Capítulo 5 - Vetores e matrizes

Vetores

Exemplo: *Suponha que se queira calcular o seno de* x *para* $x \in [0, \pi]$*, a cada* 0.1π*.*

Vetores

Exemplo: *Suponha que se queira calcular o seno de* x *para* $x \in [0, \pi]$*, a cada* 0.1π*.*

Sem usar vetores:

```
>> y0 = sin(0)
y0 =
     0
>> y1 = sin(0.1*pi)
y1 =
    0.3090
>> y2 = sin(0.2*pi)
y2 =
    0.5878
...
```

Vetores

Exemplo: *Suponha que se queira calcular o seno de* x *para* $x \in [0, \pi]$*, a cada* 0.1π*.*

Usando vetores

```
>> x = [0 0.1*pi 0.2*pi 0.3*pi 0.4*pi 0.5*pi ...
0.6*pi 0.7*pi 0.8*pi 0.9*pi pi]

 Columns 1 through 7
   0 0.3142 0.6283 0.9425 1.2566 1.5708 1.8850
 Columns 8 through 11
    2.1991 2.5133 2.8274 3.1416

>> y=sin(x)

y =
 Columns 1 through 7
   0 0.3090 0.5878 0.8090 0.9511 1.0000 0.9511
 Columns 8 through 11
   0.8090 0.5878 0.3090 0.0000
```

Construção de vetores

- Acabamos de ver a forma mais simples de criar um vetor: especificamos todos os elementos separando-os por espaços e delimitando-os por colchetes. Ex.
- Para o **MATLAB** , os colchetes são um operador de concatenação. Isto é, os elementos entre colchetes são concatenados, resultando numa matriz de dimensões apropriadas.

```
>> x = [ 1 2 3 ]
x =
     1     2     3
>> y = [ 4 5 6 ]
y =
     4     5     6
>> [ x y ]
ans =
     1     2     3     4     5     6
```

Construção de vetores

- A parte real e a parte imaginária de um número complexo também são separadas por **espaços.**
- Assim, um cuidado adicional deve ser tomado para construir vetores que contenham números complexos.

```
>> x = [1 -2i 3 4 5+6i]
x =
Columns 1 through 4
  1.0000   0-2.0000i   3.0000   4.0000
  Column 5
     5.0000 + 6.0000i

>> y = [(1-2i) 3 4 5+6i]
y =
  1.0000-2.0000i   3.0000  4.0000  5.0000+6.0000i
```

Construção de vetores

Exemplo: *Suponha agora que quiséssemos calcular o seno de x para 200 valores de x uniformemente distribuídos em $[0, \pi]$!*

Construção de vetores

Exemplo: *Suponha agora que quiséssemos calcular o seno de x para 200 valores de x uniformemente distribuídos em $[0, \pi]$!*

É necessário que haja uma forma mais automática de inserir valores em um vetor para que possamos cumprir este objetivo de forma eficiente

Construção de vetores

Exemplo: *Suponha agora que quiséssemos calcular o seno de x para 200 valores de x uniformemente distribuídos em $[0, \pi]$!*

```
x = linspace(0,pi,200)
x =
  Columns 1 through 7
     0 0.0158 0.0316 0.0474 0.0631 0.0789 0.0947
  Columns 8 through 14
0.1105 0.1263 0.1421 0.1579 0.1737 0.1894 0.2052
  Columns 15 through 21
0.2210 0.2368 0.2526 0.2684 0.2842 0.3000 0.3157
  ...
  Columns 190 through 196
2.9837 2.9995 3.0153 3.0311 3.0469 3.0627 3.0784
  Columns 197 through 200
3.0942 3.1100 3.1258 3.1416
```

Outras formas de construir vetores

- `x = logspace(<pot_ini>,<pot_fim>,<num_ele>)`
  Cria um vetor com `num_ele` elementos logaritmicamente espaçados, cujo primeiro elemento é $10^{\text{pot\_ini}}$ e o último elemento é $10^{\text{pot\_fim}}$.

- `x = <prim_ele>:<inc>:<ult_ele>`
  Cria um vetor cujo primeiro elemento é `prim_ele`, os seguintes são obtidos incrementando-se o anterior de `inc`, até o último elemento `y`, que é o maior número menor ou igual a `ult_ele`, tal que `y = prim_ele*inc*n`, para algum inteiro `n`.
  - Se `inc` for omitido, o valor `1` é utilizado em seu lugar.

