INITIATION À L'UTILISATION DU LOGICIEL MATLAB

L'objectif de ce document est de permettre à l'utilisateur d'acquérir quelques notions élémentaires de Matlab (Matrix Laboratory) en lui facilitant l'entrée dans l'univers de ce logiciel. Il ne s'agit donc pas d'être exhaustif, ni même complet, mais simplement de présenter des fonctions de bases qui interviennent dans la simulation de petits modèles et dans les représentations graphiques. Pour remplir cet objectif, le cours doit être relativement succinct. L'utilisateur qui souhaite acquérir une bonne maîtrise des outils présentés ici devra approfondir ses connaissances, notamment avec l'aide en ligne du logiciel. Remarque : même si cela peut sembler inutile, il est fortement recommandé au lecteur débutant de faire des essais sur machine au fur et à mesure de la lecture de ce chapitre, notamment en essayant toutes les fonctions sur des exemples particuliers.

Ce cours débute par une brève description de l'environnement Matlab. Il comprend ensuite une présentation de l'objet de base manipulé par Matlab, à savoir les matrices. Nous donnons alors quelques éléments permettant de tracer les graphiques souhaités par l'utilisateur. La section suivante présente les fonctions puis suit une section sur les solveurs d'équations différentielles. Afin de faciliter l'approfondissement des connaissances, nous présentons rapidement quelques fonctions supplémentaires puis une section d'exercice termine ce chapitre.

1 Les fenêtres usuelles de MatLab

Avant de présenter les instructions et les objets Matlab, nous donnons ici quelques éléments sur l'environnement de travail sous Matlab. Après le démarrage du logiciel, quelques fenêtres apparaissent, ou peuvent être ouvertes à partir du menu déroulant (`view`). Nous introduisons les fenêtres "Répertoire courant" (`current directory`), "Commandes" (`command`) et la fenêtre "Historique" (`command history`).

1.1 Fenêtre "Répertoire courant"

Elle permet de sélectionner le répertoire de travail et de visualiser ce qu'il contient. Avant toute utilisation, le répertoire dans lequel on souhaite travailler doit être sélectionné. Seuls les programmes listés dans ce répertoire (mis à part les programmes installés avec le logiciel) seront reconnus.

1.2 Fenêtre de commandes

Elle permet de saisir des valeurs, de taper des instructions ligne par ligne et d'exécuter des programmes. L'exécution d'un programme se fait directement dans cette fenêtre en tapant le nom du programme puis `ENTREE`.

1.3 Fenêtre "Historique"

Toutes les instructions saisies dans la fenêtre de commandes sont stockées dans une mémoire constituant ce qu'on appelle l'historique. C'est une liste des commandes déjà tapées, y compris lors des utilisations antérieures de Matlab. Il est possible de voir cette liste en ouvrant la fenêtre `command history`. Il est également possible, à partir de la fenêtre de commande, d'accéder à une commande déjà tapées et donc stockées dans cette liste d'historiques, en utilisant les flèches du curseur. La flèche vers le haut permet de remonter dans la liste. La flèche vers le bas permet de descendre. Il est possible d'effacer le contenu de cette fenêtre en sélectionnant l'option "`clear command history`" dans le menu déroulant "`Edit`".

2 Programmes en Matlab : les fichiers M

Un programme est une suite d'instructions enregistrées dans un fichier. En Matlab, ces fichiers sont appelés fichiers M et ont tous l'extension '`*.m`'. Un exemple de nom de fichier M est donc : '`toto.m`'. Ces fichiers peuvent être créés avec le logiciel Matlab en sélectionnant "`Nouveau fichier`" dans le menu déroulant "`Fichier`". Ne pas oublier d'enregistrer le fichier après toute modification ! L'exécution du programme se fait en tapant le nom du fichier sans l'extension `*.m` dans la fenêtre de commande.

3 Quelques commandes de base

Dans cette section, nous allons présenter quelques commandes de base pour travailler sur Matlab.

3.1 La fin de ligne de commande ';' (point-virgule)

Le point virgule est un séparateur. Il permet tout d'abord de séparer plusieurs commandes placées sur une même ligne. Il permet également de supprimer l'affichage du résultat d'une commande. Ceci peut s'avérer utile au cours de l'exécution d'un programme : sans le point-virgule à la fin des lignes de code, chaque ligne aboutissant à un résultat conduirait à l'affichage de ce résultat à l'écran, ce qui peut générer beaucoup d'affichages et un ralentissement du programme. Or tout les résultats intermédiaires ne sont généralement pas utiles. L'utilisation du point-virgule évite ce "*sur-affichage*" inutile.

