Introduction à Scilab

Scilab est un logiciel libre, disponible gratuitement sur le site `www.scilab.org`. La dernière version (celle utilisée au DLST sous Windows) est la 5.5.2.

Cette première séance de TP consiste à prendre en main le logiciel Scilab avec les principales fonctionnalités qui seront utilisées tout au long du semestre.

Suivez le déroulement de la fiche de TP, testez les différents exemples présentés. — Différents exercices sont proposés au cours de cette séance, entraînez-vous à les faire.

Un corrigé est proposé sur ALFRESCO :
`https://espaces-collaboratifs.grenet.fr/share/page/site/UJFueDDMAP110120/dashboard`
dans la partie *Espace documentaire* puis *Intro_SCILAB*
Ne vous y référez qu'une fois l'exercice terminé ou en cas de difficulté.

1 Démarrage de Scilab

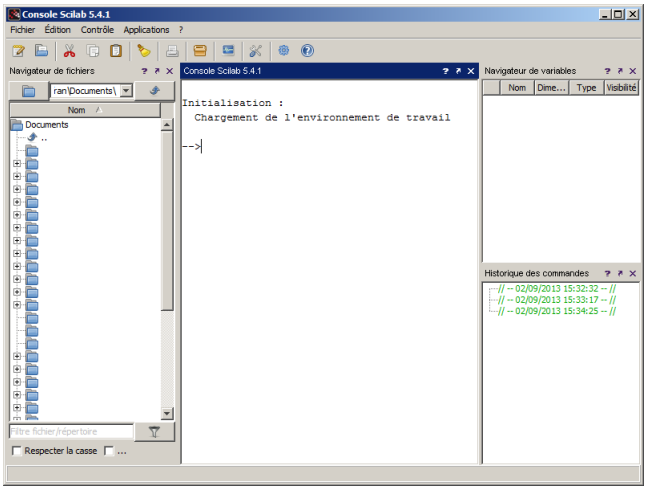
Au début de chaque séance de TP, il vous faut créer un répertoire où vous stockerez les différents fichiers utilisés lors de la séance.

Pour cette première séance du thème Courbes, ouvrez le `Poste de travail` puis le répertoire nommé `Z:` qui correspond à votre répertoire personnel sur lequel vous pouvez mettre vos propres documents et que vous retrouverez tel quel lors de chaque séance sur machine.

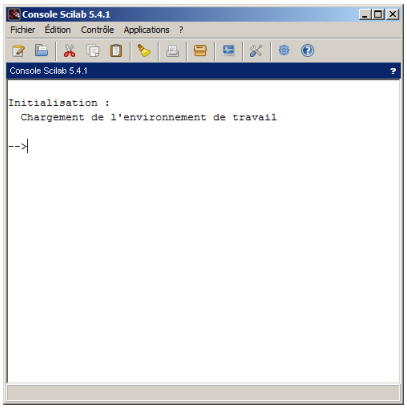
Dans ce répertoire personnel, créez un nouveau répertoire nommé `MAP110` ou `MAP120` puis dans celui-ci un nouveau répertoire nommé `COURBES` et enfin dans celui-ci un répertoire nommé `TP1`.
Au début de chaque séance suivante, il suffira de créer un nouveau répertoire à coté du répertoire `TP1`.

Démarrez SCILAB : double-cliquez sur l'icône correspondant ou utilisez le menu `Programmes` de Windows.

La première fois, la fenêtre Scilab doit s'afficher avec une *disposition intégrée* (voir la figure de gauche ci-dessous). Cette nouvelle disposition apparue avec la version 5.4 permet d'accèder à de nombreuses options de Scilab.



Disposition intégrée



Disposition simple

Pour notre part, nous allons plutôt travailler avec la *disposition simple* (voir la figure de droite ci-dessus).

- dans le menu *Edition*, choisissez l'item *Préférences*.
- dans le dialogue *Préférences de Scilab*, ouvrez l'onglet *Général*, selectionnez *Disposition du bureau*, choisissez l'option *Simple* à la place de l'option *Intégrée*, cliquez sur le bouton *Appliquer* puis sur le bouton *OK*
- quitter *Scilab* et redémarrez.

La fenêtre principale (appelée par la suite *console*) s'ouvre avec différents menus et icônes correspondants et la partie *console* où on peut entrer au clavier différentes commandes et où l'affichage de certains résultats est fait.

La première chose à faire est de "placer" *scilab* dans le répertoire que vous avez créé précédemment afin de stocker les différents fichiers de la séance : dans le nemu *Fichier*, sélectionnez l'item *Changer le répertoire courant ...* puis dans le dialogue, placez-vous dans le répertoire voulu et cliquez sur le bouton `Ok` .
On peut à tout moment changer de répertoire courant.

