Informatique
TP 1

Révisions

Sur son lecteur personnel `K:\`, créer un dossier `Scilab` puis un sous-dossier `TP1` (on ajoutera un nouveau sous-dossier `TP` à chaque nouveau TP). Dans le menu Fichier de la console, cliquer sur « Changer le répertoire courant... » et choisir le répertoire `K:\Scilab\TP1\`.

Les scripts doivent être implémentés dans SciNotes (que l'on peut ouvrir depuis la console grâce au menu Applications/SciNotes, grâce à une icône de la barre d'outils ou par l'instruction `scinotes`) et enregistré dans le répertoire courant sous la forme d'un fichier `ex0.sce` où `0` est le numéro de l'exercice (voire `ex0.0.sce` si l'on veut préciser en plus le numéro de la question). L'application SciNotes peut être intégrée à la console par glisser-déposer du bandeau grisé horizontal « SciNotes ».

En cas de besoin, on n'hésitera pas à consulter l'aide de Scilab qui indique la syntaxe précise de chaque instruction, accompagnée de nombreux exemples. On pourra également consulter le poly de M. Kaddouri disponible en ligne sur son site.

Partie 1 : un peu de programmation

Les boucles peuvent être utilisées lorsqu'on veut faire exécuter à l'ordinateur une tâche répétitive. On distinguera les deux types de boucles ci-dessous :

- Les boucles `for`, à utiliser lorsqu'on sait à l'avance combien de fois l'action répétitive doit être exécutée. La syntaxe est la suivante :

```
for x=1:n
  // instructions à répéter
end
```

  On rappelle que l'instruction `1:n` créé un vecteur contenant les éléments de $[\![1, n]\!]$. Lors de l'exécution de la boucle, la variable `x` prend successivement les valeurs contenues dans le vecteur `1:n` (ou plus généralement dans un vecteur donné) et, pour chacune d'elle, exécute les instructions à répéter. Lorsque les éléments du vecteurs `1:n` sont épuisés, on sort de la boucle et on continue l'exécution du programme.
- Les boucles `while`, à utiliser lorsqu'on ne sait pas à l'avance combien de fois l'action à répéter doit être exécutée : on l'exécute jusqu'à ce qu'une condition de sortie soit satisfaite. La syntaxe est la suivante :

```
while (condition)
  // instructions à répéter
end
```

  où `condition` est un expression vraie ou fausse (un booléen). Lors de l'exécution de la boucle, on exécute les instructions à répéter *tant que* `condition` est vraie (`condition` est donc une condition de continuation, et non de sortie) ; plus précisément, les instructions sont exécutées après avoir vérifié si `condition` est vraie. Bien entendu, `condition` doit faire intervenir des variables qui seront modifiées par les instructions à répéter de façon à ce que `condition` soit fausse après un nombre fini d'exécutions des instructions à répéter.

À toutes fins utiles, on rappelle également la syntaxe de l'alternative :

```
if (condition) then
  // instructions à exécuter si condition est vraie
[else
  // instructions à exécuter si condition est fausse]
end
```

On rappelle les instructions d'entrée/sortie que l'on peut utiliser dans un script :

- `disp(message)` a pour effet d'afficher à l'écran la chaîne de caractères ou la valeur `message` ;
- `var=input(message)` a pour effet d'afficher la chaîne de caractères `message` (à entourer de `'`), d'attendre que l'utilisateur entre au clavier une valeur puis d'affecter cette valeur à la variable `var`.

Exercice 1

1. On considère le code ci-dessous :

Listing 1 : code à analyser

```
n=15;
a=1:n;
b=1 ./a;
disp(sum(b));
```

Que fait ce script ? Expliquer l'effet de chaque instruction. Quel est le rôle du `.` devant le `/` en ligne 3 ?

2. Écrire un autre script effectuant le même calcul grâce à une boucle.

Exercice 2

Soient a et b deux réels strictement positifs.

1. Montrer que les suites $(a_n)_{n\in\mathbb{N}}$ et $(b_n)_{n\in\mathbb{N}}$ définies par les conditions initiales $a_0 = a$ et $b_0 = b$ ainsi que les relations de récurrence :

$$\forall n \in \mathbb{N}, \qquad a_{n+1} = \frac{a_n + b_n}{2}, \qquad b_{n+1} = \sqrt{a_n b_n}$$

convergent vers la même limite. Leur limite commune est appelée moyenne arithmético-géométrique des réels strictement positifs a et b.

