Feuille de travaux pratiques # 1
Remise en forme Scilab

Préambule :

- Scilab est un logiciel libre que l'on peut télécharger à l'adresse `http://www.scilab.org`.
- Scilab possède une aide en ligne consultable en tapant `help` dans la console. Plus généralement pour obtenir une aide concernant une fonction scilab `bidule`, tapez `help bidule`.
- Pour vous familiariser avec les fonctions de base de Scilab, vous trouverez de nombreux tutoriels sur la toile, par exemple ici et là.
- Par ailleurs, il existe d'excellents ouvrages pour l'option proba/stat à l'agrégation, par exemple le livre de Chaffaï et Malrieu disponible ici, et d'autres plus particulièrement consacrés à l'utilisation de Scilab, par exemple l'excellent polycopié suivant, dont plusieurs exemples et exercices du présent document sont issus.

1 Prise en main

1.1 Fenêtre de commande, éditeur, aide en ligne

Lorsque l'on lance le logiciel Scilab, une fenêtre de commande s'ouvre et une invite de commande `- ->` indique que le logiciel attend vos instructions. Cette fenêtre peut être utilisée comme une calculatrice : vous entrez un calcul, vous tapez sur la touche "entrée" et Scilab vous donne le résultat.

Lorsque les tâches à exécuter se compliquent, il est recommandé de faire appel à deux fenêtres supplémentaires : la fenêtre de l'éditeur, accessible par le menu "éditeur", qui vous permet d'écrire, de sauvegarder, de modifier et d'exécuter des programmes, et la fenêtre de l'aide en ligne, indispensable pour retrouver rapidement la syntaxe précise des fonctions dont vous ne vous servez pas tous les jours. Comme dit plus haut, on y accède en tapant `help mot-clé` ou encore `apropos mot-clé`.

On peut, dans la fenêtre de Scilab copier et coller des lignes à l'aide de la souris (y compris les exemples donnés par l'aide) ; on peut aussi naviguer dans l'historique des commandes au moyen des flèches haut et bas, cela évite de retaper les commandes si l'on exécute une fonction plusieurs fois de suite par exemple.

Scilab est un logiciel de calcul numérique et non de calcul formel : il effectue des calculs numériques approchés et non des calculs exacts. Par défaut les réels sont affichés avec 7 décimales. On peut changer cet affichage par défaut en utilisant la commande `format` :

```
-->%pi
%pi  =
3.1415927
```

```
-->format(20);%pi
%pi  =
3.14159265358979312
```

Scilab permet de manipuler aisément des nombres, des matrices de grandes tailles. Néanmoins, dans certains modèles complexes, l'espace alloué pour le stockage des variables s'avère parfois insuffisant. On peut augmenter cet espace en utilisant la commande `stacksize('max')`.

1.2 Programmation

Dès que l'on souhaite exécuter plus de deux ou trois tâches consécutives, i.e. dès que l'on sort du mode d'utilisation "supercalculatrice" de Scilab, on utilise un éditeur de texte, le plus souvent l'éditeur intégré dans Scilab, pour écrire un programme de commande. Cet éditeur est accessible par un des menus de la fenêtre de commande. On enregistre le fichier obtenu avec le suffixe `.sce` puis on exécute le fichier dans Scilab, au moyen du menu `Fichier/exécuter...`, par un raccourci clavier, en cliquant sur une icône dans l'éditeur, ou par la commande `exec('monfichier.sce')` où "monfichier.sce" est le nom du fichier en question.

Lors de l'oral vous **devez** utiliser de tels fichiers.

