

Implémentation avec R

Thibault Laurent

06 octobre 2022

Contents

1 Début avec R	1
2 Analyse exploratoire (avec une approche non compositionnelle)	4
2.1 Espace des individus	4
2.2 Espace des variables	7
3 Traitement des zéros	11
3.1 Traitement des données manquantes	12
3.2 Traitement des zéros	13
4 Les données de composition sous R	14
4.1 Le diagramme ternaire	15
4.2 Le package compositions et la classe acomp	18
4.3 Les transformations alr , clr , ilr	27
4.4 Lois dans le simplexe	33
5 Analyse exploratoire (avec une approche compositionnelle)	41
5.1 Matrice de diagramme ternaire	41
5.2 Matrice de variation	42
5.3 Apprentissage automatique dans l'espace des variables	45
5.4 Apprentissage automatique dans l'espace des individus	46
5.5 Détection des atypiques	51
6 Régression compositionnelle	51
6.1 Régression avec variable explicative compositionnelle	51
6.2 Régression avec variable dépendante compositionnelle	55

R (R Core Team 2022) est un système qui est communément appelé langage et logiciel. Il permet, entre autres, de réaliser des analyses statistiques. Plus particulièrement, il comporte des moyens qui rendent possibles la manipulation des données, les calculs et les représentations graphiques. Dans ce chapitre, après un bref rappel des fonctionnalités de base de R, nous allons présenter les principales méthodes statistiques pour données de composition, implémentées sous formes de fonctions accessibles à travers des librairies diffusées via le CRAN (The Comprehensive R Archive Network) ou Github.

1 Début avec R

Dans un tableau de recueil de données de taille $n \times p$, les n unités statistiques sont en général disposées en lignes et les p variables en colonnes. Dans R, un tel tableau est stocké sous forme d'un objet communément appelé **data.frame**. Ce dernier peut aussi être vu comme une **list** avec p éléments, où chaque élément est un vecteur de taille n , composé de valeurs de type **numeric**, **character**, **logical**, etc. Ainsi, un **data.frame**

se distingue d'un objet de type `matrix` car dans cette classe d'objet, tous les éléments sont nécessairement du même type (`numeric` par exemple).

De nombreuses librairies permettent d'importer des fichiers de données sous R. L'utilisation de l'une plutôt que l'autre vient du type de fichiers à importer. Par exemple, les fichiers texte qui portent une extension `.txt` ou `.csv` s'importent très facilement avec la fonction de base `read.table()`. En revanche, les fichiers qui portent l'extension `.xls` ou `.xlsx` requièrent l'utilisation d'une librairie externe, par exemple la librairie `readxl` disponible sur le CRAN, via la fonction `install.packages()`. Dans ce chapitre, nous allons utiliser un certain nombre de librairies que nous décrirons au fur et à mesure, mais que nous allons installer en une seule fois :

```
install.packages(c(
  "compositions", # librairie dédiée à l'analyse de données de compositions
  "missForest", # traiter les données manquantes non composées
  "energy", # tester une loi normale multivariée
  "readxl", # importer des fichiers excel
  "sf", # données spatiales
  "tidyverse", # univers tidyverse
  "xtable", # exporter sous forme de table latex
  "easyCODA", # analyse univariee et multivariee pour CoDa
  "zCompositions", # traitement des valeurs manquantes
  "RColorBrewer" # palette de couleurs
))
devtools::install_github("tibo31/codareg")
```

Cette étape permet de télécharger et d'installer localement les bibliothèques de codes. Toutefois, pour pouvoir utiliser les fonctions incluses dans celles-ci, il faut les charger dans la session en cours par l'intermédiaire de la fonction `library()` :

```
library("readxl")
```

Pour importer le fichier qui contient les résultats du 1er tour de l'élection présidentielle française de 2022, on va d'abord télécharger le fichier localement depuis le site du ministère, à l'aide de la fonction `download.file()`.

```
my_url <- "https://www.data.gouv.fr/fr/datasets/r/48a38a25-9e46-4d83-80db-947258df9409"
download.file(my_url, destfile = paste0(getwd(), "/res_2022.xlsx"))
```

Ensuite, on importe le fichier sous R à l'aide de la fonction `read_excel()`.

```
res_2022 <- read_excel("res_2022.xlsx")
class(res_2022)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

L'objet importé est de type `tibble`, qui appartient à l'univers `tidyverse` dont les principales bibliothèques (`ggplot2`, `dplyr`, `purr`, etc.) ont pour but de simplifier la syntaxe et les temps de calcul de fonctions de bases de R. Cependant, un objet `tibble` a hérité de tous les spécificités de la classe `data.frame`, y compris les fonctions de base qui s'appliquent sur ce type d'objet. Parmi ces fonctions qui permettent de décrire l'objet, la fonction `str()` donne des informations concernant la dimension du tableau de données, le nom, le type et l'affichage des premières valeurs observées pour chaque variable.

```
str(res_2022)

## #tibble [107 x 32] (S3: tbl_df/tbl/data.frame)
## $ DEP_NOM           : chr [1:107] "Ain" "Aisne" "Allier" "Alpes-de-Haute-Provence" ...
## $ DEP_CODE          : chr [1:107] "01" "02" "03" "04" ...
## $ Inscrits          : num [1:107] 438109 373544 249991 128075 113519 ...
## $ Abstentions       : num [1:107] 97541 101089 58497 29290 25357 ...
## $ Votants           : num [1:107] 340568 272455 191494 98785 88162 ...
```

```

## $ Blancs      : num [1:107] 5641 3767 3749 1478 1395 ...
## $ Nuls        : num [1:107] 1903 2828 1790 624 532 ...
## $ Exprimés    : num [1:107] 333024 265860 185955 96683 86235 ...
## $ Macron_VOIX : num [1:107] 92206 58721 49706 20800 20507 ...
## $ Macron_EXP  : num [1:107] 27.7 22.1 26.7 21.5 23.8 ...
## $ Le_Pen_VOIX : num [1:107] 86755 104342 50315 26010 19696 ...
## $ Le_Pen_EXP  : num [1:107] 26.1 39.2 27.1 26.9 22.8 ...
## $ Mélenchon_VOIX : num [1:107] 57832 41172 31013 21856 19718 ...
## $ Mélenchon_EXP : num [1:107] 17.4 15.5 16.7 22.6 22.9 ...
## $ Zemmour_VOIX : num [1:107] 27530 18266 12361 7926 6164 ...
## $ Zemmour_EXP  : num [1:107] 8.27 6.87 6.65 8.2 7.15 14 7.21 6.55 6.35 7.6 ...
## $ Pécresse_VOIX : num [1:107] 17572 10920 10319 3834 4511 ...
## $ Pécresse_EXP : num [1:107] 5.28 4.11 5.55 3.97 5.23 5.59 4.85 4.15 2.97 5.96 ...
## $ Jadot_VOIX   : num [1:107] 15843 7074 5982 3957 5013 ...
## $ Jadot_EXP    : num [1:107] 4.76 2.66 3.22 4.09 5.81 4.18 4.34 2.57 3.29 3.09 ...
## $ Lassalle_VOIX : num [1:107] 10876 6468 7782 4309 3871 ...
## $ Lassalle_EXP  : num [1:107] 3.27 2.43 4.18 4.46 4.49 2.28 4.59 3.05 8.21 2.53 ...
## $ Roussel_VOIX : num [1:107] 5938 5968 8119 2721 1925 ...
## $ Roussel_EXP   : num [1:107] 1.78 2.24 4.37 2.81 2.23 1.59 2.9 2.25 2.95 2.07 ...
## $ Dupont-Aignan_VOIX: num [1:107] 8998 5790 4216 2504 2142 ...
## $ Dupont-Aignan_EXP : num [1:107] 2.7 2.18 2.27 2.59 2.48 2.38 2.5 2.21 1.77 2.65 ...
## $ Hidalgo_VOIX   : num [1:107] 5644 2983 3280 1396 1459 ...
## $ Hidalgo_EXP    : num [1:107] 1.69 1.12 1.76 1.44 1.69 0.97 2.13 1.3 3.5 1.15 ...
## $ Poutou_VOIX   : num [1:107] 2172 2118 1503 865 801 ...
## $ Poutou_EXP    : num [1:107] 0.65 0.8 0.81 0.89 0.93 0.53 0.92 0.83 0.79 0.71 ...
## $ Arthaud_VOIX : num [1:107] 1658 2038 1359 505 428 ...
## $ Arthaud_EXP   : num [1:107] 0.5 0.77 0.73 0.52 0.5 0.29 0.62 0.81 0.45 0.67 ...

```

On observe que le jeu de données contient 107 observations : 96 départements de métropole, 10 départements d'Outre-Mer et enfin, une observation comprend les français résidant à l'étranger. Les variables contiennent le nombre d'inscrits (personnes ayant le droit de vote) qui est égale au nombre d'abstentions (personnes n'ayant pas voté) plus le nombre de votants (personne ayant voté). Le nombre de votants est égale à la somme du nombre de votes nuls ou blancs, plus le nombre d'exprimés (personnes ayant voté pour un des candidats). Enfin pour chacun des 12 candidats (Macron, Le Pen, Mélenchon, Zemmour, Pécresse, Jadot, Lassalle, Roussel, Dupont-Aignan, Hidalgo, Poutou, Arthaud), on observe d'une part le nombre de voix obtenu et d'autre part le rapport "nombre de voix" sur "nombre d'exprimés."

Ainsi, une première façon de construire des données de composition est de considérer le rapport "nombre de voix" sur "nombre d'exprimés" des 12 candidats, en ne tenant pas compte des votes nuls, blancs ni de l'abstention. C'est de cette façon que les votes sont comptabilisés. Pour construire cet objet, il suffit d'extraire les colonnes correspondantes dans `res_2022` de la façon suivante :

```

vote_share_1 <- res_2022[ , c("Macron_EXP", "Le_Pen_EXP", "Mélenchon_EXP",
  "Zemmour_EXP", "Pécresse_EXP", "Jadot_EXP", "Lassalle_EXP", "Roussel_EXP",
  "Dupont-Aignan_EXP", "Hidalgo_EXP", "Poutou_EXP", "Arthaud_EXP")]

```

On simplifie le nom des variables :

```

colnames(vote_share_1) <- c("Macron", "Le_Pen", "Mélenchon", "Zemmour",
  "Pécresse", "Jadot", "Lassalle", "Roussel", "Dupont_Aignan", "Hidalgo",
  "Poutou", "Arthaud")

```

Une deuxième façon de construire les données de composition est d'inclure une 13ème variable correspondant à la somme des votes blanc, des votes nuls et l'abstention et de considérer à présent le rapport "nombre de voix" sur "nombre d'inscrits" (au lieu de "nombre de voix" sur "nombre d'exprimés"). Pour faire cela, on crée dans un premier temps cette nouvelle variable `non_exprimés`:

```
res_2022$non_exprimés <- res_2022$Abstentions + res_2022$Blancs +
  res_2022$Nuls
```

On sélectionne les variables de comptage:

```
vote_share_2 <- res_2022[ , c("Macron_VOIX", "Le_Pen_VOIX",
  "Mélenchon_VOIX", "Zemmour_VOIX", "Pécresse_VOIX", "Jadot_VOIX",
  "Lassalle_VOIX", "Roussel_VOIX", "Dupont-Aignan_VOIX", "Hidalgo_VOIX",
  "Poutou_VOIX", "Arthaud_VOIX", "non_exprimés")]
```

On divise par le nombre d'inscrits en utilisant la fonction `sapply()`:

```
vote_share_2 <- sapply(vote_share_2, function(x) x / res_2022$Inscrits)
```

On simplifie le nom des variables :

```
colnames(vote_share_2) <- c("Macron", "Le_Pen", "Mélenchon", "Zemmour",
  "Pécresse", "Jadot", "Lassalle", "Roussel", "Dupont_Aignan", "Hidalgo",
  "Poutou", "Arthaud", "non_inscrits")
```

Exercice 1 : importer le jeu de données de votre choix sous R qui contient des données de composition. A défaut, vous pourrez utiliser un des jeux de données inclus dans le package `compositions` et dont on peut afficher la liste de la façon suivante :

```
data(package = "compositions")
```

2 Analyse exploratoire (avec une approche non compositionnelle)

Dans cette section, nous proposons d'utiliser dans un premier les fonctions de base de R pour décrire nos données. Dans la section suivante, nous utiliserons des outils spécifiques aux données de composition.

2.1 Espace des individus

Dans un premier temps, on va essayer de représenter la répartition des votes département par département. Une première façon de faire est de faire soit un diagramme en barre, soit un graphique des parts. Pour cela, on va utiliser le package `ggplot2` qui appartient au consortium `tidyverse`. Il faut dans un premier temps rendre les données `tidy`, dans le sens où le jeu de données doit avoir les colonnes suivantes : une colonne pour le nom des départements, une colonne pour le nom des candidats et enfin une colonne pour le score obtenu par le couple département/candidat. On réalise cette opération à l'aide de la fonction `pivot_longer()` (car on transforme les données dans le format long) :

```
library("tidyverse")
vote_share_long <- pivot_longer(data.frame(vote_share_1,
  dep = res_2022$DEP_NOM),
  cols = 1:12, names_to = "candidat", values_to = "share")
```

Ensuite, on représente les barres en utilisant l'univers `ggplot2`. Par défaut, les observations sont ordonnées dans l'ordre alphabétique. Il peut être intéressant d'ordonner les départements selon un critère géographique en regroupant par exemple les départements par région, ou bien selon un critère statistique en ordonnant les départements selon le score obtenu par un des candidats.

Ici, on ordonne d'abord les départements en fonction du score obtenu par le candidat Zemmour et on représente ensuite le diagramme en barre.

```
vote_share_long %>%
  mutate(dep = factor(dep, levels =
    levels((vote_share_long %>%
```

```

filter(candidat == "Zemmour") %>%
  mutate(dep = fct_reorder(dep, share)))$dep)) %>%
ggplot() +
  geom_col(aes(x = dep, fill = candidat, y = share)) +
  theme(axis.text.x = element_text(angle = 90))

```

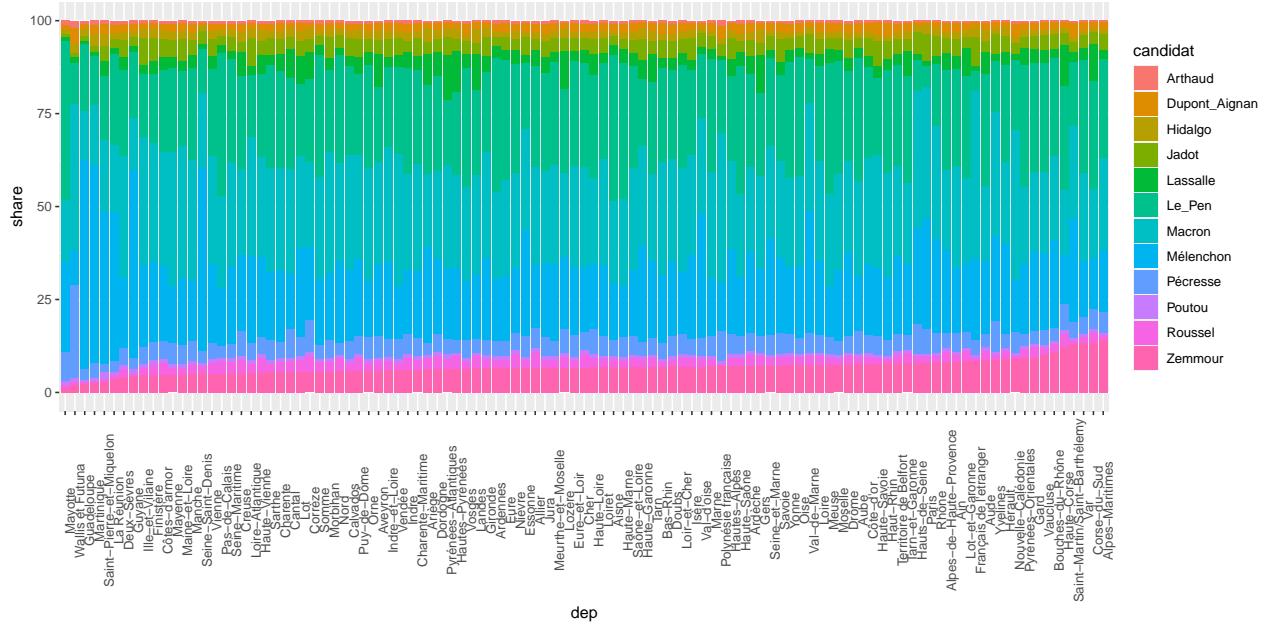


Figure 1: Diagramme en barre des votes (nombre de voix divisé par nombre d'exprimés) par département

Remarque : la palette de couleurs utilisée dans `ggplot2` attribue des couleurs avec une intensité/saturation plus ou moins égale pour chaque couleur. Seule la teinte varie, le but étant que la perception du lecteur ne soit pas influencée par une couleur plus qu'une autre.

On va récupérer les données géographiques associées aux départements, ce qui nous permettra d'une part de faire des cartes un peu plus tard, mais de classer aussi les départements en fonction de leur région d'appartenance, puisque cette information n'était pas disponible avec les données du Ministère.

On ne va expliciter ici le code utilisé, ce dernier est téléchargeable sur la page github suivante :

```

library("sf")
source("spatial/spatial_circon.R")

```

Ce code nous permet de construire l'objet `geo_dep` de classe `sf` (Pebesma 2018) qui permet de faire de l'analyse de données spatiales. On va dans un premier temps faire la jointure de cet objet avec les données d'élection :

```

geo_dep <- merge(geo_dep[, c("nom_dpt", "code_dpt", "nom_reg")],
  data.frame(vote_share_1, dep = res_2022$DEP_CODE,
             Inscrits = res_2022$Inscrits),
  by.x = "code_dpt", by.y = "dep")

vote_share_long <- pivot_longer(geo_dep, cols = 4:15, names_to = "candidat",
                                 values_to = "share")
vote_share_long %>%
  mutate(nom_reg = gsub("-", "\n", nom_reg)) %>%
  ggplot() +

```

```
geom_col(aes(x = nom_dpt, fill = candidat, y = share)) +
theme(axis.text.x = element_text(angle = 90)) +
facet_grid(. ~ nom_reg, scales = "free_x", space = "free")
```

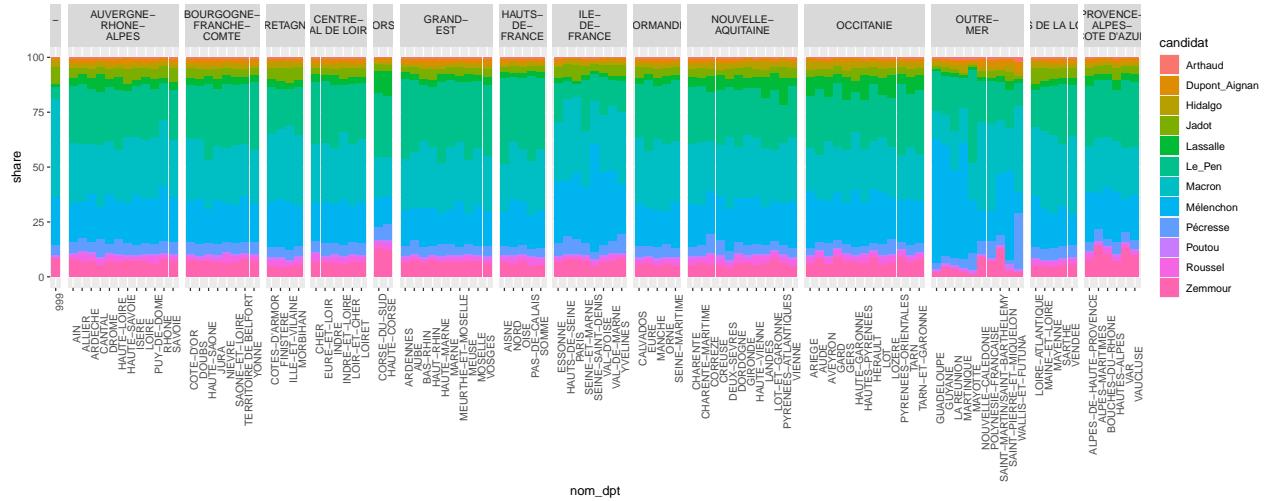


Figure 2: Diagrammes en barre des votes par département, groupés par région

On remarque qu'il existe des similitudes entre les départements d'une même région (Occitanie, Nouvelle-Aquitaine). De même, on constate des différences entre régions (l'Ile de France est très différente du Grand-Est par exemple). Toutefois ce n'est pas facile de comparer les barres entre elles car il y a trop de composantes et trop d'observations...

Pour illustrer le diagramme des parts, on a récupéré les résultats du 1er tour de l'élection présidentielle depuis l'adoption de la 5ème république. Les partis ont été agrégés en 4 groupe : droite, gauche, centre et extrême-droite.

```
time_chart <- data.frame(
  year = rep(as.Date(c("1958-01-01", "1965-01-01", "1969-01-01", "1974-01-01",
    "1981-01-01", "1988-01-01", "1995-01-01", "2002-01-01", "2007-01-01",
    "2012-01-01", "2017-01-01", "2022-01-01")), each = 4),
  vote = c(78.51, 21.49, 0, 0, 44.65, 31.72, 17.28, 6.35, 44.47, 32.22,
    23.31, 0, 35.77, 47.96, 15.11, 1.16, 20.99, 50.70, 28.31, 0,
    19.95, 49.11, 16.55, 14.39, 20.84, 40.84, 18.58, 19.74, 19.88,
    42.87, 12.63, 24.62, 31.18, 36.10, 18.57, 14.15, 27.18, 44.00,
    9.13, 19.69, 20.01, 27.85, 25.22, 26.92, 4.78, 31.92, 30.98, 32.32),
  parti = rep(c("droite", "gauche", "centre", "extrême"), 12)
)
```

Le graphique s'obtient à l'aide du package `ggplot2`:

```
ggplot(time_chart) +
  aes(x = year, y = vote, fill = parti) +
  geom_area(color = "black") +
  labs(title = "Votes au 1er tour de l'élection présidentielle",
    subtitle = "1958 à 2022",
    x = "Year",
    y = "percentage",
    fill = "Partis") +
  scale_fill_brewer(palette = "Set2") +
  theme_minimal()
```

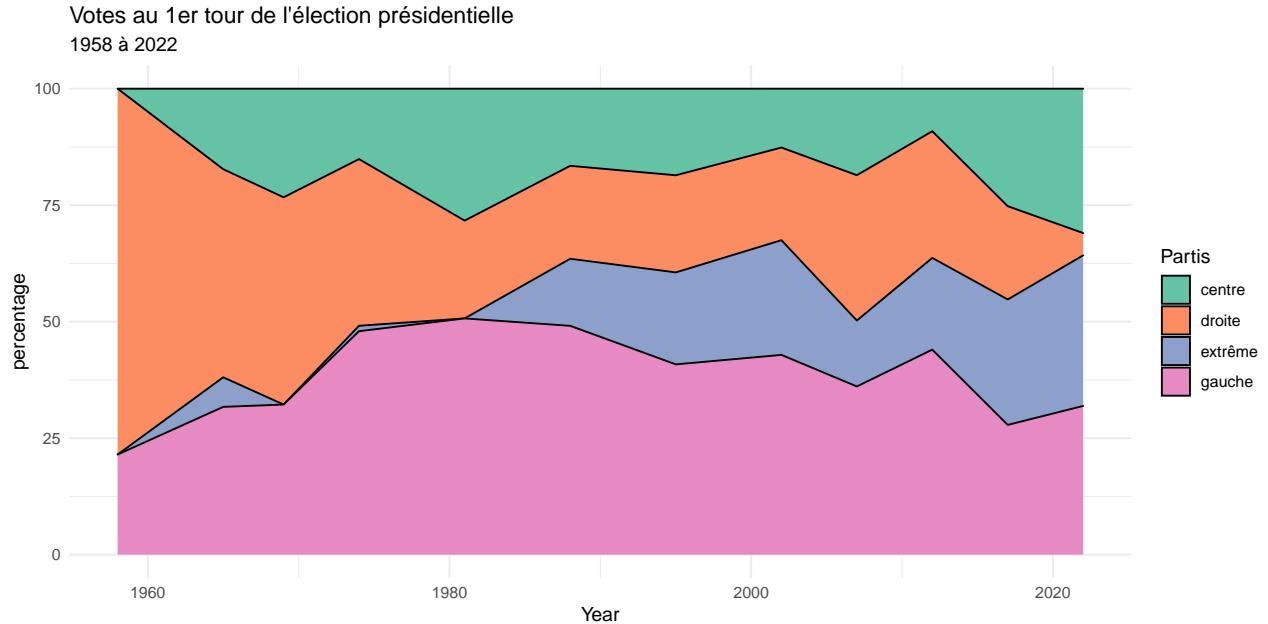


Figure 3: Evolution des parts des principaux partis entre 1958 et 2022

Ce graphique nous permet d'observer la perte de vitesse des partis de gauche et de droite alors que le centre et l'extrême-droite ont gagné du terrain.

Enfin, on va terminer cette section en réalisant la carte des vainqueurs par département. Pour faire cela, on détermine le candidat qui a obtenu le plus grand nombre de voix par département.

```
geo_dep$vainqueur <- names(vote_share_1) [
  apply(st_drop_geometry(geo_dep[, names(vote_share_1)]), 1, which.max)]
plot(geo_dep[, "vainqueur"], main = "", key.pos = 1,
  key.width = lcm(1.3), key.length = .7)
```

On voit se dessiner un effet géographique extrêmement important dans la préférence des français pour un candidat plutôt qu'un autre.

2.2 Espace des variables

On peut considérer chaque composante comme étant une variable quantitative et utiliser les outils usuels de la statistique exploratoire, comme calculer les indicateurs du minimum, maximum, moyenne, médiane, quartiles, etc. Ces indicateurs s'obtiennent simultanément à l'aide de la fonction `summary()` :

```
xtable::xtable(t(sapply(vote_share_1, function(x)
  c(min = min(x), q = quantile(x, 0.25), median = median(x), mean = mean(x),
    q = quantile(x, 0.75), max= max(x)))),
  caption = "Statistique descriptive des parts de vote")
```

Afin de comparer les variables entre elles, on peut également représenter une boîte à moustache par variable.

