CADERNO DE EXERCÍCIOS E RESPOSTAS

LISTA DE EXERCÍCIOS:

Exercícios Capítulo 1

1.1) Efetue as seguintes conversões de base:

a) $(10.1011)_2 = (?)_{10}$

b) $(10.57)_{10} = (?)_2$

1.2) Converta os números da base fatorial para a base decimal, conforme os exemplos (a) e (d):

a) $(3021)_{F!} = 3*4! + 0*3! + 2*2! + 1*1! = (77)_{10}$

b) $(4321)_{F!} = (?)_{10}$

c) $(10000)_{F!} = (?)_{10}$

d) $(0.02)_{F!} = 0*1! + 0/2! + 2/3! = (2/6)_{10} = (1/3)_{10} = (0.33333333...)_{10}$

e) $(0.113)_{F!} = (?)_{10}$

f) $(321.123)_{F!} = (?)_{10}$

Observe que, nos exercícios (d), (e) e (f), temos representações exatas de números racionais que, na base decimal, são dízimas periódicas.

Dica: existe uma base alternativa em que todo número racional tem representação finita; de acordo com o matemático George Cantor, trata-se da base fatorial. Conceitualmente, a base fatorial $F!$ é semelhante à decimal, com a diferença de que, em um número $X_{F!} = (a_n a_{n-1} \ldots a_1 \,.\, a_{-1} a_{-2} \ldots a_{-m})_{F!}$, cada a_i somente pode assumir um valor do intervalo $0 < a_i < |i|$, em que

$(a_n\, a_{n-1} \ldots a_1)_{F!} = \left(\sum_{i=n}^{1} a_i\ i! \right)$ é a parte inteira e $(0.\, a_{-1}\, a_{-2} \ldots a_{-m})_{F!} = \left(\sum_{i=1}^{m} a_{-i} / (i+1)! \right)_{10}$ é a parte fracionária.

Observe que $(4321)_{F!}$ terá como seu sucessor $(10000)_{F!}$

$\downarrow$ $\downarrow$

$(119)_{10}$ $(120)_{10}$

1.3) Converta os números para as bases na ordem determinada e indique onde poderá haver perda de dígitos significativos:

a) $(10111.1101)_2 = (?)_{16} = (?)_{10}$

b) $(\mathrm{BD.0E})_{16} = (?)_2 = (?)_{10}$

c) $(41.2)_{10} = (?)_2 = (?)_{16}$ (defina o número de *bits* representáveis)

1.4) Na representação $F(2,\ 3, -3, +3)$, com três *bits* significativos totais e normalização com $d_1 \neq 0 = 1$ alocado depois do ponto, ou não polarizada, calcule:

a) O número de mantissas representáveis.
b) O número de expoentes representáveis.
c) O número de elementos representáveis.
d) Defina as regiões de *underflow* e *overflow*.
e) Estime a precisão decimal equivalente.
f) Transforme o padrão $F(2,\ 3, -3, +3)$ do exercício 1.4 em padrão **IEEE 754** aproveitando ao máximo os 7 *bits* disponíveis.

1.5) Na representação $F(2,3,0,7)$ padrão **IEEE 754** e normalização com $d_1 \neq 0 = 1$ representado implicitamente antes do ponto, mais três *bits* significativos depois do ponto (quatro *bits* significativos totais) e polarização $p = +3$:

Se $0 < e < 7$, então $v = (-1)^s\, 2^{e-3} (1.f)_2$;

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

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

Se $e = 7$, então v pertence à região de *overflow*.

Calcule:

a) O número de mantissas representáveis.
b) O número de expoentes representáveis.
c) O número de elementos representáveis.
d) Defina as regiões de *underflow* e *overflow*.
e) Estime a precisão decimal equivalente.
f) Represente o “zero”.

1.6) Avalie as regiões de *underflow* e *overflow* para a variável `double` de 64 $bits$, que tem 52 $bits$ na parte fracionária, e teste os seus limites em alguma linguagem de programação (C, Java, Octave,...).

1.7) Avalie a precisão decimal equivalente da variável de 64 $bits$ por meio das três formas apresentadas na seção **Complementando...** ao final do Capítulo 1.

1.8) Execute o algoritmo, a seguir, com cinco variáveis de tipos Reais assumindo os seguintes valores:

$h = 1/2$

$x = 2/3 - h$

$y = 3/5 - h$

$e = (x + x + x) - h$

$f = (y + y + y + y + y) - h$

$g = e / f$

Imprima os resultados com 25 dígitos.

Observação: use variáveis Reais de 32 e/ou 64 $bits$ e explique a causa dos resultados obtidos para $g = e / f$, que teoricamente deveria ser uma indeterminação $0/0$, caso não houvesse erros de arredondamento.

1.9) Obtenha as duas raízes de $x^2 + 62.10x + 1 = 0$ utilizando $F(10,\ 4, -99, +99)$, com apenas 4 dígitos significativos totais no armazenamento e nas operações. Para tal:

a) Use a fórmula tradicional para obter as duas raízes.

b) Use a fórmula tradicional para obter a 1ª raiz e a fórmula racionalizada para obter a 2ª raiz, ambas com adição de parcelas.

c) Avalie os erros relativos nas duas formas de cálculo das raízes, para as raízes obtidas em (a) e em (b), sabendo que os seus valores exatos (com 6 dígitos) são $x_1 = -0.0161072$ e $x_2 = -62.0839$.

1.10) Calcule os valores numéricos da função $f(x) = 1 - \cos(x)$ e da sua forma trigonometricamente equivalente $g(x) = (sen(x) * sen(x)) / (1 + \cos(x))$ para $x = 10^{-i}$, $i = 1 \text{ a } 10$. Defina qual das formas é a mais exata.

1.11) Implemente o algoritmo, a seguir, em um computador com processamento numérico.

```
inteiro n
real x
defina n
x = 1/n
imprimir 'valor inicial x: ',x (com 30 significativos)
Para i = 1 até 100
       x = (n + 1) *x - 1
       imprimir i , x (com 30 significativos)
Fim para
```

Observação: use variáveis Reais de 32 e/ou 64 *bits* e explique a causa dos diferentes resultados obtidos para x, que teoricamente deveria ser constante, caso não houvesse erros de arredondamento.

a) Teste $n = 2,\ 3,\ 10 \text{ e } 16$ e avalie a evolução de x com o número de repetições i.

b) Mostre a instabilidade numérica gerada ao longo das repetições e explique a razão dos diferentes valores de x gerados nas repetições para cada n.

1.12) Dadas algumas estimativas do valor exato e de valores aproximados numericamente em um algoritmo, avalie o erro absoluto, relativo e percentual existentes nestas situações:

a) Valor Aproximado = 1102.345 e Valor Exato = 1100.9.

b) Valor Aproximado = 0.01245 e Valor Exato = 0.0119.

c) Compare os erros absolutos de (a) e (b) e verifique que este pode não refletir a realidade.

1.13) Dado o número decimal $x = -(10.05)_{10}$:

a) Calcule os 32 $bits$ da variável IEEE 754 que armazenam x em computadores. Mostre explicitamente a parcela binária que foi arredondada.

b) Calcule o decimal armazenado e o erro de arredondamento percentual gerado.

1.14) Dado o número binário $x = (1\ 00000000\ 10000000000000000000001)_2$ armazenado no padrão IEEE 754 de 32 $bits$, calcule o decimal x correspondente.

1.15) Dado o número decimal $x = -(1.51 * 10^{+37})_{10}$, calcule os 32 $bits$ da variável x armazenada no padrão IEEE 754 de 32 $bits$ e mostre explicitamente a parcela binária que foi arredondada.

1.16) Dado o número decimal exato $x = -(1.1 * 10^{-41})_{10}$:

a) Calcule os 32 $bits$ da variável x armazenada no padrão IEEE 754 de 32 $bits$ e mostre explicitamente a parcela binária arredondada.

b) Calcule o erro exato relativo percentual de arredondamento gerado na conversão de x decimal para binário da variável IEEE de 32 $bits$.

1.17) Explique como você calcularia em computador o erro devido aos arredondamentos existentes nos armazenamentos e nas operações usadas

para obter uma dada solução utilizando variáveis de 32 $bits$. Está disponível a variável de 64 $bits$.

1.18) Implemente um algoritmo que calcule e imprima o erro relativo percentual de **arredondamento** da função $\exp(x)$ calculada na variável *float*, por meio de aproximação por série de MacLaurin, com grau $n=4$ em $x=-0.111$, conforme a expressão a seguir:

$$\exp(x)=e^{x}\cong 1+\frac{x}{1!}+\frac{x^{2}}{2!}+\frac{x^{3}}{3!}+\frac{x^{4}}{4!}+...+\frac{x^{n}}{n!}$$

1.19) Implemente um algoritmo que calcule e imprima o erro relativo de **truncamento** da função $\exp(x)$ calculada na variável *double*, por meio de aproximação por série de MacLaurin, com $n=4$ em $x=-0.111$, conforme a expressão a seguir:

$$\exp(x)=e^{x}\cong 1+\frac{x}{1!}+\frac{x^{2}}{2!}+\frac{x^{3}}{3!}+\frac{x^{4}}{4!}+...+\frac{x^{n}}{n!}$$

Exercícios Capítulo 2

2.1) Classifique os possíveis tipos de soluções dos sistemas de equações lineares aplicando o método de Gauss com pivotamento parcial:

a) $\begin{cases} x_1 + 0.5x_2 - 1.75x_3 = -1 \\ 2x_1 + 1.5x_2 + 4.75x_3 = 8 \\ 4x_1 - 2x_2 + 1x_3 = 7 \end{cases}$

b) $\begin{cases} x_1 + 0.5x_2 - 1.75x_3 = -1 \\ 2x_1 + 1.5x_2 + 4.75x_3 = 8 \\ 3x_1 + 2x_2 + 3x_3 = 7 \end{cases}$

c) $\begin{cases} x_1 + 0.5x_2 - 1.75x_3 = -1 \\ 2x_1 + 1.5x_2 + 4.75x_3 = 8 \\ 3x_1 + 2x_2 + 3x_3 = 8 \end{cases}$

2.2) Dado o seguinte sistema: $\begin{cases} 1x_1 + 2.5x_2 + 2x_3 = -1 \\ x_1 - 2x_2 + 4x_3 = 10 \\ 3x_1 + 0.1x_2 - x_3 = -3 \end{cases}$

a) Resolva esse sistema de equações lineares pelo método de Gauss utilizando pivotação parcial.

b) Resolva esse sistema de equações lineares pelo método de Gauss utilizando pivotação total.

c) Calcule o determinante da sua matriz de coeficientes utilizando previamente o processo de eliminação adotado pelo método de Gauss em (a) e em (b).

2.3) Monte um algoritmo, tipo função do Octave, que receba o número de linhas n e a matriz A expandida representativa de um sistema, calcule e retorne o determinante da matriz de coeficientes, escalonada pelo processo de eliminação adotado pelo método de Gauss.

2.4) Dado o sistema de equações lineares: $\begin{cases} 3x_1 + 1.5x_2 + 4.75x_3 = 8 \\ 4.01x_1 + x_2 + 3x_3 = 7 \\ x_1 + 0.5x_2 - 0.05x_3 = -1 \end{cases}$

a) Resolva o sistema pelo método de eliminação de Gauss sem pivotamento.

b) Resolva o sistema pelo método de eliminação de Gauss com pivotamento parcial.

c) Resolva o sistema pelo método de eliminação de Gauss com pivotamento total.

d) Resolva o sistema pelo método de inversão de matriz com pivotamento parcial.

2.5) Monte um algoritmo, tipo função do Octave, que receba o número de equações *n* e *a* matriz de coeficientes A representativa do sistema, calcule e retorne a matriz inversa dos coeficientes de A usando uma extensão do método de eliminação de Gauss com pivotamento parcial.

2.6) Resolva o sistema de equações lineares pelo método de Crout:

$$\begin{cases} 4x_1 - x_2 + x_3 = 0 \\ -x_1 + 4.25x_2 + 2.75x_3 = 1 \\ x_1 + 2.75x_2 + 3.5x_3 = -1 \end{cases}$$

2.7) Resolva o sistema de equações lineares pelo método de Cholesky:

$$\begin{cases} 4x_1 - x_2 + x_3 = 0 \\ -x_1 + 4.25x_2 + 2.75x_3 = 1 \\ x_1 + 2.75x_2 + 3.5x_3 = -1 \end{cases}$$

2.8) Se para resolver um sistema de equações lineares da ordem de $n = 10$ (10 equações com 10 incógnitas) em computador, utilizando o método de eliminação de Gauss sem pivotamento, a solução é encontrada em **0.1 segundos** de processamento, estime o tempo que esse mesmo computador levaria para resolver um sistema da ordem de $n = 1000$ equações utilizando o mesmo método.

2.9) Avalie o condicionamento do sistema, a seguir, pelos dois critérios estabelecidos:

$$\begin{cases} x_1 + 0.5x_2 - 0.05x_3 = -1 \\ 3x_1 + 1.5x_2 + 4.75x_3 = 8 \\ 4.01x_1 + 2x_2 + 3x_3 = 7 \end{cases}$$

2.10) Que cuidados devemos tomar ao resolver sistemas mal condicionados por métodos diretos?

2.11) Seria indicado resolver sistemas mal condicionados por métodos iterativos? Justifique.

2.12) Monte um algoritmo otimizado tipo `function` *`AA=fescalona(n,A)`* que determine e retorne da `function` a matriz escalonada triangular superior *AA* expandida com B, a partir das entradas (*n,A*), em que n é o número de equações e $A = [A_0 \; B]$ é a matriz expandida original de um sistema genérico $A_0 * X = B$.

2.13) Sabendo que um computador X opera 10^6 operações em ponto flutuante por segundo, e que o número total de operações aritméticas envolvidas no método de Gauss é da ordem de $O((2/3)n^3)$ operações, responda:

a) Quanto tempo de CPU será necessário para resolver um sistema de $n = 1000$ equações nesse mesmo computador?

b) Quanta memória é necessária para armazenar um sistema de $n = 1000$ equações utilizando variáveis *double* (de 64 *bits* = 8 *Bytes*) na forma de matriz expandida?

2.14) Dado o seguinte sistema de equações:

$$\begin{cases} 2x_1 - x_2 + x_3 = -1 \\ x_1 - 2x_2 + x_3 = 1 \\ x_1 - 0.1x_2 - x_3 = 3 \end{cases}$$

a) Determine as matrizes *L* e *U* e a solução $S = \{x_1, x_2, x_3\}$ do sistema dado pelo método de decomposição *LU*, de Crout.

b) Avalie os resíduos finais das equações e verifique se a solução obtida tem uma precisão satisfatória (de acordo com o número de dígitos adotado).

2.15) Dados $m=4$ sistemas $A*X=B^m$, cada um de $n=3$ equações, a seguir, com a mesma matriz A e com $m=4$ termos independentes B^m diferentes:

$$\begin{array}{cccc} m=1 & m=2 & m=3 & m=4 \\ \begin{cases} 4x_1+x_2+2x_3=1 \\ x_1-2x_2+x_3=4 \\ x_1+0.1x_2-x_3=-3 \end{cases} & \begin{cases} 4x_1+x_2+2x_3=-1 \\ x_1-2x_2+x_3=10 \\ x_1+0.1x_2-x_3=-3 \end{cases} & \begin{cases} 4x_1+x_2+2x_3=7 \\ x_1-2x_2+x_3=-4 \\ x_1+0.1x_2-x_3=3 \end{cases} & \begin{cases} 4x_1+x_2+2x_3=5 \\ x_1-2x_2+x_3=3 \\ x_1+0.1x_2-x_3=1 \end{cases} \end{array}$$

$$\mathrm{A}=\begin{bmatrix} 4 & 1 & 2 \\ 1 & -2 & 1 \\ 1 & 0.1 & -1 \end{bmatrix} \quad \text{e} \quad \mathrm{B}=\begin{bmatrix} 1 & -1 & 7 & 5 \\ 4 & 10 & -4 & 3 \\ -3 & -3 & 3 & 1 \end{bmatrix}$$

Dadas as matrizes L_{nxn} e U_{nxn} decompostas da matriz A_{nxn} pelo método de Crout ($A=L*U$), monte um algoritmo genérico que determine as m soluções $S^m=\{x_1^m, x_2^m, x_3^m\}$ desses m sistemas $A*X=B^m$, por meio das duas substituições, $L*C^m=B^m$ e $U*X^m=C^m$, propostas por Crout.

Disponível: a `function [LU]=fLUCrout(n,A)`, que calcula e retorna as matrizes L_{nxn} e U_{nxn} (sem pivotação), sobrepostas em uma única matriz *LU*, por decomposição da matriz A_{nxn} pelo método de Crout ($A=L*U$).

2.16) Determine a solução x do sistema composto de uma matriz tridiagonal pelo método de Gauss otimizado:

$$\begin{cases} x_1-x_2=5 \\ x_1-3x_2-x_3=4 \\ x_2-2x_3+x_4=10 \\ x_3-x_4+x_5=3 \\ x_4+x_5=-3 \end{cases}$$

2.17) Monte um algoritmo que determine a solução do sistema dado na questão 2.16 pelo método de Gauss otimizado para matriz tridiagonal.

2.18) Dado o seguinte sistema linear:

$$\begin{cases} x_1+2x_2+x_3=4 \\ x_1+0.1\,\mathrm{x}_2+x_3=-3 \\ 4x_1+x_2+2x_3=1 \end{cases}$$

a) Verifique se esse sistema possui diagonal dominante.

b) Efetue uma pivotação parcial em todas as linhas.

c) Verifique novamente se o sistema pivotado apresenta diagonal dominante.

d) Determine a solução do sistema pivotado pelo método de Gauss-Seidel, com critério $Max\left|x_i^{k+1} - x_i^k\right| \leq \varepsilon$ $(\varepsilon = 10^{-2})$, partindo da solução inicial (0,0,0).

e) Se esse sistema for resolvido por métodos eliminativos (Gauss, por exemplo), verifique se é um sistema mal condicionado. Justifique.

2.19) Monte um algoritmo que otimize o método de eliminação gaussiana para solver um sistema de matriz "pentadiagonal" genérico de n equações com 5 faixas, conforme esta estrutura:

$$\begin{bmatrix} c_1 & d_1 & e_1 & 0 & \cdots & \cdots & 0 \\ b_2 & c_2 & d_2 & e_2 & \cdots & \cdots & 0 \\ a_3 & b_3 & c_3 & d_3 & e_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a_{n-1} & b_{n-1} & c_{n-1} & d_{n-1} \\ 0 & 0 & \cdots & \cdots & a_n & b_n & c_n \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

2.20) Dado o seguinte sistema de equações:

$$\begin{cases} x_1 + x_2 = 4 \\ x_1 - x_2 = 2 \end{cases}$$

a) Verifique se o sistema tem convergência garantida.

b) Determine sua solução pelo método de Jacobi.

c) Determine sua solução pelo método de Gauss-Seidel.

d) Como podemos sair desses ciclos repetitivos gerados nas sequências obtidas por Jacobi e por Gauss-Seidel?

e) Determine a sua solução por Gauss-Seidel usando um fator de sub-relaxação (com precisão de 6 dígitos).

2.21) Dado o sistema linear de ordem $n = 1000$, com 4 tipos de equações:

$$\begin{cases} i=1 & \rightarrow x_i + x_{i+1} = 150 \\ i=2,3,\ldots,\left(\dfrac{n}{2}\right) & \rightarrow x_{i-1} + 9x_i + x_{i+1} + x_{i+100} = 100 \\ i=\left(\dfrac{n}{2}+1\right),\ldots,(n-1) & \rightarrow x_{i-100} + x_{i-1} + 9x_i + x_{i+1} = 200 \\ i=n & \rightarrow x_{i-1} + x_i = 300 \end{cases}$$

Responda às seguintes questões:

a) Esse sistema tem convergência garantida para a solução quando resolvido por métodos iterativos, como o de Jacobi ou de Gauss-Seidel? Justifique.

b) É recomendável testar a utilização de fatores de relaxação? Justifique.

c) Monte um algoritmo otimizado que calcule e imprima a solução S desse sistema linear com 10 dígitos significativos exatos pelo método que efetue o menor número de operações aritméticas em ponto flutuante. Justifique a escolha do método adotado.

2.22) Dado o sistema linear de ordem $n_2 = 400$, com 4 tipos de equações:

$$\begin{cases} x_i - x_{i+1} = 0.1 & \rightarrow \quad para \;\; i=1 \\ -x_{i-1} + 2x_i - x_{i+1} = 0.1 & \rightarrow \quad para \;\; i=2,...,n_1-1 \\ -x_{i-2} - x_{i-1} + 3x_i - x_{i+1} = 0.2 & \rightarrow \quad para \;\; i=n_1,...,n_2-1 \\ -x_{i-1} + 2x_i = 0.3 & \rightarrow \quad para \;\; \mathrm{i}=n_2 \end{cases}$$

Responda às seguintes questões:

a) Considerando $n_1 = 300$ e $n_2 = 400$, a convergência será garantida se o sistema for resolvido por métodos iterativos? Justifique sua resposta.

b) Se esse sistema convergir “lentamente” para a solução por métodos iterativos, como a sua convergência pode ser acelerada? Justifique sua resposta.

c) Se esse sistema convergir “oscilando” para a solução por métodos iterativos, como a sua convergência pode ser acelerada? Justifique sua resposta.

d) Determine a solução S e o resíduo máximo das equações desse sistema, para $n_1 = 3$ e $n_2 = 4$, pelo método de Gauss (sem pivotação).

e) Determine uma solução S desse sistema para $n_1 = 3$ e $n_2 = 4$, com erro máximo estimado por $Max\,|\,x_i^{k+1} - x_i^k\,|$ de sua escolha, pelo método de Gauss-Seidel (sem fator de relaxação).

2.23) Dado o sistema linear tridiagonal com quatro tipos de equações:

$$\begin{cases} 2x_i - x_{i+1} = -1 & \rightarrow \quad para \;\; i = 1 \\ -x_{i-1} + 3x_i - x_{i+1} = 1 & \rightarrow \quad para \;\; i = 2,...,n_1 - 1 \\ -x_{i-1} + 3x_i - x_{i+1} = 2 & \rightarrow \quad para \;\; i = n_1,...,n_2 - 1 \\ -x_{i-1} + x_i = 3 & \rightarrow \quad para \;\; \mathrm{i} = n_2 \end{cases} \Rightarrow \quad \mathrm{t}_i x_{i-1} + r_i x_i + d_i x_{i+1} = b_i$$

Com $n_1 = 30$ e $n_2 = 50$ (total $n = 50$ equações), responda às questões:

a) Esse sistema tem convergência garantida quando resolvido por métodos iterativos? Justifique.

b) Se o sistema convergir oscilando, é recomendado testar a utilização de fatores de sub-relaxação? Justifique.

c) Se esse sistema convergir lentamente, é recomendado testar a utilização de fatores de sobrerrelaxação? Justifique.

d) Monte um algoritmo otimizado, tipo função do Octave, que:

i. receba 4 vetores "genéricos" para cada coeficiente não nulo da matriz tridiagonal: t, r, d, b; e

ii. calcule e retorne a solução exata x do sistema linear definido por t, r, d, b, pelo método direto de Gauss otimizado para matrizes tridiagonais.

e) Monte um algoritmo otimizado t,ipo função do Octave, que:

i. receba 4 vetores "genéricos" para cada coeficiente não nulo no mesmo formato usado no algoritmo para matrizes tridiagonais: t, r, d, b;

ii. calcule e retorne uma solução aproximada x do sistema linear definido por t, r, d, b, pelo método de Gauss-Seidel, operando apenas os coeficientes t, r, d, b não nulos, com fator de relaxação "*fator*" e com critério "relativo" de parada $\max\left|\left(x - xi\right) / x\right| < tolerancia$

.

f) Monte um algoritmo principal, tipo *main.m,* para:

i. definir os 4 vetores do sistema tridiagonal, t, r, d, b, para o sistema dado;

ii. determinar a sua solução exata pelo algoritmo otimizado para matrizes tridiagonais;

iii. determinar a sua solução aproximada x com $fator=1.2$ e $tolerancia=10^{-6}$ pelo algoritmo de Gauss-Seidel; e

iv. determinar o erro de truncamento máximo da solução aproximada x através de duas formas: via solução exata direta obtida para matrizes tridiagonais e via solução exata estimada pelo método de Gauss-Seidel. Ao executar este algoritmo, compare as duas formas de cálculo de erros e verifique que são equivalentes.

Dados para item (f.iv):

`function x=fTrid(n,t,r,d,b)` e

`function x=fGaussSeidelrelax(n,t,r,d,b,fator,tolerancia).`

Exercícios Capítulo 3

3.1) A equação $f(x) = x * tg(x) - 1 = 0$ (x radianos) tem infinitas raízes, quase periódicas.

a) Determine um intervalo que contenha a segunda raiz real positiva (excluir descontinuidades).

b) Determine o seu valor aproximado com erro máximo e método de sua escolha. Escreva a raiz e o critério de parada atingido.

c) Supondo que não seja conhecida a expressão da derivada de $f(x)$ da equação, monte um algoritmo completo que determine a sua segunda raiz real positiva pelo método da secante. Imprima a raiz aproximada e os erros encontrados.

3.2) Compare a eficiência entre os métodos de quebra da bisseção, falsa posição e falsa posição modificado e depois compare o melhor deles com o método de Newton determinando a raiz real positiva de $x^{10} - 2 = 0$ ($x = \sqrt[10]{2}$), com todos os dígitos exatos na variável *double*. Sugestão de intervalo inicial: $[0, 2]$ e $x_0 = 1$.

3.3) Compare o método da Newton tradicional de 1ª ordem com o método de Newton de 2ª ordem, partindo de uma mesma condição inicial x_0 e determinando uma raiz positiva de $x^{10} - 2 = 0$ ($x = \sqrt[10]{2}$) com todos os dígitos exatos na variável *double*. Estime o número de iterações e o número de operações aritméticas. Sugestão de valor inicial: $x_0 = 1$.

3.4) Compare a eficiência do método da secante com o método de Müller partindo de uma mesma condição inicial $x_0 = 1$ e determine uma raiz positiva de $P_{10}(x) = x^{10} - 2 = 0$ ($x = \sqrt[10]{2}$).

Sugestão de valores iniciais:

Secante: $x_0 = 1$ e $x_1 = x_0 * 1.01$

Müller: $x_0 = 1$, $x_1 = x_0 * 1.005$ e $x_2 = x_0 * 1.01$

3.5) Dada a equação $f(x) = x * tg(x) - 1 = 0$ (x radianos), monte um algoritmo que determine as cinco primeiras raízes positivas de $f(x) = 0$, com erro máximo $|f(x^k)| \leq 1e-15$ pelo método de Newton. Considere valores iniciais de cada raiz: $x_0 = [1,\ 3,\ 6,\ 9,\ 12]$, respectivamente.

3.6) Elabore uma rotina para efetuar a divisão entre dois números $d = a / c = a * (1 / c)$ sem utilizar a operação de divisão.

