Cálculo Numérico — Algoritmos

Valdenir de Souza Junior

Abril de 2007

Sumário

1 Introdução

O presente documento traz os algoritmos dos métodos numéricos mostrados em sala, a serem implementado na disciplina Cálculo Numérico.

Os algoritmos devem ser implementados usando o GNU Octave. Os trabalhos deverão ser apresentados por escrito, constando o código fonte da implementação e a execução de dois exemplos, para cada método implementado.

2 Raízes de Equações

Nos algoritmos para encontrar raízes de equações, mostrados a seguir, adotou-se um número de iterações máximo de 90. Esse limite é importante para evitar

que o programa entre em *looping*, caso a convergência não seja atingida. Evidentemente a quantidade limite para as iterações pode variar e depende da precisão de cálculo requerida.

Embora não seja mostrado nos algoritmos, na implementação é importante mostrar os resultados parciais, como as tabelas que são construídas ao resolver o método sem a ajuda de um programa. Isso ajuda no acompanhamento e verificação dos resultados. Em algumas situações o método pode levar a uma divergência momentânea, para depois conduzir à solução. Esse comportamento só pode ser acompanhado se os resultados parciais forem mostrados.

Nos métodos da bisseção, falsa posição e método da secante, os valores dos extremos do intervalo de pesquisa, a e b, devem ser tais que $f(a) \times f(b) < 0$, de forma a assegurar haja raiz no intervalo, para o caso de funções contínuas no intervalo. Se essa condição não for atendida, o algoritmo é interrompido.

2.1 Método da Bisseção

Algoritmo: Método da bisseção

declare
  a, b, {limites inferior e superior do intervalo}
  $\bar{x}$, {valor médio entre a e b}
  ε, {precisão requerida}
  n {contador de iterações}
numérico
declare $f(x)$ **função numérica**

leia a, b, ε

se $f(a) \times f(b) \geq 0$ **então**
  escreva "Valores de a e b inválidos"
  interrompa o algoritmo
fim se

$n \leftarrow 1$
repita
  $\bar{x} \leftarrow \dfrac{b-a}{2}$
  se $|f(\bar{x})| < \varepsilon$ **ou** $n > 90$ **então**
    interrompa
  fim se
  se $f(\bar{x}) \times f(b) < 0$ **então**
    $a \leftarrow \bar{x}$
  senão
    $b \leftarrow \bar{x}$
  fim se
  $n \leftarrow n + 1$
fim repita

se $n > 90$ **então**
  escreva "Convergência não obtida"
senão
  escreva $\bar{x}$, $f(\bar{x})$
fim se

Algoritmo 1: Método da bisseção para raízes

2.2 Método de Newton-Raphson

Algoritmo: Método de Newton-Raphson

declare
  x, {aproximação para a raiz, a cada iteração}
  ε, {precisão requerida}
  n {contador de iterações}
numérico
declare
  $f(x)$, {função que se quer achar a raiz}
  $df(x)$ {derivada da função $f(x)$}
função numérica

leia x, ε

$n \leftarrow 1$
repita
  $x \leftarrow x - \dfrac{f(x)}{df(x)}$
  se $|f(x)| < \varepsilon$ **ou** $n > 90$ **então**
    interrompa
  fim se
  $n \leftarrow n + 1$
fim repita

se $n > 90$ **então**
  escreva "Convergência não obtida"
senão
  escreva x, $f(x)$
fim se

Algoritmo 2: Método de Newton-Raphson para raízes

2.3 Método da Falsa Posição (Método das Cordas)

Algoritmo: Método da falsa posição

declare
  a, b, {limites inferior e superior do intervalo}
  $\bar{x}$, {média ponderada entre a e b}
  ε, {precisão requerida}
  n {contador de iterações}
numérico
declare $f(x)$ **função numérica**

leia a, b, ε

se $f(a) \times f(b) \geq 0$ **então**
  escreva "Valores de a e b inválidos"
  interrompa o algoritmo
fim se

