



REPUBLIQUE DU SENEGAL



Un Peuple - Un But - Une Foi

MINISTERE DE L'ECONOMIE DU PLAN ET DE LA COOPÉRATION

AGENCE NATIONALE DE LA STATISTIQUE ET DE LA
DÉMOGRAPHIE

ECOLE NATIONALE DE LA STATISTIQUE ET DE L'ANALYSE
ÉCONOMIQUE



Projet BIG DATA, ITS4

RÉGRESSION LINÉAIRE AVEC RHadoop

Rédigé par :

Khariratou DIALLO

Danis Rikel JIOGUE TAMATIO

Supervisé par :

Dr Cheikh BA

Chargé du cours de BIG DATA à l'ENSAE - Dakar

Mars 2020



REMERCIEMENTS

Qu'il nous soit permis d'exprimer notre profonde gratitude à tous ceux qui ont participé à l'élaboration de la présente étude.

Nos remerciements vont à l'endroit de tout le personnel administratif de l'ENSAE, particulièrement au directeur de l'école, M. Abdou DIOUF, à notre responsable de filière Dr. Souleymane FOFANA, au responsable de la filière ISE, M. Idrissa DIAGNE, à Dr. Souleymane DIAKITE, ingénieur statisticien économiste et tout le corps professoral de l'ENSAE, pour leur dévotion à faire de l'excellence, la devise de l'école.

Nous tenons également à remercier l'ensemble des enseignants qui interviennent dans notre formation afin de faire de nous les as de la statistique. Un remerciement tout particulier à l'enseignant du cours de BIG DATA, Dr Cheikh BA qui en plus des cours est un très bon conseiller soucieux de notre réussite scolaire et professionnel.

Nous ne saurions terminer ces remerciements sans y associer les élèves de l'ENSAE, particulièrement nos camarades de classe de la promotion 2016-2017 auprès de qui nous avons beaucoup appris mais aussi tous ceux qui, de près ou de loin, nous ont soutenus avec leurs prières, leurs encouragements.

AVANT PROPOS

L'Ecole Nationale de la Statistique et de l'Analyse Economique (ENSAE) est une école à vocation sous régionale créée en 2008 et située à Dakar. Elle constitue une direction de l'Agence Nationale de la Statistique et de la Démographie (ANSD) et s'intègre dans un réseau coordonné par le Centre d'Appui aux Écoles de Statistique Africaines (CAPESA), avec l'Institut Sous régional de Statistique et d'Économie Appliquée (ISSEA-Yaoundé) et l'École Nationale Supérieure de Statistique et d'Économie Appliquée (ENSEA-Abidjan).

L'École propose des formations qui se déroulent dans trois cycles : les Techniciens Supérieurs de la Statistique (TSS), les Ingénieurs des Travaux Statistiques (ITS) et les Ingénieurs Statisticiens Economistes (ISE).

Durant la formation des Ingénieurs des Travaux Statistiques, il est prévu un cours de BIG DATA en 4ième année pour l'option informatique décisionnelle. Le cours porte essentiellement sur l'introduction aux concepts clés en BIG en DATA. Lors de ce cours, les élèves acquièrent des connaissances théoriques et pratiques. Dans le cadre de la mise en pratique et l'approfondissement des connaissances, il est demandé à la fin du cours un projet. Ainsi, ce présent rapport a pour objectif de proposer un outil de régression linéaire à travers le framework HADOOP sur l'environnement HDSF et paradigme mapreduce.