3.7) Crie uma rotina para obter $\sqrt[n]{c}$ sem o uso da potenciação/radiciação.

3.8) Dados o grau n e os coeficientes Reais a_i de uma equação polinomiais $P_n(x) = 0$, monte o algoritmo de busca, tipo função do *Octave*, que localize todas as suas raízes reais, positivas e negativas, dentro de um intervalo total $[-r_{\max},\ r_{\max}]$ e armazene em subintervalos de comprimento $h = 0.01$ que contenham cada raiz Real.

3.9) Crie uma rotina que calcule a primeira divisão de $P_n(x)$ por $(x-u)$ e retorne o primeiro resto R, que é o valor de $P_n(x=u)$.

3.10) Crie uma rotina que efetue $P_n(x)/(x-u)$ até a k-ésima divisão sucessiva e retorne o novo grau $n-k$ e os novos coeficientes do último quociente da divisão.

3.11) Crie uma rotina que calcule até a $(n+1)$-ésima divisão sucessiva de $P_n(x)$ por $(x-u)$ e retorne os $n+1$ restos R .

3.12) Crie uma rotina que calcule a derivada de ordem genérica k de um polinômio $P_n(x)$ em $x=u$ baseada no resto R_{k+1} da $(k+1)$-ésima divisão sucessiva de $P_n(x)$ por $(x-u)$.

3.13) Dado um polinômio de grau n e os restos R das divisões sucessivas de $P_n(x)$ por $(x-\alpha)$, crie uma rotina que calcule a multiplicidade M de uma raiz α de uma equação polinomial $P_n(x)=0$ através do número de restos R (dados) numericamente nulos. Considere "numericamente nulo" todo resto em módulo menor do que $R\text{limite}=0.1$ (com $P_n(x)=0$ normalizado pelo a_1).

3.14) Dada a equação polinomial $x^4+4x+1=0$:

a) Monte um quadro com as possibilidades de suas quatro raízes aplicando a regra de sinais de Descartes (nulas, positivas, negativas e/ou complexas).

b) Defina os módulos máximo e mínimo dessas raízes por alguma cota.

c) Estime os quatro valores iniciais para as quatro raízes de $P_4(x)=0$, considerando os resultados da regra de sinais de Descartes e dos módulos máximo e mínimo dessas raízes (todas raízes reais).

d) Determine uma segunda raiz real x_2 de $P_4(x)=0$ reduzindo o grau do polinômio inicial $P_4(x)$ através da primeira raiz $x_1=-0.250992157$ (dada) com o método de Newton otimizado para polinômios, a partir do valor inicial $xi_2=-1.5$, com erro máximo $|P_3(x_2)|\leq 0.001$

3.15) Dada a equação polinomial $x^4+x^2+4x+1=0$:

a) Monte um quadro com as possibilidades de suas quatro raízes aplicando a regra de sinais de Descartes (nulas, positivas, negativas e/ou complexas).

b) Defina os módulos máximo e mínimo dessas raízes.

c) Estime quatro valores iniciais para as quatro raízes de $P_4(x)=0$ considerando os resultados da regra de sinais e dos módulos máximo e mínimo dessas raízes.

d) Determine uma segunda raiz real x_2 de $P_4(x)=0$, a partir do valor inicial $xi_2=-1.25$, no polinômio de grau reduzido $P_3(x)=0$, dada a

primeira raiz $x_1 = -0.269472035$. Use o método de Newton otimizado para polinômios com erro máximo $|P_4(x_2)| \leq 10^{-9}$.

e) Refine uma segunda raiz real supondo que seu valor parcial seja $x_2 = -1.24938$ obtido de $P_3(x) = 0$. Use o método de Newton otimizado para polinômios com o polinômio original e erro máximo $|P_4(x_2)| \leq 10^{-9}$.

3.16) Dada a equação polinomial $x^5 + x^4 - x^3 - x^2 + 0.1x = 0$:

a) Extraia a primeira raiz nula de $P_5(x) = 0$ e determine o polinômio resultante $P_4(x)$.

b) Determine quatro valores iniciais (complexos) para as raízes de $P_4(x) = 0$ escolhidos randomicamente dentro do limite das cotas mínima e máxima. Use também a regra de Descartes (positivas, negativas e/ou complexas, nula já foi extraída em (a)).

c) Sabendo que a última raiz encontrada x_4 de $P_4(x) = 0$ é $+0.97363$, calculada em $P_1(x) = 0$ por reduções sucessivas de grau de $P_4(x) = 0$, purifique essa última raiz aplicando o método de Newton otimizado no polinômio exato original, sem erros da redução de grau $P_4(x) = 0$, com critério de parada máximo $|P_4(x)| \leq 10^{-9}$.

3.17) Monte um algoritmo que determine todos os n zeros, Reais, Complexos, simples ou múltiplos de um polinômio $P_n(x)$ definido pelos seus coeficientes a_i, com 16 dígitos significativos exatos e critério de parada $|\Delta x| \leq 10^{-14}$.

Dados:

-xi=fLocaliza(n,a)% xi é uma raiz inicial de Pn(x), genericamente complexa dentro de um círculo limitado pelas 3 cotas;

-[x M]=fNRPol(n,a,xi,tolerancia)%determina raiz 'x' e sua multiplicidade M, do polinômio de grau 'n' e coeficientes 'a', a partir de ‘xi’ com criterio ‘tolerancia’;

-b=fDivBrio(n,a,xi)%determina 'n+1' coeficientes 'b' da primeira divisão Pn(x), de grau 'n' e coeficientes 'a', por (x-xi), via Briot-Ruffini;

3.18) Dada a equação $x^3 - 3.3\,x^2 + 3.63\,x - 1.331 = 0$:

a) Determine as cotas-limite máxima e mínima, segundo a cota do módulo máximo, a cota de Cauchy e a cota de Kojima, para as raízes de $x^3 - 3.3\,x^2 + 3.63\,x - 1.331 = 0$. Qual das três cotas é a mais exata?

b) Faça uma análise do polinômio $x^3 - 3.3\,x^2 + 3.63\,x - 1.331 = 0$ e determine três valores iniciais para suas raízes.

c) Determine a única raiz de $x^3 - 3.3\,x^2 + 3.63\,x - 1.331 = 0$, com $|P_3(x_2)| \leq 10^{-9}$, sabendo que é uma raiz real positiva, a partir do valor inicial $xi = 1.0$.

d) Prove, ou justifique, que a multiplicidade da raiz $x = 1.1$ de $x^3 - 3.3\,x^2 + 3.63\,x - 1.331 = 0$ é $M = 3$. Sugestão: Use as divisões sintéticas de Briot-Ruffini.

3.19) Dada $x^3 - 2.7x^2 + 2.43x - 0.729 = 0$:

a) Determine as cotas-limite máxima e mínima segundo a cota do módulo máximo.

b) Determine as cotas-limite máximas e mínimas segundo a cota de Cauchy (escolha um critério de parada);

c) Determine as cotas-limite máximas e mínimas segundo a cota de Kojima.

d) Qual dessas cotas seria a mais adequada, mais exata?

e) Determine três valores iniciais randômicos para suas três raízes, dentro das cotas obtidas.

f) Determine uma raiz x de $P_3(x) = 0$ e sua multiplicidade M, com $|P_3(x)| \leq 10^{-15}$, a partir do valor inicial $xi = 1.0$, pelo método de Newton corrigido com multiplicidade M.

Exercícios Capítulo 4

4.1) Calcule a solução X para o sistema de $n=2$ equações não lineares pelo método de Newton:

$$\begin{cases} f_1(x_1, x_2) = \sin(x_1) + \cos(x_2) - 1 = 0 \\ f_2(x_1, x_2) = x_1^{\,2} + x_2^{\,2} - 3 = 0 \end{cases}$$

Considerando como valores iniciais $X^0 = [1,\ 1]$ e como critério limite de parada $\max(|\Delta x_j|) < 10^{-2}$.

4.2) Monte um algoritmo que determine e imprima a solução X do sistema de $n=2$ equações não lineares pelo método de Newton:

$$\begin{cases} f_1(x_1, x_2) = \sin(x_1) + \cos(x_2) - 1 = 0 \\ f_2(x_1, x_2) = x_1^{\,2} + x_2^{\,2} - 3 = 0 \end{cases}$$

Considerando como valores iniciais $X^0 = [1,\ 1]$ e como critério limite de parada $\max(|\Delta x_j|) < 10^{-14}$.

4.3) Aplique um algoritmo que determine e imprima a solução X do sistema de $n=2$ equações não lineares pelo método de Newton com derivadas calculadas numericamente:

$$\begin{cases} f_1(x_1, x_2) = \sin(x_1) + \cos(x_2) - 1 = 0 \\ f_2(x_1, x_2) = x_1^{\,2} + x_2^{\,2} - 3 = 0 \end{cases}$$

Considerando como valores iniciais $X^0 = [1,\ 1]$ e como critério limite de parada $\max(|\Delta x_j|) < 10^{-14}$.

4.4) Aplique um algoritmo que determine e imprima a solução X do sistema de $n=2$ equações não lineares pelo método de Broyden com derivadas calculadas numericamente:

$$\begin{cases} f_1(x_1, x_2) = \sin(x_1) + \cos(x_2) - 1 = 0 \\ f_2(x_1, x_2) = x_1^{\,2} + x_2^{\,2} - 3 = 0 \end{cases}$$

Considerando como valores iniciais $X^0 = [1,\ 1]$ e como critério limite de parada $\max(|\Delta x_j|) < 10^{-14}$.

4.5) A equação de estado de Van der Walls modela a relação entre a pressão P, o volume v e a temperatura T de um gás através da função $\left(P + \frac{a}{v^2}\right)(v - b) = R * T$, em que $R = 8.314\ (\mathrm{J/mol} * \mathrm{K})$ é a constante universal dos gases, a e b são parâmetros característicos de cada gás. Para determinado gás, os valores de P, v e T estão no quadro a seguir:

Estados físicos:	1	2
T (K)	300	600
v (m3/kmol)	0.5	0.2
P (kPa)	6235.10	49881.50

Com essa função, aplicamos os dois valores do quadro e geramos o sistema de duas equações não lineares a seguir:

$$\begin{cases} f_1(a,b) = \left(P_1 + \frac{a}{v_1^{\,2}}\right)(v_1 - b) - R * T_1 = 0 \\ f_2(a,b) = \left(P_2 + \frac{a}{v_2^{\,2}}\right)(v_2 - b) - R * T_2 = 0 \end{cases}$$

Monte um algoritmo que determine os 2 parâmetros a e b desse gás, com critério de parada $\sum_{j=1}^{n} \left|\Delta x_j\right| < 10^{-14}$, usando o método de Newton com derivadas numéricas, a partir da solução inicial $\left(a^{(0)},\ b^{(0)}\right) = (0.2,\ 0.2)$.

4.6) Determine uma solução X do sistema de 3 equações não lineares:

$$\begin{cases} f_1(x_1, x_2, x_3) = \sin(x_1)\cos(x_2) + x_3 - 1.5 = 0 \\ f_2(x_1, x_2, x_3) = x_1^{\,2} + x_2^{\,2} + x_3^{\,2} - 3 = 0 \\ f_3(x_1, x_2, x_3) = x_1 + x_2 + x_3 - 3.1 = 0 \end{cases}$$

Use os métodos de Newton e de Broyden com derivadas numéricas, exatidão máxima possível e variáveis *double*. Teste diferentes valores iniciais X^0, incluindo os complexos.

Exercícios Capítulo 5

5.1) Suponha que você precise avaliar a função $f(x) = sen(x)$, em $x \in [0, \pi/2]$ (x radianos), utilizando apenas operações algébricas via interpolação polinomial, para utilizar posteriormente em um processador embarcado, e não em um computador com biblioteca completa de funções:

a) Determine o grau n mínimo necessário do polinômio interpolador $P_n(x)$ para que o erro de truncamento máximo estimado seja da ordem $10^{-2} (< 10^{1/2} * 10^{-2})$, via limite estabelecido pelo resto da série de Taylor (**Corolário 2)**.

b) Determine a expressão de um interpolador polinomial $P_n(x)$ representativo de $f(x)$ com $n = 2$ no intervalo $x \in [0, \pi/2]$.

c) Avalie o erro de truncamento máximo estimado pelo **Corolário 2** e o erro máximo exato $|P_2(x) - sen(x)|$. Verifique que o erro máximo exato deve ficar abaixo do erro de truncamento máximo estimado pelo **Corolário 2**.

d) Elabore um algoritmo que plote o interpolador $P_n(x)$, representativo de $f(x) = sen(x)$, em $x \in [0, \pi/2]$, para grau n genérico.

5.2) Suponha que você esteja usando uma linguagem de programação muito eficiente computacionalmente, mas que não dispõe de todas as bibliotecas matemáticas necessárias. Então, utilizando apenas operações algébricas na função $f(x) = \exp(sen(x))$, $x \in [0, \pi/2]$:

a) Avalie $P_n(x = 0.378)$ com $n = 2$ e calcule o erro exato, via determinação do polinômio interpolador escrito na base canônica.

b) Avalie $P_n(x = 0.378)$ com $n = 2$ e calcule o erro exato, via utilização do interpolador escrito na base dos polinômios de Lagrange.

c) Avalie $P_n(x = 0.378)$ com $n = 2$ e calcule o erro exato, via utilização do interpolador de Gregory-Newton com diferenças divididas.

5.3) Suponha que você precise calcular a função $f(x) = \cos(x)$, em $x \in [0, \pi/2]$ (radianos), utilizando apenas operações algébricas via interpolação polinomial:

a) Monte um algoritmo completo que determine o erro máximo exato do interpolador $P_n(x)$, representativo de $f(x)=\cos(x)$, no intervalo $x\in[0,\pi/2]$, para um grau n genérico, e plote os erros exatos em cada x do intervalo (escolha um número de pontos).

b) Monte um algoritmo completo de busca do menor grau n que incremente o n gradativamente, enquanto o erro máximo exato entre o interpolador $P_n(x)$ e a função exata $f(x)=\cos(x)$, no intervalo $x\in[0,\pi/2]$, seja maior do que $O(10^{-6})$. Plote os erros exatos em cada x do intervalo para o n mínimo (escolha um número de pontos para plotar o gráfico).

5.4) Dada a tabela com quatro pontos:

i	1	2	3	4
x_i	0	1	2	3
y_i	-3	-2	4	0

a) Determine a segunda *spline* cúbica natural, correspondente ao segundo intervalo.

b) Monte um algoritmo que determine os coeficientes das três *splines* cúbicas, com extremos quadráticos, aproximadoras dessa função.

c) Elabore um algoritmo genérico de busca que determine em qual das três *splines* (em qual intervalo) deve ser calculado o valor de y correspondente a um determinado x. Por exemplo, qual intervalo contém $x=1.3$?

d) Determine o valor de y em $x=1.3$.

e) Plote o gráfico das $m=3$ *splines*.

5.5) Monte um algoritmo com três segmentos de curvas de Bezier que desenhe o perfil superior de um aerofólio. Esses três segmentos devem passar por quatro pontos de ancoragem, A, B, C e D, com inclinações pré-definidas conforme segue:

%A(0,0) com inclinação de 45º depois de A;

%B(2,1) com inclinação de 0º antes de B e inclinação -15º depois de B;

%C(8,0.2) com inclinação de -8º antes de B e inclinação de -8º depois de B;
%D(10,0) com inclinação de -5º antes de D.
Experimente diferentes pontos intermediários.

Exercícios Capítulo 6

6.1) Podemos avaliar uma função como $f(x) = sen(x)$, em $x \in [-1, +1]$ (x radianos), utilizando apenas operações algébricas como adição, subtração, multiplicação e divisão?

a) Uma alternativa de aproximação é o interpolador polinomial $P_n(x)$. Assim, elabore um algoritmo de busca que determine o grau n mínimo necessário, e os coeficientes de $P_n(x)$, para que o erro de truncamento máximo exato entre $P_n(x)$ e $f(x)$ seja da ordem de $O(10^{-2})$ ($< \sqrt{10} * 10^{-2}$).

Sugestão: monte um algoritmo de busca que incremente sequencialmente o valor de n enquanto o erro de truncamento máximo exato esteja maior do que $< \sqrt{10} * 10^{-2}$;

b) Uma segunda alternativa de representação é a expansão de $f(x)$ em termos da série de Maclaurin $M_n(x)$. Assim, determine ou monte um algoritmo de busca que determine o grau n mínimo necessário, e os coeficientes de $M_n(x)$, para que o erro de truncamento máximo exato entre $M_n(x)$ e $f(x)$ seja da ordem de $O(10^{-2})$ ($< \sqrt{10} * 10^{-2}$).

c) Uma terceira alternativa de representação é a expansão algébrica de $f(x)$ em termos da série de Tchebychev-Maclaurin $T_n(x)$. Assim, determine algebricamente os coeficientes da série de Tchebychev-Maclaurin $T_n(x)$ para $n = 3$ e $n = 5$, o seu erro máximo exato entre $T_n(x)$ e $f(x)$. Os erros máximos normalmente estão nas extremidades do intervalo $[a, b]$, mas, para a série de Tchebyschev, os erros estão distribuídos no intervalo, então calcule erros em alguns pontos de $[a, b]$ e tome o maior desses erros como referência.

d) Determine numericamente, via teorema de Tchebychev, os coeficientes da série de Tchebyschev $T_n(x)$ para $n = 3$ e $n = 5$, e o seu erro máximo exato entre $T_n(x)$ e $f(x)$.

e) Outra representação é a expansão de $f(x)$ em termos da série racional de Padé $R_{nm}(x)$. Assim, determine ou monte um algoritmo que

determine os coeficientes da aproximação de Padé $R_{32}(x)$ a partir de Maclaurin com grau total $M = 5$, e o seu erro de truncamento máximo exato entre $R_{nm}(x)$ e $f(x)$.

f) Plote os gráficos dos erros exatos entre as aproximações de Tchebyschev e Padé com $f(x)$.

6.2) Aproxime $f(x) = \ln(x)$ em $x \in [+0.1, +2.0]$ com interpolador polinomial, série de Maclaurin, Tchebyshev e Padé e determine os graus necessários para que os erros máximos sejam da ordem de $O(10^{-6})$. Imprima os erros máximos atingidos em cada caso.

Exercícios Capítulo 7

7.1) A tabela, a seguir, com m=6 pontos obtidos experimentalmente, relaciona a **medição do volume** adimensional de álcool gerado V em um reator fictício em função da sua temperatura adimensional T de reação:

%T=[0.2 0.4 0.6 0.8 0.9 1.0];

%V=[0.04 0.14 0.30 0.45 0.61 0.69];

Considere que o comportamento do volume de álcool gerado V em função da sua temperatura de reação T é **conhecido** e dado por uma função não polinomial $V(T) = \ln\left(a + b * T^2\right)$:

a) Determine as duas equações **não lineares** que permitem calcular diretamente os parâmetros a e b através da minimização do desvio quadrático $D(a,b)$ entre $V(T)$, calculado em cada ponto T_k, e o valor efetivamente medido de V_k, para $k=1$ até m:

$$D(a,b) = \sum_{k=1}^{m} \left(\ln\left(a + b * T_k^2\right) - V_k \right)^2 .$$

b) Determine os parâmetros a e b resolvendo as duas equações **não lineares** obtidas em (a), de modo a levar em conta todas as m medições experimentais.

c) Calcule o módulo do desvio local em todos os T_k e o Coeficiente de Determinação R^2.

d) Plote um gráfico com os m pontos experimentais, a função ajustada e uma função interpoladora que passe sobre os m pontos. Diga qual foi a melhor aproximação para os m pontos.

7.2) Em que situações você empregaria o **ajuste** polinomial em vez da **interpolação** polinomial?

7.3) Quando determinamos um aproximador através da **interpolação** polinomial, este satisfaz uma determinada condição. Qual é essa condição?

7.4) Quando determinamos um aproximador, polinomial ou não, através do **ajuste** pelos mínimos quadrados, esta função satisfaz uma determinada condição. Qual é essa condição?

7.5) Quando ajustamos um polinômio $P_n(x)$ de grau n a uma tabela de $m = n+1$ pontos, estamos obtendo o próprio polinômio interpolador. Justifique essa afirmação.

7.6) A tabela, a seguir, relaciona, experimentalmente, a medição do **volume** de álcool gerado em uma mistura em função da sua **temperatura** de reação:

Temperatura(^oC)	13.9	37.0	67.8	79.0	85.5	93.1	99.2
Volume(cm^3)	1.04	1.18	1.29	1.35	1.28	1.21	1.06

a) Determine as funções representativas dos pontos tabelados por **ajuste** de curvas na faixa medida. Sugestão: use função polinomial de grau $n=1$ e $n=2$.

b) Determine uma função representativa dos pontos tabelados por **interpolação** polinomial na faixa medida.

c) Plote um gráfico com os pontos experimentais e as funções aproximadoras obtidas em (a) e em (b).

d) Calcule o Coeficiente de Determinação R^2.

e) Analise e defina a metodologia mais adequada para representar o comportamento do volume de álcool em função da temperatura na faixa medida. Justifique sua resposta.

7.7) A tabela, a seguir, com $m=4$ pontos, obtida experimentalmente, relaciona a viscosidade adimensional V de um material fictício em função da sua temperatura adimensional T:

T	0.00	0.39	0.78	1.18
$V(T)$	0.99	0.92	0.71	0.38

Considere que o comportamento da viscosidade V em função da temperatura T do material é conhecido e dado por uma função não polinomial $V(T) = a * T + b * \cos(T)$, ($T$ radianos):

a) Monte as duas equações **lineares**, efetue cada somatório e calcule os parâmetros a e b através de ajuste de curvas “direto”, minimizando o desvio quadrático total, de modo a levar em conta todas as $m = 4$ medições experimentais. Use o método de Cholesky para resolver essas equações lineares.

b) Calcule os m desvios locais, o seu valor médio e o coeficiente de determinação.

c) Plote o gráfico da função ajustada e dos desvios (resíduos locais).

d) Avalie se o ajuste ficou razoável.

Exercícios do Capítulo 8

8.1) Considere a integral $I = \int_1^2 \sqrt[3]{(x+1)}\ dx$:

a) Determine o número n de subdivisões necessárias para que I tenha um erro de truncamento máximo inferior a 10^{-6} quando calculado pelo método dos Trapézios T_n.

b) Se você integrar I numericamente com o número n de subdivisões obtidos no item (a), o erro total cometido será inferior a 10^{-6}? Justifique sua resposta.

c) Determine em computador o número n de subdivisões de modo que o erro exato e estimado de T_n seja mínimo ($\text{Erro estimado } T_n = |T_n - T_{2n}|$);

d) Determine o número n de subdivisões necessárias para que S_n tenha um erro de truncamento máximo inferior a 10^{-6} calculado pelo método de Simpson.

e) Determine o valor do m necessário para que G_m tenha um erro de truncamento máximo inferior a 10^{-6} ao efetuarmos essa integral pelo método de Gauss-Legendre.

f) Determine a integral aproximada de pelos três métodos citados anteriormente, com $n = 2$ subdivisões (ou $m = n+1 = 3$ para Gauss-Legendre).

g) Avalie o erro exato e o erro estimado em cada caso.

h) Qual dos métodos de integração referidos nesta questão é computacionalmente o mais eficiente?

8.2) Considere a integral $I = \int_1^2 \sqrt{x}\ dx$:

a) Calcule o número m mínimo de pontos do intervalo de integração necessário para que I tenha um erro de truncamento máximo $O(10^{-5})$ quando calculada pelo método de Gauss-Legendre.

b) Calcule a integral numérica de Gauss-Legendre com $m = 3$ pontos.

c) Monte um algoritmo de busca que determine e imprima o número de pontos m mínimo (tabela com m até 5 pontos) para que a integral aproximada pelo método de Gauss-Legendre G_m tenha um erro de truncamento máximo estimado, através de $|G_m - G_{m+1}|$, da ordem de 10^{-6}. Dado $Gm = fGm(a,b,m)$, que determina o valor da integral numérica Gm de $f(x)$ pelo método de Gauss-Legendre (m até 5 pontos).

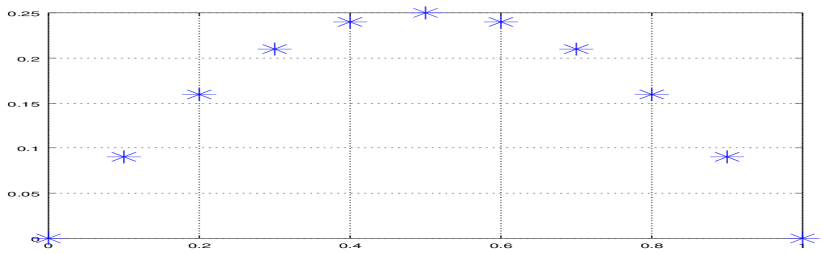
Sugestão: monte um algoritmo que incremente sequencialmente o valor de m enquanto o erro de truncamento máximo estimado estiver maior do que $10^{1/2} * 10^{-6}$.

8.3) Se $y = f(x)$ for contínua em $[a, b]$, então o comprimento L da curva gerada por $f(x)$ nesse domínio é fornecido por: $L = \int_a^b \sqrt{1+(f'(x))^2}\,dx$.

Determine o comprimento L do segmento de curva da $f(x) = \sqrt{x^3}$ do ponto de coordenadas $(1, 1)$ até $(3, 3\sqrt[3]{3})$. Use o método de Gauss-Legendre com exatidão 10^{-8}.

8.4) Em um projeto foram estabelecidas as coordenadas de 11 pontos (x_i, y_i), conforme a tabela a seguir:

x_i	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
y_i	0.00	0.09	0.16	0.21	0.24	0.25	0.24	0.21	0.16	0.09	0.00



a) Calcule a área gerada entre esses pontos e o eixo das abcissas pelo método dos Trapézios, T_n.

b) Calcule essa mesma área pelo método de Simpson, S_n.

c) Qual das duas aproximações numéricas resulta no cálculo mais exato dessa área? Justifique com argumentos matemáticos.

d) É conhecido da literatura pertinente que o método de Simpson é mais exato do que o método dos Trapézios. Calcule uma estimativa do erro de truncamento da área calculada pelo método dos Trapézios.

e) Se durante o desenvolvimento do projeto fosse necessário calcular essa área com mais exatidão, que ações poderiam ser adotadas? Justifique.