3.2 Commentaire dans un programme : '%' (pourcent)

Dans un programme (=une suite de lignes de commandes), il peut s'avérer utile de commenter certaines lignes. Ces commentaires peuvent permettre à un tiers utilisateur de comprendre le code. Ils peuvent également permettre à l'utilisateur de se remémorer le sens d'une commande. Les commentaires peuvent être introduit dans n'importe quelle ligne de code, après l'insertion du caractère `%` (pourcent). Tout le texte placé après ce caractère dans la ligne de code ne sera plus interprété.

3.3 Effacer le contenu de la mémoire (`clear`)

Il est possible de vider le contenu de la mémoire, et donc de supprimer les valeurs contenues dans les variables, au moyen de la fonction `clear`. Cette fonction, si elle est utilisée seule, efface l'ensemble de toutes les variables. Si elle est suivie du nom d'une variable, seul le contenu de la variable nommée est effacé : `clear x` efface le contenu de la variable `x`.

3.4 Stocker des résultats ou des données dans un fichier (`save`)

Lorsqu'à la fin d'un programme, on veut mettre le contenu d'une variable dans un fichier, pour une éventuelle utilisation ultérieure par exemple, il est possible de le faire avec la fonction `save`. Par exemple, supposons que la variable `X` soit un tableau de 10 lignes et 20 colonnes contenant les résultats d'un calcul que l'on veut conserver. L'instruction : `save 'nom_de_fichier' X -ASCII` permet de mettre le contenu de `X` dans un fichier dont le nom sera `nom_de_fichier`. Ce fichier se présentera sous la forme d'un tableau de 10 lignes et 20 colonnes de caractères ASCII, réutilisables par d'autres logiciels.

3.5 Lire des données dans un fichier (`load`)

De la même manière que précédemment, il peut être utile de lire des données dans un fichier pour faire un traitement dessus, ou simplement pour les représenter graphiquement. L'instruction `load` le permet. La ligne `load nom_de_fichier.ext` met le contenu du fichier `nom_de_fichier.ext` (où `ext` est une extension quelconque) dans la variable `nom_de_fichier`, à condition que le contenu du fichier `nom_de_fichier.ext` soit en caractères ASCII.

3.6 Lire des données dans un fichier (`textread`)

L'instruction textread permet également de lire des données dans un fichier. La ligne `X=textread('nom_de_fichier.ext')` met le contenu du fichier `nom_de_fichier.ext` dans la variable `X`.

4 Introduction au calcul numérique

Les opérations arithmétiques élémentaires sont notées par les signes classiques, `+` pour l'addition, `-` pour la soustraction, `*` pour la multiplication, `/` pour la division, `^` pour l'exponentiation. Le logiciel Matlab distingue les minuscules et les majuscules, de telle sorte que la variable `a` n'est pas la même que la variable `A`. La hiérarchie des opérations est respectée (l'exposant est prioritaire par rapport au produit, le produit est prioritaire par rapport à la somme). Par exemple, la commande `2^3` donne `8` et la commande `(5-3)^2` donne `4`. Les calculs sont exécutés avec une grande précision, mais les

résultats ne sont affichés, de façon standard, qu'avec 4 chiffres après le point décimal. Pour afficher le résultat de façon plus précise, taper `format long`, et pour revenir au format standard, taper `format short`. Dans l'exemple qui suit, `pi` est la notation Matlab pour le nombre π. La saisie de l'instruction `format long` suivie de `pi` donne `3.141592653589793` et la saisie de l'instruction `format short` suivie de `pi` donne `3.1416`. La lettre `i` est utilisée pour noter le nombre complexe i ($i^2 = -1$). Les opérations mentionnées précédemment s'appliquent aux nombres complexes. En outre, on peut également extraire la partie réelle d'un nombre complexe `z` avec l'instruction `real(z)`, sa partie imaginaire est donnée par l'instruction `imag(z)`, l'instruction `conj(z)` donne le complexe conjugué et `norm(z)` donne son module.

4.1 Prendre la partie entière : `fix`

L'instruction `fix` permet d'extraire la partie entière d'un nombre à virgule (décimal). L'instruction `fix(pi)` par exemple, donnera 3. Avec le logiciel Matlab, toutes les opérations réalisables sur les nombres le sont directement sur les vecteurs et matrices. Nous passons donc tout de suite à l'introduction de la manipulation de ces concepts.

5 Vecteurs et Matrices

Nous présentons ici l'objet de base du logiciel Matlab : les *matrices*. Le langage Matlab est appelé "*langage matriciel*". Nous verrons que si ce type de langage nécessite un petit investissement au début de la part de l'utilisateur, il présente de nombreux avantages en termes d'efficacité et d'ergonomie. A partir de cette section, le cours peut aisément être suivi en se mettant sur un ordinateur muni de Matlab et en essayant les instructions présentées.

