iprior: An R Package for Regression Modelling using I-priors

Haziq Jamil
London School of Economics

Wicher Bergsma
London School of Economics

Abstract

This is an overview of the R package **iprior**, which implements a unified methodology for fitting parametric and nonparametric regression models, including additive models, multilevel models, and models with one or more functional covariates. Based on the principle of maximum entropy, an I-prior is an objective Gaussian process prior for the regression function with covariance kernel equal to its Fisher information. The regression function is estimated by its posterior mean under the I-prior, and hyperparameters are estimated via maximum marginal likelihood. Estimation of I-prior models is simple and inference straightforward, while small and large sample predictive performances are comparative, and often better, to similar leading state-of-the-art models. We illustrate the use of the **iprior** package by analysing a simulated toy data set as well as three real-data examples, in particular, a multilevel data set, a longitudinal data set, and a dataset involving a functional covariate.

Keywords: Gaussian, process, regression, objective, prior, empirical, Bayes, RKHS, kernel, EM, algorithm, Nyström, random, effects, multilevel, longitudinal, functional, I-prior, R.

Updated: 18 December 2017

1. Introduction

For subject $i \in \{1, \ldots, n\}$, assume a real-valued response y_i has been observed, as well as a row vector of p covariates $x_i = (x_{i1}, \ldots, x_{ip})$, where each x_{ik} belongs to some set $\mathcal{X}_k$, $k = 1, \ldots, p$. To describe the dependence of the y_i on the x_i, we consider the regression model

$$\begin{aligned} y_i &= \alpha + f(x_i) + \epsilon_i \\ \epsilon_i &\overset{\text{iid}}{\sim} \text{N}(0, \psi^{-1}) \end{aligned} \tag{1}$$

where f is some regression function to be estimated, and α is an intercept. When f can be parameterised linearly as $f(x_i) = x_i^\top \beta$, $\beta \in \mathbb{R}^p$, we then have the ordinary linear regression—a staple problem in statistics and other quantitative fields. We might also have that the data is separated naturally into groups or levels by design, for example, data from stratified sampling, students within schools, or longitudinal measurements over time. In such a case, we might want to consider a regression function with additive components

$$f(x_{ij}, j) = f_1(x_{ij}) + f_2(j) + f_{12}(x_{ij}, j)$$

where x_{ij} denotes the ith observation in group $j \in \{1, \ldots, m\}$. Again, assuming a linear parameterisation, this is recognisable as the multilevel or random-effects linear model, with

f_2 representing the varying intercept via $f_2(j) = \beta_{0j}$, f_{12} representing the varying slopes via $f_{12}(x_{ij}, j) = x_{ij}^\top \beta_{1j}$, and f_1 representing the linear component as above.

Moving on from linear models, smoothing models may be of interest as well. A myriad of models exist for this type of problem, with most classed as nonparametric regression, and the more popular ones are LOcal regrESSion (LOESS), kernel regression, and smoothing splines. Semiparametric regression models combines the linear component with a non-parameteric component.

Further, the regression problem is made interesting when the set of covariates $\mathcal{X}$ is functional—in which case the linear regression model aims to estimate coefficient functions $\beta : \mathcal{T} \to \mathbb{R}$ from the model

$$y_i = \int_{\mathcal{T}} x_i(t)\beta(t)\,\mathrm{d}t + \epsilon_i.$$

Nonparametric and semiparametric regression with functional covariates have also been widely explored.

On the software side, there doesn't seem to be a shortage of packages in R (R Core Team 2017) to fit the models mentioned above. The `lm()` function provides simple and multiple linear regression in base R. For multilevel models, **lme4** (Bates, Mächler, Bolker, and Walker 2015) is widely used. These two are likelihood based methods, but Bayesian alternatives are available as well in **rstanarm** Stan Development Team (2016b) and **brms** (Bürkner 2017). For smoothing models, base R provides the functions `smooth.spline()` for modelling with smoothing splines, and `ksmooth()` for Nadaraya-Watson kernel regression. The **mgcv** package (Wood 2017) is an extensive package that is able to fit (generalised) additive models. Finally, the **fda** package (Ramsay, Wickham, Graves, and Hooker 2017) fits functional regression models in R.

The **iprior** package provides a platform in R to estimate a wide-range of regression models, including the ones described above, using what we call the I-prior methodology. As such, it can be seen as a unifying methodology for various regression models. Estimation and inference is simple and straightforward using maximum likelihood, and thus the package provides the end-user with the tools necessary to analyse and interpret various types of regression models. Prediction is also possible, with small and large sample performance comparative to, though often better than, other methodologies such as the closely related Gaussian process regression.

1.1. The I-prior regression model

For the regression model stated in (1), we assume that the function f lies in a reproducing kernel Hilbert space (RKHS) of functions $\mathcal{F}$ with reproducing kernel $h : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Often, the reproducing kernel (or simply kernel, for short) is indexed by one or more parameters which we shall denote as η. Correspondingly, the kernel is rightfully denoted as h_η to indicate the dependence of the parameters on the kernels, though where this is seemingly obvious, might be omitted.

The definition of an RKHS entails that any function in $\mathcal{F}$ can be approximated arbitrarily well by functions of the form

$$f(x) = \sum_{i=1}^{n} h(x, x_i)w_i \tag{2}$$

where $w_1, \ldots, w_n$ are real-valued[1]. We define the *I-prior* for our regression function f in (1) as the distribution of a random function of the form (2) when each of the w_i are independently distributed as $\mathrm{N}(0, \psi)$, with ψ being the model error precision. As a result, we may view the I-prior for f as having the Gaussian process distribution

$$\mathbf{f} = \big(f(x_1), \ldots, f(x_n)\big)^\top \sim \mathrm{N}_n(\mathbf{0}, \psi\mathbf{H}_\eta^2) \tag{3}$$

with $\mathbf{H}_\eta$ an $n \times n$ matrix with (i, j) entries equal to $h_\eta(x_i, x_j)$, and $\mathbf{0}$ a length n vector of zeroes. The covariance kernel of this prior distribution is related to the Fisher information for f (Bergsma 2017), and hence the name I-prior—the 'I' stands for information. The prior mean of zero for the regression function is typically reasonable, and this is what the **iprior** package implements.

As with Gaussian process regression (GPR), the function f is estimated by its posterior mean. In the normal model, the posterior distribution for the regression function conditional on the responses $\mathbf{y} = (y_1, \ldots, y_n)$,

$$p(\mathbf{f}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f})}{\int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})\,\mathrm{d}\mathbf{f}}, \tag{4}$$

can easily be found, and it is in fact normally distributed. The posterior mean for f evaluated at a point $x \in \mathcal{X}$ is given by

$$\mathsf{E}\big[f(x)|\mathbf{y}\big] = \mathbf{h}_\eta^\top(x) \cdot \overbrace{\psi\mathbf{H}_\eta\big(\psi\mathbf{H}_\eta^2 + \psi^{-1}\mathbf{I}_n\big)^{-1}\mathbf{y}}^{\tilde{\mathbf{w}}} \tag{5}$$

where we have defined $\mathbf{h}_\eta^\top(x)$ to be the vector of length n with entries $h_\eta(x, x_i)$ for $i = 1, \ldots, n$. Incidentally, the elements of the n-vector $\tilde{\mathbf{w}}$ defined in (5) are the posterior means of the random variables w_i in the formulation (2). The posterior variance of f is given by

$$\mathsf{VAR}\big[f(x)|\mathbf{y}\big] = \mathbf{h}_\eta^\top(x)\big(\psi\mathbf{H}_\eta^2 + \psi^{-1}\mathbf{I}_n\big)^{-1}\mathbf{h}_\eta^\top(x). \tag{6}$$

These are of course well-known results in Gaussian process literature—see, for example, Rasmussen and Williams (2006) for details.

1.2. Estimation

The kernel parameter η and the error precision ψ (which we collectively refer to as the model hyperparameters of the covariance kernel θ) can be estimated in several ways. One of these is direct optimisation of the marginal log-likelihood—the most common method in the Gaussian process literature.

$$\begin{aligned}\log L(\theta) &= \log \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f})\,\mathrm{d}\mathbf{f} \\ &= -\frac{n}{2}\log 2\pi - \frac{1}{2}\log|\mathbf{\Sigma}_\theta| - \frac{1}{2}\mathbf{y}^\top\mathbf{\Sigma}_\theta^{-1}\mathbf{y}\end{aligned}$$

[1] That is to say, $\mathcal{F}$ is spanned by the functions $h(\cdot, x)$. More precisely, $\mathcal{F}$ is the completion of the space $\mathcal{G} = \mathrm{span}\{h(\cdot, x)|x \in \mathcal{X}\}$ endowed with the squared norm $\|f\|_{\mathcal{G}}^2 = \sum_{i=1}^n \sum_{i=1}^n w_i w_j h(x_i, x_j)$ for f of the form (2). See, for example, Berlinet and Thomas-Agnan (2011) for details.

where $\boldsymbol{\Sigma}_\theta = \psi\mathbf{H}_\eta^2 + \psi^{-1}\mathbf{I}_n$. This is typically done using conjugate gradients with a Cholesky decomposition on the covariance kernel to maintain stability, but the **iprior** package opts for an eigendecomposition of the kernel matrix (Gram matrix) $\mathbf{H}_\eta = \mathbf{V} \cdot \text{diag}(u_1, \ldots, u_n) \cdot \mathbf{V}^\top$ instead. Since $\mathbf{H}_\eta$ is a symmetrix matrix, we have that $\mathbf{V}\mathbf{V}^\top = \mathbf{I}_n$, and thus

$$\boldsymbol{\Sigma}_\theta = \mathbf{V} \cdot \text{diag}(\psi u_1^2 + \psi^{-1}, \ldots, \psi u_n^2 + \psi^{-1}) \cdot \mathbf{V}^\top$$

for which the inverse and log-determinant is easily obtainable. This method is relatively robust to numerical instabilities and is better at ensuring positive definiteness of the covariance kernel. The eigendecomposition is performed using the **Eigen** C++ template library and linked to **iprior** using **Rcpp** (Eddelbuettel and Francois 2011). The hyperparameters are transformed by the **iprior** package so that an unrestricted optimisation using the quasi-Newton L-BFGS algorithm provided by `optim()` in R. Note that minimisation is done on the deviance scale, i.e., minus twice the log-likelihood. The direct optimisation method can be prone to local optima, in which case repeating the optimisation at different starting points and choosing the one which yields the highest likelihood is one way around this.

Alternatively, the expectation-maximisation (EM) algorithm may be used to estimate the hyperparameters, in which case the I-prior formulation in (2) is convenient. Substituting this into (1) we get something that resembles a random effects model. By treating the w_i as "missing", the tth iteration of the E-step entails computing

$$Q(\theta) = \mathsf{E}\left[\log p(\mathbf{y}, \mathbf{w}|\theta)\middle|\mathbf{y}, \theta^{(t)}\right]. \tag{7}$$

As a consequence of the properties of the normal distribution, the required joint and posterior distributions $p(\mathbf{y}, \mathbf{w})$ and $p(\mathbf{w}|\mathbf{y})$ are easily obtained. The M-step then maximises the Q function above, which boils down to solving the first order conditions

$$\frac{\partial Q}{\partial \eta} = -\frac{1}{2}\,\text{tr}\left(\frac{\partial \boldsymbol{\Sigma}_\theta}{\partial \eta}\tilde{\mathbf{W}}^{(t)}\right) + \psi \cdot \mathbf{y}^\top \frac{\partial \mathbf{H}_\eta}{\partial \eta}\tilde{\mathbf{w}}^{(t)} \tag{8}$$

$$\frac{\partial Q}{\partial \psi} = -\frac{1}{2}\mathbf{y}^\top\mathbf{y} - \text{tr}\left(\frac{\partial \boldsymbol{\Sigma}_\theta}{\partial \psi}\tilde{\mathbf{W}}^{(t)}\right) + \mathbf{y}^\top\mathbf{H}_\eta\tilde{\mathbf{w}}^{(t)} \tag{9}$$

equated to zero. Here, $\tilde{\mathbf{w}}$ and $\tilde{\mathbf{W}}$ are the first and second posterior moments of $\mathbf{w}$. The solution to (9) can be found in closed-form, but not necessarily for (8). In cases where closed-form solutions exist, then it is just a matter of iterating the update equations until a suitable convergence criterion is met (e.g. no more sizeable increase in successive log-likelihood values). In cases where closed-form solutions do not exist for θ, the Q function is again optimised with respect to θ using the L-BFGS algorithm.

The EM algorithm is more stable than direct maximization, and is especially suitable if there are many scale parameters. However, it is typically slow to converge. The **iprior** package provides a method to automatically switch to the direct optimisation method after running several EM iterations. This then combines the stability of the EM with the speed of direct optimisation.

