R

Xavier Thibert-Plante* [1]

*McGill University

16 février, 2010

[1] xavier.thibert-plante@mail.mcgill.ca

Introduction

Ce qui ne sera pas couvert aujourd'hui

- Statistique
- Design expérimental

Ce qui sera couvert aujourd'hui

- Rudiments de la programmation de haut niveau
- Outils pour en apprendre plus par vous-même

Introduction

Ce que R ne fera pas pour vous

- Entrer vos données
- Ne vous avertira pas si vous utilisez le mauvais test statistique

Ce que R fera pour vous

- Les tests statistiques
- Figures

Introduction

Quitter

```
> q()
```

ou

```
> quit()
```

Vous pouvez sauver votre session (variables et fonctions) pour la continuer plus tard.

Introduction

Initialisation

- Création d'un répertoire <votreNom> sur le bureau
- Ouvrir R
- Changer le répertoire pour <votreNom> en utilisant File->Change dir
- Ouvrir un éditeur de texte, Notepad, afin d'y écrire vos commandes

Introduction

R et interface graphique

- R est un programme en ligne de commande, la souris n'est pas utile.
  - Avantage : script
  - Inconvénient : quand on ne sait pas quoi écrire, on se sent seul.

Variables

- Scalaire

  ```
  > a <- 1
  ```

  équivalent à `a=1`

  ```
  > 1 -> a
  ```

- Un vecteur

  ```
  > b <-c(1,2,3)
  ```

Variables

Vide

- Vecteur

```
> a <- array(NA,dim=10)
> a[4] <- 5
```

- Matrice

```
> b <- matrix(NA,ncol=10,nrow=30)
> b[30,3]<-1
```

Variables

Aide

RTFM

```
> help(array)
> help(matrix)
```

Variables

Générer des vecteurs et des matrices

- Séquence (vecteur)

```
> a<-array(seq(1,10,2))
```

- Vecteur aléatoire

```
> a<-array(rnorm(10,mean=15,sd=3))
```

- Séquence (matrice)

```
> b<-matrix(seq(1,20),ncol=2,nrow=10)
```

- Matrice aléatoire

```
> b<-matrix(runif(21),ncol=3,nrow=7)
```

Variables

Exercice

- Générez un vecteur de 10000 élements composé de deux distributions normales de moyenne 10 et 15 et qui ont chacune une variance de 16.

Variables

Opérations arithmétiques simples

```
> a+a
> a+5
> 1+b
> 5*a
> a*a
```

```
> a*b
> b*b
> a-a
> a-5
> a/2
```

Variables

Anatomie

- x est une matrice avec des lignes et des colonnes

```
> x[,1] # première colonne
> x[,2] # deuxième colonne
> x[1,] # première ligne
```

- x[ligne,colonne]

Donnée

Lire les données

Visitez ma page web

- http://sites.google.com/site/xavierthibertplante
- R workshop
- Téléchargez la base de données dans le répertoire <votreNom>

Donnée

Modifier la base de données

- Enlever tous les caractères spéciaux (#$%&?+=-)
- Éviter les espaces dans les noms première ligne (’colOne’ vs ’col one’)
- Save as CSV (Coma Separated Variable)

Donnée

Saisie de données

- Utiliser un tableur
- Sauver sous le format CSV (coma separated variable)
- Regarder votre fichier sous un éditeur de texte (Notepad)
- Chacune des colonnes doit seulement avoir un type de donnée (sauf la première ligne)
- Télécharger la version CSV de ma page web http://sites.google.com/site/xavierthibertplante

Donnée

Importer les données

- Ouvrir R

```
> x<-read.csv("hendryetAl.csv")
```

- Maintenant la base de données s'appelle x

Variables

Anatomie

- x est une matrice avec des lignes et des colonnes

```
> x[,1] # première colonne
> x[,2] # deuxième colonne
> x[1,] # première ligne
```

- x[ligne,colonne]

