Régression non-paramétrique

Yassine ABDOU-Mohamed BERRAMOU

31/12/2019

Table des matières

1 Préambule **1**
1.1 Objectif de l'étude 1
1.2 Importation et description de la table 1

2 Estimateur par des modèles de regression polynmiale : **3**

3 Estimateur par des B-splines **8**

4 Estimateur à noyau de la régression **10**

5 Estimation par splines quadratiques pénalisées **17**

6 Conclusion **23**

7 Travail supplémentaire **23**

1 Préambule

1.1 Objectif de l'étude

Ce travail est une application des méthodes non-paramétriques faites dans le cours.

Nous disposons d'une table de données nommée ***fossil*** et notre objectif sera d'estimer la variable d'intérêt **strontium.ratio** en fonction de l'**âge**. Pour cela, nous allons commencer avec une estimation par régression polynômiale, ensuite une estimation non-paramétrique par les fonctions **Splines**, l'estimation à noyau et finalement, l'estimation par splines quadratiques pénalisées.

1.2 Importation et description de la table

Afin d'entamer notre analyse, nous allons tout d'abord importer notre table de données.

La table de données est en format *(.txt)*, pour cela nous allons utiliser la fonction **read.table()** pour l'importer en précisant (**header = T**) pour que les noms des variables soient prises en compte.

```
fossil <-
  read.table("data/fossil.txt",
    header = T,
    encoding = "UTF-8"
  ) %>%
  mutate(age_norm = (age-min(age))/(max(age)-min(age)))

fossil %>%
  head(10) %>%
  kable(
    format = "latex",
    booktabs = T,
    caption = "Les dix premières lignes de notre table de données",
```

```
    col.names = c("Age (M.a)", "Rapports_strontium", "Age normalisé")
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

TABLE 1 – Les dix premières lignes de notre table de données

Age (M.a)	Rapports_strontium	Age normalisé
91.78525	0.707343	0.0000000
92.39579	0.707359	0.0195591
93.97061	0.707410	0.0700104
95.57577	0.707438	0.1214335
95.60286	0.707463	0.1223015
112.33691	0.707320	0.6583956
112.43502	0.707316	0.6615388
112.52251	0.707274	0.6643416
112.60125	0.707220	0.6668642
108.66200	0.707378	0.5406658

Notre table de données contient 106 observations sur des coquilles fossiles et deux variables, la première est **Age** en millions d'années et la deuxième **strontium.ratio** qui représente les rapports des isotopes du strontium.

```
Resum_fossil <- fossil %>%
  dplyr::select(-3) %>%
  summary() %>%
  as.data.frame() %>%
  separate(Freq, c("description", "freq"), ":") %>%
  pivot_wider(names_from = description, values_from = freq) %>%
  select(-Var1) %>%
  rename(variables = Var2)

Resum_fossil %>%
  kable(
    format = "latex",
    booktabs = T,
    caption = "Résumé des variables",
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

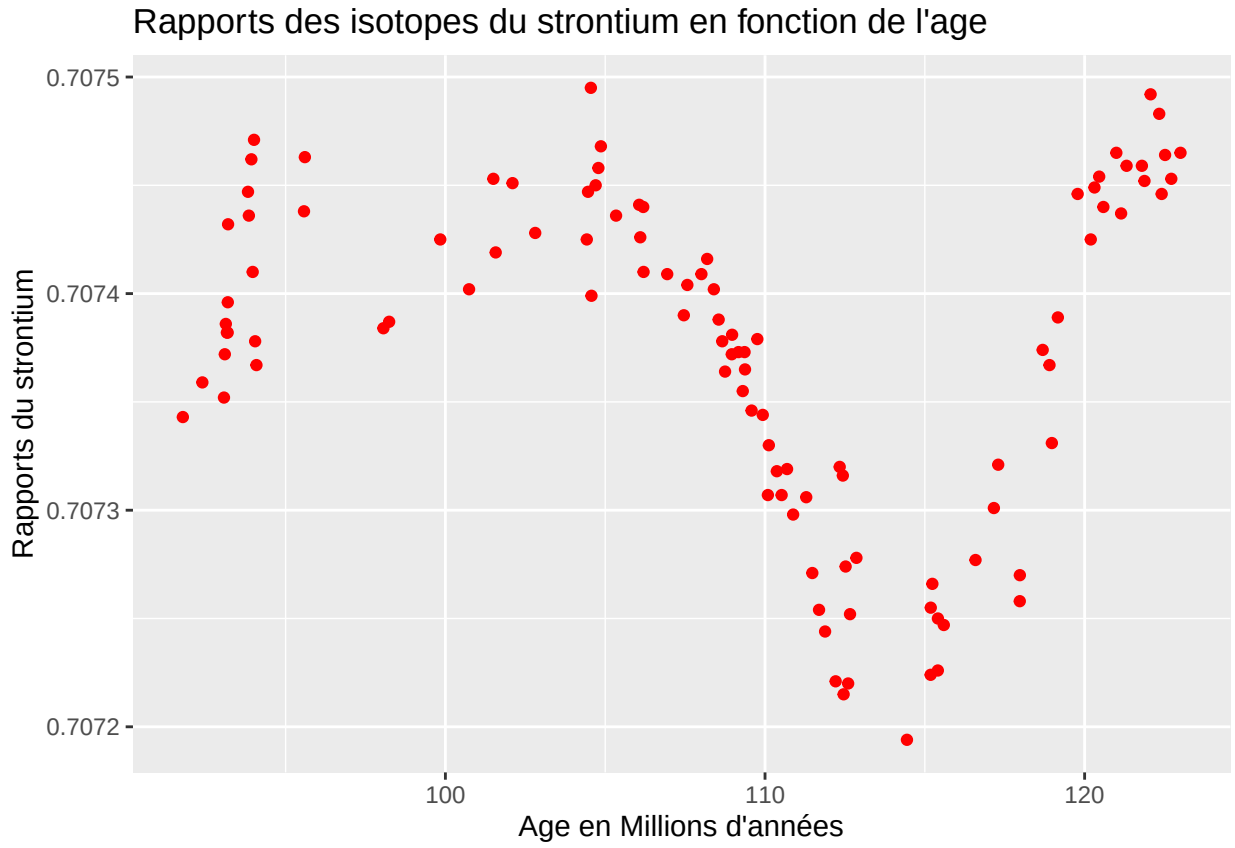
TABLE 2 – Résumé des variables

variables	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
age	91.79	104.43	109.48	108.78	115.41	123.00
strontium.ratio	0.7072	0.7073	0.7074	0.7074	0.7074	0.7075

Nous constatons d'après la table ci-dessus que la variable **age** varie entre *91.79* et *123* millions d'année et la variable **strontium.ratio** varie entre *0.7072* et *0.7075*.

2 Estimateur par des modèles de regression polynmiale :

```
# Représentation graphique
gg <- fossil %>%
  ggplot() +
  geom_point(mapping = aes(
    x = age,
    y = strontium.ratio
  ),col = "red") +
  labs(
    x = "Age en Millions d'années",
    y = "Rapports du strontium",
    title = "Rapports des isotopes du strontium en fonction de l'age"
  ) +
  theme_grey()

gg
```



Nous constatons d'après le graphe ci-dessus que la relation entre nos deux variables **strontium.ratio~age** ne suit pas une tendance linéaire.

Pour cela, nous allons utiliser des méthodes non-linéaires, nous allons commencer par la régression polynômiale *(degré 4)*

```
Reg_poly4 <- lm(strontium.ratio ~ poly(age, 4), data = fossil)

resum_coef <- Reg_poly4 %>%
```

```
  tidy()

Resum_poly4 <- Reg_poly4 %>%
  summary()

resum_coefR <- data.frame(
  R_squared = Resum_poly4$r.squared,
  adj_R_squared = Resum_poly4$adj.r.squared,
  Sigma2 = Resum_poly4$sigma
)

merge(resum_coef, resum_coefR) %>%
  kable(
    format = "latex",
    booktabs = T,
    caption = "Résumé du modèle de la régression polynômiale",
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

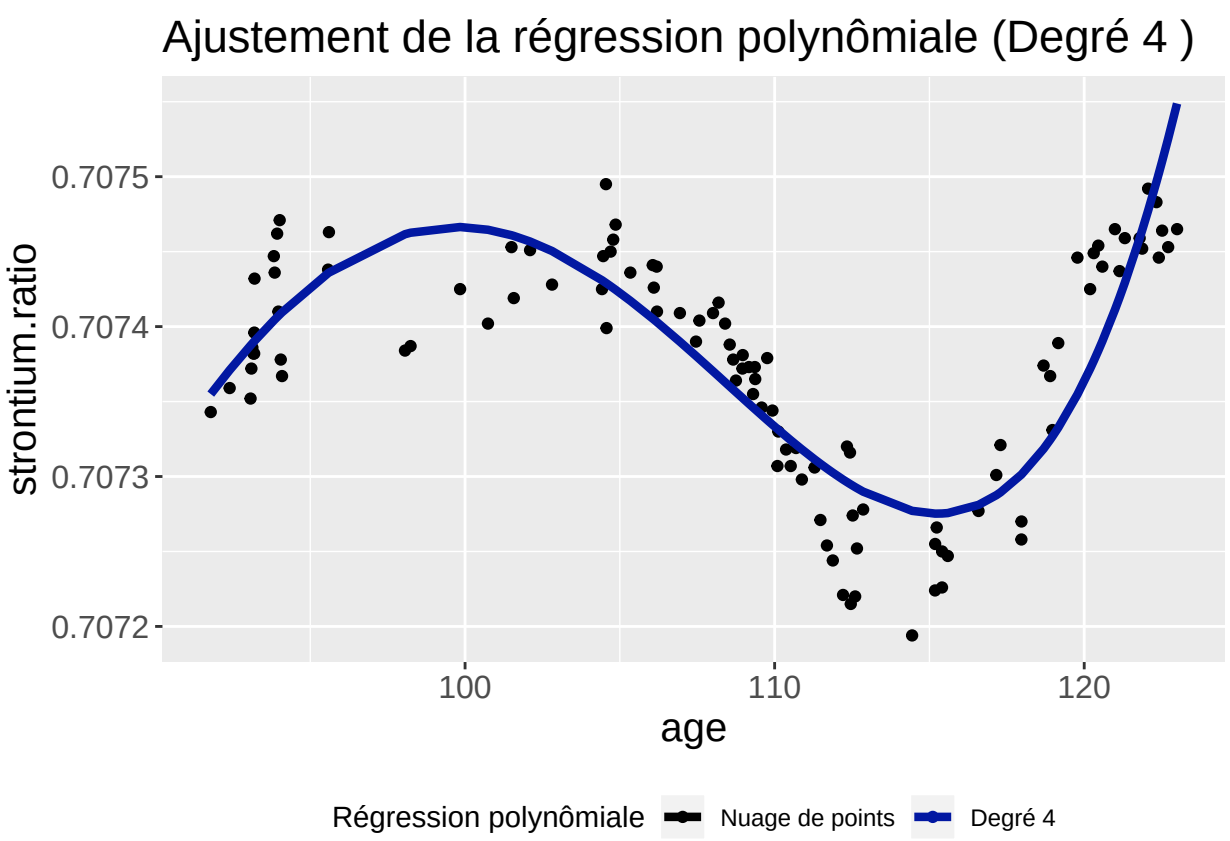
TABLE 3 – Résumé du modèle de la régression polynômiale

term	estimate	std.error	statistic	p.value	R_squared	adj_R_squared	Sigma2
(Intercept)	0.7073741	3.90e-06	180976.191900	0.0000000	0.7306704	0.7200039	4.02e-05
poly(age, 4)1	-0.0001334	4.02e-05	-3.315808	0.0012703	0.7306704	0.7200039	4.02e-05
poly(age, 4)2	0.0002639	4.02e-05	6.558368	0.0000000	0.7306704	0.7200039	4.02e-05
poly(age, 4)3	0.0005872	4.02e-05	14.591703	0.0000000	0.7306704	0.7200039	4.02e-05
poly(age, 4)4	0.0001071	4.02e-05	2.660940	0.0090657	0.7306704	0.7200039	4.02e-05

Nous constatons que notre **R-carré** est égale à *0.7306704* et que toutes les variables sont très significatives avec un résidu de *4.02e-05*.

