Université de Strasbourg
Techniques d'analyse numérique L3S6
Année 2016-2017

TP 1 : Introduction à Scilab

L'objectif de ce TP est de se familiariser avec le logiciel Scilab (acronyme pour Scientific Laboratory).

1 Demarrage

1. Lancer scilab; une console apparaît
2. Ouvrir un fichier (cliquer sur une des deux icônes en haut à gauche)
3. Ecrire une commande ou rien du tout
4. Enregistrer le fichier ("TP1.sce")
5. Executer le fichier (bouton play)

Pour le point 2., on utilise ici l'éditeur intégré de Scilab. Il est possible d'uitliser un autre éditeur (emacs, TextWrangler sur mac...). L'execution se fait alors sur la console Scilab.

Par la suite, il faudra remplir le fichier, le sauvegarder et l'executer régulièrement (pour éviter de mauvaises surprises). Des petits tests (qui ne seront pas sauvegardés par contre) peuvent être faits directement sur la console et on peut revenir sur les instructions précédentes avec la flèche ↑. Le raccourci Ctrl+A permet d'aller au début de la ligne et Ctrl+E à la fin de la ligne. On pensera aussi à utiliser Ctrl+C/Ctrl+V pour d0tytopd'cilab.]T -13.333 -37.329 1d [(3.)-500(Ec [(")3T /F16 10.90

A noter que les variables courantes sont accessibles avec la commande `who`.

Dans un code, il est important de mettre des commentaires (`//`), pour donner des explications. Sur une ligne, tout ce qui suit le symbole `//` est ignoré.

On peut écrire plusieurs instructions sur une ligne en les séparant par une virgule (pas très conseillé pour la lisibilité).

La variable `ans` contient le résultat de la dernière opération effectuée.

2 Premiers calculs

Scilab manipule des nombres réels en double précision. Sur des données de l'ordre de l'unité, cela se traduit pratiquement par le fait que l'erreur de calcul est de l'ordre de 1e-15 (notation scientifique: $= 10^{-15}$, parfois noté 1E-15, ou 1D-15). L'affichage ne donne en général que 7 chiffres significatifs (alors que la précision en calcul donne 15). Pour avoir un affichage d'un nombre réel avec plus de chiffres, on peut utiliser la commande issue du C `printf("%1.200g\n",nom_de_la_variable)`. Attention, tous les chiffres ne sont pas significatifs. Il s'agit normalement du nombre binaire stocké écrit en écriture décimale.

1. Définir une variable valant 1+1e-15.
2. Enlever 1 à cette variable et afficher le résultat
3. Recommencer avec 1e-16 (ou 1e-14) à la place de 1e-15.
4. Vérifier ce phénomène en utilisant uniquement des nombres entiers.

Ainsi, si on fait une erreur de l'ordre de 1e-15 pour une quantité de l'ordre de l'unité, on peut raisonnablement l'imputer à des erreurs d'arrondi. Notons aussi, qu'on essaye dans la mesure du possible de se ramener à faire des calculs sur des données de l'ordre de l'unité quitte à faire des renormalisations/adimensionnements.

3 Calculatrice

Scilab peut faire office de calculatrice. On retrouve les principales fonctionnalités: opérations basiques, fonctions usuelles,variables prédéfinies telles que π ou e, nombres complexes. On peut retrouver dans l'aide avec la commande `help nom_fonction`, ou `apropos mot_cle`.

1. Chercher de l'aide sur la fonction sinus
2. Retrouver les syntaxes des fonctions usuelles

3. Chercher de l'aide sur les variables prédéfinies (utiliser `help %`, puis `help percent`).

4. Stocker la partie réelle (resp. la partie imaginaire) de $A = \frac{1+2i}{1+i}$ dans la variable reA (resp. la variable imA). Calculer le module et l'argument de A. On pourra utiliser la commande `atan(imag(z),real(z))`.

4 Les vecteurs

Une notion importante à comprendre dans scilab est la notion de vecteur qui généralise celle de scalaire (nombre).

En réalité, toute donnée de base est une matrice. Ainsi, un nombre est traité comme une matrice 1×1 et un vecteur ligne (resp. un vecteur colonne) comme une matrice $n \times 1$ (resp. $1 \times n$).

Commençons par quelques exemples. Effectuer les calculs suivants dans la console et interpréter.

