MAP0151 - Cálculo Numérico e Aplicações
Lista 4 (Correção)

Mais uma vez, 5 questões implicam que cada uma vale 2.0 pontos.

(Questão 1) Acredito que esta era essencialmente uma questão para fazer um monte de contas e, assim sendo, eu esperava encontrar todas as etapas de eliminação e de pivotamento explicitadas na resolução do exercício.

Como eu não gosto de fazer contas e tenho meu *SciLab*, resolvi implementar algumas rotinas para fazer as contas por mim. Além disso, resolvi o problema aplicando uma estratégia de *pivotamento parcial* (isto é, escolha de pivôs em colunas), em vez de uma estratégia de *pivotamento total* (escolha de pivôs na matriz inteira). Aceitei na correção quem resolveu o problema usando qualquer uma das duas estratégias.

Comecei implementando subrotinas que eu sei que são necessárias ao Método da Eliminação de Gauss (MEG) com pivotamento parcial. Inicialmente implementei uma função que troca duas linhas de uma matriz.

```
// troca_linhas.sci

// devolve a matriz A com a linha i
// trocada pela linha j

function X = troca_linhas (A, i, j)
  l = A(i,:);
  A(i,:) = A(j,:);
  A(j,:) = l;
  X = A;
endfunction
```

Também implementei uma função que, dada uma matriz A e uma etapa de eliminação do MEG j determina qual o melhor pivô para aquela

etapa (toma o maior pivô em módulo na coluna j abaixo da diagonal principal inclusive)

```
// maior_pivo.sci

// dada uma coluna j da matriz A
// devolve o indice i da linha
// com maior pivo

function i = maior_pivo (A, j)
  [nlin, ncol] = size(A);

  // observe que comeco a procurar pivos
  // abaixo da diagonal principal
  i = j;
  for k = j:nlin do
    if abs(A(i,j)) < abs(A(k,j))
      i = k;
    end
  end
endfunction
```

bem como o *Método da Substituição Retroativa*, que resolve um sistema triangular superior.

```
// subst_retro.sci

// algoritmos de substituicao retroativa:
// dada uma matriz triangular superior A e
// um vetor b, resolve o sistema linear
// Ax = b.

function x = subst_retro (A, b)

  // estou supondo que a matriz A
  // eh quadrada n x n e que b eh
  // um vetor coluna n x 1.

  [nlin, ncol] = size(A)
  n = nlin;
```

```
  // inicializando o vetor-resposta
  x = zeros(n, 1);

  // aqui comeca efetivamente a
  // substituicao retroativa
  x(n) = b(n) / A(n,n);
  for j = (n - 1):-1:1 do
    s = 0;
    for k = (j + 1):n do
      s = s + A(j,k) * x(k);
    end
    x(j) = (b(j) - s) / A(j,j);
  end
endfunction
```

Finalmente, implementei o MEG com pivotamento parcial, mas com algumas sutilezas. Coloquei na saída do método um vetor p com as etapas de pivotamento (isto é, as permutações das linhas) anotadas, de forma que a partir de p eu possa reconstruir a matriz de permutações P e também a P^{-1}. Coloquei também na saída do método a matriz A escalonada, *mas colocando os multiplicadores de cada etapa no lugar dos zeros da matriz escalonada*, de modo a obter facilmente a decomposição LU de PA. Minha implementação do MEG ficou assim:

```
// gausspv.sci

// metodo de gauss com pivotamento

function [p, B, x] = gausspv (A, b)

  // importando os metodos necessarios
  getf ("troca_linhas.sci");
  getf ("maior_pivo.sci");
  getf ("subst_retro.sci");

  [nlin, ncol] = size(A);
  n = nlin;

  // inicializacao do vetor em que
  // anotarei as pivotacoes
  p = zeros(n);
```

```
  // mprintf ("%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", A, b);
  // mprintf ("\n");

  for i = 1:n do
    // fazendo a pivotacao e anotando o pivo
    m = maior_pivo (A,i);
    A = troca_linhas (A, i, m);
    b = troca_linhas (b, i, m);

    // mprintf ("%.2f\t%.2f\t%.2f\t%.2f\t %.2f\n", A, b);
    // mprintf ("\n");

    p(i) = m;

    for j = (i + 1):n do
      // calculando o multiplicador
      l = A(j,i) / A(i,i);

      // eliminando a j-esima linha de A,
      // tomando o cuidado de nao baguncar
      // os multiplicadores
      A(j,i:n) = A(j,i:n) - l * A(i,i:n);
      b(j) = b(j) - l * b(i);
      A(j,i) = l;

      // mprintf ("%.2f\t%.2f\t%.2f\t%.2f\t%.2f\n", A, b);
      // mprintf ("\n");

    end
  end

 // NESTE PONTO A MATRIZ 'A' JA ESTA ESCALONADA,
  // E 'b' EH O VETOR LADO DIREITO MODIFICADO.

  // resolvendo o sistema escalonado por
  // substituicao retroativa
  x = subst_retro (A, b);

  B = A;
endfunction
```

