Scilab II

1 Rappels

Attention : Les listes ci-dessous vous permettent de retrouver rapidement des fonctions utiles de Scilab. Il faut bien sûr consulter l'aide de Scilabpour comprendre exactement comment les utiliser.

Opérateurs de base

`[instruction], [instruction];` — affiche (,) ou n'affiche pas (;) le résultat de l'instruction

`//` — permet d'ajouter un commentaire (non lu par Scilab)

`...` — permet de passer à la ligne sans compilation (très utile pour les lignes de code longues dans un programme

`clear() clf()` — réinitialise toutes les variables, efface les graphiques précédents

`+ - * ^ / \` — opérations courantes

`.+ .- .* .^ ./ .\` — les mêmes pérations terme à terme

`sqrt %i, %pi, %e` — racine carrée, constantes usuelles

Opérateurs logiques

`%T \%F` — true, false

`& | ~` — et, ou, non

`== = <>` — test d'égalite, affectation, différent

`<= >= < >` — inférieur ou égal, etc.

`pause()` — utilisée dans le code d'une fonction, l'instruction stope l'execution de celle-ci ; l'utilisateur doit indiquer ce qu'elle/il veut faire (`abort, resume ...`)

`xpause(t)` — interruption du programme pendant un temps t (microsecondes)

`n=input('message')` — demande la valeur de n à l'utilisateur via la question oju l'instruction `message` (par exemple `'indiquer le nombre de joueurs'`)

`[u,v]=find(M==x)` — recherche les positions (u, v) de M telles que $M(u, v) == x$

`for (condition) do (instruction) end` — la boucle courante

`while (condition) do (instruction) end` — la boucle conditionée

`if..then..else..end` — évident

`isequal(x,y,...)` — compare les arguments. Si ils sont tous identiques alors la fonction retourne %t ; sinon elle retourne %f.

Type de variables

`type(x)` — retourne le type de la variable `x`

La réponse est un entier : par exemple, on a 1 si x est une matrice réelle ou complexe, 2 si x est une matrice polynômiale, 4 si x est une matrice booléenne, 10 matrice de chaines de caractères, 14 si x est une liste, ...

`liste()` — Crée une liste vide. Les éléments de la liste peuvent être de types différents. Par exemple `E=liste('a',[1 2])` crée une liste de deux éléments, un caractère et un vecteur ligne.

Fonctions usuelles

`abs sign min max` — module, signe, min, max

`floor(z) ceil(z) round(z) int(z)` — partie entière, `-floor(-z)`, arrondi, troncature

`rand()` — nombre aléatoire entre 0 et 1 ; appelée avec des arguments, par exemple `rand(2,3)`, l'instruction retourne une matrice 2×3 de nombres aléatoires

`sqrt real imag conj` — racine carrée, parties réelle et imaginaire, conjugué

`cos sin tan cotg` — pas de commentaire

`cosh sinh tanh` — pas de commentaire

`acos asin atan` — pas de commentaire

`exp log log10` — pas de commentaire

`deff('y=f(x)','y=1/x')` — syntaxe de définition de la fonction $f : x \to 1/x$

`feval(M,f)` — évaluation de la fonction f sur les termes de la matrice M

`integrate('f(x)','x',a,b) intg` — intégrale de f sur l'intervalle $[a, b]$

`fsolve(init, f)` — résoud numériquement $f = 0$ avec l'initialisation de l'algorithme en $x_0 =$ `init`

`ode(y0,t0,T,f)` — résolution numérique des équations différentielles

Calcul matriciel

`a:delta:b` — vecteur ligne d'extrémités a et b et de pas δ

`linspace(a,b,n)` — n nombres régulièrement espacés entre a et b

`[1,2,3;4,5,6]` — une matrice de type 2×3

`size(A) size(A,1)` — donne le type de la matrice A, respectivement le nombre de lignes de A

`A(i,j) A(i,j)=0` — élément de A en position (i, j) ; respectivement remplace cet élément par 0

`[A,B] [A;B]` — concaténation (horizontale, verticale) si les types sont compatibles

`zeros(A) zeros(i,j) ones(A) ones(i,j)` — matrice nulle de même type que A, de type $i \times j$, etc

`eye(i,j) eye(A)` — matrice unité (ou presque) de type $i \times j$ (ou du type de A)

`rand(i,j) rand(A)` — matrice aléatoire de type $i \times j$ (ou du type de A)

`diag([5,-2,3]) diag(A,d)` — matrice diagonale de diagonale le vecteur $(5, -2, 3)$ (on peut aussi utiliser un argument optionnel d pour la d-ième diagonale), d-ème diagonale de la matrice A

`A(3:5,2:3) A(3:5,:)=[]` — matrice extraite et respectivement suppression des lignes $3, 4, 5$

`A+B  A-B  A*B  A^2  A^(-1)  A/B  A\B` opérations sur les matrices

`A.*B  A.^2  A.^(-1)  A./B  A.\B` opérations terme à terme

`inv(A) sum(A) prod(A)` — inverse de la matrice carrée A, somme et produit de ses coefficients

`norm(A) norm(A,1)` — norme L^2 et norme L^1 de A

`A'` — adjointe — conjuguée de la tansposée de A

`spec(A) [vect,val]=spec(A)` — valeurs propres de A, vecteurs propres et valeurs propres associées

`trace(A) det(A)` — trace et déterminant de A

`find(M==3)` — recherche le nombre 3 dans la matrice M et retourne une liste des positions lues collonne par collonne

`vectorfind(M,v,'r')` — recherche la ligne v dans la matrice M

`[E,k]=unique(M,'c')` — élimine les doublons de colonnes dans la matrice M

Polynômes

`poly([r1,r2,r3,'t')` — polynôme unitaire de racines r_1, r_2, r_3

`poly([c0,c1,c2,'t','c')` — polynôme $c_0 + c_1 t + c_2 t^2$

`%s=poly(0,'s')` — définit `%s` comme le polynôme formel homogène de degré 1 — on peut ne pas utiliser le symbole `%`

`coeff(p)` — donne les coefficients (puissances croissantes)

`degree` — donne le degré d'un polynôme

`roots(p)` — donne les racines

`p+q, p*q, p^2, p/q` opérations sur des polynômes en une même variable

`derivat(p)` — dérive une fraction rationelle (Scilab peut travailler avec une seule variable formelle ; pour cette raison la variable n'apparait pas dans `derivat`.)

