Université Paris Descartes
UFR de Mathématiques et Informatique
45, rue des Saints-Pères 75270 Paris cedex 06

MASTER 1

PROGRAMMATION

Introduction et prise en main de `Scilab`

Scilab est un logiciel libre de calcul numérique, qui a son propre langage de programmation et un environnement interactif. On peut exécuter une commande soit en la tapant directement en ligne de commande dans la fenêtre graphique de Scilab soit en exécutant un script qui est une suite de commandes qui se trouvent dans un fichier dont l'extension est "**.sci**" ou "**.sce**". Une documentation en ligne est disponible sur le site `www.scilab.org`, ou à l'aide des commandes **help** et **apropos**.
Cette introduction a pour objectif de vous apprendre ou rappeler l'utilisation élémentaire de Scilab. Une partie de ce document est tutorielle et peut être exécutée entièrement en ligne de commande, mais dès qu'il faudra construire des programmes plus complexes on aura recourt à des scripts et fonctions afin de gagner en efficacité. Pendant la partie tutorielle, vous êtes encouragés à taper les commandes et à en comprendre le résultat. Faire suivre une commande par un " ;" avant de taper [ENTREE] fait que le résultat du calcul n'est pas visualisé. Si vous voulez voir le résultat il ne faut pas mettre de " ;". De plus le caractère // introduit un commentaire, tout ce qui le suit sur la ligne est ignoré. Exemple

```
3+4; // Vous n'êtes pas obligés de taper les commentaires.
3+4  // ou comment utiliser 10^9 opérations par seconde pour calculer 7
```

1 Les bases

Dans Scilab, toutes les variables sont des matrices dont les entrées sont des réels (représentés par un type "double") ou des complexes ($x + iy$, où x et y sont réels). On dit qu'une matrice est de taille $m \times n$ si elle a m lignes et n colonnes. Un **vecteur colonne** est une matrice qui a une seule colonne (de taille $n \times 1$) et un **vecteur ligne** est une matrice qui a une seule ligne (de taille $1 \times n$).

2 Générer une matrice

Il y a divers moyens de générer une matrice ou un vecteur

- **zeros(m,n)** : Génère une matrice de taille $m \times n$ pleine de zéros.
- **ones(m,n)** : Cette fois-ci la matrice est pleine de 1.
- **eye(m,n)** : Matrice identité de taille m.
- **linspace(x,y,n)** : Vecteur de n valeurs régulièrement espacées entre x et y. Par exemple `linspace(1.3,15.8,5)`
- **a :b :c** : Génère un vecteur ligne commençant à a et incrémenté de b tant que l'on est encore plus petit ou égal à c. Si on tape $a : c$ l'incrément par défaut est 1. Essayez par exemple
  `0:3:6`
  `0:3:7`
  `1:6`
  `0:0.1:1`

```
0:-0.1:1
1:-0.1:0
```

— **rand(m,n)** : matrice de taille $m \times n$ remplie de nombres aléatoires tirés selon la loi uniforme sur $[0, 1]$. Si on veut plutôt qu'ils suivent une loi gaussienne centrée réduite, on écrira **rand(m,n,"normal")**.

— **Entrer directement les valeurs entre crochets** :

```
[1 2 3] \\ vecteur ligne 1 2 3
[1;2; 3] \\ vecteur colonne
[1 2; 3 4] \\ Une matrice 2 par 2
[1 2; 3] \\ UNE ERREUR (voir concaténation plus bas)
```

— **Indices et sous-matrices**.

