

Fiche de Travaux Pratiques


Analyse et traitement des données Algèbre linéaire, ACP et régression linéaire

Master 1 Informatique Parcours « Données et Systèmes Connectés » (DSC) Saint-Étienne, France

Fabrice Muhlenbach

Laboratoire Hubert Curien, UMR CNRS 5516
Université Jean Monnet de Saint-Étienne
18 rue du Professeur Benoît Luras
42000 SAINT-ÉTIENNE, FRANCE
<http://perso.univ-st-etienne.fr/muhlfabr/>

Résultats attendus


L'objectif de cette séance de TP est de se familiariser avec les fonctions existant dans  pour travailler avec l'algèbre linéaire, l'analyse en composantes principales et la régression linéaire.

1 Algèbre linéaire

Une des plus importantes applications de l'algèbre linéaire est la possibilité de résoudre des systèmes d'équations linéaires. Par exemple, soient les équations l_1 et l_2 suivantes :

$$(1) \quad l_1 : 3x_1 - 4x_2 = 6$$

$$(2) \quad l_2 : x_1 + 2x_2 = -3$$

Afficher avec  les deux lignes l_1 et l_2 correspondant aux deux équations ((1) et (2)) et chercher à résoudre celles-ci de manière graphique.

Solution de manière algébrique classique :

- (1) $3x_1 - 4x_2 = 6$
- et (2) $x_1 + 2x_2 = -3$
- (1) $\Leftrightarrow 4x_2 = 3x_1 - 6 \Leftrightarrow x_2 = (3x_1 - 6)/4$
- et (2) $\Leftrightarrow 2x_2 = -x_1 - 3 \Leftrightarrow x_2 = (-x_1 - 3)/2$

```
x1 <- seq(-5, 5, length=200)
l1 <- (3*x1 - 6) / 4
l2 <- (-x1 - 3) / 2

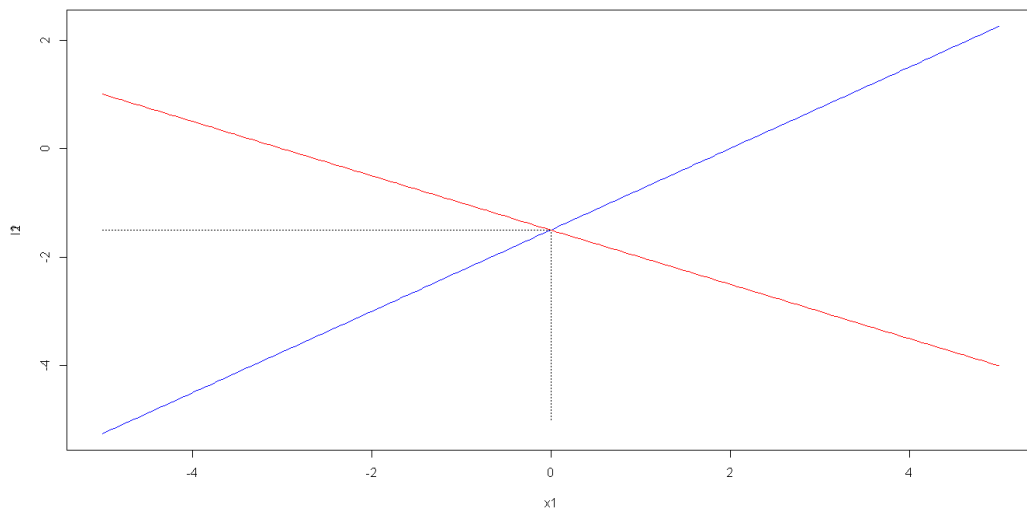
plot(x1, l1, col="blue", type="l", lwd=1, ylim=range(c(l1, l2)))
par(new=TRUE)
plot(x1, l2, col="red", type="l", lwd=1, ylim=range(c(l1, l2)))
```

Ces deux lignes l_1 (en bleu) et l_2 (en rouge) sont affichées dans la figure suivante à partir de laquelle il est possible de retrouver une solution pour laquelle $x_1 = 0$ et $x_2 = -\frac{3}{2}$. De cette figure, il suit qu'il y a 3 cas de solutions pour ce système :

1. exactement une solution (quand les deux lignes se coupent en un point) ;
2. une infinité de solutions (quand les deux lignes coïncident) ;
3. pas de solution (quand les lignes sont parallèles mais pas identiques) et, dans ce cas, le système d'équation linéaire est dit « inconsistant ».

