INTRODUCTION AU LOGICIEL R

4. Statistiques avancées. Boucles.

Anne Dubois, Julie Bertrand, Emmanuelle Comets

emmanuelle.comets@inserm.fr

INSERM UMR738

Remise en jambe (graphes)

Le jeu de données women disponible dans R donne la taille et le poids moyen pour des femmes américaines âgées de 30 à 39 ans (mesurées dans la deuxième moitié du vingtième siècle. Les mesures étaient réalisées habillées).

1. Affichez les données et regardez la page d'aide décrivant le jeu de données
2. Les unités sont données en inches pour la taille et en livres pour le poids. Transformez votre jeu de données pour retrouver des unités métriques (cm et kg).

Remise en jambe (tests)

Le jeu de données CO2 disponible dans R est tiré d'une expérience testant la résistance au froid d'une herbe. Deux représentants de cette espèce vivent au Canada et aux Etats-Unis, et on a mesuré pour 6 différents plants de chaque origine la croissance de la plante (en terme de quantité de CO_2 absorbé). La moitié des plants ont été réfrigérés ("chilled") avant l'expérience afin d'évaluer la résistance au froid.

1. Affichez les données et regardez la page d'aide décrivant le jeu de données
2. Tracer sur la même page les histogrammes des concentrations de CO_2 absorbé (2 figures)
3. Calculer la moyenne de CO_2 absorbée selon que la plante a été ou non réfrigérée.
4. Le traitement a-t-il un effet significatif sur la quantité de CO_2 absorbé ?

Plan

Régression linéaire (1)

On utilise la base cats de la library MASS

```
> library(MASS)
> pairs(cats)
```

On souhaite expliquer le poids du coeur (Hwt) par :

- le poids du corps (Bwt)
- le sexe (Sexe)

L'exploration univariée (test t et corrélation paramétrique) conserve les 2 variables explicatives mais met en évidence un lien entre Bwt et Sex. On construit donc :

```
> cats.lm <- lm(Hwt~Bwt*Sex,data=cats)
> summary(cats.lm)
```

Régression linéaire (2)

On obtient :

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.9813     1.8428   1.618 0.107960
Bwt           2.6364     0.7759   3.398 0.000885 ***
SexM         -4.1654     2.0618  -2.020 0.045258 *
Bwt:SexM      1.6763     0.8373   2.002 0.047225 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05

Residual standard error: 1.442 on 140 degrees of freedom
Multiple R-Squared: 0.6566,  Adjusted R-squared: 0.6493
F-statistic: 89.24 on 3 and 140 DF,  p-value: < 2.2e-16
```

Régression linéaire (3)

cats.lm est un objet de type lm. On peut visualiser ces composantes par names(cats.lm) ou attributes(cats.lm).

On peut en extraire notamment :

- les prédictions : cats.lm$fitted.values
- les coefficients du modèle : cats.lm$coefficients ou coefficients(cats.lm)
- les résidus : resid(cats.lm)
- les résidus "studentisés": studres(cats.lm)
- la formule du modèle : formula(cats.lm)

Exercice

En utilisant la régression précédente

- Dessiner :
  1. le qqplot des résidus studentisés (+ 1$^{\text{ère}}$ bissectrice)
  2. le graphe des résidus studentisés vs les prédictions
  3. le graphe des distances de Cook
- Regarder les attributs de l'objet summary(cats.lm)
- Afficher (à partir de cats.lm ou summary(cats.lm))
  1. le R^2 ajusté de la régression
  2. le nombre de degrés de liberté résiduels
  3. la matrice de variance-covariance des paramètres estimés
- Modifier la formule pour faire tourner le modèle suivant : Hwt$\sim$Bwt+Sex
  - comparer ce modèle avec le précédent en utilisant une ANOVA

Notes : fonctions associées à des objets

On a déjà vu au cours dernier avec la fonction plot qu'il y avait des fonctions surchargées capables de répondre différemment selon le type d'objet donné en argument.
On a accès à la liste des fonctions définies par methods()

```
methods(plot)
```

Symétriquement, on peut vouloir déterminer les fonctions spécifiques définies pour un type d'objet donné. On utilise l'argument class="type_d_objet" de la fonction.

