Mastère de Mathématiques. Université de Rennes I
Analyse réelle et complexe TP No 2.

Introduction à MATLAB: 2ème partie

1. Utilitaire graphique

Lisez l'explication sur la Feuille No. 1 de la représentation graphique des résultats (Chapitre 7). Une clarification concernant la commande `title` : si vous avez, par exemple, deux nombres réels qui s'appellent `x` et `y`, alors le titre pourrait être `title(['Dessin avec x=',x,' et y=',y])`

Exercice 1. Générez, à l'aide d'un programme prog1.m, deux périodes des deux signaux x(t)=cos(t) et y(t)=sin(t) et visualisez les deux signaux sur une même figure, l'une en rouge et l'autre en bleu à l'aide de la fonction plot(t,x,'b',t,y,'r'). Mettre une légende.

Exercice 2. Visualisez sur une même figure, à l'aide d'un programme prog2.m, les quatre signaux cos(t), sin(t),log10(t) et exp(t) en utilisant la fonction subplot(.,.,.) qui divise l'écran en quatre sous figures : (2,2,1),(2,2,2),(2,2,3) et (2,2,4).
>>subplot(2,2,1),plot(t,x);

2. Saisie d'une donnée au clavier:

Il est possible d'assigner une valeur provenant du clavier à une variable en utilisant la commande "input". Ceci veut dire que l'utilisateur doit manuellement initialiser les variables une fois que le programme est exécuté.

Pour saisir une variable x à partir du clavier, on utilise l'instruction: x=input('x=') ;
Pour afficher à l'écran un message personnel suivi d'un retour à la ligne :
fprintf('message personnel \n');
Exemple:
Tapez:
>>clear
>>clc
>>test= input('S.V.P entrer un chiffre \n');
>>2
>>test

3. Les fonctions

Si plusieurs de vos programmes personnels utilisent en commun une liste d'instructions, il est préférable de regrouper ces instructions sous forme d'un programme indépendant. A chaque besoin on appelle le dit programme par son nom : c'est une fonction.
Une fonction possède des paramètres d'entrée et des paramètres de sortie, dont la syntaxe de déclaration est la suivante :
function [sortie1,sortie2,...]=nom_fonction(entrée1,entrée2,...)
Le programme matlab correspondant à la fonction doit porter le même nom que la fonction : nom_fonction.m
Exemple :
Calcul de la moyenne arithmétique m d'un vecteur v de dimension n :

$$m = \frac{1}{n}\sum_{i=0}^{n} v_i.$$

La fonction moyenne.m comportera les instructions suivantes :

```
function resultat=moyenne(v,n); %déclaration de la fonction
r=0; %initialisation de la moyenne
for k=1:n
r=r+v(k);
end
resultat=r/n;
```

Exercice 3. Créez une fonction puissance.m qui donne la valeur efficace s et la puissance moyenne p d'un signal.
x étant un vecteur de composantes $x(i), i = 1..n$, la valeur efficace de x est :

$$s = \sqrt{p} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x(i)^2}.$$

Testez cette fonction sur un signal sinusoïdal d'amplitude A et de période 1.
Comparer avec la fonction matlab std.

Exercice 4. (Simulation et calcul exact d'une loi de Poisson) Soit $\lambda \in \mathbf{R}_+$ (pour nous, des valeurs de λ entre 1 et 20 marcheront bien). Prenons `TailleEchantillon` (par exemple = 500) réalisations d'une variable aléatoire qui prend la valeur 1 avec une probabilité de $p = \frac{\lambda}{\texttt{TailleEchantillon}}$ et la valeur 0 avec une probablité de $1 - p$. En moyenne, il y a alors λ occurrences de "1", mais parfois il y en a un peu plus, et parfois moins. Si on repete ce test `NombreDeTests`(= 2000, par exmple) fois, et on dessine un histogramme du nombre d'occurrences de "1" dans les `NombreDeTests` test, on obtient un dessin comme dans la figure.

