Introduction au logiciel R et applications

Marc J. Mazerolle
Centre d'étude de la forêt-UQAT

cef
Centre d'étude de la forêt

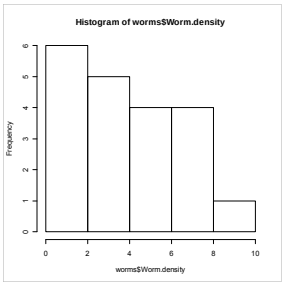
Université du Québec
en Abitibi-Témiscamingue

Graphiques

Graphiques dans R

histogramme

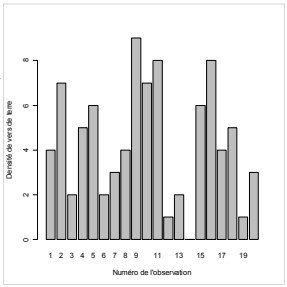
```
> hist(worms$Worm.density)
```



Graphiques dans R

diagramme en bâtons

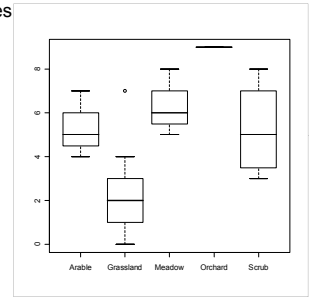
```
> barplot(worms$Worm.density,
xlab="Numéro de l'observation",
ylab="Densité de vers de terre",
names.arg=c(1:20))
```



Graphiques dans R

diagramme boîte et moustaches

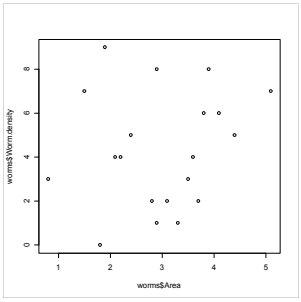
```
> boxplot(worms$Worm.density~
worms$Vegetation)
```



Graphiques dans R

graphique de dispersion

```
> plot(worms$Worm.density~
worms$Area)
```



Attention:

Certaines fonctions (graphiques et autres) ont plusieurs arguments.

`plot(x, y, ...)`

Si on nomme explicitement chaque argument, on peut les placer dans n'importe quel ordre.

`plot(x=2, y=10)` ou `plot(y=10, x=2)`

Si on donne seulement les valeurs, il faut qu'elles soient entrées dans le bon ordre.

`plot(2, 10)` ≠ `plot(10, 2)`

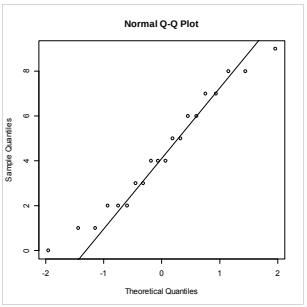
Graphiques dans R

graphique quantile-quantile

```
> qqnorm(worms$Worm.density)
```

pour vérifier la normalité des résidus

```
> qqline(worms$Worm.density)
```



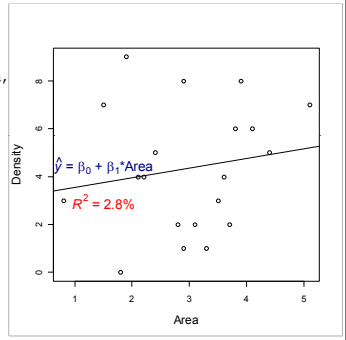
Graphiques dans R

Faire rouler une régression, puis ajouter la droite sur un graphique avec l'équation prédictive

```
> mod1<-lm(Worm.density~Area,
data=worms)

> abline(reg=mod1)

> text(x= , y= , labels= ,
col= , family= , cex=, ...)
```



Graphiques dans R

Autres fonction graphiques pratiques

`lines( )` pour faire des lignes

`pie( )` pour faire des tartes

Pour plus de possibilités, consulter le site R graph gallery:

http://addictedtor.free.fr/graphiques/

134 différents types

Graphiques dans R

Grande flexibilité dans la construction de graphiques:
on peut modifier tout, tout, tout

```
> ?par
```

`par` présente la majorité des arguments possibles à ajouter dans `plot( )` et plusieurs autre fonctions graphiques

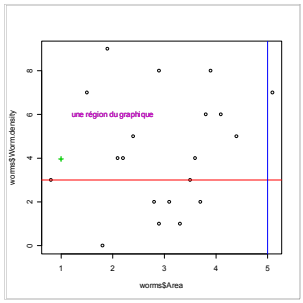
ex.: axes, étiquettes, échelle, police, taille (caractères, étiquettes), italiques, gras, couleur, lettres grecques, exposants, indices, style de d'axe, symboles, marges, ajout de légendes, texte, etc...

Graphiques dans R

un des gros avantages, c'est que l'on peut facilement ajouter des éléments sur le graphique après qu'il soit dessiné

```
> abline(h=3, v=5, col=c("red",
 "blue"))
> points(x=1, y=4, pch="+",
col=3, cex=1.5)

> text(x=2, y=6, labels=
"une région du graphique",
col="magenta")
```



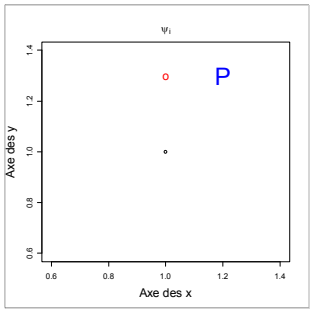
Graphiques dans R

```
> plot(x=1,y=1,
ylab="Axe des y",
xlab="Axe des x",
cex.lab=1.5)

> mtext(side=3,
text=expression(psi[i]),
line=1, cex=1.3)

> points(x=1, y=1.3, pch="o",
col=2, cex=1.5)

> points(x=1.2,y=1.3, pch="P",
col="blue", cex=3)
```



Graphiques dans R

Mettre plusieurs éléments avec barres d'erreurs et légende

Gain de masse moyen chez des animaux selon le type de diète et supplément

un groupe avec supplément agri

```
> agri
  Means   SD   Diet
1 26.35 1.84 barley
2 23.30 1.23   oats
3 19.64 1.42  wheat
```

un groupe avec contrôle

```
> cont
  Means   SD   Diet
1 23.30 1.41 barley
2 20.49 1.01   oats
3 17.41 0.92  wheat
```