Donnée

Vérifications

- Nombre de lignes dans le fichier et dans R

  ```
  > length(x[,1])
  ```

- Nombre de colonnes:

  ```
  > length(x[1,])
  ```

Donnée

Nom des colonnes

- Syntaxe : <variableName>$<columnName>

  ```
  > x$Years
  ```

  équivalent à

  ```
  > x[,18]
  ```

- Attention aux majuscules:

  ```
  > x$years
  ```

  ne fonctionne pas

Donnée

Votre base de données dans R

- Première colonne: nom (pas d'espace ni de caractères spéciaux dans le nom)
- Chaque colonne est d'un seul type
- Sauver en 'csv'
- Regarder le fichier dans un éditeur de texte (vérifier la séparation ";" ou "," et le point "." ou ",")
- Adapter le `read.csv` en fonction des données
  `> help(read.csv)`
- importer vos données >
  `votreNom<-read.csv("nomfichier.csv")`
- Tester le nombre de colonnes et de lignes

Manipulations à une variable

graphique

- Importer la base de données

```
> x<-read.csv("hendryetAl.csv")
```

- Histogramme des Haldanes

```
> hist(x$Haldanes)
```

- Changer le nombre de barres

```
> hist(x$Haldanes,breaks=100)
```

- Spécifier la position des barres

```
> hist(...,
breaks=seq(from=-1.2,to=0.8,by=0.1))
```

Manipulations à une variable

graphique

- Changer la couleur des barres

  ```
  > hist(...,col=2)
  ```

- Changer la couleur de contour des barres

  ```
  > hist(...,border=3)
  ```

Manipulations à une variable

Test de normalité

- Shapiro-Wilk normality test

```
> shapiro.test(x$Haldanes)
```

- Kolmogorov-Smirnov test

```
> ks.test(x$Haldanes,"pnorm",
  mean=mean(x$Haldanes,na.rm=T),
  sd=sd(x$Haldanes,na.rm=T))
```

- Retourne les valeurs et non leur sens.

Tableaux

- Importer une nouvelle base de données

```
> mydata<-read.csv("fruits.csv")
```

- Extraire l'information

```
> str(mydata)
> help(aggregate)
> aggregate(mydata$Fruit,
list(mydata$Manger), mean)
> aggregate(mydata$Fruit,
list(Manger=mydata$Manger), mean)
> aggregate(x=mydata$Fruit,
by=list(Manger=mydata$Manger), FUN=mean)
```

et non

```
> aggregate(mydata$Fruit, mydata$Manger,
mean)
> help(aggregate)
```

Tableaux

Sommaire

- Les données

```
> n.fruit <- aggregate(mydata$Fruit,
list(Manger=mydata$Manger), length)
> mean.fruit <- aggregate(mydata$Fruit,
list(Manger=mydata$Manger), mean)
> sd.fruit <- aggregate(mydata$Fruit,
list(Manger=mydata$Manger), sd)
```

- Construire le tableau

```
> summary.table <- cbind(n.fruit[,2],
mean.fruit[,2], sd.fruit[,2])
> summary.table
```

- Ajouter les noms

```
> dimnames(summary.table) <-
list(n.fruit[,1], c("n","mean","SD"))
```

Tableaux

Plus complexes

- Importer une nouvelle base de données

```
> mydata<-read.csv("croissance.csv")
```

- Extraire l'information

```
> str(mydata)
> aggregate(mydata$gain,
list(mydata$supplement,mydata$diete), mean)
```

- Autre format de table

```
> mean.gain<-tapply(mydata$gain,
list(mydata$supplement,mydata$diete), mean)
```

Barplot

- Importer une nouvelle base de données

```
> mydata<-read.csv("fruits.csv")
```

- Extraire l'information
- Les données

```
> n.fruit <- tapply(mydata$Fruit,
list(Manger=mydata$Manger), length)
> mean.fruit <- tapply(mydata$Fruit,
list(Manger=mydata$Manger), mean)
> sd.fruit <- tapply(mydata$Fruit,
list(Manger=mydata$Manger), sd)
```

