

$$-32768_{10} = (1000\ 0000\ 0000\ 0000)_2 = (8000)_{16} \text{ (sem sinal “-”)}$$

Agora observe o que acontece quando executamos a soma de uma “unidade” com o “maior positivo” da variável *short int*, 32767_{10} , ou seja, quando calculamos um número na sua região de *overflow*:

$$\begin{array}{r}
 \text{Unidade} \quad \quad \quad 1_{10} = (0000\ 0000\ 0000\ 0001)_2 = (0001)_{16} \\
 +\text{Maior positivo} \quad \quad +32767_{10} = (0111\ 1111\ 1111\ 1111)_2 = (7FFF)_{16} \\
 \hline
 \quad \quad \quad 1_{10} + 32767_{10} = (1000\ 0000\ 0000\ 0000)_2 = (8000)_{16}
 \end{array}$$

Esse resultado armazenado em *short int* é interpretado como um *complemento de dois* (por iniciar com 1), portanto é um número negativo.

O complemento de dois

Na aritmética de ponto flutuante, um número iniciado pela unidade é um negativo armazenado como complemento de dois, que precisa ser convertido de volta ao seu valor negativo original, fazendo um complemento de dois. Então, vamos interpretar qual é o verdadeiro número armazenado por $1_{10} + 32767_{10} = (1000\ 0000\ 0000\ 0000)_2 = (8000)_{16}$:

$$\begin{array}{r}
 (1000\ 0000\ 0000\ 0000)_2 \\
 (0111\ 1111\ 1111\ 1111)_2 \rightarrow \text{complemento de um} \\
 \quad \quad \quad \quad \quad \quad +1 \rightarrow \text{soma 1} \\
 \hline
 -(1000\ 0000\ 0000\ 0000)_2 = -(8000)_{16} = -(32768)_{10}
 \end{array}$$

Logo, $1_{10} + 32767_{10} = -(32768)_{10}$

Atenção! Cuidado com as operações com inteiros, pois podemos achar que $1_{10} + 32767_{10} = (32768)_{10}$, mas estamos armazenando, na verdade, o primeiro número negativo -32768_{10} . Logo, **não podemos ultrapassar os limites dos tipos inteiros**, porque os resultados obtidos trocam de sinal e normalmente não existem avisos aos usuários, ou seja, podemos executar algoritmos com números inconsistentes sem saber disso.

Existem outras formas derivadas da variável *short int* de 16 bits, como *long int*, com 32 bits, e *long long int*, com 64 bits.

1.9 TIPOS DE ERROS EXISTENTES EM REPRESENTAÇÕES DIGITAIS

É muito importante conhecer as possibilidades de erros das representações numéricas digitais executadas em calculadoras ou computadores e entender as suas causas, para poder estabelecer a confiabilidade de um algoritmo e de seus resultados.

A seguir, confira os principais tipos de erros que podem existir em representações numéricas digitais.

1.9.1 Erros inerentes

Erros inerentes são aqueles existentes nos **dados de entrada** de um algoritmo numérico. Decorrem, por exemplo, de medições experimentais, de simulações numéricas etc.

1.9.2 Erros de arredondamento

Os erros de arredondamento ocorrem quando são desprezados os dígitos menos significativos, que não são fisicamente confiáveis na representação numérica, ou estão além da capacidade de armazenamento.

1.9.2.1 Arredondamento manual

Como já vimos, arredondamento é o processo de eliminação de dígitos menos significativos de um número. O objetivo é sempre representar o número com menor quantidade de dígitos significativos totais e ainda minimizar os erros decorrentes dessa redução de significativos, conforme tópico sobre arredondamento decimal na seção **Complementando...** ao final deste capítulo.

Exemplo 1.36: represente os números listados, a seguir, com quatro dígitos significativos:

$69.348 \cong 69.35 \rightarrow$ parcela descartada $0.\underline{8}$ é maior que $0.5 \rightarrow +1$ no dígito anterior.
 $69.344\underline{33} \cong 69.34 \rightarrow$ parcela descartada $0.\underline{433}$ é menor que $0.500 \rightarrow$ dígito anterior inalterado.
 $69.33\underline{5} \cong 69.34 \rightarrow$ parcela descartada é $0.\underline{5}$ e dígito anterior ímpar $\rightarrow +1$ no dígito anterior.
 $69.34\underline{5} \cong 69.34 \rightarrow$ parcela descartada é $0.\underline{5}$ e dígito anterior par \rightarrow dígito anterior inalterado.

No **Exemplo 1.36**, realizamos o arredondamento de forma ponderada, baseado em critérios de minimização de erro gerado. Mas também podemos realizar o arredondamento por cancelamento puro: quando a parte indesejada do número é simplesmente cancelada, independentemente do seu valor, assumindo um erro de arredondamento global maior.

1.9.2.2 Arredondamento digital

O arredondamento de representações digitais, em calculadoras ou computadores, acontece por limitação na capacidade de armazenamento, especialmente em: Racionais ilimitados, Irracionais, na mudança de base e em operações aritméticas.

A tradicional aritmética de ponto flutuante é aplicada a problemas de natureza contínua. Devido às limitações de representação dos números no computador, precisamos utilizar o arredondamento para representá-los de forma discreta no mundo digital.

A seguir, vamos apresentar alguns exemplos de armazenamento digital.

1.9.2.2.1 Armazenamento de Racionais ilimitados

Exemplo 1.37: represente a fração $(1/3)_{10}$ em $F(10, 10, -99, +99)$ (normalização usual: ponto (vírgula) depois do 1º dígito significativo).

Solução:

$$(1/3)_{10} = (\underbrace{3.333333333333\dots}_{t=10} * 10^{-1})_{10}$$

$$(1/3)_{10} \neq (3.333333333 * 10^{-1})_{10} \rightarrow \text{representação arredondada de } (1/3)_{10}$$

Exemplo 1.38: represente a fração decimal $(1/10)_{10}$ na variável binária de 16 bits $F(2, 10, -15, +15)$ IEEE 754.

Solução:

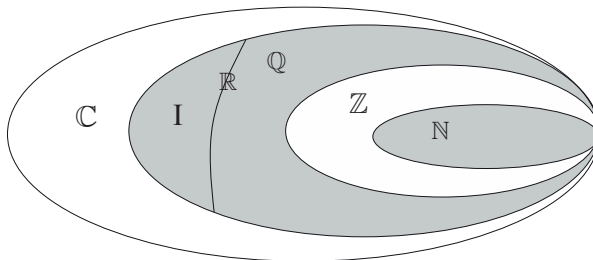
$$(1/10)_{10} = (1/1010)_2 = (1.\underbrace{1001100110011001\dots}_{t=10})_2 * 2^{-4} \rightarrow \text{dízima periódica}$$

$$\text{Logo, } (1/10)_{10} \cong (1.1001100110)_2 * 2^{-4}$$

1.9.2.2.2 Armazenamento de Irracionais

O conjunto dos números Irracionais I compreende todas as representações por meio de dízimas não periódicas e infinitas, o qual, unido aos números Racionais \mathbb{Q} gera o conjunto dos Reais \mathbb{R} ($\mathbb{Q} \cup I$), conforme Figura 1.2.

Figura 1.2: Diagrama dos Conjuntos Numéricos



Fonte: Elaboração própria.

Os Complexos \mathbb{C} generalizam as representações dos números. Um Complexo com parte imaginária nula se torna um Real \mathbb{R} , logo $\mathbb{C} \supset \mathbb{R}$.

Exemplo 1.39: represente em $F(10, 10, -99, +99)$ os seguintes Irracionais.

Solução:

$$\begin{aligned} \text{a) } \pi &= 3.141592653589\dots \\ \pi &= (3.141592653589\dots)_{10} * 10^0 \\ \pi &= (3.141592654)_{10} * 10^0 \end{aligned}$$

$$\begin{aligned} \text{b) } \sqrt{2} &= 1.414213562373\dots \\ \sqrt{2} &= (1.414213562373\dots)_{10} * 10^0 \\ \sqrt{2} &\cong (1.414213562)_{10} * 10^0 \end{aligned}$$

1.9.2.2.3 Armazenamento com mudança de base

Conforme vimos, a representação de números na base binária é amplamente utilizada em computadores, devido às suas vantagens na representação de dados e na implementação de operações aritméticas.

Assim, toda grandeza física, expressa inicialmente em base decimal, é armazenada nos computadores e operada em base binária, por isso são necessárias conversões entre as bases decimal-binária e vice-versa, podendo a primeira ser fonte de erro de arredondamento.

No **Exemplo 1.40**, apresentaremos um caso de erro, na sua forma percentual, do armazenamento com mudança de base do **Exemplo 1.38**.

Exemplo 1.40: calcule o erro de arredondamento percentual ocorrido no armazenamento da fração decimal exata $(0.1)_{10}$ na variável binária de 16 bits $(2, 10, -15, +15)$ IEEE 754.

Solução:

$$(0.1)_{10} = (0.0001100110011\underline{0011}\dots)_2 \quad \rightarrow \text{gerou dízima periódica binária.}$$

$$(0.1)_{10} = 2^{-4} * (1.\underbrace{1001100110}_{t=10}\underbrace{0110011\dots}_{\text{arredondar}})_2 \quad \rightarrow \text{antes do arredondamento.}$$

$$(0.1)_{10} \cong 2^{-4} * (1.1001100110)_2 = (0.099975585)_{10} \quad \rightarrow \text{depois do arredondamento.}$$

Nesse caso, podemos calcular o **erro de arredondamento** em uma calculadora decimal, pois temos o Valor Exato VE original e o Valor Aproximado VA pós-arredondamento, ambos na base decimal:

$$VE = (0.1)_{10}$$

$$VA = 2^{-4} * (1.1001100110)_2 = (0.099975585)_{10}$$

$$\text{Erro Relativo \%} = \left| \frac{VA - VE}{VE} \right| * 100\% = -0,02441\%$$

1.9.2.2.4 Armazenamento de resultados de operações aritméticas

Os operadores lógicos e aritméticos operam números armazenados em variáveis, representadas em ponto flutuante e/ou inteiras, e conseqüentemente ficam com abrangência limitada conforme a notação adotada pela variável utilizada para o armazenamento dos resultados. Vejamos os exemplos a seguir.

Exemplo 1.41: efetue a adição de $a = 0.0165$ e $b = 10.51$ em $F(10, 4, -10, +10)$.

Solução:

Representação em ponto flutuante:

$$a = 1.650 * 10^{-2}$$

$$b = 1.051 * 10^1$$

Agora, vamos implementar a adição de forma simplificada, sempre usando alinhamento pelo maior expoente:

$$\begin{array}{r} a = 0.001\underline{650} * 10^1 \\ +b = 1.051 \quad * 10^1 \\ \hline a + b = 1.052\underline{650} * 10^1 \\ a + b \cong 1.053 \quad * 10^1 \end{array}$$

Observe que o menor número a , quando alinhado pelo maior expoente (de b), gera três dígitos fora da faixa de abrangência de armazenamento, 650, que normalmente são armazenados como dígitos de guarda durante a operação aritmética, mas são arredondados na representação final.

Exemplo 1.42: efetue a adição de $a = (10.01)_2$ e $b = (0.0101)_2$ em $F(2, 4, -15, +15)$.

Solução:

Representação em ponto flutuante:

$$\begin{aligned} a &= (1.001)_2 * 2^1 \\ b &= (1.010)_2 * 2^{-2} \end{aligned}$$

Vamos novamente implementar a adição de forma simplificada:

$$\begin{array}{r} a = (1.001)_2 \quad * 2^1 \\ +b = (0.001010)_2 * 2^1 \\ \hline a + b = (1.010010)_2 * 2^1 \\ a + b \cong (1.010)_2 \quad * 2^1 \end{array}$$

Observe que o menor número b , quando alinhado pelo maior expoente (de a), gera 3 dígitos, 010, fora da faixa de abrangência de armazenamento que acabam sendo arredondados.

Esses fatos ocorrem, geralmente, na soma de números com potências muito diferentes. Nesses casos, o número de menor expoente pode perder dígitos significativos, total ou parcialmente, diante do número de maior expoente. Ou seja, devido à faixa limitada de abrangência dos registradores em ponto flutuante, o número menor perde dígitos significativos quando comparado com o número maior.

Exemplo 1.43: efetue a subtração $a - b$ com $a = 1.351$ e $b = 1.369$ em $F(10, 4, -10, +10)$.

Solução:

Efetuando a subtração de forma simplificada, temos:

$$\begin{aligned} a &= +1.351 \cdot 10^0 \\ -b &= -1.369 \cdot 10^0 \\ \hline a - b &= -0.018 \cdot 10^0 \\ a - b &= -1.800 \cdot 10^{-1} \end{aligned}$$

Note que o resultado final não sofreu arredondamentos, mas também perdeu dígitos significativos, pois as parcelas a e b têm quatro dígitos significativos, e a subtração $a - b$ tem apenas dois. Isso ocorre quando as parcelas subtraídas têm valores próximos.

Exemplo 1.44: reescreva as expressões das soluções de equações de segundo grau de forma a minimizar perdas de dígitos significativos.

Solução:

Para $a \cdot x^2 + b \cdot x + c = 0$, temos que as suas raízes na forma convencional são:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$$

Ou alternativamente, por racionalização do numerador, obtemos:

$$x_{1,2} = \frac{-2 * c}{b \mp \sqrt{b^2 - 4 * a * c}}$$

Observe que as duas formas podem apresentar perdas de significação diferentes quando as parcelas b e $\sqrt{b^2 - 4 * a * c}$ forem de magnitudes próximas e estiverem sujeitas a uma operação de subtração.

Assim, recomendamos que sejam utilizadas, em conjunto, as duas expressões propostas nesse **Exemplo 1.44**, escolhendo o sinal do radicando de acordo com o sinal de b ($sign(b)$), de modo que as parcelas b e $\sqrt{b^2 - 4 * a * c}$ fiquem sujeitas sempre à operação de adição, nas duas fórmulas, usando uma para obter x_1 , e outra para obter x_2 , ou seja:

$$x_1 = \frac{-b - sign(b)\sqrt{b^2 - 4 * a * c}}{2 * a} \quad (\text{Fórmula tradicional})$$

$$x_2 = \frac{-2 * c}{b + sign(b)\sqrt{b^2 - 4 * a * c}} \quad (\text{Fórmula racionalizada})$$

Exemplo 1.45: considere os seguintes sistemas lineares:

$$\text{a) } \begin{cases} x_1 + 3x_2 = 4 \\ x_1 + 3.00001x_2 = 4.00001 \end{cases} \quad \text{solução exata } Sa = \{1, 1\}$$

$$\text{b) } \begin{cases} x_1 + 3x_2 = 4 \\ x_1 + 2.99999x_2 = 4.00002 \end{cases} \quad \text{solução exata } Sb = \{10, -2\}$$

Observe que o sistema de equações (b) pode ser derivado do sistema (a), com uma pequena perturbação em dois de seus coeficientes, devido hipoteticamente a arredondamentos, e que provoque uma variação ao coeficiente a_{22} de 3.00001 para 2.99999 e ao b_2 de 4.00001 para 4.00002, da ordem de 0.025%. Note, ainda, que essa pequena variação acarreta uma variação enorme na solução do sistema (b) da ordem de até 900%.

Denominamos esses sistemas altamente sensíveis a variações nos seus coeficientes de **sistemas mal condicionados**, cuja abordagem trataremos no Capítulo 2.

Exemplo 1.46: efetue a divisão a / b , com $a = 1332$ e $b = 0.9876$, em $F(10, 10, -99, +99)$.

Solução:

$$a / b = 1348.724179829890643\dots$$

Esse resultado está sujeito ao armazenamento em apenas 10 dígitos significativos e precisará ser arredondado:

$$a / b = 1348.724180$$

Dessa forma, o armazenamento limitado causará uma perda de dígitos significativos e gerará um resultado aproximado.

Nas operações de divisão entre duas parcelas quaisquer, normalmente são gerados resultados com um número de dígitos maior do que o limite da representação em ponto flutuante.

1.9.2.3 Consequências dos erros de arredondamento

Vamos conhecer agora as principais consequências dos erros de arredondamento.

1.9.2.3.1 Propagação de erros devido à perda de significação

A perda de dígitos significativos nas operações aritméticas é a principal consequência dos erros de arredondamento, conforme exemplos já vistos anteriormente.

A seguir, apresentaremos um exemplo de propagação de erros devido a perdas de significação **sequenciais**.

Exemplo 1.47: para a função $f(x) = 37500/(25 - x^2)$, obtenha os valores de $f(x)$ em $x = 4.999$ e em $x = 4.9990005$ em uma calculadora científica com representação de 10 dígitos.

Solução:

Calculando parcialmente os termos de $f(x)$, temos para:

$$\begin{aligned} x = 4.999 & \quad \rightarrow \quad x^2 = 24.99000100 & \quad \rightarrow \quad VE = 24.99000100 \\ x = 4.9990005 & \quad \rightarrow \quad x^2 = 24.99000600 & \quad \rightarrow \quad VE = 24.99000599900025 \end{aligned}$$

em que VE é o valor exato.

Observe que, para $x = 4.9990005$, o valor parcial de x^2 sofre perda de dígitos significativos, ficando com apenas 8 dígitos significativos, enquanto o valor exato tem 16 dígitos exatos.

Para:

$$\begin{aligned} x = 4.999 & \quad \rightarrow \quad (25 - x^2) = 9.999 * 10^{-3} & \quad \rightarrow \quad VE = 9.999 * 10^{-3} \\ x = 4.9990005 & \quad \rightarrow \quad (25 - x^2) = 9.994001 * 10^{-3} & \quad \rightarrow \quad VE = 9.99400099975 * 10^{-3} \end{aligned}$$

Observe agora que, para $x = 4.9990005$, o valor parcial de $(25 - x^2)$ também sofre perda de dígitos significativos, ficando com apenas 7 dígitos significativos devido ao arredondamento, enquanto o seu valor exato tem 16 dígitos.

No cálculo de $f(x)$ em uma única instrução, utilizando os dígitos internos de guarda para ampliar a representação (para isso use diretamente os resultados gerados na tela, não os redigite), temos:

$$\begin{aligned} x = 4.999 & \quad \rightarrow \quad f(4.999) & = & 3750375.038 & \rightarrow \quad VE = 3750375.0375 \\ x = 4.9990005 & \rightarrow \quad f(4.9990005) & = & 3752250.975 & \rightarrow \quad VE = 3753350.975455187 \end{aligned}$$

Por fim, observe que, para $x = 4.9990005$, o valor total de $f(x)$ também sofre perda de mais dígitos significativos, ficando com apenas 10 dígitos (limite da calculadora), enquanto o valor exato tem mais de 16 dígitos exatos.

Uma variação de 0.00001% na variável independente x , que poderia vir de um erro inerente do dado de entrada, propaga uma variação na ordem de 0.05% no resultado final de $f(x)$, ou seja, gera uma alteração (erro) no resultado quase 5.000 vezes maior do que a alteração do dado de entrada. Isso caracteriza uma instabilidade intrínseca a esse modelo matemático que deve ser evitada, se o modelo tiver outra forma possível, para minimizar a propagação de erros que sempre vão existir nos dados de entrada.

1.9.2.3.2 Instabilidade numérica nos algoritmos

A acumulação **sucessiva** de erros de arredondamento, em um algoritmo de repetição, pode conduzir a resultados absurdos, por exemplo, a função $f(x) = (n + 1) * x - 1$ é uma constante para $x = 1/n$, pois $f(1/n) = (n + 1) * 1/n - 1 = 1 + 1/n - 1 = 1/n$, mas gera valores absurdos a partir de pequenos arredondamentos iniciais. Verifique essa afirmação sobre a desestabilização de resultados implementando o **Exercício 1.11** em computador.

Na próxima seção, apresentaremos o principal erro dos métodos de aproximação numérica, que são decorrentes de truncamentos de cada tipo de aproximação.

1.9.3 Erros de truncamento

Os erros de truncamento são erros intrínsecos da **aproximação** dos métodos numéricos. Ocorrem sempre que sustamos uma sequência de cálculo infinita, tornando-a finita, por conveniência ou por incapacidade de execução. Assim, ocorrem erros de truncamento, além dos erros de arredondamento, em praticamente todos os métodos e algoritmos do cálculo numérico, conforme veremos nos capítulos subsequentes.

Exemplo 1.48: seja a seguinte expansão em série infinita de Taylor e Maclaurin para a função $\exp(x) = e^x$:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

Sabemos que não é possível usar os infinitos termos dessa série para obter um valor de e^x , então precisamos estabelecer um limite para o número de termos utilizados. Essa limitação no número de termos gera um erro de truncamento na série aproximada que corresponde ao somatório dos termos abandonados.

Podemos representar de forma exata uma função $f(x_0 + x)$ por meio de expansão em série de Taylor convergente, dada genericamente por:

$$f(x_0 + x) = f(x_0) + f'(x_0) \frac{x}{1!} + f''(x_0) \frac{x^2}{2!} + f'''(x_0) \frac{x^3}{3!} + \dots + f^{(n)}(x_0) \frac{x^n}{n!} + \dots$$

Expandindo a função e^x em torno de $x_0 = 0$, temos:

$$f(x_0 = 0) = e^0 = 1$$

$$f'(x_0 = 0) = e^0 = 1$$

$$f''(x_0 = 0) = e^0 = 1$$

$$\vdots$$

$$f^{(n)}(x_0 = 0) = e^0 = 1$$

$$\text{Logo, } f(0 + x) = e^{0+x} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + O(x^{(n+1)})$$

Se aproximamos e^x com apenas n termos, estamos desprezando a parcela infinita de termos a partir do termo $n + 1$,

$$O(x^{(n+1)}) = \frac{x^{n+1}}{(n+1)!} + \frac{x^{n+2}}{(n+2)!} + \dots$$

$$\text{Logo, } e^x \cong 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

Assim, o termo $O(x^{(n+1)})$ caracteriza o **erro de truncamento** da aproximação nesse caso, ou seja, o erro da aproximação em relação ao valor exato.

Lembre-se de que x representa o incremento entre o ponto inicial $x_0 = 0$ e o ponto final $0 + x$.

Observe na Tabela 1.1, a convergência de e^1 pelo valor aproximado $VA^{(n)} \cong 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$, em função de n , comparado com o valor exato $e = 2.718281828459045\dots$

Tabela 1.1 – Convergência de e^1

n	$VA^{(n)}$	$VA^{(n)} - VA^{(n-1)}$	$VE = e$	$Erro\ exato = VA^{(n)} - VE $
0	1.0000000000000000	-	2.718281828459040	1.71828182845904
1	2.0000000000000000	1.0000000000000000	2.718281828459040	0.71828182845904
2	2.5000000000000000	0.5000000000000000	2.718281828459040	0.21828182845904
3	2.6666666666666670	0.1666666666666667	2.718281828459040	0.05161516179237
4	2.7083333333333330	0.0416666666666667	2.718281828459040	0.00994849512571
5	2.7166666666666670	0.0083333333333333	2.718281828459040	0.00161516179237
6	2.7180555555555560	0.0013888888888889	2.718281828459040	0.00022627290348
7	<u>2.718253968253970</u>	0.00019841269841	2.718281828459040	0.00002786020507
8	<u>2.718278769841270</u>	0.00002480158730	2.718281828459040	0.00000305861777
9	2.718281525573190	0.00000275573192	2.718281828459040	0.00000030288585
10	2.718281801146380	0.00000027557319	2.718281828459040	0.00000002731266
11	2.718281826198490	0.00000002505211	2.718281828459040	0.00000000226055
12	2.718281828286170	0.00000000208768	2.718281828459040	0.00000000017287
13	2.718281828446760	0.00000000016059	2.718281828459040	0.00000000001228
14	2.718281828458230	0.00000000001147	2.718281828459040	0.00000000000081
15	2.718281828458990	0.00000000000076	2.718281828459040	0.00000000000005
16	2.718281828459040	0.00000000000005	2.718281828459040	0.00000000000000
17	2.718281828459050	0.00000000000000	2.718281828459040	0.00000000000001
18	2.718281828459050	0.00000000000000	2.718281828459040	0.00000000000001

Fonte: Elaboração própria.

Nesse exemplo, podemos obter o resultado aproximado com 5 dígitos significativos totais da seguinte forma:

- a) Preliminarmente, podemos estimar o número de dígitos confiáveis como aqueles que permanecem constantes entre uma aproximação com $n - 1$ e outra com n parcelas, que pode ser quantificado pelas diferenças em módulo, chamado “critério de parada”, dado por $|VA^{(n)} - VA^{(n-1)}|$. Assim, se quisermos um resultado com 5 dígitos (4 dígitos depois do ponto (vírgula)), poderemos interromper a sequência com $n = 8$ termos, conforme a planilha desse exemplo, cujo critério de parada fica inferior a $1 * 10^{-4}$, no caso $0.2480158730 * 10^{-4}$ (destacado em negrito).
- b) Idealmente, deveríamos avaliar o número de dígitos exatos calculando o erro exato de cada aproximação, conforme $Erro\ exato = |VA^{(n)} - VE|$ apresentado na última coluna da tabela desse exemplo, com VE disponível. Assim, obtemos o mesmo resultado, com 4 dígitos exatos depois do ponto, interrompendo a sequência com $n = 7$ termos, cujo $erro\ exato$ é inferior a $1 * 10^{-4}$, no caso $0.2786020507 * 10^{-4}$ (destacado em negrito), mas raramente o valor exato VE está disponível.
- c) O resultado que obtivemos com o critério de parada em (a) é mais conservativo do que o resultado obtido com o erro exato em (b), ambos com limite $1 * 10^{-4}$, pois em (a) temos resultado mais exato do que em (b). Normalmente, podemos usar os critérios de parada em um processo repetitivo para obtenção de resultados por aproximações numéricas como referência para o erro, mas esse tema será abordado posteriormente para cada método numérico apresentado ao longo deste livro.

Exemplo 1.49: aproximações numéricas de derivadas de funções. Por

definição, $f'(x)$ é dada por, $f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$

Se não pudermos obter esse limite exato, podemos efetuar uma aproximação numérica tomando o incremento h como finito e promovendo refinamentos de resultados em razão do h adotado. Assim, podemos obter uma sequência de aproximações sucessivas de $f'(x)$, com exatidão crescente, usando h cada vez menores, mas não podemos chegar a um h numericamente

nulo ($h \cong O(10^{-16})$ para variáveis do tipo *double*), pois nesse caso, o numerador do limite se torna nulo por perda de significação. Então, também precisamos quebrar o processo matemático de refinamentos sucessivos, assumindo um erro de truncamento de limite tolerável.

Exemplo 1.50: aproximações numéricas para derivada de 2ª ordem, a partir de três pontos vizinhos de $f(x)$, com espaçamento h :

$$\frac{f(x_0 - h)}{x_0 - h} \quad \frac{f(x_0)}{x_0} \quad \frac{f(x_0 + h)}{x_0 + h}$$

Podemos somar a série de Taylor para $f(x_0 + h)$ com a de $f(x_0 - h)$:

$$\begin{aligned} f(x_0 + h) + f(x_0 - h) &= f(x_0) + f'(x_0)\frac{h}{1!} + f''(x_0)\frac{h^2}{2!} + f'''(x_0)\frac{h^3}{3!} + \dots + f^{(n)}(x_0)\frac{h^n}{n!} + \dots \\ &\quad + f(x_0) - f'(x_0)\frac{h}{1!} + f''(x_0)\frac{h^2}{2!} - f'''(x_0)\frac{h^3}{3!} - \dots + f^{(n)}(x_0)\frac{h^n}{n!} + \dots \end{aligned}$$

Resultando em

$$f(x_0 + h) + f(x_0 - h) = 2f(x_0) + 2f''(x_0)\frac{h^2}{2!} + 2f^{(4)}(x_0)\frac{h^4}{4!} + \dots$$

Isolando a segunda derivada, temos:

$$f''(x_0) = (f(x_0 + h) + f(x_0 - h) - 2f(x_0))/h^2 + f^{(4)}(x_0)\frac{h^2}{4!} + \dots$$

$$f''(x_0) = (f(x_0 + h) + f(x_0 - h) - 2f(x_0))/h^2 + O(h^2)$$

O termo de segunda ordem $O(h^2)$ representa o somatório dos infinitos termos truncados decorrentes da aproximação em série e podemos defini-lo como erro de truncamento da aproximação,

$$O(h^2) = f^{(4)}(x_0)\frac{h^2}{4!} + f^{(6)}(x_0)\frac{h^4}{6!} + \dots$$

O erro de truncamento da aproximação $O(h^2)$ pode ser reduzido com h . Então, desprezando esse termo $O(h^2)$, assumimos um erro de truncamento de 2ª ordem e temos uma aproximação para $f''(x)$ dada por

$$f''(x_0) \cong (f(x_0 + h) + f(x_0 - h) - 2f(x_0)) / h^2$$

1.9.4 Medida de erros

Podemos fazer a avaliação de erros numéricos se tivermos a disponibilidade do **valor exato** VE para o resultado desejado. Dessa forma, podemos calcular qualquer erro numérico por meio das formas apresentadas anteriormente usando o valor aproximado obtido VA e o valor exato VE . A representação dos erros numéricos em forma de erro relativo (ou percentual) é mais representativa e usual.

Também podemos obter uma **estimativa** dos erros numéricos, em resultados de algoritmos numéricos que não têm solução exata conhecida, utilizando o próprio algoritmo numérico a fim de obter uma **estimativa de valores exatos**, ou mais próximos do exato, mas precisaremos usar mais recursos computacionais, como precisão dupla, para minimizar os arredondamentos; e mais termos nas séries, mais repetições de cálculos, entre outros, para minimizar os erros de truncamento.

No **Exemplo 1.51**, apresentaremos a estimativa de erro de arredondamento de um valor VA obtido por algoritmo executado com certa precisão limitada. Nesse exemplo, podemos **estimar** um valor mais exato, que denotaremos por VE^e como sendo aquele obtido com o próprio algoritmo numérico operando com **mais precisão**, como **precisão duplicada**, ou seja, reavaliemos o valor obtido numericamente, no mesmo sistema de numeração (normalmente binário), mas agora com precisão superior, e esses novos resultados serão mais exatos.

Exemplo 1.51: estime o erro de arredondamento gerado no armazenamento de $x = (0.1)_{10}$ em uma variável binária de 32 *bits* usando recursos da própria base binária.

Solução:

Se simularmos esse cálculo em linguagem C/C++, temos:

$x = (0.1)_{10}$ armazenado em 32 *bits* (*float*) gera o decimal:

$x = 0.100000001490116$

e

$xx = (0.1)_{10}$ armazenado em 64 *bits* (*double*) gera o decimal:

$xx = 0.10000000000000000055511151$

que também é um valor aproximado, mas tem o dobro de dígitos exatos em relação a x .

Logo, podemos obter o erro de arredondamento estimado de x pela diferença entre $VA = x$ e $VE^e = xx$ (estimativa do valor exato):

$$\begin{aligned} \text{Erro exato estimado} &= |VA - VE^e| \\ &= |0.100000001490116 - 0.10000000000000000055511151| = 0.1490116 * 10^{-8} \end{aligned}$$

Observe que esse erro é calculado na própria base binária, portanto usa valores decimais aproximados e mesmo assim gera um erro estimado compatível com o próprio erro exato.

No **Exemplo 1.52**, apresentaremos a estimativa de erro de truncamento de um valor VA obtido numericamente. Podemos estimar um valor mais exato VE^e como sendo aquele obtido com o próprio algoritmo numérico operando com mais exatidão nas aproximações e mais termos na série aproximadora. Por segurança, devemos estimar o valor mais exato VE^e , em precisão duplicada, visando minimizar os efeitos de erros de arredondamentos sobre os erros de truncamento.

Exemplo 1.52: calcule o erro de truncamento da aproximação $VA^{(n=7)}$ que obtivemos no **Exemplo 1.48** para $n = 7$ parcelas.

Solução:

O valor aproximado $VA^{(n=7)}$ obtido com $n = 7$ é 2.718253968253970, e sabemos que tem 5 dígitos exatos totais (4 decimais depois do ponto).

Podemos simular um valor exato VE^e de referência operando com n maior. Idealmente obtemos o valor exato com $n \rightarrow \infty$, o que normalmente é inviável, então precisamos estimar um número finito mínimo de parcelas, como $2n = 14$ parcelas, gerando o seguinte resultado: 2.718281828458230.

Observe que esse valor VE^e também é aproximado, no caso com $2n = 14$ parcelas, mas contém bem mais dígitos exatos que o $VA^{(n=7)}$ inicial. Esse VE^e tem 12 dígitos exatos totais.

Então,

$$VA^{(n=7)} = 2.718253968253970 \text{ (com } n = 7\text{)}$$

$$VE^e = 2.718281828458230 \text{ (} VE^e \text{ estimado com } 2n = 14\text{)}$$

$$VE = 2.718281828459040 \text{ (valor exato com infinitas parcelas, limitado a 16 dígitos significativos)}$$

Logo,

$$\text{Erro exato} = |VA^{(n)} - VE| = 0.2786020507 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n)} - VE^e| = 0.2786020426 * 10^{-04}$$

Assim, podemos estimar o erro exato usando simulações do valor exato VE^e , mas precisamos otimizar os recursos computacionais necessários para essa simulação. Observe que, se o valor exato estimado VE^e for calculado por VA com menos parcelas (mas $n > 7$), também teremos resultados razoáveis:

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=14)}| = 0.2786020426 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=13)}| = 0.2786019279 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=12)}| = 0.2786003220 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=11)}| = 0.2785794452 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=10)}| = 0.2783289242 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=9)}| = 0.2755731922 * 10^{-04}$$

$$\text{Erro exato estimado} = |VA^{(n=7)} - VA^{(n=8)}| = 0.2480158730 * 10^{-04}$$

Note que todas essas simulações do erro de truncamento de $VA^{(n=7)}$ geraram resultados com a mesma ordem de grandeza, portanto, para essa aproximação em série de MacLaurin, foi suficiente simular o valor exato, VE^e , com apenas $n + 1$ parcelas. Assim, o erro estimado $|VA^{(n=7)} - VA^{(n=8)}| = 0.2480158730 * 10^{-04}$ foi da mesma ordem do erro exato.

Os erros estimados anteriormente foram “todos” inferiores ao critério de parada que utilizamos no **Exemplo 1.48**, para $n = 7$ parcelas, $|VA^{(n)} - VA^{(n-1)}| = 1.9841269841 * 10^{-04}$. Logo, podemos usar o critério de parada como limite superior do erro de truncamento estimado, nesse tipo de aproximação, considerando que, se o critério de parada já é aceito como suficientemente pequeno, o erro exato será ainda menor.

Um cálculo com mais exatidão, para servir de estimativa do valor exato, naturalmente exige mais recursos computacionais, mas deve ser aplicado para validação dos resultados obtidos por algum critério de parada, conforme veremos detalhadamente para cada tipo de aproximação numérica aplicada ao longo deste livro.

Por fim, existem formas alternativas de estimar valores exatos do resultado desejado que minimizam os erros numéricos de arredondamento e/ou truncamento além das presentes no escopo deste livro. São elas:

- a) **simulações numéricas utilizando matemática intervalar**: possibilitam limitar o erro existente a um intervalo aritmético;
- b) **variável de comportamento estatístico**: possibilita prever os limites do erro segundo um tratamento estatístico das variáveis envolvidas;
- c) **simulação do algoritmo em Sistemas de Computação Algébrica (SCA)**: possibilita recorrer a simulações com precisão ilimitada e, em alguns casos especiais, proceder à simulação exata do algoritmo;
- d) **aritmética discreta de corpos finitos**: possibilita que todos os elementos de um corpo tenham uma representação na memória do computador, portanto não há erros de arredondamento, sendo possível utilizá-la especificamente para problemas discretos,

como aqueles encontrados em criptografia, códigos de correção de erros, filtros digitais etc.; e

- e) **representação de números fracionários:** baseada no padrão IEEE chamado de *unum (universal number)*⁹, possibilita uma série de vantagens na sua representação para evitar os arredondamentos, adicionando aos campos *bit* de sinal, expoente e mantissa, já conhecidos, mais três campos (*utag*): um *bit* que marca se o número é exato ou se está entre valores exatos (*ubit*), um campo para o tamanho do expoente e outro para o tamanho da fração, entretanto ainda não está difundida nas linguagens de programação comerciais (GUSTAFSON, 2015).



Saiba mais sobre essa inovadora proposta no livro *The End of Error: unum computing*, de John Gustafson, conhecido pela formulação da Lei de Gustafson.

1.10 CONCLUSÕES

O perfeito entendimento das causas dos erros numéricos, gerados em calculadoras e computadores, é um fator decisivo para que você, como analista numérico, possa fazer a escolha do método computacionalmente mais eficiente e a validação dos resultados. Assim, para resolver um determinado modelo matemático, o analista numérico deve se preocupar com:

- a) a escolha de métodos de resolução com menor número de operações aritméticas envolvidas; e
- b) a escolha de uma linguagem de programação para implementar o algoritmo que represente as variáveis envolvidas com precisão suficientemente grande.

No final de todo o processo, o analista numérico deve ser capaz de:

- a) obter uma solução de custo mínimo, ou seja, com menor demanda de memória utilizada e menor tempo de processamento; e

- b) dimensionar o grau de confiabilidade dos resultados obtidos como solução do modelo matemático, ou seja, apresentar o resultado obtido e o seu erro máximo.

Lembre-se de que obter resultados de algoritmos numéricos é relativamente fácil, pois qualquer algoritmo compilado gera resultados numéricos. Para assegurar que esses resultados são as respostas esperadas, é necessária a **estimativa dos erros numéricos** associada à solução aproximada do modelo matemático.

A estimativa dos erros numéricos de truncamento associada à solução aproximada do modelo matemático será discutida ao longo de cada capítulo deste livro.

COMPLEMENTANDO...

Nesta seção, detalharemos os conceitos de alguns termos utilizados no Capítulo 1.

Arredondamento decimal

O objetivo de um arredondamento é representar o número com menor número de dígitos significativos totais e ainda minimizar os erros decorrentes dessa redução de significativos, seja qual for a base. Vamos detalhar esse conceito através do exemplo a seguir.

Exemplo 1.53: represente os seguintes números decimais com 4 dígitos significativos totais:

a) $32.4374 = 32.43 + 0.74 \cdot 10^{-2}$

Solução:

Aqui precisamos definir o que fazer com a parcela $0.74 \cdot 10^{-2}$, que corresponde aos 2 dígitos menos significativos dessa representação. Podemos aproximá-la para $1.00 \cdot 10^{-2}$ ou para $0.00 \cdot 10^{-2}$, e o critério de escolha deve ser tal que o resultado vai gerar o menor erro no número final arredondado, ou seja,

i) se $0.74 \cdot 10^{-2} \cong 1. \cdot 10^{-2}$, o erro é $0.26 \cdot 10^{-2}$; e

ii) se $0.74 \cdot 10^{-2} \cong 0. \cdot 10^{-2}$, o erro é maior, $0.74 \cdot 10^{-2}$.

Logo, $32.4374 = 32.43 + 0.74 \cdot 10^{-2} \cong 32.43 + 1. \cdot 10^{-2} \cong 32.44$

b) $32.4324 = 32.43 + 0.24 \cdot 10^{-2}$

Solução:

Aqui também precisamos definir o que fazer com a parcela $0.24 \cdot 10^{-2}$, e aproximar $0.24 \cdot 10^{-2} \cong 0. \cdot 10^{-2}$ gerará menor erro.

Logo, $32.4324 = 32.43 + 0.24 \cdot 10^{-2} \cong 32.43 + 0. \cdot 10^{-2} \cong 32.43$

$$c) 32.4350 = 32.43 + 0.50 * 10^{-2}$$

Solução:

Podemos aproximar a parcela $0.50 * 10^{-2}$ para $1.00 * 10^{-2}$ ou para $0.00 * 10^{-2}$, e nesse caso o erro gerado é o mesmo. Então, recorremos a um critério de distribuição dos erros para esse grupo de números que tenham fração 0.50, de modo que alguns números ganhem um pouco (aproximando para $1.00 * 10^{-2}$) e outros percam um pouco (aproximando para $0.00 * 10^{-2}$). Para isso, fazemos uma distribuição estatística, de modo que 50% dos números ganhem um pouco e outros 50% percam um pouco. Isso foi padronizado com critério baseado no dígito anterior à parcela a ser arredondada, que pode ser um número par ou um ímpar, com 50% de chances em cada caso. Assim, para os números com dígito anterior **par**, convencionou-se arredondar para $0.00 * 10^{-2}$, e para números com dígito anterior **ímpar**, convencionou-se arredondar para $1.00 * 10^{-2}$. Nesse exemplo, arredondamos a parcela $0.50 * 10^{-2}$ para $1.00 * 10^{-2}$ (pois o dígito anterior 3 é **ímpar**).

$$\text{Logo, } 32.4350 = 32.43 + 0.50 * 10^{-2} \cong 32.43 + 1. * 10^{-2} \cong 32.44$$

$$d) 32.4450 = 32.44 + 0.50 * 10^{-2}$$

Solução:

Nesse exemplo, arredondamos a parcela $0.50 * 10^{-2}$ para $0.00 * 10^{-2}$ (pois o dígito anterior 4 é **par**).

$$\text{Logo, } 32.4450 = 32.44 + 0.50 * 10^{-2} \cong 32.44 + 0. * 10^{-2} \cong 32.44$$

Precisão

Precisão é um conceito objetivo que estabelece a quantidade de algarismos significativos que representam um número. A precisão de uma representação digital, de uma variável, por exemplo, é definida como o número de dígitos significativos totais da mantissa na base β de armazenamento; e precisão decimal equivalente d baseada nos d significativos decimais equivalentes. Assim, para variáveis de 32 *bits* padrão IEEE 754, com $v = (-1)^s(1.f)_2 2^{e-127}$, temos $t = 23$ *bits* depois do ponto (vírgula), uma precisão binária total de 24 *bits* ($1 + t$) e precisão decimal equivalente de 7 a 8 significativos.

A precisão decimal d equivalente pode ser definida pela equivalência direta da faixa de abrangência entre os dígitos mais e menos significativos em cada base; pela equivalência direta entre os expoentes dos dígitos menos significativos da mantissa em cada base; e pela verificação direta do número de dígitos decimais exatos que podem ser armazenados.

Faixa de abrangência entre os dígitos mais e menos significativos em cada base

Para variáveis de 32 *bits* padrão IEEE 754, temos um total de 24 *bits* de precisão binária, pois existe um dígito implícito antes do ponto (vírgula), $d_1 = 1$.

A mantissa total de dígitos significativos é dada por:

$$v = (d_1.d_2d_3\dots d_{22}d_{23}d_{24})_2, \text{ logo:}$$

- a) o dígito binário mais significativo está antes do ponto (vírgula), $d_1 = 1$, que não é armazenado nos 32 *bits*, e equivale a:
 $Dmsb = (1.000000000000000000000000)_2 = 1 * 2^0 = 1.0$;
- b) o dígito binário menos significativo está no **23º registro armazenado**, d_{24} , pois temos $t = 23$ *bits* depois do ponto (vírgula) e equivale a:
 $Dlsb = (0.000000000000000000000001)_2 = 1 * 2^{-23} = 1.1920928955 * 10^{-7}$

23º registro armazenado



O expoente decimal correspondente ao dígito binário menos significativo é considerado a representação da precisão decimal equivalente da variável no padrão IEEE. No Octave o valor do *bit* menos significativo é armazenado na função *eps*.

Se somarmos esses dois valores, teremos a faixa de abrangência representada pelos dígitos em binário ou em decimal:

$$\begin{array}{ll}
 Dmsb = 1 * 2^0 = 1.0 * 10^0 & \text{– 1º dígito decimal, antes do ponto.} \\
 Dlsb = 1 * 2^{-23} = 1.1920928955 * 10^{-7} & \text{– 7º dígito decimal, depois do ponto.} \\
 Dmsb + Dlsb = (\underline{1.000000000000000000000001})_2 & \text{– 24 dígitos binários totais (sublinhados).} \\
 = (\underline{1.0000001} \ 190928955)_{10} & \text{– 8 dígitos decimais totais (sublinhados).}
 \end{array}$$

O decimal equivalente a d_{10} é $10^{-(d-1)}$ ($d - 1$ é o número de dígitos decimais depois do ponto (vírgula) e representa o dígito menos significativo da calculadora), então existe uma equivalência entre a representação binária de $t + 1$ bits totais e a decimal de d decimais:

$$2^{-((t+1)-1)} = 10^{-(d-1)}$$

$$2^{-t} = 10^{1-d}$$

$$d = 1 + t * \log(2)$$

$$d = 7.92368990 \text{ (para } t = 23 \text{ bits, depois do ponto (vírgula))}$$

Portanto, temos uma precisão decimal equivalente entre $d = 7$ e 8 decimais significativos, ou seja, uma calculadora científica de 8 decimais seria suficiente para armazenar os $1 + t = 24$ bits de forma exata.

Verificação direta do número de dígitos decimais exatos que podem ser armazenados

Trata-se de um teste direto (experimentação numérica), no qual podemos armazenar um decimal exato **conhecido** na variável que se deseja testar a precisão decimal. Assim, se tomarmos os decimais conhecidos, a seguir, e armazenarmos em uma variável de 32 bits padrão IEEE 754, usando um tipo *float* no C, Java, entre outros, teremos:

$$\text{a) } (1/3) \cong 0.\underline{3333333}43267441^* \\ 0.3333333333333333\dots^{**}$$

- * temos 7 decimais exatos
- ** valor exato

$$\text{b) } (1234567890123456789012345) \cong \underline{1.2345679}4679859e + 24^* \\ 1.23456890123456789012345e + 24^{**}$$

- * temos 7 decimais exatos
- ** valor exato

Note que, se classificarmos esses números quanto à sua exatidão, teríamos que: (a) está mais próximo do π exato do que (b); (b) é menos exato do que (c); e (c) é o mais exato de todos.

Normalmente, os resultados mais **exatos** (com menores *erros*) são obtidos com as representações mais **precisas** (com mais dígitos significativos representados).

Você encontrará exercícios atualizados e revisados de todos os capítulos deste livro no **Caderno de Exercícios e Respostas** disponível no *link*: <http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>.

Faça o *download* do **Caderno** e reforce o seu aprendizado realizando os exercícios.

SOLUÇÃO COMPUTACIONAL DE SISTEMAS DE EQUAÇÕES LINEARES

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- executar a implementação computacional de métodos eliminativos para solver sistemas de equações lineares com poucos coeficientes nulos, embora estes tenham problemas de acúmulo de erros de arredondamento, especialmente os sistemas mal condicionados; e
- executar a implementação computacional de métodos iterativos clássicos para solver sistemas de equações lineares com muitos coeficientes nulos.

Vamos estudar neste capítulo a implementação computacional de métodos básicos para *solver* sistemas de equações lineares densos ou esparsos (com muitos coeficientes nulos), de acordo com métodos *clássicos disponíveis na literatura*, objetivando resolver os sistemas que surgem nos modelos matemáticos abordados nos cursos de graduação.



Utilizamos a palavra "solver" em vez de "resolver" com o intuito de lembrá-lo de que devemos escolher o método mais adequado para obter a solução de determinado problema matemático, assegurar a sua qualidade e não apenas reproduzir resultados.



As obras *Análise numérica*, de Burden e Faires, e *Numerical Mathematics and Computing*, de Cheney e Kincaid, são excelentes referências para estudo dirigido.

Um sistema de n equações lineares a n incógnitas é toda expressão do tipo:

$$A * X = B \Rightarrow \begin{cases} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{cases} \Rightarrow \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad (1)$$

em que A é a matriz de coeficientes, X é o vetor de incógnitas, B é o vetor de termos independentes e n é a ordem do sistema. Dessas três notações, a primeira apenas facilita o uso algébrico do sistema linear, a segunda é a forma desenvolvida, e a terceira é a forma matricial que possibilita a sua implementação computacional.

A seguir, um exemplo de sistema de equações lineares com $n = 3$:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ -0.319x_1 + 0.884x_2 + 0.279x_3 = 0 \end{cases} \quad (2)$$

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 \\ 0.421 & 0.784 & -0.207 \\ -0.319 & 0.884 & 0.279 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Podemos também armazenar as n equações de um sistema de equações lineares através de uma estrutura de dados matricial única. Para tal representação matricial do sistema (1), vamos adotar a estrutura em forma de matriz expandida, que consiste no armazenamento da matriz de coeficientes A concatenada com a coluna de termos independentes B , de forma que cada linha dessa matriz expandida armazene uma equação, conforme sistema (3):

$$[A \vdots B] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & \vdots & a_{1,n+1} \\ a_{21} & a_{22} & \cdots & a_{2n} & \vdots & a_{2,n+1} \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} & \vdots & a_{n,n+1} \end{bmatrix} \quad (3)$$

Note na matriz expandida que $a_{i,n+1} = b_i, i = 1, 2, \dots, n$. Nos algoritmos, normalmente a matriz expandida $[A \vdots B]$ será denotada e armazenada como única matriz A .

A solução de um sistema de equações lineares $A * X = B$ é toda matriz coluna S que o satisfaça, isto é, tal que $A * S = B$. Quanto à sua solução S , um sistema de equações lineares pode ser possível e determinado, quando S for único; possível e indeterminado, quando existirem infinitas soluções S ; e impossível, quando não existir uma solução S .

Desconsiderando o *método direto de Cramer*⁹ devido à sua ineficiência, os métodos de solução de sistemas de equações lineares podem ser agrupados em três famílias:

- a) **Métodos eliminativos:** nesta metodologia, transformamos a matriz A do sistema $A * X = B$ em uma matriz mais simples, porém preservando a solução S do sistema original. Isso é sempre possível quando fazemos uso das operações elementares sobre linhas da matriz representativa do sistema. Tais operações são a troca de linhas, a adição de linhas, bem como a multiplicação de uma linha por uma constante não nula. Da álgebra linear sabemos que tais operações não alteram a solução S do sistema, uma vez que uma operação elementar sobre uma linha equivale a uma operação sobre a respectiva equação.
- b) **Métodos iterativos:** nesta metodologia, para solver $A * X = B$, inicialmente atribuímos um candidato $X^{(0)}$ para a solução S e posteriormente geramos uma sequência recursiva de vetores coluna $X^{(k)}$, $k = 1, 2, \dots$, cujo limite, se existir, será a solução S procurada.
- c) **Métodos de otimização:** nesta metodologia, partindo do sistema $A * X = B$, geramos uma função matricial $F(X) = X^T * A * X - 2 * B^T * X$ (em que T indica transposta) e tentamos obter o seu mínimo funcional, que é sempre a matriz $A^{-1} * B$, justamente a solução S do sistema linear.

Devido à complexidade algébrica dos métodos de otimização, que está fora do escopo deste livro, não apresentaremos nenhum método dessa família.



O método de Cramer é um teorema da álgebra linear que fornece a solução de um sistema de equações lineares em termos de $n + 1$ determinantes. Para saber mais, consulte: <https://pt.wikipedia.org/wiki/Regra_de_Cramer>. Acesso em: 26 set. 2016.

2.1 SOLUÇÃO ELIMINATIVA DE $A * X = B$

A solução eliminativa de $A * X = B$ é normalmente utilizada em sistemas lineares com matrizes **densas** (com poucos coeficientes nulos) e de **ordem n de porte médio** (valor relativo ao processador utilizado).

A seguir, vamos apresentar alguns algoritmos de métodos eliminativos mais comuns, avaliar os **erros** que afetam as soluções fornecidas, bem como a sua **complexidade** computacional.

2.1.1 Método de eliminação de Gauss

O **método de eliminação de Gauss** consiste em aplicar um conjunto de operações elementares em um sistema linear com o objetivo de torná-lo um sistema de resolução mais simples. Usando esse conjunto de operações, podemos alterar sua matriz expandida, de modo que a matriz dos coeficientes seja transformada em uma **matriz triangular**, superior ou inferior, visando simplificar o sistema de equações para que a sua solução possa ser obtida diretamente por **substituições retroativas** ou **sucessivas**, respectivamente.

Esse processo de eliminação, também denominado **escalonamento**, é obtido através da aplicação sucessiva de operações elementares sobre linhas na matriz expandida, buscando anular elementos não nulos para torná-la uma matriz triangular. Podemos associá-lo a um processo de pivotamento parcial ou total que promove a troca seletiva de linhas (ou colunas) visando tomar pivôs (elementos das diagonais principais) com maior módulo possível, para evitar a presença de pivôs nulos e minimizar a acumulação de erros de arredondamento.

Exemplo 2.1: resolva o sistema de equações lineares, a seguir, pelo método de eliminação de Gauss (sem trocas de linhas e/ou colunas) adotando operações aritméticas com apenas **4 dígitos significativos totais** e arredondamento ponderado, para exemplificar o efeito do acúmulo de arredondamentos.

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ -0.319x_1 + 0.884x_2 + 0.279x_3 = 0 \end{cases}$$

Solução:

Na forma matricial, temos:

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 \\ 0.421 & 0.784 & -0.207 \\ -0.319 & 0.884 & 0.279 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

E, na forma de matriz expandida, temos:

$$A = \begin{bmatrix} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & 0.884 & 0.279 & \vdots & 0 \end{bmatrix}$$

Processo de triangularização

O processo de triangularização deve seguir uma sequência otimizada de passos k (que definem um **algoritmo**). Nesse caso, cada passo k corresponderá à utilização da linha $i = k$ para eliminar os coeficientes da coluna $j = k$ usando operações elementares sobre linhas.

Os índices comumente utilizados para acessar um elemento de uma matriz a_{ij} são: i , para acessar a linha (1º índice); e j , para acessar a coluna (2º índice), enquanto o índice k vai definir os passos do algoritmo. A i -ésima linha é denotada por L_i , e a j -ésima coluna é denotada por C_j .

Vamos acompanhar os passos de resolução do **Exemplo 2.1** para depois estabelecê-los na forma de algoritmo computacional.

Primeiro passo: definido pelo índice $k = 1$, usamos a primeira linha $i = 1$ (em negrito) para eliminar a primeira coluna $j = 1$ das linhas abaixo da linha $i = 1$. Poderíamos estabelecer outra ordem para as operações elementares sobre linhas, mas vamos adotar um algoritmo genérico que independa dos valores dos coeficientes e use o menor número de operações aritméticas possível. Então, nesse primeiro passo, substituímos as linhas $i = 2$ e $i = 3$ pelo resultado da subtração delas próprias com a linha $i = 1$ multiplicada por um fator adequado, de tal modo que as suas colunas $j = 1$ sejam zeradas, logo:

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{0.832} & \mathbf{0.193} & \vdots & \mathbf{1} \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & 0.884 & 0.279 & \vdots & 0 \end{bmatrix} \begin{array}{l} L_2 \leftarrow L_2 - (0.421 / \mathbf{0.448})L_1 \Rightarrow L_2 \leftarrow L_2 - 0.9397L_1 \\ L_3 \leftarrow L_3 - (-0.319 / \mathbf{0.448})L_1 \Rightarrow L_3 \leftarrow L_3 + 0.7121L_1 \end{array}$$

Observações:

a) Na linha L_3 , por exemplo, $L_3 \leftarrow L_3 - (-0.319 / 0.448)L_1$
 $\Rightarrow L_3 \leftarrow +0.7121 * L_1$,

no elemento $i = 3$ e $j = 1$, teremos a seguinte operação de eliminação:

$$a_{31} = a_{31} - (-0.319 / 0.448) a_{11}$$

$$a_{31} = -0.319 - (-0.319 / 0.448) 0.448 = 0 \text{ (na ausência de arredondamentos).}$$

Então, podemos definir o fator multiplicativo sempre como a razão entre o valor do coeficiente que deve ser zerado e o valor do coeficiente pivô (diagonal principal) da linha correspondente ao passo, $i = k$. Esse mesmo fator multiplicativo deve ser aplicado na operação elementar sobre toda a linha $i = 3$, em todas as suas colunas j .

b) As linhas 2 e 3 sofrem alterações através da adição dos termos $-0.9397 * L_1$ e $0.7121 * L_1$, ou seja, se esses termos adicionais tiverem arredondamentos, serão levados para as linhas 2 e 3 resultantes.

c) Como esses mesmos fatores multiplicativos devem ser aplicados para todas as colunas j das linhas $i = 2$ e $i = 3$, é importante que sejam determinados uma única vez, armazenados e usados em todas as colunas de cada linha i .

d) Esses fatores multiplicativos, -0.937 e 0.7121 , aplicados sobre a linha $i = 1$, sofreram arredondamentos no quarto dígito decimal (quarto dígito significativo), logo levarão esse erro no quarto dígito decimal para as novas linhas 2 e 3 geradas a seguir.

Aplicando essas operações elementares sobre as linhas $i = 2$ e $i = 3$, zeraremos a coluna $j = 1$, conforme segue:

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \\ 0 & 1.476 & 0.4164 & \vdots & 0.7121 \end{bmatrix}$$

Note que as linhas 2 e 3 foram alteradas pelos arredondamentos gerados, mas, na coluna $j = 1$, os resultados teóricos previstos são exatos e nulos,

logo não devemos efetuar essas operações aritméticas, basta atribuir esses resultados nulos no algoritmo. Se executarmos as operações na coluna $j = 1$, teremos valores residuais não exatos.

Segundo passo: definido pelo índice $k = 2$, usamos a segunda linha $i = 2$ (em negrito) para eliminar a segunda coluna $j = 2$:

$$\left[\begin{array}{cccc|c} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ \mathbf{0} & \mathbf{0.002170} & \mathbf{-0.3884} & \vdots & \mathbf{-0.9397} \\ 0 & 1.476 & 0.4604 & \vdots & 0.9397 \end{array} \right] \begin{array}{l} L_3 \leftarrow L_3 - (1.476 / \mathbf{0.002170})L_2 \\ L_3 \leftarrow L_3 - 680.2L_2 \end{array}$$

Observe que o fator multiplicativo na linha 2 é muito maior do que a unidade, 680.2, então sofreu arredondamentos no primeiro dígito fracionário (quarto dígito significativo), logo levará esse erro para a nova linha 3, gerada a seguir:

$$\left[\begin{array}{cccc|c} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \\ 0 & 0 & 264.6 & \vdots & -639.9 \end{array} \right]$$

Observe também que a linha 3 sofreu eliminações duas vezes, nos passos $k = 1$ e $k = 2$, e isso caracteriza um processo cumulativo de operações aritméticas com erros de arredondamento. Nessa fase, o sistema simplificado já está na forma de matriz triangular superior, que deve ser equivalente ao sistema original apresentado nesse exemplo, pois as operações elementares sobre linhas aplicadas não podem alterar as equações.

Então, a solução do sistema original:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ -0.319x_1 + 0.884x_2 + 0.279x_3 = 0 \end{cases}$$

deveria ser a mesma do sistema de equações simplificado a seguir:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0x_1 + 0.002170x_2 - 0.3884x_3 = -0.9397 \\ 0x_1 + 0x_2 + 264.6x_3 = 639.9 \end{cases}$$

Processo de retrossubstituição sucessiva

Realizamos o processo de retrossubstituição sucessiva a partir da última equação, determinando a última incógnita e substituindo-a na equação anterior, e assim sucessivamente até obtermos todas as incógnitas.

Primeiro passo: note que o valor de x_3 pode ser diretamente obtido a partir da equação 3, uma vez que essa equação independe de x_1 e x_2 :

$$0x_1 + 0x_2 + 264.6x_3 = 639.9 \quad x_3 = 639.9 / 264.6 \quad x_3 = 2.418$$

Segundo passo: posteriormente, podemos obter os valores de x_2 e x_1 das equações 2 e 1, respectivamente:

$$x_2 = (-0.9397 - (-3884x_3)) / 0.002170 \quad x_2 = -0.2529 \quad (\text{com } x_3 = 2.418)$$

$$x_1 = (1 - 0.832x_2 - 0.193x_3) / 0.448 \quad x_1 = 1.660 \quad (x_3 = 2.418 \text{ e } x_2 = -0.2529)$$

Portanto, a solução obtida para o sistema do **Exemplo 2.1**, com precisão de apenas 4 dígitos significativos, é:

$$S = \{ 1.660, -0.2529, 2.418 \}$$

Lembre-se de que essa solução S foi obtida de um sistema simplificado, cujos coeficientes já sofreram acúmulo de arredondamentos, e não do sistema original.

Se a matriz dos resíduos $R = [A * S - B]$ de cada uma das equações do sistema linear proposto fosse obtida para aferição da solução, deveríamos ter valores nulos, mas normalmente obtemos valores residuais, não nulos, decorrentes dos arredondamentos acumulados. Por exemplo, se substituirmos

a solução S em cada equação original do **Exemplo 2.1**, teremos os seguintes resíduos em módulo:

$$R_1 = | 0.448x_1 + 0.832x_2 + 0.193x_3 - 1 | = 0.0000588 \cong 0.00006$$

$$R_2 = | 0.421x_1 + 0.784x_2 - 0.207x_3 - 0 | = 0.0000604 \cong 0.00006$$

$$R_3 = | -0.319x_1 + 0.884x_2 + 0.279x_3 - 0 | = 0.07848 \cong 0.08$$

O resíduo da equação 3 ficou alto, equivalente ao primeiro dígito decimal ($0.08 \cong 0.1$). Seria esperado um resíduo no quarto dígito decimal, pois temos uma precisão de 4 dígitos significativos. Esse erro decorre principalmente dos arredondamentos dos fatores multiplicativos de cada linha. No próximo exemplo, vamos aplicar uma metodologia para reduzir esse resíduo.

Para comparação, também podemos calcular o erro de arredondamento estimado sobre cada x_i da solução usando uma estimativa do valor exato da solução S_{exato} . Nesse método, temos **somente erros de arredondamento** associados à solução, então o seu valor exato estimado pode ser obtido usando variáveis com mais precisão, com mais de 4 dígitos significativos, para ter menos arredondamentos. Na solução, a seguir, usamos o mesmo algoritmo aplicado no **Exemplo 2.1**, mas operamos com as variáveis *double* de 16 dígitos significativos, gerando os seguintes resultados:

$$S_{\text{exato}} = \{1.561414494051542, -0.199881764024819, 2.418590333334499\}$$

Os resíduos dessa solução exata estimada, obtida com precisão *double*, são:

$$R_{\text{exato}} = \{000000000000e+00, 1.11022302462e-16, 5.68434188608e-14\}$$

(em notação científica)

E os erros estimados, $\text{Erro} = | S - S_{\text{exato}} |$, são:

$$\text{Erro } x_1 = | 1.660 - 1.561414494051542 | = 0.098585505948458 \cong 0.1$$

$$\text{Erro } x_2 = | -0.2529 - (-0.199881764024819) | = -0.053018235975181 \cong 0.05$$

$$\text{Erro } x_3 = | 2.418 - 2.418590333334499 | = 0.000590333334499 \cong 0.0006$$

Perceba que os erros de arredondamento acumulados no processo de eliminação foram propagados também para o primeiro dígito decimal, nos valores de x_1 e x_2 , e, para o quarto dígito, no valor do x_3 . Observe que os

arredondamentos afetam toda a solução, pois em um sistema de equações não temos como dizer que um valor de x é mais exato do que outro. Assim, a solução está afetada no primeiro dígito.

Podemos resumir o algoritmo de **eliminação gaussiana** no seguinte formulário:

Triangularização:

$$\begin{aligned}
 k &= 1, \dots, n - 1 && \text{(define o passo } k\text{)} \\
 i &= k + 1, \dots, n && \text{(define linhas } i\text{)} \\
 j &= k + 1, \dots, n + 1 && \text{(define colunas } j\text{)} \\
 a_{ij} &= a_{ij} - \left(\frac{a_{ik}}{a_{kk}} \right) a_{kj} && \text{(corresponde a } L_i \leftarrow L_i - \left(\frac{a_{ik}}{a_{kk}} \right) L_k \text{)}
 \end{aligned}$$

Retrossubstituição:

$$\begin{aligned}
 i &= n && \\
 x_i &= a_{i, n+1} / a_{ii} && \text{(corresponde à resolução da última equação)} \\
 i &= n - 1, \dots, 1 && \\
 x_i &= \left(a_{i, n+1} - \sum_{j=i+1}^n a_{ij} * x_j \right) / a_{ii} && \text{(corresponde à resolução da } i\text{-ésima equação)}
 \end{aligned}$$

No próximo exemplo, vamos implementar o processo de **pivotação parcial** visando evitar linhas com possíveis zeros na diagonal principal de cada linha $i = k$ e ainda minimizar erros de arredondamento acumulados.

2.1.1.1 Pivotação parcial

A **pivotação parcial** consiste em:

- no início de cada passo k das eliminações, escolher para a_{kk} o elemento de maior módulo entre os coeficientes a_{ik} , $i = k, k + 1, \dots, n$; e
- trocar as linhas k e i , se necessário.

Exemplo 2.2: resolva o sistema de equações lineares, a seguir, pelo método de eliminação de Gauss com **pivotamento parcial** utilizando operações aritméticas com 4 dígitos significativos e arredondamento ponderado. Esse sistema é o mesmo do **Exemplo 2.1**, apenas com as equações em outra ordem.

$$\begin{cases} -0.319x_1 + 0.884x_2 + 0.279x_3 = 0 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \end{cases}$$

Solução:

Na forma de matriz expandida, temos:

$$A = \begin{bmatrix} -0.319 & 0.884 & 0.279 & \vdots & 0 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ 0.448 & 0.832 & 0.193 & \vdots & 1 \end{bmatrix}$$

Processo de triangularização

Antes de promover o processo de triangularização, devemos escolher e reposicionar a melhor linha $i = k$ possível.

Primeiro passo: definido pelo índice $k = 1$, a **pivotação parcial** deve redefinir a melhor linha $i = 1$, correspondente ao primeiro pivô ($k = 1$). Para tal:

- a) Localizamos o maior módulo da coluna, $j = 1$ nesse caso, entre os coeficientes destacados em negrito:

$$j = 1$$

$$i = 3 \begin{bmatrix} -\mathbf{0.319} & 0.884 & 0.279 & \vdots & 0 \\ \mathbf{0.421} & 0.784 & -0.207 & \vdots & 0 \\ \mathbf{(0.448)} & 0.832 & 0.193 & \vdots & 1 \end{bmatrix} \quad (\text{maior módulo está na linha } i = 3)$$

- b) Trocamos linhas, entre a linha $i = 3$, com o melhor coeficiente, e a linha $i = 1$:

$$\begin{bmatrix} -0.319 & 0.884 & 0.279 & \vdots & 0 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ (0.448) & 0.832 & 0.193 & \vdots & 1 \end{bmatrix} \begin{array}{l} L_1 \leftarrow L_3 \\ \\ L_3 \leftarrow L_1 \end{array} \quad (\text{troca da linha } L_1 \text{ com } L_3 \text{ e vice-versa})$$

- c) Geramos a matriz pivotada:

$$\begin{bmatrix} (0.448) & 0.832 & 0.193 & \vdots & 1 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & 0.884 & 0.279 & \vdots & 0 \end{bmatrix}$$

Observe que, na pivotação parcial, apenas trocamos a ordem das equações mudando-as de linhas.

- d) Depois da pivotação, procedemos à **eliminação gaussiana** correspondente ao primeiro passo ($k = 1$):

$$\begin{bmatrix} (0.448) & 0.832 & 0.193 & \vdots & 1 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & 0.884 & 0.279 & \vdots & 0 \end{bmatrix} \begin{array}{l} L_2 \leftarrow L_2 - (0.421/0.448)L_1 \Rightarrow L_2 \leftarrow L_2 - 0.9397L_1 \\ L_3 \leftarrow L_3 - (-0.319/0.448)L_1 \Rightarrow L_3 \leftarrow L_3 + 0.7121L_1 \end{array}$$

- e) Geramos nova matriz escalonada:

$$\begin{bmatrix} (0.448) & 0.832 & 0.193 & \vdots & 1 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \\ 0 & 1.476 & 0.4164 & \vdots & 0.7121 \end{bmatrix}$$

Segundo passo: definido pelo índice $k = 2$, a **pivotação parcial** deve redefinir a melhor linha $i = 2$, correspondente ao segundo pivô ($k = 2$). Para tal:

- a) Realizamos a busca parcial do maior módulo da coluna $j = 2$ (busca a partir da segunda linha, em **negrito**, pois a primeira linha já foi usada para anular a primeira coluna no passo anterior e não pode ser usada novamente para anular a segunda coluna, uma vez que alteraria a coluna $j = 1$ já zerada):

$$i = 3 \begin{array}{c} j = 2 \\ \left[\begin{array}{cccc} (0.448) & 0.832 & 0.193 & \vdots & 1 \\ 0 & \mathbf{0.002170} & -0.3884 & \vdots & -0.9397 \\ 0 & \mathbf{(1.476)} & 0.4164 & \vdots & 0.7121 \end{array} \right] \end{array}$$

b) Trocamos linhas, entre a linha $i = 3$ e a linha $i = 2$:

$$\left[\begin{array}{cccc} (0.448) & 0.832 & 0.193 & \vdots & 1 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \\ 0 & (1.476) & 0.4164 & \vdots & 0.7121 \end{array} \right] \begin{array}{l} L_2 \leftarrow L_3 \\ L_3 \leftarrow L_2 \end{array}$$

c) Geramos a matriz escalonada:

$$\left[\begin{array}{cccc} (0.448) & 0.832 & 0.193 & \vdots & 1 \\ 0 & (1.476) & 0.4164 & \vdots & 0.7121 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \end{array} \right]$$

Observe que, no **Exemplo 2.1**, o pivô antigo, $a_{22} = 0.002170$, era um valor pequeno e gerava um fator multiplicativo elevado, 680.2, que carregava um erro de arredondamento no primeiro dígito fracionário (quarto significativo).

d) Procedemos a triangularização, correspondente ao segundo passo ($k = 2$):

$$\left[\begin{array}{cccc} (0.448) & 0.832 & 0.1930 & \vdots & 1 \\ 0 & (1.476) & 0.4164 & \vdots & 0.7121 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \end{array} \right] \begin{array}{l} L_3 \leftarrow L_3 - (0.002170/1.476)L_2 \\ L_3 - 0.001470L_2 \end{array}$$

Agora o novo pivô, $a_{22} = 1.476$, é um valor maior e gera um fator multiplicativo menor, 0.001470, que, nesse caso, sofreu arredondamentos no sexto dígito fracionário (quarto significativo), logo levará apenas esses erros para a nova linha 3 gerada, conforme segue:

$$\left[\begin{array}{cccc} (0.448) & 0.832 & 0.1930 & \vdots & 1 \\ 0 & (1.476) & 0.4164 & \vdots & 0.7121 \\ 0 & 0 & -0.3893 & \vdots & -0.9412 \end{array} \right]$$

A nova linha $i = 3$ ficou com valores próximos aos anteriores à eliminação deste passo, ou seja, a linha $i = 3$ foi pouco alterada.

Processo de retrossubstituições sucessivas

O sistema de equações simplificado do **Exemplo 2.2**, gerado a partir do processo de eliminação gaussiana, é:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0x_1 + 1.476x_2 + 0.4164x_3 = 0.7121 \\ 0x_1 + 0x_2 - 0.3893x_3 = -0.9412 \end{cases}$$

Note que podemos obter diretamente o valor de x_3 a partir da equação 3, e posteriormente os de x_2 e x_1 a partir dos valores obtidos anteriormente.

$$\begin{aligned} x_3 &= -0.9412 / (-0.3893) & x_3 &= 2.418 \\ x_2 &= (0.7121 - 0.4164 x_3) / 1.476 & x_2 &= -0.1997 \\ x_1 &= (1 - 0.832 x_2 - 0.193 x_3) / 0.448 & x_1 &= 1.561 \end{aligned}$$

Portanto, a solução do sistema dado no **Exemplo 2.2** é:

$$S = \{ 1.561, -0.1997, 2.418 \}$$

Os resíduos $R = [A * S - B]$ dessa solução são:

$$\begin{aligned} R_1 &= |-0.319x_1 + 0.884x_2 + 0.279x_3 - 0| = 0.0001282 \cong 0.0001 \\ R_2 &= |0.421x_1 + 0.784x_2 - 0.207x_3 - 0| = 0.0000902 \cong 0.0001 \\ R_3 &= |0.448x_1 + 0.832x_2 + 0.193x_3 - 1| = 0.0001484 \cong 0.0001 \end{aligned}$$

Pelos resíduos informados, observamos que a solução obtida no **Exemplo 2.2** está mais próxima da exata porque os resíduos estão no quarto dígito fracionário, diferentemente do **Exemplo 2.1**, em que os resíduos chegavam ao primeiro dígito.

Nesse caso, também podemos calcular o erro de arredondamento comparando a solução obtida com precisão de 4 dígitos a uma solução mais precisa obtida com o mesmo algoritmo usando precisão *double*:

$$S_{\text{exato}} = \{ 1.561414494051471, -0.199881764024781, 2.418590333334500 \}$$

Observe que os erros estimados, $\text{Erro} = [S - S_{\text{exato}}]$, são:

$$\text{Erro } x_1 = | 1.561 - 1.561414494051471 | = 0.000414494051471 \cong 0.0004$$

$$\text{Erro } x_2 = | -0.1997 - (-0.199881764024781) | = 0.000181764024781 \cong 0.0002$$

$$\text{Erro } x_3 = | 2.418 - 2.418590333334500 | = 0.000590333334500 \cong 0.0006$$

Agora, perceba que os erros estimados de S também foram menores no **Exemplo 2.2** com pivotação parcial, ou seja, a solução S com pivotação parcial acumulou menos erros de arredondamento.

Com o processo de pivotamento parcial:

- a) eliminamos possíveis pivôs nulos, caso haja possibilidades de troca de linhas; e
- b) conseguimos uma redução nos efeitos cumulativos de erros de arredondamento (diminuição da perda de significação), pois os novos pivôs são os maiores possíveis de cada coluna, gerando menores fatores multiplicativos, cujos arredondamentos ocorrem em dígitos menos significativos.

Na próxima seção, vamos implementar o processo de **pivotação total** visando obter soluções computacionalmente mais estáveis em relação às perturbações introduzidas por arredondamentos.

2.1.1.2 Pivotação total

Alternativamente, podemos implementar o método de eliminação de Gauss usando a **pivotação total**, que é computacionalmente um pouco mais eficiente, induzindo normalmente a um menor erro de arredondamento acumulado, de forma a obter soluções um pouco mais estáveis em relação às perturbações

introduzidas por arredondamentos. No **pivotamento total**, ou **completo**, procuramos o elemento de maior módulo entre todos os elementos disponíveis na matriz de coeficientes promovendo trocas de linhas e colunas. Para avaliar as consequências dessas trocas de linhas e colunas, devemos interpretar os elementos da matriz expandida como termos das equações do sistema, assim:

- a) a troca de **linhas** significa apenas trocar a ordem na apresentação das **equações**; e
- b) a troca de **colunas** significa trocar a ordem de apresentação das **incógnitas** do sistema.

Exemplo 2.3: resolva o sistema de equações lineares, a seguir, usando **pivotação total**, operações aritméticas com 4 dígitos significativos e arredondamento ponderado.

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ -0.319x_1 + 0.884x_2 + 0.279x_3 = 0 \end{cases}$$

Solução:

Na forma de matriz expandida, temos:

$$A = \begin{bmatrix} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & 0.884 & 0.279 & \vdots & 0 \end{bmatrix}$$

Primeiro passo: na pivotação total, correspondente ao primeiro pivô ($k = 1$):

- a) Buscamos o maior módulo entre todos os elementos da matriz de coeficientes:

$$\begin{array}{c}
 j = 2 \\
 \begin{bmatrix} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & \mathbf{0.884} & 0.279 & \vdots & 0 \end{bmatrix} \\
 i = 3
 \end{array}
 \begin{array}{ccc}
 x_1 & x_2 & x_3
 \end{array}$$

(o coeficiente de maior módulo está na coluna $j = 2$ e na linha $i = 3$, em negrito).

Observe que foi acoplada uma linha adicional, embaixo da matriz de coeficientes, para o armazenamento da ordem de apresentação das incógnitas envolvidas. Então, inicialmente temos a ordem natural das incógnitas x_1, x_2 e x_3 , na primeira, segunda e terceira colunas.

b) Efetuamos a troca de linhas e colunas:

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0.421 & 0.784 & -0.207 & \vdots & 0 \\ -0.319 & \mathbf{0.884} & 0.279 & \vdots & 0 \end{bmatrix} \begin{array}{l} L_1 \leftarrow L_3 \\ \\ L_3 \leftarrow L_1 \end{array}$$

$$\begin{array}{ccc} x_1 & x_2 & x_3 \\ C_1 & C_2 & \\ \uparrow & \uparrow & \\ C_2 & C_1 & \end{array}$$

Trocamos a linha L_1 com L_3 e vice-versa, e a coluna C_1 com C_2 e vice-versa.

c) Geramos a matriz pivotada:

$$\begin{bmatrix} \mathbf{0.884} & -0.319 & 0.279 & \vdots & 0 \\ 0.784 & 0.421 & -0.207 & \vdots & 0 \\ 0.832 & 0.448 & 0.193 & \vdots & 1 \end{bmatrix}$$

$$\begin{array}{ccc} x_2 & x_1 & x_3 \end{array}$$

Note que, no processo de pivotamento total, a ordem de apresentação das incógnitas x_i envolvidas foi alterada para x_2, x_1 e x_3 .

Segundo passo: no processo de triangularização, correspondente ao primeiro passo ($k = 1$), temos:

$$\begin{bmatrix} \mathbf{0.884} & -0.319 & 0.279 & \vdots & 0 \\ 0.784 & 0.421 & -0.207 & \vdots & 0 \\ 0.832 & 0.448 & 0.193 & \vdots & 1 \end{bmatrix} \begin{array}{l} L_2 \leftarrow L_2 - (0.784/0.884) L_1 \Rightarrow L_2 \leftarrow L_2 - 0.8869L_1 \\ L_3 \leftarrow L_3 - (0.832/0.884) L_1 \Rightarrow L_3 \leftarrow L_3 - 0.9412L_1 \end{array}$$

$$\begin{array}{ccc} x_2 & x_1 & x_3 \end{array}$$

$$\begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & 0.7039 & -0.4544 & \vdots & 0 \\ 0 & 0.7482 & -0.06959 & \vdots & 1 \end{bmatrix}$$

$$\begin{array}{ccc} x_2 & x_1 & x_3 \end{array}$$

Terceiro passo: na pivotação total, correspondente ao segundo pivô ($k = 2$):

- a) Buscamos o maior módulo entre os elementos da matriz, a partir da segunda linha e segunda coluna:

$$j = 2$$

$$i = 3 \begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & 0.7039 & -0.4544 & \vdots & 0 \\ 0 & \mathbf{0.7482} & -0.06959 & \vdots & 1 \end{bmatrix}$$

$$\begin{array}{ccc} x_2 & x_1 & x_3 \end{array}$$

(elemento de maior módulo localizado na coluna $j = 2$ e linha $i = 3$).

- b) Efetuamos somente troca de linhas, uma vez que, na coluna, a matriz já está naturalmente pivotada:

$$\begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & 0.7039 & -0.4544 & \vdots & 0 \\ 0 & \mathbf{0.7482} & -0.06959 & \vdots & 1 \end{bmatrix} \begin{array}{l} L_2 \leftarrow L_3 \\ L_3 \leftarrow L_2 \end{array}$$

$$\begin{array}{ccc} x_2 & x_1 & x_3 \end{array}$$

c) Geramos a matriz pivotada:

$$\begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & \mathbf{0.7482} & -0.06959 & \vdots & 1 \\ 0 & 0.7039 & -0.4544 & \vdots & 0 \end{bmatrix}$$

$$\begin{matrix} x_2 & x_1 & x_3 \end{matrix}$$

Quarto passo: no processo de triangularização, correspondente ao segundo passo ($k = 2$), temos:

$$\begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & \mathbf{0.7482} & -0.06959 & \vdots & 1 \\ 0 & 0.7039 & -0.4544 & \vdots & 0 \end{bmatrix} \begin{matrix} L_3 \leftarrow L_3 - (0.7039 / 0.7482)L_2 \\ L_3 \leftarrow L_3 - 0.9408L_2 \end{matrix}$$

$$\begin{matrix} x_2 & x_1 & x_3 \end{matrix}$$

$$\begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & \mathbf{0.7482} & -0.06959 & \vdots & 1 \\ 0 & 0 & -0.3889 & \vdots & -0.9408 \end{bmatrix}$$

$$\begin{matrix} x_2 & x_1 & x_3 \end{matrix}$$

Quinto passo: no processo de retrossubstituição, temos:

$$\begin{bmatrix} 0.884 & -0.319 & 0.279 & \vdots & 0 \\ 0 & \mathbf{0.7482} & -0.06959 & \vdots & 1 \\ 0 & 0 & -0.3889 & \vdots & -0.9408 \end{bmatrix}$$

$$\begin{matrix} x_2 & x_1 & x_3 \end{matrix}$$

Retornando à forma de equações (note a nova ordem das incógnitas):

$$\begin{cases} 0.884x_2 - 0.319x_1 + 0.279x_3 = 0 \\ 0x_2 + 0.7482x_1 - 0.06959x_3 = 1 \\ 0x_2 + 0x_1 - 0.3889x_3 = -0.9408 \end{cases}$$

$$\begin{aligned}x_3 &= -0.9408/(-0.3889) & x_3 &= 2.419 \\x_1 &= (1 + 0.06959 x_3) / 0.7482 & x_1 &= 1.561 \\x_2 &= (0 + 0.319 x_1 - 0.279 x_3) / 0.884 & x_2 &= -0.2002\end{aligned}$$

Neste ponto, precisamos reordenar a solução sabendo que a ordem das incógnitas obtidas na retrossubstituição é $\{x_2, x_1, x_3\}$, enquanto a solução ordenada é $S = \{x_1, x_2, x_3\} = \{1.561, -0.2002, 2.419\}$.

Os resíduos $R = [A * S - B]$ dessa solução são:

$$\begin{aligned}R_1 &= | 0.448x_1 + 0.832x_2 + 0.193x_3 - 1 | = 0.0005 \\R_2 &= | 0.421x_1 + 0.784x_2 - 0.207x_3 - 0 | = 0.0003 \\R_3 &= | -0.319x_1 + 0.884x_2 + 0.279x_3 - 0 | = 0.0003\end{aligned}$$

E os erros estimados, $Erro = [S - S_{\text{exato}}]$, obtidos foram:

$$\begin{aligned}Erro x_1 &= | 1.561 - 1.561414494051471 | = 0.000414494051471 \cong 0.0004 \\Erro x_2 &= | -0.2002 - (-0.199881764024781) | = -0.000318235975219 \cong 0.0003 \\Erro x_3 &= | 2.419 - 2.418590333334500 | = 0.000590333334500 \cong 0.0006\end{aligned}$$

Os resultados obtidos com pivotação total no **Exemplo 2.3** são equivalentes aos obtidos com pivotação parcial no **Exemplo 2.2**, considerando os resíduos e os erros estimados. Estatisticamente, temos poucos tipos de sistemas que têm solução significativamente melhor com pivotação total em relação à sua solução com pivotação parcial.

2.1.1.3 Quantitativo de soluções de $A * X = B$

As soluções dos sistemas de equações lineares podem ser classificadas segundo três possibilidades distintas:

- a) **Sistema determinado:** quando o sistema de equações tem solução única, a matriz de coeficientes é não singular, isto é, o seu determinante é não nulo. Ocorre quando todas as equações do

sistema são linearmente independentes, ou seja, nenhuma equação é combinação linear de outras.

- b) **Sistema indeterminado:** quando o sistema de equações tem infinitas soluções, a matriz de coeficientes é singular, ou seja, o seu determinante é nulo. Ocorre quando temos um sistema de equações lineares cujos coeficientes possuem alguma relação de dependência, por exemplo, uma equação é gerada a partir da combinação linear de outra(s), ou já temos menos equações do que incógnitas. Podemos concluir que a(s) equação(ões) gerada(s) a partir da combinação linear de outras existentes não acrescenta(m) informação(ões) nova(s) ao conjunto de equações do sistema. Dessa forma, o sistema se comporta como se tivesse menos equações do que incógnitas, deixando alguma(s) incógnita(s) livre(s) de equações próprias que restrinjam o(s) seu(s) valor(es). No método de eliminação de Gauss, podemos constatar esse fato observando que, no final do processo de eliminação, uma ou mais linhas inteiras da matriz expandida se anula(m), inclusive o(s) correspondente(s) termo(s) independente(s) se anula(m).
- c) **Sistema impossível:** quando o sistema de equações não tem solução alguma, a matriz de coeficientes também é singular. Ocorre quando alguma equação do sistema é impossível de ser satisfeita. Por exemplo, se alguma equação do sistema é gerada a partir da combinação linear “parcial”, de apenas um dos lados de duas outras equações, então geramos uma equação inconsistente. No método de Gauss, podemos constatar esse fato observando que, no final do processo de eliminação, uma ou mais linhas da matriz de coeficientes se anulam, mas o termo independente não se anula. Na prática, significa ter uma equação com todos os coeficientes de x nulos e com um termo independente não nulo, o que é impossível de ser satisfeito.

O algoritmo de retrossubstituição adotado neste livro tratará das três possibilidades de solução.

Nas possibilidades (b) e (c), os elementos nulos podem não ser necessariamente iguais a zero, mas podem assumir valores residuais.

Um sistema possível de resolver, mas indeterminado, comporta-se conforme o exemplo a seguir:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \quad (\text{com } L_3 = L_2) \end{cases}$$

Se aplicarmos a eliminação gaussiana, chegaremos à seguinte matriz triangular superior na forma expandida:

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 & \vdots & 1 \\ 0 & 0.002170 & -0.3884 & \vdots & -0.9397 \\ 0 & 0 & 0 & \vdots & 0 \end{bmatrix}$$

(usando aritmética exata em $i = 3$)

Observe que a terceira linha, ou equação, foi completamente eliminada no segundo passo, $k = 2$, não contribuindo mais com a solução do sistema. Logo, temos um sistema efetivo de $n = 2$ equações, mas com três incógnitas:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0x_1 + 0.002170x_2 - 0.3884x_3 = -0.9397 \\ 0x_1 + 0x_2 + 0x_3 = 0 \end{cases}$$

$$\Rightarrow \begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0x_1 + 0.002170x_2 - 0.3884x_3 = -0.9397 \end{cases}$$

Assim, o nosso algoritmo da função retrossubstituição deve tratar sistemas com uma linha a menos do que o número de incógnitas, ou seja, com uma equação sendo combinação linear de outra(s), cujo coeficiente da diagonal principal da última linha e o seu termo independente sejam numericamente nulos. Desse modo, se esses coeficientes tiverem valores nulos ou residuais, devido aos arredondamentos, então atribuiremos um valor escolhido para o x_n , pois o algoritmo de retrossubstituição inicia com o cálculo do x_n . Além disso, a linha que se torna nula será a última, sempre que a pivotação for aplicada; e, se o método não tiver pivotação associada, algum pivô poderá se tornar nulo antes, ainda dentro do algoritmo de eliminação.

Já um sistema **impossível** de resolver se comporta conforme o exemplo a seguir:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 0 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 50 \end{cases}$$

(L_3 difere de L_2 somente pelo termo independente)

Se aplicarmos a eliminação gaussiana, chegaremos à seguinte matriz expandida:

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 & : & 1 \\ 0 & 0.002170 & -0.3884 & : & -0.9397 \\ 0 & 0 & 0 & : & 49.0603 \end{bmatrix}$$

(usando aritmética exata em $i = 3$)

Observe que a terceira linha, ou equação, foi parcialmente eliminada no passo $k = 2$, mas ficou com um valor não nulo no termo independente:

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0x_1 + 0.002170x_2 - 0.3884x_3 = -0.9397 \\ 0x_1 + 0x_2 + 0x_3 = 49.0603 \end{cases}$$

Lembre-se de que o algoritmo da função retrossubstituição também deve levar em conta essa possibilidade de ter o coeficiente da diagonal principal da última linha numericamente nulo, enquanto o seu termo independente é não nulo. Portanto, a última equação desse sistema é **impossível** de ser satisfeita, pois não possui solução, ou seja, uma das equações do sistema é “inconsistente”. Nesse caso, o algoritmo não tem como tratar um sistema inconsistente. Como alternativa, podemos apenas interromper o algoritmo, ou atribuir um valor **NaN** (Not a Number) para o vetor solução, indicando solução como conjunto vazio.



O símbolo IEEE NaN é um resultado para operações indefinidas como ($Inf - Inf$), ($0/0$) e para quaisquer operações envolvendo NaN.

Neste capítulo e ao longo desta obra, vamos apresentar todos os [algoritmos](#) (compactados) no **Caderno de Algoritmos**, que disponibilizamos no *link* <<http://sergiopeters.prof.ufsc.br/algoritmos-livro/>>, indicando o título do algoritmo correspondente ao método ou exemplo apresentado. Aproveite este momento para fazer o *download* deste Caderno e conferir o primeiro algoritmo, disponibilizado no arquivo **Cap2ElimGauss.m**, sobre eliminação gaussiana com pivotação parcial ou total aplicado ao **Exemplo 2.3**.



Nos algoritmos exemplos deste livro, usamos principalmente o *software* livre GNU Octave, compatível com MathLab®, disponível em: <<https://www.gnu.org/software/octave/>>.

No algoritmo de eliminação de Gauss, o processo de pivotação, total ou parcial, está inserido antes da etapa da eliminação. No processo de escalonamento, as operações nas colunas envolvem algebricamente todos os j , *for* $j = 1: n + 1$, pois são operações sobre as linhas i inteiras; mas a operação na coluna $j = k$ tem resultado conhecido, uma vez que sabemos que esse resultado exato é sempre nulo. Assim, essa operação aritmética conhecida foi suprimida no algoritmo, *for* $j = k + 1: n + 1$, e atribuímos o resultado nulo conhecido para a coluna $j = k$ depois do encerramento do *for* j . Além disso, as operações nas colunas j à esquerda de k ($j = 1: k$) envolveriam somente operações com coeficientes nulos, que também são desnecessárias e por isso também são suprimidas.

Sobre os **Exemplos 2.1, 2.2 e 2.3** que desenvolvemos, algumas considerações são importantes e merecem destaque:

- a) Os resíduos encontrados servem de parâmetros para validar uma solução, ou seja, servem para verificar o número de dígitos exatos da solução, mas não para sistemas mal condicionados, nos quais não basta calcular os resíduos, como veremos mais adiante.
- b) Os erros calculados nos exemplos capturam somente os erros de arredondamento da solução, pois não temos erros de truncamento envolvidos no método de eliminação gaussiana.
- c) As soluções com menores erros de arredondamento acumulados foram obtidas com pivotação parcial.

- d) Para minimizar o efeito cumulativo dos erros de arredondamento, podemos tentar modificar as operações elementares do processo de escalonamento da seguinte forma:

Substituímos a linha a ser eliminada pelo produto entre a própria linha e o elemento pivô, subtraído do produto entre a linha do pivô e o elemento a ser eliminado. Por exemplo, se a linha L_3 sofre a seguinte operação de eliminação do elemento a_{31} com o pivô da primeira linha a_{11} no passo $k = 1$, temos:

$$L_3 \leftarrow L_3 - \frac{a_{31}}{a_{11}} L_1$$

Sabendo que o objetivo dessa operação é anular o elemento a_{31} , podemos modificá-la desde que mantenhamos o resultado nulo na primeira coluna. Assim, se

$$L_3 - \frac{a_{31}}{a_{11}} L_1 = 0, \text{ para } j = 1, \quad a_{31} - \frac{a_{31}}{a_{11}} a_{11} = 0$$

Então, podemos multiplicar essa equação pelo elemento pivô a_{11} , resultando em uma forma alternativa equivalente que mantém o resultado nulo para primeira coluna da matriz. Mas esse processo sem divisões exige uma multiplicação a mais em cada linha, aumentando o número total de operações aritméticas:

$$a_{11} * L_3 - a_{31} * L_1 = 0 \text{ para } j = 1, \quad a_{11} * a_{31} - a_{31} * a_{11} = 0$$

Com essa forma alternativa de aplicar as operações de eliminação, temos um menor acúmulo de erros de arredondamento, pois não haverá divisões ao longo do processo de eliminação, sendo necessária apenas uma divisão no final do processo de retrossubstituições, no momento de determinar as incógnitas x_i . Mesmo assim, esse procedimento alternativo pode gerar erros por perda de significação, especialmente quando temos pivôs de grande magnitude ou quando o número de equações é elevado. Nesses casos, ocorre um acúmulo de operações de multiplicação sobre as linhas do sistema, amplificando seus valores e podendo até atingir a região de *overflow*, sobretudo nas últimas linhas da matriz escalonada.

2.1.2 Método de Gauss-Jordan

O método de Gauss-Jordan consiste em transformar a matriz dos coeficientes, que compõe a matriz expandida, em identidade. Então, depois de transformada, a última coluna dessa matriz expandida será a solução de $A * X = B$. Essa transformação é obtida através da aplicação sucessiva de operações elementares sobre linhas buscando a eliminação de todos os coeficientes da respectiva coluna, exceto o da diagonal principal. Também podemos associar esse método a um processo de pivotamento parcial ou total.

Não detalharemos o algoritmo desse método aqui porque ele necessita do dobro de operações aritméticas em relação ao método de eliminação gaussiana e, por conseguinte, acumula mais arredondamentos.

2.1.3 Método da inversão de matrizes

No método de inversão de matrizes, multiplicamos o sistema $A * X = B$ pela matriz inversa A^{-1} de A :

$$A^{-1}(A * X) = A^{-1}(B)$$

Utilizando a associatividade do produto matricial, o sistema resulta em:

$$(A^{-1} * A)X = A^{-1} * B$$

$$(I)X = A^{-1} * B$$

$$X = A^{-1} * B$$

Portanto, podemos obter o vetor de incógnitas X apenas multiplicando a matriz inversa A^{-1} pelo vetor de termos independentes B . Trata-se de um método eficiente quando dispomos da inversa da matriz A ; caso contrário, teremos o custo adicional da determinação da inversa.

Podemos obter a matriz inversa de A de várias maneiras. Aqui, vamos usar o próprio método de escalonamento de Gauss-Jordan aplicado à matriz aumentada $[A \dot{ : } I]$, conforme o **Exemplo 2.4**.

Exemplo 2.4: resolva o sistema de equações lineares, a seguir, usando o método da inversão de matrizes. Utilize o processo de pivotamento parcial para evitar pivôs nulos e diminuir o acúmulo de arredondamentos.

$$\begin{cases} 3x_1 + 1.5x_2 + 4.75x_3 = 8 \\ 4x_1 + 2x_2 + 3x_3 = 7 \\ 2x_1 + 5x_2 + 3x_3 = -12 \end{cases}$$

Solução:

Agora, adotamos operações aritméticas com 4 dígitos significativos e arredondamento ponderado. Assim, na forma matricial, temos:

$$\begin{bmatrix} 3 & 1.5 & 4.75 \\ 4 & 2 & 3 \\ 2 & 5 & 3 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 7 \\ -12 \end{bmatrix}$$

Para a geração da matriz inversa de A , adotamos um método prático clássico: geramos a matriz aumentada $[A \dot{=} I]$ composta da matriz A concatenada com a matriz identidade I da mesma ordem de A :

$$\begin{bmatrix} 3 & 1.5 & 4.75 & \dot{=} & 1 & 0 & 0 \\ 4 & 2 & 3 & \dot{=} & 0 & 1 & 0 \\ 2 & 5 & 3 & \dot{=} & 0 & 0 & 1 \end{bmatrix}$$

Através de operações elementares sobre linhas, transformamos a matriz A na matriz identidade I , e conseqüentemente a matriz identidade inicial transforma-se na inversa A^{-1} . Trata-se de um procedimento análogo ao que é utilizado no método de Gauss-Jordan, mas, nesse caso, estendido para $2n$ colunas.

A seguir, vamos apresentar os passos para a obtenção de A^{-1} .

Primeiro passo: utilizando o método de pivotação parcial, correspondente ao primeiro pivô ($k = 1$):

a) Buscamos o maior módulo da coluna $j = 1$:

$$j = 1$$

$$i = 2 \begin{bmatrix} 3 & 1.5 & 4.75 & \vdots & 1 & 0 & 0 \\ (4) & 2 & 3 & \vdots & 0 & 1 & 0 \\ 2 & 5 & 3 & \vdots & 0 & 0 & 1 \end{bmatrix}$$

(elemento de maior módulo da coluna $j = 1$ está na linha $i = 2$)

b) Trocamos linhas:

$$\begin{bmatrix} 3 & 1.5 & 4.75 & \vdots & 1 & 0 & 0 \\ (4) & 2 & 3 & \vdots & 0 & 1 & 0 \\ 2 & 5 & 3 & \vdots & 0 & 0 & 1 \end{bmatrix} \begin{array}{l} L_1 \leftarrow L_2 \\ L_2 \leftarrow L_1 \end{array}$$

(troca da linha L_1 com L_2 e vice-versa)

c) Geramos a matriz pivotada:

$$\begin{bmatrix} 4 & 2 & 3 & \vdots & 0 & 1 & 0 \\ 3 & 1.5 & 4.75 & \vdots & 1 & 0 & 0 \\ 2 & 5 & 3 & \vdots & 0 & 0 & 1 \end{bmatrix}$$

Segundo passo: no processo de normalização do primeiro pivô ($k = 1$), temos:

$$\begin{bmatrix} 4 & 2 & 3 & \vdots & 0 & 1 & 0 \\ 3 & 1.5 & 4.75 & \vdots & 1 & 0 & 0 \\ 2 & 5 & 3 & \vdots & 0 & 0 & 1 \end{bmatrix} L_1 \leftarrow L_1 / 4$$

$$\begin{bmatrix} 1 & 0.5 & 0.75 & \vdots & 0 & 0.25 & 0 \\ 3 & 1.5 & 4.75 & \vdots & 1 & 0 & 0 \\ 2 & 5 & 3 & \vdots & 0 & 0 & 1 \end{bmatrix}$$

Terceiro passo: no processo de diagonalização, correspondente ao primeiro passo ($k = 1$), temos:

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 3 & 1.5 & 4.75 & 1 & 0 & 0 \\ 2 & 5 & 3 & 0 & 0 & 1 \end{array} \right] \begin{array}{l} L_2 \leftarrow L_2 - 3L_1 \\ L_3 \leftarrow L_3 - 2L_1 \end{array}$$

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \\ 0 & 4 & 1.5 & 0 & -0.5 & 1 \end{array} \right]$$

Quarto passo: na pivotação parcial, correspondente ao segundo pivô ($k = 2$):

- a) Buscamos o maior módulo da coluna $k = 2$ (a partir da segunda linha, pois a primeira linha já foi utilizada no processo de eliminação):

$$j = 2$$

$$i = 3 \left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \\ 0 & (4) & 1.5 & 0 & -0.5 & 1 \end{array} \right]$$

(o elemento de maior módulo da coluna $j = 2$ está na linha $i = 3$).

- b) Trocamos linhas:

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \\ 0 & (4) & 1.5 & 0 & -0.5 & 1 \end{array} \right] \begin{array}{l} L_2 \leftarrow L_3 \\ L_3 \leftarrow L_2 \end{array}$$

(troca da linha L_2 com L_3 e vice-versa)

- c) Geramos a matriz pivotada:

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 4 & 1.5 & 0 & -0.5 & 1 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \end{array} \right]$$

Note que a pivotação parcial eliminou um pivô nulo.

Quinto passo: no processo de normalização do segundo pivô ($k = 2$), temos:

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 4 & 1.5 & 0 & -0.5 & 1 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \end{array} \right] L_2 \leftarrow L_2/4$$

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 1 & 0.375 & 0 & -0.125 & 0.25 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \end{array} \right]$$

Sexto passo: no processo de diagonalização, correspondente ao segundo passo ($k = 2$), temos:

$$\left[\begin{array}{ccc|cc} 1 & 0.5 & 0.75 & 0 & 0.25 & 0 \\ 0 & 1 & 0.375 & 0 & -0.125 & 0.25 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \end{array} \right] \begin{array}{l} L_1 \leftarrow L_1 - 0.5L_2 \\ L_3 \leftarrow L_3 - 0L_2 \end{array}$$

$$\left[\begin{array}{ccc|cc} 1 & 0 & 0.5625 & 0 & 0.3125 & -0.125 \\ 0 & 1 & 0.375 & 0 & -0.125 & 0.25 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \end{array} \right]$$

Sétimo passo: no processo de normalização do terceiro pivô ($k = 3$), temos:

$$\left[\begin{array}{ccc|cc} 1 & 0 & 0.5625 & 0 & 0.3125 & -0.125 \\ 0 & 1 & 0.375 & 0 & -0.125 & 0.25 \\ 0 & 0 & 2.5 & 1 & -0.75 & 0 \end{array} \right] L_3 \leftarrow L_3/2.5$$

$$\left[\begin{array}{ccc|cc} 1 & 0 & 0.5625 & 0 & 0.3125 & -0.125 \\ 0 & 1 & 0.375 & 0 & -0.125 & 0.25 \\ 0 & 0 & 1 & 0.4 & -0.3 & 0 \end{array} \right]$$

Oitavo passo: no processo de diagonalização, correspondente ao terceiro passo ($k = 3$), temos:

$$\left[\begin{array}{ccc|cc} 1 & 0 & 0.5625 & 0 & 0.3125 & -0.125 \\ 0 & 1 & 0.375 & 0 & -0.125 & 0.25 \\ 0 & 0 & 1 & 0.4 & -0.3 & 0 \end{array} \right] \begin{array}{l} L_1 \leftarrow L_1 - 0.5625L_3 \\ L_2 \leftarrow L_2 - 0.375L_3 \end{array}$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -0.225 & 0.4812 & -0.125 \\ 0 & 1 & 0 & -0.15 & -0.0125 & 0.25 \\ 0 & 0 & 1 & 0.4 & -0.3 & 0 \end{array} \right]$$

Então,

$$A^{-1} = \begin{bmatrix} -0.225 & 0.4812 & -0.125 \\ -0.15 & -0.0125 & 0.25 \\ 0.4 & -0.3 & 0 \end{bmatrix}$$

Para obter o vetor solução X , efetuamos a multiplicação entre A^{-1} e B , logo:

$$X = A^{-1} * B = \begin{bmatrix} -0.225 & 0.4812 & -0.125 \\ -0.15 & -0.0125 & 0.25 \\ 0.4 & -0.3 & 0 \end{bmatrix} \cdot \begin{bmatrix} 8 \\ 7 \\ -12 \end{bmatrix} = \begin{bmatrix} 3.069 \\ -4.288 \\ 1.100 \end{bmatrix}$$

Observe que a troca de linhas efetuadas na pivotação parcial não afeta a matriz inversa e não deve alterar a ordem dos elementos do termo independente. Então, a solução do sistema é a seguinte:

$$S = \{ 3.069, -4.288, 1.100 \}$$

A **vantagem** desse método de inversão de matriz sobre a eliminação gaussiana é a possibilidade de **reuso da matriz inversa** A^{-1} para outros sistemas com a mesma matriz A de coeficientes e com diferentes vetores B de termos independentes.

O algoritmo do método de inversão de matrizes necessita de quase o triplo de operações aritméticas que o método de eliminação gaussiana, por conseguinte acumula mais arredondamentos. Por isso, só vale o esforço computacional se a matriz inversa resultante puder ser reutilizada muitas vezes.

A seguir, vamos apresentar métodos que fazem uso de decomposição de matrizes em vez de eliminação.

2.1.4 Método de decomposição LU (de Crout)

Matrizes quadradas não singulares podem ser decompostas no produto de duas matrizes triangulares L e U , em que L é uma matriz triangular inferior e U é uma matriz triangular superior, de modo que $A = L * U$.

Além disso, se atribuirmos valores fixos aos elementos da diagonal principal, seja na matriz L ($l_{ii} = 1$, método de Doolittle) ou na U ($u_{ii} = 1$, método de Crout), essa decomposição será única.

Para a solução de $A * X = B$, podemos decompor A , segundo o método de Crout, em:

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ \vdots & \vdots & \dots \\ l_{n1} & l_{n2} & l_{nn} \end{bmatrix} \text{ e } U = \begin{bmatrix} 1 & u_{12} & u_{1n} \\ 0 & 1 & u_{2n} \\ \vdots & \vdots & \dots \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

tal que $L * U = A$. Então, o sistema $A * X = B$ torna-se $(L * U) X = B$.

Também podemos associar o produto das matrizes L e U , $(L * U) X = B$, como $L (U * X) = B$.

Considerando $U * X = C$ (vetor auxiliar C desconhecido), resolvemos primeiro $L * C = B$ determinando C e, depois, $U * X = C$ determinando X .

A vantagem do método de decomposição LU sobre o método de eliminação gaussiana é a possibilidade de reuso das matrizes L e U decompostas para outros sistemas que usem a mesma matriz A de coeficientes e com diferentes vetores B de termos independentes.

Vamos exemplificar a determinação dos elementos de L e U para uma matriz 3×3 na qual a multiplicação das matrizes $L * U$ pode ser usada para definir os valores de l_{ij} e u_{ij} em função de a_{ij} , pois o resultado do produto $L * U$ é a matriz A original, logo:

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} * \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Então, nesse exemplo, o produto $L * U$ gera 9 equações lineares com 9 incógnitas, uma para cada elemento da matriz:

$$\begin{bmatrix} l_{11} * 1 + 0 * 0 + 0 * 0 & l_{11} * u_{12} + 0 * 1 + 0 * 0 & l_{11} * u_{13} + 0 * u_{23} + 0 * 1 \\ l_{21} * 1 + l_{22} * 0 + 0 * 0 & l_{21} * u_{12} + l_{22} * 1 + 0 * 0 & l_{21} * u_{13} + l_{22} * u_{23} + 0 * 1 \\ l_{31} * 1 + l_{32} * 0 + l_{33} * 0 & l_{31} * u_{12} + l_{32} * 1 + l_{33} * 0 & l_{31} * u_{13} + l_{32} * u_{23} + l_{33} * 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Podemos obter de forma direta a solução desse sistema de equações usando os seguintes passos:

Passo $k = 1$: resolvemos as equações correspondentes à primeira coluna $j = 1$ da matriz $L * U = A$:

$$l_{11} * 1 + 0 * 0 + 0 * 0 = a_{11} \quad l_{11} = a_{11}$$

$$l_{21} * 1 + l_{22} * 0 + 0 * 0 = a_{21} \quad \Rightarrow \quad l_{21} = a_{21}$$

$$l_{31} * 1 + l_{32} * 0 + l_{33} * 0 = a_{31} \quad l_{31} = a_{31}$$

Para a primeira linha $i = 1$ (à direita da diagonal principal), temos:

$$l_{11} * u_{12} + 0 * 1 + 0 * 0 = a_{12} \quad \Rightarrow \quad u_{12} = a_{12} / l_{11}$$

$$l_{11} * u_{13} + 0 * u_{23} + 0 * 1 = a_{13} \quad u_{13} = a_{13} / l_{11}$$

Passo $k = 2$: resolvemos as equações correspondentes à segunda coluna $j = 2$ (a partir da diagonal principal):

$$l_{21} * u_{12} + l_{22} * 1 + 0 * 0 = a_{22} \quad \Rightarrow \quad l_{22} = a_{22} - l_{21} * u_{12}$$

$$l_{31} * u_{12} + l_{32} * 1 + l_{33} * 0 = a_{32} \quad l_{32} = a_{32} - l_{31} * u_{12}$$

Para a segunda linha $i = 2$ (à direita da diagonal principal), temos:

$$l_{21} * u_{13} + l_{22} * u_{23} + 0 * 1 = a_{23} \quad \Rightarrow \quad u_{23} = (a_{23} - l_{21} * u_{13}) / l_{22}$$

Passo $k = 3$: resolvemos as equações correspondentes à terceira coluna $j = 3$ (a partir da diagonal principal):

$$l_{31} * u_{13} + l_{32} * u_{23} + l_{33} * 1 = a_{33} \quad \Rightarrow \quad l_{33} = (a_{33} - (l_{31} * u_{13} + l_{32} * u_{23}))$$

E não temos linhas à direita da diagonal principal da última linha.

