TP Statistique

Manal El Karchouni, Roxane Gall, Jean Haberer

01 Avril 2020

0. Visualisation de chemins

Lecture du fichier des villes :

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

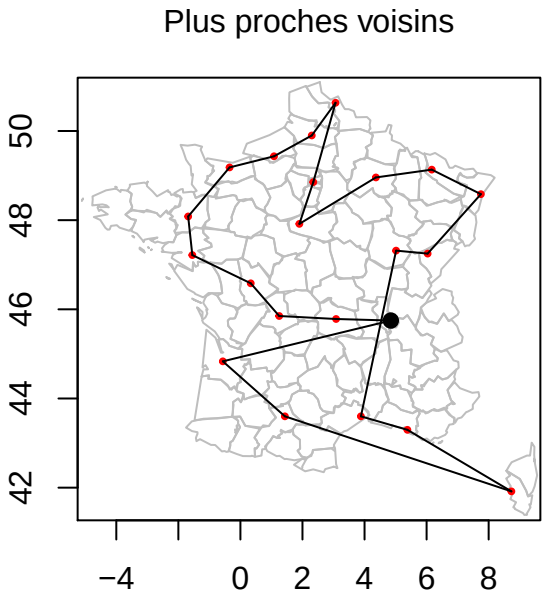
```
## 'data.frame':    22 obs. of  5 variables:
##  $ EU_circo : Factor w/ 7 levels "ÃŽle-de-France",..: 6 6 4 3 7 4 3 2 3 4 ...
##  $ region   : Factor w/ 22 levels "Alsace","Aquitaine",..: 22 10 19 11 2 5 9 3 6 17 ...
##  $ ville    : Factor w/ 22 levels "Ajaccio","Amiens",..: 11 1 2 3 4 5 6 7 8 9 ...
##  $ latitude : num  45.7 41.9 49.9 47.2 44.8 ...
##  $ longitude: num  4.847 8.733 2.3 6.033 -0.567 ...
```

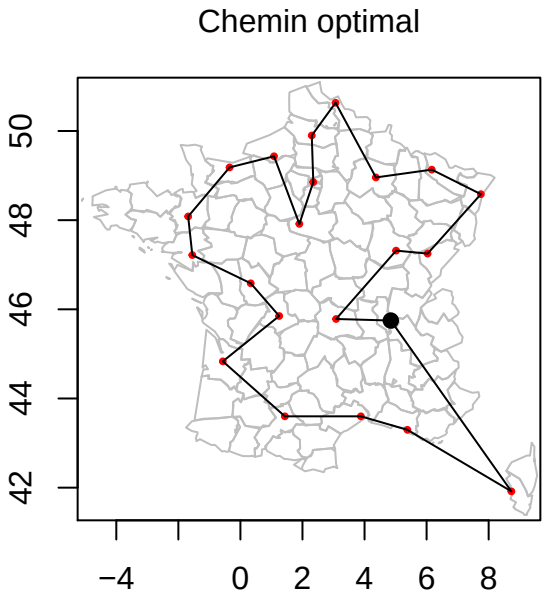
Représentation des chemins par plus proches voisins et du chemin optimal :

```
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='Plus proches voisins')
plotTrace(coord[pathOpt, ], title='Chemin optimal')
```





Les longueurs des trajets (à vol d'oiseau) valent respectivement, pour la méthode des plus proches voisins :

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

et pour la méthode optimale :

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

Ceci illustre bien l'intérêt d'un algorithme de voyageur de commerce. Nous allons dans la suite étudier les performances de cet algorithme.

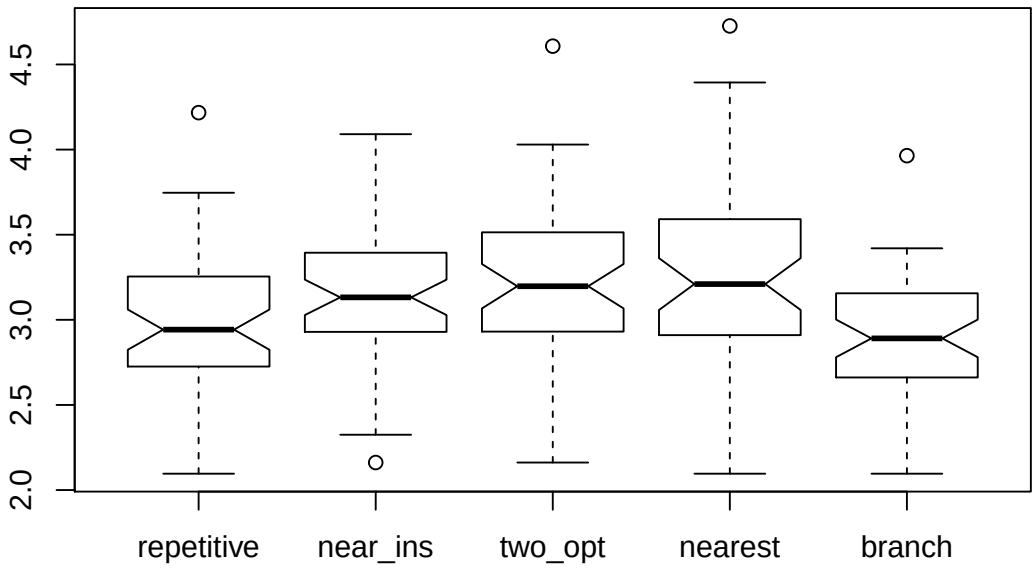
1. Comparaison d'algorithmes

Nombre de sommets fixes et graphes "identiques".

```
 n <- 10
sommets <- data.frame(x = runif(n), y = runif(n))
```

