'

Polycopié de Matlab/Octave

G. Dauphin et C. Kulcsar

Table des matières

Le document présente un certain nombre d'explications, d'instructions, de questions et d'exercices. Ces instructions sont à exécuter de manière à les comprendre. Les questions sont des applications directes des instructions et ne devraient pas poser de problèmes. Les exercices peuvent être réalisés aussi avec un petit nombre d'instructions. Ces instructions sont a testés d'abord en ligne. Pour les exercices, il est important de sauvegarder les instructions que vous proposez dans un fichier texte afin de les montrer quand un enseignant passe vous voir. Le fichier texte peut être un script écrit avec l'éditeur Matlab/Octave.

L'annexe de ce document présente des idées permettant d'idées pour comprendre pourquoi une instruction ou un programme ne

fonctionne pas.

1 Opérations sur les nombres et les vecteurs

1.1 Utilisation de la ligne de commande et aide en ligne

Matlab/Octave est un logiciel qui interprète les commandes. Il est donc possible soit de les entrer successivement dans la fenêtre de commande.

```
disp('Bonjour'),
```

Cette commande `disp` permet d'afficher un message. D'une façon générale il est conseillé de tester les commandes dans cette fenêtre avant de les utiliser dans un programme.

Il est possible de rappeler cette commande en tapant le début de la commande et en appuyant sur la flèche supérieure.

La fonction d'aide mentionnée tout au long de ce document est le `help`, qui renseigne à la fois sur les différentes rubriques (qui correspondent à différents répertoires) et sur les fonctions elles-mêmes.

Comme dans l'exemple ci-dessus, les fonctions s'utilisent généralement avec des parenthèses.

En annexe E, (p. 100) figurent des exemples de messages d'erreurs et leur possible signification.

1.2 Opérations de bases

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

```
A =
1 2 3 4
5 6 7 8
```

Notez qu'il est important de bien mettre les espaces entre les chiffres sinon cette expression sera interprétée comme un vecteur à une colonne et deux lignes.

Vous pouvez remarquer qu'alors il est apparu sur la droite de la fenêtre principale de Matlab appelée *Command Window*, sous l'onglet *Workspace* une nouvelle variable notée A avec sa valeur. Sous Octave, cette apparition a lieu à gauche dans une fenêtre appelée *Espace de Travail*. Le terme *Workspace*, espace de travail, est l'ensemble des données disponible en mémoire. Cette matrice A est considérée comme un *double*. *double* fait référence au type défini dans le langage C qui est une indication de la place mémoire occupée par la variable dans la mémoire.

```
     class(A)
ans=
double
```

Remarquez que pour les fonctions comme `class`, `help` etc.. et pour les variables comme `A`, Matlab/Octave fait une distinction entre les minuscules et les majuscules. Ainsi
Matlab:

```
     a
??? Undefined function or variable 'a'.
```

Octave:

```
  a
error: 'a' undefined near line 1 column 1
```

Ce message indique qu'il ne reconnaît pas la variable `a` bien que la variable `A` soit déjà défini. Il est d'usage d'utiliser des minuscules pour les noms de fonctions et pour les variables scalaires, en revanche les matrices sont en générales des majuscules.

Calculez les dimensions d'une matrice :

```
     size(A)
ans =
2 4
```

Cela signifie qu'il y a 2 lignes et 4 colonnes.

Remarquez que quand on met une quantité à calculer sans préciser une variable, il annonce dans l'affichage que la variable **ans** a pris la valeur qui vient d'être calculée. Et en effet on peut faire des calculs à partir de cette variable.

```
     size(ans)
ans =
     1     2
```

Ici la variable **ans** qui a été définie à l'instruction précédente était une matrice contenant [1, 2] autrement dit une matrice à une ligne et deux colonnes.

Calculez la longueur d'un vecteur (remarquez que le t est avant le h) :

```
     u=[1 2 3 4];
     length(u)
ans =
4
```

Avec Matlab, il est possible de copier et coller un ensemble d'instruction, puis de les exécuter, tandis qu'avec Octave, chaque instruction est exécutée avant que la suivante soit collée.

A priori il faut éviter d'utiliser cette instruction `length` sur une matrice et si on le fait le résultat est la plus grande des dimensions de `A`.

On peut aussi utiliser l'instruction `size` d'une autre façon

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

`size(A,1)` donne le nombre de ligne de la matrice, (le changement de ligne est considéré comme un déplacement le long de la *première* dimension). `size(A,2)` donne le nombre de colonnes de la matrice, (le changement de colonnes est considéré comme un déplacement le long de la *deuxième* dimension).

Accédez à un élément :

```
     A(2,3)
ans =
7
```

La notation correspond à celle des composantes d'une matrices. En l'occurence il s'agit de la valeur correspondant à la deuxième ligne et la troisième colonne.

Notez que la virgule a été utilisée ici de deux façons, dans le premier cas (pour l'instruction `size`) comme une séparation entre

deux arguments transmis à la fonction `size` et dans le deuxième cas comme une séparation entre l'indice correspondant au numéro de ligne et l'indice correspondant au numéro de colonne. Les décimales sont séparées de la partie entière par un "." et non une ",".

L'apostrophe correspond à la transposée d'une matrice [1].

```
     A'
ans =
1 5
2 6
3 7
4 8
```

Multiplication matricielle : il faut respecter les règles mathématiques habituelles.

```
     v=[1 3 5 7 11 13 17 19 23 29]
v =
1 3 5 7 11 13 17 19 23 29
     p=v*v
```

Matlab

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

Octave

```
error: operator *: nonconformant arguments (op1
```

1. La transposée d'une matrice est notée A^T (voir `https://fr.wikipedia.org/wiki/Matrice_transpos%C3%A9e`). Des détails sur le fonctionnement de la commande sont disponibles avec `help punct`.

```
     p=v*v'
ans =
2394
     B=A*A
```

Matlab:

```
??? Error using ==> *
Inner matrix dimensions must agree.
```

Octave:

```
error: operator *: nonconformant arguments (op1
```

```
     P=A*A'
P =
30 70
70 174
     B=A'*A
B =
26 32 38 44
32 40 48 56
38 48 58 68
44 56 68 80
```

On peut concaténer des matrices facilement :

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

```
      C=[C C]
C =
1  2  3  4  1  2  3  4
5  6  7  8  5  6  7  8
9  10  11  12  9  10  11  12
```

ou modifier un élément :

```
      C(2,7)=-10
C =

 1      2      3      4      1      2      3      4
      5      6      7      8      5      6    -10      8
      9     10     11     12      9     10     11     12
```

Cherchez à réduire la taille de la fenêtre de commande Matlab/Octave[2] de telle sorte que la même commande

```
 C(2,7)=-10
```

produise l'affichage suivant :

```
C =


  Columns 1 through 4

     1      2      3      4
     5      6      7      8
     9     10     11     12
```

2. Pour réduire la taille de cette fenêtre de commande placez la souris sur le bord droit de cette fenêtre de commande et déplacez la souris vers l'intérieur en gardant appuyée la touche droite de la souris.

```
  Columns 5 through 8

     1     2     3     4
     5     6   -10     8
     9    10    11    12
```

Les lignes `Columns 1 through 4` et `Columns 5 through 8` sont juste là pour indiquer les colonnes de la matrices qui sont affichées.

Remarquez que même avec une fenêtre Matlab/Octave assez large, on a un affichage modifié

```
     C(2,7)=sqrt(C(1,3))
C =
Columns 1 through 7
1.0000 2.0000 3.0000 4.0000 1.0000 2.0000 3.0000
5.0000 6.0000 7.0000 8.0000 5.0000 6.0000 1.7321
9.0000 10.0000 11.0000 12.0000 9.0000 10.0000 11.0000
Column 8
4.0000
8.0000
12.0000
```

Dans cet exemple, l'instruction la composante trouvée à la première ligne troisième colonne dans ce que valait la matrice C avant sa modification, cette valeur est trois. La racine carrée de cette valeur est ensuite placée dans la matrice C à la deuxième ligne et la septième colonne.

On peut aussi inclure des formules :

```
>>  a=4
```

```
>>  b=1
>>  y=[-3.5 sqrt(B(4,1)) exp(0.5^2/2)*(a+b)]
y =
-3.5000 6.6332 5.6657
```

On a ainsi obtenu un vecteur composé d'une ligne et trois colonnes.

On peut utiliser le séparateur "," au lieu d'un espace.

```
>> y=[-3.5, sqrt(B(4,1)), exp(0.5^2/2)*(a+b)]
```

La signification détaillée de cette instruction peut être obtenue en examinant successivement les différents termes (0.5^2 signifie 0.5×0.5) :

```
>> sqrt(B(4,1))
>> 0.5^2
>> exp(0.5^2/2)
>> (a+b)
>> exp(0.5^2/2)*(a+b)
```

On peut écrire une longue ligne sur plusieurs lignes en mettant à la fin de chaque petite ligne "..."

```
 >>    y=[-3.5, sqrt(B(4,1)), exp(0.5^2/2)*...
 >>    (a+b)]
y =
-3.5000 6.6332  5.6657
```

Les nombres décimaux sont indiqués avec un point et non une virgule comme on le fait en français. Ainsi $3,5$ est interprété par Matlab/Octave comme une succession de deux chiffres trois et cinq et non comme la moitié de sept.

Attention : la taille de y n'est pas fixée définitivement. Affectez une valeur à une coordonnée non encore utilisée fait ajuster automatiquement la taille d'un vecteur ou d'une matrice en rajoutant autant de zéros qu'il est nécessaire.

```
      y(7)=3.5
y =
-3.5000 6.6332  5.6657 0 0 0 3.5000
```

1.3 Opérations élément par élément

Il est possible et souvent recommandé de faire des calculs vectorisés, c'est-à-dire en traitant tous les éléments d'un vecteur ou d'une matrice en même temps. Pour faire des calculs en une seule instruction sur chaque élément d'une matrice ou d'un vecteur, on met un point avant l'opérateur :

- A.^n : élève chaque élément de A à la puissance n
- A.^(1/2) ou `sqrt(A)` : calcule la racine carrée de chaque élément de A

D'autres calculs peuvent être effectués de cette façon. Par exemple
Matlab:

```
>> X=exp(A).*B(1:2,1:4)
ans =
1.0e+005 *
0.0007 0.0024 0.0076 0.0240
0.0475 0.1614 0.5264 1.6693
```

Octave:

```
>> X=exp(A).*B(1:2,1:4)
X =
```

```
Columns 1 and 2:

      70.67533       236.44980
    4749.22109     16137.15174

Columns 3 and 4:

     763.25040      2402.31860
   52638.39160    166933.64727
```

La notation `1.0e+005` signifie en fait 1.0×10^5, voir [3]. L'affichage ici signifie que le premier élément indiqué est multiplié par l'ensemble des éléments qui suit. On peut afficher les valeurs des composantes de X une par une avec

```
>> X(1)
ans =  70.675
```

On peut aussi afficher l'ensemble des valeurs de X sans mettre en commun un multiplicateur avec

```
>> num2str(X)
```

Cette fonction sert en pratique à convertir une valeur, un vecteur ou une matrice en une chaîne de caractère. Utilisez la fonction `class` définie en page p. 4 pour vérifier qu'il s'agit bien d'une chaîne de caractère. [4]

3. C'est une notation qui existe en C, il est à noter que `10e5` signifie en fait 10×10^5, c'est-à-dire 10^6.

4. Il est en apparence possible de faire des calculs numériques sur l'objet retourné par la fonction `num2str` parce que Matlab/Octave interprète cette chaîne de caractère comme un vecteur d'entier où chaque lettre est remplacé par l'entier correspondant au code ASCII de chaque lettre. Naturellement ces calculs n'ont aucun sens.

Dans tout ce qui suit, vous utiliserez mettrez au début de chaque session la commande suivante :

```
format shortg
```

de cette façon le vecteur X est maintenant affiché ainsi :

```
>> X =
       70.675        236.45        763.25         2402.3
       4749.2         16137         52638   1.6693e+005
```

Pour revenir au format par défaut, c'est la commande :

```
>> format short
```

Pour multiplier chaque élément de F avec chaque élément de G, on met un point avant l'opérateur de multiplication :

```
>> F=[9 2; 1 6; 2 5]
F =
9 2
1 6
2 5
>> G=[1 4; 7 1; 5 6]
G =
1 4
7 1
5 6
>> F.*G
ans =
9 8
7 6
10 30
```

De même, on divise chaque élément de F par chaque élément de G de même position en faisant :

```
>> F./G
ans =
9.0000 0.5000
0.1429 6.0000
0.4000 0.8333
```

D'autres exemples :

```
>> A./A.^2
ans =
1.0000 0.5000 0.3333 0.2500
0.2000 0.1667 0.1429 0.1250
```

Matlab:

```
>> exp((1-v).^2)
ans =
1.0e+210 *
Columns 1 through 7
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
Columns 7 through 10
0.0000 1.5795 Inf
```

Octave:

```
ans =
 Columns 1 through 6:
    1.0000e+00    5.4598e+01    8.8861e+06    4
 Columns 7 through 10:
   1.5114e+111   5.1453e+140   1.5795e+210
```

```
>> tan((1-v./v.^3).^2)
ans =
Columns 1 through 7
0 1.0095 1.3176 1.4271 1.5024 1.5177 1.5340
Columns 8 through 10
1.5386 1.5445 1.5493
>> 1./(log(B)-1.5).^(-0.5)
ans =
1.3259 1.4020 1.4620 1.5114
1.4020 1.4795 1.5399 1.5891
1.4620 1.5399 1.6001 1.6491
1.5114 1.5891 1.6491 1.6977
```

Remarquez que `log` désigne en fait le logarithme à base e que l'on note en mathématiques ln :

```
>> e=exp(1)
e=
2.7183
>> l=log(e)
l=
1
```

Il est à distinguer de $\log_{10}$ qui est le logarithme à base 10

```
>> l1=log10(10)
l1=
1
```

1.4 L'opérateur " :" d'énumération

L'opérateur d'énumération s'utilise avec la syntaxe suivante début : incrément : fin où l'incrément n'a pas à être précisé quand il vaut 1.