Table des matières

REMERCIEMENTS	1
AVANT-PROPOS	2
1 Régression Linéaire : Cas de l'ajustement affine	4
1.1 Description de l'ajustement affine	4
1.2 Solution MapReduce pour l'ajustement affine	5
1.2.1 Pré-traitement des données & split	5
1.2.2 Les couples en entrée et sortie pour chacun des Mappers et Reducers	5
1.2.3 Programme Map	6
1.2.4 Programme Reduce	8
1.2.5 Application	9
2 Utilisation de R pour l'application : librairie RHadoop	10
2.1 Difficultés d'installation	10
2.2 Application	11
2.2.1 Instruction d'installation de la librairie rmr2	12
2.2.2 Etape de prétraitement, split et fonction Map pour l'ajus- tement affine sous R	12
2.2.3 Fonction Reduce pour l'ajustement affine sous R	14
2.2.4 Illustration graphique	16

Chapitre 1

Régression Linéaire : Cas de l'ajustement affine

Dans l'optique de bien comprendre, expliquer ou prédire le fonctionnement d'une ou de plusieurs variables appelées **variable(s) à expliquer ou dépendante(s)**, à partir d'une ou de plusieurs variables appelées **variables explicatives ou indépendantes**, on fait généralement appel à la technique de **régression linéaire**. Ainsi, dans le cas de plusieurs variables considérées comme étant des variables indépendantes on parle de **Régression linéaire multiple**, tandis que lorsqu'on dispose d'une seule variable explicative, elle est considéré comme étant la **régression linéaire simple**. Dans la suite, nous allons particulièrement nous intéresser à cette situation. Dans un premier temps, nous allons faire une brève description de cette méthode, ensuite, nous présenterons les différentes étapes d'application de cette technique de régression dans le cas d'une simulation d'une situation de données massives avec un algorithme *MapReduce*.

1.1 Description de l'ajustement affine

L'ajustement affine, dont il est question dans ce document traduit une situation où l'on recherche la droite permettant d'expliquer le comportement d'une variable dépendante y comme étant une fonction affine d'une autre variable indépendante x . La situation se traduit par l'équation suivante :

$$(a) \quad Y_i = \beta + aX_i$$

Avec :

- Y_i : variable d'intérêt (variable à expliquer, variable dépendante) ;
- X : variable explicative, variable indépendante ;
- β : la constante du modèle, qui est un réel ;
- a : coefficient associé à la variable X .

Résoudre l'équation (a) revient à chercher une droite qui s'ajuste le mieux possible à ce nuage de points. Parmi toutes les droites possibles, on retient celle qui jouit d'une propriété remarquable : c'est celle qui rend minimale la somme des carrés des écarts des valeurs observées Y_i à la droite $Y_i^* = aX_i + \beta$. Si ϵ_i représente cet écart, appelé aussi **résidu**, le principe des *moindres carrés ordinaire (MCO)* consiste à choisir les valeurs de a et de b qui minimisent

$$E = \sum_{i=1}^n (\epsilon_i^2) = \sum_{i=1}^n (Y_i^* - (aX_i + \beta))^2$$

A l'issue de la résolution de ce programme les coefficients *aetb* obtenus sont ainsi spécifiés :

$$\Rightarrow a^* = \frac{Cov(x,y)}{V(x)} ;$$

$$\Rightarrow b^* = \bar{Y} - a\bar{X} ;$$

$$(a) \quad \bar{X} = \frac{1}{n} * \sum_{i=1}^n x_i ; \bar{Y} = \frac{1}{n} * \sum_{i=1}^n y_i ;$$

$$(b) \quad V(X) = \frac{1}{n} * \sum_{i=1}^n (x_i - \bar{X}) ; V(Y) = \frac{1}{n} * \sum_{i=1}^n (y_i - \bar{Y}) ;$$

$$(c) \quad Cov(x; y) = \frac{1}{n} * \sum_{i=1}^n (x_i - \bar{X})(y_i - \bar{Y}) = \frac{1}{n} * \sum_{i=1}^n x_i * y_i - \bar{X} * \bar{Y}$$