```
methods(class="lm")
 [1] add1.lm*            addterm.lm*         alias.lm*           anova.lm
 [5] boxcox.lm*          case.names.lm*      confint.lm*         cooks.distance.lm* [9
...
[33] rstudent.lm         summary.lm          variable.names.lm* vcov.lm*
```

Pour appeler la fonction correspondante, on peut omettre le .lm :

```
> anova(cats.lm)
...
```

Régression logistique (1)

On utilise la base birthwt de la library MASS.

On sélectionne les variables :

- low (petit poids de naissance)
- age (age de la mère)
- smoke (tabagisme de la mère)
- ptl (nombre d'accouchements prématurés)
- ht (antécédents d'hypertension) :

On cherche à expliquer le petit poids de naissance. On utilise :

```
> lbwt <- data.frame(low=factor(birthwt$low),age=birthwt$age,
ptl=birthwt$ptl,smoke=(birthwt$smoke>0),ht=(birthwt$ht>0))
```

Régression logistique (2)

La syntaxe pour une régression logistique est :

```
> lbwt.glm <- glm(low~age+ptl+smoke+ht,data=birthwt,family=binomial)
> summary(lbwt.glm)
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.07602    0.78544   0.097   0.9229
age         -0.05977    0.03360  -1.779   0.0753 .
ptl          0.79284    0.33117   2.394   0.0167 *
smoke        0.56401    0.33545   1.681   0.0927 .
ht           1.27318    0.61991   2.054   0.0400 *

    Null deviance: 234.67  on 188  degrees of freedom
Residual deviance: 217.18  on 184  degrees of freedom
```

Régression logistique (3)

On essaie d'enlever une variable :

```
> drop1(glm(low~age+ptl+smoke+ht,data=birthwt,family=binomial)
+ ,test="Chisq")
Single term deletions
Model:
low ~ age + ptl + smoke + ht
       Df Deviance     AIC     LRT Pr(Chi)
<none>     217.177 227.177
age     1  220.513 228.513   3.336 0.06778 .
ptl     1  223.281 231.281   6.104 0.01349 *
smoke   1  219.994 227.994   2.816 0.09332 .
ht      1  221.446 229.446   4.268 0.03883 *
```

On enlève la variable ayant la p-value la plus élevée, ici smoke.

Exercice

- Terminer la sélection des variables
- Tracer des graphes pour évaluer votre modèle
- Calculer le pourcentage de prédictions correctes, calculé comme suit
  - pour chaque sujet, on prédit la probabilité d'obtenir une réponse, notée p
  - si p est supérieure ou égale à 0.5, on prédit une réponse pour ce sujet (ici, un petit poids de naissance, low=1)
  - sinon, on prédit une non-réponse (low=0)
  - on construit le tableau de contingence suivant en calculant pour chaque case le pourcentage correspondant

		Prédit	
		low=0	low=1
Observé	low=0		
	low=1		

Pour aller un peu plus loin...

On peut aussi effectuer une sélection automatique par sélection pas à pas, grâce à la fonction step, qui fait appel à drop1 et add1.

Essayer les 2 façons suivantes de procéder :

- depuis le modèle complet dont on définit d'abord la formule

```
scope.lm<-paste(names(birthwt)[2:10],collapse="+")
scope.lm<-paste(names(birthwt)[1],scope.lm,sep="~")
lbwt.full <- glm(as.formula(scope.lm),family=binomial,data=birthwt)
step(lbwt.full)
```

- depuis le modèle sans variables explicatives, en spécifiant les termes à inclure

```
lbwt.base <- glm(low~1,family=binomial,data=birthwt)
summary(lbwt.base)
```

Calcul du nombre de sujets nécessaire (1)

R intègre désormais des fonctions de calcul de puissance et du nombre de sujets pour une comparaison de 2 moyennes :

```
>  power.t.test(delta=10,sd=15,power=0.90,type="paired")

     Paired t test power calculation

              n = 25.6399
          delta = 10
             sd = 15
      sig.level = 0.05
          power = 0.9
    alternative = two.sided

