Familiarisation à Scilab pour le lycée

Claude Gomez
INRIA

Février 2007

L'objectif de cette familiarisation à Scilab est de faire découvrir les différentes fonctionnalités de base du logiciel, dans le cadre d'une utilisation en classes scientifiques de lycée : secondaire, STS, CPGE... Cette présentation s'adresse aux enseignants et se limite volontairement à l'essentiel. Elle doit permettre le démarrage dans l'utilisation de Scilab.

Vous pouvez réaliser toutes les commandes présentées dans le document avec une version de Scilab à partir de la 4.0.

Table des matières

1 Prise en main

Lorsque Scilab est lancé, la fenêtre de commande s'ouvre et après l'invite de commande « --> », il suffit de saisir une commande et de taper sur la touche « ENTRÉE » pour avoir le résultat. On utilise ainsi Scilab comme une super calculatrice.

Tout ce qui suit « --> » est donc entré par l'utilisateur. En dessous se trouve le résultat donné par Scilab.

Dans la fenêtre de commande, on ne peut remonter avec la souris pour exécuter à nouveau un calcul, en revanche on peut utiliser les touches « flèche HAUT » et « flèche BAS » du clavier pour récupérer une commande précédente en se promenant dans l'historique des commandes.

1.1 Calculs numériques simples

Dans Scilab tous les calculs sont numériques. Les nombres ont une précision maximale de 16 chiffres décimaux et leur valeur absolue est comprise entre environ 10^{-308} et 10^{+308} (norme IEEE-754), voir l'annexe).

```
--> 2+3
 ans  =
5
```

`%e` et `%pi` représentent respectivement e et π :

```
--> %e
 %e  =
2.7182818
--> %pi
 %pi  =
3.1415927
```

Par défaut les résultats sont affichés avec 10 caractères, comprenant le point décimal et le signe. Si l'on veut plus de chiffres on utilise la fonction `format`. Pour avoir par exemple 20 caractères, ce qui nous donnera 18 chiffres, on fait :

```
--> format(20)
--> %e
 %e  =
2.71828182845904509
--> %pi
 %pi  =
3.14159265358979312
```

Retour à l'affichage par défaut et quelques calculs.

```
--> format(10)
--> exp(log(10))
```

```
 ans  =
10
```

Noter que `log`, comme dans la grande majorité des logiciels de ce type, représente le logarithme népérien ln. On utilise les fonctions `log2` et `log10` pour avoir respectivement le logarithme en base 2 et en base 10.

`%i` représente le i des complexes en entrée et s'affiche i en sortie.

```
--> 2+3*%i
 ans  =
2 + i * 3
```

Quelques calculs :

```
--> sin(%pi/4)
 ans  =
0.7071068
```

```
--> sin(%pi)
 ans  =
1.225E - 16
```

La valeur de $\sin(\pi)$ ci-dessus n'est pas `0`, mais une valeur très petite qui correspond à `0` au vu de la précision de la machine : n'oublions pas que l'on fait du calcul numérique (voir l'annexe).

Si l'on ne veut pas que le résultat s'affiche, on utilise un point virgule « ; » à la fin de la ligne de commande, le calcul est effectué mais ne s'affiche pas.

```
--> (1+sqrt(5))/2
 ans  =
1.618034
```

```
--> (1+sqrt(5))/2;
```

1.2 Aide en ligne

Pour accèder à l'aide en ligne, on peut utiliser au choix le menu « ? » puis cliquer sur le sous-menu « Aide Scilab ». Sous Windows la touche « F1 » permet d'y accéder directement. On peut également exécuter la commande `help` :

```
--> help sin
```

Une partie de cette aide est en français, le reste en anglais. Les manuels coontiennent des exemples d'utilisation que l'on peut exécuter sous Scilab en utilisant le copier/coller entre la fenêtre d'aide et la fenêtre Scilab.

