coco : modeling covariate-based covariance functions for nonstataionary Gaussian processes

Federico Blasi[1]
federico.blasi@uzh.ch

Reinhard Furrer[1]
reinhard.furrer@math.uzh.ch

[1]Department of Mathematical Modeling and Machine Learning, University of Zurich

Abstract

This is the vignette for the R package **coco**. The package provides a set of functions for analyzing data that is indexed in space, and can be considered Gaussian. It allows for more flexible representations of the spatial structure compared to classical implementations by modeling different sources of nonstationarity with covariates information available at each index location. The package is aimed to beginner and expert users, providing functions that are intuitive to use with outputs easy to interpret, but also with the possibility of reusing most of the functions to extend the scope and usability of the presented functions. The package offers efficient C++ routines for constructing the associated covariance matrices. It also offers a variation of the general covariance model suitable for very large datasets, which, in combination with the R package `spam`, benefits from inducing sparseness over the covariance matrix to ease the computational needs when analyzing spatial data with thousands of observations. This vignette briefly introduces the modeling approach and provides a brief walk-through analysis with synthetic example data.

Keywords: Gaussian process, covariance function, prediction, very large datasets, R package

1 Introduction

It is often assumed that spatial variable of interest that varies over a continuous domain of interest can be represented by a Gaussian process (GP) with a specific mean function $m(\cdot)$ and covariance function $\mathcal{C}(\cdot,\cdot)$. A key advantage of GP is that these two functions fully determine the distribution at some sampling locations $\{\mathbf{s}_1,\ldots,\mathbf{s}_n\}$, and therefore play a key role in the representation of the process. The choice of $m(\cdot)$ and $\mathcal{C}(\cdot,\cdot)$ is broad and typically depends on the analyst and problem at hand. In its classical setting, the mean function $m(\cdot)$ is assumed to take a regression shape as $m(\cdot;\boldsymbol{\beta}) = \mathbf{X}\boldsymbol{\beta}^{\mathrm{T}}$, where $\boldsymbol{\beta}$ is an unknown vector of parameters of some length m_1, and where the covariance function $\mathcal{C}(\cdot,\cdot)$ is chosen from one of the well-known covariance models, such as the Exponential, Spherical, Wendland, and Matérn families among others, with a parameter vector $\boldsymbol{\psi}$ of length m_2, that shape characteristics of the spatial structure of the GP in a global sense, such as variance (or partial-sill), scale (or range), and smoothness. For more about classical covariance functions, see [Gelfand et al., 2010]. The use of a particular covariance function implicitly states assumptions on how the process covariates at different locations. In its simplest setting, the use of these covariance functions implies that the underlying process is stationary and isotropic, meaning that the correlation structure of the process can be simplified based on the Euclidean distance between locations with a global common variance. Isotropy and Stationarity, while convenient from a modeling and computational efficiency point of view (given the low dimensional parameterization of the associate covariance models), has not aged well, partially after the leap in quality and quantity of spatial data during the last two decades. The oversimplification of the covariance structure often leads to a limited representation of the prediction distributions.

Over the past two decades, a consistent and extensive development of user-friendly R package implementations has been made available to users around the globe to facilitate the analysis of spatial analysis, for both small and very large sample sizes. Among them, we can mention (in descending order by their number of total downloads from CRAN at the time of writing this) `spam` [Furrer et al., 2023] [Furrer and Sain, 2010], [Gerber and Furrer, 2015], [Gerber et al., 2017], which provide efficient Fortran routines for handling, storing, and displaying sparse matrices, fundamental for an efficient implementation of the tapering approach [Furrer et al., 2006]. `fields` package [Douglas Nychka et al., 2021], an extensive package involving curve, surface, and function fitting with an emphasis on splines,

spatial data, and spatial statistics. `spatstat` [Baddeley et al., 2015] [Baddeley et al., 2013] [Baddeley and Turner, 2005], which focus on functions for analysing spatial point patterns in 2D, 3D, and space-time. `gstat` [Pebesma, 2004] [Gräler et al., 2016], offering an extensive set of functions for variogram modeling, prediction, and visualization, with support for sf [Pebesma and Bivand, 2023a][Pebesma, 2018], and stars [Pebesma and Bivand, 2023b] package. `convoSPAT` [Risser and Calder, 2017] offering functions for modeling convolution-based nonstationary Gaussian process models to point-referenced spatial data. `GpGp` providing functions for fitting and doing predictions with Gaussian process models using Vecchia's (1988) approximation [Vecchia, 1988] [Guinness, 2018]. `varycoef` [Dambon et al., 2021], implementing maximum likelihood estimation method for estimation and prediction of Gaussian process-based spatially varying coefficient (SVC) models. `GeoModels` [Bevilacqua and Gaetan, 2015] [Bevilacqua et al., 2016] [Vallejos et al., 2020], providing functions for Gaussian and Non-Gaussian spatial and spatio-temporal data analysis, based on standard likelihood and which provides functions for the analysis of skewed Gaussian processes, with composite likelihood functions, a likelihood approximation method called weighted composite likelihood based on pairs. `spmodel` [Dumelle et al., 2023], providing functions for fitting, summarizing, and predictions of a variety of spatial statistical models applied to point-reference and area (lattice) data. To the best of our knowledge, none of these packages offers user-friendly implementations of closed-form flexible modular covariance functions for GP's as, for example, in [**?**].

The `coco` package provides a user-friendly and efficient way of representing single-realization and multiple-realizations of Gaussian processes with a covariate-based spatially varying covariance function. This means that we can adjust the local spatial structure of the process based on features at the spatial index, such as elevation, ridges, lakes, cloud coverage, etc. This covariance function allows us to consider different source of nonstationarity of the spatial structure in a modular way. Under the hood, the computationally intensive routines benefit from efficient storage of data structures, parallel computing, and the efficiency of C++. At the very core of the `coco` package lies the `coco` class, which enables us to easily perform tasks such as optimization of the model's parameters, visualization, prediction, as well as conditional simulation. With `coco`, we offer a toolkit that offers modular modeling of a Gaussian process that enables the user to consider seven aspects that shape the mean and the spatial structure.

This vignette introduces the key features of the R package **coco**. The remainder of this vignette is organized as follows. Section 2 introduces the basics of Gaussian-based spatial analysis. Section 3 introduces the package functionalities through a data analysis example. Finally, Section 4 concludes with a short discussion.

2 Basics of spatial statistics

In this section, we introduce the very basics of spatial analysis. Be aware that this addresses only key aspects. The user show can refer to [Gelfand et al., 2010] for a proper introduction to maximum likelihood spatial analysis.

