UNIVERSIDADE FEDERAL DO ABC

BC1419 Cálculo Numérico (1º 2020) - LISTA 1
Entrega: 02/04/2020
Prof. André Pierro de Camargo

1 Orientações gerais

- A linguagem de programação utilizada na implementação dos métodos é de livre escolha.
- Peça ajuda ao professor sempre que necessário.
- Realize testes preliminares com cada método para identificar possíveis erros de implementação. Por exemplo: teste o método de eliminação de Gauss em sistemas de equações cuja solução seja previamente conhecida e veja se o resultado obtido é coerente.
- Discuta com os outros grupos e também com o professor caso encontre resultados aparentemente sem sentido. Isso pode (ou não) ser um erro de implementação, ou mesmo um indicativo de que o método utilizado possui as suas limitações.

Cada grupo deverá entregar

- Um relatório contendo a resolução dos exercícios teóricos e **TUDO** o que foi solicitado nos exercícios práticos (leiam a descrição com bastante atenção).
- O arquivo contendo o código fonte utilizado nos exercícios práticos.

Enviar o material por e-mail para andre.camargo@ufabc.edu.br, especificando o número do grupo.

Observação: Na Seção 3 encontram-se alguns dos algoritmos que serão utilizados nos exercícios práticos.

2 Exercício prático

Na tabela abaixo encontram-se as medidas da distância percorrida (x) por um objeto em queda livre em função do tempo de queda (t).

i =	0	1	2	3	4	5	6	7	8	9
t_i (1/60) s	0	1	2	3	4	5	6	7	8	9
x_i (cm)	0.0	1	2.4	4.1	6	8.2	10.6	13.4	16.4	19.7
i =	10	11	12	13	14	15	16	17	18	19
t_i (1/60) s	10	11	12	13	14	15	16	17	18	19
x_i (cm)	23.3	27	31.2	35.5	40.1	45	50.2	55.6	61.3	67.3
i =	20	21	22	23	24	25	26	27	28	29
t_i (1/60) s	20	21	22	23	24	25	26	27	28	29
x_i (cm)	73.6	80.1	86.9	94	101.3	109	116.9	125	133.4	142.1

Tabela 1: Dados experimentais sobre o movimento de um corpo em queda livre (cedido pelo IF-USP).

Para fins de obter um modelo de previsão, isto é, uma função $x(t)$ que represente o movimento do corpo de forma satisfatória, iremos utilizar Splines cúbicos.

Dado um conjunto de dados $\Delta : \{(t_0, x_0), (t_1, x_1), \ldots, (t_n, x_n)\}$, um Spline cúbico interpolador de Δ é uma função $S_\Delta : \mathbb{R} \to \mathbb{R}$ tal que

- S_Δ é duas vezes continuamente diferenciável.
- Para cada $i = 0, 1, \ldots, n-1$, S_Δ coincide com um polinômio de grau menor ou igual a 3 em $[t_i, t_{i+1}]$.
- $S_\Delta(t_i) = x_i,\ i = 0, 1, \ldots, n$.

É possível mostrar (veja, por exemplo, *Introduction to Numerical Analysis* de Stoer. J., and Bulirsch, R.) que, dados números x_0'' e x_n'', existe um único Spline cúbico interpolador de Δ cujas derivadas segundas satisfazem

$$S_\Delta(t_0)'' = x_0'' \text{ e } S_\Delta(t_n)'' = x_n''.$$

Para calcular $S_\Delta(t)$, utilizamos a expressão polinomial de $S_\Delta(t)$ em cada um dos intervalos $[x_i, x_{i+1}],\ i = 0, 1, \ldots, n-1$. Defina

$$h_{i+1} := t_{i+1} - t_i,\ i = 0, 1, \ldots, n-1.$$

Vale que $S_\Delta(t)\ =$

$$\frac{M_i}{6h_{i+1}}(t_{i+1} - t)^3 + \frac{M_{i+1}}{6h_{i+1}}(t - t_i)^3 + A_i(t - t_i) + B_i, \quad \text{para } t \text{ em } [t_i, t_{i+1}], \tag{1}$$