- Barplot `> barplot(mean.fruit)`
- Peut être mieux!

```
> help(barplot)
```

Barplot

- Ajout du nom des axes

```
> barplot(mean.fruit,
xlab="Traitement",
ylab="Production de fruit",
ylim=c(0,100))
```

- Ajout des barres d'erreur

```
> mids<-barplot(mean.fruit,
xlab="Traitement",
ylab="Production de fruit",
ylim=c(0,100))
> arrows(mids,mean.fruit+sd.fruit,
mids, mean.fruit - sd.fruit)
```

- Presque

```
> help(arrows)
```

Barplot

```
> mids<-barplot(mean.fruit,
xlab="Traitement",
ylab="Production de fruit",
ylim=c(0,100))
> arrows(mids,mean.fruit+sd.fruit,
mids, mean.fruit - sd.fruit,
angle=90, code=3)
> text(mids,5, paste("N = ", n.fruit))
```

Barplot

Plus complexe

```
> myData<-read.csv("croissance.csv")
> mean.gain<-tapply(myData$gain,
list(myData$supplement,myData$diete),
mean)
> sd.gain<-tapply(myData$gain,
list(myData$supplement,myData$diete),
sd)
> n.gain<-tapply(myData$gain,
list(myData$supplement,myData$diete),
length)
> barplot(mean.gain)
```

Presque

Barplot

Plus complexe

```
> mids<- barplot(mean.gain,beside=T,
xlab="Type nourriture",
ylab="Gain",
ylim=c(0,35),
col=grey(c(0,0.3,0.6,1)))
> arrows(mids, mean.gain+sd.gain,
mids, mean.gain-sd.gain,
angle=90, code=3, length=0.1)
> text(mids, 2, paste(n.gain), col=c("white",
rep("black",3)))
> legend("topright",
legend=rownames(mean.gain),
fill=grey(c(0,0.3,0.6,1)))
```

Simple graphique

```
> plot(mydata$Racine, mydata$Fruit)
> plot(mydata$Racine, mydata$Fruit,
   xlab="Largeur des racines",
   ylab="Production de fruit")
> plot(mydata$Racine, mydata$Fruit,
   xlab="Largeur des racines",
   ylab="Production de fruit",
   pch=21,bg="grey",cex=2.0)
```

Intact versus mangé

Simple graphique

```
> clr<-ifelse(mydata$Manger == "Mange",
"Green","Blue")
> plot(mydata$Racine, mydata$Fruit,
    xlab="Largeur des racines",
    ylab="Production de fruit",
    pch=21,bg=clr,cex=2.0)
> legend("topleft",
    legend=c("Mange", "Intact"), pch=21,
    pt.bg=c("Green", "Blue"), pt.cex=2.0)
```

Manipulations à deux variables

Graphiques

Modifications possibles sur un graphique.

- Titre du graphique

```
> plot(..., main="Titre")
```

- Nom des axes

```
> plot(..., xlab="nomX",ylab="nomY")
```

- Taille des points du graphique

```
> plot(..., cex=2.0)
```

- Taille du nom des axes

```
> plot(..., cex.lab=2.0)
```

- Taille des valeurs sur les axes

```
> plot(..., cex.axis=2.0)
```

- Limite des axes

```
> plot(..., xlim=c(0,100),ylim=c(0,2))
```

Manipulations à deux variables

Graphiques

Ajout de points sur un graphique.

- Un seul point

```
> points(x=50,y=0)
```

- Type de point

```
> points(...,pch=2)
```

- Couleur du point

```
> points(..., col=2)
```

- Taille du point

```
> points(..., cex=2.0)
```

- Plusieurs points

```
> points(x=c(1,2,3),y=c(1,2,3))
```

Manipulations à deux variables

Graphiques

Ajout de lignes sur un graphique.