Resumindo:

$$k = 1$$

$$l_{11} = a_{11}$$

$$l_{21} = a_{21} \quad u_{12} = a_{12} / l_{11}$$

$$l_{31} = a_{31} \quad u_{13} = a_{13} / l_{11}$$

$$k = 2$$

$$l_{22} = a_{22} - l_{21} * u_{12}$$

$$l_{32} = a_{32} - l_{31} * u_{12} \quad u_{23} = (a_{23} - l_{21} * u_{13}) / l_{22}$$

$$k = 3$$

$$l_{33} = a_{33} - l_{31} * u_{13} - l_{32} * u_{23}$$

Então, o algoritmo de decomposição segue esta sequência de cálculos:

- a) determinamos os elementos da primeira coluna de L e da primeira linha de U ; e
- b) determinamos os elementos da segunda coluna de L e da segunda linha de U ; e assim sucessivamente.

Obtidos L e U , devemos resolver $L * C = B$ determinando C por substituições sucessivas à frente:

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$\begin{cases} l_{11} * c_1 = b_1 \\ l_{21} * c_1 + l_{22} * c_2 = b_2 \\ l_{31} * c_1 + l_{32} * c_2 + l_{33} * c_3 = b_3 \end{cases} \Rightarrow \begin{cases} c_1 = (b_1) / l_{11} \\ c_2 = (b_2 - (l_{21} * c_1)) / l_{22} \\ c_3 = (b_3 - (l_{31} * c_1 + l_{32} * c_2)) / l_{33} \end{cases}$$

Note que o cálculo de C , do sistema $L * C = B$, segue o mesmo formato do cálculo dos elementos de U . Uma vez obtidos L , U e C , devemos resolver $U * X = C$ determinando X por retrossubstituições sucessivas:

$$\begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

$$\begin{cases} 1 * x_1 + u_{12} * x_2 + u_{13} * x_3 = c_1 \\ 0 * x_1 + 1 * x_2 + u_{23} * x_3 = c_2 \\ 0 * x_1 + 0 * x_2 + 1 * x_3 = c_3 \end{cases} \Rightarrow \begin{cases} x_1 = c_1 - (u_{12} * x_2 + u_{13} * x_3) \\ x_2 = c_2 - (u_{23} * x_3) \\ x_3 = c_3 \end{cases}$$

Sugerimos usar o armazenamento das colunas L e U sobrepostas na mesma área de memória de A , o que otimiza o armazenamento e as operações de pivotação, uma vez que a diagonal de U é unitária e não precisa ser armazenada:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \Rightarrow \begin{bmatrix} l_{11} & u_{12} & u_{13} \\ l_{21} & l_{22} & u_{23} \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

De uma forma geral, para sistemas de ordem n , teremos o seguinte formulário:

Primeiro passo: $k = 1$

$$l_{i1} = a_{i1} \quad \forall i = 1, 2, 3, \dots, n$$

$$u_{1j} = a_{1j} / l_{11} \quad \forall j = 2, 3, \dots, n$$

Segundo passo: $k = 2, 3, \dots, n - 1$

$$l_{ij} = a_{ij} - \sum_{r=1}^{j-1} l_{ir} * u_{rj} \quad \forall i = k, k + 1, \dots, n \text{ e } j = k$$

$$u_{ij} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{r=1}^{i-1} l_{ir} * u_{rj} \right) \quad \forall j = k + 1, \dots, n \text{ e } i = k$$

Terceiro e último passo: $k = n$

$$l_{ij} = a_{ij} - \sum_{r=1}^{j-1} l_{ir} * u_{rj} \quad i = j = n$$

O método de Crout com pivotação gera erros menores do que sem pivotação. O efeito da pivotação sobre o método de Crout é análogo ao seu efeito sobre o método de Gauss, apresentado na seção anterior, ou seja, reduz o acúmulo de erros de arredondamento, conforme o **Exemplo 2.5**.

Exemplo 2.5: resolva o sistema, a seguir, pelo método de Crout **com pivotamento parcial** (é imprescindível incluir o(s) vetor(es) B do sistema na troca de linhas da pivotação para não alterar as equações), usando 4 dígitos significativos e arredondamento ponderado.

$$\begin{cases} 0.448x_1 + 0.832x_2 + 0.193x_3 = 1 \\ 0.421x_1 + 0.784x_2 - 0.207x_3 = 2 \\ -0.319x_1 + 0.884x_2 + 0.279x_3 = 0 \end{cases}$$

Observações:

- a) o(s) vetor(es) de termos independentes B será(ão) colocado(s) à direita da matriz para que seja(m) pivotado(s) juntamente com as linhas de A ; e
- b) no algoritmo, os valores da matriz decomposta L e U serão armazenados na própria área de memória de A .

Na sequência do **Exemplo 2.5**, os valores da matriz decomposta L e U também são escritos na mesma matriz A , sendo diferenciados da matriz A original da seguinte forma:

- a) representação em negrito: valores novos já decompostos em L e U ; e
- b) representação normal: valores originais da matriz A .

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \text{ depois da decomposição } LU \rightarrow \begin{bmatrix} \mathbf{l}_{11} & \mathbf{u}_{12} & \mathbf{u}_{13} \\ \mathbf{l}_{21} & \mathbf{l}_{22} & \mathbf{u}_{23} \\ \mathbf{l}_{31} & \mathbf{l}_{32} & \mathbf{l}_{33} \end{bmatrix}$$

Solução:

Primeiro passo: na decomposição LU (método de Crout), correspondente ao primeiro passo ($k = 1$):

- a) Definimos a primeira coluna $j = 1$ da matriz L , l_{i1} para $i = 1, 2, 3$:
 $\rightarrow l_{i1} = a_{i1}$, $i = 1, 2, 3, \dots, n$ (L equivale à coluna original de A)

$$\begin{bmatrix} 0.448 & 0.832 & 0.193 \\ 0.421 & 0.784 & -0.207 \\ -0.319 & 0.884 & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

 C_1
 \uparrow
 C_1

resultando em

$$\begin{bmatrix} \mathbf{(0.448)} & 0.832 & 0.193 \\ \mathbf{0.421} & 0.784 & -0.207 \\ \mathbf{-0.319} & 0.884 & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

- b) Observamos que a pivotação parcial deve ocorrer depois do cálculo dos valores de l_{ij} na coluna $j = k$ e antes do cálculo dos valores u_{ij} da linha $i = k$, para que cada elemento l_{kk} , que acaba de ser calculado na diagonal principal, seja não nulo e também tenha o maior módulo possível. Para o primeiro passo ($k = 1$), a pivotação poderia ser feita ainda antes do cálculo de l_{i1} , pois $l_{i1} = a_{i1}$. A matriz l_{ij} , parcialmente calculada (em negrito), já se encontra pivotada neste exemplo:

$$\begin{bmatrix} \mathbf{(0.448)} & 0.832 & 0.193 \\ \mathbf{0.421} & 0.784 & -0.207 \\ \mathbf{-0.319} & 0.884 & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

- c) Definimos a primeira linha $i = 1$ da matriz U , u_{1j} para $j = 2, 3$:

$$\rightarrow u_{1j} = a_{1j} / l_{11}, \quad j = 2, 3, \dots, n$$

$$\begin{bmatrix} \mathbf{(0.448)} & 0.832 & 0.193 \\ \mathbf{0.421} & 0.784 & -0.207 \\ \mathbf{-0.319} & 0.884 & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \quad L_1 \leftarrow L_1 / l_{11}$$

resultando em

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ \mathbf{0.421} & 0.784 & -0.207 \\ \mathbf{-0.319} & 0.884 & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Segundo passo: na decomposição LU , correspondente ao segundo passo ($k=2$):

a) Definimos a coluna $j=2$ da matriz L , l_{i2} para $i=2, 3$:

$$\rightarrow l_{ij} = a_{ij} - \sum_{r=1}^{j-1} l_{ir} * u_{rj} \quad (i = k, k+1, \dots, n \text{ e } j = k)$$

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ \mathbf{0.421} & 0.784 & -0.207 \\ \mathbf{-0.319} & 0.884 & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

$$\begin{matrix} C_2 \\ \uparrow \end{matrix}$$

$$C_2 - l_{i1} * u_{12}$$

resultando em

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ \mathbf{0.421} & \mathbf{0.002203} & -0.207 \\ \mathbf{-0.319} & \mathbf{1.476} & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Observe que a pivotação parcial, correspondente ao segundo pivô ($k=2$), deve ocorrer depois do cálculo dos valores de l_{ij} na coluna $j=k$ e antes do cálculo dos valores u_{ij} da linha $i=k$, conforme mencionado no item (b) do **Primeiro passo**.

b) Procedemos com a busca parcial do maior módulo da coluna $j=2$ (busca a partir da segunda linha):

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ \mathbf{0.421} & \mathbf{0.002203} & -0.207 \\ -\mathbf{0.319} & \mathbf{(1.476)} & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \end{matrix}$$

Maior coeficiente em módulo encontrado na linha $i = 3$.

c) Trocamos linhas:

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ \mathbf{0.421} & \mathbf{0.002203} & -0.207 \\ -\mathbf{0.319} & \mathbf{(1.476)} & 0.279 \end{bmatrix} \begin{matrix} 1 \\ 2 \quad L_2 \leftarrow L_3 \\ 3 \quad L_3 \leftarrow L_2 \end{matrix}$$

resultando em

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ -\mathbf{0.319} & \mathbf{(1.476)} & 0.279 \\ \mathbf{0.421} & \mathbf{0.002203} & -0.207 \end{bmatrix} \begin{matrix} 1 \\ 3 \\ 2 \end{matrix}$$

d) Definimos a linha $i = 2$ da matriz U , u_{2j} para $j = 3$:

$$\rightarrow u_{ij} = \frac{1}{l_{ii}} \left(a_{ij} - \sum_{r=1}^{i-1} l_{ir} * u_{rj} \right) \quad (j = k + 1, \dots, n \text{ e } i = k)$$

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ -\mathbf{0.319} & \mathbf{(1.476)} & 0.279 \\ \mathbf{0.421} & \mathbf{0.002203} & -0.207 \end{bmatrix} \begin{matrix} 1 \\ 3 \quad L_2 \leftarrow (L_2 - l_{21} * u_{13}) / l_{22} \\ 2 \end{matrix}$$

resultando em

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ -\mathbf{0.319} & \mathbf{(1.476)} & \mathbf{0.2821} \\ \mathbf{0.421} & \mathbf{0.002203} & -0.207 \end{bmatrix} \begin{matrix} 1 \\ 3 \\ 2 \end{matrix}$$

Terceiro passo: na decomposição LU correspondente ao terceiro passo ($k = 3$):

a) Definimos a coluna $j = 3$ da matriz L , l_{i3} para $i = 3$:

$$\rightarrow l_{ij} = a_{ij} - \sum_{r=1}^{j-1} l_{ir} * u_{rj} \quad (i = j = n)$$

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ -\mathbf{0.319} & \mathbf{(1.476)} & \mathbf{0.2821} \\ \mathbf{0.421} & \mathbf{0.002203} & -\mathbf{0.207} \end{bmatrix} \begin{matrix} 1 \\ 3 \\ 2 \end{matrix}$$

$$C_3$$

$$\uparrow$$

$$C_3 - l_{31} * u_{13} - l_{32} * u_{23}$$

resultando em

$$\begin{bmatrix} \mathbf{0.448} & \mathbf{1.857} & \mathbf{0.4308} \\ -\mathbf{0.319} & \mathbf{(1.476)} & \mathbf{0.2821} \\ \mathbf{0.421} & \mathbf{0.002203} & -\mathbf{0.3890} \end{bmatrix} \begin{matrix} 1 \\ 3 \\ 2 \end{matrix}$$

Logo,

$$L = \begin{bmatrix} 0.448 & 0 & 0 \\ -0.319 & 1.476 & 0 \\ 0.421 & 0.002203 & -0.3890 \end{bmatrix} \text{ e } \begin{bmatrix} 1 & 1.857 & 0.4308 \\ 0 & 1 & 0.2821 \\ 0 & 0 & 1 \end{bmatrix}$$

Decomposta a matriz A , vamos agora obter a solução do sistema através da solução de dois sistemas triangulares:

$$L * C = B \Rightarrow \begin{cases} 0.448c_1 + 0c_2 + 0c_3 = 1 \\ -0.319c_1 + 1.476c_2 + 0c_3 = 0 \\ 0.421c_1 + 0.002203c_2 - 0.3890c_3 = 2 \end{cases}$$

Observe que o vetor B do sistema também trocou de linhas em razão da pivotação:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} \begin{array}{l} L_2 \leftarrow L_3 \\ L_3 \leftarrow L_2 \end{array} \Rightarrow \begin{bmatrix} b_1 \\ b_3 \\ b_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}$$

Aplicando substituições sucessivas à frente, a partir de c_1 , temos:

$$\begin{array}{ll} c_1 = (1)/0.448 & c_1 = 2.232 \\ c_2 = (0 - (-0.319 c_1))/1.476 & \Rightarrow c_2 = 0.4824 \\ c_3 = (2 - (0.421 c_1 + 0.002203 c_2))/(-0.3890) & c_3 = -2.723 \end{array}$$

Resolvendo

$$U * X = C \Rightarrow \begin{cases} 1x_1 + 1.857x_2 + 0.4308x_3 = 2.232 \\ 0x_1 + 1x_2 + 0.2821x_3 = 0.4824 \\ 0x_1 + 0x_2 + 1x_3 = -2.723 \end{cases}$$

E aplicando o processo de retrossubstituições sucessivas a partir de x_3 :

$$\begin{array}{ll} x_3 = -2.723 & x_3 = -2.723 \\ x_2 = (0.4824 - 0.2821 x_3) & \Rightarrow x_2 = 1.251 \\ x_1 = (2.232 - (1.857 x_2 + 0.4308 x_3)) & x_1 = 1.082 \end{array}$$

Então, a solução obtida é:

$$S = \{1.082, 1.251, -2.723\}$$

Para determinar o erro da solução anterior, obtemos a solução exata calculada com 16 dígitos significativos e pivotação parcial, resultando em:

$$S_{\text{exato}} = \{1.08321566094182, 1.25034738133745, -2.72315874287407\}$$

Daí o erro de arredondamento da solução S será:

$$\text{Erro} = [S - S_{\text{exato}}]$$

$$\text{Erro } x_1 = |1.082 - 1.08321566094182| \cong 1.22e-03$$

$$\text{Erro } x_2 = |1.251 - 1.25034738133745| \cong 6.53e-04$$

$$\text{Erro } x_3 = |-2.723 - (-2.72315874287407)| \cong 1.59e-04$$

Podemos ver que os erros da solução S obtida com **pivotação parcial** são compatíveis com a precisão de 4 dígitos significativos adotados, embora na primeira equação ainda ocorra uma propagação de erros para o terceiro dígito fracionário.

Exemplo 2.6: resolva os três sistemas, a seguir, pelo método de Crout com **pivotamento parcial** (é imprescindível incluir o(s) vetor(es) B do sistema na troca de linhas da pivotação) e com 4 dígitos significativos:

$$\begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 = 1 \\ x_1 + 2x_2 - 3x_3 - 4x_4 = 0 \\ -x_1 + 0x_2 + 2x_3 + 3x_4 = 1 \\ x_1 - 4x_2 - x_3 + x_4 = 2 \end{cases} \quad \begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 = 1 \\ x_1 + 2x_2 - 3x_3 - 4x_4 = 0 \\ -x_1 + 0x_2 + 2x_3 + 3x_4 = 1 \\ x_1 - 4x_2 - x_3 + x_4 = -2 \end{cases} \quad \begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 = -1 \\ x_1 + 2x_2 - 3x_3 - 4x_4 = 0 \\ -x_1 + 0x_2 + 2x_3 + 3x_4 = 1 \\ x_1 - 4x_2 - x_3 + x_4 = 2 \end{cases}$$

Observe que esse exemplo demonstra a real aplicação do método de decomposição LU , que ocorre quando vários sistemas têm a mesma matriz de coeficientes A , mas com diferentes vetores de termos independentes B . Portanto, com as mesmas matrizes L e U , decompostas de A , podemos resolver os três sistemas diferentes.

Agora, vamos gerar uma matriz com os coeficientes das incógnitas A concatenada com os três vetores B^1, B^2, B^3 de termos independentes, para que, na pivotação parcial, a troca de linhas corresponda a trocas de equações:

$$\begin{array}{c} A \\ \left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & -3 & -4 \\ -1 & 0 & 2 & 3 \\ 1 & -4 & -1 & 1 \end{array} \right] : \left[\begin{array}{c} B^1 \\ B^2 \\ B^3 \end{array} \right] \left[\begin{array}{c} -1 \\ 0 \\ 1 \\ 2 \end{array} \right] \end{array} \quad \text{ou} \quad \begin{array}{c} A \\ \left[\begin{array}{cccc} 1 & 2 & 3 & 4 \\ 1 & 2 & -3 & -4 \\ -1 & 0 & 2 & 3 \\ 1 & -4 & -1 & 1 \end{array} \right] : \left[\begin{array}{c} B \\ B \\ B \end{array} \right] \left[\begin{array}{c} -1 \\ 0 \\ 1 \\ 2 \end{array} \right] \end{array}$$

Solução:

Primeiro passo: $k = 1$

- a) Definimos a primeira coluna $j = 1$ da matriz L , l_{i1} ($i = 1, 2, 3, 4$), que é igual à matriz A original:

$$\begin{bmatrix} \mathbf{1} & 2 & 3 & 4 \\ \mathbf{1} & 2 & -3 & -4 \\ -\mathbf{1} & 0 & 2 & 3 \\ \mathbf{1} & -4 & -1 & 1 \end{bmatrix} : \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ -2 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

- b) Lembramos que a pivotação parcial, correspondente ao primeiro pivô ($k = 1$), deve ocorrer depois do cálculo dos valores de l_{ik} . Nesse caso, a matriz já se encontra pivotada:

$$\begin{bmatrix} \mathbf{(1)} & 2 & 3 & 4 \\ \mathbf{1} & 2 & -3 & -4 \\ -\mathbf{1} & 0 & 2 & 3 \\ \mathbf{1} & -4 & -1 & 1 \end{bmatrix} : \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ -2 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

- c) Definimos a primeira linha $i = 1$ da matriz U , u_{1j} ($j = 2, 3, 4$), que é igual à matriz A dividida pelo pivô $l_{11} = 1$:

$$\begin{bmatrix} \mathbf{(1)} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & 2 & -3 & -4 \\ -\mathbf{1} & 0 & 2 & 3 \\ \mathbf{1} & -4 & -1 & 1 \end{bmatrix} : \begin{bmatrix} 1 \\ 0 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ -2 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

Segundo passo: $k = 2$

- a) Definimos a segunda coluna $j = 2$ da matriz L , l_{i2} ($i = 2, 3, 4$), que é igual à matriz A menos um somatório de produtos cruzados da sua linha e coluna, conforme demonstramos ao lado das matrizes:

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & \mathbf{0} & -\mathbf{3} & -\mathbf{4} \\ -\mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{3} \\ \mathbf{1} & (-\mathbf{6}) & -\mathbf{1} & \mathbf{1} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \\ -\mathbf{2} \end{bmatrix} \begin{bmatrix} -\mathbf{1} \\ \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \end{bmatrix}, \text{ em que } \begin{array}{l} \mathbf{0} = \mathbf{2} - (\mathbf{1} * \mathbf{2}) \\ \mathbf{2} = \mathbf{0} - (-\mathbf{1} * \mathbf{2}) \\ -\mathbf{6} = -\mathbf{4} - (\mathbf{1} * \mathbf{2}) \end{array}$$

- b) Observamos que a pivotação parcial, correspondente ao segundo pivô ($k = 2$), deve ocorrer depois do cálculo dos valores de l_{ik} . Nesse caso, a linha 4 é trocada com a linha 2, em A e B :

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & (-\mathbf{6}) & -\mathbf{1} & \mathbf{1} \\ -\mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{3} \\ \mathbf{1} & \mathbf{0} & -\mathbf{3} & -\mathbf{4} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ -\mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{1} \\ \mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix}$$

- c) Definimos a segunda linha $i = 2$ da matriz U , u_{2j} ($j = 3, 4$), que é igual a da matriz A , menos um somatório de produtos cruzados da sua linha e coluna, dividido pelo pivô $l_{22} = -6$, conforme demonstramos abaixo:

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & (-\mathbf{6}) & \mathbf{0.6667} & \mathbf{0.5} \\ -\mathbf{1} & \mathbf{2} & \mathbf{2} & \mathbf{3} \\ \mathbf{1} & \mathbf{0} & -\mathbf{3} & -\mathbf{4} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ -\mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} -\mathbf{1} \\ \mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix}$$

em que

$$\mathbf{0.6667} = [-\mathbf{1} - (\mathbf{1} * \mathbf{3})] / (-\mathbf{6})$$

$$\mathbf{0.5} = [1 - (\mathbf{1} * \mathbf{4})] / (-\mathbf{6})$$

Terceiro passo: $k = 3$

- a) Definimos a terceira coluna $j = 3$ da matriz L , l_{i3} ($i = 3, 4$):

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & \mathbf{-6} & \mathbf{0.6667} & \mathbf{0.5} \\ \mathbf{-1} & \mathbf{2} & \mathbf{3.667} & \mathbf{3} \\ \mathbf{1} & \mathbf{0} & \mathbf{(-6)} & \mathbf{-4} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{-2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{-1} \\ \mathbf{2} \\ \mathbf{1} \\ \mathbf{0} \end{bmatrix}$$

em que

$$\mathbf{3.667} = 2 - (-1 * 3 + 2 * \mathbf{0.6667})$$

$$\mathbf{-6} = -3 - (1 * 3 + 0 * \mathbf{0.6667})$$

- b) Observamos que a pivotação parcial, correspondente ao terceiro pivô ($k = 3$), ocorre depois do cálculo dos valores de l_{ik} . Nesse caso, a linha 4 é trocada com a linha 3, em A e B :

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & \mathbf{-6} & \mathbf{0.6667} & \mathbf{0.5} \\ \mathbf{1} & \mathbf{0} & \mathbf{(-6)} & \mathbf{-4} \\ \mathbf{-1} & \mathbf{2} & \mathbf{3.667} & \mathbf{3} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{-2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{-1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

- c) Definimos a terceira linha $i = 3$ da matriz U , u_{3j} ($j = 4$):

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & \mathbf{-6} & \mathbf{0.6667} & \mathbf{0.5} \\ \mathbf{1} & \mathbf{0} & \mathbf{(-6)} & \mathbf{1.333} \\ \mathbf{-1} & \mathbf{2} & \mathbf{3.667} & \mathbf{3} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{-2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{-1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

em que

$$\mathbf{1.333} = [-4 - (1 * 4 + 0 * \mathbf{0.5})] / (-6)$$

Quarto passo: $k = 4$

a) Definimos a quarta coluna $j = 4$ da matriz L , l_{i4} ($i = 4$):

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & \mathbf{(-6)} & \mathbf{0.6667} & \mathbf{0.5} \\ \mathbf{1} & \mathbf{0} & \mathbf{(-6)} & \mathbf{1.333} \\ \mathbf{-1} & \mathbf{2} & \mathbf{3.667} & \mathbf{1.112} \end{bmatrix} : \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{1} \\ \mathbf{-2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{-1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix}$$

em que

$$\mathbf{1.112} = \mathbf{3} - (\mathbf{-1} * \mathbf{4} + \mathbf{2} * \mathbf{0.5} + \mathbf{3.667} * \mathbf{1.333})$$

Observe que a decomposição LU é feita uma única vez, e agora podemos aplicá-la aos três sistemas, fazendo as substituições para cada B :

$$A * X = B^1$$

$$A * X = B^2$$

$$A * X = B^3$$

Primeiro sistema – $A * X = B^1$:

Resolvendo $L * C = B^1$ (observe que B^1 foi pivotado), temos

$$\begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{-6} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{0} & \mathbf{-6} & \mathbf{0} \\ \mathbf{-1} & \mathbf{2} & \mathbf{3.667} & \mathbf{1.112} \end{bmatrix} * \begin{bmatrix} \mathbf{c_1} \\ \mathbf{c_2} \\ \mathbf{c_3} \\ \mathbf{c_4} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{c_1} \\ \mathbf{c_2} \\ \mathbf{c_3} \\ \mathbf{c_4} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \mathbf{-0.1667} \\ \mathbf{0.1667} \\ \mathbf{1.549} \end{bmatrix}$$

Resolvendo $U * X = C^1$, temos

$$\begin{bmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{0} & \mathbf{1} & \mathbf{0.6667} & \mathbf{0.5} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1.333} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} * \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \\ \mathbf{x_4} \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ \mathbf{-0.1667} \\ \mathbf{0.1667} \\ \mathbf{1.549} \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \mathbf{x_3} \\ \mathbf{x_4} \end{bmatrix} = \begin{bmatrix} \mathbf{-0.1516} \\ \mathbf{0.3248} \\ \mathbf{-1.898} \\ \mathbf{1.549} \end{bmatrix}$$

Segundo sistema – $A * X = B^2$:

Resolvendo $L * C = B^2$, temos

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -6 & 0 & 0 \\ 1 & 0 & -6 & 0 \\ -1 & 2 & 3.667 & 1.112 \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \\ 0.1667 \\ 0.3495 \end{bmatrix}$$

Resolvendo $U * X = C^2$, temos

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0.6667 & 0.5 \\ 0 & 0 & 1 & 1.333 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0.5 \\ 0.1667 \\ 0.3495 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -0.5498 \\ 0.5247 \\ -0.2992 \\ 0.3495 \end{bmatrix}$$

Terceiro sistema – $A * X = B^3$:

Resolvendo $L * C = B^3$, temos

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -6 & 0 & 0 \\ 1 & 0 & -6 & 0 \\ -1 & 2 & 3.667 & 1.112 \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ 0 \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} -1 \\ -0.5 \\ -0.1667 \\ 1.449 \end{bmatrix}$$

Resolvendo $U * X = C^3$, temos

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 0 & 1 & 0.6667 & 0.5 \\ 0 & 0 & 1 & 1.333 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -1 \\ -0.5 \\ -0.1667 \\ 1.449 \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -0.8504 \\ 0.1742 \\ -2.098 \\ 1.449 \end{bmatrix}$$

Confira o algoritmo de decomposição LU , de Crout, aplicado ao **Exemplo 2.6** de três sistemas lineares com opção de pivotação parcial, no *link* <<http://sergiopeters.prof.ufsc.br/algoritmoslivro/>> sob o título **Cap2LUCrout.m**.

No algoritmo de decomposição LU , de Crout, o processo de pivotação parcial está inserido depois do cálculo dos valores de l_{ik} e imediatamente antes do cálculo dos valores de u_{kj} .

Na Tabela 2.1, retomamos o número de operações em ponto flutuante envolvidas em cada um dos métodos numéricos diretos apresentados até este momento.

Tabela 2.1 – Comparativo entre as operações em ponto flutuante em métodos diretos

MÉTODOS	ORDEM DO NÚMERO DE OPERAÇÕES
Gauss	$O(2n^3 / 3)$
Gauss-Jordan	$O(4n^3 / 3)$
Inversão	$O(8n^3 / 3)$
Crout	$O(2n^3 / 3)$

Fonte: Elaboração própria.

Essa ordem do número de operações aritméticas define a complexidade do algoritmo de um método numérico. Os métodos de inversão de matrizes e Gauss-Jordan são computacionalmente os menos eficientes, pois envolvem o maior número de operações aritméticas, enquanto os métodos de Gauss e Crout são os que envolvem o menor esforço computacional, porém todos são de ordem $O(n^3)$ operações aritméticas.

Tabela 2.2 – Comparativo do número de operações em ponto flutuante realizadas pelos dois métodos mais eficientes

OPERAÇÕES	MÉTODO DE CROUT	MÉTODO DE GAUSS
Adição e subtração	$(2n^3 + 9n^2 - 5n - 6) / 6$	$(2n^3 + 3n^2 + n - 6) / 6$
Multiplicação	$(2n^3 + 3n^2 - 5n) / 6$	$(2n^3 + 3n^2 - 5n) / 6$
Divisão	$(n^2 + n) / 2$	$(n^2 + n) / 2$
Total	$(4n^3 + 15n^2 - 7n - 6) / 6$	$(4n^3 + 9n^2 - n - 6) / 6$

Fonte: Elaboração própria.

No método de Crout, temos um pouco mais de adições e subtrações do que no método de Gauss, mas é uma diferença de ordem inferior, $6n^2$, e as outras operações são de mesmo número.

Assim, quando temos que resolver **vários sistemas** com a mesma matriz de coeficientes A e com m **diferentes vetores** de termos independentes do tipo $A * X^m = B^m$, podemos:

- Utilizar o método de Crout para obter as matrizes decompostas L e U uma única vez e efetuar as m substituições sucessivas $L * C^m = B^m$ e $U * X^m = C^m$ tantas vezes quantas forem necessárias para obter as respectivas soluções C^m e X^m , correspondentes a cada B^m . Nesses casos, temos apenas um custo de $(2 * n^2 - n)$ operações, referente à substituição dupla para obter cada solução X^m , e um custo de armazenamento de duas matrizes, L e U que podem ser sobrepostas em uma única matriz, conforme feito no algoritmo de decomposição LU , de Crout, apresentado.
- Obter a matriz inversa A^{-1} uma única vez e aplicar apenas o produto final $X^m = A^{-1} * B^m$ para cada sistema m . O custo dessa etapa de operações do produto da inversa é igual ao das duas substituições sucessivas com as matrizes L e U , e custo de armazenamento de uma única matriz.
- Aplicar o método de eliminação de Gauss a uma matriz estendida com todas as colunas de B^m , incluindo-as conforme esta matriz com dados do **Exemplo 2.6**:

$$\left[\begin{array}{cccc|ccc} 1 & 2 & 3 & 4 & 1 & 1 & -1 \\ 1 & 2 & -3 & -4 & 0 & 0 & 0 \\ -1 & 0 & 2 & 3 & 1 & 1 & 1 \\ 1 & -4 & -1 & 1 & 2 & -2 & 2 \end{array} \right]$$

$$\Rightarrow \left[\begin{array}{cccc|ccc} 1 & 2 & 3 & 4 & 1 & 1 & -1 \\ 0 & -6 & -4 & -3 & 1 & -3 & 3 \\ 0 & 0 & -4 & -8 & -1 & -1 & 1 \\ 0 & 0 & 0 & 1.1111... & 1.7222... & 0.3888... & 1.6111... \end{array} \right]$$

Depois de obter a matriz triangularizada dos coeficientes, que também é uma matriz U triangular superior, efetuamos as retrossubstituições sucessivas $U * X^m = B^m$ para obter a solução X^m correspondente a

cada B^m escalonado, com custo de n^2 operações, como nas seguintes soluções, aqui obtidas em precisão *double*:

$$X = \begin{bmatrix} -0.150000 & -0.500000 & -0.850000 \\ 0.325000 & 0.525000 & 0.175000 \\ -1.900000 & -0.300000 & -2.100000 \\ 1.550000 & 0.350000 & 1.450000 \end{bmatrix}$$

Nesse caso, não há possibilidade de reuso da matriz triangularizada, como podemos fazer com a matriz decomposta ou com a matriz inversa. Essa última metodologia pode ser indicada para resolver sistemas de equações lineares que estejam dentro de um processo iterativo que atualiza os coeficientes das equações do sistema a cada iteração, não havendo necessidade de armazenar a matriz transformada para reuso posterior.

2.1.5 Método de Cholesky

No caso particular de sistemas com matrizes de coeficientes **simétricas** e **positivas definidas**, podemos obter uma simplificação no método de decomposição para obter as matrizes L e U , chamado de método de Cholesky ($O(n^3/6)$ operações). Com esse método, podemos obter a matriz U diretamente pela transposta de L , ou seja, $U = L^T$, ($A = L * L^T$), segundo as seguintes equações:

a) operações no primeiro passo: $k = 1$

$$l_{11} = \sqrt{a_{11}}$$

$$l_{i1} = a_{i1} / l_{11} \quad \forall i = 2, 3, \dots, n$$

b) operações nos passos intermediários: $k = 2, 3, \dots, n - 1$

$$l_{kk} = \left[a_{kk} - \sum_{r=1}^{k-1} l_{kr}^2 \right]^{1/2}$$

$$l_{ik} = \frac{1}{l_{kk}} \left(a_{ik} - \sum_{r=1}^{k-1} l_{ir} * l_{kr} \right) \quad \forall i = k+1, \dots, n$$

c) operações no último passo: $k = n$

$$l_{nn} = \left[a_{nn} - \sum_{r=1}^{n-1} l_{nr}^2 \right]^{1/2}$$

O método de Cholesky também é utilizado para verificar eficientemente se uma dada matriz simétrica A é positiva definida ou não; pois, na decomposição anterior, se não aparecerem radicandos negativos, então a matriz simétrica é positiva definida.

No [algoritmo de Cholesky](#), o processo de pivotação não pode ser aplicado, para não alterar a simetria da matriz, conforme podemos conferir no arquivo `Cap2LUCholesky.m` do [Caderno de Algoritmos](#).



No algoritmo de Cholesky, por motivos estritamente didáticos, temos a opção de imprimir a matriz U para visualização, mas essa matriz U nem precisa ser armazenada uma vez que $U = L^T$.

Na sequência, vamos apresentar metodologias para resolver sistemas representados em matrizes do tipo banda.

2.1.6 Solução de sistemas com a matriz de coeficientes do tipo banda

Alguns modelos matemáticos envolvem sistemas de equações lineares especias deste tipo:

$$\begin{bmatrix} a_{11} & a_{12} & 0 & 0 & \cdots & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \cdots & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & \cdots & \cdots & a_{(n-1,n-2)} & a_{(n-1,n-1)} & a_{(n-1,n)} \\ 0 & 0 & \cdots & \cdots & 0 & a_{(n,n-1)} & a_{(n,n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} \quad (4a)$$

Nesse sistema, a matriz dos coeficientes A é constituída de apenas três faixas de elementos não nulos: a faixa da diagonal principal e as faixas supra

Primeiro passo: $k = 1$, eliminação do t_2 :

$$\left[\begin{array}{cccccccc} r_1 & d_1 & & & & & & \vdots & b_1 \\ t_2 & r_2 & d_2 & & & & & \vdots & b_2 \\ & t_3 & r_3 & d_3 & & & & \vdots & \vdots \\ & & \ddots & \ddots & \ddots & & & \vdots & \vdots \\ & & & \ddots & \ddots & \ddots & & \vdots & \vdots \\ & & & & t_{n-1} & r_{n-1} & d_{n-1} & \vdots & b_{n-1} \\ & & & & & t_n & r_n & \vdots & b_n \end{array} \right] L_2 \leftarrow L_2 - \frac{t_2}{r_1} L_1$$

$$\left[\begin{array}{cccccccc} r_1 & d_1 & & & & & & \vdots & b_1 \\ 0 & r_2 & d_2 & & & & & \vdots & b_2 \\ & t_3 & r_3 & d_3 & & & & \vdots & \vdots \\ & & \ddots & \ddots & \ddots & & & \vdots & \vdots \\ & & & \ddots & \ddots & \ddots & & \vdots & \vdots \\ & & & & t_{n-1} & r_{n-1} & d_{n-1} & \vdots & b_{n-1} \\ & & & & & t_n & r_n & \vdots & b_n \end{array} \right]$$

Nas quais devem ser operadas apenas as colunas de r_2 e b_2 : $r_2 = r_2 - \frac{t_2}{r_1} d_1$ e $b_2 = b_2 - \frac{t_2}{r_1} b_1$, pois d_2 não é alterado, $d_2 = d_2 - \frac{t_2}{r_1} 0$, e nenhuma outra coluna da linha $i = 2$ é alterada, pois $0 = 0 - \frac{t_2}{r_1} 0$.

Segundo passo: generalizando para um passo k qualquer que altera apenas a linha $i = k + 1$, temos:

$$\left[\begin{array}{cccccccc} r_1 & d_1 & & & & & & \vdots & b_1 \\ 0 & r_2 & d_2 & & & & & \vdots & b_2 \\ & \ddots & \ddots & \ddots & & & & \vdots & \vdots \\ & & 0 & r_{i-1} & d_{i-1} & & & \vdots & b_{i-1} \\ & & & t_i & r_i & d_i & & \vdots & b_i \\ & & & & \ddots & \ddots & \ddots & \vdots & \vdots \\ & & & & & t_n & r_n & \vdots & b_n \end{array} \right] L_i \leftarrow L_i - \frac{t_i}{r_{i-1}} L_{i-1}$$

Então, geramos a matriz triangularizada a seguir:

$$\begin{bmatrix} r_1 & d_1 & & & & \vdots & b_1 \\ 0 & r_2 & d_2 & & & \vdots & b_2 \\ & \ddots & \ddots & \ddots & & \vdots & \vdots \\ & & 0 & r_{i-1} & d_{i-1} & \vdots & b_{i-1} \\ & & & 0 & r_i & d_i & \vdots & b_i \\ & & & & \ddots & \ddots & \ddots & \vdots \\ & & & & & 0 & r_n & \vdots & b_n \end{bmatrix} \quad (5)$$

Em que os novos coeficientes são $r_i = r_i - \frac{t_i}{r_{i-1}} d_{i-1}$ e $b_i = b_i - \frac{t_i}{r_{i-1}} b_{i-1}$, para $i = 2, 3, \dots, n$.

Assim, podemos gerar um algoritmo simples para efetuar a triangularização da matriz de coeficientes. Genericamente, para $i = 2, 3, \dots, n$. (equivalente a $k = 1$ até $n - 1$), temos:

$$aux = \frac{t_i}{r_{i-1}}$$

$$r_i = r_i - aux * d_{i-1}$$

$$b_i = b_i - aux * b_{i-1}$$

Com r_1 , d_1 e b_1 inalterados na primeira linha.

Então, por retrossubstituições sucessivas aplicadas sobre as equações simplificadas, dadas pela eq. (5), obtemos:

$$x_n = b_n / r_n$$

Para $i = n - 1, n - 2, \dots, 2, 1$, temos:

$$x_i = (b_i - d_i * x_{i+1}) / r_i$$

Dessa forma, obtemos a solução do sistema tridiagonal com o mínimo de operações possíveis manipulando apenas os elementos não nulos.

Nesses casos, o pivotamento de linhas ou colunas não deve ser aplicado, pois isso alteraria a estrutura em forma de banda da matriz. Esses sistemas nunca têm diagonal principal nula, pois são sistemas resultantes de modelos matemáticos de discretização de domínios nos quais são relacionados um ponto central i , com seus vizinhos anterior $i - 1$ e posterior $i + 1$, como no método das diferenças finitas (BURDEN; FAIRES, 2011), que não faz parte do escopo deste livro.

Esse algoritmo requer apenas $8n - 7$ operações, sendo $5(n - 1)$ operações para a eliminação e $3(n - 1) + 1$ operações para o procedimento de retrossubstituições.

Exemplo 2.7: resolva o sistema de equações, a seguir, de forma otimizada:

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 1 & 1 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 1 & -1 & 1 & 0 & \vdots & 2 \\ 0 & 0 & -1 & 1 & 1 & \vdots & -1 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix}$$

Solução:

a) Triangularização baseada no pivô $k = 1$ opera a linha $i = 2$:

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 1 & 1 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 1 & -1 & 1 & 0 & \vdots & 2 \\ 0 & 0 & -1 & 1 & 1 & \vdots & -1 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix} L_2 \leftarrow L_2 - \frac{1}{1}L_1$$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 0 & 2 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 1 & -1 & 1 & 0 & \vdots & 2 \\ 0 & 0 & -1 & 1 & 1 & \vdots & -1 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix}$$

b) Triangularização baseada no pivô $k = 2$ opera a linha $i = 3$:

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 0 & 2 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 1 & -1 & 1 & 0 & \vdots & 2 \\ 0 & 0 & -1 & 1 & 1 & \vdots & -1 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix} L_3 \leftarrow L_3 - \frac{1}{2}L_2$$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 0 & 2 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 0 & -0.5 & 1 & 0 & \vdots & 1.5 \\ 0 & 0 & -1 & 1 & 1 & \vdots & -1 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix}$$

c) Triangularização baseada no pivô $k = 3$ opera a linha $i = 4$:

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 0 & 2 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 0 & -0.5 & 1 & 0 & \vdots & 1.5 \\ 0 & 0 & -1 & 1 & 1 & \vdots & -1 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix} L_4 \leftarrow L_4 - \frac{(-1)}{(-0.5)}L_3$$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 & \vdots & 0 \\ 0 & 2 & -1 & 0 & 0 & \vdots & 1 \\ 0 & 0 & -0.5 & 1 & 0 & \vdots & 1.5 \\ 0 & 0 & 0 & -1 & 1 & \vdots & -4 \\ 0 & 0 & 0 & -1 & 2 & \vdots & -2 \end{bmatrix}$$

d) Triangularização baseada no pivô $k = 4$ opera a linha $i = 5$:

$$\left[\begin{array}{ccccc|c} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 1 \\ 0 & 0 & -0.5 & 1 & 0 & 1.5 \\ 0 & 0 & 0 & -1 & 1 & -4 \\ 0 & 0 & 0 & -1 & 2 & -2 \end{array} \right] L_5 \leftarrow L_5 - \frac{(-1)}{(-1)} L_4$$

$$\left[\begin{array}{ccccc|c} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 2 & -1 & 0 & 0 & 1 \\ 0 & 0 & -0.5 & 1 & 0 & 1.5 \\ 0 & 0 & 0 & -1 & 1 & -4 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{array} \right]$$

e) Retrossubstituições:

$$\begin{cases} 1 x_5 = 2 \\ -1 x_4 + 1 x_5 = -4 \\ -0.5 x_3 + 1 x_4 = 1.5 \\ 2 x_2 - 1 x_3 = 1 \\ 1 x_1 - 1 x_2 = 0 \end{cases}$$

Então, a solução obtida é:

$S = \{5, 5, 9, 6, 2\}$ (temos uma solução exata, pois não houve arredondamentos).

No algoritmo de Gauss para matriz tridiagonal (**Cap2tridiagonal.m**), apresentado no **Caderno de Algoritmos**, os valores de t_i e d_i foram preenchidos pelo símbolo *NaN* (*Not a Number*), para completar as n posições dos vetores t_i e d_i , e também para chamar atenção que esses valores não existem no sistema de equações e não podem ser utilizados, mas poderia ser qualquer outro valor numérico, como o zero.

Podemos estender esses métodos específicos para matrizes bandas com mais faixas, como a *matriz pentadiagonal*⁹. Também podemos generalizar a otimização de métodos eliminativos para qualquer *matriz esparsa não estruturada*⁹ propondo um algoritmo que armazene as alterações necessárias em um sistema ao longo de cada passo do método eliminativo, na forma de um mapeamento de índices de cada um desses passos, e, depois, aplicando o método de eliminação com esse mapeamento predefinido para operar apenas os coeficientes não nulos necessários por quantas vezes forem necessárias.



Temos um exemplo de sistema com matriz pentadiagonal no **Exercício 2.19**.



Confira outros exemplos de matrizes esparsas não estruturadas em faixas e um algoritmo de escalonamento otimizado genérico para suas soluções na seção **Complementando...** ao final deste capítulo.

A seguir, vamos apresentar uma característica inerente a alguns sistemas lineares que, combinada com erros de arredondamento nos coeficientes do sistema, pode afetar a confiabilidade da solução.

2.1.7 Sistemas lineares mal condicionados

Consideremos o seguinte sistema linear:

$$\begin{cases} x_1 + 3x_2 = 4 \\ x_1 + 3.00001x_2 = 4.00001 \end{cases}$$

cuja solução exata é $S = \{1, 1\}$.

Tomemos agora um sistema de equações, derivado do sistema dado, que sofreu uma pequena perturbação em dois de seus coeficientes, impondo uma variação ao coeficiente a_{22} de 3.00001 para 2.99999 e ao b_2 de 4.00001 para 4.00002, conforme segue:

$$\begin{cases} x_1 + 3x_2 = 4 \\ x_1 + 2.99999x_2 = 4.00002 \end{cases}$$

cuja solução exata é $\bar{S} = \{10, -2\}$.

Note que essa pequena variação em dois coeficientes, da ordem de 0.025%, acarreta uma variação enorme na solução do sistema da ordem de 900%.

Classificamos esses sistemas altamente sensíveis a variações nos seus coeficientes como sistemas **mal condicionados**.

Imagine agora que essas pequenas variações são devidas a erros de arredondamento, que certamente estão presentes em todos os métodos diretos, no armazenamento e manipulação dos coeficientes das equações. Então, nesses casos, podemos obter soluções irreais dos sistemas originais, uma vez que as soluções são obtidas em sistemas simplificados, por operações sucessivas de eliminação ou decomposição, que inevitavelmente acumulam pequenas alterações nos coeficientes das matrizes simplificadas.

Por isso, é necessário tomar cuidados especiais na resolução desses sistemas, devido à grande sensibilidade da solução em relação a variações nos seus coeficientes. Devemos primeiramente identificar o sistema de equações mal condicionado, de preferência antes de resolver o sistema. Para tal, podemos recorrer a alguns critérios, como:

- a) Se o determinante da matriz A for considerado pequeno; ou se o determinante normalizado de A , denotado por $\|\det(A)\|$, for pequeno:

$$\|\det(A)\| = \frac{|\det(A)|}{\prod_{i=1}^n \alpha_i} \ll 1 \quad (6a)$$

Em que $\alpha_i = \sqrt{\sum_{j=1}^n a_{ij}^2} \quad \forall i \in \{1, 2, \dots, n\}$, e α_i mede a magnitude de cada linha i e deve ser calculado sobre a **matriz original**.

Então, se $\|\det(A)\| \ll 1$, o sistema é considerado **mal condicionado**. Na prática, adotamos um limite conservativo, por exemplo, se $\|\det(A)\| < 0.01$, consideramos conservativamente que o sistema $A * X = B$ é mal condicionado e podemos adotar as devidas precauções.



Os símbolos \ll e \gg representam, respectivamente, muito menor e muito maior.

b) Se o número de condicionamento $Cond(A)$ for considerado grande, ou seja:

$$Cond(A) = \|A\|_{\infty} * \|A^{-1}\|_{\infty} \gg 1 \quad (6b)$$

Então, se $Cond(A) \gg 1$, o sistema é considerado mal condicionado.

A norma infinita de A , denotada por $\|A\|_{\infty}$, é definida da seguinte forma:

$$\|A\|_{\infty} = \underset{1 \leq i \leq n}{Max} \left\{ \sum_{j=1}^n |a_{ij}| \right\}$$

e corresponde ao maior valor entre as somas dos módulos dos elementos de cada linha da matriz A . Também, para efeitos práticos, podemos estabelecer que, se $Cond(A) > 100$, então o sistema $A * X = B$ é conservativamente mal condicionado.

A dificuldade de aplicação desse último critério está na obtenção do valor da matriz inversa A^{-1} , cujo custo é aproximadamente o triplo do custo do método de eliminação gaussiana, enquanto o primeiro critério precisa apenas do determinante de A , que podemos obter através do produto da diagonal principal da matriz triangularizada (no método de Gauss, $\det(A) = \prod_{i=1}^n a_{ii}$, em que os a_{ii} são elementos da diagonal da matriz A triangularizada) ou através do produto da diagonal principal da matriz L decomposta de A (no método LU , de Crout, $\det(A) = \prod_{i=1}^n l_{ii} * 1$, em que os l_{ii} são elementos da diagonal de L e a diagonal de U é composta de valores unitários).

Uma desvantagem desse cálculo do determinante sobre a matriz simplificada pelas operações elementares de eliminação, ou decomposição, é que essa forma final simplificada já sofreu o acúmulo de erros de arredondamento sucessivos, não é exata e pode gerar alguma inconsistência em sistemas muito mal condicionados ou singulares. Por exemplo, em matrizes singulares, podemos avaliar um determinante residual, diferente de zero, gerando um determinante não nulo inconsistente para a matriz original singular. Logo, precisamos considerar determinantes residuais como nulos, como fizemos no algoritmo de retrossubstituições anterior, que leva em conta a possibilidade de uma das linhas da matriz ser **dependente** das demais.

Exemplo 2.8: avalie o condicionamento do sistema

$$\begin{cases} x_1 + 3x_2 = 4 \\ x_1 + 3.00001x_2 = 4.00001 \end{cases}$$

Solução:

Para tal, temos:

a) Verificar se $\|\det(A)\| \ll 1$:

$$|\det(A)| = 0.00001$$

$$\alpha_1 = \sqrt{1^2 + 3^2} = \sqrt{10}$$

$$\alpha_2 = \sqrt{1^2 + (3.00001)^2} \cong \sqrt{10}$$

$$\|\det(A)\| = \frac{0.00001}{\sqrt{10}\sqrt{10}} = 10^{-6}$$

Se $\|\det(A)\| = 10^{-6} \ll 1 \rightarrow$ mal condicionado

(conservativamente, consideraremos $\ll 1$ como < 0.01)

b) Ou verificar se $Cond(A) \gg 1$:

$$A = \begin{bmatrix} 1 & 3 \\ 1 & 3.00001 \end{bmatrix}$$

$$\|A\|_{\infty} = |1| + |3.00001| = 4.00001 \cong 4$$

A norma infinita $\|A\|_{\infty}$ é o valor máximo entre as somas dos módulos dos coeficientes de cada linha da matriz.

$$A^{-1} = \begin{bmatrix} 300000.9999980346 & -299999.9999980346 \\ -99999.9999993449 & 99999.9999993449 \end{bmatrix}$$

(obtida em computador com 16 dígitos significativos)

$$\|A^{-1}\|_{\infty} \cong |300001| + |300000| \cong 600001$$

$$\text{Cond}(A) = 4.00001 * 600001 \cong 2.4 * 10^6$$

Se $\text{Cond}(A) \cong 2.4 * 10^6 \gg 1 \Rightarrow$ confirma o mal condicionamento (conservativamente, consideraremos $\gg 1$ como > 100)

Depois de identificar um sistema como mal condicionado, é muito importante **minimizar os efeitos de alterações nos seus coeficientes**, então recomendamos:

- a) Usar um método de solução que não altere a forma original das equações, como é o caso dos **métodos iterativos**, que veremos mais adiante; porém, os métodos iterativos nem sempre convergem para a solução.
- b) Adotar algoritmos com o **menor número de operações aritméticas possível**, conseqüentemente menor acúmulo de arredondamentos. Sugerimos a aplicação de métodos diretos como a eliminação gaussiana ou a decomposição LU , uma vez que têm o menor número total de operações entre os métodos diretos apresentados, da ordem de $(O(n^3/3))$. Naturalmente, recomendamos que cada algoritmo seja internamente otimizado, e use o mínimo de operações aritméticas, como os que apresentamos neste livro.
- c) Usar algum processo de **pivotamento**, indispensável sempre que surgirem zeros nas posições dos pivôs durante a aplicação das operações sucessivas de eliminação ou decomposição. No entanto, nos casos específicos de sistemas mal condicionados, o processo de pivotação também é indispensável para minimizar a propagação dos efeitos dos arredondamentos.
- d) Utilizar a **precisão máxima** disponível na implementação dos cálculos envolvidos na resolução do sistema. É interessante testar o sistema de equações calculando resultados com precisões diferentes, como simples e dupla (por exemplo em C : *float* e *double*), para fazer uma comparação entre os resultados e verificar se os erros decorrentes de arredondamentos sucessivos são significativos ou não.

Em sistemas de equações mal condicionados, temos uma característica peculiar na avaliação dos resíduos das equações, ou seja, nem sempre a solução mais próxima da exata gera os menores resíduos.

Por exemplo, suponha duas soluções aproximadas diferentes, S_1 e S_2 , para o sistema apresentado no **Exemplo 2.8**,

$$\begin{cases} x_1 + 3x_2 = 4 \\ x_1 + 3.00001x_2 = 4.00001 \end{cases}$$

Calculando os resíduos $\{R_1, R_2\}$ dessas duas equações, temos

$$\begin{cases} R_1 = |(x_1 + 3x_2) - 4| \\ R_2 = |(x_1 + 3.00001x_2) - 4.00001| \end{cases}$$

Para $S_1 = \{4, 0\} \Rightarrow$ Resíduos de $S_1 = \{0, 0.00001\}$ e

Para $S_2 = \{1.5, 1\} \Rightarrow$ Resíduos de $S_2 = \{0.5, 0.5\}$

Aparentemente, pela avaliação dos resíduos das equações, a solução mais próxima da exata é S_1 , pois gera menores resíduos; porém, sabendo que a solução exata é $S = \{1, 1\}$, percebemos que a solução S_2 está mais próxima da exata do que S_1 .

Concluimos que a avaliação dos resíduos das equações de um sistema não pode ser generalizada como critério para análise da proximidade de uma solução em relação ao seu valor exato.

Ainda sobre os métodos diretos, observamos que:

- a) Em sistemas de equações lineares que relacionam incógnitas com **grandezas de magnitudes diferentes**, como metros e milímetros, podem existir coeficientes também com magnitudes bastante diferentes, por isso também recomendamos o uso de processos de pivotação.

- b) Todos os **métodos diretos**, para resolução de sistemas de equações lineares, não apresentam **erros de truncamento**, pois são métodos teoricamente exatos. Esses têm apenas erros de arredondamento acumulados, que podemos avaliar comparando a solução aproximada obtida com outra solução com mais dígitos exatos (por exemplo, com o dobro de dígitos significativos, se disponível).

A seguir, vamos apresentar métodos alternativos para obter soluções aproximadas de sistemas de equações com grande número de incógnitas, mas controlando o erro de truncamento.

2.2 SOLUÇÃO DE SISTEMAS LINEARES POR MÉTODOS ITERATIVOS

Quando o sistema de equações lineares $A * X = B$ tiver algumas características especiais, como: a ordem n elevada (n é o número de equações); a matriz dos coeficientes A possuir grande quantidade de elementos nulos (matriz esparsa); e os coeficientes puderem ser gerados através de alguma lei de formação; em geral, será mais eficiente resolvê-lo através de um método iterativo (repetitivo), **sem operar com os coeficientes nulos**, desde que a convergência seja possível.

Se um algoritmo de um método direto, como o do método de Gauss, for aplicado a um sistema com matriz de coeficientes esparsa, muitas operações aritméticas desnecessárias serão empregadas, por exemplo, na manipulação de zeros com outros zeros, a menos que sejam evitadas pelo próprio algoritmo, como na seção 2.1.6.

Uma solução iterativa de $A * X = B$ consiste em tomar uma solução inicial $X^{(0)}$ (estimada de alguma forma) para a solução $S = \{x_1, x_2, x_3, \dots, x_n\}$ procurada e construir uma sequência $X^{(k)} = \{x_i^{(k)}\}_{k=0}^{\infty}$ de soluções **aproximadas** tal que, no limite:

$\lim_{k \rightarrow \infty} X^{(k)} = S$, se essa sequência for convergente.

Observe que as soluções de sistemas de equações lineares obtidas por **métodos iterativos** são soluções aproximadas a cada iteração k (contador de repetições), portanto têm **erros de truncamento**, além de erros de arredondamento; mas, nesse caso, os arredondamentos não são acumulados, uma vez que os coeficientes das equações não são alterados ao longo do processo.

Então, a questão fundamental é como gerar essa sequência convergente.

Em $A * X = B$, tomando a matriz A e particionando-a na forma $A = C + E$, com C não singular, temos:

$$(C + E)X = B \Rightarrow C * X = B - E * X \Rightarrow X = C^{-1} (B - E * X) \quad (7)$$

Então, aplicando uma solução inicial estimada $X^{(0)} = \{x_1^{(0)}, x_2^{(0)}, x_3^{(0)}, \dots, x_n^{(0)}\}$ no lado direito da eq. (7), temos:

$$X^{(1)} = C^{-1} (B - E * X^{(0)})$$

fazendo $X^{(0)} = X^{(1)}$ e reaplicando-o no lado direito da eq. (7), obtemos um $X^{(2)}$ e assim sucessivamente, resultando em uma sequência de vetores:

$$\begin{aligned} X^{(2)} &= C^{-1} (B - E * X^{(1)}) \\ X^{(3)} &= C^{-1} (B - E * X^{(2)}) \\ &\vdots \\ X^{(k+1)} &= C^{-1} (B - E * X^{(k)}) \end{aligned} \quad (8)$$

Se a sequência dada na eq. (8) for convergente, então existe:

$$\lim_{k \rightarrow \infty} X^{(k+1)} = \lim_{k \rightarrow \infty} X^{(k)} \quad (9)$$

Aplicando o limite a ambos os lados da eq. (8), temos:

$$\lim_{k \rightarrow \infty} X^{(k+1)} = C^{-1} (B - E * \lim_{k \rightarrow \infty} X^{(k)}) \quad (10)$$

Aplicando a eq. (9) na eq. (10), temos:

$$\lim_{k \rightarrow \infty} X^{(k+1)} = C^{-1} * B - C^{-1} * E * \lim_{k \rightarrow \infty} X^{(k+1)} \Rightarrow (1 + C^{-1} * E) * \lim_{k \rightarrow \infty} X^{(k+1)} = C^{-1} * B \quad (11)$$

Multiplicando a eq. (11) pela matriz C , temos:

$$(C + E) * \lim_{k \rightarrow \infty} X^{(k+1)} = B \quad (12)$$

Mas $(C + E) = A$, então:

$$A * \lim_{k \rightarrow \infty} X^{(k+1)} = B$$

Logo,

$$\lim_{k \rightarrow \infty} X^{(k+1)} = S \text{ é a solução de } A * X = B$$

O problema agora é que, para gerar a eq. (8), necessitamos da matriz inversa C^{-1} . Portanto, essa inversa, por questão de eficiência, tem que ser obtida facilmente.

A seguir, vamos abordar duas maneiras, não exclusivas, de particionar a matriz A de modo que a matriz C seja trivialmente invertida.

2.2.1 Método iterativo de Jacobi

No sistema $A * X = B$, tomamos a matriz A e a particionamos na forma $A = L + D + U$, em que:

- a) $D =$ diagonal principal de A ;
- b) $L =$ matriz triangular inferior estrita, obtida apenas dos elementos infradiagonais de A ; e
- c) $U =$ matriz triangular superior estrita, obtida apenas dos elementos supradiagonais de A .

Então, considerando que $C = D \Rightarrow E = L + U$ e aplicando-as na eq. (8), temos:

$$X^{(k+1)} = D^{-1}(B - (L + U) * X^{(k)}) \quad (13)$$

Obtemos trivialmente os elementos de D^{-1} tomando os recíprocos dos coeficientes da diagonal principal D de A . Assim, expressando a eq. (13) na forma desenvolvida para cada equação i , resulta que:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j < i}}^{i-1} a_{ij} x_j^{(k)} - \sum_{\substack{j=i+1 \\ j > i}}^n a_{ij} x_j^{(k)} \right) \quad (14)$$

Então, para resolver $A * X = B$ pelo método Jacobi, tomamos uma solução inicial $X^{(0)} = \{x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}\}$, isolamos diretamente a i -ésima incógnita x_i na i -ésima equação i , e aplicamos a eq. (14) somente com os coeficientes a_{ij} **não nulos** necessários (os coeficientes nulos não influenciam no resultado e não devem ser considerados nos cálculos para fins de otimização). Esses algoritmos são específicos para cada sistema, pois usam apenas os coeficientes não nulos, que devem ser identificados por algum mapeamento prévio.

Exemplo 2.9: resolva o [sistema](#)⁹, a seguir, pelo método de Jacobi:

$$\begin{cases} 3x_1 - x_2 - x_3 = 1 \\ x_1 + 3x_2 + x_3 = 5 \\ x_1 - x_2 + 2x_3 = 2 \end{cases}$$



Para esse sistema de pequeno porte, com coeficientes não nulos, seria recomendado usar um método direto, mas aplicaremos um método iterativo para permitir a reprodução manual do processo envolvido, com objetivos didáticos.

Solução:

Iniciamos montando as equações evolutivas para cada incógnita do sistema, isolando x_i da sua respectiva equação i . Na forma matricial, é equivalente a isolar as incógnitas x_i de cada diagonal principal:

$$\begin{cases} x_1^{(k+1)} = (1 + x_2^{(k)} + x_3^{(k)}) / 3 \\ x_2^{(k+1)} = (5 - x_1^{(k)} - x_3^{(k)}) / 3 \\ x_3^{(k+1)} = (2 - x_1^{(k)} + x_2^{(k)}) / 2 \end{cases}$$

Valor inicial: $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) = (0, 0, 0)$

Consideremos k como passo iterativo. Assim, usando a solução inicial em $k = 0$, podemos calcular a solução em $k + 1$ a seguir:

$$\begin{cases} x_1^{(1)} = (1 + 0 + 0) / 3 = 0.333 \\ x_2^{(1)} = (5 - 0 - 0) / 3 = 1.667 \\ x_3^{(1)} = (2 - 0 + 0) / 2 = 1.000 \end{cases}$$

A partir dessa primeira solução aproximada, atualizamos os valores iniciais $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) = (0.333, 1.667, 1.000)$ e calculamos uma segunda solução aproximada e assim sucessivamente, conforme a Tabela 2.3:

Tabela 2.3 – Valores da solução aproximada para cada iteração pelo método de Jacobi

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0	0	0
1	0.333	1.667	1
2	1.222	1.222	1.667
3	1.296	0.704	1
4	0.901	0.901	0.704
5	0.868	1.132	1
6	1.044	1.044	1.132

Fonte: Elaboração própria.

Note que a aproximação $(x_1^{(6)}, x_2^{(6)}, x_3^{(6)}) = (1.011, 1.044, 1.132)$, obtida em 6 iterações, está convergindo para uma solução S (nesse caso, $S_{\text{exato}} = \{1, 1, 1\}$).

Neste ponto, precisamos estabelecer um critério de parada que determine quão próxima da solução exata está a sequência convergente $x_i^{(k)}$, sem conhecermos a solução exata. Entre os critérios mais comuns, estão:

$$a) \text{ Max } \left\{ |x_i^{(k+1)} - x_i^{(k)}| \right\} \leq \varepsilon \Rightarrow i = 1, 2, 3, \dots, n$$

Corresponde à máxima diferença absoluta entre os valores da iteração nova e da iteração anterior, entre todas as incógnitas i .

$$b) \text{ Max } \left\{ \left| \frac{x_i^{(k+1)} - x_i^{(k)}}{x_i^{(k+1)}} \right| \right\} \leq \varepsilon \Rightarrow i = 1, 2, 3, \dots, n$$

Corresponde à máxima diferença relativa entre os valores da iteração nova e da iteração anterior, entre todas as incógnitas i .

$$c) \text{ Max } \left\{ |r_i^{(k+1)}| \right\} \leq \varepsilon \Rightarrow i = 1, 2, 3, \dots, n$$

Corresponde ao maior resíduo entre todas as equações i , $r_i^{(k+1)} = b_i - \sum_{j=1}^n a_{ij}x_j^{(k+1)}$, que já foi aplicado para aferição de soluções de métodos diretos, mas pode gerar valores inconsistentes para sistemas mal condicionados.

No caso do **Exemplo 2.9**, quando aplicamos o critério $\text{Max} \left\{ |x_i^{(k+1)} - x_i^{(k)}| \right\} \leq \varepsilon$, das diferenças de valores sucessivos, temos:

Tabela 2.4 – Valores da solução aproximada pelo método de Jacobi com critério de parada

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$ x_1^{(k+1)} - x_1^{(k)} $	$ x_2^{(k+1)} - x_2^{(k)} $	$ x_3^{(k+1)} - x_3^{(k)} $
0	0	0	0	-	-	-
1	0.333	1.667	1	0.333	1.667	1
2	1.222	1.222	1.667	0.889	0.555	0.667
3	1.296	0.704	1	0.074	0.518	0.667
4	0.901	0.901	0.704	0.395	0.197	0.296
5	0.868	1.132	1	0.033	0.197	0.296
6	1.044	1.044	1.132	0.176	0.088	0.132

Fonte: Elaboração própria.

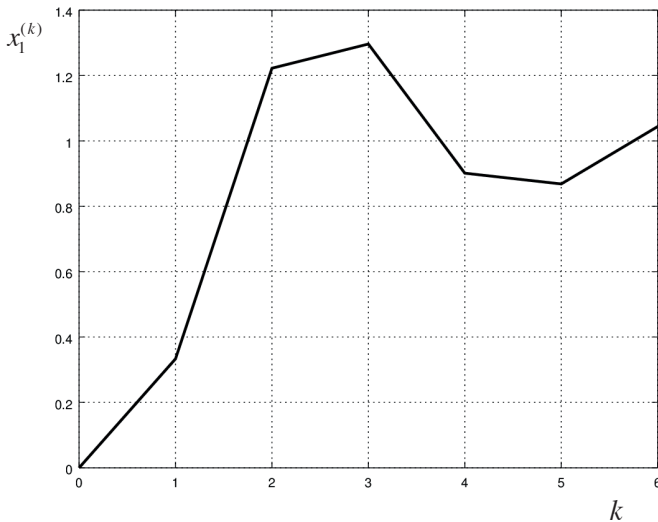
Portanto, no **Exemplo 2.9**, o critério alcançado depois de 6 iterações é:

$$\text{Max } |x_i^{(k+1)} - x_i^{(k)}| = 0.176$$

Note que, nesse exemplo, o processo iterativo é do tipo oscilatório, isto é, as incógnitas aumentam e diminuem ao longo das iterações, o que prejudica a convergência, tornando o processo mais lento.

Veja, no Gráfico 2.1, a evolução de $x_1^{(k)}$ com o nível iterativo k usando o método de Jacobi.

Gráfico 2.1 – Evolução de $x_1^{(k)}$ com o nível iterativo k aplicando o método de Jacobi



Fonte: Elaboração própria.

A seguir, vamos apresentar um método iterativo normalmente mais eficiente do que o método de Jacobi.

2.2.2 Método iterativo de Gauss-Seidel

No método iterativo de Gauss-Seidel, temos $A * X = B$ com a partição de A na forma $A = L + D + U$, como no método de Jacobi, porém tomando $C = D + L$, $E = U$ e aplicando na eq. (8):

$$\text{a) } X^{(k+1)} = (D + L)^{-1} (B - U * X^{(k)}) \quad (15)$$

Multiplicando a eq. (15) pela matriz $(D + L)$, temos:

$$(D + L)X^{(k+1)} = B - U * X^{(k)}$$

$$D * X^{(k+1)} = B - L * X^{(k+1)} - U * X^{(k)}$$

$$X^{(k+1)} = D^{-1} (B - L * X^{(k+1)} - U * X^{(k)}) \quad (16)$$

que, na forma desenvolvida, se torna:

$$x_i^{(k+1)} = (b_i - \sum_{\substack{j=1 \\ j < i}}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{\substack{j=i+1 \\ j > i}}^n a_{ij} x_j^{(k)}) / a_{ii} \Rightarrow i = 1, 2, \dots, n \quad (17)$$

Essa operacionalização é semelhante à do método de Jacobi, exceto que, na eq. (16), utilizamos sempre os valores de x_j **mais atualizados** possíveis no lado direito da equação iterativa, ou seja, já são usados os valores anteriormente calculados de $x_j^{(k+1)}$ das equações com $j < i$ dentro da própria iteração em andamento e, na eq. (14), utilizamos apenas os valores de $x_j^{(k)}$ da iteração anterior, $\forall j$.

Exemplo 2.10: resolva o sistema, a seguir, pelo método de Gauss-Seidel:

$$\begin{cases} 3x_1 - x_2 - x_3 = 1 \\ x_1 + 3x_2 + x_3 = 5 \\ x_1 - x_2 + 2x_3 = 2 \end{cases}$$

Solução:

Iniciamos montando as equações evolutivas para cada incógnita do sistema e isolando x_i da respectiva equação i , mas usando as incógnitas mais

atualizadas disponíveis no lado direito das equações, que correspondem àquelas multiplicadas pelos coeficientes de L da eq. (16):

$$\begin{cases} x_1^{(k+1)} = (1 + x_2^{(k)} + x_3^{(k)}) / 3 \\ x_2^{(k+1)} = (5 - x_1^{(k+1)} - x_3^{(k)}) / 3 \\ x_3^{(k+1)} = (2 - x_1^{(k+1)} + x_2^{(k+1)}) / 2 \end{cases}$$

Note que as equações evolutivas utilizam os valores disponíveis mais atualizados possíveis.

Valor inicial: $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) = (0, 0, 0)$

Assim, usando a solução inicial em $k = 0$, podemos calcular a solução em $k + 1$ a seguir:

$$\begin{cases} x_1^{(1)} = (1 + 0 + 0) / 3 = 0.333 \\ x_2^{(1)} = (5 - 0.333 - 0) / 3 = 1.555 \\ x_3^{(1)} = (2 - 0.333 + 1.555) / 2 = 1.611 \end{cases}$$

Observe que $x_1^{(1)} = 0.333$, calculado na primeira equação $i = 1$, já foi utilizado na equação de $x_2^{(1)}$ e assim por diante, conforme a Tabela 2.5.

Tabela 2.5 – Valores da solução aproximada pelo método de Gauss-Seidel

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$ x_1^{(k+1)} - x_1^{(k)} $	$ x_2^{(k+1)} - x_2^{(k)} $	$ x_3^{(k+1)} - x_3^{(k)} $
0	0	0	0	-	-	-
1	0.333	1.555	1.611	0.333	1.555	1.611
2	1.388	0.666	0.638	1.055	0.888	0.972
3	0.768	1.197	1.214	0.620	0.531	0.575
4	1.137	0.882	0.872	0.368	0.314	0.341
5	0.918	1.069	1.075	0.218	0.186	0.202
6	1.048	0.958	0.955	0.129	0.110	0.120

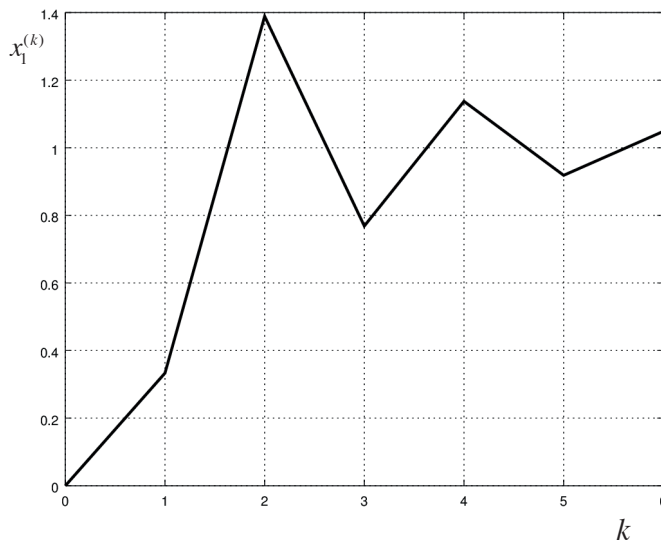
Fonte: Elaboração própria.

Temos a solução aproximada $(x_1^{(6)}, x_2^{(6)}, x_3^{(6)}) = (1.048, 0.958, 0.955)$ com critério de parada $Max |x_i^{(k+1)} - x_i^{(k)}| = 0.129$ atingido em 6 iterações.

Nesse exemplo, o processo iterativo correspondente à aplicação do método de Gauss-Seidel também é um processo oscilatório (podemos verificar que $x_i^{(k+1)} - x_i^{(k)}$ tem sinais alternados), mas agora temos um processo de convergência um pouco mais rápido porque no método de Gauss-Seidel são tomados os valores disponíveis mais atualizados.

Veja, no Gráfico 2.2, a evolução também oscilatória de $x_1^{(k)}$ (no eixo vertical) com o nível iterativo k (no eixo horizontal) usando o método de Gauss-Seidel.

Gráfico 2.2 – Evolução de $x_1^{(k)}$ com o nível iterativo k aplicando o método de Gauss-Seidel



Fonte: Elaboração própria.

A seguir, vamos apresentar condições suficientes para que um processo iterativo de soluções de sistemas lineares seja convergente.

2.2.3 Convergência dos métodos iterativos

Nas seções anteriores, desenvolvemos duas formas iterativas, equações (14) e (17), de construir uma sequência $\{X^{(k)}\}_{k=0}^{\infty}$ de possíveis aproximações para a solução de $A * X = B$. Agora, abordaremos quando se dá a convergência dessa sequência. Para tal, necessitamos de dois conceitos adicionais:

Definição 1: no sistema $A * X = B$, sejam $Z_i = \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|$ $i = 1, \dots, n$, teremos **diagonal dominante** se ocorrer:

a) $|a_{ii}| \geq Z_i$ para $i = 1, \dots, n$;

e

b) $|a_{ii}| > Z_i$, para pelo menos uma linha i de A .

Definição 2: se $|a_{ii}| > Z_i, \forall i = 1, \dots, n$, o sistema $A * X = B$ terá **diagonal estritamente dominante**.

2.2.3.1 Teorema de convergência – critério de Scarborough

Segundo o teorema de convergência de Scarborough: se o sistema $A * X = B$ tiver **diagonal dominante**, ou **diagonal estritamente dominante**, tanto a sequência construída pelo método de Jacobi quanto a de Gauss-Seidel convergirão para a solução S .

Veja a aplicação da Definição 1 aos Exemplos 2.9 e 2.10, conforme a análise a seguir:

a) Se $|a_{ii}| \geq Z_i$ para todas as linhas $i = 1, \dots, n$:

$$\begin{cases} 3x_1 - x_2 - x_3 = 1 & |3| \geq |-1| + |-1| \quad V \\ x_1 + 3x_2 + x_3 = 5 & |3| \geq |1| + |1| \quad V \\ x_1 - x_2 + 2x_3 = 2 & |2| \geq |1| + |-1| \quad V \end{cases}$$

temos uma condição verdadeira, pois o módulo de cada elemento da diagonal principal é maior ou igual à soma dos módulos dos demais elementos da respectiva linha.

b) Se $|a_{ii}| > Z_i$ para alguma linha $i = 1, \dots, n$:

$$\begin{cases} 3x_1 - x_2 - x_3 = 1 & |3| > |-1| + |-1| & V \\ x_1 + 3x_2 + x_3 = 5 & |3| > |1| + |1| & V \\ x_1 - x_2 + 2x_3 = 2 & |2| \geq |1| + |-1| & \end{cases}$$

A condição também é verdadeira para as duas primeiras equações.

Logo, esse sistema, resolvido nos **Exemplos 2.9** e **2.10**, satisfaz o critério de **Scarborough** da diagonal dominante e portanto tem convergência garantida.

Com relação à convergência dos métodos iterativos, algumas características importantes devem ser salientadas, entre as quais destacamos:

- A convergência para a solução $S = \lim_{k \rightarrow \infty} \{X^{(k)}\}$ não depende do valor inicial $X^{(0)}$. Portanto, a escolha do $X^{(0)}$ adequado afeta apenas a quantidade de iterações necessárias. Quanto mais próxima a solução estimada $X^{(0)}$ estiver da solução exata S , mais rápida será a convergência (se convergir).
- O **teorema de convergência** contém uma condição suficiente, porém não necessária. Portanto, se esse teorema for verdadeiro, a sequência S convergir; se não for, nada podemos afirmar.

Exemplo 2.11: verifique que, em $\begin{cases} x_1 + 2x_2 = 3 \\ x_1 - 3x_2 = -2 \end{cases}$, Gauss-Seidel fornece uma sequência convergente, mesmo não satisfazendo o teorema de convergência:

$$\begin{cases} x_1 = 3 - 2x_2 \\ x_2 = (2 + x_1) / 3 \end{cases}$$

Na Tabela 2.6, temos a evolução iterativa da solução aproximada por Gauss-Seidel:

Tabela 2.6 – Evolução iterativa da solução aproximada por Gauss-Seidel

k	x_1	x_2
0	0	0
1	3	0.666667
2	1.666667	1.666667
3	-0.333333	1.222222
4	0.555556	0.555556
5	1.888889	0.851852
6	1.296296	1.296296
20	0.982658	0.982658
30	1.002284	1.002284
40	0.999699	0.999699
50	1.000040	1.000040
60	0.999995	0.999995
70	1.000001	1.000001
76	1	1

Fonte: Elaboração própria.

- c) Um sistema que não tenha diagonal dominante pode, em alguns casos, ser transformado para ter diagonal dominante através de troca de linhas e/ou colunas (pivotamento parcial ou total).
- d) O **critério de convergência de Scarborough** enunciado anteriormente não é o único. Podemos verificar na literatura outros critérios de convergência.
- e) O teorema de convergência, baseado na dominância da diagonal principal da matriz de coeficientes do sistema, indica uma redução dos erros da solução aproximada em cada iteração. Por exemplo, no sistema dos **Exemplos 2.9 e 2.10**:

$$\begin{cases} 3x_1 - x_2 - x_3 = 1 \\ x_1 + 3x_2 + x_3 = 5 \\ x_1 - x_2 + 2x_3 = 2 \end{cases}$$

temos as seguintes equações iterativas:

$$\begin{cases} x_1^{(k+1)} = (1 + x_2^{(k)} + x_3^{(k)}) / 3 \\ x_2^{(k+1)} = (5 - x_1^{(k+1)} - x_3^{(k)}) / 3 \\ x_3^{(k+1)} = (2 - x_1^{(k+1)} + x_2^{(k+1)}) / 2 \end{cases}$$

Na equação de x_1 do sistema anterior, a solução inicial $X^{(0)}$ é nula, logo $X^{(0)}$ tem um erro unitário em todas as incógnitas (pois a solução exata é 1 para cada incógnita). Observe que a solução para x_1 na primeira iteração é 0.333, logo tem um erro menor do que 1.0, ou seja, erro de 0.667. Isso decorre da redução de erros promovida pela própria equação iterativa de x_1 , na qual os dois valores de x_2 e x_3 , cada um com seu erro interno 1.0, são multiplicados por coeficientes unitários e divididos pela diagonal principal 3.0, logo ocorre uma redução de erro aplicado na equação de x_1 , passando do erro inicial que era 1.0 para o novo erro $2/3$ em $x_1 = 0.333$. Assim, **quanto maior for a diagonal principal**, mais dominante esta será e **maior** também será a **redução de erro** da solução de uma iteração para outra.

- f) No sistema $\begin{cases} x_1 + 2x_2 = 3 \\ x_1 - 3x_2 = -2 \end{cases}$ do **Exemplo 2.11**, que não tem diagonal dominante e mesmo assim converge, as equações iterativas são:

$$\begin{cases} x_1 = 3 - 2x_2 \\ x_2 = (2 + x_1) / 3 \end{cases}$$

Observe que na equação de x_1 ocorre uma amplificação do erro inicial de x_2 , que é multiplicado por 2 e propagado para x_1 , mas na 2ª equação, de x_2 , ocorre uma redução do erro existente em x_1 , que é multiplicado por $1/3$ e depois atribuído a x_2 . Assim, ocorre uma redução global de erros promovida mais fortemente pela equação de x_2 . Note que a segunda equação tem diagonal bem dominante $|-3| > |1|$ e acaba compensando a não dominância da diagonal da primeira equação $|1| < |2|$.

A seguir, vamos apresentar um artifício matemático para tentar otimizar um processo iterativo.

2.2.4 Aplicação de coeficientes de relaxação

Nos casos de sistemas de equações lineares com diagonal principal pouco dominante, o processo iterativo poderá convergir oscilando, ou muito lentamente, por isso recomendamos usar fatores de sub ou sobre-relaxação, para tentar reduzir o número de iterações. Nos casos de sistemas que não tenham diagonal dominante e o Gauss-Seidel seja divergente, a sub-relaxação pode até o tornar convergente.

Por exemplo, para obter o valor mais atualizado de uma incógnita iterativa $x_i^{(k+1)}$ qualquer, podemos escrevê-la da seguinte forma:

$$x_i^{(k+1)} = x_i^{(k)} + \Delta x_i^{(k+1)}$$

em que $x_i^{(k)}$ é o valor antigo, do passo iterativo k ; e $\Delta x_i^{(k+1)}$ é o valor da atualização imposta ao valor antigo $x_i^{(k)}$ para atingir o valor novo $x_i^{(k+1)}$ do passo iterativo $k + 1$.

Quando o processo iterativo não converge bem, devemos tentar impor um fator artificial na atualização do valor novo, ou seja, não aplicar 100% da atualização calculada $\Delta x_i^{(k+1)}$ sobre o valor antigo $x_i^{(k)}$, por meio de um fator de relaxação λ , com $0 < \lambda < 2$, da seguinte forma:

$$x_i^{(k+1)} = x_i^{(k)} + \lambda * \Delta x_i^{(k+1)} \quad (18)$$

Quando o processo iterativo converge **oscilando**, ou até mesmo quando diverge, devemos tentar aplicar um fator de redução, desaceleração ou amortecimento na atualização do valor novo, ou seja, impomos um fator de **sub-relaxação** λ , com $0 < \lambda < 1$ na eq. (18).

Quando o processo iterativo converge **lentamente**, podemos tentar aplicar um fator de ampliação ou aceleração na atualização do valor novo, ou seja, impomos um fator de **sobre-relaxação** λ , com $1 < \lambda < 2$, também na eq. (18).

Observe que, se $\lambda = 1$, temos a equação evolutiva original $x_i^{(k+1)} = x_i^{(k)} + 1 * \Delta x_i^{(k+1)}$, com 100% da atualização calculada pelo método original. No caso do método de Gauss-Seidel, que fornece diretamente o valor novo $x_i^{(k+1)}$, podemos reescrever o valor do incremento $\Delta x_i^{(k+1)} = x_i^{(k+1)} - x_i^{(k)}$ dentro da equação anterior, gerando uma equação alternativa:

$$\begin{aligned}
 x_i^{(k+1)} &= x_i^{(k)} + \lambda (x_i^{(k+1)} - x_i^{(k)}) \\
 x_i^{(k+1)} &= (1 - \lambda) x_i^{(k)} + \lambda * x_i^{(k+1)}
 \end{aligned} \tag{19}$$

Por exemplo, a aplicação de relaxação para o método de Gauss-Seidel, dada pela eq. (17), gera a seguinte equação geral:

$$x_i^{(k+1)} = (1 - \lambda)x_i^{(k)} + \lambda \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j < i}}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{\substack{j=i+1 \\ j > i}}^n a_{ij}x_j^{(k)} \right) \Rightarrow i = 1, 2, \dots, n \tag{20}$$

Lembramos que a eq. (20) somente deve ser aplicada com os coeficientes a_{ij} não nulos, para evitar operações aritméticas desnecessárias.

Exemplo 2.12: resolva o sistema, a seguir, pelo método de Gauss-Seidel adotando uma sub-relaxação $\lambda = 0.5$ (lembrando que, no Exemplo 2.10, a solução sofreu oscilações ao longo das iterações, logo é indicada uma sub-relaxação).

$$\begin{cases}
 3x_1 - x_2 - x_3 = 1 \\
 x_1 + 3x_2 + x_3 = 5 \\
 x_1 - x_2 + 2x_3 = 2
 \end{cases}$$

Solução:

Montando as equações evolutivas para cada incógnita do sistema, conforme a eq. (20), temos:

$$\begin{cases}
 x_1^{(k+1)} = (1 - \lambda) x_1^{(k)} + \lambda (1 + x_2^{(k)} + x_3^{(k)}) / 3 \\
 x_2^{(k+1)} = (1 - \lambda) x_2^{(k)} + \lambda (5 - x_1^{(k+1)} - x_3^{(k)}) / 3 \\
 x_3^{(k+1)} = (1 - \lambda) x_3^{(k)} + \lambda (2 - x_1^{(k+1)} + x_2^{(k+1)}) / 3
 \end{cases}$$

Valor inicial: $(x_1^{(0)}, x_2^{(0)}, x_3^{(0)}) = (0, 0, 0)$ e $\lambda = 0.5$

Na Tabela 2.7, temos os valores de 6 iterações da solução aproximada com fator de sub-relaxação $\lambda = 0.5$.

Tabela 2.7 – Valores da solução aproximada pelo método de Gauss-Seidel com fator de sub-relaxação

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$ x_1^{(k+1)} - x_1^{(k)} $	$ x_2^{(k+1)} - x_2^{(k)} $	$ x_3^{(k+1)} - x_3^{(k)} $
0	0	0	0	-	-	-
1	0.166	0.805	0.659	0.166	0.805	0.659
2	0.494	1.043	0.967	0.327	0.238	0.307
3	0.748	1.069	1.063	0.254	0.025	0.096
4	0.896	1.041	1.067	0.147	0.027	0.004
5	0.966	1.014	1.046	0.069	0.026	0.021
6	0.993	1.000	1.024	0.026	0.014	0.021

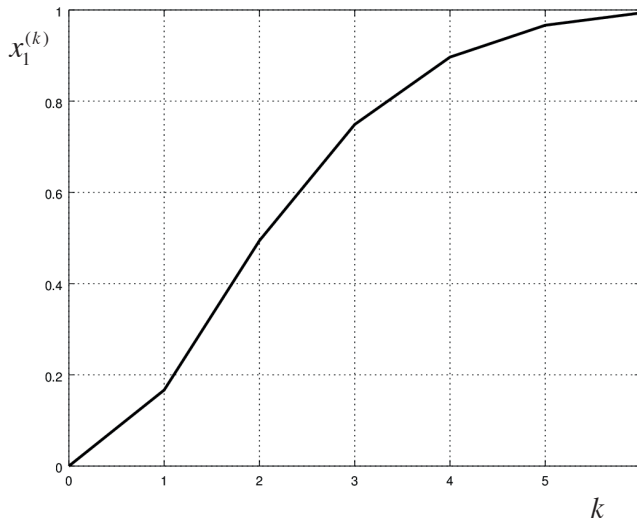
Fonte: Elaboração própria.

A solução aproximada $(x_1^{(6)}, x_2^{(6)}, x_3^{(6)}) = (0.993, 1.000, 1.024)$ foi atingida com critério de parada $\text{Max } |x_i^{(k+1)} - x_i^{(k)}| = 0.026$ em 6 iterações.

O processo iterativo com o fator de sub-relaxação ficou mais estável, gerando um processo de convergência monotônico (diferenças entre iterações sucessivas $x_i^{(k+1)} - x_i^{(k)}$ mantêm os mesmos sinais para cada passo iterativo), e conseqüentemente temos um processo iterativo mais rápido.

Note que, com as mesmas 6 iterações do método de Gauss-Seidel, atingimos um critério de parada máximo de 0.026 com sub-relaxação, contra 0.129 sem sub-relaxação. O Gráfico 2.3 demonstra a evolução de $x_1^{(k)}$ (no eixo vertical) e nível iterativo k (no eixo horizontal) usando o método de Gauss-Seidel e sub-relaxação $\lambda = 0.5$.

Gráfico 2.3 – Evolução de $x_1^{(k)}$ e nível iterativo k aplicando o método de Gauss-Seidel e sub-relaxação $\lambda = 0.5$



Fonte: Elaboração própria.

Recomendamos sempre aplicar previamente um teste com fatores de relaxação. Você pode experimentar pelo menos um fator de relaxação menor do que 1 (0.9) ou outro maior do que 1 (1.1) e avaliar os seus efeitos, se aumentam ou diminuem as iterações. Esses testes podem ser feitos com critérios de parada mais grosseiros, para serem mais rápidos. Também é importante buscar um fator de relaxação “ótimo”, que promova o menor número de iterações totais.

Ficou claro até este momento que as soluções obtidas para os sistemas de equações lineares são aproximadas e precisam ter um grau de confiabilidade definido. Então, quais são os erros associados a essa solução aproximada S ?

No **Exemplo 2.12**, vimos que os erros de arredondamento da solução S , obtida com apenas 4 dígitos totais, estavam presentes a partir do terceiro dígito fracionário: $X^{(6)} = \{0.993, 1.000, 1.024\}$. Lembremos que, nos métodos iterativos, o arredondamento não se propaga para dígitos mais significativos como nos métodos eliminativos, porque usa os coeficientes originais de cada equação. Então, se quisermos resultados com mais dígitos significativos, devemos usar precisão maior, como a variável *double* de 16 dígitos significativos, e teríamos resultados conforme a Tabela 2.8.

Tabela 2.8 – Valores da solução aproximada pelo método de Gauss-Seidel e com fator de sub-relaxação em precisão *double*

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$Max x_i^{(k+1)} - x_i^{(k)} $
0	0	0	0	-
1	0.166666666666667	0.805555555555555	0.659722222222222	0.805555555555555
2	0.494212962962963	1.043788580246910	0.967255015432099	0.327546296296296
3	0.748947080761317	1.069193940757890	1.063689222715190	0.254734117798354
4	0.896620734292838	1.041211977544270	1.067992422170450	0.147673653531521
5	0.966511100432207	1.014855401671690	1.046082286395100	0.069890366139369
6	0.993411831560569	1.000845347843240	1.024899522268220	0.026900731128362

Fonte: Elaboração própria.

Observe que as soluções de sistemas de equações lineares obtidas por métodos iterativos são soluções aproximadas a cada iteração, portanto têm **erros de truncamento**, além dos erros de arredondamento presentes no último dígito significativo, logo devemos ter uma forma de avaliar esses erros de truncamento associados a cada solução aproximada obtida.

Por exemplo, qual é o erro de truncamento máximo da solução obtida anteriormente em 6 iterações?

$$X^{(6)} = \{0.9934118315605686, 1.0008453478432351, 1.0248995222682158\}$$

Considerando que os erros de arredondamento foram minimizados ao máximo (pelo uso da variável *double*), podemos estimar uma solução S mais próxima da exata minimizando ao máximo os erros de truncamento decorrentes das aproximações, ou seja, podemos teoricamente executar infinitas aproximações, pois no limite de iterações:

$$\lim_{k \rightarrow \infty} X^{(k+1)} = S \text{ é a solução de } A * X = B$$

Ou seja, depois de infinitas iterações, vamos atingir a solução exata do sistema. Nesse exemplo, com poucas equações, podemos fazer iterações até que o critério de parada seja o mínimo possível para as variáveis *double* utilizadas, chegando aos resultados apresentados na Tabela 2.9.

Tabela 2.9 – Valores da solução aproximada pelo método de Gauss-Seidel com fator de sub-relaxação no limite da variável *double*

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$Max x_i^{(k+1)} - x_i^{(k)} $
0	0	0	0	-
1	0.1666666666666667	0.8055555555555555	0.6597222222222222	0.8055555555555555
2	0.494212962962963	1.043788580246910	0.967255015432099	0.327546296296296
3	0.748947080761317	1.069193940757890	1.063689222715190	0.254734117798354
4	0.896620734292838	1.041211977544270	1.067992422170450	0.147673653531521
5	0.966511100432207	1.014855401671690	1.046082286395100	0.069890366139369
6	0.993411831560569	1.000845347843240	1.024899522268220	0.026900731128362
12	0.999754544313626	0.999648218060907	0.999924062782415	4.66504270821e-04
55	1	1	1	0.

Fonte: Elaboração própria.

Note que, na iteração 55, o critério de parada $Max |x_i^{(k+1)} - x_i^{(k)}|$ atingiu valor nulo, portanto o algoritmo chegou ao limite da precisão digital disponível, de modo que a solução obtida não vai mais variar usando precisão *double*:

$$S = \{1, 1, 1\}$$

(nesse caso é uma solução exata mesmo, pois os seus valores são inteiros)

Agora, podemos calcular os erros de truncamento da solução $X^{(6)}$ obtida com 6 iterações em relação à solução exata S :

$$Erro X^{(6)} = |X^{(6)} - S| = \{6.588168e-03, 8.453478e-04, 2.489952e-02\}$$

$$Erro de truncamento máximo |X^{(6)} - S| = \{0.0248995222682158\}$$

O Erro de truncamento máximo 0.0248995222682158 é da mesma ordem de grandeza que o critério de parada baseado na máxima diferença entre duas iterações sucessivas $Max |x_i^{(k+1)} - x_i^{(k)}| = 0.0269007311283617$

e então esse critério de parada poderá ser usado como estimativa de erro de truncamento.

Alternativamente, podemos estimar uma **solução mais próxima da exata** através de soluções aproximadas pelo próprio método iterativo com algumas iterações a mais, por exemplo, usando o **dobro de iterações ou o dobro de precisão no critério de parada**. Assim, para calcular o erro de truncamento da solução aproximada com 6 iterações, $X^{(6)}$, podemos compará-la à solução aproximada mais precisa, ou com 12 iterações, ou com o critério de parada inicial 0.0269007311283617 ao quadrado, para ter o dobro de precisão (dobro de dígitos considerados exatos), que ambas estarão mais próximas da exata do que $X^{(6)}$:

$$X^{(6)} = \{0.9934118315605686, 1.0008453478432351, 1.0248995222682158\}$$

$$X^{(12)} = \{0.999754544313626, 0.999648218060907, 0.999924062782415\}$$

Veja que $X^{(12)}$ tem um erro menor do que $X^{(6)}$, comparado com S exato:

$$\text{Erro } X^{(12)} = |X^{(12)} - S| = \{2.454557e-04, 3.517819e-04, 7.593722e-05\}$$

(ou seja, tem 4 dígitos significativos exatos)

Assim, os erros de truncamento de $X^{(6)}$ estimados em relação a $X^{(12)}$ são:

$$\text{Erro } X^{(6)} = |X^{(6)} - X^{(12)}| = \{0.00634271, 0.00119713, 0.0249755\}$$

$$\text{Erro de truncamento máximo } |X^{(6)} - X^{(12)}| = \{0.02497545948580049\}$$

Então, o erro máximo de $X^{(m)}$ estimado, com m iterações, por comparação ao valor exato estimado $X^{(2m)}$, no caso, $|X^{(6)} - X^{(12)}| = 0.02497545948580049$, é da mesma ordem de grandeza do erro calculado por comparação com S exato, $|X^{(6)} - S| = 0.0248995222682158$.

Essas estimativas de erros de truncamento merecem todo cuidado, pois o critério das diferenças pode ter atingido valor suficientemente pequeno em alguns casos, entretanto a solução ainda pode estar longe do valor exato, como nos processos de convergência lenta.

Exemplo 2.13: calcule a solução do sistema, a seguir, com erro máximo de truncamento da ordem de 10^{-6} , com $n_1 = 3$ e $n_2 = 4$ e fator de relaxação otimizado para esse sistema:

$$\begin{cases} 2x_i - x_{i+1} = 0.1 & \longrightarrow & i = 1 \\ -x_{i-1} + 2x_i - x_{i+1} = 0.1 & \longrightarrow & i = 2, \dots, n_2 - 1 \\ -x_{i-2} - x_{i-1} + 2x_i = 0.2 & \longrightarrow & i = n_1, \dots, n_2 - 1 \\ -x_{i-1} + x_i = 0.3 & \longrightarrow & i = n_2 \end{cases}$$

Solução:

Observe que esse sistema não tem diagonal dominante (as diagonais de cada linha são iguais à soma dos módulos de seus coeficientes vizinhos, nenhuma é maior), portanto não tem convergência garantida, logo vamos implementar um algoritmo de Gauss-Seidel com uso de fatores de relaxação, partindo da solução inicial nula e fazendo iterações até que o critério máximo das diferenças entre iterações sucessivas seja $\text{Max} |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$. Vamos usar precisão *double*, para minimizar a influência de arredondamentos.

Obtemos as equações iterativas por isolamento das incógnitas x_i em cada equação i , ou seja, isolamos as incógnitas multiplicadas pelas diagonais principais (sistema na forma matricial):

$$\begin{cases} x_i = (0.1 + x_{i+1}) / 2 & \longrightarrow & i = 1 \\ x_i = (0.1 + x_{i-1} + x_{i+1}) / 2 & \longrightarrow & i = 2, \dots, n_2 - 1 \\ x_i = (0.2 + x_{i-2} + x_{i-1}) / 2 & \longrightarrow & i = n_1, \dots, n_2 - 1 \\ x_i = 0.3 + x_{i-1} & \longrightarrow & i = n_2 \end{cases}$$

Note que, no lado direito das equações iterativas, temos as mesmas incógnitas x calculadas no lado esquerdo das equações, portanto serão sempre as incógnitas mais atualizadas disponíveis.

Sem fator de relaxação, $\lambda = 1.0$, conforme a Tabela 2.10, temos:

Tabela 2.10 – Valores de solução aproximada pelo método de Gauss-Seidel com critério de parada $Max |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$x_4^{(k)}$	$Max x_i^{(k+1)} - x_i^{(k)} $
0	0	0	0	0	
1	0.050000	0.075000	0.162500	0.462500	0.462500
2	0.087500	0.175000	0.231250	0.531250	0.100000
3	0.137500	0.234375	0.285938	0.585938	0.059375
4	0.167187	0.276563	0.321875	0.621875	0.042188
5	0.188281	0.305078	0.346680	0.646680	0.028516
32	2.3333e-01	3.6666e-01	4.0000e-01	7.0000e-01	9.6743e-07

Fonte: Elaboração própria.

Com 32 iterações, atingimos $Max |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$ e temos:

$$X^{(32)} = \{0.233331807351778, 0.366664582137096, 0.399998194744437, 0.699998194744437\}$$

$$|x_i^{(k+1)} - x_i^{(k)}| = \{7.08210504629e - 07, \mathbf{9.67433540499e - 07}, 8.37822022592e - 07, 8.37822022647e - 07\}$$

Com fator de relaxação $\lambda = 1.1$, $Max |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$ em 27 iterações.

Com fator de relaxação $\lambda = 1.2$, $Max |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$ em 21 iterações.

Com fator de relaxação $\lambda = 1.3$, $Max |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$ em 24 iterações.

Então, vemos que o fator de relaxação ótimo está em torno de $\lambda = 1.2$, mas qual é o **erro máximo de truncamento** dessa solução obtida com $n = 21$ iterações e $Max |x_i^{(k+1)} - x_i^{(k)}| < 10^{-6}$?

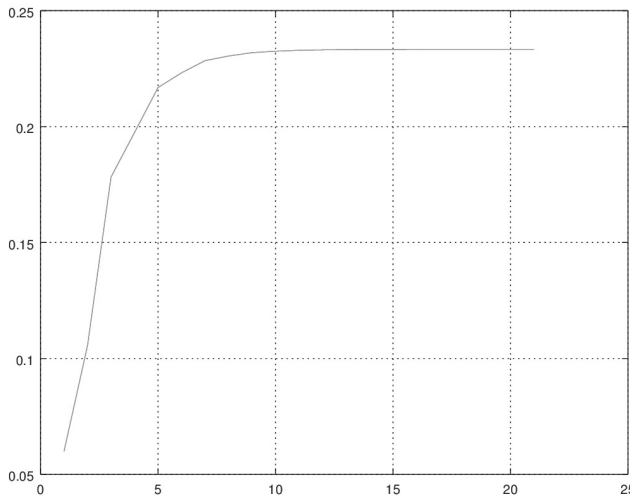
Para

$$X^{(21)} = \{0.233332454256926, 0.366665578771161, 0.399999140943902, 0.699999251437254\}$$

$$|x_i^{(k+1)} - x_i^{(k)}| = \{7.69513875675e-07, \mathbf{9.26761935082e-07}, \\ 7.46579149946e-07, 6.62960115027e-07\}$$

Observe que o processo de convergência do sistema do **Exemplo 2.13**, com fator de relaxação $\lambda = 1.2$, é monotônico, conforme o Gráfico 2.4, mostrando a evolução de x_1 em 21 iterações.

Gráfico 2.4 – Evolução de x_1 no eixo vertical e iterações k no eixo horizontal usando Gauss-Seidel com fator de relaxação $\lambda = 1.2$



Fonte: Elaboração própria.

Para calcular os erros estimados, precisaremos de uma solução mais próxima da exata, então podemos fazer iterações a mais (com o dobro de iterações, por exemplo) ou tentar atingir a solução exata S no limite da variável *double*. Nesse caso, com 61 iterações, o critério $|x_i^{(k+1)} - x_i^{(k)}| \forall i$ atinge valor zero numérico, a solução aproximada fica estacionária e é adotada como a solução exata estimada:

$$X^{(61)} = \{ 0.2333333333333333, 0.3666666666666667, 0.4000000000000000, \\ 0.7000000000000000 \}$$

$$|x_i^{(k+1)} - x_i^{(k)}| = \{0, 0, 0, 0\}$$

Então, o Erro máximo estimado de $X^{(21)}$ é

$$\text{Erro max } X^{(21)} = |X^{(21)} - X^{(61)}| = 1.08789550551e - 06$$

Enquanto o critério de parada é

$$\max |x_i^{(k+1)} - x_i^{(k)}| = 9.26761935082e - 07$$

Novamente verificamos que o critério de parada baseado nas diferenças das soluções entre iterações sucessivas tem a mesma ordem de grandeza dos erros estimados:

$$\max |x_i^{(k+1)} - x_i^{(k)}| = 0.926761935082e - 06$$

$$\text{Erro max } X^{(21)} = |X^{(21)} - X^{(61)}| = 1.08789550551380e - 06$$

Vamos implementar um algoritmo do método de Gauss-Seidel com fator de relaxação de modo que, no lado direito das equações iterativas, temos as mesmas incógnitas x calculadas no lado esquerdo das equações, sendo sempre as incógnitas mais atualizadas disponíveis, pois, à medida que os x são calculados, já são utilizados nas próximas equações. Confira o algoritmo **Cap2Gauss_Seidel.m** no **Caderno de Algoritmos** disponível para *download* no link: <<http://sergiopeters.prof.ufsc.br/algoritmos-livro/>>.

Quando um sistema esparso é gerado na forma de matriz completa, com todos os seus coeficientes nulos incluídos, é imprescindível primeiro gerar uma lista de coeficientes não nulos e aplicar os métodos iterativos operando apenas esses coeficientes não nulos.

Vimos até agora que o método de Gauss-Seidel é mais eficiente do que o de Jacobi, mas isso não é uma regra geral. O método de Jacobi se torna mais rápido no caso de processamento paralelo e/ou vetorial. No **Exemplo 2.14**, vamos apresentar um caso particular em que o método de Jacobi também é o mais rápido, mesmo em processamento sequencial normal.

Exemplo 2.14: determine a solução do sistema $\begin{cases} x_1 + 2x_2 - 2x_3 = 1 \\ x_1 + x_2 + x_3 = 1 \\ 2x_1 + 2x_2 + x_3 = 1 \end{cases}$ a partir

da solução inicial nula usando Jacobi, Gauss-Seidel e Gauss-Seidel com fator de sub-relaxação 0.3. Observe que esse sistema não tem convergência garantida, pois não tem diagonal dominante.

Solução:

a) Solução determinada pelo método de Jacobi:

Tabela 2.11 – Valores de solução aproximada pelo método de Jacobi

k	x_1	x_2	x_3
0	0	0	0
1	1	1	1
2	1	-1	-3
3	-3	3	1
4	-3	3	1

Fonte: Elaboração própria.

Vemos que o sistema converge com três iterações.

b) Solução determinada pelo método de Gauss-Seidel:

Tabela 2.12 – Valores da solução aproximada pelo método de Gauss-Seidel

k	x_1	x_2	x_3
0	0	0	0
1	1	0	-1
2	-1	3	-3
3	-11	15	-7
4	-43	51	-15
5	-131	147	-31

Fonte: Elaboração própria.

Vemos que o sistema diverge, sempre ampliando os valores.

- c) Solução determinada pelo método Gauss-Seidel com fator de sub-relaxação 0.3 (a sub-relaxação é indicada porque Gauss-Seidel produziu uma sequência divergente):

Tabela 2.13 – Valores da solução aproximada pelo método de Gauss-Seidel com fator de sub-relaxação

k	x_1	x_2	x_3
0	0	0	0
1	0.3	0.21	-0.006
2	0.3804	0.33468	-0.13325
3	0.285523	0.488593	-0.25774
4	0.052064	0.703719	-0.33389
5	-0.28612	0.978607	-0.34921
12	-3.24028	3.210688	0.633227
31	-2.72791	2.792591	0.947815
70	-2.99400	2.995765	0.993771
100	-2.99938	2.99956	0.999436
189	-3	2.999999	1
190	-3	3	1

Fonte: Elaboração própria.

Vemos que o sistema converge para o valor exato, mas com 190 iterações. Logo, nesse sistema sem convergência garantida, o método de Gauss-Seidel com fator de sub-relaxação 0.3 amortece as grandes variações da solução de uma iteração para outra, e também consegue obter uma sequência convergente, mas bem mais lenta do que a conseguida com o método de Jacobi (Tabela 2.11).