1.1. Longueur des chemins

Comparaison des longueurs de différentes méthodes :



- boxplots

Ces boxplots représentent la distribution des longueurs des chemins hamiltoniens donnés par les 5 méthodes sur 50 réalisations de graphes aléatoires (n=10 sommets dont les coordonnées suivent des lois uniformes sur [0,1]). Les algorithmes représentés sont, de gauche à droite, repetitive, nearest_insertion, two_opt, nearest et branch. En moyenne, on observe que tous ont des médianes qui n'ont pas exactement la même valeur mais sont proches de 3.0. La distribution varie entre les différents algorithmes et est asymétrique pour certains. Par exemple nearest a le plus grand étalement, avec une très légère asymétrie vers les valeurs plus basses. En moyenne, sa valeur maximale est supérieure à 4 et son 3ème quartile vaut environ 3.5. Sa médiane est également légèrement plus élevée que pour les autres algorithmes. Nearest donne en moyenne de plus hautes valeurs de longueur du chemin hamiltonien mais est l'algorithme le plus variable. A l'inverse branch a un étalement beaucoup plus faible et des valeurs globalement plus basses que les autres algorithmes (max, 3ème et 1er quartiles, médiane et min inférieurs). C'est donc l'algorithme le plus performant.

Nous allons désormais comparer les algorithmes Nearest et Branch and Bound à l'aide d'hypothèses paramétriques. Soit m_n et m_b les espérances respectives des algorithmes nearest et de branch and bound. Nous réalisons le test de l'hypothèse nulle $(H_0) -> m_{nn} - m_b <= 0$ (l'espérance de nearest est inférieure à celle de branch and bound) contre son hypothèse alternative $(H_1) -> m_{nn} - m_b > 0$.

- test entre 'nearest' et 'branch'

```
t.test(nearest-branch,NULL,"greater")
```

```
##
##  One Sample t-test
##
## data:  nearest - branch
## t = 8.3472, df = 49, p-value = 2.827e-11
## alternative hypothesis: true mean is greater than 0
```

```
## 95 percent confidence interval:
##  0.3015106       Inf
## sample estimates:
## mean of x
## 0.3772898
```

Le test d'hypothèse renvoie la p-valeur, qui résume les résultats du test. Ici en moyenne, la p-valeur est $\alpha_c = 4,383737 * 10^{-10}$. Cette valeur est en général très faible : cela signifie que l'on peut rejeter l'hypothèse avec un faible risque de se tromper (risque de l'ordre de 10^{-10}). Autrement dit, on peut valider l'hypothèse (H1) indiquant que les espérances des algorithmes des plus proches voisins sont supérieures à celles des algorithmes de Branch and Bound, avec un faible risque d'erreur. En conclusion, l'algorithme de Branch and Bound est plus performant que Nearest. Cette hypothèse est vérifiée sur le box plot précédent.

Maintenant nous réalisons les tests pour comparer 2 à 2 les longueurs moyennes obtenues par les algorithmes. La matrice obtenue donne les p-valeurs du test de l'hypothèse nulle $(H_0) m_i = m_j$ contre l'hypothèse alternative $(H_1) m_i \neq m_j$ avec $i \neq j$.

Nous obtenons la matrice résultat ci-dessous :

- tests 2 à 2

```
results<-as.vector(mat)
methods<-rep(colnames(mat), each=50)
pairwise.t.test(results, methods, p.adjust.method='bonferroni')
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  results and methods
##
##            branch  near_ins nearest repetitive
## near_ins   0.03854 -        -       -
## nearest    0.00022 1.00000  -       -
## repetitive 1.00000 0.88997  0.02034 -
## two_opt    0.00478 1.00000  1.00000 0.20621
##
## P value adjustment method: bonferroni
```

Différentes observations sont à noter sur ces résultats :

La p-valeur pour les couples (branch,repetitive), (near_insertion,nearest), (near_insertion,repetitive), (near_insertion,two_opt) et (nearest,two_opt) est en moyenne élevée. On ne peut donc pas rejeter l'hypothèse H_0. Cela signifie que les résultats des algorithmes sont similaires. La p-valeur pour (branch,near_insertion), (branch,two_opt), (nearest, repetitive) et (repetitive,two_opt) est généralement plus faible. On peut donc rejeter l'hypothèse H_0 avec un plus faible risque de se tromper (plus α_c est petit, plus le risque est faible) : nous obtenons en moyenne 5,07% pour branch et two_opt, 47,98% pour branch et near_insertion etc.. Cela signifie que leurs résultats sont différents. Les résultats dont on peut affirmer qu'ils sont les plus éloignés, avec la p-valeur la plus faible en moyenne $\alpha_c = 0.0074$ et donc un risque de se tromper inférieur à 1 %, sont ceux entre branch et nearest. Cela rejoint l'hypothèse de la question précédente et le box plot initial qui suggérait que branch était plus performant que nearest.

1.2. Temps de calcul

Comparaison des temps à l'aide du package microbenchmark.

Finalement nous allons effectuer une comparaison des temps d'exécution à l'aide du package microbenchmark afin de déterminer l'algorithme le plus rapide.

