Capítulo 8 - Strings

Strings

- O **MATLAB** é usado extensivamente para cálculos. Sendo assim, é bem mais comum o seu uso com números. Entretanto, nas situações em que há necessidade de se lidar com textos, **MATLAB** também oferece ferramentas que permitem este tipo de manipulação.
- Strings são cadeias de caracteres, um outro tipo de dado do **MATLAB** .
- Internamente são vetores de caracteres que possuem representação na tabela ASCCII. Cada caracter usa 2 bytes (diferentemente dos 8 bytes usados para representar o tipo *double*), espaço suficiente para armazenamento do código ASCII do caracter.
- O **MATLAB** possui diversas funções para manipulação de strings. Descreveremos algumas delas, as outras podem ser encontradas na bibliografia e no *help on line*.

Construção de strings

- Para criar uma string delimitamos a seqüência de caracteres por apóstrofos:

```
>> S = 'Exemplo de string'
S =
Exemplo de string

>> size(S)
ans =
     1    17
```

- Apóstrofos dentro de uma string são representados por dois apóstrofos consecutivos:

```
>> S1 = 'Caixa d''agua'
S1 =
Caixa d'agua
```

Caracteres $\longrightarrow$ código ASCII

- Para visualizar os códigos ASCII de uma string S basta executar alguma operação aritmética (abs, $real$, etc) tendo S como argumento.

```
>> S = 'Exemplo de string';
>> S_num = double(S)
S_num =
  Columns 1 through 7
    69   120   101   109   112   108   111

  Columns 8 through 14
    32   100   101    32   115   116   114

  Columns 15 through 17
   105   110   103

>> char(S_num)
ans =
Exemplo de string
```

Caracteres ← código ASCII

A função *char* faz essa conversão:

- Valores numéricos inferiores a 0 geram um *warning* e a conversão é feita para o valor 0 (char=null);

```
>> char(-10)
Warning: Out of range or non-integer...


ans =
```

- Valores numéricos superiores a 255 são antes convertidos para o resto da sua divisão por 256:

```
>> char(double('a'))
ans = a
>> char(double('a'+256))
ans = a
```

Manipulação de strings

- Strings são vetores; portanto, podem ser manipuladas da mesma forma que os vetores númericos:

```
>> S
S =
Exemplo de string

>> S(3:12)
ans =
emplo de s

>> S(end:-1:1)
ans =
gnirts ed olpmexE

>> (S')'
ans =
Exemplo de string
```

Concatenação de strings

- O operador de concatenação é o mesmo: $[\,]$

```
>> S1 = '. Outro exemplo de string'
S1 = . Outro exemplo de string

>> [S S1]
ans =
Exemplo de string. Outro exemplo de string
```

- O comando $disp$ suprime o nome da variável de saída. Assim, no exemplo anterior se usássemos `disp([S S1])` seria suprimida a linha `ans =`.

Concatenação de strings

- Strings podem ter múltiplas linhas, mas todas as linhas devem ter o mesmo número de colunas.

```
>> [S; S1]
 ??? Error using ==> vertcat
All rows in the bracketed expression must have
the same number of columns.

>> whos
  Name        Size                Bytes  Class
  S           1x17                   34  char array
  S1          1x25                   50  char array
  (...)

>> S = ['          ' S]; % + colunas

>> disp([S; S1])
          Exemplo de string
. Outro exemplo de string
```

Funções *char* e *strvcat*

- Concatenam strings de diversos tamanhos ao longo de uma coluna. Após a operação todas as linhas possuem o mesmo número de colunas. A função *strvcat* faz o mesmo mas ignora strings vazias.

```
>> char('um','','dois','tres')
ans =
um

dois
tres

>> strvcat('um','','dois','tres')
ans =
um
dois
tres
```

Função *deblank*

- *deblank*: elimina os espaços em branco criados à direita para alinhamento em matrizes.

```
>> S2=strvcat('um','','dois','tres');

>> size(S2(1,:))
ans =
     1     4

>> size(deblank(S2(1,:)))
ans =
     1     2
```

Função $strcat$

- $strcat$: executa a concatenação horizontal. É preciso que todas as strings possuam o mesmo número de linhas. Espaços em branco à direita são ignorados.

```
>> S = char('','um','dois','tres');
>> S1 = strvcat('','um','dois','tres');

>> strcat(S,S1)
Error using ==> strcat (...)

>> S = [S(2,:);S(3,:);S(4,:)];  size(S)
ans =
     3     4

>> strcat(S,S1)
ans =
umum
doisdois
trestres
```

Conversão de números para strings

- Função $int2str$: inteiros $\rightarrow$ strings (arredonda o número antes da conversão quando este não é inteiro);

```
>> int2str(eye(2))
ans =
1  0
0  1

>> disp(size(ans)) %'1  0' tem 4 caracteres
     2     4
```

