Initiation à Matlab (et octave)

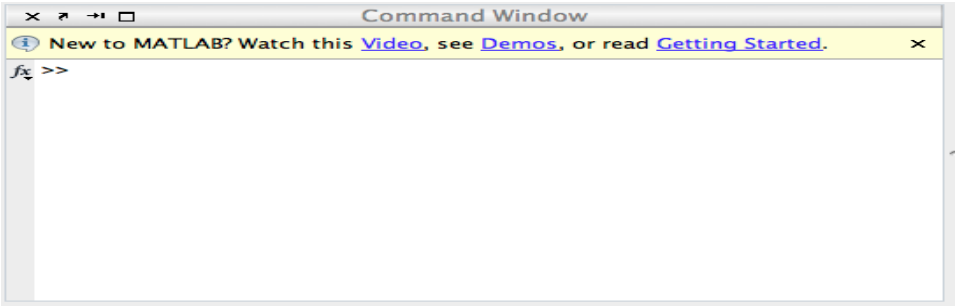
Si vous n'avez pas **Matlab**, vous pouvez utiliser **octave**
→ **octave est compatible avec Matlab et libre/open-source**

Notions de base

Il est recommandé de créer un répertoire de travail (par exemple TPsRCP208) pour enregistrer vos fichiers dans ce répertoire.

- Pour démarrer matlab :
  - Dans l'environnement unix, ouvrir une fenêtre linux et taper la commande : matlab &
    - Le caractère & permet d'exécuter cette commande en arrière plan (le caractère d'invite du terminal réapparaît immédiatement, permettant de lancer d'autres commandes à partir du même terminal).
    - Cette commande (matlab) va entrainer l'ouverture d'une interface Matlab.
  - Dans les environnements Windows ou MacOs, il suffit de cliquer sur l'icône de l'application.
- L'interface Matlab se compose d'une fenêtre principale divisée en sous-fenêtres.

  Il y a une grande fenêtre : **Command Window** (fenêtre de commande)



  - C'est la fenêtre d'interaction avec Matlab.
  - Toute commande tapée dans cette fenêtre après l'invite >> est immédiatement exécutée (après la frappe de la touche *return*).
  - Le caractère >> signifie que Matlab attend vos instructions.

  Pour indiquer à Matlab que le répertoire de travail est TPsRCP208 défini précédemment, cliquer sur ( ou ) en haut de la fenêtre principale de Matlab puis sélectionner ce répertoire.
- On peut utiliser Matlab de deux façons différentes.
  - La première est le mode en ligne : l'utilisateur exécute ses commandes une à une dans la fenêtre de commande.
  - La seconde consiste en l'exécution d'un ensemble de commandes stockées dans un M-fichier (un fichier dont le nom porte le suffixe <<**.m**>>).
- Matlab est particulièrement performant pour le calcul matriciel. En fait, toute variable de Matlab est une matrice :
  - un scalaire est une matrice de dimension $(1, 1)$.
  - un vecteur colonne est une matrice de dimension $(n, 1)$.
  - un vecteur ligne est une matrice de dimension $(1, n)$
- Placez-vous dans la fenêtre de commande.

Tapez les commandes suivantes une à une (après l'invite >>) et observez les résultats obtenus :
Je vais utiliser ⟶ bleu : pour les commandes tapées (après l'invite >>)
⟶ noir : pour les résultats obtenus

- Les éléments d'un vecteur ligne sont séparés par un espace ou une virgule.

```
>> v1=[ 1 2 3 4 5 6 ]
v1 =

     1     2     3     4     5     6

>> v2=[ 1,2,3,4,5,6 ]
v2 =

     1     2     3     4     5     6
```

- Lorsqu'on termine la commande par un point virgule, le résultat n'est pas affiché.

```
>> n=6;
```

- Pour afficher le contenu d'une variable entrez simplement son nom.

```
>> n
n =

     6
```

- Matlab fait la distinction entre majuscules et minuscules ($n \neq N$)

```
>> N
??? Undefined function or variable 'N'.
```

- Le double point est l'opérateur d'incrémentation. Le résultat de $x{=}n_1{:}p{:}n_2$ est un vecteur ligne contenant les valeurs de n_1 à n_2 par incrément de p (le premier élément est n_1). Par défaut, l'incrément est de 1.

```
>> v3= 1:1:6
v3 =

     1     2     3     4     5     6

>> v4= 1:1:6.5
v4 =

     1     2     3     4     5     6

>> v5= 1:n
v5 =

     1     2     3     4     5     6
```