Les indices en Scilab commencent à 1 : **h(1)** est le premier élément du vecteur **h**. **A(2,1)** est l'élément situé à la ligne 2 colonne 1 de la matrice **A**. Si **A** est une matrice de taille $m \times n$, la matrice

```
B = A(i:j,k:l)
```

est la sous-matrice de **A** correspondant aux lignes i à j et aux colonnes k à l. On extrait la i-ième ligne (resp. colonne) de **A** par la commande

```
A(i,:)  (resp. A(:,i))
```

Essayez par exemple

```
h=0:9;
h(1:3)
sum(h(1:10)) \\ somme de tous les éléments de h
sum(h(1:length(h))) \\ length(h) renvoie la longueur d'un vecteur
sum(h(1:2:length(h))) \\ h(1)+h(3)+h(5)+...
h(length(h):-1:1) \\ renvoie le vecteur h à l'envers
h(2:4)=ones(1,3) \\ remplace certaines entrées de h par des 1
```

3 Opérations sur les matrices

— **A+B** : Renvoie la somme des deux matrices qui doivent être de même taille. **Exception** : Si A ou B est un scalaire (une matrice de taille 1×1), il est ajouté à toutes les entrées de l'autre matrices. L'opération "-" fonctionne de la même manière pour la soustraction

```
ones(2,2)+ones(2,2)
[1 2; 3 4]+1 \\ on ajoute 1 à toutes les entrées.
```

— **A*B** : Renvoie la multiplication des matrices A et B qui doivent avoir des tailles respectives de $m \times n$ et $n \times l$. Le résultat est de taille $m \times l$. Notez que si $m = l = 1$ alors cette opération est un produit scalaire entre le vecteur ligne A et le vecteur colonne B. Il y a la même exception que ci-dessus.

```
[1 2; 3 4]*[-2 1; 1.5 -0.5]
[1 2; 3 4]*[1 0; 0 1]
[1:20]*[1:20]'  \\ A' est la matrice transposée (et conjuguée) de A
(1:3)'*(1:3)
```

— **A.*B** : Multiplication point par point. A et B doivent avoir la même taille. Comparez

```
[1 2; 3 4]*[-2 1; 1.5 -0.5]
[1 2; 3 4].*[-2 1; 1.5 -0.5]
```

— **Fonctions unaires** : Scilab dispose d'un grand nombre de fonctions usuelles prédéfinies telles que : sin, cos, exp, tan, log.... Les appliquer à une matrice signifie appliquer la fonction à chaque élément de la matrice

```
exp((0:10)) \\ fonction exponentielle appliquée au vecteur $(0,1\dots,10)$.
```

— **sum(v)** : Renvoie la somme des éléments de v.

```
sum(ones(1,10))
sum(ones(10,1))
```

```
sum(ones(2,3))
sum((1:10).*(1:10)) \\ somme des carrés des entiers de 1 à 10
```

— **[A B]** : Si A et B sont deux matrices avec le même nombre de lignes, renvoie la concaténation horizontale des deux.

```
[ ones(2,6) zeros(2,4) ones(2,5)] \\ matrice à deux lignes identiques 6x1,4x0,5x1
```

— **[A ;B]** : A et B ont le même nombre de colonnes, renvoie la concaténation verticale

— **A(:)** : Quelque soit la forme de A, renvoie un vecteur colonne qui est la concaténation verticale des colonnes de A. Utile pour s'assurer que deux vecteurs ont la même forme (colonne)

```
A=[1 2; 3 4];
A(:)
u=[1 2 2 1]
v=[1 ;3; 4; 5]
u.*v \\ erreur
u(:).*v(:) \\on est sûr de la forme
```

— **x.^ A** : élève le scalaire x à la puissance chacune des entrées de A

```
(-1/2).^(0:10) \\ les puissances de -1/2 de 0 à 10
```

— **Constantes** Scilab contient plusieurs constantes, comme le nombre complexe i, que l'on appelle avec la commande %i, ou le nombre π, avec la commande %pi.

```
exp(2*%i*%pi*0.123*(0:10)) \\ Onde de Fourier sur Z de fréquence 0.123,  entre 0 et10
real(exp(2*%i*%pi*0.123*(0:10))) \\ Partie réelle de l'onde
plot(real(exp(2*%i*%pi*0.123*(0:10)))) \\ affichage graphique
```

Exercices

1. On considère les vecteurs de $\mathbb{R}^3$ suivants

$$\vec{u} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \ \vec{v} = \begin{pmatrix} -5 \\ 2 \\ 1 \end{pmatrix} \text{ et } \vec{w} = \begin{pmatrix} -1 \\ -3 \\ 7 \end{pmatrix}.$$

   a. Créer des tableaux correspondants à ces vecteurs.

