INITIATION A SCILAB

M1-M2

MODELISATION EN BIOLOGIE DES POPULATIONS ET DES ECOSYSTEMES

MODELISATION DU FONCTIONNEMENT DES ECOSYSTEMES

PARTIE I & II

(VERSION 1.1)

Soudani Kamel (Professeur)- Fac. Sciences Orsay- Université Paris Sud
kamel.soudani@u-psud.fr

PLAN

Introduction

Scilab est un logiciel libre de calcul scientifique développé depuis 1990 par l'INRIA (Institut National de Recherche en Informatique et Automatique) et l'ENPC (Ecole Nationale des Ponts et Chaussées). Il est disponible gratuitement sur le site Internet **www.scilab.org**. Dans sa syntaxe, son architecture et son fonctionnement, Scilab est très proche au logiciel Matlab commercialisé par (Mathworks Inc). Depuis 2003, le développement de Scilab se fait dans le cadre d'un groupe d'entreprises et d'organismes de recherche constitué de : Axs Ingenierie, CEA, CNES, Cril Technology, Dassault-Aviation, EDF, ENPC, Esterel Technologies, INRIA, PSA Peugeot Citroën, Renault, Thales.

I. Interface Scilab

Démarrer **Scilab 4.0** :

1- Dans la fenêtre principale (Environnement Scilab) : le symbole « --> » marque l'invite de commande Scilab (commandes en ligne).

Dans le menu Fichier - changer de répertoire et se mettre dans le répertoire de travail (E\MOD). Utiliser la commande ***ls*** pour vérifier que c'est le bon répertoire. ***ls*** est l'équivalent de la commande ***dir*** du DOS. La commande ***dir*** fonctionne aussi sous Scilab.

2- Activer l'éditeur Scilab (= « Scipad ») en cliquant sur Editeur du menu principal. L'éditeur Scilab est l'interface d'écriture et du lancement d'exécution des programmes.
L'éditeur Scipad peut être aussi lancé à partir de l'invite des commandes Scilab en tapant : ***scipad.*** Remarquer que Scilab différencie entre les minuscules et les majuscules. Par exemple si on tape **Scipad** au lien de ***scipad***, Scilab affiche un message d'erreur.

II. Commandes en ligne sous Scilab

II.1 Commandes de gestion de fichiers sous Scilab

On commencera pas quelques commandes équivalentes aux commandes disponibles sous UNIX ou DOS.

Sur la l'invite de commande :

→ ***chdir*** E:\ ou ***cd*** permet d'aller dans disque E.

→ ***ls*** ou **dir** permet d'afficher le contenu du répertoire actif (E:\).

La commande **ls** peut être aussi utilisée avec des options. Par exemple ***ls*** *.txt affiche tous les fichiers ayant l'extention *txt*.

→ ***mkdir*** MOD\TD créer en une seule fois un sous répertoire TD dans un répertoire MOD.

→ ***rmdir*** MOD\TD supprimer le répertoire TD dans le répertoire MODE.

→ ***pwd*** affiche le répertoire courant de Scilab.

→ ***who*** affiche la liste de toutes les variables utilisées dans la session Scilab (voir plus loin).

→ ***gethistory()*** retourne l'historique des commandes utilisées dans les sessions antérieures.

→ ***savehistory***('E:\MOD\histScila') sauve l'historique dans un fichier qu'on nomme histScila dans le répertoire E:\MOD. Le fichier histScila est lisible par un simple éditeur de texte.

→***resethistory()*** efface l'historique

→ ***clc*** effacer toutes les commandes dans la fenêtre Scilab

→ ***clear*** efface le contenu en mémoire de toutes les variables sauf celles protégées. Elle peut être utilisée pour effacer une seule variable qu'on nomme A par exemple : **clear** A

*Il est fortement conseillé de mettre les commandes **clc** et **clear** sur les deux premières lignes d'un programme Scilab édité par l'éditeur Scipad afin de rafraichir l'écran et vider la mémoire Scilab.*

→ **exit** quitter la session Scilab
Pour d'autres commandes, voir l'aide de Scilab section ***Utilitaires***.

II.2 Utilisation de l'invite Scilab comme une calculatrice

Ecrire sur la fenêtre de commande puis Entrée du clavier :

→ 2+3*4

Scilab répond par :
ans =
14
ans pour answer (réponse). Par défaut Scilab nomme la variable résultat ans.

Ecrire :

→ result = 2+3*4
Scilab répond par :
result =
14

Result est le nom de la variable correspondant au résultat du calcul. Result est alors considéré comme une variable enregistrée en mémoire et qu'on peut utiliser dans la suite de la session Scilab.

Ecrire puis Entrée du clavier :

```
→ a = 2
a =
2
→ result/a
ans =
7
```

On a donc affecté à la variable *a* la valeur 2 puis on a divisé la variable result/a.

Le calcul peut s'effectuer aussi en utilisant des fonctions comme : exponentielle (exp ()), sinus (sin()), cosinus (cos()). Pour la syntaxe exacte des autres fonctions, voir la section ***fonctions élémentaires*** dans le menu ***aide*** de Scilab.

III. Editeur de programmes Scipad

L'éditeur de programme Scipad permet l'écriture de lignes de codes en respectant une syntaxe de la même manière que dans d'autres langages de programmation comme le Turbo Pascal, C, Fortran, etc.

Une fois exécuté, le programme dirige les sorties directement vers l'interface Scilab vue précédemment (sortie écran) ou vers une autre sortie de type graphique ou fichier en écriture. Toutes les variables ou constantes utilisées dans le programme sont sauvegardées en mémoire et sont accessibles directement à partir de l'invite de commandes.

1. Lancer l'éditeur Scipad et écrire le code simple suivant :

```
clc
clear
//Programme pour le calcul la taille d'une population après n années
//Nbre d'individus au temps t=0 est P=P0
P0=25
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5
//Nbr d'individus après (n) nombre d'années est P
n=10
P= ((r+1)^n)*P0
printf('La taille de la population après %d années est %f
individus',n,P)
```

Exemple 1

2. Exécuter ce programme : Menu Exécuter – Load in Scilab. Enregistrer le programme sous Exemple1.sce.

sce étant l'extension des fichiers édités par scipad.

Analysons tout d'abord le prgramme (Exemple1.sce)

// Pour saisir des commentaires. Toute ligne précédée par // n'est pas exécutée par Scilab. Donc seules les lignes 5, 8, 10, 11 et 12 qui seront exécutées.

L'exécution est séquentielle, c'est-à-dire ligne par ligne depuis la première jusqu'à la dernière.

Ligne 5 : on affecte la valeur 25 à la variable P0
Ligne 8 : on affecte la valeur 1.5 à la variable r
Ligne 10 : on affecte la valeur 10 à la variable n
Ligne 11 : on calcule le nombre d'individus après un nombre d'années n.

$$P = P0\,(1+r)^n$$

Ligne 12 : on affiche le résultat en utilisant la fonction **printf** disponible dans Scilab. Cette fonction est une fonction parmi tant d'autres de la librairie Scilab.

Toutes ces lignes sont affichées sur la fenêtre Scilab. Pour ne pas afficher une ligne ajouter **;** à la fin de la ligne. L'affichage peut ralentir énormément l'exécution. Utiliser ; afin de ne pas afficher certaines lignes.

Toutes les variables citées ci-dessus sont accessibles via l'invite de commande Sciab.

```
→ P0
P0  =
25.
→ r
r    =
1.5
```

L'utilisation de **printf** s'effectue comme suit :

printf(format, var1, var2, etc.)

format correspond au type de données :
%d pour afficher une variable de type entier naturel.
%f pour afficher une variable de type réel.
%s pour afficher un caractère ou une chaîne de caractères.