 NOTE: n is number of *pairs*, sd is std.dev. of *differences* within pairs
```

Calcul du nombre de sujets nécessaire (2)

Si on renseigne n au lieu de la puissance, on obtient une estimation de cette puissance.
Pour la comparaison de 2 proportions, le principe est le même :

```
> power.prop.test(power=0.90,p1=0.5,p2=0.7)

     Two-sample comparison of proportions power calculation

              n = 123.9986
...
 NOTE: n is number in *each* group
```

Exercice

On vous demande de faire le calcul du nombre de sujets nécessaire pour un essai clinique.

- Il s'agit d'évaluer un nouveau traitement pour le cancer du larynx. On fait les hypothèses suivantes :
  - la proportion de répondeurs dans le bras référence (traité avec le traitement usuel) est de 40%
  - les fabricants du nouveau médicament espèrent un taux de réponse de 60% pour le nouveau traitement
- Calculer le nombre de sujets nécessaire, pour une puissance de 95%
- Le laboratoire pharmaceutique peut se permettre d'enrôler 2000 patients (en tout)
  - quelle différence minimale entre les deux traitements peuvent-ils mettre en évidence (on supposera que le traitement de référence est suffisamment bien connu)?

Survie : librairie survival

On commence par charger la librairie survival :

```
> library(survival)
```

On utilise la base melanom de ISwR : on s'intéresse en particulier à la durée de suivi Days et au statut status (1 : mort liée au mélanome, 2 : vivant à la fin de l'étude, 3 : mort d'une autre cause).

```
> library(ISwR)
> data(melanom)
```

On attribue d'abord les évènements à status==1:

```
> mel.surv <- Surv(melanom$days,melanom$status==1)
> mel.surv
  [1]   10}   30}   35}   99}  185   204   210   232   232}  279   295   355}
...
  [205] 5565}
```

Les censures sont marquées par un "}".

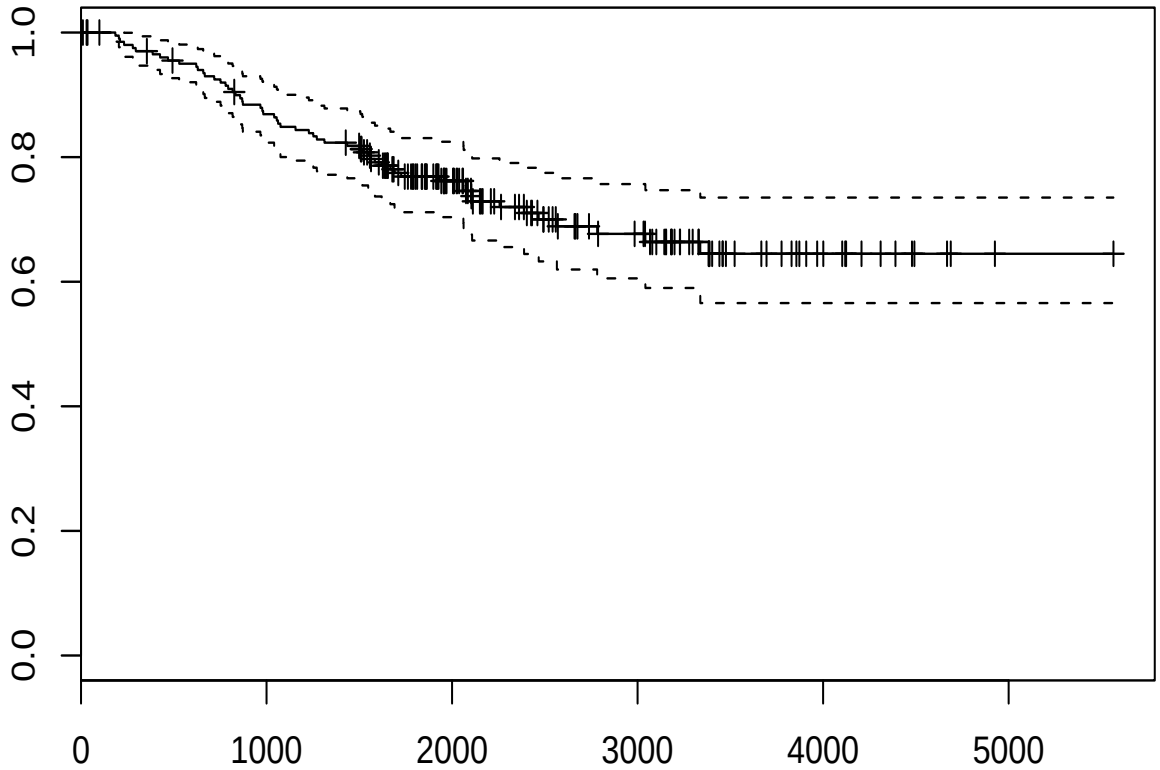
Kaplan-Meier

La syntaxe pour effectuer une analyse par Kaplan-Meier est :