2. Écrire un script demandant à l'utilisateur d'entrer les valeurs de a, b et n puis calculant et affichant une valeur approchée à la précision 10^{-n} de la moyenne arithmético-géométrique de a et b.

Une fonction est une nouvelle instruction créée par l'utilisateur. Elle prend des arguments et renvoie un résultat. La syntaxe de base est la suivante :

```
function y=f(x)
  // instructions à exécuter pour calculer le résultat à renvoyer
  y= // résultat à renvoyer
endfunction
```

On définit ainsi une fonction appelée `f` (le nom peut bien sûr être modifié). La variable `y` (qui apparaît deux fois dans le code précédent) est muette, elle ne sert qu'à indiquer à la fin du code de la fonction le résultat à renvoyer. De même que `x` qui désigne un élément générique auquel on peut appliquer la fonction.
Une fois une telle fonction déclarée dans un script (enregistré et exécuté !), l'exécution dans la console de l'instruction `f(1)` aura pour effet de calculer $f(1)$, c'est-à-dire d'exécuter les instructions figurant dans le code de la fonction en remplaçant `x` par la valeur 1.
On peut plus généralement définir des fonctions `[y1, ..., yp] = f(x1, ..., xn)` prenant éventuellement plusieurs arguments et renvoyant éventuellement plusieurs résultats.

Exercice 3

On pose :

$$\forall x \in \mathbb{R}, \quad \forall n \in \mathbb{N}, \quad \mathrm{P}_n(x) = \sum_{k=0}^{n} \frac{x^k}{k!}.$$

1. Écrire une fonction Scilab `STexp(n,x)` renvoyant la valeur de $\mathrm{P}_n(x)$, calculée grâce à une boucle. On pourra remarquer pour accélérer les calculs que :

$$\forall x \in \mathbb{R}, \quad \forall k \in \mathbb{N}, \quad \frac{x^{k+1}}{(k+1)!} = \frac{x}{k+1}\frac{x^k}{k!}.$$

2. Calculer $\mathrm{P}_n(x)$ pour différentes valeurs de n et de x puis comparer à $\mathbf{e}^x$.

3. Écrire une autre fonction réalisant le même calcul qu'en **1.** mais au moyen d'opérations sur des matrices (comme dans le listing 1). On pourra utiliser l'instruction `cumprod` (l'aide de Scilab est votre amie...).

Partie 2 : utilisation des instructions graphiques de scilab

On rappelle qu'étant donnés deux vecteurs `x` et `y` de même taille, l'instruction `plot2d(x,y)` a pour effet de relier les points de coordonnées (x_i, y_i). Si `x` est une subdivision assez fine d'un segment $[a, b]$ et `y` contient les images par une fonction f des éléments de `x`, alors le tracé est la représentation graphique de la fonction f sur l'intervalle $[a, b]$.

Deux commandes sont utiles pour créer une subdivision d'un intervalle $[a, b]$ (faites des tests !) :

- `a:h:b` a pour effet de créer un vecteur contenant les termes inférieurs à `b` d'une progression arithmétique de premier terme `a` et de pas `h`.
- `linspace(a,b,n)` a pour effet de créer un vecteur contenant une subdivision régulière du segment $[a, b]$ formée de n points.

Plus généralement, si `x` est un vecteur de taille n et `Y` une matrice à n lignes et p colonnes `y1`, …, `yp`, l'instruction `plot2d(x,Y)` a pour effet de superposer sur un même graphe les représentations graphiques de `y1`, …, `yp` en fonction de `x`.

On pourra consulter l'aide de Scilab pour obtenir davantage de détails sur l'instruction `plot2d`. La fenêtre graphique peut être nettoyée en utilisant l'instruction `clf`.

Exercice 4

1. Superposer sur une même figure les représentations graphiques des fonctions sin, $x \longmapsto x$, $x \longmapsto x - \frac{x^3}{3!}$ et $x \longmapsto x - \frac{x^3}{3!} + \frac{x^5}{5!}$ sur l'intervalle $[-\pi, \pi]$.

2. Modifier le graphique précédent pour que les axes se croisent à l'origine et les ordonnées varient de -1 à 1.

Exercice 5

On admet que la fonction

$$\mathrm{sh} : x \in \mathbb{R} \longmapsto \frac{\mathbf{e}^x - \mathbf{e}^{-x}}{2}$$

est bijective de $\mathbb{R}$ sur lui-même. Représenter graphiquement la fonction sh ainsi que sa réciproque (que l'on ne cherchera pas à calculer) sur l'intervalle $[-3, 3]$.