Supplementary Materials for Fast and Scalable Inference for Spatial Extreme Value Models

2024-03-11

Abstract

This document provides all code to reproduce the Figures and Tables in "Fast and Scalable Inference for Spatial Extreme Value Models" by [name withheld for double-blind review]. It also provides (i) a comparison of different Max steps for the Max-and-Smooth algorithm and (ii) a comparison of the proposed Laplace estimator to the exact Max-and-Smooth implementation of Jóhannesson et al. (2022).

Contents

1 Preliminaries

Our models are implemented in the R/C++ package **SpatialGEV** included in the Supplementary Material as a `tar.gz` file. **SpatialGEV** provides tool to perform approximate inference for a number of generalized extreme value (GEV) models with spatially-varying parameters modelled via a Gaussian process, which we refer to as GEV-GP models.

1.1 GEV-GP models

Let y_{ij} denote observation j at spatial location i, of which the two-dimensional spatial coordinates are given by $\boldsymbol{x}_i$. Then the general form of the GEV-GP models that can be fit with the package is

$$\begin{aligned} y_{ij} &\overset{\text{iid}}{\sim} \text{GEV}(a(\boldsymbol{x}_i), \exp(b(\boldsymbol{x}_i)), \exp(s(\boldsymbol{x}_i)) \\ u(\boldsymbol{x}) &\sim \text{GP}(\boldsymbol{c}_u(\boldsymbol{x})'\boldsymbol{\beta}_u(\boldsymbol{x}), \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}' \mid \boldsymbol{\eta}_u), \end{aligned} \tag{1}$$

where $\text{GEV}(a, b_o, s_o)$ denotes a GEV distribution for which the PDF is given by

$$f(y \mid a, b_o, s_o) = \frac{1}{b_o} t(y)^{s_o+1} \exp\{-t(y)\}, \qquad t(y) = \begin{cases} (1 + s_o \cdot \frac{y-a}{b_o})^{-1/s_o} & s_o \neq 0, \\ \exp\{\frac{y-a}{b_o}\}, & s_o = 0, \end{cases} \tag{2}$$

$u \in \{a, b, s\}$ is any subset of the GEV parameters which we would like to model as spatially varying, and $\text{GP}(\mu(\boldsymbol{x}), \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}'))$ is a Gaussian process with mean function $\mu(\boldsymbol{x})$ and covariance kernel $\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')$. The GEV distribution has bounded support depending on the value of (a, b_o, s_o): it is bounded below by $a - b_o/s_o$ when $s_o > 0$, bounded above by $a - b_o/s_o$ when $s_o < 0$, and unbounded when $s_o = 0$. **SpatialGEV** is designed to analyze data consisting of large extreme events. By restricting attention to $s_o = \exp(s) > 0$ it only considers GEV models which do not upper bound the data.

1.2 Setup

SpatialGEV is expected to run on R version 4.3 or higher. In order to install it, first we install the package dependencies (and other packages used in the code provided here):

```
pkg_deps <- c("INLA", "rstan", "tmbstan", "TMB", "Matrix",
              "rlang", "evd", "mvtnorm", "mclust", "SpatialExtremes",
              "dplyr", "tidyr", "ggplot2", "ggpubr", "fields",
              "kableExtra", "doParallel",
              "MASS", "SparseM", "tidyverse", "data.table")
install.packages(pkg_deps, dependencies = TRUE)
```

Now we install the **SpatialGEV** package as follows:

```
install.packages("SpatialGEV_1.0.0.tar.gz", repos = NULL, type = "source",
                 INSTALL_opts = "--install-tests")
```

We may test that the installation was successful by running the following code *after quitting and restarting R*:

```
testthat::test_package("SpatialGEV", reporter = "progress")
```

After checking that the installation was successful, we can load the package and its dependencies in order to run the remainder of the code in this document.

```
library(SpatialGEV) # Our model and inference method
library(INLA) # For constructing Matérn SPDE approximation
# HMC-NUTS
library(rstan)
library(tmbstan)
# MCMC with TMB models
library(TMB)
library(Matrix)
library(rlang)
# Data simulation
library(evd)
library(mvtnorm)
library(mclust)
library(SpatialExtremes)
library(dplyr)
library(tidyr)
# Plotting and tables
library(ggplot2)
library(ggpubr)
library(fields)
library(kableExtra)
# Parallelization
library(doParallel)
# Helper functions
source("paper_code_helper.R")
source("laplace_mcmc.R") # mcmc for TMB models
```

2 Small-scale study on smooth surfaces

To run the simulation study, first simulate data from 400 locations using the following reproducible code. The simulated random effects are plotted in Figure 1.

```
set.seed(123)
small_sim <- simulate_data_small()
par(mfrow=c(1,3))
grid_plot(small_sim$lon, small_sim$lat,
          small_sim$a_mat, "a")
grid_plot(small_sim$lon, small_sim$lat,
          small_sim$logb_mat, "b")
grid_plot(small_sim$lon, small_sim$lat,
          small_sim$logs_mat, "s")
```

2.1 Laplace method via joint Normal approximation (Laplace-MQ)

The function `spatialGEV_fit()` in the **SpatialGEV** package is used to find the mode and quadrature of the Laplace approximation to the marginal hyperparameter posterior, which is the basis of a Normal

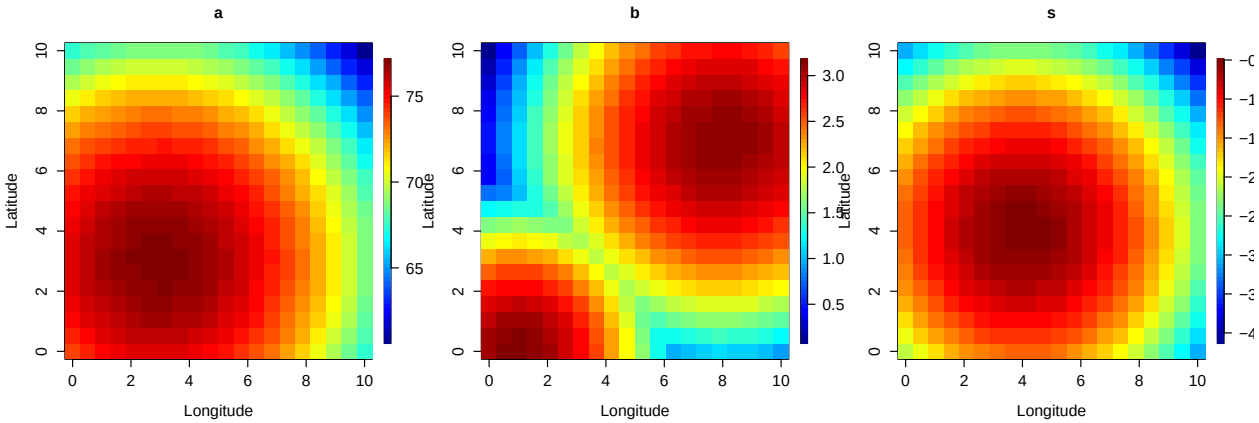

Figure 1: Simulated random effects plotted on regular lattices

approximation to the posterior of both hyperparameters and random effects referred to as Laplace-MQ in our paper. The function `spatialGEV_sample()` is then used to sample from this distribution via a sparse representation of its precision matrix. In order to do this, a C++ implementation of the full log-posterior arising from the GEV-GP model (1) – which we refer to as the "model template" – is processed by the **TMB** package (Kristensen et al. 2016), which provides a highly efficient implementation of the Laplace approximation. C++ model templates for various spatial GEV-GP models are in the `src/TMB` folder of **SpatialGEV**. The file names are of the format `model_params_kernel.hpp` where `params` indicates which parameter(s) of the GEV distribution are treated as spatially varying, and `kernel` indicates which kernel to use for the spatial GP covariance matrix. Auxilliary functions used in the model templates are located in `inst/include/SpatialGEV/utils.hpp`. The **TMB** template for the GEV-GP model used in the paper is `model_abs_spde.hpp`, for which the C++ code is provided in Appendix A.

In order to fit the Laplace-MQ approximation, we first set the model priors and the initial parameters for the optimization procedure underlying the mode-quadrature calculation. Please see `?SpatialGEV::spatialGEV_fit` for details.

```
n_loc <- nrow(small_sim$locs)
init_param <- list(
  a = rep(60, n_loc),
  log_b = rep(3, n_loc),
  s = rep(-2,n_loc),
  beta_a = 60, beta_b = 3, beta_s = -2,
  log_sigma_a = -1, log_kappa_a = -1,
  log_sigma_b = -1, log_kappa_b = -1,
  log_sigma_s = -1, log_kappa_s = -1
)
# Weakly informative priors on beta
beta_prior <- list(beta_a=c(0,100), beta_b=c(0,50), beta_s=c(0,20))
```

We choose the Matérn SPDE kernel for the spatial covariance matrix by setting `kernel="spde"` and specify that all three GEV parameters a, b, s are random by setting `random="abs"`. We restrict the shape parameter s to be positive by setting `reparam_s="positive"`.

```
# Get mode and quadrature for Normal approximation
start_lap_mq <- Sys.time()
fit_lap_mq <- spatialGEV_fit(
  data=small_sim$data, locs=small_sim$locs,
```

```
  random = "abs",
  init_param = init_param,
  reparam_s = "positive", kernel="spde",
  beta_prior = beta_prior, silent = TRUE
)
# Posterior sampling from the Normal approximated posterior
sam_lap_mq <- spatialGEV_sample(fit_lap_mq, n_draw=10000)
time_lap_mq <- difftime(Sys.time(), start_lap_mq, units="secs")
```

Since the Matérn SPDE kernel is used, `spatialGEV_fit()` constructs a mesh on the spatial domain using functions from the **R-INLA** package (Lindgren and Rue 2015), resulting in an increase number of locations in the model than in the original dataset. We extract the location indices corresponding to the original locations using `meshidxloc` from the `spatialGEV_fit` object. Now we can obtain the posterior mean and standard deviation of the random effects using the Laplace method, and calculate the mean absolute errors of the random effect posterior means against their true values.

```
post_lap_mq <- spatialGEV_get_posteriors(fit_lap_mq, sam_lap_mq)
mae_a_lap_mq <- mean(abs(small_sim$a-post_lap_mq$a_mean))
mae_b_lap_mq <- mean(abs(small_sim$logb-post_lap_mq$logb_mean))
mae_s_lap_mq <- mean(abs(small_sim$logs-post_lap_mq$s_mean))
mae_z_lap_mq <- mean(abs(small_sim$z_true-post_lap_mq$z_mean))
```

2.2 Exact Laplace posterior via MCMC (Laplace-MCMC)

The Normal approximation to the Laplace posterior is known to underestimate the posterior uncertainty of the hyperparameters, which we confirm in the paper. However, it is possible to explore the exact Laplace joint posterior distribution via MCMC. Specifically, one uses MCMC to explore the Laplace marginal hyperparameter posterior. Next one combines each hyperparameter draw $\boldsymbol{\theta}$ with a conditional posterior draw of the random effects $\boldsymbol{u}$, which given the data $\boldsymbol{y}$ and $\boldsymbol{\theta}$ are multivariate normal with mean and variance being nonlinear functions of $\boldsymbol{y}$ and $\boldsymbol{\theta}$. Fortunately though, this mean and variance are byproducts of computing the Laplace hyperparameter posterior at each value of $\boldsymbol{\theta}$, such that MCMC sampling from the exact Laplace posterior of both hyperparameters and random effects incurs little extra cost over MCMC sampling from just the Laplace marginal hyperparameter posterior.

The code below implements this MCMC procedure on parallel chains using a random walk proposal for the hyperparameters with variance matrix proportional to that of the Laplace-MQ approximation. This MCMC method is referred to as Laplace-MCMC in our paper.

