Simulations

Jawad Boulahfa

06/05/2020

Première partie: Simulations sur le modèle nnls

```
rm(list = ls())
```

Installation du package test

```
#devtools::install_github("Jawad-Boulahfa/test")
```

Chargement des packages

```
# Pour pcls2 et les fonctions de simulation
library(test, quietly = TRUE)

# Pour le calcul parallèle
library(foreach, quietly = TRUE)
library(iterators, quietly = TRUE)
library(parallel, quietly = TRUE)
library(doParallel, quietly = TRUE)

# Pour ggplot2 et la manipulation des dataframes
library(tidyverse, quietly = TRUE)

# Pour tracer plusieurs graphiques en même temps
library(gridExtra, quietly = TRUE)

# Pour construire des heatmap
library(reshape2, quietly = TRUE)

# Pour construire des heatmap
library(hrbrthemes)
library(plotly)
library(webshot)
```

Initialisation

Listes pour les valeurs de n et de σ

```
liste_n <- seq(from = 1000, to = 11000, by = 1000)
liste_sigma <- seq(from = 0, to = 100, by = 10)
liste_sigma[1] <- 1
```

Pour alléger l'écriture, on stocke la taille de ces listes dans une variable.

```
taille <- length(liste_n)*length(liste_sigma)
```

Choix de la valeur de beta.

```
beta = c(0, 1)
```

Initialisation du calcul parallèle

```
nb_one_simu <- 200
cores <- detectCores()
```

Choix du nombre de classes pour les histogrammes

```
classes_hist <- 50
```

Choix du α pour les IC

```
alpha <- 0.05
```

Remarque préliminaire: La fonction one.simu simule un jeu de données puis calcule $\hat{\beta}_{nnls}$ et $\hat{\beta}_{lm}$. Elle affiche les résultats ainsi que le récapitulatif des paramètres choisis sous la forme d'un data.frame.

Simulations pour toutes les valeurs de n et de sigma

Pour chaque valeur de n et chaque valeur de σ dans des listes préalablement définies, on effectue une simulation en répétant 200 fois la fonction one.simu.

On construit une liste pour stocker les résultats de chaque simulation et plusieurs listes pour stocker les histogrammes que l'on va tracer.

On construit ensuite, pour chacune des simulations, des listes pour stocker: - le biais, la variance, et l'erreur quadratique moyenne - les intervalles de confiance - les rapports d'amplitude des intervalles de confiance de chaque composante de $\hat{\beta}_{nnls}$ et $\hat{\beta}_{lm}$.

On construit également une liste pour stocker les résultats théoriques attendus pour le biais, la variance et l'erreur quadratique moyenne de chaque composante de $\hat{\beta}_{nnls}$.

D'autres listes sont également construites pour pouvoir tracer les histogrammes. Ces listes sont également retournées par la fonction.

Enfin, on stocke les légendes des dataframes construits dans des listes.

Il est important de noter que pour chaque valeur de n, on fixe la matrice de design $\mathcal{X}$.

Le paramètre "cores" est utilisé pour le calcul parallèle.

On doit ajouter le paramètre FUN pour éviter une erreur (la fonction one.simu n'est pas trouvée sinon).

```
simulations <- test::simulations(cores = cores,
                          liste_n = liste_n,
                          liste_sigma = liste_sigma,
                          beta = beta, nb_one_simu = nb_one_simu,
                          classes_hist = classes_hist, alpha = alpha)
```

```
noquote(names(simulations)) # retire les guillemets
```

```
##  [1] liste_final_df
##  [2] liste_plot_nnls_1
##  [3] liste_plot_nnls_2
##  [4] liste_plot_lm_1
##  [5] liste_plot_lm_2
##  [6] liste_comparison_1
##  [7] liste_comparison_2
##  [8] liste_plot_comparison_1
##  [9] liste_plot_comparison_2
## [10] liste_comparison_1_without_0
## [11] liste_comparison_2_without_0
## [12] liste_plot_comparison_1_without_0
## [13] liste_plot_comparison_2_without_0
## [14] liste_resultats_df
## [15] liste_comments_resultats_df
## [16] liste_resultats_theoriques_df
## [17] liste_comments_resultats_theoriques_df
## [18] liste_IC_df
## [19] liste_comments_IC_df
## [20] liste_rapports_1
## [21] liste_comments_rapports_1
## [22] liste_rapports_2
## [23] liste_comments_rapports_2
```

On change les noms pour plus de lisibilité.

```
liste_final_df <- simulations$liste_final_df

liste_plot_nnls_1 <- simulations$liste_plot_nnls_1
liste_plot_nnls_2 <- simulations$liste_plot_nnls_2

liste_plot_lm_1 <- simulations$liste_plot_lm_1
liste_plot_lm_2 <- simulations$liste_plot_lm_2

liste_comparison_1 <- simulations$liste_comparison_1
liste_comparison_2 <- simulations$liste_comparison_2

liste_comparison_1_without_0 <-
  simulations$liste_comparison_1_without_0
liste_comparison_2_without_0 <-
  simulations$liste_comparison_2_without_0

liste_plot_comparison_1 <- simulations$liste_plot_comparison_1
liste_plot_comparison_2 <- simulations$liste_plot_comparison_2

liste_plot_comparison_1_without_0 =
  simulations$liste_plot_comparison_1_without_0
liste_plot_comparison_2_without_0 =
  simulations$liste_plot_comparison_2_without_0

liste_resultats_df <- simulations$liste_resultats_df
liste_comments_resultats_df <- simulations$liste_comments_resultats_df

liste_resultats_theoriques_df <-
  simulations$liste_resultats_theoriques_df
```

```
liste_comments_resultats_theoriques_df <-
  simulations$liste_comments_resultats_theoriques_df

liste_IC_df <- simulations$liste_IC_df
liste_comments_IC_df <- simulations$liste_comments_IC_df

liste_rapports_1 <- simulations$liste_rapports_1
liste_rapports_2 <- simulations$liste_rapports_2

liste_comments_rapports_1 <- simulations$liste_comments_rapports_1
liste_comments_rapports_2 <- simulations$liste_comments_rapports_2
```

Vérification valeurs négatives

On vérifie que $\hat{\beta}_{nnls_1} \geq 0$ et $\hat{\beta}_{nnls_2} \geq 0$ pour toutes les simulations.

```
verif_1 <- vector("list", taille)
verif_2 <- vector("list", taille)

for(k in 1:100){verif_1[[k]] <-
  sum(liste_final_df[[k]][,5] < 0)}
for(k in 1:100){verif_2[[k]] <-
  sum(liste_final_df[[k]][,6] < 0)}
```

```
print(sum(unlist(verif_1)))
```

```
## [1] 0
```

```
# on obtient 0, donc il n'y a aucun \hat\beta_{nnls_1} < 0
print(sum(unlist(verif_2)))
```

```
## [1] 0
```

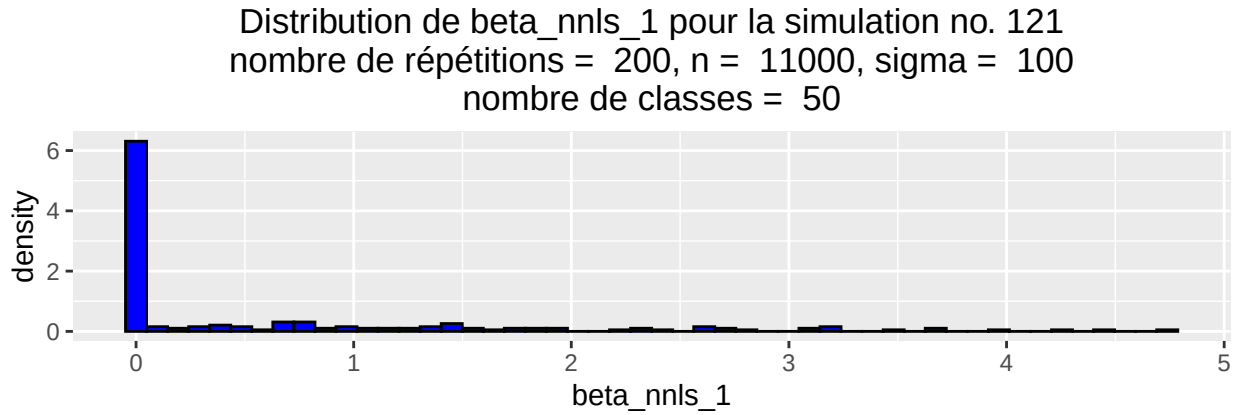
```
# on obtient 0, donc il n'y a aucun \hat\beta_{nnls_2} < 0
```

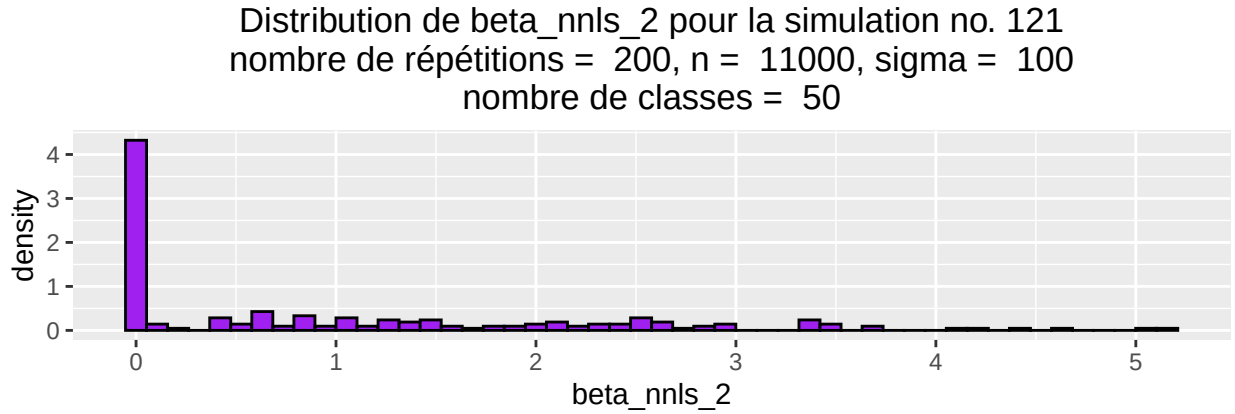
Comme on obtient 0 dans les deux cas, on en déduit qu'il n'y a aucun $\hat{\beta}_{nnls_1} < 0$ et aucun $\hat{\beta}_{nnls_2} < 0$ dans toutes nos simulations. C'est bien ce que l'on souhaitait.

Quelques histogrammes

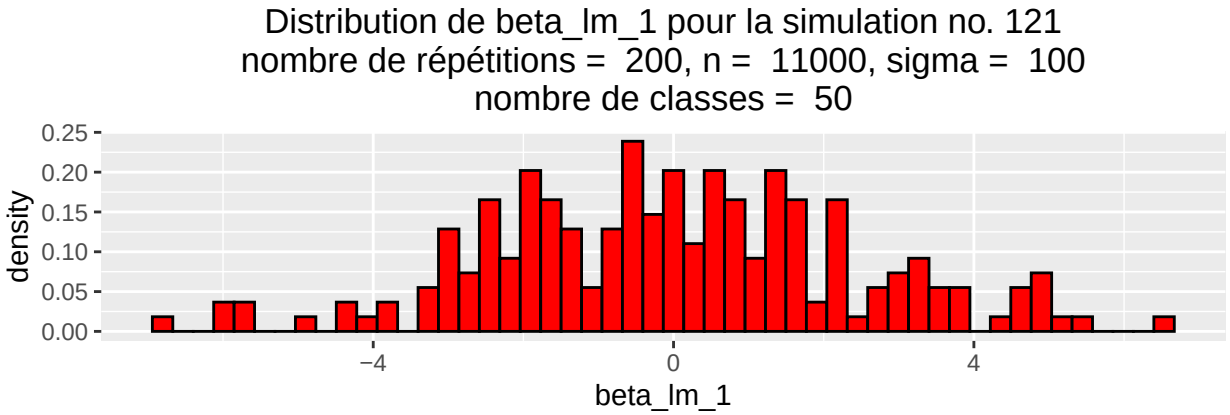
On a déjà construit les histogrammes, donc on peut les afficher directement.

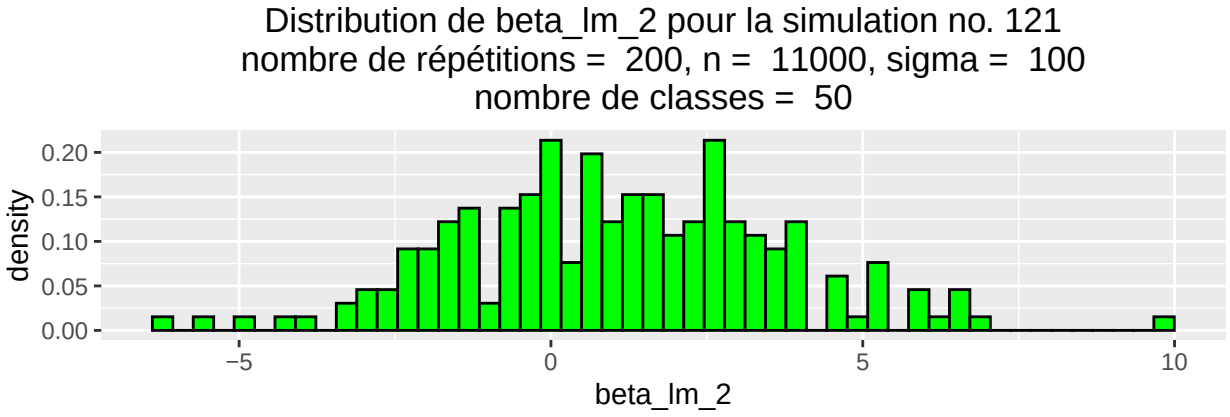
```
# Dernière simulation (nnls)
grid.arrange(liste_plot_nnls_1[[taille]],
             liste_plot_nnls_2[[taille]],
             nrow=2, ncol=1)
```



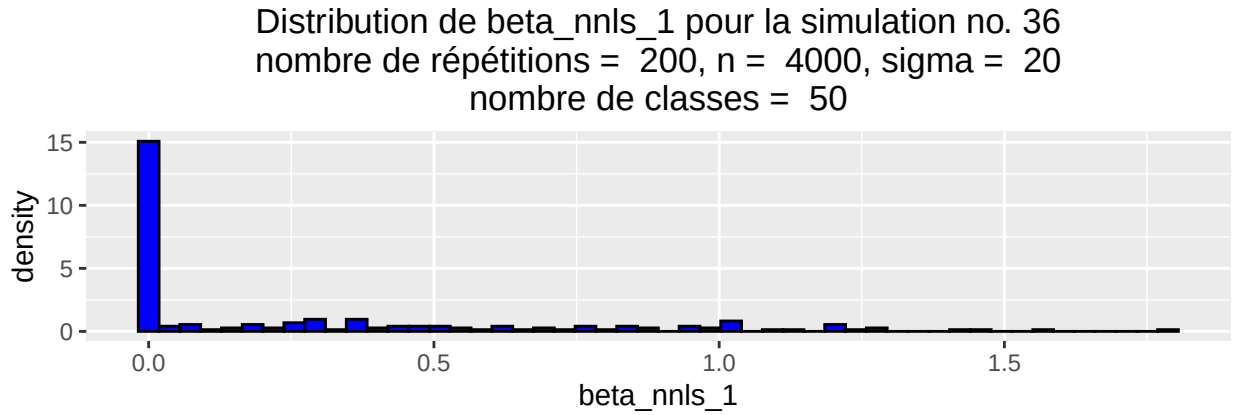


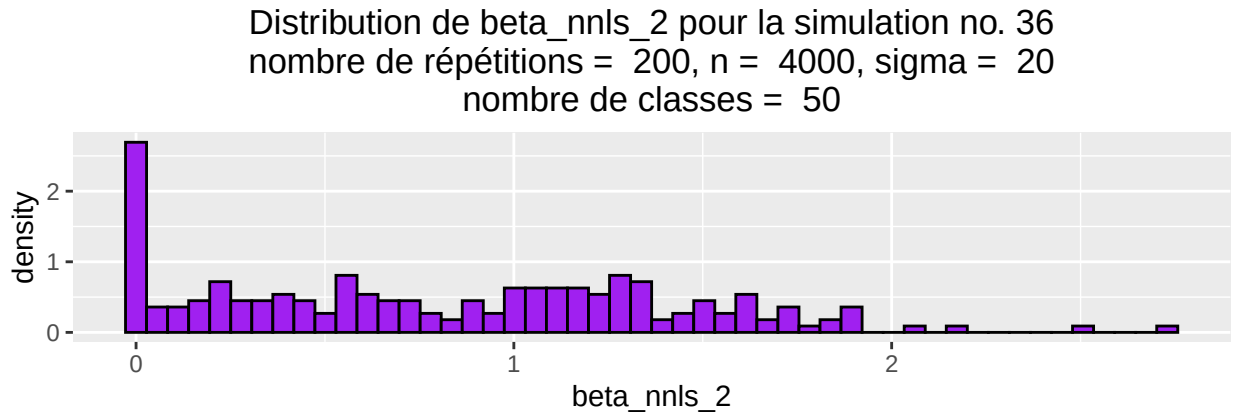

```
# Dernière simulation (lm)
grid.arrange(liste_plot_lm_1[[taille]],
             liste_plot_lm_2[[taille]],
             nrow=2, ncol=1)
```



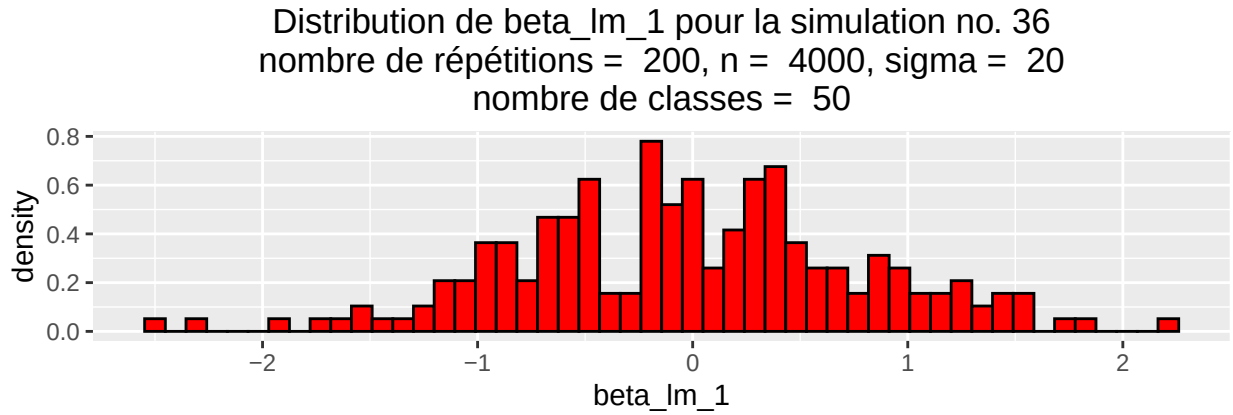


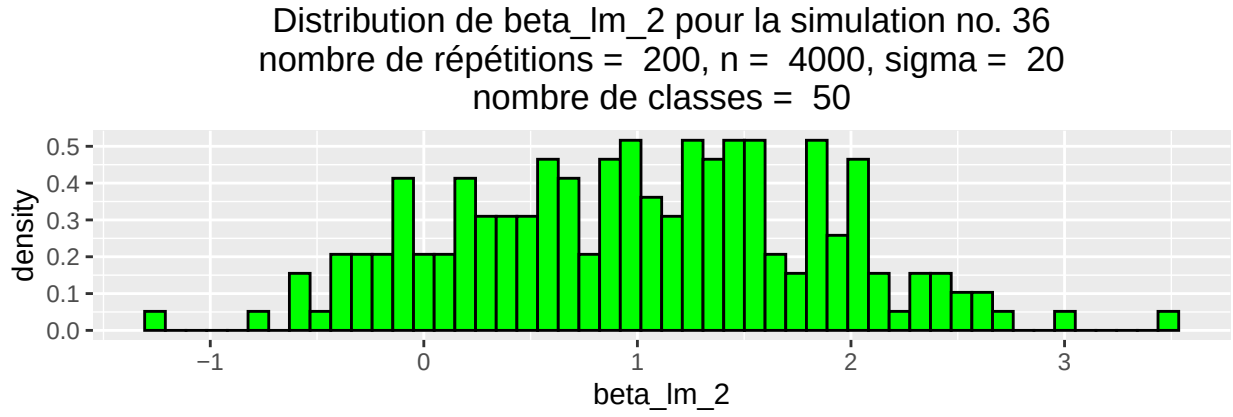

```
# 36ème simulation (nnls)
grid.arrange(liste_plot_nnls_1[[36]],
             liste_plot_nnls_2[[36]],
             nrow=2, ncol=1)
```