   b. Calculer $\vec{u} + \vec{v}$, $\vec{u} + 3\,\vec{v} - 5\,\vec{w}$, $\frac{1}{5}\,\vec{w}$.

   c. En utilisant les commandes appropriées, calculer $\|\vec{u}\|_2$, $\|\vec{v}\|_1$, $\|\vec{w}\|_\infty$ et le cosinus de l'angle formé par les vecteurs $\vec{u}$ et $\vec{v}$.

2. On considère les matrices suivantes :

$$A = \begin{pmatrix} 2 & 3 \\ 6 & 5 \end{pmatrix} \text{ et } B = \begin{pmatrix} 2 & 3 & 4 \\ 7 & 6 & 5 \\ 2 & 8 & 7 \end{pmatrix}.$$

   a. Créer des tableaux correspondants à A et B.

   b. En utilisant les commandes appropriées, calculer leurs déterminants, inverses et valeurs propres et vecteurs propres associés [1].

3. Pour $n \in \mathbb{N}$, $n \geq 2$, créer le tableau correspondant à la matrice tridiagonale d'ordre n définie par

$$A_n = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}.$$

1. On consultera l'aide en ligne de la commande `spec`.

4. Scilab est beaucoup plus efficace lorsqu'il travaille sur des matrices que lorsqu'il exécute des boucles. On propose de se servir de la fonction timer() pour l'observer en exécutant les instructions suivantes :

```
n = 400;
timer();
for j=1:n, for i=1:n, a(i,j) = cos(i)*cos(j); end; end;
t1 =timer(); clear a;
```

Puis

```
timer();
a=zeros(n,n);
a = cos(1:n)'*sin(1:n);
t2 =timer();
```

4 Scripts et fonctions

Un programme Scilab est une suite d'instructions. Ces instructions peuvent être placées dans un fichier (que l'on appelle un script) à l'aide d'un éditeur de texte : l'éditeur associé *scipad* ou un autre. Si le script s'appelle **monscript.sce**, on l'exécute avec la commande

```
exec monscript.sce
```

Il est possible de contrôler le niveau d'affichage de la commande exec avec la syntaxe

```
exec("monscript.sce",n)
```

où n peut prendre des valeurs entre -1 (pas d'affichage) et 2 (afficher tous les détails).
Une fonction Scilab définie par l'utilisateur est un fichier de script spécial qui commence par une ligne du type

```
function [out]=mafonction(a,b,c)
```

La variable "out" sera celle qui contiendra le résultat de "mafonction". a,b et c sont les paramètres de la fonction. Voici un exemple complet de fonction

```
function out=mafonction(a,b,c)
tmp=a+b; \\tmp est une variable locale qui disparaitra à la fin de la fonction
out=tmp*c; \\ la valeur de out à la fin de l'execution est la réponse de "mafonction"
```

Avant de pouvoir utiliser cette fonction en ligne de commande, chargez la en tapant :

```
exec mafonction.sci
```

Exercices - Ecriture de fonctions

1. Écrire une fonction nommée `polaire`, prenant comme arguments d'entrée les coordonées cartésiennes (x, y) d'un point de $\mathbb{R}^2$ et renvoyant en sortie les coordonnées polaires (r, θ) de ce point [2].

2. On rappelle que, en vertu du théorème de Pythagore, on a $r = \sqrt{x^2 + y^2}$ et que, pour obtenir l'angle θ dans l'intervalle $[0, 2\pi[$, on utilise les formules suivantes :

$$\theta = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{si } x > 0 \text{ et } y \geq 0, \\ \arctan\left(\frac{y}{x}\right) + 2\,\pi & \text{si } x > 0 \text{ et } y < 0, \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{si } x < 0, \\ \frac{\pi}{2} & \text{si } x = 0 \text{ et } y > 0, \\ \frac{3\,\pi}{2} & \text{si } x = 0 \text{ et } y < 0. \end{cases}$$

