POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science in Mathematical Engineering

Preprocessing of centred logratio transformed density functions using smoothing splines

The splineDensity R-package

Alessia Di Blasi,
Federico Pavone,
Gianluca Zeni

APSC project (8 cfu)

Academic Year 2017-1018

Contents

Abstract **3**

1 Description of the problem to solve **4**
1.1 Introduction 4
1.2 B-spline representation 5
1.3 The optimal smoothing problem 7

2 Implementation and code structure **10**
2.1 Overview 10
2.1.1 Input data 10
2.1.2 Count zeros: the Bayesian-multiplicative treatment . . 11
2.1.3 Workflow 12
2.1.4 Output 12
2.2 R/C++ interface 12
2.2.1 `.Call` and `RcppEigen` 12
2.2.2 Debug 13
2.3 C++ implementation 14
2.3.1 `parametersManager`, `dataManager` and `densityEstimator` classes 14
2.3.2 Solve method 14
2.3.3 Cross-validation function 16
2.4 OpenMP parallelization 16

3 Practical examples **20**
3.1 Application to particle-size curves in heterogeneous aquifers . 20
3.2 Application to the well-known Iris flower data set 23

4 Installation **25**
4.1 How to enable OpenMP parallelization 26

Bibliography **28**

Abstract

Density estimation is a fundamental problem in explorative data analysis. Due to the specific properties of density functions, it is not possible to tackle this kind of problem in the usual L^2 metric, instead Bayes spaces has to be considered to take into account all the geometric features of density functions. Hence raw data has to be transformed, using a proper mapping, in order to be able to work in L^2.
Our project focuses on the implementation of the solution proposed in [4]: optimal smoothing splines for centred logratio transformed density functions. In this way it is possible to satisfy our ultimate goal: preprocessing of discretized distributional observations.

Chapter 1

Description of the problem to solve

1.1 Introduction

In the analysis of large-scale database systems, information is frequently summarized using a density function, that is, Borel measurable, positive function on a support I with a unit integral constraint.

Even though it might seem that density functions are just a special case of functional data, standard FDA methods appear to be inappropriate for their treatment, as they do not consider the particular constrained nature of the data.

This problem is well known in the finite dimensional setting, where specific techniques have been worked out to deal with compositional data, i.e., multivariate data carrying only relative information, usually represented in proportions or percentages.
Those techniques are mainly based on a geometric perspective grounded on the Aitchison geometry in the simplex, which properly incorporates the compositional nature of the data (see the 'old but gold' pioneer [2]).

In this context, probability density functions have recently been interpreted as functional compositional data, i.e., functional data carrying only relative information. To handle this kind of data, the Aitchison geometry has been lately extended to the so called Bayes spaces: a Hilbert space structure for σ-finite measures, including probability measures, has been worked out, as in [8].

In order to resort to standard statistical analysis of density functions, a mapping from the Bayes space to the standard L^2 space is needed.

An isometric isomorphism between the Bayes space and the Hilbert space L^2 of (equivalence classes of) square-integrable real functions on the support I is defined by the centred log-ratio (clr) transformation, explicitly

$$clr[f(x)] = f_c(x) = ln(f(x)) - \frac{1}{\eta}\int_I ln(f(x))\,dx$$

where $f(x)$ is a density function, η is the length of the interval I.
The antitranformation is defined as:

$$clr^{-1}[f(x)] = \frac{exp(f_c(x))}{\int_I f_c(x)\,dx}.$$

It's immediate to notice that the following constraint holds:

$$\int_I clr[f(x)]\,dx = 0.$$

This additional condition needs to be taken into account for computation and analysis on clr-transformed density functions.

The natural and logical step in smoothing density functions is thus just to express the data in the L^2 space of clr-transformed densities and to perform the computations there.
Consequently, it is guaranteed that all the theoretical properties hold and no additional reasoning is necessary.
Nevertheless, the above condition has to be taken into consideration as the smoothing is estimated.

1.2 B-spline representation

The main goal in the preprocessing of discretized distributional observations is finding a smoothed version of histogram data, in other words going from compositional functional data to smooth density functions.

In order to do this, [4] works with the B-spline basis function system.
Spline functions are often chosen for the approximation of non-periodic functional data, thanks their fast computation and flexibility.
The most famous system of spline for this kind of problem is the smoothing B-spline basis system, developed by de Boor ([7]). The choice is motivated by the fact that the resulting coefficients of basis functions can be directly used for statistical analysis.

We recall here the basic facts about B-splines, using the same notation of the reference paper [4].

Let u, v the extremals of the support I, $\Delta\lambda := \lambda_0 = u < \lambda_1 < \cdots < \lambda_g < v = \lambda_{g+1}$ a sequence of knots, and $S_k^{\Delta\lambda}[u, v]$ the vector space of polynomial

splines of degree k defined on the interval $[u, v]$ using the knots $\Delta\lambda$. It is known that $dim(S_k^{\Delta\lambda}[u, v]) = g + k + 1$.
A spline in this space has the following form:

$$s_k(\mathbf{x}) = \sum_{i=-k}^{g} b_i B_i^{k+1}(\mathbf{x})$$

where B_i^{k+1} are B-splines of degree k and form a basis in $S_k^{\Delta\lambda}[u, v]$.
For this representation, additional knots are required:

$$\lambda_{-k} = \cdots = \lambda_{-1} = \lambda_0, \quad \lambda_{g+1} = \cdots = \lambda_{g+k+1}.$$