- Les éléments d'un vecteur colonne sont séparés par un point virgule.

```
>> v6=[ 1;2;3;4;5;6 ]
v6 =

     1
     2
     3
     4
     5
     6
```

- On peut transposer un vecteur avec l'apostrophe.

```
>> v7=[ 1 2 3 4 5 6 ]'
v7 =

     1
     2
     3
     4
     5
     6

>> v8=v1'
```

```
v8 =

     1
     2
     3
     4
     5
     6
```

>> v9=v8';

– length donne le nombre d'éléments d'un vecteur.

```
>> length(v8)
ans =

     6
```

"ans" (answer) est une variable permanente de Matlab, qui prend systématiquement la valeur d'une expression affichée, si celle-ci n'est pas affectée à une variable. "ans" se substitue en quelque sorte à cette variable d'affectation manquante.

```
>> length(v9)
ans =

     6
```

– La commande who permet de lister les variables définies à un instant donné.

```
>> who
Your variables are:

ans  n    v1   v2   v3   v4   v5   v6   v7   v8   v9
```

– La commande whos donne des informations suplémentaires concernant ces variables.

```
>> whos
 Name        Size              Bytes  Class     Attributes

 ans         1x1                   8  double
 n           1x1                   8  double
 v1          1x6                  48  double
 v2          1x6                  48  double
 v3          1x6                  48  double
 v4          1x6                  48  double
 v5          1x6                  48  double
 v6          6x1                  48  double
 v7          6x1                  48  double
 v8          6x1                  48  double
 v9          1x6                  48  double
```

– On peut effacer certaines variables avec la commande clear suivie des noms de ces variables.

```
>> clear v1 v2
>> who
Your variables are:

ans  n    v3   v4   v5   v6   v7   v8   v9
```

– Pour effacer toutes les variables, clear all ou clear.

```
>> clear all
>> who
```

– Les lignes d'une matrice sont séparées entre elles par un point virgule

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

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

– On peut transposer une matrice avec l'apostrophe.

```
>> B = A'
B =

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

– size donne les deux dimensions d'une matrice.

```
>> size(A)
ans =

     4     3

>> [d1 d2] = size(A);
>> d1
d1 =

     4

>> d2
d2 =

     3
```

– le signe de pourcentage permet de mettre ce qui suit sur une ligne en commentaire.

```
>> % nombre de lignes
>> nbrelignes=size(A,1)
nbrelignes =

     4

>> nbrecolonnes=size(A,2) % nombre de colonnes
nbrecolonnes =

     3
```

– Un élément quelconque d'une matrice est référencé par ses indices de ligne et de colonne.

```
>> a=A(1,2)
a =

     2
```

– En matlab, les éléments d'une matrice sont rangés par colonnes.

```
>> A(:)
ans =

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

>> b=A(5)
b =

     2
```

>> A(2,:) ⟶ donne tous les éléments de la deuxième ligne de A

```
ans =

     4     5     6
```

>> A(:,3) ⟶ donne tous les éléments de la troisième colonne de A

```
ans =

     3
     6
     9
    12
```

>> A(2:4,:) ⟶ donne les lignes 2 à 4

```
ans =

     4     5     6
     7     8     9
    10    11    12
```

>> A=[1 2 0;3 0 4;5 6 0];B=[1 1 1;2 2 2;3 3 3];c=10;

>> c^3 ⟶ élévation à la puissance 3 de c

```
ans =

        1000
```

>> A.^2 ⟶ élève à la puissance 2 chaque élément de A

```
ans =

     1     4     0
     9     0    16
    25    36     0
```

>> A+B ⟶ Addition

```
ans =

     2     3     1
     5     2     6
     8     9     3
```

>> A-B ⟶ Soustraction

```
ans =

     0     1    -1
     1    -2     2
     2     3    -3
```

>> c*A ⟶ Multiplication par un scalaire

```
ans =

    10    20     0
    30     0    40
    50    60     0
```

>> c+A ⟶ Ajout de c aux éléments de A

```
ans =

    11    12    10
    13    10    14
    15    16    10
```

>> A*B ⟶ Produit matriciel standard

```
ans =

     5     5     5
    15    15    15
    17    17    17
```

>> A.*B ⟶ Multiplication terme à terme
⟶ tableau dont les éléments ont pour valeur $a_{ij} * b_{ij}$

```
ans =

     1     2     0
     6     0     8
    15    18     0
```

>> A./B ⟶ Division terme à terme
⟶ tableau dont les éléments ont pour valeur a_{ij}/b_{ij}