5.1 Définition : les vecteurs et les matrices

Une matrice est un tableau de nombres. Sur le plan mathématique, on peut voir dans le rappel d'algèbre linéaire qu'une matrice peut représenter une application linéaire. Ceci est important pour comprendre le produit matriciel. Un vecteur est une matrice particulière. Du point de vue des mathématiques, les vecteurs se présentent sous formes de colonnes, mais en

informatique, un vecteur est simplement une suite finie de nombres, mis sous forme de colonnes ou de lignes. Il y a deux façons simples de définir un vecteur en Matlab : 1) `u=[1 2 1 4]` est un vecteur avec 4 coordonnées avec une ligne et quatre colonnes. `v=[1;2;1;4]` est un vecteur avec 4 coordonnées avec 4 lignes et une colonne. On peut transposer un vecteur (c'est-à-dire transformer un vecteur en ligne en vecteur en colonne, et réciproquement, au moyen de l'apostrophe : dans l'exemple précédent, on a `u'=v` et `v'=u`. 2) `u=0:0.2:3` est un vecteur dont les coordonnées vont de 0 à 3 avec un incrément de 0.2 : u=(0,0.2,0.4,0.6,...,2.8,3). Que contient la variable `v=u'` ? C'est le même principe pour les matrices : `A=[1 2 1;2 1 4]` est la matrice :

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 4 \end{pmatrix}$$

Noter dans l'exemple précédent que les colonnes sont séparées par des espaces.

Exercice : Mettre le contenu de la matrice suivante dans la variable `M` :

$$\begin{pmatrix} 1 & 2 & 1 & 3 \\ 4 & 5 & 1 & 4 \\ 1 & 2 & 1 & 1 \end{pmatrix}$$

Exercice : Mettre le contenu de la transposée de la matrice précédente dans la variable `T`.

5.2 Construction de vecteurs "réguliers" : `linespace` et `xmin:dx:xmax`

Pour générer un suite de nombres avec un incrément fixé entre deux de ces nombres, par exemple la suite qui va de 3 à 8 de 0.2 en 0.2, on peut procéder de deux manières différentes. La commande `linspace(2,5,8)` donne une suite de 8 éléments commençant par 2 et terminant par 5, et l'instruction `3:0.2:8` donne un vecteur ligne de nombres partant de 3, avec une incrémentation de 0.2 et s'arrêtant si le terme suivant dépasse 8. La première méthode est utilisée quand on veut une suite de longueur donnée d'avance. La deuxième est utilisée quand on veut un écart défini d'avance entre 2 éléments successifs.

5.3 Matrices de coefficients identiques : `zeros` et `ones`

Il est possible de construire une matrice de taille quelconque contenant des coefficients identiques. Par exemple, `zeros(n,m)` produit une matrice de

`n` lignes et `m` colonnes ne contenant que des 0 et `ones(n,m)` donne une matrice de `n` lignes et `m` colonnes ne contenant que des 1.

Exercice : Que donnera l'instruction `2.3*ones(5,4)` ?

5.4 Extraire une partie d'une matrice

Il est possible d'extraire une partie d'une matrice. Par exemple, si A est une matrice de 12 lignes et 15 colonnes et qu'on souhaite extraire la sous-matrice partant de la ligne 3 jusqu'à la ligne 7 et la colonne 4 jusqu'à la colonne 12, on utilisera l'instruction `A(3:7,4:12)`. De manière générale, l'instruction `A(n1:n2,m1:m2)` extrait la sous-matrice allant de la ligne n_1 jusqu'à la ligne n_2 et la colonne m_1 jusqu'à la colonne m_2. La commande `A(:,2)` permet d'extraire toute les lignes de la colonne 2 et l'instruction `A(3:5,:)` donne la sous-matrice formée de toutes les colonnes de A mais en ne gardant que les éléments des lignes 3 à 5.

Exercice : Que donnera la suite d'instruction : `A=ones(6,6); B=2*ones(8,8); B(2:7,2:7)=A; 2*B`

5.5 Matrice identité : `eye`

L'instruction `eye(n)` permet de construire la matrice identité à n lignes et n colonnes, c'est-à-dire une matrice ne contenant que des 0, sauf sur la diagonale où se trouvent des 1.

5.6 Taille d'un vecteur ou d'une matrice : `length` et `size`

L'instruction `length` permet de connaitre la taille d'un vecteur (ligne ou colonne). Par exemple, si u est un vecteur de 5 coordonnées, l'instruction `length(u)`donne `5`. La taille d'une matrice n'est pas toujours connue à l'avance, par exemple si elle obtenue après lecture dans un fichier de données, ou bien si sa dimension dépend du résultat d'un calcul. L'instruction `size` permet de déterminer la taille d'une matrice. Par exemple `size(A)` est un vecteur d'une ligne et deux colonnes. Sa première coordonnée correspond au nombre de lignes de `A`, sa seconde coordonnée contient le nombre de colonnes de `A`. Si `A` est une matrice de 5 lignes et 10 colonnes, l'instruction `size(A)` donnera le vecteur `[5 10]`. De la même manière, la ligne de commande `[a`

`b]=size(A)` permet de mettre dans la variable `a` le nombre de lignes de la matrice `A` et dans la variable `b` le nombre de colonne de `A`.