2 Travailler avec Scilab

2.1 Mode interactif

On peut utiliser SCILAB comme une calculette interactive, en mode console.
• Exemple : Dans la console, tapez les instructions suivantes :

```
2.5*4-7/2
exp(1)
(1+%i)*(2-3*%i)
3^2 , 4/3
```

⇨ les fonctions mathématiques usuelles sont définies
⇨ le calcul en nombre complexe est possible, la constante $i = \sqrt{-1}$ est notée `%i`. Par la suite, on utilisera aussi la constante $\pi \simeq 3,1416$ en utilisant la notation `%pi`
⇨ notez le role de la virgule (`,`)

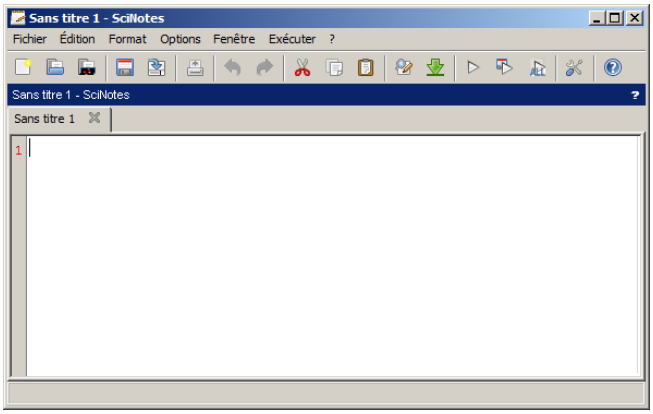
Il est possible d'utiliser des variables afin de stocker le résultat d'un calcul afin de pouvoir s'en servir ensuite.
• Exemple : Dans la console, tapez les instructions suivantes :

```
H = 2; R = 1.5; V = %pi * R*H
H, R, V
```

⇨ notez le rôle du point-virgule (`;`) par rapport à la virgule (`,`).
⇨ on peut éventuellement réafficher, modifier et réexécuter les différentes instructions entrées précédemment en utilisant les touches *flèches haut* ↑ et *bas* ↓ du clavier.

2.2 Utilisation de fichiers d'instructions

L'utilisation de la console peut devenir fastidieuse dès que le nombre d'instructions à écrire augmente. Il est alors préférable d'écrire les instructions dans un éditeur, pour ensuite les exécuter, les modifier et éventuellement les sauvegarder dans des fichiers qui pourront être réutilisés plus tard.



Pour utiliser l'éditeur, cliquez sur l'icône 
ou bien sélectionnez l'item *SciNotes* du menu *Application* :
l'éditeur de texte intégré de Scilab s'ouvre.

Dans la fenêtre de l'éditeur, tapez les instructions suivantes :

```
a = 2
b = 1
c = a/b
```

et sauvegardez-le (menu *Fichier*, item *Sauvegarder sous ...*) en le nommant `prog1.sce`.
⇨ par convention, les scripts Scilab contenant une suite d'instructions ont un nom avec le suffixe `.sce`

Il est alors possible d'exécuter une suite d'instructions en les sélectionnant à la souris dans l'éditeur puis en tapant simultanément les touches `Ctrl` et `E`,
ou bien d'exécuter l'ensemble du fichier en tapant simultanément les touches `Ctrl`, `Majuscule` et `E`.
• Exemple : Testez ces deux possibilités avec le fichier `prog1.sce`

⇨ remarquez que dans le cas de l'exécution de l'ensemble du fichier, rien n'est écrit à l'écran. Dans ce cas, la présence ou l'absence de points-virgules ne changent rien, il faut alors explicitement utiliser la procédure `disp` afin d'afficher une valeur.
• Exemple : Complétez le script `prog1.sce` en rajoutant l'instruction

```
disp(c)
```

à la fin du fichier, puis refaites une exécution complète : la valeur de `c` est écrite dans la console.

Une fois créé et sauvegardé, un fichier-script Scilab peut être réutilisé par la suite.
• Exemple : Fermez l'éditeur de texte, puis dans la console, tapez l'instruction

```
exec("prog1.sce")
```

et ensuite, tapez l'instruction

```
exec("prog1.sce", -1)
```

et notez la différence entre les deux.
⇨ on peut aussi utiliser l'item *Exécuter ...* du menu *Fichier* de la fenêtre *console*.

2.3 Espace de travail

Scilab stocke en mémoire les différentes variables que vous avez défini dans vos différentes instructions. L'instruction Scilab `clear` permet de supprimer certaines (ou la totalité) des variables que vous avez définies.
• Exemple : Dans la console, tapez les instuctions suivantes en observant le résultat pour chaque ligne d'instructions :

```
a=1; b=2; c=3;
a,b,c
clear c
a,b,c
clear
a
b
c
```

2.4 Aide en ligne

Pour avoir de l'aide sur les différentes fonctionnalités et instructions de Scilab, utilisez l'instruction `help` ou cliquez sur l'icône  .

Pour avoir l'aide sur une instruction particulière, utilisez l'onglet *loupe* du dialoque d'aide ou tapez l'instruction `help` *instruction*.

• Exemple : Pour avoir l'aide sur l'instruction `clear`, tapez l'instruction `help clear`

3 Vecteurs et matrices

Scilab effectue tous ses calculs en utilisant des nombres *réels flottants* qui permet entre autres de manipuler des nombres *entiers relatifs*.