On peut faire la liste des nombres multiples de 3 inférieurs à 30, voir [5], [6].

```
>> nb=0:3:30
```

L'ajustement de l'incrément permet de faire une suite décroissante :

```
>> nbRev=30:-1:0
```

L'ordre dans lequel Matlab/Octave lit l'expression a son importance

```
     vect1=1./(0.1:0.1:0.5)
vect1 =
   10.0000    5.0000    3.3333    2.5000    2.0000
```

Pour comprendre cette instruction, on peut tester

```
     0.1:0.1:0.5
ans =
    0.1000    0.2000    0.3000    0.4000    0.5000
```

et observer qu'il appliqué l'inverse à chacune des composantes du vecteur.

Les parenthèses ici sont importantes car l'instruction suivante Matlab:

5. Quand `u` est un vecteur, il est important de ne pas utiliser `u` de la façon suivante : `u:2:25`.

6. En C on aurait le même résultat avec `int i,j=0; for(i=0;i<=30;i=i+3) tab[j++]=i;`

```
    vect2=1./0.1:0.1:0.5
vect2 =
   Empty matrix: 1-by-0
```

Octave:

```
>> vect2=1./0.1:0.1:0.5
vect2 = [](1x0)
```

est en fait interprétée de la façon suivante

```
>> vect2=(1./0.1):0.1:0.5
```

L'indication **Empty matrix** n'est pas une erreur, il signifie seulement que l'objet retourné ne contient aucune valeur. [7]

Il permet également d'accéder à tout ou partie des éléments d'une ligne, d'une colonne ou d'une matrice entière.

```
>> B
B =
26 32 38 44
32 40 48 56
38 48 58 68
44 56 68 80
```

Pour avoir toute la 3ème ligne de B :

```
>> B(3,:)
ans =
38 48 58 68
```

ou avoir toute la 2ème colonne de B :

7. Il ne contient ici aucune valeur parce que 1./0.1 est supérieure à 0.5.

```
>> B(:,2)
ans =
32
40
48
56
```

ou avoir toute la matrice B sous la forme d'un seul vecteur (les colonnes sont mises les unes sous les autres) :

```
>> B(:)
ans =
26
32
38
44
32
40
48
56
38
48
58
68
44
56
68
80
```

A la description ci-dessus correspond la numérotation des éléments d'une matrice de 1 à $M \times N$, colonne après colonne. Ainsi, si l'on

tape

```
>> B(15)
ans =
68
```

le résultat obtenu est le 15ème élément de B lorsque la numérotation commence à 1 et continue le long des colonnes jusqu'à 16 puisque B est de taille 4×4. Cet élément est aussi adressé par

```
>> B(4,3)
ans =
68
```

Autres exemples d'utilisation de " :"
Soit Q une matrice nulle de dimensions 3x4,

```
>> Q=zeros(3,4)
Q =
0 0 0 0
0 0 0 0
0 0 0 0
```

on peut modifier une ligne en une seule instruction

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

ou une colonne

```
>> Q(:,2)=(1:3)'
Q =
0 1 0 0
0 2 0 0
1 3 3 4
```

Et en effet la deuxième colonne de Q qui valait

$$\begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

vaut maintenant

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Remarquez que si on retire les parenthèses on change l'effet de l'apostrophe et le vecteur suivant devient un vecteur ligne alors qu'il était un vecteur colonne précédemment.

```
1:3'
```

Le sens des fonctions est en général modifié quand on modifie le nombre d'arguments transmis. Ainsi la fonction `size` avec deux paramètres signifie le nombre de colonnes ou de lignes du premier argument en fonction de la valeur contenue dans le deuxième argument.

```
>> X=ones(2,4);
>> size(X,1),
>> size(X,2),
```

Une autre façon de faire de l'affectation consiste à définir d'abord la taille de la matrice puis à la remplir comme s'il s'agissait d'un vecteur.

```
>> A=zeros(3);
>> A(:)=1:size(A,1)*size(A,2);
>> A
A =
     1     4     7
     2     5     8
     3     6     9
```

En pratique l'opérateur d'énumération permet souvent de remplacer un programme que l'on aurait écrit en C avec l'instruction `for`. Ainsi si l'on veut calculer la somme des nombres inférieurs ou égaux à 10, on aurait écrit en C, (il ne sert à rien ici de recopier les lignes qui suivent, Matlab/Octave n'interprète pas le C).

```
int val=0;
for(int i=0;i<9;i++) val=val+i;
printf("La somme est %d\n",val);
```

Ce même programme s'écrit en Matlab/Octave

```
>> sum(0:10)
```

La commande `sum` signifie somme des éléments d'un vecteur Par exemple la somme des trois nombres 5,1,3 peut s'implémenter de la façon suivante

```
>> sum([5 1 3]),
```

La fonction **sum** et l'opérateur d'énumération peuvent être utilisés pour calculer des sommes de séries. Par exemple une approximation de la somme $\Sigma_{n\geq 1}\frac{1}{n^2}$ peut être ainsi calculée :

```
>> somme=sum(1./((1:10000).^2))
somme=
1.6448
```

La fonction **sum** appliquée à une matrice permet d'obtenir la somme des colonnes, la somme des lignes ou la somme de tous les éléments

```
>> E1=[1 2 3; 2 3 4];
>> sum(E1,1),
ans =
     3     5     7
>> sum(E1,2),
ans =
     6
     9
>> sum(sum(E1)),
ans =
    15
```

En pratique la fonction **linspace** permet aussi de définir un vecteur composé de valeurs également réparties entre deux bornes.

La commande **end** permet de spécifier le dernier élément, cela peut vouloir dire le dernier élément d'un vecteur, la dernierè colonne, la dernière ligne ou le dernier élément d'une matrice

```
>> u=1:12
u =
```

```
     1     2     3     4     5     6     7     8     9
>> u(end)
ans =
    12
```

A la place de `u(end)`, on aurait pu écrire

```
>> u(length(u))
ans =
    12
```

La commande **`end`** permet par exemple de sélectionner les deux derniers éléments d'une liste de valeurs :

```
>> u(end-2:end)
ans =
   10 11 12
```

La commande **`reshape`** permet de modifier la disposition des éléments dans la matrice :

```
>>  A=reshape(1:12,3,4)
A =

     1     4     7    10
     2     5     8    11
     3     6     9    12
```

Dans cette dernière expression, à `A` est affectée une nouvelle matrice obtenue en redisposant les chiffres de 1 à 12 de l'expression `1:12` sous la forme d'une matrice de trois lignes et quatre colonnes. Remarquez que cette nouvelle disposition les chiffres sont disposés successivement sur les différentes colonnes.

```
     A(:,end)
ans =
    10
    11
    12
     A(end,:)
 3     6     9    12
     A(end,end)
ans =
    12
     A(end)
ans =
    12
```

Conformément à la convention en C, il est possible d'indiquer l'exposant d'un nombre avec la lettre 'e'
Matlab:

```
>> 3e-5
ans =
  3.0000e-005
```

Octave:

```
>> 3e-5
ans =  0.000030000
```

Cette valeur signifie 3×10^{-5}.

Notez que
Matlab:

```
>> 10e-5
ans = 1.0000e-004
```

Octave:

```
>> 10e-5
ans = 0.00010000
```

signifie $10 \times 10^{-5} = 10^{-4}$ La commande **ans** désigne la dernière valeur entrée.

1.5 Priorités des opérations

Les opérations sont interprétées d'abord suivant les parenthèses puis suivant les règles de priorités et dans le cas où les niveaux de priorités sont identiques alors l'interprétation se fait de gauche à droite.

Remarquez que pour calculer $\frac{a}{bc}$, il est possible, (comme dans une calculatrice), d'écrire

```
      10/2/5
ans=1
```

plutôt que

```
      10/(2*5)
ans=1
```

L'exposant est prioritaire sur la multiplication qui est prioritaire sur l'addition

```
      1+3*2^4
ans=49
      1+(3*(2^4))
ans=49
```

L'application d'une fonction est prioritaire par rapport à une opération

```
     sum(1:3).^2
ans=36
     sum((1:3).^2)
ans=14
```

Lorsque les règles de priorités ne conviennent pas, il est possible en rajoutant des parenthèses de s'assurer que le calcul se fasse dans l'ordre souhaité.

1.6 Des outils pour l'affichage

Il existe des fonctions spécifiques permettant l'affichage à l'écran avec ou sans contrôle du format d'affichage. La fonction `disp` permet d'afficher le contenu d'une variable non vide.

```
>> disp(A)
    1    4    7   10
    2    5    8   11
    3    6    9   12
     m=max(y);
     disp(m)
6.6332
```

Le format d'affichage des variables peut être modifié via `format`. Pour afficher par exemple 15 chiffres après la virgule, on exécute l'instruction suivante :

```
     format long
```

qui change l'affichage pour la suite de la session.
Matlab:

```
     cos(pi/2)
ans =
6.123233995736766e-17
     disp(m)
6.633249580710800
```

Octave:

```
     cos(pi/2)
ans =
6.123031769111886e-17
     disp(m)
6.633249580710800
```

On revient au format d'affichage classique avec

```
     format short
```

On peut l'utiliser pour afficher à la fois une phrase et un résultat, mais il faut alors convertir le nombre en caractères en utilisant `num2str`.

```
     disp(['la valeur c''est ', num2str(m)])
la valeur c'est 6.6332
```

Il est bien sûr important de bien mettre `num2str`, sinon on risque d'avoir :

```
     disp(['la valeur c''est ',54])
la valeur c'est 6
```

En effet 54 est le numéro du caractère `6` dans la table ASCII.

L'argument de **`disp`** est ici une chaîne de caractères :

```
['la valeur c''est ', num2str(m)]
```

c'est-à-dire un vecteur composé de caractères. Les éléments de la chaîne de caractère passée en argument de disp doivent être compatibles du point de vue dimensions : on ne peut mettre côte à côte que des chaînes de caractères ayant le même nombre de lignes.
Matlab:

```
>> disp(['la valeur c''est ', num2str(A)])
??? Error using ==> horzcat
All matrices on a row in the bracketed expression must
same number of rows.
```

Octave:

```
>> disp(['la valeur c''est ', num2str(A)])
error: horizontal dimensions mismatch (1x16 vs 2x10)
```

On peut aussi utiliser **`num2str`** pour augmenter le nombre de chiffre significatif

```
    disp(['la valeur c''est ', num2str(sqrt(2),5)])
la valeur c'est 1.4142
```

Exercice 1 *On cherche des commandes courtes similaires au programme Matlab/Octave qui permet de sommer des entiers page 23, donc en utilisant l'opérateur d'énumération plutôt*

que des boucles `for`*. L'idée pour former des vecteurs contenant des nombres paires ou impaires, c'est de mettre 2 pour le pas d'énumération (la quantité ajoutée entre deux valeurs énumérées successsives). Cet exercice est un peu difficile, si vous êtes bloqués sur une question plus de 10 minutes, regardez la réponse dans F, (p. 102).*

1. *Donnez la commande permettant de calculer un vecteur ligne noté* `u`*. Ce vecteur contient tous les nombres entiers pairs de 1 à 30, dont le résultat est :*

```
u =
  Columns 1 through 6
     2     4     6     8    10    12
  Columns 7 through 12
    14    16    18    20    22    24
  Columns 13 through 15
    26    28    30
```

2. *Donnez la commande permettant de calculer un vecteur ligne* `v` *contenant un élément sur deux de* `u`*, ces éléments sont* $u(1), u(3), \ldots$ *On cherche à obtenir ceci :*

```
v =
     2     6    10    14    18    22    26    30
```

   Pour trouvez cette commande vous pouvez d'abord chercher à synthétiser la liste des indices de u qui nous intéresse en utilisant la longueur du vecteur u (qui est donné par `length(u)`*), puis en appliquant le vecteur u à ces données et en utilisant* `end` *au lieu de* `length`*, (voir p. 25).*

3. *w = vecteur ligne dont les coordonnées paires forment le vecteur v et les coordonnées impaires valent le carré des éléments de v. Voici le résultat que l'on cherche à obtenir*

```
w =
  Columns 1 through 14
     4     2    36     6   100    10   196    14
  Columns 15 through 16
   900    30
```

Pour cela vous pourrez d'abord un vecteur `w` *d'une ligne et 16 colonnes des zéros au moyen de la fonction* `zeros`*. Puis vous pourrez placer l'opérateur d'énumération à l'intérieur des parenthèses de* `w`*. Vous pourrez faire d'abord l'affectation des composantes d'indices impaires puis l'affectation des composantes d'indices paires.*

4. *Calculez le vecteur* `a` *dont les composantes sont définies par*

$$a = \pi - (x - (x - \pi))$$

pour x prenant les valeurs 10^0, 10^1, 10^2,... ,10^{19}, 10^{20}, (on pourra considérer le vecteur `10.^(0:20)`*). π s'obtient en Matlab/Octave avec la variable prédéfinie* `pi`*.*

5. *L'affichage de a est un particulier, parce que sous Matlab, l'exposant le plus élevé est placé tout au début et est mis en facteur de toutes les autres composantes. Pour avoir un affichage plus classique, vous pouvez utiliser* `format shortg`*. Attention* `format` *ne s'utilise qu'avec*

un seul argument comme `short` *ou* `shortg` *sans préciser une valeur à afficher. Cette commande modifie l'affichage de toutes les valeurs qui suivent. Commentez les valeurs effectivement trouvées pour a.*

Remarquez que normalement toutes les composantes de a devraient être nulles pour toutes les composantes ? Comment expliquez que ce n'est pas ce que l'on observe. Pour comprendre ce phénomène, vous pouvez tester les instructions

```
>> (1+10^20)-10^20
>> (0.000000045+10^10)-10^10
```

1.7 Chaînes de caractères

Nous avons vu que toute variable dans Matlab/Octave est une matrice, un scalaire n'étant rien d'autre qu'une matrice 1x1. Dans ce contexte, une chaîne de caractères est un vecteur dont chaque composante est un caractère de la chaîne. Un grand nombre de fonctions sont disponibles (`help strfun`) pour la manipulation des chaînes de caractères. Nous nous intéresserons en particulier aux fonctions de conversion de nombres en chaînes de caractères et vice versa (fonctions utiles pour l'affichage), ainsi qu'aux fonctions d'évaluation d'une chaîne de caractères (`eval` et `feval`).

```
      message='Initiation a Matlab'
message =
Initiation a Matlab
```

```
      size(message)
ans =
1 19
```

Ainsi message est aussi vu comme une matrice à une ligne et 19 colonnes.

