# Initiation à la statistique avec R, code et compléments chapitre 9

*Frédéric Bertrand et Myriam Maumy-Bertrand*

*11 décembre 2018*

```
#Chapitre 9
require(BioStatR)
```

```
## Loading required package: BioStatR
```

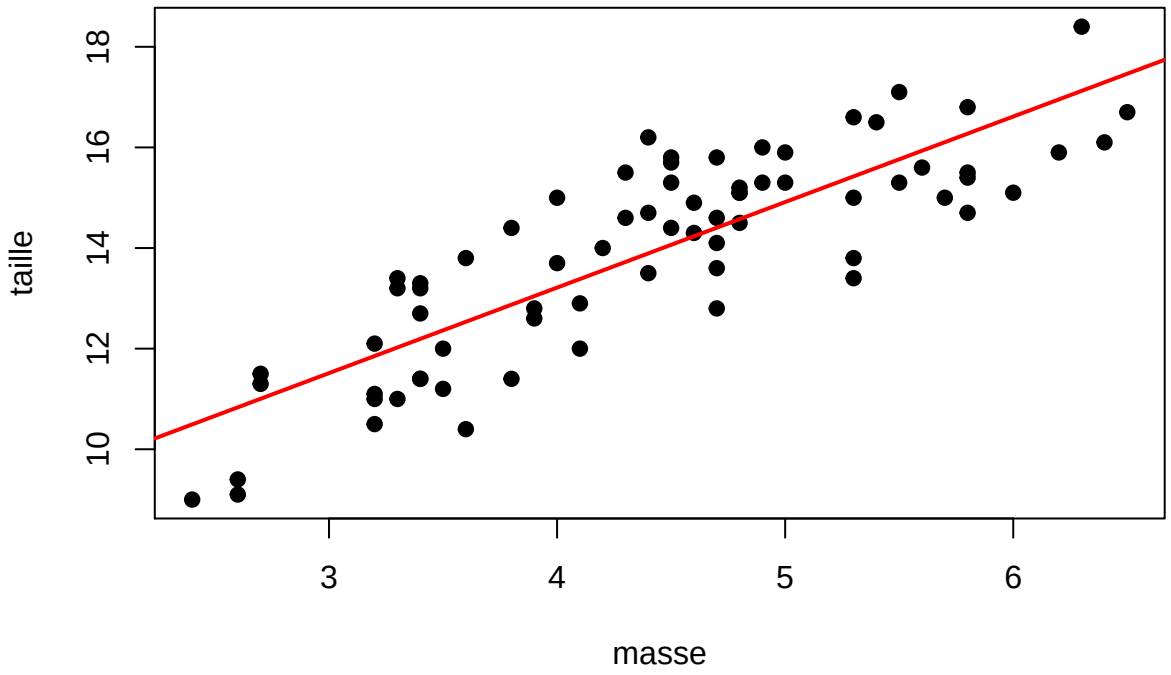
```
#page 377
#Exercice 9.1
#1)
lauriers<-subset(Mesures,subset=(Mesures$espece=="laurier rose"))
plot(taille~masse,data=lauriers,pch=19)

#page 378
#3)
droite_lauriers<-lm(taille~masse,data=lauriers)
coef(droite_lauriers)
```

```
## (Intercept)       masse
##    6.413523    1.700114
```

```
#4)
fitted(droite_lauriers)
```

```
##      181      182      183      184      185      186      187      188
## 14.74408 16.95423 13.21398 12.02390 14.57407 15.93416 14.06404 17.12424
##      189      190      191      192      193      194      195      196
## 13.55400 13.04397 16.27419 14.40406 16.61421 17.46427 14.91409 15.76415
##      197      198      199      200      201      202      203      204
## 14.40406 16.10418 12.53393 15.59414 15.42413 14.91409 14.06404 13.89403
##      205      206      207      208      209      210      211      212
## 14.57407 14.06404 11.85389 14.40406 13.21398 16.27419 15.76415 13.89403
##      213      214      215      216      217      218      219      220
## 12.36392 13.89403 13.72401 13.38399 15.42413 14.40406 15.42413 14.40406
##      221      222      223      224      225      226      227      228
## 14.74408 13.38399 14.23405 14.57407 12.19391 12.19391 16.27419 14.57407
##      229      230      231      232      233      234      235      236
## 13.04397 12.19391 14.06404 12.02390 12.02390 12.53393 12.36392 12.87396
##      237      238      239      240      241      242      243      244
## 11.85389 12.87396 15.42413 16.27419 14.23405 11.85389 13.72401 11.00383
##      245      246      247      248      249      250      251      252
## 10.83382 10.49380 10.83382 11.85389 17.29426 12.19391 12.19391 11.00383
```

```
#page 379
#5)
abline(coef(droite_lauriers),col="red",lwd=2)
```




```
#6)
predict(droite_lauriers,(masse=4.8))
```

```
##        1
## 14.57407
```

```
#fonctionne comme predict(droite_lauriers,list(masse=4.8))
#7)
residuals(droite_lauriers)[lauriers$masse==4.8]
```

```
##         185         205         224         228
##  0.52592789  0.62592789 -0.07407211  0.52592789
```

```
#page 380
#8)
mean(lauriers$taille)
```

```
## [1] 13.91528
```

```
6.413523+1.700114*mean(lauriers$masse)
```

```
## [1] 13.91528
```

```
coef(droite_lauriers)[1]+coef(droite_lauriers)[2]*mean(lauriers$masse)
```

```
## (Intercept)
##    13.91528
```

```
#9)
summary(droite_lauriers)
```

```
##
## Call:
## lm(formula = taille ~ masse, data = lauriers)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -2.1339 -0.8590  0.1310  0.9259  2.3060
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.4135     0.5924   10.83   <2e-16 ***
## masse         1.7001     0.1309   12.99   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.12 on 70 degrees of freedom
## Multiple R-squared:  0.7068, Adjusted R-squared:  0.7026
## F-statistic: 168.8 on 1 and 70 DF,  p-value: < 2.2e-16
```

```
#page 381
#10)
anova(droite_lauriers)
```

```
## Analysis of Variance Table
##
## Response: taille
##           Df Sum Sq Mean Sq F value    Pr(>F)
## masse      1 211.57 211.573  168.76 < 2.2e-16 ***
## Residuals 70  87.76   1.254
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#11)
summary(droite_lauriers)
```

