Analyse de données fonctionnelles avec le paquet `fda`

Christophe Pouzat

Jeudi 29 mars 2012

Outline

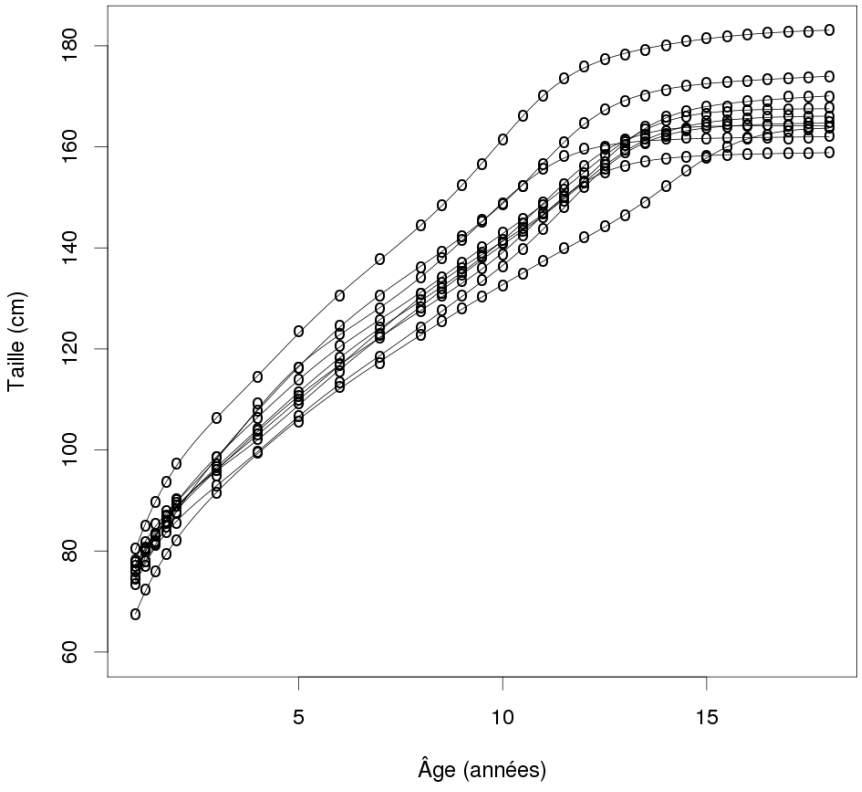
Sommaire

L'analyse de données fonctionnelles, c'est quoi ?

- l'analyse de données fonctionnelles **n'est pas l'analyse fonctionnelle des mathématiciens**, même si elle fait appel à certains outils de cette discipline ;
- les données fonctionnelles ressemblent aux données vectorielles, **mais ressemblance n'implique pas identité** ;
- c'est cette différence qui rend l'analyse fonctionnelle beaucoup plus « marrante » – de mon point de vue – que l'analyse multivariée classique.

La croissance des filles : Un premier exemple de données fonctionnelles

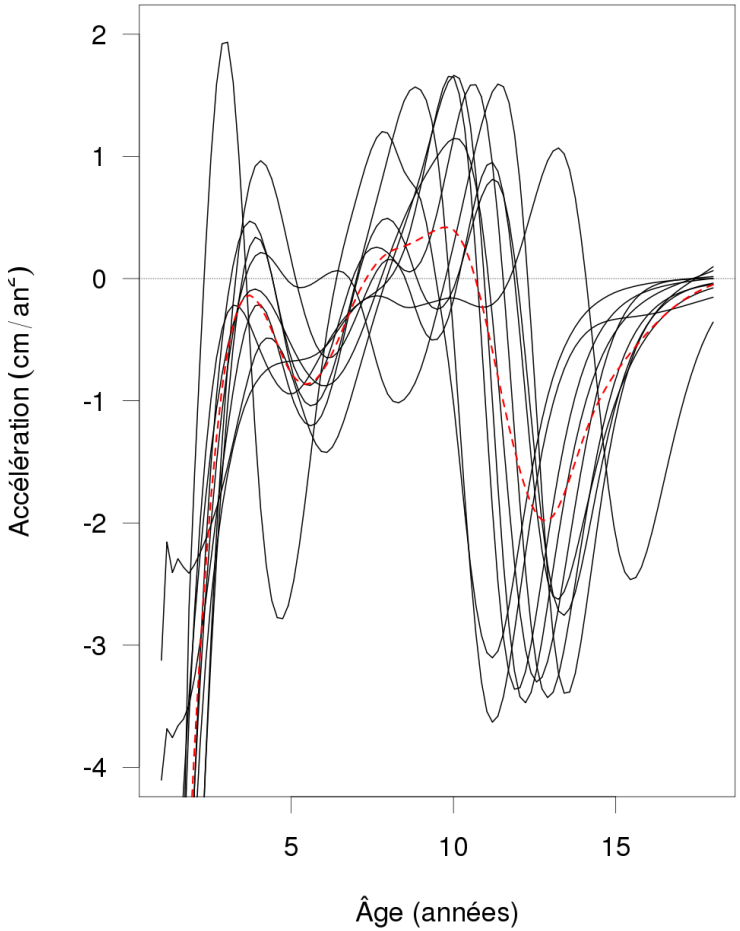
Étude de croissance de Berkeley



Les tailles de 10 filles mesurées à 31 âges. Les cercles marquent les 31 âges de mesures *de répartition non-uniforme*. L'erreur sur chaque mesure est de l'ordre de 3 mm. En principe, les mesures auraient pu être effectuées à n'importe quel âge.

La croissance des filles : Dérivée seconde

Poussée de croissance pubertaire



1. Certaines propriétés remarquables, apparaissent beaucoup plus nettement sur les dérivées premières ou secondes des données.
2. Résumer les données par une moyenne à âge fixe n'est pas forcément une bonne idée.
3. Il peut être intéressant de séparer les variations d'amplitude des variations « temporelles ».