Considerações:

- a) Se o sistema satisfizer os critérios de convergência, mesmo que não tenha uma diagonal muito dominante, não é obrigatória a aplicação de fatores de sobre ou sub-relaxação, pois o processo iterativo já tem convergência garantida, mas é recomendável aplicá-lo para tentar acelerar o processo iterativo, conforme cada caso. Lembremos que,

- quanto maior for a diagonal principal, maior será a redução de erros de uma iteração para outra e mais rápida será a convergência.
- b) Se o sistema **não** tiver **convergência garantida**, recomendamos testar o efeito de fatores de sub-relaxação ou de sobre-relaxação, conforme o seu comportamento iterativo. O fator de relaxação pode transformar um processo iterativo divergente em convergente.
 - c) A utilização de fatores de **sub-relaxação** ($0 < \lambda < 1$) pode acelerar a convergência de processos **iterativos oscilatórios** (como no **Exemplo 2.10**, quando as incógnitas x_i crescem e diminuem com certa alternância).
 - d) A utilização de fatores de **sobre-relaxação** ($1 < \lambda < 2$) pode acelerar a convergência de processos **iterativos lentos** quando a atualização total $\Delta x_i^{(k+1)} = x_i^{(k+1)} - x_i^{(k)}$ é pequena a cada iteração. Essa variação do método de Gauss-Seidel é conhecida na literatura pertinente como Sucessive Over Relaxation (SOR).
 - e) As soluções aproximadas por métodos iterativos podem ser: **convergentes**, para um valor **estacionário** (quando termina o processo iterativo); **oscilatórias**; ou **divergentes**. Como vimos nos exemplos anteriores, os fatores de relaxação podem transformar um processo iterativo divergente em convergente, e um processo oscilatório em não oscilatório.
 - f) A escolha adequada do fator de relaxação λ ($0 < \lambda < 2$) nos permite conduzir o processo iterativo a uma *performance* “ótima”, atingindo a convergência com o menor número possível de iterações, mas, para descobrir esse fator “ótimo”, precisamos fazer testes sucessivos (que podem ser feitos com poucas iterações usando inicialmente critério de parada maior do que o desejado).
 - g) Para a solução aproximada $x_i^{(m)}$, obtida por métodos iterativos com m iterações, ou $x_i^{(\varepsilon)}$, obtida com o limite para o critério de parada ε , o erro máximo de truncamento é normalmente da mesma ordem do critério de parada baseado na máxima diferença entre as últimas iterações sucessivas, $\text{Max} |x_i^{(m)} - x_i^{(m-1)}|$. Podemos calcular o erro estimado da solução aproximada em relação a um valor de referência estimado:

- i) ou com o dobro de iterações $2m$: $x_i^{(2m)}$;
- ii) ou com limite para o critério de parada com o dobro de precisão (dobro de dígitos) ε^2 : $x_i^{(\varepsilon^2)}$.

Os valores exatos estimados nos itens (i) e (ii) têm praticamente o mesmo valor. Logo, os erros de truncamentos podem ser estimados por:

$$Erro(x_i^{(m)}) = |x_i^{(m)} - x_i^{(2m)}| \text{ ou}$$

$$Erro(x_i^{(\varepsilon)}) = |x_i^{(\varepsilon)} - x_i^{(\varepsilon^2)}|, \text{ para } i = 1, 2, \dots, n.$$

Esses cálculos de erros estimados precisam ser confirmados para cada tipo de sistema de equações.

2.3 CONCLUSÕES

Existem outros métodos de resolução de sistemas lineares, mas nos propomos a apresentar, neste capítulo, uma família de métodos que permitisse a solução de sistemas de equações lineares de médio porte com poucos coeficientes nulos, como a dos métodos eliminativos, embora estes tenham problemas de acúmulo de erros de arredondamento, especialmente os sistemas mal condicionados. Além disso, apresentamos uma família que permitisse a solução de sistemas de grande porte com muitos coeficientes nulos, como a dos métodos iterativos clássicos, além de apresentarmos algoritmos específicos para cada sistema, justamente para não operarmos esses coeficientes nulos desnecessariamente.

COMPLEMENTANDO...

Nesta seção, vamos discutir uma proposta de generalização do método de eliminação gaussiana otimizada para sistemas esparsos através de mapeamento duplo dos índices de coeficientes não nulos.

Sistema esperso genérico

Um algoritmo otimizado de Gauss operando sobre um sistema esperso genérico, organizado ou não em faixas e armazenado na forma de matriz expandida com os coeficientes nulos originais incluídos, pode ser implementado em duas etapas.

Primeira etapa

Na etapa de pré-processamento da matriz expandida representativa do sistema podem ser geradas e armazenadas duas listas duplas encadeadas, do tipo árvores binárias de busca (FEOFILOFF, 2008-2009): a **primeira lista** com o **mapeamento inicial** para os índices das linhas não nulas de cada coluna; e outra para os índices das colunas não nulas de cada linha, obtidas por varredura simples de cada elemento não nulo da matriz. Esses índices das duas listas são armazenados em árvores binárias de busca e não em vetores comuns para ter maior velocidade de acesso aos coeficientes. Essas árvores contêm o valor inteiro da posição de cada coeficiente, na linha ou na coluna, conforme o caso, e os endereços dos índices vizinhos à esquerda (anterior) e à direita (posterior). Na sequência, geramos uma **segunda lista** aplicando todos os passos k da eliminação gaussiana clássica, a cada linha i não nula abaixo de k , mas operando apenas as alterações nos índices iniciais dos coeficientes não nulos das linhas e das colunas já armazenados na primeira lista, sem ainda aplicar as respectivas operações em ponto flutuante aos coeficientes reais não nulos. Nesta fase, são geradas e armazenadas as alterações no mapeamento inicial dos índices dos coeficientes não nulos, de acordo com o processo clássico de escalonamento de Gauss, ou seja,

procedemos a execução de cada passo k do algoritmo gerando e armazenando uma segunda lista, bidimensional, para linhas e para colunas, que devem ser operadas em cada passo k , através de funções de inserção e remoção de índices encadeados na árvore de busca inicial (FEOFILOFF, 2008-2009), de acordo com os passos do algoritmo do método de eliminação gaussiana.

Depois dessa varredura nos passos do algoritmo, geramos e armazenamos um chamado **mapeamento instantâneo** de índices não nulos, para ser usado na aplicação de cada passo k do escalonamento da matriz de coeficientes, como se fosse uma foto dos índices dos coeficientes não nulos instantaneamente antes da aplicação do passo k do escalonamento. Essas listas bidimensionais instantâneas são armazenadas na forma de duas matrizes de índices inteiros geradas a partir das árvores binárias de busca. Em cada linha k , dessas duas matrizes, são armazenadas as respectivas linhas i e colunas j de coeficientes não nulos congeladas imediatamente antes da aplicação do passo k do método de eliminação gaussiana, de modo que contenham somente os índices das linhas e colunas que realmente precisam ser manipuladas, por conter coeficientes não nulos.

No **Caderno de Algoritmos**, confira a geração desses mapeamentos na função **fINDEX(A)** apresentada no algoritmo **GaussEsparsa.c**.

O processamento de operações de inserção e remoção de índices em árvores binárias de busca foi aplicado por ser mais rápido do que se essas operações fossem aplicadas diretamente com os vetores desses índices.

Segunda etapa

Nesta etapa, as eliminações são operadas em ponto flutuante somente com os coeficientes não nulos, usando os índices de linhas e colunas não nulas. Portanto, a segunda etapa é independente da primeira e podemos aplicá-la quantas vezes forem necessárias, conforme a função **fGaussEsparsa** do mesmo algoritmo, usando os índices do mapeamento instantâneo de coeficientes não nulos prefixados na primeira etapa.

Na Tabela 2.14, apresentamos testes de *performance* no tratamento de um sistema de $n = 2000$ equações, conforme o algoritmo **GaussEsparsa.c**,

com largura L de faixa dupla de dispersão dos coeficientes não nulos variável entre 10 e 1000 (limite), de acordo com a equação genérica a seguir:

$$\begin{cases} i = 1 & x_i + x_{i+1} = 150 \\ i = 2 \text{ até } (n/2) & x_{i-1} + 3x_i + x_{i+1} + x_{i+L} = 100 \\ i = (n/2 + 1) \text{ até } (n-1) & x_{i-L} + x_{i-1} + 3x_i + x_{i+1} = 200 \\ i = n & x_{i-1} + x_i = 300 \end{cases} \quad (21)$$

Em seguida, computamos o tempo para definição da indexação inicial e da indexação bidimensional instantânea de cada passo k imediatamente antes da eliminação gaussiana, o tempo para a resolução otimizada do sistema pré-mapeado e o tempo para a resolução pelo método clássico de Gauss-Seidel iterativo, otimizado por fator λ de relaxação (no caso $\lambda = 1.9$), com critério de parada $\max(|\Delta x_i|) < 10^{-4}$. Observe que esse exemplo de sistema não tem convergência garantida pelo critério de convergência de Scarborough.

Tabela 2.14 – Tempos de processamento para o mapeamento da matriz esparsa, para a solução com o método otimizado e para a solução com Gauss-Seidel

Largura L da faixa da dispersão	Indexação completa (ms)	Método otimizado, operando não nulos (ms)	Gauss-Seidel otimizado (ms)
$L = 0.5\%$ de n	58	1	020
$L = 5\%$ de n	388	12	156
$L = 10\%$ de n	3509	67	337
$L = 15\%$ de n	17155	213	714
$L = 20\%$ de n	50507	486	1224
$L = 30\%$ de n	253422	1573	2634
$L = 40\%$ de n	699212	3808	4532
$L = 50\%$ de n (max)	1310824	6911	7741

Nota: *ms = milissegundo = 10^{-3} segundos.

Fonte: Elaboração própria.

Por esses exemplos, verificamos que sistemas com coeficientes não nulos concentrados mais próximos da diagonal principal são resolvidos mais rapidamente tanto no pré-processamento quanto nas operações aritméticas de eliminação.

Nessa otimização, o mapeamento instantâneo gera valores de índices não nulos tanto mais espalhados pela matriz quanto maior for a largura inicial da faixa não nula. Sistemas com largura de dispersão L maiores geram

mais operações de inserção, ou seja, preenchem mais a matriz esparsa, conseqüentemente geram mais operações aritméticas. Por exemplo, sistemas com largura de faixa L a partir de 300 (15% n), para os dois lados da diagonal, exigem tempo de processamento (17155 *ms*) superior ao método de Gauss completo (14415 *ms*), mas na segunda fase de eliminação tornam-se muito mais rápidos (usam apenas 213 *ms* de processamento). Assim, o método apresentado se torna muito eficiente para aplicações em que a estrutura das matrizes de coeficientes é fixa, de modo que o mapeamento instantâneo pode ser gerado uma única vez e reaproveitado para operar diferentes valores de coeficientes armazenados na mesma estrutura. Verificamos também, nos exemplos testados, que o algoritmo otimizado proposto é sempre mais rápido do que métodos iterativos clássicos, cujas soluções são aproximadas com critério de parada relativamente grosseiro 10^{-4} , enquanto as soluções obtidas pelo algoritmo apresentado são exatas, exceto pelos arredondamentos acumulados. Por ser um método direto, ainda independe de critérios de convergência ou da classe de sistema envolvido.

Acesse o **Caderno de Exercícios e Respostas**, disponível no *link* <<http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>>, para realizar os exercícios propostos ao reforço do seu aprendizado.

SOLUÇÃO DE EQUAÇÕES NÃO LINEARES A UMA INCÓGNITA

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- determinar valores iniciais para raízes de uma equação não linear $f(x) = 0$;
- determinar raízes de uma equação $f(x) = 0$ por métodos de quebra e de linearização (iterativos);
- determinar todas as raízes, reais e/ou complexas, para equações polinomiais, considerando os efeitos da multiplicidade;
- avaliar a precisão do resultado obtido; e
- utilizar os algoritmos disponibilizados.

Em muitos modelos matemáticos de várias áreas, como engenharia e economia, ocorre a necessidade de determinar uma incógnita x que satisfaça uma equação $f(x) = 0$. Esse é um problema que ocupa os matemáticos há milênios. Apenas como exemplo emblemático, destacamos os papiros do escriba egípcio Ahmes (1680 a 1620 a.C.), nos quais foram encontradas aproximadamente 80 equações propostas, bem como a denominação dada por ele ao seu “estudo” sobre as equações: “Direção para saber todas as coisas obscuras”.

Para iniciar este capítulo, vamos apresentar três definições básicas importantes:

Definição 1: solução ou raiz da equação $f(x) = 0$ é todo $\alpha \in \mathbb{C}$ (Complexos), tal que $f(\alpha) = 0$.

Definição 2: zero da função $f(x)$ é todo $\alpha \in \mathbb{C}$ (Complexos), tal que $f(\alpha) = 0$.

Por exemplo:

$$a) \quad x^3 - 2x^2 + 2x = 0 \Rightarrow \alpha_1 = 0, \alpha_2 = 1 - i, \alpha_3 = 1 + i \quad (i = \sqrt{-1})$$

$$b) \quad e^{3x^3} - 2 = 0 \quad \Rightarrow \quad 3x^3 \ln(e) = \ln(2) \Rightarrow \alpha = \sqrt[3]{\ln(2)/3}$$

$$c) \quad \text{sen}(x) = 1 \quad \Rightarrow \quad \alpha_k = 2k\pi + \pi/2 \text{ com } k \in \mathbb{Z}$$

$$d) \quad 4 \cos(x) = e^x \quad \Rightarrow \quad \alpha_1 = -1.5158641228050098 \text{ e}$$

$$\alpha_2 = 0.9047882178730189$$

$$e) \quad e^{2x} = -3 \quad \Rightarrow \quad \alpha = (1/2) * i \left(2\pi * n + \pi - i * \ln(3) \right), n \in \mathbb{Z}$$

Note que nem sempre é possível resolver a equação de forma explícita, ou seja, nem sempre conseguimos isolar a incógnita em um dos lados da equação para obter explicitamente uma raiz α .

Pelas equações dadas anteriormente, concluímos que:

- a) Uma equação $f(x) = 0$ pode:
 - i) não ter solução Real;
 - ii) ter única solução;
 - iii) ter uma quantidade finita de soluções; e
 - iv) ter uma quantidade infinita de soluções.
- b) A solução de $f(x) = 0$, pela técnica do isolamento da incógnita, pode ser simples, difícil ou até impossível de ser obtida.
- c) Resolver equações exige conhecimento de outras *metodologias*⁹, além da forma de isolamento da incógnita.



Neste livro, usaremos metodologias iterativas para obter aproximações de soluções Reais e Complexas, com destaque para a metodologia proposta por Newton, conforme encontramos nas obras de Burden e Faires (2011) e de Cheney e Kincaid (2012).

Definição 3: um **método iterativo** obedece sempre a duas etapas em sua execução:

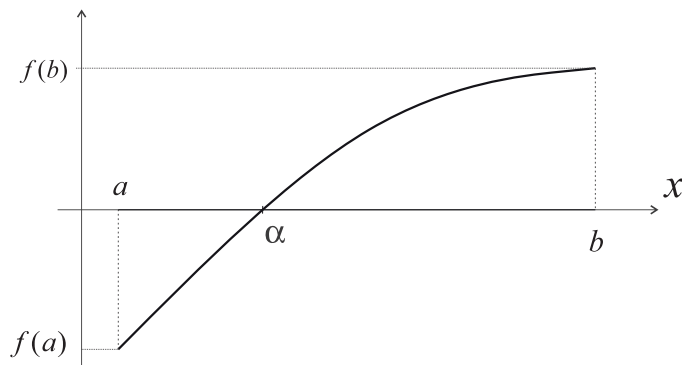
- a) isolamento, ou localização, de uma região do domínio da função geradora $y = f(x)$ que contenha cada solução desejada (aproximação inicial da raiz); e
- b) refinamento da solução isolada até a precisão requerida.

3.1 ISOLAMENTO OU LOCALIZAÇÃO DE SOLUÇÕES DE $f(x) = 0$

Podemos efetivar o isolamento, ou a localização, de uma região do domínio da função geradora que contenha a solução desejada da equação $f(x) = 0$ através de:

- a) Conhecimento prático sobre o problema modelado que resultou na equação, com a consequente estimativa da solução esperada.
- b) Traçado do esboço gráfico da função geradora $y = f(x)$, uma vez que toda raiz **Real** é uma interseção desse gráfico com o eixo das abscissas x , conforme o Gráfico 3.1.

Gráfico 3.1 – Intervalo que contém uma raiz Real α de $f(x) = 0$



Fonte: Elaboração própria.

- c) Teorema de Bolzano: numa $f(x) = 0$, se $y = f(x)$ for contínua em $[a, b]$ e $f(a) * f(b) \leq 0$ então $\exists \alpha \in [a, b] / f(\alpha) = 0$.
Note que esse teorema fornece uma condição suficiente, mas não necessária, para a existência de soluções reais, isto é, se satisfeita a condição, existirá solução, senão nada poderemos afirmar. Além disso, a solução em $[a, b]$ pode não ser única.
- d) Agrupamento das funções geradoras, $y = f(x)$, em classes com características especiais como as funções polinomiais, por exemplo, e utilização das suas propriedades algébricas, como veremos na seção 3.3.

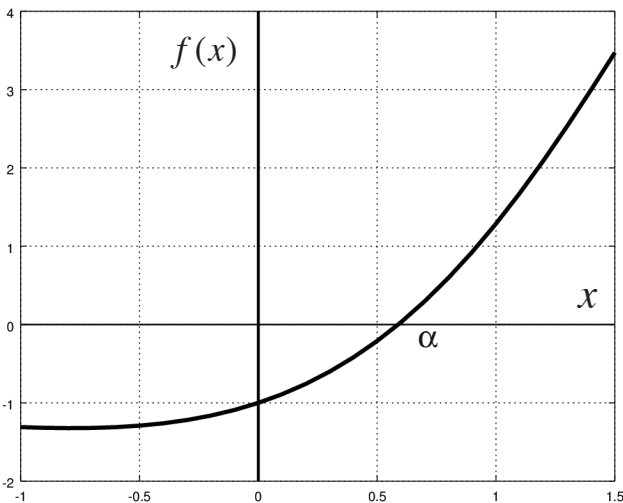
Vamos primeiramente localizar raízes de equações quaisquer, conforme o **Exemplo 3.1**.

Exemplo 3.1: isole, ou localize, um intervalo que contenha a primeira raiz positiva α de $e^x \operatorname{sen}(x) - 1 = 0$.

Solução:

Observe, no Gráfico 3.2, que a raiz está localizada no intervalo $[0, 1]$ ou $[0.5, 1]$, pois $f(x) = e^x \operatorname{sen}(x) - 1$ corta o eixo da abscissa x (com $y = 0$) nesses intervalos:

Gráfico 3.2 – Esboço da $f(x) = e^x \operatorname{sen}(x) - 1$



Fonte: Elaboração própria.

Também, como a $f(x) = e^x \operatorname{sen}(x) - 1$ é contínua em toda reta Real, e no intervalo $[0, 1]$ verificamos que $f(0) = -1$ e $f(1) = +1.287$, ou seja, $f(0) \cdot f(1) < 0$, pelo teorema de Bolzano, temos assegurado que existe raiz $\alpha \in [0, 1]$.

Tratada a questão da localização de uma solução de $f(x) = 0$, vamos partir para o processo de refinamento da solução isolada.

3.2 REFINAMENTO DA SOLUÇÃO ISOLADA

Depois de isolar uma raiz α em um intervalo $[a, b]$ do domínio de $f(x)$, devemos refiná-la por meio de técnicas específicas, para melhorar a

sua precisão. Essas técnicas podem ser agrupadas em três grandes famílias de métodos:

- a) **Métodos de quebra:** para os quais inicialmente temos de obter um intervalo $[a, b]$, tal que $\alpha \in [a, b]$, e o particionar em uma sequência de subintervalos menores $[a_k, b_k]$, mas que continuem contendo α , tal que $\lim_{k \rightarrow \infty} (|b_k - a_k|) = 0$. Esses métodos têm **convergência garantida**, mas normalmente são lentos.
- b) **Métodos de linearização:** para os quais obtemos uma aproximação inicial x_0 para a raiz α (x_0 valor estimado na etapa de isolamento) e posteriormente construímos uma sequência iterativa de valores:

$$\{x_k\}_{k=0}^{\infty} / \lim_{k \rightarrow \infty} (x_k) = \alpha, \text{ se a sequência for convergente.}$$

Para gerar essa sequência iterativa, linearizamos a equação não linear $f(x) = 0$, resultando em $x = g(x)$, conforme veremos na seção 3.2.2; obtemos o valor $x_1 = g(x_0)$; repetimos o processo com o $x_2 = g(x_1)$; e assim sucessivamente:

$$x_{k+1} = g(x_k)$$

Esses métodos **não têm convergência garantida**, mas podem ter convergência rápida, dependendo da forma iterativa estabelecida.

- c) **Métodos híbridos**⁹: consistem na mescla de métodos das duas metodologias citadas anteriormente, tentando associar a vantagem da convergência garantida na primeira com a alta velocidade de convergência da segunda.



Devido à sua maior complexidade, os métodos híbridos não fazem parte do escopo deste livro.

A seguir, vamos apresentar três tipos de métodos da primeira família.

3.2.1 Métodos de quebra

3.2.1.1 Método da biseção (ou bipartição) para raízes reais

Em $f(x) = 0$, se $f(x)$ for contínua em $[a, b]$ e ocorrer $f(a) * f(b) \leq 0$, então existirá uma raiz real α dentro desse intervalo, $\alpha \in [a, b]$. Logo, se $\alpha \neq a$ ou $\alpha \neq b$, uma aproximação para essa raiz pode ser tomada como o valor médio $\bar{x}_1 = \frac{a + b}{2}$.

Assim, o algoritmo do método da biseção é estabelecido com o objetivo de:

- a) reduzir o intervalo inicial $[a, b]$ a um subintervalo tão pequeno quanto necessário, porém contendo a solução α ; e
- b) para reduzir esse intervalo inicial, procedemos da seguinte forma:
 - i) obtemos o primeiro valor médio do intervalo:

$$\bar{x}_1 = \frac{a + b}{2}$$

então, temos três valores de x : a , b e \bar{x}_1 ;

- ii) redefinimos um novo intervalo $[a, b]$, com metade do comprimento inicial, considerando que:

$$\text{se } f(\bar{x}_1) = 0 \Rightarrow \alpha = \bar{x}_1$$

senão verificamos em qual subintervalo, $[a, \bar{x}_1]$ ou $[\bar{x}_1, b]$, α está:

$$\begin{cases} \text{se } f(a) * f(\bar{x}_1) < 0 \Rightarrow \alpha \in [a, \bar{x}_1], & b = \bar{x}_1 \text{ e } f(b) = f(\bar{x}_1) \\ \text{senão } f(\bar{x}_1) * f(b) < 0 \Rightarrow \alpha \in [\bar{x}_1, b], & a = \bar{x}_1 \text{ e } f(a) = f(\bar{x}_1) \end{cases}$$

Uma vez redefinido o novo intervalo, calculamos um segundo valor médio \bar{x}_2 :

$$\bar{x}_2 = \frac{a + b}{2}$$

então, temos três novos valores de x : a , b e \bar{x}_2 ;

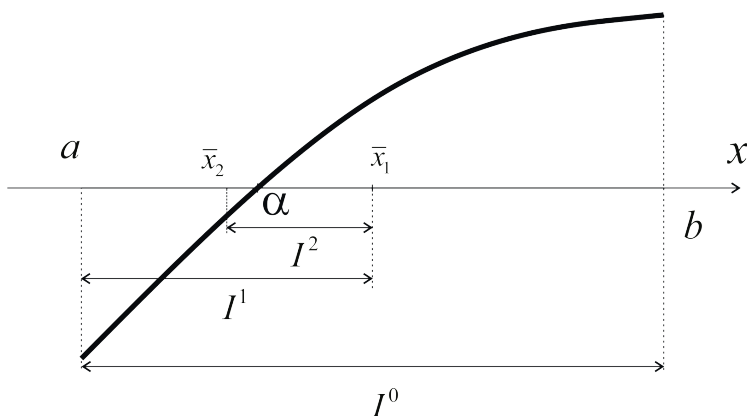
iii) com esses novos a , b e \bar{x}_2 , voltamos ao passo (ii), considerando $\bar{x}_1 = \bar{x}_2$, enquanto algum critério de parada não for satisfeito.

No final, o último x médio, calculado sobre o último subintervalo $[a, b]$, é a melhor aproximação para a raiz, $\alpha \cong (a + b) / 2$, sendo o comprimento desse subintervalo uma medida da precisão da raiz desejada.

A **bisseção** é um método usado frequentemente para obter a primeira aproximação da solução. Esse método também é chamado de **método da busca binária**, ou **método da dicotomia**.

No exemplo ilustrado no Gráfico 3.3, $\alpha \in [a, \bar{x}_1] \Rightarrow b = \bar{x}_1$, ou seja, o novo valor de b deve ser redefinido com o primeiro valor médio, \bar{x}_1 , para que α fique contido no primeiro subintervalo $I^1 = [a, \bar{x}_1]$, que é menor do que o intervalo inicial $I^0 = [a, b]$. Assim, repetimos o processo de partições, avaliando os valores extremos e médio de cada subintervalo I^k , com $k = 1, 2, 3, 4, \dots$, até que algum critério de parada seja satisfeito, por exemplo, $|b - a| < \varepsilon$, com os últimos a e b , de acordo com a precisão desejada.

Gráfico 3.3 – Método da bisseção, ou bipartição, do intervalo $[a, b]$



Fonte: Elaboração própria.

Neste ponto, podemos levantar uma questão fundamental: será que a sequência de valores médios \bar{x}_k converge para a raiz α , isto é, $\lim_{k \rightarrow \infty} (\bar{x}_k) = \alpha$?

Note que o comprimento de cada novo subintervalo é $|b_k - a_k| = \frac{|b - a|}{2^k}$, em que a e b são os valores extremos do intervalo inicial que contém a raiz α e k indica o número de partições.

Como $\alpha \in [a_k, b_k]$ e $\lim_{k \rightarrow \infty} |b_k - a_k| = \lim_{k \rightarrow \infty} \left(\frac{|b - a|}{2^k} \right) = 0 \Rightarrow$ o processo é convergente.

Então, o ponto médio do último subintervalo será uma aproximação da solução α com erro não superior ao comprimento desse subintervalo. Teoricamente, a raiz α será atingida quando o número k de partições tender ao infinito.

Exemplo 3.2: obtenha a primeira raiz positiva α de $e^x \operatorname{sen}(x) - 1 = 0$, situada em $[0, 1]$, por bisseção, com $n = 5$ partições do intervalo $[0, 1]$ (n denota o número total de partições).

Solução:

A partir do intervalo inicial $I^0 = [0, 1]$, $k = 0$, geramos o \bar{x}_k médio inicial e fazemos a seleção para definir o primeiro subintervalo, particionado, $I^1 = [0.5, 1]$ (em negrito), pois houve troca de sinais de $f(x) = e^x \operatorname{sen}(x) - 1$ entre \bar{x}_k e b . Em $k = 1$, partimos desse primeiro intervalo particionado $I^1 = [0.5, 1]$ e, a cada nova partição k , estabelecemos um novo subintervalo $[a, b]$ e uma nova aproximação \bar{x}_k para a raiz $\alpha \in [a, b]$.

Tabela 3.1 – Resultados do Exemplo 3.2

k	a	$\bar{x}_k = (a+b)/2$	b	$f(a)$	$f(\bar{x}_k)$	$f(b)$	$ b-a $	2^k
0	0	0.5	1	-1	-0.21	+1.287	1.0	1
1	0.5	0.75	1	-0.21	+0.443	+1.287	0.5	2
2	0.5	0.625	0.75	-0.21	+0.093	+0.443	0.25	4
3	0.5	0.5625	0.625	-0.21	-0.064	+0.093	0.125	8
4	0.5625	0.59375	0.625	-0.064	+0.013	+0.093	0.0625	16
5	0.5625	0.578125	0.59375	-0.064	-0.02584	+0.013	0.03125	32

Fonte: Elaboração própria.

Ao final de $n = 5$ partições, a raiz α está contida no subintervalo final $[a, b] = [0.5625, 0.59375]$, também grifado em negrito, de comprimento $(1 - 0) / 32$. Normalmente, a melhor raiz aproximada é a média entre a e b finais da última partição, neste caso seria $\bar{x}_5 = 0.578125$. No algoritmo apresentado, consideramos uma partição k completa quando os valores extremos do subintervalo final $[a, b]$ e o seu valor \bar{x}_k médio estão definidos.

O critério de parada baseado no comprimento do subintervalo final da partição $|b - a|$ é o mais utilizado nesse método e, neste caso, depois de 5 partições, temos $|b - a| = 0.03125$, ou seja, a raiz α é aproximada dentro de um subintervalo de comprimento máximo 0.03125, que é de *ordem* $O(10^{-2})$, logo \bar{x}_5 tem *dígitos convergidos* somente até o primeiro dígito fracionário, $\bar{x}_5 \cong \underline{0.578125}$ (sublinhados).

Ordem de grandeza dos números

Um número $x = m * 10^n$ em ponto flutuante é de ordem de grandeza:

- $O(10^n)$ se o multiplicador $m \in [10^0, 10^{1/2}]$ ($10^{1/2} = \sqrt{10} \cong 3.1622\dots$), e
- $O(10^{n+1})$ se $m \in [10^{1/2}, 10^1]$.

Por exemplo:

$x = 2.12345 * 10^{-6}$ é da ordem $O(10^{-6})$

$x = 5.12345 * 10^{-6}$ é considerado da ordem $O(10^{-5})$



Dígito convergido: é aquele que não mais sofrerá alteração durante as iterações.

E quando devemos parar o processo de partições sucessivas?

Podemos predefinir a quantidade n de bipartições por meio da previsão do comprimento do último subintervalo $|b^{(n)} - a^{(n)}|$ desejado, dado por:

$$|b^{(n)} - a^{(n)}| = \frac{|b-a|}{2^n} \cong \varepsilon \Rightarrow n = \ln\left(\frac{|b-a|}{\varepsilon}\right) / \ln(2) \quad (1)$$

Então, depois da n -ésima bipartição, o comprimento do subintervalo será da ordem de ε com a raiz α contida nesse último subintervalo $[b^{(n)}, a^{(n)}]$.

Exemplo 3.3: calcule o número n de bipartições necessárias para aproximar $\alpha \in [0, 1]$ com precisão $\varepsilon = 10^{-10}$, ou seja, com comprimento máximo do subintervalo final menor ou igual a 10^{-10} .

Solução:

Aplicando a eq. (1): $n = \ln\left(\frac{1-0}{10^{-10}}\right) / \ln(2) = 33.2 \Rightarrow n = 34$.

Ou seja, serão necessárias 34 partições para que o último subintervalo que contenha a raiz α seja menor de 10^{-10} e que a raiz α esteja exata até o seu décimo dígito significativo, depois do ponto (vírgula).

Se a quantidade n de bipartições não for previamente determinada, quando deveremos interromper, ou truncar, o processo de bipartições para assegurar certa precisão estabelecida? Ou seja, qual é realmente o erro de truncamento existente em uma solução \bar{x}_n aproximada em n partições?

A maioria das soluções tem erros de arredondamento e de truncamento. Como já vimos nos capítulos anteriores, o arredondamento pode ser minimizado se usarmos precisão maior, por isso vamos efetuar todos os cálculos com a variável *double*, padrão real IEEE de 64 bits.

No cálculo do **erro de truncamento**, precisamos de uma solução exata “estimada” x_e para comparar com a nossa solução aproximada \bar{x}_n , obtida depois de n partições, pois as soluções exatas não estão disponíveis. Então,

Erro truncamento estimado de $\bar{x}_n = |\bar{x}_n - x_e| \quad (2)$

Uma solução “exata” x_e estimada deve ter mais dígitos significativos exatos do que \bar{x}_n , e normalmente podemos obtê-la aplicando o *mesmo método* de aproximações numéricas, mas com mais repetições, então x_e pode ser:

- o **valor numericamente exato** de uma solução aproximada com até infinitas partições; ou
- o **valor aproximado** com um número mínimo de partições a mais do que o do valor aproximado, como o **dobro de partições**, por exemplo.



Esse mesmo método deve ter sido testado previamente com alguma solução exata conhecida.

No **Exemplo 3.3**, depois de 51 partições, atingimos critérios de parada $|b - a| = 0.0000000000000004$ e $|f(x)| \cong 0.000000000000000000$ com os seguintes resultados:

$$a = 0.5885327439818608$$

$$b = 0.5885327439818613$$

$$\bar{x}_{51} = 0.5885327439818611 \Rightarrow f(\bar{x}_{51}) = 0$$

A raiz \bar{x}_{51} encontrada atinge $f(\bar{x}_{51})$ numericamente nula, com 16 dígitos significativos exatos, ou seja, é uma raiz aproximada no limite da precisão da variável *double* adotada.

A raiz \bar{x}_{10} encontrada com o dobro de iterações é $\bar{x}_{10} = 0.58837890625$, obtida com critério de parada $|b - a| = 0.0009765625$.

O erro de truncamento da nossa solução aproximada $\bar{x}_5 = 0.578125$, obtida depois de $n = 5$ partições e com critério de parada $|b - a| = 0.03125$, é o seguinte:

$$\text{Erro de } \bar{x}_5 = |\bar{x}_5 - \bar{x}_{51}| = |0.578125 - 0.588532743981861| = 0.010407743981861 \text{ comparando com } \bar{x}_{51}.$$

$$\text{Erro de } \bar{x}_5 = |\bar{x}_5 - \bar{x}_{10}| = |0.578125 - 0.58837890625| = 0.01025390625 \text{ comparando com } \bar{x}_{10}.$$

O erro de truncamento mais exato possível de \bar{x}_5 é 0.010407743981861 , obtido por comparação com \bar{x}_{51} , e o erro estimado por comparação com \bar{x}_{10} é 0.01025390625 e ambos são da mesma ordem de grandeza $O(10^{-2})$.

Assim, a solução aproximada $\bar{x}_5 = 0.578125$ tem erro de truncamento ≈ 0.01 , enquanto o critério de parada é maior, mas da mesma ordem $O(10^{-2})$, $|b - a| = 0.03125$.

Portanto, é razoável estimar o valor “exato” de uma raiz com o mesmo método aproximador fazendo o **dobro** de partições ou com limite ao quadrado do critério de parada, para então calcular o erro de truncamento exato estimado:

Erro truncamento estimado de $\bar{x}_n = |\bar{x}_n - \bar{x}_{2n}|$ (3a)

$$\bar{x}^\varepsilon = |\bar{x}^\varepsilon - \bar{x}^{\varepsilon^2}| \quad (3b^\circ)$$



Vimos no Capítulo 2 que a estimativa de um valor “exato” de uma solução também pode ser obtida adotando critério de parada com o dobro de precisão (limite do critério de parada ε^2).

Observe que no **Exemplo 3.2** os critérios de parada alternativos $|f(\bar{x}_5)| = 0.0258393$ ou $|\bar{x}_5 - \bar{x}_4| < 0.015625$ também são de ordem de grandeza semelhante à ordem do critério de parada utilizado, $|b - a| = 0.03125$, e todos são superiores ao erro de truncamento real estimado.

Normalmente, podemos usar esses critérios de parada como limite superior do erro de truncamento, pois, uma vez que eles tenham atingido valores aceitáveis, os erros de truncamento devem ser ainda menores.

Como mencionamos no Capítulo 2, para cada método discutido ao longo desta obra, vamos apresentar todos os algoritmos compactados no **Caderno de Algoritmos** disponível para **download** no *link* <http://sergiopeters.prof.ufsc.br/algoritmos-livro/>, indicando o arquivo extensão **.m** do algoritmo correspondente ao método ou exemplo apresentado, como no arquivo **Cap3MetodosParticao.m**, que contém o método da bissecção e demais métodos de partição deste capítulo. Lembre-se sempre de testá-los no momento em que o indicamos para depois continuar a sua leitura.

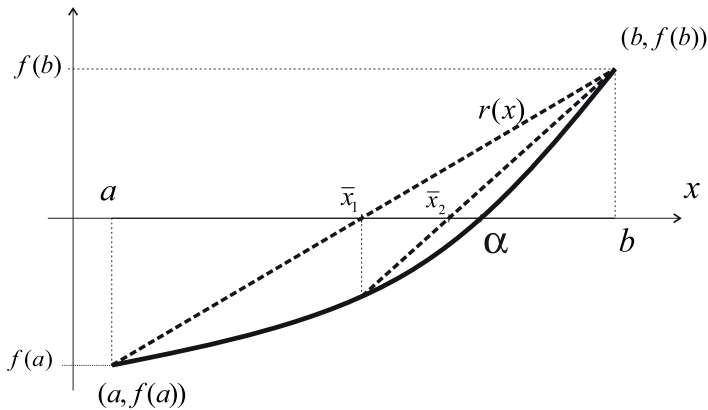
3.2.1.2 Método da falsa posição

Mesmo intuitivamente, percebemos que o método da bissecção é lento, pois reduz o intervalo de busca da raiz α em apenas 50% a cada partição.

Então, como particionar o intervalo $[a, b]$ para obter uma convergência mais rápida para a raiz α ?

No método da **falsa posição**, particionamos cada intervalo por meio de um novo valor de \bar{x}_k , que é a interseção entre uma reta $r(x)$ e o eixo das abscissas x , conforme o Gráfico 3.4.

Gráfico 3.4 – Método da falsa posição



Fonte: Elaboração própria.

O método da falsa posição consiste nas seguintes etapas:

- Tomar um intervalo inicial $[a, b]$, com $\alpha \in [a, b]$, na etapa de localização.
- Obter os pontos extremos do intervalo $(a, f(a))$ e $(b, f(b))$.
- Definir a reta $r(x)$ que passa por esses pontos extremos:

$$r(x) = f(a) + \left(\frac{f(b) - f(a)}{b - a} \right) (x - a) \quad (4)$$

- Calcular o valor do novo \bar{x}_k que particiona o intervalo, de modo que $r(\bar{x}_k) = 0$, ou seja, \bar{x}_k é a interseção entre a reta $r(x)$ e o eixo das abscissas x e não é uma média aritmética simples do intervalo, como no método da bisseção. Esse valor de \bar{x}_k é equivalente a uma média ponderada pelos valores das funções nas extremidades a e b :

$$r(\bar{x}_k) = 0 \Rightarrow \bar{x}_k = a - \frac{f(a) * (b - a)}{(f(b) - f(a))} \quad (5)$$

Alternativamente, também usamos a forma

$$\bar{x}_k = \frac{a * f(b) - b * f(a)}{f(b) - f(a)} \quad (\text{com uma multiplicação a mais}).$$

Então, a partir de a , b e \bar{x}_k , procedemos como no método da bisseção.

e) Se $f(\bar{x}_k) = 0 \Rightarrow \alpha = \bar{x}_k$.

Senão, pelo teorema de Bolzamo, decidiremos onde a raiz α ficou, se em $[a, \bar{x}_k]$ ou se em $[\bar{x}_k, b]$,

$$\begin{cases} \text{se } f(a) * f(\bar{x}_1) < 0 \Rightarrow \alpha \in [a, \bar{x}_1], & b = \bar{x}_1 \text{ e } f(b) = f(\bar{x}_1) \\ \text{senão } f(\bar{x}_1) * f(b) < 0 \Rightarrow \alpha \in [\bar{x}_1, b], & a = \bar{x}_1 \text{ e } f(a) = f(\bar{x}_1) \end{cases}$$

Redefinimos o novo subintervalo que contém a raiz $\alpha \in [a, b]$ e os valores das funções nesses pontos.

f) Com o novo subintervalo atualizado $[a, b]$, recalculamos o novo \bar{x}_k para a próxima partição:

$$\bar{x}_k = a - \frac{f(a) * (b - a)}{(f(b) - f(a))}$$

Com a, b e \bar{x}_k , retornamos ao item (e), enquanto algum critério de parada não for satisfeito, análogo ao método da bissecção.

g) No final, o valor do \bar{x}_n calculado sobre o último subintervalo, depois de n partições do intervalo inicial, é considerado a melhor aproximação para a raiz.

O critério de parada baseado no valor do subintervalo final $|b - a|$ não é indicado se uma das extremidades a ou b ficar fixa, como apresentamos no Gráfico 3.4, no qual o ponto $x = b$ inicial fica fixo ao longo das iterações.

O critério de parada do método da falsa posição pode ser baseado no valor das diferenças entre a raiz aproximada a cada partição \bar{x}_k e o seu valor anterior \bar{x}_{k-1} , $|\bar{x}_k - \bar{x}_{k-1}| < \varepsilon$, ou no valor da função em \bar{x}_k , $|f(\bar{x}_k)| < \varepsilon$, ou uma combinação de ambas.

Veremos outros critérios de parada na seção 3.2.2.

Exemplo 3.4: obtenha a primeira raiz positiva α de $e^x \operatorname{sen}(x) - 1 = 0$, situada em $[0, 1]$, pelo método da falsa posição, com 5 partições, e calcule o erro de truncamento atingido.

Solução:

Tabela 3.2 – Resultados do Exemplo 3.4

k	a	$\bar{x}_k = a - f(a)(b-a) / (f(b) - f(a))$	b	$f(a)$	$f(\bar{x}_k)$	$f(b)$	$ \bar{x}_k - \bar{x}_{k-1} $
0	0	0.43719	1.0	-1.0	-0.34444	+1.28736	-
1	0.43719	0.555986	1.0	-0.34444	-0.079729	+1.28736	0.118800
2	0.555986	0.581881	1.0	-0.079729	-0.016551	+1.28736	0.0258952
3	0.581881	0.5871886	1.0	-0.016551	-0.0033553	+1.28736	0.0053073
4	0.5871886	0.5882617	1.0	-0.0033553	-0.00067690	+1.28736	0.0010731
5	0.5882617	0.5884781340	1.0	-0.00067690	-0.00013643	+1.28736	0.00021638

Fonte: Elaboração própria.

Observe que aplicamos o teorema de Bolzano para redefinir cada novo subintervalo $[a, b]$, conforme a troca de sinais das funções em a , \bar{x}_k e b . Em $k = 5$ partições ponderadas do intervalo inicial, atingimos o subintervalo final $[0.5882617, 1.0]$, e a raiz aproximada dentro desse subintervalo deve ser $\bar{x}_5 \cong 0.5884781340$, com critério de parada $|\bar{x}_k - \bar{x}_{k-1}| < 0.0002163821$, ambos destacados em negrito. No algoritmo apresentado, também consideramos uma partição k completa quando os valores extremos do subintervalo particionado são redefinidos e o respectivo valor do \bar{x}_k é calculado. O último \bar{x}_k é considerado a raiz aproximada e o seu erro de truncamento pode ser estimado como fizemos no método da bisseção, comparando $\bar{x}_5 = 0.5884781340$ com $\bar{x}_{10} = 0.5885327258$:

$$\text{Erro de } \bar{x}_5 = |\bar{x}_5 - \bar{x}_{10}| = |0.5884781340 - 0.5885327258| = 0.0000545918$$

Para a validação desse erro, podemos atingir o valor aproximado mais exato possível, no limite do critério de parada para 16 dígitos significativos da variável *double*, em 23 partições, $\bar{x}_{23} = 0.5885327439818610$, enquanto no método da bisseção precisamos de 51 partições.

$$\text{Erro de } \bar{x}_5 = |\bar{x}_5 - \bar{x}_{23}| = |0.5884781340 - 0.5885327439818610|$$

$$\text{Erro de } \bar{x}_5 = |\bar{x}_5 - \bar{x}_{23}| = 0.000054609981861$$

Observe que o erro de truncamento estimado com o dobro de partições também gera uma ótima estimativa para o erro de truncamento.

No método da falsa posição, o valor do critério de parada adotado $|\bar{x}_k - \bar{x}_{k-1}| < 0.00021638$ ($O(10^{-4})$) também é superior ao erro de truncamento estimado, 0.000054609981861 ($O(10^{-4})$), logo o critério de parada pode ser adotado como limite superior do erro de truncamento. Observe, ainda, que o critério $|f(\bar{x}_k)| = 0.000136426$ também é da ordem de grandeza $O(10^{-4})$.

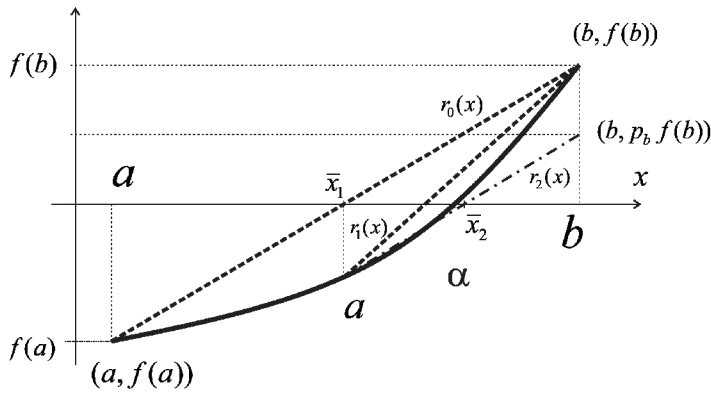
No **Caderno de Algoritmos**, apresentamos o algoritmo do método da falsa posição no arquivo **Cap3MetodosParticao.m**.

Na Tabela 3.2, o ponto $b = 1.0$ ficou fixo ao longo das partições, pois $\alpha \in [\bar{x}_k, b]$, conforme esboçado no Gráfico 3.4. Esse ponto que ficou fixo, sem atualização ao longo das partições, pode atrasar o processo de convergência. Para corrigir esse tipo de convergência unilateral, podemos destravar o ponto que ficou fixo usando o método da **falsa posição modificado**, por exemplo, ou adotar os métodos de linearização, que veremos na sequência.

3.2.1.3 Método da falsa posição modificado

Para destravar algum ponto que fique fixo, seja a ou seja b , na aplicação do método da falsa posição, podemos adotar o método da falsa posição modificado, que impõe uma **redução na magnitude da função** aplicada ao ponto que esteja fixo, em cada partição k , conforme detalhado no Gráfico 3.5.

Gráfico 3.5 – Método da falsa posição modificado



Fonte: Elaboração própria.

Assim, depois do cálculo inicial de $f(a)$, $f(b)$ e de $\bar{x}_1 = a - f(a)(b - a) / (f(b) - f(a))$, análogo ao método da falsa posição, precisamos:

- Identificar o ponto que fica fixo na partição, que é sempre aquele que continua sendo usado na redefinição do novo subintervalo; nesse caso, b fica fixo, pois $\alpha \in [\bar{x}_1, b]$, e o novo a é atualizado por \bar{x}_1 .
- Reduzir o valor original da função correspondente ao ponto fixo: nesse caso $f(b)$, ao qual podemos aplicar um fator p_b proporcional ao valor original da função no ponto não fixo, no caso a , dividido pela soma dos valores das funções neste ponto não fixo a e no ponto médio \bar{x}_1 :

$$p_b = f(a) / (f(a) + f(\bar{x}_1)) \quad (6)$$

Assim, *reduzimos de $f(b)$ para $f(b) = f(b) * p_b$* e usaremos esse novo valor no cálculo de \bar{x}_2 através da eq. (5).



Se a é o ponto fixo, aplicamos o fator de redução p_a sobre $f(a)$, em que $p_a = f(b) / (f(b) + f(\bar{x}_k))$.

Observe no Gráfico 3.5 que a reta $r_1(x)$ corresponde à aplicação da 2ª partição do método da falsa posição usando os valores originais das funções $f(a)$ e $f(b)$, enquanto com o método da falsa posição modificado usamos a função reduzida $f(b) = f(b) * p_b$, que corresponde à nova reta $r_2(x)$, menos inclinada do que no método da falsa posição com $r_1(x)$, o que gera \bar{x}_2 mais próximo da raiz α , nesse exemplo.

Quando reduzimos o valor da função sobre o ponto fixo, aumentamos a velocidade de convergência, pois, a partir da partição seguinte, o novo valor de b é destravado e assume o \bar{x}_2 , conforme exemplo do Gráfico 3.5. Paralelamente, devemos continuar atualizando o ponto não fixo, no outro extremo do intervalo. No caso do Gráfico 3.5, atualizamos $a = \bar{x}_1$ e $f(a) = f(\bar{x}_k)$, mantendo sempre o teorema de Bolzano válido, conforme segue:

a) Se $f(\bar{x}_k) = 0 \Rightarrow \alpha = \bar{x}_k$

Senão, pelo teorema de Bolzano, decidiremos onde a raiz α ficou, se em $[a, \bar{x}_k]$ ou em $[\bar{x}_k, b]$:

i) quando a é o ponto fixo ($f(a)$ é reduzida):

$$\rightarrow \text{se } f(a) * f(\bar{x}_k) < 0, \quad b = \bar{x}_k, \quad f(b) = f(\bar{x}_k) \text{ e } f(a) = f(a) * p_a$$

ii) quando b é o ponto fixo ($f(b)$ é reduzida):

$$\rightarrow \text{se } f(\bar{x}_k) * f(b) < 0, \quad a = \bar{x}_k, \quad f(a) = f(\bar{x}_k) \text{ e } f(b) = f(b) * p_b$$

b) Depois das atualizações do valor extremo não fixo, do valor da sua função e do valor da função reduzida sobre o ponto fixo, recalculamos o \bar{x}_k do intervalo como no método da falsa posição:

$$\bar{x}_k = a - \frac{f(a) * (b - a)}{f(b) - f(a)}$$

c) Com os novos $a, b, f(a), f(b)$ e \bar{x}_k , voltamos ao primeiro passo (a) até que algum critério de parada seja satisfeito.

Exemplo 3.5: obtenha a primeira raiz positiva α de $e^x \operatorname{sen}(x) - 1 = 0$, situada em $[0, 1]$, pelo método da falsa posição modificado, com 5 partições, e calcule o erro de truncamento atingido.

Solução:

Tabela 3.3 – Resultados do Exemplo 3.5

k	a	$\bar{x}_k = a - f(a)(b-a) / (f(b) - f(a))$	b	$f(a)$	$f(\bar{x}_k)$	$f(b)$	$ \bar{x}_k - \bar{x}_{k-1} $
0	0	0.4371861	1.0	-1.0000	-0.344443	+1.28736	-
$p_b = f(a) / (f(a) + f(\bar{x}_k)) = 0.7438021478$				-1.0000	-0.344443	+0.957537	
1	0.4371861	0.5860805	1.0	-0.34444	-0.006117	+0.957537	0.1488943
$p_b = f(a) / (f(a) + f(\bar{x}_k)) = 0.9825494019$				-0.34444	-0.006117	+0.940828	
2	0.5860805	0.5887545	1.0	-0.00617	+0.00055	+0.940828	0.0026740
$p_a = f(b) / (f(b) + f(\bar{x}_k)) = 0.9994114861$				-0.00614	+0.00055	+0.940828	
3	0.58860805	0.5885323	0.58875447	-0.00614	-1.12e-06	+5.54e-04	2.222e-04
$p_b = f(a) / (f(a) + f(\bar{x}_k)) = 0.9998176448$				-0.00614	-1.12e-06	+5.539e-4	
4	0.5885323	0.588532744	0.58875447	-1.12e-6	-1.12e-06	+5.539e-4	4.464e-07
$p_a = f(b) / (f(b) + f(\bar{x}_k)) = 0.9999999013$				-1.115e-6	-1.12e-06	+5.539e-4	
5	0.5885323	0.5885327439818612	0.588532744	-1.115e-6	+2.22e-16	+5.46e-11	2.189e-11

Fonte: Elaboração própria.

Observe que também aplicamos o teorema de Bolzano para redefinir cada novo subintervalo $[a, b]$, conforme a troca de sinais das funções em a e b . Em $n = 5$ partições do intervalo inicial, atingimos $\bar{x}_5 = 0.5885327439818612$ com a função $|f(\bar{x}_k)| = +2.22e - 16$ numericamente nula, embora o critério de parada adotado $|\bar{x}_k - \bar{x}_{k-1}| < 2.189e - 11$ ainda não seja nulo.

O erro de truncamento existente em \bar{x}_5 pode ser estimado como fizemos no método da bissecção, usando um valor de referência para comparação. Felizmente esse método é de convergência bem mais rápida do que os anteriores e atinge o limite nulo do critério de parada para 16 dígitos significativos com apenas 6 partições, $\bar{x}_6 = 0.5885327439818612$. Assim, o erro de truncamento é numericamente nulo, conforme segue:

Erro de truncamento estimado de $\bar{x}_5 = |\bar{x}_5 - \bar{x}_6| < O(10^{-16})$

Também podemos dizer que o erro de truncamento é menor do que o critério de parada atingido, $|\bar{x}_k - \bar{x}_{k-1}| < 2.189e - 11$, e observe que o valor da função já é numericamente nulo $|f(\bar{x}_5)| = 2.22044604925031e - 16$.

Agora, confira no **Caderno de Algoritmos** o algoritmo do método da falsa posição modificado no arquivo **Cap3MetodosParticao.m**.

Comparando os três métodos de partição apresentados, obtivemos a primeira raiz positiva de $e^x \operatorname{sen}(x) - 1 = 0$, situada em $[0, 1]$, no limite do critério de parada para a variável *double* de 16 dígitos significativos com:

- a) 51 partições no método da bissecção;
- b) 23 partições no método da falsa posição; e
- c) 6 partições no método da falsa posição modificado.

Os três métodos convergiram para o mesmo valor de raiz α , à exceção de algum arredondamento final:

- a) 0.5885327439818611(bissecção);
- b) 0.5885327439818610 (falsa posição);
- c) 0.5885327439818612 (falsa posição modificado).

Vamos abordar, a seguir, a família de métodos de linearização iterativa.

3.2.2 Métodos de linearização

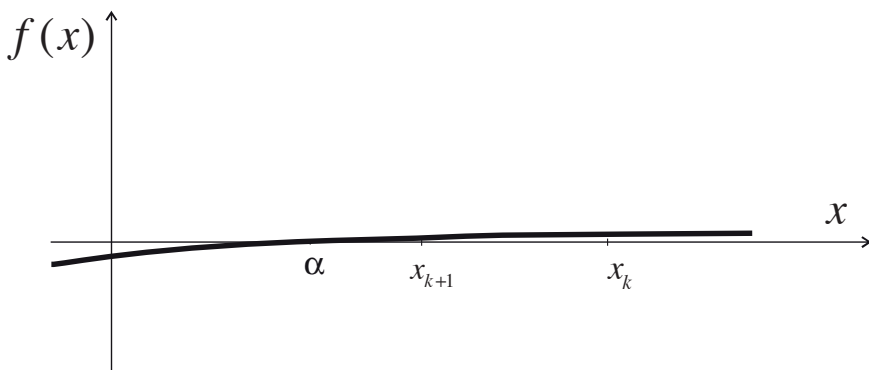
Nos métodos dessa família, buscamos linearizar a equação $f(x) = 0$, isolando um valor de x como **incógnita**, através de uma função do tipo $x = g(x)$, em que $g(x)$ é uma função de iteração avaliada em algum valor inicial de x .

Temos vários critérios de parada aplicáveis aos processos iterativos, alguns já vimos na seção anterior, como:

- a) $|x_{k+1} - x_k| < \varepsilon$ – esse critério falha quando x_k converge lentamente;
- b) $\left| \frac{x_{k+1} - x_k}{x_{k+1}} \right| < \varepsilon$ – o valor relativo é mais realístico, principalmente quando x_{k+1} é muito pequeno (mas deve ser diferente de zero) ou muito grande;
- c) $|f(x_{k+1})| < \varepsilon$ – esse critério falha quando $f(x)$ tem valores pequenos, conforme o Gráfico 3.6;
- d) $|f(x_{k+1})| + \left| \frac{x_{k+1} - x_k}{x_{k+1}} \right| < \varepsilon$ – é um critério composto, que abrange os dois anteriores.

No Gráfico 3.6, observe que, apesar de x_{k+1} ainda estar longe da raiz α , o critério de parada $|f(x_{k+1})| < \varepsilon$ já pode ter sido satisfeito (quando $f(x)$ tem valores pequenos). Por outro lado, o critério $|x_k - x_{k-1}| < \varepsilon$ também pode ser satisfeito rapidamente quando x_k converge lentamente. Portanto, algum critério de parada sempre precisa ser estabelecido, mas não deve ser usado isoladamente, sendo necessário fazer uma validação do resultado aproximado obtido com algum cálculo do erro de truncamento.

Gráfico 3.6 – Processo iterativo de convergência, quando $f(x)$ tem valores pequenos



Fonte: Elaboração própria.

3.2.2.1 Método da iteração linear

O método clássico de linearização é a **iteração linear**. Embora seja normalmente lento e instável, é apresentado aqui por questões didáticas. Nesse método, obtemos a função de iteração $g(x)$ simplesmente isolando uma das variáveis x da própria função geradora $f(x)$ e aplicando inicialmente o valor estimado x_0 , da etapa do isolamento da raiz. Assim,

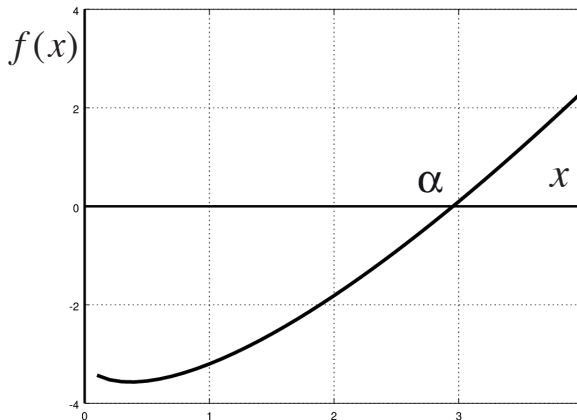
$$\begin{aligned} x_1 &= g(x_0) \\ x_2 &= g(x_1) \\ x_3 &= g(x_2) \\ &\vdots \\ x_{k+1} &= g(x_k) \end{aligned} \quad (7)$$

em que os subíndices $k + 1$ e k indicam o nível iterativo novo e anterior, respectivamente.

Exemplo 3.6: determine, por iteração linear, a raiz de $x * \ln(x) - 3.2 = 0$, $\alpha \in [2, 3]$, com o máximo de dígitos exatos disponível em sua representação digital.

Solução:

Gráfico 3.7 – Raiz de $f(x) = x * \ln(x) - 3.2 = 0$



Fonte: Elaboração própria.

O Gráfico 3.7 mostra que a raiz está entre $a = 2$ e $b = 3$, logo podemos adotar qualquer valor inicial x_0 desse intervalo.

A seguir, vamos definir a função de iteração $g(x)$ isolando o x de duas formas possíveis:

- a) $x_0 * \ln(x) - 3.2 = 0$: isolamos como incógnita o x que aparece em $\ln(x)$, logo:

$$\ln(x) = 3.2 / x_0 \Rightarrow x = e^{3.2/x_0}$$

Note que, nessa função iterativa, a solução x depende da estimativa da própria solução x_0 , portanto esperamos que esse cálculo melhore o valor de x em relação a x_0 . Assim, adotando o valor inicial médio em $[a, b]$, $x_0 = 2.5$, calculamos um novo $x = e^{3.2/2.5} = 3.59663972556928$.

Se agora adotarmos esse novo x como um novo valor inicial x_0 , então $x_0 = 3.59663972556928$, poderemos melhorar a raiz pela segunda vez, conforme a tabela iterativa a seguir:

Tabela 3.4 – Resultados do Exemplo 3.6

k	x_k	$x_{k+1} = e^{(3.2/x_k)}$	$ x_{k+1} - x_k $	$ f(x_{k+1}) $
0	2.5	3.59663972556928	1.09663972556928	1.40369884872868
1	3.59663972556928	2.43444635592001	1.16219336964927	1.03402594272603
2	2.43444635592001	3.72276714234931	1.28832078642930	1.69345547768941
3	3.72276714234931	2.36215810797272	1.36060903437659	1.16954640016981
4	2.36215810797272	3.87557252617237	1.51341441819964	2.05021252467948
⋮	⋮	⋮	⋮	⋮
999	1.34504630366580	10.79518554638936	9.45013924272355	22.48282865377773
1000	10.79518554638936	1.34504630366580	9.45013924272355	2.80129002385048
1001	1.34504630366580			

Fonte: Elaboração própria.

Na Tabela 3.4, os critérios de parada $|x_{k+1} - x_k|$ e $|f(x_{k+1})|$ não estão diminuindo, e o valor x_{k+1} da raiz aproximada está oscilando, aumentando e diminuindo a cada iteração sem ocorrer a convergência.

Observe que $g(x_k) = e^{3.2/x_k}$ é uma função iterativa do tipo exponencial que, neste exemplo, está amplificando o valor de x_k e conseqüentemente o seu erro. Então, precisamos reescrever a função iterativa $x = g(x)$.

- b) $x * \ln(x_0) - 3.2 = 0$: nessa tentativa, isolamos como incógnita o primeiro x da função, logo:

$$x = 3.2 / \ln x_0$$

Note que, nessa função iterativa, a solução x também depende da estimativa da solução x_0 , mas essa forma envolve uma função do tipo logaritmo que, neste exemplo, tende a reduzir o valor do x_0 e conseqüentemente o seu erro.

Adotando o mesmo $x_0 = 2.5$ como valor inicial, temos:

Tabela 3.5 – Resultados do Exemplo 3.6 com outra equação iterativa

k	x_k	$x_{k+1} = 3.2 / \ln(x_k)$	$ x_{k+1} - x_k $	$ f(x_{k+1}) $
0	2.5	3.492341337399333	0.992341337399333	1.167425623659166
1	3.492341337399333	2.558828299018562	0.933513038380771	0.795854258488439
2	2.558828299018562	3.405887761076910	0.847059462058348	0.973934620906707
3	3.405887761076910	2.611167118156381	0.794720642920529	0.693808868186438
4	2.611167118156381	3.334037325418966	0.722870207262585	0.814794334833144
5	3.334037325418966			
⋮	⋮	⋮	⋮	⋮
404	2.95416552327889	2.95416552327888	9.32587340685131e-15	8.88178419700125e-15
405	2.95416552327888	2.95416552327889	8.43769498715119e-15	8.88178419700125e-15
406	2.95416552327889			

Fonte: Elaboração própria.

A raiz obtida em $n = 406$ iterações é $x_{406} = 2.95416552327889$, destacada em negrito, com critério de parada:

$$|x_{k+1} - x_k| = 8.43769498715119e - 15$$

No final das iterações, as aproximações de x_k vão se alternando de uma iteração para outra e os critérios $|x_{k+1} - x_k|$, $|f(x_{k+1})|$ ficam estagnados.

Esse processo iterativo é altamente sensível à escolha da função de iteração $g(x)$, podendo gerar sequências convergentes, oscilatórias ou divergentes. Em alguns casos, podemos atenuar as oscilações usando fatores de amortecimento (*sub-relaxação*) no processo de atualização da incógnita x_{k+1} .



Apresentamos os fatores de sub-relaxação no Capítulo 2.

Para obter uma raiz com precisão máxima, é necessário que, a cada iteração, o valor calculado se aproxime da solução, ou seja, que o método convirja para o valor exato da raiz.

No caso dos métodos de bissecção, falsa posição e falsa posição modificado, não precisamos nos preocupar com a convergência, que está sempre garantida, uma vez que isolamos a raiz dentro de um dado intervalo e os métodos de quebra a mantêm dentro de um subintervalo desse intervalo inicial. Já nos métodos iterativos de linearização, a convergência não é garantida. A cada iteração, podemos nos aproximar ou nos afastar da solução. Portanto, antes de resolver um problema por meio desse método, recomendamos tentar verificar se haverá ou não a convergência.

A seguir, vamos apresentar o teorema de convergência, que coloca condições suficientes, porém não necessárias, para que o método de iteração linear seja convergente.

3.2.2.1.1 Teorema de convergência

Seja uma função $f(x)$ contínua em um intervalo $[a, b]$, α uma raiz contida em $[a, b]$ e $g(x)$ uma função de iteração obtida a partir de $f(x) = 0$:

- se $g(x)$ e $g'(x)$ forem contínuas em $[a, b]$; e
- se $|g'(x_k)| < 1 \forall x_k \in [a, b]$.

Então, $\lim_{k \rightarrow \infty} x_k = \alpha$, isto é, a sequência x_k converge para a solução α de $f(x) = 0$.

Porém, na maioria das vezes fica mais difícil obter e testar as $g'(x_k)$ do que aplicar diretamente o método e avaliar se os resultados estão convergindo ou não. No **Exemplo 3.6 (a)**, é fácil perceber que x_k aumenta e diminui alternadamente e, portanto, o método não converge. No **Exemplo 3.6 (b)**, também percebemos que x_k oscila, aumenta e diminui alternadamente, mas acaba convergindo. Para minimizar essas oscilações, podemos aplicar o fator de sub-relaxação, conforme o **Exemplo 3.7**.

Exemplo 3.7: determine a raiz de $x * \ln(x) - 3.2 = 0$, $\alpha \in [2, 3]$ com o máximo de dígitos significativos disponível em sua representação digital, usando fator de sub-relaxação para atenuar as oscilações nas iterações.

Solução:

A forma iterativa que adotamos no **Exemplo 3.6 (b)**, $x = 3.2 / \ln(x_0)$, foi oscilatória mas convergente. Então, recomendamos usar um fator de amortecimento ou sub-relaxação $0 < \lambda < 1$, como adotado no método de Gauss-Seidel aplicado no Capítulo 2 para determinação iterativa de soluções x oscilatórias de sistemas de equações lineares:

$$x = x_0 + \lambda * \Delta x \quad (8a)$$

Em que $\Delta x = x - x_0$ é o incremento iterativo original do método usado, pois esse x é o valor novo dado por $x = 3.2 / \ln(x_0)$.

Reescrevendo a eq. (8a), e teremos:

$$x = x_0 + \lambda * \Delta x = x_0 + \lambda (x - x_0) = (1 - \lambda)x_0 + \lambda * x \quad (8b)$$

Note que, se $\lambda = 1$, teremos o cálculo original.

Adotando $x_0 = 2.5$ e $\lambda = 0.5$, e teremos a função iterativa:

$$x = (1 - \lambda)x_0 + \lambda (3.2 / \ln(x_0))$$

Tabela 3.6 – Resultados do Exemplo 3.7

k	x_k	$x_{k+1} = (1 - \lambda)x_k + \lambda(3.2 / x_k)$	$ x_{k+1} - x_k $	$ f(x_{k+1}) $
0	2.5	2.996170668699666	0.496170668699666	0.0878030292836272
1	2.9961706686996665	2.956163274914294	0.0400073937853724	0.0041624239393347
2	2.9561632749142941	2.95424314678781	1.920128126488e-03	1.617075723499e-04
3	2.95424314678781	2.95416850626476	7.464052305028e-05	6.214206048849e-06
4	2.95416850626476	2.95416563786223	2.868402523503e-06	2.387018933625e-07
⋮	⋮	⋮	⋮	⋮
10	2.95416552327889	2.954165523278883	9.325873406851e-15	4.440892098501e-16
11	2.954165523278883	2.954165523278883	0.000000000000000	0.000000000000000
12	2.954165523278883			

Fonte: Elaboração própria.

Assim, obtivemos a raiz em $n = 12$ iterações, $x_{12} = 2.95165523278883$, em negrito, com critérios de parada numericamente nulos $|x_{k+1} - x_k| = 0.0$ e $|f(x_{k+1})| = 0$, portanto erro de truncamento no limite da precisão da variável *double*.

Podemos testar outros valores de λ e verificar que $\lambda = 0.5$ é mesmo o valor ótimo, permitindo a convergência no menor número de iterações. Lembre-se de que, sem o fator de amortecimento $\lambda = 0.5$, conforme o **Exemplo 3.6 (b)**, foram necessárias 406 iterações para atingir a mesma solução com 16 dígitos significativos.

Acesse o *link* <<http://sergiopeters.prof.ufsc.br/algoritmos-livro/>> para baixar o **Caderno de Algoritmos** e conferir o algoritmo do método da iteração linear no arquivo **Cap3MetodosIterativos.m**.

Também podemos implementar algoritmos puramente iterativos diretamente em **calculadoras científicas**, desde que tenham o recurso de armazenamento *Ans* (*Answer*) para armazenar o último valor digitado, desta forma:

- digitamos o valor inicial x_0 , por exemplo, 2.5, e pressionamos a tecla “=” para armazená-lo em *Ans*;
- digitamos a equação iterativa do novo x , com x_0 representado pelo *Ans* (valor da raiz anterior):

$3.2 / \ln(Ans)$, no caso do **Exemplo 3.6** (b), ou
 $(1 - 0.5) * Ans + 0.5 * 3.2 / \ln(Ans)$, no caso do **Exemplo 3.7**; e

- c) pressionando a tecla “=” será gerado um novo x , que fica automaticamente armazenado em Ans e atualiza x_0 . Desse modo, a cada “=” pressionado, teremos uma nova iteração gerada no *display* da calculadora.

Vamos apresentar agora o método de Newton, que é mais rápido e estável do que o método da iteração linear.

3.2.2.2 Método de Newton

O método de linearização de Newton é o mais utilizado na solução de equações. Para se obter a sua função de linearização, a $f(x)$ pode ser aproximada de duas formas.

Para $f(x) = 0$ com as funções $f(x)$ e $f'(x)$ contínuas em um intervalo suficientemente próximo da solução desejada, podemos gerar uma sequência de aproximações dessa solução da seguinte forma:

- a) Tomamos uma solução inicial x_0 .
- b) Calculamos o ponto $(x_0, f(x_0))$ e o coeficiente angular $f'(x_0)$ da reta $r(x)$ tangente à $f(x)$ nesse ponto x_0 , conforme o Gráfico 3.8. Então, essa reta será definida por:

$$r(x) = f(x_0) + f'(x_0)(x - x_0) \quad (9)$$

- c) Calculamos a interseção de $r(x)$ com o eixo das abscissas x :

→ $r(x) = 0$ na eq. (9), obtemos a função de iteração $x = g(x)$:

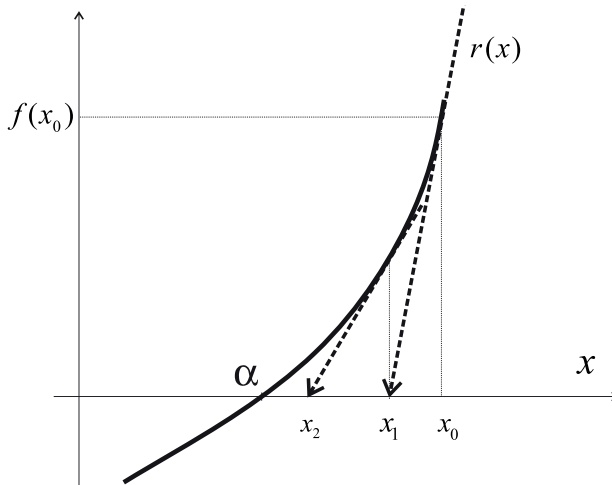
$$x - x_0 = -f(x_0) / f'(x_0) \quad \text{para o incremento de } x \text{ e}$$

$$x = x_0 - f(x_0) / f'(x_0) \quad \text{para } x \text{ (} g(x) = x - f(x) / f'(x) \text{)}.$$

Repetindo sucessivamente o passo (c), sempre usando o último valor calculado como solução anterior x_k , obtemos a expressão de recorrência:

$$x_{k+1} = x_k - f(x_k)/f'(x_k) \quad (10)$$

Gráfico 3.8 – Interpretação geométrica do método de Newton e sua reta tangente $r(x)$



Fonte: Elaboração própria.

Note que a eq. (10) gera uma sequência de valores x_k , $k = 0, 1, 2, \dots$, conforme o Gráfico 3.8, que, se for convergente, existirá um valor final α para $\lim_{k \rightarrow \infty} x_{k+1} = \lim_{k \rightarrow \infty} x_k = \alpha$. Por consequência, aplicando esses limites na eq. (10), temos:

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

Devido à continuidade de $f(x)$ e $f'(x)$, temos:

$$\lim_{k \rightarrow \infty} f(x_k) = f(\lim_{k \rightarrow \infty} x_k) \text{ e } \lim_{k \rightarrow \infty} f'(x_k) = f'(\lim_{k \rightarrow \infty} x_k),$$

então

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

Como $\lim_{k \rightarrow \infty} x_{k+1} = \lim_{k \rightarrow \infty} x_k = \alpha$, temos que:

$$\alpha = \alpha - \frac{f(\alpha)}{f'(\alpha)} \rightarrow \frac{f(\alpha)}{f'(\alpha)} = 0 \text{ e } f(\alpha) = 0, \text{ desde que } f'(\alpha) \neq 0.$$

Logo, α é uma solução pois $f(x = \alpha) = 0$.

Portanto, se a sequência gerada pela eq. (10) for convergente, o seu limite será sempre uma solução. Mais ainda, normalmente, a velocidade de convergência ou divergência desse método é do tipo quadrática, isto é, o incremento $\Delta x = x_{k+1} - x_k$ da aproximação de x_{k+1} é aproximadamente o quadrado do incremento da aproximação anterior, fato que não ocorre nos métodos de quebra.

Alternativamente, podemos obter o método de Newton como uma linearização da função $f(x)$ através de uma série de Taylor truncada:

- seja $f(x)$ uma função contínua no intervalo $[a, b]$;
- seja α um zero contido nesse intervalo, com função $f(x)$ e derivada $f'(x)$ contínuas em $[a, b]$, mas com $f'(x) \neq 0$.

Desse modo, podemos encontrar uma aproximação x_{k+1} para a raiz de $f(x) = 0$ no intervalo $[a, b]$ utilizando a expansão de $f(x_{k+1})$ em série de Taylor em torno de um valor inicial estimado x_k , em que $\Delta x = x_{k+1} - x_k$.

Logo, $x_{k+1} = x_k + \Delta x$ e

$$f(x_{k+1}) = f(x_k + \Delta x) = \underbrace{f(x_k) + f'(x_k) \frac{\Delta x^1}{1!} + f''(x_k) \frac{\Delta x^2}{2!} + \dots}_{\text{função original, representada em série de Taylor}}$$

Fazendo $f(x_{k+1}) = 0$, temos

$$f(x_k) + f'(x_k) \frac{\Delta x^1}{1!} + f''(x_k) \frac{\Delta x^2}{2!} + \dots = 0$$

Para determinar a incógnita Δx da série de Taylor, Newton assumiu que:

- a) x_k está em um intervalo suficientemente próximo da solução;
- b) Δx é suficientemente pequeno; e a função $f(x)$ pode ser representada apenas pelos seus dois primeiros termos da série de Taylor, truncando os termos de ordem superior a 2, portanto resulta uma aproximação de 1ª ordem equivalente à reta $r(x)$ definida pela eq. (9),

$$f(x_k + \Delta x) \cong f(x_k) + f'(x_k)\Delta x = 0$$

Observe que a equação não linear original, $f(x) = 0$, foi substituída por uma equação linearizada, aproximada, cuja incógnita x está presente em $\Delta x = x - x_k$.

Então, calculamos o novo x_{k+1} aproximado por

$$\Delta x = -\frac{f(x_k)}{f'(x_k)} \quad (11a)$$

e

$$x_{k+1} = x_k + \Delta x \quad (11b)$$

Note que o valor x_{k+1} da fórmula de Newton é uma aproximação de 1ª ordem para a raiz de $f(x) = 0$, não é o seu valor exato, pois houve um erro de truncamento da série de Taylor representativa de $f(x)$ da ordem de $O(\Delta x^2)$ dado por:

$$O(\Delta x^2) = f''(x_k)\frac{\Delta x^2}{2!} + f'''(x_k)\frac{\Delta x^3}{3!} + \dots$$

Assim, em vez de resolver a $f(x) = 0$ completa, com infinitas parcelas na série, resolvemos apenas uma aproximação de $f(x) = 0$, com apenas duas parcelas:

$$\underbrace{f(x_k) + f'(x_k) \frac{\Delta x^1}{1!} + f''(x_k) \frac{\Delta x^2}{2!} + \dots = 0}_{\text{eq. original, representada em série de Taylor com infinitas parcelas}} \Rightarrow \underbrace{f(x_k) + f'(x_k) \frac{\Delta x^1}{1!} = 0}_{\text{eq. simplificada, representada com 2 parcelas}}$$

No **Exercício 3.3** deste capítulo, propomos uma avaliação comparativa do método de Newton de 1ª ordem apresentado anteriormente, com uma extensão para um método de Newton de 2ª ordem (com os 3 primeiros termos da série de Taylor). Nesse exercício, poderemos avaliar que o número de operações aritméticas totais é menor no método de Newton tradicional de 1ª ordem.

No **Caderno de Exercícios e Respostas**, disponível no *link* <<http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>>, você encontrará os exercícios atualizados deste livro.

Agora, vamos apresentar um exemplo de aplicação do método de Newton e fazer sua comparação com o método da iteração linear.

Exemplo 3.8: determine a raiz de $x * \ln(x) - 3.2 = 0$ com $\alpha \in [2, 3]$ por Newton e compare sua eficiência em relação ao método de iteração linear.

Solução:

Dado x_0 , podemos determinar Δx e um novo x , conforme a eq. (10) ou as eqs. (11a) e (11b):

$$\Delta x = -\frac{f(x_0)}{f'(x_0)}$$

$$x = x_0 + \Delta x$$

$$f(x_0) = x_0 * \ln(x_0) - 3.2$$

$$f'(x_0) = \ln(x_0) + 1$$

Adotando valor inicial $x_0 = 2.5$, como no **Exemplo 3.6**, temos:

Tabela 3.7 – Resultados do Exemplo 3.8

k	x_k	$\Delta x = -\frac{f(x_k)}{f'(x_k)}$	$x_{k+1} = x_k + \Delta x$	$ \Delta x $
0	2.500000000000000	4.74496460892097e-01	2.97449646089210	4.74e-01
1	2.97449646089210	-2.02976178978191e-02	2.95419884299428	2.03e-02
2	2.95419884299428	-3.33196251968217e-05	2.95416552336908	3.33e-05
3	2.95416552336908	-9.01983101529758e-11	2.95416552327888	9.02e-11
4	2.95416552327888	0.00000000000000e+00	2.95416552327888	0.00e00
5	2.95416552327888			

Fonte: Elaboração própria.

A raiz obtida é 2.95416552327888, destacada em negrito, com critério de parada numericamente nulo $|\Delta x| = |x_{k+1} - x_k| \cong 0$, em $n = 5$ iterações, enquanto no método de iteração linear foram necessárias $n = 406$ iterações (**Exemplo 3.6 (b)**), sem fator de sub-relaxação. Então, o erro de truncamento também está no limite da precisão da variável *double*.

Também podemos implementar esse algoritmo de Newton em **calculadora científica** com recurso *Ans* (*Answer*) desta forma:

- digitamos o valor inicial, por exemplo 2.5, e pressionamos a tecla “=” para armazená-lo em *Ans*;
- digitamos a equação iterativa com x_0 representado pelo *Ans* (valor da resposta anterior), no caso:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

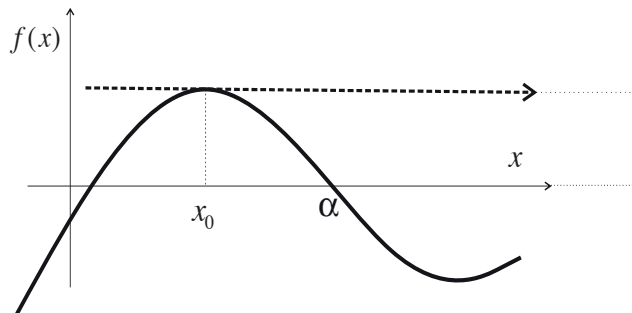
Ans - (*Ans* * ln (*Ans*) - 32) / (ln (*Ans*) + 1), no caso do **Exemplo 3.8**; e

- c) pressionando a tecla “=”, sucessivamente, o valor gerado será um novo x , que já fica armazenado em *Ans* e atualiza x_0 . Desse modo, a cada “=” pressionado, teremos uma nova iteração gerada no *display* da calculadora.

Algumas considerações sobre o método de Newton:

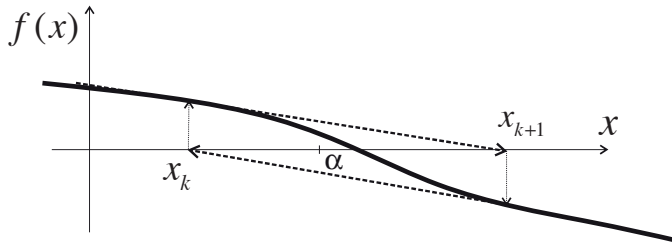
- a) Note que o método de Newton tem convergência rápida, e podemos comprovar que, normalmente, a convergência ou divergência desse método é do tipo quadrática, isto é, o incremento $|\Delta x|$ da iteração $(k + 1)$ é aproximadamente o quadrado do incremento da iteração anterior (k) , conforme vemos no quadro de resultados do **Exemplo 3.8**, onde o expoente de Δx dobra a cada iteração.
- b) O novo valor para x_{k+1} é o ponto de corte da reta tangente $r(x)$ com o eixo das abscissas, pois x_{k+1} é o zero dessa reta $r(x)$ aproximadora de $f(x)$.
- c) Pode ocorrer, durante a determinação dos x_{k+1} , que a reta tangente à $f(x)$ em $(x_k, f(x_k))$ seja paralela ao eixo da abscissa x , ou seja, $f'(x_k) = 0$, como no Gráfico 3.9 em $k = 0$.

Gráfico 3.9 – Método de Newton quando o valor inicial da raiz gera derivada nula



Fonte: Elaboração própria.

- d) Também pode ocorrer um ciclo repetitivo, ou *looping*, dos valores na sequência gerada pelos x_k e conseqüentemente o processo não converge nem diverge, conforme o Gráfico 3.10.

Gráfico 3.10 – Exemplo de *looping* dos valores na sequência gerada pelos x_k 

Fonte: Elaboração própria.

Para evitar a divisão por zero e os *loopings*, temos algumas possíveis soluções:

- i) escolher x_0 mais próximo possível de α ;
- ii) testar o uso de fatores de relaxação, conforme o **Exemplo 3.7** do método da iteração linear; e
- iii) tentar utilizar uma variante do método de Newton fixando o denominador em algum x_0 , com $f'(x_0) \neq 0$, desde que seja suficientemente próximo da raiz.

O algoritmo do método de Newton está disponível no **Caderno de Algoritmos** no arquivo **Cap3MetodosIterativos.m**.

3.2.2.3 Método da secante

Como o método de Newton necessita da $f'(x)$ e pode ser difícil ou até inviável obter a sua expressão, podemos utilizar aproximações para os valores de $f'(x)$.

Por definição, temos que:

$$f'(x_0) = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

Então, se Δx for suficientemente pequeno e não nulo, poderemos usar a aproximação $f'(x_0) \cong \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$, denominada de derivada numérica.

Assim, a derivada numérica depende de dois pontos, x_0 e $x_1 = x_0 + \Delta x$, e podemos calculá-la por:

$$f'(x_0) \cong \frac{f(x_1) - f(x_0)}{(x_1 - x_0)} \quad (12)$$

ou

$$f'(x_0) \cong \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} \quad (13)$$

Logo, aplicamos a derivada numérica dada pelas eqs. (12) ou (13) na eq. (10) e obtemos:

$$x_2 = x_0 - f(x_0) \Big/ \left(\frac{f(x_1) - f(x_0)}{(x_1 - x_0)} \right) \quad (14)$$

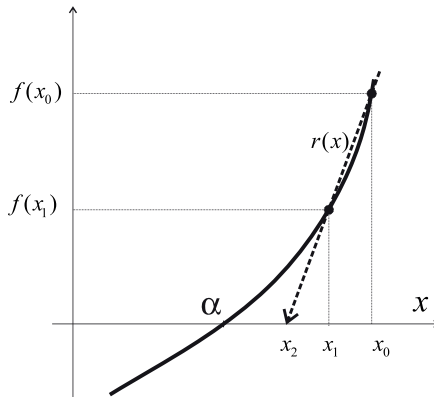
Calculado o novo valor x_2 , atualizamos os valores mais antigos x_0 e x_1 pelos valores mais atualizados disponíveis $x_0 = x_1$ e $x_1 = x_2$ e voltamos ao cálculo da derivada.

Observamos que o valor x_0 mais antigo de todos é descartado e os valores mais atualizados x_1 e x_2 são mantidos.

Se usarmos o próprio Δx , calculado no método de Newton, como incremento para cálculo da derivada, este assumirá naturalmente o último valor Δx calculado.

Repare que o x_2 é a interseção entre a reta secante, gerada pelos dois pontos anteriores $(x_0, f(x_0))$ e $(x_1, f(x_1))$, com o eixo das abscissas x , conforme o Gráfico 3.11.

Gráfico 3.11 – Método da secante



Fonte: Elaboração própria.

Para algoritmização desse método, podemos:

a) utilizar x_0 e Δx , ambos estimados inicialmente;

b) aproximar a derivada por meio da eq. (13)

$$f'(x_0) \cong \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x};$$

c) calcular um novo incremento Δx e solução x de cada iteração pelo método de Newton original, eqs. (11a) e (11b);

d) atualizar $x_0 = x$ e o incremento Δx (usando o próprio incremento do método de Newton); e

e) voltar ao passo (b), da mesma forma que no método de Newton original.

De modo alternativo, podemos aplicar diretamente a eq. (14) da seguinte forma:

a) estimar x_0 e x_1 iniciais;

b) calcular o novo valor x_2 , pela eq. (14);

c) calcular $f(x_2)$;

d) atualizar os valores mais antigos:

$$x_0 = x_1 \text{ e } f(x_0) = f(x_1) \text{ bem como } x_1 = x_2 \text{ e } f(x_1) = f(x_2);$$

e) voltar ao passo (b).

Observamos que, neste último algoritmo, a função $f(x)$ é calculada apenas uma vez, no passo (c), e apenas atualizada no passo (d), portanto é uma forma mais otimizada.

A fórmula do método da secante, eq. (14), é semelhante à do método da falsa posição, eq. (5), mas aqui não é necessário um intervalo inicial que contenha a solução.

Exemplo 3.9: determine a raiz de $x * \ln(x) - 3.2 = 0$ com $\alpha \in [2, 3]$ pelo método da secante.

Solução:

Adotando $x_0 = 2.5$ e $\Delta x = 2.5 * 10^{-06}$ como valores iniciais e aplicando o método da secante, temos:

Tabela 3.8 – Resultados do Exemplo 3.9

k	x_k	$f(x_k)$	$\Delta x = -\frac{f(x_k)}{f'(x_k)}$	$x_{k+1} = x_k + \Delta x$
0	2.500000000000000	1.91629123165171	4.74496337141240e-01	2.97449633714124
1	2.97449633714124	2.16590331986335	-1.95868762828261e-02	2.95490946085841
2	2.95490946085841	2.08014636987656	-7.45080498042262e-04	2.95416438036037
3	2.95416438036037	2.08308971185042	1.14298781384402e-06	2.95416552334818
4	2.95416552334818	2.08321640956046	-6.93020566723873e-11	2.95416552327888
5	2.95416552327888	2.08320359351340	0.00000000000000e+00	2.95416552327888
6	2.95416552327888			

Fonte: Elaboração própria.

A raiz obtida é 2.95416552327888, destacada em negrito, com critérios de parada numericamente nulos para $|\Delta x|$ e $|f(x_k)|$, em 6 iterações, enquanto o método de Newton obteve a mesma solução em 5 iterações, portanto o método da secante, no **Exemplo 3.9**, exige pouco esforço computacional adicional em relação ao método de Newton com a vantagem de dispensar a derivada analítica. Então, neste exemplo, o erro de truncamento também chegou no limite da precisão da variável *double*, mas devemos limitar o valor de Δx a um valor mínimo, para que a diferença entre $f(x_0 + \Delta x)$ e $f(x_0)$ não seja numericamente nula, ou $f'(x_k)$ não seja nula.

Disponibilizamos as duas formas do algoritmo do método da secante no **Caderno de Algoritmos** no arquivo **Cap3MetodosIterativos.m** como método de Newton com derivadas numéricas e como método da secante.

A seguir, vamos apresentar uma adaptação do método de Newton para os casos em que a solução α de $f(x) = 0$ seja um ponto crítico.

3.2.2.4 Tratamento geral de funções com derivada nula em $x = \alpha$

Para equações de raízes simples (não repetidas), com $f'(x)$ nulo em algum dos seus valores x_k , podemos tentar outro(s) valor(es) inicial(ais) ou fixar um valor aproximado da derivada, entre outras possibilidades.

Para equações de raízes múltiplas (repetidas), a fórmula do método de Newton apresentará, além de $\lim_{x \rightarrow \alpha} f(x) \rightarrow 0$, também $\lim_{x \rightarrow \alpha} f'(x) \rightarrow 0$, o que provoca convergência lenta, isto é, perde a convergência quadrática, e gera uma indeterminação $0/0$ para Δx no limite de convergência, conforme veremos na próxima seção.

Vamos primeiramente avaliar os efeitos da convergência com derivada tendendo a zero em uma função não polinomial e algumas das possíveis metodologias de soluções apresentadas na literatura pertinente.

Exemplo 3.10: obtenha uma raiz de $e^x - x - 1 = 0$ com $x_0 = 1.0$ por meio do método de Newton.

Solução:

$$f(x) = e^x - x - 1$$

$$f'(x) = e^x - 1$$

Aplicando a fórmula do método de Newton, conforme as eqs. (11a) e (11b), com $x_0 = 1$, e destacando que a raiz exata dessa equação é 0.0 , temos:

Tabela 3.9 – Resultados do Exemplo 3.10

k	x_k	$f(x_k)$	$f'(x_k)$	$\Delta x = -\frac{f(x_k)}{f'(x_k)}$	$x_{k+1} = x_k + \Delta x$	$ f(x) + \Delta x $
0	1.000000e+00	7.182818e-01	1.718282e+00	-4.180233e-01	5.819767e-01	6.256190e-01
1	5.819767e-01	2.075957e-01	7.895724e-01	-2.629217e-01	3.190550e-01	3.196937e-01
2	3.190550e-01	5.677201e-02	3.758270e-01	-1.510589e-01	1.679962e-01	1.659948e-01
3	1.679962e-01	1.493591e-02	1.829321e-01	-8.164730e-02	8.634887e-02	8.548502e-02
4	8.634887e-02	3.837726e-03	9.018660e-02	-4.255317e-02	4.379570e-02	4.352636e-02
⋮						
22	3.389852e-07	5.750955e-14	3.389853e-07	-1.696521e-07	1.693332e-07	1.696521e-07
23	1.693332e-07	1.443290e-14	1.693332e-07	-8.523373e-08	8.409945e-08	8.523373e-08
24	8.409945e-08	3.552714e-15	8.409945e-08	-4.224420e-08	4.185525e-08	4.224420e-08
25	4.185525e-08	0.000000e+0	4.185525e-08	0.000000e+00	4.185525e-08	0.000000e+00
26	4.185525e-08					

Fonte: Elaboração própria.

Em 26 iterações, geramos $x_{26} = 4.185525e - 08$, destacada em negrito, com critérios de parada numericamente nulos para $|\Delta x|$ e $|f(x_k)|$. Como o Δx se tornou nulo, a partir de $x_{26} = \underline{0.00000004185525}$, os demais valores de x_k ficariam constantes e com apenas 8 dígitos exatos (sublinhados).

Observe, na Tabela 3.9, que:

- Os critérios de parada foram reduzidos a zero, mas a raiz convergida ainda não é a exata, pois deveria ser zero em todos os seus dígitos significativos. Com variável *double*, seriam 16 dígitos exatos, mas a raiz aproximada x_{26} atingiu apenas 8 dígitos exatos.
- Ocorreu a perda da convergência quadrática, uma vez que o Δx novo foi reduzido apenas pela metade do anterior a cada iteração. A convergência quadrática, normal no método de Newton, reduz o Δx novo para o quadrado do Δx da iteração anterior.

Para resolver esse problema, Cheney e Kincaid (2012) apresentam duas possíveis soluções:

- a) Uma delas usa uma correção na fórmula de cálculo do incremento das raízes pela multiplicidade M da raiz α da seguinte forma:

$$\Delta x = -\frac{f(x_k)}{f'(x_k)} \quad (15a)$$

$$x_{k+1} = x_k + M\Delta x \quad (15b)$$

essa correção apresenta melhores resultados para casos como o do **Exemplo 3.10** (resultados não apresentados neste livro), mas é **necessário conhecer previamente** essa **multiplicidade** M . Nesse exemplo, como $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$, no limite da precisão digital das variáveis utilizadas (no caso *double*), podemos adotar multiplicidade equivalente $M = 2$, conforme **Propriedade 13** de polinômios, que veremos mais adiante.



Veremos como calcular a multiplicidade estimada na seção 3.3, sobre raízes de equações polinomiais.

- b) Outra possibilidade utiliza uma correção na fórmula de cálculo do incremento da raiz quando $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$, determinando alternativamente a raiz de $g(x) = f(x) / f'(x) = 0$, pois toda raiz α que anula $f(x)$ também anula $g(x)$, então, aplicamos o método de Newton à função modificada $g(x) = 0$:

$$\Delta x = -\frac{g(x_k)}{g'(x_k)} = -\frac{\frac{f(x_k)}{f'(x_k)}}{\frac{[f'(x_k)]^2 - f(x_k)f''(x_k)}{[f'(x_k)]^2}} = -\frac{f(x_k)f'(x_k)}{[f'(x_k)]^2 - f(x_k)f''(x_k)} \quad (16)$$

ou

$$\Delta x = -\hat{M} \frac{f(x_k)}{f'(x_k)} \quad (17)$$

em que

$$\hat{M} = \frac{1}{g'(x_k)} = \left\{ \frac{[f'(x_k)]^2}{[f'(x_k)]^2 - f(x_k)f''(x_k)} \right\} \quad (18)$$

Com $x_{k+1} = x_k + \Delta x$, dado pela eq. (11b).

Exemplo 3.11: reaplique o método de Newton, modificando $f(x) = 0$ para $g(x) = \frac{f(x)}{f'(x)} = 0$, conforme as eqs. (17) e (18), e obtenha uma raiz de: $e^x - x - 1 = 0$ com $x_0 = 1.0$.

Solução:

$$f(x) = e^x - x - 1$$

$$f'(x) = e^x - 1$$

$$f''(x) = e^x$$

Tabela 3.10 – Resultados do Exemplo 3.11

k	x_k	$f(x_k)$	$f'(x_k)$	$f''(x_k)$	\tilde{M} eq. (18)	$\Delta x = -\tilde{M} \frac{f(x_k)}{f'(x_k)}$
0	1.000000e+00	7.182818e-01	1.718282e+00	2.718282e+00	2.952492e+00	-1.234211e+00
1	-2.342106e-01	2.540578e-02	-2.088048e-01	7.911952e-01	1.855412e+00	2.257523e-01
2	-8.458280e-03	3.567061e-05	-8.422609e-03	9.915774e-01	1.994377e+00	8.446390e-03
3	-1.189018e-05	7.068790e-11	-1.189011e-05	9.999881e-01	1.999991e+00	1.189014e-05
4	-4.176666e-11	0.000000e+00	-4.176670e-11	1.000000e+00	1.000000e+00	0.000000e+00
5	-4.176666e-11					

Fonte: Elaboração própria.

Observe que:

- em $k = 3$, o valor de \tilde{M} tende a 2, com $f(x \rightarrow \alpha) \rightarrow 0$ e $f'(x \rightarrow \alpha) \rightarrow 0$; e
- em $k = 4$, $f(x_k) = 0$, logo \tilde{M} se torna unitário ($\tilde{M} = 1$ sempre que $f(x) = 0$), $\Delta x = 0$ e x_5 não vai mais se alterar.

Assim, $x_5 = \underline{0.0000000000}417667$, também em negrito, é uma raiz convergida com 11 dígitos exatos (sublinhados), e apesar de ser melhor do que a solução obtida no **Exemplo 3.10** (com 8 dígitos exatos), os critérios de parada também zeraram antes de chegarmos à raiz com precisão de 16 dígitos da variável *double* utilizada.

Segundo Schröder (1992), o termo $1/g'(x_k)$ dado pela eq. (18) representa uma estimativa clássica do valor da multiplicidade \hat{M} de raízes, mas essa estimativa é válida para raízes estimadas próximas da raiz exata α .

No **Exemplo 3.11**, a função e sua derivada tendem a zero, $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$, fazendo \hat{M} convergir para 2 (antes de $f(x)$ se tornar nula), conforme **Propriedade 13** de polinômios, que veremos mais adiante.

Essa correção, dada pelas eqs. (17) e (18), também apresentou melhores resultados, mas não atinge a precisão no limite da variável *double*.

Observamos que as eqs. (15) e (17) serão equivalentes à medida que a eq. (18) da multiplicidade estimada \hat{M} se torne a multiplicidade M exata, ao longo de um processo iterativo.

Em ambos os casos anteriores, usando as eqs. (15) ou (17), o incremento Δx tende a zero, mas a raiz convergida ainda fica distante do valor exato, não atingindo precisão em todos os dígitos disponíveis.

Uma terceira alternativa, mais robusta, para se determinar raízes quando $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$, pode ser a aplicação da **Regra de L'Hospital** ao cálculo de Δx para resolver a indeterminação numérica que ocorre no limite do processo de convergência, como aplicado por [Galántai e Hegedus \(2010\)](#).



No artigo, os autores apresentam uma abrangente revisão bibliográfica sobre aceleradores de convergência do método de Newton e formas de aproximação da estimativa do valor da multiplicidade.

Neste livro, vamos propor um algoritmo alternativo aplicando a Regra de L'Hospital como acelerador do método de Newton, primeiramente para funções não polinomiais.

Pela Regra de L'Hospital:

Sejam $f_1(x)$ e $f_2(x)$ funções contínuas e deriváveis em um intervalo I , com $f_2'(x) \neq 0, \forall x \in I$, e seja a um valor interno de I , tal que $f_1(a) = f_2(a) = 0$, supondo que exista $\lim_{x \rightarrow a} \left(\frac{f_1'(x)}{f_2'(x)} \right)$, então existe $\lim_{x \rightarrow a} \left(\frac{f_1(x)}{f_2(x)} \right)$ e $\lim_{x \rightarrow a} \left(\frac{f_1(x)}{f_2(x)} \right) = \lim_{x \rightarrow a} \left(\frac{f_1'(x)}{f_2'(x)} \right)$.

Nesse caso, vamos considerar que, no limite da convergência iterativa, teremos uma indeterminação $0/0$ na eq. (11a) de Δx , quando $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$ para a raiz $x \rightarrow \alpha$ convergida.

A Regra de L'Hospital pode ser diretamente aplicável na equação do cálculo do incremento Δx , eq. (11a), nesse caso, com $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$, teremos:

$$\lim_{x_k \rightarrow \alpha} (\Delta x) = \lim_{x_k \rightarrow \alpha} \left(-\frac{f(x_k)}{f'(x_k)} \right) = \lim_{x_k \rightarrow \alpha} \left(-\frac{f'(x_k)}{f''(x_k)} \right)$$

Então, na região de convergência, $x_k \rightarrow \alpha$, teremos:

$$\Delta x = -\frac{f'(x_k)}{f''(x_k)} \quad (19)$$

Com x_{k+1} dado pela mesma eq. (11b).

Exemplo 3.12: replique o método de Newton, modificado pela Regra de L'Hospital, eq. (19), para obter uma raiz de $e^x - x - 1 = 0$ a partir de $x_0 = 1.0$.

Solução:

$$f(x) = e^x - x - 1$$

$$f'(x) = e^x - 1$$

$$f''(x) = e^x$$

Podemos aplicar a fórmula do método de Newton modificado, conforme a eq. (19), pois sabemos previamente dos **Exemplos 3.10 e 3.11** que, na região da raiz, temos $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$, então:

Tabela 3.11 – Resultados do Exemplo 3.12

k	x_k	$f'(x_k)$	$f''(x_k)$	$\Delta x = -\frac{f'(x_k)}{f''(x_k)}$	$x_{k+1} = x_k + \Delta x$	$f(x)$	$ f(x) + \Delta x $
0	1.000000e+00	1.718282e+00	2.718282e+00	-6.321206e-01	3.678794e-01	7.678842e-02	7.089090e-01
1	3.678794e-01	4.446679e-01	1.444668e+00	-3.077994e-01	6.008007e-02	1.841501e-03	3.096409e-01
2	6.008007e-02	6.192157e-02	1.061922e+00	-5.831087e-02	1.769199e-03	1.565957e-06	5.831244e-02
3	1.769199e-03	1.770765e-03	1.001771e+00	-1.767635e-03	1.564111e-06	1.223022e-12	1.767635e-03
4	1.564111e-06	1.564112e-06	1.000002e+00	-1.564110e-06	1.223322e-12	0.000000e+00	1.564110e-06
5	1.223322e-12	1.223244e-12	1.000000e+00	-1.223244e-12	7.783746e-17	0.000000e+00	1.223244e-12
6	7.783746e-17	0.000000e+00	1.000000e+00	0.000000e+00	7.783746e-17	0.000000e+00	0.000000e+00
7	7.783746e-17						

Fonte: Elaboração própria.

Assim, em $k = 7$ iterações, calculamos $x_7 = 7.78374589094591e-17$, com $\Delta x = 0$ e $f(x) = 0$, logo $x_7 = \underline{0.0000000000000000778374589094591}$ é a raiz encontrada com 16 dígitos significativos (sublinhados).

Considerações sobre o método de Newton modificado pela Regra de L'Hospital:

- esse processo iterativo corrigido recupera a convergência quadrática, em que Δx na iteração $k + 1$ é o quadrado do Δx da iteração k anterior, conforme os valores de Δx na tabela de resultados do **Exemplo 3.12** (observe que o expoente negativo da notação científica de cada Δx dobra de valor a cada iteração); e
- nesse processo é necessário conhecer previamente o comportamento da função e das suas derivadas ao longo da convergência, no caso, a eq. (19) é válida somente para $f(x) \rightarrow 0$ e $f'(x) \rightarrow 0$. Se as demais derivadas também tenderem a zero, será necessário reaplicar a Regra de L'Hospital.

Vamos aprofundar a questão de “derivada e função tenderem a zero simultaneamente” ao longo do processo de convergência na seção 3.3.6 sobre equações polinomiais.

3.2.2.5 Rotinas de funções predefinidas

O método de Newton também pode ser utilizado para elaborar rotinas de cálculo de valores de funções predefinidas para máquinas digitais, conforme os exemplos a seguir.

3.2.2.5.1 Recíproco de um número

Fazemos uso do método de Newton para elaborar uma rotina dedicada a obter o recíproco $1/c$ com $c \in \mathbb{C} - \{0\}$ (complexos não nulos).

Com essa rotina, desejamos determinar um x tal que

$$x = 1/c \text{ ou } 1/x = c \Rightarrow f(x) = \frac{1}{x} - c = 0$$

Determinando a raiz dessa equação, sem usar a divisão, teremos o recíproco de c . Aplicando Newton, conforme as eqs. (11a) e (11b), com

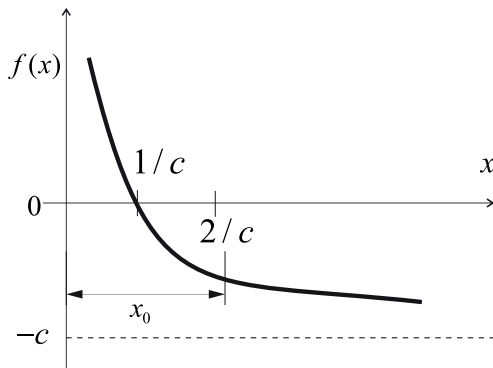
$$f(x) = \frac{1}{x} - c = 0 \text{ e } f'(x) = -\frac{1}{x^2} \Rightarrow \Delta x = -\frac{\frac{1}{x_k} - c}{-\frac{1}{(x_k)^2}} = (x_k)^2 \left(\frac{1}{x_k} - c \right)$$

$$\text{teremos } x_{k+1} = x_k + \Delta x \Rightarrow x_{k+1} = x_k (2 - c * x_k).$$

Qual deverá ser o valor inicial x_0 para que essa expressão sempre convirja?

No Gráfico 3.12, temos a representação da função geradora $f(x) = \frac{1}{x} - c$ para $c > 1$.

Gráfico 3.12 – Representação de $f(x) = 1/x - c$ para $c > 1$



Fonte: Elaboração própria.

Devemos estimar o valor inicial x_0 mais próximo de $1/c$, mas para garantimos a convergência, o valor inicial x_0 deve estar contido no intervalo $0 < x_0 < 2/c$. Por exemplo, se c é da ordem de $O(10^e) \Rightarrow$ o valor inicial estimado para x_0 deverá ser da ordem de $O(10^{-e})$.

Exemplo 3.13: obtenha o recíproco de $c = 425.32$ sem utilizar a operação de divisão.

Solução:

Considerando que c é da ordem de $O(10^3)$, pois $c = 0.42532 * 10^3$, tomamos $x_0 = 0.001 = 10^{-3}$ e aplicamos a equação $\Rightarrow x_{k+1} = x_k (2 - c * x_k)$:

Tabela 3.12 – Resultados do Exemplo 3.13

k	x_k	$ \Delta x $
0	0.001000000000000000	
1	0.001574680000000000	5.74680000000000e-04
2	0.00209472925400723	5.20049254007232e-04
3	0.00232320085777991	2.28471603772682e-04
4	0.00235083814577217	2.76372879922607e-05
5	0.00235117083601085	3.32690238675701e-07
6	0.00235117088309978	4.70889325795976e-11
7	0.00235117088309978	8.67361737988404e-19

Fonte: Elaboração própria.

Então, temos o valor equivalente ao exato $1/425.32 = 0.00235117088309978$ em 7 iterações, com critério de parada $|\Delta x| < 10^{-16}$, convergido no limite da precisão da variável *double*.

No **Caderno de Algoritmos**, confira o algoritmo de Newton para o recíproco de c no arquivo **Cap3reciproco.m**.

3.2.2.5.2 Raiz quadrada de um número

Fazemos uso do método de Newton para criar uma rotina que possa obter \sqrt{c} , $c \in \mathbb{R}^+$. Com essa rotina, desejamos um x tal que $x = \sqrt{c}$, o que equivale a obter a raiz positiva da equação polinomial quadrática $x^2 - c = 0$ sem usar a radiciação.

Desse modo, aplicamos o método de Newton, conforme as eqs. (11a) e (11b), com: $f(x) = x^2 - c$ e $f'(x) = 2x$,

$$\Delta x = -\frac{(x_k)^2 - c}{2x_k}$$

$$x_{k+1} = x_k - \left(\frac{(x_k)^2 - c}{2x_k} \right) \Rightarrow \left(x_k + \frac{c}{x_k} \right) 0.5$$

Como a $f(x) = x^2 - c$ é estritamente crescente e convexa para todo x positivo, então a expressão anterior do x_{k+1} gera uma sequência sempre convergente para raiz quadrada de c , para todo $x_0 > 0$.

A questão é: como tomar um x_0 para que essa convergência seja a mais rápida possível?

Uma boa alternativa é tomar

$$x_0 = \left(1.68 - \frac{1.29}{0.84 + m} \right) 10^p \quad (3.16^q) \quad (20)$$

Em que m é a mantissa de x dada por $(0 . d_1 \dots d_t)$.

Esse valor inicial é o resultado da aproximação da raiz quadrada de c , expresso na forma $c = (0 . d_1 \dots d_t)10^{2p+q}$, com p inteiro e $q = 0$ ou $q = 1$, e a expressão entre parênteses sendo uma aproximação ajustada para a mantissa $m = (0 . d_1 \dots d_t)$

Exemplo 3.14: obtenha $\sqrt{685.72} \Rightarrow c = 0.68752 * 10^3$.

Solução:

Tomamos o valor inicial:

$$\begin{cases} m = 0.68572 \text{ (mantissa)} \\ p = 1 \\ q = 1, \text{ pois } 2p + q = 3 \end{cases}$$

$$x_0 = \left(1.68 - \frac{1.29}{0.84 + 0.68572} \right) 10^1 \quad (3.16)^1 = 26.370122$$

Tabela 3.13 – Resultados do Exemplo 3.14

k	x_k	$ \Delta x / x $
0	26.370122	
1	26.18689694144919144	6.996822e-03
2	26.18625594488216635	62.447836e-05
3	26.18625593703689702	2.995949e-10
4	26.18625593703689702	0

Fonte: Elaboração própria.