Graphiques dans R

on crée le graphique vide sans axe x

```
plot( ... )
```

on ajoute l'axe x avec les étiquettes appropriées

```
axis( ... )
```

on ajoute les points pour les groupes agrimore et contrôle

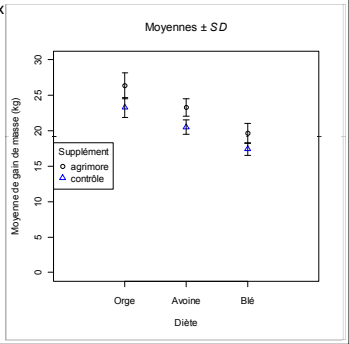
```
points( ... )
```

on ajoute les barres d'erreurs

```
arrows( ... )
```

on ajoute la légende

```
legend( ... )
```



Graphiques dans R

Ajouter un intervalle de confiance autour d'une droite de prédiction

on crée le graphique des valeurs brutes

```
plot(Worm.density ~ Area, ... )
```

on calcule les valeurs prédites à partir de l'équation de régression de Worm.density ~ Area et *SE*

```
predict( ..., se.fit = TRUE)
```

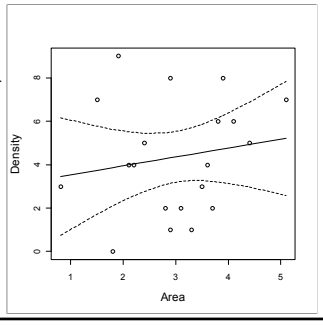
on ajoute la droite de régression

```
lines( ... )
```

on calcule les limites de l'*IC* à 95%

on ajoute les limites de l'*IC* à 95%

```
lines( ... )
```



Graphiques dans R

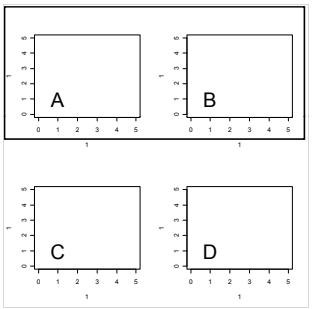
On peut facilement mettre plusieurs graphiques dans une seule fenêtre

`> par(mfrow=c(rangées,colonnes))` remplit une rangée à la fois

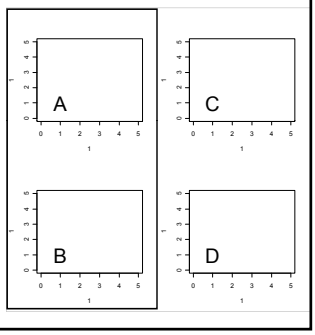
`> par(mfcol=c(rangées,colonnes))` remplit une colonne à la fois

Graphiques dans R

`> par(mfrow=c(2,2))`



`> par(mfcol=c(2,2))`



Graphiques dans R

Dans Windows, on peut copier-coller directement de R dans traitement de texte (format vectoriel)

On peut aussi sauvegarder en différents formats tels que `pdf`, `eps`, `jpeg`, `bitmap` (voir `?Devices`)

pour créer fichier `pdf`:

```
> pdf(file="C:\\Fig1.pdf")
> plot(x=c(1,2), y=c(6,5))
> dev.off( )
```

`dev.off( )` indique à R que la figure est complète et qu'on peut passer à l'appareil graphique précédent

pour créer fichier `postcript`:

```
> postscript(file="C:\\Fig1.ps")
> plot(x=c(1,2), y=c(6,5))
> dev.off( )
```

Graphiques dans R

pour créer fichier eps:

1) on affiche le graphique dans la fenêtre habituelle (à l'écran):

```
> windows( )
> plot(x=c(1,2), y=c(6,5))
```

2) on convertit le graphique en eps:

```
> dev.copy2eps(file="C:\\Fig1.eps")
```

fortement recommandé d'utiliser R pour faire vos graphiques

...il suffit d'oser faire le saut

Statistiques de base

Distributions statistiques

Fonctions associées aux distributions statistiques

`dnorm( )` fonction de densité de probabilité

```
> dnorm(x=5,mean=2,sd=3)
[1] 0.08065691
```

densité de la fonction lorsque x = 5 pour distribution normale avec $\mu = 2$ et $\sigma = 3$.

`pnorm( )` fonction de distribution (probabilité cumulative)

```
> 1-pnorm(q=1.96, mean=0, sd=1)
[1] 0.02499790
```

probabilité d'observer x > 1.96 dans une distribution normale centrée-réduite.

$1 - P(X \leq 1.96) = p$

`qnorm( )` fonction de quantile

```
> qnorm(p=0.975, mean=0, sd=1)
[1] 1.959964
```

quantile associé à une probabilité de 0.025 pour une distribution normale centrée-réduite (inverse de `pnorm( )`).

$1 - P(X \leq x) = 0.025$

Distributions statistiques

Fonctions associées aux distributions statistiques

`rnorm( )` générer observations aléatoires provenant d'une distribution normale

```
> var1<-rnorm(n=30, mean=12, sd=3)
```
génère 30 observations provenant d'une distribution normale avec $\mu = 12$ et $\sigma = 3$.

```
> var1
[1] 11.637296  8.818745  8.548053 10.528768 11.916288 11.852185 11.897473
12.513795 13.869845  8.005414 12.540423 10.237092 10.547166  5.900474 10.276910
17.227563 10.774479  7.130979  9.368063 15.500556 18.154208 12.687525 10.602007
11.039222  9.475898 13.373740 12.765565  9.316305 14.211309 10.649276
```

Plusieurs fonctions pour d'autres distributions ont la même structure

Ex.

