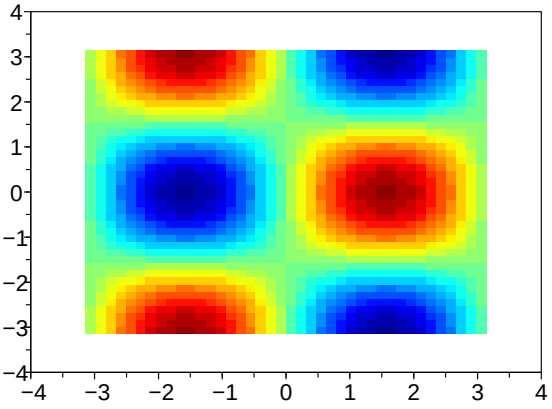
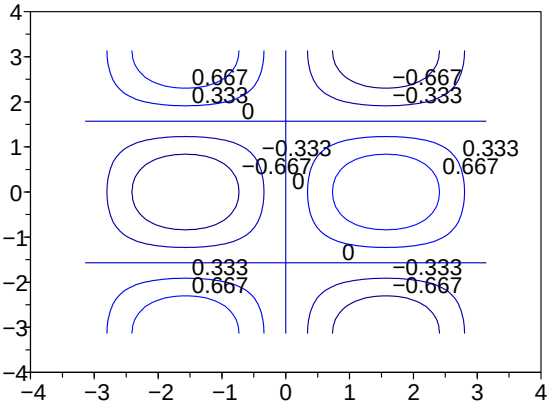
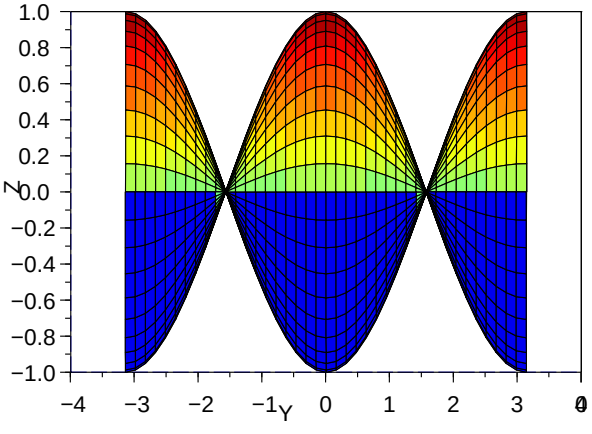
Formation à Scilab







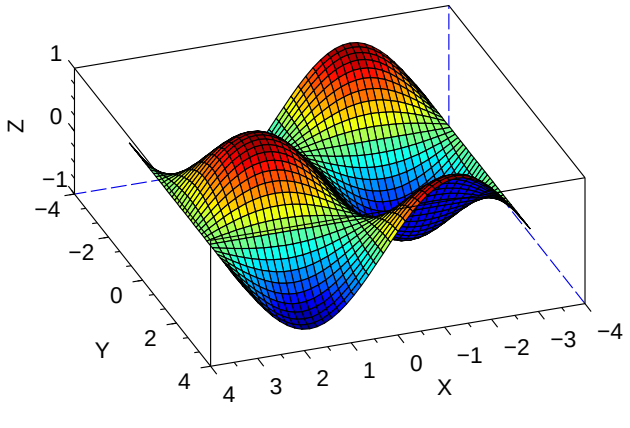


Table des matières

1 Introduction à Scilab

1.1 Téléchargement du logiciel

Scilab (Scientific Laboratory) est un logiciel de calcul numérique développé par l'Institut National de Recherche en Informatique et en Automatique (**INRIA**) et disponible gratuitement sur le site :

http://www.scilab.org

Pour le télécharger, il suffit d'aller sur le site ci-dessus et de cliquer ensuite sur :
"Download/Scilab/products…", puis de choisir le téléchargement souhaité :

Windows XP, Windows Vista, Windows 7, Windows 8, Linux ou bien Mac OS X

suivant l'ordinateur et le système d'exploitation. Dans le cas de windows, il faut ensuite choisir 32 bits (ordinateur un peu ancien) ou 64 bits (ordinateur récent).

1.2 Historique et différentes versions

Le projet initial est né en 1982, à partir de Mathlab qui était encore dans le domaine public. Dans les années 1990, des chercheurs de l'INRIA et de l'ENPC mettent au point la première version de de ψlab (Psilab, même prononciation), qui sort en 1994 (version 1.1). Le Groupe Scilab développe ensuite le logiciel jusqu'à la version 2.7 en 2002.

Début 2003, l'Inria décide de créer le Consortium Scilab pour faire face à l'augmentation du nombre de personnes utilisant Scilab et pour assurer son avenir et son développement.

En 2008, la version 4.1.2 est développée et en 2009 la version 4.4 avec le nouveau mode graphique (à partir des versions 4.x).

Scilab Entreprises est ensuite crée en 2010 sur le modèle économique Open Source et depuis 2012, le logiciel Scilab est développé par Scilab Entreprises.
La version actuelle est la version 5.4. Certaines fonctions graphiques présentes dans les anciennes versions (comme xset, xget, xdel ou xbasc) ont disparu à partir de la version 5.

Lors de la consultation de documentations sur Scilab, il est nécessaire de repérer à quelle version elles font référence car certaines commandes ont disparu des versions récentes et certaines pratiques ont été simplifiées ou rendues plus efficaces par l'introduction de nouvelles commandes.

1.3 Console - Editeur Scinotes - Aide

A l'ouverture de Scilab, une **fenêtre de commandes** ou **console** s'ouvre et une invite de commande `-->` indique que le logiciel attend les instructions.

Cette première fenêtre **fonctionne comme une calculatrice**. Elle est étudiée dans la partie 2 et permet de manipuler les commandes de base de Scilab.

Dans le menu édition, "**Effacer la console**", permet d'avoir une fenêtre de commandes vierge, on peut aussi taper la commande **clc (CLear Console)** qui remet l'écran de la console à l'état initial, sans supprimer les variables en cours dans la session.

Dès que l'on a besoin d'écrire des **programmes un peu développés**, on utilisera la **fenêtre d'édition Scinotes**, qui permet d'**écrire**, de **sauvegarder**, de **modifier** des programmes complets, avec des **commentaires** détaillés. Même dans le cas d'utilisation de quelques commandes simples, le fait d'utiliser l'éditeur Scinotes permet de garder une trace des manipulations successives effectuées.

Par ailleurs, dès que l'on veut créer une fonction, tracer des graphes, il est pratique d'utiliser la fenêtre Scinotes.

L'utilisation de la fenêtre Scinotes sera étudiée en détail dans les sections 3, 4 et 5.

Pour ouvrir la fenêtre Scinotes, on clique sur l'icône située en haut à gauche de l'écran "**démarrer Scinotes**" qui ouvre une seconde fenêtre dans laquelle on pourra écrire le programme considéré.

Lors de l'écriture d'un programme il est très utile d'utiliser la **fenêtre d'aide** de Scilab, qui peut être ouverte en cliquant sur le **point d'interrogation**. Dans cette fenêtre, en cliquant sur l'**icône de recherche** (loupe à droite), on a accès à la **syntaxe de toutes les commandes**. Grâce aux opérations de **copier** (**ctrl C**) et **coller** (**ctrl V**) on peut alors recopier directement des morceaux de programme ré-utilisables dans la fenêtre Scinotes.

Pour passer d'une fenêtre à l'autre, il suffit de cliquer sur l'un des icônes en bas de l'écran.