In matrix notation, the spline can be rewritten as:

$$s_k(\mathbf{x}) = \mathbf{C}_{k+1}(\mathbf{x})\mathbf{b},$$

where $\mathbf{C}_{k+1}(\mathbf{x})$ is the *collocation matrix*, defined as

$$\mathbf{C}_{k+1}(\mathbf{x}) = \begin{bmatrix} B_{-k}^{k+1}(x_1) & \dots & B_g^{k+1}(x_1) \\ \vdots & \ddots & \vdots \\ B_{-k}^{k+1}(x_n) & \dots & B_g^{k+1}(x_n) \end{bmatrix} \in \mathbb{R}^{n,g+k+1}$$

and $\mathbf{b} = [b_{-k}, \dots, b_g]^T$ is *the vector of B-spline coefficients* of $s_k(\mathbf{x})$.
Moreover, using properties of B-splines, it is possible to write the derivative of order l of the spline $s_k(\mathbf{x})$ as

$$s_k^{(l)}(\mathbf{x}) = \mathbf{C}_{k+1-l}(\mathbf{x})\, \mathbf{b}^{(l)},$$

where $\mathbf{b}^{(l)}$ can be computed recursively:

$$\mathbf{b}^{(l)} = \mathbf{D}_l \mathbf{L}_l \mathbf{b}^{(l-1)} = \mathbf{D}_l \mathbf{L}_l \dots \mathbf{D}_1 \mathbf{L}_1 \mathbf{b} = \mathbf{S}_l \mathbf{b},$$

$$\mathbf{D}_j = (k + 1 - j)\, \mathrm{diag}(d_{-k+j}, \dots, d_g)$$

$$\text{with } d_i = \frac{1}{\lambda_{i+k+1-j} - \lambda_i} \qquad \forall i = -k + j, \dots, g,$$

$$\mathbf{L}_j = \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix} \in \mathbb{R}^{g+k+1-j,g+k+2-j}.$$

1.3 The optimal smoothing problem

Our goal is to find a function $f(x)$ that is a smooth approximation, close enough to the given data points (x_i, y_i). Hence $f(x)$ has to fulfil the following optimization problem:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & \int_{x_1}^{x_n} [f^{(l)}(x)]^2 \, dx \\ \text{subject to} \quad & \sum_{i=1}^{n} [w_i(y_i - f(x_i))]^2 \leq S \end{aligned}$$

where the objective function is a measure of non-smoothness of $f(x)$ and the constraint is a measure of the closeness of fit that takes into account data accuracy using weights.
The solution is known to be a natural spline $s_k(\mathbf{x})$ of degree $k = 2l - 1$. It follows that, for $l \geq 2$,

$$s_k^{(l+j)}(x_1) = s_k^{(l+j)}(x_n) = 0, \quad j = 0, 1, \ldots, l-2.$$

In order to find $\mathbf{b}$, we can plug this result into the dual problem:

$$\underset{f}{\text{minimize}} \quad J_l(f)$$

$$\text{where} \qquad J_l(f) := \int_{x_1}^{x_n} [f^{(l)}(x)]^2 \, dx + \alpha \sum_{i=1}^{n} [w_i(y_i - f(x_i))]^2$$

Given data (x_i, y_i), $u \leq x_i \leq v$, stored as $\mathbf{x} = [x_1, \ldots, x_n]^\top$, $\mathbf{y} = [y_1, \ldots, y_n]^\top$, let $w_i \geq 0$, $i = 1, \ldots, n$ be the weights related to each observation, $\Delta\lambda$ the sequence of knots, $n \geq g + 1$ and given $\alpha \in (0, 1)$, the functional $J_l(f)$ can be rewritten using matrix notation:

$$J_l(\mathbf{b}) = \mathbf{b}^\top \mathbf{N}_{kl} \mathbf{b} + \alpha [\mathbf{y} - \mathbf{C}_{k+1}(\mathbf{x})\mathbf{b}]^\top \mathbf{W} [\mathbf{y} - \mathbf{C}_{k+1}(\mathbf{x})\mathbf{b}]$$

where $\mathbf{N}_{kl} = \mathbf{S}_l^\top \mathbf{M}_{kl} \, \mathbf{S}_l$ is positive semidefinite,

$$\mathbf{M}_{kl} = \begin{bmatrix} (B_{-k+l}^{k+1-l}, B_{-k+l}^{k+1-l}) & \ldots & (B_g^{k+1-l}, B_{-k+l}^{k+1-l}) \\ \vdots & \ddots & \vdots \\ (B_{-k+l}^{k+1-l}, B_g^{k+1-l}) & \ldots & (B_g^{k+1-l}, B_g^{k+1-l}) \end{bmatrix} \in \mathbb{R}^{g+k+1-l, g+k+1-l}$$

and

$$(B_i^{k+1-l}, B_j^{k+1-l}) = \int_u^v B_i^{k+1-l}(x) B_j^{k+1-l}(x) \, dx.$$

$\mathbf{M}_{kl}$ is positive definite, because each element is the scalar product in $L^2([u, v])$ of basis functions.

Since we are working with smoothed clr-transformed density functions, we have to add the condition:

$$\int_u^v s_k(x)\,dx = 0.$$

From B-spline properties we know that

$$s_k(\mathbf{x}) = \sum_{i=-k}^{g} b_i B_i^{k+1}(\mathbf{x})$$

is the derivative of the spline

$$s_{k+1}(\mathbf{x}) = \sum_{i=-k-1}^{g} c_i B_i^{k+2}(\mathbf{x})$$

if

$$b_i = (k+1)\frac{c_i - c_{i-1}}{\lambda_{i+k+1} - \lambda_i} \quad \forall i = -k, \ldots, g.$$

Hence

$$0 = \int_u^v s_k(x)\,dx = [s_{k+1}(x)]_u^v = s_{k+1}(\lambda_{g+1}) - s_{k+1}(\lambda_0) = c_g - c_{-k-1},$$

therefore $c_g = c_{-k-1}$.
Finally, we have identified a relationship between $\mathbf{b}$ and $\mathbf{c}$:

$$\mathbf{b} = \mathbf{DKc},$$

where $\mathbf{D} = \mathbf{D}_0$ and

$$\mathbf{K} = \begin{bmatrix} 1 & 0 & 0 & \ldots & -1 \\ -1 & 1 & 0 & \ldots & 0 \\ 0 & -1 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & -1 & 1 \end{bmatrix} \in \mathbb{R}^{g+k+1, g+k+1}.$$

