Master 2 Econométrie Statistique

Data Science for Social Sciences (International Track)

Fourier and B-Splines

Python's scikit-fda vs R's fda

Matthieu Donati Arranz

August 25, 2024

School: **Université Toulouse Capitole - Toulouse School of Economics**,

2 rue du Doyen Gabriel Marty 31042 Toulouse

1 Table of Content

Contents

2 List of references

List of Figures

List of Tables

3 Abstract

With Pr. Anne-Ruiz Gazen's help, I worked on a subject about the Python package *scikit-fda* and its difference with the R's package *fda* more commonly used for Functional Data Analysis. We will see that they can be some differences in the results depending on which package is used.

4 Introduction

The R package fda is the most widely used tool for Functional Data Analysis (FDA). However, some individuals prefer to code in Python. In this paper, we will examine the Python package *scikit-fda* and compare it with the R package *fda*, which is more commonly used for FDA. Specifically, the study focuses on functional data analysis (FDA) regression, roughness-penalised FDA regression, and Generalised Cross-Validation (GCV) scores, using Fourier and B-splines as bases. We will explore how the results can vary depending on which package is used.

Before starting the comparisons, some background on Functional Data Analysis will be provided. The context, data, and computational methods being compared are also presented.

5 Description of the subject

5.1 Reminder on Functional data analysis

FDA is similar to linear regression but applied to functions rather than scalar variable. In a regular linear regression, the explanatory variables X have dimensions $N \times P$, where N is the number of observations and P is the number of predictors, and the response variable Y has dimension N, representing the variable of interest. For FDA, the concept is similar, but the data at hand represents functions:

- $t_i \in \mathbb{T}$ the domain of the function, $i \in [\![1, N]\!]$

- $y_i \in \mathbb{Y}$ the co-domain, $i \in [\![1, N]\!]$
- ϕ an Infinite (Schauder) basis from $\mathbb{T}$ to $\mathbb{Y}$.
- ϕ_{prm} Truncation for the computer representation of ϕ.
- L a Loss function from $\mathbb{R}^{n_{basis}}$ to $\mathbb{R}^+$ where $n_{basis} = dim(\phi_{prm})$

This paper will focus only on functions from $\mathbb{T} \subset \mathbb{R}$ to $\mathbb{Y} \subset \mathbb{R}$.

Most Common truncated bases for FDA from $\mathbb{R}$ to $\mathbb{R}$ include:
[4, See Section 3]

- **Monomial basis**:
$$\phi_{prm}(t) = \sum_{i=0}^{n_{basis}-1} a_i \cdot t^i$$
- **Fourier basis**:
$$\phi_{prm}(t) = a_0 + \sum_{k=1}^{\frac{n_{basis}-1}{2}} \left(a_k \cos\left(\frac{2\pi k t}{T}\right) + b_k \sin\left(\frac{2\pi k t}{T}\right)\right)$$
- **Polynomial splines**: Polynomial splines are piecewise polynomials defined over a sequence of knots. A common choice is cubic splines, which are piecewise cubic polynomials that are twice continuously differentiable. These functions can be represented by a basis of truncated polynomial functions.
$$s(x) = \sum_{j=0}^{k} b_j \cdot x^j + \sum_{l=1}^{g} c_l \cdot (x - \lambda_l)_+^k$$
- **B-spline basis**: B-spline basis functions are a special case of polynomial splines. Zero-degree (piecewise constant) B-splines:
$$j \in [\![1, n_{basis}]\!] B_{j,0}(t) = \begin{cases} 1 & \text{if } t_j \leq t < t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

$$j \in [\![1, n_{basis}]\!]$$

Higher-degree B-splines ($k \geq 1$):

$$B_{j,k}(t) = \frac{t - t_j}{t_{j+k} - t_j} B_{j,k-1}(t) + \frac{t_{j+k+1} - t}{t_{j+k+1} - t_{j+1}} B_{j+1,k-1}(t)$$

- **Finite element basis**: Finite element basis functions are typically piecewise linear or higher-order polynomials defined over subintervals of the domain.

  Piecewise linear (first-order) finite element basis functions:

$$\phi_j(t) = \begin{cases} \frac{t - t_{j-1}}{t_j - t_{j-1}} & \text{if } t_{j-1} \leq t < t_j \\ \frac{t_{j+1} - t}{t_{j+1} - t_j} & \text{if } t_j \leq t < t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

  Thus, the finite element basis representation is:

$$\phi_{prm}(t) = \sum_{j=1}^{n_{basis}} a_i \phi_j(t)$$

Only B-Splines and Fourier bases will be studied in this report.
The goal is to create a function that would be "close" to the "true" one generating our data . This function would be a sum of weighted functions from our truncated bases:

$$\hat{y} = c' \phi_{prm}(t) \text{ where } \hat{y} \in \mathbb{Y}, t \in \mathbb{T}$$

The goal is to find a vector of coefficients $c \in \mathbb{R}^{n_{basis}}$ that minimises the Loss function.

$$y \approx c' \phi_{prm}(t) \text{ where } y, \in \mathbb{Y}, t \in \mathbb{T}$$

Usually loss functions penalise the distance between the variable of interest y and $c'\phi_{prm}(t)$ but reward the smoothness of $c'\phi_{prm}(t)$. [5, see Section 5] Different ways to penalise distance and roughness exist, leading to various loss functions and therefore various optimal coefficients and representation.

$$\text{Distance Penalty} = \|y(t) - c(t)\|_{p,T} = \begin{cases} \left(\int_T |y(t) - c'\phi_{prm}(t)|^p \, dt\right)^{1/p} & \text{if } p \in \mathbb{N}, \\ \max\limits_{t\in[a,b]}(|y(t) - c'\phi_{prm}(t)|) & \text{if } p = \infty \end{cases} \tag{1}$$

But since the true function is unknown, the Distance Penalty has to be replaced by :

$$\text{Distance Penalty} = \|y - c(t)\|_p = \begin{cases} \left(\sum\limits_{i\in[\![1,N]\!]} |y_i - c(t_i)|^p\right)^{1/p} & \text{if } p \in \mathbb{N}, \\ \max\limits_{i\in[\![1,N]\!]}(|y_i - c(t_i)|) & \text{if } p = \infty, \end{cases} \tag{2}$$

where t takes its values from $\mathbb{T}$ and y from $\mathbb{Y}$

$$\text{Roughness Penalty}(c) = \lambda \int_a^b c'L\phi_{prm}(t)(c'L\phi_{prm}(t))'dt \quad \text{where } L \text{ is a differential operator.} \tag{3}$$

$$\text{Roughness Matrix} = R = \int_a^b L\phi_{prm}(t)(L\phi_{prm}(t))'dt \quad \text{of dimension } p \cdot p \tag{4}$$

$$\text{Roughness Penalty}(\lambda, c) = \lambda c' \cdot R \cdot c \tag{5}$$

The Goal is to compute...

$$C_\lambda = \underset{c\in\mathbb{R}^p}{\text{Argmin}} \left(\text{Distance Penalty}(c) + \text{Roughness Penalty}(\lambda, c)\right) \tag{6}$$

Very often $p = 2$ and L is the second derivative, but other roughness penalties can be applied [5, *p.84*] . When it is the case, $C = (\phi_{prm}(t)'\phi_{prm}(t) + \lambda \cdot R)^{-1}\phi_{prm}(t)'y$. Only this penalty will be studied in this paper.

General Cross-Validation (GCV) is commonly used to determine the right amount of smoothing [5, *p.97*]:
It helps in choosing the best λ value. The goal is to minimise:

$$\text{GCV} = \frac{SSE_\lambda}{\left(1 - \frac{df_\lambda}{n}\right)^2} \tag{7}$$

Where SSE is the sum of squared errors for the rough penalised regression on the entire sample, and df is the degree of freedom. Several ways to compute the degree of freedom exist, the most famous being $df_\lambda = \text{Trace}(H_\lambda)$ where $H_\lambda = \phi_{prm}(t)(\phi_{prm}(t)'\phi_{prm}(t) - \lambda \cdot R)^{-1}\phi_{prm}(t)'$. This degree of freedom will be used in this paper.

5.2 Presentation of the data

Two datasets will be used in this paper:

5.2.1 Cyclic Artificial Data

A dataset was generated artificially from a pre-defined function. There is a "true" curve and a "noisy" curve.

$$True = f(t)$$

$$Noisy = True + \epsilon = f(t) + \epsilon \text{ where } \epsilon \text{ follow a random low}$$

- Generated on Python with numpy
- $f(t) = 1.5 \cdot \cos(20\pi t) + 2 \cdot \sin(4\pi t)$
- t is 10001 observations equally spaced from 0 to 10
- f$\epsilon \sim \mathcal{N}(0, 1.4^2)$
- Norm of $True = 5.59017$ for norm L_2

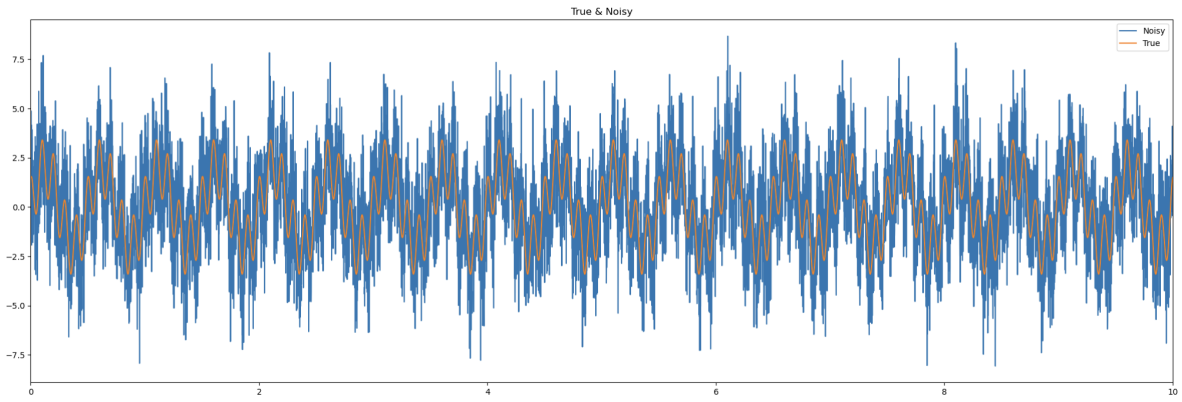

Figure 1: Cyclic Data, Noisy & True

5.2.2 Real life Data: Daily Temperature in Vietnam from 1981 to 2023

The real-life data consists of average, minimum, and maximum temperatures recorded daily at different Vietnamese locations from 1981 to 2023.
There is no distinction between true and noisy data for this dataset as it is not artificially generated. From this dataset, only one function is considered:

- d January 1st to December 31 from 0 to 1.
  (February 29 was excluded for convenience)
- $avg = avg(temp(d))$ the mean average temperature for a day of the year, across years and locations.
- Norm of $avg = 24.043202$ for norm L_2

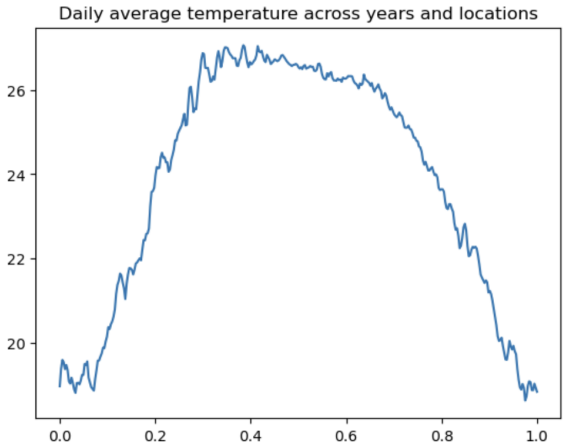

Figure 2: Daily average Temperature across location and years

5.3 Illustration of the Scikit-fda package in Python

5.3.1 Illustration on cyclic artificial data

For this subsection, the data is artificially generated; therefore, the true value of y is available. An interesting distance to compute is the distance between the *true* curve and the truncated basis representation of the noisy curve. With real-life data, this is usually impossible because the true function is not available (otherwise, FDA would not be necessary).
Compared bases:

- Fourier with parameters $n_{\text{basis}} = 11$ and period $= 0.5$, $F(11, 0.5)$
- B-Splines with parameter $n_{\text{basis}} = 101$, $BS(101)$

Loss function without Roughness penalty (2)

Distance to	**Fourier ($n_{\text{basis}} = 11$, period = 0.5)**	**B_splines ($n_{\text{basis}} = 101$)**
True	0.162	3.379
Noisy	5.34	6.29

Table 1: Distances using Fourier and B-splines Methods

One might think that Fourier performs well. But there is a reason why. The chosen parameters generate the elements of the function. Indeed the 10^{th} function of the Fourier basis when period is 0.5 is $\cos\left(\frac{2\pi 5x}{0.5}\right) = cos(20\pi x)$ and the 3^{rd} is $\sin\left(\frac{2\pi 1x}{0.5}\right) = \sin(4x)$.

With a real life dataset, it is rare to have a perfectly cyclic functional data, and even when it is the case, to find the right parameters that can generate all the elements of the function is even rarer. The following graph illustrate this problem. A grid search is performed on Period and n_{basis}, with period going from 0 to 1 with steps of size 0.01 and n_{basis} going from 1 to 21 2 by 2.

