Introduction à Scilab

1 Qu'est-ce que Scilab ?

Scilab est un logiciel de **calcul numérique**, c'est-à-dire qu'il permet d'effectuer des calculs sur des valeurs approchées réelles ou complexes (comme les calculatrices usuelles), à l'inverse de logiciels comme Maple qui permettent de faire du **calcul symbolique** (résolution exacte d'équations, calcul exact d'intégrales, manipulation d'expressions algébriques, etc.).
Scilab peut se voir comme une « calculatrice améliorée » spécialisée dans les **calculs sur des matrices**, avec une vaste bibliothèque de fonctions prédéfinies, d'importantes capacités de **représentation graphique** de données, le tout pouvant faire l'objet d'une **programmation**.
Scilab est un **logiciel libre gratuit**, téléchargeable à `http://www.scilab.org`, disponible sur toutes les plateformes usuelles (Windows, Mac, Linux), qui a initialement été développé par des chercheurs de l'INRIA et de l'ENPC. Il existe de nombreux programmes similaires à Scilab, parmi lesquels on peut citer Matlab (le plus connu, qui est un logiciel propriétaire payant), Octave (libre gratuit, très proche de Matlab) ou R (libre gratuit, spécialisé en statistiques). Du côté des logiciels de calcul symbolique, citons Maple et Mathematica (propriétaires, payants), ainsi que Maxima et Sage (libres, gratuits).
NB. Bien que proche de Matlab, Scilab n'en est pas un clône, et les programmes écrits pour Matlab doivent en général être légèrement retouchés pour fonctionner avec Scilab. Scilab fournit un outil de conversion, pour les modifications les plus simples.

2 Manipulation du logiciel

NB. L'interface du logiciel (menus, icônes, raccourcis au clavier) dépend de la version installée, pas les commandes.

2.1 Fenêtres

Au lancement, Scilab ne présente qu'une fenêtre : la **console**. Et éventuellement, selon les versions, d'autres sous-fenêtres contenant la liste des variables ou une arborescence de fichiers. On aura constamment besoin d'une autre fenêtre : l'**éditeur**, qui s'obtient par le menu ("Applications"). De plus, lorsque l'on demandera des graphiques, ils apparaîtront dans de nouvelles fenêtres de **figures**. Enfin, on aura souvent une fenêtre ouverte avec l'**aide** de Scilab (qui s'obtient via le menu, ou **F1**).

1) Ouvrir l'éditeur et l'aide. Dans l'éditeur, taper "`6*7`", sauvegarder le fichier (Ctrl-S, et donner un nom avec l'extension `.sce`) puis taper ctrl-L (ou, par le menu, cliquer sur "Exécuter avec écho") : la console affiche le résultat.

2) Dans la console, taper `x=0:0.01:5;plot(x,sin(x))` puis Entrée, pour un exemple de figure (qui sera expliqué après). Copier et coller cette commande dans l'éditeur, sauvegarder (ctrl-S) et taper à nouveau ctrl-L.

2.2 Modes d'utilisation

Comme on vient de le voir, il y a deux façons complémentaires d'utiliser Scilab :
– par la console, en tapant une commande puis Entrée pour l'exécuter
– par l'éditeur, en tapant une suite de commandes (une par ligne) puis en les faisant exécuter par Scilab une par une.
Un inconvénient de la première méthode est de ne pas sauvegarder dans un fichier les commandes lancées (ceci dit, la touche flèche ↑ permet de retrouver les commandes précédentes). On utilisera surtout la console pour tester le fonctionnement d'une commande avant de l'intégrer à un fichier dans l'éditeur, ou pour un calcul très bref qu'on n'a pas besoin de reproduire. Ce sera le cas de tout le début du TP.
Pour la seconde méthode (éditeur), on peut exécuter le fichier de deux façons :
– avec **ctrl-L** ("exécuter le fichier avec écho") pour afficher dans la console les résultats de chaque ligne, à l'exception de celles qui finissent par un point-virgule.
– avec ctrl-E ou **F5**, pour ne pas afficher les résultats dans la console, à moins de le demander explicitement (avec `disp(variable)`, cf. suite). Avec F5, Scilab commence par sauvegarder le fichier, ce qui est un bon réflexe à avoir.
→ Dans le début du TP, on pourra tester les commandes sur la console, puis les copier dans l'éditeur pour sauvegarde.

