Sistemas e Sinais

Laboratório 0 (parte b)

(MATLAB®: as instruções *if-else*, *switch*, *for*, *while* e funções)

Miguel Pedro Silva e João Reis

Instituto Superior Técnico,
Dep. de Engenharia Mecânica - Secção Sistemas,
Grupo de Controlo Automação e Robótica

Exercício 1

- A função sigmoide é a a função que geralmente define a relação entrada/saída de um neurónio artificial (o elemento constituinte das redes neuronais).
- A sua expressão é a seguinte:

$$y = 1 / (1+ e^{-Kx})$$

- Escreva uma função, denominada sigmoide, que recebe um array de valores x, um valor real K, e retorna um array com os valores de y.

- Teste a função com um script de teste.

Exercício 1 (cont.) solução

```
function y = sigmoide(x,K)
% A função SIGMOIDE calcula a relação
% entrada-saída da função y = 1 / (1+ e^(-K*x)).
%
% Chamada: y =  sigmoide(x,K), em que K é um escalar
% e x pode ser um escalar ou um array uni-dimensional

[linhas_K, colunas_K] = size(K);
[linhas_x, colunas_x] = size(x);
if (linhas_x > 1 && colunas_x > 1)
    error('x tem que se um array uni-dimensional.')
elseif ~(linhas_K == 1 && colunas_K == 1)
    error('K tem que ser um escalar.')
end
y =  1./(1 + exp(-K*x));
```

Exercício 1 (cont.) *script* de teste

```
%Este script testa a função SIGMOIDE
clc;
clear all;
close all;

disp('Este script testa a função SIGMOIDE');
%pede o valor de x e testa o valor
x_incorrecto = true;
while (x_incorrecto)
   entrada_x = ...
   input('Introduza um escalar ou array uni-dimens. x [e.g.: [-5:0.01:5]]: ');
   [linhas_entrada_x, colunas_entrada_x] = size(entrada_x);
   x_incorrecto = (linhas_entrada_x > 1 && colunas_entrada_x > 1);
end
```

Exercício 1 (cont.) *script* de teste

```
%pede 3 valores de K e testa os seus valores
K_incorrecto = true;
while (K_incorrecto)
   entrada_K_1 = input('Introduza um escalar K_1 [e.g.: 1]: ');
   entrada_K_2 = input('Introduza um escalar K_2 [e.g.: 10]: ');
   entrada_K_3 = input('Introduza um escalar K_3 [e.g.: 100]: ');
   entrada_K = [entrada_K_1, entrada_K_2, entrada_K_3];
   [linhas_entrada_K, colunas_entrada_K] = size(entrada_K);
   K_incorrecto = (linhas_entrada_K ~= 1 || colunas_entrada_K ~= 3);
end

figure(100), hold on;
for j=1:3,
   subplot(3,1,j),
   plot(entrada_x, sigmoide(entrada_x, entrada_K(j)));
   title(['K_', int2str(j), ' = ', int2str(entrada_K(j))]), grid on;
end
```

Exercício 2

- Escreva uma função, denominada remove_elementos, que receba como argumentos dois *arrays* (vectores).

  O primeiro *array* é o *array* de onde se pretende remover elementos, e o segundo *array* é constituído pelos índices dos elementos que se pretendem retirar.

  A função deve retornar o *array* com os elementos removidos, e um outro *array* com os valores que foram removidos.

- Teste a função com um *script* de teste.

Exercício 2 (cont.) solução

```
function [arr_mod, elem_remov] = remove_elementos(arr,ind_a_remov)
% A função REMOVE_ELEMENTOS recebe dois arrays unidimendionais
% como parâmetros de entrada.
% O primeiro array é o array que se pretende remover elementos,
% e o segundo array é constituídos pelos índices que se pretendem
% retirar ao array a manipular.
% A função retornar o array com os elementos removidos, e um outro
% array com os valores que foram removidos do array manipulado.

[linhas_arr, colunas_arr] = size(arr);
[linhas_ind_a_remov, colunas_ind_a_remov] = size(ind_a_remov);

if (linhas_arr > 1 && colunas_arr > 1)
   error('O primeiro argumento tem que ser uni-dimensional.');
elseif (linhas_ind_a_remov > 1 && colunas_ind_a_remov > 1)
   error('O segundo argumento tem que ser uni-dimensional.');
```

Exercício 2 (cont.) solução

```
elseif (linhas_ind_a_remov == 0 && colunas_ind_a_remov == 0),
   % se receber um array vazio []
   arr_mod = arr;      %array mantém-se
   elem_remov = [];    %elementos removidos é um array vazio []
   return;
elseif (min(ind_a_remov) < 1 || ...
                max(ind_a_remov) > (max([linhas_arr, colunas_arr])))
   error('Não é possível remover um, ou mais, dos índices desejados.');
end

arr_logico = logical(ones(linhas_arr, colunas_arr));
arr_logico(ind_a_remov) = false;      % coloca um zero lógico nas
                                      % posições a remover
arr_mod = arr(arr_logico);            %remove os elementos
elem_remov = arr(ind_a_remov);        %obtem os elementos removidos
```

Exercício 2 (cont.) *script* de teste

```
%Este script testa a função REMOVE_ELEMENTOS

clc;
clear all;
close all;

disp('Este script testa a função REMOVE_ELEMENTOS');
%pede o array uni-dimensional a modificar e testa-o
array_incorrecto = true;
while (array_incorrecto)
   entrada_array = input('Introduza um array uni-dimensional: ');
   [linhas_entrada_array, colunas_entrada_array] = ...
                                                  size(entrada_array);
    array_incorrecto = (linhas_entrada_array > 1 && ...
                                          colunas_entrada_array > 1);
end
```

