Bases de programmation en Scilab

avec quelques remarques sur la compatibilité avec Matlab

14 janvier 2017

Pierre Rousselin

La dernière version de ce document est disponible sur le site :
https://www.math.univ-paris13.fr/~rousselin/

N'hésitez pas à me faire part de vos commentaires et à me signaler les coquilles, erreurs et imprécisions à l'adresse indiquée sur ce site.

Cette introduction à Scilab suppose que le lecteur a déjà des connaissances (même vagues et lointaines) en programmation (sait ce qu'est une variable, une boucle `for`, une fonction,...) Si tel n'est pas le cas, vous pouvez m'envoyer vos questions.

J'ai essayé d'être le moins bavard possible et en général, plutôt que d'expliquer ce que font les commandes, je vous invite à les tester vous-même dans des exercices dont l'intitulé comporte parfois les éléments de syntaxe ou les fonctions nécessaires à sa résolution.

Introduction

Scilab fait partie de la famille des langages de programmation :

interprétés : On ne produit pas de « code machine », c'est un programme appelé interpréteur (ici, le *logiciel* Scilab) qui se charge de comprendre et exécuter nos instructions.

de haut niveau : On ne travaille pas avec les données brutes de la mémoire vive et les instructions du micro-processeur mais avec des abstractions de celles-ci.

vectorisés : On gagne du temps de calcul et du temps de programmation en faisant des opérations directement sur des vecteurs et des matrices et non sur leurs éléments.

à typage dynamique : Les variables peuvent changer de type au cours de l'exécution du programme.

tournés vers le calcul numérique : Bien que l'on puisse en théorie tout faire, ce n'est pas un langage généraliste et il vaut mieux ne pas le pousser trop loin en dehors de son domaine d'expertise.

Scilab partage ces traits avec Matlab (payant, propriétaire), GNU Octave (gratuit, libre), Python + Numpy + Scipy[1](gratuit, libre) et d'autres encore.

Scilab est proche de Matlab, mais, contrairement à Octave il n'est pas compatible avec lui. Un programme écrit en Matlab doit donc être un peu modifié pour pouvoir tourner sur Scilab. Nous essaierons autant que possible de donner les points de divergence entre les deux langages, afin de pouvoir utiliser les références contenant du code Matlab (par exemple [BC07] et l'excellent mais épuisé [Tou99]). Il y a aussi de bonnes références avec du code Scilab, comme [Dec10] et les fiches sur la page personnelle de Laurent Tournier (http://www.math.univ-paris13.fr/~tournier/enseignement.htm).

Exercice 1 Premier contact

1. Lancer Scilab et cliquer dans la console, puis entrer la commande `2 + 2` suivie de la touche entrée.

1. sauf que Python est, lui, un langage généraliste, ce qui a ses avantages et ses inconvénients

2. Créer un répertoire `tp_scilab` à l'emplacement de votre choix (par exemple dans `C:\Users\MonNom\` sous Windows, ou `/home/mon_login/` sous linux) en utilisant l'une des deux méthodes qui suivent.

   Avec l'interface graphique : Aller dans la partie « navigateur de fichiers » de l'interface graphique de Scilab. Si elle n'est pas présente, cliquer sur l'onglet Application du menu puis sur Navigateur de Fichiers. Naviguer dans l'arborescence pour se trouver dans le répertoire de votre choix, puis cliquer sur l'icône en haut à gauche puis sur « Nouveau Dossier ». Entrer le nom du dossier, puis s'y rendre.

   En ligne de commande : Dans la console, entrer par exemple :

```
--> cd("~")
--> mkdir("tp_scilab")
--> cd("tp_scilab")
```

3. Ouvrir l'éditeur, en cliquant sur ou bien en tapant `scinotes()` dans la console.
4. Reproduire les instructions suivantes puis taper sur `F5`. Enregistrer le fichier en le nommant par exemple `premier_script.sce`.

```
X = linspace(-2*%pi, 2*%pi, 200);
Y = sin(X);
plot(X, Y);
```

5. Après s'être remis de ses émotions, écrire le deuxième script suivant.

```
a = 2
b = 3
a + b
```

   Appuyer sur `F5`. Que s'est-il passé ? Ensuite, appuyer sur `Ctrl+l`. Que s'est-il passé ?
6. Mettre des `;` à la fin des deux premières lignes et rappuyer sur `Ctrl+l`.
7. Ajouter en quatrième ligne les commandes `disp("a + b ="); disp(a+b);` puis appuyer sur `F5`.
8. Sélectionner la première ligne puis appuyer sur `Ctrl+e`.
9. Dans la console, taper `help exp`, puis `apropos tan`[2].
10. S'il n'est pas déjà ouvert, ouvrir le « Navigateur de variables » dans l'onglet « Applications ».
11. Dans la console, entrer les commandes `clear` (puis entrée) puis `a` (et entrée).
12. Entrer la commande `clc`.

Récapitulons. Le logiciel Scilab comporte plusieurs applications dont les plus importantes sont la console et l'éditeur Scinotes[3]. Que l'on entre son programme comme une

2. si cela ne fonctionne pas, c'est que la documentation n'est pas installée, et il faut l'installer manuellement

3. mais si vous avez déjà vos habitudes avec un autre éditeur, rien ne vous empêche de l'utiliser à la place, vérifiez simplement qu'il sera bien disponible le jour de l'épreuve

suite d'instruction dans la console ou en exécutant un script, c'est en gros la même chose : Scilab exécute une à une les instructions qui lui sont transmises.

Une instruction doit se terminer par l'un des éléments syntaxiques suivants :

— un point-virgule (`;`) : dans ce cas, les affectations ne provoquent pas d'affichage.

— une virgule (`,`) : dans ce cas les affectations provoquent un affichage.

— un passage à la ligne : il y a là aussi affichage.

— un point-virgule et un passage à la ligne : sans affichage.

Ce comportement est modifié lorsqu'on exécute un script « sans echo » avec la touche `F5` depuis l'éditeur ou avec la commande `exec("mon_script.sce")` dans la console. Dans ce cas, les seuls affichages sont ceux demandés explicitement avec par exemple les fonctions `disp` ou `printf`. Pour exécuter un script avec echo, on appuie sur `Ctrl+l` pour exécuter tout le script ou on entre la commande `exec("mon_script.sce", 1)` dans la console. Noter également la possibilité de n'exécuter (avec echo) seulement une partie du script avec `Ctrl+e` (la sélection s'il y en a une, du début jusqu'au curseur, sinon). Attention, seul l'appui sur `F5` fait une sauvegarde automatique. On n'oubliera donc pas de sauvegarder régulièrement (par exemple en appuyant sur `Ctrl+s`).

Il me semble qu'une bonne pratique dans les scripts et les fichiers de fonctions (voir section 2) serait de systématiquement terminer toutes ses instructions par un point-virgule et un passage à la ligne et de n'utiliser que des affichages explicites (avec `disp` et `printf` par exemple) en exécutant toujours avec `F5`.

En Scilab, tout ce qui figure entre les caractères `//` et la fin de la ligne est ignoré[4], ce qui permet de commenter le code lorsque c'est nécessaire ou lorsque l'on ne veut exécuter qu'une partie d'un programme.

Comme le passage à la ligne a un rôle syntaxique (fin d'une instruction), il faut pour pouvoir écrire une instruction sur plusieurs lignes, dans la console ou dans l'éditeur, taper `..` avant la touche entrée.

```
--> x = 1 + 3 + 8 + ..
--> 9 + 10 // Ceci est un commentaire
    x =
       31.
```

1 Scilab comme calculatrice

Dans cette partie, on pourra utiliser au choix la console ou l'éditeur. Le tableau 1 donne les priorités des opérateurs dans Scilab[5] (source : https://rosettacode.org/wiki/Operator_precedence#Scilab, car ceci ne figure pas dans la documentation).

Exercice 2 Nombres et calcul, type `double`, opérations `+, -, *, /, ^`

1. Calculer :

$$\frac{\frac{2}{3}}{7} \quad \frac{2}{\frac{3}{7}} \quad \frac{2}{3\times 7} \quad \frac{2}{3}\times 6 \quad \left(2^3\right)^2 \quad 2^{3^2} \quad (-1)^2 \quad -1^2$$

4. `%` en Matlab
5. Attention, les priorités ne sont pas les mêmes dans Matlab !