$n \leftarrow 1$
repita
  $\bar{x} \leftarrow \dfrac{a.f(b) - b.f(a)}{f(b) - f(a)}$
  se $|f(\bar{x})| < \varepsilon$ **ou** $n > 90$ **então**
    interrompa
  fim se
  se $f(\bar{x}) \times f(b) < 0$ **então**
    $a \leftarrow \bar{x}$
  senão
    $b \leftarrow \bar{x}$
  fim se
  $n \leftarrow n + 1$
fim repita

se $n > 90$ **então**
  escreva "Convergência não obtida"
senão
  escreva $\bar{x}$, $f(\bar{x})$
fim se

Algoritmo 3: Método da falsa posição para raízes

2.4 Método da Secante

Algoritmo: Método da secante

declare
  a, b, {limites inferior e superior do intervalo}
  c, {aproximação para a raiz}
  ε, {precisão requerida}
  n {contador de iterações}
  numérico
declare $f(x)$ **função numérica**

leia a, b, ε

se $f(a) \times f(b) \geq 0$ **então**
  escreva "Valores de a e b inválidos"
  interrompa o algoritmo
fim se

$n \leftarrow 1$
repita
  $c \leftarrow b - \dfrac{f(b).(b-a)}{f(b)-f(a)}$
  se $|f(c)| < \varepsilon$ **ou** $n > 90$ **então**
    interrompa
  fim se
  $a \leftarrow b$
  $b \leftarrow c$
  $n \leftarrow n + 1$
fim repita

se $n > 90$ **então**
  escreva "Convergência não obtida"
senão
  escreva c, $f(c)$
fim se

Algoritmo 4: Método da secante para raízes

3 Interpolação

Nos algoritmos para interpolação a seguir, o método de Newton usa o algoritmo para contrução da tabela de diferenças divididas, mostrado à parte. Os pontos (x, y) conhecidos devem ser lidos por meio de uma matriz na forma

$$\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

Os algoritmos dos métodos de Lagrange e de Newton mostrados lêem a tabela e um valor de x, calculam e escrevem o correspondente valor de y interpolado.

3.1 Método de Lagrange

Algoritmo: Interpolação com Método de Lagrange

declare

$T_{m\times 2}$,	{tabela com os pontos (x, y) conhecidos }
x,	{abcissa do ponto interpolado (valor lido)}
y,	{ordenada do ponto interpolado (valor calculado)}
P,	{produtório}
m	{números de pontos conhecidos}
i, j	{variáveis auxiliares}

numérico

leia m, T, x

$y \leftarrow 0$
para i **de** 1 **até** m **faça**
  $P \leftarrow 1$
  para j **de** 1 **até** m **faça**
    se $i \neq j$ **então**
      $P \leftarrow P \times \dfrac{x - T(j,1)}{T(i,1) - T(j,1)}$
    fim se
  fim para
  $y \leftarrow y + P \times T(i,2)$
fim para

escreva y

Algoritmo 5: Interpolação pelo método de Lagrange

3.2 Método de Newton, com Diferenças Divididas

Algoritmo: Tabela de diferenças divididas

declare
- $T_{m\times 2}$, {tabela com os pontos (x, y) conhecidos }
- $F_{m\times m}$, {tabela com as diferenças divididas }
- m {números de pontos conhecidos}
- i, j, p, k {variáveis auxiliares}

numérico

leia m, T

$F \leftarrow 0$ {zera toda a matriz}
para i **de** 1 **até** m **faça**
  $F(i, 1) \leftarrow T(i, 2)$
fim para