When analyzing a Gaussian process $Z(\cdot)$ at a set of locations $\{\mathbf{s}_1, \ldots, \mathbf{s}_n\} \in \mathcal{D}$, being $\mathcal{D} \subset \mathcal{R}^2$, it is a common practice to assume that $Z(\cdot)$ can be decomposed as

$$Z(\mathbf{s}) = m(\mathbf{s}) + Y(\mathbf{s}) + \epsilon(\mathbf{s}) \ , \quad \mathbf{s} \in \mathcal{D},$$

where $m(\mathbf{s})$ is the mean function, $Y(\mathbf{s})$ is a zero-mean spatial process where $\mathcal{C}(Y(\mathbf{s}_i), Y(\mathbf{s}_j)) = \mathcal{C}(\cdot, \cdot; \boldsymbol{\psi})$, being $\mathcal{C}(\cdot, \cdot; \boldsymbol{\psi})$ a valid covariance function with parameter vector $\boldsymbol{\psi}$, and where $\epsilon(\mathbf{s})$ is an iid zero-mean white-noise process. At the sampling locations $\{\mathbf{s}_1, \ldots, \mathbf{s}_n\} \in \mathcal{D}$, the process Z follows a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}$ of length n and the $n \times n$ positive-definite covariance matrix $\boldsymbol{\Sigma}_Z$, with elements $[\boldsymbol{\Sigma}_Z]_{i,j} = \mathcal{C}(\mathbf{s}_i, \mathbf{s}_j; \boldsymbol{\psi})$. Considering $\mathbf{z}$ as the available sample of length n, optimization of the model's parameter relies on minimizing

$$-2l(\boldsymbol{\psi}, \boldsymbol{\beta}) = n \log(2\pi) + \log\det \boldsymbol{\Sigma}_{\mathrm{Z}} + (\mathbf{z} - \boldsymbol{\mu})^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{Z}}^{-1} (\mathbf{z} - \boldsymbol{\mu}).$$

Moreover, considering that for a specific fixed $\boldsymbol{\psi}_0$ there is a closed-form solution for $\boldsymbol{\beta}$, the optimization can be carried out only for $\boldsymbol{\psi}$ as

$$-2l_p(\boldsymbol{\psi}) = n \log(2\pi) + \log\det \boldsymbol{\Sigma}_{\mathrm{Z}}(\boldsymbol{\psi}_0) + \mathbf{z}^{\mathrm{T}} \mathbf{P}(\psi) \mathbf{z}, \tag{1}$$

where $\boldsymbol{\Sigma}_{\mathrm{Z}}(\boldsymbol{\psi})$ is the covariance matrix based on $\boldsymbol{\psi}$, and $\mathbf{P}(\cdot) = \boldsymbol{\Sigma}_{\mathrm{Z}}^{-1}(\cdot) - \boldsymbol{\Sigma}_{\mathrm{Z}}^{-1}(\cdot)\mathbf{X}(\mathbf{X}'\boldsymbol{\Sigma}_{\mathrm{Z}}^{-1}(\cdot)\mathbf{X})^{-1}\mathbf{X}'\boldsymbol{\Sigma}_{\mathrm{Z}}{}^{-1}(\cdot)$. Then, $\boldsymbol{\beta}$ is retrieved as

$$\boldsymbol{\beta}(\boldsymbol{\psi}_0) = (\mathbf{X}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{Z}}(\boldsymbol{\psi}_0)^{-1} \mathbf{X})^{-1} \mathbf{X}^{\mathrm{T}} \boldsymbol{\Sigma}_{\mathrm{Z}}(\boldsymbol{\psi}_0)^{-1} \mathbf{z}. \tag{2}$$

Equation (1) can be optimized by numerical algorithms as well. Finally, $\hat{\boldsymbol{\beta}}_{\text{ML}} = \hat{\boldsymbol{\beta}}(\hat{\boldsymbol{\psi}}_{\text{ML}})$.

Predictions of the process Z at new locations $\{\mathbf{s}_1^*, \ldots, \mathbf{s}_k^*\}$ based on the adjusted model are computed as the conditional distribution of multivariate Gaussian distributions, defined as

$$\mathbf{Z}_{\text{pred}}|\mathbf{Z} = \mathbf{z} \sim \mathcal{N}_k(\mathbf{X}^p\boldsymbol{\beta} + \boldsymbol{\Sigma}_{\text{PZ}}\Sigma_{\text{Z}}^{-1}(\mathbf{z} - \mathbf{X}\boldsymbol{\beta}), \boldsymbol{\Sigma}_{\text{P}} - \boldsymbol{\Sigma}_{\text{PZ}}\boldsymbol{\Sigma}_{\text{Z}}^{-1}\boldsymbol{\Sigma}_{\text{PZ}}^{\text{T}}), \tag{3}$$

where $\boldsymbol{\Sigma}_{\text{PZ}}$ is a matrix of dimension $k \times n$ with elements $[\boldsymbol{\Sigma}_{\text{PZ}}]_{i,j} = \mathcal{C}(\mathbf{s}_{0_i}, \mathbf{s}_j)$, i.e., a matrix that encodes how the process at the unseen locations covariates with the process at the observed locations. Where $\boldsymbol{\Sigma}_{\text{P}}$ is a matrix $k \times k$ with elements $\mathcal{C}(\mathbf{s}_i^*, \mathbf{s}_j^*; \boldsymbol{\psi})$. As expected, we see that the conditional distribution relies heavily on the covariance function $\mathcal{C}(\cdot, \cdot)$, with the assumptions of stationarity and isotropy limiting prediction capabilities.

Classical covariance models usually represent the spatial structure globally. For example, the Matérn covariance model [Furrer, 2016] is defined as

$$\mathcal{C}(h; \boldsymbol{\psi}) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{8\nu}\frac{h}{\gamma}\right)^{\nu} \mathcal{K}_\nu\left(\sqrt{8\nu}\frac{h}{\gamma}\right), \tag{4}$$

where h is the Euclidean distance between two spatial locations $\mathbf{s}_i$ and $\mathbf{s}_j$, $\sigma > 0, \gamma > 0, \nu > 0, \mathcal{K}_\nu(\cdot)$ is the modified Bessel function of the second kind of order ν [Abramowitz and Stegun, 1970], $\Gamma(\cdot)$ is the Gamma function, and $\boldsymbol{\psi} = (\sigma^2, \gamma, \nu)^{\text{T}}$.

A more flexible covariance model is offered in `coco`, which is a parametric version of the kernel introduced in [Paciorek and Schervish, 2006], for which convenient decisions for each of the aspects has been taken. The covariance function takes the form of

$$\mathcal{C}_{\text{R}}(\mathbf{s}_i, \mathbf{s}_j; \mathbf{x}_i, \mathbf{x}_j, \boldsymbol{\psi}) = \sigma_i^{\mathbf{x}}\sigma_j^{\mathbf{x}}|\boldsymbol{\Sigma}_i^{\mathbf{x}}|^{1/4}|\boldsymbol{\Sigma}_j^{\mathbf{x}}|^{1/4} \left|\frac{\boldsymbol{\Sigma}_i^{\mathbf{x}} + \boldsymbol{\Sigma}_j^{\mathbf{x}}}{2}\right|^{-\frac{1}{2}} \mathcal{M}_{\nu_{ij}^{\mathbf{x}}}\left(\sqrt{Q_{ij}}\right) + \text{I}(\mathbf{s}_i = \mathbf{s}_j)\sigma_{\epsilon,i}^X\sigma_{\epsilon,j}^X, \tag{5}$$

where $\sigma_i^{\mathbf{x}}$, $\boldsymbol{\Sigma}_i^{\mathbf{x}}$, and $\nu_{ij}^{\mathbf{x}}$ are parametric functions related to specific characteristics of the spatial structure, σ_ϵ^2 is the parameter associated to the white noise process $\epsilon(\cdot)$, and where $\text{I}(A)$ is an indicative function when the event A is present. In particular, we define $\nu_{ij}^{\mathbf{x}} = \sqrt{\nu_i}\sqrt{\nu_j}$. Finally, Q_{ij} is a semi-distance metric [Schoenberg, 1938] of the shape of

$$Q_{ij} = (\mathbf{s}_i - \mathbf{s}_j)^{\text{T}}\left(\frac{\boldsymbol{\Sigma}_i + \boldsymbol{\Sigma}_j}{2}\right)^{-1}(\mathbf{s}_i - \mathbf{s}_j), \quad \mathbf{s}_i, \mathbf{s}_j \in \mathcal{D}, \tag{6}$$

whereas Q_{ij} reduces to the Euclidean distance when $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_j = \gamma\mathbf{I}$, with $\gamma > 0$. We present a brief summary in Table 1, describing the different characteristics of the model and how these relates to (5). For a detailed description of the model, see **??**.