1.3 Les variables

Scilab n'est pas un système de calcul formel comme Maple, Mathematica ou MuPAD. Il n'y a pas de variables formelles. Toute variable doit avoir une valeur. Par exemple, demander la valeur de la variable `a` sans lui donner de valeur produit une erreur :

```
--> a
  !--error 4
undefined variable : a
```

Les messages d'erreur sont affichés en anglais.

En revanche si l'on affecte une valeur à la variable `a`, il n'y a plus d'erreur :

```
--> a=%pi/4
 a  =
0.7853982
--> a
 a  =
0.7853982
```

On peut utiliser n'importe quel nom de variable qui n'est pas déjà défini par le système :

```
--> Pisur2 = %pi/2
 Pisur2  =
1.5707963
```

Si l'on affecte pas de résultat à une variable, il existe une variable par défaut appelée `ans` (« answer » en anglais) à qui la valeur est affectée automatiquement :

```
--> tan(4*%pi/9)
 ans  =
5.6712818
--> ans
 ans  =
5.6712818
```

2 Les matrices et les calculs arithmétiques

Aujourd'hui la notion de matrices est très peu enseignée dans les classes du secondaire. Il est cependant utile de savoir comment les utiliser, le calcul matriciel étant à la base des calculs dans Scilab. En revanche, il est toujours possible d'utiliser les vecteurs en les considérant comme des données qui contiennent des suites de nombres.

2.1 Les matrices

En Scilab tout est matrice : par exemple un scalaire est une matrice de dimension 1×1.

Pour définir un vecteur colonne :

```
--> v=[1;2;3]
 v  =
 1
 2
 3
```

Pour définir un vecteur ligne :

```
--> v=[1 2 3]
 v  =
 1  2  3
```

Pour définir une matrice :

```
--> m= [1 2 3;4 5 6;7 8 9]
 m  =
 1  2  3
 4  5  6
 7  8  9
```

Un espace « » permet de passer d'une colonne à la suivante et un point virgule « ; » d'une ligne à l'autre.

Pour accéder aux éléments ou les modifier, on utilise des parenthèses :

```
--> m(2,3)
 ans  =
6
--> m(2,3)=23
 m  =
 1  2  3
 4  5  23
 7  8  9
```

L'opérateur « : » permet de définir des vecteurs de nombres. Par exemple pour définir un vecteur ligne avec des valeurs qui s'incrémentent de 1 :

```
--> 1:10
 ans  =
 1  2  3  4  5  6  7  8  9  10
```

ou qui s'incrémentent de 2 :

```
--> 1:2:10
 ans  =
 1  3  5  7  9
```

L'opérateur « : » permet aussi d'accéder à des parties de vecteurs ou de matrices. Ci-dessous la notation `v(5:10)` permet d'obtenir le sous vecteur $v_5, v_6, v_7, v_8, v_9, v_{10}$:

```
--> v=10:-1:1
 v  =
 10  9  8  7  6  5  4  3  2  1
--> v(5:10)
 ans  =
 6  5  4  3  2  1
```

Enfin l'opérateur « : » sert à désigner toutes les lignes ou toutes les colonnes d'une matrice. Par exemple pour avoir la deuxième ligne de la matrice `m` :

```
--> m(2,:)
 ans  =
 4   5   23
```

et la troisième colonne :

```
--> m(:,3)
 ans  =
 3
 23
 9
```

```
--> m(2,:)
 ans  =
 4   5   23
```

Le symbole « $ » désigne le dernier élément d'une ligne ou d'une colonne :

```
--> v($)
 ans  =
1
```

Pour obtenir la transposée d'une matrice ou d'un vecteur, il suffit d'utiliser l'apostrophe « ' » :

```
--> m'
 ans  =
 1   4   7
 2   5   8
 3   23  9
```

Pour résoudre un système linéaire, il suffit d'utiliser l'anti-slash « \ » :

```
--> x=m\[1;1;1]
 x  =
 −1
  1
  0
```

```
--> m*x
 ans  =
 1
 1
 1
```

On remarque encore que la troisième valeur du vecteur `x` trouvé n'est pas `0`, mais une valeur très petite qui correspond à `0` au vu de la précision de la machine (voir l'annexe).