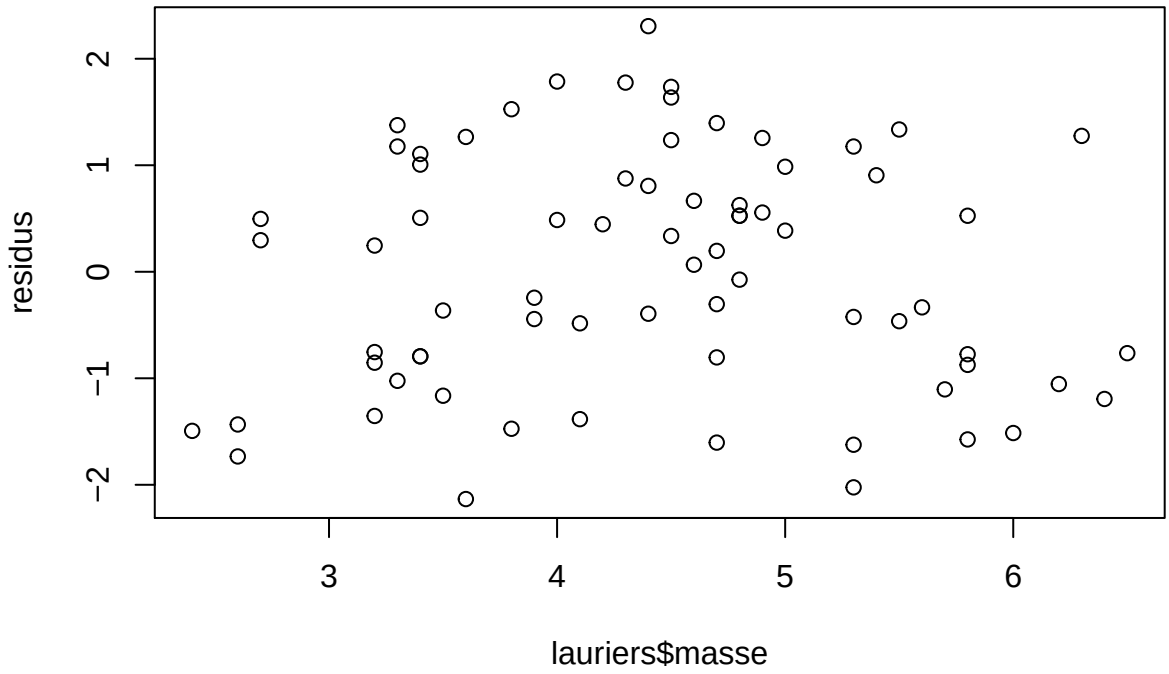
```
##
## Call:
## lm(formula = taille ~ masse, data = lauriers)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1339 -0.8590  0.1310  0.9259  2.3060
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   6.4135     0.5924   10.83   <2e-16 ***
## masse         1.7001     0.1309   12.99   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.12 on 70 degrees of freedom
## Multiple R-squared:  0.7068, Adjusted R-squared:  0.7026
## F-statistic: 168.8 on 1 and 70 DF,  p-value: < 2.2e-16
```

```
#page 382
#12)
residus<-residuals(droite_lauriers)
shapiro.test(residus)
```

