TP Statistiques

Gabriel CANAPLE, BOYER THOMAS, LECLUSE Martin, FAYALA Mohamed

19 Janvier 2024

(décrire la régression, l'état des résidus, le test de normalité, est-elle centrée, variance est constante ? indépendance ?) test de shapiro, student, chi2 régression linéaire = chap 6 du cours y = ax + b + epsilon y = variable expliquée a, b = inconnues x = variable explicative epsilon = résidus test de pertinence : a=0? test de biais : b=0?

on peut aussi avoir plusieurs variables explicatives : y = a1$x1$ + $a2$x2 + ... + an*xn + epsilon

PENSER A définir le risque alpha pour chacun des tests (si on sait pas quoi prendre prendre 5%)

Introduction

L'objectif du TP est de comparer les performances d'algorithmes de calcul de circuit hamiltonien dans un graphe, selon les critères de longueur du chemin, et de temps d'exécution.

Nos conclusions se basent tout au long du TP sur les méthodes statistiques abordées en cours : test d'hypothèse, tests multiples, régression linéaire.

Dans une première partie, nous montrerons une visualisation du problème résolu par les algorithmes, puis nous comparerons ces derniers sur le critère de la longueur du chemin hamiltonien trouvé, et ensuite sur le temps d'exécution. Enfin, nous essayerons de proposer une régression linéaire de la complexité de l'algorithme Branch & Bound.

1. Visualisation de chemins

Voici les données présentes dans le fichier "DonneesGPSvilles.csv" :

```
set.seed(35)
villes <- read.csv('DonneesGPSvilles.csv',header=TRUE,dec='.',sep=';',quote="\"")
str(villes)
```

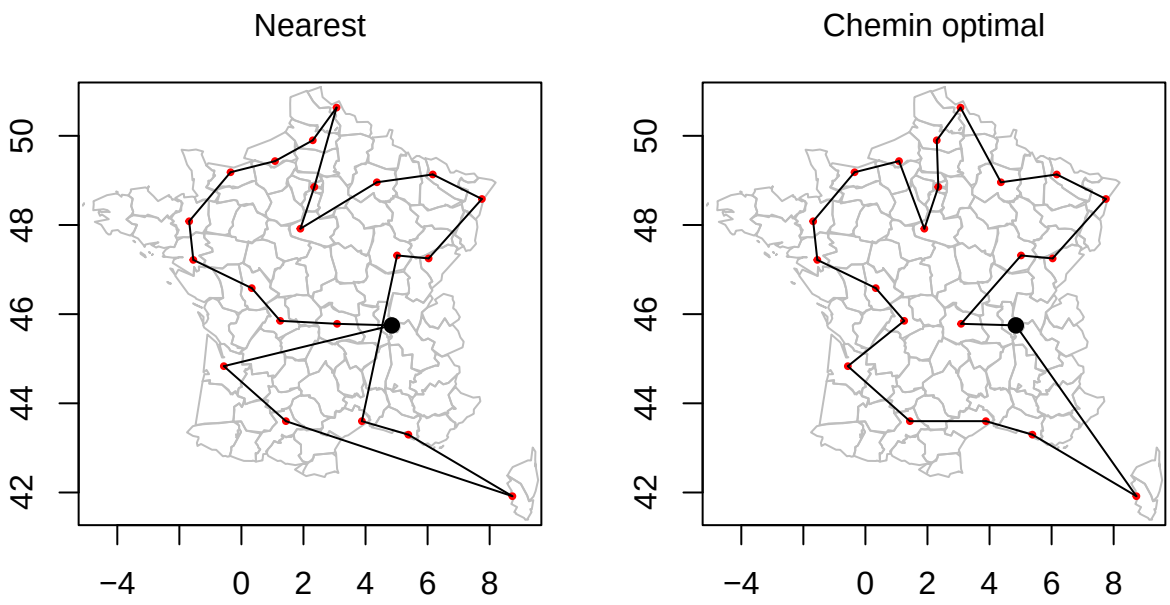
```
## 'data.frame':    22 obs. of  5 variables:
##  $ EU_circo : chr  "Sud-Est" "Sud-Est" "Nord-Ouest" "Est" ...
##  $ region   : chr  "Rhône-Alpes" "Corse" "Picardie" "Franche-Comté" ...
##  $ ville    : chr  "Lyon" "Ajaccio" "Amiens" "Besançon" ...
##  $ latitude : num  45.7 41.9 49.9 47.2 44.8 ...
##  $ longitude: num  4.847 8.733 2.3 6.033 -0.567 ...
```

Et voici comment nous visualisons les résultats données par les algorithmes (ici, avec l'exemple de l'algorithme "nearest", comparé au chemin optimal fourni) :

```r
set.seed(35)
coord <- cbind(villes$longitude,villes$latitude)
dist <- distanceGPS(coord)
voisins <- TSPnearest(dist)

pathOpt <- c(1,8,9,4,21,13,7,10,3,17,16,20,6,19,15,18,11,5,22,14,12,2)

par(mfrow=c(1,2),mar=c(1,1,2,1))
plotTrace(coord[voisins$chemin,], title='Nearest')
plotTrace(coord[pathOpt,], title='Chemin optimal')
```



Le total de la longueurs des trajets (à vol d'oiseau) vaut respectivement, pour la méthode des plus proches voisins :

```
## [1] 4303.568
```

et pour la méthode optimale :

```
## [1] 3793.06
```

On remarque que le résultat de l'algorithme est peu performant, avec de incohérences que l'on peut facilement voir sur la carte, et qui sont confirmées par le total de la longueur des chemins.

Cela illustre l'intérêt de notre démarche : il faut pouvoir objectifier la performance des algorithmes pour les comparer entre eux, et sélectionner le meilleur (selon certains critères).

C'est l'objet des parties suivantes.

2. Comparaison d'algorithmes

On fixe le nombre de sommets du graphe à 10. Les coordonnées cartésiennes suivent des lois uniformes sur [0,1] :

```
set.seed(35)
n <- 10 #nombre de noeuds

#exemple de lancement unitaire
sommets <- data.frame(x = runif(n), y = runif(n))
couts <- distance(sommets)
TSPsolve(dist,'nearest') #LE RESULTAT ICI ME PARAIT BIZARRE (4000 alors que ya que des coordonnées entr
```

```
## [1] 4303.568
```

