GSR-PDE

Alberto Colombo, Giulio Perin

Tutor: Prof.ssa M.L. Sangalli, Dott.ssa E. Arnone

June 2020

Contents

1 Introduction **5**

2 Theoretical framework **6**
2.1 GAM . . . 7
2.2 SR-PDE . . . 7
2.3 GSR-PDE . . . 8
2.4 Functional PIRLS Algorithm . . . 9
2.4.1 Iterative scheme for FPIRLS . . . 10
2.4.2 Solution for FPIRLS - Laplacian case . . . 10
2.4.3 Solution for FPIRLS - general case discussion . . . 12
2.5 Numerical solution with FEM . . . 13
2.5.1 Woodbury decomposition . . . 14
2.5.2 Areal data . . . 15
2.6 Selection of the smoothing parameter . . . 15
2.7 Scale parameter estimate . . . 16

3 Code Structure **17**
3.1 R/C++ interface . . . 17
3.2 Core method . . . 18
3.2.1 Interaction with `MixedFERegression` . . . 21
3.2.2 Data structure: `RegressionDataGAM` . . . 23
3.2.3 FPIRLS and the PDE generalization . . . 25
3.2.4 Factory object . . . 26
3.3 Scale parameter and variance estimates . . . 27
3.4 GCV computation . . . 28
3.5 Minor changes . . . 29
3.5.1 GCV bug fixing . . . 29
3.5.2 Warning message in GCV . . . 29
3.5.3 Areal option included . . . 29

4 Installation **30**
4.0.1 Installation issues . . . 30
4.1 Documentation with Doxygen . . . 30
4.2 Quick start . . . 31

5 Simulation study **33**
5.1 Simulation description . . . 33
5.2 Simulations . . . 34
5.2.1 Binomial 2D Square domain . . . 34
5.2.2 Gamma 2D Square domain . . . 37
5.2.3 Poisson 2D Square domain . . . 40
5.2.4 Exponential 2D Square domain . . . 43
5.2.5 Gamma 2D C-Shaped domain . . . 46

5.2.6 Poisson 2D Circular domain 49
5.2.7 Poisson 2.5D Hub domain 52
5.2.8 Gamma 3D Sphere 55
5.2.9 Poisson Areal 2D Square domain 59
5.3 R/C++ performance comparison 62
5.4 Bug fixing in `fdaPDE 1.1` library 63

6 Conclusions **64**

Appendices **66**

A typedefs **66**

B MixedFERegression proposal **66**

Abstract

Spatial regression with PDE (SR-PDE) penalization is a recent class of methods for the analysis of spatial and functional data over complex domains. Its main applicational toolbox is the `fdaPDE` package for the R software.
This work aims to extend the range of applicability of the software to the treatment of non-gaussian distributed observations. In particular a new method for spatial regression in the context of generalized linear models has been introduced. This method, for the first time, enables to model spatially distributed data having any distribution within the exponential family, considering a differential regularization, as in SR-PDE. The exponential family includes all well-known distributions, including continuous distributions, such as gamma and exponential and discrete distributions, such as binomial and poisson. This enormously extends the applicability of this class of methods. The estimation problem is solved using a combination of maximum likelihood and FEM discretization, and it is able to achieve the same broad flexibility currently available for the gaussian case.

1 Introduction

Linear models are statistical models in which a univariate response is modelled as the sum of a linear predictor and a zero mean random error term. The linear predictor depends on some predictor variables, measured with the response variable, and some unknown parameters, which must be estimated. A key feature of linear models is that the linear predictor depends linearly on these parameters and the random error term is assumed to be gaussian distributed.
Generalized linear models (GLMs) relax the strict linearity assumption of linear models, by allowing the expected value of the response to depend on a smooth monotonic function of the linear predictor (the so called *link function*). Similarly the assumption that the response is normally distributed is relaxed by allowing it to follow any distribution from the exponential family. Within this family we can find the most common distributions, both continuous and discrete, such as gamma distribution, commonly used to model waiting times and queues, the poisson distribution allowing the model of counting data and the binomial distribution, which enables the extension of the method also to classification problems.
A Generalized Additive Model (GAM) is a GLM in which part of the linear predictor is specified in terms of a sum of smooth functions of predictor variables. The exact parametric form of these functions is unknown, as is the degree of smoothness appropriate for each of them.
Linear models are often the default choice for many statistical applications and they led to the development of Spatial Regression with Partial Differential Equation penalization (SR-PDE), a class of spatial additive models for the gaussian case in which just one smooth function is considered, and it is somewhat constrained to fit a PDE from which its smoothness depends upon.
We studied a method to generalize the SR-PDE models to the GAM setting as proposed by [14] and we embed the proposed algorithm within the `fdaPDE` package, which is the standard reference library for handling SR-PDE models in the R software.
The `fdaPDE` library is written in C++ and designed with a particular emphasis on efficiency. Taking advantage of these facts, our algorithm presents a major improvement of performance w.r.t. the experimental version of [14], fully written in R language. Moreover, its inclusion in the `fdaPDE` package will be soon available on CRAN repository and we hope this will help broaden the usage of SR-PDE models within the R community.

This report is organized as follow. The first chapter will concisely explain the theoretical framework of this work and details the relevant results needed to fully comprehend the proposed algorithm. The second part will guide the reader inside the `fdaPDE` package code, highlighting the main contributions developed during the course of this project. A third section is then devoted to the analysis of the new package functionalities both in terms of performance and accuracy. Finally, a last chapter will briefly summarize our work and open a discussion about possible future directions of this project.

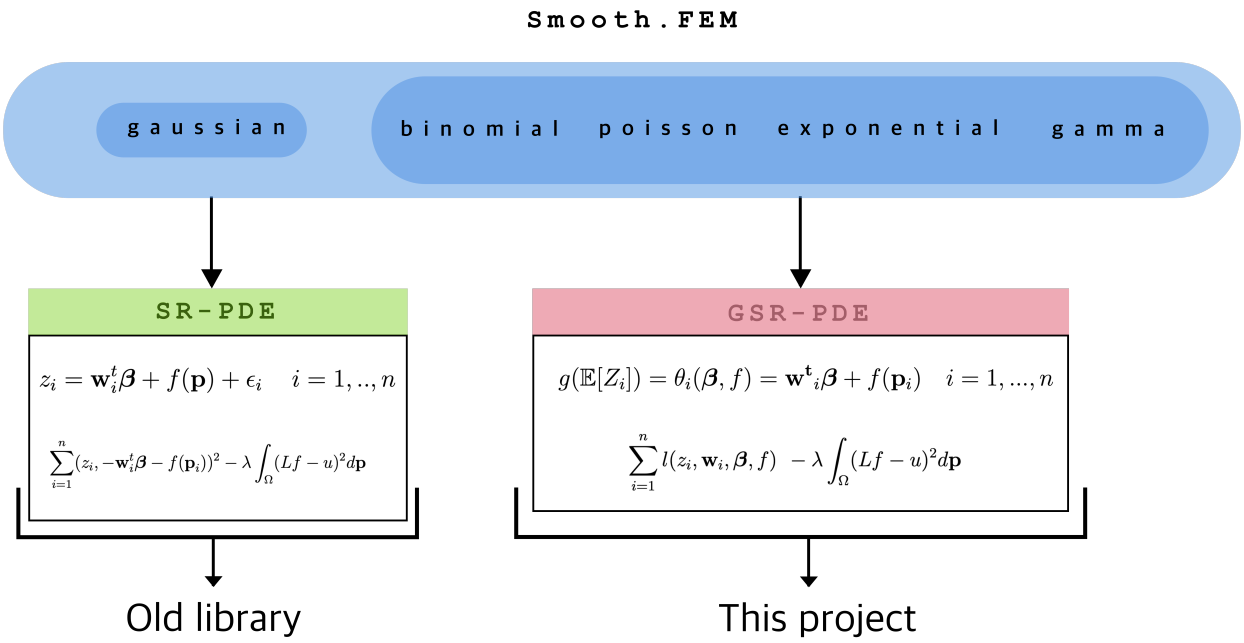


Figure 1: Model scheme: with GSR-PDE non-gaussian can be handle.

2 Theoretical framework

In this section we will describe the theoretical and computational framework for generalized additive models (GAMs) with spatial differential regularization. The main reference for for the model and the algorithm presented here can be found in [14].

In the following we will consider as parametrization of a distribution from the exponential family the one given by:

$$f_Y(y;\theta,\phi) = exp\{y\theta - b(\theta)/a(\phi) + c(\phi, y)\}$$

where

- $a(\cdot)$, $b(\cdot)$ and $c(\cdot)$ are functions subject to some regularity constraints;
- $b'(\theta) = g^{-1}(\theta)$, if $g(\cdot)$ is the canonical link function;
- we assume that $a(\phi) = \phi$, this being the case of the most common distributions in the exponential family;
- we denote by $V(\cdot)$ a function that satisfy $\text{var}(Y) = V(\mu)\phi$;
- the canonical parameter θ is a function of both $\boldsymbol{\beta} \in \mathbb{R}^q$ and $f \in \mathcal{F}$ as we will see in the following.

2.1 GAM

Generalized Additive Models have been proposed for the first time by Hastie and Tibshirani ([9], [10]) in the attempt to blend properties of generalized linear models (GLMs) and additive models. The GAM general structure has the form:

$$g(\mathbb{E}[Z_i]) = \theta_i = \mathbf{w}_i^t\boldsymbol{\beta} + f_1(w_{1i}) + f_2(w_{2i}) + f_3(w_{3i}, w_{4i}) + ... \quad i = 1, .., n$$

where Z_i is distributed according to a distribution in the exponential family.
It is thus composed by a parametric part $\mathbf{w}_i^t\boldsymbol{\beta}$, where $\mathbf{w}_i \in \mathbb{R}^q$ is the vector of covariates and $\boldsymbol{\beta}$ is an unknown parameter vector indicating the effect of the covariates on the value of θ_i, and a non parametric part composed by unknown smooth functions f_j of the covariates. The estimated parameter $\boldsymbol{\theta}$, commonly referred as *canonical parameter*, is then related with the expected value of Z_i through continuosly differentiable and striclty monotone function $g(\cdot)$ generally called *link function* in such a way that $\theta_i = g(\mathbb{E}[Z_i])$.

GAMs allow a rather flexible dependence specification of the model from covariates, however this flexibility requires to define both how to represent the smooth functions f_j and to choose how smooth they are. We will see in the following sections how these specifications have been addressed in our case. For a more structured and complete description of GAMs we refer our readers to [17].

2.2 SR-PDE

Spatial Regression with Partial Differential Equation regularization (SR-PDE) is a modern class of models focusing on the analysis of data displaying complex spatial or spatio-temporal dependencies. In particular SR-PDE consists in a regression model with a regularization term that involves a Partial Differential Equation (PDE). The basic idea is to exploit PDE modeling, extensively used in science and engineering, in order to include prior knowledge about the fenomenon under study and capturing anisotropies and non-stationarities in the data. The general SR-PDE model can be set up as follow.
Let $\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_n$ be n data locations scattered over a bounded spatial domain $\Omega \in \mathbb{R}^2$ with smooth boundary $\partial\Omega \in \mathcal{C}^2$. Let $z_i \in \mathbb{R}$ be the value of a variable of interest in the location $\mathbf{p}_i$ and, if present, let $\mathbf{w}_i$ be a vector of covariates associated with the observation z_i in $\mathbf{p}_i$. Consider the semi-parametric additive model

$$z_i = \mathbf{w}_i^t\boldsymbol{\beta} + f(\mathbf{p}) + \epsilon_i \quad i = 1, .., n \tag{1}$$

where $\boldsymbol{\beta} \in \mathbb{R}^q$ is an unknown vector of regression coefficients indicating the effect of the covariates on the variable of interest, $f : \Omega \to \mathbb{R}$ is an unknown deterministic field capturing the spatial relationship among data, and ϵ_i $i = 1...n$ are random gaussian errors with zero mean and finite constant variance. The estimation of the unknowkn $(\boldsymbol{\beta}, f)$ is achieved through the minimization of the following functional

$$\sum_{i=1}^{n}(z_i, -\mathbf{w}_i^t\boldsymbol{\beta} - f(\mathbf{p}_i))^2 - \lambda\int_\Omega (Lf - u)^2 d\mathbf{p} \tag{2}$$

where λ is a positive smoothing parameter and $(Lf - u)^2$ represent the misfit of f w.r.t. a generic PDE of the form $Lf = u$. Where the operator L can be expressed, in its general form, as

$$Lf = div(K\nabla f) + \boldsymbol{b} \cdot \nabla f + cf \tag{3}$$

The minimum of (2) is generally found by rewriting the functional in a suitable variational form. In particular the functional is well defined for $\boldsymbol{\beta} \in \mathbb{R}^q$ and $f \in H^2(\Omega)$. Moreover let $V(\Omega) \subseteq H^2(\Omega)$ the subspace of $H^2(\Omega)$ characterized by the chosen boundary conditions for the PDE and consider the problem under homogeneous boundary conditions and forcing terms (see [3], [4] for non-homogeneous BC and forcing terms), then we have

Proposition 1. *There exists a unique pair of estimators $(\hat{\boldsymbol{\beta}}, \hat{f}) \in \mathbb{R}^q \times V(\Omega)$ which minimizes (2). Moreover,*

$$\hat{\boldsymbol{\beta}} = (W^t W)^{-1} W^t (z - \hat{\mathbf{f}}_n) \tag{4}$$

and $\hat{f}$ satisfies

$$\mathbf{v}_n^t Q \hat{\mathbf{f}}_n + \lambda \int_\Omega (Lv)(L\hat{f}) = \mathbf{v}_n^t Q \mathbf{z}, \forall v \in V(\Omega) \tag{5}$$

where W is the design matrix with rows $\mathbf{w}_i$ and $Q = I - W(W^t W)^{-1} W^t$ is the matrix that projects onto the orthogonal complement of $\mathbb{R}^n$ with respect to the subspace of $\mathbb{R}^n$ spanned by the columns of W.

This results leads the functional minimization to a fourth order differential problem which solution cannot be found analytically. The most common, even if not the only possible (see [15] for an alternative solution via isogeometric analysis), approach is to discretize the variational problem (5) via Finite Element Method (FEM). The usage of numerical methods exploiting this technique has conduct to the development of the package `fdaPDE` for the `R` statistics software which constitutes the landscape in which our work has been done. In particular, `fdaPDE` has included, so far, only SR-PDE models with different settings, whereas our work aims to extend the package to deal with a more general class of models, called GSR-PDE (Figure 1), that we are going to introduce in the next section.

2.3 GSR-PDE

The SR-PDE model can be extended to any distribution within the exponential family, exploiting the GAM formulation in the SR-PDE framework. The aim of this work is to extend `fdaPDE` software to deal with this new class of models. Their formulation is as follow.
Let $Z_1, ...Z_n$ be independent random variable having a distribution that belongs to the exponential family and let $\mathbf{w}_i \in \mathbb{R}^q$ be the vector of covariates.
The model of the expected value of Z_i, at location $\mathbf{p}_i$ is given by

$$g(\mathbb{E}[Z_i]) = \theta_i(\boldsymbol{\beta}, f) = \mathbf{w^t}_i \boldsymbol{\beta} + f(\mathbf{p}_i) \quad i = 1, ..., n \tag{6}$$

where g is the link function associated with the considered distribution, $\boldsymbol{\beta}$ is the vector of covariates effect and f is a spatial field over Ω as in the previous section.
Following the same approach of SR-PDE model, we aim to estimate $\boldsymbol{\beta}$ and f maximizing the penalized log-likelihood:

$$\sum_{i=1}^{n} l(z_i, \mathbf{w}_i, \boldsymbol{\beta}, f) - \lambda \int_{\Omega} (Lf - u)^2 d\mathbf{p} \tag{7}$$

where $l(\cdot; \theta_i)$ is the log-likelihood for the considered distribution.

This expression coincides with the minimization of the regularized least-squares in (2) when the considered distribution is Gaussian. On the other hand this approach allow the choice of any distribution within the exponential family; hence, this model generalization broadens enormously the applicability of the proposed technique.
However, the algorithm used to solve the optimization problem arising in this setting is different from the standard SR-PDE (gaussian case), in particular this is due to the fact that the likelihood in (6) is not guarantee to be convex anymore, and iterative algorithms of the Newton-type should be used. Specifically, it is not possible to characterize the solution of the problem as done in proposition 1 and the analogous of the non-convex functional in (7) for GAM models is generally optimized through Penalized Iterative Reweighted Least Squares (PIRLS). The generalization of this method with PDE penalization will be the argument of next section.

2.4 Functional PIRLS Algorithm

In order to estimate $\boldsymbol{\beta}$ and the spatial functon f we implement a functional variation of the PIRLS algorithm (FPIRLS). The method has been proposed by M. Wilhelm & L. M. Sangalli in [14].

Consider the more general penalized log-likelihood

$$\mathcal{L}_p(\boldsymbol{\beta}, f) = \mathcal{L}(\boldsymbol{\beta}, f) - \frac{\lambda}{2} m(f, f), \tag{8}$$

where $\mathcal{L}$ is the log-likelihood and $m(\cdot, \cdot) : \mathcal{F} \times \mathcal{F} \to \mathbb{R}$ is any bilinear, symmetric, semi-definite positive form.

It can be shown (see [14] appendix 2 for details) that the problem of maximizing equation (8) with respect to $(\boldsymbol{\beta}, f)$ is equivalent to minimizing the following functional with respect to $(\boldsymbol{\beta}, f)$:

$$\mathcal{J}_\lambda(\boldsymbol{\beta}, f) = ||\mathbf{V}^{-\frac{1}{2}}(\mathbf{z} - \boldsymbol{\mu}(\boldsymbol{\beta}, f))||^2 + \lambda\, m(f, f) \tag{9}$$

where $\mathbf{z} = (z_1, ..., z_n)^t$ is the vector of observed data values; $\boldsymbol{\mu} = (\mu_1, ..., \mu_n)^t$ is the mean vector given by $\boldsymbol{\mu} = g^{-1}(\boldsymbol{\theta})$; $\mathbf{W} \in \mathbb{R}^{n \times q}$ denotes the design matrix, whose i-th row is given by the covariates vector $\mathbf{w}_i$; $\mathbf{f}_n = (f(\mathbf{p}_1), ...f(\mathbf{p}_n))^t$ is the vector of evaluations of the spatial field f at the n spatial locations and $\mathbf{V}$ is an $n \times n$ diagonal matrix with entries $V(\mu_1), ..., V(\mu_n)$ where $V(\cdot)$ is the variance function. Notice that both $\mathbf{V}$ and $\boldsymbol{\mu}$ depend on $(\boldsymbol{\beta}, f)$.

2.4.1 Iterative scheme for FPIRLS

Let $\boldsymbol{\mu}^{(k)}$ be an estimate of $\boldsymbol{\mu}(\boldsymbol{\beta}, f)$ after k iterations. The first order development of $\boldsymbol{\mu}(\boldsymbol{\beta}, f)$ in the neighbourhood of the current value $\boldsymbol{\mu}^{(k)}$ is to be considered in the space $\mathbb{R}^q \times \mathcal{F}$ and yields the following quadratic approximation of $\mathcal{J}_\lambda(\boldsymbol{\beta}, f)$:

$$\tilde{\mathcal{J}}_\lambda(\boldsymbol{\beta}, f) = ||(\mathbf{P^{(k)}})^{\frac{1}{2}}(\tilde{\mathbf{z}}^{(k)} - \mathbf{W}\boldsymbol{\beta} - \mathbf{f}_n)||^2 + \lambda\, m(f, f) \tag{10}$$

where:

- $\mathbf{P^{(k)}} = (\mathbf{G^{(k)}})^{-2}(\mathbf{V^{(k)}})^{-1}$
- $\tilde{\mathbf{z}}^{(k)}$ is the current pseudo-data, $\tilde{\mathbf{z}}^{(k)} = \mathbf{G^{(k)}}(\mathbf{z} - \boldsymbol{\mu}^{(k)}) + \boldsymbol{\theta}^{(k)}$
- $\mathbf{G}^{(k)}$ is the $n \times n$ diagonal matrix with entries $g'(\mu_1^{(k)}), .., g'(\mu_n^{(k)})$
- $\boldsymbol{\theta}^{(k)} = (g(\mu_1^{(k)}), ..., g(\mu_n^{(k)}))$

The functional form in (10) can then be specialized choosing $m(f, f)$ as the misfit of a generic PDE $Lf = u$ as in the SR-PDE model. Moreover, it worth to notice that the parametric part of $\mathcal{J}_\lambda$ resembles the objective function of IRLS algorithm with $P^{(k)}$ as the weight matrix.
The operative form of the algorithm is shown below, where we start from an initial guess $\boldsymbol{\mu}^{(0)}$ equal to $\frac{1}{2}(\mathbf{z}+\frac{1}{2})$ for binary valued distributions, and equal to $\mathbf{z}$ otherwise. The stopping criterion is based on a sufficiently small variation of two successive values of the functional (10)

Algorithm 1 FPIRS

Require: $\boldsymbol{\mu}^{(0)}$ set to $\mathbf{z}$
1: **while** (Not Convergence) **do**
2: Compute $\tilde{\mathbf{z}}^{(k)}$ and $\mathbf{P^{(k)}}$
3: Find $\boldsymbol{\beta}^{(k+1)}$ and $\mathbf{f}^{(k+1)}$ that jointly min $\mathcal{J}_\lambda^{(k)}(\boldsymbol{\beta}, f)$
4: Set $\boldsymbol{\mu}^{(k+1)} = g^{-1}(\mathbf{W}\boldsymbol{\beta}^{(k+1)} + \mathbf{f}_n^{k+1})$
5: **end while**

We still need a method to solve *step 3* and minimize (10) in $(\boldsymbol{\beta}, f)$. However this is a problem similar to (2) except for the presence of the weight matrix P.