- Une seule ligne entre (x1,y1) et (x2,y2)

```
> lines(x=c(x1,x2),y=c(y1,y2))
```

- Type de ligne

```
> lines(...,lty=2)
```

- Couleur de la ligne

```
> lines(..., col=2)
```

- Taille de la ligne

```
> lines(..., lwd=2.0)
```

- Plusieurs lignes reliées

```
> lines(x=c(x1,x2,x3),y=c(y1,y2,y3))
```

Sauver l'information

Figures

- Facile : cliquer file-> save as -> jpeg
- Plus efficace

```
> jpeg("fileName.jpg")
> plot(...)
> dev.off()
```

- Plus d'options avec la ligne de commande

```
> help(jpeg)
```

Sauver l'information

Figures

- Format
  - postscript
  - pdf
  - jpeg
  - png
  - bmp
  - tiff
- Options
  - taille (`width, height`)
  - compression (`quality`)
  - taille des points (`pointsize`)

Manipulations à deux variables

Filtrer les données

```
> x<-read.csv("hendryetAl.csv")
> x$Years
> x$Years>100
> x$Years==111
> x$Years!=124
> cond<-x$Years>100
> x$Years[cond]
> x$Haldanes[cond]
> x$Haldanes[x$Years>100]
```

Manipulations à deux variables

Liste des conditions

>	Plus grand
>=	Plus grand ou égal
<	Plus petit
<=	Plus petit ou égal
==	Égal
!=	Pas égal
&	Et
|	ou

Manipulations à deux variables

Filtrer les données

```
> cond<-x$Years>100 & x$Years<113
> cond<-x$Years<100 | x$Years>113
> cond<-x$Years>100 & x$Haldanes>=0
> x$Years[cond]
> x$Haldanes[cond]
```

Sauver l'information

Variables

```
> help(save)
> save(x,y,z,file="saveXYZ.RData")
```

Tout l'espace de travail

```
> save.image(file="workspace.RData")
```

Sauver l'information

Importer l'information

```
> load("saveXYZ.RData")
> load("workspace.RData")
```

Ce qui se trouve dans l'espace de travail

```
> ls()
```

Régression linéaire

- Haldane en fonction de la durée des generations?
- Syntaxe : `> x$Haldanes~ x$GLength`
  `> my.lm<-lm(x$Haldanes~ x$GLength)`
- Plus d'information:
  `> summary(lm(x$Haldanes~ x$GLength))`
  ou
  `> summary(my.lm)`

Régression linéaire

- Extraire l'information du modèle

  ```
  > attributes(my.lm)
  ```

- Résidus

  ```
  > my.lm$residuals
  ```

- Valeurs prédites

  ```
  > my.lm$coefficients
  ```

Régression linéaire

- Extraire plus l'information du modèle

```
> my.summ.lm<-summary(my.lm)
> attributes(my.summ.lm)
```

- R^2

```
> my.summ.lm$r.squared
```

- F-statistiques

```
> my.summ.lm$fstatistic
```

Régression linéaire

À quoi ça ressemble

```
> plot(x$Haldanes~ x$GLength)
```

Equivalent à

```
> plot(x$GLength,x$Haldanes)
```

Modèle linéaire

- Valeur absolue des Haldanes en fonction de la durée des générations?

```
> lm(x$HaldanesAbs~ x$GLength)
```

- Plus d'information :

```
> summary(lm(x$HaldanesAbs~ x$GLength))
```

Régression linéaire

À quoi ça ressemble

```
> plot(x$HaldanesAbs~ x$GLength)
```

- La courbe de régression sur le graphique
  ```
  > help.search("regression")
  ```
- Google : R regression line plot
- Cette fonction n'est pas dans le programme de base

Nouveau paquet

Installation

```
> install.packages()
```

- Trouver un fournisseur près (Canada ou USA)
- Sélectionner le paquet que vous voulez (car)

Nouveau paquet

Initialiser et utiliser le nouveau paquet

```
> library(car)
> help(reg.line)
> rg<-x$HaldanesAbs~ x$GLength
> plot(rg,xlab="Glength",ylab="Absolute
Haldanes")
> reg.line(lm(rg))
```

Faites une régression linéaire avec vos données

Régression linéaire

Votre tour

```
> lm(votreNom$a~ votreNom$b)
> summary(lm(votreNom$a~ votreNom$b))
```