On lance 50 simulations par algorithme, ce qui donne le résultat suivant (premières lignes uniquement) :

```
##       insertion repet_nn  two_opt  nearest   branch
##  [1,]  2.633229 2.633229 2.794858 3.115361 2.633229
##  [2,]  2.825005 2.825005 2.825005 2.825005 2.825005
##  [3,]  2.770953 2.772088 2.686786 2.772088 2.686786
##  [4,]  2.190799 2.278822 2.383056 2.340315 2.190799
##  [5,]  2.912380 2.912380 3.273722 3.190969 2.912380
##  [6,]  2.595074 2.595074 2.595074 2.655550 2.595074
##  [7,]  2.646389 2.645636 2.645636 2.646389 2.645636
##  [8,]  2.745412 2.858674 2.858674 2.858674 2.745412
##  [9,]  2.835688 2.923733 3.376079 3.859938 2.835688
## [10,]  3.373221 3.207543 3.862590 3.207543 3.176170
## [11,]  3.124444 3.124444 3.616421 3.124444 3.124444
## [12,]  2.890743 2.773278 2.773278 2.773278 2.773278
## [13,]  2.848830 2.844378 3.119311 2.844378 2.844378
## [14,]  2.519599 2.519599 2.728469 2.704847 2.519599
## [15,]  3.184302 3.461219 3.184302 3.677381 3.052222
## [16,]  2.884755 2.518767 2.518767 3.205048 2.518767
## [17,]  2.850912 3.068675 3.022136 3.633584 2.850912
## [18,]  2.354301 2.561375 2.452921 2.952219 2.354301
## [19,]  3.061982 3.047227 3.047227 3.047227 3.047227
## [20,]  2.740308 2.926457 2.954929 3.176949 2.721728
## [21,]  3.564941 3.653052 4.192473 3.653052 3.372445
## [22,]  3.781262 3.482438 3.631811 3.669414 3.429176
## [23,]  2.657181 2.556435 3.224014 2.974170 2.556435
## [24,]  2.461914 2.461914 2.589318 2.830472 1.577505
## [25,]  2.744475 2.744475 2.955204 2.744475 2.744475
## [26,]  2.803303 2.844387 3.003194 2.861528 2.803303
## [27,]  3.364041 3.364041 4.145997 3.469081 3.364041
## [28,]  2.979277 3.002270 3.472056 3.002270 2.979277
## [29,]  3.262933 3.175538 3.314849 3.428114 3.175538
## [30,]  2.591442 2.689641 2.696248 2.782370 2.579881
## [31,]  3.230837 3.183502 3.195253 3.545229 3.080201
## [32,]  2.709654 2.874452 2.730658 3.313752 2.709654
## [33,]  2.680623 2.680623 2.680623 2.680623 2.680623
## [34,]  3.326839 3.326839 3.898505 3.337159 3.326839
## [35,]  2.580709 2.668735 2.855464 2.857004 2.580709
```

```
## [36,]  2.930667 3.103744 2.930667 3.103744 2.930667
## [37,]  2.647981 2.647981 3.472386 2.647981 2.647981
## [38,]  3.132980 3.151552 3.151552 3.568476 3.064892
## [39,]  3.265676 3.048137 3.048137 4.035719 3.048137
## [40,]  3.287300 3.287300 3.287300 3.341229 3.188474
## [41,]  3.117638 3.216568 3.266287 3.261300 3.090471
## [42,]  2.831934 2.817945 2.817945 3.141993 2.817945
## [43,]  3.547758 3.196025 3.196025 3.413817 3.196025
## [44,]  2.894102 2.956698 2.956201 3.376576 2.894102
## [45,]  2.491742 2.658104 2.620072 2.841973 2.478914
## [46,]  2.359488 2.533346 3.180524 2.533346 2.359488
## [47,]  3.553958 3.374373 3.749130 3.611851 3.374373
## [48,]  2.390917 2.540526 2.975510 2.691639 2.390917
## [49,]  3.110281 3.334878 3.334878 3.800788 3.110281
## [50,]  2.575929 2.746920 2.575929 2.821483 1.960579
```

2.1. Longueur des chemins

On s'intéresse ici à la longueur des chemins retournée par chacune des méthodes. Notre but est de détermnier quel algorithme est le plus performant selon ce critère, c'est-à-dire renvoie une longueur la plus faible possible.

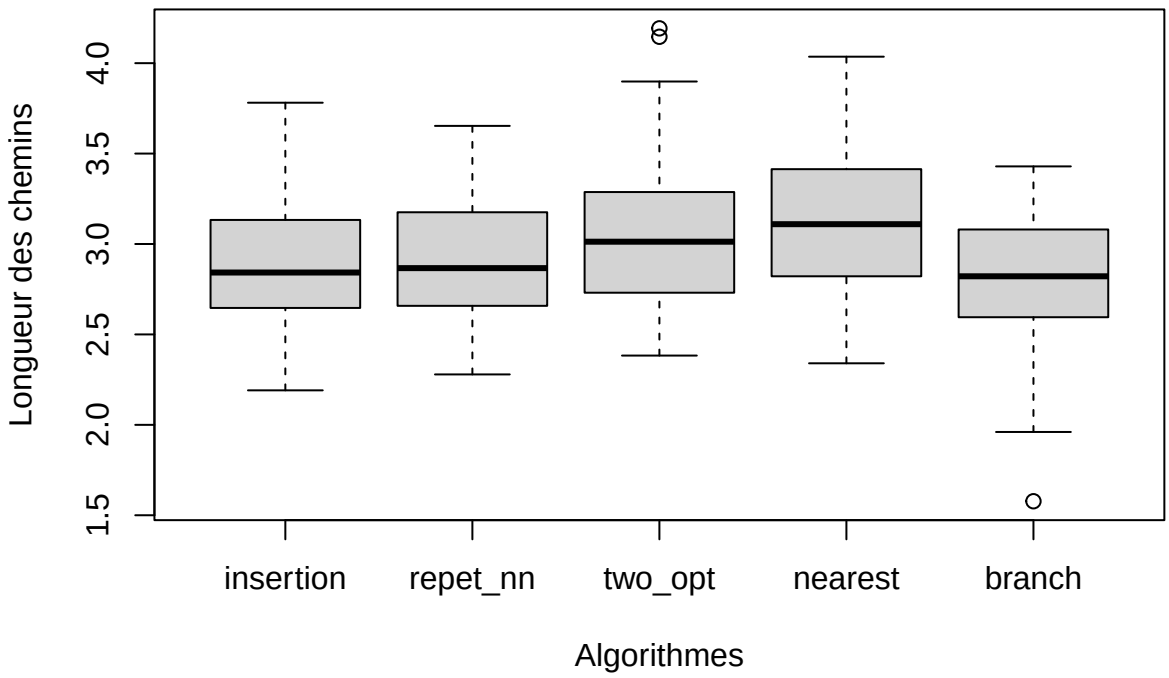
Pour répondre à cette question, on visualisera d'abord les résultats, avant de mener des études plus approfondies, d'abord sur les algorithmes "nearest" et "branch & bound", puis entre tous les algortihmes 2 à 2.

Observation des résultats

Voici d'abord une visualisation des résultats sous forme de boxplot :

```
set.seed(35)
res2 <- as.vector(res)
meth_names <- c('insertion','repetitive_nn','two_opt','nearest','branch')
methods2 <- rep(meth_names,each=nsimu)

boxplot(res,
        main="Résultats des algorithmes de calcul de circuit hamiltonien",
        xlab="Algorithmes",
        ylab="Longueur des chemins")
```



On observe des différences entre les résultats des algorithmes. Celles-ci sont a priori assez faibles, puisque les espaces interquartiles se chevauchent tous (cela est sûrement du aux petits écrats entre les coordonnées). La moyenne de la longueur des chemins est comprise entre 2.8 et 3.0 pour tous.