$p \leftarrow m - 1$
$k \leftarrow 1$
para j **de** 2 **até** m **faça**
  para i **de** 1 **até** p **faça**
    $F(i, j) \leftarrow \dfrac{F(i + 1, j - 1) - F(i, j - 1)}{T(i + k, 1) - T(i, 1)}$
  fim para
  $p \leftarrow p - 1$
  $k \leftarrow k + 1$
fim para

escreva F

Algoritmo 6: Construção da tabela de diferenças divididas

Algoritmo: Interpolação com Método de Newton, com diferenças divididas

declare
- $T_{m\times 2}$, {tabela com os pontos (x, y) conhecidos }
- x, {abcissa do ponto interpolado (valor lido)}
- y, {ordenada do ponto interpolado (valor calculado)}
- P, {produtório}
- m {números de pontos conhecidos}
- i, j {variáveis auxiliares}

numérico

leia m, T, x
Calcule $F_{m\times m}$, usando algoritmo anterior

$P \leftarrow 1$
$y \leftarrow T(1, 2)$
para i **de** 2 **até** m **faça**
  $P \leftarrow P \times (x - T(i - 1, 1))$
  $y \leftarrow y + P \times F(1, i)$
fim para

escreva y

Algoritmo 7: Interpolação pelo método de Newton

4 Sistemas de Equações Lineares

Os métodos de solução de sistemas lineares apresentadados dividem-se entre métodos diretos e métodos iterativos. Nos métodos diretos, as solução do sistema de equações é feita com um número finito de operações. Nessa classe incluem-se os métodos de Gauss e a decomposição LU. Nos métodos iterativos, são calculadas seqüências de aproximações para o vetor-solução $\{x\}$ do sistema de equações, a partir de uma aproximação inicial. Nessa classe incluem-se os métodos de Jacobi e Gauss-Seidl.

O algoritmo de retrosubstituição a seguir deve ser usado como etapa final do método de Gauss e como parte da solução no método LU.

Algoritmo: Retrosubstituição

declare

$A_{n\times n}$,	{matriz dos coeficientes, a ser lida }
b_n,	{vetor dos termos independentes }
x_n,	{vetor-solução }
n,	{número de equações do sistema }
k, j	{variáveis auxiliares}

numérico

leia $n, A_{n\times n}, b_n$

$x(n) \leftarrow \dfrac{b(n)}{A(n,n)}$

para k **de** $n-1$ **até** 1 **faça**

$\quad x(k) \leftarrow \left[b(k) - \sum_{j=k+1}^{n} A(k,j) \times x(j)\right] \div A(k,k)$

fim para

escreva x_n

Algoritmo 8: Solução de um sistema triangular superior por retrosubstituição

4.1 Método de Gauss

O algoritmo do método de Gauss mostrado não conta com estratégia alguma de pivoteamento. Os possíveis problemas que podem advir de tal simplificação foram discutidos em sala. O algoritmo irá atuar sobre a matriz aumentada do sistema, $[D] = [A \vdots b]$.

Algoritmo: Método de Gauss

declare
$D_{n\times n+1}$, {matriz aumentada, a ser lida }
x_n, {vetor-solução }
n {número de equações do sistema }
m, {multiplicador }
f, {fase de solução }
i, j {linha e coluna de D }
numérico

leia $n, D_{n\times n+1}$

{Escalonamento da matriz D }
para f **de** 1 **até** $n-1$ **faça**
  para i **de** $(f+1)$ **até** n **faça**
    $m \leftarrow -D(i,f)/D(f,f)$
    para j **de** 1 **até** $n+1$ **faça**
      $D(i,j) \leftarrow D(i,j) + m \times D(f,j)$
    fim para
  fim para
fim para

{Retrosubstituição }
$x(n) \leftarrow \dfrac{D(n,n+1)}{D(n,n)}$
para i **de** $n-1$ **até** 1 **faça**
  $x(i) \leftarrow \left[D(i,n+1) - \sum_{j=i+1}^{n} D(i,j) \times x(j)\right] \div D(i,i)$
fim para

{Escrita de resultados }
escreva x

Algoritmo 9: Método de Gauss para solução de um sistema linear

4.2 Método LU

O algoritmo do método LU, mostrado a seguir tem três partes principais. Na primeira parte é feita a decomposição da matriz dos coeficientes, na forma $[A] = [L][U]$. Na segunda parte é resolvido o sistema $[L]\{y\} = \{b\}$, por substituição. Na terceira parte é feita a retrosubstituição, de forma a resolver o sistema $[U]\{x\} = \{y\}$, de forma a encontrar o vetor solução x, do sistema linear.