Afficher la solution (l'intersection des deux lignes) sous forme de pointillés.

```
segments(-5, -1.5, 0, -1.5, col="black", lty=3)
segments(0, -5, 0, -1.5, col="black", lty=3)
```




L'algèbre linéaire sert, bien entendu, à résoudre les systèmes d'équations linéaires. Pour cela, on représente le système de la manière suivante :

$$\left\{ \begin{array}{l} 3x_1 - 4x_2 = 6 \\ x_1 + 2x_2 = -3 \end{array} \right\} \text{ avec } Ax = b, \text{ où } A = \begin{bmatrix} 3 & -4 \\ 1 & 2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad b = \begin{bmatrix} 6 \\ -3 \end{bmatrix},$$

et on peut le résoudre en tant que :

$$x = A^{-1}b = \begin{bmatrix} 0.2 & 0.4 \\ -0.1 & 0.3 \end{bmatrix} \begin{bmatrix} 6 \\ -3 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.5 \end{bmatrix}$$

Avec , les matrices sont inversées et les systèmes d'équations linéaires sont résolus au moyen de la fonction `solve()` (en utilisant la méthode de décomposition LU) ou avec la fonction `qr.solve()` (en utilisant la méthode de décomposition QR).

La multiplication de matrices peut être effectuée au moyen de l'opérateur `%*%`.

Par exemple :


```
A <- matrix(c(3, 1, -4, 2), ncol=2) ; print(A)
Ainv <- solve(A) ; print(Ainv)
b <- matrix(c(6, -3), ncol=1) ; print(b)
x <- Ainv %*% b ; print(x)
```

La matrice x est la solution :

$$x = \begin{bmatrix} 0 \\ -1.5 \end{bmatrix}$$

$$\Leftrightarrow x_1 = 0 \text{ et } x_2 = -3/2$$

Problème

Avec , trouver la solution des équations suivantes :

$$(1) \quad 3x_1 + 2x_2 = 7$$

$$(2) \quad x_1 - 3x_2 = -5$$

Solution

```
> A <- matrix(c(3, 1, 2, -3), ncol=2) ; print(A)
      [,1] [,2]
[1,]    3    2
[2,]    1   -3
> Ainv <- solve(A) ; print(Ainv)
      [,1] [,2]
[1,] 0.27272727 0.1818182
[2,] 0.09090909 -0.2727273
> b <- matrix(c(7, -5), ncol=1) ; print(b)
      [,1]
[1,]    7
[2,]   -5
> x <- Ainv %*% b ; print(x)
      [,1]
[1,]    1
[2,]    2
```

La matrice x est la solution :

$$x = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$\Leftrightarrow x_1 = 1$ et $x_2 = 2$

On peut vérifier le résultat de la manière suivante :

```
> x1 <- 1
> x2 <- 2
> 3*x1 + 2*x2
[1] 7
> x1 - 3*x2
[1] -5
```

2 Analyse en Composantes Principales (ACP)

Comment représenter des données de grandes dimensions ?

Trois caractéristiques essentielles des planètes de notre système solaire sont respectivement :

- a) la distance de la planète au soleil (en unité astronomique : 1 UA = 150 millions de km) ;
- b) le diamètre de la planète au niveau de l'équateur (en km) ;
- c) la densité de la planète (en g/cm³).

	Distance	Diamètre	Densité
Mercure	0.387	4878	5.42
Vénus	0.723	12104	5.25
Terre	1.000	12756	5.52
Mars	1.524	6787	3.94
Jupiter	5.203	142800	1.31
Saturne	9.539	120660	0.69
Uranus	19.180	51118	1.29
Neptune	30.060	49528	1.64
Pluton	39.530	2300	2.03


Récupérer le fichier CSV de données (sur le site Claroline) afin de le stocker sur un répertoire (par exemple, depuis le répertoire de travail, sous un répertoire appelé “R”, dans un sous-répertoire “data”), et utiliser les noms de la première colonne pour nommer chacune des lignes (après cela, on peut supprimer la première colonne devenue inutile).

```
library(readr)
planets <- as.data.frame(read_csv(file = "~/R/data/planets.csv",
                                   col_names = TRUE))

planets
summary(planets)
names <- planets[,1]
names
rownames(planets) <- names
planets
```

```
planets[,1] <- NULL
planets
```