```
> mel.survfit <- survfit(mel.surv)
```

Pour obtenir le graphe de la survie en fontion du suivi :

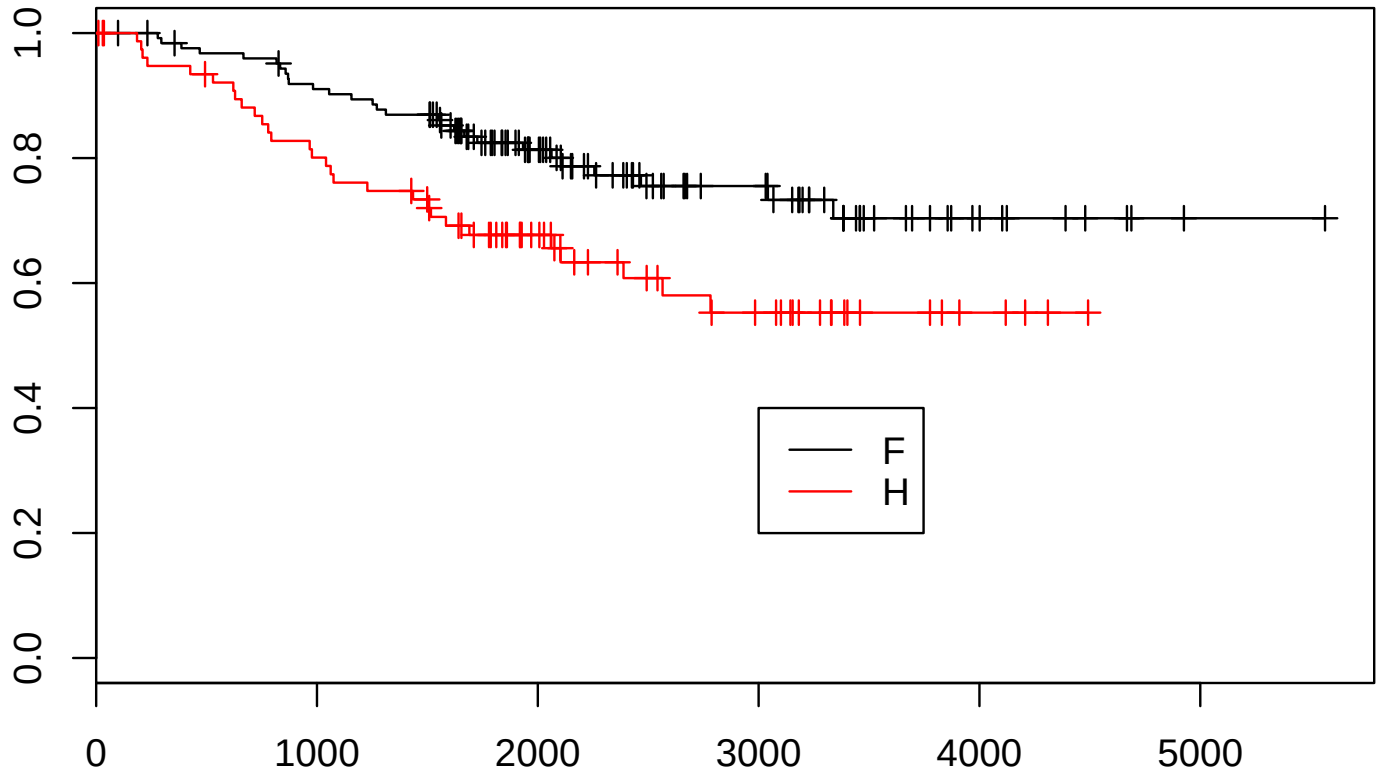
```
> plot(mel.survfit)
```



Graphique des données de survie

Pour analyser la survie selon le sexe (1 : F, 2 : H):