Régression linéaire

Plus de variables

- Deux facteurs

```
> m1<- x$HaldanesAbs~ x$GLength+x$Years
```

- Interaction

```
> m2<- x$HaldanesAbs~ x$GLength:x$Years
```

- Deux facteurs + interaction :

```
> m3<- x$HaldanesAbs~ x$GLength*x$Years
```

équivalent à :

```
> m3<- x$HaldanesAbs~ x$GLength + x$Years +
x$GLength:x$Years
```

Faites une régression linéaire avec plus de facteurs

ANOVA

Initialisation

- Nouveau jeu de données `anova.txt` : regardez-le dans un éditeur
- Lire un tableau

```
> z<-read.table("anova.txt")
```

- Donner des noms aux colonnes

```
> names(z) <- c("response", "category",
"replicat", "coVar")
```

- Raccourci pour les variables

```
> attach(z)
> response
> detach(z)
> response
> attach(z)
```

ANOVA

Catégories

- Tout est numérique par défaut
- Définir les valeurs de catégorique comme “facteurs”

```
> category<-factor(category)
```

ANOVA

Première étape

- Écrire le modèle

```
> mod1<-response~ category
```

- Regarder le modèle

```
> boxplot(mod1)
```

- Régression linéaire du modèle

```
> mod1.lm<-lm(mod1)
```

- Visualiser le modèle

```
> plot(mod1.lm)
```

- Extraire l'information à propos du modèle

```
> summary(mod1.lm)
```

- Faire l'ANOVA

```
> anova(mod1.lm)
```

ANCOVA

Modèles

- Vérification de routine :

```
> is.factor(category)
```

- Visualiser les données:

```
> plot(response~
coVar,pch=as.numeric(category))
```

- Modèle complet :

```
> ResFull<-response~ category+ coVar +
category:coVar
```

- Même pente, différente ordonnées à l'origine:

```
> ResCD<-response~ category+ coVar
```

- Tou est en commun :

```
> ResE<-response~ coVar
```

ANCOVA

Modèles

- Régression linéaire

```
> ResFull.lm<-lm(ResFull)
> ResCD.lm<-lm(ResCD)
> ResE.lm<-lm(ResE)
```

- Visualiser les modèles

```
> plot(ResFull.lm)
> plot(ResCD.lm)
> plot(ResE.lm)
```

ANCOVA

Analyse

- Extraire l'information des modèles :

```
> summary(ResFull.lm)
> summary(ResCD.lm)
> summary(ResE.lm)
```

- ANCOVA

```
> anova(ResFull.lm, ResE.lm)
> anova(ResFull.lm, ResCD.lm, ResE.lm)
```

ANCOVA

Commande utile

- Vérifier l'égalité de la variance entre les groupes

```
> tapply(response, category, var,
na.rm=TRUE)
```

- Vérifier la normalité des groupes

```
> tapply(response, category, function(x)
shapiro.test(x))
```

Programmation simple

- Boucles :
  for
  while
- Conditions :
  if

Programmation simple

Boucles

```
> for (variable temporaire in vecteur){
>     opérations
> }
```

ou

```
> while (condition){
>     opération
>     changer un élément de la condition
> }
```

Programmation simple

Exemple de boucle

Fibonacci

```
> x<- array(1,dim=10)
> for (i in seq(3,length(x))) {
>    x[i]<-x[i-1]+x[i-2]
> }
```

ou

```
> x<- array(1,dim=10)
> i<-3
> while (i <= length(x)) {
>    x[i]<-x[i-1]+x[i-2]
>    i<-i+1
> }
```

Programmation simple

Conditions

```
> if (condition) {
>     opération 1
> } else {
>     opération 2
> }
```

Manipulations à deux variables

Liste des conditions

>	Plus grand
>=	Plus grand ou égal
<	Plus petit
<=	Plus petit ou égal
==	Égal
!=	Pas égal
&	Et
|	Ou

Programmation simple

Exemple de conditions

```
> x<-25
> if (x>0) {
>     x<-x+1
> } else {
>     x<-x-1
> }
```