5.7 Maximum et minimum d'une collection de nombres : `max` et `min`

Supposons que 10 nombres soient contenus dans le vecteur (ligne ou colonne) `u`. L'instruction `min(u)` permet de connaître le plus petit des 10 nombres. L'instruction `max(u)` donnera le plus grand des 10 nombres. Si `A` est une matrice avec 5 lignes et 6 colonnes par exemple, l'instruction `min(A)` sera un vecteur ligne de 6 colonnes, chacune des 6 coordonnées contiendra le minimum des 5 nombres de contenu dans la colonne de `A` correspondante. L'instruction `max` fonctionne de la même façon. Par exemple, si on considère :

$$A = \begin{pmatrix} 1 & 2 & 1 & 3 \\ 4 & 5 & 1 & 4 \\ 1 & 2 & 1 & 1 \end{pmatrix}$$

alors `min(A)=[1 2 1 1]` et `max(A)=[4 5 1 4]`.

Exercice : que donnera l'instruction `max(min(A))` ? et `min(max(A))` ?

5.8 Somme de deux matrices : `A+B`

Si les deux matrices `A` et `B` sont de mêmes dimensions, leur somme a un sens et donne une matrice de même dimension dont le coefficient de la ligne i et de la colonne j est $a_{ij} + b_{ij}$. La somme des deux matrices est obtenue avec l'instruction `A+B`.

Remarque : l'opération `3+x` ajoute 3 à chaque composante de x, de même l'opération `3*x` multiplie par 3 chaque composante de x.

5.9 Somme et produit d'une collection de nombres : `sum` et `prod`

La syntaxe de ces instructions est semblable à celle des instructions `min` et `max` décrites ci-dessus. Par exemple `sum(u)`, si `u` est un vecteur, donne la somme des coordonnées de `u`, `sum(A)`, si `A` est une matrice, donne un vecteur ligne dont le nombre de coordonnées est égal au nombre de colonnes de `A`, et dont chaque coordonnée est la somme des coordonnées de la colonne de

`A` correspondante. L'instruction `prod` fonctionne de la même manière mais retourne des produits d'éléments à la place de leurs somme.

Exercice : que donnera `sum(A)` où `A` est la matrice donnée ci-dessus (dans le paragraphe décrivant `min` et `max`) ?

5.10 Produit de matrices

Il existe plusieurs manières de réaliser un produit de deux matrices dont les deux principales sont résumées ci-dessous.

5.10.1 Le produit matriciel usuel : `A*B`

Ce produit correspond à la composition des applications linéaires associées aux matrices `A` et `B`. Il aboutit à une matrice dont les coefficients c_{ij} sont donnés par la formule :

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

C'est le produit de matrice au sens mathématique du terme.

5.10.2 Produit scalaire des vecteurs u et v : `u'*v`

Le produit scalaire du vecteur colonne `u` par le vecteur colonne `v` s'obtient par la commande : `u'*v`. Au cas où une métrique particulière est définie et représentée par la matrice `M`, le produit scalaire au sens de la métrique `M` est obtenu avec la commande : `u'*M*v`.

5.10.3 Le produit terme à terme : `A.*B`

Il peut s'avérer utile de pouvoir multiplier les termes de la matrice `A` par ceux de la matrice `B` (nécessairement de même dimension). Par exemple, supposons que la matrice `A` contient les taux de croissance d'une population donnée et à différents endroits et `B` contient les abondances de cette population aux mêmes endroits. Le produit terme à terme sera donc une matrice contenant les taux de production de la population aux endroits considérés. Le produit terme à terme des matrices `A` et `B` s'obtient de la manière suivante : `A.*B` (noter le point placé avant la multiplication).

5.11 Matrice d'éléments aléatoires : `rand` et `randn`

L'instruction `rand` permet d'obtenir un nombre aléatoire entre 0 et 1, selon une loi uniforme. Pour construire une matrice de n lignes et m colonnes contenant des nombres aléatoires entre 0 et 1, on peut utiliser l'instruction `rand(n,m)`. Pour avoir un nombre aléatoire entre 0 et 5, il suffit de taper `5*rand` et pour avoir un nombre aléatoire entre 3et 8, on tapera `3+5*rand`. La fonction `randn` fonctionne de la même manière mais les nombres sont alors tirés selon une loi normale de moyenne 0 et d'écart-type 1. Pour tirer au hasard des nombres selon une loi normale de moyenne 3 et d'écart-type 2 par exemple, on peut utiliser l'instruction `2*randn+3`.

Exercice : Que donnera l'instruction `A=fix(7*rand(4,4)+3)` ?

