

Cours de Modélisation et Programmation

Cyrille Bertelle et Rodolphe Charrier
LITIS
UFR Sciences et Techniques Le Havre
25, rue Philippe Lebon - BP 1123
76063 Le Havre Cedex
cyrille.bertelle@univ-lehavre.fr ; rodolphe.charrier@univ-lehavre.fr

16 décembre 2016

Table des matières

1 Modélisation : définition, classification et techniques	3
1.1 Slides	3
1.2 Des outils et assistants numériques pour modéliser et/ou exploiter des modèles	3
2 Introduction à Octave	5
2.1 Aperçu et environnement	5
2.1.1 Comment obtenir de l'aide ?	6
2.1.2 Mode d'utilisation - console, éditeur, batch processing	8
2.2 Eléments de base	10
2.3 La manipulation de matrices et vecteurs	14
2.4 Visualiser un graphe simple	17
2.5 Ecrire et exécuter un script	18
2.6 Entrées-sorties et fichiers	19
2.6.1 Lecture par le clavier et affichage	19
2.6.2 Utilisation de fichiers	20
2.7 La programmation sous Octave	22
2.7.1 Branchements et boucles	23
2.7.2 Les fonctions	25
2.8 Graphisme	30
2.8.1 Compléments	30
2.9 Modélisation et systèmes différentiels	31
3 Modèles d'évolution des populations	35
3.1 Un aperçu des systèmes	35
3.2 Description de la dynamique de population	36
3.2.1 Définitions	36
3.2.2 Observations	36
3.3 Les modèles continus de croissance déterministe	36
3.3.1 Modèle de Malthus et loi exponentielle	36
3.3.2 Modèles de Verhulst et loi logistique	38
3.3.3 Loi de Gompertz	41

TABLE DES MATIÈRES

3.3.4	TP : représentation graphique de modèles différentiels avec Scilab	41
3.4	Identification et validation d'un modèle	41
3.4.1	Identification et ajustement d'un modèle	41
3.4.2	Validation d'un modèle	46
3.5	Modèles discrets de dynamique de population	54
3.5.1	Construction d'un modèle discret et interprétation - application à la loi exponentielle	54
3.5.2	Analyse d'un modèle discret - application à la loi logistique	55
3.5.3	TP : Etude de modèles discrétilisés avec Scilab, à partir de la loi logistique	57
4	Modèles d'interaction de populations	58
4.1	Présentation des différents types d'interactions étudiées	58
4.2	Modèle de coopération de populations	59
4.3	Modèle de compétition de populations	66
4.4	Modèle de type proies-prédateurs	72
4.4.1	TP : Implémentation des modèles d'interaction de population avec Scilab	72
4.5	Modèles d'épidémiologie	77
4.5.1	Modèles SI	77
4.5.2	Modèles SIR	78
4.5.3	TP : Applications sous Populus	79
5	Modèles individus-centrés et application sous Netlogo	80
5.1	Introduction aux approches systémiques et à la modélisation comportementale	80
5.2	TP : Introduction et implémentation en Netlogo	80
6	Modélisation par conception graphique	81
6.1	Modèles compartimentaux	81
6.2	TP : Diagrammes de Forrester et implémentation avec le module "systèmes dynamiques" de Netlogo	81

Chapitre 1

Modélisation : définition, classification et techniques

Sommaire

1.1 Slides	3
1.2 Des outils et assistants numériques pour modéliser et/ou exploiter des modèles	3

1.1 Slides

Inclure pdf de présentation :

1. Qu'est ce qu'un modèle ?
2. Modélisation mathématique
3. Classification des modèles traités dans ce cours

1.2 Des outils et assistants numériques pour modéliser et/ou exploiter des modèles

L'objectif de ce cours consiste aussi à étudier un certain nombre d'outils informatiques qui s'avèrent être très utiles dans le cadre de la démarche de modélisation. Nous en dressons une liste ci-après, qui ne se veut pas exhaustive :

- Les langages de programmation qui proposent des approches plus ou moins structurées. On notera par exemple, les langages populaires aujourd'hui et basés sur l'approche orientée objet.
Leurs caractéristiques est d'être très généraux et seule l'imagination est la limite du modélisateur programmeur. Ils ont l'inconvénient de nécessiter un temps d'apprentissage relativement long ce qui en exclut leur étude dans le

CHAPITRE 1. MODÉLISATION : DÉFINITION, CLASSIFICATION ET TECHNIQUES

cadre de ce cours (mais vous êtes sollicités pour suivre leur apprentissage par ailleurs).

- Les outils dits de “calcul avancé” tels que Matlab, Octave, Scilab, Maple, Mathematica, Maxima, Mupad, Sage, Python ...
Ce sont encore des outils généraux qui permettent de manipuler facilement (et donc avec un apprentissage beaucoup plus rapide) des calculs numériques et différentiels, en se rapprochant d'une démarche et notation mathématique standard. On étudiera Octave dans le cadre de ce cours, en raison de son accès et diffusion libre et de sa grande proximité avec Matlab qui fait souvent office de référence.
- Les plateformes de calcul dédiées à des modèles déjà programmés et sur lesquels l'utilisateur a la faculté de jouer sur les valeurs de certains paramètres, ainsi que de visualiser des graphiques. Dans le cadre d'une exploration plus approfondie du contenu de ce cours, notamment sur la partie dynamique de population, on pourra consulté la plateforme "Populus"¹ qui présente une très grande variété de modèles de ce type.
- Les plateformes de modèles individus centrées qui permettent de décrire des mondes artificiels dans lesquels on décrit les comportements des entités qui les peuplent. On aborde ici les modèles dits discrets. Dans le cadre de ce cours, on utilisera la plateforme Netlogo.

Dans le chapitre suivant, on va découvrir l'outil de calcul Octave. Grâce à lui, on pourra résoudre facilement des modèles différentiels et tracer notamment des graphiques de leurs solutions. Cet outil sera largement exploité dans un certain nombre de chapitre qui suivront.

1. Populus de Don Alstad, est un logiciel développé et mis à disposition par le College of Biological Sciences de l'Université du Minnesota : <http://cbs.umn.edu/populus/overview>

Chapitre 2

Introduction à Octave

Sommaire

2.1	Aperçu et environnement	5
2.1.1	Comment obtenir de l'aide ?	6
2.1.2	Mode d'utilisation - console, éditeur, batch processing	8
2.2	Eléments de base	10
2.3	La manipulation de matrices et vecteurs	14
2.4	Visualiser un graphe simple	17
2.5	Ecrire et exécuter un script	18
2.6	Entrées-sorties et fichiers	19
2.6.1	Lecture par le clavier et affichage	19
2.6.2	Utilisation de fichiers	20
2.7	La programmation sous Octave	22
2.7.1	Branchements et boucles	23
2.7.2	Les fonctions	25
2.8	Graphisme	30
2.8.1	Compléments	30
2.9	Modélisation et systèmes différentiels	31

2.1 Aperçu et environnement

GNU **Octave** est le logiciel libre offrant actuellement la meilleure compatibilité avec le logiciel commercial de calcul numérique **Matlab**. C'est donc un logiciel de calcul scientifique, de visualisation et de programmation performant dont les dernières versions sont 100% compatibles dans Matlab (la branche de développement actuelle est la 3.6 et la version que vous allez utiliser pour ce cours sous linux fait partie de la branche 3.2).

Tout comme Matlab, Octave est un langage interprété et fonctionne à la fois en ligne de commande et par exécution de scripts stockés dans des "M-files" (extension ".m"). Ce langage est interprété et converti dans un langage plus bas niveau, le P-code (pseudo code Matlab). Il est également interfaçable avec les langages C, C++.

Octave est fourni ici avec l'IDE QtOctave qui permet l'édition de scripts, leur exécution et débogage, les requêtes en ligne de commande, l'exploration locale des fichiers, l'aide en ligne, des menus de fonctions dédiées, etc ... Ces deux logiciels sont librement distribués et multi-plateformes. Octave peut être complété par un certain nombres de paquets complémentaires ("Packages" équivalents des Tool-boxes de Matlab) dédiés à certains domaines spécifiques du calcul numérique. Ce sont des archives de M-files qui peuvent être installés depuis QtOctave ou en ligne de commande par la commande **pkg**. L'ensemble de ces paquets peut être téléchargé depuis <http://octave.sourceforge.net/packages.php>.

Il est donc possible d'aborder les méthodes usuelles de haut niveau traitant de :

- algèbre linéaire et matriciel
- polynômes et fonctions rationnelles
- interpolation et approximation
- optimisation linéaire, quadratique et non linéaire
- EDO / EDP
- Contrôle classique et robuste
- Traitement du signal
- Statistiques
- Graphisme 2D/3D
- ...

Le lancement de l'IDE QtOctave (dans la rubrique Applications/Sciences de votre menu Ubuntu) donne un ensemble de fenêtres visibles sur la figure 2.1. Nous trouvons à la fois un terminal de commande pour Octave, un éditeur intégré basique pour écrire ses scripts, un inspecteur de variables, un navigateur de fichiers un historique des commandes. L'éditeur utilisé peut être remplacé par un éditeur externe de votre choix en configurant son chemin d'accès dans le menu **Config/General Configuration**.

2.1.1 Comment obtenir de l'aide ?

Une des manières les plus simples d'obtenir de l'aide avec Octave est de lancer la rubrique Octave Help du menu Help de QtOctave. La documentation complète d'Octave est alors disponible en anglais dans une fenêtre **Help Browser** (cf. figure 2.2). Pour rechercher une fonction particulière utiliser le champ de saisie et cocher "Global search". Cette documentation est également obtenue en tapant "**doc**" dans la ligne de commande de la console.

CHAPITRE 2. INTRODUCTION À OCTAVE

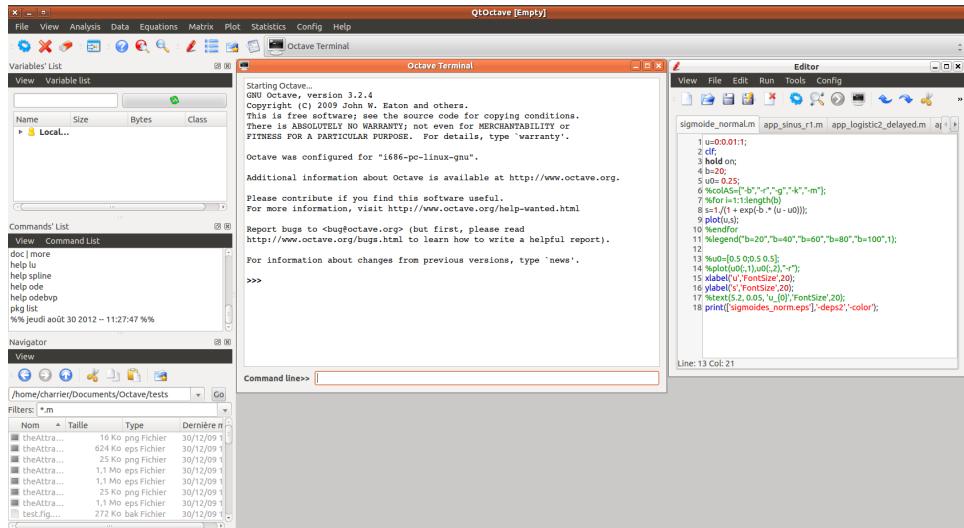


FIGURE 2.1 – Copie d'écran QtOctave



FIGURE 2.2 – QtOctave Help Browser

Pour obtenir de l'aide sur une fonction particulière, par ex. sur *plot*, il suffit simplement de taper dans la console de commande Octave **help plot** ou **doc plot**.

On peut aussi accéder à une importante ressource documentaire sur des versions récentes d'Octave sur la page web du cours de l'EPFL consacrée à ce logiciel à cette adresse :

http://enacit1.epfl.ch/cours_matlab/

Un wiki spécialisé est également accessible sur

http://wiki.octave.org/Main_Page.

Finalement, on pourra aussi accéder à des aides sous la forme d'exemples nombreux de codes dans les fichiers d'aide sur les fonctions disponibles dans Octave.

2.1.2 Mode d'utilisation - console, éditeur, batch processing

Utilisation en mode console

Octave s'utilise principalement par interaction avec la fenêtre console, au travers d'une boucle du type
lecture-évaluation-résultat

Voici une première interaction élémentaire :

```
>>> s="Hello World!"  
s=  
Hello World!  
>>> disp(s)  
Hello World!
```

Les flèches \uparrow et \downarrow permettent une navigation dans l'historique des commandes et la touche <TAB> lance une complétion automatique facilitant la saisie des commandes (cf. figure 2.3).

Une aide dynamique est aussi disponible dans le menu Help de QtOctave. Elle ouvre une fenêtre complémentaire qui fournit des informations intéressantes sur les fonctions possibles.

Exécution de fichiers de scripts / commandes

A partir de l'éditeur intégré ou d'un autre éditeur, on peut créer un script de commandes dans un M-file (extension .m). On peut l'exécuter ensuite

- soit depuis le menu Run/Run de la fenêtre d'édition de QtOctave,
- soit par la commande `source ("nomDuScript.m")` depuis le dossier du script dans la console de commande,
- soit en tapant directement le nom du script (sans extension) depuis le dossier du script dans la console de commande.

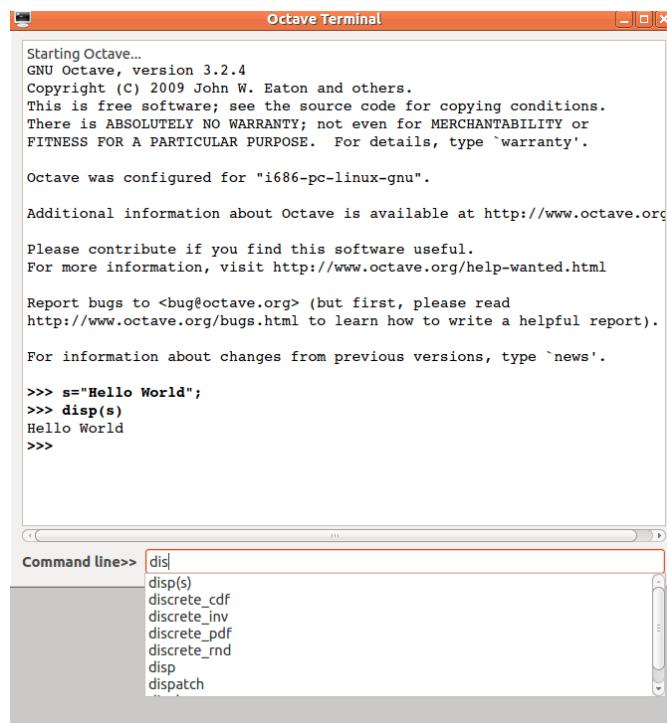


FIGURE 2.3 – Complétion automatique pour Octave

Mode batch

On peut lancer Octave en mode batch soit pour disposer d'un environnement simplifié (sans fenêtrage), soit pour lancer, en externe, des scripts et fichiers .m.

Sous Linux,

- octave lance l'interpréteur de commandes d'Octave,
- octave -silent monScript.m lance le script correspondant sur Octave, sans afficher les messages d'ouverture Octave et quitte automatiquement après exécution. Pour récupérer les résultats dans un fichier, il suffit de terminer la commande précédente par >fichierResultats.txt.

On peut également rendre exécutible un script en rajoutant en en-tête la ligne suivante donnant le chemin d'accès absolu au binaire Octave :

```
#!/usr/bin/octave --silent
```

puis l'exécuter avec la séquence suivante :

```
$ chmod u+x script.m
$ ./script.m > fichierResultats.txt
```

2.2 Eléments de base

Octave est un langage interprété permettant des constructions dynamiques de programmes. Généralement, comme pour *Matlab*, on peut dire que presque tout dans Octave est une matrice (sauf des structures de données moins usuelles ; ce sera notamment des listes, par exemple, mais que l'on sera amené à peu utiliser dans la suite).

Malgré cette remarque, nous allons commencer par décrire des types de données simples (des matrices 1x1) avant de passer aux aspects plus généraux de traitements de matrice.

Types des nombres - réels, complexes, entiers

Réels

Octave manipule par défaut tout nombre comme un réel à virgule flottante en double précision (codés sur 64 bits). Les réels possèdent donc une précision à 16 chiffres décimaux significatifs et permettent de représenter des nombres allant de 10^{-308} à 10^{+308} . Il est cependant possible de réduire cette précision à une simple précision grâce à la fonction de conversion. `single()`.

Remarque 1 *Attention, certaines fonctions ou opérateurs arithmétiques d'Octave mêlant entiers et réels rendent par défaut un entier. Il faut donc forcer la conversion des entiers en double grâce à la fonction `double(entier)` pour obtenir un résultat de type réel.*

Entiers

Les entiers sont obtenus via des fonctions de conversion vers un encodage en 8, 16, 32 ou 64 bits, soit les fonctions suivantes :

- `int8`, `int16`, `int32`, `int64` pour les entiers signés
- `uint8`, `uint16`, `uint32`, `uint64` pour les entiers non signés.

Nombres complexes

Les nombres complexes s'écrivent sous leur forme mathématique habituelle en décomposant la partie réelle et la partie imaginaire avec le nombre `i`, imaginaire pur dont le carré vaut -1.

Les parties réelles et imaginaires sont stockées sous forme de réels en double précision.

Par exemple la fonction `conj()` :

```
>>> conj(5 - 6i)
ans = 5 + 6i
```

retourne le complexe conjugué du nombre passé en paramètre.

Variables réelles ou numériques

Voici une séquence d'instructions illustrant une manipulation très classique des variables numériques. On notera l'opérateur d'affectation représenté par un seul signe = qu'il ne faut pas confondre avec le double symbole == qui désigne l'opérateur booléen de comparaison.

```
>>> x=1  
x= 1  
>>> x=x*2  
x= 2
```

En ajoutant un point-virgule ; à la fin d'une commande, on demande de ne pas générer d'affichage suite à l'exécution de la commande.

- x^2 ou $x * 2$ désigne l'élévation au carré
- $x/y = xy^{-1}$
- $x\y = x^{-1}y$

Noms de variables

Un nom de variable sous Octave commence par une lettre puis par au choix :

- des lettres de l'alphabet en minuscules ou majuscules, avec différenciation de ces deux types ("case sensitive")
- des nombres de 0 à 9
- un caractère _

Le nombre de caractères utilisables est limité à 63 caractères sur Matlab/Octave. Les variables une fois créées résident en mémoire dans le "**Workspace**" d'Octave.