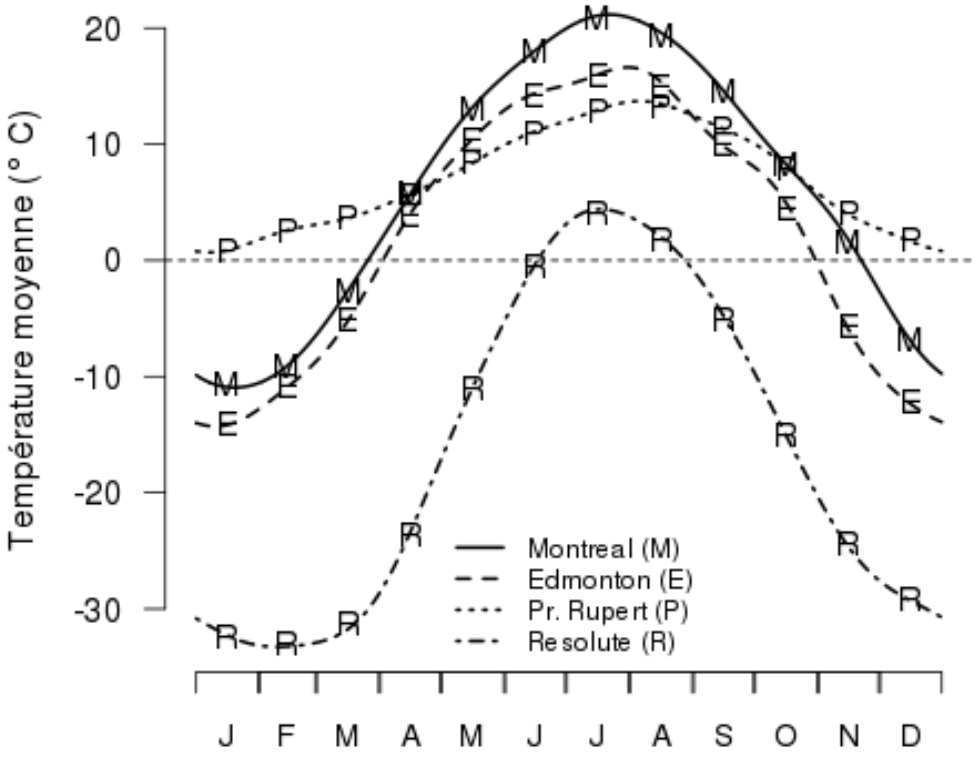
La météo canadienne : où passer ses prochaines vacances

Jeu de données `CanadianWeather`

Les 31 stations (27 disques rouges et 4 nommées) du jeu de données. Le jeu contient les temperatures et précipitations moyennes quotidiennes, moyennées de 1960 à 1994.

Températures moyennes et estimations fonctionnelles

Températures des 4 stations

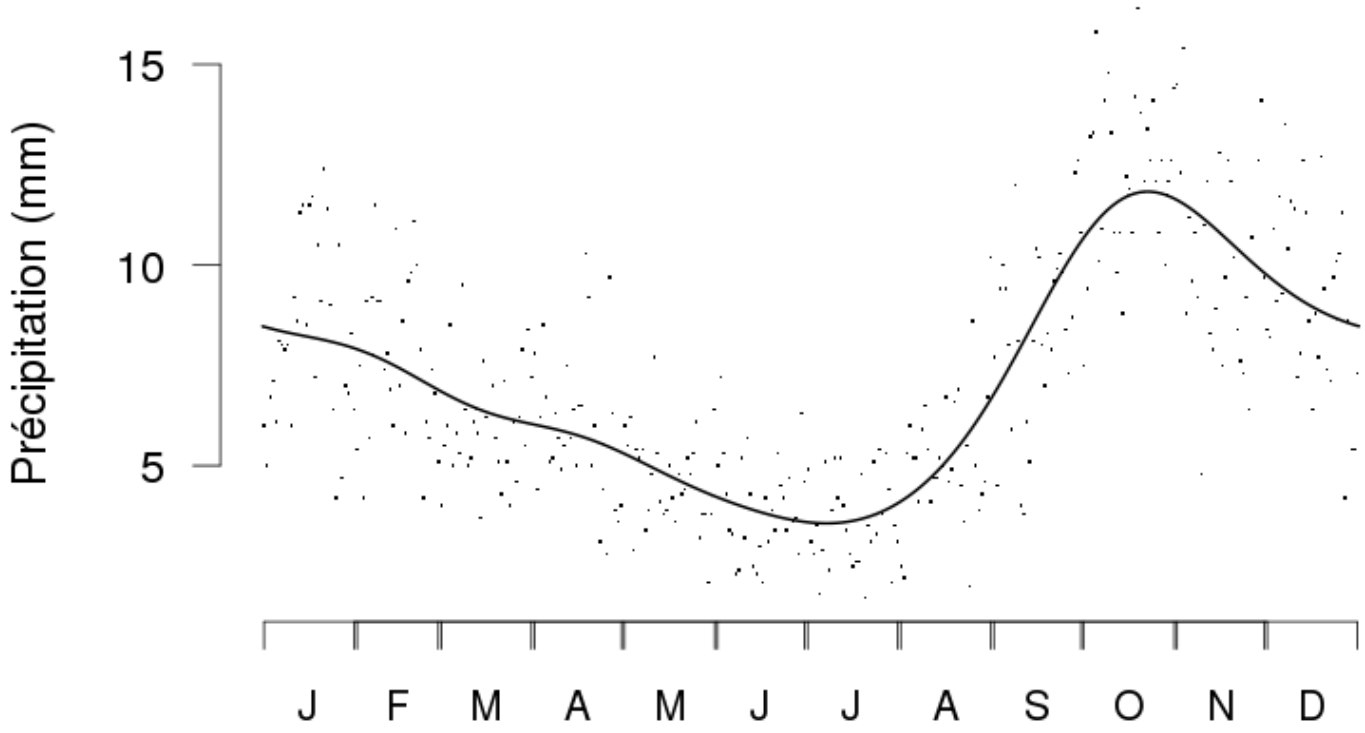


Les moyennes mensuelles sont figurées par les lettres. Les courbes sont des estimations fonctionnelles (avec une base de Fourier) pour chaque station.

Les premières étapes de l'analyse de données fonctionnelles : interpolation et lissage

- « l' élément fonctionnel » i d'un échantillon se présente comme un ensemble fini de mesures : $\{ y_{i1}, \ldots, y_{in} \}$;
- la première tâche est de convertir ces mesures en fonction x_i dont les valeurs $x_i(t)$ sont calculables pour toute valeur de l'argument t (dans un certain domaine) ;
- si les observations sont faites sans erreur, on fait une interpolation ;
- si les observations sont faites avec erreur, on fait du lissage.