Quelques fonctions utiles :

La fonction `size` retourne un vecteur ligne de dimension 2 qui contient la taille du vecteur ou de la matrice que l'on a mis en argument :

```
--> size(m)
 ans  =
 3  3
```

```
--> size(v)
 ans  =
 1  10
```

```
--> size(v')
 ans  =
 10  1
```

Si l'on utilise la chaîne de caractère « * » comme deuxième argument de la fonction `size`, elle retourne la multiplication des deux dimensions de son son premier argument. Cela permet d'obtenir la dimension d'un vecteur ligne ou colonne et ainsi de la récupérer dans une variable pour l'utiliser plus tard, dans une boucle par exemple :

```
--> v=[1 2 3]
 v  =
 1  2  3
--> n=size(v,"*")
 n  =
3
--> v=[1;2;3]
 v  =
 1
 2
 3
--> n=size(v,"*")
 n  =
3
```

La fonction `rand` permet d'obtenir une matrice aléatoire de dimension $m \times n$ avec des nombres compris entre 0 et 1 :

```
--> rand(3,4)
 ans  =
 0.2113249   0.3303271   0.8497452   0.0683740
 0.7560439   0.6653811   0.6857310   0.5608486
 0.0002211   0.6283918   0.8782165   0.6623569
--> rand(1,4)
 ans  =
 0.7263507   0.1985144   0.5442573   0.2320748
```

Les fonctions `sum` et `prod` calculent respectivement la somme et le produit des éléments de leurs argument :

```
--> sum(1:10)
 ans  =
55
--> prod(1:10)
 ans  =
3628800
```

La fonction `find` permet de rechercher des éléments dans un vecteur ou une matrice et retourne dans un verteur les indices correspondants.

Pour trouver tous les éléments du vecteur v plus petits que 5 on fera :

```
--> v=[1 5 2 8 14 7 3 2 1 6 12]; find(v<5)
 ans  =
 1  3  7  8  9
```

Le vecteur résultat $(1,3,7,8,9)$ nous indique que les éléments v_1, v_3, v_7, v_8 et v_9 sont plus petits que 5.

Et pour trouver tous les éléments du vecteur v égaux à 3 on fera :

```
--> v=[1 2 3 2 3 4 2 3 5]; find (v==3)
 ans  =
 3  5  8
```

Le vecteur résultat $(3,5,8)$ indique que les éléments v_3, v_5 et v_8 sont égaux à 3.

2.2 Les calculs arithmétiques

Attention : les opération arithmétiques usuelles sont des opérations matricielles !

« * » est la multiplication matricielle et « .* » est la multiplication élément par élément.

```
--> m=[1 2;3 4]
 m  =
 1  2
 3  4
--> m*m
 ans  =
 7   10
 15  22
--> m.*m
 ans  =
 1   4
 9   16
```

« / » est la *division matricielle*[1] et « ./ » est la division élément par élément.

```
--> m/m
 ans  =
 1  0
 0  1
--> m./m
 ans  =
 1  1
 1  1
```

Les opérations appliquées à des vecteurs ont des vecteurs comme résultat.

```
--> x=1:5
 x  =
 1  2  3  4  5
--> 2*sin(x)+x^2
 ans  =
 2.682942   5.8185949   9.28224   14.486395   23.082151
```

Attention : l'opérateur « / » représente la division matricielle, donc il est nécessaire d'utiliser la division élément par élément « ./ » pour réaliser le type de calcul suivant :

```
--> cos(x)./(2*sin(x)+x^2)
 ans  =
 0.2013843   −0.0715202   −0.1066545   −0.0451212   0.0122892
```

Attention au piège suivant : « 1./x » est considéré comme « 1. / x » et il faut mettre un espace entre « 1 » et « . » pour utiliser la division élément par élément « 1 ./x ».

[1] A/B est la matrice X solution de l'équation matricielle $X \times B = A$.