L'un des intérets en calcul numérique est de pouvoir facilement et rapidement effectuer des calculs sur des tableaux de valeurs. Scilab permet de manipuler des variables de type *tableau* puis d'effectuer des opérations sur celles-ci.

3.1 Définition, création de vecteurs et matrices

Un *vecteur* est un tableau formé d'une seule ligne ou d'une seule colonne.

• Exemple : Avec l'éditeur de texte, créez un nouveau fichier nommé `tableaux.sce` avec les instructions suivantes :

```
v1 = [2 3 5 7 11 13]; // création d'un vecteur-ligne : séparer
v2 = [1,3,5];         // les valeurs par des espaces ou des virgules
w1 = [0;3;5;6];       // création d'un vecteur-colonne : séparer
                      // les valeurs par des points-virgules
disp(v1), disp(v2), disp(w1) // afficher les vecteurs
```

puis exécutez-les.

⇨ sur une ligne d'instruction, ce qui suit les deux symboles `//` est un commentaire.

Une *matrice* est un tableau avec un nombre quelconque de ligne et un nombre quelconque de colonne (un vecteur est une matrice particulière).

• Exemple : Complétez le fichier `tableaux.sce` en ajoutant les instructions suivantes :

```
A = [1 2;3 4;5 6;7 8]; // matrice avec 4 lignes et 2 colonnes
B = [4 6 -2;5 8 3];    // matrice avec 2 lignes et 3 colonnes
C = [1 2;0 3];         // matrice carrée avec 2 lignes et 2 colonnes
```

exécutez-les, puis dans la console Scilab, examinez le résultat des instructions suivantes :
`[v1 v1]` `[v1;v1]` `[w1 w1 w1]` `[w1;w1;w1]` `[v2 v1]` `[v1 ; v2 v2]` `[w1 , [v2;v2;v2;v2]]`
`[C C C]` `[C;C;C]` `[A;C]`
puis testez les instructions suivantes :
`[v1 w1]` `[v2;v1]` `[w1;v2]` `[A B]` `[A;B]` `[A C]`
⇨ on peut assembler des tableaux en un tableau plus grand uniquement si les dimensions concordent.

3.2 Accès aux éléments d'un tableau

Pour accèder à un élément d'un tableau, il suffit d'utiliser le ou les indices correspondants (chaque indice est un entier supérieur ou égal à 1) :

- pour un vecteur, chaque élement est repéré par un indice
  • Exemple : Observez les valeurs des instructions suivantes :

```
v1(1), v1(2) , v2(3) , w1(1) , w1(4)
```

- pour une matrice, chaque élement est repéré par deux indices, le premier est l'indice de ligne et le second est l'indice de colonne
  • Exemple : Observez le résultat des instructions suivantes :

```
A(1,1) , A(1,2) , A(2,1) , B(2,3) , C(1,2)
```

- on peut aussi extraire une ligne ou une colonne particulière d'une matrice.
  • Exemple : Observez le résultat des instructions suivantes :

```
A(1,:) , A(3,:) , A(:,2) , B(:,3) , C(1,:)
```

3.3 Création de tableaux particuliers

Scilab fournit différentes procédures afin de créer des tableaux particuliers.

- la procédure `zeros(m,n)` crée un tableau avec m lignes et n colonnes et dont toutes les valeurs sont égales à 0.
  • Exemple : Testez les instructions suivantes :

```
zeros(2,3) , zeros(3,5) , zeros(1,7) , zeros(3,1)
```

- la procédure `ones(m,n)` crée un tableau avec m lignes et n colonnes et dont toutes les valeurs sont égales à 1.
  • Exemple : Testez les instructions suivantes :

```
ones(2,3) , ones(3,5) , ones(1,6) , ones(4,1)
```

- la procédure `linspace(a,b,n)` crée un vecteur-ligne formé des n valeurs équiréparties entre `a` et `b`.
  • Exemple : Testez les instructions suivantes :

```
linspace(1,5,5) , linspace(0,4,9) , linspace(10,1,4)
```

  ⇨ on remarque que le *pas* (la différence entre deux valeurs consécutives) est donnée par la formule $\texttt{pas} = (\texttt{b} - \texttt{a})/(\texttt{n} - 1)$.
- si on souhaite construire un vecteur-ligne en spécifiant le *pas*, on utilisera alors la notation `a:pas:b`
  • Exemple : Testez les instructions suivantes : `1:1:5` `0:0.5:4` `10:-3:1` `1:5` `0:9`
  ⇨ si le *pas* n'est pas spécifié (`a:b`), il est égal à 1.