- $num2str$: números quaisquer $\rightarrow$ strings (detalhes em *help*)

```
>> num2str([logspace(0,2,4);logspace(1,3,4)])
ans =
 1       4.641589       21.54435            100
10       46.41589       215.4435           1000

>> disp(size(ans))
     2    44
```

Função $mat2str$

- Converte para uma string padrão que seria usada caso quiséssemos construir o vetor na janela de comando do **MATLAB** .

```
>> mat2str(pi*eye(2))
ans =
[3.14159265358979 0; 0 3.14159265358979]

>> size(ans)
ans =
     1    40

>> mat2str([0,0,1; 0   0 1])
ans =

[0 0 1;0 0 1]
```

Funções $sprintf$ e $fprintf$

- Funções que escrevem dados de acordo com uma formatação especificada pelos seus parâmetros;
- Semelhantes àquelas de mesmo nome da linguagem C, padrão ANSI;
- A função $fprintf$ é muito usada para converter resultados numéricos para o formato ASCII e anexá-los a um arquivo de dados. O identificador do arquivo é o primeiro parâmetro da função; se ausente, implica que o resultado será exibido na janela de comandos;
- A função $sprintf$ tem o mesmo comportamento de $fprintf$, mas escreve seu resultado em uma string em vez de um arquivo.
- Veremos alguns exemplos de seus usos, mas maiores informações podem ser obtidas no *help on line* e em outros exemplos e usos constantes da bibliografia.

Exemplo de uso de $fprintf$ e $sprintf$

```
>> num = 3; raio = sqrt(7); area = pi*raio^2;

>> fprintf('Circulo%2d: raio=%.5f, area=%.5g'...
, num, raio, area)
Circulo 3: raio=2.64575, area=21.991

>> sprintf('Circulo%2d: raio=%.5f, area=%.5g'...
, num, raio, pi*raio^2)
ans =
Circulo 3: raio=2.64575, area=21.991

>> disp(class(ans))
char
```

- Note que a saída de $sprintf$ ocorre em uma variável (string), permitindo que haja tratamento deste resultado posteriormente. Por exemplo: inserir dados em um gráfico, criar nomes para arquivos, entre outros.

Conversão de strings para números

- $str2num$: significado usual, *help on line* para mais detalhes;

```
>> S = num2str(pi*eye(2)),  N = str2num(S)
S =                          N =
3.1416            0          3.1416            0
     0      3.1416                0    3.1416

>> whos
  Name        Size              Bytes  Class
  N           2x2                  32  double array
  S           2x18                 72  char array
Grand total is 40 elements using 104 bytes

>> pi*eye(2) - N % perda de precisão
ans =
   1.0e-05 *
   -0.7346         0
         0   -0.7346
```

Conversão de strings para números

- A função $str2num$ permite que usemos expressões nas strings, mas não permite o uso de variáveis.

```
>> r = 10;
>> S = '[2*pi*r pi 1+2i]'
S =
[2*pi*r pi 1+2i]

>> str2num(S) % com o r não funciona
ans =
     []

>> S = '[2*pi*10 pi 1+2i]';

>> disp(str2num(S))
  62.8319   3.1416   1.0000 + 2.0000i
```

Conversão de strings para números

- $str2double$: quando a conversão exige um valor único de precisão dupla. Costuma ser mais rápida, mas tem um escopo de atuação mais limitado.

```
>> str2double('inf') % converte o infinito
ans =
   Inf

>> disp(class(ans))
double

>> str2double('1+2i') % OK com complexos.
ans =
   1.0000 + 2.0000i

>> disp(class(ans)) % Não com variáveis e exp.
double

>> str2double('pi')
ans =
   NaN
```

Função $sscanf$

- Esta função lê os dados de uma string seguindo um formato especificado. Veremos apenas alguns exemplos, veja *help on line* para maiores detalhes.

```
>> V = version
V =
6.0.0.88 (R12)

>> sscanf(V,'%f')          >> disp(sscanf(V,'%f', 2))
ans =                          6
    6.0000                     0
         0
    0.8800                  >> disp(sscanf(V,'%d'))
                               6
>> disp(sscanf(V,'%s')) %remove espaço em branco
6.0.0.88(R12)
```

Execução de strings

- Em algumas situações pode ser necessário 'executar' uma string como se fosse uma sentença qualquer do **MATLAB** . Isto é, obviamente, muito mais do que fazem $str2num$ e $str2double$;

- A execução de strings é feita pelas funções $eval$ e $evalc$. Estas recorrem a todo o interpretador do **MATLAB** para avaliar qualquer string que respeite a sintaxe do **MATLAB** . Consequentemente estas funções exigem muito do computador;

- A função $feval$ é uma função para execução de strings cujo escopo de atuação é bem mais restrito e portanto não recorre ao interpretador do **MATLAB** . Em virtude disso esta é bem mais rápida que as acima.