Dans cette figure, les valeurs `lambda=4.5`, `TailleEchantillon`= 500 et `NombreDeTests`= 2000 ont été utilisés. En plus, l'histogramme ne regarde le nombre d'occurrences de "1" qu'entre 0 et `4*floor(lambda)`.

On sait théoriquement combien de fois, en moyenne, on devrait avoir exactement $k \in \mathbb{N}$ occurrences de "1", à savoir

$$\texttt{NombreDeTests} \cdot exp(-\lambda)\frac{\lambda^k}{k!}$$

Le graphe de cette fonction $\mathbb{N} \to \mathbf{R}, k \mapsto \texttt{NombreDeTests} \cdot exp(-\lambda)\frac{\lambda^k}{k!}$ est également dessiné dans la figure.

Réaliser :
• Un programme `poisson.m` qui demande à l'utilisateur de rentrer les valeurs de λ, `TailleEchantillon` et `NombreDeTests`, qui fait la simulation, et qui dessine ensuite un histogramme comme dans la figure.
• Une fonction `poissontheo(lambda)` qui renvoie un vecteur

$$\exp(-\lambda) \cdot \left(\frac{\lambda^0}{0!}, \frac{\lambda^1}{1!}, \frac{\lambda^2}{2!}, \ldots, \frac{\lambda^{k_{max}}}{k_{max}!}\right) \quad \text{où} \quad k_{max} = 4 * \texttt{floor(lambda)}$$

• Une continuation du programme `poisson.m` qui superpose le graphe de `NombreDeTests*poissontheo` sur l'histogramme précédent.

4. Les boucles

En plus des commandes vues jusqu'à maintenant, Matlab permet d'inclure dans des fichier.m des instructions de programmation classiques.

4.1. **Conditions: if : : : else : : : end.** La syntaxe est la suivante :

```
if (test)
commandes
else
autres commandes
end
```

On peut également imbriquer des if : : : else les uns dans les autres à l'aide de l'instruction elseif.

```
if (test 1)
commandes
elseif (test 2)
commandes
elseif (test 3)
...
else
commandes
end
```

Le test est une expression booléenne (vrai ou faux).

4.2. **Répétitions: for : : : end.** La syntaxe est la suivante :

```
for (k = liste)
commandes
end
```

On peut naturellement imbriquer des boucles for : : : end les unes dans les autres. Attention à ne pas utiliser les variables [i] et [j] comme itérateurs car ces variables représentent le nombre complexe.

Exemple

```
%nombres de Fibonacci et nombre d'or a=1;
fib=1;
for (k = [1:10])
tmp=a;
a=fib;
fib=fib+tmp
or=fib/a
end
```

4.3. **Répétitions: while : : : end.** La syntaxe est la suivante :

```
while (test)
commandes
end
```

L'exemple ci-dessus affiche des matrices aléatoires en boucle. La commande drawnow force Matlab à dessiner à chaque boucle et non une seule fois à la fin.

```
k=0;
figure;
while (k ¡ 50)
k = k+1;
A=rand(10,10);
imagesc(A);
axis off equal;
drawnow;
end
```

L'execution des boucles est BEAUCOUP plus lente que la modification de vecteurs à l'aide des

opérations "composante par composante". Illustrons cela avec la montre automatique (faites help tic pour voir comment elle marche) :

```
>> v=[0:0.01:100*pi]; M=length(v)
>> tic, w=v.*[1:M]; TimeDirect=toc
TimeDirect =
0.0159
>> tic, for i=1:M, v(i)=v(i)*i; end, TimeBoucle=toc
TimeBoucle =
0.7330
>> TimeBoucle/TimeDirect
ans =
46.2328
>> max(v-w)
ans =
0
```