Exercice 1 :

en utilisant les instructions vues précédemment, écrire (si possible de la manière la plus simple) les instructions pour créer les tableaux suivants :

$$v1 = \begin{pmatrix} 5 & 6 & 7 & 8 & 9 & 10 \end{pmatrix} \quad v2 = \begin{pmatrix} 0 & 0 & 0 & 5 & 6 & 7 & 8 & 9 & 10 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$v3 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 9 & 7 & 5 & 3 & 1 \end{pmatrix}$$

$$M1 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 3 & 5 & 7 & 9 \\ 8 & 6 & 4 & 2 & 0 \\ 8 & 6 & 4 & 2 & 0 \end{pmatrix} \quad M3 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 \end{pmatrix}$$

3.4 Opérations arithmétiques sur les tableaux

Une fois un tableau défini, on peut réaliser certaines opérations arithmétiques sur l'ensemble des éléments d'un tableau ou entre tableaux de même dimensions.

• Exemple : Testez les instructions suivantes :

```
M = [1 2 3;4 5 6]
M+2 // ajoute 2 à tous les éléments de M
M-3 // retranche 3 à tous les éléments de M
M*(-4) // multiplie tous les éléments de M par -4
M/10 // divise tous les éléments de M par 10
```

On peut aussi effectuer des opérations arithmétiques terme à terme entre deux tableaux de même dimensions.

• Exemple : Testez les instructions suivantes :

```
M = [1 2 3;4 5 6] , P = [4 7 1;0 2 8]
M+P , P+M , M-P , P-M
```

Par contre si on souhaite multiplier ou diviser terme à terme les éléments de deux tableaux, il faudra utiliser les opérateurs `.*` et `./` au lieu de `*` et `/`

• Exemple : Testez les instructions suivantes :

```
M.*P , P.*M , P./M , 1 ./M   // les inverses des éléments de M
```

⇨ pour la dernière instruction, il faut nécessairement mettre un espace avant l'opérateur `./`

De même, on peut appliquer la puissance terme à terme en utilisant l'opérateur `.^`

• Exemple : Testez les instructions suivantes :

```
v1 = 0:9 // les 10 premiers entiers
v2 = v1 .^ 2 // les 10 premiers carrés
v3 = v1 .^ 3 // les 10 premiers cubes
w  = 2 .^ v1 // les 10 premieres puissances de 2
```

⇨ pour effectuer une opération entre deux tableaux, ils doivent nécessairement être de même dimensions.

• Exemple : Testez les instructions suivantes :

```
M = [1 2 3;4 5 6] , P = [1 1;1 1]
M+P
M.*P
```

3.5 Transposition

La transposition d'un tableau consiste à inverser le rôle des lignes et colonnes. Elle s'effectue avec le symbole ' (`apostrophe` ou `quote`)

• Exemple : Tapez les instructions suivantes :

```
v = 1:8 , w = [4;5;6;7] , M = [1 2 3;4 5 6]
```

puis l'instruction `v'` puis l'instruction `w'` puis l'instruction `M'`.

Exercice 2 :

en utilisant les instructions vues précédemment, écrire (si possible de la manière la plus simple) les instructions pour créer les tableaux suivants :

$$M1 = \begin{pmatrix} 1 & 3 & 5 & 7 \\ 2 & 4 & 6 & 8 \\ 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 10 \\ 5 & 7 & 9 & 11 \\ 6 & 8 & 10 & 12 \\ 7 & 9 & 11 & 13 \end{pmatrix} \quad M2 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 5 & 5 & -2 & -2 & -2 \\ 5 & 5 & -2 & -2 & -2 \end{pmatrix} \quad M3 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 \end{pmatrix}$$

3.6 Dimensions d'un tableau

A tout moment, il est possible de connaitre les dimensions d'un tableau avec les procédures `size` et `length`.

• Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles.

```
v = 1:8 , w = [4;5;6;7] , M = [1 2 3;4 5 6]
size(v) // donne les dimensions du tableau v : [nb_ligne nb_colonne]
size(w) // donne les dimensions du tableau w
size(M) // donne les dimensions du tableau M
size(M,1) // donne le nombre de lignes de M
size(M,2) // donne le nombre de colonnes de M
length(v) // donne le nombre de valeurs du tableau v
length(w) // donne le nombre de valeurs du tableau w
length(M) // donne le nombre de valeurs du tableau M
```

3.7 Chaîne de caractères

Scilab permet aussi de manipuler des chaînes de caractères (tableaux de caractères). Une chaîne de caractères est délimitée par des *quotes* (') ou par des *double-quotes* (") .

• Exemple : Tapez les instructions suivantes :

```
s1 = 'MAP', s2 = "110", s3 = " 120"
disp(s1+s2)
disp(s1+s3)
```

⇨ pour deux chaînes de caractères, l'opérateur + permet de les *concaténer*.