Exemple :
→ **printf**('La valeur de P0 est %d. La veleur de r est %f',P0,r)

IV – Le Langage de Programmation Scilab

IV-1. Structures de contrôle d'un programme

Ces structures sont les ordres permettant d'orienter l'exécution d'un programme.

Boucle for – end

La boucle **for** permet l'exécution répétée d'une suite d'instructions situées entre les mots clés ***for*** – début de la boucle et ***end*** – fin de la boucle. Le nombre de répétitions est déterminé par la valeur

maximale prise par un compteur de répétitions ainsi que par la valeur initiale du compteur et le pas d'incrémentation.

Syntaxe : ***for*** *compteur= debut : pas : fin*
Instruction1
Instruction2
…..
end

Par défaut, Scilab prend un pas de 1. On peut alors écrire : ***for*** *compteur= debut : fin*

Exercice 1 : Modifier le programme situé ci-dessus (Exemple1.sce) afin de calculer et afficher le nombre d'individus de la population pour chaque année (de 1 à 10). Enregistrer le programme sous le nom Exemple2.sc.

Corrigé de l'exercice 1. (Exemple2.sce)

```
clc
clear
//Programme pour le calcul de la taille d'une population après n années
//Nbre d'individus au temps t=0 est P=P0
P0=25
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5
//Nbr d'individus aprés (n) nombre d'années est P
n=10
for i=1:1:10
P= ((r+1)^i)*P0
printf('La taille de la population après %d années est %f individus',i,P);
end
```

Boucle while - end

La boucle while permet de répéter l'exécution d'une suite d'instructions (tant qu'une expression (condition de répétition) est vérifiée.

Syntaxe : ***while*** *expression*
Instruction1
Instruction2
…
end

expression est une condition logique qui peut être soit vraie (1) ou fausse (0). Si la condition est vraie, l'exécution demeure dans la boucle, sort de la boucle dans le cas contraire.

Les opérateurs de comparaison sont les suivants :

= =	égal à
<	inférieur à

```
>          supérieur à
<=         inférieur ou égal à
>=         supérieur ou égal à
<> ou ~=   différent de
```

« & » (et) et « | » (ou) : on peut tester des conditions multiples. Par exemple :

```
((a==2) & (c>=5))     //Cette commande a pour résultat « vrai » si à
                      la fois a=2 et c >= 5, et faux si l'une au moins de
                      ces deux conditions est fausse.

((a==2) | (c>=5))     //Cette commande a pour résultat « vrai » si
                      une des deux conditions est vraie, ou les deux.
```

Exercice 2 : Dans Exemple2.sce, calculer et afficher le nombre d'individus de la population pour chaque année (de 1 à 10) en utilisant la boucle while au lieu de la boucle for. Enregistrer ce programme sous le nom Exemple3.sce.

Corrigé de l'exercice 2. (Exemple3.sce)

```
clc
clear
//Programme pour le calcul de la taille d'une population après n années
//Nbre d'individus au temps t=0 est P=P0
P0=25
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5
// Nbr d'individus aprés (n) nombre d'années est P
n=10
i=1;
while(i<=10)
P= ((r+1)^i)*P0
printf('La taille de la population après %d années est %f individus',i,P);
i=i+1
end
```

if -then-else

L'ordre conditionnel if évalue une expression qui, si elle est vraie (au sens booléen), commande l'exécution de la suite d'instructions comprise entre les mots-clés then et else (ou end). Si l'expression est fausse, c'est la suite d'instructions se trouvant entre les mots-clés else et end qui est exécutée. La partie else…end est facultative (un end doit alors délimiter la dernière instruction du then). Des constructions sont également possibles avec le mot-clé elseif. Le mot-clé then peut être remplacé par un retour-chariot ou une virgule.

Syntaxe 1: **if** *expression then*
Instruction1
Instruction2
else

```
        Instruction3
        Instruction4
    end
```

Exercice 3 : Modifier l'Exemple3.sce afin de limiter la taille de la population à une valeur maximale à ne pas dépasser. Pour l'exemple, on fixera ce seuil à 10000. Afficher l'année pour laquelle ce seuil est atteint. Enregistrer ce programme sous le nom Exemple4.sce.

```
clc
clear
//Programme pour le calcul de la taille d'une population après n années
//Nbre d'individus au temps t=0 est P=P0
P0=25;
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5;
//  Nbr d'individus aprés (n) nombre d'années est P
n=10;
i=1;
seuil=10000;
while(i<=10)
P= ((r+1)^i)*P0
    if (P<=seuil) then
    printf('La taille de la population après %d années est %f
individus',i,P);
    else
    printf('La population a atteint le seuil de %d individus entre l''année
%d et l''année %d',seuil,i-1,i)
    i=n;
// cette dernière ligne permet de forcer la sortie de la boucle while
    end
    i=i+1;
end
```

select…case

La syntaxe de l'utilisation de cette instruction conditionnelle s'effectue comme suit :

```
select expr,
   case expr1 then instructions1,
   case expr2 then instructions2,
   ...
   case exprn then instructions,
   [else instructions],
 end
```

Notes:
- La seule contrainte est que chaque mot-clé "*then*" soit sur la même ligne que le "*case*" correspondant.
- Le mot-clé *then* peut être remplacé par un passage à la ligne ou une virgule. Les instructions1 sont exécutées si expr1=expr, etc.

Exercice 4 : Modifier le programme situé ci-dessus (Exemple1.sce) afin d'effectuer le calcul en considérant deux cas. Le premier cas : croissance de la population sans mortalité et le second cas en considérant une mortalité qu'on note m. On considérera que m est une proportion constante de la population totale. Pour l'exemple, on prendra m = 15% de la population totale. Enregistrer le programme sous le nom Exemple5.sce.

```
clc
clear
//Programme pour le calcul de la taille d'une population après n années
//Nbre d'individus au temps t=0 est P=P0
P0=25
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5
//  Nbr d'individus aprés (n) nombre d'années est P
n=10

//mortalité m : proportion constante du Nbr d'individus de la population

mortalite=1
select mortalite
case 0 then P= ((r+1)^n)*P0
printf('La taille de la population après %d années est %f individus',n,P)
case 1
m=0.15
P= ((r-m+1)^n)*P0
end
```

Ordre break

L'ordre break force la sortie d'une boucle. Généralement, il faut éviter d'utiliser cette instruction car son utilisation traduit le plus souvent une faiblesse dans l'algorithme.

```
clc
clear
P0=25
r=1.5
seuil=10000
for i=1:10
P= ((r+1)^i)*P0
if(P>seuil) then break
end
end
```

IV-2. Fonctions sous Scilab

Définition en ligne d'une fonction

Dans l'aide de Scilab, la définition en ligne d'une fonction se fait comme suit :

deff('[s1,s2,...]=nom_function(e1,e2,....)',text [,opt])

Parameters

e1,e2,..., : arguments d'entrée de la fonction
s1,s2,..., : arguments de sortie de la fonction
text : la fonction écrite
opt : options
'c' : la fonction est compilée (par défaut)
'n' : la fonction est non compilée.

Exemple : Dans l'exemple 1, le nombre d'individus de la population peut être calculé en utilisant la fonction suivante :

```
clc
deff('[P]=Nb_individus(P0,n,r)','P=((r+1)^n)*P0')
```

Ecrire cette ligne dans l'éditeur Scipad, enregistrer ensuite le fichier dans Exemple6.sce. Exécuter ensuite le programme.
Dans l'invite de commande Scilab, l'appel de la fonction peut se faire comme suit :

```
→ Nb_individus(25,10,1.5)
ans  =

    238418.58
→
```

Création de fonctions dans l'environnement Scilab à l'aide du mot clé function

Syntaxe :

```
function <arguments_sortie>=<nom_de_la_fonction><arguments_entrée>
       <instructions>
        endfunction
```

Vérifions le fonctionnement du code suivant :

```
function P=Pop_fonction(n,r,P0)
P=((r+1)^n)*P0
endfunction
n=10
r=1.5
P0=25
Pop_fonction(n,r,P0)
```

La fonction Pop_fonction prend pour entrées n, r et P0 et rend la taille de la population.