A la première ouverture de Scilab, trois autres fenêtres apparaissent : le navigateur de variables, le navigateur de fichiers et le navigateur de commandes. Ces trois fenêtres seront peu utilisées dans un premier temps.

2 Usage de la console

Dans tout ce chapitre on pourra se référer au **dictionnaire Pascal - Scilab** pour avoir de manière condensée la syntaxe des commandes.

2.1 Premiers exemples - Calculs numériques

Dans la fenêtre de commande, tapez :

– – > x=1

les lignes suivantes s'affichent :

```
x =
   1.
– – >
```

Scilab a créé automatiquement une variable notée x, à laquelle la valeur 1 a été affectée, et le résultat de l'affectation s'affiche. Il n'est pas nécessaire dans Scilab de déclarer les variables contrairement au cas des langages compilés comme Pascal.

Si vous tapez maintenant :

– – > y=2 ;

une variable y a bien été créée, on lui a affecté la valeur 2, mais le résultat ne s'affiche pas, à cause de la présence du **;** (ce qui peut être utile si on simule par exemple 10 000 échantillons des valeurs prises par une variable aléatoire suivant une loi normale, ou une matrice 100×100).

Si l'on veut maintenant, afficher quand même la valeur de y, on utilise la commande **disp** :

– – > disp(y)

Les lignes suivantes s'affichent :

2.
– – >

Si l'on désire à présent faire afficher la **chaîne de caractères** "bonjour", on tape :

– – > disp('Bonjour')

Les lignes suivantes s'affichent :

Bonjour
– – >

Remarque : En Scilab les chaînes de caractères peuvent être écrites indifféremment 'Bonjour' ou "Bonjour".

Si vous tapez maintenant :

– – > x=x+3

les lignes suivantes s'affichent :

x =
4.
– – >

Le logiciel a rajouté 3 à la valeur de x, puis a affecté le résultat à la variable x (l'ancienne valeur de x a été "écrasée" par la nouvelle).

Dès qu'une variable a été créée à un moment donné, Scilab la conserve avec sa valeur jusqu'à ce que l'on sorte du logiciel.

Les fonctions usuelles peuvent être utilisées :

abs (valeur absolue), sqrt (racine carrée) , log (logarithme népérien),
exp , cos , sin , floor (partie entière),

ainsi que toutes les opérations usuelles :

$+$, $-$, $*$ (multiplication), / (division), ^ (puissance).

Si vous tapez :

– – > 2*3

les lignes suivantes s'affichent :

ans =
6.
– – >

Une variable appelée ans (pour answer) garde en mémoire le résultat du dernier calcul non affecté.

Si l'on veut **réutiliser une commande** déjà tapée sans avoir à la réécrire, on peut utiliser les **flèches** : ↑ ou ↓ .

Les **constantes** e et π sont prédéfinies. On peut les utiliser en tapant **%e** et **%pi** .

Si l'instruction est trop longue pour être tapée sur une seule ligne, on peut la continuer sur la ligne suivante en tapant **deux points** :

```
– – > y=cos(1 + log(x)^2) ..
– – > /(3 + x^4)
y =
   0.0000308
– – >
```

Remarque : Les réels sont codés en **64 bits** dans Scilab : un bit pour le signe, 11 bits pour l'exposant et 52 bits pour la mantisse, ce qui donne des réels compris entre 2.10^{-308} et 2.10^{308} avec une précision de l'ordre de 2.10^{-16}. Cette valeur est celle de la variable prédéfinie $\%eps$:

```
– – > %eps
%eps =
   2.220D-16
– – >
```

Travaux pratiques 1 : Prise en main

Tapez successivement les commandes suivantes et vérifiez ce qui se passe, en comprenant les messages d'erreur éventuels :

$x = 5$		$x = x + 7$
$x = 7;$		disp('x')
$8x$		$y =$ sqrt(x^3)
$8 * x$		disp(y, 'valeur de y');
pi	puis	%e
%pi		$z = 3;$
$3 * 5$		disp(Z);
$x + 12;$		%pi
disp(x)		format(25); %pi

2.2 Calculs matriciels

Dans tout ce paragraphe on pourra se référer au Dictionnaire Pascal-Scilab et à la feuille sur le calcul matriciel.

Pour les matrices de petite dimension, la manière la plus simple est de les introduire en écrivant leurs coefficients entre crochets, les séparateurs des lignes étant les points virgule, les séparateurs entre les éléments de chaque ligne étant soit des virgules, soit des espaces :

– – > A=[1,2,3 ;4,5,6 ;7,8,9]

les lignes suivantes s'affichent :

```
A =
   1. 2. 3.
   4. 5. 6.
   7. 8. 9.
- - >
```

On peut ensuite effectuer les opérations usuelles entre les matrices :

$*$(produit matriciel), $+$ (addition de matrices), $-$ (soustraction de matrices).

Le **produit** d'un **réel** par une **matrice** s'effectue simplement :

```
- - > B=2*A
B =
    2. 4.  6.
    8. 10. 12.
14. 16. 18.
- - >
```

De même que la **puissance** :

```
- - > C=A^2
C =
    30. 36. 42.
   66. 81. 96.
   102. 126. 150.
- - >
```

Plus généralement, si f est une **fonction** (comme sin, cos, exp, floor, abs, log), la fonction f appliquée à la matrice A dont les coefficients sont notés $a_{i,j}$, donne une matrice C dont les coefficients sont égaux aux $f(a_{i,j})$:

```
- - > C=cos(A)
C =
    0.54    - 0.42   - 0.99
  - 0.65     0.28     0.96
    0.75    - 0.15   - 0.91
- - > D=2*A+A^2
D =
   32.   40.   48.
   74.   91.   108.
   116.  142.  168.
- - >
```

Scilab est un langage essentiellement fondé sur les structures matricielles, ainsi un réel est en fait pour Scilab une matrice 1×1. Une matrice peut avoir des coefficients réels, complexes, chaînes de caractères etc.

En plus des opérations matricielles usuelles, Scilab permet d'effectuer des **opérations coefficient par coefficient ou opérations pointées :**

.$*$ (produit coefficient par coefficient), ./ (division coefficient par coefficient),
.^ (puissance coefficient par coefficient).

Par exemple :

```
- - > E=A .* A
E =
   1.   4.  9.
   16. 25. 36.
   49. 64. 81.
```

Pour introduire une matrice nulle possédant n lignes, p colonnes, on utilise la commande **zeros(n,p)** :

```
- - > F=zeros(2,3)
F =
    0. 0. 0.
    0. 0. 0.
- - >
```

Pour introduire une matrice possédant n lignes, p

colonnes, ne contenant que des 1, on utilise la commande **ones(n,p)** :

```
– – >G= ones(3,2)
G =
    1. 1.
    1. 1.
    1. 1.
– – >
```

La commande **eye(n,n)** permet de créer la matrice I_n (matrice diagonale ne comportant que des 1 sur la diagonale). Voir aussi la commande **eye(n,p)** qui peut s'avérer utile.

Pour obtenir la **transposée** d'une matrice A on utilise la commande **A'** (**attention**, dans le cas d'une matrice à **coefficients complexes**, **A'** donne la **transconjuguée** de A).