8.5) Considere a integral $I = \int_0^1 \ln(x)\, dx$, $\left(I_e = \int_0^1 \ln(u)\, du = u(\ln(u) - 1)\Big|_0^1 = -1 \right)$

a) Quais métodos podem ser aplicados para calcular numericamente essa integral imprópria? Justifique.

b) Monte uma função $Gm = fGm(a, b, m)$ para integrar numericamente uma função $f(x)$ entre $[a,\ b]$ pelo método de Gauss-Legendre, com até $m = 7$ pontos, em precisão *double*.

c) Determine e imprima $Gm(m)$ e os erros exatos de $Gm(m)$, com $m = 2$ até 7 pontos. Dado $Gm = fGm(a, b, m)$, que determina o valor da integral numérica Gm de $f(x)$ pelo método de Gauss-Legendre (m até 7 pontos)

8.6) Dada a função erro, também chamada função erro de Gauss, criada para calcular a integral da distribuição normal, definida por $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-z^2} dz$, elabore um algoritmo principal e suas funções e calcule e imprima cada item a seguir:

a) O n mínimo para que o valor de $erf(x)$ em $x = 1$ pelo método dos Trapézios T_n tenha erro máximo estimado da ordem de 10^{-6}. Calcule T_n e o seu erro exato (use a função $erf(x)$ disponível como valor exato).

b) O n mínimo para que o valor de $erf(x)$ em $x = 1$ pelo método de Simpson S_n tenha erro máximo estimado da ordem de 10^{-6}. Calcule S_n e o seu erro exato (use a função $erf(x)$ disponível como valor exato).

c) O m mínimo para que o valor de $erf(x)$ em $x=1$ pelo método de Gauss-Legendre G_m tenha erro máximo estimado da ordem de 10^{-6}. Calcule G_m e erro exato de G_m (use a função $erf(x)$ disponível como valor exato).

d) Os m (use m mínimo de (c)) coeficientes do polinômio interpolador $P_n(z)$, na forma geral, de grau $n=m-1$, que passe sobre os pontos os m pontos (z_k, y_k) calculados internamente no método de Gauss-Legendre.

e) A integral exata desse polinômio interpolador $P_n(z)$, de grau $n=m-1$, que passe sobre os m pontos (z_k, y_k) utilizados internamente pelo método de Gauss-Legendre e compare com G_m.

f) Trace um gráfico com a função integranda exata $f(z)=2/sqrt(pi)*\exp(-z^2)$, com a função aproximadora polinomial $P_n(z)$, e marque os m pontos (z_k, y_k) utilizados internamente no método de Gauss-Legendre para calcular a integral aproximada G_m.

8.7) Determine a integral $\int_{-1}^{+1} \frac{\ln(1+x)}{\sqrt{1-x^2}} dx$ por Gauss-Legendre e Gauss-TChebyshev com erro máximo exato $O(10^{-7})$ sabendo que o seu valor exato é $I_e = -\pi * \ln(2)$. Observe que essa não é uma função bem comportada ($\ln(1+x)$ é uma função assintótica) e analise a adequação de cada método à integração proposta.

Exercícios Capítulo 9

9.1) Resolva numericamente o PVI de 1ª ordem $y' - y = x^2$ com $x \in [0.0,\ 0.2]$, condição inicial $y(0) = 0$, e imprima a tabela de resultados x e y. Obtenha $y(0.2)$ pelo métodos numéricos de:

a) Euler com $n = 2$ subdivisões do intervalo.

b) Runge-Kutta de 2ª ordem com $n = 2$ subdivisões do intervalo.

c) Runge-Kutta de 4ª ordem com $n = 2$ subdivisões do intervalo.

d) Calcule o erro total estimado do valor de $y(0.2)$ obtido por Runge-Kutta de 2ª ordem com $n = 1$, única divisão, comparando-o com o cálculo $y(0.2)$ com $n = 2$ subdivisões e calcule o erro exato, para comparação ($ye(x) = 2 * \exp(x) - x^2 - 2 * x - 2$).

9.2) Para o PVI $x * y' + 2y = 0$ com $x \in [1.0,\ 1.2]$ e $y(1.0) = 1.0$:

a) Obtenha o valor de $y(1.2)$ pelo método de Euler com $n = 4$ divisões do domínio.

b) Monte um algoritmo que calcule e imprima o valor de $y(1.2)$ obtido pelo método de Runge-Kutta de 4ª ordem com $n = 8$ divisões.

c) Monte um algoritmo que calcule e imprima uma estimativa do erro total existente na solução numérica do item anterior.

9.3) Para a equação diferencial ordinária de 2ª ordem $x * y'' + 2y + x = 0$ com $x \in [1.0,\ 1.2]$, $y(1.0) = 1.0$ e $y'(1.0) = -1.0$ (valores iniciais):

a) Transforme-a em um sistema de duas EDOs de 1ª ordem equivalente com os respectivos valores iniciais.

b) Elabore um algoritmo que resolva o sistema de 2 equações diferenciais ordinárias de 1ª ordem equivalente, pelo método de Runge-Kutta de 4ª ordem. Use um n da sua escolha e imprima a solução (x, y) em todos os $n+1$ pontos obtidos.

9.4) Determine o sistema de 2 equações diferenciais ordinárias de 1ª ordem, $y_1'(x)$ e $y_2'(x)$, que represente a equação diferencial ordinária de 2ª ordem $y''y'y = -(x^{3/2})/8$, $x \in [1.0,\ 1.2]$ e condições iniciais, $y(1.0) = 1.0$ e $y'(1.0) = 1/2$.

9.5) Dada a equação diferencial ordinária de 2ª ordem, $x * y'' + 2 * y' = 0$, com $x \in [1.0,\ 1.2]$:

a) Transforme essa EDO de 2ª ordem em duas EDOs de 1ª ordem em $y_1' = fy_1(x, y_1, y_2)$ e $y_2' = fy_2(x, y_1, y_2)$. Monte essas duas *functions*: `y=fy1(x,y1,y2)` e `y=fy2(x,y1,y2)`;

b) Dadas duas condições iniciais para a EDO de 2ª ordem, $y(x=1) = 1$ e $y'(x=1) = -1$, e as *functions* com as derivadas de y_1 e y_2: `y=fy1(x,y1,y2)` e `y=fy2(x,y1,y2)`, monte um algoritmo completo que determine $n+1$ valores de x, y_1 e y_2 pelo método de Runge-Kutta de 4ª ordem, para um n adequado ao intervalo de integração.

c) Dadas duas condições de contorno para a EDO de 2ª ordem, $y(x=1) = 1$ e $y(x=2) = 0.5$, monte um algoritmo que determine o valor da condição inicial desconhecida $y'(x=1) = C$ pelo "Shooting Method", ou Método de Newton, para um n adequado ao intervalo de integração, considerando que estejam disponíveis:

i) uma *function* que determine $n+1$ valores de x, y_1 e y_2 pelo de Runge-Kutta de 4ª ordem:

`[x y1 y2]=fsis2EDORK4(n,a,b,x,y1,y2)`, a partir do número de intervalos n, dos limites do domínio de cálculo $x \in [a,b]$ e de valores iniciais x, y_1 e y_2;

ii) as *functions*: `y=fy1(x,y1,y2)` e `y=fy2(x,y1,y2)`.

9.6) Dada a equação diferencial ordinária de 3ª ordem (equação de Blasius, que representa o escoamento de fluido sobre uma placa plana) $F''' + (1/2) * F * F'' = 0$, $x \in [0,10]$ com condições de contorno:

$F(x=0) = 0$

$F'(x=0) = 0$

$F''(x=0)=C$ (condição desconhecida (mira))

$F'(x=10)=1$ (condição de contorno conhecida (alvo)).

a) Monte um algoritmo que determine a condição inicial $F''(x=0)$, para completar as três condições iniciais necessárias, usando o *Shooting Method*, ou método de Newton (Secante), com o Runge-Kutta de 4ª ordem e erro máximo em C da ordem de $1e-8$.

b) Depois de determinar o C do item (a), plote os gráficos de F, F' e F'' ao longo do eixo x.

c) Determine o erro de truncamento máximo exato estimado da solução aproximada $F(x)$ com $n=128$ e imprima a localização desse erro máximo.

9.7) Dada a equação diferencial ordinária de 4ª ordem $F''''+6/x^4=0$, $x\in[1,2]$ com 4 condições de contorno:

$F(x=1)=0$

$F'(x=1)=1$

$F(x=2)=\ln(2)$

$F'(x=2)=0.5$

Temos 2 condições desconhecidas em $x=1$:

$F''(x=1)=C_1$

$F'''(x=1)=C_2$

e 2 condições conhecidas em $x=2$:

$F(x=2)=\ln(2)=D_1$

$F'(x=2)=0.5=D_2$

a) Use o método de Newton com derivadas numéricas, com Runge-Kutta de 4ª ordem e determine as duas condições iniciais desconhecidas, C_1 e C_2, com erro no limite da precisão da variável *double*, para completar as quatro condições iniciais necessárias.

b) Depois de determinar as duas condições iniciais desconhecidas, C_1 e C_2, determine a solução F com um número de subdivisões n (por

tentativas) de modo que o seu erro máximo fique na ordem de $1e-6$. Plote o gráfico de F ao longo do eixo x (solução exata $Fe(x) = \ln(x)$).

RESPOSTAS DOS EXERCÍCIOS:

Respostas Capítulo 1

(1.1)

a) $(2.6875)_{10}$; b) $(1010.10010001111010...)_2$

(1.2)

b) $(119)_{10}$; c) $(120)_{10}$; e) $(0.79166666...)_{10}$; f) $(23.95833333...)_{10}$

(1.3)

a) $(17.D)_{16} = (23.8125)_{10}$ (não haverá perdas, a menos que o número de dígitos significativos decimais seja menor do que o necessário).

b) $(1011\ 1101.0000\ 1110)_2 = (189.0546875)_{10}$ (não haverá perdas, a menos que o número de dígitos significativos decimais seja menor do que o necessário).

c) $(101001.00110011...)_2 = (29.33333...)_{16}$ (haverá perdas de significação, independentemente do número de dígitos significativos decimais disponível).

(1.4)

a) 4;

b) 7;

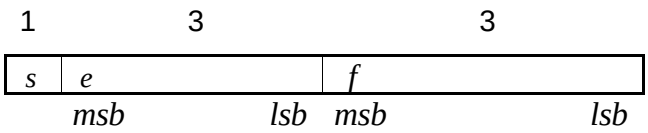
c) 57;

d) ($MP = 7$ e $mp = 0.0625$);

e) $2^{-(t-1)} = 10^{-(d-1)} \rightarrow 2^{-(3-1)} = 10^{-(d-1)} \rightarrow d = 1.602059991$ - precisão 1-2 decimais ou $2^{-(3-1)} = 2.5 * 10^{-1}$ - precisão ~1 decimal, devido ao expoente -1);

f) O padrão **IEEE 754** equivalente seria $F(2,3,0,7)$, normalizado com $d_1 \neq 0 = 1$ representado implicitamente antes do ponto (vírgula), mais três $bits$ de f na

parte fracionária alocados depois do ponto e polarização $p=+3$ (então o verdadeiro expoente seria $(e-3)$, conforme:



Se $0<e<7$, então $v = (-1)^s 2^{e-3}(1.f)_2$;

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

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

Se $e=7$, então v pertence a região de *overflow*.

(1.5)

a) 8;

b) 8;

c) 129;

d) ($MP=15$ e $mp=0.03125$);

e) $2^{-(t)} = 10^{-(d-1)} \rightarrow 2^{-(-3)} = 10^{-(d-1)} \rightarrow d = 1.903089987$ – precisão 1-2 decimais ou

$2^{(-3)} = 1.25 * 10^{-1}$ – precisão ~1 decimal, devido ao expoente -1);

(precisão mais abrangente do que no exercício **1.4**)

f) $(-1)^s 2^{-2}(0.)$.

(1.6)

$MP = (-1)^s 2^{(2046-1023)}(1.1111.....11111)_2 = 8.98846567e+307$;

$mp = (-1)^s 2^{(-1022)}(0.0000....00001)_2 = 4.94065646e-324$.

(1.7)

a)

$Dmsb = 1 * 2^0 = 1.0 * 10^0$ – 1º dígito decimal, antes do ponto.

$+ Dlsb = 1 * 2^{-52} = 2.220446049250 * 10^{-16}$ – 16º dígito decimal, depois do ponto.

$= \underline{1.0000000000000000}222...$ – precisão de 17 dígitos decimais totais

b) $d = 1 + t * \log(2) = 16.65355977$ (para $t = 52$ $bits$, depois do ponto (vírgula)), logo a precisão decimal equivalente está entre 16 e 17 decimais.

c) Armazene $x = 0.1$ (decimal exato) em *double* e imprima-o com 30 decimais: $x = 0.1000000000000000055511151231 26$ (tem 17 decimais exatos, sublinhados). A precisão decimal equivalente está entre 16 e 17 dígitos.

1.8)

% em Octave, com *double*:

```
h= 1/2
x= 2/3 - h
y= 3/5 - h
e= (x + x + x) - h
f= (y + y + y + y + y) - h
g= e/f
printf("\n %.30f",g)
```

Respostas:

$h = 0.50000$

$x = 0.16667$

$y = 0.10000$

$e = -1.1102 * 10^{-16}$

$f = -1.1102 * 10^{-16}$

$g = 1$

Vemos que $g = e / f$ gerou um valor inteiro porque os valores residuais de e e f foram iguais em *double*. Isso serve de alerta, pois provavelmente não desconfiaríamos de um resultado "inteiro" como $g = 1$, que, na verdade, se trata de um resultado inconsistente. O valor exato seria $0 / 0$.

1.9)

a) ($x_1 = -0.02$; $x_2 = 62.08$) (ambos pela equação normal).

b) ($x_1 = -0.01611$ (pela equação racionalizada); e $x_2 = 62.08$ (pela equação normal).

c) em (a) o erro relativo de -0.02 é 0.2417, e de 62.08 é 0.00006282;
em (b) o erro relativo de -0.01611 é 0.0001738, e de 62.08 é 0.00006282.

(1.10)

i	$x=10^{-i}$	$f(x)=1-\cos(x)$	$g(x)=sen(x)*sen(x)/(1+\cos(x))$
1	0.1	0.004995835	0.004995835
2	0.01	4.99996e-05	4.99996e-05
3	0.001	5e-07	5e-07
4	0.0001	5e-09	5e-09
5	0.00001	5e-11	5e-11
6	0.000001	5.00044e-13	5e-13
7	0.0000001	4.996e-15	5e-15
8	0.00000001	0	5e-17
9	0.000000001	0	5e-19
10	1e-10	0	5e-21

Observe que a forma $g(x)=sen(x)*sen(x)/(1+\cos(x))$, apesar de envolver mais operações, fornece resultados mais exatos do que na forma $f(x)=1-\cos(x)$, pois na $g(x)$ não ocorre a perda de significação pela subtração de parcelas quase iguais.

(1.11)

a)

```
%em Octave
n=10
x=1/n
printf("\n x=%.30f\n",x)
for i=1:100
    i
    x=(n+1)*x-1
printf("\n x=%.30f\n",x)
end%i
```

Por exemplo, para $n=3$, o valor inicial de $x=1/3=0.333333333$ (valor arredondado com 10 dígitos totais em uma calculadora científica, sem usar os dígitos internos de guarda) muda completamente em 14 repetições (destacada em cinza):

i	x
0	0.333333333
1	0.333333332
2	0.333333328
3	0.333333312
4	0.333333248
5	0.333332992
6	0.333331968
7	0.333327872
8	0.333311488

9	0.333245952
10	0.332983808
11	0.331935232
12	0.327740928
13	0.310963713
14	0.243854851
15	-0.024580598

b) Valores de $n = 2,\ 10 \text{ e } 16$ geram sequências exatas em calculadoras científicas decimais, mas apenas $n = 2 \text{ e } 16$ geram sequências exatas em computadores com armazenamento em registros binários. Para $n = 10$, temos $x = 1/10 = 0.1$, exato em decimal, mas em computadores binários temos um valor de x com arredondamento, como vimos no **Exemplo 1.29**. Esse arredondamento inicial é ampliado a cada repetição, pois é multiplicado por $(n+1)$, aumentando os erros e desestabilizando os resultados. O mesmo ocorre em computadores para $n = 3$.

(1.12)

a) (Erro absoluto = 1102.345-1100.9 =1.445);
(Erro relativo = (1102.345-1100.9)/1100.9=0.001313);
(Erro percentual = (1102.345-1100.9)/1100.9*100%=0.1313%).

b) (Erro absoluto = 0.01245-0.01190=0.00055);
(Erro relativo =(0.01245-0.01190)/0.01190=0.04622);
(Erro percentual = (0.01245-0.01190)/0.01190*100%=4.622%).

c) Em (a) o Erro absoluto é o mais alto, porém representa apenas 0.1313% do valor exato; em (b) o Erro absoluto é o mais baixo, porém representa um erro maior do que em (a), pois o valor aproximado corresponde a 4.622% do valor exato.

(1.13)

a) Dígitos significativos da parcela arredondada $(0.11001100...)_2$ que foi aproximada para cima gerando s, e, f:

1	10000010	01000001100110011001101

b)

$VA = (-10.05000019)_{10}$;

$VE = (-10.05000000)_{10}$;

$Erro\ percentual = 1.897856\text{e}-06\%$.

(1.14)

$(-5.877473\text{e}-39)$.

(1.15)

Dígitos significativos da parcela arredondada $(0.1010.....)_2$ que foi aproximada para cima:

1	11111010 0	01101011100001001111000

(1.16)

a) Dígitos significativos da parcela arredondada $(0.110111001100000...)_2$ que foi aproximada para cima;

1	00000000	00000000001111010101010

b)

$VA = (-1.100019294\text{e}-41)_{10}$;

$VE = (-1.100000000\text{e}-41)_{10}$;

$Erro\% = 1.754044998\text{e}-03\%$.

(1.17)

Executaria o algoritmo em computador:

a) com as variáveis 32 *bits* disponíveis e obteria a solução aproximada VA;
b) com as variáveis 64 *bits* disponíveis e obteria uma solução aproximada mais próxima da exata, que pode ser considerada a solução exata estimada VE; e
c) depois, calcularia o erro estimado $|VA - VE|$.

(1.18)

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int fat(int n) //funcaofatorialrecursiva
{
       if (n)
```

```
       return n*fat(n-1);
       elsereturn 1;
}

//Algoritmo com simulação usando variaveis de 32 bits, float
int main()
{
floatxs=-0.111,yas;
doublexd=xs,ye,yad,ErroArredRelativo;
inti,n=4;
ye=exp(xs);
//Calculo exp(x) via serie de Taylor em 32 bits, float:
yas=1.;
for(i=1;i<=n;i++)
   {
yas=yas+pow(xs,i)/fat(i);
}
//Calculo exp(x) via serie de Taylor em 64 bits, double:
yad=1.;
for(i=1;i<=n;i++)
   {
yad=yad+pow(xd,i)/fat(i);
    }
printf("\nn=%d, x=%f, yas=%.20lf, yad=%.20lf\n",n,xs,yas,yad);
ErroArredRelativo=fabs((yas-yad)/yad)*100.;
printf("\nErro de Arredondamento Relativo percentual=%.20lf\n",ErroArredRelativo);
    //Resultados mostram 8 digitos exatos
}
```

(1.19)

```
% Algoritmo para calcular o erro de truncamento estimado em Octave, usando variaveisdouble
formatlong;
   x=-0.111
   n=4;
ye=exp(x)
%Calculo aproximado de exp(x) via serie de Taylor em 64 bits com n=4 parcelas:
xd=x
yan=1.;
fori=1:n
yan=yan+power(xd,i)/factorial(i);
end%for
yan
%Calculo aproximado de exp(x) via serie de Taylor em 64 bits com 2n=8 parcelas, para servir de valor exato estimado:
xd=x
ya2n=1.;
for i=1:2*n
ya2n=ya2n+power(xd,i)/factorial(i);
end%for
ya2n
Errorelativoestimado=abs(yan-ya2n)/ya2n
Errorelativoexato=abs(yan-ye  )/ye
```

Respostas Capítulo 2

(2.1)

a) Matriz expandida escalonada:

A=4.00000 -2.00000 1.00000 7.00000

0.00000 2.50000 4.25000 4.50000

0.00000 0.00000 -3.70000 -4.55000

Logo, sistema possível e determinado: S={ 1.29730, -0.29054, 1.22973 }.

b)

Matriz expandida escalonada:

A = 3.00000 2.00000 3.00000 7.00000

0.00000 0.16667 2.75000 3.33333

0.00000 0.00000 -0.00000 -0.00000

Logo, sistema possível e indeterminado: S={ 11.00000, 20.00000, 0.00000 } para $x_3 = 0$.

c)

Matriz expandida escalonada:

A=3.00000 2.00000 3.00000 8.00000

0.00000 0.16667 2.75000 2.66667

0.00000 0.00000 -0.00000 -1.00000

Logo, sistema impossível: S={ NaN, NaN, NaN }.

(2.2)

a)

A=3.00000 0.10000 -1.00000 -3.00000

0.00000 2.46667 2.33333 0.00000

0.00000 0.00000 6.25676 11.00000

S={ -0.35853, -1.66307, 1.75810 }

Resíduos={ 4.4409e-16, 0.0000e00, 0.0000e+00 }.

b)

A=4.00000 -2.00000 1.00000 10.00000

0.00000 3.50000 0.50000 -6.00000

0.00000 0.00000 3.30714 -1.18571

Nova ordem das incógnitas ={ 3, 2, 1 }

Xa={ 1.75810, -1.66307, -0.35853 } %solução auxiliar, incógnitas pivotadas.

S= { -0.35853, -1.66307, 1.75810 } %solução final na ordem correta.

Resíduos={ 0, 0, 0 }.

c)

Realize o escalonamento e calcule o determinante via produto da diagonal principal da matriz triangularizada:

A =1.00000 2.50000   2.00000   -1.00000

    0.00000 -4.50000   2.00000   11.00000

    0.00000   0.00000 -10.28889 -18.08889

determinante=46.300.

Observação: o módulo do determinante independe da aplicação ou não de processos de pivotação.

(2.3)

```
function det=fdeterminante(n,A)
        for k=1:n-1
                for i=k+1:n
                        aux=A(i,k)/A(k,k);
                        for j=k+1:n+1 %operações com mínimo de colunas
                                A(i,j)=A(i,j)-aux*A(k,j);
                        end %for j
                        A(i,k)=0;%valor exato, calculado algebricamente
                end %for i
        end %for k
        det=1.;
        for i=1:n
        det=det*A(i,i); %produto da diagonal principal
        end
end %function
```

(2.4)

a)

A =3.00000 1.50000   4.75000   8.00000

    0.00000 -1.00500 -3.34917 -3.69333

    0.00000 0.00000 -1.63333 -3.66667

S={ 1.0153, -3.8062,   2.2449 }.

Resíduos={ 0.0000e+00, 2.6645e-15, 4.4409e-16 }.

b)

A =4.01000 1.00000   3.00000   7.00000

     0.00000 0.75187   2.50561   2.76309

     0.00000 0.00000 -1.63333 -3.66667

S={ 1.0153, -3.8062,   2.2449 }.

Resíduos={ 1.7764e-15, 0.0000e+00, 0.0000e+00} .

c)

A =4.75000 3.00000   1.50000   8.00000

    0.00000 2.11526 0.05263   1.94737

0.00000 0.00000 0.49012 -1.86549

Nova ordem das incógnitas={ 3, 1, 2 }.

Xa={ 2.2449, 1.0153, -3.8062 }.

S={ 1.0153, -3.8062, 2.2449 }.

Resíduos={ 8.8818e-16, 0.0000e+00, 2.2204e-16}.

d)

A^{-1} =-0.31475 0.49751 -0.05077

0.64991 -0.99502 2.04031

0.20408 -0.00000 -0.61224

S={1.0153, -3.8062, 2.2449 }.

Resíduos={1.7764e-15, 8.8818e-16, 0.0000e+00 }.

(2.5)

```
clear
clc
n=3
A =[1   33  30 ;
    40  11  70 ;
    22  50  104 ;]
A1=finversa(n,A)
Afericao=A1*A-eye(n) %deve gerar a matriz nula
```

```
function A1=finversa(n,A)
       A=[A eye(n)] %matriz aumentada: A concatenada com eye (identidade)
       for k=1:n
% Escolhe a melhor linha i=k, antes de efetuar a eliminacao
% retorna a matriz expandida A com linhas trocadas.
              [A]=fpivotacaoparcial(k,n,A);
              A(k,k+1:2*n)=A(k,k+1:2*n)/A(k,k);A(k,k)=1.;%normalização
              for i=1:k-1  %linhas acima de k
                     aux=A(i,k);
                     for j=k+1:2*n %operações com mínimo de colunas
                            A(i,j)=A(i,j)-aux*A(k,j);
                     end %for j
                     A(i,k)=0;%valor exato, calculado algebricamente
              end %for i
              for i=k+1:n  %linhas abaixo de k
                     aux=A(i,k);
                     for j=k+1:2*n %operações com mínimo de colunas
                            A(i,j)=A(i,j)-aux*A(k,j);
                     end %for j
                     A(i,k)=0;%valor exato, calculado algebricamente
              end %for i
              end %for k
A1=A(:,n+1:2*n);%toma apenas os elementos da 2a.parte da matriz, onde estava I
end %function
```

```
function [A]=fpivotacaoparcial(k,n,A)
%permutacao de linhas para escolher a melhor linha i=k(maior modulo na diagonal
principal)
Amax=abs(A(k,k));
imax=k; %linha que contem o maior modulo
for i=k+1:n
    if abs(A(i,k))>Amax
       Amax=abs(A(i,k));
       imax=i;
    end
end
```