A função *eval*

```
>>[triang(1:3).tipo] = deal('ret','isosc','desc');
>>[triang.center]=deal(zeros(1,2),ones(1,2),rand(1,2));
>> names = fieldnames(triang);

>> S = sprintf('t = triang(%d).%s',1,names(1))
??? Error using ==> sprintf
 Function 'sprintf' not defined for variables of class 'cell'
% precisamos de uma string e não de uma célula
>> S = sprintf('t = triang(%d).%s',1,names{1})
S =
t = triang(1).tipo

>> eval(S)
t = ret
>>eval(sprintf('t = triang(%d).%s',3,names{1}))
t = desc
```

A função $eval(try, catch)$

- Quando ocorre um erro na execução da string, o **MATLAB** emite uma mensagem de erro e nada é atribuído à variável de saída;
- É possível controlar o que ocorre na execução da string usando a opção *try-catch*. A sintaxe é $eval(try, catch)$. A primeira parte (*try*) é a string que será avaliada, a segunda parte (*catch*) contém uma outra string que apenas será executada se ocorrer um erro na execução da primeira parte.

```
>>eval(sprintf('t=circle(%d).%s',3,names{1}),'NaN')
t =
NaN
```

A função $eval(try, catch)$ - exemplo

```
>> eval(sprintf('t =TR(%d).%s',3,names{1}),...
't =NaN;Flag=1;')
>>
>> t,Flag
t =
    NaN
Flag =
     1

>>lasterr
ans =
Undefined variable 'TR'.
```

- Não ocorre saída na execução da função $eval$ porque a parte *catch* contém **';'** nos seus comandos;
- A função *lasterr* armazena a string que descreve o erro que ocorreu na tentativa de executar a parte *try*.

A função $evalc$

- A entrada e atuação desta função são exatamente as mesmas que as da função $eval$. A diferença reside na saída: a função $evalc$ retorna o resultado como uma string de caracteres.
- Este recurso surgiu da necessidade de exibir os resultados como um texto em um outra janela, por exemplo uma GUI.

```
>> eval(sprintf('t =triang(%d).%s',3,names{1}))
ans =
t =
desc

>>disp(class(ans))
char
```

A função $feval$

- Função para execução de strings que são nomes de funções. Isto é, esta função assume que a string seja um nome de função válido (desta forma não recorrendo ao interpretador) seguida de seus argumentos.

```
>> nameF = 'cos';
>> x = linspace(0,pi,5);

>> y = feval(nameF,x)
% y = nameF(x)
>> y =
   0   0.70711   6.1232e-017   -0.70711   -1
```

- O uso desta função não se limita a funções de argumentos únicos, tanto de entrada, quanto de saída. Sua sintaxe geral é:

```
[x,y,z,...]=feval('func',a,b,c,...)
```

Vetores de células com strings

- Vetores de células são usados para contornar a exigência de que em um vetor de caracteres com múltiplas linhas, estas precisem ter o mesmo número de colunas. Este é o uso mais comum de vetores de células.
- Para isso construímos um vetor de células em que cada elemento do vetor é uma string (de tamanhos quaisquer).

```
>> clear all
>> V = {'->Elemento 11' '->El. 12' ;...
 '->Elemento 21' '->Elemento 22'}
V =
    '->Elemento 11'    '->El. 12'
    '->Elemento 21'    '->Elemento 22'

>> disp(class(V))
cell
```

Funções úteis

- $iscellstr(V)$: retorna 1 se todas as células do vetor de células são strings e 0 caso contrário.
- $char$ e $strvcat$: convertem um vetor de células, cujas células são strings, em um vetor de strings.

```
>> V = {'->El. 11' '->El. 12' ; '->E21' ''}
V =  '->El. 11'    '->El. 12'
     '->E21'                ''
>> S1 = char(V), S2 = strvcat(V)
S1 =                    S2 =
->El. 11                ->El. 11
->E21                   ->E21
->El. 12                ->El. 12
>> whos
  Name        Size              Bytes  Class
  S1          4x8                  64  char array
  S2          4x8                  64  char array
  V           2x2                 410  cell array
Grand total is 89 elements using 538 bytes
```

Funções úteis

- $cellstr$: converte um vetor de strings em um vetor (coluna) de células, onde cada célula é uma string.

```
>> S1
S1 =
->El. 11
->E21
->El. 12


>> disp(cellstr(S1))
    '->Elemento 11'
    '->Elemento 21'
    '->El. 12'
                  ''
```

- Note que não voltamos ao vetor de células original. Se o vetor de células original fosse um vetor coluna teríamos retornado ao vetor original.