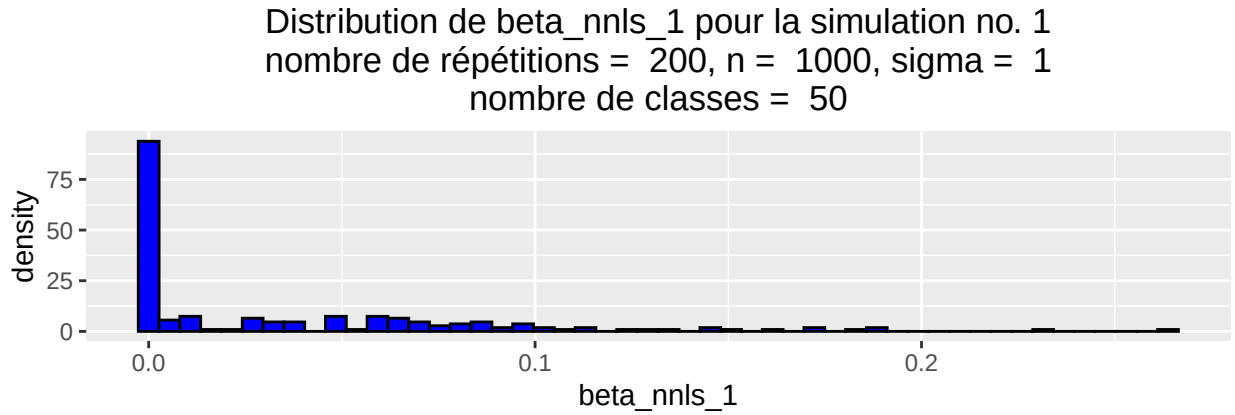


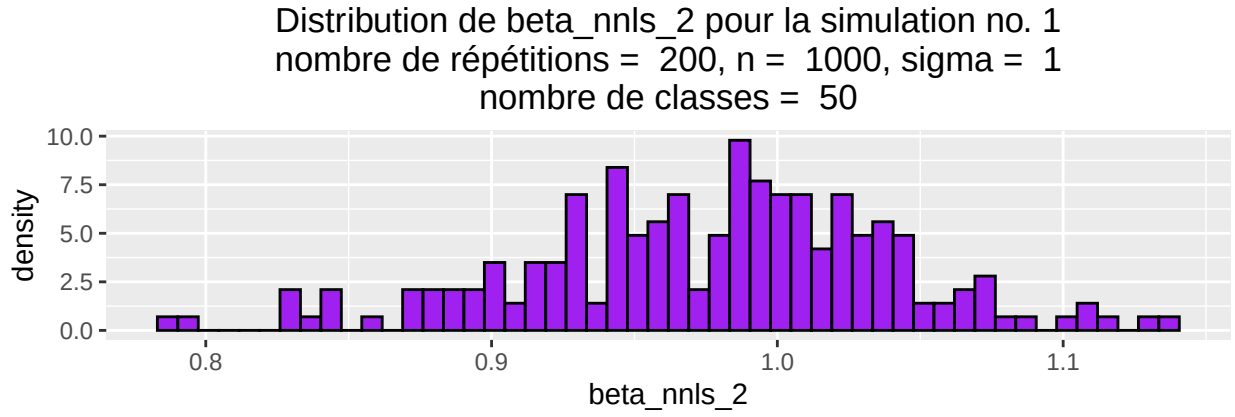

```
# 36ème simulation (lm)
grid.arrange(liste_plot_lm_1[[36]],
             liste_plot_lm_2[[36]],
             nrow=2, ncol=1)
```



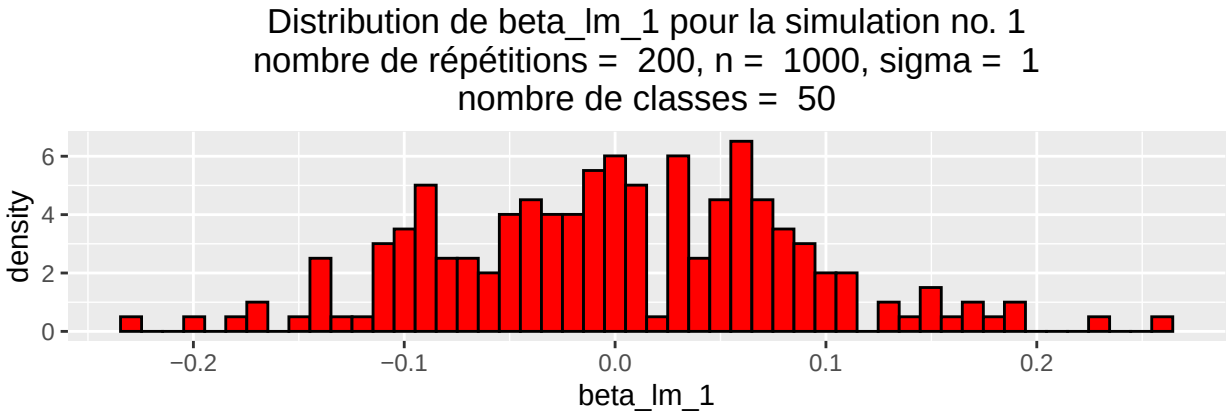


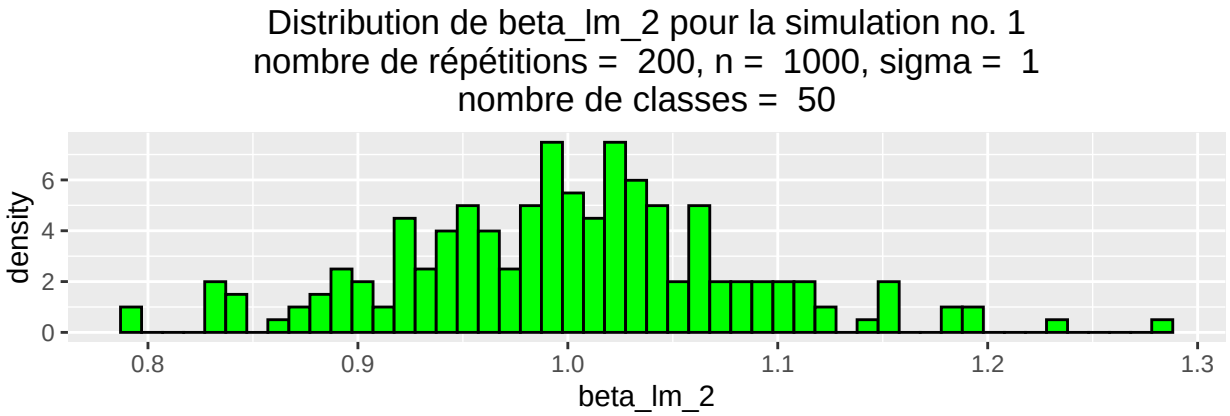

```
# Première simulation (nnls)
grid.arrange(liste_plot_nnls_1[[1]],
             liste_plot_nnls_2[[1]],
             nrow=2, ncol=1)
```





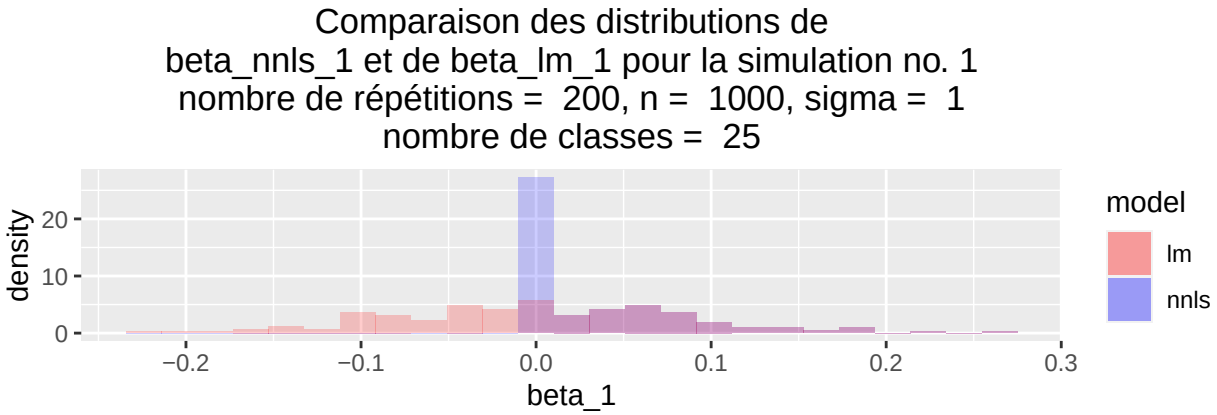

```
# Première simulation (lm)
grid.arrange(liste_plot_lm_1[[1]],
             liste_plot_lm_2[[1]],
             nrow=2, ncol=1)
```

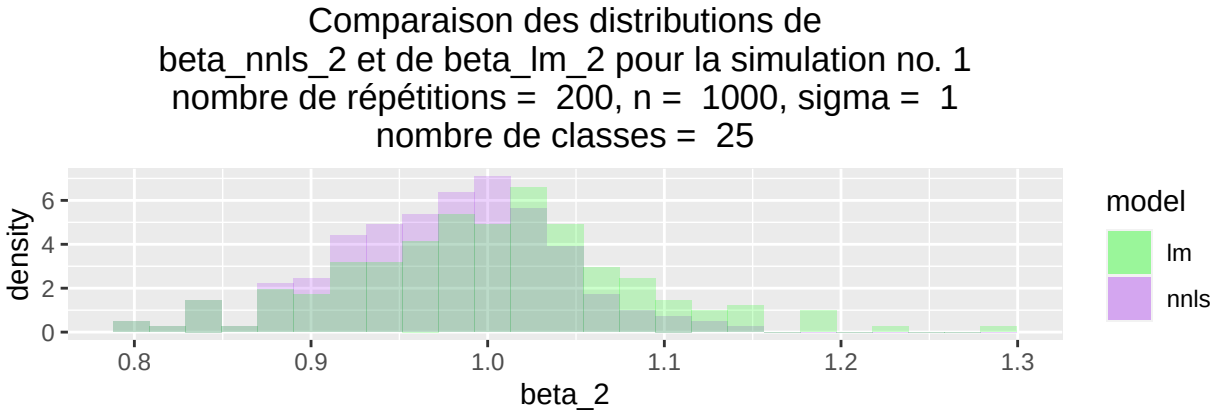




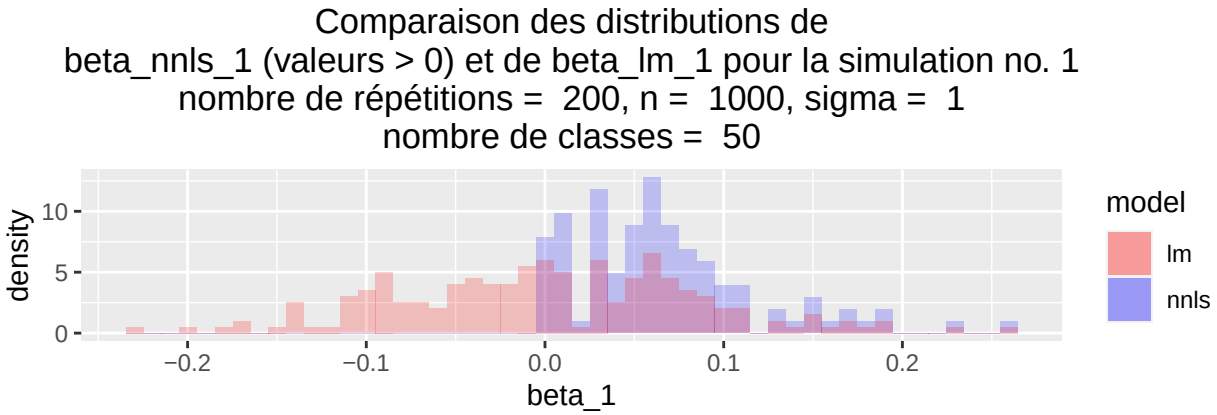
Histogrammes comparatifs pour $n = 1000, \sigma = 1$

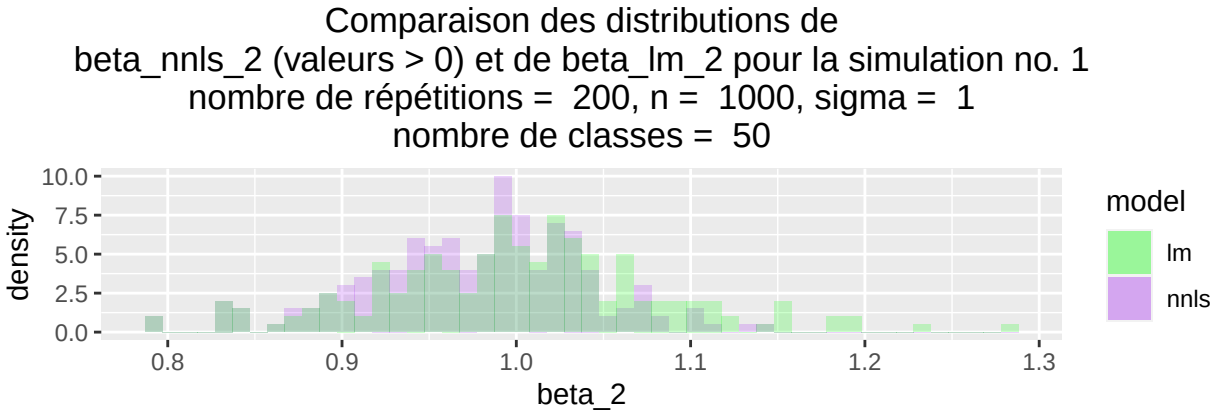
```
# Première simulation
grid.arrange(liste_plot_comparison_1[[1]],
             liste_plot_comparison_2[[1]],
             nrow=2, ncol=1)
```





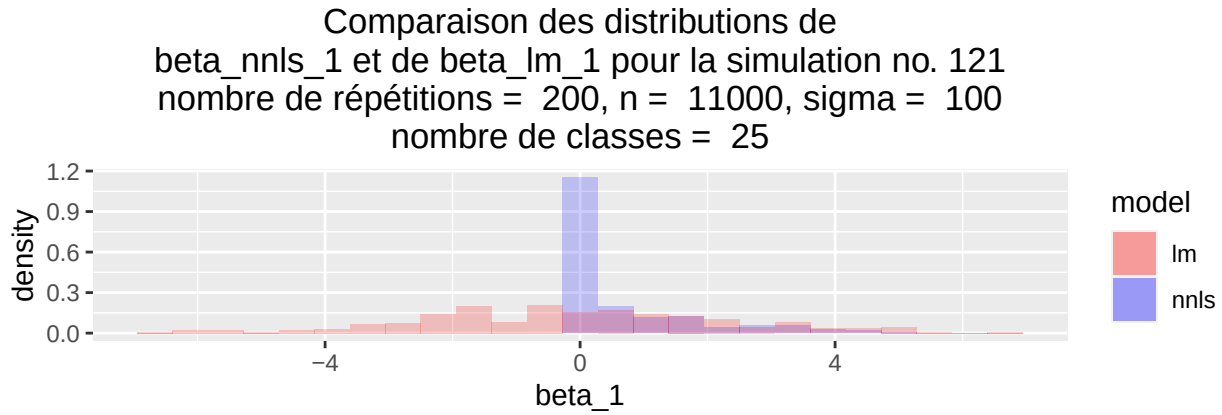

```
# Première simulation
grid.arrange(liste_plot_comparison_1_without_0[[1]],
             liste_plot_comparison_2_without_0[[1]],
             nrow=2, ncol=1)
```

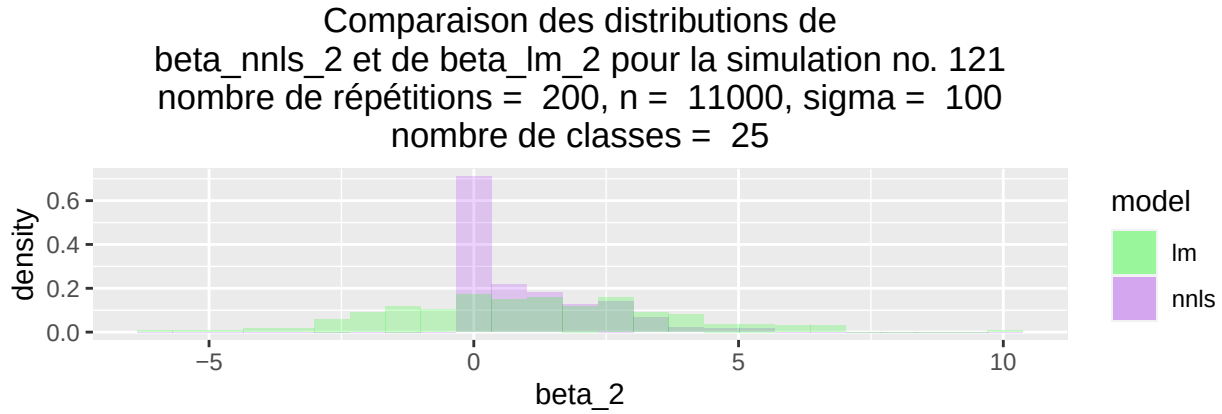




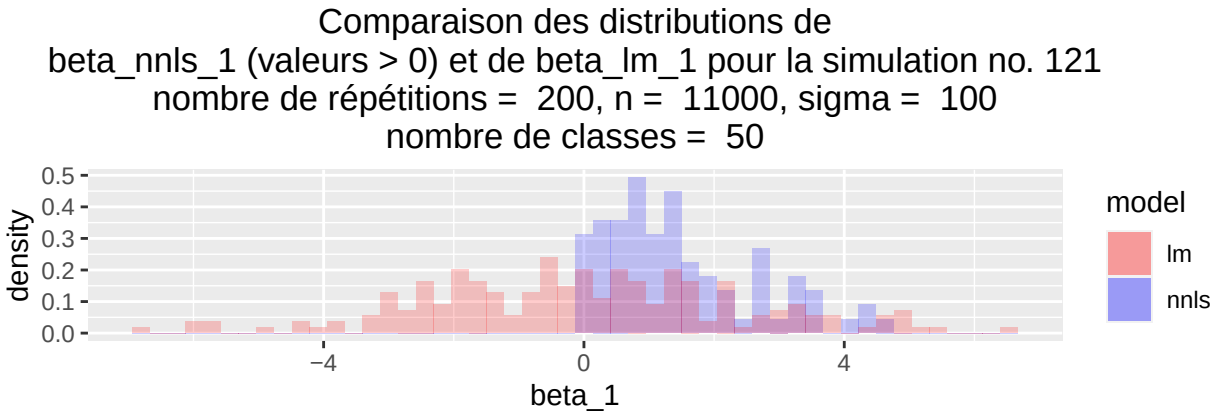
Histogrammes comparatifs pour $n = 1.1 \times 10^4, \sigma = liste_sigma[length(liste_n)]$