Il semble tout de même que "insertion", "repet_nn" et "branch" aient des résultats très similaires et bas, alors que "two_opt" et "nearest" se démarquent du reste, avec des résultats plus élevés.

Au sujet de la variance, on observe que "repet_nn" semble avoir une valeur faible, avec des résultats des 1er et 9e déciles plus proche de la moyenne que pour les autres algorithmes. "nearest", "insertion,"two_opt" au contraire paraissent avoir une variance plus élevée. Concernant "branch", la dispersion des valeurs semble surtout être forte pour les valeurs faibles : le 9e décile est assez peu supérieur à la moyenne, alors que le 1er décile y est très inférieur.

Test de normalité

Afin de vérifier si l'on peut faire des tests de Student avec les résultats obtenus, nous vérifions qu'il satisfont tous l'hypothèse de normalité avec un test de Shapiro :

```
shapiro.test(res[,1])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 1]
## W = 0.97675, p-value = 0.4246
```

```
shapiro.test(res[,2])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 2]
## W = 0.97601, p-value = 0.3983
```

```
shapiro.test(res[,3])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 3]
## W = 0.95152, p-value = 0.03939
```

```
shapiro.test(res[,4])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 4]
## W = 0.96797, p-value = 0.1911
```

```
shapiro.test(res[,5])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  res[, 5]
## W = 0.95604, p-value = 0.06067
```

Avec un seuil de risque à 5%, on ne rejette pas l'hypothèse de normalité pour toutes les distributions sauf "two_opt". Pour cette dernière, on considère malgré tout qu'on peut la considérer comme une distribution normale, car la p-valeur reste proche de 5%. On peut donc faire des tests de Student sur ces résultats.

Comparaison de "nearest" et de "branch & bound"

Notre premier objectif est de comparer les performances de "nearest" et de "branch & bound". En effet, ce sont les algorithmes qui ont a priori le plus de chances d'être significativement différents (cf boxplot).

On cherche à savoir si "nearest" est significativement moins performant que "branch & bound", donc on pose les hypothèse suivantes $H(0) : m_n - m_b <= 0$ et $H(1) : m_n - m_b > 0$. De cette manière, on contrôle le risque de conclure que "nearest" est meilleur que "branch & bound" alors que ce n'est pas le cas. A noter que, ici, l'algorithme le plus performant trouve une longueur plus faible.

Pour ce test, on prend un seuil de risque de 5%.

Précisons également que nous faisons la soustraction $m_n - m_b$ car nous avons ici des calculs faits sur le même échantillon, et qu'elle nous permet de limiter les incertitudes : en effet, on construit une troisième gaussienne (la différence des deux premières), que l'on compare à une valeur constante, plutôt que de comparer deux gaussiennes entre elles.

```r
nearest_branch <- res[,4] - res[,5]
t.test(res[,4], res[,5], alternative = "greater")
```

```
##
##  Welch Two Sample t-test
##
## data:  res[, 4] and res[, 5]
## t = 4.0462, df = 97.621, p-value = 5.211e-05
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  0.1814195       Inf
## sample estimates:
## mean of x mean of y
##  3.119036  2.811328
```

Avec une p-valeur de 0.005%, on conclut qu'on rejette l'hypothèse nulle, et que m_n est significativement moins performant que m_b. De plus, la très faible p-valeur nous donne une grande confiance dans ce résultat.

Tests deux à deux

L'objectif de cette analyse est de déterminer si l'on peut regrouper les différents algorithmes par performance sur le critère de longueur des chemins trouvés. Cela permettra de présenter les résultats sous une forme plus intelligible, et éventuellement de faire plus tard des choix entre les algorithmes pus facilement (en prenant d'autres critères comme le temps d'exécution par exemple).

Ici, on teste $H(0) : m_i = m_j$ contre $H(1) : m_i ! = m_j$ pour tous les algorithmes 2 à 2. On prend un seuil de risque de 5%.

L'ajustement du seuil de risque pour les test multiples se fait avec la méthode Bonferroni.

```r
result <- pairwise.t.test(res2,methods2, p.adjust.method = "bonferroni")
result
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  res2 and methods2
##
##              branch  insertion nearest repetitive_nn
## insertion    1.00000 -         -       -
## nearest      0.00052 0.03281   -       -
## repetitive_nn 1.00000 1.00000   0.07124 -
## two_opt      0.00441 0.16646   1.00000 0.32098
##
## P value adjustment method: bonferroni
```

Les deux seuls résultats significatifs sont obtenus lorsqu'on fait le test sur "branch & bound" et "nearest"/"two_opt". Les p-valeurs de respectivement 0.05% et 0.4% nous permettent de rejetter l'hypothèse d'agalité des résultats entre les algorithmes avec confiance.

Pour le reste des tests, aucune différence significative n'émerge. Cela nous pose a priori un problème : on cherche à séparer les algorithmes en différentes classes, selon les différences significatives de résultat. Cependant, étant donné que les test multiples nous donnent la p-valeur de chaque test, on peut se satisfaire

de faire le classement en comparant les p-valeur. On cherche spécialement à faire des classes telles que les algorithmes au sein de chaque classe aient des résultats similaires, mais qu'ils aient des résultat différents de ceux des algorithmes des autres classes.

On décide donc de regrouper "branch & bound", "repetitive_nn", et "insertion" dans la même classe : p-valeurs normalisées de 100% lors des tests entre ces trois algorithmes, et de maximum 32% lors des tests avec les deux autres algorithmes. Pour les mêmes raisons, on regroupe "nearest" et "two_opt" dans une seconde classe.

Pour lever le problème que pose l'absence de p-valeurs significativement faibles lors de ces tests, et ainsi améliorer la confiance que l'on a dans ce classement, on pourrait imaginer faire plus de tests, ou bien prendre des graphes avec des coordonnées plus éloignées.

2.2. Temps de calcul

Après avoir étudié les performances des algorithmes selon la longueur des chemins trouvés, nous étudions les performances en termes de temps de calcul.

On exécute les algorithmes 20 fois, les coordonnées des points sont comme précédemment générés par une lois uniforme sur [0,1].

On utilise pour cela le package "microbenchmark" :

```
set.seed(35)

microbenchmark::microbenchmark(TSPsolve(jeuDeDonnees, method=methods[1]),TSPsolve(jeuDeDonnees, method=m
```

```
## Unit: microseconds
##                                         expr      min        lq       mean
##  TSPsolve(jeuDeDonnees, method = methods[1])  343.252  350.5245  368.21110
##  TSPsolve(jeuDeDonnees, method = methods[2]) 3264.960 3385.9320 3879.63435
##  TSPsolve(jeuDeDonnees, method = methods[3])  326.563  331.0955  367.82350
##  TSPsolve(jeuDeDonnees, method = methods[4])    8.518    8.9270   10.71175
##  TSPsolve(jeuDeDonnees, method = methods[5])  681.702 1541.6830 2401.54945
##     median        uq      max neval cld
##   358.1980  371.5895  492.250    20 a
##  3889.0100 4208.9975 4988.316    20  b
##   336.5640  373.7915  512.719    20 a
##     9.7765   12.4405   15.883    20 a
##  1962.4595 3350.9700 5708.584    20   c
```

Les lignes correspondent dans l'ordre à : "insertion", "repet_nn", "two_opt", "nearest" et "branch".