sendo M_i, A_i e B_i constantes que dependem de Δ e podem ser determinadas da seguinte forma: $M_0, M_1, \ldots, M_n$ formam o vetor solução do seguinte sistema de equações

$$\underbrace{\begin{bmatrix} 2 & \lambda_0 & 0 & \ldots & \ldots & 0 \\ \mu_1 & 2 & \lambda_1 & \ldots & \ldots & 0 \\ 0 & \mu_2 & 2 & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & \lambda_{n-1} \\ 0 & \ldots & \ldots & \ldots & \mu_n & 2 \end{bmatrix}}_{A} \times \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}, \tag{2}$$

sendo $\lambda_0 = 0,\ \mu_n = 0,\ d_0\ =\ 2x_0'',\ d_n = 2x_n''$ e

$$\begin{cases} \mu_i \ = \ \dfrac{h_i}{h_i + h_{i+1}}, \quad \lambda_i \ = \ \dfrac{h_{i+1}}{h_i + h_{i+1}}, \\ d_i \ = \ \frac{6}{h_i + h_{i+1}} \left(\frac{x_{i+1} - x_i}{h_{i+1}} - \frac{x_i - x_{i-1}}{h_i} \right), \end{cases}$$

$i = 1, 2, \ldots, n-1$. As constantes A_i e B_i são calculadas a partir de $M_0, M_1, \ldots, M_n$:

$$\begin{cases} B_i \ = \ y_i - \dfrac{M_i}{6} h_{i+1}^2 \\ A_i \ = \ \dfrac{x_{i+1} - x_i}{h_{i+1}} - \dfrac{M_{i+1} - M_i}{6} h_{i+1},\ i = 0, 1, \ldots, n-1. \end{cases}$$

Supondo que $S_\Delta(t)$ represente fielmente o movimento do corpo, qual seria o tempo de queda necessário para o corpo percorrer uma distância de $(100 - 2.3\tau)$ cm, sendo τ o número do seu grupo? Para responder essa pergunta, implemente o método de Newton e resolva a equação $S_\Delta(t) - (100 - 2.3\tau) = 0$ com precisão $\epsilon = 10^{-10}$ (utilize uma linguagem de programação de sua preferência).

Observação: Note que, para calcular a função

$$f(t) \;=\; S_\Delta(t) - (100 - 2.3\tau)$$

e sua derivada em um certo valor de t é necessário, segundo a fórmula (1), descobrir a qual intervalo $[t_i, t_{i+1}]$ o valor t pertence. Isso pode ser feito com um algoritmo de busca binária (ver Sessão 3).

2.1 Tarefas

2.1.1 Tarefa 1

Monte e resolva o sistema de equações (2) por dois métodos

- Método da Eliminação de Gauss.
- Método de Jacobi com precisão $\epsilon = 10^{-8}$ e com o vetor nulo $y^{(0)}$ como chute inicial (ver algoritmo na Seção 3).

Nos dois casos,use x_0'' e x_n'' definidos por

$$x_0'' \;=\; \frac{\frac{x_2 - x_1}{t_2 - t_1} - \frac{x_1 - x_0}{t_1 - t_0}}{t_2 - t_0} \quad \text{e } x_n'' \;=\; \frac{\frac{x_n - x_{n-1}}{t_n - t_{n-1}} - \frac{x_{n-1} - x_{n-2}}{t_{n-1} - t_{n-2}}}{t_n - t_{n-2}}.$$

Observação: Para verificar se a implementação dos métodos para resolver o sistema (2) está correta, utilize o teste da Seção 5.