For this grid search the following parameters generate the function's elements:

- period $= 0.5$ and $n_{\text{basis}} \geq 11$
- period $= 1$ and $n_{\text{basis}} = 21$

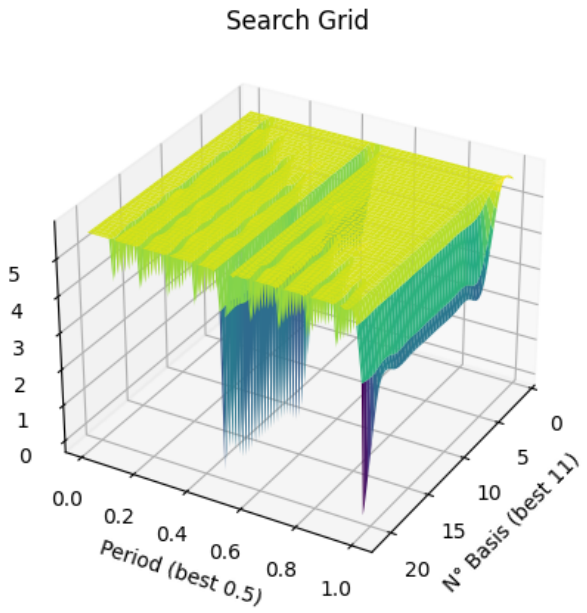

Figure 3: Grid Search on Fourier

A slight change from the parameters generating the function's elements drastically change the loss. The loss goes from close to 0 to approximately 5 very fast which is much for than what we obtained for B-splines(101): 3.38. The best parameters are unsurprisingly period $= 0.5$ and $n_{\text{basis}} = 11$ since it is the basis with the lowest number of functions among bases generating the function's elements (which lowers overfitting) on $Noisy$.

Since the loss depends heavily on whether the parameters generates the function's elements or not, an idea could be to take a very large period and an even larger n_{basis} to be sure not to miss the true elements. The problem is that it creates many variables which generates a curse of dimensionality because of$Noisy$'s noisy nature.

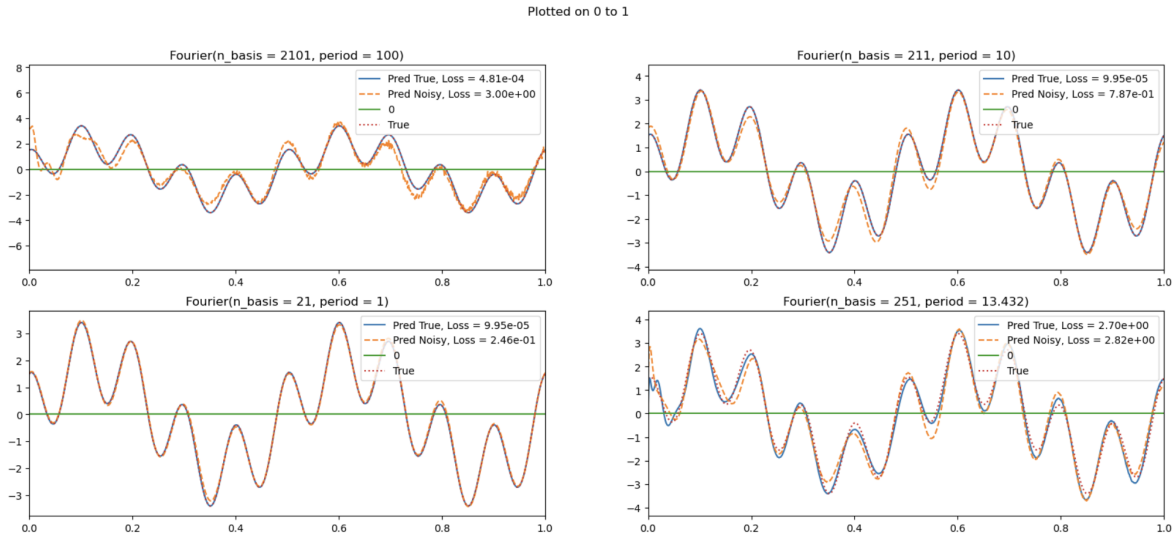

Figure 4: Fourier performances depending on n_{basis} and *period*

$F(2101, 100)$, $F(211, 10)$, and $F(21, 1)$ generate the function elements, but the fewer n_{basis}, the better, as we can see in Figure 4, with their respective losses being 2.66, 0.79, and 0.25.

The parameters $n_{\text{basis}} = 251$ and $period = 13.432$ with a loss of 2.82 do not generate the function elements. However, having both a high n_{basis} and a high period generates two function elements that are close to the real ones.

Therefore, if the basis generates the function elements, having a lower number of n_{basis} is preferable. However, when this is not achievable, it's important to find a balanced number of bases. If the number of bases is too low, there is a high chance that the elements will not be adequately generated, leading to estimates that are far from accurate (as seen with approximately 5 for GS earlier). On the other hand, if n_{basis} is too high, even if it captures the true function elements, the loss can be significant if the data is very noisy. In such cases, it is better to choose a balanced number of bases, even if it does not perfectly generate the function elements. For example, $Loss(F(2101, 100)) = 3$, whereas $Loss(F(251, 13.432)) = 2.7$.

Grid Search for B-Splines

On figure 5) the loss evolution resembles to a step function, with several plateaux and

some sudden drops. On this figure the B-Splines are computed on the noisy data. This graph display observable loss and true loss depending on the number of n_{basis}. The orange curve represents the observable loss, which is the distance between the noisy B-Splines and the noisy curve. The blue curve represents the true loss, which is the distance between the noisy B-Splines and the True curve (not observable in real life).

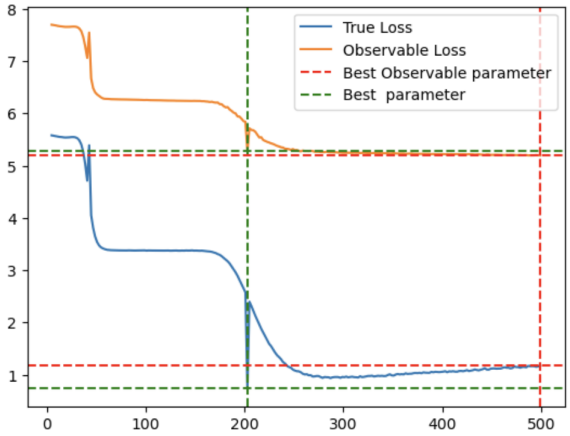

Figure 5: Bsplines Grid Search Cyclic Data

3 Remarks:

1) For $n_{basis} = 203$ B-Splines performs very well.

distance to $True$ for $n_{basis} = 202$ is 2.538 computed on noisy data

distance to $True$ for $n_{basis} = 203$ is 0.732 computed on noisy data

distance to $True$ for $n_{basis} = 204$ is 2.444 computed on noisy data

Distance to	$BS(202)$	$BS(203)$	$BS(204)$
True	2.538	0.732	2.444

Table 2: B-Splines Loss evolution around $n_{basis} = 203$

$n_{basis} = 203$ performs very well because for this number of basis, $a = 0$, $b = 10$, and each knot almost corresponds to a local minimum and maximum of the function. It is rare for real life data to find knots that perfectly fit the data automatically, the same way it is rare to use a Fourier basis that generates the function elements.

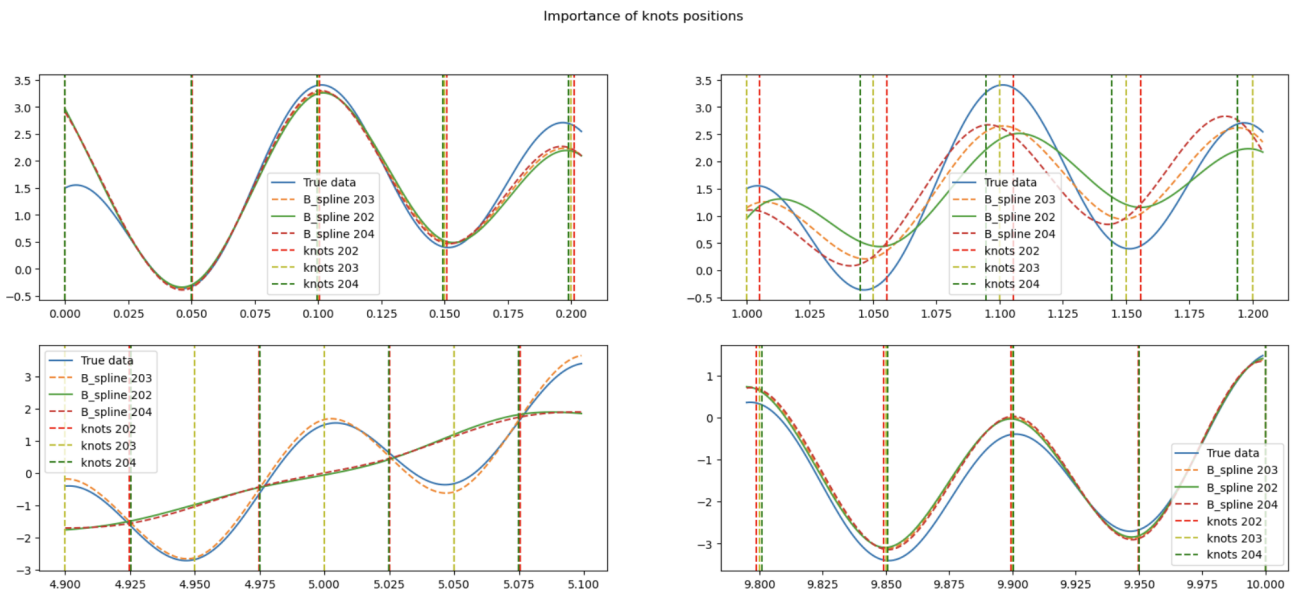

Figure 6: Bsplines performances depending on n_{basis} and knots positions

2) There is a plateau from around 61 to around 171. Location of these knots do not increase knowledge on the data.

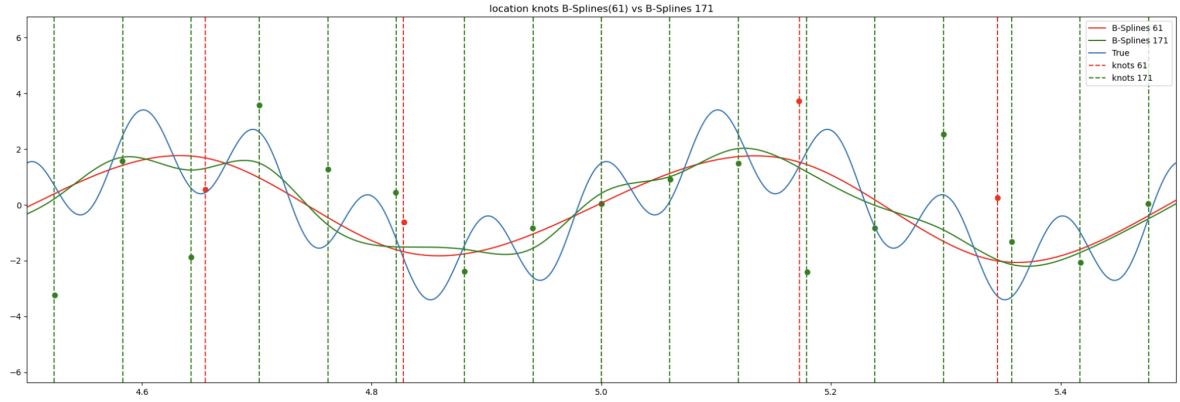

Figure 7: Knots positions B-Splines(61) vs B-Splines(171)

3) Once again, overfitting penalises the results. One way to address this issue is to penalise roughness. This is where the roughness-penalised fitting criterion comes into play.

Since Overfitting penalises the result, let's test if penalising roughness helps:

Loss function with Roughness penalty:

Different values of lambda were tested on different basis :

λ	$F(11, 0.5)$	$F(211, 10)$	$F(97, 7.43)$	$BS(501)$
10^{-8}	**+1.887e-03**	-5.464e-03	-7.485e-06	-1.380e-01
10^{-7}	+6.574e-02	-5.259e-02	-7.483e-05	-1.306e+00
10^{-6}	+5.076e+00	**-3.256e-01**	-7.463e-04	-8.990e+00
10^{-5}	+1.976e+02	+1.123e+01	-7.265e-03	**-1.793e+01**
10^{-4}	+1.169e+03	+1.711e+02	-5.721e-02	+8.075e+01
10^{-3}	+1.855e+03	+3.053e+02	**-1.335e-01**	+1.700e+02
0.01	+2.044e+03	+3.423e+02	+2.808e+00	+1.950e+02
0.1	+2.772e+03	+4.907e+02	+3.018e+01	+2.911e+02
1	+3.278e+03	+5.939e+02	+4.940e+01	+3.589e+02
10	+3.353e+03	+6.091e+02	+5.234e+01	+3.705e+02
10^2	+3.361e+03	+6.107e+02	+5.268e+01	+3.725e+02
10^3	+3.362e+03	+6.108e+02	+5.273e+01	+3.731e+02

Table 3: Change of distance to true data after roughness penalty and λ Values

For these Fourier bases, results were not impressive. Scikit-fda struggles for $Fourier(2101, 100)$ to compute the representation with a penalty. For B-Splines(501), results were good for some λ values especially $\lambda = 10^{-5}$, the distance to the true data decreased by 17.93% (Tab. 3).