For completeness, it should be mentioned that a full Bayesian treatment of the model is possible, with additional priors on the hyperparameters set. Markov chain Monte Carlo (MCMC) methods can then be employed to sample from the posteriors of the hyperparameters, with point estimates obtained using the posterior mean or mode, for instance. Additionally, the

posterior distribution encapsulates the uncertainty about the parameter, for which inference can be made. Posterior sampling can be done using Gibbs-based methods in **WinBUGS** (Lunn, Thomas, Best, and Spiegelhalter 2000) or **JAGS** (Plummer 2003), and both have interfaces to R via **R2WinBUGS** (Sturtz, Ligges, and Gelman 2005) and **runjags** (Denwood 2016) respectively. Hamiltonian Monte Carlo (HMC) sampling is also a possibility, and the Stan project (Carpenter, Gelman, Hoffman, Lee, Goodrich, Betancourt, Brubaker, Guo, Li, and Riddell 2017) together with the package **rstan** (Stan Development Team 2016a) makes this possible in R. All of these MCMC packages require the user to code the model individually, and we are not aware of the existence of MCMC-based packages which are able to estimate GPR models. This makes it inconvenient for GPR and I-prior models, because in addition to the model itself, the kernel functions need to be coded as well and ensuring computational efficiency would be a difficult task. Note that this full Bayesian method is not implemented in **iprior**, but described here for completeness.

Finally, a note on the intercept. Given the regression model (1) subject to an I-prior (3), the marginal likelihood of the intercept α (after integrating out the I-prior) can be maximised with respect to α, which yields the sample mean for y as the ML estimate for intercept.

1.3. Computational considerations

Computational complexity for estimating I-prior models (and in fact, for GPR in general) is dominated by the inversion of the $n \times n$ matrix $\boldsymbol{\Sigma}_\theta = \psi \mathbf{H}_\eta^2 + \psi^{-1}\mathbf{I}_n$, which scales as $O(n^3)$ in time. As mentioned earlier, the **iprior** package inverts this by way of the eigendecomposition of $\mathbf{H}_\eta$, but this operation is also $O(n^3)$. For the direct optimisation method, this matrix inversion is called when computing the log-likelihood, and thus must be computed at each Newton step. For the EM algorithm, this matrix inversion appears when calculating $\tilde{\mathbf{w}}$, the posterior mean of the I-prior random effects. Furthermore, storage requirements for I-priors models are similar to that of GPR models, which is $O(n^2)$.

The machine learning literature is rich in ways to resolve this issue, as summarised by Quiñonero-Candela and Rasmussen (2005). One such method is to use low-rank matrix approximations. Let $\mathbf{Q}$ be a matrix with rank $q < n$, and that $\mathbf{Q}\mathbf{Q}^\top$ can be used to approximate the kernel matrix $\mathbf{H}_\eta$. Then

$$(\psi \mathbf{H}_\eta^2 + \psi^{-1}\mathbf{I}_n)^{-1} \approx \psi \left[\mathbf{I}_n - \mathbf{Q} \left(\left(\psi^2 \mathbf{Q}^\top \mathbf{Q} \right)^{-1} + \mathbf{Q}^\top \mathbf{Q} \right)^{-1} \mathbf{Q}^\top \right],$$

obtained via the Woodbury matrix identity, is a potentially much cheaper operation which scales $O(nq^2)$—$O(q^3)$ to do the inversion, and $O(nq)$ to do the multiplication (because typically the inverse is premultiplied to a vector). When the kernel matrix itself is sufficiently low ranked (for instance, when using the linear kernel for a low-dimensional covariate) then the above method is exact. However, other interesting kernels such as the fractional Brownian motion (fBm) kernel or the squared exponential kernel results in kernel matrices which are full rank.

Another method of approximating the kernel matrix, and the method implemented by our package, is the Nyström method (Williams and Seeger 2001). The theory has its roots in approximating eigenfunctions, but this has since been adopted to speed up kernel machines. The main idea is to obtain an (approximation to the true) eigendecomposition of $\mathbf{H}_\eta$ based on a small subset $m \ll n$ of the data points. Reorder the rows and columns and partition the

kernel matrix as

$$\mathbf{H}_\eta = \begin{pmatrix} \mathbf{A}_{m\times m} & \mathbf{B}_{m\times(n-m)} \\ \mathbf{B}^\top_{m\times(n-m)} & \mathbf{C}_{(n-m)\times(n-m)} \end{pmatrix}.$$

The Nyström method provides an approximation to the lower right block $\mathbf{C}$ by manipulating the eigenvectors and eigenvalues of $\mathbf{A}$, an $m \times m$ matrix, together with the matrix $\mathbf{B}$ to give

$$\mathbf{H}_\eta \approx \begin{pmatrix} \mathbf{V}_m \\ \mathbf{B}^\top \mathbf{V}_m \mathbf{U}_m^{-1} \end{pmatrix} \mathbf{U}_m \begin{pmatrix} \mathbf{V}_m^\top & \mathbf{U}_m^{-1} \mathbf{V}_m^\top \mathbf{B} \end{pmatrix}$$

where $\mathbf{U}_m$ is the diagonal matrix containing the m eigenvalues of $\mathbf{A}$, and $\mathbf{V}_m$ is the corresponding matrix of eigenvectors. An orthogonal version of this approximation is of interest, which has been studied by Fowlkes, Belongie, and Malik (2001), which allows us to easily calculate the inverse of $\mathbf{\Sigma}_\theta$. Estimating I-prior models using the Nyström method takes $O(nm^2)$ times and $O(nm)$ storage.

1.4. Comparison to other software packages

It is possible to reformulate the I-prior model as a standard GPR model, by using "squared" kernels and a suitable reparameterisation of the hyperparameters (for details see Appendix A). With this in mind, there are a number of excellent GPR software packages available which may potentially be used to fit I-prior models. In this short review, we specifically describe R packages, and briefly outline the limitations for I-prior modelling.

The package **kernlab** (Karatzoglou, Smola, Hornik, and Zeileis 2004) provides a comprehensive toolkit for not just GPR, but also other kernel-based machine learning models. The package provides what is termed "dot product primitives (kernels)", i.e., the functions to calculate kernels and kernel matrices are exposed to the end-user. Furthermore, the package allows user-defined kernel functions to extend their S4 methods which then means I-prior modelling is possible. However, kernel hyperparameters must be fixed, and therefore it is not possible to estimate the hyperparameter values from the data.

The package **gptk** (Kalaitzis, Honkela, Gao, and Lawrence 2014) on the other hand, originally implemented in MATLAB, is able to estimate hyperparameters by log-likelihood maximisation. Kernel support is minimal, with the main kernel used being the SE kernel. The authors of the package have also developed another GPR package called **gprege** (Kalaitzis and Lawrence 2011), but it seems it is geared towards the specific usage in time-series genetic expression analysis.

The package **GPfit** (MacDonald, Ranjan, and Chipman 2015) focuses effort on developing a clustering based, gradient type, optimisation algorithm to estimate GPR models using the squared exponential kernel. There is also no option to change or modify this kernel nor to supply a user-defined kernel.

Finally, **GPFDA** (Shi and Cheng 2014) is a specialised package which performs GPR with functional covariates, though unidimensional covariates are also supported. Kernel support is limited, with only the linear and SE kernel built-in as of the time of writing, though the package does estimate the hyperparameters and the error precision via maximum likelihood (using conjugate gradient methods).

A common theme among all these packages, besides using the SE kernel as standard, is the lack of a feature to estimate kernel parameters. Though when this feature is available, it is

not possible to supply a user-defined kernel in which to perform I-prior modelling. Apart from **kernlab**, which only offers the functionality to use fixed hyperparameter values, there does not seem to be a suitable package to estimate I-prior models.

2. The iprior package

There are three main functions in the **iprior** package. The first are the various `kern_*()` functions to create kernel matrices. The second is the `kernL()` function to "load the kernels" and prepare an I-prior model. The third is the `iprior()` function itself, which either takes a prepared kernel-loaded object from `kernL()`, or takes in a model directive straight from the user, and proceeds to fit an I-prior model. A schematic of the package is shown in the figure below.

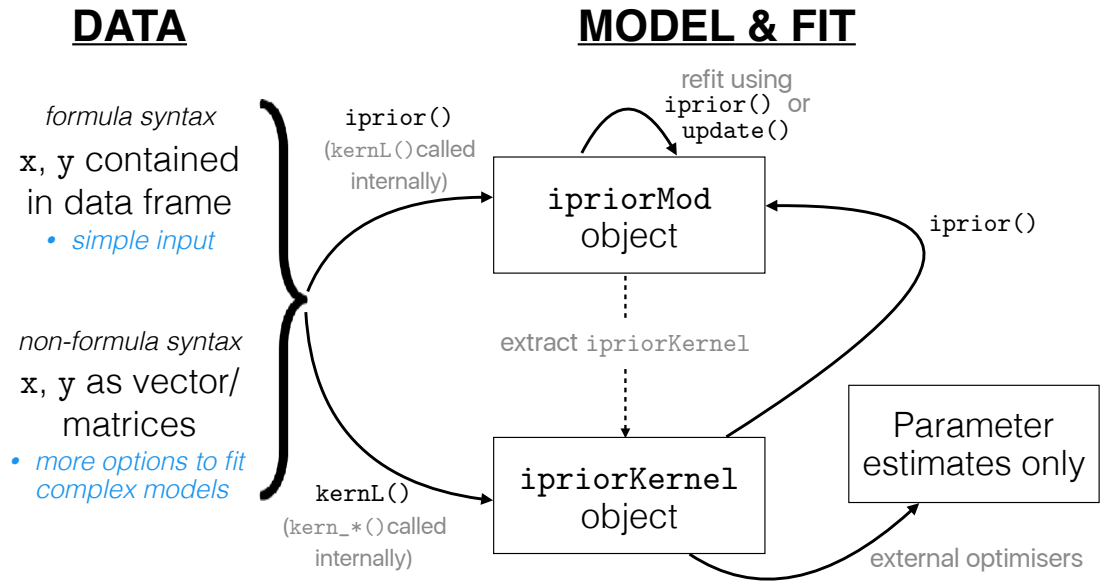

Figure 1: A schematic of the package. The main `iprior()` function can be called directly, which in turn calls `kernL()` internally and that calls the particular `kern_*()` functions. Alternatively, these can be called individually.

2.1. Kernels

The building blocks of the I-prior models are kernel functions: Symmetric, positive-definite functions which maps pairs of inputs from $\mathcal{X}$ to a real number, i.e., $h : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. The usefulness of kernels are particularly well-known in the machine learning literature, with many methods taking advantage of what is known as the "kernel trick" (Bishop 2006, Section 6). For our purposes, kernels give rise to reproducing kernel Hilbert spaces (RKHS), which provides us the mathematical structure necessary to perform I-prior modelling. The choice of kernel determines the space of functions that the I-prior functions resides in, and this package supports five types of RKHSs, which are explained below. With the exception of the Pearson kernel, the set $\mathcal{X}$ for the kernels are assumed to be $\mathbb{R}^d$, for some $d \in \mathbb{N}$.

In what follows, each of the kernel functions have an associated scale parameter λ. The scale of the RKHS over the set of covariates may be arbitrary, so scale parameters, typically estimated from the data, are introduced to resolve this. Setting scale parameter values are optional when calling the kernel functions in **iprior**, with default values of one. Some of the kernels are also defined by additional parameters.

The canonical linear kernel

The function `kern_linear()` (or the alternatively named `kern_canonical()`) implements the kernel given by

$$h_\lambda(x, x') = \lambda \cdot \langle x, x' \rangle_{\mathcal{X}}.$$

This gives rise to an RKHS of linear functions.

The fractional Brownian motion (fBm) kernel

The function `kern_fbm()` implements the fractional Brownian motion kernel with Hurst coefficient $\gamma \in (0, 1)$, as defined by

$$h_{\lambda,\gamma}(x, x') = -\frac{\lambda}{2}\left(\|x - x'\|_{\mathcal{X}}^{2\gamma} - \|x\|_{\mathcal{X}}^{2\gamma} - \|x'\|_{\mathcal{X}}^{2\gamma}\right).$$

This gives rise to functions suitable for smoothing models. The value of the Hurst coefficient acts as a smoothing parameter. The default value $\gamma = 0.5$ is used in the package.

The squared exponential (SE) kernel

The function `kern_se()` implements the de-facto kernel used in GPR, as defined by

$$h_{\lambda,l}(x, x') = \lambda \cdot \exp\left(-\frac{\|x - x'\|_{\mathcal{X}}^2}{2l^2}\right).$$

The length scale parameter $l > 0$ determines how much the smooth exponential functions wiggles. Parameterised differently, this kernel is also known as the Gaussian kernel or the radial basis function (RBF) kernel. The default length scale is set to one.

The d-degree polynomial kernel

The function `kern_poly()` implements polynomial kernel with degree d and offset $c \geq 0$, as given by

$$h_{\lambda,c,d}(x, x') = \left(\lambda \cdot \langle x, x' \rangle_{\mathcal{X}} + c\right)^d.$$

This allows modelling with functions with effect similar to that achieved by polynomial regression. In other words, squared, cubic, or higher order terms may be added to the regression function simply by choosing the correct degree d. The offset parameter may be estimated or left at the default of zero.

The Pearson kernel

The so-called Pearson RKHS contains functions that map a countably finite set $\mathcal{X}$ to the reals, and so the kernel is used for categorical covariates. Let P be a probability distribution

over $\mathcal{X}$. The function `kern_pearson()` implements the kernel defined by

$$h_\lambda(x, x') = \lambda \cdot \left(\frac{\delta_{xx'}}{\mathsf{P}(X = x)} - 1 \right),$$

where δ is the Kronecker delta. The package uses the empirical distribution in lieu of the true distribution P. The Pearson RKHS is so named due to its relation with the Pearson chi-square statistic as seen from the *Hilbert-Schmidt independence criterion* (HSIC) (Gretton, Bousquet, Smola, and Schölkopf 2005).