```
g_poly4 <- fossil %>%
  ggplot() +
  geom_point(mapping = aes(y = strontium.ratio, x = age, colour = "black")) +
  geom_line(
    mapping = aes(
      x = age,
      y = predict(Reg_poly4),
      colour = "#0218a2",
    ),
    size = 1.5
  ) +
  scale_color_identity(
    name = "Régression polynômiale",
    breaks = c("black", "#0218a2"),
    labels = c(
      "Nuage de points",
      "Degré 4"
    ),
    guide = "legend"
  ) +
  labs(title = "Ajustement de la régression polynômiale (Degré 4 )") +
```

```
  theme(legend.position = "bottom", legend.box = "horizontal") +
  theme_ggplot()

g_poly4
```



Comparativement à la régression linéaire simple, la régression polynomiale nous donne une courbe plus ajustée à notre jeu de données.

Nous allons dans la suite de cette partie calculer la précision prédictive de la régression polynômiale pour différents degrés. Pour cela nous allons dans un premier temps séparer notre jeu de données en **base d'apprentissage** et en **base de test**, ensuite nous allons adopter la **validation croisée** comme une méthode de séparation.

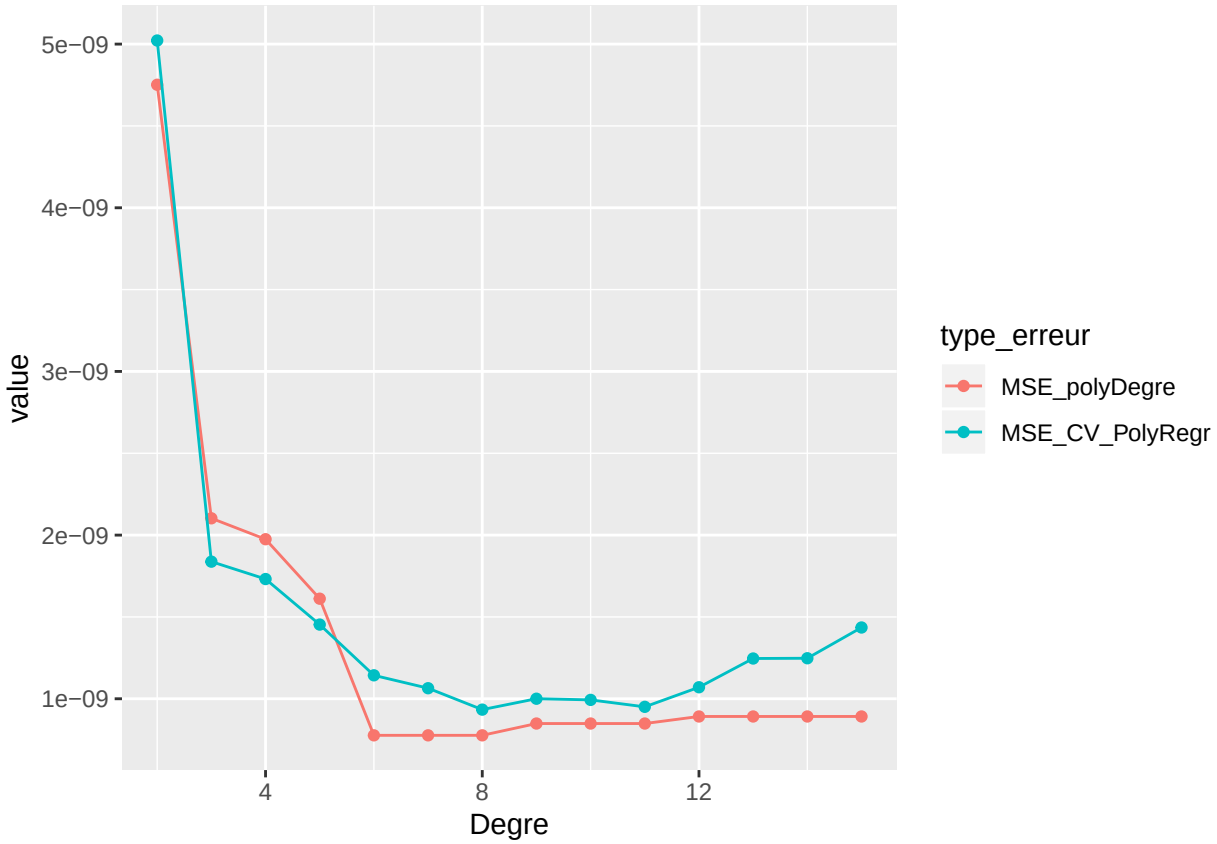
```
set.seed(0207)
N_ligne <- nrow(fossil)
vect_random <- sample(1:N_ligne, 0.7 * N_ligne)

Data_train <- fossil %>%
  slice(vect_random)

Data_test <- fossil %>%
  slice(-vect_random)

age_test <- Data_test$age
age_train <- Data_train$age

strontium.ratio_test <- Data_test$strontium.ratio
```

```r
strontium.ratio_train <- Data_train$strontium.ratio
```

```r
Reg_poly_n <-
  lapply(2:15, function(x) {
    lm(strontium.ratio ~ poly(age, x, raw = TRUE), data = Data_train)
  })

MSE_polyDegre <-
  lapply(1:length(Reg_poly_n), function(x) {
    Reg_poly_n[[x]] %>% err_prediction(Data_test)
  }) %>%
  unlist()
```

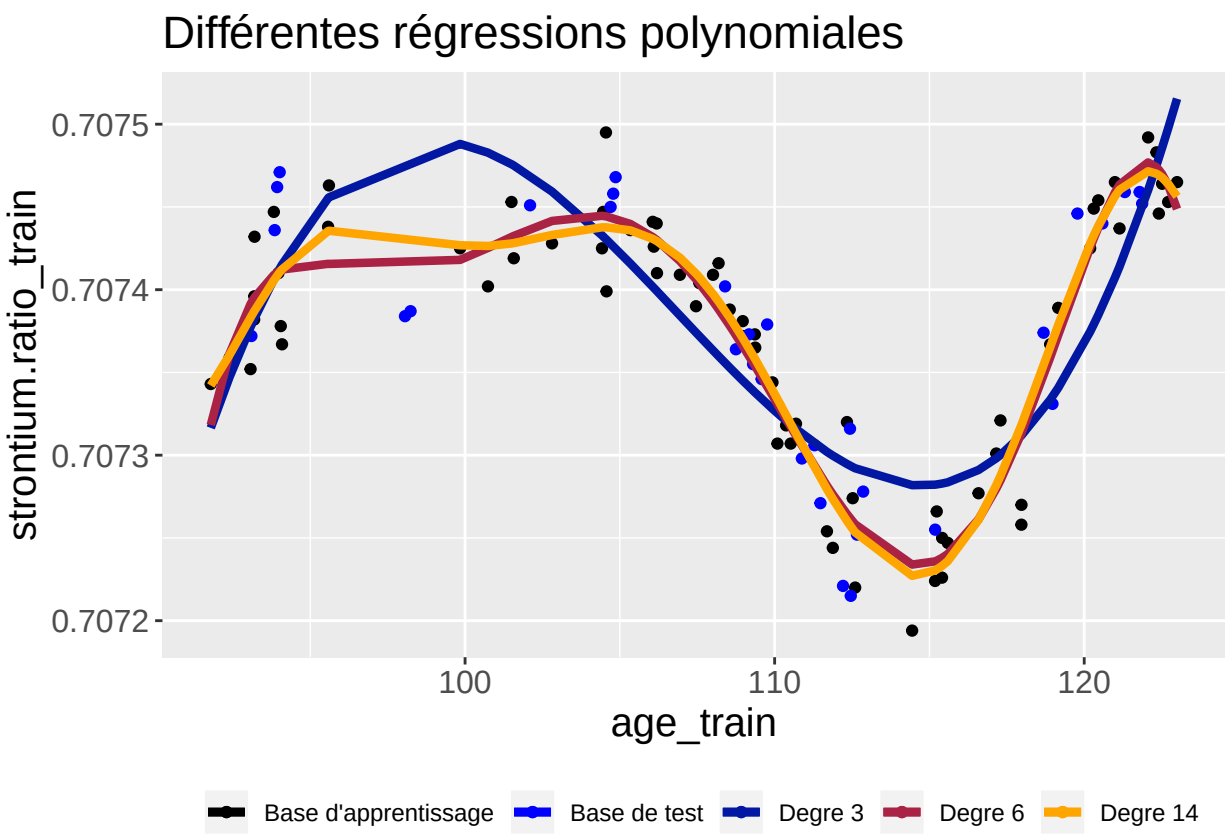
Pour la validation croisée nous allons utiliser la fonction **cv.polywog** prédéfinie dans R, cette dernière nous donne la possibilité de choisir le nombre de **fold** souhaité.