```
aux=A(k,:);A(k,:)=A(imax,:);A(imax,:)=aux;
end
```

(2.6)

L = 4.0000000.000000 0.000000
-1.0000004.0000000.000000
1.0000003.0000001.000000

U = 1.000000-0.250000 0.250000
0.0000001.0000000.750000
0.0000000.0000001.000000

C = { 0.0000000, 0.250000, -1.750000 }

X = { 0.8281250, 1.562500, -1.750000 }

(2.7)

L = 2.0000000 0.0000000 0.0000000
-0.5000000 2.0000000 0.0000000
0.5000000 1.5000000 1.0000000

U = 2.0000000 -0.5000000 0.5000000 (transposta de L)
0.0000000 2.0000000 1.5000000
0.0000000 0.0000000 1.0000000

C = { 0.0000000, 0.5000000, -1.7500000 }

X = { 0.8281250, 1.5625000, -1.7500000 }

Observe que a decomposição de Choleky gera a matriz, L e $U=L^T$, diferentes das L e U de Crout, mas apresentam a mesma solução final X.

(2.8)

Com n=10 equações, temos $2/3*(10^3)$ operações envolvidas; e, com n=1000 equações, teremos $2/3*(1000^3)$ operações. Como o tempo de processamento necessário é diretamente proporcional ao número de operações aritméticas, então o tempo total para n=1000 é $(1000^3)/(10^3)*0.1$ segundos = 100000 segundos $\cong$ 28 horas.

(2.9)

Pelo determinante da matriz escalonada:

A=4.01000 2.00000 3.00000 7.00000
0.00000 0.00374 2.50561 2.76309

0.00000 0.00000 -1.63333 -3.66667

α_1 =1.1192 (calculado sobre a matriz original)

α_2 =5.8149 (“)

α_3 =5.3926(“)

det(A)=-0.024500 (calculado sobre a matriz simplificada).

Determinante Normalizado (A)= 6.9814e-004<0.01 -> mal condicionado.

Pelo número de condição Cond(A):

A =1.000000 0.500000 -0.050000

3.000000 1.500000 4.750000

4.010000 2.000000 3.000000

Soma dos módulos de cada linha de A ={ 1.5500, 9.2500, 9.0100}

$|| A ||_\infty = 9.25$

A^{-1} =-204.08163 -65.30612 100.00000

410.10204 130.63265 -200.00000

-0.61224 0.20408 -0.00000

Soma dos módulos de cada linha de A^{-1} ={ 369.38776, 740.73469, 0.81633 }

$|| A^{-1} ||_\infty = 740.73469$.

$Cond(A) = || A ||_\infty * || A^{-1} ||_\infty = 6851.8 > 100 \Rightarrow$ Mal condicionado.

(2.10)

Evitar ao máximo as alterações de coeficientes ao longo dos métodos numéricos devido aos arredondamentos:

a) usar precisão (número de dígitos significativos) alta (variáveis *double*, por exemplo);

b) evitar operações aritméticas desnecessárias (algoritmo otimizado); e

c) usar pivotação parcial ou total em métodos eliminativos/diretos.

(2.11)

Sim, seria indicado usar métodos iterativos, se possível, pois os coeficientes utilizados nesses métodos são sempre os originais, ou seja, não são alterados por arredondamentos. As equações iterativas são apenas reescritas com as incógnitas x(i) isoladas.

(2.12)

*function AA=fescalona(n,A), da seção*2.2.1.

(2.13)

a) Tempo=(2/3)*$(1000)^3$ operações/10^6 operações/segundo = 667 segundos.

b) Memória=1000*1001 coeficientes * 8B = $8*10^6$ B ≅ 8MB.

(2.14)

a)

LU = 2.00000 -0.50000 0.50000

1.00000 -1.50000 -0.33333

1.00000 0.40000 -1.36667

C={ -0.50000, -1.00000, -2.85366 }.

X={ -0.048780, -1.95122, -2.853659 }.

b)

Resíduos ≅ { 1e-06, 1e-06, 1e-06 } compatíveis com a precisão adotada 7 dígitos totais).

(2.15)

```
%Matriz de coeficientes de ordem n x n
n=3 % numero de equaçoes
A=[     4  1   2 ;
        1 -2   1 ;
        1 0.1  -1;]
%Matriz de m vetores de termos independentes b:
m=4%Equivale a 4 sistemas diferentes, com a mesma matriz A e
% 4 vetores b de termos independentes concatenados e
% armazenados nas 4 colunas de B;
B=[     1  -1  7  5 ;
        4  10 -4  3 ;
      -3  -3   3  1 ;]
     [LU]=fLUCrout(n,A) %Determina L e U sobrepostas na mesma matriz
% Resolvendo os m sistemas, para o termo independente t=1:m
for t=1:m
for i=1:n
        b(i)=B(i,t);   % Definindo um unico vetor b
end
   t
   x=fsubstituicao2(n,LU,b)  %determina c e x
end %t
```

```
function x=fsubstituicao2(n,A,b)
%L.c=b
        c(1)=b(1)/A(1,1);
        for i=2:n
        soma=0;
                for j=1:i-1
```

```
                soma=soma+A(i,j)*c(j);
                end
            c(i)=(b(i)-soma)/A(i,i);
        end
        c
%U.x=c
        x(n)=c(n);
        for i=n-1:-1:1
        soma=0;
                for j=i+1:n
                soma=soma+A(i,j)*x(j);
                end
        x(i)=(c(i)-soma);
        end
end
```

(2.16)

X = { 2.1000, -2.9000, -10.6000, -8.3000, 5.3000 }

(2.17)

```
    n=5
    t=[ 0  1   1   1   1 ]; %faixa à esquerda da principal
    r=[ 1  3  -2  -1   1 ]; %diagonal principal
    d=[-1 -1   1   1   0 ]; %faixa à direita da principal
    b=[ 5  4  10   3  -3 ]; %termo independente
%Escalonamento:
        for i=2:n
                aux=t(i)/r(i-1);
                r(i)=r(i)-aux*d(i-1);
                b(i)=b(i)-aux*b(i-1);
                t(i)=0;
        end%fori
%Retrosubtituição
        x(n)=b(n)/r(n);
        for i=n-1:-1:1
                x(i)=(b(i)-d(i)*x(i+1))/r(i);
        end%for i
x
```

(2.18)

a) Não, pois na linha i=1 -> |1|>=|2|+|1| F (falso).

b) Trocamos as equações, através das linhas da matriz, de modo que o maior coeficiente de cada coluna fique na diagonal principal:

$$\begin{cases} 4x_1 + x_2 + 2x_3 = 1 \\ x_1 + 2x_2 + x_3 = 4 \\ x_1 + 0.1\ x_2 + x_3 = -3 \end{cases}$$

c) Não, pois:

na linha i=1	-> |4|>=|1|+|2|	V
na linha i=2	-> |2|>=|1|+|1|	V
na linha i=3	-> |1|>=|1|+|0.1|	F

Mas esse sistema pivotado ficou quase com diagonal dominante e possivelmente convirja.

d) Depois de seis iterações:

X={ 2.0324, 3.6840, -5.4008 }, $Max\left|x_i^{k+1} - x_i^k\right| \leq 3.4248e-04$.

e)

det(A)=3.8000, $\alpha_1 = 2.4495$, $\alpha_2 = 1.4177$, $\alpha_3 = 4.5826$ (calculados na matriz original),

Determinante Normalizado (A)=0.23878.

Logo, não é um sistema mal condicionado.

(2.19)

```
% Algoritmo de matriz penta diagonal:
n=6%(para n>=6)
%a(i)*x(i-2)+b(i)*x(i-1)+c(i)*x(i)+d(i)*x(i+1)+e(i)*x(i+2)=f(i); p/ todo i
        a=[ 0  0   1  -1  -1  2  ]; %2a. diagonal à esquerda da principal
        b=[ 0  1   1  -1  -1  3  ]; %1a. diagonal à esquerda da principal
        c=[ 3 -4   6   8   9  5  ]; %    diagonal principal
        d=[-1 -1   1   1   5  0  ]; %1a. diagonal à direita da principal
        e=[-1 -1   1   6   0  0  ]; %2a. diagonal à direita da principal
        f=[ 2  1   2  -1  -2  5  ]; %termos independentes
%Escalonamento:
for k=1:n-2
        i=k+1;
                aux=b(i)/c(k);
                b(i)=0; %opcional
                c(i)=c(i)-aux*d(k);
                d(i)=d(i)-aux*e(k);
                %e(i)=cte;
                f(i)=f(i)-aux*f(k);
        i=k+2;
                aux=a(i)/c(k);
                a(i)=0; %opcional
                b(i)=b(i)-aux*d(k);
                c(i)=c(i)-aux*e(k);
                %d(i)=cte;
                %e(i)=cte;
                f(i)=f(i)-aux*f(k);
end
k=n-1;
        i=k+1;
                aux=b(i)/c(k);
                b(i)=0; %opcional
                c(i)=c(i)-aux*d(k);
                d(i)=d(i)-aux*e(k);
                %e(i)=cte;
                f(i)=f(i)-aux*f(k);

%Retrosubstituicoes:
        x(n)=f(n)/c(n);
        x(n-1)=(f(n-1)-d(n-1)*x(n))/c(n-1);
        for i=n-2:-1:1
                x(i)=(f(i)-d(i)*x(i+1)-e(i)*x(i+2))/c(i);
        end
x
%Aferição com os n residuos:
residuos(1)=abs(                        +c(1)*x(1)+d(1)*x(1+1)+e(1)*x(1+2)-f(1));
residuos(2)=abs(           +b(2)*x(2-1)+c(2)*x(2)+d(2)*x(2+1)+e(2)*x(2+2)-f(2));
for i=3:n-2
residuos(i)=abs(a(i)*x(i-2)+b(i)*x(i-1)+c(i)*x(i)+d(i)*x(i+1)+e(i)*x(i+2)-f(i));
end
residuos(n-1)=abs(a(n-1)*x(n-1-2)+b(n-1)*x(n-1+1)+c(n-1)*x(n-1)+d(n-1)*x(n-1+1)-f(n-1));
residuos(n  )=abs(a(n  )*x(n-2  )+b(n  )*x(n-1  )+c(n  )*x(n  )                -f(n  ));
residuos
rmax=max(residuos)
```

(2.20)

a) Esse sistema não tem convergência garantida, pois não tem diagonal dominante. Todos os módulos das diagonais são iguais à soma dos módulos dos coeficientes vizinhos.

b)

k	x_1	x_2
0	0	0
1	4	-2
2	6	2
3	2	4
4	0	0
5	4	-2
6	6	2
7	2	4
8	0	0

A solução entra em um ciclo repetitivo e não converge.

c)

k	x_1	x_2
0	0	0
1	4	2
2	2	0
3	4	2
4	2	0
5	4	2
6	2	0

A solução também entra em um ciclo repetitivo e não converge.

d) Como os valores de x_i oscilam de uma iteração para outra, possivelmente a aplicação de um fator de sub-relaxação vai atenuar a amplitude de oscilação e deve conduzir à convergência.

e) Testando fatores de sub-relaxação entre 0.5 e 0.9, chegaremos a um valor otimizado 0.8 que leva à convergência no menor número de iterações (≤10):

k	x_1	x_2
0	0	0
1	3.2	0.96
2	3.072	1.0496
3	2.97472	0.989696
4	3.003187	1.000489
5	3.000246	1.000295
6	2.999813	0.99991

7	3.000035	1.00001
8	2.999999	1.000001
9	2.999999	0.999999
10	3	1

(2.21)

a) Sim, pois tem diagonal dominante:

na linha i=1 $\Rightarrow$ |1|>=|1| V (verdadeiro)

na linha i=2:n/2 $\Rightarrow$ |9|>=|1|+|1|+|1| V

na linha i=n/2+1:n-1 $\Rightarrow$ |9|>=|1|+|1|+|1| V

na linha i=n $\Rightarrow$ |1|>=|1| V

na linha i=2:n/2 $\Rightarrow$ |9|> |1|+|1|+|1| V.

b) Sim, é sempre importante testar fatores de relaxação, mas não é mandatório, uma vez a matriz tem diagonal fortemente dominante (maioria das linhas tem módulo da diagonal principal três vezes a soma dos módulos dos demais coeficientes). Assim já deve ter convergência rápida, mas provavelmente o uso de fatores de sobrerrelaxação vai acelerar o processo ainda mais.

c) Usamos o método de Gauss-Seidel com fator de sobrerrelaxação, pois acelera a convergência:

```
%Algoritmo de Gauss-Seidel:
clear
clc
n=1000
f=1.3
f1=1-f;
for i=1:n
  x(i)=0;
end%for
k=0;
x(1);
dif=1;
while(dif>1.e-10  && k<1000 )
        k=k+1;
        xi=x;
        i=1;x(i)=f1*x(i)+f*(150-x(i+1));
        for i=2:n/2
                x(i)=f1*x(i)+f*(100-x(i-1)-x(i+1)-x(i+100))/9;
        end%for
        fori=n/2+1:n-1
                x(i)=f1*x(i)+f*(200-x(i-100)-x(i-1)-x(i+1))/9;
        end%for
        i=n;x(i)=f1*x(i)+f*(300-x(i-1));
        x(1);
dif=max(abs((x.-xi)./x));
end%while
k
dif
x1=x %VA - valor aproximado

%Erro de truncamento (OPCIONAL):
```

```
while(dif>1e-15 && k<1000 )
        k=k+1;
        xi=x;
        i=1;x(i)=f1*x(i)+f*(150-x(i+1));
        for i=2:n/2
                x(i)=f1*x(i)+f*(100-x(i-1)-x(i+1)-x(i+100))/9;
        end%for
        fori=n/2+1:n-1
                x(i)=f1*x(i)+f*(200-x(i-100)-x(i-1)-x(i+1))/9;
        end%for
        i=n;x(i)=f1*x(i)+f*(300-x(i-1));
        x(1);
dif=max(abs((x.-xi)./x));
end%while
k
dif
x2=x;%VE - valor exato estimado
ErroTrunc=max(abs((x1.-x2)./x2))
```

(2.22)

a) Sim, pois tem diagonal dominante:

na linha i=1 -> |1|>=|-1| V

na linha i=2:n_1-1 -> |2|>=|-1|+|-1| V

na linha i=n_1:n_2-1 -> |3|>=|-1|+|-1|+|-1| V

na linha i=n_2 -> |2|>=|-1| V

na linha i=n_2 -> |2|>|-1| V.

b) A matriz de coeficientes é pouco dominante (apenas um linha tem diagonal principal maior do que a soma dos coeficientes vizinhos (em módulos), então é recomendável testar fatores de relaxação para tentar acelerar o processo de convergência. Se a convergência for lenta, recomendamos usar fatores de sobrerrelaxação, que ampliam o incremento das incógnitas em cada iteração.

c) Se a convergência for oscilatória, recomendamos usar fatores de sub-relaxação, que reduzem o incremento das incógnitas em cada iteração.

d) S={ 2.0000; 1.9000; 1.7000; 1.0000 }.

e) O valor inicial não está definido, então escolhemos o valor nulo:

k	x_1	x_2	x_3	x_4	$Max \mid x_i^n - x_i^{n-1} \mid$
0	0	0	0	0	-
1	0.10000	0.10000	0.13333	0.21667	0.21667
2	0.20000	0.21667	0.27778	0.28889	0.14444
3	0.31667	0.34722	0.38426	0.34213	0.13056

Podemos parar com poucas iterações e apresentar uma solução com o critério atingido, por exemplo:

S={ 0.31667; 0.34722; 0.38426; 0.34213 } com critério de parada $Max \mid x_i^n - x_i^{n-1} \mid = 0.13056$.

Para atingir 10 dígitos significativos, precisaríamos de 268 iterações:

x = { 2.00000, 1.90000, 1.70000, 1.00000 }

$Max\,|\,x_i^n - x_i^{n-1}\,| = 9.8225\mathrm{e}-11.$

(2.23)

a) Sim, pois tem diagonal dominante. Todas as linhas têm módulo da diagonal maior ou igual à soma dos módulos coeficientes vizinhos e várias linhas têm diagonais maiores.

b) Se a convergência for oscilatória, recomendamos usar fatores de sub-relaxação, que reduzem o incremento das incógnitas em cada iteração.

c) Se a convergência for lenta, recomendamos usar fatores de sobrerrelaxação, que ampliam o incremento das incógnitas em cada iteração e podem acelerar a convergência.

d)

```
function x=fTrid(n,t,r,d,b)
%Escalonamento
for i=2:n
        aux=t(i)/r(i-1);
        r(i)=r(i)-aux*d(i-1);
        b(i)=b(i)-aux*b(i-1);
        t(i)=0;
end%for
%Retrosubtituição
x(n)=b(n)/r(n);
for i=n-1:-1:1
        x(i)=(b(i)-d(i)*x(i+1))/r(i);
end%for
end
```

e)

```
function x=fGaussSeidelrelax(n,t,r,d,b,fator,tolerancia)
dif=1; iter=0;fator1=1-fator;
for i=1:n
xa(i)=0.; x(i)=xa(i);
end%for
while dif>tolerancia && iter<500
iter=iter+1;
x(1)=fator1*xa(1)+fator*(b(1)-d(1)*x(2))/r(1);
for i=2:n-1
  x(i)=fator1*xa(i)+fator*(b(i)-t(i)*x(i-1)-d(i)*x(i+1))/r(i);
end%for
x(n)=fator1*xa(n)+fator*(b(n)-t(n)*x(n-1))/r(n);
dif=max(abs((x.-xa)./x));
xa=x;
end%while
end
```

f)

```
n1=30;
n2=50;
i=1;           t(i)= 0;r(i)=2;d(i)=-1;b(i)=-1;
for i=2 :n1-1  t(i)=-1;r(i)=3;d(i)=-1;b(i)= 1; end
for i=n1:n2-1  t(i)=-1;r(i)=3;d(i)=-1;b(i)= 2; end
i=n2;          t(i)=-1;r(i)=1;d(i)= 0;b(i)= 3;
n=n2;
fator=1.2;
tolerancia=1.e-6;
xe=fTrid(n,t,r,d,b); %Solução exata obtida por algoritmo tridiagonal
```

```
x=fGaussSeidelrelax(n,t,r,d,b,fator,tolerancia); % solução aproximada x, com criterio 1e-6
x2=fGaussSeidelrelax(n,t,r,d,b,fator,tolerancia^2); % solução mais exata que x, com criterio 1e-12
Erro1=max(abs((x.-xe)./xe))
Erro2=max(abs((x.-x2)./x2))
%Os dois Erros são equivalentes: A forma de calculo de uma solução mais exata com criterio (tolerancia^2) gera uma solução com o dobro de dígitos exatos.
```

Respostas Capítulo 3

(3.1)

a) $x(2) \in [\pi,\ 1.3\pi]$

b) $x(2) = 3.425618459$ e critério $|\Delta x| < 1e-09$

c)

```
%main.m
% Primeira fase: Localização da raiz inicial
%Resultado de uma análise gráfica: 2o. valor inicial, positivo
xI = [ pi ]
tolerancia = 1e-15;
% Segunda fase: Aproximação do valor da raiz com erro controlado
    [x Erro] = NewtonNumerico(xI, tolerancia);
'2a. raiz =' x
'Erro<=' dif
```

```
function [x dif]= NewtonNumerico(x, tolerancia)
% método da secante (Newton com derivada numérica)
    dif = 1;
    k = 0;
    limite = 50;
    delta_x = 1e-6;
while (dif > tolerancia && k < limite)
        k++;
% Núcleo do método
        derivadaNumerica = (f(x + delta_x) - f(x)) / delta_x;
        delta_x = -f(x)/derivadaNumerica;
        x = x+delta_x;
% Núcleo do método
dif = abs(delta_x);
    end
end
```

```
function y = f(x)
    y = x.*tan(x).-1;
end
```

(3.2)

Bisseção:

50ª partição:

$a = 1.0717734625362922$

$b = 1.0717734625362940$

$\overline{x} = 1.0717734625362931$ – raiz encontrada depois de 50 iterações

$|b - a| = 0.0000000000000018$

$|f(x)| = 0.0000000000000007$

Falsa posição:

2027ª partição:

$a = 1.0717734625362345$

$b = 2.0000000000000000$ _ b ficou fixo

$\overline{x} = 1.0717734625362354$ – raiz encontrada somente depois de 2.027 iterações

$|\overline{x} - \overline{x}_0| = 0.0000000000000009$

$|f(\overline{x})| = 0.0000000000010780$

Falsa posição modificado:

17ª partição:

$a = 1.0717734625362922$

$b = 1.0717734626940287$ – observe que b foi destravado.

$\overline{x} = 1.0717734625362931$ – raiz encontrada depois de apenas 17 iterações.

$|\overline{x} - \overline{x}_0| = 0.0000000000000009$

$|f(\overline{x})| = 0.0000000000000007$

Entre os três métodos de quebra vistos, o método de falsa posição modificado é o mais eficiente, pois converge com menos passos iterativos e operações aritméticas.

Newton:

6ª iteração:

$x = 1.0717734625362931$ – raiz encontrada depois de 6 iterações.

$|x - x_0| = 0.0000000000000002$

$|f(x)| = 0.0000000000000007$

O método de Newton é mais eficiente do que os métodos de quebra, pois converge com menos passos iterativos e operações aritméticas.

(3.3)

Partindo de um valor inicial x_k, definimos o próximo valor $x_{k+1} = x_k + \Delta x$ e sua função expandida em série de Taylor:

$$f(x_{k+1}) = f(x_k + \Delta x) = f(x_k) + f'(x_k)\frac{\Delta x}{1!} + f''(x_k)\frac{\Delta x^2}{2!} + \cdots$$

Fazendo $f(x_{k+1}) = f(x_k + \Delta x) = 0$, temos

$$f(x_k) + f'(x_k)\frac{\Delta x}{1!} + f''(x_k)\frac{\Delta x^2}{2!} + \cdots = 0$$

Para determinar Δx, Newton de 1ª ordem, tradicional, assumimos que x_k está em um intervalo suficientemente próximo da solução desejada e que Δx é suficientemente pequeno para que possamos representar a função $f(x)$

apenas pelos seus dois primeiros termos da série de Taylor, truncando a partir do termo de 2ª ordem:

$$f(x_k) + f'(x_k)\frac{\Delta x}{1!} = 0$$

resolvendo a equação linear:

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

No método de Newton de 2ª ordem, podemos representar a função $f(x)$ apenas pelos seus três primeiros termos da série de Taylor, truncando a partir do termo de 3ª ordem:

$$f(x_k) + f'(x_k)\frac{\Delta x}{1!} + f''(x_k)\frac{\Delta x^2}{2!} = 0$$

Determinamos Δx resolvendo a equação de 2º grau modificada (para gerar menos arredondamentos), de modo a gerar o menor Δx (com maior denominador) e dar mais estabilidade ao longo do processo de convergência:

$$\Delta x = -\left[\frac{2f(x_k)}{f'(x_k) + sign(f'(x_k))\sqrt{(f'(x_k))^2 - 4(f''(x_k)/2)f(x_k)}}\right]$$

Newton de 1ª ordem clássico

$x_7 = 1.07177346253629$ em $k = 7$ iterações

Critério $dif = 5.94953787645010e-17$

Newton de 2ª ordem:

$x_4 = 1.07177346253629$ em $k = 4$ iterações

Critério $dif = 5.94953787645010e-17$

Logo, o método de Newton de 2ª ordem convergiu em menos iterações, atingiu a mesma solução com todos os dígitos *double*, mas não foi mais rápido, uma vez que usa mais operações aritméticas na equação de Δx, além de chamar três funções f, f' e f'' em vez de duas funções no método tradicional: f e f'.

(3.4)

Método da secante:

$x = 1.0717734625362931$ – raiz encontrada depois de 8 iterações.

$|x - x_0| = 3.57005957018434e-17$

$|f(x)| = 0.0000000000000007$

Método de Müller:

$x = 1.07177346253629$ – raiz encontrada depois de 6 iterações.

$|x - x_0| = 0.0000000000000000$

$|f(x)| = 0.0000000000000007$

(3.5)

```
% Primeira fase: Localização da(s) raiz(es) inicial(is)
%Resultado de uma análise gráfica: 5 valores iniciais positivos
tolerancia = 1e-15;
xI = [1.  3.  6.  9.  12]
% Segunda fase: Aproximação do valor da raiz com erro controlado
for i = 1 : 5
    i
    x(i) = Newton(xI(i), tolerancia);
end
x
```

```
function x = Newton(x, tolerancia)
    dif = 1;
    k = 0;
    limite = 50;
    while (dif > tolerancia && k < limite)
k++;
% Núcleo do método
        delta_x = -f(x)/derivada_f(x);
        x = x+delta_x;
% Núcleo do método
dif = abs(delta_x);
    end
    'Newton'
    k
    x
    dif
end
```

```
function y = f(x)
    y = x.*tan(x).-1;
end
```

```
function y = derivada_f(x)
    y = tan(x).+x.*sec(x).*sec(x);
end
```

(3.6)

```
%main.m
a=3.22234;
c=4.77665;
d=a*freciproco(c)
```

```
%Algoritmo de Newton com o Recíproco de C
function x=freciproco(C)
%1/C=x=>? ; f(x)=C-1/x; f'(x)=-1/(x*x)
%x=xi-f(xi)/df(xi)  ; x=x-(C-1/x)/(1/(x*x))  ;  x=x*(2-x*C);
%Valor inicial grosseiro, mas tem convergência garantida
```

```
if(C>1) x=0+eps;
else    x=1+eps;
end
x=0.001
cont=0;dif=1;
while( dif>1.e-16 && cont<100 )
   cont=cont+1
   xi=x;
   x=x*(2-x*C)
   dif=abs((x-xi)/x)
end
end
```

(3.7)

```
%Algoritmo de Newton com o Raiz de ordem N de C:
%sqrtN(C)=x=>? ; f(x)=x^N-C; f'(x)=N*x^(N-1)
%x=xi-f(xi)/df(xi); x=x-(x^N-C)/(N*x^(N-1));
%Valor inicial grosseiro, mas tem convergência garantida
if(C>1) x=1+eps;
else    x=0+eps;
end
cont=0;dif=1;
while( dif>1.e-16 && cont<100 )
        cont++;
        xi=x;
        XN1=xi; for i=2:N-1 XN1=XN1*xi; end
        XN=XN1*xi;
        x= x-(XN-C)/(N*XN1);
        dif=abs((x-xi)/x);
end
end
```

(3.8)

```
function [A B]=fLocalizaRaizReal(n,coef)
%valores limites para raizes iniciais reais dentro de [-Rmax,Rmax]:
Rmax= fmodulomax(n,coef);
h=0.01; %subintervalos de busca de raizes reais
%indice da raiz real ir
ir = 0;
% varredura de subintervalos entre [-Rmax,-Rmin]
a = -Rmax;
b = a + h;
while ( b < Rmax-h )
%
        fa=Pn(n,coef,a);
        fb=Pn(n,coef,b);
        if ( fa*fb<0 )
                ir = ir + 1;
                A(ir)=a; B(ir)=b;
        end
% define novo subintervalo de teste
        a = b;
        b = a + h;
end
end
```

(3.9)

```
function R = fDivisaoPol(n, a, u)
    % NÚCLEO BRIOT RUFFINI
        b(1) = a(1);
        for i = 2 : n+1
            b(i) = a(i) + u * b(i-1);
end
        R=b(n+1);
end
```

(3.10)

```
function [n,a] = fDivisaoPol(n, a, u, k)
    for d = 1 : k
    % NÚCLEO BRIOT RUFFINI
```

```
        b(1) = a(1);
        for i = 2 : n+1
            b(i) = a(i) + u * b(i-1);
        end
        n = n-1; a = b;
    % FIM NÚCLEO
end
    aux = a(1:n+1); a = aux;
end
```

(3.11)

```
function r = fRestos(n, a, u)
    nI = n;
    for d = 1 : nI
        % NÚCLEO BRIOT RUFFINI
        b(1) = a(1);
        for i = 2 : n+1
            b(i) = a(i) + u * b(i-1);
        end
        r(d) = b(n+1);
        n = n-1; a = b;
        % FIM NÚCLEO
    end
    r(nI+1) = a(1);
end
```

(3.12)

```
function derivadak = fDivisaoPol(n, a, u, k)
%k<n
    for d = 1 : k+1
    % NÚCLEO BRIOT RUFFINI
        b(1) = a(1);
        for i = 2 : n+1
            b(i) = a(i) + u * b(i-1);
        end
        R(d)=b(n+1);
        n = n-1;a = b;
    % FIM NÚCLEO
end
derivadak=factorial(k)*R(k+1);
end
```

(3.13)