Sous l'invite de Scilab

```
-->P
  !--error 4
undefined variable : P
```

Dans l'environnement Scilab, P est non définie.
Essayons encore comme suit :

```
P=100
function P=Pop_fonction(n,r,P0)
P=((r+1)^n)*P0
endfunction
n=10
r=1.5
P0=25
Pop_fonction(n,r,P0)
printf('La valeur de P après l'appel de la fonction est %f',P)
```

Après Exécution du code ci-dessus, Scilab rend le résultat suivant :

La valeur de P après l'appel de la fonction est 100.000000

La valeur prise par la variable P n'est pas modifiée par l'exécution. P est alors dite locale à la fonction. P est une variable locale. La fonction peut seulement lire les variables du corps du programme mais elle ne les modifie pas. Pour pouvoir les modifier, ces variables doivent être déclarées comme étant des variables globales en utilisant le mot clés ***global****.*

```
global P
P=10
function P=Pop_fonction(n,r,P0)
global P
P=((r+1)^n)*P0
return(P)
endfunction
n=10
r=1.5
P0=25
Pop_fonction(n,r,P0)
```

(Remarquer que P est déclaré deux fois ainsi que la commande return(P))

Création de fichiers de fonctions et appel de ces fonctions

Afin de pouvoir utiliser la fonction dans plusieurs programmes, celle-ci peut être écrite dans un fichier à part en respectant la même syntaxe. L'appel de la fonction dans le programme se fait en utilisant la commande exec('Nom_fichier). Nom_fichier contient aussi le chemin exact du fichier. Par exemple : ***exec(E:\MOD\Nom_fichier.sce)***

Exemple : Appel de la fonction Pop_fonction.

Dans un fichier qu'on nommera **Calcul_Pop.sce,** écrire la fonction suivante :

```
function P=Pop_fonction(n,r,P0)
global P
P=((r+1)^n)*P0
return(P)
endfunction
```

Ensuite, écrire un autre programme appelant cette fonction :

```
n=10
r=1.5
P0=25
exec('calcul_pop.sce')
Pop_fonction(n,r,P0)
```

IV-3. Vecteurs et matrices

IV-3.1 Un petit rappel de l'algèbre des matrices

Une matrice est un tableau de n lignes et m colonnes. Si on note M la matrice et a_{ij} les éléments de la matrice, la matrice M = [a_{ij}] de taille (n,m) est composée des éléments a_{ij}, i allant de 1 à n et j allant de 1 à m. i est donc l'indice de ligne et j est l'indice de colonne.

1. Addition

La somme de deux matrices A et B est une matrice dont chaque élément est une somme des éléments de mêmes indices des deux matrices.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1m} \\ b_{21} & b_{22} & \dots & b_{2m} \\ \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nm} \end{bmatrix} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \dots & a_{1m}+b_{1m} \\ a_{21}+b_{21} & a_{22}+b_{22} & \dots & a_{2m}+b_{2m} \\ \dots & \dots & \dots & \dots \\ a_{n1}+b_{n1} & a_{n2}+b_{n2} & \dots & a_{nm}+b_{nm} \end{bmatrix}$$

Pour pouvoir être additionnées, les deux matrices A et B doivent être de mêmes dimensions (même nombre de lignes et même nombre de colonnes).

2. Multiplication

1. Par un scalaire : $\lambda \lfloor a_{ij} \rfloor = \lfloor \lambda a_{ij} \rfloor$
2. Par une matrice : produit matriciel

Le produit matriciel se fait entre deux matrices dont le nombre de colonnes de la matrice de gauche est égal au nombre de lignes de la matrice de droite :

Si A = (n,m) et B (p,q) : le produite matriciel entre A et B => m = p ; la matrice résultante est dimension (p,q).

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & \dots & \dots. & \dots. \\ a_{n1} & a_{n2} & \dots. & a_{nm} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots. & \dots. \\ b_{n1} & b_{n2} & \dots. & bp \end{bmatrix} = \begin{bmatrix} a_{11} \times b_{11} + a_{12} \times b_{21} + \dots + a_{1m} \times b_{n1} & \dots \\ \dots. & \dots \\ \dots & \dots \quad \dots. \quad \dots. \\ & \dots. \end{bmatrix}$$

Exemple :

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ae+bf \\ ce+df \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} j & m \\ k & n \\ l & o \end{bmatrix} = \begin{bmatrix} aj+bk+cl & am+bn+co \\ dj+ek+fl & dm+en+fo \\ gj+hk+il & gm+hn+io \end{bmatrix}$$

3. Vecteur propre et valeur propre d'une matrice

Soit A une matrice carrée de n lignes et n colonnes et X un vecteur colonne de n lignes et λ étant un scalaire. Considérons l'équation suivante :

$$A\ X = \lambda\ X$$

Pour X non nul, les valeurs λ qui vérifient cette équation sont appelées **valeurs propres** de la matrice **A**. Les vecteurs correspondants sont appelés **vecteurs propres**.

IV-3.2 Matrices et Tableaux sous Scilab

Matrices

Considérons par exemple la matrice suivante (3 lignes, 4 colonnes) :

$$\begin{pmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 1 \\ 9 & 6 & 7 & 11 \end{pmatrix}$$

Nous allons commencer par "entrer" cette matrice dans Scilab. Il existe plusieurs manières de procéder. Nous allons ici l'entrer comme une liste d'éléments. Pour ce faire, il suffit de respecter quelques règles syntaxiques simples :

- Sur la même ligne, les éléments doivent être séparés par des espaces ou des virgules.
- La fin d'une ligne est signalée par un point-virgule.

- L'ensemble de la liste de valeurs doit être entourée de crochets, [ ].

Pour entrer la matrice que nous nommerons matrice [S], nous taperons ainsi :

```
S = [16 3 2 13; 5 10 11 1; 9 6 7 11]
```

```
-->S = [16 3 2 13; 5 10 11 1; 9 6 7 11]
```

Et Scilab affichera aussitôt la matrice de la manière suivante :

```
S  =

    16.    3.     2.     13.

    5.     10.    11.    1.

    9.     6.     7.     11.
```

L'accès à l'élément 2 (1er ligne, 3e colonne) se fait en tapant : S(1,3). En règle générale, l'accès à l'élément situé sur la ligne i et la colonne j se fait en tapant : S(i,j). Les opérations arithmétiques peuvent se faire alors entre les différents éléments de la matrice. Par exemple, pour somme les éléments de la première ligne :

```
     Somme=S(1,1)+S(1,2)+S(1,3)+S(1,4)
```

```
-->Somme=S(1,1)+S(1,2)+S(1,3)+S(1,4)

Somme  =

    34.
```

On peut aussi effectuer la même opération comme suit :

```
-->sum(S(1,:))

 ans  =

    34.
```

sum est une fonction prédéfinie Scilab pour calculer une somme. Ici, le calcul s'effectue sur la première ligne et toutes les colonnes. L'opérateur : signifie toutes les colonnes.