Matlab permet de gérer les caractères accentués copiés qu'ils soient codés en code ANSI ou en UTF8, en revanche ils provoquent une erreur ou ne sont pas pris en compte en Octave.

Concaténez deux chaînes de caractères se fait simplement en rajoutant des éléments au vecteur.

```
      message=[message, ' pour les debutants']
message =
Initiation à Matlab pour les debutants
      class(message)
ans =
char
```

2 Manipulations diverses sur les éléments des vecteurs et matrices

2.1 Matrices particulières, séquences pseudo aléatoires

On peut obtenir différents types de matrices en utilisant des fonctions Matlab qui créent des matrices de structure particulière (help elmat). Il suffit de préciser la taille de la matrice. Par exemple, la fonction zero produit une matrice nulle, ones et eye produisent respectivement une matrice dont tous les éléments sont à 1 et une

matrice dont les éléments de la diagonale principale valent 1 (matrice identité dans le cas des matrices carrées).

```
     zeros(5,3)
ans =
0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
     ones(1,8)
ans =
1 1 1 1 1 1 1 1
     eye(10,5)
ans =
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Sont disponibles également les matrices de type Hankel, Toeplitz, ... Dans le cas de matrices de grandes dimensions creuses, c'est-à-dire pour les matrices ne contenant que très peu de composantes non-nulles, on peut (doit) utiliser une structure de matrice creuse (sparse)

pour des raisons de place mémoire. La matrice est alors stockée en mémoire et affichée d'une façon différente. Au lieu de préciser toutes les valeurs, ce nouveau format consiste à lister uniquement les valeurs non-nulles et les indices associées à ces valeurs non-nulles. Tout un ensemble de fonctions pour matrices creuses est disponible sous Matlab (**help sparfun**). Un exemple simple :
Matlab:

```
     S=sparse(2,2) % Création d'une matrice 2x2 ne
                   % contenant que des zéros
     S =
All zero sparse: 2-by-2
>> S(1,2)=1; S % On affecte à l'élément (1,2) la
                % valeur 1 et on affiche S
     S =
(1,2) 1
```

Octave:

```
>> S=sparse(2,2)
S =
Compressed Column Sparse (rows = 2, cols = 2, r
>> S(1,2)=1; S % On affecte à l'élément (1,2) l
                % valeur 1 et on affiche S
S =
Compressed Column Sparse (rows = 2, cols = 2, r
  (1, 2) ->  1
```

2.2 Manipulations des matrices

Retournement vertical ou horizontal, rotation de 90 degrés, remise en forme de matrices ou vecteurs se font en un clin d'oeil grâce

aux fonctions flipud : flip up/down fliplr : flip left/right rot90 : ... reshape : voir le dico !

```
      x=[1 9 6 5 3 6 4 0 1 2 5 9]
x =
1 9 6 5 3 6 4 0 1 2 5 9
```

Pour former une matrice à partir d'un vecteur, on utilise reshape qui permet de passer de n'importe quelle matrice de dimensions $M \times N$ à une matrice de dimensions $P \times Q$ pourvu que $M \times N = P \times Q$. La réorganisation des éléments se fait colonne par colonne.

```
      R=reshape(x,3,4)
R =
1 5 4 2
9 3 0 5
6 6 1 9
```

On peut créer des matrices diagonales ou extraire des diagonales avec la même fonction `diag`, extraire la partie triangulaire supérieure (triu) ou inférieure (tril), voir l'aide en ligne.

```
      d1=diag(B)
d1 =
26
40
58
80
```

Matlab:

```
>> diag(d1)
ans =
```

```
    26     0     0     0
     0    40     0     0
     0     0    58     0
     0     0     0    80
```

Octave:

```
 diag(d1)
ans =
Diagonal Matrix
   26    0    0    0
    0   40    0    0
    0    0   58    0
    0    0    0   80
```

La fonction **repmat** permet de dupliquer une matrice ou un vecteur :

```
     D=repmat(A,3,2)
D =
1 2 3 4 1 2 3 4
5 6 7 8 5 6 7 8
1 2 3 4 1 2 3 4
5 6 7 8 5 6 7 8
1 2 3 4 1 2 3 4
5 6 7 8 5 6 7 8
```

Il existe une fonction pour trier les valeurs d'un vecteurs (ou les caractères).

```
     u=[1 6 3 4];
     sort(u)
ans =
     1     3     4     6
```

Pour résumer, on peut dire que pour concaténer les éléments, on se sert des virgules ou des espaces pour associer horizontalement, un point virgule pour associer verticalement et dans les deux cas un crochet ouvrant et fermant autour. On se sert des parenthèses pour transmettre une valeur, un vecteur ou une matrice à une fonction ou pour accéder à une composante particulière d'un vecteur ou d'une matrice. On se sert des virgules lorsqu'on transmet à une fonction plusieurs valeurs ou lorsqu'il est nécessaire de donner plusieurs coordonnées pour accéder à un élément particulier d'une matrice. On se sert d'un ou plusieurs deux points lorsqu'on veut définir un vecteur ligne au moyen d'une énumération. La partie décimale d'un nombre est séparé dans l'affichage et la saisie par un point et non une virgule.

Exercice 2 *1. M_1 : dimensions 4×7, les éléments lus en parcourant successivement les lignes correspondent aux 28 premiers nombres impairs pris par ordre croissant. La solution s'obtient avec l'utilisation de* `reshape` *et d'une apostrophe. Le résultat doit être le suivant :*

```
M1 =
     1     3     5     7     9    11    13
    15    17    19    21    23    25    27
    29    31    33    35    37    39    41
    43    45    47    49    51    53    55
```

2. M_2 : dimensions 4×7, les éléments lus en parcourant successivement les lignes correspondent aux inverses des 28 premiers nombres impairs pris par ordre croissant, La solution s'obtient en utilisant de

`./`, `reshape`, *d'une apostrophe et d'une énumération* `(2*28-1):-2:1`.

```
M2 =
  Columns 1 through 5
     0.018182     0.018868     0.019608      0.020
      0.02439     0.025641     0.027027      0.028
     0.037037         0.04     0.043478      0.047
     0.076923     0.090909      0.11111       0.14
  Columns 6 through 7
     0.022222     0.023256
     0.032258     0.034483
     0.058824     0.066667
      0.33333            1
```

3. *Calculez* M_3 *qui est une matrice formée des matrices* M_1 *et* M_2 *disposées l'une sous l'autre.*
4. *Calculez le vecteur* z *qui est un vecteur ligne contenant tous les éléments de* M_3 *lus en ligne*
5. *Calculez, sans utiliser* `reshape`*, la matrice* M *de dimensions* 4×7*, dont les termes valent*

$$M(i,j) = \frac{\log_{10}\left(M_1(i,j)\right)}{M_1(i,j)}$$

6. *Quelles sont les dimensions de* $P = M_1 M^T$ *? P est la matrice* M_1 *multipliée matriciellement par la transposée de* M*. Calculez cette matrice.*
7. *Calculez le vecteur colonne* c_3 *qui est la troisième colonne de P, voir*[8]

8. *Calculez le vecteur ligne l_2 qui est la deuxième ligne de P.*
9. *Calculez le vecteur colonne v formés des éléments de c_3 et l_2 à la suite.*

2.3 Générer des nombres aléatoirement

La fonction `rand` permet d'obtenir une matrice dont les composantes suivent une loi uniforme sur l'intervalle $[0, 1]$.

```
rand(2,3)
```

Pour changer l'intervalle caractérisant la loi uniforme, il suffit d'appliquer une multiplication et une addition à la matrice obtenue.

La fonction `randn` permet d'obtenir une matrice dont les composantes suivent une loi gaussienne centré et d'écart-type 1.

```
randn(2,3)
```

Pour changer l'espérance et l'écart-type, il suffit respectivement d'appliquer une addition et une multiplication à la matrice obtenue.

Le générateur de nombre aléatoire est en réalité un algorithme déterministe qui est rendu aléatoire avec l'instruction
Matlab:

```
>> rng('shuffle')
```

Octave:

```
>> rand ("seed", "reset")
```

Cette instruction est presque l'équivalent en C de `srand(time(NULL));` Attention cette instruction ne doit être appelée à chaque fois que l'on fait un tirage aléatoire, il s'agit plutôt d'une fonction à appeler en début de séance de TP. La section A (p. 97) donne une expérimentation illustrant l'utilité de la fonction `rng('shuffle')`.

2.4 Un peu d'algèbre linéaire

Outre les opérations vues dans les paragraphes précédents, des opérations d'algèbre linéaire sont disponibles sous Matlab (help matfun). Par exemple `inv` permet de calculer l'inverse d'une matrice. De même on peut résoudre un système d'équations linéaires avec $X = A\backslash B$ qui correspond à peu près à $X = A^{-1}B$ (et en fait donne la solution de $A * X = B$). Ce procédé utilise la notion de pseudo-inverse.

```
      G=[1 4; 7 1; 5 6];
      G=[G [0 3 9]']
G =
1 4 0
7 1 3
5 6 9
      G\[1:3]'
ans =
0.2239
0.1940
0.0796
```

```
      G*ans
ans =
1.0000
2.0000
3.0000
```

Voici un exemple de problème qui se résout de cette façon, Pierre achète 2 kg de pomme et 3 kg de poire, il paye 18euros. Son ami, lui, a achété 500 g de pomme et 1 kg de poire et il a payé $5,5$ euros. Quel est le prix des pommes et quel est le prix des poires ? Ce problème se met en équation en appelant X le vecteur dont les composantes x_1, x_2 correspondent respectivement au prix des pomme et des poires. Les composantes vérifient le système suivant :

$$\begin{cases} 2x_1 + 3x_2 &= 18 \\ 0.5x_1 + x_2 &= 5.5 \end{cases}$$

Ces équations se mettent sous la forment de l'équation matricielle suivante

$$\begin{bmatrix} 2 & 3 \\ 0.5 & 1 \end{bmatrix} X = \begin{bmatrix} 18 \\ 5.5 \end{bmatrix}$$

Ce problème se résout sous Matlab/Octave

```
>> A=[2 3; 0.5 1]; B=[18;5.5]; X=A\B,
X=
3
4
```

Ainsi le prix des pommes est de 3 euros le kilo et celui des poires est de 4 euros le kilo. [9]

9. Attention à toujours faire la différence entre \ et / et aussi au fait que les vecteurs doivent ici être sur une colonne.

Ce problème peut aussi se résoudre de la façon suivante et c'est peut-être plus simple à se souvenir :

```
>> A=[2 3; 0.5 1]; B=[18;5.5]; X=inv(A)*B,
X=
3
4
```

Ici `inv(A)` est ce qu'en mathématique on note A^{-1}. Le système s'écrit $AX = B$ et se résout en $X = A^{-1}B$.

Exercice 3 (2) *On cherche à résoudre le système*

$$\begin{cases} 2x + 3y &= 5 \\ 3x + 2y &= 5 \end{cases}$$

Pour cela on définit une matrice A qui contient les coefficients devant x et y et un vecteur B qui contient les valeurs à droite du signe $=$ et on utilise l'instruction Matlab inv. Ecrivez les instructions Matlab qui vous ont permis de résoudre le système.

Sont disponibles entre autres le déterminant, la trace, le calcul du polynôme caractéristique, du déterminant, du rang, du conditionnement, les décompositions LU, de Cholesky. Toutes ces notions ont en commun de pouvoir être calculés à partir d'une matrice. Elles sont définis par exemple sur Wikipedia

— `rank` :
`https://fr.wikipedia.org/wiki/Rang_(alg%C3%A8bre_l`

En particulier une matrice de taille $N \times N$ est inversible si et seulement si son rang (`rank`) est égal à N.

- `poly` : `https://fr.wikipedia.org/wiki/Polyn%C3%B4me_caract`
- `roots` : Ce sont les racines du polynôme donné par `poly`.
- `eig` : `https://fr.wikipedia.org/wiki/Valeur_propre,_vecte`

```
>> rank(G) % Rang de G
ans =
3
>> p=poly(G) % Coefficients du polynôme
% caractéristique de G par puissances
% décroissantes
p =
1.0000 -11.0000 -27.0000 201.0000
>> roots(p) % Racines du polynôme dont les
% coefficients sont dans le vecteur p
ans =
11.8469
-4.5642
3.7173
```

Matlab:

```
>> [vectp,valp]=eig(G) % Calcul des vecteurs et valeurs
% propres associés à G
vectp =
0.5777 -0.4760 0.1248
-0.8037 -0.3234 0.3385
```