```
function M = fMultiplicidade(R)
    Rlimite = 0.1;
    M = 1;
    somaRestos = abs(R(1)) + abs(R(2));
    while somaRestos < Rlimite
        M = M + 1;
        somaRestos = somaRestos + abs(R(M+1));
    end
end
```

(3.14)

a)

$P_4(x) = +1x^4 + 4x + 1 = 0$

+ + + = 0 trocas: Número Raízes positivas $\leq$ 0.

$P_4(-x) = +1x^4 - 4x + 1 = 0$

+ - + = 2 trocas: Número Raízes negativas $\leq$ 2.

Pela regra de Descartes (nulas=0):

P	N	C
0	2	2

0 0 4
b)

	Módulo máximo	Cauchy	Kojima
raioMin =	0.200000000000000	0.249088379839461	0.200000000000000
raioMax =	5.00000000000000	1.66325193877147	2.58740105196820

c) Não temos certeza da existência de raízes reais, então adotaremos raízes iniciais complexas:
raio min = 0.249088379839461
raio max = 1.66325193877147
raio médio = 0.956170159305465
componente real = 0.676114403613116,
componente imaginária = 0.676114403613116
xi(1) = 0.676114403613116 + 0.676114403613116*i*
xi(2) = 0.676114403613116 - 0.676114403613116*i*
xi(3) = -0.676114403613116 + 0.676114403613116*i*
xi(4) = -0.676114403613116 - 0.676114403613116*i*

d)
x(1) = -0.250992157
x(2) = -1.493358557

(3.15)
a)
$P_4(x) = + x^4 + x^2 + 4x + 1 = 0$
+ + + + = 0 trocas: Número Raízes positivas $\leq$ 0.
$P_4(x) = + x^4 + x^2 - 4x + 1 = 0$
+ + - + = 2 trocas: Número Raízes negativas $\leq$ 2.
Pela regra de Descartes (nulas=0):
P N C
0 2 2
0 0 4
b)

	Módulo máximo	Cauchy	Kojima
Raio Min =	0.200000000000000	0.235409152887475	0.200000000000000
Raio Max =	5.00000000000000	1.85614161546732	2.58740105196820

Maior Raio min = 0.235409152887475
Menor Raio max = 1.85614161546732
c) Não temos certeza da existência de raízes reais, então adotaremos complexas:
raio médio = 1.04577538417740
componente real = 0.739474865749805,
componente imaginária = 0.739474865749805

xi(1) = 0.739474865749805 + 0.739474865749805*i*
xi(2) = 0.739474865749805 - 0.739474865749805*i*
xi(3) = -0.739474865749805 + 0.739474865749805*i*
xi(4) = -0.739474865749805 + 0.739474865749805*i*
d)
x(1) = -0.269472035
x(2) = -1.249380557656920 - 0.000000000000000*i*
e)
x(2) = -1.249380557656920

(3.16)

a) $P_4(x) = +x^4 + x^3 - x^2 - x^1 + 0.1 = 0$

b)

$P_4(x) = +x^4 + x^3 - x^2 - x^1 + 0.1 = 0$

+ + - - + = 2 trocas: ≤ Raízes positivas ≤ 2.

$P_4(-x) = +x^4 - x^3 - x^2 + x^1 + 0.1 = 0$

+ - - + + = 2 trocas: ≤ Raízes negativas ≤ 2.

Pela regra de Descartes (nulas=0 em (b)):

P N C
2 2 0
0 2 2
2 0 2
0 0 4

Módulo máximo Cauchy Kojima
Raio Min =
0.0909090909090909 0.0909235415434433 0.0759746926647958
Raio
Max = 2.00000000000000 1.84909306160101 2.00000000000000

Menor Raio max = 1.84909306160101
Maior Raio min = 0.0909235415434433
Raio componente médio = 0.970008301572228
componente real = 0.685899447848968,
componente imaginária = 0.685899447848968
Como não há nenhuma garantia de ter raiz real, adotammos valores iniciais imaginários:
xi(1) = 0.685899447848968 + 0.685899447848968*i*
xi(2) = 0.685899447848968 - 0.685899447848968*i*
xi(3) = -0.685899447848968 + 0.685899447848968*i*
xi(4) = -0.685899447848968 - 0.685899447848968*i*

c)
x(4) = 0.9736322813985405

(3.17)

```
function [x M]= roots2(a)
tol = 1e-15;
```

```
    n = length(a)-1;
    nI = n
    indiceRaiz = 0;
    while n > 0
        indiceRaiz = indiceRaiz + 1
    % Primeira fase de localização das raízes:
        xI = fLocaliza(n, a)
    % Segunda fase: Aproximação da raiz por Newton-Rhapson
        [x(indiceRaiz) M(indiceRaiz) dif k] = fNR_Pol(n, a, xI, tol)
    % Terceira fase : Redução de grau para obter as demais raízes
          for k=1:M(indiceRaiz) %M divisoes sucessivas p/ redução de grau
              b=fDivBrio(n,a,xi)
              n=n-1;a=b;
          end
    end
end
```

(3.18)

a)

Módulo máximo Cauchy Kojima

Raio Min = 0.268292682926829 0.285913190372845 0.23245735194570

Raio Max = 4.63000000000000 4.23205378675289 5.20525588832577)

Maior raio Min = 0.285913190372845 e

Menor raio Max = 4.23205378675289

b)

Pela regra de Descartes:

P N C

3 0 0

1 0 2

raio médio = 2.25898348856287 deve ser a ordem de grandeza do módulo das 3 raízes iniciais:

xi(1)=+2.25898348856287

xi(2)=1.59734254335125 + 1.59734254335125*i*

xi(3)= 1.59734254335125 - 1.59734254335125*i*

c)

k	xi	R_1	R_2	R_3	R_4	$\hat{M}$	dx	x	$dif = \lvert dx \rvert + \lvert R_1 \rvert$
1	1.000	-0.001	0.030	-0.300	1	2	0.050	1.05	0.0490000
2	1.050	-0.000125	0.007500	-0.150000	1	2	0.025	1.075	0.0248750
3	1.075	-0.00001562	0.001875	-0.075000	1	3	0.025	1.1	0.02501562
4	1.100	0.00000000	0.000000	0.000000	1	3	0	1.1	0.00000000

$x = 1.1$ e $M = 3$

d) $R_1 = R_2 = R_3 = 0$, ou seja, 3 restos nulos, 3 divisões exatas de $P_3(x)$ por $(x - 1.1)$.

(3.19)

a) raio Min = 0.212598425196850 e raioMax = 3.70000000000000

b) raio Min = 0.233928973941419 e raioMax = 3.46258946188873

c) raio Min = 0.190192378864668 e raioMax = 4.25884572681199

d) A maior cota mínima raio Min = 0.233928973941419 e
a menor cota máxima raio Max = 3.46258946188873.

e) raio médio = 1.84825921791507 deve ser a ordem de grandeza do módulo das 3 raízes iniciais:

xi(1)= 1.30691662637829 + 1.30691662637829*i*

xi(2)=1.30691662637829 - 1.30691662637829*i*

xi(3)=-1.30691662637829 + 1.30691662637829*i*

f) $x = 0.9$ e $M = 3$.

Respostas Capítulo 4

(4.1)

$X = [1.0021, 1.4127]$ em $k = 3$ iterações.

$Max(|\Delta x|) = 8.9001e-04$

(4.2)

```
format long
n=2;
xi=[+1.  +1.]
k=0;
tol=1.e-2;
dif=1;
while ( dif>tol && k<15 )
       k=k+1;
       a(1,1)=cos(xi(1));  a(1,2)=-sin(xi(2));
       a(2,1)=2*xi(1);            a(2,2)=2*xi(2);
       a(1,3)=-(sin(xi(1))+cos(xi(2))-1.);
       a(2,3)=-(xi(1)^2   +xi(2)^2   -3.);
       dx=fgauss(n,a); %resolve sistema por Gauss
       x=xi+dx;
       dif=max(abs(dx));
       xi=x;
end%while
x
dif
```

(4.3)

$X = [1.00208679989160, -1.41273566015834]$ em $k = 7$ iterações

$Max(|\Delta X|) = 1e-16$

(4.4)

$X = [1.00208679989160, -1.41273566015834]$ em $k = 8$ iterações

$\sum_{j=1}^{n} \left|\Delta x_j\right| = 1.43813183575807\mathrm{e}-15$

(4.5)

```
clear
format long
n=2
%primeira iteração
xi = [0.2 0.2]
dif = 1.;
k = 0.;
dx = [1e-6 1e-6]
while (dif > 1.e-14 && k <20)
    k=k+1
       A(1,1) = (f1([xi(1)+dx(1) xi(2)      ])-f1(xi))/dx(1);
       A(1,2) = (f1([xi(1)       xi(2)+dx(2)])-f1(xi))/dx(2);
       A(1,3) = -f1(xi);
```

```
        A(2,1) = (f2([xi(1)+dx(1) xi(2)      ])-f2(xi))/dx(1);
        A(2,2) = (f2([xi(1)       xi(2)+dx(2)])-f2(xi))/dx(2);
        A(2,3) = -f2(xi);
        dx=fgauss(n,A) % função que calcula a solução dos sistema pelo metodo de Gauss
        x = xi + dx
        %segunda iteracao
        xi=x;
        dif = min(abs(dx./x)); %Usa-se a mínima diferença para calcular as derivadas
end%while
x
```

```
function y=f1(x)
R=8.314;
T=[300  600];
v=[0.5  0.2];
P=[6235.10     49881.50];
y=(P(1)-x(1)/v(1)^2)*(v(1)-x(2))-R*T(1);
end%function
```

```
function y=f2(x)
R=8.314;
T=[300  600];
v=[0.5  0.2];
P=[6235.10     49881.50];
y=(P(2)-x(1)/v(2)^2)*(v(2)-x(2))-R*T(2);
end%function
```

$X = [-0.0999999999997286,\ 0.1000000000000000]$ em $k = 4$ iterações

$\sum_{j=1}^{n} \left| \Delta x_j \right| = 2.26574783732633\text{e}-16$.

A solução obtida tem máxima exatidão para a variável *double* adotada, portanto os erros de truncamento estão na mesma ordem dos erros de arredondamento.

(4.6)

Se estimarmos valores iniciais iguais, geramos sistemas lineares sem solução (impossíveis); se estimarmos valores iniciais reais diferentes entre si, o processo iterativo não converge, pois não existe solução real; e se estimarmos valores iniciais complexos, conseguimos convergir para uma solução complexa, com 15 dígitos exatos, por exemplo:

Solução inicial $Xi = \begin{bmatrix} 1.000000000000000 + 1.000000000000000i \\ 0.000000000000000 - 1.100000000000000i \\ 1.000000000000000 + 0.900000000000000i \end{bmatrix}$

Solução $X = \begin{bmatrix} 1.03445566832301 - 0.36616921091200i \\ 1.02382850219911 + 0.14862183934687i \\ 1.04171582947788 + 0.21754737156514i \end{bmatrix}$ em $k = 14$ iterações

$Max(|\Delta X|) = 2.82983844005847e-15$

Respostas Capítulo 5

(5.1)

a)

$n = 2 \quad \Rightarrow \quad h = \dfrac{\pi/2 - 0}{2} = \pi/4$

$f(x) = sen(x) \;\Rightarrow\; f'(x) = \cos(x) \;\Rightarrow\; f''(x) = -sen(x) \;\Rightarrow\; f'''(x) = \text{-}\cos(x)$

$M = \max_{x \in [0,\ \pi/2]} |f'''(x)| = |f'''(0)| = \cos(0) = 1$

Então, $Erro(x) \le \left| \dfrac{1 * (\pi/4)^{(2+1)}}{4 * (2+1)} \right| = 0.040373 = 4.0373 * 10^{-2} \cong O(10^{-1})$.

$n = 3 \quad \Rightarrow \quad h = \dfrac{\pi/2 - 0}{3} = \pi/6$

$f(x) = sen(x) \;\Rightarrow\; \ldots \;\Rightarrow\; f^{(4)}(x) = sen(x)$

$M = \max_{x \in [0,\ \pi/2]} |f^{(4)}(x)| = |f^{(4)}(\pi/2)| = sen(\pi/2) = 1 = |f^{(4)}(\pi/2)| = sen(\pi/2) = 1$

Então, $Erro(x) \le \left| \dfrac{1 * (\pi/6)^{(3+1)}}{4 * (3+1)} \right| = 0.0046976 = 4.6976 * 10^{-3} \cong O(10^{-2})$

n mínimo $= 3$.

b)

x=[0.00000 0.78540 1.57080]

y=[0.00000 0.70711 1.00000]

$\Delta^k y_1 = [0.90032 \quad -0.33575]$

$P_2(x) = 0.00000 + 1.16401 * x - 0.33575 * x^2$

c)

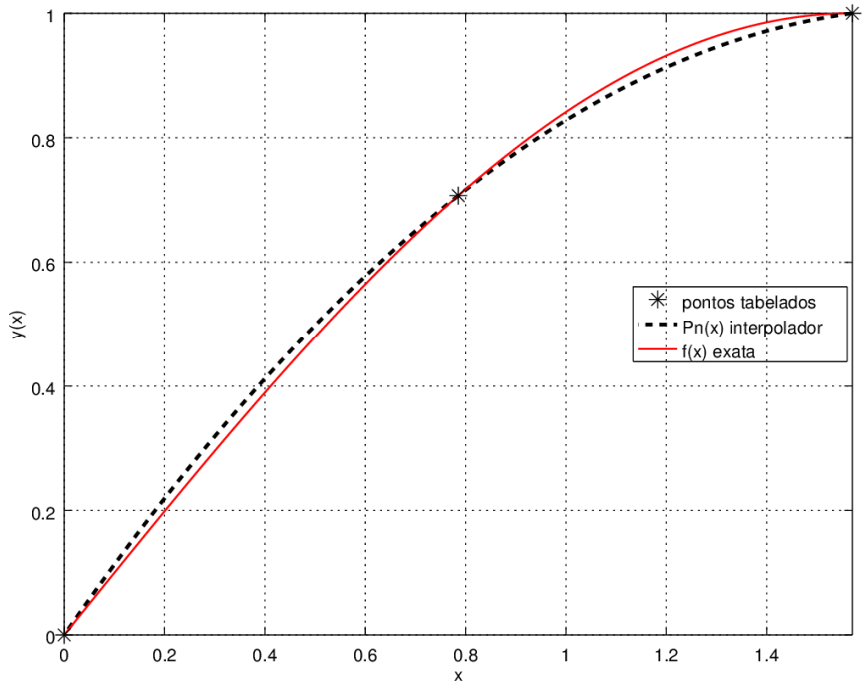
Erro máx. exato entre $P_n(x)$ e $f(x) \cong 0.023531 \cong O(10^{-2})$ (na região do meio dos subintervalos das 2 extremidades).

$Erro\ Corolario\ 2 \cong 0.040373 \cong O(10^{-1})$

d)

```
clear
clc
a=0;
b=pi/2;
n=2;
h=(b-a)/n;
x=a:h:b
y=sin(x)
% Interpolação polinomial de Gregory Newton
difdiv1=fdifdiv(n,x,y) %vetor com diferenças no ponto i = 1
np = 100; %N. de pontos para plotar os resultados aproximados
```

```
hp = (x(n+1)-x(1))/np;
xp = x(1):hp:x(n+1);
ye = sin(xp); %valores exatos
yip=fgregoryn(n,x,y,difdiv1,xp); %y aproximado para cada xp(i)
erro=max(abs(yip.-ye))
plot(x,y,'x',xp,yip,"r;Pn(x) interpolador;",xp,ye,"k;f(x) exata;")
grid
Obs: difdiv1=fdifdiv(n,x,y) e yip=fgregoryn(n,x,y,difdiv1,xp) estão postados no cap. 5.
```



(5.2)

a)

x = [0.000000000000000 0.785398163397448 1.570796326794897]

y = [1.00000000000000 2.02811498164747 2.71828182845905]

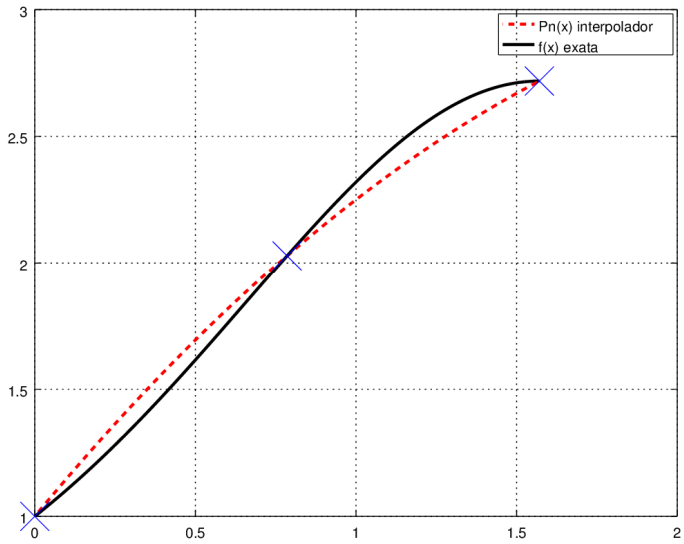
$\Delta^k y_1 = [\ 1.309036651168228 \quad -0.273930440250349\ ]$

C = [1.000000000000000 1.524181115839506 -0.273930440250349]

em $xt = 0.378$, $yt = 1.53700018476260$, $Erro\ yt = 0.0906223305918501$

O resultados de (4a), (4b) e (4c) são os mesmos.

$Erro\max = 0.106033484619552$



(5.3)

a)

```
format long
clear
clc
a=0;
b=pi/2;
      n=2;
      h=(b-a)/n;
      x=a:h:b
      y=cos(x)
      % Interpolação polinomial de Gregory Newton
      difdiv1=fdifdiv(n,x,y) %vetor com diferenças no ponto i = 1
      np = 20*n; %N. de pontos para plotar os resultados aproximados
      hp = (x(n+1)-x(1))/np;
      xp = x(1):hp:x(n+1);
      ye = cos(xp); %valores exatos
      yip=fgregoryn(n,x,y,difdiv1,xp); %y aproximado para cada xp(i)
      erro=abs(yip.-ye);
      erromax=max(erro)
plot(xp,erro,"--k;Erro entre Pn(x) e f(x);",'LineWidth',2)
grid
```

Obs: difdiv1=fdifdiv(n,x,y) e yip=fgregoryn(n,x,y,difdiv1,xp) estão postados no cap. 5.

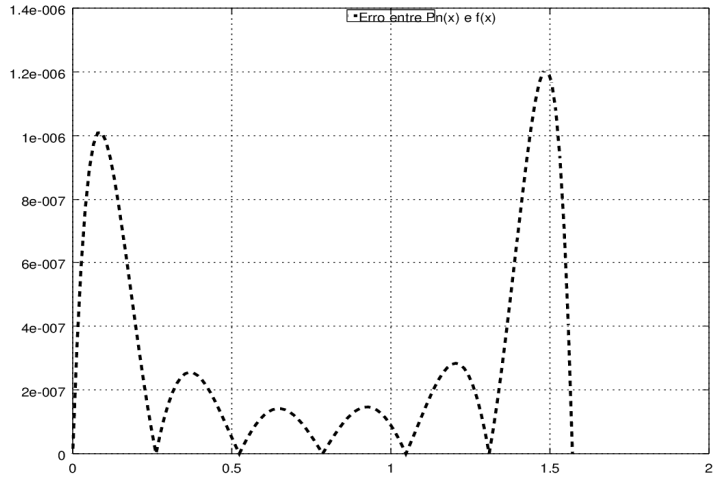
b)

```
format long
clear
clc
a=0;
b=pi/2;
n=1; %valor inicial de n
erromax=1;
while (erromax>sqrt(10)*1e-6) %O(1e-6)
      n=n+1
      h=(b-a)/n;
      x=a:h:b;
      y=cos(x);
      % Interpolação polinomial de Gregory Newton
      difdiv1=fdifdiv(n,x,y); %vetor com diferenças no ponto i = 1
      np = 20*n; %N. de pontos para plotar os resultados aproximados
      hp = (x(n+1)-x(1))/np;
```

```
      xp = x(1):hp:x(n+1);
      ye = cos(xp); %valores exatos
      yip=fgregoryn(n,x,y,difdiv1,xp); %y aproximado para cada xp(i)
      erro=abs(yip.-ye);
      erromax=max(erro)
end
plot(xp,erro,"--k;Erro entre Pn(x) e f(x);",'LineWidth',2)
grid
```

Obs: difdiv1=fdifdiv(n,x,y) e yip=fgregoryn(n,x,y,difdiv1,xp) estão postados no cap. 5.

$n = 6 \rightarrow Erro\max = 1.20543389893479e-06$



(5.4)

a) *[a b c d]=fSplinea(n,x,y) % Splines naturais NAS PONTAS da figura S1=0 e Sn+1=0*

Coeficientes da segunda Spline: a(2)=-5; b(2)=6; c(2)=5; d(2)=-2.

S=[0 12 -18 0];

b)

```
clear
clc
x=[     0       1        2       3 ]
y=[    -3      -2        4       0 ]
n=length(x)-1
%3.  Cálculo por n Splines
%teremos 'n' splines cubicas, uma para cada intervalo, formando um sistema tridiagonal
de 'n-1' equacoes para S (derivadas de segunda ordem)
a=[];b=[]; %zera memoria
  [a b c d]=fSplineb(n,x,y) % Splines quadráticas NAS PONTAS da figura S1=S2 e Sn+1=Sn
np=16;    %4 sub-divisões para cada sub-intervalo entre x(i) e x(i+1) para plotagem
xpp=[];ypp=[];
for i=1:n
    xp=x(i):(x(i+1)-x(i))/np:x(i+1);
      for k=1:np+1
          yp(k)=a(i)*(xp(k)-x(i))*(xp(k)-x(i))*(xp(k)-x(i))+b(i)*(xp(k)-x(i))*(xp(k)-
x(i))+c(i)*(xp(k)-x(i))+d(i);
      end
      xpp=[xpp xp];ypp=[ypp yp];
end
plot(x,y,'*k','markersize',10,xpp,ypp,'-k','LineWidth',2)
```

Obs.: [a b c d]=fSplineb(n,x,y,h) esta postado no cap. 5.

S = [8.75000000000000 8.75000000000000 -13.75000000000000 -13.75000000000000]

a = [0.000000000000000 -3.750000000000000 0.000000000000000]

b = [4.37500000000000 4.37500000000000 -6.87500000000000]

c = [-3.37500000000000 5.37500000000000 2.87500000000000]

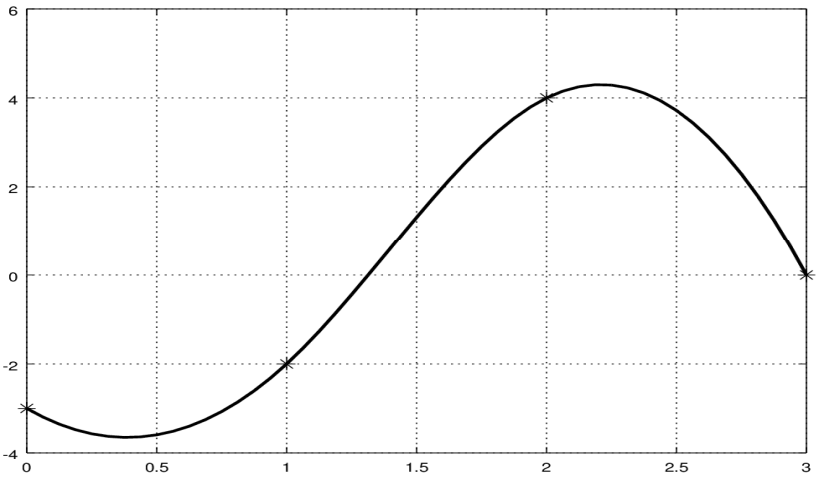
d = [-3 -2 4]

c)

```
x=[0    1       2        3  ] %vetor ordenado, crescente
n=length(x)
% dado xp=1.3, qual spline 'is' o representa?
xp=1.3
if (xp>=x(1) && xp<=x(n))
%Busca binaria
  is = 1;ie = n;
  while ((ie-is)>1)
     m = fix((is + ie)*0.5);%fix(x) retorna o menor inteiro de x
     if (xp >= x(m)) is = m;
     else            ie = m;
     end
  end
  is
else
  is=nan %indica que xp esta fora da faixa dos pontos tabelados
end
```

d) xp =1.30000000000000; is =2; yp =-0.0949999999999998.

e)



(5.5)

```
%Curvas de Bezier para aerofolio:

clear
clc
n=4; %pontos
%A(0 ,0   )  com inclinação de 45º depois de A;
%B(2 ,1   )  com inclinação de 0º antes de B e inclinação -15º depois de B;
%C(8 ,0.2)   com inclinação de -8º antes de B e com inclinação de -8º depois de B;
%D(10,0   )   com inclinação de -5º antes de D.

np=100;
h=1/np; %Espaçamento do parametro t
```

```
xx=[];yy=[];
xp=[];yp=[];

