Feuille de TP n^o 1

Initiation à Scilab ; résolution approchée des équations non-linéaires

1 Initiation à Scilab

Lancez `scilab` à l'aide du menu de Windows (ou par la commande `scilab` si vous êtes sous Linux).

Manipulation de vecteurs et matrices

On peut utiliser `scilab` comme une calculatrice qui travaille avec des nombres (réels ou complexes) et avec des vecteurs et des matrices. L'ajout d'un point-virgule à la fin d'une ligne de commande supprime l'affichage.

Pour commencer, tapons les commandes suivantes

Définition de vecteurs et matrices

```
-> 2+5*3^2
-> (2+%i)*(1+%i)
-> x=[1,2,3]
-> y=[4;5;6]
-> z=[1,2;4,5];
-> z
-> t=1:4
-> u=0:0.1:1
```

Notez que la virgule sépare les entrés dans une ligne d'une matrice et le point-virgule sépare les lignes. Le symbole `:` sert à définir une suite arithmétiques, avec le pas égal à 1 ou non (on verra que le même syntaxe est utilisé dans les boucles).

Opérations avec des vecteurs et des matrices

La matrice transposée : `-> z'`
(Attention : pour les matrices complexes cela remplace aussi tous les coefficients par leurs conjugués)
Prendre le carré d'une matrice : `-> z^2`
D'autres exemples (en supposant qu'on a déjà tapé `-> x=[1,2,3]` `->y=[4;5;6]`

```
-> x+y'
-> A=[2,-1,0;1,2,0;0,0,1]
-> A*y
```

On constate donc que les opérations (+ pour addition et * pour multiplication) sur les vecteurs et matrices sont définis dans `scilab` de la même manières que dans nos cours de mathématiques.

Notons aussi la commande pour résoudre un système d'équations linéaires : `x=A\y` veut dire sous `scilab` qu'on cherche le vecteur x qui est la solution du systèmes $Ax = y$.

```
-> X=A\y
-> A*X
```

Notez que les variables `X` et `x` ne sont pas identiques.

Opérations composante par composante

```
-> x.^2
-> x./y'
-> (1)./x
-> z+1
```

Remarquez la différence entre `1/x` et `1./x` et entre `z^2` et `z.^2`

On peut aussi appliquer les fonctions standard (sin, cos, exp, ...) à des vecteur ou des matrices. Dans ce cas, la fonction est appliquée à chaque composante du vecteur (de la matrice).

Programmation avec `scilab`

Il est possible d'utiliser des fichiers pour regrouper des instructions scilab. Pour les rédiger, on peut utiliser un éditeur de texte intégré à `scilab` (menu Applications/SciNotes)

Exécution de scripts

Créer le fichier contenant les instructions suivantes

```
n = 5;
for i = 1 : n
  x(i) = i;
end
```

et le sauvegarder sous le nom *script.sce*, par exemple.

Remarque : les fichiers contenants des instructions exécutable par `scilab` sont couramment appelés des scripts.

L'exécution dans `scilab` s'effectue

— soit à l'aide du menu de l'éditeur (Executer/Enregistrer et exécuter ou F5) ;
— soit avec la commande de `scilab -> exec script.sce` (dans ce cas, il faut faire savoir à `scilab` dans quel répertoire se trouve le fichier : le menu Fichier/Changer le répertoire courant) ;
— soit (c'est l'option la plus simple pour nous) en sélectionnant la partie du texte dans l'éditeur et en choisissant "Evaluer la sélection" dans le menu contextuel du clic souris droit (ou CTRL-E).

Remarques sur notre premier script `scilab`

— La commande `for` sert pour définir des boucles avec un compteur. Notez l'analogie avec la commande de création de vecteur contenant des suites arithmétiques (au fait, on peut avoir le pas de boucle différent de 1 aussi dans la commande `for`) Il existe d'autres type de boucle, notamment la boucle du type "tant que", en anglais et en scilab c'est "while".
— Notez le syntaxe pour accéder à la composante numéro i du vecteur x : `x(i)`
— Notre script crée le vecteur-colonne contenant les nombres 1,2,3,4,5. On rappelle qu'on pourrait le faire avec des commandes plus simples : `x=1:5; x=x'`
— Notons une autre astuce pour faire la même chose (cette astuce pourra nous servir dans le futur pour stocker les résultats des calculs sur itérations) : on peut ajouter des nouvelles composantes à un vecteur déjà existant :

```
n = 5;
x=[];
for i = 1 : n
  x = [x;i];
end
```