Construção de vetores

- É possível construir vetores associando as várias técnicas que acabamos de descrever. Considere o exemplo a seguir:

```
>> a = 1:5;

>> b = 1:2:9;

>> c = [b a]
c =
     1   3   5   7   9   1   2   3   4   5

>> d =[a(1:2:5) 1 0 1]
d =
     1   3   5   1   0   1
```

Acesso a elementos de vetores

- `x(<ind>)`: exibe o `ind`-ésimo elemento do vetor `x`.
- `x(<n_i>:<n_f>)`: exibe do `n_i`-ésimo elemento até o `n_f`-ésimo elemento do vetor `x`.
- `x(<n_i>:end)`: exibe do `n_i`-ésimo elemento ao último elemento do vetor `x`.
- `x(<n_i>:<inc>:<n_f>)`: exibe os elementos cujos índices estão em `[n_i, n_f]`, começando pelo elemento de índice `n_i` e usando o incremento `inc`.
  - Note que o último elemento exibido pode não ser o elemento de índice `n_f`.

Acesso a elementos de vetores

- O operador de concatenação (colchetes) permite exibir os elementos de um vetor na ordem que desejarmos, bem como repetindo elementos. Considere o exemplo a seguir:

```
>> x = (10:10:100)
x =
    10   20   30   40   50   60   70   80   90  100
>> x([8 2 5 1 1 1])
ans =
    80   20   50   10   10   10
```

- `[8 2 5 1 1 1]` é um vetor que é usado para indexar os elementos de x que desejamos exibir.
- Note que podemos repetir índices.
- É preciso que os índices sejam válidos.

Acesso a elementos de vetores

- Se o usuário tentar exibir elementos cujos índices não são inteiros, o **MATLAB** arredonda os índices.
- Se os valores dos índices arredondados forem válidos, o **MATLAB** exibe os elementos correspondentes e emite um *warning*; se não forem, o **MATLAB** retorna uma mensagem de erro.

```
>> x = (10:10:100)
x =
    10   20   30   40   50   60   70   80   90 100
>> x(3.2)
Warning: Subscript indices must be integer values.
ans =
    30
>> x(10.9)
Warning: Subscript indices must be integer values.
???  Index exceeds matrix dimensions.
```

Vetor coluna

- Como criar um vetor coluna:
  - Como se cria vetor linha, mas substituir o “espaço” (ou “`,`”) por “`Enter`” ou “`;`”.

```
>> x = [1;2;3]
x =
     1
     2
     3
>> x = [1
2
3]
x =
     1
     2
     3
```

Vetor coluna

- Transpôr um vetor linha.
  - Operador “’”: para vetores reais é a transposta, para vetores complexos, a transposta conjugada.
  - Operador “.’”: para vetores reais e complexos é a transposta.

```
d =
  Columns 1 through 4
   1.0 + 1.0i 2.0 + 2.0i 3.0 + 3.0i 4.0 + 4.0i
  Column 5
   5.0 + 5.0i
>> e = d'                          >> f = d.'
e =                               f =
  1.0000 - 1.0000i                  1.0000 + 1.0000i
  2.0000 - 2.0000i                  2.0000 + 2.0000i
  3.0000 - 3.0000i                  3.0000 + 3.0000i
  4.0000 - 4.0000i                  4.0000 + 4.0000i
  5.0000 - 5.0000i                  5.0000 + 5.0000i
```

Criação de matrizes

- Extensão das idéias de vetores linha e vetores colunas: elementos nas linhas separados por *brancos* ou “,” e elementos em diferentes colunas por “Enter” ou “;”.

```
>> a = [ 1 2 3; 4,5,6;7 8,9]
a =
     1     2     3
     4     5     6
     7     8     9

>> a =[1 2 3
4,5 6
7,8,9]
a =
     1     2     3
     4     5     6
     7     8     9
```

Criação de matrizes

Já vimos que podemos criar matrizes assim:

```
>> A = [a b; c d]
```

onde a, b, c e d são escalares.

- O **MATLAB** encara mesmo um escalar como uma matriz 1×1.
- Assim, é uma extensão natural criar matrizes desta forma, quando a, b, c e d não são escalares.
- **Mas** deve-se cuidar que as dimensões estejam corretas.