```
ggplot(vote_share_long) +
  geom_boxplot(aes_string("candidat","share"))+
  theme_bw()
```

On notera également la fonction `DOT()` du package `easyCODA` (Greenacre 2018) qui permet de représenter un "dot chart," c'est-à-dire un point par observation et par variable.

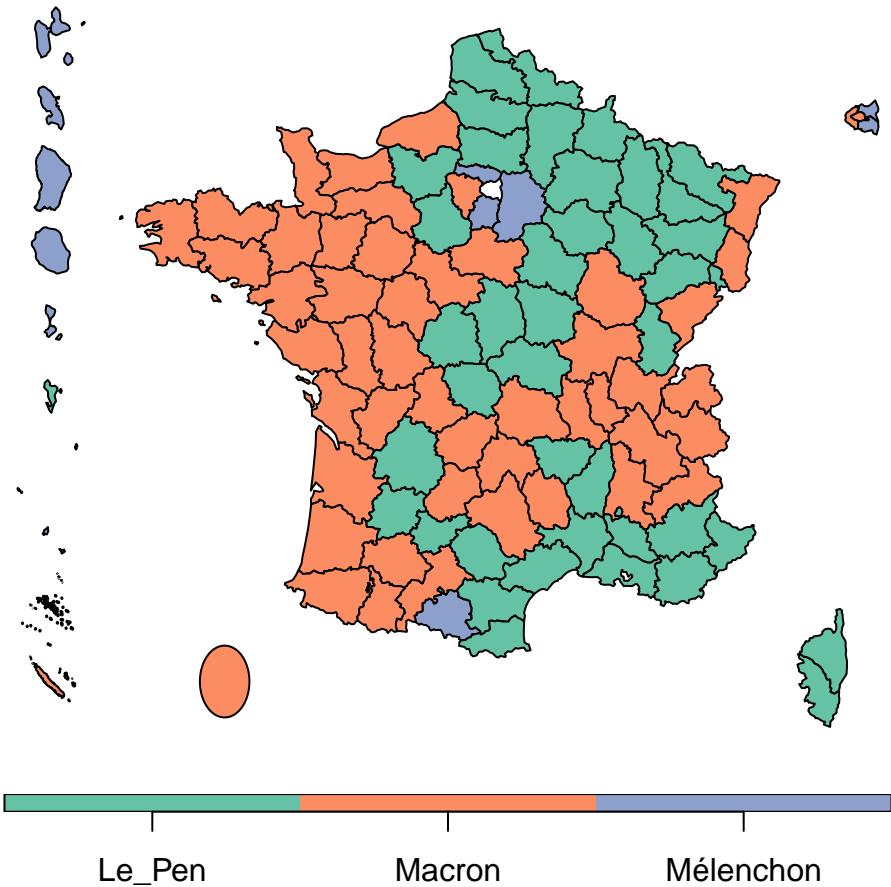


Figure 4: Cartographie des candidats qui ont remporté le plus de voix par département

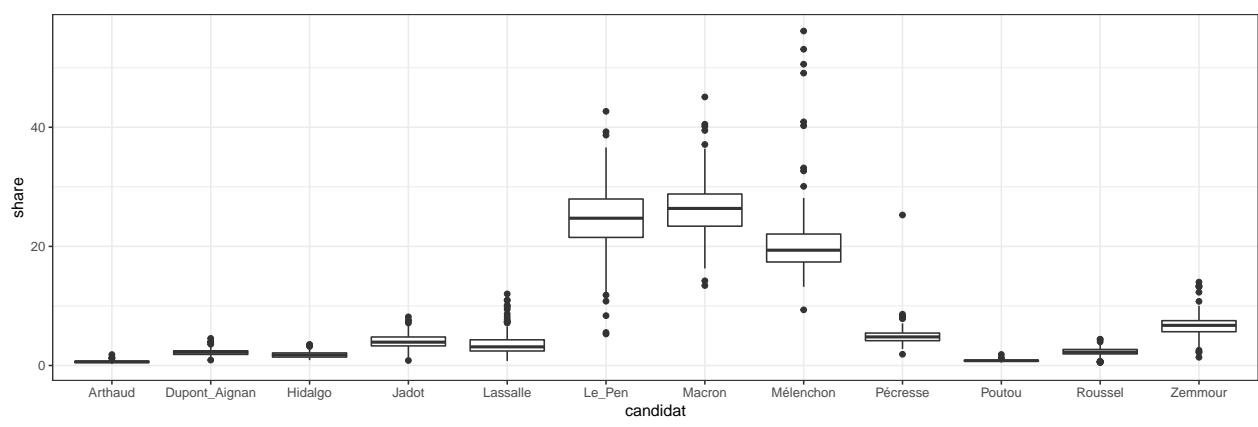


Figure 5: Boîte à moustache des parts prises une à une

	min	q.25%	median	mean	q.75%	max
Macron	13.43	23.39	26.37	26.66	28.80	45.09
Le_Pen	5.29	21.50	24.73	24.48	27.95	42.68
Mélenchon	9.35	17.38	19.36	21.25	22.07	56.16
Zemmour	1.38	5.67	6.74	6.85	7.53	14.00
Pécresse	1.89	4.17	4.80	5.06	5.45	25.27
Jadot	0.84	3.29	3.92	4.11	4.79	8.17
Lassalle	0.76	2.44	3.13	3.80	4.31	12.02
Roussel	0.49	1.94	2.25	2.28	2.70	4.41
Dupont_Aignan	0.91	1.86	2.22	2.25	2.47	4.55
Hidalgo	0.91	1.42	1.77	1.82	2.12	3.52
Poutou	0.48	0.72	0.81	0.82	0.90	1.85
Arthaud	0.26	0.48	0.63	0.63	0.73	1.83

Table 1: Statistique descriptive des parts de vote

```
library(easyCODA)
DOT(vote_share_1)
```

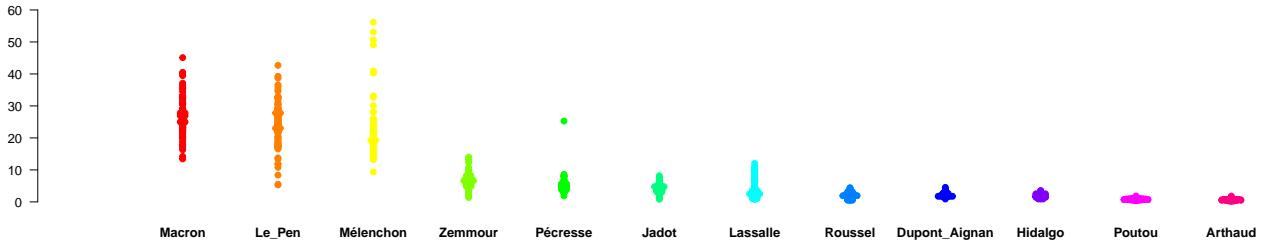


Figure 6: Dot chart des variables prises une à une

Parmi les outils présentés dans le chapitre 4, on présente également le graphique de confiance de 95% :

```
plotdata <- vote_share_long %>%
  st_drop_geometry() %>%
  select(candidat, share) %>%
  group_by(candidat) %>%
  summarize(mean = mean(share),
            ci_1 = quantile(share, 0.05),
            ci_2 = quantile(share, 0.95))
ggplot(plotdata, aes(x = candidat,
                      y = mean,
                      colour = candidat)) +
  geom_point(size = 3) +
  geom_line(size = 1) +
  geom_errorbar(aes(ymin = ci_1,
                    ymax = ci_2),
                width = .1)
```

On peut finalement représenter carte par carte, le score réalisé par chaque parti. Ici on utilise la librairie `maps` (Giraud 2022) pour faire une cartographie des scores réalisés par les trois premiers candidats. On utilise une palette de couleur différente selon le candidat; en revanche, on garde le même intervalle de discrétilisation pour chaque carte, afin que la luminosité soit la même d'une carte à une autre.

```
library("maps")
par(mfrow = c(1, 3), oma = c(0, 0, 0, 0), mar = c(0, 0, 0, 0))
```

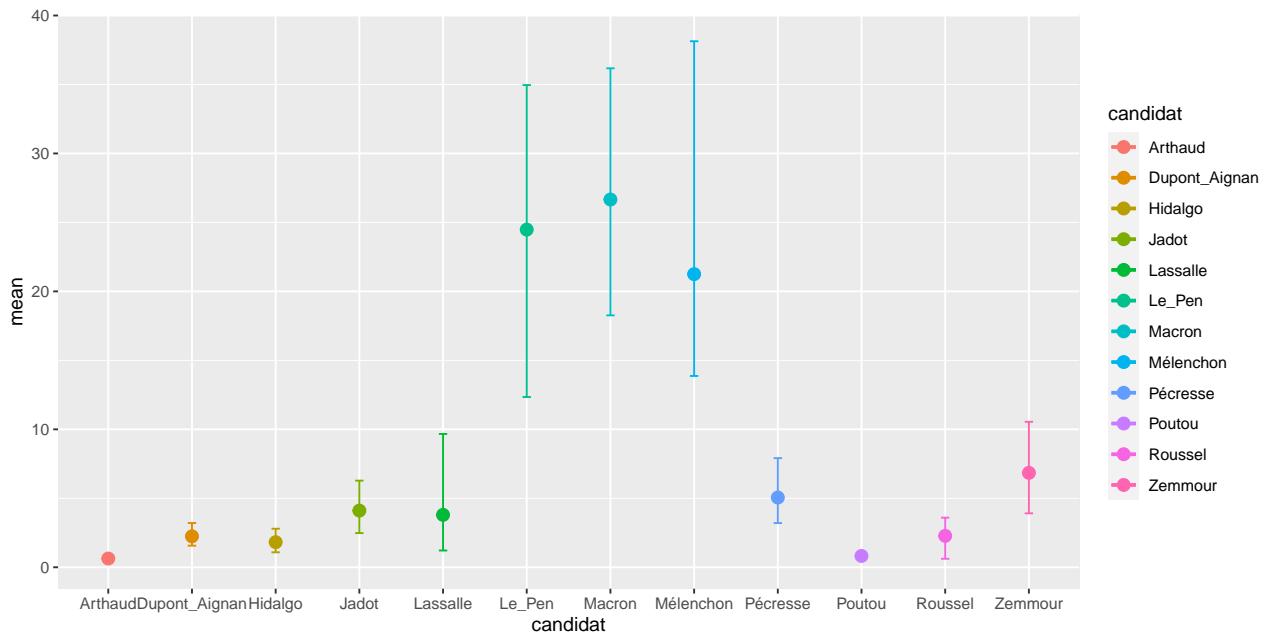


Figure 7: Graphique de confiance

```

candidat <- c("Macron", "Le_Pen", "Mélenchon")
ma_palettes <- c("OrYel", "Dark Mint", "Reds")
for (i in 1:3) {
  mf_map(x = geo_dep, var = candidat[i], type = "choro",
    pal = ma_palettes[i],
    breaks = c(5, 15, 20, 22.5, 25, 27.5, 30, 45, 60),
    leg_title = candidat[i],
    leg_val_rnd = 2)
}
  
```

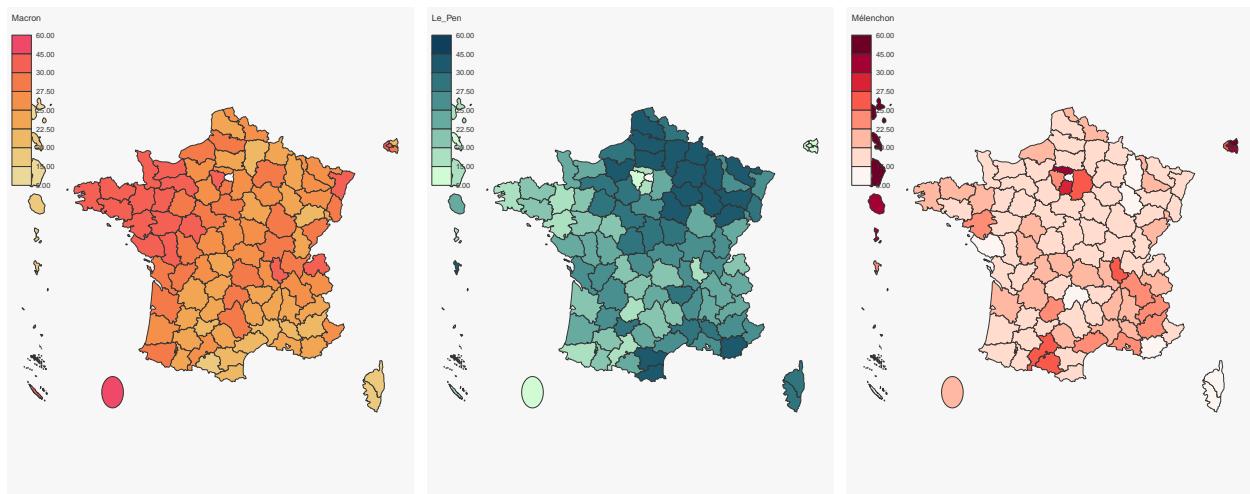


Figure 8: Cartographie des parts obtenus par les trois premiers candidats

Dans cette section, on a vu que l'analyse univariée pouvait très bien s'appliquer sur chaque composante des données de composition. En revanche, utiliser les outils usuels de la statistique multivariée introduit

nécessairement un malaise dû à la nature des données de composition. A titre d'exemple, nous avons représenté les nuages de point entre le score obtenu par Macron et celui obtenu par Le Pen, en considérant trois mesures différentes : candidats dans les deux jeux de données (celui avec uniquement le score des candidates et celui avec le score des candidats ainsi que le nombre d'abstentions).

```
par(mfrow = c(1, 3), oma = c(0, 0, 0, 0), mar = c(4, 4, 1, 1))
plot(Macron_VOIX ~ `Le Pen_VOIX`, data = res_2022,
     main = "Nombre de voix")
plot(Macron_EXP ~ `Le Pen_EXP`, data = res_2022,
     main = "Nombre de voix / nombre exprimés")
plot(Macron ~ Le_Pen, data = vote_share_2,
     main = "Nombre de voix / nombre inscrits")
```

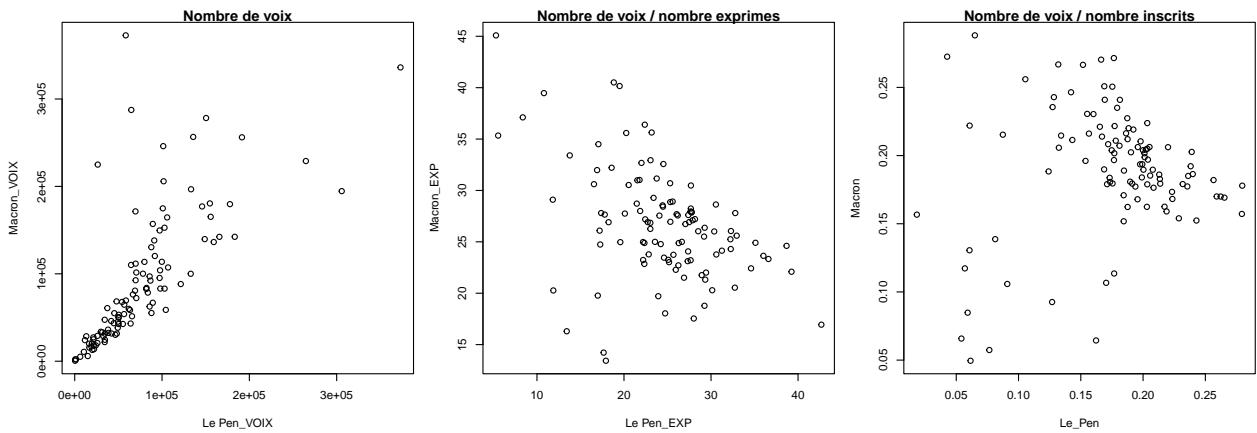


Figure 9: Nuage de points des voix obtenus par Macron et Le Pen en valeurs absolue (sur la gauche), en part d'exprimés (au centre) et en part d'inscrits (sur la droite)

On arrive à des observations très différentes : dans le premier graphique, on observe un effet positif dû à l'effet “nombre” (plus des départements sont peuplés, plus les scores des deux candidats seront élevés), alors que dans les deuxième et troisième graphique, l'effet est négatif dans les deux cas; cependant, il est difficile d'interpréter cette corrélation négative car les scores des deux candidats sont intrinsèquement liés dû à la nature même des données (une variable pouvant être exprimée en fonction de l'autre à cause de la clôture); enfin, dans le troisième graphique, il semble y avoir deux groupes (le plus petit étant probablement un groupe d'outliers) semblent identifiés. Nous avons un exemple de “spurious correlation” et de biais négatif, propre aux données de composition, qui est une des raisons ayant motivé l'introduction de la géométrie d'Aitchison, pour rendre les interprétations “cohérentes” entre les différentes sous-compositions.

Exercice 2 : utiliser les outils présentés dans cette section sur le jeu de données que vous avez choisi.

3 Traitement des zéros

Cette section illustre les principales méthodes d'imputation présentées dans le Chapitre 5. Le package utilisé est **zcompositions** (Palarea-Albaladejo and Martin-Fernandez 2015). On va reprendre le même jeu de données nommé **LPdataZM** utilisé dans le chapitre 5.

```
library(zCompositions)
data(LPdataZM)
```

Dans R, on distingue généralement les zéros des valeurs manquantes qui sont codées par **NA** (non available). On peut traiter les zéros séparément des valeurs manquantes si on estime que la cause des zéros est différente de celle des valeurs manquantes. Par exemple, un zéro peut être dû à une présence tellement faible d'une composition qu'on n'a pas réussi à la mesurer, alors qu'une valeur manquante peut être due à une erreur de

saisie.

Dans un premier temps, on va s'intéresser aux données manquantes que nous allons traiter comme des données non compositionnelles avant de s'intéresser au problème des zéros.

3.1 Traitement des données manquantes

On peut représenter les différentes configurations des valeurs manquantes à l'aide de la fonction `zPatterns()`. Pour chaque ligne i de la matrice imprimée, une cellule (i, j) est coloriée pour indiquer que la variable j est manquante dans cette configuration. Ainsi, dans l'exemple ci-après, la première ligne correspond à la configuration où aucune variable n'est manquante, alors que la seconde ligne correspond à la configuration où la variable Rb est manquante. La barre affichée sur chaque ligne i correspond au nombre d'observations incluses dans la configuration i . Dans l'exemple ci-après, 65% des observations appartiennent à la 1ère configuration, autrement dit, 65% des observations n'ont aucune valeur manquante. Enfin, la barre associée à une colonne j indique le nombre de valeurs manquantes observé dans la composante j . Dans cet exemple, on constate que seules les composantes Ni et Ba n'ont pas de valeurs manquantes alors que la composante Y est celle qui en présente le plus.

```
zPatterns(LPdataZM, label = NA)
```

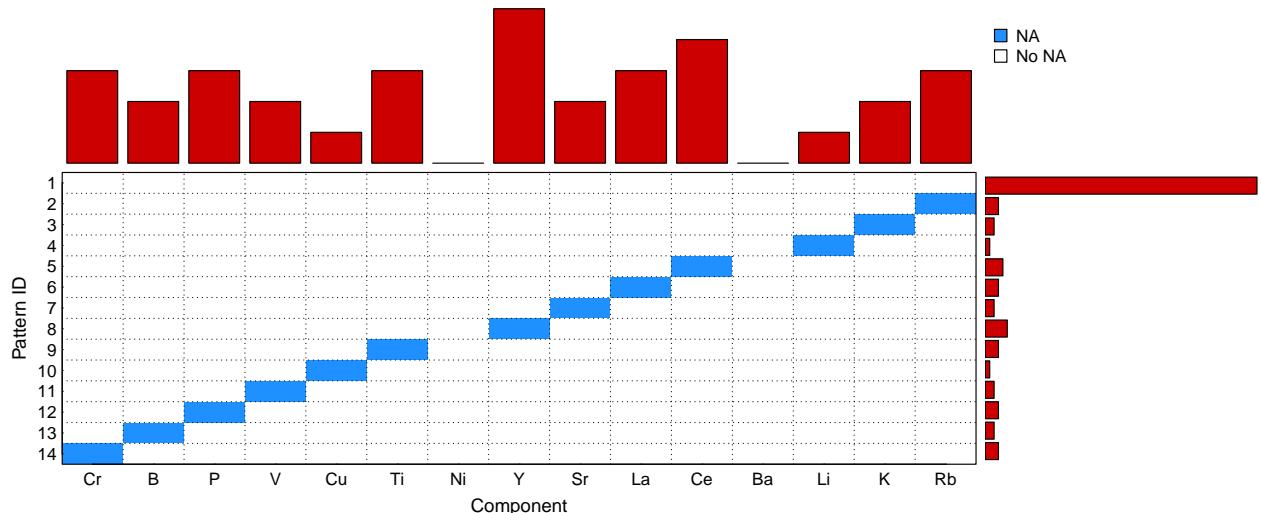


Figure 10: Configuration des valeurs manquantes par individu et variable

```
## Patterns ('+' means NA, '-' means No NA)
##
##   Patt.ID Cr B P V Cu Ti Ni Y Sr La Ce Ba Li K Rb No.Unobs Patt.Perc
##      1   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 0    64.58
##      2   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + 1    3.12
##      3   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - 1    2.08
##      4   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - - 1    1.04
##      5   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - - 1    4.17
##      6   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - - 1    3.12
##      7   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - - 1    2.08
##      8   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - + - - 1    5.21
##      9   - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1    3.12
##     10  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1    1.04
##     11  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1    2.08
##     12  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1    3.12
##     13  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - 1    2.08
```

```

##          14 + - - - - - - - - - - - - - - - - - - - - 1      3.12
##
## Percentage cells by component
##   Cr   B   P   V   Cu   Ti   Ni   Y   Sr   La   Ce   Ba   Li   K   Rb
## 3.12 2.08 3.12 2.08 1.04 3.12 0.00 5.21 2.08 3.12 4.17 0.00 1.04 2.08 3.12
##
## Overall percentage cells: 2.36%

```

Il existe une large littérature sur le traitement de données manquantes et de nombreuses librairies R traitent le sujet (voir par exemple `missForest`, `DMwr`). Une première solution pour traiter les valeurs manquantes est d'utiliser des méthodes pour données non compositionnelles. Dans notre exemple, nous allons commencer par considérer les valeurs manquantes comme “Missing At Random” et les traiter sous leur forme non clôturées en appliquant une méthode d’imputation basée sur les forêts aléatoires :

```

require("missForest")

## Le chargement a nécessité le package : missForest
LPdataZM_nm <- missForest(LPdataZM)$ximp

```

3.2 Traitement des zéros

A présent, on représente les différentes configurations des zéros, et on ajoute dans le graphique les moyenne géométriques des compositions associées à chaque configuration avec l’option `show.means = TRUE`. Le nombre de zéros est trois fois plus importants que les valeurs manquantes. Par ailleurs, il existe des configurations où plusieurs composantes présentent plusieurs zéros à la fois (configuration 5 par exemple).

```

zPatterns(LPdataZM_nm, label = 0, show.means = TRUE)

```

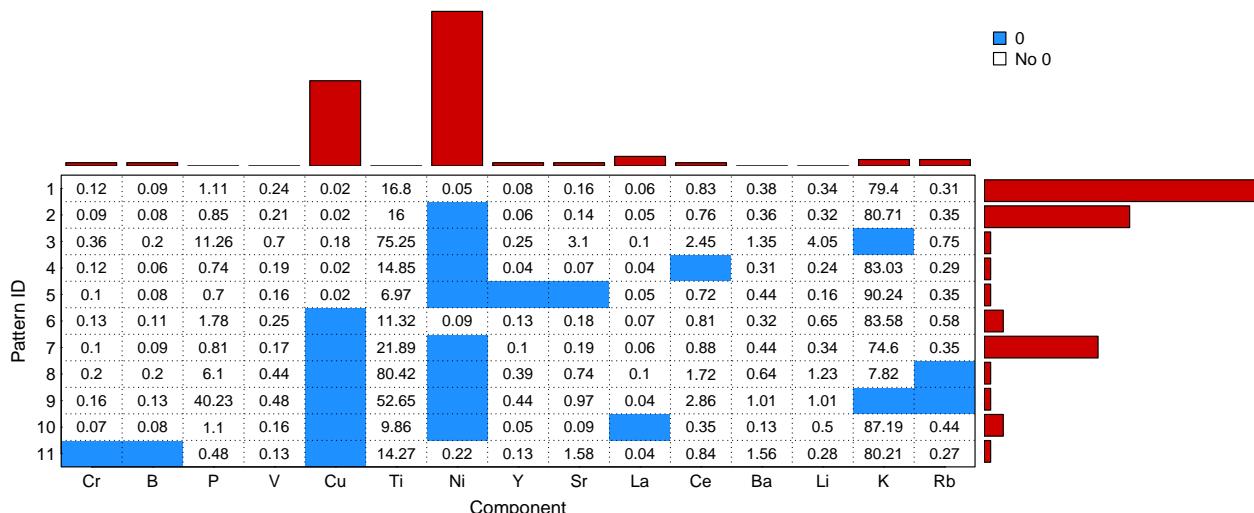


Figure 11: Configuration des zéros par individu et variable

```

## Patterns ('+' means 0, '-' means No 0)
##
##   Patt.ID Cr B P V Cu Ti Ni Y Sr La Ce Ba Li K Rb No.Unobs Patt.Perc
##    1      - - - - - - - - - - - - - - - - - - - - - - - 0       44.79
##    2      - - - - - - - + - - - - - - - - - - - 1       23.96
##    3      - - - - - - - + - - - - - - - - - + - - - 2        1.04
##    4      - - - - - - - + - - - - + - - - - - - - - 2        1.04
##    5      - - - - - - - + + + - - - - - - - - - - - 3        1.04

```

```

##      6 - - - + - - - - - - - - - - - - - - - - - - - 1  3.12
##      7 - - - + - + - - - - - - - - - - - - - - - - 2 18.75
##      8 - - - + - + - - - - - - - - - - - + 3  1.04
##      9 - - - + - + - - - - - - - - - + + 4  1.04
##     10 - - - + - + - - - + - - - - - - - 3  3.12
##     11 + + - - + - - - - - - - - - - - 3  1.04
##
## Percentage cells by component
##   Cr    B    P    V    Cu    Ti    Ni    Y    Sr    La    Ce    Ba    Li
## 1.04 1.04 0.00 0.00 28.12 0.00 51.04 1.04 1.04 3.12 1.04 0.00 0.00
##   K    Rb
## 2.08 2.08
##
## Overall percentage cells: 6.11%

```

Dans un premier temps, nous allons supprimer la colonne Ni qui contient plus de 50% de zéros.