TABLE 1 – Règles de priorités en Scilab

Priorité	Description	Opérateurs	Associativité
1	Parenthèses	()	
2	Conjugaison	’	
3	Puissance	^	à droite
4	Multiplication et division	* /	à gauche
5	Plus et moins unaires	+ -	à gauche
6	Addition et soustraction	+ -	à gauche
7	Comparaison	== ~= > < <= >=	à gauche
8	Non logique	~	à gauche
9	Et logique	&	à gauche
10	Ou logique	|	à gauche

2. À l’aide de `ans` (et de la touche ↑), calculer les 10 premières puissances de 2.
3. Calculer :

$$\sin(\%\mathsf{pi}) \quad \sqrt{2}^2 - 2 \quad \%\mathsf{eps} \quad 1 + \%\mathsf{eps} - 1 \quad 1 + \frac{\%\mathsf{eps}}{2} - 1 \quad 1 - 1 + \frac{\%\mathsf{eps}}{2}$$

$$2^{1023} \quad 2^{1024} \quad -\%\mathsf{e}^{800} \quad 2^{-1023} \quad 2^{-1024} \quad \frac{1}{0}$$

4. Calculer :

$$2 + \%\mathsf{inf} \quad \%\mathsf{inf} \times -\%\mathsf{inf} \quad \%\mathsf{inf} - \%\mathsf{inf} \quad \sin(\%\mathsf{inf}) \quad \sqrt{\%\mathsf{inf}}$$

Remarque 1. En Matlab, `%pi, %e, %eps, %inf, %nan, %t, %f` sont désignés respectivement par `pi, e, eps, Inf, NaN, 1, 0`. L’opération `1 / 0` ne provoque pas d’erreur mais simplement un avertissement et renvoie la valeur `Inf`. Le nombre $2^{-1023+1-30}$ (par exemple) est représentable sous une forme dite « sous-normale », ce que ne semble pas faire Scilab par défaut et le zéro est signé.

En Scilab, les opérateurs sur les booléens « et », « ou » et « non » sont respectivement `&`, `|` et `~`. Ils ont une priorité très basse (voir tableau 1). Les constantes « vrai » et « faux » sont `%t` et `%f`.

Exercice 3 Booléens, opérations `&, |, ~`

1. Tester les propositions suivantes :

$$2+2=4 \quad 2+2=5 \quad 2 \leq 2 \quad 2>3 \quad (1<2) \text{ et } (0=1) \quad (1>2) \text{ ou } (0=1)$$

$$\text{non } 2<3 \quad \text{non } 0=1 \quad \text{non (Vrai et Faux)}$$

2. Tester les codes suivants et expliquer les résultats obtenus :
   (a) `(2 + 2 == 4)* 15`
   (b) `2.5 & 15`
   (c) `0. | %f`
   (d) `3 | 1 / 0`

(e) `1 & %nan`

Remarque 2. En Matlab, en plus de `&, |`, on dispose d'opérateurs « court-circuits » `&&` (resp. `||`) qui n'évaluent pas le second opérande si le premier opérande est évalué à faux (resp. vrai). Enfin, la conversion de `NaN` en booléen est illégale en Matlab.

Exercice 4 Variables

1. Tester les codes suivants et expliquer les résultats obtenus.
   (a) `x = 4;`
   (b) `x = 4`
   (c) `x = 3, y = 7; z = 5`
   (d) `x = 1; y = x; y = y + 1; x, y`
   (e) `x = 1, typeof(x), x = %t, typeof(x)`
2. Calculer $\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}-1}+\frac{1}{\sqrt{2}+1}\right)$ en utilisant une variable intermédiaire.

Remarque 3. En Matlab, on remplace la fonction `typeof` par la fonction `class`. Le type `constant` de Scilab est remplacé par `double`.

2 Fonctions

Le plus souvent, on définit des fonctions dans des fichiers textes qui portent l'extension `.sci`. On les charge dans l'environnement de travail de la même façon qu'on exécute un script. Voici un exemple d'un tel fichier.

```
1  // fichier exemple_fonctions.sci
2
3  function valeur_de_retour = super_fonction(x1, x2)
4      valeur_de_retour = x1 + 3*x2 ;
5  endfunction
6
7  function [a, b] = autre_fonction(x)
8      a = x * 3;
9      b = x * x ;
10 endfunction
```

La ligne `function valeur_de_retour = super_fonction(x1, x2)` est le *prototype* de la fonction `super_fonction`. Ce prototype nous dit que `super_fonction` a deux *paramètres* et retourne une seul *valeur de retour*. Dans les lignes suivantes, qu'on appelle *corps de la fonction*, on doit donner une valeur à `valeur_de_retour` en utilisant les paramètres, des opérations, et ce que bon nous semble. Lorsqu'on atteint le mot-clé `endfunction`, la valeur de retour, dans son état actuel est retournée.

Voici un exemple d'utilisation de cette fonction :

```
1  --> exec("exemple_fonctions.sci")
2  --> y = super_fonction(1, 2)
3      y =
4          7.
```

La fonction `autre_fonction` retourne deux valeurs. Attention, si l'on ne précise rien lors de l'appel, seule la première valeur est retournée !

```
--> autre_fonction(3)
    ans =
        9.
```

Pour pouvoir accéder aux deux valeurs, il faut les demander explicitement avec la syntaxe suivante :

```
--> [triple, carre] = autre_fonction(3)
    carre =
        16
    triple =
        12
```

Exercice 5 `function [y1, ...] = ma_fonction(x1, ...); expr ; endfunction`

Programmer les fonctions suivantes dans l'éditeur et les tester.

On créera un fichier `tp1_ex4.sci` que l'on chargera dans l'interface avec la commande `exec("tp1_ex4.sci")`, ou bien avec la touche `F5` depuis l'éditeur.

1. La fonction `cube` ayant pour paramètre x et qui renvoie x^3.
2. La fonction `f` ayant pour paramètres x et y et qui renvoie $2 * x + 3 * y - 3$.
3. La fonction `g` ayant pour paramètre x et qui renvoie le couple $(\cos(x), \sin(x))$.
4. La fonction `hello` n'ayant aucun paramètre et qui affiche "Hello World !", suivi d'un passage à la ligne.

Bien que cela ne soit pas obligatoire, c'est une « bonne pratique » de mettre une extension `.sci` à un fichier Scilab ne contenant que des fonctions et `.sce` pour un script.

Le jour de l'oral (et les autres jours aussi), je vous conseille de créer un dossier (par exemple `chaines_markov`) dans lequel il y aura un fichier de fonctions (par exemple `chaines_markov.sci`) qui regroupera toutes les fonctions ou presque que vous aurez écrites et un script par expérience ou illustration (par exemple `marche_aleatoire.sce` et `temps_atteintes.sce`) qui chacuns commenceront par charger le fichier de fonctions. Le script `temps_atteintes.sce` utilisant les fonctions du fichier `chaines_markov.sci` (qui se trouve dans le même répertoire) pourra donc commencer de la façon suivante [6]

```
// fichier "temps_atteintes.sce"
// Simulations de temps d'atteintes pour la
// marche aléatoire symétrique dans Z^2
exec("chaines_markov.sci");
// pour la fonction rnd_marche_sym
disp("Attention les yeux");
depart = [ 0 0 ];
arrivee = [10 10];
position = depart;
temps = 0;
```

6. ne pas chercher à vraiment comprendre tout le programme maintenant

```
while or(position ~= arrivee)
    position = rnd_marche_sym(position);
    temps = temps + 1;
end
disp("temps = ");
disp(temps);
```

Remarque 4. *En Matlab*, il n'y a qu'une seule extension (`.m`) mais bien deux types de fichiers différents : les fichiers de fonctions qui ne contiennent que des fonctions mais dont seule la première est appelable et les fichiers scripts qui ne peuvent contenir des fonctions que depuis cette année (2016 !). En Matlab (2016b ou plus) ou Octave (toute version), on pourra donc obtenir le même (bon) comportement que Scilab avec l'astuce suivante :

```
% fichier mes_fonctions.m pour Octave ou Matlab 2016b+
% Plein de supers outils pour le calcul des puissances de 3
1; % On fait croire que c'est un fichier de script

function y = trois_puissance(n)
    y = 3^n;
end

% Une deuxieme fonction dans un fichier !
function y = trois_puissance_moins(n)
    y = 3^(-n);
end
```

On peut ensuite charger les fonctions avec `run("mes_fonctions.m")`, si l'on se trouve dans le même répertoire.