Commentaires et lignes de continuation

Un commentaire est une ligne démarrant par %. Pour commenter un bloc entier il suffit de l'encadrer par %{ et %}

Une ligne se terminant par 3 points successifs . . . sera prolongée par la ligne suivante.

```
>>> % c'est un commentaire  
>>> x = 1 . . .  
>>> + 2 . . .  
>>> + 3  
x= 6
```

Fonctions mathématiques élémentaires

Octave intègre les fonctions mathématiques courantes :

<code>sqrt(var)</code>	Racine carrée de <code>var</code> . Remarque : pour la racine n -ème de <code>var</code> , faire <code>var^(1/n)</code>
<code>exp(var)</code>	Exponentielle de <code>var</code>
<code>log(var)</code> <code>log10(var)</code> <code>log2(var)</code>	Logarithme naturel de <code>var</code> (de base e), respectivement de base 10, et de base 2 Ex: <code>log(exp(1))</code> => 1, <code>log10(1000)</code> => 3, <code>log2(8)</code> => 3
<code>cos(var)</code> et <code>acos(var)</code>	Cosinus, resp. arc cosinus, de <code>var</code> . Angle exprimé en radian
<code>sin(var)</code> et <code>asin(var)</code>	Sinus, resp. arc sinus, de <code>var</code> . Angle exprimé en radian
<code>sec(var)</code> et <code>csc(var)</code>	Sécante, resp. cosécante, de <code>var</code> . Angle exprimé en radian
<code>tan(var)</code> et <code>atan(var)</code>	Tangente, resp. arc tangente, de <code>var</code> . Angle exprimé en radian
<code>cot(var)</code> et <code>acot(var)</code>	Cotangente, resp. arc cotangente, de <code>var</code> . Angle exprimé en radian
<code>atan2(dy,dx)</code>	Angle entre -pi et +pi correspondant à <code>dx</code> et <code>dy</code>
<code>cart2pol(x,y {,z})</code> et <code>pol2cart(zh,r {,z})</code>	Passage de coordonnées cartésiennes en coordonnées polaires, et vice-versa
<code>cosh</code> , <code>acosh</code> , <code>sinh</code> , <code>asinh</code> , <code>sech</code> , <code>asch</code> , <code>tanh</code> , <code>atanh</code> , <code>coth</code> , <code>acoth</code>	Fonctions hyperboliques...
<code>factorial(n)</code>	Factorielle de <code>n</code> (c'est-à-dire : $n \cdot (n-1) \cdot (n-2) \cdots \cdot 1$). La réponse retournée est exacte jusqu'à la précision de 15 chiffres avec un exposant)
<code>rand</code> <code>rand(n)</code> <code>rand(n,m)</code>	Génère un nombre aléatoire compris entre 0.0 et 1.0 Génère une matrice carrée $n \times n$ de nb. aléatoires compris entre 0.0 et 1.0 Génère une matrice $n \times m$ de nb. aléatoires compris entre 0.0 et 1.0
<code>fix(var)</code> <code>round(var)</code> <code>floor(var)</code> <code>ceil(var)</code>	Troncature à l'entier, dans la direction de zéro (donc 4 pour 4.7, et -4 pour -4.7) Arrondi à l'entier le plus proche de <code>var</code> Le plus grand entier qui est inférieur ou égal à <code>var</code> Le plus petit entier plus grand ou égal à <code>var</code> Ex: <code>fix(3.7)</code> et <code>fix(3.3)</code> => 3, <code>fix(-3.7)</code> et <code>fix(-3.3)</code> => -3 <code>round(3.7)</code> => 4, <code>round(3.3)</code> => 3, <code>round(-3.7)</code> => -4, <code>round(-3.3)</code> => -3 <code>floor(3.7)</code> et <code>floor(3.3)</code> => 3, <code>floor(-3.7)</code> et <code>floor(-3.3)</code> => -4 <code>ceil(3.7)</code> et <code>ceil(3.3)</code> => 4, <code>ceil(-3.7)</code> et <code>ceil(-3.3)</code> => -3
<code>mod(var1,var2)</code> <code>rem(var1,var2)</code>	Fonction <code>var1</code> "modulo" <code>var2</code> Reste ("remainder") de la division de <code>var1</code> par <code>var2</code> Remarques: - <code>var1</code> et <code>var2</code> doivent être des scalaires réels ou des tableaux réels de même dimension - <code>rem</code> a le même signe que <code>var1</code> , alors que <code>mod</code> a le même signe que <code>var2</code> - les 2 fonctions retournent le même résultat si <code>var1</code> et <code>var2</code> ont le même signe Ex: <code>mod(3.7, 1)</code> et <code>rem(3.7, 1)</code> retournent 0.7, mais <code>mod(-3.7, 1)</code> retourne 0.3, et <code>rem(-3.7, 1)</code> retourne -0.7
<code>idivide(var1, var2, 'rule')</code>	Division entière. Fonction permettant de définir soi-même la règle d'arrondi. Implémentée depuis Octave 3.6.0
<code>abs(var)</code>	Valeur absolue (positive) de <code>var</code> Ex: <code>abs([3.1 -2.4])</code> retourne [3.1 2.4]
<code>sign(var)</code>	(signe) Retourne "1" si <code>var>0</code> , "0" si <code>var=0</code> et "-1" si <code>var<0</code> Ex: <code>sign([3.1 -2.4 0])</code> retourne [1 -1 0]
<code>real(var)</code> et <code>imag(var)</code>	Partie réelle, resp. imaginaire, de la <code>var</code> complexe

La plupart de ces fonctions prennent un argument et en retournent un autre. Elles sont "vectorisées", c'est à dire qu'elles manipulent des matrices.

Exemple :

```
>>> x = cos(2)
x= -0.4161468
>>> y = sin(2)
y= 0.9092974
>>> x^2 + y^2
ans= 1.
>>> ans*2
ans= 2.
>>> z=sin([x y])
```

CHAPITRE 2. INTRODUCTION À OCTAVE

```
Z =  
-0.40424    0.78907
```

On notera la variable `ans` qui évalue l'expression courante et qui peut être réutilisée.

Variables mathématiques prédéfinies

Dans Matlab/Octave, plusieurs variables mathématiques sont prédéfinies :

Constante	Description
<code>pi</code>	3.14159265358979 (la valeur de "pi")
<code>i ou j</code>	racine de -1 (<code>sqrt(-1)</code>) (nombre imaginaire)
<code>e ou exp(1)</code>	2.71828182845905 (la valeur de "e")
<code>Inf ou inf</code>	infini (par exemple le résultat du calcul <code>5/0</code>)
<code>Nan ou nan</code>	indéterminé (par exemple le résultat du calcul <code>0/0</code>)
<code>NA</code>	valeur manquante
<code>realmin</code>	env. $2.2e^{-308}$: le plus petit nombre positif utilisable (en virgule flottante double précision)
<code>realmax</code>	env. $1.7e^{+308}$: le plus grand nombre positif utilisable (en virgule flottante double précision)
<code>eps</code>	env. $2.2e^{-16}$; c'est la précision relative en virgule flottante double précision (ou le plus petit nombre représentable par l'ordinateur qui est tel que, additionné à un nombre, il crée un nombre juste supérieur)
<code>true</code>	vrai ou <code>1</code> ; mais n'importe quelle valeur différente de <code>0</code> est aussi "vrai" <small>Ex: si <code>nb</code> vaut <code>0</code>, la séquence <code>if nb, disp('vrai'), else, disp('faux'), end</code> retourne "vrai", mais si <code>nb</code> vaut n'importe quelle autre valeur, elle retourne "faux"</small>
<code>false</code>	faux ou <code>0</code>

Booléens

Les booléens sont des variables qui peuvent prendre deux valeurs, `false` ou `0`, et `true` ou `1` (ou dans certains cas de conditions logiques toute valeur différente de `0`) .

Les opérateurs booléens sont listés ci-dessous

Opérateur ou fonction	Description
<code>~ expression</code> <code>not (expression)</code> <code>! expression</code>	Négation logique (rappel: NON <code>0 => 1</code> ; NON <code>1 => 0</code>)
<code>expression1 & expression2</code> <code>and (expression1, expression2)</code>	ET logique. Si les <code>expressions</code> sont des matrices, retourne une matrice (rappel: <code>0 ET 0 => 0</code> ; <code>0 ET 1 => 0</code> ; <code>1 ET 1 => 1</code>)
<code>expression1 && expression2</code>	ET logique "short circuit". A la différence de <code>&</code> ou <code>and</code> , cet opérateur est plus efficace, car il ne prend le temps d'évaluer <code>expression2</code> que si <code>expression1</code> est vraie. En outre: - sous Octave: retourne un scalaire même si les <code>expressions</code> sont des matrices - sous MATLAB: n'accepte pas que les <code>expressions</code> soient des matrices
<code>expression1 expression2</code> <code>or (expression1, expression2)</code>	OU logique. Si les <code>expressions</code> sont des matrices, retourne une matrice (rappel: <code>0 OU 0 => 0</code> ; <code>0 OU 1 => 1</code> ; <code>1 OU 1 => 1</code>)
<code>expression1 expression2</code>	OU logique "short circuit". A la différence de <code> </code> ou <code>or</code> , cet opérateur est plus efficace, car il ne prend le temps d'évaluer <code>expression2</code> que si <code>expression1</code> est fausse. En outre: - sous Octave: retourne un scalaire même si les <code>expressions</code> sont des matrices - sous MATLAB: n'accepte pas que les <code>expressions</code> soient des matrices
<code>xor (expression1, expression2)</code>	OU EXCLUSIF logique (rappel: <code>0 OU EXCL 0 => 0</code> ; <code>0 OU EXCL 1 => 1</code> ; <code>1 OU EXCL 1 => 0</code>) Pour des opérandes binaires, voir les fonctions <code>bitand</code> , <code>bitcmp</code> , <code>bitor</code> , <code>bitxor</code> ...

ainsi que les fonctions logiques suivantes :

Fonction	Description
<code>isfloat(var)</code>	Vrai si la variable var est de type réelle (simple ou double précision), faux sinon (entière, chaîne...) Sous Octave, implémenté depuis la version 3.2.0
<code>isempty(var)</code>	Vrai si la variable var est vide (de dimension 1x0), faux sinon. Notez bien qu'il ne faut pas entourer var d'apostrophes, contrairement à la fonction <code>exist</code> . Ex: si <code>vects5:1</code> ou <code>vects[]</code> , alors <code>isempty(vect)</code> retourne "1"
<code>ischar(var)</code>	Vrai si var est une chaîne de caractères, faux sinon. Ne plus utiliser <code>isstr</code> qui va disparaître.
<code>exist('objet')</code> <code>{,'var builtin file dir')</code>	Vérifie si l'objet spécifié existe. Retourne "1" si c'est une variable, "2" si c'est un M-fille, "3" si c'est un MEX-fille, "4" si c'est un MDL-fille, "5" si c'est une fonction built-in, "6" si c'est un P-fille, "7" si c'est un directoire. Retourne "0" si aucun objet de l'un de ces types n'existe. Notez bien qu'il faut ici entourer objet d'apostrophes, contrairement à la fonction <code>isempty</code> ! Ex: <code>exist('sgprt')</code> retourne "5", <code>exist('axis')</code> retourne "2", <code>exist('variable_inexistante')</code> retourne "0"
<code>isinf(var)</code>	Vrai si la variable var est infinie positive ou négative (<code>Inf</code> ou <code>-Inf</code>)
<code>isnan(var)</code>	Vrai si la variable var est indéterminée (<code>NaN</code>)
<code>isfinite(var)</code>	Vrai si la variable var n'est ni infinie ni indéterminée Ex: <code>isfinite([0/0 NaN 4/0 pi -Inf])</code> retourne [0 0 1 0]

Chaînes de caractères

Elles sont délimitées par des apostrophes :

```
>>> x = 'coucou'
x = coucou
```

Une chaîne de caractères est un vecteur ligne ce qui explique que la concaténation se fasse par la mise entre crochets [].

Ainsi la commande :

```
>>> ['bonjour ' 'tout le monde']
ans = bonjour tout le monde
```

respecte les espaces blancs contrairement à la fonction `strcat(chaine1, chaine2, ...)` qui les supprime. Il existe de nombreuses fonctions de manipulation de chaînes de caractères que l'on peut découvrir et apprendre avec le systèmes d'aide, les démos de la documentation Octave.

Une spécificité d'Octave par rapport à Matlab est d'accepter également les chaînes déclarées entre guillemets intégrant des caractères spéciaux :

- \t pour la tabulation
- \n pour le passage à la ligne.

Ainsi par exemple :

```
>>> disp("Ce texte \n s'affiche sur 2 lignes avec une \t tabulation")
Ce texte
    s'affiche sur 2 lignes avec une    tabulation
```

2.3 La manipulation de matrices et vecteurs

Dans cette section nous donnons les bases concernant la description des vecteurs et matrice puis l'apprentissage et la maîtrise du calcul matriciel se poursuit par l'intermédiaire d'exercices. Il est fortement recommandé d'utiliser l'aide en ligne d'Octave et les sources documentaires données en début de chapitre.

Elément de base sur la plupart des outils de calcul numériques Matlab/Octave/Scilab et autres, une matrice peut contenir des nombres réels ou complexes. On peut la définir en respectant les règles de constructions suivantes :

- Elle commence par un crochet ouvrant [et se termine par un crochet fermant]
- Les éléments d'une même ligne sont séparés par des blancs ou des virgules
- Les lignes sont séparées par des points virgules ;

Ci-dessous un exemple d'une matrice carrée de dimension (3, 3) (renvoyé par la fonction `size()`), suivi d'un vecteur ligne comportant des nombres complexes, puis le même vecteur transposé en vecteur colonne grâce à l'opérateur apostrophe `'`, ainsi que la matrice transposée de A. Enfin, la récupération d'un élément de matrice se fait avec des parenthèses comportant la ligne et la colonne de l'élément, ou avec l'opérateur `:` qui permet de prendre en compte une série de lignes ou de colonnes. Le dernier exemple montre l'utilisation de l'opérateur `:` pour générer des vecteurs lignes par série linéaires, puis assemblées pour former une matrice. La fonction `linspace(début, fin, nbval)` permet aussi de générer un vecteur ligne de `nbval` valeurs linéairement espacées entre la valeur `début` et `fin`.

```
>>> A = [ 1 1 1; 2 4 8; 3 9 27]
A =
1     1     1
2     4     8
3     9    27
>>> size(A) % renvoie la taille de A
ans =
3     3
>>> y = [1 + i, - 3 * i, -1]
y =
1 + 1i  -0 - 3i  -1 + 0i
>>> y'      % vecteur transposé
ans =
1 - 1i
-0 + 3i
-1 - 0i
>>> A' % matrice transposée de A
ans =
1     2     3
1     4     9
1     8    27
>>> A(2,2)
ans = 4
```

```
>>> A(2,:) %renvoie la deuxième ligne de A
ans =
2     4     8
>>>A(2:3,:)%renvoie les lignes 2 et 3 de A
ans =
2     4     8
3     9    27
>>>B=[1:1:3 ; 1:3:9 ; 9:-1:7] % génération d'une matrice par 3 vecteurs
B =
1     2     3
1     4     7
9     8     7
>>>v=linspace(-1,3,5)
ans =
-1     0     1     2     3
```

Des matrices ou vecteurs spécifiques sont prédéfinis. Nous donnons ici les principaux (à tester sur vos postes) :

- Construction d'une matrice identité
»> I = eye(4,4)
- Construction d'une matrice diagonale à partir d'un vecteur
»> Y = diag(y)
- Extraction de la diagonale d'une matrice
»> a = diag(A)
- Construction d'une matrice de zéros (taille de la matrice en entrée)
»> I = zeros(3,4)
- idem pour une matrice de uns (taille de la matrice en entrée)
»> I = ones(3,4)

Exercice 1 *Tester les instructions/constructions suivantes et comprendre leur fonctionnalité*

- ones (3,4)
- zeros(3,4)
- triu(A)
- tril(A)
- rand (3,4)

2.4 Visualiser un graphe simple

On va construire le graphe de la fonction

$$y = e^{-x} \sin(4x) \text{ pour } x \in [0, 2\pi]$$

On commence alors par créer un maillage de l'intervalle $[0, 2\pi]$ avec la fonction `linspace` (cf. section précédente) :

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

x est alors un vecteur de 101 valeurs pour donner 100 intervalles.

On calcule ensuite les valeurs de la fonction pour chaque composante du vecteur *x*, ce qui se fait en une seule instruction puisque Octave manipule "naturellement" des vecteurs et matrices :

```
>>>y=exp(-x).*sin(4*x);
```

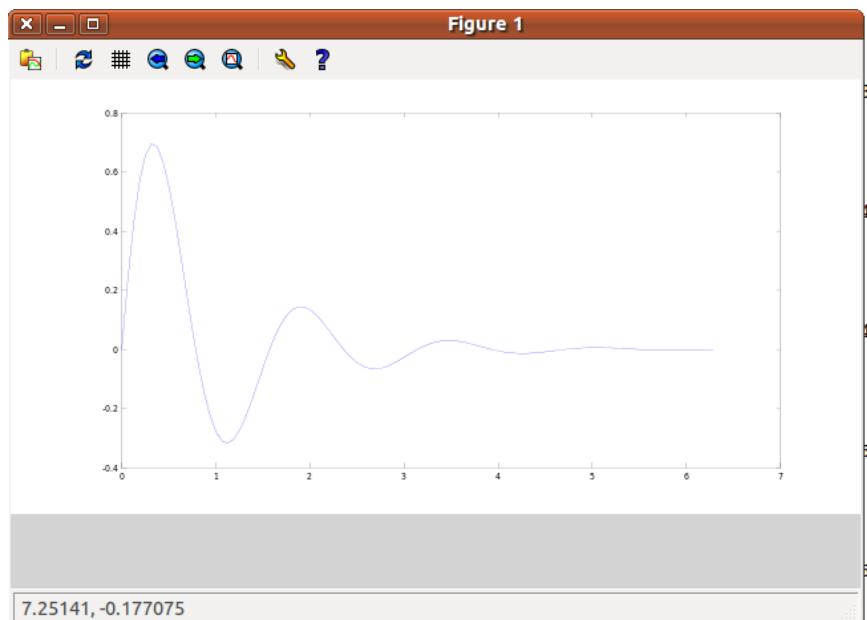
Noter ici la notation pointée de l'opérateur arithmétique qui effectue l'opération pointée élément par élément. Cette syntaxe est particulièrement utile sur les matrices, et simplifie grandement l'écriture algorithmique. Le calcul est également optimisé dans ce cas.

Le tracé du graphique se fait alors par l'instruction `plot`. Elle est suivie ici par l'instruction `title` qui permet d'afficher un titre :

```
>>>plot(x,y)
>>>title('y=exp(-x)*sin(4x)')
```

L'instruction permet de tracer une courbe passant par les points dont les coordonnées sont données dans les vecteurs *x* pour les abscisses, et *y* pour les ordonnées. Les points sont reliés par des segments de droites.

Remarque : pour une description exhaustive des possibilités graphiques que l'on étudiera plus tard, consulter l'excellent site http://enacit1.epfl.ch/cours_matlab/graphiques.html#graph_2d.



2.5 Ecrire et exécuter un script

L'écriture de scripts Octave/Matlab ("M-files") peut se faire dans l'éditeur de QtOctave et peut être exécuté depuis cet environnement. Pour les autres formes de mise en œuvre, nous renvoyons à la section 2.1.2.

Voici un petit exemple que l'on nommera "script1.m" :

```
% mon premier script Octave
a = input(' Rentrer la valeur de a : ');
b = input(' Rentrer la valeur de b : ');
n = input(' Nb d'intervalles n : ');
% calcul des abscisses
x = linspace(a,b,n+1);
% calcul des ordonnées
y = exp(-x).*sin(4*x);
% représentation graphique
plot(x,y);
xlabel('x'); % label sur les abscisses
ylabel('y'); % label sur les ordonnées
title('y=exp(-x)*sin(4x)');
```

Exercice 2 Écrire et exécuter le script précédent.

2.6 Entrées-sorties et fichiers

Voici un tableau récapitulatif tiré de http://enacit1.epfl.ch/cours_matlab/mfiles.html#entrees_sorties qui résume les principales fonctions concernées :

	Ecriture	Lecture
Interactivement	Écriture à l'écran • non formaté: <code>disp(chaine variable)</code> (une seule variable) • formaté: <code>fprintf(format, variable(s))</code> (ou <code>fprintf(...)</code>)	Lecture au clavier • non formaté: <code>var = input(prompt [, 's'])</code> • formaté: <code>var = scanf(format)</code>
Sur chaîne de caractères (le 1er <code>s</code> signifiant string)	• <code>string = sprintf(format, variable(s))</code> • autres fonctions : <code>mat2str ...</code>	• <code>var mat = sscanf(string, format [,size])</code> • autres fonctions : <code>strread ...</code>
Sur fichier texte (le 1er <code>f</code> signifiant file)	• <code>fprintf(file_id, format, variable(s) ...)</code> • autres fonctions : <code>save -ascii dlmwrite ...</code>	• <code>var = fscanf(file_id, format [,size])</code> • <code>line = fgetl(file_id)</code> • <code>string = fgets(file_id [,nb_car])</code> • autres fonctions : <code>load(fichier), textread, fileread, dimread ...</code>
Sur fichier binaire	• <code>fwrite(...)</code>	• <code>fread(...)</code>

2.6.1 Lecture par le clavier et affichage

Une saisie au clavier s'effectue comme cela a été illustré dans l'exemple de script donné dans la section précédente. On utilise la commande `input`.