2. Ecrire une fonction nommée `profil` qui associe à toute matrice A à m lignes et n colonnes son profil, c'est-à-dire la suite d'entiers $\{\phi(i)\}_{i=1,\dots,m}$ avec ϕ une application de $\{1,\dots,m\}$ dans $\mathbb{N}$ telle que

$$\phi(i) = \begin{cases} \inf\{j \in \{1,\dots,n\} \mid a_{ij} \neq 0\} & \text{si la } i^{\text{ème}} \text{ ligne de } A \text{ est non nulle,} \\ n+i & \text{sinon.} \end{cases}$$

3. **Suite de Fibonacci**
   (a) Écrire une boucle calculant les valeurs des vingt premiers termes de la suite de Fibonacci définie par
   $$u^{(0)} = 0,\ u^{(1)} = 1 \text{ et } u^{(k+2)} = u^{(k+1)} + u^{(k)},\ \forall k \in \mathbb{N},$$
   et conservant ces valeurs dans un vecteur.
   (b) Écrire une boucle calculant les termes successifs de la suite de Fibonacci dont la valeur est inférieure ou égale à 50000 et afficher le dernier de ces termes.
   (c) Écrire enfin une fonction `fibonacci(n)` calculant de manière itérative le $n^{\text{ième}}$ terme de la suite de Fibonacci, sans toutefois conserver les valeurs de tous les termes de la suite.
4. **Suites adajacentes**
   On définit deux suites $(u^{(k)})_{k\in\mathbb{N}}$ et $(v^{(k)})_{k\in\mathbb{N}}$ par
   $$u^{(0)} = 1,\ v^{(0)} = 2,\ u^{(k+1)} = \frac{u^{(k)} + v^{(k)}}{2},\ v^{(k+1)} = \sqrt{u^{(k+1)} v^{(k)}},\ \forall n \in \mathbb{N}.$$
   On admet que ces suites sont adjacentes, de limite $\dfrac{\sqrt{27}}{\pi}$.
   (a) Écrire un programme lisant [3] un entier n et affichant l'approximation du nombre π obtenue à partir de la valeur de $v^{(n)}$.
   (b) Écrire un programme lisant un réel ε strictement positif et affichant l'approximation du nombre π obtenue à partir de la valeur de $v^{(n)}$, premier terme de la suite $(v^{(k)})_{k\in\mathbb{N}}$ à satisfaire la condition
   $$\left| \frac{u^{(n)} - v^{(n)}}{u^{(n)} + v^{(n)}} \right| \leq \varepsilon,$$
   avec ε un réel strictement positif fixé
5. **Calcul d'une valeur approchée de π par la méthode de Monte-Carlo**
   Pour obtenir une valeur approchée du nombre π par la méthode de Monte-Carlo, on tire « au hasard » [4], dans un carré de côté de longueur égale à 2, des points de coordonnées (x, y) et l'on vérifie s'ils appartiennent ou non au disque de rayon 1 et de centre le centre du carré. Ces points pouvant être tirés avec la même probabilité dans l'ensemble du carré, le rapport entre le nombre de points tirés dans le disque et le nombre de points tirés au total tend, lorsque le nombre de tirages tend vers l'infini, vers le rapport des surfaces du cercle et du carré, soit $\frac{\pi}{4}$, en vertu de la loi des grands nombres.
   (a) Au moyen de la commande `rand`, qui génère une suite de nombres réels jouant le rôle d'une réalisation d'une suite de variables aléatoires continues, indépendantes et identiquement distribuées selon la loi uniforme sur l'intervalle $[0, 1]$, écrire une fonction prenant comme argument le nombre de tirages à réaliser et renvoyant la valeur approchée de π obtenue par la méthode de Monte-Carlo décrite ci-dessus (pour simplifier, on pourra se restreindre au quart de carré contenu dans l'orthant positif de $\mathbb{R}^2$).