Note: The following chunk of code takes over 20 hours to run.

```
# set up number of MCMC iterations
n_chain <- 6
n_iter <- 45e3
n_burnin <- 5e3
var_scale <- .5 # scale for the MCMC proposal
cl <- makeCluster(n_chain) # Set up parallelization
registerDoParallel(cl)
parallel::clusterSetRNGStream(cl, iseed = 234)

system.time(
  lap_all_chains <- foreach(i = 1:n_chain, .export = "spatialGEV_fit") %dopar% {
    start_time <- Sys.time()
    # step 1: construct TMB adfun object
```

```
    # Need to create adfun in each parallel task otherwise code fails
    lap_adf <- spatialGEV_fit(
      data=small_sim$data, locs=small_sim$locs,
      random = "abs",
      init_param = init_param,
      reparam_s = "positive", kernel="spde",
      beta_prior = beta_prior,
      adfun_only = TRUE, silent = TRUE
    )$adfun
    # step 2: fit the Laplace-MVN approximation
    # used to construct the MCMC proposal distribution
    lap_fit <- nlminb(start = lap_adf$par,
                      objective = lap_adf$fn,
                      gradient = lap_adf$g)
    lap_fixed_pars <- get_mvn(lap_adf, select = "fixed")
    # step 3: mcmc sampling
    lap_prop_sim <- function(prev, obj) {
      # random walk proposal with variance proportional to Laplace-MVN
      x <- mvtnorm::rmvnorm(
        n = 1,
        mean = prev,
        sigma = var_scale * lap_fixed_pars$var)
      drop(x)
    }
    lap_samples <- laplace_mcmc(
      obj = lap_adf,
      n_iter = n_iter,
      fixed_init = lap_fixed_pars$mean,
      prop_sim = lap_prop_sim,
      prop_lpdf = NULL,
      print_every = 10
    )
    time_taken <- difftime(Sys.time(), start_time)
    lap_samples['time'] <- time_taken
    lap_samples$fixed <- lap_samples$fixed[-(1:n_burnin),]
    lap_samples$random <- lap_samples$random[-(1:n_burnin),]
    lap_samples
})
stopCluster(cl)
```

```
lapmc_chain_times <- unlist(lapply(
  lap_all_chains, function(chain) chain$time
))
meshidxloc <- fit_lap_mq$meshidxloc

# Get random effect posteriors from the MCMC chains
lapmc_res_per_chain <- get_random_posteriors_mcmc(lap_all_chains, meshidxloc)

# Extract posterior means of the random effects
lapmc_mean_per_chain <- lapply(lapmc_res_per_chain,
                               function(chain) chain$pos_means)
lapmc_mean <- apply(simplify2array(lapmc_mean_per_chain), c(1, 2), mean)
```

```
# Extract posterior samples of z10
z_draws_lapmc <- do.call(cbind, lapply(lapmc_res_per_chain,
                                       function(chain) chain$z_draws))

# Get posterior samples of all fixed effects as a matrix
lapmc_sam_fixed <- do.call(rbind,
                           lapply(lap_all_chains, function(chain){chain$fixed}))
sams_betaa_lapmc <- lapmc_sam_fixed[, "beta_a"]
sams_betab_lapmc <- lapmc_sam_fixed[, "beta_b"]
sams_betas_lapmc <- lapmc_sam_fixed[, "beta_s"]
sams_siga_lapmc <- lapmc_sam_fixed[,"log_sigma_a"]
sams_sigb_lapmc <- lapmc_sam_fixed[,"log_sigma_b"]
sams_sigs_lapmc <- lapmc_sam_fixed[,"log_sigma_s"]
sams_kapa_lapmc <- lapmc_sam_fixed[,"log_kappa_a"]
sams_kapb_lapmc <- lapmc_sam_fixed[,"log_kappa_b"]
sams_kaps_lapmc <- lapmc_sam_fixed[,"log_kappa_s"]
rm('lap_all_chains'); rm('lapmc_res_per_chain'); rm('lapmc_mean_per_chain')

save(lapmc_mean, sams_betaa_lapmc, sams_betab_lapmc, sams_betas_lapmc,
     sams_siga_lapmc, sams_sigb_lapmc, sams_sigs_lapmc, sams_kapa_lapmc,
     sams_kapb_lapmc, sams_kaps_lapmc,
     file="rda_files/lapmc_results.RData")
```

Calculate the mean absolute values of the random effect estimates against their true values:

```
a_lapmc <- lapmc_mean[,1]
logb_lapmc <- lapmc_mean[,2]
logs_lapmc <- lapmc_mean[,3]
z_lapmc <- lapmc_mean[,4]
mae_a_lap_mc <- mean(abs(small_sim$a-a_lapmc))
mae_b_lap_mc <- mean(abs(small_sim$logb-logb_lapmc))
mae_s_lap_mc <- mean(abs(small_sim$logs-logs_lapmc))
mae_z_lap_mc <- mean(abs(small_sim$z_true-z_lapmc))
```

2.3 Max-and-Smooth via joint Normal approximation (Maxsmooth-MQ)

Next, we use the Max-and-Smooth approach for model inference. This method consists of approximating the GEV contribution to the posterior resulting from (1) by independent multivariate normals at each spatial location, with mean and variance obtained by mode-quadrature of the GEV likelihood arising from (2) based on only the observations at a given location. This results in a so-called Normal-Normal random-effects model for which (i) the marginal hyperparameter posterior and (ii) the random-effects posterior conditioned on both hyperaparameters and data are both analytically tractable.

In this section, we fit a jointly Normal approximation to the Max-and-Smooth posterior distribution, referred to as Maxsmooth-MQ, analogous to the Laplace-MQ approximation in Section 2.1.

2.3.1 Comparison of different Max step methods

The Max step of Max-and-Smooth is the mode-quadrature calculation at each spatial location. We consider three different methods for the Max step:

- `evd`: This method computes the mode and quadrature of the GEV likelihood using the function `evd::gevfit()` from the **evd** package (Stephenson 2002). Note that `evd::gevfit()` considers a more general GEV model than ours, i.e., with a shape parameter s_o in (2) for which our shape parameter corresponds to $s = \log(s_o)$. For some locations this produced an estimate $\hat{s}_o < 0$, in which case the `evd` method could not be used for the given location under our restricted GEV model.
- `j22`: Using the generalized GEV likelihood of Jóhannesson et al. (2022) described in Section 1.6 of the Supplementary Material of that paper. This method consists of finding the mode and quadrature of a penalized GEV likelihood. Details are provided in Section 5. For now, we point out that `j22` also occasionally resulted in an estimate $\hat{s}_o < 0$, such that it could not be used for all spatial locations.
- `tmb`: Using a generalized likelihood approach designed for our restricted GEV parametrization. In this case, the penalty corresponds to the log of a Normal$(0, 100^2)$ distribution on s – a very weak penalty which simply ensures that the mode does not escape to the boundary value $s_o = 0$. Model fitting was done with **TMB**, which uses automatic differentation to quickly perform the optimization at each spatial location using the gradient-based optimizer `stats:nlminb()`. The `tmb` method successfully computed the mode and quadrature of the Max step at each spatial location.

Figures 2 and 3 display the point estimates and standard errors for each of these three Max step methods at each of the spatial locations for which they are defined.

```
set.seed(123)
# allocate space for max-step of each method
mle_set <- matrix(NA, n_loc, 3)
colnames(mle_set) <- c("a", "log_b", "log_s")
mle_set <- list(evd = mle_set, j22 = mle_set, tmb = mle_set)
var_set <- array(NA, dim = c(3, 3, n_loc))
var_set <- list(evd = var_set, j22 = var_set, tmb = var_set)
start_max_step <- Sys.time()
# loop through locations
for(i in 1:n_loc) {
  yi <- small_sim$data[[i]]
  # evd method
  max_fit <- maxsmooth_maxstep(y = yi, method = "evd")
  mle_set$evd[i,] <- max_fit$est
  var_set$evd[,,i] <- max_fit$var
  # tmb method
  max_fit <- maxsmooth_maxstep(y = yi, method = "tmb",
                               s_prior = c(0, 100))
  mle_set$tmb[i,] <- max_fit$est
  var_set$tmb[,,i] <- max_fit$var
  # j22 method
  max_fit <- maxsmooth_maxstep(y = yi, method = "j22")
  mle_set$j22[i,] <- max_fit$est
  var_set$j22[,,i] <- max_fit$var
}
max_step_time <- difftime(Sys.time(), start_max_step)
```

```
# mle plot
tmb_vs_j22 <- max_step_compare(mle_set, var_set,
                               y = "tmb", x = "j22", type = "mle")
tmb_vs_evd <- max_step_compare(mle_set, var_set,
                               y = "tmb", x = "evd", type = "mle")
j22_vs_evd <- max_step_compare(mle_set, var_set,
```

```
                                y = "j22", x = "evd", type = "mle")
mle_plots <- ggarrange(tmb_vs_j22, tmb_vs_evd, j22_vs_evd, ncol=1, nrow=3,
                       common.legend = TRUE)
mle_plots
```

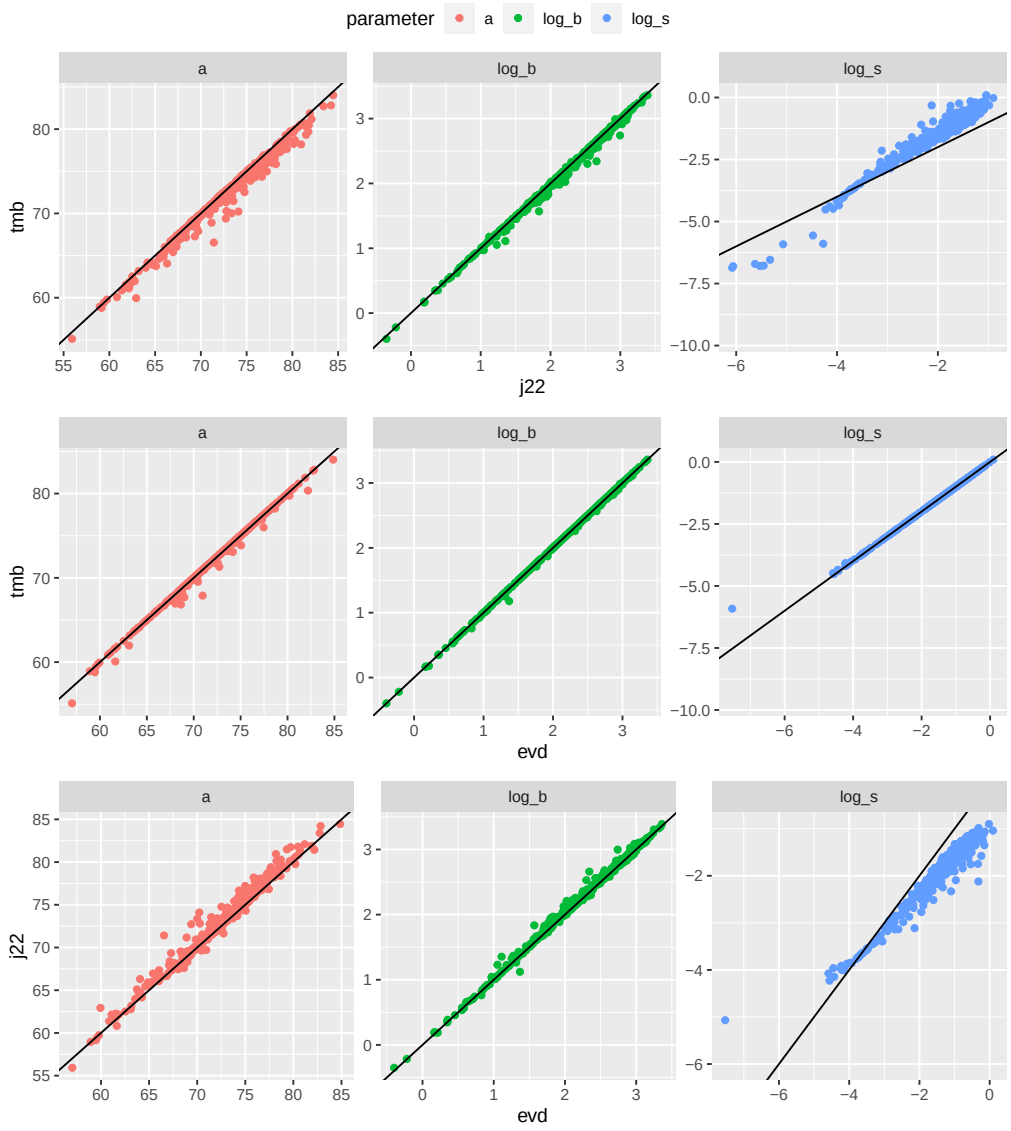

Figure 2: Comparison of point estimates for different Max step methods.

```
# se plot
tmb_vs_j22 <- max_step_compare(mle_set, var_set,
                               y = "tmb", x = "j22", type = "se") +
  scale_x_log10() + scale_y_log10()
tmb_vs_evd <- max_step_compare(mle_set, var_set,
                               y = "tmb", x = "evd", type = "se") +
  scale_x_log10() + scale_y_log10()
j22_vs_evd <- max_step_compare(mle_set, var_set,
                               y = "j22", x = "evd", type = "se") +
```

```
  scale_x_log10() + scale_y_log10()
se_plots <- ggarrange(tmb_vs_j22, tmb_vs_evd, j22_vs_evd, ncol=1, nrow=3,
                      common.legend = TRUE)
se_plots
```

