Les fonctions

Pour définir une fonction (ou procédures) en Scilab, la méthode la plus courante est de l'écrire dans un fichier, dans lequel on pourra d'ailleurs mettre plusieurs fonctions (en regroupant par exemple les fonctions qui correspondent à un même thème ou une même application). Chaque fonction doit commencer par l'instruction :

function [y1,y2,…,yn]=nomfonction(x1,x2,…,xp)

où les xi sont les arguments d'entrée (ou paramètres), les yj étant les arguments de sortie (ou résultats). Une fonction peut avoir 0, 1 ou plusieurs paramètres (placés entre parenthèses) ainsi que 0, 1 ou plusieurs résultats (placés entre crochets).

Il n'y a pas de mot clé délimitant la fin d'une fonction (comme end ou end function). Dans le cas où le fichier contient plusieurs fonctions, ce rôle est joué par chaque déclaration de fonction via le mot clé function ou par la fin du fichier pour la dernière fonction (dernier saut de ligne).

Remarques :
- La première ligne du fichier de fonctions doit commencer par l'instruction function
- La dernière instruction doit obligatoirement être suivie d'un passage à la ligne sinon l'interpréteur ne la prend pas en compte.
- La tradition est de suffixer les noms des fichiers contenant les fonctions en .sci (en salle informatique, avec le bloc-note, nous le laisserons le suffixer en fichier texte c'est-à-dire en .txt)

Dans la pratique :
- Utilisez un éditeur de texte pour définir votre fichier de fonctions que vous enregistrerez dans votre répertoire de travail (ou un répertoire de fonctions)
- Dans Scilab, vous devez lui indiquer où se trouve les fonctions. Il faut alors charger le fichier avec l'instruction getf en lui indiquant le chemin et/ou le fichier en question.
- Vous pouvez alors utiliser les noms de fonctions comme des mots clés supplémentaires du langage Scilab.

Exemple : Construction du script de la fonction factorielle dans un éditeur de texte que l'on enregistre dans le fichier fact.sci ou fact.txt) :

```
function [y]=factorielle(n)
p=1
for i=1:n do
p=p*i
end
y=p
```

Puis, dans Scilab (si vous avez déclaré le bon répertoire de travail avec Change Directory) :

```
--> getf('fact.sci')
--> disp(factorielle(10))
    3628800
```

Rôles des variables

Dans l'écriture des fonctions, l'argument d'entrée n et l'argument de sortie y sont des variables ou arguments formels. Alors que dans son utilisation resultat = factorielle(10), les arguments utilisés sont appelés arguments effectifs.

Le (ou les) argument effectif d'entrée peut être une constante, une variable, le résultat d'une expression.
Dans une fonction, vous avez accès (en lecture uniquement) à toutes les variables des niveaux supérieurs. On peut alors utiliser les variables globales de programmes (si elles existent). Cela permet de ne pas consommer trop de mémoires (pour la recopie dans une autre variable d'entrée) et de temps (pour le temps de la recopie). Si vous tentez de modifier cette variable globale, une nouvelle variable interne (à la fonction) est créée, la variable de même nom du niveau supérieur n'est alors pas modifiée.

Exemple
```
function [y1]=test(x1)
y1=x1+c  // on utilise la variable globale c (si elle existe …)
disp((c,'c = ')  // disp permet de visualiser des variables dans l'ordre contraire de la déclaration
c=rand()  // une variable interne c est créée
disp(c,'c = ')  // affichage de la variable interne
```

Passage par paramètres

Les premières versions de Scilab donnaient les variables d'entrée par valeur, c'est-à-dire que seules les valeurs des variables étaient affectées aux arguments formels. Une modification de ces arguments formels n'avait aucune incidence sur les variables initiales.
A partir de la version 2.4 de Scilab, les variables d'entrées sont passées par référence (ou adresse). Ce n'est plus la valeur mais la référence de la variable qui est donnée et ceci afin de diminuer la place en mémoire des définitions de variables. Par contre, dès que cette variable est susceptible d'être modifiée, une copie en est donnée et la variable initiale n'est pas modifiée.
Il n'est donc pas possible de modifier une variable d'un programme à partir d'une fonction.