Aspect	Related to	Description	Model
mean	μ	mean function	$\mathbf{X}_1\boldsymbol{\beta}$
std.dev	σ^X	marginal standard deviation	$\exp(0.5\mathbf{X}_2\boldsymbol{\alpha})$
scale	$\boldsymbol{\Sigma}^X$	local scale	$\exp(\mathbf{X}_3\boldsymbol{\theta}_1)$
aniso	$\boldsymbol{\Sigma}^X$	local geometric anisotropy	*scale* $\exp(\mathbf{X}_4\boldsymbol{\theta}_2)$
tilt	$\boldsymbol{\Sigma}^X$	(restricted) local tilt	$\cos(\text{logit}(\mathbf{X}_5\boldsymbol{\theta}_3))$
smooth	ν^X	local smoothness	$(\nu_u - \nu_l)/(1 + \exp(-\mathbf{X}_6\boldsymbol{\phi}) + \nu_l$
nugget	σ_ϵ^X	local micro-scale variability	$\exp(\mathbf{X}_7\zeta)$

Table 1: Summary of the available source of nonstationarity that can be modeled with the **coco** package.

where $\boldsymbol{\beta}, \boldsymbol{\alpha}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \boldsymbol{\phi}$ and ζ are unknown parameters related to each of the aspects, ν_l ν_u are the lower and upper considered limits for the spatially varying smoothness aspect, and where $\mathbf{X}_i$ means that the associated design matrix is specific for the model.

3 Introducing the coco package via data analysis

Here, we illustrate a simple example of the key functionalities of `coco` aimed to model, visualize, predict, and simulate GP's process with flexible covariance models. The Vignette introduces the two available

models offered in **coco**: a GP with dense covariance structure, which implements a covariance structure that can adopt up to seven spatially varying aspects, and one GP model aimed for very large data sets, allowing up to four spatially varying aspects. The **coco** R package is available on Github and can be installed and loaded as follows

```
library("devtools")
install_github("blasif/coco")
library("coco")
```

We then introduce both models with a short data analysis of synthetic datasets included in the package, one for each considered model.

3.1 Modeling GP's with flexible dense covariance functions

We showcase the key functions related to the `dense` model via data analysis of the `holes` synthetic dataset. We load the `holes` object by

```
data("holes", package = "coco")
```

which is an object of type `list` containing two dataset, the usual `training` and `test` datasets. Each of these datasets contains five variables, two related to the spatial indexing (x, y), two covariates (`cov.one` and `cov.two`), and the process response (`z`). We have 6000 and 540 observations in the training and test dataset, respectively. In Figure 1 we present an overview of the training dataset.

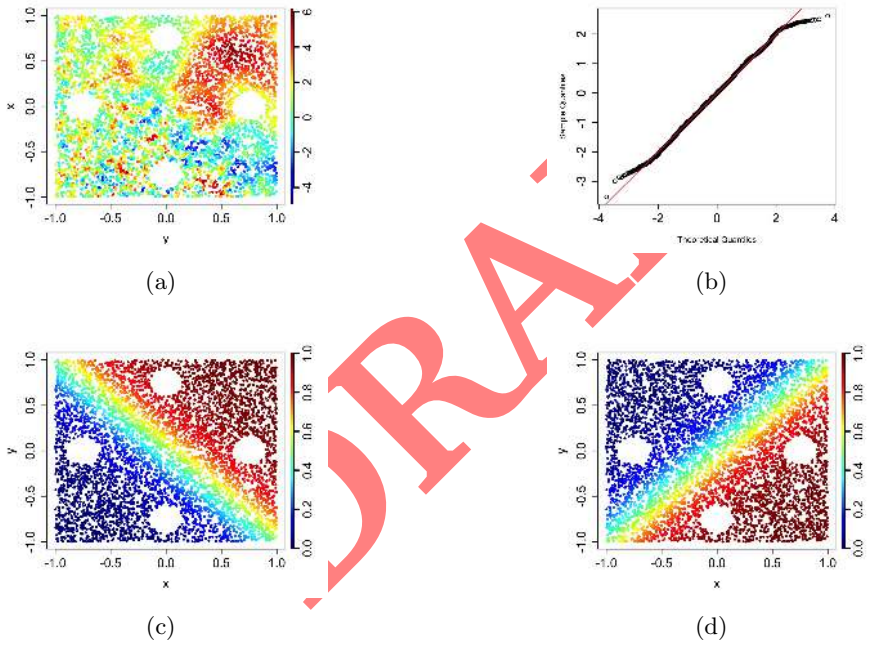
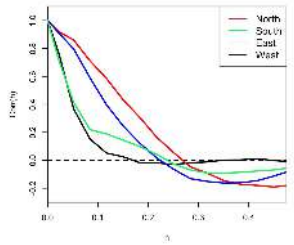


Figure 1: (a) spatial distribution of the sample `z`. (b) Normal Q-Q of the scaled sample `z`. (c) and (d) present the spatial distribution of the covariates `cov.one` and `cov.two`, respectively. Four *holes* are present in the training dataset, which corresponds to locations of the test dataset. The challenge lies on providing predictions to these locations with good uncertainty quantification.

A first inspection of panel 1a reveals a pattern related to the scale of the process. We see that for large values of `cov.one` in Panel 1c, there appears to be a higher correlation between z than for lower values of `cov.one`. This is seen mainly due to the slow transitions between colors leading to larger 'clusters' when compared to the lower half. On a second level, and more subtle now, higher values of `cov.two` in Panel 1d, seem to be associated with a greater spread of process values. This can be seen particularly in the upper left region of the domain when compared to the bottom right region. To complete the inspection of Figure 1, we see that the scaled distribution of **z** (after scaling) in Panel 1b follows approximately a standard Gaussian distribution.

For further inspection, we define four regions naturally defined by placing a cross centered at the middle of the domain. We call each of these regions by the main four cardinal points. We compute the



(a) Empirical correlograms

Region	$\hat{\sigma}^2$
North	1.32
West	1.20
South	1.82
East	2.14

(b) Empirical variances

Figure 2: Empirical correlograms and variances for the training dataset of the `holes` dataset.

empirical correlograms (via the `gstat` package) and variances for each of the four defined regions, as shown in Figure 2.

By inspecting the correlograms for each of the mentioned regions at Panel 2a, we see how the correlation decays differently for the North & East region on one side, whereas in comparison to the South & West regions, the correlations decays slower. This supports the idea that the process exhibits two different spatial structures with considerably different scales. On the right side of Figure 2, we have empirical values of the variances for each region, with the South and East regions exhibiting sensible higher values.