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  residus
## W = 0.96531, p-value = 0.04439
```

```
#page 383
plot(lauriers$masse,residus)
```




```
pdf("residusmasse.pdf")
plot(lauriers$masse,residus)
dev.off()
```

```
## pdf
##   2
```

```
#Les résidus ont l'air corrects => homoscédasticité des erreurs ok et
#absence d'effet systématique
#Approche par permutation valide

#13)
if(!("lmPerm" %in% rownames(installed.packages()))){install.packages("lmPerm")}
library(lmPerm)
lmp(taille~masse,lauriers)
```

```
## [1] "Settings:  unique SS : numeric variables centered"
```

```
##
## Call:
## lmp(formula = taille ~ masse, data = lauriers)
##
## Coefficients:
## (Intercept)        masse
##       13.92         1.70
```

```
#page 384
perm_laurier<-lmp(taille~masse,lauriers,center=FALSE)
```

```
## [1] "Settings:  unique SS "
```

```r
summary(perm_laurier)
```

```
##
## Call:
## lmp(formula = taille ~ masse, data = lauriers, center = FALSE)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1339 -0.8590  0.1309  0.9259  2.3060
##
## Coefficients:
##       Estimate Iter Pr(Prob)
## masse      1.7 5000   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.12 on 70 degrees of freedom
## Multiple R-Squared: 0.7068,  Adjusted R-squared: 0.7026
## F-statistic: 168.8 on 1 and 70 DF,  p-value: < 2.2e-16
```

```r
#page 385
#14)
confint(droite_lauriers)
```

```
##                2.5 %   97.5 %
## (Intercept) 5.232099 7.594947
## masse       1.439098 1.961131
```

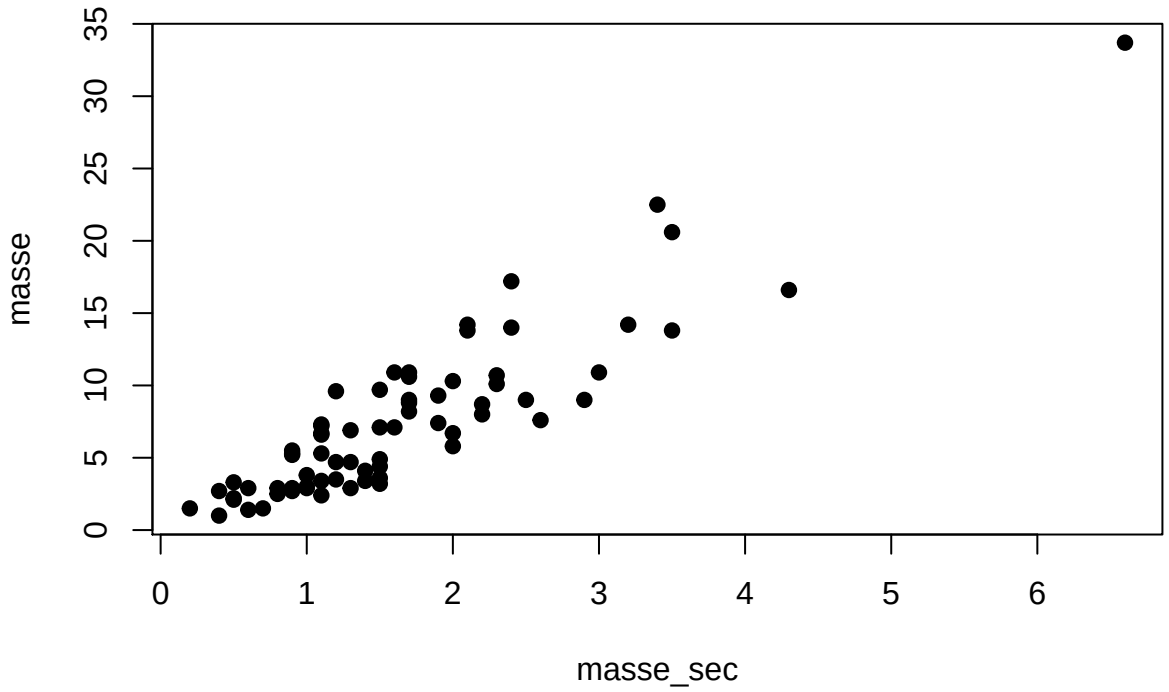
```r
predict(droite_lauriers,list(masse=c(4.8)),interval="confidence")
```