3. Utiliser par exemple la fonction `input` pour demander à l'utilisateur de fournir une réponse.

3. On appelle méthode de Monte-Carlo toute méthode visant à calculer une approximation numérique par utilisation d'un procédé aléatoire. Le nom de ce type de méthode, qui fait allusion aux jeux de hasard pratiqués dans le célèbre casino d'un des quartiers de la cité-État de la principauté de Monaco, a été inventé en 1947 par Nicholas Metropolis et publié pour la première fois en 1949 dans un article co-écrit avec Stanislas Ulam.

4. C'est-à-dire selon une loi de probabilité uniforme.

(b) Donner un ordre du nombre de tirages nécessaires pour obtenir plus de deux décimales exactes de π. Que dire de l'efficacité de cette méthode ?

6. **Programmation récursive**

En informatique, une fonction dite récursive lorsqu'elle s'appelle elle-même. En pratique, une telle fonction aura toujours au moins une instruction conditionnelle, afin que, dans certains cas au moins, il n'y ait pas d'appel récursif (sans quoi la fonction s'appellerait indéfiniment jusqu'à la saturation de la pile, provoquant une interruption du programme). Le concept de fonction récursive est généralement opposé à celui de fonction itérative, qui s'exécute sans s'invoquer ou s'appeler explicitement.
Bien que cette forme de programmation aboutisse à des programmes concis et proches des formulations mathématiques qui en sont à l'origine, il peut parfois être mal indiqué ou même catastrophique d'employer la récursivité (toute fonction récursive pouvant être remplacée par une fonction itérative), comme on le vérifiera à la troisième question du présent exercice.

(a) Écrire une fonction récursive `rfactorielle(n)` calculant $n!$.

(b) Écrire, en utilisant la fonction `pmodulo` donnant le reste de la division euclidienne de deux entiers, une fonction récursive `PGCD(a,b)` renvoyant le plus grand commun diviseur [5] des entiers naturels a et b calculé par l'algorithme d'Euclide [6].

(c) Écrire une fonction récursive `rfibonacci(n)` calculant le $n^{\text{ième}}$ terme de la suite de Fibonacci et comparer son temps d'exécution avec celui de la fonction `fibonacci(n)` de l'exercice 3.

5 Représentation graphique

La commande

```
plot2d(x,y)
```

où x et y sont deux vecteurs de même dimension, produit le graphe de la ligne brisée obtenue en reliant chaque point de coordonnée $(x(i), y(i))$ au point suivant de coordonnée $(x(i+1), y(i+1))$. Les vecteurs x et y doivent donc avoir au moins deux composantes. Lorsque le vecteur x est omis, il est pris par défaut égal à $(1 : n)$ où n est la dimension du vecteur y. Pour tracer le graphe de la fonction sinus sur $[0, 2\pi]$ avec 100 points, on utilisera la commande :

```
x=linspace(0,2*%pi,100);
plot2d(x,sin(x))
```

On peut aussi utiliser le menu Zoom pour zoomer sur une partie du graphe.
Par défaut, la fonction **plot2d** n'efface pas la fenêtre graphique si elle est déjà ouverte, mais elle superpose son tracé dans la fenêtre courante. Si, à la suite des commandes précédentes, on fait :

```
t=linspace(0,4*%pi,100);
plot2d(t,0.5*cos(t))
```

on obtiendra le graphe des deux fonctions sinus et cosinus, dans une échelle commune (l'axe des x allant de 0 à 4π après le deuxième tracé). Si l'on ne souhaite pas superposer les courbes, il est donc nécessaire de commencer par effacer la fenêtre graphique. Cela se fait par la commande clf().

5. C'est-à-dire le plus grand entier naturel qui divise simultanément ces deux entiers.

6. Cet algorithme est basé sur la propriété suivante : *on suppose que $a \geq b$ et on note r le reste de la division euclidienne de a par b ; alors le pgcd de a et b est le pgcd de b et r*. En pratique, il suffit donc de faire des divisions euclidiennes successives jusqu'à trouver un reste nul. Le dernier reste non nul est le PGCD.