Pour **concaténer** deux matrices ayant le même nombre de lignes, on procède de la manière suivante :

```
– – >H=[F,G']
H =
    0. 0. 0. 1. 1. 1.
    0. 0. 0. 1. 1. 1.
– – >
```

Remarque : Ces commandes permettent d'initialiser les matrices avant de pouvoir faire des calculs de manière rapide et sans utiliser de boucles(cf. travaux dirigés 2). Il vaut mieux autant que possible, utiliser les commandes matricielles pour les initialisations, car cela rend la programmation plus compacte, plus efficace et cela rend l'exécution plus rapide.

Travaux pratiques 2 : Triangle de Pascal - initialisation

Grâce aux commandes matricielles ci-dessus, créer une matrice 10×10 de la forme $\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$. Cette matrice sera ensuite complétée dans les travaux pratiques grâce à une boucle for, de manière à obtenir le triangle de Pascal.

Pour avoir accès au nombre de lignes et au nombre de colonnes d'une matrice, on utilise la commande **size** :

```
– – >H = size(G) ;
  H =
      3. 2.
– – >
```

Pour accéder au coefficient de la matrice A situé à la $i^{ème}$ ligne et à la $j^{ème}$ colonne, on écrit :

```
– – >a = A(2,3)
  a =
      6
– – >
```

Pour introduire des vecteurs lignes dont les coefficients sont régulièrement espacés (ce qui sera indispensable lors des tracés de graphes), comme `[1 3 5 7 9 11]` on dispose de deux commandes :

L =1 :2 :11
$L =$ `linspace(1,11,6)`

La commande **linspace(1,11,6)** découpe le segment $[1, 11]$ en 5 sous-segments ; le vecteur créé est de longueur 6. De même :

```
-->L1=0 :2 :12
L1 =
    0. 2. 4. 6. 8. 10. 12.
-->L2=linspace(0,12,7)
L2 =
    0. 2. 4. 6. 8. 10. 12.
```

La commande **L=1 :10** est équivalente à la commande L=1 :1 :10
(la longueur du pas lorsqu'elle est égale à 1 peut être omise).

La commande **L=A(1, :)** permet d'extraire la première ligne de A,
(le symbole **:** représentant le vecteur de tous les éléments d'une ligne ou d'une colonne) et **C=A(:,2)** permet d'extraire la deuxième colonne de A :

```
-->L = A(1, :)
  L =
     1. 2. 3.
-->C = A( :,2)
  C =
     2.
     5.
     8
-->
```

On peut aussi extraire des matrices, supprimer une ligne ou une colonne etc ... on pourra se reporter à l'aide de Scilab ou à la feuille sur le calcul matriciel pour obtenir la syntaxe exacte.

Les commandes pour obtenir la transposée, l'inverse (si elle existe), le rang, la longueur d'une matrice ligne ou d'une matrice colonne, figurent dans le tableau du dictionnaire Pascal-Scilab. Elles seront aussi testées dans les travaux pratiques ci-dessous.

Travaux pratiques 3 : Calculs matriciels

Tapez successivement les commandes suivantes et vérifiez ce qui se passe, en comprenant les messages d'erreur éventuels :

$x = [10, 11, 12]$		$y1 = A * y$		$t = 0 : 0.5 : 2$
$y = [10, 11, 12]'$		$x1 = x + 2$		$n = \text{length}(t)$
$A = [1, 2; 3, 4; 5, 6]$		$x2 = x * 2$		$[n, p] = \text{size}(A)$
$A = [A, y]$		$A(1, 2)$		$z = \exp(x)$
$I = \text{eye}(3, 3)$	puis	$C1 = A(:, 1);$	puis	$\text{abs}([-4 : 1 : 4])$
$C = A * I$		$x\hat{}2$		$\text{abs}([-4 : 4])$
$C = A. * I$		$x.\hat{}2$		$u = \text{linspace}(0, 1, 5)$
$D = \text{diag}(A)$		$M = \text{ones}(3, 4)$		$u = \text{linspace}(0, 2, 5)$
$y0 = A * x$		$N = \text{zeros}(2, 3)$		$x * x$

2.3 Simulation de variables aléatoires

Scilab dispose de deux commandes pour générer des nombres pseudo-aléatoires : **`rand`** et **`grand`**. L'usage de rand est assez limité, celui de grand permet de nombreux types de simulations (lois usuelles, chaînes de Markov etc ...).

1. Commande rand.

   La commande **A=rand(n,p)** génère une matrice A possédant n lignes et p colonnes, dont les coefficients pseudo-aléatoires sont les valeurs prises par des variables aléatoires indépendantes et uniformément distribuées suivant toutes **la loi uniforme sur** $[0,1]$ (comme dans le cas de l'instruction random du Pascal).

   De même, la commande **A=rand(n,p,'n')** génère une matrice possédant n lignes et p colonnes, dont les coefficients pseudo-aléatoires sont les valeurs prises par des variables aléatoires indépendantes et uniformément distribuées suivant toutes **la loi normale centrée réduite**.

2. Commande grand.

   La commande **grand** permet de simuler toutes les lois usuelles :

Loi binomiale de paramètre N et p	y1 = grand(m, n, "bin", N, p) ;
Loi de Poisson de paramètre λ	y2 = grand(m, n, "poi", lambda) ;
Loi normale de paramètre μ et σ^2**(variance)**	y3 = grand(m, n, "nor", mu, sigma) ;
Loi uniforme à densité sur $[a,b]$	y4 = grand(m, n, "unf", a, b) ;
Loi uniforme discrète sur $[\![1,n]\!]$	y5 = grand(m, n, "uin", 1, n) ;
Loi exponentielle **d'espérance** $\frac{1}{\lambda}$	y6 = grand(m, n, "exp", 1/lambda) ;
Loi géométrique de paramètre p	y7 = grand(m, n, "geom", p) ;
Loi Gamma de paramètres b et υ	y8 = grand(m, n, "gam", b, nu) ;

   On remarquera que dans le cas de la **loi normale**, c'est l'**écart-type** qui est fourni comme paramètre (et pas la variance) et que dans le cas de la **loi exponentielle**, c'est **l'espérance** qui est fournie (et non pas le paramètre λ).

3 Programmation - Généralités.

3.1 Introduction.

L'**interpréteur de commande** de Scilab permet d'**exécuter une commande** (sans ou avec écho [1] selon la présence ou non du **;**) dès que l'on appuie sur Entrée. Ceci est relativement pratique pour utiliser Scilab en mode "**calculatrice**" et faire quelques essais concernant la syntaxe du langage Scilab, pour une première prise en main mais, dès lors que l'on doit **modifier un paramètre** et **relancer un certain nombre de calculs**, il s'avère nécessaire de **sauver toutes les commandes dans un fichier** qui sera par la suite interprété dans son intégralité par Scilab : on parle alors de "**script**" en Scilab (ou de **programme**).

1. affichage du résultat de la commande

La fenêtre principale de Scilab dispose d'un menu "**Applications**". Lorsqu'on ouvre ce menu, on peut lancer l'éditeur de texte **SciNotes** (ou SciPad dans les versions antérieures).

On peut aussi lancer Scinotes grâce au bouton "**démarrer Scinotes**" en haut à gauche de l'écran . On tape alors les commandes et les fonctions que l'on veut dans la fenêtre Scinotes en enregistrant régulièrement grâce au menu "**fichier**", puis "**enregistrer sous**" pour le premier enregistrement, puis "**enregistrer**" pour les suivants (le fichier doit être enregistré avec l'extension **.sce**).

Avant de travailler sur un programme donné, il est très utile de **créer un répertoire** dans lequel on enregistrera ce fichier ainsi que tous les fichiers reliés.