X^2	*t* de Student	*F*	Poisson	binomiale	uniforme
`dchisq( )`	`dt( )`	`df( )`	`dpois( )`	`dbinom( )`	`dunif( )`
`pchisq( )`	`pt( )`	`pf( )`	`ppois( )`	`pbinom( )`	`punif( )`
`qchisq( )`	`qt( )`	`qf( )`	`qpois( )`	`qbinom( )`	`qunif( )`
`rchisq( )`	`rt( )`	`rf( )`	`rpois( )`	`rbinom( )`	`runif( )`

Statistiques descriptives

`mean( )` calcule la moyenne

```
> moy_var1<-mean(var1)
> moy_var1
[1] 11.37889
```

`sd( )` calcule l'écart-type

```
> sd_var1<-sd(var1)
> sd_var1
[1] 2.715868
```

} pas surprenant, car on a généré des données provenant d'une distribution normale avec $\mu = 12$ et $\sigma = 3$.

`median( )` calcule la médiane

```
> median(var1)
[1] 10.90685
```

`quantile( )` calcule les quantiles (quartiles par défaut)

```
> quantile(var1)
      0%       25%       50%       75%      100%
 5.900474  9.666196 10.906851 12.650750 18.154208
```

Statistiques descriptives

`rank( )` calcule les rangs

```
> rank(var1)
[1] 17  5  4 11 20 18 19 21 26  3 22  9 12  1 10 29 15  2  7 28 30
23 13 16  8 25 24  6 27 14
```

Pas de fonction pour *SE*

```
> SE_var1<-sd(var1)/sqrt(length(var1))
> SE_var1
[1] 0.4958474
```

on peut créer facilement des fonctions dans R

Créer ses propres fonctions dans R

R comme langage de programmation

R contient plusieurs fonctions déja programmées

Il est très facile de créer ses propres fonctions avec `function( )`

La formule pour calculer la *SE*:
`sd(var1)/sqrt(length(var1))`

Le plus simple, c'est d'identifier les éléments dont on aura besoin.

Pour calculer la *SE*, on a donc besoin:

n (nombre d'observations)
l'écart-type

Créer une fonction dans R

Une fonction est un autre type d'objet et on lui assigne un nom.

```
> SE_fonction<-function(x) {
```

`function( )` indique qu'on crée une fonction, et on mets les arguments de la fonction entre `( )`

ici, l'argument indique qu'on utilise une variable *x*

le corps de la fonction (i.e., les calculs ou manipulations à effectuer) se retrouvent entre `{ }`

Créer une fonction dans R

Une fonction est un autre type d'objet et on lui assigne un nom.

```
> SE_fonction<-function(x) {
```

`n_x<-length(x)`	on calcule le *n* de la variable *x*
`sd_x<-sd(x)`	on calcule la *SD* de la variable *x*
`sd_x/sqrt(n_x)`	on calcule la *SE* avec les éléments requis
`}`	on ajoute } pour indiquer que la fonction est terminée

```
> SE_fonction(x=var1)
[1] 0.4958474
```

on utilise la fonction nouvellement crée

Toutes les fonctions déja incluses dans R ont cette structure.

Créer une fonction dans R

On peut créer toutes sortes de fonction pour effectuer des tâches répétitives.

Un autre exemple de fonction simple: la racine carrée

```
> racine_car<-function(valeur) {
valeur^(1/2)
}
```

l'argument est valeur

on calcule la racine carrée

On fait rouler la fonction

```
> racine_car(16)
[1] 4
```

```
> racine_car(329)
[1] 18.13836
```

On vérifie si la fonction donne le même résultat que `sqrt( )`

```
> sqrt(16)
[1] 4
```

```
> sqrt(329)
[1] 18.13836
```

Créer une fonction dans R

Une fonction peut contenir plusieurs arguments

la racine $n^{ième}$

```
> racine_x<-function(valeur,niveau) {
valeur^(1/niveau)
}
> racine_x(8,3)
[1] 2
```

on a un argument pour la valeur, et un autre pour le niveau requis

```
> racine_x<-function(valeur,niveau=2) {
valeur^(1/niveau)
}
> racine_x(16,2)
[1] 4
> racine_x(16)
[1] 4
```

on peut assigner des valeurs par défaut (racine carrée)

Créer une fonction dans R

On peut créer une fonction pendant la session.

Mieux encore, on peut sauvegarder notre fonction dans un fichier séparé et y accéder lorsqu'on en a besoin avec la commande `source( )`.

```
> source(file="C:/chemin_sur_ordi/ma_fonction.txt")
```

Créer une fonction dans R

Une fonction peut retourner plusieurs éléments à la fois.
Extraire la somme, min et max d'une variable

```
> sum_max_min<-function(variable){
the_sum<-sum(variable)
the_min<-min(variable)
the_max<-max(variable)
list("Sum_of_observations" = the_sum, "Minimum_value" = the_min,
"Maximum_value" = the_max)
}
```

`list( )` permet de dresser une liste des résultats

```
> x<-rnorm(10)
> sum_max_min(x)
$Sum_of_observations
[1] -0.7677199

$Minimum_value
[1] -2.700584

$Maximum_value
[1] 1.031162
```

Affiche directement les valeurs à l'écran

Créer une fonction dans R

Plus pratique de créer un objet qui contient l'output de la fonction pour pouvoir extraire éléments par la suite (faire des graphiques, d'autres calculs, etc...)

```
> result<-sum_max_min(x)
```

```
> class(result)
[1] "list"
```

une liste est un autre type d'objet

On peut vérifier le contenu de la liste avec `names( )`

```
> names(result)
[1] "Sum_of_observations" "Minimum_value"       "Maximum_value"
```

Créer une fonction dans R

Pour visualiser tous les résultats

```
> result
$Sum_of_observations
[1] -0.7677199

$Minimum_value
[1] -2.700584

$Maximum_value
[1] 1.031162
```

Pour extraire les éléments, on utilise le nom de l'objet et l'opérateur $

```
> result$Sum_of_observations
[1] -0.7677199
```

```
> result$Minimum_value
[1] -2.700584
```

```
> result$Maximum_value
[1] 1.031162
```

Créer une fonction dans R

De façon générale, on utilise la même approche pour accéder à l'output d'une analyse dans R.