L'opérateur « : »

L'opérateur « : » est l'un des opérateurs les plus importants de Scilab. Il s'emploie de plusieurs manières. Ainsi l'expression :

```
     1:10
```

désigne un vecteur ligne formé des entiers compris entre 1 et 10, ce qui conduit à l'affichage de :

```
--M=1:10
M  =

    1.    2.    3.    4.    5.    6.    7.    8.    9.    10.
```

Il est possible de spécifier un espacement différent de 1 entre les valeurs successives. Ainsi :

```
-->M=10:-1:1
M  =

    10.    9.    8.    7.    6.    5.    4.    3.    2.    1.

-->M=10:-2:1
 M  =

    10.    8.    6.    4.    2.
```

Revenons à la matrice S = [16 3 2 13; 5 10 11 1; 9 6 7 11]

```
S  =

    16.    3.     2.     13.

    5.     10.    11.    1.

    9.     6.     7.     11.
```

L'opérateur (:)permet aussi d'accéder à une portion de la matrice. Si par exemple, on veut construire une nouvelle matrice qu'on note A composée des deux premières lignes et des deux dernières colonnes, alors A=S(1:2,3:4)

```
-->A=S(1:2,3:4)
 A  =

    2.     13.
    11.    1.

-->S(1,:)
 ans  =

    16.    3.    2.    13.
```

Correspond à une matrice ligne composée de la première ligne de la matrice S.

```
-->S(:,1)
```

```
 ans  =

    16.
    5.
    9.
```

Correspond à la matrice colonne composée de la première colonne de la matrice S.

Taille de la matrice

La taille de la matrice s'exprime sous la forme : (nb de lignes, nb de colonnes). Elle est obtenue par la fonction size(S). Le résultat est :

```
ans  =

!   3.    4. !
```

Pour récupérer le nombre de ligne ou le nombre de colonne d'une matrice, on peut procéder comme suit :

```
-->Taille=size(S)

 Taille  =

    3.    4.

-->nb_ligne=Taille(1,1)

 nb_ligne  =

   3.

-->nb_colonne=Taille(1,2)

 nb_colonne  =

    4.
```

Somme de lignes et de colonnes

La fonction sum() permet d'effectuer la somme des éléments des lignes ou des colonnes.

La somme des éléments situés sur les colonnes se fait comme suit :

```
     sum(S,'c')
```

conduira à l'affichage de :

```
     ans=
```

```
      !34.!
      !27.!
      !33.!
```

Alors que la commande : sum(S,'r')

aura pour sortie :

```
       ans=

        ! 30. 19. 20. 25. !
```

Concaténation

On appelle concaténation l'opération consistant à fusionner de petits objets (ici des matrices) en une entité plus grande. L'opérateur de concaténation des matrices est []. Ainsi :

```
→a = [ 1 2 3 ]; b= [ 4 5 6]; c = [ 7 8 9];

→d= [a b c]

   d=

    ! 1. 2. 3. 4. 5. 6. 7. 8. 9. !

→e= [a; b; c]

    e =

    ! 1. 2. 3. !

    ! 4. 5. 6. !

    ! 7. 8. 9. !
```

suppression de lignes et de colonnes

```
-->S

S  =

    16.    3.     2.     13.
    5.     10.    11.    1.
    9.     6.     7.     11.
```

Pour supprimer la dernière colonne de la matrice S :

```
-->S(:,4)=[]

 S  =
    16.    3.     2.
```

5. 10. 11.
9. 6. 7.

Exemple simple : écriture d'un programme pour remplir une matrice de taille n * m avec des valeurs aléatoires selon une loi uniforme. Chaque élément est indicé en i et j. On utilisera une boucle for qui parcourt les m éléments de la première ligne puis une seconde qui parcourt toutes lignes.

```
clc              //effacer l'écran scilab
clear            // effacer toutes les variabes en mémoire
n=5
m=6
for i=1:5   // parcourir les lignes de 1 à 5
for j=1:6   // parcourir les 6 colonnes de chacune des lignes
A(i,j)=rand()//Remplir A(i,j) par des valeurs aléatoire entre 0 et 1
end              // fin de la boucle j
end              // fin de la boucle i
```

Rand() : fonction Scilab rendant des valeurs aléatoires entre 0 et 1 selon la loi uniforme. Toutes les valeurs entre 0 et 1 sont équiprobables. (voir plus loin)

```
A  =

    0.4256872    0.2461561    0.9229532    0.1000746    0.4678218    0.3950498

    0.0366117    0.5175369    0.8325452    0.6104832    0.1871112    0.0189575

    0.8433565    0.0748595    0.8532815    0.0124590    0.1867539    0.4920584

    0.7489608    0.9414957    0.2124056    0.579502     0.2628148    0.4360987

    0.9110545    0.8082667    0.8102653    0.2590428    0.4139087    0.3599928
```

```
// une façon plus simple d'écrire le code ci-dessus
A=rand(n,m)

// rand(n,m) rend une matrice n * m dont les éléments sont
aléatoires entre //0 et 1 (Uniforme).
```

Tableaux

Un Tableau est une matrice. Scilab offre la possibilité d'effectuer des opérations sur les tableaux sans pour autant les considérer comme des matrices et donc non soumis aux conditions de l'algèbre linéaire. Par exemple, on peut multiplier deux tableaux élément par élément en utilisant l'opérateur **.*.** On peut aussi utiliser les opérateurs suivants :

.\ division à gauche par éléments
.* multiplication par éléments
./ division à droite par éléments
.^ puissance par éléments

Opérateur ./ (Le tableau S est divisé par le tableau b élément/élément)

```
-->S=[1 2; 4 8]
 S  =

    1.    2.
    4.    8.

-->b=[2 2; 2 2]
 b  =

    2.    2.
    2.    2.

-->S./b
 ans  =

    0.5    1.
    2.     4.
```

Opérateur .*

```
-->S.*b
 ans  =

    2.    4.
    8.    16.
```

Opérateur .\ (le tableau b est divisé par le tableau S)

```
-->S.\b
 ans  =

    2.     1.
    0.5    0.25
```

L'opérateur .^ (Puissance par élément du tableau S)

```
-->S.^2
 ans  =

    1.     4.
    16.    64.
```

Autres opérations sur une matrice ou un «tableau»

```
-->exp(S)        rend l'exponentielle de la matrice élément/élément
 ans  =

    2.7182818    7.3890561
    54.59815     2980.958

-->sqrt(S)       rend la racine carré de la matrice élément/élément
 ans  =
```

1. 1.4142136
2. 2.8284271

Pour d'autres opérations (log, cos, sin, etc…) voir l'aide Scilab.

Opérations matricielles

A'	transposée de A
rank	rang
inv	inverse
expm	exponentielle matricielle
det	déterminant
trace	trace
poly(A,"x")	polynôme caractéristique de A
spec	valeurs propres de A
bdiag	diagonalisation
svd	décomposition en valeurs singulières
A\b	solution de A*x=b
b/A	solution de x*A=b
linsolve(A,b)	solution de A*x=-b
mean(A)	moyenne de la matrice

V. Entrées – Sorties en mode fichier

Sauvegarde d'une ou plusieurs variables dans un fichier ascii (texte)

Soit le programme suivant :

Syntaxe :
--> **savematfile('Nom_fichier','Nom_Var1','Nom_Var2', '.. ','-ascii')**

Pour sauvegarder en binaire, supprimer '-ascii'

```
//Programme pour le calcul du nombre d'individus d'une population
après (n) années
//Nbre d'individus au temps t=0 est P=P0
P0=25
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5
//  Nbr d'individus aprés (n) nombre d'années est P
n=10
for i=1:1:10
annee(i)=i;
```

```
P(i)= ((r+1)^i)*P0
end
```

A la sortie du programme ci-dessus, deux variables Tableau peuvent être récupérées : la tableau *annee* et le tableau *P*.

```
-->who_user      (liste de variable dans l'environnement Scilab)
 User variables are:
 P         annee     i         n         r         P0
 scipad    modelica_libs
 using 1698 elements out of 4984309
```

Les deux variables P et annees sont accessibles en tapant :

```
-->P
 P  =

    62.5
    156.25
    390.625
    976.5625
    2441.4062
    6103.5156
    15258.789
    38146.973
    95367.432
    238418.58

-->annee
 annee  =

    1.
    2.
    3.
    4.
    5.
    6.
    7.
    8.
    9.
    10.

--> savematfile('Res_P.txt','P','annee','-ascii')
```

Ouvert sous Bloc Notes, le fichier Res_P.txt est le suivant :

```
 6.2500000E+01
 1.5625000E+02
 3.9062500E+02
 9.7656250E+02
 2.4414062E+03
 6.1035156E+03
 1.5258789E+04
 3.8146973E+04
 9.5367432E+04
```

```
 2.3841858E+05
 1.0000000E+00
 2.0000000E+00
 3.0000000E+00
 4.0000000E+00
 5.0000000E+00
 6.0000000E+00
 7.0000000E+00
 8.0000000E+00
 9.0000000E+00
 1.0000000E+01
```

Remarquer que les deux variables sont sauvées l'une après l'autre.