```
> mel.survfit2 <- survfit(mel.surv~sex)
> plot(mel.survfit2,col=c(1,2))
> legend(3000,0.4,c("F","H"),lty=c(1,1),col=c(1,2))
```



(le summary a permis de repérer les groupes)

Test du log-rank

Pour tester une différence entre les courbes de survie :

```
> survdiff(Surv(days,status==1)~sex)
Call:
survdiff(formula = Surv(days, status == 1) ~ sex)

        N Observed Expected (O-E)^2/E (O-E)^2/V
sex=1 126       28     37.1      2.25      6.47
sex=2  79       29     19.9      4.21      6.47

 Chisq= 6.5  on 1 degrees of freedom, p= 0.011
```

Par défaut on obtient un test du log-rank. L'option rho=1 donne un test de Gehan-Wilcoxon.

Modèle de Cox

La syntaxe est la suivante :

```
> mel.cox<-coxph(Surv(days, status == 1) ~ sex+strata(ulc))
> summary(mel.cox)
  n= 205

     coef exp(coef) se(coef)    z     p
sex 0.482      1.62    0.267 1.80 0.071

    exp(coef) exp(-coef) lower .95 upper .95
sex      1.62      0.618      0.96      2.73

Rsquare= 0.016   (max possible= 0.9 )
Likelihood ratio test= 3.23  on 1 df,   p=0.0721
Wald test            = 3.25  on 1 df,   p=0.0712
Score (logrank) test = 3.31  on 1 df,   p=0.0687
```

Plan

L'algorithmique

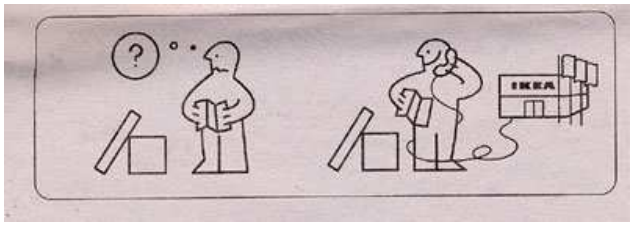
- L'algori-quoi?
  - vient du joli nom d'Al-Kwarizmi, mathématicien persan (780-850)
  - popularisé par Alan Turing, logicien anglais (1912-1954), auteur de la fameuse machine de Turing (l'ancêtre/le concept de l'ordinateur)
- D'accord, mais c'est quoi ?
- L'algorithmique est une démarche logique pour la résolution d'un problème
  - étudier les étapes nécessaires à la réalisation d'une tâche
  - décrire ce qui doit être fait pour résoudre le problème
- Avantages :
  - abstraction
  - indépendant du langage de programmation utilisé ensuite

L'algorithmique, ça ne passera pas par moi!

En fait, nous suivons tous des algorithmes sans le savoir :

- 500 g de spaghettis
- 1 oignon
- 2 gousses d'ail
- 1 carotte
- 1 branche de céleri
- 850 g de tomates
