MATLAB Avançado
Aula 2

Melissa Weber Mendonça
melissa.mendonca@ufsc.br

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Estruturas

Outra maneira de armazenar dados heterogêneos é usar *estruturas*: cada estrutura é composta de campos que podem conter quaisquer tipos de dados (assim como as células), e que são referenciados por *nome*. Para criarmos uma estrutura chamada `dados` com o campo chamado `Nome`, podemos usar diretamente a sintaxe

```
>>  dados.Nome = 'Melissa'
>>  dados.Sobrenome = 'Mendonca'
```

ou

```
>>  dados = struct('Nome', 'Melissa', ...
        'Sobrenome', 'Mendonca')
```

Exemplos de uso

```
>>  dados = struct('Nome', 'Melissa', ...
        'Sobrenome', 'Mendonça')
>>  dados(1)
>>  dados(1).Nome
>>  dados(2) = struct('Nome', 'Fulano', ...
        'Sobrenome', 'Beltrano')
>>  dados(1)
>>  dados(2)
>>  dados.Nome
>>  [nome1, nome2] = dados.Nome
```

Obs. Para criarmos uma struct vazia, podemos usar o comando

```
>>  vazia = struct([])
```

Campos

As structs possuem campos nomeados, o que pode tornar mais fácil acessar os dados armazenados nesse tipo de variável. Alguns comandos do MATLAB permitem fazer isso.

- O comando `fieldnames(s)` permite recuperar em uma célula a lista dos nomes dos campos da struct `s`.

Campos

As structs possuem campos nomeados, o que pode tornar mais fácil acessar os dados armazenados nesse tipo de variável. Alguns comandos do MATLAB permitem fazer isso.

- O comando `fieldnames(s)` permite recuperar em uma célula a lista dos nomes dos campos da struct `s`.
- O comando `s = orderfields(s1)` ordena os campos da struct `s1` de modo que a nova struct `s` tem os campos em ordem alfabética.

Campos

As structs possuem campos nomeados, o que pode tornar mais fácil acessar os dados armazenados nesse tipo de variável. Alguns comandos do MATLAB permitem fazer isso.

- O comando `fieldnames(s)` permite recuperar em uma célula a lista dos nomes dos campos da struct `s`.
- O comando `s = orderfields(s1)` ordena os campos da struct `s1` de modo que a nova struct `s` tem os campos em ordem alfabética.
- O comando `s = orderfields(s1, s2)` ordena os campos da struct `s1` de forma que a nova struct `s` tenha os nomes dos campos na mesma ordem em que aparecem na struct `s2` (as structs `s1` e `s2` devem ter os mesmos campos).

Campos

As structs possuem campos nomeados, o que pode tornar mais fácil acessar os dados armazenados nesse tipo de variável. Alguns comandos do MATLAB permitem fazer isso.

- O comando `fieldnames(s)` permite recuperar em uma célula a lista dos nomes dos campos da struct `s`.
- O comando `s = orderfields(s1)` ordena os campos da struct `s1` de modo que a nova struct `s` tem os campos em ordem alfabética.
- O comando `s = orderfields(s1, s2)` ordena os campos da struct `s1` de forma que a nova struct `s` tenha os nomes dos campos na mesma ordem em que aparecem na struct `s2` (as structs `s1` e `s2` devem ter os mesmos campos).
- O comando `s = orderfields(s1, c)` ordena os campos em `s1` de forma que a nova struct `s` tenha campos na mesma ordem em que aparecem na célula `c` (a célula `c` deve conter apenas os nomes dos campos de `s1`, em qualquer ordem).

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Structs e células

Podemos preencher uma struct usando um comando só, determinando os valores possíveis para cada campo através de células. Por exemplo, no caso anterior, poderíamos ter entrado o comando

```
>>  dados = struct('Nome', {'Melissa', 'Fulano'}, ...
        'Sobrenome', {'Mendonça', 'Beltrano'})
```

para criar a mesma struct.
Se quisermos preencher vários campos com o mesmo valor, não precisamos nos repetir. Por exemplo,

```
>>  dados = struct('Nome', {'Melissa', 'Fulano'}, ...
        'Sobrenome', {'Mendonça', 'Beltrano'}, ...
        'Presentes', {'sim'})
```

cell2struct

A função **cell2struct** cria uma estrutura a partir dos dados contidos na célula:
Se Exemplo:

```
>>  tabela = {'Melissa', 'Mendonça', 'sim';
        'Fulano', 'Beltrano', 'sim'}
>>  campos = {'Nome', 'Sobrenome', 'Presente'};
>>  s = cell2struct(tabela, campos, 2);
>>  s(1)
>>  s(2)
```

struct2cell

Por outro lado, o comando

```
>>  célula = struct2cell(struct)
```

cria uma célula a partir da estrutura *struct*.
Exemplo:

```
>>  celula = struct2cell(s)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Funções

Funções

Na matemática,

$$f(x) = y.$$

Entrada: x
Saída: y
Ação: f.
Exemplo: $f(x) = x^2$.

Funções já prontas

Exemplos:

```
n = input('Entre com um numero:')
nfat = factorial(n)
texto = num2str(25)
```

Funções

No MATLAB, uma função é um arquivo `minhafuncao.m` com a sintaxe

minhafuncao.m

```
function  [y] = minhafuncao(x)
          % Descricao da funcao
          comandos;
```

Uma vez construida a função, podemos chamá-la no console, usando