Exemple 1 Quelques saisies au clavier

```
>>> n=input('entrer la dimension n')
entrer la dimension n 2
n =
2

>>> A=input('entrer une matrice')
entrer une matrice [1 2;3 4]
A =
1     2
3     4
```

L'entrée d'une chaîne de caractères se fait encore avec la fonction `input`, en ajoutant un second paramètre qui est la constante '`s`'.

Exemple 2 Saisie d'une chaîne de caractères

```
>>> nomfichier=input('Donner le nom du fichier','s')
Donner le nom du fichier result.txt
nomfichier = result.txt
>>>
```

Une des commandes les plus usuels pour afficher les variables est `disp` qui n'affiche qu'une seule variable/chaîne de caractère à la fois.

Exemple 3 *Affichage de données avec "disp"*

```
>>> A=[1, 2; 3, 4]
>>> disp(A)
 1   2
 3   4
```

On peut aussi utiliser l'instruction "fprintf" qui est en fait celle qui est héritée du langage C (ici en absence d'identifiant de fichier, c'est la sortie standard qui est utilisée).

Exemple 4 *Affichage avec la fonction "fprintf"*

```
>>> a=64; b=sin(a);
>>> fprintf('le sinus de %2.f \n est %f ',a,b)
le sinus de 64
est 0.920026
```

Le format d'affichage des nombres est hérité du C. On ne donnera pas plus de détails ici sur les spécifications de format pour `fprintf` (cf. site indiqué ci-dessus).

2.6.2 Utilisation de fichiers

Pour des détails complémentaires sur ces questions nous renvoyons le lecteur vers le site de l'EPFL http://enacit1.epfl.ch/cours_matlab/mfiles.html#entrees_sorties. Attention, les deux fonctions citées `fileread` et `textread` évoquées sur le site ne font plus partie des fonctions standards dans Octave 3.2. Il faut les ajouter via le package "io" disponible dans la liste des paquets installables par QtOctave. Nous ne les utiliserons pas ici.

Lecture/écriture directe d'une matrice dans un fichier

Deux fonctions peuvent être utilisées principalement dans ce cas, `dlmread` et `dlmwrite` :

- `dlmread` permet de lire l'intégralité d'un fichier texte de données numériques dans une matrice avec éventuellement l'indication d'un délimiteur.

Exemple 5 *Lecture de données numériques d'un fichier texte Écrire quelques données en lignes et colonne dans un fichier `donnees.txt` puis exécuter le script suivant :*

```
>>> M=dlmread('donnees.txt'); % vecteur colonne
>>> disp(M);
>>> M2=dlmread('donnees.txt', " ", [0, 0, 4, 4])
```

La dernière instruction renvoie la matrice des valeurs comprises entre l'élément (0,0) et l'élément (4,4), le séparateur étant noté comme un espace blanc.

- `dlmwrite` permet d'écrire des données dans un fichier, distinguables par un caractère séparateur à préciser si besoin.

Exemple 6 *écriture de données numériques dans un fichier texte*

```
>>> dlmwrite('donnees.txt', M, '-append');
```

Le dernier argument ("append") signale une écriture en fin de fichier, le séparateur est un espace par défaut.

Pour les autres options possibles, consulter la documentation Octave.

Fonctions classiques dérivées du langage C

Pour commencer, travailler sur un fichier nécessite de le créer puis de l'ouvrir avec la fonction `fopen` qui renvoie un identifiant pour le fichier. Cet identifiant sert ensuite pour de nombreuses fonctions de manipulation de fichiers, notamment pour fermer le fichier par `fclose`, ainsi que la lecture par `fscanf` et l'écriture par `fprintf`.

Lecture formatée Typiquement la procédure de lecture sur un fichier quelconque procède par une boucle qui termine avec le signal de fin de fichier rendu par la fonction `feof`.

Supposons un fichier texte 'data.txt' constitué de :

```
101 Martin Cahier 2 3.50
102 Charrier Crayon 5 2.95
```

Exemple 7 *Script de lecture de données formatées dans un fichier*

```
% fopen : le mode d'ouverture est indiqué par le second argument
% 'r' lecture; 'w' écriture; 'a' ajout en fin)
data_id = fopen('data.txt', 'r') ;
no = 1 ;
while not(feof(data_id))
    % lecture des éléments 1 à 1 (dernier argument de fscanf)
    No_client(no)    = fscanf(data_id,'%u',1) ;
    Nom{no,1}         = fscanf(data_id,'%s',1) ;
    Article{no,1}     = fscanf(data_id,'%s',1) ;
    Nb_articles(no)   = fscanf(data_id,'%u',1) ;
    Prix_unit(no)     = fscanf(data_id,'%f',1) ;
    no = no + 1 ;
end
status = fclose(data_id) ;
```

Noter que les chaînes de caractères sont stockées dans des tableaux cellulaires (utilisation des accolades pour accéder à un élément).

Les principaux types de format (similaire au C) sont :

- %u et %d désignent respectivement un entier positif (naturel) et un entier relatif (positif ou négatif)
- %f désigne les flottants
- %s désigne les chaînes de caractères

Exercice 3 Tester les instructions précédentes sur le fichier proposé et sur un fichier de données que vous construirez. Tester également les fonctions de lecture de ligne `fgetl` et/ou de caractère `fgets`

Ecriture formatée Avec le même type de format on peut écrire dans un fichier précédemment ouvert en écriture via `fprintf`

Exemple 8 Écriture d'un fichier avec `fprintf` Tester le script suivant (si vous ne précisez aucun `file_id`, on écrit sur la sortie standard) :

```
>>> a=0:0.1:1;
>>> b=sin(a);
>>> fprintf(file_id, '%1.2f %1.4f \n', [a;b]);
0.00 0.0000
0.10 0.0998
0.20 0.1987
0.30 0.2955
0.40 0.3894
0.50 0.4794
0.60 0.5646
0.70 0.6442
0.80 0.7174
0.90 0.7833
1.00 0.8415
```

Les formats peuvent intégrer des spécifications d'affichage (ici nombre avant et après la virgule pour les flottants)

2.7 La programmation sous Octave

Octave propose un langage de programmation complet compatible à 100% avec Matlab, de haut niveau dans le sens où existent de nombreuses fonctions ou procédures de calcul dédiées aux mathématiques que nous n'aurons pas à développer. Il s'agit d'un langage de ce point de vue plus "simple" que les langages classiques (C, C++, Java, ...) car il intègre déjà des primitives sophistiquées, notamment pour manipuler les "matrices". Dans un soucis de simplification, le langage ne nécessite pas de déclarations qui sont gérées automatiquement par l'interpréteur. Une autre facilité appréciable est l'intégration d'une bibliothèque graphique.

Cependant, le langage Octave/Matlab comme tout langage interprété est plus lent que les langages compilés. Cependant certaines instructions sont optimisées pour accélérer les calculs comme les opérateurs arithmétiques matriciels, plus efficace que des boucles classiques. Ceci dit, un programme Matlab/Octave nécessitera facilement 10 fois plus de temps qu'un programme compilé.
C'est pourquoi Octave est interfaçable avec des langages compilés pour optimiser certains calculs.

2.7.1 Branchements et boucles

If - instructions conditionnelles

Exemple 9 *Test avec alternative* Ecrire et exécuter le script suivant en modifiant les valeurs de n :

```
n=2;
if (n==1)
    disp('un');
elseif (n==2)
    disp('deux');
else
    disp('autre');
endif
```

On peut enchaîner plusieurs tests avec elseif et pas else if qui serait interprété comme un nouveau if imbriqué.

La condition logique de sélection ne nécessite pas de parenthèses. Il peut s'agir d'une matrice sur laquelle on vérifie que tous les éléments ne sont pas faux (différents de 0). Attention aux opérateurs logiques qui peuvent être différents des opérateurs arithmétiques.

Switch - alternatives

Exemple 10 Instruction switch Même principe que dans l'exemple précédent

```
switch n
    case{1}
        disp('un');
    case{2}
        disp('deux');
    otherwise
        disp('autre');
endswitch
```

Seule l'instruction qui suit directement un case est exécutée (et pas les suivantes) si la comparaison est vraie. otherwise (qui est facultatif) sera effectué si tous les tests précédents ont échoués.

For - boucle

Pour construire une boucle de type `for`, on utilise un index qui décrit un ensemble de valeurs. On utilisera souvent l'opérateur deux-points " :" que l'on a vu dans le chapitre précédent. L'itérateur de la boucle `for` peut être plus généralement une matrice, pour laquelle chaque itération renvoie un vecteur colonne.

Exemple 11 Premier exemple élémentaire

```
for i=1:2:5  
    disp(i)  
endfor
```

Exemple 12 Exemple utilisant une matrice Qu'affiche la commande ci-dessous ?

```
>>> for n=[1 5 2;4 4 4] , n, endfor
```

Remarque 2 Un point important sur l'utilisation de la boucle `for` est de savoir si il n'est pas possible de la remplacer par une opération matricielle, c'est à dire une opération "vectorisée". En effet il peut y avoir un facteur de performance de l'ordre de 10 à 100 entre ces deux calculs au bénéfice des opérations vectorisées.

While - boucle

Une boucle "while" conditionne la répétition d'une séquence d'instructions de manière conditionnée à la vérification d'une expression booléenne.

Exemple 13 Une boucle "while" élémentaire calculant la somme des 10 premiers entiers Lancer le script ci-dessous :

```
s=0; i=1;  
while (i<=10)  
    s=s+i  
    i=i+1  
endwhile
```

Pour ce calcul spécifique, on pourra utiliser plus efficacement la fonction prédéfinie "sum" :

Exemple 14 Utilisation de la fonction "sum"

```
>>> sum(1:10)  
ans =  
55.
```

Remarque 3 La boucle "while" produit le même effet que la boucle "for" et il faut à nouveau se demander si une opération vectorielle ne peut pas la remplacer de manière plus efficace.

Break et continue - sorties

Il s'agit de deux instructions qui sont à bannir pour les "puristes" de l'algorithme (car traduisant une faiblesse d'analyse et des comportements invérifiables).

L'instruction "break" permet de sortir d'une boucle. Typiquement on vérifie une condition qui lorsqu'elle est satisfaite, ne nécessite plus de continuer la boucle (similaire à un traitement d'exception dans d'autres langages).

Exemple 15 *Utilisation basique de "break"* Exécuter ce script pour observer l'effet du "break"

```
s=0; i=1;
while (true)
    if (i>10)
        break
    endif
    s=s+i
    i=i+1
endwhile
```

L'instruction "continue" permet de signaler à l'intérieur d'une boucle que l'on doit "sauter" la fin des instructions restant à exécuter dans le corps de la boucle pour passer directement à l'itération suivante.

Exemple 16 *Utilisation basique de "continue"*

```
s=0; i=0;
while (i<=10)
    if (modulo(i, 2)==0)
        i=i+1
        continue
    endif
    s=s+i
    i=i+1
endwhile
```

Remarque 4 La construction précédente, qui est un exemple de construction algorithmique à ne pas suivre, s'obtient plus simplement et efficacement avec la seule instruction : `sum(1:2:10)`.

2.7.2 Les fonctions

La démarche essentielle qui permet d'élaborer des programmes avec Matlab/Octave, est basée sur la décomposition fonctionnelle, c'est à dire à élaborer un programme de manière incrémentale en définissant des briques de base fonctionnelles. On

va ainsi construire des fonctions de base que l'on va assembler ou utiliser dans d'autres fonctions plus élaborées et abstraites.

De nombreuses fonctions prédéfinies sont disponibles dans Octave (fonctions mathématiques notamment), mais on peut en définir de nouvelles à volonté. Un même script peut comporter plusieurs fonctions qui peuvent s'appeler au sein du script.

Remarque 5 *Cependant pour appeler une fonction à l'extérieur du script où elle est définie, celle-ci devra être déclarée en première ligne du script et porter impérativement le même nom que le fichier du script.*

Une fonction Octave est déclarée à l'aide du mot clef `function` de la manière suivante :

```
function [arg_sortie, ...]=ma_fonction(arg_entree, ...)
```

où à gauche les arguments de sortie sont listés entre crochets s'il y en a plusieurs, tandis que les arguments d'entrée sont passés entre parenthèses à la fonction nommée. Les arguments sont passés par valeur à la fonction et non par référence.

L'appel ultérieur de la fonction se fait par la syntaxe suivante :

```
% si un seul argument de sortie
var_sortie=ma_fonction(var_entree1, var_entree2, ...);
% si plusieurs arguments de sortie
[var_sortie1, var_sortie2...]=ma_fonction(var_entree1, var_entree2, ...);
```

Remarque 6 *Les variables déclarées dans la fonction ont une portée locale à la fonction. Pour les rendre visibles de l'extérieur de la fonction, il faudra les déclarer précédée du mot-clé `global`.*

Exemple 17 *Décomposition LU d'une matrice (exécuter les lignes d'instructions suivantes puis utiliser "help" pour comprendre ce qu'elles font)*

```
>>>A=rand(3,3);
>>>[L,U]=lu(A)
>>>[L,U,P]=lu(A)
```

Bien écrire une fonction

1. déclarer la fonction (nom arguments d'entrée / sortie),
2. utiliser des commentaires pour renseigner l'aide en ligne :
 - première ligne de commentaire immédiatement après la déclaration, la "H1-line", permet de résumer la fonction avec ses mots-clés utilisés par les commandes d'aide `help` ou `lookfor`
 - autres commentaires d'explications de ce que fait la fonction
3. déclaration des variables globales ou statiques s'il y en a (les variables statiques sont immuables et déclarées précédées du mot-clé `persist`),

4. écriture du code proprement dit de la fonction et commentaires associés,
5. penser à affecter des valeurs à chaque argument de sortie,
6. fin de fonction signalée par le mot-clé `endfunction`.

Exemple 18 Écriture d'une fonction renvoyant un vecteur Écrire la fonction suivante dans un fichier de script et la tester dans un autre script ou en ligne de commande.

```
function [somme,produit]=fsomprod(a,b)
%FSOMPROD somme et produit de 2 nombres, vecteurs ou matrices
%   Usage: [S,P]=FSOMPROD(V1,V2)
%           Retourne matrice S contenant la somme de V1 et V2,
%           et matrice P contenant le produit de V1 et V2
%           élément par élément

if (nargin~=2)
    error('cette fonction attend 2 arguments');
endif
if (~isequal(size(a),size(b)))
    error('les 2 arg. n''ont pas la même dimension');
endif

somme=a+b;
produit=a.*b; % produit élément par élément !
endfunction      % sortie de la fonction
```

Noter l'utilisation de `nargin` qui renvoie le nombre d'arguments passés à la fonction, `isequal` pour vérifier l'égalité de vecteurs, et `error` pour afficher un message sur la sortie standard.

Remarque 7 Il est essentiel dans le code de la fonction de trouver des instructions qui affectent des valeurs à chacun des arguments de sortie. Si ces affectations ne sont pas présentes, une erreur sera générée à l'appel de la fonction.

Fonctions récursives

Nous donnons ci-dessous un nouvel exemple permettant d'illustrer la construction de fonctions récursives, ne nécessitant avec Octave aucun signalement spécifique.

Exemple 19 Fonction récursive factorielle Écrire la fonction ci-dessous dans un fichier "facto.m" et faire des tests d'exécution sous Octave en se plaçant dans le même dossier que le script.

```
function p=facto(n)
if n<=1
```

```
p=1;
else
    p=n*facto(n-1);
endif
endfunction
```

Remarque 8 Compléments vus en TP : des fonctions particulières de manipulation de fonctions permettant notamment de servir d'aide au débogage (*disp*, *pause*, *keyboard*, *dbstop*, *dbclear*, *echo*, *error*, *warning*, *nargin*, *nargout*, *typeinfo* ...).

Utiliser une fonction comme argument d'une autre fonction

(cf doc Octave)

Pointeur de fonction Une fonction est elle-même une variable du type "function" que l'on peut manipuler grâce à un pointeur de fonction, ce qui permet de passer une fonction en argument d'entrée pour une autre.

On récupère le pointeur d'une fonction par l'opérateur @ suivi du nom de la fonction, que l'on peut ensuite utiliser par exemple dans la fonction *quad* (intégration) ou *lsode* (résolution de système dynamique). L'exemple suivant montre l'intégration d'un *sin* sur $[0, \pi]$:

Exemple 20 Utilisation d'un pointeur de fonction

```
>>>f1 = @sin;
>>>quad (f1, 0, pi)
ans = 2
>>> feval(f1,pi/4)
ans = 0.70711
>>> f1(pi/4)
ans = 0.70711
```

On a utilisé la commande *feval* pour exécuter la fonction, ou directement par son nom suivi de parenthèses contenant les arguments.

Fonction anonyme Une autre façon de passer une fonction en paramètre d'une autre est de définir des fonctions anonymes. Cela se réalise par le même opérateur dans la syntaxe générale suivante :

@(liste d'arguments) expression

Ci-dessous quelques exemples tirés de la doc Octave pour illustrer ce fonctionnement :

Exemple 21 Fonction anonyme

```
>>> f = @(x) x.^2; % permet également de définir f
>>> quad(f, 0, 10)
ans = 333.33
>>> quad(@(x) sin(x), 0, pi) % en fonction anonyme
ans = 2
>>> a = 1; b = 2; % les paramètres sont valués
>>> quad(@(x) betainc(x, a, b), 0, 0.4)
ans = 0.13867
```

Construire des fonctions de manière dynamique

Matlab/Octave propose de définir des fonctions "in-line" dans une chaîne de caractère passée en argument de la fonction `inline`.

Exemple 22 Définition d'une fonction "in-line"

```
>>> f = inline("x^2 + 2");
>>> f(2)
ans = 6
```

Cette possibilité est vraiment intéressante car elle permet de définir du code de manière dynamique : une chaîne de caractères contenant la totalité ou des éléments d'une fonction, peut-être lue au cours de l'exécution d'un programme et être utilisée en temps que fonction dans la suite du déroulement du programme.

Fonction eval Matlab/Octave offre aussi la possibilité d'évaluer une expression qui est décrite dans une chaîne de caractères, grâce à la fonction `eval` et l'exécute.

Exemple 23 Conversion de chaîne de caractères en expression évaluable ou en instructions exécutables Écrire le script suivant et l'exécuter en entrant une fonction déjà vue précédemment :

```
fonction = input('Quelle est la fonction y=fct(x) à tracer ?','s');
min_max = input('Indiquez [xmin xmax] : ');
x = linspace(min_max(1),min_max(2),100);
eval(fonction,'error("fonction incorrecte")');
plot(x,y);
```

Bibliothèques de fonctions

Pour consulter les fonctions disponibles par packages Octave, ouvrir la page http://octave.sourceforge.net/functions_by_package.php.

2.8 Graphisme

(cf. http://enacit1.epfl.ch/cours_matlab/graphiques.html)

2.8.1 Compléments

L'exemple précédent a montré l'utilisation conjointe de la fonction `plot` et de `eval`, la fonction `fplot` ("function plot") permet de tracer une courbe selon une fonction définie "inline" ou définie par une chaîne de caractères, ou référencée par un pointeur. En 3D, la fonction `ezplot3` permet de tracer une fonction paramétrée, c'est-à-dire définie par l'expression des trois coordonnées en fonction du même paramètre (le temps par exemple).

Exemple 24 Voici un exemple de tracé de fonction de la courbe représentative d'une fonction à une variable réelle...

```
function [y]=sigmoid(x)
y=1./(1 + exp(-20 .* (x - 0.25)));
endfunction
%
fplot(@sigmoid, [0, 1], 'r');
```

...puis le tracé d'une courbe paramétrée :

```
>>>ezplot3('t*sin(t)', 't*cos(t)', 't', [0, 10*pi]);
```

La surface représentative de la dernière construction est illustrée par la figure 2.4.