Responda as seguintes questões:

1. (1.0) Mostre que a matriz A do sistema (2) é estritamente diagonal dominante e calcule o número

$$\sigma := \max_i \sigma_i < 1, \quad \sigma_i := \frac{1}{|a_{i,i}|} \sum_{j \neq i} |a_{i,j}|, \quad i = 1, 2, \ldots, n. \tag{3}$$

2. (1.0) Sendo $y^{(1)}, y^{(2)}, \ldots, y^{(k)}, \ldots$ os vetores gerados pelo método de Jacobi e y^* a solução exata do sitema, utilize a estimativa

$$||y^{(k)} - y^*||_\infty \leq \frac{\sigma^k}{1-\sigma}||y^{(1)} - y^{(0)}||_\infty$$

para estimar (sem calcular $y^{(k)}$) o número mínimo necessário N de iterações para que se tenha

$$||y^{(N)} - y^*||_\infty \ \leq \ 10^{-8}.$$

3. (2.0) Calcule y^* utilizando o método da Eliminação de Gauss

4. (1.0) Resolva o sistema efetuando-se N iterações do método de Jacobi e verifique se o N encontrado satisfaz a inequação acima.

5. Exiba as soluções obtidas (por Gauss e por Jacobi) com **TODAS** as casas decimais disponíveis.

2.1.2 Tarefa 2

Para resolver uma equação da forma $f(u) = 0$, por um processo iterativo com sequência de aproximações $y_0, y_1, y_2, \ldots$, utilize o seguinte critério de parada: pare na iteração k se

$$f(y_k + \epsilon) * f(y_k - \epsilon) < 0.$$

1. (3.0) Escolha um intervalo de busca $[a, b]$ e um chute inicial y_0 que satisfaçam as hipóteses do Teorema da convexidade (ver Seção 7), de modo a garantir a convergência da sequência de iterações.

   Sugestão: para facilitar a análise, escolha a e b de modo que o intervalo $[a, b]$ esteja contido no intervalo $[t_j, t_{j+1}]$ que contém a raíz da equação.

2. (2.0) Resolva a equação $f(t) = 0$ pelo método de Newton; com precisão $\epsilon = 10^{-10}$ e utilizando o critério de parada descrito acima.

3. Para fins de conferência, escreva em uma tabela as 5 primeiras iterações **ESCREVA TODAS AS CASAS DECIMAIS DISPONÌVEIS**.

3 Algoritmos para resolver sistemas de equações lineares

Considere um sistema linear de m equações com m incógnitas dado na forma matricial por

$$A \times x = y, \tag{4}$$

sendo A a matriz (quadrada $m \times m$) do sistema e $y \in \mathbb{R}^m$ o vetor de termos independentes. A seguir são listados os algoritmos para resolver o sistema de equações (4). Nesses algoritmos, os índices da matriz e do vetor são representados por números de 1 até m. Para implementação em linguagens que utilizam outro tipo de indexação (a maioria das linguagens começa com o índice zero) será preciso fazer os ajustes necessários.

3.1 Método da eliminação de Gauss

Entrada: A, y, m = ordem da matriz A.
Saída: O vetor x, solução do sistema (4).

Parte 1: Escalonar (triangularizar o sistema)

Para j de 1 até m-1, faça

- Se $A_{j,j} = 0$, encontre k tal que $A_{k,j} \neq 0$ e troque as linhas j e k da Matriz A. Se isso não for possível, então a matriz A é singular e convém exibir uma mensagem de erro.
- Para i de j+1 até m, faça // elimina todos os elementos de A da // coluna j que estão abaixo da diagonal
  - Defina $\mu = -\frac{A_{i,j}}{A_{j,j}}$ // define o multiplo da linha j a ser somado à linha i da matriz A.
  - Para k de j até m, faça
    - $A_{i,k} = A_{ik} + \mu\, A_{j,k}$ // faz a operação desejada entre as linhas // da matriz A.
  - $y_i = y_i + \mu\, y_j$ // atualiza o vetor de termos independentes.

Parte 2: Resolver o sistema triangular resultante

- Para i de m até 1, faça
  - $x_i = y_i$.
  - Para j de $i+1$ até m faça
    - $x_i = x_i - A_{i,j}\ x_j$.
  - $x_i = x_i / A_{i,i}$.

4 Algoritmo de Jacobi (para o sistema (2), apenas)

Entrada: A, y, $n+1$ = ordem do sistema, $x^{(0)}$, tol, n_{max} = número máximo permitido de iterações.
Saída: O vetor x, solução do sistema (4).