Une manière simple de résoudre ce problème est de concaténer les deux tableaux :

```
-->B=[annee P]
 B  =

    1.      62.5
    2.      156.25
    3.      390.625
    4.      976.5625
    5.      2441.4062
    6.      6103.5156
    7.      15258.789
    8.      38146.973
    9.      95367.432
    10.     238418.58

-->savematfile('P_Result.txt','B','-ascii')
```

Les deux variables seront enregistrées dans deux colonnes distinctes.

On peut aussi utiliser la commande write :

```
-->savematfile('P_Result.dat',B,)
```

Sauvegarde d'une matrice S dans un fichier nommé matrice .dat :

→ `write('matrice.dat',S)`

Lecture d'une ou plusieurs variables dans un fichier ascii (texte)

La commande la plus facile à utiliser est la commande *read*.

→ `A=read(' P_Result.txt',Nbligne,Nbvariables)`

La commande **read** nécessite de spécifier le nombre de lignes et le nombre de variables.

VI. Graphiques sous Scilab

Les principales commandes graphiques sont les suivantes

plot2d(x,y)	Tracé de la courbe passant par les points (x,y)
plot2d1('oll',x,y)	Idem, avec échelle logarithmique sur les deux axes
fplot2d(x,f)	Tracé de la courbe (x,f(x))
plot3d(x,y,z)	Tracé de la surface passant par les points (x,y,z)
contour(x,y,z,n)	Tracé de n courbes de niveau d'une surface
hisplot(n,data)	Histogramme de l'échantillon data divisé en n classes

x,y sont des vecteurs

```
//Programme pour le calcul du nombre d'individus d'une population
après (n) années
//Nbre d'individus au temps t=0 est P=P0
P0=25
//Nombre de descendants par individu et par an est r
//Hyp: pas de mortalité ni chez les adultes ni chez les nouveaux nés
r=1.5
//  Nbr d'individus aprés (n) nombre d'années est P
n=10
for i=1:1:10
annee(i)=i;
P(i)= ((r+1)^i)*P0
end
plot2d(annee,P);
xset('background',4)
xtitle(['Taille de la population'],'Années','Nbre individus')
```

./.

INITIATION A SCILAB

M1-M2

MODELISATION EN BIOLOGIE DES POPULATIONS ET DES ECOSYSTEMES

MODELISATION DU FONCTIONNEMENT DES ECOSYSTEMES

PARTIE II – EXEMPLES

(VERSION 1.1)

Soudani Kamel (Professeur)- Fac. Sciences Orsay- Université Paris Sud kamel.soudani@u-psud.fr

Modélisation à base d'équations différentielles : évolution en temps discret et en temps continu des équations de Lotka – Voltera proies-prédateurs

Considérons le modèle cité dans la partie I. L'effectif de la population au temps t+1 est donné par l'équation suivante : $P_{t+1}=P_t+r*P_t$
r étant un taux d'accroissement moyen de la population entre le temps t et t+1 (r :un nombre de descendants par individu de la population pendant un pas de temps de 1). Dans cette équation, le temps est discret et le pas de temps est 1 (r étant défini pour ce pas de temps). Considérons maintenant un pas de temps quelconque Δt et gardons r comme étant le taux d'accroissement pendant le pas de temps Δt. On peut alors écrire que :

$$\frac{P_{t+\Delta t}-P_t}{\Delta t}=rP_t \qquad (1)$$

Ainsi, on peut d'une manière itérative déterminer les quantités $P_{t+\Delta t}$ en fonction des quantités P_t ,r et P_0 :

$$P_{t+\Delta t}=P_t+r\Delta tP_t \qquad (2)$$

Dans l'équation (1), lorsque Δt tend vers 0, on obtient l'équation différentielle suivante :

$$\frac{dP}{dt}=rP \qquad (3)$$

Cette équation donne l'accroissement de la population pendant un temps infinitésimal *dt*. r est le taux d'accroissement pendant ce temps *dt*.

En temps discret, la résolution de cette équation différentielle peut se faire d'une manière itérative selon l'équation (2) :

$$P_{t+\Delta t}=P_t+r\Delta tP_t$$

```
clc
clear
r=1.5
// On définit P initial au temps t=0 ;
P0=25
//j : indice du tableau P qui stockera les valeurs P(j)
j=1
//Le premier élément du tableau P(1)=P0
P(1)=P0;
//On définit un tableau qui stockera le temps, le premier élément est 0
temps(1)=0;
// on definit le pas de temps Deltat
Deltat=0.1;
for i=Deltat:Deltat:10
j=j+1;
temps(j)=temps(j-1)+Deltat;
P(j)=P(j-1)+r*Deltat*P(j-1);
end
plot2d(temps,P)
```

Dans le programme ci-dessus, le pas de temps Δt est égal à 0.1 et r est défini pendant ce pas de temps. Il est bien évident que la valeur finale de P est d'autant plus importante que Δt est petit.

En effet, l'expression de P est la suivante :

$$P_i = (1+r)^{\frac{t}{\Delta t}} P_0 \text{ , } t = i\Delta t, \quad i = 0,1,2,...,n \qquad (2)$$

Lorsque le modèle est conçu et défini en temps continu, la résolution analytique de l'équation différentielle (Eq.1) aboutit à l'expression suivante :

$$P_t = P_0 e^{rt} \qquad (3)$$

Il est bien évident que les résultats obtenus par les expressions (2) et (3) sont différents. Cette différence est d'autant plus grande que Δt est grand.

Pour résoudre une équation différentielle, Scilab utilise la fonction ode (ordinary differential equation). L'utilisation se fait comme suit :

```
y=ode(y0,t0,t,f)

 y0   : vecteur ou matrice réelle (conditions initiales).
 t0   : réel (instant initial).
 t    : vecteur réel (instants où la solution est renvoyée).
 f    : fonction externe à résoudre (Equation différentielle).
```

La fonction f peut être définie en utilisant les instructions ***deff*** ou ***function*** de Scilab (voir Partie I). Pour une utilisation avec l'instruction ode, la fonction doit avoir la syntaxe suivante : ydot = f(t,y) où t est un scalaire (le temps), y un vecteur (l'état). Cette fonction renvoie le second membre de l'équation différentielle dy/dt=f(t,y).

Exemples :

$$\frac{dy}{dt} = y^2 - y.\sin(t) + \cos(t)$$

pour des valeurs de t comprise entre 0 et π et la condition initiale : y = 0 pour t= 0.

```
deff("[ydot]=f(t,y)","ydot=y^2-y*sin(t)+cos(t)")
y0=0;
t0=0;
t=0:0.1:%pi;
y=ode(y0,t0,t,f);
plot(t,y)
```

Ou pour $\frac{dP}{dt} = rP$

```
deff("[Pprim]=Pop(t,P)","Pprim=r*P")
r=2.5
y0=100;
t0=0;
t=0:0.1:10;
P=ode(y0,t0,t,Pop);
plot(t,P)
```

Ici, la fonction notée Pop(t,P) est de la forme f(t,y). Le nom de la fonction est Pop. Les solutions pour les différentes valeurs de t sont récupérées dans la variable P grâce à l'instruction P=ode(y0,t0,t,Pop).