Hence the functional to minimize can be rewritten depending on $\mathbf{c}$:

$$J_l(\mathbf{c}) = (\mathbf{DKc})^\top \mathbf{N}_{kl} \mathbf{DKc} + \alpha[\mathbf{y} - \mathbf{C}_{k+1}(\mathbf{x})\mathbf{DKc}]^\top \mathbf{W}[\mathbf{y} - \mathbf{C}_{k+1}(\mathbf{x})\mathbf{DKc}]$$

In order to find the minimum, we set to 0 the derivative of $J_l(\mathbf{c})$ w.r.t. $\mathbf{c}$ and we obtain:

$$[\alpha^{-1}(\mathbf{DK})^\top \mathbf{N}_{kl}\mathbf{DK} + (\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK})^\top \mathbf{W}\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK}]\mathbf{c} = (\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK})^\top \mathbf{W}\mathbf{y}.$$

If $\alpha^{-1}(\mathbf{DK})^\top \mathbf{N}_{kl}\mathbf{DK} + (\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK})^\top \mathbf{W}\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK}$ is invertible, then there exists a unique solution $\mathbf{c}^\star$:

$$\mathbf{c}^\star = [\alpha^{-1}(\mathbf{DK})^\top \mathbf{N}_{kl}\mathbf{DK} + (\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK})^\top \mathbf{W}\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK}]^{-1}\mathbf{K}^\top\mathbf{D}^\top\mathbf{C}_{k+1}^\top\mathbf{W}\mathbf{y},$$

otherwise we can find a minimum norm solution

$$\mathbf{c}^\star = [\alpha^{-1}(\mathbf{DK})^\top \mathbf{N}_{kl}\mathbf{DK} + (\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK})^\top \mathbf{W}\mathbf{C}_{k+1}(\mathbf{x})\mathbf{DK}]_m^-\mathbf{K}^\top\mathbf{D}^\top\mathbf{C}_{k+1}^\top\mathbf{W}\mathbf{y}.$$

In both cases the vector of B-spline coefficients $\mathbf{b}^\star$ is obtained by:

$$\mathbf{b}^\star = \mathbf{DK}\mathbf{c}^\star.$$

Chapter 2

Implementation and code structure

2.1 Overview

We briefly explain here what kind of data are expected as input, the general workflow and the output of the software. In the following, more detailed implementation is discussed with focus on the R interface in section 2.2 and the shared-memory parallelization in 2.4.

2.1.1 Input data

User provides a dataset of observations, each row a statistical unit, all belonging to the same statistical model and with the same length.
Those are the data for our problem and they fulfil hypothesis of compositional data observation, e.g. sum constraint. Using the histogram example, each row consist of a histogram and all the histograms come from the same underlying model, e.g. proportion of annual income aggregated in classes for different regions of a given country as in [4]. The values of a given row correspond to the "height" of each class of the histogram.

User provides also what in splines lexicon are called the *control points*, i.e. the middle point abscissa of each class of the histogram. Those abscissa are the same for all the rows of the dataset.

Strictly related to the optimal smoothing problem, other parameters must be provided: spline degree k, penalization order l, smoothness tuning parameter α and spline knots. Spline knots can be passed as a given vector or they can be built by default equispaced given the size.

2.1.2 Count zeros: the Bayesian-multiplicative treatment

In order to perform the clr map we have to compute a logarithm, so we have to ensure that all data are strictly positive. But some observation could be zero for different reason.
In this framework we consider just the count-zero case using the lexicon of [5], i.e. unobserved positive values that may have been observed with a larger number of trials or with a different sampling design.

We decided to rely on the Bayesian-multiplicative (BM) treatment described in [5], which basically use a conjugate Multinomial-Dirichlet model for the observation with uniform prior setting $t = (1/D, .., 1/D)$:

$$\boldsymbol{C}_i|\boldsymbol{\pi} \sim Multi(n_i, \boldsymbol{\pi})$$

$$\boldsymbol{\pi} \sim Dir(D, s\boldsymbol{t})$$

where D is the dimension and s tunes the strength of the prior. Using different values for s, we distinguish different priors, in particular we let the user choose among the ones showed in table 2.1.

Prior	s	$\alpha_j = s \cdot t_j$	$\frac{c_j + s \cdot t_j}{n + s}$	$c_j = 0 \to t_j \cdot \frac{s}{n+s}$
Haldane	0	0	c_j/n	0
Perks	1	$\frac{1}{D}$	$\frac{c_j + 1/D}{n+1}$	$\frac{1}{D(n+1)}$
Jeffreys	$\frac{D}{2}$	$\frac{1}{2}$	$\frac{c_j + 1/2}{n + D/2}$	$\frac{1}{2n+D}$
Bayes-Laplace	D	1	$\frac{c_j + 1}{n + D}$	$\frac{1}{n+D}$
Square root (SQ)	$\sqrt{n}$	$\sqrt{n}/D$	$\frac{c_j + \sqrt{n}/D}{n + \sqrt{n}}$	$\frac{1}{D(\sqrt{n}+1)}$

Figure 2.1: BM priors (source [5]): $\boldsymbol{c}_i = (c_{i1}, .., c_{iD})$ is the observed vector with some count-zeros, while $n_i = \sum_{j=1}^{D} c_{ij}$ is the total mass. Keep in mind that $t_i = 1/D \quad \forall i$ since uniform priors are assumed.