```r
# Validation croisée
set.seed(0207)
cv1 <- cv.polywog(strontium.ratio ~ scale(age),
                  data = fossil,
                  degrees.cv = 2:15,
                  nfolds = 7,
                  thresh = 1e-4)
err_1 <- cv1$results
MSE_CV_PolyRegr <- err_1[,3]
```



Nous constatons que l'erreur minimale obtenue par la régression polynômiale par validation croisée est plus grande que celle obtenue par la division classique du jeu de données.

On conclut que la régression polynômiale de degré **6** donne une erreur minimale de **7.766777e-10**.



Le graphe ci-dessus représente trois courbes obtenues par régression polynômiale de degré *3, 6 et 14*.

Nous constatons que la courbe de degré 3 est relativement proche aux nuages de points, cependant nous remarquons que la courbe de degré 6 colle bien aux valeurs inférieurs à **105** comparativement à la courbe de degré 14, et elles sont toutes les deux superposées pour toutes les valeurs supérieures à **105**.

TABLE 4 – Résumé du modèle de la régression polynômiale (Degré 6)

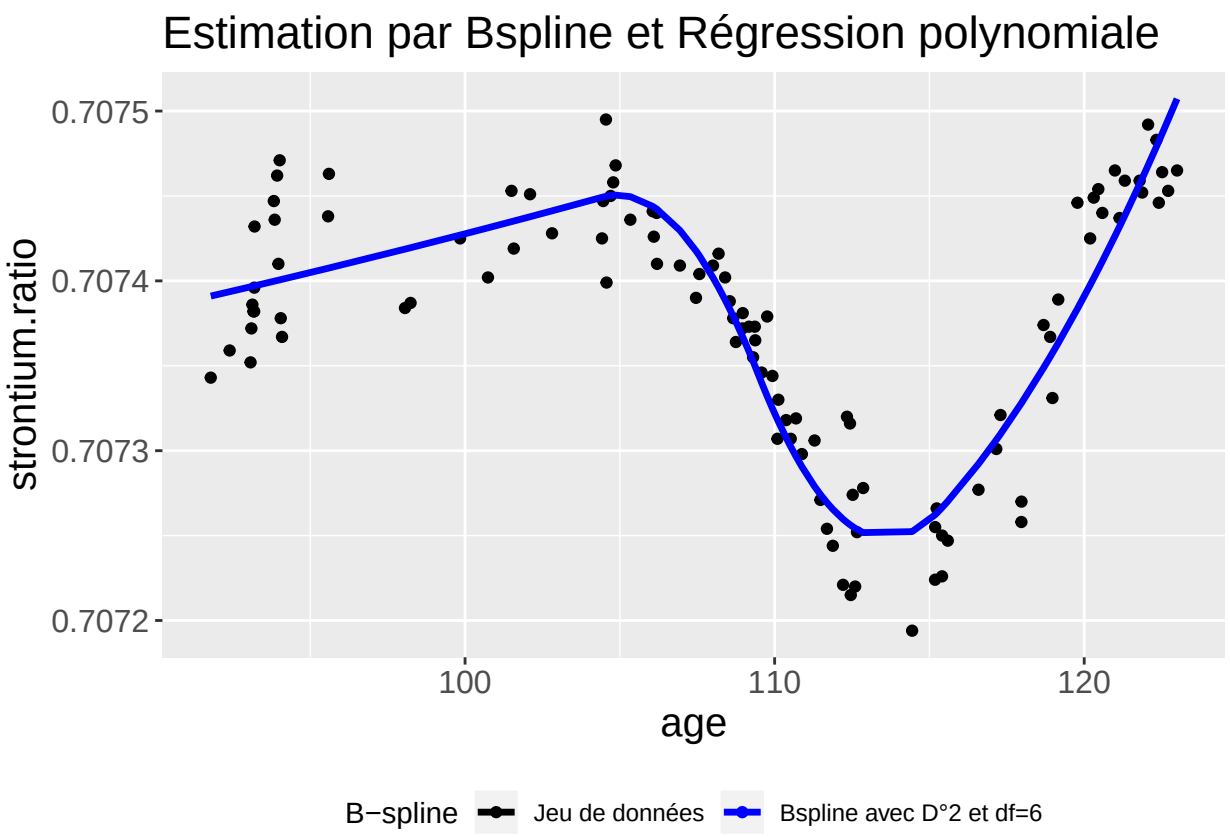
term	estimate	std.error	statistic	p.value	R2	aR2
(Intercept)	-124.2648481	14.9800365	-8.295364	0	0.9011614	0.8923102
poly(age, x, raw = TRUE)1	7.1067538	0.8441932	8.418397	0	0.9011614	0.8923102
poly(age, x, raw = TRUE)2	-0.1680332	0.0197857	-8.492678	0	0.9011614	0.8923102
poly(age, x, raw = TRUE)3	0.0021143	0.0002469	8.564963	0	0.9011614	0.8923102
poly(age, x, raw = TRUE)4	-0.0000149	0.0000017	-8.634780	0	0.9011614	0.8923102
poly(age, x, raw = TRUE)5	0.0000001	0.0000000	8.701658	0	0.9011614	0.8923102
poly(age, x, raw = TRUE)6	0.0000000	0.0000000	-8.765149	0	0.9011614	0.8923102

Nous constatons que les valeurs de R^2 et R^2 ajustés sont plus grandes que celles obtenues par le polynôme de degré 4. Cela nous prouve que ce modèle est bel et bien meilleur que celui choisi avant.

3 Estimateur par des B-splines

Les B-splines sont des fonctions définies par morceaux par des polynômes, elles sont plus flexibles car elles proposent de rajouter des noeuds et aussi elles adaptent les coefficients entre chaque deux noeuds pour donner une approche plus précise à notre nuage de points.

```
attach(fossil)
Bbase <- lm(strontium.ratio~bs(age,df=6,degree=2, intercept=TRUE)-1)
detach(fossil)
```



Le graphe ci-dessus représente une estimation du nuage de point de notre jeu de données par Bsplines avec un degré 2, et 6 degrés de liberté.

```
err_prediction(Bbase, fossil)
```

```
## [1] 8.900049e-10
```

L'erreur obtenue est égale à **8.900049e-10**.

Nous constatons que l'estimation par Bspline estime mieux notre jeu de données, car sa courbe colle bien aux nuages de points comparativement à la régression polynomiale.

Nous allons maintenant appliquer ce modèle pour prédire les valeurs de notre variable d'intérêt **strontium.ratio**. Pour cela nous allons commencer par le choix du degré de liberté qui donne l'erreur minimale.

```
attach(Data_train)
Reg_Bspline <-
  lapply(1:15, function(x) {
    lapply(1:15, function(y) {
      lm(strontium.ratio ~ bs(
```

```
        age,
        df = x + y + 1,
        degree = x,
        intercept = TRUE
      ) - 1)
    })
  })

MSE_Bspline <-
  lapply(1:length(Reg_Bspline), function(x) {
    lapply(1:length(Reg_Bspline), function(y) {
      Reg_Bspline[[x]][[y]] %>% err_prediction(Data_test)
    })
  }) %>%
  unlist()

detach(Data_train)
```

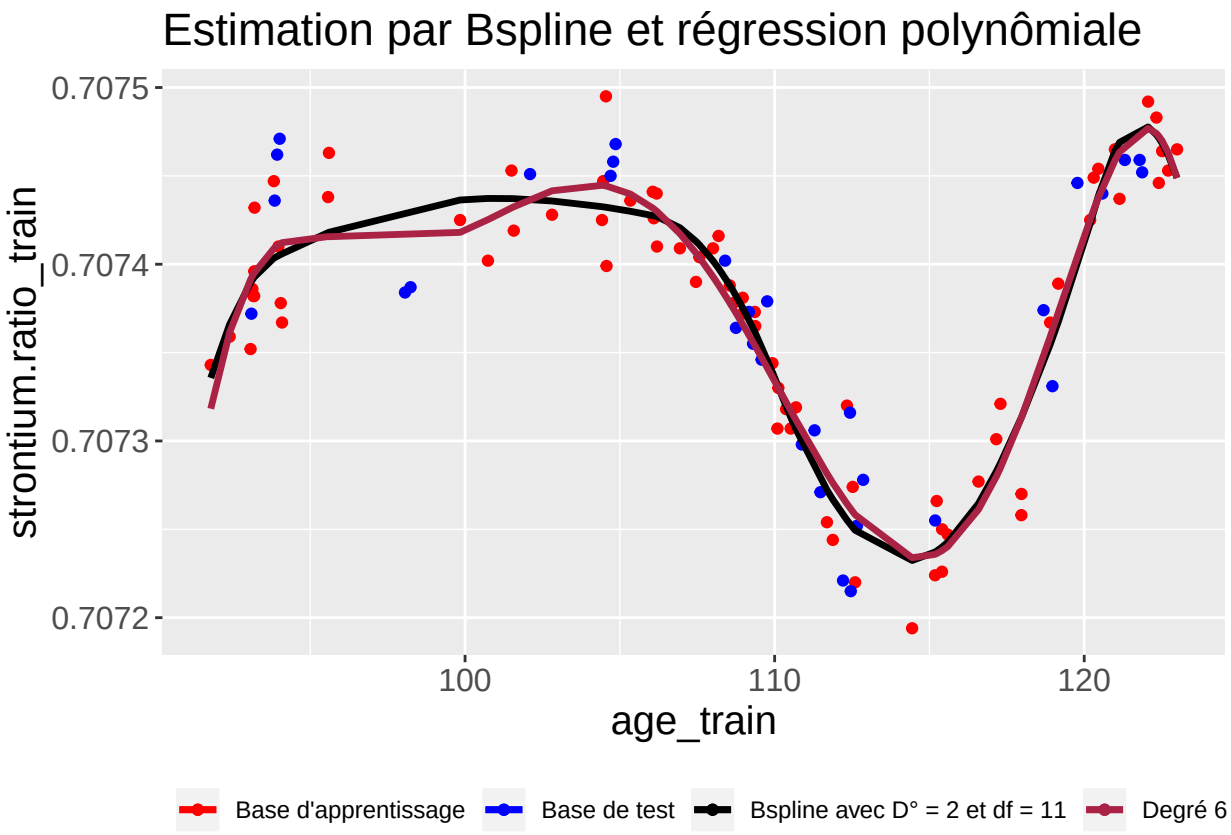
```
options( "digits"=14, "scipen"=0)