Exercício 2 (cont.) *script* de teste

```
%pede o array uni-dimensional com os índices a remover e testa-o
indices_incorrecto = true;
while (indices_incorrecto)
    entrada_indices = ...
            input('Introduza um array com os índices a remover: ');
    [linhas_entrada_indices, colunas_entrada_indices]= ...
                                                size(entrada_indices);
    indices_incorrecto = ...
       (linhas_entrada_indices > 1 && colunas_entrada_indices > 1);
end
[arr_mod, elem_remov] = ...
                    remove_elementos(entrada_array, entrada_indices);

disp('array original:');
disp(entrada_array);
disp('Array modificado:');
disp(arr_mod);
disp('Elementos removidos:');
disp(elem_remov);
```

Exercício 3

- Escreva uma função, denominada calcula_polinomio, que receba como argumentos:
  - 1 *array* (vector), com os coeficientes do polinómio (por ordem decrescente);
  - 1 *array* (vector), com os pontos para os quais se pretende calcular o valor do polinómio;
- A função deve retornar:
  - 1 array com a avaliação do polinómio nos pontos pretendidos.

- Teste a função com um *script*.

Exercício 3 (cont.) solução

```
function [val_pol] = calcula_polinomio(coef, pontos)
% A função CALCULA_POLINOMIO recebe dois arrays unidimendionais
% como parâmetros de entrada:
% um array com os coeficientes do polinómio (por ordem decrescente);
% um array com os pontos para os quais que se pretende calcular o
%           valor do polinómio;
% A função retorna:
% um array com a avaliação do polinómio nos pontos pretendidos.

[linhas_coef, colunas_coef] = size(coef);
[linhas_pontos, colunas_pontos] = size(pontos);

if (linhas_coef > 1 && colunas_coef > 1)
   error('O primeiro argumento tem que ser uni-dimensional.');
elseif (linhas_pontos > 1 && colunas_pontos > 1)
   error('O segundo argumento tem que ser uni-dimensional.');
elseif (linhas_coef == 0 && colunas_coef == 0)
   error('O primeiro argumento tem que ser não-vazio.');
```

Exercício 3 (cont.) solução

```
elseif (linhas_pontos == 0 && colunas_pontos == 0)
   error('O segundo argumento tem que ser não-vazio.');
end

if linhas_coef > 1
   coef = coef'; % se for um vector coluna transforma em vector linha
end

for k=1:length(pontos)
    vec_k_aux = [pontos(k)*ones(1,length(coef))];
                                %vector linha com cada ponto k
    vec_exp_aux = [length(coef)-1:-1:0];
                                %vector linha com os exponentes
    val_pol(k) = sum(coef.*(vec_k_aux.^vec_exp_aux));
end
```

Exercício 3 (cont.) *script* de teste

```
%Este script testa a função CALCULA_POLINOMIO
clc; clear all; close all;

disp('Este script testa a função CALCULA_POLINOMIO');
% pergunta quantos polinómios pretende testar
valor_incorrecto = true;
while (valor_incorrecto)
   num_pol = input('Introduza o número de polinómios a testar [1 a 3]: ');
   valor_incorrecto = (num_pol > 3 || num_pol < 1);
end

% pede um array uni-dimensional com os pontos a calcular
pontos = input('Introduza um array com os pontos a calcular: ');
% pede um array com os coeficientes do polinómio
for k=1:num_pol,
   pol = input('Introduza um array com os coefs. do polinómio: ');
   figure(k); hold on;
   plot(pontos, calcula_polinomio(pol,pontos)),
   title(['Polinómio ' int2str(k)]), grid on;
end
```

Exercício 4

- A famosa sequência de Fibonacci, foi descrita por Leonardo de Pisa, também conhecido como Fibonacci (1170-1250) , para descrever o crescimento de uma população de coelhos. Os números descrevem o número de casais numa população de coelhos depois de *n* meses, considerando um conjunto pressupostos.
  (ver: http://pt.wikipedia.org/wiki/N%C3%BAmero_de_Fibonacci)

A sequência é dada pela seguinte expressão:

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Primeiros termos: (0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...),

Exercício 4 (cont.)

- Implemente uma função recursiva (denominada fibo.m) que calcule o enésimo termo da sequência de Fibonacci.

- Teste a função com um *script*.

Exercício 4 (cont.) solução

```
function [termo_n] = fibo(n)
% A função FIBO recebe
% como parâmetro de entrada:
% um valor n inteiro >=1;
% A função retorna:
% o n-termo da sequência de Fibonacci.

if (n < 1 || round(n) ~= n)
    error('O valor tem que ser um inteiro maior ou igual a 1.');
end
switch (n)
    case 1,
        termo_n = 0;
    case 2,
        termo_n = 1;
    otherwise,
        termo_n = fibo(n-1) + fibo(n-2);
end
```

Exercício 4 (cont.) *script* de teste

```
%Este script testa a função FIBO
clc; clear all; close all;

disp('Este script testa a função FIBO');
%pede o termo N até ao qual se pretende calcular a sequência
valor_incorrecto = true;
while (valor_incorrecto)
    N = input(['Introduza o termo N até onde pretende calcular ', ...
                                                'a seq. de Fibonacci: ']);
    valor_incorrecto = (N < 1 || round(N) ~= N);
end
seq=[];
for k=1:N
    seq=[seq,fibo(k)];
end
figure(20);
stem(1:N, seq), grid on;
title(['Sequência de Fibonacci até ao termo ' int2str(N)]);
```