1.2 Solution MapReduce pour l'ajustement affine

Le paradigme *MapReduce* est un environnement qui sert à répondre à l'une des principales problématiques de traitement des données massives. En effet, pour ces types de données, la méthode de traitement repose sur un algorithme ayant la capacité d'être distribué, synchrone et cohérent. Le framework **Hadoop** grâce à son système de fichier HDFS et son environnement de programmation (résolution de tâche) *paradigme MapReduce* permet de bénéficier la capacité en intégrant toutes ces exigences en plus d'un système de replication permettant d'assurer la résilience de l'équipement en cas d'éventuelle panne. Dans cette partie, il est question d'écrire le programme *MapReduce* pour la résolution de notre équation d'ajustement affine précédemment définie dans le cas de données massives. Pour parvenir à cette fin, nous allons suivre les 4 étapes suivantes, d'abord faire un pré-traitement de nos données, ensuite définir la façon dont nos données seront entrées dans le programme map et reduce. Par la suite nous donnerons le pseudo code d'exécution du Map et le pseudo code du programme Reduce et enfin nous présenterons le cas pratique de la table1.

1.2.1 Pré-traitement des données & split

On dispose d'un tableau de deux colonnes et n lignes, n étant le nombre d'individus. La première colonne représente la variable explicative c'est-à-dire les x et la deuxième colonne représente la variable à expliquer c'est-à-dire les y.

Considérons que nous disposons un fichier composé de l'ensemble des observations des variables X et Y observées sur une population bien définie (exemple : la population des patients de l'hôpital FANN de Dakar depuis 2000, la variable X : l'âge du patient, Y : le poids du patient. On recherche l'ajustement entre le poids et la taille de ces derniers).

1.2.2 Les couples en entrée et sortie pour chacun des Mappers et Reducers

— **Pour les mapper :**

Les données en entrée pour le mapper seront les couples sous la forme (null , jeu

de données) où le jeu de données est un tableau à deux colonnes (la première ligne étant X et la deuxième ligne Y) et n lignes, n étant le nombre d'individus. Ainsi, on aura les données en entrée sous la forme (null, le tableau de données).

Pour la sortie des mappers, on aura besoin des couples composés des données de x, des données de y et du nombre d'individus. La clé sera alors le vecteur (x,y,n) et la valeur sera une liste (vecteur des données de x, vecteur des données de y, l'entier représentant le nombre d'individus). On aura donc les couples en sortie des Mappers sous la forme suivante ((x,y,n) , vecteur des données de x, vecteur des données de y, le nombre d'individus). Par exemple, ((x, y, n), (124, 54, 86, 35), (659, 48, 87, 32), 4).

— **Pour les reducers :**

Après l'opération MAP, le shuffle and Sort de Hadoop permet de regrouper les couples par clé. Sauf que dans notre cas, on a une clé unique. Les couples en sortie des Mappers seront les couples en entrée des Reducers. Les couples seront de la forme ((x,y,n), vecteur des données de x, vecteur des données de y, le nombre d'individus).

Après l'exécution du programme Reduce, les couples en sortie des Reducers seront (a ; valeur) et (b ;valeur).

1.2.3 Programme Map

Le pseudo code pour le Map est le suivant :

Function Mapper

Fonction Map(null, data)

Début

```
Taille_ligne = LONGUEUR(Fragment[0]); // Représente le nombre d'observation de la var X ou Y
value_X = Vecteur(taille = Taille_ligne // Design du Vecteur des observation de la variable X ;
value_Y = Vecteur(taille = Taille_ligne // Design du Vecteur des observation de la variable X ;
valeur_M = liste (vecteur(taille = Taille_ligne), vecteur(taille = Taille_ligne), Taille_ligne) //Design de la valeur de sortie du
mapper
```

```
Pour j = 1 à j <= 2, faire :
```

```
    si(j==1), faire //X est sur la première ligne de la base de donnée Récupération des observations de X
```

```
        pour i = 1 à i < Taille_ligne, faire
```

```
            value_X[i] = data[i,j]
```

```
            valeur_M[1][i] = value_X[i]
```

```
        finPour
```

```
    finSi
```

```
    Sinon si(j==2), faire //Y est sur la deuxième ligne de la base de donnée Récupération des observations de Y
```

```
        pour i = 1 à i < Taille_ligne, faire
```

```
            value_Y[i] = data[i,j]
```

```
            valeur_M[2][i] = value_Y[i]
```

```
        finPour
```

```
    finSi
```

```
finPour
```

```
Cle = vecteur(" X" , " Y" , « n" )
```

```
GENRER (cle, valeur_M)
```