Plusieurs fonctions préprogrammées stockent l'output dans un objet et on peut extraire les éléments d'intérêt par la suite (pas besoin de rouler à nouveau l'analyse)

Les Boucles

Faire des boucles

Permet d'éviter de faire des tâches répétitives et ennuyeuses

Ex. simple:

générer 100 échantillons de 30 observations suivant une distribution normale $N(\mu = 10, \sigma = 1)$, et calculer la moyenne de chaque échantillon

en premier, on crée un objet pour stocker les moyennes calculées (output)

```
> output<-rep(x=NA, times=100)
```

on crée la boucle

```
> for (i in 1:100) {
simdata<-rnorm(n=30, mean=10, sd=1)
output[i]<-mean(simdata)
}
```

le calcul ou la manipulation à effectuer pour chaque itération est entre { }

on génère 1 échantillon de 30 obs, calcule la moyenne, et stocke la valeur (répété 100 fois)

Faire des boucles

```
> output
10.187493 10.057872 10.400478 10.150307  9.948904  9.713075 10.050015
 9.799055  9.928758 10.038981  9.775561 10.241103  9.966706 10.058531
10.099403  9.814790  9.884374  9.840617 10.112026  9.886627 10.165894
10.076277  9.505789  9.976422  9.977487  9.741762  9.859750  9.689165
10.421739 10.101839  9.932801  9.985449 10.054757 10.102494 10.121907
10.093705  9.793565 10.159726  9.902479  9.986425 10.311193  9.801151
10.102233  9.943137  9.999567 10.246178  9.798786 10.160431  9.928409
10.119910  9.896694 10.186127  9.690116 10.020563  9.829083  9.890244
 9.996526 10.089126 10.014033  9.466073 10.234593 10.022860  9.917595
10.120968  9.827922  9.961950  9.961035  9.874266 10.256765 10.042088
 9.957012 10.080500 10.117559 10.311178  9.997452  10.038956 10.128459
10.041864 10.044567  9.857874 10.207001 10.004664   9.804787  9.897710
10.256215  9.955179 10.073420  9.935395  9.673318 10.363609  10.016773
10.211726 10.145192 10.032976 10.085238 10.085375  9.843844 10.056572
10.159683  9.921390
> mean(output)
[1] 10.00619
```

Faire des boucles

Très pratique pour manipulation avant analyse

Application facile pour bootstrap, ou tests de randomisation

Possibilité de manipulations plus complexes:

Ex. importer fichier de texte provenant de 1000 fichiers d'output d'un autre logiciel, trouver les lignes correspondant aux estimés, extraire ces valeurs, faire un tableau de résultats dans R et l'exporter en texte par la suite pour sauvegarder.

Peuvent être imbriquées l'une dans l'autre

Peuvent être incluses dans fonction

Pour faire du bootstrap

L'échantillon original:

```
dbh <- c(3, 22, 8, 50, 9, 22, 26, 46, 28, 16, 33, 45, 18, 34)
```

bootstrap (on veut trouver la *SE*)

```
nsims<-1000
```
on fixe le nombre d'itérations

```
boot_out<-rep(NA, nsims)
```
on crée un objet pour stocker l'output

```
for (i in 1:nsims) {
```
on débute la boucle (`i = 1`)

```
   bootsample<-sample(x=dbh, size=length(dbh), replace=TRUE)
```
on crée un échantillon de bootstrap à partir de l'échantillon original (`dbh`)

```
boot_out[i]<-mean(bootsample)
```
on calcule la moyenne pour l'échantillon de bootstrap et on stocke la valeur dans `boot_out[i]`

```
}
```
on recommence la boucle pour (`i = 2`)

La *SE* de la statistique est donné par `sd(boot_out)`

Faire des boucles

Certaines fonctions agissant comme des boucles:

`tapply( )` applique une fonction sur un élément selon les différents niveaux d'un facteur

Ex. calculer la moyenne de densité de vers de terre selon les différents types de végétation

```
> tapply(X=worms$Worm.density, INDEX=worms$Vegetation, FUN=mean)
   Arable Grassland    Meadow   Orchard     Scrub
 5.333333  2.444444  6.333333  9.000000  5.250000
```

Autres fonctions similaires pour différents types d'objets:

`apply( )`
`sapply( )`
`lapply( )`

Analyses statistiques

Le χ^2 pour tableaux de contingence

`chisq.test( )` permet de tester si les proportions sont les mêmes selon un ou plusieurs facteurs d'un tableau de contingence

La proportion d'individus entre trois classes d'âge est-elle la même?

```
> Freq<-c(124,176, 180)
> chisq.test(Freq, p=c(1/3, 1/3, 1/3))
```

p = proportions conformes à H_0

```
  Chi-squared test for given probabilities

  data:  Freq
  X-squared = 12.2, df = 2, p-value = 0.002243
```

Le test-*t* pour deux groupes

```
> data(sleep)
> sleep[1:12,]
   extra group
1    0.7     1
2   -1.6     1
3   -0.2     1
4   -1.2     1
5   -0.1     1
6    3.4     1
7    3.7     1
8    0.8     1
9    0.0     1
10   2.0     1
11   1.9     2
12   0.8     2
```

On utilise le jeu de données sleep déja contenu dans R (`?sleep`)

Un de deux médicaments administré à 20 patients (10 de chaque groupe) et on mesure le nombre d'heures de sommeil de plus par rapport à un contrôle.