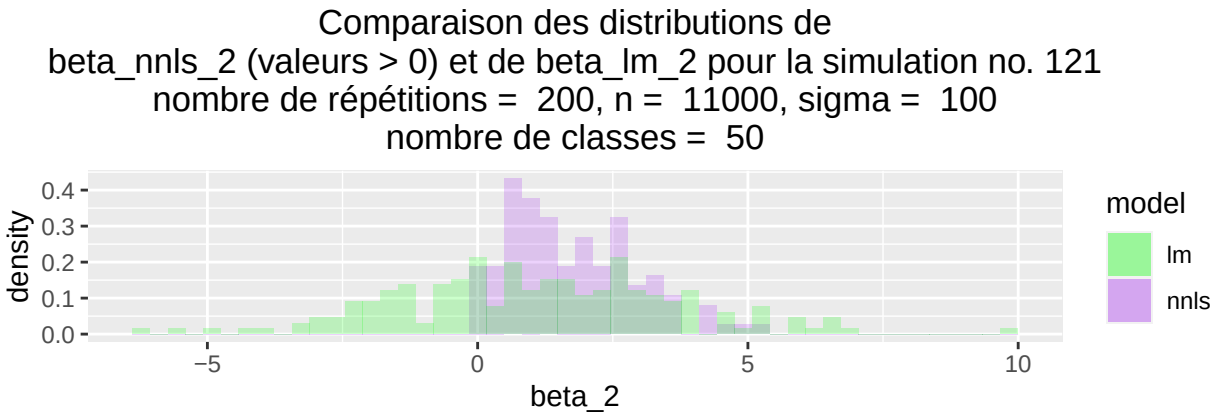
```
# Dernière simulation
grid.arrange(liste_plot_comparison_1[[taille]],
             liste_plot_comparison_2[[taille]],
             nrow=2, ncol=1)
```






```
# Dernière simulation
grid.arrange(liste_plot_comparison_1_without_0[[taille]],
             liste_plot_comparison_2_without_0[[taille]],
             nrow=2, ncol=1)
```





Tests de normalité pour la dernière simulation ($nombre\ de\ \ répétitions = 200,\ n = 1.1 \times 10^4,\ \sigma = 100$)

```
shapiro_1 <-
  shapiro.test(liste_final_df[[taille]]$beta_nnls_1)
pvalue_1 <- shapiro_1$p.value
```

```
print(shapiro_1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  liste_final_df[[taille]]$beta_nnls_1
## W = 0.65281, p-value < 2.2e-16
```

Le test de Shapiro-Wilk rejette l'hypothèse de normalité de $\hat{\beta}_{nnls_2}$ ($p - value \simeq 5.3220919 \times 10^{-20}$). On conclut donc que $\hat{\beta}_{nnls_1}$ n'est pas gaussien.

```
shapiro_2 <-
  shapiro.test(liste_final_df[[taille]]$beta_nnls_2)
pvalue_2 <- shapiro_2$p.value
```

```
print(shapiro_2)
```

```
##
##  Shapiro-Wilk normality test
##
```

```
## data:  liste_final_df[[taille]]$beta_nnls_2
## W = 0.79661, p-value = 2.051e-15
```

Le test de Shapiro-Wilk rejette l'hypothèse de normalité de $\hat{\beta}_{nnls_2}$ ($p - value \simeq 2.0508619 \times 10^{-15}$). On conclut donc que $\hat{\beta}_{nnls_2}$ n'est pas gaussien.

Test de normalité pour chacune des simulations

Pour chacune des simulations, on va effectuer un test de Shapiro-Wilk pour $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{nnls_2}$. On compte ensuite le nombre de fois où l'hypothèse de normalité n'est pas rejetée. Enfin, on calcule la fréquence de rejet de cette hypothèse en %.

```
tests_beta_nnls_1_list <- vector("list", taille)
tests_beta_nnls_2_list <- vector("list", taille)
niveau <- 0.05
non_rejet_1 <- 0
non_rejet_2 <- 0

for(n in 1:length(liste_n))
{
  for(sigma in 1:length(liste_sigma))
  {
    # Indice pour insérer dans les listes
    indice <- (n - 1) * length(liste_sigma) + sigma

    tests_beta_nnls_1_list[[indice]] <-
      shapiro.test(liste_final_df[[indice]]$beta_nnls_1)
    tests_beta_nnls_2_list[[indice]] <-
      shapiro.test(liste_final_df[[indice]]$beta_nnls_2)

    if(tests_beta_nnls_1_list[[indice]]$p.value > niveau)
    {
      non_rejet_1 = non_rejet_1 + 1
    }

    if(tests_beta_nnls_2_list[[indice]]$p.value > niveau)
    {
      non_rejet_2 = non_rejet_2 + 1
    }

  }
}

rejet_1 <- 100*(1-non_rejet_1/taille)
rejet_2 <- 100*(1-non_rejet_2/taille)
```

On affiche les résultats obtenus pour $\hat{\beta}_{nnls_1}$.

```
cat(paste0(
  "Nombre de fois où on ne rejette pas l'hypothèse de normalité pour beta_nnls_1 = ",
  non_rejet_1, "\n"), sep = "\n")
```

```
## Nombre de fois où on ne rejette pas l'hypothèse de normalité pour beta_nnls_1 = 0
```

```
cat(paste0(
  "Pourcentage de rejet de l'hypothèse de normalité pour beta_nnls_1 = ",
```

```
  rejet_1, "%"))
```

```
## Pourcentage de rejet de l'hypothèse de normalité pour beta_nnls_1 = 100%
```

On affiche les résultats obtenus pour $\hat{\beta}_{nnls_2}$.

```
cat(paste0(
  "Nombre de fois où on ne rejette pas l'hypothèse de normalité pour beta_nnls_2 = ",
  non_rejet_2, "\n"), sep = "\n")
```

```
## Nombre de fois où on ne rejette pas l'hypothèse de normalité pour beta_nnls_2 = 16
```

```
cat(paste0(
  "Pourcentage de rejet de l'hypothèse de normalité pour beta_nnls_2 = ",
  rejet_2, "%"))
```

```
## Pourcentage de rejet de l'hypothèse de normalité pour beta_nnls_2 = 86.7768595041322%
```

Pour $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{nnls_2}$, on a affiché le nombre de fois où on n'a pas rejeté l'hypothèse de normalité ainsi que la proportion de rejet de cette hypothèse en %.

Sur l'ensemble de nos simulations, le test de Shapiro-Wilk rejette l'hypothèse de normalité de $\hat{\beta}_{nnls_1}$ dans 100% des cas. Pour $\hat{\beta}_{nnls_2}$, le test de Shapiro-Wilk rejette l'hypothèse de normalité dans 86.7768595% des cas.

On remarque que ce nombre est plus petit pour $\hat{\beta}_{nnls_2}$. Cela s'explique par le fait que plus β_i s'éloigne de 0, plus $\hat{\beta}_{nnls_i}$ se rapproche de $\hat{\beta}_{lm_i}$ pour tout $i \in \{1, 2\}$, avec $\hat{\beta}_{lm_i}$ gaussien.

On en conclut que les lois des composantes de $\hat{\beta}_{nnls}$ ne sont pas simplement des lois gaussiennes, mais des lois plus difficiles à déterminer qui dépendent de la valeur de β choisie au départ.

Quelques évaluations quantitatives de nos modèles

Biais, variance, erreur quadratique moyenne

```
# Dernière simulation
cat(liste_comments_resultats_df[[taille]], sep = "\n")
```

```
## Simulation no.121
## Nombre de répétitions = 200
## n = 11000
## sigma = 100
```

```
print(liste_resultats_df[[taille]])
```

```
##              biais variance      MSE
## nnls_1  0.60976709 1.103640 1.475456
## lm_1   -0.02012080 5.869768 5.870173
## nnls_2 -0.01016571 1.562746 1.562850
## lm_2    0.07371485 6.454293 6.459727
```

Avec $n = 1.1 \times 10^4$ et $\sigma = 100$, on obtient un biais plus grand en valeur absolue avec le modèle nnls, mais une variance plus petite, par rapport au modèle lm. Au final, l'erreur quadratique moyenne obtenue avec le modèle nnls est toujours plus petite que celle obtenue avec le modèle lm, ce qui donne un argument en faveur du modèle nnls.

```
# Résultats théoriques attendus
cat(liste_comments_resultats_theoriques_df[[taille]], sep = "\n")
```

```
## Résultats théoriques attendus pour la simulation no.121
```

```
## n = 11000
## sigma = 100
```

```
print(liste_resultats_theoriques_df[[taille]])
```

```
##              biais variance      MSE
## nnls_1  0.63352037 1.024628 1.425976
## nnls_2 -0.07577583 1.408312 1.414054
```

On remarque que les valeurs simulées sont ici proches des résultats théoriques pour $\hat{\beta}_{nnls_1}$. Cependant, pour $\hat{\beta}_{nnls_2}$, on obtient des valeurs simulées plus petites que les résultats théoriques.

```
# Première simulation
cat(liste_comments_resultats_df[[1]], sep = "\n")
```

```
## Simulation no.1
## Nombre de répétitions = 200
## n = 1000
## sigma = 1
```

```
print(liste_resultats_df[[1]])
```

```
##               biais    variance         MSE
## nnls_1  0.034493572 0.002542528 0.003732335
## lm_1    0.001648598 0.007115394 0.007118112
## nnls_2 -0.023689913 0.004041428 0.004602640
## lm_2    0.000802492 0.006614335 0.006614979
```

Avec $n = 1000$ et $\sigma = 1$ le constat est le même.

```
# Résultats théoriques attendus
cat(liste_comments_resultats_theoriques_df[[1]], sep = "\n")
```

```
## Résultats théoriques attendus pour la simulation no.1
## n = 1000
## sigma = 1
```

```
print(liste_resultats_theoriques_df[[1]])
```

```
##              biais    variance         MSE
## nnls_1  0.03353638 0.002408626 0.003533315
## nnls_2 -0.02500799 0.004281603 0.004907002
```

On remarque que les valeurs simulées sont ici proches des résultats théoriques sauf pour le biais de $\hat{\beta}_{nnls_2}$.

Intervalles de confiance au niveau de confiance 0.95

```
# Dernière simulation
cat(liste_comments_IC_df[[taille]])
```

```
## Simulation no.121
## Nombre de répétitions = 200
## n = 11000
## sigma = 100
```

```
print(liste_IC_df[[taille]])
```

```
##           IC_nnls_1   IC_lm_1 IC_nnls_2   IC_lm_2
## Borne inf  0.000000 -5.042688  0.000000 -3.319267
## Borne sup  3.627579  4.871918  4.129257  6.129188
```

Avec$n = 1.1 \times 10^4$ et $\sigma = 100$, les amplitudes des intervalles de confiance (au niveau de confiance 95%) obtenues avec le modèle nnls sont toujours plus petites que celles obtenues avec le modèle lm. On remarque que pour la deuxième composante, l'intervalle de confiance obtenu avec le modèle nnls est très proche de celui obtenu avec le modèle lm. Cela s'explique par le fait que plus β_i s'éloigne de 0, plus $\hat{\beta}_{nnls_i}$ se rapproche de $\hat{\beta}_{lm_i}$ pour tout $i \in \{1, 2\}$.

```
cat(liste_comments_IC_df[[30]])
```

```
## Simulation no.30
## Nombre de répétitions = 200
## n = 3000
## sigma = 70
```

```
print(liste_IC_df[[30]])
```

```
##           IC_nnls_1   IC_lm_1 IC_nnls_2   IC_lm_2
## Borne inf  0.000000 -6.845133  0.000000 -5.417470
## Borne sup  4.844949  6.773429  5.349863  7.924373
```

Avec $n = 1000$ et $\sigma = 1$, le constat est le même.

Etude des rapports d'amplitude et des longueurs des intervalles de confiance

Initialisation

```
liste_longueurs_nnls_1 <- vector("list", taille)
liste_longueurs_nnls_2 <- vector("list", taille)

liste_longueurs_lm_1 <- vector("list", taille)
liste_longueurs_lm_2 <- vector("list", taille)
```

Calcul des longueurs des intervalles de confiance

```
for(i in 1:taille)
{
  liste_longueurs_nnls_1[[i]] <- diff(liste_IC_df[[i]]$IC_nnls_1)
  liste_longueurs_nnls_2[[i]] <- diff(liste_IC_df[[i]]$IC_nnls_2)
  liste_longueurs_lm_1[[i]] <- diff(liste_IC_df[[i]]$IC_lm_1)
  liste_longueurs_lm_2[[i]] <- diff(liste_IC_df[[i]]$IC_lm_2)
}
```

Heatmaps

On construit un dataframe contenant les valeurs de n, les valeurs de σ et les rapports d'amplitude des intervalles de confiance (pour $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{lm_1}$).

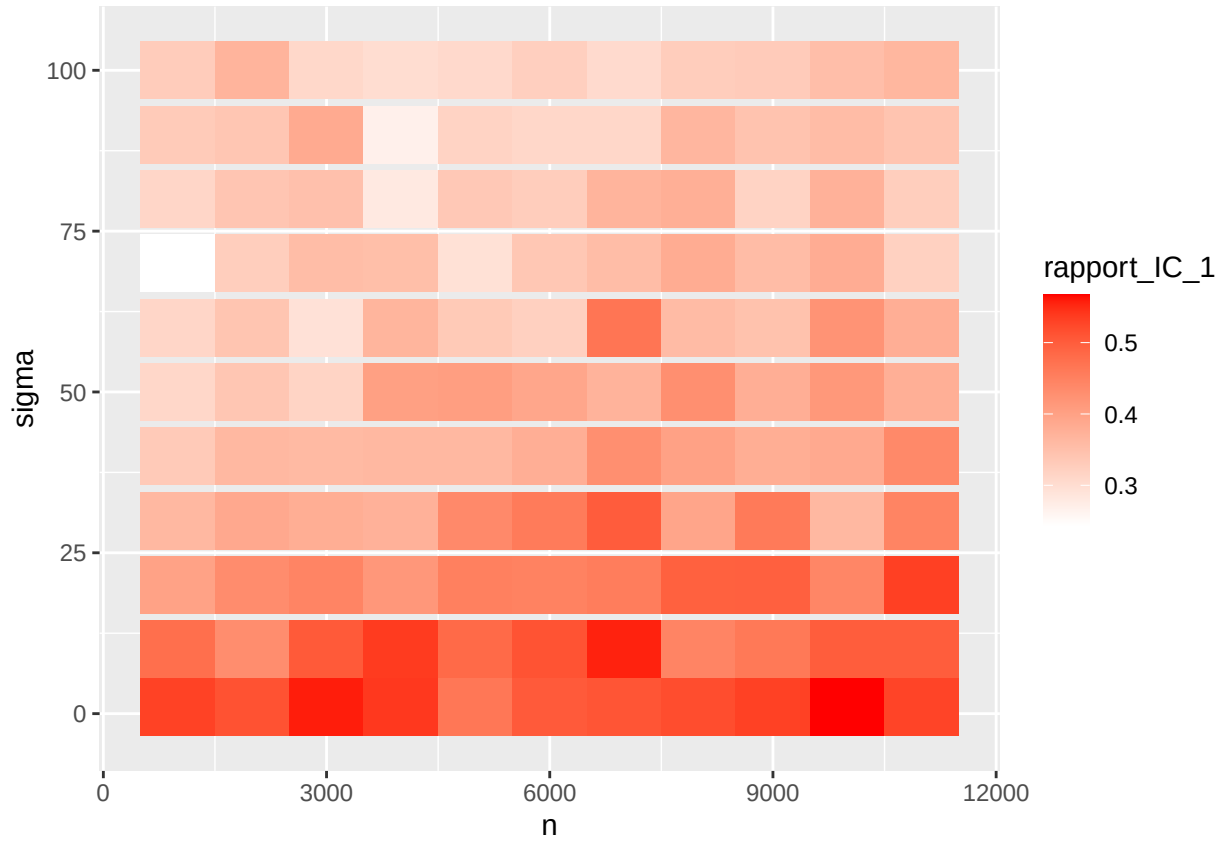
```
# Dans cet ordre sinon les rapports d'amplitude
# sont mal attribués, on fait d'abord varier sigma,
# puis n dans nos simulations
amplitude_1 <- expand.grid(sigma = liste_sigma, n = liste_n)

# Ajout d'une colonne pour les rapports d'amplitude
amplitude_1$rapport_IC_1 <- unlist(liste_rapports_1)

# Ajout de colonnes pour les longueurs
amplitude_1$longueurs_nnls_1 <- unlist(liste_longueurs_nnls_1)
```

```r
amplitude_1$longueurs_lm_1 <- unlist(liste_longueurs_lm_1)

# Ajout d'une colonne "text"
amplitude_1 <- amplitude_1 %>%
  mutate(text = paste0("n = ", n, "\n",
                       "sigma = ", sigma, "\n",
                       "amplitude_nnls = ", longueurs_nnls_1, "\n",
                       "amplitude_lm = ", longueurs_lm_1, "\n",
                       "rapport_IC_1 = ", rapport_IC_1))

# on échange deux colonnes
amplitude_1 <- amplitude_1[, c("n", "sigma", "longueurs_nnls_1", "longueurs_lm_1",
                               "rapport_IC_1", "text")]
```

On construit un heatmap qui illustre la variation du rapport des intervalles de confiance en fonction de n et de σ.

```r
heatmap_1 <- ggplot(
  data = amplitude_1,
  aes(n, sigma, fill = rapport_IC_1, text = text)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  theme_ipsum()
```

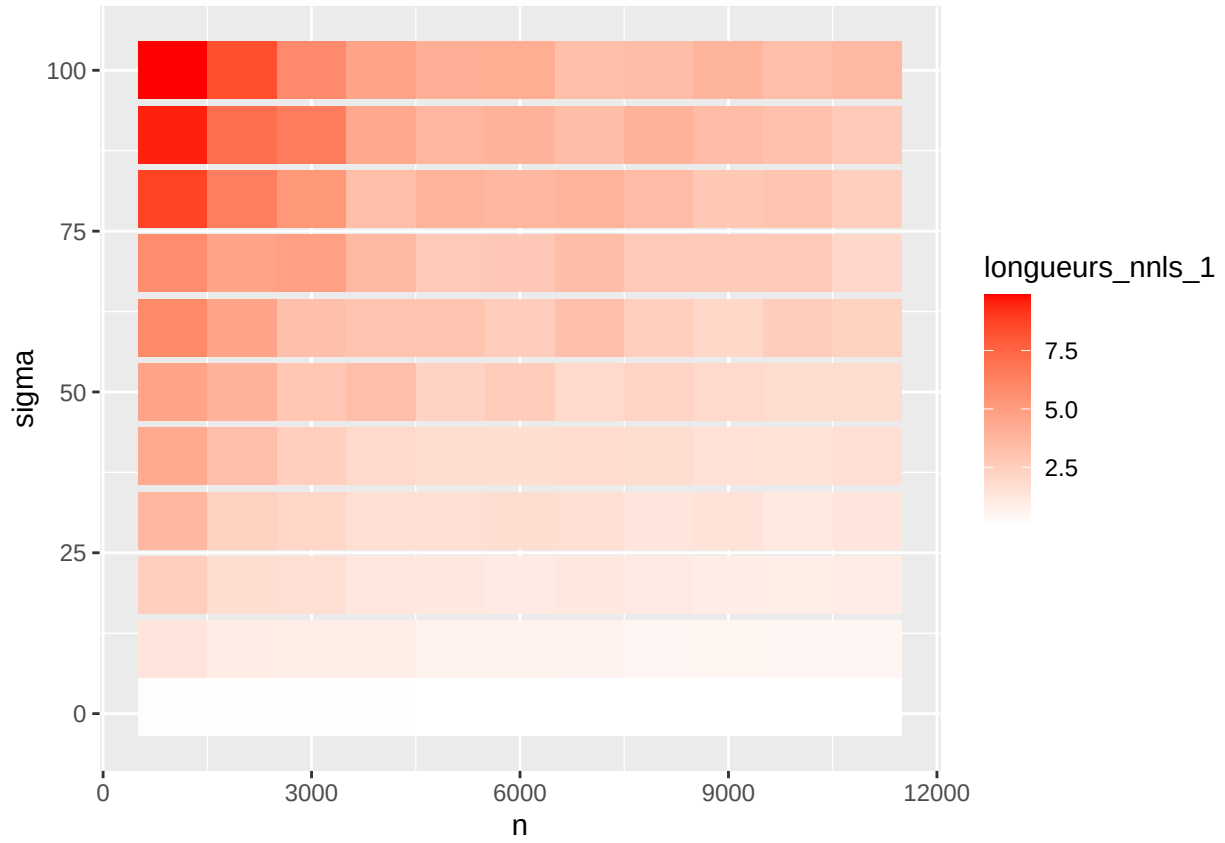
```r
ggplot(
  data = amplitude_1,
  aes(n, sigma, fill = rapport_IC_1)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red")
```