Pour faciliter la mise au point et la relecture du programme par vous-même et sa compréhension par le jury, vous **devez également le commenter**. Une ligne de commentaires commence par `//`. De manière générale, Scilab ignore tout ce qui, sur une ligne, suit la commande `//`.

```
Début programme
[...]
// on définit maintenant les abscisses et ordonnées
x=[0:0.1:1] ; // le vecteur des abscisses
y=sin(x) ; // le vecteur des ordonnées
[...]
Fin du programme
```

L'éditeur de Scilab permet de commenter et de décommenter simplement des blocs de texte, ce qui est particulièrement utile si vous voulez n'exécuter qu'une partie de votre programme. Une autre méthode consiste à placer le code à ignorer entre les commandes `if` et `fi` comme suit :

```
Début programme
[...]
\if{
Code à ignorer
}\fi
[...]
Fin du programme
```

La dernière alternative est particulièrement pratique lorsque qu'un programme de fonctionne pas et que l'on souhaite savoir "òu ça coince".

1.3 Importation de données

Dans certains textes de modélisation de l'option proba/stat figurent des données qui sont disponibles au format excel (`.xls` ou `.xlsx`) ou au format texte (`.txt`). Ces données peuvent être importées dans Scilab pour ensuite être exploitées et illustrées. Les commandes de base sont les suivantes :

```
[a,b,c,d] = xls_open('file_path');
[e,f] = xls_read(g,h);
```

Exercice 1 Télécharger le fichier `donnees.xls` dans votre espace de travail. Importer le tableau dans Scilab, et tracer la seconde colonne en fonction de la première.

2 Opérations matricielles

L'objet de base pour Scilab est une matrice, par exemple, un réel est simplement une matrice 1×1. Les matrices peuvent avoir des coefficients réels, complexes, booléens, chaînes de caractères, polynômes etc. Les coefficients d'une matrice doivent cependant tous être de même type. Il faut, chaque fois qu'on le peut, utiliser les opérations matricielles prédéfinies. Le gain de temps, par rapport à l'utilisation de boucles, peut être considérable comme le montre l'exemple suivant :

```
-->timer();
-->for j=1:100
-->for i=1:100
-->B(i,j)=rand(1,1,'n');
-->end
-->end
-->timer()
ans  = 1.3
```

```
-->timer();
-->C=rand(100,100,'n');
-->timer()
ans  = 0.02
```

2.1 Commandes usuelles

Voici quelques commandes à connaître pour déterminer la taille d'une matrice A :

`size(A)`	nombre de lignes et de colonnes
`size(A,"r")` ou `size(A,1)`	nombre de lignes
`size(A,"c")` ou `size(A,2)`	nombre de colonnes
`size(A,"*")`	nombre total d'éléments

D'autres commandes utiles pour manipuler les vecteurs,

`x:y`	nombres de `x` à `y` par pas de 1
`x:p:y`	nombres de `x` à `y` par pas de `p`
`linspace(x,y)`	100 nombres entre `x` et `y`
`linspace(x,y,n)`	`n` nombres entre `x` et `y`
`v(i)`	`i`-ième coordonnée de `v`
`v($)`	dernière coordonnée de `v`
`v(i1:i2)`	coordonnées `i1` à `i2` de `v`
`v(i1:i2)=[]`	supprimer les coordonnées `i1` à `i2` de `v`
`[m,xmin]=min(v)`	minimum et position du minimum du vecteur `v`
`norm(v), norm(v,1)`	norme ℓ^2 et ℓ^1 de `v`

et les commandes analogues pour les matrices :

`A(i,j)`	coefficient d'ordre `i,j` de `A`
`A(i1:i2,:)`	lignes `i1` à `i2` de `A`
`A($,:)`	dernière ligne de `A`
`A(i1:i2,:)=[]`	supprimer les lignes `i1` à `i2` de `A`
`A(:,j1:j2)`	colonnes `j1` à `j2` de `A`
`A(:,$)`	dernière colonne de `A`
`A(:,j1:j2)=[]`	supprimer les colonnes `j1` à `j2` de `A`
`diag(A)`	coefficients diagonaux de `A`
`[V,l]=spec(A)`	spectre de la matrice `A` avec vecteurs propres
`[D,P]=bdiag(A)`	diagonalisation de `A` avec en prime la matrice de passage `P`