Les résultat permettent de séparer les algorithmes en 2 classes :

- "insertion", "two_opt" et "nearest" : exécution plus rapide, avec une moyenne comprise entre 7 et 260 microsecondes
- "repet_nn" et "branch & bound" : exécution plus lente, avec une moyenne supérieur à 2200 microsecondes

Si l'on compare avec les résultats de la partie précédente, et la longuer des chemins, on

3. Etude de la complexité de l'algorithme Branch and Bound

Dans cette partie, on étudie la complexité de Branch & Bound, en étudiant le temps d'exécution en fonction de la taille du graphe.

3.1. Comportement par rapport au nombre de sommets : premier modèle

On crée à chaque fois 10 graphes, pour des valeurs de n entre 4 et 20 inclus. Comme précédemment, on génèère les coordonnées des points avec une loi uniforme sur [0,1]. Nous construisons un modèle de régression linéaire simple du temps d'exécution de Branch&Bound en fonction du nombre de sommets n. Introduisons

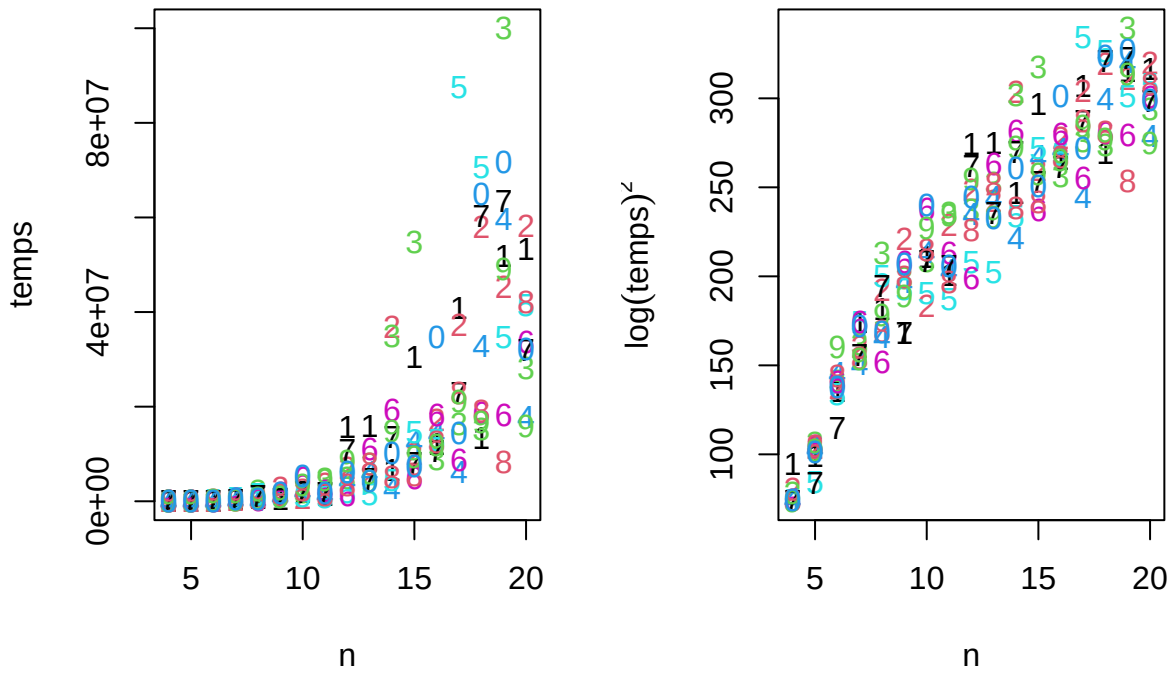
```
set.seed(35)
seqn <- seq(4,20,1)
```

On construit la matrice des résultats :

```
set.seed(35)
temps  <- matrix(nrow = length(seqn), ncol=10)
for (i in 1:17) {
  temps[i,] =
  microbenchmark(TSPsolve(couts, method = 'branch'),
  times = 10,
  setup = { n <- seqn[i]
  couts <- distance(cbind(x = runif(n), y = runif(n))) }
  )$time
}
```

Et on affiche les résultats sur un graphe. Les premiers résultats nous laissant penser à une relations de forme $exp(n/2)$, on affiche aussi $\log(temps)^2$:

```
set.seed(35)
par(mfrow=c(1,2)) # 2 graphiques sur 1 ligne
matplot(seqn, temps, xlab='n', ylab='temps')
matplot(seqn, log(temps)^2, xlab='n', ylab=expression(log(temps)^2))
```



L'observation des résultats nous permet de suspecter une relation linéaire, même si le deuxième graphe montre une relation qui semble légèrement logarithmique.

On crée le modèle de régression linéaire de $\log(temps)^2$ en fonction de n :

```
set.seed(35)
vect_temps <- log(as.vector(temps))^2
vect_dim <- rep(seqn, times=10)
temps.lm <- lm(vect_temps ~ vect_dim)
summary(temps.lm)
```

```
##
## Call:
## lm(formula = vect_temps ~ vect_dim)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -61.401 -19.426  -0.892  18.693  56.245
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  58.9363     4.9520    11.9   <2e-16 ***
## vect_dim     13.4881     0.3821    35.3   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 24.4 on 168 degrees of freedom
## Multiple R-squared:  0.8812, Adjusted R-squared:  0.8805
## F-statistic:  1246 on 1 and 168 DF,  p-value: < 2.2e-16
```

Test de pertinence

Ici, la p-valeur de l'hypothèse $a = 0$ (ligne vect_dim) est extrêmement inférieure à 5%, ainsi, l'hypothèse $a = 0$ est rejetée avec une grande confiance. Le modèle linéaire est pertinent.

Etude du biais

Ici, la p-valeur de l'hypothèse $b = 0$ (ligne vect_dim) est extrêmement inférieure à 5%, ainsi, l'hypothèse $b = 0$ est rejetée avec une grande confiance : il y a un biais dans la relation.

Etude des résidus

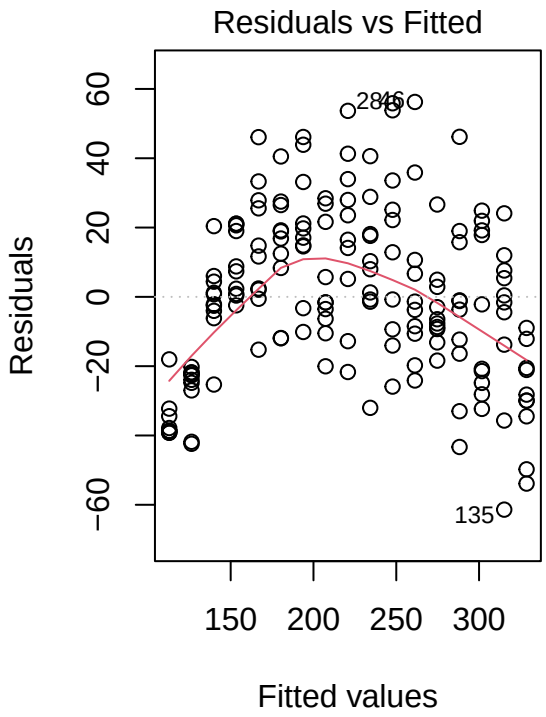
Test de normalité Comme précédemment, le test de normalité est per le test de Shapiro, avec un seuil de risque de 5% :