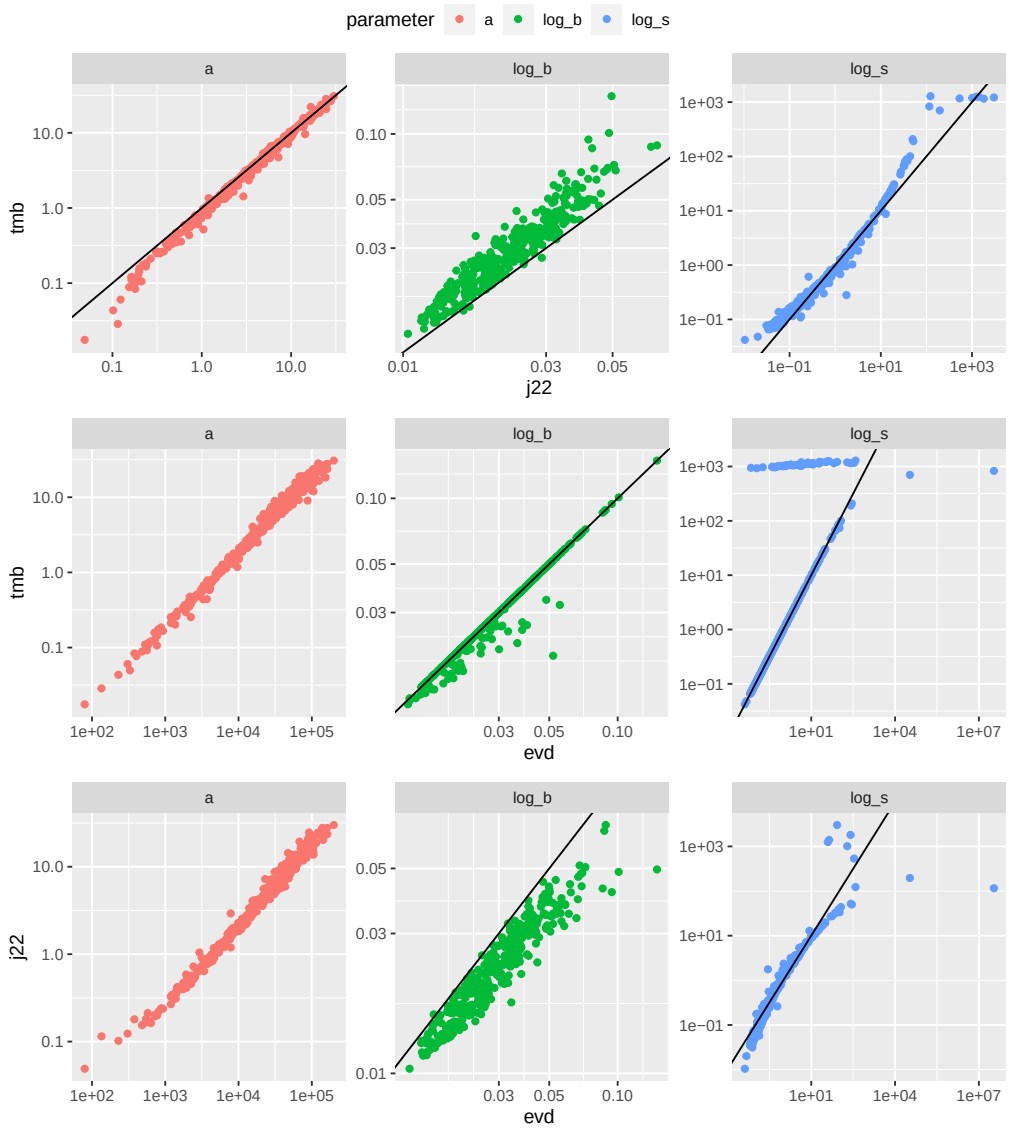

Figure 3: Comparison of standard error estimates for different Max step methods.

The three methods give similar point estimates for all random effects at different locations. However, the standard errors obtained using `evd` are quite different from those obtained `j22` and `tmb`, especially for a and s. The SEs for `tmb` and `j22` are similar overall, with `tmb` producing somewhat larger SEs for b across all locations. `tmb` also gives larger SEs for s at a several locations.

In consistency with our modelling assumption of $s_o > 0$, we implement the Max-and-Smooth method with Max step estimates obtained from `tmb` method. The `j22` method is employed in our comparison with the exact Max-and-Smooth implementation of Jóhannesson et al. (2022) in Section 5.

The Smooth step of Max-and-Smooth refers to inference for the hyperparameter and random effects under the Normal-Normal approximation. We note that the exact marginal hyperparameter posterior for this Normal-

Normal approximation can be computed very efficiently using **TMB**, because the Laplace approximation for a Normal-Normal model is in fact exact. For this reason, we are able to use `spatialGEV_fit()` and `spatialGEV_sample()` to implement the Maxsmooth-MQ approximation in the code below.

```
start_ms_mq <- Sys.time()
fit_ms_mq <- spatialGEV_fit(data=list(est=mle_set$tmb,
                                      var=var_set$tmb),
                            locs=small_sim$locs,
                            random="abs",
                            method="maxsmooth",
                            init_param=init_param,
                            reparam_s = "positive", kernel="spde",
                            beta_prior = beta_prior, silent = TRUE)
sam_ms_mq <- spatialGEV_sample(fit_ms_mq, n_draw=10000)
maxsmooth_mq_time <- difftime(Sys.time(), start_ms_mq)

# MAEs of random effects
post_ms_mq <- spatialGEV_get_posteriors(fit_ms_mq, sam_ms_mq)
mae_a_ms_mq <- mean(abs(small_sim$a-post_ms_mq$a_mean))
mae_b_ms_mq <- mean(abs(small_sim$logb-post_ms_mq$logb_mean))
mae_s_ms_mq <- mean(abs(small_sim$logs-post_ms_mq$s_mean))
mae_z_ms_mq <- mean(abs(small_sim$z_true-post_ms_mq$z_mean))
```

2.4 Exact Max-and-Smooth posterior via MCMC (Maxsmooth-MCMC)

It is possible to use MCMC to explore the exact Max-and-Smooth joint posterior distribution, i.e., without the additional Normal approximation, using a procedure analoguous to what we have described for the Laplace method in Section 2.2. This is referred to as Maxsmooth-MCMC in our paper and is implemented in the code below.

Note: The following chunk of code takes over 9 hours to run.

```
# step 0: specify MCMC #chains and #iterations
n_chain <- 6
n_iter <- 45e3
n_burnin <- 5e3
var_scale <- .5 # proposal scale
# step 1: max step
ms_method <- "tmb"
ms_s_prior <- c(0, 100)
# allocate memory
param_names <- c("a", "log_b", "s")
ms_data <- list(
  est = matrix(NA, n_loc, 3,
               dimnames = list(NULL, param_names)),
  var = array(NA, dim = c(3, 3, n_loc),
              dimnames = list(param_names, param_names, NULL))
)
# loop through data to get max-step estimates
for(i in 1:n_loc) {
  yi <- small_sim$data[[i]]
  ms_fit <- maxsmooth_maxstep(y = yi,
                              method = ms_method,
```

```
                              s_prior = ms_s_prior)
  ms_data$est[i,] <- ms_fit$est
  ms_data$var[,,i] <- ms_fit$var
}
cl <- makeCluster(n_chain) # Set up parallelization
registerDoParallel(cl)
parallel::clusterSetRNGStream(cl, iseed = 234)
system.time(
  ms_all_chains <- foreach(i = 1:n_chain, .packages = "SpatialGEV") %dopar% {
    start_time <- Sys.time()
    # initialize the model in tmb
    ms_adf <- spatialGEV_fit(data=ms_data,
                             locs=small_sim$locs,
                             random="abs",
                             method="maxsmooth",
                             init_param=init_param,
                             reparam_s = "positive", kernel="spde",
                             beta_prior = beta_prior, silent = TRUE)$adfun
    # fit the Laplace-MVN approximatioun
    ms_fit <- nlminb(start = ms_adf$par,
                     objective = ms_adf$fn,
                     gradient = ms_adf$g)
    ms_fixed_pars <- get_mvn(ms_adf, select = "fixed")
    # MCMC sampling from full posterior
    ms_prop_sim <- function(prev, obj) {
      x <- mvtnorm::rmvnorm(
        n = 1,
        mean = prev,
        sigma = var_scale * ms_fixed_pars$var)
      drop(x)
    }
    ms_samples <- laplace_mcmc(
      obj = ms_adf,
      n_iter = n_iter,
      fixed_init = ms_fixed_pars$mean,
      prop_sim = ms_prop_sim,
      prop_lpdf = NULL,
      print_every = 10
    )
    time_taken <- difftime(Sys.time(), start_time)
    ms_samples['time'] <- time_taken
    ms_samples$fixed <- ms_samples$fixed[-(1:n_burnin),]
    ms_samples$random <- ms_samples$random[-(1:n_burnin),]
    ms_samples
})

stopCluster(cl)
```

Next, we extract the posterior means of the random effects and the samples of the fixed effect posteriors.

Note: The code below takes up a lot of memory.

```
msmc_chain_times <- unlist(lapply(ms_all_chains, function(chain)chain$time))
meshidxloc <- fit_ms_mq$meshidxloc
# Get posterior samples of random effects from the MCMC chains
msmc_res_per_chain <- get_random_posteriors_mcmc(ms_all_chains, meshidxloc)

# Get posterior means of the random effects
msmc_mean_per_chain <- lapply(msmc_res_per_chain, function(chain) chain$pos_means)
msmc_mean <- apply(simplify2array(msmc_mean_per_chain), c(1, 2), mean)

# Get posterior samples of z10
z_draws_msmc <- do.call(cbind, lapply(msmc_res_per_chain,
                                      function(chain) chain$z_draws))

# Get posterior samples of all fixed effects as a matrix
msmc_sam_fixed <- do.call(rbind,
                          lapply(ms_all_chains, function(chain){chain$fixed}))
sams_betaa_msmc <- msmc_sam_fixed[, "beta_a"]
sams_betab_msmc <- msmc_sam_fixed[, "beta_b"]
sams_betas_msmc <- msmc_sam_fixed[, "beta_s"]
sams_siga_msmc <- msmc_sam_fixed[,"log_sigma_a"]
sams_sigb_msmc <- msmc_sam_fixed[,"log_sigma_b"]
sams_sigs_msmc <- msmc_sam_fixed[,"log_sigma_s"]
sams_kapa_msmc <- msmc_sam_fixed[,"log_kappa_a"]
sams_kapb_msmc <- msmc_sam_fixed[,"log_kappa_b"]
sams_kaps_msmc <- msmc_sam_fixed[,"log_kappa_s"]
rm('ms_all_chains'); rm('msmc_res_per_chain'); rm('msmc_mean_per_chain')

save(msmc_mean, sams_betaa_msmc, sams_betab_msmc, sams_betas_msmc,
     sams_siga_msmc, sams_sigb_msmc, sams_sigs_msmc, sams_kapa_msmc,
     sams_kapb_msmc, sams_kaps_msmc,
     file="rda_files/msmc_results.RData")
```

```
# MAEs of random effects
a_msmc <- msmc_mean[,1]
logb_msmc <- msmc_mean[,2]
logs_msmc <- msmc_mean[,3]
z_msmc <- msmc_mean[,4]
mae_a_ms_mc <- mean(abs(small_sim$a-a_msmc))
mae_b_ms_mc <- mean(abs(small_sim$logb-logb_msmc))
mae_s_ms_mc <- mean(abs(small_sim$logs-logs_msmc))
mae_z_ms_mc <- mean(abs(small_sim$z_true-z_msmc))
```

2.5 Full Bayesian inference on the joint posterior with Stan (HMC-NUTS)

The benchmark for evaluating the speed and accuracy of the Laplace and Max-and-Smooth approximations is the exact joint posterior on the hyperparameters and random effects arising from the GEV-GP model (1), which we explore using MCMC via the HMC-NUTS algorithm provided by the **rstan** package (Stan Development Team 2020). We use the **tmbstan** package (Monnahan and Kristensen 2018) to efficiently and conveniently connect the C++ implementations of the **SpatialGEV** models to the **rstan** automatic differentiation engine.