MSE_data_min <- data_erreurs_Bspline %>%
  filter(MSE_Bspline == min(MSE_Bspline))

MSE_data_min %>%
  kable(
    format = "latex",
    booktabs = T,
    caption = "Estimation par Bsplines",
    col.names = c("Degré de liberté","Degré", "Erreur obtenue")
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

TABLE 5 – Estimation par Bsplines

Degré de liberté	Degré	Erreur obtenue
11	2	7.8914e-10

L'MSE minimale est égale à *7.891356e-10*, elle est obtenue avec un degré 2 et 11 degrés de liberté.

```
Bbase <-
  lm(strontium.ratio_train ~ bs(
    age_train,
    df = 11,
    degree = 2,
    intercept = TRUE
  ) - 1)
```



4 Estimateur à noyau de la régression

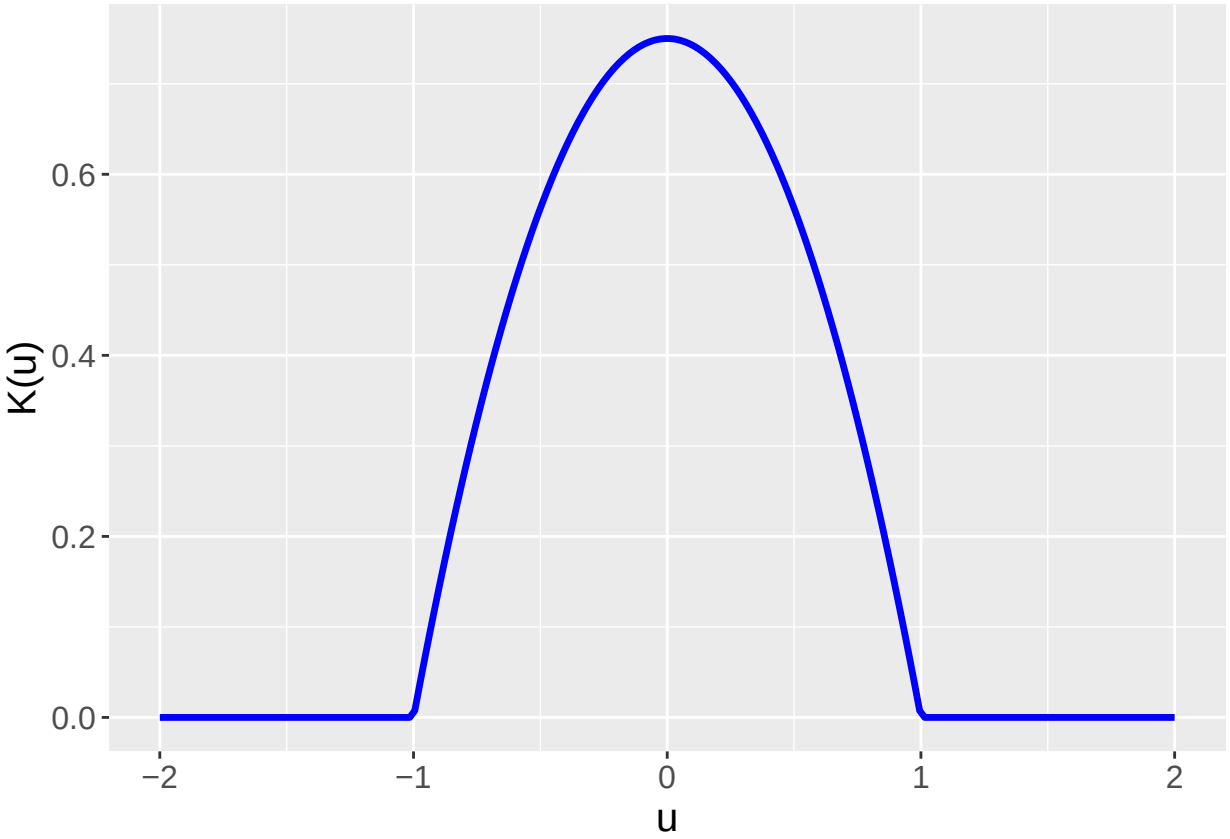
Afin de trouver une meilleure approche de la relation entre la variable à expliquer **strontium.ratio** et la variable explicative **age**, nous allons construire un estimateur à noyau. Pour cela nous allons utiliser le noyau d'**Epanechnikov**, qui est défini par

$$K(u) = \frac{3}{4}(1-u^2)1_{|u|\leq 1}$$

.

Nous allons tout d'abord coder la fonction **K(u)**, puis représenter la forme de ce noyau.

```
u <- seq(-2, 2, length.out = 200)

ggplot() +
  geom_line(mapping = aes(x = u, y = K(u)),
            col = "blue",
            size = 1.2) +
  theme_ggplot()
```



Nous allons maintenant construire une fonction **hat__f** qui va permettre d'estimer la fonction de régression de **strontium.ratio** au point **age**.

```
# Epanechnikov quadratic kernel
#' Title
#'
#' @param x_0
#' @param x
#' @param h
#'
#' @return
#' @export
#'
#' @examples
K_lambda <- function(x_0, x, h){
  K(abs(x - x_0) / h)
}

# average
#' Title
#'
#' @param x_0
#' @param x
#' @param h
#' @param y
#'
#' @return
```

```
#' @export
#'
#' @examples
hat_f <- function(x_0, x = x, h, y = y){
  return(sum(K_lambda(x_0, x, h) * y) / sum(K_lambda(x_0, x, h)))
}
```

La fonction ci-dessus nous calcule la prédiction de notre variable d'intérêt **y** en utilisant la fonction codée précèdement *hat_f*.

Dans notre cas, nous allons prendre **n = nrow(fossil)**, **x = age**, **y = strontium.ratio** et **h = 7** .

```
#' Prédiction de y
#'
#' @param n
#' @param x_0
#' @param x
#' @param h
#' @param y
#'
#' @return
#' @export
#'
#' @examples
pred_y <-
  function(n, x_0, x, h, y) {
    lapply(1:n, function(t) {
      hat_f(x_0[t], x, h, y)
    }) %>%
      unlist()
  }

attach(fossil)
ratio.pred <- pred_y(nrow(fossil),
                     age,
                     age,
                     7,
                     strontium.ratio)
detach(fossil)
```

```
attach(fossil)

MSE_ker <-((strontium.ratio - ratio.pred)^2) %>%
  mean()

MSE_ker
```

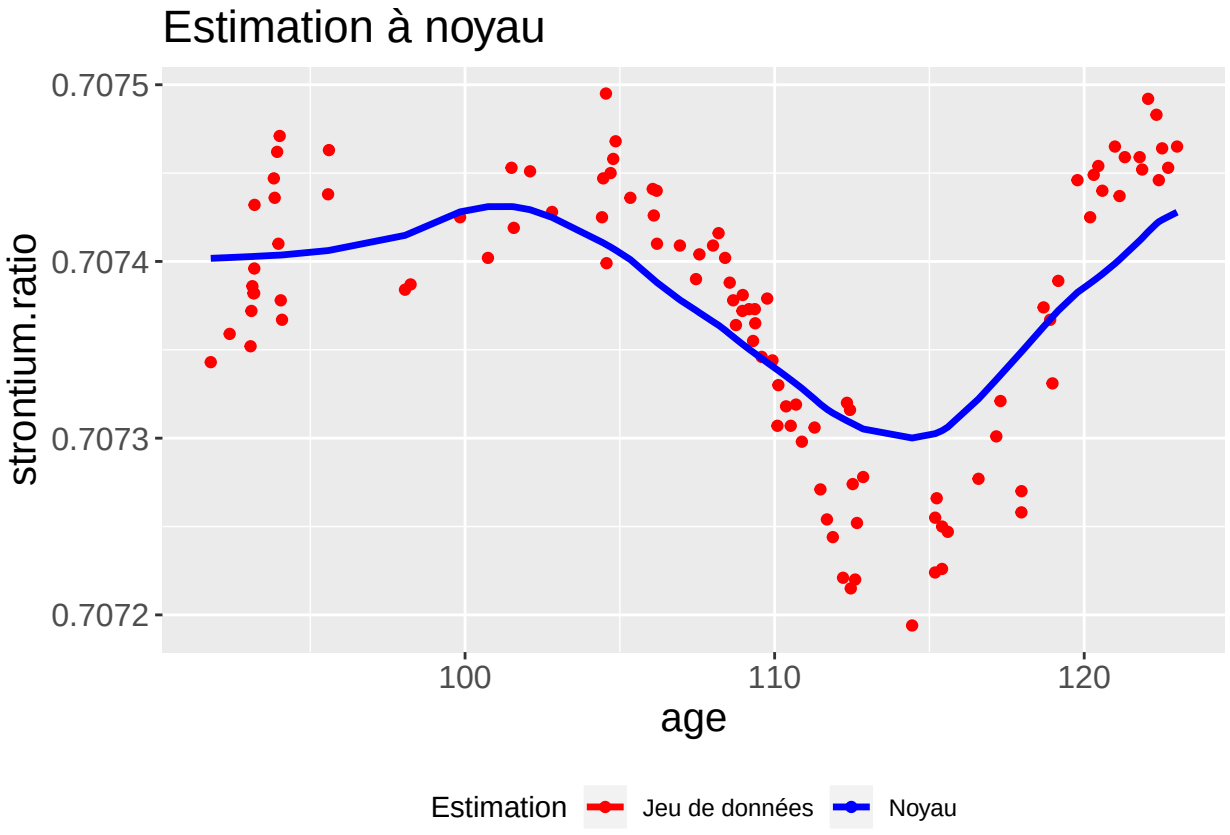
```
## [1] 1.9791845784736e-09
```

```
detach(fossil)
```