5.12 Déterminant, trace et éléments propres d'une matrice

Si `A` est une matrice carrée, l'instruction `det(A)` donne son déterminant et `trace(A)` donne sa trace. On peut extraire ses valeurs propres avec l'instruction `eig(A)`. Enfin, pour avoir les valeurs propres et les vecteurs propres (c'est-à-dire la matrice de passage permettant de diagonaliser la matrice), il faut poser `[P,D]=eig(A)`. Dans ce cas, la matrice `P` est la matrice de passage et `D` est la matrice diagonalisée.

5.13 Moyenne et écart-type : `mean` et `std`

Si `A` est une matrice, l'instruction `mean(A)` donne un vecteur ligne dont les coordonnées sont les moyennes des colonnes de la matrice `A`. De la même manière, l'instruction `std(A)` donnera un vecteur ligne dont les coordonnées sont les écarts-types de chacune des colonnes de `A`.

6 Chaines de caractères

Pour définir une chaîne de caractères, on utilise les quotes '. Par exemple, `mot='abcd'` définit une chaine de caractère stockée dans une mémoire appelée mot et contenant 4 caractères. Les informations concernant la manipulation des chaines de caractères peuvent être obtenues dans l'aide par l'index `string` (=chaine de caractères).

7 Programmes

Un programme est un fichier édité avec un éditeur de texte (évidemment, il y en a un qui est disponible sous Matlab) et dont le nom se termine par l'extension `.m`. Ce fichier contient une liste d'instructions qui seront effectuées dans leur ordre d'apparition dans la liste dès l'exécution du programme. L'exécution d'un programme peut se faire de plusieurs manières, par exemple en tapant le nom du fichier dans la fenêtre de commande ou en sélectionnant la fenêtre contenant le programme et en tapant F5. Si le programme suivant est enregistré dans le fichier `monprogramme.m`, il suffit de taper `monprogramme` dans la fenêtre de commande pour l'exécuter.

Exemple :

```
clear;
x=3;
y=5;
x+y
```

8 Graphiques

Cette section présente quelques instructions graphiques usuelles.

8.1 Les figures : `figure`

Les graphiques se mettent dans des fenêtres appelées "figures". Lorsque plusieurs graphiques sont tracés, ils peuvent être mis dans des fenêtres différentes, les figures sont alors numérotées. Par défaut, lorsqu'une commande lance un graphique, il se place dans la dernière figure active (sélectionnée) et écrase le graphique qui s'y trouvait. Si aucune figure n'est ouverte, le fait de lancer un graphique ouvre une figure automatiquement. Si l'utilisateur souhaite que le graphique se mette dans la figure numéro 3 (même si aucune autre figure n'existe), il suffit qu'il utilise la commande : `figure(3)`.

8.2 Le tracé de courbes dans le plan : `plot`

La fonction plot permet de placer des points de coordonnées $(x; y)$ sur un graphique. Nous illustrons diverses possibilités à travers quelques exemples.

Exemple 1 :

x=[1 2 3 4]; % définit le vecteur ligne de coordonnées $(1,2,3,4)$ et le nomme x
y=[5 1 2 1]; % définit le vecteur ligne de coordonnées $(5,1,2,1)$ et le nomme y
plot(x,y) % trace y en fonction de x et joint les points (x,y) par des segments de droites.

Exemple 2 :

x=[1 2 3 4]; % définit le vecteur ligne de coordonnées $(1,2,3,4)$ et le nomme x
y=[5 1 2 1]; % définit le vecteur ligne de coordonnées $(5,1,2,1)$ et le nomme y
plot(x,y,'+') % Positionne les points de coordonnées (x,y) définis ci-dessus au moyen du symbole '+'. D'autres symboles peuvent être utilisés (+,.,o).

Exemple 3 :

x=-5:0.1:5; % définit le vecteur ligne dont la première coordonnées est -5, la dernière est 5 et la différence entre deux coordonnées successives est 0.1
y=sin(x); % définit le vecteur y dont les coordonnées sont les sinus des coordonnées de x.
plot(x,y,'--') % trace le graphe de la fonction sinus entre -5 et 5, en pointillé.

Exemple 4 :

x=-5:0.1:5; % définit le vecteur ligne dont la première coordonnées est -5, la dernière est 5 et la différence entre deux coordonnées successives est 0.1
y=sin(x); % définit le vecteur y dont les coordonnées sont les sinus des coordonnées de x.
z=cos(x); % définit le vecteur z dont les coordonnées sont les cosinus des coordonnées de x.
plot(x,y,'r',x,z,'b') % trace le graphe de la fonction sinus entre -5 et 5, en rouge et sur le même graphique, trace le graphe de la fonction cosinus entre -5 et 5 en bleu.