This brief inspection of the `holes` dataset suggest models for shaping the spatially-varying aspects for the *std.dev* and *scale*. Based on this, we will consider a model for the local variance as a function of `cov.one` and a model for the scale as a function of `cov.two` and `cov.one`. Yet, to highlight some package functionalities, we will also consider `cov.two` for the *scale* aspect. Finally, we will consider the remaining aspects fixed to their true values to simplify the analysis.

Given that we have some intuition on the models for *alpha* and *scale*, the next step is to create the `coco` package to create an `coco` object. Given that the main functions of the package are built around this object, it is easier to build this first and perform different tasks related to spatial analysis. Advanced users of R have also the option of using the accessory functions of the package to extend the functionalities of the package, which are called `getFunction()`, being `Function` a specific task required to run different functions. The coco object returns an *S*4 object storing information about the type of coco model as well as information related to the dataset, locations, etc. One key argument is `model.list`, which specifies the different spatially varying aspects (and mean). For this particular example, we define it as

```
model.list <- list("mean"     = formula(~ 1),
                   "std.dev"  = formula(~ 1 + cov.one + cov.two),
                   "scale"    = formula(~ 1 + cov.one + cov.two),
                   "aniso"    = 0,
                   "tilt"     = 0,
                   "smooth"   = 3/2,
                   "nugget"   = -Inf)
```

This establishes seven models, one for each available aspect. In particular, we consider a global mean, and we associate both *std.dev* and *scale* aspects to the covariates `cov.one` and `cov.two`, plus an intercept. This defines 3 parameters for each model. As mentioned above, the remaining aspects are considered fixed to their true parameters. `aniso` and the `tilt`, are set to 0, *smooth* aspects set to 3/2 directly specify the smoothness of the process; and `nugget` is set to `-Inf`, meaning absence of this aspect as well (given its log parameterization). We take a 1000 random sample from the *holes* dataset and we store it in an object called holes. We then proceed to create and store the `coco` object by

```
coco_object <- coco(type       = "dense",
                    model.list = model.list,
                    locs       = as.matrix(holes[[1]][, 1:2]),
                    z          = holes[[1]]$z,
                    data       = holes[[1]])
```

Let's inspect each of the arguments of the coco function. Argument `type = "dense"` means that we are modeling a Gaussian process with a dense covariance matrix; arguments `locs`, `z`, and `data`, relates to information about the locations, response variable, and dataset with covariates, respectively. It is important that `locs` is a matrix type to ensure the arguments passed to the `C++` functions work as expected. Finally, we must also provide the mentioned model.list in the `model.list` argument. Under the hood, efficient handling of the design matrix for each of the models are carried out to reduce memory

consumption.

As a side comment, if we decide to model the *smooth* aspect, we need to provide additional information specifying the boundaries of the *smooth* aspect. This information is passed as `smooth_limits` item of a list named `info`, specifying the lower and upper bounds of the smoothness aspect, for example, by

```
coco(..., info = list("smooth_limits" = c(0.1, 1)))
```

This means that the specified aspect will be constrained to $(0.1, 1)$. The rationale behind constraining the range of variability of the smoothness aspect is a pragmatic approach to promote numerical stability in the estimation procedures.

Having built our coco object, fitting the model is straightforward. The optimization of the 7 parameters in our model is done with the function

```
coco_object <- cocoOptim(coco_object, ncores = 4)
```

the `cocoOptim` function returns the updated `coco` object with information on the optimization, so we save it in the same object to save storage. The parameters are optimized via the `L-BFGS-B` numerical optimizer [Byrd et al., 1995], and in parallel thanks to the `OptimParalell` R package [Gerber and Furrer, 2019]. Extra arguments of `cocoOptim` such as `boundaries` and `optim.control` are optional and can be investigated in the help section. The `cocoOptim` function provides defaults upper and lower bounds for each of the parameters as well as a suggested optim.control list for the optimization routine.

By default, the parameters to be optimized differ from the specified ones to reduce the correlation between the parameters. This aids during the optimization routine and promotes estimations with lower uncertainties and fewer iterations of the optimization routine. These transformed parameters are stored in the `coco.object@output$pars`. We can recover the intended parameters for *std.dev* and *scale* with the function `getEstims(coco_object)[2:3]` (by default it outputs 7 lists, one for each aspect), which returns

```
$std.dev
[1]  0.771793213 0.009118481 0.576093069
$scale
[1] -1.6295528  0.9908970  0.0511023
```

Each element of the vectors within each of the aspects relates to the parameters in the order of how they were specified in the model.list starting from the first aspect. We see that the estimates for the parameter of `cov.one` for *std.dev*, and `cov.two` for the *scale* are very close to 0. Assuming that the maximum likelihood estimator follows asymptotically a Gaussian distribution we can draw approximate confidence intervals for the parameters. To do so, we need the Hessian matrix, which can be approximated via the function `getHessian`, as

```
Hessian_coco <- getHessian(coco_object, ncores = 4)
```

notice that we can specify the number of cores, meaning that the routine that computes an approximate of the Hessian matrix is run in parallel, saving a considerable amount of time when the sample size and number of parameters is large.

We provide the fitted coco object and the number of cores we want to use to compute the approximate Hessian matrix. Once we have this information, we can retrieve approximate confidence intervals of the parameters in the model with the function `getCIs`.

```
getCIs(coco_object, inv.hess = solve(Hessian_coco), alpha = 0.05)
```

Beyond the fitted `coco` object, we have to provide the inverse of the Hessian matrix, and the confidence level α.

```
              2.5 %      97.5 %
mean1     0.84854447  1.2395687
std.dev1  0.52747247  1.0161140
std.dev2 -0.21336357  0.2316005
std.dev3  0.41585700  0.7363291
scale1   -1.73727097 -1.5218347
scale2    0.89830263  1.0834914
scale3   -0.02373882  0.1259434
```

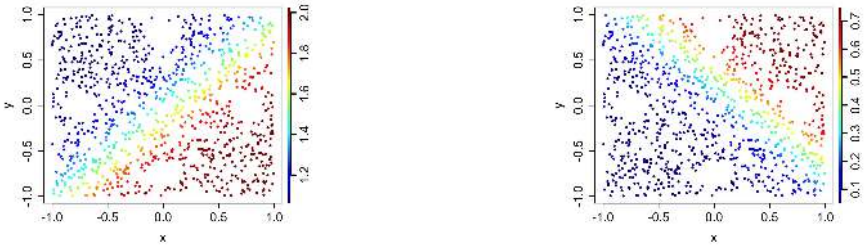
For `alpha2` and `scale3`, the 0 is included in the CIs. This would be analog to performing a two-tail univariate hypothesis testing over the parameter with a probability type I error of α. This gives us some intuition on how to reduce the model. We drop the terms `cov.one` for *std.dev* and `cov.two` for *scale*, and fit it again storing it in the same object name. The CIs for the final model are

```
             2.5 %      97.5 %
mean1     0.8318155   1.2378472
std.dev1  0.6199807   0.9255969
std.dev2  0.3957672   0.5740780
scale1   -1.7070386  -1.5461666
scale3    0.9331792   1.0371132
```

We can compute the associated BIC values with the function `getBIC()`, leading to a reduction of the BIC for the simpler model (1351.672 vs. 1363.701). A quick overview of the adjusted models are available via `plot(coco.object)`, but we will show how this can be done manually for the two aspects of interest

```
spat_effects <- getSpateffects(coco_object)
par(par = c(1, 2))
quilt.plot(coco_object@locs, spat_effects$sd)
quilt.plot(coco_object@locs, spat_effects$eff_x)
```

In Figure 3 we see the output of this code (up to visual enhancements). The spatial distribution of these two aspects is mainly given by the underlying covariates we specified in the `model.list`. For the marginal standard deviation, we see that it ranges from 1 to 2, 2 times as large, while for the effective scale, the effect of allowing a spatially varying aspect is stronger, ranging from 0.1 to 0.7, 7 times as large.