L'erreur quadratique moyenne obtenue est de **1.979185e-09**

```
fossil %>%
  ggplot() +
  geom_point(mapping = aes(y = strontium.ratio,
                           x = age,
```

```
                          colour = "red")) +
  geom_line(mapping = aes(y = ratio.pred,
                          x = age,
                          colour = "blue"),
            size = 1.2) +
  scale_color_identity(
    name = "Estimation",
    breaks = c("red",
               "blue"),
    labels = c(
      "Jeu de données",
      "Noyau"
    ),
    guide = "legend"
  ) +
  labs(title = "Estimation à noyau ") +
  theme(legend.position = "bottom",
        legend.box = "horizontal") +
  theme_ggplot()
```



Dans le but d'améliorer notre prédiction, nous allons utiliser la validation croisée pour diviser notre jeu de données. Pour cela nous allons la recoder à la main :

```
cv_noyau <- function(n, x, x_0, y_fossil, y_rat.pre, h) {
  w <-
    lapply(1:n, function(z) {
```

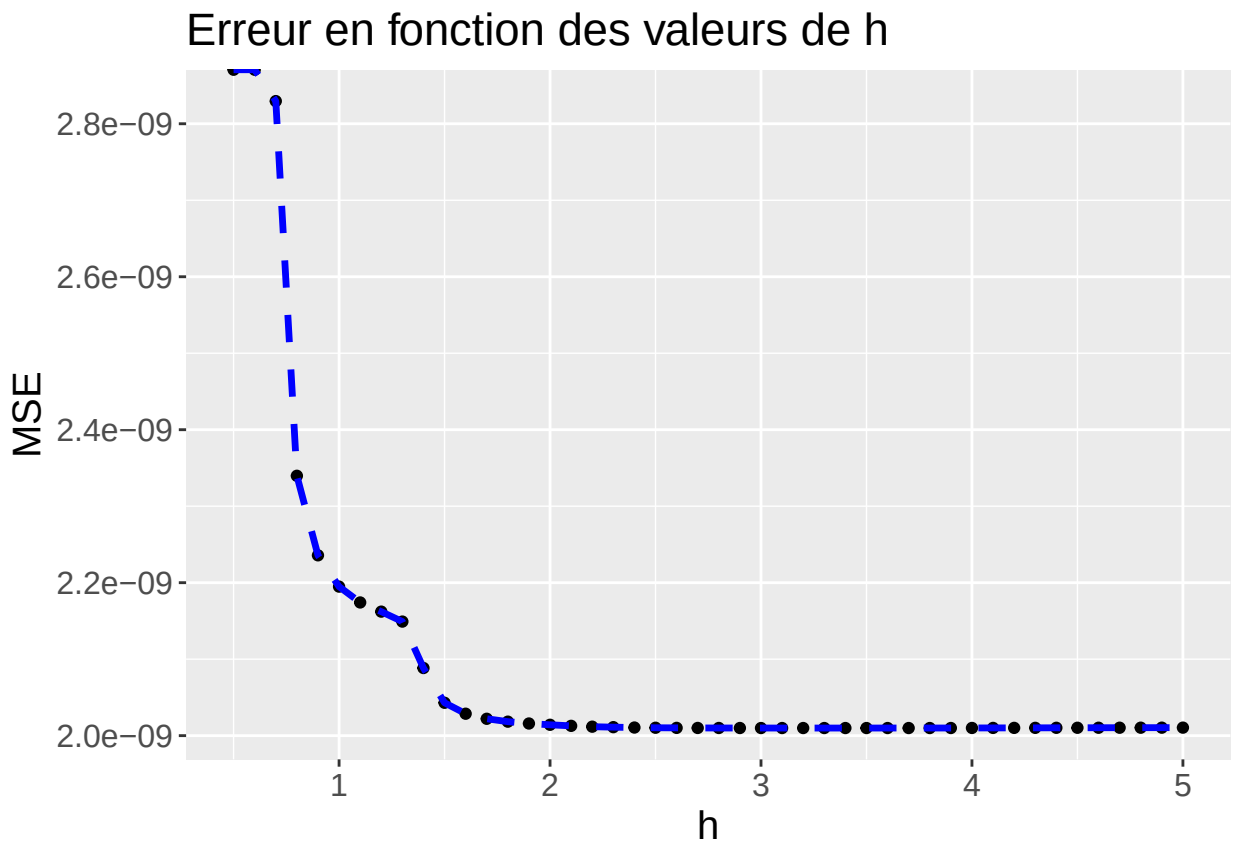
```
      K((x - x_0[z]) / h) / sum(K((x - x_0[z]) / h))
    }) %>%
    unlist()

  e <-
    lapply(1:n, function(v) {
      ((y_fossil[v] - y_rat.pre[v]) / (1 - w[v])) ^ 2
    }) %>%
    unlist()

  CV_erreur <- mean(e)

  return(CV_erreur)
}
```

```
attach(fossil)
error_CV <- cv_noyau(nrow(fossil), age, age, strontium.ratio, ratio.pred, 7)
error_CV
```

```
## [1] 2.0105183050202e-09
```

```
h <- seq(from=0.5,to=5, by=0.1)
cv_noyau_h <-
  lapply(h, function(j) {
    cv_noyau(nrow(fossil), age, age, strontium.ratio, ratio.pred, j)
  }) %>%
  unlist()


Data_erreur_h2 <- data.frame(
  h = h,
  cv_noyau_h = cv_noyau_h
)
```

```
Data_erreur_h2 %>%
  ggplot() +
  geom_point(
    mapping = aes(y = cv_noyau_h, x = h),
    colour = "black",
    size = 1.5,
    shape = 21,
    fill = "black"
  ) +
  geom_line(
    mapping = aes(y = cv_noyau_h, x = h),
    colour = "blue",
    size = 1.2,
    linetype = "dashed"
  ) +
  labs(title = "Erreur en fonction des valeurs de h",
       y = "MSE") +
  theme_ggplot()
```




```
h_opt = Data_erreur_h2 %>% slice(which(cv_noyau_h==min(cv_noyau_h)))

h_opt %>%
  kable(
    format = "latex",
    booktabs = T,
    caption = "Estimation à noyau",
    col.names = c("Valeur de h", "Erreur obtenue")
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

TABLE 6 – Estimation à noyau

Valeur de h	Erreur obtenue
3.2	2.00995e-09

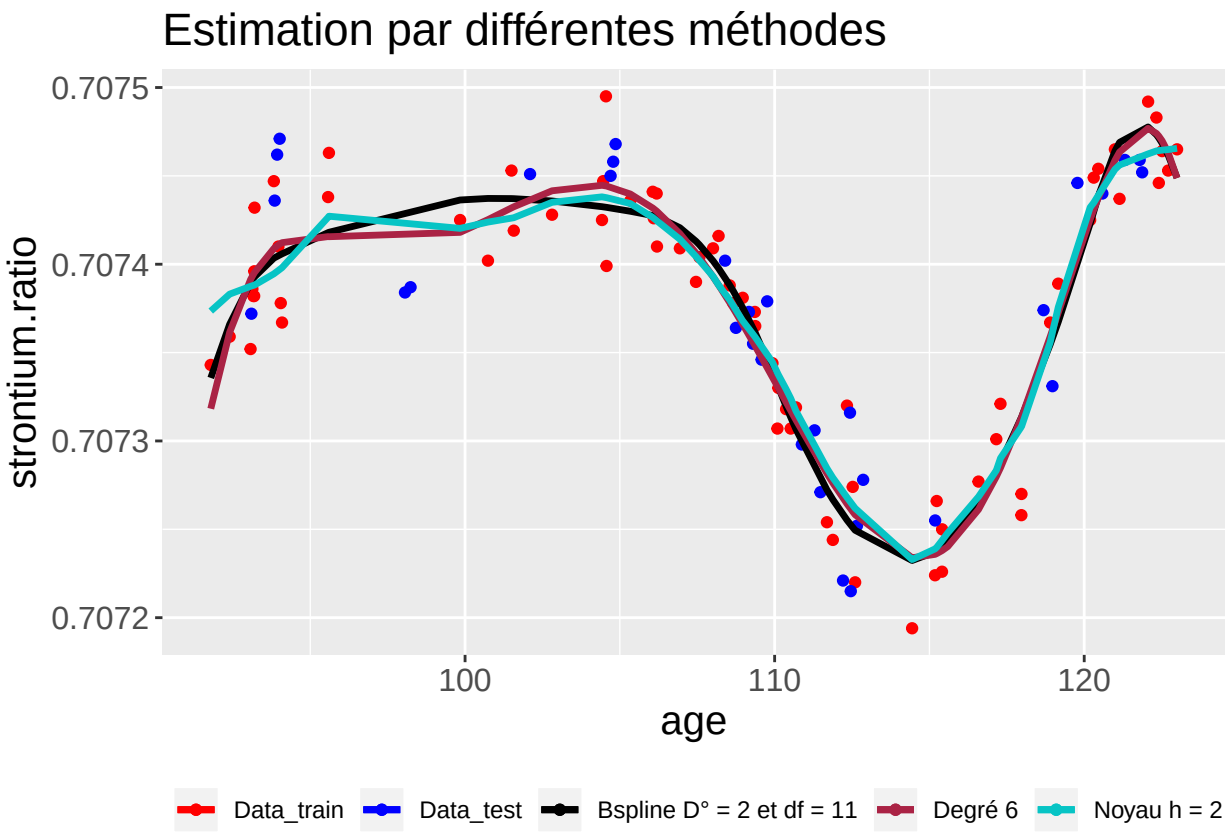
L'erreur minimale est obtenue avec $\mathbf{h = 3.2}$.

Nous observons que l'erreur a largement diminué entre $\mathbf{h = 0.5}$ et $\mathbf{h = 2}$, cependant, elle a l'air constante pour les valeurs de **h** supérieures à 2.

Afin de gérer le compromis *biais-variance*, nous allons prendre $\mathbf{h = 2}$.

Nous allons maintenant représenter les courbes obtenues par les trois méthodes (*Régression polynômiale*, *Bspline* et *Estimation à noyau*) :