8.3 Plusieurs graphiques sur la même figure : subplot

Il est possible de tracer plusieurs graphiques sur la même figure au moyen de l'instruction `subplot` (sous-graphique). Un exemple est donné ci-dessous

pour illustrer le fonctionnement de cette instruction.
Exemple :

```
x=-5:0.1:5; % définit le vecteur ligne dont la première coordonnées
est -5, la dernière est 5 et la différence entre deux coordonnées
successives est 0.1
y=sin(x); % définit le vecteur y dont les coordonnées sont les sinus
des coordonnées de x.
z=cos(x); % définit le vecteur z dont les coordonnées sont les cosinus
des coordonnées de x.
subplot(2,1,1) % définit deux sous-graphiques (2 lignes de graphiques
et 1 colonne) et sélectionne le 1er des deux sous-graphiques.
plot(x,y,'r') % trace le graphe de la fonction sinus entre -5 et
5, en rouge.
subplot(2,1,2) % définit deux sous-graphiques (2 lignes de graphiques
et 1 colonne) et sélectionne le 2ème des deux sous-graphiques.
plot(x,z,'g') % trace le graphe de la fonction cosinus entre -5 et
5 en vert.
```

8.4 Le tracé de courbes en dimension 3 : `plot3`

Pour tracer une courbe paramétrée avec 3 coordonnées, on utilisera l'instruction `plot3`. L'instruction `grid on` permettra de placer un quadrillage utile pour se repérer en dimension 3. L'instruction `grid off` permet de supprimer le quadrillage.
Exemple :

```
u=-10:0.1:10; % définit le vecteur ligne dont la première coordonnées
est -10, la dernière est 10 et la différence entre deux coordonnées
successives est 0.1
x=sin(u); % définit le vecteur x dont les coordonnées sont les sinus
des coordonnées de u.
y=cos(u); % définit le vecteur y dont les coordonnées sont les cosinus
des coordonnées de u.
z=u; % définit le vecteur z dont les coordonnées sont celles de u.
plot3(x,y,z); % trace la courbe en trois dimensions.
```

8.5 Les images : `image` et `imagesc`

Une image pixelisée peut être vue de manière schématique comme une matrice dans laquel chaque élément a une valeur correspondant à une couleur donnée. Pour représenter l'image contenue dans une matrice A, on peut utiliser l'instruction `image(A)`. L'instruction `image(x,y,A)` permet de spécifier les échelles des axes en prenant les valeurs contenues dans les vecteurs x et y. Avec l'instruction `image`, les valeurs numériques des éléments correspondent à des couleurs bien spécifiques (voir l'aide en ligne). Si l'on souhaite utiliser toutes la palette de couleur disponible sans modifier le contenu de la matrice, on utilise l'instruction `imagesc`. Les lettres 'sc' ajoutées correspondent à l'abbréviation de '*scaling*', qui signifie 'renormaliser'. Cela implique que la plus petite valeur du tableau A sera de la couleur correspondant au début de la palette de couleurs et la plus grande valeur du tableau A sera de la couleur correspondant à la fin de la gamme de couleurs. La palette de couleur correspond sans renormalisation à des nombres compris entre 0 et 255.

8.6 Les surfaces : `surf, surfc et meshgrid`

Une fonction de deux variables peut être représentée par un graphe : ce graphe est une surface, à chaque couple (x, y) est associé un nombre z. L'instruction `surf` a une syntaxe similaire à celle décrite pour l'instruction `image` ci-dessus. Attention, la dimension de z est liée à celles de x et de y. Le nombre de lignes de x et de z doivent être identiques, le nombre de lignes de y doit être égal au nombre de colonnes de z.
Exemple :

```
x=linspace(-5,5,100);
y=linspace(-5,5,100);
z=cos(x)'*sin(y);
surf(x,y,z);
```

L'instruction `surfc` a les mêmes fonctionnalités que la fonction `surf` décrite précédemment mais elle trace également les courbes de niveau, le lecteur pourra regarder ce que fait la liste d'instructions précédente dans laquelle on remplacera l'instuction `surf` par `surfc`. L'instruction `[X,Y]=meshgrid(x,y)`, où x et y sont des vecteurs, permet de construire deux matrices X et Y telles que X contient les coordonnées de x autant de fois que le nombre de coordonnées de y et Y contient les coordonnées de y autant de fois que le nombre de coordonnées de x. Les matrices X et Y peuvent alors être utilisées pour évaluer les images du couple (x, y) et donnant une matrice qui possède la

dimension adéquate pour utiliser ensuite la fonction `surf`. Par exemple, la liste d'instructions suivante illustre l'utilisation de `meshgrid` :
Exemple :

```
x=linspace(-5,5,100);
y=linspace(-5,5,100);
[X,Y]=meshgrid(x,y);
Z=cos(X).*sin(Y);
surf(X,Y,Z);
```

8.7 Titres de figures et étiquettes sur les axes :

L'instruction `title('écrire le titre de la figure ici')` permet de mettre un titre sur une figure. L'instruction `xlabel('écrire l"etiquette de l"axe des abscisses ici')` permet de mettre une étiquette sur l'axe des abscisses. L'instruction `ylabel('écrire l"etiquette de l"axe des abscisses ici')` permet de mettre une étiquette sur l'axe des ordonnées. L'instruction `zlabel('écrire l"etiquette de l"axe des cotes ici')` permet de mettre une étiquette sur l'axe des cotes.