3 Le tracé de courbes en 2 dimensions

Scilab possède un grand nombre de commandes qui permettent de tracer des courbes et de modifier les caractéristiques des tracés : type du tracé, couleur, épaisseur, graduation des axes, etc. Nous ne présenterons ici qu'un aperçu de la fonction `plot`, dont la syntaxe est similaire à celle de Matlab. Il est aussi possible de tracer des surfaces, en utilisant par exemple la fonction `surf`, similaire également à celle de Matlab, et de réliser des animations.

On définit d'abord un vecteur de 100 valeurs régulièrement espacées entre $-\pi$ et π :

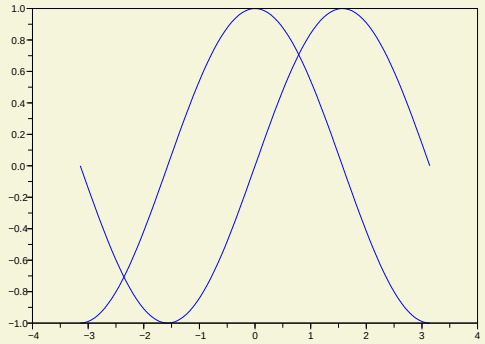
```
--> x=linspace(-%pi,%pi,100);
```

On trace la courbe de la fonction sinus :

```
--> plot(x,sin(x))
```

On peut superposer une autre courbe :

```
--> plot(x,cos(x))
```



On peut aussi tracer les deux courbes en même temps (la commande `clf` efface la figure) :

```
--> clf; plot(x,sin(x),x,cos(x))
```

Il faut noter que les arguments de la fonction `plot` sont toujours des nombres réels. Si l'on donne des nombres complexes comme argument, la fonction `plot` utilise leur partie réelle sans donner de message d'erreur. Par exemple, si l'on trace la fonction ln entre -10 et 10 en utilisant la commande

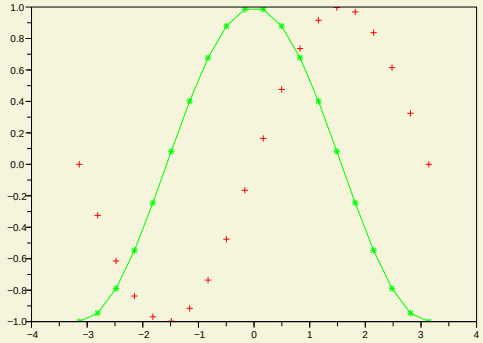
```
x=linspace(-10,10,100); plot(x,log(x)),
```

la fonction `plot` tracera entre -10 et 10 la partie réelle du logarithme qui est complexe pour ces valeurs. En fait, c'est $\Re(\ln(x))$ que l'on tracera.

Pour les couleurs, les tracés avec des marques, etc., essayez par exemple :

```
--> x=linspace(-%pi,%pi,20);
--> clf; plot(x,sin(x),"+r",x,cos(x),"*-g")
```

où « `"+r"` » signifie que l'on trace la courbe en rouge (`r` pour « `red` » = rouge) avec des points en forme de « + » et « `*-g` » signifie que l'on trace la courbe en vert (`g` pour « green » = vert) avec à la fois des points en forme de « * » et une ligne continue signalée par « - ».

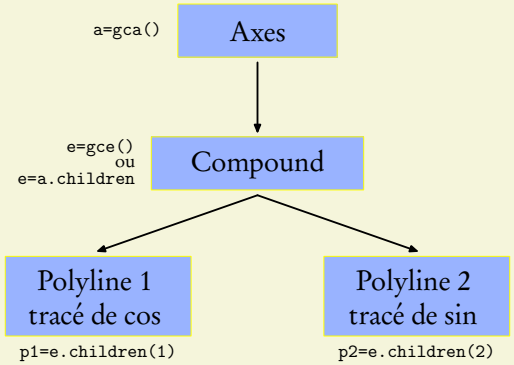


En fait les tracés de courbes sont réalisés dans des figures qui sont formées d'objets graphiques, comme les axes, les tracés (appelés « Polyline » pour les courbes), les labels etc. Ces objets ou parties de la figure sont appelés « entités » et possèdent des propriétés que l'on peut modifier.