2.3 La console comme calculatrice : nombres réels et complexes, booléens

- **nombres réels** : `14.54` (utiliser un point et non une virgule), `2.2D-16` ($2{,}2 \cdot 10^{-16}$)
- **opérations** : `+`, `-`, `*`, `/`, `^` (exponentiation, avec les touches AltGr et 9), parenthèses `(` et `)` autour d'expressions
- **fonctions** : `floor` (partie entière), `abs` (valeur absolue), `sqrt` (racine carrée), `exp` (exponentielle), `log` (logarithme népérien), `sin`, `cos`, `tan`, `asin`, `acos`, `atan`, `real` (partie réelle), `imag` (partie imaginaire), `conj` (conjugué), `sinh`, etc., en plaçant l'argument entre parenthèses : `atan(1)`
- **constantes** : `%i` (`sqrt(-1)`), `%pi` (`4*atan(1)`), `%e` (`exp(1)`)

NB. Tous les calculs sont approchés, il faut donc être vigilant vis-à-vis de possibles erreurs d'arrondis.
La commande `format(10)` affiche 10 chiffres significatifs au lieu de 8, **mais** ne change pas la précision des calculs.
La constante `%eps` ($2^{-52} =\simeq 10^{-16}$) donne la distance en-deçà de laquelle deux réels sont considérés égaux.

3) Calculer $(\sqrt{2})^2 - 2$, $e^{i\pi}$, $(1+\sqrt{2})^{10}$, $3\sqrt[12]{2}\sqrt[7]{5}$, $e^{\pi} - \pi$, $\left(-\frac{1}{2} + i\frac{\sqrt{3}}{2}\right)^3$, `1+%eps-1`, `1+%eps/2-1`, `0.1+0.2-0.3`

- **constantes booléennes** : `%T` (vrai), `%F` (faux)
- **opérations** : `&` (et), `|` (ou), `~` (non, obtenu aves les touches AltGr et 2)
- **comparaisons (à valeurs booléennes)** : `<`, `<=` (inférieur ou égal), `>`, `>=`, `==`, `<>` (différent)

4) Tester `2<3`, `1+1==2`, `tan(atan(1))==1`

De plus `%T` est assimilé à `1` et `%F` à `0` dans les opérations.

5) Que renvoie `(2>1)*9` ?

2.4 Variables

Après chaque calcul, la console affiche "`ans =`" puis le résultat. En fait, `ans` (pour "answer") est une **variable** à laquelle Scilab affecte le résultat du dernier calcul, et que l'on peut réutiliser ensuite dans une commande.
NB. La variable `ans` n'est utile que pour les calculs dans la console, pas dans l'éditeur.

6) Trouver les 5 premiers coefficients du développement en fraction continue de π ($\pi = a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \cdots}}$ avec $a_i \in \mathbb{N}^*$)
Indication : dans la console, taper `%pi+0` (pour faire un calcul dont le résultat est π), puis `1/(ans-floor(ans))`, etc.

7) Trouver d'ici 30 secondes une valeur approchée à 10^{-4} d'un réel a tel que $a > 0$ et $a = 2\sin(a)$. Indication : utiliser une suite définie par récurrence convergeant vers a. Utiliser `ans` pour calculer les termes successifs.

On peut aussi définir ses propres variables pour réutiliser le résultat d'un calcul, par la commande d'**affectation** "`=`".
Par exemple, `a=sqrt(2)` affecte à la variable `a` le résultat de l'opération `sqrt(2)`, et l'on peut ensuite demander `a^2-2`, `b=(a+1)^2` ou même `a=1/a` (qui assigne à `a` le résultat de l'opération `1/a` : attention, dans Scilab `=` n'est pas un signe d'égalité mais d'affectation, à la différence de `==`).
En guise de nom de variable, toute suite de caractères est possible (hormis les mots réservés par Scilab), par exemple `racine_de_deux=sqrt(2)` ou `r2=sqrt(2)`. On pourrait même poser `sqrt=sqrt(2)` mais la fonction `sqrt` ne serait plus définie ensuite... D'ailleurs, Scilab imprime un avertissement.

8) Calculer $\frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}-1} + \frac{1}{\sqrt{2}+1}\right)$ en utilisant une variable intermédiaire

3 Erreurs et aide

3.1 Erreurs

En cas d'erreur dans une commande, Scilab indique la ligne (dans l'éditeur) et en général la nature de l'erreur.

9) Tester `1/0`, `srqt(3)`, `2*(1+1, 1+2)`, `*1`, `asin(2)`, `tan(%pi/2)`

Programme bloqué. Si par erreur on a demandé à Scilab un calcul trop long (ou infini), on peut l'interrompre manuellement via le menu "Contrôle→Abandonner", ou en tapant Ctrl-C puis `abort`.