```
LPdataZM_nm <- LPdataZM_nm[, !(names(LPdataZM_nm) %in% "Ni")]
```

A présent, nous allons utiliser les différentes méthodes présentées dans le chapitre 5. Pour l'imputation multiplicative simple (fonction `multRepl()`), nous avons besoin de déterminer pour chaque composante, le seuil de détection. Pour cela, nous avons tout simplement choisi la valeur minimum strictement supérieur à 0.

```
dl <- apply(LPdataZM_nm, 2, function(x) min(x[x != 0]))
LPdataZM_multRepl <- multRepl(LPdataZM_nm, label = 0, dl = dl)
```

Pour l'imputation multiplicative log-normale (fonction `multLN()`) :

```
LPdataZM_multLN <- multLN(LPdataZM_nm, label = 0, dl = dl)
```

Pour l'imputation par l'algorithme EM (fonction `lrEM()`) :

```
LPdataZM_lrEM <- lrEM(LPdataZM_nm, label = 0, dl = dl)
```

```
## No. iterations to converge: 14
```

Remarque : dans le package `robCompositions`, on notera les fonctions `impKNNa()`, `impCoda()` qui permettent également ‘de faire de l’imputation

4 Les données de composition sous R

En 2022, environ 18000 librairies étaient disponibles via le CRAN. Parmi ce nombre volumineux et toujours croissant de librairies, une vingtaine seulement contient les mots clés **compositional data**. Deux bibliothèques en particulier contiennent l’essentielle des méthodes statistiques appliquées aux données de composition. Il s’agit de :

- `compositions` (van den Boogaart, Tolosana-Delgado, and Bren 2022)
- `robCompositions` (Filzmoser, Hron, and Templ 2018)

Nous allons uniquement utiliser la première librairie, dont le principal atout est la création de la classe d’objet `acomp`, sur laquelle on pourra appliquer les opérations géométriques dans l’espace du simplexe. La seconde bibliothèque implémente des méthodes statistiques (imputation, analyse multidimensionnelles) robustes (c’est-à-dire peu sensibles aux valeurs extrêmes ou aberrantes) appliquées aux données de composition, mais nous ne l’utiliserons pas dans ce document. Enfin, nous ferons appel à la librairie `ggtern` (Hamilton and Ferry 2018) pour se rapprocher de la philosophie `ggplot2` afin de représenter les diagrammes ternaires.

4.1 Le diagramme ternaire

La fonction `ggtern()` du même package `ggtern` (Hamilton and Ferry 2018) permet de représenter un diagramme ternaire en quelques lignes de codes. Pour cela, il suffit de spécifier le nom des trois composantes à représenter dans la fonction `aes()`. On réfère le lecteur au chapitre 2 pour une bonne compréhension du diagramme ternaire. Dans l'exemple suivant, on a représenté le score des trois principaux candidats. On remarque que la fonction se charge de faire la clôture des trois sous-composantes qu'on lui a fourni. Il est possible de représenter sur le diagramme ternaire des informations supplémentaires, en utilisant par exemple des couleurs pour spécifier les niveaux d'une variable qualitative (argument `colour`) ou en utilisant des points de tailles proportionnelles à une variable quantitative (argument `size`). Pour illustrer cette fonction, nous avons représenté successivement les données brutes, avec une couleur différente selon le gagnant du département et enfin, des tailles de points proportionnelles au nombre d'inscrits.

```
library("ggtern")
p1 <- ggtern(data = vote_share_1, mapping = aes(x = Macron,
                                                 y = Le_Pen, z = Mélenchon)) +
  geom_point(size = 1.5)
p2 <- p1 + theme_rgbw()
grid.arrange(p1, p2, ncol = 2)
```

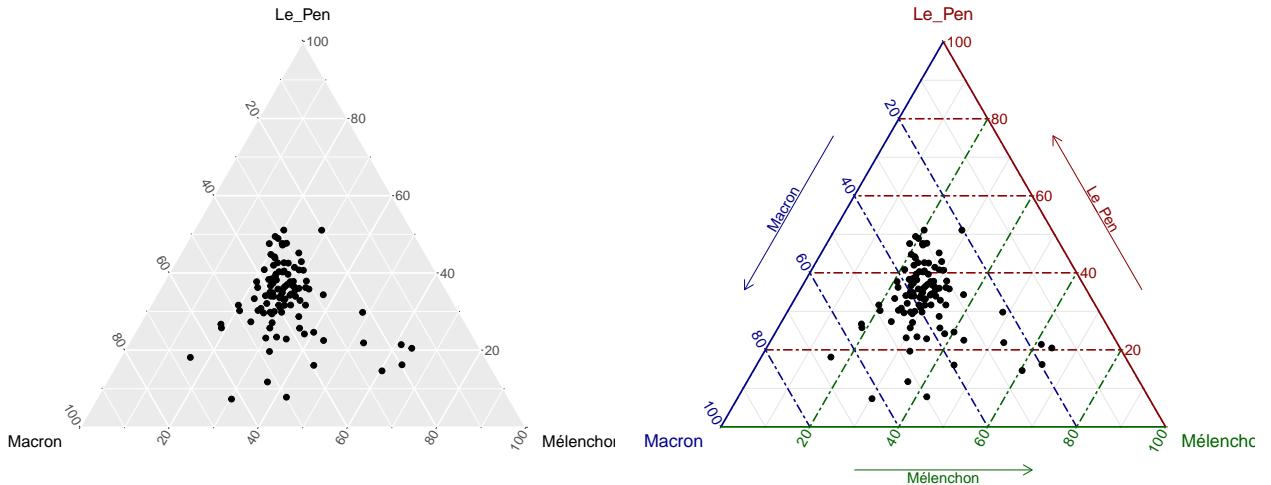


Figure 12: Diagramme ternaire des trois candidats arrivés en tête du 1er tour, avec ajout des couleurs sur les axes dans la figure de droite, afin de simplifier la lecture des échelles

```
p3 <- ggtern(data = geo_dep, mapping = aes(x = Macron, y = Le_Pen, z = Mélenchon)) +
  geom_point(aes(colour = vainqueur), size = 1.5)
p4 <- ggtern(data = res_2022, mapping = aes(x = Macron_VOIX, y = `Le_Pen_VOIX`,
                                             z = Mélenchon_VOIX)) +
  geom_point(aes(size = Inscrits))
grid.arrange(p3, p4, ncol = 2)
```

Enfin, nous pouvons représenter plusieurs diagrammes ternaires en fonction de la région :

```
ggtern(data = geo_dep, mapping = aes(x = Macron, y = Le_Pen, z = Mélenchon)) +
  geom_point(mapping = aes(size = Inscrits, colour = vainqueur)) +
  facet_wrap(~ nom_reg)
```

On citera comme alternative à la librairie `ggtern` et à la philosophie `ggplot2`, la jeune librairie `Ternary` (Hamilton and Ferry 2018) qui produit des diagrammes ternaires intéressants. On verra que le package `compositions` permet aussi de réaliser des diagrammes ternaires et notamment des matrices de diagrammes ternaires avec la fonction `plot.acomp()`.

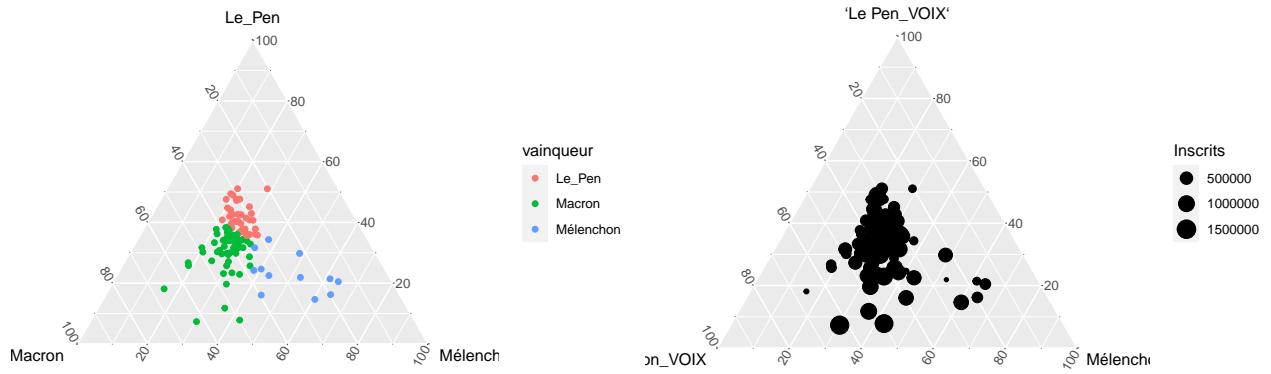


Figure 13: Ajout d'une information supplémentaire qualitative (à gauche) et quantitative (à droite)

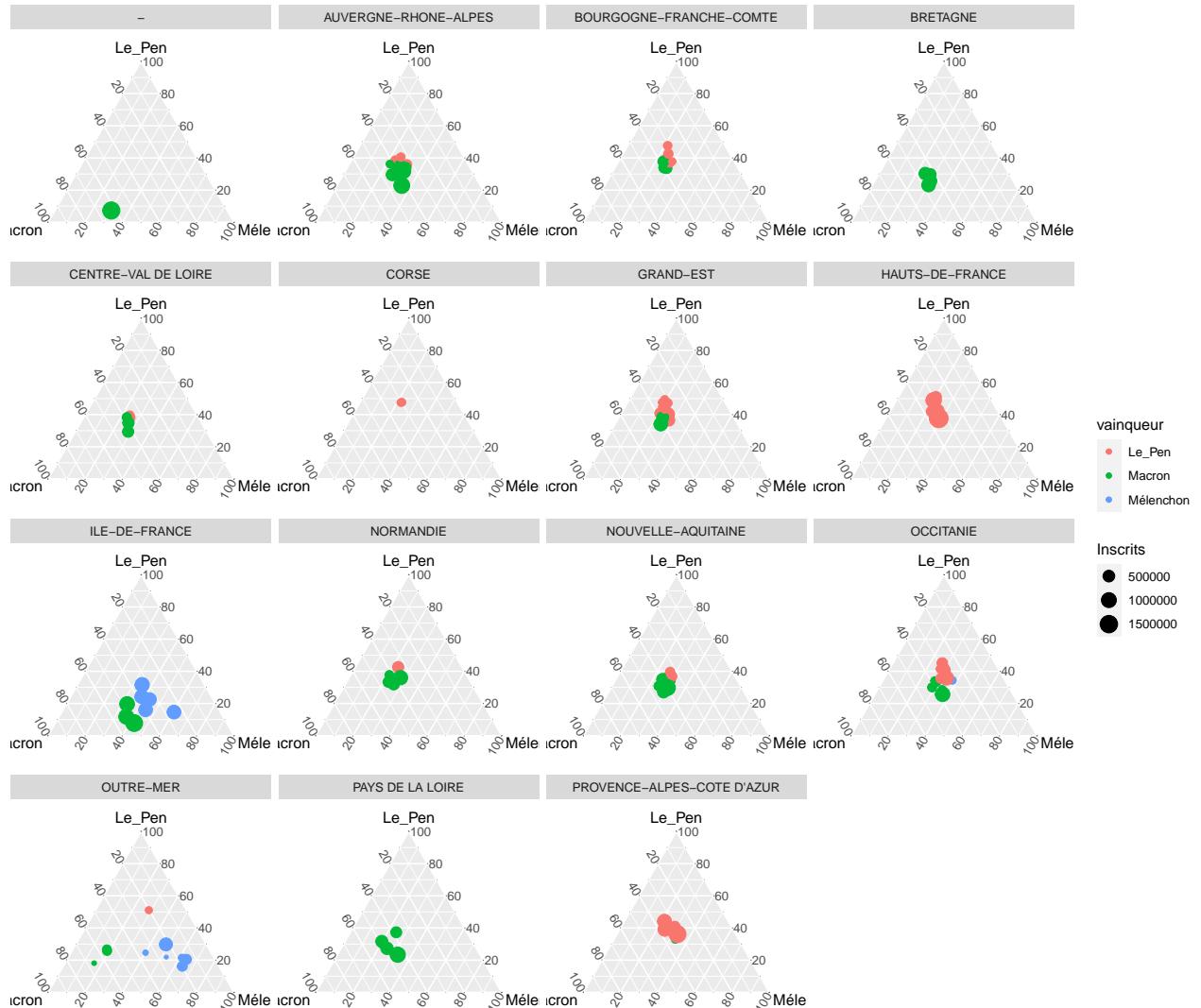


Figure 14: Diagrammes ternaires conditionnelles à la région

Remarque : pour construire le triangle de Maxwell présenté dans le chapitre 1, on associe à chaque point du simplexe (x_1, x_2, x_3) la couleur ($r = x_1, g = x_2, b = x_3$) (dans ce cas particulier, les bordures sont autorisées car on n'a pas besoin d'appliquer des logarithmes). Pour représenter l'ensemble des couleurs possibles, on va d'abord créer une séquence de points dans le simplexe à l'aide de la fonction `seq_simplex()` que nous avons créée. L'argument `nb_noeud` correspond au nombre de noeuds possible observé sur une arrête du diagramme ternaire.

```
seq_simplex <- function(nb_noeud, zero = T) {
  interval <- seq(0, 1, length.out = nb_noeud)
  res <- NULL
  for(i in 1:nb_noeud) {
    for(j in 1:(nb_noeud - i)) {
      res <- rbind(res,
        c(x = 1 - interval[j] - interval[i],
          y = interval[j],
          z = interval[i]))
    }
  }
  res <- res[-nrow(res), ]
  if (!zero) {
    res <- res[!apply(res, 1, function(x) any(x == 0)), ]
  }
  return(res)
}
```

Ensuite, on crée une couleur par point du triangle à l'aide de la fonction `rgb()` et on obtient le triangle de Maxwell :

```
my_data <- as.data.frame(seq_simplex(100))

ggtern(data = my_data, mapping = aes(x = x, y = y, z = z)) +
  geom_point(size = 1.5, col = rgb(my_data$x, my_data$y, my_data$z))
```

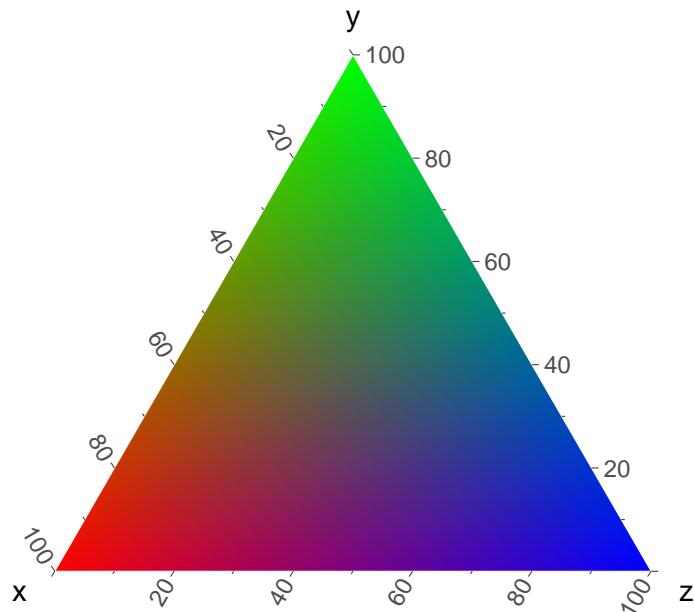


Figure 15: Triangle de Maxwell

Enfin, dans le cas où les données sont temporelles, il est possible de joindre les observations par un trait. Ici, on reprend l'exemple des résultats des élections présidentielles depuis 1958; dans ce cas particulier, on doit d'abord transformer les données du format “long” au format “wide” avec l'option `pivot_wider()`. Ensuite, on ne sélectionne ici que trois partis et on joint les points entre eux avec la fonction `geom_line()` :

```
time_chart_wide <- pivot_wider(time_chart,
    names_from = parti, values_from = vote)
ggtern(data = time_chart_wide,
    mapping = aes(x = droite, y = gauche, z = extrême)) +
  geom_point(size = 1.5) +
  geom_line()
```

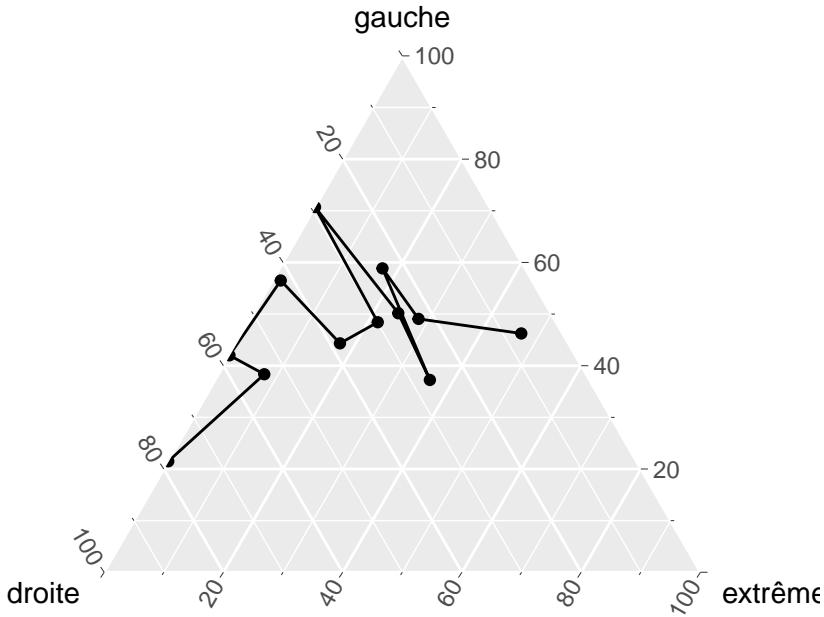


Figure 16: Diagramme ternaire sur des données temporelles

4.2 Le package `compositions` et la classe `acomp`

La fonction `acomp` définit une classe d'objet `acomp` en faisant la clôture du jeu de données qu'on lui fournit en entrée. Autrement dit, qu'on applique la fonction sur le nombre de voix des candidats ou bien, le rapport “nombre de voix” / “nombre d’exprimées,” le résultat sera le même. Ici, on applique d'abord la fonction `acomp()` sur le nombre de voix obtenus par les 3 candidats :

```
library("compositions")
comp_a <- acomp(res_2022[, c("Macron_VOIX", "Le Pen_VOIX", "Mélenchon_VOIX")])
names(comp_a) <- c("Macron", "Le Pen", "Mélenchon")
```

Ensuite, on applique la fonction `acomp()` sur le rapport “nombre de voix” sur “nombre d’exprimées” :

```
comp_b <- acomp(res_2022[, c("Macron_EXP", "Le Pen_EXP", "Mélenchon_EXP")])
names(comp_b) <- c("Macron", "Le Pen", "Mélenchon")
```

On affiche ensuite les résultats obtenus pour le 1er département et on constate que les résultats sont quasi-identiques (les différences observées étant dues probablement aux arrondies décimales diffusées dans la variable “nombre de voix” sur “nombre d’exprimées”) :

```
comp_a[1, ]
```

```

##      Macron     Le Pen     Mélenchon
## "0.3893950" "0.3663749" "0.2442302"
## attr(,"class")
## [1] "acomp"
comp_b[1, ]

```

```

##      Macron     Le Pen     Mélenchon
## "0.3893967" "0.3663338" "0.2442694"
## attr(,"class")
## [1] "acomp"

```

Remarque : dans le package `compositions`, il existe une fonction `clo()` qui permet de faire la clotûre sur un vecteur, sans créer d'objet `acomp`. Par exemple :

```
clo(c(1, 2, 3))
```

```
## [1] 0.1666667 0.3333333 0.5000000
```

Lorsqu'on extrait un sous-ensemble de colonnes, le résultat est une sous-composition sur laquelle la clotûre est faite automatiquement :

```
head(comp_a[, c("Macron", "Le Pen")])
```

```

##      Macron     Le Pen
## [1,] "0.5152296" "0.4847704"
## [2,] "0.3601123" "0.6398877"
## [3,] "0.4969556" "0.5030444"
## [4,] "0.4443495" "0.5556505"
## [5,] "0.5100863" "0.4899137"
## [6,] "0.4840416" "0.5159584"
## attr(,"class")
## [1] "acomp"

```

Par ailleurs, on convertit un objet `acomp` en `matrix` de la façon suivante :

```
head(as(comp_a[, c("Macron", "Le Pen")], "matrix"))
```

```

##      Macron     Le Pen
## [1,] 0.5152296 0.4847704
## [2,] 0.3601123 0.6398877
## [3,] 0.4969556 0.5030444
## [4,] 0.4443495 0.5556505
## [5,] 0.5100863 0.4899137
## [6,] 0.4840416 0.5159584

```

Lorsqu'on applique la fonction `acomp()` sur une composition avec des zéros, cela crée un BDL (“below detection limit”) à l'emplacement des zéros :

```

chimie <- data.frame(Cr = c(27.50, 30.40, 25.60),
                      B = c(17, 23, 14),
                      P = c(148, 433, 135),
                      V = c(29, 42, 33),
                      Cu = c(2.7, 3.8, 0),
                      Ti = c(4335, 3305, 3925),
                      Ni = c(0, 16.6, 14.2))
chimie_acomp <- accomp(chimie)
chimie_acomp

```

	Cr	B	P	V	Cu
##					

```

## [1,] "0.006031760" "0.003728724" "0.03246184" "0.006360765" "0.0005922092"
## [2,] "0.007888318" "0.005968135" "0.11235664" "0.010898334" "0.0009860398"
## [3,] "0.006173435" "0.003376097" "0.03255522" "0.007957943" "BDL"
##      Ti          Ni
## [1,] "0.9508247" "BDL"
## [2,] "0.8575951" "0.004307437"
## [3,] "0.9465130" "0.003424327"
## attr(),"class")
## [1] "acomp"

```

On peut utiliser une des méthodes présentées dans le chapitre 5 pour remplacer ces zéros. Ici, on va simplement remplacer les zéros par la valeur $a \times l_d$, où a est généralement pris égal à $2/3$ et l_d est la valeur de détection limite de la composante d .

```
chimie_acomp_2 <- zeroreplace(chimie_acomp, d = rep(0.001, 7), a = 2/3)
```

Si on souhaite sélectionner des composantes sans faire la clotûre, on peut utiliser la fonction `aplus()` :

```
aplus(chimie_acomp_2, c("Ti", "Ni"))
```

```

##      Ti          Ni
## [1,] "0.9508247" "0.0006666667"
## [2,] "0.8575951" "0.0043074368"
## [3,] "0.9465130" "0.0034243272"
## attr(),"class")
## [1] "aplus"

```

Enfin, si on souhaite faire l'amalgamation de ces deux composantes en une seule en prenant la somme arithmétique, on peut utiliser la fonction `totals()` :

```
totals(aplus(chimie_acomp_2, c("Ti", "Ni")))
```

```
## [1] 0.9514914 0.8619025 0.9499373
```

Pour créer un nouveau jeu de données qui tient en compte cette amalgamation, il faut recréer un objet `acomp` de la façon suivante :

```

acomp(
  cbind(aplus(chimie_acomp_2, c("Cr", "B", "P", "V", "Cu")),
    Ni_Ti = totals(aplus(chimie_acomp_2, c("Ti", "Ni"))))
)

```

```

##      Cr          B          P          V          Cu
## [1,] "0.006027741" "0.003726240" "0.03244021" "0.006356527" "0.0005918146"
## [2,] "0.007888318" "0.005968135" "0.11235664" "0.010898334" "0.0009860398"
## [3,] "0.006169322" "0.003373848" "0.03253353" "0.007952642" "0.0006662225"
##      Ni_Ti
## [1,] "0.9508575"
## [2,] "0.8619025"
## [3,] "0.9493044"
## attr(),"class")
## [1] "acomp"

```

Une autre façon de faire l'amalgamation est de sélectionner un certain nombre de composantes et faire la moyenne géométrique de toutes les autres. Cela se fait au moyen de la fonction `acompmargin()` :

```
acompmargin(chimie_acomp_2, c("Cr", "B", "P", "V", "Cu"))
```

```

##      Cr          B          P          V          Cu
## [1,] "0.08112402" "0.05014939" "0.4365947" "0.08554897" "0.007964904"

```

```

## [2,] "0.03966449" "0.03000932" "0.5649581" "0.05479963" "0.004958062"
## [3,] "0.05734161" "0.03135870" "0.3023874" "0.07391692" "0.006192297"
## *
## [1,] "0.3386180"
## [2,] "0.3056104"
## [3,] "0.5288031"
## attr(),"class")
## [1] "acomp"

```

4.2.1 Perturbation, puissance

On remarque que les vecteurs affichés ont un attribut appelé `acomp`. Cela implique qu'il ne s'agit pas d'un vecteur "classique," mais un vecteur compositionnel. La conséquence est que les opérations de type `+`, `*` appliqués sur cet objet ne seront pas celles de la géométrie euclidienne, mais celles de la géométrie d'Aitchison. Par exemple, l'opérateur `+` entre deux vecteurs de composition va appliquer l'opération de perturbation. Ainsi, si on perturbe les deux premiers vecteurs de composition :

```

a <- comp_a[1, ]
b <- comp_a[2, ]
d <- a + b
d

```

```

##      Macron      Le Pen     Mélenchon
## "0.3213754" "0.5372960" "0.1413286"
## attr(),"class")
## [1] "acomp"

```

Cela est équivalent à faire le produit entre les composantes de départ et de faire ensuite la clotûre :

```

produit <- res_2022[1, c("Macron_VOIX", "Le_Pen_VOIX", "Mélenchon_VOIX")] *
  res_2022[2, c("Macron_VOIX", "Le_Pen_VOIX", "Mélenchon_VOIX")]
d_bis <- produit / sum(produit)
d_bis

```

```

##    Macron_VOIX Le_Pen_VOIX Mélenchon_VOIX
## 1   0.3213754    0.537296    0.1413286

```

De même l'opérateur `*` correspond à l'opérateur puissance. Par exemple, on applique à la première observation la puissance 1/2 :

```
d * (1 / 2)
```

```

##      Macron      Le Pen     Mélenchon
## "0.3382777" "0.4373950" "0.2243273"
## attr(),"class")
## [1] "acomp"

```

ce qui est équivalent à faire :

```

power <- d_bis ^ 0.5
power / sum(power)

```

```

##    Macron_VOIX Le_Pen_VOIX Mélenchon_VOIX
## 1   0.3382777    0.437395    0.2243273

```

Il est possible de représenter directement les objets `acomp` dans un diagramme ternaire avec la fonction `plot.acomp()`. On représente trois exemples où on calcule la moyenne dans le simplexe entre deux compositions où l'une des deux est fixée alors que l'autre se rapproche d'un sommet.