```
ans =

    1.0000    2.0000         0
    1.5000         0    2.0000
    1.6667    2.0000         0
```

>> det(A) ⟶ déterminant de A (d'une matrice carrée)

```
ans =

   16.0000
```

>> inv(A) ⟶ inversion de A (d'une matrice carrée (det(A)≠0))

```
ans =

   -1.5000         0    0.5000
    1.2500         0   -0.2500
    1.1250    0.2500   -0.3750
```

>> B*inv(A)

```
ans =

    0.8750    0.2500   -0.1250
    1.7500    0.5000   -0.2500
    2.6250    0.7500   -0.3750
```

>> X=B/A ⟶ Donne la solution de X*A=B
⟶ effectue B*inv(A)

```
X =

    0.8750    0.2500   -0.1250
    1.7500    0.5000   -0.2500
    2.6250    0.7500   -0.3750
```

>> inv(A)*B

```
ans =

         0         0         0
    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000
```

>> X=A\B ⟶ Donne la solution de A*X=B
⟶ effectue inv(A)*B

```
X =

         0         0         0
    0.5000    0.5000    0.5000
    0.5000    0.5000    0.5000
```

– Matlab dispose de deux constantes particulières :
inf (pour infinity) et NaN (pour Not a Number)

>> det(B)

```
ans =

     0
```

>> D=inv(B)

```
Warning: Matrix is singular to working precision.

D =

   Inf   Inf   Inf
   Inf   Inf   Inf
   Inf   Inf   Inf
```

>> D*7

```
ans =

   Inf   Inf   Inf
   Inf   Inf   Inf
   Inf   Inf   Inf
```

>> a=1/0

```
a =

   Inf
```

```
>> b=0/0
b =

   NaN
```

– On peut tester si une variable est finie, non finie, ou indéterminée en utilisant les fonctions : isfinite, isinf et isnan

```
>> isinf(a)
ans =

     1
```

```
>> isfinite(a)
ans =

     0
```

```
>> isnan(a)
ans =

     0
```

```
>> isnan(b)
ans =

     1
```

>> clear all ⟶ effacer toutes les variables

>> A1 = ones(3,4) ⟶ matrice 3x4 remplie de 1.

```
A1 =

     1     1     1     1
     1     1     1     1
     1     1     1     1
```

>> A2=zeros(4,5) ⟶ matrice 4x5 remplie de 0.

```
A2 =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

>> A3=eye(4) ⟶ matrice identité

```
A3 =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

– linspace(a,b,n) ⟶ crée un vecteur ligne de n éléments régulièrement espacés sur l'intervalle [a b] (bornes comprises).

>> V1=linspace(1,10,5) ⟶ vecteur de 5 éléments régulièrement espacés sur [1 10]

```
V1 =

    1.0000    3.2500    5.5000    7.7500   10.0000
```

– Si on dépasse la taille d'une matrice, Matlab remplit avec des 0 les cases où aucune valeur n'est spécifiée.

```
>> A=[1 2; 3 4]
A =

     1     2
     3     4
```

>> A(3,4)=9

```
A =

     1     2     0     0
     3     4     0     0
     0     0     0     9
```

>> A=[1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]

```
A =

     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

>> V=diag(A) ⟶ extrait la diagonale d'une matrice sous forme d'un vecteur

```
V =

     1
     6
    11
    16
```

>> diag(V) ⟶ crée une matrice diagonale à partir d'un vecteur

```
ans =

     1     0     0     0
     0     6     0     0
     0     0    11     0
     0     0     0    16
```

>> B=reshape(A,2,8) ⟶ permet de réorganiser une matrice

```
B =

     1     9     2    10     3    11     4    12
     5    13     6    14     7    15     8    16
```

Soit A une matrice de dimensions m et n
- Si $mxn = pxq$ alors reshape(A,p,q) donne une matrice de dimensions p et q
- Si $mxn \neq pxq$ alors reshape(A,p,q) donne une erreur

>> C=flipud(A) ⟶ retournement vertical de A

```
C =

    13    14    15    16
     9    10    11    12
     5     6     7     8
     1     2     3     4
```

>> fliplr(A) ⟶ retournement horizontal de A

```
ans =

     4     3     2     1
     8     7     6     5
    12    11    10     9
    16    15    14    13
```

>> rot90(A) ⟶ rotation de A de 90 degrés

```
ans =

     4     8    12    16
     3     7    11    15
     2     6    10    14
     1     5     9    13
```

– **help** suivie d'un nom de commande permet d'otenir des informations sur l'utilisation de la commande.