3.2 Aide

Scilab possède une aide assez complète (essentiellement en anglais, mais en partie en français aussi), en général assortie d'exemples. On y a accès par
– la touche **F1**, le menu ou la commande `help`
– taper `help tan` dans la console si l'on souhaite de l'aide sur la fonction `tan`
– taper `apropos tangent` pour voir les pages contenant le mot `tangent` (peut aussi se faire depuis la fenêtre d'aide).
De plus, dans la console ou l'éditeur, on peut utiliser la touche tabulation (avec deux flèches) pour **compléter automatiquement** la commande (taper `sq` puis tabulation pour tester).

4 Vecteurs et matrices

Matlab signifie "Matrix Laboratory", et Scilab est aussi tout particulièrement conçu pour manipuler des matrices. Comme d'habitude, une **matrice** est un tableau de valeurs réelles ou complexes, et on parle de **vecteur** s'il n'a qu'une ligne, ou une colonne. Pour Scilab, toute variable est une matrice, les scalaires étant des matrices à une ligne et une colonne.

4.1 Définir une matrice

De façon explicite, on peut définir

```
A=[ a11, a12, a13;
a21, a22, a23]
```

(le passage à la ligne est facultatif). On retient qu'une **virgule** (ou un **espace**) sépare les colonnes, et un **point-virgule** sépare les lignes. La **matrice vide** est `[]`.
On peut ensuite faire appel à `A(1,1)`, `A(1,2)`, etc. pour accéder et modifier les coefficients. Pour les vecteurs, il suffit d'indiquer la composante significative : `A(5)` au lieu de `A(5,1)` (si c'est un vecteur-colonne) ou `A(1,5)` (vecteur-ligne).
Attention. Les indices des matrices (et donc des vecteurs) commencent toujours à 1.

10) Définir la matrice $\begin{pmatrix} 1 & i \\ -i & 1 \end{pmatrix}$ et les vecteurs $\begin{pmatrix} 3 & 1 & 5 \end{pmatrix}$ et $\begin{pmatrix} 4 \\ 0 \\ -7 \end{pmatrix}$

Dans le cas de grandes matrices (par exemple des données, en statistiques), on se donne plutôt un fichier texte qui liste les coefficients ligne par ligne, en les séparant par des espaces (ou des tabulations), et on l'importe sous forme de matrice `A` par `A=fscanfMat('fichier')`. Quelques lignes de texte au début du fichier peuvent décrire la nature des données.

11) Télécharger le fichier `http://www.math.univ-paris13.fr/~tournier/matrice.txt` (avec un navigateur internet) puis le charger dans Scilab

On peut aussi définir une matrice en partant d'une fonction qui renvoie une matrice particulière :
– `zeros(m,n)` (matrice nulle), `eye(n,n)` (matrice identité), `ones(m,n)` (matrice constituée de 1), `diag([a1, ..., a_n])` (matrice diagonale). Sous la forme `zeros(A)`, `eye(A)`, `ones(A)` ces matrices sont de même taille que `A`
– pour obtenir des vecteurs lignes équirépartis, `linspace(x1,x2,n)` (de `x1` à `x2` en `n` colonnes) ou `x1:delta:x2` (de `x1` à `x2` (au plus) par accroissements de taille `delta` qui peut être négatif), ou `x1:x2` (idem, avec `delta=1`) ; vu les problèmes d'arrondis, `linspace` est préférable si on veut être sûr du nombre de colonnes

12) Définir les matrices $\begin{pmatrix} 1 & 2 & 3 & \cdots & 15 \end{pmatrix}$, $\begin{pmatrix} 0 & \frac{\pi}{8} & \frac{2\pi}{8} & \cdots & \pi \end{pmatrix}$ et $\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ (sans taper chaque coefficient)

`size(A)` fournit la taille de `A` (c'est un vecteur : `[lignes, colonnes]`)
Pour un vecteur, `length` fournit sa longueur (c'est un entier).

4.2 Sous-matrices

Si `u` et `v` sont des vecteurs d'entiers, `A(u,v)` est la sous-matrice correspondante. L'exemple essentiel est `A(k1:k2,l1:l2)` pour la sous-matrice donnée par les lignes `k1` à `k2` et les colonnes `l1` à `l2`.
Cas particuliers : "`$`" désigne le plus grand indice (par exemple, `A(2:$,1:2)`), et "`:`" désigne tous les indices (par exemple, `A(:,2)` donne la deuxième colonne).