2.4.2 Solution for FPIRLS - Laplacian case

In this section we aim to find a solution to *step 3*, i.e. finding the minima in $(\boldsymbol{\beta}, f)$ of the functional in (6). For this purpose we need to specialize the bilinear form $m(\cdot, \cdot)$.
Let's start considering the simpler case in which we have $m(f, f) = \int_\Omega (\Delta f)^2 d\mathbf{p}$. Moreover let's consider for simplicity the case in which we impose homogenous Neumann boundary condition on our PDE penalty, i.e. we search for f in the following functional space

$$\mathcal{F} = H^2_{\mathbf{n}_0} = \{f \in H^2 | (\nabla f)^t \mathbf{n} = 0 \text{ on } \partial\Omega\}$$

In this setting we can consider the following projection matrices

$$\mathbf{H} = \mathbf{W}(\mathbf{W}^t\mathbf{P}\mathbf{W})^{-1}\mathbf{W}^t\mathbf{P} \tag{11}$$

$$\mathbf{Q} = (\mathbf{I} - \mathbf{H})^t\mathbf{P}(\mathbf{I} - \mathbf{H}) \tag{12}$$

and the following holds:

Proposition 2. *Assume the design matrix* $\mathbf{W}$ *has full rank and the diagonal weight matrix* $\mathbf{P}$ *has strictly positive entries. Then exists a unique pair* $(\tilde{\boldsymbol{\beta}}, \tilde{f}) \in \mathbb{R}^q \times H^2_{\mathbf{n}_0}$ *which minimizes (10). Moreover,*

- $\tilde{\boldsymbol{\beta}} = (\mathbf{W}^t\mathbf{P}\mathbf{W})^{-1}\mathbf{W}^t\mathbf{P}(\tilde{\mathbf{z}} - \tilde{\mathbf{f}}_n)$
- $\tilde{f}$ *satisfies:*

$$\mathbf{u}_n^t\mathbf{Q}\tilde{\mathbf{f}}_n + \lambda \int_\Omega (\Delta u)(\Delta \tilde{f}) = \mathbf{u}_n^t\mathbf{Q}\tilde{\mathbf{z}} \quad \forall u \in H^2_{\mathbf{n}_0} \tag{13}$$

where $\mathbf{u}_n$ *is the vector of the evaluation of function* $u \in \mathcal{F}$ *at the* n *location points.*

Proposition 2 highlights the fact that the computational effort in finding the minimizers relies on the solution of the variational problem (13) in $\mathcal{F}$.
However in [13] is shown that we can require less regularity for the function f and search for it in $H^1_{\mathbf{n}_0}(\Omega)$, in particular the problem (13) is equivalent to finding $(\tilde{f}, \tilde{h}) \in H^1_{\mathbf{n}_0}(\Omega) \times H^1_{\mathbf{n}_0}(\Omega)$ s.t.

$$\begin{aligned} \mathbf{u}_n^t\mathbf{Q}\tilde{\mathbf{f}}_n + \lambda \int_\Omega (\nabla u)(\nabla \tilde{h}) &= \mathbf{u}_n^t\mathbf{Q}\tilde{\mathbf{z}} \\ -\int_\Omega (\nabla \tilde{f})^t \nabla v &= \int_\Omega \tilde{h} v. \end{aligned} \tag{14}$$

for any $(u, v) \in H^1_{\mathbf{n}_0}(\Omega) \times H^1_{\mathbf{n}_0}(\Omega)$.
Here we can think about $\tilde{h}$ as the misfit from the Poisson problem used as penalty, since the equality $\Delta\tilde{f} = \tilde{h}$ holds in $H^1_{\mathbf{n}_0}(\Omega)$.

We have now a full characterization of the solution to the minimization problem and we are now ready to solve (14) numerically through FEM. We will show for a metter of compactness, the FEM discretization only for the Laplace problem, however we will briefly discuss solution in some more general frameworks in the next section.

2.4.3 Solution for FPIRLS - general case discussion

Different generalizations can be found specifying some details in the formulation of the minimization problem (2). In particular one can choose to act over one of the following objects:

- the likelihood $l(\cdot)$ of the data distribution model
- the bilinear form $m(\cdot, \cdot)$
- the nature of the domain Ω
- the nature of observations

The data distribution does not affect the form of the algorithm significantly, indeed we do not take into account any particular distribution in order to derive the problem (14). The main difference among distributions relies on the particular form of the link function $g(\cdot)$.
The specialization of the bilinear form $m(\cdot, \cdot)$ address the problem of including prior information, expressed in the form of a PDE, into the model. Previous work on SR-PDE treated mainly three forms of PDE: the Poisson equation $\Delta f = 0$, the general PDE case $Lf = 0$, with L defined as (3) and the more general space varying equation $L_{\mathbf{p}} f = u(p)$ where the parameters of L_p are allowed to vary inside Ω, i.e. $K = K(x, y)$, $b = b(x, y)$ and $c = c(x, y)$. A further variation on the PDE can be achieved using different boundary conditions.
The nature of the domain Ω can also change, in particular it can be a bounded set in $\mathbb{R}^{dim}$ with $dim = 2, 3$ or eventually it can be a non planar surface Γ embedded in $\mathbb{R}^3$. We will refer to this last case in this report with an abuse of notation indicating it as $2.5D$ in contrast to the $2D$ and $3D$ cases encountered in $\mathbb{R}^{dim}$.

Lastly, we can also adapt the model in order to deal with areal observations. Specifically, let $D_1, D_2, ..., D_n$ be disjoint sub-regions of the domain Ω. Over each sub-domain D_i, we observe the realization z_i of a real variable of interest Z_i and a vector of covariate information $\mathbf{w}_i \in \mathbb{R}^q$. We assume $Z_1, ..., Z_n$ are independent, with Z_i having a distribution within the exponential family, with mean μ_i and common scale parameter ϕ. Then we can model μ_i as

$$g(\mu_i) = \mathbf{w}_i^t \boldsymbol{\beta} + \int_{D_i} f \tag{15}$$

where we replace the point-wise evaluation of f in (8) with the integral of the spatial field over the subdomain D_i, hence we redefine the vector $\mathbf{f}_n$ as $\mathbf{f}_n = (\int_{D_1} f, .., \int_{D_n} f)^t$. A second option for areal data consist in treating the mean functional value instead of the sum, i.e. a model

$$g(\mu_i) = \mathbf{w}_i^t \boldsymbol{\beta} + \frac{1}{|D_i|} \int_{D_i} f \tag{16}$$

and in this case the vector $\mathbf{f_n}$ is $(\frac{1}{|D_1|} \int_{D_1} f, ..., \frac{1}{|D_n|} \int_{D_n} f)^t$.
Areal models are of particular interest for many applications. Recent Covid-19 data in Italy for example, are collected by provinces and they represent the sum of the number of positive

cases in a given geographical area. They can be modeled in our setting by a *Poisson* model for counting data as in (5.2.6). Another interesting recent application of areal data can be found in the 5G context. The new generation of LTE connection has indeed a shorter range and new transmitter cells have to be install around cities. In order to minimize costs, mobile network operators can be interested in the distribution of the average number of users (or data-traffic), which can be modeled by a Poisson (or gamma) model[1].

Remark: `fdaPDE` has been recently updated in order to handle even spatio-temporal data. We will not cover the treatment of this last class of models, for which we refer to [1], [2].

2.5 Numerical solution with FEM

The solution to the variational problem (14) cannot be found analytically. We hence made use finite element method (FEM) in order to find an approximate numerical solution.
The functions and integrals in Equation (14) can be approximated using functions in the finite element space $\mathcal{F}_K \subset H^1_{\mathbf{n}_0}(\Omega)$, where K is the dimension of the subspace, which is spanned by the set of basis functions $\psi_1, ..., \psi_K$. Both linear and quadratic FEM are considered in `fdaPDE`. In this setting we can reformulate our problem as

Problem: Find $(\tilde{f}, \tilde{h}) \in \mathcal{F}_K \times \mathcal{F}_K$, for any $(u, v) \in \mathcal{F}_K \times \mathcal{F}_K$ that satisfy (14), where the integrals are computed over the domain triangulation $\Omega_{\mathcal{T}}$.

Let $\mathbf{\Psi}$ be the $n \times K$ matrix of K basis at n locations $\mathbf{p}_1, ..., \mathbf{p}_n$

$$\begin{bmatrix} \boldsymbol{\psi}^t(\mathbf{p}_1) \\ \vdots \\ \boldsymbol{\psi}^t(\mathbf{p}_n) \end{bmatrix}$$

and define the mass and stiffness matrices:

$$\mathbf{R}_0 = \int_{\Omega_{\mathcal{T}}} \boldsymbol{\psi}\boldsymbol{\psi}^t, \quad \mathbf{R}_1 = \int_{\Omega_{\mathcal{T}}} (\nabla\boldsymbol{\psi})^t(\nabla\boldsymbol{\psi})$$

Proposition 3. *The discrete counter part of Equation (14) is given by the linear system*

$$\begin{bmatrix} \mathbf{\Psi}^t\mathbf{Q}\mathbf{\Psi} & -\lambda\mathbf{R}_1 \\ -\lambda\mathbf{R}_1 & -\lambda\mathbf{R}_0 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{f}} \\ \tilde{\mathbf{h}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Psi}^t\mathbf{Q}\tilde{\mathbf{z}} \\ \mathbf{0} \end{bmatrix} \tag{17}$$

which admits a unique pair of solutions $\tilde{\mathbf{f}}, \tilde{\mathbf{h}}$ *that are respectively the coefficients of the basis expansion of* $\tilde{f}$ *and* $\tilde{h}$.

Let $\mathbf{G} = \mathbf{R}_1\mathbf{R}_0^{-1}\mathbf{R}_0$ and $\mathbf{S} = \mathbf{\Psi}(\mathbf{\Psi}^T\mathbf{Q}\mathbf{\Psi} + \lambda\mathbf{G})^{-1}\Psi^T\mathbf{Q}$ .Then, using the functional version

[1]Privacy laws enforce operators to collect data on wide regions instead of pointwise locations, thus the need of areal models.

of the PIRLS algorithm, and thanks to **Proposition.2** and **Proposition.3**, we obtain the following expressions for the maximizers $\hat{\boldsymbol{\beta}}$ and $\hat{\mathbf{f}}$ of the penalized log-likelihood (2):

$$\hat{\boldsymbol{\beta}} = (\mathbf{W}^t\mathbf{P}\mathbf{W})^{-1}\mathbf{W}^t\mathbf{P}(\mathbf{I}-\mathbf{S})\tilde{\mathbf{z}} \tag{18}$$

$$\hat{\mathbf{f}} = (\boldsymbol{\Psi}^t\mathbf{Q}\boldsymbol{\Psi} + \lambda\mathbf{G})^{-1}\boldsymbol{\Psi}^t\mathbf{Q}\tilde{\mathbf{z}} \tag{19}$$

and the vector of evaluations of $\hat{\mathbf{f}}$ at the n data locations is given by

$$\hat{\mathbf{f}}_n = \boldsymbol{\Psi}\hat{\mathbf{f}} = \mathbf{S}\tilde{\mathbf{z}}$$

where the vector of pseudo-data $\tilde{\mathbf{z}}$, and the matrices $\mathbf{P}$ and $\mathbf{Q}$ are those obtained at the convergence of the PIRLS algorithm.

2.5.1 Woodbury decomposition

Solving the linear system (17) can be expensive since its dimension scales quadratically with the number of nodes K used in the discretization of Ω. However a suitable decomposition can be found for the system matrix such that the linear problem reduces to the inversion of a sparse matrix and a smaller matrix of dimension $q \times q$. In order to achieve the suitable decomposition consider the following equivalent form for our matrix $\mathbf{Q}$

$$\mathbf{Q} = (\mathbf{I}-\mathbf{H})^t\mathbf{P}(\mathbf{I}-\mathbf{H}) = (\mathbf{I}-\mathbf{H})\mathbf{P}$$

which exploit the fact that the matrix $\mathbf{P}$ is diagonal and $\mathbf{I}-\mathbf{H}$ is a projection matrix, thus symmetric and idempotent. With this new form of $\mathbf{Q}$ at hand we can decompose the system matrix as follow

$$\begin{bmatrix} \boldsymbol{\Psi}^t\mathbf{Q}\boldsymbol{\Psi} & -\lambda\mathbf{R}_1 \\ -\lambda\mathbf{R}_1 & -\lambda\mathbf{R}_0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\Psi}^t\mathbf{P}\boldsymbol{\Psi} & -\lambda\mathbf{R}_1 \\ -\lambda\mathbf{R}_1 & -\lambda\mathbf{R}_0 \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Psi}^t(-\mathbf{P}\mathbf{H})\boldsymbol{\Psi} & 0 \\ 0 & 0 \end{bmatrix} = \mathbf{E} + \mathbf{B} \tag{20}$$

and $\mathbf{B}$ can be written as $\mathbf{B} = \mathbf{U}\mathbf{C}\mathbf{V}$ with

$$\mathbf{U} = \begin{bmatrix} \boldsymbol{\Psi}^t\mathbf{P}\mathbf{W} \\ 0 \end{bmatrix} \quad \mathbf{C} = -(\mathbf{W}^t\mathbf{P}\mathbf{W})^{-1} \quad \mathbf{V} = \begin{bmatrix} \mathbf{W}^t\mathbf{P}\boldsymbol{\Psi} & 0 \end{bmatrix} \tag{21}$$

where we have used the fact $\mathbf{H} = \mathbf{W}(\mathbf{W}^t\mathbf{P}\mathbf{W})^{-1}\mathbf{W}^t\mathbf{P}$.

With the above decomposition we can apply the Woodbury identity which states

Proposition 4 (Woodbury identity)**.**

$$\mathbf{M}^{-1} = (\mathbf{E}+\mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{E}^{-1} - \mathbf{E}^{-1}\mathbf{U}(\mathbf{C}^{-1} + \mathbf{V}\mathbf{E}^{-1}\mathbf{U})^{-1}\mathbf{V}\mathbf{E}^{-1}$$

where $\mathbf{M} \in \mathbb{R}^{2K\times 2K}$, $\mathbf{E} \in \mathbb{R}^{2K\times 2K}$, $\mathbf{U} \in \mathbb{R}^{2K\times q}$, $\mathbf{C} \in \mathbb{R}^{q\times q}$ *and* $\mathbf{V} \in \mathbb{R}^{q\times 2K}$

Thus we reduce the linear problem to the computation of the inverse of $\mathbf{E}$ which is sparse in our case, and the inverse of $\mathbf{C}^{-1} + \mathbf{V}\mathbf{E}^{-1}\mathbf{U}$ which has dimension $q \times q$. Notice that we can also have $q = 0$, i.e. no covariates, and in this case we just have to solve a sparse system.

2.5.2 Areal data

In order to treat areal data we have to redefine $\mathbf{\Psi}$ as

$$\big[\mathbf{\Psi}\big]_{ij} = \int_{D_1} \psi_j$$

Then, the equivalent discrete system of (17) for areal data is given by

$$\begin{bmatrix} \mathbf{\Psi}^t\mathbf{A}\mathbf{Q}\mathbf{\Psi} & -\lambda\mathbf{R}_1 \\ -\lambda\mathbf{R}_1 & -\lambda\mathbf{R}_0 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{f}} \\ \tilde{\mathbf{h}} \end{bmatrix} = \begin{bmatrix} \mathbf{\Psi}^t\mathbf{A}\mathbf{Q}\tilde{\mathbf{z}} \\ \mathbf{0} \end{bmatrix} \tag{22}$$

where the form of the matrix $\mathbf{A}$ depends on the areal model considered. If we use a model dealing with average values over the domains, i.e. $\mathbf{f}_{\mathbf{n}i} = \frac{1}{|D_i|}\int_{D_i} f\,dp$ then we define $\mathbf{A} = diag(\frac{1}{|D_1|}, .., \frac{1}{|D_n|})$, while if we consider a pure integral model with $\mathbf{f}_{\mathbf{n}i} = \int_{D_i} f\,dp$, then the matrix $\mathbf{A}$ is the simple identity $\mathbf{I}$.

2.6 Selection of the smoothing parameter

The solution of the system (17) has to be computed at each iteration of the FPIRLS algorithm until convergence is reached and allows us to find the numerical solution of problem (10) for a fixed value of the smoothing parameter *lambda*. However the choice of λ is not straightforward. In general one should chose the parameter which maximizes goodness of fit w.r.t. a given metric, and the choice of different metrics can lead to different criteria. We chose as a measure of the goodness of fit the *deviance* of the distribution considered, i.e.

$$D(\boldsymbol{\theta}) = 2(l_{max} - l(\boldsymbol{\theta}))\phi \tag{23}$$

where ϕ is the scale parameter of the distribution, l is the log-likelihood of the model and l_{max} is the maximum likelihood achieved given the measured data and correspond to the likelihood of the overfitted model, where the number of parameters coincides with the number of data and the prediction for $\mathbb{E}[z_i]$ is simply z_i. Notice that D represents a distance between our model and the *best* achievable one, and also we do not actually need to compute ϕ since D/ϕ can be used as well.
With this choice of goodness of fit we may choose the smoothing parameter as the one minimizing the generalized cross validation (GCV) criterion [8]:

$$GCV(\lambda) = \frac{n||\mathbf{y} - \boldsymbol{\mu}(\hat{\boldsymbol{\beta}}, \hat{\mathbf{f}})(\lambda)||_d^2}{[n - \gamma\, tr\mathbf{D}(\lambda)]^2} \tag{24}$$

where $\boldsymbol{\mu}(\hat{\boldsymbol{\beta}}, \hat{\mathbf{f}})(\lambda)$ is the fitted mean at the convergence of FPRILS, γ is a constant real value usually equal to 1 (values grater than 1 are occasionally used to avoid overfitting) and the norm $||\cdot||_d$ is the one induced by the *deviance* of the distribution considered, and become the standard $\mathbb{R}^n$ norm in the gaussian case.
$\mathbf{D}(\lambda) = \mathbf{H} + \mathbf{QS}$ is the influence matrix of the generalized additive model and it is defined as

the matrix mapping observations into parameter estimates $\hat{\boldsymbol{\theta}} = \mathbf{Dz}$. The trace of the influence matrix $tr\mathbf{D} = tr(\mathbf{H} + \mathbf{QS}) = q + tr\mathbf{S}$ can be used as a measure of the equivalent degrees of freedom (dof) of the model [6]. Computation of $tr\mathbf{S}$ is the most expensive operation in the GCV criterion, for this reason `fdaPDE` provides two different options for the users, an `exact` computation and a `stochastic` approximation which requires less numerical effort (see [5] for details).

Remark: the choice of λ by minimizing (24) is not the only possible (one can see [16] for an interesting discussion on smoothness selection in GAM context), however this choice represent the direct generalization of the smoothing selection criterion currently adopted for gaussian models and already implemented in `fdaPDE`.

2.7 Scale parameter estimate

Any distribution of the exponential family is described by two parameters, the mean μ and the scale parameter ϕ. The estimation of the mean does not require the estimation of the scale parameter but only of the canonical parameter. To estimate the scale parameter, we must estimate the mean for all the observations. A classical estimator of the scale parameter is

$$\hat{\phi} = \frac{||\mathbf{V}^{-1/2}(\mathbf{y} - \hat{\boldsymbol{\mu}})||^2}{n - trD} \tag{25}$$

where $\hat{\boldsymbol{\mu}}$ is the estimated mean at the convergence, $\mathbf{V}$ is the $n \times n$ diagonal matrix with entries $V(\hat{\mu}_1), ..., V(\hat{\mu}_n)$ and $\mathbf{D}$ is the influence matrix.

3 Code Structure

A basic experimental version of code for a GSR-PDE model was already developed in R by M. Wilhelm in 2016. The available function handled both pointwise and areal observations for $2D$ data with a Laplace penalization and it included a wide range of distributions within the exponential family. Our work consists in the implementation of the GSR-PDE model in C++ within the `fdaPDE` library and its extension to more general PDE penalizations and to data distributed over $3D$ and $2.5D$ domains.
The main goal of the work is dual. On one side a C++ implementation speeds up a lot the execution process as we will see in the simulation section, which is particularly interesting for large and dense meshes. On the other side it allows the inclusion of GAM models within the `fdaPDE` package, providing a way to reach a wider audience of users whenever its new version will be uploaded on CRAN.

In order to include GSR-PDE models within the library we have to face the following main issues:

- Introduction of the iterative scheme for FPIRLS
- Solution of problem (10) exploiting existing methods
- Scale parameter computation
- GCV computation
- R/C++ interface

We are going to analyze them in details through the following sections.

Remark: some type definitions has been added in the `fdaPDE` package, please refer to appendix A for any doubts.

3.1 R/C++ interface

The R interface for the final users is shared with the classical method. An additional string `family` with default value set to `"gaussian"` is present, alongside with additional parameters for the iterative scheme (`max_iter`, `treshold`). When the family string is chosen differently from gaussian FPIRLS method will be called, while the standard method is called otherwise. We also added a new feature for areal data with the flag `areal.data.avg` which perform areal smoothing considering the average of the function in the mesh triangle $\mathbf{f}_{ni} = \frac{1}{D_i}\int_{D_i} f$ when `true` or the simple sum $\mathbf{f}_{ni} = \int_{D_i} f$ when `false`.

The R function then preprocess the data before calling the C++ methods with the `.Call` interface. In particular it checks user input validity, enforce the right type for each variable and manages the discrepancy between R and C++ indexes (R enumeration starts from 1 instead of 0).
R variables passed through `.Call` become the inputs of the corresponding C++ function.