Examples

The `kern_*()` functions take in vectors or matrices `x`, and another optional vector or matrix `y`. If `y` is not supplied, then `y` is taken to be `x`. The function returns a matrix with entries `[i, j]` entries equal to h(`x[i, ]`, `y[j, ]`).

```
R> # The linear kernel
R> x <- rnorm(5)
R> kern_linear(x, 1:5)
R> # The fBm kernel with Hurst = 0.5
R> y <- rnorm(3)
R> kern_fbm(x, y, gamma = 0.5)
R> # The SE kernel with length scale = 1
R> x <- matrix(rnorm(5 * 3), nrow = 5, ncol = 3)
R> kern_se(x, l = 1)
R> # The polynomial kernel with degree = 2 and offset = 0
R> y <- matrix(rnorm(3 * 3), nrow = 3, ncol = 3)
R> kern_poly(x, y, d = 2, c = 0)
R> # The Pearson kernel
R> x <- factor(1:5)
R> kern_pearson(x)
```

The Sobolev-Hilbert inner product for functional covariates

Suppose that we have functional covariates x in the real domain, and that $\mathcal{X}$ is a set of differentiable functions. If so, it is reasonable to assume that $\mathcal{X}$ is a Sobolev-Hilbert space with inner product

$$\langle x, x' \rangle_{\mathcal{X}} = \int \dot{x}(t)\dot{x}'(t)\,\mathrm{d}t,$$

so that we may apply the linear, fBm or any other kernels which make use of inner products by making use of the polarisation identity. Furthermore, let $z \in \mathbb{R}^T$ be the discretised realisation of the function $x \in \mathcal{X}$ at regular intervals $t = 1, \ldots, T$. Then

$$\langle x, x' \rangle_{\mathcal{X}} \approx \sum_{t=1}^{T-1} (z_{t+1} - z_t)(z'_{t+1} - z'_t).$$

For discretised observations at non-regular intervals then a more general formula to the above one might be used.

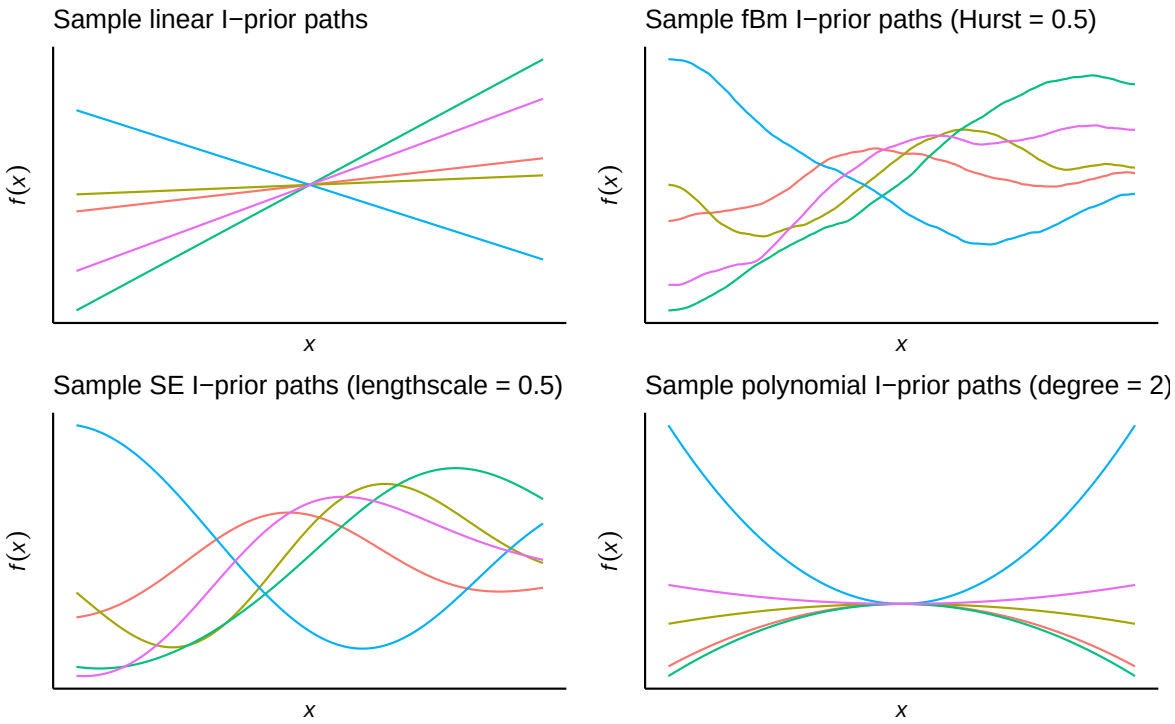

Figure 2: Samples of function paths following an I-prior under various kernels. These were generated according to (2) with $\psi = 1$.

Centred kernels

As a remark, the package implements *centred versions* of the above kernels when estimating I-prior models, which resolves the possibly arbitrary origin of the RKHS over the set of covariates. Generally, the centred kernel is defined as

$$h_{\text{cen}}(x, x') = h(x, x') - \frac{1}{n}\sum_{i=1}^{n} h(x, x_i) - \frac{1}{n}\sum_{i=1}^{n} h(x_i, x') + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} h(x_i, x_j),$$

such that the sum over columns in the kernel matrix is zero[2]. The use of centred kernels in the `iprior()` function is non-negotiable. However, when calling the `kern_*()` functions independently, an additional option `centre = FALSE` may be set to retrieve the non-centred versions of the respective kernels.

2.2. The kernel loader

For the remainder of Section 2, we shall be looking at the `Orange` data set available in base R, which contains $n = 35$ growth records of the circumference of the trunks of seven orange trees. The data set also contains the age of the trees (in days) at the time of measurement. The tree labels are treated as nominal variables, and the rest of the data set as real, continuous measurements. For simplicity, we shorten the variable names

[2]The polynomial kernel is not centred this way, but the inner product within it is. The intention of using centred kernels is to achieve centering of the feature space embedding of the data.

```
R> names(Orange) <- c("tree", "age", "circ")
R> head(Orange)

##   tree  age circ
## 1    1  118   30
## 2    1  484   58
## 3    1  664   87
## 4    1 1004  115
## 5    1 1231  120
## 6    1 1372  142
```

Basic syntax

The `kernL()` function readies an I-prior model for estimation according to the user's specification of the model. It determines the hyperparameters to be estimated (or fixed), performs the necessary kernel matrix calculations, and outputs an object of class `ipriorKernel`. This can then be passed to the `iprior()` function for estimation, which is explained in Section 2.3. An I-prior model may be specified using formula or non-formula syntax, and the most basic syntax is as follows:

```
R> mod <- kernL(circ ~ age + tree, data = Orange)    # formula syntax
R> with(Orange, mod <<- kernL(y = circ, age, tree))  # non-formula syntax
```

This would fit the following model:

$$\texttt{circ} = \alpha + f(\texttt{age}, \texttt{tree}) + \epsilon$$
$$\epsilon \sim \mathrm{N}(0, \psi^{-1})$$

with $f \in \mathcal{F}$ (an RKHS), and an I-prior on the regression function f. In the I-prior methodology, we assume an additive decomposition of the regression function into constituent functions dictated by the desired effect of the covariates. For instance, we could assume that

$$f(\texttt{age}, \texttt{tree}) = f_1(\texttt{age}) + f_2(\texttt{tree})$$

where f_1 lies in the linear RKHS $\mathcal{F}_1$ and f_2 lies in the Pearson RKHS $\mathcal{F}_2$, so that $\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2$. This would have the effect of regressing separate "straight line" functions with similar slopes on the covariate `age` for each `tree`—in other words, it is a varying-intercept model. Omitting the `tree` variable in the syntax above would fit a linear regression model of `circ` on `age`.

Interactions

A varying-slope effect can be achieved by assuming

$$f(\texttt{age}, \texttt{tree}) = f_1(\texttt{age}) + f_2(\texttt{tree}) + f_{12}(\texttt{age}, \texttt{tree}).$$

Here, f_{12} is assumed to lie in the so-called tensor product RKHS $\mathcal{F}_1 \otimes \mathcal{F}_2$ with kernel $h_{12} = h_1 h_2$. To fit this model, the additional option `interactions` should be called when using non-formula syntax. In the formula syntax, then use the regular expression for interactions.

```
R> # formula syntax
R> mod <- kernL(circ ~ age + tree + age:tree, data = Orange)
R> # non-formula syntax
R> with(Orange, {
+    mod <<- kernL(y = circ, age, tree, interactions = "1:2")
+  })
```

The syntax for specifying interactions in the non-formula syntax is `"a:b"` to indicate that the variable in position `a` interacts with the variable in position `b`. As seen above, this is automatically dealt with when using formula. The resulting output from `print()` contains information regarding the I-prior model prescribed.

```
R> print(mod)

## Sample size: 35
## No. of covariates: 2
## Object size: 140.6 kB
##
## Kernel matrices:
##  1 linear [1:35, 1:35] 646646 352329 207584 -65825 -248365 ...
##  2 pearson [1:35, 1:35] 4 4 4 4 4 4 4 -1 -1 -1 ...
##  3 linear x pearson [1:35, 1:35] 2586583 1409318 830335 -263299 -993461 ...
##
## Hyperparameters to estimate:
## lambda[1], lambda[2], psi
##
## Estimation methods available:
## direct, em, mixed, fixed
```

As the output tells us, the I-prior model "loaded" has $n = 144$ samples and one covariate using the linear RKHS. The hyperparameters to estimate are the scale parameters `lambda[1]` and `lambda[2]`, and the error precision `psi`. Note that since the kernel for the interaction effect is simply the product of the two kernels, there are no additional hyperparameters as a result. The estimation methods, selectable during the `iprior()` fit, are the direct optimisation method, the EM algorithm, and the mixed (EM plus direct) method. It is also possible just to obtain the posterior regression estimate based on fixed hyperparameters. Setting user-defined hyperparameter values for the scale parameters, error precision, and other kernel parameters are explained next.

Specifying kernels and setting hyperparameters

If instead we assumed that $\mathcal{F}_1$ is the fBm RKHS, we can achieve a smooth effect of the covariate `age` on the response variable. This is achieved by specifying the `kernel` option:

```
R> mod <- kernL(circ ~ ., data = Orange, kernel = "fbm")
R> mod$kernels
```

```
##      tree       age
## "pearson" "fbm,0.5"
```

Notice that factor type objects are automatically treated with the Pearson kernel, and this is not able to be overridden unless the data is preprocessed beforehand and converted to a `numeric` class object. If there are multiple variables in the model, it is possible to specify them individually, as follows:

```
R> Orange$tree <- as.numeric(Orange$tree)
R> mod <- kernL(circ ~ age + tree, data = Orange,
+               kernel = c("se,0.5", "poly3,1"))
R> get_kernels(mod)

##       age      tree
##  "se,0.5" "poly3,1"
```

In this example, the variable `tree` has been converted to a `numeric` class, and applied a polynomial kernel of degree three and offset equal to one. Meanwhile, the kernel for `age` has been set to a SE kernel with lengthscale 0.5. Note the use of the comma (`,`) to specify the hyperparameter for the kernel. The syntax is `"<kernel name>,<value>"`, where `<kernel name>` can be one of `fbm`, `se` or `poly` (the `linear` and `pearson` kernels do not have hyperparameters associated with them, except the scale). Omission of the `,<value>` is allowed, in which case the default values for the kernels are set. To set the degree d of the polynomial kernel, use `poly` or `poly2` for $d = 2$, `poly3` for $d = 3$, and so on.

It is also possible to set the values of the scale parameters and error precision by including the arguments `lambda = <value>` and/or `psi = <value>`, and this is especially relevant if the user would like these values to be treated as fixed. Note that setting values for any of the hyperparameters above do not indicate that they should not be estimated; this is explained in the next subsection.

Selecting the hyperparameters to estimate

With the current five kernels supported by the package, users may select which of the hyperparameters should be estimated in the I-prior model. This is specified by calling the logical options listed in the Table 1. Here's an example:

```
R> (kernL(circ ~ age + tree, Orange, kernel = "fbm", est.hurst = TRUE))

## Sample size: 35
## No. of covariates: 2
## Object size: 37.7 kB
##
## Kernel matrices:
##  1 fbm,0.5 [1:35, 1:35] 529 216 87 -107 -205 ...
##  2 pearson [1:35, 1:35] 4 4 4 4 4 4 4 -1 -1 -1 ...
##
```

Option	Description	Default
`est.lambda`	Estimate scale parameters?	`TRUE`
`est.hurst`	Estimate fBm Hurst coefficients?	`FALSE`
`est.lengthscale`	Estimate SE lengthscales?	`FALSE`
`est.offset`	Estimate polynomial offsets?	`FALSE`
`est.psi`	Estimate the error precision?	`TRUE`
`fixed.hyp`	Quick `TRUE`/`FALSE` toggle for all `est.*`	`NULL`

Table 1: The various options for which hyperparameters to estimate.

```
## Hyperparameters to estimate:
## lambda[1], lambda[2], hurst[1], psi
##
## Estimation methods available:
## direct, em, mixed, fixed
```

By default, the scale parameters and error precision are estimated, while the other hyperparameters are not. Note that the package does not optimise the degree of the polynomial kernel. To quick set all `est.*` options to `TRUE`/`FALSE`, use the `fixed.hyp` option. When there are several covariates using the same kernel, it is not possible to choose which of the covariates kernel hyperparameters to fix and which to estimate—the current implementation of the package is to either estimate all of them, or to fix all of them.