13) Demander la taille de la matrice précédente, et extraire sa première ligne et sa dernière colonne

On peut changer toute une sous-matrice : `A(2:3,1:3)=zeros(2,3)`
voire si possible la supprimer (modifiant la dimension) : `A(:,2)=[]` supprime la deuxième colonne.

4.3 Opérations

On peut concaténer (« accoler ») des matrices : `[A, B]` place `A` et `B` côte-à-côte et `[A ; B]` place `A` et `B` l'une par-dessus l'autre (si les dimensions le permettent).
Par exemple, pour ajouter un élément à la fin du vecteur-ligne `v` : `v=[v, 42]`
De plus,
– `A'` est la transposée de `A`

14) Tester avec une matrice complexe : est-ce juste la transposée ?

15) Calculer le produit scalaire des vecteurs lignes `x=1:10` et `y=10:-1:1`

– `*`, `^`, `+` sont les opérations matricielles usuelles sur les matrices et s'appliquent aussi entre matrices et scalaires

16) Que donne l'opération `A+5` où `A` est une matrice ? Que donne `2^A` où `A` est une matrice ? Et `A^-1` si `A` est carrée ?

– `.*`, `.^`, `./` sont des opérations coefficient-par-coefficient

17) Pour une matrice `A` inversible (`A=ones(3,3)+diag([0,1,2])`), comparer `1/A`, `1./A`, `1 ./A`, `ones(A)/A` et `ones(A)./A`

18) Définir les vecteurs $(n^2)_{1\leq n\leq 10}$ et $(\frac{n}{1+n^2+n^3})_{1\leq n\leq 10}$ (commencer par poser `x=1:10` et faire des calculs sur `x`)

– `<`, `>`, `<=`, etc., s'appliquent coefficient-par-coefficient entre matrices ou avec un scalaire

19) Poser `v=rand(1,10)` et regarder le résultat de `v<0.5`

La plupart des fonctions sur les réels (ou complexes) s'appliquent aussi coefficient-par-coefficient : `sin(A)`, `exp(A)` (ce n'est pas l'exponentielle matricielle `expm`), `sqrt(A)` (ce n'est pas la racine matricielle `sqrtm`), etc.

20) Définir les vecteurs lignes $(\sin\frac{k\pi}{10})_{0\leq k\leq 10}$, $(\exp(k^2)-k)_{1\leq k\leq 10}$ et $((k+5)^k)_{1\leq k\leq 10}$ (commencer par poser `x=1:10` et faire des opérations à partir de `x`)

21) Faire afficher une table de multiplication (la matrice $(ij)_{0\leq i,j\leq 10}$) sans écrire de boucle (par un produit matriciel)

22) Étant donné un vecteur ligne `x` (disons `x=1:5`), comment obtenir une matrice formée de 7 lignes égales à `x` ? (plusieurs méthodes)

– `sum(A)` est la somme des coefficients de `A`
– `sum(A,1)` est la somme selon la première coordonnée, c'est un vecteur-ligne donnant la somme de chaque colonne
– `sum(A,2)` est de même un vecteur-colonne donnant la somme de chaque ligne
– `mean(A)`, `variance(A)` donnent la moyenne et la variance des valeurs de `A`
– `cumsum(A)` calcule les sommes cumulées (suite des sommes partielles) et `cumsum(A,1)` ou `cumsum(A,2)` effectue cette opération colonne-par-colonne ou ligne-par-ligne.
– `prod(A)` est le produit des coefficients de `A` (avec les mêmes variations que `sum`)

23) Calculer $\sum_{k=1}^{N} \frac{1}{k^2}$, $\sum_{k=0}^{N} \sin\frac{k\pi}{2N}$ et $\prod_{k=2}^{N}\left(1-\frac{1}{k^2}\right)$ où $N = 1000$

– si `v` est un vecteur de booléens (obtenu avec un test, cf. 19), `find(v)` renvoie les indices où `v` est égal à `%T`.

24) Quels éléments de la suite $(\sin(n))_{1\leq n\leq N}$ sont-ils positifs ? (choisir une valeur de N, par exemple `N=100`) Combien y en a-t-il ? Définir la suite restreinte à ces indices. Et combien de termes appartiennent à $[0,\frac{1}{2}]$?

– `det(A)`, `kernel(A)`, `[P,D]=spec(A)` donnent le déterminant, une base du noyau (en colonnes) et le spectre (dans `D`) ainsi que des vecteurs propres (dans `P`, dans le même ordre) de la matrice carrée `A`
– `linsolve(A,b)` renvoie une solution de $AX + b = 0$, s'il y en a une (même si `A` est singulière, ou non carrée)

25) Poser `A=[zeros(1,5);rand(4,5)]`. Donner le spectre de cette matrice, et un vecteur du noyau (via `spec` et `kernel`). Vérifier $PDP^{-1} = A$. Même chose avec `A=[1,2;0,1]`.