```
0.1425 0.8178 0.9326
16
valp =
-4.5642 0 0
0 3.7173 0
0 0 11.8469
```

oct

```
>> [vectp,valp]=eig(G)
vectp =
   0.12483   0.57775  -0.47602
   0.33851  -0.80368  -0.32337
   0.93265   0.14253   0.81782
valp =
Diagonal Matrix
   11.8469         0         0
         0   -4.5642         0
         0         0    3.7173
```

Malgré les apparences, les résultats sont en fait identiques entre Matlab et Octave, c'est seulement l'ordre des valeurs propres qui a changé. Noter que l'on retrouve bien sûr `diag(valp)` dans `roots(p)`.

Exercice 4 *Cet exercice porte sur les matrices particulières et manipulations avec les fonctions* `diag, mean, std, rand, randn, reshape, sum, zeros, >`.

1. *Z = matrice de dimensions* 4×5 *dont les éléments pris en colonne sont les nombres allant de 0 à -1,9 par pas de -0,1.*
2. *F = matrice de dimensions* 9×5 *dont la partie haute est la matrice Z précédente et le reste est nul*
3. *D = matrice diagonale* 6×6 *dont les éléments diagonaux valent* 10^{-3}*, voir*[10]
4. *U = matrice de dimensions 3x6 dont les éléments sont des tirages pseudo-aléatoires uniformes sur l'intervalle [0, 1]*
5. *s1 = somme des colonnes de U*
6. *s2 = somme de tous les éléments de U*
7. *b = vecteur ligne de longueur 1000 dont les éléments sont des tirages pseudo-aléatoires gaussiens de moyenne nulle et d'écart-type 2. Des indications sont fournies dans l'annexe C (p. 98).*
8. *m = moyenne empirique de b (faites le calcul avec* `mean` *puis le même calcul avec* `sum`*).*
9. *v = variance empirique de b (faites le calcul avec* `std` *puis le même calcul avec* `sum`*). Remarquez que Matlab/Octave estime l'écart-type en divisant l'expression obtenue par* $n-1$ *où* $n-1$ *est le nombre d'échantillons considérés.*

La section B (p. 98) donne des indications pour répondre aux deux dernières questions.

3 Espace de travail

3.1 Sauvegarder des variables

Pour sauver des données sous Matlab/Octave, il faut d'abord choisir un répertoire où les mettre. L'espace de travail dispose d'un répertoire courant indiqué par la commande inspirée de Windows

```
dir
```

En Matlab, il est aussi possible d'utiliser la commande inspirée de Linux Matlab:

```
>> ls
```

On peut se déplacer d'un répertoire vers le répertoire parent avec l'instruction suivante

```
cd ..
```

Si `nom_rep` est le nom d'un répertoire contenu dans le répertoire courant il est possible d'aller dans ce répertoire avec l'instruction

```
cd nom__rep
```

Ces deux dernières utilisations de la commande `cd` montrent comment utiliser un chemin relatif.

On peut aussi utiliser un chemin absolu noté `chemin_abs`, qui est formé (sous Windows) d'une lettre pour le disque contenant les données suivi de " :" puis une succession de chemins relatifs depuis le répertoire de racine du disque dur indiqué. Pour trouver le chemin absolu il suffit de sélectionner le chemin absolu tel qu'il peut être visible dans une fenêtre Windows montrant un répertoire et de le

coller dans une fenêtre de commande Matlab/Octave. Lorsqu'il y a un espace dans le nom d'un répertoire, il suffit de mettre une apostrophe avant et après le groupe de mots.

Le répertoire courant est indiqué sur l'environnement dans la fenêtre à gauche sous l'onglet *Current Folder*, il peut aussi être obtenu en ligne de commande avec l'instruction `pwd`.

On peut noter que la commande `cd c:` que le disque dur courant devient *c :* mais cela ne signifie pas que le répertoire courant est `c:`. Pour indiquer que le répertoire courant soit *c :*, il faut taper

```
cd c:\
```

Question 1 *Choisissez un répertoire où vous enregistrez votre travail. Changez le répertoire courant de l'espace du travail en le répertoire de travail puis inversement du répertoire de travail au répertoire initial en utilisant d'une part un chemin absolu puis un ensemble de commandes qui chacune permet de passer un répertoire à l'un de ces répertoires contenu ou d'un répertoire à son répertoire parent.*

On peut sauver tout ou une partie des variables de l'espace de travail (voir p. 5) dans des fichiers de données Matlab (avec l'extension .mat, à rappeler avec Octave). Par exemple, sauvons la variable surface dans le fichier `disque.mat` :
Matlab:

```
a=5; b=2; r=2; surface=pi*r^2;
save disque surface a
```

```
    dir
. .. disque.mat donnees
```

Octave:

```
    a=5; b=2; r=2; surface=pi*r^2;
    save disque.mat surface a
    dir
. .. disque.mat donnees
```

L'instruction save permet de sauver des données sous plusieurs formats ASCII ou binaire. Le format Matlab par défaut est le format binaire, tandis que le format par défaut sous Octave est ASCII. L'enregistrement sous Octave de données sous format binaire lisible depuis Matlab est :
Octave:

```
>> save -mat-binary donnee.mat
```

Matlab:

```
>> save -ascii donnee.mat
```

Cette dernière instruction n'est pas possible pour les données **sparse** et n'est pas lisible depuis Octave. Sous Matlab quand on enregistre des données sous format ASCII, il faut repréciser l'extension.

On peut remarquer ici que la fonction **save** est ici utilisée sans les parenthèses. Cela est en réalité toujours possible pour toutes les fonctions, sauf que cela revient à traiter les arguments comme des chaînes de caractères. Ainsi la commande **save** utilisée est équivalente à
Matlab:

```
>> save('disque','surface','a');
```

Octave:

```
>> save('disque.mat','surface','a');
```

La syntaxe suivante permet de sauver toutes les variables de l'espace de travail dans un fichier Matlab : Matlab:

```
    save nom_fichier
```

Octave:

```
    save nom_fichier.mat
```

En Matlab, si nom_fichier est omis, les variables sont sauvées par défaut dans le fichier matlab.mat. On peut aussi sauver une ou plusieurs variables dans un même fichier Matlab :

```
save nom_fichier variable1 variable2 variableN
```

Il est aussi possible de sauvegarder ces données sur le répertoire de travail noté `rep_trav` sans que le répertoire courant soit `rep_trav`. Ce répertoire de travail doit contenir l'information sur le disque dur contenant ce répertoire. Il suffit pour cela d'adapter la commande précédente

```
save rep_trav\nom_fichier variable1 variable2 variableN
```

Exercice 5 *On reprend la variable* `a` *calculée dans l'exercice 1 (p. 30 ou juste après). Essayez avec la commande* `save` *de faire les actions suivantes :*

1. *Sauvez la variable* `a` *dans le fichier ASCII resultat1.txt, en utilisant save, avec les options qui conviennent.*
2. *Trouvez un moyen de sauver les valeurs contenues dans* a^T *dans le fichier ASCII resultat2.txt, et comparez avec la question précédente.*

Vérifiez dans le fichier texte les valeurs enregistrées.

L'objet de la première question de l'exercice 5 est de fournir un fichier texte contenant les données se la variable a. La réponse est en haut de cette page ou dans la page qui précède. L'objet de la deuxième question de l'exercice 5 est de faire la même chose avec la transposée de a, mais on ne peut exactement utiliser la même syntaxe. La réponse à trouver consiste à copier la transposée dans une autre variable et de faire ce qui a été fait à la précédente question avec cette nouvelle variable.

3.2 Supprimer des variables

Une variable peut être effacée de l'espace de travail via `clear variable1 variable2` Attention : l'instruction `clear all` efface toutes les variables de l'espace de travail !

Question 2 *Enregistrez la variable A dans un fichier binaire de données*[11]*, supprimez cette variable de l'espace de travail et retrouvez cette variable à partir du fichier de données, et ce, sans avoir changé de répertoire par défaut.*

3.3 Affichez les résultats

On accède aux éléments du vecteur comme vu précédemment.

```
      t=['la valeur est ', num2str(m)]
t =
la valeur est 6.633
      t(12:18)
ans =
st 6.63
```

Les fonctions de conversion qui nous intéressent plus particulièrement : **num2str** et **str2num** pour transformer un nombre en une chaîne de caractères (number to string) ou transformer une chaîne de caractères en nombre (string to number).

```
      r=2;
      circonference=2*pi*r;
      message=['La circonference vaut ', num2str(circon
message =
La circonference vaut 12.5664
      nb=str2num('12.5') % Transforme en nombre 12.5 la
                         % de caractères '12.5'
nb =
12.5000
      message = ['L''incontournable valeur de pi est ',
' à la quatrieme decimale pres.']
message =
L'incontournable valeur de pi est 3.1416 à la quatrieme
```

Remarquez que dans l'exemple précédent, Matlab/Octave n'a pas reprécisé la valeur de r après l'instruction **r=2**, cela provient de

la présence d'un point virgule qui sert seulement à éviter ce genre d'affichage. En pratique il est utile de mettre des points virgule pour ne laisser que les affichages réellement importants.

3.4 Valeurs particulières

Matlab/Octave dispose d'un certain nombre de constantes prédéfinies : `pi` désigne π, `i` et `j` désigne le nombre complexe i. Par contre e n'est pas une constante prédéfinie. Voici comment coder $e^{j\frac{\pi}{3}}$

```
      exp(j*pi/3)
ans =
   0.5000 + 0.8660i
```

Il faut cependant faire attention que si `j` ou `pi` ont été utilisés pour stocker d'autres valeurs ceci ne peut plus fonctionner. Dans ce cas il suffit de faire

```
clear j,pi,
```

On peut noter que l'apostrophe vue p. 9 qui permettait d'appliquer la transposée à une matrice est aussi pour les complexes la commande permettant d'obtenir l'expression conjuguée.

```
      exp(j*pi/3)'
ans =
   0.5000 - 0.8660i
```

De fait quand on applique l'apostrophe à une matrice complexe, le résultat est la transposée conjuguée. On obtient la transposée non conjuguée en rajoutant un point avant l'apostrophe.

Les fonctions `real` et `imag` permettent d'obtenir la partie réelle et imaginaire d'un complexe.

De même la notation `abs` qui permettait p. 57 de calculer la valeur absolue d'un nombre réel est aussi pour les nombre complexes la commande permettant de calculer le module d'un nombre complexe.

```
      abs(exp(j*pi/3))
ans =
1
```

Exercice 6 (15b) *Calculez la partie imaginaire du complexe* $z = (1-j).^2 * e^{j\frac{\pi}{6}}$, *(j étant le nombre complexe de partie réelle nulle et de partie imaginaire égale à* 1*). Indiquez le résultat en format long.*

4 Expressions logiques, contrôles et boucles

4.1 Contrôle avec `if`

La syntaxe de cette structure de contrôle est la suivante :

```
if expression logique
  suite d'instructions 1;
else
  suite d'instructions 2;
end
```

Il vaut mieux utiliser des opérateurs pour réaliser des tests, plutôt que l'instruction `if` à chaque fois que c'est possible. Par exemple les instructions suivantes

```
      if a>0.5, b=1, else b=0, end
b =
1
```

peuvent être remplacées par

```
      b = (a>0.5)
b =
1
```

Toute variable non nulle est considérée comme une expression logique vraie lors d'un test :

```
      if a, disp('a est non nul'), else disp('a est nul
a est non nul
```

La fonction `exist` permet entre autre de savoir si une variable existe.

```
     if exist('inconnu') inconnu=inconnu+1; else inconn
```

Pour tester le fonctionnement de cette commande, il suffit de l'exécuter plusieurs fois. Elle permet par exemple d'utiliser la même expression pour initialiser et incrémenter un compteur.

4.2 Expressions logiques

Pour comparer des valeurs, on distingue les opérateurs $<$, $<=$, $>$, $>=$, $==$, $\sim=$. L'opérateur $\sim=$ signifie "est différent de". Le résultat de ces comparaisons vaut 1 si c'est vrai et 0 sinon.

```
      test= 1<0
test =
0
```

En pratique quand on veut savoir si deux valeurs sont identiques ou sont différentes, il vaut mieux éviter d'utiliser $==$ ou $\sim=$ à moins de savoir que ce sont des entiers. Pour cela on peut remarquer que x est proche de y à 10^{-10} peut s'écrire mathématiquement $|x-y| < 1e-10$ et de même x n'est pas proche de y à 10^{-10} peut s'écrire mathématiquement $|x-y| > 1e-10$

Aussi pour vérifier si $\cos(\frac{\pi}{6}) = \frac{\sqrt{3}}{2}$, on aboutit à une absurdité en écrivant que :

```
     test=(cos(pi/6)==sqrt(3)/2)
test=
0
```

La raison est touts simplement que $\cos(\frac{\pi}{6}) - \frac{\sqrt{3}}{2}$ est évalué par Matlab/Octave à environ : $1.1102e-016$.

On utilise la fonction `abs` signifie la valeur absolue

```
     abs(-3)
ans=
3
```

Et il vaut mieux faire le test suivant

```
     test=(abs(cos(pi/6)-sqrt(3)/2)<1e-10)
test=
1
```

Le résultat, `test` est considéré comme un bouléen, dit *logical*

```
      class(test)
ans=
logical
```

Il est maintenant possible de combiner plusieurs tests en utilisant les opérateurs logiques (listés dans `help ops`) AND (&), OR (|), NOT (∼)

```
      a=4; b=1;
      a<3 | b==0
ans =
0
      ~(a>3 | b~=0)
ans =
0
```

A la différence du C, il n'existe pas en Matlab/Octave d'opérateur ternaire ? :, mais on peut interpréter les valeurs bouléennes comme des valeurs numériques. Par exemple la séquence d'instructions : *si* $x > 5$ *alors* $y = 5$ *sinon* $y = x$ qui aurait pu s'implémenter en Matlab/Octave avec

```
      if x>5 y=5, else y=x; end
```

Cette séquence peut aussi s'implémenter avec

```
      x=6
      y=x+(5-x)*(x>5)
y=
5
      x=3
      y=x+(5-x)*(x>5)
x=2
```

Cette derniière implémentation a l'avantage d'être utilisable pour des vecteurs ou des tableaux et de permettre ainsi d'éviter l'utilisation d'une boucle.

Une autre différence avec le C, est que lorsque Matlab/Octave évalue une expression logique, c'est l'ensemble des termes de l'expression logique qu'il évalue, même lorsqu'une partie de ces termes ne sont pas utiles à la détermination du résultat global.

Toutes ces fonctionnalités s'adaptent aux matrices.

