Capítulo 16
Álgebra matricial

Considerações iniciais

- O **MATLAB** surgiu como uma interface amigável para facilitar o uso de subrotinas de álgebra linear numérica de alto desempenho (posteriormente outros recursos foram adicionados ao **MATLAB** que diversificaram em muito a sua aplicabilidade);

- Assim, o **MATLAB** fornece um vasto conjunto de funções para lidar com álgebra matricial;

- **Ressaltamos que, apesar do MATLAB aceitar vetores de n dimensões, a álgebra matricial é restrita a vetores de até 2 dimensões.**

Álgebra linear

- Um dos problemas centrais da álgebra linear é a resolução de sistemas de equações lineares simultâneas. Isto é, a resolução de problemas que podem ser escritos na forma

$$Ax = y,$$

onde A é uma matriz de coeficientes, x um vetor coluna de incógnitas e y um vetor coluna de valores.

- O estudo detalhado das condições em que um problema como o acima possui soluções e como encontrar soluções, sejam estas exatas ou aproximadas, para este problema foge ao objetivo deste curso. Para um estudo abrangente deste tópico sugerimos

  Álgebra Linear, de Boldrini et. al, Harper and Row,

  entre outras boas referências sobre o tópico.

Resolvendo $Ax = y$ no MATLAB

Vejamos algumas formas de resolver $Ax = y$ no **MATLAB**.

- Especificamos a matriz A e o vetor y:

```
>> A = (reshape(1:9,3,3))';
>> A(3,3) = 0
A =
     1     2     3
     4     5     6
     7     8     0
```

```
>> y = [366;804;351]
y =
   366
   804
   351
```

Análise inicial de $Ax = y$

- Lembre-se que, para uma matriz $A_{m\times n}$, o sistema de equações $Ax = y$ possui solução única se, e somente se, posto$(A) =$ posto$([A\ y]) = n$, onde o posto é o maior número de linhas (ou colunas) de A linearmente independentes.
- No caso em questão, obtemos

```
>> rank(A)
ans =
    3

>> rank([A y])
ans =
    3
```

Análise inicial de $Ax = y$

- Outra propriedade do sistema que deve ser verificada é o seu *número de condição*, $c := \| A \| \; \| A^{-1} \|$, que é uma medida da confiabilidade do erro de uma solução aproximada. Queremos que o número de condição seja pequeno.

```
>> cond(A)
ans =
  35.1059
```

Resolvendo $Ax = y$

- De forma direta: $x = A^{-1}y$

```
>> x = inv(A)*y
x =
  25.0000
  22.0000
  99.0000
```

- $inv(A)$: calcula a inversa de A;
- "*": operador de multiplicação de matrizes.

- Usando o operador de divisão à esquerda (mais recomendável)

```
>> x = A\ y
x =
  25.0000
  22.0000
  99.0000
```

- Neste caso é feita a fatoração LU da matriz A;
- Esta solução requer menos operações de ponto flutuante, sendo portanto mais rápida, e, em geral, mais precisa para problemas grandes.

Resolvendo $Ax = y$

- Quando o **MATLAB** não conseguir encontrar uma solução, ou não puder fazê-lo com a precisão desejada, uma mensagem de erro ou uma 'warning' será exibida na tela.

```
>> A(3,3) = 9;  % faz L3 = 2*L2-L1
>> disp(rank(A)), disp(cond(A))
     2
   9.2151e+16

>> x = inv(A)*y
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.203039e-18.
x =
   1.0e+18 *
   -2.8086
    5.6172
   -2.8086
```

Quando não há solução para $Ax = y$

- Isto ocorre quando $rank(A) \neq rank([A\ y])$, ou seja, o vetor coluna y não pertence ao espaço de soluções;
- Para este caso o **MATLAB** também encontra soluções úteis na prática, como a solução de quadrados mínimos.
  - É obtida usando os operadores de divisão ('$\backslash$' e '$/$');
  - Esta solução minimiza o quadrado da norma do resíduo $e = Ax - y$.
- Além de calcular a solução de quadrados mínimos com os operadores de divisão, o **MATLAB** possui algumas variantes:
  - $lscov$: resolve o problema dos quadrados mínimos quando a matriz de covariância de dados é conhecida;
  - $lsqnonneg$: encontra a solução de quadrados mínimos não negativos onde todos os componentes da soluçao são forçados a serem positivos;

Exemplo

```
>> A = [1 2 3;4 5 6;7 8 0;2 5 8]
A =
     1     2     3
     4     5     6
     7     8     0
     2     5     8

>> y =[366 804 351 514]'
y =
   366
   804
   351
   514
```

```
>> x = A\y
x =
  247.9818
 -173.1091
  114.9273

>> e = A*x-y
e =
 -119.4545
   11.9455
         0
   35.8364

>> disp(norm(e))
  125.2850
```

Sistema indeterminado

- Ocorre quando temos infinitas soluções para o sistema $Ax = y$. Isto significa que

$$rank(A) = rank([A\ y]) = k < n.$$

  Ou seja, há $n - k$ graus de liberdade.

- Dentre as infinitas soluções duas podem ser facilmente obtidas pelo **MATLAB**:
  - aquela que possui o maior número de elementos nulos. Esta é obtida pelo uso do operador de divisão; e
  - a solução de norma mínima (solução onde a norma do vetor x é mínima), cujo algoritmo é baseado no cálculo da *pseudo-inversa*. Podemos obter essa solução com o comando $x = pinv(A) * y$.