26) Résoudre $\begin{cases} x_1 + x_2 + x_3 &= 3 \\ x_1 - x_2 + x_3 &= 1 \\ x_1 - 3x_2 + x_3 &= -1 \end{cases}$

5 Représentations graphiques

Pour afficher la valeur d'une variable dans la console même quand on exécute sans écho (avec F5), on utilise `disp(x)` ("display"), qui peut aussi servir à afficher une chaîne de caractères.

27) Tester `x=5;disp('x =');disp(x);`

Pour afficher de nombreuses valeurs, il est souvent préférable de les représenter graphiquement.

5.1 Graphes avec "plot"

Partant de deux vecteurs `x` et `y` de même longueur, `plot(x,y)` représente les points `(x(i),y(i))` et les relie dans l'ordre d'apparition.

```
x=0:0.01:1; plot(x,sqrt(x))
```

L'échelle s'adapte automatiquement aux valeurs, et la couleur est choisie dans une liste (elle change cycliquement). Les graphes successifs se superposent. On utilise `clf` pour vider la fenêtre.

28) Représenter les graphes de sin et cos sur $[0,\pi]$ (avec 100 points, par exemple)

29) Représenter le graphe de $\frac{\sin x}{x}$ sur [0,100]

On modifie le rendu via un troisième paramètre : par exemple `plot(x,y,'+r')` ou `plot(x,y,'g')`. Description :
– les caractères `.+xo-` changent le type de ligne (points, croix, croix, rond, trait continu)
– les caractères `kbrgcym` changent la couleur (black, blue, red, green, cyan, yellow, magenta)

30) Représenter en points bleus les 30 premiers termes de la suite $u_n = \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k}$ (penser à `cumsum`) et en rouge une droite correspondant à sa limite $\ln 2$

On peut ajouter titre, noms des axes et légende (s'il y a plusieurs courbes) :
- `xtitle('titre','axe X','axe Y')` donne un titre au graphe et aux axes
- `legend(['courbe 1', 'courbe 2', 'courbe 3'],2)` donne des légendes pour chaque courbe (dans l'ordre de dessin), et `2` indique la position (`2` pour en haut à gauche, `1` pour en haut à droite,...)

31) Représenter les fonctions $x \mapsto x^k$ sur [0,2] pour $k \in \{0.5,1,2,3,4\}$ et ajouter titre et légendes (après la section suivante, on pourra faire une boucle pour automatiser la répétition des commandes)

Pour modifier la plage de valeurs représentée, on peut utiliser :
- `zoom_rect([xmin,ymin,xmax,ymax])` (attention aux crochets : le paramètre est un vecteur) *(rarement utile)*
- `isoview(xmin,xmax,ymin,ymax)` pour avoir la même échelle sur les deux axes (attention à l'ordre des paramètres)

32) Dessiner un cercle de centre (0,0) et de rayon 2 : par équation cartésienne, et aussi par équation paramétrique

33) Représenter la courbe d'équation polaire $r = \sin^2 \frac{\theta}{2}$ (toujours avec la fonction `plot`)

5.2 Gestion de la fenêtre graphique

Pour afficher plusieurs graphes simultanément, on utilise en général plusieurs fenêtres.
On commence par choisir une fenêtre par `scf(0)` ("select current figure") pour ouvrir la fenêtre 0 par exemple, puis `scf(1)`, etc.
Toutes les opérations (y compris `clf`) se rapportent alors la figure courante.

```
x=0:0.01:1;
scf(0);clf;
plot(x,ones(x)./(1+x.^2));
scf(1);clf;
plot(x,x.^2);
```

34) Tester cet exemple

On peut aussi afficher plusieurs graphes dans la même fenêtre : `subplot(m,n,k)` découpe la fenêtre courante en `m` lignes et `n` colonnes, et définit comme fenêtre courante la `k`-ième (dans le sens de lecture). Par exemple,

```
x=linspace(0,2*%pi,100);
subplot(2,1,1);
plot(x,sin(x));
subplot(2,1,2);
plot(x,cos(x));
```

35) Tester cet exemple, puis ajouter en-dessous le graphe de $\sin + \cos$ sur $[0,2\pi]$

5.3 Quelques autres types de graphes

- `histplot(n, x)` donne un histogramme des valeurs dans le vecteur `x`, découpées en `n` classes ; on choisit la couleur via un troisième paramètre, à valeurs entières
- `plot2d2(x,y)` dessine la fonction constante par morceaux qui vaut `y(i)` sur `[x(i),x(i+1)[` (la dernière valeur de `y` n'est pas utilisée) ; on choisit la couleur via un troisième paramètre, à valeurs entières

36) Tracer un histogramme des N premières valeurs de la suite $(\{n\sqrt{2}\})_{n\geq 1}$, où $\{x\} = x - \lfloor x \rfloor$ est la partie fractionnaire de x. Quelle est la propriété ainsi illustrée ?