Exemple d'application de microbenchmark :

```
microbenchmark<-microbenchmark(TSPsolve(couts,'repetitive_nn')
               ,TSPsolve(couts,'branch')
               ,TSPsolve(couts,'nearest')
               ,TSPsolve(couts,'nearest_insertion')
               ,TSPsolve(couts,'two_opt')
               ,times=20, setup={  sommets <- data.frame(x = runif(10), y = runif(10))
               couts <- distance(sommets)})
summary(microbenchmark)
```

```
##                                   expr    min      lq      mean   median
## 1     TSPsolve(couts, "repetitive_nn") 7694.4 9312.70 13359.555 11178.15
## 2            TSPsolve(couts, "branch") 2228.5 3787.55  7598.195  7247.50
## 3           TSPsolve(couts, "nearest")   20.5   27.65    31.095    30.85
## 4 TSPsolve(couts, "nearest_insertion")  965.3 1284.95  2723.655  1636.70
## 5           TSPsolve(couts, "two_opt")  654.1  828.05  1015.355  1031.35
##         uq     max neval cld
## 1 13705.45 37035.9    20   c
## 2 10682.80 17825.1    20  b
## 3    33.80    48.2    20 a
## 4  2213.75 17064.7    20 a
## 5  1197.05  1376.6    20 a
```

Nous observons ici les résultats de la fonction microbenchmark pour comparer les temps des 5 méthodes étudiées sur 20 graphes. D'après les moyennes des différents temps d'exécution, le classement des algorithmes du plus long au plus rapide en moyenne est : repetitive, branch, nearest, nearest_insertion et two_opt. Le plus long algorithme à s'exécuter à tout point de vue est indéniablement repetitive (son min est supérieur à toutes les moyennes des autres algorithmes et à 4 des 5 valeurs max). A l'inverse le plus rapide est nearest (son max est inférieur à toutes les valeurs min des autres algorithmes). Cependant on ne peut pas analyser la rapidité des algorithmes uniquement sur leur moyenne. Par exemple, l'algorithme two_opt a une moyenne plus faible que celle de nearest_insertion mais une valeur max plus grande. Cela signifie que pour certains cas two_opt peut être plus lent que nearest_insertion. En tenant compte des différents critères et des ordres de grandeurs les algorithmes sont classés en 3 catégories : **a** de l'ordre des dizaines et centaines de secondes pour nearest, nearest_insertion et two_opt, **b** de l'ordre de milliers de secondes pour branch et **c** de l'ordre des dizaines de milliers de secondes pour repetitive. Une autre observation intéressante peut être de regarder la répartition des valeurs c'est-à-dire la différence entre le 1er et 3ème quartile ou entre les valeurs extrêmes (min et max). Par exemple nous aurions pu présenter un box plot pour mieux les visualiser. L'algorithme Branch a régulièrement une très grande dispersion de ses valeurs : en moyenne nous obtenons un min d'environ 1959s et un max à environ 21 135s avec une médiane de 5558s. Cette asymétrie vers de grandes valeurs explique que sa moyenne soit plus élevée que sa médiane et cela signifie que le temps d'exécution est variable et va exploser dans certains cas. Cela s'expliquerait sûrement par sa complexité exponentielle en temps. Il aurait été intéressant de comparer l'efficacité des algorithmes selon la taille des graphes analysés.

2. Etude e la complexité de l'algorithme Branch and Bound

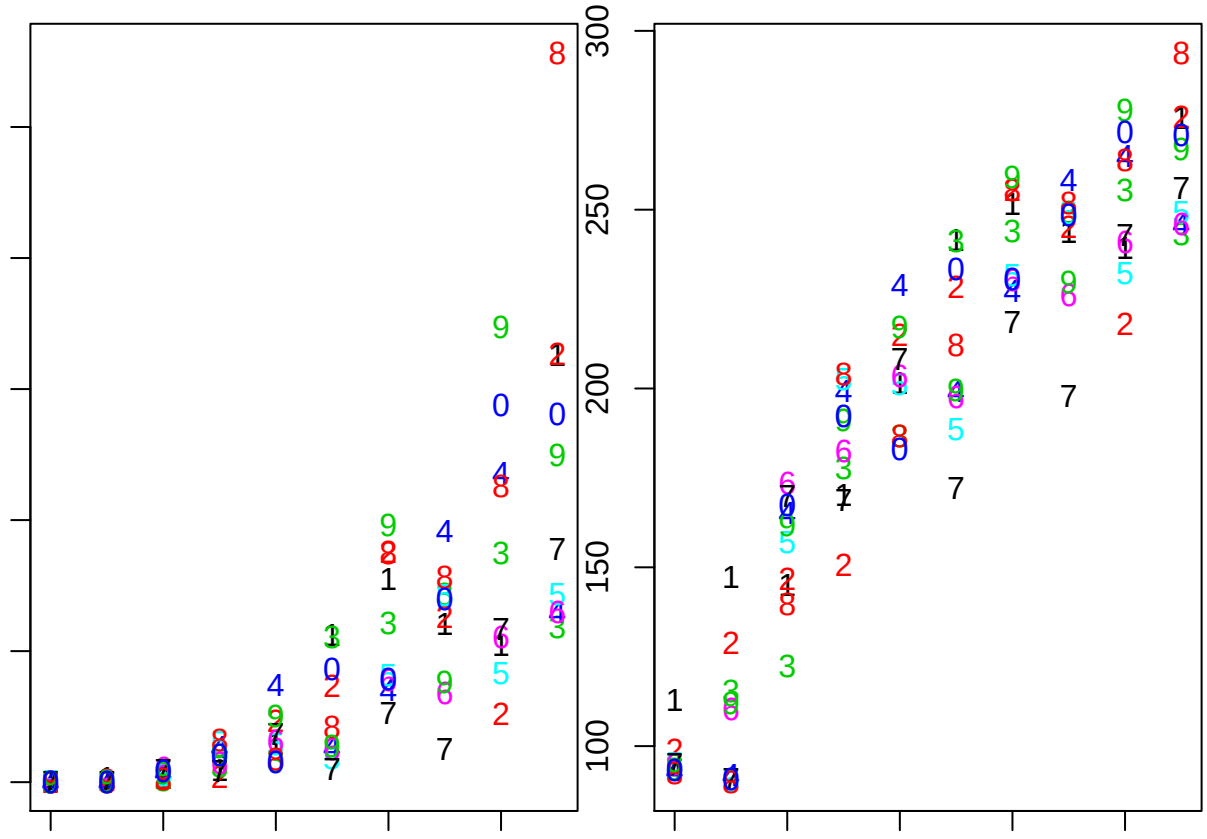
2.1. Comportement par rapport au nombre de sommets : premier modèle