Exemplo

```
>> A = [1 2; 3 4];

>> B = [5 6 ; 7 8];

>> C =[ 8 9 ; 10 11];

>> D = [ 12 13 ; 14 15];

>> E =[ A B ; C D ]

E =
     1     2     5     6
     3     4     7     8
     8     9    12    13
    10    11    14    15
```

Criação de matrizes

- É possível mesclar técnicas diferentes de construção de vetores para construir uma matriz. Exemplos:

```
>> b=[1:5;2:2:10]
b =
     1     2     3     4     5
     2     4     6     8    10

>> b= [1:5; logspace(0,1,5); linspace(10,20,5)]
b =
    1.0000    2.0000    3.0000    4.0000    5.0000
    1.0000    1.7783    3.1623    5.6234   10.0000
   10.0000   12.5000   15.0000   17.5000   20.0000
```

- As dimensões devem ser compatíveis.

Criação de matrizes - mais exemplos

```
>> A = [ 1 2 3; 4 5 6;7 8 9];

>> B = A(3:-1:1, 1:3)
B =
     7     8     9
     4     5     6
     1     2     3
```

- A matriz B é construída a partir da matriz A. As linhas de B são determinadas pelo vetor que antecede a vírgula, e as colunas pelo que sucede a vírgula.

Criação de matrizes

```
>> A = [ 1 2 3; 4 5 6;7 8 9];

>> B = A(3:-1:1, 1:3)
B =
     7     8     9
     4     5     6
     1     2     3
```

- Os comandos a seguir fazem exatamente o mesmo que o exemplo anterior.
  - `B = A(end:-1:1, 1:3)`: a palavra **end** diz que é para usar dimensão de índice máximo. No caso, esta dimensão é linha e seu valor é 3.
  - `B = A(3:-1:1,:)`: neste caso o **“:”** indica que é para que todas as colunas sejam consideradas.

Outros exemplos

```
>> B = A(end:-1:1,:)
B =
     7     8     9
     4     5     6
     1     2     3

>> C = [A B(:,[1 3])]
C =
     1     2     3     7     9
     4     5     6     4     6
     7     8     9     1     3

>> B = A(3:-1:2, 1:2)
B =
     7     8
     4     5
>>  B=A([1 1],2:3)
B =
     2     3
     2     3
```

Operações elemento a elemento

Sejam $A = [a_1 \ldots a_n]$ um vetor e c um escalar.

Operação	Resultado
Adição	$A + c = [a_1 + c \ldots a_n + c]$
Subtração	$A - c = [a_1 - c \ldots a_n - c]$
Multiplicação	$A * c = [a_1 * c \ldots a_n * c]$
Divisão	$A/c = c \backslash A = [a_1/c \ldots a_n/c]$
Exponenciação	$A.\hat{}\, c = [a_1^c \ldots a_n^c]$
	$c.\hat{}\, A = [c^{a_1} \ldots c^{a_n}]$

Não há problemas de o vetor A ser bi-dimensional

Exemplo: vetor e escalar

```
>> c = 2; A = 1:2:10;
>> A+c
ans =
     3     5     7     9    11
>> A-c
ans =
    -1     1     3     5     7
>> A*c
ans =
     2     6    10    14    18
>> A/c % mesmo que c\A
ans =
    0.5000  1.5000  2.5000  3.5000  4.5000
>> A.^c
ans =
     1     9    25    49    81
>> c.^A
ans =
     2     8    32   128   512
```

Operações elemento a elemento

Sejam $A = [a_1 \ldots a_n]$ e $B = [b_1 \ldots b_n]$ vetores.

Operação	Resultado
Adição	$A + B = [a_1 + b_1 \ldots a_n + b_n]$
Subtração	$A - B = [a_1 - b_1 \ldots a_n - b_n]$
Multiplicação	$A.*B = [a_1 * b_1 \ldots a_n * b_n]$
Divisão esq.	$A./B = [a_1/b_1 \ldots a_n/b_n]$
Divisão dir.	$B.\backslash A = [a_1/b_1 \ldots a_n/b_n]$
Exponenciação	$A.\hat{}B = [a_1^{b_1} \ldots a_n^{b_n}]$

Os vetores A e B podem ser bi-dimensionais, desde que as dimensões sejam compatíveis.

Exemplo - vetor e vetor

```
>> A = 1:5; B=5:-1:1;
>> A+B
ans =
     6     6     6     6     6
>> A-B
ans =
    -4    -2     0     2     4
>> A.*B
ans =
     5     8     9     8     5
>> A./B
ans =
    0.2000    0.5000    1.0000    2.0000    5.0000
>> A.\B
ans =
    5.0000    2.0000    1.0000    0.5000    0.2000
>> A.^B
ans =
     1    16    27    16     5
```