4 Fonctions

Scilab fournit un certain nombre de fonctions notamment les principales fonctions mathématiques (voir la partie *Elementary functions* de l'aide en ligne).
La plupart des fonctions peuvent s'appliquer aussi bien à une seule valeur qu'à un tableau de valeurs.
• Exemple : Testez les instructions suivantes :

```
sqrt(4)
k = 0:12
sqrt(k) // les valeurs [sqrt(0),sqrt(1),sqrt(2),...,sqrt(11),sqrt(12)]
t = k*%pi/12
cos(t) // les valeurs [cos(0),cos(pi/12),cos(2*pi/12),...,cos(pi)]
```

On peut à partir des opérateurs et fonctions de Scilab, créer ses propres fonctions.
• Exemple : Pour calculer les valeurs de la fonction $f(x) = \dfrac{1}{1+x^2}$ pour les valeurs $x = 0$, $x = 0,5$, $x = 1$, ..., $x = 9,5$ et $x = 10$, tapez les instructions suivantes :

```
x = 0:0.5:10
y = (1) ./ (1+x.^2)
```

On aimerait pouvoir définir la fonction $f(x)$ puis l'utiliser à l'aide de l'instruction `y=f(x)`.

Scilab permet à l'utilisateur de créer ses propres fonctions, et éventuellement de les sauvegarder sous forme de fichier pour les réutiliser ultérieurement.
• Exemple : Avec l'éditeur de texte, créez le fichier suivant en le nommant `ex_fct1.sce`

```
// definition de la fonction f
function y = f(x)
    y = (1) ./ (1+x.^2)
endfunction

//----- utilisation de la fonction f -----

// calcul de f(x) pour x=0  x=0,5  x=1  ... x=9,5  et  x=10
x = 0:0.5:10
y = f(x)
disp(y)

// calcul de f(t) pour 100 valeurs de t équiréparties entre -10 et 10
t = linspace(-10,10,100)
z = f(t)
disp(z)
```

puis exécutez le script `ex_fct1.sce`.
⇨ il est important d'utiliser les opérations terme à terme (`.*`, `./`, `.^`) pour pouvoir utiliser la fonction avec un tableau de valeurs.
⇨ une fois la fonction définie, on peut l'utiliser avec n'importe quelle variable (pas nécessairement avec des variables ayant les mêmes noms que dans la définition de la fonction).
⇨ dans le cas d'un tableau avec un grand nombre de valeurs, Scilab demande à l'utilisateur de continuer ou non l'affichage du tableau dans la console.

Une ou plusieurs fonctions peuvent être définies dans un fichier séparé puis être utilisées dans un script Scilab.
• Exemple : Avec l'éditeur de texte, créez le fichier suivant en le nommant `f.sci`

```
// definition de la fonction f
function y = f(x)
    y = (1) ./ (1+x.^2)
endfunction
```

puis créez le fichier suivant en le nommant `ex_fct2.sce`

```
// chargement du contenu du fichier nommé f.sci
exec("f.sci", -1);

//----- utilisation de la fonction f -----

// calcul de f(x) pour x=0  x=0,5  x=1  ... x=9,5  et  x=10
x = 0:0.5:10
y = f(x)
disp(y)

// calcul de f(t) pour 100 valeurs de t entre -10 et 10
t = linspace(-10,10,100)
z = f(t)
disp(z)
```

puis exécutez le script `ex_fct2.sce`.

⇨ par convention, les fichiers contenant uniquement des définitions de fonctions ont un nom avec le suffixe `.sci` alors que les scripts Scilab ont un nom avec le suffixe `.sce`

Exercice 3 :

écrire un fichier nommé `f3.sci` contenant la définition de la fonction $y = f_3(x) = \exp(\sqrt{x/10})$ puis écrire un script Scilab nommé `ex_fct3.sce` qui permet de calculer les deux vecteurs formés des valeurs

$$u = f_3(t) \times \cos(t) \text{ et } v = f_3(t) \times \sin(t)$$

pour 1000 valeurs de t entre 0 et 10.

5 Graphique

Scilab dispose de fonctionnalités afin de tracer des graphiques à partir de données sous forme de tableaux.

5.1 Représentation de données

Les procédures Scilab `plot` et `bar` permettent de représenter des données.

Pour illustrer ceci, prenons l'exemple des inscriptions de bacheliers dans les 3 universités de Grenoble entre 2005 et 2008 :

	Grenoble 1	Grenoble 2	Grenoble 3
2005	2949	2955	1129
2006	2919	2837	1097
2007	2786	2886	942
2008	2789	2792	817

Créez un vecteur-colonne contenant les années :

```
Annees = [2005 ; 2006 ; 2007 ; 2008]' // ou bien Annees = (2005:2008)'
```

et une matrice contenant les effectifs

```
Effectifs = [2949 2955 1129 ; 2919 2837 1097 ; ...
             2786 2886  942 ; 2789 2792  817 ]
```

On peut alors extraire de la matrice `Effectifs` une ligne ou une colonne particulière (pour obtenir un vecteur-ligne ou vecteur-colonne).

```
Grenoble1 = Effectifs(:,1)
Grenoble2 = Effectifs(:,2)
Annee2006 = Effectifs(2,:)
```

On peut alors représenter un vecteur sous forme d'un diagramme à barres :

```
figure();
bar(Grenoble1)
figure();
bar(Annee2006)
```

⇨ l'instruction `figure()` permet de créer une nouvelle fenêtre graphique.