Então, temos o resultado equivalente a $\sqrt{685.72} = 26.18625593703689702$ em 4 iterações, com critério de parada relativo $|\Delta x / x| < 10^{-16}$, convergido em variável *double*.

O algoritmo de Newton para determinar a raiz quadrada de c está disponível no **Caderno de Algoritmos** no arquivo **Cap3raizquadrada.m**.

3.3 SOLUÇÃO DE EQUAÇÕES POLINOMIAIS

Vamos abordar a solução de equações da classe das **polinomiais** devido à sua importância histórica, à frequência com que aparecem nos modelos matemáticos e às particularidades algébricas desse tipo de equações, a partir de metodologias clássicas (BURDEN; FAIRES, 2011; CHENEY; KINCAID, 2012), e uma proposta alternativa a respeito da determinação de raízes múltiplas baseada em Galántai e Hegedus (2010).

Uma equação polinomial é toda expressão do tipo $P_n(x) = 0$, em que $P_n(x) = a_1 x^n + a_2 x^{n-1} + \dots + a_n x + a_{n+1}$ é um polinômio de grau n se $a_1 \neq 0$.

A solução das equações $P_n(x) = 0$ tem o seguinte histórico de desenvolvimento:

$$\begin{aligned} n = 1 &\Rightarrow P_1(x) = a_1 x^1 + a_2 = 0 && \text{(Euclides, séc. III a.C.)} \\ &\Rightarrow \alpha = \frac{a_2}{a_1} \end{aligned}$$

$$\begin{aligned} n = 2 &\Rightarrow P_2(x) = a_1 x^2 + a_2 x^1 + a_3 = 0 && \text{(Sridhara/Bhaskara, séc. XII)} \\ &\Rightarrow \alpha = \frac{-a_2 \pm \sqrt{a_2^2 - 4a_1 a_3}}{2a_1} \end{aligned}$$

$$n = 3 \Rightarrow P_3(x) = a_1 x^3 + a_2 x^2 + a_3 x^1 + a_4 = 0 \quad \text{(Cardano/Tartaglia, 1535)}$$

Para resolvê-la por radiciação, substituímos $x = y - a_2 / (3a_1)$ em $P_3(x) = 0 \Rightarrow y^3 + p y + q = 0$, em que $p = a_3 - a_2^2 / (3a_1)$ e $q = a_4 + (2a_2^3 - 9a_2 a_3) / (27a_1)$ gera as soluções:

$$\Rightarrow \alpha_1 = \sqrt[3]{\frac{-q}{2} + \sqrt{\frac{p^3}{27} + \frac{q^2}{4}}} + \sqrt[3]{\frac{-q}{2} - \sqrt{\frac{p^3}{27} + \frac{q^2}{4}}}$$

$$\Rightarrow \alpha_2 = \sqrt[3]{\frac{-q}{2} + \sqrt{\frac{p^3}{27} + \frac{q^2}{4}}} + \sqrt[3]{\frac{-q}{2} + \sqrt{\frac{p^3}{27} + \frac{q^2}{4}}}$$

$$\Rightarrow \alpha_3 = \sqrt[3]{\frac{-q}{2} - \sqrt{\frac{p^3}{27} + \frac{q^2}{4}}} + \sqrt[3]{\frac{-q}{2} - \sqrt{\frac{p^3}{27} + \frac{q^2}{4}}}$$

$n = 4 \Rightarrow P_4(x) = a_1x^4 + a_2x^3 + a_3x^2 + a_4x^1 + a_5 = 0$ (Ferrari, aprox. 1550)
O método de Ferrari permite a redução de qualquer equação de grau $n = 4$ para $n = 3$.

$$n = 5 \Rightarrow P_5(x) = a_1x^5 + a_2x^4 + a_3x^3 + a_4x^2 + a_5x^1 + a_6 = 0$$

Em 1824, Abel provou ser impossível solver por **radiciação** todas as $P_n(x) = 0$ para $n \geq 5$ e, em 1832, Galois classificou as possíveis e impossíveis.



Em decorrência da impossibilidade de solver por radiciação todas as polinomiais, precisamos usar outras metodologias. Neste estudo, continuaremos fazendo uso dos métodos iterativos.

Uma característica importante das equações $P_n(x) = 0$ é que sabemos previamente quantas raízes elas possuem. Para tanto, basta usar o teorema fundamental da álgebra e seus corolários, os quais provam que:

Propriedade 1: toda $P_n(x) = 0$ possui exatamente n soluções α , reais ou complexas, distintas ou repetidas (múltiplas).

Como vamos usar métodos iterativos para solver uma $P_n(x) = 0$ qualquer, vamos abordar antes algumas formas de localização das raízes Reais e Complexas, para facilitar a escolha de valores iniciais x_0 .

3.3.1 Localização de raízes de equações polinomiais

Conforme vimos no início deste capítulo, uma raiz Real de $f(x) = 0$ pode ser localizada via teorema de Bolzano. Nesta seção, vamos apresentar algumas propriedades algébricas dos polinômios que ampliam as possibilidades de localização de todas as n raízes de $P_n(x) = 0$, Reais e/ou Complexas.

A determinação de intervalos $[a, b]$ e círculos, que contenham respectivamente todas as raízes Reais e Complexas, pode ser estabelecida por cotas-limite. Vamos apresentar algumas dessas cotas a seguir.

Propriedade 2 – Cota do módulo máximo: em uma $P_n(x) = a_1x^n + a_2x^{n-1} + \dots + a_nx + a_{n+1} = 0$, com $a_1 \neq 0$, para toda raiz $\alpha \in \mathbb{C} \Rightarrow |\alpha| < r_{max}$, em que

$$r_{max} = 1 + \frac{\max\{|a_2|, |a_3|, \dots, |a_{n+1}|\}}{|a_1|}.$$

A origem desse limite máximo r_{max} de todas as raízes de $P_n(x) = 0$ também é atribuído a Cauchy, embora seja normalmente menos exato do que a chamada “cota de Cauchy”, que veremos na **Propriedade 3**.

Dessa forma,

- a) se α for Real, ela estará sempre contida no intervalo $(-r_{max}, +r_{max})$, que poderá ser rastreado por um processo de varredura usando o teorema de Bolzano para localizar subintervalos menores que contenha(m) raiz(es) Real(is); e
- b) se a raiz for um Complexo, então estará contida no círculo C de centro $(0, 0)$ e raio r_{max} .

Lembre-se de que o módulo de um número complexo $a + b * i$ é dado por $\sqrt{a^2 + b^2}$, em que $(i = \sqrt{-1})$.

Podemos também calcular um raio mínimo r_{min} para as raízes de $P_n(x) = 0$, ou seja, um raio de um círculo interno que não contenha raízes, conforme o Gráfico 3.13.

Para calcular o raio mínimo, geramos um polinômio auxiliar $P_n^{**}(x) = x^n P_n(1/x)$ cujos zeros são os recíprocos das raízes de $P_n(x) = 0$, desde que o $P_n(x) = 0$ não contenha raízes nulas ($a_{n+1} \neq 0$), para se evitar divisões por zero.

Então, para $P_n(x) = 0$ com $a_{n+1} \neq 0$, temos:

$$P_n(1/x) = a_1(1/x)^n + a_2(1/x)^{n-1} + \dots + a_n(1/x) + a_{n+1} = 0$$

Então, $P_n^{**}(x) = x^n P_n(1/x)$ resulta em:

$$P_n^{**}(x) = a_1 + a_2x + \dots + a_nx^{n-1} + a_{n+1}x^n = 0 \quad (21)$$

Podemos aplicar, dessa forma, a mesma cota máxima à $P_n^{**}(x) = 0$ e determinar um raio máximo auxiliar r_2 , limite superior das suas raízes, via

$$r_2 = 1 + \frac{\max\{|a_1|, |a_2|, |a_3|, \dots, |a_n|\}}{|a_{n+1}|} \quad (22)$$

Observe na eq. (22) que é necessário $a_{n+1} \neq 0$, pois esse coeficiente que se tornou o de maior grau em $P_n^{**}(x)$ passa ao denominador.

O recíproco desse limite r_2 corresponde ao raio mínimo das raízes de $P_n(x) = 0$, ou seja,

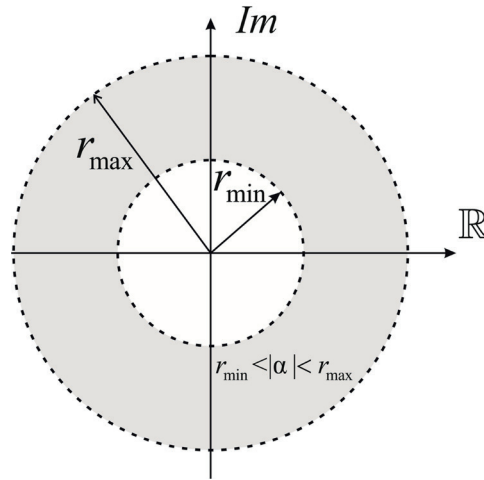
$$r_{\min} = 1/r_2 \quad (23a)$$

Neste caso,

$$r_{\min} = 1 / \left(1 + \frac{\max\{|a_1|, |a_2|, |a_3|, \dots, |a_n|\}}{|a_{n+1}|} \right) \quad (23b)$$

Logo, a maior raiz de $P_n^{**}(x) = 0$ está dentro do limite r_2 , mas essa sua maior raiz corresponde à menor raiz de $P_n(x) = 0$.

Lembramos que a **Propriedade 2** vale para raízes Reais e Complexas, ou seja, esses limites do módulo das raízes podem ser interpretados em um gráfico cartesiano com eixo Real e Imaginário (Im = eixo puramente Complexo), onde toda raiz α de $P_n(x) = 0$ está contida no intervalo $r_{\min} < |\alpha| < r_{\max}$, ou seja, α está dentro do anel destacado em cinza do Gráfico 3.13.

Gráfico 3.13 – Anel que contém todas as raízes α de $P_n(x) = 0$ 

Fonte: Elaboração própria.

Exemplo 3.15: localize as raízes de $3x^6 + 4x^3 - 2x^2 - 6 = 0$ por meio das cotas-limite de **módulo máximo e mínimo**.

Solução:

Como $P_6(x) = 3x^6 + 4x^3 - 2x^2 - 6$, aplicando a **Propriedade 2**, com $a_1 = 3$ e $\max\{|0|, |0|, |4|, |-2|, |0|, |-6|\} = 6$, temos pela eq. (22):

$$r_{\max} = 1 + \frac{6}{|3|} = 3$$

Para calcular um raio mínimo para essas raízes, podemos aplicar a eq. (23b):

$$r_{\min} = 1 / \left(1 + \frac{\max\{|3|, |0|, |0|, |4|, |-2|, |0|\}}{|-6|} \right)$$

$$r_{\min} = 0.6$$

Assim, as raízes de $P_6(x) = 0$ estão entre $0.6 < |\alpha| < 3.0$.

Esse resultado pode ser comprovado quando determinamos as 6 raízes de $P_6(x) = 0$ (via função `roots()` do Octave), conforme segue:

$$x_1 = -1.373431183527068 + 0.0000000000000000i$$

$$x_2 = 0.580108028676416 + 1.135516978130294i$$

$$x_3 = 0.580108028676416 - 1.135516978130294i$$

$$x_4 = 1.035657483096530 + 0.0000000000000000i$$

$$x_5 = -0.411221178461147 + 0.834073854054218i$$

$$x_6 = -0.411221178461147 - 0.834073854054218i$$

cuja raiz de módulo:

a) máximo é x_1 (módulo 1.373431183527068);

b) mínimo é x_5 (módulo 0.929936584736740).

Propriedade 3 – Cota de Cauchy: na equação polinomial $P_n(x) = 0$, todas as suas raízes α Reais ou Complexas satisfazem $|\alpha| \leq r_{max}$, sendo

$$r_{max} = \lim_{k \rightarrow \infty} x_k$$

$$\text{com } x_{k+1} = \left(\sum_{i=2}^{n+1} \left| \frac{a_i}{a_1} \right| x_k^{n-i+1} \right)^{1/n}$$

Nessa propriedade, são aplicados k passos iterativos a partir do valor inicial $x_0 = 0$ até que algum critério de parada seja satisfeito.

Exemplo 3.16: calcule as cotas-limite máxima e mínima de Cauchy das raízes da equação polinomial $3x^6 + 4x^3 - 2x^2 - 6 = 0$, com precisão de 16 dígitos significativos.

Solução:

A partir de $x_0 = 0$ (valor inicial) e $a_1 = 3$, geramos a equação iterativa, como se o x do termo de maior grau fosse isolado, análogo a um método de iteração linear, considerando coeficientes em módulo e com valores de x_k tomados em um passo iterativo anterior, logo:

$$x_{k+1} = \left((4x_k^3 + 2x_k^2 + 6) / 3 \right)^{1/6}$$

Nesse caso, para uma cota com 16 dígitos significativos, então $|x_{k+1} - x_k| < 10^{-15}$:

Tabela 3.13 – Resultados do Exemplo 3.16

k	x_k	$ x_{k+1} - x_k $
0	0	-
1	1.12246204830937	1.12246204830937
2	1.29541512584639	0.172953077537016
3	1.34864877400131	0.0532336481549256
⋮		
31	1.37343118352707	6.66133814775094e-16

Fonte: Elaboração própria.

O valor de x_k com 16 dígitos convergidos é o $r_{max} = 1.37343118352707$ com $|x_{k+1} - x_k| \cong 0$.

Para calcular a cota-limite mínima de Cauchy, tomamos o polinômio auxiliar $P_n^{**}(x) = x^n P_n(1/x)$, definido pela eq. (21), que contém os inversos multiplicativos das raízes de $P_6(x) = 0$:

$$P_6^{**}(x) = x^6 P_6(1/x) = 3 + 4x^3 - 2x^4 - 6x^6 = 0$$

Então, a partir de $x_0 = 0$ (valor inicial), geramos a equação iterativa a seguir:

$$x_k = \left((2x_k^4 + 4x_k^3 + 3) / 6 \right)^{1/6}$$

Depois da convergência, concluímos que a maior raiz de $P_6^{**}(x)$ está dentro do limite $r_2 = 1.11793643455463$, mas essa maior raiz de $P_6^{**}(x)$ corresponde ao inverso do limite da menor raiz de $P_6(x) = 0$. Logo, o limite inferior das raízes de $P_6(x) = 0$ é dado pela eq. (23a),

$$r_{min} = 1 / r_2 = 0.894505241166406$$

Portanto, pela cota de Cauchy, todas as 6 raízes de $P_6(x) = 0$ também estão no intervalo $0.894505241166406 \leq |\alpha| \leq 1.37343118352707$.

Propriedade 4 – Cota de Kojima: na equação polinomial $P_n(x) = 0$ ($n \geq 2$), toda a raiz α , Real ou Complexa, satisfaz $\alpha \leq |r_{max}|$ e $r_{max} = max_1 + max_2$. Os valores de max_1 e max_2 são os dois maiores valores da sequência:

$$q_i = \left\{ \left| \frac{a_{i+1}}{a_1} \right|^{\frac{1}{i}} \right\} \text{ com } i = 1, \dots, n.$$

Exemplo 3.17: calcule as cotas-limite de raízes de Kojima para o polinômio $P_6(x) = 3x^6 + 4x^3 - 2x^2 - 6 = 0$.

Solução:

Como os coeficientes são $a_1 = 3$, $a_2 = 0$, $a_3 = 0$, $a_4 = 4$, $a_5 = -2$, $a_6 = 0$ e $a_7 = -6$, então a sequência de fatores q_i é:

$$q_i = \left\{ \left| \frac{0}{3} \right|^{\frac{1}{1}}, \left| \frac{0}{3} \right|^{\frac{1}{2}}, \left| \frac{4}{3} \right|^{\frac{1}{3}}, \left| \frac{-2}{3} \right|^{\frac{1}{4}}, \left| \frac{0}{3} \right|^{\frac{1}{5}}, \left| \frac{-6}{3} \right|^{\frac{1}{6}} \right\} = \{0, 0, 1.100642, 0.903602, 0, 1.122462\}$$

Tomando os dois maiores valores: $max_1 = 1.122462$ e $max_2 = 1.100642$, temos

$$r_{max} = max_1 + max_2 = 2.223104$$

Para determinar o raio interno do anel, podemos gerar o polinômio auxiliar dado pela eq. (21),

$$P_6^{**}(x) = -6x^6 + 0x^5 - 2x^4 + 4x^3 + 0x^2 + 0x + 3 = 0,$$

que contém os inversos multiplicativos das raízes de $P_6(x) = 0$, cuja sequência de fatores q_i de $P_6^{**}(x)$ é:

$$\left\{ \left| \frac{0}{-6} \right|^{\frac{1}{1}}, \left| \frac{-2}{-6} \right|^{\frac{1}{2}}, \left| \frac{4}{-6} \right|^{\frac{1}{3}}, \left| \frac{0}{-6} \right|^{\frac{1}{4}}, \left| \frac{0}{-6} \right|^{\frac{1}{5}}, \left| \frac{3}{-6} \right|^{\frac{1}{6}} \right\} = \{0, 0.577350, 0.873580, 0, 0, 0.890899\}$$

Tomando $max_1 = 0.890899$ e $max_2 = 0.873580$, temos

$$r_2 = max_1 + max_2 = 1.764479$$

então a cota mínima interna dada pela eq. (23a) é:

$$r_{min} = 1/r_2 = 0.5667395.$$

Logo, as seis raízes de $P_6(x) = 0$ também estão no intervalo $0.5667395 \leq \alpha \leq 2.223104$.

Comparando as 3 cotas de $P_6(x) = 0$:

- a) $|\alpha| \in (0.6, 3.0)$ cota do módulo máximo – **Propriedade 2**;
- b) $|\alpha| \in [0.8945041, 1.373431]$ cota de Cauchy – **Propriedade 3**;
- c) $|\alpha| \in [0.5667395, 2.223104]$ cota de Kojima – **Propriedade 4**.

Como as três cotas garantem a inclusão de todas as raízes, então podemos tomar o menor intervalo possível, com o maior r_{min} e o menor r_{max} , ou seja, tomar o anel mais estreito. Entre os **Exemplos 3.15, 3.16 e 3.17**, os limites do menor intervalo foram dados pela cota de Cauchy.

Dispondo das cotas máxima e mínima, podemos fazer uma varredura de possíveis raízes reais subdividindo este intervalo em subintervalos e testando cada um com o teorema de Bolzano, no lado negativo entre $[-r_{max}, -r_{min}]$ e no lado positivo entre $[+r_{min}, +r_{max}]$, verificando as trocas de sinais da função. Essa varredura de possíveis raízes reais com o teorema de Bolzano só é válida para equações polinomiais de coeficientes Reais.

Por último, podemos completar a estimativa das n raízes com valores iniciais de raízes Complexas, distribuídos “randomicamente” dentro do anel de raio $r \in [r_{min}, r_{max}]$, conforme o algoritmo estabelecido no arquivo **fLocaliza.m** que apresentamos no **Caderno de Algoritmos**.

Para as raízes de $P_6(x) = 3x^6 + 4x^3 - 2x^2 - 6 = 0$, localizamos dois valores iniciais Reais, conforme o teorema de Bolzano, enquanto os 4 seguintes são estabelecidos como Complexos, para abranger possíveis raízes Reais ou Complexas.

Um possível resultado para o vetor de raízes iniciais x_i gerado pela função $fLocaliza(n, a)$ seria:

$$x_{i_1} = -1.365449094487721 + 0.0000000000000000i$$

$$x_{i_2} = +1.030200914835262 + 0.0000000000000000i$$

$$x_{i_3} = -0.937643005591872 + 0.852667185210998i$$

$$x_{i_4} = -0.709960325540485 - 0.887984203422577i$$

$$x_{i_5} = 0.846274902134810 + 0.716025620017412i$$

$$x_{i_6} = 0.741095894499941 + 0.654794134744737i$$

Quando não temos certeza da existência de raízes Reais, devemos sempre adotar valores iniciais Complexos, que podem convergir para raízes Complexas ou Reais, conforme o caso. Entretanto, um valor inicial real jamais convergirá para um Complexo pelo método de Newton.

Para melhorar a localização de raízes, vamos estudar a natureza das raízes de equações polinomiais, se são Reais e/ou Complexas, usando algumas propriedades específicas.

3.3.2 Natureza de raízes de equações polinomiais

Nesta seção, vamos apresentar algumas propriedades que qualificam as raízes quanto à sua natureza, embora as cotas que apresentamos anteriormente já permitam uma boa localização de raízes iniciais.

Propriedade 5: as raízes Complexas de equações polinomiais de coeficientes Reais sempre ocorrem aos pares conjugados, isto é, se $\alpha = a + b * i$ for raiz, $\alpha = a - b * i$ também será raiz.

Exemplo 3.18: determine as raízes de $x^2 - x + 2 = 0$.

Solução:

Pela solução de equações de 2º grau, temos: $\alpha = \frac{1 \pm \sqrt{1-8}}{2} \Rightarrow \alpha = \frac{1 \pm \sqrt{-7}}{2}$
 $\alpha_1 = 0.5 + \sqrt{7/4} i$ e $\alpha_2 = 0.5 - \sqrt{7/4} i$

Consequência:

Toda $P_n(x) = 0$ de coeficientes reais com grau ímpar possui ao menos uma raiz Real.

Propriedade 6: em $P_n(x) = 0$, com $a_{n+1} \neq 0$, se n for **par** e $a_1 * a_{n+1} < 0 \Rightarrow$ existirão ao menos duas raízes Reais, uma positiva e outra negativa.

Por exemplo, em $1x^{12} - 5x^2 + 3x - 2 = 0$, temos que $a_1 * a_{n+1} = (1) * (-2) < 0$, logo possui pelo menos uma raiz Real positiva e uma negativa.

Propriedade 7: em $P_n(x) = 0$, com $a_{n+1} \neq 0$, se existir algum $k = 2, 3, \dots, n$, tal que $a_k^2 \leq a_{k+1} * a_{k-1} \Rightarrow$ existem raízes Complexas.

Essa propriedade é uma condição suficiente, mas não necessária, ou seja, se não for satisfeita, nada poderemos afirmar sobre a existência de raízes Complexas.

Propriedade 8: regra dos sinais de Descartes para polinômios com **coeficientes Reais**.

Segundo essa regra, em polinômios com os termos colocados em ordem crescente ou decrescente de grau, o número de zeros Reais positivos é igual ao número de permutações de sinal dos coeficientes não nulos de $P_n(x)$ ou menor por uma diferença par. O número de zeros Reais negativos do polinômio é igual ao número de permutações de sinais dos coeficientes de $P_n(-x)$ ou menor também por uma diferença par.

Exemplo 3.19: determine o quadro de possibilidades das raízes de $x^5 + x^4 - x^3 - x^2 = 0$.

Solução:

Podemos inicialmente fatorar os binômios com as duas **raízes nulas** ou aplicar diretamente a regra de Descartes, considerando as raízes nulas no quadro de possibilidades.

$$P_5(x) = +x^5 + \underbrace{x^4 - x^3}_{1 \text{ troca}} - x^2 = (x-0)^2 (+x^3 + \underbrace{x^2 - x^1}_{1 \text{ troca}} - x^0)$$

Então, ocorre “uma” mudança de sinal entre o segundo e o terceiro coeficientes. Portanto, $P_5(x)$ possui apenas uma raiz Real positiva. Para obter as possibilidades do número de raízes Reais negativas, geramos um polinômio auxiliar, $P_5^*(x) = P_5(-x)$, cujas raízes positivas de $P_5^*(x)$ correspondem às raízes negativas do polinômio original $P_5(x)$:

$$P_5^*(x) = P_5(-x) = (-x)^5 + (-x)^4 - (-x)^3 - (-x)^2 = \underbrace{-x^5 + x^4}_{1 \text{ troca}} + \underbrace{x^3 - x^2}_{1 \text{ troca}} = 0$$

esse polinômio auxiliar $P_5^*(x)$ tem “duas” permutações de sinal (até duas raízes positivas), logo o polinômio original $P_5(x)$ possui até duas raízes negativas, mas se tiver menos, será por uma diferença “par”, podendo não ter raízes negativas, conforme a Tabela 3.14:

Tabela 3.14 – Número de possibilidades de raízes Reais e Complexas

Positivas	Negativas	Nulas	Complexas (complemento para n)
1	2	2	0
1	0	2	2

Fonte: Elaboração própria.

Para confirmar o resultado, observe a fatoração em binômios deste polinômio do **Exemplo 3.19**:

$$P_5(x) = +x^5 + x^4 - x^3 - x^2 = (x - (-1))^2 (x - (+1))(x - 0)^2 = 0$$

Então, temos três raízes Reais distintas: -1 (aparece duas vezes, multiplicidade $M = 2$), $+1$ (aparece uma vez) e 0 (aparece duas vezes).

Sobre as propriedades que qualificam as raízes quanto à sua natureza, observamos que:

- a) podem ser incorporadas em um algoritmo de localização de raízes para definir com mais precisão os valores iniciais das raízes de $P_n(x) = 0$; e
- b) sempre que não forem localizadas raízes Reais em número suficiente, devemos usar valores iniciais Complexos, que são mais abrangentes, pois podem convergir para raízes Reais ou Complexas.

A seguir, vamos apresentar formas eficientes de calcular o valor numérico de uma função polinomial e de sua(s) derivada(s), para depois aplicar o método de Newton.

3.3.3 Cálculo eficiente de valores das funções polinomiais e suas derivadas

Para obter valores numéricos de $P_n(x)$ e de suas derivadas, vamos mostrar qual é a forma mais eficiente de representar $P_n(x)$.

Se expressarmos $P_n(x)$ na forma padrão, considerando que as potências inteiras de x são calculadas por produtos e não por potenciações, reaproveitando produtos efetuados, $x^{i+1} = \underbrace{x * x^i}_{1 \text{ multiplicação}}$, teremos que:

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + \dots + a_n x + a_{n+1} \Rightarrow$$

$$P_n(x) = a_{n+1} + a_n * x + a_{n-1} * x * x + \dots + a_2 * x * x^{n-2} + a_1 * x * x^{n-1}$$

Logo, $P_n(x = u)$ necessitará de $\begin{cases} n & \text{adições e} \\ 2n-1 & \text{multiplicações} \end{cases}$

Alternativamente, expressando $P_n(x)$ na forma linear de Horner, temos:

$$P_n(x) = \left(\dots \left((a_1 * x + a_2) * x + a_3 \right) * x + \dots + a_n \right) * x + a_{n+1}$$

Então, $P_n(x = u)$ necessitará apenas de $\begin{cases} n \text{ adições} & \text{e} \\ n \text{ multiplicações} \end{cases}$

No **Caderno de Algoritmos**, apresentamos um algoritmo para calcular o valor numérico de um polinômio $P_n(x)$, em $x = u$, por Horner, no arquivo **fPnH1.m**.

Neste livro, vamos calcular $P_n(x)$ por meio de uma forma equivalente à de Horner, que é a divisão sintética de polinômios, ou regra de **Briot-Ruffini**, de acordo com a **Propriedade 9** a seguir.

Propriedade 9: o valor numérico de $P_n(x)$ em $x = u$ é o resto R_1 da primeira divisão de $P_n(x)$ por $(x - u)$.

Pelo algoritmo da divisão de polinômios, temos que

$$P_n(x) = (x - u)P_{n-1}(x) + R_1 \quad (24)$$

Então, fazendo $x = u$,

$$P_n(x = u) = (u - u)P_{n-1}(u) + R_1 = R_1 \rightarrow \boxed{P_n(x = u) = R_1}$$

Considerando o polinômio quociente dessa primeira divisão como $P_{n-1}(x) = b_1x^{n-1} + b_2x^{n-2} + \dots + b_n$, em que os coeficientes b_i podem ser obtidos da eq. (24), pela igualdade termo a termo, temos:

$$b_1 = a_1$$

$$b_i = a_i + u * b_{i-1} \quad \forall i = 2 \text{ até } n + 1$$

$$\text{em que resto } R_1 = b_{n+1}.$$

Então, se usarmos a divisão sintética de polinômios, o cálculo de $P_n(x = u) = R_1$ também necessitará apenas de n adições e n multiplicações.

Propriedade 10: se α é um zero de $P_n(x)$, então $P_n(x)$ será divisível por $(x - \alpha)$ e por consequência $R_1 = 0$.

Propriedade 11: o valor numérico de $P'_n(x)$ em $x = u$ é o resto R_2 da segunda divisão sucessiva de $P_n(x)$ por $(x - u)$.

Tomando a primeira divisão, temos:

$$P_n(x) = (x - u)P_{n-1}(x) + R_1$$

Efetuada uma segunda divisão sintética, via divisão de $P_{n-1}(x)$ por $(x - u)$, temos:

$$P_{n-1}(x) = (x - u)P_{n-2}(x) + R_2$$

Substituindo $P_{n-1}(x)$ em $P_n(x)$, temos:

$$P_n(x) = (x - u)((x - u)P_{n-2}(x) + R_2) + R_1$$

$$P_n(x) = (x - u)^2 P_{n-2}(x) + (x - u)R_2 + R_1$$

Derivando $P_n(x)$, reescrito na forma anterior, temos:

$$P'_n(x) = 2(x - u)P_{n-2}(x) + (x - u)^2 P'_{n-2}(x) + R_2 + 0$$

Então, fazendo $x = u$,

$$P'_n(x = u) = 2(u - u)P_{n-2}(u) + (u - u)^2 P'_{n-2}(u) + R_2 = R_2 \rightarrow P'_n(x = u) = R_2$$

A **Propriedade 11** pode ser generalizada para o cálculo dos valores de todas as n derivadas de ordem superior de uma função $P_n(x)$, conforme a **Propriedade 12**.

Propriedade 12: o valor da derivada k -ésima de $P_n(x)$ em $x = u$ é dado por $\frac{d^k P_n}{dx^k}(u) = P_n^{(k)}(u) = k! R_{k+1}$, em que R_{k+1} é o resto da $(k + 1)$ -ésima divisão sucessiva de $P_n(x)$ por $(x - u)$.

Exemplo 3.20: obtenha $P_4(2)$, $P'_4(2)$, $P''_4(2)$, $P'''_4(2)$ e $P_n^{(4)}(2)$ em $P_4(x) = 2x^4 + 3x - 2$ aplicando a **Propriedade 12**.

Solução:**Primeira divisão:** de $P_4(x) = 2x^4 + 3x - 2$ por $(x - 2)$ $a_1 = 2, a_2 = 0, a_3 = 0, a_4 = 3, a_5 = -2$ (coeficientes do polinômio original):

$$b_1 = 2$$

$$b_2 = 0 + 2 * 2 = 4$$

$$b_3 = 0 + 4 * 2 = 8$$

$$b_4 = 3 + 8 * 2 = 19$$

$$b_5 = -2 + 19 * 2 = 36 \Rightarrow R_1 = 36$$

$$P_4(x=2) = P_4^{(0)}(x=2) = 0! * R_{0+1} = 1 * R_1 = 1 * 36 = 36$$

(também podemos obter o valor do polinômio pela **Propriedade 9**)**Segunda divisão:** de $P_3(x) = 2x^3 + 4x^2 + 8x + 19$ por $(x - 2)$ $a_1 = 2, a_2 = 4, a_3 = 8, a_4 = 19$ (esses coeficientes a_i foram atualizados pelos coeficientes b_i do quociente anterior):

$$b_1 = 2$$

$$b_2 = 4 + 2 * 2 = 8$$

$$b_3 = 8 + 2 * 8 = 24$$

$$b_4 = 19 + 24 * 2 = 67 \Rightarrow R_2 = 67$$

$$P_4'(x=2) = P_4^{(1)}(x=2) = 1! * R_{1+1} = 1! * R_2 = 1 * 67 = 67$$

Terceira divisão: de $P_2(x) = 2x^2 + 8x + 24$ por $(x - 2)$ $a_1 = 2, a_2 = 8, a_3 = 24$:

$$b_1 = 2$$

$$b_2 = 8 + 2 * 2 = 12$$

$$b_3 = 24 + 12 * 2 = 48 \Rightarrow R_3 = 48$$

$$P_4''(x=2) = P_4^{(2)}(x=2) = 2! * R_{2+1} = 2! * R_3 = 2 * 48 = 96$$

Quarta divisão: de $P_1(x) = 2x + 12$ por $(x - 2)$ $a_1 = 2, a_2 = 12$:

$$b_1 = 2$$

$$b_2 = 12 + 2 * 2 = 16 \Rightarrow R_4 = 16$$

$$P_4'''(x=2) P_4^{(3)}(x=2) = 3! * R_{3+1} = 3! * R_4 = 3 * 2 * 1 * 16 = 96$$

Quinta divisão: do polinômio de grau $n = 0$, $P_0(x) = 2$ por $(x - 2)$

$$a_1 = 2.$$

Como não temos como fazer a divisão de um polinômio de grau $n = 0$ por um binômio $(x - 2)$ de grau $n = 1$, podemos tomar o quociente dessa quinta divisão como 0 (zero), e o resto como o próprio $P_0(x) = 2$, pois $P_0(x) = (x - 2) * 0 + 2$:

$$b_1 = 2 \Rightarrow R_5 = 2$$

$$P_4^{(4)}(x=2) = P_4^{(4)}(x=2) = 4! * R_{4+1} = 4! * R_5 = 4 * 3 * 2 * 1 * (2) = 48$$

Podemos, ainda, validar a **Propriedade 12** comparando os resultados com as derivadas analíticas $P_4'(x) = 8x^3 + 3$, $P_4''(x) = 24x^2$, $P_4'''(x) = 48x$ e $P_4^{(4)}(x) = 48$.

Observamos que, dispondo das formas específicas de determinação de valores iniciais, do valor numérico do polinômio e de suas derivadas, podemos aplicar métodos de refinamento para as raízes de $P_n(x) = 0$. Aqui vamos aplicar o método de Newton, como mostraremos a seguir.

3.3.4 Método de Newton para $P_n(x) = 0$

A partir de um valor inicial x_0 , localizado previamente como apresentado nas seções 3.3.1 e 3.3.2, calculamos os valores de $P_n(x_0)$ e de $P_n'(x_0)$ na primeira iteração $k = 0$ pelas **Propriedades 9 e 11**:

$$P_n(x_0) = R_1$$

$$P_n'(x_0) = R_2$$

Depois, calculamos o incremento de x dado pela eq. (11a) do *método de Newton*, conforme segue:



Para polinomiais, o método de Newton também é chamado de Birge-Vieta.

$$\Delta x = -\frac{P_n(x_k)}{P_n'(x_k)} = -\frac{R_1}{R_2} \quad (25)$$

e calculamos o valor novo $x_{k+1} = x_k + \Delta x$ dado pela eq. (11b).

Exemplo 3.21: calcule a raiz positiva de $2x^3 - x - 2 = 0$, por Newton, a partir de $x_0 = 1$, com 16 dígitos significativos exatos.

Solução:

$$P_3(x) = 2x^3 + 0x^2 - 1x - 2x^0 \quad (a_1 = +2, a_2 = 0, a_3 = -1, a_4 = -2)$$

Usando a divisão sintética de Briot-Ruffini para determinar os valores de $P_3(x)$ e $P_3'(x)$, temos:

Primeira iteração: a partir de $x_0 = 1$.

Primeira divisão: $P_3(x) = 2x^3 + 0x^2 - 1x - 2$ por $(x - x_0)$.

$$b_1 = a_1 = +2$$

$$b_2 = a_2 + b_1 * x_0 = 0 + 2 * 1 = 2$$

$$b_3 = a_3 + b_2 * x_0 = -1 + 2 * 1 = 1$$

$$b_4 = a_4 + b_3 * x_0 = -2 + 1 * 1 = -1 = R_1 = P_3(x_0 = 1)$$

Segunda divisão: $P_2(x) = 2x^2 + 2x + 1$ por $(x - x_0)$.

$$c_1 = b_1 = 2$$

$$c_2 = b_2 + c_1 * x_0 = 2 + 2 * 1 = 4$$

$$c_3 = b_3 + c_2 * x_0 = 1 + 1 * 4 = 5 = R_2 = P_3'(x_0 = 1)$$

Logo, $R_1 = -1$ e $R_2 = 5$:

$$\Delta x = -\frac{R_1}{R_2} = -\frac{(-1)}{5} = 0.2 \text{ e}$$

$$x_1 = x_0 + \Delta x = 1 + 0.2 = 1.2$$

Segunda iteração: a partir de $x_1 = 1.2$.

Primeira divisão: $P_3(x) = 2x^3 + 0x^2 - 1x - 2$ por $(x - x_1)$:

$$P_3(1.2) = R_1 = 0.256$$

Segunda divisão:

$$P_3'(1.2) = R_2 = 7.64$$

$$\Delta x = -0.256/7.64 = -0.0335078534031414 \text{ e}$$

$$x_2 = 1.2 + \Delta x = 1.16649214659686$$

Terceira iteração: a partir de $x_2 = 1.16649214659686$.

Primeira divisão: $P_3(x) = 2x^3 + 0x^2 - 1x - 2$ por $(x - x_2)$:

$$P_3(1.16649214659686) = R_1 = 0.00800874528246043$$

Segunda divisão:

$$P_3'(1.16649214659686) = R_2 = 7.16422356843290$$

$$\Delta x = -\frac{0.00800874528246043}{7.16422356843290} = -0.00111788042430008 \text{ e}$$

$$x_3 = 1.16649214659686 + \Delta x = 1.16537426617256$$

Até a terceira iteração, temos $\alpha_1 \cong 1.16537426617256$, com critério de parada $|\Delta x| \cong O(10^{-3})$.

Esse processo iterativo deve prosseguir até que algum critério de parada seja satisfeito quando os 16 dígitos significativos estiverem convergidos, o

que ocorre quando o critério de parada $|\Delta x| < 10^{-15}$ é atingido, depois de 6 iterações. Então, a primeira raiz convergida é $\alpha_1 \cong 1.165373043062415$.

Para determinar a segunda e a terceira raízes, e evitar obter novamente a primeira raiz α_1 , não basta partir de uma estimativa inicial diferente, precisamos efetuar *reduções ou deflações*⁹ do grau da seguinte forma:

- Tomamos $P_n(x)$ e o dividimos por $(x - \alpha_1)$, ou seja, obtemos $P_{n-1}(x)$; nesse caso, com resto R_1 numericamente nulo (residual), pois nas divisões exatas os restos seriam exatamente nulos. Esse polinômio de grau reduzido $P_{n-1}(x)$ não contém mais o zero α_1 .
- Reaplicamos uma nova estimativa inicial para esse novo zero do polinômio de grau reduzido com o mesmo método de Newton.
- Calculado o novo zero, retornamos ao item (a) até completar a redução de grau desejada, que pode ser até graus $n = 2, 1$ ou 0 , pois esses polinômios têm raízes de solução algébrica simples.



Se não fizermos as reduções do grau para separar as raízes já determinadas, fatorando o polinômio, o(s) novo(s) valor(es) inicial(is) poderão convergir para raiz(es) já obtida(s).

Exemplo 3.22: calcule a segunda e a terceira raiz de $2x^3 - x - 2 = 0$, sabendo que a primeira raiz é $\alpha_1 \cong 1.165373043062415$, com 16 dígitos significativos convergidos.

Solução:

Inicialmente, devemos reduzir o grau do polinômio $P_3(x)$, extraindo a primeira raiz $\alpha_1 \cong 1.165373043062415$ já encontrada, fazendo $P_3(x) / (x - \alpha_1)$, gerando um polinômio quociente de grau $n = 2$:

$$P_2(x) = 2x^2 + 2.33074608612483x + 1.71618865899311$$

com resto residual $R_1 = 4.44089209850063e - 16$.

Logo, o polinômio original $P_3(x)$ fica fatorado da seguinte forma:

$$P_3(x) = (2x^2 + 2.33074608612483x + 1.71618865899311) * (x - \alpha_1) + 4.44089209850063e - 16$$

Observe que essa divisão pode ser considerada numericamente exata, uma vez que o seu resto é apenas residual e pode ser descartado, então $P_3(x)$ fatorado se torna:

$$P_3(x) = (2x^2 + 2.33074608612483x + 1.71618865899311) * (x - \alpha_1)$$

Logo, as duas raízes restantes de $P_3(x) = 0$ estão em:

$$P_2(x) = (2x^2 + 2.33074608612483x + 1.71618865899311) = 0$$

Esse polinômio de 2º grau resultante pode ser resolvido diretamente, determinando a segunda e a terceira raiz.

Podemos também reaplicar o método de Newton a esse polinômio de 2º grau para determinar todas as raízes pelo mesmo algoritmo.

Logo, as três raízes convergidas com 16 dígitos significativos da variável *double* são:

$$\alpha_1 = +1.165373043062415 + 0.0000000000000000i$$

$$\alpha_2 = -0.582686521531207 - 0.720118564628364i$$

$$\alpha_3 = -0.582686521531207 + 0.720118564628364i$$

E o polinômio final fatorado é $P_3(x) = a_1(x - \alpha_1)(x - \alpha_2)(x - \alpha_3)$ (com $a_1 = 2$).

Na próxima seção, vamos propor um algoritmo simples, robusto e eficiente para determinar raízes múltiplas (repetidas) de equações polinomiais, de modo que possa determinar simultaneamente as suas multiplicidades.

3.3.4.1 Determinação de raízes de equações polinomiais com multiplicidade

Vamos primeiramente estabelecer a **Propriedade 13**, que determina a multiplicidade M de uma raiz α conhecida, e comparar com o número de divisões exatas.

Propriedade 13: uma raiz α de $P_n(x) = 0$ possui multiplicidade M se, e somente se, $\frac{d^i P_n}{dx^i}(\alpha) = 0, \forall i = 0, 1, \dots, M-1$ e $\frac{d^M P_n}{dx^M}(\alpha) \neq 0$.

No **Exemplo 3.23**, vamos mostrar a equivalência da **Propriedade 13** para determinar a multiplicidade M com o número M de divisões sucessivas exatas (com restos nulos).

Exemplo 3.23: determine a multiplicidade M da raiz $\alpha = 2$ de $x^4 - 5x^3 + 6x^2 + 4x - 8 = 0$.

Solução:

Vamos primeiro calcular $P_4(x = 2)$ e suas derivadas analiticamente:

$$P_4(x) = x^4 - 5x^3 + 6x^2 + 4x - 8 \quad \Rightarrow P_4(2) = 0$$

$$P_4'(x) = 4x^3 - 15x^2 + 12x + 4 \quad \Rightarrow P_4'(2) = 0$$

$$P_4''(x) = 12x^2 - 30x + 12 \quad \Rightarrow P_4''(2) = 0$$

$$P_4'''(x) = 24x - 30 \quad \Rightarrow P_4'''(2) = 18$$

Pela **Propriedade 13**, temos que a multiplicidade de $\alpha = 2$ é $M = 3$, pois gera até derivada 2ª nula, e a derivada 3ª é não nula. Se aplicarmos as divisões sucessivas de Briot-Ruffini, teremos 3 restos nulos e $R_4 = 3$, logo a multiplicidade de $\alpha = 2$ também é $M = 3$, pois tem três divisões exatas.

A seguir, vamos definir um algoritmo sequencial para tratamento de raízes múltiplas desenvolvido a partir das quatro etapas a seguir, cada uma apresentando um exemplo com parte da metodologia aplicada.

Primeira etapa: vamos tentar determinar uma raiz múltipla como se fosse simples, de multiplicidade $M = 1$, usando a eq. (25) do método de Newton tradicional, em um polinômio exemplo cuja raiz é de multiplicidade $M = 3$.

Exemplo 3.24: determine as três raízes de $P_3(x) = x^3 - 3x^2 + 3x - 1 = 0$ com 16 dígitos exatos, considerando multiplicidade $M = 1$. Observe que a sua única raiz é $\alpha_1 = 1.0$ e tem multiplicidade $M = 3$, pois $P_3(x) = (x - 1)^3$.

Solução:

Pelos cálculos de cotas-limite, temos que todas as raízes iniciais estão entre $R_{\min} = 0.259921049896187$ e $R_{\max} = 3.84732210184363$, e que:

- uma raiz inicial Real, localizada por varredura, pode ser: $x_0 = 1.1$;
- duas possíveis raízes iniciais Complexas, randômicas, localizadas dentro do limite das cotas, podem ser:

$$+1.36284988456718 - 1.59235983373818i$$

$$+0.91142849471678 + 1.75484834399166i$$

Aplicando o método de Newton através das eqs. (25) e (11b), a partir de $x_0 = 1.1$, teremos o seguinte processo iterativo (em *double*):

Tabela 3.15 – Resultados do Exemplo 3.24

k	$P_3(x_k) = R(1)$	$P_3'(x_k) = R(2)$	$\Delta x = -\frac{R(1)}{R(2)}$	$x_{k+1} = x_k + \Delta x$
0	9.999999999998900e-04	3.000000000000020e-02	-3.33333333332940e-02	1.066666666666670
1	2.962962962960170e-04	1.333333333333540e-02	-2.22222222219780e-02	1.044444444444470
2	8.779149519977380e-05	5.925925925932770e-03	-1.481481481494470e-02	1.029629629629530
3	2.601229487364520e-05	2.633744855948810e-03	-9.876543209905690e-03	1.019753086419620
4	7.707346629359790e-06	1.170553269302930e-03	-6.584362140092560e-03	1.013168724279530
5	2.283658260271440e-06	5.202458974511440e-04	-4.389574759666210e-03	1.008779149519860
⋮				
21	7.993605777301130e-15	1.198849464145720e-09	-6.667731034102110e-06	1.000013322676890
22	2.220446049250310e-15	5.324816143570390e-10	-4.169995713244410e-06	1.000009152681170
23	0.000000000000000e+00	2.513149688354590e-10	0.000000000000000e+00	1.000009152681170

Fonte: Elaboração própria.

Em $k = 23$, podemos observar que o critério de parada composto $|P_3(x)| + |\Delta x|$ é zerado, mas a raiz aproximada $x_{23} = 1.000009152681170$ ainda não atingiu seu valor exato. Sabemos que a raiz exata é 1.0, mas a raiz convergida x_{23} tem apenas 6 dígitos significativos exatos e o processo iterativo estagnou, pois Δx já é nulo e não vai mais alterar x nos próximos passos iterativos. Então, a primeira raiz atingida é $\alpha_1 = 1.000009152681170$, considerando multiplicidade $M = 1$.

Fazendo a redução de grau com essa primeira raiz aproximada, temos:

$$n = 2 \Rightarrow P_2(x) = 1.0x^2 - 1.99999084731883x + 0.999990847402598$$

E as demais raízes são:

$$\alpha_2 = 0.999995424 + 0.000007926i$$

$$\alpha_3 = 0.999995424 - 0.000007926i$$

Assim, as três raízes convergidas são diferentes entre si e com apenas 6 dígitos significativos exatos, ou seja, deveríamos ter obtido três raízes $\alpha_1 = 1.0$ com 16 dígitos exatos, ou seja, multiplicidade $M = 3$.

Segunda etapa: vamos agora avaliar os efeitos de uma raiz múltipla sobre os restos das divisões sucessivas de $P_n(x) = 0$ por $(x - \alpha)$.

Exemplo 3.25: determine a multiplicidade M da raiz $\alpha = 1.0$ de $P_3(x) = x^3 - 3x^2 + 3x - 1 = 0$.

Solução:

Pela divisão sintética de Briot-Ruffini, temos:

	α	a_1	a_2	a_3	a_4	
1ª Divisão	1	1	-3	3	-1	
			1	-2	1	
2ª Divisão	1	1	-2	1	0	$\Rightarrow R_1$
			1	-1		
3ª Divisão	1	1	-1	0		$\Rightarrow R_2$
			1			
4ª Divisão	1	1	0			$\Rightarrow R_3$

$$\boxed{1} \Rightarrow R_4 \text{ (não podemos efetuar a divisão de 1 por } (x - 1) \text{)}$$

Terceira etapa: vamos assumir que a multiplicidade M é conhecida antes de conhecer a raiz correspondente.

O método de Newton para raízes não repetidas, ou seja multiplicidade $M = 1$, dado pela eq. (25), prevê que:

$$\Delta x = -\frac{P_n(x_k)}{P_n'(x_k)} = -\frac{R_1}{R_2}$$

Esse é o método de Newton clássico, mas observe que, no caso de aproximações de raízes com multiplicidade maior do que 1, R_1 e R_2 tendem a zero juntos, reduzindo a taxa de convergência (perde a taxa de convergência quadrática, como vimos nos **Exemplos 3.10** e **3.24**), podendo até gerar uma indeterminação $0/0$ no limite de convergência. Então,

a) Se $\Delta x = \lim_{x \rightarrow \alpha} \left(-\frac{P_n(x)}{P_n'(x)} \right) = 0$ (para raízes não repetidas)

$$\Delta x = -\frac{P_n(x_k)}{P_n'(x_k)} = -\frac{0!R_1}{1!R_2} \text{ para } M = 1$$

b) Se $\Delta x = \lim_{x \rightarrow \alpha} \left(-\frac{P_n(x)}{P_n'(x)} \right) = \frac{0}{0} = ?$ (indeterminação) à medida que x_k

tende à raiz α , então poderemos aplicar a regra de L'Hospital, como vimos na seção 3.2.2.4, uma vez que:

$$\lim_{x \rightarrow \alpha} \left(\frac{P_n(x)}{P_n'(x)} \right) = \lim_{x \rightarrow \alpha} \left(\frac{P_n'(x)}{P_n''(x)} \right), \text{ então}$$

$$\Delta x = -\frac{P_n'(x_k)}{P_n''(x_k)} = -\frac{1!R_2}{2!R_3} \text{ para } M = 2$$

c) Se $\Delta x = \lim_{x \rightarrow \alpha} \left(\frac{P_n'(x)}{P_n''(x)} \right) = \frac{0}{0} = ?$ à medida que x_k tende à raiz α , então deveremos reaplicar a regra de L'Hospital,

$$\lim_{x \rightarrow \alpha} \left(\frac{P_n'(x)}{P_n''(x)} \right) = \lim_{x \rightarrow \alpha} \left(\frac{P_n''(x)}{P_n'''(x)} \right), \text{ então}$$

$$\Delta x = -\frac{P_n''(x_k)}{P_n'''(x_k)} = -\frac{2!R_3}{3!R_4} \text{ para } M = 3$$

d) Para M genérico, temos:

$$\Delta x = -\frac{P_n^{(M-1)}(x_k)}{P_n^{(M)}(x_k)} = -\frac{(M-1)!R_M}{M!R_{M+1}} = -\frac{(M-1)!R_M}{M(M-1)!R_{M+1}}$$

$$\boxed{\Delta x = -\frac{R_M}{M * R_{M+1}}} \text{ para } M \text{ qualquer} \quad (26)$$

Com $x_{k+1} = x_k + \Delta x$ dado pela mesma eq. (11b).

Exemplo 3.26: determine a raiz de $P_3(x) = x^3 - 3x^2 + 3x - 1 = 0$ com 16 dígitos exatos, sabendo que a multiplicidade dessa raiz é $M = 3$.

Solução:

Vamos partir do mesmo valor inicial $x_0 = 1.1$, imprimir os 4 restos das divisões sucessivas de $P_3(x)$ por $(x - x_0)$, usar $M = 3$ e aplicar a eq. (26):

$$\Delta x = -\frac{R_3}{3 * R_{3+1}}$$

Tabela 3.16 – Resultados do Exemplo 3.26

k	x_k	R_1	R_2	R_3	R_4	M	Δx (eq.26)	x_{k+1}	$ P_3(x_k) + \Delta x $
0	1.1	0.0010000	0.0300000	0.3000000	1.0	3	-0.1	1.0000	0.101
1	1.0	0	0	0	1.0	3	0.0	1.0000	0.0
2	1.0								

Fonte: Elaboração própria.

Observe que obtemos a raiz exata $\alpha_1 = 1.0$ em duas iterações: na primeira, já calculamos a raiz exata; e, na segunda iteração, confirmamos essa raiz e sua multiplicidade $M_1 = 3$ gerando 3 restos nulos, mas para tal tivemos que usar um valor de multiplicidade previamente conhecido. Nesse exemplo de raiz única, se usarmos qualquer outro valor inicial da raiz, teremos o mesmo resultado convergido em 2 iterações. Perceba a dificuldade adicional de aplicar essa correção no método de Newton, dada pela eq. (26), que é a necessidade de determinar previamente o valor da multiplicidade M de uma raiz por meio de sua aproximação x_k .

Segundo o trabalho de Schröder (1992), a eq. (18) representa uma estimativa clássica do valor da multiplicidade \hat{M} que, adaptada com os restos dos polinômios representativos de f, f' e f'' , resulta em,

$$\hat{M} = \frac{(f'(x_k))^2}{(f'(x_k))^2 - f(x_k)f''(x_k)} = \frac{(R_2)^2}{(R_2)^2 - R_1*(2*R_3)} \quad (27)$$

Mas essa estimativa depende de vários fatores, como da proximidade da raiz x_k com a raiz exata α e do número de raízes múltiplas presentes no polinômio.

Para polinômios com única raiz múltipla, a eq. (27) gera a multiplicidade correta, independentemente do valor inicial da raiz, por exemplo:

- para $P_3(x) = (x - 1)^3$, em $x_0 = +1.1$, teremos $\hat{M} = 3.0000000000001910$; e
- para $P_3(x) = (x - 1)^3$, em $x_0 = -110$, teremos $\hat{M} = 3.0000000000000000$.

Para polinômios com duas ou mais raízes múltiplas, a estimativa dada pela eq. (27) é boa somente para valores de raízes nas proximidades da raiz exata, como:

- a) Para $P_8(x) = (x + 1)^2 (x - 0.9)^6$, em $x_0 = -1.001$, teremos $\hat{M} = 2.006315782479700$;
- b) Para $P_8(x) = (x + 1)^2 (x - 0.9)^6$, em $x_0 = +0.91$, teremos $\hat{M} = 6.024375198569969$; e
- c) Para $P_8(x) = (x + 1)^2 (x - 0.9)^6$, em $x_0 = +110$, teremos $\hat{M} = 7.999556745299021$ (\hat{M} inconsistente).

Então, para raízes Complexas ou relativamente afastadas da raiz exata, as estimativas de \hat{M} dadas pela eq. (27) falham e podem conduzir a resultados inconsistentes. Assim, precisamos de uma forma alternativa para estimar a multiplicidade \hat{M} de raízes genéricas.

Quarta etapa: vamos agora avaliar o comportamento dos restos com a aproximação de x_k com a raiz exata α .

Vamos propor neste livro uma alternativa prática para determinar a estimativa da multiplicidade \hat{M} de cada raiz aproximada x_k através de um contador do número \hat{M} de restos próximos de zero: $R_1, R_2, R_3, \dots, R_{\hat{M}}$, ou seja, do número de restos cujos valores em módulo fiquem abaixo de um limite mínimo estabelecido por experimentos numéricos, definido aqui pelo parâmetro R_{lim} no algoritmo proposto no **Caderno de Algoritmos**.

Quando x_k estiver convergido para a raiz α , teremos \hat{M} restos convergidos para valores “numericamente nulos (residuais)”. Então, por esse algoritmo, uma raiz aproximada x_k já é considerada previamente de multiplicidade \hat{M} , quando \hat{M} restos, ou quando a soma dos módulos desses restos, $R_1, R_2, R_3, \dots, R_{\hat{M}}$, estiver abaixo do resto limite R_{lim} .

Segundo uma sequência de experimentos numéricos, verificamos que esse limite R_{lim} depende basicamente da distância entre as raízes, mas que ainda não foram determinadas.

Então, considerando os coeficientes normalizados de modo que $a_1 = 1$, propomos que o limite R_{lim} seja o produto $p = R_{lim1} * R_{lim2}$, em que:

- a) $R_{lim1} = 10\%$ do valor mínimo das diferenças entre todos os coeficientes em módulo, normalizado pelo maior dos coeficientes em módulo. Se essa diferença mínima for nula, adotaremos $R_{lim1} = 0.1$; e
- b) R_{lim2} é o inverso da potência de 10 elevada ao número máximo de dígitos “fracionários” dos coeficientes (esse valor mede a influência direta da precisão das raízes).

Esse produto p deverá ser limitado a um valor mínimo para não gerar R_{lim} nulo, então vamos adotar um mínimo 10^{-8} para considerar restos nulos sempre que ficarem menores que 10^{-8} (equivalente à metade da precisão adotada, para algoritmos construídos com variáveis *double* de 16 dígitos significativos), resultando na equação:

$$R_{lim} = \max(p, 10^{-8}) \quad (28)$$

O parâmetro R_{lim} é dependente da distância entre as raízes, do grau do polinômio e da precisão das variáveis utilizadas para o processamento numérico.

Esse parâmetro R_{lim} captura bem raízes múltiplas espaçadas e ao mesmo tempo relativamente próximas entre si. Estas últimas precisam convergir até restos menores, por exemplo, da ordem de 10^{-8} , para serem consideradas numericamente nulas, conforme os exemplos a seguir.

Exemplo 3.27: determine as três raízes de $x^3 - 3x^2 + 3x - 1 = 0$ com 16 dígitos exatos e suas respectivas multiplicidades M (sabemos que a raiz é $\alpha = 1.0$ e a sua multiplicidade é $M = 3$).

Solução:

Tomamos $x_0 = 1.1$, conforme o **Exemplo 3.24**, calculamos o limite para os restos pela eq. (28), com:

- a) $R_{lim1} = 0.1$ (pois a mínima diferença entre os 4 diferentes coeficientes é zero); e

- b) $R_{lim2} = 1/10^0$ (pois o número máximo de dígitos “fracionários” dos coeficientes $\{1, -3, +3, -1\}$ é zero, uma vez que são coeficientes inteiros, sem frações decimais).

Aplicando a eq. (28), teremos o seguinte resto limite:

$$R_{lim} = \max(R_{lim1} * R_{lim1}, 10^{-8}) = \max(0.1 * 1, 10^{-8}) = 0.1$$

Aplicando o método de Newton modificado com a eq. (26) e estimando a multiplicidade \hat{M} a cada iteração pelo número \hat{M} de restos próximos de zero, temos:

Tabela 3.17 – Resultados do Exemplo 3.27

k	x_k	R_1	R_2	R_3	R_4	\hat{M}	Δx (eq.26)	x_{k+1}	$ P_3(x_k) + \Delta x $
0	1.100	0.0010000	0.0300000	0.3000000	1.0	2	-0.05	1.050	0.0510
1	1.050	1.2500e-04	7.5000e-03	1.5000e-01	1.0	2	-2.5e-02	1.025	2.5125e-02
2	1.025	1.5625e-05	1.8750e-03	7.5000e-02	1.0	3	-2.5e-02	1.000	2.5016e-02
3	1.000	0	0	0	1.0	3	0.0	1.000	0
4	1.000								

Fonte: Elaboração própria.

Em $k = 0$ e $k = 1$, R_1 e R_2 estão abaixo de $R_{lim} = 0.1$ e a sua soma também é menor do que $R_{lim} = 0.1$ (destacados em negrito), ou seja, 2 restos são considerados suficientemente pequenos, logo a multiplicidade estimada é $\hat{M} = 2$.

Em $k = 2$, R_1 , R_2 e R_3 estão abaixo de $R_{lim} = 0.1$ e a sua soma mantém-se menor do que $R_{lim} = 0.1$ (destacados em negrito), ou seja, 3 restos são considerados suficientemente pequenos, logo a multiplicidade estimada é $\hat{M} = 3$.

Em $k = 3$, quarta iteração, temos a confirmação da raiz convergida para 1.0 (exata) e da sua multiplicidade $M = 3$, pois 3 restos, nesse caso, são exatamente zero.

Então, os resultados desse algoritmo de Newton modificado para raízes múltiplas são exatos, $x_1 = 1.0$, e $M_1 = 3$, conforme o **Exemplo 3.27**, enquanto os resultados obtidos com multiplicidade $M = 1$ geram resultados com apenas 6 dígitos significativos exatos (análogos aos resultados da função `roots()` do Octave), conforme o **Exemplo 3.24**.

Exemplo 3.28: determine as raízes de

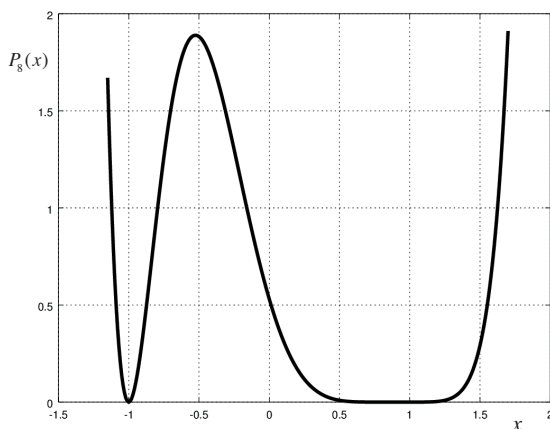
$$P_8(x) = 1x^8 - 3.4x^7 + 2.35x^6 + 4.32x^5 - 7.1685x^4 + 1.56006x^3 + 3.287061x^2 - 2.480058x + 0.531441 = 0$$

com 16 dígitos exatos e suas respectivas multiplicidades (polinômio fatorado equivalente: $P_8(x) = (x + 1)^2 (x - 0.9)^6 = 0$).

Solução:

Nenhuma raiz Real foi localizada pelo teorema de Bolzano, pois ambas as raízes são de multiplicidade par e $P_8(x)$ não produz trocas de sinais, conforme o Gráfico 3.14.

Gráfico 3.14 – Representação da função $P_8(x)$



Fonte: Elaboração própria.

As raízes iniciais complexas são geradas randomicamente pela função `fLocaliza(n, a)` apresentada no **Caderno de Algoritmos**.

O limite para os restos é determinado por:

- a) $R_{lim1} = 0.00157548999093255$, pois a mínima diferença entre todos os coeficientes ocorre entre $|-3.4|$ e $|+3.287061|$, que foi normalizada pelo maior coeficiente em módulo $|-7.1685|$; e
- b) $R_{lim2} = 1/10^6$, pois o número máximo de dígitos “fracionários” dos coeficientes de $P_8(x)$ é 6 (são coeficientes com no máximo 6 dígitos depois do ponto (vírgula)).

Aplicando a eq. (28), teremos o seguinte resto limite:

$$R_{lim} = \max(R_{lim1} * R_{lim2}, 10^{-8}) = \max(0.00157548999093255 * 10^{-6}, 10^{-8}) = 10^{-8}$$

Note que, neste exemplo, os restos só serão considerados suficientemente pequenos quando ficarem abaixo de $R_{lim} = 10^{-8}$, ou seja, quando a raiz aproximada estiver bem próxima da raiz convergida.

Assim, os resultados obtidos são duas raízes exatas com suas respectivas multiplicidades:

$$x_1 = +0.9000000000000000 + 0.0000000000000000i \text{ e } M_1 = 6$$

$$x_2 = -1.0000000000000000 + 0.0000000000000000i \text{ e } M_2 = 2$$

enquanto os resultados obtidos com multiplicidade $M = 1$ (análogo à função `roots()` do Octave) geram resultados com apenas 3 a 6 dígitos significativos exatos.

Para cada raiz de multiplicidade M , devem ser efetuadas M reduções de grau para então determinar as próximas raízes.

Nos exemplos testados, se a soma dos módulos dos restos, $R_1, R_2, R_3, \dots, R_{\tilde{M}}$, for menor do que R_{lim} , consideraremos que a multiplicidade estimada da raiz x_k será \tilde{M} . Desse modo, podemos acelerar o processo de convergência,

usando a eq. (26), por meio de sucessivas estimativas de \hat{M} , que devem ser confirmadas no final, na última iteração, quando todos os restos até R_M tornam-se valores residuais. Consequentemente, R_{M+1} será naturalmente diferente de zero, e a eq. (26) será aplicável até o limite da precisão digital das variáveis envolvidas, obtendo raízes com a precisão máxima possível para a variável utilizada, no caso *double*.

Quando as raízes estão próximas entre si, os restos para valores de x nessa região diminuem muito, como se as raízes próximas fossem uma única raiz múltipla, então precisamos de um valor R_{lim} menor do que um R_{lim} para raízes mais espaçadas, a fim de eliminar uma possível falsa multiplicidade. Nesses casos, temos de fazer o processo iterativo chegar mais próximo da raiz exata para então assumir que essa raiz aproximada tenha multiplicidade M .

São aspectos importantes sobre o comportamento dos restos:

- a) Em $P_3(x) = x^3 - 3x^2 + 3x - 1 = (x - 1)^3$, temos três raízes iguais a 1.0, $M = 3$ e 3 restos convergidos são nulos. No processo iterativo, os restos se reduzem gradativamente, mas sempre em ordem crescente $R_1 < R_2 < R_3$, até o quarto resto R_4 , que é sempre igual ao primeiro coeficiente a_1 .
- b) No exemplo testado por Rall (1966) para o polinômio, $P_5(x) = (x - 1 + \varepsilon)(x - 1)(x - 1 - \varepsilon)(x - 2)(x - 3)$ com $\varepsilon = 10^{-2}$ e 10^{-3} , temos três raízes próximas a 1.0 e todas com multiplicidade $M = 1$. Nesse caso, com $\varepsilon = 10^{-2}$ e para a primeira raiz encontrada, os restos convergem para R_1 da ordem de $O(10^{-15})$, R_2 na ordem de $O(10^{-3})$, R_3 na ordem de $O(10^{-2})$ e o quarto resto é da ordem da unidade. Esses valores de restos poderiam gerar uma falsa multiplicidade $M = 3$ se o R_{lim} estabelecido fosse, por exemplo, da ordem de $O(10^{-2})$, mas aplicamos o R_{lim} proposto pela eq. (28), que assumiu o valor mínimo 10^{-8} , ou seja, considerou apenas o R_1 como resto nulo e manteve a multiplicidade correta $M = 1$. Se fosse o caso de uma multiplicidade $M = 3$ verdadeira, os restos desse polinômio decresceriam até ficarem todos menores do que $R_{lim} = 10^{-8}$ e teríamos obtido $M = 3$.

c) Para outro polinômio com 5 raízes próximas como

$$P_{10}(x) = -x^{10} - 7x^9 + 20.95x^8 - 34.75x^7 + 34.5004x^6 - 20.5012x^5 \\ + 6.7512x^4 - 0.9504x^3 - 0x^2 - 0x + 0$$

($P_{10}(x) = (x-0)^3(x-0.8)(x-0.9)(x-1.0)^3(x-1.1)(x-1.2)$), pelo método de Newton modificado para raízes múltiplas são geradas as seguintes raízes:

$$x_1 = 0 \quad \Rightarrow M_1 = 3$$

$$x_2 = 0.8000000000143045 \quad \Rightarrow M_2 = 1$$

$$x_3 = 0.89999999993704603 \quad \Rightarrow M_3 = 1$$

$$x_4 = 1.0000000000113123 \quad \Rightarrow M_4 = 3$$

$$x_5 = 1.10000000003988399 \quad \Rightarrow M_5 = 1$$

$$x_6 = 1.2000000000078299 \quad \Rightarrow M_6 = 1$$

(resultados não exatos, mas com no mínimo 10 dígitos significativos exatos), enquanto pelo método de Newton tradicional teremos raízes múltiplas calculadas como raízes distintas e com apenas 4 dígitos significativos exatos:

$$1.199999999821068 + 0.000000000000000i \\ 1.100000002521077 + 0.000000000000000i \\ 1.000159100194941 + 0.000275675116168i \\ 1.000159100194941 - 0.000275675116168i \\ 0.999681795585784 + 0.000000000000000i \\ 0.900000001769856 + 0.000000000000000i \\ 0.799999999912334 + 0.000000000000000i \\ 0.000000000000000 + 0.000000000000000i \\ 0.000000000000000 + 0.000000000000000i \\ 0.000000000000000 + 0.000000000000000i$$

(resultados análogos aos obtidos pela função *roots()* do Octave ou pelo *Wolfram Alpha*).

- d) Para polinômios de grau elevado e multiplicidade também elevada, o algoritmo de Newton modificado para raízes múltiplas proposto neste livro também foi eficiente. Por exemplo, para a equação polinomial $P_{20}(x) = (x - 1)^{20} = 0$ expandida, temos:
- uma única raiz $x_1 = 1.0$ com multiplicidade $M_1 = 20$ (20 raízes iguais a x_1); e
 - com algoritmo de Newton original, sem a correção de multiplicidade ou com a função `roots()` do Octave, temos 20 raízes distintas e imprecisas como as duas exemplificadas a seguir:

$$1.322055777843924 + 0.055758394623173i$$

$$0.754913929640771 + 0.0000000000000000i$$

Por fim, alertamos que obtivemos os parâmetros-limite da eq. (28) por meio de diversos testes numéricos com diferentes polinômios, que devem ser novamente experimentados para especificidades de novas famílias de polinômios. Há também uma dependência natural das raízes em relação ao seu valor inicial, que foi minimizada por um algoritmo de localização precisa de raízes iniciais, conforme `fLocaliza(n, a)`.

Nos casos de raízes muito próximas entre si, é recomendada a utilização de técnicas como a determinação dos autovalores da matriz companheira usando o método QR, conforme Serre (2002).

Na próxima seção, vamos detalhar o processo de purificação das raízes determinadas nos polinômios de grau reduzido visando minimizar o efeito do acúmulo de erros de arredondamentos decorrentes dessas reduções de grau.

3.3.4.2 Purificação de raízes obtidas com redução de grau

Considere a equação polinomial:

$$P_4(x) = x^4 - 11.101x^3 + 11.1111x^2 - 1.0111x + 0.001 = 0$$

que tem as seguintes raízes exatas:

$$\alpha_1 = 0.001$$

$$\alpha_2 = 0.1$$

$$\alpha_3 = 1$$

$$\alpha_4 = 10$$

Considere que a quarta raiz aproximada pelo método de Newton seja $\alpha_4 = 10.000005$, com multiplicidade $M = 1$, e que tenha sido encontrada em $P_1(x) = 0$ pelas reduções sucessivas de grau:

$$\alpha_1 \text{ é obtida em } P_4(x) = 0$$

$$\alpha_2 \text{ é obtida em } P_3(x) = 0$$

$$\alpha_3 \text{ é obtida em } P_2(x) = 0$$

$$\alpha_4 \text{ é obtida em } P_1(x) = 0$$

Então, α_4 contém erros devido ao método de Newton e devido às reduções sucessivas de grau, que implicam arredondamentos nos coeficientes dos sucessivos quocientes. Isso acumula erros de arredondamento e pode gerar instabilidade numérica.

Uma solução para minimizar esse problema pode ser o chamado processo de **purificação** dos resultados parciais, aplicado já a partir da segunda raiz α_2 , porque é a primeira raiz a ser obtida em um polinômio que já sofreu a redução de grau e arredondamentos nos seus coeficientes.

Assim, para purificar α_4 , por exemplo, reaplicamos o método de Newton, como se fôssemos obter a primeira raiz, usando:

- a) o polinômio original, $P_4(x) = 0$, que é o único “puro”, sem erros de arredondamento nos seus coeficientes, pois não sofreu redução de grau; e
- b) o valor inicial como cada *raiz aproximada*⁹, obtida nos polinômios de grau reduzido.



Note que essas raízes aproximadas através do polinômio de grau reduzido já são próximas das raízes exatas, então podem ser usadas como valores iniciais no polinômio original diminuindo o risco de ocorrer convergência para uma raiz já obtida.

O resultado da purificação de $\alpha_4 = 10.000005$ em $P_4(x) = 0$ (original) é a quarta raiz purificada $\alpha_4 = 10.0000000000000002$, com precisão de 16 dígitos significativos.

Antes de continuar a sua leitura, volte ao **Caderno de Algoritmos** e confira o algoritmo de Newton completo para localização, determinação e purificação de todas as raízes de equações polinomiais, Reais ou Complexas, de qualquer multiplicidade M , no arquivo **Cap3NewtonPolinomios.m**.

Considere que esse algoritmo de Newton completo:

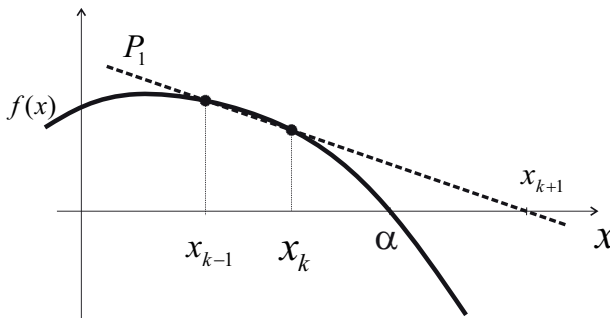
- a) apresenta uma alternativa para determinação de raízes múltiplas de equações polinomiais, espaçadas e/ou próximas entre si, convergindo até o limite da precisão digital disponível (no caso *double*);
- b) recupera a taxa quadrática de convergência otimizando o número total de operações aritméticas para obtenção de raízes múltiplas;
- c) tem princípios matemáticos simples; e
- d) utiliza parâmetros-limite que podem ser estendidos a outros tipos de polinômios.

3.3.5 Determinação de raízes de equações pelo método de Müller

O método de Müller para a determinação de raízes de equações é uma extensão do método da secante que busca minimizar alguns problemas presentes nesse método e no de Newton, como não encontrar diretamente raízes Complexas, a menos que seja dado um valor inicial Complexo e seja utilizada a aritmética complexa na sua aplicação.

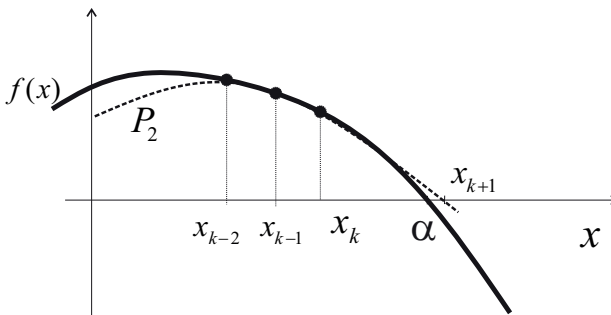
Para apresentar o método de Müller (BURDEN; FAIRES, 2011), vamos analisar os Gráficos 3.15 e 3.16, que comparam a resolução de uma equação $f(x) = 0$ pelos métodos da secante e de Müller.

Gráfico 3.15 – Método da secante (P_1 reta)



Fonte: Elaboração própria.

Gráfico 3.16 – Método de Müller (P_2 parábola)



Fonte: Elaboração própria.

No método da secante, calculamos o zero da **função aproximadora linear** que passa por dois pontos da curva de $f(x) = 0$, conforme o Gráfico 3.15. Já no método de Müller, calculamos o zero da **função aproximadora quadrática** que passa por três pontos da curva de $f(x) = 0$ a ser resolvida, conforme o Gráfico 3.16. Seria equivalente a estendermos a aproximação de 1ª ordem da série de Taylor para gerar o método de Newton, mas usando agora uma aproximação de 2ª ordem da série de Taylor, conforme propomos no **Exercício 3.3 do Caderno de Exercícios e Respostas** disponível no [link <http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>](http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/).

Supondo que três aproximações iniciais, x_{k-2} , x_{k-1} e x_k , são estimadas na região da raiz de $f(x) = 0$, a nova aproximação x_{k+1} seria obtida de um zero do polinômio quadrático $P_2(x)$ que passa por essas três aproximações iniciais de $f(x)$, logo

$$P_2(x) = a(x - x_k)^2 + b(x - x_k) + c \quad (29)$$

Aplicando as condições $P_2(x_{k-2}) = f(x_{k-2})$, $P_2(x_{k-1}) = f(x_{k-1})$ e $P_2(x_k) = f(x_k)$, obtemos o sistema de equações lineares:

$$\begin{cases} P_2(x_{k-2}) = a(x_{k-2} - x_k)^2 + b(x_{k-2} - x_k) + c = f(x_{k-2}) \\ P_2(x_{k-1}) = a(x_{k-1} - x_k)^2 + b(x_{k-1} - x_k) + c = f(x_{k-1}) \\ P_2(x_k) = a(x_k - x_k)^2 + b(x_k - x_k) + c = f(x_k) \end{cases} \quad (30)$$

que, resolvido diretamente, explicita os coeficientes a , b e c :

$$a = q * f(x_k) - q(1+q)f(x_{k-1}) + q^2 * f(x_{k-2}) \quad (31a)$$

$$b = (2q+1)f(x_k) - (1+q)^2 f(x_{k-1}) + q^2 * f(x_{k-2}) \quad (31b)$$

$$c = (1+q)f(x_k) \quad (31c)$$

onde

$$q = \frac{x_k - x_{k-1}}{x_{k-1} - x_{k-2}} \quad (32)$$

Agora, encontramos a nova aproximação x_{k+1} resolvendo a equação de 2º grau $P_2(x_{k+1}) = 0$:

$$x_{k+1} = x_k - (x_k - x_{k-1}) \left[\frac{2c}{b \pm \sqrt{b^2 - 4ac}} \right] \quad (33)$$

Note que, na eq. (33), temos duas possibilidades de obter x_{k+1} , a depender do sinal escolhido para o radicando, mas esse sinal deve ser escolhido de modo que o valor do módulo do denominador seja sempre o maior possível, isto é, o sinal para o radicando deve ser o mesmo sinal de b . Essa escolha gera um denominador maior em magnitude, evita possibilidades de denominador nulo, minimiza arredondamentos e gera um valor de x_{k+1} mais próximo de x_k , dando mais estabilidade ao método numérico, conforme vimos na seção de erros por perda de significação do Capítulo 1.

Assim, escolhemos a seguinte expressão para atualizar x_{k+1} ,

$$x_{k+1} = x_k - (x_k - x_{k-1}) \left[\frac{2c}{b + \text{sign}(b)\sqrt{b^2 - 4ac}} \right] \quad (34)$$

Então, aplicamos iterativamente a fórmula dada na eq. (34) até que um critério de parada seja satisfeito. Note que, se o radicando gerar números Complexos, podemos obter raízes Complexas de $f(x) = 0$ diretamente a partir de valores iniciais Reais, o que é impossível de ocorrer com o método de Newton ou da secante.

Exemplo 3.29: determine a raiz de $x * \ln(x) - 3.2 = 0$ pelo método de Müller com o máximo de dígitos significativos disponíveis.

Solução:

Escolhido o primeiro valor inicial $x_k = 2.5$ para $k = 0$ em $[2, 3]$ e gerado outros dois valores anteriores, x_{k-2} e x_{k-1} , nessa região de proximidade da raiz, temos:

$$x_k = 2.5$$

$$x_{k-1} = x_k * 0.99 = 2.475$$

$$x_{k-2} = x_k * 0.98 = 2.450$$

Tabela 3.18 – Resultados do Exemplo 3.29

k	x_{k-2}	x_{k-1}	x_k	x_{k+1}	$ x_{k+1} - x_k $
0	2.450	2.475	2.500	2.95286713407486	0.45557167
1	2.475	2.500	2.95286713407486	2.95416246266100	0.0013017
2	2.500	2.95286713407486	2.95416246266100	2.95416552326093	3.0606e-06
3	2.95286713407486	2.95416246266100	2.95416552326093	2.95416552327888	1.7952e-11
4	2.95416246266100	2.95416552326093	2.95416552327888	2.95416552327888	8.8818e-16
5	2.95416552326093	2.95416552327888	2.95416552327888	2.95416552327888	4.4409e-16

Fonte: Elaboração própria.

$$x_6 = 2.95416562327888$$

Observe que encontramos a raiz x_6 em 6 iterações com 16 dígitos significativos de precisão, assim como no método da secante, ao passo que com o método de Newton foram necessárias 5 iterações, para este mesmo exemplo.

Veremos agora um exemplo de polinômio com raízes Complexas.

Exemplo 3.30: determine uma raiz de $x^4 - 2x^3 + 6x^2 - 8x + 8 = 0$ com o máximo de dígitos significativos disponíveis pelo método de Müller (sabendo que as raízes são: $1 + i$, $1 - i$, $2i$ e $-2i$).

Solução:

Vamos escolher o primeiro valor inicial $x_k = 2.0$ para $k = 0$, dentro das cotas-limite, e gerar os outros dois valores, x_{k-2} e x_{k-1} , nessa região de proximidade da raiz. Desse modo, temos:

$$x_k = 2.0$$

$$x_{k-1} = x_k * 0.99 = 1.98$$

$$x_{k-2} = x_k * 0.98 = 1.96$$

Tabela 3.19 – Resultados do Exemplo 3.30

k	x_{k-2}	x_{k-1}	x_k	x_{k+1}	$ x_{k+1} - x_k $
0	1.96000 + 0.00000i	1.98000 + 0.00000i	2.00000 + 0.00000i	1.31997 + 0.66667i	0.95231
1	1.98000 + 0.00000i	2.00000 + 0.00000i	1.31997 + 0.66667i	1.13016 + 0.84954i	0.26357
2	2.00000 + 0.00000i	1.31997 + 0.66667i	1.13016 + 0.84954i	1.11082 + 1.04137i	0.19281
3	1.31997 + 0.66667i	1.13016 + 0.84954i	1.11082 + 1.04137i	0.99908 + 1.01479i	0.11485
⋮					
24	1.00000 + 1.00000i	1.00000 + 1.00000i	1.00000 + 1.00000i	1.00000 + 1.00000i	0.00000

Fonte: Elaboração própria.

$$x_{25} = 1.00000 + 1.00000i$$

Observe que na 1ª iteração, efetuada em $k = 0$, já geramos uma 1ª aproximação x_{k+1} Complexa para a raiz e encontramos a raiz convergida em 25 iterações com 16 dígitos significativos de precisão.

Podemos determinar as demais raízes pelo processo de redução de grau.

Exemplo 3.31: determine uma raiz de $x^3 - 3x^2 + 3x - 1 = 0$ com o máximo de dígitos significativos disponíveis pelo método de Müller (sabendo que a raiz é 1.0 e tem multiplicidade $M = 3$).

Solução:

Escolhemos um primeiro valor inicial $x_k = 2.0$ para $k = 0$, dentro das cotas-limite, e geramos os outros dois valores, x_{k-2} e x_{k-1} , nessa região de proximidade da raiz:

$$x_k = 2.0$$

$$x_{k-1} = x_k * 0.99 = 1.98$$

$$x_{k-2} = x_k * 0.98 = 1.96$$

Depois de 180 iterações, atingimos o critério de parada

$$|x_{k+1} - x_k| = 3.57470423575603e - 16$$

e a raiz:

$$x = 1.00000266769999 + (5.0057518865228e - 06)i$$

com precisão de apenas 6 dígitos significativos, como acontece no método de Newton clássico sem a devida correção da multiplicidade.

Na sua forma original, o método de Müller não é adequado para determinar raízes múltiplas, não atingindo resultados no limite da precisão digital disponível, como também ocorre no método de Newton clássico.

Agora, confira o algoritmo de Müller no arquivo **Cap3Muller.m** no seu **Caderno de Algoritmos**.

Por fim, para aprofundar os estudos de cada capítulo, disponibilizamos um **Caderno de Exercícios e Respostas** para *download* no *link* <<http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>>.

3.4 CONCLUSÕES

Neste capítulo, cumprindo seu objetivo principal, tratamos da solução numérica de equações a uma variável $f(x) = 0$ quaisquer. Conforme alertamos no início da abordagem desse problema, esta não é uma tarefa simples, especialmente devido à inexistência de um método ou mesmo de uma metodologia geral que seja efetiva em todas as equações.

Apresentamos três metodologias, que não são as únicas existentes, e analisamos detalhadamente o comportamento algébrico, gráfico, numérico e computacional de pelo menos dois métodos típicos de cada uma.

Também tratamos minuciosamente da questão da influência da multiplicidade nas soluções de equações polinomiais $P_n(x) = 0$, disponibilizando uma alternativa para obter todas as raízes de equações polinomiais com a máxima precisão possível.

RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES NÃO LINEARES

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- aplicar métodos iterativos para obter a solução de um sistema não linear e avaliar a precisão do resultado obtido; e
- utilizar os algoritmos disponibilizados.

No Capítulo 2, abordamos a solução de sistemas de equações lineares $A * X = B$ e, no Capítulo 3, tratamos da solução de equações não lineares com uma única incógnita $f(x) = 0$. Como na modelagem matemática dos fenômenos físicos, também podem estar envolvidas relações **não lineares** entre várias incógnitas, as quais geram um sistema de equações não lineares, os mesmos devem ser resolvidos.

Vamos iniciar o estudo das equações não lineares apresentando duas definições importantes.

Definição 1: sistema de n equações não lineares com n incógnitas é toda expressão do tipo:

$$\begin{cases} f_1(x_1, x_2, x_3, \dots, x_n) = 0 \\ f_2(x_1, x_2, x_3, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, x_3, \dots, x_n) = 0 \end{cases} \Rightarrow F(X) = 0$$

Em que

$$F(X) = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \text{ e } X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Observe que temos um vetor F de n funções e um vetor X de n incógnitas. Por exemplo:

$$\text{a) } \begin{cases} f_1(x_1, x_2) = x_1 + 2x_2^3 - 3 = 0 \\ f_2(x_1, x_2) = 3x_1^2 + x_2^2 - 7 = 0 \end{cases} \Rightarrow F = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} \text{ e } X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$\text{b) } \begin{cases} f_1(x_1, x_2, x_3) = 3x_1 - \cos(x_1 x_3) - 0.5 = 0 \\ f_2(x_1, x_2, x_3) = x_1^3 - x_1 x_2 x_3 - 5 = 0 \\ f_3(x_1, x_2, x_3) = e^{x_1 x_2} - \ln(x_1^2 + x_2 x_3) = 0 \end{cases} \Rightarrow F = \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix} \text{ e } X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Definição 2: solução de $F(X) = 0$ é todo vetor.

$$\alpha = [\alpha_1, \alpha_2, \dots, \alpha_n]^T, \text{ onde } \alpha_i \in \mathbb{C} / F(\alpha) = 0$$

Por exemplo:

$$\begin{cases} x_1 x_2 = 1 \\ x_1^2 + x_2^2 - 4x_1 + 2x_2 = 4 \end{cases} \Rightarrow \begin{cases} f_1(x_1, x_2) = x_1 x_2 - 1 = 0 \\ f_2(x_1, x_2) = x_1^2 + x_2^2 - 4x_1 + 2x_2 - 4 = 0 \end{cases}$$

$$\alpha = \begin{bmatrix} 0.604068 \\ 1.655442 \end{bmatrix}$$

Note a dificuldade de isolar, via métodos abstrativos, a solução α já a partir de $n = 2$ equações, restando, por consequência, o uso de métodos iterativos.

Aqui vamos apresentar três métodos iterativos da família newtoniana, que podem ser considerados extensões adaptadas daqueles que vimos no Capítulo 3, para resolver uma única equação a uma variável.

A solução de um sistema não linear consiste em determinar pontos no subespaço n dimensional do domínio do problema que satisfaçam o conjunto de equações. Para os sistemas lineares, temos apenas três possibilidades de quantidade de soluções: **não existência**, **solução única** ou **infinitas soluções**. Já para os sistemas não lineares, podemos ter **nenhuma**, **uma**, um **número finito** ou até um **número infinito** de soluções e inexistem formas de determinar algebricamente a quantidade de soluções de um sistema genérico $F(X) = 0$.

No **Exemplo 4.1**, vamos apresentar um sistema de duas equações não lineares com duas incógnitas, no qual geometricamente as quatro soluções são os pontos de interseção dos gráficos das funções geradoras das duas equações.

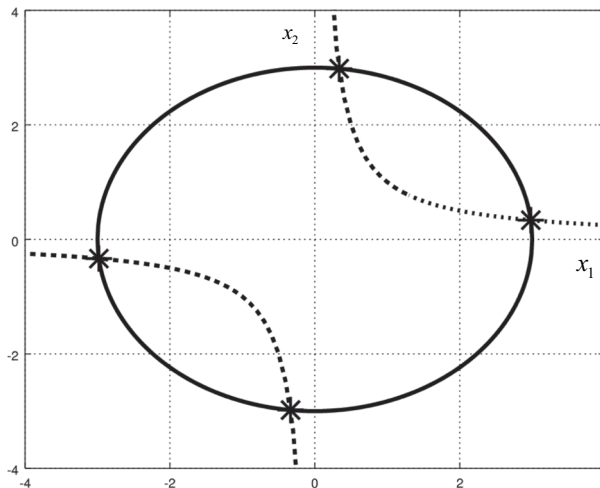
Exemplo 4.1: localize geometricamente as soluções de:

$$\begin{cases} x_1^2 + x_2^2 - 9 = 0 \\ x_1 x_2 - 1 = 0 \end{cases}$$

Solução:

Como este exemplo envolve apenas duas equações simples com duas incógnitas, basta gerar os gráficos de x_2 em função de x_1 , correspondendo a uma circunferência na primeira e uma hipérbole na segunda, resultando:

Gráfico 4.1 – Interseção dos gráficos das funções do **Exemplo 4.1**



Fonte: Elaboração própria.

Portanto, no Gráfico 4.1, temos quatro pontos de interseção das duas funções, que são as quatro soluções do sistema dado.

A aproximação geométrica de uma solução Real (\mathbb{R}), ou a determinação de valores iniciais de uma solução Real, somente é possível para sistemas de duas ou três equações.

Para uma solução geral por métodos iterativos, normalmente tomamos valores iniciais baseados em conhecimentos sobre as grandezas físicas envolvidas no modelo matemático representado pelo sistema de equações. Além disso, alguns sistemas não lineares podem ter raízes Complexas (\mathbb{C}), sendo necessário fornecer uma solução inicial Complexa.

Nas próximas seções, vamos apresentar três métodos iterativos da família newtoniana, pois os de outras famílias necessitam de fundamentações que estão além do escopo deste livro.

4.1 MÉTODO DE NEWTON

No sistema não linear $F(X) = 0$,

$$\begin{cases} f_1(x_1, x_2, \dots, x_n) = 0 \\ f_2(x_1, x_2, \dots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \dots, x_n) = 0 \end{cases} \quad (1)$$

Tomando um vetor solução inicial $X^{(0)}$, calculamos um vetor incremento $\Delta X = X - X^{(0)}$ e obtemos o vetor solução aproximada de incógnitas $X = X^{(0)} + \Delta X$, em que:

$$X^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix}, \quad \Delta X = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix} = \begin{bmatrix} x_1 - x_1^{(0)} \\ x_2 - x_2^{(0)} \\ \vdots \\ x_n - x_n^{(0)} \end{bmatrix} \quad \text{e} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Aplicando a expansão em série de Taylor em cada uma das n funções não lineares $f_i(x_1, x_2, \dots, x_n)$, $i = 1, \dots, n$, em torno do ponto inicial $X^{(0)}$, com $\Delta x_j = x_j - x_j^0$, $j = 1, \dots, n$, analogamente à expansão em série de Taylor aplicada no método de Newton apresentado no Capítulo 3, temos:

$$\begin{cases} f_1(X) = f_1(X^{(0)}) + \frac{\partial f_1(X^{(0)})}{\partial x_1} \Delta x_1 + \frac{\partial f_1(X^{(0)})}{\partial x_2} \Delta x_2 + \dots + \frac{\partial f_1(X^{(0)})}{\partial x_n} \Delta x_n + O(\Delta x_j)^2 \\ f_2(X) = f_2(X^{(0)}) + \frac{\partial f_2(X^{(0)})}{\partial x_1} \Delta x_1 + \frac{\partial f_2(X^{(0)})}{\partial x_2} \Delta x_2 + \dots + \frac{\partial f_2(X^{(0)})}{\partial x_n} \Delta x_n + O(\Delta x_j)^2 \\ \vdots \\ f_n(X) = f_n(X^{(0)}) + \frac{\partial f_n(X^{(0)})}{\partial x_1} \Delta x_1 + \frac{\partial f_n(X^{(0)})}{\partial x_2} \Delta x_2 + \dots + \frac{\partial f_n(X^{(0)})}{\partial x_n} \Delta x_n + O(\Delta x_j)^2 \end{cases} \quad (2)$$

Em cada $f_i(X) = 0, i = 1, \dots, n$, desprezando os termos de ordem superior $O(\Delta x_j)^2$ e reescrevendo a eq. (2) para a forma matricial, resulta o seguinte sistema linear:

$$\begin{bmatrix} \frac{\partial f_1(X^{(0)})}{\partial x_1} & \frac{\partial f_1(X^{(0)})}{\partial x_2} & \dots & \frac{\partial f_1(X^{(0)})}{\partial x_n} \\ \frac{\partial f_2(X^{(0)})}{\partial x_1} & \frac{\partial f_2(X^{(0)})}{\partial x_2} & \dots & \frac{\partial f_2(X^{(0)})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(X^{(0)})}{\partial x_1} & \frac{\partial f_n(X^{(0)})}{\partial x_2} & \dots & \frac{\partial f_n(X^{(0)})}{\partial x_n} \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \vdots \\ \Delta x_n \end{bmatrix} = - \begin{bmatrix} f_1(X^{(0)}) \\ f_2(X^{(0)}) \\ \vdots \\ f_n(X^{(0)}) \end{bmatrix} \quad (3)$$

Resolvendo por eliminação gaussiana o sistema linear dado pela eq. (3), determinamos os valores de cada Δx_j e posteriormente obtemos os novos valores das incógnitas do sistema não linear via $x_j = x_j^{(0)} + \Delta x_j, j = 1, \dots, n$. Esse processo precisa ser repetido para melhorar a precisão da solução, atualizando o conjunto de valores iniciais $X^{(0)}$ pelos novos valores X calculados. Com o novo $X^{(0)}$, geramos e resolvemos o sistema linear novamente e o atualizamos até que algum critério de parada seja satisfeito. Tal critério pode ser o mesmo entre aqueles utilizados na solução iterativa de sistemas lineares.

Observe que, considerando um contador de iteração k , o método de Newton para sistemas de equações não lineares torna-se:

$$J(X^{(k)}) * \Delta X = - F(X^{(k)}) \quad (4)$$

$$X^{(k+1)} = X^{(k)} + \Delta X \quad (5)$$

Em que

$$J(X) = \begin{bmatrix} \frac{\partial f_1(X)}{\partial x_1} & \frac{\partial f_1(X)}{\partial x_2} & \dots & \frac{\partial f_1(X)}{\partial x_n} \\ \frac{\partial f_2(X)}{\partial x_1} & \frac{\partial f_2(X)}{\partial x_2} & \dots & \frac{\partial f_2(X)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n(X)}{\partial x_1} & \frac{\partial f_n(X)}{\partial x_2} & \dots & \frac{\partial f_n(X)}{\partial x_n} \end{bmatrix} \quad (6)$$

A matriz $J(X)$ contém todas as derivadas parciais de 1ª ordem possíveis da função $F(X)$ e é denominada de **jacobiана** dessa função.

Exemplo 4.2: resolva $\begin{cases} e^{x_1} + x_2 = 1 \\ x_1^2 + x_2^2 = 4 \end{cases}$ por Newton, com 3 iterações a partir de $X^{(0)} = \begin{bmatrix} +1 \\ -1 \end{bmatrix}$, e calcule o erro de truncamento da solução atingida.

Solução:

Temos

$$F(X) = \begin{cases} f_1(x_1, x_2) = e^{x_1} + x_2 - 1 = 0 \\ f_2(x_1, x_2) = x_1^2 + x_2^2 - 4 = 0 \end{cases}$$

Obtendo a jacobiана e o sistema de equações lineares para essas duas equações, conforme as eqs. (4) e (5), temos:

$$\begin{bmatrix} \frac{\partial f_1(X)}{\partial x_1} & \frac{\partial f_1(X)}{\partial x_2} \\ \frac{\partial f_2(X)}{\partial x_1} & \frac{\partial f_2(X)}{\partial x_2} \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = - \begin{bmatrix} f_1(X) \\ f_2(X) \end{bmatrix}$$

$$\begin{bmatrix} e^{x_1} & 1 \\ 2x_1 & 2x_2 \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = - \begin{bmatrix} e^{x_1} + x_2 - 1 \\ x_1^2 + x_2^2 - 4 \end{bmatrix} \quad (*)$$

Primeira iteração: aplicamos $X^{(0)} = \begin{bmatrix} +1 \\ -1 \end{bmatrix}$ em (*)

$$\begin{bmatrix} e^1 & 1 \\ 2 & -2 \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = - \begin{bmatrix} e-2 \\ -2 \end{bmatrix} \Rightarrow \begin{cases} \Delta x_1 = +0.075766 \\ \Delta x_2 = -0.924234 \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = +1.0758 \\ x_2^{(1)} = -1.9242 \end{cases}$$

Critério de parada: $\sum_{j=1}^n |\Delta x_j| = 1.000000$.

Segunda iteração: aplicamos $X^{(1)} = \begin{bmatrix} +1.0758 \\ -1.9242 \end{bmatrix}$ em (*)

$$\begin{bmatrix} 2.9322372 & 1 \\ 2.1515314 & -3.8484686 \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = - \begin{bmatrix} 0.0080029 \\ 0.8599495 \end{bmatrix} \Rightarrow \begin{cases} \Delta x_1 = -0.066295 \\ \Delta x_2 = +0.186389 \end{cases}$$

$$\begin{cases} x_1^{(2)} = +1.0095 \\ x_2^{(2)} = -1.7378 \end{cases}$$

Critério de parada: $\sum_{j=1}^n |\Delta x_j| = 0.2527$.

Terceira iteração: aplicamos $X^{(2)} = \begin{bmatrix} +1.0095 \\ -1.7378 \end{bmatrix}$ em (*)

$$\begin{bmatrix} 2.7441484 & 1 \\ 2.0189416 & -3.4756897 \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = - \begin{bmatrix} 0.0080029 \\ 0.0391360 \end{bmatrix} \Rightarrow \begin{cases} \Delta x_1 = -0.0052822 \\ \Delta x_2 = +0.0081916 \end{cases}$$

$$\begin{cases} x_1^{(3)} = +1.0042 \\ x_2^{(3)} = -1.7297 \end{cases}$$

Critério de parada: $\sum_{j=1}^n |\Delta x_j| = 0.013474$.

Podemos obter o erro de truncamento dessa solução aproximada alcançada na

terceira iteração $\begin{cases} x_1^{(3)} = 1.0042 \\ x_2^{(3)} = -1.7297 \end{cases}$, com critério de parada $\sum_{j=1}^n |\Delta x_j| = 0.013474$,

por comparação com uma solução estimada com o dobro de iterações ou com o quadrado do limite para o critério de parada, conforme vimos no Capítulo 2 para soluções de sistemas lineares por métodos iterativos. Nesse exemplo, a solução calculada em variável *double* atinge critério de convergência numericamente nulo em 6 iterações, no limite da precisão envolvida:

$$\Rightarrow \begin{cases} x_1^{(6)} = +1.00416873847466 \\ x_2^{(6)} = -1.72963728702587 \end{cases}$$

Então, os erros de truncamento de cada incógnita x_1 e x_2 são:

$$\text{Erro de truncamento } X^{(3)} = \begin{bmatrix} | 1.0042 - (1.00416873847466) | \\ | -1.7297 - (-1.72963728702587) | \end{bmatrix} = \begin{bmatrix} 3.1262e-05 \\ 6.2713e-05 \end{bmatrix}$$

O erro máximo de truncamento da solução obtida com 3 iterações é $6.2713 \cdot 10^{-05}$, enquanto o critério de parada $\sum_{j=1}^n |\Delta x_j|$ atingido em 3 iterações é 0.013474, então, nesse exemplo, o critério de parada também pode ser usado como limite superior do erro de truncamento.

Nos sistemas de pequeno porte, em que esse método seja convergente, podemos rapidamente chegar à solução com precisão no limite das variáveis adotadas, ou seja, atingir a solução com 16 dígitos significativos para variável *double*, e obter erro de truncamento na mesma ordem dos erros de arredondamento.

Exemplo 4.3: resolva o sistema de equações não lineares

$$\begin{cases} x_1^2 + x_2^2 + x_3^2 - 1 = 0 \\ 2x_1^2 + x_2^2 - 4x_3 = 0 \\ 3x_1^2 - 4x_2 + x_3^2 = 0 \end{cases}$$

Utilizando o método de Newton, tome o vetor inicial $X^{(0)} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$ e critério de parada $\max |\Delta x_i| < 0.005$.

Solução:

A partir de

$$F(X) = \begin{cases} f_1(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2 - 1 \\ f_2(x_1, x_2, x_3) = 2x_1^2 + x_2^2 - 4x_3 \\ f_3(x_1, x_2, x_3) = 3x_1^2 - 4x_2 + x_3^2 \end{cases}$$

obtemos a matriz jacobiana,

$$J(X) = \begin{bmatrix} 2x_1 & 2x_2 & 2x_3 \\ 4x_1 & 2x_2 & -4 \\ 6x_1 & -4 & 2x_3 \end{bmatrix}$$

Aplicando iterativamente as eqs. (4) e (5), temos:

Primeira iteração: para $k=0$ e $X^{(0)} = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$, temos:

$$J(X^{(0)}) = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -4 \\ 3 & -4 & 1 \end{bmatrix}$$

$$F(X^{(0)}) = \begin{cases} f_1 = 0.5^2 + 0.5^2 + 0.5^2 - 1 = -0.25 \\ f_2 = 2 * 0.5^2 + 0.5^2 - 4 * 0.5 = -1.25 \\ f_3 = 3 * 0.5^2 - 4 * 0.5 + 0.5^2 = -1.00 \end{cases}$$

Aplicando a eq. (3), temos o sistema linear:

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & -4 \\ 3 & -4 & 1 \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \Delta x_3 \end{bmatrix} = - \begin{bmatrix} -0.25 \\ -1.25 \\ -1.00 \end{bmatrix}$$

cuja solução é

$$\Delta X^{(0)} = \begin{bmatrix} 0.375 \\ 0 \\ -0.125 \end{bmatrix}$$

Então, os novos valores do vetor X são dados por:

$$X^{(1)} = X^{(0)} + \Delta X = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.375 \\ 0 \\ -0.125 \end{bmatrix} = \begin{bmatrix} 0.875 \\ 0.500 \\ 0.375 \end{bmatrix} \text{ com } \max |\Delta x_j| = 0.375.$$

Segunda iteração: a partir de $k = 1$ e $X^{(1)} = \begin{bmatrix} 0.875 \\ 0.500 \\ 0.375 \end{bmatrix}$, temos:

$$X^{(2)} = \begin{bmatrix} 0.7898166 \\ 0.4966216 \\ 0.3699324 \end{bmatrix} \text{ com } \max |\Delta x_j| = 0.0851834.$$

Terceira iteração: a partir de $k = 2$ e $X^{(2)} = \begin{bmatrix} 0.7898166 \\ 0.4966216 \\ 0.3699324 \end{bmatrix}$, temos:

$$X^{(3)} = \begin{bmatrix} 0.7852104 \\ 0.4966114 \\ 0.3699228 \end{bmatrix} \text{ com } \max |\Delta x_j| = 0.0046062.$$

O processo convergiu para $X^{(3)}$ com critério de parada $\max |\Delta x_j| < 0.005$ em 3 iterações. Mantendo o mesmo procedimento, podemos chegar a uma solução convergida no limite da precisão das variáveis *double*, em 6 iterações e $\max |\Delta x_j| = 0.0$, dada por:

$$X^{(6)} = \begin{bmatrix} -0.785196933062355 \\ -0.496611392944656 \\ -0.369922830745872 \end{bmatrix}$$

Então, os erros de truncamento de cada incógnita (x_1 , x_2 e x_3) obtida em 3 iterações seriam:

$$\text{Erro de truncamento } X^{(3)} = \begin{bmatrix} |0.7852104 - 0.785196933062355| \\ |0.4966114 - 0.496611392944656| \\ |0.3699228 - 0.369922830745872| \end{bmatrix} = \begin{bmatrix} 1.306694e-05 \\ 1.392945e-06 \\ 2.830746e-06 \end{bmatrix}$$

O erro máximo de truncamento da solução obtida com 3 iterações é $1.306694e-05$, enquanto o critério de convergência $\max |\Delta x_j|$ atingido em 3 iterações é $4.6e-03$; portanto, também nesse exemplo, o critério de convergência pode ser usado como limite superior do erro de truncamento. E o erro máximo de truncamento da solução exata, estimada em 6 iterações $X^{(6)}$, está na ordem dos erros de arredondamento, pois atingiu critério de convergência numericamente nulo.

4.2 MÉTODO DE NEWTON COM DERIVADAS PARCIAIS NUMÉRICAS

Além da necessidade de obter previamente uma solução inicial $X^{(0)}$, o maior problema do método Newton é ter que calcular as n^2 derivadas parciais para obter a matriz jacobiana $J(X)$, conforme a eq. (6).

A alternativa mais simples para contornar esse problema consiste em simular as n^2 derivadas parciais $\frac{\partial f_i(X)}{\partial x_j}$ por meio de aproximações numéricas, conforme feito no método da secante do Capítulo 3.

Por definição, sabemos que a derivada exata de uma função com uma variável é:

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Por extensão, a derivada parcial exata de uma função de várias variáveis é definida por:

$$\frac{\partial f_i(x_1, \dots, x_n)}{\partial x_j} = \lim_{\Delta x_j \rightarrow 0} \frac{f_i(x_1, x_2, \dots, x_j + \Delta x_j, \dots, x_n) - f_i(x_1, x_2, \dots, x_j, \dots, x_n)}{\Delta x_j}$$

Se utilizarmos um Δx_j relativamente pequeno, podemos obter uma aproximação para as derivadas parciais:

$$\frac{\partial f_i(x_1, \dots, x_n)}{\partial x_j} \cong \frac{f_i(x_1, x_2, \dots, x_j + \Delta x_j, \dots, x_n) - f_i(x_1, x_2, \dots, x_j, \dots, x_n)}{\Delta x_j}$$

Então, geramos uma aproximação numérica da matriz jacobiana por meio de:

$$\begin{aligned}
 J(X) \cong & \begin{bmatrix} \frac{f_1(x_1 + \Delta x_1, x_2, \dots, x_n) - f_1(x_1, x_2, \dots, x_n)}{\Delta x_1} & \dots & \frac{f_1(x_1, x_2, \dots, x_n + \Delta x_n) - f_1(x_1, x_2, \dots, x_n)}{\Delta x_n} \\ \frac{f_2(x_1 + \Delta x_1, x_2, \dots, x_n) - f_2(x_1, x_2, \dots, x_n)}{\Delta x_1} & \dots & \frac{f_2(x_1, x_2, \dots, x_n + \Delta x_n) - f_2(x_1, x_2, \dots, x_n)}{\Delta x_n} \\ \vdots & & \vdots \\ \frac{f_n(x_1 + \Delta x_1, x_2, \dots, x_n) - f_n(x_1, x_2, \dots, x_n)}{\Delta x_1} & \dots & \frac{f_n(x_1, x_2, \dots, x_n + \Delta x_n) - f_n(x_1, x_2, \dots, x_n)}{\Delta x_n} \end{bmatrix} \\
 & (7)
 \end{aligned}$$

O custo computacional para aproximar uma $J(X^{(k)})$ é de $(n^2 + n)$ cálculos de valores das funções f_i (são n^2 funções nos pontos incrementados e n no ponto inicial). Precisamos estabelecer, ainda, um Δx_j inicial para cada incógnita x_j . Neste livro, optamos por usar na matriz jacobiana o mesmo Δx_j calculado no incremento do método de Newton. No caso de ocorrer convergência, tanto a matriz jacobiana numérica quanto a solução convergem simultaneamente com a mesma precisão.