```
##        fit      lwr      upr
## 1 14.57407 14.29212 14.85602
```

```r
predict(droite_lauriers,list(masse=c(4.8)),interval="prediction")
```

```
##        fit      lwr      upr
## 1 14.57407 12.32318 16.82496
```

```r
#page 386
#Exercice 9.2
#1)
bignones<-subset(Mesures5,subset=(Mesures5$espece=="bignone"))[,c(1,4)]
plot(masse~masse_sec,data=bignones,pch=19)
```



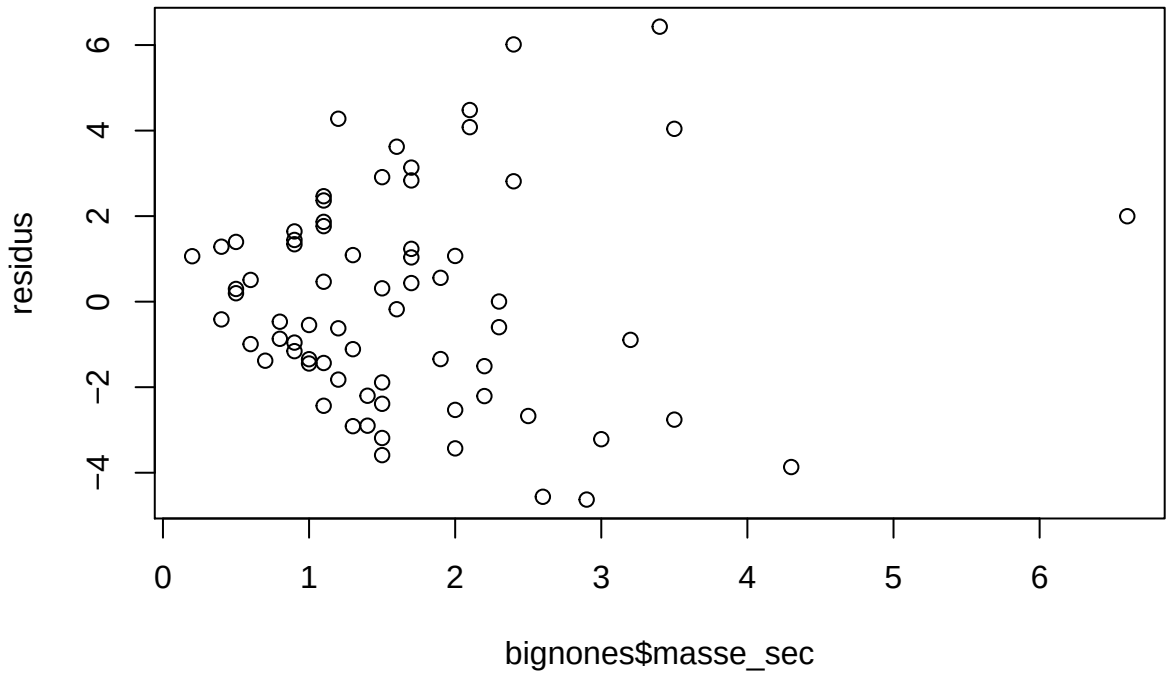

```
pdf("figure94.pdf")
plot(masse~masse_sec,data=bignones,pch=19)
dev.off()
```

```
## pdf
##   2
```

```
#3)a)
droite_bignones<-lm(masse~masse_sec,data=bignones)
coef(droite_bignones)
```

```
## (Intercept)   masse_sec
##  -0.5391407   4.8851935
```

```
#page 387
residus<-residuals(droite_bignones)
plot(bignones$masse_sec,residus)
```