First, we set initial values and priors for running HMC-NUTS with **rstan**, which samples from the model described by a **TMB** template.

```
# Create a TMB ADFun object using our spatialGEV_fit() function by setting adfun_only
# This object will be fed to tmbstan() to perform HMC sampling
obj_s <- spatialGEV_fit(data=small_sim$data, locs=small_sim$locs, random = "abs",
                        init_param = init_param,
                        reparam_s = "positive", kernel="spde",
                        beta_prior = beta_prior,
                        adfun_only = TRUE)
```

```
## Constructing atomic D_lgamma
## Constructing atomic D_lgamma
```

```
n_s <- obj_s$mesh$n
meshidxloc <- obj_s$mesh$idx$loc
init_param_stan <- list(a = rep(60, n_s),
                        log_b = rep(3, n_s),
                        s = rep(-2,n_s),
                        beta_a = 60, beta_b = 3, beta_s = -2,
                        log_sigma_a = -1, log_kappa_a = -1,
                        log_sigma_b = -1, log_kappa_b = -1,
                        log_sigma_s = -1, log_kappa_s = -1)
```

Next, we run HMC-NUTS on the model template using `tmbstan::tmbstan()`.

Note: This code might take a few days to run.

```
n_chains <- 6
n_iter <- 8000
options(mc.cores=n_chains)
fit_hmc <- tmbstan(obj_s$adfun, chains=n_chains, iter=n_iter,
                   init=function(){init_param_stan}, seed=123,
                   verbose=TRUE, silent=FALSE, refresh=10)
```

The following code computes the posterior mean and standard deviation from the HMC-NUTS samples and calculate the mean absolute errors of the random effect posterior means against their true values.

```
summary_hmc <- summary(fit_hmc)$summary
# Random effects indices
a_ind_h <- paste0("a","[",meshidxloc,"]")
logb_ind_h <- paste0("log_b","[",meshidxloc,"]")
logs_ind_h <- paste0("s","[",meshidxloc,"]")

# Posterior mean
a_hmc <- summary_hmc[a_ind_h, "mean"]
logb_hmc <- summary_hmc[logb_ind_h, "mean"]
b_hmc <- exp(logb_hmc)
logs_hmc <- summary_hmc[logs_ind_h, "mean"]
s_hmc <- exp(logs_hmc)

# Posterior sd
a_sd_hmc <- summary_hmc[a_ind_h, "sd"]
logb_sd_hmc <- summary_hmc[logb_ind_h, "sd"]
logs_sd_hmc <- summary_hmc[logs_ind_h, "sd"]
```

```r
# Get posterior mean of z_10
sam_hmc <- as.matrix(fit_hmc)
z_draws_h <- get_posterior_z10(sam_hmc, a_ind_h, logb_ind_h, logs_ind_h)
z_hmc <- apply(z_draws_h, 1, mean)
z_sd_hmc <- apply(z_draws_h, 1, sd)

# Posterior samples for the hyperparameters
sams_betaa_hmc <- sam_hmc[,"beta_a"]
sams_betab_hmc <- sam_hmc[,"beta_b"]
sams_betas_hmc <- sam_hmc[,"beta_s"]
sams_siga_hmc <- sam_hmc[,"log_sigma_a"]
sams_sigb_hmc <- sam_hmc[,"log_sigma_b"]
sams_sigs_hmc <- sam_hmc[,"log_sigma_s"]
sams_kapa_hmc <- sam_hmc[,"log_kappa_a"]
sams_kapb_hmc <- sam_hmc[,"log_kappa_b"]
sams_kaps_hmc <- sam_hmc[,"log_kappa_s"]
```

To obtain a similarity measure for the HMC-NUTS samples versus samples obtained using the four other estimators (Laplace-MQ, Laplace-MCMC, Maxsmooth-MQ, Maxsmooth-MCMC), we perform the following calculation:

1. At a given location i, we calculate the two-sample Kolmogorov-Smirnov (K-S) test statistic for $z_{10}(\boldsymbol{x}_i)$ between the HMC-NUTS samples and the corresponding samples obtained from each of the four other estimators.
2. We average the K-S statistic of all 400 locations for each of the four estimators.

The larger the value of this K-S metric, the more similar the approximate posteriors for z_{10} are to the true posteriors (as calculated by HMC-NUTS).

```r
# KS statistics
ks_z_lapmq <- ks_z_lapmc <- ks_z_msmq <- ks_z_msmc <- rep(NA, nrow(z_draws_h))
# K-S statistic between HMC samples of z10 vs Laplace samples of z10
set.seed(123)
for (i in 1:length(ks_z_lapmq)){
  ks_z_lapmq[i] <- ks.test(as.vector(post_lap_mq$z_draws[i,]),
                           z_draws_h[i,])$statistic
}

# KS statistic between HMC samples of z10 vs Laplace-MCMC samples of z10
set.seed(123)
for (i in 1:length(ks_z_lapmc)){
  ks_z_lapmc[i] <- ks.test(as.vector(z_draws_lapmc[i,]), z_draws_h[i,])$statistic
}

# KS statistic between HMC samples of z10 vs Max-and-smooth-MQ samples of z10
set.seed(123)
for (i in 1:length(ks_z_msmq)){
  ks_z_msmq[i] <- ks.test(as.vector(post_ms_mq$z_draws[i,]),
                          z_draws_h[i,])$statistic
}

# KS statistic between HMC samples of z10 vs MS-MCMC samples of z10
```

```r
set.seed(123)
for (i in 1:length(ks_z_msmc)){
  ks_z_msmc[i] <- ks.test(as.vector(z_draws_msmc[i,]), z_draws_h[i,])$statistic
}

ks_stat_df <- data.frame(cbind(c(mean(ks_z_lapmq), sd(ks_z_lapmq)),
                               c(mean(ks_z_lapmc), sd(ks_z_lapmc)),
                               c(mean(ks_z_msmq), sd(ks_z_msmq)),
                               c(mean(ks_z_msmc), sd(ks_z_msmc))
                               ))
colnames(ks_stat_df) <- c("Laplace-MQ", "Laplace-MCMC",
                          "Max-Smooth-MQ", "Max-Smooth-MCMC")
rownames(ks_stat_df) <- c("mean", "sd")
```

Finally, we check the mean absolute errors of the HMC-NUTS results.

```r
mae_a_hmc <- mean(abs(small_sim$a-a_hmc))
mae_b_hmc <- mean(abs(small_sim$logb-logb_hmc))
mae_s_hmc <- mean(abs(small_sim$logs-logs_hmc), na.rm=TRUE)
mae_z_hmc <- mean(abs(small_sim$z_true-z_hmc))
```

2.6 Figures and tables for the small-scale simulation study

Table 1 summarizes the mean absolute values of parameter estimates against their true values from different methods.

```r
mae_small <- data.frame(c(mae_a_hmc, mae_a_lap_mq, mae_a_lap_mc, mae_a_ms_mq, mae_a_ms_mc),
                        c(mae_b_hmc, mae_b_lap_mq, mae_b_lap_mc, mae_b_ms_mq, mae_b_ms_mc),
                        c(mae_s_hmc, mae_s_lap_mq, mae_s_lap_mc, mae_s_ms_mq, mae_s_ms_mc),
                        c(mae_z_hmc, mae_z_lap_mq, mae_z_lap_mc, mae_z_ms_mq, mae_z_ms_mc),
                        row.names = c("Stan-HMC", "Laplace (MQ)", "Laplace (MCMC)",
                                      "Max-and-Smooth (MQ)",
                                      "Max-and-Smooth (MCMC)"))
colnames(mae_small) <- c("MAE($a$)", "MAE($b$)", "MAE($s$)", "MAE($z_{10}$)")
make_table(mae_small, caption = "Summary of mean absolute errors.")
```

Table 1: Summary of mean absolute errors.

	MAE(a)	MAE(b)	MAE(s)	MAE(z_{10})
Stan-HMC	0.384	0.048	0.102	2.262
Laplace (MQ)	0.384	0.051	0.111	2.192
Laplace (MCMC)	0.383	0.050	0.107	2.186
Max-and-Smooth (MQ)	0.603	0.076	0.487	3.136
Max-and-Smooth (MCMC)	0.602	0.076	0.483	3.066

Table 2 summarizes the K-S statistics comparing the z_{10} posterior samples from each method vs those from HMC-NUTS. The closer the K-S statistic is to zero, the more the posterior samples from the method resemble the posterior samples from the benchmark HMC-NUTS results.

```
make_table(ks_stat_df,
           caption = "Summary of the K-S statistic.")
```

Table 2: Summary of the K-S statistic.

	Laplace-MQ	Laplace-MCMC	Max-Smooth-MQ	Max-Smooth-MCMC
mean	0.030	0.027	0.341	0.339
sd	0.013	0.012	0.179	0.173

Figure 4 displays the random effect posterior mean estimates versus the their corresponding true values.

```
mytheme <- create_ggplot_theme(text_size=10)

# All information to plot
rng_a <- range(c(small_sim$a, a_hmc, post_lap_mq$a_mean, post_ms_mq$a_mean,
                 a_lapmc, a_msmc))
rng_b <- range(c(small_sim$logb, logb_hmc, post_lap_mq$logb_mean,
                 post_ms_mq$logb_mean, logb_lapmc, logb_msmc))
rng_s <- range(c(small_sim$logs, logs_hmc, post_lap_mq$s_mean,
                 post_ms_mq$s_mean, logs_lapmc, logs_msmc))
rng_z <- range(c(small_sim$z_true, z_hmc, post_lap_mq$z_mean, post_ms_mq$z_mean,
                 z_lapmc, z_msmc))

true_z10_lab <- expression(paste("True ", z[10], "(x)"))
est_z10_lab <- expression(paste("Estimated ", z[10], "(x)"))

scatterp_range_list <- rep(list(rng_a, rng_b, rng_s, rng_z), 5)
scatterp_x_list <- rep(list(small_sim$a, small_sim$logb, small_sim$logs,
                            small_sim$z_true), 5)
scatterp_y_list <- list(a_hmc, logb_hmc, logs_hmc, z_hmc,
                        post_lap_mq$a_mean, post_lap_mq$logb_mean,
                        post_lap_mq$s_mean, post_lap_mq$z_mean,
                        a_lapmc, logb_lapmc, logs_lapmc, z_lapmc,
                        post_ms_mq$a_mean, post_ms_mq$logb_mean,
                        post_ms_mq$s_mean, post_ms_mq$z_mean,
                        a_msmc, logb_msmc, logs_msmc, z_msmc)
scatterp_title_labels <- paste0("(", letters[1:20], ")")
scatterp_titles <- rep(c("HMC-NUTS", "Laplace (MQ)", "Laplace (MCMC)",
                         "Max-and-Smooth (MQ)", "Max-and-Smooth (MCMC)"),
                       each=4)
scatterp_xlab_list <- rep(list("True a", "True b", "True s", true_z10_lab),
                          5)
scatterp_ylab_list <- rep(list("Estimated a", "Estimated b", "Estimated s",
                               est_z10_lab), 5)

scatterplot_list <- lapply(1:20, function(i) {
  true_vs_est_scatterplot(
    true = scatterp_x_list[[i]],
    est = scatterp_y_list[[i]],
    axis_lim = scatterp_range_list[[i]],
    title = paste(scatterp_title_labels[i], scatterp_titles[i]),
```

```
    x_lab = scatterp_xlab_list[[i]],
    y_lab = scatterp_ylab_list[[i]],
    theme = mytheme
  )
})
ggarrange(plotlist=scatterplot_list, nrow=5, ncol=4)
```