Le test-*t* pour deux groupes

`t.test( )` teste différence de moyennes entre deux groupes

Plusieurs arguments possibles:

`alternative = c("two.sided", "less", "greater")` test bilatéral ou unilatéral

`paired = FALSE` test pour groupes appariés ou indépendants

`mu = 0` H_0: $\mu_1 - \mu_2 = 0$ ($\mu_1 = \mu_2$)

`var.equal = FALSE` test avec variances égales ou hétérogènes

`conf.level = 0.95` seuil pour IC

Le test-*t* pour deux groupes

Faisons rouler la fonction avec les données

```
> t.test(extra~group, mu=0, data=sleep)
```

`data=sleep` indique que les données se trouvent dans l'objet `sleep`

Par défaut, suppose que les variances sont hétérogènes (si égales, on peut spécifier `var.equal = TRUE`)

```
 Welch Two Sample t-test

data:  extra by group
t = -1.8608, df = 17.776, p-value = 0.0794
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -3.3654832  0.2054832
sample estimates:
mean in group 1 mean in group 2
           0.75            2.33
```

Le test-*t* pour deux groupes

Plus pratique si on assigne l'output à un objet

```
> result<-t.test(extra~group, mu=0, data=sleep)
```

`names( )` pour connaître le contenu de l'output

```
> names(result)
[1] "statistic"   "parameter"   "p.value"     "conf.int"
"estimate"    "null.value"  "alternative" "method"      "data.name"
```

On peut extraire n'importe quel des éléments

```
> result$statistic
        t
-1.860813
> result$estimate
mean in group 1 mean in group 2
           0.75            2.33
```

La corrélation de Pearson

`cor( )` calcule corrélation entre variables

on peut générer des données

```
> x1<-rnorm(n=30, mean=2, sd=1)
> x2<-rpois(n=30, lambda=5)
> cor(x1,x2)
[1] 0.2605317
```

`cor.test( )` calcule corrélation entre variables et teste H_0: $\rho = 0$.

```
> cor.test(x1,x2)
        Pearson's product-moment correlation
data:  x1 and x2
t = 1.4279, df = 28, p-value = 0.1644
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.1100687  0.5675316
sample estimates:
      cor
0.2605317
```

La régression linéaire

```
> chocs<-read.table("C:\\chocs_red.dat", header=TRUE)
> chocs
                Size Price
Dark.Bounty     50.0  0.88
Bounty          50.0  0.88
Milo.Bar        40.0  1.15
Viking          80.0  1.54
KitKat.White    45.0  1.15
KitKat.Chunky   78.0  1.40
Cherry.Ripe     55.0  1.28
Snickers        60.0  0.97
Mars            60.0  0.97
Crunchie        50.0  1.28
Tim.Tam         40.0  1.10
Turkish.Delight 55.0  1.28
Mars.Lite       44.5  0.97
Dairy.Milk.King 75.0  1.58
Maltesers       60.0  1.55
MandMs          42.5  1.18
```

Prix de barres de chocolat australiennes en fonction de masse.

La régression linéaire

`lm( )` ajuste des modèles linéaires supposant une distribution normale des erreurs (ANOVA, régression multiple, ANCOVA)

Plusieurs arguments possibles, voir `?lm`

On fait tourner la régression linéaire

```
> mod_reglin<-lm(Price~Size, data=chocs)
```

si nécessaire, on peut ajouter des variables explicatives séparées par `+` dans la formule du modèle: `Price ~ Size + Fat + Size:Fat`

`:` indique interaction entre deux variables

La régression linéaire

`summary( )` extrait un résumé de l'output

```
> summary(mod_reglin)
Call:
lm(formula = Price ~ Size, data = chocs)
Residuals:
     Min       1Q   Median       3Q      Max
-0.28031 -0.14368  0.07184  0.12546  0.29969
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.574368   0.213738   2.687  0.01769 *
Size        0.011266   0.003768   2.989  0.00975 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1891 on 14 degrees of freedom
Multiple R-Squared: 0.3896,     Adjusted R-squared: 0.346
F-statistic: 8.937 on 1 and 14 DF,  p-value: 0.009753
```

La régression linéaire

Éléments contenus dans l'output

```
> names(mod_reglin)
 [1] "coefficients"  "residuals"     "effects"       "rank"
"fitted.values" "assign"        "qr"            "df.residual"   "xlevels"
[10] "call"          "terms"         "model"

 > mod_reglin$coefficients
 (Intercept)        Size
  0.57436804  0.01126566

> mod_reglin$residuals
    Dark.Bounty          Bounty        Milo.Bar          Viking    KitKat.White
    -0.25765117     -0.25765117      0.12500546      0.06437895      0.06867715
  KitKat.Chunky     Cherry.Ripe        Snickers            Mars        Crunchie
    -0.05308972      0.08602052     -0.28030779     -0.28030779      0.14234883
       Tim.Tam Turkish.Delight       Mars.Lite Dairy.Milk.King       Maltesers
     0.07500546      0.08602052     -0.10569002      0.16070727      0.29969221
         MandMs
     0.12684130
```

La régression linéaire

Il existe plusieurs fonctions génériques permettant d'extraire des éléments additionnels d'objet contenant de l'output

`logLik( )` le log-likelihood du modèle

```
> logLik(mod_reglin)
'log Lik.' 5.009943 (df=3)
```

`influence.measures( )` effet de levier, distances de Cook, dfbeta

```
> influence.measures(mod_reglin)
Influence measures of
         lm(formula = Price ~ Size, data = chocs) :
                dfb.1_ dfb.Size  dffit cov.r  cook.d    hat inf
Dark.Bounty    -0.2427  0.16205 -0.416 0.919 0.07970 0.0737
Bounty         -0.2427  0.16205 -0.416 0.919 0.07970 0.0737
Milo.Bar        0.2712 -0.23449  0.303 1.274 0.04766 0.1556
Viking         -0.2013  0.23336  0.262 1.628 0.03646 0.3045   *
...
```

La régression linéaire

`rstudent( )` les résidus de Student (bonne mesure de valeurs extrêmes)

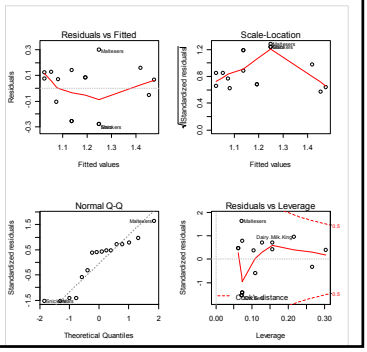
`anova( )` retourne un tableau d'ANOVA pour le modèle

```
> anova(mod_reglin)
Analysis of Variance Table

Response: Price
          Df  Sum Sq Mean Sq F value   Pr(>F)
Size       1 0.31969 0.31969  8.9369 0.009753 **
Residuals 14 0.50081 0.03577
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

La régression linéaire

`plot( )`
génère 4 graphiques diagnostiques



La régression linéaire

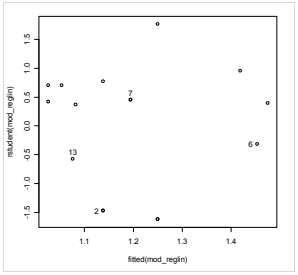
Graphique des résidus de Student en fonction de valeurs prédites