`poly(A,'x')` — polynôme caratéristique d'une matrice carrée A

2 Quelques remarques

Dérivation

Comme Scilab est un logiciel de calcul numérique il ne possède pas un opérateur de dérivation calculant la fonction dérivée d'une fonction donnée. Il y a toutefois une exception pour les polynômes et les fractions rationnelles (quotient de deux polynômes) : la fonction `derivat` fournit la dérivée *formelle* de toute fraction rationnelle. Noter que cet opérateur calcule la dérivée formelle et il ne s'applique pas aux (autres) fonctions, même "usuelles" $(\sin, \log \dots)$ ou polynomiales.

Exemple 2.1. Essayer la suite d'instructions ci-dessous.

```
X=poly(0,'X')
P=X^2-3*X+1
F=P/(X-1)
derivat(F)
```

Dans le cas général, pour une fonction f donnée, on peut définir sa dérivée numérique en x_0 par

```
function r=numDiff(f,x0,h)
  r = (f(x0+h)-f(x0-h))/(2*h)
endfunction
```

La précision est bien sûr obtenue en choisissant des petites valeurs pour h.

Exercice 1. Soit $f(x) = x^3 \ln(x+1)$. Calculer la dérivée de f. Etudier la différence entre $f'(a)$ et `numDiff`(f, a, h) en fonction de h pour différent a fixés. (Dans la commande `plot2d` on peut utiliser le paramètre `'ll'` qui choisit l'echelle logarithmique le long des deux axes.)

Intégration

Scilab possède une "boite noire" nommée `integrate` qui, malheureusement, présente quelques bugs. Essayer

```
integrate('cos(x)','x',-1,1)  // ok
integrate('sin(x)','x',-1,1)  // bug
integrate('sin(x)','x',-1,2)  // ok
```

(Il faut tester les instructions avant de les utiliser.) En fonction du problème, on peut appliquer des techniques d'approximation classiques, comme les sommes de Riemann ou la méthode des trapèzes.

Exercice 2. Ecrire une fonction `integrale` qui prend comme arguments une fonction f et un nombre entier n et qui rend la somme de Riemann associée à f sur $[0,1]$, $\sum_{j=1}^{n} f(j/n)/n$ de deux façons différentes : en construisant un vecteur de taille $n+1$ contenant la division équidistante de $[0,1]$; en utilisant une variable locale I et en effectuant une boucle qui, à la j-ième itération ajoute $f(j)/n$ à I. Comparer ces deux fonctions du point de vue de la rapidité.

3 Instructions graphiques

La commande de principale de Scilab pour dessiner des figures planes est l'instruction `plot2d(x,y)`. Cette commande permet de tracer une ligne brisée en reliant par des segments les points de coordonnées $(x(i), y(i))$ et $(x(i+1), y(i+1))$. Plusieurs tracés successifs sont affichés sur la même fenêtre graphique jusqu'à son effacement via `clf()` ou la destruction de la fenêtre.

Exemple 3.1. Sur la figure construite par les commandes suivantes, le cercle apparaît ovale car on a laissé Scilab décider de l'échelle en fonctions des données envoyées à `plot2d` et l'échelle choisie n'est donc pas forcément isométrique.

```
// cercle
t = [0:0.02:2*%pi]'
x = cos(t); y = sin(t)
plot2d(x,y)
// tangente y=-x+sqrt(2)
A = [0;sqrt(2)]; B = [sqrt(2);0]
plot2d(A,B)
```

On peut spécifier le comportement de `plot2d` en utilisant des paramètres optionnels. Ces paramètres sont tous de la forme `clef=`*valeur* ; les plus utiles sont les suivants :

- `axesflag` pour gérer le positionnement des axes (par exemple, `1` crée un rectagle avec les axes en bas et à gauche — position par défault —, `2` crée un rectagle sans axes, `3` les axes en bas et à droite, `4` les axes centrés au millieu du rectangle donné par `rect`, `axesflag=5` comme `4` mais avec le rectangle visible....
- `frameflag` pour gérer la taille de la fenêtre et les échelle ; `frameflag=3` ou `4` pour une échelle isométrique, `5` ou `6` pour avoir une échelle avec des graduations "simples", `frameflag=0` pour superposer un graphe à la figure précédente en conservant l'échelle.
- `rect=[xmin,ymin,xmax,ymax]` permet de délimiter la fenêtre graphique par une boite.
- `style` permet de définir le style de chaque courbe, une couleur pour les valeurs positives ou une marque pour les valeurs négatives. Pour les couleurs on peut utiliser la table des couleurs par défaut de Scilab, que l'on peut visualiser avec `getcolor()`: `1` pour black, `2` pour blue, `3` pour green, `4` pour cyan, `5` pour red, `6` pour magenta, `7` pour yellow, `8` pour white, `9` pour violet...

Autres commandes

- `winsid` permet de lister les fenêtres graphiques existantes
- `scf` permet de définir une fenêtre comme fenêtre graphique courante via, par exemple, son numéro (récupéré par winsid)