onde, para ver o retrato de cada etapa de eleminação / permutação, basta descomentar os `mprintf`s no código (SOMENTE PARA MATRIZES 4×4 COMO A DO PROBLEMA!).

Quando pus tudo para 'rodar' (os passos que fiz e várias curiosidades podem ser encontrados no *script* `questao1.sce`), a coisa ficou assim:

$$\left[\begin{array}{cccc|c} 6.00 & -2.00 & 2.00 & 4.00 & 12.00 \\ 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 3.00 & -13.00 & 9.00 & 3.00 & 27.00 \\ -6.00 & 4.00 & 1.00 & -18.00 & -38.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 6.00 & -2.00 & 2.00 & 4.00 & 12.00 \\ 3.00 & -13.00 & 9.00 & 3.00 & 27.00 \\ -6.00 & 4.00 & 1.00 & -18.00 & -38.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.50 & 2.00 & -1.00 & -1.00 & -5.00 \\ 3.00 & -13.00 & 9.00 & 3.00 & 27.00 \\ -6.00 & 4.00 & 1.00 & -18.00 & -38.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.50 & 2.00 & -1.00 & -1.00 & -5.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ -6.00 & 4.00 & 1.00 & -18.00 & -38.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.50 & 2.00 & -1.00 & -1.00 & -5.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ -0.50 & 0.00 & 4.00 & -13.00 & -21.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ 0.50 & 2.00 & -1.00 & -1.00 & -5.00 \\ -0.50 & 0.00 & 4.00 & -13.00 & -21.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ 0.50 & -0.18 & 0.36 & -0.91 & -1.64 \\ -0.50 & 0.00 & 4.00 & -13.00 & -21.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ 0.50 & -0.18 & 0.36 & -0.91 & -1.64 \\ -0.50 & -0.00 & 4.00 & -13.00 & -21.00 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ -0.50 & -0.00 & 4.00 & -13.00 & -21.00 \\ 0.50 & -0.18 & 0.36 & -0.91 & -1.64 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ -0.50 & -0.00 & 4.00 & -13.00 & -21.00 \\ 0.50 & -0.18 & 0.09 & 0.27 & 0.27 \end{array}\right]$$

$$\left[\begin{array}{cccc|c} 12.00 & -8.00 & 6.00 & 10.00 & 34.00 \\ 0.25 & -11.00 & 7.50 & 0.50 & 18.50 \\ -0.50 & -0.00 & 4.00 & -13.00 & -21.00 \\ 0.50 & -0.18 & 0.09 & 0.27 & 0.27 \end{array}\right]$$

e, por fim, aplicando Substituição Retroativa, obtemos

$$x = \begin{bmatrix} 1 \\ -3 \\ -2 \\ 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 12.00 & -8.00 & 6.00 & 10.00 \\ 0.25 & -11.00 & 7.50 & 0.50 \\ -0.50 & -0.00 & 4.00 & -13.00 \\ 0.50 & -0.18 & 0.09 & 0.27 \end{bmatrix}$$

$$p = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 4 \end{bmatrix}$$

Você pode ver no arquivo `questao1.sce` como obter as matrizes P, L e U a partir de p e B.