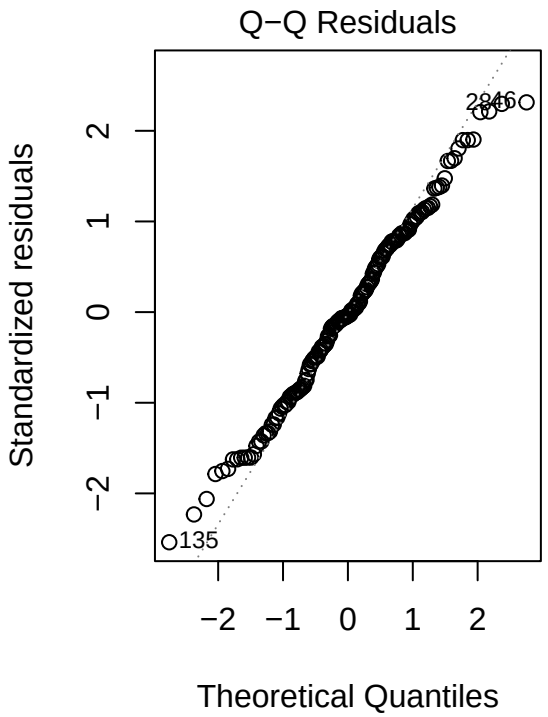
```
shapiro.test(residuals(temps.lm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(temps.lm)
## W = 0.9923, p-value = 0.5025
```

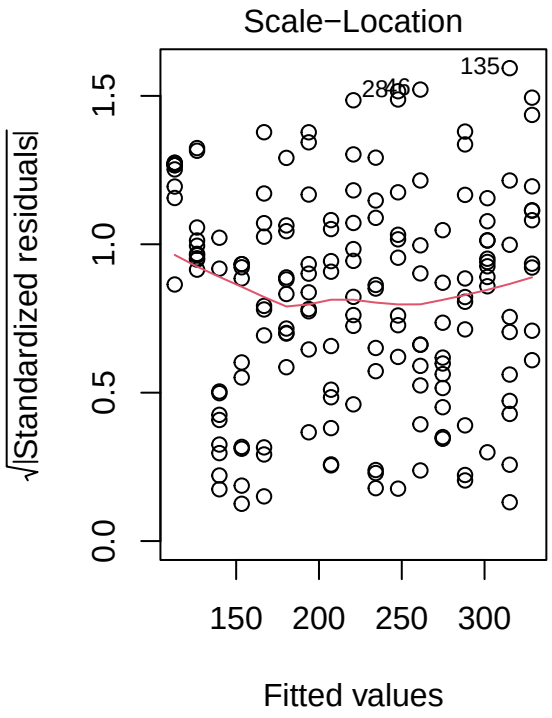
On obtient une p-valeur = 0.37 > 5%. On ne rejette pas l'hypothèse de normalité : les résidus suivent une loi normale.

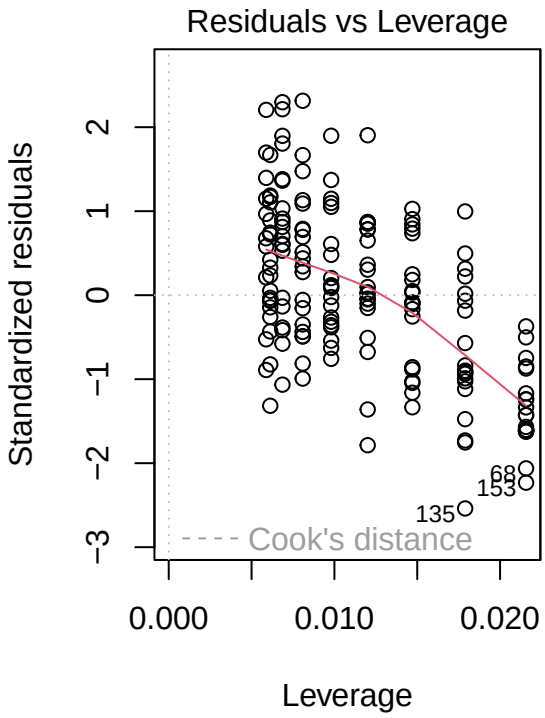
```
par(mfrow=c(1,2)) # 4 graphiques sur 2 lignes et 2 colonnes
plot(temps.lm)
```





Etude graphique





Résiduals vs Fitted : la courbe n'est ni horizontale ni homogène.

Normal Q-Q : l'ensemble des points sont sur la diagonale avec quelques exceptions (points 135 et 160). On peut en déduire que les résidus suivent une loi normale

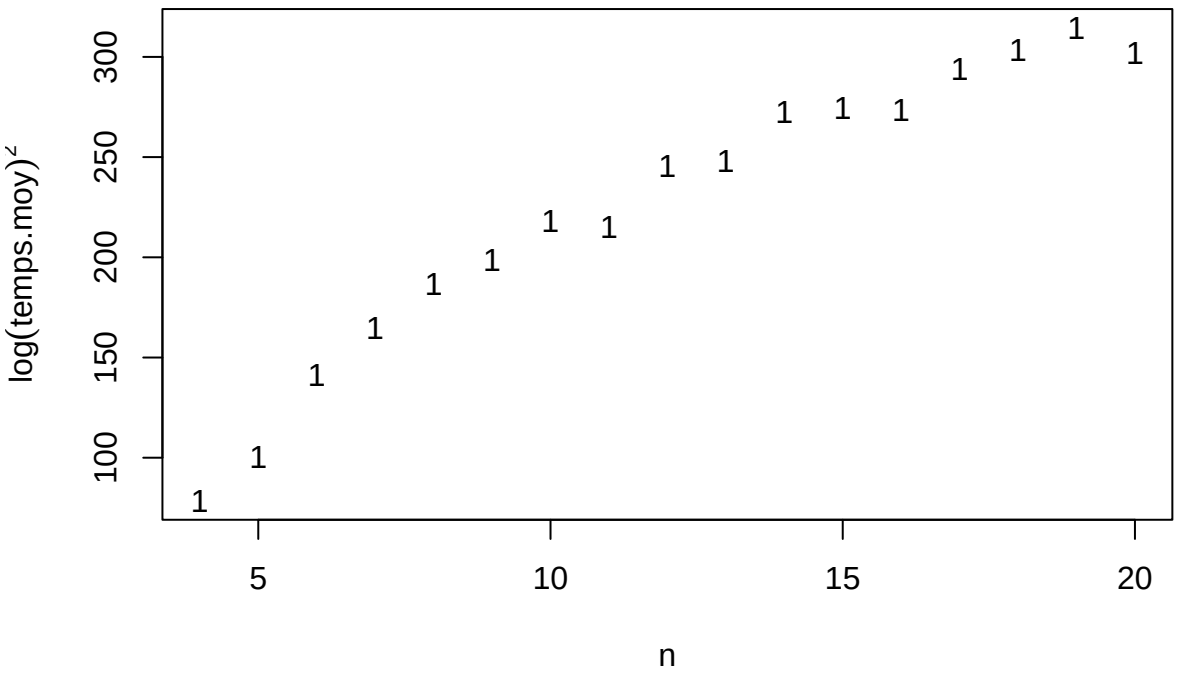
Scale location : courbe moins concave que Résiduals vs Fitted mais toujour pas horizontale