```
> plot(rstudent(mod_reglin)~fitted(mod_reglin))
```

On peut identifier des points sur un graphique avec `identify( )`

```
> identify(rstudent(mod_reglin)
~fitted(mod_reglin))
```

cliquer sur un point du graphique, identifie le numéro de l'observation



ANOVA, ANCOVA, régression multiple

Dépendamment de la nature des variables explicatives, `lm( )` peut faire des ANOVA's, ANCOVA's ou régressions multiples

pour obtenir les tables d'ANOVA, utiliser `anova( )` ou `summary.aov( )`

on peut extraire les mêmes éléments de tous ces modèles

GLM's

`glm( )` ajuste des modèles linéaires généralisés pour lequel on spécifie une distribution (normale, Poisson, binomiale, gamma)

la syntaxe est très semblable à `lm( )` et les fonctions d'extraction fonctionnent de façon similaire

On peut faire une régression de Poisson à partir des variables du jeu de données `species`.

GLM's

On importe le jeu de données:

```
> species <- read.table(file = "/home/ mazerolm/species.txt",
                        header = TRUE)
> head(species)
    pH   Biomass Species
1 high 0.4692972      30
2 high 1.7308704      39
3 high 2.0897785      44
4 high 3.9257871      35
5 high 4.3667927      25
6 high 5.4819747      29
```

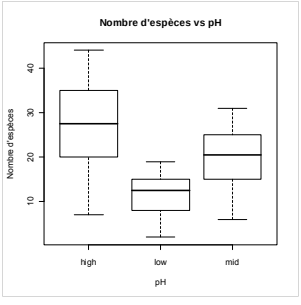
`Species`: nombre d'espèces observées dans le quadrat

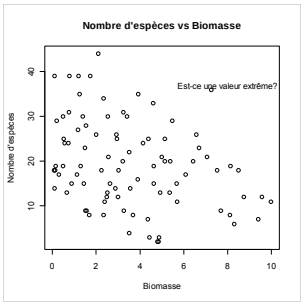
`pH`: pH du sol du quadrat

`Biomass`: biomasse dans le quadrat

On peut visualiser les relations entre `Species` et `pH` ou `Biomass`.

GLM's





GLM's

Ajustons une régression de Poisson:

```
> mod_GLM1 <- glm(Species ~ Biomass + pH, family = poisson,
                  data = species)
> summary(mod_GLM1)
...
Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.59586  -0.69887  -0.07373   0.66472   3.56040

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.84894    0.05281  72.885  < 2e-16 ***
Biomass     -0.12756    0.01014 -12.579  < 2e-16 ***
pHlow       -1.13639    0.06720 -16.910  < 2e-16 ***
pHmid       -0.44516    0.05486  -8.114 4.88e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)
```

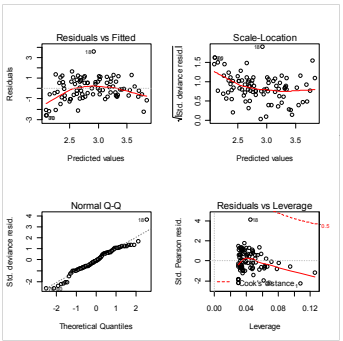
supposition à vérifier

GLM's

On peut utiliser certains graphiques pour diagnostiquer des problèmes

```
> layout(mat = matrix(1:4,
  nrow = 2, ncol = 2))
> plot(mod_GLM1)
```

Ajustement plus ou moins satisfaisant résidus vs valeurs prédites



GLM's

```
> mod_GLM2 <- glm(Species ~ Biomass + pH + Biomass:pH, family =
poisson, data = species)

> summary(mod_GLM2)
...

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   3.76812    0.06153  61.240  < 2e-16 ***
Biomass      -0.10713    0.01249  -8.577  < 2e-16 ***
pHlow        -0.81557    0.10284  -7.931 2.18e-15 ***
pHmid        -0.33146    0.09217  -3.596 0.000323 ***
Biomass:pHlow -0.15503    0.04003  -3.873 0.000108 ***
Biomass:pHmid -0.03189    0.02308  -1.382 0.166954
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)
```

GLM's

Modèle avec interaction `Biomass:pH` s'ajuste mieux aux données

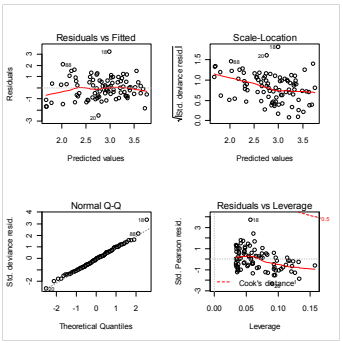
Vérification de supposition de dispersion = 1 (variance = moyenne)

```
> chisq <-
sum(residuals(mod_GLM2,
type = "pearson")^2)

> residual.df <-
➢mod_GLM2$df.residual

> chisq/residual.df
[1] 0.9970014
```

OK



Interprétation des résultats

À partir des estimés:

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   3.76812    0.06153  61.240  < 2e-16 ***
Biomass      -0.10713    0.01249  -8.577  < 2e-16 ***
pHlow        -0.81557    0.10284  -7.931 2.18e-15 ***
pHmid        -0.33146    0.09217  -3.596 0.000323 ***
Biomass:pHlow -0.15503    0.04003  -3.873 0.000108 ***
Biomass:pHmid -0.03189    0.02308  -1.382 0.166954
...
```

Conclusion:

l'effet de la biomasse est plus fort à pH low qu'à pH high
(effet de biomasse pour pH high = `-0.1071`)
(effet de biomasse pour pH low = `-0.1071 + -0.1550 = -0.2622`)

l'effet de la biomasse à pH mid ne diffère pas de l'effet à pH high
(effet de biomasse pour pH mid = `-0.1071 + -0.0319 = -0.1390`)

GLM's

On peut facilement faire une régression logistique avec `glm( )`

Jeu de donnée neural (vu précédemment).