>> A=rand(4,4)

```
A =

    0.8147    0.6324    0.9575    0.9572
    0.9058    0.0975    0.9649    0.4854
    0.1270    0.2785    0.1576    0.8003
    0.9134    0.5469    0.9706    0.1419
```

>> help rand

```
>> help rand
 RAND Uniformly distributed pseudorandom numbers.
    R = RAND(N) returns an N-by-N matrix containing pseudorandom values drawn
    from the standard uniform distribution on the open interval(0,1).  RAND(M,N)
    or RAND([M,N]) returns an M-by-N matrix.  RAND(M,N,P,...) or
    RAND([M,N,P,...]) returns an M-by-N-by-P-by-... array.  RAND returns a
    scalar.  RAND(SIZE(A)) returns an array the same size as A.
  etc ...
```

>> D=randn(4,1)

```
D =

   -0.1241
    1.4897
    1.4090
    1.4172
```

>> help randn

```
>> help randn
 RANDN Normally distributed pseudorandom numbers.
    R = RANDN(N) returns an N-by-N matrix containing pseudorandom values drawn
    from the standard normal distribution.  RANDN(M,N) or RANDN([M,N]) returns
    an M-by-N matrix. RANDN(M,N,P,...) or RANDN([M,N,P,...]) returns an
    M-by-N-by-P-by-... array. RANDN returns a scalar.  RANDN(SIZE(A)) returns
    an array the same size as A.
  etc ...
```

- Utiliser la commande help sur les fonctions suivantes :
  exp, log, abs, sqrt, round, floor et sign
- **help** (sans nom de commande) : donne la liste de toutes les commandes par thèmes
  >> help
- Tapez la commande **helpwin** (elle ouvre la fenêtre d'aide Matlab).
  Sélectionnez, par exemple, le répertoire **ops** et observez son contenu.

- La commande pour quitter Matlab **à la fin de la séance** est : quit

Fichiers SCRIPT et FUNCTION

Lorsque la suite des instructions à exécuter devient importante, il vaut mieux les écrire dans un fichier texte et le sauvegarder sous un nom suivi de l'extension .m (l'extension .m est obligatoire).

- On distingue deux types de fichiers : les fichiers **SCRIPT** et **FUNCTION** (fonction).
  - Le fichier SCRIPT permet de lancer les même instructions que celles écrites directement à l'invite MATLAB. Ce fichier peut être aisément modifier en utilisant un éditeur de texte.

    Taper les lignes suivantes dans la fenêtre d'un éditeur de texte (de votre choix) :

    ```
    % script test1.m
    A=[1 2 0;3 0 4;5 6 0]
    B=[1 1 1;2 2 2;3 3 3]
    C=A+B
    ```

    Sauver (sauvegarder) le fichier dans le répertoire de travail (TPsRCP208, par exemple) sous le nom de script1.m
    Placez-vous dans la fenêtre de commande et tapez la commande script1 (sans le .m)

```
>> script1      ⟶ sans le .m
A =

     1     2     0
     3     0     4
     5     6     0


B =

     1     1     1
     2     2     2
     3     3     3


C =

     2     3     1
     5     2     6
     8     9     3
```

– Les fichiers de type FUNCTION permettent d'étendre les possibilités de Matlab en vous autorisant à créer vos propres fonctions.

function résultat = nom_fonction (liste de paramètres)

* Les variables locales d'une fonction ne sont pas disponibles à l'invite Matlab. Chaque fonction possède son propre espace de travail et toute variable apparaissant dans le corps d'une fonction est locale à celle-ci, à moins qu'elle ait été déclarée comme globale

global <<nom de la variable>>

dans tous les espaces de travail où cette variable est utilisée.

* **Le nom d'une fonction doit être le même que le nom du fichier texte (M-fichier) dans lequel elle est stockée.**
* **Il ne faut jamais désigner une variable par le même nom qu'un fichier Matlab (M-fichier : script ou function).**
* Un M-fichier (script ou fonction) peut être crée en utilisant n'importe quel éditeur de texte.
* Dans les versions récentes de Matlab il existe un petit éditeur intégré que l'on peut appeler à partir du menu **file**.