```
>>  y = minhafuncao(x)
```

Observação. Uma função deve sempre ter o mesmo nome que o arquivo no qual ela está salva.

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Exemplo

Construir uma função que calcule a média dos 3 elementos de um vetor x.

media.m

```
function  [y] = media(x)
          y = (x(1) + x(2) + x(3))/3;
```

Qual a diferença entre um *script* e uma *função*?

Um *script* é um arquivo que contém uma sequência de comandos, mas não exige entrada ou saída.
Uma função deve, obrigatoriamente, ter pelo menos uma entrada e uma saída.

Argumentos de entrada e saída

Se tivermos mais de um argumento de entrada, basta separá-los por vírgulas:

```
s = soma(x,y)
```

Se tivermos mais de um argumento de saída, precisamos escrevê-los entre colchetes:

```
[a,b] = somaesubtracao(x,y)
```

Se quisermos aplicar a mesma função a um conjunto de valores, basta colocarmos os valores em um vetor:

```
m = f([-2 1 3])
```

Funções com número variável de argumentos

- O comando `nargin`, executado dentro do corpo de uma função, retorna o número de argumentos de entrada para o qual a função está definida.
- O comando `nargin(f)`, em que `f` é uma função, retorna o número de argumentos de entrada da função `f`, e pode ser executado fora da função (inclusive no console).
- O comando `nargout`, executado dentro do corpo de uma função, retorna o número de argumentos de saída para o qual a função está definida.
- O comando `nargin(f)`, em que `f` é uma função, retorna o número de argumentos de saída da função `f`, e pode ser executado fora da função (inclusive no console).

Funções com número variável de argumentos

`varargin` é uma variável de entrada que permite que a função receba qualquer número de argumentos de entrada.

Exemplo:

somas.m

```
function [y] = somas(x,varargin)
    if nargin == 1
      disp('Nada a calcular.')
    elseif nargin == 2
      y = x + varargin{1};
    elseif nargin == 3
      y = x + varargin{1} + varargin{2};
    else
      disp('Argumentos demais!')
    end
```

Funções com número variável de argumentos

`varargout` é uma variável de saída que permite que a função devolva qualquer número de argumentos de saída.

Exemplo:

valores.m

```
function [f,varargout] = valores(x)
    f = x^2;
    if nargout == 2
      varargout{1} = 2*x;
    elseif nargout == 3
      varargout{1} = 2*x;
      varargout{2} = 2;
    elseif nargout > 3
      disp(['A função aceita até 3 ' ...
        'argumentos de saída.'])
    end
```

Funções anônimas

Para declarar uma função no console, sem ter que guardá-la em um arquivo, podemos usar o conceito de *função anônima*.
Exemplo:

```
>>  f = @(x) sin(x)
>>  x = pi;
>>  f(x)
```

Exemplo:

```
>>  f = @(x) x-1;
>>  min(f([-2 1 0]))
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Funções anônimas com mais de uma variável

Se quisermos criar uma função anônima com mais de uma variável, usamos

```
>>  f = @(x,y,z,t) x+y+z+t
```

Para retornarmos mais de um valor de uma função anônima, usamos o comando *deal*:

```
>>  f = @(t,u,v) deal(t+u+v,t-u+2*v)
>>  [a,b] = f(1,2,3)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Funções anônimas: Exemplo

Pra que servem funções anônimas?

```
>>  x = -3:0.1:3;
>>  f = @(x) x.^2+3*x
>>  plot(x,f(x));
```

Manipulação de Arquivos e Tratamento de Dados

Importar dados

Para importarmos dados, o método mais fácil é utilizar a interface gráfica do MATLAB, selecionando

$$\text{File} \rightarrow \text{Import Data}$$

Para verificar os tipos de arquivo suportados e as funções disponíveis, consulte o Help.

importdata

Para importarmos dados de maneira automática, podemos usar o comando

```
>>  importdata(arquivo, separador, ncabecalho)
```

Leitura de dados numéricos: `load`

Para lermos um arquivo com dados **numéricos** chamado `dados.txt`, usamos o comando

```
>>  load dados.txt;
```

Em seguida, na variável `dados` estarão contidos os valores obtidos do arquivo `dados.txt`.
Se quisermos também podemos usar a sintaxe

```
>>  A = load('dados.txt')
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Abrir e fechar um arquivo

Para abrir um arquivo chamado `nome.txt`, usamos o comando

```
arquivo = fopen('nome.txt')
```

Sempre que abrimos um arquivo, precisamos fechá-lo antes de sair do nosso programa. Para isso, usamos o comando

```
fclose(arquivo)
```

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Comandos: leitura

Para ler dados de um arquivo, precisamos indicar que tipo de informação estamos procurando. Isto é feito através dos *formatos* abaixo:

- Números inteiros: `%d` ou `%u`
- Números reais: `%f` (notação decimal) ou `%e` (notação científica)
- Texto com espaços: `%c`
- Texto sem espaços: `%s`
- Nova linha: `\n` (sinaliza o fim de uma linha de dados)

Para lermos dados de um arquivo em uma célula, usamos

```
C = textscan(arquivo,'%d')
```

Para lermos dados de um arquivo em uma matriz, usamos

```
A = fscanf(arquivo,'%d')
```

Exemplo

1. Crie um arquivo chamado

   `info.txt`

   no mesmo diretório em que está salvando seus programas, com um número inteiro dentro.
2. No console, faça

```
>> arquivo = fopen('info.txt')
>> a = fscanf(arquivo,'%d')
>> fclose(arquivo)
```

Verifique que a variável `a` vale o mesmo que seu inteiro no arquivo.