1. `A=[1,2,3,4,5],B=[0,2,4,6,8,10],D=[]`
2. `A=1:5,B=0:2:10,D=[2:1]`
3. `C=0:1/3:1`
4. `C=0:1/3+2*%eps:1`
5. `C=linspace(0,1,4)`

On évitera en particulier de faire des itérations sur des quantités non entières (cas 3. et 4.). On préfèrera la solution avec linspace. Notons aussi que `a:n:b` fonctionne dans le cas $a > b \in \mathbb{Z}$, and $n < 0 \in \mathbb{Z}$.

On aura souvent besoin de faire une discrétisation uniforme d'un intervalle $[x_{\min}, x_{\max}]$ en N sous intervalles.

1. Fixer des paramètres $N, x_{\min}, x_{\max}$ (au choix!)
2. Définir un vecteur x correspondant à la discrétisation uniforme en utilisant `linspace`
3. Définir le même vecteur, en utilisant `:`, mais sans incrément non entier.
4. Que vaut `x(1)`, `x(4)`? Que peut-on dire de `x(0)`?
5. Signification de `x($),length(x),x(2:3),x(:),x([1,1])`
6. Concaténation: `y=[0 x 0]`
7. Affectation multiple: `[a,b,c]=(1:2,2:3,3:5)` (fonctionne aussi pour les scalaires)

8. Donner le vecteur x lu à l'envers.
9. Ajout d'élément et suppression: `x(5)=2`, `x(2)=[]`

Pour le point 3., on utilisera le fait que les opérations principales (additions, multiplications...) définies a priori pour les scalaires se généralisent au cas vectoriel. Ceci est une propriété fondamentale. On peut souvent écrire des morceaux de programmes en utilisant ce principe (programmation vectorielle). Cela donne généralement une forme plus compacte et permet d'executer les calculs plus rapidement (scilab est un langage interprété).

Pour le point 4., on remarquera que la convention pour les tableaux (vecteurs) est de commencer à 1 (comme en fortran, maple) et non par 0 (comme en C, python).

Pour le point 10., les performances peuvent être fortement affectées, comme l'allocation se fait ici dynamiquement; il est donc déconseillé d'utiliser cette procédure de manière abusive.

5 Les matrices

Les matrices généralisent la notion de vecteurs. On va commencer à nouveau par quelques exemples

1. `A=[1 2 3;4 5 6;7 8 9]`
2. Comment accède-t-on à l'élément $a_{1,2}$?
3. `help eye` (matrice identité)
4. `help zeros` (matrice nulle)
5. `help ones` (matrice avec que des 1)
6. `help diag` (matrice diagonale; sur ou sous diagonale)
7. `help matrix` (reformatage)
8. Définir la matrice par blocs de taille $2N \times 2N$ suivante
$$\begin{pmatrix} I_N & -I_N \\ -I_N & I_N \end{pmatrix}.$$
9. Générer un vecteur et une matrice aléatoires à l'aide de la fonction `rand`
10. Suppression: `A=matrix(1:9,3,3); A(:,[2 3])=[]`
11. Exemple de concaténation `A=[1:3; eye(2,2) [4;5] ]`
12. Extraction: `diag,tril,triu`
13. `sum,cumsum,prod,cumprod,mean,max,min,size`

On retrouve aussi la plupart des outils de calcul matriciels

1. `det()` Calcul du déterminant
2. `trace()` Calcul de la trace (somme des éléments diagonaux)
3. `inv()` Calcul de l'inverse
4. `spec()` Calcul des vecteurs et valeurs propres
5. `rank()` Calcul du rang d'une matrice
6. `qr()` Décomposition QR d'une matrice
7. `lu()` Décomposition LU d'une matrice
8. `chol()` Décomposition de Cholesky d'une matrice

Il est possible de considérer une matrice comme un vecteur, ce vecteur est composé des colonnes de la matrice mises bout à bout.

On peut aussi utiliser les matrices avec les booléens & (et), | (ou), (non) et les opérateurs de comparaison.