2.1 Représentation par paires

La jeu de données comporte 3 variables numériques. Nous pouvons représenter les données avec une représentation des variables par paires (2 par 2) avec la fonction `pairs()` de .

```
pairs(planets)
```

Il est difficile de parvenir à une conclusion pertinente à partir de l’observation de ce graphique. Avec les variables de diamètre et de distance, quelques points sont concentrés dans un unique ensemble (un *cluster*). Ce problème est lié à l’échelle astronomique et peut être résolu en tenant compte d’une transformation logarithmique des variables.

```
planets.log <- log(planets)
colnames(planets.log) <- paste("log(", colnames(planets), ")", sep="")
pairs(planets.log)
```

Cette nouvelle représentation est meilleure que la précédente, néanmoins nous avons plusieurs graphiques à étudier (si nous avons d variables, nous avons $d \times (d-1)$ graphiques à voir, ou au moins $\frac{d \times (d-1)}{2}$ graphiques à étudier), et nous n’avons pas d’information sur le nom des planètes représentées.

2.2 Représentation en 2D et affichage de la taille


Dans ce contexte, nous avons une variable représentant le diamètre de la planète (la seconde variable). Nous pouvons utiliser cette information pour changer la taille du point représenté dans le graphique et utiliser les deux autres variables comme axes des abscisses et des ordonnées.

Nous pouvons afficher notre graphique dans une nouvelle fenêtre (en utilisant la fonction `win.graph` pour les utilisateurs de Windows ou `x11()` pour les utilisateurs de Linux ou macOS) avec une taille de 800×600 pixels. Le résultat est affiché sur la Figure 1.

```
win.graph(800,600,10)
plot(x = planets.log[,1], y= planets.log[,3], cex=(round(planets.log[,2])),
     xlab = colnames(planets.log[1]), ylab = colnames(planets.log[3]))
text(x=planets.log[,1], y=planets.log[,3],
     labels=names, cex = 0.8, col = "blue")
```

Ce graphique (Figure 1) est un peu trompeur. On peut en effet voir deux groupes de cercles. Les diamètres des planètes, représentés par la taille des cercles, ne sont pas représentés de la même manière dans les deux groupes (en raison de la transformation logarithmique).

2.3 Représentation en 3D

Avec , il est possible de représenter aisément les données en 3 dimensions au moyen du package `rgl`. Si ce package n’est pas installé sur votre ordinateur, il suffit d’utiliser la fonction usuelle pour l’ajouter (`install.packages("rgl")`) ou, plus simplement, depuis RStudio, d’aller dans l’onglet “Packages”, menu “Install” de la fenêtre située en bas à droite).