Exemple de lissage



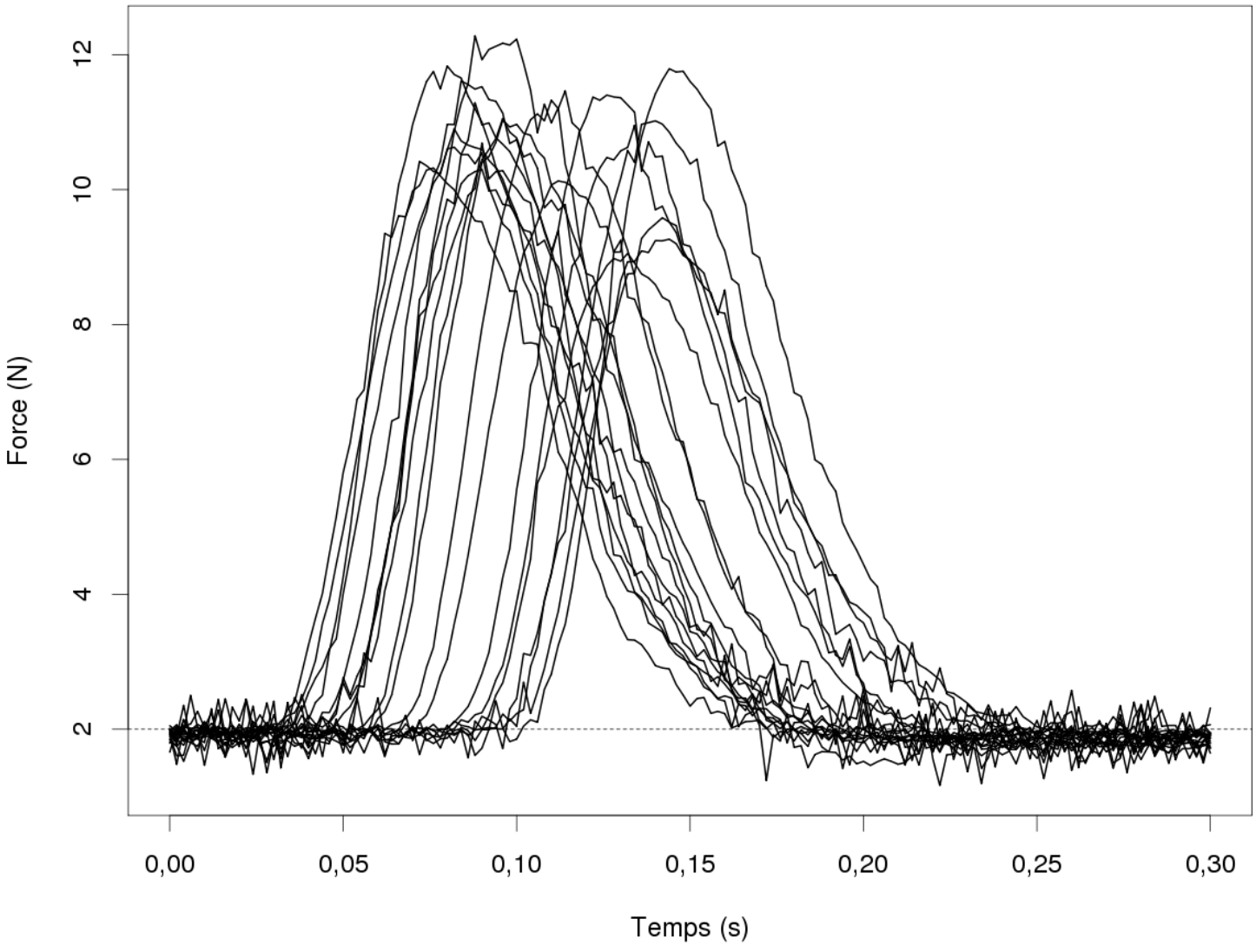
Les points représentent la pluie quotidienne moyenne à *Prince Rupert* en Colombie britannique.

Les premières étapes de l'analyse de données fonctionnelles : alignement

La figure suivante présente des données de biomécanique :

- 20 enregistrements de la force exercée entre le pouce et l'index d'un sujet ;
- le sujet devait maintenir une force « de repos » de l'ordre de 2 Newton avant de développer une force maximale avant de retourner au niveau de repos ;
- le but de la manipe est l'étude du groupe musculaire contrôlant le pouce et l'index ;
- le problème qui se pose avant de commencer l'analyse proprement dite, et celui de l'alignement des traces. . .

Les données de force du groupe pouce-index



Après les premières étapes, l'exploration de la variabilité

- jusqu'ici nous avons vu essentiellement des exemples de visualisation d'échantillons de fonctions ;
- nous allons aussi être intéressés par des statistiques descriptives comme la moyenne, l'écart type ou la fonction de covariance ;
- nous allons aussi vouloir généraliser l'analyse en composantes principales.

Quelques références

- nous allons utiliser le paquet `fda` (*functional data analysis*) développé par James Ramsay et ses collaborateurs ;
- le paquet est disponible sur `CRAN` ;
- il contient aussi une version des fonctions pour `Matlab` (mais je ne les ai pas testées) ;
- le bouquin « Functional Data Analysis with R and MATLAB » de Ramsay Hooker et Graves, *Springer*, Use R !, 2009 est un bon point de départ (bouquin « associé » au paquet `fda`) ;
- le bouquin « Functional Data Analysis » de Ramsay et Silverman, *Springer*, 2006, est plus théorique, plus complet mais compréhensible ;
- le bouquin « Nonparametric Functional Data Analysis » de Ferraty et Vieu, 2006, couvre à peu près le même domaine que le précédent ; il est un peu plus théorique et utilise une approche différente pour passer des données brutes aux fonctions.

Sommaire

Construction des fonctions

- On se donne un ensemble $\{\phi_k\}$, k = 1,. . . ,K de fonctions de base ;
- Une fonction « quelconque » $x(t)$ est alors définie ou approximée par une somme des fonctions bases multipliées par des coefficients ;
- plus explicitement on va avoir : $x(t) = \sum_{k=1}^{K} c_k \phi_k(t)$;
- on a donc deux problèmes à résoudre :
  - le choix de l'ensemble $\{\phi_k\}$, c.-à-d. le choix de la base ;
  - étant données des observations $y_i = x(t_i) + \epsilon_i$, comment choisir les coefficents $\{c_k\}$?

Les bases de `fda` (1)

Le paquet `fda` permet de travailler avec huit types de bases :

- base constante, $x(t) =$ cste, avec `create.constant.basis` ;
- base « monomiale », $x(t) = c_1 + c_2 t + c_3 t^2 + \cdots$, avec `create.monomial.basis` ;
- base polynomiale, $x(t) = c_1 + c_2(t - ctr) + c_3(t - ctr)^2 + \cdots$, avec `create.polynomial.basis` ;
- base « en puissances », $x(t) = \sum_{k=1}^{K} c_k (t - ctr)^{p_k}$, où les p_k ne sont pas nécessairement entiers et pas nécessairement positifs, avec `create.power.basis` ;
- ...

Les bases de `fda` (2)

- ...
- base « polygonale », $x(t)$ est linéaire par morceaux, un cas spécial de *splines* (voir ci-dessous), avec `create.polygonal.basis` ;
- base « exponentielle », $x(t) = \sum_{k=1}^{K} c_k \exp(b_k t)$, avec `create.exponential.basis` ;
- base « de Fourier », $x(t) = c_1 + \sum_{i=1}^{(K-1)/2} c_{2i} \sin(i 2\pi / T) + \sum_{j=1}^{(K-1)/2} c_{2j+1} \cos(j 2\pi / T)$, avec `create.fourier.basis` ;
- base de fonctions splines, $x(t)$ est polynomiale par morceaux avec conditions de continuité sur x et ses dérivées aux jointures entre les morceaux, avec `create.bspline.basis`.