Algoritmo: Método LU

declare

$A_{n\times n}$,	{matriz dos coeficientes, a ser lida }
$L_{n\times n}$,	{matriz triangular inferior, com 1 na diagonal principal }
$U_{n\times n}$,	{matriz triangular superior }
x_n,	{vetor-solução }
y_n,	{vetor-solução auxiliar }
n,	{número de equações do sistema }
i, j, k	{variáveis auxiliares }

numérico

leia $n, A_{n\times n}, b_n$

{Decomposição de $[A]$ }
$L_{n\times n} \leftarrow I_{n\times n}$ {Faz L igual à matriz identidade }
$U_{n\times n} \leftarrow 0_{n\times n}$ {Faz U igual à matriz nula }
$U(1,1) \leftarrow A(1,1)$
para i **de** 2 **até** n **faça**
  para j **de** 1 **até** $(i-1)$ **faça**
    $L(i,j) \leftarrow \left[A(i,j) - \sum_{k=1}^{j-1} L(i,k) \times U(k,j)\right] \div U(j,j)$
    $U(j,i) \leftarrow \left[A(j,i) - \sum_{k=1}^{j-1} L(j,k) \times U(k,i)\right]$
    se $i - j = 1$ **então**
      $U(i,i) \leftarrow \left[A(i,i) - \sum_{k=1}^{i-1} L(i,k) \times U(k,i)\right]$
    fim se
  fim para
fim para

{Resolve $[L]\{y\} = \{b\}$ }
$y(1) \leftarrow \dfrac{b(1)}{L(1,1)}$
para k **de** 2 **até** n **faça**
  $y(k) \leftarrow \left[b(k) - \sum_{j=1}^{k-1} L(k,j) \times y(j)\right] \div L(k,k)$
fim para

{Resolve $[U]\{x\} = \{y\}$
$x(n) \leftarrow \dfrac{y(n)}{U(n,n)}$
para k **de** $n-1$ **até** 1 **faça**
  $x(k) \leftarrow \left[y(k) - \sum_{j=k+1}^{n} U(k,j) \times x(j)\right] \div U(k,k)$
fim para

{Escrita de resultados }
escreva x

Algoritmo 10: Método LU para solução de um sistema linear

4.3 Método de Choleski

No método de Choleski, a matriz dos coeficientes é decomposta na forma $[A] = [U]^T[U]$. A solução do sistema a partir da matriz decomposta, é semelhante ao usado no método LU, resolvendo-se o sistema $[U]^T\{y\} = \{b\}$, por

substituição e em seguida resolvendo-se $[U]\{x\} = \{y\}$, de forma a encontrar o vetor solução x, do sistema linear. No algoritmo a seguir é mostrada apenas a parcela de decomposição da matriz. As etapas seguintes, para a resolução do sistema são semelhantes às mostradas no algoritmo do método LU, substituindo $[L]$ por $[U]^T$.

Algoritmo: Fatoração de Uma Matriz pelo Método de Choleski
declare
$A_{n\times n}$, {matriz dos coeficientes, a ser lida }
$U_{n\times n}$, {matriz triangular superior }
n, {número de equações do sistema }
i, j, k {variáveis auxiliares }
numérico

leia $n, A_{n\times n}$

$U_{n\times n} \leftarrow 0_{n\times n}$ {Faz U igual à matriz nula }
$U(1,1) \leftarrow A(1,1)$
para i **de** 1 **até** n **faça**
  $U(i,i) \leftarrow \sqrt{A(i,i) - \sum_{k=1}^{i-1}(U(k,i))^2}$
  se $i < n$ **então**
    para j **de** $(i+1)$ **até** n **faça**
      $U(i,j) \leftarrow \left[A(i,j) - \sum_{k=1}^{i-1} U(k,j) \times U(k,i)\right] \div U(i,i)$
    fim para
  fim se
fim para

escreva U

Algoritmo 11: Algoritmo para decomposição de uma matriz pelo método de Choleski, para solução de um sistema linear