```

par(mfrow = c(1, 3))
b1 <- acomp(c(0.2, 0.5, 0.3))
a1 <- acomp(c(0.4, 0.1, 0.5))
plot(a1)
plot(b1, add = T)
plot((a1 + b1) * (1 / 2), add = T, pch = 16, col = "red")
a2 <- acomp(c(0.4, 0.01, 0.59))
plot(a2)
plot(b1, add = T)
plot((a2 + b1) * (1 / 2), add = T, pch = 16, col = "red")
a3 <- acomp(c(0.4, 0.001, 0.599))
plot(a3)
plot(b1, add = T)
plot((a3 + b1) * (1 / 2), add = T, pch = 16, col = "red")

```

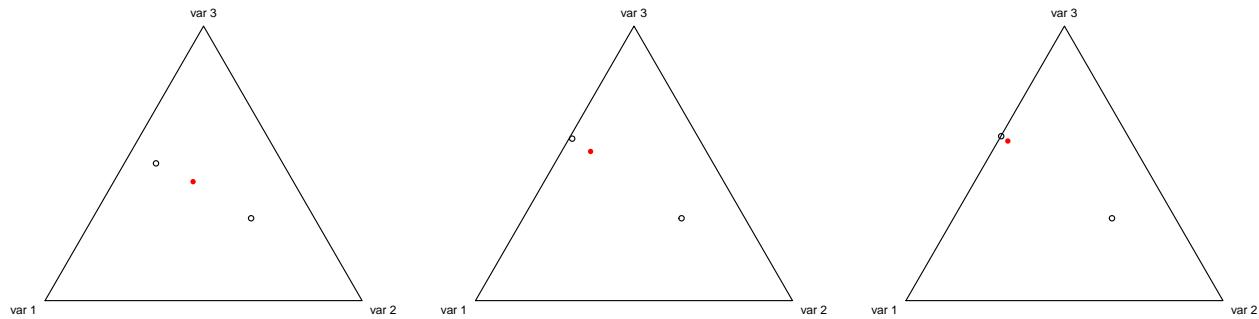


Figure 17: Illustration de la moyenne dans le simplexe de deux vecteurs de composition

De même, il est possible d'ajouter une ligne brisée associée à un jeu de données de type `acomp` avec la fonction `lines.acomp()`. Ici, on représente pour plusieurs valeurs de k , les compositions $k \odot d$, où d est la moyenne dans le simplexe des vecteurs de composition a et b

```

plot(a)
plot(b, add = T)
plot(d * (1 / 2), add = T, pch = 16, col = "red")
lines(d * seq(-100, 100, length.out = 1000), lty = 2)

```

4.2.2 Centrage des données

Nous allons montrer ci-après un exemple d'utilisation de l'opérateur de perturbation. On considère la teneur en protéines, glucides, lipides de différents types de farine (source : Cqual).

```

farine <- data.frame(type = c("T110", "T150", "T45", "T55", "T65", "T80",
  "mais", "pois chiche", "riz", "sarrasin", "seigle"),
  proteines = c(10.3, 12.2, 9.94, 9.9, 14.9, 10.9, 6.23, 22.4, 8, 11.5, 8.7),
  glucides = c(70.2, 66.67, 76.31, 75.2, 69.1, 74.97, 78.74,
    57.90, 74.8, 68.43, 71.63),
  lipides = c(1.5, 1.52, 0.82, 1, 1, 1.18, 2.1, 6.69, 2.5, 2.19, 1.37))

```

On représente le diagramme ternaire et on remarque que la plupart des points sont concentrés proches du sommet “glucides” ce qui rend difficile la comparaison entre les farines (à part celle qui est très différente des autres).

```

farine_comp <- accomp(farine[, c("proteines", "glucides", "lipides")])
ggtern(data = farine, mapping = aes(x = proteines, y = lipides, z = glucides)) +

```

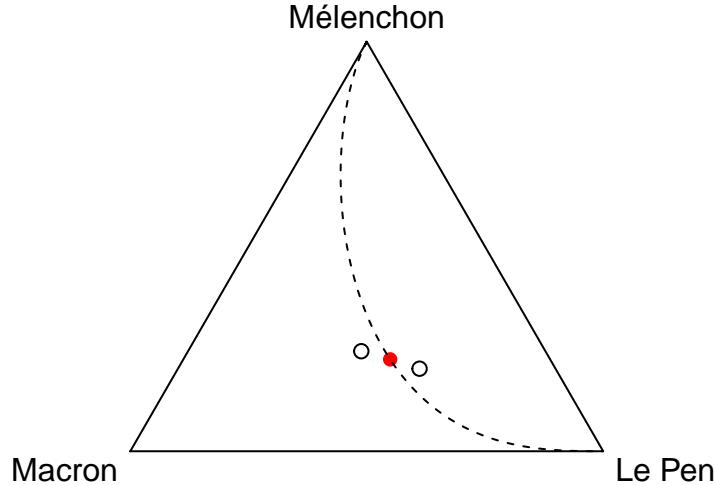


Figure 18: Illustration de la moyenne dans le simplexe de deux vecteurs de composition et représentation d'une droite

```
geom_point(size = 1.5) +
  geom_point(data = data.frame(t(as.matrix(farine_comp))),
             mapping = aes(x = proteines, y = lipides, z = glucides), col = "red") +
  theme(legend.position = c(0, 1), legend.justification = c(1, 1)) +
  theme_rbw()
```

Une solution proposée par (Von Eynatten, Pawlowsky-Glahn, and Egozcue 2002) est de centrer les données dans la géométrie d'Aitchison. Cela revient à calculer la moyenne dans le simplexe puis d'appliquer l'opérateur soustraction dans le simplexe sur chaque observation. Pour calculer la moyenne géométrique, cela revient à calculer la moyenne en utilisant les opérateurs du simplexe ($\bar{x} = \frac{1}{n} \odot (x_1 \oplus \dots \oplus x_n)$) :

```
farine_mean <- farine_comp[1, ]
for (k in 2:nrow(farine_comp)) {
  farine_mean <- farine_mean + farine_comp[k, ]
}
farine_mean <- farine_mean / nrow(farine_comp)
```

Remarque : on peut directement utiliser la fonction `mean()` qui va appliquer directement cette formule sur les données de composition

```
mean(farine_comp)
```

```
##   proteines     glucides      lipides
## "0.12897500" "0.85135822" "0.01966678"
## attr(,"class")
## [1] "acomp"
```

On centre les données de la manière suivante ($x_k \ominus \bar{x}$, $k = 1, \dots, n$):

```
farine_comp_ce <- farine_comp - farine_mean
```

Ensuite, on va créer une fonction qui va nous permettre d'ajouter des étiquettes dans le diagramme ternaire. La fonction `compo_to_ternary()` permet de donner les coordonnées cartésiennes d'une composition dans le diagramme ternaire.

```
compo_to_ternary <- function(s_3, A = c(0, 0), B = c(1, 0),
                             C = c(0.5, sqrt(3) / 2)) {
```

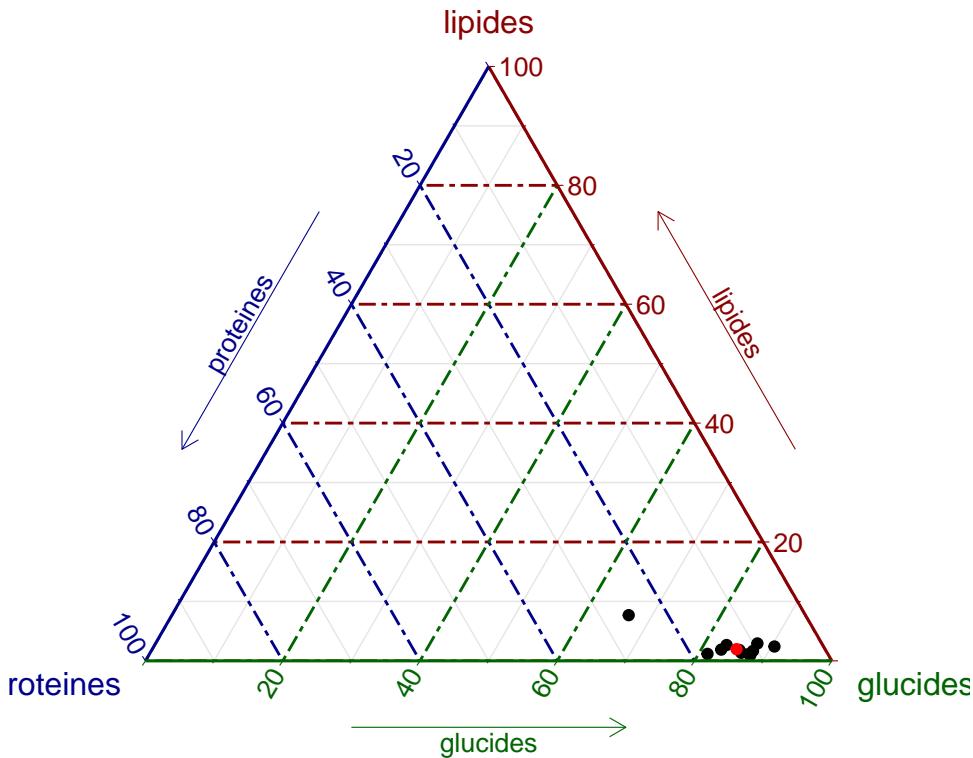


Figure 19: Diagramme ternaire de la composition en protéine, glucides et lipides de plusieurs types de farines

```

if (length(s_3) == 3)
  s_3 <- t(matrix(s_3))
Y_simplex_x <- s_3[, 1] * A[1] + s_3[, 2] * B[1] + s_3[, 3] * C[1]
Y_simplex_y <- s_3[, 1] * A[2] + s_3[, 2] * B[2] + s_3[, 3] * C[2]
return(cbind(Y_simplex_x, Y_simplex_y))
}

```

Dans le diagramme ternaire qui représente les données centrées, on représente également les nouvelles échelles correspondantes. Il est intéressant de noter que le centrage d'une droite reste une droite. Ainsi, nous avons représenté par exemple les segments qui représentent l'ensemble des points du simplexe tel que *lipides* = 0.1, 0.05, 0.01

```

op <- par(oma = c(.1, .1, .1, .1), mar = c(0.3, 1.2, .5, 1.4))
plot(farine_comp_ce, pch = 16, cex = 0.5, labels = "")
text(c(0.04, .99, 1/2), c(-.06, -.06, sqrt(3)/2 + 0.02),
  c("proteines", "glucides", "lipides"), pos = 3, cex = 0.8,
  col = c("#E16A86", "#50A315", "#009ADE"))
coord_leg <- compo_to_ternary(farine_comp_ce)
text(coord_leg[, 1], coord_leg[, 2], farine$type, cex = 0.5, pos = 3)
# l'axe Lipides
for (k in c(0.01, 0.05, 0.1, 0.25)) {
  c_1 <- c(0.000001, 1 - k, k)
  c_2 <- c(1 - k, 0.000001, k)
  lines(acomp(rbind(acomp(c_1) - farine_mean,
                     accomp(c_2) - farine_mean)), steps = 1, lty = 4,
        lwd = 0.7, col = "#009ADE")
  coord_leg <- compo_to_ternary(acomp(c_1) - farine_mean)
}

```

```

text(coord_leg[, 1], coord_leg[, 2], k, cex = 0.5, pos = 4, col = "#009ADE")
}
# l'axe Glucides
for (k in c(0.75, 0.9, 0.95, 0.99)) {
  c_1 <- c(0.000001, k, 1 - k)
  c_2 <- c(1 - k, k, 0.000001)
  lines(acomp(rbind(acomp(c_1) - farine_mean,
    accomp(c_2) - farine_mean)), steps = 1, lty = 4, lwd = 0.7, col = "#50A315")
  coord_leg <- compo_to_ternary(acomp(c_2) - farine_mean)
  text(coord_leg[, 1], coord_leg[, 2], k, cex = 0.5, pos = 1,
    srt = -120, col = "#50A315")
}
# l'axe Protéines
for (k in c(0.05, 0.1, 0.25)) {
  c_1 <- c(k, 0.000001, 1 - k)
  c_2 <- c(k, 1 - k, 0.000001)
  lines(acomp(rbind(acomp(c_1) - farine_mean,
    accomp(c_2) - farine_mean)), steps = 1, lty = 4, lwd = 0.7, col = "#E16A86")
  coord_leg <- compo_to_ternary(acomp(c_1) - farine_mean)
  text(coord_leg[, 1], coord_leg[, 2], k, cex = 0.5,
    pos = 3, srt = 120, col = "#E16A86")
}
# moyenne géométrique
g <- compo_to_ternary(farine_mean - farine_mean)
points(g[, 1], g[, 2], pch = 15, col = "red")

```

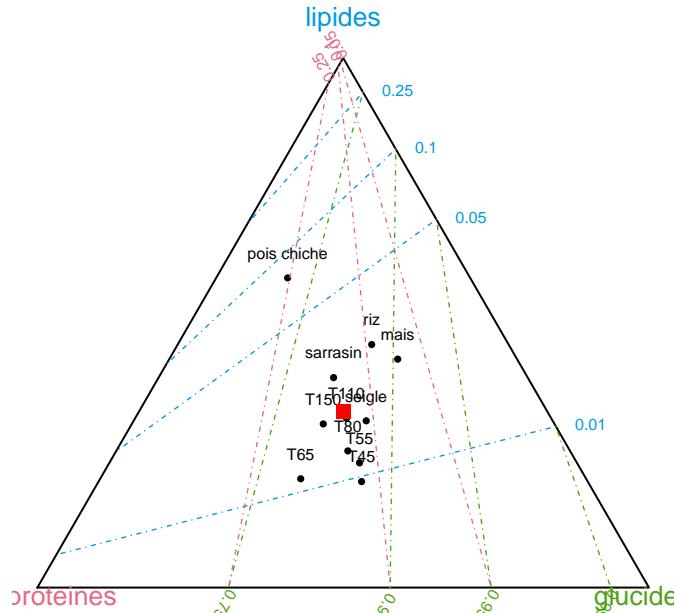


Figure 20: Diagramme ternaire des données de composition lipides, glucides, protéines, centré autour de la moyenne géométrique

On avait déjà identifié la farine de pois chiche comme étant très différente des autres. A présent, on peut également voir que les faines de type T45, T55, T65, T80 sont moins chargées en lipides que les autres. Les farines de maïs et de riz sont quant à elles moins chargées en protéines.

4.2.3 Produit scalaire, norme et distance

Le produit scalaire entre deux vecteurs du simplex s'effectue avec la fonction `scalar()` :

```
scalar(farine_comp[1, ], farine_comp[2, ])
```

```
## [1] 7.271207
```

Cela revient à effectuer le calcul suivant :

```
D <- 3
ps <- 0
x <- farine_comp[1, ]
y <- farine_comp[2, ]
for (j in 1:(D - 1))
  for (i in (j + 1):D)
    ps <- ps + log(x[i] / x[j]) * log(y[i] / y[j])
(ps <- ps / 3)
```

```
## glucides
## 7.271207
```

La fonction `norm()` permet de calculer la norme d'un vecteur:

```
norm(x)
```

```
## [1] 2.719452
```

```
norm(y)
```

```
## [1] 2.678204
```

Ce qui est équivalent à :

```
norm_x <- 0
norm_y <- 0
for (j in 1:(D - 1)) {
  for (i in (j + 1):D) {
    norm_x <- norm_x + log(x[i] / x[j]) * log(x[i] / x[j])
    norm_y <- norm_y + log(y[i] / y[j]) * log(y[i] / y[j])
  }
}
norm_x <- sqrt(norm_x / 3)
norm_y <- sqrt(norm_y / 3)
```

La distance d'Aitchison est obtenu avec la fonction `dist()`. Par défaut, elle calcule toutes les distances entre les individus d'un object `acomp`. Ici, on ne calcule que la distance entre les deux premiers individus :

```
dist(farine_comp[1:2, ])
```

```
##           1
## 2 0.1605667
```

Ce qui est équivalent à faire :

```
sqrt(scalar(x-y, x-y))
```

```
## [1] 0.1605667
```

Remarque : la fonction `dist()` permet de calculer d'autres types de distance (Manhattan ou Minkowski)

4.3 Les transformations `alr`, `clr`, `ilr`

Les trois principaux types de transformation présentées dans le chapitre 3 s'obtiennent avec les fonctions `alr()` (pour *additive log ratio*), `clr()` (pour *centered log-ratio*) et `ilr()` (pour *isometric log-ratio*) et `alrinv()`, `clrinv()` et `ilrinv()` pour leurs inverses. Par ailleurs, les fonctions `ilr2clr()` et `clr2ilr()` permettent de passer d'une transformation à une autre.

Par défaut, `alr()` utilise comme composition de référence la dernière de la liste. On peut changer la référence avec l'option `ivar`. Concernant la fonction `ilr()`, elle prend considère par défaut la matrice de contraste du pivot (aussi appelée Helmert), calculée avec la fonction `ilrBase()` et qui vaut :

$$V = \begin{pmatrix} \frac{D-1}{\sqrt{D(D-1)}} & 0 & \cdots & 0 & 0 \\ \frac{-1}{\sqrt{D(D-1)}} & \frac{D-2}{\sqrt{(D-1)(D-2)}} & \cdots & 0 & 0 \\ \frac{-1}{\sqrt{D(D-1)}} & \frac{-1}{\sqrt{(D-1)(D-2)}} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{-1}{\sqrt{D(D-1)}} & \frac{-1}{\sqrt{(D-1)(D-2)}} & \cdots & \frac{2}{\sqrt{6}} & 0 \\ \frac{-1}{\sqrt{D(D-1)}} & \frac{-1}{\sqrt{(D-1)(D-2)}} & \cdots & \frac{-1}{\sqrt{6}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{D(D-1)}} & \frac{-1}{\sqrt{(D-1)(D-2)}} & \cdots & \frac{-1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$$

```
V <- matrix(c(2 / sqrt(6), - 1 / sqrt(6), - 1 / sqrt(6),
            0, 1 / sqrt(2), - 1 / sqrt(2)), ncol = 2)
alr_a <- alr(comp_a, ivar = 3)
clr_a <- clr(comp_a)
ilr_a <- ilr(comp_a, V = V)
```

Nous verrons par la suite quelle stratégie il est possible d'adopter pour définir une matrice de contraste judicieuse.

4.3.1 Propriétés des transformations

La transformation $clr(x)$ fait intervenir la moyenne géométrique de x qui vaut $g(x) = (x_1 \times x_2 \times \dots \times x_D)^{1/D}$. Nous allons cartographier $g(x)$ dans le cas où $D = 3$. On constate que $g(x)$ est strictement supérieur à 0 et inférieur ou égale à $1/3$, cette dernière valeur étant atteinte pour le point neutre $(1/3, 1/3, 1/3)$. Plus on s'éloigne du point neutre, plus $g(x)$ diminue, les valeurs les plus faibles étant localisées proche du sommet.

Nous allons à présent vérifier certaines propriétés de la transformation `clr`.

- La somme des composantes de la transformation `clr` vaut 0

```
all(round(apply(clr_a, 1, sum), 12) == 0)
```

```
## [1] TRUE
•  $clr(\mathbf{x}) = G_D \ln(\mathbf{x}) = (\mathbf{I}_D - \frac{1}{D} \mathbf{1}_{D \times D}) \ln(\mathbf{x})$ 
(diag(3) - matrix(1/3, 3, 3)) %*% log(as.numeric(comp_a[1, ]))

##          [,1]
## [1,]  0.1758067
## [2,]  0.1148695
## [3,] -0.2906762
clr_a[1, ]

##      Macron     Le Pen   Mélenchon
## 0.1758067 0.1148695 -0.2906762
```

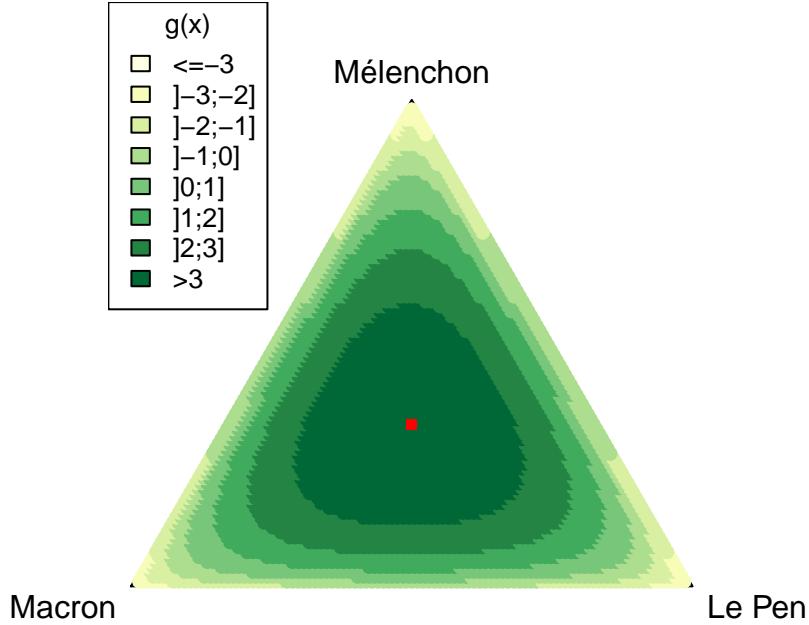


Figure 21: Cartographie des valeurs de $g(x)$ dans le diagramme ternaire

```
## attr(,"class")
## [1] "rmult"
```

On s'intéresse à présent aux propriétés de ilr :

- $\text{ilr}_V(\mathbf{x} \oplus \mathbf{y}) = \text{ilr}_{V_D}(\mathbf{x}) + \text{ilr}_{V_D}(\mathbf{y})$

```
ilr(x + y)
```

```
## [1] 2.557975 -4.750513
```

```
## attr(,"class")
```

```
## [1] "rmult"
```

```
ilr(x) + ilr(y)
```

```
## [1] 2.557975 -4.750513
```

```
## attr(,"class")
```

```
## [1] "rmult"
```

- $\text{ilr}_V(\alpha \odot \mathbf{x}) = \alpha \cdot \text{ilr}_V(\mathbf{x})$

```
ilr(2 * x)
```

```
## [1] 2.714165 -4.713277
```

```
## attr(,"class")
```

```
## [1] "rmult"
```

```
2 * ilr(x)
```

```
## [1] 2.714165 -4.713277
```

```
## attr(,"class")
```

```
## [1] "rmult"
```

- $\langle \mathbf{x}, \mathbf{y} \rangle_A = \langle \text{ilr}_V(\mathbf{x}), \text{ilr}_V(\mathbf{y}) \rangle_E$

```

scalar(x, y)

## [1] 7.271207
sum(ilr(x) * ilr(y))

## [1] 7.271207
•  $\|x\|_A = \|\text{ilr}_V(x)\|_E$ 
norm(x)

## [1] 2.719452
sqrt(sum(ilr(x) * ilr(x)))

## [1] 2.719452
•  $d_A(x, y) = d_E(\text{ilr}_{V_D}(x), \text{ilr}_{V_D}(y))$ 
norm(x-y)

## [1] 0.1605667
dist(rbind(
  ilr(x),
  ilr(y)
))

##           1
## 2 0.1605667

```

4.3.2 Signification des axes

A présent, nous allons représenter le nuage des points des scores des trois principaux candidats dans les deux espaces transformés alr et ilr, avant de faire la représentation des coordonnées clr :

```

par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), oma = c(0, 0, 0, 0))
plot(alr_a[, 1], alr_a[, 2], xlab = "alr 1", ylab = "alr 2", main = "ALR")
abline(h = 0, v = 0, lty = 2)
plot(ilr_a[, 1], ilr_a[, 2], xlab = "ilr 1", ylab = "ilr 2", main = "ILR")
abline(h = 0, lty = 2)

```

L'interprétation des coordonnées dans les espaces transformés n'est pas triviale, mais il peut apporter des informations complémentaires. Par exemple, dans l'espace alr ayant pour référence Mélenchon, le nuage de points des deux composantes $\log(\frac{\text{Macron}}{\text{Mélenchon}})$ et $\log(\frac{\text{Le Pen}}{\text{Mélenchon}})$ permet de découper le nuage de points en 4 quadrants :

- le quadrant H-H (high-high) où Mélenchon a été battu par Macron et Le Pen
- le quadrant L-L (low-low) où Mélenchon a battu Macron et Le Pen
- le quadrant H-L (high-low) où Mélenchon a été battu par Macron et a battu Le Pen
- le quadrant L-H (low-high) où Mélenchon a été battu par Macron et a battu Le Pen

Par construction, l'axe 2 de l'espace ilr oppose Mélenchon et Le Pen : des valeurs positives indiquent un score de Le Pen supérieur à celui de Mélenchon. L'axe 1 représente la quantité $\frac{1}{\sqrt{6}} \log(\frac{\text{Macron}^2}{\text{Le Pen} \times \text{Mélenchon}})$ et est plus compliqué à interpréter : plus les valeurs sont fortement positives, plus le score de Macron est fort comparativement à celui de Le Pen ou Mélenchon. Pour résumer le lien qui attache les données de composition avec l'espace ilr (avec le choix de la matrice V précédente), nous avons discréteisé les valeurs des deux premières composantes ilr 1 et ilr 2 et les avons représentés dans le diagramme ternaire avec des couleurs différentes.

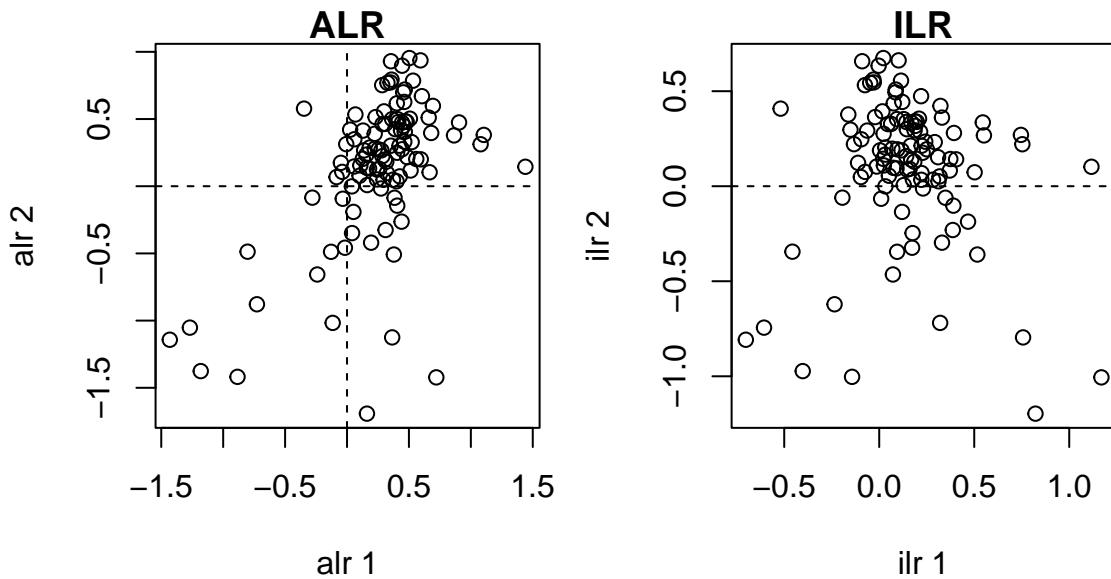


Figure 22: Représentation des scores des trois principaux candidats dans les espaces alr et ilr

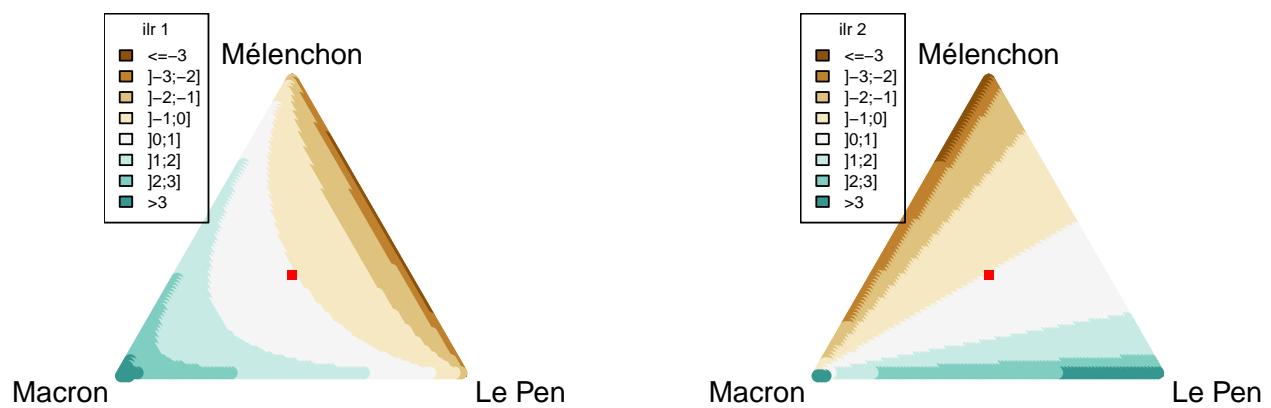


Figure 23: Cartographie des valeurs des ilr 1 et ilr 2 dans le diagramme ternaire

L'espace CLR est en dimension \mathbb{R}^3 ; une façon de les visualiser et de représenter les paires des CLR dans des nuages de points.