On peut utiliser l'éditeur de courbes pour modifier tous les paramètres d'une figure. On y accède à partir du menu « Éditer » de la fenêtre graphique. Par exemple le sous-menu « Propriétés de la figure » permet d'accéder à toutes les parties de la figure et le sous-menu « Démarrer sélecteur d'entités » permet de cliquer avec la souris sur la partie de la figure que l'on veut modifier.

On peut aussi modifier les paramètres d'une figure à l'aide de commandes. Par exemple, traçons deux courbes par :

```
--> x=linspace(-%pi,%pi,100);
--> clf(); plot(x,sin(x),x,cos(x))
```



Hiérarchie du tracé

Sur la figure précédente, est dessiné de façon simplifiée l'arbre qui correspond au tracé effectueé. En effet, chaque traé est composés d'objets appelés « entités » qui ont des propriétés et des liens hiérarchiques entre eux. La commande `gca` (« get current axes ») permet d'accèder à l'entité « Axes » courant, c'est-à-dire aux axes de la figure que l'on vient de tracer et la commande `gce` (« get current entity ») permet à la dernière entité tracée, c'est-à-dire à l'entité « Compound » qui est l'ensemble des deux courbes tracées, appelées « Polyline ». Et l'on passe d'une entité à ses enfants en utilisant le champ `children`.

Donc en utilisant les commandes et la syntaxe suivante :

```
--> e=gce(); p1=e.children(1);
```

où l'on définit d'abord la variable `e` qui correspond à l'entité « Compound », c'est-à-dire à l'ensemble des deux courbes et la variable `p1` qui représente le tracé du cosinus, c'est-à-dire le premier fils de l'entité « Compound ». On pourrait de même définir la variable qui représente le tracé du sinus en écrivant `p2=e.children(2)`.

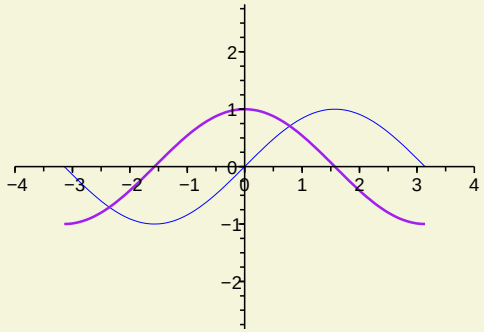
On peut alors modifier les propriétés des courbes, par exemple l'épaisseur et la couleur, de la courbe du cosinus :

```
--> p1.thickness=5;
--> p1.foreground=color("purple");
```

Et en utilisant la commande `gca`, on peut modifier les axes de la figure. Il est possible de changer par exemple leur épaisseur, la taille de la police des graduations et de rendre le repère orthonormé avec les axes se coupant en `(0,0)`.

```
--> a=gca();
--> a.thickness=3;
--> a.font_size=4;
--> a.x_location="middle";
--> a.y_location="middle";
--> a.isoview="on";
```

où `middle` signifie que l'axe doit passer par `0` et `isoview` signifie que les axes sont orthonormés.



Noter que les commandes précédentes ne sont valables que pour la figure courante. Si l'on veut modifier le comportement par défaut du graphique, la fonction `gda` (get default axes) doit être utilisée, car elle retourne les axes par défaut. Par exemple, si l'on veut des axes orthonormés et que les axes se coupent au point (0,0) tout le temps, on écrit une fois pour toute les commandes :

```
a=gda();
a.x_location="middle";
a.y_location="middle";
a.isoview="on";
```

Pour que Scilab exécute toujours ces commandes à son lancement, il faut les mettre dans son fichier d'initialisation. Celui-ci s'appelle « `scilab.ini` ». S'il n'existe pas, il faut le créer. Il doit se trouver dans un dossier particulier de la machine. Pour avoir le nom de ce dossier, taper `SCIHOME` dans Scilab. Par exemple sous Linux cela donne :

```
--> SCIHOME
 SCIHOME  =
/home/jms/.Scilab/scilab-4.1
```

4 Éditeur de texte

Pour lancer l'éditeur de texte, utilisez le menu « Éditeur ».