Par exemple, dans le **répertoire "Scilab"**, on créera un sous-répertoire "**analyse**", puis à l'intérieur, un sous-répertoire "**series**". On veillera ensuite à **donner au fichier un nom explicite**, comme serie_Riemann.

Le fait de bien gérer l'arborescence des fichiers et de donner des noms précis permet ensuite de **réutiliser rapidement des fichiers contenant des programmes déjà testés** (cela permet de gagner beaucoup de temps et de retrouver très rapidement les syntaxes des commandes et des boucles après une période sans utilisation du logiciel).

Une fois que les commandes et les fonctions ont été écrites dans le fichier **.sce**, pour exécuter le programme, on peut utiliser le menu "**exécuter**", puis "**enregistrer et exécuter**" de la fenêtre Scinotes.

Par précaution, Scilab ré-enregistre le fichier juste avant l'exécution, ce qui permet en cas d'erreur d'exécution (runtime error), de ne pas perdre le contenu du fichier sur lequel on travaille. Les résultats de l'exécution du programme apparaissent alors dans la **console** (fenêtre de commandes) à laquelle on a accès en cliquant sur l'icône correspondante en bas d'écran.

Dès qu'un fichier contenant des déclarations de fonctions ou des variables a été **exécuté dans Scilab**, il a été **chargé en mémoire**, et les **fonctions correspondantes peuvent être utilisées** dans d'autres programmes (ou dans la fenêtre Scinotes). La gestion des fichiers ne pose en général pas de problème particulier (on travaille en général dans un seul répertoire courant).

Scilab travaille à partir d'un répertoire de base, qui est donné par la commande **pwd (**print working directory). C'est là qu'il va chercher par défaut les fichiers à charger ou à exécuter.

On peut le changer par la commande **chdir**.
(par exemple : chdir(' \Mes documents\Scilab') en relatif par rapport au répertoire courant).

A défaut, il faut saisir le chemin d'accès complet du fichier que l'on souhaite charger ou sauvegarder. Le plus facile est d'utiliser le **menu de l'interface**. Dans l'interface (console) de Scilab, les seules commandes qui apparaîtront seront les exécutions ou les chargements répétés de fichiers externes. Il est conseillé de **toujours maintenir ouvertes les deux fenêtres** : la fenêtre Scilab, et la fenêtre d'édition SciNotes.

Scilab distingue **deux** sortes de fichiers :

– ***Les fichiers de sauvegarde d'une session***.

  Ce sont des fichiers binaires, créés (en console) par la commande **`save`** et rappelés par **`load`** ou grâce au menu "**fichier**" puis "**ouvrir un fichier**" dans la console. Ceci permet de reprendre un calcul en conservant les valeurs déjà affectées. On peut aussi sauver des variables dans un fichier texte par **`write`** et les rappeler par **`read`**.

– ***Les fichiers de commandes et de fonctions***.

  Ce sont des fichiers texte.

Ils contiennent tout d'abord des suites d'instructions Scilab, qui sont exécutées successivement par la commande **exec** : dans la console, on tape **exec**(”repertoire_du_script/script.sce”) ou encore par le menu ” **Fichier**” , on choisit **exécuter** puis le fichier "script.sce" ou encore grâce au choix du menu ”**Exécuter**” de la **fenêtre** Scinotes, on lance cette commande, si le fichier "script.sce" y est ouvert. Attention, la dernière ligne du fichier doit obligatoirement se terminer par un retour-chariot pour être prise en compte.

Ils peuvent aussi contenir des **déclarations de fonctions** (suites d'instructions définissant un sous programme limité à une action simple), ce qui donne une **modularité** au programme (cela correspond aux procédures et aux fonctions du Pascal).

Un fichier de commandes peut réaliser les mêmes tâches qu'une fonction et réciproquement. Pour une utilisation courante ou de mise au point, les fichiers de commandes permettent de suivre le contenu de toutes les variables.

Pour une programmation plus avancée, il est préférable de définir des fonctions, car leurs variables internes restent locales. Un même fichier **.sce** peut contenir plusieurs fonctions.

Commentaires : Afin de rendre un script plus compréhensible, il est judicieux de placer des commentaires, qui apparaissent à droite des caractères **”//”** :

// Ceci est commentaire qui ne sera ni affiché ni interprété par Scilab

Le fait de rajouter les caractères ”//” en début de ligne permet de la transformer en commentaires et d'éviter de l'exécuter, ce peut être pratique si l'on veut exécuter juste une partie de programme, ou ne pas exécuter une ou deux commandes (qui servent dans un premier temps à tester le programme, cf. paragraphe ci-dessous : conseils de programmation).

Aide intégrée : Il suffit de taper **help ma_commande** pour accéder aux détails de celle-ci. Sans l'argument **ma_commande**, on obtient l'accès à l'aide sur l'ensemble des commandes classées par rubriques. On peut aussi utiliser le **point d'interrogation** en haut à droite de l'écran dans les fenêtres Scinotes ou dans la console.

3.2 Conseils de programmation.

Pour programmer vite et efficacement, et pour que le programme écrit soit ré-utilisable facilement, il est conseillé de respecter les règles suivantes (qui ne sont d'ailleurs pas spécifiques de Scilab) :

Organisation des fichiers dans l'ordinateur :

1. **Avant de commencer à travailler**, créer une **structure arborescente de fichiers claire et cohérente** (cette structure pourra être modifiée ensuite en fonction des nouveaux fichiers à intégrer), lorsque l'on dispose de plusieurs centaines de fichiers, il vaut mieux qu'ils soient bien ”rangés” si on veut les retrouver facilement plus tard.

2. Prendre un **fichier déjà existant** avec des commandes proches de celles que l'on veut utiliser, utiliser ”**enregistrer sous**” et créer ainsi un nouveau fichier, enregistré au bon emplacement, avec des parties de programme déjà écrites, **avant même de commencer à programmer** (rien n'est plus rageant que d'écrire des commandes pendant 20 minutes, de faire une fausse manipulation et de tout perdre !).

Une autre solution est de créer un nouveau fichier, de le sauvegarder au bon emplacement, puis d'utiliser des opérations de copier (ctrl C) - coller (ctrl V) pour récupérer des morceaux de programmes déjà écrits.

3. Choisir des **noms de fichiers explicites** : si les fichiers s'appellent fichier1, fichier2 ... ou activité1, activité2 ... ils seront difficiles à retrouver (recherche par mot clé), alors que **fn_densite_loi_norm.sce** sera facile à retrouver (**éviter les accents et les espaces** dans les noms de fichier car cela pose des problèmes à certains logiciels).

4. **Sauvegarder** très régulièrement (toutes les 2 min par exemple, ensuite cela devient un réflexe) le fichier sur lequel on travaille, et créer systématiquement des **copies** dans un répertoire de sauvegarde, sur disque dur externe ou sur une **clé usb**.