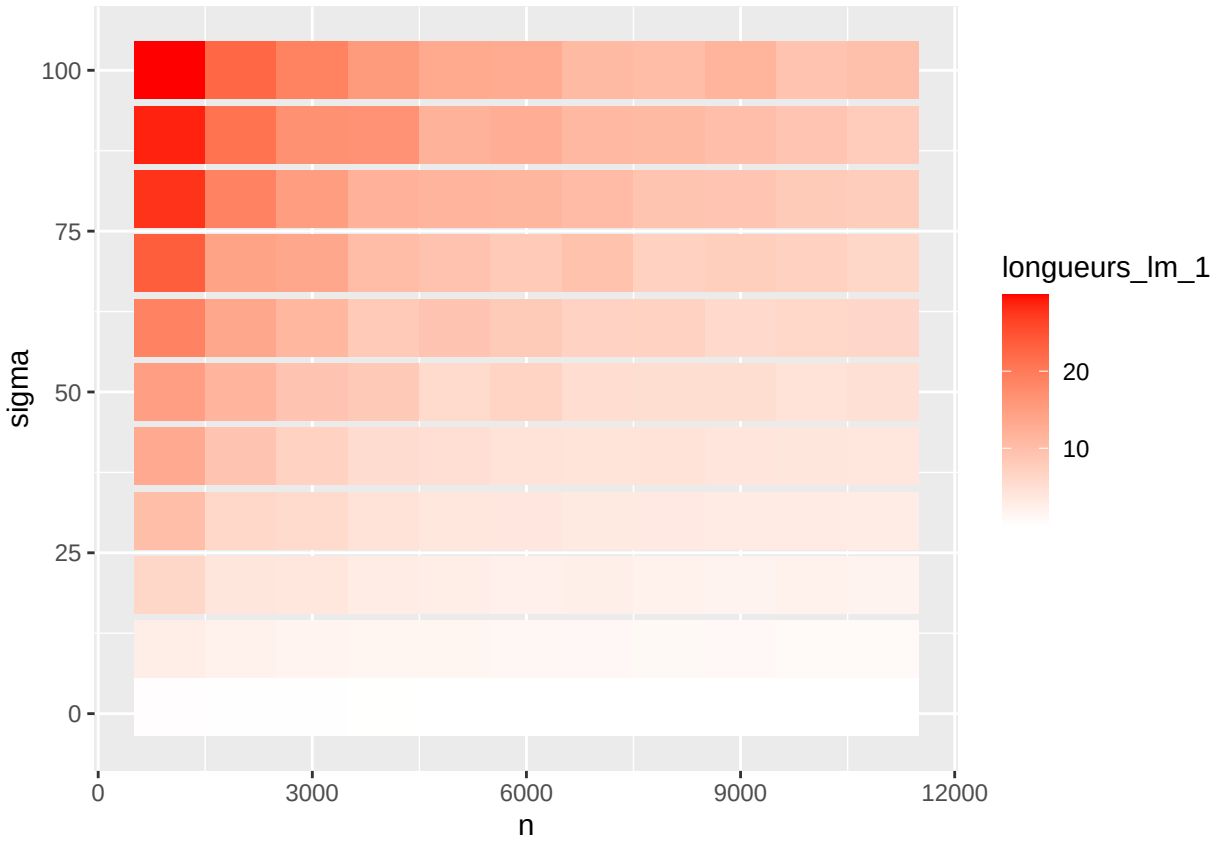
Cet heatmap permet de remarquer que les rapports d'amplitude des intervalles de confiance (pour $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{lm_1}$) ont tendance à diminuer lorsque σ augmente.

Etudions maintenant les longueurs de chacun des intervalles de confiance (pour $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{lm_1}$).

```
ggplot(
  data = amplitude_1,
  aes(n, sigma, fill = longueurs_nnls_1)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red")
```




```
ggplot(
  data = amplitude_1,
  aes(n, sigma, fill = longueurs_lm_1)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red")
```



Le fait que les rapports des intervalles de confiance diminuent lorsque σ augmente vient du fait que les longueurs des intervalles de confiance obtenus pour $\hat{\beta}_{nnls_1}$ augmentent moins vite que celles des intervalles de confiance obtenus pour $\hat{\beta}_{lm_1}$.

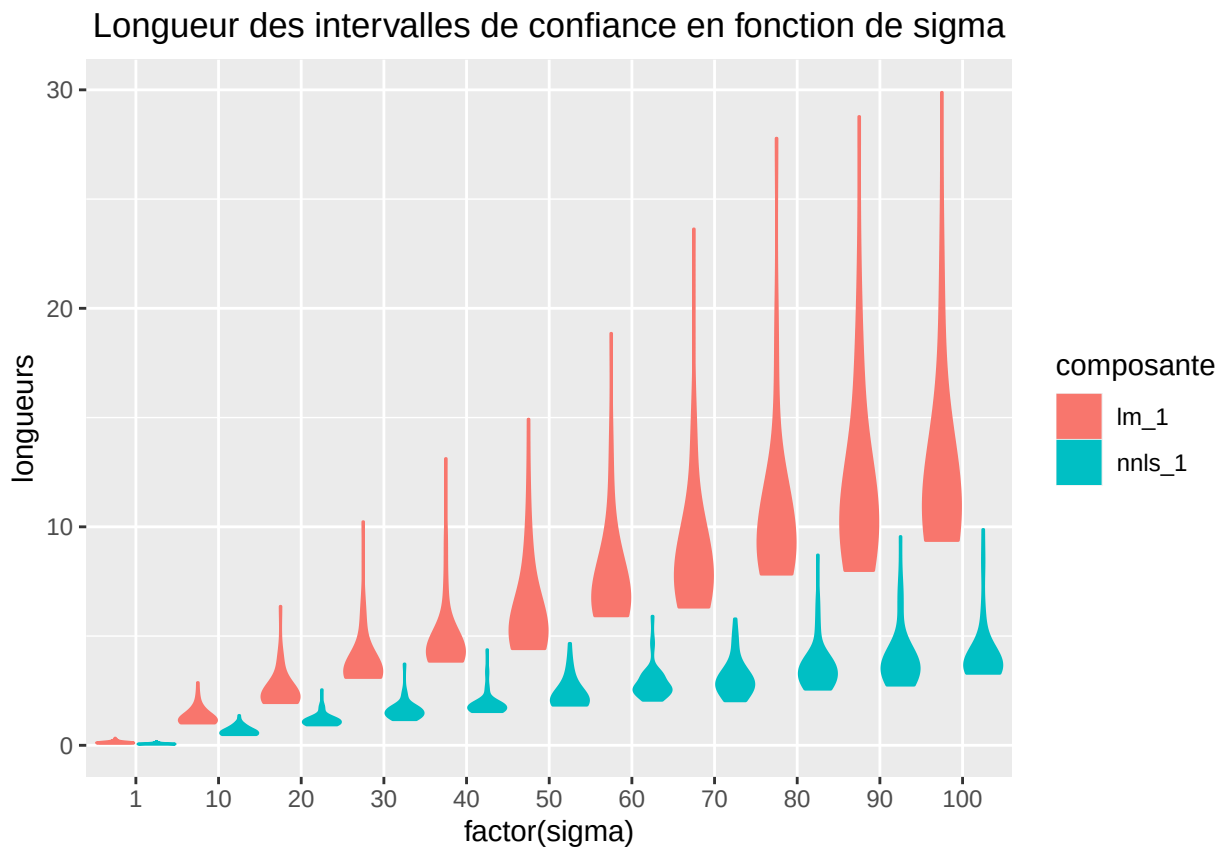
Pour s'en convaincre on peut construire des violin plots et des nuages de points correspondant aux longueurs des intervalles de confiance obtenus pour $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{lm_1}$ en fonction de σ pour toutes les valeurs de n.

```
liste_violin_plots_1 <- vector("list", length(liste_n))
```

```
longueurs_amplitude_1 <-
  data.frame(n = amplitude_1$n, sigma = amplitude_1$sigma,
             longueurs = c(amplitude_1$longueurs_nnls_1,
                           amplitude_1$longueurs_lm_1),
             composante = c(rep("nnls_1",
                                length(amplitude_1$longueurs_nnls_1)),
                            rep("lm_1", length(amplitude_1$longueurs_lm_1))
                            )
             )
```

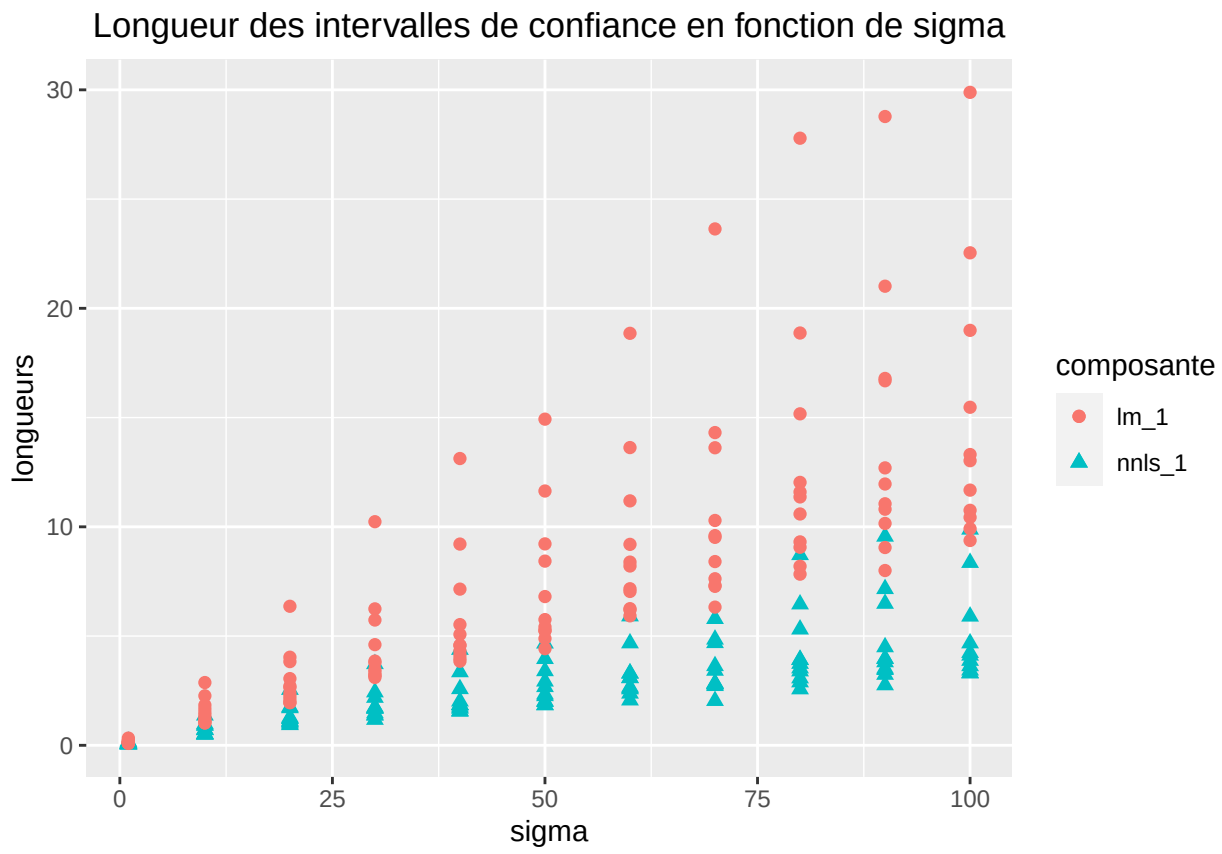
Violin plots pour toutes les valeurs de n.

```
ggplot(subset(longueurs_amplitude_1),
       aes(x = factor(sigma), y = longueurs, width = n,
           color = composante, fill = composante)) +
geom_violin(scale = "width", position=position_dodge(width=1)) +
  ggtitle(paste0(
    "Longueur des intervalles de confiance en fonction de sigma"))
```



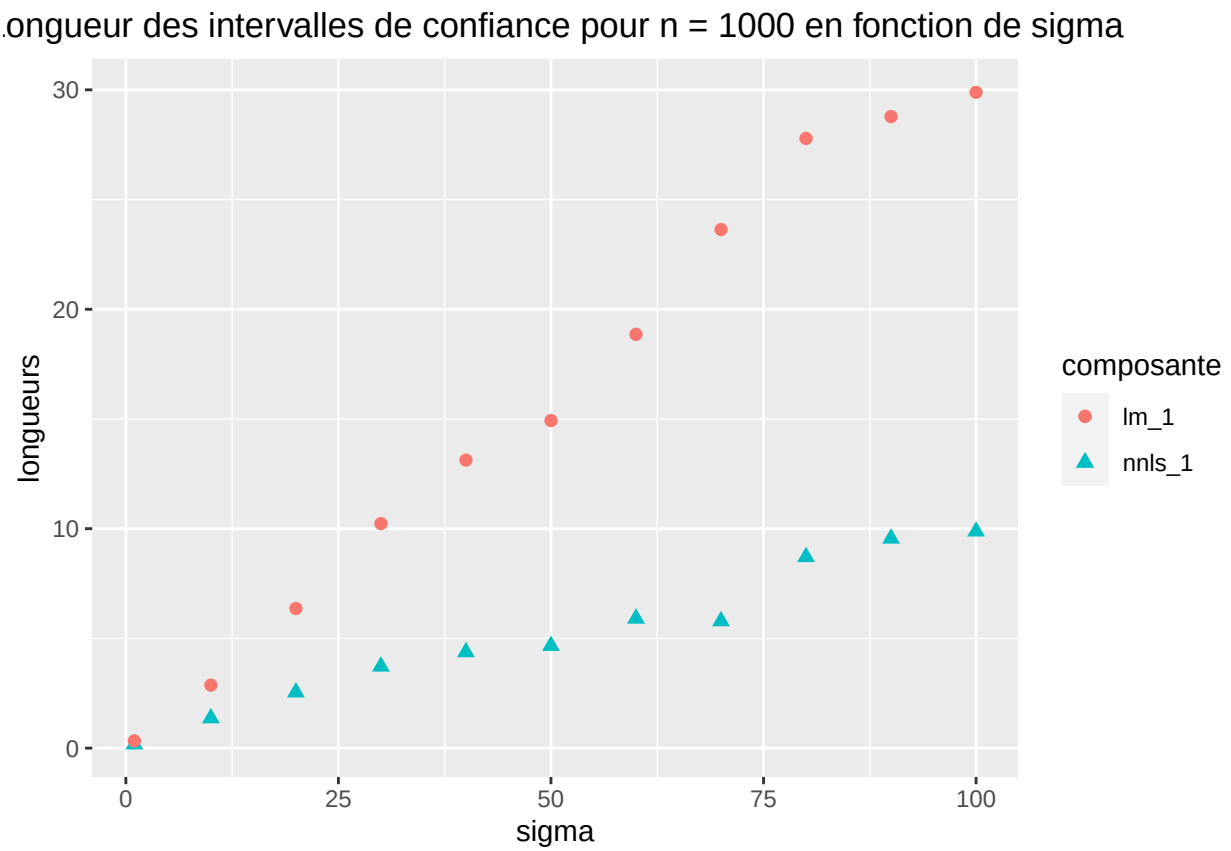
Nuages de points pour toutes les valeurs de n.

```
ggplot(longueurs_amplitude_1, aes(x = sigma, y = longueurs)) +
  geom_point(data = subset(longueurs_amplitude_1,
                           composante == "nnls_1"),
             aes(colour = composante, shape = composante), size = 2) +
  geom_point(data = subset(longueurs_amplitude_1,
                           composante == "lm_1"),
             aes(colour = composante, shape = composante), size = 2) +
  ggtitle(
    "Longueur des intervalles de confiance en fonction de sigma"
    )
```



Nuages de points pour $n = 1000$.

```
ggplot(longueurs_amplitude_1, aes(x = sigma, y = longueurs)) +
  geom_point(data = subset(longueurs_amplitude_1,
                           composante == "nnls_1" & n == 1000),
             aes(colour = composante, shape = composante), size = 2) +
  geom_point(data = subset(longueurs_amplitude_1,
                           composante == "lm_1" & n == 1000),
             aes(colour = composante, shape = composante), size = 2) +
  ggtitle(
    "Longueur des intervalles de confiance pour n = 1000 en fonction de sigma"
    )
```