```
pdf("figure95.pdf")
plot(bignones$masse_sec,residus)
dev.off()
```

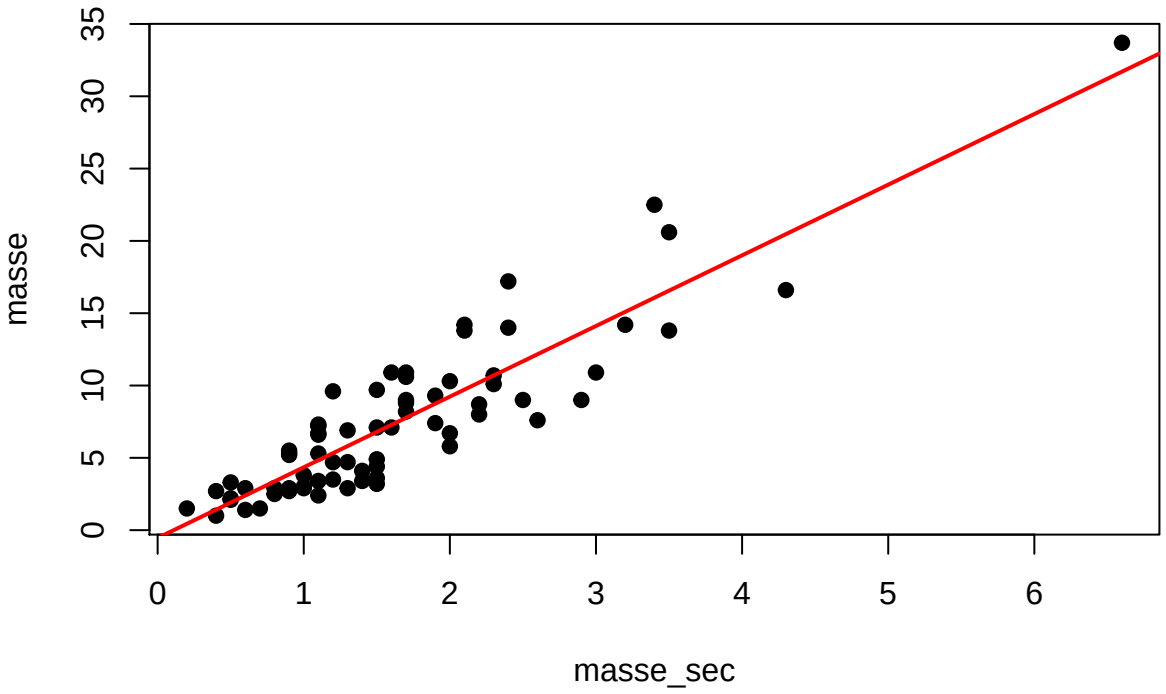
```
## pdf
##   2
```

```
#Les résidus n'ont l'air corrects car ils présentent une forme en trompette,
#ce qui remet en question de l'homoscédasticité des erreurs. Nous procéderons
#dans la suite à un test pour nous assurer que ce défaut est significatif au
#seuil de \alpha=5%. Par contre les résidus ont l'air répartis aléatoirement
#au-dessus ou en-dessous de l'axe des abscisses. Vous notez également l'absence
#d'un effet systématique qui se traduirait par exemple par une forme de banane.
#L'hypothèse d'indépendance n'est pas remise en question.
#Malgré l'inhomogénéité des variances l'estimation de la pente et de l'ordonnée
#à l'origine reste sans biais. Il sera, par contre, nécessaire tenir compte de
#l'hétéroscédasticté des erreurs pour la mise en oeuvre des procédures de test et
#la construction des intervalles de confiance.

#page 388
#4)
fitted(droite_bignones)
```

```
##       111       112       113       114       115       116       117
## 14.116440  4.834572 16.070517 31.703136 16.559037 20.467191  9.719766
##       118       119       120       121       122       123       124
##  9.719766 11.185324 10.208285 15.093478  7.765688  7.277169  1.903456
##       125       126       127       128       129       130       131
##  6.788650  8.742727 11.185324 10.696804 13.627920  6.788650  7.277169
##       132       133       134       135       136       137       138
##  0.437898  6.300130 10.208285  8.742727  4.834572  5.811611  3.369014
##       139       140       141       142       143       144       145
##  4.834572 10.696804 16.559037  7.765688  9.231246  7.765688 11.673843
```

```
##       146       147       148       149       150       151       152
##  7.765688  5.323091  7.765688  3.857533  2.880495  4.834572  2.391975
##       153       154       155       156       157       158       159
##  4.346053  5.323091  4.834572  6.788650  5.323091  5.811611  3.857533
##       160       161       162       163       164       165       166
##  1.903456  1.903456  2.391975  1.414937  1.414937  3.369014  3.857533
##       167       168       169       170       171       172       173
##  3.857533  9.231246  4.834572  5.811611  4.346053 12.162362  6.788650
##       174       175       176       177       178       179       180
##  4.346053  9.231246  4.834572  6.788650  6.788650  6.300130  3.857533
```