4.4 Método de Jacobi

O método de Jacobi para a solução de sistemas de equações lineares é um método iterativo, onde parte-se de uma aproximação inicial e refina-se a solução, até que a diferença entre o vetor-solução gerado pela iteração corrente ($x^{(k)}$) e o gerado pela iteração anterior ($x^{(k-1)}$) seja pequena.

Para medir a diferença entre o vetor, uma abordagem é medir a distância entre $x^{(k)}$ e $x^{(k-1)}$, por

$$M^{(k)} = \max_{1\leq i\leq n} |x_i^{(k)} - x_i^{(k-1)}|$$

Analogamente, pode-se usar como critério de parada o teste do erro relativo, dado por

$$M_r^{(k)} = \frac{M^{(k)}}{\max_{1\leq i\leq n} |x_i^{(k)}|}$$

Deve ser colocado no algoritmo um limite máximo para o número de iterações, a fim de evitar o *looping*, caso não haja convergência.

Ao implementar os métodos de Jacobi, aqui mostrado, e de Gauss-Seidl, mostrado na seção a seguir, é importante imprimir os resultados parciais de x, e de $M^{(k)}$ de forma a acompanhar a convergência do método.

Algoritmo: Método de Jacobi

declare

$A_{n\times n}$,	{matriz dos coeficientes, a ser lida }
b_n,	{vetor dos termos independentes }
x_n,	{vetor-solução }
$(x_{\text{ant}})_n$,	{vetor-solução obtido na iteração anterior }
ε,	{tolerância de cálculo }
$n_{\max}$,	{quantidade máxima de iterações }
i, k	{variáveis auxiliares }

numérico

leia $n, A_{n\times n}, b_n, (x_{\text{ant}})_n, \varepsilon, n_{\max}$

$x \leftarrow 0_n$ {zera os termos de x }
$k \leftarrow 1$
repita
  para i **de** 1 **até** n **faça**
    $x(i) \leftarrow \left[b(i) - \sum_{\substack{j=1 \\ j\neq i}}^{n} A(i,j) \times x_{\text{ant}}(j) \right] \div A(i,i)$
  fim para
  se $k > n_{max}$ **ou** $\max_{1\leq i\leq n} |x(i) - x_{\text{ant}}(i)| < \varepsilon$ **então**
    interrompa
  fim se
  $x_{\text{ant}} \leftarrow x$
  $k \leftarrow k + 1$
fim repita

escreva x

Algoritmo 12: Método de Jacobi, para solução de um sistema linear

4.5 Método de Gauss-Seidl

O método de Gauss-Seidl é uma variação do método de Jacobi, onde para o cálculo do termo i do vetor-solução x da iteração corrente, os termos já calculados, anteriores a i, são usados no cálculo.

Algoritmo: Método de Jacobi

declare
$A_{n\times n}$, {matriz dos coeficientes, a ser lida }
b_n, {vetor dos termos independentes }
x_n, {vetor-solução }
$(x_{\text{ant}})_n$, {vetor-solução obtido na iteração anterior }
ε, {tolerância de cálculo }
$n_{\max}$, {quantidade máxima de iterações }
i, k {variáveis auxiliares }
numérico