These objects are then available in the C++ environment as `SEXP`, i.e. "Simple EXPressions", which are pointers to a *variant type* that can handle all the usual types of R objects. On the other side the C++ functions become visible to the `.Call` interface when embedded in the `extern C` macro and listed in the `fdaPDE_init.c` file [12].

The `C++` interface function (`gam_Laplace`, `gam_PDE` or `gam_PDE_space_varying` depending on PDE penalization used) is then responsible to initialize a `RegresionDataGAM` object, to read the variables containing the template parameters for the PDE penalization and the mesh structure, and to use them to call the `GAM_skeleton` function with the right specialization, which is where the factory create the `FPIRLS` object and the core method is called. Finally, results are retrieved from `FPIRLS` and returned to R. In doing so one should create a pointer to an R list using the `PROTECT` macro which prevent the pointed object from being cancelled by the R garbage collector. Indeed R memory is not freed by the programmer, but a routine is called from time to time and marks unused memory as re-usable. The R list is then filled with the requested data and returned to R.
The hierarchy of the function called is shown below

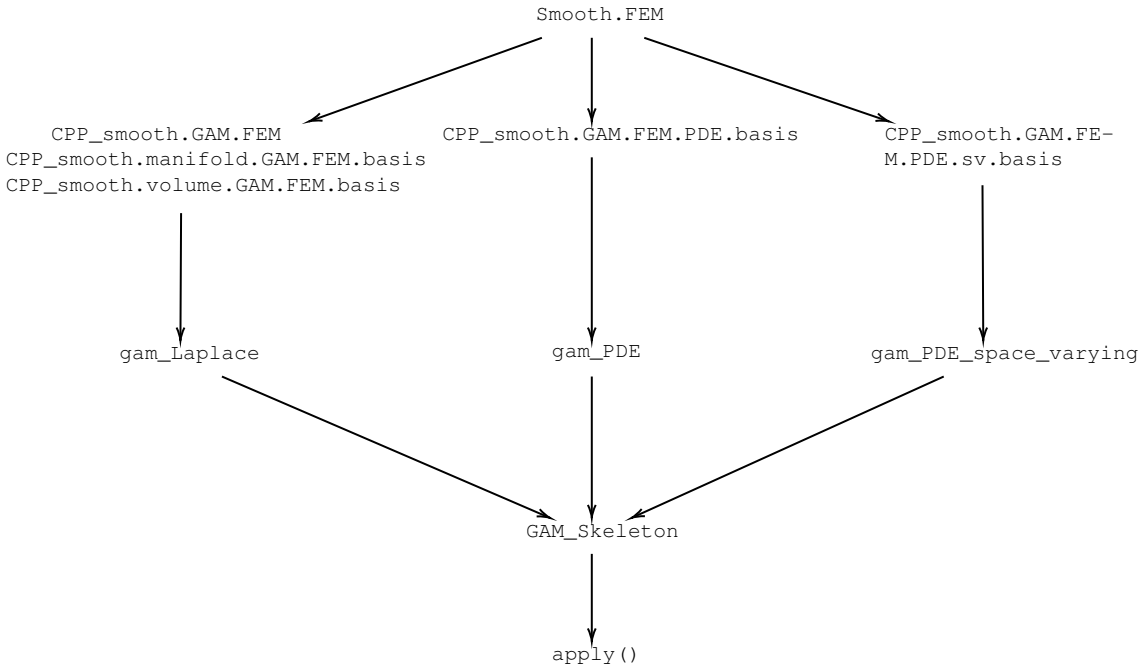


Figure 2: FPRILS algorithm structure in `R/C++`.
Notice that we have different C++ interfaces for different penalization: $\Delta f = 0$ for `Laplace`, $Lf = 0$ for `PDE` and $Lf - u = 0$ for `PDE_space_varying`, where the coefficient of the equation ($\mathbf{K}$,$\mathbf{b}$,c) are allowed to be spatial functions in this last case.

3.2 Core method

The core class is `FPIRLS_Base`. It is an abstract class and it stores a reference to the mesh object, which is defined by three template parameters (the order of the FE basis functions,

the dimension of the data manifold `mydim` and the dimension of the embedding space `ndim`), a GAM data structure containing main regression informations and an object used to perform the standard gaussian regression as in (10). Further variables store the main elements used in FPIRLS algorithm (1), in particular `pseudoObservations_` correspond to our $\tilde{\mathbf{z}}$ and `WeightsMatrix_` is our $\mathbf{P}$.
Main variables are stored for each value of λ thus we collect them into `std::vectors`. Particular exception is made for results drawn from `MixedFERegression`, since in this case the containers are `MatrixXv` as used to collect results in spatio-temporal framework, hence with both λ_s (spatial) and λ_t (temporal). However we prefer to not extend this container on other variables since we believe an `std::vector` would be more appropriate and safe than an Eigen structure for a collection of elements. The class definition is shown in Figure 3.

As we see, we have a bunch of pure virtual methods. These correspond to the functions typical of each distribution and will be overridden by children and specialized for each distribution in the exponential family. These allow us to implement a general abstract method concerning the solution of our problem via FPIRLS, which shares the same structure for any distribution except those virtual functions.

The hierarchy structure of the class is shown in Fiugre 4. Each terminal class specifies the virtual methods of `FPRILS_Base` with the right functions of the chosen distribution. Adding new distributional forms within the same settings is made pretty simple, since just a new child class of `FPRILS` has to be added. The first inheritance, `FPIRLS`, is used to generalize the PDE penalization and we will discuss it later; for the purpose of this section we can think of `FPIRLS` and `FPIRLS_Base` as the same class.

The main method in `FPIRLS_Base` is `apply()`. It performs the iterative algorithm for each value of λ and handles the GCV computation. In other words, for each λ and until the algorithm converges it applies the following steps:

- `compute_G`:
  compute the diagonal matrix $\mathbf{G}$ as $\mathbf{G}_{ii} = diag(g'(\hat{\boldsymbol{\mu}}_i))$;
- `compute_Weights`
  compute the diagonal matrix $\mathbf{P}$ as $\mathbf{P}_{ii} = diag(\frac{1}{(g'(\hat{\boldsymbol{\mu}}_i))^2} \cdot \frac{1}{var(\hat{\boldsymbol{\mu}}_i)})$;
- `compute_pseudoObs`
  compute the vector $\tilde{\mathbf{z}}$ as $\tilde{\mathbf{z}}_i = \mathbf{G}_{ii}(\mathbf{z}_i - \hat{\boldsymbol{\mu}}_i) + g(\hat{\boldsymbol{\mu}}_i)$;
- `updatePseudodata`
  this is a method of `RegressionDataGAM` which sets up the new vector of (pseudo)observations $\tilde{\mathbf{z}}$ and the weights $\mathbf{P}$ for the regression step. Notice that we fully exploit the *by-reference* storage used for `RegressionData` object in `MixedFERegression` class, indeed doing this way the regression object has new data available and it is ready for the next step.
- `update_solution`
  perform the regression step of FPIRLS using updated data and the `MixedFERegression` object;

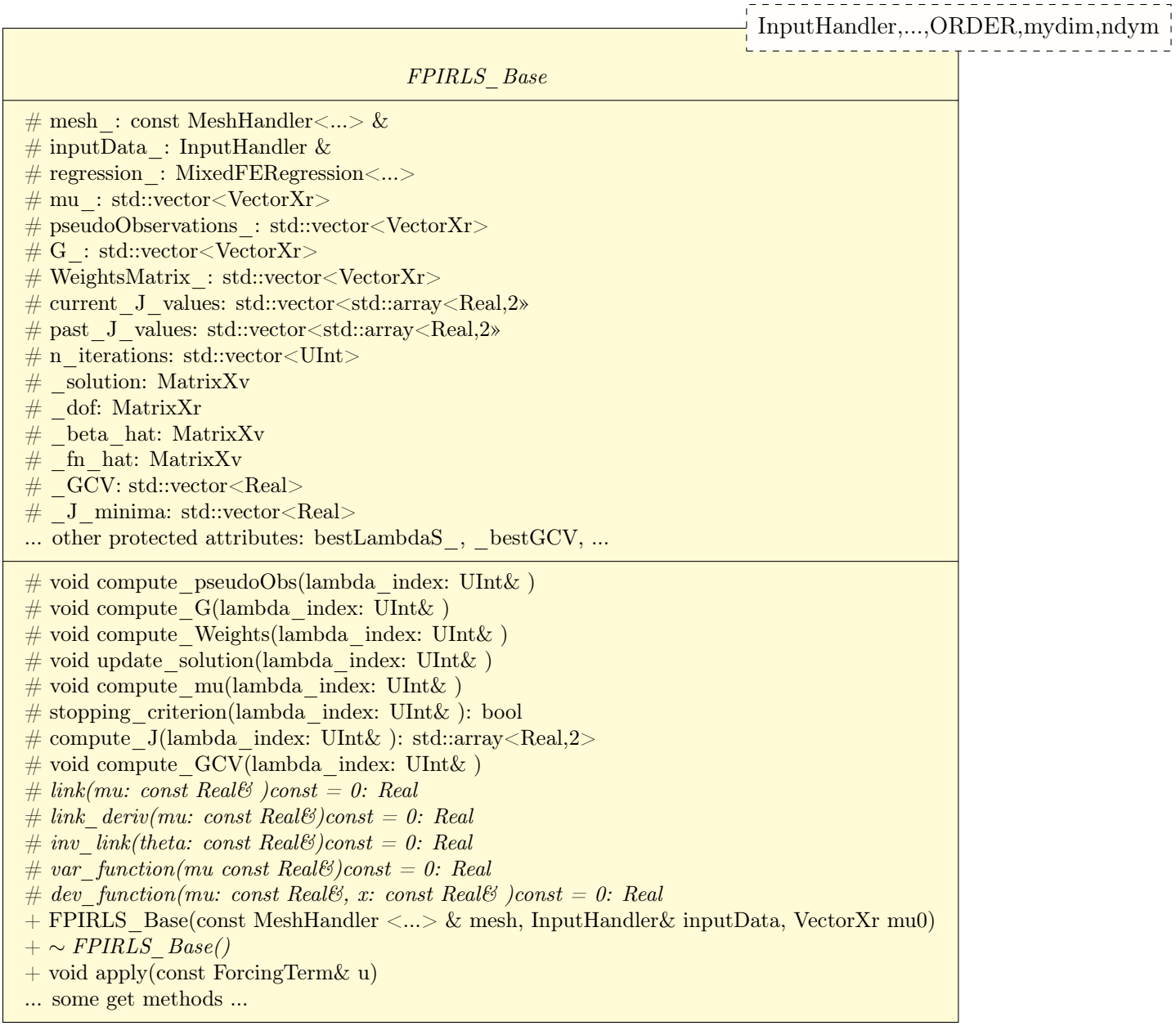


Figure 3: UML diagram for the `FPIRLS_Base` class

- `compute_mu`
  compute mean parameter estimate $\hat{\boldsymbol{\mu}}$ according to $\hat{\boldsymbol{\mu}}_i = g^{-1}(\mathbf{w}_i^t\boldsymbol{\beta} + \mathbf{f}_{\mathbf{n}i})$;

- update `J_value`
  compute the functional in (9) dividing its calculation between parametric and non-parametric part. Indeed the parametric part of (9) is then reused for the estimate of the scale parameter ϕ (if requested).
  The non-parametric part is computed exploiting mass matrix $\mathbf{R}_0$ retrieved from `MixedFERegression` object, and the coefficients of the finite element representation of h (eq.14), which corresponds to the laplacian Δf in the basic case and to the general misfit $Lf - u$ when

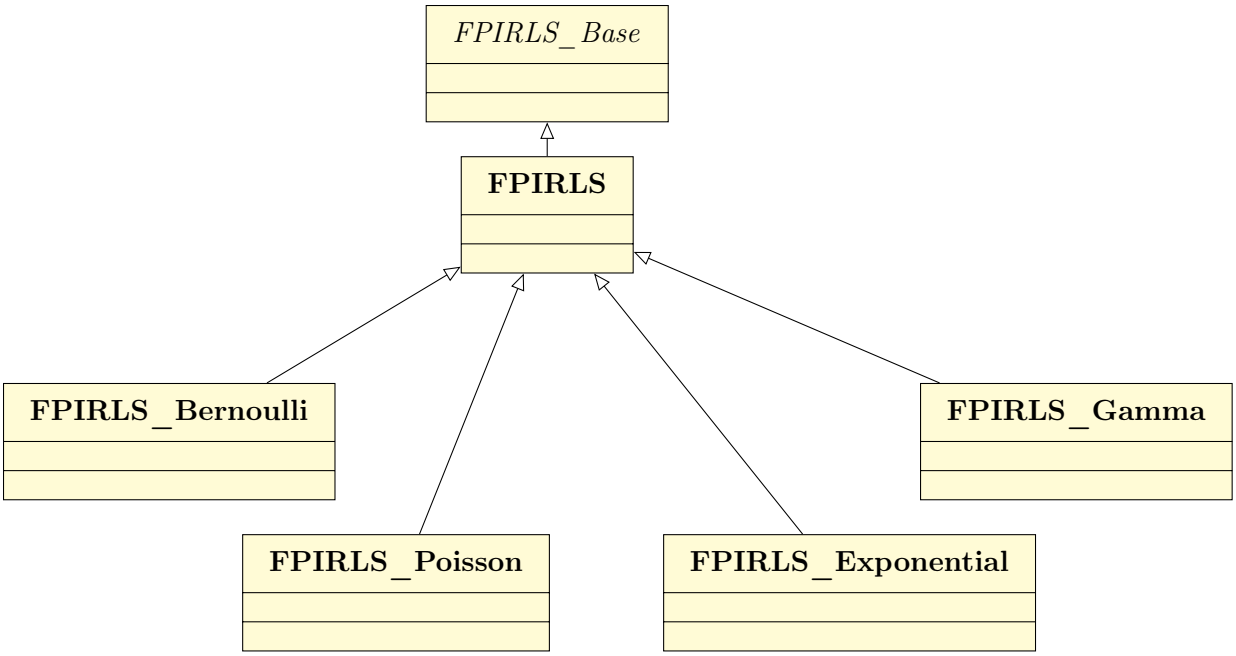


Figure 4: FPIRLS hierarchical structure

a more general PDE is used. In other terms we have:

$$\int_\Omega (Lf - u)^2 \mathbf{dp} = \mathbf{h}^t \mathbf{R}_0 \mathbf{h}$$

The iterative algorithm stops according to the boolean returned by `stopping_criterion`. Here the value reached by the functional $\mathcal{J}_\lambda$ at the current step k is confronted to the value at previous step $k-1$ and if a minor change has occurred ($<$ `threshold`) the algorithm stops. A `max_steps` is also used as upper bound for the number of iterations.
At convergence the GCV criterion is computed (we will discuss it later) and the operation repeated for the next λ.

3.2.1 Interaction with `MixedFERegression`

The two crucial steps in terms of computational effort are the solution of the linear regression problem (10) and the DoF computation (used in GCV criterion) and they are both performed by the already existent `MixedFERegression` class, which is also the main class for many library functionalities. The class however was not designed for multiple usages as it is requested in iterative methods, and thus several issues arise. The class indeed performs the full regression procedure, from matrices construction to the loop over λ (for GCV-minimum grid search) and solution of system (17) within the same `apply()` method, since it was designed for a single regression call. In our case instead we would like to avoid the construction of unchanged matrices when the regression task is performed multiple times and loop over λ on the whole `FPIRLS` algorithm in order to compute GCV only at convergence.

The latter issue will be addressed by the `RegrassionDataGAM` class and we will discuss in next section, while about the former we thought the best way to avoid superfluous computations yet preserving the regression class structure was to store a full `MixedFERegression` object responsible for all regression tasks among `FPIRLS` attributes and to add in the regression class three additional flags telling if the unchanged matrices have already been computed or not. In particular we modify the first lines in the `apply` method of `MixedFERegressionBase` inserting conditional statements as follow

```
if(regressionData_.getNumberOfRegions()>0 && !isAComputed){
  setA();
  isAComputed = true;
}

if(!isPsiComputed){
    setPsi();
    isPsiComputed = true;
}

typedef EOExpr<Mass> ETMass; Mass EMass; ETMass mass(EMass);
if(!isR1Computed){
  Assembler::operKernel(oper, mesh_, fe, R1_);
  isR1Computed = true;
}
if(!isR0Computed){
  Assembler::operKernel(mass, mesh_, fe, R0_);
  isR0Computed = true;
}

```

This way the computation of matrices $\mathbf{A}$, $\mathbf{\Psi}$, $\mathbf{R}_0$ and $\mathbf{R}_1$ is performed only the first time a regression step is done (i.e. first value of λ, first FPIRLS step), and saved for any future use of the object.

Another important change we need to make in `MixedFERegressionBase` concerns the introduction of the diagonal matrix of weights $\mathbf{P}$. It is stored as a `VectorXr` in the `RegressionData` class and used in the following methods of `MixedFERegression`:

- `LeftMultiplyByQ`
  This method return the result of the operation $\mathbf{Qu}$, where $\mathbf{u}$ is a matrix (or vector) passed as argument to the method.
  Here the structure of $\mathbf{Q}$ and $\mathbf{H}$ is different for non-gaussian distributions, thus we had to modify computations including $\mathbf{P}$ where needed.
  Particular care has been taken for the LU factorization of $\mathbf{W}^t\mathbf{PW}$. The old code used a flag (`isWTWfactorized_`) in order to avoid repeating the factorization, however we need to reset the flag at each new iteration of FPRILS since $\mathbf{P}$ is changing. We achieved this adding a public method `recomputeWTW` which set the flag to false and it is called before a new regression step is performed.

- `apply`
  In addition to the already mentioned flags added in the method, we modified the

composition of the N-W block of the FE system matrix. When distribution is not gaussian indeed $\mathbf{\Psi}^t\mathbf{\Psi}$ is replaced by $\mathbf{\Psi}^t\mathbf{P}\mathbf{\Psi}$.

- `getRightHandData`
  Here the right hand side of system (17) is computed. The main concern for non-gaussian data is the case where no covariates are present. Indeed in this situation the classical $\mathbf{Q}$ collapse to the identity and in the old code the product $\mathbf{Qz}$ is not performed at all. However in FPRILS scenario the matrix $\mathbf{Q}$ become $\mathbf{P}$ when no covariates are present, so the matrix-vector multiplication has to take place.

- `system_factorize`
  In this method the Woodbury decomposition of the system (17) is computed as in (21). We take care to insert the weights matrix where and if needed.

Finally, a last consideration on this class usage has to be done. In our case the regression method `apply()` is called multiple times in `FPIRLS` to solve the weighted problem (10), where pseudo-observations $\tilde{\mathbf{z}}^{(k)}$ and weight matrix $\mathbf{P}^{(k)}$ are dependant on the iteration step k, thus recomputed each time. Their corresponding objects are present among `RegressionData` attributes (as `observations_` and `WeightsMatrix_` respectively) and a `const` reference of `RegressionData` is saved in `MixedFERegression`. We think this last storage method was adopted for memory-saving purposes in the first place, however it allows the `FPIRLS` class to modify $\tilde{\mathbf{z}}^{(k)}$ and $\mathbf{P}^{(k)}$ in its `inputData_` object making directly available their changes also inside `regression_`. Despite its efficiency, this method makes the changes in the regression object not completely explicit and transparent. We still believe it is the most appropriate way to precede given the `MixedFERegression` structure, but we want to stress this point especially for future developers.

3.2.2 Data structure: `RegressionDataGAM`

The main class for data handling in `fdaPDE` is `RegressionData`. It stores almost any variable used for regression except the ones relying on the mesh structure. This class has been first designed for a simpler laplacian penalization and then extended in order to deal with more general PDEs; in particular two additional classes inherit from `RegressionData` and are suited for a general elliptic penalization and a space varying version of it. The class hierarchy is reported in the UML diagram in Figure 5.
We of course added the new `WeightsMatrix_` to correct the regression step. However `RegressionData` does not fit the iterative nature of the algorithm and thus a new data class was needed. In particular the new class should present the following features

- store iterative variables provided by the user (`treshold`, `max_iter`) alongside with a copy of initial data $\mathbf{z}$ and allow the update of $\tilde{\mathbf{z}}$ and $\mathbf{P}$ at each FPIRLS step;

- exhibit a general behaviour when used in `MixedFERegression` in order to easily extend the method for elliptic and space varying PDE;

- delay the computation of GCV until convergence is reached as mentioned in previous section.