Exercice : Equations Lotka-Voltera proies – prédateurs (Modifié d'après Hendrik Davi 2004/2005)

L'évolution des populations de proies X et de prédateurs Y est définie comme suit :

$$\frac{dX}{dt} = r \times X - g \times X \times Y$$

r est le taux d'accroissement de la population de proies et g le taux d'efficacité de la prédation. Cette relation suppose que les ressources sont non limitantes et que la seule limitation de l'accroissement de la population est provoquée par les prélèvements par les prédateurs.

$$\frac{dY}{dt} = e \times g \times X \times Y - m \times Y$$

e est un coefficient de conversion spécifiant le nombre de jeunes prédateurs générés par proies capturées. m est le taux de mortalité des prédateurs. Dans cette dernière équation, le prédateur se nourrit d'une manière exclusive des proies X.

Dans cet exercice, les simulations s'effectueront en utilisant des procédures itératives et la fonction ode de Scilab.

La procédure itérative est basée comme indiqué ci-dessus en discrétisant le système comme suit :

$$X_{t+1} = X_t + h \times (r \times X_t - g \times X_t \times Y_t)$$
$$Y_{t+1} = Y_t + h \times (e \times g \times X_t \times Y_t - m \times Y_t)$$

Cette méthode est dite la méthode d'Euler.

Questions :

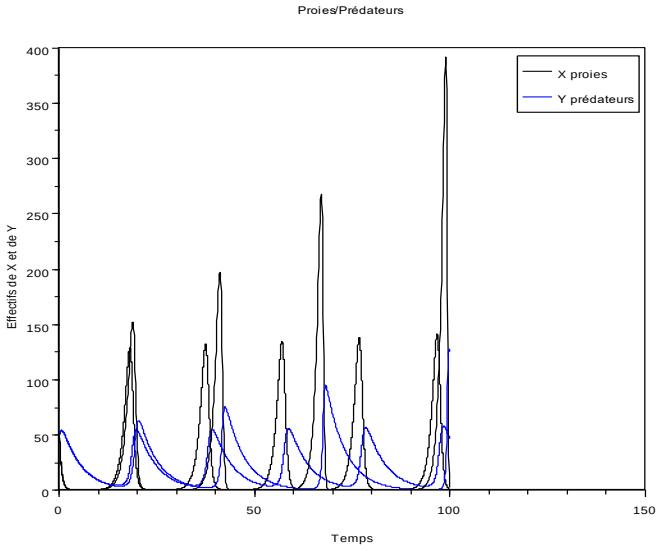
Considérons les constantes suivantes :

```
r=1;
g=0.05;
e=0.2;
m=0.2;
X0=50 ;
Y0=50 ;
```

Simulation numérique :

1/.Calculer l'évolution des populations de proies et de prédateurs avec différents pas de simulation (h=0.01, h=0.1…).
2/.Tracer le graphique de l'évolution de la population pour les différentes valeurs de h.

```
clc
clear all
//htest tableau contenant les valeurs de h à tester
htest=[0.01 0.1];
for i=1:2
//boucle pour parcourir htest
h=htest(i)
r=1;
g=0.05;
e=0.2;
m=0.2;
X(1)=50 ;
Y(1)=50 ;
t(1)=0;
j=1;
for i=h:h:100
j=j+1;
t(j)=t(j-1)+h;
X(j)=X(j-1)+h*(r*X(j-1)-g*X(j-1)*Y(j-1));
Y(j)=Y(j-1)+h*(e*g*X(j-1)*Y(j-1)-m*Y(j-1));
end
set(gca(),"auto_clear","off")
//permet de ne pas effacer la fenêtre graphique et superposer ...
//les figures pour h=0.1 et h=1
plot2d(t,[X Y],rect=[0,0,150,400])
X=[];
Y=[];
t=[];
//vider les tableaux X, Y et t
xtitle('Proies/Prédateurs','Temps','Effectifs de X et de Y')
legends(['X proies';'Y prédateurs'],[1,2],1)
end
```



Simulation analytique :

Le temps est ici considéré comme continu ce qui diffère de l'approche de simulation discrète.

On définit des paramètres globaux (« global ») qui pourront être définis dans le programme principal et dans la fonction. Le mot clés global dans le programme et dans le corps de la fonction permet de rendre une variable accessible et modifiable par la fonction.

```
global r g e m
r=1; g=0.05; e=0.2; m=0.2;
```

On définit la fonction f qui donne la vitesse d'apparition ou de disparition des proies et des prédateurs :

```
function[ydot]=f(t,y)
global r g e m
proies=r*y(1)-g*y(1)*y(2);
pred=-m*y(2)+e*g*y(1)*y(2);
ydot=[proies;pred];
endfunction
```

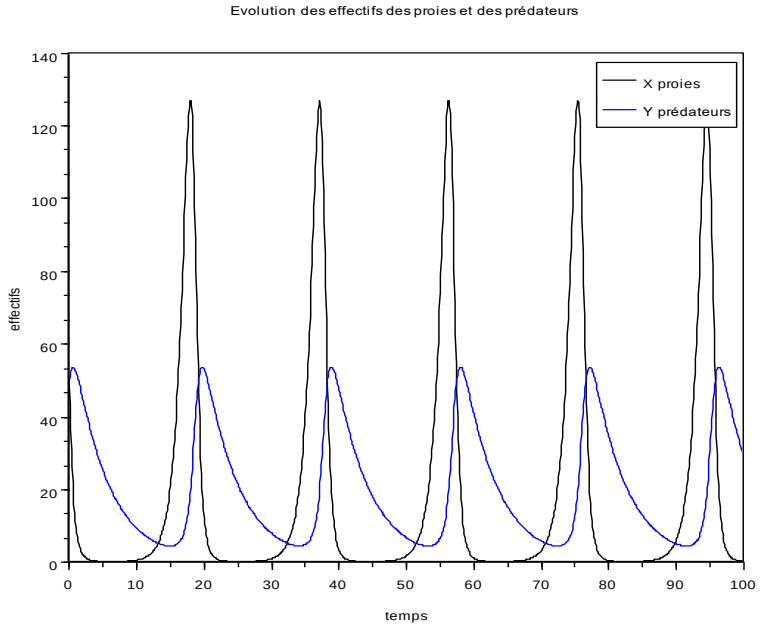
1/. Tracer les graphiques de l'évolution des proies et des prédateurs au cours du temps.
2/. Tracer les graphiques de l'évolution de chaque population en fonction de l'autre.