(Questão 2) Novamente, acho mais fácil escrever um programa que faz o serviço:

```
// inverte.sci

function I = inverte (A)
```

```
  getf("maior_pivo.sci");
  getf("troca_linhas.sci");

  [nlin, ncol] = size(A);
  n = nlin;

  I = eye (n, n);

  // escalonando de cima para baixo
  for i = 1:n do
    m = maior_pivo (A, i);
    A = troca_linhas (A, i, m);
    I = troca_linhas (I, i, m);
    for j = (i + 1):n do
      l = A(j,i) / A(i, i);
      A(j,:) = A(j,:) - l * A(i,:);
      I(j,:) = I(j,:) - l * I(i,:);
    end
  end

  // escalonando de baixo para cima
    for i = n:-1:1 do
    for j = (i - 1):-1:1 do
      l = A(j,i) / A(i, i);
      A(j,:) = A(j,:) - l * A(i,:);
      I(j,:) = I(j,:) - l * I(i,:);
    end
  end

  // limpando a diagonal
  for i = 1: n do
    I(i,:) = I(i,:) / A(i,i);
    A(i,i) = 1;
  end
endfunction
```

Aplicando o método acima à matriz

$$A_1 := \begin{bmatrix} 3 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 3 \end{bmatrix}$$

obtemos

$$A_1^{-1} = \begin{bmatrix} 0.4 & -0.1 & -0.1 \\ -0.1 & 0.4 & -0.1 \\ -0.1 & -0.1 & 0.4 \end{bmatrix}$$

(as etapas do escalonamento podem ser encontradas no arquivo `etapas2.txt`, à parte).

Para a matriz

$$A_2 := \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

obtemos

$$A_2^{-1} = \begin{bmatrix} -0.25 & 0.5 & 0.25 \\ 0.5 & -1 & 0.5 \\ 0.25 & 0.5 & -0.25 \end{bmatrix}$$

(tente fazer com a função `inverte.sci`).

(Questão 3) Esta questão eu achei um pouco mais difícil que as outras. O que farei a seguir será apenas dar uma idéia da demonstração do fato sugerido no enunciado do exercício (os detalhes – as contas e as justificativas das passagens – ficam para os corajosos, sugiro no mínimo tentar). Não esperei que nenhum de vocês seguisse o caminho que eu farei.

O que vou fazer é demonstrar o referido fato por indução na ordem n da matriz. Um passo importante seria escrever formamelmente (matematicamente) as matriz A, L e U do enunciado, esta parte fica como exercício para vocês. Os casos $n = 1$ e $n = 2$ são triviais (verifique!). Vamos verificar que o fato é válido para o caso $n = 3$, pois é mais interessante e ilustrativo que os casos $n = 1, 2$ (mas estes devem ser verificados também!).

Nesse caso temos que

$$A_3 = \begin{bmatrix} b_1 & c_1 & 0 \\ a_2 & b_2 & c_2 \\ 0 & a_3 & b_3 \end{bmatrix}$$

e suponha que as constantes $b_1, b_2, b_3, c_1, c_2, a_2, a_3$ são tais que A_3 seja inversível, isto é, possa ser escalonada (esta hipótese é mais importante do que parece a primeira vista, principalmente nos casos $n = 1, 2, 3$: por quê?).