In real life, the distance to true data is not available since the true data is not available. Therefore, to select the penalty value λ, the General Cross Validation is criterion is used. Let's test for this data, bases, and λ, if GCV is efficient with scikit-fda:

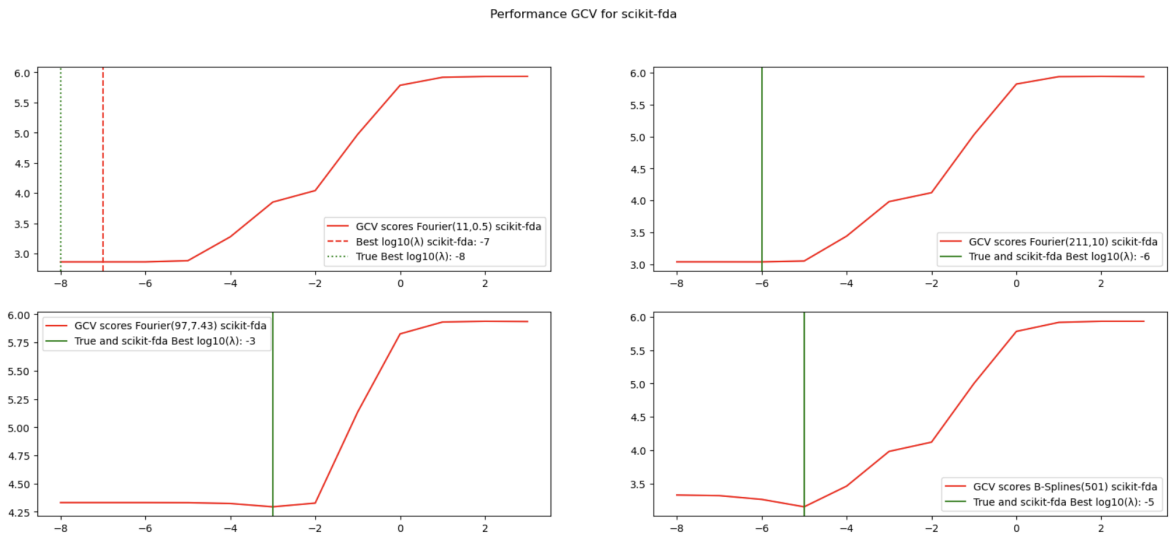

Figure 8: GCV accuracy for Scikit-fda

Apart from Fourier(11,0.5) (but still very close), scikit-fda's GCV selected the best lambda value for all the tested bases.
An attempt to perform a GCV grid search was made but was unsuccessful.

5.3.2 Illustration on the Vietnam climate data

Studied bases:

- $Fourier(n_{basis} = 301, period = 1),\ F(301, 1)$
- $Fourier(n_{basis} = 101, period = 1),\ F(101, 1)$
- $Fourier(n_{basis} = 31, period = 1),\ F(31, 1)$
- $B - Splines(n_basis = 25),\ BS(25)$

Unlike with the artificial data, the Vietnam climate dataset does not have true or noisy curves; only the recorded data is available. Therefore, the true loss cannot be directly measured.

- In the first part basis representation will be made without roughness penalties. The loss functions will just be equal to the distance between the recorded data and the basis representation estimates.
- In a second part a roughness penalty will be added.

- In the last part the goal will be to determine which penalty strength *lambda* is the best for each one of these bases using GCV.

Estimates without penalty

Distance measure:

$$||avg - \phi_{prm}(j)||_{2,0 \text{ to } 1} \text{ where } \phi_{prm} \text{ is the FDA regression on } avg$$

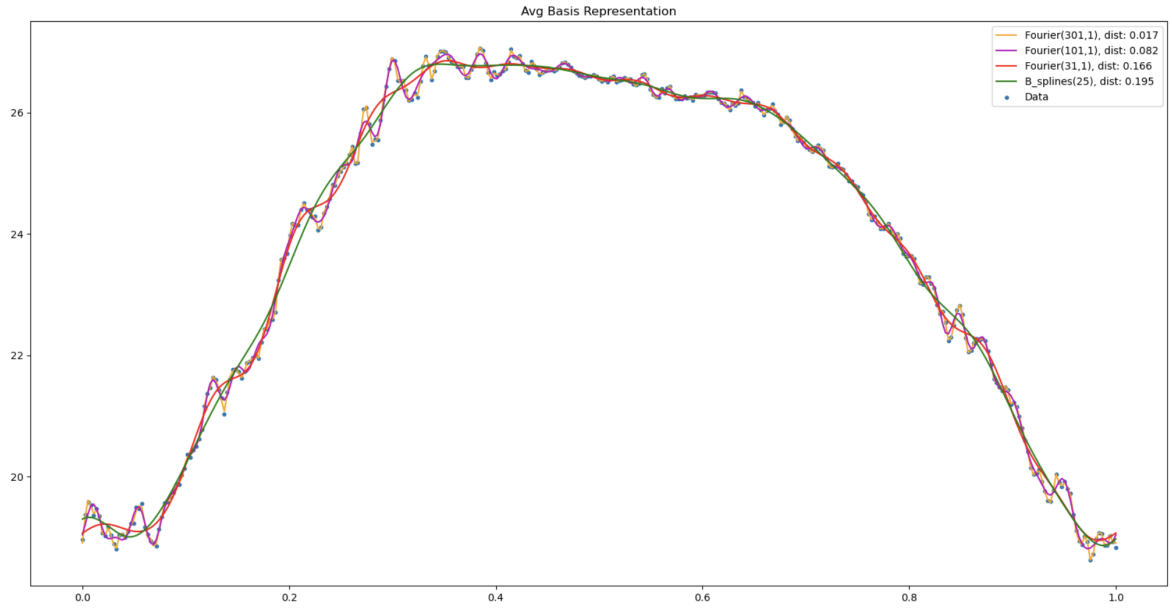

Figure 9: Several *avg* bases representations

For Fourier when n_{basis} increases the Loss decreases. It is not surprising since Period is fixed. All the elements of Fourier(31,1) are included in Fourier(101,1) which is included in Fourier(301,1).

Let's perform a Grid Search first on Fourier

Remark on Fourier: Even if it the grid search is in 3d due to the fact that there are 2 parameters, when $n_{basis}1 < n_{basis}2$ if $per1 = per2 = per$ then $Loss(Fourier(n_{basis}1, per)) > Loss(Fourier(n_{basis}2, per))$. Therefore, in order to obtain the best parameters for this loss function, only keeping the biggest n_basis of the 3d Grid Search and testing different period values would be enough. Another remark on Fourier, if $n_{basis} = n$ then Loss function in theory loss function should be equal to 0 if the Fourier Bases are

orthonormal.

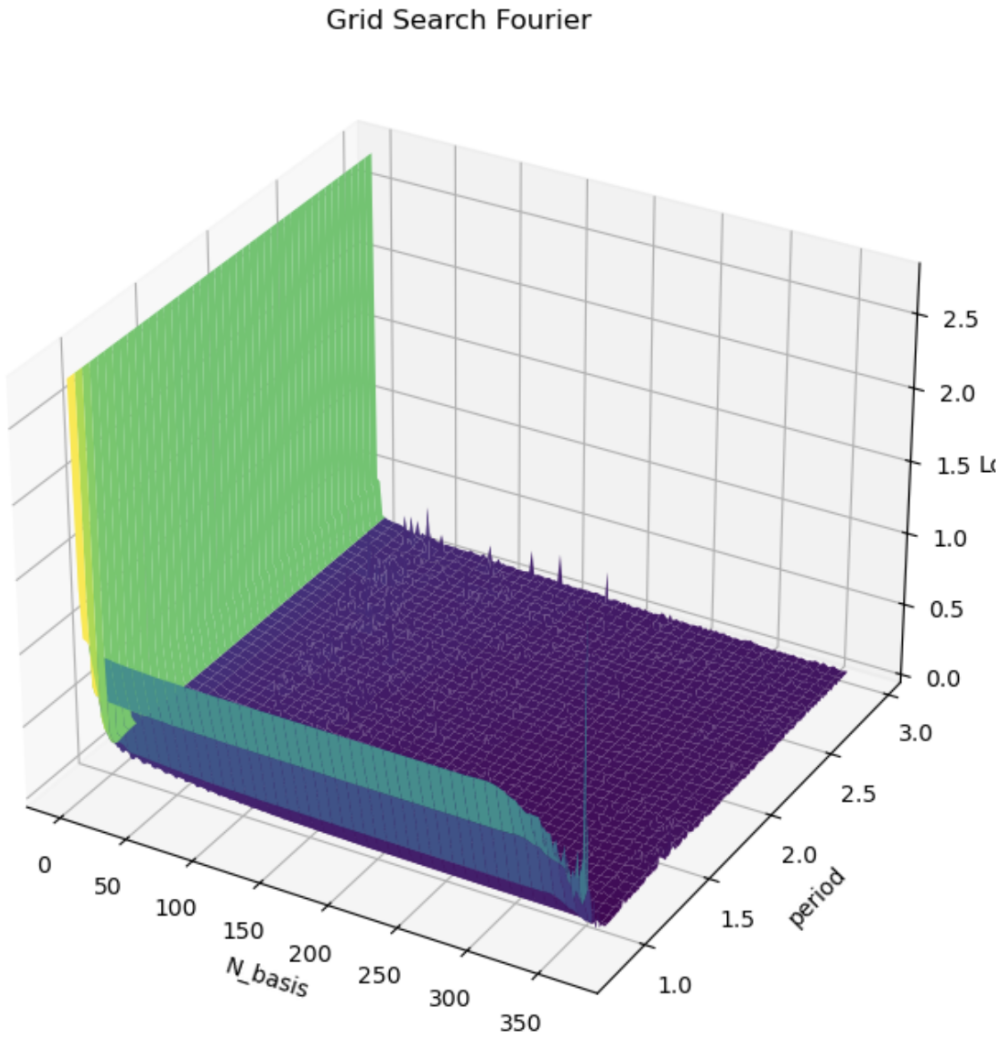

Figure 10: GS Fourier

It can be difficult to make them with Figure 10 only, but several remarks can be made

- When n_{basis} is equal to 365, the loss function is sometimes fairly different from 0 which is not normal in theory if the base was orthonormal. But the Fourier Bases are orthonormal only if the function is defined from 0 to per. For instance when period $= 0.8$, $\Phi_{F(365,0.8)}(j)$ might no be full rank (rank is 330 not 365), which causes problems to compute coefficients.

- $Loss(Fourier(n1_{basis}, per)) < Loss(Fourier(n2_{basis}, per))$ whereas $n1_{basis} < n2_{basis}$ sometimes, which is absurd since all the bases elements from the $Fourier(n_{basis}1, per)$

are included in $Fourier(n_{basis}2, per)$. For instance for period = 1.29, for n_basis equal to 11, 13, 15 loss are respectively 0.216455, 0.203420, 0.203425.

- Some regressions coefficients cannot be computed. Sckikit-fda struggles to invert some matrices, which is necessary to compute regression coefficients. It is the case for instance for Fourier(57, 2.63).

Computational limits can be a barrier.

Let's perform a Grid Search first on B-Splines

Since we fixed the knots position system and the B-Splines order, the B-Splines grid-search plotting is in 2d.

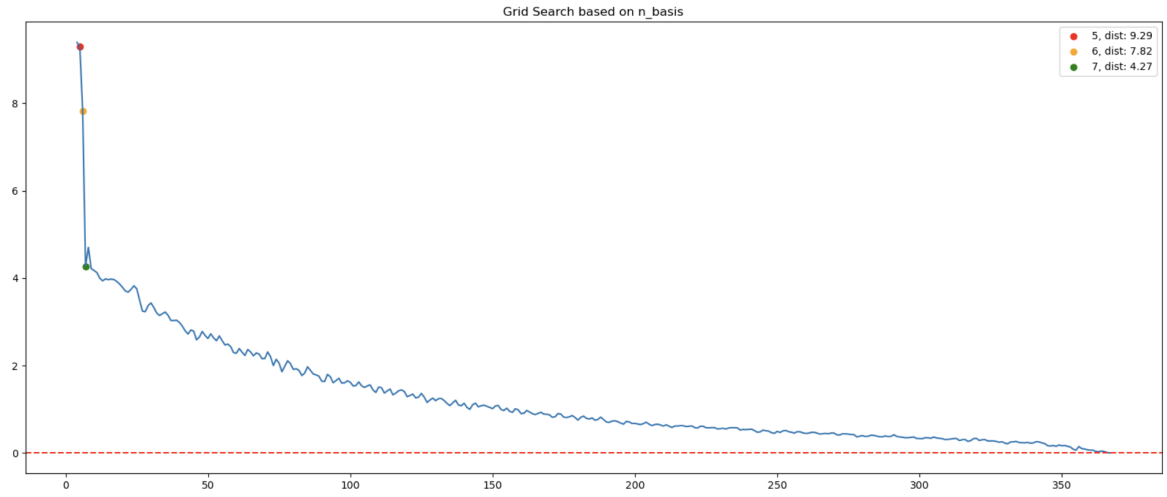

Figure 11: Grid Search B-Splines

Import gap are observed from $n_{basis} = 5$ to $n_{basis} = 6$ where Loss goes from 9.29 to 7.82 and equally from $n_{basis} = 6$ to $n_{basis} = 7$ where Loss decreases to 4.27. Then Loss decreases slower. Let's try to understand why.