Lorsque l'on veut stocker des données obtenues de façon itérative par un programme/une boucle, on peut augmenter la taille du vecteur de stockage au fur et à mesure, ou définir une fois pour toutes le vecteur de stockage, et le remplir au fur et à mesure. La deuxième solution a l'avantage d'être plus explicite quand à la taille finale du vecteur de données.

```
// X contiendra les données
// issues de la boucle
for j=1:100
[...]
aj=... ;
X=[X, aj];
end
```

```
// X contiendra les données
// issues de la boucle,
// on initialise à zéro
X=zeros(1,100);
for j=1:100
[...]
aj=... ;
X(j)=aj;
end
```

Les commandes suivantes permettent de "construire" des matrices de manière efficace i.e. en limitant l'usage de boucles.

`zeros(m,n)`	matrice nulle de taille `m,n`
`ones(m,n)`	matrice de taille `m,n` dont les coefficients valent 1
`eye(m,n)`	matrice identité de taille `min(m,n)`, complétée par des zéros
`rand(m,n)`	matrice de taille `m,n` à coefficients aléatoires uniformes sur $]0,1[$
`diag(v)`	matrice diagonale dont la diagonale est le vecteur `v`
`diag(v,k)`	matrice dont la `k`-ième diagonale est le vecteur `v`
`diag(A)`	extrait la diagonale de la matrice `A`
`diag(A,k)`	extrait la `k`-ième diagonale de la matrice `A`
`triu(A)`	annule les coefficients au-dessous de la diagonale
`tril(A)`	annule les coefficients au-dessus de la diagonale
`toeplitz`	matrices à diagonales constantes
`[A]=testmatrix(name,n)`	matrices remarquables (carré magique, Hilbert etc.)
`sp=sparse(ij,v [,mn])`	outil pour la manipulation de matrices creuses

On rappelle enfin les commandes pour les opérations usuelles sur les matrices :

`+, -`	addition, soustraction
`*,^`	multiplication, puissance (matricielles)
`.*,.^`	multiplication, puissance terme à terme
`./`	division terme à terme
`sum(A)`	somme des éléments de `A`
`prod(A)`	somme des éléments de `A`
`cumsum(A)`	somme cumulative des éléments de `A`
`cumprod(A)`	produit cumulatif des éléments de `A`
`[B,...]=gsort(A,... )`	tri des élements des `A` (nombreuses options)
`kron(A,B)`	produit de Kronecker des matrices `A` et `B`
`A\b`	solution de $A * x = b$
`solve(A,b)`	idem en calcul symbolique
`lusolve(A,b)`	idem pour une matrice `A` creuse
`expm(A)`	exponentielle de la matrice `A`

2.2 Quelques exercices simples pour s'entrainer

À chaque fois, on cherchera à coder de façon "économique", i.e. on s'efforcera de raisonner en terme matriciel, on essaiera de limiter le nombre d'opérations, l'usage de boucles etc.

Exercice 2 Écrire un vecteur aléatoire de 10 réels. Réécrire ce vecteur dans l'ordre inverse.

Exercice 3 Écrire un vecteur de taille 10 dont les éléments sont successivement $+1$ et -1.

Exercice 4 Construire les matrices suivantes en utilisant le moins d'opérations possibles.

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 2 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 2 \\ 1 & 1 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 3 \end{pmatrix}.$$

Exercice 5 Écrire (sans utiliser de boucle) les matrices carrées d'ordre 6 suivantes :

1. Matrice diagonale, dont la diagonale contient les entiers de 1 à 6.
2. Matrice contenant les entiers de 1 à 36, rangés par lignes.
3. Matrice dont toutes les lignes sont égales au vecteur des entiers de 1 à 6.
4. Matrice diagonale par blocs, contenant un bloc d'ordre 2 et un d'ordre 4. Les 4 coefficients du premier bloc sont égaux à 2. Le deuxième bloc contient les entiers de 1 à 16 rangés sur 4 colonnes.
5. Matrice $A = (a_{i,j})$ dont les coefficients sont $a_{i,j} = (-1)^{i+j}$.
6. Matrice contenant des 1 sur la diagonale, des 2 au-dessus et au-dessous, puis des 3, etc., jusqu'aux coefficients d'indices (1,6) et (6,1) qui valent 6.