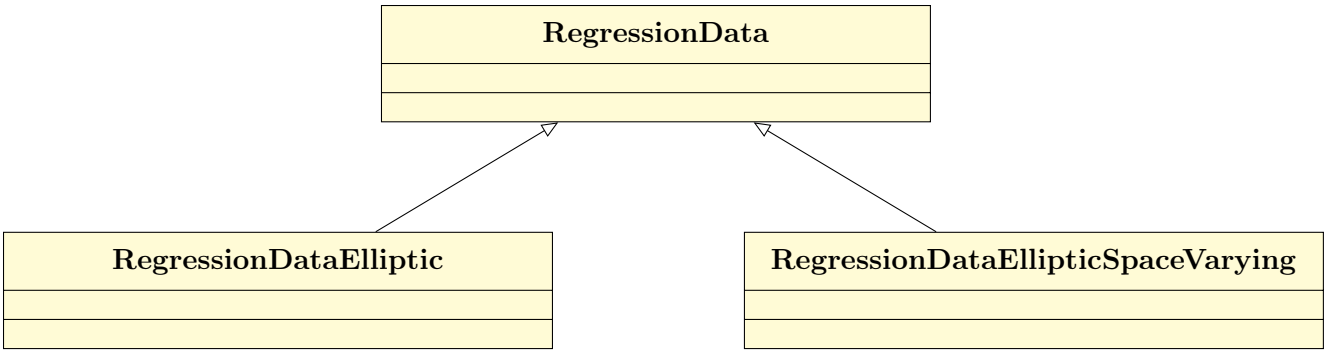


Figure 5: `RegressionData` hierarchical structure

The new class is called `RegressionDataGAM` and take the above issues in the following way. The first point is achieved inheriting from `RegressionData` and thus being able to modify `Observations_` and `WeightsMatrix_` with an ad-hoc method `updatePseudodata`, while the additional copy of the initial observations **z** is stored by the constructor during instantiation of `RegressionDataGAM` objects.
The inheritance from `RegressionData` only would have limited the application to the laplacian penalization case, hence we designed `RegressionDataGAM` as a template class where the template parameter is used to select the right parent class. This avoids code repetition while allowing the required generalization. Typedefs have then been created to clarify the code.

```
template<typename RegressionHandler>
class  RegressionDataGAM : public RegressionHandler
{ ... }

typedef RegressionDataGAM<RegressionData> GAMDataLaplace;
typedef RegressionDataGAM<RegressionDataElliptic> GAMDataElliptic;
typedef RegressionDataGAM<RegressionDataEllipticSpaceVarying>
   GAMDataEllipticSpaceVarying;
```

The generalization required has then been achieved adding the right template specialization for the three new classes to `MixedFERegression`. For instance for the laplacian penalization case we have:

```
template<...>
class MixedFERegression<GAMDataLaplace,...> : public MixedFERegressionBase
   <RegressionData, ...>
{
public:
  MixedFERegression(const MeshHandler<..>& mesh, const RegressionData&
   regressionData):MixedFERegressionBase<RegressionData,...> (mesh,
   regressionData){};
...
}
```

so that when `GAMDataLaplace` is passed to `FPRILS_Base` as template parameter (`InputHandler`), the instantiation of the `MixedFERegression` object is the desired one (with `RegressionData`),

even maintaining a general template class for `InputData_`; and this basically avoid the usage of an additional template parameter for linear regression specification.
Finally, the delayed computation of GCV criterion is achieved acting on the `GCV` flag of the underlying `RegressionData` part of `RegressionDataGAM`. It is a flag used by `MixedFERegression` to know if GCV computaton is required or not, and it is basically always set to `false` by the GAM constructor, which instead store in `GCV_GAM` the true information. The same happens for the vector of λ, indeed `global_lambda_` stores the full array in the GAM structure, while a single value is properly chosen for the regression step within the FPRILS cycle. In this way we are enforcing the `while` loop (FPIRLS iteration) to be nested into the `for` one (multiple λ for GCV) and not the vice-versa. A simple argument for this enforcement relies on the fact that for different values of λ the FPRILS convergence is reached in a different number of steps. It moreover eases a possible task parallelization on each λ.

Finally we want to discuss here an important issue we faced in the design of the data structure and its interaction with `FPIRLS` class.
In particular, in `FPIRLS` we want to have a full `RegressionData` object to pair with `regression_` in order to perform the linear regression step and to be able to modify a couple of its attributes ($\tilde{\mathbf{z}}$, $\mathbf{P}$) at each step of the algorithm as well as store the initial observation vector $\mathbf{z}$ and the array of λ. Moreover, since we expect the dataset to be potentially large, we desire to achieve all of this avoiding useless copies.
We realized the desired behaviour storing in `FPIRLS` a (non-`const`) reference to the input `RegressionDataGAM` object structured as explained above. Nevertheless, we are aware of two main drawbacks of this implementation. First we have an *input* object, that can be ideally though as a constant static entity, which actually present a public method (`updatePseudodata`) which allows runtime modifications. Second, since we do not store a full copy of the data structure in `FPIRLS`, we allow the implicit modification of an object external to the `FPIRLS` class. This last practice in particular can potentially cause issues whenever multiple objects share references to the same `RegressionDataGAM` object. Our choice has been mainly driven by efficiency, which is a key feature of the `fdaPDE` library, but we want to call the attention on potential issues especially for future usages of the `RegressionDataGAM` class.

3.2.3 FPIRLS and the PDE generalization

In order to further generalize the applicability of the method to different PDE penalizations and avoid any code replication, we introduced an intermediate hierarchy of inheritance (`FPIRLS_Base` into `FPIRLS`) performing a template specialization. In particular, we specialize the call to the `apply` method of `FPIRLS_Base` method based on the presence of the forcing term for a space varying PDE.

```
1 // Laplace or Elliptic
2 template <typename InputHandler, typename Integrator, UInt ORDER, UInt
     mydim, UInt ndim>
3 void FPIRLS<InputHandler,..>::apply(){
4
5   FPIRLS_Base<InputHandler,..>::apply(ForcingTerm(std::vector<Real>(1)));
```

```
6
7 }
8
9 // SpaceVarying
10 template <typename Integrator, UInt ORDER, UInt mydim, UInt ndim>
11 void FPIRLS<GAMDataEllipticSpaceVarying,..>::apply(){
12
13   this->isSpaceVarying = true;
14   FPIRLS_Base<GAMDataEllipticSpaceVarying,..>::apply(this->inputData_.getU
       ());
15
16 }
```

This, together with the template specialization on `MixedFERegression` discussed in the previous section and small changes involving the forcing term present in `FPIRLS::apply()` enable us to perform the regression task with any PDE penalization.

3.2.4 Factory object

Due to the presence of multiple classes children of `FPIRLS` and the different initialization of $\boldsymbol{\mu}_0$ the generation of the GAM regression object has been assigned to a factory object.
The factory class `FPIRLSfactory` present a single static method which return a `unique_ptr` to a new object of the desired class. This result is achieved by a call to `make_unique` which has been introduced in the library by Luca Negri for `FEFPCAfactory` [11], since actually the `fdaPDE` package relies on the C++11 standard.
The static method of the factory object takes also care of building the default initial guess $\boldsymbol{\mu_0}$ and manages to set up the flag for scale parameter (ϕ) estimation when ϕ is not provided by the user. The full method is shown below

```
1 static std::unique_ptr<FPIRLS<InputHandler, Integrator, ORDER,  mydim,
     ndim>> createFPIRLSsolver(const std::string &family, const MeshHandler<
     ORDER,mydim,ndim>& mesh, InputHandler& inputData, VectorXr& mu0, Real
     scale_parameter)
2   {
3     //initial checks: m0 must be initialized correctly for the different
     distributions
4
5     if(mu0.size() == 0){
6       VectorXr y = inputData.getObservations();
7       if( family == "binomial" ){ //binary outcomes
8         mu0 = VectorXr::Zero(y.size());
9         for(UInt i = 0; i < y.size(); i++){
10            mu0[i] = 0.5 * (y[i] + 0.5);
11        }//end for-i
12      }else{ // not-binary outcome
13        mu0 = y;
14      }
15    }//end if
16
17    if(family=="poisson"){
```

```
          for(UInt i = 0; i < mu0.size(); i++){
            if(mu0[i]<=0) mu0[i] = 1;
      }
        }

    // Manage scale_parameter
    bool scale_parameter_flag = false;
    if( (family=="gamma") && scale_parameter<0){
      scale_parameter_flag = true;
    }

    if(family=="binomial"){
        return make_unique<FPIRLS_Bernoulli<InputHandler, Integrator,
   ORDER, mydim, ndim>>(mesh, inputData, mu0);
    }else if(family=="poisson"){
        return make_unique<FPIRLS_Poisson<InputHandler, Integrator, ORDER,
     mydim, ndim>>(mesh, inputData, mu0);
    }else if(family=="exponential"){
        return make_unique<FPIRLS_Exponential<InputHandler, Integrator,
   ORDER, mydim, ndim>>(mesh,inputData,mu0);
    }else if(family=="gamma"){
        return make_unique<FPIRLS_Gamma<InputHandler, Integrator, ORDER,
   mydim, ndim>>(mesh,inputData,mu0, scale_parameter, scale_parameter_flag
   );
    }

    return std::unique_ptr<FPIRLS<InputHandler, Integrator, ORDER, mydim,
     ndim>>(nullptr);
  }
```

With the factory method handling the creation of `FPIRLS` objects, the addition of new distributions for the `FPIRLS` algorithm is made pretty simple. Indeed it suffices to create a new class in `FPIRLS.h` inheriting from `FPIRLS`, and write down the virtual methods containing the typical distribution functions (link function, its derivative and so on). Then the new class has to be added in the factory with its respective `family` string and only if the distribution present binary outcomes, add the `family` string also to the $\boldsymbol{\mu}_0$ conditional statement.
On the other hand, the interface for an `FPRIRLS` object instantiation, which is present in `fdaPDE.cpp`, remains always the same:

```
std::unique_ptr<FPIRLS<...>> fpirls = FPIRLSfactory<...>::
   createFPIRLSsolver(family, mesh, GAMData, mu0, scale_parameter);
```

3.3 Scale parameter and variance estimates

Estimation of scale parameter comes with equation (25). However it happens for many distributions in the exponential family to have a constant scale parameter equal to 1, hence we would like to estimate it only for certain distributions. Moreover, the user can specify its own scale parameter when it is known and in this case its estimation is avoided. In the C++ code the factory object produces a flag (`scale_parameter_flag_`) storing the information on

whether the scale parameter has to be computed or not. Notice that the flag is considered only for distributions which allow a scale parameter different from 1, otherwise it is not even used in the constructor. A devoted method `compute_variance_est()` is present in `FPIRLS_Base` for the estimation of ϕ and it is called at the end of the algorithm when needed. It computes the parameter using (25) and estimates the variance for the return in R. However since we have $Var(\mathbf{Z}) = V(\boldsymbol{\mu})\phi$, which is a vector, following the method used by Wilhelm in its code we estimate the variance as the weighted mean of the vector, i.e.:

$$\widehat{Var}(\mathbf{Z}) = \frac{1}{n}\sum_i \left(\frac{V(\hat{\mu}_i)}{\hat{\mu}_i}\hat{\phi}\right) \tag{26}$$

where $\hat{\phi}$ is replaced by true ϕ if equal to 1 or if supplied by the user.

3.4 GCV computation

As we highlighted in the theoretical section, generalized cross validation criterion is used for the selection of best λ via grid search minimization. It is computed when asked by the user with the `GCV` flag in the R interface and requires the computation of the degrees of freedom of the model. Moreover the user can choose to perform the *exact* computation of DoFs or to adopt a more pragmatic *stochastic* approximation of them, which requires less computation [5].
In particular for FPIRLS, GCV computation is performed for each λ at the convergence of the algorithm. This basically requires the estimation of DoF only for the last iteration of the `while` loop in the `apply` method, but in general there is no way to infer a priori the number of iterations required for convergence. Moreover, the estimates of GCV for gaussian data make use of a $\mathbb{R}^n$ norm in (24), while in general the norm induced by the deviance function of the distribution is required. For these reasons we could not exploit the private method `computeDegreesOfFreedom` in `MixedFERegressionBase` which is designed for being executed inside its own `apply`, but we had to make it public.
This way we allow the usage of `computeDegreesOfFreedom` independently from `apply` and in particular we can execute it when FPIRLS convergence is reached. We highlight the fact that no additional computation is performed w.r.t. the old method, since when `computeDegreesOfFreedom` is called the regression object stored in the `FPIRLS` class is the same as in the last iteration of the `while` loop and no matrix has to be re-computed.
Finally, in order to delay the DoF computation we set the `DOF` parameter used by the `MixedFERegression` object always as false and store the a copy of the actual parameter in `RegressionDataGAM`.
In the last version of `fdaPDE` package a further feature has been added, in particular the user can insert his own DoF estimates in a `MatrixXr` object called `DOF_Matrix` (matrix since it is designed also for spatio-temporal models) such that when the matrix is present the DoF computation step is skipped and the user data exploited for GCV computation. This same feature has also been extended for the GSR case.

3.5 Minor changes

Alongside with the implementation of our method we also bring some minor changes to the rest of the library.

3.5.1 GCV bug fixing

The most important of this minor changes concerns the detection and fixing of a bug in the *exact* computation of GCV. In particular in the case of a model with no covariates and with location coinciding with mesh nodes the old version of the library produced an *exact* DoF always equal to 0, thus the model always tends to overfit. The bug was present both in the `1.0` version of the `fdaPDE` (the one available on CRAN) and on the most recent `1.1` version. This behaviour was due to a missing option in the conditional statements inside the `computeDegreesOfFreedomExact` implementation. The case with no covariates and location on the mesh nodes was not considered and its actual computation skipped.
The expected behaviour has been restored as shown in section (5.4).

3.5.2 Warning message in GCV

We realized several of our tests have been initially carried on with a λ which was not optimal. This was due to the fact that the optimal lambda was not in the sequence of lambdas specified by the user for the GCV minimum search, thus the *best* λ returned by the method was on the boundary of that sequence and not in a stationary point. In order to avoid this kind of misinterpretation we include a `warning` message for these situations within the R wrapper function.

3.5.3 Areal option included

A discrepancy was present between the areal model of `fdaPDE` and the areal model in Wilhelm R code. In particular, `fdaPDE` always implemented the model version with the average of the function on the subdomain D_i, while in Wilhelm model an approach considering the sum of the function was implemented. The model considering the sum is critical in many application, especially when dealing with counting data (i.e. Poisson models) as done for the crime study in Portland by [14]. The main difference in the code implementation consists in the diagonal matrix $\mathbf{A}$, which elements correspond to $\frac{1}{|D_i|}$ in `fdaPDE` model and to 1 in Wilhelm one.
We slightly modify the set method of the matrix $\mathbf{A}$ in the code such that the user can decide the desired model to adopt. On the final user interface a new parameter `areal.data.avg` has been added to account for this option.

4 Installation

The source code described in this report can be found and downloaded from `https://github.com/was-albi/PACS_ColomboPerin`, a fork of the official `fdaPDE` repository. We proposed two different methods in order to install the package in the R environment.

Download the `.zip` file from the repository, unzip it, and for the installation choose one of the two following methods:

1. `R console`:
   `install.packages("/path/to/PACS_ColomboPerin-master", type='source', repos=NULL)`
2. From the Terminal:
   `$ R CMD build <path to PACS_ColomboPerin-master>`
   `$ R CMD INSTALL <path name of the R library tree>`

4.0.1 Installation issues

`fdaPDE` depends upon some other R packages for graphical features and for the use of Eigen library, in particular during the installation phase the user may be asked to install manually those R dependencies: `rgl`, `plot3D`, `geometry`, `RcppEigen`, `plot3Drgl`.
In particular, for Linux users, `rgl` installation can cause some issue. If an error like

```
checking for X...  no
configure:  error:  X11 not found but required, configure aborted.
```

or

```
checking GL/glu.h usability...  no
checking GL/glu.h presence...  no
checking for GL/glu.h...  no
configure:  error:  missing required header GL/glu.h
```

occur during the installation process, it can be solved installing

`xorg-dev libx11-dev mesa-common-dev libglu1-mesa-dev`.

4.1 Documentation with Doxygen

To build the Doxygen documentation of the C++ code, you need to go to the `/src` folder and type from the terminal

`doxygen -g <config-file>`.

This will create a configuration file that can be edited to customize the output. Once edited, you need to run Doxygen with the command

`doxygen <config-file>.`

4.2 Quick start

In order to help the users to move the first steps with the `smooth.FEM(...)` function for the Generalized Additive Models, we introduce inside the R package the `/tests` folder. It contains some simple R scripts where data are generated and then the model is estimated. The most simple example is the `BINOMIAL_SQUARE2D.R` script, where the simulation is explained in the following section (5.2.1). The script can be described in five steps:

- Loading of library and data: once installed, `fdaPDE` library can be loaded, and then the `BINOMIAL_SQUARE_2D.RData` file containing our example data is loaded as well.
- Description of data variables. In particular we have:
  - `response` a vector where each element is a binomial sample. This is our observation container;
  - `loc` a nx2 matrix where each row specifies the spatial coordinates of the corresponding response sample;
  - `sol_exact` the true field evaluated over the locations;
- Plotting the data variables

```
# mesh
{...}
# mesh with points
{...}
# true field
{...}
# sampled data
{...}
# mu (no covariates)
{...}
```

- Fitting the response

```
# Set the lambda parameter sequence for GCV computation
lambda = 10^seq(-5,0,length.out = 20)
# Fit the observations using the exact GCV method
output_CPP_exact <- fdaPDE::smooth.FEM(location = loc, observations =
    as.numeric(response), FEMbasis =FEMbasis, covariates = NULL, GCV=
   T, GCVmethod = "Exact",lambda = lambda, max.steps=15, fam=FAMILY,
   mu0=NULL, scale.param=NULL)
# Fit the observations using the stochastic GCV method
output_CPP_stoch <- fdaPDE::smooth.FEM(location = loc, observations =
    as.numeric(response), FEMbasis =FEMbasis, covariates = NULL, GCV=
   T, GCVmethod = "Stochastic", lambda = lambda, max.steps=15, fam=
   FAMILY, mu0=NULL, scale.param=NULL)
```

– Save and plot the estimated results

```
# save the best estimated function for both exact and stochastic GCV
   method
f_est_ex = output_CPP_exact$fit.FEM$coeff[,best_lambda_ex]
f_est_stoc = output_CPP_exact$fit.FEM$coeff[,best_lambda_stoch]

# ------------- Results plot --------------

# plot settings
{...}
# estimated field: exact GCV ----------
{...}
# estimated field: Stochastic GCV ------
{...}
```

The others scripts in the `/test` folder contains different examples for different distributions and settings.

5 Simulation study

In this section we will show some simulation studies for the new functionalities added to the library. We will also provide a performance comparison with the experimental R version of the code.
It worth to say we also tested FPIRLS method in all different settings against direct algorithm for the gaussian case (thus when FPIRLS coincide with the direct method) and against the R version where available. This way we assessed the validity of the code, however these tests were not reported for the sake of space.

5.1 Simulation description

In the following we will consider a mesh of K nodes and a set of n observations measured at n location points $\mathbf{p}_i \;\; i = 1, .., n$. Data observations will be generated according to the GAM model

$$g(\boldsymbol{\mu}) = \mathbf{W}\boldsymbol{\beta} + \mathbf{f}_n \quad , \quad z_i \sim \mathcal{D}(z \,|\, \mu_i) \tag{27}$$

where $g(\cdot)$ is the link function, $\mathbf{W}$ is the design matrix containing our covariates vectors, $\boldsymbol{\beta}$ is a vector of fixed parameters, $\mathbf{f_n}$ is the vector of function evaluation at the observation points and the parameter $\boldsymbol{\mu}$ is the vector containing the mean of the distribution $\mathcal{D}(z)$ at the location $\mathbf{p}_i$. The distribution $\mathcal{D}(z)$ is the one from which data are generated and it incorporates the source of noise of our simulation setting.
We asses the goodness of fit by computing the RMSE (Root Mean Square Error), i.e. for a set of M simulations indexed by the index j we have

$$RMSE_f(j) = \sqrt{\sum_{k=1}^{K} \frac{(\mathbf{f}_k - \hat{\mathbf{f}}_k)^2}{K}} \qquad j = 1, ..., M \tag{28}$$

where $\mathbf{f}_k$ is the k-th coefficient of the FE representation of f. RMSE is computed both for *exact* and *stochastic* GCV for the solution with best λ. It enables a valuation of the goodness of fit for the entire spatial field f since it is an estimate of the $L^2(\Omega)$ norm of the error. Moreover, following [14] we also define a spatial analogue of RMSE, which enables the evaluation of the goodness of fit locally on the simulation points $\mathbf{p}_i$

$$SpRMSE_f(\mathbf{p}) = \sqrt{\sum_{j=1}^{M} \frac{(f(\mathbf{p}) - \hat{f}_j(\mathbf{p}))^2}{M}} \tag{29}$$

we will refer to this last performance index as *Spatial field RMSE* (thus the notation $SpRMSE$).
Alongside with RMSE we will show the distribution of the estimated coefficients $\hat{\boldsymbol{\beta}}$ when covariates are present.

5.2 Simulations

For the pointwise simulation study we will consider the list of set-ups indicated in table (1). We will first show all the new distributions the library can handle in a simple setting, and then we will generalize with some of them to show the possible usage with much complex domains and penalizations.

Distribution	**Mesh**	**Note**	**Sim.**
Binomial	2D-Square	no covariates, 800 locations	(5.2.1)
Gamma	2D-Square	no covariates, 800 locations	(5.2.2)
	2D Mesh C	with covariates	(5.2.5)
	3D Sphere	with covariates	(5.2.8)
Poisson	2D-Square	no covariates, 800 locations	(5.2.6)
	2D Circular Mesh	no covariates, Dirichlet BC, and PDE space	(5.2.6)
	2.5 Hub Mesh	with covariates	(5.2.7)
	2D-Square	areal data, no covariate	(5.2.9)
Exponential	2D-Square	no covariates, 800 locations	(5.2.4)

Table 1: Simulations

5.2.1 Binomial 2D Square domain

We start our simulation chapter with a model for *binomial* distributed values, i.e. we will consider the realization of a binary random variable Z over pointwise locations.
The canonical link function for this type of data is the *logit* function. It is the logarithm of the odds ratio: $log(\frac{p}{1-p})$, where p represent the probability of a positive outcome for the variable Z, while $1-p$ represent the probability of a negative outcome. Its inverse is the sigmoid or logistic function, widely used in mathematics and statistics[2], from which it takes the name[3]. The model descending from it is note in literature as *logistic regression model* and it opens up the possibility of dealing with binary classification problems.