% parte 1: segmento AB
x(1)=0;    y(1)=0;
x(2)=1;    y(2)=1; %inclinação inicio = 45º entre 1 e 2
x(3)=1.9;  y(3)=1; %inclinação final  =  0º entre 3 e 4
x(4)=2;    y(4)=1;
cx=3*(x(2)-x(1));bx=3*(x(3)-x(2))-cx;ax=(x(4)-x(1))-(cx+bx);
cy=3*(y(2)-y(1));by=3*(y(3)-y(2))-cy;ay=(y(4)-y(1))-(cy+by);
t=0;
for i=1:np+1
   xx3(i)=x(1)+t*(cx+t*(bx+t*ax));
   yy3(i)=y(1)+t*(cy+t*(by+t*ay));
   t=t+h;
end%for
xx=[xx xx3];yy=[yy yy3];xp=[xp x];yp=[yp y];
% parte 2: segmento BC
x(1)=2;   y(1)=1;
x(2)=3;   y(2)=1+1*tan(-15*pi/180);  %inclinação inicio = -15º
x(3)=8-1; y(3)=0.2-1*tan(-8*pi/180); %inclinação final = -8º
x(4)=8;   y(4)=0.2;
cx=3*(x(2)-x(1));bx=3*(x(3)-x(2))-cx;ax=(x(4)-x(1))-(cx+bx);
cy=3*(y(2)-y(1));by=3*(y(3)-y(2))-cy;ay=(y(4)-y(1))-(cy+by);
t=0;
for i=1:np+1
   xx3(i)=x(1)+t*(cx+t*(bx+t*ax));
   yy3(i)=y(1)+t*(cy+t*(by+t*ay));
   t=t+h;
end%for
xx=[xx xx3];yy=[yy yy3];xp=[xp x];yp=[yp y];
% parte 3: segmento CD
x(1)=8;    y(1)=0.2;
x(2)=8+0.5;  y(2)=0.2+0.5*tan(-8*pi/180); %inclinação inicio = -8º
x(3)=10-0.5; y(3)=0-0.5*tan(-05*pi/180);   %inclinação final = -5º
x(4)=10;   y(4)=0;
cx=3*(x(2)-x(1));bx=3*(x(3)-x(2))-cx;ax=(x(4)-x(1))-(cx+bx);
cy=3*(y(2)-y(1));by=3*(y(3)-y(2))-cy;ay=(y(4)-y(1))-(cy+by);
t=0;
for i=1:np+1
   xx3(i)=x(1)+t*(cx+t*(bx+t*ax));
   yy3(i)=y(1)+t*(cy+t*(by+t*ay));
   t=t+h;
end%for
xx=[xx xx3];yy=[yy yy3];xp=[xp x];yp=[yp y];
plot(xx,yy,'g',xp,yp,'*')
```

Respostas Capítulo 6

(6.1)

$n = 3$ – Erro P_3 max = $0.00121794952022780 \cong O(10^{-3})$ ($n = 2$ foi testado e não foi suficiente)

Diferenças Divididas =

[0.771414432017616 0.157627243778130 −0.157627243778130]

b)

$n = 3$ – Erro exato M_3 max = $0.00813765147456313 \cong O(10^{-2})$

Coef Maclaurin =

[0.000000000000000 1.000000000000000 0.000000000000000 −0.166666666666667]

c)

Tchebychev-Maclaurin com grau inicial de Maclaurin $n = 5$:

%Erro de truncamento máximo (resto máximo da série):

$M_5(x) = |sen(x)| * (1-0)^{(5+1)} / (5+1)! = 1/6! = 1/720 = 1.388888e-03 \cong O(10^{-3})$

%Erro de truncamento máximo (1º termo abandonado em séries com sinais alternadas):

$M_5(x) = 1/7! = 1.98412698413e-04 \cong O(10^{-4})$

%Então, tomamos o menor Erro máximo de $M_5(x) = 1.98412698413e-04 \cong O(10^{-4})$.

%Série de Maclaurin, em ordem crescente de grau:

$M_5(x) = 0.0x^0 + 1.0x^1 + 0.0x^2 - 0.166666666666667x^3 + 0.0x^4 + 0.008333333333333x^5$

Série de Tchebychev a partir de Maclaurin:

$TC_5(T) = 0. + (169/192)T_1 - (5/128)T_3 + (1/1920)T_5$

truncando o último termo dessa série de Tchebychev:

%Erro de truncamento máximo total $= 1/7! + 1/1920 = 7.192460e-04 \cong O(10^{-3})$ fica maior do que a ordem de erro de $M_5(x)$)

$n = 3$

$\mathrm{TC}_3(T) = 0. + (169/192)T_1 - (5/128)T_3$

$TC_3(x) = 0. * 1 + (169/192) * x - (5/128) * (4 * x^3 - 3 * x)$

$TC_3(x) = (383 * x)/384 - (5 * x^3)/32$

Cujo erro exato máximo de $TC_3(x) = 5.67503606437536e-04 \cong O(10^{-3})$

Grau inicial de Maclaurin $n = 7$:

$\%\, M_7(x) = 0.0x^0 + 1.0x^1 + 0.0x^2 - (1/6)x^3 + 0.0x^4 + (1/120)x^5 + 0.0x^6 - (1/5040)x^7$

%Erro de truncamento máximo (resto máximo da série):

$M_7(x) = |sen(x)| * (1-0)^{(7+1)} / (7+1)! = 1/8! = 1/40320 = 2.4801587301 * 10^{-05} \cong O(10^{-5})$

%Erro de truncamento máximo (1º termo abandonado em séries com sinais alternadas):

$M_7(x) = 1/9! = 2.75573192239859e-06 \cong O(10^{-6})$

% Então, tomamos o menor Erro máximo de $M_7(x) = 2.75573192239859e-06 \cong O(10^{-6})$

$\%\ \mathrm{TC}_7(x) = 1.0\mathrm{T}_1 - (1/6)(\mathrm{T}_3 + 3\mathrm{T}_1)/4 + (1/120)(\mathrm{T}_5 + 5\mathrm{T}_3 + 10\mathrm{T}_1)/16 +$
$-(1/5040)(\mathrm{T}_7 + 7\mathrm{T}_5 + 21\mathrm{T}_3 + 35\mathrm{T}_1)/64$

$\%\, TC_7(T) = 0. + (169/192)T_1 - (601/15360)T_3 + (23/46080)T_5 - (1/322560)T_7$

%Erro de truncamento máximo total=$1/9! + 1/322560 = 5.85593e-06 \cong O(10^{-5})$ também fica maior do que a ordem de erro de $M_7(x)$.

$n = 5$

$\%\, TC_5(T) = 0. + (169/192)T_1 - (601/15360)T_3 + (23/46080)T_5$

$\%\, TC_5(T(x)) = 0. + (11521x)/11520 - (959x^3)/5760 + (23x^5)/2880$

Cujo erro exato máximo de $TC_5(x) = 1.08876303214656e-04 \cong O(10^{-4})$

As aproximações algébricas de Tchebychev-Maclaurin de graus reduzidos geraram erros maiores do que as séries originais de Maclaurin de grau original.

d)

Tchebyschev com coeficientes numéricos, via teorema de Tchebychev:

$n = 3$

$b = [2.83773005094190e-17 \quad 8.80101171489865e-01$
$3.41948691584548e-18 \quad -3.91267079653375e-02]$

Erro TC $n = 3$ numerico Max = $5.01502505832752e-04 \cong O(10^{-3})$

$n = 5$

$b = [2.83773005094190e-17 \quad 8.80101171489865e-01 \quad 3.41948691584548e-18$
$-3.91267079653375e-02 \quad 4.89386309254769e-17 \quad 4.99515460422549e-04]$

Erro TC $n = 5$ numerico Max = $3.01373744460154e-06 \cong O(10^{-6})$

As aproximações numéricas dos coeficientes de Tchebychev geraram séries com erros menores do que as aproximações algébricas Tchebychev-Maclaurin. Com grau $n = 5$ a série de Tchebychev aproxima $f(x)$ com erro máximo da ordem de $O(10^{-6})$.

e)

Erro exato max de Padé $R_{32}(x) = 2.01143538055337e-04 \cong O(10^{-4})$

f)

```
format long
clc
clear
%f(x)=sen(x);                          com x entre [a=-1;b=+1]
n = 1;
a = -1
b = +1
np = 100; %N. de pontos para plotar os resultados aproximados
hp = (b-a)/np;
xp = a:hp:b;
ye = sin(xp); %valores exatos para plotagem

erromaxPn=1;
while (erromaxPn>sqrt(10)*1e-2) %O(1e-2)
       n=n+1
       h = (b - a)/n; % passo
       x = a:h:b;
%      t=0.5.*(b.-a).+0.5.*(b.+a);
       y = sin(x);
       % Interpolação polinomial de Gregory Newton
       difdiv1=fdifdiv(n,x,y); %vetor com diferenças no ponto i = 1      ye  =  sin(xp);
%valores exatos
  transpose(difdiv1)
       yip=fgregoryn(n,x,y,difdiv1,xp); %y aproximado para cada xp(i)
       erroPn=abs(yip.-ye);
       erromaxPn=max(erroPn);
end
'Interpolacaopolinomial:'
n
erromaxPn

%Serie de Maclaurin
%coeficientes de Maclaurin ordem crescente de grau
for i=1:5
       ii=2*i;
       c(ii-1)=0;
       c(ii)=(-1)^(i-1)/factorial(2*i-1);
end
c
n=0;
erromaxMn=1;
while (erromaxMn>sqrt(10)*1e-2) %O(1e-2)
       n=n+1
%      t=0.5.*(b.-a).+0.5.*(b.+a);
       % Aprox. serie Maclaurin
       yiM =fPnH(n, c, xp);
       erroMn=abs(yiM.-ye);
       erromaxMn=max(erroMn)
end
'Maclaurin'
n
erromaxMn
'coeficientes da serie de MacLaurim:'
c(1:n+1)

'Tchebyschev algebrico n=3'
n=5;
%M5(x)=0.0*x0+1.0*x1+0.0*x2-0.166666666666667*x3+0.0*x4+0.008333333333333*x5
%Erro de truncamento maximo de M5(x)=|sen(x)|*(1-0)(5+1)/(5+1)!=1/720=1.388888e-3=O(10-
3) (resto máximo da série)
```

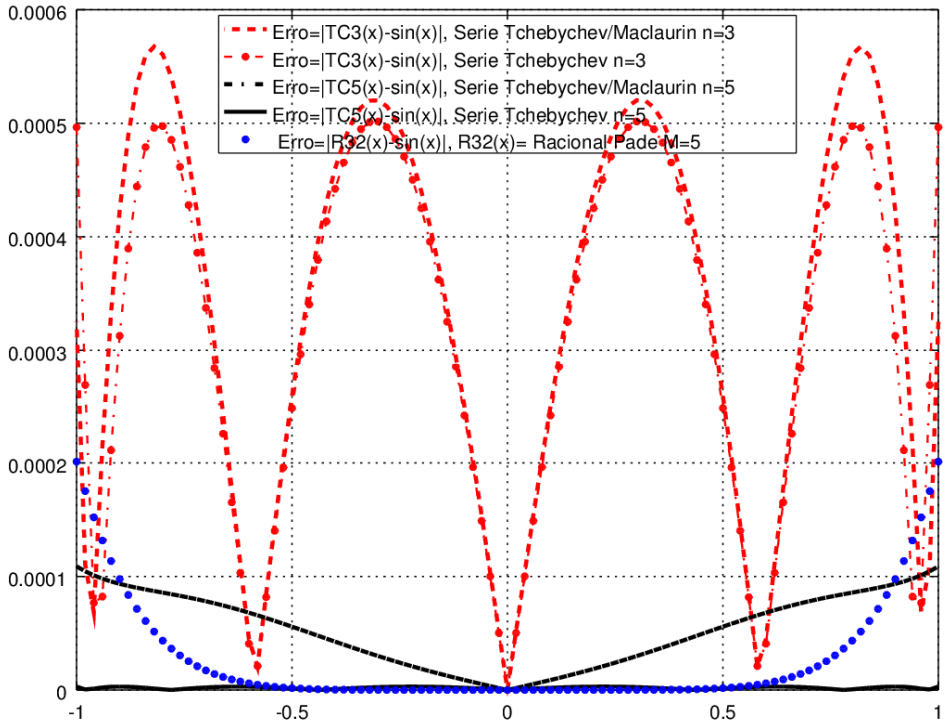
```
%Erro de truncamento maximo de M5(x)= 1/7! =1.98412698413e-04=O(10-4) (1º termo abandonado em séries com sinais alternadas - não vale para termos com variação dupla)
%C5(T)= 0. + 169/192*T1 - (5*T3)/128 + T5/1920 (truncando o último termo da série de Tchebychev)
%Erro de truncamento maximo total = 1/6!+1/1920 = 1.909722222e-3=O(10-3)  (ordem de M5(x))
%C3(T)= 0. + 169/192*T1 - (5*T3)/128
%C3(x)= 0.*1 + 169/192*x - (5*(4*x3-3*x))/128
%C3(x)= (383*x)/384-(5*x3)/32
n=3
C=[0 383/384 0 -5/32 ]
yiT = fPnH(n, C, xp);
erroTn1=abs(yiT.-ye);
erromaxTn1=max(erroTn1)

'Tchebyschev-Maclaurin (algébrico) n=5'
n=7;
%M7(x)=0.0*x0+1.0*x1+0.0*x2-0.166666666666667*x3+0.0*x4+0.008333333333333*x5+0.0*x6-1.98412698412698e-04*x7
%M7(x)=0.0*x0+1.0*x1+0.0*x2-1/6*x3+0.0*x4+1/120*x5+0.0*x6-1/5040*x7
%Erro de truncamento maximo de M7(x)= |sen(x)|*(1-0)(7+1)/(7+1)!=1/8!=1/40320= 2.4801587301e-05=O(10-5)
%Erro de truncamento maximo de M7(x)= 1/9! =2.75573192239859e-06=O(10-6) (1º termo abandonado em séries com sinais alternadas - nao vale para seres com grau variando de 2 em 2)
%C7(x)= 1.0*T1-1/6*(T3+3*T1)/4+1/120*(T5+5*T3+10*T1)/16-1/5040*(T7+7*T5+21*T3+35*T1)/64
%C7(T)= 0. + 169/192*T1 -(601*T3)/15360+(23*T5)/46080-T7/322560
%Erro de truncamento maximo total = 1/8!+1/322560 = 2.790178571e-05 =O(10-5) (ordem de M7(x))
%C5(T)= 0. + 169/192*T1 -(601*T3)/15360+(23*T5)/46080
%C5(T)= 0. + 169/192*x -(601*(4*x3-3*x))/15360+(23*(16*x^5-20*x3+5*x))/46080
%C5(T)=0. + (11521 x)/11520 - (959 x^3)/5760 + (23 x^5)/2880
n=5
C=[0. +(11521)/11520 0 -(959)/5760  0.  +(23)/2880 ]
yiT2 = fPnH(n, C, xp);
erroTn2=abs(yiT2.-ye);
erromaxTn2=max(erroTn2)

'coeficientes Tchebyschev (numérico) n=3'
n=3
b=fTchebychev(n);

%Serie de Tchebychev em função dos polinomios de Tchebychev Ti(x) grau 3:
YTC3=b(1).*1+b(2).*xp.+b(3).*(2.*xp.^2.-1).+b(4).*(4.*xp.^3.-3.*xp);
erroTCaprox3=abs(YTC3.-ye);
erroTCaproxMax3=max(abs(YTC3.-ye))

'Tchebyschev numerico n=5'
n=5
b=fTchebychev(n);
%Serie de Tchebychev em função dos polinomios de Tchebychev Ti(x) grau 5:
YTC5=b(1).*1+b(2).*xp.+b(3).*(2.*xp.^2.-1).+b(4).*(4.*xp.^3.-3.*xp).+b(5).*(8.*xp.^4.-8.*xp.^2.+1).+b(6).*(16.*xp.^5-20.*xp.^3.+5.*xp);
erroTCaprox5=abs(YTC5.-ye);
erroTCaproxMax5=max(abs(YTC5.-ye))

'Pade R32'
%R32 = (a(1) + a(2)x + a(3)x^2 + a(4)x^3)/(1 + b(1)x + b(2)*x^2)
npade=3
mpade=2
[a b]=fPade(npade,mpade,c)
yPade=fPnH(npade,a,xp)./fPnH(mpade,b,xp);
erroR32=abs(yPade.-ye);
erromaxR32=max(erroR32)

plot(xp,erroTn1,"--r;Erro=|TC3(x)-sin(x)|,        Serie        Tchebychev        algebrica n=3;",'linewidth',2,xp,erroTCaprox3,"-.-r;Erro=|TC3(x)-sin(x)|,      Serie      Tchebychev
```

```
numerica  n=3;",'linewidth',2,xp,erroTn2,"-.k;Erro=|TC5(x)-sin(x)|,  Serie  Tchebychev
algebrica n=5;",'linewidth',2,xp,erroTCaprox5,"-k;Erro=|TC5(x)-sin(x)|, Serie Tchebychev
numerica n=5;",'linewidth',2,xp,erroR32,".b; Erro=|R32(x)-sin(x)|, R32(x)= RacionalPade
M=5;",'linewidth',2)
legend('location','north')
grid on
```

```
function y=fx0(x)
  y=sin(x);
end
```

g)

Gráfico de erros do exercício 6.1:



Observe que os erros da aproximação numérica de Tchebychev de grau $n = 5$ ficam bem inferiores aos das demais aproximações, conforme linha contínua, bem próxima ao zero, pois o seu erro está na ordem de $O(10^{-6})$. As séries de **Tchebychev** são indicadas para funções suaves como as trigonométricas (não assintóticas).

(6.2)

a) Para **interpolador polinomial**:

$P_{44}(x) \rightarrow$ Erro P_{44} Max = $1.81988e-06$ $(O(10^{-6}))$ (por Lagrange)

$P_{44}(x) \rightarrow$ Erro P_{44} Max = $1.49443e-06$ $(O(10^{-6}))$ (por Gregory-Newton)

$P_{56}(x)$ $\rightarrow$ Erro P_{56} Max = $8.05958e-05$ ($O(10^{-4})$) (se aumentarmos $n \rightarrow$ o erro cresce ainda mais)

b) Para **séries de Maclaurin**:

para grau 104 $\rightarrow$ Erro Mac Max = $2.51976e-06$ ($O(10^{-6})$)

c) Para **série de Tchebyschev**

para grau 24 $\rightarrow$ Erro Cheb Max = $2.3690784e-06$ ($O(10^{-6})$)

d) Para a **racional de Padé**:

$R_{11,11}(x)$ $\rightarrow$ Erro Padé Max = $2.51542897e-06$ ($O(10^{-6})$)

Nesse exercício, atingimos as aproximações desejadas com **interpolação** polinomial de grau 44, **séries de Maclaurin** de grau 104, e **Padé** de grau total 22. **Padé** foi a aproximação mais eficiente, sendo indicada para funções do tipo $f(x) = \ln(x)$, que é assintótica na região de $x = 0$.

(7.1)

a)

$$D(a,b)=\sum_{k=1}^{m}\left(\ln\left(a+b*T_k^2\right)-V_k\right)^2$$

$$\frac{\partial D(a,b)}{\partial a}=2\sum_{k=1}^{m}\left[\ln\left(a+b*T_k^2\right)-V_k\right]*\left[\frac{1}{a+b*T_k^2}\right]=0$$

$$\frac{\partial D(a,b)}{\partial a}=2\sum_{k=1}^{m}\left[\ln\left(a+b*T_k^2\right)-V_k\right]*\left[\frac{T_k^{\,2}}{a+b*T_k^2}\right]=0$$

Então, temos que resolver as duas equações não lineares a seguir:

$$\begin{cases} f_1(a,b)=\sum_{k=1}^{m}\left[\ln\left(a+b*T_k^2\right)-V_k\right]*\left[\frac{1}{a+b*T_k^2}\right]=0 \\ f_2(a,b)=\sum_{k=1}^{m}\left[\ln\left(a+b*T_k^2\right)-V_k\right]*\left[\frac{T_k^{\,2}}{a+b*T_k^2}\right]=0 \end{cases}$$

b)

a = 0.993976624397992

b = 0.988369280851848

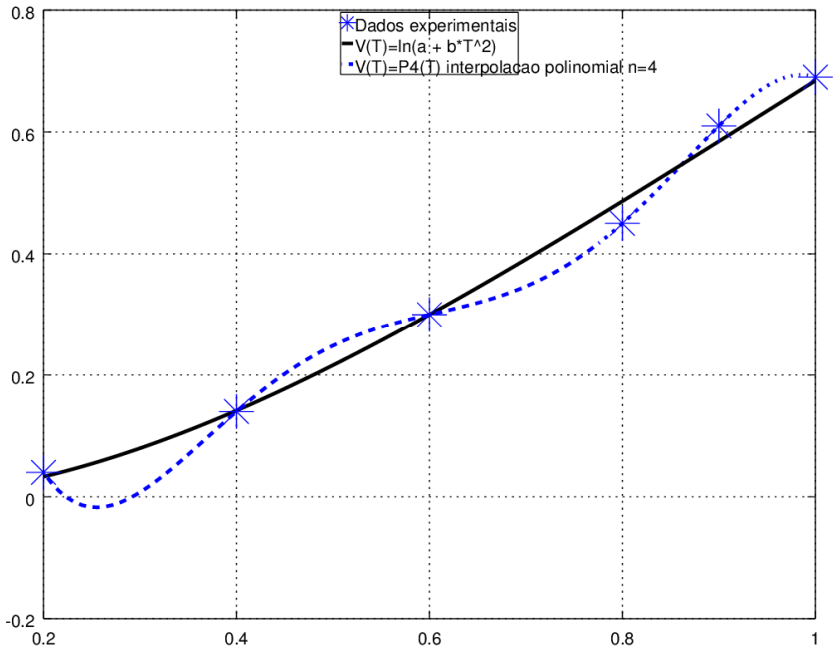
c)

Desvios = [7.03787366849873e-03 1.59999936018476e-03 5.12971037643362e-05 3.64507336445424e-02 2.52425062477697e-02 5.71905598469447e-03]

$R^2 = 0.993843456588333$

d)

Gráfico da aproximação via ajuste direto à função não polinomial $V(T) = ln(a + b * T^2)$ e via interpolação polinomial.



Observamos que o ajuste direto da função original aos m=6 pontos representa bem a tendência dos pontos (linha contínua) até mesmo fora da faixa dos pontos. Também minimiza os erros experimentais e tem um coeficiente de determinação razoável. O interpolador polinomial de grau n=5 passa sobre todos os m=6 pontos e como esses pontos experimentais tem erros inerentes, o interpolador fica muito sinuoso e acentua erros nos intervalos entre os pontos tabelados (linha tracejada).

```
clear
clc
m=6;
T=[ 0.2    0.4    0.6    0.8   0.9   1.0  ];
V=[ 0.04   0.14   0.30   0.45  0.61  0.69 ];
% Ajuste DIRETO para determinação de coeficientes não lineares de funções ajustadas
% V(T)=ln(a + b*T^2)
xi=[0.01   0.01]
s=fNewton2h(xi);
a=s(1)
b=s(2)
%Desvio local D=abs((a+b*T(k)^(-2))^(-1)- V(k))
D=abs(log(a+b.*T.^2).-V)
%Coeficiente de Determinação:
ym=0;    for k=1:m ym=ym+V(k);                          end
ym=ym/m;
SQT=0; for k=1:m SQT=SQT+(V(k)  -ym             )^2; end %soma dos quadrados totais
SQE=0; for k=1:m SQE=SQE+(V(k)  -log(a+b*T(k)^2))^2; end %soma do quadrado dos residuos
R1=1-SQE/SQT %coeficiente de determinação simplificado
np=100;
Tp=T(1):(T(m)-T(1))/np:T(m);
Vp=log(a.+b.*Tp.^2);
%Interpolação polinomial de grau n=m-1=4
n=m-1
difdiv1=fdifdiv(n,T,V); %vetor com diferenças no ponto i = 1      ye = sin(xp); %valores
exatos
yip=fgregoryn(n,T,V,difdiv1,Tp); %y interpolado para cada xp(i)
```

```
plot(T,V,'*','markersize',20,Tp,Vp,'-k','LineWidth',2,Tp,yip,'--b','LineWidth',2)
legend('Dados experimentais','V(T)=ln(a + b*T^2)','V(T)=P4(T) interpolacao polinomial
n=4','location','north')
grid on
```

```
function x=fNewton2h(xi)
dx=[1.e-6 1.e-6];
dif=1; k=0;
while (dif>1.e-14 && k<30)
      k=k+1;
A=[(h1([xi(1)+dx(1),xi(2)])-h1(xi))/dx(1),(h1([xi(1),xi(2)+dx(2)])-h1(xi))/dx(2),      -
h1(xi);
  (h2([xi(1)+dx(1),xi(2)])-h2(xi))/dx(1),(h2([xi(1),xi(2)+dx(2)])-h2(xi))/dx(2),      -
h2(xi);];
      dx=fgauss(2,A);
      x=xi+dx;
      xi=x;
      dif=max(abs(dx));
end %while
end
```

```
function y= h1(x)
m=6;
T=[ 0.2    0.4    0.6    0.8   0.9   1.0  ];
V=[ 0.04   0.14   0.30   0.45  0.61  0.69 ];
y=0; %V(T)=ln(a + b*T^2)
for k=1:m
   y=y+(log(x(1)+x(2)*T(k)^2)-V(k))*1/(x(1)+x(2)*T(k)^2);
end
end
```

```
function y= h2(x)
m=6;
T=[ 0.2    0.4    0.6    0.8   0.9   1.0  ];
V=[ 0.04   0.14   0.30   0.45  0.61  0.69 ];
y=0; %V(T)=ln(a + b*T^2)
for k=1:m
   y=y+(log(x(1)+x(2)*T(k)^2)-V(k))*T(k)^2/(x(1)+x(2)*T(k)^2);
end
end
```

(7.2)

Empregaria **interpolação** polinomial quando os pontos tabelados disponíveis têm boa exatidão ou quando há função geradora. Empregaria o **ajuste** polinomial quando os pontos disponíveis contêm erros inerentes de experimentos.

(7.3)

O polinômio interpolador deve **passar sobre** cada ponto tabelado.

(7.4)

O polinômio ajustador deve **passar o mais próximo** possível dos pontos tabelados, com o mínimo desvio quadrático total.

(7.5)

Quando ajustamos um polinômio $P_n(x)$ de grau n a uma tabela de m=n+1 pontos, estamos obtendo um ajustador com desvio “nulo” em relação aos pontos tabelados, ou seja, é um polinômio que passa tão perto que chega a passar sobre todos os pontos e coincide com o próprio interpolador polinomial.

(7.6)

a)

Ajustes polinomiais de grau 1 e 2:

n_1 = 1, $a_1(i)$ = [1.10740979879120150 0.00138408287794235]

n_2 = 2, $a_2(i)$ = [7.98711647191078e-01 1.70500728185132e-02 -1.38369748604815e-04]

b)

Por Gregory Newton de grau n_3 = 6:

$\Delta^k y_1$ = [6.06060606060606e-03 -4.61814005413262e-05 1.36249473646770e-06

-2.93639508346414e-07 1.72213076619883e-08 -1.01243651158218e-09]

ou

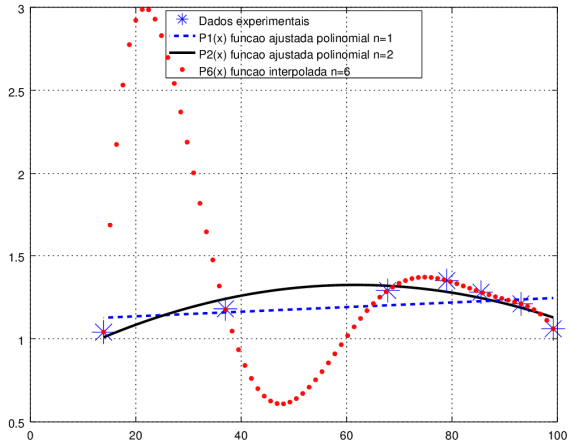
n_3 = 6, $a_3(i)$ = [-2.61806618536017e+01 3.97977288055509e+00 -2.03034014312234e-01

4.93865080648589e-03 -6.24866582641935e-05 3.98201166972283e-07 -1.01243651158712e-09]

f

c)

Gráfico dos m=7 pontos experimentais, das duas funções aproximadoras ajustadas e da função interpoladora.



d)

$R_1^2 = 0.13712176984618$

$R_2^2 = 0.800928569331274$

$R_3^2 = 1.0$

e)

A melhor representação para o comportamento do volume de álcool em função da temperatura na faixa medida é o obtido via ajuste quadrático, n=2, conforme o gráfico de (c), em que a interpolação polinomial fica inviável, muito sinuosa e o ajuste linear não captura a leve curvatura dos pontos experimentais. Observe que o Coeficiente de Determinação R^2 não deve ser usado isoladamente como informação para a definição da aproximação mais adequada, pois, para a interpolação polinomial, temos coeficientes de determinação unitários, com resíduos (desvios locais) nulos, e no intervalo entre os pontos tabelados podemos ter uma aproximação inconsistente.