- Créez un fichier **sommeproduit.m** contenant les instructions suivantes :

Fichier texte **sommeproduit.m**

```
function [somme produit] = sommeproduit(M)
% somme et produit d'une matrice
somme=sum(M);
produit=prod(M);
```

- Le nom de la fonction doit être le même que le nom du fichier texte dans lequel elle est stockée (voir les noms en rouge).
- Placez-vous dans la fenêtre de commande et tapez les commandes suivantes :

```
>> W=[1 2 3 4]
>> [s p]=sommeproduit(W)
s =
    10

p =
    24

>> A=[1 2 0;3 0 4;5 6 0]
>> [s p]=sommeproduit(A)
```

```
s =

     9     8     4

p =

    15     0     0
```

- Certaines fonctions sont plutôt destinées à agir sur des vecteurs (lignes ou colonnes). Elles agissent aussi sur les matrices, mais colonne par colonne.

  Dans notre fonction sommeproduit

  - somme=sum(M);
    - Si M est un vecteur, elle retourne la somme des éléments de ce vecteur.
    - Si M est une matrice, elle retourne une liste dont chacun des éléments est la somme des éléments de chaque colonne.
  - produit= prod(M)
    - Si M est un vecteur, elle retourne le produit des éléments de ce vecteur.
    - Si M est une matrice, elle retourne une liste dont chacun des éléments est le produit des éléments de chaque colonne.

```
>> A
A =

     1     2     0
     3     0     4
     5     6     0
```

>> max(A) ⟶ renvoie une liste contenant la valeur maximale de chaque colonne

```
ans =

     5     6     4
```

>> min(A) ⟶ renvoie une liste contenant la valeur minimale de chaque colonne
>> mean(A) ⟶ renvoie une liste contenant la moyenne des éléments de chaque colonne

```
ans =

    3.0000    2.6667    1.3333
```

>> std(A) ⟶ écart type

```
ans =

    2.0000    3.0551    2.3094
```

>> median(A) ⟶ médiane

>> W

```
W =

     1     2     3     4
```

>> res=cumsum(W) ⟶ renvoie une liste contenant la somme cumulée des éléments de W

```
res =

     1     3     6    10
```

→ Contenu de la variable res

res(1)=W(1)=1
res(2)=W(1)+W(2) =1+2=3
res(3)=W(1)+W(2)+W(3)=1+2+3=6
res(4)=W(1)+W(2)+W(3)+W(4) =1+2+3+4=10

>> cumsum(A) ⟶ cumsum colonne par colonne

```
ans =

     1     2     0
     4     2     4
     9     8     4
```

>> res=cumprod(W) ⟶ renvoie une liste contenant le produit cumulé des éléments de W

```
res =

     1     2     6    24
```

→ res(1)=W(1)=1
res(2)=W(1)xW(2) =1x2=2
res(3)=W(1)xW(2)xW(3)=1x2x3=6
res(4)=W(1)xW(2)xW(3)xW(4) =1x2x3x4=24

>> V=rand(1,7)

```
V =

    0.6991    0.8909    0.9593    0.5472    0.1386    0.1493    0.2575
```

>> V=sort(V) ⟶ ordonne les éléments de V par ordre croissant

```
V =

    0.1386    0.1493    0.2575    0.5472    0.6991    0.8909    0.9593
```

>> V=rand(4)

```
V =

    0.1299    0.3371    0.5285    0.6541
    0.5688    0.1622    0.1656    0.6892
    0.4694    0.7943    0.6020    0.7482
    0.0119    0.3112    0.2630    0.4505
```

>> sort(V,1) ⟶ ordonne les éléments de chaque colonne

```
ans =

    0.0119    0.1622    0.1656    0.4505
    0.1299    0.3112    0.2630    0.6541
    0.4694    0.3371    0.5285    0.6892
    0.5688    0.7943    0.6020    0.7482
```

>> sort(V,2) ⟶ ordonne les éléments de chaque ligne

```
ans =

    0.1299    0.3371    0.5285    0.6541
    0.1622    0.1656    0.5688    0.6892
    0.4694    0.6020    0.7482    0.7943
    0.0119    0.2630    0.3112    0.4505
```

Boucles et tests

>> help for

>> help if

>> help while

```
>> help switch
```

- La manipulation matricielle est bien plus rapide que l'exécution de boucles

Exemple 1

- Créez un fichier test1.m contenant les instructions suivantes :

```
i=1 ;
for j=0:0.01:200
   y(i)=sin(j);
   i=i+1;
end
```

- Créez un fichier test2.m contenant les instructions suivantes :

```
j=0:0.01:200;
y=sin(j);
```

- Ces 2 programmes construisent le même vecteur. On peut comparer leur temps d'exécution.
  Placez-vous dans la fenêtre de commande et tapez les commandes suivantes :

```
>> tic;test1;tempstest1=toc
>> tic;test2;tempstest2=toc
```