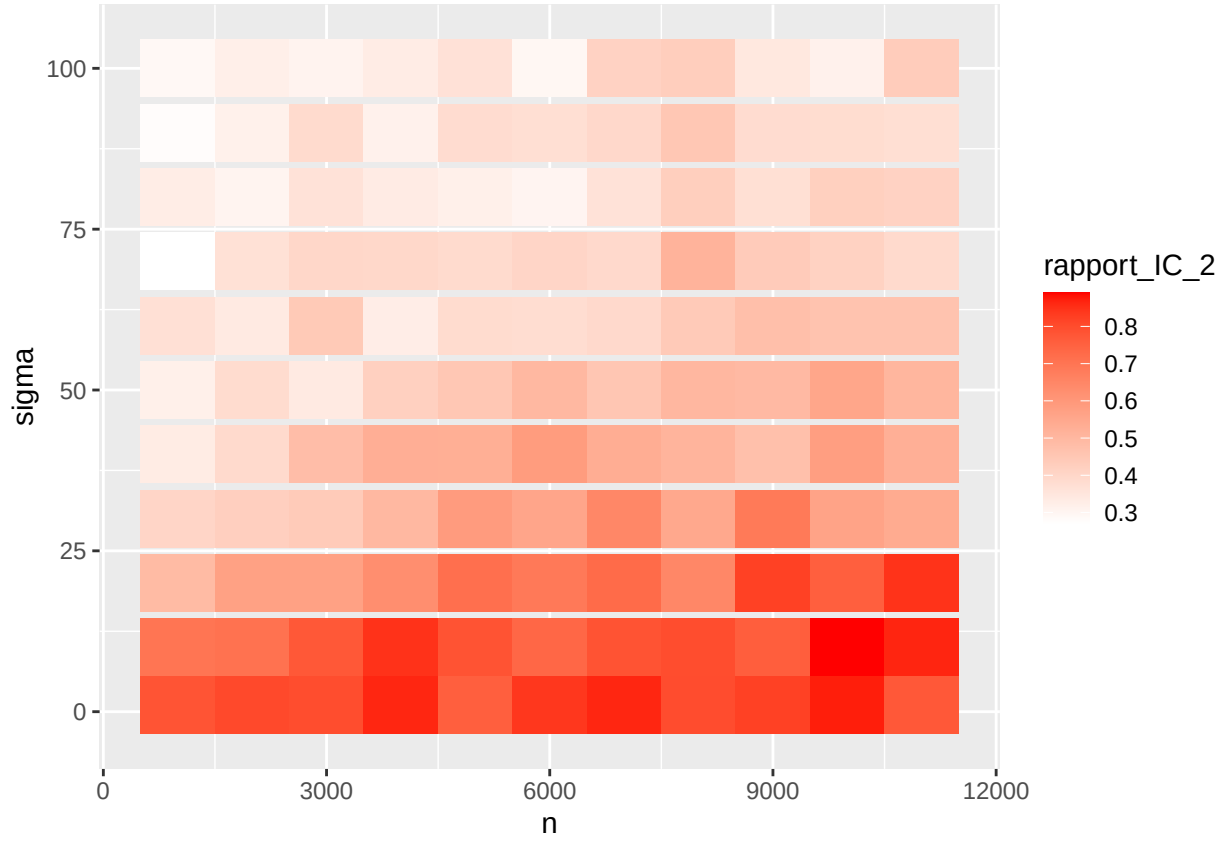


On construit un dataframe contenant les valeurs de n, les valeurs de σ et les rapports d'amplitude des intervalles de confiance (pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$).

```
# Dans cet ordre sinon les rapports d'amplitude
# sont mal attribués, on fait d'abord varier sigma,
# puis n dans nos simulations
amplitude_2 <- expand.grid(sigma = liste_sigma, n = liste_n)
# Ajout d'une colonne pour les rapports d'amplitude
amplitude_2$rapport_IC_2 <- unlist(liste_rapports_2)
amplitude_2$longueurs_nnls_2 <- unlist(liste_longueurs_nnls_2)
amplitude_2$longueurs_lm_2 <- unlist(liste_longueurs_lm_2)


# Ajout d'une colonne "text"
amplitude_2 <- amplitude_2 %>%
  mutate(text = paste0("n = ", n, "\n",
                       "sigma = ", sigma, "\n",
                       "amplitude_nnls = ", longueurs_nnls_2,
                       "amplitude_lm = ", longueurs_lm_2,
                       "rapport_IC_2 = ", rapport_IC_2))

# on échange deux colonnes
amplitude_2 <- amplitude_2[, c("n", "sigma", "longueurs_nnls_2", "longueurs_lm_2",
                               "rapport_IC_2", "text")]
```

On construit un heatmap qui illustre la variation du rapport des intervalles de confiance en fonction de n et de σ.

```
ggplot(
  data = amplitude_2,
  aes(n, sigma, fill = rapport_IC_2)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red")
```



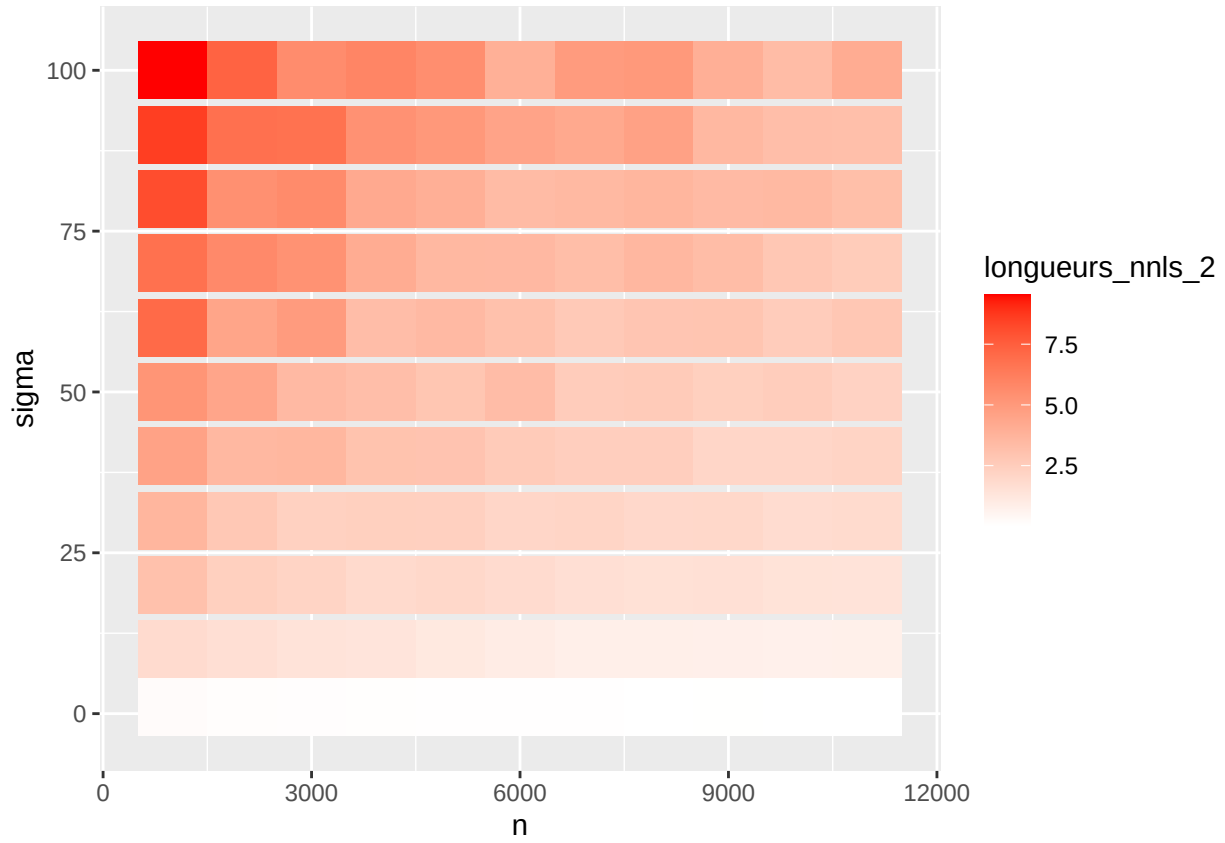
Cet heatmap permet de remarquer que les rapports d'amplitude des intervalles de confiance (pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$) ont également tendance à diminuer lorsque σ augmente.

On constate ici que les rapports des intervalles de confiance obtenus pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$ sont généralement plus élevés que ceux obtenus pour i$\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{lm_1}$. Autrement dit, les amplitudes des intervalles de confiance obtenus sont plus proches pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$ par rapport à ceux de $\hat{\beta}_{nnls_1}$ et $\hat{\beta}_{lm_1}$.

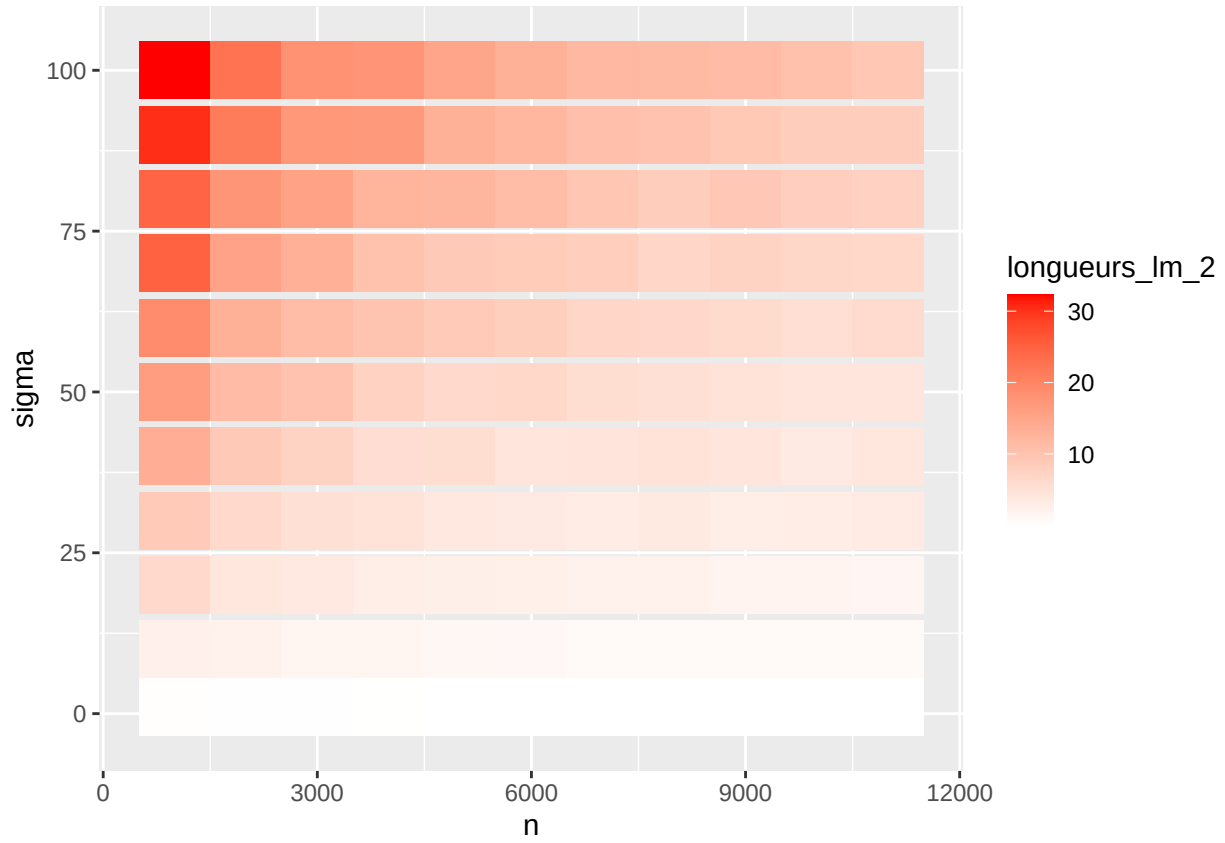
Cela s'explique par le fait que plus β_i s'éloigne de 0, plus $\hat{\beta}_{nnls_i}$ se rapproche de $\hat{\beta}_{lm_i}$ pour tout $i \in \{1, 2\}$.

Etudions maintenant les longueurs de chacun des intervalles de confiance (pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$).

```
ggplot(
  data = amplitude_2,
  aes(n, sigma, fill = longueurs_nnls_2)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red")
```




```
ggplot(
  data = amplitude_2,
  aes(n, sigma, fill = longueurs_lm_2)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red")
```



On remarque que les longueurs des intervalles de confiance obtenus pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$ augmente lorsque σ augmente ce qui est cohérent (par construction, la longueur des intervalles de confiance augmente lorsque σ augmente).

Le fait que les rapports des intervalles de confiance diminuent lorsque σ augmente vient du fait que les longueurs des intervalles de confiance obtenus pour $\hat{\beta}_{nnls_2}$ augmentent moins vite que celles des intervalles de confiance obtenus pour $\hat{\beta}_{lm_2}$.

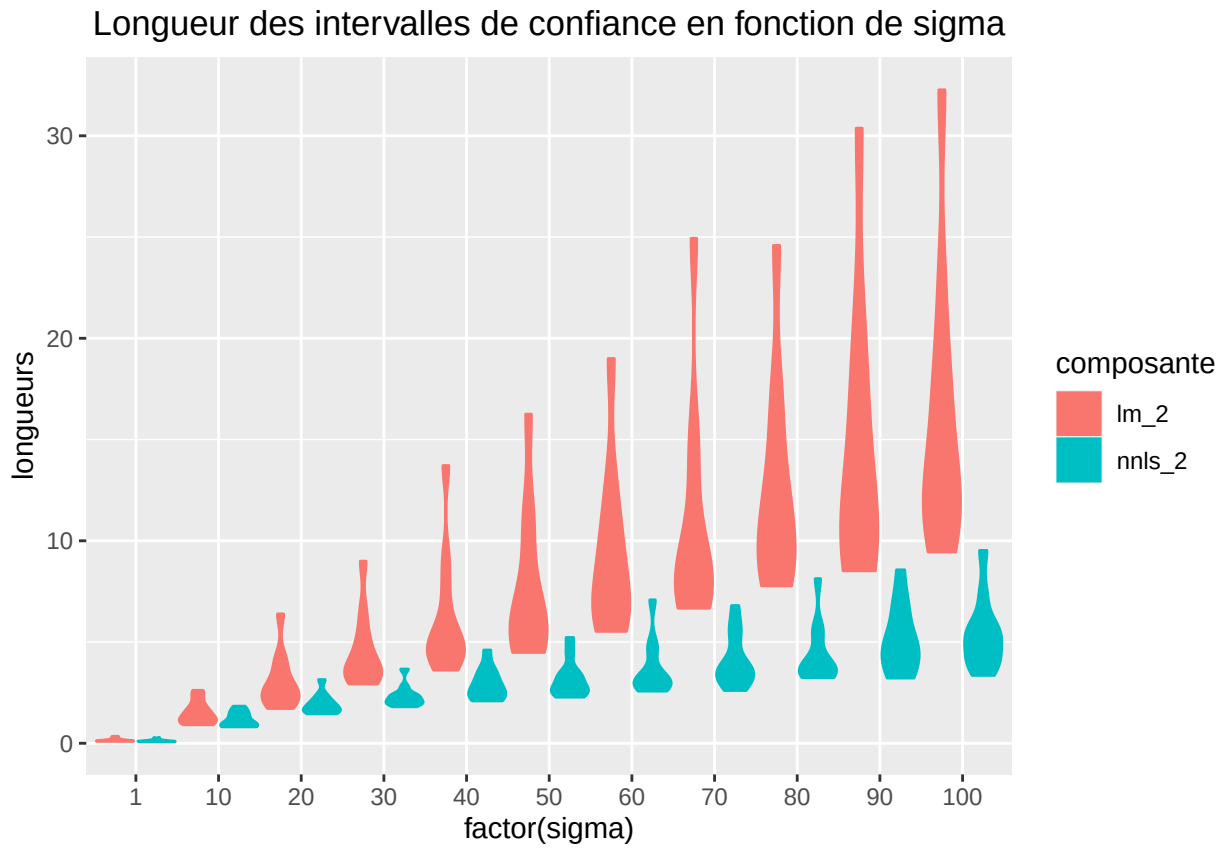
Pour s'en convaincre on peut construire des violin plots et des nuages de points correspondant aux longueurs des intervalles de confiance obtenus pour $\hat{\beta}_{nnls_2}$ et $\hat{\beta}_{lm_2}$ en fonction de σ pour toutes les valeurs de n.

```
longueurs_amplitude_2 <-
  data.frame(n = amplitude_2$n, sigma = amplitude_2$sigma,
             longueurs = c(amplitude_2$longueurs_nnls_2,
                           amplitude_2$longueurs_lm_2),
             composante = c(rep("nnls_2",
                                length(amplitude_2$longueurs_nnls_2)),
                            rep("lm_2", length(amplitude_2$longueurs_lm_2))
                            )
             )
```

Violin plots pour toutes les valeurs de n.

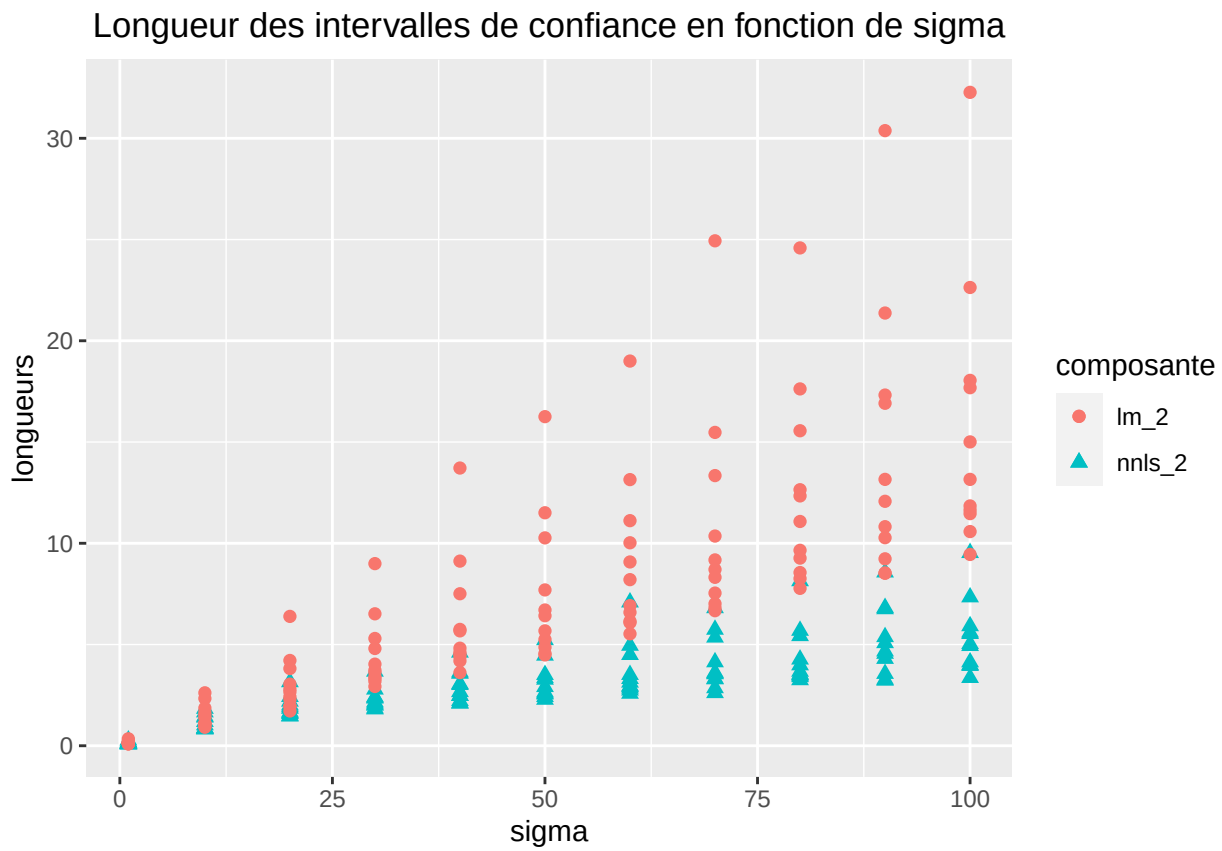
```
ggplot(subset(longueurs_amplitude_2),
       aes(x = factor(sigma), y = longueurs,
           color = composante, fill = composante)) +
geom_violin(scale = "width", position=position_dodge(width=1)) +
  ggtitle(paste0(
```

```
    "Longueur des intervalles de confiance en fonction de sigma"))
```