Finally the BM treatment "corrects" the count-zeros using the mean Bayesian posterior estimate. So if $\boldsymbol{c}_i = (c_{i1}, .., c_{iD})$ is the observed vector with some count-zeros, defined $n_i = \sum_{j=1}^{D} c_{ij}$ the total mass, we use the posterior estimate formula on $\boldsymbol{x}_i = \boldsymbol{c}_i/n_i$, i.e. we replace $\boldsymbol{x}_i$ with $\boldsymbol{r}_i$ defined as:

$$r_{ij} = \begin{cases} t_{ij} \frac{s_i}{n_i + s_i}, & \text{if } x_{ij} = 0 \\ x_{ij}(1 - \sum_{k:x_{ik}=0} t_{ik} \frac{s_i}{n_i+s_i}), & \text{if } x_{ij} > 0 \end{cases}$$

This has been implemented in the `zeros` header and source files.

2.1.3 Workflow

Parameters of the spline problem, including knots and control points, are stored in `densityEstimator` class. This class contains all the fixed information of the problem, common to every row of the dataset to be processed. Through corresponding methods all the matrices requested to build the linear system described previously are computed and stored using `Eigen` in such class.

`dataManager` class deals with data. It reads a data-row, perform the zeros treatment and through the `pacs` method solve the B-spline problem calling the `solve` method of the `densityEstimator` object. The result is written in a proper `Eigen` structure.
Using the `plotData` method the resulting spline is finally evaluated in a given grid of points.

The reason of this structure of the code is due to the parallelization and it will be better understood in section 2.4.

2.1.4 Output

Output consist in B-spline coefficients for each statistical unit and possibly a grid of values in which resulting smoothing functions are evaluated at.
Size of the grid can be given as input parameter, otherwise it is handled by default.

2.2 R/C++ interface

2.2.1 `.Call` and `RcppEigen`

Code structures, implementation choices and C++ programming style reflect the necessity to develop an efficient implementation of the discussed density estimation method together with the flexibility and the comfort of the R environment for further analysis.
All of that is done via the R package facility with our `splineDensity` package available in the git repository (see section 4 for installation details).

The interface used to execute C++ code in R is `.Call`, described in [1]. Let us provide a simple example of how it works. Consider a simple C++ function. It looks like this:

```
#include <R.h>
#include <Rinternals.h>
extern "C"{
SEXP add(SEXP a, SEXP b) {
```

```
SEXP result = PROTECT(allocVector(REALSXP , 1)); REAL(result
    )[0] = asReal(a) + asReal(b); UNPROTECT (1);
return result;
} }
```

and this is what we run in R:

```
dyn.load("mysharedlib.so")
add <- function(a, b) { .Call("add", a, b)
}
```

After we compile the C++ code in a dynamic library, via command line with `R CMD SHLIB` or using the compiler directly, we can load the library in the R environment using `dyn.load()`.
Once our object is loaded, we can directly use the C++ function with `.Call("add",a,b)`, i.e. defining an R function named "add".

In the C++ code all kind of object that come from R, or should be returned to R, are of the `SEXP` type. The powerful advantage of this interface is that all R variables are not passed by copy, but by a particular kind of pointer (the `SEXP` type). As a consequence, to prevent that R garbage collection destroys objects which should be returned, it is important to use the `PROTECT()` function.
Finally, all the code that R should be able to use (e.g. the defined function) must be included in the `extern "C"` scope.
`SEXP` object can be copied in C++ objects through some specific functions defined in the R header files as `REAL()` or `INTEGER()`.

We decided also to rely on *Rcpp* and *RcppEigen* libraries (a lot of work on the topic, especially done by Eddelbuettel, is available, among the others [3]) which easily allow to interface R matrixes with `Eigen` structures, combining the `Eigen::Map` method with the `Rcpp::as` function, e.g.:

```
SEXP Rmatrix;
Eigen::Map<Eigen::MatrixXd>  newMatrix(as<Eigen::Map<Eigen::
    MatrixXd>> (Rmatrix));
```

Note that this leads to have a complex type for the `newMatrix` object, working with a row of it you have to consider the following type:

```
Eigen::Block<Eigen::Map<Eigen::Matrix<double, -1, -1>,0,
    Eigen::Stride<0, 0> >, 1, -1, false>
```

2.2.2 Debug

It could happen that R session crashes due to some memory leakage, in such cases it could be hard to find the problem since the lack of reports from the

R session. We have find really helpful the debug mode, which can be started with the following command from the terminal:

```
R -d <debugger>
```

where `-d` stays for "debug". In particular we relied a lot on the famous memory error detector *valgrind* and we found it really useful.

2.3 C++ implementation

2.3.1 `parametersManager`, `dataManager` and `densityEstimator` classes

- `parametersManager`: it includes problem parameters shared among all statistical units such as control points, knots, spline degree, penalization order and α value. No objects of this class are actually declared, but it is conceptually useful to distinguish it from its child `densityEstimator`.

- `densityEstimator`: it inherits publicly from `parametersManager`. It has methods to compute all the necessary matrixes and the final linear system described in 1.3. It has a method to evaluate the J functional and the `solve` method which is described in 2.3.2. Such method is used on each row (parameters and problem matrix are common) and returns the estimated B-spline coefficients.

- `dataManager`: it is used to interface the data with the smoothing process. It reads a row of the data set and then performs the BM treatment (section 2.1.2). The `pacs` method has the `densityEstimator` object as argument and calls the `solve` method saving the output (spline coefficients) in a given matrix.
  It includes also methods to evaluate the estimated function in a grid of points both in the clr space and the reference space, respectively `plotData` and `plotData_Clr` methods.

2.3.2 Solve method

As already discussed in section 1.3, the smoothing problem could not have a solution, in such case a minimum norm solution should be provided.
We decided to rely on the `FullPivHouseholderQR` solver provided by the `Eigen` library, which apply Householder rank-revealing QR decomposition with full pivoting. This decomposition performs a very prudent full pivoting in order to be rank-revealing and achieve optimal numerical stability.