8.8 Compléter un graphique : `hold on`

Losqu'on trace un graphique dans une figure, celui-ci prend la place des éventuels graphiques déjà présents dans la figure. Si on souhaite ajouter une courbe à un graphiqe déjà existant par exemple, il faut utiliser l'instruction `hold on`. Celle-ci permet de laisser un graphique 'actif', c'est-à-dire que les instructions graphiques qui suivent seront réalisées sur le même graphique. Pour désactiver le graphique, on utilise l'instruction `hold off`. Dans l'exemple suivant, on trace d'abord le graphe de la fonction sin entre -5 et 5, puis on trace sur le même graphique le graphe de la fonction cos.
Exemple :

```
x=-5:0.1:5;
plot(x,sin(x));
hold on
plot(x,cos(x))
hold off
```

9 Les fonctions en Matlab

9.1 Les fonctions usuelles

Toutes les fonctions numériques usuelles (logarithme, exponentielle, fonctions trigonométriques, ...) sont disponibles et s'appliquent directement sur les matrices. L'image d'une matrice par une application usuelle est une matrice de même taille dont les coefficients sont les images respectives des éléments de la matrice initiale par la fonction. Ainsi `sin(x)`, `cos(x)`, `tan(x)` donnent le sinus, le cosinus, la tangente de `x`, respectivement et `asin(x)`, `acos(x)`, `atan(x)` donnent les fonctions trigonométriques inverses. `log(x)` et `exp(x)` donnent le logarithme et l'exponentielle de `x` respectivement.
Exercice : Tracer le graphique de la fonction cos entre -2πet 2π.

9.2 Définir de nouvelles fonctions : `function`

Il est possible de construire de nouvelle fonction à l'aide de l'instruction `function`. Une fonction se développe dans un fichier différent. A partir d'arguments (entrée), elle détermine des variables de sortie. Programmons par exemple la fonction $y = \frac{x^2}{2+\sin(x)}$ définie sur $\mathbb{R}$. Dans cet exemple, x est la variable d'entrée (argument) et y est la variable de sortie. Pour cela, dans un fichier vide, on tape la liste d'instructions suivantes :
`function y=mafonction(x);`
`y=x.^2./(2+sin(x));` Il faut sauvegarder le fichier et il est recommandé de nommer le fichier de la même manière que la fonction, dans notre exemple le fichier se nommerait `mafonction.m`. Cette fonction peut être ensuite appelée par un programme ou directement dans la ligne de commande. Par exemple, `mafonction(2)` donnera `1.3749`. Il faut noter le point '.' placé avant les opérateurs dans la fonction précédente. Cela permet d'appliquer directement la fonction à un vecteur ou à une matrice. L'exemple suivant montre comment utiliser cette possibilité pour tracer le graphe de notre fonction `mafonction.m`.
Exemple :

```
x=-10:0.1:10;
y=mafonction(x);
figure(1)
plot(x,y)
```

Il faut savoir que toutes les variables informatiques qu'on utilise dans une fonction sont locales, sauf les variables en argument et les variables

de sortie. Dans l'exemple suivant, la variable a est locale :
Exemple :
```
function y=mafonction(x);
a=1;
y=x.^2./(2+sin(x))+a;
```
Cela signifie que sa valeur n'est donnée que dans le cadre de la fonction et qu'en dehors de cette fonction, la variable `a` est différente. Cela permet de définir des fonctions sans se préoccuper de ne pas utiliser des variables déjà utilisées par ailleurs. Par contre, dans certains cas, il peut être utile d'avoir des variables globales, c'est - à - dire qui prennent la même valeur quel que soit l'endroit où elles sont utilisées (dans la fonction et dans le programme principal, ou dans différentes fonctions par exemple). L'instruction `global` permet de définir des variables globales. Supposons que la variable `a` est été affectée de la valeur 3 et soit définie dans un programme principal par exemple comme une variable globale. Dans l'exemple suivant, la variable `a` vaut encore 3 :
```
function y=mafonction(x);
global a
y=x.^2./(2+sin(x))+a;
```