```r
Data_train %>%
  attach()

ratio.pred_train <-
  pred_y(nrow(Data_train), age, age, 2, strontium.ratio)

Data_train %>%
  ggplot() +
  geom_point(mapping = aes(y = strontium.ratio,
                           x = age,
                           colour = "red")) +
  geom_point(
    data = Data_test,
    mapping = aes(
      y = Data_test$strontium.ratio,
      x = Data_test$age,
      colour = "blue"
    )
  ) +
  geom_line(mapping = aes(age, predict(Bbase), colour = "black"),
            size = 1.2) +
  geom_line(mapping = aes(
    x = age,
    y = predict(Reg_poly_n[[5]]),
    colour = "#aa2345"
  ),
  size = 1.2) +
  geom_line(mapping = aes(y = ratio.pred_train,
                          x = age,
                          colour = "#07c4c5"),
            size = 1.2) +
  scale_color_identity(
    name = "",
    breaks = c("red", "blue", "black", "#aa2345", "#07c4c5"),
    labels = c(
      "Data_train",
      "Data_test",
      "Bspline D° = 2 et df = 11",
      "Degré 6",
      "Noyau h = 2 "
    ),
    guide = "legend"
  ) +
  labs(title = "Estimation par différentes méthodes") +
  theme(legend.position = "bottom", legend.box = "horizontal") +
  theme_ggplot()
```




```
Data_train %>%
  detach()
```

Nous constatons d'après le graphe ci-dessus que l'estimation à noyau donne une meilleure approche pour toutes les valeurs inférieures à **105** millions d'années et supérieures à **110**, cependant nous remarquons que toutes les méthodes estiment bien notre variable d'intérêt pour les valeurs entre **105** et **110** millions d'années.

5 Estimation par splines quadratiques pénalisées

Nous allons dans cette partie construire une fonction qui permettra d'estimer une fonction de régression à l'aide de splines pénalisées quadratiques.

D'après le cours, nous avons les formules suivantes :

La position des noeuds est aux quantiles de la variables explicatives x .

K optimale :

$$K_{opt} = min(0.25\times \text{ number of unique } x_i\,;\, 35)$$

Matrice de lissage :

$$S_\lambda = X(X^TX + \lambda D)^{-1}X^T$$

Les valeurs ajustées sont données par :

$$\hat{y} = S_\lambda y$$

La validation croisée généralisée (GCV) :

$$GCV(\lambda) = \sum_{i=1}^{n} (\frac{Y_i - \hat{Y}_i}{1 - \frac{1}{n} Trace(S_\lambda)})^2$$

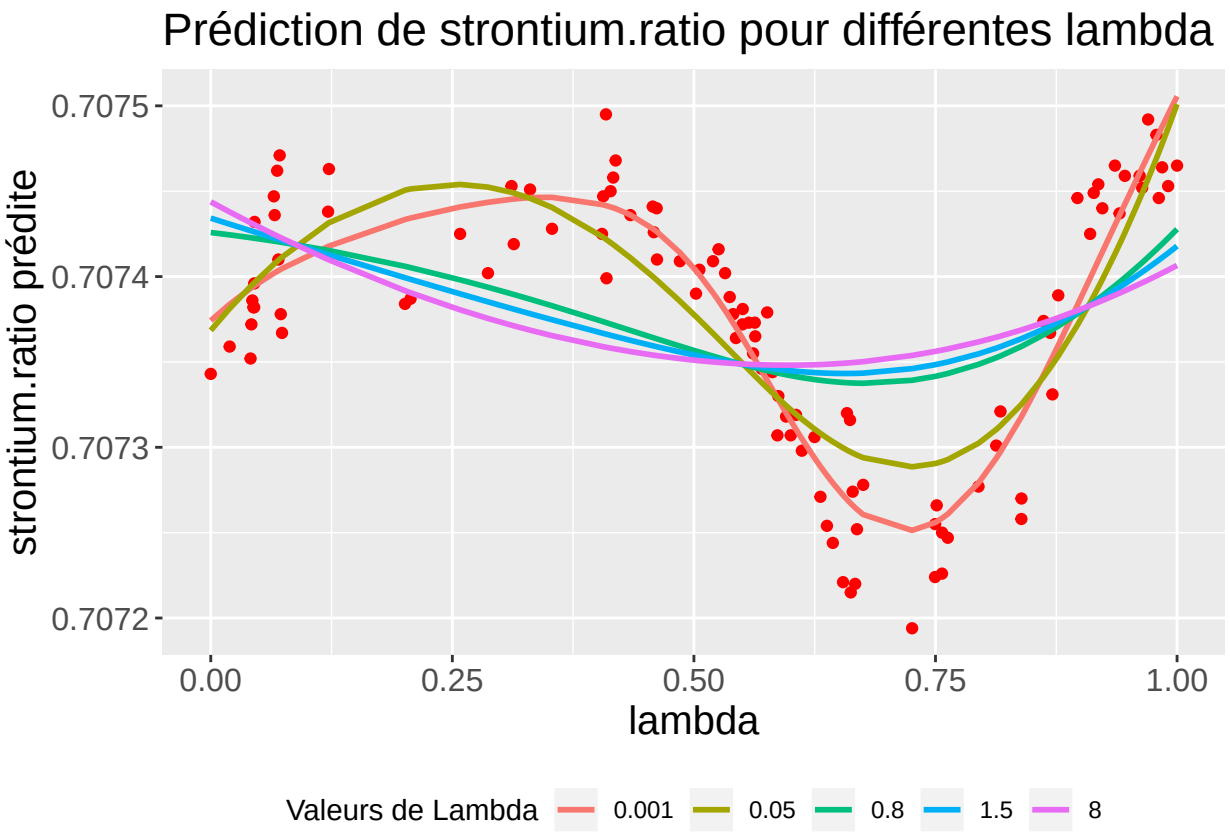
Nous allons nous servir de ces formules pour coder notre fonction.

```
#' Calcule de la trace d'une matrice
#'
#' @param A
#'
#' @return
#' @export
#'
#' @examples
trace <- function(A) {
  n <- dim(A)[1] # Dimension de la matrice

  tr <- 0 # Initialisation de la valeur initiale de la trace

  tr <- lapply(1:n, function(k)
    A[k, k]) %>%
    unlist() %>%
    sum()

}
```

```
fossil %>%
  attach()

K_optimale <- (0.25 * fossil %>%
                 nrow()) %>%
  round() %>%
  min(35)

reg.spline <- function(xech, yech, lambda) {
  quantile_noeuds <-
    seq(0, 1, length = K_optimale + 2)[-c(1, K_optimale + 2)]

  noeud_pos <- quantile(xech, prob = quantile_noeuds)

  Z <- outer(xech, noeud_pos, "-")

  Xpol <- Z ^ 2 * (Z > 0)

  X <- cbind(rep(1, fossil %>%
                   nrow()), xech, xech ^ 2, Xpol)

  S_lambda <- X %*% solve(crossprod(X) + lambda * diag(c(rep(0, 3),
                                                         rep(1, K_optimale)))) %*%
    t(X)


  Y_chap <- S_lambda %*% yech
```

```
  Trace <-
    solve(crossprod(X) + lambda * diag(c(rep(0, 3),
                                         rep(1, K_optimale)))) %*%
    crossprod(X) %>%
    trace()

  return(list(Trace, Y_chap))


}

grille_lambda <- c(0.001, 0.05, 0.8, 1.5, 8)

y_chap_lambda <-
  map(grille_lambda, function(i)
    reg.spline(age_norm, strontium.ratio, i)[2] %>% unlist())

fossil %>%
  detach()
```

```
Data_pred_lambda <- do.call(cbind, y_chap_lambda) %>%
  melt() %>%
  mutate(
    lambda = rep(grille_lambda, each = nrow(fossil)),
    age = rep(fossil$age_norm, length(grille_lambda))
  )


Data_pred_lambda %>%
  ggplot() +
  annotate(
    geom = 'point',
    x = fossil$age_norm,
    y = fossil$strontium.ratio,
    colour = "red"
  ) +
  geom_line(aes(x = age, y = value, col = as.factor(lambda)), size = 1) +