Exercice 6 Écrire la matrice $A = (a_{i,j})$ d'ordre 12 contenant les entiers de 1 à 144, rangés par lignes. Extraire de cette matrice les matrices suivantes :

1. Coefficients $a_{i,j}$ pour i de 1 à 6 et j de 7 à 12,
2. Coefficients $a_{i,j}$ pour $i + j$ pair,
3. Coefficients $a_{i,j}$ pour $i, j = 1, 2, 5, 6, 9, 10$.

3 Opérations logiques, boucles

3.1 Opérateurs logiques et booléens

L'objet de base de Scilab étant une matrice, les opérations logiques de bases comme le "et" logique (`&`), le "ou" logique (`|`), la négation (`~`) sont aussi implémentées matriciellement, ce qui autorise à traiter simultanément de nombreuses opérations logiques et ainsi considérablement réduire la taille des programmes et l'utilisation de boucles.

```
--> a = [1, -1, 2, -2, 3] ;
--> b = (a>0)
--> b = T F T F T
```

```
--> a = [1, -1, 2, -2,  ; 2 , 3, -5, 6 ] ;
--> b = (a>0)|(abs(a)>5)
--> b = [ T F T F ; T T F T ]
--> c = (a>0)&(abs(a)>5)
--> c = [ F F F F ;  F F F T ]
```

Par ailleurs, le passage grâce à la fonction `bool2s` des valeurs logiques `T` (true) et `F` (faulse) aux valeurs numériques 1 et 0 permet de traiter les opérations logiques "et" et "ou" à l'aide de multiplication et addition matricielles. Notez que Scilab interprête `T` (true) et `F` par 1 et 0 dès lorsque que l'on utilise une opération numérique, rendant alors superflu le passage par `bool2s` :

```
--> a = 1 ;
--> b = (a>0)
--> b = T
--> c = (a >2)
--> c = T
--> b+c = 1
--> b*c = 0
```

Voici à titre d'exemple quelques opérations logiques matricielles :

```
--> a = [1, -1, 2, -2, 3] ;
--> b = (a>0)
--> b = [ T F T F T ]
--> bool2s(b) = [ 1 0 1 0 1 ]
--> c = a.*b
--> c = [1, 0, 2, 0, 3]
--> d = 2.*b
--> d = [2, 0, 2, 0, 2]
```

```
--> a = [1, -1, 2, -2  ; 2 , 3, -5, 6 ] ;
--> b = (a>0)|(abs(a)>5)
--> b = [ T F T F ; T T F T ]
--> c = (a>0)&(abs(a)>5)
--> c = [ F F F F ;  F F F T ]
--> d = (a>0).*(abs(a)>5)
--> c = [ 0 0 0 0 ;  0 0 0 1 ]
```

Une commande fort utile pour sélectionner des éléments vérifiant une certaine propriété dans un vecteur/une matrice est la commande `find` dont voici une illustration :

```
--> a = rand(1, 5)
--> a = [ 0.0437334    0.4818509    0.2639556    0.4148104    0.2806498  ] ;
--> b = find(a>0.4)
--> b = [ 2. 4]
--> c = a(b)
--> c =  [ 0.4818509    0.4148104 ]
--> a =rand(2,3)
--> a = [ 0.0437334  0.4818509  0.2639556 ; 0.4148104   0.2639556  0.4806498 ]
--> b= find(a>0)
--> b= [ 2. 4. 6.]
```

3.2 Boucles

Même si l'on cherche en général à l'éviter pour la rapidité d'exécution des programmes, l'usage des boucles est parfois inévitable. Les syntaxes des boucles `for`, `while` et des instructions conditionnelles sont les suivantes :

```
i = 0; aux=2;
while (i<5)&(aux<10)
  aux=aux*aux;
  i = i + 1;
end
```

```
n=5;
for i = 1:n
   a(i)=1/(i+1);
   a(i)=1/(i+1);
end
```

```
if i == j then
   a(i,j) = 2;
else
   a(i,j) = 0;
end,
```

On veillera à bien indenter les codes dans l'éditeur de Scilab, cela permet entre autres de vérifier que les boucles `for` ou `while` ou chaque condition `if` est bien "fermée" par un `end`.