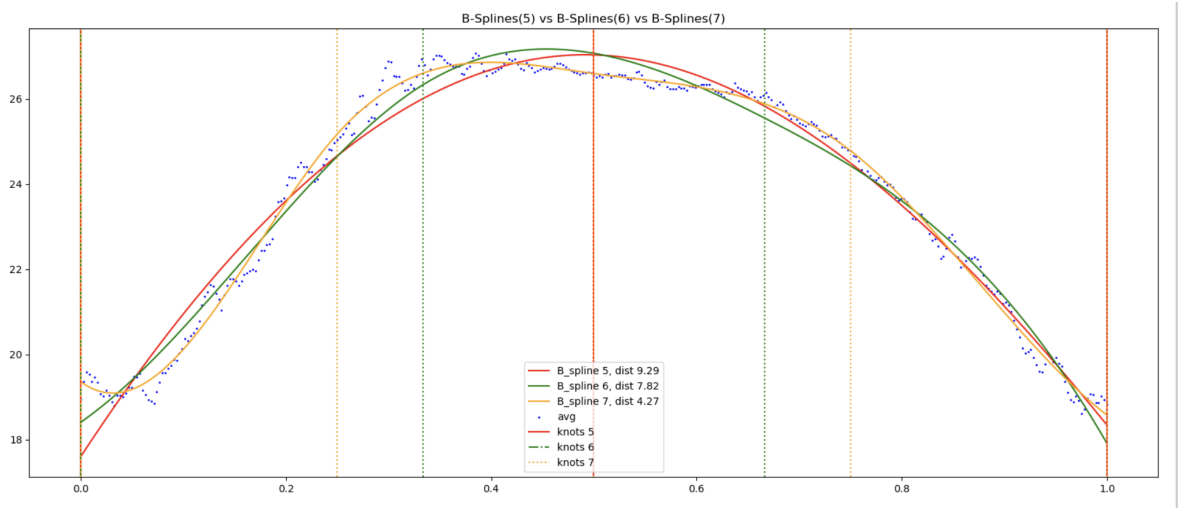

Figure 12: B-Splines(5) vs B-Splines(6) vs B-Splines(7)

Knots presence for *B-Splines*(6) in the middle of the left side and in the middle of the right side helps capturing the slightly asymmetric nature of avg. For *B-Splines*(7) knots are also present in each side middle which also helps capturing the asymmetric nature of avg. But additionally the one in the middle combined with the two former ones, allow to detect that *avg* is not totally unimodal but between unimodal and bimodal.

Roughness penalised fitting criterion

In the previous section, especially with B-Splines, it was highlighted that in general the bigger n_{basis} was, the closer the basis representation was to the data. But the bases representations where not smoothed and very close to an interpolation. This is usually not the goal in FDA when suspect the data to be noisy. The goal is to obtain a curve weights closeness to the data and smoothness. As shown in Equation (6), the optimisation problem involves balancing the distance and roughness penalties.

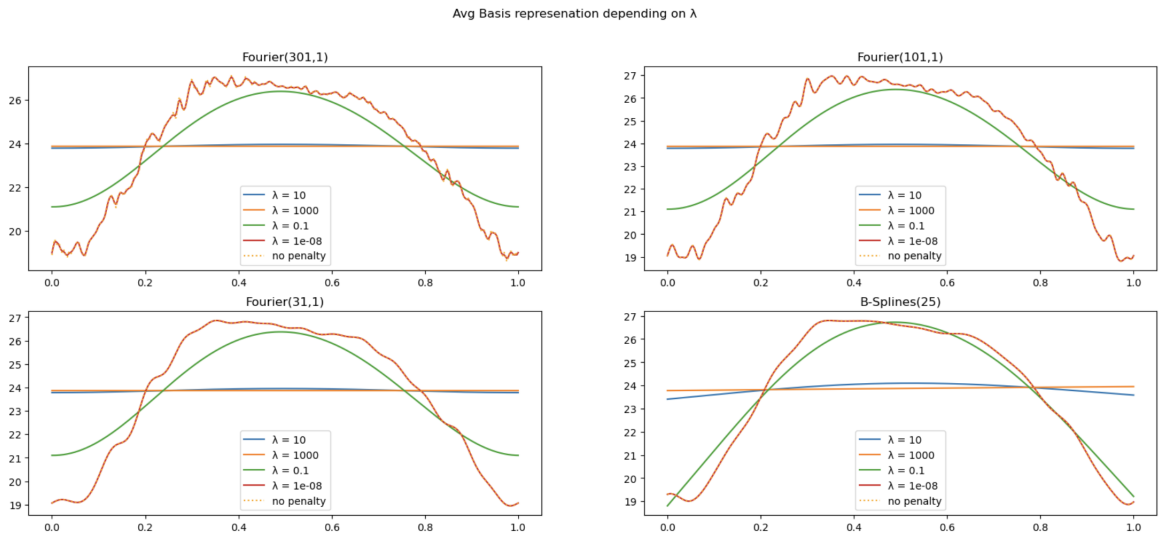

Figure 13: Roughness penalised fitting criterion

Remarks:

- Unsurprisingly, the bigger λ is, the smoother the basis representation are.
- When $\lambda = 1e-08$ there are barely any differences with the non-penalised regression.
- When λ is big the penalised regression is almost equal to a constant equal to the mean for Fourier bases.
- If I had to chose a λ among these ones, 0.1 would be my preference. It is not completely flat as 10 and 1000 but it is rather smoothed compared to $\lambda = 1e-08$ In the next section we will see that it is not the case when using GCV criterion.

Fourier(301,1), Fourier(101,1) and Fourier(31,1) when rounded to the 5th decimal, have very similar results:

$$\hat{avg} = 23.87471 + 0.05257 \cdot \sin(2\pi t) - 2.63296 \cdot \cos(2\pi t) - 0.04508 \cdot \sin(4\pi t) - 0.13936 \cdot \cos(4\pi t) - 0.00326 \cdot \sin(6\pi t) - 0.00136 \cdot \cos(6\pi t) + 6e-05 \cdot \sin(8\pi t) - 0.00062 \cdot \cos(8\pi t) + 0.00015 \cdot \sin(10\pi t) - 0.00026 \cdot \cos(10\pi t) + 6e-05 \cdot \sin(12\pi t) + 5e-05 \cdot \cos(12\pi t) + 4e-05 \cdot \sin(14\pi t) + 2e-05 \cdot \cos(14\pi t) + 1e-05 \cdot \sin(16\pi t) + 3e-05 \cdot \cos(16\pi t) + 3e-05 \cdot \cos(18\pi t) + 1e-05 \cdot \sin(20\pi t) + 1e-05 \cdot \cos(20\pi t) + 1e-05 \cdot \sin(22\pi t) - 1e-05 \cdot \cos(22\pi t) - 1e-05 \cdot \cos(24\pi t) - 1e-05 \cdot \cos(26\pi t)$$

The greater the inside coefficients are, the bigger the penalty will be. Indeed 26 is the biggest inside coefficients used (when coefficients are rounded to the 5th decimal). When the double derivative is applied is means the integral is multiplied by 26^2. It is here not worth using bigger than 26. When the penalty is not applied, even for $n_{basis} = 301$, most bases elements get a fairly big coefficient. For instance, the coefficient of $cos(300\pi)$ is 0.00058.

Let's now use generalised cross-validation criterion to determine which penalty for each one of these basis.

Generalised cross-validation criterion (7)

Let's now observe for bases Fourier(301,1), Fourier(101,1), Fourier(31,1) and B-Splines(25) the GCV scores:

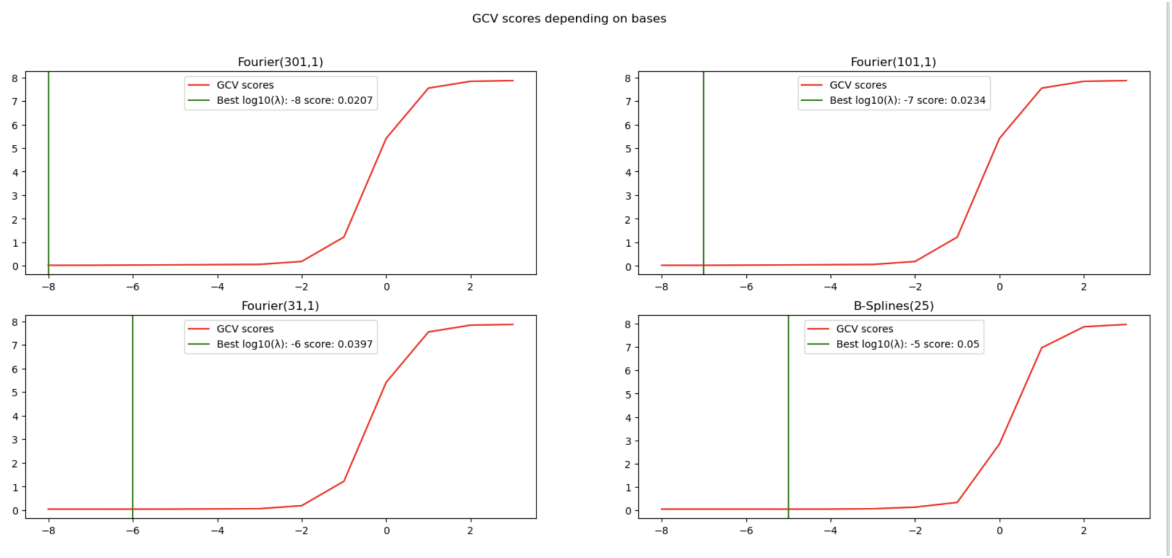

Figure 14: GCV

0.1 is the best λ for none of them. For $Fourier(301, 1)$ the best is -8 with a socre of 0.0207, which is the minimum therefore to continue reducing $\log_{10}(\lambda)$ might result in even better GCV scores. For $Fourier(101, 1)$ and $Fourier(31, 1)$, $B\text{-}Splines(25)$ the best $\log_{10}(\lambda)$ (scores) are respectively -7 (0.0234), -6 (0.0379) and -5 (0.05). From $\log_{10}(\lambda) = -8$ to $= -3$ GCV scores appear to be stable the 4 bases. Penalising Roughness is therefore not that much encouraged for these basis. Bases representations with these penalties: The difference is barely noticeable.

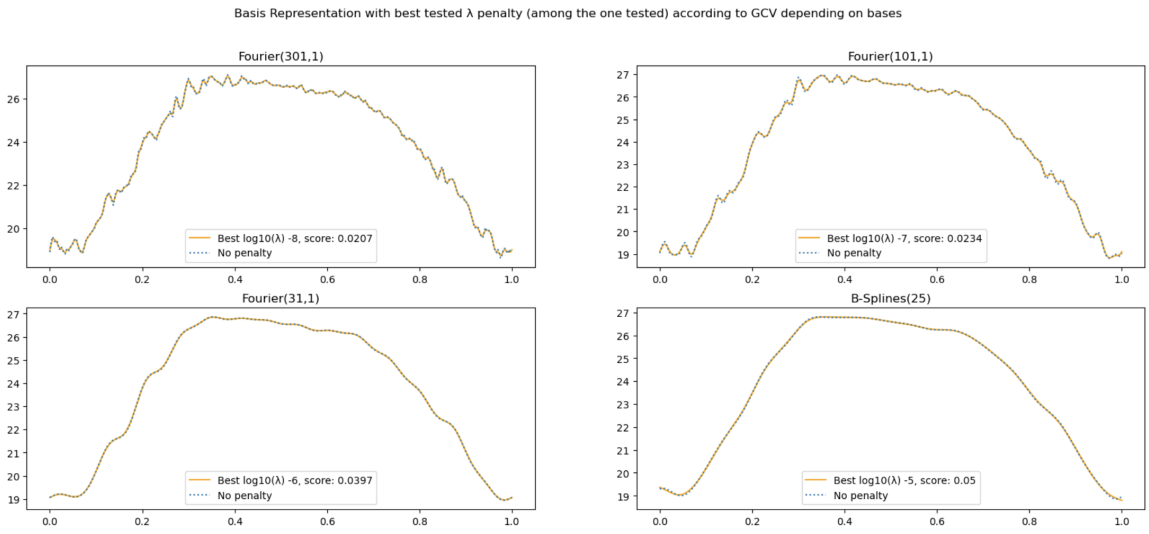

Figure 15: Penalised Basis Representation based on GCV criterion vs unpenalised

After performing a grid search for B-Splines, with n_{basis} from 4 to 368 and $log_{10}(\lambda)$ from - 10 to 3 the best parameters according to the GCV criterion were $n_{basis} = 143$ with $\lambda = 10^{-8}$:

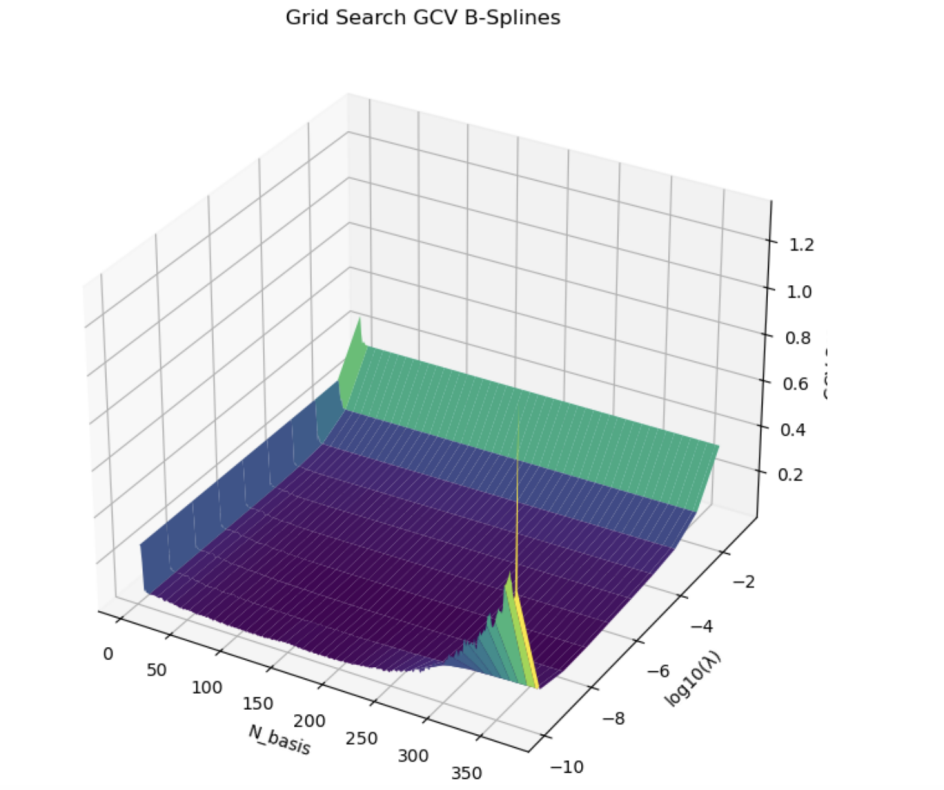

Figure 16: Grid Search GCV B-Splines (cropped)

GCV CV on Fourier was unsuccessful, Computations were too overwhelming for Python.