  labs(
    title = "Prédiction de strontium.ratio pour différentes lambda",
    x = "lambda",
    y = "strontium.ratio prédite",
    colour = "Valeurs de Lambda"
  ) +
  theme(legend.position = "bottom",
        legend.box = "horizontal") +
  theme_ggplot()
```

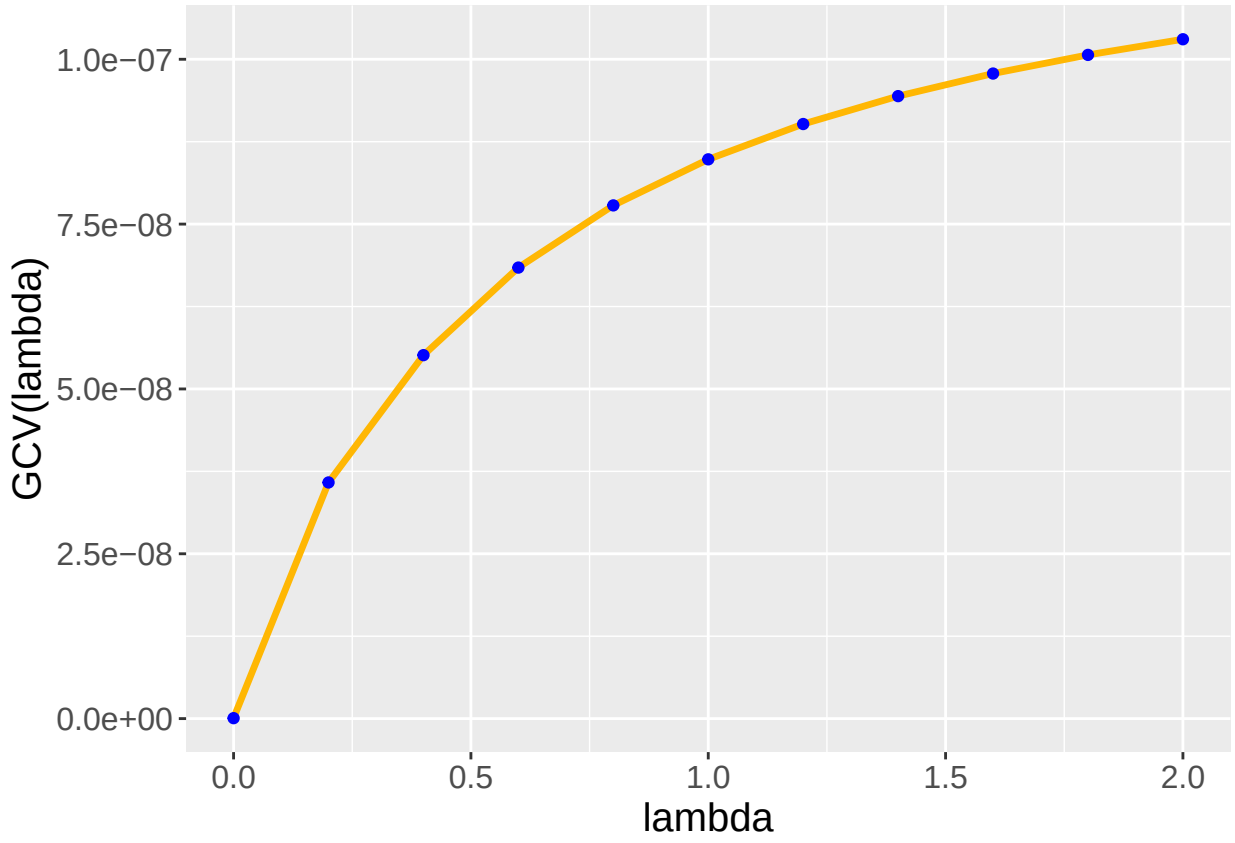


Nous constatons que plus la valeur de λ est petite plus la prédiction est précise.

Afin de choisir la valeur de λ qui donne la meilleure précision, nous allons nous servir de la **validation croisée généralisée**.

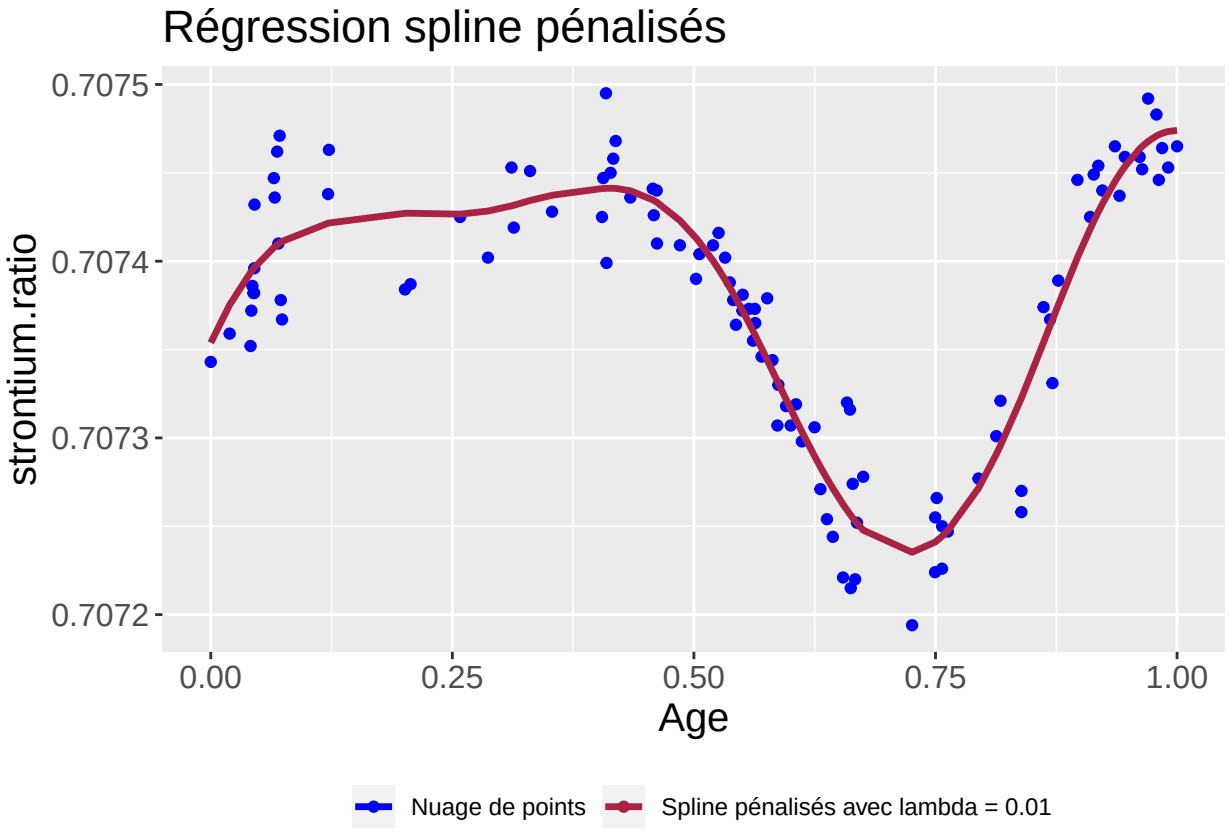
```
grille_lambda1 <- seq(0, 2, by = 0.2)

y_chap_lambda1 <-
  map(grille_lambda1, function(i)
    reg.spline(age_norm, strontium.ratio, i)[2] %>% unlist())

trace_lambda <-
  map(grille_lambda1, function(i)
    reg.spline(age_norm, strontium.ratio, i)[1] %>% unlist())

GCV <- function(yech, y_chap, tr_lambda) {
  form1 <- (yech - y_chap) ^ 2 / ((1 - mean(tr_lambda)) ^ 2)

  return(sum(form1))
}

GCV1 <-
  lapply(1:length(grille_lambda1), function(i) {
    GCV(fossil$strontium.ratio, y_chap_lambda1[[i]], trace_lambda[[i]])
  })
```

```r
GCV_lambda <- GCV1 %>%
  unlist() %>%
  cbind(lambda = grille_lambda1) %>%
  as.data.frame() %>%
  rename(GCV_val = '.')

GCV_lambda %>%
  ggplot() +
  geom_line(
    mapping = aes(x = lambda, y = GCV_val),
    col = "#ffb703",
    size = 1.2
  ) +
  geom_point(
    mapping = aes(x = lambda, y = GCV_val),
    col = "blue",
    shape = 21,
    fill = "blue"
  ) +
  labs(y = "GCV(lambda)") +
  theme_ggplot()
```



D'après ce graphe, l'erreur minimale par validation croisée est obtenue avec des valeurs de λ plus petites. Nous allons prendre $\lambda = 0.0001$:

```r
fossil %>%
  ggplot() +
  geom_point(mapping = aes(x = age_norm, y = strontium.ratio, col = "blue")) +
  geom_line(mapping = aes(
    x = age_norm,
    y = unlist(reg.spline(age_norm, strontium.ratio, 0.0001)[2]),
    col = "#aa2345"
  ),
  size = 1.2) +
  scale_color_identity(
    name = "",
    breaks = c("blue", "#aa2345"),
    labels = c("Nuage de points",
               "Spline pénalisés avec lambda = 0.01"),
    guide = "legend"
  ) +
  labs(title = "Régression spline pénalisés", x = "Age", y = "strontium.ratio") +
  theme(legend.position = "bottom", legend.box = "horizontal") +
  theme_ggplot()
```




```r
y_pred <- reg.spline(age_norm, strontium.ratio, 0.0001)[2] %>%
  unlist()

MSE_Bs_gener <- mean((y_pred-strontium.ratio)^2)

MSE_Bs_gener
```

```
## [1] 6.4005242236182e-10
```

L'erreur obtenue est égale **6.400524e-10**.

6 Conclusion

Nous avons étudié dans ce **TP** différentes méthodes non-paramétriques afin de choisir celle qui estime bien notre variable d'intérêt.

Nous avons commencé avec l'estimation par régression polynômile, ensuite par Bsplines, à noyau et finalement l'estimation par Bsplines pinalisées.

Le tableau ci-dessus présente l'erreur obtenue par chacune de ces méthodes :

TABLE 7 – L'erreur obtenue par chaque méthode

Méthodes utilisées	Erreurs obtenues
Erreur_RegPoly	**7.76680e-10**
Erreur_Bspline	**7.89140e-10**
Erreur_noyau	**2.00995e-09**
Erreur_spline_pena	**6.40050e-10**

Nous déduisons d'après cette table que l'estimation par **Splines quadratiques pénalisées** avec $\lambda = 0.0001$ est la plus adaptée à notre jeu de données car elle donne la faible erreur.

Nous remarquons aussi que l'erreur obtenue par la régression polynômiale (*Avec degré 6*) est plus petite que celle obtenue par les Bspline (*Degré 2, Degrée de liberté 11*).

7 Travail supplémentaire

Pour l'estimation à noyau de la régression, nous avons utilisé la méthode LOOCV (*leave-one-out-cross-validation) pour diviser notre base de données, cette méthode consiste à piocher un individu au hasard du jeu de données et le considéré comme un échantillon de test et ainsi de suite jusqu'au dernier individu.

Nous avons essayé de diviser notre jeu de données par validation croisée, voici les résultats obtenus