Structuration interne d'un programme :

1. Choisir des **noms de variables et de fonctions explicites**, si les variables s'appellent x,y,z,t,u,v,w, le lecteur doit faire un effort pour comprendre ce qu'elles représentent, alors que si un entier est noté n, une somme partielle de série est notée somme_partielle, ou s_partielle, ou somme, c'est beaucoup plus clair et cela ne prend pas plus de temps à écrire grâce aux **raccourcis clavier ctrl C** (copier) et **ctrl V** (coller)

2. Commencer par écrire quelques lignes de programme, les sauvegarder, puis les tester (en rajoutant éventuellement des commandes disp() pour vérifier l'évolution des variables). Une fois que le **bloc de petite taille** fonctionne correctement, compléter peu à peu le programme (le fait d'écrire en une seule fois un programme long complique ensuite le "debuggage" car il est difficile de repérer où se situent les erreurs),

3. Ecrire des **commentaires** dès que c'est nécesssaire, en particulier lors des passages un peu délicats, ou pour présenter une fonction, ou une suite de commandes qui forment un bloc,

4. Lors de l'usage d'une nouvelle commande, **utiliser l'aide de Scilab** et les opérations de copier (ctrl C) - coller (ctrl V) pour obtenir directement des lignes de commandes avec une syntaxe correcte,

5. Toujours privilégier les **structures modulaires ou par blocs** (utilisation de fonction) plutôt que d'avoir un programme principal très long et dont les blocs n'apparaissent pas clairement (ils ne sont donc pas réutilisables directement dans un autre contexte),

6. Dans les instructions conditionnelles ou les boucles, utiliser l'**indentation** via les tabulations (dans Scinotes cela est fait automatiquement).

4 Programmation - Variables et structures de contrôle.

Pour l'ensemble de ce paragraphe, on pourra se reporter utilement au dictionnaire Pascal-Scilab.

4.1 Variables.

Elles sont créées dès qu'elles sont introduites avec affectation (`=`) d'un contenu au cours de la session. Leur type est déterminé selon leur contenu. Les types des variables ne doivent pas être déclarés et ils peuvent changer à tout moment par une nouvelle affectation.

`who` ou `whos` permet de connaître (à partir de la console) les variables (globales) actuellement utilisées.

Dans les récentes versions les variables en cours sont visibles dans la fenêtre d'inspection des variables.

clear permet d'effacer toutes les variables, **clear x** permet d'effacer la variable **x.**

4.2 Entrées/sorties des données/résultats.

Lorsqu'un **programme** est **exécuté**, les **valeurs des variables ne sont pas affichées** (même dans le cas où figure par exemple la commande **x=3** sans point-virgule dans le programme), sauf si on choisit dans le menu **"exécuter"**, l'option **...fichier avec écho**. Pour pouvoir faire afficher la valeur de la variable, il faut nécessairement utiliser la commande **disp.**

1. Sortie écran : affichage d'un message, d'une variable.

   La commande **disp**(b, "bonjour") affiche le message "bonjour" **puis** le contenu de la variable b. (équivalent de la procédure "write()" de Pascal). Attention : les arguments doivent être introduits dans l'ordre inverse par rapport à l'affichage.

2. Entrée clavier

   x=**input**("Donnez une valeur pour x : ") (équivalent des instructions "writeln('Donner une valeur pour x : ') ; read(x) ;" de Pascal).

3. Entrée fichier

   On peut utiliser la commande y=**read**("fichier.txt") pour créer une variable y dont le contenu est dans le fichier.

4. Sortie fichier

   On peut utiliser la commande **write**("fichier.txt", A) pour écrire le contenu de A dans un fichier texte.

4.3 Structures de contrôle.

4.3.1 Structures conditionnelles

Voici la syntaxe de la structure conditonnelle de base qui permet de donner des instructions différentes selon qu'un test est vrai ou faux :

if . . .then . . .else . . .end

Plus généralement avec des alternatives multiples :

if condition1 **then**
    suite d'instructions 1
elseif condition 2 **then**
    suite d'instructions 2
. . . . . .
elseif condition N **then**
    suite d'instructions N
else
    suite d'instructions N+1
end

4.3.2 Boucle for.

L'instruction `for` permet de faire une itération sur les composantes d'un vecteur ligne (valeurs du compteur) donné.

Dans la commande ci-contre, l'instruction sera exécutée pour x prenant successivement comme valeurs les composantes de v.

```
for x = v ;
suite d' instructions portant ou non sur x ;
end
```

La boucle `for` peut aussi s'appliquer à une matrice A quelconque. Dans la commande ci-contre, l'instruction sera exécutée pour x prenant successivement comme valeurs les colonnes de la matrice A.

```
for x = A ;
instructions portant ou non sur x ;
end
```

Exemple :

```
for i=1 :5 ;
    disp(i) ; // Affiche i allant de 1 à 5 (par pas de 1)
end
```

Travaux pratiques 4 : Suite définie par récurrence

Créer un nouveau répertoire "suite", puis créer et enregistrer un nouveau fichier "suite_recurrente.sce" dans ce répertoire.
Ecrire un programme qui calcule le n^{ème} terme de la suite (u_n) définie par :

$$\begin{cases} u_1 = \frac{1}{2} \\ u_{n+1} = a u_n (1 - u_n) \quad \text{si } n \geq 1 \end{cases}$$

a étant un nombre réel dont on fera varier les valeurs.
On pourra visualiser le comportement de la suite (u_n) au moment de l'étude de la fonction $plot2d$.

Travaux pratiques 5 : Triangle de Pascal (suite).

Créer et enregistrer un nouveau fichier "Triangle_Pascal.sce" dans un sous-répertoire bien choisi. Reprendre les commandes des travaux pratiques 2, de manière à obtenir un programme qui en fonction de la valeur de n, calcule les coefficients binomiaux $\binom{n}{k}$ pour $0 \leq k \leq n$. Tester le programme pour différentes valeurs de n.

4.3.3 Boucle while.

`while` : Cette instruction permet de faire une boucle de type "tant que". La syntaxe est la suivante :

```
while condition,
    ..... // tant que condition est vraie, cette partie est exécutée en boucle
end
```

Remarque : La commande **`break`** permet de quitter une boucle for ou while.
`halt` ou **`pause`** permet de faire une pause dans l'exécution d'un script.

Travaux pratiques 6 : Méthode de dichotomie

On considère la fonction f définie sur $]0,+\infty[$ par $f(x)=\ln x-\frac{1}{x^2}$.
Montrer que l'équation $f(x)=0$ admet une unique solution c sur $]0,+\infty[$ et que $c\in]1,2[$.
Ecrire un programme Scilab qui calcule grâce à une boucle While une valeur approchée de c à 10^{-2} près, puis à 10^{-6} près (on pourra utiliser l'aide de Scilab pour obtenir rapidement la syntaxe correcte de la boucle while).

4.3.4 Résumé :

Récapitulatif des commandes principales				
Pour	for	x = vecteur,	instruction ;	end
Tant que	while	booleen,	instruction ;	end
Si	if	booleen then,	instruction ;	end
Sinon	else		instruction ;	end
Sinon si	elseif	booleen then,	instruction ;	end

4.4 Modularité – Définition de fonctions.

4.4.1 Syntaxe de déclaration d'une fonction renvoyant une seule valeur.

La déclaration d'une fonction renvoyant une seule valeur se présente de la manière suivante :

```
function y = nom_de_la_fonction(x1,x2,...,xm) ;
    ...//corps de la fonction
endfunction ;
```

Toutes les variables utilisées à l'intérieur de la fonction sont par défaut locales. Par exemple, si une variable est notée z à l'extérieur de la fonction et que la lettre z est utilisée pour désigner une variable à l'intérieur de la fonction, cela ne modifiera pas la valeur de la variable z à l'extérieur.

Dans la pratique, il vaut toujours mieux (pour la clarté de la programmation), choisir des noms différents pour les variables à l'intérieur de la fonction et à l'extérieur.

On peut forcer une variable locale dans une fonction à être globale (c'est-à-dire que la fonction modifie la valeur de la variable) mais ce cas se présente très rarement en Scilab.