Exemplo

- As operações elemento a elemento podem ser usadas em expressões matemáticas, desde que respeitadas as dimensões dos vetores envolvidos. A precedência dos operadores é a tradicional.

```
>> A = [1 2 3; 4 5 6];  B = [7 8 9; 10 11 12];

>> 2*A - 1
ans =
     1     3     5
     7     9    11
>> 2*A - B
ans =
    -5    -4    -3
    -2    -1     0
>> A.^(B-1)
ans =
           1         128        6561
      262144     9765625   362797056
```

Expansão escalar

- Considere o seguinte exemplo:

```
>> A = [1 2; 3 4];

>> 1./A
ans =
    1.0000    0.5000
    0.3333    0.2500
```

- O escalar 1 no numerador é expandido para um vetor de mesma dimensão de A e então é executada a operação de divisão elemento a elemento entre vetores.
- O processo de expandir escalares para vetores é denominado *expansão escalar* e é extensivamente usado no **MATLAB** .

Funções úteis para criar matrizes

Podem ter um ou dois parâmetros:

- com um parâmetro gera uma matriz quadrada;
- com dois parâmetros, `m,n`, gera uma matriz de ordem `m`$\times$`n`.

- `ones`: Matriz formada apenas de 1's.
- `zeros`: Matriz formada apenas de 0's.
- `eye`: Matriz de ordem `m`$\times$`n` com os elementos $(i, i) = 1$, para $i = 1 : \min\{$`m`, `n`$\}$. Demais elementos são zeros.
- `rand`: Matrizes com elementos aleatórios uniformemente distribuídos no intervalo $[0..1]$.

Funções úteis para gerar matrizes

- `diag(vetor <,pos>)`:

  Cria uma matriz, colocando os elementos do vetor `vetor` em uma diagonal paralela à diagonal principal.

  Esta paralela é determinada pelo parâmetro `pos`, que quando ausente é assumido como 0.

```
>> a = 1:2
a =
     1     2

>> diag(a)
ans =
     1     0
     0     2

>> diag(a,1)
ans =
     0     1     0
     0     0     2
     0     0     0

>> diag(a,-1)
ans =
     0     0     0
     1     0     0
     0     2     0
```

Funções `diag` - outro uso

- `diag(vetor <,pos>)`:

  Se o vetor já existe, esta função extrai a diagonal determinada pelos parâmetros da função.

  O parâmetro `pos` possui a mesma função do caso anterior.

```
>> a = [1,2,3;4,5,6;7,8,9];

>> diag(a)
ans =
     1
     5
     9

>> diag(a,1)
ans =
     2
     6

>> diag(a,-1)
ans =
     4
     8
```

Mais exemplos

```
>> A = [1,2,3;4,5,6]
A =
     1     2     3
     4     5     6

>> B = A(end:-1:1,:)
B =
     4     5     6
     1     2     3

>> C = [A B(:,[1 3])]
C =
     1     2     3     4     6
     4     5     6     1     3

>> B = A(1:2, 2:3)
B =
     2     3
     5     6
```

Exemplo de indexação por vetor

- Neste caso o vetor C é utilizado para indexar os elementos do vetor A que formarão B.

```
>> A = [ 1 2 3; 4 5 6;7 8 9]
A =
     1     2     3
     4     5     6
     7     8     9

>> C = [1 3]
C =
     1     3

>> B = A(C,C)
B =
     1     3
     7     9
```

Acesso a elementos específicos

```
>> A = [ 1 2 3; 4 5 6] % constrói uma matriz 2 x 3
A =
     1     2     3
     4     5     6

>> A(2,3) = 0 % modifica o valor do elemento A(2,3)
A =
     1     2     3
     4     5     0

>> A(3,5) = 1 % coloca o valor 1 no elemento A(3,5)
a =
     1     2     3     0     0
     4     5     0     0     0
     0     0     0     0     1
```

- Note que no último caso o tamanho de `A` é expandido e preenchido com zeros para que o elemento `A(3,5)` possa existir.

Inserindo linhas e colunas

- Para preencher uma coluna toda com certo escalar podemos proceder da seguinte forma:

```
>> A(:,4) = [3; 3; 3]
A =
     1     2     3     3     0
     4     5     0     3     0
     0     0     0     3     1
```

- Dadas as possibilidades de erros, há uma forma alternativa que envolve expansão escalar:

```
>> A(:,4) = 3
A =
     1     2     3     3     0
     4     5     0     3     0
     0     0     0     3     1
```

Inserindo linhas e colunas

- A idéia anterior se aplica para linhas também e pode estar associada à necessidade de se expandir a matriz.

```
>> A = [1 9; 2 5];

>> A(:,5) = 1
A =
     1     9     0     0     1
     2     5     0     0     1

>> A(4,:) = 3
A =
     1     9     0     0     1
     2     5     0     0     1
     0     0     0     0     0
     3     3     3     3     3
```