6 Bases de programmation

Dorénavant, on utilisera l'éditeur et non la console : on aura souvent besoin d'écrire plusieurs lignes. On pourra également sauvegarder dans plusieurs fichiers afin d'éviter de relancer toutes les instructions précédentes à chaque fois.

6.1 Séquence de commandes

En écrivant une commande par ligne dans l'éditeur, celles-ci seront exécutées l'une après l'autre par Scilab en pressant ctrl-L ou **F5**. C'est la forme la plus simple de programme : une suite de commandes.

```
commande1
commande2
...
```

Il est possible de mettre plusieurs commandes sur une ligne, en les séparant par des points-virgules `;`. En fait, on peut toujours mettre un point-virgule à la fin d'une commande : si on utilise **F5**, ça n'a pas d'intérêt, par contre si on utilise **ctrl-L** ("exécuter le fichier avec écho"), alors seul le résultat des commandes sans point-virgule final apparaît dans la console, ce qui peut être utile pour suivre l'exécution du programme et parfois repérer une erreur.

37) Comparer les résultats de l'exécution des deux lignes suivantes dans l'éditeur avec **F5** et **ctrl-L** :

```
a=%i
b=a^2;
```

6.2 Commentaires

On peut (et même on *doit*) compléter son programme par des commentaires, c'est-à-dire du texte explicatif. Pour qu'il ne soit pas considéré comme une commande, on le précède de `//`

```
// Commentaire
commande1 // commentaire sur la commande
commande2
```

6.3 Branchement conditionnel

Scilab (comme tout language de programmation) permet aussi d'exécuter certaines commandes uniquement si une condition est réalisée, en utilisant la structure suivante :

```
if(condition)
    // Si la condition est vraie, les commandes suivantes sont exécutées
    commande_oui_1
    commande_oui_2
    ...
else
    // Si la condition est fausse, les commandes suivantes sont exécutées
    commande_non_1
    commande_non_2
    ...
end
```

où `condition` est un booléen (par exemple, `a<0`) et les points de suspension remplacent une ou plusieurs commandes.
Si la condition est vraie, alors le premier bloc de commandes est exécuté, sinon le deuxième est exécuté.
La partie `else ...` est optionnelle.
L'indentation des blocs de commandes n'est pas indispensable, mais facilite la relecture du programme.

6.4 Boucle "for"

Pour faire répéter à Scilab une suite de commandes plusieurs fois, on utilise la structure suivante :

```
for compteur=debut:fin
    // suite de commandes à répéter pour compteur = debut, debut+1,...,fin :
    commande_1
    ...
end
```

où `compteur` est un nom de variable, et `debut` et `fin` sont deux entiers (par exemple, `for i=1:10`). Ceci équivaut à :

```
compteur=debut
commande_1
...
compteur=debut+1
commande_1
...
(...)
compteur=fin
commande_1
...
```

Autrement dit le bloc de commandes est exécuté successivement avec `variable` égal à `debut` puis à `debut+1`, etc., jusqu'à `fin`.

38) Écrire un programme qui calcule avec une boucle "for" la somme et le produit des carrés des entiers de 1 à `N` (où `N` est une variable définie en début de programme). On pourra commencer par :

```
N=15
S=0 // somme initiale
P=1 // produit initial
for ...
```

*(Au passage, comparer l'exécution avec **F5** et **ctrl-L**)*

39) À l'aide de deux boucles "for" imbriquées, et d'une structure "if... then...", écrire un programme (bête) qui compte le nombre de paires (i,j) d'entiers positifs telles que $i^2 + j^2 < 1000$.

En fait, on peut définir plus généralement la structure "`for compteur=vecteur`" qui exécute la suite de commandes avec `compteur` successivement égal à chaque coefficient du vecteur **ligne** `vecteur`.