```
#5)
plot(masse~masse_sec,data=bignones,pch=19)
abline(coef(droite_bignones),col="red",lwd=2)
```



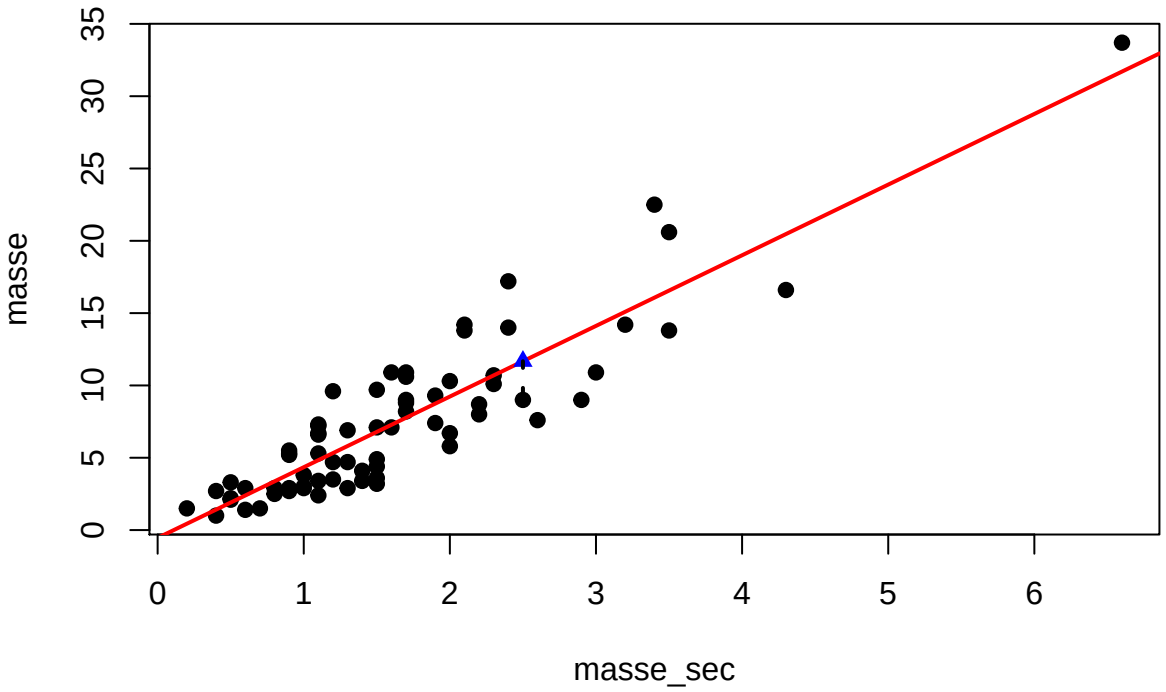

```
pdf("figure96.pdf")
plot(masse~masse_sec,data=bignones,pch=19)
abline(coef(droite_bignones),col="red",lwd=2)
dev.off()
```

```
## pdf
##   2
```

```
#6)
predict(droite_bignones,(masse_sec=2.5))
```

```
##        1
## 11.67384
```

```
plot(masse~masse_sec,data=bignones,pch=19)
abline(coef(droite_bignones),col="red",lwd=2)
points(2.5,predict(droite_bignones,(masse_sec=2.5)),pch=17,col="blue")
segments(2.5, bignones$masse[bignones$masse_sec==2.5],2.5,
  predict(droite_bignones,(masse_sec=2.5)),lty=2,lwd=2)
```




```
pdf("figure96residusmasselinepoint.pdf")
plot(masse~masse_sec,data=bignones,pch=19)
abline(coef(droite_bignones),col="red",lwd=2)
points(2.5,predict(droite_bignones,(masse_sec=2.5)),pch=17,col="blue")
segments(2.5, bignones$masse[bignones$masse_sec==2.5],2.5,
  predict(droite_bignones,(masse_sec=2.5)),lty=2,lwd=2)
dev.off()
```

```
## pdf
##   2
```

```
#page 389
#7)
residuals(droite_bignones)[bignones$masse_sec==2.5]
```

```
##       145
## -2.673843
```

```
#8)
mean(bignones$masse)
```

```
## [1] 7.521429
```

```
-0.5391407+4.8851935*mean(bignones$masse_sec)
```

```
## [1] 7.521429
```

```
coef(droite_bignones)[1]+coef(droite_bignones)[2]*mean(bignones$masse_sec)
```

```
## (Intercept)
##    7.521429
```

```
#page 390
#9)
summary(droite_bignones)
```

```
##
## Call:
## lm(formula = masse ~ masse_sec, data = bignones)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.6279 -1.7444 -0.2961  1.4310  6.4295
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.5391     0.5657  -0.953    0.344
## masse_sec     4.8852     0.2912  16.774   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.497 on 68 degrees of freedom
## Multiple R-squared:  0.8054, Adjusted R-squared:  0.8025
## F-statistic: 281.4 on 1 and 68 DF,  p-value: < 2.2e-16
```

```
#10)
anova(droite_bignones)
```

```
## Analysis of Variance Table
##
## Response: masse
##           Df  Sum Sq Mean Sq F value    Pr(>F)
## masse_sec  1 1754.44 1754.44  281.36 < 2.2e-16 ***
## Residuals 68  424.01    6.24
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#page 391
#12) et 13)
residus<-residuals(droite_bignones)
shapiro.test(residus)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residus
## W = 0.98209, p-value = 0.4161
```

```
length(residus)
```

```
## [1] 70
```

```
#Les résidus sont au nombre de 70 supérieur ou égal à 30. Le test de normalité
#est donc fiable. La $p$-valeur du test est strictement supérieure à \alpha=5%,
#le test n'est pas significatif. Nous conservons, par défaut, l'hypothèse H0
#de normalité des erreurs.