Lors de la sortie de la fonction, toutes les variables internes (donc propres à la fonction) sont détruites.

Instructions spécifiques

Déboguage de fonctions
On peut dans un premier temps utiliser la fonction disp(v1,v2, …) qui permet de visualiser l'évolution de certaines variables de la fonction.
On peut également mettre une ou plusieurs instructions pause en des endroits stratégiques de la fonction. Lorsque Scilab rencontre cette instruction, le déroulement du programme s'arrête et vous pouvez examiner la valeur de toutes les variables déjà définies (l'invite --> se transforme en -1->). Lorsque vos observations sont finies, la commande resume fait repartir le déroulement des instructions (jusqu'à l'éventuelle prochaine pause).

Interruption d'une procédure
L'instruction return permet d'interrompre le déroulement d'une fonction et de revenir au programme ayant fait appel à cette fonction. Si la fonction a été déclarée comme fournissant un résultat, la (ou les variables) constituant ce résultat doivent avoir reçu préalablement une valeur.

Exemple : La fonction suivante fournit l'inverse de son paramètre, après avoir testé que celui-ci n'est pas nul ; dans ce dernier cas, elle rend la valeur infini :

```
function [z]=inv(x)
// division
```

```
if x = = 0 then z=%inf; return
end
z = 1/x
```

Exemple d'une fonction renvoyant deux arguments : Résolution d'une équation du second degré avec $\Delta > 0$

```
Function [x1,x2] = resol(a,b,c)
if (a==0) then error("on ne traite pas le cas a=0!")
else
   delta   = b^2-4*a*c
   if (delta <0) then error("on ne traite que le cas où delta > 0")
   else if (b<0) then x1=(-b-sqrt(delta))/(2*a);x2=(-b+sqrt(delta))/(2*a);
                      end
  end
end
```

Attention, il y a deux valeurs, si on exécute dans Scilab resol(1.e-08,0.8,1.e-08)

```
ans =
         -1.250e-08
```

Le résultat est mis dans ans mais ans est seule pour récupérer les deux résultat, par défaut, elle prend le premier des deux résultat. Pour récupérer les deux résultats, il faut utiliser la syntaxe :

```
--> [x1,x2]=resol(1.e-08,0.8,1.e-08)
x2 =
  - 800000000.
x1 =
  - 1.250e-08
```

Exercices sur les fonctions

Calculs approchés de solutions d'équations

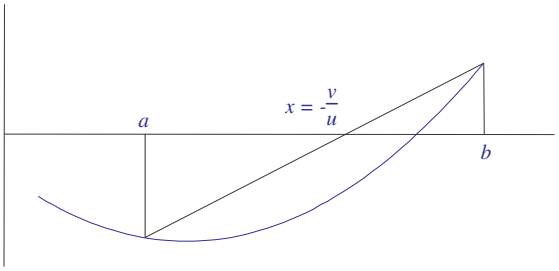
Sur un intervalle $[a;b]$ donné, on recherche la solution d'une équation, ce qui revient à rechercher le zéro d'une fonction c'est-à-dire la valeur x_0 telle que $f(x_0) = 0$.

Méthode par interpolation linéaire :

Sur l'intervalle $[a;b]$, on interpole la fonction par une fonction affine soit du type $g(x) = ux + v$. On détermine ensuite la solution $x = -\frac{v}{u}$.

Si $f(a)$ et $f(x)$ sont de même signe alors a prend la valeur de x sinon b prend la valeur de x.

On itère ce processus jusqu'à une précision désirée sur la valeur approchée de la solution.