Il est recommandé d'écrire toutes les lignes de commandes et les fonctions dans l'éditeur et de les charger ensuite dans Scilab à partir du sous-menu « Charger dans Scilab » du menu « Exécuter ». Pour cela il faudra auparavant avoir sauvegardé le contenu de l'éditeur de texte dans un fichier, ce que ne manquera pas de demander Scilab.

On peut aussi bien entendu utiliser le copier/coller entre la fenêtre de l'éditeur et la fenêtre de commande de Scilab.

5 La programmation

5.1 Conditionnels

Les structures classiques suivantes existent :

- `if ... then ... else ... end`
- `if ... then ... elseif ... elseif ... end`

Les opérateurs de comparaison sont « == », « <> », « > » et « < ».

5.2 Les boucles

Quelques exemples de boucles :

```
--> for i=1:10; disp(i^2); end
1
4
9
16
25
36
49
64
81
100
```

```
--> a=1:10; for i=a; disp(i^2); end
1
4
9
16
25
36
49
64
81
100
```

```
--> for i=10:-1:1; disp(i^2); end
100
81
64
49
36
25
16
```

```
9
4
1
```

où la fonction `disp` (« `display` ») permet d'afficher les valeurs des variables.

On peut aller à la ligne au lieu de mettre des points virgules et ainsi écrire :

```
--> for i=1:10
-->   disp(i^2)
--> end
1
4
9
16
25
36
49
64
81
100
```

Voici un autre exemple qui mélange des conditions et une boucle. Il calcule les termes successifs de la fameuse suite de SYRACUSE :

$$u_n = \begin{cases} \dfrac{u_{n-1}}{2} & \text{si } u_{n-1} \text{ est impair} \\ 3u_{n-1}+1 & \text{sinon} \end{cases}$$

pour laquelle la conjecture est que cette suite finit toujours par boucler sur 4, 2 et 1. Ci-après `nmax` est le nombre maximal d'éléments de la suite. On a pris 1000 comme premier élément de la suite et à la fin de l'exécution, on trouve dans `n` le rang à partir duquel la suite boucle et dans `u` les éléments de la suite.

```
--> nmax=1000;
--> u(1)=1000;
--> for n=2:nmax
-->   if int(u(n-1)/2)*2 == u(n-1) then
-->     u(n)=u(n-1)/2;
-->   else
-->     u(n)=3*u(n-1)+1;
-->   end
-->   if u(n) == 1 then break end
--> end
--> n
 n  =
112
```

où la fonction `int` arrondit son argument vers 0 et `break` permet de sortir de la boucle.

Pour pouvoir refaire le calcul précédent avec diverses valeurs de u_1, on va utiliser des fonctions.

5.3 Les fonctions

Scilab dispose d'un langage qui permet de définir de nouvelles commandes grâce à l'écriture de fonctions.

Définissons une fonction simple dont le nom est `euros` avec un argument et une valeur de retour. L'argument est ici représenté par la variable `f` et la valeur de retour par la variable `r`. Notez que ces noms de variables n'ont aucun lien avec des variables de même nom qui pourraient être définies dans Scilab.

```
--> function r=euros(f); r=f/6.55957; endfunction
--> euros(100)
 ans  =
15.244902
```

La valeur retournée par la fonction doit être la valeur de la variable `r` à l'intérieur de la fonction.

Ici aussi, on peut aller à la ligne au lieu de mettre des points virgules et écrire :

```
--> function r=euros(f)
--> r=f/6.55957
--> endfunction
Warning :redefining function: euros
```

Scilab signale ici que l'on redéfinit une fonction.