Figure 5 displays the posteriors of the spatial hyperparameters as density plots for the five methods considered in the paper.

```
# Parameter estimate uncertainty for hyperparameters
hist_df <- data.frame(beta_a=c(sam_lap_mq$parameter_draws[,"beta_a"],
                               sams_betaa_lapmc,
                               sams_betaa_msmc,
                               sam_ms_mq$parameter_draws[,"beta_a"],
                               sams_betaa_hmc),
                      beta_b=c(sam_lap_mq$parameter_draws[,"beta_b"],
                               sams_betab_lapmc,
                               sams_betab_msmc,
                               sam_ms_mq$parameter_draws[,"beta_b"],
                               sams_betab_hmc),
                      beta_s=c(sam_lap_mq$parameter_draws[,"beta_s"],
                               sams_betas_lapmc,
                               sams_betas_msmc,
                               sam_ms_mq$parameter_draws[,"beta_s"],
                               sams_betas_hmc),
                      log_sigma_a=c(sam_lap_mq$parameter_draws[,"log_sigma_a"],
                                    sams_siga_lapmc,
                                    sams_siga_msmc,
                                    sam_ms_mq$parameter_draws[,"log_sigma_a"],
                                    sams_siga_hmc),
                      log_kappa_a=c(sam_lap_mq$parameter_draws[,"log_kappa_a"],
                                    sams_kapa_lapmc,
                                    sams_kapa_msmc,
                                    sam_ms_mq$parameter_draws[,"log_kappa_a"],
                                    sams_kapa_hmc),
                      log_sigma_b=c(sam_lap_mq$parameter_draws[,"log_sigma_b"],
                                    sams_sigb_lapmc,
                                    sams_sigb_msmc,
                                    sam_ms_mq$parameter_draws[,"log_sigma_b"],
                                    sams_sigb_hmc),
                      log_kappa_b=c(sam_lap_mq$parameter_draws[,"log_kappa_b"],
                                    sams_kapb_lapmc,
                                    sams_kapb_msmc,
                                    sam_ms_mq$parameter_draws[,"log_kappa_b"],
                                    sams_kapb_hmc),
                      log_sigma_s=c(sam_lap_mq$parameter_draws[,"log_sigma_s"],
                                    sams_sigs_lapmc,
                                    sams_sigs_msmc,
                                    sam_ms_mq$parameter_draws[,"log_sigma_s"],
                                    sams_sigs_hmc),
                      log_kappa_s=c(sam_lap_mq$parameter_draws[,"log_kappa_s"],
                                    sams_kaps_lapmc,
```

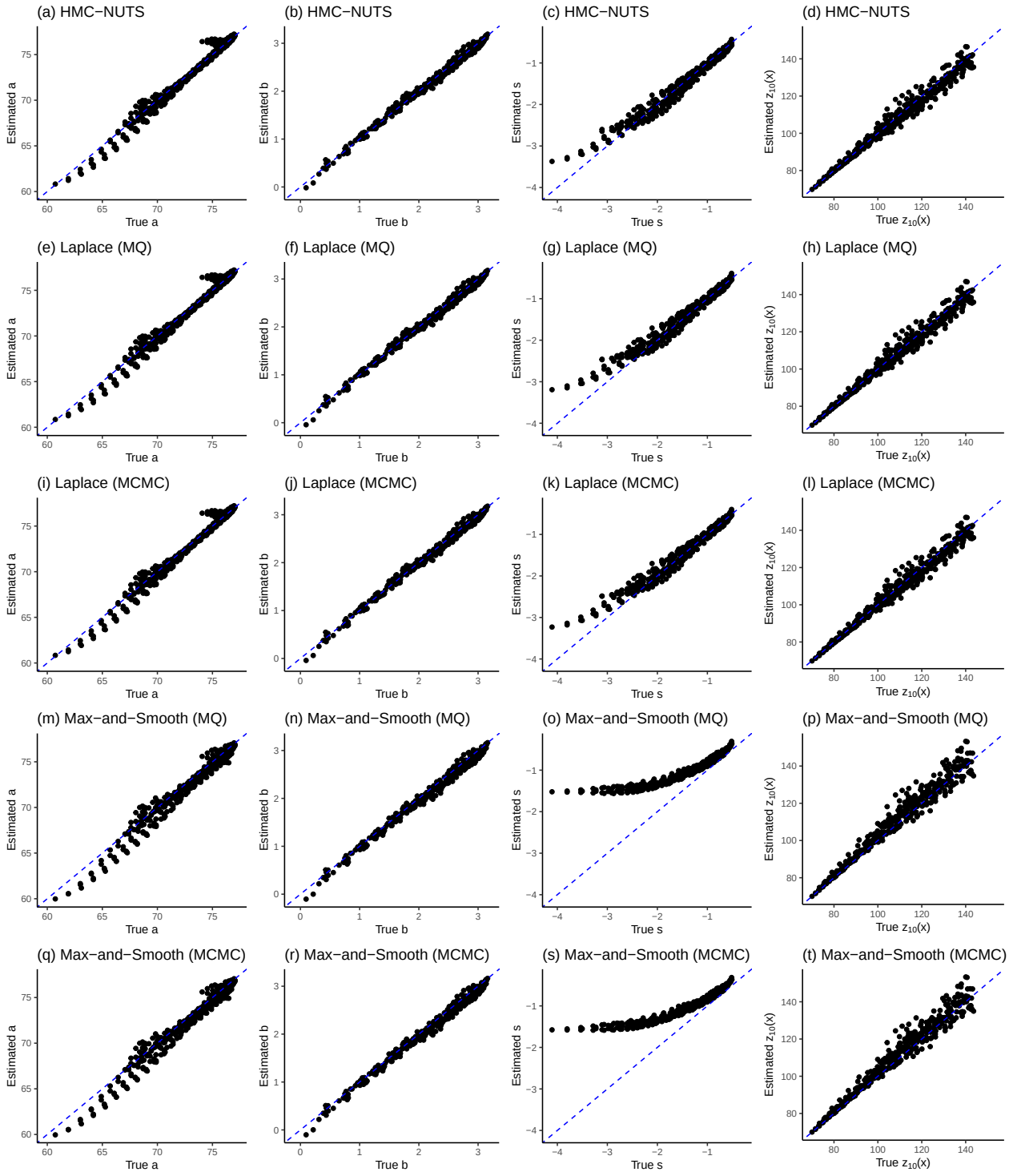

Figure 4: True vs posterior mean of random effects and return levels for various estimators.

```
                                    sams_kaps_msmc,
                                    sam_ms_mq$parameter_draws[,"log_kappa_s"],
                                    sams_kaps_hmc),
                        Method=c(rep("Laplace (MQ)", dim(sam_lap_mq$parameter_draws)[1]),
                                 rep("Laplace (MCMC)", length(sams_betaa_lapmc)),
                                 rep("Max-Smooth (MCMC)", length(sams_betaa_msmc)),
                                 rep("Max-Smooth (MQ)", dim(sam_ms_mq$parameter_draws)[1]),
                                 rep("HMC-NUTS",length(sams_betaa_hmc)))
                        )

hist_labels <- c("beta_a", "log_sigma_a", "log_kappa_a",
                      "beta_b", "log_sigma_b", "log_kappa_b",
                      "beta_s", "log_sigma_s", "log_kappa_s")
hist_xlab_list <- list(expression(beta[a]), expression(log(sigma[a]^2)),
                       expression(log(kappa[a])),
                       expression(beta[b]), expression(log(sigma[b]^2)),
                       expression(log(kappa[b])),
                       expression(beta[s]), expression(log(sigma[s]^2)),
                       expression(log(kappa[s])))
hist_title_list <- list(expression("(a) Posterior distribution for"~beta[a]),
                      expression("(b) Posterior distribution for"~log(sigma[a]^2)),
                      expression("(c) Posterior distribution for"~log(kappa[a])),
                      expression("(d) Posterior distribution for"~beta[b]),
                      expression("(e) Posterior distribution for"~log(sigma[b]^2)),
                      expression("(f) Posterior distribution for"~log(kappa[b])),
                      expression("(g) Posterior distribution for"~beta[s]),
                      expression("(h) Posterior distribution for"~log(sigma[s]^2)),
                      expression("(i) Posterior distribution for"~log(kappa[s])))
hist_xlim_list <- list(c(0, 120), c(-2, 10), c(-6, 0),
                       c(-5, 10), c(-2, 10), c(-6, 0),
                       c(-4.5, 2.5), c(-4, 8), c(-6, 0))

mytheme <- create_ggplot_theme(14)
histplot_list <- lapply(1:9, function(i){
  ggplot(hist_df, aes(x=get(hist_labels[i]), color=Method)) +
    geom_density(size=1.5) + xlim(hist_xlim_list[[i]]) +
    xlab(hist_xlab_list[[i]]) + mytheme +
    ggtitle(hist_title_list[[i]])
})

ggarrange(plotlist=histplot_list, ncol=3, nrow=3,
          common.legend = TRUE)
```

We check the relative differences between our Laplace posterior means and HMC-NUTS posterior means in Figure 6.

```
par(mfrow=c(2,2), mar=c(5, 5, 4, 6))
grid_plot(small_sim$lon, small_sim$lat, matrix((post_lap_mq$a_mean-a_hmc)/a_hmc,
                                               ncol=sqrt(n_loc)),
          title="Rel. diff. between Laplace and HMC-NUTS for a", cex=1.1)
grid_plot(small_sim$lon, small_sim$lat, matrix((post_lap_mq$logb_mean-logb_hmc)/logb_hmc,
                                               ncol=sqrt(n_loc)),
          title="Rel. diff. between Laplace and HMC-NUTS for b", cex=1.1)
```

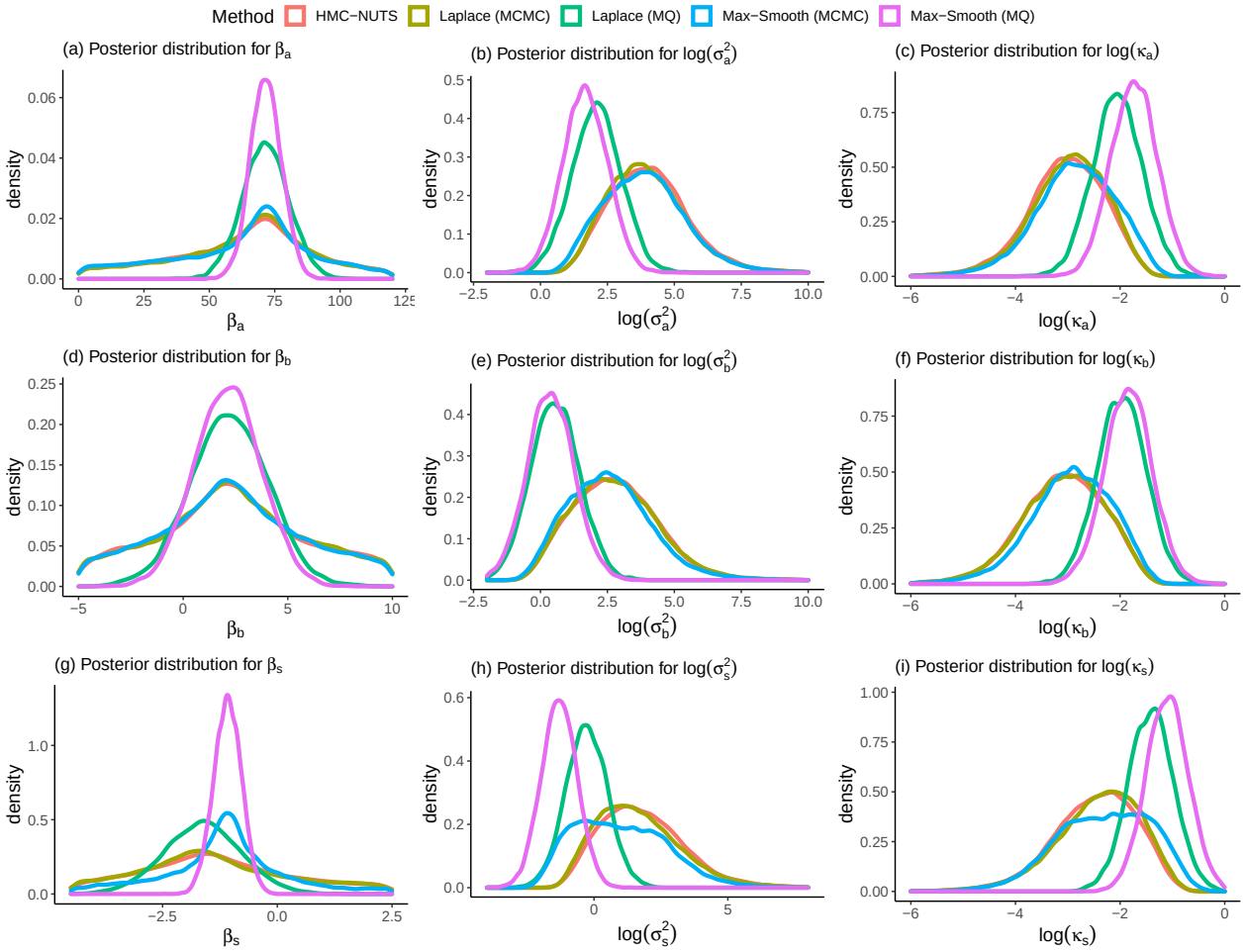

Figure 5: Posterior distributions of the hyperparameters.

```
grid_plot(small_sim$lon, small_sim$lat, matrix((post_lap_mq$s_mean-logs_hmc)/logs_hmc,
                                               ncol=sqrt(n_loc)),
          title="Rel. diff. between Laplace and HMC-NUTS for s", cex=1.1)
```