5.4 Comparison between Python and R

5.4.1 Presentation of the Computation methods

-4 computation methods

Their short descriptions :

- Python's scikit-fda *Py s-d*

  Remark:

  - Very little litterature on this package. [3, The only found litterature, written by the package's
  - There is a mistake for Fourier basis description (08/2024). The the first basis element of a Fourier orthonormal basis is not $\frac{1}{\sqrt{2}}$ but $\frac{1}{\sqrt{period}}$.[7, *l.82*]
  - By defaults for B-Splines, there are $n_{basis} - 2$ equidistant knots from a to b and the order is equal to 4.

- Python's by hand (with scikit-fda tools) *Py bh*

  For this computation method numpy and scikit-fda are used. To inverse matrix numpy is used, but to compute basis representation of T , and to compute the integral of its double derivative for the penalty scikit-fda is used.

- R's fda *R fda*

  Remark :

  - The most famous package for fda
  - When applying Roughness penality with Fourier, if $\frac{b}{period} \in \mathbb{N}$ then the penalty matrix is integrated only from 0 to *period* instead of from 0 to b.
    In theory this situation does not change the result since when it is the case the lambda adjust the strengh of the penalty. Therefore during GCV $best_{\text{adj\_pen}}(\lambda) = \frac{b}{period} best(\lambda)$.
    Since the goal here is to compare results, it is better to use the same formula for every method. Therefore, when it is the case, the penalty matrix has to be multiplied by $\frac{b}{period} \in \mathbb{N}$ to get back the true penalty matrix from the formula (quote formula from paper and paper).

- By defaults for B-Splines, there are $n_{basis} - 2$ equidistant knots from a to b and the order is equal to 4. Even though the end-knots (a & b) are used for the computations they do not appear in the parameters of the basis.

- R by hand (with R's fda tools) *R bh*
  Remark:
  - For this computation method fda aND the R function solve() are used..
  - To inverse matrix solve() is used
  - To compute basis representation of T , and to compute the integral of its double derivative for the penalty fda is used.

5.4.2 Comparisons

For Fourier Bases and B-Splines Bases

- Simple FDA regression (distance functions, distance coefficients)
- Roughness penalised fitting criterion distance functions, distance coefficients)
- General Cross Validation (distance scores)

5.4.3 Comparisons on artificial Cyclic data

Simple FDA regression

Method	**Coefficients Distance**	**Functions Distance**
Fourier(11, 0.5)	1.077e-10	4.817e-10
Fourier(211, 10)	3.390e-07	3.390e-07
Fourier(97, 7.43)	2.822e-10	3.274e-10
B-Splines(501)	6.640e-08	6.562e-09

Table 4: Distance Metrics for Fourier and B-Splines with various parameters

The **Distance of the coefficients** is computed as the $norm_2$ of the difference of the coefficients from scikit-fda and R's fda
The **Distance of Functions** is computed as the L_2 norm of the difference between the

function from R's fda vs Scikit-fda.

Roughness penalised fitting criterion

Remark : On R when applying Roughness penality with Fourier, if $\frac{b}{period} \in \mathbb{N}$ then the penalty matrix is integrated only from 0 to *period* instead of from 0 to b. Therefore, when it is the case, the penalty matrix has to be multiplied by $\frac{b}{period} \in \mathbb{N}$ to get back the true penalty matrix from the formula.

Tested λ values will be 10^i for $i \in [\![-8, 3]\!]$

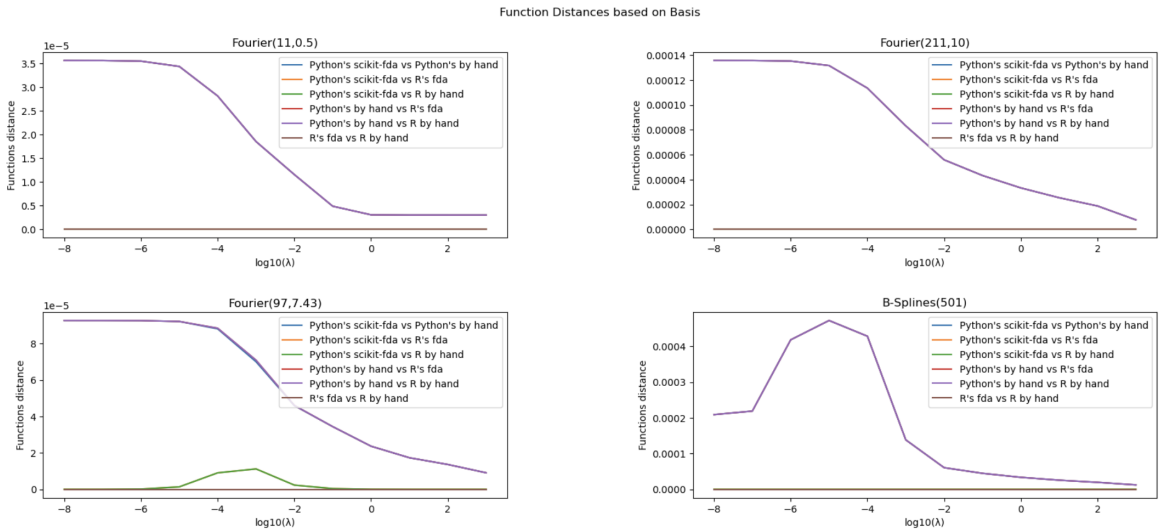

Figure 17: Distance based on method, basis, λ

Methods Comparison	Fourier(11, 0.5)	Fourier(211, 10)	Fourier(97, 7.43)	B-Splines(501)
Py s-f v Py bh	**2.161e-04** (4.833e-05)	**9.194e-04** (9.194e-04)	**6.721e-04** (6.074e-04)	**2.080e-03** (3.255e-02)
Py s-f v R fda	**1.360e-13** (3.041e-14)	**8.001e-13** (8.001e-13)	**2.455e-05** (2.337e-05)	**4.057e-10** (2.896e-09)
Py s-f v R bh	**1.384e-13** (3.095e-14)	**7.459e-13** (7.459e-13)	**2.455e-05** (2.337e-05)	**4.888e-10** (3.509e-09)
Py bh v R fda	**2.161e-04** (4.833e-05)	**9.194e-04** (9.194e-04)	**6.735e-04** (6.090e-04)	**2.080e-03** (3.255e-02)
Py bh vs R bh	**2.161e-04** (4.833e-05)	**9.194e-04** (9.194e-04)	**6.735e-04** (6.090e-04)	**2.080e-03** (3.255e-02)
R fda vs R bh	**1.144e-13** (2.558e-14)	**1.628e-13** (1.628e-13)	**1.269e-13** (1.106e-13)	**1.448e-10** (1.061e-09)

Table 5: $\sum$ over λ-values of **Functions** (Coefficients) distances methods×basis

Functions and Coefficients are very similar. Python by hand is more different than the 3 other methods. For the three other distances between each other are of order $< 10^{-13}$ whereas with Python by hand it of order $> 10^{-6}$.

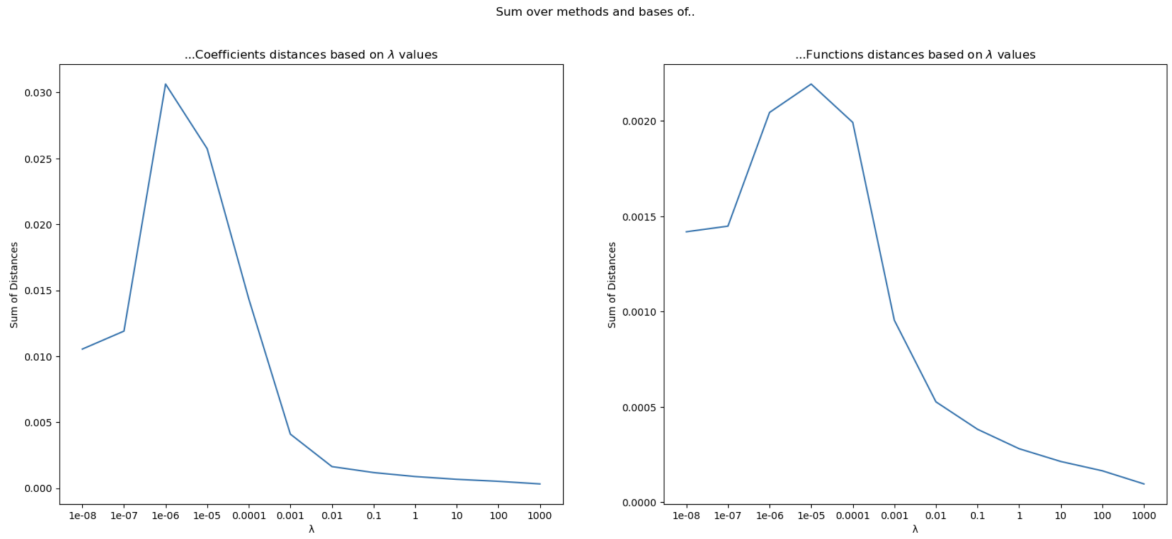

Figure 18: Sum of Distances across methods and bases by λ

λ	1e-08	1e-07	1e-06	1e-05	0.0001	0.001	0.01	0.1	1	10	100	1000
Function	1.419e-03	1.448e-03	2.044e-03	2.193e-03	1.992e-03	9.543e-04	5.265e-04	3.827e-04	2.807e-04	2.138e-04	1.649e-04	9.612e-05
Coefficients	1.054e-02	1.190e-02	3.064e-02	2.574e-02	1.435e-02	4.091e-03	1.628e-03	1.176e-03	8.736e-04	6.641e-04	5.106e-04	3.141e-04

Table 6: Sum of distances across methods and bases of by λ values

GCV Comparison:

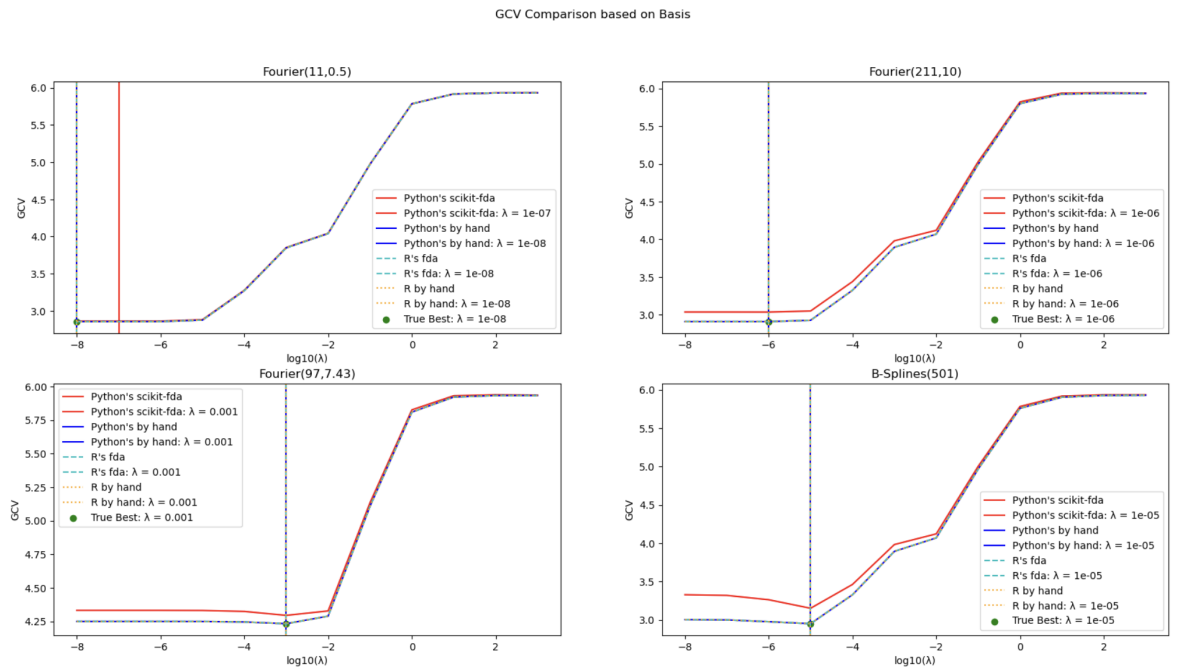

Figure 19: Cyclic Artificial Data: GCV comparison on several bases

Python's scikit-fda does not have the same scores as the three other methods. It is

curious since Python's scikit-fda was more similar to the two R's methods than Python by hand for the roughness penalised regression. The GCV is here efficient. Indeed, it finds the best lambda among the tested values for all the tested bases (apart from scikit-fda for Fourier(11,0.50) but still very close).