Testez les instructions suivantes en observant le résultat après chacune d'elles :

```
xdel(winsid())          // supprime toutes les fenetres graphiques
figure();               // creer une nouvelle fenetre graphique
bar(Annees, Grenoble1) // les données de Grenoble1
                        // avec les années en abscisse
clf()                   // effacer la fenetre graphique courante
bar(Annees, Effectifs) // les données des 3 universités
                        // avec les années en abscisse
clf()                   // effacer la fenetre graphique courante
bar(Effectifs')         // les données des 4 années
```

⇨ pour plus d'info sur l'instruction `bar`, tapez l'instruction `help bar`.

L'instruction `plot` est équivalente à l'instruction `bar` mais en faisant une réprésentation point par point.

• Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles :

```
xdel(winsid()) // supprime toutes les fenetres graphiques

// représentation des données de Grenoble1 point par point
figure();
plot(Grenoble1, '.')

// représentation des données de Grenoble1 en reliant les pts entre eux
clf()
plot(Grenoble1, '-')

// representation des données de Grenoble1 et de Grenoble2
// avec les années en abscisse
clf()
```

```
plot(Annees, Grenoble1, 'k-') // en noir tracé continu
plot(Annees, Grenoble2, 'r--') // en rouge tracé pointillé
```

5.2 Représentation de points du plan

Si les deux premiers arguments de la procédure `plot` sont deux vecteurs alors le premier vecteur correspond à des abscisses et le second vecteur correspond à des ordonnées.
On peut alors représenter un ensemble de points (x_i, y_i), $1 \leq i \leq n$ du plan, en définissant un vecteur $\mathtt{x} = [x_1\ x_2\ \ldots\ x_n]$ et un vecteur $\mathtt{y} = [y_1\ y_2\ \ldots\ y_n]$, chaque vecteur contenant n valeurs.
• Exemple : Pour représenter les 4 points $(-1,-2)$, $(1,-2)$, $(1,2)$ et $(-1,2)$, d'abord définir les deux vecteurs pour les abscisses et les ordonnées :

```
x1 = [-1  1  1 -1]
y1 = [-2 -2  2  2]
```

puis effectuer le tracé :

```
figure()
plot(x1,y1,'.')
```

⇨ on remarque que les limites du repère correspondent aux limites des données soit l'intervalle $[-1,1]$ en abscisse et l'intervalle $[-2,2]$ en ordonnée, et les points sont peu visibles.

Pour modifier les limites du repère, il suffit d'utiliser l'instruction `replot([xmin,ymin,xmax,ymax])`
• Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles :
`replot([-3 -2 3 2])` , `replot([-5 -2 5 4])` , `replot([0 -3 4 3])`

On peut modifier le mode de tracé en modifiant le troisième paramètre de la procédure `plot`.
• Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles :

```
clf(), plot(x1,y1,'g-'), replot([-2 -4 2 4])
clf(), plot(x1,y1,'k-'), plot(x1,y1,'ro'), replot([-2 -4 2 4])
```

⇨ on peut effectuer différents tracés dans une même fenêtre graphique en effectuant plusieurs instructions `plot`.
• Exemple : Rajoutez les instructions suivantes :

```
x2 = [-1  0  1  0 -1]
y2 = [ 0  1  0 -1  0]
plot(x2,y2,'c:'), plot(x2,y2,'m*')
```

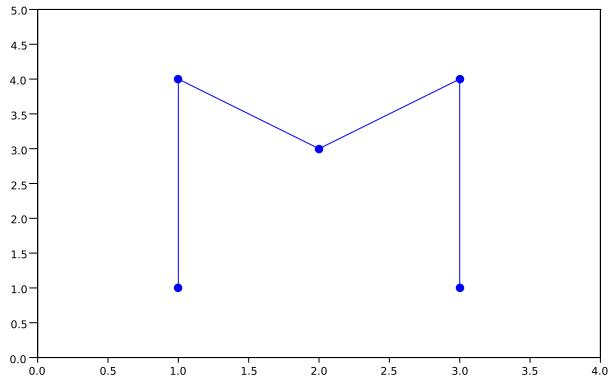
⇨ les points des vecteurs `x2` et `y2` correspondent à un carré, ce que la représentation graphique ne donne pas nécessairement (on voit plutôt un losange). Pour avoir une représentation graphique plus juste, il faut faire en sorte que le repère soit normalisé (même échelle en abscisse et en ordonnée).
Exécutez l'instruction suivante :

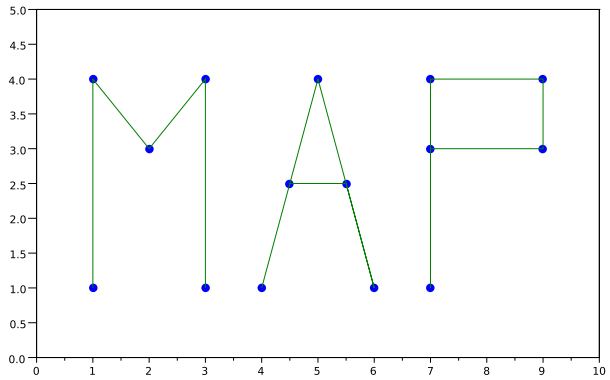
```
set(gca(),"isoview","on")
```

⇨ pour plus d'info sur l'instruction `plot`, tapez l'instruction `help plot`.