Programmation :

On dispose de la fonction suivante permettant d'obtenir les images d'un réel x par une fonction :

```
Function [y]=f(x)
    y=exp(x/2)-4*x;
```

1) Ecrire la fonction **[u,v]=Recherche_Equation(x1,x2)** permettant de déterminer les coefficients u et v de la droite passant par les points $(x_1,f(x_1))$ et $(x_2,f(x_2))$.

2) Ecrire la fonction **[a,b]=Une_Etape(x,y)** permettant d'effectuer une étape dans la résolution par interpolation linéaire.

3) Ecrire le programme principal demandant à l'utilisateur les bornes de l'intervalle de départ puis de répéter les étapes de la résolution et obtenir une valeur approchée x_0 jusqu'à ce que, par exemple, $f(x_0)$ soit inférieur à une précision donnée.

La devinette

On veut écrire un programme obéissant aux règles suivantes :

- l'ordinateur génère un entier aléatoire compris entre 0 et 999, demande ensuite à l'utilisateur de deviner ce nombre.
- Pour chaque entier proposé par l'utilisateur :
  - l'ordinateur précise en cas d'échec s'il est plus petit ou plus grand que le nombre mystérieux, puis demande une nouvelle proposition à l'utilisateur.
  - en cas de succès, il indique le nombre de propositions qui ont été nécessaire pour deviner le nombre mystérieux puis le programme s'arrête.

Sachant que Scilab possède la fonction prédéfinie rand(), donnant un nombre aléatoire entre 0 et 1, ainsi que la partie entière d'un réel positif int(x), écrire un programme en Scilab réalisant les objectifs ci-dessus.

La série harmonique

On démontre que la *série* de somme partielle $S_n = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$ tend vers $+\infty$ lorsque n tend vers $+\infty$.

Le problème consiste à trouver le nombre de termes de cette *série*, dite harmonique, nécessaire au dépassement d'une valeur arbitraire donnée :

$$1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n} > \text{valeur}$$

1) Ecrire une fonction function [s]=harmonic(n) qui, pour un entier n passé en paramètre, calcule la somme

$s = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$.

2) Ecrire un programme dans Scilab

- demandant à l'utilisateur une valeur strictement plus grande que 1 et plus petite ou égale à 8 (et répétant la saisie jusqu'à ce que la condition soit satisfaite)
- affichant le nombre de termes nécessaire au dépassement de cette valeur et le résultat de la somme (vous pourrez utiliser la fonction harmonic définie ci-dessus).

Autour des fractions

Partie A Calculs sur les fractions

Vous allez dans un premier temps créer un ensemble de procédure et fonctions permettant de représenter les fractions et de travailler avec.
Les fractions seront représentées sous la forme d'un tableau de deux éléments, le premier contiendra le numérateur et le second le dénominateur. Exemple, le tableau [2,3] représente la fraction $\frac{2}{3}$.

1) Ecrire la fonction **[t]=lire** permettant la donnée d'une fraction (représentée par le tableau *t*) par un utilisateur au clavier. Vous devrez vous assurez que le dénominateur est non nul.
2) Ecrire la fonction **ecrire(f)** permettant de visualiser la fraction sous la forme p/q ou simplement p si q = 1. Pour l'affichage, vous pourrez utiliser l'instruction disp(string(f(1))+' / '+string(f(2))) dans le premier cas et adapter les autres affichages à partir de cette proposition.
Si q < 0, vous préférerez l'affichage de la forme -p/(-q) comme sur l'exemple suivant
Si f(1) = 8 et f(2) = -3, l'affichage sera -8/3