The Nyström method

The Nyström method of approximating the kernel matrix is supported by the package. This is set by calling the option `nystrom = TRUE`. This would then use a random sample of 10% of the total data available to estimate the kernel matrix. Alternatively, the `nystrom` option can be set to be a number equal to the Nyström sample size m as described in Section 1.3. Additionally, the seed for random sampling can be controlled by supplying a value to `nys.seed`.

```
R> (kernL(circ ~ age, Orange, kernel = "se", nystrom = 10, nys.seed = 123))

## Sample size: 35
## No. of covariates: 1
## Object size: 18.5 kB
##
## Kernel matrices:
##  1 se,1 [1:10, 1:35] 0.841 -0.159 -0.159 -0.159 -0.159 ...
##
## Hyperparameters to estimate:
## lambda, psi
##
## Estimation methods available:
## direct (Nystrom), fixed (Nystrom)
```

2.3. Model fitting and post-estimation

The main function to estimate I-prior models in the form of `ipriorKernel` objects is the `iprior()` function. Based on information from the `ipriorKernel`, it dispatches the estimation procedure to a selected subroutine and estimates the hyperparameters of the I-prior model, if any. The resulting fit is an object of class `ipriorMod`. Users may select from `"direct"`, `"em"`, `"mixed"`, or `"fixed"` as an option to supply to `method` in the `iprior()` function. For instance,

```
R> mod <- kernL(circ ~ age + tree + age:tree, data = Orange)
R> mod.fit <- iprior(mod, method = "direct")  # default method
R> mod.fit <- iprior(mod, method = "em")      # The EM algorithm
```

The `iprior()` function is also a wrapper function for the `kernL()` and estimation procedures. This means that users may call the `iprior()` function directly, with the exact same options that are available to `kernL()`, without having to invoke the two step procedure of calling `kernL()` and then `iprior()`. The following code fits the above model in one step using the EM algorithm and the fBm kernel.

```
R> mod <- iprior(circ ~ age + tree + age:tree, data = Orange, kernel = "fbm",
+                method = "em")
```

Exposing the kernel loader function for the end user has two advantages. Firstly, I-prior models can sometimes involve high-dimensional matrix multiplications, and these may take a long time to process. By "loading the kernel", the user is able to have a stored `ipriorKernel` object which can then be reused and fitted . A situation where this would be useful would be when the user would like to restart the EM algorithm from different set of starting values, or change to a different estimation procedure. Rather than having to go through all the kernel loading process all over again, the user can simply call the saved `ipriorKernel` object from memory. The caveat of storing the loaded kernel matrices is that it may take a large amount of memory since I-prior models have $O(n^2)$ storage requirements—this is the price to pay for the convenience. In practice on modern computers, however, this is unlikely to be a bottleneck.

Secondly, the `kernL()` function allows for flexible model fitting. The `logLik()` and `deviance()` S3 methods can be used on an `ipriorKernel` object to calculate the log-likelihood and deviance respectively at the hyperparameter value `theta`. There is flexibility in the sense that the user can then estimate the hyperparameters of the I-prior model through means other than `optim()`'s L-BFGS algorithm or the EM algorithm, which is built in to the `iprior()` function. A wide range of optimisation packages are available in R—see https://cran.r-project.org/web/views/Optimization.html for details.

Control options

Besides the estimation `method` and model options from `kernL()`, the other main option is the `control` of fitting methods. The user supplies the `iprior()` function a list containing the `control` options to modify. The available `control` options are:

1. `maxit`

The maximum number of iterations for either the L-BFGS optimisation procedure or the EM algorithm. Defaults to 100.

2. `em.maxit`

   When using `method = "mixed"`, this controls the number of initial EM steps before switching to the direct optimisation method. Defaults to 5.

3. `stop.crit`

   The stopping criterion for either the L-BFGS optimisation procedure or the EM algorithm. The algorithm terminates when it is not able to improve the log-likelihood value by `stop.crit`. Defaults to `1e-8`.

4. `theta0`

   The initial values for the hyperparameters. By default, these are set to random values, but may be changed by the user. Note that the hyperparameters have been transformed so that an unconstrained optimisation can be performed—see Appendix B for details.

5. `restarts`

   The estimation procedure can be restarted multiple times from different initial values by setting `restarts = TRUE`. This is especially useful when local optima are present. The run with the highest log-likelihood value is chosen automatically. By default, the number of restarts is equal to the number of available cores on the machine, and each run is parallelised on each core. This is achieved using the R packages **foreach** (Revolution Analytics and Weston 2015) and **doSNOW** (Microsoft Corporation and Weston 2017).

6. `no.cores`

   The number of cores to make available to `iprior()` for the parallel restarts. By default, it detects and sets this to the maximum number of cores available on the machine.

7. `par.maxit`

   The `maxit` for each parallel run when using `restarts`. By default, this is set to 5 only, with the intent of continuing estimation from the run with the highest likelihood value.

8. `silent`

   A logical option to turn on or off progress feedback from the estimation procedures.

An example for setting `control` options is shown below.

```
R> mod <- kernL(circ ~ age + tree + age:tree, data = Orange)
R> # Set a higher number of maximum iterations and more lenient stop.crit
R> mod.fit <- iprior(mod, control = list(maxit = 500, stop.crit = 1e-3))
R> # Set the mixed method to run more EM steps
R> mod.fit <- iprior(mod, control = list(em.maxit = 50))
R> # Start from lambda = c(2, 2) and psi = 0.5
R> mod.fit <- iprior(mod, control = list(theta0 = c(2, 2, log(0.5))))
R> # Perform four restarts on four cores
R> mod.fit <- iprior(mod, control = list(restarts = 4, no.cores = 4))
```

```
R> # Completely turn off reporting
R> mod.fit <- iprior(mod, control = list(silent = TRUE))
```

Refit and update

The `iprior` function is written as an S3 generic which is able to dispatch on `ipriorMod` objects as well. A practical situation for this is when the EM algorithm has not fully converged within the `maxit` supplied, then one can simply run

```
R> mod.fit <- iprior(mod.fit)
```

which then estimates the I-prior model from the previous obtained hyperparameter values. This is possible because the resulting `ipriorMod` object also contains the `ipriorKernel` object (from `kernL()`). Even better still, running

```
R> update(mod.fit, iter.update = 100)
```

updates the `ipriorMod` object with another 100 iterations using the same estimation method, and then overwrites `mod.fit` in environment without having to explicitly do the assignment. It is also possible to set a new estimation `method` and supply a new `control` list.

Post-estimation

There are several methods written for `ipriorMod` objects. The `print()` and `summary()` are ways to inspect the resulting fit and obtain information required for analysis and inference. This includes information on the I-prior model and fit information such as the model call/formula, the kernels used on the covariates, the estimation method and relating convergence information. The summary also includes the estimated hyperparameter values, along with their standard errors and p-values for an asymptotic Z-test of normality, the log-likelihood value and training mean squared error.

```
R> mod.fit <- iprior(circ ~ . ^ 2, data = Orange, method = "em",
+                    control = list(maxit = 5000))

## ================================
## Converged after 2446 iterations.

R> print(summary(mod.fit), wrap = TRUE)  # wrap option to neaten LaTeX output

## Call:
## iprior(formula = circ ~ .^2, data = Orange, method = "em",
##   control = list(maxit = 5000))
##
## RKHS used:
## Pearson (tree)
```

```
## Linear (age)
##
## Residuals:
##     Min.  1st Qu.   Median  3rd Qu.     Max.
## -16.7965  -7.2506  -0.5123   7.3863  18.1857
##
## Hyperparameters:
##           Estimate    S.E.      z P[|Z>z|]
## lambda[1]  -9.9940  3.5640 -2.804    0.005 **
## lambda[2]  -0.0002  0.0001 -2.372    0.018 *
## psi         0.0110  0.0030  3.644   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Closed-form EM algorithm. Iterations: 2446/5000
## Converged to within 1e-08 tolerance. Time taken: 2.667519 secs
## Log-likelihood value: -160.6596
## RMSE of prediction: 8.882306 (Training)
```

Inference on the scale parameters λ essentially give us a sense of importance of that covariate on the response, since functions in the RKHS are of the form

$$f(x) = \lambda \sum_{i=1}^{n} h(x, x_i) w_i,$$

as we saw earlier in (2). Thus, a scale parameter which is not statistically significant implies that the contribution of that function can be assumed to be nil. Furthermore, model comparisons can be done via log-likelihood ratio tests whose test statistic follows an asymptotic χ^2 distribution with degrees of freedom equal to the difference in the number of parameters estimated. When the number of parameters are the same, then it is a matter of selecting models with higher likelihood. Such a method of comparing marginal likelihoods can be seen as Bayesian model selection using empirical Bayes factors.

Fitted values may be obtained using `fitted()`, and predicted values at a new set of data points `newdata` using `predict()`. There is an option so that both function return the credibility interval for the predictions at the `alpha` level of significance. Although the hyperparameters are estimated using maximum likelihood, the regression function itself possesses a posterior distribution, and this is used for the credibility intervals. The `newdata` must be similar to the original data used to fit the model—for formula syntax, this must be a data frame containing identical column names; for non-formula syntax, the variables must be supplied as a list.

```
R> fitted(mod.fit)
```

```
## Training RMSE: 8.882306
##
## Predicted values:
##        1        2        3        4        5        6        7        8
```

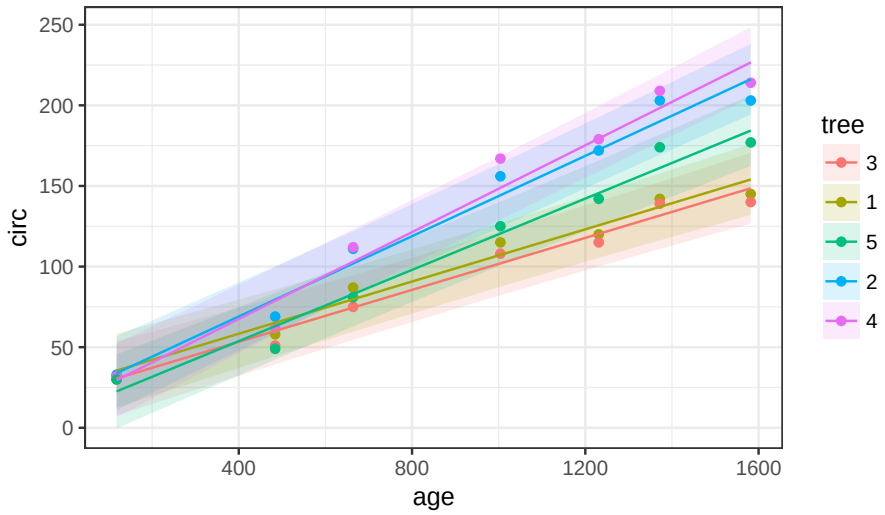

Figure 3: Plot of fitted regression line for the `Orange` data set. Confidence bands for predicted values are also shown.

```
##  35.508  65.139  79.711 107.236 125.614 137.029 154.030  33.899
##       9      10
##  79.481 101.898
## # ... with 25 more values
```

```
R> predict(mod.fit, Orange[1:5, ], intervals = TRUE, alpha = 0.05)
```

```
## Test RMSE: 6.726375
##
## Predicted values:
##      2.5%    Mean   97.5%
## 1  12.578  35.508  58.439
## 2  44.426  65.139  85.851
## 3  59.653  79.711  99.769
## 4  87.499 107.236 126.974
## 5 105.404 125.614 145.824
```

Several plot functions have been written for `ipriorMod` objects, with the `plot()` S3 method pointing to the `plot_fitted()` function. This plots the fitted regression line through the data points, with the response variable on the y-axis and the covariate on the x-axis. Thus, it is only useful when the data is able to be presented on a two-dimensional plane. The other plot functions are described in Table 2.

3. Modelling examples

We demonstrate the use of the **iprior** package with modelling a toy example from a simulated data set, as well as three other real-data examples.