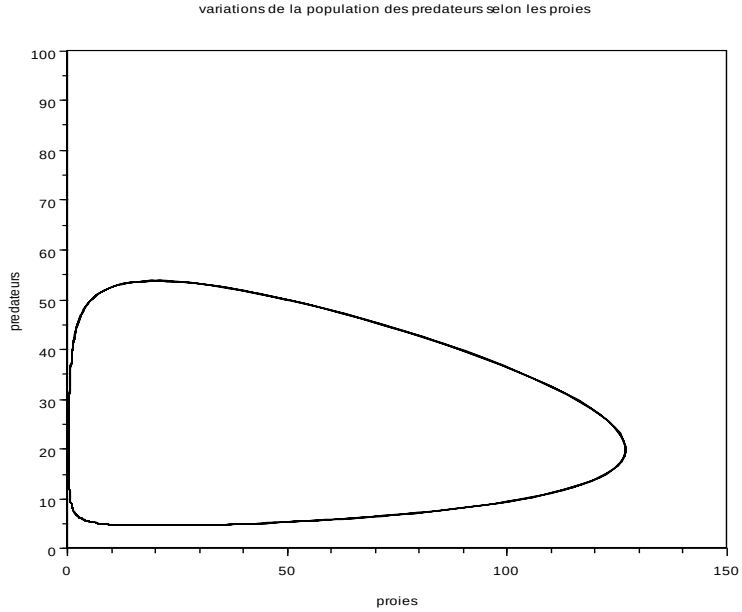
3/. Etude des différents cas : donner des explications succinctes de chaque résultat.
avec r=1; g=0.05; e=0.2; m=0.2;
avec r=1.08; g=0.2; e=0.4; m=0.04;
5/. Comparer les approches itératives et continues sur un même graphique.

```
sol=[];
function[ydot]=f(t,y)
global r g e m
proies=r*y(1)-g*y(1)*y(2);
pred=e*g*y(1)*y(2)-m*y(2);
ydot=[proies;pred];
endfunction
global r g e m
r=1; g=0.05; e=0.2; m=0.2;
t=0:0.1:100;
y0=[50;50]
sol=ode(y0,0,t,f);

effectif=sol';
//permet de transformer le tableau sol en un tableau de deux colonnes
proies=effectif(:,1);
predateurs=effectif(:,2);
xset("window",1)
//permet d'ouvrir une fenêtre 1 pour la figure suivante
plot2d(t,[proies predateurs])
```

```
xax='temps'
yax='effectifs'
xtit='Evolution des effectifs des proies et des prédateurs'
xtitle(xtit,xax,yax)
legends(['X proies';'Y prédateurs'],[1,2],1)
xset("window",2)
//permet d'ouvrir une fenêtre 2 pour la figure suivante
plot2d(proies,predateurs,rect=[0,0,150,100])
//rect  permet de fixer l'échelle des graduations =[xmin,ymin,xmax,ymax]
xax='proies'
yax='predateurs'
xtit='variations de la population des predateurs selon les proies'
xtitle(xtit,xax,yax)
```





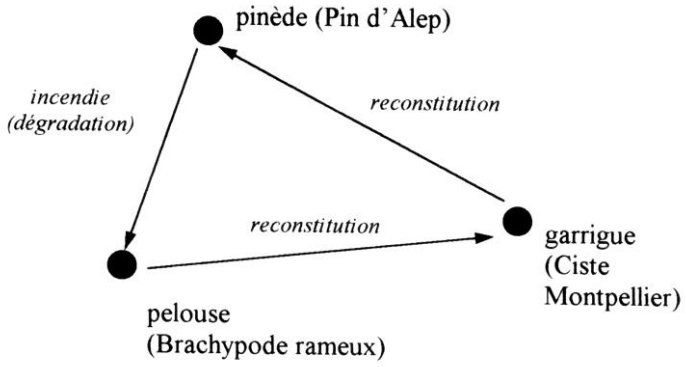
Modélisation des processus stochastique : Chaîne de Markov

Processus stochastique : L'état présent du système est aléatoire et ne dépend pas de l'état précédent. Si on note x(t) l'état du système au temps t. Le processus est dit stochastique, lorsque d'une manière consécutive (t1, t2, ti, ...tn) produit des états x(ti) aléatoires. x(ti) est une réalisation d'une variable aléatoire X(t).

Chaînes de Markov

Dans une chaîne de Markov, l'évolution du système est discrète. L'état initial du système est connu. Le passage d'un état quelconque (à t) à l'état suivant (t+Δt) se réalise selon un probabilité donnée et fixe. Cette probabilité est appelée probabilité de transition. L'ensemble de probabilités constitue la matrice de transition.

Considérons le cas suivant :



(Tiré de l'ouvrage modélisation et simulation d'écosystème de P. Coquillard et D.R.C Hill).

Un espace géographique caractérisé par l'existence à des proportions différents de trois couverts végétaux : Pinède, Garrigue et Pelouse. Soient Ppi, Pg et Ppe, les proportions d'occupation respectives de chacun des trois couverts. Pp+Pg+Ppe=100%. Dans le processus temporel de succession végétale, la pinède est totalement dégradée et se transforme en Pelouse. La pelouse est totalement reconstruite en Garrigue et enfin la garrigue est totalement reconstruite en Pinède.

Dans le cas général, notons les probabilités de passage d'un état à un autre comme suit :

Probabilité pi->pe : probabilité de passage d'une pinède en pelouse
Probabilité pe->pi : probabilité de passage d'une pelouse en pinède
Probabilité pe->g : probabilité de passage d'une pelouse en garrigue
Probabilité g->pe : probabilité de passage d'une garrigue en pelouse
Probabilité g>pi : probabilité de passage d'une garrigue en pinède
Probabilité pi->g : probabilité de passage d'une pinède en garrigue
Probabilité pi->pi : probabilité d'une pinède de rester pinède
Probabilité pe->pe : probabilité d'une pelouse de rester pelouse
Probabilité g->g : probabilité qu'une garrigue de rester garrigue.

Les proportions en surface entre deux temps successifs peuvent se calculer comme suit :

Ppe(t+Δt)=Ppe(t)*(pe->pe)+Pg(t)*(g->pe)+Ppi(t)*(pi->pe)
Pg(t+Δt)=Ppe(t)*(pe->g)+Pg(t)*(g->g)+ Ppi(t)*(pi->g)
Ppi(t+Δt)=Ppe(t)*(pe->pi)+Pg(t)*(g->pi)+Ppi(t)*(pi->pi)

Le système d'équations ci-dessus peut s'écrire sous une forme matricielle :

$$\begin{bmatrix} Ppe \\ Pg \\ Ppi \end{bmatrix}_{t+\Delta t} = \begin{bmatrix} pe \to pe & g \to pe & pi \to pe \\ pe \to g & g \to g & pi \to g \\ pe \to pi & g \to pi & pi \to pi \end{bmatrix} \begin{bmatrix} Ppe \\ Pg \\ Ppi \end{bmatrix}_t$$

On peut alors conclure que :

$P_{t+\Delta t} = [T] * P_t$

Si on note P_0 les proportions en surface au temps t = 0 et un pas de temps Δt de 1. Le système au-dessus peut se simplifier comme suit :

$P_t = [T] * P_{t-1} = [T]^n * P_0$

Pour l'exemple de la succession ci-dessus, on peut écrire :

$$\begin{bmatrix} Ppe \\ Pg \\ Ppi \end{bmatrix}_t = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} Ppe \\ Pg \\ Ppi \end{bmatrix}_{t-1}$$

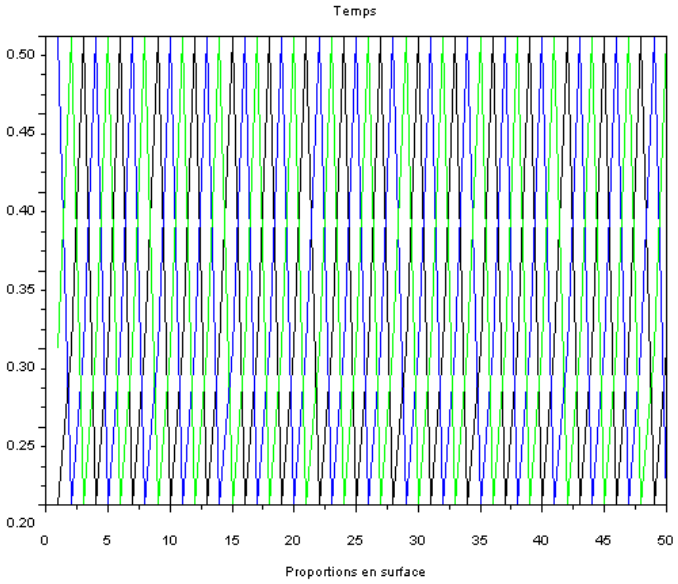
A l'état stationnaire (proportion de chaque thème inchangée entre t et t+1), on obtient $P_{t+1} = P_t$
On a alors : $T P_t = \lambda P_t$

On peut alors conclure que l'état stationnaire est obtenu pour P en tant que vecteur propre droite associé à la valeur propre λ=1. La structure de ce vecteur propre correspond aux proportions des différents thèmes d'occupation lorsque l'état d'équilibre est atteint. La commande scilab :eigenmarkov(T) donne les vecteurs propres gauche et droite associés à la matrice T. Noter que T est la matrice de transition telle que la somme des éléments en ligne (colonnes) doit égal à 1. Pour plus de détails, consultez l'aide Scilab.