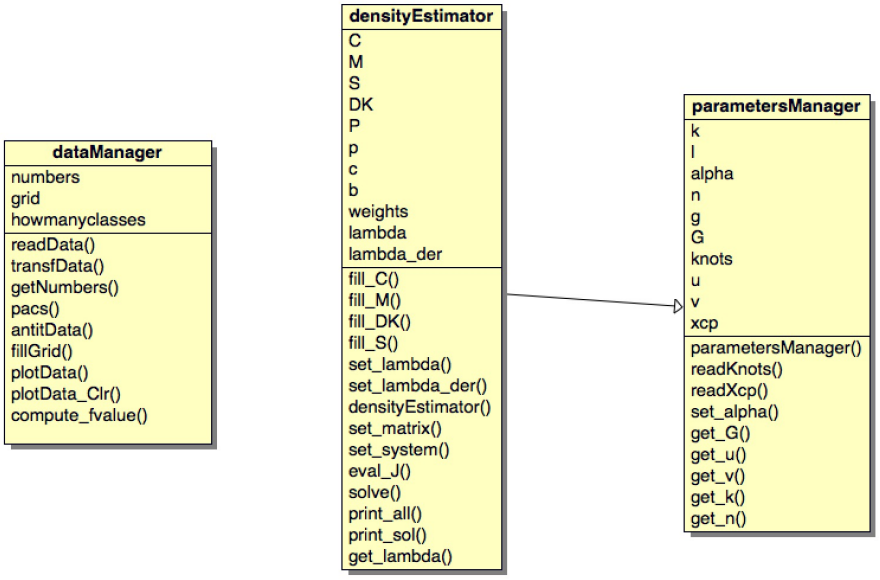


Figure 2.2: Classes diagram

Rank-revealing property is very important in case we want to find a solution when the problem is not solvable. Kernel knowledge allows us to analytically find a minimum norm solution in case its dimension is 1 or 2:

$$A\boldsymbol{x} = \boldsymbol{b}$$

If $\dim(\mathrm{Ker}(A))= 1$,

$$\boldsymbol{k} \in \mathrm{KerA} \quad \text{and} \quad \hat{\boldsymbol{x}} \quad \text{found solution}$$

$$c = \frac{< \hat{\boldsymbol{x}}, \boldsymbol{k} >}{< \boldsymbol{k}, \boldsymbol{k} >}$$

$$\boldsymbol{x}_{\min} = \hat{\boldsymbol{x}} - c\boldsymbol{k}$$

else if $\dim(\mathrm{Ker}(A))= 2$, let

$$\boldsymbol{k_1}, \boldsymbol{k_2} \in \mathrm{KerA} \quad \text{and} \quad \hat{\boldsymbol{x}} \quad \text{found solution,}$$

then, doing elementary computations,

$$\left(\|\boldsymbol{k_2}\|^2 - \frac{< \boldsymbol{k_1}, \boldsymbol{k_2} >^2}{< \boldsymbol{k_1}, \boldsymbol{k_1} >} \right) c_2 = < \hat{\boldsymbol{x}}, \boldsymbol{k_2} - \frac{< \boldsymbol{k_1}, \boldsymbol{k_2} >}{< \boldsymbol{k_1}, \boldsymbol{k_1} >} \boldsymbol{k_1} >$$

$$c_1 = -\frac{< \boldsymbol{k_1}, \boldsymbol{k_2} >}{< \boldsymbol{k_1}, \boldsymbol{k_1} >} c_2 + \frac{< \hat{\boldsymbol{x}}, \boldsymbol{k_1} >}{< \boldsymbol{k_1}, \boldsymbol{k_1} >}$$

$$\boldsymbol{x}_{\min} = \hat{\boldsymbol{x}} - c_1\boldsymbol{k_1} - c_2\boldsymbol{k_2}$$

Otherwise, when kernel dimension is greater than 2 we decided to apply Tychonoff regularization ($\lambda = 0.01$) to the matrix problem in order to achieve full rank and find a solution. Obviously in such case we will be solving an approximation of our problem:

$$\hat{A} = A + \lambda I$$

$$\text{new problem} \quad \hat{A}\boldsymbol{x} = \boldsymbol{b}$$

All that is done by the `solve` method of the `densityEstimator` object through a `switch` statement over the kernel dimension.

2.3.3 Cross-validation function

In addition to the resolution of the optimal problem, we were asked to provide a cross-validation method to asses the best α among a vector of choices. Such validation has been designed to be a sort of leave-one-out (LOO) algorithm: one at a time each control point (c.p.), which is not an extremal, is left out, the B-spline is estimated and the error is computed as the sum of the differences between the left out values and the corresponding estimate given by the B-spline averaged among all the rows:

$$\forall \alpha_i \in \boldsymbol{\alpha} \quad \text{error}_{LOO}(\alpha_i) = \sum_{k=1}^{\#rows} \sum_{j=1}^{\#c.p.} \frac{|y_{k,j} - \hat{y}_{k,j}|}{\#\text{rows}}$$

After that, the best α value is found and cross-validation errors among with the evaluations of the J functional of the minimization problem are returned to the user, with best choice highlighted.

2.4 OpenMP parallelization

In section 2.1.3 it was mentioned the fact that we perform the same operation over each row of the given data set in a `for` loop.
Precisely, the matrix associated to the linear system is fixed and what changes is the constant term, which depends on the data. Therefore, after constructing the problem matrix, we loop over the row in the following way:

```
for(std::size_t i = 0; i < nrow; i++)
{
  obj.readData(data.row(i), prior);
  obj.transfData();
```

```
    obj.pacs(dens,bsplineMat.row(i));
    obj.plotData(dens, numPoints, bsplineMat.row(i),
        yvalueMat.row(i));
    obj.plotData_Clr(dens, numPoints, bsplineMat.row(i),
        yvalueMatClr.row(i));
}
```

where `obj` is a `dataManager` object and `dens` a `densityEstimator` object.

This commands are easily parallelizable in shared memory framework as OpenMP, it is enough to ask each thread to solve the problem for a subset of rows.