Aplicando o Método da Eliminação de Gauss à matriz A_3 obtemos o seguinte:

$$\begin{bmatrix} b_1 & c_1 & 0 \\ a_2 & b_2 & c_2 \\ 0 & a_3 & b_3 \end{bmatrix} \to \begin{bmatrix} b_1 & c_1 & 0 \\ \frac{\mathbf{a_2}}{\mathbf{b_1}} & \left(b_2 - \frac{a_2}{b_1}c_1\right) & c_2 \\ \mathbf{0} & a_3 & b_3 \end{bmatrix}$$

$$\to \begin{bmatrix} b_1 & c_1 & 0 \\ \frac{\mathbf{a_2}}{\mathbf{b_1}} & \left(b_2 - \frac{a_2}{b_1}c_1\right) & c_2 \\ \mathbf{0} & \left(\frac{\mathbf{a_3}}{\mathbf{b_2} - \frac{\mathbf{a_2}}{\mathbf{b_1}}\mathbf{c_2}}\right) & \left(b_3 - \frac{a_3}{b_2 - \frac{a_2}{b_1}c_2}c_2\right) \end{bmatrix}$$

Agora, denotando

$$\begin{aligned} \beta_1 &:= b_1 \\ \alpha_2 &:= \frac{a_2}{\beta_1} = \frac{a_2}{b_1} \\ \beta_2 &:= b_2 - \alpha_2 c_1 = b_2 - \frac{a_2}{b_1}c_1 \\ \alpha_3 &:= \frac{a_3}{\beta_2} = \frac{a_3}{b_2 - \frac{a_2}{b_1}c_1} \\ \beta_3 &:= b_3 - \alpha_3 c_2 = b_3 - \frac{a_3}{b_2 - \frac{a_2}{b_1}c_1}c_2 \end{aligned}$$

temos, na matriz escalonada dor sistema anterior

$$A_3 \to \begin{bmatrix} b_1 & c_1 & 0 \\ \alpha_{\mathbf{2}} & \beta_2 & c_2 \\ \mathbf{0} & \alpha_{\mathbf{3}} & \beta_3 \end{bmatrix}$$

de modo que a decomposição LU da matriz A_3 fica assim:

$$L_3 := \begin{bmatrix} 1 & 0 & 0 \\ \alpha_2 & 1 & 0 \\ 0 & \alpha_3 & 1 \end{bmatrix}$$

$$U_3 := \begin{bmatrix} b_1 & c_1 & 0 \\ 0 & \beta_2 & c_2 \\ 0 & 0 & \beta_3 \end{bmatrix}$$

ou seja, para o caso $n = 3$ a proposição está verificada.

Suponha agora que para todo $n = k-1 \geq 1$ a proposição está verificada e suponha definidos indutivamente

$$\begin{aligned}\beta_1 &:= b_1 \\ \beta_j &:= b_j - \frac{a_j}{\beta_{j-1}} c_{j-1},\ 2 \leq j \leq k\end{aligned}$$

e

$$\alpha_j := \frac{a_j}{\beta_{j-1}},\ 2 \leq j \leq k.$$

de modo que $\beta_j = b_j - \alpha_j c_{j-1}$ para $2 \leq j \leq k$. *Mostre, como exercício, que cada um dos β_j e dos α_j está bem definido (DICA: basta mostrar que $\beta_j \neq 0$ para todo $j \in \{2, \ldots, k\}$.*

Vamos então verificar, com base nisso, a a proposição para o caso $n = k$. Note que, do enunciado, temos o seguinte: defina para cada $n \in 2, \ldots, k$

$$L_n := \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \alpha_2 & 1 & 0 & & 0 \\ 0 & \alpha_3 & 1 & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ 0 & & \cdots & \alpha_n & 1 \end{bmatrix}$$

$$U_n := \begin{bmatrix} \beta_1 & c_1 & 0 & \cdots & 0 \\ 0 & \beta_2 & c_2 & & 0 \\ \vdots & & \ddots & \ddots & \vdots \\ & & & \beta_{n-1} & c_{n-1} \\ 0 & \cdots & & 0 & \beta_n \end{bmatrix}$$