R Function	Description
Methods	
`coef`	Extracts the estimates of the hyperparameters that have been estimated.
`sigma`	Extracts the estimate of the model standard deviation of errors (i.e., square root of the inverse error precision).
`fitted`	Returns the fitted value of the responses $\hat{y}_1, \ldots, \hat{y}_n$.
`predict`	Calculates fitted values from a new set of covariates.
`resid`	Returns the residuals $\hat{\epsilon}_1, \ldots, \hat{\epsilon}_n$.
`logLik`	Returns the log-likelihood value of the fitted I-prior model at the ML estimates.
`deviance`	Returns twice the negative log-likelihood value.
Accessor functions	
`get_intercept`	Obtains the intercept of the regression function.
`get_hyp`	Obtains all values of the model hyperparameters (both estimated and fixed values).
`get_lambda`	Obtains scale parameters used for the RKHS.
`get_psi`	Obtains the model error precision.
`get_se`	Obtains the standard errors for the estimated hyperparameters.
`get_kernels`	Obtains the RKHS used for the regression functions.
`get_kern_matrix`	Obtains the kernel matrix $\mathbf{H}_\eta$.
`get_prederror`	Obtains the (root) mean squared error of prediction.
`get_estl`	Obtains information on which hyperparameters were fixed and which were estimated.
`get_method`	Obtains the estimation method used to fit the model.
`get_convergence`	Obtain the convergence information.
`get_niter`	Obtain the number of iterations (Newton or EM steps) taken to fit the model.
`get_time`	Obtains the time taken to run the estimation procedure.
`get_size`	Obtains the size of the `ipriorKernel` object (where most of the large matrices are stored).
Plots	
`plot`	This currently points to `plot_fitted()` for convenience.
`plot_fitted`	Plot of fitted regression line.
`plot_resid`	Plot of residuals against fitted values.
`plot_iter`	Plot of the log-likelihood values over time/iteration.
`plot_ppc`	Plot of a posterior predictive check of the observed versus fitted distribution of the responses.

Table 2: Available methods, accessor functions and plot functions for an object of class `ipriorMod`.

3.1. Using the Nyström method

In this section, we investigate the use of the Nyström method of approximating the kernel matrix in estimating I-prior models applied to a toy data set. The data is obtained by randomly generating data points according to the true regression model

$$y_i = \text{const.} + 0.35 \cdot \phi(x_i; 1, 0.8^2) + 0.65 \cdot \phi(x_i; 4, 1.5^2) + \mathbb{1}[x_i > 4.5] \cdot e^{1.25(x_i - 4.5)} + \epsilon_i$$
$$\epsilon_i \overset{\text{iid}}{\sim} \text{N}(0, 0.9^2)$$

where $\phi(x; \mu, \sigma^2)$ represents the PDF of a normal distribution with mean μ and variance σ^2. The features of this regression function are two large bumps at the centres of the mixed Gaussian PDFs, and also a small bump right after $x > 4.5$ caused by the additional exponential function. The true regression function goes to positive infinity as x increases, and to zero as x decreases. 2,000 (x, y) points in the domain $x \in (-1, 5.5)$ have been generated by the built-in `gen_smooth()` function, which generates data from the regression model above[3].

```
R> dat <- gen_smooth(n = 2000, xlim = c(-1, 5.5), seed = 1)
R> head(dat)
```

```
##            y         X
## 1  0.6803514 -2.608953
## 2  3.6747031 -2.554039
## 3 -1.1563508 -2.381275
## 4  2.2657657 -2.280259
## 5  2.5398243 -2.214122
## 6  1.2929592 -2.170532
```

One could fit the regression model using all available data points, with an I-prior from the fBm-0.5 RKHS of functions as follows (note that the `silent` option is used to suppress the output from the `iprior()` function):

```
R> (mod.full <- iprior(y ~ X, dat, kernel = "fbm",
+                       control = list(silent = TRUE)))
```

```
## Log-likelihood value: -4355.075
##
##  lambda     psi
## 2.30244 0.23306
```

To implement the Nyström method, the option `nystrom = 50` was added to the above function call, which uses 50 randomly selected data points for the Nyström approximation.

```
R> (mod.nys <- iprior(y ~ X, dat, kernel = "fbm", nystrom = 50,
+                      control = list(silent = TRUE)))
```

[3]Random fluctuations have also been added to the (x, y) coordinates.

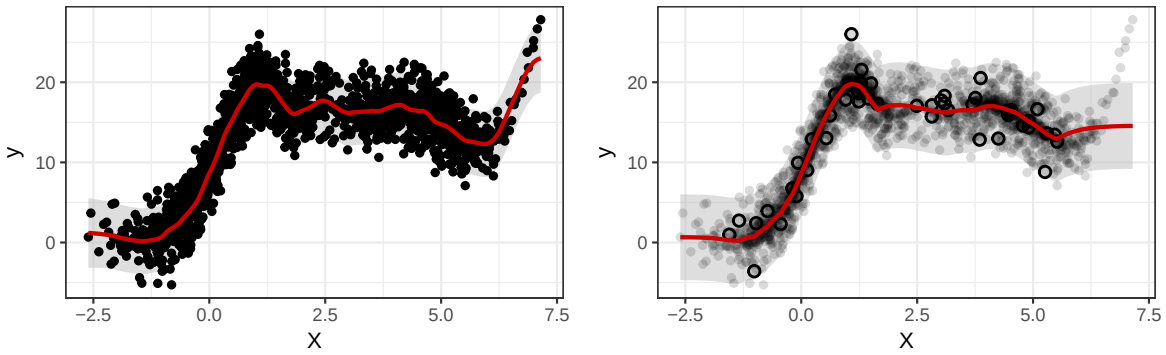

Figure 4: Plot of predicted regression function for the full model (left) and the Nyström approximated method (right). For the Nyström plot, the data points that were active are shown by circles with bold outlines.

```
## Log-likelihood value: -1945.33
##
##  lambda     psi
## 1.64833 0.13538
```

The hyperparameters estimated for both models are slightly different. The log-likelihood is also different, but this is attributed to information loss due to the approximation procedure. Nevertheless, we see from Figure 4 that the estimated regression functions are quite similar in both the full model and the approximated model. The main difference is that the the Nyström method was not able to extrapolate the right hand side of the plot well, because it turns out that there were no data points used from this region. This can certainly be improved by using a more intelligent sampling scheme. The full model took a little under 15 minutes to converge, while the Nyström method took just seconds. Storage savings is significantly higher with the Nyström method as well.

```
R> get_time(mod.full); get_size(mod.full, units = "MB")

## 14.41963 mins
## 128.2 MB

R> get_time(mod.nys); get_size(mod.nys)

## 1.262599 secs
## 965.2 kB
```

3.2. Random effects models

In this section, a comparison between a standard random effects model and the I-prior approach for estimating varying intercept and slopes model is illustrated. The example concerns control data[4] from several runs of radioimmunoassays (RIA) for the protein insulin-like growth

factor (IGF-I) (explained in further detail in Davidian and Giltinan 1995, Section 3.2.1). RIA is a in vitro assay technique which is used to measure concentration of antigens—in our case, the IGF-I proteins. When an RIA is run, control samples at known concentrations obtained from a particular lot are included for the purpose of assay quality control. It is expected that the concentration of the control material remains stable as the machine is used, up to a maximum of about 50 days, at which point control samples from a new batch is used to avoid degradation in assay performance.

```
R> data(IGF, package = "nlme")
R> head(IGF)
```

```
## Grouped Data: conc ~ age | Lot
##   Lot age conc
## 1   1   7 4.90
## 2   1   7 5.68
## 3   1   8 5.32
## 4   1   8 5.50
## 5   1  13 4.94
## 6   1  13 5.19
```

The data consists of IGF-I concentrations (`conc`) from control samples from 10 different lots measured at differing `age`s of the lot. The data were collected with the aim of identifying possible trends in control values `conc` with `age`, ultimately investigating whether or not the usage protocol of maximum sample age of 50 days is justified. Pinheiro and Bates (2000) remarks that this is not considered a longitudinal problem because different samples were used at each measurement.

We shall model the IGF data set using the I-prior methodology using the regression function

$$f(\texttt{age}, \texttt{Lot}) = f_1(\texttt{age}) + f_2(\texttt{Lot}) + f_{12}(\texttt{age}, \texttt{Lot})$$

where f_1 lies in the linear RKHS $\mathcal{F}_1$, f_2 in the Pearson RKHS $\mathcal{F}_2$ and f_{12} in the tensor product space $\mathcal{F}_{12} = \mathcal{F}_1 \otimes \mathcal{F}_2$. The regression function f then lies in the RKHS $\mathcal{F} = \mathcal{F}_1 \oplus \mathcal{F}_2 \oplus \mathcal{F}_{12}$ with kernel equal to the sum of the kernels from each of the RKHSs[5]. The explanation here is that the `conc` levels are assumed to be related to both `age` and `Lot`, and in particular, the contribution of `age` on `conc` varies with each individual `Lot`. This gives the intended effect of a linear mixed-effects model, which is thought to be suitable in this case, in order to account for within-lot and between-lot variability. We first fit the model using the **iprior** package, and then compare the results with the standard random effects model using `lme4::lmer()`. The command to fit the I-prior model using the EM algorithm is

```
R> mod.iprior <- iprior(conc ~ age * Lot, IGF, method = "em")
```

```
## =======================================
## Converged after 57 iterations.
```

[4]This data is available in the R package **nlme** (Pinheiro, Bates, DebRoy, Sarkar, and R Core Team 2017).
[5]This is often known as the functional ANOVA decomposition.

```
R> summary(mod.iprior)

## Call:
## iprior(formula = conc ~ age * Lot, data = IGF, method = "em")
##
## RKHS used:
## Linear (age)
## Pearson (Lot)
##
## Residuals:
##    Min. 1st Qu.  Median 3rd Qu.    Max.
## -4.4889 -0.3798 -0.0090  0.2563  4.3973
##
## Hyperparameters:
##           Estimate   S.E.      z P[|Z>z|]
## lambda[1]   0.0000 0.0002 -0.004    0.997
## lambda[2]   0.0007 0.0030  0.238    0.812
## psi         1.4576 0.1366 10.672   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Closed-form EM algorithm. Iterations: 57/100
## Converged to within 1e-08 tolerance. Time taken: 2.789562 secs
## Log-likelihood value: -291.9033
## RMSE of prediction: 0.8273639 (Training)
```

To make inference on the covariates, we look at the scale parameters `lambda`. We see that both scale parameters for `age` and `Lot` are close to zero, and a test of significance is not able to reject the hypothesis that these parameters are indeed null. We conclude that neither `age` nor `Lot` has a linear effect on the `conc` levels. The plot of the fitted regression line in Figure 5 does show an almost horizontal line for each `Lot`.

The standard random effects model, as explored by Davidian and Giltinan (1995) and Pinheiro and Bates (2000), is

$$\begin{gathered}
\texttt{conc}_{ij} = \beta_{0j} + \beta_{1j}\texttt{age}_{ij} + \epsilon_{ij} \\
\begin{pmatrix}\beta_{0j}\\ \beta_{1j}\end{pmatrix} \sim \mathrm{N}\left(\begin{pmatrix}\beta_0\\ \beta_1\end{pmatrix}, \begin{pmatrix}\sigma_0^2 & \sigma_{01}\\ \sigma_{01} & \sigma_1^2\end{pmatrix}\right) \\
\epsilon_{ij} \sim \mathrm{N}(0, \sigma^2)
\end{gathered}$$

for $i = 1, \ldots, n_j$ and the index j representing the 10 `Lots`. Fitting this model using `lmer`, we can test for the significance of the fixed effect β_0, for which we find that it is not (p-value $= 0.616$), and arrive at the same conclusion as in the I-prior model. However, we notice that the package reports a perfect negative correlation between the random effects, σ_{01}. This indicates a potential numerical issue when fitting the model—a value of exactly -1, 0 or 1 is typically imposed by the package to force through estimation in the event of non-positive definite covariance matrices arising. We can inspect the eigenvalues of the covariance matrix for the random effects to check that they are indeed non-positive definite.