Fin

1.2.4 Programme Reduce

Function Reduce

```

Fonction Moyenne_x(valeur_M){
    som_x = 0
    pour i = 0 à i < valeur_M[3], faire
        som_x += valeur[1][i]
    finPour
    retourner(som_x / valeur_M[3])
}

Fonction Moyenne_y(valeur_M){
    som_y = 0
    pour i = 0 à i < valeur_M[3], faire
        som_y += valeur[2][i]
    finPour
    retourner(som_y / valeur_M[3])
}

Fonction Variance_X(valeur_M){
    som_carre_x = 0
    pour i = 0 à i < valeur_M[3], faire
        som_carre_x += valeur[1][i] * valeur[1][i]
    finPour
    var_x = som_carre_x / valeur_M[3] - Moyenne_x(valeur_M) * Moyenne_x(valeur_M)
    retourner(var_x)
}

Fonction covariance(valeur_M){
    som_xy = 0
    pour i = 0 à i < valeur_M[3], faire
        som_xy += valeur[1][i] * valeur[2][i]
    finPour
    cov_xy = som_xy / valeur_M[3] - Moyenne_x(valeur_M) * Moyenne_y(valeur_M)
    retourner(cov_xy)
}

Fonction Reduce(key, val)
Début
    a.estim = covariance(valeur_M) / Variance_X(valeur_M)
    b.estim = Moyenne_y - a.estim * Moyenne_x(valeur_M)
    cle = vecteur("a.estim", "b.estim")
    valeur = vecteur(a.estim, b.estim)
    RENVOYER(cle, valeur)
Fin

```

1.2.5 Application

Dans l'exemple de la table 1, nous avons un tableau à 2 dimensions. Nous allons disposer le tableau de telle sorte que nous aurons un tableau de 2 colonnes et 7 lignes. Ainsi, on aura besoin d'un seul mapper pour traiter le couple en entrée (.,jeux de données). Le mapper se chargera de transformer le couple en entrée en un couple de la forme (x,y,n), vecteur des données de x, vecteur des données de y, le nombre d'individus). Dans ce cas précis, le couple sera sous la forme ((x,y,n), (100,110,120,130,140,150,160), (105,95,75,68,53,46,31), 7).

Finalement, un reducer sera nécessaire pour faire les calculs et renvoyer le résultat attendu a et b.

Données d'entrée

X	Y
100	105
110	95
120	75
130	68
140	53
150	46
160	31

MAP

$((\underline{x,y,n}), \{ (100,110,120,130,140,150,160), (105,95,75,68,53,46,31), 7 \})$

REDUCE

$\underline{\text{MoyX}} = 130$

$\underline{\text{MoyY}} = 67,57$

$\underline{\text{VarX}} = 400$

$\underline{\text{Cov}} = -488,57$

Résultat final

$(\underline{a}, -1, 2, \dots)$

$(\underline{b}, 226, \dots)$

Utilisation de R pour l'application : librairie RHadoop

2.1 Difficultés d'installation

Pour l'installation de cette librairie, deux difficultés principales ont été repertoriées :

1. **l'identification et l'installation des packages sous-jacents** : ces derniers vont permettre aux différentes librairies de HADOOP dans R d'être indépendant, c'est-à-dire de pouvoir faire des traitement de bases de données sous r, manipuler les chaînes de caractères etc..

```
install.packages(c("Rcpp","RJSONIO","bitops","digest","functional"))  
install.packages(c("stringr","plyr","reshape2","dplyr","R.methodsS3"))  
install.packages(c("caTools","Hmisc","bit64","rJava","rjson"))
```