In our simulation study we set the spatial field f as

$$f(x,y) = -\frac{5}{2}\sin(2\pi x)\cos(2\pi y) + \frac{4}{5}\sin(3\pi x) \tag{30}$$

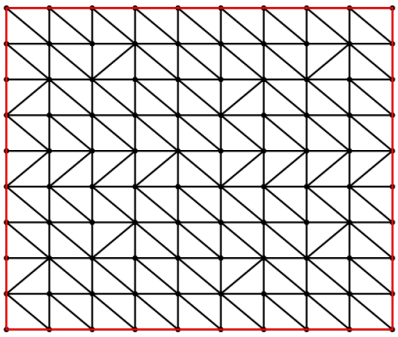
and the data are generated according to the following model

$$\mu(\mathbf{p}) = g^{-1}(f(\mathbf{p})) \quad Z_i \sim Be(\mu(\mathbf{p}_i)) \tag{31}$$

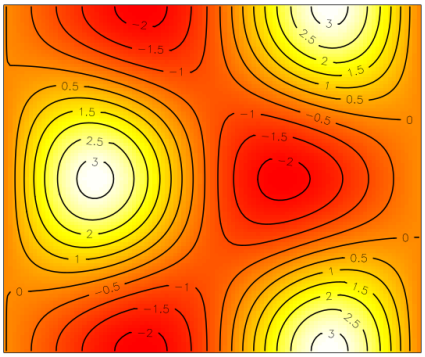
[2]Logistic function was developed as a model of population growth in the first half of IX century and often used in biostatistics from XX century on. More recent applications see it, for instance, as activation function in neural network architectures.

[3]The word logit is indeed a portmanteau of **log**istic-un**it**.

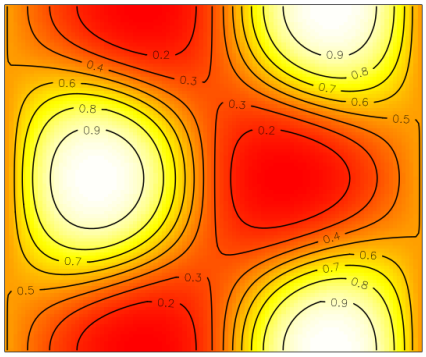
The domain is the squared mesh included in the *data* section of the package. We randomly select $n = 800$ observations within mesh boundaries and consider a simple laplacian penalization.
Simulations settings and results are shown respectively in Figure (6) and (7).



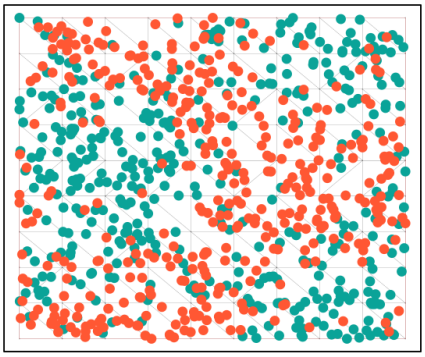
(a) Squared Mesh



(b) True Field

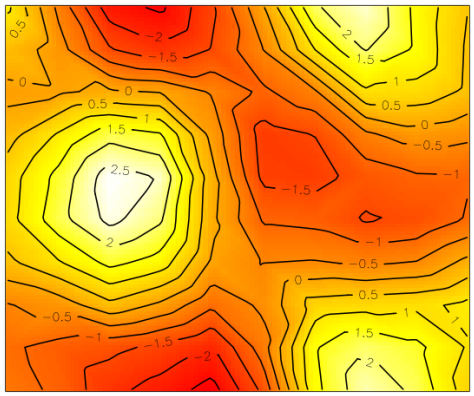


(c) Mean parameter ($\mu(\mathbf{p})$)

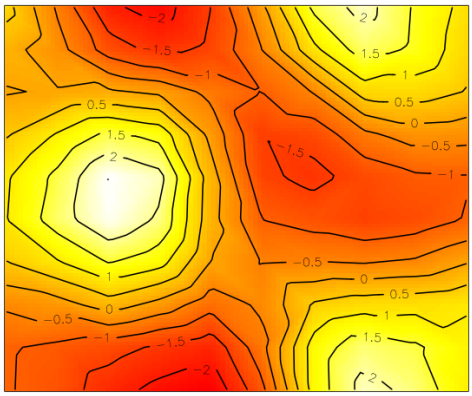


(d) Samples: true values green circles, false values red crosses.

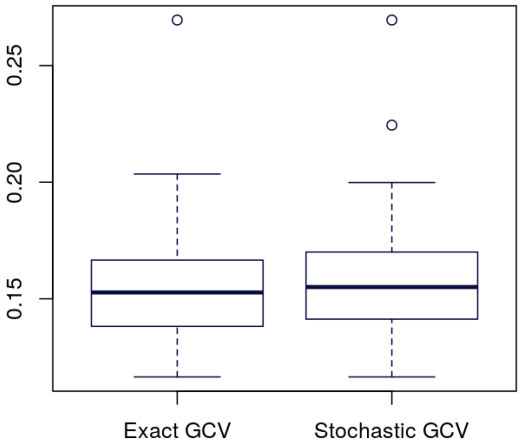
Figure 6: Simulation settings for *binomial* data. $n = 800$ observations on a simple squared mesh. Laplace penalization Δf.



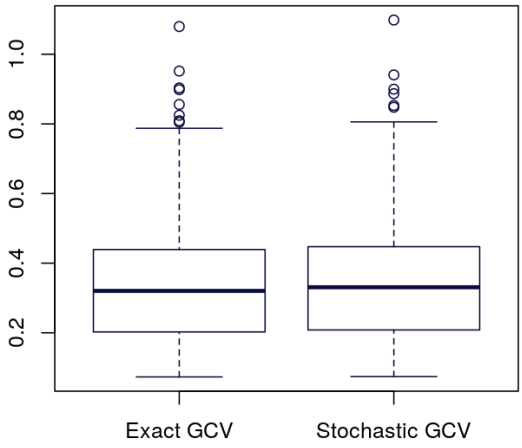
(a) Fitted field (optimal lambda, exact GCV)



(b) Fitted field (optimal lambda, stochastic GCV)



(c) RMSE: Exact vs Stochastic



(d) Spatial field RMSE: Exact vs Stochastic

Figure 7: Logistic regression (*binomial* data) on 2D squared domain. Fitted fields reconstruct well the spatial field shown in Figure 1.

5.2.2 Gamma 2D Square domain

In this simulation we will consider data generated according to a *gamma* distribution. Gamma distribution is a continuous long-tailed distribution generally defined by two parameters called *shape* (α) and rate (β). It is a well known distribution in statistics and information theory, where it is note for being the *maximum entropy probability distribution*[4] for a positive random variable X for which $\mathbb{E}[X] = \mu = \frac{\alpha}{\beta}$. In other words, it can be seen as the distribution which minimizes the amount of prior information built into it. It is interesting to notice that many physical systems tend to move towards maximal entropy configurations over time according to the second law of thermodynamics, thus the gamma distribution can be seen as a more natural assumption for those kind of systems than gaussian one.
Moreover, gamma distribution is commonly used to model waiting times: events that occur independently with some average rate (β) are modeled with a Poisson process and the waiting times between α occurrences of the event are gamma distributed.

In our simulation study we set the spatial field f as

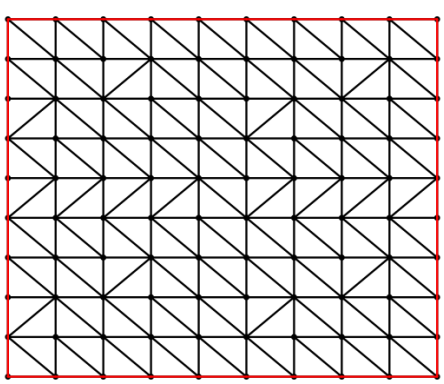
$$f(x, y) = -\frac{3}{2}\sin(2\pi x)\cos(2\pi y) + \frac{2}{5}\sin(3\pi x) \tag{32}$$

and we generate data according to the following model

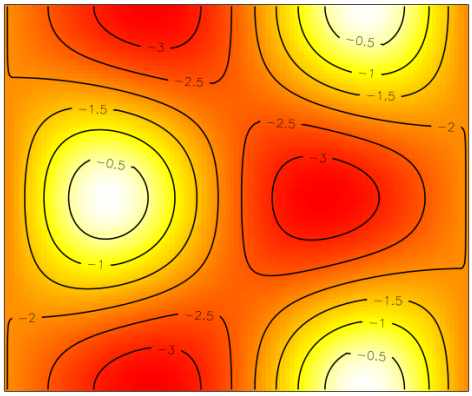
$$\mu(\mathbf{p}) = g^{-1}(f(\mathbf{p})) \quad Z_i \sim Gamma(\mu(\mathbf{p}_i), 1) \tag{33}$$

The domain is the squared mesh included in the *data* section of the package. We randomly select $n = 800$ observations within mesh boundaries and consider a simple laplacian penalization.
Simulations settings and the results are respectively shown in Figure (8) and (9).

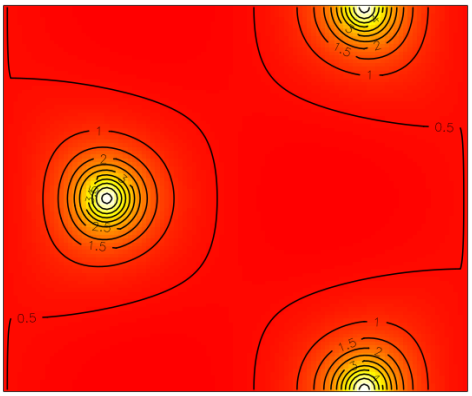
[4]The information entropy, introduced by Claude Shannon in 1948, is a basic quantity in information theory associated to any random variable, which can be interpreted as the average level of "information", or "uncertainty" inherent in the variable's possible outcomes.



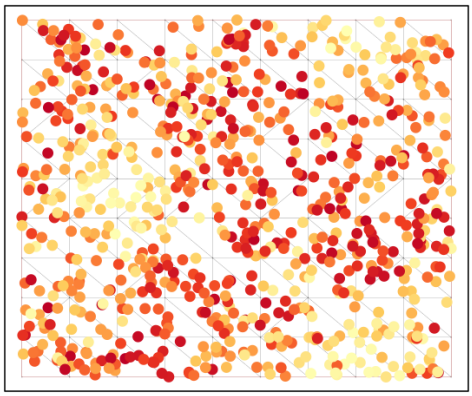
(a) Squared Mesh



(b) True Field

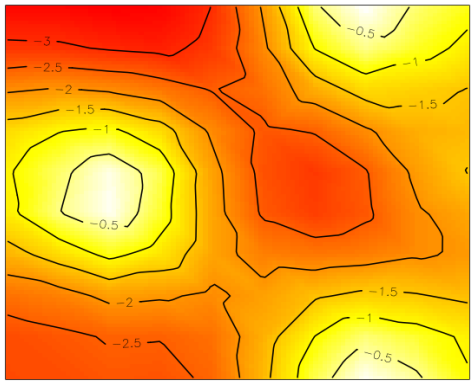


(c) Mean parameter ($\mu(\mathbf{p})$)

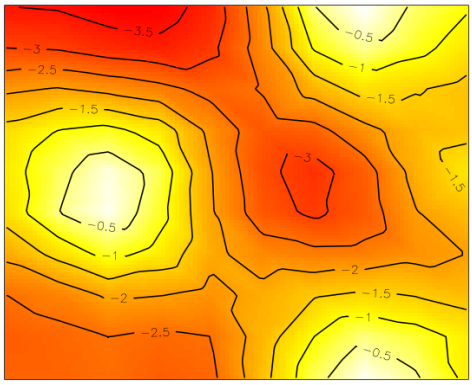


(d) Samples: high values yellow, small values red.

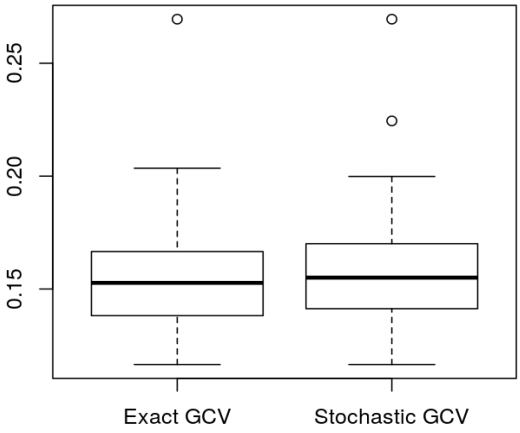
Figure 8: Simulation settings for *gamma* distributed data: $n = 800$ observations on a simple squared mesh. Standard laplace penalization Δf is used.



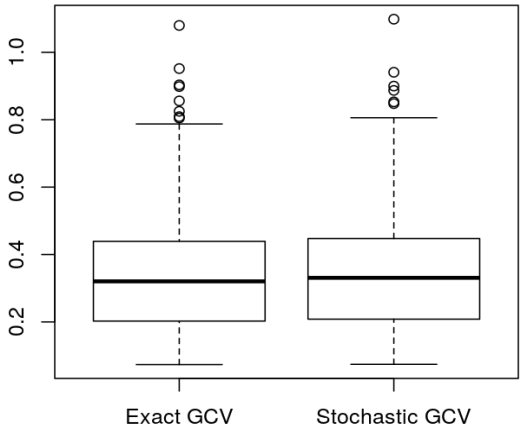
(a) Fitted field (optimal lambda, exact GCV)



(b) Fitted field (optimal lambda, stochastic GCV)



(c) RMSE: Exact vs Stochastic



(d) Spatial field RMSE: Exact vs Stochastic

Figure 9: Gamma regression on 2D squared domain. Fitted fields reconstruct well the spatial field shown in Figure 8. Notice that the estimated field is the same in the exact and stochastic GCV case, indeed the same λ has been selected by the two methods in this case.

5.2.3 Poisson 2D Square domain

In the following we will made use of data generated from a *Poisson* distribution. The Poisson distribution is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time (or space) when these events occur with a constant mean rate (λ) and independently of the time since the last event. It is commonly used to model counting data and has numerous applications when combined with a spatial model as in our case. For instance, application of the pointwise model can be the evaluation of the richest regions for fishing in a given reservoir. Indeed, one can register the number of fishes caught by fishermen at each location and reconstruct the underlying fish density. Similarly one can imagine to search for regions of major attraction in a city counting the number of photos posted on social media in a given time-slot[5].

In our simulation study we set the spatial field f as

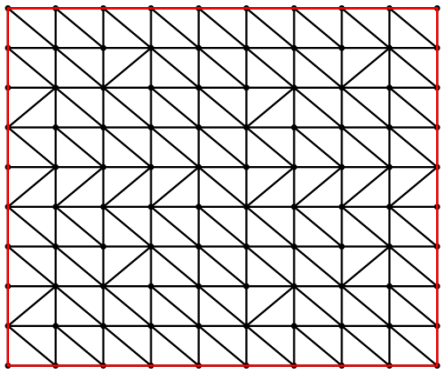
$$f(x, y) = -\frac{3}{2} sin(2\pi x) cos(2\pi y) + \frac{2}{5} sin(3\pi x) + 2 \tag{34}$$

and the data are generated according to the following model

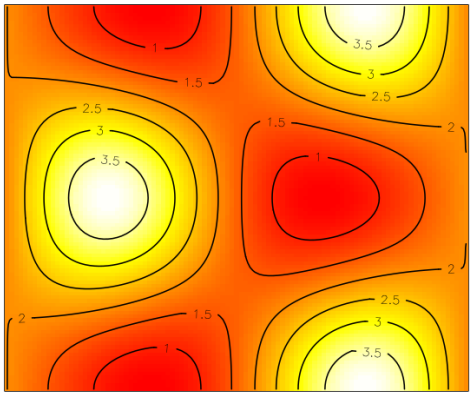
$$\mu(\mathbf{p}) = g^{-1}(f(\mathbf{p})) \quad Z_i \sim Poisson(\mu(\mathbf{p}_i)) \tag{35}$$

The domain is the squared mesh included in the *data* section of the package. We randomly select $n = 800$ observations within mesh boundaries and consider a simple laplacian penalization.
Simulations settings and the results are respectively shown in Figure (10) and (11).

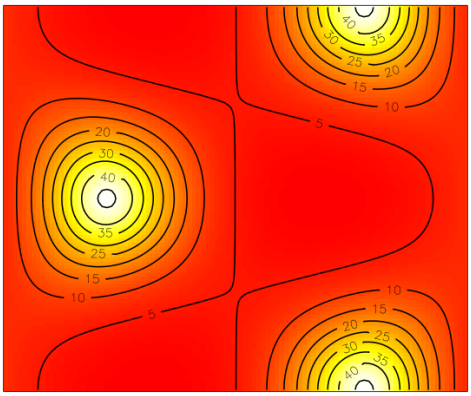
[5] Often metadata on social media contents register a geo-reference of where the content has been generated.



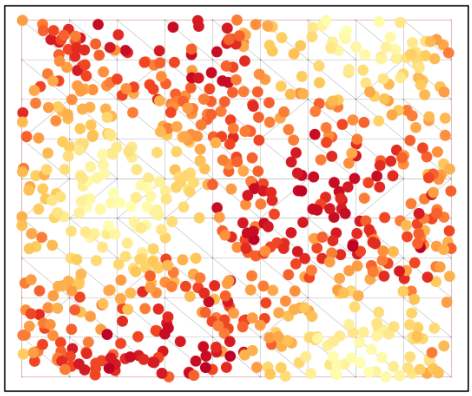
(a) Squared Mesh



(b) True Field

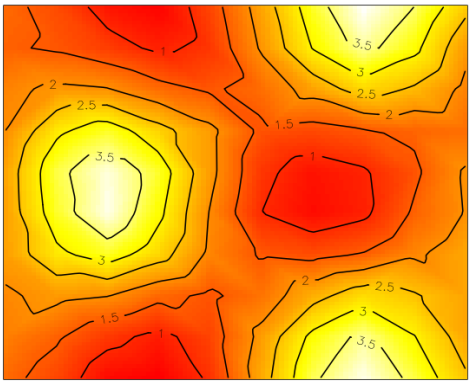


(c) Mean parameter ($\mu(\mathbf{p})$)

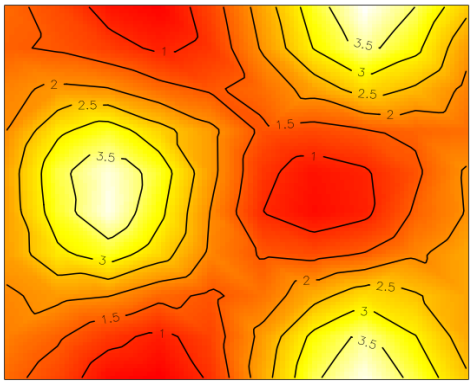


(d) Samples: high values yellow, small values red.

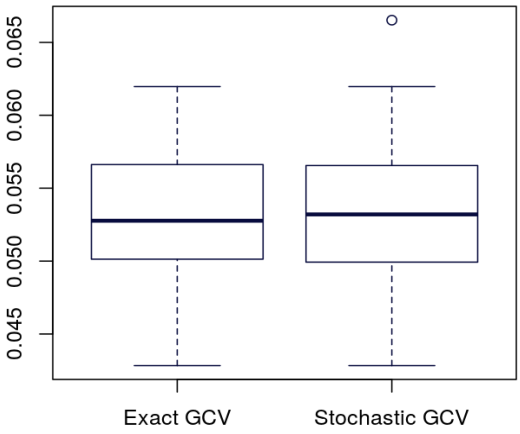
Figure 10: Simulation settings for *Poisson* distributed data: $n = 800$ observations on a simple squared mesh. Standard laplace penalization Δf is used.



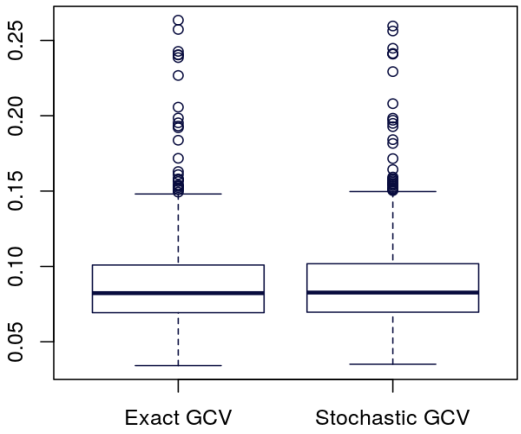
(a) Fitted field (optimal lambda, exact GCV)



(b) Fitted field (optimal lambda, stochastic GCV)



(c) RMSE: Exact vs Stochastic



(d) Spatial field RMSE: Exact vs Stochastic

Figure 11: Poisson regression on 2D squared domain. Fitted fields reconstruct well the spatial field.

5.2.4 Exponential 2D Square domain

We use here a model for *exponential* distributed values. The exponential distribution is a special type of gamma distribution which describes the time for a continuous process to change state. It is often implemented apart from gamma distribution in many packages since it requires a single parameter (the mean) and it is thus of much easy setting.