Exercice 4 :

écrire les instructions Scilab afin d'obtenir les deux figures suivantes :





6 Programmation

Scilab dispose d'un langage avec instructions structurées afin d'écrire des programmes complexes.

6.1 Entrée-sortie

La procédure `disp` permet l'affichage d'une variable ou d'une expression.
La procédure `input` permet à l'utilisateur d'entrer une valeur, un tableau ou une chaîne de caractères.
• Exemple : créez le fichier suivant en le nommant `ex_entree_sortie.sce`

```
n = input("Entrer un entier : ")
disp("n = " + string(n))
disp("n*n = " + string(n*n))

t = input("Entrer un tableau de valeurs (entre crochets) : ")
disp(t)
```

puis exécutez le script `ex_entree_sortie.sce`.
⇨ la procédure `string` convertit une valeur en chaine de caractères.

6.2 Test

L'instruction `if` permet d'exécuter une suite d'instructions si et seulement si une expression est vraie.
La syntaxe est :

`if` *expression_booleenne*
  instructions
`end`

⇨ c'est l'instruction générique **si** *expression_booleenne* **alors** *instructions* .

On peut aussi exécuter une suite d'instructions si une instruction est vraie et une autre suite d'instruction dans le cas contraire. La syntaxe est :

```
if expression_booleenne
   instructions1
else
   instructions2
end
```

⇨ c'est l'instruction générique | **si** *expression* **alors** *instructions1* **sinon** *instructions2* | .

• Exemple : créez le fichier suivant en le nommant `ex_test.sce`

```
n = input("Entrer un entier n : ")
disp(n)


if n>2 then
    disp("n est supérieur à 2")
end

if n==0 then
    disp("n est nul")
else
    disp("n est non nul")
end
```

puis exécutez le script `ex_test.sce`.

⇨ les opérateurs de comparaison sont :

<	>	==
inférieur strictement à	supérieur strictement à	égal à
<=	>=	<>
inférieur ou égal à	supérieur ou égal à	différent de

⇨ les opérateurs booléens sont :

&	|	~
ET	OU	NON

6.3 Boucle

L'instruction `while` permet de répéter une suite d'instructions tant qu'une expression booléenne est vraie. La syntaxe est :

```
while expression_booleenne
   instructions
end
```

⇨ c'est l'instruction générique | **tant que** *expression_booleenne* **faire** *instructions* | .

• Exemple : créez le fichier suivant en le nommant `ex_boucle1.sce`

```
n = input("Entrer un entier : ")

// calcul de la somme des entiers de 0 à n
i = 0; somme=0;
while i<=n
    somme = somme+i;
```

```
    i = i+1;
end
disp(somme)
```

puis exécutez le script `ex_boucle1.sce`.

L'instruction `for` permet de répéter une suite d'instructions pour un ensemble de valeurs
La syntaxe est :

`for` *variable* `=` *ensemble_valeur*
instructions
`end`

⇨ c'est l'instruction générique **pour** *variable* **variant dans** *ensemble_valeur* **faire** *instructions* .

En général, on l'utilise avec la syntaxe suivante :
`for` *variable* `=` *valeur_min*`:` *valeur_max*
instructions
`end`

et les instructions sont exécutées pour *variable* = *valeur_min*, *variable* = *valeur_min*+1, ...,
jusqu'à *variable* = *valeur_max*
⇨ c'est l'instruction générique
pour *variable* **variant de** *valeur_min* **à** *valeur_max* **avec un pas de 1 faire** *instructions* .

• Exemple : créez le fichier suivant en le nommant `ex_boucle2.sce`

```
disp("Exemple 1")
for i = 1:10
    disp(i)
end

disp("Exemple 2")
v = [2 7 8 3 9 1]
somme = 0;
for i = v
    disp(i)
    somme = somme+i;
end
disp("La somme des elements de v est " + string(somme));
```

puis exécutez le script `ex_boucle2.sce`.

Exercice 5 :

écrire un script Scilab nommé `exercice5.sce` et qui effectue les opérations suivantes :
- demande à l'utilisateur d'entrer une valeur positive a
- calcule le vecteur u formé de 10 valeurs de la manière suivante :

$$u(1) = \frac{a+1}{2} \quad \text{et} \quad u(n) = \frac{a}{2\,u(n-1)} + \frac{u(n-1)}{2} \text{ pour } n \text{ variant de 2 à 10}$$

- affiche les valeurs du vecteur u ainsi que les valeurs au carré du vecteur u.