Essa alternativa para o método de Newton, com derivadas parciais numéricas, é indicada para a resolução de sistemas em computador a fim de facilitar a entrada de dados, o que permite a generalização do seu algoritmo.

No próximo exemplo, vamos utilizar essa alternativa manualmente com objetivos unicamente didáticos.

Exemplo 4.4: resolva o sistema não linear $\begin{cases} x_1 * x_2 = 1 \\ x_1 - e^{x_2} = 0 \end{cases}$ pelo método de Newton utilizando derivadas parciais numéricas para aproximar a matriz jacobiana. Tome $X^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, incremento inicial $\Delta x_j = 0.1$, para $j = 1, 2$ e critério de parada $\max |\Delta x_j| < 0.00001$.

Solução:

Do sistema dado, temos:

$$\begin{cases} f_1(x_1, x_2) = x_1 x_2 - 1 = 0 \\ f_2(x_1, x_2) = x_1 - e^{x_2} = 0 \end{cases}$$

Primeira iteração:

$$X^{(0)} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix} = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix} \text{ e incremento inicial } \Delta X = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

Determinando numericamente as derivadas da jacobiana, conforme eq. (7), resulta:

$$J(X^{(0)}) \cong \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$a_{11} = \frac{\partial f_1}{\partial x_1} \cong \frac{f_1(x_1^{(0)} + \Delta x_1, x_2^{(0)}) - f_1(x_1^{(0)}, x_2^{(0)})}{\Delta x_1} \cong \frac{(1.1 \cdot 1 - 1) - (1.0 \cdot 1 - 1)}{0.1} = 1$$

$$a_{12} = \frac{\partial f_1}{\partial x_2} \cong \frac{f_1(x_1^{(0)}, x_2^{(0)} + \Delta x_2) - f_1(x_1^{(0)}, x_2^{(0)})}{\Delta x_2} \cong \frac{(1 \cdot 1.1 - 1) - (1 \cdot 1.0 - 1)}{0.1} = 1$$

$$a_{21} = \frac{\partial f_2}{\partial x_1} \cong \frac{f_2(x_1^{(0)} + \Delta x_1, x_2^{(0)}) - f_2(x_1^{(0)}, x_2^{(0)})}{\Delta x_1} \cong \frac{(1.1 - e^1) - (1.0 - e^1)}{0.1} = 1$$

$$a_{22} = \frac{\partial f_2}{\partial x_2} \cong \frac{f_2(x_1^{(0)}, x_2^{(0)} + \Delta x_2) - f_2(x_1^{(0)}, x_2^{(0)})}{\Delta x_2} \cong \frac{(1 - e^{1.1}) - (1 - e^{1.0})}{0.1}$$

$$a_{22} \cong -2.85884195487388$$

Aplicando iterativamente as eqs. (4) e (5), temos:

$$\begin{bmatrix} 1 & 1 \\ 1 & -2.85884195487388 \end{bmatrix} * \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = - \begin{bmatrix} 0 \\ -1.71828182845905 \end{bmatrix}$$

$$\begin{cases} \Delta x_1 = +0.445284323264077 \\ \Delta x_2 = -0.445284323264077 \end{cases} \Rightarrow \begin{cases} x_1^{(1)} = 1.445284323264077 \\ x_2^{(1)} = 0.554715676735923 \end{cases}$$

Critério de parada: $\max |\Delta x_j| = 0.445284323264077$.

Segunda iteração:

$$X^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1.445284323264077 \\ 0.554715676735923 \end{bmatrix} \text{ e } \Delta X = \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \begin{bmatrix} +0.445284323264077 \\ -0.445284323264077 \end{bmatrix}$$

$$\begin{cases} x_1^{(2)} = 1.762918095884706 \\ x_2^{(2)} = 0.569994120607396 \end{cases}$$

Critério de parada: $\max |\Delta x_j| = 0.3176337726206295$.

Repetindo esse processo iterativo até a 5ª iteração, temos:

$$\begin{cases} x_1^{(5)} = 1.763222834351872 \\ x_2^{(5)} = 0.567143290409792 \end{cases}$$

Critério de parada atingido: $\max |\Delta x_j| = 1.43228360503785e-08$.

Você já fez o *download* do **Caderno de Algoritmos** apresentado no Capítulo 2? Caso ainda não o tenha feito, acesse o *link* <<http://sergiopeters.prof.ufsc.br/livro-calculo-numerico-computacional/>> para conferir o algoritmo do método de Newton no arquivo **Cap4SistemasNaolineares.m**.

Considerações importantes sobre o método de Newton com derivadas exatas e numéricas:

- a) Quando resolvemos o **Exemplo 4.2** pelo método de Newton com derivadas exatas, atingimos a solução exata em 6 iterações, e se usarmos derivadas numéricas, como no **Exemplo 4.4**, necessitaremos de 7 iterações.
- b) No método de Newton com derivadas numéricas não podemos convergir os incrementos Δx até valores muito pequenos (abaixo do limite de precisão das variáveis utilizadas), por exemplo, para um valor de $x_1^{(0)} = 1$; não podemos atingir valores de Δx_1 menores do que $O(10^{-16})$, pois esse Δx_1 perde totalmente sua significação em $x_1 = x_1^{(0)} + \Delta x_1$, e o valor incrementado fica igual ao valor inicial, $x_1 = x_1^{(0)} + \Delta x_1 = x_1^{(0)}$, gerando derivada nula. Assim, também usamos como critério de parada um valor mínimo para Δx_j .
- c) O método de Newton com derivadas numéricas é genérico e mais fácil de ser implementado, bastando fornecer um vetor de funções, entretanto exige maior esforço computacional. Então, sempre que possível, é mais rápido usar o método de Newton fornecendo também as funções das derivadas parciais exatas. O problema desse método é o custo da quantidade de operações aritméticas a serem executadas em cada iteração, pois, para gerar a jacobiana, precisamos obter n^2 valores de funções derivadas, ou $n^2 + n$ valores de funções, no caso da derivação numérica, e $O(2n^3/3)$ operações para solver cada sistema linear por Gauss.

A seguir, vamos apresentar um método que reduz o custo do processamento de cada iteração em relação ao método de Newton.

4.3 MÉTODO DE BROYDEN

Para tentar resolver um sistema não linear $F(X) = 0$, procedemos desta forma:

- a) Tomamos uma solução inicial $X^{(0)}$.
- b) Obtemos, usando a derivação numérica (ou a analítica, se possível), a matriz jacobiana $J(X^{(0)})$, a sua inversa $J^{-1}(X^{(0)})$ (conforme vimos no Capítulo 2) e efetuamos a primeira iteração como no método de Newton via:

$$\Delta X = -J^{-1}(X^{(0)}) * F(X^{(0)}) \quad (8a)$$


$$X^{(1)} = X^{(0)} + \Delta X \quad (8b)$$

ou

$$X^{(1)} = X^{(0)} - J^{-1}(X^{(0)}) * F(X^{(0)}) \quad (8c)$$

- c) Para cada uma das demais iterações k , determinamos $X^{(k+1)}$ sem calcular a jacobiana novamente (essa é a grande vantagem deste método), obtendo diretamente a sua inversa $J^{-1}(X^{(k)})$, por meio da fórmula de Sherman-Morrison (BURDEN; FAIRES, 2011), usando os valores iniciais obtidos no item (b), considerando $\Delta F = F(X^{(k)}) - F(X^{(k-1)})$:

$$J^{-1}(X^k) = J^{-1}(X^{k-1}) + \frac{\left[(\Delta X - J^{-1}(X^{k-1}) * \Delta F) \Delta X^T \right] J^{-1}(X^{k-1})}{\Delta X^T \left[J^{-1}(X^{k-1}) * \Delta F \right]} \quad (9)$$

 Superíndice^T, presente na equação (9), indica matriz transposta.

- d) Substituindo a eq. (9) na eq. (8c), temos:

$$X^{k+1} = X^k - J^{-1}(X^k) * (X^k) \quad (10)$$

- e) Repetimos os passos (c) e (d) até que algum critério de parada seja satisfeito. Note que, nesse método, a partir da segunda iteração, executamos $O(n^2)$ operações em cada iteração, pois efetuamos apenas adições e multiplicações de matrizes.

Exemplo 4.5: resolva $\begin{cases} e^{x_1} + x_2 = 1 \\ x_1^2 + x_2^2 = 4 \end{cases}$ por Broyden, com 3 iterações a partir de

$$X^{(0)} = \begin{bmatrix} +1 \\ -1 \end{bmatrix}. \text{ Calcule também a solução com precisão máxima.}$$

Solução:

$$\text{Temos } F(X) = \begin{cases} f_1(x_1, x_2) = e^{x_1} + x_2 - 1 = 0 \\ f_2(x_1, x_2) = x_1^2 + x_2^2 - 4 = 0 \end{cases}$$

Aplicando a 1ª iteração do método de Newton, obtendo a jacobiana inicial e sua inversa, aplicando as eqs. (8a) e (8b), ou (8c) e calculando $\Delta F = F(X^{(k)}) - F(X^{(k-1)})$, temos:

$$J(X) = \begin{bmatrix} \frac{\partial f_1(X)}{\partial x_1} & \frac{\partial f_1(X)}{\partial x_2} \\ \frac{\partial f_2(X)}{\partial x_1} & \frac{\partial f_2(X)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} e^{x_1} & 1 \\ 2x_1 & 2x_2 \end{bmatrix}$$

$$J(X^{(0)}) = \begin{bmatrix} e^1 & 1 \\ 2 \cdot 1 & 2(-1) \end{bmatrix} = \begin{bmatrix} e & 1 \\ 2 & -2 \end{bmatrix}$$

$$J^{-1}(X^{(0)}) = \begin{bmatrix} 0.268941421369995 & 0.13447071068499 \\ 0.268941421369995 & -0.365529289315002 \end{bmatrix}$$

$$F(X^{(0)}) = \begin{bmatrix} 0.718281828459045 \\ -2.0000000000000000 \end{bmatrix}$$

$$\Delta X = \begin{bmatrix} -0.9242343145200196 \\ 0.0757656854799806 \end{bmatrix}$$

$$X^{(1)} = \begin{bmatrix} 1.07576568547998 \\ -1.92423431452002 \end{bmatrix}$$

$$F(X^{(1)}) = \begin{bmatrix} 0.00800289814302335 \\ 0.85994950723254249 \end{bmatrix}$$

Calculamos $\Delta F = F(X^{(k)}) - F(X^{(k-1)})$ para iniciar o método de Broyden:

$$\Delta F = F(X^{(1)}) - F(X^{(0)}) = \begin{bmatrix} -0.710278930316022 \\ 2.859949507232542 \end{bmatrix}$$

E, aplicando iterativamente as eqs. (9) e (10), temos:

Primeira iteração: a partir de $k = 1$, por Broyden, via eqs. (9) e (10),

$$J^{-1}(X^{(1)}) = \begin{bmatrix} 0.2921643816775817 & 0.0990520599255256 \\ 0.2073926822465813 & -0.2716578247461259 \end{bmatrix} \text{ via eq. (9)}$$

$$X^{(2)} = \begin{bmatrix} 0.988247753569072 \\ -1.692282044505352 \end{bmatrix} \text{ via eq. (10)}$$

$$\sum_{j=1}^n |\Delta x_j| = 0.319470201925577$$

Segunda iteração: $k = 2$

$$X^{(3)} = \begin{bmatrix} 1.00301691571886 \\ -1.72788110366784 \end{bmatrix}$$

$$\sum_{j=1}^n |\Delta x_j| = 0.0503682213122766$$

Terceira iteração: $k = 3$

$$X^{(4)} = \begin{bmatrix} 1.00417409441167 \\ -1.72962989476839 \end{bmatrix}$$

$$\sum_{j=1}^n |\Delta x_j| = 0.00290596979335936$$

Seguindo os passos iterativos, chegamos à solução:

$$X = \begin{bmatrix} 1.00416873847466 \\ -1.72963728702587 \end{bmatrix} \text{ em } k = 8 \text{ iterações}$$

$$\sum_{j=1}^n |\Delta x_j| = 2.62331707805421e-16$$

Exemplo 4.6: elabore um algoritmo que resolva o sistema não linear

$$\begin{cases} x_1^2 + x_2^2 + x_3^2 - 1 = 0 \\ 2x_1^2 + x_2^2 - 4x_3 = 0 \\ 3x_1^2 - 4x_2 + x_3^2 = 0 \end{cases}$$

utilizando os métodos de Newton e de *Broyden*, com critério de parada

$\sum_{j=1}^n |\Delta x_j| < 10^{-14}$, e Newton e Broyden com derivadas parciais numéricas e *critério*

de parada min $|\Delta x_i| < 10^{-14}$ partindo de um vetor inicial $X^{(0)} = [0.5, 0.5, 0.5]^T$.

Calcule o resíduo máximo das equações e compare os resultados entre os métodos aplicados.



Para saber mais sobre o método de Broyden, consulte Burden e Faires (2011) e o artigo disponível em: https://es.wikipedia.org/wiki/M%C3%A9todo_de_Broyden. Acesso em: 3 nov. 2016.



Esse critério de parada com mínimo desvio é necessário para que, na simulação das derivadas não sejam geradas derivadas nulas por perda de significação.

Solução:

Comparação entre os métodos:

- a) Método de Newton com derivadas parciais exatas:

$$X = [0.785196933062355, 0.496611392944656, 0.369922830745872]^T$$

$$\text{Resíduo máximo } F = 2.22044604925031e-16$$

$$\text{Critério de parada atingido: } \sum_{j=1}^n |\Delta x_j| = 7.14758491003723e-17$$

$$\text{Número de iterações } k = 6$$

- b) Método de Newton com derivadas parciais numéricas:

$$X = [0.785196933062355, 0.496611392944656, 0.369922830745872]^T$$

$$\text{Resíduo máximo } F = 3.13082892944294e-14$$

$$\text{Critério de parada atingido: } \min |\Delta x_j| = 2.40985795343054e-17$$

$$\text{Número de iterações } k = 6$$

- c) Método de Broyden com derivadas parciais exatas:

$$X = [0.785196933062355, 0.496611392944656, 0.369922830745872]^T$$

$$\text{Resíduo máximo } F = 2.77555756156289e-16$$

$$\text{Critério de parada atingido: } \sum_{j=1}^n |\Delta x_j| = 2.67455660204432e-15$$

$$\text{Número de iterações } k = 8$$

- d) Método de Broyden com derivadas parciais numéricas:

$$X = [0.785196933062355, 0.496611392944656, 0.369922830745872]^T$$

$$\text{Resíduo máximo } F = 2.77555756156289e-16$$

$$\text{Critério de parada atingido: } \min |\Delta x_j| = 4.03368022884459e-16$$

$$\text{Número de iterações } k = 8$$

Os algoritmos de Newton e Broyden aplicados ao **Exemplo 4.6** estão no **Caderno de Algoritmos** no arquivo **Cap4SistemasNaolineares.m**.

4.4 CONCLUSÕES

Tomando como referência apenas resultados típicos, como os colhidos no **Exemplo 4.6**, com a aplicação dos quatro métodos abordados neste capítulo, podemos concluir que:

- a) Os métodos de Newton são os mais recomendados para sistemas não lineares de ordem baixa ou moderada. O Newton geral, quando for possível a determinação direta da jacobiana; o Newton com derivadas numéricas, quando não. Já para sistemas de ordem mais elevada, sugerimos o método de Broyden, o qual normalmente necessita de mais iterações para a mesma precisão, porém o custo de cada iteração é compensadoramente menor.
- b) A recomendação anterior é válida para os casos em que ocorre a convergência, pois, se os métodos em questão continuarem divergindo depois de várias tentativas de tomada de novas soluções iniciais $X^{(0)}$, indicamos a utilização de outros métodos matematicamente mais robustos e que façam uso das derivadas direcionais, como o método das **estimativas descendentes**, que encontram um **mínimo local** da função somatório dos quadrados das funções não lineares de modo que esse mínimo local corresponde à solução do sistema não linear, independentemente do valor inicial adotado (BURDEN; FAIRES, 2011).

Antes de iniciar o estudo do próximo capítulo, aplique o seu conhecimento respondendo aos exercícios do **Caderno de Exercícios e Respostas** disponível para *download* no link <<http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>>.

APROXIMAÇÃO POLINOMIAL POR INTERPOLAÇÃO, *SPLINE* E CURVAS DE BÉZIER

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- efetuar interpolações de funções com uma ou várias variáveis independentes;
- executar aproximações de funções por interpolação *spline*; e
- executar aproximações de funções e não funções por curvas de Bézier.

Nos Capítulos 5, 6 e 7, abordaremos o tópico central do cálculo numérico: a aproximação de uma função $y = f(x)$, $x \in [a, b]$, por meio de outra função $z = g(x)$.

Você pode se perguntar: por que utilizamos uma função aproximadora, ou representante, em vez de usar diretamente a função original? Há dois motivos básicos para essa opção:

Motivo 1: podemos **conhecer apenas um conjunto discreto de pontos** da função $y = f(x)$, do tipo

x_i	x_1	x_2	...	x_{n+1}
$y_i = f(x_i)$	y_1	y_2	...	y_{n+1}

Isso ocorre em situações como:

- a) Coleta de valores amostrais em experimentos para geração de bases de dados, por exemplo, a amostra a seguir contém a relação entre o tempo de uso e o respectivo índice de desgaste de uma peça

x (Tempo de uso em meses)	1	2	4	5	6
y (Índice de desgaste)	0.05	0.09	0.18	0.23	0.31

- b) Em sistemas de computação gráfica, o usuário indica alguns pontos de referência, ou controle, e o computador deve preencher os caminhos intermediários desses pontos sem distorções e com alta velocidade de resposta. Para tanto, precisamos obter funções aproximadoras desses caminhos.

Motivo 2: a função $y = f(x)$ pode possuir **expressão conhecida, porém ser indisponível, ineficiente** ou até impossível de ser utilizada.

Isso ocorre em situações como:

- a) A construção de funções predefinidas para bibliotecas de linguagens computacionais, que devem ter rotinas para obter valores confiáveis de funções como $y = e^x$, $y = \text{sen}(x)$, $y = \text{cos}(x)$, $y = \text{tg}(x)$, entre outras, utilizando apenas as operações aritméticas elementares.
- b) Nos sistemas dedicados, muitas vezes temos que obter, em tempo real, valores de funções extensas e compostas de outras funções específicas, por exemplo:

$$f(x) = \text{sen} \left[\exp \left(\sqrt[3]{\text{coth}(x)} \right) \right]$$

Nesses casos, o tempo de processamento acumulado de cada uma das subfunções pode ser inviável, sendo obrigatória a aproximação da $f(x)$ por uma função $g(x)$ única com tempo de resposta factível.

- c) Alguns modelos matemáticos são representados por funções, cuja expressão algébrica pode ser de uso inviável. Por exemplo, para efetuar analiticamente a integral definida $I = \int_0^2 e^{-x^2} dx$, temos que obter uma aproximadora $z = g(x)$ da função integranda original $f(x) = e^{-x^2}$ que tenha primitiva conhecida, pois $f(x) = e^{-x^2}$ tem primitiva impossível de ser expressa como uma função elementar.

Para fazer uso da função aproximadora, a questão fundamental que precisamos responder é: quem pode ser a aproximadora $z = g(x)$?

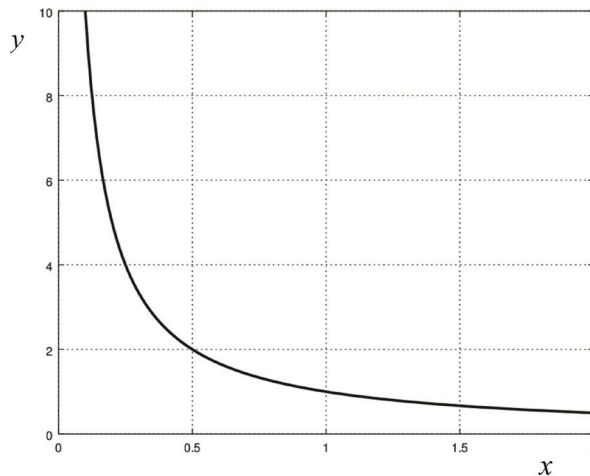
A premissa básica é que a função $z = g(x)$ deva ser simples e “bem comportada”, isto é, ser contínua, facilmente computável, derivável, integrável etc. Por consequência, o universo de busca das famílias de aproximadoras $z = g(x)$ restringe-se às funções do tipo:

a) **Polinomiais:** $g(x) = \sum_{i=1}^{n+1} a_i x^{i-1} = P_n(x)$, cujas vantagens são:

- i) envolvem somente operações elementares;
- ii) facilmente deriváveis, integráveis etc.;
- iii) formam um anel, ou seja, todas as transformações algébricas nelas aplicadas resultam em outro polinômio.

b) **Racionais:** $g(x) = \frac{P_n(x)}{Q_m(x)}$, em que $P_n(x)$ e $Q_m(x)$ são polinômios de graus n e m , respectivamente, e permitem a aproximação de funções com gráfico assintótico, como o Gráfico 5.1.

Gráfico 5.1 – Função assintótica



Fonte: Elaboração própria.

c) **Trigonométricas:** $g(x) = \sum_{i=1}^m (a_i * \text{sen}(i * x) + b_i * \text{cos}(i * x))$ que, fornecem aproximações de funções representativas de fenômenos oscilantes.

Em virtude de suas características, as funções polinomiais são as mais utilizadas como aproximadoras. Contudo, tais vantagens teriam pouca valia se não existisse o suporte teórico de que podem sempre aproximar funções. Tal garantia encontra-se no **teorema de Weierstrass**:

“Se $y = f(x)$ for contínua em $[a, b]$, então $\forall \varepsilon > 0$ sempre existe $P_n(x)$ e n dependente da precisão ε , tal que $|P_n(x) - f(x)| < \varepsilon, \forall x \in [a, b]$.”

Ou seja, sempre existe uma aproximadora polinomial $P_n(x)$ que esteja tão próxima quanto se queira de uma $f(x)$ contínua no domínio $[a, b]$. Como esse teorema assegura a existência da aproximadora, mas não fornece um modo de determiná-la, temos mais uma questão a responder: como obter essa aproximadora?

A resposta dessa questão será abordada em todas as seções deste capítulo.

5.1 APROXIMAÇÃO POR INTERPOLAÇÃO POLINOMIAL

Para aproximar uma função $y = f(x)$ contínua em $[a, b]$, podemos proceder como segue:

- a) Tomamos $n + 1$ pontos $(x_i, y_i = f(x_i))$ com $i = 1, 2, \dots, n+1, x_i \in [a, b]$, conforme:

x_i	x_1	x_2	x_{n+1}
$y_i = f(x_i)$	y_1	y_2	y_{n+1}

com $x_1 = a$ e $x_{n+1} = b$, ou usamos uma função já representada por $n + 1$ pontos de uma tabela proveniente de um levantamento de dados.

- b) Tomamos um polinômio genérico de grau n :

$$P_n(x) = \sum_{i=1}^{n+1} a_i x^{i-1} = a_1 + a_2 x + \dots + a_n x^{n-1} + a_{n+1} x^n \quad (1)$$

E aplicamos a seguinte condição de aproximação:

$$P_n(x_i) = y_i, \quad i = 1, 2, \dots, n + 1 \quad (2)$$



Isto é, impomos a condição de que essa função aproximadora **polinomial deve passar por todos os $n + 1$ pontos** tabelados conforme item (a), ou seja, o erro ou desvio, sobre esses pontos escolhidos para definir a aproximadora, é nulo.

Dessa condição, resulta a expressão:

$$\begin{cases} P_n(x_1) = a_1 + a_2x_1 + \dots + a_nx_1^{n-1} + a_{n+1}x_1^n = y_1 \\ P_n(x_2) = a_1 + a_2x_2 + \dots + a_nx_2^{n-1} + a_{n+1}x_2^n = y_2 \\ \vdots \\ P_n(x_{n+1}) = a_1 + a_2x_{n+1} + \dots + a_nx_{n+1}^{n-1} + a_{n+1}x_{n+1}^n = y_{n+1} \end{cases} \quad (3a)$$

Que é um sistema com $n + 1$ equações lineares e $n + 1$ incógnitas a_i . Quando reescrevemos esse sistema na forma matricial, temos:

$$\begin{bmatrix} 1 & x_1 & \dots & x_1^{n-1} & x_1^n \\ 1 & x_2 & \dots & x_2^{n-1} & x_2^n \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 1 & x_{n+1} & \dots & x_{n+1}^{n-1} & x_{n+1}^n \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n+1} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{bmatrix} \quad (3b)$$

Resolvendo o sistema (3b), obtemos os $n + 1$ coeficientes a_i do polinômio que passa por todos os $n + 1$ pontos $(x_p, y_i = f(x_i))$.



Note que o número de pontos é igual ao número de coeficientes incógnitos a_i .

Esse polinômio aproximador é denominado de **interpolador** da função. Os elementos u_{ij} da matriz dos coeficientes do sistema $U * A = Y$ dado na eq. (3b) têm lei de formação genérica dada por:

$$u_{ij} = x_i^{j-1} \quad (4)$$

Resolvendo o sistema linear $U * A = Y$ conforme abordamos no Capítulo 2, obtemos os coeficientes do polinômio interpolador.

Exemplo 5.1: aproxime, por interpolação, a função $y = \ln(x)$, $x \in [1.0, 2.0]$ dividindo esse intervalo em $n = 2$ subintervalos iguais e estime $\ln(1.14)$. Estime ainda o erro do $\ln(1.14)$ obtido pelo aproximador de grau 2 comparando-o com seu valor exato.

Solução:

$$h = (b - a) / n = (2.0 - 1.0) / 2 = 0.5$$

i	1	2	$2 + 1 = 3$
x_i	1.0	1.5	2.0
$y_i = \ln(x_i)$	0.0000000000000000	0.405465108108164	0.693147180559945

$$n + 1 = 3 \text{ pontos} \Rightarrow n = 2 \text{ subintervalos} \Rightarrow P_2(x) = a_1 + a_2x + a_3x^2$$

Aplicando o sistema dado na eq. (3b), temos:

$$\begin{bmatrix} (1.0) & (1.0)^1 & (1.0)^2 \\ (1.0) & (1.5)^1 & (1.5)^2 \\ (1.0) & (2.0)^1 & (2.0)^2 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0.0000000000000000 \\ 0.405465108108164 \\ 0.693147180559945 \end{bmatrix}$$

E resolvendo esse sistema por eliminação de Gauss, com variáveis *double*, temos:

$$a_1 = -1.164279323185479$$

$$a_2 = 1.399845394498246$$

$$a_3 = -0.235566071312767$$

Assim,

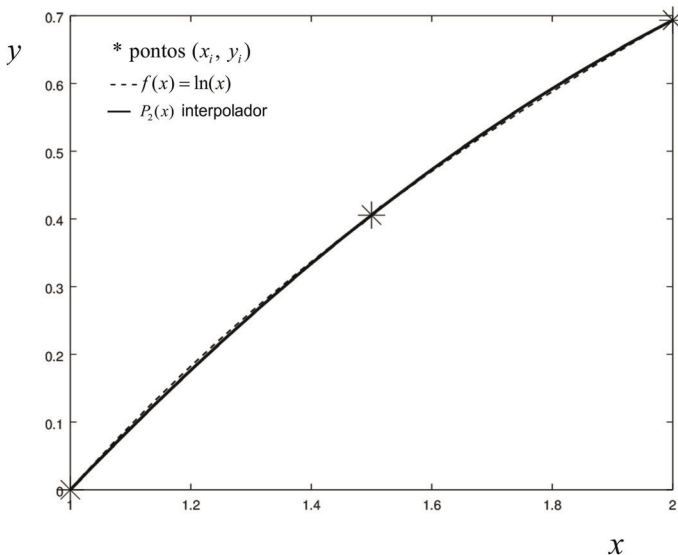
$$P_2(x) = -1.164279323185479 + 1.399845394498246x - 0.235566071312767x^2 \cong \ln(x)$$

Valor aproximado local: $P_2(1.14) = 0.125402760264449 \Rightarrow$ avaliação por Horner.

Valor exato local: $f(1.14) = \ln(1.14) = 0.131028262406404 \Rightarrow$
 $Erro\ local = |P_2(1.14) - f(1.14)| = 0.00562550214195459$

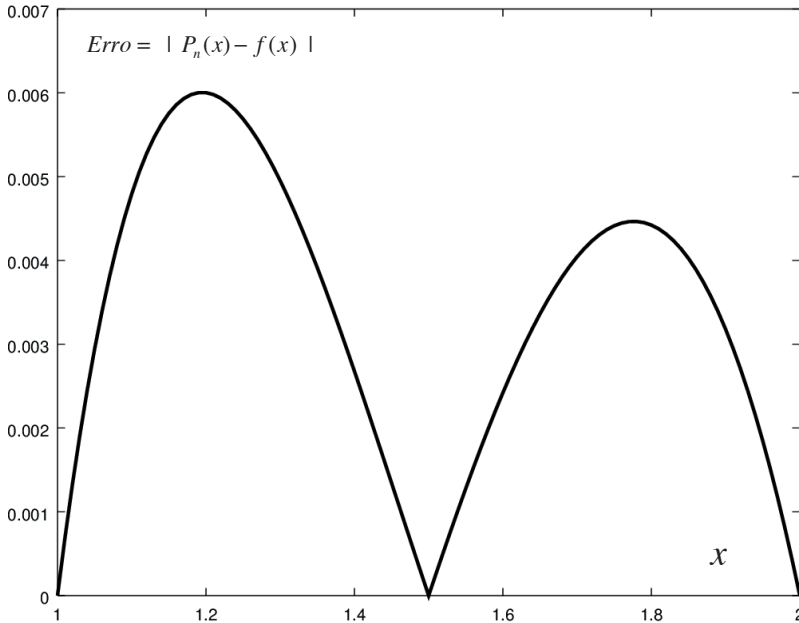
Podemos avaliar a aderência da função interpoladora $P_2(x)$ com a função exata $f(x) = \ln(x)$ no Gráfico 5.2.

Gráfico 5.2 – Função interpoladora $P_2(x)$ e função exata $f(x) = \ln(x)$



Fonte: Elaboração própria.

Podemos também representar o *erro local* exato entre $P_2(x)$ e $f(x)$ no intervalo $x \in [1.0, 2.0]$, conforme o Gráfico 5.3.

Gráfico 5.3 – Erro local exato entre $P_2(x)$ e $f(x)$ 

Fonte: Elaboração própria.

Observe que os *erros locais*, entre a função aproximadora $P_2(x)$ e a exata $f(x)$, são nulos sobre os pontos usados para definir o interpolador $(x_p, y_i = f(x_i))$, e o erro máximo é $ErroMax \cong 0.006$. Observe também que os erros têm pico máximo na região intermediária de cada subintervalo entre os pontos $(x_p, y_i = f(x_i))$ interpolados e serão maiores nos subintervalos das duas extremidades, conforme pode ser visto no **Exercício 5.3**.

No **Caderno de Exercícios e Respostas**, disponível no *link* <<http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>>, você encontrará os exercícios atualizados deste livro.

Já no **Caderno de Algoritmos**, disponível no *link* <<http://sergiopeters.prof.ufsc.br/algoritmos-livro/>>, apresentamos *um algoritmo que implementa a interpolação polinomial geral* no arquivo **Cap5exem5.1coefinterPn1D.m**, contendo o cálculo dos coeficientes a_i do polinômio interpolador e seu uso.



Veremos, nas seções 5.1.2.1 e 5.1.2.2, que existem algoritmos alternativos para determinar o polinômio interpolador.

Fazendo uma análise da técnica de aproximação por interpolação polinomial, devemos considerar pelo menos três questões fundamentais:

- Será que o sistema $U^* A = Y$, dado pela eq. (3b), sempre tem solução? Caso exista solução, esta será única?
- Será possível melhorar a eficiência computacional (menor tempo de resposta e demanda de memória) para a obtenção e o uso do $P_n(x)$?
- Qual é o erro de truncamento máximo associado ao se tomar $P_n(x)$ como aproximador de $f(x)$, $\forall x \in [a, b]$ com $x \neq x_i$?

Vamos tratar de responder a cada uma dessas questões nas três seções a seguir.

5.1.1 Unicidade do interpolador

Conforme detalhamos anteriormente, para obter o polinômio interpolador de uma função definida por:

x_i	x_1	x_2	...	x_{n+1}
$y_i = f(x_i)$	y_1	y_2	...	y_{n+1}

geramos um sistema de equações lineares dado pela eq. (3b), cuja matriz U dos coeficientes é especial, devido à lei de formação dos seus elementos. Esse tipo de matriz é denominado de **Vandermonde**, cuja forma simples e eficiente de obter o seu determinante é:

$$\text{Det}(U) = \text{Det} \begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} & x_1^n \\ 1 & x_2 & \cdots & x_2^{n-1} & x_2^n \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ 1 & x_{n+1} & \cdots & x_{n+1}^{n-1} & x_{n+1}^n \end{bmatrix} = \prod_{\substack{i=n+1,2 \\ j=i-1,1 \\ (i \neq j)}} (x_i - x_j) \quad (5a)$$

ou

$$\begin{aligned}
 \text{Det}(U) &= (x_{n+1} - x_n)(x_{n+1} - x_{n-1}) \dots (x_{n+1} - x_1) \\
 &\quad * (x_n - x_{n-1})(x_n - x_{n-2}) \dots (x_n - x_1) \\
 &\quad \vdots \\
 &\quad * (x_3 - x_2)(x_3 - x_1) \\
 &\quad * (x_2 - x_1)
 \end{aligned} \tag{5b}$$

Como no conjunto de pontos $(x_i, y_i = f(x_i))$ geradores da matriz U não existem valores de x repetidos por se tratar de uma função, isto é, $x_i \neq x_j$ para $i \neq j$, logo $\text{Det}(U) \neq 0$, conforme a eq. (5a) ou (5b), e o sistema $U * A = Y$ dado na eq. (3b) terá solução única. Assim, o polinômio gerado será único, independentemente da maneira de expressá-lo. Essa demonstração generaliza o fato conhecido de que por dois pontos quaisquer passa uma única reta.

Na sequência, vamos mostrar formas mais eficientes de determinar o polinômio interpolador de grau n .

5.1.2 Determinação eficiente do interpolador

A consequência da unicidade do interpolador é que podemos tentar determiná-lo sem a geração do sistema linear (3b), que demanda a ordem de $O(n^2)$ multiplicações para gerar a matriz U e cuja solução por eliminação de Gauss ou Crout exige número de operações aritméticas na ordem de $O(2n^3 / 3)$.

Para a função discretizada

i	1	2	$n + 1$
x_i	x_1	x_2	x_{n+1}
$y_i = f(x_i)$	y_1	y_2	y_{n+1}

propomos representações de polinômios interpoladores em bases alternativas, pois a eq. (1) é escrita como uma combinação linear direta da base canônica dos polinômios: $\{x^0, x^1, \dots, x^{n-1}, x^n\}$, com coeficientes a_p , ou seja,

$$P_n(x) = \sum_{i=1}^{n+1} a_i x^{i-1} = a_1 x^0 + a_2 x^1 + \cdots + a_n x^{n-1} + a_{n+1} x^n$$

5.1.2.1 Expressão do interpolador polinomial $P_n(x)$ na base dos polinômios de Lagrange

Nesse caso, o interpolador $P_n(x)$ é expresso na forma de uma combinação linear da base definida por polinômios de Lagrange de grau n , $L_i(x)$, $i = 1, \dots, n + 1$,

$$L_i(x) = \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{(x - x_j)}{(x_i - x_j)} \quad (6a)$$

com as propriedades:

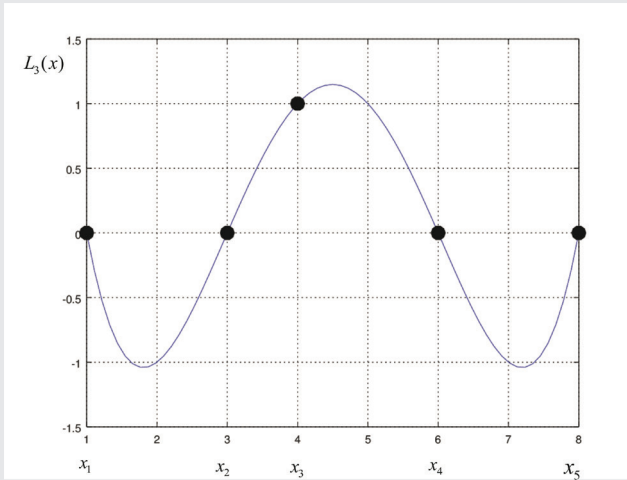
$$L_i(x_i) = 1 \text{ para qualquer } i \text{ e } L_i(x_j) = 0 \text{ para } j \neq i \quad (6b)$$

Cada polinômio $L_i(x)$ se anula em todos os pontos conhecidos x_j , com exceção de um deles, x_i . Os polinômios de Lagrange podem ser considerados uma função “peso”, com valor 1 sobre x_i e 0 sobre x_j . Cada $L_i(x)$ é definido em torno de um x_i e é uma função linearmente independente de outro $L_r(x)$ com $r \neq i$.

Por exemplo, para o conjunto de 5 pontos:

i	1	2	3	4	5
x_i	1	3	4	6	8

O polinômio de Lagrange de grau 4 definido em torno de $x_3 = 4$, $L_3(x)$ resulta no Gráfico 5.4.

Gráfico 5.4 – Polinômio de Lagrange $L_3(x)$ de grau 4 em $x_3 = 4$ 

Fonte: Elaboração própria.

Observe que o polinômio $L_3(x)$ do Gráfico 5.4 tem valor nulo sobre $x_1 = 1$, $x_2 = 3$, $x_4 = 6$, $x_5 = 8$ e valor unitário em $x_3 = 4$.

Assim, um interpolador genérico $P_n(x)$ de grau n pode ser expresso como combinação linear da **base de polinômios de Lagrange** $L_i(x)$, ponderados diretamente por **coeficientes** com valores iguais a y_i , em função das propriedades estabelecidas na eq. (6b), conforme segue:

$$P_n(x) = \sum_{i=1}^{n+1} y_i L_i(x) \quad (7)$$

$$P_n(x) = \sum_{i=1}^{n+1} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{(x - x_j)}{(x_i - x_j)} \quad (8a)$$

ou

$$\begin{aligned}
 P_n(x) &= y_1 \frac{(x-x_2)(x-x_3) \dots (x-x_{n+1})}{(x_1-x_2)(x_1-x_3) \dots (x_1-x_{n+1})} \\
 &+ y_2 \frac{(x-x_1)(x-x_3) \dots (x-x_{n+1})}{(x_2-x_1)(x_2-x_3) \dots (x_2-x_{n+1})} \\
 &+ \dots \\
 &+ y_{n+1} \frac{(x-x_1)(x-x_2) \dots (x-x_n)}{(x_{n+1}-x_1)(x_{n+1}-x_2) \dots (x_{n+1}-x_n)}
 \end{aligned} \tag{8b}$$

Como $L_i(x_i) = 1$ e $L_i(x_j) = 0$, então

$$\begin{cases}
 P_n(x_1) = y_1 \\
 P_n(x_2) = y_2 \\
 \vdots \\
 P_n(x_{n+1}) = y_{n+1}
 \end{cases}$$

Ou seja, $P_n(x)$ passa por todos os pontos (x_i, y_i) .

Como o interpolador $P_n(x)$ dado pela eq. (8a) passa por todos os $n + 1$ pontos (x_i, y_i) , ele será o mesmo que o obtido anteriormente pelo sistema $U * A = Y$ dado pela eq. (3b), em consequência da unicidade do polinômio interpolador. Na forma de Lagrange não temos a necessidade de gerar e nem de resolver um sistema linear, basta substituir diretamente os valores dos pontos (x_i, y_i) e o x desejado na eq. (8a).

Exemplo 5.2: determine o interpolador de Lagrange para a função discretizada:

x_i	0	1	3	4
$y_i = f(x_i)$	2	4	0	1

Aproxime $f(x = 2)$ e $f(x = 5)$.

Solução:

Temos $n + 1 = 4$ pontos de $f(x)$ e $n = 3$.

$$\begin{aligned}
 P_3(x) &= y_1 \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} \\
 &+ y_2 \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)} \\
 &+ y_3 \frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)} \\
 &+ y_4 \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)}
 \end{aligned}$$

$$\begin{aligned}
 P_3(x) &= 2 \frac{(x-1)(x-3)(x-4)}{(0-1)(0-3)(0-4)} + 4 \frac{(x-0)(x-3)(x-4)}{(1-0)(1-3)(1-4)} \\
 &+ 0 \frac{(x-0)(x-1)(x-3)}{(3-0)(3-1)(3-4)} + 1 \frac{(x-0)(x-1)(x-3)}{(4-0)(4-1)(4-3)}
 \end{aligned}$$

Resultando nas estimativas:

$$f(x=2) \cong P_3(x=2) = 2.167$$

$$f(x=5) \cong P_3(x=5) = 8.667$$

Também poderíamos expandir algebricamente a expressão de $P_n(x)$ dada pela eq. (8b) (base de Lagrange) e chegaríamos à mesma expressão dada pela eq. (1) (base canônica) com coeficientes dados pela eq. (3b), possivelmente com menos erros de arredondamento.

Já para efeitos de otimização do algoritmo, podemos reescrever a expressão dada pela eq. (8a) desta forma:

$$\left\{ \begin{aligned}
 Num &= \prod_{j=1}^{n+1} (x - x_j) \\
 P_n(x) &= \sum_{i=1}^{n+1} y_i \frac{Num}{(x - x_i) \prod_{\substack{j=1 \\ j \neq i}}^{n+1} (x_i - x_j)}, \quad \forall x \neq x_i
 \end{aligned} \right. \quad (8c)$$

Antes de continuar a sua leitura, teste os algoritmos na forma da eq. (8a) e na forma otimizada da eq. (8c) disponíveis no **Caderno de Algoritmos** no arquivo **Cap5exem5.2PnLagrange1D.m**.

Observe que o número de operações envolvido nesse algoritmo otimizado, eq. (8c), fica reduzido a $2n^2 + 9n$ operações aritméticas para cada avaliação de $P_n(v) \cong f(v)$, com a restrição de que $v \neq x_i$.

5.1.2.2 Interpolador de Gregory-Newton com diferenças

Há duas definições importantes a considerar em relação ao polinômio de Gregory-Newton.

Definição 1: para a função discretizada

i	1	2	...	$n+1$
x_i	x_1	x_2	...	x_{n+1}
$y_i = f(x_i)$	y_1	y_2	...	y_{n+1}

definimos as **diferenças divididas** $\Delta^k y_i$, no sentido ascendente, por:

$$\Delta^1 y_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}, \quad \forall i = 1, \dots, n \Rightarrow \text{diferença de 1ª ordem.}$$

$$\Delta^2 y_i = \Delta(\Delta y_i) = \frac{\Delta y_{i+1} - \Delta y_i}{x_{i+2} - x_i}, \quad \forall i = 1, \dots, n-1 \Rightarrow \text{diferença de 2ª ordem.}$$

$$\Delta^k y_i = \frac{\Delta^{k-1} y_{i+1} - \Delta^{k-1} y_i}{x_{i+k} - x_i}, \quad \forall i = 1, \dots, n+1-k \Rightarrow \text{diferença de } k\text{-ésima ordem.}$$

Note a semelhança entre as definições das diferenças divididas e das derivadas de várias ordens de uma função com expressão conhecida.

Exemplo 5.3: determine todas as diferenças divididas da função discretizada

i	1	2	3	4
x_i	1	3	4	7
$y_i = f(x_i)$	2	0	1	3

Solução:

Aplicando a definição 1 temos:

i	x_i	y_i	$\Delta^1 y_i$	$\Delta^2 y_i$	$\Delta^3 y_i$
1	1	2	-1	2/3	-1/8
2	3	0	1	-1/12	-
3	4	1	2/3	-	-
4	7	3	-	-	-

$$\Delta^1 y_1 = \frac{[(0)-(2)]}{(3-1)} = -\frac{2}{2} = -1$$

$$\Delta^2 y_1 = \frac{[(1)-(-1)]}{(4-1)} = \frac{2}{3}$$

$$\Delta^3 y_1 = \frac{[(-1/12)-(2/3)]}{(7-1)} = -\frac{1}{8}$$

Observe que a diferença dividida $\Delta^3 y_i$, em $i = 1$ (destacada em negrito), depende de todos os pontos e de todas as diferenças anteriores.

Definição 2: para a função discretizada

i	1	2	$n + 1$
x_i	1	3	x_{n+1}
$y_i = f(x_i)$	2	0	y_{n+1}

definimos o polinômio interpolador $P_n(x)$ de grau n , na forma de **Gregory-Newton**, como combinação linear da base

$$\{x^0, (x-x_1), (x-x_1)(x-x_2), \dots, \prod_{j=1}^n (x-x_j)\}$$

ponderada diretamente pelos **coeficientes** dados pelas diferenças divididas, $\Delta^k y_1$ conforme segue:

$$P_n(x) = y_1 + \sum_{k=1}^n \Delta^k y_1 \left[\prod_{j=1}^k (x - x_j) \right] \quad (9a)$$

ou

$$P_n(x) = y_1 + \Delta y_1(x - x_1) + \Delta^2 y_1(x - x_1)(x - x_2) + \dots + \Delta^n y_1(x - x_1)\dots(x - x_n) \quad (9b)$$

resulta que $P_n(x)$ passa por todos os pontos discretizados, pois

$$\left\{ \begin{array}{l} P_n(x_1) = y_1 \\ P_n(x_2) = y_1 + \frac{(y_2 - y_1)}{(x_2 - x_1)}(x_2 - x_1) = y_2 \\ P_n(x_3) = y_3 \\ \vdots \\ P_n(x_{n+1}) = y_{n+1} \end{array} \right.$$

Todas essas igualdades podem ser comprovadas por indução finita. Logo, o polinômio $P_n(x)$ dado pela eq. (9a) também é uma forma alternativa do único interpolador de grau n que passa sobre todos os pontos discretizados, assim como as formas dadas anteriormente pelas equações (1) e (8a).

Exemplo 5.4: determine o interpolador de Gregory-Newton da função

x_i	1	3	4	7
$y_i = f(x_i)$	2	0	1	3

Estime $f(x = 5)$ e $f(x = 7.5)$.

Solução:

Temos $n + 1 = 4 \Rightarrow n = 3$

$$P_3(x) = y_1 + \Delta y_1(x - x_1) + \Delta^2 y_1(x - x_1)(x - x_2) + \Delta^3 y_1(x - x_1)(x - x_2)(x - x_3)$$

Usando as diferenças divididas obtidas no **Exemplo 5.3**, temos:

$$P_3(x) = 2(1) + (-1)(x - 1) + (2/3)(x - 1)(x - 3) + (-1/8)(x - 1)(x - 3)(x - 4)$$

E efetuando as estimativas, resultam:

$$f(5.0) \cong P_3(5.0) = 2.233$$

$$f(7.5) \cong P_3(7.5) = 2.203$$

Na algoritmização desse interpolador, podemos fazer o armazenamento das diferenças divididas na forma de um vetor, haja vista que apenas as diferenças no 1º ponto, em $i = 1$, são utilizadas na forma final, portanto não há necessidade de uma matriz para esse fim. No momento da determinação de todas as diferenças, bastará usar mais um vetor auxiliar.

Confira o algoritmo de Gregory-Newton no **Caderno de Algoritmos** no arquivo **Cap5exem5.4PnGregoryNewton1D.m**. A seguir, apresentamos algumas considerações sobre os interpoladores estudados até este momento.

O número de operações aritméticas executadas em cada tipo de interpolador é:

- a) Interpolador geral de base canônica: $(n - 1)n$ para gerar o sistema, $O(2n^3/3)$ para solvê-lo e $2n$ operações para obter $f(v) \cong P_n(v)$.
- b) Interpolador de Lagrange (otimizado): $2n^2 + 9n$ operações para obter $f(v) \cong P_n(v)$.
- c) Interpolador de Gregory-Newton: $3(n^2 + n)/2$ operações para obter as diferenças divididas e $4n$ operações para obter $f(v) \cong P_n(v)$.

Cada método será mais eficiente de acordo com a aplicação e o número de valores a serem estimados. Por exemplo, devido às características das expressões algébricas, Gregory-Newton é mais eficiente quando temos que efetuar muitas estimativas em um mesmo conjunto de pontos, pois as diferenças divididas podem ser calculadas previamente e reutilizadas quantas vezes forem necessárias. Já o método de Lagrange é mais eficiente quando temos de efetuar estimativas em várias funções discretas com os mesmos valores independentes x , de modo que os produtórios que envolvem apenas x , e que acompanham os valores dependentes y_i , possam ser calculados previamente e reutilizados.

No método de Gregory-Newton é possível acrescentar um ponto qualquer no final de um conjunto de pontos discretos existente, mesmo que a sequência de pontos fique desordenada, e avaliar o novo interpolador correspondente incluindo mais uma parcela de diferenças divididas. Por exemplo, para acrescentar o 5º ponto (9,5) à função discretizada do Exemplo 5.3, procedemos desta forma:

i	x_i	y_i	$\Delta^1 y_i$	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$
1	1	2	-1	2/3	-1/8	3/160
2	3	0	1	-1/12	1/40	-
3	4	1	2/3	1/15	-	-
4	7	3	1	-	-	-
5	9	5	-	-	-	-

Assim, acrescentamos o termo $\Delta^4 y_1(x-x_1)(x-x_2)(x-x_3)(x-x_4)$ ao interpolador $P_3(x)$ existente, gerando o $P_4(x)$ a seguir:

$$P_4(x) = 2 + (-1)(x-1) + (2/3)(x-1)(x-3) + (-1/8)(x-1)(x-3)(x-4) + (3/160)(x-1)(x-3)(x-4)(x-7)$$

No método de Lagrange, também podemos acrescentar um ponto qualquer a um conjunto de pontos existente utilizando o método de Neville, que faz uso de relações de recorrência para avaliar o novo interpolador (BURDEN; FAIRES, 2011).

Para (x_i, y_i) ($i = 1, 2, \dots, n + 1$) com $x_{i+1} > x_i$ e espaçamento constante ($x_{i+1} - x_i = h, \forall i$), os cálculos das diferenças divididas $\Delta^k y_i$ podem ser simplificados. Nesse caso, definimos as chamadas diferenças finitas $\bar{\Delta}^k y_i$, no sentido ascendente, por:

$$\bar{\Delta} y_i = y_{i+1} - y_i \Rightarrow \text{diferença de 1ª ordem.}$$

$$\bar{\Delta}^k y_i = \bar{\Delta}^{k-1} y_{i+1} - \bar{\Delta}^{k-1} y_i \Rightarrow \text{diferença de } k\text{-ésima ordem.}$$

Logo, a relação das diferenças finitas com as diferenças divididas é a seguinte:

$$\Delta^k y_i = \frac{\bar{\Delta}^k y_i}{h^k k!}$$

Essa expressão substituída na eq. (9a) resulta no seguinte interpolador:

$$P_n(x) = y_1 + \sum_{k=1}^n \frac{\bar{\Delta}^k y_1}{h^k k!} \left[\prod_{j=1}^k (x - x_j) \right] \quad (10)$$

Assim, nas estimativas de valores $f(\beta) \cong P_n(\beta)$:

- a) Se $\beta \in [a, b]$, temos uma **interpolação**.
- b) Se $\beta \notin [a, b]$, temos uma **extrapolação**.

Caso queiramos obter o valor de $x = \beta = ?$, correspondente ao valor $y = \gamma, f(\beta) = \gamma$, teremos uma **interpolação inversa**. Para efetuá-la, podemos:

- a) Se os pontos discretos forem de uma função injetora, isto é, $\forall x_i \neq x_j \Rightarrow y_i \neq y_j$, inverter as variáveis, tornando os y_i independentes e os x_i dependentes; efetuar a interpolação em (y_i, x_i) , obtendo o polinômio $P_n(y)$, e a estimativa nesse polinômio em $y = \gamma$ gerará $\beta = P_n(\gamma)$.
- b) Se os pontos discretos não forem de uma função injetora, obter o interpolador $P_n(x)$; igualar esse polinômio ao γ , resultando numa equação polinomial $P_n(x) = \gamma$; resolver essa equação $P_n(x) - \gamma = 0$ conforme vimos no Capítulo 3; e desconsiderar as $n - 1$ soluções espúrias.

Exemplo 5.5: na função, a seguir, representamos o consumo de energia elétrica, em anos, de um determinado local. Determine quando ($t = ?$) o consumo C atingirá o limite instalado de 7.5 MW.

Tempo t (ano)	85	89	93	95	96	...	?
Consumo C (MW)	5	5.7	6.2	6.7	7.0	...	7.5

Solução:

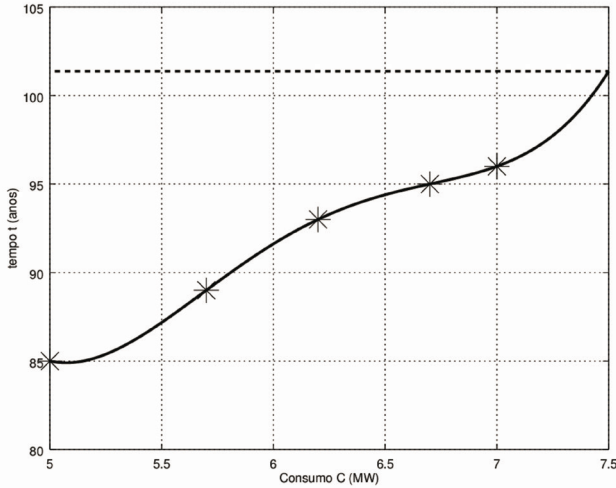
Neste exemplo, temos um problema de extrapolação (para fora dos limites de dados disponíveis).

Como os dados são de uma função injetora, se invertermos as variáveis, ela resultará na função:

Consumo C (MW)	5	5.7	6.2	6.7	7.0	...	7.5
Tempo t (ano)	85	89	93	95	96	...	?

No Gráfico 5.5a, temos a expressão de tempo em decorrência do consumo.

Gráfico 5.5a – Comportamento do tempo t (ano) em função do Consumo C (MW)



Fonte: Elaboração própria.

Obtendo o interpolador de Gregory-Newton para o tempo (t) como função do Consumo C (KW), temos:

$$t_4(C) = 85 + (5.7143)(C - 5) + (1.9048)(C - 5)(C - 5.7) + (-3.4734)(C - 5)(C - 5.7)(C - 6.2) + (2.9546)(C - 5)(C - 5.7)(C - 6.2)(C - 6.7)$$

Logo, se Consumo $C = 7.50\text{MW}$, então tempo $t = 101.37$ (calculado via algoritmo de Gregory-Newton). Então depois do quarto mês de 2001, o consumo deverá atingir 7.5MW .

Alternativamente, podemos também obter diretamente uma expressão para o Consumo C em função do tempo t :

$$P_4(t) = 7.5, t = ?$$

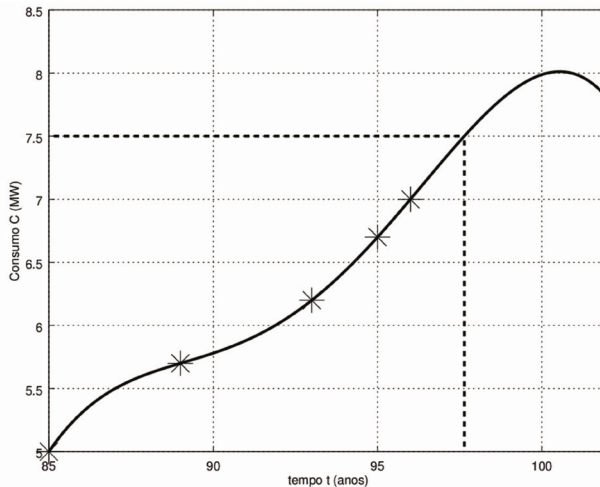
Tempo t (ano)	85	89	93	95	96	...	?
Consumo C (MW)	5	5.7	6.2	6.7	7.0	...	7.5

Como $n + 1 = 5$ pontos, obtemos o interpolador de grau $n = 4$ aqui expresso na forma geral:

$$C_4(t) = -3.00325e-04 * t^4 + 1.11426e-01 * t^3 - 1.54789e+01 * t^2 \\ + 9.54391e+02 * t - 2.20353e+04$$

Fazendo $C_4(t) = 7.5$, obtemos $t = 97.64$ via Método de Newton, excluindo soluções expúrias. Assim, depois do 7º mês de 1997, o consumo deverá atingir 7.5MW.

Gráfico 5.5b – Comportamento do consumo de energia C (MW) em função do tempo t (ano)



Fonte: Elaboração própria.

Observe que, em casos de extrapolação (avaliações fora do intervalo $[a, b]$), podemos ter comportamentos diferentes daquele observado no intervalo de medições $[a, b]$. No Gráfico 5.5a, observamos um crescimento acentuado do tempo a partir do consumo 7.0MW e, no Gráfico 5.5b, observamos uma inversão inesperada de consumo a partir do ano 2000 ($t = 100$). Logo, o resultado mais confiável deve ser o mais conservador, com o tempo de 97.64 anos para atingir o consumo 7.5MW. Esse é um exemplo típico de aplicação em um [problema de previsão por extrapolação](#)⁹.



No Capítulo 7, abordaremos uma metodologia mais adequada para esse tipo de problema.

Por fim, a aproximação por interpolação polinomial também pode ser estendida a funções com várias variáveis independentes, como veremos na seção 5.2.

5.1.3 Avaliação do erro de truncamento na interpolação

Quando aproximamos uma função $y = f(x)$ com expressão conhecida por um interpolador $P_n(x)$ e $x \in [x_1, x_{n+1}]$, cometemos um erro de truncamento em cada $x \neq x_i$ a ser estimado definido por:

$$\text{Erro } P_n(x) = |P_n(x) - f(x)|, \quad \forall x \in [x_1, x_{n+1}] \tag{11}$$

A relação entre a aproximanda $f(x)$ e sua interpoladora $P_n(x)$ pode ser observada no exemplo do Gráfico 5.2, enquanto o respectivo erro pode ser visto no Gráfico 5.3.

A delimitação do erro de truncamento $\text{Erro } P_n(x = \bar{x})$, para qualquer $\bar{x} \in [x_1, x_{n+1}]$, indicará o grau de confiança dos resultados fornecidos pela aproximação $P_n(\bar{x}) \cong f(\bar{x})$.


Podemos obter o erro de truncamento do interpolador polinomial $P_n(x)$ antes de determiná-lo via:

Teorema 1: se $P_n(x)$ é o interpolador de

i	1	2	...	$n + 1$
x_i	x_1	x_2	...	x_{n+1}
$y_i = f(x_i)$	y_1	y_2	...	y_{n+1}

com $f(x)$ continuamente diferenciável em $[x_1, x_{n+1}]$, então $\forall \bar{x} \in [x_1, x_{n+1}]$,

$$\exists \xi \in (x_1, x_{n+1}) / \text{Erro } P_n(\bar{x}) = |P_n(\bar{x}) - f(\bar{x})| = \left| \frac{f^{(n+1)}(\xi) \prod_{i=1}^{n+1} (\bar{x} - x_i)}{(n+1)!} \right|$$

 $f^{(n+1)}(x)$ refere-se à derivada de ordem $(n + 1)$ de $f(x)$. A demonstração da expressão do erro contida no **Teorema 1** e das demais fórmulas simplificadas na sequência podem ser encontradas na obra *Cálculo numérico: aspectos teóricos e computacionais*, de Ruggiero e Lopes, 2ª edição, 1997.

A determinação do erro de truncamento usando a expressão do **Teorema 1** é muito difícil, pois:

- pode ser complicado, ou impossível, obter $f^{(n+1)}(x)$ (derivada $n+1$ -ésima de $f(x)$);
- não sabemos qual é o valor de ξ , apenas conhecemos a sua região de localização; e
- para cada valor \bar{x} a ser estimado, temos que reavaliar o *Erro* $P_n(\bar{x})$.

Embora o **Teorema 1** seja de difícil aplicação, tem grande valia devido aos dois próximos corolários.

Corolário 1: sob as hipóteses do **Teorema 1**, se $M = \max_{x \in [x_1, x_{n+1}]} |f^{(n+1)}(x)|$, então $\text{Erro } P_n(\bar{x}) \leq \frac{M}{(n+1)!} \left| \prod_{i=1}^{n+1} (\bar{x} - x_i) \right|$ em um ponto específico $x = \bar{x}$.

Trata-se do majorante do **erro local**, em um ponto específico $x = \bar{x}$ (limite superior do erro em $x = \bar{x}$).

Exemplo 5.6: delimite o erro de truncamento cometido em $\bar{x} = 0.65$ ao aproximar $f(x) = \ln(x)$ por um interpolador polinomial, considerando $x \in [0.4, 0.7]$ e $n = 3$ subdivisões.

Solução:

Para $n = 3$, então $h = (0.7 - 0.4)/3 = 0.1$

i	1	2	3	4
x_i	0.4	0.5	0.6	0.7
$y_i = f(x_i)$	-0.916290732	-0.693147181	-0.510825624	-0.356674944

Aplicando o **Corolário 1**, temos:

$$f(x) = \ln(x) \Rightarrow f'(x) = 1/x \Rightarrow f''(x) = -1/x^2 \Rightarrow f'''(x) = 2/x^3 \Rightarrow f^{(4)}(x) = -6/x^4$$

$f^{(4)}(x)$ é uma função de módulo decrescente em $[0.4, 0.7]$, e o seu máximo local é $M = |f^{(4)}(0.4)| = 234.375$. Então,

$$\text{Erro } P_n(0.65) \leq \frac{234.375}{(3+1)!} \left| \prod_{i=0}^3 (0.65 - x_i) \right| = \frac{234.375}{(3+1)!} |(0.65 - x_1)(0.65 - x_2)(0.65 - x_3)(0.65 - x_4)|$$

$$\text{Erro } P_n(0.65) \leq \frac{234.375}{(3+1)!} |(0.65 - 0.4)(0.65 - 0.5)(0.65 - 0.6)(0.65 - 0.7)| \\ \leq 9.15 \cdot 10^{-4}$$

Comparando com o valor exato de $\ln(x)$, temos:

$$f(\bar{x} = 0.65) = \ln(0.65) = -0.430782916$$

$$P_3(\bar{x} = 0.65) = -0.431019619 \text{ por Lagrange.}$$

$$\text{Erro exato } P_n(\bar{x}) = |P_n(\bar{x}) - f(\bar{x})| = 2.36703 \cdot 10^{-4}.$$

Em $\bar{x} = 0.65$, vemos que o erro exato $2.36703 \cdot 10^{-4}$ é menor do que o erro de truncamento máximo dado pelo **Corolário 1**, $\text{Erro } P_n(0.65) \leq 9.15 \cdot 10^{-4}$, como era esperado. Observe que $\bar{x} = 0.65$ é um ponto médio de um dos intervalos e deve ser um dos pontos de maior erro de truncamento de todo intervalo $[a, b]$.

Seria interessante ter o majorante do erro global em todo o intervalo $[a, b]$ e não apenas em um ponto específico $x = \bar{x}$. Assim, temos o limite superior global do erro no intervalo $[a, b]$ dado pelo **Corolário 2**.

Corolário 2: se na forma discretizada da aproximanda ocorrer $x_{i+1} > x_i$ (pontos ordenados) e $x_{i+1} - x_i = h$ (igualmente espaçados) $\forall i$, então

$$\text{Erro } P_n(x) < \frac{M h^{n+1}}{4(n+1)} \text{ com } M = \max_{x \in [x_1, x_{n+1}]} |f^{(n+1)}(x)| \text{ em todo intervalo } [x_1, x_{n+1}].$$

Trata-se do majorante do erro global, em todo x do intervalo.

Exemplo 5.7: delimite o erro de truncamento máximo no intervalo $x \in [0.4, 0.7]$ ao aproximar $f(x) = \ln(x)$ por um interpolador polinomial e com $n = 3$ subdivisões. Considere os x_i ordenados e igualmente espaçados. Avalie o erro exato em $x = 0.65$ e compare se ele fica abaixo do erro máximo do intervalo $[a, b]$.

Solução:

$$n = 3 \quad \Rightarrow \quad h = \frac{0.7 - 0.4}{3} = 0.1$$

$$\begin{aligned} f(x) = \ln(x) &\Rightarrow f'(x) = 1/x \Rightarrow f''(x) = -1/x^2 \Rightarrow f'''(x) = 2/x^3 \Rightarrow \\ f^{(4)}(x) &= -6/x^4 \end{aligned}$$

$$M = \max_{x \in [0.4, 0.7]} |f^{(4)}(x)| = \max | -6/x^4 | = 6/(0.4)^4 = 234.375$$

Então, o erro máximo de todo o intervalo é:

$$\text{Erro } P_n(x) < \left(\frac{234.375(0.1)^4}{4 * 4} \right) = 0.00146484375$$

O erro de truncamento máximo de todo o intervalo, dado pelo **Coroário 2**, $\text{Erro } P_3(x) < 0.00146484375$, é sempre maior do que o erro máximo em algum ponto do intervalo, como em $x = 0.65$, $\text{Erro } P_3(0.65) < 9.15 * 10^{-4}$, do **Exemplo 5.6**.

Exemplo 5.8: obtenha o erro de truncamento máximo cometido ao aproximarmos $f(x) = e^x$ e $x \in [2, 2.4]$ via $P_n(x)$ com os x_i ordenados e igualmente espaçados em $n = 4$ intervalos. Avalie o erro exato em $x = 2.33$ e compare se ele fica abaixo do erro máximo do intervalo.

Solução:

$$n = 4 \Rightarrow h = \frac{2.4 - 2}{4} = 0.1$$

$$\text{Temos } f(x) = e^x \Rightarrow f'(x) = e^x \Rightarrow f''(x) = e^x \Rightarrow f'''(x) = e^x \Rightarrow \\ f^{(4)}(x) = e^x \Rightarrow f^{(5)}(x) = e^x$$

$$M = \max_{x \in [2, 2.4]} |f^{(5)}(x)| = |f^{(5)}(2.4)| = e^{2.4}$$

$$\text{Então, o erro máximo } \textit{Erro } P_n(x) < \left(\frac{e^{2.4}(0.1)^5}{4 * 5} \right) = 5.51159 * 10^{-6}$$

Verificação:

$$P_4(2.33) = 10.2779431277041$$

$$e^{2.33} = 10.2779415330434$$

$$\textit{Erro exato}(2.33) = |10.2779431277041 - 10.2779415330434| = 1.59466 * 10^{-6}.$$

Observe que o erro exato em $x = 2.33$ fica abaixo do erro máximo: $1.59466 * 10^{-6} < 5.51159 * 10^{-6}$, como era esperado.

Exemplo 5.9: calcule o grau n mínimo do interpolador polinomial $P_n(x)$ necessário para que o erro máximo entre o interpolador $P_n(x)$ e $f(x) = \ln(x)$ seja menor do que $1 * 10^{-6}$, em todo intervalo $x \in [1, 2]$, dividindo-o em n partes iguais.

Solução:

Uma possibilidade é usar o **Corolário 2** e estabelecer tentativas com valores de n (inteiros) para calcular o valor do erro máximo de truncamento, facilitando a obtenção das derivadas $f^{(n+1)}(x)$:

Primera tentativa: $n = 3 \Rightarrow h = \frac{2-1}{3} = 0.333333333\dots$

$$f(x) = \ln(x) \Rightarrow f'(x) = 1/x \Rightarrow f''(x) = -1/x^2 \Rightarrow f'''(x) = 2!/x^3 \Rightarrow f^{(4)}(x) = -3!/x^4$$

$$f^{(n+1)}(x) = (-1)^n n! x^{-(n+1)}$$

$$M = \max_{x \in [1, 2]} |f^{(4)}(x)| = 3! 1^{-4} = 6$$

Com *Erro* $P_n(x) < \left(\frac{6 (0.33333333)^4}{4 * 4} \right) \leq 4.6496 * 10^{-3}$, precisamos aumentar o n para baixar esse limite do erro, então vamos tentar $n = 6$.

Segunda tentativa: $n = 6 \Rightarrow h = \frac{2-1}{6} = 0.16666666\dots$

$$M = \max_{x \in [1, 2]} |f^{(7)}(x)| = 6! 1^{-7} = 720$$

Com *Erro* $P_n(x) < \left(\frac{720 (0.16666667)^7}{4 * 7} \right) \leq 9.1858 * 10^{-05}$, ainda precisamos aumentar o n , então vamos tentar $n = 8$.

Terceira tentativa: $n = 8 \Rightarrow h = \frac{2-1}{8} = 0.125$

$$M = \max_{x \in [1, 2]} |f^{(9)}(x)| = 8! 1^{-9} = 40320$$

Com *Erro* $P_n(x) < \left(\frac{40320 (0.125)^9}{4 * 9} \right) \leq 8.3447 * 10^{-06}$, também precisamos aumentar o n , por isso vamos tentar $n = 10$.

Quarta tentativa: $n = 10 \Rightarrow h = \frac{2-1}{10} = 0.1$

Temos:

$$M = \max_{x \in [1, 2]} |f^{(11)}(x)| = 10! 1^{-11} = 3628800$$

Então, $Erro P_{10}(x) < \left(\frac{3628800 (0.1)^{11}}{4 * 11} \right) = 8.2473 * 10^{-07}$ é menor do que 10^{-6} .

Logo, podemos usar grau $n = 10$ para calcular o polinômio interpolador $P_n(x)$ representativo de $f(x) = \ln(x)$ em $x \in [1, 2]$, e o erro máximo de truncamento não ultrapassará 10^{-6} . Com $n = 9$, esse erro máximo de truncamento será $2.6018 * 10^{-06}$, maior do que $1 * 10^{-6}$, mas ainda é da ordem de $O(10^{-6})$.

Outra possibilidade é calcular o valor do *erro exato* de truncamento em um número grande de pontos do intervalo e tomar o seu valor máximo, uma vez que a função exata $f(x) = \ln(x)$ está disponível para comparação através de várias bibliotecas de funções e em calculadoras. Também podemos estabelecer um algoritmo de busca do menor grau n enquanto o erro máximo exato estiver acima de $1 * 10^{-6}$, calculando o erro para graus n crescentes do polinômio interpolador. Nesse caso, concluímos que com $n = 7$ os erros em todo o intervalo $x \in [1, 2]$ são menores do que $8.7302 * 10^{-07}$, conforme o arquivo `Cap5exem5.9_busca_menor_n.m` do *Caderno de Algoritmos*.

5.2 INTERPOLAÇÃO DE FUNÇÕES COM VÁRIAS VARIÁVEIS INDEPENDENTES

Até agora efetuamos interpolações de funções com uma única variável independente, unidimensionais; entretanto, na modelagem matemática da maioria dos fenômenos, ocorre a influência de várias variáveis independentes sobre o valor observado. São aplicações típicas a modelagem de superfícies de uma área geográfica, a partir das cotas (alturas) nos pontos de uma grade retangular, para fins de cálculos topográficos, a modelagem em computação gráfica, entre outras.

Por exemplo, como estimar o valor de $w = f(u, v)$ quando a função aproximanda depende de duas variáveis independentes x e y ?

Seja uma função com duas variáveis independentes a ser aproximada por um polinômio interpolador:

$$\begin{cases} f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \\ (x, y) \rightarrow z = f(x, y) \end{cases} \text{ com } x \in [a, b] \text{ e } y \in [c, d] \quad (\text{domínio retangular}).$$

Para determinar o interpolador, acompanhe o passo a passo a seguir.

Primeiro passo: dividimos $[a, b]$ em n partes gerando $(n + 1) x_i$ e $[c, d]$ em m partes gerando $(m + 1) y_j$.

Segundo passo: obtemos $(n + 1)(m + 1)$ valores amostrais $z_{ij} = f(x_i, y_j)$ da aproximanda, conforme a matriz a seguir:

$x_i \backslash y_j$	x_1	x_2	x_3	...	$x = u$...	x_{n+1}
y_1	z_{11}	z_{21}	z_{31}	...	↓	...	$z_{(n+1)1}$
y_2	z_{12}	z_{22}	z_{32}	...	↓	...	$z_{(n+1)2}$
⋮	⋮	⋮	⋮		↓		⋮
$y = v$	→	→	→	...	$f(u, v) = ?$...	⋮
⋮	⋮	⋮	⋮				⋮
y_{m+1}	$z_{1(m+1)}$	$z_{2(m+1)}$	$z_{3(m+1)}$	$z_{(n+1)(m+1)}$

Terceiro passo: fixamos uma das variáveis (por convenção será fixada o x).

Quarto passo: para cada coluna de x_i fixada, $i = 1$ a $n + 1$, separamos os pontos:

y_j	y_1	y_2	y_3	...	v	...	y_{m+1}
z_{ij}	z_{i1}	z_{i2}	z_{i3}		$t_i = PY_m(y = v)$		$z_{i(m+1)}$

Interpolando em $y = v$, geramos um $t_i = PY_m(y = v)$ através de seu interpolador simples de grau m , $PY_m(y)$.

Quinto passo: do passo anterior resulta o conjunto t_i em função de x_i :

x_i	x_1	x_2	x_3	...	$x = u$...	x_{n+1}
t_i	t_1	t_2	t_3		$PX_n(x = u)$		$t_{(n+1)}$

Agora, obtemos o seu interpolador de grau n , $PX_n(x)$ e estimamos $f(u, v) \cong PX_n(x = u)$.

Observe que a estimativa via interpolação de um único $f(u, v)$ envolve $(n + 1) + 1 = (n + 2)$ interpolações simples. Podemos estender esse mesmo raciocínio para três, quatro, ..., k variáveis independentes. Por exemplo, a estimativa de $f(u, v, w)$, em três dimensões, envolve $(n + 1)$ interpolações duplas e uma simples no final $\Rightarrow (n + 1)(n + 2) + 1 = n^2 + 3n + 3$ interpolações simples. Logo, a interpolação múltipla provoca um crescimento exponencial no volume de operações a serem executadas. Por isso, é muito importante usar um interpolador simples envolvendo o menor número de operações aritméticas possível.

Com o **Exemplo 5.10**, vamos apresentar um caso de interpolação bidimensional via Gregory-Newton.

Exemplo 5.10: monte um algoritmo que aproxime a $f(x, y) = x^4 + y^4$ em $u = 8.7$ e $v = 5.5$ via interpolação dupla com Gregory-Newton, dados 5 valores de $x \in [8.5, 9.6]$ e 6 valores de $y \in [5.0, 7.5]$ (na forma de vetor do *Octave*):

x_i	8.5	8.9	9.3	9.5	9.6
y_j					
5.0					
5.7					
6.2					
6.7					
7.0					
7.5					

O algoritmo solicitado no **Exemplo 5.10** está disponível no arquivo **Cap5interpolacao 2D.m** do **Caderno de Algoritmos**.

Essa forma multidimensional de interpolação pode ser estendida através de outras bases dos polinômios interpoladores, como de Lagrange, em cada direção (RICE, 1983). Por Lagrange, devemos definir os valores de $z_{ij} = f(x_i, y_j)$ e obter o valor interpolado em qualquer x e y diretamente por meio de polinômio duplo $P_{nm}(x, y)$, de grau n em x e m em y , conforme a equação a seguir,

$$P_{nm}(x, y) = \sum_{i=1}^{n+1} \sum_{j=1}^{m+1} z_{ij} L_i(x) L_j(y)$$

Em que,

$$L_i(x) = \prod_{\substack{k=1 \\ k \neq i}}^{n+1} \frac{(x - x_k)}{(x_i - x_k)}, \quad i = 1, \dots, n+1$$

$$L_j(y) = \prod_{\substack{k=1 \\ k \neq j}}^{m+1} \frac{(y - y_k)}{(y_j - y_k)}, \quad j = 1, \dots, m+1$$

Os polinômios de Lagrange $L_i(x)$ e $L_j(y)$ geram fatores peso fixos para um conjunto bidimensional fixo de pontos, ou seja, para uma mesma malha de pontos discretos (x_i, y_j) e uma mesma posição (x, y) de interpolação, o que torna essa forma de interpolação mais eficiente para conjuntos de pontos fixos em que tenhamos apenas diferentes valores de z_{ij} .

Agora confira o algoritmo do **Exemplo 5.10** com interpolação bidimensional via polinômios de Lagrange no arquivo **Cap5interpolacao2D.m**.

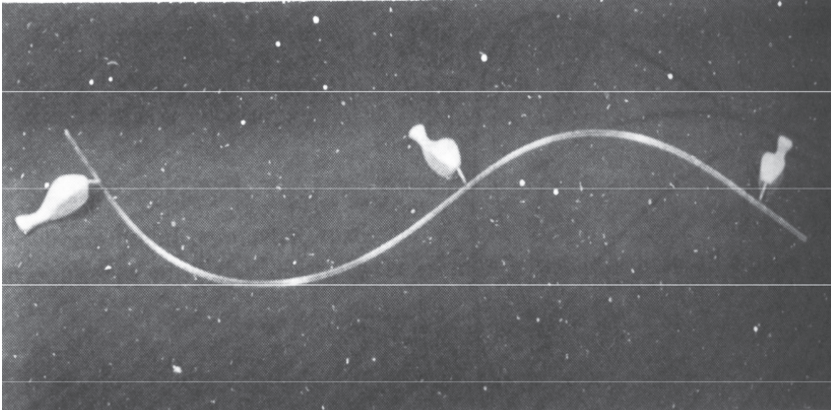
A interpolação de Lagrange bidimensional também só é válida para um domínio retangular, $x \in [a, b]$ e $y \in [c, d]$, ou seja, não é válida para domínios de pontos aleatoriamente distribuídos no espaço. Neste último caso, podemos aplicar uma interpolação com erro de 2ª ordem conhecida como **pseudolaplaciano**, proposta por Holmes e Connel (1989). Nesse método, calculamos um valor interpolado usando apenas alguns valores discretizados conhecidos, baseado no cálculo adequado de fatores peso aplicado a cada valor discretizado, de tal forma que as aproximações geradas são exatas para funções lineares. Essa metodologia por ser aplicada para domínios multidimensionais.

Lembramos que a função aproximadora $z = g(x)$ pode pertencer a outras famílias de funções, como vimos no início deste capítulo, não se restringindo apenas às polinomiais, como o exemplo no **Exercício 4.5** do Capítulo 4. O importante é que a condição de aproximação seja mantida, ou seja, os **erros ou desvios locais**, sobre os pontos escolhidos para definir a aproximadora, **devem ser nulos**.

Na próxima seção, vamos apresentar alternativas de aproximação mais adequadas para desenhar curvas.

5.3 APROXIMAÇÃO POR INTERPOLAÇÃO *SPLINE*

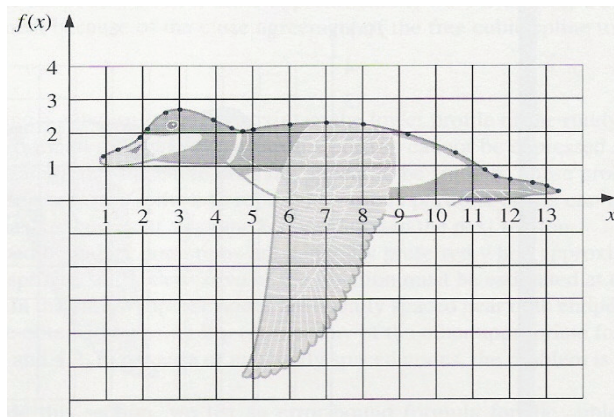
Originalmente, *splines* eram régua flexíveis, de madeira ou plástico, que podiam ser curvadas de forma a passar por um dado conjunto de pontos (x_i, y_i) chamados de “nós”. Essas *splines* originais usavam pesos (*ducks*) que eram fixados nas áreas de interesse causando a deformação da estrutura de madeira ou plástico de acordo com a curva desejada (Figura 5.1). Foram muito utilizadas em desenhos de engenharia nos tempos em que não havia recursos computacionais. Apesar de serem usadas desde o século XIX, somente no fim da década de 1960 foi desenvolvida uma formulação matemática desse problema. Tal formalização possibilitou o desenvolvimento de vários sistemas computadorizados que utilizam aproximações gráficas de funções como os conhecidos CAD/CAM.

Figura 5.1 – Spline física original e pesos (*ducks*)

Fonte: Instituto de Computação UFF (2016).

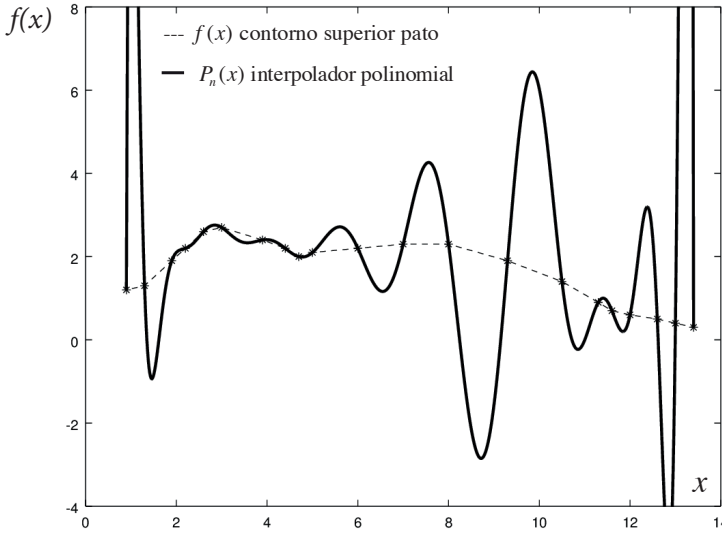
As maiores dificuldades relacionadas à interpolação polinomial convencional (Lagrange, Gregory-Newton etc.) ocorrem ou quando tomamos um polinômio interpolante de grau pequeno, que gera elevados erros de truncamento, ou quando tomamos um polinômio interpolador de grau elevado, que tende a ter gráficos sinuosos. Veja a representação do contorno superior de um pato voando do Gráfico 5.6, e o mesmo contorno usando interpolação polinomial no Gráfico 5.7.

Gráfico 5.6 – Representação do contorno superior de um pato voando



Fonte: Burden e Faires (2011).

Gráfico 5.7 – Contorno superior do pato voando representado por 21 pontos, conectados e destacados com marcador *, e interpolador polinomial de grau $n = 20$, em linha contínua



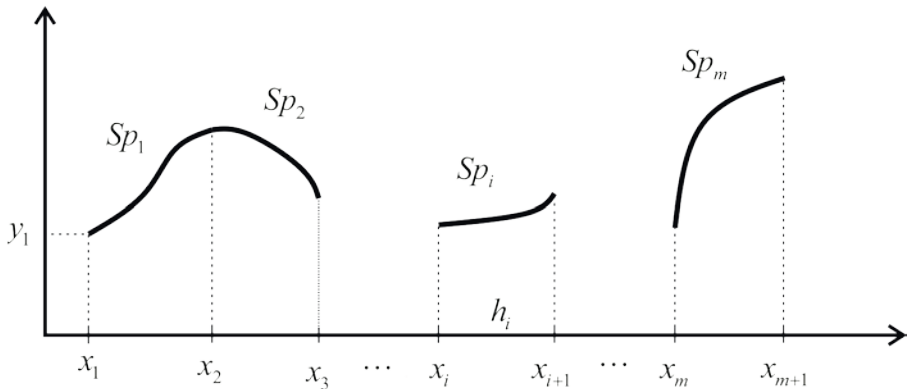
Fonte: Elaboração própria.

Nas aplicações em que a aproximanda muda de comportamento no domínio de interesse, como a do Gráfico 5.6, a aproximadora deve também acompanhar essas mudanças e efetuar as junções de modo mais suave possível, evitando as sinuosidades como as da interpoladora do Gráfico 5.7.

Uma técnica de aproximação possível consiste em dividir o intervalo de interesse em vários subintervalos e aproximá-los separadamente com polinômio de grau pequeno, efetuando as junções destes polinômios da forma mais suave possível.

Definição 3: **interpolação *spline*** é uma técnica de aproximação que consiste em dividir o domínio $[a, b]$ de interesse e interpolar separadamente cada subintervalo, efetuando as junções entre os interpoladores da forma mais suave possível, conforme o Gráfico 5.8.

Gráfico 5.8 – Splines genéricas aplicadas em cada subintervalo do domínio



Fonte: Elaboração própria.

Por conveniência teórica e prática, e por questões de otimização, são utilizados interpoladores polinomiais cúbicos $Sp_i(x)$, com grau 3 fixo (*splines* cúbicas). Essa técnica e suas variantes constituem um dos fundamentos da computação gráfica.

Para aproximar um caminho funcional $y = f(x)$, $x \in [a, b]$ por *splines* cúbicas, procedemos da seguinte maneira:

Primeiro passo: dividimos $[a, b]$ em m subintervalos convenientes $[x_i, x_{i+1}]$ tal que $x_{i+1} > x_i$ e comprimentos $h_i = x_{i+1} - x_i$, para $i = 1, 2, \dots, m$, e geramos $m + 1$ pontos amostrais (x_i, y_i) do caminho.

Segundo passo: obtemos um polinômio aproximador $Sp_i(x)$ de grau 3 para cada um dos m subintervalos $[x_i, x_{i+1}]$, expresso na forma:

$$Sp_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i \text{ para } i = 1, 2, \dots, m \quad (12)$$

satisfazendo as seguintes condições de aproximação:

- $Sp_i(x_i) = y_i$, para todo ponto $i = 1, 2, \dots, m$ e $Sp_m(x_{m+1}) = y_{m+1}$.
- $Sp_{i-1}(x_i) = Sp_i(x_i) = y_i$, para todo ponto interno $i = 2, \dots, m \Rightarrow$ Condição de continuidade.

Note que cada polinômio $Sp_i(x)$, relativo ao intervalo $[x_i, x_{i+1}]$, deve passar pelos seus dois pontos extremos (x_i, y_i) e (x_{i+1}, y_{i+1}) . Logo, em cada ponto x_p , os valores dos dois polinômios que nele incidem são iguais.

- c) $Sp'_{i-1}(x_i) = Sp'_i(x_i)$, para todo ponto $x_i, i = 2, \dots, m \Rightarrow$ **Condição de suavidade.**

Ou seja, em cada ponto x_p , a inclinação dos dois polinômios que nele incidem são iguais.

- d) $Sp''_{i-1}(x_i) = Sp''_i(x_i)$, para todo ponto $x_p, i = 2, \dots, m \Rightarrow$ **Velocidade de encurvamento.**

Ou seja, em cada ponto interno, a velocidade de encurvamento dos dois polinômios que nele incidem são iguais.

Num *spline mecânico*, conforme a Figura 5.1, essas propriedades correspondem a:

- a) a *spline* deve passar sobre os “nós” (x_i, y_i) ;
- b) a *spline* não quebra ou não forma ângulos agudos; e
- c) a *spline* assume a forma que minimiza a energia potencial.

Terceiro passo: estimamos $f(u)$ primeiro localizando o subintervalo que contenha o u . Definido que $u \in [x_p, x_{i+1}] \Rightarrow f(u) \cong Sp_i(u)$, conforme o Gráfico 5.8.

Para definir os $Sp_i(x)$ de cada subintervalo $[x_p, x_{i+1}]$, com $i = 1, 2, \dots, m$, expressos conforme a eq. (12), calculamos a primeira e segunda derivadas de $Sp_i(x)$:

$$Sp'_i(x) = 3a_i(x - x_i)^2 + 2b_i(x - x_i) + c_i \quad (13)$$

$$Sp''_i(x) = 6a_i(x - x_i) + 2b_i \quad (14)$$

renomeamos as derivadas de segunda ordem da eq. (14):

$$S_i = Sp_i''(x_i) \text{ e}$$

$$S_{i+1} = Sp_{i+1}''(x_{i+1}) = Sp_i''(x_{i+1}).$$

e aplicamos as condições estabelecidas nos itens (a), (b), (c) e (d) do **segundo passo**:

a) Condição (a) aplicada na eq. (12):

$$Sp_i(x_i) = d_i = y_i \quad (15)$$

$$Sp_i(x_{i+1}) = a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i = y_{i+1} \quad (16)$$

b) Condição (d) aplicada na eq. (14):

$$Sp_i''(x_i) = 2b_i = S_i \Rightarrow b_i = S_i / 2 \quad (17)$$

$$Sp_i''(x_{i+1}) = 6a_i h_i + 2b_i = S_{i+1} \quad (18)$$

Substituindo a eq. (17) na eq. (18), temos:

$$a_i = \frac{(S_{i+1} - S_i)}{6h_i} \quad (19)$$

E substituindo as eqs. (19), (17) e (15) na eq. (16), temos:

$$\frac{(S_{i+1} - S_i)}{6h_i} h_i^3 + \frac{S_i}{2} h_i^2 + c_i h_i + y_i = y_{i+1}$$

$$c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{S_{i+1} h_i + 2S_i h_i}{6} \quad (20)$$

Agrupando os coeficientes, temos:

$$\begin{cases} a_i = (S_{i+1} - S_i) / (6h_i) \\ b_i = S_i / 2 \\ c_i = (y_{i+1} - y_i) / h_i - (S_{i+1} + 2S_i)h_i / 6 \\ d_i = y_i \end{cases} \quad \forall i = 1, 2, \dots, m \quad (21)$$

Assim, temos os coeficientes das *splines* cúbicas em função dos valores das curvaturas S_i e S_{i+1} nas extremidade de cada subintervalo. Para obter esses valores, calculamos $Sp'(x_i)$ e $Sp'_{i-1}(x_i)$ através da eq. (13) aplicada em x_i :

$$Sp'_i(x_i) = c_i$$

$$Sp'_{i-1}(x_i) = 3a_{i-1}h_{i-1}^2 + 2b_{i-1}h_{i-1} + c_{i-1}$$

e utilizamos a condição (c) do **segundo passo**, $Sp'_i(x_i) = Sp'_{i-1}(x_i)$, resultando em:

$$c_i = 3a_{i-1}h_{i-1}^2 + 2b_{i-1}h_{i-1} + c_{i-1} \quad (22)$$

Substituindo as expressões da eq. (21) na eq. (22), temos:

$$h_{i-1}S_{i-1} + 2(h_{i-1} + h_i)S_i + h_iS_{i+1} = 6[(y_{i+1} - y_i) / h_i - (y_i - y_{i-1}) / h_{i-1}] \quad (23)$$

para $i = 2, \dots, m$ ($m - 1$ equações), resultando no seguinte sistema de equações lineares não quadrado:

$$\begin{bmatrix} h_1 & 2(h_1 + h_2) & h_2 & & & & \\ & h_2 & 2(h_2 + h_3) & h_3 & & & \\ & & & \ddots & & & \\ & & & & \ddots & & \\ & & & & & h_{m-1} & 2(h_{m-1} + h_m) & h_m \\ & & & & & & & S_{m+1} \end{bmatrix} \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ \vdots \\ S_m \\ S_{m+1} \end{bmatrix} = 6 \begin{bmatrix} \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\ \vdots \\ \vdots \\ \frac{y_{m+1} - y_m}{h_m} - \frac{y_m - y_{m-1}}{h_{m-1}} \end{bmatrix} \quad (24)$$

Como temos $m + 1$ pontos e conseqüentemente $m + 1$ incógnitas S_i e apenas $m - 1$ equações ($i = 2, 3, \dots, m$) na eq. (24), então temos um sistema possível, mas indeterminado. Para que tenhamos solução única, isto é, sem ambigüidades, precisamos impor duas condições especiais (duas equações) diretamente nos pontos extremos de $[a, b]$ envolvendo S_1 e S_{m+1} . Dependendo de tais condições, podemos ter vários tipos de *splines* cúbicas:

- a) Supondo que o caminho tende a ser **linear nos extremos**, o que equivale a prescrever duas equações adicionais, $S_1 = 0$ e $S_{m+1} = 0$, com soluções explícitas para S_1 e S_{m+1} , que, substituídas diretamente na eq. (24), geram o sistema tridiagonal a seguir:

$$\begin{bmatrix} 2(h_1 + h_2) & h_2 & & & & \\ h_2 & 2(h_2 + h_3) & h_3 & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & h_{m-1} & 2(h_{m-1} + h_m) \end{bmatrix} \cdot \begin{bmatrix} S_2 \\ S_3 \\ \vdots \\ \vdots \\ S_m \end{bmatrix} = 6 \begin{bmatrix} \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\ \vdots \\ \vdots \\ \frac{y_{m+1} - y_m}{h_m} - \frac{y_m - y_{m-1}}{h_{m-1}} \end{bmatrix} \quad (25)$$

- b) Supondo que o caminho tende a ser de **forma quadrática** nos extremos, o que equivale a prescrever duas equações adicionais, $S_1 = S_2$ e $S_{m+1} = S_m$, que também podem ser substituídas diretamente na eq. (24), gerando o sistema linear tridiagonal a seguir:

$$\begin{bmatrix} (3h_1 + 2h_2) & h_2 & & & & \\ h_2 & 2(h_2 + h_3) & h_3 & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ & & & & h_{m-1} & (2h_{m-1} + 3h_m) \end{bmatrix} \cdot \begin{bmatrix} S_2 \\ S_3 \\ \vdots \\ \vdots \\ S_m \end{bmatrix} = 6 \begin{bmatrix} \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\ \vdots \\ \vdots \\ \frac{y_{m+1} - y_m}{h_m} - \frac{y_m - y_{m-1}}{h_{m-1}} \end{bmatrix} \quad (26)$$

- c) Quando os valores das curvaturas nos **extremos** são **conhecidos** previamente, isto é, $S_1 = u$ e $S_{m+1} = v$, basta substituí-los na eq. (24), gerando o sistema tridiagonal a seguir:

$$\begin{bmatrix} 2(h_1 + h_2) & h_2 & & & & & & \\ h_2 & 2(h_2 + h_3) & h_3 & & & & & \\ & & \ddots & & & & & \\ & & & \ddots & & & & \\ & & & & \ddots & & & \\ & & & & & h_{m-1} & 2(h_{m-1} + h_m) & \\ & & & & & & & \end{bmatrix} \begin{bmatrix} S_2 \\ S_3 \\ \vdots \\ \vdots \\ S_m \end{bmatrix} = 6 \begin{bmatrix} -h_1 u + \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \frac{y_4 - y_3}{h_3} - \frac{y_3 - y_2}{h_2} \\ \vdots \\ \vdots \\ -h_m v + \frac{y_{m+1} - y_m}{h_m} - \frac{y_m - y_{m-1}}{h_{m-1}} \end{bmatrix} \quad (27)$$

- d) Se assumirmos valores de curvatura para os extremos S_1 e S_{m+1} obtidos por **extrapolação linear** de S_2 e S_3 , e de S_{m-1} e S_m , respectivamente, teremos também duas equações adicionais:

$$\frac{S_2 - S_1}{h_1} = \frac{S_3 - S_2}{h_2} \quad \text{e} \quad \frac{S_{m+1} - S_m}{h_m} = \frac{S_m - S_{m-1}}{h_{m-1}}$$

Que também devem ser adicionadas ao sistema geral dado pela eq. (24), mas agora gerando um sistema com lei de formação diferenciada para as duas equações adicionais:

$$\text{Para } i=1 \quad \Rightarrow \quad h_2 S_1 - (h_1 + h_2) S_2 + h_1 S_3 = 0$$

$$\text{Para } i=m+1 \quad \Rightarrow \quad h_m S_{m-1} - (h_{m-1} + h_m) S_m + h_{m-1} S_{m+1} = 0$$

Assim, gerando um sistema de $m + 1$ equações para resolver diretamente as $m + 1$ incógnitas com dois coeficientes extras à matriz tridiagonal (destacados em negro), temos:

Note que o sistema linear dado pela eq. (28) não é mais tridiagonal devido às duas equações adicionais.

Em todos os casos, depois de resolvidos os sistemas, temos uma solução única para $S_1, S_2, S_3, \dots, S_{m+1}$, já incluída a respectiva condição de extremos S_1 e S_{m+1} . Substituindo os valores de S_i nas eqs. (21), obtemos os coeficientes a_p, b_p, c_p, d_p , das m *splines* cúbicas no intervalo $[a, b]$, definida pela eq. (12).

Exemplo 5.11: aproxime a função discretizada, a seguir, por *splines* cúbicas dos 4 tipos apresentados, em $[0, 4]$, com 4 subintervalos $h = 1$, e estime $f(x = 2.5)$.

i	1	2	3	4	5
x_i	0	1	2	3	4
y_i	-3	-2	5	24	61

(esse conjunto de pontos provém da $f(x) = x^3 - 3$).

Solução:

Como, $h_i = 1$ para $i = 1, 2, \dots, 4$ ($m = 4$ intervalos e $m + 1 = 5$ pontos):

a) Com *splines* na forma linear nos extremos, conforme a eq. (25):

$$\begin{bmatrix} 2(h_1 + h_1) & h_2 & 0 \\ h_2 & 2(h_2 + h_3) & h_3 \\ 0 & h_3 & 2(h_3 + h_4) \end{bmatrix} * \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = 6 \begin{bmatrix} (y_3 - y_2) / h_2 - (y_2 - y_1) / h_1 \\ (y_4 - y_3) / h_3 - (y_3 - y_2) / h_2 \\ (y_5 - y_4) / h_4 - (y_4 - y_3) / h_3 \end{bmatrix}$$

Calculando os valores, obtemos o sistema:

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix} * \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 36 \\ 72 \\ 108 \end{bmatrix}$$

Sendo a solução $S = \{S_2, S_3, S_4\} = \{6.42857, 10.28571, 24.42857\}$, com $S_1 = S_5 = 0$.

Assim, para o primeiro intervalo, $x \in [0, 1]$, $i = 1$, temos:

$$\begin{cases} a_1 = (S_2 - S_1) / (6h_1) = 1.07143 \\ b_1 = S_1 / 2 = 0 \\ c_1 = (y_2 - y_1) / h_1 - (S_2 + 2S_1)h_1 / 6 = -0.071429 \\ d_1 = y_1 = -3 \end{cases}$$

$$Sp_1(x) = 1.07143(x-0)^3 + 0(x-0)^2 - 0.071429(x-0) - 3, x \in [0, 1]$$

As demais *splines* são as seguintes:

$$Sp_2(x) = 0.64286(x-1)^3 + 3.21429(x-1)^2 + 3.142857(x-0) - 2, x \in [1, 2]$$

$$Sp_3(x) = 2.35714(x-2)^3 + 5.14286(x-2)^2 + 11.50000(x-2) + 5, x \in [2, 3]$$

$$Sp_4(x) = -4.07143(x-3)^3 + 12.21429(x-3)^2 + 28.857143(x-3) + 24, x \in [3, 4]$$

Podemos obter uma aproximação de $f(x = 2.5)$ via $Sp_3(x = 2.5) = 12.330$.

b) Com *splines* de extremos quadráticos, conforme a eq. (26):

$$\begin{bmatrix} (3h_1 + 2h_2) & h_2 & 0 \\ h_2 & 2(h_2 + h_3) & h_3 \\ 0 & h_3 & (2h_3 + 3h_4) \end{bmatrix} * \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = 6 \begin{bmatrix} (y_3 - y_2) / h_2 - (y_2 - y_1) / h_1 \\ (y_4 - y_3) / h_3 - (y_3 - y_2) / h_2 \\ (y_5 - y_4) / h_4 - (y_4 - y_3) / h_3 \end{bmatrix}$$

Calculando os valores, obtemos o sistema:

$$\begin{bmatrix} 5 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 5 \end{bmatrix} * \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 36 \\ 72 \\ 108 \end{bmatrix}$$

Sendo a solução $S = \{S_2, S_3, S_4\} = \{4.8, 12, 19.2\}$, com $S_1 = S_2 = 4.8$ e com $S_{m+1} = S_m = 19.2$ ($m = 4$).

Assim,

$$Sp_1(x) = 0(x - 0)^3 + 2.4(x - 0)^2 - 1.4(x - 0) - 3 \text{ para } x \in [0, 1]$$

$$Sp_2(x) = 1.2(x - 1)^3 + 2.4(x - 1)^2 + 3.4(x - 0) - 2, \text{ para } x \in [1, 2]$$

$$Sp_3(x) = 1.2(x - 2)^3 + 6(x - 2)^2 + 11.8(x - 2) + 5, \text{ para } x \in [2, 3]$$

$$Sp_4(x) = 0(x - 3)^3 + 9.6(x - 3)^2 + 27.4(x - 3) + 24, \text{ para } x \in [3, 4]$$

Podemos obter uma aproximação de $f(x = 2.5)$ via $Sp_3(x = 2.5) = 12.550$.

c) Com *splines* de extremos conhecidos, conforme a eq. (27):

Por exemplo: $S_1 = u = 0$ e $S_2 = v = 24$ (que são os valores exatos de $f(x) = x^3 - 3$ discretizada).

$$\begin{bmatrix} 2(h_1 + h_1) & h_2 & 0 \\ h_2 & 2(h_2 + h_3) & h_3 \\ 0 & h_3 & 2(h_3 + h_4) \end{bmatrix} * \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = 6 \begin{bmatrix} -h_1 u + (y_3 - y_2) / h_2 - (y_2 - y_1) / h_1 \\ (y_4 - y_3) / h_3 - (y_3 - y_2) / h_2 \\ -h_m v + (y_5 - y_4) / h_4 - (y_4 - y_3) / h_3 \end{bmatrix}$$

Calculando os valores, obtemos o sistema:

$$\begin{bmatrix} 4 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 4 \end{bmatrix} * \begin{bmatrix} S_2 \\ S_3 \\ S_4 \end{bmatrix} = \begin{bmatrix} 36 \\ 72 \\ 84 \end{bmatrix}$$

Sendo a solução $S = \{S_1, S_2, S_3, S_4, S_5\} = \{0, 6.0, 12.0, 18.0, 24.0\}$

Assim,

$$Sp_1(x) = 1(x - 0)^3 + 0(x - 0)^2 + 0(x - 0) - 3 \text{ para } x \in [0, 1]$$

$$Sp_2(x) = 1(x - 1)^3 + 3(x - 1)^2 + 3(x - 0) - 2, \text{ para } x \in [1, 2]$$

$$Sp_3(x) = 1(x - 2)^3 + 6(x - 2)^2 + 12(x - 2) + 24, \text{ para } x \in [2, 3]$$

$$Sp_4(x) = 1(x - 3)^3 + 9(x - 3)^2 + 27(x - 3) + 24, \text{ para } x \in [3, 4]$$

Podemos obter uma aproximação de $f(x = 2.5)$ via $Sp_3(x = 2.5) = 12.625$.

- d) Com *splines* de extremos extrapolados a partir das duas curvaturas internas vizinhas, conforme a eq. (28):

$$\begin{bmatrix} h_2 & -(h_1+h_2) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & h_{m-1} & 2(h_{m-1}+h_m) & h_m & \\ & & h_m & -(h_{m-1}+h_m) & h_{m-1} & \end{bmatrix} * \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{bmatrix} = 6 \begin{bmatrix} 0 \\ (y_3-y_2)/h_2 - (y_2-y_1)/h_1 \\ (y_4-y_3)/h_3 - (y_3-y_2)/h_2 \\ (y_5-y_4)/h_4 - (y_4-y_3)/h_3 \\ 0 \end{bmatrix}$$

Observamos que esse sistema é quase tridiagonal, exigindo apenas duas alterações no algoritmo (ambos coeficientes adicionais destacados em negrito):

$$\begin{bmatrix} 1 & -2 & \mathbf{I} & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & 1 & \\ & & 1 & 4 & 1 \\ & & \mathbf{I} & -2 & 1 \end{bmatrix} * \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ S_4 \\ S_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 36 \\ 72 \\ 108 \\ 0 \end{bmatrix}$$

Sendo a solução:

$$S = \{S_1, S_2, S_3, S_4, S_5\} = \{-4.6957, 3.9130, 12.5217, 18.0000, 23.4783\}.$$

Assim,

$$Sp_1(x) = 1.43478(x-0)^3 - 2.3478(x-0)^2 + 1.9130(x-0) - 3 \text{ para } x \in [0, 1]$$

$$Sp_2(x) = 1.43478(x-1)^3 + 1.9565(x-1)^2 + 3.6087(x-0) - 2, \text{ para } x \in [1, 2]$$

$$Sp_3(x) = 0.91304(x-2)^3 + 6.2609(x-2)^2 + 11.8261(x-2) + 5, \text{ para } x \in [2, 3]$$

$$Sp_4(x) = 0.91304(x-3)^3 + 9.0000(x-3)^2 + 27.0870(x-3) + 24, \text{ para } x \in [3, 4]$$

Podemos obter uma aproximação de $f(x = 2.5)$ via $Sp_3(x = 2.5) = 12.592$.

Para efeito de comparação, temos que a $f(x)$ discretizada corresponde a $f(x) = x^3 - 3$, cujo valor exato de $f(x = 2.5) = 12.625$. Assim,

- a) com *spline* de extremos lineares, temos $Sp_3(2.5) = 12.330$;
- b) com *spline* de extremos quadráticos, temos $Sp_3(2.5) = 12.550$;
- c) com *spline* de curvatura de extremos definidos, temos $Sp_3(2.5) = 12.625$ (atribuídos valores exatos nos extremos); e
- d) com *spline* de curvatura extrapolada nos extremos, temos $Sp_3(2.5) = 12.592$.

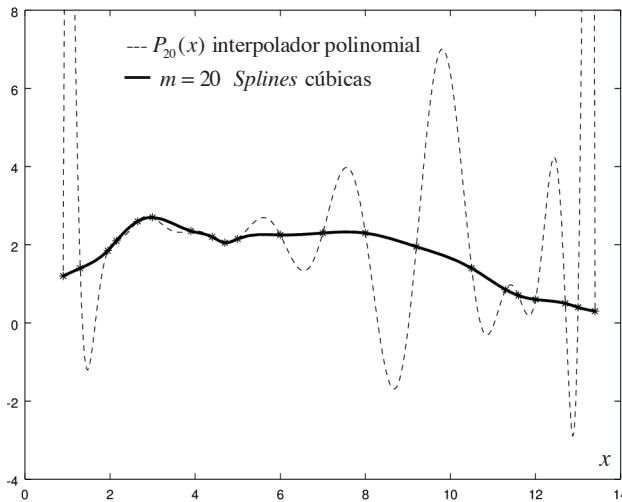
Logo, para esse exemplo com curvatura conhecida e exata nos extremos, a *spline* aproximadora e a função aproximanda $f(x)$ são iguais. Genericamente, sempre que temos disponíveis os extremos S, é recomendável que eles sejam usados, conforme a eq. (27).