2. **L'installation préalable du package centrale "ravro"** : Ce n'est qu'après l'installation de ce dernier que l'on peut installer les différents packages de HADOOP (rhbase, rmr2, rhdfs, plyrmr) dans r. En effet, la librairie permet de s'assurer que les packages sous-jacents au fonctionnement des packages de HADOOP sont bien présents. S'il en manque il va renvoyer un message d'erreur en précisant la librairie manquante.
3. **installation du package rmr2**. Car, tel qu'indiqué sur le graphe ci-dessous on constate bien la présence fichier zip *rmr-3.3.1 for Windows*. Or ce dernier crée amalgame dans la mesure où il ne conduit pas à l'installation du package. C'est plutôt *rmr-3.3.1* qui permet d'installer.

4. **Définition de l'environnement de travail HADOOP** : Cette instruction est reconnue par l'erreur suivante de R :

Error in hadoop.cmd() : Please make sure that the env. variable HADOOP_CMD is set.

Pour résoudre ce problème, il faut au préalable en début du script définir cet environnement de travail. L'environnement de travail que avons adopté est le suivant :

```
Sys.setenv(HADOOP_CMD="/usr/local/hadoop220/bin/hadoop")  
Sys.setenv(HADOOP_STREAMING="/usr/local/hadoop220/share/hadoop/tools/lib/  
streaming-2.2.0.jar")  
Sys.getenv("HADOOP_CMD")
```

Downloads

Antonio Piccolboni edited this page on 14 Feb 2015 · 89 revisions

Download The Latest Official RHadoop Releases

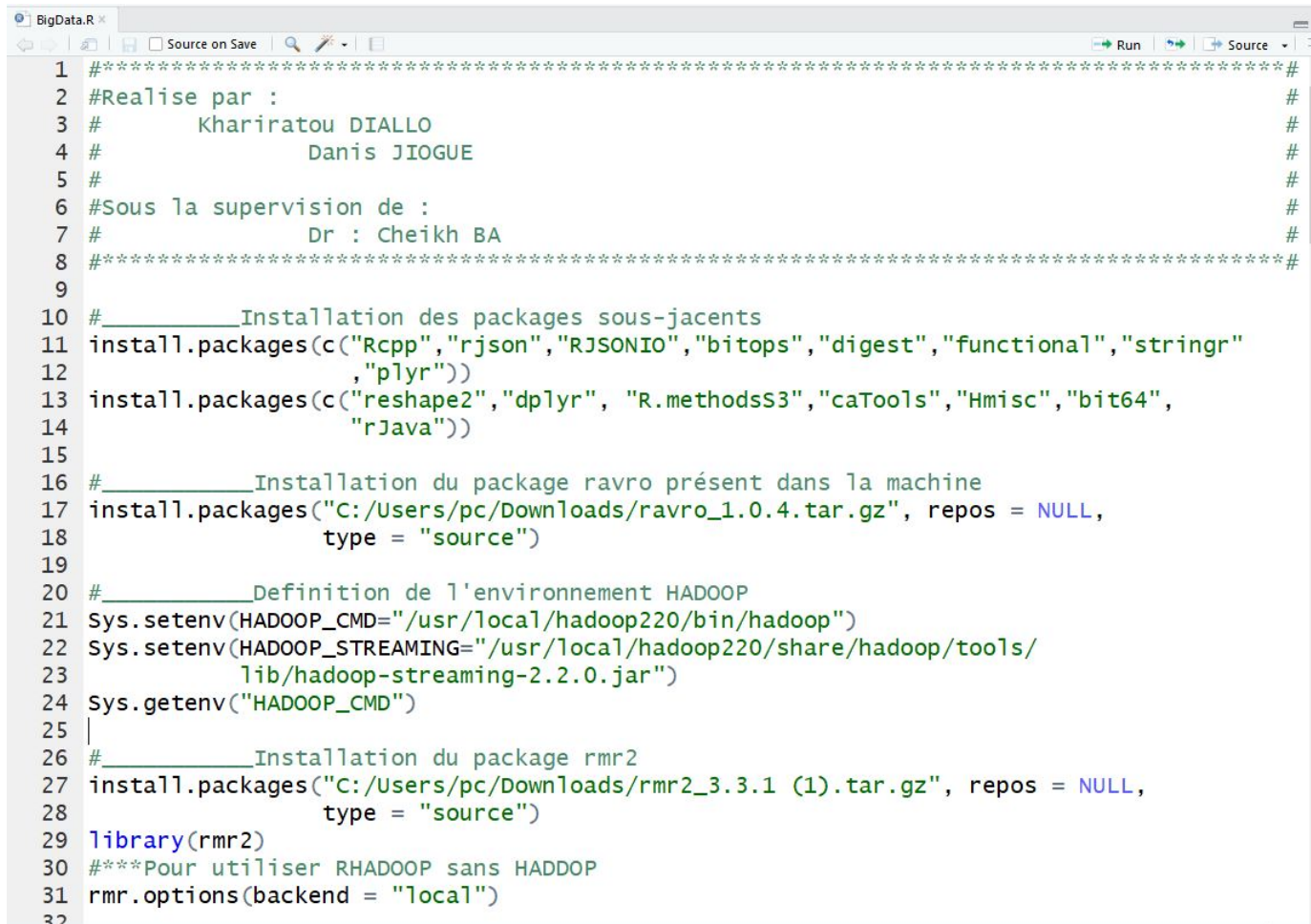
- [ravro-1.0.3](#)
- [plyrmr-0.6.0](#)
- [rmr-3.3.1](#)
- [rmr-3.3.1 for Windows](#)
- [rhdfs-1.0.8](#)
- [rhdfs-1.0.8 for Windows](#)
- [rhbase-1.2.1](#)