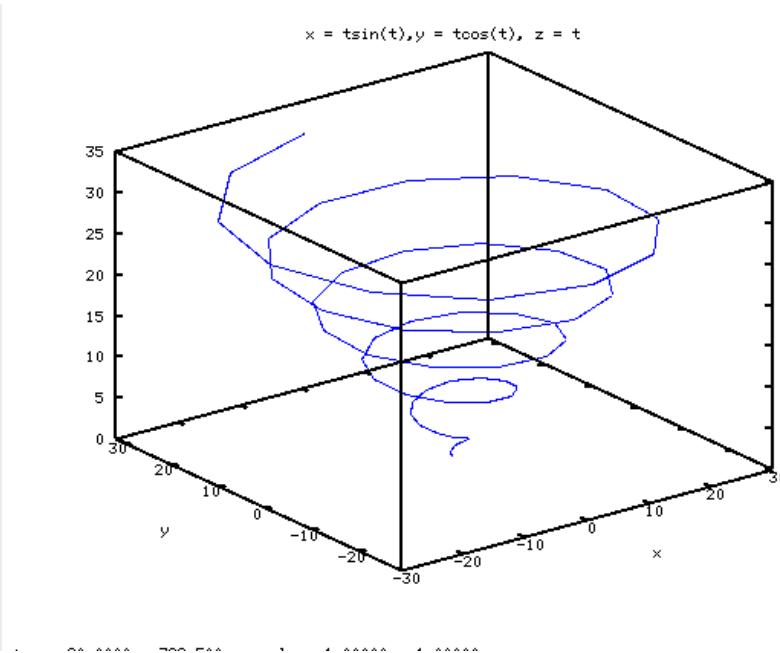


FIGURE 2.4 – Courbe représentative d'une fonction paramétrée avec `ezplot`

2.9 Modélisation et systèmes différentiels

Matlab/Octave permet de résoudre *numériquement* des équations différentielles. On ne cherche donc pas ici une solution analytique de la solution d'une équation différentielle mais on calcule une approximation de la solution à une succession de pas de temps, à partir d'une condition initiale.

Soit une équation avec une condition initiale (problème de Cauchy) :

$$\begin{cases} \frac{du(t)}{dt} = f(t, u(t)) \\ u(t_0) = u_0 \end{cases} \quad (2.1)$$

où $u : \mathbb{R} \rightarrow \mathbb{R}^n$, $u_0 \in \mathbb{R}^n$ et $f : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$.

Le théorème de Cauchy-Lipschitz nous dit que si f est lipschitzienne par rapport à sa deuxième variable, sur un intervalle fermé contenant u_0 ¹, alors il existe une solution unique pour la fonction u , donnant ainsi une valeur unique de cette fonction en une valeur t donnée, à condition de ne pas sortir de l'intervalle dans lequel la condition de Cauchy-Lipschitz est énoncée. Dans la suite on supposera que cette condition de régularité de la fonction f est vérifiée.

1. c'est à dire que pour cet intervalle fermé I , il existe k réel strictement positif tel que pour toute valeur de t et tout couple de valeurs de (u, v) dans l'intervalle I , $\|f(t, u) - f(t, v)\| < k\|u - v\|$

La fonction `lsode` est la fonction Octave qui va permettre de calculer une solution numérique, c'est à dire de l'"intégrer" sur un intervalle démarrant en t_0 , en partant du vecteur colonne u_0 .

Pour cela, il faut commencer par définir la fonction f du problème de Cauchy décrit ci-dessus (2.1). On la définit comme une fonction Octave classique :

```
function [f] = SecondMembreEDO(u,t)
    // on décrit ici les composantes de f
endfunction
```

Remarque 9 *Même si la fonction f est autonome (c'est à dire indépendante de t), il faudra quand même mettre t comme second argument de f , afin qu'ultérieurement, il soit identifié comme la variable de la solution cherchée.*

Exemple 25 *On considère l'équation de Van der Pol :*

$$y'' = c(1 - y^2)y' - y$$

que l'on reformule classiquement comme un système de deux équations différentielles du premier ordre, en posant

$$\begin{aligned} u_1(t) &= y(t) \\ u_2(t) &= y'(t) \end{aligned}$$

Soit le système équivalent

$$\frac{d}{dt} \begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} u_2(t) \\ c(1 - u_1^2(t))u_2(t) - u_1(t) \end{bmatrix}$$

Ce que l'on décrit par la fonction Octave suivante, en fixant $c = 0.4$ (attention à l'ordre des arguments d'entrée de la fonction, le temps vient en second !) :

```
function [f] = vanDerPol(u,t)
    f(1)=u(2)
    f(2)=0.4*(1-u(1)^2)*u(2) - u(1)
endfunction
```

Pour résoudre numériquement l'équation, il faut donc définir une distribution uniforme de valeurs de t sur un intervalle, à partir de t_0 , avec par exemple, la fonction `linspace`. On appelle ensuite la fonction `lsode` dont les arguments sont les suivants :

- les deux premiers paramètres correspondent à la condition initiale du problème de Cauchy, ce qui correspond ici au vecteur $[u_1(t_0), u_2(t_0)]$ et t_0 ;
- le troisième est le vecteur des valeurs successives de t ;
- le quatrième est la fonction f du problème de Cauchy (2.1).

Ceci correspond ainsi à deux lignes de code prenant la forme suivante :

```
t = linspace(t0,T,m);
[U] = lsode("vanDerPol",u0, t);
```

On récupère alors une matrice U telle que $U(i, j)$ est la solution approchée de $u_i(t(j))$.

On donne ci-après un traitement complet de l'équation de Van Der Pol. On commence par tracer le champ de vecteur correspondant de la fonction f du problème de Cauchy pour une valeur quelconque de t , ici prise à 0, sachant que cette équation est autonome, ce champ de vecteur sera le même quelque soit la valeur de t . Selon la construction du problème de Cauchy (2.1), ce champ vectoriel va décrire les tangentes aux trajectoires correspondant à des portraits de phase, c'est à dire aux courbes $(u_1(t), u_2(t))$, paramétrées par le temps. On effectue ensuite une résolution numérique de l'équation différentielle (avec `ode`). On trace alors par-dessus le champ vectoriel, la courbe de portrait de phase démarrant à la condition initiale. On trace ensuite dans une autre fenêtre graphique, l'évolution de chacune des composantes de la solution par rapport au temps.

```
printf("Résolution d'une équation de Van der Pol.\n");
% déclaration de la fonction en second membre
function [f] = vanderpol(u,t)
    f(1)=u(2);
    f(2)=0.4*(1-u(1)^2)*u(2)-u(1);
endfunction

% tracé des lignes de champ
clf;
[x, y] = meshgrid (-5:0.5:5);
h = quiver (x, y, y, 0.4*(1-x.^2).*y-x, 0.5);
set (h, "maxheadsize", 0.1);
hold on;
% calcul de la solution avec la condition initiale :
m=500;
T=30;
t=linspace(0,T,m);
u0 = [-2.5; 2.5];
% marquage de la condition initiale
plot(u0(1),u0(2),'b*');
%affichage de la solution
[u] =lsode("vanderpol",u0,t);
plot(u(:,1),u(:,2),'r');
%titre, légendes, etc...
title('Equation de Van Der Pol : plan de phase (y(t), dy(t)/dt)');
xlabel('y(t)'); ylabel('dy(t)/dt');
```

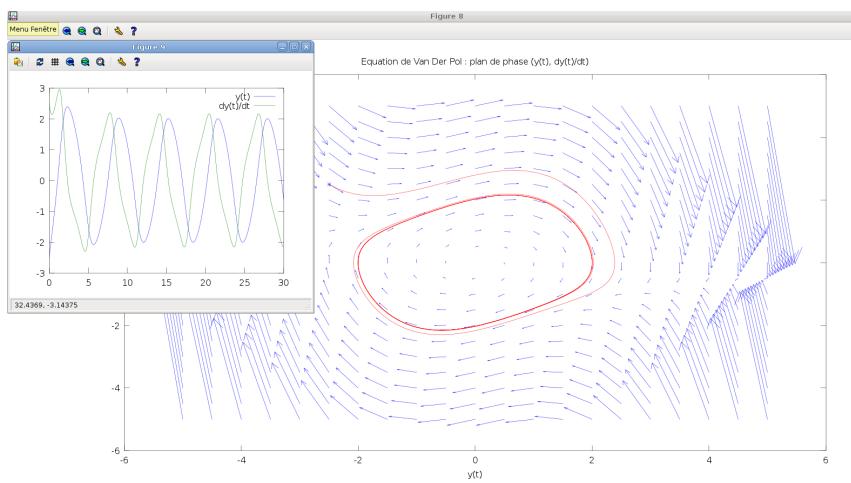


FIGURE 2.5 – Tracé du champ de vecteur et de la solution de lsode

```

hold off;
figure;
plot(t,u);
legend('y(t)', 'dy(t)/dt');
    
```

Remarque 10 *Derrrière la résolution numérique de la fonction lsode, il y a plusieurs algorithmes possibles que l'on peut contrôler soi-même en indiquant celui que l'on souhaite utiliser (non décrit ici ... voir l'aide si nécessaire). D'autre part un paquet spécifique odepkg que l'on peut installer depuis QtOctave (menu "Config/Install Octave Packages") existe et propose des fonctions spécialisées dans le traitement des équations différentielles.*

Chapitre 3

Modèles d'évolution des populations

Sommaire

3.1	Un aperçu des systèmes	35
3.2	Description de la dynamique de population	36
3.2.1	Définitions	36
3.2.2	Observations	36
3.3	Les modèles continus de croissance déterministe	36
3.3.1	Modèle de Malthus et loi exponentielle	36
3.3.2	Modèles de Verhulst et loi logistique	38
3.3.3	Loi de Gompertz	41
3.3.4	TP : représentation graphique de modèles différentiels avec Scilab	41
3.4	Identification et validation d'un modèle	41
3.4.1	Identification et ajustement d'un modèle	41
3.4.2	Validation d'un modèle	46
3.5	Modèles discrets de dynamique de population	54
3.5.1	Construction d'un modèle discret et interprétation - application à la loi exponentielle	54
3.5.2	Analyse d'un modèle discret - application à la loi logistique	55
3.5.3	TP : Etude de modèles discrétisés avec Scilab, à partir de la loi logistique	57

3.1 Un aperçu des systèmes

Cf. slides de présentation

3.2 Description de la dynamique de population

3.2.1 Définitions

3.2.2 Observations

Cf. slides de présentation.

3.3 Les modèles continus de croissance déterministe

Nous étudions dans les sous-sections suivantes, différentes lois de dynamique d'une population, en partant des plus simples et des plus intuitives. Nous caractérisons leurs expressivités et leurs limitations pour voir comment il est possible de les étendre.

3.3.1 Modèle de Malthus et loi exponentielle

Dans le chapitre d'introduction, nous avons présenté un énoncé simple permettant de décrire l'évolution d'une population et nous en avons déduit une expression mathématique sous forme d'une équation différentielle.

Pendant un intervalle de temps δt , la variation δN du nombre d'individus de la population est proportionnel à N et δt .

Soit la traduction mathématique :

$$\delta N = k\delta t$$

où k est une constante de proportionnalité. Si $k > 0$, alors la population croît. Si $k < 0$, la population décroît. On a

$$\frac{\delta N}{\delta t} = kN$$

En considérant maintenant N comme une fonction du temps t , qui est continue et dérivable. En faisant tendre δt vers 0, alors de l'expression précédente, on obtient :

$$N'(t) = \frac{dN(t)}{dt} = kN(t)$$

On a donc une équation différentielle du premier ordre dont l'inconnue, $N(t)$, est une fonction. Cette équation ne décrit que la manière dont la population évolue. On ne pourra résoudre cette équation concrètement (i.e. c'est à dire connaître la valeur de $N(t)$, à un instant t donné) si l'on dispose d'une condition initiale du type

$$N(0) = N_0 \quad \text{connu}$$

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS

qui consiste donc à connaître l'effectif initial de la population à l'instant considéré (à $t = 0$). Le problème ainsi posé (problème de Cauchy) :

$$\begin{cases} \frac{dN(t)}{dt} = kN(t) \\ N(0) = N_0 \end{cases}$$

possède bien une solution unique qui est

$$N(t) = N_0 e^{kt}$$

En effet, si on dérive cette solution, on a

$$N'(t) = N_0 k e^{kt} = kN(t)$$

$N(t)$ est donc une loi exponentielle croissante si $k > 0$ et décroissante si $k < 0$ (voir la figure 3.1).

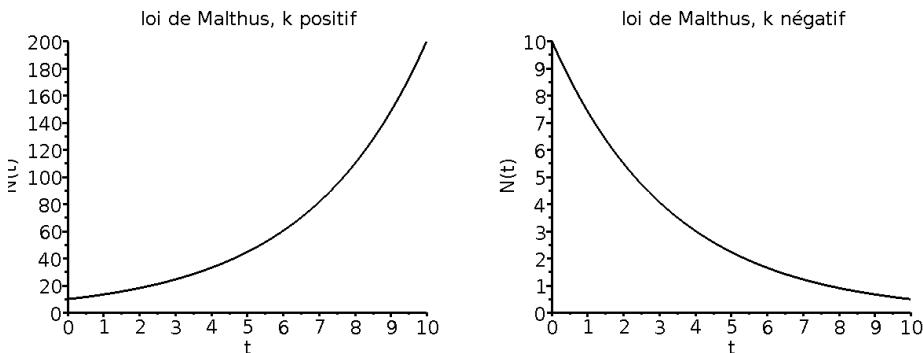


FIGURE 3.1 – Loi de Malthus - sur la figure de gauche, la loi croissante (k positif) ; sur le figure de droite, la loi est décroissante (k négatif)

Ce modèle a été proposé par Malthus au 18^{eme} siècle (1798) et il prédisait une insuffisance des ressources de la planète pour les générations futures. Le coefficient k est alors défini comme la différence entre le taux τ_n de naissance et le taux de mortalité τ_m :

$$k = \tau_n - \tau_m.$$

Remarque 11 *Ce modèle est utilisé tel quel pour étudier la désintégration d'atomes radioactifs (et alors $k < 0$) mais il est assez mal adapté aux croissances de population observées dans la nature qui se trouvent être généralement limitées par l'environnement.*

Pour valider le modèle à partir d'effectifs relevés dans la réalité, il est préférable de tracer la courbe en coordonnées semi-logarithmiques car elle correspond alors à une droite. Une régression linéaire appliquée à ce changement de variable permet d'ajuster les paramètres (voir figure 3.2).

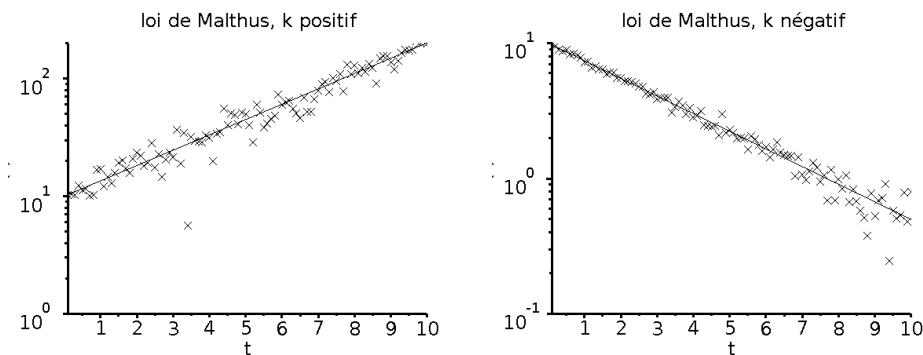


FIGURE 3.2 – Ajustement de données réelles par une loi de Malthus en échelle semi-logarithmique, dans le cas croissant (k positif) à gauche et dans le cas décroissant (k négatif) à droite

3.3.2 Modèles de Verhulst et loi logistique

Le modèle de Malthus d'évolution d'une population ne prend pas en compte divers facteurs qui interviennent dans le développement et la dynamique :

- la limitation du milieu nutritif ou spatial ;
- les inhibitions ou interactions dues à la proximité d'un grand nombre d'individus ;
- les régulations biologiques, ...

Pour affiner le modèle précédent

$$\frac{dN(t)}{dt} = kN(t)$$

on suppose que le coefficient de proportionnalité, k , varie en fonction de $N(t)$, il devient donc une fonction de N , soit noté $k(N)$.

Si l'on suppose que le nombre d'individus de la population est limité par une valeur maximale, N^* , en raison du milieu qui ne permet de nourrir que N^* individus, il faut donc que $k(N)$ diminue lorsque N s'approche de N^* pour s'annuler en $N = N^*$.

On prend alors $k(N) = \gamma(N^* - N)$, avec γ constant.

On appelle *équation logistique* proposée par Verhulst en 1838 :

$$\frac{dN(t)}{dt} = \gamma(N^* - N(t))N(t).$$

En prenant $N(0) = N_0$ comme condition initiale, on montre que la solution s'écrit

$$N(t) = \frac{N^*}{1 + \left(\frac{N^*}{N_0} - 1\right) e^{-\gamma N^* t}}$$

C'est la fonction logistique.

Exercice 4 Vérifier que $N(t)$ est bien la solution de l'équation logistique et qu'elle vérifie la condition initiale.

Les tracés graphiques des courbes représentatives de $N(t)$ et de $\frac{dN(t)}{dt}$ sont donnés dans la figure 3.3 lorsque la loi logistique est croissante, et dans la figure 3.5 lorsque la loi logistique est décroissante.

On notera l'existence d'un point d'inflexion caractérisant le phénomène de régulation.

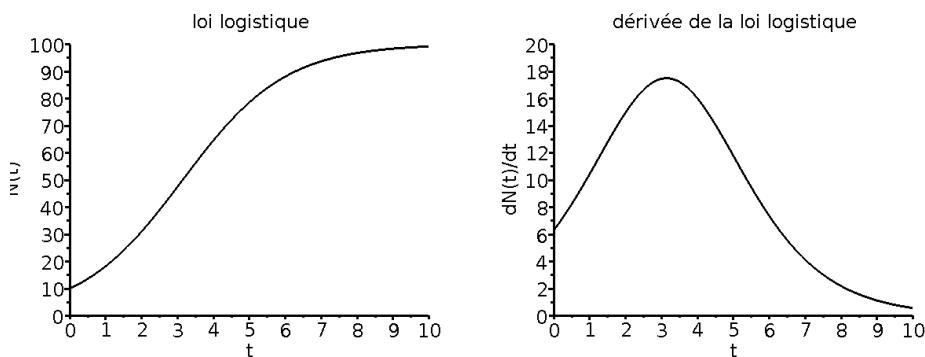


FIGURE 3.3 – Loi logistique croissante sur la partie droite et sa dérivée sur la partie gauche

On trace également des courbes représentatives de $N(t)$ et de $\frac{dN(t)}{dt}$ dans le cas de la décroissance, c'est à dire que $k(N) < 0$. Mais $k(N) = \gamma(N^* - N)$ est négatif soit si $\gamma < 0$, soit si $N^* - N < 0$, c'est à dire $N > N^*$, signifiant que l'on est au-dessus de l'effectif maximal. Les deux cas sont représentés dans les figures 3.4 et 3.5.