Exercice 6 Portée des variables et passage des variables par valeur

Que font les scripts suivants ?

1.

```
function incremente(x)
    x = x + 1;
endfunction
x = 3
incremente(x);
x
```

2.

```
function y = cree_carre_et_cube(x)
y = x*x;
cube = x*x*x;
endfunction
```

```
cree_carre_et_cube(45)
cube
```

3.

```
 a = 34;
function y = tres_sale(x)
    y = x + a;
endfunction
tres_sale(2)
a = 22
tres_sale(2)
```

4.

```
function afficher_et_annuler_a()
    disp(a);
    a = 0;
endfunction
a = 35;
afficher_et_annuler_a();
a
```

Récapitulons. En Scilab les arguments sont passés *par valeur* cela signifie que l'on travaille avec *une copie des valeurs des variables, et non les variables elles-mêmes*. C'est pourquoi, telle qu'elle est définie, la fonction `incremente` ne sert strictement à rien.

Le deuxième script illustre le fait que les variables définies à l'intérieur des fonctions sont *locales aux fonctions*. On ne peut donc pas y avoir accès après l'exécution de la fonction. Il faut utiliser les valeurs de retour.

Le troisième script montre qu'on peut, ce qui est presque toujours une très mauvaise idée, utiliser des variables globales, c'est-à-dire définies en dehors des fonctions, dans les fonctions.

Le dernier script montre qu'on ne peut pas modifier les variables globales. En cas de redéfinition ou modification, une nouvelle variable, locale elle, est créée et *masque* la variable globale portant le même nom.

Remarque 5. En Matlab, les variables globales doivent être déclarées comme telles au début du corps de la fonction (`global a;`).

En Scilab, cette même instruction (`global a;`) permet, en plus de la lecture possible par défaut, de modifier la variable globale a. Ces subtilités n'ont vraiment aucune importance, car à de très rares et très précises exceptions près, IL NE FAUT PAS UTILISER DE VARIABLE GLOBALE DANS LES FONCTIONS !

Dans le cas de fonctions dont le corps est très court, on peut préférer la syntaxe présentée dans l'exercice suivant [7].

7. En fait, cette construction permet surtout, avec quelques efforts, de définir dynamiquement des fonctions. Nous n'en parlerons plus tard que si cela s'avère vraiment nécessaire

Exercice 7 `deff("[y1, ...] = ma_fonction(x1, ...)", "expr")`
Reprendre les questions 1 et 2 de l'exercice 4 avec l'instruction `deff`.

Remarque 6. En Matlab, la syntaxe est `ma_fonction = @(x1, x2, ...)[...]`.

3 Structures de contrôle

Exercice 8 `if cond1; exp1; elseif cond2; exp2; else; exp3; end`

1. Programmer la fonction `absolue` ayant pour paramètre x et renvoyant sa valeur absolue.
2. Programmer la fonction suivante :

$$h(x) = \begin{cases} x+1 & \text{si } x \leq 1 \\ 2x & \text{si } 1 < x < 3 \\ x^2 & \text{sinon} \end{cases}$$

Remarque 7. On peut aussi, en Scilab, utiliser la construction avec des `then` :
`if cond1 then exp1 ; elseif cond2 then exp2 ; else exp3; end`

Exercice 9 `for compteur = debut:pas:fin ; expression ; end`

1. Calculer la somme suivante :
$$1^2 + 2^2 + \ldots + 30^2$$
2. Calculer la somme des cubes des nombres impairs inférieurs à 100.
3. Écrire une fonction qui prend en paramètre n et qui calcule u_n, défini de la façon suivante :
$$\begin{cases} u_0 &= 1 \\ u_{n+1} &= \sqrt{1+u_n} \end{cases}$$
4. Écrire une fonction qui calcule la somme suivante :
$$\sum_{i=1}^{n} \sum_{j=1}^{i} j$$
5. (Source : https://projecteuler.net/archives) Trouver la somme de tous les entiers entre 1 et 1000 qui sont multiples de 3 ou de 5. On pourra utiliser la fonction `modulo(x,n)`.

Exercice 10 `while condition ; expression ; end`

1. Calculer le plus petit entier N tel que $\ln(\ln(N)) \geq 2$.
2. Soit $\ell = \frac{1+\sqrt{5}}{2}$. Calculer le plus petit entier N tel que $|\ell - u_N| \leq 10^{-6}$, où (u_n) est la suite définie lors de l'exercice précédent.

On peut également utiliser les fonctions récursives, c'est-à-dire qui s'appellent elles-mêmes. En voici un exemple complètement inutile.

```
function y = compte(n)
    if n == 0
        y = 0;
    else
        y = 1 + compte( n-1 );
    end
endfunction
```

Exercice 11 Récursivité

En utilisant une fonction récursive, c'est-à-dire qui s'appelle elle-même, écrire une fonction `facto` qui calcule la factorielle d'un entier donné en argument.

Pour conclure cette partie, voici un exemple de programme assez idiot utilisant les instructions `break` (qui permet de sortir immédiatement d'une boucle) et `continue` (qui permet d'aller directement au prochain passage de la boucle) dans une boucle infinie.

```
// Super-calculateur de racines carrées
while 1 // 1 est converti en %t
    reponse = input("Voulez-vous que je calcule ..
        une racine carrée ? (0 pour non)")
    if reponse == 0
        break;
    end
    x = input("Donnez votre nombre :");
    if x < 0
        disp("Nombre négatif, pas de racine réelle");
        continue;
    end
    rac = sqrt(x);
    disp("Sa racine carrée vaut :");
    disp(rac);
end
```

4 Vecteurs et matrices

Exercice 12 `A = [ a11 a12 ... a1n ; a21 ... a2n ; am1 ... amn ]`

1. Dans un script, créer la matrice $A = \begin{bmatrix} 3 & -1 & 2 & 5 \\ 1 & 0 & -10 & 3 \\ -1 & 1 & 3 & 10 \end{bmatrix}$.
2. Taper les commandes suivantes, commenter.
   (a) `A(2,3)`
   (b) `size(A)`
   (c) `typeof(A)`
   (d) `length(A)`
   (e) `size(A,'c')`

(f) `size(A, 'r')`
(g) `size(A, '*')`
(h) `A(1, $)`
(i) `A($, 2)`
(j) `A(12, 76)`

3. Même question pour :
   (a) `A(1, 2)= 3`
   (b) `A(1, $)= 78`
   (c) `A(4, 2)= 52`
4. Même question pour :
   (a) `A'`
   (b) `sum(A, 'r')`
   (c) `sum(A, 'c')`
   (d) `sum(A)`
   (e) `cumsum(A)`
   (f) `cumprod(A,'c')`

Remarque 8. En Matlab, le symbole `$` est remplacé par `end` et `sum` fait par défaut la somme colonne par colonne. Enfin, la commande `reshape` remplace `matrix`.

Remarque 9. En Scilab et en Matlab, le premier indice dans un tableau est `1`, contrairement à C ou Python (où c'est `0`).

Exercice 13 Création de Matrices

1. Taper les commandes suivantes, commenter.
   (a) `zeros(2, 6)`
   (b) `zeros(5)`
   (c) `zeros(A)`
   (d) `ones(2, 6)`
   (e) `eye(3,3)`
   (f) `rand(3,5)`
   (g) `diag([1 2 3])`
   (h) `diag(A, -1)`
   (i) `diag(diag(A))`
2. Même question avec
   (a) `1:10`
   (b) `2:3:20`
   (c) `1:.1:%pi`
   (d) `5:-1:0`
   (e) `linspace(0, 1, 10)`

(f) `matrix(1:10, 2, 5)`

3. Même question avec
   (a) `B = [ A [1 2 ; 3 4 ; 5 6] ]`
   (b) `A = [ A ; 1:4 ]`

Remarque 10. En Matlab, `zeros(5)` donne une matrice carrée de taille 5×5, `zeros([2 3])` donne une matrice 2×3, et `zeros(A)` ne fonctionne que si `A` est une matrice ligne comme précédemment.