Moreover, OpenMP gives the possibility to make same objects private to each thread through the `private` keyword. If those objects exist before the `#pragma` statement then the `firstprivate` keyword should be used to initialize them with their values.
In a simple way, the previous code becomes the following:

```
#pragma omp parallel private(obj) firstprivate(dens)
 {
#pragma omp for
    for(std::size_t i = 0; i < nrow; i++)
    {
      obj.readData(data.row(i), prior);
      obj.transfData();
      obj.pacs(dens, bsplineMat.row(i));
      obj.plotData(dens, numPoints, bsplineMat.row(i),
          yvalueMat.row(i));
      obj.plotData_Clr(dens, numPoints, bsplineMat.row(i),
          yvalueMatClr.row(i));
   }
 }
```

Thanks to this multithreads implementation we have achieved a satisfactory speed-up in the computations when a large data set is provided.

We tested the code on a dataset of 10^6 rows and 12 columns, i.e. 12 histogram cells, on a 12 cores Intel(R) Core(TM) i7-3930K CPU @ 3.20GHz machine, leaving out one core dedicated to the OS.
Results are shown in figure 2.3.

Cross-validation function (section 2.3.3) has been parallelized over the different α values:

```
#pragma omp parallel private(obj) firstprivate(dens)
 {
    Eigen::MatrixXd threadBsplineMat(nrow,dens.get_G());
    Eigen::ArrayXd N;
```

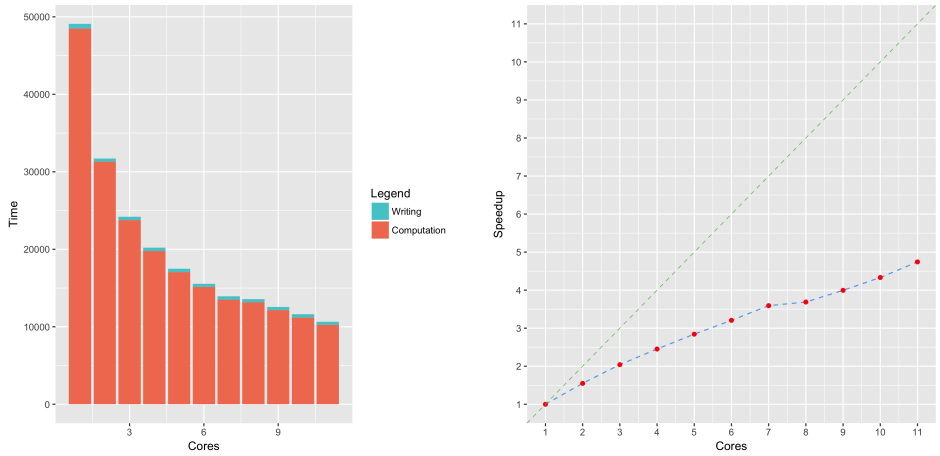


Figure 2.3: Intel(R) i7-3930K CPU @ 3.20GHz: on the left, the time (milliseconds) required by the software divided in computation time (to solve the problem) and writing time (to write the solution in R objects, this is done by just one thread). On the right, speed-up plot.

```
    int span;
    long double fvalue;

     #pragma omp for
    for(std::size_t z = 0; z < alpha_size; z++)
    {
      dens.set_alpha(alpha[z]);
      for(std::size_t i = 0; i < nrow; i++)
      {
        for(std::size_t j = 1; j < ncol-1; j++)
        {
          // LOO-CV, leave out column j, fit spline and
              compute error
          dens.readXcp(Xcp,Xcpsize,j);
          dens.set_matrix();
          dens.set_system();
          obj.readData(data.row(i),prior,j);
          obj.transfData();
          obj.pacs(dens, threadBsplineMat.row(i));
          Jvalues(z)+=dens.eval_J(obj.getNumbers())/nrow;

          // computing error using spline coefficients,
              fvalue is the predicted value
          // [...]
        }
      }
    }
}
```

This time it was necessary to reserve each thread an `Eigen` matrix (`threadBsplineMat`) where to store the result of its partial computation. Note that every object defined inside the `#pragma omp` statement is automatically private to each thread.
Comparison to asses the best α value is done outside the parallelized scope by one single thread.

Chapter 3

Practical examples

3.1 Application to particle-size curves in heterogeneous aquifers

We consider here a data set, kindly provided by professor Menafoglio, obtained at the Lauswiesen site, located in the Neckar river valley, near the city of Tübingen (Germany).
The subsurface system in the area has been characterized through extensive information obtained at a number of boreholes, which are employed to perform sedimentological as well as hydraulic analyses.

Of specific relevance to our study are the available 406 particle-size curves (PSCs) sampled along 12 fully penetrating vertical boreholes. Figure 3.1 depicts a sketch of the borehole network and sampling locations at the site.

The aquifer is made up by alluvial material overlain by stiff silty clay and underlain by hard silty clay. The site characterization has been based on stratigraphic information collected at a set of monitoring and pumping wells.
PSCs describe the local distributions of grain sizes within the aquifer system. A set of twelve discrete sieve diameters (i.e., from a minimum of 0.063 up to 100.0 mm) were employed to reconstruct these curves by way of grain sieve analysis on soil samples, as in 3.2.

From the mathematical viewpoint, a particle-size density is a probability density function, associated with the distribution of particle sizes within a given soil sample.