Methods	$F(11, 0.5)$	$F(211, 10)$	$F(97, 7.43)$	$BS(501)$
Py s-f v Py bh	1.479e-02	2.974e-01	1.998e-01	6.000e-01
Py s-f vs R fda	1.481e-02	2.974e-01	1.998e-01	6.000e-01
Py s-f vs R bh	1.481e-02	2.974e-01	1.998e-01	6.000e-01
Py bh vs R fda	2.365e-05	2.487e-05	2.172e-05	2.632e-05
Py bh vs R bh	2.365e-05	2.487e-05	2.172e-05	2.632e-05
R fda vs R bh	4.396e-15	5.564e-15	5.617e-15	8.469e-13

Table 7: Comparison Metrics for Different Methods and Bases

It is difficult to explain the GCV discrepancy. There is very little literature on Python's scikit-fda package (only one paper) and the code structure is very complex to explore.

5.4.4 Comparison on the Vietnam climate data

In this section, results differences depending on the bases and methods used will be explored.

First let's compare the results, when there are...

no roughness penalties:

Methods	$F(301, 1)$	$F(101, 1)$	$F(31, 1)$	$BS(25)$
Py bh vs R bh	2.860e-12 (2.862e-12)	2.718e-12 (2.716e-12)	2.805e-12 (2.806e-12)	2.738e-12 (1.326e-11)
Py bh vs R fda	2.760e-12 (2.762e-12)	2.718e-12 (2.717e-12)	2.710e-12 (2.709e-12)	2.762e-12 (1.337e-11)
Py s-f vs Py bh	1.977e-13 (2.007e-13)	8.939e-14 (8.840e-14)	2.702e-14 (2.598e-14)	3.067e-14 (2.113e-13)
Py s-f vs R bh	2.799e-12 (2.798e-12)	2.753e-12 (2.752e-12)	2.830e-12 (2.830e-12)	2.720e-12 (1.318e-11)
Py s-f vs R fda	2.699e-12 (2.699e-12)	2.754e-12 (2.753e-12)	2.734e-12 (2.733e-12)	2.743e-12 (1.329e-11)
R fda vs R bh	1.039e-13 (1.012e-13)	1.965e-14 (1.539e-14)	1.001e-13 (1.004e-13)	4.101e-14 (3.008e-13)

Table 8: Comparison of distances using various methods and bases

Results are extremely similar. The small errors might even be due to the transfer of coefficients from R to Python via a csv file.

When the bases representation are plotted, they are all on top of each other:

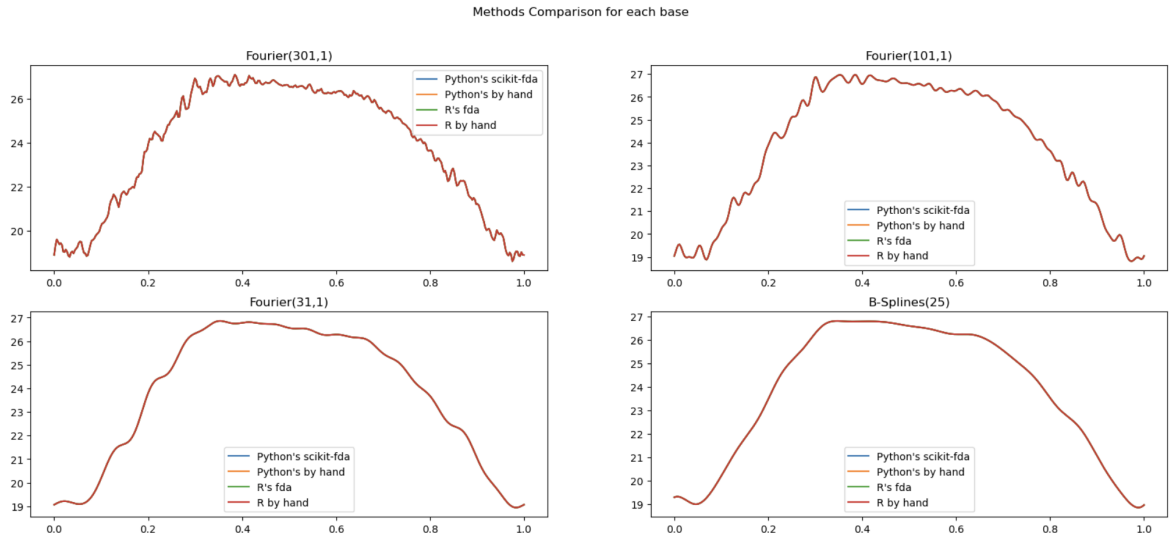

Figure 20: Basis Representation for each method

When ...

Roughness penalised fitting criterion is applied:

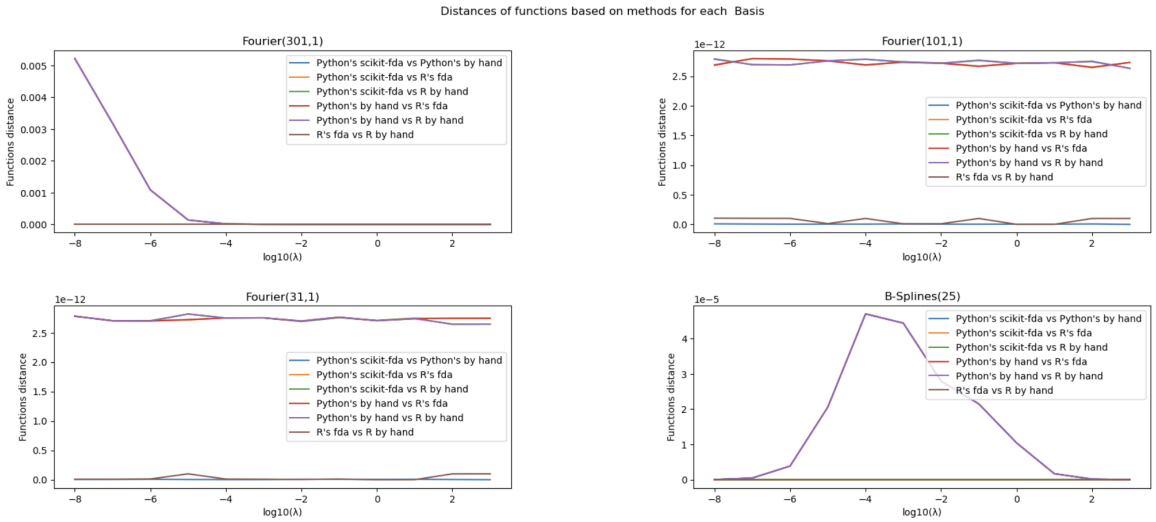

Figure 21: Distances between methods' bases representations for Roughness penalised fitting criterion

For Fourier(301,1), we notice a slight difference when λ tend to 0. This is surprising because, in the previous sections, the results were almost identical when there was no penalty (which is the same as $\lambda = 0$). It might be that the Fourier(301,1) curve becomes more similar to B-Splines(25) if $log_10(\lambda)$ starts further back than -8 (Fig.21).

For Fourier(101,1) and Fourier(31,1), the results are very close, with a similarity on the scale of $1e-12$. Even though the results are very similar, the R methods tend to be more similar to each other than to other methods. The same cannot be said about Python. In Fourier(301,1) and B-Splines(25), the results from Scikit-fda are closer to R's results than to Python's by-hand ones, but this is not the case for Fourier(101,1) and Fourier(31,1). The numerical results are as follows:

Methods	$F(301,1)$	$F(101,1)$	$F(31,1)$	$BS(25)$
Py s-f v Py bh	9.627e-03 (9.627e-03)	5.714e-14 (5.714e-14)	5.192e-14 (5.192e-14)	1.779e-04 (2.253e-03)
Py s-f vs R fda	3.254e-11 (3.254e-11)	3.269e-11 (3.269e-11)	3.281e-11 (3.281e-11)	2.432e-09 (1.339e-08)
Py s-f vs R bh	3.283e-11 (3.283e-11)	3.279e-11 (3.279e-11)	3.271e-11 (3.271e-11)	5.843e-09 (3.053e-08)
Py bh vs R fda	9.627e-03 (9.627e-03)	3.267e-11 (3.267e-11)	3.281e-11 (3.281e-11)	1.779e-04 (2.253e-03)
Py bh vs R bh	9.627e-03 (9.627e-03)	3.276e-11 (3.276e-11)	3.271e-11 (3.271e-11)	1.779e-04 (2.253e-03)
R fda vs R bh	3.669e-13 (3.669e-13)	7.379e-13 (7.379e-13)	3.629e-13 (3.629e-13)	4.803e-09 (2.456e-08)

Table 9: Distances between methods' bases representations for R.P.F.C.

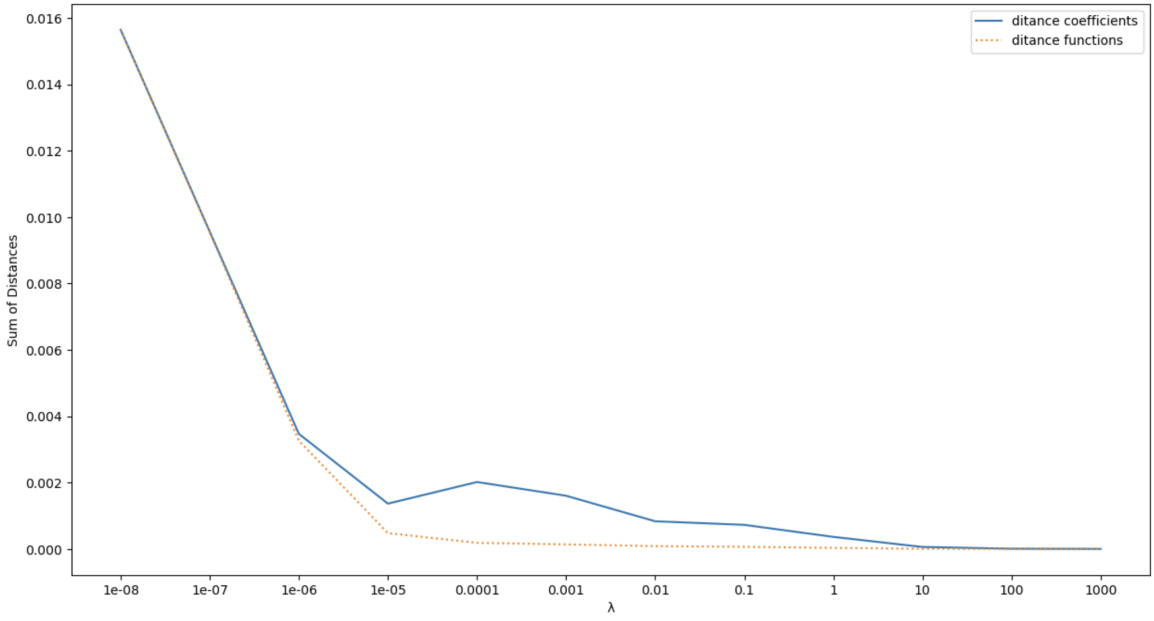

Figure 22: Sum of distances across bases and methods for each λ

Parameter	**1e-08**	**1e-07**	**1e-06**	**1e-05**	**0.0001**	**0.001**	**0.01**	**0.1**	**1**	**10**	**100**	**1000**
funct.	1.564e-02	9.533e-03	3.261e-03	4.751e-04	1.833e-04	1.373e-04	8.421e-05	6.441e-05	3.128e-05	5.070e-06	5.426e-07	7.118e-08
coef.	1.564e-02	9.560e-03	3.468e-03	1.365e-03	2.016e-03	1.602e-03	8.330e-04	7.252e-04	3.627e-04	5.893e-05	6.296e-06	7.036e-07

Table 10: Sum of distances across bases and methods for each λ

It is interesting to notice that the distance of coefficients and functions are extremely close from 10^{-8} to 10^{-6}. The results are rather similar when λ is big but they are more distant where λ is small. This probably comes from Fourier(301,1), where disparities could be observed for Python's by hand compared to the 3 other methods. [Top left Fig. 21]

Let's see if there are differences for λ selection with...

the GCV criterion:

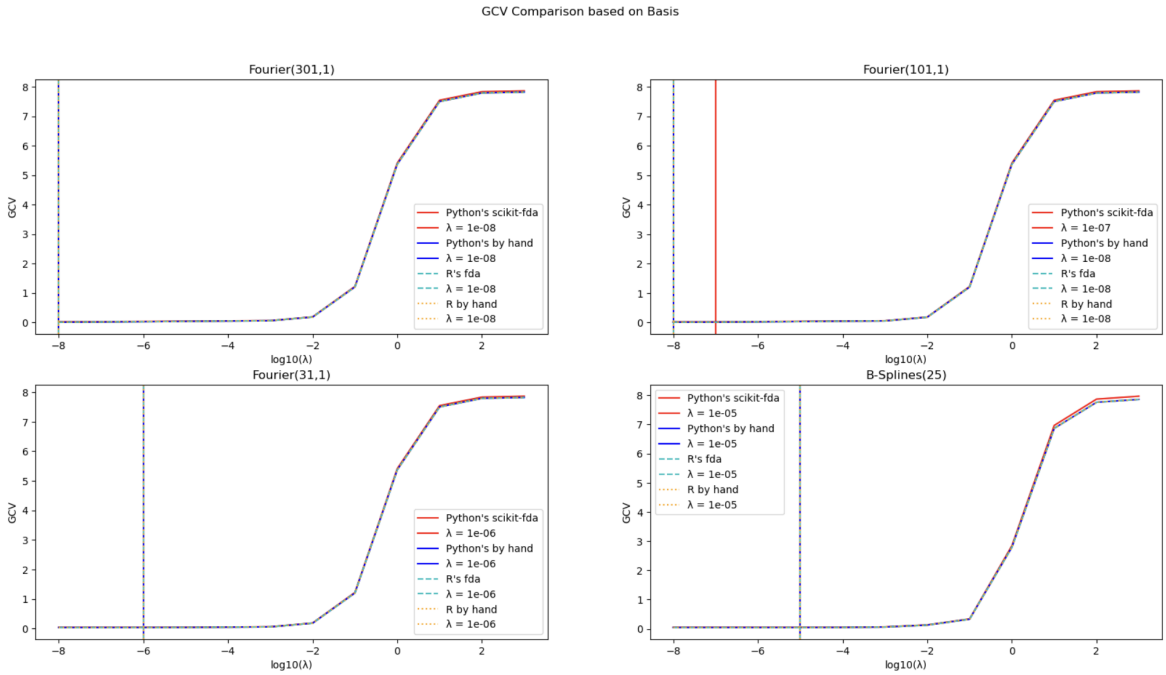

Figure 23: GCV Comparison between methods

The 4 methods have the same results for the 4 tested bases except for $Fourier(101, 1)$ where there is a little difference with scikit-fda recommending 10^{-7} whereas the 3 other methods recommend 10^{-8}.

Methods	$F(301,1)$	$F(101,1)$	$F(31,1)$	$BS(25)$
Py s-f v Py bh	9.008e-02	8.953e-02	8.878e-02	1.920e-01
Py s-f vs R fda	8.999e-02	8.953e-02	8.878e-02	1.920e-01
Py s-f vs R bh	8.999e-02	8.953e-02	8.878e-02	1.920e-01
Py bh vs R fda	4.847e-04	4.604e-12	4.618e-12	6.601e-06
Py bh vs R bh	4.847e-04	4.604e-12	4.607e-12	6.601e-06
R fda vs R bh	1.436e-14	1.023e-14	1.812e-14	6.883e-11

Table 11: N_2 distance between methods' GCV scores

In the table 11, it can be observed, even if it is not by much, that scikit-fda would be the outlier if one had method had to be qualified as such. It is curious because in the previous subsection, the results of scikit-fda were extremely similar to the R's ones and Python by hand was the outlier. There is little literature about this package, and it was complicated to decrypt the code [6, *l.144*], therefore I was not able to find reason why it was the case. The only litterature about the function that compute GCV score is [3, *p.13*]. The formula they display is the same as 7 but for some reason the results are different.

6 Subject Summary and Conclusion

After reviewing the fundamentals of Functional Data Analysis (FDA) and basis representations, we explored two datasets.

The first dataset, "Artificial Cyclic Data" was generated from a known highly cyclic function. We created two versions: one without noise, termed "true," and another with noise, termed "noisy." The objective was to evaluate whether FDA smoothing methods could recover data similar to the true dataset from the noisy version. Initially, we observed the effects of a loss function without roughness penalty. Fourier basis functions were highly effective when the data elements were generated by the basis functions. However, their performance was average otherwise. It was crucial to select an appropriate n_{basis}; too small or too large values could lead to suboptimal results. We found that, in some cases, having a well-chosen n_{basis} even when the basis functions did not

generate the data elements was better than using a larger n_{basis} with generating functions, due to issues like overfitting.

The Generalised Cross-Validation (GCV) criterion proved to be quite accurate, selecting the best option in 3 out of 4 tested bases. Unlike the artificial data, we could not make similar comparisons with the Vietnam temperature data due to the absence of clear "noisy" and "true" data. Nonetheless, some interesting observations emerged during the Grid Search with Fourier basis functions without roughness penalty. For certain values where $n_{\text{basis}} = n$, the loss function was not zero, indicating that the Φ matrix was not full rank. This could be attributed to a mismatch between the basis period and the function period. Additionally, in some cases, adding two Fourier bases with the same period but different n_{basis} resulted in higher loss for the basis with a larger n_{basis}, which was theoretically unexpected.

The GCV did not show a significant difference between penalised and unpenalised basis representations, making it hard to distinguish between them. We attempted a Grid Search on B-Splines, which was successful, but faced challenges with Fourier basis functions using scikit-fda, and the results were inconclusive.

In the second part, we compared four different computational methods: Python's Scikit-fda, a manual Python implementation (using a mix of scikit-fda and numpy), R's fda package, and manual R implementation (using R's fda and the 'solve' function). The comparison covered three aspects: unpenalised roughness basis representation, penalized roughness basis representation, and GCV.

For both datasets, we observed that results for unpenalised basis representations were very close across methods. However, discrepancies appeared in the other two aspects. Python's manual method was an "outlier" for roughness penalised regressions basis representation, showing slightly different results compared to the other methods, which were very close to each other (with differences of 10^{-14} and 10^{-11} for the artificial and Vietnam datasets, respectively, versus 10^{-3} and 10^{-2} with Python by hand). In terms

of GCV, scikit-fda was the "outlier," selecting different λ values in some cases.

In conclusion, the discrepancies observed might be due to overwhelming computations for computers, different computational methods, or a coding error from me. Further investigation is required to determine the cause of these discrepancies. Someone with a higher level at decoding Python package might understand where the differences come from.

A another subject that could be interesting and link to the subject is to do the same comparisons but for registration.

7 Bibliography

References

[1] Matthieu, D. A. (2024a). Python cyclic artificial. https://colab.research.google.com/drive/13ASCjvajqzu9Fj9FH_nD_yY_JO8lVHpQ. Accessed: 2024-08-24.

[2] Matthieu, D. A. (2024b). Python vietnam. https://colab.research.google.com/drive/1CBP-x3qkTzrl1ffaF8QcZG_cTer2eCGX. Accessed: 2024-08-24.

[3] Ramos-Carreño, C., Torrecilla, J., Carbajo-Berrocal, M., Marcos, P., and Suárez, A. (2024). scikit-fda : A python package for functional data analysis. *Journal of Statistical Software*, 109.

[4] Ramsay, J., Hooker, G., and Graves, S. (2009). *Functional Data Analysis with R and MATLAB*. Springer New York, New York, NY.

[5] Ramsay, J. O. and Silverman, B. W. (2005). *Functional Data Analysis*. Springer Series in Statistics. Springer, New York, NY.

[6] scikit-fda developers, T. (2023). Github scikit-fda: Functional data analysis in python, preprocessing, smoothing, validation.py. [Accessed 24-08-2024].

[7] scikit-fda developers, T. (2024). Github scikit-fda: Functional data analysis in python, fourier basis.

8 Annex

8.1 Code

8.1.1 Python code to generate Cyclic Artificial Data

```
import numpy as np


a,b = 0,10
```

```
n = (b - a)*100
x = np.linspace(a,b,n)
y1 = np.cos(2*np.pi*x) +  2*np.sin(4*np.pi*x)

max_noise = round(np.max(np.abs(y1))/2,1)
print(max_noise)

nsim = 1000
import random
random.seed(10)
noise = np.random.normal(0, 1, n * nsim).reshape(nsim, n)
noise_level = np.linspace(0,max_noise,int(max_noise*10)+1)
noises = np.array([noise * nlev for nlev in noise_level])

noisy_y1 = y1 + noises

Noisy = (y1 + noises[-1][-1]).round(3)
true = y1.round(3)
```

8.2 R code Artificial

: Import the dataset used in Python:

```
library(fda)
library(ggplot2)
library(dplyr)
library(demography)
library(readr)

Data <- read_csv("Data_to_export.csv")[,-1]
```

Now let's define true and noisy curves:

```
a = 0
b = 10
n = 1000*(b-a)+1
x = seq(from = a, to = b, length.out = n )
true = round(unlist(t(Data)[,1]),3)
noisy  = round(unlist(t(Data)[,2]),3)
```

Let's check if coefficients are the same in Python and R. Define truncated bases:

```
Fourier5 <- create.fourier.basis(c(a, b), nbasis = 11, period = 0.5)
Fourier5_ <- create.fourier.basis(c(a, b), nbasis = 211, period = 10)
Fourier5__ <- create.fourier.basis(c(a, b), nbasis = 97, period = 7.43)
```

```
BSplines5 <- create.bspline.basis(c(a, b), nbasis = 101)
Bsplines_ <- create.bspline.basis(rangeval=c(a, b), nbasis = 501)
```

Function to display Fourier:

```
if (!requireNamespace("latex2exp", quietly = TRUE)) {
  install.packages("latex2exp")
}
if (!requireNamespace("fda", quietly = TRUE)) {
  install.packages("fda")
}

library(latex2exp)
library(fda)

k_coef <- function(k, coef, per, round_digit = 3) {
  if (coef == as.integer(coef)) {
    coef <- as.integer(coef)
  }
  inside_coef <- (k %/% 2) * 2 / per
  if (inside_coef == as.integer(inside_coef)) {
    inside_coef <- as.integer(inside_coef)
  } else {
    inside_coef <- round(inside_coef, round_digit)
  }
  if (k < 1) {
    return("k-\\geq-1")
  } else if (k == 1) {
    return(as.character(coef))
  } else if (k %% 2 == 0) {
    return(sprintf("%s\\cdot-\\sin\\left(%s\\pi-t\\right)", coef, inside_coef))
  } else {
    return(sprintf("%s\\cdot-\\cos\\left(%s\\pi-t\\right)", coef, inside_coef))
  }
}

Display_Fourier_Function <- function(Fourier_basis, y, round_digit = 3) {
  per <- Fourier_basis$params
  co <- c(smooth.basis(x, y, Fourier_basis)$fd$coefs)
  co[1] <- co[1] / sqrt(2)
  co <- round(co / sqrt(per / 2), round_digit)
  co <- ifelse(co == as.integer(co), as.integer(co), co)
  ind <- which(co != 0)
  latex_str <- paste(sapply(ind, function(i) {
    k_coef(i, co[i], per, round_digit)
  }), collapse = "-+-")
  latex_str <- gsub("\\+-", "-", latex_str)
  return(latex2exp::TeX(paste0("$", latex_str, "$")))
}
```

Compute non-penalized basis representation:

```
#Display_Fourier_Function(Fourier5, noisy, 5)
#c(smooth.basis(x, noisy, Fourier5)$fd$coefs)
#Display_Fourier_Function(Fourier5_, noisy, 5)
#c(smooth.basis(x, noisy, Fourier5_)$fd$coefs)
#Display_Fourier_Function(Fourier5__, noisy, 5)
#c(smooth.basis(x, noisy, Fourier5__)$fd$coefs)
#c(smooth.basis(x, noisy, Bsplines_)$fd$coefs)
```

Yes, they are the same. It is not surprising; after all, it is just a simple regression.
Let's see now what happens when we add a penalty. Integral of the squared double derivative to be more accurate.
When b/period in $\mathbb{N}$, then a small change is needed: pen_mat = (b/period)pen_mat.

```
Four__test <- create.fourier.basis(c(a, b), nbasis = 5, period = 13.4)
Four__test1 <- create.fourier.basis(c(a, b), nbasis = 5, period = 5)
```

```
Four__test2 <- create.fourier.basis(c(a, b), nbasis = 5, period = 3.875)

base_pen <- function(basis){
  as.data.frame(getbasispenalty(basis, Lfdobj = int2Lfd(2)))
}

base_pen(Four__test)
base_pen(Four__test1)
base_pen(Four__test2)

pen_basis <- function(Four, log10_lbd, Fourier = TRUE) {
  coef = 1
  if (Fourier == TRUE) {
    cc = b / Four$params
    if (cc == as.integer(cc)) {
      coef = cc
    }
  }
  pen_matrix = coef * getbasispenalty(Four, Lfdobj = int2Lfd(2))
  fdo = fdPar(Four, lambda = 10**(log10_lbd), Lfdobj = int2Lfd(2), penmat = pen_matrix)
  return(c(smooth.basis(x, noisy, fdParobj = fdo)$fd$coef))
}
```

Compute different penalized regressions based on basis and lambda with FDA:

```
#Ff = c()
#for (l in seq(-8, 3)) {
#  Ff <- c(Ff, pen_basis(Fourier5, l), pen_basis(Fourier5_, l), pen_basis(Fourier5__, l), pen_basis(Bsplines_, l, FALSE))
#}
#write.csv(as.data.frame(Ff), file = "Pen_base_noisy.csv")
```

Penalized FDA by hand:
1st step:

```
f_s1 = penalised_fda_by_hand1s(Fourier5, x)
f_s1_ = penalised_fda_by_hand1s(Fourier5_, x)
f_s1__ = penalised_fda_by_hand1s(Fourier5__, x)
bs_s1 = penalised_fda_by_hand1s(Bsplines_, x, FALSE)
```