Récupération du temps sur 10 graphes pour différentes valeurs de n.

```
seqn <- seq(4, 13, 1)
temps<-matrix(0, length(seqn), 10)
for(i in 1:length(seqn))
temps[i, ]<-microbenchmark(TSPsolve(couts, method = 'branch'),times = 10,
                           setup = { n <- seqn[i]
                                 couts <- distance(cbind(x = runif(n), y = runif(n)))})$time
```

Visualisation de temps en fonction de n puis de $\log(temps)^2$ en fonction de n :

```
par(mfrow=c(1,2)) # 2 graphiques sur 1 ligne
par(mar=c(1,1,1,1))
matplot(seqn, temps, xlab='seqn', ylab='temps')
matplot(seqn, log(temps)^2, xlab='seqn', ylab='expression(log(temps)^2)')
```



Nous remarquons que l'évolution du temps en fonction de n ne se comporte pas de manière linéaire. En revanche après le changement de variable on peut bien appliquer un modèle de régression linéaire pour $\log(temps)^2$ en fonction de n.

Ajustement du modèle linéaire de $\log(temps)^2$ en fonction de n.

```
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
## -44.280 -15.893   1.441  16.542  43.366
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  35.1203     6.8743   5.109  1.6e-06 ***
## vect_dim     18.8142     0.7662  24.556  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.01 on 98 degrees of freedom
## Multiple R-squared:  0.8602, Adjusted R-squared:  0.8588
## F-statistic:   603 on 1 and 98 DF,  p-value: < 2.2e-16
```

Analyse de la validité du modèle :

Pertinence des coefficients et du modèle,

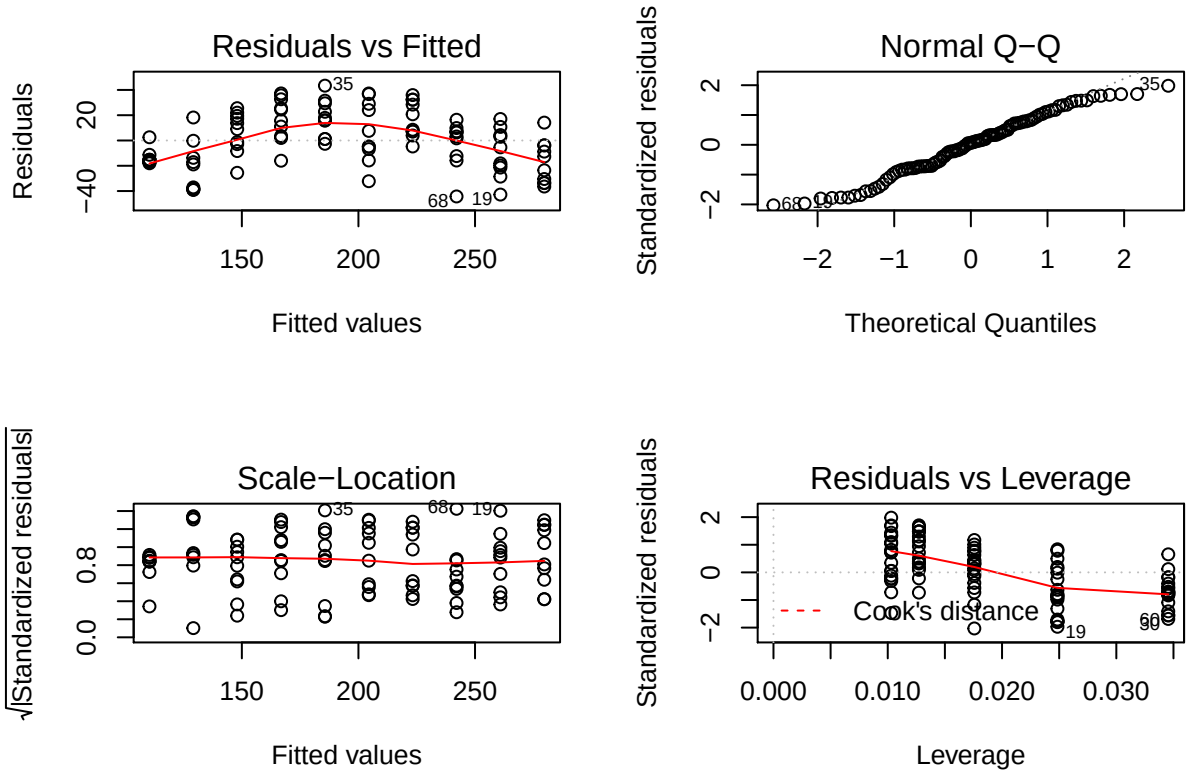
À l'aide des informations décrites par R, nous remarquons que: Les coefficients ont un taux d'erreur important allant en moyenne jusqu'à 7.3277 pour l'ordonnée à l'origine et 0.8167 pour le coefficient directeur, nous ne pouvons donc pas affirmer la pertinence des coefficients du modèle linéaire. La p-valeur de chaque coefficient quand à elle est très petite, inférieure à $2.2e^{-16}$ durant chacune de nos exécution pour le coefficient directeur et égale en moyenne à 0.0805% pour l'ordonnée à l'origine.