As such, available data consist of a set of constrained curves, spatially distributed. The statistical characterization of PSCs plays a key role in the classification of soil types, for inferring hydraulic parameters (e.g., porosity,

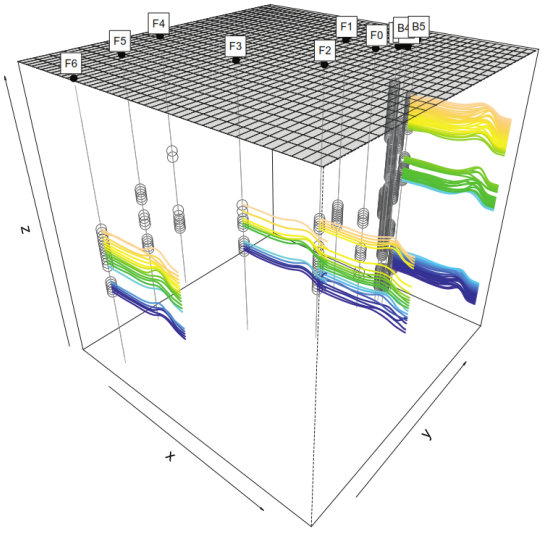


Figure 3.1: Three-dimensional representation of particle-size densities at the Lauswiesen site. Grey points represent measurement locations, colored curves represent a subset of the dataset of PSCs. Colors indicate the depth of the sampling locations along the borehole.

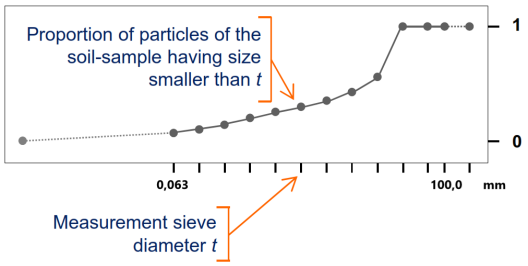


Figure 3.2: Example of particle-size raw data from sieze measurements, from clay and silt grains up to granules and pebbles.
We interpret PSCs as cumulative distribution functions and their derivatives as probability density functions. Grain sieve analysis of soil samples, indeed, yields a discrete representation of the curves by measuring selected particle diameters which, in turn, correspond to quantiles of the particle-size curve.

permeability and hydraulic conductivity), and reconstructing the internal architecture of the groundwater system.

In this vein, the study of PSCs may be concerned, as in [6], with a geostatistical analysis, e.g. in order to identify clusters which represent the occurrence of different soil types and distinguish geomaterials at the site, and to characterize the spatial distribution of each identified textural class, and to provide Kriging estimates of the heterogeneous distribution of PSCs at unsampled locations.

Classification of aquifer geomaterials and estimation of their spatial arrangement is relevant to properly reconstruct the internal architecture of groundwater systems which can play a critical role in controlling contaminant spreading on different scales.

As already pointed out, in the present case study an estimate of the particle-size curve at a given spatial location s is available only for a set of $N = 12$ sieve diameters.

To achieve any of the goals listed above, as in several other practical field situations, preprocessing of the raw data is required to obtain smooth estimates of the PSCs and associated densities.

The support of the PSCs has been assumed to be compact, upon setting the data support as $I = [\log(0.001), \log(200)]$, consistent with the type of lithology at the site.

Figure 3.3 depicts the resulting smoothed curves, obtained applying the algorithm described in chapter 1.

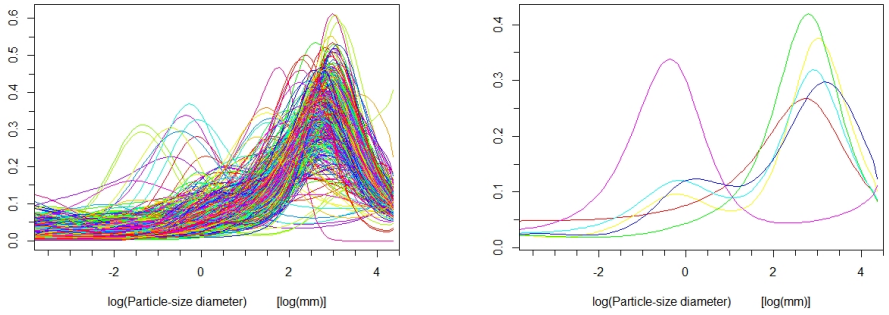


Figure 3.3: Smoothed particle-size densities: on the left, all in one picture, and, on the rigth, just a few of them, chosen quite randomly, to highligth the variety in the soil composition for different sampling locations

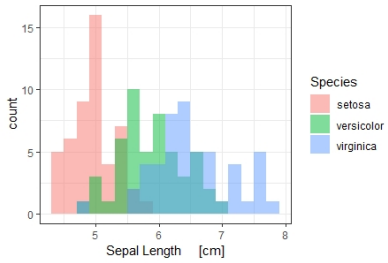


Figure 3.4: Histogram of the variable Sepal Length for the Iris data set

3.2 Application to the well-known Iris flower data set

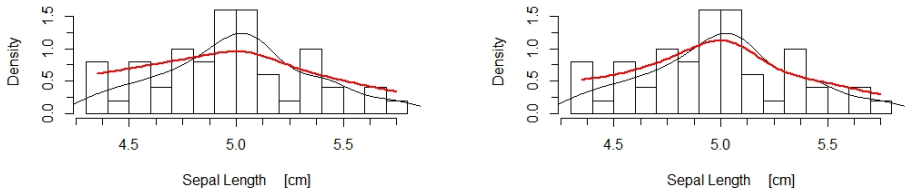
The Iris flower data set is a multivariate data set introduced in the 1930s by Sir Ronald Fisher, who collected the data to quantify the morphologic variation of Iris flowers of three related species.

The dataset consists of 50 samples from each species of Iris flowers (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample, they are the length and the width of sepal and petal, in centimeters.

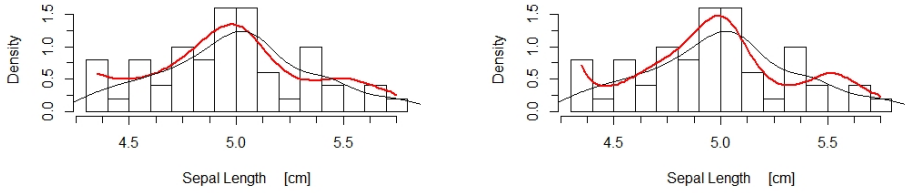
We focus our attention on the variable related to the length of sepal, shown in figure 3.4.