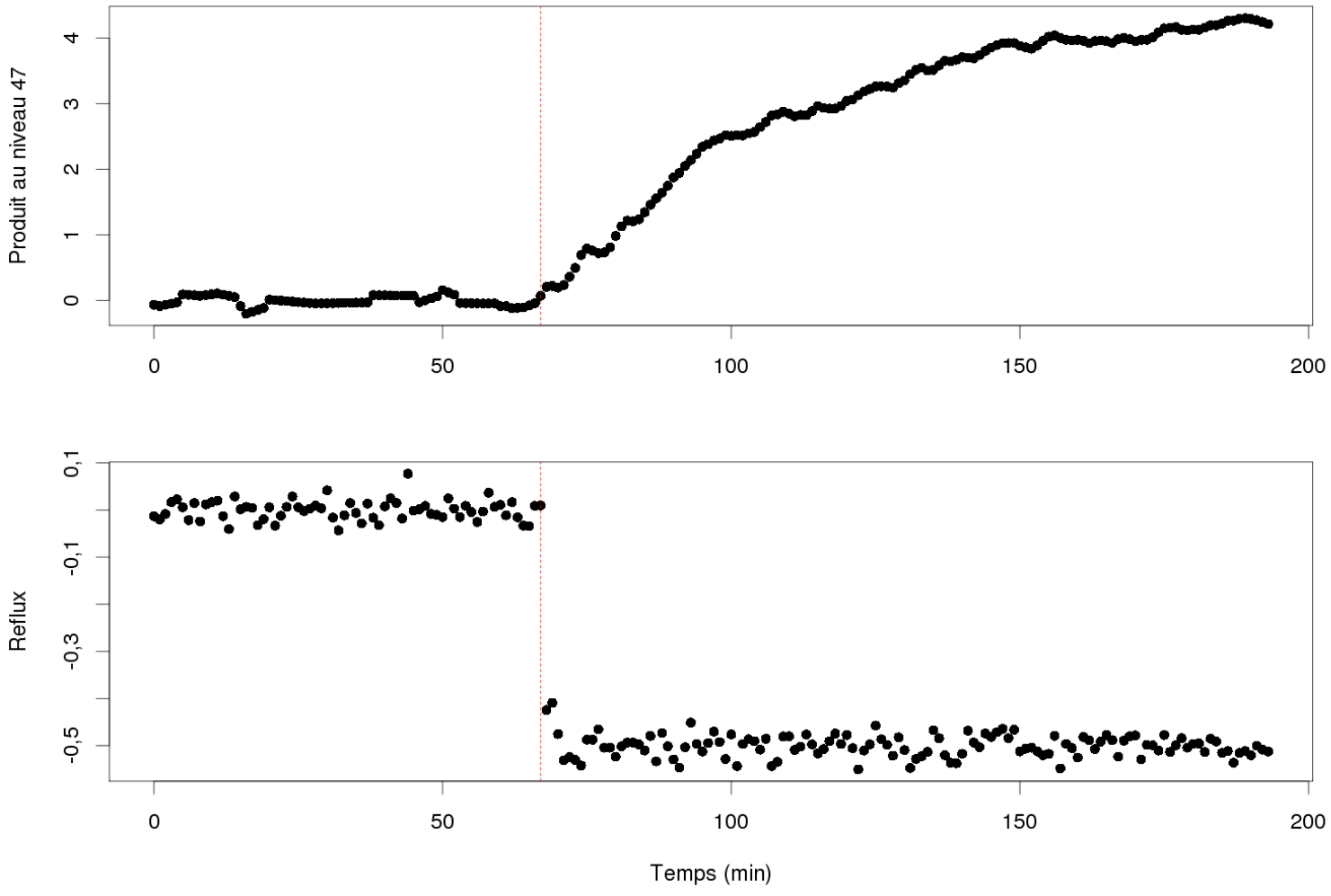
Exemples

Pour des exemples de constructions de bases et des méthodes `print`, `summary` et `plot` associées, voir la première démo. . .

Construction d'objets de classe `fd`

- comme nous venons de le voir, nous construisons nos fonctions en « combinant » une base avec des coefficients ;
- les fonctions type `create.bspline.basis` nous permettent de construire une base ;
- nous allons illustrer la construction d'objet de classe `fd` (*functional data*) avec le jeu `refinery` du paquet ;
- ce jeu de données est un `data.frame` qui contient les résultats d'une expérience dans une colonne de distillation d'une raffinerie de pétrole ;
- le `data.frame` comporte trois variables : `Time` (le temps en minutes), `Tray47` (la quantité de produit au niveau 47 de la colonne), `Reflux` (le flux de vapeur dans la colonne au niveau 47, la valeur est négative quand le flux est sortant) ;
- la figure suivante montre clairement une « discontinuité » à la 67^{e} minute.

Les données du jeu `refinery`



Construction de la base pour l'estimation fonctionnelle de `Tray47`

- le cas qui nous intéresse est un peu particulier *du fait de la discontinuité de la dérivée de la variable* `Tray47` à la 67^{e} minute ;
- nous allons utiliser une base de fonctions splines (cubiques) en localisant *plusieurs nœuds* à la 67^{e} minute ;
- cela va nous permettre d'avoir une dérivée discontinue ;
- on le fait de la façon suivante :

```
data(refinery)
refinKnots <- c(seq(0,67,len=3),67,seq(67,193,len=3))
refineryBS <- create.bspline.basis(c(0,193),norder=4,
                                   breaks=refinKnots)
```

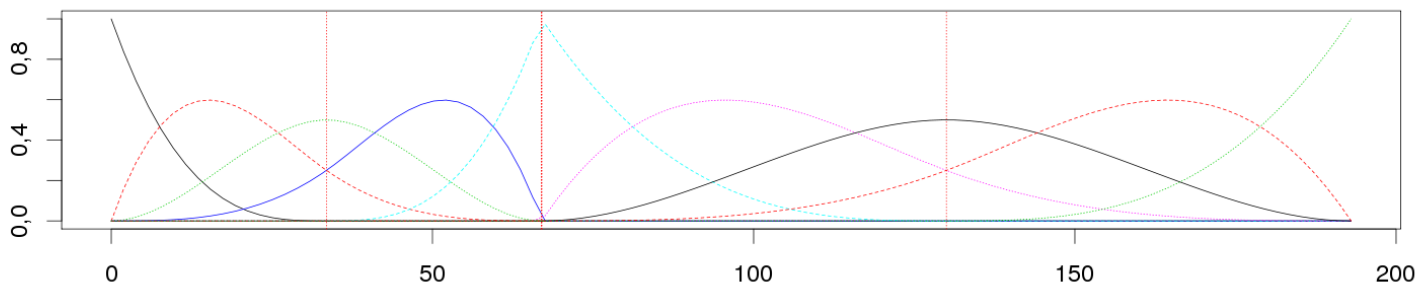
print de refineryBS

Un visualisation rapide de refineryBS donne :

```
refineryBS


Basis object:

  Type:   bspline

  Range:  0  to  193

  Number of basis functions:  9

  Order of spline:  4
[1] "  Interior knots"
[1]  33,5  67,0  67,0  67,0 130,0
```

plot de refineryBS

```
par(cex=2)
plot(refineryBS,cex=2)
```



Obtention des coefficients « à la main » (1)