```
par(mfrow = c(1, 3), mar = c(4, 4, 1, 1), oma = c(0, 0, 0, 0))
plot(clr_a[, 1], clr_a[, 2], xlab = "clr 1", ylab = "clr 2")
abline(h = 0, v = 0, lty = 2)
plot(clr_a[, 1], clr_a[, 3], xlab = "clr 1", ylab = "clr 3")
abline(h = 0, v = 0, lty = 2)
plot(clr_a[, 2], clr_a[, 3], xlab = "clr 2", ylab = "clr 3")
abline(h = 0, v = 0, lty = 2)
```

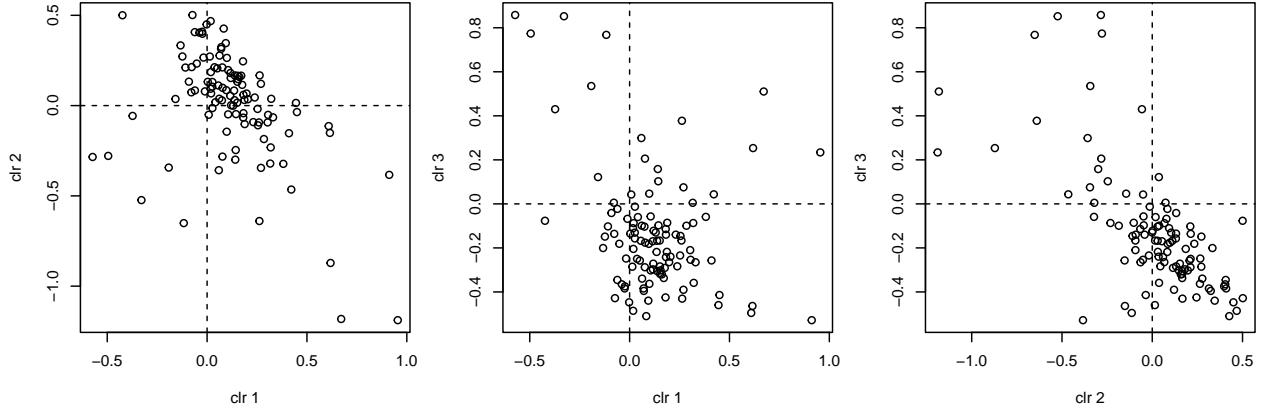


Figure 24: Représentation des scores dans l'espace clr

En cartographiant les valeurs des clr dans des diagrammes ternaires, on constate que le premier axe de clr 1 a la même signification que le premier axe de ilr 1. Plus les valeurs sont fortes et positives, plus le score de Macron est supérieur à celui de Le Pen ou Mélenchon. Finalement, les axes 2 et 3 de l'espace CLR s'interprètent comme le 1er axe, en remplaçant Macron par Le Pen (axe 2) et Mélenchon (axe 3).

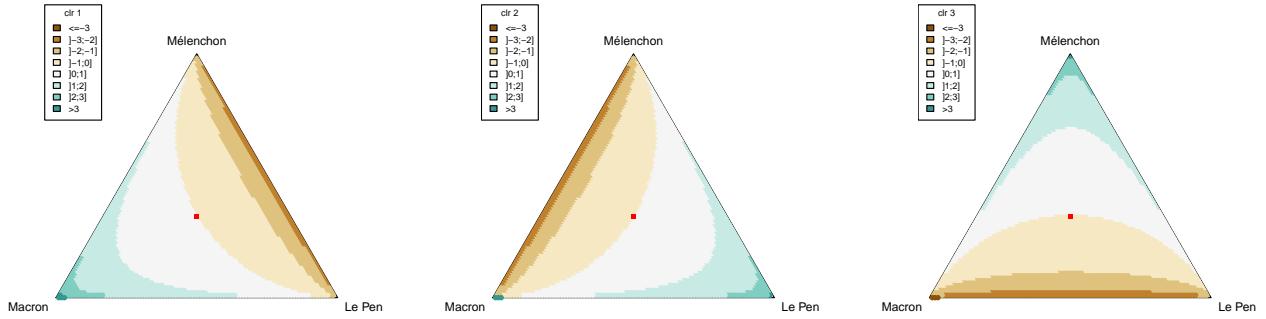


Figure 25: Cartographie des valeurs des CLR dans le diagramme ternaire

4.3.3 Transformation d'objets particuliers

On peut représenter des objets géométriques connus (une droite, un cercle, un carré, etc) dans l'un des espaces transformés et appliquer la transformation inverse pour voir l'équivalent dans le simplexe. On va commencer par représenter plusieurs droites :

On s'intéresse à présenter à des cercles et des ellipses :

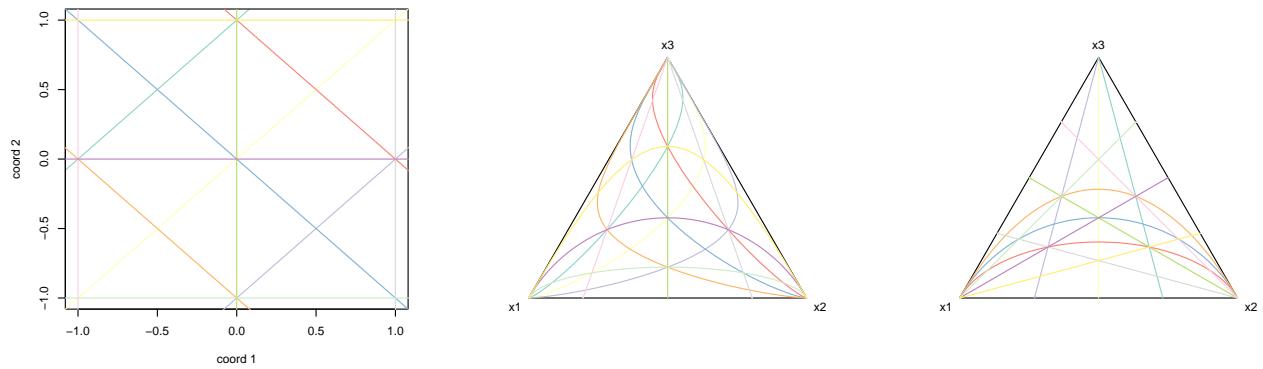


Figure 26: Transformations de droites dans le simplexe lorsque l'espace initial est ilr et alr

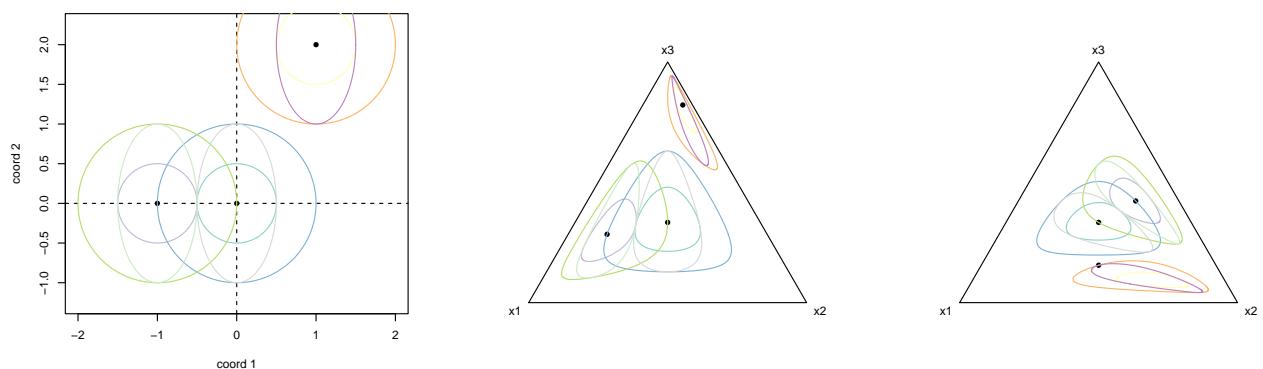


Figure 27: Transformations de cercles et d'ellipses dans le simplexe lorsque l'espace initial est ilr et alr

4.3.4 Moyenne et Variance

Avec les données de composition, le calcul des moyennes et de variances se font dans les espaces transformés. Quel que soit la transformation utilisée, on retrouve la même information; par exemple, si on calcule la moyenne au sens “classique” dans les trois espaces et on les re-transforme dans le simplexe, on retrouve le même résultat :

```
mean_alr <- apply(alr_a, 2, mean)
mean_ilr <- apply(ilr_a, 2, mean)
mean_clr <- apply(clr_a, 2, mean)
alrInv(mean_alr)

##      Macron      Le Pen
## "0.3746182" "0.3348370" "0.2905448"
## attr(,"class")
## [1] "acomp"

ilrInv(mean_ilr, V = V)

## [1] "0.3746182" "0.3348370" "0.2905448"
## attr(,"class")
## [1] "acomp"

clrInv(mean_clr)

##      Macron      Le Pen      Mélenchon
## "0.3746182" "0.3348370" "0.2905448"
## attr(,"class")
## [1] "acomp"
```

Si on applique la fonction `var()` sur un objet `acomp`, cela revient à calculer la matrice de variance-covariance $\hat{\Sigma}$ dans l'espace ilr avec la fonction `var()` :

```
var(ilr_a)

##           [,1]      [,2]
## [1,] 0.082446877 -0.009284042
## [2,] -0.009284042  0.143154111
```

et de revenir dans le simplexe à l'aide de la transformation $V\hat{\Sigma}V^t$:

```
V %*% var(ilr_a) %*% t(V)

##           [,1]      [,2]      [,3]
## [1,] 0.05496458 -0.03284244 -0.02212215
## [2,] -0.03284244  0.09067835 -0.05783591
## [3,] -0.02212215 -0.05783591  0.07995806
```

On retrouve le même résultat que :

```
var_s <- var(comp_a)
```

4.4 Lois dans le simplexe

Nous allons présenter ici quelques lois décrites dans le chapitre 7.

4.4.1 Loi multinomiale

La fonction de base `rmultinom()` permet de simuler selon une loi $\mathcal{M}(p_1, \dots, p_D, N)$ (l'argument `size` correspond au paramètre N et l'argument `prob` au vecteur de probabilités). Par exemple, pour simuler 1000 observations distribuées selon une loi $\mathcal{M}(1/6, 1/3, 1/2, N = 30)$, on procède ainsi :

```
my_multi <- t(rmultinom(1000, size = 30, prob = c(1/6, 1/3, 1/2)))
```

Il peut toutefois y avoir des composantes avec des valeurs 0; dans notre exemple, il y en a 7. Nous allons les conserver ici, mais il pourrait être utile de faire appel aux outils proposés dans le chapitre 5 pour remplacer les zéros. Comme il y a de nombreuses valeurs qui sont dupliquées, nous allons les représenter avec une couleur différente selon le nombre de fois qu'elles apparaissent. La fonction `aggregate()` va nous permettre de récupérer toutes les valeurs distinctes et de calculer leur nombre :

```
my_multi_ag <- aggregate(rep(1, nrow(my_multi)), by = list(x1 = my_multi[, 1],
                                                               x2 = my_multi[, 2],
                                                               x3 = my_multi[, 3])), FUN = sum)
```

Finalement, on définit la palette de couleurs avec le package `RColorBrewer` et on utilise la fonction `ggtern()` qui fera directement l'opération de clôture sur les données de comptage :

```
pal1 <- RColorBrewer::brewer.pal(9, "YlGn")
bk <- seq(0, 45, by = 5)
ind <- findInterval(my_multi_ag$x, bk, all.inside = TRUE)

ggtern(data = my_multi_ag, mapping = aes(x = x1, y = x2, z = x3)) +
  geom_point(size = 1.5, col = pal1[ind])
```

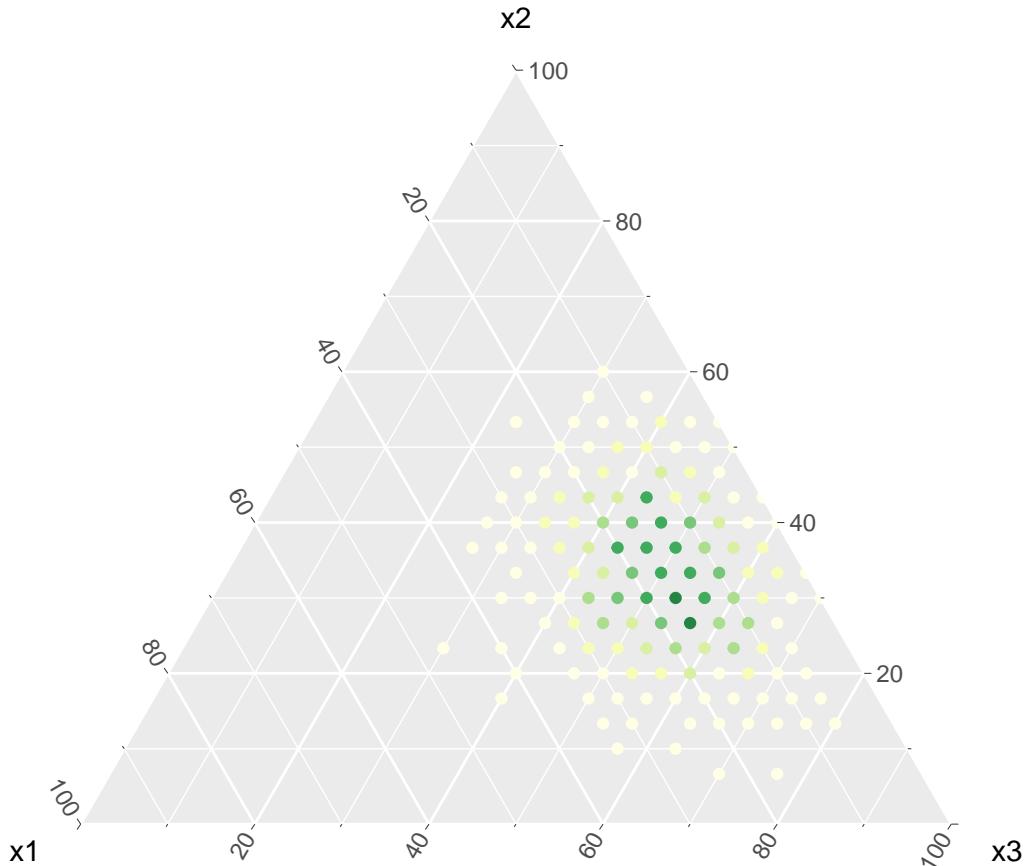


Figure 28: Echantillon simulé selon une loi $M(1/6, 1/3, 1/2, N = 30)$

Remarque : la fonction `dmultinom()` permet de calculer la fonction de densité d'une loi multinomiale lorsqu'on lui donne en entrée une donnée de comptage.

4.4.2 Loi normale dans le simplexe

Dans un premier temps, nous allons simuler des données selon des lois normales dans le simplexe avec les paramètres suivants : $\mu_1 = (1/3, 1/3, 1/3)$, $\mu_2 = (0.5, 0.3, 0.2)$ et $\mu_3 = (0.2, 0.2, 0.6)$ et $\Sigma_1 = 0.1 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$,

$\Sigma_2 = 0.1 \begin{pmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ et $\Sigma_3 = 0.1 \begin{pmatrix} 1 & 0.8 & 0.2 \\ 0.8 & 1 & .5 \\ 0.2 & 0.5 & 1 \end{pmatrix}$, où μ_k représente la moyenne dans le simplexe alors que

Σ_j représente la matrice de variance-covariance dans l'espace clr. Il s'agit des paramètres d'entrée de la fonction `rnorm.acomp()` qui permet de simuler des données. La fonction `ellipses()` permet de représenter l'ellipse $(\psi - \alpha)T(\psi - \alpha)^t = r^2$. En choisissant α comme étant la moyenne dans le simplexe et T la matrice de variance covariance dans l'espace clr, cela revient à tracer les intervalles de confiance sous forme de lignes de niveau (pour cela, on doit choisir le paramètre `r` selon une loi de χ^2 à $D - 1$ degrés de libertés). On remarque que l'allure des nuages des points est très différente d'un ensemble de paramètres à un autre. Il est même difficile de distinguer quelle est l'influence du paramètre de la moyenne et celle de la variance

```
library(latex2exp)
my_mean_vec <- acomp(rbind(
  c(1/3, 1/3, 1/3),
  c(0.5, 0.3, 0.2),
  c(0.2, 0.2, 0.6)
))

my_var_list <- list(
  0.1 * matrix(c(1, 0, 0, 0, 1, 0, 0, 0, 1), ncol = 3),
  0.1 * matrix(c(3, 0, 0, 0, 2, 0, 0, 0, 1), ncol = 3),
  0.1 * matrix(c(1, 0.8, 0.2, 0.8, 1, 0.5, 0.2, 0.5, 1), ncol = 3)
)
opar <- par(mar = c(1, 1, 1, 1), mfrow = c(3, 3))
for(k in 1:3) {
  for(j in 1:3) {
    my_mean <- my_mean_vec[k, ]
    my_var <- my_var_list[[j]]
    plot(my_mean, pch = 15, col = "red", main = "données simulées")
    for(p in c(0.5, 1:9, 9.5)/10) {
      r <- sqrt(qchisq(p = p, df = 2))
      ellipses(my_mean, my_var, r, col="grey")
    }
    xr <- rnorm.acomp(n = 100, mean = my_mean, var = my_var)
    plot(xr, add = TRUE, pch = 19, cex = 0.5)
    if(j == 1)
      text(0.02, 0.5, TeX(sprintf(r'(\mu_{%g})', k)), cex = 0.95)
    if(k == 1)
      text(0.5, 0.97, TeX(sprintf(r'(\Sigma_{%g})', j)), cex = 0.95)
  }
}
}
```

A présent, on va s'intéresser au jeu de données des élections et on va vérifier si sa distribution pourrait être celle d'une loi normale dans le simplexe. Dans un premier temps, on va comparer notre jeu de données avec un jeu de données simulé issu d'une loi normale dans le simplexe. On remarque qu'un grand nombre d'observations de notre échantillon sort de la zone de confiance de niveau 95% et il est donc peu probable que notre échantillon soit issu d'une loi normale dans le simplexe.

```
opar <- par(mar = c(3, 3, 1, 1), mfrow = c(1, 2))
plot(clrInv(mean_clr), pch = 15,
```

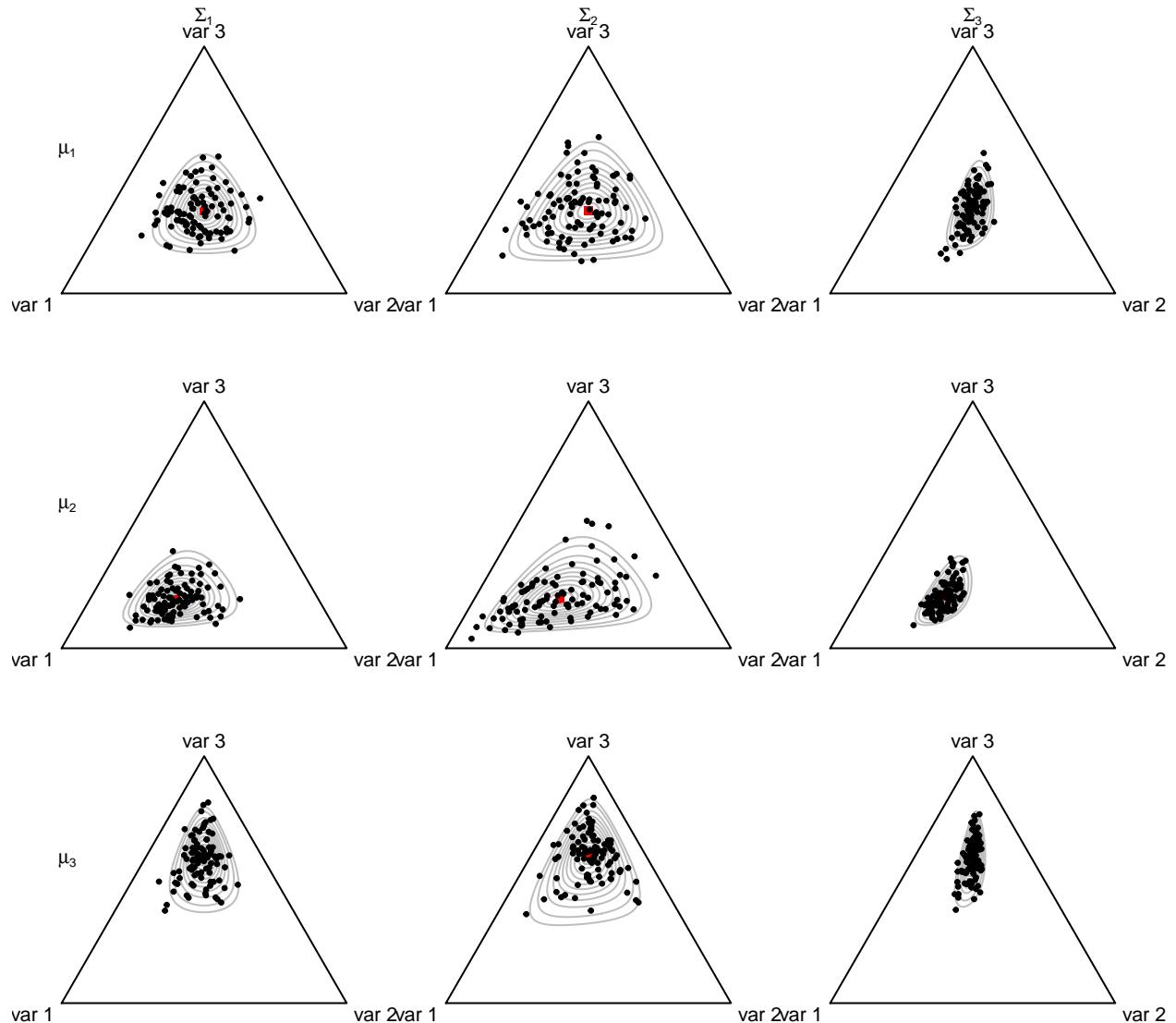


Figure 29: Distribution de lois normales dans le simplexe pour différents paramètres de la moyenne (en lignes) et de la matrice de variance-covariance dans l'espace clr (en colonnes)

```

    col = "red", main = "données simulées")
for(p in c(0.5, 1:9, 9.5)/10) {
  r <- sqrt(qchisq(p = p, df = 2))
  ellipses(clrInv(mean_clr), var_s, r, col="grey")
}
xr <- rnorm.acomp(n = 107, mean = clrInv(mean_clr), var = var_s)
plot(xr, add = TRUE, pch = 19, cex = 0.5)

plot(clrInv(mean_clr), pch = 15, col = "red",
      main = "données observées")
for(p in c(0.5, 1:9, 9.5)/10) {
  r <- sqrt(qchisq(p = p, df = 2))
  ellipses(clrInv(mean_clr), var_s, r, col="grey")
}
plot(comp_a, add = T, pch = 19, cex = 0.5)

```

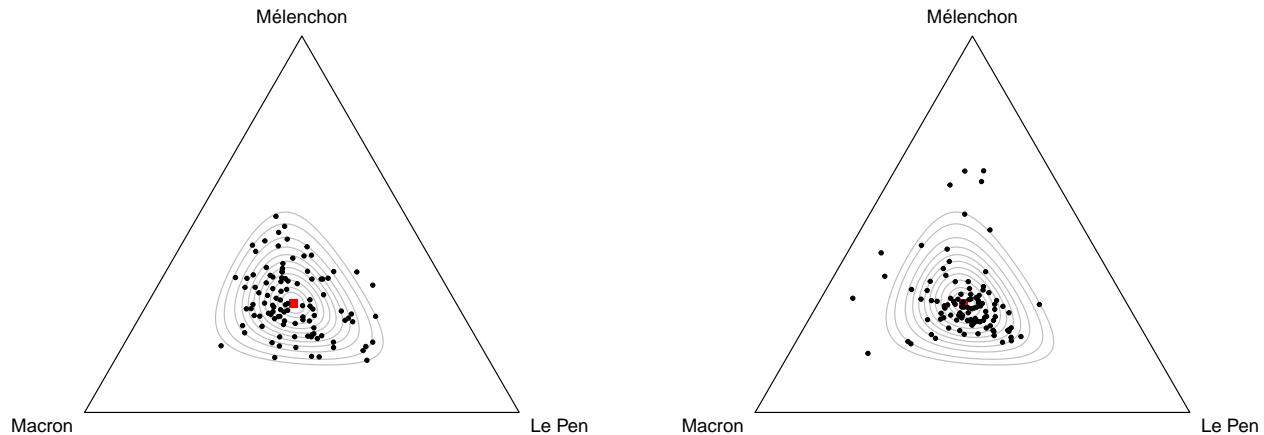


Figure 30: Données simulées selon une loi normale du simplexe (figure sur la gauche); données observées (figure sur la droite)

```
par(opar)
```

Une première solution pour tester si notre jeu de données est issu d'une loi normale dans le simplexe, est de regarder si les lois marginales sont toutes issues d'une loi normale. On peut utiliser la fonction `qqnorm()` qui s'applique sur des données de composition et calcule pour chaque log-ratio le qqplot associé. Dans cet exemple, les log-ratio où intervient la composante "Mélenchon" semblent très éloignés d'une loi normale, ce qui a l'air de confirmer que le jeu de données n'est pas issu d'une loi normale dans le simplexe.