Voici un exemple de fonction :

```
function d = densite_loi_normale_cr(x) ;        // Densité d'une loi
    d = ( 1 / sqrt(2*%pi)) * exp(- x^2 / 2) ; // normale centrée réduite
endfunction
```

Travaux pratiques 7 : Loi normale centrée réduite.

Créer un nouveau répertoire et un fichier "lois_normales".
Dans ce fichier écrire les trois lignes ci-dessus (déclaration de la fonction densite_loi_normale_cr), puis taper les commandes suivantes :

```
x = [-4 :0.1 :4] ;
y = densite_loi_normale_cr(x) ;
plot(x,y) ;
```

Les tracés des graphes des fonctions seront étudiés dans le chapitre suivant.

Travaux pratiques 8 : Méthode des rectangles

(D'après Escp 2010 - Voie Technologique)
Enregistrer le fichier "suite_recurrente.sce" sous le nom "integrale_rectangle.sce" dans un nouveau répertoire "integrales".
On note h_n la fonction définie sur $[0,1]$ par :
$h_n(x) = \frac{x^n}{x^2+3x+2}$. Créer une fonction qui calcule $h_n(x)$ en fonction de x et tester cette fonction. Ecrire ensuite un programme complet qui demande à l'utilisateur la valeur de n puis qui calcule une valeur approchée de $\int_0^1 \frac{x^n}{x^2+3x+2}dx$. On pourra diviser le segment en $N = 10$, $N = 100$, $N = 1000$, puis $N = 10\,000$ sous-segments et comparer les résultats obtenus. On note :
$\forall n \geq 1, u_n = \int_0^1 \frac{x^n}{x^2+3x+2}dx$.
Que peut-on dire de $\lim\limits_{n\to+\infty} u_n$? On interprétera ce résultat grâce à la fonction plot2d dans le paragraphe sur les tracés des courbes représentatives des fonctions.
Que peut-on conjecturer quand à la limite de nu_n lorsque n tend vers $+\infty$? (grâce à Scilab).

4.4.2 Syntaxe générale de déclaration d'une fonction :

La définition générale d'une fonction commence obligatoirement par une ligne qui déclare le nom de la fonction, les variables d'entrée x1,x2,...,xm et le vecteur des variables de sortie [y1,y2,...,yn] (qui peut être vide [] – analogue à une procédure en Pascal) :

```
function [y1, y2, ..., yn] = nom_de_la_fonction(x1, x2, ..., xm)
.................                          //corps de la fonction
endfunction
```

N'oubliez pas de terminer la dernière ligne par un retour-chariot ou endfunction.

4.4.3 Remarques concernant la programmation et les variables :

Certaines **erreurs difficiles à trouver** proviennent de **confusions entre noms de variables ou de fonctions**. Scilab **garde en mémoire tous les noms introduits tant qu'ils n'ont pas été libérés par `clear`**. Il est donc prudent de donner des noms assez explicites aux variables.

Les variables introduites dans la session ou dans les fichiers de commandes sont globales.

Par défaut, toutes les variables introduites à l'intérieur d'une fonction sont locales.

5 Remarques sur la programmation (en seconde lecture).

Le logiciel Scilab étant fondé sur les matrices, il faut le plus souvent possible utiliser les opérations matricielles prédéfinies.

Voici deux exemples illustrant ce fait.

1. Temps d'exécution et utilisation de la commande timer().

   Pour comparer l'efficacité des algorithmes, on dispose de **`timer()`** qui permet de compter le temps CPU écoulé.

- > timer() - > for i = 1 :1000 ; - > for j = 1 :1000 ; - > B(i,j)=rand(1,1,'n') ; - > end ; - > end ; - > timer() ans = 1.7472112	- > timer() - > C=rand(1000,1000,'n') ; - > timer() ans = 0.0780005

   La deuxième version est plus de 22 fois plus rapide, de plus elle utilise mieux les potentialités du langage. Dans le premier cas, si la matrice avait été initialisée, le temps d'exécution aurait été beaucoup plus court (tel que le programme est écrit ici, il faut à chaque étape redéfinir l'espace mémoire alloué).

2. Un autre exemple :

   Voici un exemple illustrant cette logique. Si $v = (v_i)_{i=1\ldots n}$ et $w = (w_j)_{j=1\ldots m}$ sont deux vecteurs, on souhaite définir la matrice $A = (a_{i,j})$, où $a_{i,j} = v(i)^{w(j)}$. Il y a plusieurs solutions. Dans les commandes qui suivent, v est un vecteur colonne et w est un vecteur ligne.

```
for i=1 :n                          // La matrice A n'a pas été initialisée
    for j=1 :m                      // elle est donc créée de proche en proche, ce qui fait
        A(i,j) = v(i)^w(j) ;        //  perdre du temps au programme lors de l'éxecution.
    end ;                           // De plus, les deux boucles imbriquées rendent
end                                 // le programme compliqué
```

```
A=v^w(1) ;
for j=2 :m                          // Plus efficace
    A=[A,v^w(j)] ;
end

A = (v*ones(w)) .^(ones(v)*w)     // Préférable (utilisation des potentialités matricielles de Scilab)
```

Scilab est un outil de calcul plus que de développement. Pour un problème compliqué, on aura tendance à utiliser Scilab pour réaliser des maquettes de logiciels et tester des algorithmes, quitte à lancer ensuite les gros calculs dans un langage compilé comme C. Cela ne dispense pas de chercher à optimiser la programmation en Scilab, en utilisant au mieux la logique du calcul matriciel.

6 Tracés de graphes.

Dans tout ce paragraphe on pourra utilement se reporter aux documents : **Dictionnaire Pascal - Scilab** et **Tracés de figures avec plot2d**.

Les fonctionnalités graphiques de Scilab sont très variées, il serait trop long de les détailler ici. Pour plus d'informations se reporter l'aide en ligne (rubrique **Graphics**).

6.1 Différentes méthodes.

Pour tracer la courbe représentative d'une fonction f de une variable, Scilab trace une ligne brisée entre les points $(x_1, f(x_1)), (x_2, f(x_2)), \ldots, (x_n, f(x_n))$ .Il faut donc d'une manière ou d'une autre fournir au logiciel le vecteur ligne $x = [x_1, x_2, \ldots, x_n]$ et le vecteur ligne $y = [f(x_1), f(x_2), \ldots, f(x_n)]$ (ou les vecteurs colonnes correspondants dans le cas de l'utilisation de fplot2d)

Pour créer le vecteur ligne x on a deux possibilités. Par exemple pour découper le segment $[0, 1]$ en 100 sous segments, on dispose des solutions équivalentes suivantes :

- x= 0 :0.01 :100 ;
- x=linspace (0,1,101) ;

Pour créer le vecteur y et tracer le graphe, on dispose alors de trois méthodes différentes :

- on génère le vecteur ligne y grâce aux opérations coefficients par coefficients (par exemple, si $f(x) = x^2$, on écrit $y = x.\hat{}2$), puis on utilise les fonctions **plot** ou **plot2d**, cette méthode présente cependant certains inconvénients liés à l'usage des opérations matricielles (cf. la remarque ci-dessous),
- on déclare la fonction f préalablement, puis on génère le vecteur ligne y en utilisant la fonction **feval** (si on reprend l'exemple précédent, on écrira : $y = feval(x, f)$) puis on utilise les fonctions **plot** ou **plot2d** comme ci-dessus,
- on déclare la fonction f préalablement, puis on utilise la fonction **fplot2d**, qui utilise comme argument un **vecteur colonne** au lieu d'un vecteur ligne.