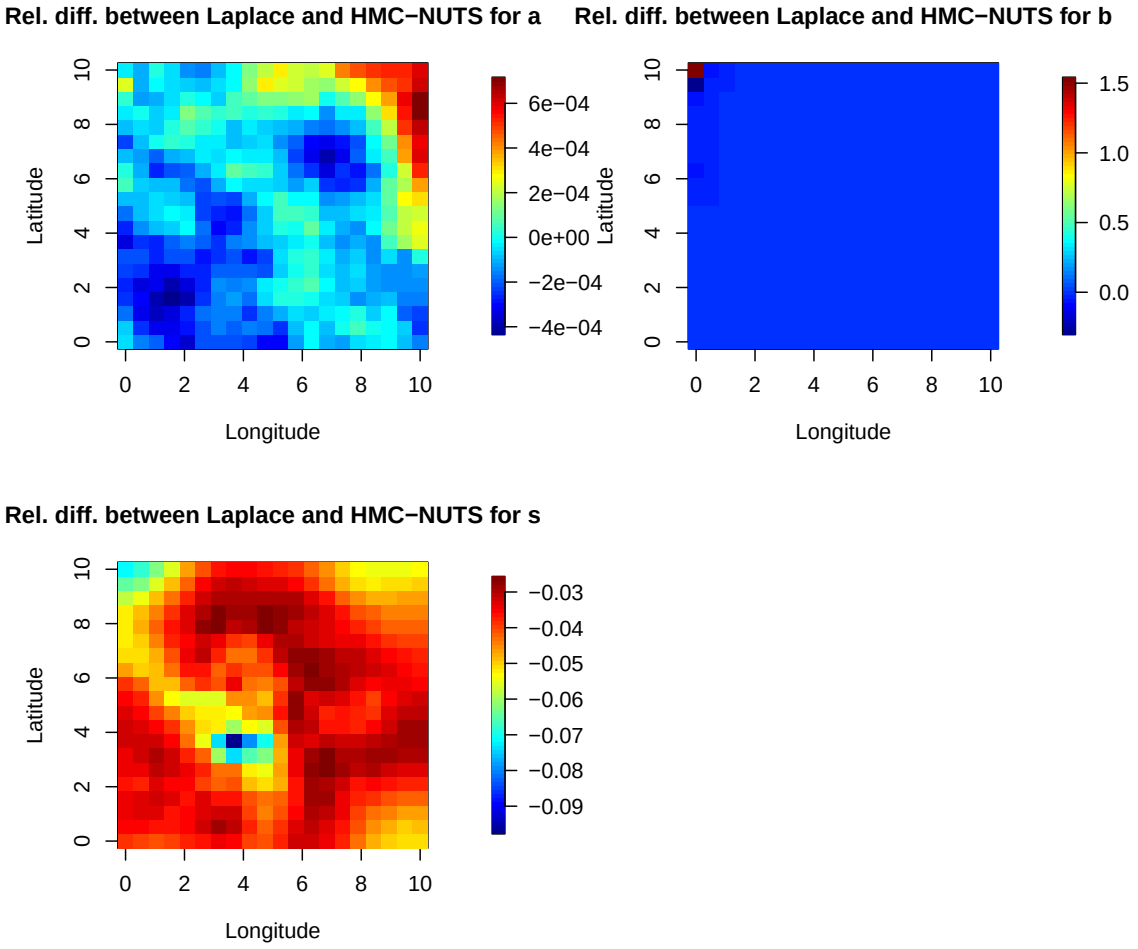

Figure 6: Relative difference between Laplace posterior means and HMC-NUTS posteriors means.

Finally, we plot the difference between true and estimated a, b, s, and z_{10} on heatmaps in Figure 7.

```
par(mfrow=c(2,2), mar=c(5,5,4,6))
grid_plot(small_sim$lon, small_sim$lat, matrix(post_lap_mq$a_mean-small_sim$a,
                                               ncol=sqrt(n_loc)),
          title="Absolute error in a", cex=1.1)
grid_plot(small_sim$lon, small_sim$lat, matrix(post_lap_mq$logb_mean-small_sim$logb,
                                               ncol=sqrt(n_loc)),
          title="Absolute error in b", cex=1.1)
grid_plot(small_sim$lon, small_sim$lat, matrix(post_lap_mq$s_mean-small_sim$logs,
                                               ncol=sqrt(n_loc)),
          title="Absolute error in s", cex=1.1)
grid_plot(small_sim$lon, small_sim$lat, matrix(post_lap_mq$z_mean-small_sim$z_true,
                                               ncol=sqrt(n_loc)),
          title="Absolute error in z", cex=1.1)
```

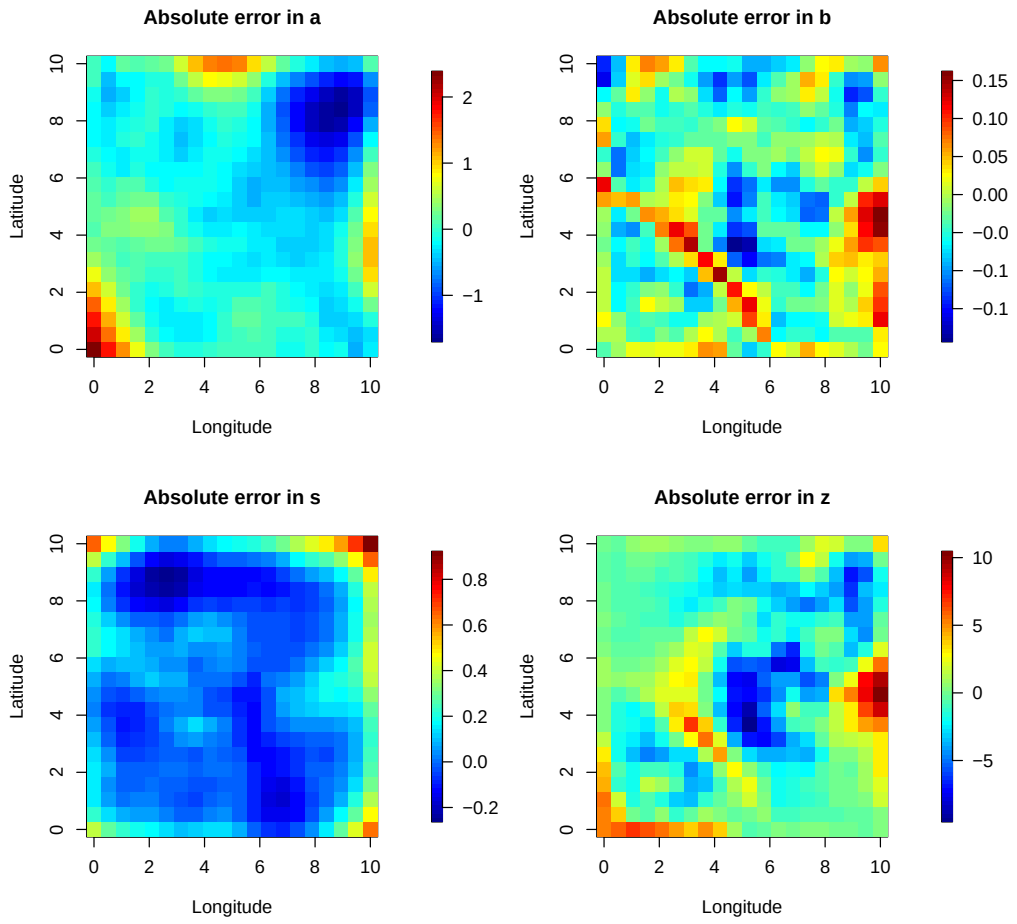

Figure 7: Estimation errors for the random effects and the return levels.

3 Large-scale simulation study

In this section, we consider a much larger dataset that consists of 6,400 locations, compared to 400 locations in the small-scaled simulation study. Moreover, the spatial variation of each GEV parameter is less smooth compared to the small simulation, as shown in Figure 8. The Delta method is used to compute the posterior mean and standard deviation of each random effect and the return levels $z_{10}(\boldsymbol{x})$.

First, simulate data on an 80×80 grid:

```
set.seed(222)
big_sim <- simulate_data_big()

# Plot simulated data on maps
par(mfrow=c(1,3))
grid_plot(big_sim$lon, big_sim$lat, big_sim$a_mat,
          title="a", cex=1.1)
grid_plot(big_sim$lon, big_sim$lat, big_sim$logb_mat,
          title="b", cex=1.1)
grid_plot(big_sim$lon, big_sim$lat, big_sim$logs_mat,
          title="s", cex=1.1)
```

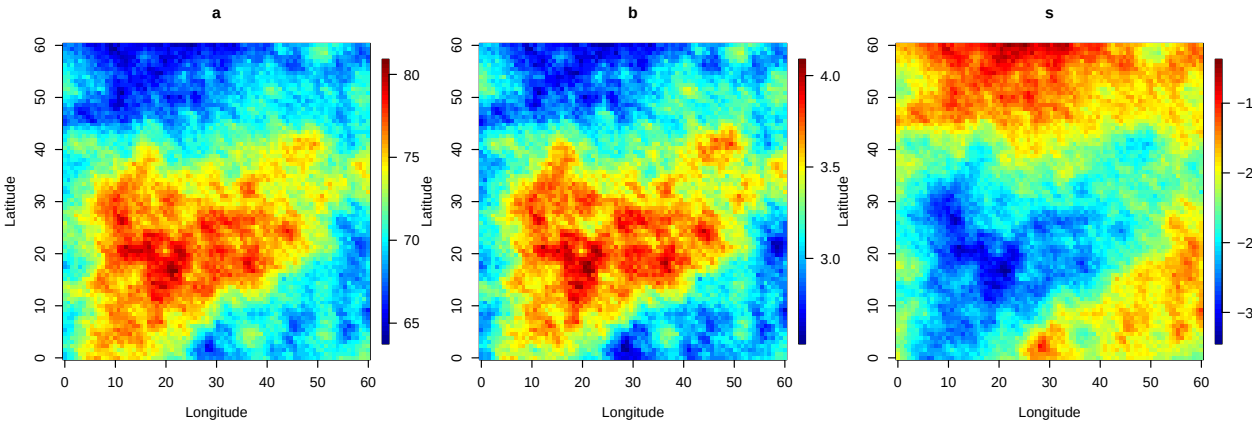

Figure 8: Simulated random effects plotted on regular lattices

3.1 Laplace Method via joint Normal approximation (Laplace-MQ)

Due to time and memory limitations, the only posterior method which could scale up to this large simulation study was Laplace-MQ. Even as such, we could not compute the posterior distribution for $z_{10}(\boldsymbol{x})$ via simulation as we did in the small-scale study, as this would require sampling from the $6400 \times 3 = 19,200$ dimensional posterior distirbution of the random effects. Instead, a Delta-method is used to approximate the posterior distribution of z_{10} using far less computation time and memory.