- 375 ml de vin rouge
- 500 g de viande hachée -
- 500 ml de bouillon Maggi
- persil
- 1 cuillère à café de sucre
- 2 cuillères à soupe d'huile

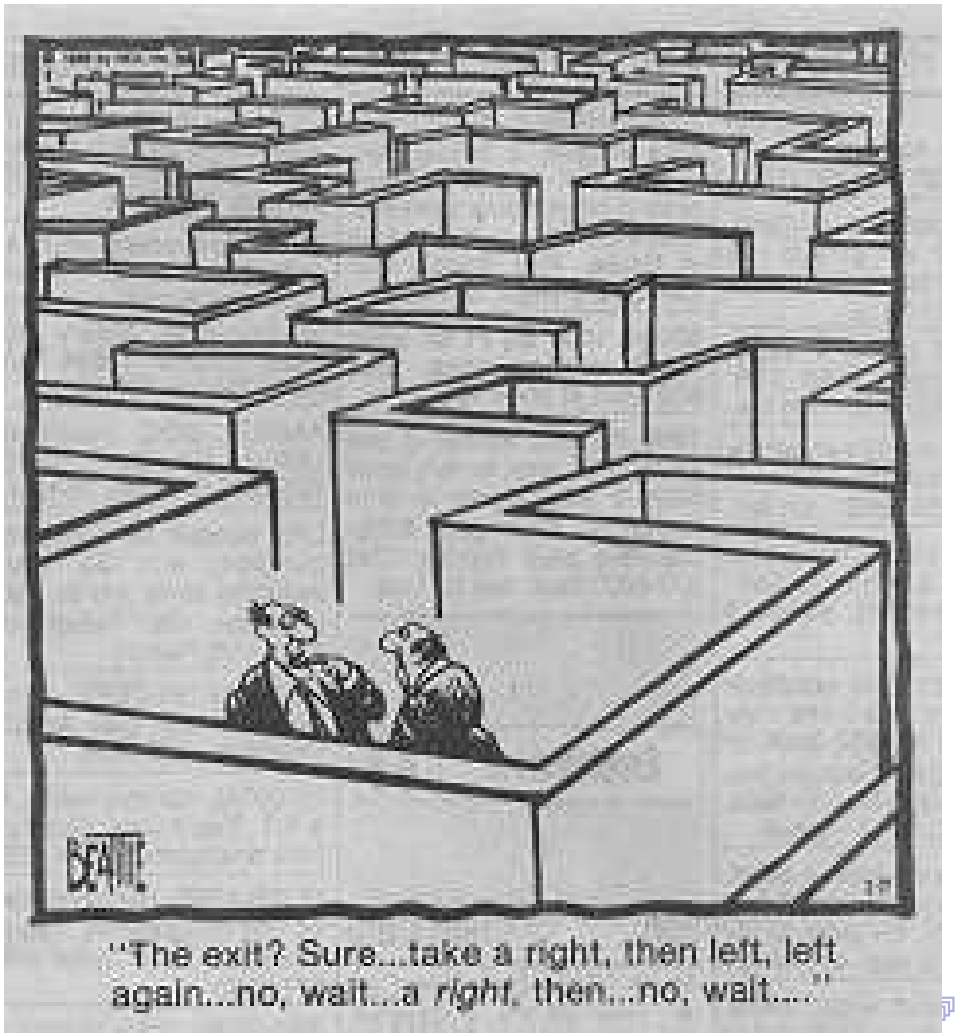
En suivant une recette de cuisine ...



En montant un meuble Ikéa

L'algorithmique, je ne sais pas faire...

S'il vous est déjà arrivé d'indiquer un chemin...



L'algorithmique, je ne sais pas faire...

... ou de faire chercher un objet à quelqu'un :

alors vous avez déjà conçu et fait exécuter un algorithme

Ca a l'air simple

- Le problème : un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter
  - exclut les références culturelles ou les sous-entendus
- En informatique, heureusement, il n'y a pas ce problème
  - les choses auxquelles ont doit donner des instructions sont les ordinateurs
  - ceux-ci ont le bon goût d'être tous strictement aussi idiots les uns que les autres
- corollaire : si votre algorithme ne marche pas, c'est que vous n'avez pas donné les bonnes instructions
  - pas la peine de l'insulter
  - mettez-vous plutôt à la place de la machine
- un algorithme qui marche, c'est :
  - 95% de rigueur
  - 5% d'intuition ou à défaut, d'expérience

Exemple d'algorithme

Algorithme pour cuire des spaghettis

1. mettre 1 litre d'eau salée dans une casserole
2. ajouter de l'huile d'olive
3. allumer le feu
   - attendre que l'eau bouille
4. au moment de l'ébullition, ajouter les spaghettis
5. réduire le feu
6. si on aime les spaghettis *al dente* les égoutter après 6 minutes
7. sinon les égoutter après 8 minutes
8. éteindre le feu et manger les spaghettis.

Structures de contrôle

- Tests
  - détermine le comportement de l'algorithme en réponse à différentes valeurs de variables
- Boucles
  - permet de répéter une opération plusieurs fois
  - liées aux tests : la boucle se répète pour un nombre fini d'itérations ou tant qu'une condition n'est pas réalisée

Exemple de boucle (1)

Algorithme du programmateur :

- Tant qu'il y a du café :