```
qqnorm(comp_a)
```

On peut également utiliser un test basé sur la statistique d'énergie E (Székely and Rizzo 2005), implémenté dans le package `energy` et qui permet de tester la loi normale multivariée sur les données transformées dans l'espace ilr. Dans notre exemple, l'hypothèse que les données sont distribuées selon une loi normale multivariée est rejetée comme on pouvait s'y attendre :

```
energy::mvnorm.etest(ilr(comp_a), R = 199)
```

```
##
## Energy test of multivariate normality: estimated parameters
##
## data: x, sample size 107, dimension 2, replicates 199
## E-statistic = 5.9466, p-value < 2.2e-16
```

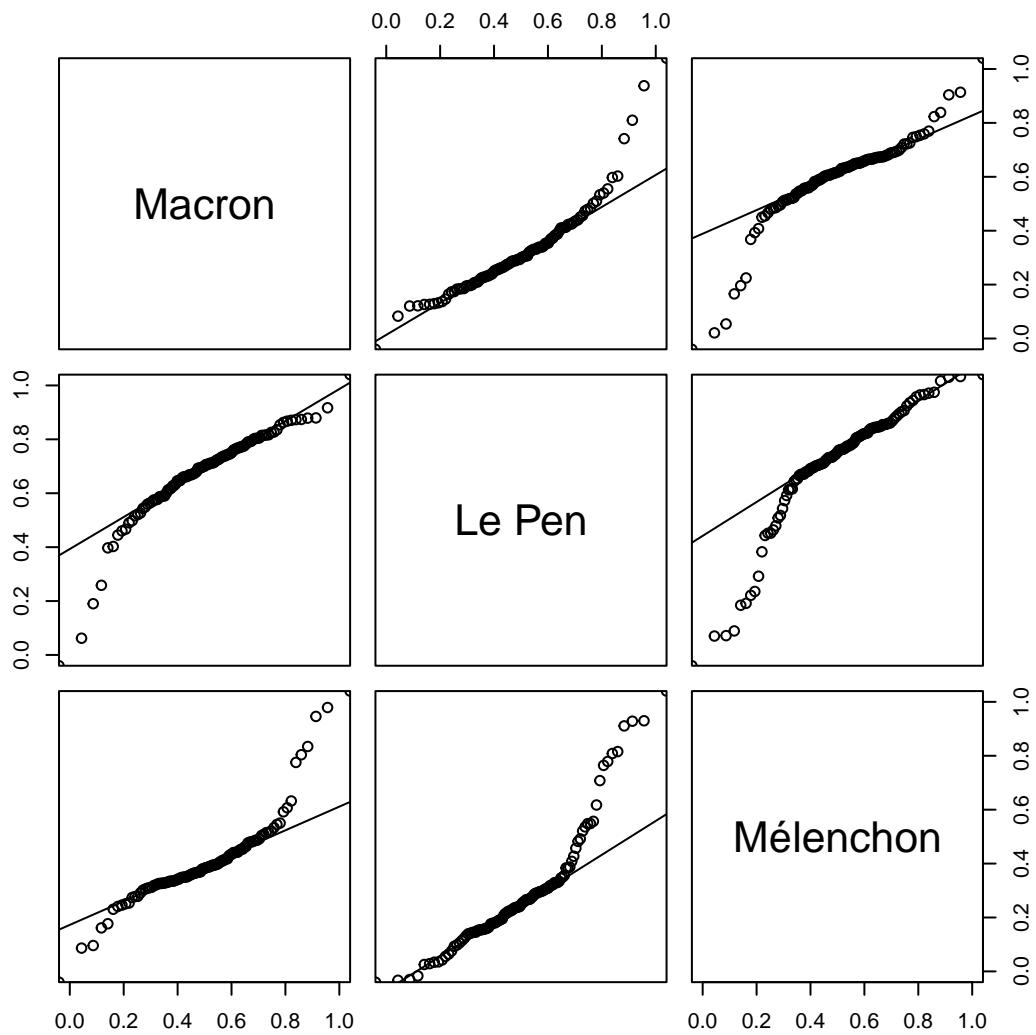


Figure 31: QQplot des lois marginales sur les log-ratio

4.4.3 Autres distributions

- Dirichlet : la fonction `rDirichlet.acomp()` permet de simuler des données selon une loi de Dirichlet et la fonction `dDirichlet()` retourne la fonction de densité. Dans l'exemple suivant, nous avons représenté la fonction de densité pour différents paramètres lorsque $D = 2$.

```
x <- seq(0, 1, 0.01)
my_comp <- acomp(cbind(x, 1 - x))
par(mfrow = c(1, 3))
plot(x, dDirichlet(my_comp, alpha = c(A = 0.3, B = 0.3)), type = "l", ylab = "f(x)",
     main = TeX(r"(\alpha_1=0.3, \alpha_2=0.3)"))
plot(x, dDirichlet(my_comp, alpha = c(A = 1, B = 1)), type = "l", ylab = "f(x)",
     main = TeX(r"(\alpha_1=1, \alpha_2=1)"))
plot(x, dDirichlet(my_comp, alpha = c(A = 2, B = 2)), type = "l", ylab = "f(x)",
     main = TeX(r"(\alpha_1=2, \alpha_2=2)"))
```

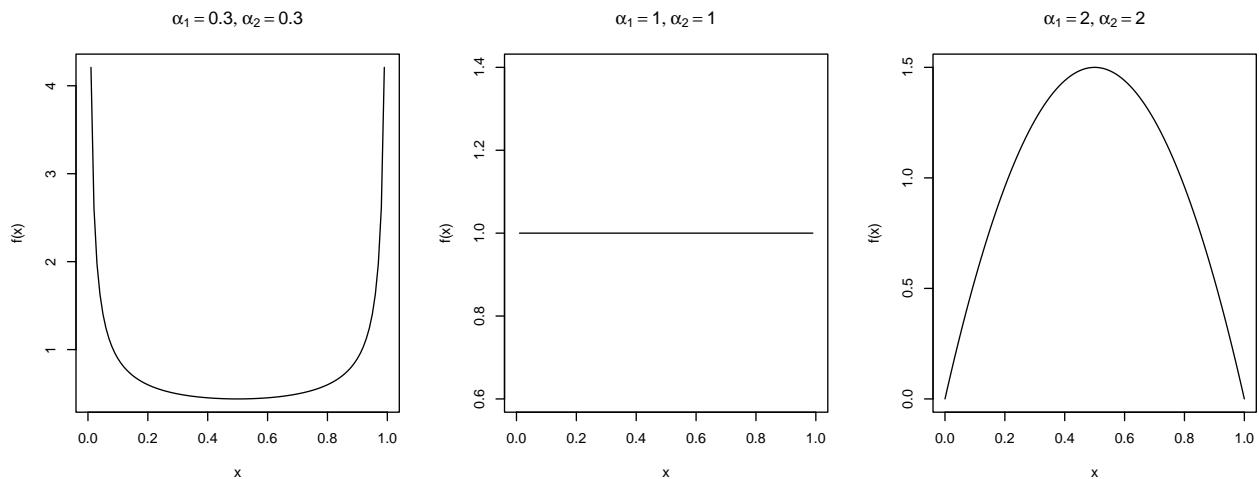


Figure 32: Fonction de densité de la loi de Dirichlet lorsque $D = 2$

On mentionnera la fonction `fitDirichlet()` qui permet d'estimer les paramètres de la loi de Dirichlet à partir d'un jeu de données. Par exemple, sur les données présidentielles :

```
fit_d <- fitDirichlet(comp_a)
```

Ce qui nous permet ensuite de représenter la distribution de la loi de Dirichlet avec les paramètres estimés.

```
opar <- par(mar = c(3, 3, 1, 1))
myalpha = fit_d$alpha
plot(comp_a)
plot(acomp(myalpha), pch = 16, col = "red", add = T)
aux <- seq(from=0, to=1, by=0.01)
myx <- expand.grid(x=aux, y=aux)
c60 <- cos(pi/3)
s60 <- sin(pi/3)
myfun <- function(x){
  y <- c(x[1]-x[2]*c60/s60, x[2]/s60)
  y <- c(1-sum(y),y)
  dd <- ifelse(any(y < 0), NA, dDirichlet(y, alpha = myalpha))
  return(dd)
}
dx <- apply(myx, 1, myfun)
```

```

dim(dx) <- c(101, 101)
contour(dx, asp = 1, levels = quantile(
  dDirichlet(rDirichlet.acomp(1000, myalpha), myalpha),
  c(0.5, 1:9, 9.5)/10, na.rm = T),
  add = TRUE, col = "grey")

```

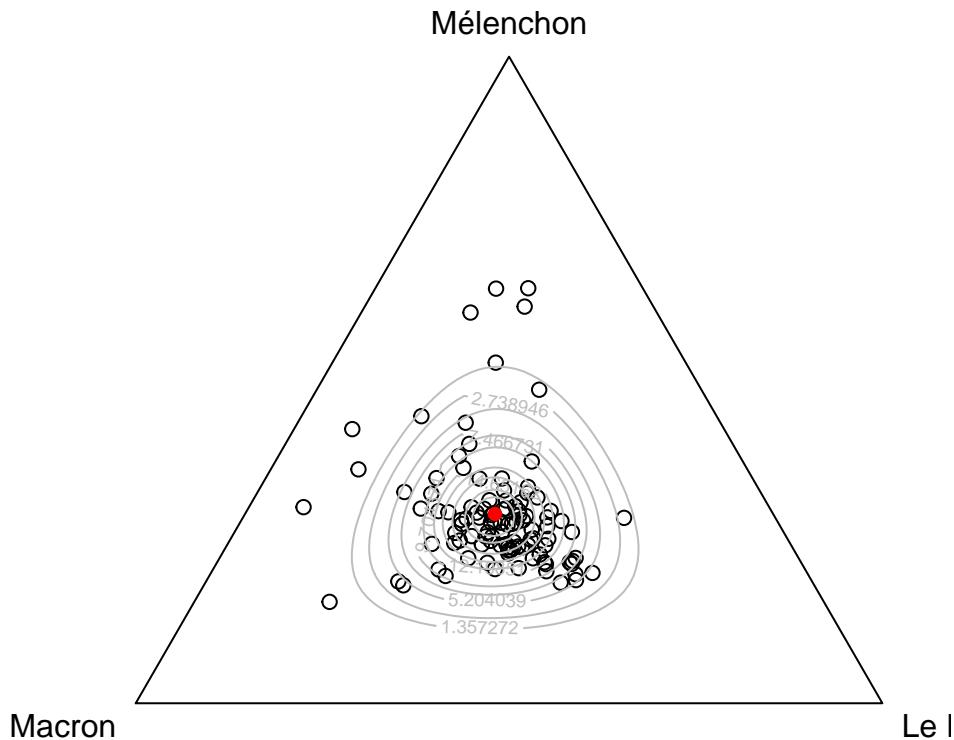


Figure 33: Estimation d'une loi de Dirichlet sur les données d'élection

```
par(opar)
```

- Student : on va reprendre ici l'algorithme donné dans le chapitre 7 avec $\mu^* = (0, 0)$, $\Sigma^* = \begin{pmatrix} 2 & -1.5 \\ -1.5 & 2 \end{pmatrix}$, $\nu = 5$ et $V = \begin{pmatrix} -1/\sqrt{2} & -1/\sqrt{6} \\ 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & 2/\sqrt{6} \end{pmatrix}$. On simule \mathbf{Z} :

```

Z <- mvrnorm(n = 100, mu = c(0, 0),
  Sigma = matrix(c(2, -1.5, -1.5, 2), 2, 2))

```

On simule $u \sim \chi^2_\nu$:

```
u <- rchisq(1, df = 5)
```

On calcule $\mathbf{Y} = \sqrt{\nu/\mu}Z + \mu^*$

```
Y <- sqrt(5 / u) * Z
```

On calcule l'ilr inverse de \mathbf{Y}

```
X <- ilrInv(Y)
```

Enfin, on représente les données dans le simplexe :

```
plot(X)
```

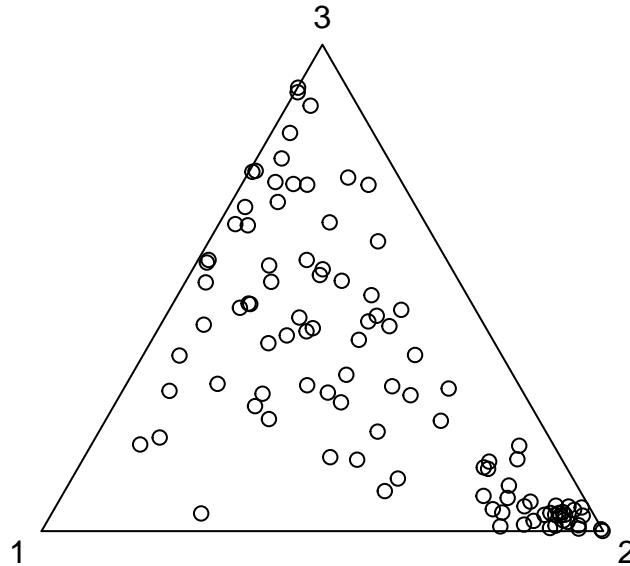


Figure 34: Données simulées selon une loi de Student multivariée sur le simplexe

- On notera les fonctions `rAitchison()` et `dAitchison()` qui permettent de simuler et calculer la fonction de densité d'une loi d'Aitchison.

Exercice 3 : utiliser les outils présentés dans cette section sur le jeu de données que vous avez choisi.

5 Analyse exploratoire (avec une approche compositionnelle)

5.1 Matrice de diagramme ternaire

Nous avons vu dans la section précédente comment représenter le diagramme ternaire sur des données de compositions où $D = 3$. Lorsque $D > 3$, on peut représenter des matrices de diagramme ternaires, où la cellule (i, j) de la matrice représente le diagramme ternaire de la sous-composition $(x_i, x_j, *)$ avec plusieurs façons de définir la troisième composante. Prenons par exemple les parts de marché de cinq constructeurs automobiles obtenues sur une période de 152 mois consécutifs. Avant de s'intéresser au diagramme ternaire, nous allons représenter chaque composante comme étant une série temporelle à l'aide de la fonction `zoo()` du package `zoo` (Zeileis and Grothendieck 2005):

```
library(codareg)
data(BDDSegX)
Y_s <- BDDSegX[, c("S_A", "S_B", "S_C", "S_D", "S_E")]
colnames(Y_s) <- c("A", "B", "C", "D", "E")
Y_s_zoo <- zoo::zoo(as(Y_s, "matrix"), BDDSegX[, "Date"])
Y_s <- acomp(Y_s)
```

On observe que les parts de B et C dominent le marché et ont globalement eu tendance à augmenter les derniers mois, B ayant été pendant quelques mois au-dessus de C. Le constructeur A était au début à peu près au même niveau que E, puis sa part a augmenté possiblement au détriment de E dont la part est celle semble avoir le plus baissé au cours du temps. Le constructeur D a également diminué sa part de marché au fil du temps.

```
D_market <- ncol(Y_s)
hues <- seq(15, 375, length = D_market + 1)
```

```

my_col <- hcl(h = hues, l = 65, c = 100)[1:D_market]
par(las = 1, mar = c(3, 4, 1, 1))
plot(Y_s_zoo, screens = 1, col = my_col, lwd = 2,
     ylab = "Market share automobile", xlab = "Time", ylim = c(0, 0.5))
legend("topleft", legend = c("A", "B", "C", "D", "E"), lty = 1, col = my_col,
       horiz = T, cex = 0.75, lwd = 2)

```

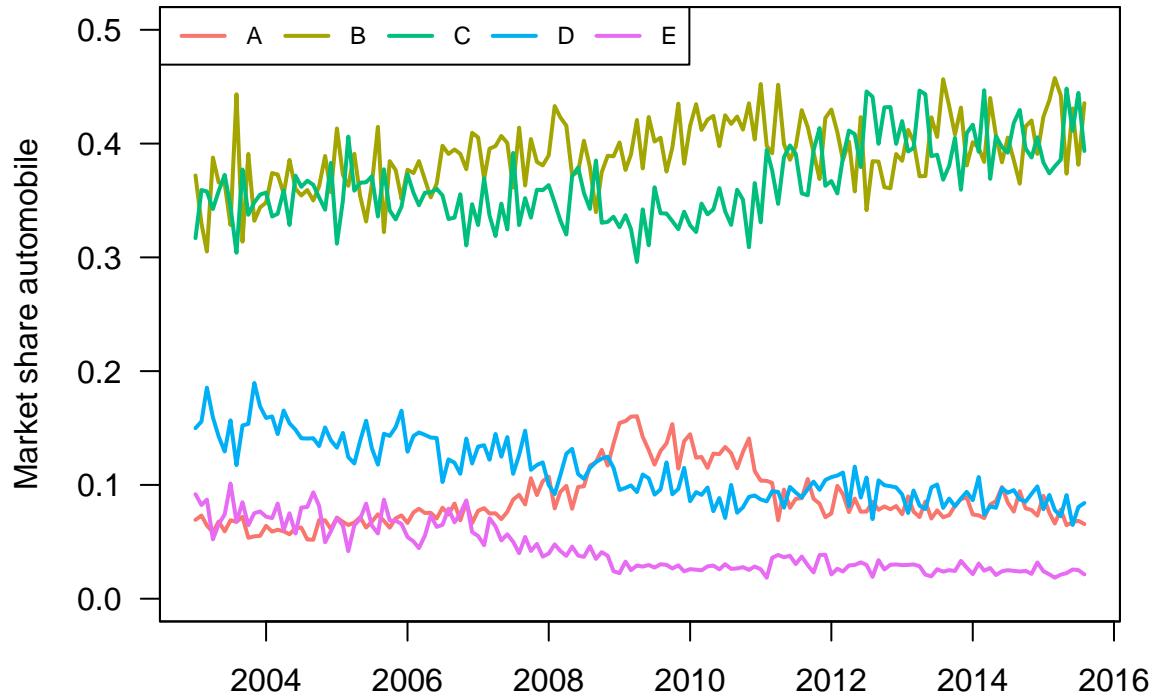


Figure 35: Série des parts de marchés automobile des 5 marques de constructeurs étudiées

Pour représenter la matrice de diagramme ternaire, on utilise directement la fonction `plot()` du package `compositions`. Si on choisit l'option `margin="acomp"` (option par défaut), la troisième composante (représentée par `"*"`) est calculée comme étant la moyenne géométrique de toutes les parts hormis les deux premières. On peut regarder les triangles ligne par ligne. Si on regarde par exemple la première ligne, on constate que la part A est clairement dominée par la part B (triangle $(A, B, *)$) et la part C (triangle $(A, C, *)$). La moyenne géométrique des trois autres composantes dans les deux premiers triangles n'est pas particulièrement grande ceci s'expliquant par le fait que dans la dernière composante, il y a une composante très forte (B ou C), les deux autres étant relativement faibles. En revanche, le triangle $(A, D, *)$ montre que A et D ont à peu près eu les mêmes parts de marché, celles-ci étant significativement plus faible que la moyenne géométrique des trois autres parts. Enfin, le triangle $(A, E, *)$ montre que la part de E est un peu plus élevée que celle de A, mais les deux parts A et E sont négligeables par rapport à la moyenne géométrique des trois autres parts.

```
plot(Y_s, margin = "acomp", pch = 3, cex = 0.6)
```

Remarque : il est possible de centrer les données avec l'option `center = TRUE` en utilisant la formule de centrage dans la géométrie d'Aitchison. Par ailleurs, si on choisit l'option `margin="rcomp"`, la troisième composante est prise comme étant la somme arithmétique de toutes les autres composantes.

5.2 Matrice de variation

La matrice de variation \mathbf{T} s'obtient avec la fonction `variation()`:

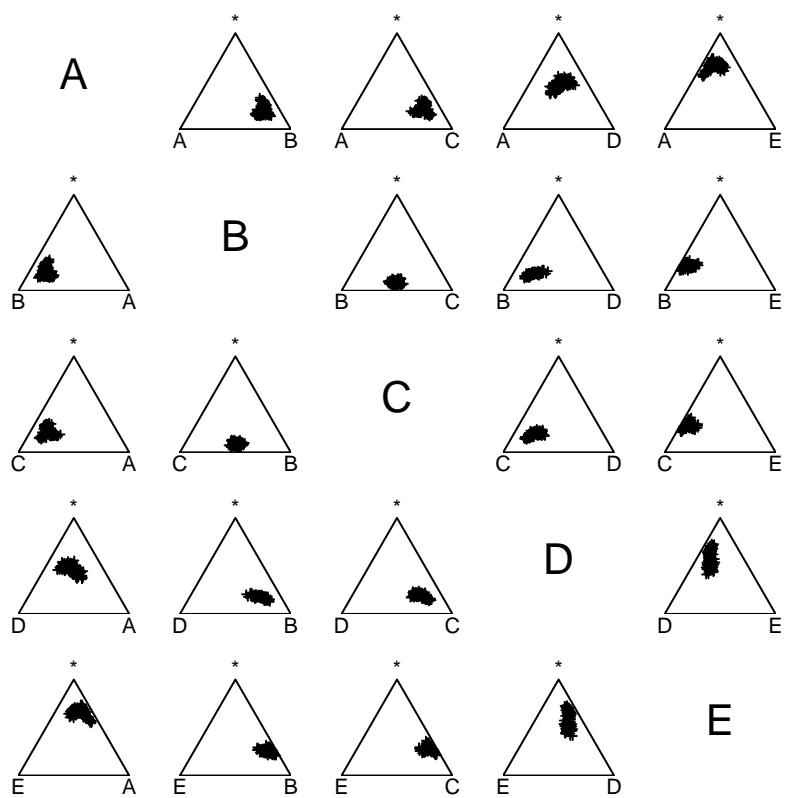


Figure 36: Matrice de diagramme ternaires, avec la troisième composante égale à la moyenne géométrique des autres composantes

```
var_T <- variation(Y_s)
```

La cellule (i, j) est calculée de la façon suivante $\tau_{ij} = \text{var}(\log \frac{x_i}{x_j})$. Par exemple pour la cellule (1, 2) :

```
var(log(Y_s[, 1] / Y_s[, 2]))
```

```
## [1] 0.06370582
```

Il est facile de voir que \mathbf{T} est symétrique sachant que $\log \frac{a}{b} = -\log \frac{b}{a}$ et que $\text{var}(-c) = \text{var}(c)$.

Remarque : si $x_i = kx_j$, avec k un scalaire, alors $\tau_{ij} = 0$. Autrement dit, une faible valeur de τ_{ij} implique une variance faible de $\log \frac{x_i}{x_j}$ et donc une “bonne proportionnalité” entre x_i et x_j . Ainsi, on observe que la valeur de τ_{ij} entre A et E est la plus élevée (on avait effectivement remarqué que les parts de A et de E étaient très proches au début, puis elles se sont éloignées l’une de l’autre). De même la valeur de τ_{ij} entre B et C est la plus faible (on avait effectivement que B et C se suivaient au cours du temps).

En utilisant la fonction `boxplot()` sur un objet `acomp`, on représente la matrice des boîtes à moustaches des log-ratio par paires.