(a) sd (b) eff. scale

Figure 3: adjusted effects of the reduced model.

Once we have adjusted the parameters of our model, we can easily use the fitted coco object to predict the process values at the *holes*, for which we require (and have) information of `cov.one` and `cov.two` at every prediction location. To compare the performance of our model, we also estimate a classical Matérn model. Since a classical Matérn model in 4 is a simplification of the regression model, we can use the `coco` package to fit this model as well. This is done with the following `model.list`

```
model.list <- list("mean"    = formula(~ 1),
                   "std.dev" = formula(~ 1),
                   "scale"   = formula(~ 1),
                   "aniso"   = 0,
                   "tilt"    = 0,
                   "smooth"  = 3/2,
                   "nugget"  = -Inf)
```

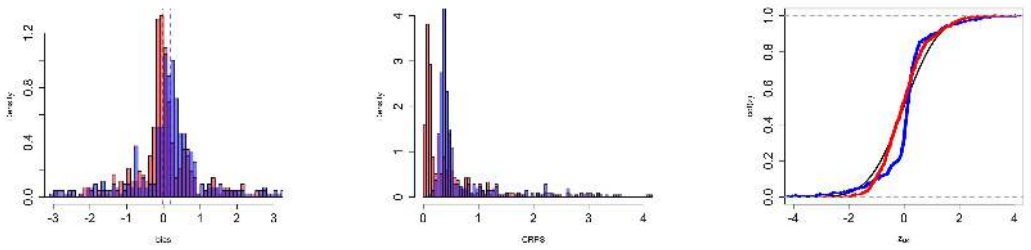
This means that we are only allowing a global mean, and globals parameter for *std.dev* and *scale*. We call the classical optimized model `Optim.classic`. The estimated parameters are of 1.62 for the standard deviation, and a scale of 0.12, values within the range of variability of the regression model.

To perform predictions, we call the function `cocoPredict()` by

```
Pred.ours <- cocoPredict(coco_object,
                         newdataset = holes[[2]],
                         newlocs    = as.matrix(holes[[2]][,1:2]),
                         type       = "pred")
```

In addition to specifying the fitted model, we need to provide information about the prediction covariates and their locations (`newdataset` and `newlocs`, respectively). Metrics such as the CRPS [Gneiting and Raftery, 2007] can be computed with `getCRPS` and `getLogrank` with the information provided by `cocoPredict`, as

```
CRPS_regre <- getCRPS(z.pred = holes[[2]]$z,
                      mean.pred = Pred.ours$trend + Pred.ours$mean,
                      sd.pred = Pred.ours$sd.pred)
```



(a) bias (b) CRPS (c) empirical cdf

Figure 4: metrics related to prediction under both models. In blue classical model. In red, spatially-varying model.

In Figure 4 we present different aspects of the prediction capabilities of both models. In Panel 4a the bias for both models is shown, with small differences. For this specific dataset, where the regression model makes a difference is quantifying the uncertainty of the predictions, which led to a reduction of the CRPS values (Panel 4b), with a reduction of 28% of the mean CRPS, when compared to the classical model. This better representation of the variance of the prediction leads to honoring better the assumption that the conditional distribution is Gaussian, as we can appreciate in Panel 4c, where the typical S shape of the CDF for the classical model is evident, meaning that in general overshoots the uncertainty, translating this into over-smoothing of the predictions. For further inspection, we desegregate the CDFs of both models by region, as shown in Figure 5.

(a) (b)

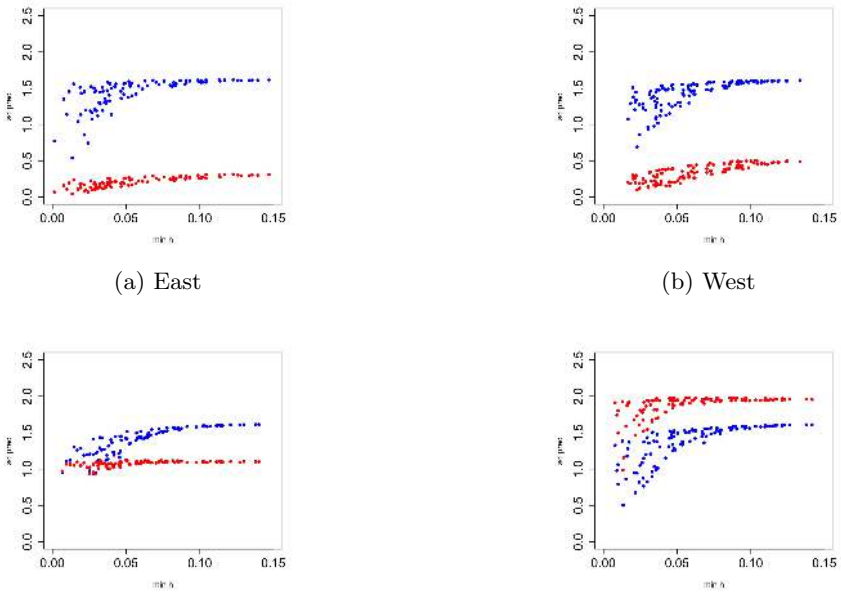
Figure 5: empirical cdf's by regions. In red, spatially-varying model. In blue, classical model.

We see that, when desegregated by region, the assumption stated in Equation (3) is preserved better when employing the spatially-varying model, than when employing the classical model, where it seems that the aggregation of many regions with different spatial structures led to an approximation of this assumption. We can inspect this further in Figure 6, where we inspect the associated standard deviation of the prediction with the minimum distance of the prediction location to a training location.

In Figure 6 we see a key difference between models. While the classical model is only able to provide one unique variance allocation as the minimum distance between the prediction location and the training locations increases, the regression model can adjust to the local spatial structures. For the East a West regions, we have a local spatial structure with almost 4 times larger scale than its classical counterpart, translating this to more information and therefore less uncertainty. For those scenarios with lower scale, there is also a clear contrast between the classical model. For the West region, we have lower local variance in comparison to the classical model, leading to an overall smaller variance, but for the South region, which is a low-scale, high-variance location, the regression model resolves by allocating an overall higher uncertainty in comparison to the classical model.

We finish the analysis of the predictions by inspecting the filled-up maps with the true predictions as well as the true values in Figure 7. As expected, there is little to no difference when comparing the two models with respect to point predictions. For this particular dataset, where there is no spatial trend, we can (partially) recover the process information if the prediction location is within the range of the (effective) correlation between the training locations. The advantage of more flexible models over classical models diminishes as the distance between the prediction location and the training prediction decreases (or increases) relative to the estimated scale of the spatial structure.