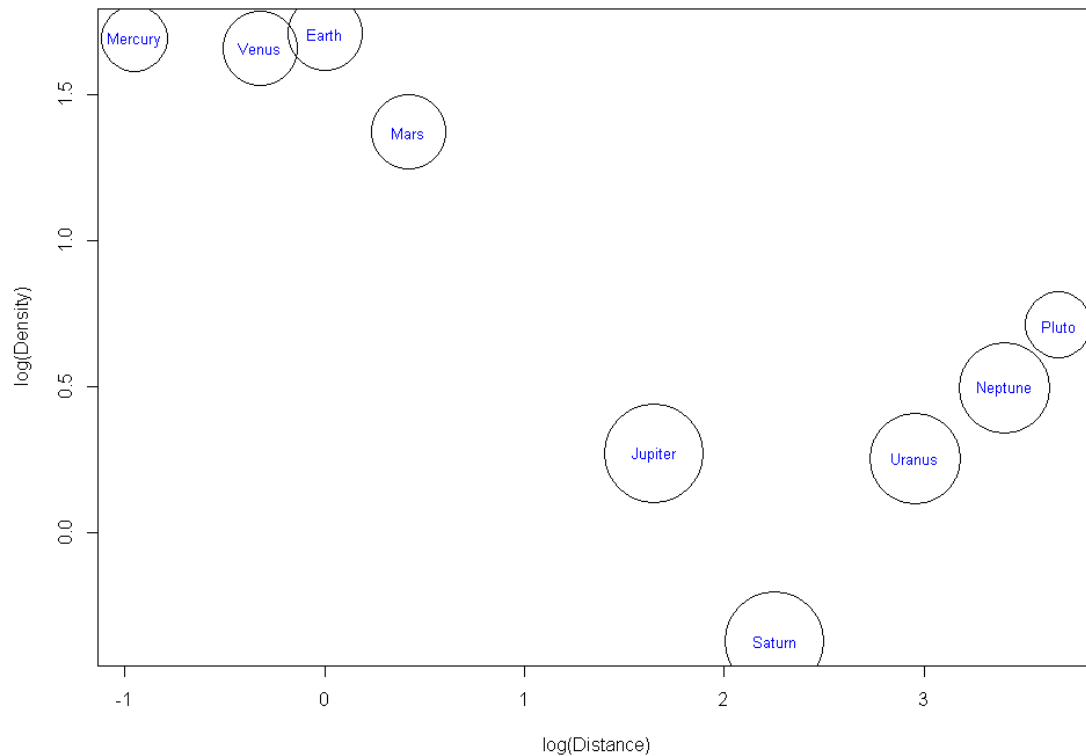


FIG. 1 – Représentation en 2D des planètes.

```
library(rgl) plot3d(x=planets$log[,1],
  y=planets$log[,2],
  z=planets$log[,3],
  xlab=colnames(planets$log[,1]),
  ylab=colnames(planets$log[,2]),
  zlab=colnames(planets$log[,3]))
text3d(x=planets$log[,1],
  y=planets$log[,2],
  z=planets$log[,3],
  text=names,
  cex=0.8,
  col="blue")
```

Ce graphique en 3 dimensions, représenté sur la Figure 2, est interactif. On peut observer qu'il y a deux groupes, ou "clusters", avec un premier composé de Mercure, Vénus, la Terre et Mars, et un autre groupe composé de Jupiter, Saturne, Uranus et Neptune. Il y a aussi un point unique (que l'on appelle en statistiques "outlier") correspondant à Pluton, dont le statut de planète a été remis en question par l'Union Astronomique Internationale (IAU) qui a reclassifié Pluton dans la catégorie des « planètes naines » en 2006.