2.2 Application

En guise d'application, tel que dit dans l'énoncé, nous allons appliquer l'algorithme précédemment avec le logiciel R, l'Environnement de Développement Intégré (IDE) utilisé est ***rstudio*** qui donne plus de convivialité dans l'exécution des tâches. Afin de mieux illustrer le travail, on s'aidera des captures d'écran. Ainsi, dans la suite, nous présenterons d'abord les instructions relatives à l'installation de la librairie ***rmr2***, ensuite nous procéderons à la présentation des fonctions de map et reduce avec les résultats à l'appui.

2.2.1 Instruction d'installation de la librairie rmr2

La sortie des instructions pour l'installation du package est illustrée dans le graphique suivant



```

1  #####
2  #Realise par :
3  #      Khariratou DIALLO
4  #      Danis JIOGUE
5  #
6  #Sous la supervision de :
7  #      Dr : Cheikh BA
8  #####
9
10 #_____Installation des packages sous-jacents
11 install.packages(c("Rcpp","rjson","RJSONIO","bitops","digest","functional","stringr"
12                  ,"plyr"))
13 install.packages(c("reshape2","dplyr", "R.methodss3","caTools","Hmisc","bit64",
14                  "rJava"))
15
16 #_____Installation du package ravro présent dans la machine
17 install.packages("C:/Users/pc/Downloads/ravro_1.0.4.tar.gz", repos = NULL,
18                 type = "source")
19
20 #_____Definition de l'environnement HADOOP
21 Sys.setenv(HADOOP_CMD="/usr/local/hadoop220/bin/hadoop")
22 Sys.setenv(HADOOP_STREAMING="/usr/local/hadoop220/share/hadoop/tools/
23             lib/hadoop-streaming-2.2.0.jar")
24 Sys.getenv("HADOOP_CMD")
25 |
26 #_____Installation du package rmr2
27 install.packages("C:/Users/pc/Downloads/rmr2_3.3.1 (1).tar.gz", repos = NULL,
28                 type = "source")
29 library(rmr2)
30 #***Pour utiliser RHADOOP sans HADDOP
31 rmr.options(backend = "local")
32