Residuals vs Leverage : les points sont éloignés de la distance de Cook = 1

3.2. Comportement par rapport au nombre de sommets : étude du comportement moyen

Récupération du temps moyen et ajustement du modèle linéaire de $\log(temps.moy)^2$ en fonction de n.

```
temps.moy <- rowMeans(temps)
matplot(seqn, log(temps.moy)^2, xlab='n', ylab=expression(log(temps.moy)^2))
```



Analyse de la validité du modèle :

- pertinence des coefficients et du modèle,

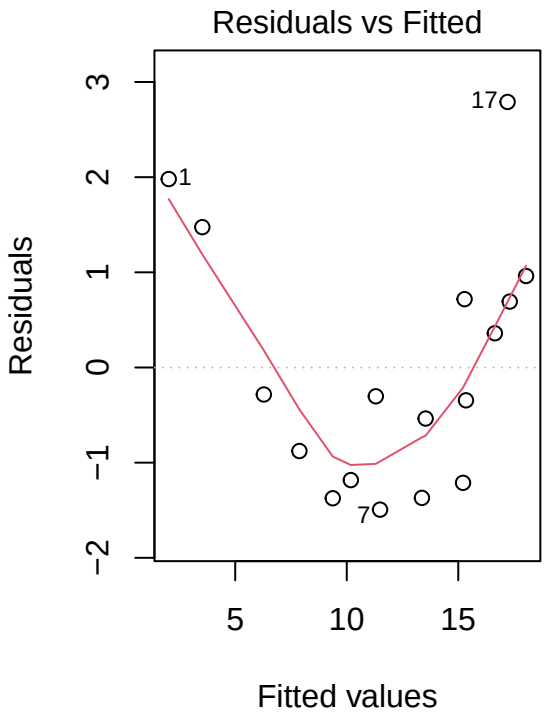
```
vect_moy <- log(as.vector(temps.moy))^2
vect_dim_moy <- rep(seqn)
temps.moy.lm <- lm(vect_dim_moy ~ vect_moy)
summary(temps.moy.lm)
```

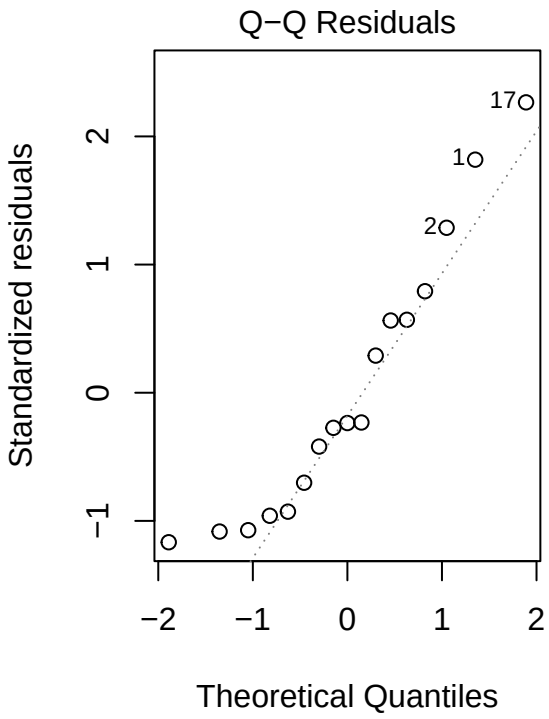
```
##
## Call:
## lm(formula = vect_dim_moy ~ vect_moy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.4935 -1.1830 -0.3027  0.7181  2.7904
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.307741   1.082672  -3.055  0.00802 **
## vect_moy     0.067885   0.004586  14.801 2.34e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.32 on 15 degrees of freedom
## Multiple R-squared:  0.9359, Adjusted R-squared:  0.9316
## F-statistic: 219.1 on 1 and 15 DF,  p-value: 2.344e-10
```

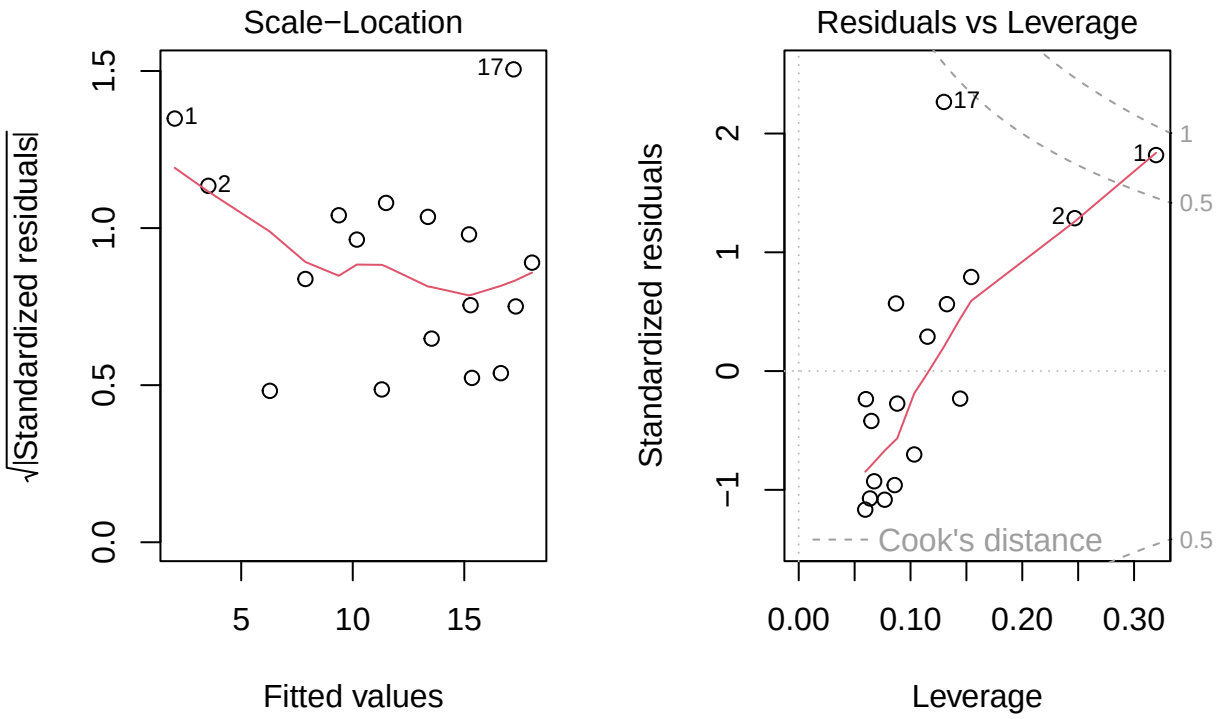
Comme dans le modèle précédent, les p-valeurs des tests d'hypothèse $a = 0$ et $b = 0$ sont inférieurs à 5%. On considère donc que ces hypothèses peuvent être rejetées et qu'un modèle linéaire est pertinent. On peut néanmoins remarquer que la p-valeur est sensiblement plus haute que dans le test précédent avec toutes les valeurs, surtout pour le test sur la nullité du biais.