  ↪ La commande tic déclenche un chronomètre
  ↪ La commande toc arrête le chronomètre et retourne le temps écoulé depuis tic

Exemple 2

- Créez un fichier test3.m contenant les instructions suivantes :

```
n=6;
for i=1:n
    for j=1:n
       if i==j
          S(i,j)=2;
       elseif abs(i-j)==1
          S(i,j)=1;
       else
          S(i,j)=0;
       end
    end
end
```

Placez-vous dans la fenêtre de commande et tapez les commandes suivantes :

```
>> test3
>> S
S =

     2     1     0     0     0     0
     1     2     1     0     0     0
     0     1     2     1     0     0
     0     0     1     2     1     0
     0     0     0     1     2     1
     0     0     0     0     1     2
```

→ S est une matrice tridiagonale.

Quelle est son temps d'exécution pour n=2000 ?
(**n'essayez pas avec n très grand : complexité en n^2**)

- En utilisant la commande **help**, expliquez ce que donne chacune des instructions suivantes:

```
>> n=6;
>> T=2*eye(n)+diag(ones(n-1,1),1)+diag(ones(n-1,1),-1)
>> T=2*eye(6)+diag(diag(ones(5)),1)+diag(diag(ones(5)),-1)

T =

     2     1     0     0     0     0
     1     2     1     0     0     0
     0     1     2     1     0     0
     0     0     1     2     1     0
     0     0     0     1     2     1
     0     0     0     0     1     2
```

→ T est aussi une matrice tridiagonale

Quelle est son temps d'exécution pour n=2000 ? (comparez avec celui de **test3**)

```
>> isequal(S,T)
ans =

     1

>> S==T
ans =

     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1
     1     1     1     1     1     1

>> S~=T
ans =

     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     0     0     0     0     0
     0     0     0     0     0     0

>> A=[1 2 0;3 0 4;5 6 0]
>> B=[1 1 1;2 2 2;3 3 3]
>> isequal(A,B)
>> A==B
>> A~=B
>> T1=[A B];
>> T2=[A;B];
>> W=[1 2 3 4]
>> repmat(W,3,1)
>> C=ceil(9*rand(8,5))
```

  * ceil : arrondit vers l'entier immédiatement au-dessus ⟶ ceil(9.8)=10 et ceil(9.3)=10
  * floor : arrondit vers l'entier immédiatement au-dessous ⟶ floor(9.8)=9 et floor(9.3)=9
  * round : arrondit vers l'entier le plus près ⟶ round(9.8)=10 et round(9.3)=9

```
>> [i,j]=find(C==5)
>> k=find(C==5)
>> a=6;
>> disp(['le carrée de ' num2str(a) ' est : ' num2str(a*a)])
```

Fichiers de sauvegarde

```
>> clear all
>> A=eye(3)
>> b=2
>> V=[1,2,3]
```

>> save S1 A b V ⟶ les variables A, b et V seront sauvegardées dans un fichier S1.mat

- save nomdefichier nomdevariable ··· nomdevariable

```
>> clear all
>> whos
```

→ Pour retrouver les variables A, b et V, il suffit de taper load S1

```
>> load S1
>> whos
```

Les tableaux de cellules

La classe cell modélise les tableaux de cellules qui sont des sortes de tableaux dont les éléments (cellules) peuvent être de type différent.

```
>> D=cell(3,2)
D =

    []    []
    []    []
    []    []
>> D{3}='TOTO';
>> D{5}=9;
>> D{1}=1:5;
>> D
D =

    [1x5 double]     []
              []    [9]
    'TOTO'           []

>> D{3}
ans =

TOTO

>> D{1}
ans =

     1     2     3     4     5

>> A=[1 2 0;3 0 4;5 6 0]
>> E={A sum(A) sum(sum(A)) cumsum(A)}
E =

    [3x3 double]    [1x3 double]    [21]    [3x3 double]

>> E{2}
ans =

     9     8     4
```

→ help cell pour plus d'informations

Les structures

```
>> F.nom='TOTO';
>> F.age=25;
>> F.profession='Medecin';
>> F
F =

            nom: 'TOTO'
            age: 25
     profession: 'Medecin'

>> F(2).nom='TOTO2';
>> F(2).age='30';
>> F(2).profession='Informaticien';
>> F
F =

1x2 struct array with fields:
    nom
    age
    profession

>> F(2)
ans =

          nom: 'TOTO2'
          age: '30'
    profession: 'Informaticien'

>> F(3)=struct('nom','TOTO3','age',50,'profession','chimiste')
F =

1x3 struct array with fields:
    nom
    age
    profession

>> F(3)
ans =

            nom: 'TOTO3'
            age: 50
     profession: 'chimiste'

>> F(3).profession
ans =

chimiste
```