```
{
    Pianoter sur son ordinateur
    Boire du cafe
}
```

Bloc

- Se coucher

Exemple de boucle (2)

Importance de la condition d'arrêt

- Tant que Température_de_l'eau $< 100^{\circ}C$

```
{
    Ne rien faire
}
```

- Mettre les spaghettis dans l'eau

Exemple de test

Algorithme du joueur de foot

- si je mets un coup de boule alors
  - je me fais exclure
  - mon équipe perd
- sinon
  - je peux gagner la coupe du monde

Exercice

Décrire (en mots) un algorithme pour ranger les élèves de la classe du plus petit au plus grand.
Pour le fun : exécutez l'algorithme. Vérifiez que vous êtes bien rangés.

Tests

Condition :

```
if (cond) {
instructions si conditions remplies} else {
instructions si conditions non remplies}
```

Switch binaire

```
ifelse(cond, instructions_if_yes, instructions_if_no)
```

Switch à plusieurs réponses :

```
switch(expression,instructions...)
```

où il y a un jeu d'instructions pour chaque valeur possible de l'expression.

Exemple de test

Condition :

```
> xcal<-rnorm(1)
> xcal
[1] 0.2327076
> if(xcal>0) cat(log(xcal),"\n") else cat("negative number\n")
-1.457972
> xcal<-rnorm(1)
> xcal
[1] -1.296807
> if(xcal>0) cat(log(xcal),"\n") else cat("negative number\n")
negative number
```

Switch binaire :

```
> vec<-rnorm(10)
> vec
 [1]  0.8850705 -1.3299311  1.0065950 -0.6761939  0.6344586 -0.3062798
 [7] -0.6141385  0.2193833  0.3246979  1.3854910
> ifelse(vec>0,"positif","négatif")
 [1] "positif" "négatif" "positif" "négatif" "positif" "négatif" "négatif"
 [8] "positif" "positif" "positif"
```

Exercice

1. En utilisant la fonction scan (cours 2) :
   - lire un nombre au clavier
   - afficher "Ce nombre est positif" ou "Ce nombre est négatif" de façon appropriée
2. Ecrire un script qui :
   - tire un nombre entier au hasard entre 0 et 100
   - lit un nombre au clavier donné par l'utilisateur
   - si le nombre correspond, afficher "c'est gagné!"
   - sinon, afficher "trop haut" ou "trop bas" selon que le chiffre donné par l'utilisateur est supérieur ou inférieur au chiffre initial

Boucles (1)

Pour effectuer n fois une opération, on utilise
`for (i in vec) {MonAction}`

```
> y<-NULL
> for (i in 1:10) y[i]<-i
> y
 [1]  1  2  3  4  5  6  7  8  9 10
```

ou encore

```
> y<-c()
> for (i in 1:10) {
y<-c(y,i)
}
> y
 [1]  1  2  3  4  5  6  7  8  9 10
```

Boucles (2)

On peut mettre n'importe quel vecteur (ou assimilé) dans les valeurs possibles de i. Exemple de petit programme :

```
prenom <- c("John","Paul","George")
nbamis<-0
for(i in prenom) {
cat(i,"est un membre du groupe \n")
nbamis<-nbamis+1
}
cat("Il y a",nbamis,"amis dans ce groupe\n")
```

qui donne à l'exécution

```
John est un membre du groupe
Paul est un membre du groupe
George est un membre du groupe
Il y a 3 amis dans ce groupe
```

Les accolades ici sont obligatoires car il y a plusieurs instructions dans la boucle. Si on les oublie, le compteur nbamis ne sera incrémenté qu'une fois, après avoir effectué la boucle.

Boucles (3)

Reprenons notre matrice x :

```
> x<-matrix(1:12,ncol=4)
> x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Pour afficher une à une les lignes de la matrice, on fait une boucle sur un indice i qui va de 1 à la hauteur de la matrice, et on lui demande d'afficher la ligne correspondante à chaque étape de la boucle :

```
> for(i in 1:3) cat(x[i,],"\n")
1 4 7 10
2 5 8 11
3 6 9 12
```