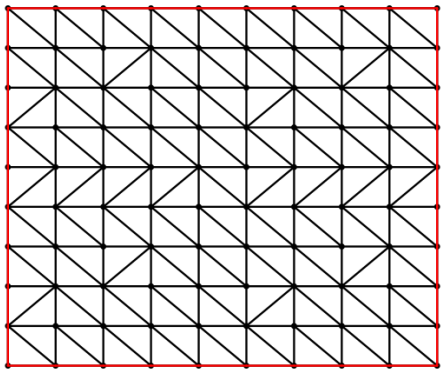
In our simulation study we set the spatial field f as

$$f(x, y) = -\frac{3}{2}\sin(2\pi x)\cos(2\pi y) + \frac{2}{5}\sin(3\pi x) \tag{36}$$

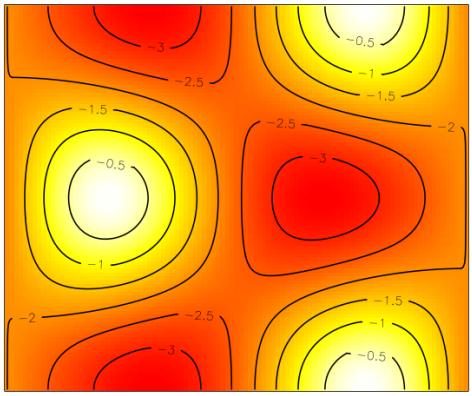
and the data are generated according to the following model

$$\mu(\mathbf{p}) = g^{-1}(f(\mathbf{p})) \quad Z_i \sim Exp(\mu(\mathbf{p}_i)) \tag{37}$$

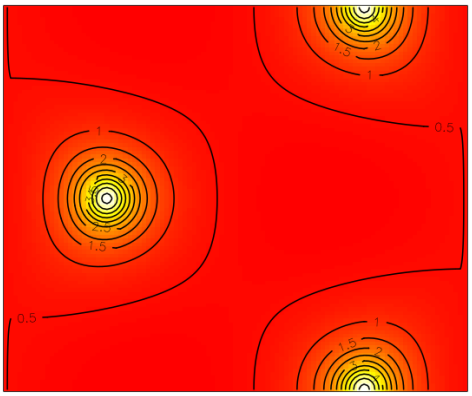
The domain is the squared mesh included in the *data* section of the package. We randomly select $n = 800$ observations within mesh boundaries and consider a simple laplacian penalization. Simulations settings and the results are respectively shown in Figure (12) and (13).



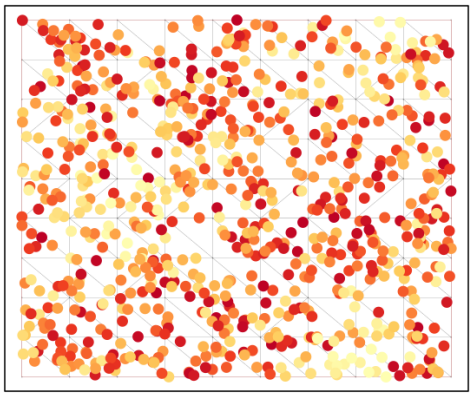
(a) Square Mesh



(b) True Field

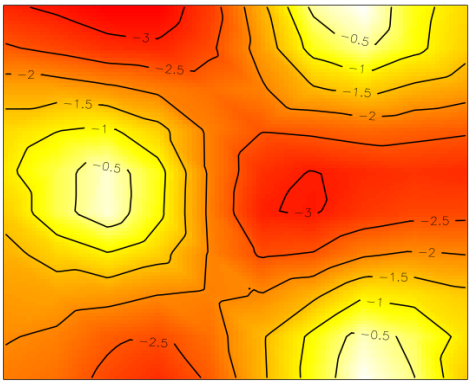


(c) Mean parameter ($\mu(\mathbf{p})$)

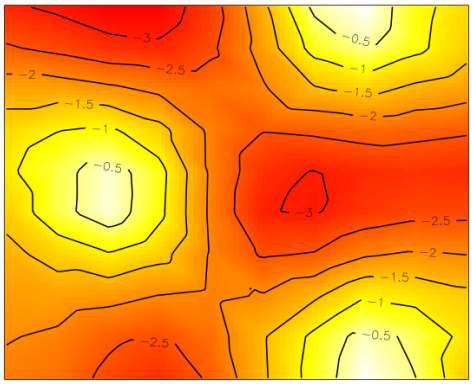


(d) Samples: high values yellow, small values red.

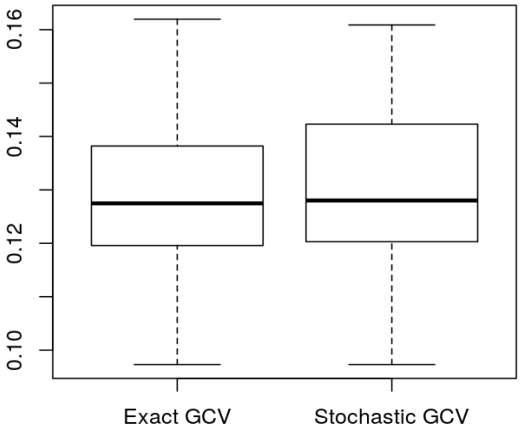
Figure 12: Simulation settings for *exponential* data: $n = 800$ observations on a simple squared mesh. Standard laplace penalization Δf is used.



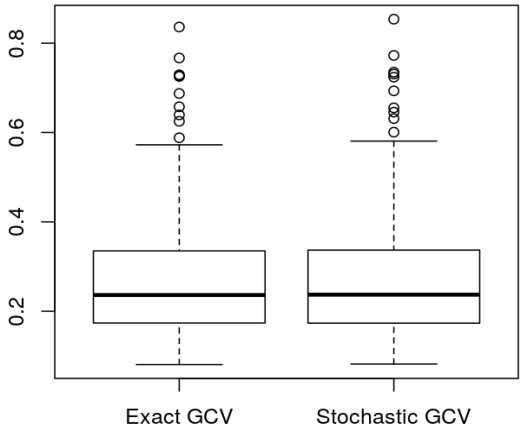
(a) Fitted field (optimal lambda, exact GCV)



(b) Fitted field (optimal lambda, stochastic GCV)



(c) RMSE: Exact vs Stochastic



(d) Spatial field RMSE: Exact vs Stochastic

Figure 13: Gamma regression on 2D squared domain. Fitted fields reconstruct well the spatial field shown in Figure 12.

5.2.5 Gamma 2D C-Shaped domain

We use here a model for *gamma* distributed values, registered over a much complex 2D domain. In particular let $a(x, y)$ and $d(x, y)$ be the following functions

$$a(x,y) = \begin{cases} \frac{\pi}{4} + x, & \text{if } x \geq 0\,,\ y > 0 \\ -\frac{\pi}{4} - x, & \text{if } x \geq 0\,,\ y \leq 0 \\ -\frac{1}{2} arctan(\frac{y}{x}), & \text{if } x < 0 \end{cases} \tag{38}$$

$$d(x,y) = \begin{cases} -\frac{1}{2} + y, & \text{if } x \geq 0\,,\ y > 0 \\ -\frac{1}{2} - y, & \text{if } x \geq 0\,,\ y \leq 0 \\ \sqrt{x^2 + y^2} - \frac{1}{2}, & \text{if } x < 0 \end{cases} \tag{39}$$

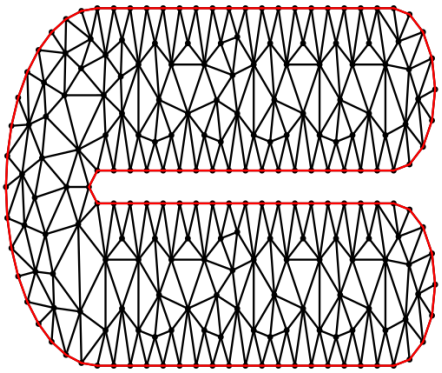
then our spatial field is given by

$$f(\mathbf{p}) = f(x,y) = a(x,y) + d(x,y)^2 \tag{40}$$

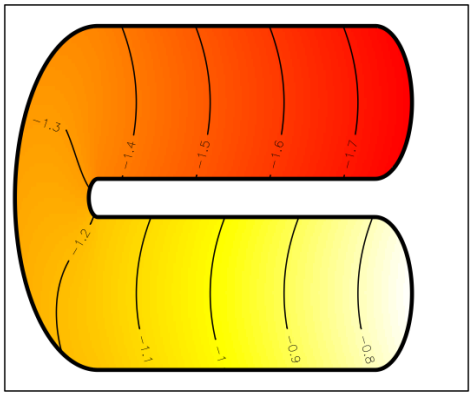
and the data are generated according to the following model

$$g^{-1}(\mu(\mathbf{p})) = \theta(\mathbf{p}) = \mathbf{W}\boldsymbol{\beta} + f(\mathbf{p}) \quad Z_i \sim Gamma(\mu_i, 1) \tag{41}$$

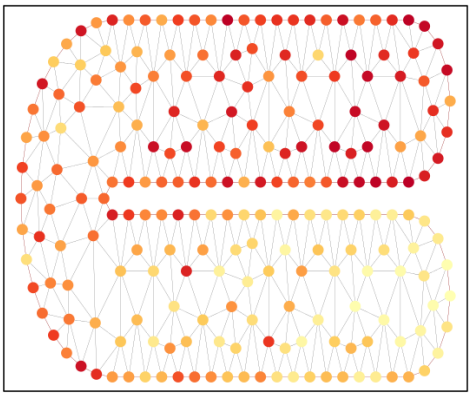
where $\boldsymbol{\beta} = (-0.4, 0.3)^t$, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix}$ with $\mathbf{w}_1 \sim beta(1.5, 2)$, $\mathbf{w}_2 \sim beta(3, 2)$.
The domain is the C-shaped mesh included in the *data* section of the package. The observations are $n = 228$ and their location coincides with mesh nodes.
Settings and results are shown respectively in Figure (14) and (15).



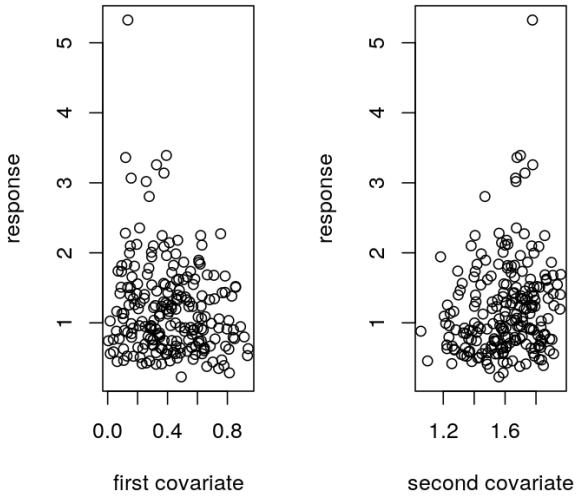
(a) C-Mesh



(b) True Field

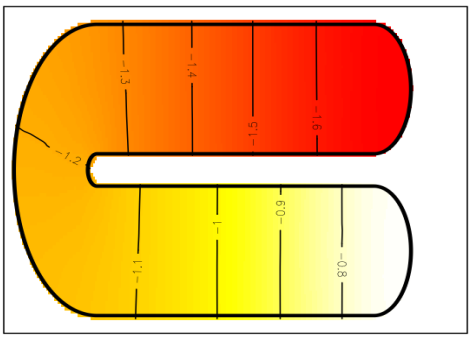


(c) Samples: high values yellow, small values red

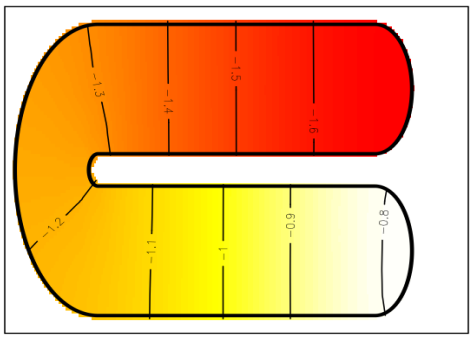


(d) Covariates

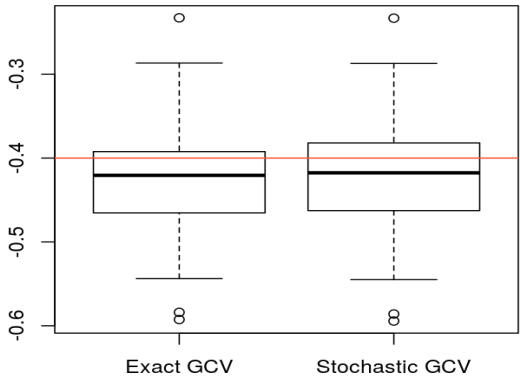
Figure 14: Simulation settings for *gamma* data: 2D mesh C domain with $n = 228$ locations coinciding with mesh nodes. Samples seems to follow an inverse pattern w.r.t. spatial field due to the nature of gamma inverse link ($-\frac{1}{\theta}$). Responses are also shown in function of covariates: slightly negative and positive trends corresponding to β values are visible.



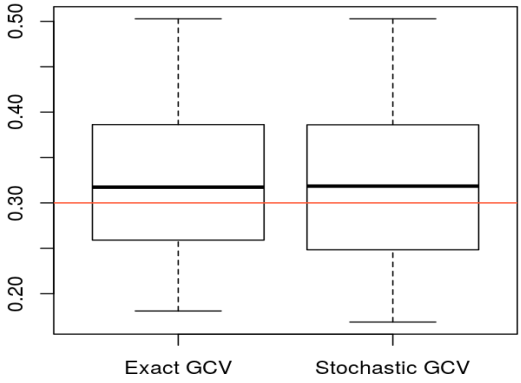
(a) Fitted field (optimal lambda, exact GCV)



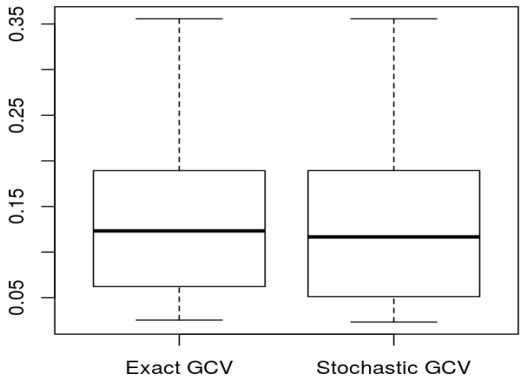
(b) Fitted field (optimal lambda, stochastic GCV)



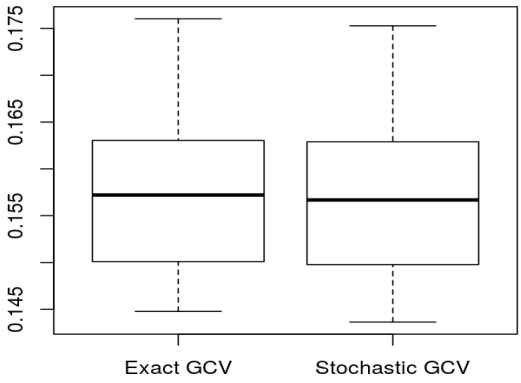
(c) boxplot of β_1 distribution around true value (red line)



(d) boxplot of β_2 distribution around true value (red line)



(e) RMSE: Exact vs Stochastic



(f) Spatial field RMSE: Exact vs Stochastic

Figure 15: Gamma regression over a 2D C-shaped domain with covariates and locations coinciding with mesh nodes. Fitted fields reconstruct well the spatial field and follow the geometry of the domain appropriately. The same is true for beta coefficients which tend to the right β values. Both RMSE and spatial RMSE present good results.

5.2.6 Poisson 2D Circular domain

We use a model for *Poisson* distributed values, including PDE penalization. Including complex PDE penalization in the model can be interesting since allow us to include prior knowledge in the model. In the example of fishermen introduced in section () for instance, one can include information about known water flows in the reservoir or model schools of fish through a diffusion process.

In our simulation study we set the spatial field f as

$$f(\mathbf{p}) = f(x, y) = \begin{cases} 0, & \text{if } x^2 + y^2 = 0 \\ 2\sin(\sqrt{(x^2 + y^2)}\frac{\pi}{2}), & \text{otherwise} \end{cases} \tag{42}$$

We model the PDE used as penalty as a diffusion process with homogeneous Dirichlet boundary conditions:

$$\begin{cases} -div(K\nabla f) = u, & \text{in } \Omega \\ f = 0, & \text{on } \partial\Omega \end{cases} \tag{43}$$

where the diffusion matrix is:

$$K = \begin{bmatrix} \frac{5}{6} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \tag{44}$$

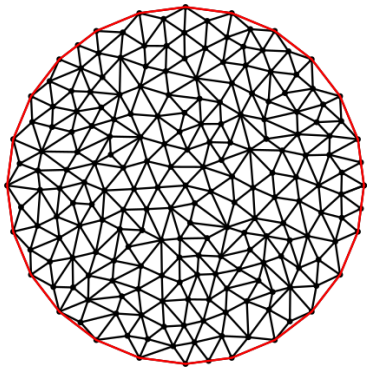
with the forcing term

$$u = \begin{cases} -1, & \text{if } x^2 + y^2 < 1 \\ 0, & \text{otherwise} \end{cases} \tag{45}$$

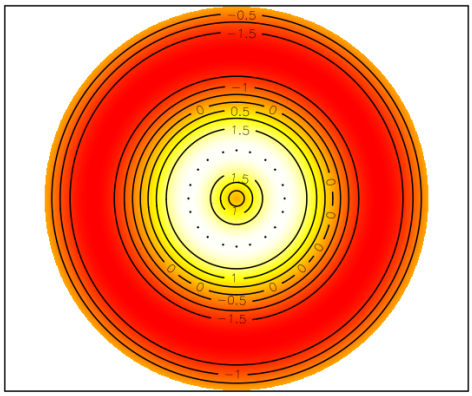
The data are thus generated according to the following model

$$g^{-1}(\mu(\mathbf{p})) = \theta(\mathbf{p}) = f(\mathbf{p}), \quad Z_i \sim Poisson(\mu_i) \tag{46}$$

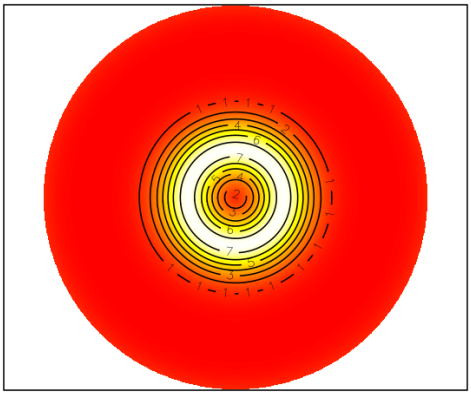
The domain is a 2D circular mesh of radius 4 generated. The observations are $n = 611$ of which 25 coinciding with boundary nodes and 587 have a location randomly selected within the domain.
Simulations settings and the results are respectively shown in Figure (16) and (17).



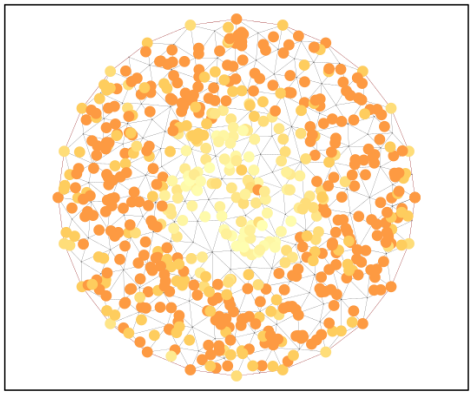
(a) Circular Mesh



(b) True Field

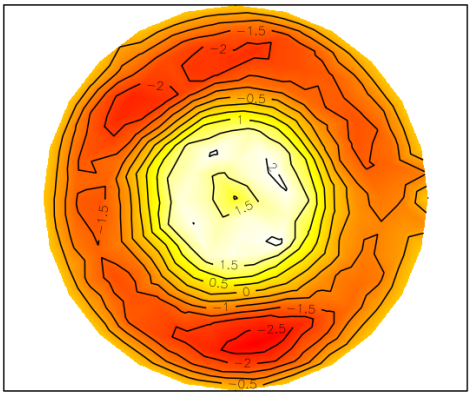


(c) Mean parameter ($\mu(\mathbf{p})$)

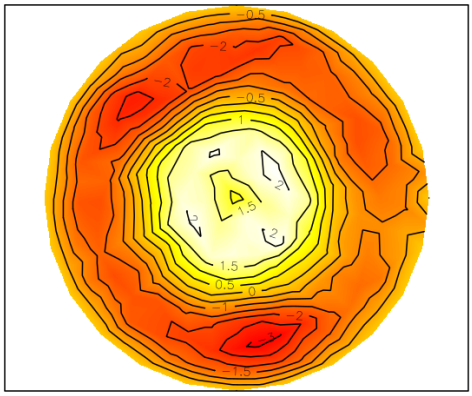


(d) Samples: high values yellow, small values red.

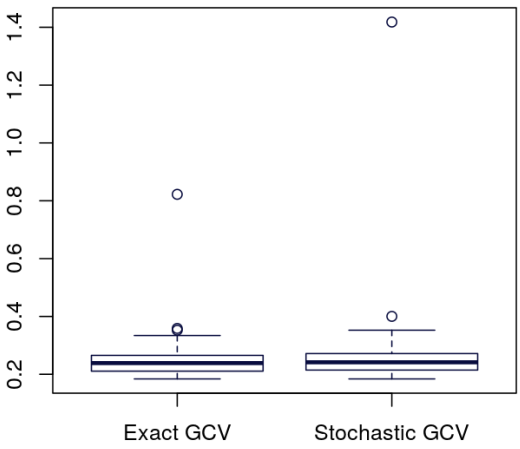
Figure 16: Simulation settings for Poisson data over a 2D circular domain. PDE penalization set according to (43).



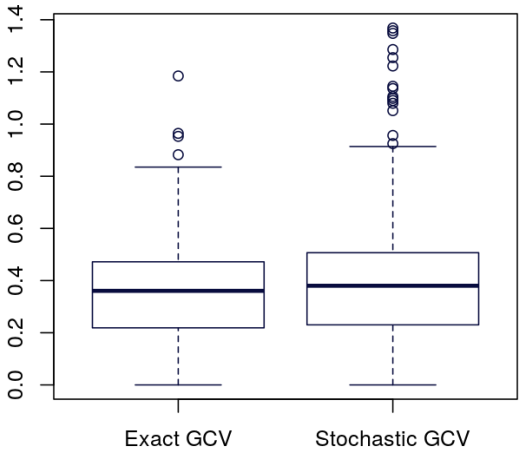
(a) Fitted field (optimal lambda, exact GCV)



(b) Fitted field (optimal lambda, stochastic GCV)



(c) RMSE: Exact vs Stochastic



(d) Spatial field RMSE: Exact vs Stochastic

Figure 17: Poisson regression over 2D circular domain withPDE penalization. Fitted fields reconstruct well the spatial field. Moreover the diffusive process included by PDE 43 influenced the reconstructed field.