The results are represented in figures 3.5 and 3.6: to be precise, the former shows how the algorithm works for different values of α when applied to the measures related to the Setosa species, the latter presents just the other two species for a fixed α.

It is worth remembering that the parameter α is related to the fidelity term - the higher it is, the more the output density is forced to stay near to the given values of the histogram.

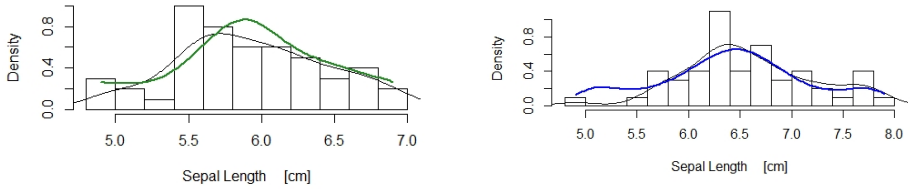


$\alpha = 10$ $\alpha = 100$



$\alpha = 1000$ $\alpha = 10000$

Figure 3.5: Data related to the Iris Setosa and the estimated density for different values of α. The black line represents the "real" density, obtained from all the data through the kernel estimate. The coloured line is the density estimated with the algorithm - which is applied to the histograms.



Iris Versicolor *Iris Virginica*

Figure 3.6: Data related to the other two species of Iris. α is fixed and set to 1000.

Chapter 4

Installation

The source code of this R package can be found and downloaded from `https://github.com/fpavone/pacs_spline_density` in the branch *master*. There are different ways of installing it on your machine.

The first one requires the use of `devtools`, a popular R package. It will also take care of installing all dependencies (Rcpp and RcppEigen) and install the splineDensity package in the same directories of all the other R packages installed.
In R, use the commands:

```
library(devtools)
install_github("fpavone/pacs_spline_density")
```

Note that it is required to have the unzip option as `internal`, you can check the unzip setting of your machine with `getOption("unzip")` and possibly change it with `options(unzip = "internal")`.

As an alternative, still using `devtools`, it's possible to install the package from the terminal. Once the source code is downloaded and unzipped, it is enough to run from the package root folder the following commands:

```
R -e "library(devtools); install()" --silent
```

To build the documentation in Roxygen, then:

```
R -e "library(devtools); document()" --silent
```

This second command will create a subfolder named man where the .Rd files will be stored. This will be useful when calling for "help" from R (e.g., ?smoothSplines).

A different way to complete the installation does not require any additional package, but will throw errors if the packages required for the functioning of splineDensity are not installed.
From the terminal, after downloading the code, you need to run:

```
R CMD INSTALL -l <path name of the R library tree>
   <path name of the package to be installed>
```

To test the successful installation of the package, it is possible to run the example in the subdirectory tests, either from terminal

```
Rscript test_particle.r
```

or loading it directly in R. Otherwise the R command `example(smoothSplines)` does the same job.

The `Doxygen` documentation of the C++ code can be built from the *src* folder using the provided *Doxyfile*. To run Doxygen type:

```
doxygen Doxyfile
```

Keep in mind that, in order to create a new documentation with different settings, you need to create a new configuration file, i.e.*Doxyfile*, with:

```
doxygen -g <config-file>
```

that can be edited to customize the output.

To get the Reference Manual, then go to the *latex* subfolder that has been generated and type `make`.

4.1 How to enable OpenMP parallelization

If your compiler supports OpenMP parallelization, once you have downloaded the splineDensity package, look for the *Makevars* in the *src* subfolder.
Set the `OPENMP` macro defined in it to a non-empty value to activate the parallelization, e.g. `OPENMP = 1`. By default, the option is disabled.

Now, follow the instructions as above to install the package.
Once finished, you have the opportunity to exploit the benefits that only a parallel implementation can provide.

4.2 Installing devtools

It may be possible to have problems in installing the R package `devtools` in Ubuntu or other Linux distributions.

If the installation throws the following errors:

```
ERROR: dependency ‘curl’ is not available for package ‘httr’
ERROR: dependencies ‘httr’, ‘memoise’ are not available
for package ‘devtool’...
Warning messages:
1: In install.packages("devtools"):
installation of package ‘httr’ had non-zero exit status
2: In install.packages("devtools"):
installation of package ‘devtools’ had non-zero exit status
```

it may be solved by installing the following libraries: `sudo yum -y install libcurl libcurl-devel` in CentOS and `sudo apt-get install libcurl4-openssl-dev libssl-dev` in Ubuntu.

Bibliography

[1] Writing R extensions. `https://cran.r-project.org/doc/manuals/R-exts.pdf`.

[2] J. Aitchison. The statistical analysis of compositional data. *Journal of the Royal Statistical Society. Series B*, 44(2):139–177, 1982.

[3] D. Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, 2013.

[4] J. Machalová, K. Hron, and G.S. Monti. Preprocessing of centred logratio transformed density functions using smoothing splines. *Journal of Applied Statistics*, 43:1419–1435, 2016.

[5] J.A. Martín-Fernández, K. Hron, M. Templ, P. Filzmoser, and J. Palarea-Albaladejo. Bayesian-multiplicative treatment of count zeros in compositional data sets. *Statistical Modelling*, 15(2):134–158, 2015.

[6] A. Menafoglio, A. Guadagnini, and P. Secchi. Stochastic simulation of soil particle-size curves in heterogeneous aquifer systems through a bayes space approach. *Water Resources Research*, 52:5708–5726, 2016b.

[7] J.O. Ramsay and B.W. Silverman. *Functional Data Analysis*. Springer, 2005.

[8] K.G. Van den Boogaart, J.J. Egozcue, and V. Pawlowsky-Glahn. Bayes hilbert spaces. *Aust. N. Z. Journal of Statistics*, 56(2):171–194, 2014.