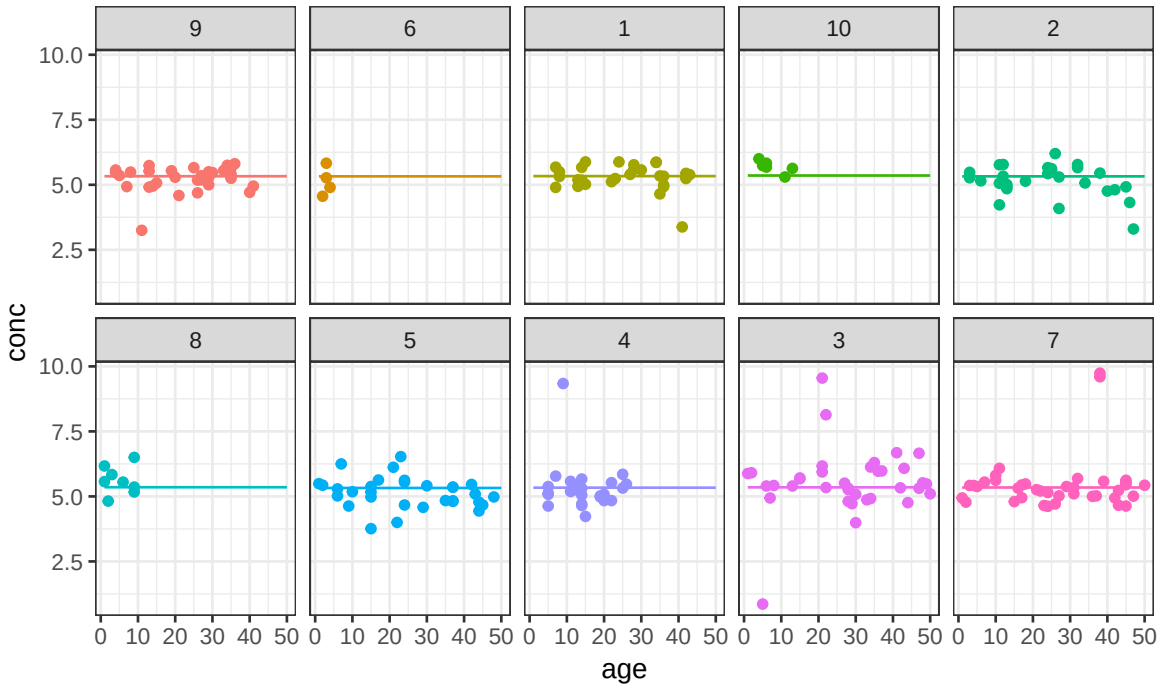

Figure 5: Plot of fitted regression line for the I-prior model on the IGF data set, separated into each of the 10 lots.

```
R> (mod.lmer <- lmer(conc ~ age + (age | Lot), IGF))

## Linear mixed model fit by REML ['lmerMod']
## Formula: conc ~ age + (age | Lot)
##    Data: IGF
## REML criterion at convergence: 594.3662
## Random effects:
##  Groups   Name        Std.Dev. Corr
##  Lot      (Intercept) 0.082507
##           age         0.008092 -1.00
##  Residual             0.820628
## Number of obs: 237, groups:  Lot, 10
## Fixed Effects:
## (Intercept)          age
##    5.374974    -0.002535

R> eigen(VarCorr(mod.lmer)$Lot)

## eigen() decomposition
## $values
## [1]  6.872939e-03 -1.355253e-20
##
```

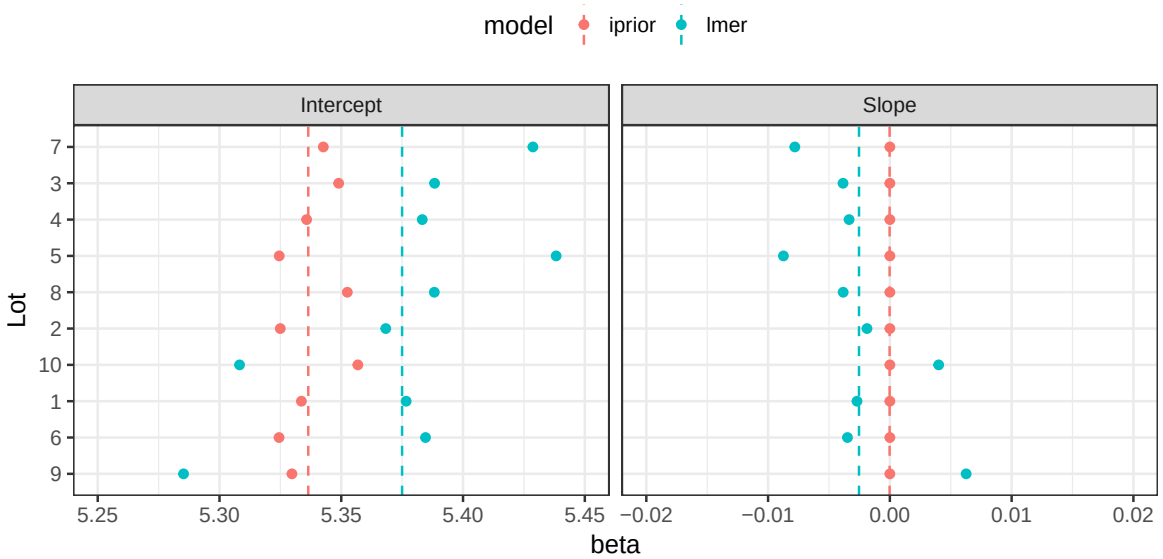

Figure 6: A comparison of the estimates for random intercepts and slopes (denoted as points) using the I-prior model and the standard random effects model. The dashed vertical lines indicate the fixed effect values.

```
## $vectors
##              [,1]        [,2]
## [1,] -0.99522490 -0.09760839
## [2,]  0.09760839 -0.99522490
```

Degenerate covariance matrices often occur in models with a large number of random coefficients. These are typically solved by setting restrictions which then avoids overparameterising the model. One advantage of the I-prior method for varying intercept/slopes model is that the positive-definiteness is automatically taken care of. Furthermore, I-prior models typically require less number of parameters to fit a similar varying intercept/slopes model – in the above example, the I-prior model estimated only three parameters, while the standard random effects model estimated a total of six parameters.

It is also possible to "recover" the estimates of the standard random effects model from the I-prior model, albeit in a slighly manual fashion. Denote by f^j the individual linear regression lines for each of the $j = 1, \ldots, 10$ `Lots`. Then, each of these f^j has a slope and intercept for

Parameter	`iprior`	`lmer`
σ_0	0.012	0.083
σ_1	0.000	0.008
ρ_{01}	0.690	-1.000

Table 3: A comparison of the estimates for the covariance matrix of the random effects using the I-prior model and the standard random effects model.

which we can estimate from the fitted values $\hat{f}^j(x_{ij})$, $i = 1, \ldots, n_j$. This would give us the estimate of the posterior mean of the random intercepts and slopes; these would typically be obtained using empirical-Bayes methods in the case of the standard random effects model.

Furthermore, σ_0^2 and σ_1^2 gives a measure of variability of the intercepts and slopes of the different groups, and this can be calculated from the estimates of the random intercepts and slopes. In the same spirit, $\rho_{01} = \sigma_{01}/(\sigma_0\sigma_1)$, which is the correlation between the random intercept and slope, can be similarly calculated. Finally, the fixed effects can be estimated from the intercept and slope of the best fit line running through the I-prior estimated `conc` values. The intuition for this is that the fixed effects are essentially the ordinary least squares (OLS) of a linear model if the groupings are disregarded. Figure 6 illustrates the differences in the estimates for the random coefficients, while Table 3 illustrates the differences in the estimates for the covariance matrix. Minor differences do exist, with the most noticeable one being that the slopes in the I-prior model are categorically estimated as zero, and the sign of the correlation ρ_{01} being opposite in both models. Even so, the conclusions from both models are similar.

3.3. Longitudinal data analysis

We consider a balanced longitudinal data set consisting of weights in kilograms of 60 cows, 30 of which were randomly assigned to treatment group A, and the remaining 30 to treatment group B. The animals were weighed 11 times over a 133-day period; the first 10 measurements for each animal were made at two-week intervals and the last measurement was made one week later. This experiment was reported by Kenward (1987), and the data set is included as part of the package **jmcm** (Pan and Pan 2016) in R. The variable names have been renamed for convenience.

```
R> data(cattle, package = "jmcm")
R> names(cattle) <- c("id", "time", "group", "weight")
R> cattle$id <- as.factor(cattle$id)  # convert to factors
R> str(cattle)

## 'data.frame': 660 obs. of  4 variables:
##  $ id    : Factor w/ 60 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ time  : num  0 14 28 42 56 70 84 98 112 126 ...
##  $ group : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
##  $ weight: int  233 224 245 258 271 287 287 287 290 293 ...
```

The response variable of interest are the `weight` growth curves, and the aim is to investigate whether a treatment effect is present. The usual approach to analyse a longitudinal data set such as this one is to assume that the observed growth curves are realizations of a Gaussian process. For example, Kenward (1987) assumed a so-called ante-dependence structure of order k, which assumes an observation depends on the previous k observations, but given these, is independent of any preceeding observations.

Using the I-prior, it is not necessary to assume the growth curves were drawn randomly. Instead, it suffices to assume that they lie in an appropriate function class. For this example, we assume that the function class is the fBm RKHS, i.e., we assume a smooth effect of time

Model	Explanation	Formula (`weight ~ ...`)
1	Growth does not vary with treatment nor among cows	`time`
2	Growth varies among cows only	`id * time`
3	Growth varies with treatment only	`group * time`
4	Growth varies with treatment and among cows	`id * time + group * time`
5	Growth varies with treatment and among cows, with an interaction effect between treatment and cows	`id * group * time`

Table 4: A brief description of the five models fitted using I-priors.

on weight. The growth curves form a multidimensional (or functional) response equivalent to a "wide" format of representing repeated measures data. In our analysis using the **iprior** package, we used the "long" format and thus our (unidimensional) sample size n is equal to 60 cows $\times$ 11 repeated measurements. We also have two covariates potentially influencing growth, namely the cow subject `id` and also treatment `group`. The regression model can then be thought of as

$$\texttt{weight} = \alpha + f(\texttt{id}, \texttt{group}, \texttt{time}) + \epsilon$$
$$\epsilon \sim \mathrm{N}(0, \psi^{-1}).$$

We assume iid errors, and in addition to a smooth effect of `time`, we further assume a nominal effect of both cow `id` and treatment `group` using the Pearson RKHS. In the **iprior** package, factor type objects are treated with the Pearson kernel automatically, and the only `model` option we need to specify is the `kernel = "fbm"` option for the `time` variable. We have opted not to estimate the Hurst coefficient in the interest of computational time, and instead left it at the default value of 0.5. Table 4 explains the five models we have fitted.

The simplest model fitted was one in which the growth curves do not depend on the treatment effect or individual cows. We then added treatment effect and the cow `id` as covariates, separately first and then together at once. We also assumed that both of these covariates are time-varying, and hence added also the interaction between these covariates and the `time` variable. The final model was one in which an interaction between treatment effect and individual cows was assumed, which varied over time.

All models were fitted using the `mixed` estimation method. Compared to the EM algorithm alone, we found that the combination of direct optimisation with the EM algorithm in the `mixed` routine fits the model about six times faster for this data set due to slow convergence of EM algorithm. Here is the code and output for fitting the first model:

```
R> # Model 1: weight ~ f(time)
R> (mod1 <- iprior(weight ~ time, cattle, kernel = "fbm", method = "mixed"))

## Running 5 initial EM iterations
## ================================================================
## Now switching to direct optimisation
```

Model	Formula (`weight ~ ...`)	Log-likelihood	Error S.D.	Number of parameters
1	`time`	-2789.23	16.33	1
2	`id * time`	-2789.20	16.32	2
3	`group * time`	-2787.03	15.91	2
4	`id * time + group * time`	-2787.03	15.91	3
5	`id * group * time`	-2276.74	3.33	3

Table 5: Summary of the five I-prior models fitted to the cow data set.

```
## final  value 1394.615060
## converged
## Log-likelihood value: -2789.231
##
##  lambda     psi
## 0.83662 0.00375
```

The results of the model fit are summarised in Table 5. We can test for a treatment effect by testing Model 4 against the alternative that Model 2 is true. The log-likelihood ratio test statistic is $D = -2(-2789.20 - (-2787.03)) = 4.35$ which has an asymptotic chi-squared distribution with $3 - 2 = 1$ degree of freedom. The p-value for this likelihood ratio test is less than 10^{-6}, so we conclude that Model 4 is significantly better than Model 2.

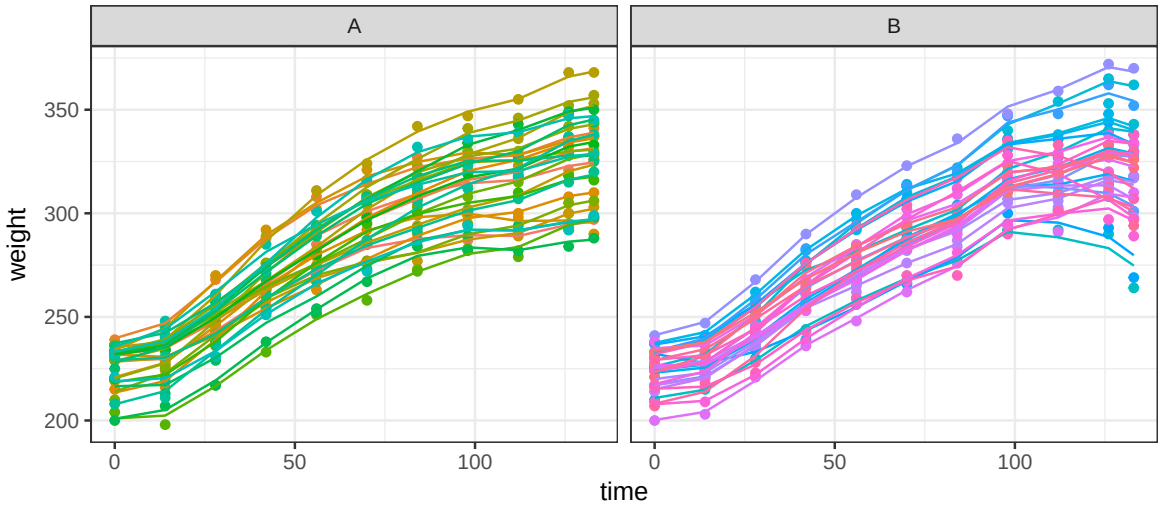

Figure 7: A plot of the I-prior fitted regression curves from Model 5. In this model, growth curves differ among cows and by treatment effect (with an interaction between cows and treatment effect), thus producing these 60 individual lines, one for each cow, split between their respective treatment groups (A or B).