3.3 Quelques exercices simples pour s'entrainer

Exercice 7 Écrire une fonction `dif` réciproque de la fonction `cumsum` agissant sur les vecteurs.

Exercice 8 Écrire la matrice Zorro de taille n, i.e. la matrice carrée de taille n dont tous les coefficients sont nuls sauf la première ligne, la dernière ligne et l'anti-diagonale qui valent 1.

Exercice 9 Étant donne un triangle $A_1A_2A_3$ et un point M dans le plan, repérés par leurs coordonnées cartésiennes, écrire une fonction qui indique si M est à l'interieur du triangle ou non.

Exercice 10 Écrire une fonction renvoyant les $n+1$ premières lignes du triangle de Pascal (complétées par des 0 pour apparaître sous forme de matrice carrée).

Exercice 11 Écrire une fonction renvoyant la liste de tous les nombres premiers inférieurs ou égaux à son argument en utilisant la méthode du crible d'Eratosthène. Représenter graphiquement la fonction de comptage des nombres premiers, i.e. la fonction qui à $x \in \mathbb{R}^+$ associe le nombre de nombres premiers inférieurs ou égaux à x.

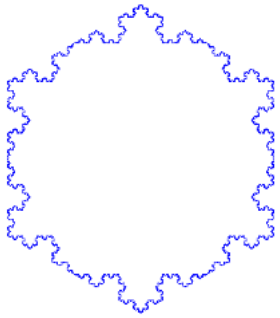
Exercice 12 Définir une fonction qui prend un vecteur $x = (x_1, \ldots, x_n)$ en entrée et retourne la matrice de Vandermonde ci-dessous ainsi que son déterminant.

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & & \vdots \\ x_1^n & x_2^n & \ldots & x_n^n \end{pmatrix}$$

On fera une version avec deux boucles imbriquées, avec une seule boucle et sans boucle. On comparera les temps d'exécution de chacune des méthodes.

4 Pour celles et ceux qui sont très à l'aise avec Scilab

Exercice 13 Écrire une fonction qui, étant donné un entier n, trace le n-ième itéré du flocon de Van Koch. Voici par exemple ce que l'on obtient pour $n = 7$, lorsque l'on part d'un hexagone régulier.



Randoniser ensuite votre algorithme pour qu'à chaque itération, seul un nombre aléatoire (à vous de choisir la loi) de triangles "ne fleurissent" sur les bords du flocon.

Exercice 14 En première approximation, le mouvement d'une corde de guitare de longueur L pincée à ses deux extémités peut s'écrire sous la forme d'une fonction $(t, x) \mapsto u(t, x)$ où $t > 0$ représente le temps, $x \in [0, L]$ la position, et $u(t, x)$ la déformation verticale de la corde à l'instant t et à l'abscisse x. On admet que chaque mouvement s'écrit comme combinaison linéaire des mouvements fondamentaux, appelés modes :

$$\begin{aligned}(t, x) &\mapsto \cos(n\omega t)\sin(n\pi x/L),\\ (t, x) &\mapsto \sin(n\omega t)\sin(n\pi x/L),\end{aligned}$$

pour tout n entier non nul, la fréquence ω étant propre à la corde.

1. En utilisant l'instruction `clf` permettant d'effacer un graphique, représenter le mouvement correspondant aux deux premiers modes d'une corde ($n = 1$ et 2 respectivement avec $\omega = 1$ et $L = 1$) ainsi que le mouvement résultant de la superposition de ces deux modes entre les instants $t = 0$ et $t = 20$.
2. Utiliser l'instruction `set` pour obtenir une meilleure animation.