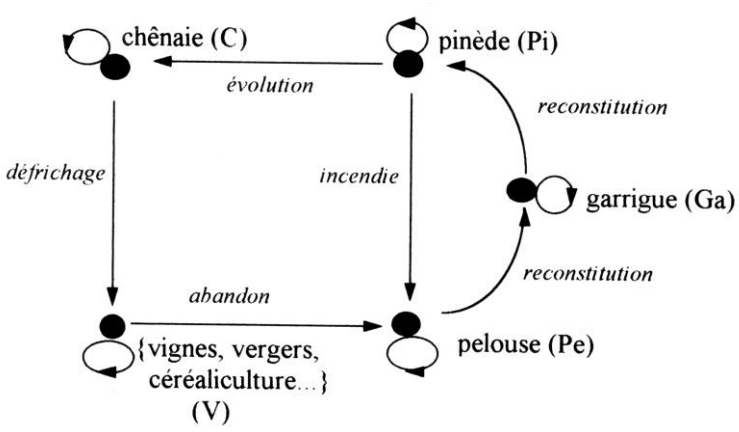
Sous Scilab :

```
clear
clc
// matrice initiale donnant les proportions
Ppe=0.5;
Pg=0.3
Ppi=0.2
P0=[Ppe;Pg;Ppi];
//matrice de probabilité
```

```
T = [0 0 1; 1 0 0;0 1 0]
n=50
for i=1:n
t=i;
Vectemps(i)=t;
P=(T^i)*P0;
Ppe(t)=P(1);
Pg(t)=P(2);
Ppi(t)=P(3);
end
mtlb_hold("on")
plot2d(Vectemps,Ppe,1)
plot2d(Vectemps,Pg,2)
plot2d(Vectemps,Ppi,3)
xtitle('Temps','Proportions en surface')
```



Considérons maintenant le cas plus complexe suivant :



(Tiré de l'ouvrage modélisation et simulation d'écosystème de P. Coquillard et D.R.C Hill).

La matrice de transition est la suivante :

T= [0.8 0. 0. 0. 0.1 ; 0.2 0.7 0. 0. 0. ; 0. 0.3 0.4 0. 0.25 ; 0. 0. 0.6 0.2 0. ; 0. 0. 0. 0.8 0.65]

Exercice : Simulation d'un processus stochastique markovien

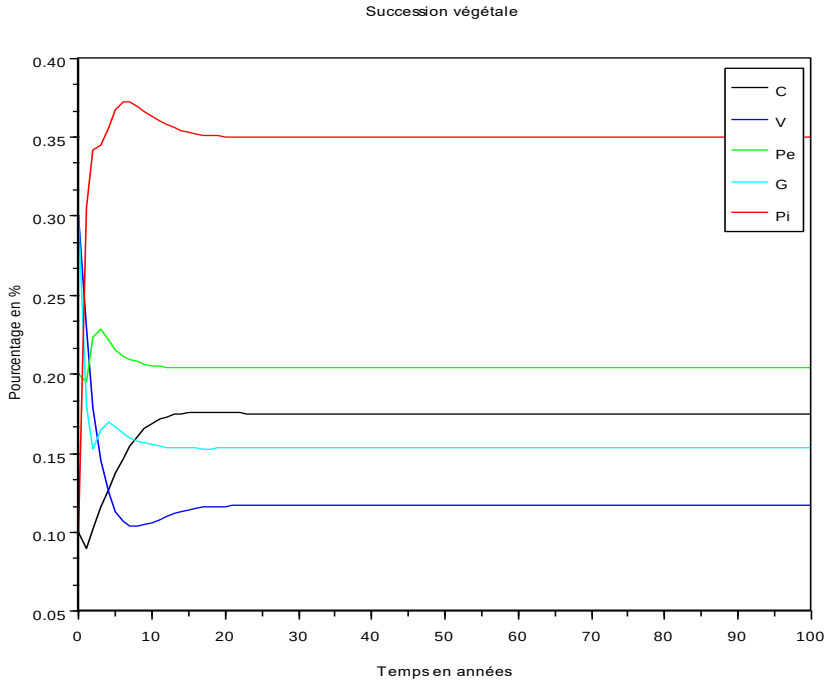
1/ Créer sous l'éditeur de texte Scipad la matrice de transition donnée ci-dessus :

T= [0.8 0. 0. 0. 0.1 ; 0.2 0.7 0. 0. 0. ; 0. 0.3 0.4 0. 0.25 ; 0. 0. 0.6 0.2 0. ; 0. 0. 0. 0.8 0.65]

L'ordre est C, V, Pe, G et Pi

2 / Créer la matrice donnant l'état du système au temps t=0
Les proportions sont les suivantes P : 0.1, 0.3, 0.2, 0.3, 0.1 respectivement pour C, V, Pe, G et Pi.
3/ Déterminer l'état du système au temps t=1
4/ Déterminer l'état du système aux temps t sur une période de 100 ans.
5/ Tracer ensuite sur le même graphique l'évolution de tous les thèmes cités ci-dessus.
6/ Trouver les valeurs propres gauche et droite de la matrice de transition (utiliser la commande eigenmarkov, voir l'aide).
7/Stocker les résultats sous la forme d'une matrice dont les lignes contiennent les proportions de chaque temps pour les temps t=0 au temps t=100.

```
clear
//Dans l'ordre : C, V, Pe, G et Pi
P0=[0.1; 0.3; 0.2; 0.3; 0.1] ;
T= [0.8 0 0 0 0.1 ;0.2 0.7 0 0 0 ;0 0.3 0.4 0 0.25 ;0 0 0.6 0.2 0;0 0 0 0.8 0.65]
Mat_temp=[];
for i=1:100
P=T^i*P0;
Mat_temp=[Mat_temp;P'];
end
Mat_Prop=[P0';Mat_temp];
C=Mat_Prop(:,1);
V=Mat_Prop(:,2);
Pe=Mat_Prop(:,3);
G=Mat_Prop(:,4);
Pi=Mat_Prop(:,5);
temps=0:1:100;
plot2d(temps,[C,V,Pe,G,Pi])
legends(['C';'V';'Pe';'G';'Pi'],[1,2,3,4,5],1)
xtitle('Succession végétale','Temps en années','Pourcentage en %')
savematfile('Succession.txt','Mat_Prop','-ascii')
```



Dans cet exemple, la matrice de transition est fournie telle que : $P_{t+1}=T*P_t$, P est un vecteur colonne. Pour utiliser l'instruction eigenmarkov, il est nécessaire de considérer le transposé de T pour satisfaire la condition : somme des éléments en ligne = 1. En effet, $P_{t+1}=T*P_t$ (P est un vecteur colonne) est équivalente à $Q_{t+1}=Q_t*R$ avec Q_{t+1} est le transposé de P_{t+1}, Q_t est le transposé de P_t et R est le transposé de T. Q sont des vecteurs ligne. L'opérateur (') permet d'effectuer le transposé d'une matrice. Par exemple le transposé de T qu'on note R est obtenu sous Scilab comme suit : → R=T'

Donc :
T*P0=P0'*T'
P0 est un vecteur colonne et P0' transposé de P0 est un vecteur ligne

A l'état stationnaire, les proportions respectives de C,V,Pe,G est Pi est obtenu par :

```
-->eigenmarkov(T')
 ans  =

    0.1751825    0.1167883    0.2043796    0.1532847    0.3503650
```