Exemplo

- Gerar uma matriz cujos elementos são iguais.

```
>> d = pi;
>> d*ones(2,3) % mais lento
ans =
    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
>> d+zeros(2,3) % lento
ans =
    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
>> d(ones(2,3)) % rápida
ans =
    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
>> repmat(d,2,3) % mais rápida
ans =
    3.1416    3.1416    3.1416
    3.1416    3.1416    3.1416
```

Exemplo

- Os dois primeiros enfoques são mais lentos porque envolvem operações elemento a elemento. O segundo é um pouco melhor que o primeiro porque adição é uma operação “mais barata” que multiplicação. São bons enfoques para matrizes que não são muito grandes.

- O s dois últimos enfoques são mais rápidos, apesar de menos intuitivos, e envolvem indexação de vetores.

  - **`d(ones(r,c))`:** cria um vetor $r \times c$ e usa este vetor para indexar e replicar o escalar `d`.

    A criação de um vetor temporário é o que torna este enfoque mais lento que o seguinte (apesar de não serem usadas operações de ponto flutuante).

Exemplo

- **`repmat(d,r,c):`** a função `repmat` tem o objetivo de replicar matrizes. Sendo d um escalar, esta função executará os seguintes passos:

```
D(r*c)= d;                % cria um vetor linha cujo
                          % elemento r*c é d.
D(:)= d;                  % executa uma expansão escalar
                          % para preencher todos os
                          % elementos de D com d.
D = reshape(D,r,c)        % redistribui os r*c elementos
                          % do vetor linha na  forma de
                          % uma matriz r x c.
```

Voltaremos às funções repmat e reshape adiante.

Função *reshape*

- Esta função redistribui os elementos de um vetor como especificado nos argumentos.

```
>> A = 1:6
A =
     1     2     3     4     5     6

>> reshape(A,2,3)
ans =
     1     3     5
     2     4     6

>> reshape(A,[2 3])
ans =
     1     3     5
     2     4     6

>> reshape(A,1,5)
??? Error using ==> reshape
To RESHAPE the number of elements must not change.
```

Função *repmat*

- Esta função é usada para replicar vetores como especificado nos argumentos.

```
>> A = reshape(1:4,2,2)
A =
     1     3
     2     4

>> repmat(A,1,3) % = repmat(A,[1 3]) = [A A A]
ans =
     1     3     1     3     1     3
     2     4     2     4     2     4

>> repmat(A,2,2) % = repmat(A,2) = [A A; A A]
ans =
     1     3     1     3
     2     4     2     4
     1     3     1     3
     2     4     2     4
```

Funções úteis

- $size(A)$: Retorna as dimensões do vetor.

```
>> A = reshape(1:8,2,4);
>> size(A)                  >> [r,c] = size(A)
ans =  2   4                r = 2
                            c = 4

>> size(A,1)                >> size(A,2)
ans = 2                     ans = 4
```

- $numel(A)$: Número de elementos do vetor A.
- $length(A)$: Retorna o tamanho da maior dimensão do vetor. Se for um vetor unidimensional o resultado é o mesmo retornado por $size(A)$.
- $ndims(A)$: Número de dimensões do vetor A. Esta função é equivalente a $length(size(A))$.

Funções de manipulação de vetores

- $fliup(A)$: "intercambia elementos de um vetor ao longo do eixo horizontal central".

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

```
>> flipud(A)
ans =
     3     6     9
     2     5     8
     1     4     7
```

- $fliplr(A)$: "intercambia elementos de um vetor ao longo do eixo vertical central".

```
>> fliplr(A)
ans =
     7     4     1
     8     5     2
     9     6     3
```

Funções de manipulação de vetores

- $rot90(A, k)$: Rotaciona o vetor $k * 90$ graus no sentido anti-horário. Se k estiver ausente, assume $k = 1$.

```
>> A= reshape(1:6,2,3)
A =
     1     3     5
     2     4     6
```

```
>> rot90(A,2)
ans =
     6     4     2
     5     3     1
```

- $triu(A)$: Produz uma matriz cuja parte triangular superior (incluindo a diagonal principal) é a mesma de A e o restante dos elementos são zeros.
- $tril(A)$: Análoga à anterior, mas triangular inferior.

```
>> tril(A)
ans =
     1     0     0
     2     4     0
```

```
>> triu(A)
ans =
     1     3     5
     0     4     6
```