- étude des hypothèses sur les résidus.

```
par(mfrow=c(1,2)) # 4 graphiques sur 2 lignes et 2 colonnes
plot(temps.moy.lm)
```


Residuals vs Fitted
Residuals
Fitted values
17
1
7
−2
−1
0
1
2
3
5
10
15


Q−Q Residuals
Standardized residuals
Theoretical Quantiles
17
1
2
−2
−1
0
1
2



Résiduals vs Fitted : la courbe n'est ni horizontale ni homogène.

Normal Q-Q : les points suivent globalement la diagonale, sauf les points 1, 7 et 17. On garde confiance dans le fait que les résidus suivent une loi normale

Scale location : courbe moins concave que Résiduals vs Fitted mais toujour pas horizontale

Residuals vs Leverage : les points sont éloignés de la courbe de distance 1, mais un point (n°1) la franchit.

```
shapiro.test(residuals(temps.moy.lm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(temps.moy.lm)
## W = 0.92267, p-value = 0.1638
```

Avec une p-valeur de 26% et un seuil de risque de 5%, on ne rejette pas l'hypothèse de normalité.

En conclusion, on considère que notre modèle de régression est pertinent pour ce modèle. Cependant, on sent bien que le fait d'avoir réduit le nombre de points affaiblit un peu la confiance que l'on a dans ce dernier. Il semble donc qu'il y ait à faire un arbitrage entre confiance dans le modèle et temps de calcul : dans notre cas, ces derniers sont rapides, mais sur d'autres jeu de données (plus volumineux, avec plus de dimensions) on peut imaginer que cette préoccupation devienne importante.

3.3. Comportement par rapport à la structure du graphe

Lecture du fichier 'DonneesTSP.csv'.

```
data.graph <- read.csv(file='DonneesTSP.csv',header=TRUE)
str(data.graph)
```

```
## 'data.frame':    70 obs. of  8 variables:
##  $ tps      : num  53692 144081 997803 2553322 6333009 ...
##  $ dim      : int  4 6 8 10 12 14 16 18 20 4 ...
##  $ mean.long: num  0.391 0.442 0.334 0.276 0.254 ...
##  $ mean.dist: num  0.665 0.592 0.537 0.506 0.502 ...
##  $ sd.dist  : num  0.276 0.259 0.246 0.238 0.227 ...
##  $ mean.deg : num  3 5 7 9 11 13 15 17 19 3 ...
##  $ sd.deg   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ diameter : num  1 1 1 1 1 1 1 1 1 1 ...
```

Ajustement du modèle linéaire de $\log(temps.moy)^2$ en fonction de toutes les variables présentes. Modèle sans constante.

```
data.graph$log.tps <- log(data.graph$tps)#log(donnees$tps)^2
data.graph$sqrt.dim <- sqrt(data.graph$dim)
data.graph$tps <- c() #on retire les variables tps et dim devenues inutiles
data.graph$dim <- c()
str(data.graph)
```

```
## 'data.frame':    70 obs. of  8 variables:
##  $ mean.long: num  0.391 0.442 0.334 0.276 0.254 ...
##  $ mean.dist: num  0.665 0.592 0.537 0.506 0.502 ...
##  $ sd.dist  : num  0.276 0.259 0.246 0.238 0.227 ...
##  $ mean.deg : num  3 5 7 9 11 13 15 17 19 3 ...
##  $ sd.deg   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ diameter : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ log.tps  : num  10.9 11.9 13.8 14.8 15.7 ...
##  $ sqrt.dim : num  2 2.45 2.83 3.16 3.46 ...
```

```
modele.complet = lm(data.graph$log.tps~., data = data.graph)
summary(modele.complet)
```

```
##
## Call:
## lm(formula = data.graph$log.tps ~ ., data = data.graph)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78776 -0.15715  0.01542  0.17260  0.65036
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.4903426  0.5450715  11.907  < 2e-16 ***
## mean.long   -4.8152962  0.7294055  -6.602 1.05e-08 ***
## mean.dist   -0.0020048  0.0010633  -1.886  0.06404 .
## sd.dist      0.0048105  0.0006652   7.231 8.55e-10 ***
## mean.deg    -0.1367369  0.0425459  -3.214  0.00208 **
## sd.deg       0.1399515  0.0872430   1.604  0.11376
```

```
## diameter    -0.0646816  0.1566329  -0.413  0.68107
## sqrt.dim     3.4191719  0.2391476  14.297  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2912 on 62 degrees of freedom
## Multiple R-squared:  0.986,  Adjusted R-squared:  0.9844
## F-statistic: 622.6 on 7 and 62 DF,  p-value: < 2.2e-16
```

Comme précédemment, on s'intéresse à la dernière colonne Pr(>|t|), qui donne la p-valeur du test d'hypothèse $H(0) : var = 0$. On prend ici aussi un seuil de risque de 5%.

Avec ce test, on trouve que les résultats pour "mean.dist", "sd.deg" et "diameter" ne permettent pas de rejeter H0. Il semble donc que ces variables ne soient pas pertinentes pour notre modèle.

Toutes les autres variables : biais, "mean.long", "sd.dist", "mean.deg", et "sqrt.dim" sont pertinentes.

On utilise la fonction step pour faire une sélection de variables :

```
step(modele.complet)
```

```
## Start:  AIC=-165.23
## data.graph$log.tps ~ mean.long + mean.dist + sd.dist + mean.deg +
##     sd.deg + diameter + sqrt.dim
##
##             Df Sum of Sq     RSS      AIC
## - diameter   1    0.0145  5.2711 -167.038
## <none>                    5.2566 -165.230
## - sd.deg     1    0.2182  5.4748 -164.384
## - mean.dist  1    0.3014  5.5581 -163.327
## - mean.deg   1    0.8757  6.1324 -156.444
## - mean.long  1    3.6951  8.9517 -129.965
## - sd.dist    1    4.4335  9.6902 -124.417
## - sqrt.dim   1   17.3311 22.5877  -65.176
##
## Step:  AIC=-167.04
## data.graph$log.tps ~ mean.long + mean.dist + sd.dist + mean.deg +
##     sd.deg + sqrt.dim
##
##             Df Sum of Sq     RSS      AIC
## <none>                    5.2711 -167.038
## - sd.deg     1    0.2065  5.4776 -166.349
## - mean.dist  1    0.6554  5.9265 -160.835
## - mean.deg   1    0.9820  6.2531 -157.080
## - mean.long  1    3.8220  9.0931 -130.869
## - sd.dist    1    4.9133 10.1844 -122.935
## - sqrt.dim   1   18.7788 24.0499  -62.785

##
## Call:
## lm(formula = data.graph$log.tps ~ mean.long + mean.dist + sd.dist +
##     mean.deg + sd.deg + sqrt.dim, data = data.graph)
##
## Coefficients:
```

```
## (Intercept)    mean.long    mean.dist      sd.dist     mean.deg       sd.deg
##    6.396008    -4.854857    -0.002284     0.004883    -0.140823     0.126916
##    sqrt.dim
##    3.444077
```