```
      A=reshape(1:8,2,4);
      A>4
ans =
     0     0     1     1
     0     0     1     1
```

Des opérateurs spécifiques aux matrices sont :

- `any` : teste s'il existe au moins un élément non nul dans un vecteur, ou dans chaque colonne d'une matrice
- `all` : teste si tous les éléments d'un vecteur sont non nuls, ou tous les éléments de chaque colonne d'une matrice
- `find` : renvoie les coordonnées des éléments non nuls d'un vecteur ou d'une matrice

Par exemple

```
      A=3:6;
      test=(A>4)
test =
0 0 1 1
      any(test)
ans =
```

```
1
      all(test)
ans =
0
```

Ces fonctionnalités s'appliquent aussi pour les matrices de la même façon que pour les autres fonctions, (avec un seul argument, ces fonctions donnent un vecteur ligne dont les composantes correspondent à l'application successive de la fonction sur chaque colonne, avec deux arguments, ces fonctions donnent un vecteur ligne si le deuxième argument est 1 et un vecteur colonne si le deuxième argument est 2).

Si l'on veut tester tous les éléments de la matrice, on devra écrire

```
      all(A(:)>4)
ans =
0
```

ou de façon équivalente

```
      all(test(:))
ans =
0
```

La fonction find s'utilise par exemple comme suit

```
>> A=reshape(1:8,2,4);
>> [indi, indj] = find(A>4)
indi =

     1
     2
```

```
     1
     2


indj =

     3
     3
     4
     4
```

Les indices lignes sont dans indi et les indices colonnes dans indj, en effet les valeurs dans les matrices en Matlab/Octave sont rangées conformément à la notation des composantes matrices et doivent être lues dans l'ordre successif des colonnes. Ainsi le deuxième élément de A qui vérifie la condition $A > 4$ est sur la deuxième ligne et la première colonne et vaut 5.

On peut aussi récupérer les indices des éléments qui satisfont au test sous la forme d'un seul vecteur :

```
>> k=find(A>4)
k =

     5
     6
     7
     8
```

Les valeurs de `k` sont les indices des éléments de `A(:)`. L'indice $k = 6$ désigne ici la deuxième ligne et la troisième colonne et correspond

à la valeur $A(6) = 7$. On peut remplacer directement des éléments d'une matrice qui satisfont à un test grâce à l'instruction `find` :

```
>> A(find(A>4))=-1
A =
   1   3  -1  -1
   2   4  -1  -1
```

Tous les éléments de A supérieurs à 4 ont été remplacés par -1.

```
      find(Q) % Donne les indices des éléments non
% nuls de Q(:)
ans =
3
4
5
6
9
12
```

On peut passer d'un indiçage $MN \times 1$ à un indiçage $M \times N$ grâce à `ind2sub` (index to subscript).

4.3 La boucle `for`

La syntaxe de cette boucle est classique :

```
for indice=1:4
  indice,
end
```

Il est recommandé d'utiliser le moins possible de boucles, car elles sont plus lentes qu'un traitement matriciel. A chaque fois que l'on sera tenté de faire une boucle, il faudra se poser la question : "N'y a-t-il pas moyen de faire autrement ?". L'exemple classique que l'on donne pour illustrer ce propos est le suivant : la ligne

```
for k=1:50, v(k)=k; end
```

est équivalente à l'instruction

```
v=1:50;
```

Ou encore, la boucle

```
for k=1:50
w(k)=v(k)^2+2*v(k)-1;
end
```

est équivalente à l'instruction

```
w=v.^2+2*v-1;
```

qui n'utilise que le traitement matriciel. En conclusion de ce paragraphe :

VECTORISATION D'INSTRUCTIONS = RAPIDITÉ D'EXÉCUTION + RAPIDITÉ DE PROGRAMMATION

Matlab/Octave permet très facilement de changer et d'augmenter la taille des variables que l'on utilise, cependant pour que l'exécution d'un certain nombre d'instructions aille plus vite, il est préférable de définir à l'avance la taille maximale de la variable.

Exercice 7 *Cet exercice porte sur les chaînes de caractères, tests, contrôles, ['a'] all any disp eval if/else for length load num2str numel strfind*

1. *B = matrice U seuillée à 0,5 (c'est-à-dire que chaque composantes de B valent 1 lorsque la composante de U est supérieure à ce seuil et 0 sinon). U est calculé dans l'exercice 4 (p. 46).*
2. *s = chaîne de caractères composée de la phrase "l'arnaque du siècle"*
3. *Quelle est la longueur de s ?*
4. *i = position des 'a' dans s*
5. *Remplacez les 'a' par des 'o' dans s*
6. *Rajoutez à s la chaîne " : un reportage signé Jean-Claude Dusse"*
7. *Testez si la chaîne contient un 'ort', et affichez le message "'ort' trouve" le cas échéant, en utilisant* `strfind`.
8. *On considère le vecteur* `c` *défini par l'exercice 1 (p. 30 ou juste après). Déterminez les indices des coordonnées de* `c` *inférieures à* 10^{-14} *et remplacez-les par 0.*
9. *Chargez en mémoire le fichier matrices.mat, qui contient quatre matrices D1, D2, D3, D4. Dans la matrice D4, affectez à la variable n le nombre d'éléments négatifs. S'il est supérieur à la moitié du nombre d'éléments de D4, affichez le message "il y a plus d'éléments négatifs que positifs", sinon affichez "il y a plus d'éléments positifs que négatifs"*

5 Réaliser un programme sous Matlab/Octave

5.1 Utilisation d'un fichier de commande

On peut mettre une commande dans un fichier de commande. L'extension sous Matlab/Octave de ce type de fichier est ".m". Un des onglets en haut à gauche de l'environnement graphique (*New script*) ou à travers le menu déroulant la sélection successive de *File*, *New*, *Script* permettent d'ouvrir un `m-file` qui est justement un éditeur de texte. En fait n'importe quel éditeur de texte peut tout aussi bien convenir. Les raccourcis clavier `Ctr-C` puis `Ctr-V` permettent de copier puis coller une commande déjà écrite (et déjà testée) dans la fenêtre de commande vers le fichier de commande. L'indentation (i.e. le nombre d'espaces en début de ligne de commande) ne modifie pas la façon dont l'instruction sera exécutée.

L'exécution des lignes de commandes écrites dans le `m-file` se fait à travers l'éditeur de texte en cliquant sur pour Matlab le triangle vert, et pour Octave, le triangle jaune superposé à une roue dentée apparaissant dans l'éditeur de texte. Il est possible aussi de tester un ensemble de lignes de commandes du fichier de commandes en copiant ces lignes vers la fenêtre de commande avec les mêmes raccourcis clavier `Ctr-C` puis `Ctr-V`.

L'enregistrement de ce fichier se fait à travers les menus déroulants et la sélection successive de *File/Fichier*, *save, enregistrer* ou *save as, enregistrer sous*. Il faut alors préciser le répertoire et le nom du fichier. Attention ce fichier doit être enregistré pendant la séance de TP dans un répertoire correspondant à un disque dur local. Il faut aussi que ce répertoire soit autorisé en écriture, par exemple

il ne peut pas être dans le répertoire Windows ou à l'endroit où le logiciel Matlab/Octave est installé, parce que probablement qu'il n'est pas possible d'écrire sur ces répertoires. En fin de séance de TP, si vous souhaitez conserver ce fichier de commande, alors il doit être enregistré soit sur le répertoire qui vous est attribué sur le réseau, soit sur une clef USB, soit sous la forme d'un message dans une boîte mail.

Question 3 *Réalisez un fichier de commande contenant une commande permettant d'afficher "Bonjour". Sauvegarder ce fichier de commande. Fermez le fichier et le logiciel Matlab/Octave. Ouvrez à nouveau Matlab/Octave et ce fichier et testez les trois façons d'executer cette ligne de commande.*

Dans tout ce qui suit, vous pourrez utiliser ce fichier de commande pour garder une trace des commandes que vous avez écrites.

5.2 Définir une aide en ligne du programme réalisé

Vous pouvez vous-même écrire les informations d'aide pour un script, une fonction ou un répertoire que vous avez créés. Pour les scripts, les commentaires (i.e. les lignes précédées d'un %) situés en début de fichier (à partir de la 1ère ligne) seront pris en compte par le `help`. Pour les fonctions, les commentaires doivent commencer à la 2ème ligne juste après la définition de fonction (située sur la 1ère ligne). Les commentaires doivent contenir une description de la

fonction avec sa syntaxe, la date de réalisation, le nom de la personne qui a écrit le programme, etc. Pour le répertoire, il suffit d'y mettre un fichier `Contents.m` contenant en commentaires l'aide générale pour le répertoire et la liste des fonctions disponibles avec une brève description. On rappelle qu'avant de créer sa propre fonction, il est impératif de vérifier si une fonction du même nom n'existe pas déjà sous Matlab/Octave. (faire un `help` ou utiliser `exist`) pour rechercher des fonctions existant sous Matlab/Octave.

5.3 Script ou fonction

Un script ou une fonction est un fichier dont l'extension est `.m` composé d'une succession de lignes de commandes. La différence est qu'une fonction prend en argument un certain nombre de paramètres avec un passage par copie un renvoie un certains nombre de paramètres avec passage par copie. Les paramètres peuvent être des valeurs, des chaînes de caractères ou des tableaux ou d'autres types de variables. Il n'y a pas besoin de fonction `return`[12]. Voici un exemple de fonction

```
function [a,b]=echanger(a,b)
  tmp=a;
  a=b;
  b=tmp;
```

Le nom de la fonction est déterminée par le nom du fichier et non par le nom qui suit la commande `function`.

Notez que l'instruction `echanger(a,b)` ne va pas modifier les valeurs de `a` et `b`, il faut appeler cette fonction de la façon suivante :

12. Cette fonction existe quand même, elle permet de terminer l'exécution d'une fonction, les valeurs renvoyées ne doivent pas être placées après cette fonction.

```
[a,b]=echanger(a,b);
```

5.4 Appel d'un script ou d'une fonction

Pour exécuter un script ou une fonction, il faut que cette fonction soit sauvegardée dans un répertoire, ce répertoire doit être soit celui désigné par la fonction `pwd` (voir p. 48), soit que ce répertoire fasse parti de la liste des répertoires où Matlab fera sa recherche. On peut ajouter un nouveau répertoire à cette liste de répertoire avec l'onglet *addpath* puis *setpath* ou alors avec la commande Matlab/Octave `addpath`.

Question 4 *Changer le répertoire de travail par défaut pour être dans un répertoire personnel d'une part avec la commande* `addpath` *et d'autre part les onglets de l'environnement Matlab/Octave. Exécutez en faisant appel en ligne de commande le script réalisé à la question 3, (p. 66).*

Question 5 *Enregistrez et utilisez le programme* `echange`*. Que fait ce programme ? Rajoutez une aide en ligne à cette fonction et vérifiez que cette aide en ligne fonctionne.*

5.5 Débogage

Lorsqu'une erreur se produit dans un script ou une fonction, un message apparaît dans la fenêtre de commande, qui renseigne sur le type de l'erreur qui s'est produite. Le langage étant interprété, si les programmes sont simples et courts, il est facile de faire les corrections directement sans avoir besoin d'outils particuliers de débogage. En revanche, quand les programmes sont plus élaborés, il devient utile de faire appel aux outils de débogage mis à disposition (help debug). On ne décrit ici brièvement que les plus élémentaires, soit `keyboard`, `return`, `dbstop`, `dbclear`, `dbquit`. Le débogage peut aussi être géré directement à l'aide du menu de l'éditeur de fichiers Matlab/Octave, onglet debug. L'instruction `keyboard` se place dans un script ou une fonction. Elle interrompt l'exécution du programme et rend la main à l'utilisateur avec le prompt K>> pour Matlab et `debug`> pour Octave au lieu de >>. L'utilisateur peut dès lors interroger toutes les variables et les fonctions de l'espace où se trouve l'instruction `keyboard` : mis à l'intérieur d'une fonction, on a accès aux valeurs des variables locales à la fonction, alors que celles-ci ne sont pas disponibles depuis l'espace de travail. Pour revenir à l'exécution normale, on tape `return`. On peut également placer des points d'arrêt dans une fonction grâce à `dbstop`, puis à partir de là avancer pas à pas avec `dbstep`, ou quitter le mode debug avec `dbquit`. Voir les fonctions de débogage disponibles dans le help debug, ainsi que leur syntaxe.

Lorsque l'on cherche des informations sur une fonction, il faut toujours

1. Commencer par utiliser l'aide en ligne de commande
2. Lire à partir du début (remonter si besoin avec le curseur à

droite) pour l'utilisation standard

3. Utiliser l'aide HTML (onglet Help/ Matlab Help de la barre de menu) si l'on désire plus de détails

5.6 Fonctions

Le fait de décomposer un programme en un ensemble de fonctions distinctes qui s'appellent les unes les autres présentent l'intérêt que les variables utilisées au sein des différentes fonctions n'interagissent avec le reste du programme qu'a travers les entrées et sorties prévues.

Il y a deux façons de constituer un programme composé d'un grand nombre de fonctions soit au sein d'un même fichier d'extension `.m` soit en prévoyant pour chaque fonction un fichier particulier d'extension `.m`. Il est possible de mélanger les deux conventions, à condition de connaître la règle suivante. Au sein d'un fichier donné, toute fonction peut être appelée par une autre fonction, en revanche une fonction, un script ou une ligne de commande ne peut appeler une fonction placée sur un autre fichier à moins que cette fonction soit la première de ce fichier.

Pour rendre plus clair l'étendue d'une fonction et la distinguer des autres fonctions d'un même fichier `.m`, il est souhaitable d'indiquer la fin d'une fonction. Ceci peut se faire avec `end` en Matlab/Octave ou avec `endfunction` en Octave.

6 Affichage dans la fenêtre de commande, conversion de nombres en caractères

Pour gérer le format d'affichage à l'écran, on utilise les fonctions `fprintf` et `sprintf`, qui permettent de spécifier le format voulu :

```
    help fprintf
.........
```

Les fonctions `fprintf` et `sprintf` ont la même syntaxe qu'en C.