#page 392
#Le test de White est un cas particulier du test de Breusch-Pagan qui est
#disponible dans le bibliothèque lmtest
if(!("lmtest" %in% rownames(installed.packages()))){install.packages("lmtest")}
library(lmtest)
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
bptest(droite_bignones, ~ masse_sec + I(masse_sec^2), data = bignones)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  droite_bignones
## BP = 20.238, df = 2, p-value = 4.031e-05
```

```
## White test (Table 5.1, p. 113)
#bptest(cig_lm2, ~ income * price + I(income^2) + I(price^2), data = CigarettesB)

#Le test de White permet de s'intéresser aux deux hypothèses :
#"H0 : les erreurs sont homoscédastiques"
#contre
#"H1 : les erreurs sont hétéroscédastiques".
#L'hypothèse de normalité des erreurs n'a été remise en cause, le test de White
#est donc fiable. La $p$-valeur du test est inférieure ou égale à \alpha=5%,
#le test est significatif. Nous rejetons l'hypothèse H0 d'homoscédasticité
#des erreurs et décidons que l'hypothèse alternative d'hétéroscédasticité
#des erreurs est vraie.

#Comme nous l'avions perçu graphiquement, les erreurs ne sont pas homoscédastiques,
#il faut tenir compte de cette inhomogénéité des variances lors de l'estimation
#des paramètres du modèle puis de la mise en oeuvre des tests de student ou
#du test global de Fisher pour la régression.
if(!("sandwich" %in% rownames(installed.packages()))){install.packages("sandwich")}
library(sandwich)
vcovHC(droite_bignones)
```

```
##             (Intercept)  masse_sec
## (Intercept)   0.2782291 -0.1714076
## masse_sec    -0.1714076  0.1397390
```

```
#Estimation, tenant de l'inhomogénéité des variances, de la matrice de
#variance-covariance des estimateurs \hat\beta_0 et \hat\beta_1.
coeftest(droite_bignones, df="inf", vcov=vcovHC)
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.52747 -1.0221   0.3067
## masse_sec    4.88519    0.37382 13.0684   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Tests de student des coefficient \beta_0 et \beta_1.

#page 393
```

```r
waldtest(droite_bignones, vcov=vcovHC)
```

```
## Wald test
##
## Model 1: masse ~ masse_sec
## Model 2: masse ~ 1
##   Res.Df Df      F    Pr(>F)
## 1     68
## 2     69 -1 170.78 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#Tests de Fihser global du modèle de régression linéaire simple.

#Pour construire les intervalles de confiance autour des paramètres,
#vous poouvez utiliser la bibliothèque hcci.
if(!("hcci" %in% rownames(installed.packages()))){install.packages("hcci")}
library(hcci)
?hcci
#L'aide de la bibliothèque HCCI vous apprend qu'il existe plusieurs procédures
#permettant de tenir compte de l'hétéroscédasticité. La fonction vcovHC utilise
#la méthode HC3 par défaut  La fonction HC, la méthode HC4 avec le paramètre k=0.7
#par défaut. Les méthodes HC3, HC4 et HC5 sont recommendées. En comparant leurs
#résultats, vous constatez qu'elles aboutissent toutes aux mêmes conclusions
#au seuil de \alpha=5% : conservation, par défaut, de "H0 : \beta_0=0" pour
#le test de l'ordonnée à l'origine et décision que "H1 : \beta_1<>0" est vraie.
HC(droite_bignones,method=3)
```

```
##             [,1]       [,2]
## [1,]  0.2782291 -0.1714076
## [2,] -0.1714076  0.1397390
```

```r
coeftest(droite_bignones, df="inf", vcov=HC(droite_bignones,method=3))
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.52747 -1.0221   0.3067
## masse_sec    4.88519    0.37382 13.0684   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
#page 394
vcovHC(droite_bignones,type="HC4")
```

```
##             (Intercept)  masse_sec
## (Intercept)   0.4035131 -0.2603917
## masse_sec    -0.2603917  0.2022249
```