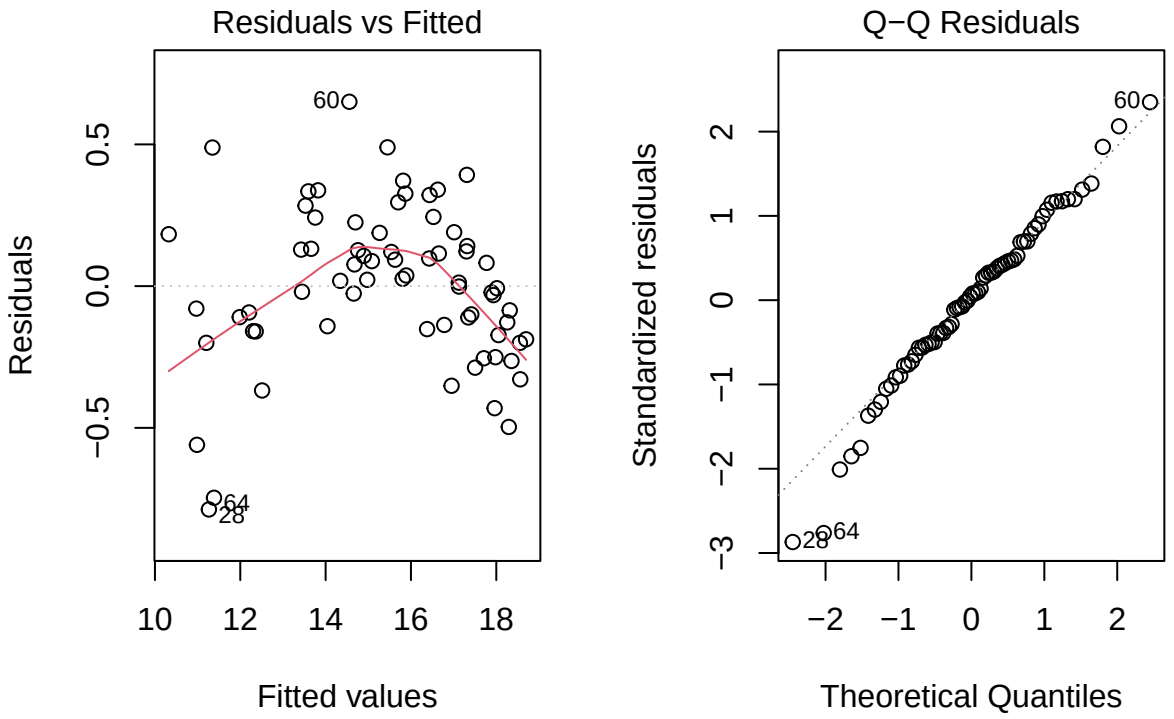
On remarque que seule la variable “diameter” a été exclue du modèle. C’était en effet une variable que nous avions identifiée comme non pertinente dans le modèle, mais les variables “mean.dist” et “sd.deg” l’étaient également, et sont conservées. Cependant, il est vrai que pour ces deux dernières valeurs, la p-valeur était moins élevée que pour “diameter”.

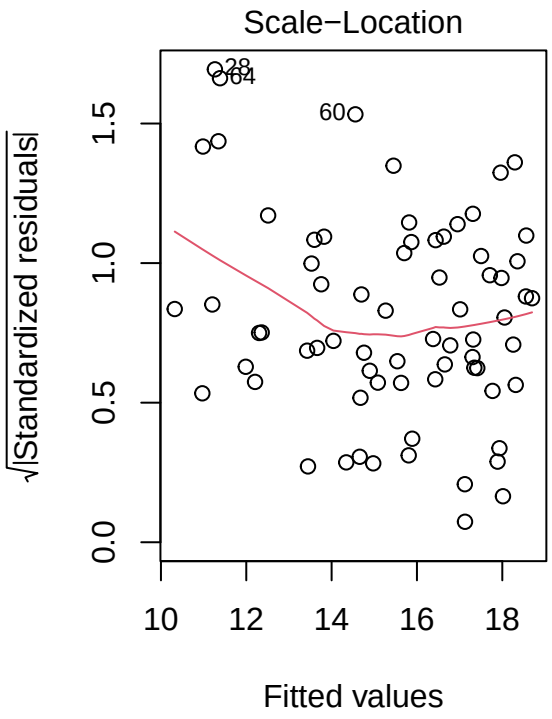
Etant donné que la sélection de variables modifie la valeur des coefficients, et que ces variables étaient “proches” d’être pertinentes, il est possible qu’une valeur modifiée et proche soit pertinente.

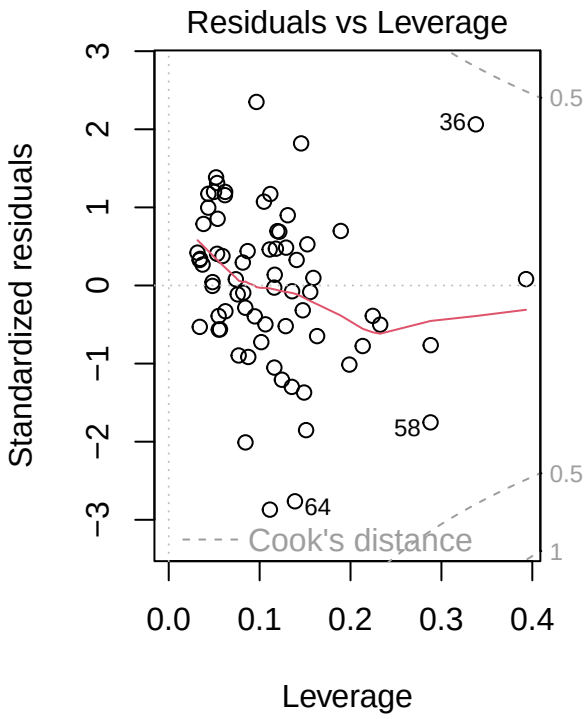
Le test de Fisher donnait une p-valeur $< 2.2e\text{-}16$: le modèle dans son ensemble est validé.

On passe ensuite à l’étude des résidus :

```
par(mfrow=c(1,2)) # 4 graphiques sur 2 lignes et 2 colonnes
plot(modele.complet)
```








```
shapiro.test(residuals(modele.complet))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(modele.complet)
## W = 0.98289, p-value = 0.455
```

Pour les mêmes raisons que précédemment, on considère que les résidus suivent une loi normale.

Cela nous permet de conclure à la validité de notre modèle.

Conclusion

En conclusion, ce TP nous a permis d'utiliser les outils de statistiques vus en cours et en TD. Nous avons pu étudier en détail les tests à faire lorsque l'on veut étudier un phénomène statistique, et l'étude des paramètres de ces tests, pour avoir une confiance maximale dans nos résultats.