- nous supposons un modèle du type : $y_i \sim x(t_i) + \epsilon(t_i)$, où les y_i sont les éléments de la variable `Tray47`, $x(t_i)$ est ce que nous cherchons et $\epsilon(t_i)$ est un bruit d'espérance nulle, de variance finie et non auto-corrélé (en tout cas on l'espère) ;
- ici les $x(t_i)$ ont la forme : $x(t_i) = \sum_{j=1}^{9} c_i \phi_j(t_i)$; où les ϕ_j sont les fonctions de notre base ;
- de façon classique pour un modèle linéaire, nous commençons par créer notre `matrice d'expérience` (*design matrix*), $\mathbf{X}$, qui sera ici une matrice à 194 lignes (nombres de temps de mesure) et à 9 colonnes (nous avons 9 fonctions dans notre base) ;
- nous obtenons $\mathbf{X}$ en appliquant la méthode `predict` à `refineryBS` :

```
refineryX <- predict(refineryBS,refinery$Time)
dim(refineryX)
```

```
 [1] 194   9
```

Obtention des coefficients « à la main » (2)

- à ce stade l'élément $(\mathbf{X})_{ij}$ est $\phi_j(t_i)$;
- si nous écrivons $\mathbf{c}$ le vecteur $(c_1,\ldots,c_9)$ et $\mathbf{x}$ le vecteur $(x(t_1),\ldots,x(t_{194}))$, notre prédiction s'écrit sous forme matricielle : $\mathbf{x} = \mathbf{X}\,\mathbf{c}$;
- l'estimateur des moindres carrés $\mathbf{b}$ de $\mathbf{c}$ est alors donné par : $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$, où $\mathbf{y}$ est le vecteur $(y_1,\ldots,y_{194})$ et où $\mathbf{X}'$ est la transposée de $\mathbf{X}$;
- si tout ça est un peu difficile pour vous, consultez les notes de D. Bates sur le sujet http ://www.stat.wisc.edu/courses/st849-bates/lectures/ ;
- avec `R` on effectue cela comme suit :

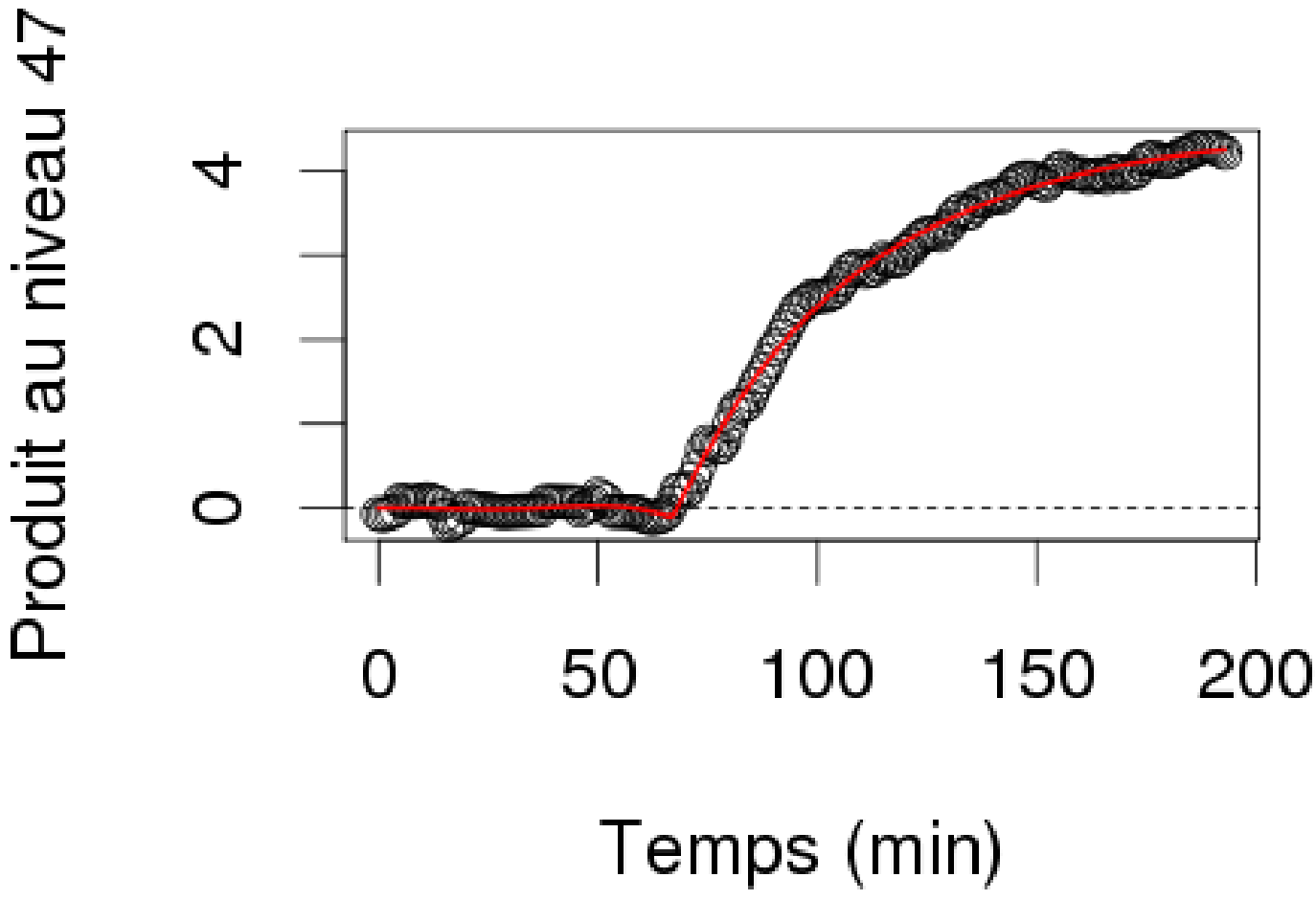
```
refineryB <- solve(crossprod(refineryX),
                   crossprod(refineryX,refinery$Tray47))
```

Construction de `refineryFD`

- maintenant que nous avons notre base, `refineryBS`, et nos coefficients, `refineryB`, nous n'avons plus qu'à les combiner pour obtenir notre première donnée fonctionnelle ;
- on emploie la fonction `fd` pour effectuer cette opération :

```
refineryFD <- fd(refineryB,refineryBS)
```

Le résultat avec la méthode `lines` appliquée à `refineryFD`



Vers une méthode automatique (1)

L'approche que nous venons de suivre présente, une fois le *type* de base choisi en fonction de la nature du problème, plusieurs inconvénients qui peuvent être formulés comme d'embêtantes questions ouvertes :

- Comment choisir le nombre de fonctions dans la base ?
- Comment positionner les fonctions ?

Vers une méthode automatique (2)

Heureusement, ces problèmes de régression non-paramétrique sont « vieux » et des solutions ont été formulées :

- on utilise assez de fonctions pour suivre « toute caractéristique intéressante » des données, ce qui peut vouloir dire autant de fonctions que d'observations (ou même plus) ;
- pour éviter de sur-ajuster les données, on emploie une pénalisation ;
- on va ainsi minimiser, pour des fonctions non-périodiques, des critères du type :

$$F(\mathbf{b}) = \sum_{j=1}^{n} (y_j - x(t_j))^2 + \lambda \int \left(\frac{d^2 x(u)}{dt^2} \right)^2 du$$

- avec $x(t) = \sum_{i=1}^{K} b_i \phi_i(t)$; λ, le paramètre de lissage ;
- on pourra être amené à changer l'ordre de la dérivée sous le signe somme suivant que x ou une de ses dérivée nous intéresse ;

Vers une méthode automatique (3)

- pour les fonctions périodiques on emploiera :

$$F(\mathbf{b}) = \sum_{j=1}^{n} (y_j - x(t_j))^2 + \lambda \int \left(\left(\frac{2\pi}{T} \right)^2 \frac{dx(u)}{dt} + \frac{d^3x(u)}{dt^3} \right)^2 du$$

- où T est la période.