```
    disp(message)
La circonférence vaut 12.5664
```

à comparer au résultat de l'instruction suivante :

```
>> fprintf('La circonférence vaut %f\n', circonference)
La circonférence vaut 12.566371
>> volume=0.33510;
>> fprintf('Aujourd''hui %s, le volume du cône vaut %2.
Aujourd'hui 07-Sep-2010, le volume du cône vaut 0.33510
>> s=sprintf('la valeur de m est %1.8f',m);
>> disp(s)
la valeur de m est 6.63324958
```

On peut aussi saisir à l'écran une valeur, et la mettre dans une variable via la fonction input.

```
>> nom=input('Tapez votre nom : ','s');
Tapez votre nom : Zongo
```

```
>> age=input('Quel est votre âge en années ? ')
Quel est votre âge en années ? 7
>> disp([nom,' ',num2str(age),' ans'])
Zongo 7 ans
>> A(1,1)=input('valeur de A(1,1) : ');
valeur de A(1,1) : 9.5
```

6.1 Affichage de graphiques

Un graphique se trace avec l'instruction `plot` en donnant sous la forme de deux vecteurs distincts de même taille, le premier représentant l'ensemble des abscisses des points et le deuxième l'ensemble des ordonnées de ces mêmes points.

```
figure(1); plot([ 1 3 4],[2 1 1],'+');
```

Dans cet exemple les points tracés sont les points de coordonnées $(1, 2), (3, 1), (4, 1)$.

Plus généralement on peut ainsi tracer une courbe parabolique $y = x^2$ pour $x \in [-1, 1]$.

```
x=-1:1e-3:1; y=x.^2;
figure(2); plot(x,y);
```

Pour afficher plusieurs courbes en même temps, il est certes possible d'utiliser les commandes `hold on` et `hold off`. Le plus simple reste cependant d'accumuler une succession de vecteurs alternant la définition des abscisses avec la définition des ordonnées.

```
x1=-1:1e-3:1.1; y1=x1.^2; x2=-1.1:1e-3:1.1; y2=0.5
figure(3); plot(x1,y1,x2,y2);
```

Dans cet exemple, en traitement de signal et en supposant que l'abscisse représente le temps en secondes, on dit que les deux signaux y_1 et y_2 ont une même période d'échantillonnage $T_e = 10^{-3}$ s et que la fréquence d'échantillonnage est $f_e = 10^3$ Hz= 1 kHz.

Sur la fenêtre, l'onglet représentant en Matlab un `+` entouré d'une petite loupe et en Octave un `Z+` permet d'agrandir une zone particulière et notamment de trouver les coordonnées de l'intersection entre les deux courbes. Cette technique peut être utilisée pour trouver les solutions approchée d'une équation à une inconnue. Par exemple, la figure tracée permet de trouver les deux solutions de $x^2 - 0.5x - 0.5 = 0$. Il s'agit de solutions approchées dépendant de la résolution utilisée pour définir les vecteurs x_1 et x_2.

Question 6 *Trouvez les solutions approchées à* 10^{-2} *de l'équation* $x^2 - 0.5x - 0.5 = 0$

Avec la fonction `plot`, il est possible de choisir la couleur des courbes `'b','m','r','k','c','g'`, la forme des points `'+','s','v','.','o'`, la forme de la courbe `':','-','-.'`, l'épaisseur de la courbe avec l'option `'Linewidth'`.

```
figure(3); plot(x1,y1,'r-.',x2,y2,'g-','Linewidth',2);
```

En Matlab, dans cette exemple, ce sont toutes les courbes qui sont tracées avec une plus grande épaisseur, tandis qu'en Octave ce n'est que la dernière courbe qui est tracée avec une plus grande épaisseur. Les instructions `legend, title, xlabel, ylabel` permettent respectivement de rajouter une légende, un titre, l'ajout de texte sur l'axe des abscisses et des ordonnées.

Exercice 8 *Cet exercice concerne le tracé de courbes (affichage 2D).*

1. *Créez la variable t contenant tous les instants d'échantillonnage à la fréquence $f_e = 1kHz$ entre 0 et $0,1$ s. Affichez en fonction du temps t le vecteur w_1 contenant les valeurs (échantillonnées) de la fonction $x(t) = 2\cos(2\pi ft + \varphi_1)$ pour une fréquence $f = 50$ Hz, une phase $\varphi_1 = \frac{\pi}{3}$. Affichez la courbe en bleu avec une épaisseur de ligne de 2 et faire apparaître les échantillons par des étoiles. Renseignez les axes et mettez un titre.*
2. *Sur le même graphique, affichez en fonction du temps et en rouge le vecteur w_2 contenant les valeurs de la fonction $y(t) = 2\cos(2\pi f_1 t + \varphi_1)\sin(2\pi f_2 t + \varphi_2)$ pour les fréquences $f_1 = 50$ Hz, $f_2 = 300$ Hz, la phase $\varphi_2 = \pi/4$, et échantillonnée à la fréquence $f_e = 1$ kHz entre 0 et 0.1 s. Mettez une épaisseur de ligne de 2 et faire apparaître les échantillons par des ronds. Changez le titre ('Signaux périodiques, Fe=1kHz.') et mettez une légende.*

Exercice 9 *Cet exercice concerne les fonctions et l'affichage de surfaces en 3D. Il concerne les commandes* `axis`, `colormap`, `function`, `grid`, `mesh`, `meshc`, `meshgrid`, `max`, `min`, `ones`, `shading`, `surf`, `surfc`.

1. *Créez une fonction Matlab/Octave* `sinusc` *qui calcule le sinus cardinal défini comme suit :* $f(x) = \frac{\sin(\pi x)}{\pi x}$ *L'argument* x *peut être un scalaire ou une matrice, et doit pouvoir être nul.*
2. *Affichez la fonction* $f(x, y) = sinc(xy)$ *pour* x *allant de* -3 *à* 3 *par pas de* 0.06 *et* y *allant de* -2 *à* 2 *par pas de* 0.04*, en utilisant meshgrid. Visualisez la surface sous forme d'un maillage avec mesh. Fixez les axes en* x *et* y *aux valeurs minimum et maximum de* x *et* y*, et l'axe* z *à* $[-0.5, 1]$*. Visualisez avec les contours. Visualisez une surface lisse et modifier ensuite le rendu de surface (*`shading`*,* `colormap`*).*

6.2 Affichages d'images

L'instruction `imshow` permet d'afficher des images en niveaux de gris ou en couleur et plus ou moins avec une résolution correspondant à ce qu'on recherche lorsqu'on regarde une image. Cette fonction utilise la convention habituelle qui veut qu'une valeure entière égale à 255 corresponde au blanc, une valeur nulle correspondant à 0 correspond au noir. Lorsqu'il ne s'agit pas de valeur entière, 1 correspond au blanc et 0 correspond au noir.

On pourrait croire que pour visualiser une image composée d'un carré blanc et d'un carré noir il suffit d'écrire

```
A=[1 0]; figure(1); imshow(A);
```

Dans cet exemple, il n'y a que deux pixels qui ont été affichés, et

sous Matlab c'est trop petit pour être vu, aussi pour afficher cette image, il vaut mieux faire

```
A=[ones(128) zeros(128)]; figure(1); imshow(A);
```

Il existe de nombreuses images dans Matlab, on peut les lister avec la commande **help imdemos**. En revanche en Octave, il faut les télécharger et les enregistrer dans le répertoire courant. Un site contenant des images est :

`http://people.math.sc.edu/Burkardt/data/tif/tif.html`

```
>> I=imread('autumn.tif');
>> figure(2); imshow(I);
```

On peut remarquer d'ailleurs que `I` n'est pas exactement une matrice, formellement il s'agit d'un tenseur d'ordre 3, mais dans la pratique on parle de matrice 3D.

```
>> size(I)
ans=
206   345     3
>> size(sum(I))
ans =
     1   345     3
>> sum(sum(sum(I)))
ans =
    23366084
```

Les opérations éléments par éléments ou l'utilisation de l'opérateur d'énumération se généralise bien à ce type de variable. En revanche il n'est pas possible d'appliquer des opérations matricielles.

L'idée consiste à faire une boucle sur la troisième dimension et à appliquer les opérations classiques sur les deux premières dimension en fixant la troisième dimension.

Pour les images cette idée s'applique bien car il n'y a que trois éléments dans la troisième dimension, ce sont les éléments correspondant au rouge, vert et bleu. Par ailleurs ici il faut utiliser l'instruction `double` pour donner la possibilité de faire des opérations (en effet quand une donnée est de type `uint8` ce qui est généralement le cas pour une image, certaines opérations ne sont pas supportées).

7 Exercices supplémentaires

Exercice 10 (1) *On cherche les deux intersections entre la courbe* $f(x) = \frac{1}{x}$ *et* $g(x) = x - 1$ *pour* $x \in [-2, 2]$.

1. *Représentez les fonctions* f *et* g *sur un graphique et lire graphiquement et avec une précision de* 10^{-1} *les valeurs de* x_1 *et* x_2 *tels que* $f(x_1) = g(x_1)$ *et* $f(x_2) = g(x_2)$. *Ecrivez les instructions Matlab/Octave qui ont permis de tracer ce graphique.*
2. *On cherche maintenant à connaître plus précisément les valeurs des intersections, pour cela vous pourrez utiliser la fonction Matlab/Octave* `fzero` *qui permet de trouver le zéro d'une fonction à proximité d'une abscisse. Déterminez* x_1 *et* x_2 *avec une précision de* 10^{-8} *(utilisez* `format long` *pour faire apparaître les décimales présentes en mémoire). Remarquez qu'il peut être souhaitable de poser* $h = f/g - 1$ *au lieu de* $h = f - g$.

Exercice 11 (4) *On définit la matrice* D *à partir de l'instruc-*

tion `D=randn(5,8);` *Ecrivez des instructions Matlab permettant de connaître le nombre total d'éléments contenus dans D.*

Exercice 12 (21) *On cherche à évaluer*

$$I = \int_0^1 \frac{dt}{1+t^2}$$

On sait que le résultat est $I = \frac{\pi}{4}$*, en effet un changement de variable* $t = \tan(u)$ *et le fait que*

$$dt = \frac{1}{\cos^2(u)} du = (1+t^2)du$$

montrent que cette intégrale peut se mettre sous la forme

$$I = \int_0^{\frac{\pi}{4}} du$$

On obtient une confirmation avec une formule des rectangles

`https://fr.wikipedia.org/wiki/Calcul_num%C3%A9rique_d'u`

$$I1 = \sum_{n=0}^{N-1} f\left(\frac{n}{N}\right)\frac{1}{N} \tag{1}$$

où $f(t) = \frac{1}{1+t^2}$

1. *Définissez une fonction notée* f *au moyen de l'instruction @ (*`help function_handle` *en Matlab et* `help @` *en Octave), de façon que l'instruction* `f(1)` *donne le résultat* 0.5.
2. *A partir de la formule (1) et de l'instruction @, définissez une fonction notée I1 qui à* N *associe l'approximation correspondante de* I*. Que faut-il choisir pour* N *pour avoir une approximation à* $1e-8$ *?*

Exercice 13 (5) *On considère la matrice B défini par l'instruction Matlab/Octave* `B=magic(7);` *Ecrivez les instructions permettant d'afficher la troisième colonne de cette matrice B.*

Exercice 14 (6) *Ecrivez les instructions Matlab/Octave permettant de générer un vecteur ligne contenant tous les entiers pairs entre 456 et 222 rangé du plus grand au plus petit. Combien y en a-t-il de ces entiers pairs ?*

Exercice 15 (7) *Tracez sur l'intervalle de temps $[-5, 5]$ le signal échelon défini par $x(t) = 1$ si $t > 0$ et $x(t) = 0$ sinon. Ecrivez les instructions Matlab/Octave permettant de créer et d'afficher ce signal.*

Exercice 16 (8) *Créez un vecteur ligne composé de 50 nombres tirés aléatoirement suivant la loi uniforme sur l'intevalle $[0, 1]$. Ecrivez les instructions Matlab/Octave permettant de créer ce vecteur.*

Exercice 17 (9) *On cherche à calculer $\Sigma_{n=1}^{\infty} \frac{1}{n^2}$ Ecrivez les instructions Matlab/Octae permettant de calculer cette somme.*

Exercice 18 (10) *Représentez graphiquement la fonction $f(x) = x^2$ pour $x \in [-1, 1]$. Ecrivez les instructions Matlab/Octave permettant de créer et de représenter graphiquement cette fonction.*

Exercice 19 (11) *Trouvez la valeur de $t \in [0, 2\pi]$ qui maximise $\cos(t)+\sin(t)$ Ecrivez les instructions Matlab/Octave permettant de trouver cette valeur de t.*

Exercice 20 (12) *On considère la matrice C défini par l'instruction Matlab/Octave* `C=toeplitz([0.5 1.5 2.5 3.5]);` *Ecrivez les instructions Matlab/Octave permettant compter le nombre de valeurs supérieures à 2 contenues dans cette matrice C.*

Exercice 21 (13) *On cherche à créer un vecteur qui contient tous les nombres de 1 à 123 par pas de 0.5. Ecrivez les instructions Matlab/Octave permettant de créer ce vecteur. Quel est la longueur du vecteur ainsi créé ?*

Exercice 22 (14) *On considère le vecteur u défini par l'instruction Matlab/Octave* `u=randn(1,50);` *Créez un nouveau vecteur v en copiant le vecteur u et remplaçant toutes les valeurs négatives de u par des zéros. Ecrivez les instructions Matlab/Octave permettant de créer ce vecteur v.*

Exercice 23 (16) *Tracez pour $t \in [0, \frac{1}{2}]$ et par pas de 0.01 le signal $x(t) = \sin(2\pi f_0 t + \frac{\pi}{7})$ avec $f_0 = 6\text{Hz}$. Ecrivez les instructions Matlab/Octave permettant de créer et d'afficher graphiquement ce signal $x(t)$*

Exercice 24 (17) *On génère un vecteur aléatoirement avec l'instruction* `u=5*randn(1,100);` *Ecrivez les instructions Matlab/Octave permettant de trouver la valeur au sein de ce vecteur qui s'approche le plus de π.*

Exercice 25 (18) *On cherche à calculer $\Sigma_{n=1}^{+\infty} \frac{(-1)^n}{n}$ Ecrivez les instructions Matlab/Octave permettant de calculer cette somme.*

Exercice 26 (19) *On cherche à fabriquer une matrice de taille 7×7 formés de 0 et de 1 de la façon suivante*