```
boxplot(Y_s)
```

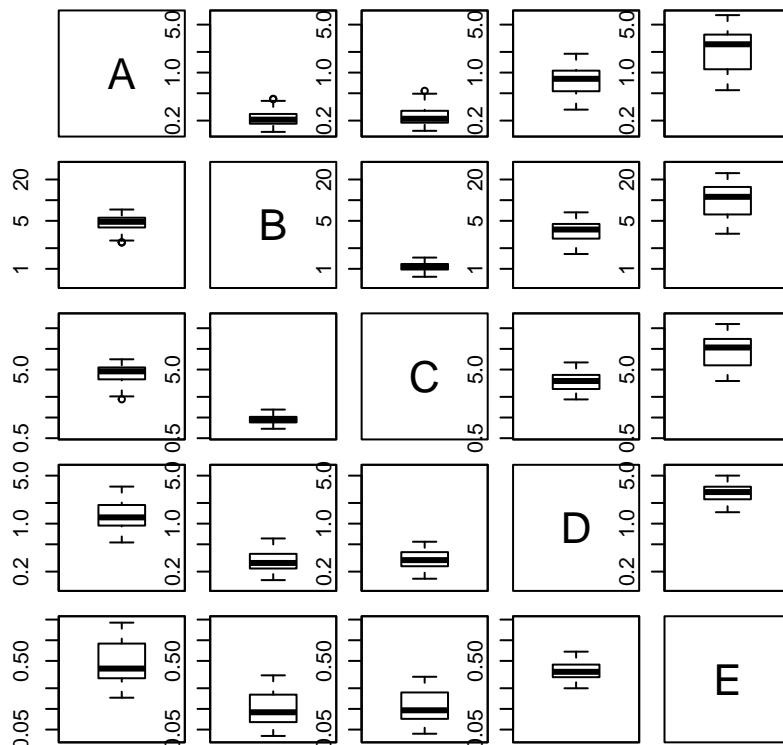


Figure 37: Matrice des boîtes à moustaches des log-ratio par paires

Enfin, la variance totale $VarTot$ est définie comme $\frac{1}{D^2} \sum \sum_{j < j'} \tau_{jj'}$:

```
VarTot <- sum(var_T * upper.tri(var_T)) / D_market ^ 2
```

La variance $VarTot$ peut également être vue comme la moyenne des variances de chaque composante `clr` et nous permet de calculer les contributions de chaque part :

```
var_j <- diag(var(clr(Y_s)))
var_j / sum(var_j)
```

```
##          A          B          C          D          E
## 0.28980668 0.07681380 0.08989247 0.08341790 0.46006915
```

Ainsi, la part A et la part E sont les deux parts qui contribuent le plus à la variance totale.

Remarque : dans la chapitre 4, des poids de pondération ont été introduits. De nombreuses fonctions du package `easyCODA` permettent de prendre en compte ces poids de pondération. Nous verrons certaines de ces fonctions dans la suite.

5.3 Apprentissage automatique dans l'espace des variables

5.3.1 Classification Ascendante Hiérarchique (CAH)

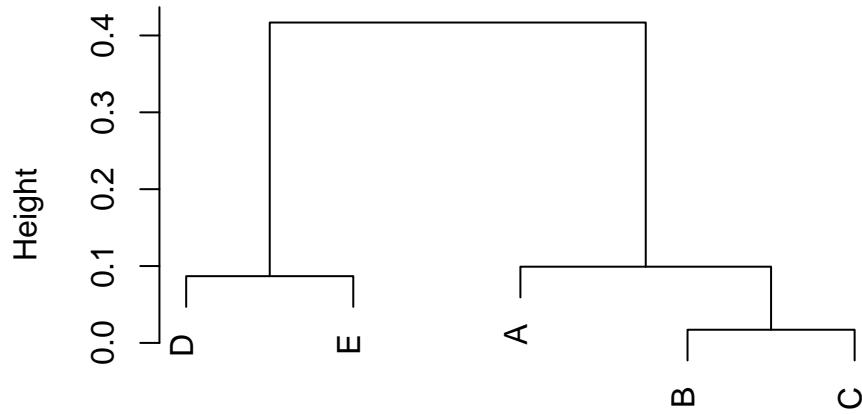
En assimilant la matrice de variation à une mesure de distance entre les variables, on peut lui appliquer une méthode de classification ascendante hiérarchique (CAH). Cela se fait avec la fonction de base `hclust()`, appliquée à la matrice de variation transformée en un objet de type `dist`. Il existe plusieurs variantes de l'algorithme et la méthode de Ward est celle que nous avons choisie par défaut (voir par exemple https://www.math.univ-toulouse.fr/~sdejean/PDF/stat_multidim_sdejean.pdf pour plus d'information sur la CAH).

Le résultat montre que B et C ont été les premiers à être regroupés, puis D et E, puis le groupe (B, C) avec A. Par ailleurs, la décision de définir à quel niveau il faut couper le dendrogramme est subjective, mais en général, il s'agit de choisir le niveau correspondant à un saut ayant la hauteur la plus élevée; dans notre exemple, cela suggère de garder deux groupes de variables :

- Les constructeurs D et E forment ainsi la première classe
- Les constructeurs A, B et C forment la deuxième classe

```
dd <- as.dist(var_T)
hc <- hclust(dd, method = "ward.D")
plot(hc, xlab = "Méthode de Ward", sub = "")
```

Cluster Dendrogram



Méthode de Ward

Figure 38: Dendrogramme associé à la CAH réalisée sur la matrice de variation des données de constructeurs automobiles.

A partir des regroupements obtenus avec la CAH, il est possible de re-définir de nouvelles coordonnées ilr (Log Ratio Balance). Pour faire cela, on peut utiliser la fonction `balance()` de la manière suivante :

```
head(balance(X = Y_s, expr = ~ (D/E)/(A/(B/C))))
```

```
##          DE/ABC      D/E      A/BC      B/C
## [1,] -0.5916379 0.3464444 -1.307002 0.11336354
## [2,] -0.6516259 0.4505016 -1.264914 -0.06084049
## [3,] -0.4489922 0.5344976 -1.335969 -0.11282554
## [4,] -0.8487391 0.7903665 -1.496492 0.08769768
## [5,] -0.8358505 0.5494637 -1.366694 0.01444637
## [6,] -0.7838092 0.3941084 -1.493843 -0.01486511
## attr(),"class")
## [1] "rmult"
```

Ce qui revient à définir la matrice de signe de la partition binaire suivante :

```
sign_binary <- matrix(c(-1, -1, -1, 1, 1,
                         0, 0, 0, 1, -1,
                         1, -1, -1, 0, 0,
                         0, 1, -1, 0, 0), byrow = T, ncol = 5)
```

Et de construire la matrice de contraste V :

```
V_binary <- sign_binary
for (j in 1:nrow(V_binary)) {
  card_J_plus <- length(which(sign_binary[j, ] == 1))
  card_J_moins <- length(which(sign_binary[j, ] == -1))
  alpha <- sqrt(card_J_plus * card_J_moins / (card_J_plus + card_J_moins))
  row_j <- V_binary[j, ]
  row_j[row_j == 1] <- 1 / card_J_plus
  row_j[row_j == -1] <- -1 / card_J_moins
  V_binary[j, ] <- alpha * row_j
}
```

Et d'appliquer la formule $V^T \log(x)$ observation par observation. Par exemple, pour la 1ère observation :

```
V_binary %*% log(as.numeric(Y_s[1, ]))
```

```
##      [,1]
## [1,] -0.5916379
## [2,]  0.3464444
## [3,] -1.3070022
## [4,]  0.1133635
```

Appliquer une méthode statistique sur une transformation ilr plutôt qu'une autre donnera les mêmes résultats lorsqu'on revient dans le simplexe. En revanche, l'analyse descriptive devrait être plus intuitive car elle se base sur une balance qui a une signification. Par ailleurs, définir les balances en utilisant des connaissances d'un expert métier (par exemple si on sait que les modèles des voitures du constructeur B et C utilisent la même technologie, mais qui est différente de celle de A), il peut être judicieux de créer les balances à partir de cette information.

5.4 Apprentissage automatique dans l'espace des individus

5.4.1 L'Analyse en Composantes Principales (ACP)

Nous avons déjà vu la fonction `dist()` du package `compositions` qui, appliquée à un object `acomp`, calcule la distance d'Aitchison entre chaque paires d'individus. On peut tout aussi bien calculer la distance euclidienne sur les données transformées dans l'espace `clr` par exemple :

```

clr_market <- clr(Y_s)
dist_market <- dist(clr_market)

```

Une fois la matrice de distance calculée, il est possible d'utiliser la fonction `cmdscale()` qui permet de réaliser un échelonnement multidimensionnel (présenté au chapitre 4) dont on peut représenter la projection sur les deux composantes choisies. On peut ajouter les CLR comme variables supplémentaires en utilisant leurs coefficients de régression sur les deux dimensions de la solution.

```

mds <- cmdscale(dist_market)
x <- mds[, 1]
y <- mds[, 2]
plot(x, y, pch = 16, cex = 0.7, xlab = "", ylab = "", main = "",
      xlim = c(-1, 1), ylim = c(-1, 1), asp = 1)
abline(h = 0, v = 0, lty = 2)
for(k in 1:5) {
  coeff_clr <- coefficients(lm(clr_market[, k] ~ x + y -1))
  points(coeff_clr[1], coeff_clr[2], col = "red", pch = 15)
  text(coeff_clr[1], coeff_clr[2], LETTERS[k], col = "red", pos = 3)
}

```

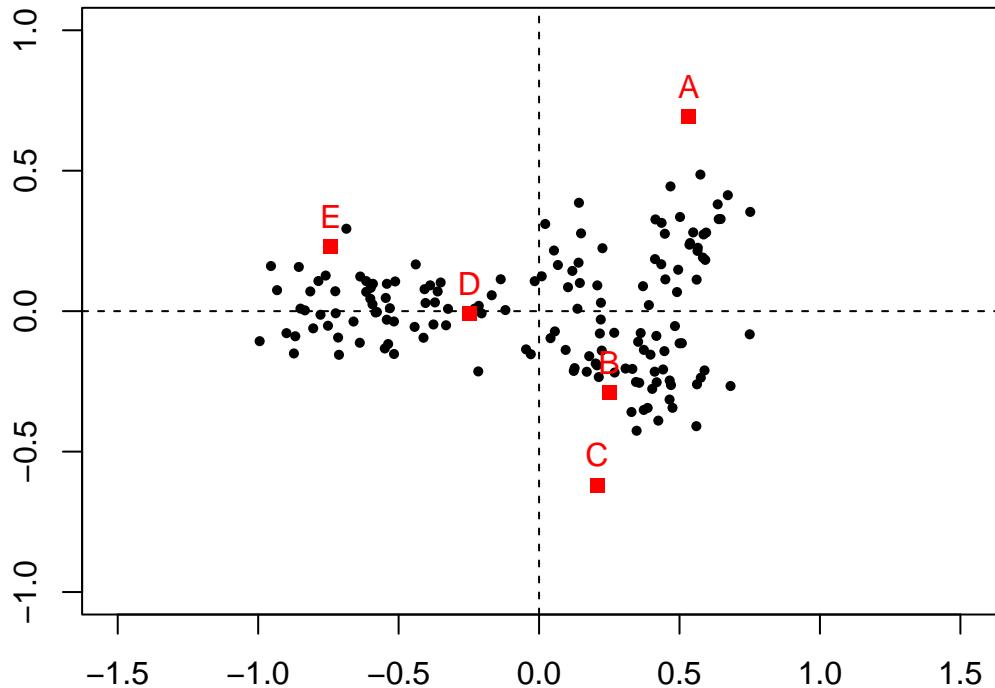


Figure 39: MDS des parts de marché automobiles

Le biplot est majoritairement utilisé pour illustrer le résultat de l'analyse en composantes principales (ACP). Pour les données de composition, l'ACP est effectuée sur les coordonnées `clr`. Pour cela, on peut utiliser directement la fonction `princomp()` sur un objet `acomp`, ensuite la fonction `biplot()` permet de représenter la projection des individus et des variables sur les deux premières composantes de l'ACP. Par ailleurs, nous avons également utilisé la fonction `screeplot()` qui permet de représenter la variance expliquée par chaque axe de l'ACP et qui nous confirme ici que deux axes sont probablement suffisants :

```

pca <- princomp(Y_s)
par(mfrow = c(1, 2), las = 1)
screeplot(pca)

```

```
biplot(pca, cex = 0.5)
abline(h = 0, v = 0, lty = 2)
```

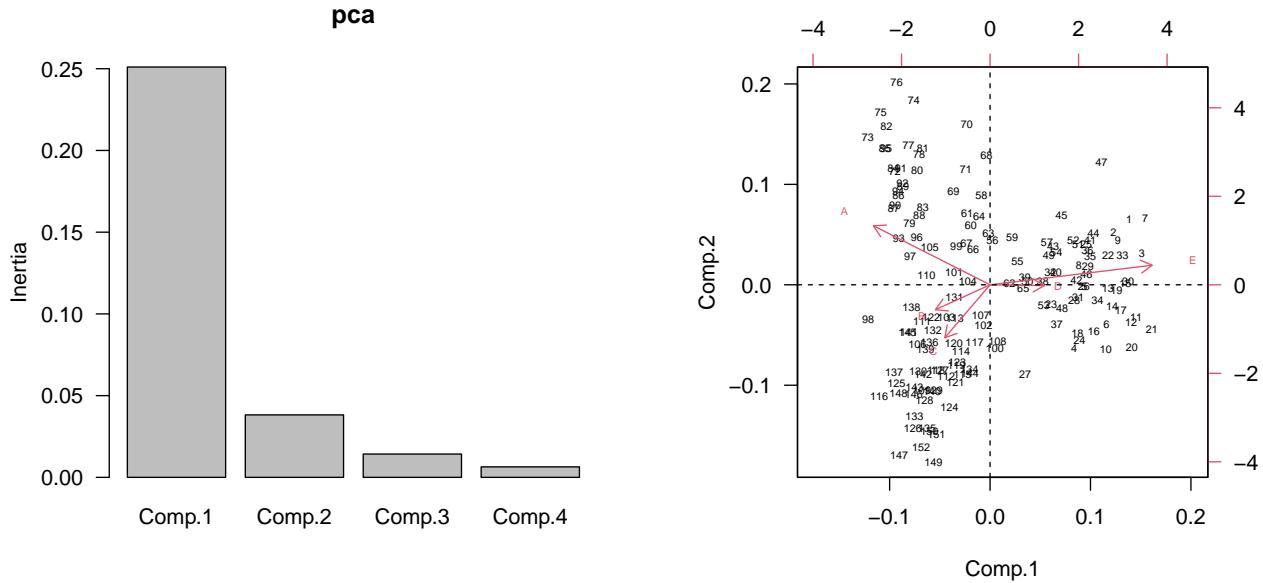


Figure 40: Biplot obtenu après une ACP réalisée sur les coordonnées CLR

L'interprétation du biplot est très différente de celle réalisée classiquement sur des données non compositionnelles. En effet, on s'intéresse ici d'une part aux rayons (*ray* en anglais) qui correspondent aux longueurs entre l'origine et les coordonnées des variables (\bar{OA} par exemple) dont le carré vaut à peu près $Var(\log(\frac{x_A}{g(x)}))$. D'autre part, on s'intéresse aux connections (*link* en anglais) ou longueurs entre les coordonnées des variables (\bar{AE} par exemple), dont le carré vaut à peu près $Var(\log(\frac{x_A}{x_E}))$.

A partir de ces longueurs, on peut faire les remarques suivantes :

- si deux variables sont proportionnelles ($x_i = kx_j$), leur log-ratio est quasi-constant et leurs coordonnées dans le biplot doivent être très proches. C'est le cas des constructeurs B et C dans notre exemple.
- si la longueur entre deux composantes est grande, le log-ratio entre les deux parts est volatile. De plus, si on observe trois rayons orientés vers des directions différentes à 120 degrés environ (par exemple \bar{OA} , \bar{OC} et \bar{OE}) et avec de fortes longueurs, alors le diagramme ternaire des trois sous-composition aura des points éparpillés dans le triangle.
- les angles entre les connections doivent approximer la corrélation entre les log-ratio
 - si deux connections sont perpendiculaires, alors les deux log-ratio ne sont pas corrélées (par exemple $\log(\frac{C}{B})$ et $\log(\frac{D}{E})$)
 - si trois composantes sont parfaitement alignées, alors les log-ratio sont fortement corrélés (dans notre exemple, B, D et E). Dans ce cas, une représentation de ces sous-composition devraient former une forme sur une seule dimension, autrement dit, ces sous-compositions sont colinéaires.

On notera la fonction `loadings()` qui permet de récupérer les coordonnées des axes principaux dans l'espace CLR. Cela peut nous permettre de représenter les axes dans le simplexe. Ici, on transforme les coordonnées de l'axe 1 dans le simplexe, avant de le représenter dans le diagramme ternaire avec la fonction `straight()`. L'idée est d'identifier les diagrammes où la direction de l'axe 1 s'ajuste le mieux aux données (les diagrammes $(E, D, *)$, $(B, D, *)$ par exemple)

```
axe_1 <- clrInv(loadings(pca)[, 1])
plot(acomp(Y_s), pch = 16, cex = 0.5) # , margin = "acomp"
straight(mean(acomp(Y_s)), axe_1)
```

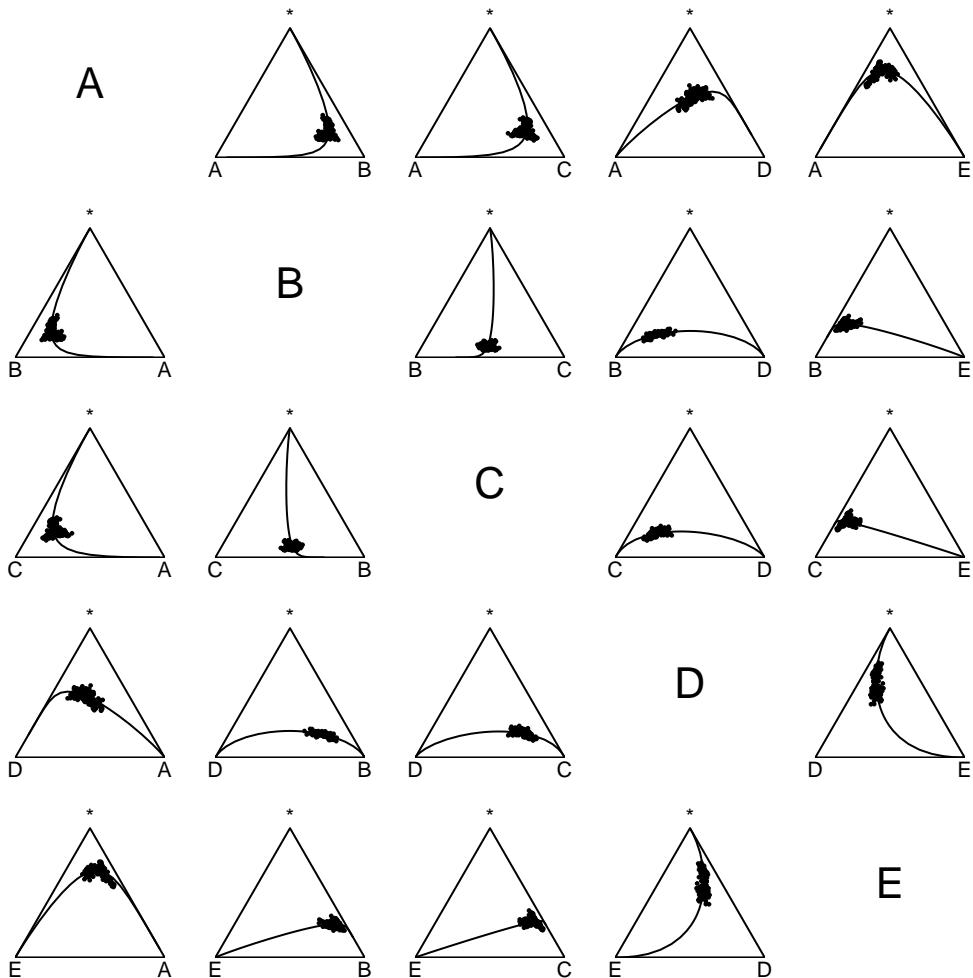


Figure 41: Représentation de l'axe 1 de l'ACP dans le simplexe

Pour prendre en compte les pondérations comme dans le chapitre 4, on utilisera les fonctions du package **easyCODA**. Ici, on reprend l'exemple du chapitre 4 sur les données archéologiques :

```
data(cups)
par(mfrow = c(1, 2))
PLOT.LRA(LRA(cups, weight = FALSE), main = "LRA non pondéré")
PLOT.LRA(LRA(cups, weight = TRUE), main = "LRA pondéré")
```

Remarque : on citera la fonction **CA()** du package **easyCODA** qui permet de faire une analyse des correspondances

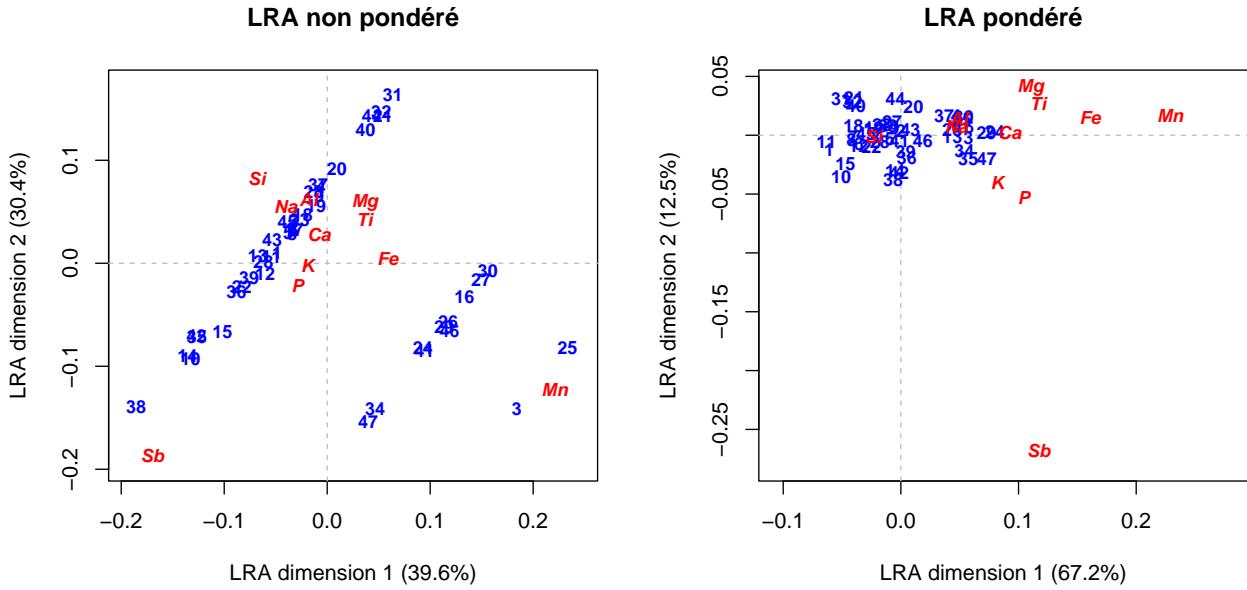


Figure 42: Biplots non-pondéré et pondéré du tableau de données archéologique

5.4.2 Classification sur les individus

Il est possible d'appliquer une méthode de classification sur les individus à partir de la projection des observations obtenus à l'aide de la MDS ou de l'ACP. Prenons par exemple les coordonnées des observations obtenus dans la MDS. On applique ici une classification à partir de l'algorithme des K -means (voir par exemple https://www.math.univ-toulouse.fr/~sdejean/PDF/stat_multidim_sdejean.pdf pour plus de détails). Cette méthode est implémentée dans la fonction `kmeans()`. Contrairement à la classification ascendante hiérarchique, l'utilisateur doit spécifier le nombre de classes choisies. Ici, l'analyse des biplot précédents laisse suggérer une répartition des observations en trois groupes.

```
clust <- kmeans(mds, 3)$cluster %>%
  as.factor()
```

Chaque groupe détecté par la classification correspond à une période de temps spécifique. On peut également représenter les groupes détectés dans un diagramme ternaire. Pour cela, nous avons choisi le diagramme des sous-compositions (A, C, E) dont l'ACP a révélé qu'il pouvait montrer les observations plus réparties et nous avons centré les données. Finalement, on constate que chaque classe est caractérisée par une composante : dans la première classe, les observations ont une valeur de E au-dessus de la moyenne géométrique, dans la deuxième classe, c'est le constructeur A qui a une valeur supérieure à la moyenne géométrique et enfin dans la troisième classe, c'est le constructeur C qui a une valeur supérieure à la moyenne géométrique.

```
par(las = 1, mar = c(3, 4, 1, 1), mfrow = c(1, 2))
plot(Y_s_zoo, screens = 1, col = my_col, lwd = 2,
     ylab = "Market share automobile", xlab = "Time", ylim = c(0, 0.5))
legend("topleft", legend = c("A", "B", "C", "D", "E"), lty = 1, col = my_col,
       horiz = T, cex = 0.75, lwd = 2)
abline(v = BDDSegX[c(60, 99), "Date"], lty = 2, col = "red")
my_col_2 <- hcl(h = hues, l = 65, c = 100)[1:3]
lines(BDDSegX[c(1, 60), "Date"], c(0, 0), lwd = 4, col = my_col_2[3])
lines(BDDSegX[c(61, 99), "Date"], c(0, 0), lwd = 4, col = my_col_2[2])
lines(BDDSegX[c(100, nrow(BDDSegX)), "Date"], c(0, 0), lwd = 4, col = my_col_2[1])

plot(Y_s[, c(1, 3, 5)], center = T, col = my_col_2[clust])
lines(Y_s[, c(1, 3, 5)])
```

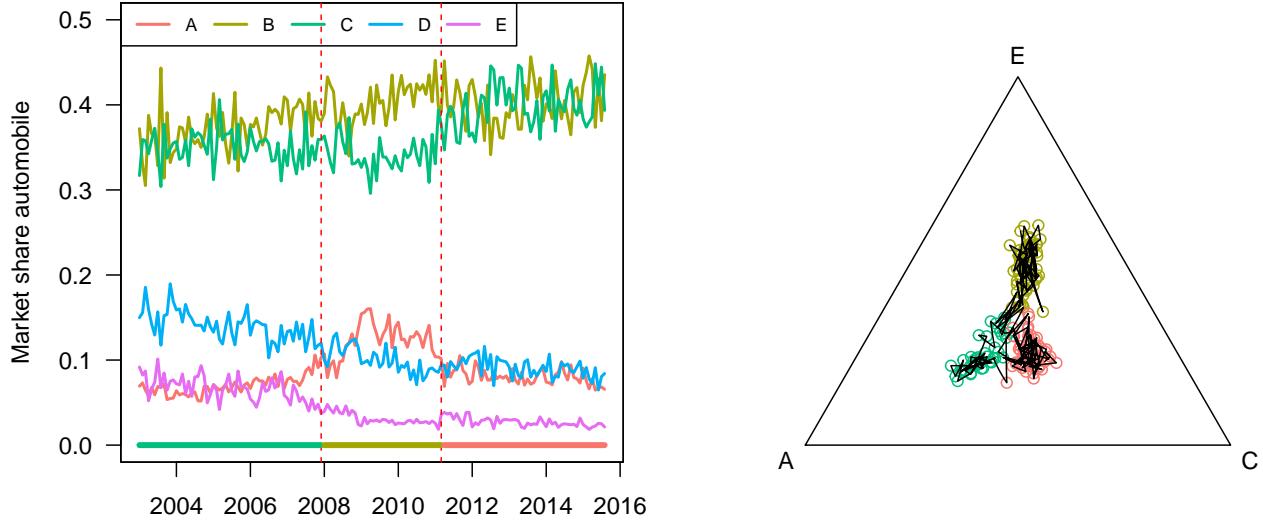


Figure 43: Représentation des trois groupes détectés dans le graphique des séries temporelles (à gauche) et dans le diagramme ternaire centré

Remarque : on citera la fonction `WARD()` de la librairie `easyCODA` qui prend en compte les poids de pondération.

5.5 Détection des atypiques

Jusqu'à présent, nous n'avons pas tenu compte de la présence d'observations atypiques. Nous présentons ici une méthode basé sur la méthode de sélection de composantes invariantes (ICS en anglais) qui permet de détecter ou non la présence d'atypiques. On utilise ici la fonction `ics.outlier()` de la librairie `ICSOOutlier` (Archimbaud, Nordhausen, and Ruiz-Gazen 2018), appliquée directement sur les coordonnées ilr du jeu de données sur les élections présidentielles. Cela permet de détecter les départements où un des candidats est sorti largement en tête.