Ajustement à λ fixé : exemple de la raffinerie (1)

Pour ajuster un modèle de splines pénalisés au données de production du niveau 47 de la raffinerie, on fait d'abord appel à la fonction `fdPar` :

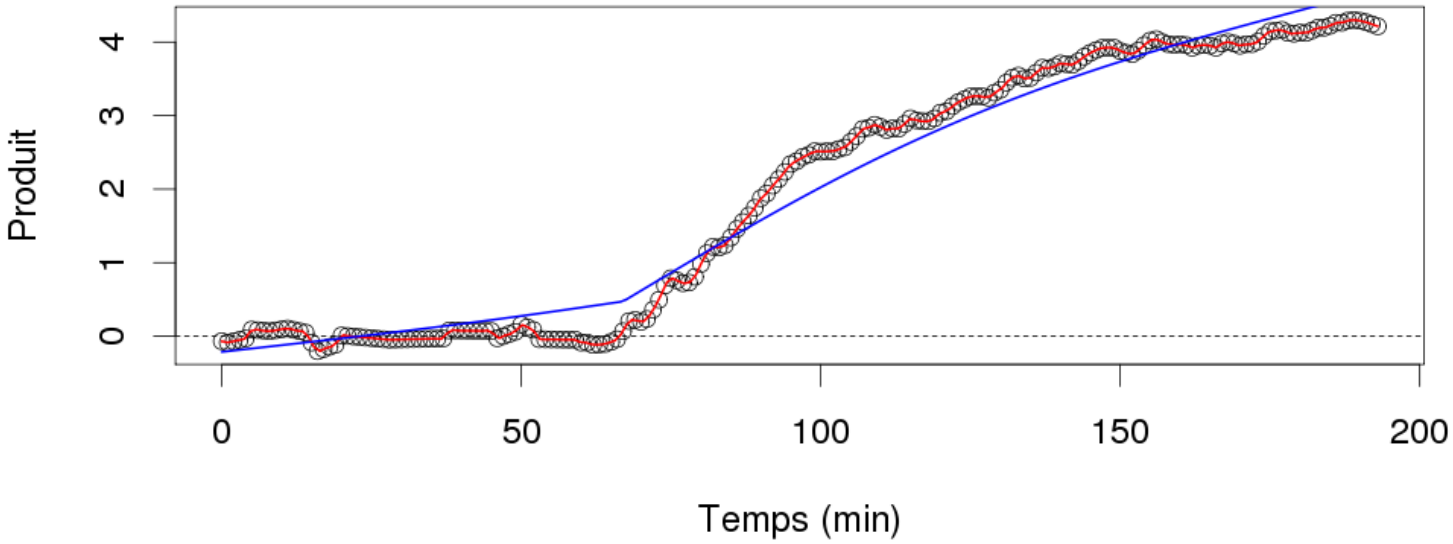
```
temps <- refinery$Time
nœuds <- sort(c(temps,67,67))
raffinerieBS <- create.bspline.basis(c(0,193),
                                     norder=4,
                                     breaks=nœuds)
raffinerie.fdPar <- fdPar(fdobj=raffinerieBS,
                          Lfdobj=2,lambda=0.01)
```

Ajustement à λ fixé : exemple de la raffinerie (2)

On effectue ensuite l'ajustement avec la fonction `smooth.basis` :

```
produit <- refinery$Tray47
raffinerie.smooth <- smooth.basis(argvals=temps,
                                  y=produit,
                                  fdParobj=raffinerie.fdPar)
```

Ajustement à λ fixé : exemple de la raffinerie (3)



En rouge, $\lambda = 0{,}01$, en bleu, $\lambda = 10^6$.

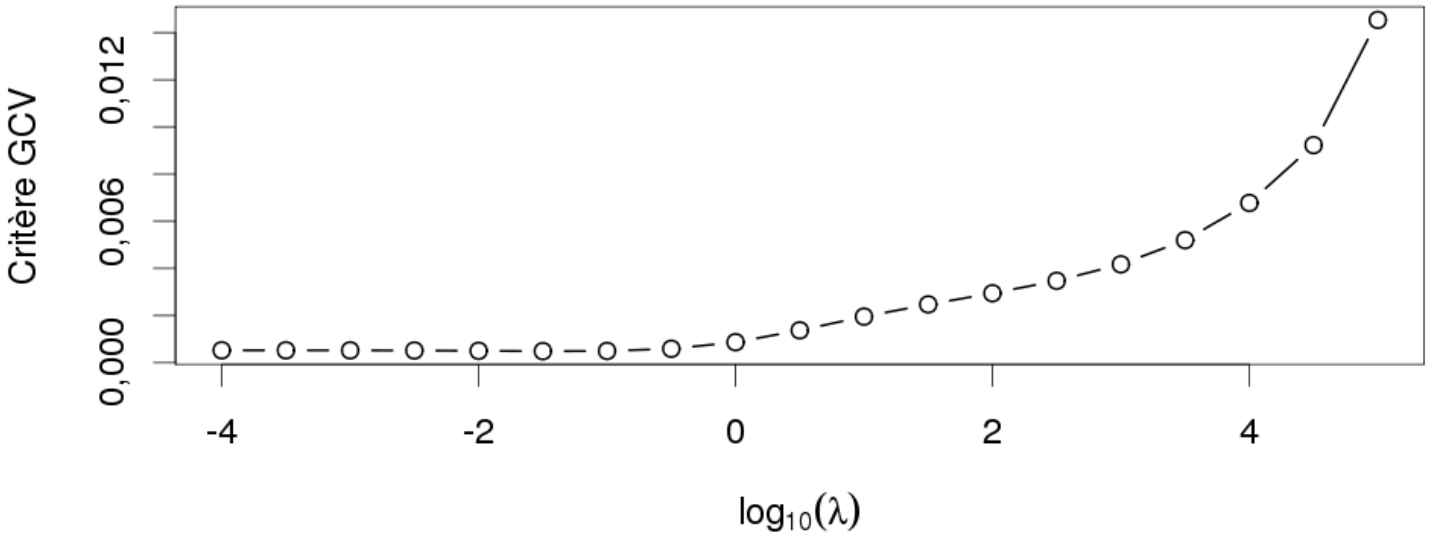
Guide pour choisir λ

Le critère de validation croisée généralisé (`GCV`, *generalized cross-validation*) peut être utilisé pour guider le choix de λ.
Ce critère est défini par :