```
clear
clc
m=7;
x=[ 13.9       37.0    67.8    79.0    85.5    93.1    99.2]
y=[ 1.04       1.18    1.29    1.35    1.28    1.21    1.06]
% estrutura (plotagem) da funcao ajustadora
np = 10*m
xp = x(1):(x(m)-x(1))/np:x(m);

% Ajuste polinomial para determinação de coeficientes
n1 = 1 % grau do polinomio ajustador (definido pela forma do grafico)
a1 = fdetajustePn(n1,m,x,y)
R1=fCoefDeterminacaoPn(n1,a1,m,x,y)
ypajuste1=fPnH(n1,a1,xp);

n2 = 2 % grau do polinomio ajustador (definido pela forma do grafico)
a2 = fdetajustePn(n2,m,x,y)
R2=fCoefDeterminacaoPn(n2,a2,m,x,y)
ypajuste2=fPnH(n2,a2,xp);
```

```
%Interpolação polinomial de grau n=m-1=5
n3=m-1
difdiv1=fdifdiv(n3,x,y) %vetor com diferenças no ponto i = 1      ye = sin(xp); %valores
exatos
ypinter=fgregoryn(n3,x,y,difdiv1,xp); %y interpolado para cada xp(i)
a3 = fInterpolador(n3,x,y)
R3=fCoefDeterminacaoPn(n3,a3,m,x,y)

plot(x,y,'*','markersize',20,xp,ypajuste1,'--b','LineWidth',2,xp,ypajuste2,'-
k','LineWidth',2,xp,ypinter,'.r','LineWidth',1)
legend('Dados  experimentais','P1(x)  funcao  ajustada  polinomial  n=1','P2(x)  funcao
ajustada polinomial n=2','P6(x) funcao interpolada n=6','location','north')
grid on
```

```
function coef = fInterpolador(n,x,y)
nsis=n+1;
for i=1:nsis
    A(i,1)=1;
    for j=2:nsis
        A(i,j)=A(i,j-1)*x(i);
    end
end
A=[A transpose(y)];
coef = fgauss(nsis, A);
end
```

Observação: as funções *function y=fPnH(n,a,xi), function difdiv1=fdifdiv(n,x,y), function f=fgregoryn(n,x,y,difdiv1,xp)* estão definidas com outros algoritmos.

(7.7)

a)

A = [2.15290000000000 1.36471838574685 1.36100000000000;
1.36471838574685 2.50595854935421 2.49041638031686]

a = 0.00336649765043649

b = 0.991964555725925

b)

desvios (residuos) locais =

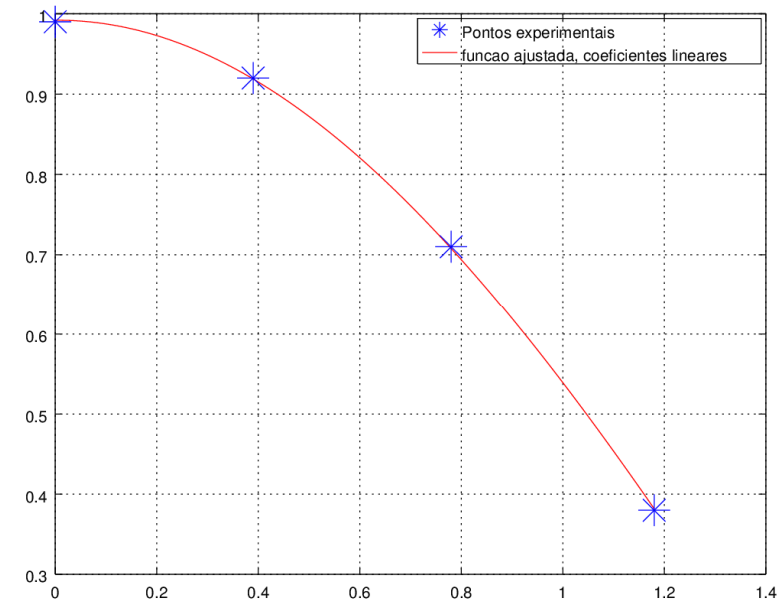
[0.00196455572592569 -0.00121006126808665 -0.00217309993876502 0.00183639139558511]

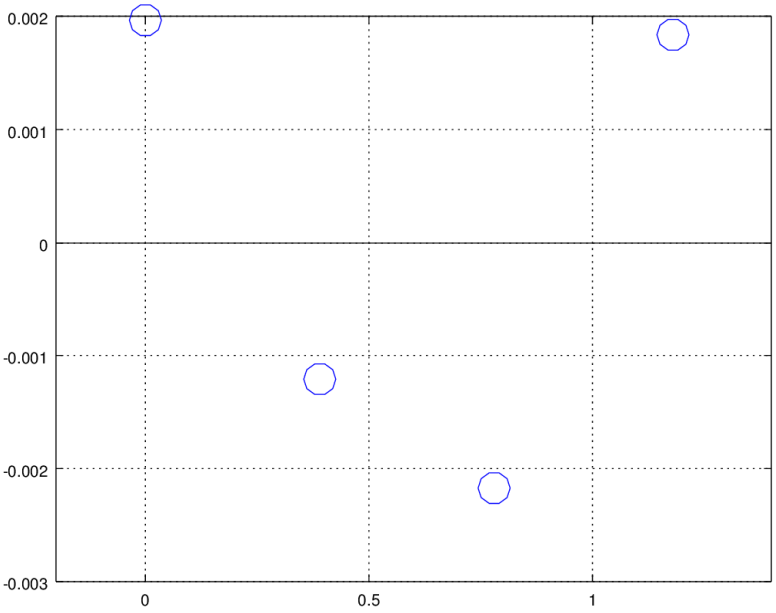
Media dos desvios = 1.04446478664783e-04

$R^2 = 0.999940362559225$

c)

Gráficos dos desvios locais relativos ao ajuste da curva V(T)= a*T+b*cos(T).





d)

Apesar do pequeno número de pontos amostrais, consideramos que: os desvios locais ficaram uniformemente distribuídos e da ordem de 10^{-3} vezes menores do que os dados V(T) medidos; sua média de desvios ficou próxima de zero (distribuição em torno do zero); o coeficiente de determinação também ficou próximo de 1; e o gráfico da função ajustada ficou bem comportado.

Então, podemos concluir que o ajuste V(T)= a*T+b*cos(T) ficou adequado aos pontos amostrais.

```
format long
clear
clc
m=4
T = [ 0.00     0.39    0.78    1.18 ]
V = [ 0.99     0.92    0.71    0.38 ]
s= fdetcoefTV(m,T,V);
a=s(1)
b=s(2)
Tp=T(1):0.01:T(m);
Vajuste = a.*Tp.+b.*cos(Tp);
Vk=a.*T.+b.*cos(T);
%Desvio locais:
D=(Vk.-V) %desvios(residuos) locais
mediaresiduos=sum(D)/m
%Coeficiente de determinacao:
ym=0;    for k=1:m ym=ym+V(k);                          end
ym=ym/m;

SQT=0; for k=1:m SQT=SQT+(V(k)  -ym                )^2; end %soma dos quadrados totais
SQE=0;  for  k=1:m  SQE=SQE+(V(k)   -(a*T(k)+b*cos(T(k))))^2;  end  %soma  do  quadrado  dos
residuos
R2=1-SQE/SQT %coeficiente de determinação simplificado

plot(T,V,"*;Pontos    experimentais;",'markersize',20,Tp,Vajuste,"-r;funcao    ajustada,
coeficientes lineares;")
%plot(T,D,'o','markersize',20)
%xlim([-0.2 1.4])
```

```
function a = fdetcoefTV(m,T,V)
%y=y+(x(1)*T(k)+x(2)*cos(T(k))-V(k))*T(k);
A(1,1)=0;for k=1:m A(1,1)=A(1,1)+T(k)^2;          end
A(1,2)=0;for k=1:m A(1,2)=A(1,2)+T(k)*cos(T(k));  end
B(1)=0;  for k=1:m B(1)  =B(1)  +T(k)*V(k);       end
%y=y+(x(1)*T(k)+x(2)*cos(T(k))-V(k))*cos(T(k));
A(2,1)=A(1,2); %simetrico
A(2,2)=0;for k=1:m A(2,2)=A(2,2)+(cos(T(k)))^2;   end
B(2)=0;  for k=1:m B(2)  =B(2)  +V(k)*cos(T(k));  end
[A transpose(B)]
a  =  fCholesky(2,A,B);  %Cholesky:se  A  é  positiva  definida,  senao  a=fgauss(2,[A
transpose(B)])
end
```

Respostas Capítulo 8

(8.1)

a)

$a = 1$

$b = 2$

$f''(x) = \frac{1}{3}\frac{(-2)}{3}(x+1)^{-5/3}$, máximo em $x = 1$

$M = \frac{1}{3}\frac{(-2)}{3}(1+1)^{-5/3} = 0.06999561388$

$10^{-6} = \frac{-h^2(2-1)0.069995613883}{12} \Rightarrow h = 0.0130934836$

$n = (b-a)/h = 76.37$, adotamos $n = 77$ ou 128.

b)

Sim, pois a fórmula do erro de truncamento máximo é obtida através do limite superior do resto da aproximação por série de Taylor e sempre gera um erro maior do que exato, portanto um n também maior do que o necessário.

c)

$n = 65$

d)

$f'''(x) = \frac{1}{3}\frac{(-2)}{3}\frac{(-5)}{3}\frac{(-8)}{3}(x+1)^{-11/3}$, máximo em $x = 1$

$M = \frac{1}{3}\frac{(-2)}{3}\frac{(-5)}{3}\frac{(-8)}{3}(1+1)^{-11/3} = 0.07777290431449834$

$10^{-6} = \frac{-h^4(2-1)0.07777290431449834}{180} \Rightarrow h = 0.219336569635750$

$n = (b-a)/h = 4.55920324486103$, adotamos $n = 6$ (inteiro par)

e)

Obtendo o m por tentativas:

$$EG_m \le (b-a)^{2m+1}\frac{(m!)^4}{(2m+1)[(2m)!]^3} Max\left|f^{2m}(x)\right| \quad \forall\, x \in [a,b]$$

$m = 2$

$M = \frac{1}{3}\frac{(-2)}{3}\frac{(-5)}{3}\frac{(-8)}{3}(1+1)^{-11/3} = 0.07777290431449834$

$EG_2 = (2-1)^{2*2+1}\frac{(2!)^4}{(2*2+1)[(2*2)!]^3}0.07777290431449834 = 1.80029871098376e\text{-}05$

$$m = 3$$

$$M = \frac{1}{3}\frac{(-2)}{3}\frac{(-5)}{3}\frac{(-8)}{3}\frac{(-11)}{3}\frac{(-14)}{3}(1+1)^{-17/3} = 0.332695201789798$$

$$EG_3 = (2-1)^{2*3+1}\frac{(3!)^4}{(2*3+1)[(2*3)!]^3}0.332695201789798 = 1.65027381840178e\text{-}07$$

f)

$$T_2 = 1.35414705919930$$

$$S_2 = 1.35516764223202$$

$$G_{2+1} = 1.35518000718270$$

g)

Erro exato $T_2 = 0.00103289915006233$

Erro exato estimado $T_2 = 7.74061008386617e-04$

Erro exato $S_2 = 1.23161173437580e-05$

Erro exato estimado $S_2 = 1.14983117971423e-05$

Erro exato $G_{2+1} = 4.88333400383567e-08$

Erro exato estimado $G_{2+1} = 4.84920221754948e-08$

h)

O método de Gauss-Legendre gera o resultado mais próximo do exato por apresentar os menores erros.

(8.2)

a)

Aplicando m por tentativas em $f(x) = x^{1/2}$

$$m = 2: \quad M = \frac{1}{2}\frac{(-1)}{2}\frac{(-3)}{2}\frac{(-5)}{2}(1)^{-7/2} = 0.9375$$

$$EG_2 = (2-1)^{2*2+1}\frac{(2!)^4}{(2*2+1)[(2*2)!]^3}0.9375 = 2.17013888888889e\text{-}04$$

$$m = 3: \quad M = \frac{1}{2}\frac{(-1)}{2}\frac{(-3)}{2}\frac{(-5)}{2}\frac{(-7)}{2}\frac{(-9)}{2}(1)^{-11/2} = 14.765625$$

$$EG_3 = (2-1)^{2*3+1}\frac{(3!)^4}{(2*3+1)[(2*3)!]^3}\ 14.765625 = 7.32421875e\text{-}06$$

b)

$G_3 = 1.21895230967666$.

c)

```
clear
clc
format long
%Obter I com erro maximo O(1e-6)
```

```
%limites da integral
a=1
b=2
%integral de gauss legendre
m=1;
ErroexatoGmEstimado=1;
while (ErroexatoGmEstimado>sqrt(10)*1e-6 && m<5)
  m=m+1
  Gm =fGm(a,b,m)
  G2m=fGm(a,b,m+1)      %Valor exato estimado c/ m+1 pontos é suficiente
  ErroexatoGmEstimado=abs(Gm-G2m)
end
m
```

(8.3)

De $f(x)=\sqrt{x^3} \Rightarrow f'(x)=(3/2)\sqrt{x} \Rightarrow L=\int_a^b \sqrt{1+(9/4)x}\,dx$

Efetuando sucessivas tentativas de aproximação por Gauss-Legendre, obtemos: para $m=6$, $G_m=3.52552445823679$;

para m=7 $G_{m+1}=3.52552401705665$ (como estimativa do valor exato); e

Erro Estimado $G_m=4.41180143528896e-07$.

(8.4)

a)

$h=0.1$

$T_n=0.165000000$

b)

$S_n=0.166666667$

c)

O cálculo efetuado pelo método de Simpson resulta em uma área mais próxima da exata porque a função integranda é aproximada por um polinômio de 2º grau em vez de um polinômio de 1º grau, como no método dos Trapézios.

d)

Como estimativa de erro da área obtida pelo método dos Trapézios T_n, podemos recorrer a algum cálculo aproximado de área que seja mais próximo do valor exato como é o caso de S_n, então:

Erro estimado $=|T_n-S_n|=|0.165000000-0.166666667|=0.0016667\cong O(10^{-3})$.

e)

Durante o projeto, poderíamos tomar mais pontos x_i, y_i para definir a integranda com mais exatidão.

(8.5)

a)

Podemos aplicar o método de integração de Gauss-Legendre porque é um método em que não avaliamos a função integranda sobre os extremos do intervalo de integração.

b)

```
function Gm=fGm(a,b,m)
%Coeficientes C(m,k) e t(m,k) do Método de Integração de Gauss-Legendre até m=7 pontos
c=[
   [2                  0                  0                  0                  0                  0                  0                  ];
   [1                  1                  0                  0                  0                  0                  0                  ];
   [5/9                8/9                5/9                0                  0                  0                  0                  ];
   [0.34785484513745385737 0.65214515486254614263 0.65214515486254614263 0.34785484513745385737 0                  0                  0                  ];
   [0.23692688505618908751 0.47862867049936646804 128/225             0.47862867049936646804 0.23692688505618908751 0                  0                  ];
   [0.17132449237917034504 0.36076157304813860757 0.46791393457269104739 0.46791393457269104739 0.36076157304813860757 0.17132449237917034504 0                  ];
   [0.12948496616886969327 0.27970539148927666790 0.38183005050511894495 512/1225             0.38183005050511894495 0.27970539148927666790 0.12948496616886969327];
 ];
t=[
   [ 0                  0                  0                  0                  0                  0                  0                  ];
   [-1/sqrt(3)          1/sqrt(3)          0                  0                  0                  0                  0                  ];
   [-sqrt(3/5)          0                  sqrt(3/5)          0                  0                  0                  0                  ];
   [-0.86113631159405257522 -0.33998104358485626480 0.33998104358485626480 0.86113631159405257522 0                  0                  0                  ];
   [-0.90617984593866399280 -0.53846931010568309104 0                  0.53846931010568309104 0.90617984593866399280 0                  0                  ];
   [-0.93246951420315202781 -0.66120938646626451366 -0.23861918608319690863 0.23861918608319690863 0.66120938646626451366 0.93246951420315202781 0                  ];
   [-0.94910791234275852453 -0.74153118559939443986 -0.40584515137739716691 0                  0.40584515137739716691 0.74153118559939443986 0.94910791234275852453];
 ];
soma=0;
for k=1:m
    x(k)=0.5*(b-a)*t(m,k)+0.5*(b+a);
    y(k)=log(x(k));                          %f(x)=ln(x)
    soma=soma+C(m,k)*y(k);
end %for
Gm=0.5*(b-a)*soma;
end
```

c)

$I_e = -1$

m	G_m	Erro exato $G_m = \lvert G_m - I_e \rvert$
2	-0.895879734614028	0.104120265385972
3	-0.947672383858322	0.0523276161416784
4	-0.968535977854581	0.0314640221454190
5	-0.979000992287376	0.0209990077126244
6	-0.984991210262344	0.0150087897376562
7	-0.988738923004894	0.011261076995105

(8.6)

```
clc
clear
format long
a=0;
b=1;
Ie = erf(b)-erf(a); %integral exata em double
disp"    METODO DOS TRAPEZIOS"
n=1;ErroEstimado=1;
while (ErroEstimado>1e-6)
  n+=1;
  Tn  = fTn(n,a,b);
  T2n = fTn(2*n,a,b);
```

```
  ErroExato = abs(Tn-Ie);
  ErroEstimado = abs(Tn-T2n);
end%while
Tn
printf("n minimo = %d\n",n)
ErroEstimado
ErroExato
disp("\n")

disp"    METODO DE SIMPSON"
n=2;ErroEstimado=1;
while (ErroEstimado>1e-6 && n<20)
n+=2;
Sn=fSn(n,a,b);
S2n=fSn(2*n,a,b);
ErroExato=abs(Sn-Ie);
ErroEstimado=abs(Sn-S2n);
end%while
Sn
printf("n minimo = %d\n",n)
ErroEstimado
ErroExato
disp("\n")

disp"    METODO DE GAUSS - LEGENDRE"
m=1;ErroEstimado=1;
while ErroEstimado>1e-6
m+=1;
[Gm,x,y]=fGm(a,b,m);
ErroExato=abs(Gm-Ie);
[G2m,xa,ya]=fGm(a,b,m+1);
ErroEstimado=abs(Gm-G2m);
end%while
Gm
printf("m minimo = %d\n",m)
ErroEstimado
ErroExato
n=m-1; %Grau do interpolador
disp" \n    POLINOMIO INTERPOLADOR"
coef=fInterpolador(n,x,y);
printf("\nPn(x)       =       %.15f       +       %.15f*x       +       %.15f*x^2       +
%.15f*x^3\n\n",coef(1),coef(2),coef(3),coef(4))
IPn   =   coef(1)*b^1/1+coef(2)*b^2/2+coef(3)*b^3/3+coef(4)*b^4/4   %integral   exata   do
interpolador
Gm_=Gm
diferenca_Gm_IPn = abs(Gm-IPn)

xp=a:0.01:b;
ye=f(xp);
yP=fPnH(n,coef,xp);
plot(x,y,"*",'markersize',20,                    xp,ye,"-k;f(x);",'linewidth',2,xp,yP,"--
r;Pn(x);",'linewidth',2)
```

```
function y=f(t)
  y=2/(sqrt(pi))*exp(-t.^2);
end


function coef = fInterpolador(n,x,y)
nsis=n+1;
for i=1:nsis
    A(i,1)=1;
    for j=2:nsis
        A(i,j)=A(i,j-1)*x(i);
    end
end
A=[A transpose(y)];
coef = fgauss(nsis, A);
end
```

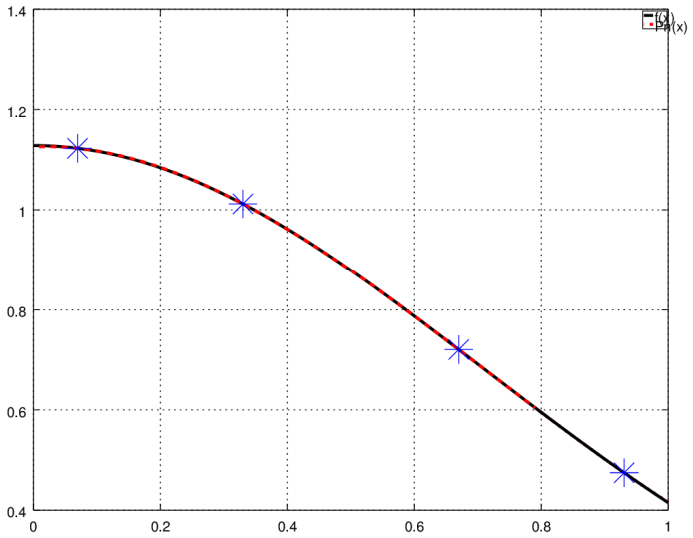
```
function Tn=fTn(n,a,b)
  h=(b-a)/n;
  t=a:h:b;
```

```
  y=f(t);
  soma=0;
  for i=2:n
    soma=soma+y(i);
  end %for i
  Tn=(0.5*h)*(y(1)+2*soma+y(n+1));
end %function
```

```
function Sn=fSn(n,a,b)
  %n obrigatoriamente par
  h=(b-a)/n;
  t=a:h:b;
  y=f(t);
  soma1=0;
  soma2=0;
  for i=2:2:n
    soma1=soma1+y(i);
  end %for i
  for i=3:2:n-1
    soma2=soma2+y(i);
  end %for i
  Sn=(h/3)*(y(1)+y(n+1)+4*soma1+2*soma2);
end %function
```

```
function [Gm,x,y]=fGm(a,b,m)
%Coeficientes C(m,k) e t(m,k) do Método de Integração de Gauss-Legendre
c=[
   [2                  0                  0                  0                  0                  0                  0                  ];
   [1                  1                  0                  0                  0                  0                  0                  ];
   [5/9                8/9                5/9                0                  0                  0                  0                  ];
   [0.34785484513745385737 0.65214515486254614263 0.65214515486254614263 0.34785484513745385737 0                  0                  0                  ];
   [0.23692688505618908751 0.47862867049936646804 128/225             0.47862867049936646804 0.23692688505618908751 0                  0                  ];
   [0.17132449237917034504 0.36076157304813860757 0.46791393457269104739 0.46791393457269104739 0.36076157304813860757 0.17132449237917034504 0                  ];
   [0.12948496616886969327 0.27970539148927666790 0.38183005050511894495 512/1225             0.38183005050511894495 0.27970539148927666790 0.12948496616886969327];
 ];
t=[
   [ 0                  0                  0                  0                  0                  0                  0                  ];
   [-1/sqrt(3)          1/sqrt(3)          0                  0                  0                  0                  0                  ];
   [-sqrt(3/5)          0                  sqrt(3/5)          0                  0                  0                  0                  ];
   [-0.86113631159405257522 -0.33998104358485626480 0.33998104358485626480 0.86113631159405257522 0                  0                  0                  ];
   [-0.90617984593866399280 -0.53846931010568309104 0                  0.53846931010568309104 0.90617984593866399280 0                  0                  ];
   [-0.93246951420315202781 -0.66120938646626451366 -0.23861918608319690863 0.23861918608319690863 0.66120938646626451366 0.93246951420315202781 0                  ];
   [-0.94910791234275852453 -0.74153118559939443986 -0.40584515137739716691 0                  0.40584515137739716691 0.74153118559939443986 0.94910791234275852453];
 ];
  soma=0;
  for k=1:m
    x(k)=0.5*(b-a)*t(m,k)+0.5*(b+a);
    y(k)=f(x(k));
    soma=soma+C(m,k)*y(k);
  end %function
  Gm=0.5*(b-a)*soma;
end %function
```

```
function y=fPnH(n,a,xi)
%Pn(xi)=a(1)+a(2)*xi+a(3)*xi^2+...+a(n)*xi^(n-1)+a(n+1)*xi^n
%Pn(xi)=a(1)+xi*(a(2)+xi*(a(3)+...+xi*(a(n)+xi*a(n+1))...))% HORNER
for ip=1:length(xi) %calcula y p/ cada elemento de xi
  y(ip)=a(n+1);
  for i=n:-1:1
    y(ip)=a(i)+y(ip)*xi(ip);
  end
end
end
```



(8.7)

Por Gauss-Legendre devemos integrar a função completa, composta do fator peso $W(x)=1/\sqrt{1-x^2}$, dado por $g(x)=\dfrac{\ln(1+x)}{\sqrt{1-x^2}}$, entre $a=-1$ e $b=+1$, e obter os seguintes resultados para cada m:

m	G_m	Erro exato $G_m = \lvert G_m - I_e \rvert$
1	0.000000000000000	2.177586090303602
2	-0.496591311683711	1.680994778619891
3	-0.804879364340271	1.372706725963331
4	-1.011018992793002	1.166567097510600
5	-1.158845877013562	1.018740213290040
6	-1.270438652942685	0.907147437360917
7	-1.357948686123227	0.819637404180375

Como a função $\ln(1+x)$ não é bem comportada no intervalo $[-1, +1]$ (tem ponto assintótico em $x \to -1$) e a tabela é limitada ao número m de pontos, uma vez que os "pesos" e "nós" são de difícil obtenção, ficamos ainda muito longe do erro desejado de ordem $O(10^{-7})$.

Por Gauss-TChebyshev aplicamos a equação (8b) apenas à função $f(x)=\ln(1+x)$ (sem o fator peso), pois essa integração já considera esse fator. Nesse método, podemos testar até valores elevados de m, uma vez que os "pesos" e "nós" são de fácil obtenção, conforme segue:

m	GT_m	Erro exato $GT_m = \mid GT_m - I_e \mid$
10^3	-2.17540850421327	2.17758609033636e-03
10^4	-2.17736833169335	2.17758610249419e-04
10^5	-2.17756431443286	2.17758707394822e-05
10^6	-2.17758391275610	2.17754750453381e-06
10^7	-2.17758587291663	2.17386975975842e-07

Observe que temos que usar valores muito elevados de número de pontos, $m = 10^7$, pois a função $\ln(1+x)$ não é bem comportada no intervalo $[-1,\ +1]$ (tem ponto assintótico em $x \to -1$), mas atingimos o Erro na ordem de $O(10^{-7})$. Portanto, nenhum dos métodos propostos é adequado para esse tipo de integração numérica em razão da função integranda assintótica.

Respostas Capítulo 9

(9.1)

a)

$$x = [\,0.000000000000000 \quad 0.100000000000000 \quad 0.200000000000000 \quad]$$

$$y = [\,0.00000000000000000 \;\; 0.00000000000000000 \;\; 0.00100000000000000 \;\;]$$

$$\text{Erro Euler exato} = 0.00180551632033976$$

b)

$$x = [\,0.000000000000000 \quad 0.100000000000000 \quad 0.200000000000000 \quad]$$

$$y2 = [\,0 \;\; 0.00000000000000000 \;\; 0.00050000000000000 \;\; 0.00310250000000000]$$

$$\text{Erro RK2 estimado} = 2.23177910034179e-04$$

$$\text{Erro RK2 exato} \quad = 2.96983679660238e-04$$

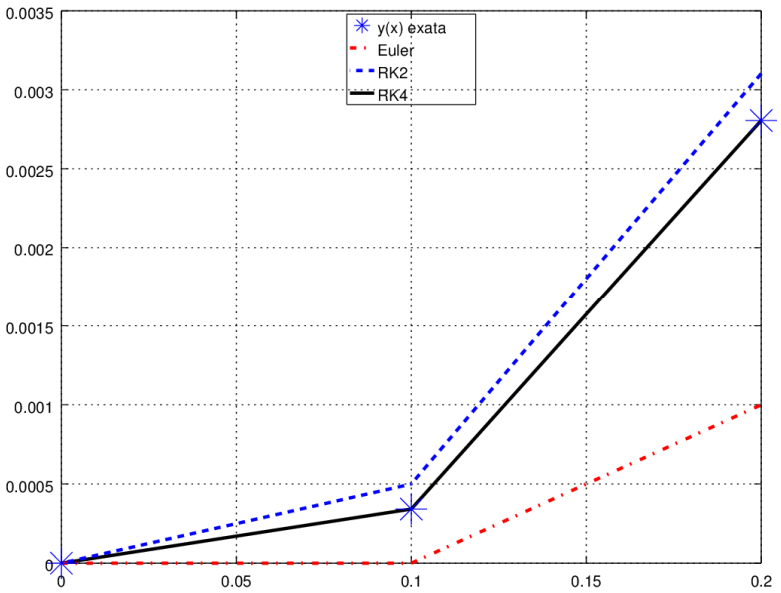
c)

$$x = [\,0.000000000000000 \quad 0.100000000000000 \quad 0.200000000000000 \quad]$$

$$y4 = [\;\; 0.00000000000000000 \;\; 0.00034187500000000 \;\; 0.00280558027864583]$$

$$\text{Erro RK4 exato} \;\; = 6.39583060714488e-08$$

Gráfico com os valores discretos das soluções x e y obtidos com os três métodos:



(9.2)

a)

$$x = [1.0000000000 \;\; 1.0500000000 \;\; 1.1000000000 \;\; 1.1500000000 \;\; 1.2000000000]$$

$$y = [\;\; 1.00000000000 \;\; 0.90000000000 \;\; 0.81428571429 \;\; 0.74025974026 \;\; 0.67588932806 \;\;]$$

b)