40) Représenter graphiquement les fonctions $x \mapsto |x|^\alpha$ sur $[-1,1]$ pour 10 valeurs de α régulièrement espacées dans $[0,2]$. Indication : commencer par "`for alpha=linspace(0,2,10)`"

6.5 Boucle "while"

Enfin, on peut faire répéter à Scilab une séquence de commandes tant qu'une expression reste vraie :

```
while(condition)
    // Bloc de commandes répétées tant que condition est vraie
    commande_1
    ...
end
```

Attention, ce type de boucle peut tourner indéfiniment si la condition n'est jamais réalisée.

41) **Important : sauvegarder votre fichier avant de faire ce qui suit.** Tester `while(1<2);end`

42) Écrire un programme qui donne le nombre de termes de la suite de terme général $u_n = \sum_{k=1}^{n} \frac{1}{k}$ qui sont inférieurs à `M` (où `M` est une variable définie en début de programme, par exemple par `M=4`)

43) Écrire un programme qui calcule les termes de la suite définie par $u_0 = 1$ et $u_{n+1} = \cos(u_n)$ jusqu'à ce que deux termes successifs diffèrent de moins de `delta`, où `delta` est fixé en début de programme.

6.6 Fonctions

En plus des fonctions prédéfinies, on peut en écrire de nouvelles pour y faire appel ensuite. Pour définir une fonction `fonct` qui dépend de deux paramètres, on pourra écrire

```
function a=fonct(x1,x2)
    // suite de commandes qui calculent la fonction et placent sa valeur dans a
    commande1 // commande qui peut dépendre de x1 et x2
    ...
    a=...
endfunction
```

On pourra ensuite appeler par exemple `fonct(2,0.23)` (si les paramètres sont supposés scalaires). Cette expression sera égale à la valeur de `a` à la fin de l'exécution du bloc de commandes ci-dessus avec `x1` et `x2` égales à `2` et `0.23`.
Attention.
- Les fonctions se terminent par `endfunction` et non `end`
- Dans le programme ci-dessus, `a`, `x1` et `x2` sont des « variables muettes » : elles n'ont d'existence que pour l'écriture de la fonction, et n'apparaissent que dans le bloc de commandes de la fonction
- Plus généralement, toute variable utilisée dans une fonction est **locale**, c'est-à-dire qu'elle n'existe qu'au sein de la fonction. Ceci est une sécurité : une fonction doit pouvoir être réutilisée dans un programme différent, donc ne doit pas modifier les variables qui y sont définies. Si on veut utiliser la valeur d'une variable dans la fonction, il faut donc en faire un paramètre supplémentaire. *On pourrait aussi utiliser des variables globales.*
- pour faire appel à une fonction, il faut qu'elle ait déjà été définie : la placer au début du programme, ou même dans un fichier différent s'il est exécuté en premier (les fonctions définies dans un fichier peuvent être utilisées dans d'autres, une fois que le fichier avec les fonctions a été exécuté)

44) Définir la fonction $f : (x,y) \mapsto \max(x,y)$ (en fait la fonction `max(v)` existe déjà, pour un vecteur `v`)

45) Définir la fonction `fraccont(x,n)` qui renvoie le vecteur-ligne des `n` premiers coefficients du développement en fraction continue du réel `x` (où le premier coefficient peut être négatif, si `x` est négatif).

NB. Il est possible à une fonction de s'appeler *elle-même*, on parle de définition **récursive**. Attention aux boucles infinies dans ce cas : dans certains cas (selon les paramètres), la fonction doit être définie sans faire appel à elle-même, et tous les autres cas finissent par se ramener à ceux-ci.

46) Définir la fonction factorielle de façon récursive : selon le schéma $0! = 1$ et $n! = n \cdot (n-1)!$
Calculer 15! et évaluer l'erreur (en pourcentage) par rapport à la formule de Stirling.

47) Définir la fonction binôme de façon récursive : selon le schéma $\binom{n}{0} = 1$ et $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$ (ou 0 si $k > n$).

Une fonction peut aussi renvoyer plusieurs valeurs. On définit alors `function [a,b,...]=fn(x1,x2,...)`, et pour enregistrer le résultat de la fonction dans des variables on pose `[a,b]=fn(x,y,z)` par exemple.