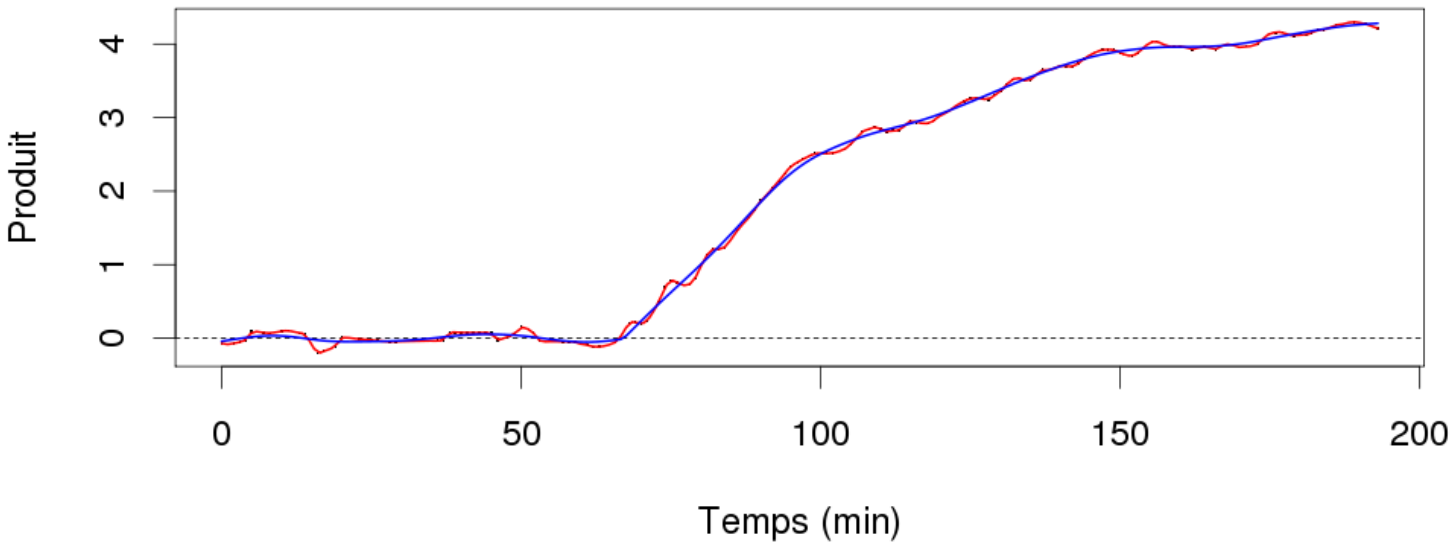
$$\mathrm{GCV}(\lambda) = \frac{n}{n - df(\lambda)} \frac{\mathrm{SSE}}{n - df(\lambda)}$$

où, `SSE` représente la somme des erreurs au carré (*sum of squared errors*) et où $df(\lambda)$ désigne le nombre « équivalent » de degrés de liberté, c'est un des éléments de l'objet de classe `fdSmooth` retourné par la fonction `smooth.basis`.

GCV pour la raffinerie (1)



GCV pour la raffinerie (2)



En rouge, le meilleur au sens du GCV ($\log \lambda = -1{,}5$) en bleu, mon choix ($\log \lambda = 3{,}5$).

Choix de λ pour des données périodiques

Voir la deuxième démo...

Autres cas...

Une des originalités très utiles de `fda` est qu'il permet d'estimer des fonctions :

- *positives* ou *négatives* avec `smooth.pos`,
  $y_i = \exp\left(\sum_{j=1}^{K} c_j \phi_j(t_i)\right) + \epsilon_i$;
- *monotones* avec `smooth.monotone`,
  $y_i = \beta_0 + \beta_1 \int_{t_0}^{t_i} \exp\left(\sum_{j=1}^{K} c_j \phi_j(u)\right) du + \epsilon_i$;
- *densités de probabilité* avec `density.fd` ;
- voir la troisième démo...

Sommaire

Statistique descriptive fonctionnelle : fonction moyenne

À un échantillon $\{x_1,\ldots,x_N\}$ de fonctions (ajustées à des données), on associe la fonction moyenne empirique :

$$\mu_x(t) = N^{-1} \sum_{i=1}^{N} x_i(t)\,.$$

On calcule cette quantité avec la méthode `mean` :

```
mean.logprec <- mean(logprecFitL[[bestIdx]]$fd)
```

Statistique descriptive fonctionnelle : fonctions variance et écart type

On associe de même une fonction variance empirique :

$$\sigma_x^2(t) = (N-1)^{-1} \sum_{i=1}^{N} (x_i(t) - \mu_x(t))^2$$

et une fonction écart type :

$$\sigma_x(t) = \sqrt{\sigma_x^2(t)}\,.$$

Cette dernière est calculée par la fonction `std.fd` :

```
std.logprec <- std.fd(logprecFitL[[bestIdx]]$fd)
```

Statistique descriptive fonctionnelle : fonction covariance

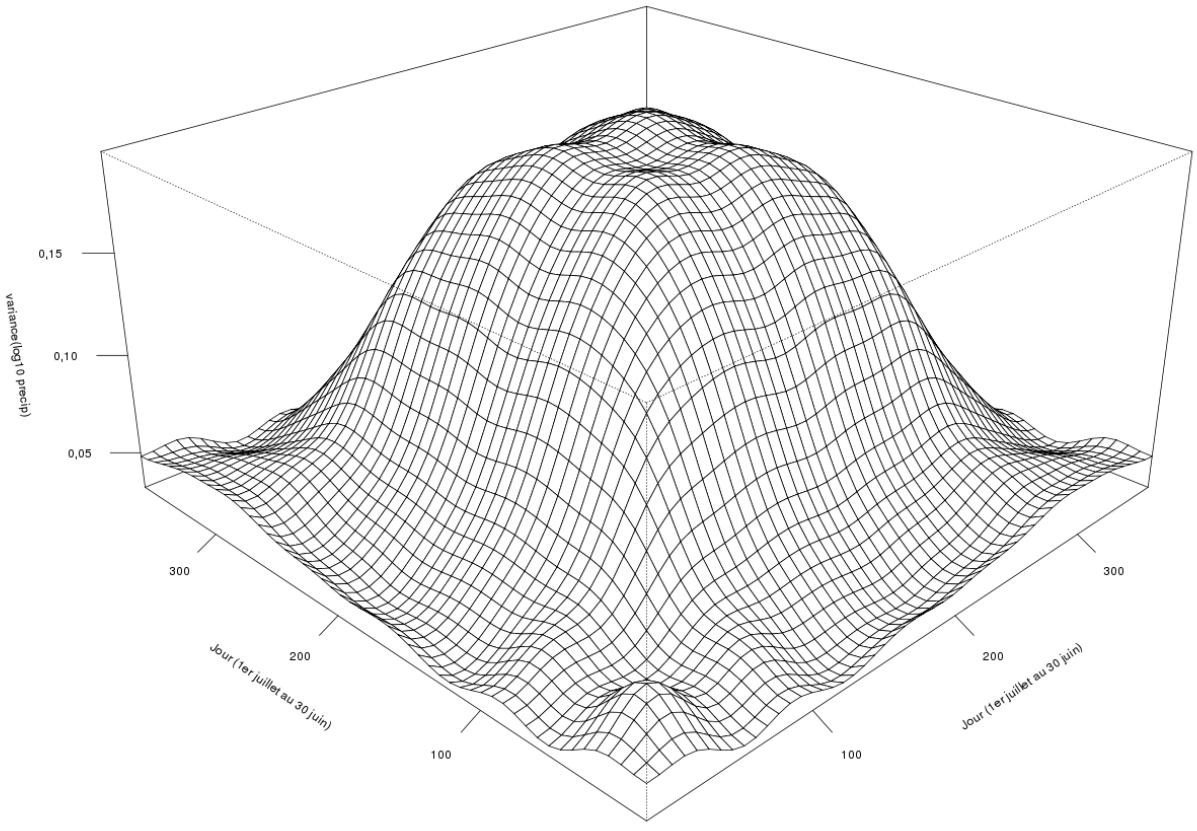
La fonction covariance $\nu(s,t)$ quantifie l'écart à la moyenne de l'élément x_i aux temps s et t. Elle est estimée par :