Exercice 14 Opérations sur les matrices

1. Entrer à nouveau la matrice A de l'exercice précédent. Entrer également les matrices B, C et D ci-dessous :

$$B = \begin{bmatrix} 1 & 10 & 0 & 2 \\ 3 & -1 & 1 & 4 \\ 2 & -1 & 0 & 2 \end{bmatrix} \quad C = \begin{bmatrix} 2 \\ 4 \\ -1 \\ 1 \end{bmatrix} \quad D = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 5 & 0 \\ -1 & 3 & 2 \end{bmatrix}$$

2. Taper les commandes suivantes et commenter :
   (a) `A + B`
   (b) `A + C`
   (c) `2 * D`
   (d) `3 * A - 2 * B`
3. Taper les commandes suivantes et commenter :
   (a) `A .* B`
   (b) `A .* C`
   (c) `A .^ 2`
   (d) `B ./ A`
   (e) `C ./ D`
4. Même question pour :
   (a) `A * B`
   (b) `A * C`
   (c) `D * B`
   (d) `D ^ 2`
   (e) `B ^ 2`
5. Même question pour :
   (a) `exp(A)`
   (b) `sin(C)`

Exercice 15 `timer()`

1. Calculer en une ligne de code, 17 caractères, la somme
$$1^2 + 2^2 + ... + 10000^2$$
2. Comparer le temps de calcul avec la solution utilisant une boucle `for`.

Exercice 16 Découpage

On considère de nouveau la matrice A de l'exercice 9.

1. Sans jamais entrer de coefficient, faire apparaître dans la fenêtre de commande les matrices suivantes :
$$\begin{bmatrix} 3 & -1 \\ 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 2 \\ 1 & 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & -10 & 3 \end{bmatrix} \quad \begin{bmatrix} 2 \\ -10 \\ 3 \end{bmatrix}$$
2. Remplacer la première ligne de A par sa troisième ligne.
3. Remplir la dernière colonne de A par des 42.
4. Entrer les instructions suivantes, puis commenter :
   (a) `A(:,$-1)= [ ]`
   (b) `A($+1,:)= 3`
   (c) `A($+1,:)= ones(1, size(A,'c'))`
   (d) `A = [2*ones( 1, size(A, 'c')); A]`
   (e) `A(:)`
   (f) `A(2:( size(A, 'r')+ 1 ):length(A))= - 42`

Remarque 11. Une affectation du type `B = A(1,:);` est une *copie profonde*. Une modification ultérieure des éléments de `B` ne modifiera pas `A`. Ce comportement est le choix de Scilab et Matlab, mais ce n'est pas canonique (c'est différent en Python par exemple).

Exercice 17 Indexation logique

Entrer la matrice `T = [-1 3 1 5 -4]`. Que font les instructions suivantes ?

1. `T > 0`
2. `T < 0 | T == 5`
3. `find(T < 0)`
4. `T( find(T < 0))`
5. `T( T < 0 )`

Nous avons vu dans cette partie que Scilab possède plusieurs outils permettant de créer des matrices et de faire des opérations sur celles-ci sans utiliser de boucle. On reconnaît un bon utilisateur de Scilab/Matlab/... [8] au nombre faible (voire nul) de boucles utilisées. Les solutions dites « vectorisées » sont toujours beaucoup plus rapides que celles contenant des boucles. Elles sont aussi souvent plus lisibles.

Mais on n'oubliera jamais non plus qu'il vaut infiniment mieux écrire un programme imparfait qui fonctionne à peu près, que pas de programme du tout.

Récapitulons les diverses constructions. On considère une matrice $A = (a_{i,j})$ de taille $m \times n$ et un vecteur colonne V de taille n.

8. ce que l'auteur de ces lignes ne prétend absolument pas être

— *L'indexation simple* (comme par exemple dans `A(3)` ou dans `A(3:$)`) permet de voir A comme un seul vecteur colonne, dans l'ordre dit « column-major », c'est-à-dire comme

$$(a_{11}, a_{21}, ..., a_{m1}, a_{12}, a_{22}, ..., a_{m2}, a_{13}, \ldots, a_{mn})$$

— *L'extraction* (au sens large)[9] d'une matrice est mathématiquement définie en considérant deux applications $\varphi : \{1, \ldots, p\} \to \{1, \ldots, m\}$ et $\psi : \{1, \ldots, q\} \to \{1, \ldots, n\}$ puis la matrice

$$A^{(\varphi,\psi)} = \left(a_{\varphi(i),\psi(j)}\right)_{1\leq i\leq p,\ 1\leq j\leq q}$$

C'est ce qui est fait dans Scilab lorsqu'on écrit par exemple `A( [2 3 1 1], [1 3 2] )`, ou `V(2:2:$)`.

— L'instruction « `matrice_extraite_de_A = matrice_de_meme_taille` » modifie les éléments de A présents dans cette matrice extraite, comme par exemple dans l'instruction `A( [1 3], [2 4 6] )= [-1 2 1 ; 1 0 0]`

— L'instruction « `matrice_extraite_de_A = nombre` » rend tous les éléments de A apparaissant dans la sous-matrice égaux au `nombre`, comme par exemple dans `A( 1:2:$ , : )= 0` .

— On peut combiner l'indexation simple avec les deux dernières constructions comme par exemple dans `A ( 2:2:$ )= 0`

— *L'indexation logique* s'applique en premier lieu à un vecteur. `V ( V > 0 )` permet d'extraire le sous-vecteur de V formé des coefficients strictement positifs. Si l'on entre `A ( A > 0 )`, la matrice A est implicitement convertie en vecteur colonne comme dans l'indexation simple.

Un exemple pour finir : la fonction `partie_positive(A)` met tous les coefficients négatifs de A à 0. Nous allons la programmer de deux façons différentes.

```
// Partie positive avec des boucles
function Y = partie_positive(X)
    [m, n] = size(X);
    Y = X;
    for i = 1:m
        for j = 1:n
            if Y(i,j) <= 0
                Y(i,j) = 0;
            end
        end
    end
endfunction
```

```
//Partie positive vectorisée
function Y=partie_positive(X)
    Y = X;
    Y( Y<0 ) = 0;
endfunction
```

9. terminologie non canonique

Exercice 18 `modulo(x,n)`

Calculer, sans boucle et avec une seule ligne de code la somme de tous les entiers entre 1 et 10000 qui sont multiples de 3 ou de 5.

Exercice 19 Gammes

1. Écrire les tables de multiplication sous la forme d'une matrice 10×10.
2. Changer la troisième colonne de façon à avoir la table de 12 à la place.
3. Supprimer la troisième colonne.
4. Remettre la table de 3 dans la troisième colonne.
5. Ajouter une ligne, de façon à avoir les multiples de 11.

Exercice 20 Gammes (2)

1. Creér une matrice 5×3 dont toutes les lignes sont égales à `[2 5 1]`.
2. Mettre la troisième ligne à 0.
3. Mettre la diagonale à 3.
4. Changer la matrice en une matrice 3×5.

Exercice 21 Gammes (3)

1. Construire un vecteur ligne T de 20 nombres aléatoires entre 0 et 1.
2. Construire une matrice ayant 5 lignes toutes égales à T.
3. Construire une matrice ayant 4 colonnes toutes égales à T.
4. Calculer la valeur moyenne des éléments de T.
5. Compter le nombre d'éléments de T qui sont inférieurs à $0,1$.
6. Construire la matrice ligne U de taille 20 telle que,

$$\forall i \in \{1,2,...,20\}, \quad U_i = \begin{cases} 0 & \text{si } T_i \leq 0,2 \\ 1 & \text{si } 0,2 \leq T_i \leq 0,5 \\ 2 & \text{sinon.} \end{cases}$$

Exercice 22 Gammes (4)

En ne faisant que des opérations matricielles, créer les vecteurs suivants :

$$\left(\frac{1}{k^2}\right)_{1\leq k\leq 10} \quad \left(\exp\left(1+\frac{k}{10}\right)\right)_{0\leq k\leq 10} \quad (2k+\sin(k))_{0\leq k\leq 10}$$

Exercice 23 Gammes (5)

1. Créer un vecteur ligne de taille 20 dont les 10 premiers éléments sont des 0 et les 10 suivants sont des 1.
2. Créer un vecteur ligne de taille 20 dont les éléments sont successivement $+1$ et -1.

3. Créer une matrice carrée de taille 10, triangulaire supérieure, dont tous les coefficients non nuls sont égaux à 1.

En Scilab, les fonctions sont des variables commes les autres. Le script suivant donne bien le résultat attendu.

```
function y = valeur_en_0(f)
    y = f(0);
endfunction

valeur_en_0(exp)
```

Exercice 24 Intégrale

Écrire une fonction `integrale(f,a,b,n)` qui calcule une valeur approchée de $\int_a^b f(t)\,\mathrm{d}t$ en utilisant une somme de Riemann comportant n termes.