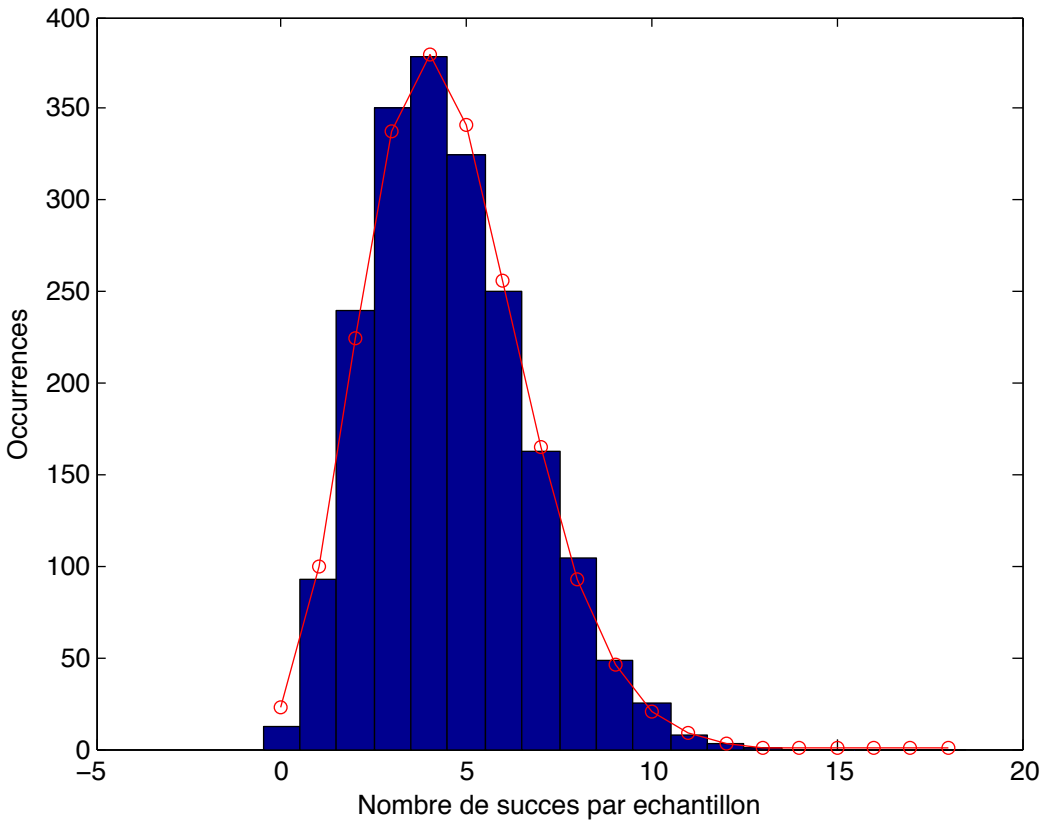
Le temps d'exécution d'une boucle est plus que 40 fois plus long !!! Et pourtant on calcule bien la même chose, car la dernière commande montre que toutes les composantes de (v-w) valent 0. En utilisant, bien entendu, une opération "composante par composante" et non pas une boucle! Toutefois, les boucles sont plus faciles à programmer lorsque il s'agit d'une opération récursive.

Exercice 5. 1) Définir en une ligne le vecteur à composantes sin(1), sin(4), sin(7), sin(10), ..., sin(10000). Faire la même chose en utilisant une boucle. Comparer les temps d'exécution.
2) Considérons les deux transformations d'un vecteur v :

```
>> for i=2:length(v), v(i)=v(i)-v(i-1); end
>> for i=1:length(v)-1, v(i)=v(i)-v(i+1); end
```

Une des deux peut être programmée avec une grande économie de temps d'exécution. Laquelle? Comment?
Réécrire l'autre transformation en utilisant une boucle "while".

À rendre : vos solutions des exercices 4 et 5.


400
350
300
250
200
150
100
50
0
Occurrences
−5
0
5
10
15
20
Nombre de succes par echantillon