Programmation simple

Premier programme : chaos

- Équation logistique : $x_{t+1} = r \times x_t(1 - x_t))$
- Exécuter la simulation pour 1000 générations
- Initialiser les conditions
- Essayer différents r entre 3 et 4
- Visualiser la série temporelle

Programmation simple

Solution : chaos

```
> x<-array(dim=1000)
> x[1]=0.5
> r<-3.7
> for (t in 2:length(x)){
>    x[t]<-r*x[t-1]*(1-x[t-1])
> }
```

Fonctions

Votre première fonction

Si vous voulez changer de paramètre facilement : écrivez une fonction

Fonctions

Avant

```
> x<-array(dim=1000)
> x[1]=0.5
> r<-3.7
> for (t in 2:length(x)){
>    x[t]<-r*x[t-1]*(1-x[t-1])
> }
```

Fonctions

Après

```
> chaos <- function(time,r,popS){
>     x<-array(dim=time)
>     x[1]=popS
>     for (t in 2:length(x)){
>         x[t]<-r*x[t-1]*(1-x[t-1])
>     }
>     return(x)
> }
```

Fonctions

Appeler votre fonction

```
> chaos(1000,3.5,0.5)
> c<-chaos(1000,3.5,0.5)
> plot(c)
> plot(chaos(1000,3.5,0.5))
```

Fonctions

Général

Tout ce que vous devez répéter, faites une fonction

Fonctions

Exercice

Faites une fonction qui retourne

$$\sum_{i=1}^{N} i$$

Fonctions

Solution

```
mySum <- function(N) {
    return(sum(seq(1,N)))
}
```

ou

```
mySum <- function(N) {
    return((N*(N+1))/2)
}
```

Fonctions

Avancées

- Pour utiliser une fonction d'un paquet:
  `require(nomPaquet)`
- Pour mettre des valeurs par défaut:
  `if(missing(nomVariable)) nomVariable <- valeurDefaut`

Calcul matriciel

- Création de vecteur

```
> V1 <- as.vector(seq(1,10))
> V2 <- as.vector(rnorm(10))
```

- Produit scalaire

```
> V3<- V1 * V2
```

- Produit vectoriel ((1 x n) * (n x 1))

```
> s1<- t(V1) %*% V2
```

- Produit vectoriel externe ((n x 1) * (1 x n))

```
> M1<- V1 %*% t(V2)
```

Calcul matriciel

- Création de matrice en assemblant des vecteurs
  Ligne par ligne `M1<-rbind(V1,V2)`
  Colonne par colonne `M2<-cbind(V1,V2)`
- Création de matrice élément par élément (colonne en premier) >
  `M3 <- matrix(c(1,2,3,4,5,6),ncol=3,nrow=2)`

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

```
> M4 <-
matrix(c(1,2,3,4,5,6),ncol=2,nrow=3)
```

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

Calcul matriciel

- Multiplication matrice par vecteur

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}$$

- `> M1 <- matrix(c(1,2,3,4,5,6), ncol=3, nrow=2)`
  `> V1 <- as.vector(c(7,8,9))`
  `> M1 %*% V1`
  NON : `> V1 %*% M1`

Calcul matriciel

- Multiplication matrice par matrice

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} \begin{pmatrix} 7 & 10 \\ 8 & 11 \\ 9 & 12 \end{pmatrix}$$

- `> M1 <- matrix(c(1,2,3,4,5,6), ncol=3, nrow=2)`
  `> M2 <- matrix(c(7,8,9,10,11,12), ncol=2, nrow=3)`
  `> M1 %*% M2`
  et non l'inverse : `> M2 %*% M1`

Calcul matriciel

Exercice

- Proportion de la population à l'équilibre

$$T = \begin{pmatrix} 0 & 1.29 & 0.702 \\ 0.7 & 0 & 0 \\ 0 & 0.2 & 0 \end{pmatrix} P = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