We can next investigate whether the treatment effect differs among cows by comparing Model 5 against Model 4. As these models have the same number of parameters, we can simply choose the one with the higher likelihood, which is Model 5. We conclude that treatment does indeed

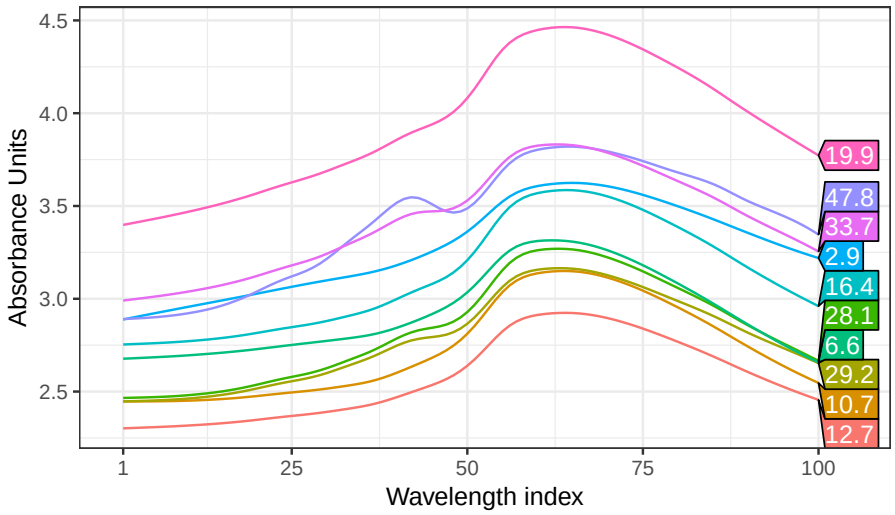

Figure 8: Sample of spectrometric curves used to predict fat content of meat. For each meat sample the data consists of a 100 channel spectrum of absorbances and the contents of moisture, fat (numbers shown in boxes) and protein measured in percent. The absorbance is $-\log 10$ of the transmittance measured by the spectrometer. The three contents, measured in percent, are determined by analytic chemistry.

have an effect on growth, and that the treatment effect differs among cows. A plot of the fitted regression curves onto the cow data set is shown in Figure 7.

3.4. Regression with a functional covariate

We illustrate the prediction of a real valued response with a functional covariate using a widely analysed data set for quality control in the food industry. The data[6] contain samples of spectrometric curve of absorbances of 215 pieces of finely chopped meat, along with their water, fat and protein content. These data are recorded on a Tecator Infratec Food and Feed Analyzer working in the wavelength range 850–1050 nm by the Near Infrared Transmission (NIT) principle. Absorption data has not been measured continuously, but instead 100 distinct wavelengths were obtained. Figure 8 shows a sample of 10 such spectrometric curves.

For our analyses and many others' in the literature, the first 172 observations in the data set are used as a training sample for model fitting, and the remaining 43 observations as a test sample to evaluate the predictive performance of the fitted model. The focus here is to use the **iprior** package to fit several I-prior models to the Tecator data set, and calculate out-of-sample predictive error rates. We compare the predictive performance of I-prior models against Gaussian process regression and the many other different methods applied on this data set. These methods include neural networks (Thodberg 1996), kernel smoothing (Ferraty and Vieu 2006), single and multiple index functional regression models (Chen, Hall, and Müller 2011), sliced inverse regression (SIR) and sliced average variance estimation (SAVE), multivariate adaptive regression splines (MARS), partial least squares (PLS), and functional additive model with and without component selection (FAM & CSEFAM). An analysis of this

[6] Obtained from Tecator (see `http://lib.stat.cmu.edu/datasets/tecator` for details). We used the version made available in the dataframe `tecator` from the R package **caret** (Kuhn *et al.* 2017).

data set using the SIR and SAVE methods were conducted by Lian and Li (2014), while the MARS, PLS and (CSE)FAM methods were studied by Zhu, Yao, and Zhang (2014). Table 6 tabulates the results of all of these methods from the various references.

Assuming a regression model as in (1), we would like to model the `fat` content y_i using the spectral curves x_i. Let $x_i(t)$ denote the absorbance for wavelength $t = 1, \ldots, 100$. From Figure 8, it appears that the curves are smooth enough to be differentiable, and therefore it is reasonable to assume that they lie in the Sobolev-Hilbert space as discussed in Section 2.1.7. We take first differences of the 100-dimensional matrix, which leaves us with the 99-dimensional covariate saved in the object named `absorp`. The `fat` and `absorp` data have been split into `*.train` and `*.test` samples, as mentioned earlier. Our first modelling attempt is to fit a linear effect by regressing the responses `fat.train` against a single high-dimensional covariate `absorp.train` using the linear RKHS and the direct optimisation method.

```
R> # Model 1: Canonical RKHS (linear)
R> (mod1 <- iprior(y = fat.train, absorp.train))

## iter   10 value 222.653144
## final  value 222.642108
## converged
## Log-likelihood value: -445.2844
##
##     lambda        psi
## 4576.86595    0.11576
```

Our second and third model uses polynomial RKHSs of degrees two and three, which allows us to model quadratic and cubic terms of the spectral curves respectively. We also opted to estimate a suitable offset parameter, and this is called to `iprior()` with the option `est.offset = TRUE`. Each of the two models has a single scale parameter, an offset parameter, and an error precision to be estimated. The direct optimisation method has been used, and while both models converged regularly, it was noticed that there were multiple local optima that hindered the estimation (output omitted).

```
R> # Model 2: Polynomial RKHS (quadratic)
R> mod2 <- iprior(y = fat.train, absorp.train, kernel = "poly2",
+                 est.offset = TRUE)
R> # Model 3: Polynomial RKHS (cubic)
R> mod3 <- iprior(y = fat.train, absorp.train, kernel = "poly3",
+                 est.offset = TRUE)
```

Next, we attempt to fit a smooth dependence of fat content on the spectrometric curves using the fBm RKHS. By default, the Hurst coefficient for the fBm RKHS is set to be 0.5. However, with the option `est.hurst = TRUE`, the Hurst coefficient is included in the estimation procedure. We fit models with both a fixed value for Hurst (at 0.5) and an estimated value for Hurst. For both of these models, we encountered numerical issues when using the direct optimisation method. The L-BFGS algorithm kept on pulling the hyperparameter towards extremely high values, which in turn made the log-likelihood value greater than the machine's

largest normalised floating-point number (`.Machine$double.xmax = 1.797693e+308`). Investigating further, it seems that estimates at these large values give poor training and test error rates, though likelihood values here are high (local optima). To get around this issue, we used the EM algorithm to estimate the fixed Hurst model, and the `mixed` method for the estimated Hurst model. For both models, the `stop.crit` was relaxed and set to `1e-3` for quicker convergence, though this did not affect the predictive abilities compared to a more stringent `stop.crit`.

```
R> # Model 4: fBm RKHS (default Hurst = 0.5)
R> (mod4 <- iprior(y = fat.train, absorp.train, kernel = "fbm",
+                   method = "em", control = list(stop.crit = 1e-3)))

## ============================================
## Converged after 65 iterations.
## Log-likelihood value: -204.4592
##
##     lambda        psi
##    3.24112 1869.32897


R> # Model 5: fBm RKHS (estimate Hurst)
R> (mod5 <- iprior(fat.train, absorp.train, kernel = "fbm", method = "mixed",
+                   est.hurst = TRUE, control = list(stop.crit = 1e-3)))

## Running 5 initial EM iterations
## ================================================================================
## Now switching to direct optimisation
## iter   10 value 115.648462
## final  value 115.645800
## converged
## Log-likelihood value: -231.2923
##
##    lambda     hurst       psi
## 204.97184   0.70382   9.96498
```

Finally, we fit an I-prior model using the SE RKHS with lengthscale estimated. Here we illustrate the use of the `restarts` option, in which the model is fitted repeatedly from different starting points. In this case, eight random initial parameter values were used and these jobs were parallelised across the eight available cores of the machine. The additional `par.maxit` option in the `control` list is an option for the maximum number of iterations that each parallel job should do. We have set it to 100, which is the same number for `maxit`, but if `par.maxit` is less than `maxit`, the estimation procedure continues from the model with the best likelihood value. We see that starting from eight different initial values, direct optimisation leads to (at least) two log-likelihood optima sites, -231.5 and -680.5.

```
R> # Model 6: SE kernel
R> (mod6 <- iprior(fat.train, absorp.train, est.lengthscale = TRUE,
+                  kernel = "se", control = list(restarts = TRUE,
+                                                par.maxit = 100)))

## Performing 8 random restarts on 8 cores
## ======================================================================
## Log-likelihood from random starts:
##     Run 1     Run 2     Run 3     Run 4     Run 5     Run 6     Run 7
## -231.5440 -680.4637 -231.5440 -680.4636 -231.5440 -231.5440 -231.5440
##     Run 8
## -680.4637
## Continuing on Run 7
## final  value 115.771932
## converged
## Log-likelihood value: -231.544
##
##      lambda lengthscale         psi
##    96.12702     0.09269     6.15448
```

Predicted values of the test data set can be obtained using the `predict()` function. An example for obtaining the first model's predicted values is shown below. The `predict()` method for `ipriorMod` objects also return the test MSE if the vector of test data is supplied.

```
R> predict(mod1, newdata = list(absorp.test), y.test = fat.test)

## Test RMSE: 2.890353
##
## Predicted values:
##  [1] 43.607 20.444  7.821  4.491  9.044  8.564  7.935 11.615 13.807
## [10] 17.359
## # ... with 33 more values
```

These results are summarised in Table 6. For the I-prior models, a linear effect of the functional covariate gives a training RMSE of 2.89, which is improved by both the qudratic and cubic model. The training RMSE is improved further by assuming a smooth RKHS of functions for f, i.e. the fBm and SE RKHSs. When it comes to out-of-sample test error rates, the cubic model gives the best RMSE out of the I-prior models for this particular data set, with an RMSE of 0.58. This is followed closely by the fBm RKHS with estimated Hurst coefficient (fBm-0.70) and also the fBm RKHS with default Hurst coefficient (fBm-0.50). The best performing I-prior model is only outclassed by the neural networks of Thodberg (1996), who also performed model selection using automatic relevance determination (ARD). The I-prior models also give much better test RMSE than Gaussian process regression[7].

[7]GPR models were fit using `gausspr()` in **kernlab**.

	RMSE	
Model	Train	Test
I-prior		
Linear	2.89	2.89
Quadratic	0.72	0.97
Cubic	0.37	0.58
Smooth (fBm-0.50)	0.00	0.68
Smooth (fBm-0.70)	0.19	0.63
Smooth (SE-0.09)	0.35	1.85
Gaussian process regression		
Linear	0.18	2.36
Smooth (SE-7.04)	0.17	2.10
Others		
Neural network[a]		0.36
Kernel smoothing[b]		1.49
Single/multiple indices model[c]		1.55
Sliced inverse regression		0.90
Sliced average variance estimation		1.70
MARS[d]		0.88
Partial least squares[d]		1.01
CSEFAM[d]		0.85

[a] Neural network best results with automatic relevance determination (ARD) quoted.
[b] Data set used was a 160/55 training/test split.
[c] These are results of a leave-one-out cross-validation scheme.
[d] Data set used was an extended version with $n = 240$, and a random 185/55 training/test split.

Table 6: A summary of the root mean squared error (RMSE)of prediction for the I-prior models and various other methods in literature conducted on the Tecator data set. Values for the methods under *Others* were obtained from the corresponding references cited earlier.

4. Summary and discussion

The **iprior** package provides methods to estimate and analyse I-prior regression models. Philosophically, the I-prior approach is very different from GPR in that the former starts with a function space and defines an automatic prior over that space, while the latter starts with a prior, to be chosen from prior experience or subjectively. From a computational point of view, the two methods differ in the specification of the covariance kernel, where the I-prior approach is particularly attractive in combination with the EM algorithm. The use of the squared Gram matrices also means that it is important that orthogonal decompositions are used, because we would be wasting computational resources in squaring the kernel matrix naively otherwise. Further, most GPR software opt to estimate models with fixed kernel parameters—this is a crucial difference in I-prior modelling in which most, if not all, hyper-

parameters are estimated for inferential purposes. We had also not come across any software package that implements the fBm kernel.

We have identified several areas of improvement, and the first is regarding estimation speed. The nature of I-prior models means that the estimation scales as $O(n^3N)$, with n being the sample size and N being the number of EM or L-BFGS iterations. As it stands, our minimal tests suggests that the **iprior** package would struggle with data sets of sizes $n \geq 5000$, unless the Nyström method is used (though this is not applicable in all cases, and the Nyström method's approximation quality needs to be accounted for as well). While every care has been taken to ensure efficiency in the code, it is clear that more work needs to be done to overcome this speed issue, either from an algorithmic standpoint or in the actual code implementation.

Furthermore, storage requirements for I-prior models is a concern for large data sets, which is the second area for improvement. The package opts to calculate and hard store the kernel matrices so that these can simply be called by the estimation methods. Although it is possible to be more efficient by recalculating the kernel matrices from the data, or by opting to store the $n \times 1$ pre-multiplied vectors that occur most frquently e.g. $\mathbf{H}_\eta\tilde{\mathbf{w}}$ and $\mathbf{\Sigma}_\theta^{-1}\mathbf{y}$, the $O(n^2)$ storage requirement still cannot be beaten as there is still an $O(n^2)$ element that needs to be elucidated and stored (temporarily) in memory. In our case, this is the eigendecomposition routine.