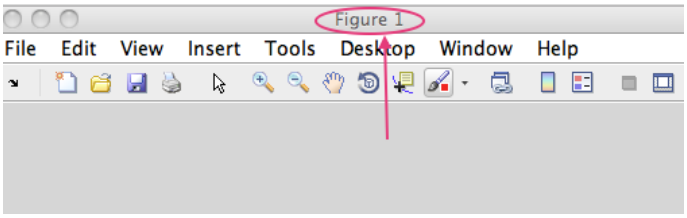
→ help struct pour plus d'informations

Entrées et sorties graphiques

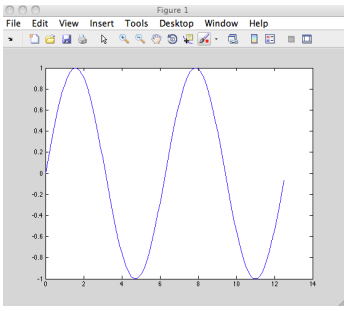
Exemple 1

```
>> figure
```

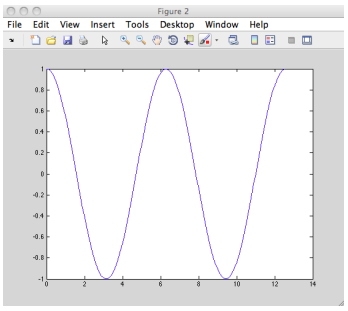
⟶ une fenêtre appelée Figure 1 apparaît



>> x=0:0.1:4*pi;
>> y=sin(x);
>>plot(x,y) ⟶ trace dans la fenêtre Figure 1 le graphe de la fonction sin(x) sur l'intervalle $[0,4\pi]$

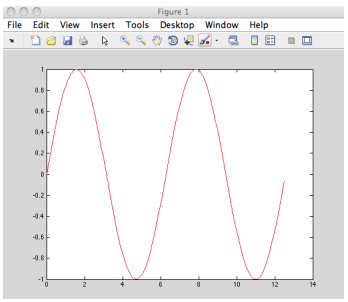


>>figure ⟶ une fenêtre appelée Figure 2 apparaît
>>plot(x,cos(x)) ⟶ trace dans la fenêtre Figure 2 le graphe de la fonction cos(x) sur l'intervalle $[0,4\pi]$

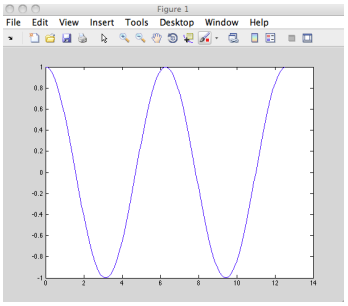


Exemple 2

>> close all ⟶ on ferme toutes les fenêtres graphiques : Figure 1 et Figure 2
>> figure ⟶ une fenêtre appelée Figure 1 apparaît
>> x=0:0.1:4*pi;
>> plot(x,sin(x),'r')
→ trace dans la fenêtre Figure 1 le graphe de la fonction sin(x) sur l'intervalle $[0,4\pi]$ en utilisant la couleur rouge (car 'r')

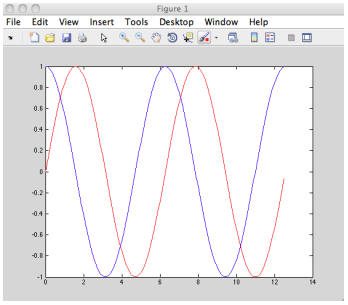


>> plot(x,cos(x))
→ trace dans la fenêtre Figure 1 le graphe de la fonction cos(x) sur l'intervalle $[0,4\pi]$
→ le graphe de la fonction sin(x) a été supprimé de la fenêtre Figure 1



Exemple 3

>> close all
>> figure
>> x=0:0.1:4*pi;
>> plot(x,sin(x),'r')
>> hold on
>> plot(x,cos(x))