```
clear
clc
a=1.0 % Valor inicial de x
b=1.2 % Valor final de x
% Condição Inicial:
x(1)=a  % Valor inicial de x
y(1)=1; % Valor inicial de y
%RK4 p/ EDO
[x y4]=fRK4(n,a,b,x,y)
y4(n+1)
```

Observação: `[x y4]=fRK4(n,a,b,x,y)` disponível no algoritmo RK4 com $f(x,y) = -2y/x$.

c)

```
%disponivel y4
[xa ya]=fRK4(2*n,a,b,x,y);
erroestimRK4=abs(y4(n+1)-ya(2*n+1))
```

(9.3)

a)

$$\begin{cases} y_1'(x) = z_1(x, y_1, y_2) = y_2, & \text{com } y_1(x=1) = y(x=1) = 1 \\ y_2'(x) = z_2(x, y_1, y_2) = (-2y_1 - x)/x, & \text{com } y_2(x=1) = y'(x=1) = -1 \end{cases}$$

b)

```
clear
clc
a=1.0  % Valor inicial de x
b=2.0  % Valor final de x
% Condição Inicial:
x(1) =a  % Valor inicial de x
y1(1)= 1.0; % Valor inicial de y1
y2(1)=-1.0; % Valor inicial de y2

%RK4 p/ sistemas de 2 EDOs
n=8  % Numero de subdivisões do intervalor [a,b]
[x y1 y2]=fRK4sist2(n,a,b,x,y1,y2)
x
y1
```

```
function z=z1(x,y1,y2)
  z=y2;
end
```

```
function z=z2(x,y1,y2)
  z=(-2.*y1.-x)./x;
end
```

```
function [x y1 y2]=fRK4sist2(n,a,b,x,y1,y2)
h=(b-a)/n; % espacamento em x
for k=1:n
    K1z1=z1(x(k),      y1(k),           y2(k)           );
    K1z2=z2(x(k),      y1(k),           y2(k)           );
    K2z1=z1(x(k)+0.5*h,y1(k)+0.5*h*K1z1,y2(k)+0.5*h*K1z2);
    K2z2=z2(x(k)+0.5*h,y1(k)+0.5*h*K1z1,y2(k)+0.5*h*K1z2);
    K3z1=z1(x(k)+0.5*h,y1(k)+0.5*h*K2z1,y2(k)+0.5*h*K2z2);
    K3z2=z2(x(k)+0.5*h,y1(k)+0.5*h*K2z1,y2(k)+0.5*h*K2z2);
```

```
    K4z1=z1(x(k)+h,       y1(k)+h*K3z1,   y2(k)+h*K3z2    );
    K4z2=z2(x(k)+h,       y1(k)+h*K3z1,   y2(k)+h*K3z2    );
    x(k+1)=x(k)+h;
    y1(k+1)=y1(k)+(h/6)*(K1z1+2*(K2z1+K3z1)+K4z1);
    y2(k+1)=y2(k)+(h/6)*(K1z2+2*(K2z2+K3z2)+K4z2);
end
end
```

(9.4)

$$\begin{cases} y_1'(x) = z_1(x, y_1, y_2) = y_2, & \text{com } y_1(x=1) = y(x=1) = 1 \\ y_2'(x) = z_2(x, y_1, y_2) = -(x^{3/2})/(8y_1y_2), & \text{com } y_2(x=1) = y'(x=1) = 1/2 \end{cases}$$

(9.5)

a)

```
function z=fy1(x,y1,y2)
  z=y2;
end
```

```
function z=fy2(x,y1,y2)
  z=-2.*y2./x;
end
```

b)

```
clear
clc
a=1.0  % Valor inicial de x
b=2.0  % Valor final de x
% Condição Inicial:
x(1) =a  % Valor inicial de x
y1(1)= 1.0; % Valor inicial de y1
y2(1)=-1.0; % Valor inicial de y2
%RK4 p/ sistemas de 2 EDOs
n=8  % Numero de subdivisões do intervalor [a,b]
[x y1 y2]=fRK4sist2a(n,a,b,x,y1,y2)
```

```
function [x y1 y2]=fRK4sist2a(n,a,b,x,y1,y2)
h=(b-a)/n; % espacamento em x
for k=1:n
    K1f1=fy1(x(k),      y1(k),          y2(k)          );
    K1f2=fy2(x(k),      y1(k),          y2(k)          );
    K2f1=fy1(x(k)+0.5*h,y1(k)+0.5*h*K1f1,y2(k)+0.5*h*K1f2);
    K2f2=fy2(x(k)+0.5*h,y1(k)+0.5*h*K1f1,y2(k)+0.5*h*K1f2);
    K3f1=fy1(x(k)+0.5*h,y1(k)+0.5*h*K2f1,y2(k)+0.5*h*K2f2);
    K3f2=fy2(x(k)+0.5*h,y1(k)+0.5*h*K2f1,y2(k)+0.5*h*K2f2);
    K4f1=fy1(x(k)+h,    y1(k)+   h*K3f1,y2(k)+   h*K3f2);
    K4f2=fy2(x(k)+h,    y1(k)+   h*K3f1,y2(k)+   h*K3f2);
    x(k+1)=x(k)+h;
    y1(k+1)=y1(k)+(h/6)*(K1f1+2*(K2f1+K3f1)+K4f1);
    y2(k+1)=y2(k)+(h/6)*(K1f2+2*(K2f2+K3f2)+K4f2);
end
end
```

c)

```
%shooting method
clear
clc
format long
a=1  % Valor inicial de x
b=2  % Valor final de x do cominio de calculo
% Condição Inicial:
x(1) =a  % Valor inicial de x
y1(1)=1; % Valor inicial de y1
%y2(1)=?;    % Valor inicial de y2 "desconhecido"
%y1(n+1)=0.5 % Valor final de y1 "conhecido" (CC)
D=0.5
%RK4 p/ sistemas de 2 EDOs
n=8  % Numero de subdivisões do intervalor [a,b] para que o Erro<1e-6
%1ª estimativa
C1=0
C2=2
dif=1;
%Dado C1  Processo iterativo:
y2(1)=C1; % Valor inicial de y2 "desconhecido" atribuido
[x y1 y2]=fRK4sist2a(n,a,b,x,y1,y2);
D1=y1(n+1) %valor incicial de D1
while dif>1e-8
%Dado C2 - Processo iterativo:
       y2(1)=C2; % Valor inicial de y2 desconhecido atribuido
       [x y1 y2]=fRK4sist2a(n,a,b,x,y1,y2);
       D2=y1(n+1)
       C=C1+(C2-C1)*(D-D1)/(D2-D1)
%atualização de valores de C1 e C2:
       C1=C2;   %(descarta C1, substitui por C2)
       D1=D2;   %valor de D1 nao precisa ser recalculado, pois é o proprio D2
       C2=C    %(atualiza C2)
%Atualizados C1 e C2 - Processo iterativo para relcalcular D2:
       dif=abs(C2-C1)
end
%ultimos valores calculados de y1 e y2 são para C correto.
[x2 y1a y2a]=fRK4sist2a(2*n,a,b,x,y1,y2);
erroestimmax3=abs(y1(n+1)-y1a(2*n+1)) %Erro estimado no final no intervalo, no ultimo
ponto
plot(x,y1,'r')
```

(9.6)

a)

$$\begin{cases} y_1'(x) = z_1(x, y_1, y_2, y_3) = y_2, & \text{com } y_1(x=0) = F(x=0) = 0 \\ y_2'(x) = z_2(x, y_1, y_2, y_3) = y_3, & \text{com } y_2(x=0) = F'(x=0) = 0 \\ y_3'(x) = z_3(x, y_1, y_2, y_3) = -(1/2) * y_1 * y_3), & \text{com } y_3(x=0) = F''(x=0) = C = ? \end{cases}$$

O valor conhecido (alvo) da condição de contorno $y_2(n+1) = F'(x=10) = 1$ é definido como $D = 1$, que permite estimar o valor da condição inicial desconhecida $y_3(1) = F''(x=0) = C$, conforme algoritmo a seguir:

```
clear
clc
format long
```

```
% Montando o sistema de EDOs para a equação de Blasius
% y1'(x)=z1(x,y1,y2,y3)= y2            = F(x)
% y2'(x)=z2(x,y1,y2,y3)= y3            = F'(x)
% y3'(x)=z3(x,y1,y2,y3)=((-0.5)*y1*y3) = F"(x)
% y1(0)= F (0)
% y2(0)= F'(0)
% y3(0)= F"(0) = C a ser calculado pelo metodo do 'shooting method'

a=0;
b=10;
% Condicoes iniciais em x=a=0
x(1)=0;
y1(1)=0;
y2(1)=0;
%y3(0)=C  valor desconhecido, mas sabemos y2(n+1)=y2(10)=D=1
tol=1e-8;

n=128;
%Conhecida condicao de contorno em x=b=10
D=1
y2(n+1)=D;

%item a) determinar C via shooting method
       Ci=0.; %valor inicial da CI desconhecida
       C=fCalculaCshooting3(n,a,b,x,y1,y2,Ci,D,tol)
       %item a) determinar C via Newton
       Ci=0.; %valor inicial da CI desconhecida
       C=fNewtonNumC3(n,a,b,x,y1,y2,Ci,D,tol)%f(C)=F(C)-D=0

%item b graficos da solução
       y3(1)=C; %Definido o valor da Condição inicias, pode-se resolver a EDO
       [x y1 y2 y3]=fsis3RK4(n,a,b,x,y1,y2,y3);

%item c graficos da solução
       [xa y1a y2a y3a]=fsis3RK4(2*n,a,b,x,y1,y2,y3);
       ErroEstimado1=abs(y1(n+1)-y1a(2*n+1))     %erro na extremidade x=b
       ErroEstimado2=abs(y1(n/2+1)-y1a(2*n/2+1)) %erro no meio do intervalo x=(a+b)/2

plot(x,y1,"linewidth",2,"--r;y1;",x,y2,"linewidth",2,"-
k;y2;",x,y3,"linewidth",2,"..b;y3;")
```

```
function [x y1 y2 y3]=fsis3RK4(n,a,b,x,y1,y2,y3)
  h=(b-a)/n;
  for k=1:n
    x(k+1)=x(k)+h;
    K1y1=fy1(x(k), y1(k)           , y2(k)           , y3(k));
    K1y2=fy2(x(k), y1(k)           , y2(k)           , y3(k));
    K1y3=fy3(x(k), y1(k)           , y2(k)           , y3(k));
    K2y1=fy1(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3);
    K2y2=fy2(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3);
    K2y3=fy3(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3);
    K3y1=fy1(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3);
    K3y2=fy2(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3);
    K3y3=fy3(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3);
    K4y1=fy1(x(k), y1(k)+h*K3y1    , y2(k)+h*K3y2    , y3(k)+h*K3y3);
    K4y2=fy2(x(k), y1(k)+h*K3y1    , y2(k)+h*K3y2    , y3(k)+h*K3y3);
    K4y3=fy3(x(k), y1(k)+h*K3y1    , y2(k)+h*K3y2    , y3(k)+h*K3y3);
    y1(k+1)=y1(k)+h*(K1y1+2*K2y1+2*K3y1+K4y1)/6;
    y2(k+1)=y2(k)+h*(K1y2+2*K2y2+2*K3y2+K4y2)/6;
    y3(k+1)=y3(k)+h*(K1y3+2*K2y3+2*K3y3+K4y3)/6;
    end % for i
end %function
```

```
function C=fCalculaCshooting3(n,a,b,x,y1,y2,Ci,D,tol)
```

```
C1=Ci; C2=C1+0.1;
k=0;
while (abs(C2-C1)>tol && k<10)
  k=k+1;

  y3(1)=C1;
  [x y1 y2 y3]=fsis3RK4(n,a,b,x,y1,y2,y3);
  D1=y2(n+1);

  y3(1)=C2;
  [x y1 y2 y3]=fsis3RK4(n,a,b,x,y1,y2,y3);
  D2=y2(n+1);

  C=C1+((D-D1)/(D2-D1))*(C2-C1);
  C1=C2; %C2 e C sao considerados os melhores valores, descarta C1
  C2=C;
  dif=max(abs(C2-C1));
end %while
k
end
```

```
function C=fNewtonNumC3(n,a,b,x,y1,y2,Ci,D,tol)%Método da secante:
 dif = 1; k = 0;dx=1e-1;
 C1=Ci;   f1=f10(n,a,b,x,y1,y2,C1,D);
 C2=Ci+dx;
while (dif>tol && k<100)
    k = k + 1; % passo k iterativo
    f2=f10(n,a,b,x,y1,y2,C2,D);
    C  = (C1*f2-C2*f1)/(f2-f1);
    C1=C2;f1=f2;
    C2=C;
    dif=abs(C2-C1);
end
k
dif;
end
```

```
function D1=f10(n,a,b,x,y1,y2,C1,D)
y3(1)=C1; % Valor inicial de y2 "desconhecido", atribuido
[x y1 y2 y3]=fsis3RK4(n,a,b,x,y1,y2,y3);
%y2(n+1) correto é D
D1=y2(n+1)-D; %f(C)=D1-D
end
```

```
function z1=fy1(x,y1,y2,y3)
  z1=y2;
end
```

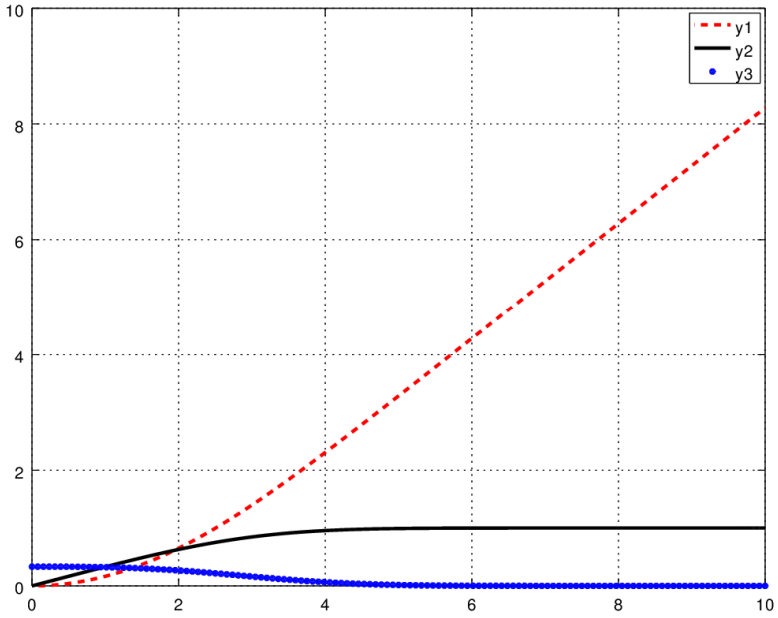
```
function z2=fy2(x,y1,y2,y3)
  z2=y3;
end
```

```
function z3=fy3(x,y1,y2,y3)
  z3=-0.5*y1*y3;
end
```

$C = 0.332057340408670$ obtido pelo *Shooting Method* com 6 iterações.

$C = 0.332057340408708$ obtido pelo método da Secante com 6 iterações.

b)



A linha continua desse gráfico, $y_2(x) = F'(x) = u/U$, representa a espessura da camada do escoamento que é afetada pela placa plana. Em $x = 5$, a fração da velocidade horizontal local u atinge 99% da velocidade paralela U longe da placa (no infinito), $y_2(x=5) = F'(x=5) = 0.99$.

c)

Erro Estimado $y_1(x=b) = 1.18594414288964\text{e-}08$ no final do intervalo.

Erro Estimado $y_1(x=(a+b)/2) = 2.40767885628657\text{e-}08$ no meio do intervalo.

(9.7)

a)

Geramos 4 PVIs:

$$\begin{cases} y_1(x) = F(x), & y_1'(x) = z_1(x, y_1, y_2, y_3) = y_2, & y_1(x=1) = F(x=1) = 0 \\ y_2(x) = F'(x), & y_2'(x) = z_2(x, y_1, y_2, y_3) = y_3, & y_2(x=1) = F'(x=1) = 1 \\ y_3(x) = F''(x), & y_3'(x) = z_3(x, y_1, y_2, y_3) = y_4 & y_3(x=1) = F''(x=1) = C_1 = ? \\ y_4(x) = F'''(x), & y_4'(x) = z_4(x, y_1, y_2, y_3) = -6/x^4, & y_4(x=1) = F'''(x=1) = C_2 = ? \end{cases}$$

E as duas funções das condições iniciais desconhecidas são montadas com os valores das condições conhecidas $y_1(x=2) = F(x=2) = D_1$ e $y_2(x=2) = F'(x=2) = D_2$, logo:

$g_1(C_1) = y_1(x=2) - D_1 = 0$, onde $y_1(x=2)$ é calculado para cada C_1 e C_2 através das EDO's
$g_2(C_2) = y_2(x=2) - D_2 = 0$, onde $y_2(x=2)$ é calculado para cada C_1 e C_2 através das EDO's

Com $n = 2^{12} = 4096$ subintervalos

A partir de $Ci = [0 \;\; 0]$, após 4 iterações e critério interno *1e-14*, atingimos

$C = [-1.00006105503037 \;\; 2.00054939090157]$

Erro exato max de $F(x)$ = 1.29018938982428e-06 em x=1.45751953125000

Erro estimado max de $F(x)$ = 2.54271684219098e-06

b)

```
%Problema de valor no contorno
%F''''+6x^(-4)=0 com solução exata F(x)=ln(x)
clear
clc
format long
a=1  % Valor inicial de x
b=2  % Valor final de x do cominio de calculo
% Condição Inicial:
x(1) =a;  % Valor inicial de x
y1(1)=0; % Valor inicial de y1
y2(1)=1; % Valor inicial de y2

%%-------------------------------------------------------------------------
%determinação das 2 Condições iniciais desconhecidas
%y3(1)=C(1)=?; % Valor inicial (C.I.) de y3 "desconhecido"
%y4(1)=C(2)=?; % Valor inicial (C.I.) de y4 "desconhecido"

%y1(n+1)=ln(2) % Valor final(C.C.) de y1 "conhecido" = D(1)
%y2(n+1)=0.5   % Valor final(C.C.) de y2 "conhecido" = D(2)
D(1)=log(2)
D(2)=0.5
%RK4 p/ sistemas de 4 EDOs
       %Vamos estabelecer o n mínimo para que a solução y1 tenha erromaximo<tolerancia
       n=2^12  % Numero de subdivisões do intervalor [a,b], tentativas
%%-------------------------------------------------------------------------
tolerancia=1e-14%para que erro maximo seja 1e-6 na solução final F.
%via metodo de Newton numerico para deternimar C(1) e C(2) das eqs.
%h1(C1)=y1(n+1)-D1=0
%h2(C2)=y1(n+1)-D2=0
Ci=[0 0] %valores iniciais das CI's y3(1)=C(1)=? e y4(1)=C(2)=?
C=fNewtonNumC4(n,a,b,x,y1,y2,Ci(1),Ci(2),D,tolerancia)%h(C)=f(C)-D=0
%%-------------------------------------------------------------------------

%Adotando o valor de C determinado acima por um dos metodos, continuamos a solucao
y3(1)=C(1);%ultimo valor calculado de C é o correto p/ y3(1), mas ainda depende de n.
y4(1)=C(2);%ultimo valor calculado de C é o correto p/ y4(1), mas ainda depende de n.
[x y1 y2 y3 y4]=fRK4sist4(n,a,b,x,y1,y2,y3,y4);
ye=log(x);%Valor exato p/ comparação
[erroFexatomax1 i]=max(abs(y1.-ye)) %Erro exato maximo em todo o dominio de x (para
aferição)
posicaoerroFexatomax=x(i)
[xa y1a y2a y3a y4a]=fRK4sist4(2*n,a,b,x,y1,y2,y3,y4); %valor mais proximo dom exato
erroFestimmax2=abs(y1(n/2+1)-y1a((2*n/2)+1)) %Erro estimado no meio intervalo.
%% algoritmo de busca opcional
```

```
plot(x,ye,'..k',x,y1,'-r')
```

```
function C=fNewtonNumC4(n,a,b,x,y1,y2,y3,y4,D,tol)%Método da Newton NUmerico:
%Para 2 eqs nao lineares
Xi(1)=y3(1); %Ci(1) condicao inicial desconhecida p/ y3
Xi(2)=y4(1); %Ci(2) condicao inicial desconhecida p/ y4
criterio=1;contador=0;
for i=1:2 Dx(i)=1e-6; end
while (criterio>tol && contador<50)
    contador=contador+1;
    %Gerando a jacobiana
    Xj=Xi;
    y3(1)=Xi(1); %Ci(1) condicoes iniciais desconhecidas
    y4(1)=Xi(2); %Ci(2) condicoes iniciais desconhecidas
    Yi=H(n,a,b,x,y1,y2,y3,y4,D); %Valor inicial

    for j=1:2 %varre as 2 colunas
        Xj(j)=Xi(j)+Dx(j); %incremento só na coluna j
        y3(1)=Xj(1); %condicao incrementada p/ y3 qdo j=1
        y4(1)=Xj(2); %condicao incrementada p/ y4 qdo j=2
%            for i=1:n
               Yj=H(n,a,b,x,y1,y2,y3,y4,D);
               A(:,j)=(Yj .- Yi)/Dx(j);
%            end
           Xj=Xi;% volta ao valor original
    end
    A(:,2+1)=-Yi;
    Dx=fgausspivparcial(2,A);
    X=Xi+Dx;
    Xi=X;
    criterio=min(abs(Dx));
end
C=X;
contador
criterio
end
```

```
function [x y1 y2 y3 y4]=fRK4sist4(n,a,b,x,y1,y2,y3,y4)
  h=(b-a)/n;
  for k=1:n
    x(k+1)=x(k)+h;
    K1y1=fh1(x(k), y1(k)          , y2(k)          , y3(k)          ,     y4(k));
    K1y2=fh2(x(k), y1(k)          , y2(k)          , y3(k)          ,     y4(k));
    K1y3=fh3(x(k), y1(k)          , y2(k)          , y3(k)          ,     y4(k));
    K1y4=fh4(x(k), y1(k)          , y2(k)          , y3(k)          ,     y4(k));

    K2y1=fh1(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3, y4(k)+0.5*h*K1y4);
    K2y2=fh2(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3, y4(k)+0.5*h*K1y4);
    K2y3=fh3(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3, y4(k)+0.5*h*K1y4);
    K2y4=fh4(x(k), y1(k)+0.5*h*K1y1, y2(k)+0.5*h*K1y2, y3(k)+0.5*h*K1y3, y4(k)+0.5*h*K1y4);

    K3y1=fh1(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3, y4(k)+0.5*h*K2y4);
    K3y2=fh2(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3, y4(k)+0.5*h*K2y4);
    K3y3=fh3(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3, y4(k)+0.5*h*K2y4);
    K3y4=fh4(x(k), y1(k)+0.5*h*K2y1, y2(k)+0.5*h*K2y2, y3(k)+0.5*h*K2y3, y4(k)+0.5*h*K2y4);

    K4y1=fh1(x(k), y1(k)+    h*K3y1, y2(k)+    h*K3y2, y3(k)+    h*K3y3, y4(k)+    h*K3y4);
    K4y2=fh2(x(k), y1(k)+    h*K3y1, y2(k)+    h*K3y2, y3(k)+    h*K3y3, y4(k)+    h*K3y4);
    K4y3=fh3(x(k), y1(k)+    h*K3y1, y2(k)+    h*K3y2, y3(k)+    h*K3y3, y4(k)+    h*K3y4);
    K4y4=fh4(x(k), y1(k)+    h*K3y1, y2(k)+    h*K3y2, y3(k)+    h*K3y3, y4(k)+    h*K3y4);

    y1(k+1)=y1(k)+h*(K1y1+2*K2y1+2*K3y1+K4y1)/6;
    y2(k+1)=y2(k)+h*(K1y2+2*K2y2+2*K3y2+K4y2)/6;
    y3(k+1)=y3(k)+h*(K1y3+2*K2y3+2*K3y3+K4y3)/6;
    y4(k+1)=y4(k)+h*(K1y4+2*K2y4+2*K3y4+K4y4)/6;
    end % for i
end %function
```

```
function Y=H(n,a,b,x,y1,y2,y3,y4,D)
    [x y1 y2 y3 y4]=fRK4sist4(n,a,b,x,y1,y2,y3,y4); %resolve o sistema uma vez para
    %y3(1)=C(1) e y4(1)=C(2)
    Y(1)=y1(n+1)-D(1);
    Y(2)=y2(n+1)-D(2);
end
```

```
function z=fh1(x,y1,y2,y3,y4)
  z=y2;
end
```

```
function z=fh2(x,y1,y2,y3,y4)
  z=y3;
end
```

```
function z=fh3(x,y1,y2,y3,y4)
  z=y4;
end
```

```
function z=fh4(x,y1,y2,y3,y4)
  z=-6./x.^4;
end
```