10 Résolution numérique de systèmes d'équations différentielles ordinaires

Comme cela est expliqué dans les premiers chapitres de cet ouvrage, il est généralement impossible de résoudre les systèmes différentiels issus de la modélisation en écologie. Les simulations numériques s'offrent alors comme une alternative intéressante pour le modélisateur. Cependant, cela nécessite un certains nombres de précautions qu'étudie la branche des mathématiques appelée "analyse numérique des équations différentielles". Certaines méthodes numériques courantes ont été programmées et sont fournies sous forme de solveurs de systèmes d'équations différentielles sous Matlab. Les solveurs sont appelés `odeXX` où `ode` est l'abbréviation de Ordinary Differential Equations et `XX` est un nombre qui va impliqué une méthode numérique particulière. Par exemple `ode45` est le solveur qui utilise une méthode appelée Runge-Kutta d'ordre 4 et qui est très fréquente. Même si cette méthode est fréquente, elle a ses limites et certains systèmes ne peuvent pas être simulés avec ce solveur. Nous décrivons l'utilisation de `ode45`, les autres solveurs s'utilisent de

la même manière et le lecteur pourra étudier les différentes possibilités sur l'aide en ligne. Il est évidemment possible de fabriquer d'autres solveurs. Les solveurs sont des fonctions Matlab qui ont (au moins) trois arguments : le nom de la fonction contenant les équations à simuler, les bornes du temps de simulation et les conditions initiales. En sortie, ils donnent un vecteur de temps où les calculs ont été effectués par discrétisation et autant de vecteurs que de variables d'état, chacun d'eux ayant pour coordonnées les valeurs de la variable d'état correspondante aux différents temps de calcul. Nous allons montrer comment simuler le système différentiel :

$$\begin{aligned} \frac{dx}{dt} &= rx\left(1-\frac{x}{K}\right)-axy \\ \frac{dy}{dt} &= eaxy-my \end{aligned}$$

Pour faire la simulation de ce système, il faut deux fichiers : l'un contient le programme principal dans lequel on fixe les conditions de simulation (durée, conditions initiales, valeurs des paramètres, etc.) et le second contient une fonction qui représente le système différentiel. Les valeurs des paramètres peuvent être entrées soit dans le programme principal, soit dans la fonction. La première option permet de mettre toutes les informations numériques dans le programme principal. Par contre, elle nécessite l'utilisation de l'instruction global pour que la fonction contenant le modèle connaisse les valeurs numériques des paramètres. Le programme principal peut être :

```
clear all;    %Efface le contenu des variables
tmax=50;    %Définit la durée de la simulation
x0=1;        %Fixe la condition initiale de la variable x
y0=1;        %Fixe la condition initiale de la variable y
global r K a e m    % Définit les variables contenant les valeurs
des paramètres du système comme des variables globales
r=1;K=2;a=1;e=0.5;m=0.1;    %Fixe les valeurs des paramètres
[t u]=ode45(@ModelPredProie,[0 tmax],[x0 y0]); %Exécute la fonction
de simulation du système
x=u(:,1);y=u(:,2);    %Renomme les variables de sorties correspondant
aux variables d'état
plot(x,y) % Trace la trajectoire issue de (x0,y0) dans l'espace des
phases.
```

On remarque que la fonction `ode45` fait appel à la fonction `ModelPredProie` qui est contenue dans le fichier `ModelPredProie.m` décrit ci-dessous :

```
function du=ModelPredProie(t,u);
du=zeros(2,1);
global r K a e m
x=u(1);y=u(2);
dx=r*x*(1-x/K)-a*x*y;
dy=e*a*x*y-m*y;
du=[dx;dy];
```

Une fois les deux fichiers ci-dessus enregistrés, l'exécution du programme principal permet d'obtenir la trajectoire du système issue de la condition initiale $(x_0, y_0) = (1, 1)$ dans l'espace des phases.

11 Les fichiers : lecture et écriture - Sauvegardes

Nous ne décrirons ici que le cas de fichiers simples (ASCII), par exemple des tableaux de chiffres ou des fichiers de texte. Supposons que le fichier mesdonnees.txt contienne un tableau de 10 lignes et 5 colonnes de chiffres. L'instruction `X=textread('mesdonnees.txt')` construit une matrice `X` à 10 lignes et 5 colonnes contenant les données du fichiers. L'instruction `X=load('mesdonnees.txt')` fait la même chose. Si maintenant on souhaite enregistrer les données de la matrice `Y` dans un fichier nommé `resultats.txt`, on pourra utiliser l'instruction `save 'resultats.txt' Y -ascii`. Tous les fichiers sont dans le répertoire courant.

12 Conclusion

De nombreuses possibilités n'ont pas été abordées dans cette annexe, qui vise juste à donner quelques éléments de base pour entrer dans l'environnement du logiciel Matlab et simuler quelques modèles. On citera ici rapidement les possibilités de réaliser des boucles dans les programmes pour réaliser un grand nombre de fois une liste d'instruction (voir les boucles for - end par exemple) ou la possibilité de tester des conditions logiques (if - end), ce qui permet de n'exécuter des parties d'un programme que sous certaines conditions. Le lecteur intéressé devra consulter l'aide en ligne. De manière générale, les instructions que nous avons présentées ici l'ont été très brièvement et de

nombreuses potentialités sont offertes à l'utilisateur pour affiner leur utilisation.