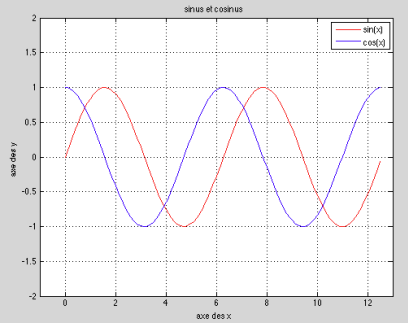


→ Les 2 graphes (sin(x) et cos(x)) apparaissent dans la même fenêtre
→ après hold on tous les tracés se superposent aux tracés déjà effectués
→ hold off fait revenir au mode de tracé normal
(le contenu de la fenêtre active sera effacé lors du prochain tracé).
→ Chaque appel à la fonction figure crée une nouvelle fenêtre
→ La dernière fenêtre crée est la fenêtre active
→ plot(x,y) trace dans la fenêtre active
→ On peut rendre active une fenêtre déjà crée en utilisant son numéro
>> close all
>> h=figure
→ une fenêtre appelée Figure 1 apparaît (c'est la fenêtre active)
→ h est le numéro de la fenêtre
>> figure
→ une fenêtre appelée Figure 2 apparaît (c'est la fenêtre active)
>> figure(h)
→ la fenêtre numéro h (donc Figure 1) devient la fenêtre active

Exemple 4

>> figure
>> x=0:0.1:4*pi;
>> plot(x,sin(x),'r')
>> hold on
>> plot(x,cos(x))
>> axis([-1 13 -2 2]) ⟶ pour modifier les limites des axes (faire help axis)
→ pour nommer les axes : xlabel et ylabel
>> xlabel('axe des x')
>> ylabel('axe des y')
>> title('sinus et cosinus') ⟶ donner un titre pour le graphique

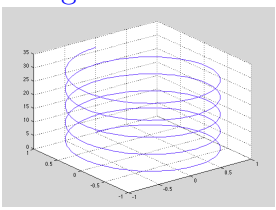
>> legend('sin(x)','cos(x)') ⟶ légende pour chaque courbe du graphique
>> grid on ⟶ superpose une grille au tracé (grid off supprime cette grille)



Exemple 5: visualisation en 3D

Visualisation des courbes

>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t); ⟶ faire help plot3
>> grid on



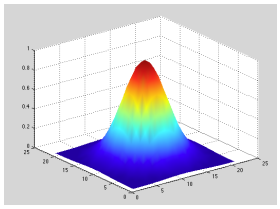
Visualisation des surfaces

>> x=1:3; y=4:9;
>> [X Y]=meshgrid(x,y)

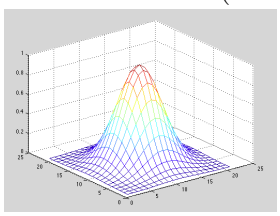
```
X =

     1     2     3
     1     2     3
     1     2     3
     1     2     3
     1     2     3
     1     2     3


Y =

     4     4     4
     5     5     5
     6     6     6
     7     7     7
     8     8     8
     9     9     9
```

→ X matrice contenant length(y) lignes identiques à x, donc X(i,j)=x(j)
→ Y matrice contenant length(x) colonnes identiques à y, donc Y(i,j)=y(i)
→ donc (X(i,j),Y(i,j)=(x(j),y(i))

>> x=-2:0.2:2; y=-2:.2:2;
>> [X Y]=meshgrid(x,y);
→ construction d'une grille modélisant le domaine $[-2,2]$x$[-2,2]$
>> $Z = exp(-X.\hat{}2 - Y.\hat{}2)$;
→ permet dévaluer les valeurs de **exp** suivant cette grille
>> figure
>> surf(Z) → affiche la surface $Z = f(x,y) = exp(-x^2 - y^2)$
>> shading interp → essayez aussi shading flat et shading faceted



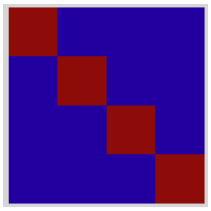
>> figure
>> mesh(Z) → idem que surf mais la surface est affiché <<en fil-de-fer>>
(surf affiche en surface remplie)



Exemple 6 Affichage de matrices
>> A=eye(4)

```
A =

     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

>> figure
>> imagesc(A) → dessine un rectangle partagé en size(A,1) x size(A,2) petits rectangles où la couleur du rectangle (i,j) dépend de la valeur $A(i, j)$.
>> axis equal
>> axis off



>> B=rand(5,5);

```
B =

    0.8147    0.0975    0.1576    0.1419    0.6557
    0.9058    0.2785    0.9706    0.4218    0.0357
    0.1270    0.5469    0.9572    0.9157    0.8491
    0.9134    0.9575    0.4854    0.7922    0.9340
    0.6324    0.9649    0.8003    0.9595    0.6787
```

>> figure
>> imagesc(B)
>> axis off