Exercices - représentations graphiques

1. Tracer une représentation graphique de la fonction $f(x) = \exp(-x)\sin(4\,x)$ échantillonnée sur 101 valeurs équidistantes sur l'intervalle $[0, 2\,\pi]$.
2. En utilisant le zoom, déterminer une valeur approchée du maximum de f sur $[0, 2\,\pi]$. Comment affiner le tracé pour préciser ce maximum ?
3. Tracer sur une même figure une représentation graphique des fonctions $x \mapsto x^2$ et $x \mapsto x^2 \sin(x)\exp(-x)$ sur l'intervalle $[-1, 1]$, en utilisant des couleurs différentes pour chacune d'entre elles.

6 Format d'affichage

Scilab utilise par défaut le format de représentation double précision de la norme IEEE 754 pour les calculs arithmétiques et toute variable est *a priori* stockée dans ce format. On peut afficher des (tableaux de) valeurs numériques ou chaînes de caractères en tapant simplement le nom des variables correspondantes ou en utilisant la commande `disp`, cette dernière réalisant l'affichage sans écrire le nom de la variable concernée.
Il existe différents formats d'affichage prédéfinis, sélectionnables grâce à la commande `format`.

1. Tester ces formatsen exécutant la suite d'instructions suivantes :

```
x=%pi^5;
disp(x) % affichage avec la notation par défaut
format('e',20) % notation au format scientifique
disp(x)
format('v',10) % notation par défaut
```

Ces changements de format ne modifient pas la précision des calculs de Scilab, seulement la visualisation des résultats. L'affichage est variable et s'adapte au nombre à représenter.
Il est aussi possible de contrôler très précisement la mise en forme d'un affichage avec la commande `mprintf`, qui permet de spécifier explicitement le format via une chaîne de caractères. Dans cette dernière, le format voulu pour chaque variable débute par le signe `%`, suivi de la longueur minimale du champ (c'est-à-dire le nombre minimal de caractères utilisés pour l'affichage) et d'un caractère de conversion (une lettre) indiquant que la valeur à afficher est un unique caractère (`%c`), une chaîne de caractères (`%s`), un entier signé en base 10 (`%d` ou `%i`) ou encore un réel en notation à séparateur fixe (`%f`), en notation « scientifique » (`%e`) ou « compacte » (`%g`). Dans les derniers cas, on peut préciser le nombre maximal de caractères utilisés (pour `%s`), le nombre de chiffres situés à droite du séparateur (pour `%f` ou `%e`) ou bien le nombre de chiffres significatifs (pour `%g`), par un point suivi d'un chiffre. Indiquons que plusieurs autres opérateurs optionnels existent (`'`, `-`, `+` ou le caractère « espace »), ainsi que des caractères d'échappement, tels que `\n` pour un saut de ligne, `\t` pour une tabulation horizontale, etc... Le tableau de variables à afficher donné en argument de la commande doit contenir au moins autant d'éléments qu'il y en a de prévu dans la chaîne de caractère donnant le format.

2. Tester les possibilités offertes par `mprintf` en exécutant les commandes suivantes :

```
mprintf('%c\n','abcdefghijklmnopqrstuvwxyz')
mprintf('%s\n','abcdefghijklmnopqrstuvwxyz')
mprintf('%.13s\n','abcdefghijklmnopqrstuvwxyz')
mprintf('%i %i %i\n',[0:2;4:6])
mprintf('%.0f\t%.1f\t%.10f\n',%pi,%pi,%pi)
mprintf('%.0e\t%.1e\t%.10e\n',%pi,%pi,%pi)
mprintf('%.0g\t%.1g\t%.10g\n',%pi,%pi,%pi)
```

puis en les modifiant. La fonction mprintf permet aussi de visualiser des tableaux de nombres et de chaînes :

```
colors = ["rouge";"vert";"bleu";"rose";"noir"];
RGB = [1 0 0;0 1 0;0 0 1;1 0.75 0.75;0 0 0];
mprintf("\nNoms\tR        \tG        \tB\n");
mprintf("%s\t%f\t%f\t%f\n",colors,RGB);
```