48) Définir une fonction qui prend `a,b,c` pour arguments et renvoie les deux racines (éventuellement confondues) du polynôme $\mathtt{a}X^2 + \mathtt{b}X + \mathtt{c}$

6.7 Quelques conseils

Afin d'écrire des programmes plus simples à relire (et donc aussi moins sujets à erreur),
- bien choisir ses noms de variables et de fonctions (sans les rendre trop longs)
- utiliser des variables plutôt que des valeurs numériques : il sera plus facile de les modifier, et leur nom peut indiquer leur signification
- indenter boucles et branchements
- commenter les programme (pour soi et pour le jury) : rôles des variables et des fonctions, explications éventuelles
- définir des fonctions dès qu'un morceau de programme prend plusieurs lignes et ressert plusieurs fois (en faisant varier certains paramètres)
- répartir le code en plusieurs fichiers : fonctions d'une part (dans un ou deux fichiers, mises bout à bout), puis un fichier pour chaque "expérience", qui exécute une fonction avec certains paramètres numériques

7 Probabilités et statistiques

7.1 Nombres aléatoires

Scilab propose plusieurs fonctions de génération de nombres pseudo-aléatoires :
- `rand()` renvoie un nombre aléatoire de loi uniforme sur [0,1]. Les appels successifs à cette fonction sont censés renvoyer les valeurs de variables indépendantes.
- `rand(m,n)` renvoie une matrice de taille `(m,n)` de nombres aléatoires indépendants de loi uniforme dans [0,1]
- `grand(m,n,...)` renvoie une matrice de taille `(m,n)` de nombres aléatoires indépendants de loi spécifiée par les paramètres suivants : voir l'aide. Par exemple, `grand(m,n,'nor',0,1)` pour la loi $\mathcal{N}(0,1)$

Attention. Dans `grand` les paramètres ne sont pas toujours les paramètres standards : par exemple, pour la loi normale, les paramètres sont la moyenne et l'écart-type (et non la variance) ; et pour la loi exponentielle, le paramètre est la moyenne (et non l'inverse de la moyenne).

49) À l'aide de `rand()`, comment simuler une variable de loi Ber(p) ? Bin(n,p) ? Geom(p) ? $\mathcal{U}([a,b])$?

50) Si U suit la loi uniforme sur [0,1], alors $-\ln U$ suit la loi $\mathcal{E}(1)$ (et $\frac{1}{\lambda}\ln U$ la loi $\mathcal{E}(\lambda)$) et $\tan(\frac{\pi}{2}(2U-1))$ suit la loi de Cauchy (méthode d'inversion). Écrire des fonctions `randexp` et `randcauchy` pour tirer des nombres aléatoires selon ces lois à partir de `rand` : d'abord sans paramètre (renvoie un seul nombre), puis avec un paramètre `n` (renvoie un vecteur de `n` variables indépendantes)

51) Si $(\xi_n)_{n\geq 1}$ sont indépendantes et de loi $\mathcal{E}(\lambda)$, alors pour tout $t>0$ la variable aléatoire $N=\max\{n\geq 0 \,|\, \xi_1+\cdots+\xi_n<t\}$ suit la loi $\mathcal{P}(\lambda t)$. En déduire une fonction pour simuler la loi $\mathcal{P}(\lambda)$ à partir de `rand`.

52) Calculer la moyenne `mean` et la variance `variance` empiriques d'un échantillon de variables de loi $\mathcal{E}(2)$

53) Représenter graphiquement la suite des moyennes $\frac{X_1+\cdots+X_n}{n}$ pour $n\geq 1$, où $X_i\sim\mathcal{U}([-1,1])$ sont indépendantes. Même question avec des variables aléatoires de Cauchy. Commentaire ?

54) Représenter graphiquement dans $\mathbb{R}^2$ 1000 points tirés indépendamment selon la loi de (X,Y) où X,Y sont indépendantes et de loi $\mathcal{N}(0,1)$

55) Représenter graphiquement la suite des sommes partielles $x+X_1+\cdots+X_n$ où $x\in\mathbb{Z}$, $P(X_i=1)=P(X_i=-1)=1/2$ (marche aléatoire simple sur $\mathbb{Z}$). Estimer la probabilité d'atteindre N (entier positif fixé) avant 0. Conjecturer sa valeur en fonction de x

56) L'urne « de Pólya » contient initialement 1 boule rouge et 1 boule noire. On tire une boule puis on la replace dans l'urne avec une nouvelle boule de la même couleur, et on répète cette opération. Représenter graphiquement l'évolution de la proportion de boules bleues. Étudier la répartition de cette proportion après un temps long (histogramme, pour commencer). Faire varier la composition initiale de l'urne.

7.2 Fonctions de répartitions

Les fonctions de la forme `cdf...` (`cdfnor`, `cdfpoi`,...) donnent la fonction de répartition et la fonction de répartition inverse. Chercher `statistics` dans l'aide.

8 Références

Il existe de nombreux documents d'introduction à Scilab sur internet. Je peux mentionner le suivant, destiné à une prépa agreg :
`http://www-fourier.ujf-grenoble.fr/~decauwer/polyscilab.pdf`