On analyse la probabilité que des patients ressentent moins de douleur suite à un traitement, selon leur âge et sexe.

```
> mod1<-glm(Reduce~Age+Gender, family=binomial(link=logit), data=neural)
```

`family=binomial` indique que l'on veut un GLM binomial (régression logistique)

`link = logit` indique que l'on veut une fonction de lien logit

GLM's

```
> summary(mod1)
Call:
glm(formula = Reduce ~ Age + Gender, family = binomial(link = logit),
    data = neural)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.1760  -0.9603  -0.7633   1.0878   1.8536
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  7.06246    6.59271   1.071    0.284
Age         -0.09679    0.08605  -1.125    0.261
GenderM     -1.22652    1.37535  -0.892    0.373
(Dispersion parameter for binomial family taken to be 1)
    Null deviance: 24.057  on 17  degrees of freedom
Residual deviance: 22.558  on 15  degrees of freedom
AIC: 28.558

Number of Fisher Scoring iterations: 4
```

Autres exemples d'analyses

Modèles linéaires mixtes et mixtes généralisés (*GLMM*)

GLM's pour mesures répétées (*GEE*'s)

Ordinations (*PCA*, *CCA*, *RDA*) et autres analyses multivariées

Randomisations: bootstrap, Monte Carlo

Modèles non-linéaires

Optimisation: modèles « construits à la main » à partir des fonctions de vraisemblance

Les packages: applications spécifiques

Package = banque de fonctions

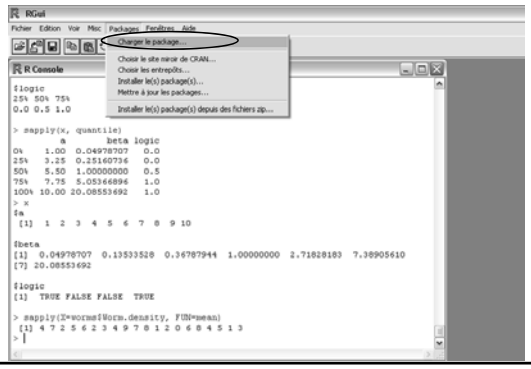
Les fonctions préprogrammées de R sont stockées dans des banques de fonctions (packages)

Lorsqu'on ouvre R, une série de banques de fonctions de base sont activées automatiquement (on peut en ajouter si voulu)

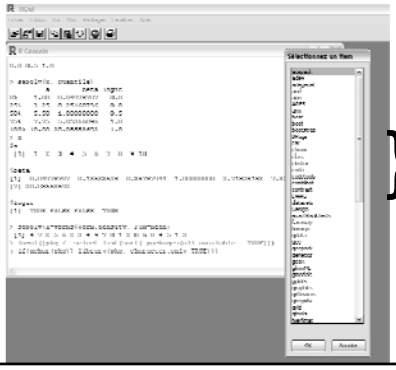
Pour « sortir des sentiers battus », il faut activer le package contenant la fonction d'intérêt (R en contient déja plusieurs avec l'installation de base)

`library( )`

Accéder aux fonctions



Accéder aux fonctions



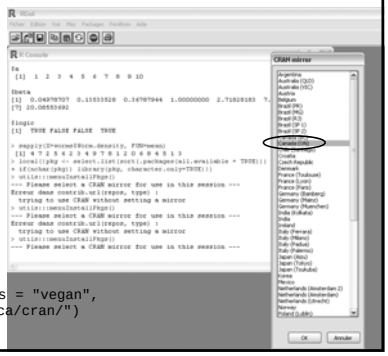
la liste des packages déja installés

Mieux encore, utiliser simplement la fonction `library( )` avec le nom du package entre ( )

e.g., `> library(MASS)`

Accéder aux fonctions

Si le package nécessaire n'est pas là, il faut le télécharger et l'installer à partir de CRAN.



Mieux encore, utiliser la fonction `install.packages( )`

e.g., `> install.packages(pkgs = "vegan", repos="http://probability.ca/cran/")`

Hors des sentiers battus

Plusieurs possibilités avec le logiciel de base

Plus de 2000 packages disponibles sur CRAN afin d'effectuer analyses moins conventionnelles, interagir avec autres logiciels, etc...

Pour faire des analyses plus pointues (*CCA*, réseaux de neurones, analyses Bayesiennes, modèles mixtes)

Hors des sentiers battus

Pour voir un bref résumé des packages, on peut aller sur CRAN

Quelques package d'intérêts

Bootstrap
`boot`, `bootstrap`

Modèles linéaires mixtes et généralisés
`MASS`, `nlme`, `lme4`

Analyses multivariées (*CCA*, *RDA*)
`vegan`, `ade4`, `cluster`

Interagir avec d'autres logiciels
`RMARK`, `R2WinBUGS`, `BRugs`, `GRASS`

Sélection de modèle et inférence multimodèles
`AICcmodavg`

Fonctions d'utilités générales
`car`, `Design`, `Hmisc`, `gregmisc`, `gmodels`, `foreign`

Quelques package d'intérêts

Applications plus « pointues »

analyses spatiales: `geoR`, `sp`, `spatial`

séries temporelles (ARIMA, etc...): `far`, `fARMA`

analyses bayesiennes: `arm`, `bayescount`, `bayesm`, `bayesmix`

génétique: `adegenet`, `ape`, `apTreeshape`, `seqinr`, `genetics`

dendrochronologie: `dplR`

Un petit conseil

Lors d'une session, n'activez seulement les packages dont vous avez besoin.

Attention aux conflits entre packages

Certains packages masquent les fonctions de base de R ce qui peut entraîner de mauvaises surprises.