3) Ecrire la fonction **[c]=pgcd(a,b)** permettant de déterminer le pgcd (plus grand commun diviseur) des deux nombres a et b.
Vous pourrez utiliser par la méthode suivante ; si ce n'est pas le cas, vous expliquerez votre démarche par un texte clair.
Si un des deux nombres est nul, l'autre est le pgcd sinon il faut remplacer le plus grand par la différence entre le plus grand et le plus petit et laisser le plus petit inchangé.
Puis recommencer ainsi avec la nouvelle paire jusqu'à ce qu'un des deux nombres soit nul. Dans ce cas, l'autre nombre est le pgcd.
4) Ecrire la fonction **[t]=reduire(f)** permettant de simplifier la fraction f grâce à la fonction précédente.
5) Ecrire la fonction **[f3]=addition(f1,f2)** permettant d'obtenir la fraction f3 comme somme des fractions f1 et f2 (avec cette somme simplifiée).
6) Ecrire la procédure **[f3]=soustraction(f1,f2)** permettant d'effectuer, de la même manière la soustraction de f1 par f2.
7) Ecrire la procédure **[f3]=multiplication(f1,f2)** afin d'obtenir la fraction simplifiée f3 comme produit de f1 par f2.
8) Terminer la construction des outils de calculs sur les fractions par la procédure **[f3]=division(f1,f2)** permettant d'obtenir la division de f1 par f2.

Partie B : Fractions continues

Connaissant la fraction $\frac{a}{b}$, la division permet d'obtenir l'écriture décimale. Réciproquement, peut-on retrouver l'écriture fractionnaire ou la meilleure approximation fractionnaire d'un nombre exprimé en écriture décimale ?
Examinons le cas d'un réel positif.
Si ce réel contient "peu" de chiffres après la virgule, une méthode consiste à exprimer ce réel comme la division d'un entier par une puissance de dix (définition d'un nombre décimal) puis de simplifier cette fraction.
Dans le cas où ce réel est obtenu suite à un calcul et s'il contient "beaucoup" de chiffres après la virgule, une autre méthode consiste à déterminer la fraction continue associée à ce nombre puis de simplifier cette fraction continue.

Exemple : $\pi = 3{,}14159265359\ldots = \mathbf{3} + 0{,}14159265359\ldots = \mathbf{3} + \dfrac{1}{7{,}062513306\ldots} = \mathbf{3} + \dfrac{1}{\mathbf{7} + 0{,}062513306\ldots}$

$$= \mathbf{3} + \frac{1}{\mathbf{7} + \dfrac{1}{15{,}9965944068\ldots}} = \mathbf{3} + \frac{1}{\mathbf{7} + \dfrac{1}{15{,}9965944068\ldots}} = \mathbf{3} + \frac{1}{\mathbf{7} + \dfrac{1}{\mathbf{15} + 0{,}9965944068\ldots}}$$

Les suite des nombres entiers [3 ; 7 , 15 , …] est la fraction continue associée au réel initial, dont des approximations de plus en plus fines sont les fractions successives :

$$\mathbf{3} \text{ puis } 3+\frac{1}{7}=\mathbf{\frac{22}{7}} \text{ puis } 3+\frac{1}{7+\frac{1}{15}}=\mathbf{\frac{333}{106}} \text{ puis …}$$

Nous nous contenterons, dans un premier temps, de la détermination des cinq premiers éléments des fractions continues qui seront donc représentées par un tableau de taille 5.

1) Ecrire la fonction **[f]=frac_cont(x)** permettant d'obtenir la fraction continue f du réel x.

2) Ecrire la fonction **reconstitution(f)** permettant la détermination puis l'affichage de la fraction simplifiée de l'approximation obtenue par la fraction continue représentée par le tableau f.

Exemple, si vous aviez la fraction continue [3 ; 7 , 15 , …] (mais elle est ici de taille 3), vous devrez déterminer puis afficher 333/106.

Partie C : Promenades aléatoires

Si on considère la suite des puissances de $\frac{3}{2}$ et si on ne regarde que les chiffres après la virgule, on tombe sur un problème que les mathématiciens ne savent pas encore résoudre : cette suite de nombres est-elle aléatoire ?