```

2.2.2 Etape de prétraitement, split et fonction Map pour l'ajustement affine sous R

La résultat de cette fonction est illustrée par la sortie suivante :

```

#-----Préparation du jeux de donnee (split & pretraitement)
data <- read.csv(choose.files(),header = T, sep = ";")
data

|
#-----FUNCTION MAPPER
cle <- "xy"
mapper <- function(null, data){
  Taille_ligne = nrow(data)
  value_X = rep(NA, times = Taille_ligne) #Recupere les observations xi
  value_Y = rep(NA, times = Taille_ligne) #Recupere les observation yi
  cle_M <- list(NA,NA) # Definition du format de la clé en sortie qui est une liste
  valeur_M <- list(rep(NA, times = Taille_ligne),rep(NA, times = Taille_ligne),Taille_ligne)
  for (j in 1:2) {
    if(j==1){ #Nous sommes a la premiere ligne donc les observations sont
      #relatives a x d'ou la cle = X
      cle_M[[1]] = "X"
      for (i in 1:Taille_ligne) {
        value_X[i] = data[i,j]
        valeur_M[[1]][i] = value_X[i]
      }
    }
    else if(j==2){ #Nous sommes a la deuxieme ligne donc les observations
      #sont relatives a Y d'ou la cle = Y
      cle_M[[2]] = "Y"
      for (k in 1:Taille_ligne) {
        value_Y[k] = data[k,j]
        valeur_M[[2]][k] = value_Y[k]
      }
    }
  }
  cle <- c("X","Y","n")
  keyval(cle,valeur_M) #on renvoie une cle est une liste comme valeur
}

```

```

> mapper <- function(null, data){
+   Taille_ligne = nrow(data)
+   value_X = rep(NA, times = Taille_ligne) #Recupere les observations xi
+   value_Y = rep(NA, times = Taille_ligne) #Recupere les observation yi
+   cle_M <- list(NA,NA) # Definition du format de la clé en sortie qui est une liste
+   valeur_M <- list(rep(NA, times = Taille_ligne),rep(NA, times = Taille_ligne),Taille_ligne)
+   for (j in 1:2) {
+     if(j==1){ #Nous sommes a la premiere ligne donc les observations sont
+       #relatives a x d'ou la cle = X
+       cle_M[[1]] = "X"
+       for (i in 1:Taille_ligne) {
+         value_X[i] = data[i,j]
+         valeur_M[[1]][i] = value_X[i]
+       }
+     }
+     else if(j==2){ #Nous sommes a la deuxieme ligne donc les observations
+       #sont relatives a Y d'ou la cle = Y
+       cle_M[[2]] = "Y"
+       for (k in 1:Taille_ligne) {
+         value_Y[k] = data[k,j]
+         valeur_M[[2]][k] = value_Y[k]
+       }
+     }
+   }
+   cle <- c("X","Y","n")
+   keyval(cle,valeur_M) #on renvoie une cle est une liste comme valeur
+ }
> |