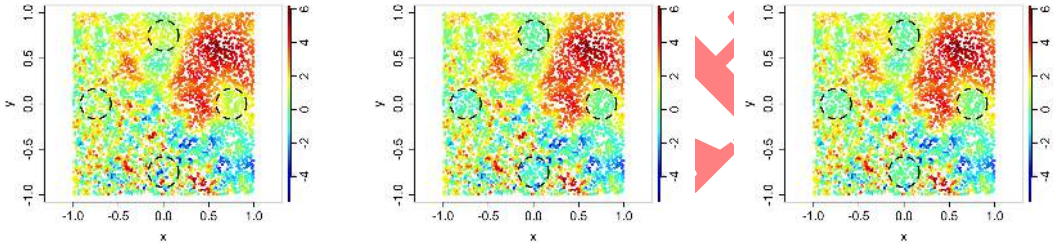
However, in some scenarios, we can achieve better point predictions. In particular, if for the same data



(a) East (b) West

(c) North (d) South

Figure 6: standard deviation of the conditional distribution $Z_{pred}|\mathbf{Z}$ vs minimum distance between the prediction location and the sample data, for the different regions. In red, spatially-varying model. In blue, classical model.



(a) (b) (c)

Figure 7: (a) shows true data, (b) training data + predictions at the *holes* based on the classical model, while (c) based on the regression model.

set, we have regions of relatively high and low scale, and here the prediction locations are at a distance from the training locations, close to the effective point scale, and where the values to be predicted are far from the trend, we may observe a better representation by the regression model than by the classical model.

To finalize this section related to the dense model, our package provides as well an integrated function to simulate random fields (marginal and conditional) with a spatially varying covariance structure. We can simulate fields by providing a coco object. We `rbind` both training and test datasets into an object called `df_full`.

```
list_formulas <- list("mean"   = as.formula(" ~ 1"),
                      "alpha"  = as.formula(" ~ 1 + cov_y"),
                      "scale"  = as.formula(" ~ 1 + cov_x"),
                      "aniso"  = 0,
                      "tilt"   = 0,
                      "smooth" = 3/2,
                      "nugget" = -Inf)

coco_sim <- coco(type = 'dense', data = df_full,
                 locs = as.matrix(df_full[,1:2]),
                 z = rep(NA,length(dim(df_full)[1])),
                 model.list = list_formulas)
```

```
)

pars_to_feed <- unlist(getEstims(optim_second))[unlist(
                  getEstims(optim_second)) != 0][-5]

sim_field <- cocoSim(coco.object = coco_sim,
                     pars = pars_to_feed,
                     n = 3, standardize = TRUE,
                     type = 'classic',
                     seed = 202020)
```

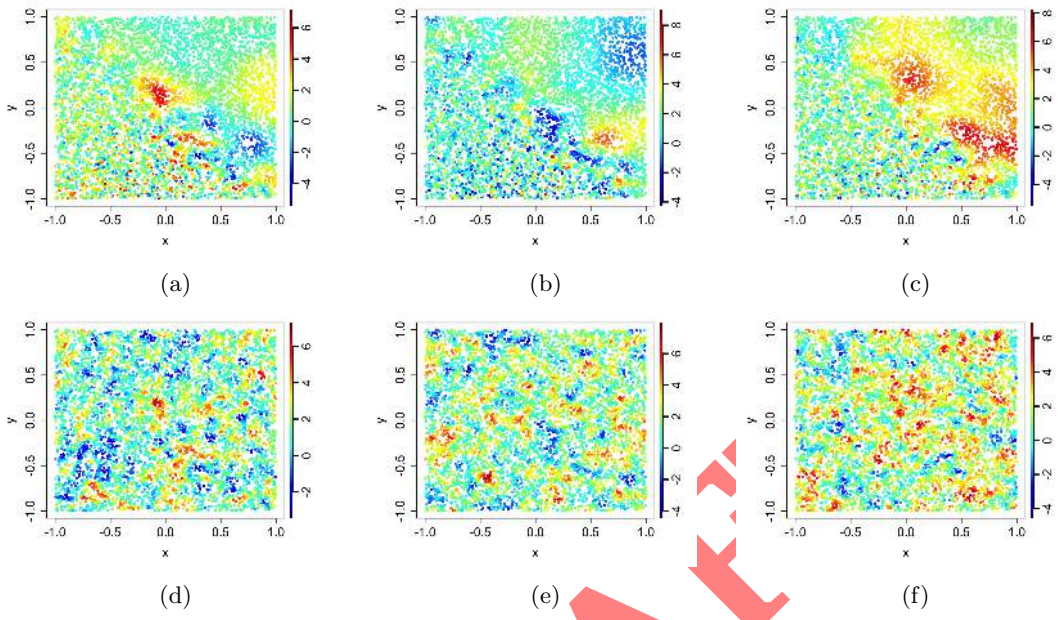


Figure 8: Marginal simulation based on estimated parameters for flexible model (top row) and classical model (bottom row). Seeds have been set common by column.

In Figure 8 we present six marginal simulations, three for each model. Again, here we can see the regions given by the different scales for the simulations from the regression model, while under the classical model, this aspect of the training data is lost. The `coco` package also allows for conditional simulation. We perform conditional simulation over the holes in Figure 9 we present the conditional simulations.

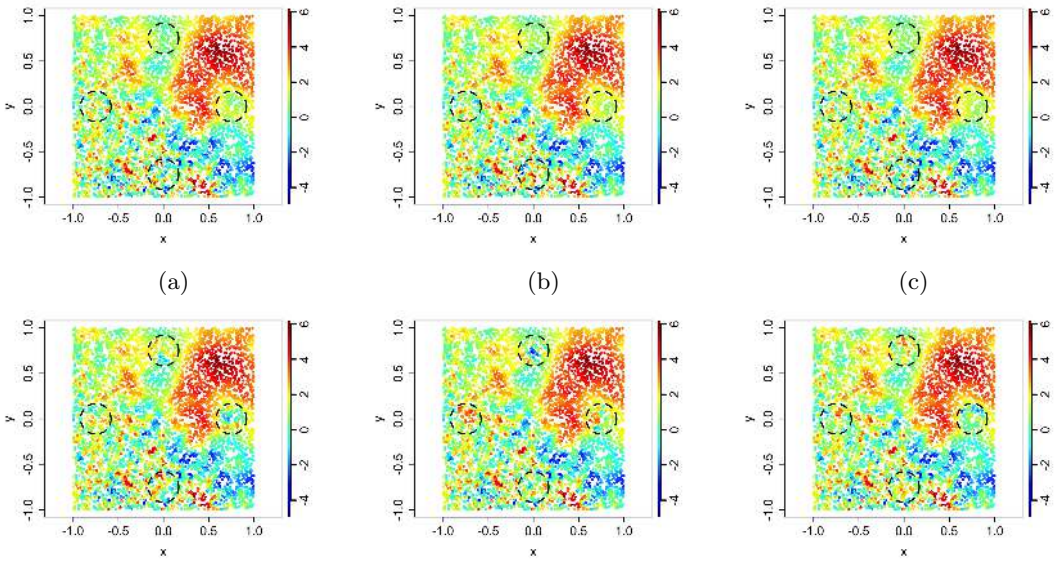
```
pars_to_feed <- unlist(getEstims(optim_second))
[unlist(getEstims(optim_second)) != 0][-5]

# with fitted object only
sim_field_cond <- cocoSim(coco.object = optim_second,
                     pars = pars_to_feed,
                     n = 3, standardize = TRUE,
                     type = 'classic',
                     cond.info = list('newdataset' = df_test,
                                      'newlocs' = as.matrix(df_test[,1:2])),
                     sim.type = 'cond',
                     seed = 202020)
```

We see how the conditional simulations honor better the true data.