-

$$P_1 = TP$$

$$P_2 = TP_1$$

$$P_3 = TP_2$$

Calcul matriciel

- Valeur propre et vecteur propre

$$\begin{pmatrix} 0 & 1.29 & 0.702 \\ 0.7 & 0 & 0 \\ 0 & 0.2 & 0 \end{pmatrix}$$

- ```
  > M1 <-
  matrix(c(0,0.7,0,1.29,0,0.2,0.702,0,0),
  ncol=3, nrow=3)
  > eigen(M1)
  ```
  ```

Calcul matriciel

- Inverse, déterminant et diagonale

$$\begin{pmatrix} 0 & 1.29 & 0.702 \\ 0.7 & 0 & 0 \\ 0 & 0.2 & 0 \end{pmatrix}$$

- `> M1 <-`
  `matrix(c(0,0.7,0,1.29,0,0.2,0.702,0,0),`
  `ncol=3, nrow=3)`
  `> inv(M1)` (fait partie du paquet *car*)
  `> M1 %*% inv(M1)`
  `> det(M1)`
  `> diag(M1)`

Équation différentielles

- Vous devez avoir *deSolve*

```
> install.packages("deSolve")
> library(deSolve)
```

- Solutions numériques et non analytiques
- Pour des solutions analytiques, regardez :
  - Mathematica
  - Maple
  - Sage (gratuit)

Équation différentielles

Équation logistique

$$\frac{dX}{dt} = rX\frac{K-X}{K}$$

```
> library(deSolve)
> parameters<-c(r=1.5,K=100)
> state<-c(X=80)
> logistic <- function(t, state, parameters){
>    with(as.list(c(state, parameters)),
>    {
>       dX=r*X*(K-X)/K
>       return(list(dX))
>    })
> }
> times <- seq(1,100,by=0.1)
> out <- as.data.frame(ode(y=state,
times=times, func=logistic, parms=parameters))
> plot(out$X)
```

Équation différentielles

Modèle SIR

$$\frac{dX}{dt} = b(d'X + DY + d'Z) - \beta XY - d'X$$

$$\frac{dY}{dt} = \beta XY - (D + \gamma)Y$$

$$\frac{dZ}{dt} = \gamma Y - d'Z$$

b : naissance

d' et D : mortalité de base et de maladie

β : taux de transmission

γ : taux de guérison

Équation différentielles

Modèle SIR

```
> parameters<-c(b=1.0, d=0.2, beta=0.1,
gamma=0.7, D=0.3)
> state<-c(X=15, Y=4, Z=0)
> SIR<- function(t,state,parameters){
>    with(as.list(c(state,parameters)),{
>    dX = b*(d*X+D*Y+d*Z) - beta*X*Y - d*X
>    dY = beta*X*Y - (D+gamma)*Y
>    dZ = gamma*Y -d*Z
>    return(list(c(dX,dY,dZ)))
> })
> }
> times<- seq(1,100,by=1)
> out<-as.data.frame(ode(y=state,times=times,
>    func=SIR,parms=parameters))
```

Équation différentielles

Visualiser le modèle SIR