5 Algèbre linéaire

Pour résoudre un système de la forme $AX = b$, où A est une matrice et b un vecteur colonne, on utilise au choix `X = A\b` ou bien `X = linsolve(A, -b)`. Pour obtenir une base du noyau, on utilise `kernel` [10] et la fonction `spec` [11] permet de connaître les valeurs propres et une base de vecteurs propres. Voir la documentation.

Exercice 25 Système

Résoudre le système suivant :

$$\begin{cases} x - y + 2z &= 3 \\ 3x - 5y + 3z &= 1 \\ x + y - 3z &= -2 \end{cases}$$

Exercice 26 Diagonalisation

Diagonaliser une matrice `A = rand(3,3)` à l'aide de la fonction `spec`. Vérifier le résultat (on pourra avoir besoin de la fonction `clean`).

Scilab comporte un grand nombre de fonctions classiques de l'algèbre linéaire. Parmi elles, mentionnons :

inv Inverse d'une matrice carrée

rank Rang d'une matrice

cond Conditionnement de la matrice

lu Factorisation LU (pour « Lower triangular, Upper triangular »)

qr Factorisation QR (Q orthogonale, R triangulaire supérieure)

chol Factorisation de Cholesky

10. `null` en Matlab
11. `eig` en Matlab

6 Graphiques

On trace les courbes avec la commande `plot`. En général, on donne à `plot` deux vecteurs ligne X et Y de même taille, indiquant les abscisses et les ordonnées des points à placer et à relier. Le document
http://www.openeering.com/sites/default/files/Plotting_in_Scilab.pdf
est un assez bon résumé des commandes permettant de dessiner des courbes.

Le script suivant permet d'obtenir la figure 1.

```
//Tracé des courbes des fonctions exp et ln
// 1000 valeurs entre .001 et 3
X1 = linspace(0.001, 3, 1000);
// les 1000 images des valeurs précédentes
Y1 = log(X1);
scf(1); // SCilab Figure : ouvrir la figure 1
clf(1); // CLear Figure : effacer la figure 1
// On trace la première courbe :
plot(X1, Y1);
// Get Current Axis : acceder aux propriétés des axes
a = gca();
a.x_location = "origin"; // Tracer l'axe x=0
a.y_location = "origin"; // Tracer l'axe y=0

xlabel(a, "$\text{Les abscisses}$", ..
"fontsize", 5, "position", [1.5 -1]);
ylabel(a, "$\text{Les ordonneés}$",..
    "fontsize", 5, "color", "r", ..
    "position", [-.2 2])
title(a, ..
    "$\text{Les fonctions } \exp \text{ et } \ln $", ..
    "fontsize", 5)
// Pour le latex, c'est tout ou rien :
// soit "$ formule latex $", soit "du texte"

// 2000 valeurs entre -3 et 3
X2 = linspace(-3, 3, 2000);
// Images par exp de ces 2000 valeurs
Y2 = exp(X2);
// On trace en rouge la courbe de exp
plot(X2, Y2, 'r');
// Première bissectrice
plot([-10, 10], [-10, 10], 'k--');
// tracée avec deux points (-10, -10)
// et (10, 10) reliés par des
// traits interrompus(--), en noir(blacK)

a.grid = [5 5];
// grille active pour x (premier 5) et y (deuxième 5)
```

```
40 // La valeur -1 enlève la grille,
41 // des valeurs différentes donnent des
42 // couleurs différentes (3 est vert, ...)
43 a.data_bounds = [-3 -5 ; 3 5]; // Changement d'échelle
44 // sous la forme [xmin ymin ; xmax ymax]
45 legend(a, "$y=\ln(x)$", "$y=\exp(x)$", [-3 6]);
```

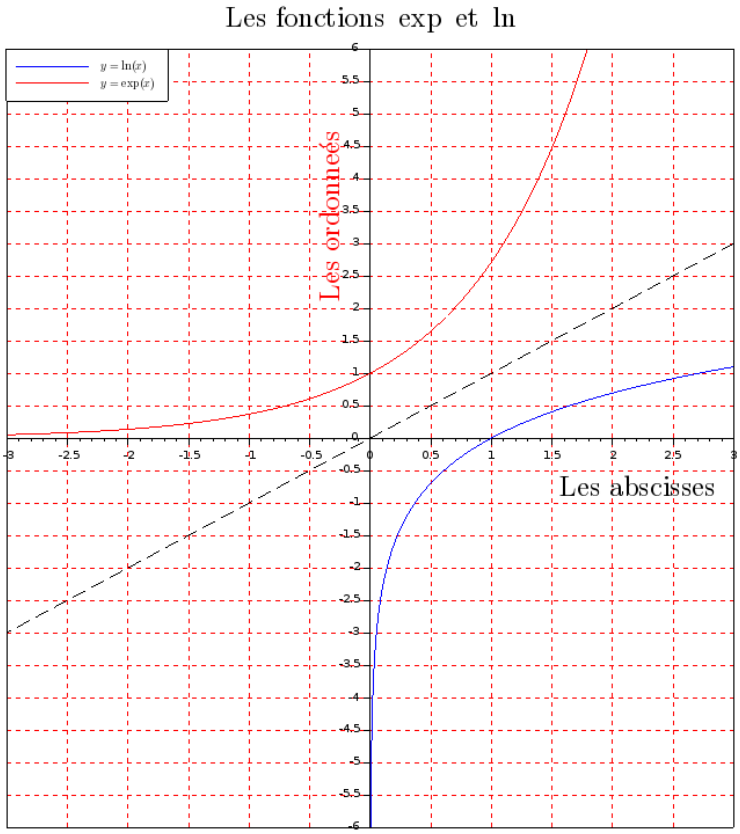

FIGURE 1 – Figure obtenue dans Scilab

Oui, ça fait un peu peur... On essaiera tout de même de retenir au moins la base :

```
1 X = linspace(a,b,n);
2 Y = f(X);
3 scf(1); clf(1);
4 plot(X, Y);
```

Les différents paramètres de la figure (titre, légende, etc) sont aussi accessibles via le menu « Édition » de la fenêtre « Figure ».

On peut supprimer la figure `k` avec la commande `xdel(k)`. La commande `winsid` permet d'obtenir la liste des identifiants des fenêtres graphiques. L'instruction `xdel(`↩

`winsid()` efface et ferme donc toutes les fenêtres graphiques.

Exercice 27

1. Tracer les courbes des fonctions $x \mapsto \sin(x)$ et $x \mapsto \cos(x)$ pour x dans $[-\pi; \pi]$.
2. Tracer la courbe de la fonction $x \mapsto x^3 - 3x^2$ pour x dans $[-10; 10]$, sans rien changer d'abord, puis pour y dans $[-100; 100]$.

Exercice 28 Tracer le cercle de centre l'origine et de rayon 1, ainsi que les segments $[AB]$, $[AC]$ et $[BC]$, où A est le point de coordonnées $(0, 1)$ et B et C sont les points du cercle d'ordonnée $-\frac{1}{2}$.

Exercice 29 Tracer la cycloïde, d'équation

$$\begin{cases} x(t) = t - \sin t \\ y(t) = 1 - \cos t \end{cases} \quad \text{pour } t \in [0\,;10].$$

Exercice 30 Tracer la cardioïde, d'équation polaire

$$\rho(\theta) = 1 + \cos(\theta).$$

Pour tracer un histogramme on utilise la fonction `histplot` [12] avec l'une des syntaxes suivantes :

classes automatiques `histplot(n,X)` où `n` est le nombre de classes et `X` le vecteur contenant les données.

classes définies par l'utilisateur `histplot(C,X)` où $C = (c_1, c_2, \ldots, c_k)$ définit les classes $[c_1\,;c_2]$, $]c_2\,;c_3]$, ..., $]c_{k-1}\,;c_k]$

On notera que l'histogramme est par défaut un histogramme de fréquences. Pour avoir un histogramme d'effectifs, on rajoutera à la fin des arguments `normalization=%f`.