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS

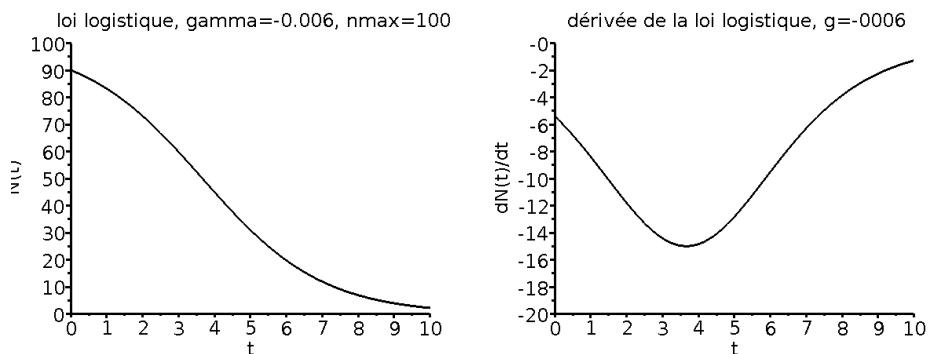


FIGURE 3.4 – Loi logistique décroissante (coef. de croissance γ négatif) sur la partie droite et sa dérivée sur la partie gauche

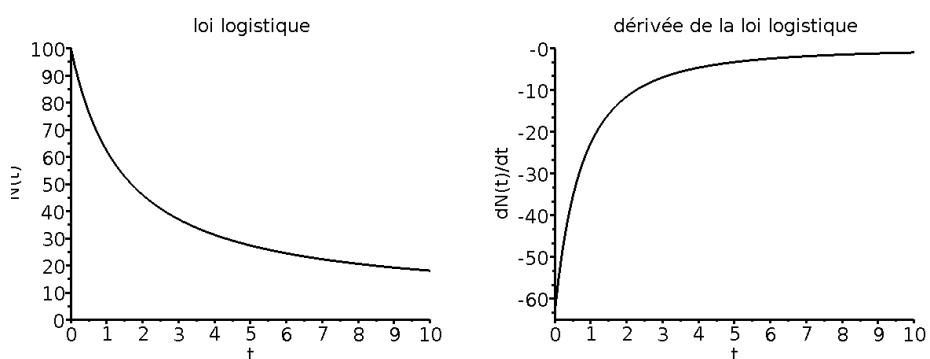


FIGURE 3.5 – Loi logistique décroissante (effectif supérieur à la population max) sur la partie droite et sa dérivée sur la partie gauche

3.3.3 Loi de Gompertz

Il s'agit à nouveau d'un raffinement de la loi exponentielle pour lequel k dépend de N , en considérant un effectif maximal de population N^* . Dans ce modèle on fait évoluer de manière logarithmique $k(N)$ quand N tend vers N^* :

$$\begin{aligned} k(N) &= \gamma(\ln(N^*) - \ln(N)) \\ k(N) &= \gamma \ln(N^*/N) \end{aligned}$$

L'équation différentielle, appelée modèle de Gompertz, s'écrit alors

$$\frac{dN(t)}{dt} = \gamma N(t) \ln\left(\frac{N^*}{N(t)}\right)$$

En prenant comme condition initiale $N(0) = N_0$, on peut montrer que la solution de l'équation différentielle s'écrit :

$$N(t) = N^* e^{\ln(N_0/N^*)e^{-\gamma t}}$$

ce modèle a été introduit par Gompertz (1825) pour effectuer des calculs en assurance vie. Il est utilisé comme modèle d'extinction de populations animales ou comme modèle de croissances de tumeurs cancéreuses.

On étudiera cette fonction plus en détail en TP, en observant graphiquement son évolution par rapport à une loi logistique.

3.3.4 TP : représentation graphique de modèles différentiels avec Sci-lab

3.4 Identification et validation d'un modèle

Dans cette section, on se place dans la situation où un modèle a été retenu pour caractériser un phénomène réel. On s'intéresse alors à deux aspects très importants pour le modélisateur qui consiste à pouvoir montrer que ce modèle reflète bien le phénomène qu'il doit représenter. On doit alors souvent "caler" certains de ses coefficients par rapports à des observations réelles et ensuite on doit ensuite valider, c'est à dire expliquer en quoi il fait sens, au delà de simples calculs techniques d'ajustement de paramètres. Ces deux aspects sont évoqués dans la suite.

3.4.1 Identification et ajustement d'un modèle

Il s'agit d'attribuer des valeurs numériques aux paramètres du modèles à partir de données expérimentales.

Supposons, par exemple que l'on dispose de données $(t_i, d_i)_{1 \leq i \leq n}$ qui font apparaître une relation linéaire comme sur la figure 3.6.

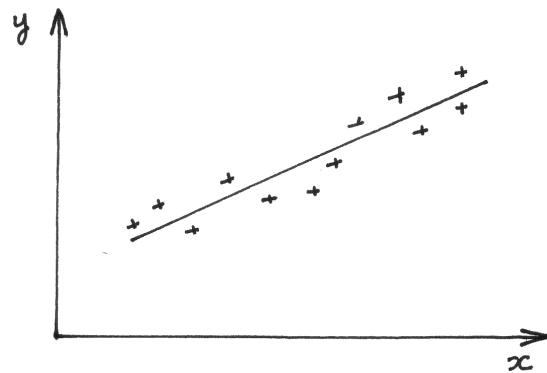


FIGURE 3.6 – Calcul de régression sur un jeu de données

on va rechercher un ajustement de ces données par un calcul de régression linéaire du type

$$\phi(t) = at + b$$

où a et b sont déterminés par la méthode des moindres carrés.

On les détermine alors en minimisant

$$S(a, b) = \sum_{i=1}^n (d_i - \phi(t_i))^2$$

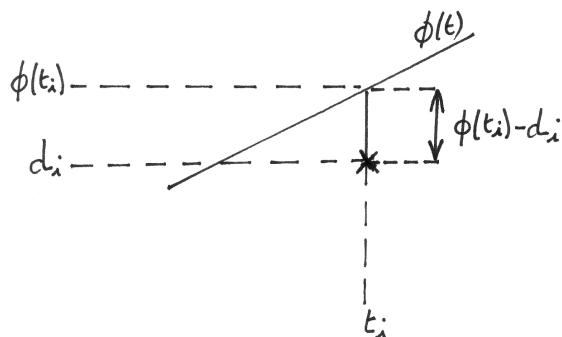


FIGURE 3.7 – Distance entre les données et la droite au sens des moindres carrés

Le minimum s'obtient en annulant les dérivées partielles de cette fonction S à deux variables :

$$\frac{\partial S}{\partial a} = \frac{\partial S}{\partial b} = 0$$

Ce qui conduit à poser un système de deux équations linéaires, d'inconnues a et b , dont la solution est

$$\begin{cases} a &= \frac{s_{td} - \frac{1}{n}s_t s_d}{s_{t^2} - \frac{1}{n}(s_t)^2} \\ b &= \frac{1}{n}(s_d - a s_t) \end{cases}$$

où on définit les différentes sommes :

$$s_t = \sum_{i=1}^n t_i ; \quad s_d = \sum_{i=1}^n d_i ; \quad s_{td} = \sum_{i=1}^n t_i d_i ; \quad s_{t^2} = \sum_{i=1}^n (t_i)^2$$

Cette méthode de régression linéaire se généralise assez facilement lorsque le modèle d'ajustement est un polynôme, comme par exemple :

$$\phi(t) = at^5 + bt^2 + c$$

D'une manière générale, si le modèle est **linéaire par rapport aux paramètres à identifier**, c'est à dire sous la forme générique suivante :

$$\phi(t) = \sum_{j=1}^m p_j f_j(t)$$

alors on utilise le même critère de minimisation, soit à minimiser la fonction suivantes qui dépend de n variables :

$$S(p_j ; 1 \leq j \leq n) = \sum_{i=1}^n (d_i - \phi(t_i))^2$$

Le calcul du minimum de cette fonction s'obtient en annulant toutes ses dérivées partielles :

$$\forall j ; 1 \leq j \leq m \quad \frac{\partial S}{\partial p_j} = 0$$

On obtient alors un système de m équation linéaires à m inconnues que l'on peut résoudre numériquement.

On s'intéresse maintenant au cas où le modèle **n'est pas linéaire par rapport aux paramètres à identifier**. Comme on le verra, cette situation est finalement assez fréquente. Dans cette situation, les calculs des dérivées partielles $\frac{\partial S}{\partial p_j}$ ne donnent plus d'expressions linéaires par rapport aux paramètres p_j à identifier. On doit donc résoudre un problème non linéaire.

On le résout par une méthode itérative qui nécessite la connaissance d'une approximation initiale des coefficients à ajuster, soit les valeurs $\{p_{j0} ; 1 \leq j \leq m\}$.

Le processus itératif consiste alors à construire une suite de l'ensemble de ces coefficients $(p_{jk} ; \quad 1 \leq j \leq m)_{k \geq 1}$ qui, de proche en proche, tend à minimiser

$$\sum_{i=1}^m (d_i - \phi(t_i))^2$$

Géométriquement, si on a deux paramètres à identifier, que l'on note a et b , alors la fonction à minimiser

$$S(a, b) = \sum_{i=1}^m (d_i - \phi(t_i))^2$$

peut être représenté graphiquement par une surface dont on recherche le minimum, telle que représentée graphiquement en 3D ou encore en courbes de niveaux, tel que dans la figure 3.8.

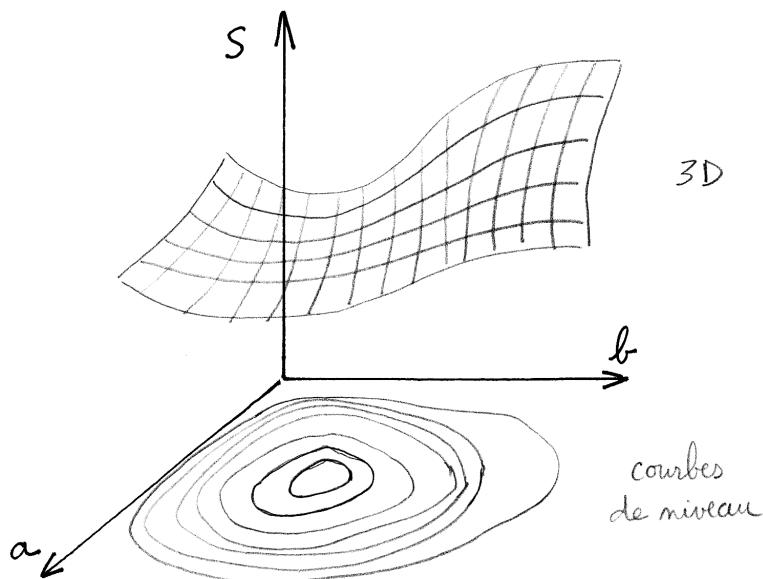


FIGURE 3.8 – Surface à minimiser correspondant au critère des moindres carrés

Un tel algorithme itératif qui consiste en fait à “linéariser” la surface par un plan à chaque itération (généralisation de la méthode de Newton, de recherche de zéros de fonctions), est très sensible aux approximations initiales des paramètres : si elles ne sont pas assez bonnes, on peut ne pas trouver la bonne solution. On peut facilement comprendre cela si par exemple, la fonction $\phi(a, b)$ n'est pas uniformément

convexe (elle change de convexité) et dans ce cas, elle risque de posséder plusieurs minima locaux comme représenté sur la figure 3.9.

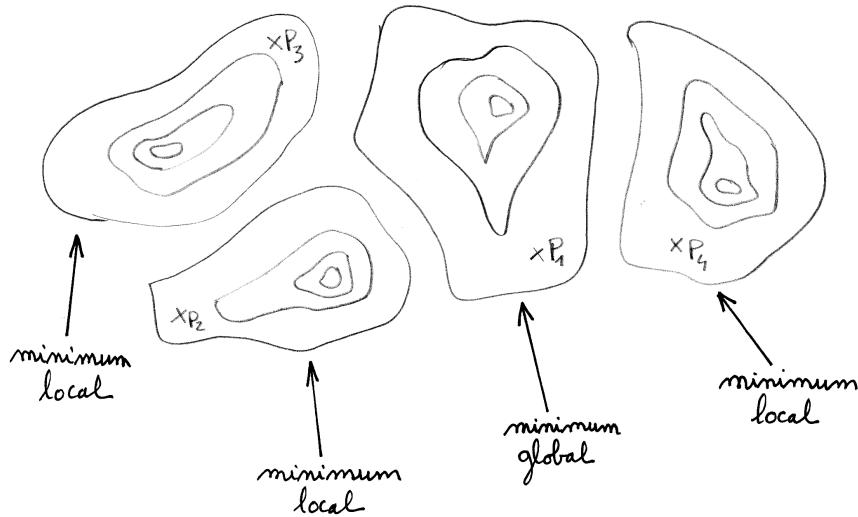


FIGURE 3.9 – recherche de minimum sur une fonction non uniformément convexe

Dans la figure précédente, si on part de P_1 , un algorithme itératif de type descente de gradient va donner le vrai minimum alors que si l'on part de $P - 2$, P_3 ou P_4 , l'algorithme trouvera un minimum local qui va attirer la recherche par descente de gradient.

Exemple 26 *Recherche “graphique” d'une identification d'une loi logistique avec Maple*

On étudie la croissance du fusarium, champignon microscopique qu'on trouve dans les sols¹.

Dans le tableau suivant, on a relevé l'effectif (en milliers d'individus) de fusarium sur un espace donné et pendant une période de 31 jours.

1. cet exemple est extrait du livre d'Alain Pavé, “Modélisation en biologie et en écologie”, éditions Aléas, 1994

t (en jours)	effectif (en milliers d'individus)
0	0,79
2	4,89
4	5,12
6	6,70
9	27,3
11	34,0
13	37,6
16	64,9
18	71,9
20	85,2
23	95
27	98
31	97,4

On modélise cette dynamique de croissance par une loi logistique. Sur les feuilles de calculs suivantes, écrites en Maple, on recherche graphiquement les paramètres de cette loi logistique qui n'est pas un modèle linéaire par rapport à ses paramètres d'identification. Cette feuille de calcul se termine également par un processus de validation qui est expliquée dans la sous-section suivante.

Exercice 5 Re-écrire cette feuille de calcul Maple et les traitements qu'elle contient avec Scilab.

3.4.2 Validation d'un modèle

Une fois que l'on a identifié les paramètres du modèle, de manière optimale (ou non ...), on se pose la question de savoir si le modèle est bien adapté aux données réelles.

Exemple 27 Validation sur une régression linéaire

Soit des données $(t_i, d_i)_{1 \leq i \leq n}$ et $\phi(t)$ un modèle. Sur la figure qui 3.10, on montre deux analyses possibles : la figure de gauche montre que le modèle est bien adapté et les points de données suivent globalement la droite de régression ; la figure de droite montre que le modèle est mal adapté et que même en ajustant au mieux la droite aux données, cette droite ne restitue pas l'évolution des données expérimentales qui suivent manifestement une autre loi.

Pour faire cette étude de validation, on utilise des tests statistiques sur les points $(d_i, \phi(t_i))_{1 \leq i \leq n}$, avec $(t_i, d_i)_{1 \leq i \leq n}$ les données et $\phi(t)$ le modèle.

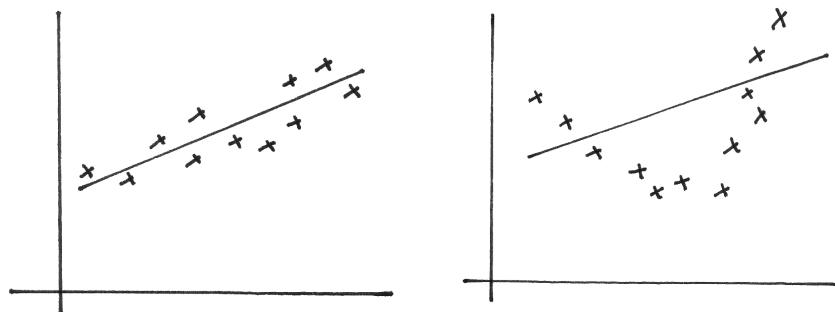


FIGURE 3.10 – Validation d'un calcul de régression linéaire. La validation est correcte sur la figure de gauche alors qu'elle ne l'est pas sur la figure de droite

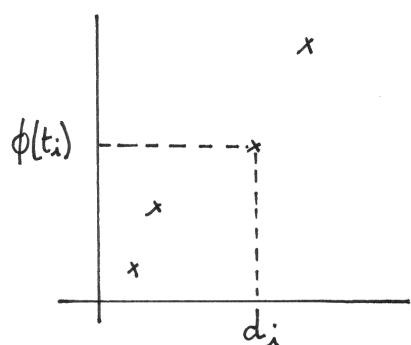


FIGURE 3.11 – Points de validation

Si le modèle est parfait, on a $\phi(t_i) = d_i$ et on obtient des points sur la droite d'équation $y = x$, soit la première diagonale.

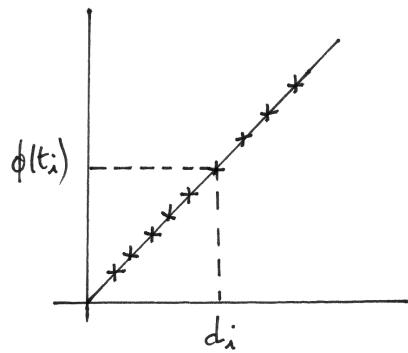


FIGURE 3.12 – Alignement parfait des points de validation

Dans le cas réel, les points ne sont pas exactement alignés et on calcul alors la droite de régression à partir de ces points $(d_i, \phi(t_i))_{1 \leq i \leq n}$.

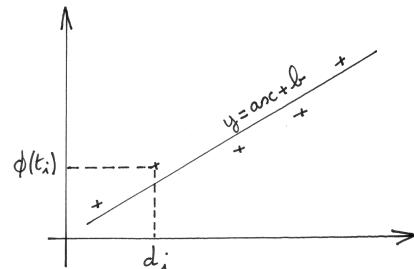


FIGURE 3.13 – Régression linéaire sur les points de validation

Si le modèle est bien adapté, alors la droite de régression, $y = ax + b$, a pour coefficients : a voisin de 1 et b voisin de 0.

On peut aussi calculer le coefficient de corrélation sur cette droite :

$$R = \frac{s_{td} - n \bar{t} \bar{d}}{n\sigma_t\sigma_d}$$

avec σ_t (resp. σ_d), l’écart-type définit par

$$\sigma_t = \sqrt{\frac{1}{n} s_{t^2} - \bar{t}^2}$$

On sait que $|R| \leq 1$,

- si $|R|$ est voisin de 1, alors la régression est correcte ;
- si R est voisin de 0, alors la régression est incorrecte.

Une autre analyse de validation peut être de regarder si l’écart entre le modèle et les mesures, $e_i = d_i - \phi(t_i)$, suit une loi de distribution normale $\mathcal{N}(0, \sigma)$.