```
0101010
1010101
0101010
1010101
0101010
1010101
0101010
```

Ecrivez des instructions Matlab/Octave permettant de créer cette matrice mais avec une taille de 49×49.

Exercice 27 (20) *On définit* $n!$ *comme étant le produit des entiers inférieurs à* n *et strictement positifs. Ecrivez les instructions Matlab/Octave permettant de calculer* $\ln(57!)$.

Exercice 28 (24) *Cet exercice est très similaire à l'exercice 12 (p. 78). On cherche maintenant à calculer*

$$J = \int_{-\sqrt{3}}^{-\sqrt{3}} \frac{dt}{1+t^2}$$

Un calcul similaire à montre que $J = \frac{2\pi}{3}$

La nouvelle formule d'approximation proposée est

$$J1 = \sum_{n=0}^{N-1} f\left(a + n\frac{b-a}{N}\right)\frac{b-a}{N}$$

1. *Définissez une fonction J1 qui à* a, b, N *associe l'approximation correspondante.*
2. *Sachant que l'erreur est très grossièrement de l'ordre de la variation de hauteur moyenne sur chaque trapèze, que faut-il choisir pour* N *pour avoir une approximation à* 10^{-8} *?*

3. *On utilise maintenant la fonction Matlab/Octave* `trapz`*, qui utilise comme argument le vecteur formé des abscisses et le vecteur formé des ordonnées correspondantes et donne l'intégrale. Ici les abscisses sont données par* $a + \frac{b-a}{N}$ *et les ordonnées par* $f(a + \frac{b-a}{N})$*. Définissez une fonction* t *qui à* a, b, N *associe le vecteur des indices et* $J2$ *qui à* a, b, N *associe le résultat de l'approximation en utilisant la fonction Matlab/Octave* `trapz`*. Que faut-il choisir pour* N *pour avoir une approximation à* 10^{-8} *?*

8 Suppléments

8.1 Fonctions en ligne

Les fonctions en ligne sont une manière simple de profiter des avantages de la notion de fonctions sans avoir à créer des fichiers supplémentaires. Il existe deux types de fonctions en ligne.

La commande `inline` requière de mettre tous les champs avec des apostrophes (ce qui est la convention en général dans Matlab/Octave). Voici un exemple de fonction en ligne :

```
f=inline('prod(1:n)','n');
f(3)
```

```
6
```

Question 7 *Que fait cette fonction ?*

Une autre façon d'écrire une fonction en ligne est d'utiliser le symbole '@', l'aide en ligne est obtenue avec `help function_handle`. Dans ce cas, il n'est plus nécessaire de mettre des apostrophes :

```
f2=@(n)prod(1:n);
f2(3),
```

Ces deux formalismes ont en commun qu'il n'est pas prévu d'utiliser des instructions de contrôle et de boucles, ni de créer de nouvelles variables et qu'il ne doit y avoir qu'une seule sortie. Le deuxième formalisme présente l'avantage qu'il est possible de composer les fonctions en lignes et d'une manière générale d'utiliser des fonctions et des variables disponibles dans l'espace de travail sans avoir à les indiquer dans les paramètres d'entrée. La convention est qu'alors ces variables conserveront la valeur qu'ils ont eu au moment où la fonction a été déclarée.

Voici un exemple d'utilisation

```
f3=@(x)1+x/f2(1)+x.^2/f2(2)+x.^3/f2(3)+x.^4/f2(4)+
f4=@(x)f3(-x);
graph=@(f)plot(0:1e-3:2,f(0:1e-3:2));
figure(1); graph(f4);
```

Question 8 *De quelle fonction f_4, est-elle une approximation ?*

Ces fonctions en ligne, ainsi que les fonctions sauvegardés dans un fichier `.m`, permettent d'utiliser les fonctions `fzero`, `fminbnd`,

`integral` qui respectivement permettent de trouver l'intersection d'une fonction avec l'axe des abscisses, le minimum d'une fonction au sein d'un intervalle, l'intégrale d'une fonction sur un intervalle.

Exercice 29 (3) *Créez un fichier Matlab/Octave qui implémente la fonction* $f_1(x) = \frac{1}{1+x^2}$. *Le nom de ce fichier est* `f1.m` *Cette fonction doit permettre de tracer le graphique de cette fonction à partir de l'instruction suivante*

```
figure(1); plot(-1:0.01:1,f1(-1:0.01:1));
```

Ecrivez les instructions Matlab/Octave contenues dans ce fichier `f1.m` *Attention à ne pas mettre dans le fichier* `f1.m`, *l'instruction* `plot` *contenant un appel à* `f1.m`.

8.2 Utilisation des fonctions pour trouver une intersection ou un minimum

Lorsqu'on cherche l'intersection entre deux fonctions (x vérifiant $f(x) = g(x)$), le zéro d'une fonction (x vérifiant $f(x) = 0$), ou le minimum d'une fonction (x vérifiant $\forall y, f(y) \geq f(x)$), on peut avoir une approximation des valeurs recherchées en représentant graphiquement les fonctions. Ainsi si `f` et `g` sont des fonctions définies en ligne ou sous forme de programmes `f.m` et `g.m`, les instructions suivantes permettent de représenter graphiquement les fonctions f et g.

```
>> f=@(x)x.^2;
```

```
>> g=@(x)1./(2+x);
>> x=-1.5:1e-4:1.5;
>> figure(1); plot(x,f(x),x,g(x));
```

Dans cet exemple 10^{-4} est la pas d'échantillonnage. Les touches zoom dans la fenêtre graphique permettent de trouver une approximation des valeurs x correspondant à la ou les intersections entre les fonctions. La/les valeurs trouvées pour l'intersection sont d'autant plus précises que le pas d'échantillonnage utilisé lors de la représentation graphique est petit.

On peut aussi utiliser les fonctions `fzero` et `fminbnd` pour trouver le passage à zéro d'une fonction ou le minimum d'une fonction. Si les fonctions f et g sont définies en ligne, les instructions suivantes permettent de trouver l'intersection proche de 1 et le minimum entre -0.5 et 0.5

```
h=@(x)f(x)-g(x);
x0=fzero(h,1); %1 est la première valeur essayée pour l
x1=fminbnd(f,-0.5,0.5); %[-0.5,0.5] est l'intervalle te
```

Si la différence entre la fonction f et g est définie sous la forme de fonctions `h.m` et si la fonction f est définie sous la forme d'une fonction `f.m`, les instructions suivantes permettent de trouver l'intersection proche de 1 et le minimum entre 1 et 2

```
x0=fzero(@h,1);
x1=fminbnd(@f,1,2);
```

8.3 Appel en ligne de l'interpréteur

La fonction `eval` évalue une chaîne de caractère, c'est-à-dire interprète la ligne de commande correspondant à la chaîne de carac-

tères.

```
>> A=[1 2 3 4;5 6 7 8]
>> i=1; j=1;
>> eval('A(i,j)=10*pi')
A =
31.4159 2.0000 3.0000 4.0000
5.0000 6.0000 7.0000 8.0000
```

Cette fonction permet d'automatiser un certain nombre de tâches. Par exemple on pourrait souhaiter mettre le contenu d'un tableau dans des variables appelées `x1...x20`. Pour illustrer cet exemple on génère un tableau

```
tableau=1:20;
```

On parcourt ensuite ce tableau avec une boucle `for`, on génère une expression qui est une chaîne de caractère et on exécute cette expression.

```
for i=1:20
  ligne=['x',num2str(i),'=tableau(',num2str(i),');'];
  eval(ligne);
end
```

La fonction feval évalue la fonction dont le nom est contenu dans la variable (de classe char) passée en argument.

```
      nom_fonction='cos'
nom_fonction =
cos
      feval(nom_fonction,pi/2)
ans =
6.1230e-017
```

On n'obtient pas 0 pour $\cos(\frac{\pi}{2})$ car il y a le problème de l'arrondi de π.

Exercice 30 *Chargez en mémoire le fichier matrices.mat, qui contient quatre matrices D1, D2, D3, D4. Faites une boucle sur les 4 matrices avec un test qui permet d'afficher, pour chacune des matrices, le message "matrice n° i : carré magique 34" si la somme de chaque ligne et chaque colonne vaut 34 pour la matrice Di.*

8.4 Fonctions avec nombre variable d'arguments

Tout comme en C, il est possible d'avoir un nombre variable d'arguments d'entrée ou de sortie en utilisant `nargin`, `varargin` et `nargout`, voir l'aide en ligne.

Exercice 31 *Ayant chargé en mémoire les quatre matrices D1, D2, D3, D4, réalisez une fonction capable de tester un nombre quelconque de matrices et d'afficher quelles sont les matrices carré magique 34. Concrètement lorsque la fonction est appelée avec l'instruction*

```
est_magique34(D2,D1,D1,D4)
```

l'affichage indique successivement pour les matrices D2,D1,D1, D4 lesquelles sont de carré magiques et lesquelles ne le sont pas.

9 Conteneurs

9.1 Assemblage de variables

Il est possible d'assembler des variables sous un nom unique comme en C avec l'instruction `struct`. L'intérêt est par exemple de passer en paramètre d'une fonction plus facilement un grand nombre de variables. Il est aussi de souder ensemble des informations dont on sait qu'elles sont toujours liées, par exemple l'instruction `dir` fournit un tableau de struct, chaque case du tableau correspond à un fichier ou un répertoire et dans chaque case, on a les champs, nom, date, taille, s'agit-il d'un répertoire, date sous un autre format.

Une structure est définie de la façon suivante :

```
S = struct('nom1', valeur_nom1, ..., 'nomN', valeu
```

ou tout simplement directement par attribution de valeurs (comme lorsque l'on définit une matrice) :

```
S.nom1 = 'Blog';
S.nom2 = 78;
S.nom3 = 4.5e-3
S =
nom1: 'Blog'
```

```
nom2: 78
nom3: 0.0045
```

où **nom1**, ..., **nomN** sont des chaînes de caractères qui définissent le nom des champs (les variables qui composent la structure), et valeur_nom1, ..., valeur_nomN sont les valeurs des champs. Les champs peuvent être de n'importe quelle classe, et l'ordre d'écriture n'a aucune importance. Au moment de l'affichage, Octave rajoute au dessus des différents champs et de leur valeur, l'indication suivante :

```
scalar structure containing the fields:
```

On accède aux valeurs des champs par :

```
S.nom1
S.nom2
...
```

Un exemple :

```
      parametres=struct('constante',0,'lambda',0.5,'pre
parametres =
constante: 0
lambda: 0.5000
precision: 0.0010
```

On change une valeur :

```
      parametres.precision=1e-4
parametres =
constante: 0
lambda: 0.5000
precision: 1.0000e-004
```

On rajoute un champ :

```
      parametres.message_erreur='erreur : valeurs incom
parametres =
constante: 0
lambda: 0.5000
precision: 0.0010
message_erreur: 'erreur : valeurs incompatibles'
```

On peut ainsi rajouter des variables, tout en gardant la même syntaxe pour la première ligne de la fonction, par exemple :

```
function sortie = derivee(p,param)
% Cette fonction calcule pour
% f(p)=(p-c)^2+lambda/(p-lambda)
% la valeur au point p de f et de la derivee
% si abs(p-lambda)>precision
% Les parametres additionnels sont dans la structure pa
% bla bla bla
c=param.constante;
lambda=param.lambda;
prec=param.precision;
erreur=param.message_erreur;
...
sortie.fonction= ...
sortie.derivee= ...
```

renvoie une valeur de sortie qui est également une structure, et l'appel à la fonction se fait simplement par

```
ma_sortie= derivee (p0,parametres);
```

après avoir affecté une valeur à `p0`.

9.2 Cellules

Les cellules en Matlab/Octave peuvent se voir comme des matrices de matrices. On crée une cellule vide N avec l'instruction cell(lignes,colonnes) :
Matlab:

```
>> N=cell(2,3)
N =
[] [] []
[] [] []
```

Octave:

```
>> N=cell(2,3)
N =
{
  [1,1] = [](0x0)
  [2,1] = [](0x0)
  [1,2] = [](0x0)
  [2,2] = [](0x0)
  [1,3] = [](0x0)
  [2,3] = [](0x0)
}
```

On accède à chaque élément d'une cellule avec des parenthèse, et les éléments peuvent être de n'importe quel type. Lorsqu'il n'y a pas assez de place, les valeurs des éléments de la cellule ne sont pas affichées ; seuls sont affichés les types et dimensions des éléments.