Voici un exemple de script avec un histogramme. On remarquera l'appel (nécessaire) à la fonction `stacksize` pour augmenter la taille de la pile.

```
1  // Il manque un commentaire important
2  // pour dire à quoi sert ce script.
3  // A vous de l'écrire.
4
5  // Augmentation de la taille de la pile
6  stacksize('max');
7  // Nombre de termes dans chaque somme
8  N = 1000;
9  // Nombre de sommes
10 n = 5000;
11
```

12. En Matlab, la fonction à appeler est `hist`.

```
12 U = rand(N,n)
13 S = sum(U, 'r');
14 S = S - N*0.5;
15 S = S / sqrt(N);
16
17 scf(2);
18 clf(2);
19 histplot(40, S);
20
21 sigma_carre = 1/12;
22 X = linspace(-2,2,1000);
23 Y = 1 / sqrt(2*%pi*sigma_carre) * exp( - X.^2 / (2*↩
     sigma_carre) );
24 plot(X, Y, 'r');
```

L'histogramme que j'ai obtenu est celui de la figure 2.

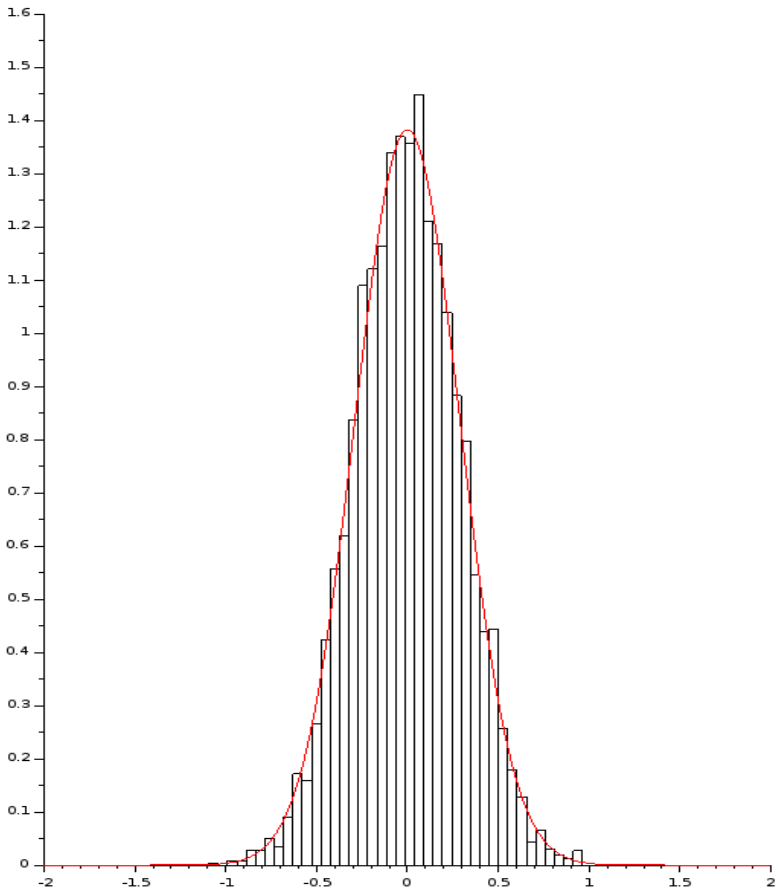

FIGURE 2 – Un histogramme et une courbe

Exercice 31

Créer une fonction `de_6_faces(n)` qui retourne un vecteur de taille n simulant n lancers d'un dé bien équilibré. Tracer des histogrammes des valeurs de retour pour $n = 10^k$, avec $k \in \{2, 3, 4, 5\}$.

Quel est le théorème illustré ?

7 Pour s'entraîner

Exercice 32 Test de primalité

Écrire une fonction `est_premier(n)` qui renvoie vrai (`%t`) si et seulement si n est entier naturel premier.

Exercice 33 (Source : [Dec10])

Écrire une fonction `triangle_pascal(n)` qui renvoie une matrice de taille $(n, n+1)$, complétée par des 0.

Exercice 34 (Source : [Dec10])

Écrire une fonction renvoyant la liste de tous les nombres premiers inférieurs ou égaux à son argument en utilisant la méthode du crible d'Ératosthène. Représenter graphiquement la fonction

$$x \mapsto \frac{\pi(x)\ln(x)}{x},$$

où $\pi(x)$ est le nombre de nombres premiers inférieurs ou égaux à x.

Exercice 35 (https://projecteuler.net/problem=2)

Trouver la somme des termes pairs de la suite de Fibonacci inférieurs ou égaux à 4 millions. Même question sans boucle ni récursivité.

Exercice 36 Formule de Stirling (d'après [Dec10])

Écrire la suite des quotients

$$q_n = \frac{n^n e^{-n}\sqrt{2\pi n}}{n!}$$

pour $n = 1, \ldots, 1000$, et représenter graphiquement la suite (q_n). Idem avec l'approximation de $n!$ par

$$n^n e^{-n}\sqrt{2\pi n}\left(1 + \frac{1}{12n}\right).$$

Exercice 37 Matrice de Vandermonde, matrice circulante

Écrire une fonction `vandermonde(V)` qui retourne la matrice de Vandermonde associée aux coefficients du vecteur V.

Faire de même pour la matrice circulante associée au vecteur V.

Exercice 38 Laplacien discret

Soit f une fonction deux fois dérivables sur un intervalle $[a\,;b]$ et $h > 0$. On appelle laplacien discret de f en $x \in]a\,;b[$ d'écart h le nombre (qui a un sens dès que h est assez petit) :

$$\Delta_h^{\text{disc}} f(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2}.$$

Résoudre de façon approchée l'équation différentielle $y'' = 0$ sur $[0\,;1]$ avec conditions aux bords $y(0) = y(1) = 1$ en approchant la dérivée seconde par le laplacien discret d'écart $\frac{1}{n}$.

Comparer avec la vraie solution.

Exercice 39 Coniques

1. Tracer l'ellipse de foyers $F(-1,0)$ et $F'(1,0)$ et de demi grand-axe $a = 2$.
2. Tracer l'hyperbole d'équation cartésienne
$$x^2 - y^2 = 1.$$
3. Tracer la parabole de foyer F et dont la directrice est l'axe des ordonnées.

Corrigés avec remarques des exercices

Exercice 2

```
/// exercice 2 : calculs en virgule flottante
// executer avec Ctrl+l, ou Ctrl+e en
// sélectionnant la ou les lignes que l'on veut

// Question 1
2 / 3 / 7 // associativité à gauche
2 / (3 / 7)
2 / (3 * 7)
2 / 3 * 6 // associativité à gauche (mais ici les parenthèses
//           peuvent rendre le code plus clair)
(2^3)^2 // associativité à droite, parenthèses nécessaires
2^3^2 // associativité à droite
(-1)^2 // parenthèses nécessaires : le - unaire a
// une plus petite priorité
-1^2 // donne -1

// Question 2
// dans la console, taper 2 puis entrée
// puis 2*ans puis entrée
// puis flèche haut, entrée, flèche haut, entrée, etc

// Question 3
sin(%pi) // n'est pas == 0 car on travaille avec une approximation
```

```
// de pi et une approximation de sin
sqrt(2)^2 - 2 // ne peut pas être égal à 0 car sqrt(2) est une
// approximation du réel racine de deux.
%eps // = 2^(-52) s'appelle le epsilon machine, c'est la différence
// entre 1 et le plus petit double > 1
1 + %eps - 1 // donne %eps, ok
1 + %eps/2 - 1 // associativité à gauche,
// 1 + %eps/2 est évalué à 1, - 1 cela donne 0
1 - 1 + %eps/2 // associativité à gauche :
// 1 - 1 est évalué à 0, + %eps/2 cela donne %eps/2
2^1023 // encore représentable en double
2^1024 // non représentable en double, renvoie %inf
-%e^800 // - %inf
2^-1023 // encore représentable en double normal
2^-1024 // non représentable en double normal
// Scilab renvoie 0
1 / 0 // scilab donne une erreur (ne renvoie pas %inf)

// Question 4
// C'est ce à quoi on s'attend, avec le bon comportement de
%inf - %inf // qui donne %nan (not a number, forme indéterminée)
sin(%inf) // qui donne aussi %nan/
```

Exercice 3

```
 // exercice 3 : calculs booléens

// question 1
2+2 == 4
2+2 == 5
2 <= 2
2 > 3
1 < 2 & 0 == 1 // la priorité basse de & et de | est pratique
1 > 2 | 0 == 1
~ 2>3
~ 0 == 1
~ (%t & %f) // parenthèses obligatoires

// question 2
(2 + 2 == 4) * 15 // donne 1 * 15 = 15 : conversion de type entre
// booléen et double : %t donne 1 et %f donne 0
```

```
2.5 & 15 // conversion de type entre double et booléen
// tout double non nul est converti en %t (même %nan, ce qui
// est à mon avis contestable)
0. | %f // conversion de 0. en %f
3 | 1/0 // les deux membres sont évalués, pas de court-circuit
1 & %nan // %nan est converti en %t
```