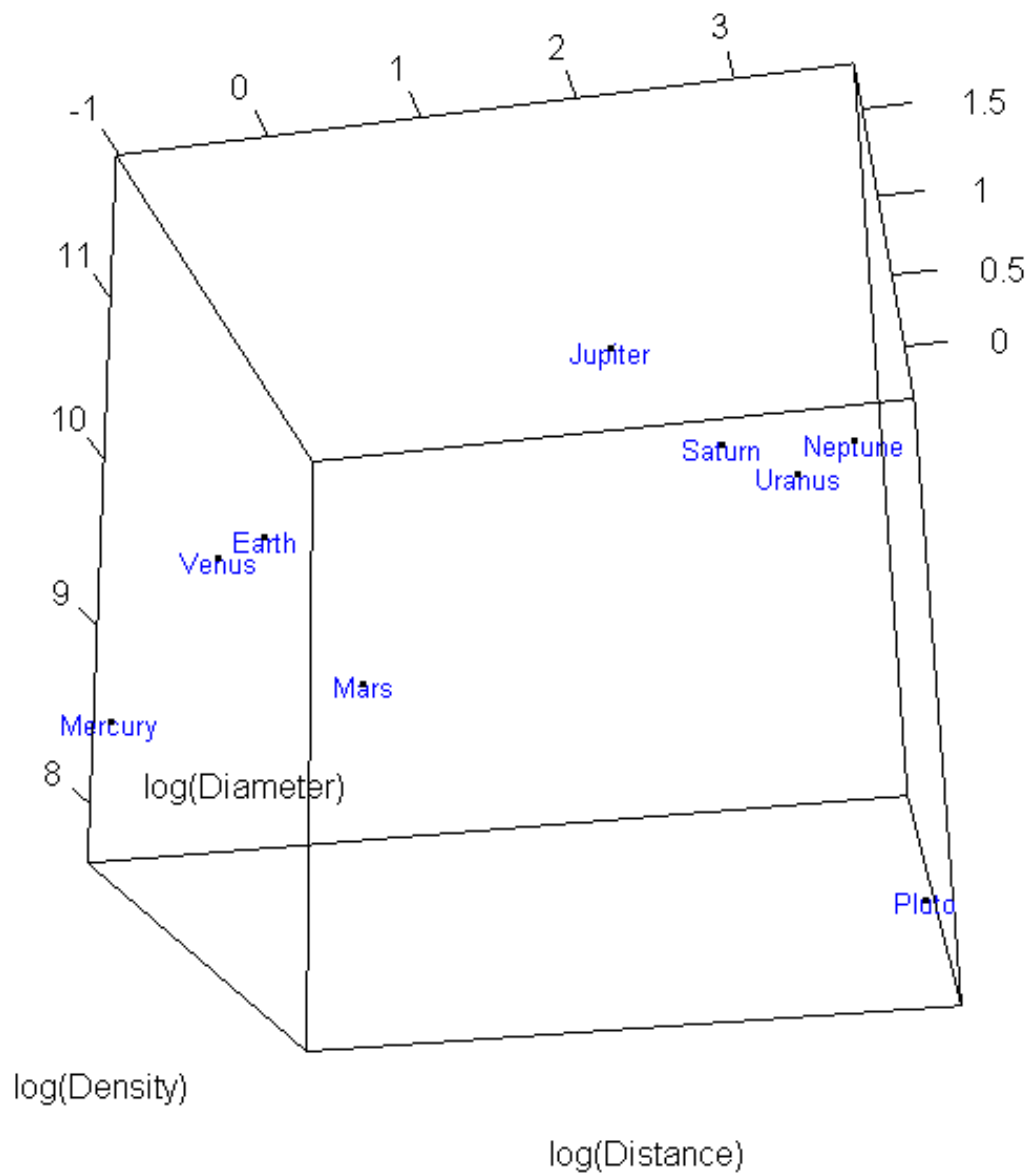


FIG. 2 – Représentation en 3D des planètes.

2.4 Représentation avec réduction de la dimensionnalité

En conclusion, même si une représentation en 3 dimensions fonctionne bien pour ce jeu de données, il faut considérer le cas où il y a bien plus de variables que simplement 3. Dans un tel contexte, il est nécessaire de procéder à une réduction de la dimensionnalité, c'est-à-dire qu'il faut trouver une représentation dans un nombre de dimensions faible (un modèle) pour des données décrites en un grand nombre de dimensions.

Avec la transformation linéaire, il est possible d'obtenir une représentation plus compacte. Cette approche est appelée « Analyse en Composantes Principales » (ACP) ou « *Principal Component Analysis* » (PCA)

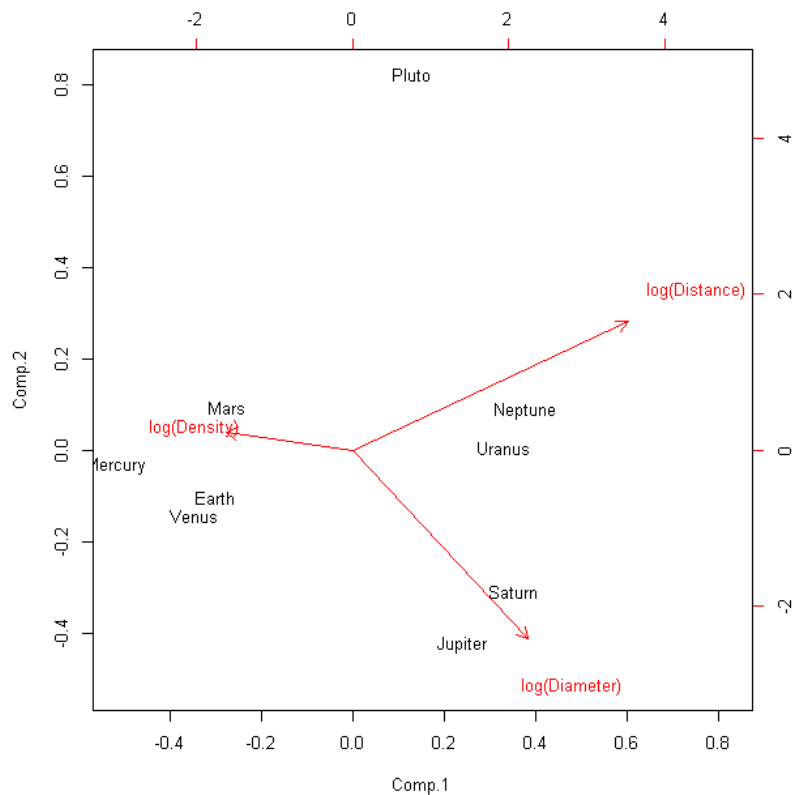


FIG. 3 – Représentation des planètes avec les 2 premières composantes.

en anglais. Cette analyse est largement utilisée pour les réductions de dimensionnalité (projeter des points d'un espace à d -dimensions à un nouveau sous-espace à M -dimensions) ainsi que pour l'extraction des caractéristiques.