```r
coeftest(droite_bignones, df="inf", vcov=vcovHC(droite_bignones,type="HC4"))
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.63523 -0.8487    0.396
```

```
## masse_sec    4.88519    0.44969 10.8634   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
vcovHC(droite_bignones,type="HC4m")
```

```
##             (Intercept)  masse_sec
## (Intercept)   0.3020561 -0.1891167
## masse_sec    -0.1891167  0.1526165
```

```
coeftest(droite_bignones, df="inf", vcov=vcovHC(droite_bignones,type="HC4m"))
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.54960  -0.981   0.3266
## masse_sec    4.88519    0.39066  12.505   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#page 395
HC(droite_bignones,method=4,k=0.7)
```

```
##            [,1]       [,2]
## [1,]  0.4035131 -0.2603917
## [2,] -0.2603917  0.2022249
```

```
coeftest(droite_bignones, df="inf", vcov=HC(droite_bignones,method=4,k=0.7))
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.63523 -0.8487    0.396
## masse_sec    4.88519    0.44969 10.8634   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
vcovHC(droite_bignones,type="HC5")
```

```
##             (Intercept)  masse_sec
## (Intercept)   0.4141695 -0.2662869
## masse_sec    -0.2662869  0.2047638
```

```
coeftest(droite_bignones, df="inf", vcov=vcovHC(droite_bignones,type="HC5"))
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.64356 -0.8377   0.4022
## masse_sec    4.88519    0.45251 10.7958   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
HC(droite_bignones,method=5)
```

```
##            [,1]       [,2]
```

```
## [1,]  0.4141695 -0.2662869
## [2,] -0.2662869  0.2047638
```

```
#page 396
coeftest(droite_bignones, df="inf", vcov=HC(droite_bignones,method=5))
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.53914    0.64356 -0.8377   0.4022
## masse_sec    4.88519    0.45251 10.7958   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
#Passons à la construction d'intervalles de confiance sur les paramètres
#\beta_0 et \beta_1 de la régression linéaire simple. Nous devons passer par
#cette étape de réécriture du modèle pour pouvoir utiliser les fonctions Pboot
#et Tbootde la bibliothèque hcci.
y = bignones$masse
x = bignones$masse_sec
model = lm(y ~ x)

#Il est possible de "fixer" le point de départ du générateur aléatoir
#pour avoir des résultats reproductibles à l'aide de la fonction set.seed
set.seed(123456)

#Commencez par utiliser une technique de bootstrap simple.
#Bootstrap percentile simple.
Pboot(model, significance = 0.05, double = FALSE, J=1000, K = 100,
  distribution = "rademacher")
```

```
## $beta
## [1] -0.5391407  4.8851935
##
## $ci_lower_simple
## [1] -1.445573  4.234628
##
## $ci_upper_simple
## [1] 0.4012009 5.5571099
##
## $ci_lower_double
## logical(0)
##
## $ci_upper_double
## logical(0)
```

```
#page 397
#Bootstrap t simple.
Tboot(model, significance = 0.05, double = FALSE, J=1000, K = 100,
  distribution = "rademacher")
```

```
## $beta
## [1] -0.5391407  4.8851935
##
## $ci_lower_simple
## [1] -1.702466  3.985557
```

```
##
## $ci_upper_simple
## [1] 0.8034019 5.7655811
##
## $ci_lower_double
## logical(0)
##
## $ci_upper_double
## logical(0)
##
## $J
## [1] 1000
##
## $K
## [1] 100
```

```
#Utilisez maintenant une technique de bootstrap double.
#Bootstrap percentile double.
Pboot(model, significance = 0.05, double = TRUE, J=1000, K = 100,
  distribution = "rademacher")
```

```
## $beta
## [1] -0.5391407  4.8851935
##
## $ci_lower_simple
## [1] -1.489805  4.194128
##
## $ci_upper_simple
## [1] 0.4318525 5.5625712
##
## $ci_lower_double
## [1] -1.636889  3.866187
##
## $ci_upper_double
## [1] 0.5447114 5.9268548
```

```
#Bootstrap t double.
Tboot(model, significance = 0.05, double = TRUE, J=1000, K = 100,
  distribution = "rademacher")
```

```
## $beta
## [1] -0.5391407  4.8851935
##
## $ci_lower_simple
## [1] -1.826238  3.955372
##
## $ci_upper_simple
## [1] 0.656839 5.789707
##
## $ci_lower_double
## [1] -2.542540  3.800285
##
## $ci_upper_double
## [1] 1.430748 5.924364
##
## $J
```

```
## [1] 1000
##
## $K
## [1] 100
```

```
#Le modèle étant hétéroscédastique, la construction d'intervalles de prédiction
#n'est pas fiable
```