leia $n, A_{n\times n}, b_n, (x_{\text{ant}})_n, \varepsilon, n_{\max}$

$x \leftarrow 0_n$ {zera os termos de x }
$k \leftarrow 1$
repita
  para i **de** 1 **até** n **faça**
    $x(i) \leftarrow$
    $\left[b(i) - \sum_{j=1}^{i-1} A(i,j) \times x(j) - \sum_{j=i+1}^{n} A(i,j) \times x_{\text{ant}}(j)\right] \div A(i,i)$
  fim para
  se $k > n_{max}$ **ou** $\max_{1\leq i\leq n} |x(i) - x_{\text{ant}}(i)| < \varepsilon$ **então**
    interrompa
  fim se
  $x_{\text{ant}} \leftarrow x$
  $k \leftarrow k + 1$
fim repita

escreva x

Algoritmo 13: Método de Gauss-Seidl, para solução de um sistema linear

5 Integração

A seguir são mostrados os algoritmos de integração numérica de funções de uma variável, pela regra do trapézio, primeira regra de Simpson e segunda regra de Simpson.

5.1 Regra do Trapézio

Algoritmo: Regra do Trapézio

declare

$a, b,$	{limites inferior e superior de integração }
$n,$	{quantidade de subintervalos }
$h,$	{incremento em x }
$x,$	{variável independente }
$S,$	{somatório }
$I,$	{resultado da integração }
i	{variável auxiliar }

numérico

declare $f(x)$ **função numérica**

leia a, b, n

$h \leftarrow \dfrac{b-a}{n}$

$x \leftarrow a$

$S \leftarrow 0$

para i **de** 1 **até** $n-1$ **faça**

  $x \leftarrow x + h$

  $S \leftarrow S + 2 \times f(x)$

fim para

$S \leftarrow f(a) + S + f(b)$

$I \leftarrow \dfrac{h}{2} \times S$

escreva I

Algoritmo 14: Integração pela Regra do Trapézio

5.2 Primeira Regra de Simpson

Algoritmo: Regra do Trapézio

declare
- $a, b,$ {limites inferior e superior de integração }
- $n,$ {quantidade de subintervalos }
- $h,$ {incremento em x }
- $x,$ {variável independente }
- $S,$ {somatório }
- $I,$ {resultado da integração }
- i {variável auxiliar }

numérico

declare $f(x)$ **função numérica**

leia a, b, n {n deverá ser par }

$h \leftarrow \dfrac{b-a}{n}$

$x \leftarrow a$

$S \leftarrow 0$

para i **de** 1 **até** $(n-1)/2$ **faça**

$x \leftarrow x + h$

$S \leftarrow S + 4 \times f(x)$

$x \leftarrow x + h$

$S \leftarrow S + 2 \times f(x)$

fim para

$x \leftarrow x + h$

$S \leftarrow f(a) + S + 4 \times f(x) + f(b)$

$I \leftarrow \dfrac{h}{3} \times S$

escreva I

Algoritmo 15: Integração pela Primeira Regra de Simpson

5.3 Segunda Regra de Simpson

Algoritmo: Segunda Regra de Simpson
declare
$a, b,$ {limites inferior e superior de integração }
$n,$ {quantidade de subintervalos }
$h,$ {incremento em x }
$x,$ {variável independente }
$S,$ {somatório }
$I,$ {resultado da integração }
i {variável auxiliar }
numérico
declare $f(x)$ **função numérica**
leia a, b, n {n deverá ser múltiplo de 3 }
$h \leftarrow \dfrac{b-a}{n}$
$x \leftarrow a$
$S \leftarrow 0$
para i **de** 1 **até** $(n-3)/3$ **faça**
  $x \leftarrow x + h$
  $S \leftarrow S + 3 \times f(x)$
  $x \leftarrow x + h$
  $S \leftarrow S + 3 \times f(x)$
  $x \leftarrow x + h$
  $S \leftarrow S + 2 \times f(x)$
fim para
$x \leftarrow x + h$
$S \leftarrow S + 3 \times f(x)$
$x \leftarrow x + h$
$S \leftarrow f(a) + S + 3 \times f(x) + f(b)$
$I \leftarrow \dfrac{3h}{8} \times S$
escreva I

Algoritmo 16: Integração pela Segunda Regra de Simpson