En conclusion, au vu du taux d'erreur important, il faut rejeter l'hypothèse selon laquelle $\log(temps)^2$ d'exécution de l'algorithme Branch and Bound suit un modèle linéaire.

Étude des hypothèses sur les résidus.

Les hypothèses sur les résidus à étudier sont les suivantes: +Loi normale +Espérance nulle +Variance constante +Indépendance Les graphiques permettant de valider ou non ces dernières sont les suivants :

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



Au vue des résultats, nous pouvons remarquer que: - Les graphiques Residuals vs Fitted et Scale-Location montrent de nombreux points très écartés du modèle, la variance des résidus n'est donc pas constante. Les résidus sont dit hétéroscédastiques. - Le graphique Normal Q-Q semble être aligné avec la droite, plus au centre qu'au niveau des extrémités. Nous aurions tendance à affirmer que l'hypothèse de Loi normale est potentiellement non réfutable (à confirmer avec d'autres tests cf test de Shapiro-Wilk) - Le graphique Residuals vs Leverage montre l'influence des échantillons. Nous remarquons l'existence d'un grand nombre d'outliers, par conséquent beaucoup de points ne contribuent pas à la construction du modèle linéaire.

Au vu de la non-validation de quelques unes de ces 4 hypothèses, le modèle n'est pas vérifié. Observons si les résidus suivent tout de même une loi normale.

Pour vérifier si les résidus suivent une loi normale, nous effectuons le test de Shapiro-Wilk :

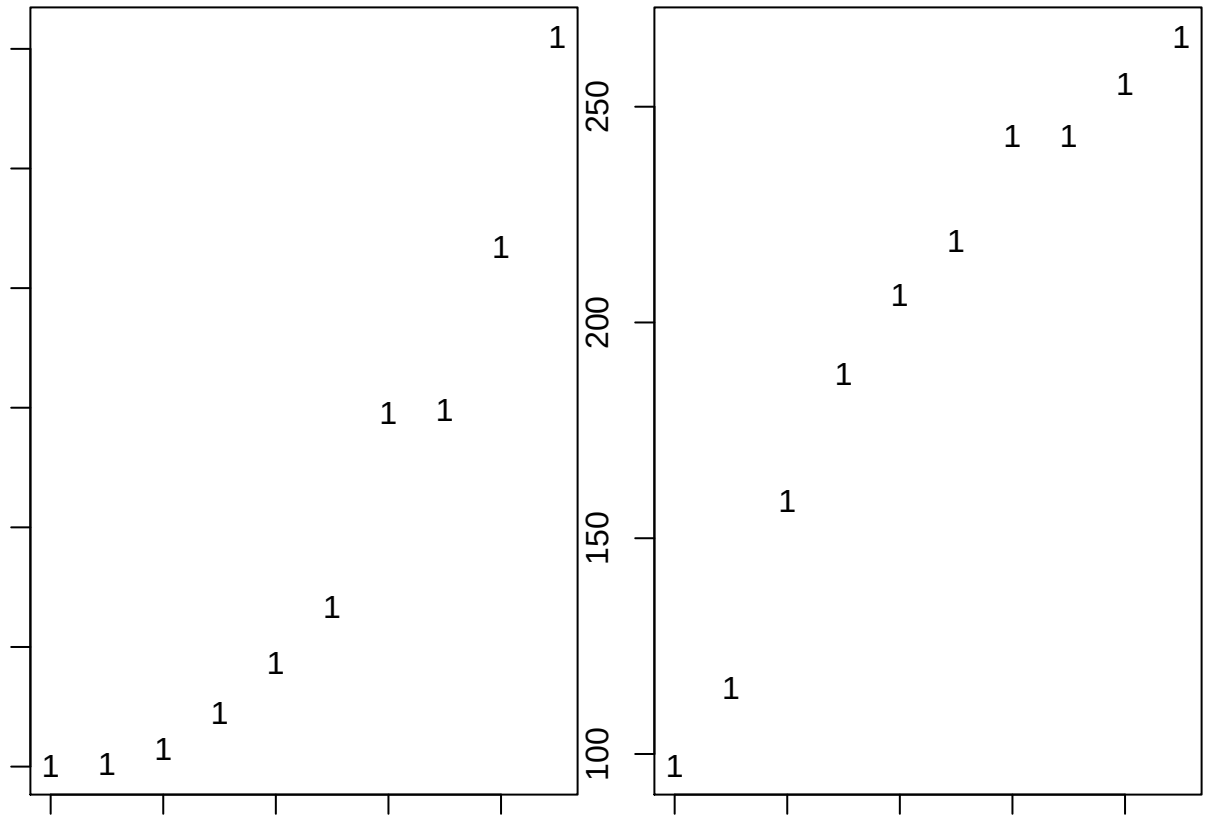
```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(temps.lm)
## W = 0.97753, p-value = 0.08524
```

Nous obtenons une p-valeur autour de 0.0004999 très faible (inférieur à 0, 1 %). Cela signifie que le modèle est cohérent et que les résidus suivent bien une loi normale.