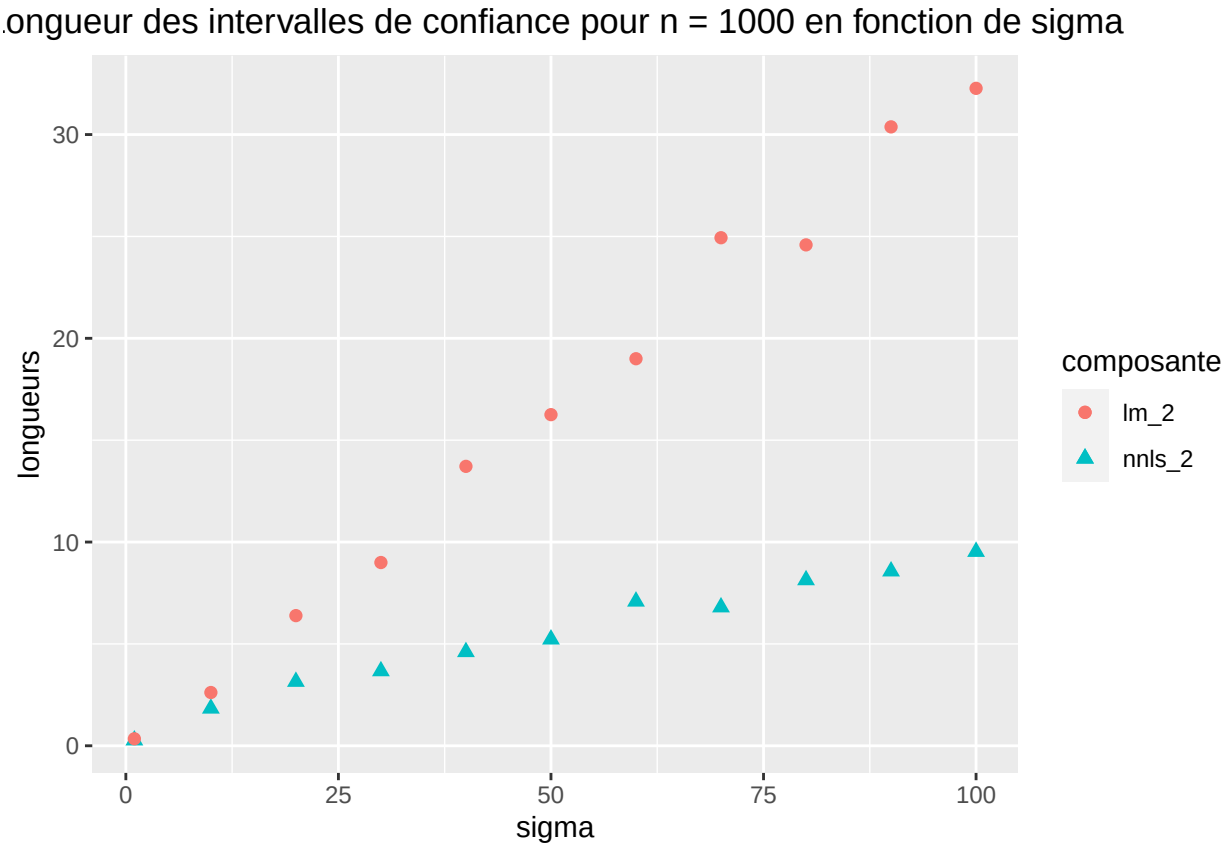
Nuages de points pour toutes les valeurs de n.

```
ggplot(longueurs_amplitude_2, aes(x = sigma, y = longueurs)) +
  geom_point(data = subset(longueurs_amplitude_2,
                           composante == "nnls_2"),
             aes(colour = composante, shape = composante), size = 2) +
  geom_point(data = subset(longueurs_amplitude_2,
                           composante == "lm_2"),
             aes(colour = composante, shape = composante), size = 2) +
  ggtitle(
    "Longueur des intervalles de confiance en fonction de sigma"
    )
```



Nuages de points pour $n = 1000$.

```
ggplot(longueurs_amplitude_2, aes(x = sigma, y = longueurs)) +
  geom_point(data = subset(longueurs_amplitude_2,
                           composante == "nnls_2" & n == 1000),
             aes(colour = composante, shape = composante), size = 2) +
  geom_point(data = subset(longueurs_amplitude_2,
                           composante == "lm_2" & n == 1000),
             aes(colour = composante, shape = composante), size = 2) +
  ggtitle("n = 10 000") +
  ggtitle(
    "Longueur des intervalles de confiance pour n = 1000 en fonction de sigma"
    )
```



Comparaison simulation / valeurs théoriques

Dans cette partie, nous allons comparer les valeurs simulées obtenues pour toutes les valeurs de n et de σ choisies avec les résultats théoriques.

Pour cela, nous commençons par construire des listes qui contiendront les résultats théoriques. La liste des indices servira pour la comparaison.

```
liste_indices <- vector("list", taille)

liste_biais_nnls_1_simu <- vector("list", taille)
liste_biais_nnls_2_simu <- vector("list", taille)

liste_biais_nnls_1_theorique <- vector("list", taille)
liste_biais_nnls_2_theorique <- vector("list", taille)

liste_variance_nnls_1_simu <- vector("list", taille)
liste_variance_nnls_2_simu <- vector("list", taille)

liste_variance_nnls_1_theorique <- vector("list", taille)
liste_variance_nnls_2_theorique <- vector("list", taille)

liste_MSE_nnls_1_simu <- vector("list", taille)
liste_MSE_nnls_2_simu <- vector("list", taille)

liste_MSE_nnls_1_theorique <- vector("list", taille)
liste_MSE_nnls_2_theorique <- vector("list", taille)
```

On remplit les listes.

```
for(i in 1:taille)
{
  liste_indices[[i]] <- i

  liste_biais_nnls_1_simu[[i]] <-
    liste_resultats_df[[i]]["nnls_1", "biais"]
  liste_biais_nnls_2_simu[[i]] <-
    liste_resultats_df[[i]]["nnls_2", "biais"]


  liste_biais_nnls_1_theorique[[i]] <-
    liste_resultats_theoriques_df[[i]]["nnls_1", "biais"]
  liste_biais_nnls_2_theorique[[i]] <-
    liste_resultats_theoriques_df[[i]]["nnls_2", "biais"]


  liste_variance_nnls_1_simu[[i]] <-
    liste_resultats_df[[i]]["nnls_1", "variance"]
  liste_variance_nnls_2_simu[[i]] <-
    liste_resultats_df[[i]]["nnls_2", "variance"]

  liste_variance_nnls_1_theorique[[i]] <-
    liste_resultats_theoriques_df[[i]]["nnls_1", "variance"]
  liste_variance_nnls_2_theorique[[i]] <-
    liste_resultats_theoriques_df[[i]]["nnls_2", "variance"]

  liste_MSE_nnls_1_simu[[i]] <-
    liste_resultats_df[[i]]["nnls_1", "MSE"]
  liste_MSE_nnls_2_simu[[i]] <-
    liste_resultats_df[[i]]["nnls_2", "MSE"]


  liste_MSE_nnls_1_theorique[[i]] <-
    liste_resultats_theoriques_df[[i]]["nnls_1", "MSE"]
  liste_MSE_nnls_2_theorique[[i]] <-
    liste_resultats_theoriques_df[[i]]["nnls_2", "MSE"]
}
```

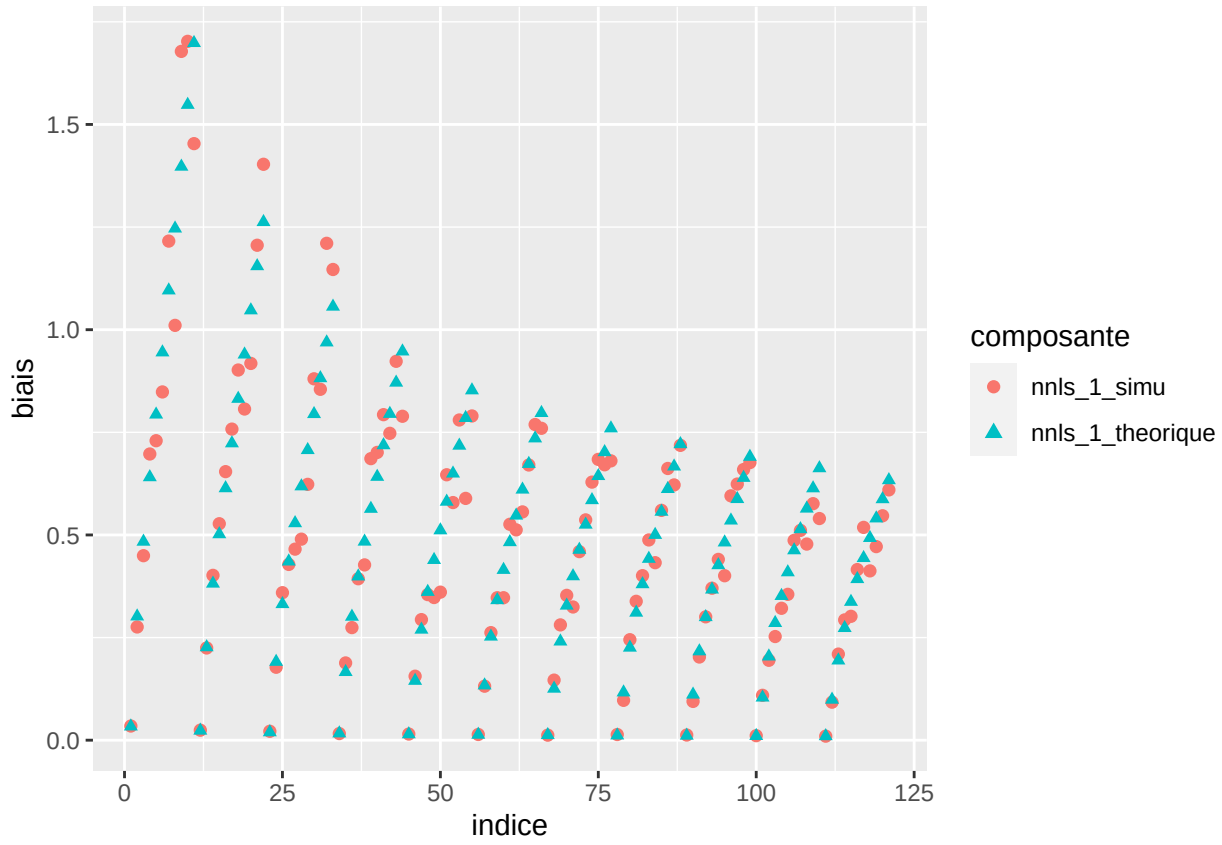
On construit deux dataframes qui nous serviront on construire les nuages de points.

```
comparaison_biais_theorique_simu <-
  data.frame(indice = unlist(liste_indices),
             biais = c(unlist(liste_biais_nnls_1_simu),
                       unlist(liste_biais_nnls_1_theorique),
                       unlist(liste_biais_nnls_2_simu),
                       unlist(liste_biais_nnls_2_theorique)),
             composante = c(rep("nnls_1_simu",
                                length(liste_biais_nnls_1_simu)),
                            rep("nnls_1_theorique",
                                length(liste_biais_nnls_1_theorique)),
                            rep("nnls_2_simu",
                                length(liste_biais_nnls_2_simu)),
                            rep("nnls_2_theorique",
```

```
                                length(liste_biais_nnls_2_theorique))))


comparaison_variance_theorique_simu <-
  data.frame(indice = unlist(liste_indices),
             variance = c(unlist(liste_variance_nnls_1_simu),
                     unlist(liste_variance_nnls_1_theorique),
                     unlist(liste_variance_nnls_2_simu),
                     unlist(liste_variance_nnls_2_theorique)),
             composante = c(rep("nnls_1_simu",
                                length(liste_variance_nnls_1_simu)),
                            rep("nnls_1_theorique",
                                length(liste_variance_nnls_1_theorique)),
                            rep("nnls_2_simu",
                                length(liste_variance_nnls_2_simu)),
                            rep("nnls_2_theorique",
                                length(liste_variance_nnls_2_theorique))))


comparaison_MSE_theorique_simu <-
  data.frame(indice = unlist(liste_indices),
             MSE = c(unlist(liste_MSE_nnls_1_simu),
                     unlist(liste_MSE_nnls_1_theorique),
                     unlist(liste_MSE_nnls_2_simu),
                     unlist(liste_MSE_nnls_2_theorique)),
             composante = c(rep("nnls_1_simu",
                                length(liste_MSE_nnls_1_simu)),
                            rep("nnls_1_theorique",
                                length(liste_MSE_nnls_1_theorique)),
                            rep("nnls_2_simu",
                                length(liste_MSE_nnls_2_simu)),
                            rep("nnls_2_theorique",
                                length(liste_MSE_nnls_2_theorique))))
```

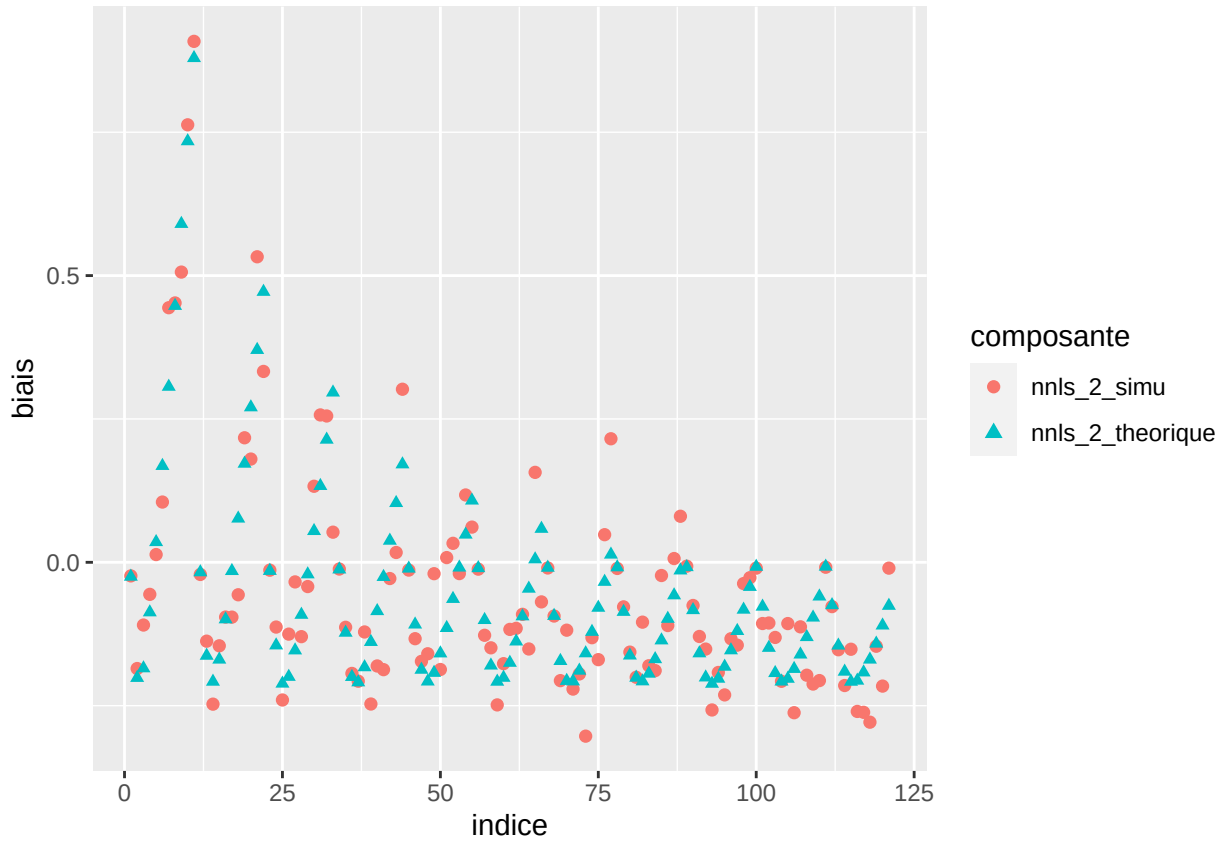
Comparaison des valeurs du biais pour la première composante.

```
ggplot(comparaison_biais_theorique_simu,
               aes(x = indice, y = biais)) +
        geom_point(data = subset(comparaison_biais_theorique_simu,
                                 composante == "nnls_1_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_biais_theorique_simu,
                                 composante == "nnls_1_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



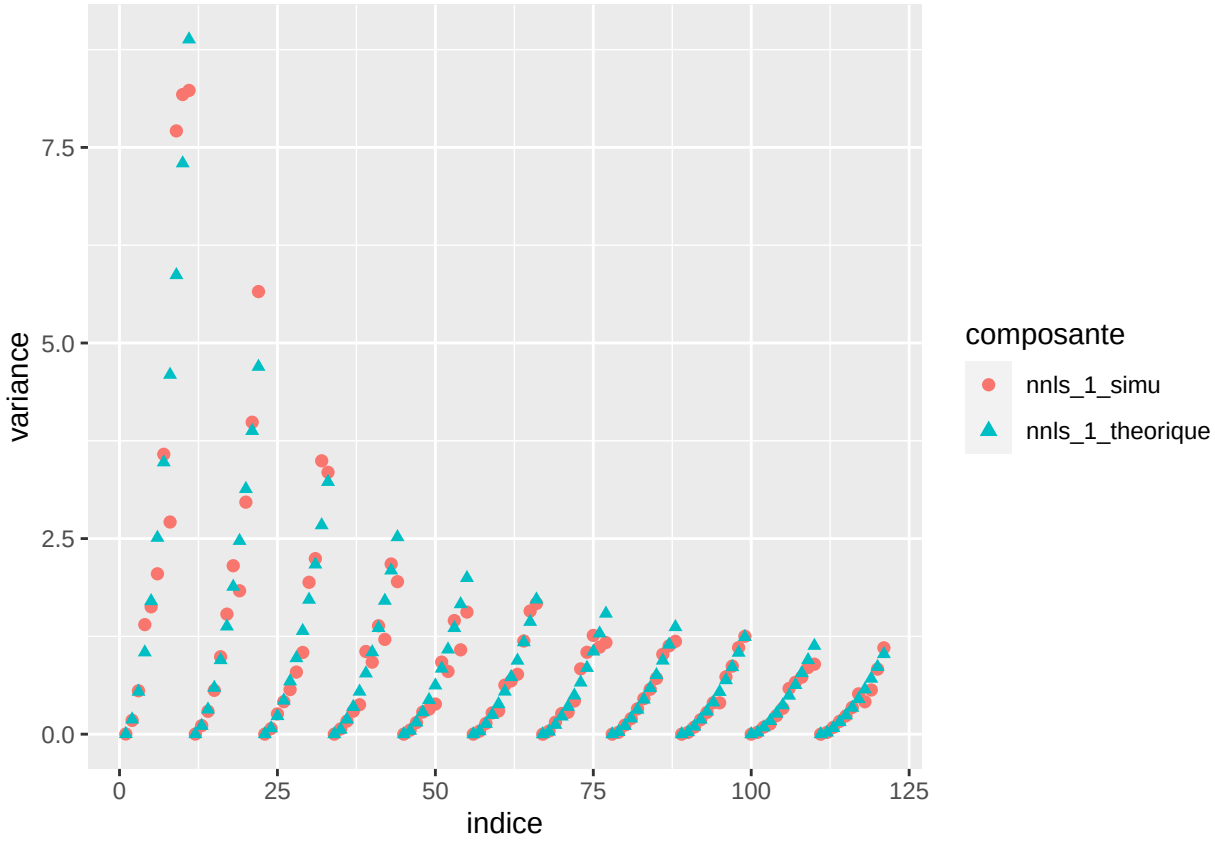
Comparaison des valeurs du biais pour la seconde composante.