Une fonction peut retourner plusieurs valeurs et avoir plusieurs arguments. Ici la fonction `minmax` a deux arguments et retourne deux valeurs.

```
--> function [m,M]=minmax(x,y)
-->  m=min(x+y); M=max(x+y);
--> endfunction
```

Pour récupérer toutes les valeurs on utilise les crochets « [» et «] ». Si l'on ne récupére qu'une seule valeur, c'est la première (la plus à gauche) qui est retournée :

```
--> x=rand(1,100); y=rand(1,100);
--> [a,b] = minmax(x,y)
 b  =
1.9605194
 a  =
0.1904729
--> a=minmax(x,y)
 a  =
0.1904729
```

Même s'il est possible de définir les fonctions dans la fenêtre de commande comme ci-dessus, il vaut toujours mieux les écrire dans l'éditeur, les sauver dans un fichier et les charger ensuite dans Scilab. Il sera ainsi beaucoup plus facile de réaliser des corrections ou des modifications. On peut charger le fichier correspondant directement à partir de l'éditeur en utilisant l'item « Charger dans Scilab » du menu « Exécuter ». Ou bien, si le fichier créé s'appelle par exemple « `toto.sce` », on pourra le charger dans Scilab en exécutant « `exec toto.sce` » dans la fenêtre de commande.

On peut alors écrire la fonction qui va calculer les éléments de

la suite de Syracuse du paragraphe précédent pour une valeur de u_1 donnée en reprenant le code écrit au paragraphe **5.2**. On appelle la fonction Scilab `syracuse`. Elle prend en entrée la valeur de u_1 et elle retourne à la fois le rang à partir duquel la suite boucle et la liste des éléments :

```
function [n,u]=syracuse(u1)
nmax=100000;
u(1)=u1;
for n=2:nmax
  if int(u(n-1)/2)*2 == u(n-1) then
    u(n)=u(n-1)/2
  else
    u(n)=3*u(n-1)+1
  end
  if u(n) == 1 then break end
end
endfunction
```

Il n'y a plus qu'à charger cette fonction dans Scilab et à l'exécuter :

```
--> exec syracuse.sce;
--> syracuse(1000)
 ans  =
112
--> [n,u]=syracuse(10000);
--> n
 n  =
30
--> u($-9:$)
 ans  =
 13
 40
 20
 10
 5
 16
 8
 4
 2
 1
```

Et on voit qu'en partant d'une valeur initiale de 1000, il faut 112 termes pour que la suite boucle et qu'il en faut seulement 30 en partant de 100000. À la fin nous n'avons affiché que les 10 derniers termes de la suite.

6 Annexe

6.1 Les calculs en nombres flottants et la norme IEEE-754

À la différence des logiciels de calcul formel, tous les calculs dans Scilab se font avec des nombres flottants, comme sur une calculatrice non formelle. Un nombre flottant est un nombre qui est codé sur un nombre entier de « mots machine » d'ordinateur, où un mot machine représente le nombre de bits que peut traiter l'ordinateur en une seule fois. Cela permet de faire des calculs très rapides car les opérations de base sont alors directement câblées dans la machine. Aujourd'hui, la plupart des ordinateurs ont des « mots machine » de 32 bits (4 octets). Sur 32 bits, on peut représenter 232 nombres, ce qui est peu. Donc on utilise généralement des nombres en double précision, soit codés sur 64 bits. Ce sont les nombres qu'utilise Scilab.

Utiliser des nombres flottants en double précision permet donc de réaliser des calculs rapides, mais comment coder ces nombres en utilisant les 64 bits disponibles ? On représente ces nombres sous la forme d'un signe, d'une mantisse qui représente le nombre de chiffres que l'on va utiliser et d'un exposant. On utilise en général le premier bit s pour coder le signe, e bits pour coder l'exposant et m bits pour coder la mantisse avec $s+e+m=64$. Il n'y a bien sûr pas unicité pour ce type de représentation et c'est la norme IEEE-754 (IEEE signifie « *Institute of Electrical and Electronics Engineers* »), normalement utilisée aujourd'hui par tous les ordinateurs qui font du calcul numérique, qui la définit comme suit :