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

Récupération du temps moyen.

```
temps.moy <- rowMeans(temps)
par(mfrow=c(1, 2)) # 2 graphiques sur 1 ligne
par(mar=c(1, 1, 1, 1))
matplot(seqn, temps.moy, xlab='seqn', ylab='temps_moy')
matplot(seqn, log(temps.moy)^2, xlab='seqn', ylab='expression(log(temps_moy)^2)')
```



La courbe *temps_moy* en fonction de n (graphe de gauche) n'a pas une forme linéaire. Nous modélisons donc, de même que dans la partie précédente, la courbe $\log(temps.moy)^2$ en fonction de n auquel on peut appliquer un modèle linéaire (graphe de droite).

Ajustement du modèle linéaire de $\log(temps.moy)^2$ en fonction de n.

```
vect_temps_moy <- log(as.vector(temps.moy))^2
temps.lm.moy <- lm(vect_temps_moy ~ seqn)
```

Analyse de la validité du modèle

```
summary(temps.lm.moy)
```

```
##
## Call:
## lm(formula = vect_temps_moy ~ seqn)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -18.075 -15.476   1.691  14.345  16.902
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   39.541     15.601   2.535    0.035 *
## seqn          18.795      1.739  10.809 4.73e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.79 on 8 degrees of freedom
## Multiple R-squared:  0.9359, Adjusted R-squared:  0.9279
## F-statistic: 116.8 on 1 and 8 DF,  p-value: 4.734e-06
```

pertinence des coefficients et du modèle

Le summary nous donne ici différents indicateurs pour évaluer le modèle de régression linéaire.

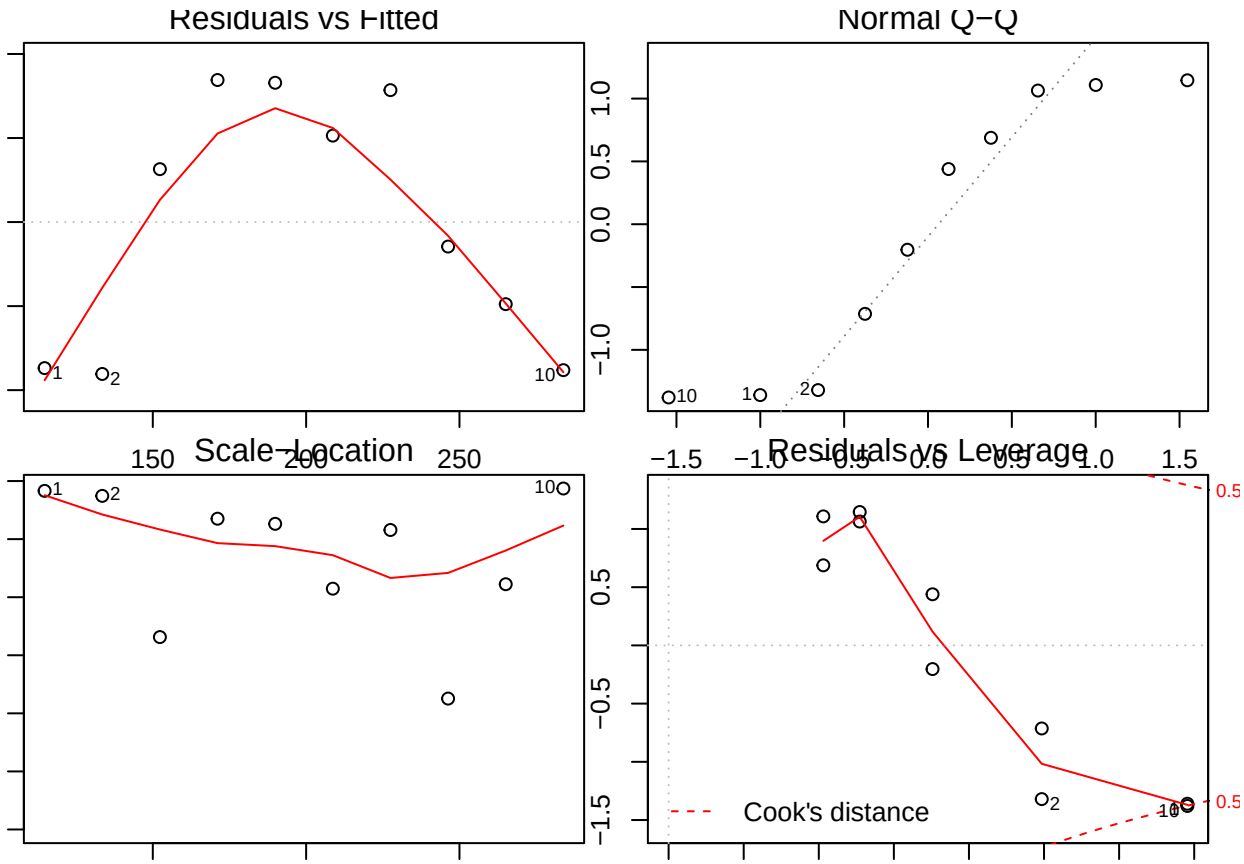
Dans un premier temps nous étudions les coefficients du modèle linéaire : la constante Intercept possède une erreur importante (une moyenne de 12,551 sur 5 exécutions). Mais elle a une p-valeur moyenne de 6,64% ce qui est important (supérieur à 5%). La valeur d'ordonnée à l'origine estimée a donc une certaine probabilité d'être erronée. le coefficient seqn a une p-valeur très faible de l'ordre de 10^{-7}. Sa valeur estimée a donc un faible risque d'être erronée. Généralement, cette valeur a une erreur relativement faible (1,399).