```
> par(mfrow=c(2,2), oma=c(0,0,3,0))
> plot(times,out$X, type="l",
   main="Susceptible", xlab="time",
   ylab="-")
> plot(times,out$Y, type="l",
   main="Infecté", xlab="time", ylab="-")
> plot(times,out$Z, type="l",
   main="Guéri", xlab="time", ylab="-")
```

Équation différentielles

Exercice : Lotka-Volterra

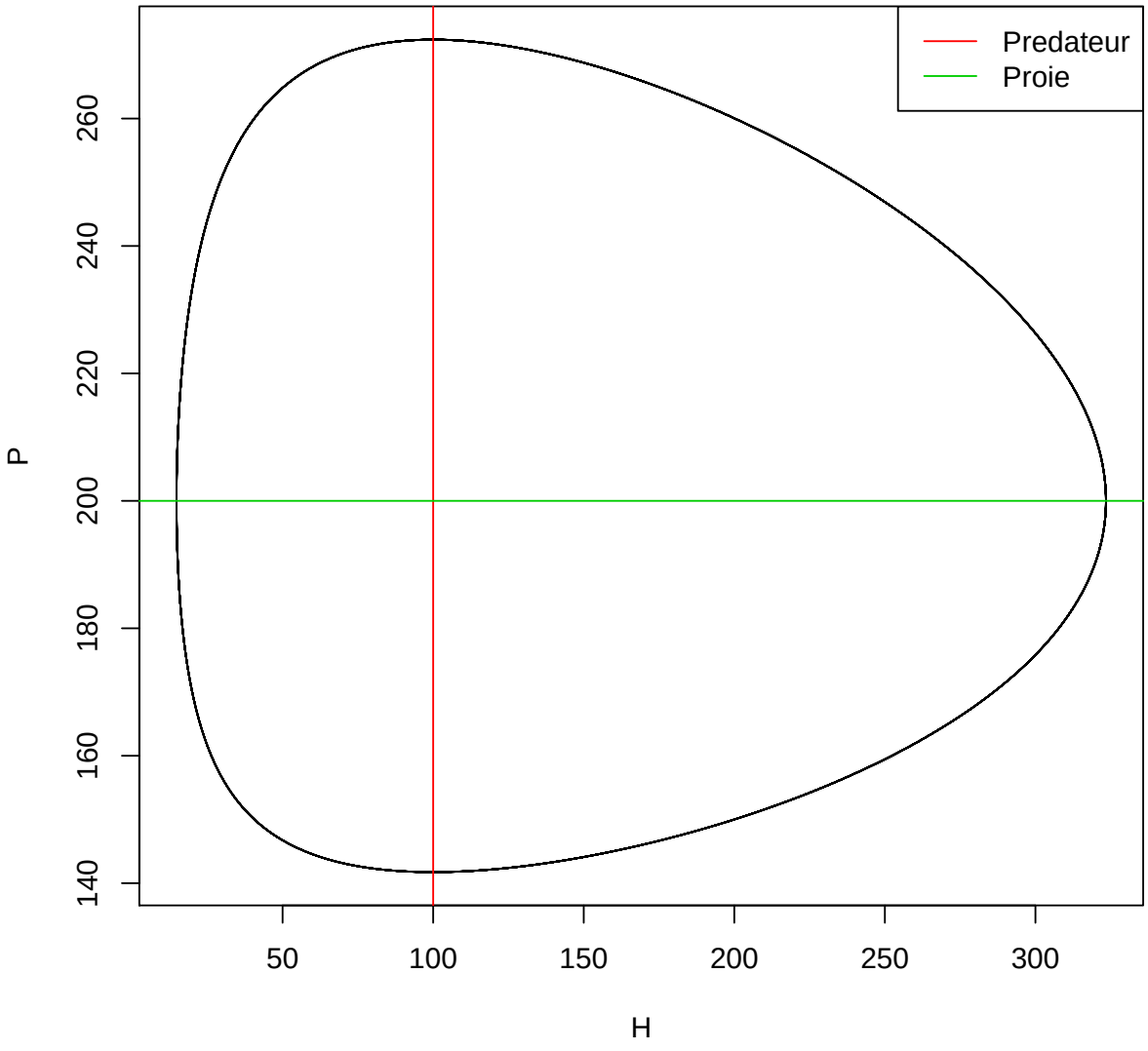
$$\frac{dH}{dt} = rH - pHP$$

$$\frac{dP}{dt} = apHP - mP$$

H : proie
P : prédateur
r : croissance des proies
p : efficacité des prédateurs
a : taux de transfert
m : mortalité des prédateurs
Tracez le plan de phase et les courbes de niveaux

Équation différentielles

Exercice : Lotka-Volterra



Conclusion

- RTFM
  `help(nomFonction)` et
  `help.search("ce que je cherche")`
- Un éditeur de texte est votre meilleur ami
  - Vérifier le format des données d'entrée CSV
  - Écrire dans un éditeur de texte AVANT de le mettre dans R
- Vous n'allez jamais endommager vos données en les utilisant dans R.

Remerciements

- Merci!