de modo que

$$L_k = \left[\begin{array}{cccccc|c} & & & & & & 0 \\ & & & & & & 0 \\ & & & L_{k-1} & & & \vdots \\ & & & & & & 0 \\ & & & & & & 0 \\ \hline 0 & 0 & \cdots & 0 & \alpha_k & & 1 \end{array}\right]$$

$$U_k = \left[\begin{array}{cccccc|c} & & & & & & 0 \\ & & & & & & 0 \\ & & & U_{k-1} & & & \vdots \\ & & & & & & 0 \\ & & & & & & c_{k-1} \\ \hline 0 & 0 & \cdots & 0 & 0 & & \beta_k \end{array}\right]$$

de modo que, denotando,

$$\mathbf{0} := \left[\begin{array}{c} 0 \\ \cdots \\ 0 \end{array}\right]_{(k-1)}$$

$$\mathbf{0}' = \left[\begin{array}{ccc} 0 & \cdots & 0 \end{array}\right]_{(k-1)}$$

$$\mathbf{v} := \left[\begin{array}{c} 0 \\ 0 \\ \cdots \\ 0 \\ c_{k-1} \end{array}\right]_{(k-1)}$$

$$\mathbf{w}' := \left[\begin{array}{ccccc} 0 & 0 & \cdots & 0 & \alpha_k \end{array}\right]_{(k-1)}$$

temos que, usando a hipótese de indução (verifique as contas!)

$$
\begin{aligned}
L_k * U_k &= \left[\begin{array}{c|c} L_{k-1} & \mathbf{0} \\ \hline \mathbf{w}' & 1 \end{array}\right] * \left[\begin{array}{c|c} U_{k-1} & \mathbf{v} \\ \hline \mathbf{0}' & \beta_k \end{array}\right] \\
&= \left[\begin{array}{c|c} L_{k-1} * U_{k-1} + \mathbf{0} * \mathbf{0}' & L_{k-1} * \mathbf{v} + \mathbf{0} * \beta_{\mathbf{k}} \\ \hline \mathbf{w}' * \mathbf{U_{k-1}} + \mathbf{1} * \mathbf{0}' & \mathbf{w}' * \mathbf{v} + \beta_{\mathbf{k}} \end{array}\right] \\
&= \left[\begin{array}{ccccc|c} & & & & & 0 \\ & & & & & 0 \\ & & A_{k-1} & & & \vdots \\ & & & & & 0 \\ & & & & & c_{k-1} \\ \hline 0 & 0 & \cdots & 0 & \alpha_k\beta_{k-1} & \alpha_k c_{k-1} + \beta_k \end{array}\right] \\
&= \left[\begin{array}{ccccc|c} & & & & & 0 \\ & & & & & 0 \\ & & A_{k-1} & & & \vdots \\ & & & & & 0 \\ & & & & & c_{k-1} \\ \hline 0 & 0 & \cdots & 0 & a_k & b_k \end{array}\right] \\
&= A_k
\end{aligned}
$$

ou seja

$$A_{k-1} = L_{k-1} * U_{k-1} \Rightarrow A_k = L_k * U_k$$

mas, como L_k é uma matriz triangular inferior com 1s na diagonal principal e U_k é uma matriz triangular superior e a *decomposição LU de uma matriz inversível é única*, então L_kU_k é a decomposição LU da matriz A_k.

Ou seja, mostramos por indução que o referido fato é válido para matrizes quadradas de qualquer ordem nas diretrizes do enunciado.

(Questão 4) Do jeito que fiz a questão anterior, esta saiu de graça: na indução já mostramos que β_j e α_j têm a forma pedida.

(Questão 5) Esse método, chamado de TDMA (*Tridiagonal matrix algorithm*), também conhecido como *algoritmo de Thomas*, está muito bem descrito no artigo da Wikipedia

`http://en.wikipedia.org/wiki/Tridiagonal_matrix_algorithm`,

o qual indico como leitura. Lá há implementações em C, C# e MatLab (este último muito parecido com o SciLab, e pode ser "traduzido" com pouquíssimo esforço).