En moyenne, le coefficient R^2 vaut 0.9656. Cette valeur proche de 1 signifie que les observations s'éloignent peu du modèle.

On en déduit donc que la modélisation linéaire est ici cohérente.

étude des hypothèses sur les résidus.

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



Nous cherchons désormais à vérifier les hypothèses sur les résidus, en étudiant les 4 graphiques générés.

Residuals vs Fitted et Scale-Location : généralement, on n'observe pas de tendance véritablement marquée, mais le point 1 est anormal. Donc l'espérance des résidus n'est pas nulle. De plus le nuage de points ne s'écarte pas donc la variance de résidus est constante. Normal Q-Q : les points sont alignés sur la diagonale donc la distribution des résidus suit une loi normale. Residuals vs Leverage : autour de six exécutions, nous remarquons l'existence d'outliers. Les points 1 et 10 sont éloignés, notamment le point 1 qui est au-delà de la distance de Cook.

En conclusion ici l'hypothèse de l'indépendance n'est pas vérifiée. Si la plupart des points semblent confirmer une loi normale (sachant qu'un point en représente 10 puisque nous avons fait une moyenne), il faudrait tout de même reprendre l'étude après avoir retiré le point 1 des observations. Cela permettrait de conclure de nouveau, et aussi de vérifier si le point 10 est aberrant et à retirer également.

Pour vérifier si les résidus suivent bien une loi normale, nous faisons le test de Shapiro-Wilk :

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

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(temps.lm.moy)
## W = 0.85223, p-value = 0.06175
```

Pour 6 exécutions, nous obtenons une valeur moyenne de p-value de 74,12%, largement supérieure à 5%. On ne peut donc rien conclure sur la loi des résidus. Il faut donc comme conclu précédemment réaliser de nouveau une étude sans le point 1.

2.3. Comportement par rapport à la structure du graphe

Nous commençons par récupérer le jeu de données 'DonneesTSP.csv' dans un data.frame nommé data.graph. Il est constitué de la mesure du temps moyen de l'algorithme Branch&Bound sur 20 exécutions sur 73 graphes similaires.

Lecture du fichier 'DonneesTSP.csv'.

```
data.graph <- data.frame(read.csv('DonneesTSP.csv'))
```

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

```
data_temps <- log(data.graph$tps)
data.graph$dim <- sqrt(data.graph$dim)
data.graph$tps<-NULL
data_temps.lm <-lm(data_temps~.,data = data.graph)
summary(data_temps.lm)
```

```
##
## Call:
## lm(formula = data_temps ~ ., 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 ***
## dim          3.4191719  0.2391476  14.297  < 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
## ---
## 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
```

Mise en oeuvre d'une sélection de variables pour ne garder que les variables pertinentes.

La fonction step appliquée permet d'éliminer les coefficients non pertinents :

```
new_lm <-step(data_temps.lm)
```

```
## Start:  AIC=-165.23
## data_temps ~ dim + mean.long + mean.dist + sd.dist + mean.deg +
##     sd.deg + diameter
##
##             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
## - dim        1   17.3311 22.5877  -65.176
##
## Step:  AIC=-167.04
## data_temps ~ dim + mean.long + mean.dist + sd.dist + mean.deg +
##     sd.deg
##
##             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
## - dim        1   18.7788 24.0499  -62.785
```