Exemple 28 *Comme indiqué précédemment, une analyse de validation est faite sur la feuille de calcul Maple qui s’intéresse à la croissance du fusarium.*

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS

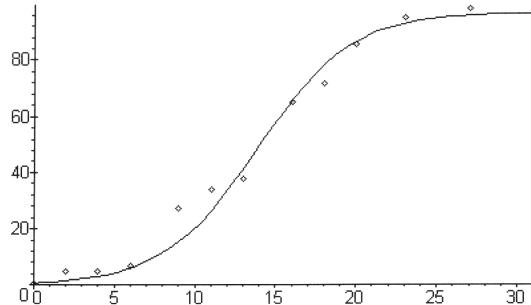
```

> restart;
> N0:=0.79;
N0 := .79
> # loi logistique
> N:=(Nm,gamma,t)->Nm/(1+(Nm/N0-1)*exp(-gamma*Nm*t));

$$N := (Nm, \gamma, t) \rightarrow \frac{Nm}{1 + \left| \frac{Nm}{N0} - 1 \right| e^{(-\gamma Nm t)}}$$

> # loi de Gompertz
> # N:=(Nm,gamma,t)->Nm*exp(ln(N0/Nm)*exp(-gamma*t));
> N := (Nm, \gamma, t) \rightarrow Nm e^{\left| \ln \left| \frac{N0}{Nm} \right| e^{(-\gamma t)} \right|}
> modele:=t->N(96.7,0.003585,t);
modele := t \rightarrow N(96.7, .003585, t)
> modplot:=plot(modele(t),t=0..31):
> readlib(readdata):
> data:=readdata(`d:/travail/maitbio/fusarium.dat`,2);
data := [[0, .790000000000000], [2., 4.8900000000000], [4., 5.1200000000000],
[6., 6.7000000000000], [9., 27.300000000000], [11., 34.],
[13., 37.600000000000], [16., 64.900000000001], [18., 71.900000000001],
[20., 85.200000000000], [23., 95.], [27., 98.], [31., 97.400000000001]]
> dataplot:=plot(data, style=point, symbol=diamond):
> display({modplot,dataplot});

```



```

> SomCar:=(Nm,gamma)->Sum((data[j][2]-N(Nm,gamma,data[j][1]))**2,j=1..nops(data));
nops(data)
SomCar := (Nm, \gamma) \rightarrow \sum_{j = 1}^{nops(data)} \left( data_j - N(Nm, \gamma, data_j) \right)^2
> evalf(SomCar(96.7,0.003585));
288.2603318

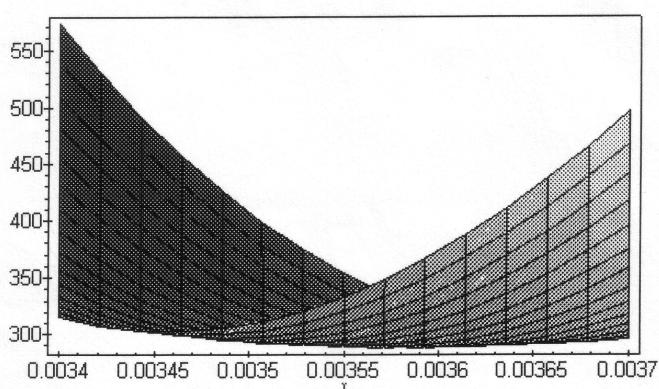
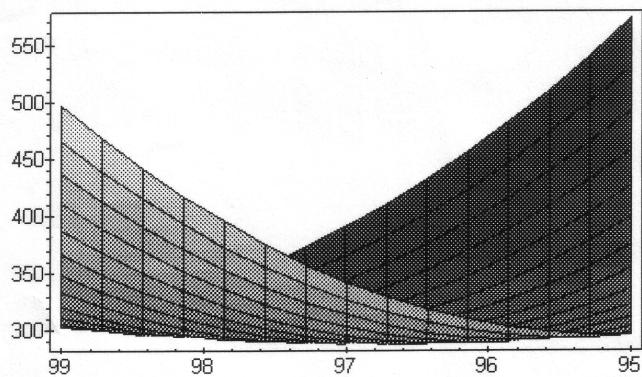
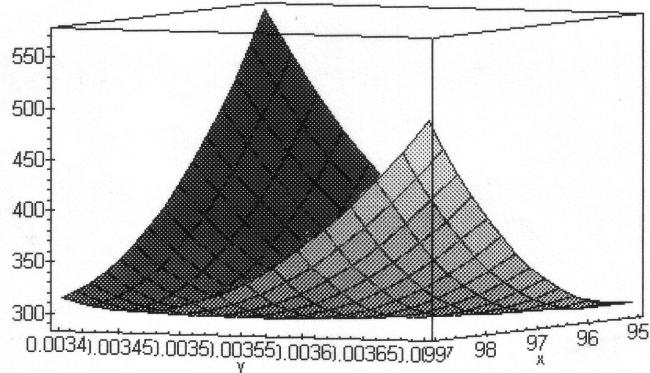
```

FIGURE 3.14 – Feuille de calcul Maple (1)

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS

```
> with(plots):
```

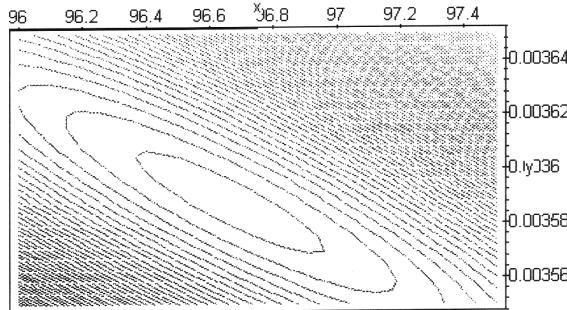
```
> plot3d(SomCar(x,y),x=95..99,y=0.0034..0.0037,grid=[15,15]);
```



```
> contourplot(SomCar(x,y),x=96..97.5,y=0.00355..0.00365,contours=50);
```

FIGURE 3.15 – Feuille de calcul Maple (2)

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS



```

> valid:= [seq( [ data[j][2], modele( data[j][1] ) ], j=1..nops(data) )];
    valid := [ [.790000000000000, .7899999999], [ 4.89000000000000, 1.567492410 ],
    [ 5.12000000000000, 3.085550725 ], [ 6.70000000000000, 5.981356053 ],
    [ 27.3000000000000, 15.20256882 ], [ 34., 26.27810605 ],
    [ 37.6000000000000, 41.33047843 ], [ 64.9000000000001, 65.62555530 ],
    [ 71.9000000000001, 78.19131372 ], [ 85.2000000000000, 86.46802812 ],
    [ 95., 92.81790986 ], [ 98., 95.69972845 ], [ 97.4000000000001, 96.44807329 ] ]
> validata:=plot(valid,style=point);
> read `d:/travail/maitbio/reglin.map`;
reglin := proc(tab)
local a,b,R,n,sx,sy,sxy,sx2,sy2;
n := nops(tab);
sx := sum(tab[i][1],i = 1 .. n);
sy := sum(tab[i][2],i = 1 .. n);
sxy := sum(tab[i][1]*tab[i][2],i = 1 .. n);
sx2 := sum(tab[i][1]^2,i = 1 .. n);
sy2 := sum(tab[i][2]^2,i = 1 .. n);
a := (sxy-sx*sy/n)/(sx2-sx^2/n);
b := (sy-a*sx)/n;
R := (sxy-sx*sy)/sqrt(sx2-sx^2/n)/sqrt(sy2-sy^2/n);
a,b,R
end
> rldata:=reglin(valid);
rldata := 1.029979153, -2.935748469, .9935943815
> freq:=x->rldata[1]*x+rldata[2];
freq := x → rldata1 x + rldata2
> regplot:=plot(freq(x),x=0..100);
> with(plots):display({regplot,validata});

```

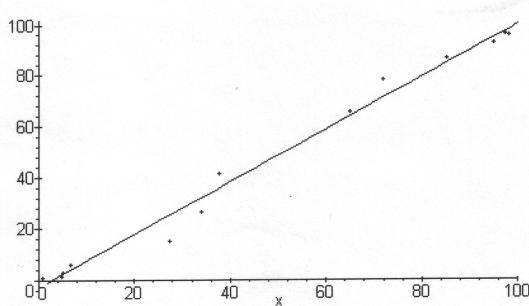


FIGURE 3.16 – Feuille de calcul Maple (3)

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS

On s'en tiendra à cette description relativement sommaire sur le problèmes de l'identification et de la validation. On aborde des problèmes parfois difficiles (en tout cas, qui déborde du cadre de ce cours) pour l'identification pouvant conduire à des méthodes d'optimisation sophistiquées². La validation fait appel à des techniques statistiques développées dans d'autres cours.

2. On pourra par exemple lire le petit fascicule de Yves Cherrault, “Optimisation, méthodes locales et globales” aux éditions PUF, 1999

3.5 Modèles discrets de dynamique de population

3.5.1 Construction d'un modèle discret et interprétation - application à la loi exponentielle

La description de l'évolution de l'effectif d'une population en fonction du temps a été faite précédemment à partir de la fonction $N(t)$ correspondant à l'effectif de la population, solution d'une équation différentielle.

Ce modèle peut être mis en œuvre si la fonction solution cherchée est continue et régulière, ce qui n'est pas toujours le cas. En effet l'étude de certaines populations font apparaître naturellement des discontinuités dues à des périodes de temps caractéristiques, telle que :

- un temps de dédoublement de cellules,
- un temps de gestation,
- ...

Il semble alors pertinent d'écrire directement le modèle sous forme discrète, en prenant des pas de temps correspondant à cette période caractéristique, plutôt que de conserver un schéma de calcul continu qui n'a plus de signification concrète.

Soit Δt une telle période de temps, partant d'une valeur initiale t_0 , on calculera les instants successifs

$$\begin{aligned} t_1 &= t_0 + \Delta t, \\ t_2 &= t_1 + \Delta t, \\ &\dots \\ t_n &= t_{n-1} + \Delta t = t_0 + n\Delta t. \end{aligned}$$

Connaissant $N(t_0) = N_0$, on va déterminer les valeurs ou approximations de $N_1 = N(t_1)$, $N_2 = N(t_2)$, ..., $N_n = N(t_n)$. On parle alors de *discrétisation*, la variable continue t est remplacée par une variable discrète qui prend une succession de valeurs t_i .

Reprendons la loi exponentielle, encore appelée loi de Malthus ; sa formulation provient de la relation précédemment établi :

$$\Delta N = kN(t)\Delta t$$

En posant $t = t_n$, $\Delta t = t_{n+1} - t_n$ et $\Delta N = N_{n+1} - N_n$, c'est à dire la différence de l'effectif à l'instant $n + 1$ et à l'instant n , on peut remplacer l'équation précédente par :

$$N_{n+1} = N_n + k\Delta t N_n$$

ou encore

$$N_{n+1} = N_n + RN_n \quad (3.1)$$

en notant $R = k\Delta t$ qui correspond au taux d'accroissement par individu et par pas de temps.

Cette formule de récurrence permet donc, à partir de N_0 , d'évaluer l'effectif de la population à chaque instant t_i , précédemment définis :

Exemple 29 On part de $N_0 = 10$ et on prend $R = 0.5$, alors on a

$$\begin{aligned} N_1 &= N_0 + RN_0 = 10 + 0.5 \cdot 10 = 15 \\ N_2 &= N_1 + RN_1 = 15 + 0.5 \cdot 15 = 22.5 \\ &\dots \end{aligned}$$

On peut interpréter schématiquement cette formule de récurrence en faisant apparaître les éléments agissant sur les coefficients de cette formule (voir la figure 3.17).

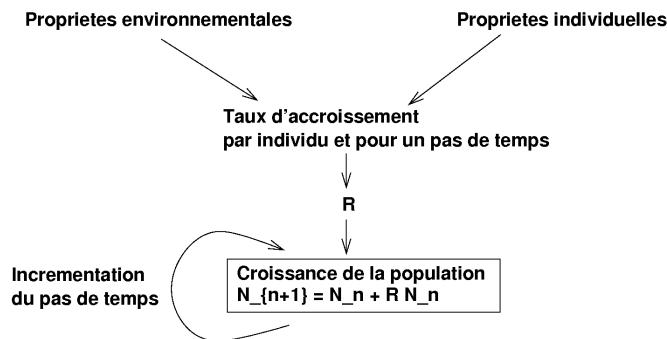


FIGURE 3.17 – Schéma de description du modèle de Malthus discret

3.5.2 Analyse d'un modèle discret - application à la loi logistique

Nous allons de la même manière que pour le loi exponentielle, passer à une discré-tisation de la loi logistique, c'est à dire, définir un pas de temps Δt et la calculer aux instants successifs

$$\begin{aligned} t_1 &= t_0 + \Delta t, \\ t_2 &= t_1 + \Delta t, \\ &\dots \\ t_n &= t_{n-1} + \Delta t = t_0 + n\Delta t. \end{aligned}$$

On définit donc la suite N_n comme étant les valeurs successives de la fonction $N(t)$ calculée à ces instants successifs. Soit $N_0 = N(t_0)$, $N_1 = N(t_1)$, ...,

CHAPITRE 3. MODÈLES D'ÉVOLUTION DES POPULATIONS

$$N_n = N(t_n).$$

Sous forme discrète, la loi logistique s'écrit :

$$\Delta N = \gamma(N^* - N(t))N(t)\Delta t$$

Soit en posant $t = t_n$, $\Delta t = t_{n+1} - t_n$ et $\Delta N = N_{n+1} - N_n$, c'est à dire la différence de l'effectif à l'instant $n+1$ et à l'instant n , on peut remplacer l'équation précédente par :

$$N_{n+1} = N_n + \gamma\Delta t(N^* - N_n)N_n$$

ou encore

$$N_{n+1} = N_n + RN_n$$

avec, ici

$$\begin{aligned} R &= \gamma\Delta t(N^* - N_n) \\ &= \gamma\Delta tN^* - \gamma\Delta tN_n \\ &= R_m - sN_n \end{aligned}$$

où $R_m = \gamma\Delta tN^*$ est un taux d'accroissement maximal par individu et pour un pas de temps et $s = \gamma\Delta t$ représente une régulation à comprendre comme une correction sur l'accroissement due aux interactions entre individus.

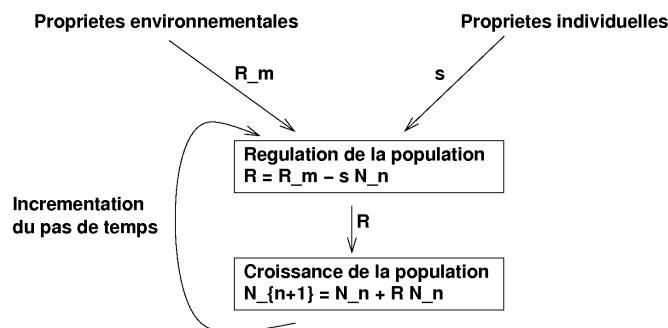


FIGURE 3.18 – Schéma de description du modèle logistique discret

Comparé au schéma correspondant à la loi exponentielle, on s'aperçoit qu'ici, le modèle est plus complexe du fait qu'il y a interaction avec le temps sur le calcul de la régulation de la population.

Exemple 30 *On part de $N_0 = 10$ et on prend $R_m = 1$ et $s = 0.001$, alors on a*

$$\begin{aligned} N_1 &= N_0 + (R_m - sN_0)N_0 = 10 + (1 - 0.001 \times 10) \times 10 = 19.9 \\ N_2 &= N_1 + (R_m - sN_1)N_1 = 19.9 + (1 - 0.001 \times 19.9) \times 19.9 = 39.4 \\ &\dots \end{aligned}$$

Pour effectuer le calcul sous Scilab :

1. On définit la fonction de récurrence

$$f(x) = (1 + R_m - sx)x$$

2. On définit une procédure qui construit la suite récurrente

$$u_n = f(u_{n-1})$$

avec u_0 donné

3. On calcule une séquence de points (i, u_i) avec $i = 0, \dots, 15$
4. On trace le graphique des points précédents.

(voir TP)

On retrouve bien une évolution comparable à la loi logistique continue. La courbe est régulière car le terme de correction due aux interactions entre individus est relativement faible et agit de manière progressive.

Lorsque l'on change ce terme de régulation, on peut faire apparaître des instabilités à l'approche du niveau d'équilibre et provoquer ainsi des phénomènes oscillatoires. Les différents cas d'évolution font l'objet du TP. On propose aussi, dans ce TP, d'introduire un terme de retard et de représenter des modifications de l'environnement. On observera alors des courbes qui correspondent pour la plupart aux observations qui ont été données au début de ce chapitre.

3.5.3 TP : Etude de modèles discrétilisés avec Scilab, à partir de la loi logistique

Chapitre 4

Modèles d'interaction de populations

Sommaire

4.1	Présentation des différents types d'interactions étudiées	58
4.2	Modèle de coopération de populations	59
4.3	Modèle de compétition de populations	66
4.4	Modèle de type proies-prédateurs	72
4.4.1	TP : Implémentation des modèles d'interaction de population avec Scilab	72
4.5	Modèles d'épidémiologie	77
4.5.1	Modèles SI	77
4.5.2	Modèles SIR	78
4.5.3	TP : Applications sous Populus	79

4.1 Présentation des différents types d'interactions étudiées

On s'intéresse dans ce chapitre, à l'évolution de la population de deux espèces distinctes qui co-existent sur un même territoire. Ces deux espèces interagissent soit à cause du partage de leur environnement commun, soit parce qu'il existe une relation de prédation entre elles.

On s'intéresse à 3 cas d'interaction :

- interactions coopératives entre deux population : le développement d'une population influe favorablement sur le développement de l'autre ;
- interactions compétitives entre deux populations : le développement d'une population se fait au détriment de l'autre, à cause du partage des ressources de l'environnement, par exemple ;

- interactions de type proies-prédateurs : la population des prédateurs croît de manière proportionnelle à l'effectif des proies et des prédateurs ; l'effectif des proies décroît de manière proportionnelle à l'effectif des proies et des prédateurs.

4.2 Modèle de coopération de populations

Soient deux populations d'effectifs $N(t)$ et $M(t)$. Le modèle de coopération proposé se construit à partir de la loi logistique appliquée à chacune des deux populations en y ajoutant un terme croisé caractérisant la coopération.

La loi logistique initiallement considérée pour la population $N(t)$ avant de prendre en compte la coopération s'écrit :

$$\frac{dN(t)}{dt} = \gamma_1(N^* - N(t))N(t)$$

On ajoute alors à l'expression $\gamma_1(N^* - N(t))$ qui caractérise l'accroissement, un terme supplémentaire qui aura pour effet d'augmenter ce facteur d'accroissement et qui est proportionnel à l'effectif de l'autre population, soit $\alpha_1 M(t)$ avec $\alpha_1 > 0$. Cela signifie donc que la présence de la population $M(t)$ augmente l'accroissement de la population $N(t)$.

La loi s'écrit alors :

$$\frac{dN(t)}{dt} = \gamma_1(N^* - N(t) + \alpha_1 M(t))N(t)$$

De la même manière, on va compléter la loi logistique décrivant la dynamique sur l'effectif de la population $M(t)$, en ajoutant au terme d'accroissement, l'expression $\alpha_2 N(t)$ signifiant que la présence de la population $N(t)$ fait augmenter l'accroissement de la population $M(t)$.

Le système complet couplant les deux équations s'écrit alors :

$$\begin{cases} \frac{dN(t)}{dt} = \gamma_1(N^* - N(t) + \alpha_1 M(t))N(t) \\ \frac{dM(t)}{dt} = \gamma_2(M^* - M(t) + \alpha_2 N(t))M(t) \end{cases}$$

On peut alors obtenir une simulation de l'évolution des deux populations, à partir de leurs effectifs initiaux respectifs : $N(0)$ et $M(0)$.

Dans un modèle discret tel que celui construit à partir de la loi logistique précédemment présenté :

$$N_{n+1} = N_n + (R - sN_n)N_n$$

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

avec :

N^* : effectif maximal ;

R : taux d'accroissement maximum par individu et pour le pas de temps $\Delta t = \gamma N^* \Delta t$;

s : coefficient d'interaction/régulation par individu $= \gamma \Delta t$.

Comme pour le modèle continu, on introduit dans ce modèle discret un terme traduisant le fait que pour chaque population, la présence de l'autre population coopérante va avoir un impact positif sur son développement. Soit les nouvelles équations des deux populations $N(t)$ et $M(t)$:

$$\begin{cases} N_{n+1} = N_n + (R_1 - s_1 N_n + k_1 M_n) N_n \\ M_{n+1} = M_n + (R_2 - s_2 M_n + k_2 N_n) M_n \end{cases}$$

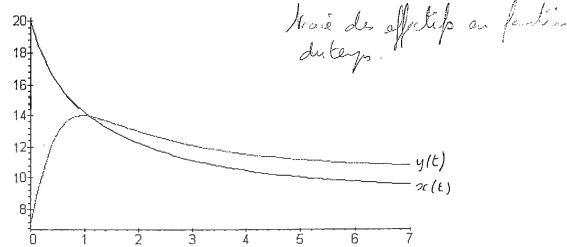
Des simulations sont données ci-après en Maple, à partir des formulations de modèles de coopération sous forme continue (vous êtes invités à reconstruire l'équivalent en Octave, à titre d'exercice.