$$\nu(s,t) = (N-1)^{-1} \sum_{i=1}^{N} (x_i(s) - \mu_x(s))(x_i(t) - \mu_x(t)) .$$

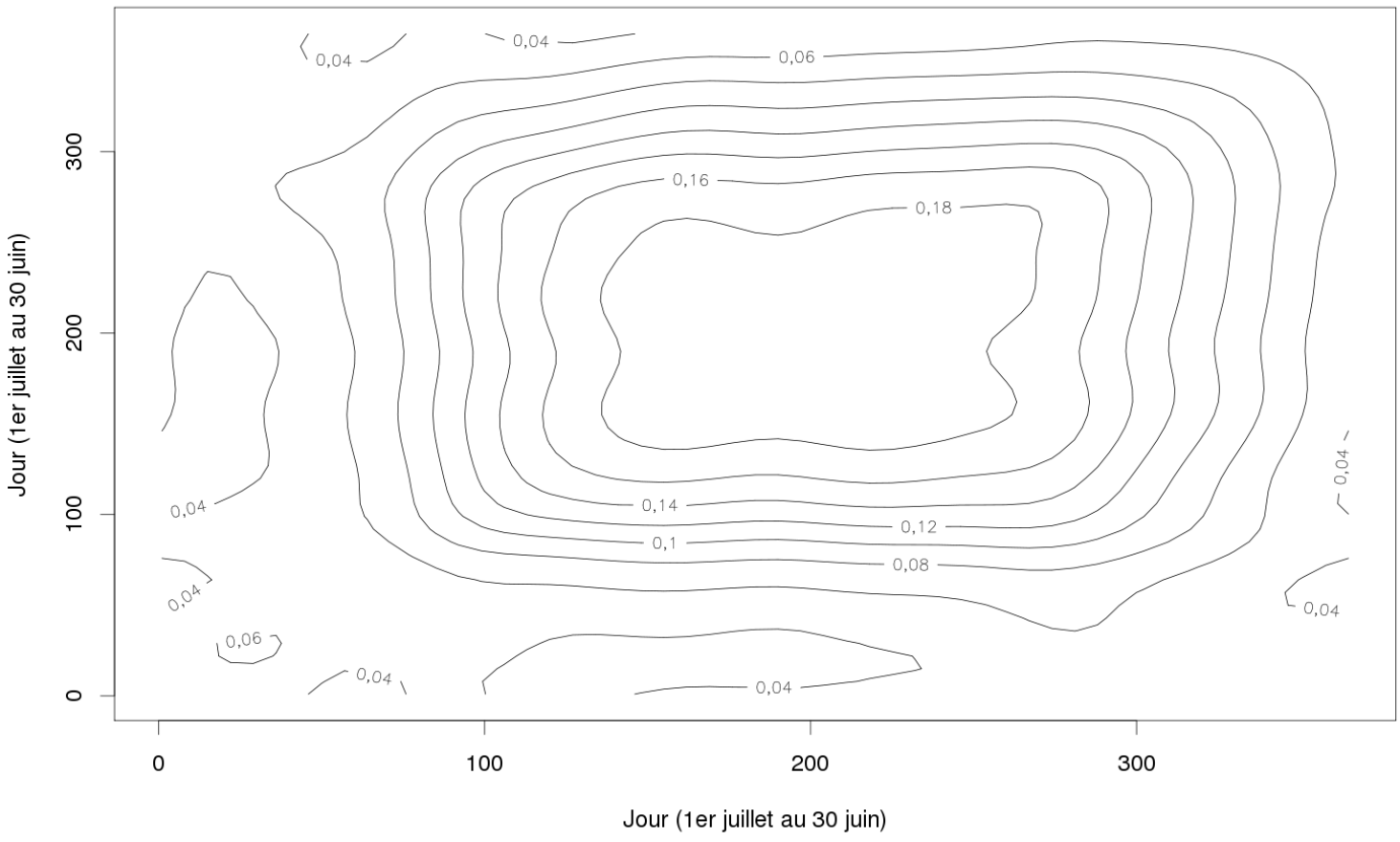
Elle est calculée par la fonction `var.fd` :

```
nu.logprec <- var.fd(logprecFitL[[bestIdx]]$fd)
```

Statistique descriptive fonctionnelle : fonction covariance



Statistique descriptive fonctionnelle : fonction covariance



Analyse en composantes principales fonctionnelle (1)

L'analyse en composantes principales s'étend au cas fonctionnel en faisant intervenir des concepts d'analyse fonctionnelle mathématique comme le produit interne ou produit scalaire, généralisation aux espaces de fonctions du produit scalaire euclidien. Pour ξ, fonction (convenablement définie) et x_i élément de notre échantillon fonctionnel, on définit le produit scalaire, $\langle \xi, x_i \rangle$, par :

$$\langle \xi, x_i \rangle \equiv \int \xi(t) x_i(t) dt \,.$$

Deux fonctions ξ et ψ sont orthogonales si : $\langle \xi, \psi \rangle = 0$. En somme, avec quelques précautions, on généralise à des espaces de fonctions l'intuition que nous avons développée sur les espaces euclidiens.

Analyse en composantes principales fonctionnelle (2)

Ainsi, la norme ou longueur d'une fonction ξ est-elle donnée par : $\|\xi\| = \sqrt{\langle \xi, \xi \rangle}$.
La première composante principale est alors définie comme la fonction ξ qui maximise :

$$\sum_{i=1}^{N} \langle \xi, (x_i - \mu_x) \rangle^2$$

tout en satisfaisant la contrainte :

$$\|\xi\| = 1 \,.$$

Les composantes successives sont obtenues en projetant dans le sous espace orthogonal à la dernière composante obtenue.

Analyse en composantes principales fonctionnelle (3)

La fonction `pca.fd` fait le travail pour nous. Et les fonctions qui lui sont associées permettent d'explorer le résultat comme le montre la démo suivante...