Afin d'illustrer cette question, vous devrez utiliser une méthode graphique :

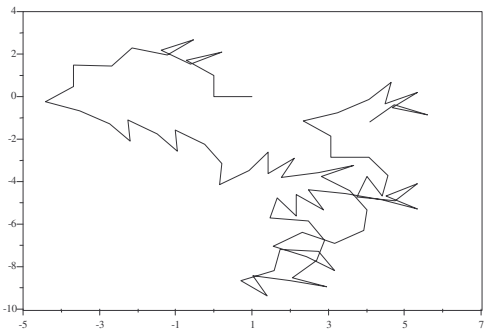
Comment construire une promenade associée à une suite modulo 1 ?

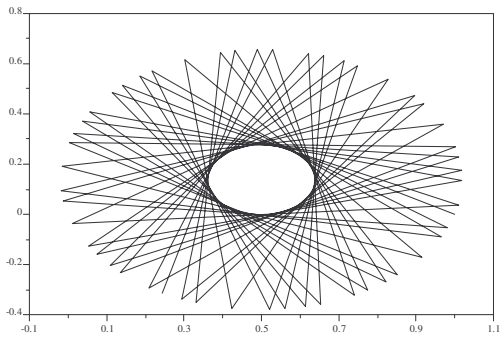
- Le point de départ est le centre d'un cercle de rayon 1.
- A la première étape, on place le premier terme de la suite, suivant sa valeur, sur le cercle et l'on joint les deux points. L'angle permettant de repérer ce point étant proportionnel à sa valeur, modulo 1.

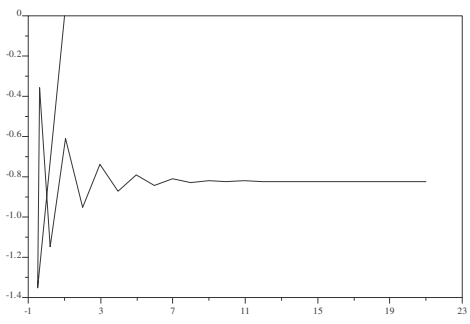
Exemple : si la première valeur est 3,258 968 … soit 0,258 968 … modulo 1, l'angle permettant de repérer ce point sur le cercle est 0,258 968 … $\times 2\pi$ rad

- Le dernier point dessiné devient le centre du nouveau cercle qui va servir à la deuxième étape et ainsi de suite…

Les schémas suivants présentent respectivement une promenade aléatoire des puissances de $\frac{3}{2}$, une autre sur les multiples de $\sqrt{2}$ et enfin des puissances de $\varphi=\frac{1+\sqrt{5}}{2}$, le "nombre d'or".







Exercices sur les fonctions

Exercice 1

On sait que si f est une fonction continue strictement monotone sur un intervalle $[a;b]$ et que $f(a)$ et $f(b)$ sont de signes contraires, l'équation $f(x) = 0$ admet une racine unique α appartenant à $]a;b[$.

Par **dichotomie**, on peut obtenir une suite de nombres convergeant vers α. L'algorithme en est le suivant :

- on connaît au départ : - la fonction f (supposée continue strictement monotone),
  - les réels a et b tels que $f(a)\times f(b)$ soit négatif,
  - une constante epsilon indiquant la précision attendue.
- la variable auxiliaire c désignera la demi-somme de a et b.
- algorithme :

```
début
    tantque (a+b)/2>epsilon
      debut
        c <-- (a+b)/2;  //afficher(c) ici si on désire toutes les approximations
        si  f(a)*f(c)<0 alors  b <-- c sinon  a <-- c;
      fin
    afficher((a+b)/2);  //ou retourner((a+b)/2) s'il s'agit d'une fonction
fin
```

1) Ecrire une fonction [y]=f(x) permettant la déclaration de la fonction f définie par cos(x) - x
2) Ecrire une *fonction* **DICHO** réalisant l'algorithme précédent, a et b étant reçus en paramètres entrants et c comme paramètre de sortie.