```
summary(new_lm)
```

```
##
## Call:
## lm(formula = data_temps ~ dim + mean.long + mean.dist + sd.dist +
##     mean.deg + sd.deg, data = data.graph)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78246 -0.15445  0.00111  0.18027  0.63156
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.3960078  0.4916227  13.010  < 2e-16 ***
## dim          3.4440771  0.2298893  14.981  < 2e-16 ***
## mean.long   -4.8548566  0.7183110  -6.759 5.25e-09 ***
## mean.dist   -0.0022837  0.0008160  -2.799  0.00680 **
## sd.dist      0.0048833  0.0006372   7.663 1.39e-10 ***
## mean.deg    -0.1408227  0.0411061  -3.426  0.00108 **
## sd.deg       0.1269156  0.0807943   1.571  0.12123
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.2893 on 63 degrees of freedom
## Multiple R-squared:  0.9859, Adjusted R-squared:  0.9846
## F-statistic:   736 on 6 and 63 DF,  p-value: < 2.2e-16
```

Nous remarquons donc que la fonction step a éliminé la variable diameter, non pertinente pour notre étude, ce qui permet de minimiser le critère AIC.

Certains coefficients sont bien retenus même s'ils ne semblent pas être d'une grande importance pour l'étude, étant donné leur faible coefficient (cf sd.dist et sd.deg)

Analyse de la validité du modèle :

Pertinence de ses coefficients :

Nous pouvons donc vérifier la pertinence du nouveau modèle construit :

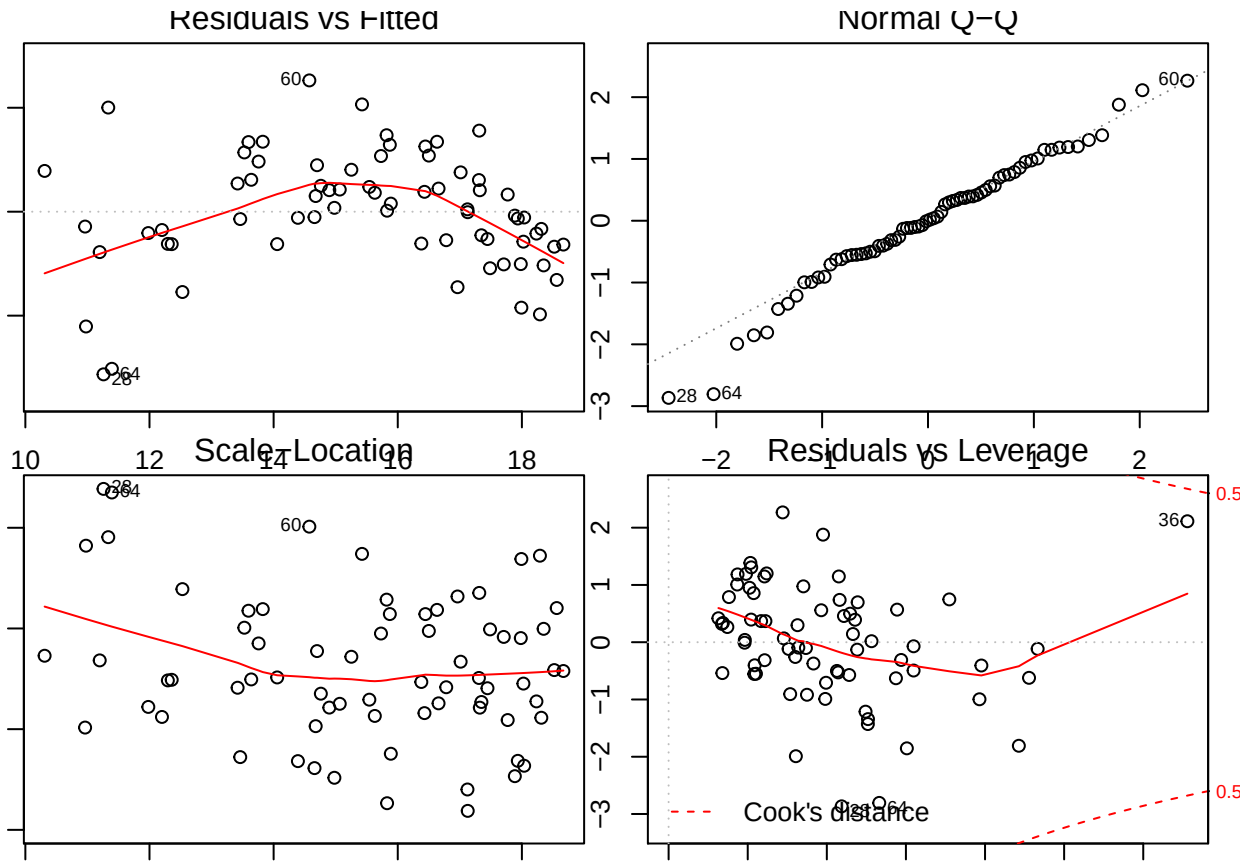
```
summary(new_lm)
```

```
##
## Call:
## lm(formula = data_temps ~ dim + mean.long + mean.dist + sd.dist +
##     mean.deg + sd.deg, data = data.graph)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.78246 -0.15445  0.00111  0.18027  0.63156
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.3960078  0.4916227  13.010  < 2e-16 ***
## dim          3.4440771  0.2298893  14.981  < 2e-16 ***
## mean.long   -4.8548566  0.7183110  -6.759 5.25e-09 ***
## mean.dist   -0.0022837  0.0008160  -2.799  0.00680 **
## sd.dist      0.0048833  0.0006372   7.663 1.39e-10 ***
## mean.deg    -0.1408227  0.0411061  -3.426  0.00108 **
## sd.deg       0.1269156  0.0807943   1.571  0.12123
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2893 on 63 degrees of freedom
## Multiple R-squared:  0.9859, Adjusted R-squared:  0.9846
## F-statistic:   736 on 6 and 63 DF,  p-value: < 2.2e-16
```

D'après les résultats présentés par la fonction summary, on peut déduire que :

Pour la plupart des coefficients, les p-value restent assez faibles ce qui nous permet de valider notre modèle multidimensionnel. En moyenne, le ratio "Multiple R-squared" vaut à peu près 0.993 (valeur très proche de 1) ce qui signifie que les observations s'éloignent peu du modèle. Nous remarquons des coefficients plus ou moins importants selon leur pertinence avec des écarts plutôt corrects (en comparant avec les modèles étudiés auparavant).

```r
par(mfrow=c(2,2)) # 4 graphiques, sur 2 lignes et 2 colonnes
par(mar=c(1,1,1,1))
plot(new_lm)
```



Nous cherchons ensuite à vérifier les hypothèses sur les résidus, en étudiant les 4 graphiques générés :

Les graphiques Residuals vs Fitted et Scale-Location : On observe que le nuage de point est réparti ce qui peut dire que la variance des résidus n'est pas constante. On remarque aussi que les points présentent une tendance pas très marquée sur le graphique ce qui nous permet de dire que l'espérance est nulle. Le graphique Normal Q-Q : Ce graphique se rapproche énormément de la linéarité . Nous pouvons donc assumer que les résidus suivent bien une loi normale. Cela reste à confirmer en effectuant le test de Shapiro-Wilk. Le graphique Residuals vs Leverage : montre l'influence des échantillons. Nous ne remarquons pas l'existence d'outliers qui représentent des points très éloignés des autres ou spécialement en dehors des bornes par rapport à la distance de Cook.

étude des hypothèses sur les résidus.

```r
shapiro.test(residuals(new_lm))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(new_lm)
## W = 0.98094, p-value = 0.3641
```

En effectuant 5 exécutions, la p-value moyenne obtenue est de 0,3641 (>5%). On peut donc conclure que les résidus ne suivent pas une loi normale.