```
ggplot(comparaison_biais_theorique_simu,
               aes(x = indice, y = biais)) +
        geom_point(data = subset(comparaison_biais_theorique_simu,
                                 composante == "nnls_2_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_biais_theorique_simu,
                                 composante == "nnls_2_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



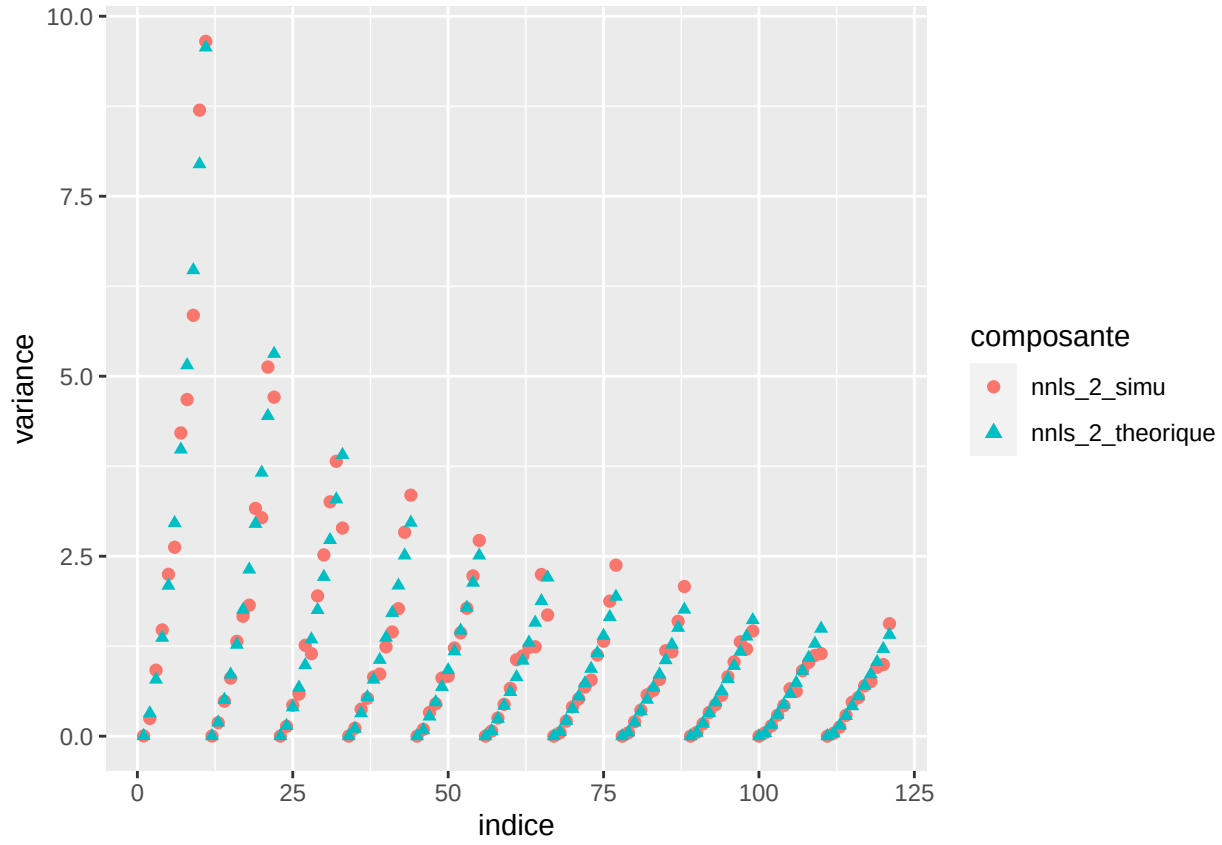
Comparaison des valeurs de la variance pour la première composante.

```
ggplot(comparaison_variance_theorique_simu,
               aes(x = indice, y = variance)) +
        geom_point(data = subset(comparaison_variance_theorique_simu,
                                 composante == "nnls_1_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_variance_theorique_simu,
                                 composante == "nnls_1_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



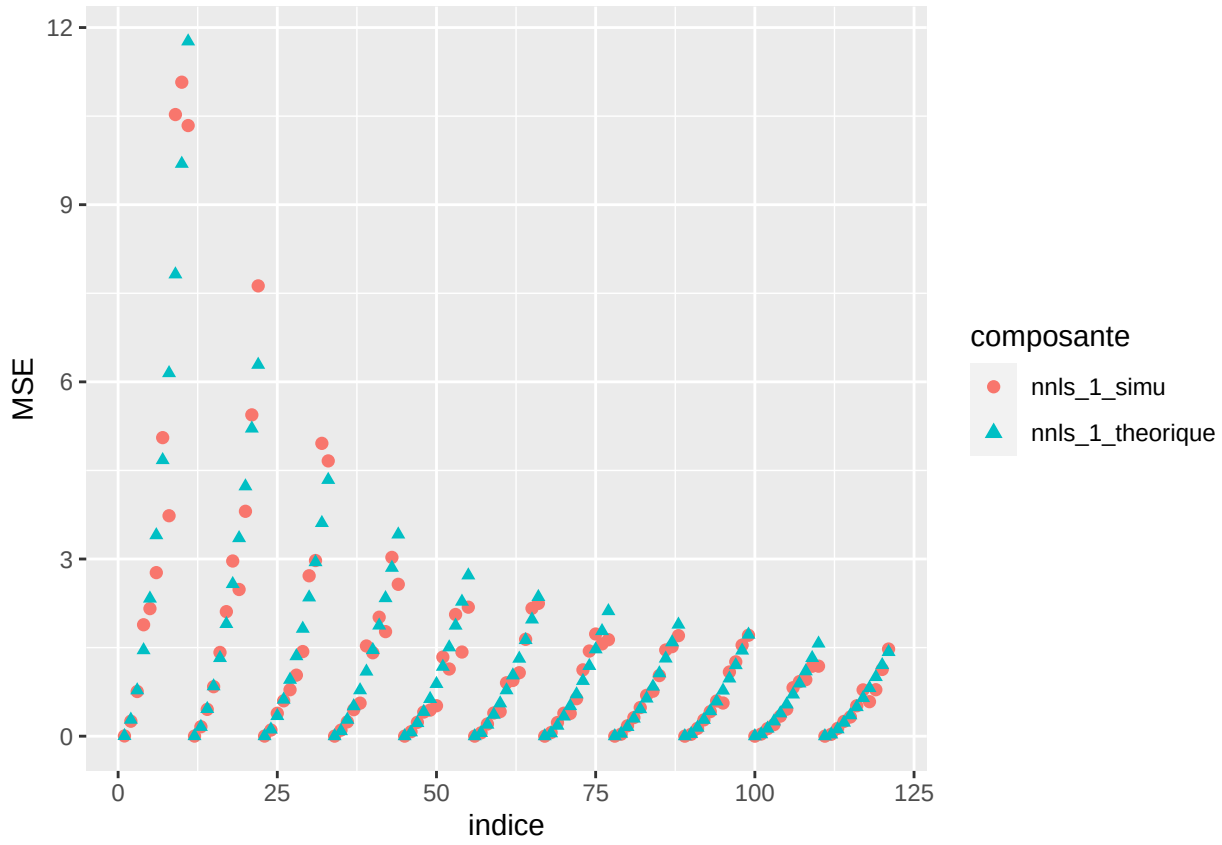
Comparaison des valeurs de la variance pour la deuxième composante.

```
ggplot(comparaison_variance_theorique_simu,
               aes(x = indice, y = variance)) +
        geom_point(data = subset(comparaison_variance_theorique_simu,
                                 composante == "nnls_2_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_variance_theorique_simu,
                                 composante == "nnls_2_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



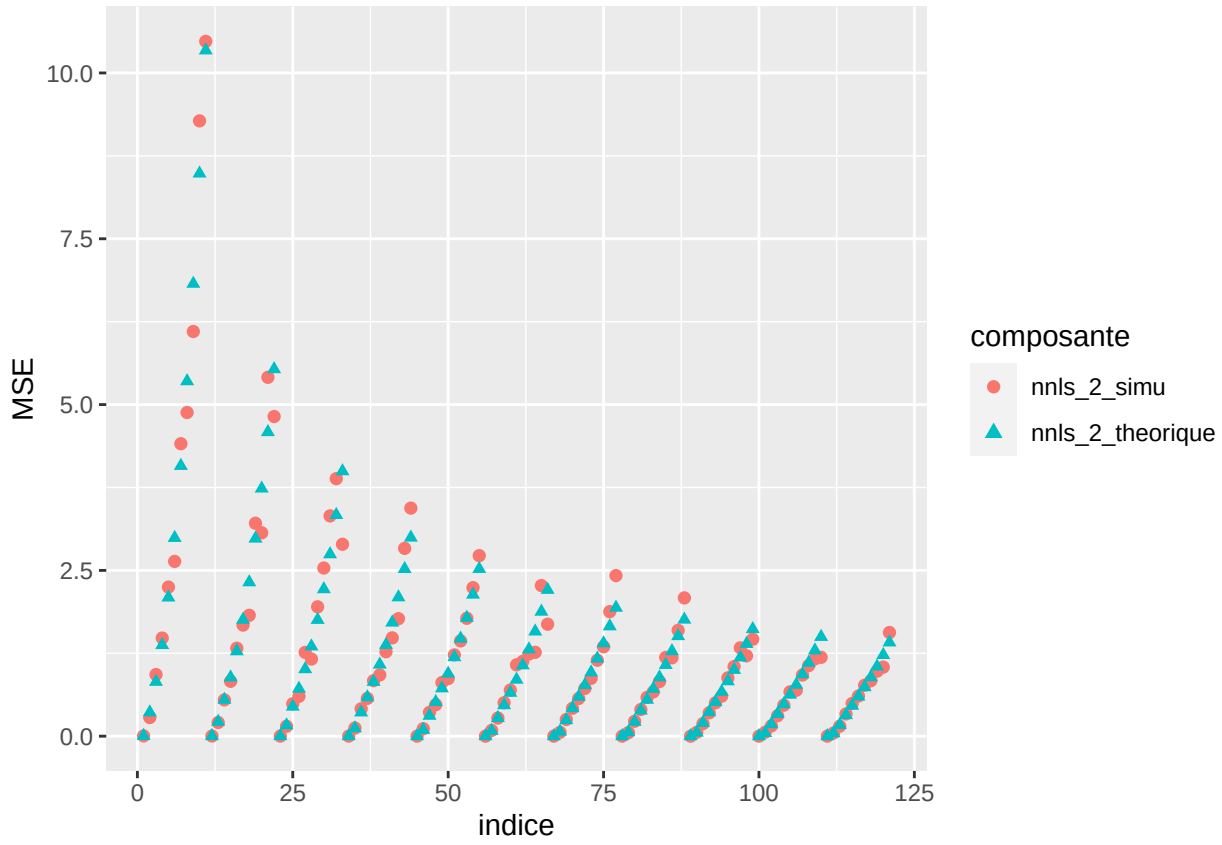
Comparaison des valeurs de l'erreur quadratique moyenne pour la première composante.

```
ggplot(comparaison_MSE_theorique_simu,
               aes(x = indice, y = MSE)) +
        geom_point(data = subset(comparaison_MSE_theorique_simu,
                                 composante == "nnls_1_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_MSE_theorique_simu,
                                 composante == "nnls_1_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



Comparaison des valeurs de l'erreur quadratique moyenne pour la seconde composante.

```
ggplot(comparaison_MSE_theorique_simu,
               aes(x = indice, y = MSE)) +
        geom_point(data = subset(comparaison_MSE_theorique_simu,
                                 composante == "nnls_2_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_MSE_theorique_simu,
                                 composante == "nnls_2_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



On conclut que les valeurs simulées pour le biais, la variance et l'erreur quadratique moyenne sont généralement proches des résultats théoriques. C'est ce que l'on souhaitait.

Comparaison simulation / valeurs théoriques sans la dépendance en n

```
liste_biais_nnls_1_simu_sans_dependance <- vector("list", taille)
liste_biais_nnls_2_simu_sans_dependance <- vector("list", taille)

liste_biais_nnls_1_theorique_sans_dependance <- vector("list", taille)
liste_biais_nnls_2_theorique_sans_dependance <- vector("list", taille)

liste_variance_nnls_1_simu_sans_dependance <- vector("list", taille)
liste_variance_nnls_2_simu_sans_dependance <- vector("list", taille)

liste_variance_nnls_1_theorique_sans_dependance <- vector("list", taille)
liste_variance_nnls_2_theorique_sans_dependance <- vector("list", taille)

liste_MSE_nnls_1_simu_sans_dependance <- vector("list", taille)
liste_MSE_nnls_2_simu_sans_dependance <- vector("list", taille)

liste_MSE_nnls_1_theorique_sans_dependance <- vector("list", taille)
liste_MSE_nnls_2_theorique_sans_dependance <- vector("list", taille)
```

On remplit les listes.

```
for(n in 1:length(liste_n))
{
  for(sigma in 1:length(liste_sigma))
  {
    i <- (n - 1) * length(liste_sigma) + sigma

    liste_biais_nnls_1_simu_sans_dependance[[i]] <-
      sqrt(liste_n[n]) * liste_resultats_df[[i]]["nnls_1", "biais"]
    liste_biais_nnls_2_simu_sans_dependance[[i]] <-
      sqrt(liste_n[n]) * liste_resultats_df[[i]]["nnls_2", "biais"]


    liste_biais_nnls_1_theorique_sans_dependance[[i]] <-
      sqrt(liste_n[n]) * liste_resultats_theoriques_df[[i]]["nnls_1", "biais"]
    liste_biais_nnls_2_theorique_sans_dependance[[i]] <-
      sqrt(liste_n[n]) * liste_resultats_theoriques_df[[i]]["nnls_2", "biais"]


    liste_variance_nnls_1_simu_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_df[[i]]["nnls_1", "variance"]
    liste_variance_nnls_2_simu_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_df[[i]]["nnls_2", "variance"]

    liste_variance_nnls_1_theorique_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_theoriques_df[[i]]["nnls_1", "variance"]
    liste_variance_nnls_2_theorique_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_theoriques_df[[i]]["nnls_2", "variance"]

    liste_MSE_nnls_1_simu_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_df[[i]]["nnls_1", "MSE"]
    liste_MSE_nnls_2_simu_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_df[[i]]["nnls_2", "MSE"]


    liste_MSE_nnls_1_theorique_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_theoriques_df[[i]]["nnls_1", "MSE"]
    liste_MSE_nnls_2_theorique_sans_dependance[[i]] <-
      liste_n[n] * liste_resultats_theoriques_df[[i]]["nnls_2", "MSE"]
  }
}
```

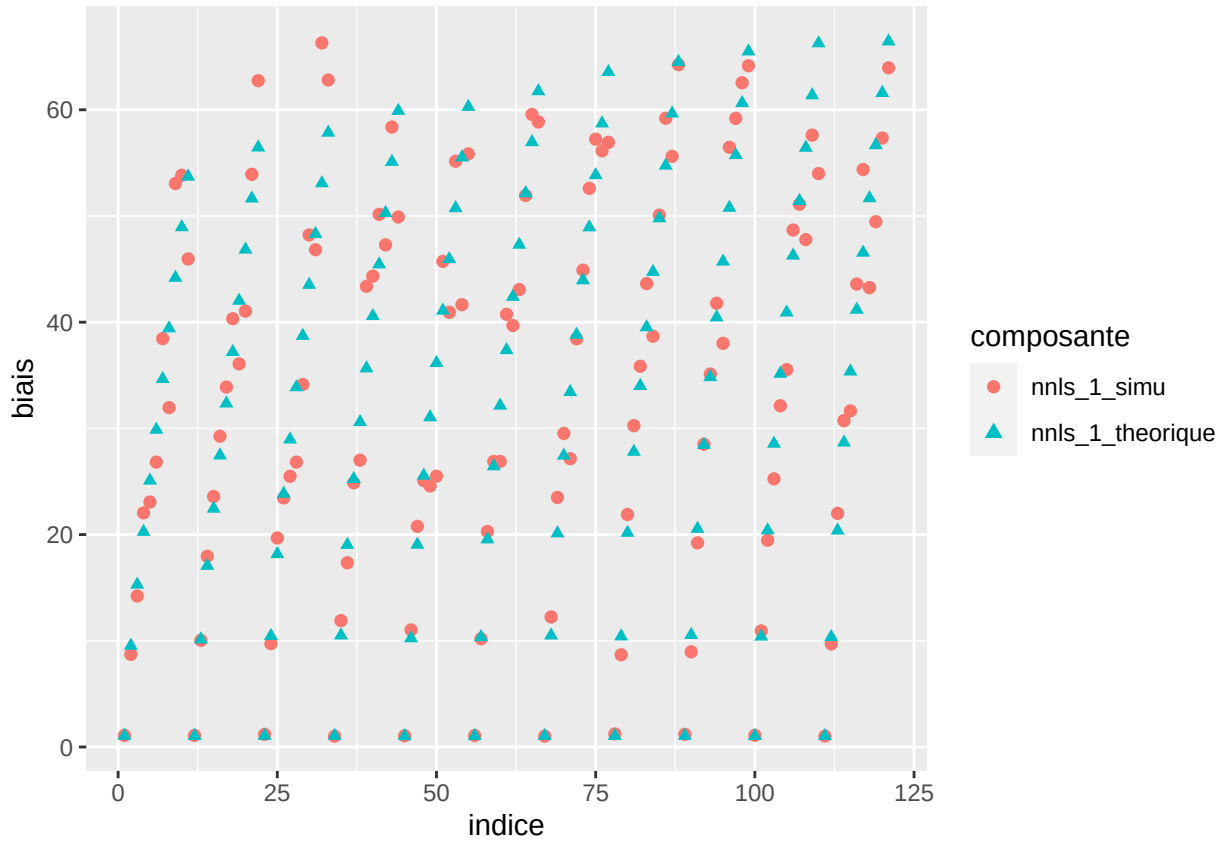
On construit deux dataframes qui nous serviront on construire les nuages de points.

```
comparaison_biais_theorique_simu_sans_dependance <-
  data.frame(
    indice = unlist(liste_indices),
    biais = c(unlist(liste_biais_nnls_1_simu_sans_dependance),
              unlist(liste_biais_nnls_1_theorique_sans_dependance),
              unlist(liste_biais_nnls_2_simu_sans_dependance),
              unlist(liste_biais_nnls_2_theorique_sans_dependance)),
    composante =
      c(rep("nnls_1_simu",
            length(liste_biais_nnls_1_simu_sans_dependance)),
        rep("nnls_1_theorique",
```

```
            length(liste_biais_nnls_1_theorique_sans_dependance)),
        rep("nnls_2_simu",
            length(liste_biais_nnls_2_simu_sans_dependance)),
        rep("nnls_2_theorique",
            length(liste_biais_nnls_2_theorique_sans_dependance)))
    )


comparaison_variance_theorique_simu_sans_dependance <-
  data.frame(
    indice = unlist(liste_indices),
    variance = c(unlist(liste_variance_nnls_1_simu_sans_dependance),
                 unlist(liste_variance_nnls_1_theorique_sans_dependance),
                 unlist(liste_variance_nnls_2_simu_sans_dependance),
                 unlist(liste_variance_nnls_2_theorique_sans_dependance)),
    composante =
      c(rep("nnls_1_simu",
            length(liste_variance_nnls_1_simu_sans_dependance)),
        rep("nnls_1_theorique",
            length(liste_variance_nnls_1_theorique_sans_dependance)),
        rep("nnls_2_simu",
            length(liste_variance_nnls_2_simu_sans_dependance)),
        rep("nnls_2_theorique",
            length(liste_variance_nnls_2_theorique_sans_dependance)))
    )


comparaison_MSE_theorique_simu_sans_dependance <-
  data.frame(
    indice = unlist(liste_indices),
    MSE = c(unlist(liste_MSE_nnls_1_simu_sans_dependance),
            unlist(liste_MSE_nnls_1_theorique_sans_dependance),
            unlist(liste_MSE_nnls_2_simu_sans_dependance),
            unlist(liste_MSE_nnls_2_theorique_sans_dependance)),
    composante =
      c(rep("nnls_1_simu",
            length(liste_MSE_nnls_1_simu_sans_dependance)),
        rep("nnls_1_theorique",
            length(liste_MSE_nnls_1_theorique_sans_dependance)),
        rep("nnls_2_simu",
            length(liste_MSE_nnls_2_simu_sans_dependance)),
        rep("nnls_2_theorique",
            length(liste_MSE_nnls_2_theorique_sans_dependance))))
```