Thirdly, we would like to develop the ability to handle multidimensional responses. This was mentioned earlier in Section 3.3, where we saw cow growth data represented as multidimensional vectors of weights over time. In the current version of the package, we needed to convert this into the "long" data format, and thus increasing the sample size from n cows to $n \times T$, i.e., T time points for each cow. Keeping the data in "wide" format would have been more computationally efficient, and open the possibility to support more general multidimensional response regression models.

Fourthly and lastly, the package can be extended to deal with non-iid errors, i.e., $(\epsilon_1, \ldots, \epsilon_n) \sim \mathrm{N}_n(\mathbf{0}, \mathbf{\Psi}^{-1})$ and Ψ has a more general symmetric form. In particular, the ability to deal with autoregressive errors would add flexibility.

References

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using **lme4**." *Journal of Statistical Software*, **67**(1), 1–48. doi:10.18637/jss.v067.i01.

Bergsma W (2017). "Regression and Classification with I-priors." *Manuscript in preparation.* arXiv:1707.00274.

Berlinet A, Thomas-Agnan C (2011). *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer-Verlag. doi:10.1007/978-1-4419-9096-9.

Bishop C (2006). *Pattern Recognition and Machine Learning*. Springer-Verlag.

Bürkner PC (2017). "brms: An R Package for Bayesian Multilevel Models Using Stan." *Journal of Statistical Software*, **80**(1), 1–28. doi:10.18637/jss.v080.i01.

Carpenter B, Gelman A, Hoffman M, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017). "Stan: A Probabilistic Programming Language." *Journal of Statistical Software, Articles*, **76**(1), 1–32. doi:10.18637/jss.v076.i01.

Chen D, Hall P, Müller HG (2011). "Single and Multiple Index Functional Regression Models with Nonparametric Link." *The Annals of Statistics*, **39**(3), 1720–1747. doi:10.1214/11-AOS882.

Davidian M, Giltinan DM (1995). *Nonlinear Models for Repeated Measurement Data*. Chapman and Hall/CRC.

Denwood M (2016). "**runjags**: An R Package Providing Interface Utilities, Model Templates, Parallel Computing Methods and Additional Distributions for MCMC Models in JAGS." *Journal of Statistical Software*, **71**(9), 1–25. doi:10.18637/jss.v071.i09.

Eddelbuettel D, Francois R (2011). "**Rcpp**: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

Ferraty F, Vieu P (2006). *Nonparametric Functional Data Analysis*. 1st edition. Springer-Verlag. doi:10.1007/0-387-36620-2.

Fowlkes C, Belongie S, Malik J (2001). "Efficient Spatiotemporal Grouping Using the Nyström Method." In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 1, pp. 231–238. doi:10.1109/CVPR.2001.990481.

Gretton A, Bousquet O, Smola A, Schölkopf B (2005). "Measuring Statistical Dependence with Hilbert-Schmidt Norms." In S Jain, HU Simon, E Tomita (eds.), *Proceedings of the 16th International Conference of Algorithmic Learning Theory*, pp. 63–77. Springer-Verlag. doi:10.1007/11564089_7.

Kalaitzis A, Honkela A, Gao P, Lawrence ND (2014). "**gptk**: Gaussian Processes Tool-Kit." R package version 1.08, URL https://CRAN.R-project.org/package=gptk.

Kalaitzis A, Lawrence ND (2011). "A Simple Approach to Ranking Differentially Expressed Gene Expression Time Courses through Gaussian Process Regression." *BMC Bioinformatics*, **12**(1), 180. doi:10.1186/1471-2105-12-180.

Karatzoglou A, Smola A, Hornik K, Zeileis A (2004). "**kernlab** - An S4 Package for Kernel Methods in R." *Journal of Statistical Software*, **11**(9), 1–20. doi:10.18637/jss.v011.i09.

Kenward MG (1987). "A Method for Comparing Profiles of Repeated Measurements." *Journal of the Royal Statistical Society C (Applied Statistics)*, **36**(3), 296–308. doi:10.2307/2347788.

Kuhn M, *et al.* (2017). "**caret**: Classification and Regression Training." R package version 6.0–77, URL https://CRAN.R-project.org/package=caret.

Lian H, Li G (2014). "Series Expansion for Functional Sufficient Dimension Reduction." *Journal of Multivariate Analysis*, **124**(C), 150–165. doi:10.1016/j.jmva.2013.10.019.

Lunn DJ, Thomas A, Best N, Spiegelhalter D (2000). "WinBUGS - A Bayesian modelling framework: Concepts, structure, and extensibility." *Statistics and Computing*, **10**(4), 325–337. doi:10.1023/A:1008929526011.

MacDonald B, Ranjan P, Chipman H (2015). "**GPfit**: An R Package for Fitting a Gaussian Process Model to Deterministic Simulator Outputs." *Journal of Statistical Software*, **64**(12), 1–23. doi:10.18637/jss.v064.i12.

Microsoft Corporation, Weston S (2017). "**doSNOW**: Foreach Parallel Adaptor for the **snow** Package." R package version 1.0.15, URL https://CRAN.R-project.org/package=doSNOW.

Pan J, Pan Y (2016). "**jmcm**: Joint Mean-Covariance Models using Armadillo and S4." R package version 0.1.7.0, URL https://CRAN.R-project.org/package=jmcm.

Pinheiro J, Bates D, DebRoy S, Sarkar D, R Core Team (2017). "**nlme**: Linear and Nonlinear Mixed Effects Models." R package version 3.1-131, URL https://CRAN.R-project.org/package=nlme.

Pinheiro JC, Bates DM (2000). *Mixed-Effects Models in S and S-plus*. Springer-Verlag. doi:10.1007/b98882.

Plummer M (2003). "JAGS: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, volume 124, p. 125. Vienna, Austria.

Quiñonero-Candela J, Rasmussen CE (2005). "A Unifying View of Sparse Approximate Gaussian Process Regression." *Journal of Machine Learning Research*, **6**, 1939–1959.

Ramsay JO, Wickham H, Graves S, Hooker G (2017). "**fda**: Functional Data Analysis." R package version 2.4.7, URL https://CRAN.R-project.org/package=fda.

Rasmussen CE, Williams CKI (2006). *Gaussian Processes for Machine Learning*. The MIT Press.

R Core Team (2017). "R: A Language and Environment for Statistical Computing." URL https://www.R-project.org/.

Revolution Analytics, Weston S (2015). "**foreach**: Provides Foreach Looping Construct for R." R package version 1.4.3, URL https://CRAN.R-project.org/package=foreach.

Shi JQ, Cheng Y (2014). "**GPFDA**: Apply Gaussian Process in Functional Data Analysis." R package version 2.2, URL https://CRAN.R-project.org/package=GPFDA.

Stan Development Team (2016a). "**RStan**: The R Interface to Stan." R package version 2.14.1, URL http://mc-stan.org/.

Stan Development Team (2016b). "**rstanarm**: Bayesian Applied Regression Modeling via Stan." R package version 2.13.1, URL http://mc-stan.org/.

Sturtz S, Ligges U, Gelman A (2005). "**R2WinBUGS**: A Package for Running WinBUGS from R." *Journal of Statistical Software*, **12**(3), 1–16. doi:10.18637/jss.v012.i03.

Thodberg HH (1996). "A Review of Bayesian Neural Networks with an Application to near Infrared Spectroscopy." *IEEE Transactions on Neural Networks*, **7**(1), 56–72. doi:10.1109/72.478392.

Williams CKI, Seeger M (2001). "Using the Nyström Method to Speed Up Kernel Machines." In *Advances in Neural Information Processing Systems 13*, pp. 682–688. The MIT Press.

Wood SN (2017). *Generalized Additive Models: An Introduction with R*. 2nd edition. Chapman and Hall/CRC.

Zhu H, Yao F, Zhang HH (2014). "Structured Functional Additive Regression in Reproducing Kernel Hilbert Spaces." *Journal of the Royal Statistical Society B (Statistical Methodology)*, **76**(3), 581–603. doi:10.1111/rssb.12036.

A. I-prior models as GPR models

Consider the following regression model

$$\begin{aligned} y_i &= f(x_i) + \epsilon_i \\ \epsilon_i &\overset{\text{iid}}{\sim} \mathrm{N}(0, \psi^{-1}) \end{aligned} \tag{10}$$

for $i = 1, \ldots, n$ with $f \in \mathcal{F}$ an RKHS with kernel h, and an I-prior on f, i.e.

$$\mathbf{f} = \big(f(x_1), \ldots, f(x_n)\big)^\top \sim \mathrm{N}_n(\mathbf{0}, \psi \mathbf{H}_\eta^2).$$

We note that among the hyperparameters of the kernel function η, there always contains a scale parameter λ and possibly some other hyperparameters ν, such that we can write the kernel function as

$$h_\eta(x, x') = \lambda h_\nu(x, x').$$

Now define the kernel

$$\begin{aligned} \tilde{\lambda} k_\nu(x, x') &= \psi \sum_{i=1}^n \sum_{j=1}^n h_\eta(x, x_i) h_\eta(x', x_j) \\ &= \psi \lambda^2 \sum_{i=1}^n \sum_{j=1}^n h_\nu(x, x_i) h_\nu(x', x_j) \end{aligned}$$

By using the parameterisation $\psi\lambda^2 \mapsto \tilde{\lambda}$, we can treat the above kernel as having scale parameter $\tilde{\lambda}$ and other hyperparameters ν. Since sums of kernels are kernels and products of kernels are kernels, the k defined above is also a valid kernel. We then have

$$\mathbf{f} \sim \mathrm{N}_n(\mathbf{0}, \mathbf{K})$$

which is the familiar Gaussian process prior with $\mathbf{K}_{ij} = \tilde{\lambda} k_\nu(x_i, x_j)$.

B. Remark on hyperparameters and standard errors

In the **iprior** package, estimation of the hyperparameters, in particular the direct method using the L-BFGS algorithm, is done without any bounds constraints on the hyperparameters. This is achieved by using the transformations listed in Table 7. In the package internals, the transformed parameters are always referred to as `theta`, while the untransformed original hyperparameters are reffered to as `param`. This distinction must be noted when supplying initial values for the `iprior()` function, as it is in the transformed parameterisation. A helpful function included in the package is `check_theta()`, which reminds the form of `theta`:

Parameter	Transformation	Remarks
Scale	$\lambda \mapsto \log \lambda$	Only if single scale parameter.
Error precision	$\psi \mapsto \log \psi$	
Hurst index	$\gamma \mapsto \Phi^{-1}(\gamma)$	FBm kernel. Φ is CDF of $\mathrm{N}(0, 1)$.
Length scale	$l \mapsto \log l$	SE kernel.
Offset	$c \mapsto \log c$	Polynomial kernel.

Table 7: Hyperparameters transformations used in the package.

```
R> mod <- kernL(circumference ~ . ^ 2, Orange, kernel = "fbm",
+               est.hurst = TRUE)
R> check_theta(mod)

## theta consists of 4:
## lambda[1], lambda[2], qnorm(hurst[2]), log(psi)
```

When using maximum likelihood, the parameters may be freely transformed without affecting the optimisation procedure. Thus, the estimates of the hyperparameters are obtained by using the respective inverse transformations in Table 7.

The standard errors however must be transformed back using the delta method. The univariate delta method states that if the ML estimate denoted by $\hat{\theta}_n$ (with a dependence on the sample size n) converges in distribution to $\mathrm{N}(\theta, \sigma^2)$, then the transformed ML estimate $g(\hat{\theta}_n)$ converges in distribution to $\mathrm{N}\left(g(\theta), \sigma^2[g'(\theta)]^2\right)$, assuming $g'(\theta)$ exists and is non-zero. Therefore, the transformed standard errors is given by $\hat{\sigma} g'(\hat{\theta}_n)$, where $\hat{\sigma}$ is the standard error for $\hat{\theta}_n$.

C. Remark on the scale parameters

If one or more scale parameters are (estimated to be) negative, then the reproducing kernel for the space of functions in which the regression function lives is not positive definite anymore. It seem arbitrary to restrict the scale parameters to the positive orthant, as the sign of of the scale parameters may be informative, especially when kernels are added and multiplied (e.g. in the varying intercept/slope model). Note that the sign of the scale parameters itself are not identified in the model (this is easily seen when having a single scale parameter in the model since the scale is squared when it appears in the likelihood) but *relative signs of the scale parameters with respect to each other* is.

The space of functions with negative scale parameters is actually called a *reproducing kernel Krein space* (RKKS). Since the building blocks for the models considered are positive definite kernels, we keep speaking about RKHSs in this paper. As with RKHSs, the user does not require any in-depth knowledge of RKKSs in order to perform I-prior modelling.

Affiliation:

Haziq Jamil, Wicher Bergsma
London School of Economics and Political Science
Department of Statistics
Columbia House
Houghton Street
London WC2A 2AE
United Kingdom
E-mail: h.jamil@lse.ac.uk
URL: https://haziqj.ml

E-mail: w.p.bergsma@lse.ac.uk
URL: http://stats.lse.ac.uk/bergsma/