Considerando um caso geral, quando as curvaturas nas extremidades não são conhecidas previamente, a interpolação com *spline* de curvatura extrapolada nos extremos gera um resultado mais realístico, com menor erro de truncamento.

O algoritmo de interpolação por *splines* cúbicas para os quatro tipos de extremidades – linear, quadrático, de extremos definidos e extremos extrapolados – está disponível no **Caderno de Algoritmos** no arquivo **Cap5Graf5.10Splines.seno.m**.

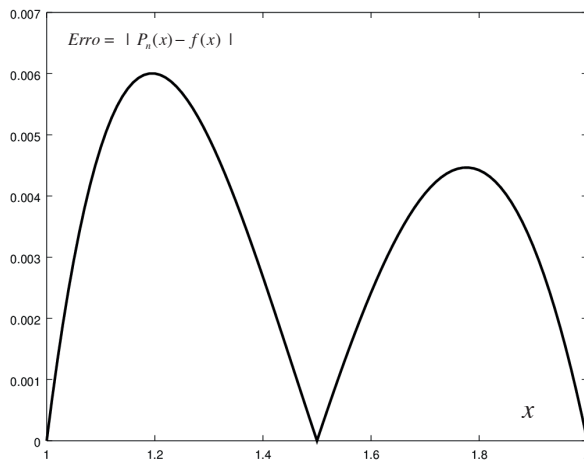
Apresentamos, nos Gráficos 5.9 e 5.10, dois exemplos de aplicação de *splines*: a representação do contorno superior do pato voando do Gráfico 5.6 e a da função $sen(x)$, respectivamente.

Gráfico 5.9 – Curva do Gráfico 5.7 representado por meio de $m = 20$ *splines* cúbicas com extremos quadráticos, em linha contínua, e interpolador $P_{20}(x)$, em linha tracejada



Fonte: Elaboração própria.

Gráfico 5.10 – Função $f(x) = \text{sen}(x)$ em $x \in [-\pi, +\pi]$ representada por meio de $m = 17$ *splines* cúbicas com extremos quadráticos, em linha contínua, e $P_{17}(x)$, em linha tracejada



Fonte: Elaboração própria.

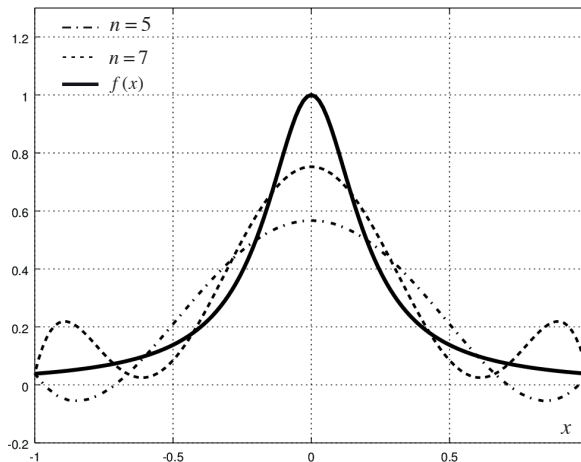
Observe que as *splines* cúbicas formaram representações suaves nos Gráficos 5.9 e 5.10, gerando curvas com derivadas e curvaturas suaves. No Gráfico 5.10, a interpolação polinomial também gerou resultados

suaves (linha tracejada, não visível), mas, no Gráfico 5.9, a interpolação gerou sinuosidades muito acentuadas, enquanto as *splines* geraram curvas suaves.

5.4 TRATAMENTO DO FENÔMENO DE RUNGE

No campo específico da análise numérica, o **fenômeno de Runge** é um problema de oscilação em funções interpoladoras polinomiais que ocorre quando usamos interpolação com polinômios de grau elevado, como mostra o Gráfico 5.11. Carl Runge descobriu esse fenômeno quando investigava erros na interpolação polinomial para aproximar certas funções, como a função assintótica $f(x) = 1 / (25 * x^2 + 1)$.

Gráfico 5.11 – Representação do fenômeno de Runge



Fonte: Fenômeno de Runge (2013).

Observe que a curva em linha contínua é a função exata $f(x) = 1 / (25 * x^2 + 1)$; a tracejada é uma interpolação polinomial de grau $n = 7$, com 8 pontos de interpolação igualmente espaçados; e a curva em traço-ponto é uma interpolação polinomial de 5º grau, com 6 pontos de interpolação igualmente espaçados.

Quando usamos $m = n + 1$ “nós” equidistantes na interpolação polinomial de grau n , o erro máximo aumenta quando a ordem do polinômio interpolador aumenta, em princípio, contrariando o teorema de Weierstrass, ou seja, as sinuosidades aumentam com o grau n do interpolador, como

demonstram os Gráficos 5.11 e 5.12. Podemos amenizar essas sinuosidades usando n *splines* cúbicas ou interpolação polinomial de grau n sobre $m = n + 1$ “nós” dos *polinômios Tchebychev de grau m* ⁹, em vez de m “nós” equidistantes, para fixar o polinômio interpolador de grau n ($n = m - 1$).



No Capítulo 6, apresentaremos as propriedades dos polinômios Tchebychev e seus “nós” (raízes dos polinômios de Tchebychev), que, mostraremos, estão mais concentrados nas extremidades do intervalo, de modo que o polinômio interpolador fica mais “amarrado” à $f(x)$, gerando menos sinuosidades.

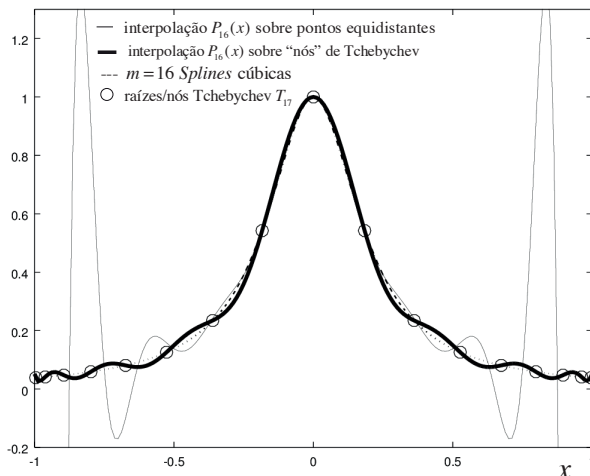
No **Exemplo 5.12**, vamos aplicar *splines* cúbicas e interpolações polinomiais à função $f(x) = 1 / (25 * x^2 + 1)$ visando reduzir os erros das aproximações.

Exemplo 5.12: aproxime a função $f(x) = 1 / (25 * x^2 + 1)$ por meio da interpolação polinomial $P_{16}(x)$ com $n + 1 = 17$ “nós” (pontos) equidistantes no intervalo normalizado $[-1, +1]$, por meio de interpolação polinomial $P_{16}(x)$ com $m = 17$ pontos definidos pelos “nós” (raízes) dos polinômios de Tchebychev T_{17} de grau $m = 17$ e por meio de $n = 16$ *splines* cúbicas. Apresente os resultados em um gráfico e calcule os erros máximos em cada caso.

Solução:

Obtida via algoritmo **Cap5Graf5.12FenomenoRunge.m** disponível no **Caderno de Algoritmos**:

Gráfico 5.12 – Representação das aproximações do Exemplo 5.12



Fonte: Elaboração própria.

Observe que a interpolação polinomial com pontos x definidos pelos “nós” do polinômio de Tchebychev reduz gradativamente as sinuosidades com o aumento do grau n (linha contínua espessa), mas as *splines* cúbicas conseguem a melhor representação (linha tracejada), que fica realmente muito próxima da função exata (linha não apresentada no gráfico), pois apresenta o menor erro de truncamento máximo:

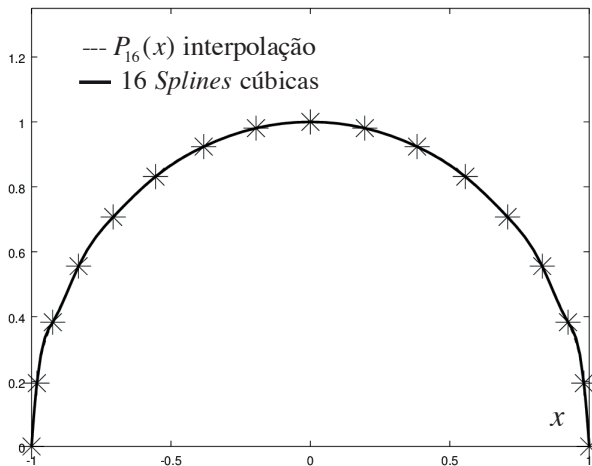
Erro máximo do polinômio com m “nós” (pontos) equidistantes é 14.370

Erro máximo do polinômio com m “nós” (pontos) de Tchebychev é 0.032603

Erro máximo de $m - 1$ *splines* é 0.0037367

As *splines* são essencialmente funções polinomiais (interpoladores), aplicadas em cada subintervalo, então podem apresentar deformações em regiões onde a curva não se comporta como “função”, por exemplo, quando uma curva fica muito vertical, como no semicírculo apresentado no Gráfico 5.13.

Gráfico 5.13 – *Spline* representando um semicírculo



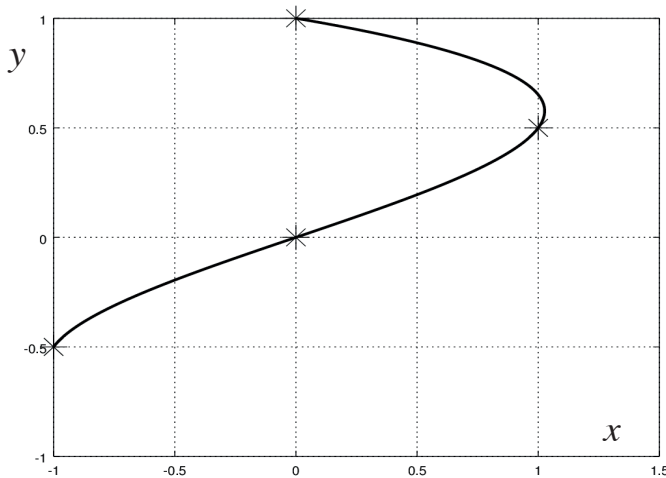
Fonte: Elaboração própria.

Nesses casos, as curvas propostas por Pierre Bézier são mais adequadas, porque uma função $y = f(x)$, ou mesmo uma relação não funcional, é separada em duas funções, de modo parametrizado, com $x = x(t)$ e $y = y(t)$, em que t é um parâmetro independente. Assim, $x = x(t)$ e $y = y(t)$ sempre serão funções bem comportadas.

5.5 APROXIMAÇÃO DE NÃO FUNÇÕES VIA PARAMETRIZAÇÃO

As técnicas de aproximação que abordamos até este momento possibilitam o traçado de caminhos que sejam funções, mas se o caminho, ou a curva, for não funcional, como o esboçado no Gráfico 5.14, precisaremos parametrizá-lo como duas funções.

Gráfico 5.14 – Curva não funcional



Fonte: Elaboração própria.

Para aproximar um caminho que seja não função, efetuamos os passos:

Primeiro passo: escolhemos $m + 1$ pontos de referência do caminho

x_i	x_1	x_2	...	x_{m+1}
y_i	y_1	y_2	...	y_{m+1}

Segundo passo: adotamos um novo parâmetro independente $t \in [0, 1]$, tomamos $m + 1$ valores ordenados t_i , em que $t_1 = 0$, $t_{i+1} = t_i + h$, $h = (1 - 0) / m$, com $i = 1, 2, \dots, m$, e geramos dois conjuntos de pontos, representando-os como duas funções discretizadas:

t_i	t_1	t_2	...	t_{m+1}
x_i	x_1	x_2	...	x_{m+1}

t_i	t_1	t_2	...	t_{m+1}
y_i	y_1	y_2	...	y_{m+1}

No Gráfico 5.14, tomando a relação não funcional:

x_i	0	1	0	-1
y_i	1	1/2	0	-1/2

E, parametrizando-a, resulta em:

t_i	0	1/3	2/3	3/3
$x(t_i)$	0	1	0	-1

t_i	0	1/3	2/3	3/3
$y(t_i)$	1	1/2	0	-1/2

Note que essas duas novas amostras de pontos agora são funções com variável independente t , isto é, cada par (x, y) do caminho é reescrito como função de t : $(x, y) = (x(t), y(t))$.

Terceiro passo: obtemos, pela técnica mais adequada ao problema, as duas aproximadoras de $x(t)$ e $y(t)$. Por exemplo, aplicando a interpolação de Gregory-Newton nos dois conjuntos de quatro pontos anteriores, resultam estes dois interpoladores:

$$x(t) \cong PX_3(t) = 0 + 3(t - 0) - 3(t - 0)(3t - 1) + (t - 0)(3t - 1)(3t - 2)$$

$$y(t) \cong PY_3(t) = 1 - (3/2)(t - 0)$$

Quarto passo: plotamos os pontos de interesse $p_k = (PX_n(t_k), PY_n(t_k))$, $\forall t_k \in [0, 1]$ para gerar o caminho desejado. Para a função do Gráfico 5.14, usando os interpoladores do passo anterior, temos, por exemplo, para $t_k = 1/2$, o ponto $p_k = (0.625, 0.25)$.

Todos os algoritmos das técnicas de aproximação de funções por interpolação polinomial e por *splines* cúbicas podem ser estendidos para aproximação de não funções via parametrização.

5.6 APROXIMAÇÃO POR CURVAS DE BÉZIER

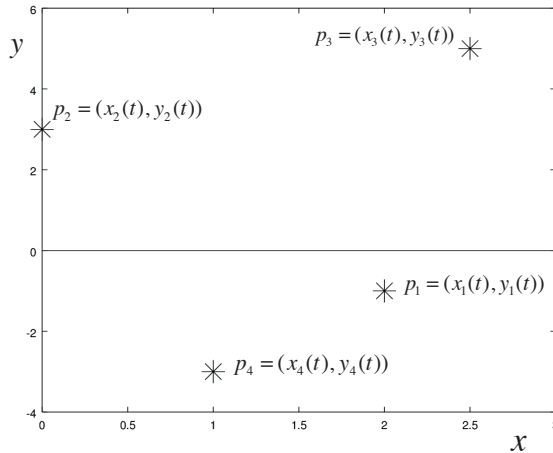
A curva de Bézier é a base matemática de sistemas de computação gráfica como o CorelDRAW® (CURVA DE BÉZIER, 2016). Foi desenvolvida pelo engenheiro Pierre Bézier, na década de 1960, então funcionário da Renault®, para simulação de *layout* de automóveis.

A aproximação por curvas de Bézier foi concebida e algebrizada para representar (aproximar) caminhos que não sejam funções, não sendo necessária a parametrização prévia como nos casos de interpolação e de *splines*, pois já é uma aproximação parametrizada na sua origem.

Para traçar um caminho definido a partir dos pontos do Gráfico 5.15, por exemplo, usando a aproximação de Bézier, devemos efetuar os passos a seguir.

Primeiro passo: tomamos $m + 1$ pontos referenciais do desenho desejado $\Rightarrow p_i = (x_i, y_i)$, $i = 1, \dots, m + 1$ e os expressamos na forma parametrizada $\Rightarrow p_i = (x_i(t), y_i(t))$, em que $t \in [0, 1]$, conforme o Gráfico 5.15:

Gráfico 5.15 – Pontos na forma parametrizada



Fonte: Elaboração própria.

Segundo passo: subdividimos os $m + 1$ pontos referenciais em subconjuntos com $(k + 1)$ pontos, $2 \leq k < m + 1$ (nos casos de termos mais do que quatro pontos). Cada um desses subconjuntos irá representar determinado comportamento do caminho.

Terceiro passo: para cada subconjunto de pontos $p_i = (x_i, y_i)$, $i = 1, \dots, k + 1$, obtemos o seu polinômio aproximador de grau k expresso na forma de Bernstein:

$$B_k(t) = \sum_{i=0}^k C_k^i (1-t)^{k-i} t^i p_{i+1} \Rightarrow \begin{cases} BX_k(t) = \sum_{i=0}^k C_k^i (1-t)^{k-i} t^i x_{i+1} \\ BY_k(t) = \sum_{i=0}^k C_k^i (1-t)^{k-i} t^i y_{i+1} \end{cases} \quad (29a)$$

Em que $C_k^i = \frac{k!}{(k-i)!i!}$ são combinações e as eqs. (29a) são os polinômios de grau k resultantes.

Expressando as eqs. (29a) na sua forma matricial equivalente, temos:

$$B_k(t) = \begin{bmatrix} C_k^0 p_1 & C_k^1 p_2 & \dots & C_k^k p_{k+1} \end{bmatrix}^* \begin{bmatrix} (1-t)^k \\ (1-t)^{k-1} t^1 \\ \vdots \\ t^k \end{bmatrix}$$

$$\left\{ \begin{array}{l} BX_k(t) = \begin{bmatrix} C_k^0 x_1 & C_k^1 x_2 & \dots & C_k^k x_{k+1} \end{bmatrix}^* \begin{bmatrix} (1-t)^k \\ (1-t)^{k-1} t^1 \\ \vdots \\ t^k \end{bmatrix} \\ BY_k(t) = \begin{bmatrix} C_k^0 y_1 & C_k^1 y_2 & \dots & C_k^k y_{k+1} \end{bmatrix}^* \begin{bmatrix} (1-t)^k \\ (1-t)^{k-1} t^1 \\ \vdots \\ t^k \end{bmatrix} \end{array} \right. \quad (29b)$$

De acordo com o grau do polinômio de Bernstein, podemos gerar vários tipos de curvas de Bézier:

$k = 2 \Rightarrow$ **Bézier quadrática (3 pontos referenciais):**

$$B_2(t) = 1(1-t)^{(2-0)} t^0 p_1 + 2(1-t)^{(2-1)} t^1 p_2 + 1(1-t)^{(2-2)} t^2 p_3 \quad (30)$$

que é equivalente a

$$\begin{cases} BX_2(t) = 1(1-t)^{(2-0)} t^0 x_1 + 2(1-t)^{(2-1)} t^1 x_2 + 1(1-t)^{(2-2)} t^2 x_3 \\ BY_2(t) = 1(1-t)^{(2-0)} t^0 y_1 + 2(1-t)^{(2-1)} t^1 y_2 + 1(1-t)^{(2-2)} t^2 y_3 \end{cases}$$

Desenvolvendo para aplicação computacional, temos:

$$\begin{cases} xx(t) = x_1 + t(bx + t \cdot ax) \\ yy(t) = y_1 + t(by + t \cdot ay) \end{cases}, \quad \forall t \in [0, 1] \quad (31)$$

Em que

$$\begin{cases} bx = 2(x_2 - x_1) \\ ax = x_1 - 2x_2 + x_3 \\ by = 2(y_2 - y_1) \\ ay = y_1 - 2y_2 + y_3 \end{cases}$$

$k = 3 \Rightarrow$ **Bézier cúbica** (4 pontos referenciais como no Gráfico 5.15):

$$B_3(t) = 1(1-t)^{(3-0)}t^0 p_1 + 3(1-t)^{(3-1)}t^1 p_2 + 3(1-t)^{(3-2)}t^2 p_3 + 1(1-t)^{(3-3)}t^3 p_4 \quad (32)$$

que é equivalente a

$$\begin{cases} BX_3(t) = 1(1-t)^{(3-0)}t^0 x_1 + 3(1-t)^{(3-1)}t^1 x_2 + 3(1-t)^{(3-2)}t^2 x_3 + 1(1-t)^{(3-3)}t^3 x_4 \\ BY_3(t) = 1(1-t)^{(3-0)}t^0 y_1 + 3(1-t)^{(3-1)}t^1 y_2 + 3(1-t)^{(3-2)}t^2 y_3 + 1(1-t)^{(3-3)}t^3 y_4 \end{cases}$$

Desenvolvendo para aplicação computacional, temos:

$$\begin{cases} xx(t) = x_1 + t(cx + t(bx + t \cdot ax)) \\ yy(t) = y_1 + t(cy + t(by + t \cdot ay)) \end{cases} \quad \forall t \in [0, 1] \quad (33)$$

Em que

$$\begin{cases} cx = 3(x_2 - x_1) \\ bx = 3(x_3 - x_2) - cx \\ ax = (x_4 - x_1) - (cx + bx) \end{cases}$$

$$\begin{cases} cy = 3(y_2 - y_1) \\ by = 3(y_3 - y_2) - cy \\ ay = (y_4 - y_1) - (cy + by) \end{cases}$$

A plotagem de uma polinomial $B_k(t)$, com $t \in [0, 1]$, gera uma curva com as seguintes condições de aproximação:

Propriedade 1:

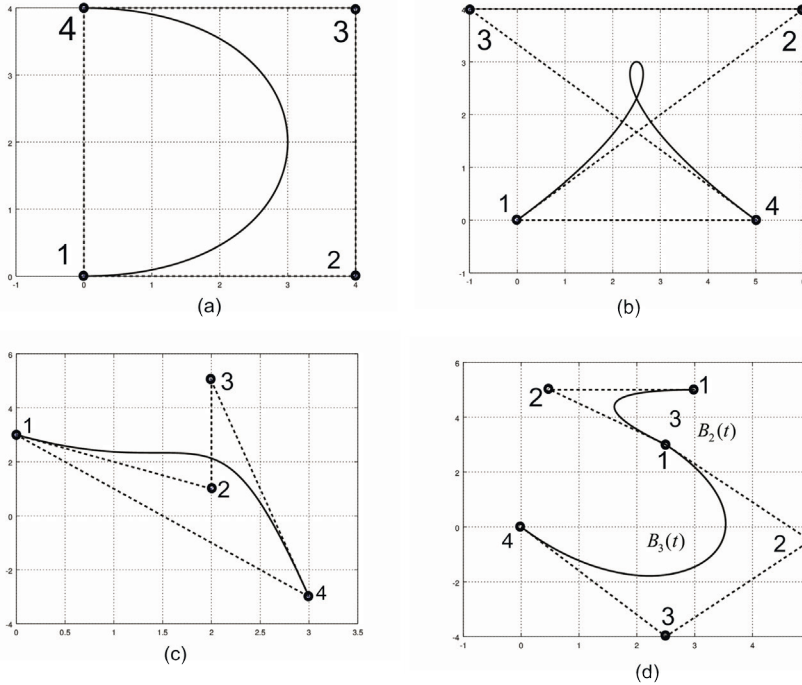
$$\begin{cases} BX_k(0) = x_1 \\ BY_k(0) = y_1 \end{cases} \text{ e } \begin{cases} BX_k(1) = x_{k+1} \\ BY_k(1) = y_{k+1} \end{cases} \text{ isto é, } B_k(t) \text{ inicia em } p_1 \text{ e termina em } p_{k+1}.$$

Propriedade 2: a reta definida pelos pontos p_1 e p_2 é tangente a $B_k(t)$ em p_1 , e a reta definida por p_k e p_{k+1} é tangente a $B_k(t)$ em p_{k+1} . Portanto, a curva de Bézier passa somente sobre os pontos p_1 e p_{k+1} e usa o(s) ponto(s) intermediário(s) como controle para definir as inclinações no início e no final do segmento.

Propriedade 3: a curva gerada pela sequência de pontos $\{BX_k(t), BY_k(t)\}$, $t \in [0, 1]$, está contida sempre no menor polígono convexo que contém os $k + 1$ pontos de referência da $B_k(t)$.

No Gráfico 5.16, apresentamos 4 curvas de Bézier, sendo que em (a), (b) e (c) temos curvas $B_3(t)$ e em (d) temos uma composição de curvas $B_2(t)$ e $B_3(t)$.

Gráfico 5.16 – Sequência de curvas de Bézier



Fonte: Elaboração própria.

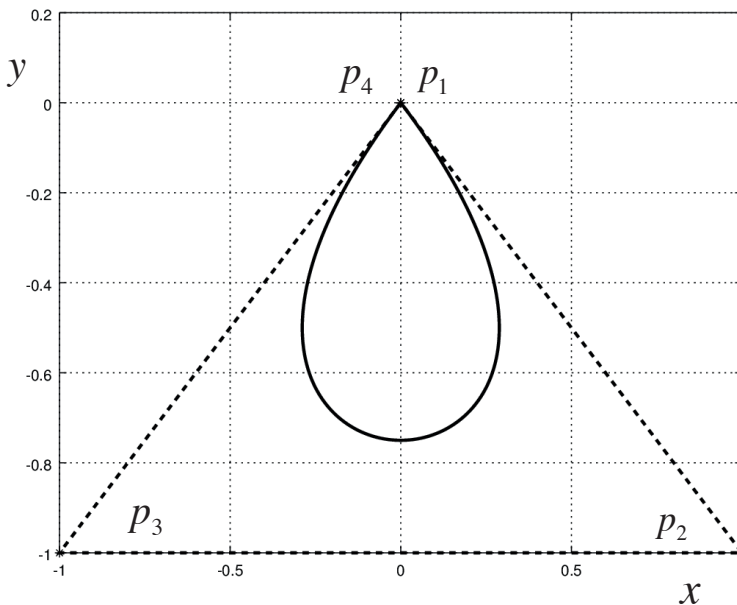
Note que as linhas tracejadas de 1 para 2 e de 3 para 4 (B_3) representam as respectivas retas tangentes (Propriedade 2). Para traçar o gráfico da curva $B_k(t)$, podemos gerar tantos pontos quanto forem necessários e assim suprir a resolução gráfica do desenho desejado. Nesse caso, basta diminuir o passo $h = 1 / np$ (np é o número de passos) com o qual se subdivide o domínio $[0, 1]$ do parâmetro t . Já para ajustar a curva gerada por $B_k(t)$ ao *layout* desejado, mantendo os extremos, basta arrastar o(s) ponto(s) de controle interno(s), o que gerará novas curvas.

Nos sistemas de computação gráfica que utilizam essa técnica de aproximação, a curva de Bézier mais utilizada é a $B_3(t)$, repetida tantas vezes quantas forem necessárias para formar um desenho de vários segmentos.

Confira no **Caderno de Algoritmos**, nos arquivos **Cap5Graf5.16.Bezier abc B3.m** e **Cap5Graf5.16.Bezier d B2 e B3.m**, os algoritmos que geram $np + 1 = 1001$ pontos de curvas $B_3(t)$ e $B_2(t)$ do Gráfico 5.16, na sua forma otimizada.

Tomando, por exemplo, os pontos referenciais $p_1 = (0, 0)$, $p_2 = (1, -1)$, $p_3 = (-1, -1)$ e $p_4 = (0, 0)$, esse algoritmo gera uma única curva de Bézier, tipo gota, conforme o Gráfico 5.17.

Gráfico 5.17 – Curva de Bézier tipo gota



Fonte: Elaboração própria.

No Gráfico 5.18, por exemplo, apresentamos um perfil superior de uma asa de avião composta de três segmentos de curvas de Bézier que devem passar por quatro pontos de ancoragem, A, B, C e D, e com inclinações predefinidas, conforme segue:

- | | |
|------------|-------------------------|
| A (0, 0) | com inclinação de 45°; |
| B (2, 1) | com inclinação de 0°; |
| C (8, 0.2) | com inclinação de -10°; |
| D (10, 0) | com inclinação de -5°; |

Verifique que somente conseguimos passar sobre os quatro pontos, A, B, C, D, se usarmos três curvas de Bézier separadas, devendo colocar dois pontos intermediários/auxiliares para cada curva, pois essas curvas passam sobre suas duas extremidades e, com a inclinação predefinida pelas retas que as ligam aos seus dois pontos intermediários. Assim, sugerimos estes *pontos intermediários*⁹ em cada segmento, na notação do Octave:

% parte 1: segmento AB

```
x(1) = 0;      y(1) = 0;
x(2) = 1;      y(2) = 1; %inclinação início = 45° entre 1 e 2
x(3) = 1.9;    y(3) = 1; %inclinação final = 0° entre 3 e 4
x(4) = 2;      y(4) = 1;
```

% parte 2: segmento BC

```
x(1) = 2;      y(1) = 1;
x(2) = 3;      y(2) = 1; %inclinação início = 0°
x(3) = 8 - 1;  y(3) = 0.2 + 1 * tan(10 * pi/180); % inclinação final = 10°
x(4) = 8;      y(4) = 0.2;
```

% parte 3: segmento CD

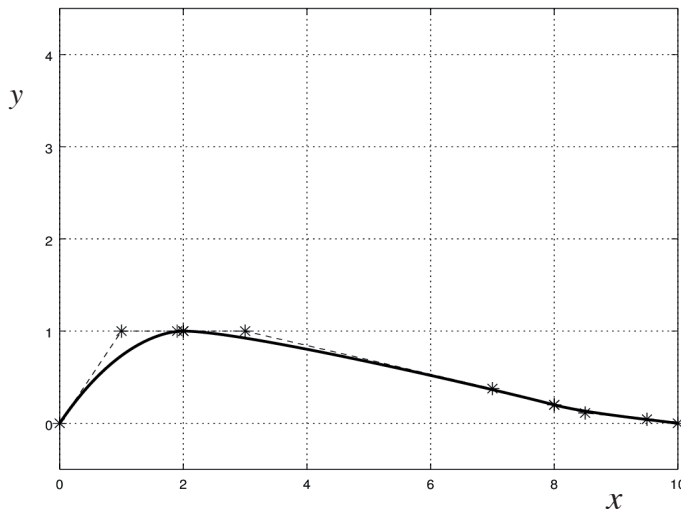
```
x(1) = 8;      y(1) = 0.2;
x(2) = 8 + 0.5; y(2) = 0.2 - 0.5 * tan(10 * pi/180); % inclinação início = 10°
x(3) = 10 - 0.5; y(3) = 0 + 0.5 * tan(05 * pi/180); % inclinação final = 5°
x(4) = 10;     y(4) = 0;
```



Explore variações desses pontos intermediários no **Exercício 5.3**.

No **Caderno de Algoritmos**, você encontra o arquivo **Cap5Graf5.18.Bezier.aerofolio.m**.

Gráfico 5.18 – Perfil superior de uma asa de avião



Fonte: Elaboração própria.

Podemos encontrar exemplos práticos de aplicação das curvas de Bézier no CorelDRAW®, na ferramenta que converte uma forma qualquer em curvas, permitindo a manipulação de “nós”.

Lembre-se de que está disponível no *link* <<http://sergiopeters.prof.ufsc.br/exercicios-e-respostas/>> o **Caderno de Exercícios e Respostas** para o aprofundamento dos estudos de cada capítulo deste livro.

5.7 CONCLUSÕES

Neste capítulo, iniciamos o estudo da teoria da aproximação de funções abordando três das suas várias metodologias. A primeira metodologia, a interpolação polinomial, que tem como condição de aproximação um princípio simples e óbvio, possui aplicações genéricas. Já a segunda metodologia, a interpolação via *splines* cúbicas, e a terceira, as curvas de Bézier, possuem princípios matemáticos mais robustos e são a base teórica de vários sistemas de computação gráfica. Nos próximos dois capítulos, o tema terá continuidade com o estudo de outras metodologias de aproximação de funções que fundamentam matemática e computacionalmente outros tipos de aplicações.

APROXIMAÇÕES POR SÉRIES E POR FUNÇÕES RACIONAIS

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- efetuar aproximações por séries de Maclaurin, Tchebychev e por funções racionais de Padé;
- calcular os erros de truncamento de cada aproximação; e
- indicar o tipo de aproximação mais adequada para cada família de função.

Neste capítulo, vamos abordar exclusivamente as aproximações de $y = f(x)$ com **expressão conhecida**, em $x \in [a, b]$, através de outra função, $z = g(x)$, mas agora utilizando aproximadoras geradas por séries e por funções racionais. Para fins de padronização e facilidade de avaliação da qualidade da $g(x)$, vamos normalizar o intervalo $[a, b]$ transformando-o no intervalo padrão $[-1, 1]$ através da transformação linear:

$$x(t) = \frac{(b-a)t + (b+a)}{2} \quad (1)$$

pois

$$\text{se } t = -1 \Rightarrow x = a$$

$$\text{se } t = +1 \Rightarrow x = b$$

A partir dessa normalização, aplicamos os métodos de aproximação à função $f(x(t))$ no intervalo padrão $t \in [-1, 1]$ e, por fim, efetuamos as aproximações na função composta $f(t(x))$, em que:

$$t(x) = \frac{2x - (b+a)}{b-a} \quad (2)$$

Exemplo 6.1: para a $f(x) = \text{sen}(2x + 3)$, $x \in [1, 15]$, obtenha o valor de $f(5)$ no domínio equivalente $[-1, 1]$.

Solução:

Para a função dada, temos que $f(x = 5) = \text{sen}(13)$. Já no domínio $[-1, 1]$, aplicando a eq. (1), temos $x(t) = 7t + 8$ e a composta $f(t(x)) = \text{sen}(14t + 19)$. Da eq. (2), para $x = 5 \Rightarrow t(x) = -3/7$. Assim, $t(x) = -3/7$, substituído em $f(t(x)) = \text{sen}(14t + 19)$, resulta em $f(t(x)) = \text{sen}(13)$, que é o mesmo valor.

Chamamos atenção para o baixo custo dessas transformações que, nas formas otimizadas, não excedem a 8 operações elementares.

Na aproximação de funções com expressão conhecida, uma questão fundamental é: **quais características ou propriedades a $z = g(x)$ deverá possuir para ser considerada uma aproximadora de qualidade?**


Uma aproximadora ideal é aquela que cumpre os seguintes requisitos:

- a) os erros de truncamento $\text{Erro}(x) = |g(x) - f(x)|$, $\forall x \in [a, b]$ devem ser mensurados previamente;
- b) os erros de truncamento $\text{Erro}(x) = |g(x) - f(x)|$ devem ser bem distribuídos em todo o domínio $[a, b]$;
- c) o tempo de resposta nas chamadas (cálculos de valores) da $z = g(x)$ deve ser mínimo; e
- d) a demanda de memória para armazenar os parâmetros identificadores da função $g(x)$ deve ser mínima.

6.1 APROXIMAÇÃO DE $y = f(x)$ POR SÉRIES

Como vimos no Capítulo 5, a aproximação de $y = f(x)$ com $x \in [a, b]$ pela interpolação polinomial consiste em dividir $[a, b]$ em n partes e obter o interpolador $P_n(x)$ (na forma geral, de Lagrange, ou Gregory-Newton), cujo erro de truncamento máximo para $h=(b-a)/n$ é da ordem de

$$\text{Erro } P_n(x) < \frac{\text{Max} \left| f^{(n+1)}(x) \right|_{x \in [a,b]} h^{n+1}}{4(n+1)}$$

 $f^{(n+1)}(x)$ refere-se à derivada de ordem $(n+1)$ de $f(x)$.

Entretanto, o tempo de resposta e a demanda de memória requerida são grandes, pois normalmente precisamos usar um interpolador de grau n elevado para ter um erro de truncamento pequeno. Logo, o interpolador, apesar de ser tentadoramente simples de obter, não satisfaz aos requisitos (c) e (d) de uma boa aproximadora de uma função com expressão conhecida. Também as *splines* cúbicas e as curvas de Bézier não foram desenvolvidas para o propósito deste capítulo. Assim, abordaremos outras técnicas de aproximação de funções com expressão conhecida que forneçam aproximadoras mais próximas da ideal.

6.1.1 Aproximação por séries de Taylor

Segundo o **teorema de Taylor**: toda função $y = f(x)$ que seja continuamente diferenciável em um domínio $[a, b]$ (ou o equivalente $[-1, 1]$) pode ser expressa exatamente pela série:

$$f(x) = f(\beta) + \frac{f'(\beta)(x-\beta)}{1!} + \frac{f''(\beta)(x-\beta)^2}{2!} + \dots + \frac{f^{(n)}(\beta)(x-\beta)^n}{n!} + \dots \quad (3a)$$

em que $\beta \in [a, b]$.

Reescrevendo a eq. (3a) com número finito de parcelas, temos:

$$f(x) = \underbrace{f(\beta) + \frac{f'(\beta)(x-\beta)}{1!} + \dots + \frac{f^{(n)}(\beta)(x-\beta)^n}{n!}}_{P_n(x)} + \underbrace{\frac{f^{(n+1)}(\xi)(x-\beta)^{n+1}}{(n+1)!}}_{R_n(x)} \quad (3b)$$

em que o termo

$$R_n(x) = \frac{f^{(n+1)}(\xi)(x-\beta)^{n+1}}{(n+1)!} \quad (3c)$$

é denominado de **resto** da série e $\xi \in (\beta, x)$ é um número com localização conhecida, mas valor desconhecido.

Conforme destacamos na eq. (3b), a primeira parte será um polinômio $P_n(x)$ de grau n , denominado de aproximador de Taylor da $y = f(x)$, se a **série for convergente**, sendo $R_n(x)$ uma estimativa do erro de truncamento.

Como vamos padronizar o domínio $[a, b]$ da aproximanda para $[-1, 1]$, podemos fixar o β da série em $\beta = 0$ (ponto médio do intervalo $[-1, 1]$):

$$\Rightarrow f(x) \cong f(0) + \frac{f'(0)x}{1!} + \frac{f''(0)x^2}{2!} + \dots + \frac{f^{(n)}(0)x^n}{n!} + R_n(x) \quad (4a)$$

Essa série é denominada de série de Taylor e Maclaurin e será denotada por $M_n(x)$.

Exemplo 6.2: confira a relação de algumas funções com suas séries de Taylor e Maclaurin:

$$a) e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

$$b) \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + \frac{(-1)^n x^{2n}}{(2n)!} + \dots$$

$$c) \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots + \frac{(-1)^{n+1} x^n}{n} + \dots$$

$$d) \arctg(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots + (-1)^{n+1} \frac{x^{2n-1}}{2n-1} + \dots$$

$$e) \int_0^x e^{-z^2} dz = x - \frac{x^3}{1!3} + \frac{x^5}{2!5} - \frac{x^7}{3!7} + \dots + \frac{(-1)^n x^{2n+1}}{n!(2n+1)} + \dots$$

$$f) \int_0^x \cos(\sqrt{z}) dz = x - \frac{x^2}{2 \cdot 2!} + \frac{x^3}{3 \cdot 4!} - \frac{x^4}{4 \cdot 6!} + \dots + \frac{(-1)^{n+1} x^n}{n(2n-2)!} + \dots$$

Note que, depois de truncar a série do **Exemplo 6.2**, item (f), temos um único polinômio aproximando uma composição de três funções (raiz quadrada, cosseno e a integral).

Uma das grandes dificuldades na aproximação de funções por séries de Taylor é a determinação do valor de $\xi \in (\beta, x)$ que gere o resto $R_n(x)$ correto, pois ele é desconhecido. A alternativa é estimá-lo pelo seu **majorante**, resultando em um limite máximo para o erro de truncamento $R_n(x)$ via

$$\text{Erro Taylor}(x) \leq \frac{|x - \beta|^{n+1} M}{(n+1)!}, \text{ onde } M = \max_{x \in [a, b]} |f^{(n+1)}(x)| \quad (4b)$$

A seguir, vamos exemplificar o uso do majorante no erro de truncamento.

Exemplo 6.3a: delimite o erro máximo cometido ao aproximar $f(x) = e^x$, $x \in [-1, 1]$ por Maclaurin com $n = 5$.

Solução:

$$\text{Pelo resto da série} \Rightarrow R_5(x) = \frac{f^{(5+1)}(\xi) |x - 0|^{5+1}}{(5+1)!}$$

tomamos o seu valor máximo, na ausência do valor de ξ , em que

$$M = \max_{x \in [-1, +1]} |f^{(6)}(x)| \Rightarrow M = \max_{x \in [-1, +1]} |e^x| = e^{+1} \text{ em } x = 1$$

$$\text{Erro Maclaurin}(x) \leq \frac{e^1 |x - 0|^{5+1}}{(5+1)!}$$

Esse é o limite do erro local, para um x específico do intervalo.

Na expressão do erro de Maclaurin, como $x \in [-1, 1]$, o erro máximo global ocorre com $|x - 0|^{5+1}$ em $x = 1$ ou $x = -1$, então

$$\text{Erro Maclaurin} \leq \frac{|1 - 0|^{5+1} e^1}{(5+1)!} = 0.003775$$

Esse é o limite do erro global, para o intervalo padrão $[-1, 1]$.

Exemplo 6.3b: determine o grau n mínimo para aproximar $f(x) = e^x, x \in [-1, 1]$ por Taylor e Maclaurin com erro global inferior a $\varepsilon = O(10^{-6})$.

Solução:

Pelo resto máximo da série de Maclaurin, temos

$$\text{Erro Maclaurin}(x) \leq \frac{M |x-0|^{n+1}}{(n+1)!} \text{ com } M = \max_{x \in [-1, +1]} |e^x| = e^+ \text{ em } x = 1$$

E o erro global máximo do intervalo ocorre em $x = 1$ ou $x = -1$:

$$\text{Erro Maclaurin Max} = \frac{e^{+1} |1-0|^{n+1}}{(n+1)!}$$

Para $n = 8 \Rightarrow \text{Erro Maclaurin Max} = 7.4908560e - 06$ de ordem $O(10^{-5})$.

Para $n = 9 \Rightarrow \text{Erro Maclaurin Max} = 7.4908560e - 07$ de ordem $O(10^{-6})$.

Logo, a aproximação de Maclaurin com grau $n = 9$ gera erro máximo da ordem de $O(10^{-6})$.

Alternativamente, temos um limite para o erro de truncamento máximo aplicável às séries convergentes e alternadas nos sinais.

Teorema 1: se a série de Taylor da $y = f(x)$ for convergente e com termos de **sinais alternados**, então o valor do resto $R_n(x)$ não será superior ao valor máximo do primeiro termo não nulo abandonado.

Exemplo 6.4: delimite o erro máximo cometido ao aproximar $f(x) = e^{-x}, x \in [-1, 1]$ por Maclaurin com $n = 5$.

Solução:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} + \dots + \frac{(-1)^n x^n}{n!} + \frac{(-1)^{n+1} x^{n+1}}{(n+1)!} + \dots$$

a) Pelo resto da série $\Rightarrow R_5(x) = \frac{f^{(5+1)}(\xi) |x-0|^{5+1}}{(5+1)!}$,

tomamos o seu valor máximo, com

$$M = \max_{x \in [-1, +1]} |f^{(6)}(x)| \Rightarrow M = \max_{x \in [-1, +1]} |e^{-x}| = e^{-(-1)} \text{ em } x = -1.$$

$$\text{Erro Maclaurin}(x) \leq \frac{e^1 |x-0|^{5+1}}{(5+1)!} \quad \forall x \in [-1, +1]$$

Esse é o limite do erro local válido, para um x específico do intervalo. Tomando o erro global máximo nas extremidades do intervalo, em $x = -1$:

$$\text{Erro Maclaurin Max} = \frac{|1-0|^{5+1} e^1}{(5+1)!} = 0.003775$$

- b) Como essa série de Maclaurin é convergente e de termos com sinais alternados, podemos aplicar o **Teorema 1** tomando o primeiro termo abandonado depois de $n = 5$, em $x = +1$, logo:

$$\text{Erro Maclaurin Max} = \left| \frac{(-1)^{5+1} (1)^{5+1}}{(5+1)!} \right| = 0.001388889$$

Então, considerando que as duas formas de cálculo de erros máximos são válidos, podemos tomar o menor limite do erro, 0.001388889 de ordem de 10^{-3} , para o aproximador de Maclaurin de grau $n = 5$:

$$e^{-x} \cong M_5(x) = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!}$$

Exemplo 6.5: determine o grau n mínimo para aproximar $f(x) = \text{sen}(x)$, $x \in [-1, 1]$ por Maclaurin com $\varepsilon = O(10^{-6})$.

Solução:

A série de Maclaurin para $\text{sen}(x)$ é formada por termos de expoentes ímpares:

$$\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + \frac{(-1)^i x^{2i+1}}{(2i+1)!} + \dots$$

Assim:

a) Tomando o erro de truncamento pelo valor máximo do resto da série

$$\text{Erro Maclaurin Max} = \frac{|x - \beta|^{n+1} M}{(n+1)!}, \text{ em que } M = \max_{x \in [a,b]} |f^{(n+1)}(x)|$$

teremos:

$$f(x) = \text{sen}(x), f'(x) = \cos(x), f''(x) = -\text{sen}(x), f'''(x) = -\cos(x),$$

$$M = \max_{x \in (a,b)} |f^{(n+1)}(x)| = \max_{x \in [a,b]} |\pm \text{sen}(x)|, \forall n+1 \text{ par; e}$$

$$M = \max_{x \in (a,b)} |f^{(n+1)}(x)| = \max_{x \in [a,b]} |\pm \cos(x)|, \forall n+1 \text{ ímpar.}$$

Como n não nulo é sempre ímpar, então $n + 1$ será sempre par, $M = |\text{sen}(1)| = 0.841470984807897$, e esse erro é máximo em $x = 1$, então

$$\text{Erro Maclaurin Max} = \frac{|1-0|^{n+1} 0.84147098}{(n+1)!} = 10^{-6}$$

Para $n = 7$ ($n + 1$ par),

$$\text{Erro Maclaurin Max} = \frac{|1-0|^{n+1} 0.84147098}{(n+1)!} = 2.0869816e-05 \quad (O(10^{-5}))$$

Para $n = 9$,

$$\text{Erro Maclaurin Max} = \frac{|1-0|^{n+1} 0.84147098}{(n+1)!} = 2.318868e-07 \quad (O(10^{-7}))$$

Logo, pelo teorema da série de Taylor, a aproximação de grau $n = 9$ tem erro de truncamento máximo menor do que a ordem de $O(10^{-6})$.

- b) Como essa série de Maclaurin é convergente e de termos com sinais alternados, podemos aplicar o **Teorema 1**:

$$\text{sen}(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + \frac{(-1)^i x^{2i+1}}{(2i+1)!} + \dots$$

Para a aproximação de grau $n = 7$, por exemplo, o primeiro termo abandonado é de grau 9, obtido com $i = 4$,

$$\left| \frac{(-1)^i (1)^{2*4+1}}{(2*4+1)!} \right| = 2.755732e-06$$

Logo, pelo **Teorema 1**, ao abandonar o termo de grau $n = 2i + 1 = 9$, já teremos erro de truncamento máximo da ordem de $O(10^{-6})$. Então, considerando que as duas formas de cálculo de erros máximos são válidos, podemos tomar a aproximação de grau $n = 7$, garantindo o erro máximo global da ordem de $O(10^{-6})$:

$$\text{sen}(x) \cong M_7(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

Exemplo 6.6: determine o grau n mínimo para aproximar $f(x) = \ln(1 + x)$, $x \in (-1, 1]$ por Maclaurin com erro máximo da ordem de $O(10^{-6})$.

Solução:

Nesse exemplo, a aplicação do majorante do teorema do resto da série de Taylor e Maclaurin, dada pela eq. (4), gera erro máximo tendendo ao infinito quando x tende a -1 . Então, a aplicação do majorante nesse teorema não tem valia prática, pois de nada adianta saber que o erro máximo é inferior a infinito. Como essa série de Maclaurin é de sinais alternados, podemos aplicar o **Teorema 1**:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} + \dots + \frac{(-1)^n x^n}{n!} + \underbrace{\frac{(-1)^{n+1} x^{n+1}}{(n+1)!} + \dots}_{R_n(x)}$$

em que

$$R_n(x) \leq \left| \frac{(-1)^{n+2} x^{n+1}}{n+1} \right| \leq 10^{-6}$$

Então, podemos avaliar o limite do erro de truncamento pelo 1º termo abandonado com $x = 1$:

$$\frac{(-1)^{n+2}}{n+1} \leq 10^{-6} \Rightarrow n \geq 1000000$$

Note que a série do **Exemplo 6.6** possui velocidade de convergência muito lenta, implicando a necessidade de uma grande quantidade de termos para assegurar uma precisão ainda relativamente baixa.

Outra característica da aproximação por Taylor e Maclaurin é a não distribuição uniforme dos erros no domínio, o que exige grande número de termos nas séries com convergência lenta para assegurar **erros mínimos** (com picos máximos sempre localizados nos **extremos**). Assim, o tempo de resposta pode ficar muito alto ou até proibitivo.

Exemplo 6.7: calcule o erro máximo **exato** ao aproximar $f(x) = e^x$, $x \in [-1, 1]$ por Maclaurin com grau $n = 5$. Lembre-se de que, no **Exemplo 6.3a**, o limite do erro de truncamento foi estimado em 0.003775.

Solução:

$$e^x \cong M_5(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} \quad (M_5(x) \text{ obtido por série de Maclaurin})$$

$$\text{Erro exato Maclaurin}(x) = |M_5(x) - e^x|$$

x	-1.0	-0.5	0	0.5	1.0
<i>Erro exato Maclaurin</i> (x)	0.0012128	0.000020243	0	0.000023354	0.00161516

Observe que os erros exatos ficaram todos abaixo do limite do erro de truncamento 0.003775 estimado pelo resto da série de Taylor e Maclaurin no **Exemplo 6.3a**. Observe também que os erros são crescentes a partir do ponto $\beta = 0$. Então, considerando a característica inerente dessa aproximação, temos que as séries de Taylor e Maclaurin não satisfazem aos requisitos (b) e (c) da aproximadora ideal.

Exemplo 6.8: monte um algoritmo que determine os coeficientes da série de Maclaurin estabelecida, a seguir, para um grau genérico, por exemplo grau $n = 20$:

$$\int_0^x e^{-z^2} dz = x - \frac{x^3}{1!3} + \frac{x^5}{2!5} - \frac{x^7}{3!7} + \dots + \frac{(-1)^i x^{2i+1}}{i!(2i+1)} + \dots$$

Solução:

Observe que esta série tem coeficientes nulos:

$$\begin{aligned} \int_0^x e^{-z^2} dz &= 0x^0 + 1x^1 + 0x^2 - \frac{x^3}{1!3} + 0x^4 + \frac{x^5}{2!5} + 0x^6 - \frac{x^7}{3!7} + 0x^8 + \dots + \frac{(-1)^i x^{2i+1}}{i!(2i+1)} + \dots \\ &= c_1x^0 + c_2x^1 + c_3x^2 + c_4x^3 + c_5x^4 + c_6x^5 + c_7x^6 + c_8x^7 + \dots + c_{2i+2}x^{2i+1} + \dots \end{aligned}$$

Os coeficientes de grau par (índice ímpar) são nulos e os de grau ímpar (índice par) são definidos pela lei de formação em função de i , conforme o algoritmo **Cap6exem6.8.m** disponível no **Caderno de Algoritmos** no link <<http://sergiopeters.prof.ufsc.br/algoritmos-livro/>>.

No **Caderno de Algoritmos**, você também encontra implementados todos os demais exemplos apresentados neste capítulo.

A seguir, vamos abordar uma técnica de aproximação que objetiva melhorar a distribuição dos erros da série de Taylor e Maclaurin bem como acelerar a sua convergência.

6.1.2 Aproximação por séries de Tchebychev

Definição1: um **polinômio de Tchebychev** de grau n de *primeira ordem* é toda expressão do tipo:

$$T_n(x) = \cos(n \arccos(x)) \text{ com } x \in [-1, 1] \quad (5)$$

Como $\arccos(x) = \theta \Rightarrow \cos(\theta) = x$,

logo

$$T_n(x) = \cos(n\theta) \text{ com } \theta \in [0, \pi]. \quad (6)$$



Existem também polinômios de Tchebychev de segunda ordem (POLINÔMIOS de Tchebychev, 2016).

Para cada n , desenvolvendo a eq. (6) com o uso de identidades trigonométricas elementares, resultam as expressões de polinômios convencionais para os $T_n(x)$:

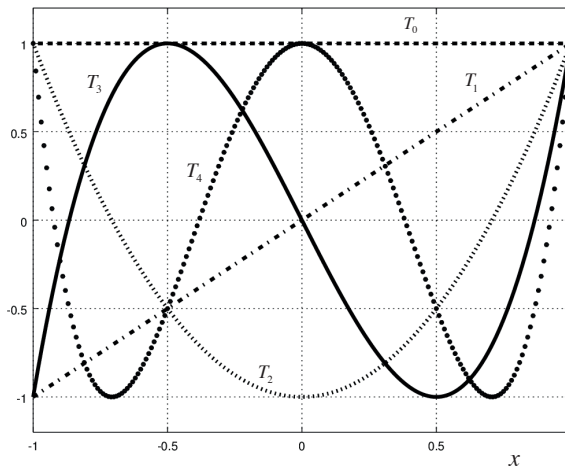
$$\begin{aligned} T_0(x) &= \cos(0 \cdot \theta) = 1 \\ T_1(x) &= \cos(1\theta) = x \\ T_2(x) &= \cos(2\theta) = 2\cos^2(\theta) - 1 = 2x^2 - 1 \\ T_3(x) &= \cos(3\theta) = 4x^3 - 3x \\ T_4(x) &= \cos(4\theta) = 8x^4 - 8x^2 + 1 \\ T_5(x) &= \cos(5\theta) = 16x^5 - 20x^3 + 5x \\ T_6(x) &= \cos(6\theta) = 32x^6 - 48x^4 + 18x^2 - 1 \\ T_7(x) &= \cos(7\theta) = 64x^7 - 112x^5 + 56x^3 - 7x \\ T_8(x) &= \cos(8\theta) = 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\ T_9(x) &= \cos(9\theta) = 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \end{aligned} \quad (7a)$$

E a fórmula recursiva para obter $T_{n+1}(x)$ é:

$$T_{n+1}(x) = 2 * x * T_n(x) - T_{n-1}(x) \quad (7b)$$

No Gráfico 6.1, apresentamos os primeiros cinco polinômios de Tchebyshev de primeira ordem.

Gráfico 6.1 – Polinômios de Tchebyshev de graus $n = 0, 1, 2, 3$ e 4



Fonte: Elaboração própria.

Nas eqs. (7a) explicitando as potências de x em função dos polinômios $T_n(x)$, que serão abreviados por T_n para facilitar o seu uso algébrico, resultam:

$$\begin{aligned} x^0 &= T_0 \\ x^1 &= T_1 \\ x^2 &= (T_2 + T_0)/2 \\ x^3 &= (T_3 + 3T_1)/4 \\ x^4 &= (T_4 + 4T_2 + 3T_0)/8 \\ x^5 &= (T_5 + 5T_3 + 10T_1)/16 \\ x^6 &= (T_6 + 6T_4 + 15T_2 + 10T_0)/32 \\ x^7 &= (T_7 + 7T_5 + 21T_3 + 35T_1)/64 \\ x^8 &= (T_8 + 8T_6 + 28T_4 + 56T_2 + 35T_0)/128 \\ x^9 &= (T_9 + 9T_7 + 36T_5 + 84T_3 + 126T_1)/256 \end{aligned} \quad (7c)$$

A seguir, vamos apresentar algumas propriedades dos polinômios de Tchebychev.

Propriedade 1: $T_n(x)$ é um polinômio de grau n e existe um único $T_n(x)$ para cada grau n . O coeficiente de x^n em $T_n(x)$ é sempre igual a 2^{n-1} .

Propriedade 2: $|T_n(x)| \leq 1, \forall x \in [-1, +1]$, então $\max_{x \in [-1, +1]} |T_n(x)| = 1$.

Propriedade 3: todas as n raízes α_k de $T_n(x) = 0$ (“nós” de Tchebychev) são obtidas diretamente via

$$\alpha_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad \forall k = 1, 2, \dots, n \quad (n > 0) \quad (8a)$$

Esses n “nós” são obtidos através de uma distribuição uniforme em $\theta_k \in (0, \pi)$ de modo que cada raiz α_k satisfaça $T_n(\alpha_k) = \cos(n \arccos(\alpha_k)) = 0$, em que $\alpha_k = \cos(\theta_k)$ e θ_k dado por,

$$\theta_k = \left(\frac{2k-1}{2n}\pi\right) \quad (8b)$$

Verificamos que $T_n(\alpha_k) = 0$, substituindo os valores de x da eq. (5) pelas raízes α_k dadas pela eq. (8a), conforme segue,

$$T_n(\alpha_k) = \cos\left(n * \arccos\left(\cos\left(\frac{2k-1}{2n}\pi\right)\right)\right) = \cos\left(n\left(\frac{2k-1}{2n}\pi\right)\right) = \cos\left((2k-1)\frac{\pi}{2}\right) = 0$$

Propriedade 4: os polinômios de Tchebychev, $T_n(x)$, formam uma sequência de polinômios **ortogonais**, com relação ao peso $W(x) = 1/\sqrt{1-x^2}$, no intervalo $x \in [-1, +1]$, ou seja:

$$\int_{-1}^{+1} \frac{T_n(x) * T_m(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0, & n \neq m \\ \pi, & n = m = 0 \\ \pi/2, & n = m \neq 0 \end{cases} \quad (9)$$

Propriedade 5 (teorema de Tchebychev): toda função $y = f(x)$ contínua em $[-1, 1]$ pode ser aproximada usando polinômios de Tchebychev por meio da série:

$$f(x) = \sum_{i=0}^{\infty} b_i T_i = b_0 + b_1 * T_1(x) + b_2 * T_2(x) + \dots + b_k * T_k(x) + \dots \quad (10a)$$

$$\text{em que } \begin{cases} b_0 = \frac{1}{\pi} \int_{-1}^{+1} \frac{f(x)}{\sqrt{1-x^2}} dx, & i = 0 \\ b_i = \frac{2}{\pi} \int_{-1}^{+1} \frac{f(x) T_i(x)}{\sqrt{1-x^2}} dx, & \forall i = 1, 2, \dots, k \end{cases} \quad (10b)$$

Podemos obter cada coeficiente b_i da série de Tchebychev, dados pelas eqs. (10b), fazendo o produto interno da eq. (10a) pelo polinômio ortogonal de Tchebychev de ordem i usando a **Propriedade 4** (ABRAMOWITZ; STEGUN, 1961).

Por exemplo, para $i = 0$:

$$\int_{-1}^{+1} \frac{f(x) * T_0(x)}{\sqrt{1-x^2}} dx = \int_{-1}^{+1} \frac{(b_0 * T_0(x) + b_1 * T_1(x) + b_2 * T_2(x) + \dots + b_k * T_k(x) + \dots) * T_0(x)}{\sqrt{1-x^2}} dx$$

Temos

$$\int_{-1}^{+1} \frac{T_0(x) * T_0(x)}{\sqrt{1-x^2}} dx = \pi$$

e demais produtos internos são nulos (pela eq. 9), logo

$$b_0 = \frac{1}{\pi} \int_{-1}^{+1} \frac{f(x) * T_0(x)}{\sqrt{1-x^2}} dx$$

Para $i = 1$:

$$\int_{-1}^{+1} \frac{f(x) * T_1(x)}{\sqrt{1-x^2}} dx = \int_{-1}^{+1} \frac{(b_0 * T_0(x) + b_1 * T_1(x) + b_2 * T_2(x) + \dots + b_k * T_k(x) + \dots) * T_1(x)}{\sqrt{1-x^2}} dx$$

Temos

$$\int_{-1}^{+1} \frac{T_1(x) * T_1(x)}{\sqrt{1-x^2}} dx = \frac{\pi}{2}$$

e demais produtos internos nulos, logo

$$b_1 = \frac{2}{\pi} \int_{-1}^{+1} \frac{f(x) * T_1(x)}{\sqrt{1-x^2}} dx$$

e assim por diante para os demais $i = 2, 3, \dots, k$.

Desse modo, para obter um polinômio aproximador de grau k para a $f(x)$ usando o teorema de Tchebychev, temos que efetuar $k + 1$ integrais definidas, que normalmente precisam ser obtidas via integração numérica de Gauss-Tchebychev, também chamada de **quadratura de Gauss-Tchebychev**, que veremos no Capítulo 8. Neste momento, vamos apenas apresentar a fórmula final para aproximar numericamente os coeficientes b_i :

$$b_0 = \frac{1}{\pi} \int_{-1}^{+1} \frac{f(x) * 1}{\sqrt{1-x^2}} dx = \frac{1}{\pi} \left[\frac{\pi}{m} \sum_{j=1}^m f(x_j) \right]$$

$$b_0 = \frac{1}{m} \sum_{j=1}^m f(x_j) \tag{11a}$$

$$b_i = \frac{2}{\pi} \int_{-1}^{+1} \frac{f(x) * T_i(x)}{\sqrt{1-x^2}} dx = \frac{2}{\pi} \left[\frac{\pi}{m} \sum_{j=1}^m f(x_j) T_i(x_j) \right]$$

$$b_i = \frac{2}{m} \sum_{j=1}^m f(x_j) T_i(x_j), \quad \forall i = 1, 2, \dots, k \tag{11b}$$

em que os x_j são m “nós” de Tchebychev: $x_j = \cos\left(\frac{2j-1}{2m}\pi\right)$, $\forall j = 1, 2, \dots, m$.

Existe um número m mínimo de “nós” de Tchebychev para obter uma integração numérica com a precisão numérica desejada.

Por esse caminho, é necessário já partir de uma **expressão da função $f(x)$, disponível em alguma biblioteca de funções**, para então aplicar o algoritmo da respectiva integração numérica.

Determinados os valores dos coeficientes b_p , devemos substituir os respectivos polinômios $T_i(x)$ de Tchebychev, dados pela eq. (7a), para obter essa aproximação diretamente em função de x .

Uma alternativa mais simples, denominada de **Tchebychev-Maclaurin**, a essas aproximações numéricas via integrais, é utilizar as demais propriedades partindo de uma aproximação da $f(x)$ por série de Maclaurin (sem necessidade de se ter previamente uma $f(x)$ de alguma biblioteca de funções), e seguir algebricamente os seguintes passos:

Primeiro passo: obtemos o aproximador de Maclaurin $M_n(x)$ para $y = f(x)$ com erro $E_{T1} = \varepsilon$ desejado:

$$f(x) = M_n(x) + E_{T1}$$

em que

$$M_n(x) = a_0x^0 + a_1x^1 + a_2x^2 + \dots + a_nx^n \quad (12)$$

Segundo passo: substituímos na eq. (12) todas as potências x^j pelas respectivas expressões em T_i dos polinômios de Tchebychev conforme a eq. (7c) e os agrupamos pelo mesmo índice, conforme segue:

$$\begin{aligned} f(x) &= a_0 * T_0 + a_1 * T_1 + a_2 * (T_2 + T_0)/2 + a_3 * (T_3 + 3T_1)/4 \\ &\quad + a_4 * (T_4 + 4T_2 + 3T_0)/8 + \dots + E_{T1} \\ f(T) &= b_0 + b_1T_1 + b_2T_2 + \dots + b_nT_n + E_{T1} \end{aligned} \quad (13)$$

em que o último termo $b_n * T_n < \frac{b_n * 1}{2^{n-1}}$, conforme as **Propriedades 1 e 2**.

Terceiro passo: truncamos a expressão dada na eq. (13) a partir de $b_{k+1} * T_{k+1}$ com $k < n$ (escolhido cuidadosamente para não aumentar a ordem de grandeza do erro de truncamento) e denotamos todas as parcelas truncadas entre $k + 1$ e n por E_{T_2} ,

$$f(T) = b_0 + b_1 T_1 + b_2 T_2 + \dots + b_k T_k + E_{T_2} + E_{T_1} \quad (14)$$

de modo que

$$|E_{T_2}| + |E_{T_1}| \leq \varepsilon$$

Quarto passo: substituímos, na eq. (14) truncada, cada T_i pela sua respectiva expressão em x^i fornecida por (7a) e agrupamos os termos em x^i , gerando o polinômio diretamente em função de x :

$$f(x) = TC_k^M(x) + E_{T_2} + E_{T_1} \quad (15a)$$

em que,

$$TC_k^M(x) = c_0 + c_1 x + \dots + c_k x^k \quad (15b)$$

Temos, então, um aproximador polinomial de Tchebychev-Maclaurin para a função $y = f(x)$ de grau $k < n$ (n é o grau do aproximador de Maclaurin na precisão E_{T_1}), dado por $f(x) \cong TC_k^M(x)$, com erro $|E_{T_2}| + |E_{T_1}| < \varepsilon$. A diferença entre os graus n e k é denominada de **efeito telescópico** do aproximador de Tchebychev. Esse efeito é inversamente proporcional à velocidade de convergência da série de Maclaurin da $f(x)$.

Exemplo 6.9: aproxime $f(x) = e^x$ em $x \in [-1, +1]$ por Tchebychev de grau $k = 3$ usando diretamente a **Propriedade 5** (teorema de Tchebychev) e partindo da expansão em série de Maclaurin de grau $n = 4$ (usando Tchebychev-Maclaurin).

Solução:

Usando a **Propriedade 5** (teorema de Tchebychev), via algoritmo de Tchebychev com as aproximações das integrais das eqs. (11), vamos obter $TC_3(x)$ de grau final $k = 3$:

$$TC_3(T_i) = 1.26606587775200T_0 + 1.1303182079849701T_1 + 0.271495339534075T_2 \\ + 0.0443368498486627T_3$$

Devemos substituir os respectivos polinômios $T_i(x)$ de Tchebychev, dados pelas expressões (7a), para obter essa aproximação diretamente em função de x .

Alternativamente, vamos usar Tchebychev-Maclaurin:

- a) Aproximação de $f(x)$ por Maclaurin ($\beta = 0$) com grau $n = 4$:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{\max |f^{(4+1)}(x)| (x-0)^{4+1}}{(4+1)!}$$

$$e^x \cong 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + E_{T1}$$

$$M_4(x) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 \text{ com erro máximo}$$

$$E_{T1} = |e^1| * |1-0|^{(4+1)} / (4+1)! = 2.26523 * 10^{-2} \cong O(10^{-2}) \text{ (pelo teorema do resto).}$$

- b) Substituição algébrica de x^i pelos respectivos polinômios de Tchebychev, conforme as eqs. (7c):

$$e^x \cong T_0 + T_1 + \frac{1}{2} \left(\frac{T_2 + T_0}{2} \right) + \frac{1}{6} \left(\frac{T_3 + 3T_1}{4} \right) + \frac{1}{24} \left(\frac{T_4 + 4T_2 + 3T_0}{8} \right) + E_{T1}$$

$$e^x \cong \frac{81}{64}T_0 + \frac{9}{8}T_1 + \frac{13}{48}T_2 + \frac{1}{24}T_3 + \frac{1}{192}T_4 + E_{T1}$$

Sabendo que o valor máximo de qualquer polinômio de Tchebychev é a unidade, conforme **Propriedade 2**, vemos que o termo $(1/192)T_4$ adicionado ao erro de truncamento existente E_{T1} não alterará a sua ordem de precisão total, pois

$$E_T \cong (1/192)T_4 + E_{T1} < 0.00520833 + 0.0226523 \cong 0.0278607 \cong O(10^{-2})$$

Então, mesmo desprezando o termo de 4ª ordem ($k = 4$) da série expandida por polinômios de Tchebychev, o erro de truncamento total E_T fica da mesma ordem de grandeza de E_{T1} , $O(10^{-2})$.

c) Truncando a série e mantendo os termos até grau $k = 3$, temos:

$$e^x \cong \frac{81}{64}T_0 + \frac{9}{8}T_1 + \frac{13}{48}T_2 + \frac{1}{24}T_3 + E_T$$

d) E substituindo os polinômios de Tchebychev T_i em função de x^i , conforme a eq. (7a), temos:

$$e^x \cong \frac{81}{64}x^0 + \frac{9}{8}x^1 + \frac{13}{48}(2x^2 - 1) + \frac{1}{24}(4x^3 - 3x) + E_T$$

$$e^x \cong \frac{191}{192} + x^1 + \frac{13}{24}x^2 + \frac{1}{6}x^3 + E_T$$

$$TC_3^M(x) = \frac{191}{192} + x^1 + \frac{13}{24}x^2 + \frac{1}{6}x^3$$

Em que $TC_3^M(x)$ é o aproximador de Tchebychev-Maclaurin de grau final $k = 3$.

Então, temos o aproximador de Tchebychev em função de T_i :

$$TC_3(T_i) = 1.26606587775200T_0 + 1.1303182079849701T_1 \\ + 0.271495339534075T_2 + 0.0443368498486627T_3$$

E o aproximador de Tchebychev-Maclaurin em função de T_i :

$$TC_3^M(T_i) = 1.265625T_0 + 1.125T_1 + 0.2708333333333333T_2 \\ + 0.04166666666666667T_3$$

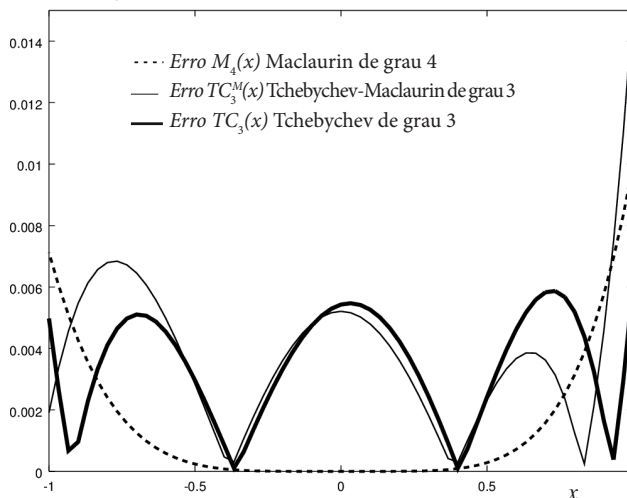
Confira o cálculo dos erros exatos para aproximação de e^x das séries de Maclaurin ($n = 4$), Tchebychev-Maclaurin ($n = 3$) e Tchebychev ($n = 3$), em relação ao valor exato, na Tabela 6.1.

Tabela 6.1 – Erros exatos em alguns pontos e erros máximos das séries de Maclaurin, Tchebychev-Maclaurin e Tchebychev

x	-1.0	-0.5	0	0.5	1.0	<i>Erro Máximo</i>
<i>Erro $M_4(x)$</i>	0.0071	0.00024	0.0	0.00028	0.0099	0.0099
<i>Erro $TC_3^M(x)$</i>	0.0019	0.0028	0.0052	0.0023	0.0151	0.0152
<i>Erro $TC_3(x)$</i>	0.0049	0.0029	0.0054	0.0024	0.0060	0.00606

Fonte: Elaboração própria.

Note que, nas duas aproximações de Tchebychev, os erros estão bem distribuídos e todos se mantêm abaixo do erro de truncamento total previsto inicialmente, $E_T = 0.0278607$, de ordem $O(10^{-2})$, mas a aproximação via teorema de Tchebychev (**Propriedade 5**) resulta em erros menores, conforme a tabela de erros anterior, representada no Gráfico 6.2 a seguir.

Gráfico 6.2: Erros da aproximação de e^x por séries de Maclaurin $M_4(x)$, Tchebychev-Maclaurin $TC_3^M(x)$ e Tchebychev $TC_3(x)$ 

Fonte: Elaboração própria.

Nesse caso, a aproximação de Tchebychev, de grau final $n = 3$, obtida via teorema de Tchebychev, promoveu uma melhor distribuição dos erros ao longo do intervalo (Gráfico 6.2). O erro exato máximo da

série de Maclaurin de grau $n = 4$ era $0.99e - 03 \cong O(10^{-2})$, e agora a série de Tchebychev de apenas grau $n = 3$ aproxima $f(x) = e^x$ com o erro exato máximo de $0.606e - 03$, ou seja, da mesma ordem de $O(10^{-2})$.

Com a aplicação do teorema de Tchebychev, podemos construir um aproximador de qualquer grau, bastando efetuar numericamente as integrações de Tchebychev (conforme o algoritmo **Cap6exem6.9.m**) e, depois, substituir os respectivos polinômios de Tchebychev para ter uma função em x .

Exemplo 6.10: aproxime, pelo teorema de Tchebychev e por Tchebychev-Maclaurin, a função $f(x) = \text{sen}(x)$ em $x \in [-1, +1]$, de modo que o erro máximo seja da ordem de $O(10^{-6})$.

Solução:

Usando a **Propriedade 5** (teorema de Tchebychev), com coeficientes obtidos via eqs. (11), obtemos aproximações até chegar ao grau final $n = 5$, de modo que o erro máximo seja da ordem de $O(10^{-6})$:

$$TC_5(T_i) = 0.880101171489875T_1 - 0.0391267079653386T_3 + 0.000499515460423624T_5$$

Em seguida, devemos *substituir os respectivos polinômios*⁹ $T_i(x)$ de Tchebychev, dados pelas eqs. (7a), para obter essa aproximação diretamente em função de x . Vamos também apresentar os erros no final.



Não estamos fazendo essa substituição neste momento para manter apenas cálculos em computador nesta fase de testes, sem necessidade de recorrer a substituições algébricas.

Alternativamente, partindo da série de Maclaurin de $\text{sen}(x)$, vamos obter a aproximação de Tchebychev-Maclaurin. Como:

$$f(x) = \text{sen}(x) = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \dots + (-1)^{i+1} \frac{x^{2i-1}}{(2i-1)!}$$

$$f(x) = \text{sen}(x) = 0 + \frac{x^1}{1!} + 0 - \frac{x^3}{3!} + 0 + \frac{x^5}{5!} + 0 - \frac{x^7}{7!} + 0 + \frac{x^9}{9!} + 0 + \dots + (-1)^{i+1} \frac{x^{2i-1}}{(2i-1)!}$$

Conforme o **Exemplo 6.5**, a série de Maclaurin de grau $n = 7$ tem limite de erro de truncamento da ordem $O(10^{-6})$, dado pelo primeiro termo abandonado $E_{T_1} = 2.755732e - 06$.

$$\text{sen}(x) \cong M_7(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

$$M_7(x) = 0 + x + 0 - \frac{x^3}{3!} + 0 + \frac{x^5}{5!} + 0 - \frac{x^7}{7!}$$

Substituindo algebricamente os x^i pelos polinômios de Tchebychev em T_i , temos:

$$M_7(T_i) \cong T_1 - \frac{(T_3 + 3T_1)/4}{3!} + \frac{(T_5 + 5T_3 + 10T_1)/16}{5!} - \frac{(T_7 + 7T_5 + 21T_3 + 35T_1)/64}{7!}$$

$$M_7(T_i) \cong \frac{8111}{9216} T_1 - \frac{601}{15360} T_3 + \frac{23}{46080} T_5 - \frac{1}{322560} T_7$$

Como $|T_7| \leq 1$, podemos truncar o termo $(1/322560) T_7 < 3.100 * 10^{-06}$, que é de ordem de grandeza do erro máximo da série de Maclaurin, $E_{T_1} = 2.755732e - 06$, e esses erros estimados somados não devem ultrapassar o limite da ordem de $O(10^{-6})$, mas $E_T \cong 2.756e - 06 + 3.100e - 06 = 5.856e - 06$, que é de ordem $O(10^{-5})$, assim mesmo vamos proceder ao truncamento de T_7 e conferir esses erros estimados através do cálculo dos erros exatos no final da aproximação. Assim, geramos a aproximadora Tchebychev-Maclaurin:

$$TC_5^M(T_i) = \frac{8111}{9216} T_1 - \frac{601}{15360} T_3 + \frac{23}{46080} T_5$$

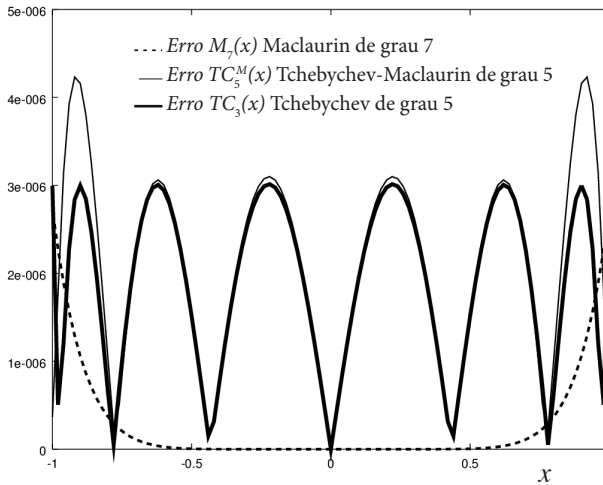
Substituindo T_i pelos polinômios de Tchebychev em x^i , temos:

$$TC_5^M(x) = \frac{8111}{9216} x^1 - \frac{601}{15360} (4x^3 - 3x) + \frac{23}{46080} (16x^5 - 20x^3 + 5x)$$

$$TC_5^M(x) = \frac{46079}{46080} x^1 - \frac{959}{5760} x^3 + \frac{23}{2880} x^5$$

$$TC_5^M(x) = 0 + \frac{46079}{46080} x^1 + 0 - \frac{959}{5760} x^3 + 0 + \frac{23}{2880} x^5$$

Gráfico 6.3 - Erros exatos entre as aproximadoras por série de Maclaurin $M_7(x)$, por Tchebychev-Maclaurin $TC_5^M(x)$ e Tchebychev $TC_5(x)$



Fonte: Elaboração própria.

Erro máximo Maclaurin de grau $n = 7$ é $2.7308e - 06 \cong O(10^{-6})$.

Erro máximo Tchebychev-Maclaurin de grau final $n = 5$ é $4.231483e - 06 \cong O(10^{-5})$ (nessa aproximação, o erro ficou um pouco acima da ordem $O(10^{-6})$).

Erro máximo Tchebychev de grau $n = 5$ é $3.013737e - 06 \cong O(10^{-6})$ (nessa aproximação, o erro ficou na ordem $O(10^{-6})$ com grau menor que Maclaurin).

Note que o erro máximo estimado inicialmente para a série de Tchebychev-Maclaurin $TC_5^M(x)$ era $5.856e - 06$, e o erro efetivamente obtido por Tchebychev-Maclaurin foi $4.231483e - 06$. O erro exato máximo da série de Maclaurin de grau $n = 7$ era $2.7308e - 06 \cong O(10^{-6})$, e agora a série de Tchebychev de apenas grau $n = 5$ aproxima $f(x) = \text{sen}(x)$ com o erro exato máximo de $3.013737e - 06$, ou seja da mesma ordem de $O(10^{-6})$ e obtido com uma série de menor grau. Nesse exemplo, a aproximação de Tchebychev de grau $n = 5$, obtida numericamente usando o teorema de Tchebychev, também promoveu uma melhor distribuição dos erros ao longo do intervalo (Gráfico 6.3).

Comparando a aproximação pelo teorema de Tchebychev com a forma algébrica de Tchebychev-Maclaurin, percebemos que a primeira gera erros menores, uma vez que os coeficientes b_i são obtidos a partir da série infinita, considerando as influências de **todos os termos da série** de Tchebychev e aplicando produtos internos em conjunto com a propriedade da ortogonalidade entre todos os polinômios T_i de Tchebychev da eq. (10a). Assim, chegaremos à série truncada de Tchebychev com o grau desejado a partir da série infinita, conforme os coeficientes dados pela eq. (10b). Na aproximação de Tchebychev-Maclaurin, partindo da série de Maclaurin truncada, consideramos apenas os termos não truncados da série.

Exemplo 6.11: aproxime $f(x) = \ln(x)$ em $x \in [1, 2]$, por Tchebychev, com erro máximo na ordem de $O(10^{-6})$, e compare com Maclaurin. Faça um gráfico com os erros.

Solução:

Primeiramente, devemos fazer a correspondente mudança de variáveis de $x \in [1, 2]$ para o intervalo padrão $t \in [-1, +1]$:

$$x(t) = \frac{(2-1)t + (2+1)}{2} = 0.5t + 1.5$$

Então, $f(x(t)) = \ln(x(t)) = \ln(0.5t + 1.5)$ no intervalo $t \in [-1, +1]$.

Aplicando o teorema de Tchebychev à função $f(x(t))$ e testando os graus necessários, chegamos ao erro máximo desejado com grau final $n = 6$:

$$TC_6(T_i) = 0.376452812919195T_0 + 0.343145750507622T_1 - 0.0294372515228597T_2 + 0.00336708925556424T_3 - 4.33275888610656e-04T_4 + 5.94707119897031e-05T_5 - 8.50296754122976e-06T_6$$

$$\text{Erro Tchebychev Max} = 1.4721e-06$$

Agora, aplicando a expansão em série de Maclaurin, teremos a aproximação para $f(t) = \ln\left(\frac{(b-a)t + (b+a)}{2}\right)$ com a e b genéricos:

$$f(t) = \ln(0.5(b+a)) + \frac{1}{1}\left(\frac{b-a}{b+a}\right)t^1 - \frac{1}{2}\left(\frac{b-a}{b+a}\right)^2 t^2 + \frac{1}{3}\left(\frac{b-a}{b+a}\right)^3 t^3 - \dots (-1)^{(i+1)} \frac{1}{i}\left(\frac{b-a}{b+a}\right)^i t^i + \dots$$

Como é uma série alternada nos sinais, podemos estimar o limite de erro tomando o primeiro termo abandonado, depois do grau n :

$$\max \left| (-1)^{(n+2)} \frac{1}{n+1} \left(\frac{b-a}{b+a}\right)^{n+1} t^{n+1} \right| \cong O(10^{-6}) \text{ em } t = 1.$$

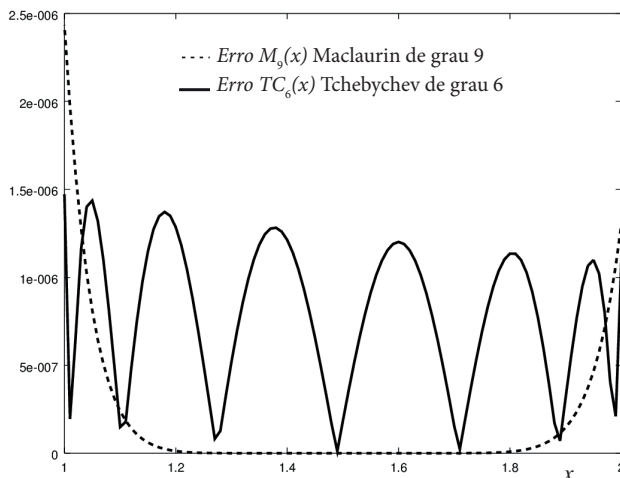
Para $n = 9$, o primeiro termo abandonado será de grau 10 e gerará o erro máximo $1.69350878 * 10^{-06}$.

Calculando os coeficientes de Maclaurin de grau $n = 9$, temos:

$$\begin{aligned} M_9(x) = & 4.05465108108164e-01 + 3.33333333333333e-01x \\ & - 5.55555555555556e-02x^2 + 1.23456790123457e-02x^3 \\ & - 3.08641975308642e-03x^4 + 8.23045267489712e-04x^5 \\ & - 2.28623685413809e-04x^6 + 6.53210529753740e-05x^7 \\ & - 1.90519737844841e-05x^8 + 5.64502926947676e-06x^9 \end{aligned}$$

Com Erro $M_9(x)$ Max = $2.4334e-06$

Gráfico 6.4 – Erros exatos das aproximadoras por série de Maclaurin $M_9(x)$ e de Tchebychev $TC_6(x)$



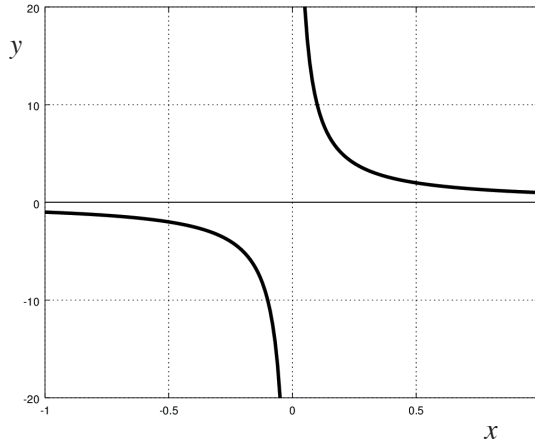
Fonte: Elaboração própria.

Então, com grau $n = 6$, a série de Tchebychev aproxima $f(x) = \ln(x)$ em $x \in [1, 2]$ com o mesmo erro máximo na ordem de $O(10^{-6})$ que uma série de Maclaurin de grau $n = 9$.

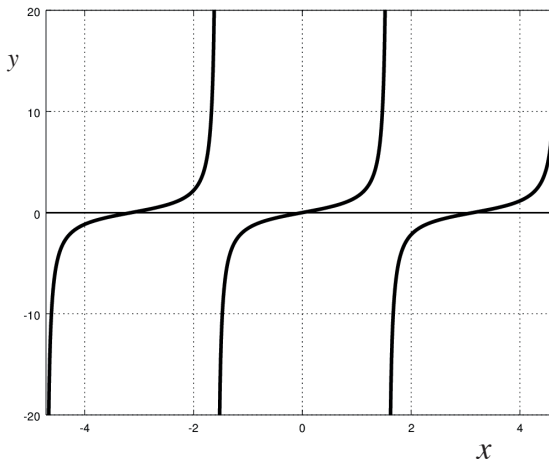
Apesar de todas as vantagens das aproximações polinomiais de $y = f(x)$, em especial as de Tchebychev, também existem desvantagens que são inerentes aos próprios polinômios aproximadores, por exemplo, seus gráficos podem se tornar sinuosos com o aumento do grau e normalmente não aproximam eficientemente funções assintóticas por possuírem comportamento não assintótico.

No **Caderno de Algoritmos**, você encontra implementados todos os exemplos deste capítulo, como **Cap6exem6.11.m**, no qual os polinômios de Tchebychev são calculados genericamente para qualquer grau, por meio da relação de recorrência dada pela eq. (7b).

Para funções $y = f(x)$ que sejam assíntotas como as funções dos Gráficos 6.5 e 6.6, o grau n de um aproximador polinomial será muito elevado, impactando diretamente nos requisitos (c) e (d) de uma aproximadora ideal.

Gráfico 6.5 – Função assintótica $y = 1/x$ 

Fonte: Elaboração própria.

Gráfico 6.6 – Função assintótica $y = tg(x)$ 

Fonte: Elaboração própria.

A seguir, vamos abordar uma técnica de aproximação de $y = f(x)$ utilizando aproximadoras racionais.

6.2 APROXIMAÇÃO RACIONAL DE PADÉ

Definição 2: uma **função racional** de grau total M é toda expressão do tipo

$$R_{nm}(x) = \frac{a_0 + a_1x + \dots + a_nx^n}{b_0 + b_1x + \dots + b_mx^m}, \text{ em que } M = n + m.$$

Por exemplo, a função hiperbólica $f(x) = \frac{1}{x}$ é uma racional de grau $M = 1$, em que $R_{01}(x) = \frac{P_0(x)}{Q_1(x)}$.

Definição 3: a **aproximação racional de Padé** de uma $y = f(x)$ consiste em obter uma $R_{nm}(x)$ tal que:

$$f(x) \cong R_{nm}(x) = \frac{a_0 + a_1x + \dots + a_nx^n}{b_0 + b_1x + \dots + b_mx^m} \quad (16)$$

Com as seguintes **condições de aproximação**:

$$f(0) = R_{nm}(0); f'(0) = R'_{nm}(0); f''(0) = R''_{nm}(0); \dots ; f^{(M)}(0) = R^{(M)}_{nm}(0) \quad (17)$$

em que $M = n + m$.

Note que essas condições de aproximação são exatamente as mesmas satisfeitas pelo aproximador de Taylor-Maclaurin, nesse caso, aplicadas para obter um polinômio de grau n , eq. (4a), enquanto no aproximador de Padé são aplicadas para determinar uma função racional de grau $M = n + m$, eq. (16).

Para obter a racional $R_{nm}(x)$ satisfazendo as condições da eq. (17), iniciamos aproximando a função original $f(x)$ por um aproximador de Maclaurin de grau M conforme os passos a seguir.

Primeiro passo: obtemos o aproximador de Maclaurin de grau $M = n + m$:

$$f(x) = M_M(x) + E_T$$

$$M_M(x) = c_0 + c_1x + c_2x^2 + \dots + c_Mx^M$$

com n e m estabelecidos previamente e $n = m$ ou $n = m + 1$.

Segundo passo: tomamos os polinômios $P_n(x) = a_0 + a_1x + \dots + a_nx^n$ e $Q_m(x) = 1 + b_1x + \dots + b_mx^m$ com os coeficientes a_k e b_j a serem determinados. Fixamos o primeiro coeficiente em $b_0 = 1$ (sem perda de generalidade) e consideramos que $M_M(x) = R_{nm}(x)$, o que resulta em:

$$c_0 + c_1x + c_2x^2 + \dots + c_Mx^M = \frac{a_0 + a_1x + \dots + a_nx^n}{1 + b_1x + \dots + b_mx^m} \quad (18)$$

Terceiro passo: aplicamos as condições de aproximação da eq. (17) na eq. (18), inicialmente explicitando as incógnitas a_k em função dos c_i e b_j , o que resulta em:

$$a) \quad f(x=0) = R_{nm}(x=0)$$

$$\Rightarrow c_0 + 0 = \frac{a_0 + 0}{1 + 0} \Rightarrow c_0 = \frac{a_0}{1} \Rightarrow a_0 = c_0$$

$$b) \quad f'(x=0) = R'_{nm}(x=0)$$

$$\Rightarrow c_1 + 2c_2x^1 + \dots + M * c_Mx^{M-1} = (a_1 + 2a_2x^1 \dots + n * a_nx^{n-1})$$

$$* (1 + b_1x + \dots + b_mx^m)^{-1} - 1(a_0 + a_1x + \dots + a_nx^n)$$

$$* (1 + b_1x + \dots + b_mx^m)^{-2} (b_1 + 2b_2x^1 \dots + m * b_mx^{m-1})$$

$$\Rightarrow c_1 = (a_1)(1)^{-1} - 1(a_0)(1)^{-2} (b_1)$$

$$\Rightarrow a_1 = c_1 + b_1 * c_0$$

$$c) \quad f''(x=0) = R''_{nm}(x=0)$$

$$\Rightarrow 2c_2 + 6c_3x^1 + \dots + M * (M-1) * c_Mx^{M-2} =$$

$$* (2a_2 + 6a_3x^1 + \dots + n * (n-1) * a_nx^{n-2}) (1 + b_1x + \dots + b_mx^m)^{-1}$$

$$- 1(a_1 + 2a_2x^1 + \dots + n * a_nx^{n-1}) (1 + b_1x + \dots + b_mx^m)^{-2}$$

$$\begin{aligned}
& * (b_1 + 2b_2x^1 + 3b_2x^2 + \dots + m * b_m x^{m-1}) - 1(a_1 + \dots + a_n x^n) \\
& * (1 + b_1x + \dots + b_m x^m)^{-2} (b_1 + 2b_2x^1 + 3b_2x^2 + \dots + m * b_m x^{m-1}) \\
& + 2(a_0 + a_1x + \dots + a_n x^n) (1 + b_1x + \dots + b_m x^m)^{-3} (b_1 + 2b_2x + \dots + m * b_m x^{m-1}) \\
& * (b_1 + 2b_2x^1 + \dots + m * b_m x^{m-1}) - 1(a_0 + a_1x + \dots + a_n x^n) (1 + b_1x + \dots + b_m x^m)^{-2} \\
& * (2b_2 + 6b_3x^1 + \dots + m * (m-1) * b_m x^{m-2}) \\
\\
\Rightarrow 2c_2 &= (2a_2)(1)^{-1} - 1(a_1)(1)^{-2} (b_1) - 1(a_1)(1)^{-2} (b_1) \\
& + 2(a_0)(1)^{-3} (b_1)(b_1) - 1(a_0)(1)^{-2} (2b_2) \\
\\
\Rightarrow 2c_2 &= 2a_2 - 2a_1b_1 + 2a_0b_1^2 - 2a_0b_2 \quad (\div 2) \\
\\
\Rightarrow a_2 &= c_2 + a_1b_1 - a_0b_1^2 + a_0b_2
\end{aligned}$$

E substituindo a_1 e a_0 , em a_2 , temos

$$\begin{aligned}
\Rightarrow a_2 &= c_2 + (b_1c_0 + c_1)b_1 - c_0b_1^2 + c_0b_2 = c_2 + c_0b_1^2 + c_1b_1 - c_0b_1^2 + c_0b_2 \\
\Rightarrow a_2 &= c_2 + b_1c_1 + b_2c_0
\end{aligned}$$

Estendendo para as demais derivadas, até ordem M , explicitamos os valores de a_k :

$$\left\{ \begin{array}{l} a_0 = c_0 \\ a_1 = c_1 + b_1c_0 \\ a_2 = c_2 + b_1c_1 + b_2c_0 \\ a_3 = c_3 + b_1c_2 + b_2c_1 + b_3c_0 \\ a_4 = c_4 + b_1c_3 + b_2c_2 + b_3c_1 + b_4c_0 \\ a_5 = c_5 + b_1c_4 + b_2c_3 + b_3c_2 + b_4c_1 + b_5c_0 \\ \vdots \\ a_n = c_n + b_1c_{n-1} + \dots + b_n c_{n-m} \end{array} \right. \quad (19)$$

E os valores dos b_j são obtidos gerando e resolvendo o sistema linear:

$$\begin{bmatrix} c_{n-m+1} & c_{n-m+2} & \cdots & c_n \\ c_{n-m+2} & c_{n-m+3} & \cdots & c_{n+1} \\ & \vdots & & \\ c_n & c_{n+1} & \cdots & c_{n+m-1} \end{bmatrix} \begin{bmatrix} b_m \\ b_{m-1} \\ \vdots \\ b_1 \end{bmatrix} = - \begin{bmatrix} c_{n+1} \\ c_{n+2} \\ \vdots \\ c_{n+m} \end{bmatrix} \quad (20a)$$

Para padronizar a apresentação do sistema da eq. (20a), conforme definimos no Capítulo 2, com as incógnitas contendo índices na ordem crescente e sem afetar a sua solução, podemos efetuar a troca das colunas da matriz dos coeficientes resultando no sistema equivalente:

$$\begin{bmatrix} c_n & c_{n-1} & \cdots & c_{n-m+1} \\ c_{n+1} & c_n & \cdots & c_{n-m+2} \\ & \vdots & & \\ c_{n+m-1} & c_{n+m-2} & \cdots & c_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = - \begin{bmatrix} c_{n+1} \\ c_{n+2} \\ \vdots \\ c_{n+m} \end{bmatrix} \quad (20b)$$

Note que a eq. (20a) forma um sistema linear simétrico de ordem $m \times m$. Se nessas equações ocorrer c_i com i negativo, então consideramos $c_i = 0$.

Primeiramente temos que resolver a eq. (20a), ou a (20b), para obter os b_j , e, depois, substituí-los nas eqs. (19), para obter os a_k .

Esses sistemas são muito mal condicionados e, nesses casos, temos um acúmulo significativo de erros de arredondamento nas soluções obtidas por métodos diretos, conforme efeitos citados no Capítulo 2. Assim, é recomendável utilizar o método de Cholesky (como no **Exemplo 6.16**) ou, se não for possível, utilizar o método de Gauss com pivotação total (como no **Exemplo 6.12**, no qual é mandatória uma pivotação, conseqüentemente o método de Cholesky não pode mais ser aplicado, uma vez que a matriz torna-se não simétrica).

A obtenção desses sistemas, dados pelas eqs. (19) e (20), pela aplicação direta das condições de aproximação, conforme a eq. (17), é extremamente trabalhosa. Alternativamente, podemos obter esses sistemas considerando as condições de aproximação dadas pela eq. (17), reescritas da seguinte forma,

$$f^{(r)}(x=0) - R_{nm}^{(r)}(x=0) = 0, \quad \forall r = 0, 1, 2, \dots, M \quad (21)$$

Observe que, na forma da eq. (21), podemos inferir que “ $x = 0$ é a única raiz da equação $f(x) - R_{nm}(x) = 0$ com multiplicidade $M + 1$ ”, conforme a **Propriedade 13**, que vimos no Capítulo 3, pois tem derivadas nulas até ordem M .

Vamos mostrar essa dedução alternativa das eqs. (19) e (20), conforme apresentado em Burden e Faires (2011), reescrevendo a eq. (18) da seguinte forma,

$$f(x) - R_{nm}(x) = c_0 + c_1x + c_2x^2 + \dots + c_Mx^M - \frac{a_0 + a_1x + \dots + a_nx^n}{1 + b_1x + \dots + b_mx^m} = 0$$

$$f(x) - R_{nm}(x) = \frac{(c_0 + c_1x + c_2x^2 + \dots + c_Mx^M)(1 + b_1x + \dots + b_mx^m) - (a_0 + a_1x + \dots + a_nx^n)}{(a_0 + a_1x + \dots + a_nx^n)} = 0$$

Essa equação também é satisfeita se apenas o seu numerador $h(x)$ for nulo, ou seja, se:

$$h(x) = (c_0 + c_1x + c_2x^2 + \dots + c_Mx^M)(1 + b_1x + \dots + b_mx^m) - (a_0 + a_1x + \dots + a_nx^n) = 0 \quad (22)$$

Então, se $x = 0$ é raiz de $h(x) = 0$, nesse caso $x = 0$ também será raiz de $f(x) - R_{nm}(x) = 0$ e será uma raiz de no mínimo multiplicidade M . Se os termos de $h(x)$ de graus menores ou iguais a M forem nulos, então todas as derivadas de $h(x)$ serão nulas até ordem M ; e, a partir da derivada de ordem $M + 1$, as derivadas podem ser não nulas, conforme a **Propriedade 13** do Capítulo 3, reescrita para $h(x)$ a seguir,

$$\begin{cases} h^{(r)}(x=0) = 0, \quad \forall r = 0, 1, 2, \dots, M \\ h^{(M+1)}(x=0) \neq 0 \end{cases}$$

Logo, $h(x)$ deve ter termos nulos de ordem 0 até $M = n + m$ e termos não nulos de ordem $M + 1 = n + m + 1$ em diante até $M + m = n + 2m$, conforme,

$$h(x) = \left(\sum_{i=0}^0 c_i b_{0-i} - a_0 \right) x^0 + \left(\sum_{i=0}^1 c_i b_{1-i} - a_1 \right) x^1 + \left(\sum_{i=0}^2 c_i b_{2-i} - a_2 \right) x^2 + \dots + \left(\sum_{i=0}^n c_i b_{n-i} - a_n \right) x^n \\ + \left(\sum_{i=0}^{n+1} c_i b_{n+1-i} - a_{n+1} \right) x^{n+1} + \left(\sum_{i=0}^{n+2} c_i b_{n+2-i} - a_{n+2} \right) x^{n+2} + \dots + \left(\sum_{i=0}^{n+m} c_i b_{n+m-i} - a_{n+m} \right) x^{n+m} + O(x^{n+m+1}) = 0$$

em que $O(x^{n+m+1})$ reúne os termos não nulos de ordem entre $M + 1$ e $M + m$.

Com $b_j = 0$, para índices $j > m$ e $a_k = 0$ para índices $k > n$, a fim de manter os respectivos graus m e n no denominador e no numerador de R_{nm} , conforme a eq. (16).

Logo, genericamente, devemos resolver o sistema a seguir,

$$\left\{ \sum_{i=0}^k c_i b_{k-i} - a_k = 0, \text{ com } k = 0, 1, 2, \dots, M \right. \quad (23)$$

Como $a_k = 0$ para índices $k > n$, devemos separar o sistema dado na eq. (23) nos dois subsistemas a seguir:

$$\left\{ \sum_{i=0}^k c_i b_{k-i} = 0, \text{ para } k = n + 1, n + 2, \dots, M \right. \quad (24)$$

$$\left\{ a_k = \sum_{i=0}^k c_i b_{k-i}, \text{ para } k = 0, 1, 2, \dots, n \right. \quad (25)$$

Para resolver esses dois subsistemas, precisamos resolver primeiramente a eq. (24), obtendo os b_j , e, depois, substituí-los na eq. (25), obtendo os a_k , como nas eqs. (20) e (19).

A seguir, vamos apresentar um exemplo de aproximação racional.

Exemplo 6.12: obtenha a aproximadora racional de Padé $R_{32}(x)$ para $f(x) = \arctg(x)$, $x \in [-1, +1]$. Avalie o erro da série de Maclaurin de grau 5 e o erro exato da $R_{32}(x)$ no intervalo dado.

Solução:

Temos $n = 3$, $m = 2$ e $M = 5$ com $f(x) = \arctg(x)$, cuja série de Maclaurin de grau $M = 5$ é:

$$f(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots + (-1)^{n+1} \underbrace{\frac{x^{2n-1}}{2n-1}}_{E_T} + \dots$$

O erro de truncamento máximo para essa série pode ser calculado pelo teorema do resto da série de Taylor, em que,

$$R_5(x) \leq \frac{\max_{x \in [-1, +1]} |f^{(5+1)}(x)| (x-0)^{5+1}}{(5+1)!}$$

$$\max_{x \in [-1, +1]} |f^{(5+1)}(x)| = |-240x(3 - 10x^2 + 3x^4) / (1 + x^2)^6| = 100.459, \text{ em } x = -0.228243$$

$$R_5(x) \leq \frac{100.459(1-0)^{5+1}}{(5+1)!} \leq 0.139526388888$$

Logo, $E_{T_1} = 0.139526389$.

Alternativamente, pelo Teorema 1, o erro de truncamento estimado na série de Maclaurin é o primeiro termo não nulo abandonado $E_{T_1} = \left| -\frac{x^7}{7} \right|_{x \in [-1, +1]}$ para séries alternadas nos sinais, dado por $E_{T_1} = 1/7 = 0.14286\dots$

Observe que temos a mesma ordem do erro máximo estimado pelo teorema do resto.

$$f(x) = \arctg(x) \cong M_5(x) = 0 + x + 0x^2 - \frac{1}{3}x^3 + 0x^4 + \frac{1}{5}x^5$$

$$\begin{aligned} c_0 &= 0 & c_3 &= -1/3 \\ \Rightarrow c_1 &= 1 & c_4 &= 0 \\ c_2 &= 0 & c_5 &= 1/5 \end{aligned}$$

Conforme a eq. (15),

$$c_0 + c_1x + c_2x^2 + c_3x^3 + c_4x^4 + c_5x^5 = \frac{P_3(x)}{Q_2(x)}$$

em que

$$P_3(x) = a_0 + a_1x + a_2x^2 + a_3x^3 \quad \text{e} \quad Q_2(x) = 1 + b_1x + b_2x^2$$

Para determinar os coeficientes b_j , aplicamos a eq. (20a):

$$\begin{bmatrix} c_2 & c_3 \\ c_3 & c_4 \end{bmatrix} \begin{bmatrix} b_2 \\ b_1 \end{bmatrix} = - \begin{bmatrix} c_4 \\ c_5 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & -1/3 \\ -1/3 & 0 \end{bmatrix} \begin{bmatrix} b_2 \\ b_1 \end{bmatrix} = - \begin{bmatrix} 0 \\ 1/5 \end{bmatrix} \Rightarrow \begin{cases} b_2 = 3/5 \\ b_1 = 0 \end{cases}$$



Nesse sistema simétrico, temos coeficiente nulo na diagonal principal e o método de Cholesky não pode ser aplicado diretamente, por isso precisamos aplicar uma pivotação, que normalmente quebra a simetria da matriz, e depois aplicar outro método eliminativo como o Gauss.

Ou aplicamos a eq. (20b) usando o método de Cholesky, pois, nesse caso, a matriz ainda se manteve simétrica:

$$\begin{bmatrix} c_3 & c_2 \\ c_4 & c_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = - \begin{bmatrix} c_4 \\ c_5 \end{bmatrix} \Rightarrow \begin{bmatrix} -1/3 & 0 \\ 0 & -1/3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = - \begin{bmatrix} 0 \\ 1/5 \end{bmatrix} \Rightarrow \begin{cases} b_1 = 0 \\ b_2 = 3/5 \end{cases}$$

E, depois, aplicando b_j na eq. (19), obtemos a_k :

$$\begin{cases} a_0 = c_0 \\ a_1 = c_1 + b_1 c_0 \\ a_2 = c_2 + b_1 c_1 + b_2 c_0 \\ a_3 = c_3 + b_1 c_2 + b_2 c_1 + b_3 c_0 \end{cases} \text{ para } m = 2 \Rightarrow b_3 = 0, \text{ pois } Q_2(x) = 1 + b_1 x + b_2 x^2 + 0x^3$$

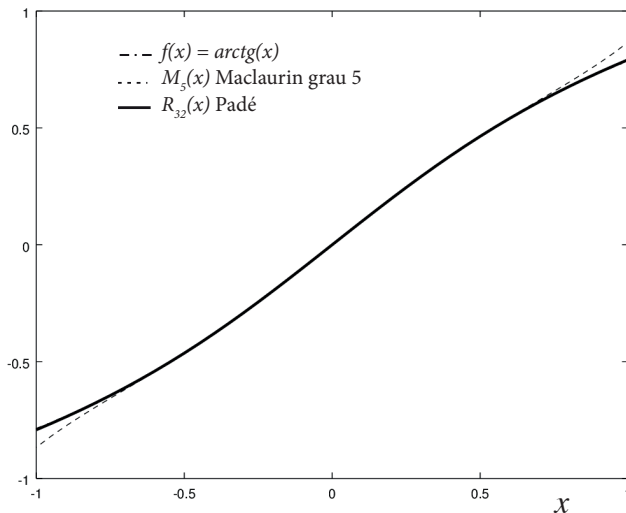
$$\begin{cases} a_0 = 0 \\ a_1 = 1 + 0 \cdot 0 = 1 \\ a_2 = 0 + 0 \cdot 1 + (3/5) \cdot 0 = 0 \\ a_3 = -1/3 + 0 \cdot 0 + (3/5) \cdot 1 + 0 \cdot 0 = 4/15 \end{cases}$$

Então,

$$f(x) = \operatorname{arctg}(x) \cong R_{32}(x) = \frac{0 + 1x + 0x^2 + (4/15)x^3}{1 + 0x + (3/5)x^2} = \frac{x(15 + 4x^2)}{15 + 9x^2}$$

Confira no **Caderno de Algoritmos**, o algoritmo de Padé no arquivo **Cap6exemplo6.12.m** aplicado ao **Exemplo 6.12**.

Gráfico 6.7 – Representação de $f(x) = \operatorname{arctg}(x)$ e suas aproximadoras por Maclaurin $M_5(x)$ e Padé $R_{32}(x)$ (a curva em traço-ponto da função exata está sob a curva em linha contínua)



Fonte: Elaboração própria.

A distribuição de erros das aproximadoras por Maclaurin e Padé em relação à função exata $f(x) = \arctg(x)$ pode ser conferida, a seguir, na Tabela 6.2 e no Gráfico 6.8.

Erro exato $M_5(x) = |M_5(x) - \arctg(x)|$ ($M_5(x)$ obtido por série Maclaurin com $n = 5$).