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

```

> restart;gc();gc();
> a1:=0.05; a2:=6; a3:=0.3;
      a1 := .05
      a2 := 6
      a3 := .3
> b1:=0.2; b2:=4; b3:=0.7;
      b1 := .2
      b2 := 4
      b3 := .7
> f:=(x,y)->a1*(a2-x+a3*y)*x;
      f := (x, y) → a1 (a2 - x + a3 y) x
> g:=(x,y)->b1*(b2-y+b3*x)*y;
      g := (x, y) → b1 (b2 - y + b3 x) y
> e1:=diff(x(t),t)=f(x(t),y(t));
      e1 :=  $\frac{\partial}{\partial t} x(t) = .05 (6 - x(t) + .3 y(t)) x(t)$ 
> e2:=diff(y(t),t)=g(x(t),y(t));
      e2 :=  $\frac{\partial}{\partial t} y(t) = .2 (4 - y(t) + .7 x(t)) y(t)$ 
> sol:=dsolve({e1,e2,x(0)=20,y(0)=7},{x(t),y(t)},numeric);
      sol := proc(rkf45_x) ... end
> with(plots):
> odeplot(sol,[t,x(t)],t,y(t)],0..7,numpoints=50);

```



```
> odeplot(sol,[x(t),y(t)],0..30,numpoints=100);
```

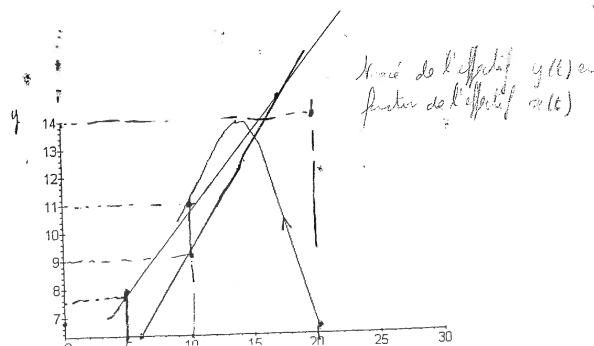


FIGURE 4.1 – Feuille de calcul Maple - modèle de coopération (1)

Commentaires sur les courbes obtenues

On visualise les évolutions de deux populations en interaction coopérative dont la première d'effectif $x(t)$ part d'un effectif initial égal à 20 et dont l'effectif "maximal" pour la partie du modèle reprenant la loi logistique correspond à $x_{max} = 6$. La seconde population $y(t)$ part d'un effectif initial égal à 7 et son effectif "maximal" pour la partie du modèle reprenant la loi logistique correspond à $y_{max} = 4$.

On voit que la population $y(t)$ se développe dès le début alors que sa valeur initiale (7) est déjà supérieure à $y_{max} = 4$. Ceci est du à la présence importante de la population $x(t)$ qui favorise son développement. Cette croissance de $y(t)$ diminuera lorsque la valeur de $x(t)$ va baisser. On constate finalement que les valeurs assymptotiques de $x(t)$ et $y(t)$ sont comprises entre 9 et 12 donc largement supérieures à x_{max} et y_{max} . En effet le terme d'interaction de population vient "corriger" la population maximale dans le modèle, comme on va l'expliquer.

Interprétation et évolutions comparées des deux populations

Si on reprend et réordonne les termes de l'équation de coopération discréétisée basée sur l'extension du modèle logistique, pour $N(t)$ on a :

$$N_{n+1} = N_n + ((R_1 + k_1 M_n) - s_1 N_n) N_n$$

Comparé à l'expression d'un loi logistique, cela revient à obtenir un nouvel effectif maximal corrigé par le terme d'interaction qui est l'expression mise en entre parenthèse :

$$R_1 + k_1 M_n$$

Ce terme que l'on va noter R_N n'est donc plus constant et va évoluer lorsque la population $M(t)$ évolue, comme on le visualise sur la courbe suivante :

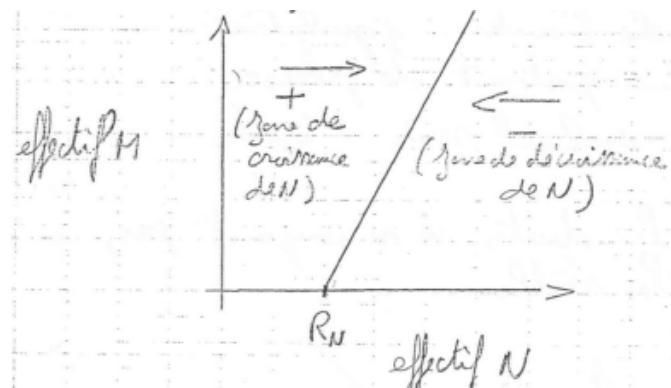


FIGURE 4.2 – Croissance relative de $N(t)$

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

Si on observe la seconde équation relative à l'évolution de la population $M(t)$, on a de la même manière :

$$M_{n+1} = M_n + ((R_2 + k_2 N_n) - s_2 M_n) M_n$$

Et on met de la même manière un nouvel effectif "maximal" pour la population $M(t)$ correspondant à :

$$R_M = R_2 + k_2 N_n$$

que l'on peut visualiser sur la courbe suivante :

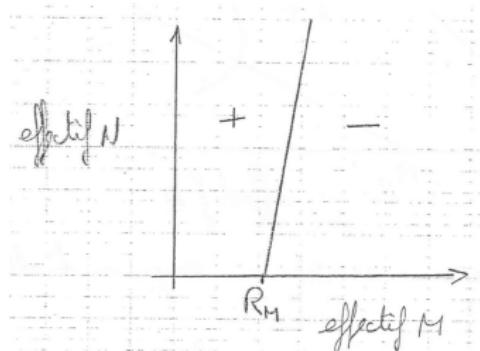


FIGURE 4.3 – Croissance relative de $M(t)$

Si nous reportons sur un même graphique ces deux droites caractérisant les deux effectifs "maximaux" pour les deux populations, on peut séparer le plan en 4 zones où il est possible de séparer les zones de croissances et de décroissances respectives des deux populations considérées, comme on le voit sur la figure suivante :

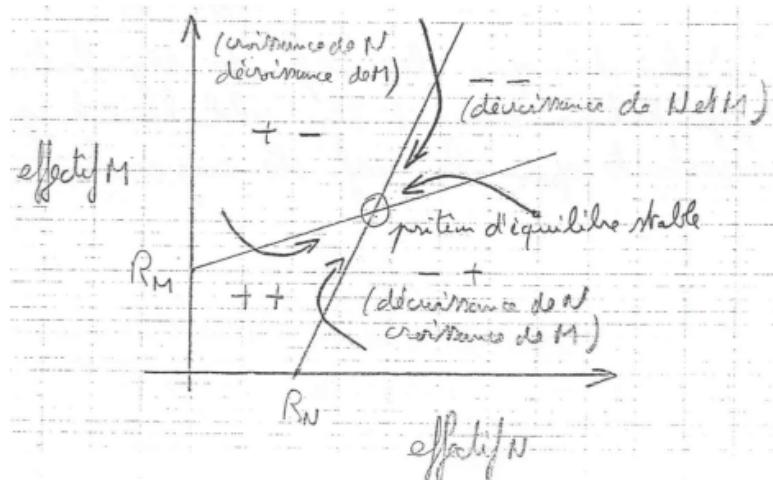


FIGURE 4.4 – Croissance relative de $N(t)$ et de $M(t)$

On peut ainsi prédire grâce aux derniers graphiques des trajectoires d'effectifs comparés entre les deux populations : l'un des axes correspond à l'effectif d'une population et l'autre axe à l'effectif de l'autre population. On appelle portrait de phase la représentation de ces trajectoires dans ce repère.

Dans la feuille de calcul Maple qui suit, on a tracé un certains nombre de trajectoires, relatives au modèle de coopération présenté sur les feuilles de calcul précédentes. On a ajouté à la main des tracés des droites R_N et R_M facilement identifiables car elles annulent l'une des dérivées des trajectoires (suivant x ou suivant y). On observe alors la convergence de ces trajectoires vers le point d'équilibre stable identifié dans la figure 4.4.

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

Modèles de coopération

(68)

```

> restart;gc();gc(0);
> a:=[0.05, 6, -1, 0.3];
      a := [.05, 6, -1, .3]
> b:=[0.2, 4, -1, 0.7];
      b := [.2, 4, -1, .7]
> e1:=diff(x(t),t)=a[1]*(a[2]+a[3]*x(t)+a[4]*y(t))*x(t);
      e1 :=  $\frac{\partial}{\partial t} x(t) = .05 (6 - x(t) + .3 y(t)) x(t)$ 
> e2:=diff(y(t),t)=b[1]*(b[2]+b[3]*y(t)+b[4]*x(t))*y(t);
      e2 :=  $\frac{\partial}{\partial t} y(t) = .2 (4 - y(t) + .7 x(t)) y(t)$ 
> solproc := proc(x0,y0)
      local nsol;
      nsol:=dsolve({e1, e2, x(0)=x0, y(0)=y0},{x(t),y(t)},numeric);
      odeplot(nsol,[x(t),y(t)],0..20,numpoints=100);
    end;
    solproc := proc(x0,y0)
      local nsol;
      nsol := dsolve({x(0) = x0, y(0) = y0, e2, e1}, {x(t), y(t)}, numeric);
      odeplot(nsol,[x(t),y(t)],0 .. 20,numpoints = 100)
    end
> xval:=[seq(i,i=1..20)];
      xval := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
> plot1:=map(solproc,xval,1):
> plot2:=map(solproc,xval,20):
> with(plots):
> p1:=display(plot1):
> p2:=display(plot2):
> display({p1,p2});

```

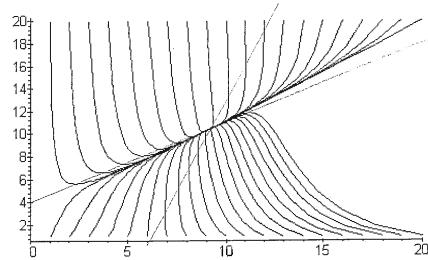


FIGURE 4.5 – Feuille de calcul Maple - modèle de coopération (2)

Un autre cas d'évolution croisé peut se produire si les droites R_N et R_M ne se coupent pas (cela dépend des coefficient directeurs de ces droites). On ne trouve pas de point d'équilibre stable et les deux populations n'atteignent pas un effectif de saturation et continuent à s'accroître de manière exponentielle, comme décrit dans la figure suivante.

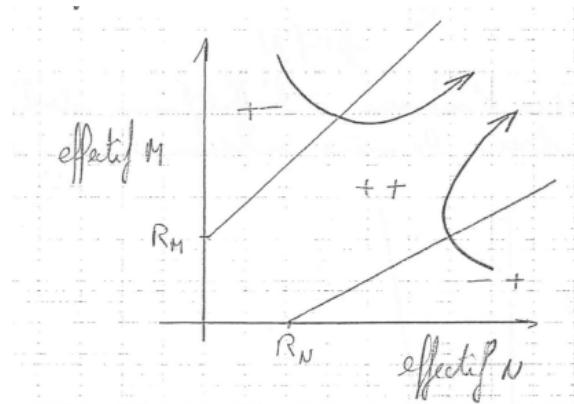


FIGURE 4.6 – Croissance relative de $N(t)$ et de $M(t)$ sans équilibre stable

4.3 Modèle de compétition de populations

Pour décrire un modèle de compétition entre deux populations qui évoluent séparément par des lois logistiques hors interaction, on va comme pour les modèles de coopération, ajouter un terme qui a pour but de faire décroître la croissance d'une population de manière proportionnelle à l'effectif de l'autre population.

Le terme additionnel qui était positif dans le modèle de coopération (et permettait de favoriser la croissance) va être identique mais négatif (pour pénaliser cette fois, la croissance).

On obtient donc les formulations suivantes pour la description en loi continues :

$$\begin{cases} \frac{dN(t)}{dt} = \gamma_1(N^* - N(t) - \alpha_1 M(t))N(t) \\ \frac{dM(t)}{dt} = \gamma_2(M^* - M(t) - \alpha_2 N(t))M(t) \end{cases}$$

Et pour la formulation en lois discrètes :

$$\begin{cases} N_{n+1} = N_n + (R_1 - s_1 N_n - k_1 M_n)N_n \\ M_{n+1} = M_n + (R_2 - s_2 M_n - k_2 N_n)M_n \end{cases}$$

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

On présente ci-après des simulations faites à partir des lois continues en Maple :

(74)

```
> restart;gc();gc(0); # modèle de compétition entre 2 populations
> a1:=0.04; a2:=30; a3:=1.1;
               a1 := .04
               a2 := 30
               a3 := 1.1
> b1:=0.1; b2:=20; b3:=0.7;
               b1 := .1
               b2 := 20
               b3 := .7
> e1:=diff(x(t),t)=a1*(a2-x(t)-a3*y(t))*x(t);
e1 :=  $\frac{d}{dt} x(t) = .04 (30 - x(t) - 1.1 y(t)) x(t)$ 
> e2:=diff(y(t),t)=b1*(b2-y(t)-b3*x(t))*y(t);
e2 :=  $\frac{d}{dt} y(t) = .1 (20 - y(t) - .7 x(t)) y(t)$ 
> sol:=dsolve({e1,e2,x(0)=1.,y(0)=10.},{x(t),y(t)},numeric);
sol := proc(rkf45_x) ... end
> with(plots):
> odeplot(sol,[[t,x(t)], [t,y(t)]],0..50);
```

FIGURE 4.7 – Feuille de calcul Maple - modèle de compétition (1)

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

(72)

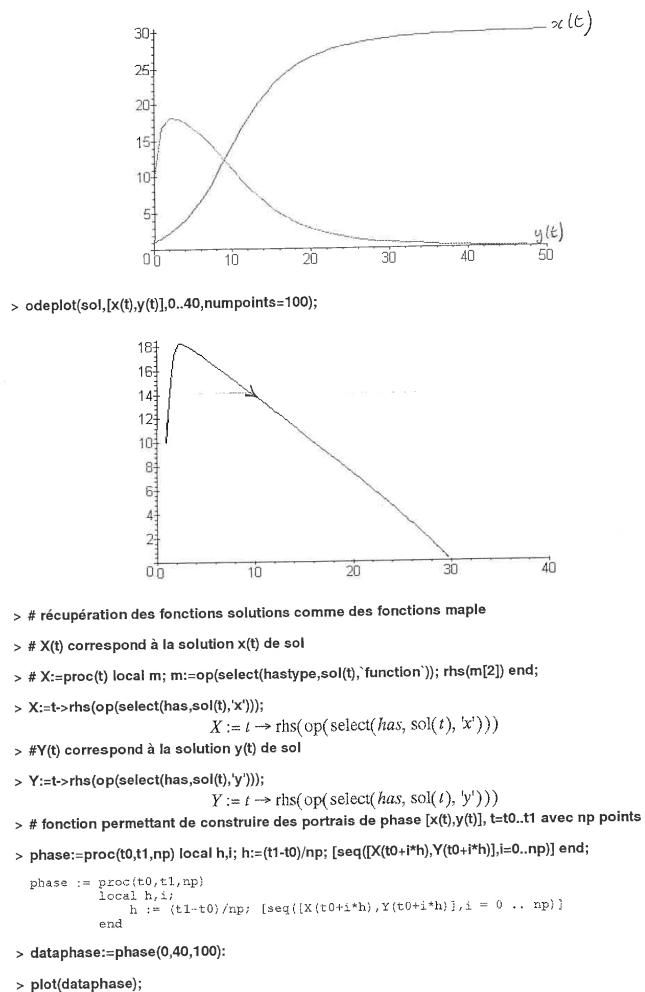
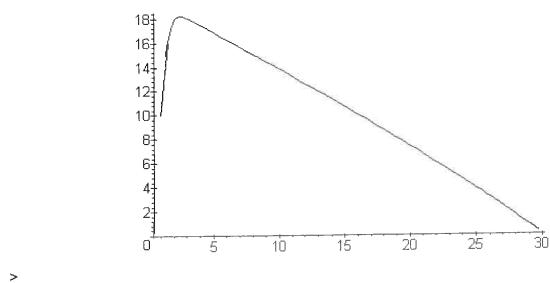


FIGURE 4.8 – Feuille de calcul Maple - modèle de compétition (2)

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

(73)



>

autres ensembles

$$(S1) \quad \begin{cases} \frac{dx}{dt} = 5x(1-x-y) \\ \frac{dy}{dt} = y(1-0.5x-0.5y) \end{cases}$$

$$(S2) \quad \begin{cases} \frac{dx}{dt} = x(1-2x-2y) \\ \frac{dy}{dt} = 5y(1-1.5x-1.5y) \end{cases}$$

en CT quadratique

$$\begin{array}{ll} x_0 = 1 & y_0 = 1 \\ x_0 = 1 & y_0 = 2 \\ x_0 = 1 & y_0 = 3 \\ x_0 = 10 & y_0 = 2 \\ x_0 = 10 & y_0 = 3 \\ x_0 = 10 & y_0 = 4 \\ \vdots & \vdots \end{array}$$

en CT quadratique

$$\begin{array}{ll} x_0 = 1 & y_0 = 1 \\ x_0 = 2 & y_0 = 1 \\ x_0 = 10 & y_0 = 3 \\ x_0 = 10 & y_0 = 4 \\ x_0 = 9 & y_0 = 6 \\ x_0 = 7 & y_0 = 8 \\ \vdots & \vdots \end{array}$$

application en environnement naturel

$$\begin{cases} \frac{dx}{dt} = \text{minut} > 0 \text{ alors } 5x(1-x-y) \text{ sinon } x(1-2x-2y) \\ \frac{dy}{dt} = \text{minut} > 0 \text{ alors } y(1-0.5x-0.5y) \text{ sinon } 5y(1-1.5x-1.5y) \end{cases}$$

periodique $\omega = 2 \pi / \sqrt{3.27}$ puis h puis $6 \pi / \sqrt{3.27}$

FIGURE 4.9 – Feuille de calcul Maple - modèle de compétition (3)

Commentaires sur les courbes obtenues

Sur l'exemple traité, on s'aperçoit que le mécanisme de compétition conduit à faire disparaître l'une des population pendant que l'autre suit une évolution qui est assez proche d'une loi logistique.

Cette évolution globale du système des deux populations qui conduit à la disparition d'une des deux populations, n'est pas la seule issue possible et dans certaines situations, les deux populations peuvent être préservées mais avec une diminution de l'effectif "maximal" initial extrait de la partie loi logistique dans le modèle (c'est à dire en annulant le terme d'interaction).

L'étude suivante permet de distinguer ces deux types d'issues.

Interprétation et évolutions comparées des deux populations

Comme nous l'avions fait pour le modèles de coopération, on peut reformuler les équations pour identifier des nouveaux coefficients d'effectif maximal qui évoluent en fonction de l'effectif de la population avec laquelle on interagit :

$$N_{n+1} = N_n + ((R_1 - k_1 M_n) - s_1 N_n) N_n$$

Comparé à l'expression d'une loi logistique, cela revient à obtenir un nouvel effectif maximal corrigé par le terme d'interaction qui est l'expression mise entre parenthèse :

$$R_1 - k_1 M_n$$

Ce terme que l'on va noter R_N n'est donc plus constant et va évoluer lorsque la population $M(t)$ évolue.

De la même manière on réécrit l'équation discrète de l'évolution de l'effectif $M(t)$:

$$M_{n+1} = M_n + ((R_2 - k_2 N_n) - s_2 M_n) M_n$$

et on obtient un nouvel effectif "maximal" pour $M(t)$ corrigé par ce terme d'interaction :

$$R_M = R_2 - k_2 N_n$$

On visualisant les droites caractérisant ces deux effectifs "maximaux" sur un même graphique, on sépare le plan en 4 zones de croissances et décroissances des deux population et on peut prédire le comportement des trajectoires du système formé par ces deux populations en compétition.