5.2.7 Poisson 2.5D Hub domain

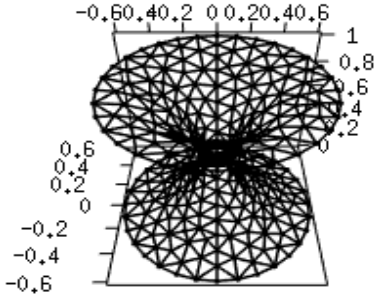
We use a model for *Poisson* distributed values. Here we model a much complex domain given by a 2D surface embedded in a 3D space. The spatial field f is given by

$$f(\mathbf{p}) = f(x, y, z) = a_1\, sin(2\pi x) + a_2\, sin(2\pi y) + a_3\, sin(2\pi z) + 7 \tag{47}$$

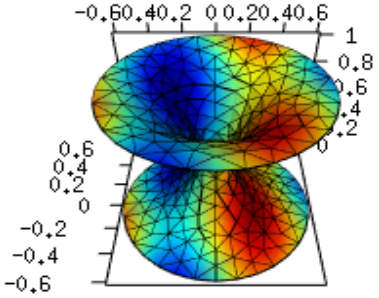
where $a_i \sim \mathcal{N}(0, 1) \quad \forall i = 1, 2, 3$ The data are generated according to the following model

$$g^{-1}(\mu(\mathbf{p})) = \theta(\mathbf{p}) = \mathbf{W}\boldsymbol{\beta} + f(\mathbf{p}) \quad Z_i \sim Poisson(\mu_i) \tag{48}$$

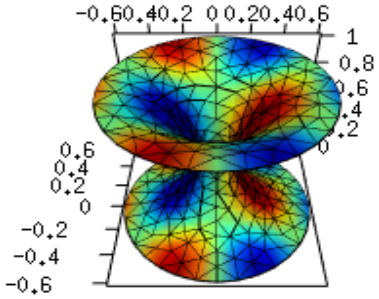
where $\boldsymbol{\beta} = (2, -3)^t$, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix}$ with $\mathbf{w}_1 = sin(2\pi x)cos(2\pi y)$, $\mathbf{w}_2 \sim \mathcal{N}(2, 0.1)$.
The domain is the Hub mesh included in the *data* section of the package. The observations are $n = 340$ and their location coincides with mesh nodes.
Simulations settings and the results are respectively shown in Figure (18) and (19).



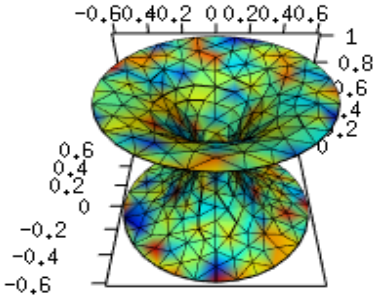
(a) Mesh 2.5D Hub



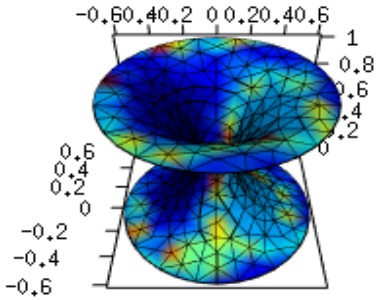
(b) True field



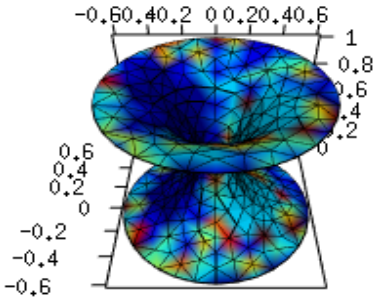
(c) First covariate



(d) Second covariate

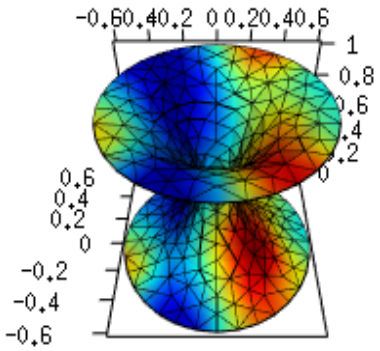


(e) Mean parameter ($\mu(\mathbf{p})$)

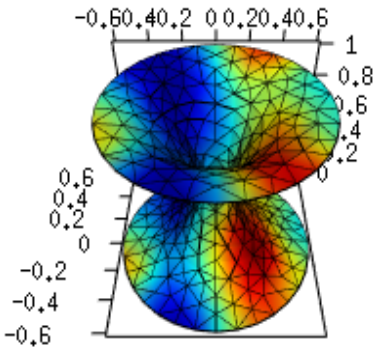


(f) Poisson realizations

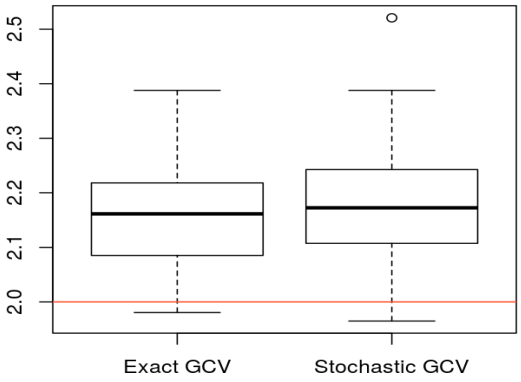
Figure 18: Simulations settings for *poisson* distributed data over a 2.5 hub domain. Observations are shown as a continuous interpolation of their discrete values.



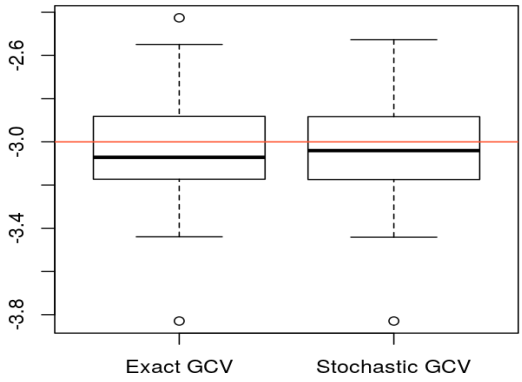
(a) Fitted field (optimal lambda, exact GCV)



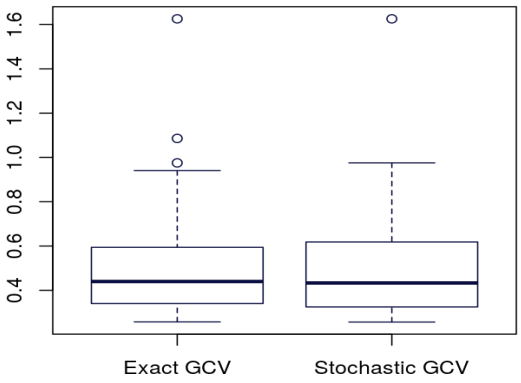
(b) Fitted field (optimal lambda, stochastic GCV)



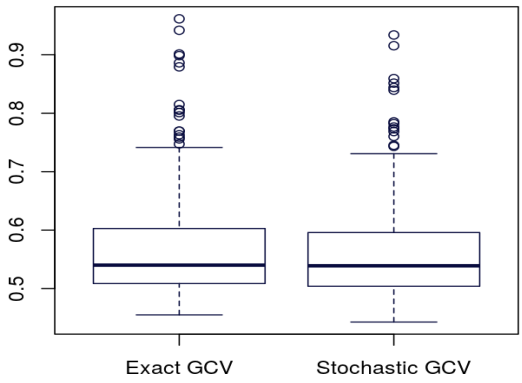
(c) boxplot of β_1 distribution around true value (red line)



(d) boxplot of β_2 distribution around true value (red line)



(e) RMSE: Exact vs Stochastic



(f) Spatial field RMSE: Exact vs Stochastic

Figure 19: Poisson regression over a 2.5D hub domain. Fitted fields approximate well the true field shown in 18b. RMSE and beta estimates shown good results too.

5.2.8 Gamma 3D Sphere

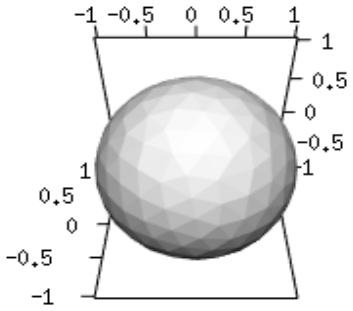
We now present a model for *gamma* distributed values over a 3D domain.
The spatial field f is given by

$$f(\mathbf{p}) = f(x, y, z) = -\,sin(x) - 2\,sin(y) - 3\,sin(z) - 7 \tag{49}$$

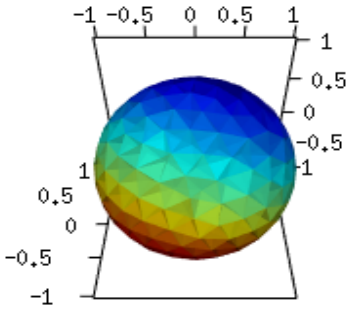
The data are generated according to the following model

$$g^{-1}(\mu(\mathbf{p})) = \theta(\mathbf{p}) = \mathbf{W}\boldsymbol{\beta} + f(\mathbf{p}) \quad Z_i \sim Gamma(\mu_i, 1) \tag{50}$$

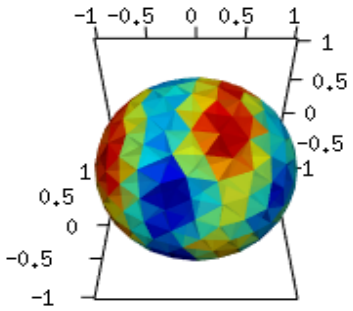
where $\boldsymbol{\beta} = (0.2, 0.4)^t$, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix}$ with $\mathbf{w}_1 = \sin(2\pi x) + sin((2\pi y)^2)$, $\mathbf{w}_2 = cos(-2\pi z)$.
The domain is the Sphere mesh included in the *data* section of the package. The observations are $n = 1069$, with 587 of these coinciding with mesh nodes and 482 randomly selected within the domain.
Simulations settings and results are respectively shown in Figure (20) and (21, (22)).



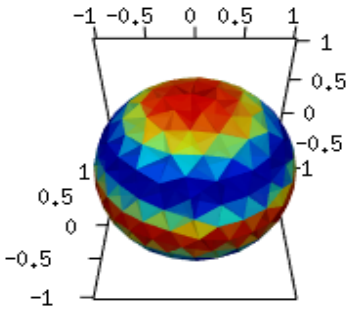
(a) Mesh Sphere



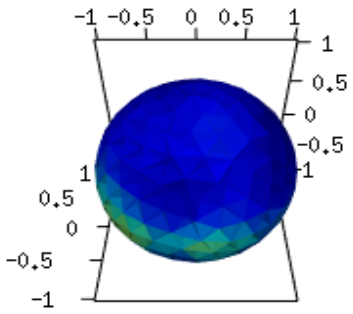
(b) True field



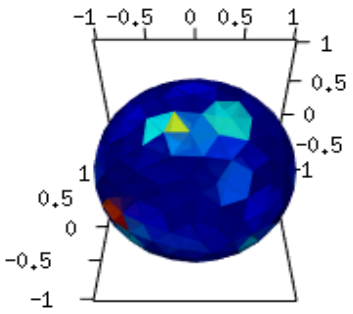
(c) First covariate



(d) Second covariate

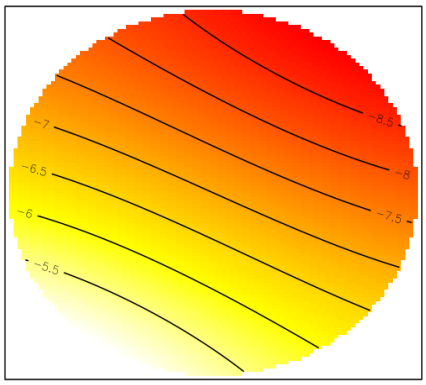


(e) Mean parameter ($\mu(\mathbf{p})$)

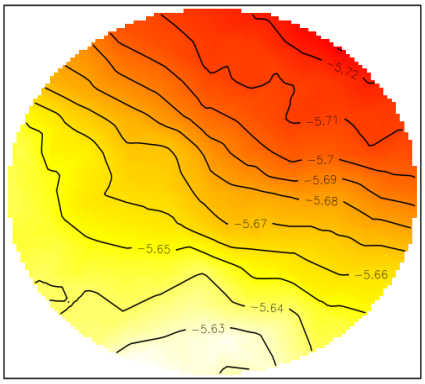


(f) Gamma samples

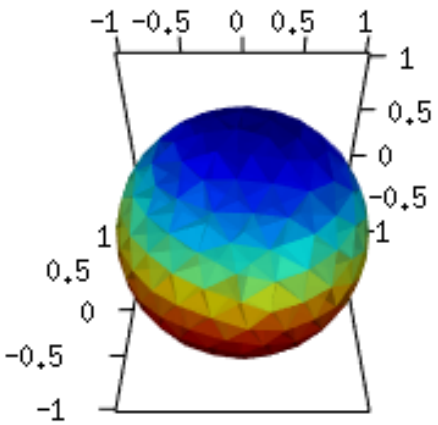
Figure 20: Gamma distributed values over a 3D spherical domain. Long tails of gamma distribution are visible in the central figure of the top panel. Covarates are shown alongside with true field and its realization over a 2D plane ($z = 0$) intersecting the spherical domain.



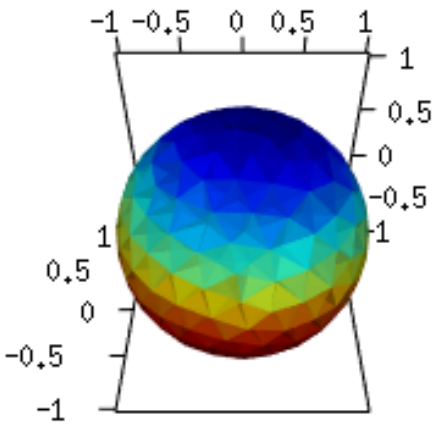
(a) 2D section true field (plane $z = 0$)



(b) Fitted field section (exact GCV)

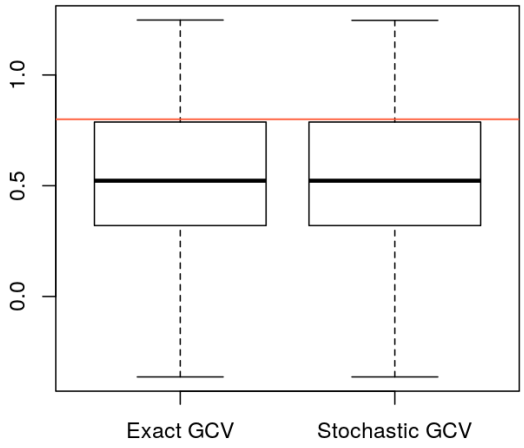


(c) Fitted field (optimal lambda, exact GCV)

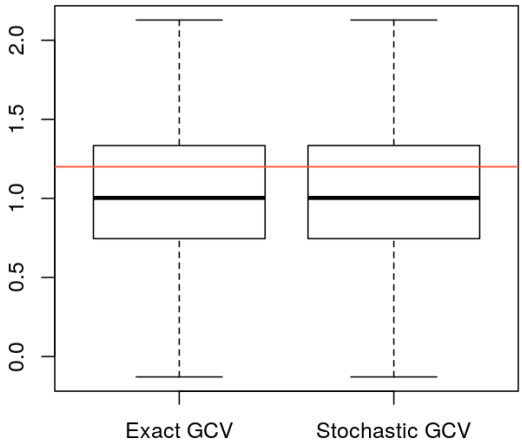


(d) Fitted field (optimal lambda, stochastic GCV)

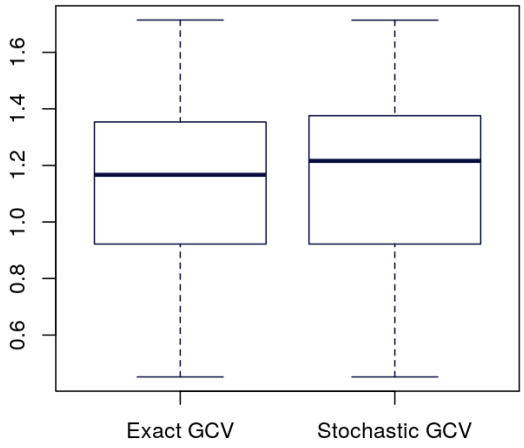
Figure 21: Gamma regression over 3D spherical domain. True fields estimation and their value over the 2D section of the sphere for $z = 0$. Results approximate well the original field showed in (20b) and (21a).



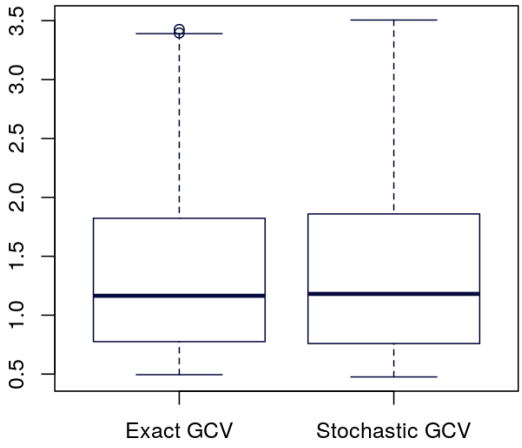
(a) boxplot of β_1 distribution around true value (red line)



(b) boxplot of β_2 distribution around true value (red line)



(c) RMSE: Exact vs Stochastic



(d) Spatial field RMSE: Exact vs Stochastic

Figure 22: Gamma regression over 3D spherical domain. RMSE and estimated β distribution present good results.

5.2.9 Poisson Areal 2D Square domain

Here we present a simulation study for *Poisson* data over areal domains. In this setting the data are measured over subregions of the mesh and considered as the total count of a given datum over these subregions. This kind of model has numerous applications among which one worth a particular mention in those times. Indeed, one can think to model the count of people affected by Covid-19 on each given region of a country and determine which factor influences the spatial distribution of the disease.
For this areal simulation study we used the Square mesh, with n=100 nodes, and 81 observation regions displaced over the domain and forming a cover of it.
The spatial field is the same used in the pointwise simulation, but scaled by a factor of 50. In other words

$$f(\mathbf{p}) = f(x, y) = 50\,(-\frac{3}{2}sin(2\pi x)cos(2\pi y) + \frac{2}{5}sin(3\pi x) + 2) \tag{51}$$

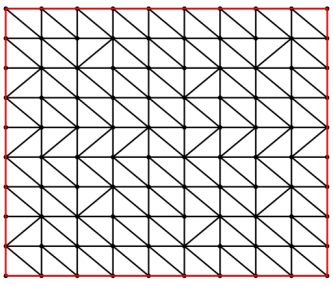
and the resulting model is the following

$$g^{-1}(\mu(\mathbf{p})) = \theta(\mathbf{p}) = \int_{D_i} f(\mathbf{p})\ d\mathbf{p} \quad Z_i \sim Poisson(\mu_i) \tag{52}$$

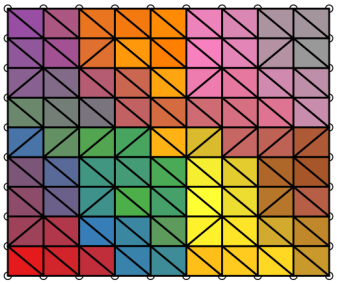
where D_i are the rectangular subdomains shown in Figure (23b). Notice that here, since we have areal observations, the form of spatial RMSE is slightly different and given by

$$RMSE(D_i) = \sqrt{\frac{1}{M}\sum_{j=1}^{M}(\int_{D_i}\hat{f} - \int_{D_i} f)^2} \quad i = 1, ..., 81. \tag{53}$$

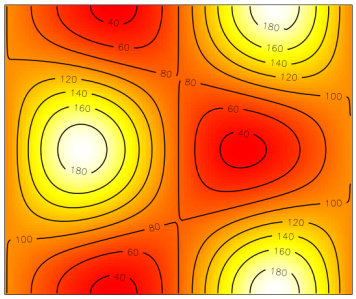
Simulations settings and results are shown respectively in Figure (23) and (24).



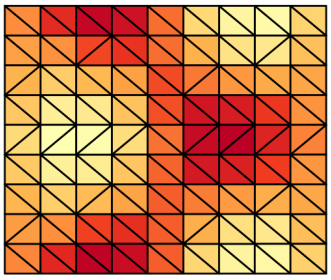
(a) Square Mesh

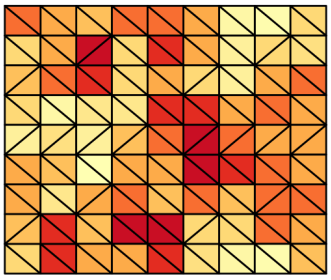


(b) Mesh with regions.