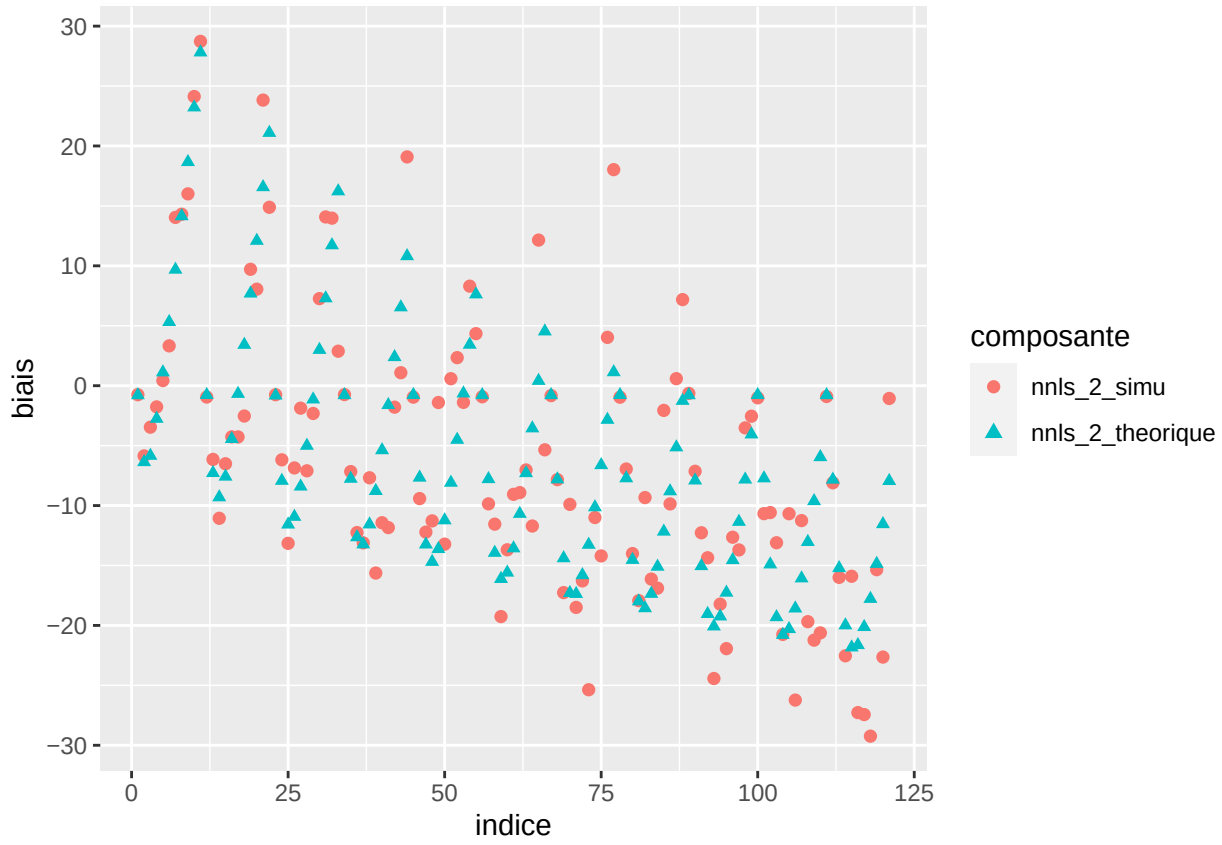
Comparaison des valeurs du biais pour la première composante.

```
ggplot(comparaison_biais_theorique_simu_sans_dependance,
               aes(x = indice, y = biais)) +
        geom_point(data = subset(comparaison_biais_theorique_simu_sans_dependance,
                                 composante == "nnls_1_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_biais_theorique_simu_sans_dependance,
                                 composante == "nnls_1_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



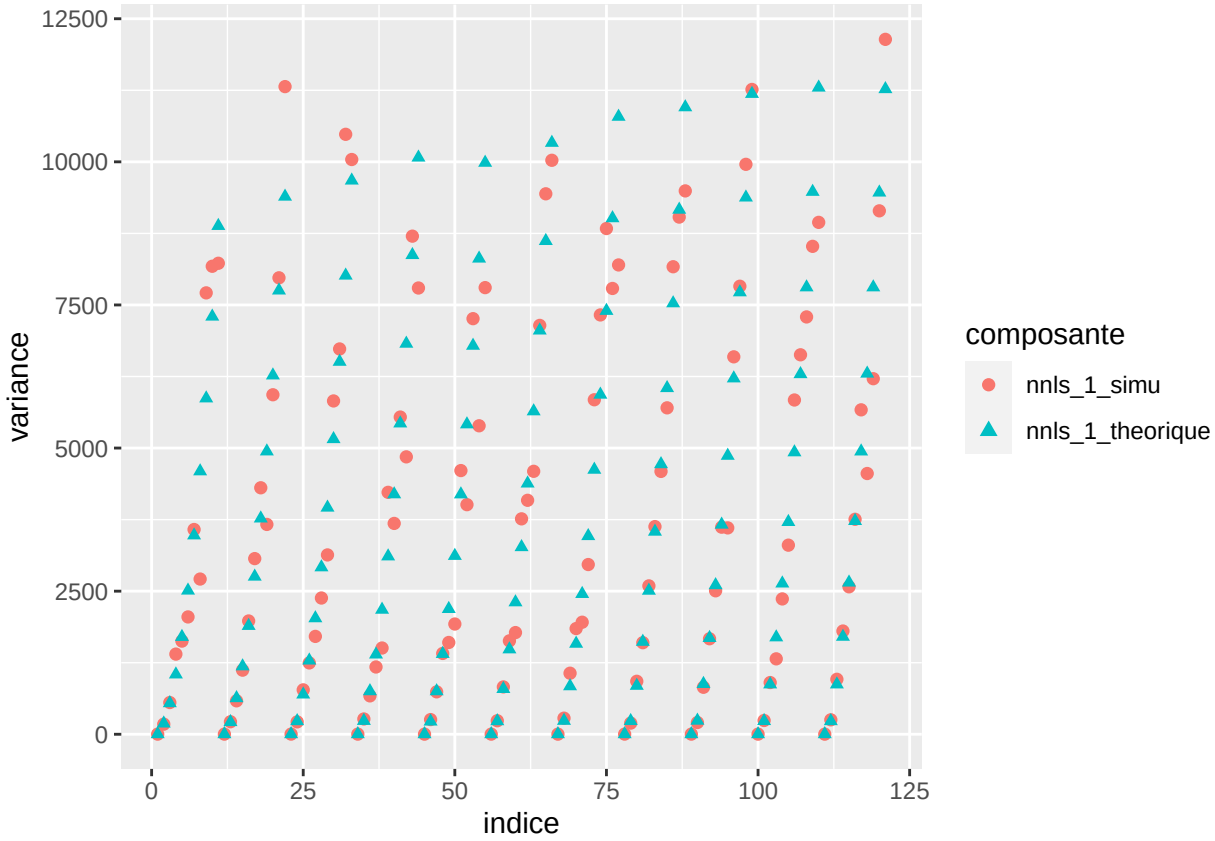
Comparaison des valeurs du biais pour la seconde composante.

```
ggplot(comparaison_biais_theorique_simu_sans_dependance,
               aes(x = indice, y = biais)) +
        geom_point(data = subset(comparaison_biais_theorique_simu_sans_dependance,
                                 composante == "nnls_2_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_biais_theorique_simu_sans_dependance,
                                 composante == "nnls_2_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



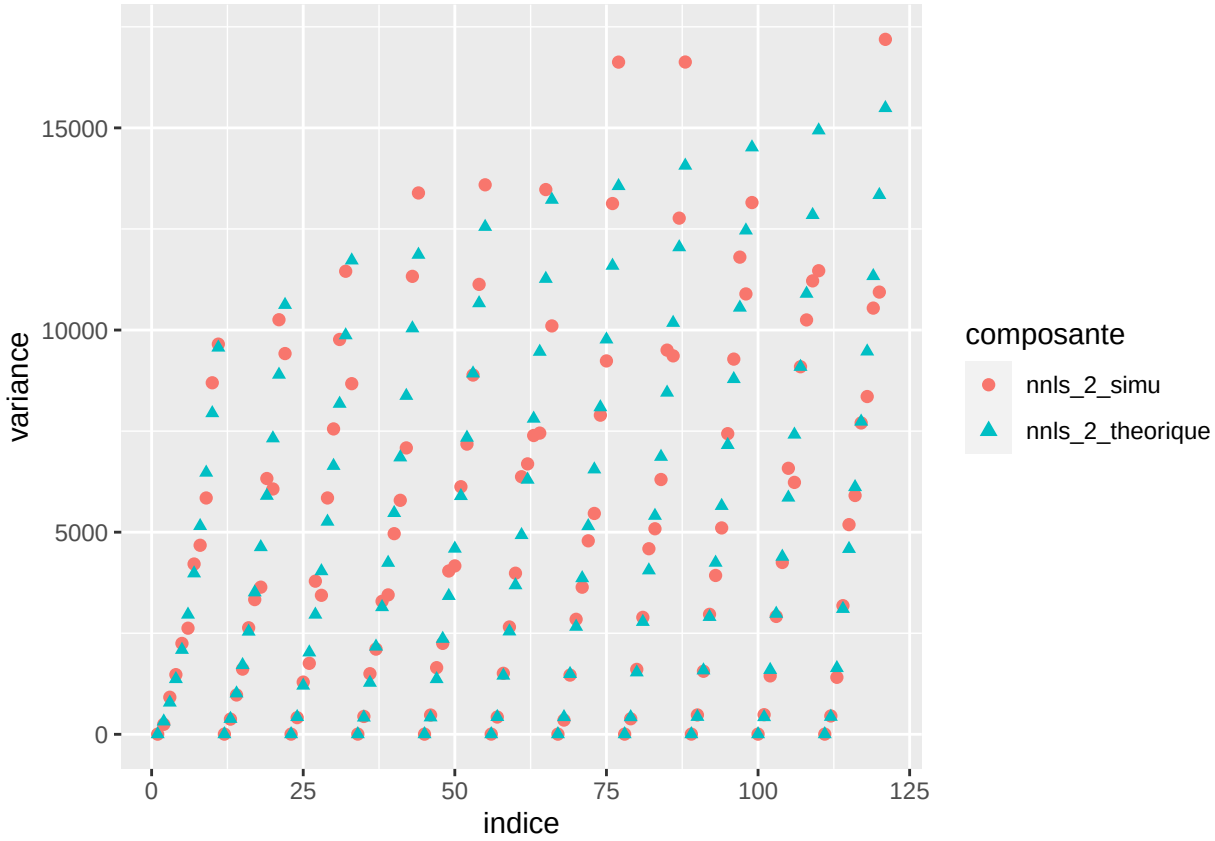
Comparaison des valeurs de la variance pour la première composante.

```
ggplot(comparaison_variance_theorique_simu_sans_dependance,
               aes(x = indice, y = variance)) +
        geom_point(data = subset(comparaison_variance_theorique_simu_sans_dependance,
                                 composante == "nnls_1_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_variance_theorique_simu_sans_dependance,
                                 composante == "nnls_1_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



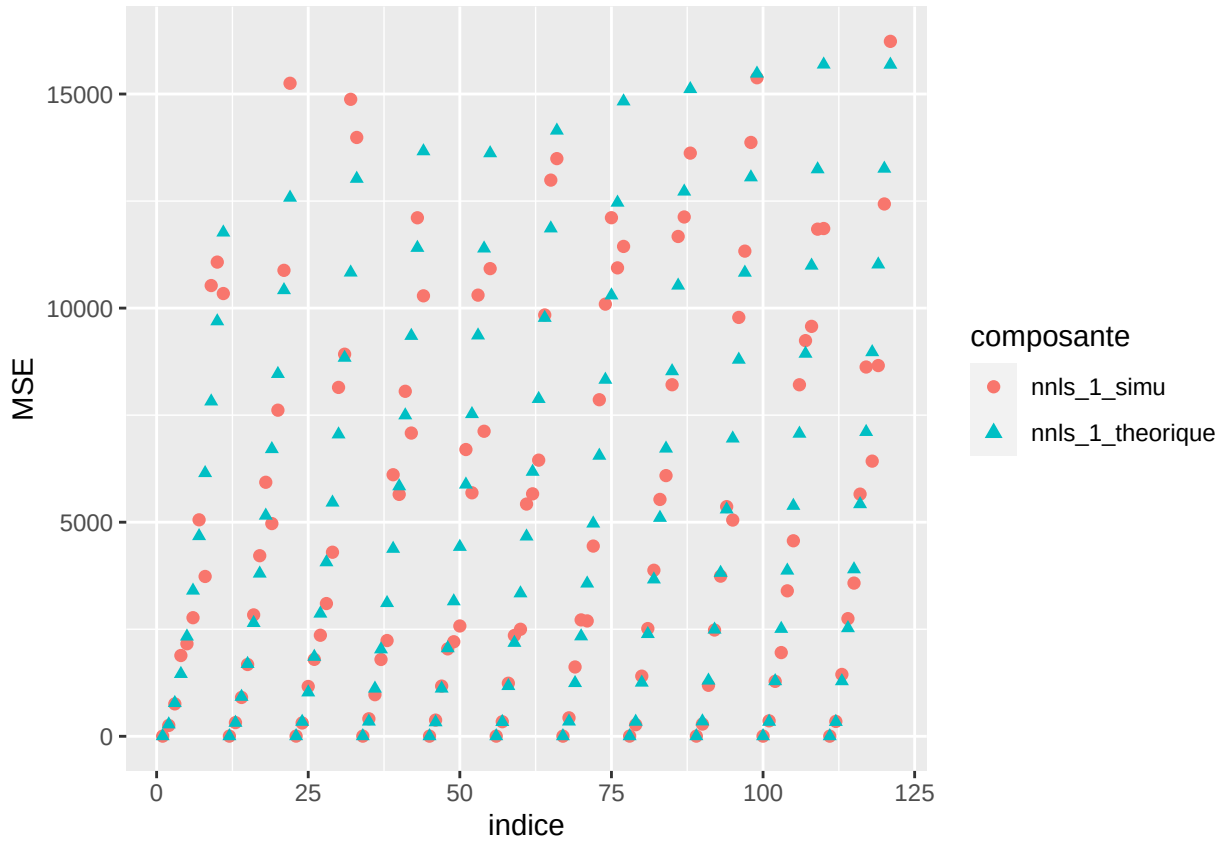
Comparaison des valeurs de la variance pour la deuxième composante.

```
ggplot(comparaison_variance_theorique_simu_sans_dependance,
               aes(x = indice, y = variance)) +
        geom_point(data = subset(comparaison_variance_theorique_simu_sans_dependance,
                                 composante == "nnls_2_simu"),
                   aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_variance_theorique_simu_sans_dependance,
                                 composante == "nnls_2_theorique"),
                   aes(colour = composante, shape = composante), size = 1.5)
```



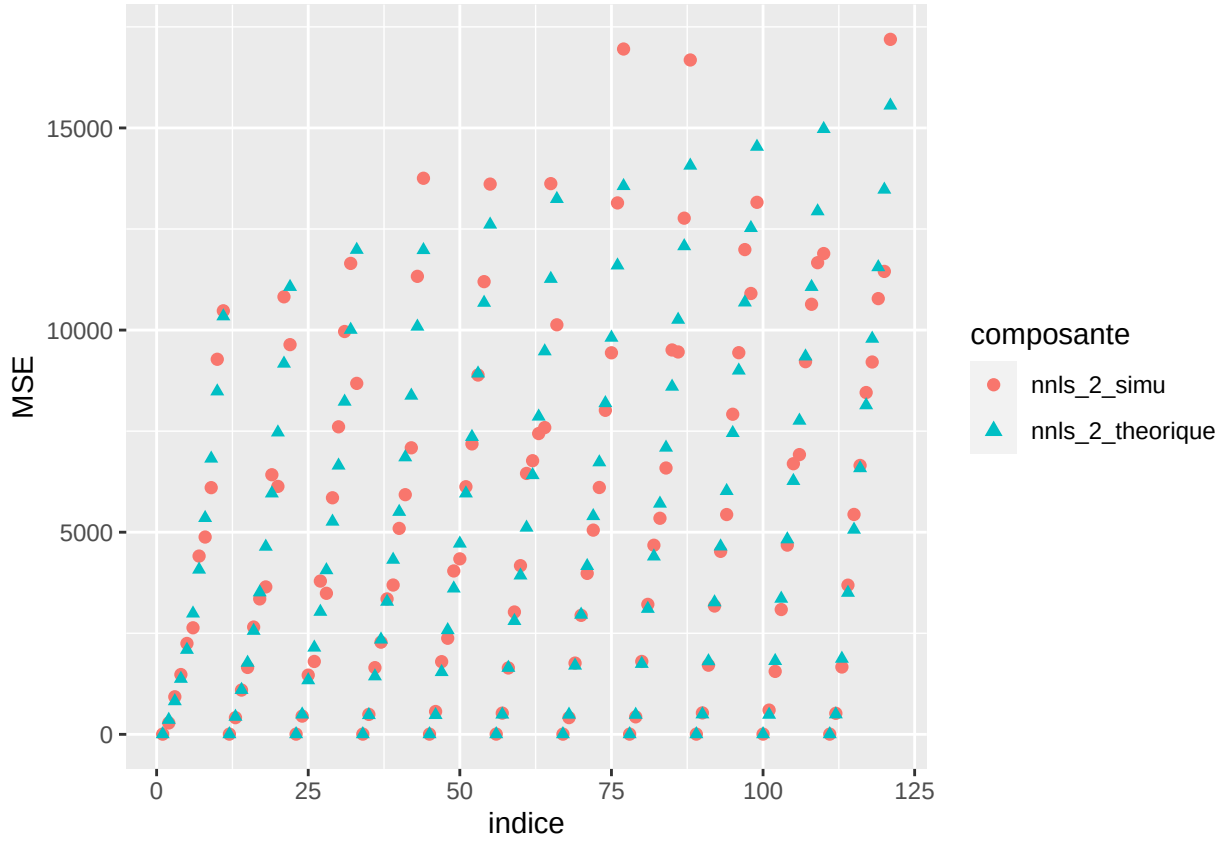
Comparaison des valeurs de l'erreur quadratique moyenne pour la première composante.

```
ggplot(comparaison_MSE_theorique_simu_sans_dependance,
               aes(x = indice, y = MSE)) +
        geom_point(data = subset(comparaison_MSE_theorique_simu_sans_dependance,
                                 composante == "nnls_1_simu"),
                           aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_MSE_theorique_simu_sans_dependance,
                                 composante == "nnls_1_theorique"),
                           aes(colour = composante, shape = composante), size = 1.5)
```



Comparaison des valeurs de l'erreur quadratique moyenne pour la seconde composante.

```
ggplot(comparaison_MSE_theorique_simu_sans_dependance,
               aes(x = indice, y = MSE)) +
        geom_point(data = subset(comparaison_MSE_theorique_simu_sans_dependance,
                                 composante == "nnls_2_simu"),
                   aes(colour = composante, shape = composante), size = 2) +
        geom_point(data = subset(comparaison_MSE_theorique_simu_sans_dependance,
                                 composante == "nnls_2_theorique"),
                   aes(colour = composante, shape = composante), size = 1.5)
```



On conclut que les valeurs simulées pour le biais, la variance et l'erreur quadratique moyenne sont généralement proches des résultats théoriques. C'est ce que l'on souhaitait.

Sauvegarde des résultats

```
save.image(file = "simulations_results.RData")
```

Chargement des résultats

```
rm(list = ls())
```

```
load(file = "simulations_results.RData")
```

```
rm(list = ls())
```