Suivant les pentes relatives de ces deux droites, on obtient des évolutions asymptotiques différentes correspondant à des issues où l'on préserve les deux populations ou bien l'une des populations disparaît, comme on l'observe sur les graphiques

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

suivants :

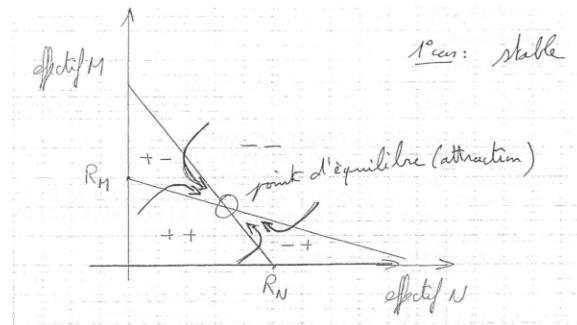


FIGURE 4.10 – Modèle de compétition - croissance relative de $N(t)$ et de $M(t)$ avec préservation des 2 populations

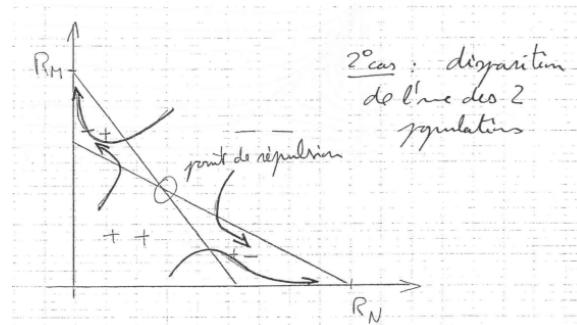


FIGURE 4.11 – Modèle de compétition - croissance relative de $N(t)$ et de $M(t)$ avec disparition d'une des 2 populations

4.4 Modèle de type proies-prédateurs

4.4.1 TP : Implémentation des modèles d'interaction de population avec Scilab

Dans un modèle de type proies-prédateurs où $x(t)$ représente l'effectif des proies et $y(t)$ celui des prédateurs, on fait les hypothèses suivantes :

- l'équation d'évolution de la population des proies suit les règles :
- en l'absence de prédateur, la population des proies se développe en suivant une loi logistique positive ;
- en présence de prédateurs, un terme de prédation affecte le développement des proies car elles sont consommées. Ce terme de prédation est proportionnel à la fois au nombre de proies et au nombre de prédateurs.

L'équation différentielle qui décrit l'évolution des proies s'écrit alors :

$$\frac{dx(t)}{dt} = ax(t) - bx(t)y(t)$$

- l'équation d'évolution de la population des prédateurs suit les règles :
- en l'absence de proie, la population des prédateurs se développe en suivant une loi logistique négative car elle n'a plus de ressources nutritives ;
- en présence de proies, un terme de prédation favorise le développement des prédateurs qui se nourrissent des proies. Ce terme de prédation est proportionnel à la fois au nombre de proies et au nombre de prédateurs.

L'équation différentielle qui décrit l'évolution des prédateurs s'écrit alors :

$$\frac{dy(t)}{dt} = -cx(t) + dx(t)y(t)$$

Le système de ces deux équations, connus sous le nom de systèmes de Lokta-Volterra est extrêmement populaire et trouve des applications au-delà de la seule biologie des populations mais sert d'inspiration à beaucoup de modèles variés (écosystème d'entreprises par exemple, ...).

Dans la suite, on donne une feuille de calcul en Maple qui simule le système de proies-prédateurs avec des tracés du développement des populations en fonction du temps mais aussi avec le tracé des portraits de phase qui sont très caractéristiques et présentent des cycles limites.

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

(78)

```

> # exemple de courbes multiples : odeplot(p, [[x,y(x)],[x,z(x)]],t=4..4,numpoints=25);
> # modèle proie-prédateur (Abell, Braselton)
> f:=(x,y) ->a*x-b*x*y;

$$f := (x, y) \rightarrow a x - b x y$$

> g:=(x,y)->-c*y+d*x*y;

$$g := (x, y) \rightarrow -c y + d x y$$

> eq_1:=diff(x(t),t)=f(x(t),y(t));

$$eq_1 := \frac{\partial}{\partial t} x(t) = a x(t) - b x(t) y(t)$$

> eq_2:=diff(y(t),t)=g(x(t),y(t));

$$eq_2 := \frac{\partial}{\partial t} y(t) = -c y(t) + d x(t) y(t)$$

> # résolution numérique des équations de Volterra
> e1:=subs(a=2,b=1,eq_1);

$$e1 := \frac{\partial}{\partial t} x(t) = 2 x(t) - x(t) y(t)$$

> e2:=subs(c=3,d=1,eq_2);

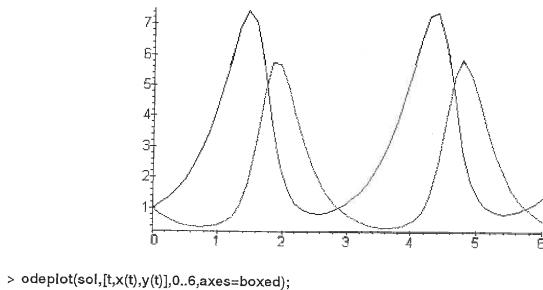
$$e2 := \frac{\partial}{\partial t} y(t) = -3 y(t) + x(t) y(t)$$

> sol:=dsolve({e1,e2,x(0)=1,y(0)=1},{x(t),y(t)},numeric);

$$sol := \text{proc}(z) \dots \text{end}$$

> with(plots):
> odeplot(sol,[t,x(t)],t=0..6);

```



> `odeplot(sol,[t,x(t)],t=0..6,axes=boxed);`

FIGURE 4.12 – Feuille de calcul Maple - modèle de proies-prédateurs (1)

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

(12)

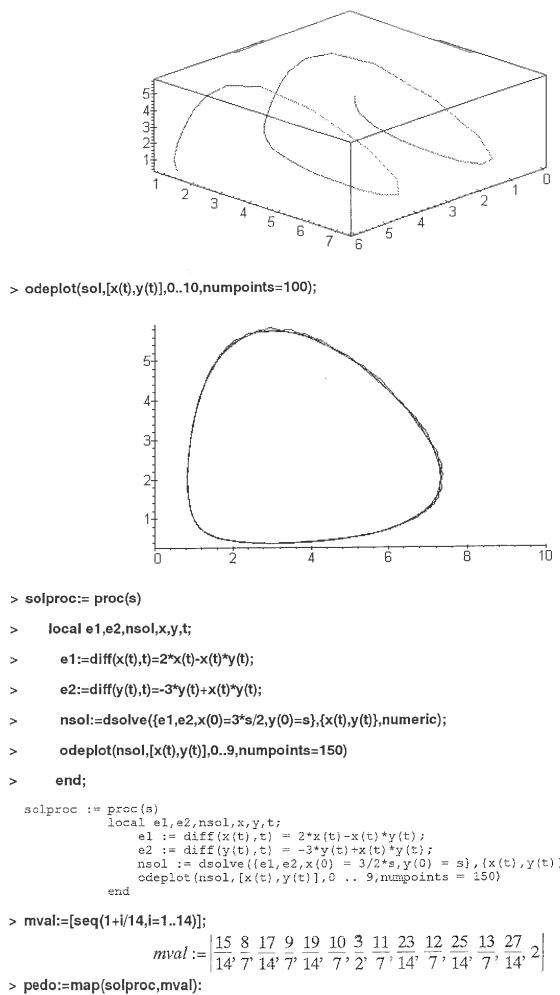


FIGURE 4.13 – Feuille de calcul Maple - modèle de proies-prédateurs (2)

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

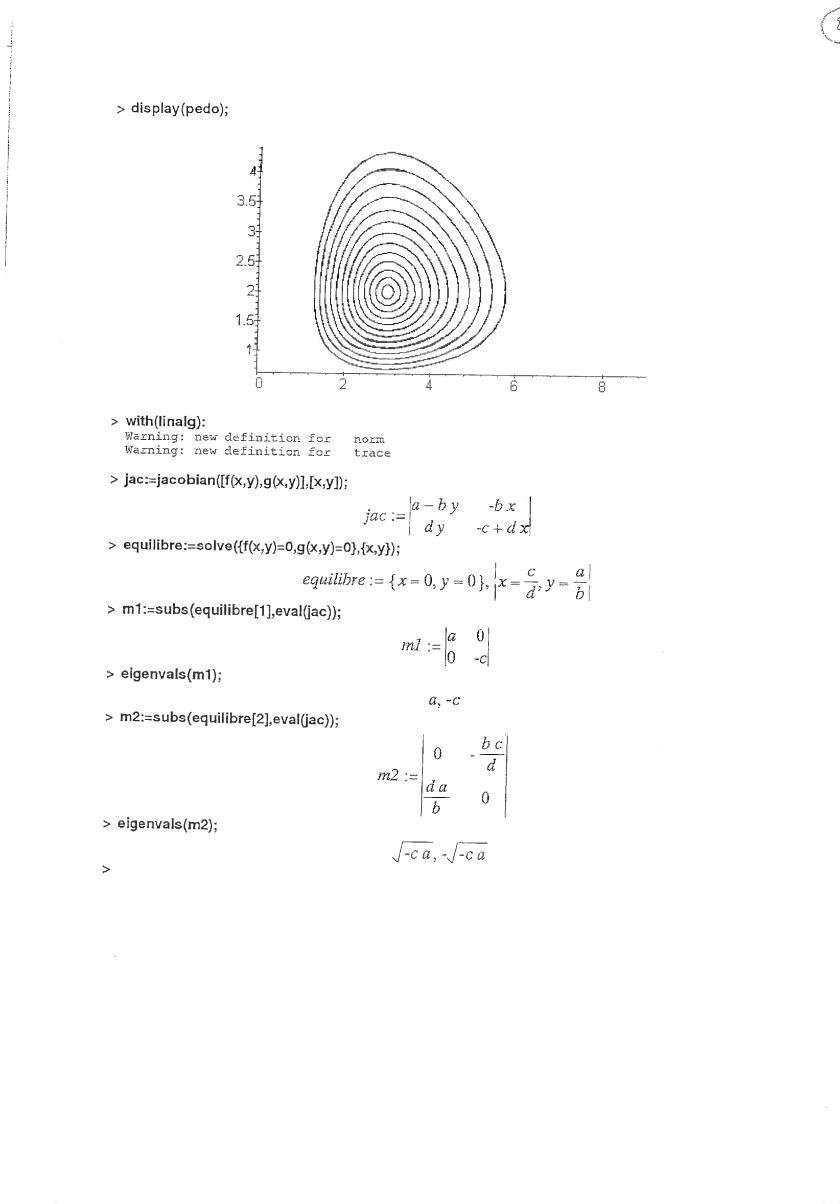


FIGURE 4.14 – Feuille de calcul Maple - modèle de proies-prédateurs (3)

CHAPITRE 4. MODÈLES D'INTERACTION DE POPULATIONS

Ces systèmes biologiques ont été utilisés pour essayer de résoudre de manière "naturelle" des régulations de populations d'espèces considérées comme nuisibles, tels que des lapins sauvages qui pouvaient ravager des cultures lorsqu'ils se développaient en trop grand nombre. Une des solutions suggérées a été d'introduire des prédateurs de ces nuisibles de manière artificielle. Sur le schéma suivant on s'aperçoit que l'introduction de ces prédateurs à un mauvais moment dans le cycle de développement du système peut provoquer un effet inverse de celui escompté en augmentant la population des nuisibles à certaines périodes.

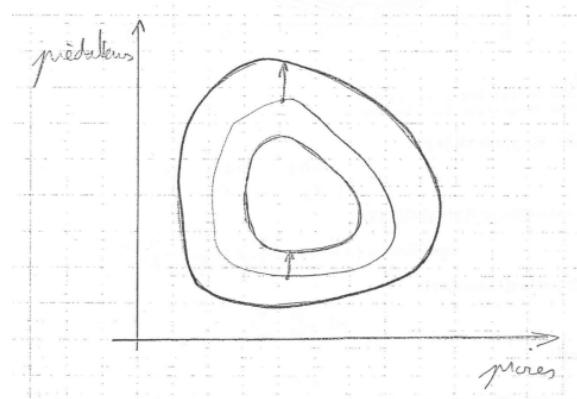


FIGURE 4.15 – Modèle proies-prédateurs - Modification du cycle par action sur l'effectif des prédateurs

L'action sur les écosystèmes reste une opération très délicate car on ne maîtrise pas toujours l'ensemble des chaînes de prédatations que l'on peut déséquilibrer de manière incontrôlable.

4.5 Modèles d'épidémiologie

On étudie ici les modèles d'infections de population par une maladie qui est portée par des agents infectieux (microbes, bactéries, virus, ...)

On se place dans le cadre d'étude où les agents infectieux sont modélisés comme des micro-parasites au sens où la dynamique de ces agents est négligée : ils sont présents mais ni propre effectif, ni la dynamique de la population qu'ils constituent ne sont représentés en tant que tels.

Par opposition à la manière de modéliser le problèmes dans la suite, la désignation de macro-parasites ferait référence à la nécessité d'étudier leur propre dynamique. On se retrouve alors sur des modèles proches dans leur conception des modèles de type proies-prédateurs sans qu'il y ait une véritable “consommation” des proies par ces agents infectieux.

Les modèles que l'on va étudier ici sont principalement dus à Anderson et May (1979, 1982), d'après des travaux plus anciens dus à Ross (1916, 1917), Kermach et Mc Kendrick (1927).

4.5.1 Modèles SI

Dans ce modèle, la population d'effectif $N(t)$, qui subit l'action de ces agents infectieux, est constituée de deux sous-catégories :

- Les individus susceptibles d'être infectés d'effectif $S(t)$ mais non encore infectés ;
- Les individus déjà infectés d'effectif $I(t)$ et qui sont alors contagieux, c'est à dire capables de transmettre la maladie à d'autres individus.

Dans le modèle SI, les hypothèses sont les suivantes :

- Les individus ne sont pas infectés à la naissance ;
- Les individus infectés transmettent la maladie de manière proportionnelle à leur effectif et à l'effectif des individus susceptibles d'être infectés ;
- Le taux de mortalité est plus grand chez les individus infectés que chez ceux qui ne le sont pas ;
- Les individus infectés peuvent guérir (avec un taux de guérison constant) et donc redevenir susceptibles d'être infectés.

On peut représenter graphiquement la dynamique de ce système en représentant les 2 compartiments homogènes correspondant aux deux catégories, S et I (voir la figure 4.16).

On en déduit alors le système différentiel suivant :

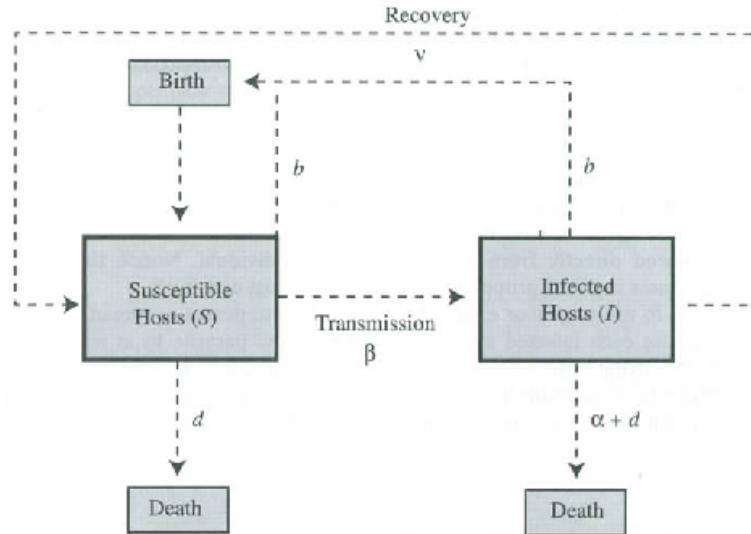


FIGURE 4.16 – Modèle SI (extrait du livre “Populus” de Don Alstad)

$$\begin{cases} \frac{dS(t)}{dt} = b(S(t) + I(t)) + \nu I(t) - dS(t) - \beta S(t)I(t) \\ \frac{dI(t)}{dt} = \beta S(t)I(t) - (\alpha + d + \nu)I \end{cases}$$

4.5.2 Modèles SIR

La limitation du modèle SI provient du fait que l'on sait qu'un élément important du phénomène d'infection est qu'un individu qui guérit passe d'abord par une phase d'immunité qui le protège d'une nouvelle contamination. L'ajout du modèle SIR porte sur le fait qu'une troisième catégorie de sous-population est introduit et correspond aux individus qui sont ainsi immunisés. On note $R(t)$ l'effectif de ces individus immunisés et les hypothèses supplémentaires sont les suivantes :

- Les individus qui ont été infectés peuvent guérir et développent alors sont alors immunisés et ainsi protégés d'attraper à nouveau l'infection.
- La population d'immunisés voit une partie de ses membres perdre leur immunité et passent donc dans l'état susceptible d'être infecté.

Le nouveau schéma du modèle SIR est donc donné par la figure 4.17.
On en déduit alors le système différentiel suivant :

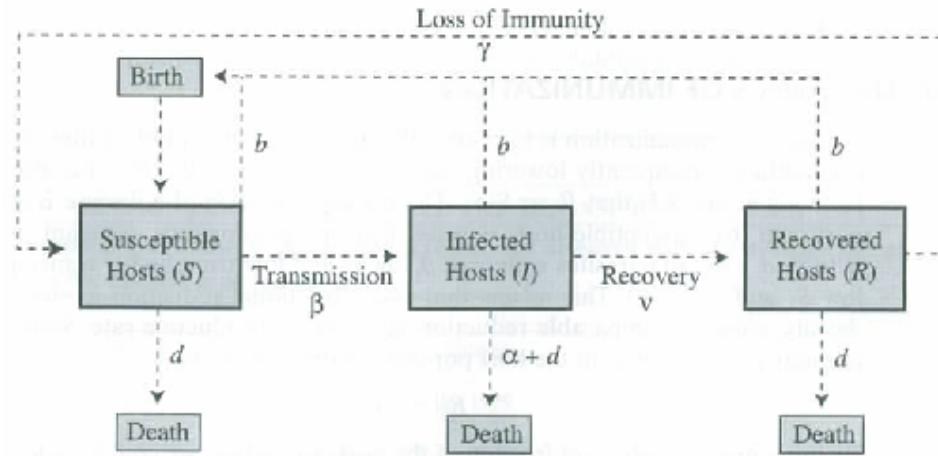


FIGURE 4.17 – Modèle SIR (extrait du livre “Populus” de Don Alstad)

$$\left\{ \begin{array}{lcl} \frac{dS(t)}{dt} & = & b(S(t) + I(t) + R(t)) + \gamma R(t) - dS(t) - \beta S(t)I(t) \\ \frac{dI(t)}{dt} & = & \beta S(t)I(t) - (\alpha + d + \nu)I \\ \frac{dR(t)}{dt} & = & \nu I(t) - (d + \gamma)R(t) \end{array} \right.$$

4.5.3 TP :Applications sous Populus

Chapitre 5

Modèles individus-centrés et application sous Netlogo

Sommaire

5.1	Introduction aux approches systémiques et à la modélisation comportementale	80
5.2	TP : Introduction et implémentation en Netlogo	80

5.1 Introduction aux approches systémiques et à la modélisation comportementale

5.2 TP : Introduction et implémentation en Netlogo

Chapitre 6

Modélisation par conception graphique

Sommaire

6.1	Modèles compartimentaux	81
6.2	TP : Diagrammes de Forrester et implémentation avec le module “systèmes dynamiques” de Netlogo	81

6.1 Modèles compartimentaux

6.2 TP : Diagrammes de Forrester et implémentation avec le module “systèmes dynamiques” de Netlogo

CHAPITRE 6. MODÉLISATION PAR CONCEPTION GRAPHIQUE

... à compléter