Dans **R**, il est facile de calculer une ACP avec la fonction `princomp`. Celle-ci réalise l'analyse en composantes principales d'une matrice numérique donnée et retourne le résultat sous forme d'un objet de classe `princomp`.

Le calcul est réalisé avec la fonction `eigen` (pour calculer les matrices des vecteurs propres et des valeurs propres) sur les matrices de corrélations et de covariances, déterminées par `cor` (calculant la corrélation).

La méthode `print` affiche les résultats de ces objets sous un format adapté et la méthode `plot` produit un graphique, tout comme la méthode `biplot`.

```
planets.pca <- princomp(planets.log)
planets.pca
win.graph(800, 600, 10)
biplot(planets.pca)
```

La Figure 3 est obtenue en utilisant la fonction `princomp` sur les planètes et en affichant les résultats obtenus sur les nouveaux axes. On peut facilement observer les deux groupes de planètes, correspondant aux planètes terrestres et aux géantes gazeuses, ainsi que l'exemple atypique (outlier) constitué par Pluton.

3 Régression linéaire


3.1 Introduction

L'analyse par régression est utilisée pour expliquer ou modéliser la relation entre une variable Y , appelée la *réponse*, la *sortie* ou la *variable dépendante* d'une part, et d'autre part un ou plusieurs *prédicteurs*, *entrées*, *variables indépendantes* ou *variables explicatives* X_1, X_2, \dots, X_p . Dans le cas d'une seule variable explicative ($p = 1$), cette analyse est appelée « régression linéaire simple ». Pour plus d'une variable explicative ($p > 1$), cette analyse est appelée « régression linéaire multiple ».

Nous utilisons la régression pour quantifier l'effet (non connu à l'avance) du changement qu'une variable produit sur une autre variable. Lorsque nous effectuons une régression, nous faisons deux hypothèses :

1. il existe une relation linéaire entre les deux variables X et Y ;
2. cette relation est de type additive (c.-à-d. de type $Y = x_1 + x_2 + \dots + x_N$).

Une régression avec deux variables explicatives ou davantage est appelée une régression multiple. Plutôt que de modéliser la réponse moyenne à travers une ligne droite, comme dans une régression simple, la régression multiple construit un modèle comme étant une fonction de plusieurs variables explicatives à la fois.

La fonction `lm` peut être utilisée pour réaliser des régressions linéaires multiples avec  et l'essentiel de la syntaxe est la même que pour les modèles de régression linéaire simple. Pour réaliser une régression linéaire multiple avec p variables explicatives, il faut utiliser la commande suivante :

```
lm(response ~ explanatory_1 + explanatory_2 + ... + explanatory_p)
```

Ici les termes `response` et `explanatory_i` de la fonction doivent être remplacés par les noms, respectivement, de la variable réponse et des variables explicatives utilisées dans l'analyse.

3.2 Régression linéaire simple sur le jeu de données Eucalyptus

Considérons l'exemple suivant : nous essayons d'estimer la hauteur d'un eucalyptus à partir de sa circonférence. Les eucalyptus (ou « gommiers ») forment un groupe très riche d'arbres originaires d'Australie. On a l'habitude de dire des eucalyptus qu'ils sont :

- petits s'ils mesurent moins de dix mètres de haut ;
- de taille moyenne s'ils font entre dix et trente mètres ;
- grands s'ils mesurent entre trente et soixante mètres ;
- très grands s'ils atteignent plus de soixante mètres (certaines espèces atteignant 90 mètres de hauteur).

Il est ainsi beaucoup plus facile d'obtenir la mesure de la circonférence (obtenue au pied de l'arbre) que sa hauteur. C'est pourquoi il est intéressant d'avoir un modèle issu de la régression linéaire permettant de prédire sa taille à partir de sa circonférence.