Exercice 4

```
// exercice 4

// Question 1

x = 4; // affectation de 4 à x, pas d'affichage
x = 4 // affectation + affichage
x = 3, y = 7; z = 5 // affichages pour x et z, pas pour y
x = 1; y = x; y = y + 1; x, y // très important
// Ici on voit le comportement de scilab dans les
// copies : toujours copie profonde (on copie
// la *valeur* de x dans y)
x = 1, typeof(x), x = %t, typeof(x)
// ici on voit la propriété de *typage dynamique*
// x est au départ un *double* (ce que Scilab
// appelle *constant*) puis c'est un *booléen*.

// Question 2
rac2 = sqrt(2);
(1 / (rac2 - 1) + 1 / (rac2 + 1)) / rac2
```

Exercice 5

```
function y = cube(x)
    y = x * x * x;
endfunction

function z = f(x, y)
    z = 2*x + 3*y - 3;
endfunction

function [a, b] = g(x)
    a = cos(x);
    b = sin(x);
```

```
endfunction

function hello()
    disp("Hello World!");
endfunction
```

Exercices 8 à 11

```
// exercice 8 sur if else
// question 1
function y = absolue(x)
    if x < 0 then
        y = -x;
    else
        y = x;
    end
endfunction
// question 2
function y=h(x)
    if x <= 1 then
        y = x + 1;
    elseif x > 1 & x < 3
        y = 2*x;
    else
        y = x^2;
    end
endfunction

// exercice 9 sur les boucles for
// question 1
function s = somme_carres(n)
    s = 0;
    for i = 1:1:n // ou i = 1:30 le pas par défaut est 1
        s = s + i^2;
    end
endfunction
// question 2
function s = somme_cubes_impairs(n)
    s = 0;
    for i = 1:2:n
        s = s + i^2;
    end
endfunction
// question 3
function u = u_n(n)
    u = 1;
```

```
    for i = 1:n
        u = sqrt(1 + u);
    end
endfunction
// question 4
function s = triangulaire(n)
    s = 0;
    for i = 1:n
        for j = 1:i
            s = s + i
        end
    end
endfunction
// question 5
function s = som_mult_3_ou_5(n)
    s = 0;
    for i = 1:n
        if (modulo(n,3) == 0 | modulo(n,5) == 0)
            s = s + i;
        end
    end
endfunction

// exercice 10 sur while
// question 1
function N = log_log(x)
    N = 2;
    while log(log(N)) < x
        N = N + 1;
    end
endfunction
// question 2
function N = tps_nb_or(epsilon)
    N = 0;
    u = 1;
    ell = (1 + sqrt(5)) / 2;
    while abs(ell - u) > epsilon
        u = sqrt(1 + u);
        N = N + 1;
    end
endfunction
// exercice 11 : récursivité
function y = facto(n)
    if n ==0 then
        y = 1;
    else
        y = n * facto(n-1);
    end
```

```
87 endfunction
```

Exercice 15

```
1  // solution vectorisée
2  timer();
3  sum((1:10000).^2)
4  timer()
5
6  // solution non vectorisée
7  s = 0;
8  for i = 1:10000
9      s = s + i*i;
10 end
11 timer()
```

7.1 Exercice 14

```
1  // Exercice 14
2  A = [ 3 -1 2 5 ; 1 0 -10 3 ; -1 1 3 10];
3  B = [1 10 0 2 ; 3 -1 1 4 ; 2 -1 0 2];
4  C = [2 ; 4 ; -1 ; 1]
5  D = [1 0 -1 ; 2 5 0 ; -1 3 2];
6
7  A + B // addition élément par élément
8  A + C // erreur de dimensions
9  2 * D // multiplication par un scalaire
10 3 * A - 2 * B // combinaison linéaire
11
12 A .* B // produit élément par élément
13 A .* C // erreur de dimension
14 A .^ 2 // puissance élément par élément
15 B ./ A // division élément par élément
16 C ./ D // erreur de dimension
17
18 A * B // produit matriciel : erreur de dimensions
19 A * C // produit matriciel
20 D * B // produit matriciel
21 D ^ 2 // puissance matricielle
22 B ^ 2 // erreur de dimension
23
24 exp(A) // exponentielle, élément par élément
25 // pour l'exponentielle de matrice (carrée) voir expm
26 sin(C) // sinus élément par élément
```

7.2 Exercice 16

```
// Exercice 16 : découpage
A = [ 3 -1 2 5 ; 1 0 -10 3 ; -1 1 3 10];

A([1 2] , [1 2])
A(1:2, 1:2)
A([1 3], [2 3])
A(2, :)
A(:, 3)

A(1, :) = A (3, :)
A(:,$) = 42

A(:, $-1) = [] // Suppression d'une colonne
A($ + 1 , :) = 3 // rajout d'une ligne de trois
A($+1, : ) = ones(1, size(A, 'c') ) // ajout d'une ligne
A(:) // A sous forme de vecteur colonne
A(2: (size(A, 'r') + 1):length(A) ) = - 42
// Mise à -42 de la 1ère sous-diagonale de A
```

7.3 Exercice 17

```
// Exercice 17
T = [-1 3 1 5 -4]
T > 0 // matrice de booléens de même taille que T
T < 0 | T == 5 // opérations logiques sur les matrices booléennes
find(T < 0) // renvoie les indices correspondant à True
T( find (T < 0) ) // éléments négatifs de T
T ( T < 0 ) // raccourci à utiliser sans modération
```

7.4 Exercice 18

```
// Exercice 18
// Méthode vectorisée
timer()
A = 1:10000;
som = sum( A( modulo(A, 3) ==  0 | modulo(A, 5) == 0 ) )
printf("somme = %f", som)
timer()

// Méthode plus traditionnelle avec boucle
```

```
somme = 0;
for i = 1:10000
    if modulo(i, 3) == 0 | modulo(i, 5) == 0
        somme = somme + i;
    end
end
somme
timer()
```

7.5 Exercice 19

```
// Exercice 19
// Pour créer la table
M = (1:10)' * (1:10)
// On change la troisième colonne
M(:,3) = 12 * (1:10)'
// On supprime la troisième colonne
M(:,3) = []
// On remet la troisième colonne d'origine
// Concaténation horizontale
M = [M(:, 1:2) , 3 * (1:10)' , M(:, 3:$)]
// Ajout d'une ligne
M($+1, :) = 11 * (1:10)
```

7.6 Exercice 20

```
// Exercice 20
// Création de la matroice
A = ones(5, 1) * [2 5 3]
// On change la troisième ligne
A(3, :) = 0
// On change la diagonale
// avec une indexation *simple*
A(1:(size(A, 'r') + 1):$) = 3
// redimensionnement
A = matrix(A, 3, 5)
```

7.7 Exercice 21

```
// Exercice 21
T = rand(1, 20)
// 5 ligne égales à T
```

```
A = ones(5, 1) * T
// 4 colonnes égales à T
B = T' * ones(1, 4)
// moyenne des coefficients
sum(T) / length(T)
// ou size(T, '*') à la place de length(T)
// Nombre de coefficients < 0.1
sum(T < 0.1)
// indexation logique + conversion implicite
// de booleen en nombre
U = (T >= .2) + (T > .5)
```

7.8 Exercice 22

```
// Exercice 22
K = 1:10;
A = 1 ./ K .^ 2
K = 0:10;
B = exp( 1 + K / 10 )
X = 0:10 ;
C = 2 * K + sin(K)
```

7.9 Exercice 23

```
// Exercice 23
A = [zeros(1, 10), ones(1, 10)]
B = ones(1, 20);
B (2:2:$) = -1; B

// Matrice triangulaire supérieure
// Première solution
// vectorisée mais hideuse
C = ones(10, 10)
ii = (1:10) ; jj = 1 ./(1:10)' ;
C((jj * ii < 1) ) = 0 ;
// Deuxième solution
// Plus jolie mais avec une boucle
CC = zeros(10,10);
for k = 1:10
    CC(k, k:$) = 1
end
// Troisième solution
// En fait il y avait une fonction prédéfinie ...
CCC = triu(ones(10, 10)
```