1. `M=(1:5) >3`
2. `and(M,'r')`
3. `A=rand(4,4);A(A>0.5)`
4. `find(A>0.5)`
5. `[i,j]=find(A>0.5)`
6. Générer 2 matrices aléatoires et trouver la matrice booléenne correspondant aux éléments de A supérieurs à ceux de B
7. Générer une matrice aléatoire de taille 5×5 et enlever 1 aux éléments > 0.5.
8. Générer une matrice aléatoire de taille 5×5 de nombres entre 0 et 10 et trouver les éléments égaux à zéro.

6 Multiplication de matrices et resolution de systemes lineaires

Le produit `A*B` est le produit matriciel (attention à la compatibilité des tailles)

Le produit `A.*B` est le produit élément entre deux matrices de même taille.

Pour faire le produit $A^{-1}B$, il faut écrire `A\B`.

Pour faire le produit BA^{-1}, il faut écrire `B/A`.

Pour retenir : dans les deux cas précédents, A est en dessous de B (pour B diviser par A); par contre comme le produit n'est pas commutatif, il faut écrire dans le bon ordre.

De la même manière pour résoudre le système linéaire Ax = b, il faut écrire `x = A\b`.

7 Representation graphique

Sur un exemple. Le plus simple est de tracer le graphe d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$ à l'aide de `plot2d`. On crée pour cela un vecteur `x` d'abscisses, par exemple :

```
--> x = linspace(0,2* \% pi,101);
```

puis on prend l'image de ce vecteur par la fonction pour créer un vecteur `y` d'ordonnées :

```
--> y = sin(x);
```

La commande `plot2d(x,y)` représente les points de coordonnées $(x(i), y(i))$ en les joignant par des droites :

```
--> plot2d(x,y)

--> plot2d2(x,y,style = 2)
```

Par défaut, Scilab superpose les figures. Pour effacer la figure précédente, on utilise la commande `clf` :

```
--> clf

--> plot2d2(x,y,style = 2)
```

On peut aussi ouvrir une autre fenêtre grâce à la commande `figure` ou `xset('window',nb)`:, avec `nb` le numéro de la fenêtre que l'on veut ouvrir.

```
--> figure;

--> plot2d(x,y,style = -1)

--> xtitle('fonction sinus',"x",'sin(x)')
```

Noter que les chaines de caractères sont notés entre apostrophes ou guillemets.

Exercice : Dessiner le cercle unité.

8 Programmation

On va développer ici des morceaux de codes plus long; il est donc plus judicieux ici d'utiliser un fichier .sce pour avoir une trace de ce qui est fait.

A noter que lorsqu'un programme ne s'arrête pas (boucle infinie, calcul trop long...), on utilisera la commande Ctrl+C et on choisira abort.

8.1 Les fonctions

De nombreuses fonctions sont définies dans Scilab. Il peut toutefois être utile (voire nécessaire) de définir de nouvelle fonctions. Il existe deux syntaxes pour écrire des fonctions :

Première méthode: `deff`. Que fait la fonction :

```
deff('[x,y] = mafonction1(a,b)',['x = a + b','y = a - b'])
```

Essayer en tapant par exemple `[x1,y1] = mafonction1(3,2)`.

Cette méthode est à utiliser quand il y a peu d'instructions.

Deuxième méthode: `function`. Que fait la fonction :

```
function [y]= mafonction2(x)

  y = x.*x;

endfunction
```

On peut enregistrer une fonction dans un fichier avec une extension **.sci**. Si le fichier **.sci** se trouve dans le répertoire courant, on peut charger la fonction en utilisant la commande `getf"nomdufichier.sci"`. Sinon, le nom du fichier doit être remplacé par le chemin d'accès complet.

8.2 Les boucles et les instructions conditionnelles

Exercice : Que fait la fonction suivante :

```
function [y] = mafonction2(x,n)
  if (n == 0)
    y = 1;
  else
    y = 1;
    for i = 1:n
      y = y.*x;
    end
  end
endfunction
```

Exercice : Programmer une fonction "naïve" qui renvoit le n-ime terme de la suite de Fibonacci definie par la récurrence suivante :

$$u_0 = u_1 = 1, \quad u_{n+2} = u_{n+1} + u_n.$$

Ecrire une fonction qui calcule u_n avec $n-1$ additions seulement (fonction récursive). Comparer les temps de calcul grâce à la commande `timer()`.