où s est codé sur 1 bit, e est codé sur 11 bits et m est codé sur 52 bits. Par définition de la norme IEEE-754, le nombre représenté est égal à $(-1)^s \times 2^{e-1023} \times 1.m$ avec la notation « $1.m$ » qui représente les puissances négatives de 2 « après la virgule ». Par exemple si la mantisse m vaut 001110...0, « $1.m$ » représente $1+0\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}+1\times 2^{-4}+1\times 2^{-5}$. Et si le signe s vaut 0 et l'exposant e vaut 10000000010 soit 1026, le nombre représenté vaut finalement :

$$(-1)^0 \times 2^{1026-1023} \times \left(1+0\times 2^{-1}+0\times 2^{-2}+1\times 2^{-3}+1\times 2^{-4}+1\times 2^{-5}\right)$$

soit $2^3+1+2^{-1}+2^{-2}$, c'est-à-dire 9,75 en écriture décimale. Avec cette norme, on peut représenter en valeur absolue des nombres compris entre 2, 225... 10^{-308} et 1,797... 10^{308} , avec une précision de 2, 22... 10^{-16} , soit environ 16 chiffres significatifs. Cette précision est d'ailleurs la valeur de la variable `%eps` en Scilab.

Mais cela ne règle pas les problèmes d'arrondis. En effet, lorsqu'un calcul aboutit à un nombre non représentable par les nombres de la norme IEEE-754, il faut l'arrondir au nombre représentable le plus proche. La façon de le faire n'est pas unique : par défaut, par excès, au plus proche... La norme IEEE-754 définit donc aussi le calcul des arrondis pour les opé-

rations $+$, $-$, $\times$, $\div$, . Toutes les machines utilisant la norme IEEE-754 doivent donc donner le même résultat dans des calculs utilisant ces cinq opérations !

En revanche, la norme IEEE-754 ne spécifie pas le type d'arrondi pour les autres fonctions, en particulier les fonctions élémentaires[2].

6.2 Les problèmes de précision

Comme les calculs réalisés ne sont pas exacts mais effectués avec une précision donnée, on obtient des résultats avec cette précision. Par exemple on peut ajouter `%eps` à 1 :

```
--> format(20)
--> 1+%eps
 ans  =
1.00000000000000022
```

où l'on rappelle que la commande `format(20)` donne les résultats avec 20 caractères. Mais rajouter la moitié de `%eps` à 1 n'a aucun effet :

```
--> 1+%eps/2
 ans  =
1
```

Ce qui peut donner dans des cas extrêmes des résultats du type :

```
--> (1+%eps/2+%eps/2)-(1+%eps)
 ans  =
-0.00000000000000022
```

Donc le résultat obtenu dans le paragraphe **1.1** pour le calcul de $\sin(\pi)$ est correct puisqu'il est égal à 0 à `%eps` près.

La conclusion pratique est qu'en calcul numérique il est en général peu prudent, sauf si l'on connaît la grandeur des nombres que l'on manipule, de faire un test d'égalité sur deux nombres flottants dont on ne connaît rien. Il vaut mieux les comparer à `%eps` près, c'est-à-dire comparer la valeur absolue de leur différence à `%eps`. Comme dans l'exemple ci-dessus, on obtient les résultats suivants :

```
--> a=1+%eps/2+%eps/2;
--> b=1+%eps;
--> a == b
 ans  =
F
--> abs(a-b) <= %eps
 ans  =
T
```

où l'affichage *T* et *F* représente respectivement les valeurs booléennes *vrai* et *faux*. Ici `a` et `b` ne sont rigoureusement pas égaux, mais sont bien égaux à `%eps` près, car `%eps/2` ajouté à 1 vaut 1.

[2] Ce sont les fonctions bâties à partir des fractions rationnelles par application répétée des opérations $+$, $-$, $\times$, $\div$, de l'exponentielles, du logarithme et de l'opération de clôture algébrique. Par exemple les fonctions trigonométriques en font partie.