Exemplo

```
>> A = ([1 2 3; 4 5 6; 7 8 0; 2 5 8])'
A =
     1     4     7     2
     2     5     8     5
     3     6     0     8

>> y = [ 366; 804; 351]
y =
   366
   804
   351

>> disp(rank(A))
     3

>> disp(rank([A y]))
     3
```

```
>> x = A\y
x =
             0
 -165.9000
   99.0000
  168.3000

>> xn = pinv(A)*y
xn =
   30.8182
 -168.9818
   99.0000
  159.0545

>> disp(norm(x))
  256.2200

>> disp(norm(xn))
  254.1731
```

Considerações finais sobre $Ax = y$

- Podemos trabalhar com a transposta do sistema $Ax = y$, isto é, $x'A' = y'$. Se for este o caso, os vetores x e y são vetores linha. Neste caso devemos usar a divisão à direita $x' = y'/A$.

- Quando encontra uma divisão, seja ela à esquerda ($\backslash$), ou à direita ($/$) o **MATLAB** decide qual algoritmo interno usar analisando a matriz de coeficientes.
  - Se a matriz for triangular, ou permutação de uma triangular, o **MATLAB** não fatora a matriz, e faz as substituições diretamente.

Recursos do MATLAB

- Além de resolver sistemas de equações lineares o **MATLAB** possui um grande número de funções matriciais que são úteis para resolver problemas numéricos de álgebra linear. Na seção 16.2 do livro há uma tabela com uma descrição sucinta de várias destas funções.
  Consulte também Help –> **MATLAB** Help –> **MATLAB** –> Mathematics –> Matrices and Linear Algebra –> Function Summary
- O **MATLAB** possui várias matrizes especiais predefinidas, isto é, funções que geram tais matrizes como a matriz identidade, matriz de 1s e a matriz de 0s que nós já vimos. Na seção 16.3 do livro há uma tabela com diversas destas matrizes.
  Algumas dessas funções podem ser encontradas digitando “help gallery” ou visitando o diretório “toolbox/matlab/elmat” na raiz do diretório onde está instalado **MATLAB**.

Matrizes esparsas

- Matrizes esparsas são matrizes que possuem poucos elementos não nulos;
- Surgem em muitas aplicações práticas. Há matrizes que surgem na prática que possuem menos de 1% de elementos não nulos;
  - Para evitar o desperdício de espaço costuma-se armazenar apenas os elementos não nulos e os índices que identificam a linha e a coluna destes elementos;
  - Para evitar o desperdício de processamento foram desenvolvidos diversos algoritmos especiais que resolvem problemas típicos de matrizes esparsas.

Matrizes esparsas

- As técnicas que envolvem matrizes esparsas (teoria e prática) são complexas. Para o usuário do **MATLAB** esta complexidade é transparente;
- As matrizes esparsas são armazenadas em variáveis como as matrizes densas;
- A maioria dos cálculos com matrizes esparsas emprega as mesmas técnicas usadas para matrizes densas.
- A seção 16.5 do livro possui uma tabela com diversas funções para a manipulação de matrizes esparsas.
- Consulte também Help –> **MATLAB** Help –> **MATLAB** –> Mathematics –> Sparse Matrices –> Function Summary

Criação de matrizes esparsas

- As matrizes esparsas são criadas através da função $sparse$. Vejamos alguns exemplos antes de colocarmos a sintaxe geral da função:

```
>> S = sparse(1:4)                >> S1 = sparse(eye(4))
S =                               S1 =
   (1,1)        1                    (1,1)        1
   (1,2)        2                    (2,2)        1
   (1,3)        3                    (3,3)        1
   (1,4)        4                    (4,4)        1
>> S2 = sparse(1:3,1:3,pi*ones(1,3))
S2 =
   (1,1)       3.1416
   (2,2)       3.1416
   (3,3)       3.1416
>> whos
  Name      Size        Bytes  Class
  S         1x4            68  sparse array
  S1        4x4            68  sparse array
  S2        3x3            52  sparse array
Grand total is 11 elements using 188 bytes
```

$sparse$ - sintaxe geral

```
>> help sparse
 SPARSE Create sparse matrix.
   S = SPARSE(X) converts a sparse or full matrix to sparse form by
   squeezing out any zero elements.

   S = SPARSE(i,j,s,m,n,nzmax) uses the rows of [i,j,s] to generate an
   m-by-n sparse matrix with space allocated for nzmax nonzeros.  The
   two integer index vectors, i and j, and the real or complex entries
   vector, s, all have the same length, nnz, which is the number of
   nonzeros in the resulting sparse matrix S .  Any elements of s
   which have duplicate values of i and j are added together.

   There are several simplifi cations of this six argument call.
   S = SPARSE(i,j,s,m,n) uses nzmax = length(s).
   S = SPARSE(i,j,s) uses m = max(i) and n = max(j).
   S = SPARSE(m,n) abbreviates SPARSE([],[],[],m,n,0).  This
   generates the ultimate sparse matrix, an m-by-n all zero matrix.

   The argument s and one of the arguments i or j may be scalars,
   in which case they are expanded so that the fi rst three arguments
   all have the same length.
```

Conversão esparsa $\rightarrow$ densa

- A conversão de uma matriz esparsa para uma matriz densa se dá pelo comando $full$.

```
>> D = full(S)
D =
    1    2    3    4
>> D2 = full(S2)
D2 =
   3.1416        0        0
        0   3.1416        0
        0        0   3.1416
>> whos
  Name      Size         Bytes  Class
  D         1x4          32  double array
  D1        4x4          128  double array
  D2        3x3          72  double array
  S         1x4          68  sparse array
  S1        4x4          68  sparse array
  S2        3x3          52  sparse array
Grand total is 40 elements using 420 bytes
```

```
>> D1 = full(S1)
D1 =
    1    0    0    0
    0    1    0    0
    0    0    1    0
    0    0    0    1
```