Utilisation de fonctions

Créer le fichier mafonction.sci contenant les instructions suivantes

```
function y = f(x)
y = x.^3;
endfunction
```

Executer ce fichier (soit par commande `exec`, soit par la commande de menu de l'éditeur, soit avec CTRL-E sur les lignes sélectionnées). Dorénavant, `scilab` connait la nouvelle fonction f et on peut l'utiliser, par exemple, comme ça :

```
-> f([1,2,3])
```

Remarque : Il n'est pas nécessaire de créer un fichier séparé pour chaque fonction. On peut très bien mettre plusieurs fonctions dans un fichier et les mélanger avec des commandes hors fonctions. Attention alors : l'exécution d'un tel fichier fait `scilab` redéfinir toutes les fonctions et re-exécuter toutes les commandes. En pratique, il est souvent utile d'exécuter juste une partie du fichier que l'on vient d'écrire ou de modifier. Cela se fait avec CTRL-E.

Remarque pour ceux qui connaissent `C++` ou d'autres langages de programmation traditionnels : l'exécution du fichier contenants des fonctions s'apparente à la compilation des fichiers sources sous `C++` etc. Cela peut produire des erreurs de compilation, si le syntaxe des commandes n'est pas correcte. Bien sûr, l'absence

des erreurs de compilation ne veut pas forcement dire que la fonction fera ce qu'elle est censée faire : on peut toujours avoir des erreurs de logique qui ne deviennent apparentes que lorsque on utilise la fonction (erreurs d'exécution).

Exercice 1.

1. Ecrire une fonction `scilab` qui prend un numéro n comme paramètre et construit le vecteur contenant n premiers membres de la suite de Fibonacci, définie par les formules de récurrence :

$$a_1 = a_2 = 1,\ a_i = a_{i-1} + a_{i-2},\ \text{pour } i \geq 3$$

2. Ecrire une fonction `scilab` qui prend n comme paramètres et construit la matrice C de taille $n \times n$ tell que $c_{ij} = \frac{1}{i+j}$.
3. Créer un vecteur-colonne b de taille n (en prenant $n = 5, 15, 50$) qui contient les nombres de 1 à n et résoudre le système $Cx = b$ où C est la matrice de la question précédente. Chaque fois, vérifier que x est bien la solution en calculant la norme du vecteur $Cx - b$ (`scilab` met à notre disposition la fonction `norm`).

Outils graphiques

La commande de base pour dessiner une courbe (typiquement graphe d'une fonction) est

```
plot(x,y)
```

Cela prend 2 vecteurs x et y de même taille et dessine la courbe qui relie successivement les points x_i, y_i avec x_{i+1}, y_{i+1}

Des exemples :

```
-> x=0:0.1:2*%pi;
-> plot(x,cos(x))
-> plot(cos(x),sin(x),'g')
```

L'argument 'g' dans la dernière ligne veut dire green (vert). On peut utiliser d'autres couleurs (devenez lesquelles) : 'r', 'b', 'y', 'k', ...

Notez que les commandes `plot` successives ajoutent des courbes dans la même fenêtre graphique. C'est souvent utile, mais si on veut l'éviter, on peut

— Fermer la fenêtre graphique courante ;
— Taper `clf` pour effacer tout dans la fenêtre graphique courante ;
— Taper `figure` pour ouvrir une nouvelle fenêtre graphique.

Notons une variante de la commande plot

```
-> x=[1 2  3 2];
-> plot(x,'*')
```

Notez qu'ici il n'y a qu'un vecteur qu'on passe à la commande. Ca fait apparaitre des petites étoiles (l'argument '*' est pour ça) avec des ordonnées données par les composantes du vecteur x et les abscisses successives $1, 2, 3, \ldots$
On peut ajouter les couleurs, par exemple `'r*'` ou changer les étoiles en `'+'`

Utilisation de l'aide en ligne

Une aide en ligne est disponible grâce à l'instruction

```
help nom_de_commande.
```

Essayez
`help if` pour avoir des information sur l'exécution conditionnelle (si – alors – sinon), `help while` pour avoir des information sur la boucle "tant que".