7 Calcul numérique sur ordinateur

Tout ordinateur utilisant des valeurs numériques utilise une certaine plage de valeurs et une certaine précision car sa mémoire est finie. Le but de cette partie (à travers différents exemples et exercices) est de montrer que les valeurs réelles utilisées par Scilab sont limiteés en ordre de grandeur ainsi qu'en précision, et qu'il faut être vigilant lorsqu'on passe d'une formulation mathématique à un calcul numérique sur ordinateur.

7.1 Ordre de grandeur pour les réels

Exercice 6 :

écrire un script Scilab qui demande à l'utilisateur d'entrer un entier positif M puis qui affiche à l'écran les valeurs n et 2^n pour n variant de 1 à M.
Utiliser ensuite ce script : à partir d'une certaine valeur de n, la valeur 2^n est considérée par Scilab comme *infinie*. Déterminer la plus grande valeur de n qui donne une valeur 2^n non infinie pour Scilab.

Exercice 7 :

même exercice mais en affichant les valeurs n et 2^{-n}. A partir d'une certaine valeur de n, la valeur 2^{-n} est considérée comme nulle. Déterminer la plus grande valeur de n qui donne une valeur 2^{-n} non nulle pour Scilab.

7.2 Précision pour les réels

De même, en Scilab, on peut utiliser les réels uniquement avec une précision limitée.

Exercice 8 :

écrire un script Scilab qui effectue les opérations suivantes :
- définir les variables $n = 0$, $a = 1$ et $b = 1 + a$
- faire une boucle **tantque** qui effectue les instructions
  augmenter n de 1,
  diviser a par 10,
  calculer $b = 1 + a$,
  et afficher n.
- exécuter la boucle tant que $b > 1$.

puis tester ce script.

Formellement (mathématiquement), la boucle ne devrait pas s'arrêter car à chaque étape de la boucle $a = 10^{-n} > 0$ et donc $b = 1 + a > 1$ et ceci pour tout n entier.

Numériquement (par calcul sur ordinateur), la boucle s'arrête pour un certain entier n. Cette valeur de n donne le nombre de chiffres (décimaux) significatifs pour les valeurs réelles manipulées par Scilab.

7.3 Des mathématiques au calcul sur ordinateur

• Exemple : ce premier exemple montre que certains nombres décimaux (fraction dont le dénominateur est une puissance de 10) ne sont pas représentés de manière exacte sur un ordinateur.
Exécutez les instructions suivantes (censées donnés le même résultat 0) :

```
(9-6-3)/10
0.9-0.6-0.3
(0.9-0.6)*10 - 3
```

Ceci montre que certaines valeurs décimales n'ont pas une représentation exacte sur un ordinateur (qui calcule plutôt avec des puissances de 2).

• Exemple : ce deuxième exemple montre que dans le cas d'une suite de calculs, certaines formules mathématiques peuvent être prises en défaut dès qu'on les effectue sur un ordinateur.

La suite numérique $(v_n)_{n\in\mathbb{N}}$ suivante :

$$v_1 = 2\sqrt{2} \quad \text{et} \quad v_{n+1} = 2^{n+1}\sqrt{2-\sqrt{4-(v_n/2^n)^2}},\ \forall n \geq 1$$

est une suite qui converge vers la valeur π.

Exercice 9 :

écrire un script Scilab qui calcule les valeurs de v_n pour n entre 1 et 30 et les affiche à l'écran.

La suite v_n ainsi calculée va s'approcher de la valeur π puis ensuite va devenir constante et égale à 0. En effet dans la formule $v_{n+1} = 2^{n+1}\sqrt{2-\sqrt{4-(v_n/2^n)^2}}$, le terme 2^{n+1} tend vers l'infini alors que le terme $\sqrt{2-\sqrt{4-(v_n/2^n)^2}}$ tend vers 0 ce qui numériquement n'est pas approprié.

Pour obtenir le bon résultat de manière numérique, il faut modifier la formule qui calcule v_{n+1} à partir de v_n :

$$v_{n+1} = 2^{n+1}\sqrt{2-\sqrt{4-(v_n/2^n)^2}} \times \frac{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}}{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}}$$

$$= 2^{n+1} \times \frac{\sqrt{\left(2-\sqrt{4-(v_n/2^n)^2}\right)\left(2+\sqrt{4-(v_n/2^n)^2}\right)}}{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}} = \frac{2^{n+1}\sqrt{2^2-\left(\sqrt{4-(v_n/2^n)^2}\right)^2}}{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}}$$

$$= \frac{2^{n+1}\sqrt{4-4+(v_n/2^n)^2}}{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}} = \frac{2^{n+1}\sqrt{(v_n/2^n)^2}}{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}} = \frac{2\ v_n}{\sqrt{2+\sqrt{4-(v_n/2^n)^2}}}$$

Exercice 10 :

en utilisant cette nouvelle formule, écrire un script Scilab qui calcule les valeurs de v_n pour n entre 1 et 30 et l'affiche à l'écran.

Dans ce cas, la suite (v_n) ainsi calculée va bien tendre vers la valeur π car dans ce cas, le dénominateur $\sqrt{2+\sqrt{4-(v_n/2^n)^2}}$ tend vers 2.