```
      N{1,1}='numéro de carte'
'numéro de carte' [] []
```

```
[] [] []
      B=[1 3;0.2 -1]
B =
1.0000 3.0000
0.2000 -1.0000
      N{2,1}=B
N =
'numéro de carte' [] []
[2x2 double] [] []
      N{1,3}=5
N =
'numéro de carte' [] [5]
[2x2 double] [] []
      N{2,2}='possibilité de rendez-vous'
N =
'numéro de carte' [] [5]
[2x2 double] [1x26 char] []
      c=N{2,2}
c =
possibilité de rendez-vous
```

Plus généralement les cellules sont des ensembles de données de nature diverse. On peut ainsi assembler diverses choses

```
I={3,'bientot'}
I={I,4}
```

On peut retrouver les données contenues dans I

```
I{2}
I{1}{1}
I{1}{2}
```

Pour retrouver les données, il y a une différence entre utiliser les parenthèses et les accolades Matlab:

```
>> I(1){1}
??? Error: ()-indexing must appear last in an index exp
>> J=I(1); J{1}
ans =
    [3]    'bientot'
```

Octave:

```
>> I(1){1}
ans =
{
  [1,1] =  3
  [1,2] = bientot
}
```

Matlab:

```
>> I{1}
ans =
    [3]    'bientot'
```

Octave:

```
>> I{1}
ans =
{
  [1,1] =  3
  [1,2] = bientot
}
```

En fait les accolades entourant un chiffre peuvent se voir comme un équivalent en C de l'application sur un pointeur `ptr` de `ptr[0]` qui est un déréférencement tandis que les parenthèses peuvent se voir comme `ptr+0` ou `&ptr[0]`. En pratique elles peuvent être évitées.

Les cellules permettent d'implémenter facilement les arbres. Ce qui est par exemple utile pour simuler l'algorithme de codage de Huffman.

Pour simuler les piles ou les files d'attente, il suffit d'utiliser la possibilité de concaténer les vecteurs ou des ensembles on peut utiliser les fonctions en lignes suivantes.

```
creerPile=@(){};
empiler=@(x,pile){x pile{:}};
dernier=@(pile)pile{1};
depiler=@(pile){pile{2:end}};
estVide=@(pile)isempty(pile);
```

L'utilisation de cette pile se fait de la façon suivante

```
pile=creerPile();
pile=empiler(2,pile);
pile=empiler('+',pile);
pile=empiler(3,pile);
disp('le contenu de la pile est '),
while(~estVide(pile))
  disp(dernier(pile)), pile=depiler(pile);
end;
le contenu de la pile est
     3
+
     2
```

10 Fonctions spécifiques à Matlab

Exercice 32 *Créez la fonction* `magneto.m` *qui selon les options d'entrée en utilisant* `strcmp`

- *enregistre du son en utilisant le périphérique par défaut, avec une fréquence d'échantillonnage de 8000 Hz et pendant 5 secondes au moyen de la fonction* `wavrecord`*. Les données sont sauvées au format Matlab.*
- *ou envoie le son enregistré précédemment sur les haut-parleurs* `wavplay`*. Dans ce cas, un paramètre d'entrée optionnel permet d'amplifier le son écouté ou de l'atténuer. Si la valeur du signal après amplification dépasse le niveau de saturation, un message d'avertissement doit s'afficher dans la fenêtre de commande.*

La fonction renvoie en sortie les valeurs minimale et maximale du signal enregistré ou écouté.

Exercice 33 *Interface graphique : Graphic User Interface (GUI)*
L'outil guide permet de créer des interfaces utilisateur graphiques (Graphic User Interface (GUI) en anglais) qui intègrent diverses fonctionnalités (menus déroulants, curseurs, boutons, etc.). Il est possible de faire appel à des fonctions Matlab ou définies par l'utilisateur.

- *Pour commencer*
  - *Un exemple de GUI est fourni, sous le nom* `GUIexemple.m`*. Exécutez ce programme.*
  - *En utilisant l'outil guide, éditez le graphique existant* `GUIexemple.fig` *et faire le lien avec le programme* `GUIexemple.m`

A l'aide de l'outil guide, créer l'interface graphique `GUImagneto.fig` *et la fonction* `GUImagneto.m` *(ATTENTION : il faut générer l'ossature de la fonction* `GUImagneto.m` *directement en appuyant sur* `Run` *, et ensuite modifier à l'intérieur les fonctions pour l'exécution des tâches). GUImagneto.m comprendra les fonctionnalités suivantes :*

- *Choix d'un fichier*
  - *Un menu déroulant permet de choisir un fichier dans une liste*
- *Enregistrement*
  - *Un bouton permet de démarrer l'enregistrement*
  - *Le son est sauvé dans le fichier .mat correspondant : au choix du menu déroulant ou au fichier audio.mat si le menu déroulant n'a pas été implémenté*
- *Écoute du signal*
  - *le fichier écouté correspond au choix du menu déroulant*
  - *un bouton démarre l'écoute du signal*
  - *un menu déroulant permet de choisir la fréquence d'échantillonnage pour l'écoute uniquement*
  - *un curseur permet de régler le volume d'écoute*
- *Visualisation du signal*
  - *une fenêtre graphique intégrée permet de visualiser le signal*
  - *en appuyant sur un bouton poussoir, la même fenêtre graphique permet de visualiser la DSP du signal*

A Synthétiser des données aléatoires

Les fonctions `rand` et `randn` utilisent des algorithmes déterministes pour simuler des données aléatoires. Voici un exemple le montrant sous Matlab.

```
Lancer Matlab
>> rand(1)
ans =
   0.8147
>> rand(1)
ans =
    0.9058
>> exit
```

La dernière commande ferme Matlab. Refaites l'expérience et constatez que vous obtenez toujours les mêmes résultats.

Pour obtenir de vraies données aléatoires, il est nécessaire d'utiliser la commande `rng('shuffle')` avant la première utilisation de `rand` ou `rand("seed", "reset")` sous Octave. Faites l'expérience.

Sous Octave, l'expérience ne fonctionne pas parce que par défaut, le générateur de données aléatoire est initialisé avec le temps CPU.

B Formules empiriques pour le calcul de la moyenne et de l'écart-type

Soient $x_1, \ldots, x_n$ un ensemble de données. La moyenne empirique est donnée par

$$m = \frac{1}{n} \sum_{i=1}^{n} x_i$$

L'écart-type empirique est donnée par

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - m)^2}$$

C Modifier la moyenne et l'écart-type d'un processus aléatoire gaussien centré et d'écart-type 1.

Lancez les commandes suivantes en observant bien à chaque fois l'échelle de droite des figures réalisées :

```
>> figure(1); plot(rand(1,1000));
>> figure(1); plot(randn(1,1000));
>> figure(1); plot(randn(1,1000));
>> figure(1); plot(53*randn(1,1000));
>> figure(1); plot(53+randn(1,1000));
```

Soit X un processus aléatoire gaussien centré et d'écart-type 1 et μ et $\sigma > 0$ deux nombres. Alors $\sigma X + \mu$ est un processus aléatoire gaussien de moyenne μ et d'écart-type σ.

En effet l'espérance de $\sigma X + \mu$ vaut

$$E[\sigma X + \mu] = \sigma E[X] + \mu = \sigma \times 0 + \mu = \mu$$

Et la variance de $\sigma X + \mu$ vaut σ^2

$$\begin{aligned}
Var(\sigma X + \mu) &= E[(\sigma X + \mu)^2] - E[\sigma X + \mu]^2 \\
Var(\sigma X + \mu) &= E[\sigma^2 X^2 + 2\sigma\mu X + \mu^2] - (\sigma E[X] + \mu)^2 \\
Var(\sigma X + \mu) &= \sigma^2 E[X^2] + 2\sigma\mu E[X] + \mu^2 - (\sigma^2 E[X]^2 + 2\sigma\mu E[X] \\
Var(\sigma X + \mu) &= \sigma^2 + 2\sigma\mu \times 0 + \mu^2 - (\sigma^2 \times 0 + 2\sigma\mu \times 0 + \mu^2) \\
Var(\sigma X + \mu) &= \sigma^2
\end{aligned}$$

On peut aussi visualiser une approximation empirique de la distribution de probabilité. Une telle approximation s'appelle un histogramme.

```
y=53+randn(1,1000);
[N,X]=hist(y,round(sqrt(length(y))));
figure(1); plot(X,N/sum(N));
```

D Questions diverses

Q 1 *Une des fenêtres de l'environnement de développement Matlab a disparu, comment la retrouver ?*

Matlab:La sélection successive des menus déroulants *Desktop*, *Desktop Layout*, *Default* permet de retrouver l'apparence initiale.
Octave:Sélectionner `Fenêtre` puis `Rétablir la disposition des fenêtres par défaut` permet de retrouver l'apparence initiale.

Q 2 *Une instruction semble fonctionner différemment de l'aide en ligne ou de l'aide sur internet.*

La fonction a pu être modifiée du fiat d'une version ultérieure de Matlab. Une explication plus fréquente est que rien n'interdit d'utiliser une fonction comme une variable, mais dans ce cas la fonction ne peut plus être utilisé en tant que fonction. L'instruction suivante

```
help=0;
```

fait que la fonction `help` ne fonctionne plus. On peut détecter ce phénomène en activant `class help` qui normalement donne `char` et ici donne `double`. La solution consiste à exécuter l'instruction suivante

```
clear help
```

E Exemples de messages d'erreurs et causes possibles de ces erreurs

`Error: unexpected Matlab operator`

Un opérateur n'a pas été reconnu, par exemple .+

Une cause possible est que le nom d'un script Matlab appelé commence par un chiffre.

`Error using plot string argument is an unknown option`

Une possibilité est que l'option `Linewidth` est présent dans les arguments de plot avant un des signaux. En fait cette option n'est

pas spécifique à un signal mais concerne tous les signaux à la fois et doit donc se trouver après tous les signaux parmi les options.

`Error using` suivi d'un nom de fonction

La fonction ainsi indiquée ne semble pas fonctionner.

`Undefined variable`

Cela peut signifier qu'il n'a pas reconnu dans cette séquence de lettre une variable ou ce qui peut-être aurait dû être une fonction.

`Undefined function or method` avec le nom d'une fonction `for input arguments of type double`

Cette fonction ne fonctionne pas.

`Error using ==> vertcat CAT argument are not consistent`

Lorsqu'on juxtapose les matrices, il est important que les tailles de ces matrices soient cohérentes : même nombre de ligne pour une juxtaposition horizontale et même nombre de colonne pour une juxtaposition verticale.

`Error: unbalanced or unexpected parenthesis or brackets`

Il manque probablement une parenthèse ouvrante ou fermante.

`Error: the expression to the left of the equal sign is not a valid target for an assignment`

Le signe égal est compris par matlab comme une affectation vers une variable et il constate qu'il n'est pas possible d'écrire sur cette variable.

`Error : index exceeds matrix dimensions`

L'instruction amène à lire la matrice sur un indice pour lequel elle n'a pas encore été affectée.

`Error: a matlab string is not terminated prope`

Il manque une deuxième apostrophe à la chaîne de caractère.

F Correction de l'exercice 1

1. `u=2:2:30;`
2. En généralisant l'idée de trouver les valeurs de $u(1), u(3), \ldots$, on pourrait se dire que la réponse est

   ```
   v=[u(1), u(3), u(5),u(7), u(9),u(11),u(13),u(15)];
   ```

   Il se trouve que Matlab permet d'accéder aux valeurs données par une liste d'indexes :

   ```
   v=u([1 3 5 7 9 11 13 15]);
   ```

   Ces indices peuvent être générés avec l'opérateur d'énumération :

```
v=u(1:2:15);
```

La valeur 15 est la longueur du vecteur `u` :

```
v=u(1:2:length(u));
```

On peut en fait utiliser l'opérateur `end` :

```
v=u(1:2:end);
```

3. La première étape est de définir un vecteur ayant la bonne taille :

   ```
   w=zeros(1,2*length(v));
   ```

   On remplit les indices impaires :

   ```
   w(1:2:end)=v.^2;
   ```

   puis les indices paires :

   ```
   w(2:2:end)=v;
   ```

4. `x=10.^(0:20);`

G Rendre visible les extensions de fichiers sous Windows 10

Il arrive parfois qu'on n'arrive pas à lire ou à écrire sur un fichier parce qu'en fait ce fichier a deux extensions, par exemple `data.mat.mat`. Aussi le fait de rendre visible les extensions de fichiers sous Windows 10 permet de résoudre certains problèmes. Il est souvent possible d'avoir le nom exact du fichier soit avec propriété soit en listant les fichiers d'un répertoire avec une commande `dir`.

Voici une procédure permettant de rendre visibles ces extensions.

— Ouvrez un fenêtre associée à un répertoire (appelé aussi Explorateur de fichier).
— Cliquez sur l'onglet Affichage.
— Cliquez sur la partie supérieure de l'option des dossiers.
— Cliquez sur Affichage
— Décochez la case associée à "Masquer les extensions des fichiers dont le type est connu".