3.2 Modeling GP's with flexible sparse covariance functions

Now, we consider the scenario where the sample size exceeds our computational resources (usually memory) to run the key functions of `coco`. In these scenarios **one** of many approaches to reduce computational requirements (especially when the aim is over the predictions) is the tapering approach [Furrer et al., 2006], for which we induce zeroes in the covariance matrix to make it sparse and then through computational efficient algorithms that benefit from these types of structures enhance the speed and memory consumption required for the key functions of the package. Our package relies on the efficient routines for sparse matrices implemented in `spam`.



(a) (b) (c)

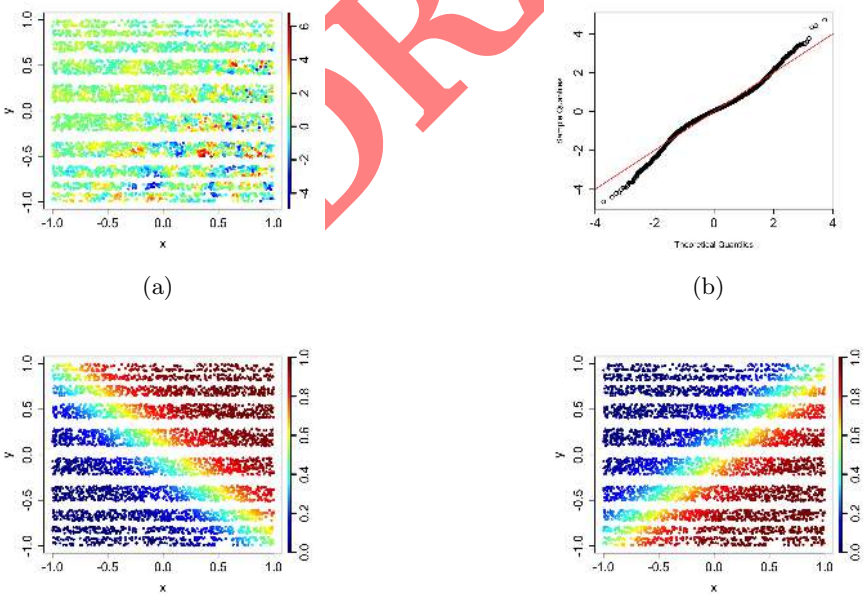
(d) (e) (f)

Figure 9: Conditional simulation based on estimated parameters for flexible model (top row) and classical model (bottom row). Seeds have been set common by column.

At the level of usage, defining, optimizing, and predicting, are almost identical as shown before. To reduce the extent of this Vignette, we focus on the difference only. For this part, we use a second synthetic dataset which can load as:

```
data("stripes", package = "coco")
```

This object contains two datasets again, one for training and one for prediction. The dimension of the training dataset is of 10000 observations and 7000 for prediction. We take a random sample size of 5000 observations.



(a) (b)

(c) (d)

Figure 10: A set of figures related to the training dataset of the `stripes` dataset. (a) spatial distribution of the response `z`. (b) Normal Q-Q of the scaled response `z`. (c) and (d) present the spatial distribution of the available covariates (`cov.one` and `cov.two` respectively). The *stripes* present in the first plot are regions to predict.

We allow spatially varying aspects for those aspects that influence prediction under very large datasets, such as the variance, local scale, smoothness, and nugget effects (and a global parameter for the scale parameter). Therefore, our new `model.list` will only specify aspects for the mean, std.dev, scale, smoothness, and nugget. Others aspects are set to zero. We define a model with both cov.one and cov.two for std.dev and smoothness and proceed to define the coco object, and the product between cov.one and cov.two for the spatially-varying scale. Then, we define the coco object as

```
coco_object <- coco(type = 'sparse',
                    model.list = list_formulas,
                    locs = as.matrix(stripes[[1]][,1:2]),
                    z = stripes[[1]]$z,
                    data = stripes[[1]],
                    info = list('smooth_limits' = c(0.5, 1.5),
                                'taper' = spam::cov.wend1,
                                'delta' = 0.25))
```

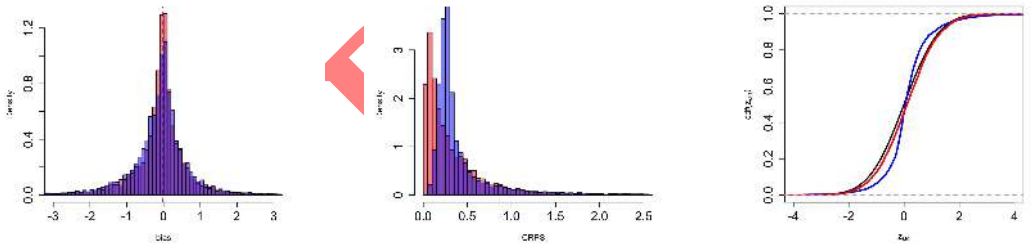
A few differences with respect to the dense coco object: beyond specifying the type of the coco object is `sparse`, we have to provide information regarding the taper function (that must be from the `spam` package), as well as the taper range. For a given taper range δ, the resulting covariance between two locations will be zero for $Q_{ij} > \delta$. Moreover, we have to provide information regarding the `smooth.limits` as mentioned before.

Given that this object is aimed at very large datasets , the modeling of the spatial structure should be focusing as well only on those aspects that are important for very large datasets such as sill, scale, nugget, and smoothness.

By creating this object, we induce sparseness over almost 95% of the matrix, meaning that the computation is done with the remaining data points. We can ask for the fitted parameters by the same function `getEstims(optim_taper)[c(2,3,6)]`:

```
$std.dev
[1] -0.07304400  0.06237891  0.98408386  0.00000000
$scale
[1] -1.4465915  0.0000000  0.0000000 -0.9626796
$smooth
[1]  0.033847097  2.174419388 -0.009839086  0.000000000
```

Although the parameters of *std.dev* and *smooth* are close to 0 we cannot draw hypothesis testing here given that the estimates are biased. Still, informal comparison of AIC or BIC can lead to a more efficient model.



(a) bias (b) CRPS (c) empirical cdf

Figure 11: different metrics related to prediction under both models. Now, bias is effectively the same under both models.

4 Discussion

This vignette introduced the R package **coco**, which offers a modular and user-friendly approach for modeling Gaussian processes with flexible covariance models. For moderate sample sizes, we can independently adjust several source of nonstationarity of the process spatial structure. We also address the big-N problem by offering a tailored model focusing only on relevant aspects for prediction under very large datasets and combining the presented covariance model with the tapering approach. The package also presents a variety of functions for visualizing, inspecting, and simulating Gaussian Processes with flexible covariance models.

(a) Classic (b) Flexible

Figure 12: Interpolation error under both models.