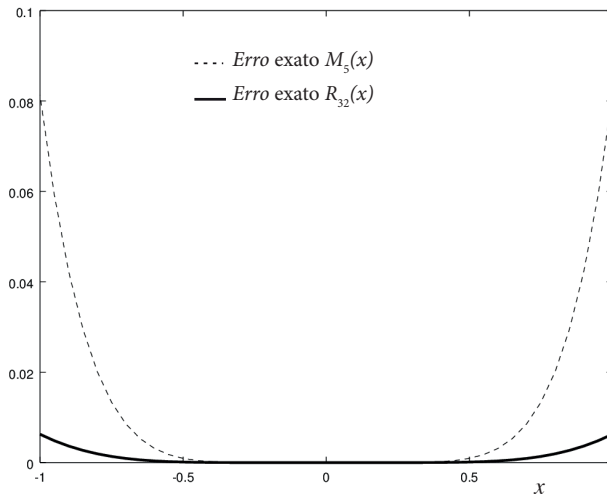
Erro exato $R_{32}(x) = |R_{32}(x) - \arctg(x)|$ ($R_{32}(x)$ obtido por Padé com $n = 3$ e $m = 2$).

Tabela 6.2 – Erros das aproximadoras por Maclaurin e Padé

x	-1	-0.5	0.0	0.5	1.0
<i>Erro exato</i> $M_5(x)$	0.081269	0.000936	0.0	0.000936	0.081269
<i>Erro exato</i> $R_{32}(x)$	0.006268	0.0001205	0.0	0.0001205	0.0062685

Fonte: Elaboração própria.

Gráfico 6.8 – Representação dos erros das aproximadoras por Maclaurin $M_5(x)$ e Padé $R_{32}(x)$



Fonte: Elaboração própria.

Note que os erros da aproximação de Padé são significativamente menores do que os erros da série de Maclaurin que gerou essa aproximação (ambas de grau $n = 5$), no entanto não estão uniformemente distribuídos, mantendo as características da série geradora, conforme o Gráfico 6.8.

Para a aproximação de Maclaurin com grau $n = 5$, temos um erro de truncamento máximo estimado de $E_{T_1} = 0.139526389$ que, se avaliado de forma exata, chega a 0.081269, conforme a Tabela 6.2, enquanto o erro máximo exato obtido com a aproximação de Padé $R_{3_2}(x)$ é no máximo 0.0062685. Para atingir um erro de truncamento dessa mesma ordem de 0.006, seria necessário uma série de Maclaurin de grau $n = 165$, enquanto na aproximação de Padé foi usado apenas grau total $M = 5$.

A seguir, vamos apresentar um exemplo comparativo das aproximações de funções com expressão conhecida apresentadas até este momento com o objetivo de validar a aproximação racional de Padé para os casos de funções assintóticas. Também vamos obter aproximações sucessivas da $f(x) = \ln(x)$, no domínio $(0, 1]$, buscando abranger as regiões próximas do ponto $x = 0$ onde a função se torna assintótica.

Exemplo 6.13: aproxime a $f(x) = \ln(x)$, no intervalo $x \in [0.1, 1]$, usando interpolador polinomial, série de Maclaurin, de Tchebychev e de Padé, de modo que os erros máximos sejam, se possível, da ordem de $O(10^{-6})$. Plote os gráficos dos erros locais, apresente os erros máximos atingidos em cada caso, compare os resultados e indique qual é a forma de aproximação mais eficiente para esta $f(x)$.

Solução:

O intervalo $x \in [0.1, 1]$ precisa ser transformado para o intervalo padrão $t \in [-1, +1]$, gerando a função $f(t) = \ln(0.45t + 0.55)$.

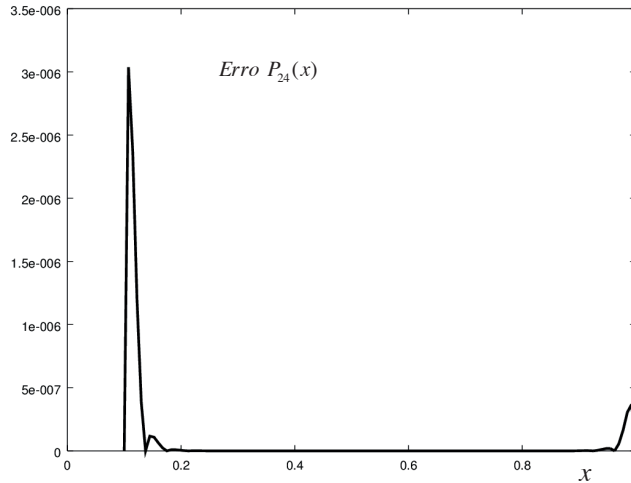
Aplicando tentativas para cada método, através dos respectivos algoritmos, teremos os seguintes resultados para os erros máximos e locais:

a) Para o **interpolador polinomial**:

grau 24 \rightarrow $Erro P_{24} Max = 3.03621213415539e - 06$ ($O(10^{-6})$).

Com erros locais conforme o Gráfico 6.9 a seguir:

Gráfico 6.9 – Erro local do interpolador polinomial P_{24} de grau 24



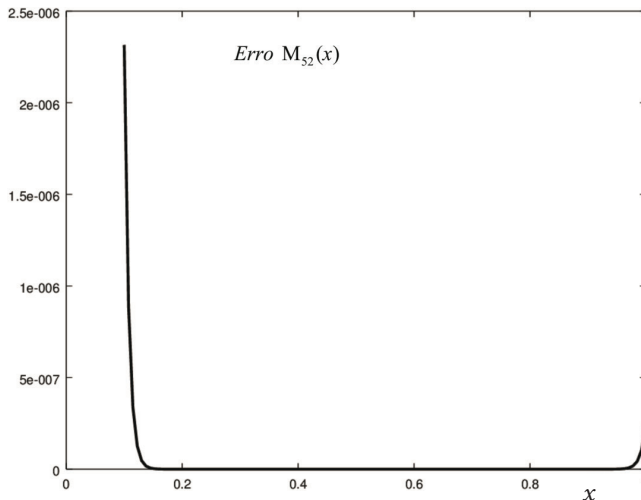
Fonte: Elaboração própria.

b) Para as **séries de Maclaurin**:

grau 52 \rightarrow $Erro M_{52} Max = 2.31509887838044e - 06$ ($O(10^{-6})$).

Com erros locais conforme o Gráfico 6.10 a seguir:

Gráfico 6.10 – Erro local de Maclaurin M_{52} grau 52



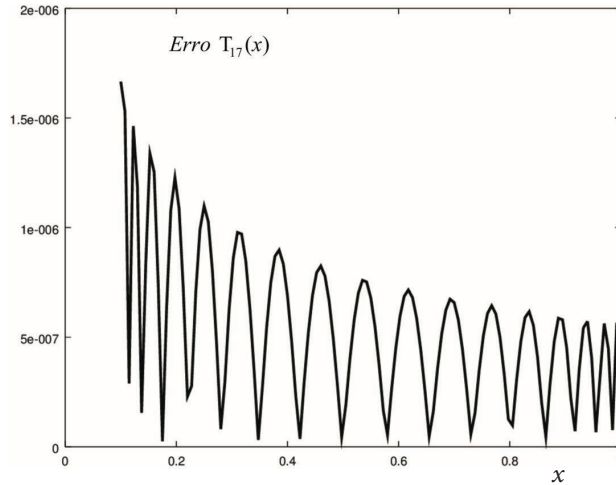
Fonte: Elaboração própria.

c) Para a **série de Tchebychev**:

grau 17 \rightarrow $Erro T_{17} Max = 1.66541020307776e - 06$ ($O(10^{-6})$).

Com erros locais conforme o Gráfico 6.11 a seguir:

Gráfico 6.11 – Erro local de Tchebychev T_{17} de grau $n = 17$



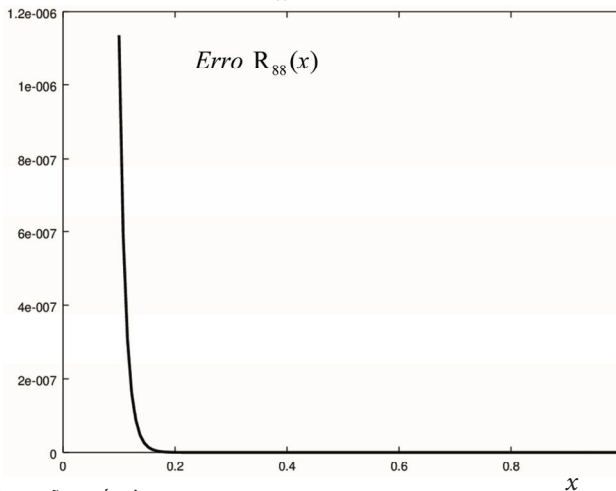
Fonte: Elaboração própria.

d) Para a **racional de Padé**:

grau $M = 8 + 8 \rightarrow$ $Erro R_{88} Max = 1.13394381573428e - 06$ ($O(10^{-6})$).

Com erros locais conforme o Gráfico 6.12 a seguir:

Gráfico 6.12 – Erro local de Padé $R_{88}(x)$



Fonte: Elaboração própria.

Observe que os erros máximos ocorrem na região mais próxima do ponto $x = 0$, que é onde a função se torna assintótica. Para esses casos, a aproximação por funções racionais de Padé se torna mais eficiente, pois aproxima $f(x)$ com os menores graus entre as aproximações testadas.

A seguir, vamos apresentar aproximações para a mesma função, mas avançando na direção do ponto $x = 0$ onde a função se torna mais assintótica, tomando o intervalo $[0.01, 0.1]$.

Exemplo 6.14: aproxime $f(x) = \ln(x)$, no intervalo $x \in [0.01, 0.1]$, com interpolador polinomial, série de Maclaurin, de Tchebychev e de Padé, de modo que os erros máximos sejam, se possível, da ordem de $O(10^{-6})$. Imprima os erros máximos atingidos em cada caso, compare os resultados e indique qual é a forma de aproximação mais eficiente.

Solução:

O intervalo $x \in [0.01, 0.1]$ precisa ser transformado para o intervalo padrão $t \in [-1, +1]$, gerando a função $f(t) = \ln(0.045t + 0.055)$.

Aplicando tentativas para cada método, teremos:

a) Para o **interpolador polinomial:**

$$\text{grau } 24 \rightarrow \text{Erro } P_{24} \text{ Max} = 3.03622053099417e - 06 \text{ (} O(10^{-6}) \text{)}.$$

b) Para as **séries de Maclaurin:**

$$\text{grau } 52 \rightarrow \text{Erro } M_{52} \text{ Max} = 2.31509887882453e - 06 \text{ (} O(10^{-6}) \text{)}.$$

c) Para a **série de Tchebychev:**

$$\text{grau } 17 \rightarrow \text{Erro } T_{17} \text{ Max} = 1.66541021329181e - 06 \text{ (} O(10^{-6}) \text{)}.$$

d) Para a **racional de Padé:**

$$\text{grau } M = 8 + 8 \rightarrow \text{Erro } R_{88} \text{ Max} = 1.13394346801243e - 06 \text{ (} O(10^{-6}) \text{)}.$$

Também para esse subintervalo, a aproximação por funções racionais de Padé se torna mais eficiente, pois aproxima $f(x)$ com os menores graus.

Vamos tomar mais um subintervalo para a mesma função, avançando mais na direção do ponto em que a função se torna assintótica, agora aproximando em $[0.001, 0.01]$.

Exemplo 6.15: aproxime $f(x) = \ln(x)$, no intervalo $x \in [0.001, 0.01]$, com interpolador polinomial, série de Maclaurin, de Tchebychev e de Padé, de modo que os erros máximos sejam, se possível, da ordem de $O(10^{-6})$. Imprima os erros máximos atingidos em cada caso, compare os resultados e indique qual é a forma de aproximação mais eficiente.

Solução:

O intervalo $x \in [0.001, 0.01]$ precisa ser transformado para o intervalo padrão $t \in [-1, +1]$, gerando a função $f(t) = \ln(0.0045t + 0.0055)$.

Aplicando sucessivas tentativas para cada método, teremos:

a) Para o **interpolador polinomial:**

$$\text{grau } 24 \rightarrow \text{Erro } P_{24} \text{ Max} = 3.03621929909070e - 06 (O(10^{-6})).$$

b) Para as **séries de Maclaurin:**

$$\text{grau } 52 \rightarrow \text{Erro } M_{52} \text{ Max} = 2.31509887793635e - 06 (O(10^{-6})).$$

c) Para a **série de Tchebychev:**

$$\text{grau } 17 \rightarrow \text{Erro } T_{17} \text{ Max} = 1.66541022306177e - 06 (O(10^{-6})).$$

d) Para a **racional de Padé:**

$$\text{grau } M = 8 + 8 \rightarrow \text{Erro } R_{88} \text{ Max} = 1.13394295997438e - 06 (O(10^{-6})).$$

Também para esse subintervalo, a aproximação por funções racionais de Padé se torna mais eficiente, pois aproxima $f(x)$ com os menores graus.

Observe que os graus necessários para aproximar $f(x) = \ln(x)$, nos sucessivos subintervalos, foram praticamente os mesmos, mas os subintervalos testados são complementares e dez vezes menores, cobrindo regiões mais próximas de $x = 0$, respectivamente nos **Exemplos 6.13, 6.14 e 6.15**, ou seja:

- a) para $x \in [0.1, 1]$, temos comprimento de intervalo 0.900;
- b) para $x \in [0.01, 0.1]$, temos comprimento de intervalo 0.090; e
- c) para $x \in [0.001, 0.01]$, temos comprimento de intervalo 0.009.

Para todos os casos testados de aproximação de funções assintóticas, as funções racionais de Padé foram as mais eficientes, ou seja, aproximam $f(x)$ com os menores graus possíveis, comprovando a característica principal desse tipo de aproximador.

Acesse o algoritmo do **Exemplo 6.13** no **Caderno de Algoritmos**, arquivo **Cap6exemplo6.13.m**, por meio do qual você poderá alterar o comprimento do intervalo e testar também os **Exemplos 6.14** e **6.15**.

A seguir, vamos avaliar a consequência do aumento sucessivo do grau M do aproximador de Padé.

Exemplo 6.16: aproxime $f(x) = \ln(x)$ em $x \in [+0.001, +1.0]$ por Padé com $n = m$ ou $n = m + 1$ para que o erro máximo seja, se possível, na ordem de $O(10^{-6})$. Imprima os erros máximos atingidos em cada teste. Compare a solução desse exemplo com os resultados dos **Exemplos 6.13**, **6.14** e **6.15**, do ponto de vista da eficiência computacional.

Solução:

Obtida via algoritmo **Cap6exem6.16.m** disponível no **Caderno de Algoritmos**:

$$\rightarrow \text{Erro } R_{8,8} \text{ Max} = 1.0442$$

$$\rightarrow \text{Erro } R_{16,16} \text{ Max} = 0.39409$$

$$\rightarrow \text{Erro } R_{32,32} \text{ Max} = 0.13720$$

$$\rightarrow \text{Erro } R_{64,64} \text{ Max} = 0.0030835$$

$$\rightarrow \text{Erro } R_{128,128} \text{ Max} = 0.0036341$$

$$\rightarrow \text{Erro } R_{256,256} \text{ Max} = 1.6256e - 04$$

Comparando o **Exemplo 6.16** com os **Exemplos 6.13**, **6.14** e **6.15**, percebemos que é computacionalmente muito mais eficiente trabalhar por partes com subintervalos menores do domínio de cálculo desejado, a fim de obter aproximadores de graus menores. Para o intervalo total $x \in [+0.001, +1.0]$ do **Exemplo 6.16**, serão necessários graus maiores do que $m = 256$

nos dois polinômios de Padé para atingirmos erros máximos de ordem $O(10^{-6})$, mas os sistemas lineares gerados a partir de $m = 256$ se tornam indeterminados ou até impossíveis de serem resolvidos, mesmo usando o método de Cholesky. Subdividindo, $x \in [+0.001, +1.0]$ em três subintervalos, $x \in [0.1, 1]$, $x \in [0.01, 0.1]$ e $x \in [0.001, 0.01]$, conforme os **Exemplos 6.13, 6.14 e 6.15**, serão necessários dois polinômios de apenas grau 8 para que os aproximadores de Padé tenham erros máximos de $O(10^{-6})$ em cada subintervalo.

Destacamos que não há uma forma conhecida de determinar previamente os valores de n e m para $R_{nm}(x)$ satisfazer a precisão desejada. Trata-se de um processo de tentativas e avaliação do erro. Porém, os melhores resultados ocorrem com $n = m$, se M for par; ou $n = m + 1$, se M for ímpar. Para exemplificar essa situação, vamos aproximar a $f(x) = e^x$ por Padé, com $M = 4$, tomando três combinações distintas de n e m e estimando os respectivos erros de truncamento máximos:

$$\text{a) } n = 2 \text{ e } m = 2 \Rightarrow e^x \cong R_{22}(x) = \frac{12 + 6x + x^2}{12 - 6x + x^2} \Rightarrow$$

$$\text{Erro } R_{22} \text{ Max} = 0.0039961$$

$$\text{b) } n = 4 \text{ e } m = 0 \Rightarrow e^x \cong R_{40}(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} \Rightarrow$$

$$\text{Erro } R_{40} \text{ Max} = 0.0099485 \text{ (Maclaurin do Exemplo 6.9)}$$

$$\text{c) } n = 0 \text{ e } m = 4 \Rightarrow e^x \cong R_{04}(x) = \frac{1}{1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \frac{x^4}{24}} \Rightarrow$$

$$\text{Erro } R_{04} \text{ Max} = 0.051615$$

Os resultados confirmam que, das três aproximações apresentadas, a R_{22} é a que apresentou menor erro de truncamento.

6.3 CONCLUSÕES

Em relação aos métodos de aproximação de uma função $y = f(x)$ com expressão conhecida e contínua no domínio $[a, b]$ de interesse, podemos concluir que a aproximação por série de Taylor e Maclaurin é suficiente e eficiente, para séries convergentes e com elevada velocidade de convergência. Mas se o gráfico da $y = f(x)$ tiver comportamento do tipo assintótico, devemos utilizar o aproximador racional de Padé. Para as funções não assintóticas e com séries de Maclaurin de convergência lenta, devemos empregar o aproximador de Tchebychev, especialmente devido ao seu acentuado efeito telescópico. Para as funções descontínuas do tipo “degrau” e periódicas no domínio de interesse, devemos aplicar a aproximação por séries de Fourier, não abordadas nesta obra, conforme apresentado em Burden e Faires (2011).

APROXIMAÇÃO PELA TENDÊNCIA VIA AJUSTE DE CURVAS

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- obter aproximadoras polinomiais definidas pela tendência de bases de dados;
- obter aproximadoras não polinomiais com coeficientes lineares e não lineares definidas pela tendência de bases de dados; e
- fazer uma análise introdutória da qualidade dos resultados.

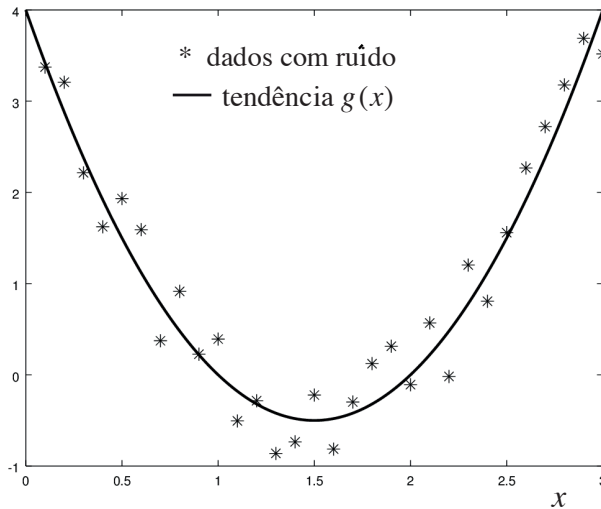
No Capítulo 5, abordamos a aproximação de uma função tabelada $(x_k, y_k = f(x_k))$ com $k = 1, 2, \dots, m$ por interpolação polinomial, *splines* cúbicas e curvas de Bézier. Agora veremos que a plotagem de tabelas oriundas de experimentos, de coleta de dados, ou de outros tipos de bases de dados, gera gráficos na forma de nuvens de pontos, também conhecidos como **diagrama de dispersão**, cuja configuração pode não ser uma função, mas globalmente sugira a **tendência** de uma função conhecida, bem como apresente os valores de y_k que podem estar afetados por erros inerentes (definidos no Capítulo 1). Nesse contexto, nenhuma das três técnicas de aproximação de funções tabeladas mencionadas será adequada para obter a função aproximadora sugerida pela tendência dos *dados*⁹.



Todos os dados utilizados neste capítulo e classificados como experimentais são fictícios.

Para visualizar como obtemos uma função aproximadora definida pela tendência de dados, observe o Gráfico 7.1.

Gráfico 7.1 – Exemplo de distribuição de dados (x_k, y_k) obtidos experimentalmente e uma função tendência $g(x)$



Fonte: Elaboração própria.

Observe que os m pontos do Gráfico 7.1 seguem uma tendência parabólica que caracteriza o seu comportamento. Nesses casos, é preferível escolher uma função que represente essa tendência em vez de tentar forçar que uma função interpoladora passe sobre todos os m pontos que genericamente podem nem ser uma função.

A aproximação de tendências desse tipo envolve quatro etapas.

Etapa 1: para aumentar o nível de confiança nas conclusões dos experimentos, geralmente temos que coletar a maior quantidade de amostras possível.

Etapa 2: os valores amostrais coletados podem estar afetados por erros inerentes (de observação, calibragem etc.), mas que não devem afetar a função aproximadora. Assim, quanto mais pontos forem coletados, maiores serão as chances de reduzir os efeitos dos erros das medições sobre a função aproximadora.

Etapa 3: a plotagem dos pontos pode sugerir uma tendência para uma função conhecida.

Etapa 4: depois da determinação da função tendência, temos que avaliar a sua real aderência com a base de dados, bem como validar o modelo e a quantidade das amostras utilizadas.

Definição 1: uma **função de ajuste** $z = g(x)$ é uma função de coeficientes parametrizados que mais se aproxima de todos os pontos amostrais (x_k, y_k) , $k = 1, 2, \dots, m$, e não necessariamente os contém.

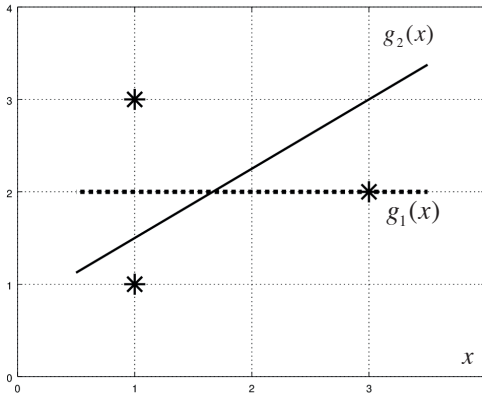
Para obter a função de ajuste, com o requisito “que mais se aproxima de todos os pontos”, recorreremos à minimização dos **desvios** (erros ou resíduos) entre os pontos amostrais observados (tabelados) e a função aproximadora, na qual o desvio de cada ponto é a distância:

$$d_k = g(x_k) - y_k \quad (1)$$

Uma alternativa seria tentar obter a $g(x)$ obedecendo a condição de aproximação:

desvio total $\sum_{k=1}^m d_k$ seja mínimo

Nesse critério, os valores dos d_k com sinais opostos compensam-se, podendo resultar num desvio total pequeno, mesmo com desvios individuais altos. Portanto, não é uma forma adequada, pois os desvios individuais também devem ser pequenos. Da mesma forma teríamos ambiguidades, isto é, existiriam várias funções que satisfariam essa alternativa para uma mesma tabela, e não distinguiríamos o bom ajuste do ruim, conforme o Gráfico 7.2, em que ambas as aproximadoras $g_1(x)$ e $g_2(x)$ geram desvios totais nulos.

Gráfico 7.2. – Duas aproximadoras, $g_1(x)$ e $g_2(x)$, com desvios totais nulosPara $g_1(x)$:

$$\sum_{k=1}^3 d_k = +1.0 - 1.0 + 0.0 = 0.0$$

Para $g_2(x)$:

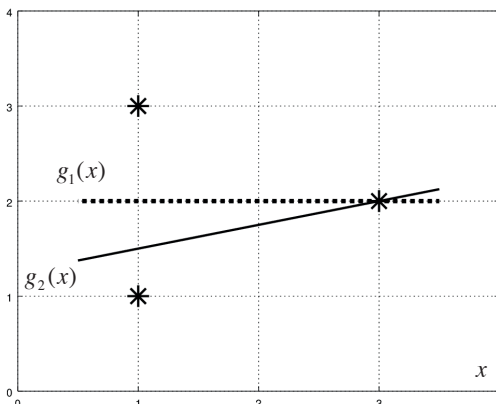
$$\sum_{k=1}^3 d_k = +0.5 - 1.5 + 1.0 = 0.0$$

Fonte: Elaboração própria.

Outra possibilidade seria eliminar o problema dos sinais usando o módulo dos desvios de modo que:

desvio absoluto total $\sum_{k=1}^m |d_k|$ seja mínimo

Esta alternativa também não permite que possamos distinguir o bom ajuste do ruim, pois pode ocorrer ambiguidade, conforme o Gráfico 7.3, bem como dificulta a minimização, pois a função módulo não é derivável no ponto de mínimo desvio.

Gráfico 7.3. – Duas aproximadoras, $g_1(x)$ e $g_2(x)$, com a mesma soma dos módulos dos desviosPara $g_1(x)$:

$$\sum_{k=1}^3 |d_k| = |+1.0| + |-1.0| + |0.0| = 2.0$$

Para $g_2(x)$:

$$\sum_{k=1}^3 |d_k| = |+0.5| + |-1.5| + |0.0| = 2.0$$

Fonte: Elaboração própria.

Note que, nesse critério, a $g_1(x)$ é a melhor aproximadora, pois é a que mais se aproxima de todos os pontos do diagrama de dispersão considerados.

Por fim, temos a alternativa:

soma quadrática dos desvios $\sum_{k=1}^m d_k^2$ seja mínima

Nesse caso, resulta que:

- desvios negativos (ao quadrado) são adicionados no desvio total;
- os grandes desvios são enfatizados;
- os pequenos desvios são minimizados; e
- o bom ajuste pode ser distinguido do ruim.

Nas duas retas de ajuste do Gráfico 7.3, temos que:

$$\text{Para } g_1(x): \sum_{k=1}^3 d_k^2 = (+1.0)^2 + (-1.0)^2 + (0.0)^2 = 2.0$$

$$\text{Para } g_2(x): \sum_{k=1}^3 d_k^2 = (+0.5)^2 + (-1.5)^2 + (0.0)^2 = 2.5$$

Observe que ocorreu menor desvio quadrático total na função $g_1(x)$, indicando que essa é a reta que mais se aproxima de todos os pontos da amostra.

Então, a essência do ajuste consiste na *minimização do desvio quadrático total D* , cuja expressão algébrica é:

$$D = \sum_{k=1}^m d_k^2 = \sum_{k=1}^m [g(x_k) - y_k]^2 \quad (2)$$



Observe que a condição de aproximação aplicada na interpolação, vista no Capítulo 5, é que os erros ou desvios locais sobre cada ponto sejam nulos, enquanto, no **ajuste, a soma dos quadrados dos erros, ou desvios locais, deve ser mínima**, e normalmente não é nula.

Por isso, a denominação dessa metodologia de aproximação por **mínimos quadrados**.

Neste capítulo, diferentemente dos demais, dispomos de um único método, uma vez que vamos obter o mínimo global da função desvio quadrático total D . As variações ocorrem apenas nos tipos de famílias a que pertencem as funções de ajuste $g(x)$ representativas da tendência do diagrama de dispersão.

A seguir, vamos apresentar o ajuste de bases de dados à família das funções polinomiais e posteriormente estendê-lo para as demais funções.

7.1 MÉTODO DOS MÍNIMOS QUADRADOS PARA AJUSTE A FUNÇÕES POLINOMIAIS

Para um diagrama de dispersão com m pontos:

x_k	x_1	x_2	...	x_m
y_k	y_1	y_2	...	y_m

e uma função representativa dada pelo polinômio $P_n(x)$ de grau n ($n < m$),

$$P_n(x) = a_1 + a_2x + \dots + a_nx^{n-1} + a_{n+1}x^n \quad (3)$$

resulta da eq. (1) que os desvios locais são

$$d_k = P_n(x_k) - y_k = a_1 + a_2x_k + \dots + a_nx_k^{n-1} + a_{n+1}x_k^n - y_k$$

e, da eq. (2), que o desvio quadrático total D é

$$D(a_1, a_2, \dots, a_{n+1}) = \sum_{k=1}^m \left[a_1 + a_2x_k + \dots + a_nx_k^{n-1} + a_{n+1}x_k^n - y_k \right]^2 \quad (4)$$

em que D é uma função com $n + 1$ variáveis (coeficientes $(a_1, a_2, \dots, a_{n+1})$), cujo **mínimo global** fornece os coeficientes $(a_1, a_2, \dots, a_{n+1})$ do polinômio ajustador.

Para minimizar $D(a_1, a_2, \dots, a_{n+1})$, primeiramente determinamos o seu **ponto crítico**, obtendo todas as suas derivadas parciais:

$$\frac{\partial D}{\partial a_1} = 0, \quad \frac{\partial D}{\partial a_2} = 0, \quad \dots, \quad \frac{\partial D}{\partial a_n} = 0$$

e gerando as equações:

$$\begin{aligned} \frac{\partial D}{\partial a_1} &= \sum_{k=1}^m 2 \left[a_1 + a_2 x_k + \dots + a_n x_k^{n-1} + a_{n+1} x_k^n - y_k \right] * 1 = 0 \\ \frac{\partial D}{\partial a_2} &= \sum_{k=1}^m 2 \left[a_1 + a_2 x_k + \dots + a_n x_k^{n-1} + a_{n+1} x_k^n - y_k \right] * x_k = 0 \\ &\vdots \\ \frac{\partial D}{\partial a_n} &= \sum_{k=1}^m 2 \left[a_1 + a_2 x_k + \dots + a_n x_k^{n-1} + a_{n+1} x_k^n - y_k \right] * x_k^{n-1} = 0 \\ \frac{\partial D}{\partial a_{n+1}} &= \sum_{k=1}^m 2 \left[a_1 + a_2 x_k + \dots + a_n x_k^{n-1} + a_{n+1} x_k^n - y_k \right] * x_k^n = 0 \end{aligned}$$

Dividindo as equações anteriores por 2 e aplicando a distributividade dos somatórios em cada equação, temos:

$$\left\{ \begin{aligned} \frac{\partial D}{\partial a_1} &= \sum_{k=1}^m a_1 + \sum_{k=1}^m a_2 x_k^1 + \dots + \sum_{k=1}^m a_n x_k^{n-1} + \sum_{k=1}^m a_{n+1} x_k^n - \sum_{k=1}^m y_k = 0 \\ \frac{\partial D}{\partial a_2} &= \sum_{k=1}^m a_1 x_k^1 + \sum_{k=1}^m a_2 x_k^2 + \dots + \sum_{k=1}^m a_n x_k^n + \sum_{k=1}^m a_{n+1} x_k^{n+1} - \sum_{k=1}^m x_k^1 y_k = 0 \\ &\vdots \\ \frac{\partial D}{\partial a_n} &= \sum_{k=1}^m a_1 x_k^{n-1} + \sum_{k=1}^m a_2 x_k^n + \dots + \sum_{k=1}^m a_n x_k^{2n-2} + \sum_{k=1}^m a_{n+1} x_k^{2n-1} - \sum_{k=1}^m x_k^{n-1} y_k = 0 \\ \frac{\partial D}{\partial a_{n+1}} &= \sum_{k=1}^m a_1 x_k^n + \sum_{k=1}^m a_2 x_k^{n+1} + \dots + \sum_{k=1}^m a_n x_k^{2n-1} + \sum_{k=1}^m a_{n+1} x_k^{2n} - \sum_{k=1}^m x_k^n y_k = 0 \end{aligned} \right. \quad (5)$$

Colocando os coeficientes a_i (eq(5)) em evidência, obtemos um sistema de $n + 1$ equações lineares a $n + 1$ incógnitas a_i , com $i = 1, 2, \dots, n + 1$ que, reescrito na forma matricial, torna-se:

$$\begin{bmatrix} m & \sum x_k^1 & \cdots & \sum x_k^{n-1} & \sum x_k^n \\ \sum x_k^1 & \sum x_k^2 & \cdots & \sum x_k^n & \sum x_k^{n+1} \\ & \vdots & & & \vdots \\ \sum x_k^n & \sum x_k^{n+1} & \cdots & \sum x_k^{2n-1} & \sum x_k^{2n} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n+1} \end{bmatrix} = \begin{bmatrix} \sum y_k \\ \sum x_k^1 y_k \\ \vdots \\ \sum x_k^n y_k \end{bmatrix} \quad (6)$$

Na eq. (6), cada somatório $\sum (\dots)$ representa $\sum_{k=1}^m (\dots)$ e

$$\sum_{k=1}^m a_1 = \underbrace{(a_1 + a_1 + \dots + a_1 + a_1)}_{m \text{ vezes}} = m * a_1.$$

A solução do sistema linear dado pela eq. (6) fornece os $n + 1$ **coeficientes** do $P_n(x)$ como o ponto crítico do desvio quadrático total $D = \sum_{k=1}^m d_k^2$. Esse conjunto de coeficientes é único sempre que $m \geq n + 1$, pois gera um sistema possível e determinado.



Não pode haver menos dados do que coeficientes a determinar para se obter uma única função ajustadora.

Esse ponto crítico é um **ponto de mínimo**, pois a função desvio quadrático total D gera matrizes menores principais da matriz **hessiana** de D com todos os determinantes positivos, de acordo com Chiang (1984), como mostraremos a seguir para o ajuste polinomial:

$$H_1 = \left| \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_1^2} \right| = m > 0$$

$$H_2 = \left| \begin{array}{cc} \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_1^2} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_1 \partial a_2} \\ \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_2 \partial a_1} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_2^2} \end{array} \right| = \left| \begin{array}{cc} m & \sum_{k=1}^m x_k \\ \sum_{k=1}^m x_k & \sum_{k=1}^m x_k^2 \end{array} \right| > 0$$



Hessiana é a matriz gerada por todas as derivadas parciais de 2ª ordem de uma função de várias variáveis.

$$H_3 = \begin{vmatrix} \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_1^2} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_1 \partial a_2} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_1 \partial a_3} \\ \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_2 \partial a_1} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_2^2} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_2 \partial a_3} \\ \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_3 \partial a_1} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_3 \partial a_2} & \frac{\partial^2 D(a_1, \dots, a_{n+1})}{\partial a_3^2} \\ \vdots & & \end{vmatrix} = \begin{vmatrix} m & \sum_{k=1}^m x_k & \sum_{k=1}^m x_k^2 \\ \sum_{k=1}^m x_k & \sum_{k=1}^m x_k^2 & \sum_{k=1}^m x_k^3 \\ \sum_{k=1}^m x_k^2 & \sum_{k=1}^m x_k^3 & \sum_{k=1}^m x_k^4 \end{vmatrix} > 0$$

e assim por diante.

Como $H_1 > 0, H_2 > 0, \dots, H_{n+1} > 0$, então a matriz hessiana é **positiva definida**, e o ponto crítico obtido $(a_1, a_2, \dots, a_{n+1})$ é um **ponto de mínimo**.



Essa condição, que classifica o ponto crítico como ponto de mínimo, pode ser aplicada sobre o desvio quadrático total D para qualquer tipo de ajuste, seja com coeficientes lineares, ou não lineares, que veremos na seção 7.2.1.

Os menores principais da matriz hessiana demonstram que o ponto crítico determinado pela eq. (6) é um ponto de mínimo desvio quadrático total. Por contradição, esse ponto crítico não poderia ser um ponto de **máximo** desvio quadrático, uma vez que sempre seria possível obter uma nova função ajustadora mais afastada dos pontos experimentais, gerando um desvio quadrático total maior e, portanto, um ponto de máximo nunca é atingido.

A seguir, vamos apresentar um exemplo de ajuste de uma função linear, também conhecido como **regressão linear**, a um conjunto de pontos amostrais tabelados.

Exemplo 7.1: ajuste o conjunto de pontos tabelados a uma **reta**. Calcule os valores de $f(5), f(10)$, o desvio quadrático total D obtido, plote o gráfico contendo o diagrama de dispersão e a respectiva reta ajustada.

x_k	1	3	4	6	8
y_k	0	1	2	4	5

Solução:

Temos $m = 5$ pontos e o polinômio ajustador de grau $n = 1 \Rightarrow P_1(x) = a_1 + a_2x$.

Da eq. (6) com $n + 1 = 2$ incógnitas, resulta o sistema:

$$\begin{bmatrix} 5 & 22 \\ 22 & 126 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 12 \\ 75 \end{bmatrix} \Rightarrow \begin{cases} a_1 = -0.945205479452053 \\ a_2 = +0.760273972602739 \end{cases}$$

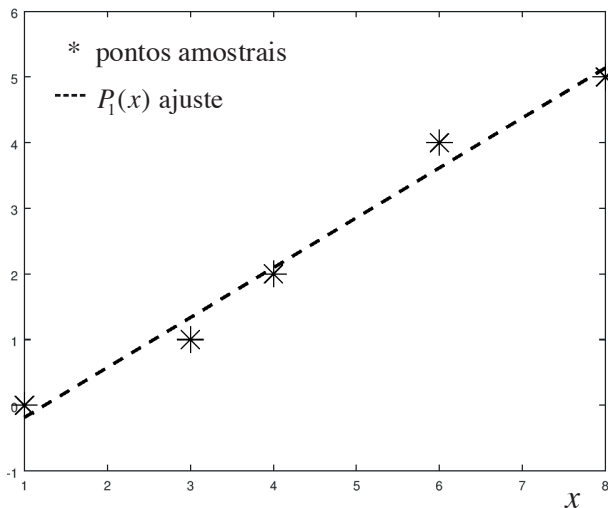
$$P_1(x) = -0.9452 + 0.7603x$$

$$f(5) \cong P_1(5) = 2.85$$

$$f(10) \cong P_1(10) = 6.65$$

$$D = \sum_{k=1}^5 (P_1(x_k) - y_k)^2 = 0.3219$$

Gráfico 7.4 – Comportamento gráfico do Exemplo 7.1



Fonte: Elaboração própria.

A seguir, confira algumas considerações importantes sobre o **Exemplo 7.1**.

A primeira questão a ser considerada é: **qual método é mais eficiente para resolver o sistema linear dado pela eq. (6)?**

Observe que esse sistema normalmente é de pequeno porte e formado por uma matriz densa, o que sugere a utilização de um método eliminativo para a resolução. Adicionalmente, note que é um sistema com a matriz dos coeficientes simétrica cujos elementos podem ser gerados pelas seguintes relações:

$$a_{ij} = \sum_{k=1}^m x_k^{i+j-2}$$

$$a_{i,n+1} = \sum_{k=1}^m (x_k^{i-1} * y_k)$$

e é uma matriz **positiva definida**. Portanto, o método mais eficiente de solução desse tipo de sistema é o de **Cholesky**, que utiliza apenas $O(n^3/6)$ operações aritméticas.

Outro ponto importante é que o sistema linear da eq. (6) é mal condicionado, como podemos verificar avaliando o seu grau de condicionamento, conforme vimos no Capítulo 2, por:

$$\|\det(A)\| = \frac{|\det(A)|}{\prod_{i=1}^{n+1} \alpha_i}, \text{ em que } \alpha_i = \sqrt{\sum_{j=1}^m a_{ij}^2}$$

No sistema do **Exemplo 7.1**, $\|\det(A)\| = 0.05059$, valor já relativamente pequeno, se comparado com a unidade, podendo ser considerado como mal condicionado. Logo, devemos utilizar preventivamente um método que envolva a menor quantidade de operações aritméticas possível para minimizar os efeitos dos arredondamentos como o método de Cholesky.

Como o $D = \sum_{k=1}^m d_k^2$ é um valor absoluto e, para uma tabela com m valores, podemos obter por mínimos quadrados até $m - 1$ polinômios de ajuste, $P_n(x)$, com graus $n = 1, 2, \dots, m - 1$, ou ainda a tabela pode ter

configuração com tendência não polinomial, devemos também utilizar um critério relativo de aferição de quão bem ajustada está a $g(x)$.

No **Exemplo 7.1**, temos que $\sum_{k=1}^m d_k^2 = 0.3219$, e, como é um valor absoluto, nada podemos afirmar sobre ele ser pequeno ou grande, isto é, **se a reta de ajuste é confiável ou não**. A única garantia que temos é que: a reta ajustada é a **melhor** entre as infinitas retas existentes.

Para quantificar a qualidade de uma regressão linear $g(x)$, obtida de uma base de dados (x_k, y_k) , $k = 1, 2, \dots, m$, podemos utilizar uma medida relativa denominada **coeficiente de determinação** (BARBETTA; REIS; BORNIA, 2010), cuja expressão na sua forma simplificada é dada por:

$$R^2 = 1 - \frac{\sum_{k=1}^m (g(x_k) - y_k)^2}{\sum_{k=1}^m (\bar{y} - y_k)^2} \quad (7)$$

em que \bar{y} é o valor médio dos valores de y_k , o termo $\sum_{k=1}^m (g(x_k) - y_k)^2$ corresponde à soma D dos quadrados dos desvios (também denominada de SQE), o termo $\sum_{k=1}^m (\bar{y} - y_k)^2$ corresponde à soma de quadrados totais (SQT), corrigida pela média aritmética, e a diferença $SQR = SQT - SQE$ é a soma de quadrados da regressão (BARBETTA; REIS; BORNIA, 2010).

Podemos interpretar esse coeficiente como a quantidade de variabilidade nos dados, que é explicada pelo modelo ajustado. Portanto, quanto maior o coeficiente de determinação (mais próximo da unidade), maior é a qualidade do ajuste do modelo aos dados (BARBETTA; REIS; BORNIA, 2010).

A seguir, vamos apresentar um exemplo de ajuste a uma função linear e a uma parabólica para introduzir uma análise de **qualidade** dos ajustes, que é conhecida como *Goodness of fit*.



Para saber mais sobre essa análise de **qualidade**, acesse o *link*: <http://www.mathworks.com/help/curvefit/evaluating-goodness-of-fit.html>. Acesso em: 7 jul. 2017.

Exemplo 7.2: para a base de dados:

x_k	0.00	0.25	0.50	0.75	1.00
y_k	1.00	1.32	1.79	1.64	1.41

Determine:

- a função de ajuste polinomial linear $n = 1$;
- a função de ajuste polinomial parabólico $n = 2$;
- os coeficientes de determinação de cada função ajustadora em relação ao pontos tabelados;
- os desvios quadráticos totais; e
- o gráfico com os m pontos da tabela, as duas funções ajustadoras e decida qual é o resultado mais confiável.

Solução:

- a) Ajuste polinomial linear: $n = 1$ e $m = 5 \Rightarrow P_1(x) = a_1 + a_2x$

Montando o sistema de equações dado pela eq. (6), temos

$$\begin{bmatrix} 5.0 & 2.5 \\ 2.5 & 1.875 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 7.1600 \\ 3.8650 \end{bmatrix} \Rightarrow \begin{cases} a_1 = 1.2040 \\ a_2 = 0.4560 \end{cases} \quad (\text{pelo Método}$$

Cholesky)

$P_1(x) = 1.2040 + 0.4560x \Rightarrow$ melhor reta possível para os $m = 5$ pontos.

- b) Ajuste polinomial parabólico: $n = 2$ e $m = 5 \Rightarrow$

$$P_2(x) = a_1 + a_2x + a_3x^2$$

Da eq. (6), temos

$$\begin{bmatrix} 5.0 & 2.5 & 1.875 \\ 2.5 & 1.875 & 1.5625 \\ 1.875 & 1.5625 & 1.3828125 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 7.1600 \\ 3.8650 \\ 2.8625 \end{bmatrix}$$

$$\Rightarrow \begin{cases} a_1 = 0.958285714285714 \\ a_2 = 2.421714285714286 \text{ (Cholesky)} \\ a_3 = -1.965714285714284 \end{cases}$$

$$P_2(x) = 0.958285714285714 + 2.421714285714286 x - 1.965714285714284 x^2$$

\Rightarrow melhor parábola possível para os $m = 5$ pontos

c) Coeficientes de Determinação:

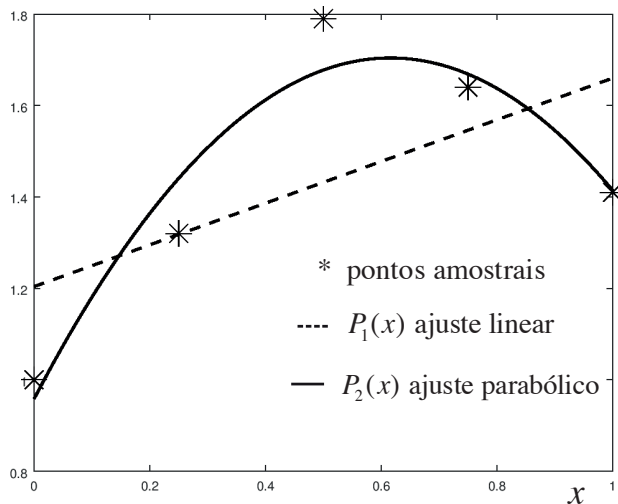
$$\begin{aligned} \text{Para a reta} & \quad \Rightarrow R_1^2 = 0.350220976608818 \\ \text{e para a parábola} & \quad \Rightarrow R_2^2 = 0.919678467484872 \end{aligned}$$

d) Desvios quadráticos totais:

$$\begin{aligned} \text{Para a reta} & \quad \Rightarrow D_1 = 0.24112 \\ \text{e para a parábola} & \quad \Rightarrow D_2 = 0.0298057142857143 \end{aligned}$$

e)

Gráfico 7.5 – Comportamento dos polinômios ajustados de grau 1 (linha tracejada) e grau 2 (linha contínua) do Exemplo 7.2



Fonte: Elaboração própria.

No **Exemplo 7.2**, o ajuste pela parábola resultou em melhores estatísticas (menor desvio quadrático total, maior coeficiente de determinação e gráfico consistente) em comparação com o ajuste pela reta. Lembremos que essa é uma avaliação preliminar, não pode ser considerada conclusiva, pois dispomos de poucos pontos amostrais.

Vários autores ainda recomendam uma análise qualitativa dos resíduos gerados pelos desvios locais d_k entre a função ajustadora e os pontos amostrais. Segundo Barbetta, Reis e Bornia (2010), para validar o modelo de ajuste e o intervalo de confiança que contém os pontos amostrais, seria desejável obter um diagrama de dispersão dos resíduos (desvios locais) com distribuição aleatória para diferentes intervalos da função aproximadora, possibilitando detectar alguma estrutura adicional nesses pontos (o que sugeriria uma função diferente) ou algum ponto discrepante que precisaria ser analisado a parte.

No **Caderno de Algoritmos**, disponível para *download* em <http://sergiopeters.prof.ufsc.br/livro-calculo-numerico-computacional/>, apresentamos o algoritmo **Cap7exemplo.7.2.m**, que ajusta uma base de dados com m pontos amostrais à função polinomial de grau n (n menor que m).

E como muitas bases de dados não possuem tendência polinomial, também vamos estudar o ajuste de pontos tabelados a funções não polinomiais.

7.2 AJUSTE POR MÍNIMOS QUADRADOS A FUNÇÕES NÃO POLINOMIAIS

Quando a plotagem dos pontos da base de dados:

x_k	x_1	x_2	...	x_m
y_k	y_1	y_2	...	y_m

sugerir uma tendência não polinomial, temos duas alternativas para obter a função de ajuste: determinação direta da ajustadora não polinomial ou tentar transformar a forma não polinomial em polinomial por meio de transformações paramétricas.

7.2.1 Determinação direta da ajustadora não polinomial

Como efetuamos para funções polinomiais, podemos deduzir diretamente o modelo matemático que irá fornecer a $g(x)$ representativa do comportamento dos pontos tabelados minimizando a eq. (2), conforme os casos a seguir:

- a) Ajuste de m pontos tabelados a uma função do tipo:

$$g(x) = a_1 x^2 + a_2 \ln(x).$$

Para obter os parâmetros a_1 e a_2 dessa função, procedemos a minimização direta do desvio quadrático total D :

$$D(a_1, a_2) = \sum_{k=1}^m d_k^2 = \sum_{k=1}^m [a_1 x_k^2 + a_2 \ln(x_k) - y_k]^2$$

Para minimizar $D(a_1, a_2)$, obtemos as suas derivadas parciais em relação a a_1 e a_2 , resultando em um sistema **linear**:

$$\frac{\partial D}{\partial a_1} = 0 \quad \text{e} \quad \frac{\partial D}{\partial a_2} = 0$$

em que

$$\frac{\partial D(a_1, a_2)}{\partial a_1} = \sum_{k=1}^m 2[a_1 x_k^2 + a_2 \ln(x_k) - y_k] x_k^2 = 0$$

$$\frac{\partial D(a_1, a_2)}{\partial a_2} = \sum_{k=1}^m 2[a_1 x_k^2 + a_2 \ln(x_k) - y_k] \ln(x_k) = 0$$

Evidenciando os coeficientes e reescrevendo na forma de equações lineares, temos

$$\begin{cases} (\sum x_k^4) a_1 + (\sum (x_k^2 \ln x_k)) a_2 = \sum (y_k x_k^2) \\ (\sum (x_k^2 \ln x_k)) a_1 + (\sum (\ln x_k)^2) a_2 = \sum (y_k \ln x_k) \end{cases}$$

Note que as duas equações formam um sistema linear porque a função aproximadora $g(x) = a_1 x^2 + a_2 \ln(x)$ é **linear em relação aos coeficientes** a_1 e a_2 e podemos tentar resolvê-lo via método de

Cholesky, uma vez que é uma matriz simétrica. Apenas não sabemos previamente se é positiva definida. Caso não seja, podemos usar a eliminação gaussiana.

b) Ajuste de m pontos tabelados a uma função do tipo

$$g(x) = a_1 x_k^2 + \text{sen}(a_2 x_k)$$

Procedendo a minimização direta do desvio quadrático total D , conforme a eq. (2), temos

$$D(a_1, a_2) = \sum_{k=1}^m d_k^2 = \sum_{k=1}^m [a_1 x_k^2 + \text{sen}(a_2 x_k) - y_k]^2$$

Para minimizar $D(a_1, a_2)$, obtemos as derivadas de D em relação a a_1 e a_2 e geramos um sistema **não linear** (não é possível colocar o coeficiente a_2 em evidência e transformá-lo em sistema linear):

$$\frac{\partial D(a_1, a_2)}{\partial a_1} = 2 \sum_{k=1}^m [a_1 x_k^2 + \text{sen}(a_2 x_k) - y_k] [x_k^2] = 0$$

$$\frac{\partial D(a_1, a_2)}{\partial a_2} = 2 \sum_{k=1}^m [a_1 x_k^2 + \text{sen}(a_2 x_k) - y_k] [x_k \cos(a_2 x_k)] = 0$$

Ou de forma equivalente:

$$\begin{cases} f_1(a_1, a_2) = \sum_{k=1}^m [a_1 x_k^2 + \text{sen}(a_2 x_k) - y_k] [x_k^2] = 0 \\ f_2(a_1, a_2) = \sum_{k=1}^m [a_1 x_k^2 + \text{sen}(a_2 x_k) - y_k] [x_k \cos(a_2 x_k)] = 0 \end{cases}$$

Note que as funções $f_1(a_1, a_2)$ e $f_2(a_1, a_2)$ são compostas de somatórios aplicados sobre todos os m pontos, e a função representativa $g(x) = a_1 x^2 + \text{sen}(a_2 x)$ é **não linear em relação ao coeficiente a_2** . Nesses casos, conforme vimos no Capítulo 4, temos que tentar obter os valores de a_1 e a_2 pelo método de **Newton**, a partir de uma solução inicial $(a_1^{(0)}, a_2^{(0)})$.

A seguir, vamos ver um exemplo de ajuste de um parâmetro $P(v, T)$, dependente de dois parâmetros independentes v e T . Mas note que o ajuste é aplicado apenas ao parâmetro P , portanto se trata de ajuste de uma única variável.

Exemplo 7.3: a equação de estado de **Redlich-Kwong** permite relacionar propriedades termodinâmicas de gases reais com P (pressão) como propriedade dependente de v (volume específico) e T (temperatura), conforme modelo:

$$P(v, T) = \frac{R * T}{(v - a_2)} - \frac{a_1}{\sqrt{T} * v * (v + a_2)}$$

Em que $R = 8.314 \text{ (J) / (mol * K)}$ é a constante universal dos gases, e os valores de a_1 e a_2 são parâmetros de cada gás que podem ser determinados a partir de valores de propriedades físicas e de P , v e T medidos experimentalmente.

Como toda medição experimental possui erros inerentes, então procedemos uma série de m medições com o intuito de compensar os erros de uma medida para outra. Nesse exemplo, efetuamos medições do parâmetro pressão P , que é função do volume específico v e da temperatura T . Descartamos medições mais afetadas por erros de medição e então serão consideradas $m = 21$ medições efetivas para a determinação dos parâmetros a_1 e a_2 , conforme segue (dados no formato Octave):

$m = 21\%$ Número de pontos experimentais

$v = [6.85 \ 7.95 \ 5.31 \ 5.24 \ 8.02 \ 8.83 \ 6.39 \ 6.72 \ 8.79 \ 8.17 \ 7.89 \ 6.86 \ 7.47 \ 8.91 \ 6.27 \ 5.58 \ 6.29 \ 8.50 \ 7.29 \ 8.39 \ 5.99]$; $\%(\text{m}^3/\text{mol})$

$T = [382. \ 320. \ 416. \ 365. \ 375. \ 431. \ 446. \ 346. \ 360. \ 461. \ 306. \ 425. \ 425. \ 415. \ 322. \ 329. \ 368. \ 355. \ 443. \ 301. \ 408.]$; $\%(\text{K})$

$P = [543. \ 382. \ 803. \ 715. \ 443. \ 457. \ 443. \ 457. \ 688. \ 503. \ 384. \ 535. \ 369. \ 602. \ 546. \ 436. \ 509. \ 596. \ 579. \ 393. \ 585. \ 339. \ 679.]$; $\%(\text{Pa})$

- Determine os parâmetros a_1 e a_2 pela minimização do desvio quadrático total;
- Calcule também o seu coeficiente de determinação R^2 ;
- Plote um gráfico com a superfície P em função de v e T .

Solução:

- A função desvio total quadrático de $P(v, T)$, conforme a eq. (2), será:

$$D(a_1, a_2) = \sum_{k=1}^m \left[\frac{R * T_k}{(v_k - a_2)} - \frac{a_1}{\sqrt{T_k} * v_k * (v_k + a_2)} - P_k \right]^2 \quad (8)$$

Para determinar os parâmetros a_1 e a_2 , vamos obter o ponto crítico de D em relação a a_1 e a_2 :

$$\frac{\partial D(a_1, a_2)}{\partial a_1} = 2 \sum_{k=1}^m \left[\left(\frac{R * T_k}{(v_k - a_2)} - \frac{a_1}{\sqrt{T_k} * v_k (v_k + a_2)} - P_k \right) * \left(-\frac{1}{\sqrt{T_k} * v_k (v_k + a_2)} \right) \right] = 0 \quad (9)$$

$$\frac{\partial D(a_1, a_2)}{\partial a_2} = 2 \sum_{k=1}^m \left(\frac{R * T_k}{(v_k - a_2)} - \frac{a_1}{\sqrt{T_k} * v_k (v_k + a_2)} - P_k \right) * \left(R * T_k \frac{(-1)(-1)}{(v_k - a_2)^2} - \frac{a_1}{\sqrt{T_k} * v_k (v_k + a_2)^2} \right) = 0 \quad (10)$$

Note que as eqs. (9) e (10) formam um sistema de equações não lineares, conforme segue:

$$\begin{cases} f_1(a_1, a_2) = \sum_{k=1}^m \left[\left(\frac{R * T_k}{(v_k - a_2)} - \frac{a_1}{\sqrt{T_k} * v_k (v_k + a_2)} - P_k \right) \left(-\frac{1}{\sqrt{T_k} * v_k (v_k + a_2)} \right) \right] = 0 \\ f_2(a_1, a_2) = \sum_{k=1}^m \left[\left(\frac{R * T_k}{(v_k - a_2)} - \frac{a_1}{\sqrt{T_k} * v_k (v_k + a_2)} - P_k \right) \left(R * T_k \frac{(-1)(-1)}{(v_k - a_2)^2} - \frac{a_1}{\sqrt{T_k} * v_k (v_k + a_2)^2} \right) \right] = 0 \end{cases} \quad (11)$$

Aplicando o método de Newton, abordado no Capítulo 4, com derivadas calculadas numericamente a partir da solução inicial $(a_1^0, a_2^0) = (1.0, 1.0)$ ⁹, resulta a solução:

$$a_1 = 367.0011181444968 \text{ e } a_2 = 1.00152293682138.$$

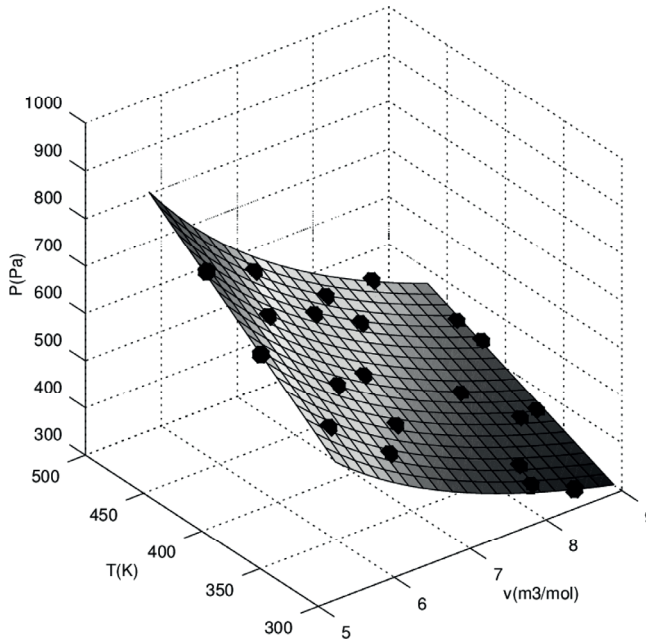
(obtidos em 5 iterações e critério relativo de parada $7.95e - 14$).



Em alguns sistemas não lineares é necessário efetuar várias tentativas de valores iniciais até se obter um solução. Além disso, esses sistemas podem ter mais de uma solução.

- b) O coeficiente de determinação obtido aplicando a eq. (7) é:
 $R^2 = 0.999975676672985$.
- c) Para traçar o gráfico da superfície P em função de v e T , usamos o algoritmo disponível no **Caderno de Algoritmos**, no arquivo **Cap7exem7.3.PVT.m**.

Gráfico 7.6 – Gráfico 3D de P em função de v e T do Exemplo 7.3



Fonte: Elaboração própria.

Se tivéssemos dois pontos com precisão adequada, (P_1, v_1, T_1) e (P_2, v_2, T_2) , poderíamos substituí-los diretamente na equação de estado de **Redlich-Kwong**, fazendo a função $P(v, T)$ passar sobre esses dois pontos, ou seja, com erros locais nulos: $P(v_k, T_k) - P_k = 0$. Dessa forma, geramos um sistema de duas equações não lineares, conforme segue:

$$\begin{cases} F_1(a_1, a_2) = \frac{R * T_1}{(v_1 - a_2)} - \frac{a_1}{\sqrt{T_1} * v_1 * (v_1 + a_2)} - P_1 = 0 \\ F_2(a_1, a_2) = \frac{R * T_2}{(v_2 - a_2)} - \frac{a_1}{\sqrt{T_2} * v_2 * (v_2 + a_2)} - P_2 = 0 \end{cases} \quad (12)$$

Assim, poderíamos determinar as duas incógnitas, a_1 e a_2 , por uma interpolação, também usando o método de Newton. Pode-se verificar também que o sistema dado pelas eqs. (12) tem solução de convergência mais estável e pode determinar valores a_1 e a_2 a partir de dois pontos escolhidos (P_1, v_1, T_1) e (P_2, v_2, T_2) da amostra, cuja solução pode servir de valor inicial $(a_1^{(0)}, a_2^{(0)})$ para as eqs. (11).

Nos casos de ajuste a funções com coeficientes não lineares, como no **Exemplo 7.3**, o R^2 não deve ser o único critério para qualificar o ajuste, uma vez que pode até gerar resultados aparentemente excelentes, mas não ser suficientemente próximo dos pontos amostrais.

A seguir, vamos mostrar algumas funções que permitem uma transformação paramétrica para a forma polinomial.

7.2.2 Ajuste por transformações de forma paramétrica em polinomial

Algumas funções não polinomiais permitem a sua transformação para uma forma polinomial através de artifícios algébricos. Para tal, podemos resolver o ajuste polinomial transformado, via eq. (6), e retornar à forma não polinomial de origem posteriormente, conforme alguns casos apresentados a seguir.

7.2.2.1 Ajuste a exponenciais: $y = a * b^x$

Para tirar a variável x do expoente, aplicamos propriedades dos logaritmos na exponencial:

$$\ln(y) = \ln(a) + x * \ln(b)$$

Fazendo $\ln(y) = z$; $\ln(a) = a_1$; $\ln(b) = a_2$, temos

$z = a_1 + a_2 * x$ (que é um polinômio de 1º grau).

Então efetuamos o ajuste linear ao conjunto de pontos (x_k, z_k) , em que $z_k = \ln(y_k)$, para determinar a_1 e a_2 . Obtidos os coeficientes a_1 e a_2 do polinômio, via eq. (6), recuperamos os valores originais de a e b através da definição de logaritmo pelas relações:

$$\ln(a) = a_1 \rightarrow a = e^{a_1}$$

$$\ln(b) = a_2 \rightarrow b = e^{a_2}$$

Exemplo 7.4: ajuste à base de dados

x_k	1	3	4	6
y_k	2.5	13	22	36

a uma curva exponencial $y = a * b^x$. Estime $f = (2.5)$, calcule o valor de x correspondente a $y = 50$, faça um gráfico da função ajustada e dos pontos tabelados.

Solução:

$$y = a * b^x \Rightarrow \ln(y) = \ln(a) + x * \ln(b)$$

$$P_1(x) = a_1 + a_2 * x$$

Com $\ln(y) = z$, $\ln(a) = a_1$, $\ln(b) = a_2$.

Montando um sistema para $(x_k, z_k = \ln(y_k))$, temos

x_k	1	3	4	5	6
y_k	2.5	8	13	22	40
$z_k = \ln(y_k)$	0.9162907	2.079442	2.564949	3.091042	3.688879

Pela eq. (6) temos

$$\begin{bmatrix} m & \sum x_k \\ \sum x_k & \sum x_k^2 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum z_k \\ \sum x_k z_k \end{bmatrix}$$

$$\begin{bmatrix} 5 & 19 \\ 19 & 87 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 12.34060353848778 \\ 55.00290177823501 \end{bmatrix} \Rightarrow \begin{cases} a_1 = 0.386180730567183 \\ a_2 = 0.547878941350098 \end{cases}$$

Retornando aos valores originais de a e b , temos

$$a = e^{0.386180730567183} = 1.47135056480854$$

$$b = e^{0.547878941350098} = 1.72958058266343$$

Logo,

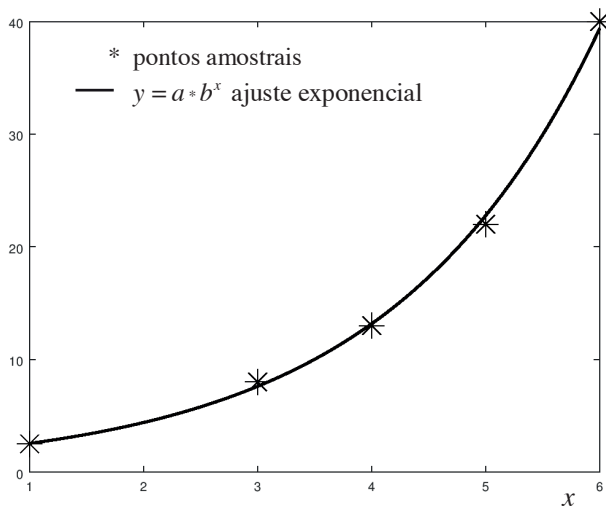
$y = 1.47135056480854 * (1.72958058266343)^x$ é a exponencial ajustadora.

Estimando:

$$f(2.5) = 1.47135056480854 * (1.72958058266343)^{2.5} = 5.78852832481353$$

$$50 = 1.47135056480854 * (1.72958058266343)^x \Rightarrow x = 6.43544040253215$$

Gráfico 7.7 – Ajuste do Exemplo 7.4



Fonte: Elaboração própria.

7.2.2.2 Ajuste a curvas geométricas: $y = a * x^b$, com $b \in \mathbb{R}$

Aplicamos as propriedades dos logaritmos

$$\ln(y) = \ln(a) + b * \ln(x)$$

Fazendo $\ln(y) = z$; $\ln(a) = a_1$; $b = a_2$ e $\ln(x) = t$, temos $z = a_1 + a_2 * t$, que é um polinômio de 1º grau em (t_k, z_k) . Do mesmo modo, efetuamos o ajuste linear ao conjunto de pontos (t_k, z_k) , em que $t_k = \ln(x_k)$ e $z_k = \ln(y_k)$, para determinar a_1 e a_2 . Obtidos os coeficientes a_1 e a_2 do polinômio transformado, recuperamos os valores originais de a e b por meio de suas relações:

$$a = e^{a_1}$$

$$b = a_2$$

7.2.2.3 Ajuste a curvas hiperbólicas: $y = \frac{1}{a_1 + a_2 * x}$

Transformamos pelo recíproco,

$$\frac{1}{y} = a_1 + a_2 * x$$

Fazendo $1/y = z$, temos $z = a_1 + a_2 * x$, que é um polinômio de grau $n = 1$, e efetuamos o ajuste linear ao conjunto de pontos (x_k, z_k) , em que $z_k = 1 / y_k$, para determinar diretamente a_1 e a_2 .

7.2.2.4 Ajuste a sigmóides: $y = (1 + e^{-(a_1 + a_2 * x)})^{-1}$

Transformamos por meio da composição do recíproco e da propriedade dos logaritmos:

$$-\ln\left(\frac{1}{y} - 1\right) = a_1 + a_2 * x$$

Fazendo $z = -\ln(1/y - 1)$, temos $z = a_1 + a_2 * x$, que é um polinômio de grau $n = 1$, e promovemos o ajuste linear ao conjunto de pontos (x_k, z_k) , em que $z_k = -\ln(1/y_k - 1)$, para determinar diretamente a_1 e a_2 .

O ajuste via transformações lineares pode não gerar o mesmo resultado que o ajuste direto da função aproximadora na sua forma original, como comprovaremos no **Exemplo 7.5**, mas a forma transformada para linear é mais rápida e sempre tem solução, enquanto a solução do sistema não linear, gerado no ajuste direto, pode ter convergência difícil e não garantida.

A seguir, vamos apresentar um exemplo de ajuste efetuado diretamente pela minimização do desvio quadrático total e compará-lo com o ajuste efetuado através de transformação paramétrica para a forma polinomial. Também vamos fazer uma introdução à análise dos resultados.

Exemplo 7.5: a tabela, a seguir, com $m = 5$ pontos, obtidos experimentalmente, relaciona o volume V adimensional de álcool gerado em um reator em função da sua temperatura adimensional T de reação:

T_k	0.2	0.4	0.6	0.8	1.0
V_k	0.04	0.14	0.26	0.39	0.50

Considerando teoricamente que a relação entre essas duas variáveis é modelada pela função não polinomial $V(T) = (a_1 + a_2 / T^2)^{-1}$:

- a) Determine os parâmetros a_1 e a_2 através de ajuste de curvas, por **transformação paramétrica** em polinomial, de modo a levar em conta todas as $m = 5$ medições experimentais.
- b) Determine os parâmetros a_1 e a_2 através da **minimização direta** do desvio quadrático total, de modo a considerar as $m = 5$ medições experimentais dadas.
- c) Calcule o coeficiente de determinação R^2 e a média dos resíduos (desvios locais) em cada caso.

- d) Plote os ajustes em cada caso.
 e) Plote os gráficos dos resíduos (desvios locais) em cada caso.
 f) Avalie os resultados.

Solução:

a) Transformando $V(T) = \frac{1}{a_1 + a_2(1/T^2)}$ na forma polinomial, resulta

$$\frac{1}{V(T)} = a_1 + a_2(1/T^2)$$

$$y = a_1 + a_2 * x$$

$$\text{Com } y_k = \frac{1}{V_k}, \quad x_k = \frac{1}{T_k^2}$$

Dados originais:

T_k	0.2	0.4	0.6	0.8	1.0
V_k	0.04	0.14	0.26	0.39	0.50

Novos dados com $x_k = \frac{1}{T_k^2}$ e $y_k = \frac{1}{V_k}$:

x_k	25.0	6.25	2.7778	1.5625	1.00
y_k	25.0	7.1429	3.8462	2.5641	2.00

Montando o sistema pela eq. (6) para (x_k, y_k) ,

$$\begin{bmatrix} m & \sum x_k \\ \sum x_k & \sum x_k^2 \end{bmatrix} * \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} \sum y_k \\ \sum x_k^1 y_k \end{bmatrix}$$

$$A = \begin{bmatrix} 5.0000 & 36.5903 & \vdots & 40.5531 \\ 36.5903 & 675.2200 & \vdots & 686.3330 \end{bmatrix}$$

Resolvemos preferencialmente por Cholesky:

$$a_1 = 1.11383182450055$$

$$a_2 = 0.956099722529505$$

b) Minimizando diretamente o desvio quadrático total via eq. (2), temos

$$D(a, b) = \sum_{k=1}^m d_k^2 = \sum_{k=1}^m \left[\frac{1}{a_1 + a_2 (1/T_k^2)} - V_k \right]^2$$

$$\frac{\partial D(a, b)}{\partial a_1} = 2 \sum_{k=1}^m \left[(a_1 + a_2 / T_k^2)^{-1} - V_k \right] \left[(-1) (a_1 + a_2 / T_k^2)^{-2} \right] = 0$$

$$\frac{\partial D(a, b)}{\partial a_2} = 2 \sum_{k=1}^m \left[(a_1 + a_2 / T_k^2)^{-1} - V_k \right] \left[(-1) (a_1 + a_2 / T_k^2)^{-2} \right] (1/T_k^2) = 0$$

Resultam estas duas equações não lineares:

$$\begin{cases} f_1(a, b) = -\sum_{k=1}^m \left[(a_1 + a_2 / T_k^2)^{-1} - V_k \right] (a_1 + a_2 / T_k^2)^{-2} = 0 \\ f_2(a, b) = -\sum_{k=1}^m \left[(a_1 + a_2 / T_k^2)^{-1} - V_k \right] (a_1 + a_2 / T_k^2)^{-2} (1/T_k^2) = 0 \end{cases}$$

Que resolvemos pelo método de Newton:

$$a_1 = 0.991635399512743$$

$$a_2 = 1.00947137630393$$

c) Calculado os respectivos coeficientes de determinação R^2 e as médias dos resíduos (desvios locais), temos:

i) por transformação paramétrica em polinomial:

$$R_1^2 = 0.997397323175315$$

$$\text{Média dos resíduos 1} = -34.2280214515672e-04$$

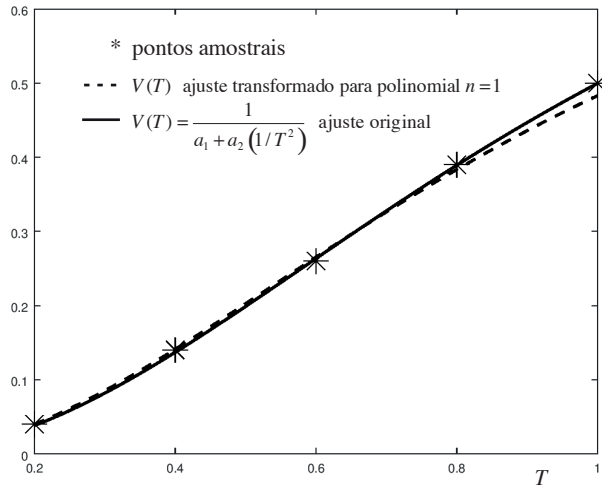
ii) por minimização direta:

$$R_2^2 = 0.999815970365156$$

$$\text{Média resíduos 2} = -4.91670466903520e-04$$

d)

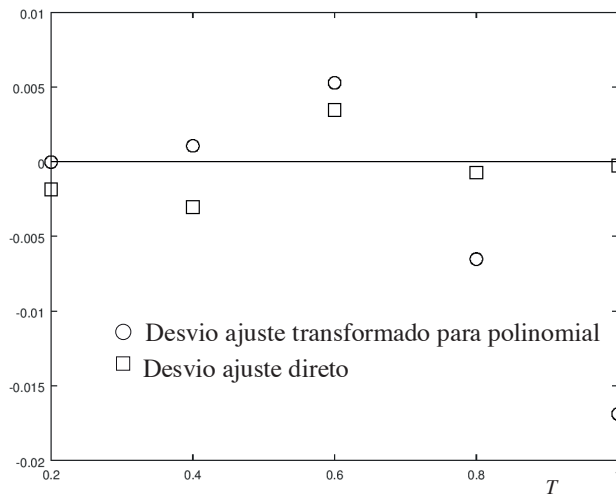
Gráfico 7.8 – Comparativo entre o ajuste transformado em polinomial e o direto do Exemplo 7.5



Fonte: Elaboração própria.

e)

Gráfico 7.9 – Representação dos resíduos (desvios locais) do Exemplo 7.5



Fonte: Elaboração própria.

- f) Mesmo com esse número reduzido de amostras, escolhido por motivos didáticos, o ajuste direto gerou um coeficiente de determinação R_2^2 mais próximo da unidade e um resíduo médio mais próximo de zero, em relação ao ajuste parametrizado. Por meio do Gráfico 7.8, também verificamos que o ajuste direto, mostrado em linha contínua, representa melhor a tendência dos $m = 5$ pontos experimentais, especialmente no final do intervalo, onde o ajuste parametrizado começa a perder aderência em relação aos pontos tabelados, fato esse comprovado no Gráfico 7.9, onde os resíduos do ajuste direto, representados por quadrados, são menores e mais próximos de uma distribuição aleatória em torno do zero, enquanto os resíduos do ajuste parametrizado, representados por círculos, são maiores e com tendência para valores negativos.

O algoritmo gerador dos resultados do **Exemplo 7.5** está disponível no **Caderno de Algoritmos em Cap7exemplo.7.5.m**.

7.3 CONCLUSÕES

Encerramos aqui a nossa abordagem sobre o ajuste de curvas salientando que os tópicos apresentados neste capítulo são apenas noções básicas sobre o assunto, sempre voltados para implementação em computador. O interessado no aprofundamento desse conteúdo para uso em experimentos reais, com a obtenção do respectivo modelo matemático confiável através do ajuste de curvas, deverá consultar textos sobre a técnica estatística da *análise de regressão*⁹.



As obras *Estatística para Cursos de Engenharia e Informática*, de Pedro Alberto Barbeta, Marcelo Menezes Reis e Antonio Cezar Bornia (2010); e *Handbook of Regression Analysis*, de Samprit Chatterjee e Jeffrey S. Simonoff (2012), são referências recomendadas sobre análise de regressão.

Sobre a qualidade do ajuste (*Goodness of fit*), recomendamos ainda buscar uma análise mais abrangente dos resultados, para além das medidas citadas neste livro, e usar também técnicas de otimização do ajuste que considerem, por exemplo, o número de parâmetros necessários em cada função ajustadora, como os *Crítérios de Informação de Akaike (AIC)*⁹.



Para iniciar sua pesquisa sobre AIC, consulte o endereço eletrônico: <<http://www.portalaction.com.br/analise-de-regressao/2715-aic-e-bic>>. Acesso em: 20 dez. 2016.


INTEGRAÇÃO NUMÉRICA

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- efetuar operações de integração numérica com os métodos newtonianos;
- efetuar operações de integração numérica com os métodos gaussianos;
- efetuar o controle dos erros envolvidos na aplicação computacional dos métodos de integração numérica abordados; e
- utilizar os algoritmos disponibilizados.

Neste capítulo, faremos uso da aproximação de funções para efetuar, via computador com processamento numérico, a *operação clássica do cálculo diferencial e integral*, que consiste em obter o valor de $I = \int_a^b f(x) dx$, em que $[a, b]$ é o **domínio de integração**, $f(x)$ a **função integranda** e a expressão I é denominada de **integral simples e definida**.

 Essa operação é emblemática para a atual era dos computadores, pois motivou a criação daquele considerado o primeiro computador eletrônico em grande escala, denominado de Eletronic Numerical Integrator and Computer (ENIAC I). Disponível em: <<https://pt.wikipedia.org/wiki/ENIAC>>. Acesso em: 12 jul. 2017.

Conceitualmente, I possui duas definições clássicas:

Definição 1: obtemos a **integral indefinida** $\tilde{I} = \int f(x) dx$ determinando outra função $F(x)$, tal que $F'(x) = f(x)$, então $\tilde{I} = F(x) + C$, em que C é uma constante e $F(x)$ é denominada de **primitiva** ou **antiderivada**.

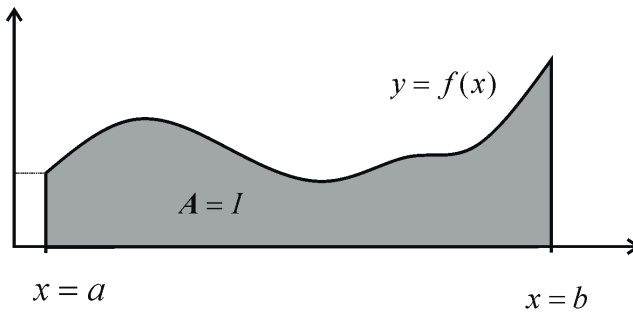
Dessa definição, notamos que a integração é a operação inversa da derivação, sendo sua operacionalização tipicamente algébrica e abstrativa.

Definição 2: obtemos **integral definida** $I = \int_a^b f(x) dx$ por meio dos seguintes passos:

- a) dividimos o intervalo $[a, b]$ em n partes de comprimento $h_i = x_{i+1} - x_i$;
- b) obtemos a soma $I_n = \sum_{i=1}^n f(x_i)h_i$; e
- c) no limite de n , temos $I = \lim_{n \rightarrow \infty} I_n$.

Essa segunda definição é denominada na literatura de **integral de Riemann**, sendo tipicamente numérica e construtiva. Se $y = f(x)$ for contínua em $[a, b]$, e considerando cada h_i como a base de um retângulo infinitesimal e cada $f(x_i)$ a respectiva altura, no limite, o valor de I será a **área** da região A subentendida pelo segmento do gráfico da $y = f(x)$ situado entre as retas $x = a$ e $x = b$, e o eixo das abscissas x , conforme o Gráfico 8.1.

Gráfico 8.1 – Área exata A limitada pela função $y = f(x)$, $x = a$, $x = b$, e $y = 0$



Fonte: Elaboração própria.

A conexão entre as duas definições de integral é o **Teorema Fundamental do Cálculo (TFC)**, por meio do qual provamos simplesmente que, se a $f(x)$ for contínua em $[a, b]$ e $F(x)$, a sua primitiva, então:

$$I = \int_a^b f(x) dx = F(b) - F(a) = A$$

Dessa forma, o TFC nos liberta de efetuar quantidades de adições que tendem ao infinito em troca da determinação da primitiva $F(x)$. Contudo, como é do conhecimento de todo estudante que cursou o primeiro Cálculo, a aplicação do TFC em uma $I = \int_a^b f(x) dx$ qualquer nem sempre é fácil, podendo ser muito difícil ou até impossível, uma vez que:

a) A integranda $y = f(x)$ pode ser apenas uma tabela do tipo:

x	x_1	x_2	...	x_{n+1}
$f(x)$	y_1	y_2	...	y_{n+1}

- b) A integranda $y = f(x)$ pode ter primitiva conhecida, porém seu uso ser completamente ineficiente, como na integral recursiva:

$$I = \int_a^b \operatorname{sen}^m(x) \cos^n(x) dx = \int_a^b \frac{\operatorname{sen}^{m-1}(x) \cos^{n+1}(x)}{n+m} dx + \frac{m-1}{n+m} \int_a^b \operatorname{sen}^{m-2}(x) \cos^n(x) dx$$

- c) Existem integrandas cujas primitivas são impossíveis de serem expressas em termos de funções elementares, por exemplo, as primitivas de:

$$I = \int_0^1 e^{-x^2} dx, \quad I = \int_2^5 \frac{dx}{\ln(x)}, \quad I = \int_0^1 \operatorname{sen}\left(\frac{\pi x^2}{2}\right) dx$$

Definição 3 – a **integração numérica de** $I = \int_a^b f(x) dx$ consiste em aplicar o TFC não diretamente na $y = f(x)$ original, mas em aproximadoras da $y = f(x)$.

Os métodos de integração numérica podem ser agrupados em duas famílias, conforme podemos ver na literatura, de acordo com a forma de aproximação da função integranda $y = f(x)$:

- a) por interpolação polinomial simples sobre pontos uniformemente distribuídos em $[a, b]$ – **métodos de Newton-Cotes**; e
- b) por interpolação polinomial sobre pontos predefinidos em $[a, b]$ – **métodos gaussianos**.

Assim, considerando a interpretação geométrica de uma integral definida, vamos desenvolver métodos e implementar algoritmos para calculá-la numericamente, com erro máximo controlado, por meio de aproximações do cálculo da área delimitada por $y = f(x)$, $x = a$, $x = b$ e $y = 0$.

8.1 INTEGRAÇÃO NUMÉRICA POR MÉTODOS DE NEWTON-COTES

Nas seções a seguir, vamos detalhar dois representantes clássicos da família de métodos de integração numérica de Newton-Cotes.

8.1.1 Método dos trapézios

Para estimar o valor de uma $I = \int_a^b f(x) dx$, procedemos como segue:

a) Dividimos $[a, b]$ em n subintervalos de comprimento uniforme

$$h = \frac{(b-a)}{n}.$$

b) Obtemos os $n + 1$ valores funcionais (x_i, y_i) , em que

$$\begin{aligned} x_1 = a, \quad x_{i+1} = x_i + h \quad (\forall i = 1, 2, \dots, n); \quad \mathbf{e} \\ y_i = f(x_i) \quad (\forall i = 1, 2, \dots, n+1). \end{aligned}$$

c) Para cada 2 pontos sucessivos (x_i, y_i) e (x_{i+1}, y_{i+1}) , determinamos o seu interpolador polinomial, por exemplo, na forma de Gregory-Newton com diferenças como:

$$P_i(x) = y_i + \frac{y_{i+1} - y_i}{h} (x - x_i)$$

Então, pelo TFC, obtemos cada área aproximada A_i no subintervalo $[x_i, x_{i+1}]$ que é dada por:

$$A_i = \int_{x_i}^{x_{i+1}} P_i(x) dx = \frac{h}{2} [y_i + y_{i+1}]$$

Adicionando as áreas correspondentes aos n subintervalos, teremos a área aproximada total denotada por T_n ,

$$T_n = \sum_{i=1}^n A_i$$

$$T_n = \frac{h}{2}[y_1 + y_2] + \frac{h}{2}[y_2 + y_3] + \dots + \frac{h}{2}[y_n + y_{n+1}]$$

$$T_n = \frac{h}{2}[y_1 + 2y_2 + 2y_3 + \dots + 2y_n + y_{n+1}]$$

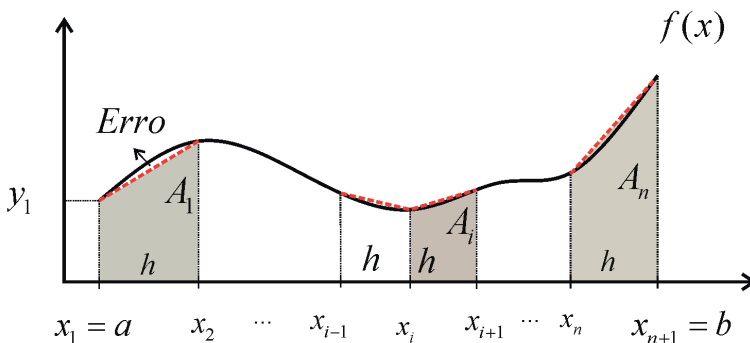
$$T_n = \frac{h}{2} \left[y_1 + 2 \sum_{i=2}^n y_i + y_{n+1} \right]$$

(1)

Finalmente, $I = \int_a^b f(x) dx \cong \sum_{i=1}^n A_i = T_n$

Observe que cada A_i é a área de um trapézio de altura h e bases y_i e y_{i+1} , conforme o Gráfico 8.2.

Gráfico 8.2 - Integral de $f(x)$ em $[a, b]$ aproximada por n trapézios



Fonte: Elaboração própria.

Da eq. (1), podemos demonstrar o:

Teorema 1: para uma $y = f(x)$ continuamente diferenciável,

$$I = \int_a^b f(x) dx = \lim_{n \rightarrow \infty} T_n.$$

Exemplo 8.1: calcule por trapézios $I = \int_a^b f(x) dx$, em que a $f(x)$ é a função discreta a seguir:

x_i	0.00	0.25	0.50	0.75	1.00
$f(x_i)$	3	5	8	4	2

Solução:

Temos $n + 1 = 5$ pontos $\Rightarrow n = 4$ intervalos e $x_{i+1} - x_i = 0.25 \Rightarrow h = 0.25$

Aplicando a eq. (1), resulta \Rightarrow

$$T_n = (0.25/2)[3 + 2(5+8+4) + 2] = 4.875 \text{ e}$$

$$I \cong T_n = 4.875$$

Podemos observar, no primeiro trapézio do Gráfico 8.2, que ocorre uma diferença, ou erro, entre a área desse trapézio e a área exata no domínio $[x_1, x_2]$. Esse erro, que é de truncamento, ocorre em cada subintervalo, sempre que a integranda não seja uma reta.

Na sequência, vamos abordar como estimamos o erro total gerado pelos n trapézios na aproximação de uma integração I pela eq. (1).

Teorema do erro de truncamento de T_n

Em uma $I = \int_a^b f(x) dx$, se $y = f(x)$ for contínua e duplamente diferenciável em $[a, b]$ e aplicarmos trapézios com intervalo $[a, b]$ subdividido em n partes iguais, então o **erro de truncamento** de T_n será:

$$ET_n = \frac{-h^2(b-a)f''(\xi)}{12}, \text{ em que } \xi \in [a, b].$$

Demonstração:

Pela interpolação polinomial geral de grau n , conforme vimos no Capítulo 5, o erro do aproximador $P_n(x)$ em relação à $f(x)$ é dado por:

$$R_n(x) = |f(x) - P_n(x)| = \left| \frac{f^{(n+1)}(\xi) \prod_{i=1}^{n+1} (x - x_i)}{(n+1)!} \right|$$

No caso da aproximação por interpolação polinomial com $n = 1$, usada no método dos trapézios, temos que $f(x) = P_1(x) + R_1(x)$, para $x \in [x_i, x_{i+1}]$:

$$f(x) = P_1(x) + \frac{f''(\xi)(x - x_i)(x - x_{i+1})}{(1+1)!}$$

$$\text{Então, } \int_{x_i}^{x_{i+1}} f(x) dx = \int_{x_i}^{x_{i+1}} \left(P_1(x) + \frac{f''(\xi)(x - x_i)(x - x_{i+1})}{(1+1)!} \right) dx$$

$$\int_{x_i}^{x_{i+1}} f(x) dx = A_i + ET_i(\xi), \text{ em que:}$$

$$A_i = \int_{x_i}^{x_{i+1}} P_i(x) dx = \frac{h}{2} [y_i + y_{i+1}] \text{ e } ET_i(\xi) = \frac{f''(\xi)}{(2)!} \int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1}) dx.$$

Desenvolvendo e integrando a parcela do erro $ET_i(\xi)$, temos

$$ET_i(\xi) = \frac{f''(\xi)}{(2)!} \int_{x_i}^{x_{i+1}} (x^2 - (x_i + x_{i+1})x + x_i x_{i+1}) dx =$$

$$\frac{f''(\xi)}{(2)!} \left(\frac{x^3}{3} - (x_i + x_{i+1}) \frac{x^2}{2} + x_i x_{i+1} x \right) \Bigg|_{x_i}^{x_{i+1}}$$

Logo, o erro em cada subintervalo é dado por:

$$ET_i(\xi) = \frac{f''(\xi)}{(2)!} \left(\frac{x_{i+1}^3 - x_i^3}{3} - (x_i + x_{i+1}) \frac{x_{i+1}^2 - x_i^2}{2} + x_i x_{i+1} (x_{i+1} - x_i) \right)$$

Substituindo $x_{i+1} = x_i + h$

$$ET_i(\xi) = \frac{f''(\xi)}{(2)!} \left(\frac{(x_i + h)^3 - x_i^3}{3} - (x_i + x_i + h) \frac{(x_i + h)^2 - x_i^2}{2} + x_i(x_i + h)(x_i + h - x_i) \right)$$

Por meio de algumas simplificações algébricas, temos que:

$$ET_i(\xi) = \frac{f''(\xi)}{(2)!} \left(\frac{h^3}{3} - \frac{h^3}{2} \right) = -\frac{f''(\xi)}{(2)!} \frac{h^3}{6} = -\frac{f''(\xi)h^3}{12}$$

Então, essa expressão do erro $ET_i(\xi)$ fica independente da posição inicial x_i de cada intervalo e dependente apenas do h :

$$ET_i(\xi) = -\frac{f''(\xi)h^3}{12}$$

Tomando o erro de truncamento em todo o intervalo $[a, b]$ como a soma dos erros de cada trapézio, temos:

$$ET_n = ET_1 + ET_2 + ET_3 + \dots + ET_n$$

$$ET_n = -\frac{f''(\xi)h^3}{12} - \frac{f''(\xi)h^3}{12} - \frac{f''(\xi)h^3}{12} + \dots - \frac{f''(\xi)h^3}{12}$$

$$ET_n = -\frac{f''(\xi)h^3 n}{12} = -\frac{f''(\xi)h^2(b-a)}{12} \text{ com } h \cdot n = (b-a)$$

Tomando o majorante de $f''(x)$, em valor absoluto, temos

$$ET_n \text{Max} = \frac{\text{Max} |f''(x)|}{12} h^2 (b-a), \quad x \in [a, b] \quad (2)$$

Exemplo 8.2: determine o n mínimo para que o erro de truncamento máximo seja menor do que $\varepsilon = 10^{-6}$ ao efetuar $I = \int_1^6 \frac{1}{1+x} dx$ pelo método dos trapézios.

Solução:

Temos $f(x) = (1+x)^{-1} \Rightarrow f'(x) = -(1+x)^{-2} \Rightarrow f''(x) = 2(1+x)^{-3}$

$$M = \max_{x \in [1,6]} |f''(x)| = f''(1) = \frac{2}{8} = 0.25$$

Aplicando a eq. (2), temos

$$10^{-6} \cong \frac{h^2 * 5 * 0.25}{12} \Rightarrow h \cong 0.0030984$$

$$n = \frac{(6-1)}{h} = 1613.7, \text{ tomando o próximo inteiro temos } n = 1614.$$

Recomendamos que você utilize n como potência de 2, para obter valores de h exatos, na base decimal e binária, portanto valores de x_i com mais precisão.

A aproximação T_n da integral I do **Exemplo 8.2**, obtida pelo método dos trapézios com $n = 1614$ subdivisões do intervalo $[1, 6]$, em precisão simples de 8 dígitos significativos, é $T_n = 1.25276315$. Como a integral exata é $I_e = 1.252762968495368$ (com 16 dígitos significativos), temos o Erro Exato = $|T_n - I_e| = 1.8 * 10^{-7}$, ou seja, um **Erro de truncamento**⁹ inferior a $\varepsilon = 10^{-6}$.



Para estimar isoladamente o **Erro de Arredondamento** de T_n , podemos calcular um T_n^* com precisão dupla (16 dígitos significativos) e compará-lo com o T_n .

Podemos calcular também o **Erro de Truncamento Estimado** = $|T_n - T_{2n}|$, usando precisão de 16 dígitos significativos, para isolar os efeitos dos arredondamentos.

Note que o Teorema 1, quando utilizado com $n \rightarrow \infty$, pode ser inválido se aplicado com precisão finita, pois os arredondamentos podem se acumular e desestabilizar os resultados. Nesses casos, existe um n limite ótimo em que conseguimos o resultado com menor erro total, mas que não é conhecido previamente. Para comprovar essa situação, vamos efetuar por trapézios a integral do **Exemplo 8.2**, aumentando sucessivamente o valor do n e

avaliando o comportamento dos resultados obtidos nos testes, que foram feitos em linguagem C, com variáveis de 32 *bits* (8 dígitos significativos), conforme a Tabela 8.1.

Tabela 8.1 – Comparativo de erros de truncamento exatos de T_n , obtido com 32 *bits*

n	Erro de truncamento exato = $ T_n - I_\epsilon $
1614	$1.8 * 10^{-7}$
2048	$2.8 * 10^{-8}$
4096	$3.0 * 10^{-7}$
32768	$2.9 * 10^{-7}$
65536	$8.5 * 10^{-6}$
131072	$3.3 * 10^{-5}$
262144	$5.6 * 10^{-5}$
4194304	$2.1 * 10^{-2}$

Fonte: Elaboração própria.

Observe que os erros de arredondamentos começaram a influenciar o T_n já a partir de 4.096 subdivisões e deturparam completamente o resultado nas últimas tentativas.

No caso de cálculos com precisão de 8 dígitos, perceba que existe um n limite ótimo em torno de $n = 2048$, ocorrendo o menor erro total. Quando aumentamos o valor de n , além de 1614, reduzimos sempre os erros de truncamento para valores menores do que 10^{-6} , mas aumentamos o erro de arredondamento acumulado, que ocorre em torno do 7º dígito significativo, nesse exemplo.

Entretanto, para T_n^* com variáveis de 64 *bits* (precisão de 16 dígitos), os erros de arredondamento influenciam muito menos, como podemos ver na Tabela 8.2.

Tabela 8.2 – Comparativo de erros de truncamento exatos de T_n^* , obtido com 64 *bits*

n	Erro de truncamento exato = $ T_n^* - I_\epsilon $
1614	$1.7 * 10^{-7}$
4194304	$1.1 * 10^{-13}$

Fonte: Elaboração própria.

No caso de cálculos com precisão de 16 dígitos, o n limite ótimo é muito maior, pois os arredondamentos acumulam erros em torno do 15º dígito significativo, nesse exemplo.

Para contornar esse problema de acúmulo de arredondamentos, podemos efetuar algumas aproximações com diferentes valores de n e analisar a **tendência de evolução** dos resultados do método dos trapézios via extrapolação dos T_n obtidos para o **limite de Romberg**, que consiste em:

- a) efetuar k aproximações $T_{(i,1)}$ por trapézios (sem extrapolação, $j=1$), iniciando com espaçamento h , $i = 1$, e reduzindo obrigatoriamente o h pela metade em cada nova aproximação, $i = 2, \dots, k$; e
- b) gerar a matriz de aproximações por extrapolação, via:

$$T_{(i,j)} = \frac{4^{j-1}T_{(i,j-1)} - T_{(i-1,j-1)}}{4^{j-1} - 1} \quad (3)$$

para cada coluna $j = 2, 3, \dots, k$, efetuar extrapolação para $i = j, \dots, k$.

Desses valores extrapolados, resulta o:

Teorema de Romberg: para uma função $y = f(x)$ continuamente diferenciável, $\lim_{k \rightarrow \infty} T_{(i,k)} = \int_a^b f(x) dx$.

Exemplo 8.3: efetue $\int_0^1 e^x dx$ por trapézios e Romberg, com $k = 5$ aproximações, iniciando com $h = 0.25$ ($n = 4$ subintervalos) e precisão de 16 dígitos.

Solução:

Aplicando a eq. (1), inicialmente com $n = 4$ ($h = 0.25$), para gerar a primeira coluna de resultados $T_{(i,1)}$; e a eq. (3), para gerar os elementos das demais colunas, temos:

$h_i \backslash j$	$i \backslash j$	1	2	3	4	5
$h/1$	1	1.72722190455717				
$h/2$	2	1.720518592164302	1.718284154699897			
$h/4$	3	1.718841128579994	1.718281974051892	1.718281828675358		
$h/8$	4	1.718421660316327	1.718281837561771	1.718281828462430	1.718281828459050	
$h/16$	5	1.718316786850094	1.718281829028016	1.718281828459099	1.718281828459046	1.718281828459046

Como os últimos resultados extrapolados são iguais, então:

$T(5, 5) = 1.718281828459046$ é o valor da integral com precisão de 16 dígitos, que é o próprio valor exato $I_e = e^1 - 1$.

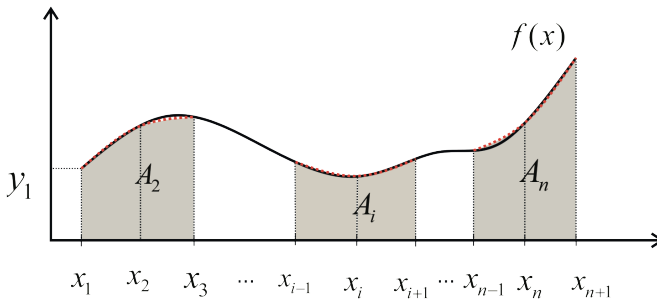
No **Caderno de Algoritmos**, você encontra o arquivo **Cap8exem8.3TnRomberg.m** com o algoritmo de Romberg.

A seguir, apresentaremos um método que, em geral, fornece resultados mais exatos do que o método dos trapézios para uma mesma quantidade de avaliações da integranda.

8.1.2 Método de Simpson

No método dos trapézios, usamos dois pontos sucessivos para interpolar a integranda com uma reta. No método de Simpson, vamos usar três pontos sucessivos para interpolar $y = f(x)$ por uma parábola $P_2(x)$ pontilhada, conforme o Gráfico 8.3.

Gráfico 8.3 – Integral de $f(x)$ aproximada em $[a, b]$ por Simpson, com áreas A_i , $i = 2, 4, \dots, n$ obtidas a cada dois subintervalos $[x_{i-1}, x_i]$ e $[x_i, x_{i+1}]$



Fonte: Elaboração própria.

Para efetuar $I = \int_a^b f(x) dx$ por Simpson, procedemos da seguinte maneira:

- a) Dividimos $[a, b]$ em n (**inteiro par**) partes iguais de comprimento $h = \frac{b-a}{n}$ e geramos os $n + 1$ pontos sucessivos (x_p, y_p) , em que $y_i = f(x_i)$, $i = 1, 2, 3, \dots, n+1$.
- b) Para cada três pontos sucessivos, (x_{i-1}, y_{i-1}) , (x_i, y_i) e (x_{i+1}, y_{i+1}) , igualmente espaçados com intervalo $h = (x_i - x_{i-1}) = (x_{i+1} - x_i)$, obtemos o seu único polinômio interpolador, por exemplo, o de Newton, com diferenças finitas ascendentes, conforme segue:

$$P_2(x) = y_{i-1} + \frac{\bar{\Delta}y_{i-1}(x-x_{i-1})}{1!h} + \frac{\bar{\Delta}^2y_{i-1}(x-x_{i-1})(x-x_i)}{2!h * h}$$

$$\text{em que } \bar{\Delta}y_{i-1} = (y_i - y_{i-1})$$

$$\bar{\Delta}^2y_{i-1} = \bar{\Delta}^1y_i - \bar{\Delta}^1y_{i-1} = (y_{i+1} - y_i) - (y_i - y_{i-1}) = (y_{i+1} - 2y_i + y_{i-1})$$

- c) Para efetuar a integral $A_i = \int_{x_{i-1}}^{x_{i+1}} P_2(x) dx$, aplicamos o TFC e integramos o interpolador polinomial usando a técnica da mudança de variáveis via $x = t * h + x_{i-1}$.

$$\text{Então, } dx = h * dt, \quad \frac{(x-x_{i-1})}{h} = t \text{ e como}$$

$$\frac{(x-x_i)}{h} = \frac{(x-(x_{i-1}+h))}{h} = \frac{(x-x_{i-1})}{h} - 1 = (t-1)$$

Logo,

$$P_2(t) = y_{i-1} + \frac{\bar{\Delta}y_{i-1}(t)}{1!} + \frac{\bar{\Delta}^2y_{i-1}(t)(t-1)}{2!}$$

$$A_i = \int_{x_{i-1}}^{x_{i+1}} P_2(x) dx = \int_0^2 P_2(t) h dt$$

Pois, se $x = x_{i-1} \rightarrow t = 0$, $x = x_{i+1} \rightarrow t = 2$ e $dx = h * dt$.

Substituindo $P_2(t)$ em A_p , temos,

$$A_i = \int_0^2 P_2(t) h dt = \int_0^2 \left(y_{i-1} + \frac{\bar{\Delta}y_{i-1}(t)}{1!} + \frac{\bar{\Delta}^2 y_{i-1}(t)(t-1)}{2!} \right) h dt$$

Assim,

$$A_i = h \left(y_{i-1} \int_0^2 dt + \frac{\bar{\Delta}y_{i-1}}{1!} \int_0^2 t dt + \left(\frac{\bar{\Delta}^2 y_{i-1}}{2!} \right) \int_0^2 (t)(t-1) dt \right)$$

$$A_i = h \left(y_{i-1} t \Big|_0^2 + \frac{\bar{\Delta}y_{i-1}}{1!} \frac{t^2}{2} \Big|_0^2 + \left(\frac{\bar{\Delta}^2 y_{i-1}}{2!} \right) \left(\frac{t^3}{3} - \frac{t^2}{2} \right) \Big|_0^2 \right)$$

E substituindo os valores das diferenças finitas ascendentes, temos:

$$A_i = h \left((y_{i-1})2 + \left(\frac{y_i - y_{i-1}}{1!} \right) 2 + \left(\frac{y_{i+1} - 2y_i + y_{i-1}}{2!} \right) \left(\frac{2^3}{3} - \frac{2^2}{2} \right) \right)$$

$$A_i = \frac{h}{3} (y_{i+1} + 4y_i + y_{i-1})$$

d) Então, a área total S_n é dada pela soma de todas as áreas com A_p , $i = 2, 4, \dots, n$:

$$S_n = \frac{h}{3} (y_1 + 4y_2 + y_3) + \frac{h}{3} (y_3 + 4y_4 + y_5) + \dots + \frac{h}{3} (y_{n-1} + 4y_n + y_{n+1})$$

$$S_n = \frac{h}{3} [(y_1 + y_{n+1}) + 4(y_2 + y_4 + \dots + y_n) + 2(y_3 + y_5 + \dots + y_{n-1})]$$

$$S_n = \frac{h}{3} \left(y_1 + 4 \underbrace{\sum_{i=2}^{n(\text{passo } 2)} y_i}_{\text{índices pares}} + 2 \underbrace{\sum_{i=3}^{n-1(\text{passo } 2)} y_i}_{\text{índices ímpares}} + y_{n+1} \right) \quad (4)$$

Essa é a fórmula de Simpson para cálculo aproximado da integral

definida, $I = \int_a^b f(x) dx \cong S_n$.

Exemplo 8.4: efetue por Simpson $I = \int_0^3 f(x) dx$, em que $f(x)$ é uma função discreta, como segue:

i	1	2	3	4	5	6	7
x_i	0.0	0.5	1.0	1.5	2.0	2.5	3.0
$y_i = f(x_i)$	4	6	7.5	8	5	3	2

Solução:

Temos: $h = x_{i+1} - x_i = 0.5$; $n = 6$ (par).

Aplicando a eq. (4), temos:

$$S_n = \frac{0.5}{3} [(4+2) + 4(6+8+3) + 2(7.5+5)] = 16.5$$

$$\text{Logo, } I = \int_0^3 f(x) dx \cong 16.5$$

Algumas considerações sobre o método de Simpson:

- Simpson, via de regra, fornece resultados mais precisos do que trapézios para um mesmo n . Também exige uma estimativa de um n ótimo, pelo mesmo motivo dos trapézios, ou seja, para um n elevado, o acúmulo dos erros de arredondamento pode ser maior do que o ganho obtido com a diminuição dos erros de truncamento, quando os resultados são obtidos em precisão baixa.
- Para determinar um n ótimo, também podemos tomar a expressão do erro de truncamento total do Simpson, que é dada por:

$$ES_n = \frac{-h^4(b-a)f^{(4)}(\xi)}{180} \quad \xi \in (a, b)$$

e utilizar o majorante da $f^{(4)}(x)$ em $[a, b]$, resultando em:

$$ES_n \text{Max} = \frac{h^4(b-a)M}{180}, \text{ em que } M = \max_{x \in [a,b]} |f^{(4)}(x)| \quad (5)$$

- c) Podemos melhorar os resultados por meio de tentativas sucessivas (com $h_{\text{novo}} = h_{\text{velho}} / 2$) e analisar o comportamento dos resultados.
- d) Não existe uma extrapolação específica para o método de Simpson, diferentemente do método dos trapézios que possui a extrapolação de Romberg.

Exemplo 8.5: determine o n mínimo para não ocorrer erro superior à

$$\varepsilon = 10^{-6} \text{ ao efetuar por Simpson a } I = \int_1^6 (1+x)^{-1} dx.$$

Solução:

Como

$$f(x) = (1+x)^{-1} \Rightarrow f'(x) = (-1)(1+x)^{-2} \Rightarrow$$

$$f''(x) = (-1)(-2)(1+x)^{-3} \Rightarrow f'''(x) = (-1)(-2)(-3)(1+x)^{-4} \Rightarrow$$

$$f^{(4)}(x) = (-1)(-2)(-3)(-4)(1+x)^{-5}$$

$$f^{(4)}(x) = 4!(1+x)^{-5} \Rightarrow \text{Max} \left| f^{(4)}(x) \right|_{x=1} = \frac{24}{(1+1)^5} = 0.75$$

Aplicando na eq. (5) \Rightarrow

$$10^{-6} = \frac{h^4(5)0.75}{180} \Rightarrow h = 0.0832358$$

$$n = \frac{5}{h} = 60.07028... \Rightarrow n = 60$$

Devido ao erro de arredondamento do passo h , para $n = 60$, recomendamos usar $n = 64 = 2^6$.

Conforme resultado do **Exemplo 8.2**, para essa mesma integral, o método dos trapézios demanda um $n = 1614$. Para o método de Simpson temos uma redução de 96% no número de cálculos de valores da função integranda (chamadas).