The Laplace-MQ method is fit using `spatialGEV_fit()` in the code below. Initial parameters and priors for the model were the same as in the small-scale study.

Note: This code takes 22 hours to run and requires 90Gb of memory.

```
# Initial values for the parameters
n_loc <- nrow(big_sim$locs)
init_param <- list(
```

```
  a = rep(60, n_loc),
  log_b = rep(3, n_loc),
  s = rep(-2,n_loc),
  beta_a = 60, beta_b = 3, beta_s = -2,
  log_sigma_a = -1, log_kappa_a = -1,
  log_sigma_b = -1, log_kappa_b = -1,
  log_sigma_s = -1, log_kappa_s = -1
)
# Weakly informative priors on beta
beta_prior <- list(beta_a=c(0,100), beta_b=c(0,50), beta_s=c(0,20))

# Model fitting
t_start <- Sys.time()
fit_big <- spatialGEV_fit(data=big_sim$data, locs=big_sim$locs, random = "abs",
                          init_param = init_param,
                          reparam_s = "positive", kernel="spde",
                          beta_prior = beta_prior, silent = TRUE,
                          return_level = TRUE, get_hessian = FALSE)
bigsim_runtime <- difftime(Sys.time(), t_start, units="secs")
report_big <- fit_big$report
meshidxloc_big <- fit_big$meshidxloc
```

```
a_ind_big <- meshidxloc_big
logb_ind_big <- length(report_big$par.random)/3+meshidxloc_big
logs_ind_big <- length(report_big$par.random)/3*2+meshidxloc_big
a_s <- report_big$par.random[a_ind_big]
logb_s <- report_big$par.random[logb_ind_big]
logs_s <- report_big$par.random[logs_ind_big]
z_s <- report_big$value
```

3.2 Figures for the large-scale simulation study

Figure 9 (a)-(d) displays the posterior mean estimates of the random effects against their corresponding true values. Figure 9 (e)-(h) examines the accuracy of the posterior uncertainty of the Laplace-MQ approximation. Without a baseline comparison to the exact posterior distribution, this is assessed by computing at each location i the z-score

$$\mathcal{Z}(\gamma_i) = \frac{\hat{\gamma}_i - \gamma_i}{\widehat{\mathrm{sd}}(\gamma_i)},$$

where γ_i is the true value at location i of the quantity $\gamma \in \{a, b, s, z_{10}\}$, $\hat{\gamma}_i$ is the estimate of its posterior mean, and $\widehat{\mathrm{sd}}(\gamma_i)$ is the estimate of its posterior standard deviation with Laplace-MQ. If the posterior uncertainty is both accurate and accurately estimated, then these z-scores should be close to a 45 degree line on a QQ plot. While there are systematic departures from this line at the lower extreme for b and s, and the upper extreme for z_{10}, the QQ plots indicated that posterior uncertainty is overall well captured by the Laplace-MQ method.

```
a_range <- range(c(big_sim$a, a_s))
b_range <- range(c(big_sim$logb, logb_s))
s_range <- range(c(big_sim$logs, logs_s))
z_range <- range(c(big_sim$z_true, z_s))

mytheme <- create_ggplot_theme(text_size=10, title_size = 12)
```

```r
big_a_est <- true_vs_est_scatterplot(
  big_sim$a, a_s,
  axis_lim = a_range,
  title = expression("(a) Estimated vs True z-score for"~a),
  x_lab = "True z-score",
  y_lab = "Estimated z-score",
  theme = mytheme
)
big_b_est <- true_vs_est_scatterplot(
  big_sim$logb, logb_s,
  axis_lim = b_range,
  title = expression("(b) Est. vs True z-score for"~b),
  x_lab = "True z-score",
  y_lab = "Estimated z-score",
  theme = mytheme
)
big_s_est <- true_vs_est_scatterplot(
  big_sim$logs, logs_s,
  axis_lim = s_range,
  title = expression("(c) Est. vs True z-score for"~s),
  x_lab = "True z-score",
  y_lab = "Estimated z-score",
  theme = mytheme
)
big_z_est <- true_vs_est_scatterplot(
  big_sim$z_true, z_s,
  axis_lim = z_range,
  title = expression("(d) Est. vs True z-score for"~z[10]),
  x_lab = "True z-score",
  y_lab = "Estimated z-score",
  theme = mytheme
)

param_sd <- summary(report_big, "random")
a_sd <- param_sd[a_ind_big, 2]
logb_sd <- param_sd[logb_ind_big, 2]
logs_sd <- param_sd[logs_ind_big, 2]
z_sd <- report_big$sd
zscore_a <- (big_sim$a - a_s)/a_sd
zscore_b <- (big_sim$logb - logb_s)/logb_sd
zscore_s <- (big_sim$logs - logs_s)/logs_sd
zscore_z <- (big_sim$z_true - z_s)/z_sd

big_qq_a <- qq_plot(
  zscore = zscore_a,
  title = expression("(e) QQ plot for z-score of"~a),
  x_lab = "Theoretical Quantile",
  y_lab = "Sample Quantile",
  theme = mytheme
)
big_qq_b <- qq_plot(
  zscore = zscore_b,
  title = expression("(f) QQ plot for z-score of"~b),
```

```
  x_lab = "Theoretical Quantile",
  y_lab = "Sample Quantile",
  theme = mytheme
)
big_qq_s <- qq_plot(
  zscore = zscore_s,
  title = expression("(g) QQ plot for z-score of"~s),
  x_lab = "Theoretical Quantile",
  y_lab = "Sample Quantile",
  theme = mytheme
)
big_qq_z <- qq_plot(
  zscore = zscore_z,
  title = expression("(h) QQ plot for z-score of"~z[10]),
  x_lab = "Theoretical Quantile",
  y_lab = "Sample Quantile",
  theme = mytheme
)
ggarrange(big_a_est, big_b_est, big_s_est, big_z_est,
          big_qq_a, big_qq_b, big_qq_s, big_qq_z,
          ncol=4, nrow=2)
```

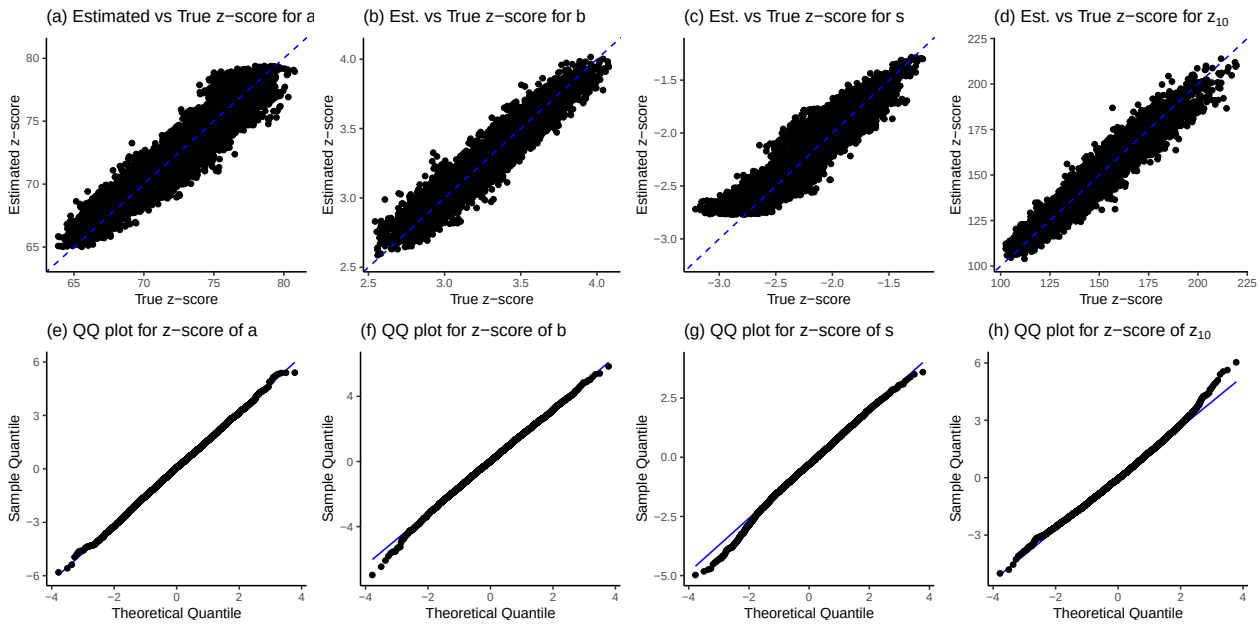

Figure 9: (a)-(d): True vs posterior mean of random effects and return levels. (e)-(h): QQ plots of z-scores for random effects and return levels.

4 Case study on monthly snowfall

The case study analyzes the monthly total snowfall data in Canada from January 1987 to December 2021. We load the preprocessed snowfall data `CAsnow` from our package **SpatialGEV**, create a spatial mesh for SPDE approximation, and plot it on the map of Canada in Figure 10.

```
data("CA-snowdata")
n_loc <- nrow(CAsnow$locs)
bnd <- inla.nonconvex.hull(as.matrix(CAsnow$locs), convex = -0.025, resolution = c(49,25))
cutoff <- 0.5
max.edge <- 2
mesh <- inla.mesh.2d(CAsnow$locs, boundary = bnd,
                     cutoff = cutoff, max.edge = max.edge)
par(mar=c(0.5,1,0.5,1))
plot(mesh, main="")
points(CAsnow$locs$cell_lon, CAsnow$locs$cell_lat, pch=20, cex=0.8, col="red")
maps::map("world", "Canada", add=TRUE)
```

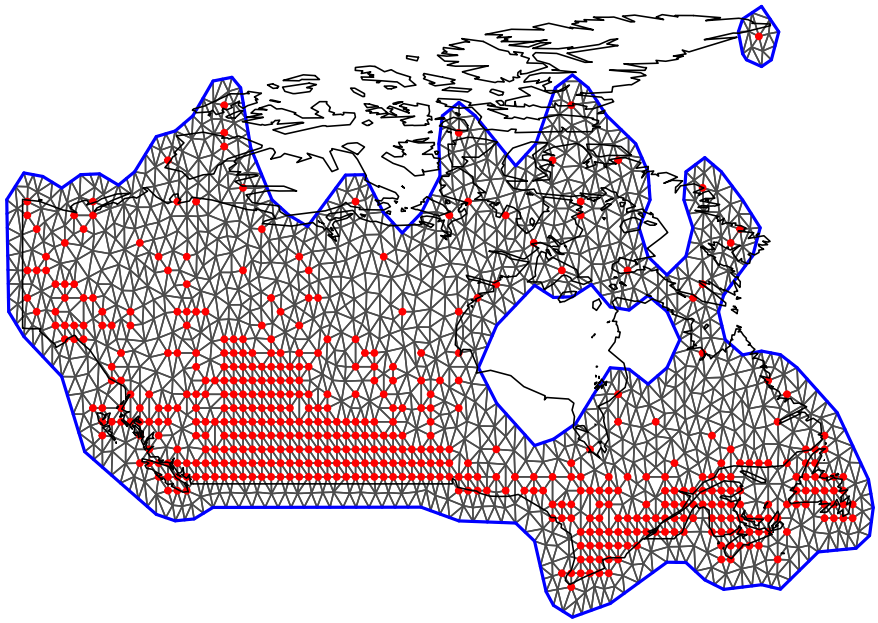

Figure 10: 509 locations at which at least 10 years of data was recorded after gridding

In Figure 11, the probability density function and cumulative density function plots of data pooled across from all time points and locations show a heavy tail. This provides evidence for using the GEV distribution to model these observations.

```
snow_pdf <- ggplot(mapping=aes(x = unlist(CAsnow$Y))) +
  geom_histogram(aes(y = after_stat(density)),
                 colour = 1, fill = "grey",
                 bins=50) +
  geom_density()+
  xlab("Maximum yearly records of monthly total snowfall (in cm)")

snow_cdf <- ggplot(mapping=aes(x = unlist(CAsnow$Y))) +
  stat_ecdf() +
  xlab("Maximum yearly records of monthly total snowfall (in cm)")
ggarrange(snow_pdf, snow_cdf, ncol=2)
```

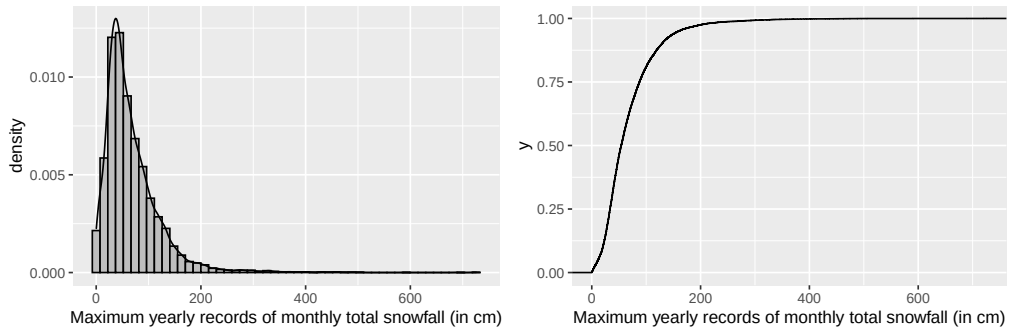

Figure 11: PDF and CDF of all 509 extreme value observations pooled across locations and time.