Pour désactiver un package, utiliser la fonction `detach( )`

```
> detach(package:nlme)
```

D'autres conseils

De nouvelles versions des packages sont téléversés sur CRAN régulièrement

Une bonne idée de vérifier de temps à autres s'il existe de nouvelles versions des packages installés

soit à l'aide du menu du GUI sous Packages... mettre à jour les packages

mieux encore, utiliser la fonction `update.packages( )`

D'autres conseils

À la fin d'une session R, on a l'option de sauvegarder la session avec tous les objets en mémoire.

Si vous sauvegardez une session modifiée sur une longue période, assurez-vous que certains des objets n'aient pas été modifiés par mégarde (un jeu de données remplacé par un vecteur du même nom)

Vérifiez ce qui est en mémoire avec `ls( )`

```
> ls()
[1] "i"       "output"  "simdata" "worms"   "x"
```

Pour supprimer des objets utiliser `rm( )`

```
> rm(x)
> ls()
[1] "i"       "output"  "simdata" "worms"
```

D'autres conseils

À la fin d'une session R, on a l'option de sauvegarder la session avec tous les objets en mémoire.

Au lieu de sauvegarder la session R, préférable de sauvegarder tout le programme (script) associé à une analyse donnée dans un fichier `Tinn-R` (ou autre éditeur).

Utilisez les commentaires abondamment dans votre script, car facilite beaucoup la tâche si on veut reproduire l'analyse, la modifier ou l'appliquer à un autre jeu de données (on a pas à réinventer la roue à chaque fois).

Comparaisons vs d'autres logiciels populaires

Tableau comparatif

	R	SAS	SPSS
Coût			
Achat	0$	$$$	$$
Mise à jour	0$	$/année	$
Difficulté	++	++	+
Support web	+++	++	+
Graphiques	+++	++	++
Transparence	+++	++	+
Interface	-	+	+++
Économie de code	++	+	+++
Contributeurs	+++	+	+

Tableau comparatif (cont.)

	R	SAS	SPSS
Analyses			
GLM	+++	+++	+
GEE	+++	++	*
GAM	++	+	-
Modèles mixtes	+++	++	+
Multivariées			
CCA	++	*	-
RDA	++	+	-
Randomisations			
bootstrap	+++	++	*
simulations	+++	++	-
Optimisation	+++	+++	-
Flexibilité	+++	+++	-

*Macro

Ressources R sur le web

CRAN:
http://cran.r-project.org/

R tips:
http://pj.freefaculty.org/R/Rtips.html

R site search:
http://finzi.psych.upenn.edu/search.html

Quick-R:
http://www.statmethods.net/index.html

R Graph Gallery:
http://addictedtor.free.fr/graphiques/allgraph.php

Livres avec bonne intro à R

Dalgaard, P. 2002. Introductory Statistics with R. Springer, New York.

Crawley, M. J. 2005. Statistics: An introduction using R. John Wiley and Sons, Chichester, UK.

Livres plus avancés avec R

Venables, W. N., and B. D. Ripley. 2002. Modern Applied Statistics with S. Springer-Verlag, New York, USA.

Venables, W. N., and B. D. Ripley. 2002. S Programming. Springer, New York, USA.

Livres plus spécifiques

Régressions et *GLM*'s

Fox, J. 2002. An R and S-PLUS companion to applied regression. Sage Publications, Inc., London, UK.

Faraway, J. J. 2006. Extending the linear model with R: generalized linear, mixed effects and nonparametric regression models. Chapman and Hall, New York, USA.

Modèles mixtes

Pinheiro, J. C., and D. M. Bates. 2000. Mixed-effect models in S and S-PLUS. Springer Verlag, New York.

Gelman, A., and J. Hill. 2007. Data analysis using regression and multilevel/hierarchical models. Cambride University Press, New York, USA.

Zuur et al. 2009. Mixed effects models and extensions in ecology with R. Springer-Verlag, New York, USA.

Livres plus spécifiques

Bootstrap et randomisations

Efron, B., and R. J. Tibshirani. 1998. An introduction to the bootstrap. Chapman & Hall/CRC, New York, USA.

Good, P. I. 2005. Introduction to statistics through resampling methods and R/S-PLUS. John Wiley & Sons, Hoboken, NJ, USA.

Analyses multivariées

Everitt, B. 2005. An R and S-PLUS companion to multivariate analysis. Springer, London, UK.

Génétique

Paradis, E. 2006. Analysis of phylogenetics and evolution with R. Springer, New York, USA.

Analyses bayesiennes

Albert, J. 2007. Bayesian computation with R. Springer, New York.

Livres plus spécifiques

Modélisation/optimisation

Bolker, B. M. 2008. Ecological models and data in R. Princeton University Press, Princeton, New Jersey.

Pawitan, Y. 2001. In all likelihood: statistical modelling and inference using likelihood. Oxford University Press, Oxford.

Royle, J. A., and R. M. Dorazio. 2008. Hierarchical modeling and inference in ecology: the analysis of data from populations, metapopulations and communities. Academic Press, New York.

Analyses spatiales

Bivand, R. S., E. J. Pebesma, and V. Gómez-Rubio. 2008. Applied spatial data analysis with R. Springer, New York, NY, USA.

Analyses temporelles (time series)

Shumway, R. H., and D. S. Stoffer. 2006. Time series analysis and its applications with R examples, 2nd edition. Springer, New York, NY, USA.

Autres ressources

Mon cours de statistiques avancées BIO8093 de 2ième et 3ième cycle
cours de 3 crédits disponible via vidéoconférence
(session hiver offert à Québec, Montréal et Chicoutimi)

- bootstrap, tests de randomisation et simulations de Monte Carlo
- GLM's avancés (gamma, binomiale négative)
- régressions logistiques ordinales et multinomiales
- analyses pour mesures répétées (GEE's)
- modèles mixtes linéaires et généralisés (GLMM's)

Ateliers de formations spécifiques en statistiques

- modèles mixtes
- analyses d'occupation de site
- modèles de capture-marquage-recapture
- ou autres sur demande (marc.mazerolle@uqat.ca)

BONNE UTILISATION!!!