Os métodos de Newton-Cotes, apesar de serem conceitualmente simples, podem exigir alto volume de operações aritméticas em determinadas integrais, mesmo se utilizarmos aceleradores como Romberg. Além disso, por necessitarem dos valores de $f(a)$ e $f(b)$, não são aplicáveis em integrais impróprias com descontinuidades nos extremos do intervalo de integração, ou com extremos infinitos do tipo:

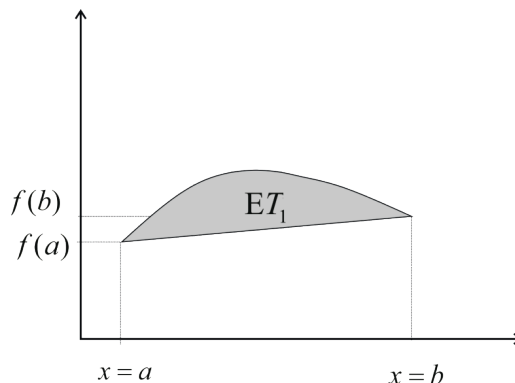
$$\int_a^{\infty} \quad \int_{-\infty}^b \quad \int_{-\infty}^{\infty}$$

Na sequência, vamos apresentar uma metodologia de integração numérica que supre essas deficiências dos métodos newtonianos.

8.2 INTEGRAÇÃO NUMÉRICA GAUSSIANA OU QUADRATURA GAUSSIANA

Considere que desejamos obter numericamente uma $I = \int_a^b f(x) dx$, hipoteticamente efetuando apenas dois cálculos de valores da integranda $y = f(x)$. Por consequência, só nos resta aplicar um único trapézio, T_1 , passando uma única reta pelos extremos $[a, b]$ do intervalo, e admitir um erro de truncamento muito elevado, como o erro destacado em cinza no exemplo do Gráfico 8.4.

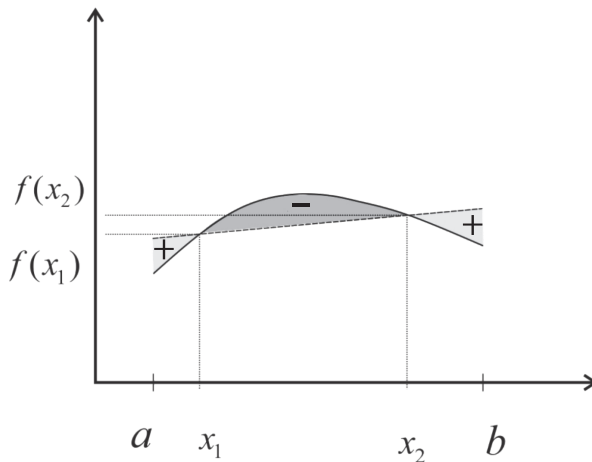
Gráfico 8.4 – Erro ET_1 da aproximação por trapézios com $m = 2$ pontos



Fonte: Elaboração própria.

O desafio fundamental é: se mantidos apenas os cálculos de valores da $y = f(x)$ em $m = 2$ pontos, seria possível diminuir o erro de truncamento gerado pelo método dos trapézios do Gráfico 8.4? A resposta de Gauss para essa questão é positiva, desde que a determinação dos dois valores funcionais que definirão a reta aproximadora sejam fixados não nos extremos do intervalo $[a, b]$, mas em pontos internos adequadamente escolhidos (x_1 e x_2), como os do Gráfico 8.5.

Gráfico 8.5 – Representação do Erro EG_2 da aproximação por Gauss-Legendre G_2 com $m = 2$ pontos



Fonte: Elaboração própria.

Observe que a reta aproximadora tracejada, proposta por Gauss-Legendre no Gráfico 8.5, compensa os erros, pois em parte a sua integração G_2 gera valores a mais (destacados com +) e em parte gera valores a menos (destacados com -), de modo que a soma desses erros EG_2 será bem menor do que o erro gerado no método dos trapézios ET_1 , ambos obtidos com $m = 2$ pontos.

Na quadratura gaussiana, os métodos de integração de $I = \int_b^a f(x) dx$ são desenvolvidos de modo que os valores da integranda $y = f(x)$ são obtidos não em quaisquer pontos desejados do domínio $[a, b]$, como pode ser feito nos métodos newtonianos, mas em pontos previamente estabelecidos. Esses pontos independem da integranda, porém dependem da quantidade de chamadas na integranda e da otimização do erro de truncamento para essa quantidade de chamadas.

Como os pontos de cálculo da integranda são prefixados, temos de padronizar o domínio $[a, b]$ em um intervalo fixo e conhecido, então vamos padronizar o domínio de integração $[a, b]$ fixando-o em $[-1, 1]$ por meio da conhecida mudança de variáveis:

$$x(t) = ((b - a) / 2) t + ((b + a) / 2) \Rightarrow dx = ((b - a) / 2) dt$$

Logo,

$$I = \int_a^b f(x) dx = ((b - a) / 2) \int_{-1}^1 \underbrace{f(((b - a) / 2)t + ((b + a) / 2))}_{g(t)} dt \quad (5)$$

Por consequência, os métodos gaussianos são aplicados em integrais normalizadas $\int_{-1}^1 g(t) dt$, e o resultado final multiplicado pelo fator $(b - a) / 2$ para obtermos $I = \int_a^b f(x) dx$.

A seguir, vamos desenvolver dois métodos típicos da quadratura gaussiana.

8.2.1 Método de Gauss-Legendre

Analogamente à soma de Riemann, aplicada no intervalo $[-1, 1]$, Gauss propõe:

$$I = \int_{-1}^1 g(t) dt \cong \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) = G_m \quad (6)$$

em que os $2m$ parâmetros indeterminados $C_{(m,k)}$ e $t_{(m,k)}$ são denominados respectivamente de “pesos” e “nós”. No método de Gauss-Legendre, esses parâmetros são obtidos de modo que a G_m **forneça resultados exatos para a I quando a integranda $g(t) = f(x(t))$ for um polinômio de grau até $2m - 1$** . Para tanto, forçamos a integração de $2m$ monômios $\{1, t, t^2, \dots, t^{2m-1}\}$, via eq. (6), de modo que G_m seja exata.

Assim, $\int_{-1}^1 g(t) dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)})$, com $g(t) = t^i$, para $i = 0, 1, 2, \dots,$

$2m - 1$, resultam as seguintes equações:

$$\left\{ \begin{array}{l} \int_{-1}^1 1 \cdot dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) \Rightarrow 2 = C_{(m,1)} \cdot 1 + C_{(m,2)} \cdot 1 + \dots + C_{(m,m)} \cdot 1 \\ \int_{-1}^1 t dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) \Rightarrow 0 = C_{(m,1)} t_{(m,1)} + C_{(m,2)} t_{(m,2)} + \dots + C_{(m,m)} t_{(m,m)} \\ \int_{-1}^1 t^2 dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) \Rightarrow 2/3 = C_{(m,1)} (t_{(m,1)})^2 + C_{(m,2)} (t_{(m,2)})^2 + \dots + C_{(m,m)} (t_{(m,m)})^2 \\ \int_{-1}^1 t^3 dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) \Rightarrow 0 = C_{(m,1)} (t_{(m,1)})^3 + C_{(m,2)} (t_{(m,2)})^3 + \dots + C_{(m,m)} (t_{(m,m)})^3 \\ \vdots \\ \int_{-1}^1 t^{2m-2} dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) \Rightarrow 2/(2m-1) = C_{(m,1)} (t_{(m,1)})^{2m-2} + C_{(m,2)} (t_{(m,2)})^{2m-2} + \dots + C_{(m,m)} (t_{(m,m)})^{2m-2} \\ \int_{-1}^1 t^{2m-1} dt = \sum_{k=1}^m C_{(m,k)} g(t_{(m,k)}) \Rightarrow 0 = C_{(m,1)} (t_{(m,1)})^{2m-1} + C_{(m,2)} (t_{(m,2)})^{2m-1} + \dots + C_{(m,m)} (t_{(m,m)})^{2m-1} \end{array} \right. \quad (7)$$

Esse sistema de equações não lineares é de ordem $2m$, e a sua solução, de grande dificuldade para ser obtida, fornece os parâmetros $C_{(m,k)}$ e $t_{(m,k)}$. Por exemplo, para $m = 2$, a expressão (7) torna-se:

$$\left\{ \begin{array}{l} C_{(2,1)} (t_{(2,1)})^0 + C_{(2,2)} (t_{(2,2)})^0 = 2 \\ C_{(2,1)} (t_{(2,1)})^1 + C_{(2,2)} (t_{(2,2)})^1 = 0 \\ C_{(2,1)} (t_{(2,1)})^2 + C_{(2,2)} (t_{(2,2)})^2 = 2/3 \\ C_{(2,1)} (t_{(2,1)})^3 + C_{(2,2)} (t_{(2,2)})^3 = 0 \end{array} \right.$$

A solução desse sistema pelo método de Newton, apresentado no Capítulo 4, fornece (em *double*):

$$\begin{aligned} C_{(2,1)} &= 1.00000000 \text{ e } t_{(2,1)} = +0.577350269189626 \\ C_{(2,2)} &= 1.00000000 \text{ e } t_{(2,2)} = -0.577350269189626 \end{aligned}$$

Generalizando a integração numérica de Gauss-Legendre para o intervalo $[a, b]$, aplicamos a eq. (6) na eq. (5) a fim de obter a aproximação de I :

$$I = \int_a^b f(x) dx = ((b-a)/2) \int_{-1}^1 \underbrace{f(x(t))}_{g(t)} dt \cong ((b-a)/2) G_m$$

Então, redefinimos a G_m da eq. (6), para o intervalo $[a, b]$, como:

$$I \cong G_m = ((b-a)/2) \sum_{k=1}^m C_{(m,k)} f(x(t_{(m,k)})) \tag{8}$$

Exemplo 8.6: determine $I = \int_0^1 e^x dx$ pelo método de Gauss-Legendre com $m = 2$ pontos.

Solução:

Aplicando diretamente a eq. (8) com $a = 0$ e $b = 1$, resulta:

m	$t_{(m,k)}$	$C_{(m,k)}$
2	$t_{(2,1)} = -1/\sqrt{3}$	$C_{(2,1)} = 1$
	$t_{(2,2)} = 1/\sqrt{3}$	$C_{(2,2)} = 1$

$$G_m = ((b-a)/2) \sum_{k=1}^m C_{(m,k)} f(x_k) = ((b-a)/2) \sum_{k=1}^m C_{(m,k)} y_k$$

k	$t_{(m,k)}$	$x_k = ((b+a)/2)t_{(m,k)} + ((b+a)/2)$	$y_k = f(x_k)$
1	$-1/\sqrt{3}$	0.211324865405187	1.23531360053585
2	$1/\sqrt{3}$	0.788675134594813	2.20047915547916

$$G_m = ((b-a)/2) [C_{(2,1)} f(x_1) + C_{(2,2)} f(x_2)]$$

$$G_m = \frac{(1-0)}{2} [1 * 1.23531360053585 + 1 * 2.20047915547916]$$

$$I \cong 1.71789637800750$$

$$I_e = 1.71828182845905 \text{ (integral exata } I_e = e^x|_0^1)$$

Na Tabela 8.3 estão listados os valores de $t_{(m,k)}$ e $C_{(m,k)}$, para $m = 1$ até 10, obtidos resolvendo o sistema não linear (7), em precisão *double*.

Tabela 8.3 – Valores de $t_{(m,k)}$ e $C_{(m,k)}$ para integração de Gauss-Legendre

m	$t_{(m,k)}$	$C_{(m,k)}$
1	$t_1 = 0$	$C_1 = 2$
2	$t_2 = -t_1 = \sqrt{3}^{-1}$	$C_1 = C_2 = 1$
3	$t_2 = 0$ $t_3 = -t_1 = \sqrt{3/5}$	$C_2 = 8/9$ $C_3 = C_1 = 5/9$
4	$t_3 = -t_2 = 0.3399810435848562648$ $t_4 = -t_1 = 0.8611363115940525752$	$C_3 = C_2 = 0.65214515486254614263$ $C_4 = C_1 = 0.34785484513745385737$
5	$t_3 = 0$ $t_4 = -t_2 = 0.53846931010568309104$ $t_5 = -t_1 = 0.90617984593866399280$	$C_3 = 128/225$ $C_4 = -C_2 = 0.47862867049936646804$ $C_5 = -C_1 = 0.23692688505618908751$
6	$t_4 = -t_3 = 0.23861918608319690863$ $t_5 = -t_2 = 0.66120938646626451366$ $t_6 = -t_1 = 0.93246951420315202781$	$C_4 = -C_3 = 0.46791393457269104739$ $C_5 = -C_2 = 0.36076157304813860757$ $C_6 = -C_1 = 0.17132449237917034504$
7	$t_4 = 0$ $t_5 = -t_3 = 0.40584515137739716691$ $t_6 = -t_2 = 0.74153118559939443986$ $t_7 = -t_1 = 0.94910791234275852453$	$C_4 = 512/1225$ $C_5 = -C_3 = 0.38183005050511894495$ $C_6 = -C_2 = 0.27970539148927666790$ $C_7 = -C_1 = 0.129484966168869693271$
8	$t_7 = -t_4 = 0.183434642495650$ $t_6 = -t_3 = 0.525532409916329$ $t_7 = -t_2 = 0.796666477413627$ $t_8 = -t_1 = 0.960289856497536$	$C_5 = -C_4 = 0.362683783372369$ $C_6 = -C_3 = 0.313706645877885$ $C_7 = -C_2 = 0.222381034453374$ $C_8 = -C_1 = 0.101228536290377$
9	$t_5 = 0$ $t_6 = -t_4 = 0.324253423403809$ $t_7 = -t_3 = 0.613371432700591$ $t_8 = -t_2 = 0.836031107326639$ $t_9 = -t_1 = 0.968160239507622$	$C_5 = 0.330239355001305$ $C_6 = -C_4 = 0.312347077039964$ $C_7 = -C_3 = 0.260610696402964$ $C_8 = -C_2 = 0.180648160694839$ $C_9 = -C_1 = 0.0812743883615805$
10	$t_6 = -t_5 = 0.148874338981631$ $t_7 = -t_4 = 0.433395394129247$ $t_8 = -t_3 = 0.679409568299024$ $t_9 = -t_2 = 0.865063366688989$ $t_{10} = -t_1 = 0.973906528517168$	$C_6 = -C_5 = 0.295524224714756$ $C_7 = -C_4 = 0.269266719309977$ $C_8 = -C_3 = 0.219086362516008$ $C_9 = -C_2 = 0.149451394150562$ $C_{10} = -C_1 = 0.0666713443086937$

Fonte: Adaptada de Gaelzer (2013).

Para fins didáticos, apresentamos a Tabela 8.4, com precisão de calculadoras científicas.

Tabela 8.4 – Valores de $t_{(m,k)}$ e $C_{(m,k)}$ para integração de Gauss-Legendre, com precisão de calculadoras e $m = 1$ até 5

m	$t_{(m,k)}$	$C_{(m,k)}$
1	$t_1 = 0$	$C_1 = 2$
2	$t_1 = -\sqrt{3}^{-1}$	$C_1 = 1$
	$t_2 = \sqrt{3}^{-1}$	$C_2 = 1$
3	$t_1 = -\sqrt{3/5}$	$C_1 = 5/9$
	$t_2 = 0$	$C_2 = 8/9$
	$t_3 = \sqrt{3/5}$	$C_3 = 5/9$
4	$t_1 = -0.861136312$	$C_1 = 0.347854845$
	$t_2 = -0.339981044$	$C_2 = 0.652145155$
	$t_3 = +0.339981044$	$C_3 = 0.652145155$
	$t_4 = +0.861136312$	$C_4 = 0.347854845$
5	$t_1 = -0.906179846$	$C_1 = 0.236926885$
	$t_2 = -0.538469310$	$C_2 = 0.478628670$
	$t_3 = 0$	$C_3 = 0.568888889$
	$t_4 = +0.538469310$	$C_4 = 0.478628670$
	$t_5 = +0.906179846$	$C_5 = 0.236926885$

Fonte: Elaboração própria.

Os valores dos m “nós” $t_{(m,k)}$ são as raízes dos polinômios ortogonais de Legendre de grau m (CARNAHAN; LUTHER; WILKES, 1990), distribuídas simetricamente no intervalo normalizado $[-1, 1]$. Assim, se forem obtidos os valores de $t_{(m,k)}$, via determinação das raízes dos polinômios ortogonais de Legendre de grau m (POLINÔMIOS..., 2017), o sistema (7), de $2m$ equações, se tornará linear e permitirá a obtenção dos m valores dos “pesos” $C_{(m,k)}$ usando as suas primeiras m equações lineares

Os polinômios de Legendre podem ser obtidos genericamente para qualquer grau n por meio de relações de recorrência, partindo dos dois primeiros polinômios $P_0(x) = 1$ e $P_1(x) = 0 + 1x$, conforme as relações:

$$P_{n+1}(x) = ((2n + 1) * x P_n(x) - n * P_{n-1}(x)) / (n + 1)$$

Para utilização numérica, obtemos diretamente os coeficientes por meio das seguintes recorrências, para cada grau n :

$$\text{a) grau } n = 1 \rightarrow a_{(n,1)} = 0; a_{(n,2)} = 1; P_1(x) = 0 + 1x$$

$$\text{b) grau } n = 2 \rightarrow a_{(n,1)} = -1/2; a_{(n,2)} = 0; a_{(n,3)} = 3/2; P_2(x) = (-1 + 0x + 3x^2)/2$$

c) graus $n = 3, 4, \dots, m$

$$k = 0 \rightarrow a_{(n,k+1)} = (1 - n) / n * a_{(n-2,k+1)}$$

$$k = 1 : n - 2 \rightarrow a_{(n,k+1)} = (1 - n) / n * a_{(n-2,k+1)} + (2 * n * 1) / n * a_{(n-1,k-1)}$$

$$k = n - 1 \rightarrow a_{(n,k+1)} = (2 * n - 1) / n * a_{(n-1,k)}$$

$$k = n \rightarrow a_{(n,k+1)} = (2 * n - 1) / n * a_{(n-1,k+1-1)}$$

$$\text{e } P_n(x) = a_{(n,1)} + a_{(n,2)}x + a_{(n,3)}x^2 + \dots + a_{(n,n+1)}x^n$$

No **Caderno de Algoritmos**, você encontra o cálculo dos valores de $t_{(m,k)}$ e $C_{(m,k)}$ para qualquer m no arquivo **Cap8CalculoCoefGaussLegendre.m**.

Por último, podemos provar, utilizando o teorema do valor médio para integrais, que o limite do erro de truncamento EG_m da integração numérica de Gauss-Legendre, utilizando a integranda original $f(x)$, é dado por

$$EG_m \leq (b-a)^{2m+1} \frac{(m!)^4}{(2m+1)[(2m)!]^3} \text{Max} |f^{(2m)}(x)|, \quad \forall x \in [a, b] \quad (9)$$

$$\lim_{m \rightarrow \infty} EG_m = 0$$

Exemplo 8.7: determine o número de pontos m mínimo para que o erro de truncamento máximo entre a integral exata $I = \int_0^6 (1+x)^{-1} dx$ e a aproximação por Gauss-Legendre G_m seja da ordem de $O(10^{-6})$.

Solução:

Derivando a integranda, resulta que:

$$f(x) = (1 - x)^{-1}, f'(x) = (-1)(1 + x)^{-2}, f''(x) = (-1)(-2)(1 + x)^{-3}$$

$$f'''(x) = (-1)(-2)(-3)(1 + x)^{-4}$$

$$f^{(4)}(x) = (-1)(-2)(-3)(-4)(1 + x)^{-5}$$

e generalizando para derivada de ordem k , temos

$$f^{(k)}(x) = (-1)^k k!(1 + x)^{-(k+1)}$$

Agora vamos calcular o erro requerido EG_m da ordem de $O(10^{-6})$ por tentativas, iniciando com $m = 6$ e usando a expressão (8).

Para $m = 6$:

$$\text{Max} \left| f^{(2m)}(x) \right|_{x=1} = f^{(12)}(1) = (-1)^{12} \frac{12!}{(1+1)^{(12+1)}} = 58472$$

$$E_{Gm} \leq (6-1)^{2*6+1} \frac{(6!)^4}{(2*6+1)[(2*6)!]^3} \text{Max} \left| f^{(2m)}(x) \right| = 0.01342557$$

Para $m = 10$:

$$\text{Max} \left| f^{(2m)}(x) \right|_{x=1} = f^{(20)}(1) = (-1)^{20} \frac{20!}{(1+1)^{(20+1)}} = 1.1601 * 10^{12}$$

$$E_{Gm} \leq (6-1)^{2*10+1} \frac{(10!)^4}{(2*10+1)[(2*10)!]^3} \text{Max} \left| f^{(2m)}(x) \right| = 0.00031719$$

Para $m = 16$:

$$\text{Max} \left| f^{(2m)}(x) \right|_{x=1} = f^{(32)}(1) = (-1)^{32} \frac{32!}{(1+1)^{(32+1)}} = 3.06325 * 10^{25}$$

$$E_{Gm} \leq (6-1)^{2*16+1} \frac{(16!)^4}{(2*16+1)[(2*16)!]^3} \text{Max} \left| f^{(2m)}(x) \right| = 1.13669 * 10^{-06}$$

Portanto, são necessários $m = 16$ pontos no método de Gauss-Legendre para que essa integral seja da ordem de $O(10^{-6})$, enquanto nos métodos de Simpson e trapézios necessitamos de $n = 60$ e $n = 1614$ subdivisões do intervalo $[a, b]$, o que equivale a usar 61 e 1615 pontos, respectivamente.

Exemplo 8.8: determine $I = \int_1^6 \frac{1}{1+x} dx$ pelo método de Gauss-Legendre com $m = 3$ pontos, calcule o erro exato, determine e plote o polinômio de grau $n = m - 1 = 2$, que passa sobre os 3 pontos (x_k, y_k) definidos pelo método, e compare a integração deste polinômio $P_{m-1}(x)$ com o resultado da integração pelo método de Gauss-Legendre.

Solução:

Aplicando diretamente a eq. (8) com $a = 1$ e $b = 6$, resulta:

m	$t_{(m,k)}$	$C_{(m,k)}$
3	$t_1 = -\sqrt{3/5}$	$C_1 = 5/9$
	$t_2 = 0$	$C_2 = 8/9$
	$t_3 = \sqrt{3/5}$	$C_3 = 5/9$

$$G_m = ((b-a)/2) \sum_{k=1}^m C_{(m,k)} f(x_k) = ((b-a)/2) \sum_{k=1}^m C_{(m,k)} y_k$$

k	$t_{(m,k)}$	$x_k = ((b-a)/2)t_{(m,k)} + ((b+a)/2)$	$y_k = f(x_k)$
1	$-\sqrt{3/5}$	1.563508325	0.390090404719087
2	0	3.5	0.2222222222222222
3	$\sqrt{3/5}$	5.436491675	0.155364140978245

$$G_3 = \frac{(b-a)}{2} [C_{(3,1)} f(x_1) + C_{(3,2)} f(x_2) + C_{(3,3)} f(x_3)]$$

$$G_3 = \frac{(6-1)}{2} [5/9 * 0.390090404719087 + 8/9 * 0.2222222222222222 + 5/9 * 0.155364140978245]$$

$$G_3 = \frac{(6-1)}{2} [0.216716891510604 + 0.197530864197531 + 0.086313411654580545]$$

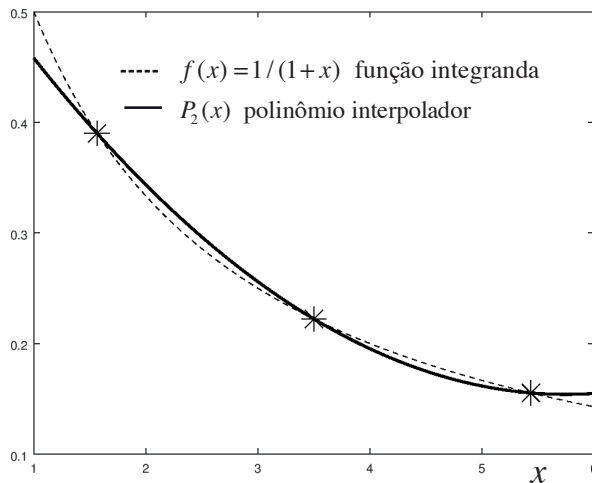
$$G_3 = \frac{(6-1)}{2} [0.50056116736271645]$$

$$I \cong G_3 = 1.25140291840679$$

$$I_e = 1.25276296849537 \text{ (integral exata } I_e = \ln(1+x)|_1^6)$$

$$\text{Erro } G_3 = |G_3 - I_e| = 0.00136005042578313 \text{ (erro exato da ordem de } O(10^{-3})\text{)}.$$

Gráfico 8.6 – Comparativo da função integranda $f(x)$ e da função aproximadora $P_2(x)$ fixada nos $m = 3$ pontos definidos no método de Gauss-Legendre



Fonte: Elaboração própria.

Se determinarmos o polinômio de grau $n = m - 1 = 2$, que passa sobre os $m = 3$ pontos (x_k, y_k) , teremos:

$$P_2(x) = 0.5993265993265994 - 0.1548821548821549x + 0.0134680134680135x^2$$

Agora, se integrarmos esse polinômio $P_2(x)$ entre $a = 1$ e $b = 6$, teremos I_{P_2} :

$$I_{P_2} = \int_a^b P_2(x) dx$$

$$I_{P_2} = \int_1^6 (0.5993265993265994 - 0.1548821548821549x + 0.0134680134680135x^2) dx$$

$$I_{P_2} = 1.25140291806958 \text{ que é idêntico a } G_3 = 1.25140291806958$$

Observe que a integração efetuada pelo Método de Gauss-Legendre G_3 (eq. 8) tem “exatamente” o mesmo resultado da integração I_{P_2} do polinômio $P_2 = P_{n=m-1}(x)$, que passa sobre os m pontos internos utilizados pelo método de Gauss-Legendre.

Considerações:

- a) Observando a expressão do erro de Gauss-Legendre, comparado com o erro do método de Simpson para o mesmo número de chamadas da função, notamos que Gauss-Legendre é mais preciso.
- b) Pelas suas características, Gauss-Legendre é numericamente mais estável do que os métodos de Newton, uma vez que a quantidade de parâmetros $t_{(m,k)}$ e $C_{(m,k)}$ é pequena e assim não vai acumular arredondamentos.
- c) Gauss-Legendre não é aplicável se a integranda $y = f(x)$ for uma tabela de pontos discretos.
- d) Gauss-Legendre fornece resultados pobres se $y = f(x)$ possuir descontinuidades dentro do intervalo $[a, b]$, como ocorre no **Exemplo 8.9**.
- e) Gauss-Legendre é de natureza aberta, isto é, a integranda não é avaliada nos extremos a e b . Em decorrência disso, esse método pode ser aplicado em integrais impróprias, em que a integranda possui descontinuidades nos extremos do intervalo a e/ou b , ou a e/ou b podem ser infinito(s), conforme podemos verificar nos **Exemplos 8.9 e 8.10**.
- f) No(s) caso(s) de extremo(s) com valor(es) infinito(s), teremos:

- i) Se apenas um extremo for ∞ , $I = \int_a^{\infty} f(x) dx$, aplicaremos a transformação de variáveis:

$$u = 1/x \Rightarrow x = 1/u \Rightarrow dx = -1/u^2 du \text{ com } [a, \infty) \rightarrow [1/a, 0] \text{ resultando}$$

$$\text{em } \int_a^{\infty} f(x) dx = - \int_{1/a}^0 \frac{f(1/u)}{u^2} du = \int_0^{1/a} \frac{f(1/u)}{u^2} du, \text{ que é uma integral}$$

originalmente imprópria, com descontinuidade no extremo, mas superada pela mudança de variáveis, $1/\infty = 0$. O método de Gauss-Legendre pode então ser aplicado como no **Exemplo 8.9**.

- ii) Se um extremo for ∞ e o outro for nulo, também poderemos aplicar o método subdividindo o intervalo de integração original em duas partes, por exemplo:

$$\int_0^{\infty} f(x) dx = \int_0^1 f(x) dx + \int_1^{\infty} f(x) dx$$

- iii) Se os dois extremos forem ∞ , também poderemos subdividir em 2 partes, recaindo em duas integrais como nos casos anteriores:

$$dx = \int_{-\infty}^a f(x) dx + \int_a^{\infty} f(x) dx$$

Exemplo 8.9: do cálculo, sabemos que a igualdade $I = \int_1^3 \frac{dx}{2-x} = \int_1^2 \frac{dx}{2-x} + \int_2^3 \frac{dx}{2-x}$ é sempre válida. Efetue numericamente essas duas formas da mesma integral por Gauss-Legendre até $m = 7$ e verifique que essa igualdade teórica não é mais verdadeira, devido ao ponto de singularidade em $x = 2$ no intervalo $[1, 3]$.

Solução:

Aplicando a eq.(8), resulta:

m	$\int_1^3 \frac{dx}{2-x}$	$\int_1^2 \frac{dx}{2-x} + \int_2^3 \frac{dx}{2-x}$
2	3	$-2.22044604925031e - 15$
3	<i>Inf</i>	0
4	$8.88178419700125e - 16$	$-7.99360577730113e - 15$
5	<i>Inf</i>	$-8.88178419700125e - 16$
6	$-1.38777878078145e - 15$	$-1.77635683940025e - 15$
7	<i>Inf</i>	$8.88178419700125e - 16$

Usando o algoritmo apresentado no **Caderno de Algoritmos**, observe que a integral $\int_1^3 \frac{dx}{2-x}$, aproximada por Gauss-Legendre, gera valores inconsistentes, conforme o m utilizado por conter um ponto de singularidade dentro do intervalo. Com m ímpar a integranda gera valor infinito (*Inf* no Octave) exatamente para o ponto medio $x = 2$.

Se o ponto de singularidade estiver localizado em algum extremo, como nas integrais particionadas, antes e depois da descontinuidade, poderemos aplicar o método de Gauss-Legendre e obter um resultado, neste caso exato, já a partir de $m = 2$. Estas integrações separadas $\int_1^2 \frac{dx}{2-x}$ e $\int_2^3 \frac{dx}{2-x}$ geram valores simétricos, então a soma dessas duas partes geram o resultado exato, que é nulo. E podemos verificar analiticamente que o valor exato dessa integral é zero pois, em $I = \int_1^2 \frac{dx}{2-x} + \int_2^3 \frac{dx}{2-x} = 0$, a área positiva entre $a = 1$ e $b = 2$ se compensa completamente com a área negativa entre $b = 2$ e $c = 3$.

Exemplo 8.10: efetue $I = \int_0^1 \ln(x) dx$ por Gauss-Legendre com $m = 3$ e compare o resultado obtido com o valor exato de $I_e = -1$.

Solução:

Aplicando a eq. (8) com $m = 3$, temos:

$$G_m = -0.94767237 \Rightarrow \text{Erro } G_m = |-0.94767237 - (-1)| = 0.05232763$$

Na próxima seção, apresentaremos um método gaussiano de integração, cujo propósito específico é a determinação otimizada dos coeficientes das séries aproximadoras de funções, usando polinômios de Tchebychev, conforme vimos no Capítulo 6.

8.2.2 Método Gauss-Tchebychev

A maneira de determinar os parâmetros do método de Gauss-Legendre pode ser utilizada para gerar outros métodos de integração com propósitos específicos. Por exemplo, para uma integral particular do tipo $I = \int_{-1}^{+1} \frac{f(x)}{\sqrt{1-x^2}} dx$, cuja $f(x)$ é uma função simples e bem comportada, mas nenhum dos métodos de Newton-Cotes poderá ser utilizado devido às descontinuidades em -1 e 1 .

Já o método de Gauss-Legendre poderá não ser eficiente (veja o **Exemplo 8.11**). Por consequência, Gauss propôs uma integração numérica na forma

$$\int_{-1}^{+1} \frac{f(x)}{\sqrt{1-x^2}} dx \cong GT_m = \sum_{j=1}^m a_j f(x_j) \quad (10)$$

Em que, na parte direita da eq. (10) foi usada apenas a parcela bem comportada $f(x)$ da integranda $f(x)/\sqrt{1-x^2}$. Podemos demonstrar que, impondo a condição de que a GT_m seja exata para integrar polinômios de grau até $2m-1$, resulte os seguintes valores para os “pesos” a_j e os “nós” x_j que são as *raízes do polinômio de Tchebychev de grau m* .

$$a_j = \pi / m$$

$$x_j = \cos\left(\frac{(2j-1)}{2m} \pi\right), \quad \forall j = 1, 2, \dots, m$$

Então

$$GT_m = \left(\frac{\pi}{m}\right) \sum_{j=1}^m f\left(\frac{(2j-1)}{2m} \pi\right) \quad (11)$$



Lembramos que as raízes do polinômio de Tchebychev de grau m já foram usadas no Capítulo 6, quando utilizamos esse método de integração, para determinar os coeficientes da série Tchebychev (VANDERGRAFT, 1983).

O erro de truncamento da eq. (11) pode ser estimado por meio da seguinte equação:

$$R_m = \frac{\pi}{(2m)! 2^{2m-1}} f^{(2m)}(\xi), \quad -1 < \xi < +1 \quad (12)$$

Agora, vamos apresentar um exemplo de integração pelo método de Gauss-Tchebychev.

Exemplo 8.11: determine o valor da integral $I = \int_{-1}^{+1} \frac{x \operatorname{sen}(x)}{\sqrt{1-x^2}} dx$ por Gauss-Tchebychev e compare-o com os resultados de Gauss-Legendre. Efetue tentativas com m crescente até que o resultado atinja a precisão utilizada.

Solução:

Aplicando as eqs. (11) e (6) para obter, respectivamente, os valores de GT_m e G_m , com $m = 2, 3, \dots, 10$, resulta:

m	GT_m	G_m
2	1.44313043631812	0.771885702690283
3	1.38167976999193	0.951800774731612
4	1.38246439058891	1.05120156898943
5	1.38245967093080	1.11295823247018
6	1.38245968742174	1.15516336143599
7	1.38245968738411	1.18587293446304
8	1.38245968738417	1.20923821130433
9	1.38245968738417	1.22762077044289
10	1.38245968738417	1.24246518890738

Por Gauss-Tchebychev, aplicamos a eq. (11) apenas à parcela $f(x) = x \operatorname{sen}(x)$ da função integranda (sem o fator peso $1/\sqrt{1-x^2}$), pois esta integração GT_m já considera esse fator). Para essa função, obtemos a convergência da integração aproximada com $m = 8$ “nós”.

Por Gauss-Legendre, utilizamos a função completa, com o fator peso,

$$f(x) = \frac{x \operatorname{sen}(x)}{\sqrt{1-x^2}},$$

mas os resultados ficam longe da convergência para a solução. Mesmo com $m = 10$, verificamos a ineficiência do método para essa integral com descontinuidades em ambas extremidades. Então, a integração por Gauss-Tchebychev é a mais adequada, por já considerar esse fator peso com descontinuidades nos extremos.

Em tempo, se a $f(x)$ que compõe a integranda possuir singularidades no domínio de integração, teremos de usar valores para o m muito elevados também para a integração por Gauss-Tchebychev.

No **Caderno de Algoritmos**, você encontra os algoritmos do método dos trapézios, de Simpson e de Gauss-Legendre no arquivo **Cap8IntegraisTnSnGm.m**, e o algoritmo do método de Gauss-Tchebychev no arquivo **Cap8IntegralGaussTchebychev.m**.

8.3 CONCLUSÕES

Nenhum dos quatro métodos de integração numérica abordados neste capítulo deve ser descartado *a priori*, pois cada um deles poderá ser o mais adequado, conforme o tipo de integranda disponível:

- a) o método dos trapézios será o mais adequado para funções integrandas com gráfico tipo escada;
- b) o método de Simpson é indicado para funções integrandas discretas (definidas por tabela de pontos);
- c) o método de Gauss-Legendre é mais eficiente, em geral, para funções integrandas com expressão conhecida;
- d) a integração numérica de Gauss-Tchebychev tem aplicações específicas e deve ser utilizada quando a função integranda já inclui o fator peso $W(x) = 1/\sqrt{1-x^2}$ na sua expressão original, como na determinação de coeficientes da série aproximadora de Tchebychev utilizada no Capítulo 6; e
- e) na aplicação do método de Gauss-Legendre, caso se esgote a tabela de parâmetros $t_{(m,k)}$ e $C_{(m,k)}$ disponível e a precisão desejada ainda não tenha sido atingida, poderemos adotar a composição de integrais $I = \int_a^b f(x) dx = \int_a^{b/2} f(x) dx + \int_{b/2}^b f(x) dx$ para tentar atingir a precisão desejada, efetuando essas duas integrais com os m parâmetros disponíveis.

Por fim, também podemos utilizar essa composição de integrais nos métodos newtonianos.

Confira se você teve um bom entendimento do que abordamos neste capítulo realizando as atividades propostas no **Caderno de Exercícios e Respostas**. Essa ação será muito importante para a fixação do conteúdo e para a continuidade dos seus estudos.

RESOLUÇÃO NUMÉRICA DE EQUAÇÕES DIFERENCIAIS ORDINÁRIAS

OBJETIVOS ESPECÍFICOS DE APRENDIZAGEM

Ao finalizar este capítulo, você será capaz de:

- determinar soluções numéricas de equações diferenciais ordinárias de 1ª ordem (Problemas de Valor Inicial – PVI) por métodos de Euler, de Runge-Kutta e de Adams-Bashforth;
- determinar soluções numéricas de PVI de ordem n pelos métodos apresentados;
- determinar soluções numéricas de Problemas de Valor no Contorno (PVC) pelo método clássico das tentativas (*shooting*) ou pelo método de Newton;
- determinar soluções numéricas de PVC com uma ou mais condições de contorno desconhecida(s); e
- utilizar os algoritmos disponibilizados.

No Capítulo 3, abordamos a solução numérica de uma equação algébrica $f(x) = 0$ envolvendo uma única variável desconhecida x . Neste capítulo, trataremos de soluções numéricas de outros tipos de equações, que envolvem a determinação de uma função desconhecida e que ampliam a capacidade de modelagem de fenômenos.

Definição 1: uma **equação diferencial** é toda equação que envolve derivada(s) de uma função desconhecida $y = y(x)$. Se nessa equação estiverem envolvidas derivadas em relação a uma única variável, até a ordem n , ela será denominada de **Equação Diferencial Ordinária** de ordem n , denotada pela sigla EDO, e genericamente representada por:

$$y^{(n)} = f(x, y, y', \dots, y^{(n-1)}) \quad \text{com } x \in [a, b], \quad (1)$$

em que $y = y(x)$ é uma função desconhecida, $y', y'', \dots, y^{(n-1)}$ são as suas respectivas **derivadas ordinárias** até a ordem $(n - 1)$.



Derivadas ordinárias são em relação a uma única variável. Se as derivadas forem em relação a mais de uma variável, teremos derivadas parciais, que geram **Equações Diferenciais Parciais**, denotadas pela sigla EDP.

Definição 2: a **solução de uma EDO** de ordem n é toda função $y = y(x)$, definida em $[a, b]$, que possui as n derivadas ordinárias nesse domínio e satisfaz à equação diferencial (1).

Como uma EDO pode ter mais de uma solução, por exemplo, a equação $y' = x^2$ possui a família de soluções $y(x) = (x^3 / 3) + C$, em que C é uma constante, temos de impor condições adicionais para obter numericamente uma solução específica. Tais condições complementares podem ser de dois tipos:

- a) Na eq. (1), impondo que $y(a) = u_1, y'(a) = u_2, \dots, y^{(n-1)}(a) = u_n$ (n condições em um único ponto $x = a$), a EDO é denominada de **Problema de Valor Inicial (PVI)**. Por exemplo, o PVI $y' = -y * \operatorname{tg}(x)$, com $y(0) = 1$ e $x \in [0, 1]$, tem solução $y(x) = \cos(x)$.
- b) Se, além de algumas condições do tipo (a), também forem impostas algumas condições na outra extremidade do intervalo, como $y(b) = v_1, y'(b) = v_2, \dots, y^{(n-1)}(b) = v_n$, a EDO será denominada de **Problema de Valor no Contorno (PVC)** (total de condições dever ser n).

As equações diferenciais ordinárias são usadas com muita frequência na modelação de fenômenos da natureza como a taxa de variação da posição x de um móvel em função do tempo t e de sua aceleração a . No caso de aceleração a constante, temos o movimento uniformemente variado, amplamente conhecido das disciplinas introdutórias de Física, que pode ser modelado pelo seguinte PVI de 2ª ordem:

$$\frac{d^2 x(t)}{dt^2} = a, \text{ em que } a = \text{aceleração do móvel.}$$

Condições iniciais:

$$\frac{dx(t=0)}{dt} = v_0, \text{ em que } v_0 = \text{valor da sua velocidade inicial.}$$

$$x(t=0) = x_0, \text{ em que } x_0 = \text{valor da sua posição inicial.}$$

Nesse caso, obtendo a solução dessa EDO por integração analítica direta, resulta:

$$\frac{d}{dt} \left(\frac{dx(t)}{dt} \right) = a \Rightarrow \int d \left(\frac{dx(t)}{dt} \right) = \int a * dt \Rightarrow \frac{dx(t)}{dt} = a * t + C_1$$

Integrando novamente a solução, temos:

$$dx(t) = (a * t + C_1) dt \Rightarrow \int dx(t) = \int (a * t + C_1) dt \Rightarrow x(t) = a * t^2 / 2 + C_1 * t + C_2$$

Aplicando as condições iniciais $\frac{dx}{dt}(t=0) = v_0$ e $x(t=0) = x_0$, temos a conhecida equação do movimento uniformemente variado:

$$x(t) = a * t^2 / 2 + v_0 * t + x_0$$

Existem vários métodos analíticos de resolução de equações diferenciais ordinárias, mas na maioria dos casos não é possível representar a solução em termos de funções elementares, como no modelo matemático anterior. Mesmo uma equação diferencial com aspecto simples como:

$$y' = x^2 + y^2$$

não pode ser resolvida em termos de funções elementares. Nesses casos, os métodos numéricos permitem encontrar as funções solução na forma discreta e aproximada.

Inicialmente, vamos abordar alguns métodos numéricos para resolver uma EDO do tipo PVI de 1ª ordem com fundamentação teórica no desenvolvimento da solução $y = y(x)$ em série de Taylor. Por esse motivo, esses métodos são chamados de métodos baseados na série de Taylor.

9.1 MÉTODO DE EULER SIMPLES

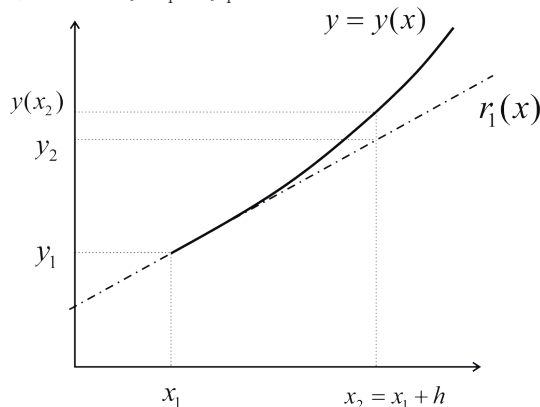
Seja um PVI de 1ª ordem:

$$y' = f(x, y) \text{ com } x \in [a, b] \text{ e condição inicial } y(x_1) = y_1 \quad (2)$$

em que $f(x, y)$ representa a derivada da função solução $y(x)$.

Suponhamos que a solução exata da equação diferencial (2) seja dada por $y = y(x)$, conforme o Gráfico 9.1 a seguir.

Gráfico 9.1 – Representação de uma solução $y = y(x)$ e da reta tangente $r_1(x)$ no ponto definido pela condição inicial $y(x_1) = y_1$



Fonte: Elaboração própria.

Como conhecemos o ponto inicial da solução desejada (x_1, y_1) , e o coeficiente angular $y'(x_1) = f(x_1, y_1)$ da reta $r_1(x)$ tangente à essa solução $y(x)$ no ponto (x_1, y_1) , então podemos obter a expressão algébrica dessa reta:

$$r_1(x) = y_1 + (x - x_1)y'(x_1) = y_1 + (x - x_1)f(x_1, y_1) \quad (3)$$

No Gráfico 9.1, considerando $h = (x_2 - x_1)$ como o espaçamento entre os dois primeiros valores da variável independente x e aplicando-o na eq. (3), temos $y_2 = r_1(x_2)$:

$$y_2 = y_1 + (x_2 - x_1)f(x_1, y_1) = y_1 + h * f(x_1, y_1), \text{ em que } x_2 = x_1 + h.$$

Dividindo o domínio $[a, b]$ em n partes iguais, resulta o passo constante $h = (b - a) / n$; e generalizando a eq.(3), para todo $k = 1, 2, \dots, n$, temos:

$$y_{k+1} = y_k + h * f(x_k, y_k) \quad (4a)$$

$$x_{k+1} = x_k + h \quad (4b)$$

Assim, podemos obter n aproximações discretas da solução $y = y(x)$ do PVI. Esse é o denominado *método de Euler simples*⁹.



Esse método é classificado como de passo simples porque cada nova aproximação y_{k+1} é calculada somente a partir da aproximação anterior y_k .

Note que $y_2 = y_1 + h * f(x_1, y_1)$ é o valor da solução $y(x)$, em $x = x_2$, desenvolvida em série de Taylor em torno de $x = x_1$ e truncada a partir do terceiro termo.

Exemplo 9.1: resolva por Euler simples o PVI:

$$y' = x - y + 2, \text{ com } y(x = 0) = 2 \text{ no domínio } [0, 1]$$

Escolha um valor inicial para o número n de subdivisões do domínio e aumente o n até que $y(x = 1)$ tenha erro estimado da ordem de 10^{-6} .

Solução:

Temos que $f(x, y) = x - y + 2$ e $y(x = 0) = 2$.

Tomando $n = 8$ para o número de subdivisões e aplicando as eqs. (4), resulta:

$$h = (1 - 0) / 8 = 0.125$$

$$x_1 = 0 \quad \text{e} \quad y_1 = 2$$

$$x_2 = x_1 + h \quad \text{e} \quad y_2 = y_1 + h * f(x_1, y_1)$$

$$x_2 = 0 + 0.125 = 0.125 \quad y_2 = 2 + 0.125 * (0 - 2 + 2) = 2$$

$$x_3 = x_2 + h \quad \text{e} \quad y_3 = y_2 + h * f(x_2, y_2)$$

$$x_3 = 0.125 + 0.125 = 0.250 \quad y_3 = 2 + 0.125 * (0.125 - 2 + 2) = 2.015625$$



É recomendável que o h inicial seja um valor relativamente pequeno, como $h \sim O(0.1)$.

Assim, sucessivamente, resultando em:

k	x_k	y_k	$K_1 = f(x_k, y_k)$	x_{k+1}	y_{k+1}
1	0	2	0.000000000	0.125	2.000000000
2	0.12500	2.000000000	0.125000000	0.250	2.015625000
3	0.25000	2.015625000	0.234375000	0.375	2.044921875
4	0.37500	2.044921875	0.330078125	0.500	2.086181641
5	0.50000	2.086181641	0.413818359	0.625	2.137908936
6	0.62500	2.137908936	0.487091064	0.750	2.198795319
7	0.75000	2.198795319	0.551204681	0.875	2.267695904
8	0.87500	2.267695904	0.607304096	1.000	2.343608916
9	1.00000	2.343608916			

Portanto, para $n = 8$, $y(x = 1) = y_9 \cong 2.343608916$.

Qual o erro associado ao último ponto da solução obtida?

Se repetirmos todo o cálculo com o dobro de subdivisões, $n = 16$, teremos um valor aproximado $y(x = 1)$ com erros de truncamento de ordem inferior, que pode ser considerado um valor estimativo mais próximo do exato do que o valor obtido com $n = 8$.

Tomando $n = 16$ divisões do domínio e aplicando as eqs. (4), temos:

$$h = (1 - 0) / 16 = 0.0625$$

$$x_2 = 0.0625 \quad \text{e} \quad y_2 = y_1 + h * f(x_1, y_1) = 2 + 0.0625 * (0 - 2 + 2) = 2$$

$$x_3 = 0.1250 \quad \text{e} \quad y_3 = y_2 + h * f(x_2, y_2) = 2 + 0.0625 * (0.0625 - 2 + 2) \\ = 2.003906250$$

Assim, sucessivamente, resultando:

k	x_k	y_k	$K_1 = f(x_k, y_k)$	x_{k+1}	y_{k+1}
1	0	2	0.000000000	0.06250	2.0000000000
2	0.06250	2.000000000	0.062500000	0.12500	2.003906250
3	0.12500	2.003906250	0.121093750	0.18750	2.011474609
4	0.18750	2.011474609	0.176025391	0.25000	2.022476196
5	0.25000	2.022476196	0.227523804	0.31250	2.036696434
6	0.31250	2.036696434	0.275803566	0.37500	2.053934157
7	0.37500	2.053934157	0.321065843	0.43750	2.074000772
8	0.43750	2.074000772	0.363499228	0.50000	2.096719474
9	0.50000	2.096719474	0.403280526	0.56250	2.121924507
10	0.56250	2.121924507	0.440575493	0.62500	2.149460475

11	0.62500	2.149460475	0.475539525	0.68750	2.179181695
12	0.68750	2.179181695	0.508318305	0.75000	2.210951589
13	0.75000	2.210951589	0.539048411	0.81250	2.244642115
14	0.81250	2.244642115	0.567857885	0.87500	2.280133233
15	0.87500	2.280133233	0.594866767	0.93750	2.317312406
16	0.93750	2.317312406	0.620187594	1.00000	2.356074130
17	1.00000	2.356074130			

Portanto, para $n = 16$, $y(x = 1) = y_{17} \cong 2.356074130$.

Erro estimado Euler = $\left| y_9^{(n=8)} - y_{17}^{(n=16)} \right| = 0.012465214 > 10^{-6}$.

Esse erro demonstra que a solução ainda está muito distante da precisão requerida. Por simulação em computador, teremos de aumentar o n até $n = 2^{15} = 32768$ para chegar a um erro estimado de $2.8068e - 06$, que é da ordem de 10^{-06} .

A seguir, apresentaremos métodos de maior precisão, para evitar o *aumento exagerado no número n* de subdivisões do domínio.



Assim como nos métodos de integração numérica de funções que vimos no Capítulo 8, esses métodos de passo simples podem acumular erros de arredondamento exagerados se n for muito grande, pois um novo ponto depende do anterior.

9.2 MÉTODOS DE RUNGE-KUTTA

Os métodos de Runge-Kutta são obtidos tomando mais termos nas séries de Taylor para aproximar soluções de PVI. Se forem cancelados os termos que contêm derivadas de ordens maiores do que p , o método será considerado de ordem p . O método de Euler simples estudado anteriormente é de primeira ordem ($p = 1$), porque despreza os termos com derivadas de ordem 2 e superiores.

Na sequência, vamos apresentar alguns métodos dessa categoria.

com expressão algébrica:

$$r_k(x) = y_k + (x - x_k)y'(x_k) = y_k + (x - x_k)f(x_k, y_k) \quad (7)$$

Assim, aplicando o método de Euler com passo h , determinamos

$$\bar{y}_{k+1} = y_k + h * f(x_k, y_k) \text{ e } x_{k+1} = x_k + h \quad (8)$$

Com os valores de x_{k+1} e de \bar{y}_{k+1} , temos a reta $r_{k+1}(x)$ definida pelo ponto (x_{k+1}, \bar{y}_{k+1}) e coeficiente angular:

$$y'(x_{k+1}) = f(x_{k+1}, \bar{y}_{k+1}) \quad (9)$$

com expressão algébrica:

$$r_{k+1}(x) = y_{k+1} + (x - x_{k+1})f(x_{k+1}, \bar{y}_{k+1}) \quad (10)$$

Tomando um novo coeficiente angular como a média aritmética simples entre os valores dados pelas eqs. (6) e (9), definimos uma nova reta, de inclinação média entre $r_k(x)$ e $r_{k+1}(x)$ e que também passe pelo primeiro ponto (x_k, y_k) , conforme segue,

$$\bar{r}_k(x) = y_k + (x - x_k)(f(x_k, y_k) + f(x_{k+1}, \bar{y}_{k+1}))/2 \quad (11)$$

Assim, podemos recalcular o segundo ponto (x_{k+1}, y_{k+1}) utilizando esse coeficiente angular médio, através de

$$y_{k+1} = y_k + h(f(x_k, y_k) + f(x_{k+1}, \bar{y}_{k+1}))/2 \quad (12a)$$

ou usar apenas os valores do primeiro ponto (x_k, y_k) , resultando:

$$y_{k+1} = y_k + h(f(x_k, y_k) + f(x_k + h, y_k + h * f(x_k, y_k)))/2 \quad (12b)$$

As eqs. (12a) e (12b) são as duas expressões algébricas equivalentes do chamado método de Euler aperfeiçoado ou modificado. Esse método também é conhecido como método de *Runge-Kutta de 2ª ordem (RK₂)*. Tradicionalmente a eq. (12b) é expressa na literatura, para todo $k = 1, 2, \dots, n$, na seguinte forma:

$$y_{k+1} = y_k + h(K_1 + K_2) / 2 \quad (13a)$$

$$x_{k+1} = x_k + h \quad (13b)$$

em que os K_1 e K_2 são coeficientes angulares:

$$K_1 = f(x_k, y_k) \quad (13c)$$

$$K_2 = f(x_k + h, y_k + h * K_1) \quad (13d)$$



Podemos comprovar que sua fórmula equivale à expansão da $y(x)$ em série de Taylor até o termo com derivada de 2ª ordem ($O(h^2)$).

Observe que a parcela $(K_1 + K_2) / 2$ é um coeficiente angular médio da função solução $y(x)$, considerando apenas as informações disponíveis no ponto inicial de cada subintervalo (x_k, y_k) .

Exemplo 9.2: resolva, pelo método de Runge-Kutta de 2ª ordem, o PVI do Exemplo 9.1, isto é, $y' = x - y + 2$, com $y(x = 0) = 2$ no domínio $[0, 1]$.

Solução:

Temos que $f(x, y) = x - y + 2$ e $y(x = 0) = 2$.

Tomando $n = 8$ para o número de subdivisões e aplicando as eqs. (13), resulta:

$$h = (1 - 0) / 8 = 0.125$$

k	x_k	y_k	$K_1 = f(x_k, y_k)$	$K_2 = f(x_k + h, y_k + h * K_1)$	x_{k+1}	y_{k+1}
1	0	2	0.000000000	0.125000000	0.125	2.007812500
2	0.125	2.007812500	0.117187500	0.227539063	0.250	2.029357910
3	0.250	2.029357910	0.220642090	0.318061829	0.375	2.063026905
4	0.375	2.063026905	0.311973095	0.397976458	0.500	2.107398752
5	0.500	2.107398752	0.392601248	0.468526092	0.625	2.161219211
6	0.625	2.161219211	0.463780789	0.530808190	0.750	2.223381022
7	0.750	2.223381022	0.526618978	0.585791606	0.875	2.292906684
8	0.875	2.292906684	0.582093316	0.634331652	1.000	2.368933244
9	1.000	2.368933244				

Portanto, para $n = 8$, $y(x = 1) = y_9 \cong 2.368933244$.

Para calcular o erro estimado, usamos $n = 16$, $h = (1 - 0) / 16 = 0.0625$, reaplicaremos as eqs. (13) que fornece: $y(x = 1) = y_{17} \cong 2.368130539$.

Erro estimado $RK_2 = |y_9^{(n=8)} - y_{17}^{(n=16)}| = 8.0271e - 04 > 10^{-6}$

Observe que a exatidão almejada ainda não foi obtida com $n = 8$, apesar de a precisão ser bem melhor do que a do método de Euler simples. Por simulação em computador, teremos que aumentar n somente até $n = 2^7 = 128$ para chegarmos a um erro estimado $2.8260e - 06$, que é da ordem de 10^{-6} .

Na sequência, vamos apresentar um método ainda mais preciso do que o de Runge-Kutta de 2ª ordem.

9.2.2 Método de Runge-Kutta de 4ª ordem

O truncamento na série de Taylor representativa da solução $y(x)$, no método de Runge-Kutta de 4ª ordem, ocorre a partir do termo de 5ª ordem ($O(h^5)$) e conduz a múltiplas possibilidades de expressões, e a mais popular, para todo $k = 1, 2, \dots, n$, é a seguinte:

$$y_{k+1} = y_k + h(K_1 + 2K_2 + 2K_3 + K_4)/6 \quad (14a)$$

$$x_{k+1} = x_k + h \quad (14b)$$

em que

$$K_1 = f(x_k, y_k) \quad (14c)$$

$$K_2 = f(x_k + h/2, y_k + (h/2)K_1) \quad (14d)$$

$$K_3 = f(x_k + h/2, y_k + (h/2)K_2) \quad (14e)$$

$$K_4 = f(x_k + h, y_k + h * K_3) \quad (14f)$$

Observe que o termo $(K_1 + 2K_2 + 2K_3 + K_4) / 6$ é uma inclinação média ponderada da função solução $y(x)$, composta por uma inclinação no ponto inicial K_1 , duas no ponto intermediário, K_2 e K_3 , e uma no ponto final K_4 do subintervalo $[x_k, x_{k+1}]$.

Exemplo 9.3: resolva o PVI do Exemplo 9.1 usando agora o método de Runge-Kutta de 4ª ordem.

Solução:

Temos que $f(x, y) = x - y + 2$ com $y(x = 0) = 2$.

Tomando inicialmente $n = 8$, $h = (1 - 0) / 8 = 0.125$, para o número de subdivisões e aplicando as eqs. (14), resulta:

$$h = (1 - 0) / 8 = 0.125$$

k	x_k	y_k	K_1	K_2	K_3	K_4
1	0	2	0.000000000	0.062500000	0.058593750	0.117675781
2	0.125	2.007497152	0.117502848	0.172658920	0.169211666	0.221351390
3	0.250	2.028801223	0.221198777	0.269873854	0.266831661	0.312844820
4	0.375	2.062289861	0.312710139	0.355665755	0.352981029	0.393587511
5	0.500	2.106531345	0.393468655	0.431376864	0.429007601	0.464842705
6	0.625	2.160262184	0.464737816	0.498191703	0.496100835	0.527725212
7	0.750	2.222367353	0.527632647	0.557155607	0.555310422	0.583218845
8	0.875	2.291862843	0.583137157	0.609191084	0.607562714	0.632191817
9	1.000	2.367880272				

Portanto, para $n = 8$, $y(x = 1) \cong y_9 = 2.367880272$.

Para calcular o erro estimado, usamos $n = 16$, $h = (1 - 0) / 16 = 0.0625$, reaplicamos as eqs. (14) e obtemos:

$$y(x = 1) \cong y_{17} = 2.367879490.$$

$$\text{Erro estimado } RK_4 = |y_9^{(n=8)} - y_{17}^{(n=16)}| = 7.8147e - 07 < 10^{-6}.$$

Logo, temos o problema proposto no **Exemplo 9.1** resolvido com a precisão desejada, $O(10^{-6})$, em apenas $n = 8$ subdivisões, usando o método de Runge-Kutta de 4ª ordem.

Apenas para efeito didático, comparamos a solução numérica e a solução analítica exata desse PVI $y(x) = (x + 1) + e^{-x}$, obtida por meio de fator integrante, na seção **Complementando...** ao final deste capítulo.

O valor exato de $y(x)$ em $x = 1$ é

$$y(x=1) = (1+1) + e^{-1} = 2.3678794411714$$

Erros exatos dos valores aproximados por Runge-Kutta de 4ª ordem são:

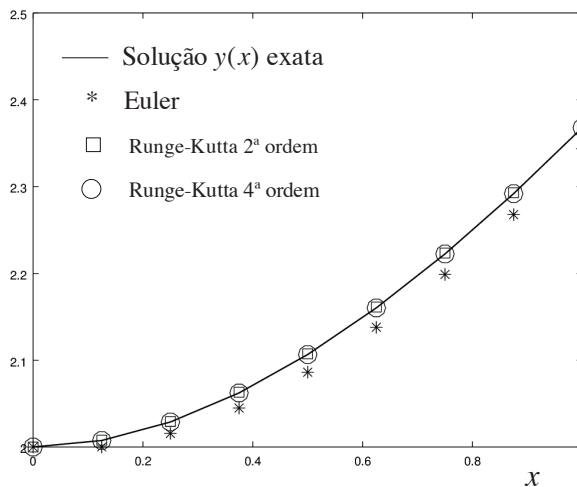
$$y(x=1) \cong y_9^{(n=8)} = 2.36788027192195 \Rightarrow \text{Erro exato de } y_9^{(n=8)} = 8.3075 \cdot 10^{-7}$$

$$y(x=1) \cong y_{17}^{(n=16)} = 2.36787949045257 \Rightarrow \text{Erro exato de } y_{17}^{(n=16)} = 4.9281 \cdot 10^{-8}$$

Note que, com $n = 16$, temos um resultado mais preciso, pois o erro exato com $n = 16$ é de ordem 10 vezes menor do que o erro com $n = 8$.

Agora, observe que, com o método de Runge-Kutta de 4ª ordem, e $n = 8$ subdivisões, temos em $y(x=1)$ o erro estimado de RK_4 é $7.8147 \cdot 10^{-7}$ (obtido comparativamente com o resultado de $n = 16$ subdivisões), enquanto o seu erro exato, comparando-o com a solução exata, é $8.3075 \cdot 10^{-7}$, ou seja, o erro estimado é da mesma ordem do erro exato.

Gráfico 9.3 – Resultados aproximados obtidos executando os algoritmos Euler Simplex, Runge-Kutta de 2ª e 4ª ordem para $n = 8$ para o Exemplo 9.1

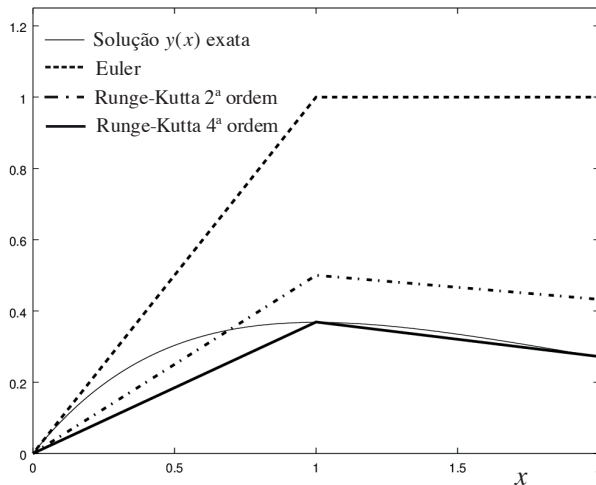


Fonte: Elaboração própria.

Perceba, no Gráfico 9.3, que, com essa resolução, os métodos de Runge-Kutta de 2ª e 4ª ordem geram praticamente os resultados exatos, tornando a visualização da sua diferença praticamente imperceptível.

Para visualizar as diferenças de cada um dos métodos apresentados, usamos o PVI exemplo $y' = e^{-x} - x * e^{-x}$, $y(x = 0) = 0$ com $x \in [0, 2]$ (solução exata $y(x) = x * e^{-x}$), e o passo $h = 1$ (passo grande para destacar o erro), conforme o Gráfico 9.4. Desse modo, podemos observar o comportamento decrescente nos erros com as soluções de Euler, de Runge-Kutta de 2ª ordem e de 4ª ordem, respectivamente.

Gráfico 9.4 – Comparativo entre soluções aproximadas e exata



Fonte: Elaboração própria.

Os algoritmos de Euler simples e Runge-Kutta de 2ª e 4ª ordem estão no **Caderno de Algoritmos**, em <http://sergiopeters.prof.ufsc.br/algoritmos-livro/>, no arquivo **Cap9exem9.1,9.2,9.3EulerRungeKutta.m** (com cálculos dos erros de forma exata e estimada, conforme os **Exemplos 9.1, 9.2 e 9.3**).

Sobre os métodos de Runge-Kutta de ordem p , concluímos que se caracterizam pelas três propriedades a seguir:

- a) são de passo simples, ou seja, a sua aplicação é autossuficiente, não necessitando de correções iterativas ou de outros métodos complementares, mas apenas dos valores no ponto inicial de cada subintervalo;
- b) não exigem o cálculo de qualquer derivada da função inclinação $f(x, y)$, mas precisam calcular $f(x, y)$ em vários pontos de cada subintervalo;
- c) a expansão da função solução $y(x)$ diretamente por série de Taylor em torno de (x_k, y_k) , que envolve as derivadas de ordem superior da função $f(x, y)$ (não apresentada neste livro), gera expressões de ordem equivalente às obtidas por Euler, Runge-Kutta de 2ª e de 4ª ordem.

Existem outras famílias de métodos numéricos de solução de PVI, além dos fundamentados em série de Taylor apresentados neste livro. Por exemplo, para o PVI $y' = f(x, y)$ com $y(x_1) = y_1$, temos:

$$y' = \frac{dy}{dx} = f(x, y) \Rightarrow dy = f(x, y)dx \Rightarrow \int_{x_k}^{x_{k+1}} dy = \int_{x_k}^{x_{k+1}} f(x, y)dx$$

Logo,

$$y_{k+1} = y_k + \int_{x_k}^{x_{k+1}} f(x, y)dx \quad (15)$$

Agora, efetuando numericamente a integral $I = \int_{x_k}^{x_{k+1}} f(x, y)dx$, por exemplo, pelo método dos trapézios, com um único intervalo, temos:

$$I = \int_{x_k}^{x_{k+1}} f(x, y) dx \cong \frac{h}{2} (f(x_k, y_k) + f(x_{k+1}, y_{k+1}))$$

que, substituída na eq. (15), resulta,

$$y_{k+1} = y_k + \frac{h}{2} (f(x_k, y_k) + f(x_{k+1}, y_{k+1})) \quad (16)$$

Assim, geramos um método de solução com uma característica completamente distinta dos já abordados, pois a incógnita y_{k+1} aparece em ambos os lados da eq. (16). Esse tipo de método é denominado de multipassos, pois ele não é autoinicializável, isto é, temos que atribuir um valor inicial para o y_{k+1} da direita da eq. (16), conforme fizemos com os métodos iterativos dos Capítulos 2 e 3. Na literatura, o leitor encontrará uma família desses métodos desenvolvidos por **Adams-Bashforth** que substituem algebricamente os valores de y_{k+1} da direita da eq. (16). Por exemplo, o primeiro método dessa família, proveniente da eq. (16), baseado na extrapolação linear de y_{k+1} a partir dos dois pontos anteriores y_{k-1} e y_k , torna-se

$$y_{k+1} = y_k + \frac{h}{2} (3 * f(x_k, y_k) - f(x_{k-1}, y_{k-1})) \quad \forall k = 2, 3, \dots, n \quad (17)$$

com erro local da ordem de $O(h^3)$. A eq. (17) calcula uma solução aproximada a partir do terceiro ponto y_3 usando os dois pontos anteriores: o valor inicial y_1 conhecido e necessitando de um método complementar, de outra família, para gerar o valor do segundo ponto y_2 .

Exemplo 9.4: resolva o mesmo PVI do **Exemplo 9.1** pelo método de **Adams-Bashforth**, eq.(17), com $n = 8$. Use o método de Runge-Kutta de 2ª ordem, eq. (13), para obter o segundo ponto y_2 .

Solução:

Para $f(x, y) = x - y + 2$, $y(x = 0) = 2$, temos:

k	x_k	y_k
1	0	2
2	0.125	2.00781250000000
3	0.250	2.02978515625000
4	0.375	2.06375122070312
5	0.500	2.10834693908691
6	0.625	2.16232883930206
7	0.750	2.22460136562586
8	0.875	2.29419666202739
9	1.000	2.37025987324887

$$y(x = 1) \cong y_9^{(n=8)} = 2.37025987324887$$

Erro estimado Adams-Bashforth = 0.00178262689949493

Erro exato Adams-Bashforth = 0.00238043207742722.

Observe que esses erros ficaram na mesma ordem de grandeza dos erros do método de Runge-Kutta de 2ª ordem. Porém, Adams-Bashforth com trapézios utiliza uma única nova chamada $f(x_k, y_k)$ no *ponto anterior* (que pode ser reaproveitada para o próximo). O Runge-Kutta de 2ª ordem utiliza duas chamadas na $f(x, y)$ para cada passo.



$f(x_{k-1}, y_{k-1})$ é sempre calculada e armazenada no passo anterior.

O Algoritmo de Adams-Bashforth com integração por trapézios está no **Caderno de Algoritmos** no arquivo **Cap9exem9.4Adams_BashforthTn.m**.

A seguir, vamos abordar a solução numérica de EDO do tipo PVI de ordem superior, por meio da sua transformação em um sistema de PVI de primeira ordem.

9.3 SOLUÇÃO NUMÉRICA DE SISTEMAS DE PVI DE 1ª ORDEM E DE PVI DE ORDEM SUPERIOR

Nesta seção, vamos estender as expressões dos métodos de solução numérica de um PVI de 1ª ordem para a solução de um PVI de ordem n por meio da solução de um sistema de n PVI de 1ª ordem. Sem perda de generalidade, vamos apresentar a extensão desses métodos apenas para o PVI de ordem $n = 2$.

Definição 3: um sistema de duas EDO de 1ª ordem com $x \in [a, b]$ é toda expressão do tipo:

$$\begin{cases} y_1' = f_1(x, y_1, y_2) \\ y_2' = f_2(x, y_1, y_2) \end{cases} \quad (18)$$

Na eq. (18), se as condições iniciais $y_1(x_1 = a) = u_1$ e $y_2(x_1 = a) = u_2$ forem valores conhecidos, teremos um sistema de dois PVI de 1ª ordem. Para resolver esse sistema, também dividimos o domínio $x \in [a, b]$ em n partes, de comprimento $h = (b - a) / n$, e adaptamos, por exemplo, o método de Euler simples, eq. (8), para todo $k = 1, 2, \dots, n$, resultando,

$$\begin{cases} y_{1,k+1} = y_{1,k} + f_1(x_k, y_{1,k}, y_{2,k}) \\ y_{2,k+1} = y_{2,k} + f_2(x_k, y_{1,k}, y_{2,k}) \\ x_{k+1} = x_k + h \end{cases} \quad (19)$$

Adaptando o método de Runge-Kutta de 2ª ordem (eq. 13) para resolver o sistema (18), resulta:

$$\begin{cases} y_{1,k+1} = y_{1,k} + 0.5h(K_1^{f_1} + K_2^{f_1}) \\ y_{2,k+1} = y_{2,k} + 0.5h(K_1^{f_2} + K_2^{f_2}) \\ x_{k+1} = x_k + h \end{cases} \quad (20a)$$



Os superíndices f_1 e f_2 indicam as funções representativas das EDO de y_1 e y_2 , respectivamente.

em que

$$\begin{aligned} K_1^{f_1} &= f_1(x_k, y_{1,k}, y_{2,k}) \\ K_1^{f_2} &= f_2(x_k, y_{1,k}, y_{2,k}) \\ K_2^{f_1} &= f_1(x_k + h, y_{1,k} + h * K_1^{f_1}, y_{2,k} + h * K_1^{f_2}) \\ K_2^{f_2} &= f_2(x_k + h, y_{1,k} + h * K_1^{f_1}, y_{2,k} + h * K_1^{f_2}) \end{aligned} \quad (20b)$$



Note que os valores de $K_1^{f_1}$ e $K_1^{f_2}$ devem ser calculados antes de calcularmos $K_2^{f_1}$ e $K_2^{f_2}$, pois estes últimos dependem de $K_1^{f_1}$ e $K_1^{f_2}$.

Então, para resolver um PVI de 2ª ordem, pode-se fazer uma mudança de variáveis, por exemplo:

$y'' = f(x, y, y')$ com $x \in [a, b]$, $y(x_1 = a) = u_1$ e $y'(x_1 = a) = u_2$ conhecidos, tomamos uma variável auxiliar $y_2 = y'$ e redefinimos a solução como $y_1 = y$ (para manter a notação de sistema) da seguinte forma:

$$\begin{cases} y_1 = y \\ y_2 = y' \end{cases} \quad (21)$$

Derivando ambas as funções das eqs. (21), o PVI de 2ª ordem torna-se o sistema:

$$\begin{cases} y_1' = y' = y_2 \\ y_2' = y'' = f(x, y, y') = f(x, y_1, y_2) \end{cases} \quad (22)$$

Com $y_1(x_1 = a) = u_1$ e $y_2(x_1 = a) = u_2$ conhecidos, podemos reescrever as eqs. (22) como:

$$\begin{cases} y_1' = f_1(x, y_1, y_2) = y_2 & \rightarrow y_1(x_1 = a) = u_1 \\ y_2' = f_2(x, y_1, y_2) = f(x, y_1, y_2) & \rightarrow y_2(x_1 = a) = u_2 \end{cases} \quad (23)$$

Temos justamente um sistema de duas EDO de 1ª ordem, como o das eqs. (18), que pode ser resolvido numericamente pelas eqs. (19) ou (20), ou por outro método adaptado a sistemas, como o Runge-Kutta de 4ª ordem.

Na sequência, vamos apresentar um exemplo de solução de PVI de 2ª ordem.

Exemplo 9.5: determine a solução do PVI, $y'' - 4y' + 4y = 8x^2 - 16x + 4$, com $x \in [0, 1]$, condições iniciais, $y(x = 0) = 0$ e $y'(x = 0) = 1$, e erro máximo menor do que 10^{-6} .

Solução:

Com os métodos apresentados, podemos resolver apenas uma EDO de 1ª ordem por vez; então podemos transformar uma EDO de 2ª ordem em um sistema de duas EDO de 1ª ordem. Para EDO de 2ª ordem, como neste exemplo, redefinimos $y(x)$ e criamos uma variável auxiliar para $y'(x)$, conforme segue:

$$\begin{cases} y_1(x) = y(x) \\ y_2(x) = y'(x) \end{cases}$$

Derivamos as duas variáveis auxiliares, $y_1(x)$ e $y_2(x)$, para gerar duas equações diferenciais:

$$y_1(x) = y(x) \Rightarrow y_1'(x) = y'(x) = y_2(x)$$

$$y_1'(x) = z_1(x, y_1, y_2) = y_2(x)$$

$$y_1(x=0) = y(x=0) = 0 \text{ (condição inicial)}$$

$$y_2(x) = y'(x) \Rightarrow y_2'(x) = y''(x) = 4y' - 4y + 8x^2 - 16x + 4 \text{ (EDO original)}$$

$$y_2'(x) = z_2(x, y_1, y_2) = 4y_2 - 4y_1 + 8x^2 - 16x + 4$$

$$y_2(x=0) = y'(x=0) = 1 \text{ (condição inicial)}$$

Resultando duas EDO de 1ª ordem, para $y_1(x)$ e $y_2(x)$, com duas condições iniciais:

$$\begin{cases} y_1'(x) = z_1(x, y_1, y_2) = y_2 & \Rightarrow y_1(x=0) = y(x=0) = 0 \\ y_2'(x) = z_2(x, y_1, y_2) = 4y_2 - 4y_1 + 8x^2 - 16x + 4 & \Rightarrow y_2(x=0) = y'(x=0) = 1 \end{cases}$$

Então, resolvemos as duas EDO simultaneamente, pois a incógnita $y_1(x)$ depende de $y_2(x)$, e vice-versa, conforme o arquivo **Cap9exem9.5sist2PVI.m** do **Caderno de Algoritmos**.

A seguir estão os resultados obtidos para $n = 43$, de forma que o erro máximo seja menor do que 10^{-6} .

k	t	y_1	y_2
1	0.00000	0.00000	1.0000
2	0.02326	0.02544	1.1894
3	0.04651	0.05537	1.3856
4	0.06977	0.08995	1.5892
5	0.09302	0.12935	1.8007

		⋮	
41	0.95349	8.23783	23.3858
42	0.97674	8.79728	24.7387
43	1.00000	9.38906	26.1672

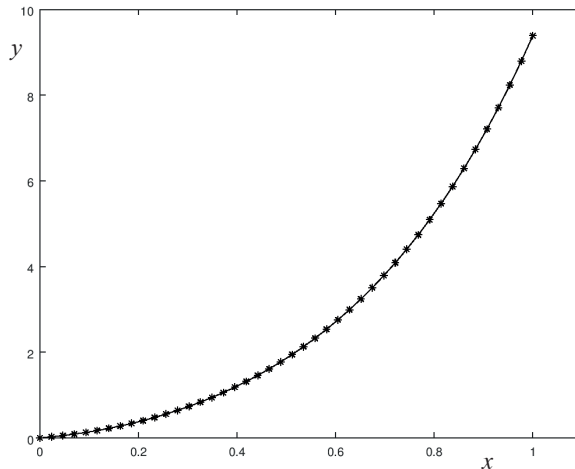
Erro máximo de $y = y_1$:

Erro exato máximo em todo $x \in [0, 1]$ é $9.28874230865517e - 07$

Erro exato máximo em $x = 1$ é $9.28874230865517e - 07$

Erro estimado máximo em $x = 1$ é $8.69081802079563e - 07$.

Gráfico 9.5 – Gráfico da solução discreta $y_1(x)$ obtida por Runge-Kutta de 4ª ordem marcada com * e a solução exata $y(x)$, em linha contínua



Fonte: Elaboração própria.

Acompanhe agora uma introdução à solução numérica de EDO do tipo Problema com Valor no Contorno (PVC).

9.4 SOLUÇÃO NUMÉRICA DE EDO COM PROBLEMA DE VALOR NO CONTORNO (PVC)

Para uma EDO de ordem $n > 1$, se o valor da sua solução $y(x)$, ou de alguma(s) de suas derivadas até ordem $n - 1$, for conhecido em mais de um ponto do domínio $[a, b]$, normalmente nos extremos deste intervalo, ela será denominada de **Problema de Valor no Contorno (PVC)** e sua solução não poderá ser obtida diretamente por meio dos métodos de resolução de PVI. Por exemplo, para uma EDO de ordem $n = 2$, $y'' = f(x, y, y')$ com domínio $x \in [a, b]$, a especificação dos valores conhecidos nos extremos pode ser sintetizada pelo sistema:

$$\begin{cases} \alpha_1 y(a) + \beta_1 y'(a) = v_1 \\ \alpha_2 y(b) + \beta_2 y'(b) = v_2 \end{cases}$$

em que α_i , β_i e v_i , com $i = 2$, são constantes Reais conhecidas, bem como os α_i e β_i não podem ser simultaneamente nulos.

Os métodos de solução de PVC podem ser agrupados na categoria dos fundamentados nas diferenças finitas, nos quais as derivadas envolvidas na EDO são aproximadas numericamente por operadores de diferenças, gerando sistemas de equações algébricas cuja solução fornece aproximações discretas da solução do PVC, conforme (FAIRES; BURDEN, 2011). Uma segunda maneira de resolver um PVC consiste em determinar a condição inicial desconhecida por meio de uma função entre esta condição inicial desconhecida e a condição de contorno conhecida, conforme desenvolveremos a seguir.

9.4.1 Métodos de determinação de uma condição inicial em função de uma condição de contorno

Para o PVC $y'' = f(x, y, y')$, $x \in [a, b]$ com condições de contorno $y(a) = v_1$ e $y(b) = D$ conhecidos, a ideia é tentar **adivinhar** um valor inicial C para o $y'(a)$ desconhecido do PVI a seguir,

$$y'' = f(x, y, y'), \quad x \in [a, b] \text{ com}$$

$$y(a) = y_1(a) = v_1 \text{ e}$$

$$y'(a) = y_2(a) = C = ?$$

de modo que, depois de resolvido esse PVI, pelos métodos abordados na seção anterior, a partir das condições iniciais (CI) atribuídas, C , possamos atingir a condição de contorno (CC), $y(b) = D$, conhecida. É o mesmo que determinar C da equação $\text{erro}(C) = Z(C) - D = 0$, em que:

- $y_1(b) = y(b) = Z(C)$ é o alvo atingido pela solução numérica da EDO, obtida a partir de $y(a) = y_1(a) = v_1$ e $y_2(a) = y'(a) = C$;
- $y_1(b) = y(b) = D$ é o alvo (condição de contorno); e
- $Z(C)$ representa o cálculo de $y_1(x = b)$, correspondente ao **alvo atingido**, em função de uma condição inicial atribuída (mira), $y_2(x = a) = C$, por meio da solução numérica do respectivo sistema de PVI.

No **Exemplo 9.6** a seguir, vamos obter a solução de um PVC de 2ª ordem aplicando o chamado *shooting method*, com o qual calculamos a condição inicial (mira) $y_2(x = 0) = C$ por meio de **correções** baseadas no valor da condição de contorno conhecida (alvo) $y_1(x = 1) = D$. Por isso, esse método de tentativas é chamado de *shooting* (tiro). Alternativamente, vamos determinar diretamente a raiz C da equação do erro entre o alvo atingido e o alvo correto, $\text{erro}(C) = Z(C) - D = 0$, pelo método de Newton com derivadas numéricas (secante).

Exemplo 9.6: determine a solução $y(x)$ da EDO de 2ª ordem $y'' - 4y' + 4y = 8x^2 - 16x + 4$, com as duas condições de contorno, $y(x=0) = 0$ e $y(x=1) = 9.389056098930666$, com erro máximo da ordem 10^{-6} . Use o *shooting method* e o método de Newton numérico para determinar a condição inicial desconhecida $y'(x=0) = C$.

Solução:

Para essa EDO de 2ª ordem, definimos as variáveis auxiliares, conforme segue:

$$y_1(x) = y(x) \Rightarrow y_1'(x) = y'(x) = y_2(x) \text{ (própria solução } y(x))$$

$$y_1'(x) = z_1(x, y_1, y_2) = y_2(x)$$

$$y_1(x=0) = y(x=0) = 0 \text{ (condição inicial conhecida)}$$

$$y_2(x) = y'(x) \Rightarrow y_2'(x) = y''(x) = 4y_2' - 4y_1 + 8x^2 - 16x + 4 \text{ (} y'' \text{ é a própria EDO)}$$

$$y_2'(x) = z_2(x, y_1, y_2) = 4y_2 - 4y_1 + 8x^2 - 16x + 4$$

$$y_2(x=0) = y'(x=0) = C$$

Em que C é uma condição inicial de valor desconhecido.

Resultam então duas EDO de 1ª ordem que precisam de duas condições iniciais:

$$\begin{cases} y_1'(x) = z_1(x, y_1, y_2) = y_2 & \Rightarrow y_1(x=0) = y(x=0) = 0 \\ y_2'(x) = z_2(x, y_1, y_2) = 4y_2 - 4y_1 + 8x^2 - 16x + 4 & \Rightarrow y_2(x=0) = y'(x=0) = C \end{cases}$$

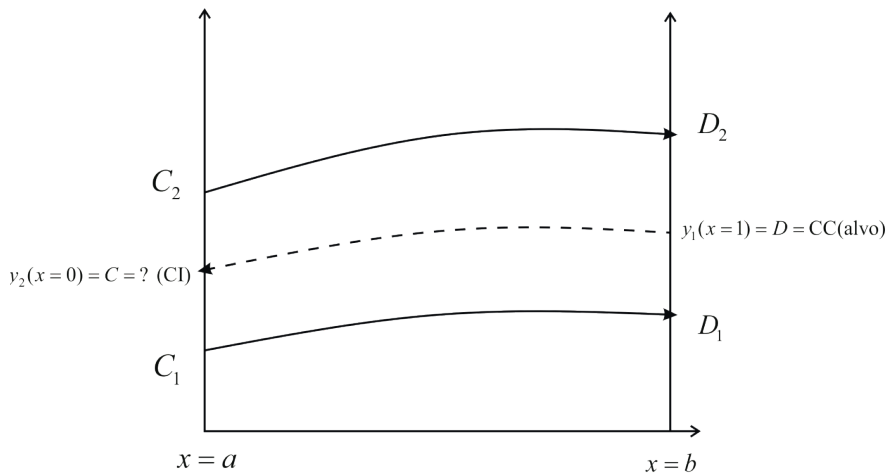
Agora precisamos determinar primeiro a condição inicial para $y_2(x=0)$ desconhecida, denominada de C , e já conhecemos o valor da condição de contorno no outro extremo do domínio de cálculo, $y_1(x=1) = y(x=1) = 9.389056098930666$, que será denominada de D .

Determinando C pelo *shooting method*

Vamos atribuir dois valores estimados para C (miras), resolver duas vezes as EDO e obter dois valores finais para $y_1(x = 1)$ em $x = 1$ (alvos atingidos) que, inicialmente, estarão errados, pois o valor correto em $x = 1$ é conhecido, $y_1(x = 1) = D = 9.389056098930666$ (alvo).

Então, nesse método, consideramos estimativas da condição inicial ausente, como se fossem miras de tiro; atribuímos dois valores iniciais de miras, C_1 e C_2 ; resolvemos as equações entre o ponto inicial a e o ponto final b ; e determinamos dois resultados para os alvos $y_1(x = 1)$ atingidos no outro extremo b (contorno), denominados como D_1 e D_2 , cujo valor correto é o alvo com o valor D da condição de contorno conhecida em b , conforme o Gráfico 9.6.

Gráfico 9.6 – *Shooting method* para determinar uma condição inicial (CI) C a partir de uma condição de contorno (CC) D conhecida



Fonte: Elaboração própria.

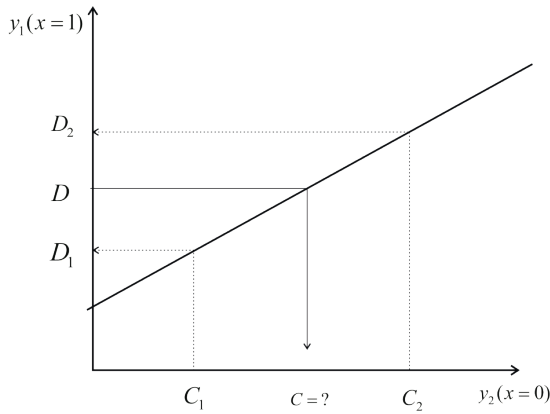
Então, relacionamos $y_2(x = 0) = C$ com $y_1(x = 1) = D$, supondo uma relação de **proporcionalidade linear** entre C e D , conforme mostrado no Gráfico 9.7, que é uma relação válida para valores de C próximos da solução a seguir.

Para $y_2(x = 0) = y'(x = 0) = C_1$, obtemos $y_1(x = 1) = y(x = 1) = D_1$.

Para $y_2(x = 0) = y'(x = 0) = C_2$, obtemos $y_1(x = 1) = y(x = 1) = D_2$.

Para $y_2(x = 0) = y'(x = 0) = C = ?$, obter $y_1(x = 1) = y(x = 1) = D = 9.389056098930666$.

Gráfico 9.7 – Relação linear entre a CC conhecida $y_1(x = 1) = D$ (alvo) e a condição inicial $y_2(x = 0) = C$ desconhecida (mira)



Fonte: Elaboração própria.

Logo, estabelecendo uma relação de proporcionalidade entre as condições iniciais C_1 e C_2 (miras propostas) com as condições de contorno D_1 e D_2 (alvos atingidos), calculadas na outra extremidade, podemos estabelecer uma relação linear entre C e D , conforme segue,

$$\frac{C - C_1}{C_2 - C_1} = \frac{D - D_1}{D_2 - D_1}$$

$$C = C_1 + (C_2 - C_1) \left(\frac{D - D_1}{D_2 - D_1} \right) \quad (24)$$

Uma vez obtido o valor de C , via eq. (24), teremos um resultado melhor do que os dois valores iniciais C_1 e C_2 . Então escolhemos dois desses três valores e refazemos o cálculo de C iterativamente. Por exemplo, vamos tomar C e C_2 como os dois melhores valores, descartando C_1 , conforme a seguinte seleção:

$$C_1 = C_2 \text{ (descarta } C_1, \text{ substitui por } C_2)$$

$$C_2 = C \text{ (atualiza } C_2, \text{ substitui por } C)$$

Assim, em um novo passo:

- redefinimos $D_1 = D_2$, pois D_1 é o valor de D_2 do passo anterior;
- recalculamos D_2 , com o novo valor de C_2 ; e
- refazemos o cálculo de C , via eq. (24), até que algum critério de parada seja satisfeito.

Podemos usar $|C_2 - C_1| < 10^{-6}$, $|D_2 - D| < 10^{-6}$ ou $|D_2 - D_1| < 10^{-6}$ como possíveis critérios de parada deste exemplo.

Determinando C pelo método de Newton

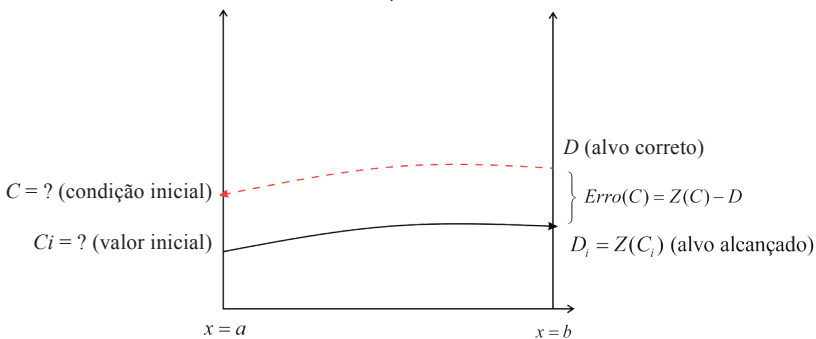
Vamos determinar C diretamente por meio da equação do erro entre o alvo atingido $Z(C)$ e o alvo D , $\text{erro}(C) = Z(C) - D = 0$, em função da condição C desconhecida, aplicando o método de Newton com derivadas numéricas (método da secante).



Observe que essa função desse erro é calculada em relação à condição de contorno correta, no caso $y_1(b) = y(b) = D$ (conhecida).

Observe, no Gráfico 9.8, o erro entre o alvo atingido $D_i = Z(C_i)$, via resolução da EDO, e o alvo D conhecido. A mira correta C será obtida quando o $\text{erro}(C) = Z(C) - D = 0$.

Gráfico 9.8 – Erro entre o alvo atingido D_i e o alvo D



Fonte: Elaboração própria.

Observe que, para cada tentativa i , de valor inicial de C_i (mira), fica definida a condição inicial $y_2(x = 0) = C_i$ do sistema de PVI. Então resolvemos as EDO pelos métodos disponíveis para PVI e obtemos o valor final para $y_1(x = 1) = D_i$ (alvo atingido), que seria equivalente a uma função $Z(C_i) = D_i$ (condição de contorno calculada). Inicialmente, o valor D_i está errado, mas o valor correto em $x = 1$ está disponível, dado por $y_1(x = 1) = D = 9.389056098930666$ (alvo). Logo, o valor correto de C deve satisfazer a equação $erro(C) = Z(C) - D = 0$, que pode ser resolvida pelo método de Newton com derivadas numéricas.

Esse mesmo método de Newton pode ser estendido para problemas de valor no contorno com duas ou mais condições iniciais desconhecidas. Para tanto basta gerar e resolver duas ou mais equações de erro entre os alvos atingidos (obtidos pela própria solução numérica do sistema de PVI) e os alvos conhecidos, conforme o **Exercício 9.6**, disponível no caderno de Exercícios.

Depois de determinar o valor da condição inicial desconhecida, no caso $C = 1$, conforme algoritmo apresentado, devemos determinar a solução da EDO e estimar o erro máximo em função do número n de subintervalos estabelecido.

Mas como vamos medir o erro máximo da solução $y(x)$ se nos dois contornos (extremos) os erros são nulos? Onde estará o erro máximo?

Possivelmente o maior erro estará no meio do intervalo $x \in [0, 1]$. Podemos estimar o erro máximo comparando a solução aproximada com n subdivisões com outra solução aproximada com $2n$ subdivisões no ponto médio, por exemplo, ou calculá-lo nos vários pontos discretos e tomar o seu valor máximo.

Neste **Exemplo 9.6**, podemos validar essa estimativa de erro comparando-o com o erro exato, conforme algoritmo correspondente, e determinar o n mínimo por sucessivas tentativas.

Resultados:

Com $n = 18$ obtemos:

$C = 1.00000377206141$ pelo *shooting method*, obtido em 2 iterações.

$C = 1.00000377206141$ pelo método de Newton, obtido também em 2 iterações.

Erro exato máximo de y_1 é $2.81176553951212e - 06$ e ocorre em $x = 0.66666667$.

Erro estimado máximo no meio do domínio, $x = 0.5$, é $2.18895585768486e - 06$.

Erro estimado máximo, é $2.62705417908293e - 06$ (nesse caso, recalculamos C e a solução aproximada y_1 com $2n$ subdivisões e os usamos como valores exatos estimados). Neste exemplo, todas as formas de erro são da mesma ordem.

Confira no **Caderno de Algoritmos** os arquivos **Cap9exep9.6ShootingNewton2.m** e **Cap9exem9.6NR2.m** com os algoritmos que resolvem o **Exemplo 9.6**, determinando C via *shooting method* e via método de Newton, respectivamente.

9.5 CONCLUSÕES

Neste último capítulo da obra, apresentamos uma abordagem superficial da solução numérica de Equações Diferenciais Ordinárias (EDO), visando suprir os tópicos constantes nos programas de disciplinas dos cursos de graduação e resolver os modelos matemáticos básicos que utilizam essa ferramenta. Como nem todas as EDO têm solução analítica conhecida, recorreremos aos métodos numéricos para a solução delas. Os métodos de passo simples da família Runge-Kutta são amplamente usados para resolver EDO, do tipo Problema de Valor Inicial (PVI), por fornecerem resultados precisos e com baixo esforço computacional, e também foram estendidos para resolver Problemas de Valor no Contorno (PVC).

O próximo passo nessa área é a solução numérica de Equações Diferenciais Parciais (EDP) por meio de métodos clássicos como diferenças finitas, elementos finitos, e volumes finitos, dentre outros. Para o método dos volumes finitos, indicamos a obra *Transferência de Calor e Mecânica dos Fluidos Computacional*, de Maliska (2004).

COMPLEMENTANDO...

Solução exata de $y' = x - y + 2$ para os **Exemplos 9.1, 9.2, 9.3:**

Reescrevendo:

$y' + y = x + 2 \Rightarrow$ multiplicando um fator integrante $u(x)$ desconhecido em ambos os lados, temos:

$$\left(\frac{dy}{dx} + y\right)u(x) = (x + 2)u(x) \Rightarrow u(x)\frac{dy}{dx} + u(x)y(x) = (x + 2)u(x)$$

Se, no lado esquerdo, tivéssemos apenas a derivada do produto $u(x)y(x)$, isso poderia facilitar a integração. Ou seja,

$$\text{Se } \frac{d(u(x)y(x))}{dx} = u(x)\frac{dy(x)}{dx} + \frac{du(x)}{dx}y(x)$$

$$\text{For igual a } u(x)\frac{dy(x)}{dx} + \mathbf{u(x)}y(x)$$

Comparando termo a termo, o fator integrante $u(x)$, caso exista, deverá satisfazer:

$$\frac{du(x)}{dx} = u(x)$$

Resolvendo essa equação de variáveis separáveis, temos:

$$\frac{du(x)}{u(x)} = dx \Rightarrow \int \frac{du(x)}{u(x)} = \int dx \Rightarrow \ln(u(x)) = x + C$$

Agora, escolhendo um valor adequado $C = 0$, temos:

$$u(x) = e^x$$

Substituindo o nosso fator integrante $u(x)$, temos:

$$\left(\frac{dy}{dx} + y\right)u(x) = \left(\frac{dy}{dx} + y\right)e^x = e^x * \frac{dy}{dx} + e^x * y = \frac{d(e^x * y(x))}{dx}$$

assim podemos trocar o termo

$$\left(\frac{dy}{dx} + y\right)e^x \text{ por } \frac{d(e^x * y(x))}{dx}$$

e a nossa EDO fica:

$$\left(\frac{dy}{dx} + y\right)e^x = \frac{d(e^x * y(x))}{dx} = (x + 2)e^x \Rightarrow$$

$$\frac{d(e^x * y(x))}{dx} = (x + 2)e^x$$

Integrando

$$\int d(e^x * y(x)) = \int (x + 2)e^x dx \Rightarrow e^x * y(x) = \int x * e^x dx + \int 2 * e^x dx$$

Integrando $\int x(e^x * dx) = x * e^x - \int e^x * 1 * dx$ por partes $\int u dv = u * v - \int v * du$,

obtemos $\int x * e^x * dx = x * e^x - e^x \Rightarrow \int x * e^x dx = (x - 1)e^x + c$

Integrando $\int 2 * e^x * dx = 2 * e^x$ diretamente, temos

$$e^x * y(x) = (x - 1)e^x + 2 * e^x + c$$

$$e^x * y(x) = (x + 1)e^x + c$$

$$y(x) = (x + 1) + c * e^{-x}$$

Aplicando a condição inicial $y(0) = 2$, temos $y(x = 0) = (0 + 1) + c * e^{-0} = 2$
 $\Rightarrow c = 1$. Logo, a solução é dada por:

$$y(x) = (x + 1) + e^{-x}$$

REFERÊNCIAS

- ABRAMOWITZ, M.; STEGUN, I. A. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. New York: Dover Books on Mathematics, 1961.
- BARBETTA, P. A.; REIS, M. M.; BORNIA, A. C. *Estatística para cursos de Engenharia e Informática*. 3. ed. São Paulo: Atlas, 2010.
- BARROSO, L. et al. *Cálculo numérico com aplicações*. 2. ed. Belo Horizonte: Harbra, 1987.
- BURDEN, R. L.; FAIRES, J. D. *Análise numérica*. São Paulo: Pioneira Thomson Learning, 2003.
- BYRNE, M. *A New Number Format for Computers Could Nuke Approximation Errors for Good*. Publicado em: 24 abr. 2016. Disponível em: <<http://motherboard.vice.com/read/a-new-number-format-for-computers-could-nuke-approximation-errors-for-good>>. Acesso em: 14 nov. 2016.
- CARNAHAN, B.; LUTHER, H. A.; WILKES, J. O. *Applied Numerical Methods*. New York: Wiley, 1990.
- CHATTERJEE, S.; SIMONOFF, J. S. *Handbook of Regression Analysis*. New York: Wiley, 2012.
- CHENEY, W.; KINCAID, D. *Numerical Mathematics and Computing*. 5. ed. Belmont: Brooks/Cole, Cengage Learning, 2004.
- CHIANG, A. C. *Fundamental Methods in Mathematical Economics*. 3. ed. New York: McGraw-Hill Inc., 1984.
- CURVA de Bézier. Disponível em: <https://pt.wikipedia.org/wiki/Curva_de_B%C3%A9zier>. Acesso em: 3 nov. 2016.
- FENÓMEMO de Runge. Disponível em: <https://pt.wikipedia.org/wiki/Fen%C3%B3meno_de_Runge#/media/File:Rungesphenomenon.png>. Acesso em: 3 nov. 2016.>.
- FEOFIOFF, P. *Projeto de algoritmos (em C) 2008-2009*. Disponível em: <<https://www.ime.usp.br/~pf/algoritmos/aulas/binst.html>>. Acesso em: 12 out. 2017.
- GAELZER, Rudi. *Integração numérica*. Disponível em: <<http://minerva.ufpel.edu.br/~rudi/grad/ModComp/MetNum/html/Apostilach3.html>>. Acesso em: 24 set. 2013.
- GALÁNTAI, A.; HEGEDUS, J. C. A study of accelerated Newton methods for multiple polynomial roots, *Numer Algor*, [s.l.], v. 54, p. 219-243, 2010.

- GUSTAFSON, J. L. *The End of Error: unun Computing*. Berkeley: CRC Press, 2015.
- HOLMES, D. G.; CONNELL, S. D., Solution of the 2d Navier-Stokes equations on unstructured adaptive grids. PROCEEDINGS OF THE AIAA 9th COMPUTATIONAL FLUID DYNAMICS CONFERENCE, 1989. p. 25-39.
- INSTITUTO ANTONIO HOUAISS. *Dicionário Eletrônico da Língua Portuguesa Houaiss*. Rio de Janeiro: Objetiva, 2009. 1 CD-ROM.
- INSTITUTO DE COMPUTAÇÃO UFF. *Spline física original e pesos (ducks)*. Niterói: IC/UFF, [2016]. 1 imagem. Disponível em: <<http://www2.ic.uff.br/~aconci/Image153.gif>>. Acesso em: 1 nov. 2016.
- MALISKA, C. R. *Transferência de calor e mecânica dos fluidos computacional*. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos – LTC, 2004.
- MATEMATIQUÊS. *Figura 1.2: o conjunto dos números reais IR*. 1 diagrama. [2003-2010]. Disponível em: <<http://www.matematiques.com.br/conteudo.php?id=200>>. Acesso em: 18 jul. 2016.
- PATTERSON, D. A.; HENNESSY, J. L. *Computer Organization and Design: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)*. 5. ed. San Francisco: Elsevier, 2007.
- POLINÔMIOS de Legendre. 28 mar. 2017. Disponível em: <https://pt.wikipedia.org/wiki/Polin%C3%B4mios_de_Legendre>. Acesso em: 25 jul. 2017.
- POLINÔMIOS de Tchebychev. 2016. Disponível em: <https://pt.wikipedia.org/wiki/Polin%C3%B4mios_de_Tchebychev>. Acesso em: 9 nov. 2016.
- PORTAL ESCOLAR. *Algarismos*. Publicado em: 28 mar. 2011. Disponível em: <<http://www.portalescolar.net/2011/03/algarismos-abscissa-altura-aritmetica.html>>. Acesso em: 30 jun. 2016.
- RALL, L. Convergence of the Newton process to multiple solutions. *Numer. Math.*, [s.l.], v. 9, p. 23-37, 1966.
- RICE, J. R. *Numerical Methods, Software, and Analysis*. Tokyo: McGraw-Hill, 1983.
- RUGGIERO, M. A. G.; LOPES, V. L. R. *Cálculo numérico: aspectos teóricos e computacionais*. 2. ed. São Paulo: Pearson Education, 1997.
- SCHRÖDER, E. Über unendlich viele Algorithmen zur Auflösung der Gleichungen. *Math. Ann.*, [s.l.], v. 2, p. 317-365, 1870. Tradução em inglês de G. W. Stewart. On Infinitely Many Algorithms for Solving Equations. *Technical Report TR-92-121*, University of Maryland, Department of Computer Science, 1992.
- SERRE, D. *Matrices: theory and applications*. New York: Springer Verlag, 2002.
- SÓ MATEMÁTICA. *Algarismo*. [1998-2016]. Disponível em: <<http://www.somatematica.com.br/biograf/khwarizmi.php>>. Acesso em: 30 jun. 2016.
- VANDERGRAFT, J. S. *Introduction to Numerical Computations*. 2. ed. Cambridge: Academic Press, 1983.

COLEÇÃO DIDÁTICA

- A interpretação de imagens aéreas
Algoritmos numéricos: sequenciais e paralelos
Análise sensorial de alimentos
Anatomia sistêmica: uma abordagem direta para o estudante
Anomalias laríngeas congênitas
Assistência social: do discurso do Estado à prática do Serviço Social
AutoCAD 2000: guia prático para desenhos em 2D
AutoCAD 2004: guia prático para desenhos em 2D
AutoCAD R14: guia prático para desenhos em 2D
AutoCAD R14: guia prático para desenhos em 3D
Avaliação nutricional de coletividades
Cálculo 1
Cálculo A
Cálculo C
Cálculo de indutância e de força em circuitos elétricos
Cálculo e Álgebra Linear com Derive
Câncer: o que você precisa saber
Cartografia: representação, comunicação e visualização de dados espaciais
Centro cirúrgico: aspectos fundamentais para Enfermagem
Classificação Decimal Universal – CDU
Construindo em alvenaria estrutural
Combustão aplicada
Desenho geométrico
Desenho técnico mecânico
Diagnóstico do meio físico de bacias hidrográficas
Elementos básicos de fotogrametria e sua utilização prática
Eletromagnetismo e cálculo de campos
Eletromagnetismo para Engenharia: estática e quase estática
Eletrônica básica: um enfoque voltado à Informática
Engenharia de protocolos com LOTOS/ISO
Estatística aplicada às Ciências Sociais
Estatística para as Ciências Agrárias e Biológicas
Experimentação na educação em Química: fundamentos, propostas e reflexões
Ferramentas de corte I
Ferramentas de corte II
Filtros seletores de sinais
Fundamentos da análise de sistemas dinâmicos
Fundamentos da Cartografia
Fundamentos de Aritmética
Fundamentos de sistemas hidráulicos
Geração de vapor
Gramática básica do Latim
Identificação de sistemas dinâmicos lineares
Influência açoriana no Português do Brasil
Inteligência Artificial: ferramentas e teorias
Introdução à Engenharia: conceitos, ferramentas e comportamentos
Introdução à Física Nuclear e de Partículas Elementares
Introdução à Matemática
Introdução à Química Inorgânica Experimental
Introdução à Teoria dos Grafos
Introdução à Topologia Geral
Introdução ao Projeto Geométrico de Rodovias
Introdução ao Laboratório de Física
Latim para o português: gramática, língua e literatura
Le Français parlé, pratique de la prononciation du Français
Macroescultura dental
Manual básico de Desenho Técnico
Maple V
Matemática: 100 exercícios de grupos
Matemática Financeira através da HP-12C
Matrizes e sistemas de equações lineares
Microbiologia: manual de aulas práticas
Monitoramento global integrado de propriedades rurais
Natação: ensine a nadar
Neuroanatomia: atlas descritivo do sistema nervoso central
Noções básicas de Geometria Descritiva
O papel da escola na construção de uma sociedade democrática
Óleos e gorduras vegetais: processamento e análise
Princípios de combustão aplicada
Promenades: textes et exercices pour la classe de Français
Propriedades químicas e tecnológicas do amido de mandioca e do polvilho azedo
Química básica: teoria e experimentos
Redação
Redação oficial: o texto técnico/científico e o texto literário
Redes de Petri
Sociologia da educação: currículo e saberes escolares
Solidworks: modelagem 3D
Taguchi e a melhoria da qualidade: uma releitura crítica
Teaching in a clever way: tarefas comunicativas para professores de Língua Inglesa do 1º grau
Tecnologia de grupo e organização da manufatura
Teoria fundamental do motor de indução
Topografia contemporânea: planimetria
Transmissão de energia elétrica: aspectos fundamentais
Unidades de informação: conceitos e competências
Ventilação industrial

Este livro foi editorado com as fontes Minion Pro, Din e Roboto.
Publicado *on-line* em: <editora.ufsc.br/estante-aberta>.

A **Coleção Didática** da Editora da UFSC é um instrumento de valorização e aperfeiçoamento do ensino de graduação. O conjunto de títulos, em constante expansão e revisão, abrange um amplo espectro de áreas de conhecimento. Explicações, figuras, exercícios e outras estratégias pedagógicas cuidadosamente concebidas viabilizam a compreensão de conceitos e de conteúdos programáticos fundamentais. Um time de autores de grande prestígio e com larga experiência em sala de aula garante a qualidade e a atualidade das obras.



9 788532 808387