Boucle "optimisée"

L'utilisation de `for` a tendance à ralentir les calculs. On fait alors appel à `apply` (et aussi `sapply, lapply, tapply`) pour appliquer une fonction à plusieurs colonnes/lignes d'une matrice/dataframe.
Syntaxe :

```
apply(x,Margin,fonction,arguments)
```

où Margin vaut 1 pour des lignes et 2 pour des colonnes.

```
> x <- matrix(1:25,ncol=5)
> apply(x,1,sum)
[1] 55 60 65 70 75
[1] 55 60 65 70 75
```

somme chaque ligne de la matrice x.

Equivalent à :

```
for (i in 1:5) {
cat(sum(x[i,])," ")
}
```

Boucles "tant que"

Boucle tant que la condition est vraie : `while`

```
while(cond) {
instr}
```

Boucle jusqu'à : `repeat`

```
repeat{
instr
if(cond) break
}
```

Note : cette boucle s'exécute toujours au moins une fois contrairement à la précédente.

Exercice

Soit la matrice :

```
x<-matrix(1:120,ncol=12)
```

- Utilisez une boucle pour afficher les moyennes de chaque ligne et les ranger dans un vecteur lmoy
- Utilisez une boucle pour afficher les moyennes de chaque colonne et les ranger dans un vecteur cmoy
- Utilisez la fonction apply pour faire la même chose et vérifiez que vous obtenez les mêmes résultats

Exercice

1. Utiliser une boucle pour afficher une par une les lettres des mots suivants, en utilisant les vecteurs LETTERS (majuscules) et letters (minuscules)
   - ABC
   - exercice
2. À l'aide de la commande apply, écrire des expressions R qui remplaceraient les fonctions suivantes :

```
a) rowSums
b) colSums
c) rowMeans
d) colMeans
```

Vérifiez vos résultats en utilisant ces fonctions (consultez l'aide de chaque fonction pour savoir comment l'utiliser).

Plan

Théorème de la limite centrale

Theorem

Soit X_n une suite de v.a. de même loi d'espérance μ et d'écart type σ. Alors la variable aléatoire $\left(\frac{X_1+X_2+...+X_n-n\mu}{\sigma\sqrt{n}}\right)$ converge en loi vers une variable aléatoire normale centrée réduite $\aleph(0,1)$.

Le petit exercice suivant se propose d'illustrer ce théorème.

Illustration du théorème centrale limite (1)

- Simuler dans une loi binomiale de paramètre $p = 0.2$
  - 1000 échantillons de taille 5
  - 1000 échantillons de taille 30,
  - 1000 échantillons de taille 100
- Pour chaque taille d'échantillon
  - calculer et stocker la moyenne des échantillons simulés
  - afficher l'histogramme des moyennes des échantillons simulés
  - afficher le QQ-plot correspondant

Illustration du théorème centrale limite (2)

- Tirer 1000 échantillons de taille 100 dans
  - une loi normale centrée réduite
  - une loi normale de moyenne 4 et d'écart-type 2
  - une loi uniforme
- Pour chaque distribution
  - calculer et stocker la moyenne des échantillons simulés
  - créér une fenêtre de graphe partagée en 2
  - afficher l'histogramme d'un échantillon simulé
  - afficher l'histogramme des moyennes des échantillons simulés

Illustration du théorème centrale limite (3)

Pour un échantillon binomial de taille 30 :

```
##génération des données
mean2 = rep(NA, 1000)
for (i in 1:1000) {
mean2[i] = mean(runif(30, min=0, max=1))
}
##histogramme

par(mfrow=c(2,1))
hist(mean2, freq=F)
x2 = seq(min(mean2), max(mean2), 0.01)
lines(x2, dnorm(x2, mean=mean(mean2), sd=sqrt(var(mean2))),
type="l", lty=2, col=2)

##qq-plot
qqnorm(mean2)
qqline(mean2)
```

Exercice

- Reprendre l'exercice sur le théorème central limite
  - à partir des simulations réalisées avec une loi exponentielle, tester pour les différentes valeurs des moyennes si la distribution des moyennes est significativement différente d'une loi normale