On peut rajouter `then` après `if ()`,à condition de laisser un espace après la parenthèse fermante:

```
if(n == 0) then
    y = 1;
  else
    y = 1;
  end
```

9 Pour s'entra^ner

Pour traiter ces exercices, on se réfèrera soit au récapitulatif des principales fonctions Scilab, soit l'aide de Scilab.

1. Définir la fonction qui à x renvoit x^2+1 et représenter la graphiquement sur l'intervalle $[-5,5]$. Ajouter un titre.

2. Définir une fonction qui prend une matrice M comme variable d'entrée et retourne une matrice N où les colonnes ont été permutées ainsi :

$$\text{si } M=(C_1,\ldots,C_n)\in M_{m,n}(\mathbb{C}),\ \text{alors } N=(C_2,\ldots,C_n,C_1)\in M_{m,n}(\mathbb{C}).$$

   Faire une version avec boucle et une version sans.

3. Définir une fonction qui prend un vecteur en entrée et retourne la matrice de Vandermonde associée. Pour $x=(x_1,..,x_m)$

$$A_x=\begin{pmatrix}1 & 1 & & 1\\ x_1 & x_2 & \ldots & x_m\\ \vdots & \vdots & & \vdots\\ x_1^{m-1} & x_2^{m-1} & \ldots & x_m^{m-1}\end{pmatrix}.$$

   On fera une version avec deux boucles imbriquées, avec une seule boucle et sans boucle. On comparera les temps de chacune des méthodes et on vérifiera que chaque version donne le même résultat. Vérifier que le déterminant est non nul si les x_i sont distincts et calculer dans ce cas son inverse.

4. Définir une fonction qui prend un vecteur en entrée et retourne le vecteur auquel on a appliqué la fonction f suivante élément par élément :

$$f(x)=\left|\begin{array}{l}x \text{ si } x<0\\ 0 \text{ sinon}\end{array}\right.$$

   Faire une version avec `if` et boucle et une version sans. Tracer la fonction sur l'intervalle $[-5,5]$.

5. Définir une fonction qui prend un vecteur x en entrée et retourne le vecteur auquel on a appliqué la fonction f suivante élément par élément :

$$f(x)=\left|\begin{array}{l}x^3 \text{ si } x<0\\ x^2 \text{ si } 0\leq x<1\\ x^4 \text{ si } x\geq 1\end{array}\right.$$

   Faire une version avec `if` et boucle et une version sans. Tracer la fonction sur l'intervalle $[-2,2]$.

6. Définir une fonction qui prend pour variables d'entrée un vecteur x et qui retourne la matrice tridiagonale

$$A = \begin{pmatrix} 2+x_0 & -1 & 0 & \dots & & \\ -1 & 2+x_1 & -1 & 0 & \dots & \\ 0 & \ddots & \ddots & \ddots & 0 & \\ & 0 & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & -1 \\ & & & 0 & -1 & 2+x_n \end{pmatrix}.$$

   On pourra utiliser les commandes `eye` et `diag`. Résoudre le système linéaire $Ax = b$ où b est égal au vecteur $(1, \dots, 1)'$ et où $x = 0$. Vérifier le résultat en utilisant la norme sup.

7. Grâce à la commande `subplot`, Tracer dans 4 sous-fenêtres (2 lignes et 2 colonnes) d'une fenêtre graphique, les courbes des fonctions $f(x) = \cos x$, $f(x) = \sin x$, $f(x) = \tan x$ et superposés $f(x) = \sin^2 x$, $f(x) = \cos^2 x$, entre $-\pi$ et π. Utiliser différentes formes de lignes et différentes couleurs. Ajouter des titres et des légendes.

8. Tester les différentes formes de graphique 3D sur les fonctions $f(x, y) = e^{-(x-1)^2-y^2}$, et $f(x, y) = \cos(x + y)$.

9. Refaire l'exercice précédent sans définir les fonctions (on utilisera `meshgrid`). Représenter leurs gradients (grâce à la commande `champ`).

10. Tracer les fonctions $x \mapsto x^i$ sur l'intervalle $[0, 1]$ avec $i \in \{1, \dots, 10\}$. On affectera une couleur distincte à chaque courbe.

    Réaliser ensuite une animation où l'on voit les courbes les unes après les autres. Ajouter un titre dynamique. On pourra se référer à la liste des commandes Scilab et utiliser les commandes `string` et `xpause`.