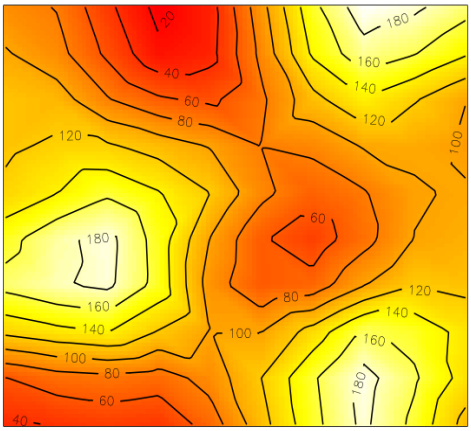
(c) True Field.



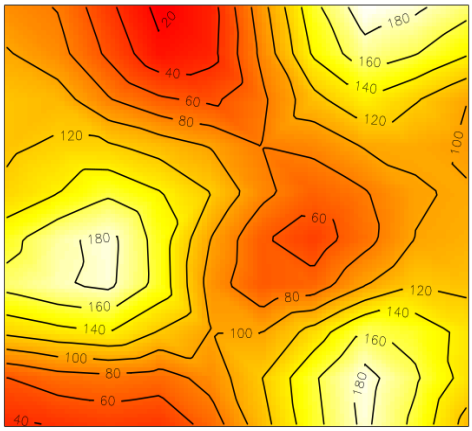


(d) Mean parameter (μ): high values yellow, small values red. (e) Samples over regions: high values yellow, small values red.

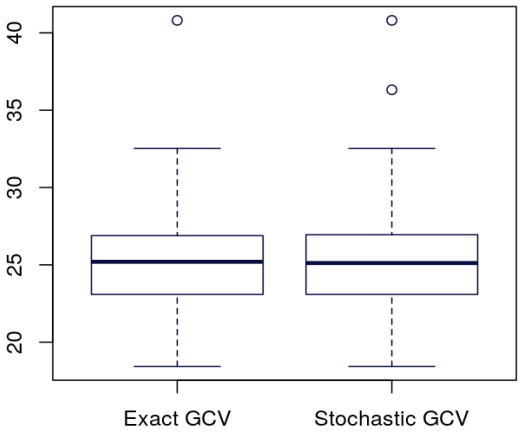
Figure 23: Simulation settings for Poisson areal data over a 2D squared domain. Regions are shown in the top right panel and each of them correspond to a rectangular domain composed by 2 triangles of the mesh. True field is shown alongside with the mean parameter integral and the sampled values over the regions.



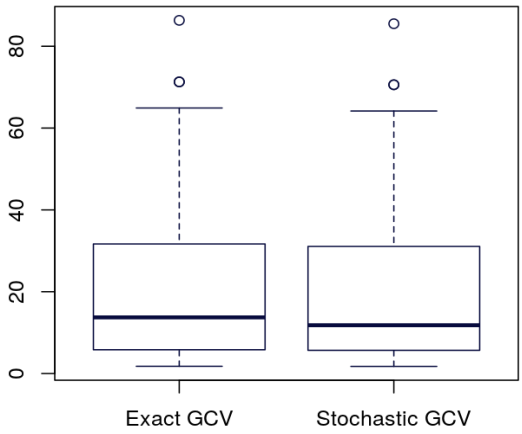
(a) Fitted field (optimal lambda, exact GCV)



(b) Fitted field (optimal lambda, stochastic GCV)



(c) RMSE: Exact vs Stochastic



(d) Spatial RMSE: Exact vs Stochastic

Figure 24: Poisson areal regression over 2D squared domain. Fitted fields reconstruct the spatial pattern of the true field with visible good results. RMSE is good w.r.t. to the true field magnitude.

5.3 R/C++ performance comparison

The implemented method has been tested on a C-shaped mesh with divergent number of nodes and performance compared with the previous R version of the function. Data has been generated according to the following model:

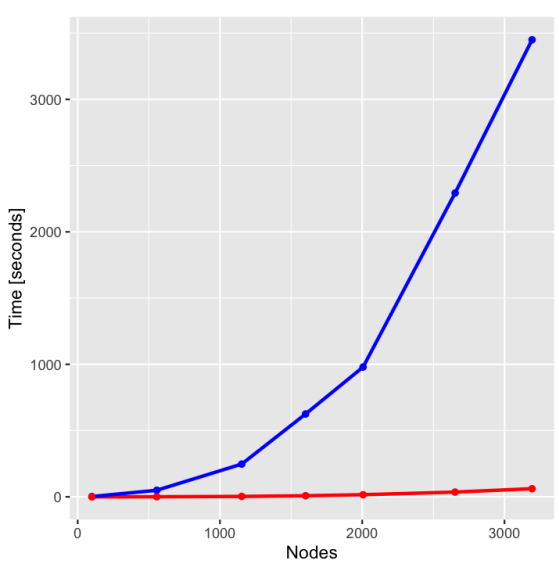
$$f(\mathbf{p}) = f(x, y) = a_1 \sin(2\pi x)\, \cos(2\pi y) + a_2 \sin(3\pi x) \tag{54}$$

with $a_1,\ a_2 \sim \mathbb{U}(-1.5, 1.5)$

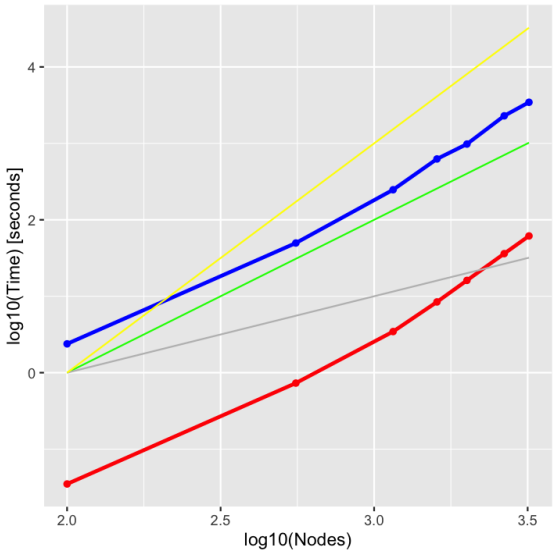
$$g(\boldsymbol{\mu}) = \boldsymbol{\theta} = \mathbf{W}\boldsymbol{\beta} + f(\mathbf{p}) \ , \quad Z_i \sim Be(\mu_i) \tag{55}$$

where $\boldsymbol{\beta} = (-0.4, 0.3)^t$, $\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix}$ with $\mathbf{w}_1 \sim beta(1.5, 2)$, $\mathbf{w}_2 \sim beta(3, 2)$.

Performance results are shown in Figure (25)



(a) Performance comparison R (blue line) vs C++ (red line)



(b) Log-Log Plot. Performance comparison R (blue line) vs C++ (red line) - logarithmic scale in time, where $f(x) = x$ (grey), $f(x) = x^2$ (green),and $f(x) = x^3$ (yellow).

Figure 25: R/C++ performance comparison.
Execution time is significantly lower in the C++ implementation, in particular the discrepancy is much evident for more dense meshes.

5.4 Bug fixing in `fdaPDE 1.1` library

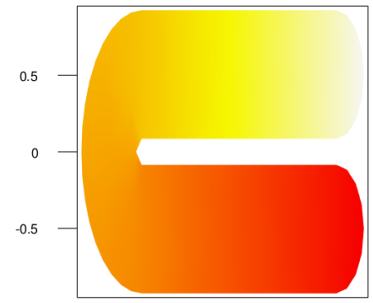
For sake of completeness we report a simulation study for assessing the bug fixing of the GCV method described in section (3.5.1). We refer to the following example.
Consider the spatial field

$$f(\mathbf{p}) = f(x,y) = \begin{cases} \frac{\pi}{4} + x, & \text{if } x \geq 0\,,\ y > 0 \\ -\frac{\pi}{4} - x, & \text{if } x \geq 0\,,\ y \leq 0 \\ -\frac{1}{2} arctan(\frac{y}{x}), & \text{if } x < 0 \end{cases} \tag{56}$$

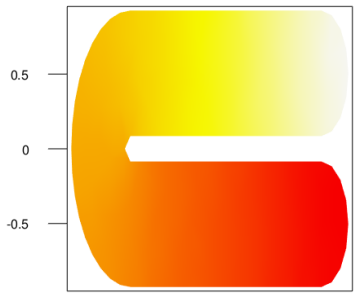
and a test function without covariates and gaussian noise

$$z(\mathbf{p}) = f(\mathbf{p}) + \epsilon\,, \quad \epsilon \sim \mathcal{N}(0, \sigma^2) \tag{57}$$

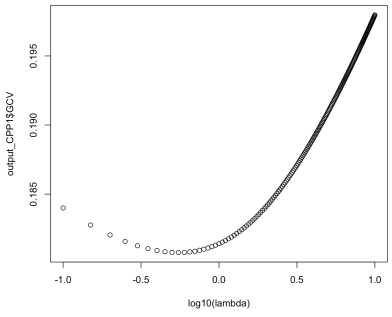
where $\sigma = range(f(\mathbf{p})) = 0.4205398$ and the location points $\mathbf{p}_i$ coincide with mesh nodes, where the mesh is the `C-shaped` mesh as defined in previous examples.
Notice that here we are using the already existing code in `fdaPDE` since we are in the gaussian setting. Here we just want to asses the correct computation of *exact* DoF, thus GCV. Results are shown in Figure (26)



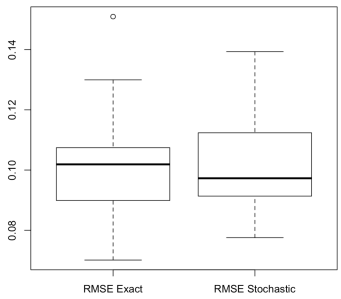
(a) True field



(b) Fitted field (optimal lambda, exact GCV)



(c) Optimal lambda concavity - Exact GCV



(d) RMSE: Exact (mean: 0.992) vs Stochastic (mean: 0.101)

Figure 26: Bug fix assessment. We can see the optimal lambda is chosen also for exact GCV: $GCV(\lambda)$ shown the typical concave behaviour around optimal value and the exact GCV result is now comparable with stochastic one. In Figure (a) and (b) high values yellow, small values red.

6 Conclusions

To sum up, the major contribution of our work has been to include a whole new class of models, GSR-PDE, within the `fdaPDE` library. The core algorithm has been inspired by the work done by M. Wilhelm in [14] and then extended to deal with more general PDE and dimension greater than 2.

Our work has start been developed on version `1.0` [7] of the package, created in 2019, and then included in the development version `1.1` available from the mid of may 2020. In this recent version also spatio-temporal model has been added, yet our work do not generalize to deal with temporal data, which thus constitute its natural continuation.

Special effort has been taken in this project to maintain the current structure of the `MixedFERegression` class. The main reason is that it constitutes the core class for a big fraction of the library functionalities, hence for many projects currently in development phase. Nevertheless our experience with FPIRLS drives us to discuss the main class defects, especially if seen as a piece of an iterative method, and suggest some possible improvement.
Specifically, the `MixedFERegression` class has been designed as a monolithic object performing all the task involved by the regression methods, from the construction of matrices dependent on the mesh and data to the computation of DoFs and GCV. This is actually suboptimal for iterative methods which tend in general to perform multiple calls to the *regression routine* with small changes in its initial setting. For FPIRLS indeed, the matrices related to the mesh are left unchanged during iterations and also the computation of DoF and GCV has to be delayed until convergence. We solved these issues in a practical way in order to produce non-structural changes in the core method, however if future developments should take into account iterative methods involving `MixedFERegression`, we suggest to re-design the class following a modular structure, in particular the new class should include:

- a set of methods handling the mesh dependencies like $\mathbf{\Psi}$, $\mathbf{A}$, $\mathbf{R_0}$, $\mathbf{R_1}$, and allow their possible independent update from class users;
- a set of methods handling the data dependencies like $\mathbf{H}$, $\mathbf{Q}$, the right hand term of (17), and allow their possible independent update;
- a public method performing the set-up and solution of system (17), i.e. the counterpart of current `apply`;
- an independent method computing DoF, like the one implemented in this project;
- a public method performing GCV with a general norm.

And in particular all the above methods should be available for a single λ call, such that the future developer could decide where to loop over λ for GCV minimum search.
A prototype of class declaration is shown in the appendix.

Future direction of this work is toward the addition of spatio-temporal models for non gaussian distributions and the inclusion of a wider set of distributions for the method.

References

[1] G. Ardenghi, A. Vicini, *Space-time regression models with differential regularization*, APSC report 2019.

[2] E. Arnone, B. Guerciotti, *Space-time separable models with differential regularization*, APSC report.

[3] L. Azzimonti, F. Nobile, L.M. Sangalli, P. Secchi, *Mixed finite elements for spatial regression with PDE penalization*, SIAM/ASA J Uncertain Quantification. 2014;2(1):305â335.

[4] L. Azzimonti, L.M. Sangalli, P. Secchi, M. Domanin, F. Nobile, *Blood flow velocity field estimation via spatial regression with PDE penalization*, J Amer Statist Assoc. 2015;110(511):1057â1071.

[5] C. Bambini, L. Giussani, *Regression models with Differential Regularization - Efficient computation of the Equivalent Degrees of Freedom*, APSC report 2017

[6] A. Buja, T. Hastie, R. Tibshirani, *Linear smoothers and additive models*, Ann Stat. 1989;17(2):453â510.

[7] A. Colli, L. Colombo, *fdaPDE 1.0*, APSC report 2019.

[8] P. Craven, G. Wahba, *Smoothing noisy data with spline functions*, Numer Math. 1978;31(4):377â403

[9] T. Hastie, R. Tibshirani *Generalized additive models* Statistical Science 1, 297â318, 1986.

[10] T. Hastie, R. Tibshirani, *Generalized Additive Models* Chapman Hall, 1990.

[11] L. Negri, *Smooth functional principal component analysis - Regularization for data distributed over 2D manifolds*, APSC report 2018

[12] R Core Team, *Writing R extensions - version 3.6.1*: `https://cran.r-project.org/doc/manuals/r-release/R-exts.html`

[13] L.M. Sangalli, J.O. Ramsay, T.O. Ramsay *Spatial spline regression models*, J R Statist Soc: Ser B (Statist Methodol). 2013;75(4):681703

[14] M. Wilhelm & L. M. Sangalli; *Generalized spatial regression with differential regularization*, Journal of Statistical Computation and Simulation. 86:13, 24972518, 2016.

[15] M. Wilhelm, L. Dedé. L. M. Sangalli, P. Wilhelm, *IGS: an IsoGeometric approach for smoothing on surfaces*, Comput. Methods Appl. Mech. Engrg., 302:7089, 2016.

[16] S. N. Wood, *Fast stable direct fitting and smoothness selection for Generalized Additive Models*, J. R. Stat. Soc. B70:495518.

[17] S.N. Wood, *Generalized additive models: an introduction with R*, CRC press.

Appendices

A typedefs

Within `fdaPDE` package the following typedefs are defined

```
typedef double Real;
typedef int UInt;
typedef Eigen::Matrix<Real,Eigen::Dynamic,Eigen::Dynamic> MatrixXr;
typedef Eigen::Matrix<UInt,Eigen::Dynamic,Eigen::Dynamic> MatrixXi;
typedef Eigen::Matrix<Real,Eigen::Dynamic,1> VectorXr;
typedef Eigen::Matrix<UInt,Eigen::Dynamic,1> VectorXi;
typedef Eigen::Matrix<VectorXr,Eigen::Dynamic,Eigen::Dynamic> MatrixXv;
typedef Eigen::SparseMatrix<Real> SpMat;
typedef Eigen::SparseVector<Real> SpVec;
typedef Eigen::Triplet<Real> coeff;
```

B MixedFERegression proposal

A proposal for a modular `MixedFERegressionBase` class
In particular new public interfaces are provided which allow to separate the main tasks performed by class objects and thus call them only when needed.

```
template<...>
class MixedFERegressionBase
{
  protected:
  const MeshHandler<ORDER, mydim, ndim> &mesh_;
  const std::vector<Real> mesh_time_;
  const UInt N_; //! Number of spatial basis functions.
  const UInt M_;
  const InputHandler& regressionData_;

// Mesh dependant objects
  SpMat matrixNoCov_; //! System matrix without
  Eigen::SparseLU<SpMat> matrixNoCovdec_; // Stores the factorization of
   matrixNoCov_
  SpMat R1_;   //! R1 matrix of the model
  SpMat R0_;   //! Mass matrix in space
  SpMat psi_;  //! Psi matrix of the model
  MatrixXr R_; //! R1 ^T * R0^-1 * R1
  VectorXr A_;    //! A_.asDiagonal() areal matrix
  bool isRcomputed_ = false;
  Eigen::SparseLU<SpMat> R0dec_; //! Stores the factorization of R0_

  MatrixXr barycenters_; //barycenter information
  VectorXi element_ids_; //elements id information

// Data dependant objects
```

```
26   bool isSpaceVarying = false; // used to distinguish whether to use the
      forcing term u in apply() or not
27   VectorXr rhs_ft_correction_;  //! right hand side correction for the
      forcing term:
28   VectorXr _rightHandSide;      //! A Eigen::VectorXr: Stores the system
      right hand side.
29   Eigen::PartialPivLU<MatrixXr> WTW_; // Stores the factorization of W^T *
       W
30   bool isWTWfactorized_ = false;
31
32 // temporal objects
33   VectorXr rhs_ic_correction_;  //! Initial condition correction (
      parabolic case)
34   SpMat Ptk_;   //! kron(Pt,IN) (separable version)
35   SpMat LR0k_;  //! kron(L,R0) (parabolic version)
36
37 // solve method objects
38   MatrixXr U_;  //! psi^T * W or psi^T * A * W padded with zeros, needed
      for Woodbury decomposition
39   MatrixXr V_;   //! W^T*psi, if pointwise data is U^T, needed for
      Woodbury decomposition
40   Eigen::PartialPivLU<MatrixXr> Gdec_;  // Stores factorization of G =  C
      + [V * matrixNoCov^-1 * U]
41
42 // outputs
43   MatrixXv _solution;           //! A Eigen::MatrixXv: Stores the system
      solution.
44   MatrixXr _dof;                //! A Eigen::MatrixXr storing the computed
       dofs
45   MatrixXr _GCV;   //! A Eigen::MatrixXr storing the computed GCV
46   UInt bestLambdaS_=0;  //!Stores the index of the best lambdaS according
      to GCV
47   UInt bestLambdaT_=0;  //!Stores the index of the best lambdaT according
      to GCV
48   Real _bestGCV=10e20;  //!Stores the value of the best GCV
49   MatrixXv _beta;   //! A Eigen::MatrixXv storing the computed beta
      coefficients
50
51   //! A member function computing the Psi matrix
52   void setPsi();
53   //! A method computing the no-covariates version of the system matrix
54   void buildMatrixNoCov(const SpMat& NWblock,  const SpMat& SWblock,
      const SpMat& SEblock);
55   //! A function that given a vector u, performs Q*u efficiently
56   MatrixXr LeftMultiplybyQ(const MatrixXr& u);
57   //! A function which adds Dirichlet boundary conditions before solving
      the system ( Remark: BC for areal data are not implemented!)
58   void addDirichletBC();
59   //! A method which takes care of missing values setting to 0 the
      corresponding rows of B_
60   void addNA();
61   //! A member function which builds the A vector containing the areas of
      the regions in case of areal data
62   void setA();
```

```
63  //! A member function returning the system right hand data
64  void getRightHandData(VectorXr& rightHandData);
65  //! A method which builds all the matrices needed for assembling
     matrixNoCov_
66  void buildSpaceTimeMatrices();
67  //! A method computing dofs in case of exact GCV, it is called by
     computeDegreesOfFreedom
68  void computeDegreesOfFreedomExact(UInt output_indexS, UInt output_indexT
     , Real lambdaS, Real lambdaT);
69  //! A method computing dofs in case of stochastic GCV, it is called by
     computeDegreesOfFreedom
70  void computeDegreesOfFreedomStochastic(UInt output_indexS, UInt
     output_indexT, Real lambdaS, Real lambdaT);
71
72    //! A function to factorize the system, using Woodbury decomposition
     when there are covariates
73  void system_factorize();
74  //! A function which solves the factorized system
75  template<typename Derived>
76  MatrixXr system_solve(const Eigen::MatrixBase<Derived>&);
77
78  public:
79  //!A Constructor.
80  MixedFERegressionBase(const MeshHandler<ORDER,mydim,ndim>& mesh, const
     InputHandler& regressionData);
81  MixedFERegressionBase(const MeshHandler<ORDER,mydim,ndim>& mesh, const
     std::vector<Real>& mesh_time, const InputHandler& regressionData);
82
83  //! A method building mesh depandant objects, callable each time the
     mesh object has been modyfied
84  template<typename A>
85  void buildMeshDependencies(EOExpr<A> oper);
86
87  //! A method building data depandant objects, callable each time the
     data object has been modyfied
88  void buildDataDependencies(const ForcingTerm & u);
89
90  //! A method which build and solve the final system for penalized
     regression
91  void apply(UInt output_indexS, UInt output_indexT);
92
93  //! A method computing beta estimates
94  VectorXr computeBetaEst(UInt output_indexS, UInt output_indexT);
95
96  //! A method computing function estimates fn
97  VectorXr computeFunctionEst(UInt output_indexS, UInt output_indexT);
98
99  //! A method computing the dofs
100 void computeDegreesOfFreedom(UInt output_indexS, UInt output_indexT,
     Real lambdaS, Real lambdaT);
101
102 //! A method computing GCV
103 void computeGCV(UInt output_indexS, UInt output_indexT);
104
```

```
105   //! A method computing GCV - overloaded for handling more general
       distance functions
106   void computeGCV(UInt output_indexS, UInt output_indexT, Real dist(Real,
       Real));
107
108     //! A wrapper for apply(lambdaS,lambdaT) which perform the for loop
      over lambda and thus the substitute of current apply()
109     void apply();
110
111   // Get methods
112   ...
113
114 };
```