Para $eps = tol + 1$, $k = 0$, $x^{(old)} = x^{(0)}$ e $x^{(new)} = x^{(0)}$

- Enquanto $eps > tol$ e $k < n_{max}$, faça
  - $x_0^{(new)} = \frac{1}{2}\left(d_0 - \lambda_0 x_1^{(old)}\right)$
  - Para i de 1 até $n-1$, faça
    - $x_i^{(new)} = \frac{1}{2}\left(d_i - \mu_i x_{i-1}^{(old)} - \lambda_i x_{i+1}^{(old)}\right)$
  - $x_n^{(new)} = \frac{1}{2}\left(d_n - \mu_n x_{n-1}^{(old)}\right)$
  - $eps = ||x^{(new)} - x^{(old)}||_\infty$
  - $x^{(old)} = x^{(new)}$

5 Algoritmo de busca binária

No nosso problema, dado um valor t, devemos encontrar a qual intervalo $]t_i; t_{i+1}]$ o ponto t pertence, isto é devemos encontrar o índice i tal que

$$ti < t \leq ti + 1. \tag{5}$$

Uma forma de fazer isso seria percorrer todos os índices i do vetor de tempos e verificar se a condição acima é satisfeita. Isso, porém, é muito custoso computacionalmente (imagine se tivéssemos um vetor muito grande). Uma forma mais rápida é utilizar um algoritmo de busca binária (semelhante ao método da bissecção que estudamos em aula). Dado um vetor ordenado $T = (t_0, t_1, \ldots t_n)$, com $t_0 < t_1 < \ldots < t_n$ e um valor de t (supondo $t_0 < t \leq t_n$), o seguinte algoritmo retorna o índice i que satisfaz (5).

- Faça $m = 0; M = n$;
- Enquanto $|M - m| > 1$, faça
  - $k =$ valor arredondado (inteiro) de $(M + m)/2$;
  - se $t > t_k$, faça $m = k$ e $M = M$;
  - se $t \leq tk$, faça $m = m$ e $M = k$;
- retorne $i = m$.

6 Teste para conferir implementação dos métodos para resolver o sistema (2)

Teste a sua implementaçao com o conjunto de dados

i =	0	1	2	3	4	5	6	7	8	9	10
t_i	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
x_i	2.00	1.81	1.64	1.49	1.36	1.25	1.16	1.09	1.04	1.01	1.00

Para esse conjunto de dados, com $n = 10$ e $x_0'' = x_n'' = 2$, os coeficientes $M_i; i = 0, 1, \ldots, n$ e A_i e B_i, $i = 0, 1, \ldots n - 1$, obtidos deverão ser, aproximadamente,

```
$M
2.00000000000000 2.00000000000001 1.99999999999998 1.99999999999999
2.00000000000005 1.99999999999998 1.99999999999998 2.00000000000005
1.99999999999995 2.00000000000002 2.00000000000000
$A
-1.8999999999999997 -1.6999999999999988 -1.5000000000000011
-1.3000000000000023 -1.0999999999999979 -0.9000000000000001
-0.7000000000000020 -0.4999999999999967 -0.3000000000000014
-0.0999999999999999
$B
1.99666666666667 1.80666666666667 1.63666666666667 1.48666666666667
1.35666666666667 1.24666666666667 1.15666666666667 1.08666666666667
1.03666666666667 1.00666666666667
```

7 Teorema da convexidade

Teorema 1 (Teorema da convexidade)**.** *Seja* $f : [a, b] \longrightarrow \mathbb{R}$ *uma função duas vezes diferenciável tal que*

- $f(a)f(b) < 0$.
- $f'(x) \neq 0 \forall x \in [a, b]$ *e* $f''(x)$ *não troca de sinal em* $[a, b]$.

Então existe uma única raíz α *de* f *em* $[a, b]$ *e, se* $x_0 \in [a, b]$ *é tal que* $x_1 \in [a, b]$*, então o método de Newton para* f *converge para* α.