Jules LELAY

3I-SI3 : Traitement du signal

TP1 : Formation Matlab

Le rendu se compose des fichiers des 3 fonction demandées ainsi que d'un script de démonstration.

Exercice 1 : Calcul du sinus cardinal

Ce premier exercice requiert le calcul du sinus cardinal d'un scalaire.
Nous commençons donc par demander en argument de cette fonction un double "x"

```
function y = exercice1(x)
    arguments
        x double
    end
```

Il nous faut ensuite calculer la valeur de retour, soit le sinus cardinal du nombre passé en paramètre, et l'assigner à la variable de retour "y"

```
if x == 0           % Si x=0
    y = 1;          %   sinc(x) = lim(sin(x)/x, x->0) = 1 retourne
else                % Sinon
    y = sin(x)/x;   %   sinc(x) = sin(x)/x retourne
end                 % Fin si
```

Nous obtenons donc la fonction suivante, retrouvable dans le fichier .m éponyme :

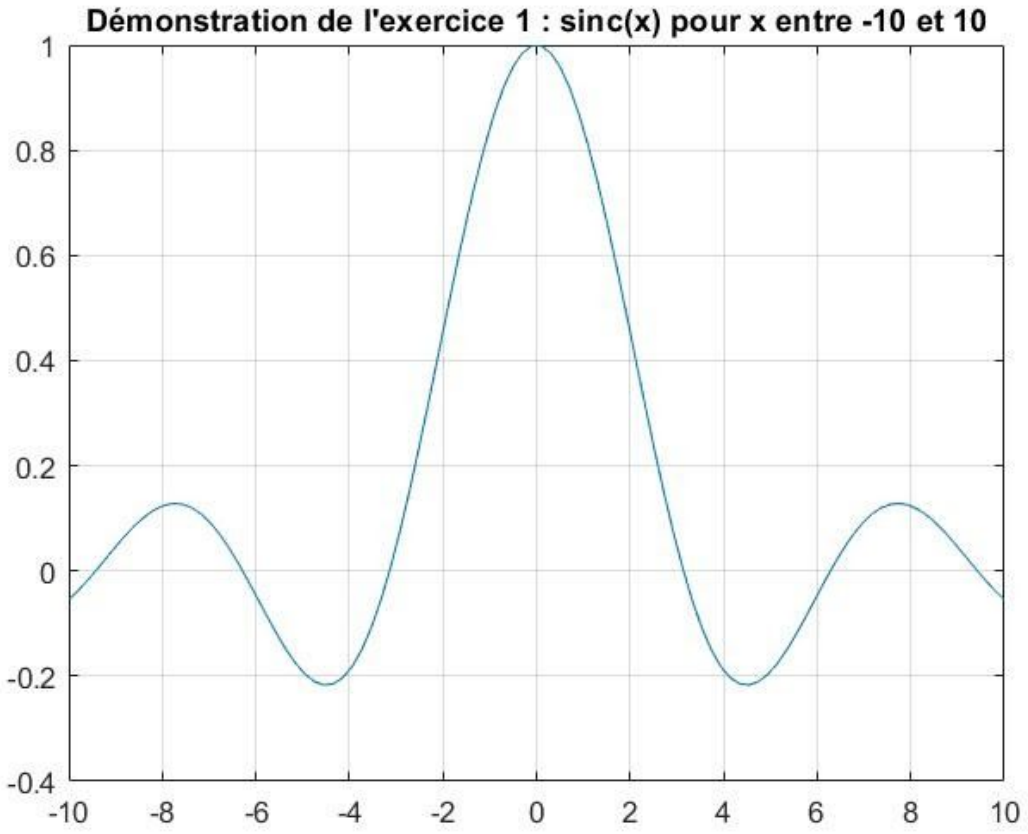
```
% Calcule du sinus cardinal d'un scalaire (1 si 0, sin(x)/x sinon)
function y = exercice1(x)
    arguments
        x double
    end

    if x == 0           % Si x=0
        y = 1;          %   sinc(x) = lim(sin(x)/x, x->0) = 1 retourne
    else                % Sinon
        y = sin(x)/x;   %   sinc(x) = sin(x)/x retourne
    end                 % Fin si
end
```

Pour la démonstration, nous générons un vecteur espacé linéairement sur l'interval $[-10; 10]$ et attribuons à un second vecteur les valeurs de retour des éléments du premier, avant d'afficher le résultat sous forme de graphique :

```
% Demonstration de l'exercice 1
X = linspace(-10,10);
Y = arrayfun(@(x) exercice1(x),X);
figure;
plot(X,Y)
title("Démonstration de l'exercice 1 : sinc(x) pour x entre -10 et 10")
grid
```

On obtient alors la représentation graphique suivante :