Pour résumer, voici les commandes dans les trois cas de figure :

Tracé avec les opérations coefficient par coefficient	Tracé en utilisant `f`	Tracé en utilisant `fplot2d`
x = [0 :0.01 :1] ; y = x .^2 ; `plot2d`(x, y) ;	`function` y=f(x) y=x^2 ; `endfunction` x = [0 :0.01 :1] ; y = `feval`(x,f) ; `plot2d`(x, y) ;	`function` y=f(x) y=x^2 ; `endfunction` x = [0 :0.01 :1]' ; `fplot2d`(x, f) ;

Exemple de problème qui peut se poser avec la première méthode :

Si l'on veut tracer le graphe de la fonction $f : x \mapsto x + \dfrac{1}{x}$ sur le segment $[1, 2]$, en utilisant les opérations coefficient par coefficient, on aurait tendance à écrire :

```
x = [1 :0.01 :2] ;
y = x + 1./x ;
plot2d( x, y ) ;
```

Mais le logiciel affiche un message d'erreur "Addition incohérente" car Scilab interprète le 1./x comme (1.)/x, c'est-à-dire 1/x (il ne fait donc pas une opération pointée). Il faut donc écrire : (1)./x pour que cette méthode fonctionne correctement.

Les deux autres méthodes, bien qu'un peu plus lourdes dans un premier temps, ne posent pas ce type de problème. De plus, la fonction f ayant été définie une fois pour toutes elle pourra être réutilisée dans d'autres circonstances (meilleure modularité, programme mieux structuré).

Remarques :

1. Pour rajouter un titre (**xtitle**), une légende(**legends**), pour choisir la fenêtre d'affichage (**rect**) , les couleurs (**style**), le type d'échelles et les graduations (**frameflag**), les axes (**axesflag**), la présence d'une grille (**xgrid**) etc .. on se reportera à la feuille "**Tracés de figures avec plot2d**" et à l'aide de Scilab.

2. Le document "Réaliser des graphiques avec Scilab" de Philippe Roux (voir bibliographie [5]) fournit une documentation complète sur le nouveau mode graphique Scilab, la manipulation des **handle** etc.

6.2 Tracés de plusieurs courbes sur un même graphique.

Si l'on ne spécifie rien au logiciel et si on lui demande de tracer plusieurs courbes successivement, il les tracera dans la même fenêtre. Par exemple les commandes :

```
x = [1 :0.01 :4] ;
y1 = (x.^2) ./ (2*x-1) ;
y2 = 1/2* x + 1/4 ;
plot2d( x, y1 ) ;
plot2d( x, y2, rect=[1,0,4,3] ) ;
```

permettent de tracer la courbe représentative de $f : x \mapsto \frac{x^2}{2x-1}$ et de l'asymptote d'équation $y = \frac{1}{2}x + \frac{1}{4}$ sur la même figure. L'argument $rect = [1, 0, 4, 3]$ permet de définir la fenêtre d'affichage (x varie entre 1 et 4, alors que y varie entre 0 et 3).

Travaux pratiques 9 : Méthode des rectangles - suite

(D'après Escp 2010 - Voie Technologique).
Reprendre le fichier précédent portant le nom "integrale_rectangle.sce". On note h_n la fonction définie sur $[0,1]$ par :
$h_n(x) = \frac{x^n}{x^2+3x+2}$. Tracer sur un même graphique les courbes représentatives de h_1, h_3, h_{40}. Interpréter et faire le lien avec les résultats trouvés précédemment.
A-t-on $\lim\limits_{n\to+\infty}\left(\lim\limits_{x\to1^-} h_n(x)\right) = \lim\limits_{x\to1^-}\left(\lim\limits_{n\to+\infty} h_n(x)\right)$?

6.3 Tracés de courbes dans plusieurs sous-fenêtres.

Pour tracer des courbes dans la même fenêtre graphique mais dans plusieurs sous-fenêtres on utilisera la commande **subplot**. La commande **subplot(m,n,p)**, où m,n et p sont trois entiers tels que $p \leq mn$, divise la fenêtre graphique courante en mn sous-fenêtres de même taille (m lignes et n colonnes) et la prochaine figure sera tracée dans la $p^{ème}$ sous-fenêtre (en les numérotant par lignes puis par colonnes) (cf. feuille **Tracés de figures avec `plot2d`**).

6.4 Tracés de courbes dans des fenêtres graphiques différentes.

Pour créer une nouvelle fenêtre graphique, on utilise la commande **xset('window',n)** où n est le numéro de la fenêtre graphique créée. Le prochain tracé sera effectué dans cette nouvelle fenêtre (cf. feuille **Tracés de figures avec `plot2d`**).

6.5 Tracé de courbes en escalier, de diagrammes en bâtons et d'un nuage de points.

Pour tracer une courbe en escalier, on utilise la fonction `plot2d2` (voir l'aide de Scilab pour la syntaxe détaillée).

Pour tracer un diagramme en bâtons, on utilise la fonction **`plot2d3`** (voir l'aide de Scilab pour la syntaxe détaillée).

Pour tracer un nuage de points, on utilise l'argument **style = n**, où n est un entier négatif (cf. la feuille **Tracés de figures avec `plot2d`**).

6.6 Effaçage de toutes les figures.

Pour effacer toutes les figures (ce qui est utile si on exécute plusieurs fois le même programme) on utilise la commande **clf** (CLear Figure).

6.7 Histogrammes.

Scilab dispose de la commande **histplot** pour tracer des histogrammes, avec **deux utilisations possibles** :

– Si x est une matrice dont les coefficients sont les données (par exemple on peut avoir généré x en utilisant la commande : x=`grand`(1,1000,"nor",1,2)), la commande `histplot`(n,x), répartit les données dans n classes de même largeur et trace l'histogramme correspondant, l'effectif de chaque classe étant normalisé par l'effectif total (par exemple, si la première classe comprend 50 données, l'ordonnée sur l'histogramme sera égale à $\frac{50}{1000}$).

– Si b est un vecteur ligne définissante les classes (si $b = [b_1, b_2, \ldots, b_p]$, les classes seront $\left[b_1, b_2\right[$, $\left[b_2, b_3\right[$, $\left[b_3, b_4\right[, \ldots, \left[b_{p-1}, b_p\right[$) et si x est une matrice dont les coefficients sont les données, la commande `histplot(b,x)`, répartit les données dans les classes définies par b et trace l'histogramme correspondant, l'effectif de chaque classe étant normalisé par l'effectif total.

Travaux pratiques 10 : Simulation d'une loi normale et tracé d'histogramme

Reprendre le fichier créé lors des travaux pratiques 7 et le sauvegarder sous un nouveau nom (par exemple "simul_normale_cr"). Compléter ce programme à l'aide de la commande **`grand`** et de la commande **`histplot`** pour qu'il trace dans une même fenêtre l'histogramme correspondant à la simulation de 1000 échantillons d'une variable aléatoire suivant une loi normale centrée réduite et la courbe représentative d'une densité d'une variable aléatoire suivant une loi normale centrée réduite.

Travaux pratiques 11 : Simulations d'une loi géométrique

A l'aide d'une boucle **`while`** et de la commande **`grand( 1, 1, "bin", 1, p);`** (qui permet de simuler une loi de Bernoulli), simuler une variable aléatoire qui suit la loi géométrique de paramètre $p = 0.2$.