```

2.2.3 Fonction Reduce pour l'ajustement affine sous R

Compte tenu de la dépendance des résultats finaux à certains opérations sous-jacents, nous allons d'abord définir certaines fonctions, notamment la fonction moyenne de x, de y ; la fonction variance x et la fonction covariance de xy.

```
#-----FUNCTION REDUCE

***Defintion des fonctions

#Fonction moyenne de X
moy_x <- function(valeur_M){
  som_x = 0
  for (a in 1:valeur_M[[3]]) {
    som_x = som_x + valeur_M[[1]][a]
  }
  #print(som_x/valeur_M[[3]])
  return(som_x/valeur_M[[3]])
}

#Fonction moyenne de Y
moy_y <- function(valeur_M){
  som_y = 0
  for (b in 1:valeur_M[[3]]) {
    som_y = som_y + valeur_M[[2]][b]
  }
  #print(som_y/valeur_M[[3]])
  return(som_y/valeur_M[[3]])
}

#Fonction variance de X
var_x <- function(valeur_M){
  som_car_x = 0
  for (c in 1:valeur_M[[3]]) {
    som_car_x = som_car_x + valeur_M[[1]][c]*valeur_M[[1]][c]
  }
  #print(som_car_x)
  varia_x = som_car_x/valeur_M[[3]] - moy_x(valeur_M)*moy_x(valeur_M)
  #print(varia_x)
  return(varia_x)
}
```



```

> #Fonction moyenne de X
> moy_x <- function(valeur_M){
+   som_x = 0
+   for (a in 1:valeur_M[[3]]) {
+     som_x = som_x + valeur_M[[1]][a]
+   }
+   #print(som_x/valeur_M[[3]])
+   return(som_x/valeur_M[[3]])
+ }
> #Fonction moyenne de Y
> moy_y <- function(valeur_M){
+   som_y = 0
+   for (b in 1:valeur_M[[3]]) {
+     som_y = som_y + valeur_M[[2]][b]
+   }
+   #print(som_y/valeur_M[[3]])
+   return(som_y/valeur_M[[3]])
+ }
> #Fonction variance de X
> var_x <- function(valeur_M){
+   som_car_x = 0
+   for (c in 1:valeur_M[[3]]) {
+     som_car_x = som_car_x + valeur_M[[1]][c]*valeur_M[[1]][c]
+   }
+   #print(som_car_x)
+   varia_x = som_car_x/valeur_M[[3]] - moy_x(valeur_M)*moy_x(valeur_M)
+   #print(varia_x)
+   return(varia_x)
+ }
> #Fonction covariance de X
> cov_xy <- function(valeur_M){
+   som_xy = 0
+   for (d in 1:valeur_M[[3]]) {
+     som_xy = som_xy + valeur_M[[1]][d] * valeur_M[[2]][d]
+   }
+   #print(som_xy)
+   covariance = som_xy/valeur_M[[3]] - moy_y(valeur_M)*moy_x(valeur_M)
+   #print(covariance)
+   return(covariance)
+ }

#Fonction covariance de X
cov_xy <- function(valeur_M){
  som_xy = 0
  for (d in 1:valeur_M[[3]]) {
    som_xy = som_xy + valeur_M[[1]][d] * valeur_M[[2]][d]
  }
  #print(som_xy)
  covariance = som_xy/valeur_M[[3]] - moy_y(valeur_M)*moy_x(valeur_M)
  #print(covariance)
  return(covariance)
}

#**** Fonction reduce
reducer <- function(key, val.list){
  a.estim = cov_xy(valeur_M)/var_x(valeur_M)
  b.estim = moy_y(valeur_M) - a.estim*moy_x(valeur_M)
  #print(a.estim);print(b.estim)
  key = c("a","b")
  valeur = c(a.estim, b.estim)
  #print(key); print(valeur)
  keyval(key,valeur)
}

> #**** Fonction reduce
> reducer <- function(key, val.list){
+   a.estim = cov_xy(valeur_M)/var_x(valeur_M)
+   b.estim = moy_y(valeur_M) - a.estim*moy_x(valeur_M)
+   #print(a.estim);print(b.estim)
+   key = c("a","b")
+   valeur = c(a.estim, b.estim)
+   #print(key); print(valeur)
+   keyval(key,valeur)
+ }
> |

```



```
#_____Appel de la fonction MapReduce

data_BD <- to.dfs(data)
lienar <- mapreduce(input=data_BD,
                    map=mapper,
                    reduce = reducer
)
result <- from.dfs(lienar)
result$key[1];result$key[4];result$val[1];result$val[4]

> data_BD <- to.dfs(data)
> lienar <- mapreduce(input=data_BD,
+                     map=mapper,
+                     reduce = reducer
+ )
> result <- from.dfs(lienar)
> result$key[1];result$key[4];result$val[1];result$val[4]
[1] "a"
[1] "b"
[1] -1.221429
[1] 226.3571
> |
```

2.2.4 Illustration graphique

Le nuage de point issu de notre ajustement linéaire est donné par le graphique suivant :