2 Méthodes de point fixe et de Newton

Exercice 2. Le but de cet exercice est de résoudre (approximativement) l'équation

$$\cos(\frac{1}{x+1}) = x \tag{1}$$

On va notamment chercher une solution positive de cette équation.

1. Introduisons la fonction $g(x) = \cos(\frac{1}{x+1})$ et remarquons que l'équation (1) est de la forme "point fixe"' $x = g(x)$.
   — Créer la fonction `scilab` qui s'appelle `g` et qui calcule $\cos(\frac{1}{x+1})$ pour un nombre x donné.
   — Montrer graphiquement que la fonction g admet l'unique point fixe sur $\mathbb{R}^+$ (pour cela, il convient à dessiner les courbes $y = g(x)$ et $y = x$ sur le même dessin).
2. Implementer la méthode du point fixe et l'appliquer pour résoudre l'équation (1). On pourra faire plusieurs implémentations de plus en plus subtils :
   — Tout d'abord, écrire un script qui exécute n itérations de la méthode de point fixe et affiche les approximations successifs (boucle "for"). Essayer la méthode avec plusieurs point de départ x_0.
   — Ajouter la commande qui stocke les approximations successifs dans un vecteur X. Visualiser ensuite ce vecteur avec `plot(X,'*')`.
   — Ecrire un script qui exécute autant d'itérations de la méthode de point fixe qu'il faut pour être sûr qu'on trouvé la solution à ε près, avec ε donné (boucle "while").
3. Implémentons maintenant la méthode de Newton pour résoudre l'équation $f(x) = 0$ et l'appliquons à l'équation (1).
   — Créer la fonction `scilab` qui s'appelle `f` et qui calcule $f(x) = \cos(\frac{1}{x+1}) - x$ pour un nombre x donné.
   — Créer la fonction `scilab` qui s'appelle `df` et qui calcule $f'(x)$ pour un nombre x donné.
   — Créer un script qui exécute n itérations de la méthode de Newton et affiche les approximations successifs (n donné).
4. Comparons les vitesse de converges de 2 méthodes ci-dessus, c'est-à-dire les vitesse avec lesquelles les suites x_n tendent vers la solution exacte x^*. Pour cela il faut déjà avoir x^*. Donc le plan du travail peut être le suivant :
   — Executons 100 itérations de la méthode de Newton. Cela donne une très bonne approximation à la solution exacte x^*. Appelons cette approximation (sous `scilab`) `xstar`.
   — Soit `Xpf` le vecteur contenant les 10 premières approximations données par la méthode du point fixe. Calculons le vecteur des erreurs `Epf=abs(Xpf-xstar)` (ici, la fonction `abs` est pour calculer la valeur absolue composante par composante).
   — Soit `XN` le vecteur contenant les 10 premières approximations données par la méthode de Newton. Calculons le vecteur des erreurs `EN=abs(XN-xstar)`.
   — Visualiser les vecteur `Epf` et `EN`. Qu'est-ce qu'on peut en conclure ? Pour voir mieux, il convient à plotter les log10 de vecteurs des erreurs.

Exercice 3. Le but de cet exercice est de résoudre (approximativement) le système de 2 équations par la méthode de Newton :

$$\begin{aligned} -5x + 2\sin x + \cos y &= 0 \\ 4\cos x + 2\sin y - 5y &= 0 \end{aligned}$$

Pour cela, on note $x_1 = x$, $x_2 = y$ et on introduit l'application $f : \mathbb{R}^2 \to \mathbb{R}^2$ qui reprend les 2 lignes du système ci-dessus

$$\begin{aligned} f_1(x) &= -5x_1 + 2\sin x_1 + \cos x_2 \\ f_2(x) &= 4\cos x_1 + 2\sin x_2 - 5x_2 \end{aligned}$$

Ainsi, le problème est reformulé comme une équation vectorielle $f(x) = 0$.

1. Créer la fonction `scilab` qui s'appelle `f`, prend un vecteur x de taille 2 comme paramètre d'entrée et calcule le vecteur f comme ci-dessus.
2. Créer la fonction `scilab` qui s'appelle `df` et qui calcule la matrice 2×2 de dérivées partielles de $f(x)$ pour un vecteur x donné.
3. Créer un script qui exécute n itérations de la méthode de Newton et affiche les approximations successifs (n donné). Tester l'algorithme avec plusieurs points de depart. Observer la convergence vers la solution (ou peut-être l'absence de convergence).