```
library(ICSOOutlier)
my_ics <- ics2(ilr_a)
icsOutlier <- ics.outlier(my_ics, level.dist = 0.05, mDist = 50, ncores = 1)
op <- par(oma = c(1, 1, 1.4, 1), mar = c(3.3, 3.3, 1, 0.7),
          las = 1, mgp = c(2.25, 1, 0), mfrow = c(1, 2))
plot(icsOutlier@ics.distances, main = "",
     pch = ifelse(icsOutlier@ics.distances > icsOutlier@ics.dist.cutoff, 16, 3),
     xlab = "Observation Number", ylab = "ICS distances", cex.main = 2)
abline(h = icsOutlier@ics.dist.cutoff)
plot(comp_a, col = ifelse(icsOutlier@ics.distances > icsOutlier@ics.dist.cutoff,
                           "red", "grey"))
```

Exercice 4 : utiliser les outils présentés dans cette section sur le jeu de données que vous avez choisi.

6 Régression compositionnelle

Dans cette section, nous allons illustrer les outils présentés dans le Chapitre 8.

6.1 Régression avec variable explicative compositionnelle

Nous allons utilisées les données portant sur un échantillon de 7713 individus de nationalité chinoises (source : <https://www.cpc.unc.edu/projects/china>). Ici la variable dépendante est l'indice de masse corporelle et

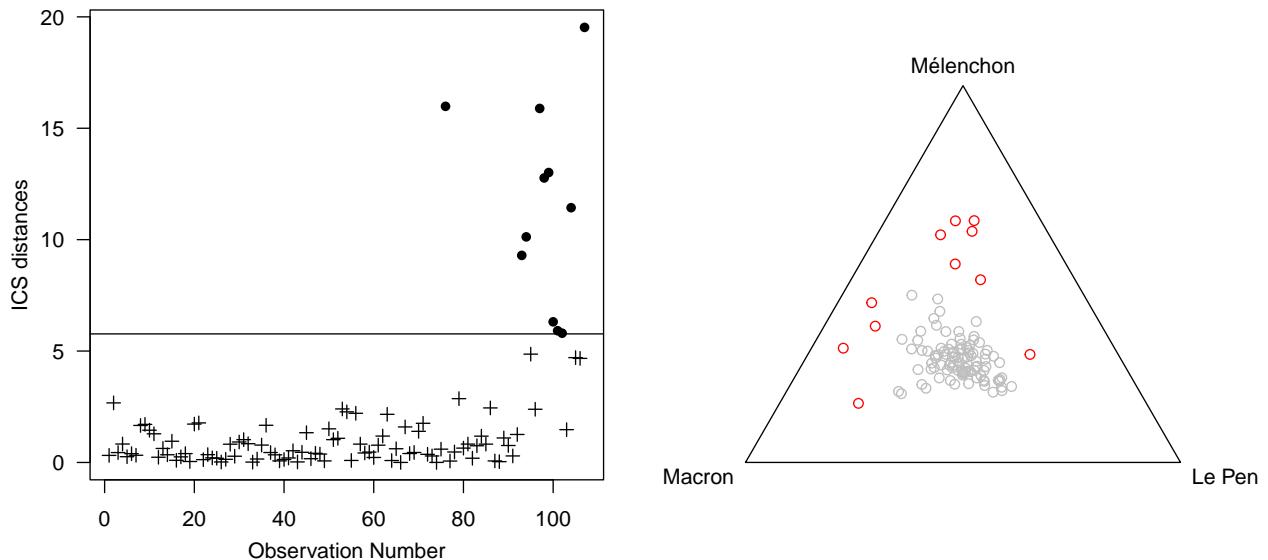


Figure 44: Détection des valeurs atypiques des élections présidentielles en utilisant la méthode ICS

nous allons utiliser comme variables explicatives la composition journalière moyenne de calorie réparties en trois composantes : les protéines (variable VC), les lipides (variable VF), les glucides (variable VP).

La première opération est de construire la composition :

```
data("CHNS11")
CHNS11 <- CHNS11 %>%
  rename(proteines = VC, lipides = VF, glucides = VP)
X_compo <- acomp(CHNS11[, c("proteines", "lipides", "glucides")])
```

On va représenter la variable X dans un diagramme ternaire et représenter les points conditionnellement à la variable Y découpée selon les classes suivantes :

- inférieur à 18.5 : sous-poids
- de 18.5 à 25 : poids normal
- de 25 à 30 : surpoids
- de 30 à 35 : obésité modérée
- supérieur à 35 : obésité sévère

D'après la figure, il n'est pas évident de déceler des différences de distribution de la variable de composition selon les différentes classes de Y .

```
bk <- c(0, 18.5, 25, 30, 35, 100)
ind <- findInterval(CHNS11$BMI, bk, all.inside = TRUE)
CHNS11$class_Y <- factor(
  c("sous-poids", "normal", "surpoids", "obesite", "obesite_plus")[ind],
  levels = c("sous-poids", "normal", "surpoids", "obesite", "obesite_plus"))
ggtern(data = CHNS11, mapping = aes(x = proteines, y = lipides, z = glucides)) +
  geom_point(mapping = aes(colour = class_Y)) +
  facet_wrap(~ class_Y)
```

Nous allons à présent appliquer un modèle de régression en expliquant la variable Y par les coordonnées ILR de la variable X :

```
ilr_nutrition <- ilr(X_compo)
ilr1 <- ilr_nutrition[, 1]
```

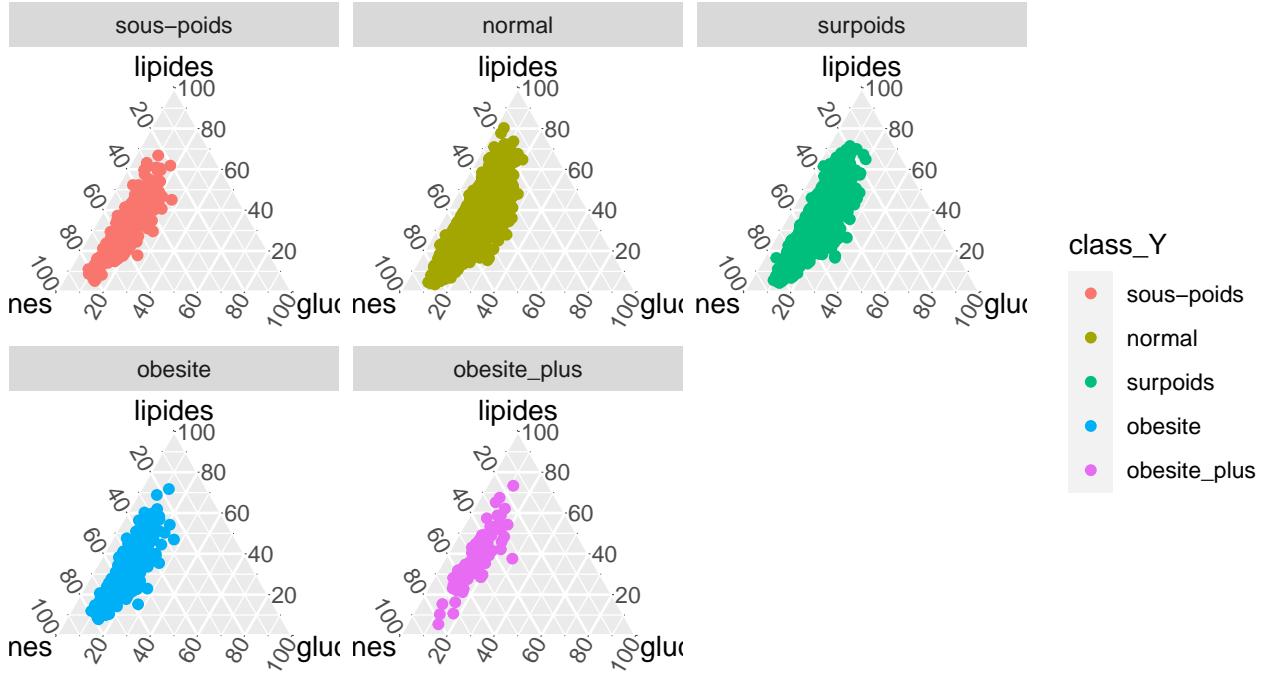


Figure 45: Diagramme ternaire de la variable X conditionnellement à la variable dépendante découpée en classe

```

ilr2 <- ilr_nutrition[, 2]
lm_1 <- lm(BMI ~ ilr1 + ilr2, data = CHNS11)
summary(lm_1)

##
## Call:
## lm(formula = BMI ~ ilr1 + ilr2, data = CHNS11)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -11.343  -2.669  -0.427   2.072  40.407 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 24.40844   0.19643 124.260 < 2e-16 ***
## ilr1        -0.55278   0.15303 -3.612 0.000306 ***  
## ilr2        -0.05791   0.16776 -0.345 0.729978    
## ---        
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 4.046 on 7710 degrees of freedom
## Multiple R-squared:  0.002259, Adjusted R-squared:  0.002 
## F-statistic: 8.729 on 2 and 7710 DF, p-value: 0.0001635

```

L'analyse des résidus (non présenté ici, mais qui s'obtient avec `plot(lm_1)`) montrerait que l'espérance des erreurs est nulle et la variance constante, mais l'hypothèse de normalité est clairement rejetée à cause de certains résidus ayant une valeur trop élevée. Le R^2 est très faible ce qui est confirmé que notre modèle n'est pas bon; cependant, nous avons choisi cet exemple à titre illustratif. Pour transformer les coefficients dans l'espace CLR, on utilise la fonction `ilr2clr()`, qui montre que l'opposition entre les protéines et les lipides

est importante dans l'explication du BMI :

```
ilr2clr(coef(lm_1)[-1])

##          1          2          3
##  0.41451577 -0.36723553 -0.04728024
## attr(),"class"
## [1] "rmult"
```

Enfin, on s'intéresse à la représentation des prédictions, d'abord dans l'espace ILR, puis ensuite dans le simplexe. Ainsi, on observe que plus la composante protéines est forte au détriment des lipides, plus la valeur prédite \hat{Y} sera forte.

```
pal1 <- RColorBrewer::brewer.pal(9, "YlGn")
seq_ilr <- expand.grid(ilr1 = seq(-3, 3, length.out = 100),
                       ilr2 = seq(-3, 3, length.out = 100))
Y_pred <- predict(lm_1, newdata =
  data.frame(ilr1 = seq_ilr[, 1], ilr2 = seq_ilr[, 2]))
bk <- seq(min(Y_pred), max(Y_pred), length.out = 9)
ind <- findInterval(Y_pred, bk, all.inside = TRUE)
op <- par(mfrow = c(1, 2))
# ilr
plot(seq_ilr[, 1], seq_ilr[, 2], col = pal1[ind],
     pch = 16, cex = 0.5, xlab = "ilr 1", ylab = "ilr 2")
# simplex
seq_simp <- ilrInv(seq_ilr)
names(seq_simp) <- c("proteines", "lipides", "glucides")
plot(seq_simp, col = pal1[ind], pch = 16)

decoup <- c("<=-23.03", "]23.03;23.49]", "]23.49;23.95]",
            "]23.95;24.40]", "]24.40;24.86]", "]24.86;25.32]",
            "]25.32;25.78]", ">25.78")
legend("topleft", legend = decoup, cex = 0.6, title = "Y pred",
       fill = pal1)
```

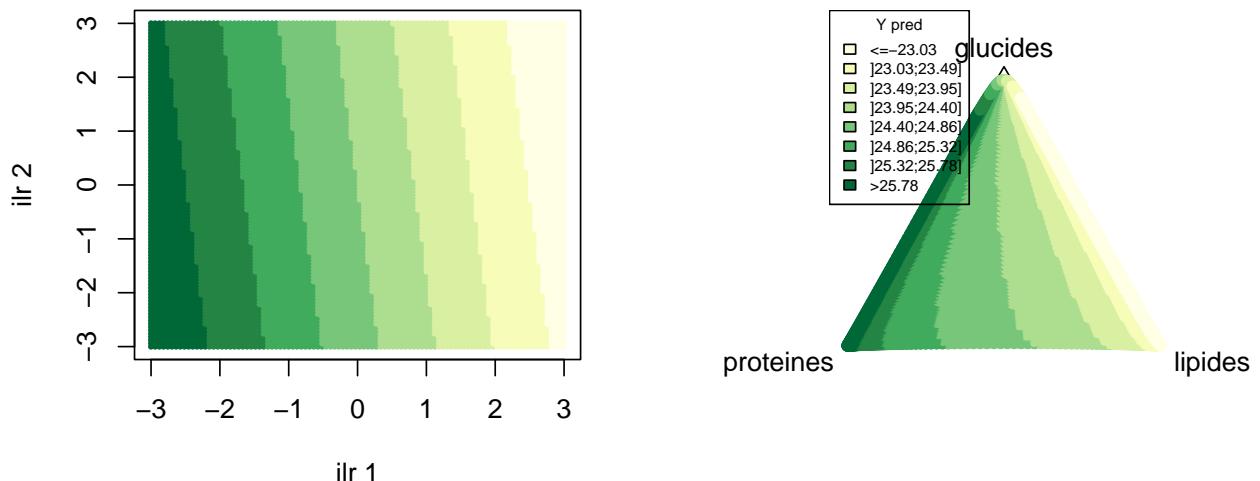


Figure 46: Représentation des prédictions où X est d'abord représenté dans l'espace ILR, puis dans l'espace du simplexe

6.2 Régression avec variable dépendante compositionnelle

Nous allons reprendre ici l'exemple du chapitre 8. Nous allons expliquer les parts de marché automobile par les variables non compositionnelle PIB et diesel. Pour cela, nous allons mettre dans la formule de la fonction `lm()` la variable compositionnelle Y dans l'espace ilr. En effet, il est possible de mettre une variable multidimensionnelle comme variable dépendante et dans ce cas l'objet retourné est de type `m1m`.

```
lm_2 <- lm(ilr(Y_s) ~ PIB_Courant_t + TTC_Gazole, data = BDDSegX)
```

On peut appliquer un certain nombre de fonctions sur un objet de type `m1m` : par exemple, la fonction `coef()` retourne les estimations des paramètres obtenus. On va récupérer les coefficients dans l'espace CLR et les mettre dans un format exploitable pour ensuite représenter un diagramme en barre des coefficients CLR :

```
my_coeff <- data.frame(
  segments = c("A", "B", "C", "D", "E"),
  as(t(ilr2clr(coef(lm_2)[-1, ])), "matrix")
)
```

Il apparaît qu'une augmentation du PIB aura tendance à favoriser la part de marché des constructeurs C, D et E au détriment du constructeur A essentiellement. Au contraire, une augmentation du prix du diesel a un impact positif sur les ventes du constructeur A et ce au détriment de tous les autres constructeurs.

```
coeff_df <- pivot_longer(my_coeff, cols = 2:3, names_to = "variable",
                         values_to = "estimate")
ggplot(coeff_df, aes(x = segments, y = estimate)) +
  geom_bar(stat = "identity") +
  facet_wrap(~variable, ncol = 5, scales = "free_y")
```

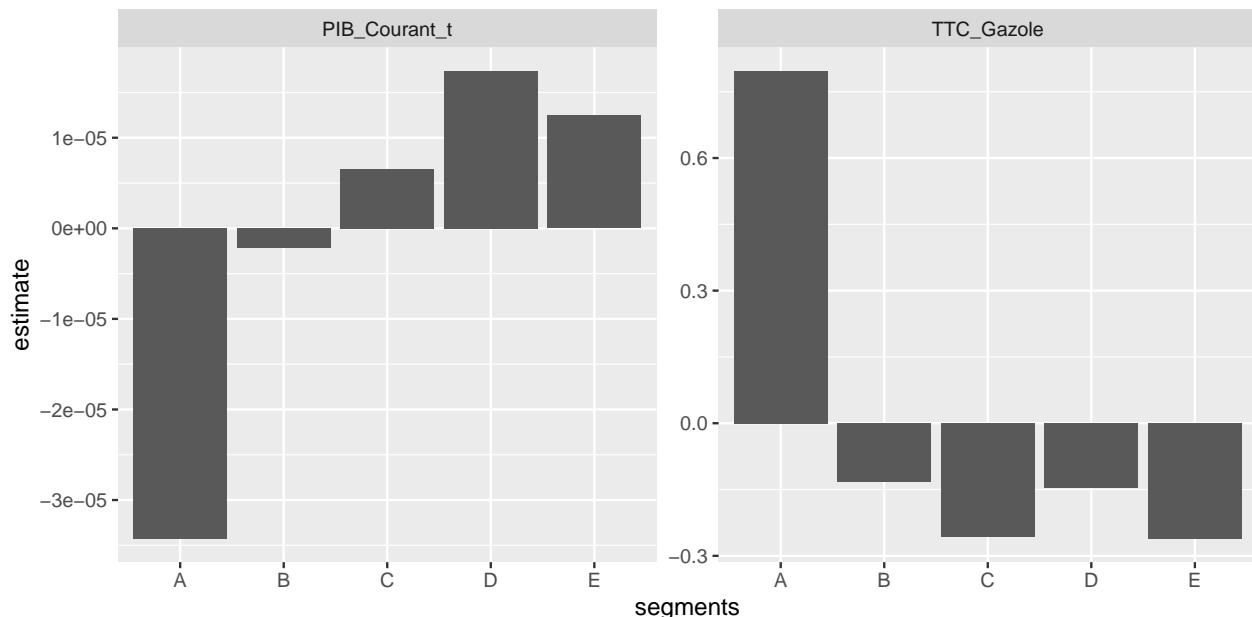


Figure 47: Représentation des coefficients dans l'espace CLR

La fonction `anova()` permet de faire des tests en tenant compte du caractère multivariée. Cela nous permet de vérifier que les deux variables sont globalement significativement différentes de 0 dans l'espace ilr.

```
anova(lm_2)
```

```
## Analysis of Variance Table
##
```

```

##          Df Pillai approx F num Df den Df      Pr(>F)
## (Intercept)   1 0.99493    7164.2      4    146 < 2.2e-16 ***
## PIB_Courant_t   1 0.81799     164.0      4    146 < 2.2e-16 ***
## TTC_Gazole     1 0.12439      5.2      4    146 0.0006197 ***
## Residuals     149
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

En revanche, il n'est pas possible d'appliquer la fonction `plot()` pour faire un diagnostique des résidus.

Enfin, on va représenter le graphique des prédictions des parts de marché en fonction de variables explicatives prises l'une après l'autre. Quand on fait varier l'une des variables explicatives entre les valeurs minimum et maximum observées, on fixe l'autre variable explicative à sa valeur médiane.

```

# prediction quand PIB_Courant_t varie
seq_PIB_Courant_t <- seq(min(BDDSegX$PIB_Courant_t), max(BDDSegX$PIB_Courant_t),
                           length.out = 100)
pred_Y <- as(ilrInv(predict(lm_2, newdata = data.frame(
  PIB_Courant_t = seq_PIB_Courant_t,
  TTC_Gazole = median(BDDSegX$TTC_Gazole)
))), "matrix")
colnames(pred_Y) <- c("A", "B", "C", "D", "E")
data_pred_1 <- data.frame(pred_Y, x = seq_PIB_Courant_t)
# prediction quand gazole varie
seq_TTC_Gazole <- seq(min(BDDSegX$TTC_Gazole), max(BDDSegX$TTC_Gazole),
                       length.out = 100)
pred_Y <- as(ilrInv(predict(lm_2, newdata = data.frame(
  PIB_Courant_t = median(BDDSegX$PIB_Courant_t),
  TTC_Gazole = seq_TTC_Gazole
))), "matrix")
colnames(pred_Y) <- c("A", "B", "C", "D", "E")
data_pred_2 <- data.frame(pred_Y, x = seq_TTC_Gazole)
# Merge des deux predictions
pred_lg <- rbind(
  data.frame(variable = "PIB",
             pivot_longer(data_pred_1, cols = 1:5, names_to = "segments",
                           values_to = "estimate")),
  data.frame(variable = "Gazole",
             pivot_longer(data_pred_2, cols = 1:5, names_to = "segments",
                           values_to = "estimate"))
)
ggplot(pred_lg, aes(x = x, y = estimate, colour = segments)) +
  geom_line() +
  facet_wrap(~variable, ncol = 5, scales = "free")

```

On citera le package `codareg` (en cours de développement) qui permet de calculer les elasticités pour chaque individu et de faire des résumés avec ces elasticités.

Archimbaud, Aurore, Klaus Nordhausen, and Anne Ruiz-Gazen. 2018. *ICSOOutlier: Outlier Detection Using Invariant Coordinate Selection*. <https://CRAN.R-project.org/package=ICSOOutlier>.

Filzmoser, Peter, Karel Hron, and Matthias Templ. 2018. *Applied Compositional Data Analysis. With Worked Examples in R*. Springer International Publishing. Springer Nature Switzerland AG, Cham, Switzerland.

Giraud, Timothée. 2022. *Mapsf: Thematic Cartography*. <https://CRAN.R-project.org/package=mapsf>.

Greenacre, M. 2018. *Compositional Data Analysis in Practice*. Chapman & Hall / CRC Press.

Hamilton, Nicholas E., and Michael Ferry. 2018. “`ggtern`: Ternary Diagrams Using `ggplot2`.” *Journal of Statistical Software, Code Snippets* 87 (3): 1–17. <https://doi.org/10.18637/jss.v087.c03>.

Palarea-Albaladejo, J, and JA Martin-Fernandez. 2015. “zCompositions – r Package for Multivariate

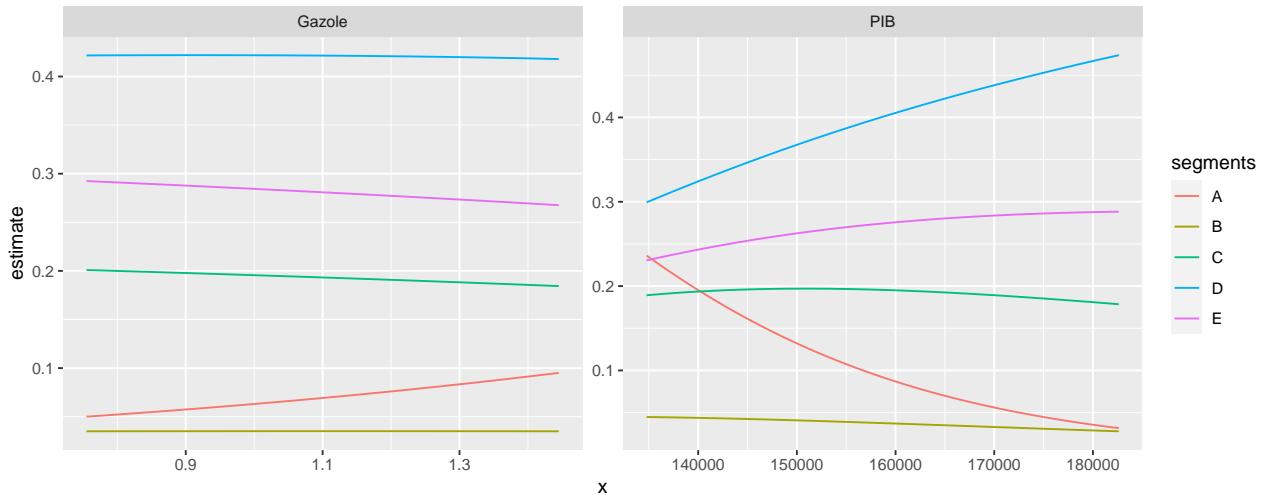


Figure 48: Représentation des prédictions des parts de marché en fonction de variables quantitatives

- Imputation of Left-Censored Data Under a Compositional Approach.” *Chemometrics and Intelligent Laboratory Systems* 143: 85–96. <http://dx.doi.org/10.1016/j.chemolab.2015.02.019>.
- Pebesma, Edzer. 2018. “Simple Features for R: Standardized Support for Spatial Vector Data.” *The R Journal* 10 (1): 439–46. <https://doi.org/10.32614/RJ-2018-009>.
- R Core Team. 2022. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Székely, Gábor J., and Maria L. Rizzo. 2005. “Hierarchical Clustering via Joint Between-Within Distances: Extending Ward’s Minimum Variance Method.” *J. Classification* 22 (2): 151–83. <http://dblp.uni-trier.de/db/journals/classification/classification22.html#SzekelyR05>.
- van den Boogaart, K. Gerald, Raimon Tolosana-Delgado, and Matevz Bren. 2022. *Compositions: Compositional Data Analysis*. <https://CRAN.R-project.org/package=compositions>.
- Von Eynatten, Hilmar, Vera Pawlowsky-Glahn, and Juan José Egozcue. 2002. “Understanding Perturbation on the Simplex: A Simple Method to Better Visualize and Interpret Compositional Data in Ternary Diagrams.” *Mathematical Geology* 34 (3): 249–57.
- Zeileis, Achim, and Gabor Grothendieck. 2005. “Zoo: S3 Infrastructure for Regular and Irregular Time Series.” *Journal of Statistical Software* 14 (6): 1–27. <https://doi.org/10.18637/jss.v014.i06>.