2nd step:

```
'''Ff_bh=c()
for(l in seq(-8,3)){
Ff_bh<-c(Ff_bh,
---------------penalised_fda_by_hand2s(f_s1$PHI,f_s1$R,10**(l),noisy),
---------------penalised_fda_by_hand2s(f_s1_$PHI,f_s1_$R,10**(l),noisy),
---------------penalised_fda_by_hand2s(f_s1__$PHI,f_s1__$R,10**(l),noisy),
---------------penalised_fda_by_hand2s(bs_s1$PHI,bs_s1$R,10**(l),noisy)
---------------)
}
Ff_bh'''

#write.csv(as.data.frame(cbind(Ff, Ff_bh)), file = "Pen_base_noisy.csv")
```

Grid Search for GCV:

```
GCV_hand_made <- function(basis, x, y, lbds, Fourier = TRUE) {
  res = penalised_fda_by_hand1s(basis, x, Fourier)
  PHI = res$PHI
  R = res$R
  GVC_bh_Splines = c()
  for (lbd in 10**lbds) {
    H = PHI %*% solve(t(PHI) %*% PHI + lbd * R) %*% t(PHI)
    df = sum(diag(H))
    SSE = sum((y %*% H - y)**2)
    GVE = SSE / (n * (1 - df / n)**2)
    GVC_bh_Splines = c(GVC_bh_Splines, GVE)
  }
```

```
  return(GVC_bh_Splines)
}

pen_matrix = 20 * getbasispenalty(Fourier5, Lfdobj = int2Lfd(2))
fdo = fdPar(Fourier5, lambda = 0.1, Lfdobj = int2Lfd(2), penmat = pen_matrix)
fdo_ = fdPar(Fourier5_, lambda = 0.1, Lfdobj = int2Lfd(2))
fdo__ = fdPar(Fourier5__, lambda = 0.1, Lfdobj = int2Lfd(2))
bs_fdo = fdPar(Bsplines_, lambda = 0.1, Lfdobj = int2Lfd(2))

log10lbd = c(-8:3)
GCV_f5 = sapply(log10lbd, function(i) lambda2gcv(i, x, noisy, fdo))
GCV_f5_ = sapply(log10lbd, function(i) lambda2gcv(i, x, noisy, fdo_))
GCV_f5__ = sapply(log10lbd, function(i) lambda2gcv(i, x, noisy, fdo__))
GCV_bs = sapply(log10lbd, function(i) lambda2gcv(i, x, noisy, bs_fdo))

hm_GCV_f5 <- GCV_hand_made(Fourier5, x, noisy, log10lbd)
hm_GCV_f5_ <- GCV_hand_made(Fourier5_, x, noisy, log10lbd)
hm_GCV_f5__ <- GCV_hand_made(Fourier5__, x, noisy, log10lbd)
hmGCV_bs <- GCV_hand_made(Bsplines_, x, noisy, log10lbd, FALSE)

GCV = as.data.frame(cbind(GCV_f5, GCV_f5_, GCV_f5__, GCV_bs, hm_GCV_f5, hm_GCV_f5_, hm_GCV_f5__, hmGCV_bs))
write.csv(GCV * 10**(15), file = "GCV10**15.csv")

GCV_f5 - hm_GCV_f5
```

8.2.1 R code, Vietnam

```
Load the data
load("~/Downloads/vietnamese_data.RData")
library(dplyr)
library(lubridate)
library(tidyr)
library(tibble)
```

Transform the data into new data frame where each row represents a day of the year at a specif location and year, and values is the average, minimum and maximum temperature for respectively donnees_t2m, donnees_t2min, donnees_t2max. 29th February was deleted for conveniance.

```
data_ <- long_data%>%mutate(day = day(day))%>%
  mutate(day_month = sprintf("%s-%02d", month, day))%>%
  arrange(long,lat,year,month,day)%>%
  select(long,lat,t2m,t2max,t2min,day_month,year)

data_ = data_%>%mutate(t2m = round(t2m,13),t2max = round(t2max,13),t2min = round(t2min,13)) #to get the same value with

donnees_t2m <- data_%>%select(-c(t2min,t2max))%>%pivot_wider(
  names_from = day_month,
  values_from = t2m
)%>%select(-matches('February-29'))%>%
  mutate(index = paste0(as.character(long), "_", as.character(lat), "_", as.character(year)))%>%
  column_to_rownames(var = "index")%>%select(-c(long,lat,year))

donnees_t2min <- data_%>%select(-c(t2m,t2max))%>%pivot_wider(
  names_from = day_month,
  values_from = t2min
)%>%select(-matches('February-29'))%>%
  mutate(index = paste0(as.character(long), "_", as.character(lat), "_", as.character(year)))%>%
  column_to_rownames(var = "index")%>%select(-c(long,lat,year))

donnees_t2max <- data_%>%select(-c(t2min,t2m))%>%pivot_wider(
  names_from = day_month,
  values_from = t2max
```

```
)%>%select(-matches('February-29'))%>%
  mutate(index = paste0(as.character(long), "_", as.character(lat), "_", as.character(year)))%>%
  column_to_rownames(var = "index")%>%select(-c(long,lat,year))

#Create CSV files
#write.csv(donnees_t2m, file = "all_data_meteo_csv.csv")
#write.csv(donnees_t2min, file = "allMin_data_meteo_csv.csv")
#write.csv(donnees_t2max, file = "allMax_data_meteo_csv.csv")

#Plot of the curves
matplot(
  t(donnees_t2m), # Transpose to plot each row as a line
  type = "l",     # Line plot
  lty = 1,        # Line type
  pch = 19,       # Point character
  col = 1:nrow(donnees_t2m), # Use different colors for each line
  xaxt = "n",     # Do not draw x-axis (we'll do it manually)
  ylab = "Values",# Y-axis label
  xlab = "Date",  # X-axis label
  main = "Yearly-Temperatures" # Title
)

avg_by_date = colMeans(donnees_t2m)
avg <- array(unlist(avg_by_date))

n = 365
a = 0
b = 1
j = seq(from = a, to = b, length.out = n)

Create a Fourier Basis and smooth the result.
Fourier <- create.fourier.basis( c(a, b), nbasis = 301, period = 1)
Fourier_ <- create.fourier.basis( c(a, b), nbasis = 101, period = 1)
Fourier__ <- create.fourier.basis( c(a, b), nbasis = 31, period = 1)
Fourier___ <- create.fourier.basis( c(a, b), nbasis = 3, period = 1)
B_splines <- create.bspline.basis(rangeval=c(a, b), nbasis=25)

options(digits = 15)
smooth_Fourier <- smooth.basis(j, avg_by_date, Fourier)
smooth_Fourier_ <- smooth.basis(j, avg_by_date, Fourier_)
smooth_Fourier__ <- smooth.basis(j, avg_by_date, Fourier__)
smooth_Fourier___ <- smooth.basis(j, avg_by_date, Fourier___)
smooth_B_splines <- smooth.basis(j, avg_by_date, B_splines)
tab <- cbind(smooth_Fourier$fd$coefs[,1],smooth_Fourier_$fd$coefs[,1], smooth_Fourier__$fd$coefs[,1],smooth_Fourier___$fc
colnames(tab) <- c("Fourier5","Fourier5_","Fourier5__","Fourier5___","B_splines")
write.csv(tab, file = "coefs_fda.csv")

#Roughness penalized fitting criterion Fourier
pen_basis <- function(Four,log10_lbd,Fourier =TRUE){

coef = 1
if(Fourier == TRUE){
cc = b/Four$params
if(cc == as.integer(cc)){ coef = cc }
}

pen_matrix = coef*getbasispenalty(Four, Lfdobj = int2Lfd(2))
fdo = fdPar(Four, lambda = 10**(log10_lbd), Lfdobj = int2Lfd(2),penmat = pen_matrix)
return( c(smooth.basis(j, avg_by_date,  fdParobj = fdo)$fd$coef ) )
}

#Compute Values penalise values for our bases with log10 lambda from -8 to 3
Ff = c()
for (l in seq(-8, 3)){
```

```
Ff <- c(Ff,pen_basis(Fourier, l),pen_basis(Fourier_, l),pen_basis(Fourier__, l),pen_basis(B_splines, l,FALSE))
}
write.csv( as.data.frame(Ff),file ="Pen_base_Vietnam.csv")

#R by hand:
penalised_fda_by_hand1s <- function(basis,x,Fourier=TRUE){
  coef = 1
  if(Fourier == TRUE){
    cc = b/basis$params
    if(cc == as.integer(cc)){ coef = cc }
    }
  PHI = eval.basis(x,basis)
  R = coef*getbasispenalty(basis, Lfdobj = int2Lfd(2))
  ret = list(PHI = PHI, R = R)
  return(ret)
}

penalised_fda_by_hand2s <- function(PHI,R,lbd,y){
  return( solve(t(PHI)%*%PHI + lbd*R)%*%t(PHI)%*%y )
}

penalised_fda_by_hand <- function(basis,x,y,lbd,Fourier=TRUE){
  s1 <- penalised_fda_by_hand1s(basis,x,Fourier)
  s2 <- penalised_fda_by_hand2s(s1$PHI,s1$R,lbd,y)
  return(s2)
}

f_s1 = penalised_fda_by_hand1s(Fourier,j)
f_s1_ = penalised_fda_by_hand1s(Fourier_,j)
f_s1__ = penalised_fda_by_hand1s(Fourier__,j)
bs_s1 = penalised_fda_by_hand1s(B_splines,j,FALSE)

By hand no penalty:
By_hand <- c(
  penalised_fda_by_hand(Fourier,j,avg_by_date,0),
  penalised_fda_by_hand(Fourier_,j,avg_by_date,0),
  penalised_fda_by_hand(Fourier__,j,avg_by_date,0),
  penalised_fda_by_hand(Fourier___,j,avg_by_date,0),
  penalised_fda_by_hand(B_splines,j,avg_by_date,0,Fourier=FALSE)
)
write.csv( as.data.frame(By_hand),file ="R_by_hand_Vietnam.csv")

#Comptute bases representations by hand:
Ff_bh = c()
for (l in seq(-8, 3)){
 Ff_bh <- c(Ff_bh ,
              penalised_fda_by_hand2s(f_s1$PHI,f_s1$R,10**(l),avg_by_date),
              penalised_fda_by_hand2s(f_s1_$PHI,f_s1_$R,10**(l),avg_by_date),
              penalised_fda_by_hand2s(f_s1__$PHI,f_s1__$R,10**(l),avg_by_date),
              penalised_fda_by_hand2s(bs_s1$PHI,bs_s1$R,10**(l),avg_by_date)
              )
}
write.csv( as.data.frame(cbind(Ff,Ff_bh)),file ="Pen_base_Vietnam.csv")

#GCV
#By hand:
GCV_hand_made <- function(basis,x,y,lbds,Fourier=TRUE){
  res = penalised_fda_by_hand1s(basis,x,Fourier)
  PHI = res$PHI
  R = res$R
  GVC_bh_Splines = c()
  for (lbd in 10**lbds){
    H = PHI%*%solve(t(PHI)%*%PHI+lbd*R)%*%t(PHI)
    df = sum(diag(H))
```

```
    SSE = sum((y%*%H - y)**2)
    GVE = SSE/(n*(1-df/n)**2)
    GVC_bh_Splines = c(GVC_bh_Splines,GVE)}
  return(GVC_bh_Splines)
}

#Results package fda:
fdo = fdPar(Fourier, lambda = 0.1, Lfdobj = int2Lfd(2))
fdo_ = fdPar(Fourier_, lambda = 0.1, Lfdobj = int2Lfd(2))
fdo__ = fdPar(Fourier__, lambda = 0.1, Lfdobj = int2Lfd(2) )
bs_fdo = fdPar(B_splines, lambda = 0.1, Lfdobj = int2Lfd(2) )

log10lbd = c(-8:3)
GCV_f5 = sapply(log10lbd, function(i) lambda2gcv(i,j, avg_by_date, fdo) )
GCV_f5_ = sapply(log10lbd, function(i) lambda2gcv(i,j, avg_by_date, fdo_) )
GCV_f5__ = sapply(log10lbd, function(i) lambda2gcv(i,j, avg_by_date, fdo__) )
GCV_bs = sapply(log10lbd, function(i) lambda2gcv(i,j, avg_by_date, bs_fdo) )

#By hand:
hm_GCV_f5 <- GCV_hand_made(Fourier,j,avg_by_date,log10lbd)
hm_GCV_f5_ <- GCV_hand_made(Fourier_,j,avg_by_date,log10lbd)
hm_GCV_f5__ <- GCV_hand_made(Fourier__,j,avg_by_date,log10lbd)
hmGCV_bs <- GCV_hand_made(B_splines,j,avg_by_date,log10lbd,FALSE)

#Gathered GCV results
GCV = as.data.frame(cbind(GCV_f5,GCV_f5_,GCV_f5__,GCV_bs,hm_GCV_f5,hm_GCV_f5_,hm_GCV_f5__,hmGCV_bs))
save the results
write.csv( GCV ,file ="R_GCV_Vietnam.csv")
```

8.2.2 Python Code Cyclic Articial Data

Google Collab .ipynb file: [1]

8.2.3 Python Code Cyclic Vietnam Data

Google Collab .ipynb file: [2]