Exercice 2 : Génération de carrés magiques

Ce second exercice requiert la génération des carrés magiques d'ordre 5 à 11 avec un pas de 2.
Nous commençons par générer l'expression utilisée puis l'évaluer, avant de boucler sur les valeurs demandées dans l'énoncer.
Nous obtenons donc la fonction suivante, retrouvable dans le fichier .m éponyme :

```
% Genere les carres magiques de dimension d de 5 à 11 avec un pas de 2
%   dans les variables M{d}
function [M5,M7,M9,M11] = exercice2()
    for i=5:2:11
        s = sprintf('M%d = magic(%d);',i,i);    % Generation de l'expression
        eval(s);                                % Evaluation de l'expression
    end
end
```

Pour la démonstration, nous générons les carrés magiques et générons, pour i prenant les valeurs impaires de 5 à 11, "Mi = *[Mi]*" avant de l'évaluer (code ci-dessous).

```
% Demonstration de l'exercice 2
disp(" ")
disp("Demonstration de l'exercice 2 :")
disp(" ")

[M5,M7,M9,M11]=exercice2();                          % Generation des carrés

for i=5:2:11                                         % Pour i 5->11 (pas: 2)
    s = sprintf('disp("M%d =") \n disp(M%d)',i,i);   %   s = M{i} = {M{i}}
    eval(s);                                         % Afficher s
end                                                  % Fin pour
```

On obtient alors le résultat suivant :

```
Demonstration de l'exercice 2 :

M5 =
    17    24     1     8    15
    23     5     7    14    16
     4     6    13    20    22
    10    12    19    21     3
    11    18    25     2     9

M7 =
    30    39    48     1    10    19    28
    38    47     7     9    18    27    29
    46     6     8    17    26    35    37
     5    14    16    25    34    36    45
    13    15    24    33    42    44     4
    21    23    32    41    43     3    12
    22    31    40    49     2    11    20

M9 =
    47    58    69    80     1    12    23    34    45
    57    68    79     9    11    22    33    44    46
    67    78     8    10    21    32    43    54    56
    77     7    18    20    31    42    53    55    66
     6    17    19    30    41    52    63    65    76
    16    27    29    40    51    62    64    75     5
    26    28    39    50    61    72    74     4    15
    36    38    49    60    71    73     3    14    25
    37    48    59    70    81     2    13    24    35

M11 =
    68    81    94   107   120     1    14    27    40    53    66
    80    93   106   119    11    13    26    39    52    65    67
    92   105   118    10    12    25    38    51    64    77    79
   104   117     9    22    24    37    50    63    76    78    91
   116     8    21    23    36    49    62    75    88    90   103
     7    20    33    35    48    61    74    87    89   102   115
    19    32    34    47    60    73    86    99   101   114     6
    31    44    46    59    72    85    98   100   113     5    18
    43    45    58    71    84    97   110   112     4    17    30
    55    57    70    83    96   109   111     3    16    29    42
    56    69    82    95   108   121     2    15    28    41    54
```

Exercice 3 : Somme des inverses des carrés supérieurs à epsilon

Ce troisième et dernier exercice requiert le calcul de la somme des inverses des carrés supérieurs au epsilon passé en paramètres.
Nous commençons donc par demander en argument de cette fonction un double “epsilon” :

```
function y = exercice3(epsilon)
    arguments
        epsilon double {mustBePositive}
    end
```

On initialise ensuite les variables locales et retour de la fonction :

```
n=1;                    % Initialisation de n à 1
i_n_sq = 1/(n^2);       % Calcul de l'inverses du carre de n
y=0;                    % Initialisation du résultat à 0
```

Enfin, tant que l’inverse du carré de n est supérieur à epsilon, on l’ajoute à la variable y, qui sera retournée :

```
while i_n_sq > epsilon  % Tant que 1/n^2 < espsilon
    y = y + i_n_sq;     %   Ajouter 1/n^2 à y
    n = n + 1;          %   Incrementer n
    i_n_sq = 1/n^2;     %   Calculer l'inverses du carre de n
end                     % Fin tant que
```

Nous obtenons donc la fonction suivante, retrouvable dans le fichier .m éponyme :

```
% Calcule la somme des inverses des carres des entiers naturels tant que
%   celui-ci est supérieur à l'argument epsilon passé en paramètre
function y = exercice3(epsilon)
    arguments
        epsilon double {mustBePositive}
    end
    n=1;                    % Initialisation de n à 1
    i_n_sq = 1/(n^2);       % Calcul de l'inverses du carre de n
    y=0;                    % Initialisation du résultat à 0

    while i_n_sq > epsilon  % Tant que 1/n^2 < espsilon
        y = y + i_n_sq;     %   Ajouter 1/n^2 à y
        n = n + 1;          %   Incrementer n
        i_n_sq = 1/n^2;     %   Calculer l'inverses du carre de n
    end                     % Fin tant que
end
```

Pour la démonstration, nous générons un vecteur espacé linéairement sur l’interval $[\varepsilon; 1]$, où ε le plus petit double positif possible, et attribuons à un second vecteur les valeurs de retour des éléments du premier, avant d’afficher le résultat sous forme de graphique :

```
% demonstration de l'exercice 3
X = linspace(eps('double'),1);
Y = arrayfun(@(x) exercice3(x),X);
figure;
plot(X,Y)
title("Démonstration de l'exercice 3")
grid
```

On obtient alors la représentation graphique suivante :