It is important to be aware of the assumptions associated with the presented model to obtain reliable results. One of the key differences in comparison to those assumptions related to classical Gaussian processes is that we replace the (weakly) stationary and isotropic assumptions with the assumption introduced by Journel and Huijbregts [1976], which defines quasi-stationarity as local stationarity. This assumption implies that the process can be considered stationary locally in a small window. Moreover, we assume that the covariance matrix is known, and thus, an implicit assumption is that the uncertainty of the parameters associated with the covariance matrix can be neglected. Finally, when considering many parameters in the covariance structure, regularization is often required to promote the covariance matrix's numerical stability. We offer an easy-to-use penalization over the product between the global scale and global smoothness. This trades computational stability with prediction capabilities and is especially important when strong misspecification is present in the mean. The penalization term is implemented in the `coco` package by specifying in the `info` argument of the `coco` function as `list('lambda'= lambda)`. When modeling the different sources of nonstationarity, it might be too optimistic to expect the functional relationships between these characteristics and the covariates. In addition, the fit quality may be affected by the noisiness of the available covariates and the spatial configuration, to name a few. Instead of defining a continuous relationship between spatial structure and the covariate, categorizing the covariate to obtain more reliable local spatial structures may be preferred in these scenarios. Deviations from these model's assumptions have yet to be investigated.

5 Computational details

The results in this Vignette were obtained using R 4.3.1 with the packages coco (v.0.1), spam (v.2.9.1), fields (v.14.1), and gstat(v.2.1-1). All computations were done using a Macbook Air M2 with 16Gb of ram.

6 Disclaimer

The employed dataset has been simulated to showcase the benefits of the coco package. In real scenarios, we might not have the required available covariate information to properly represent the spatial structure. Moreover, until mentioned otherwise, we cannot guarantee that the current version of **coco** is a stable release. Therefore use with caution. A reproducible code of this Vignette can be found in .

7 Acknowledgements

References

Alan E Gelfand, Peter Diggle, Peter Guttorp, and Montserrat Fuentes. *Handbook of spatial statistics*. CRC press, 2010.

Reinhard Furrer, Roman Flury, and Florian Gerber. *spam: SPArse Matrix*, 2023. URL `https://CRAN.R-project.org/package=spam`. R package version 2.10-0.

Reinhard Furrer and Stephan R. Sain. spam: A sparse matrix R package with emphasis on MCMC methods for Gaussian Markov random fields. *Journal of Statistical Software*, 36(10):1–25, 2010. doi:10.18637/jss.v036.i10.

Florian Gerber and Reinhard Furrer. Pitfalls in the implementation of Bayesian hierarchical modeling of areal count data: An illustration using BYM and Leroux models. *Journal of Statistical Software, Code Snippets*, 63(1):1–32, 2015. doi:10.18637/jss.v063.c01.

Florian Gerber, Kaspar Moesinger, and Reinhard Furrer. Extending R packages to support 64-bit compiled code: An illustration with spam64 and GIMMS NDVI3g data. *Computer & Geoscience*, 104:109–119, 2017. ISSN 0098-3004. doi:10.1016/j.cageo.2016.11.015.

Reinhard Furrer, Marc G Genton, and Douglas Nychka. Covariance tapering for interpolation of large spatial datasets. *Journal of Computational and Graphical Statistics*, 15(3):502–523, 2006.

Douglas Nychka, Reinhard Furrer, John Paige, and Stephan Sain. fields: Tools for spatial data, 2021. URL `https://github.com/dnychka/fieldsRPackage`. R package version 15.2.

Adrian Baddeley, Ege Rubak, and Rolf Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press, London, 2015. ISBN 9781482210200. URL `https://www.routledge.com/Spatial-Point-Patterns-Methodology-and-Applications-with-R/Baddeley-Rubak-Turner/p/book/9781482210200/`.

Adrian Baddeley, Rolf Turner, Jorge Mateu, and Andrew Bevan. Hybrids of gibbs point process models and their implementation. *Journal of Statistical Software*, 55(11):1–43, 2013. doi:10.18637/jss.v055.i11.

Adrian Baddeley and Rolf Turner. spatstat: An R package for analyzing spatial point patterns. *Journal of Statistical Software*, 12(6):1–42, 2005. doi:10.18637/jss.v012.i06.

Edzer J. Pebesma. Multivariable geostatistics in S: the gstat package. *Computers & Geosciences*, 30: 683–691, 2004.

Benedikt Gräler, Edzer Pebesma, and Gerard Heuvelink. Spatio-temporal interpolation using gstat. *The R Journal*, 8:204–218, 2016. URL `https://journal.r-project.org/archive/2016/RJ-2016-014/index.html`.

Edzer Pebesma and Roger Bivand. *Spatial Data Science: With applications in R*. Chapman and Hall/CRC, 2023a. doi:10.1201/9780429459016. URL `https://r-spatial.org/book/`.

Edzer Pebesma. Simple Features for R: Standardized Support for Spatial Vector Data. *The R Journal*, 10(1):439–446, 2018. doi:10.32614/RJ-2018-009. URL `https://doi.org/10.32614/RJ-2018-009`.

Edzer Pebesma and Roger Bivand. *Spatial Data Science: With applications in R*. Chapman and Hall/CRC, London, 2023b. doi:10.1201/9780429459016. URL `https://r-spatial.org/book/`.

Mark D. Risser and Catherine A. Calder. Local likelihood estimation for covariance functions with spatially-varying parameters: The convoSPAT package for R. *Journal of Statistical Software*, 81(14): 1–32, 2017. doi:10.18637/jss.v081.i14.

Aldo V Vecchia. Estimation and model identification for continuous spatial processes. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 50(2):297–312, 1988.

Joseph Guinness. Permutation and grouping methods for sharpening gaussian process approximations. *Technometrics*, 60(4):415–429, 2018.

Jakob A Dambon, Fabio Sigrist, and Reinhard Furrer. varycoef: An r package for gaussian process-based spatially varying coefficient models. *arXiv preprint arXiv:2106.02364*, 2021.

Moreno Bevilacqua and Carlo Gaetan. Comparing composite likelihood methods based on pairs for spatial gaussian random fields. *Statistics and Computing*, 25:877–892, 2015.

Moreno Bevilacqua, Alfredo Alegria, Daira Velandia, and Emilio Porcu. Composite likelihood inference for multivariate gaussian random fields. *Journal of agricultural, biological, and environmental statistics*, 21:448–469, 2016.

Ronny Vallejos, Felipe Osorio, and Moreno Bevilacqua. *Spatial relationships between two georeferenced variables: With applications in R*. Springer Nature, 2020.

Michael Dumelle, Matt Higham, and Jay M. Ver Hoef. spmodel: Spatial statistical modeling and prediction in R. *PLOS ONE*, 18(3):1–32, 2023. doi:10.1371/journal.pone.0282524. URL `https://doi.org/10.1371/journal.pone.0282524`.

Reinhard Furrer. Modeling dependent data: An excursion. *Script for UZH STA330*, 2016.

M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions*. Dover, New York, 1970.

Christopher J Paciorek and Mark J Schervish. Spatial modelling using a new class of nonstationary covariance functions. *Environmetrics: The official journal of the International Environmetrics Society*, 17(5):483–506, 2006.

Isaac J Schoenberg. Metric spaces and completely monotone functions. *Annals of Mathematics*, pages 811–841, 1938.

Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

Florian Gerber and Reinhard Furrer. optimparallel: An r package providing a parallel version of the l-bfgs-b optimization method. *The R Journal*, 11(1):352–358, 2019. doi:10.32614/RJ-2019-030. URL `https://doi.org/10.32614/RJ-2019-030`.

Tilmann Gneiting and Adrian E Raftery. Strictly proper scoring rules, prediction, and estimation. *Journal of the American statistical Association*, 102(477):359–378, 2007.

Andre G Journel and Charles J Huijbregts. Mining geostatistics. 1976.