Ecrire la ou les instructions sous Scilab permettant d'obtenir une solution de $\cos(x) = x$ à 10^{-10} près.

Exercice 2

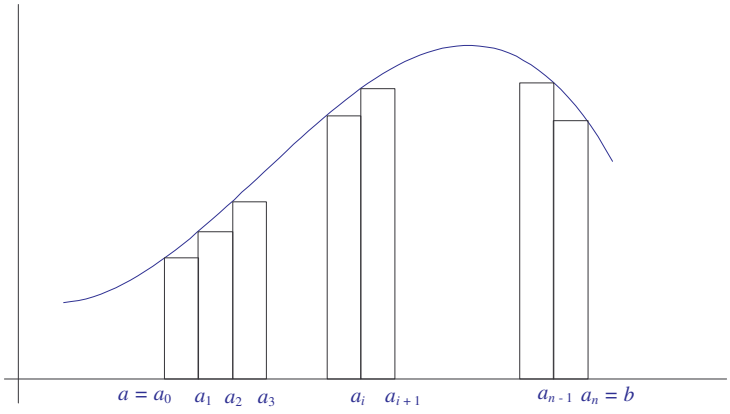
Calculs approchés d'intégrales

Si f est continue (ou continue par morceaux) sur $[a;b]$, alors f admet des primitives sur $[a;b]$ (ou est intégrable sur $[a;b]$) mais il se peut qu'on ne sache pas calculer $\int_a^b f(t)dt$.

Méthode : On subdivise $[a;b]$ en n segments de longueur $\frac{b-a}{n}$: $a_0 = a$; $a_1 = a_0 + \frac{b-a}{n}$; $a_2 = a_1 + \frac{b-a}{n}$; … ; $a_i = a_{i-1} + \frac{b-a}{n}$; … ; $a_n = b$.

Dans ce cas, $I = \sum_{i=0}^{n-1} \int_{a_i}^{a_{i+1}} f(t)dt$

On remplace $\int_{a_i}^{a_{i+1}} f(t)dt$ par une valeur approchée obtenue en prenant pour f une fonction approximant f et dont on sait calculer l'intégrale sur $[a_i;a_{i+1}]$



Méthode des rectangles : On remplace $f(t)$ sur $[a_i;a_{i+1}]$ par $f(a_i)$

On obtient alors comme valeur approchée de l'intégrale $S_n = \frac{b - a}{n} \sum_{i=0}^{n-1} f(a_i)$

Programmation :
On dispose de la fonction suivante permettant d'obtenir les images d'un réel *x* par une fonction
function [y]=f(x)
```
y = 1/(1+x*x);
```

Ecrire la fonction **[s]=somme(a,b)** permettant de calculer la somme des aires des rectangles après avoir demandé à l'utilisateur le nombre d'intervalles de la subdivision.

Exercice 3

1) Ecrire une *fonction* **PPDP** renvoyant le **P**lus **P**etit **D**iviseur **P**remier d'un entier au moins égal à 2 reçu en paramètre.
2) Ecrire une *fonction* **DECOMPOSER** affichant à l'écran la décomposition en facteurs premiers d'un entier au moins égal à 2 reçu en paramètre. Cette procédure devra appeler la fonction précédente. L'affichage devra ressembler à ceci :

360 = 2 * 2 * 2 * 3 * 3 * 5 ou à : 360 = 2^3 * 3^2 * 5.

3) Ecrire une *fonction* **SAISIR** demandant à l'utilisateur un entier au moins égal à 2 (et répétant la saisie jusqu'à ce que cette condition soit satisfaite), puis renvoyant cet entier dans un paramètre (passé bien sûr par adresse).
4) Vous pourrez réaliser le programme suivant sous Scilab (après avoir chargé le fichier de fonctions) :

```
a = SAISIR()
DECOMPOSER(a)
```