```
cv_ker <- function(k = 5, data, h, seed = 0207) {
  list_train <- list() # initialize a list
  list_test <- list()
  models_fit <- list()

  n <- data %>% # cardinal of data
    nrow()

  purchase_pred <- rep(0, n) # initialize the response vector

  klist <- 1:k %>% as.list()

  fold <- 1:k %>% # create folds
    rep(ceiling(n / k)) %>%
    sample(n)

  samples_test <- map(klist, function(i)
    data %>%
```

```
      slice(which(fold == i))) # list of data train set

  samples_train <- map(klist, function(i)
    data %>%
      slice(which(fold != i))) # list of data test set

  ker_reg_train <-
    map(klist, function(i) {
      map(1:nrow(samples_train[[i]]), function(j)
        hat_f(
          samples_train[[i]]$age[j],
          samples_train[[i]]$age,
          h,
          samples_train[[i]]$strontium.ratio
        )) %>%
        unlist()
    })

  error_cv_ker <-
    lapply(klist, function(i) {
      mean((ker_reg_train[[i]] - samples_train[[i]]$strontium.ratio) ^ 2)
    }) %>% unlist()

  return(mean(error_cv_ker))

}

erreur_cv_noyau <- cv_ker(5, data = fossil, 3)
```

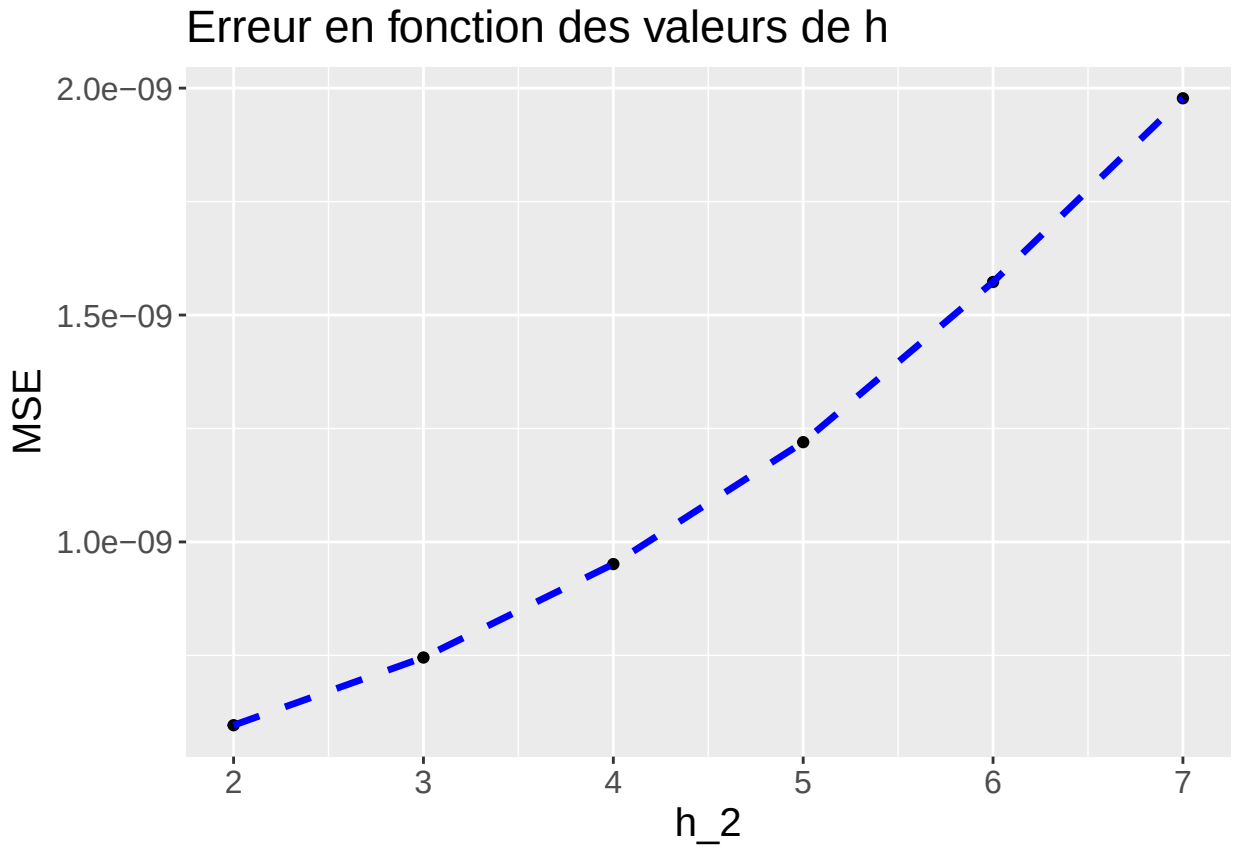
Dans le code ci-dessus, nous avons divisé notre jeu de données en utilisant la validation croisée ($K = 5$), puis nous avons appliqué la fonction **hat_f** codée auparavent pour calculer la prédiction de la variable en question et finalement nous avons calculé l'erreur moyenne des erreurs obtenue dans chaque **fold** de la validation croisée.

```
h_2 <- seq(from = 2, to = 7, by = 1)
cv_noyau_h2 <-
  lapply(h_2, function(j) {
    erreur_cv_noyau <- cv_ker(5, data = fossil, j)
  }) %>%
  unlist()


Data_erreur_h_2 <- data.frame(h = h_2,
                              cv_noyau_h = cv_noyau_h2)
```

Ici, nous avons calculé l'erreur pour différentes valeurs de **h** et nous avons obtenu le graphe suivant :

```
Data_erreur_h_2 %>%
  ggplot() +
  geom_point(
    mapping = aes(y = cv_noyau_h2, x = h_2),
    colour = "black",
    size = 1.5,
    shape = 21,
    fill = "black"
```

```
  ) +
  geom_line(
    mapping = aes(y = cv_noyau_h2, x = h_2),
    colour = "blue",
    size = 1.2,
    linetype = "dashed"
  ) +
  labs(title = "Erreur en fonction des valeurs de h",
       y = "MSE",
       v = "h") +
  theme_ggplot()
```



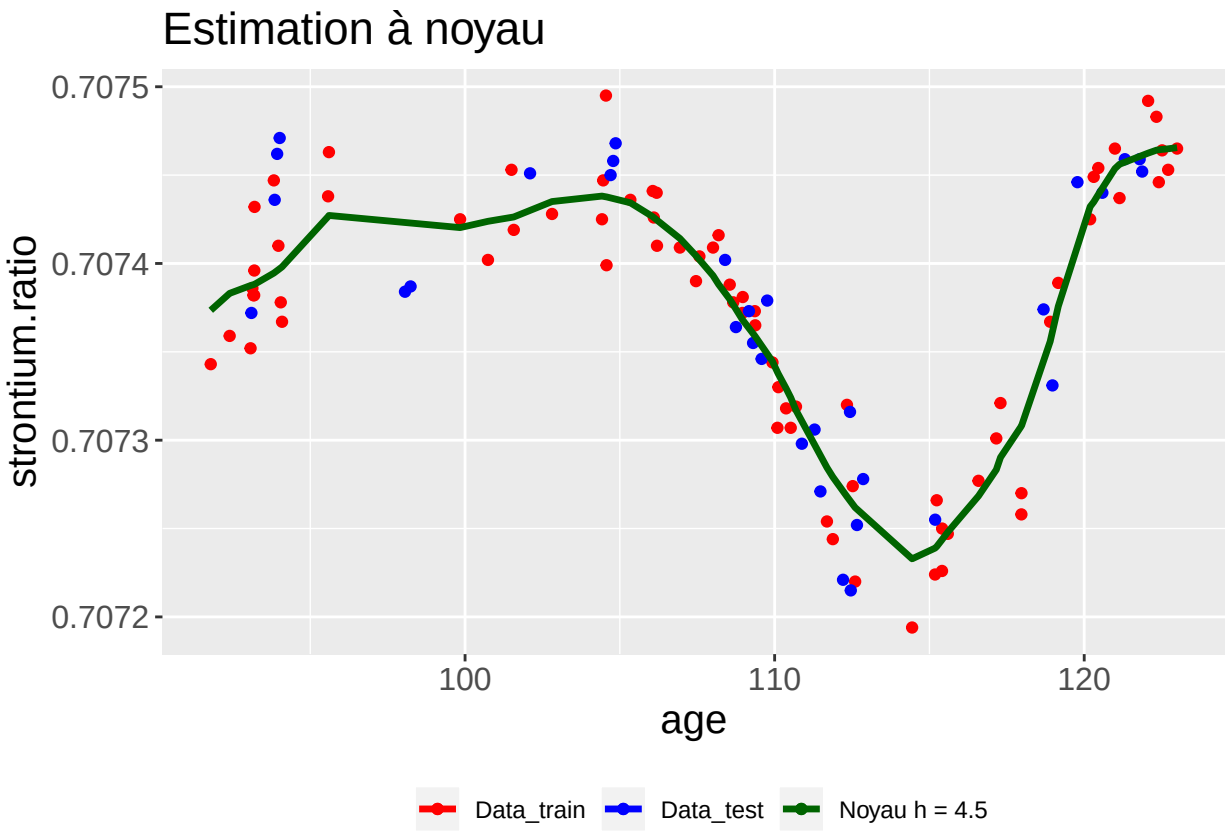
Contrairement à ce que nous avons trouvé par la méthode **LOOCV**, ici, l'erreur obtenue par validation croisée ($k = 5$) augmente avec l'augmentation de la valeur de **h**

```
h_opt2 = Data_erreur_h_2 %>% slice(which(cv_noyau_h2 == min(cv_noyau_h2)))
h_opt2 %>%
  kable(
    format = "latex",
    booktabs = T,
    caption = "Estimation à noyau par validation croisée",
    col.names = c("h", "Erreur obtenue")
  ) %>%
  kable_styling(latex_options = c("striped", "hold_position"))
```

Nous avons obtenu une erreur minimale de **5.9922353643723e-10**.

TABLE 8 – Estimation à noyau par validation croisée

h	Erreur obtenue
2	5.9613e-10



Nous n'avons pas bien compris pourquoi la courbe des erreurs en fonction des h est décroissante, par contre avec la LOOCV, nous avons eu l'inverse.

Est-ce que cela revient au fait qu'on ne peut pas appliquer la validation croisée classique pour l'estimation à noyau ?