Facultatif : Transformer les commandes précédentes pour créer une fonction qui simule une loi géométrique de paramètre p. Grâce à cette fonction obtenir un échantillon i.i.d. de taille 100 de loi géométrique de paramètre p et tracer l'histogramme correspondant dans une première sous-fenêtre en utilisant **`subplot(1,3,1)`**.

Simuler directement une loi géométrique de paramètre $p = 0.2$ à l'aide du générateur **`grand`**. Tracer dans la deuxième sous-fenêtre (grâce à **`subplot(1,3,2)`**), l'histogramme correspondant à cette deuxième simulation.

Dans une troisième sous-fenêtre, tracer la distribution théorique correspondant à une loi géométrique de paramètre 0.2.

Comparer les différents histogrammes obtenus (on pourra faire varier la taille de l'échantillon).

Travaux pratiques 12 : Simulation d'une loi de Poisson

Simuler un n-échantillon une loi de Poisson de paramètre $\lambda = 7$ à l'aide du générateur **`grand`**. Tracer l'histogramme correspondant dans une première sous-fenêtre en utilisant **`subplot(1,2,1)`**.

Dans une deuxième sous-fenêtre (à l'aide de la commande **`subplot(1,2,2)`**), tracer la distribution théorique correspondant à une loi de Poisson de paramètre 7.

Comparer (on fera varier la taille de l'échantillon ainsi que la valeur de λ).

Travaux pratiques 13 : Simulations d'une loi binomiale

A l'aide d'une boucle **`for`** et de la commande **`grand( 1, 1, "bin", 1, p);`** (qui permet de simuler une loi de Bernoulli), simuler une variable aléatoire qui suit la loi binomiale de paramètres $n = 10$ et $p = 0.3$.

Facultatif : Transformer les commandes précédentes pour créer une fonction qui simule une loi binomiale de paramètres n et p. Grâce à cette fonction obtenir un échantillon i.i.d. de taille 100 de loi binomiale de paramètres $n = 10$ et $p = 0.3$ et tracer l'histogramme correspondant dans une première sous-fenêtre en utilisant **`subplot(1,3,1)`**.

Simuler directement une loi binomiale de paramètres $n = 10$ et $p = 0.3$ à l'aide du générateur **`grand`**. Tracer dans la deuxième sous-fenêtre (grâce à **`subplot(1,3,2)`**), l'histogramme correspondant à cette deuxième simulation.

Dans une troisième sous-fenêtre, tracer la distribution théorique correspondant à une loi binomiale de paramètres 10 et 0.3.

Comparer les différents histogrammes obtenus (on pourra faire varier la taille de l'échantillon, ainsi que les valeurs de n et de p).

7 Problème de synthèse

Il s'agit de calculer par deux méthodes la valeur d'une option d'achat sur actions à l'aide du modèle de Cox, Ross, Rubinstein (1979).

Travaux pratiques 14 : Fonction calculant les coefficients binomiaux

En utilisant les commandes des travaux pratiques 5, créer une fonction qui à partir de la valeur de l'entier n, renvoie un vecteur ligne Y de la forme : $Y = \begin{pmatrix} \binom{n}{0} & \binom{n}{1} & \binom{n}{2} & \binom{n}{3} & \dots & \binom{n}{n} \end{pmatrix}$.

Travaux pratiques 15 : Fonction calculant $\max(0, S-K)$

Ecrire une fonction, utilisant l'instruction "if ... then ... else", qui à partir des valeurs des deux réels S et K calcule $\max(0, S-K)$.

Travaux pratiques 16 : Espérance C d'une variable aléatoire discrète

Soit S_0, u et K trois réels strictement positifs fixés, soit N un entier naturel non nul. Soit S la variable aléatoire discrète finie telle que :

$$\begin{aligned} S(\Omega) &= \left\{S_0 u^{N-2k},\ k \in [\![0,N]\!]\right\} \\ \forall k &\in [\![0,N]\!],\ P(S = S_0 u^{N-2k}) = \binom{N}{k} p^{N-k} (1-p)^k \end{aligned}$$

On note C la valeur de l'espérance de la variable aléatoire $\max(0, S-K)$. Montrer que :

$$C = \sum_{k=0}^{N} \binom{N}{k} p^{N-k} (1-p)^k \max\left(0, S_0 u^{N-2k} - K\right)$$

Ecrire un programme, utilisant les deux fonctions précédentes qui calcule la valeur de C. On prendra les valeurs numériques suivantes :

$$S_0 = 42,\ u = 1,01424,\ p = 0.514146,\ N = 100,\ K = 40$$

Le résultat obtenu pour C doit être égal à 5 environ.
Remarque : Le programme ci-dessus permet de calculer la valeur d'une option d'achat sur actions à l'aide du modèle de Cox, Ross, Rubinstein (1979).

Travaux pratiques 17 : Calcul d'une estimation de C par la méthode de Monte Carlo.

A l'aide du générateur **`grand`**, simuler la variable aléatoire S. En déduire une estimation de C par la méthode de Monte Carlo (on effectuera n simulations de la la variable aléatoire S, ce qui nous donnera 1000 valeurs prises par $\max(0, S-K)$, la moyenne de ces valeurs donne alors une estimation de C). On pourra prendre $n = 1000$, $n = 10000$, puis $n = 100000$. Comparer avec la valeur obtenue précédemment.

8 Table des travaux pratiques du document

9 Bibliographie et sites internet

Il y a peu d'ouvrages français sur Scilab. On peut citer :

– **Débuter en Algorithmique avec Matlab et Scilab** de **Jean-Pierre Grenier**, **Ellipses**.

Remarque : Cet ouvrage a été publié en 2007, donc il ne tient pas compte des modifications intervenues dans les dernières versions. Certaines commandes (en particulier graphiques) citées dans l'ouvrage ne sont plus utilisables dans les versions 5.x du logiciel.

– **Algorithmique et Mathématiques - Travaux pratiques et applications Scilab** de **José Ouin Ellipses** - Parution : 26-04-2010 - ISBN : 9782729854393.

Il y a de nombreux **sites internet** avec des **documents très complets**, en particulier :

1. **Le site de Scilab**
   http://www.scilab.org/fr/resources/documentation/tutorials
2. **Probabilité et statistiques avec Scilab**, Préparation à l'agrégation de mathématiques,
   Université Joseph Fourier (Grenoble 1) – Adresse internet :
   http://www-fourier.ujf-grenoble.fr/~decauwer/polyscilab.pdf
   Remarque : excellent document datant de **septembre 2010**.
3. **Une introduction à Scilab**, cours de **Bruno Pinçon**.
   Adresse internet : http://www.iecn.u-nancy.fr/~pincon/scilab/docA4.pdf
   Remarque : document très complet, attention certains chapitres ne traitent pas des versions ultérieures à la version 4.0.
4. **Démarrer en Scilab** de **B. Ycart** (Université René Descartes, Paris)
   Adresse internet :
   http://ljk.imag.fr/membres/Bernard.Ycart/polys/demarre_scilab/demarre_scilab.html
   Remarque : document complet.
5. **Réaliser des graphiques avec Scilab & Introduction à Scilab** de **Philippe Roux** (**mai 2010**).
   http://perso.univ-rennes1.fr/philippe.roux/scilab/intro/fiche_scilab.pdf
   http://perso.univ-rennes1.fr/philippe.roux/scilab/graphiques/fiche_graphiques.pdf
   Remarque : document très utile pour comprendre le nouveau mode graphique de Scilab 5.x .