Pour cela, importer le jeu de données `eucalyptus`, afficher le résumé du jeu de données et afficher la hauteur (variable `ht` en mètres) en fonction de la circonférence (variable `circ` en cm).

```
euc <- read.table("eucalyptus.txt", header=T)
summary(euc)
plot(ht~circ, data=euc)
```

Nous calculons maintenant une régression linéaire, c'est-à-dire une phase d'estimation, en utilisant la fonction `lm`. Cette fonction est utilisée pour s'ajuster à des modèles linéaires.

```
regeuc <- lm(ht~circ , data=euc )
summary(regeuc)
regeuc
```

En sortie de notre fonction, nous avons la matrice d'information sur les coefficients avec 4 colonnes et il y a autant de lignes qu'il y a de coefficients.

Le modèle est une fonction affine, c'est-à-dire une fonction obtenue par addition et multiplication de la variable par des constantes de type $f(x) = ax + b$ qui se manifeste dans notre cas sous la forme suivante :

$$ht = \beta_0 + \beta_1 \times circ$$

Avec les valeurs trouvées par le modèle, à savoir 9.0375 pour `Intercept`, c'est-à-dire l'intersection (la constante à additionner), et 0.2571 pour `circ`, c'est-à-dire la pente de la droite (la constante à multiplier), nous avons ainsi :

$$ht = 9.037476 + 0.257138 \times circ$$

Avec la fonction `attributes`, nous pouvons connaître les différents attributs de l'objet "lm" qu'est `regeuc`.

```
attributes(regeuc)
```

Les coefficients sont stockés dans la variable `coefficients` et nous pouvons y accéder au moyen de `regeuc$coefficients` ou plus simplement avec `coef(regeuc)`.

Pour vérifier la qualité du modèle par rapport aux observations, nous pouvons tracer la droite de régression sur le graphe des observations. Puisqu'il y a une certaine incertitude sur les paramètres estimés, nous traçons aussi un intervalle de confiance à 95% de la prédiction. Tout d'abord, nous traçons la ligne obtenue par le modèle de régression linéaire (fonction `abline`). Puis nous pouvons définir l'échelle des abscisses. Après cela, nous utilisons la fonction `predict` pour obtenir les prédictions du modèle avec un intervalle de confiance de 95%. Finalement, nous pouvons tracer les colonnes de la matrice de la circonférence opposé aux colonnes de l'intervalle de confiance de la prédiction.

```
abline(regeuc , col="red")
circ=seq(min(euc[, "circ"]), max(euc[, "circ"]), length=100)
grid<-data.frame(circ)
CIpred<-predict(regeuc , new=grid , interval="pred" , level=0.95)
matlines(grid$circ , cbind(CIpred) , lty=c(1,2,2) , col=2)
```

Le modèle est assez bon, mais vraisemblablement perfectible. Les données d'observation des eucalyptus ne suivent pas complètement une droite mais semblent s'arrondir à la manière d'une parabole, c'est-à-dire une fonction de type logarithmique ou une fonction avec un exposant inférieur à 1 comme la racine carrée.

3.3 Régression linéaire multiple sur le jeu de données Eucalyptus

Nous allons maintenant essayer un modèle de régression linéaire multiple de type $ht = \beta_0 + \beta_1 \times circ + \beta_2 \times \sqrt{circ}$.

```

multreg<-lm(ht~circ+I(sqrt(circ)),data=euc)
summary(multreg)
plot(ht~circ,data=euc)
circ=seq(min(euc[, "circ"]),max(euc[, "circ"]),length=100)
grid2<-data.frame(circ)
CIpred2<-predict(multreg,new=grid2,interval="pred",level=0.95)
matlines(grid2$circ,cbind(CIpred2),lty=c(1,2,2),col=2)

```

En affichant directement le modèle, nous pouvons obtenir :

- les résidus contre les valeurs ajustées ;
- les résidus standardisés en fonction des quantiles théoriques (*normal Q-Q*) ;
- la racine carrée des résidus standardisés en fonction des valeurs ajustées (emplacement d'échelle) ;
- les résidus contre l'effet de levier (*leverage*).

```

win.graph(800,600,10)
plot(regeuc)
plot(multreg)

```