7.10 Exercice 24

```
function I = integrale(f, a, b, n)
    // Bords des rectangles
    X = linspace(a, b, n);
    // largeur de chaque rectangle
    dX = X(2) - X(1);
    // J'enlève la dernière valeur
    // Sinon il y a un rectangle de trop
    I = sum( (f(X(1:$-1)) * dX) )
endfunction

// test : on calcule pi
function y = g(x)
    y = 1 ./ (1 + x .^ 2)
    // Remarques au passage :
    // Attention au '1./' qui est confus pour scilab
    // On cherche toujours à rendre les fonctions
    // compatibles avec les matrices (* -> .* , ...)
endfunction

4 * integrale(g, 0, 1, 5000)
// On trouve une valeur approchée de pi
```

7.11 Exercice 25

```
// Exercice 25
M = [1 -1 2 ; 3 -5 3 ; 1 1 -3];
// M est-elle inversible ?
det(M)
// Second membre
b = [3 ; 1 ; -2];
// Solution
M\b
```

7.12 Exercice 26

```
// Exercice 26
A = rand(3,3);
// Q matrice de passage, D diagonale
[Q, D] = spec(A);
disp(spec(A)) // affiche juste les valeurs propres
Y = inv(Q) * A * Q // devrait être diagonale mais non
```

```
clean(Y) // efface les valeurs négligeables
```

7.13 Exercice 27

```
// Exercice 27
// Question 1
X = linspace(-%pi, %pi, 500);
Y1 = cos(X);
Y2 = sin(X);
scf(1); clf(1);
plot(X, Y1); plot(X, Y2, 'r');

// Question 2
X = linspace(-10, 10, 500)
Y = X .^ 3 - 3 .* X .^2
scf(1); clf(1);
plot(X, Y);
// Changement d'échelle
a = gca(); a.data_bounds = [-10 -100 ; 10 100];
```

7.14 Exercice 28

```
// Exercice 28
// Ouverture et nettoyage
scf(1); clf(1);
// On trace un segment en faisant un simple
// plot avec deux points
plot([0 , -sqrt(3) / 2], [1, -1/2])
plot([0 , sqrt(3) / 2], [1, -1/2])
plot([-sqrt(3) / 2 , sqrt(3)/2], [-1/2 -1/2])
// Pour le cercle on trace en paramétré
Theta = linspace(-%pi, %pi, 500);
plot(cos(Theta), sin(Theta));
```

7.15 Exercice 29

```
// Exercice 29
nb_points = 500;
T = linspace(0,10,nb_points);
plot(T - sin(T), 1 - cos(T));
```

7.16 Exercice 30

```
// Exercice 30
Theta = linspace(0, 2*%pi, 500);
rho = 1 + cos(Theta)
scf(3); clf(3);
plot(rho .* cos(Theta) , rho .* sin(Theta))
```

7.17 Exercice 31

```
// Exercice 31
function D = de_6_faces(n)
    D = floor(6 * rand(1,n)) + 1;
endfunction

scf(1); clf();
for k = 1:4
    subplot(2,2,k);
    histplot(6,de_6_faces(10^(k+1)));
    legende = sprintf("10^%d valeurs", k+1);
    xtitle(legende);
end
```

Le théorème illustré à la loi des grands nombres appliquée aux différentes fonctions indicatrices $\mathbf{1}_{\{D=i\}}$, où $i = 1, \ldots, 6$ et D est une variable aléatoire uniforme sur l'ensemble fini $\{1, 2, \cdots, 6\}$. Un exemple de dessin obtenu est celui de la figure 3.

7.18 Exercice 32

Bien sûr dans la vraie vie on utiliserait la fonction `factor` de Scilab...

```
function p = est_premier(n)
    fin = floor(sqrt(n));
    for k = 2:fin
        if modulo(n,k) == 0
            p = %f;
            return
        end
    end
    p = %t;
endfunction
```

7.19 Exercice 33

```
function T = triangle_pascal(n)
    T = zeros(n,n+1);
    T(1, [1 2]) = [1 1];
    for i = 2:n
        T(i,:) = T(i-1,:) + [0 T(i-1,1:n)];
    end
endfunction
```

Références

[BC07] Bernard Bercu and Djalil Chafaï. *Modélisation stochastique et simulation*. Dunod, 2007.

[Dec10] Jean-Marc Decauwert. Probabilités et statistiques avec scilab. http://chamilo1.grenet.fr/ujf/courses/AGREGATIONDEMATHEMATIQUESMODELISATIO/document/TP/polyscilab.pdf, 2010.

[Tou99] Paul S. Toulouse. *Thèmes de probabilités et statistiques*. Dunod, Paris, 1999.

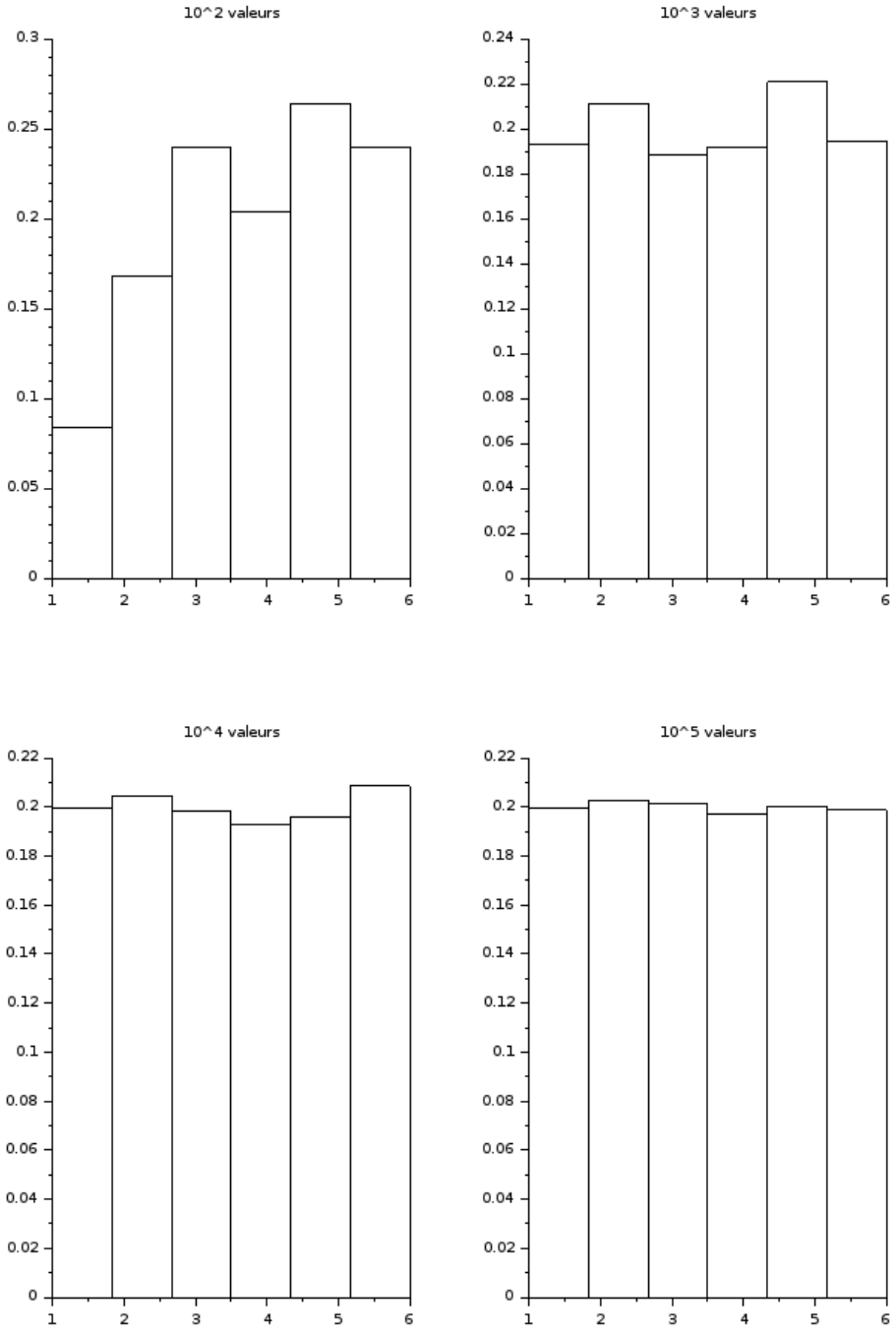

FIGURE 3 – Lancers de dés