4.1 Model fitting

We fit two different models to the data: `fit_a` which assumes only the location parameter a is random, and `fit` which is the one we eventually adopted in the paper and which assumes a and b are random but s is a fixed effect.

```
# Model a: Only a random
fit_a <- spatialGEV_fit(data=CAsnow$Y, locs=CAsnow$locs, random="a",
                        init_param = list(a=rep(40, n_loc),
                                          log_b=3,
                                          s=-2,
                                          beta_a=30,
                                          log_sigma_a=1, log_kappa_a=-3),
                      reparam_s="positive",
                      beta_prior=list(beta_a=c(0,100)),
                      kernel="spde",
                      s_prior=c(-5,3),
                      silent=FALSE,
                      boundary=bnd, cutoff=cutoff, max.edge=max.edge)
sam_a <- spatialGEV_sample(fit_a, n_draw=5000, observation=TRUE)
# Posterior predictive quantities
y_rep_90_a <- apply(sam_a$y_draws, 2, quantile, probs=0.9) # 90% posterior pred
y_rep_50_a <- apply(sam_a$y_draws, 2, quantile, probs=0.5) # 50% posterior pred

# Model ab: Only a and b random
fit <- spatialGEV_fit(data=CAsnow$Y, locs=CAsnow$locs, random="ab",
                      init_param = list(a=rep(40, n_loc),
                                        log_b=rep(3, n_loc),
                                        s=-2,
                                        beta_a=30, beta_b=3,
                                        log_sigma_a=1, log_kappa_a=-3,
                                        log_sigma_b=-2, log_kappa_b=-3),
                      beta_prior=list(beta_a=c(0,100),beta_b=c(0,100)),
                      reparam_s="positive",
                      kernel="spde",
                      s_prior=c(-5,5),
                      silent=FALSE,
                      boundary=bnd, cutoff = cutoff, max.edge = max.edge)
sam <- spatialGEV_sample(fit, n_draw=5000, observation=TRUE)
# Posterior predictive quantities
```

```
y_rep_90 <- apply(sam$y_draws, 2, quantile, probs=0.9) # 90% posterior pred
y_rep_50 <- apply(sam$y_draws, 2, quantile, probs=0.5) # 50$ posterior pred
```

4.2 Model selection

Here we show the posterior prediction check procedure described in the paper chooses model M_{ab} over model M_a. We first find the true 90% and 50% quantile at each location of the data.

```
y_true_90 <- sapply(CAsnow$Y, quantile, probs=0.9)
y_true_50 <- sapply(CAsnow$Y, quantile, probs=0.5)
```

Next, samples from the posterior predictive distribution for each model are already obtained by calling `spatialGEV_sample()`. By setting `observation=TRUE` the function samples from the posterior predictive distribution in addition to the parameter posteriors. Figure 12 displays the posterior predictive median and upper 10% quantile from model M_{ab} plotted versus the corresponding observed values at each location.

```
# Plot theme
mytheme <- create_ggplot_theme(text_size=10)
pred_ab_50 <- pospred_vs_obs_plot(y_true_50, y_rep_50, "median")+mytheme
pred_ab_90 <- pospred_vs_obs_plot(y_true_90, y_rep_90, "upper 10% quantile")+mytheme
ggarrange(pred_ab_50, pred_ab_90)
```

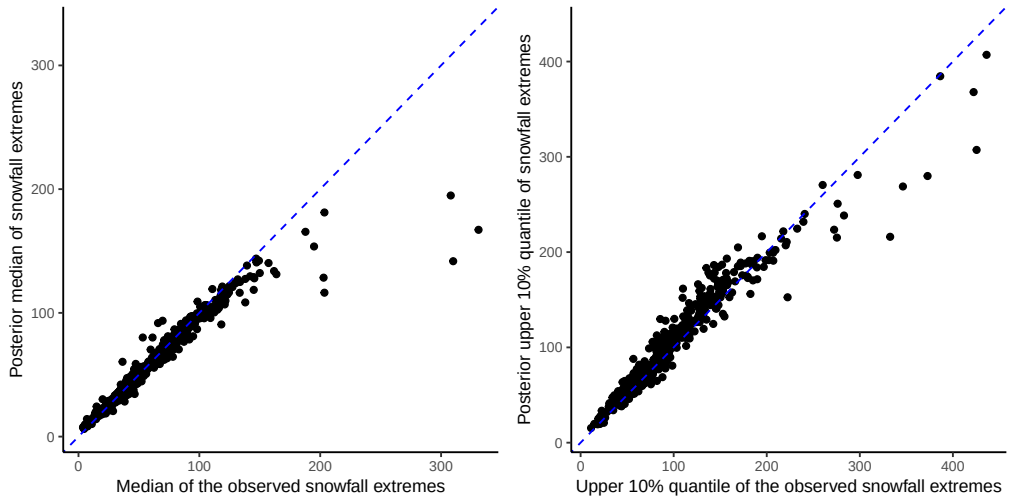

Figure 12: Posterior predicted versus empirical test statistics for model M_{ab}.

Similarly, we check the posterior predictive performance of model M_a. Figure 13 shows that model M_a performs worse than model M_{ab}.

```
pred_a_50 <- pospred_vs_obs_plot(y_true_50, y_rep_50_a, "median")+mytheme
pred_a_90 <- pospred_vs_obs_plot(y_true_90, y_rep_90_a, "upper 10% quantile")+mytheme
ggarrange(pred_a_50, pred_a_90, ncol=2)
```

4.3 Parameter inference

We obtain the parameter estimates from the posterior samples of a, b and s, and the posterior samples of z_{10} by applying the appropriate transformation of a, b and s.

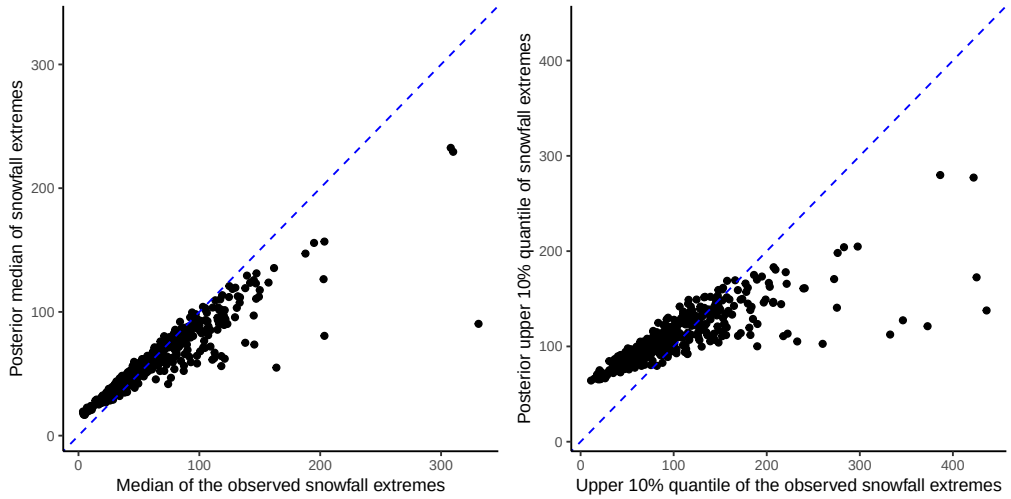

Figure 13: Posterior predicted versus empirical test statistics for model M_a.

```
case_pos_estimates <- spatialGEV_get_posteriors(fit, sam)
a_mean <- case_pos_estimates$a_mean
logb_mean <- case_pos_estimates$logb_mean
logs_mean<- case_pos_estimates$s_mean
return10 <- case_pos_estimates$z_mean
a_sd <- case_pos_estimates$a_sd
logb_sd <- case_pos_estimates$logb_sd
logs_sd <- case_pos_estimates$s_sd
return10sd <- case_pos_estimates$z_sd
```

Finally, the posterior means and standard deviations of $a(\boldsymbol{x})$, $b(\boldsymbol{x})$, and $z_{10}(\boldsymbol{x})$ are plotted in Figures 14, 15, and 16, where locations with high posterior uncertainty are marked.

```
par(mfrow=c(1,2))
map_plot(a_mean, zlim=range(a_mean),
         lon=CAsnow$locs$cell_lon, lat=CAsnow$locs$cell_lat, title="Mean of a")
map_plot(a_sd, zlim=range(a_sd),
         lon=CAsnow$locs$cell_lon, lat=CAsnow$locs$cell_lat, title="SD of a")
```

```
par(mfrow=c(1,2))
map_plot(logb_mean, zlim=range(logb_mean),
         lon=CAsnow$locs$cell_lon, lat=CAsnow$locs$cell_lat, title="Mean of b")
map_plot(logb_sd, zlim=range(logb_sd),
         lon=CAsnow$locs$cell_lon, lat=CAsnow$locs$cell_lat, title="SD of b")
```

```
# Mark the three example locations in the paper
loc1 <- which(CAsnow$locs[,1]==-80.5 & CAsnow$locs[,2]==43.5)
loc2 <- which(CAsnow$locs[,1]==-117.5 & CAsnow$locs[,2]==51.5)
loc3 <- which(CAsnow$locs[,1]==-130.5 & CAsnow$locs[,2]==56.5)
par(mfrow=c(1,2))
map_plot(return10, zlim=range(return10),
         lon=CAsnow$locs$cell_lon, lat=CAsnow$locs$cell_lat, title="Mean of z10")
```

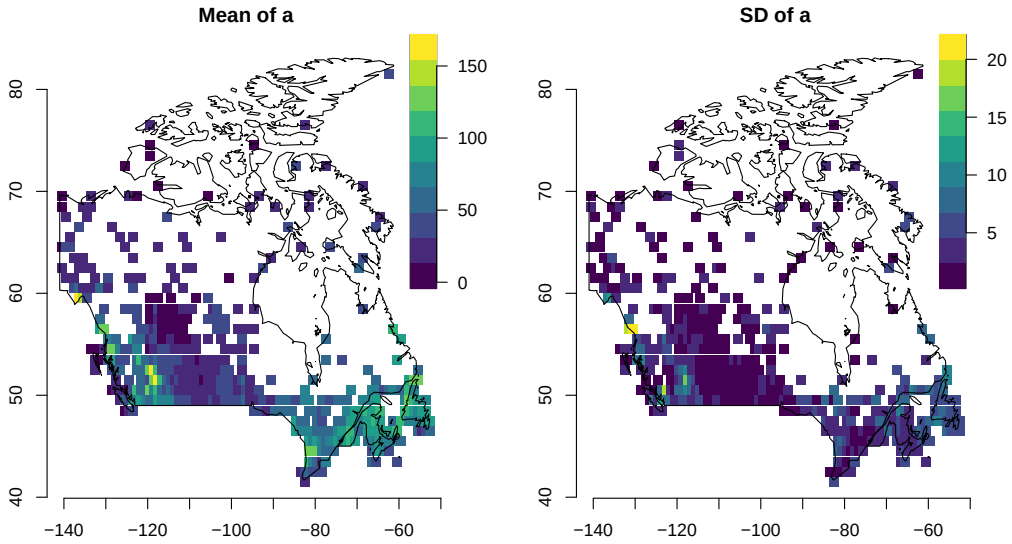

Figure 14: Posterior means and SDs for $a(\boldsymbol{x})$.

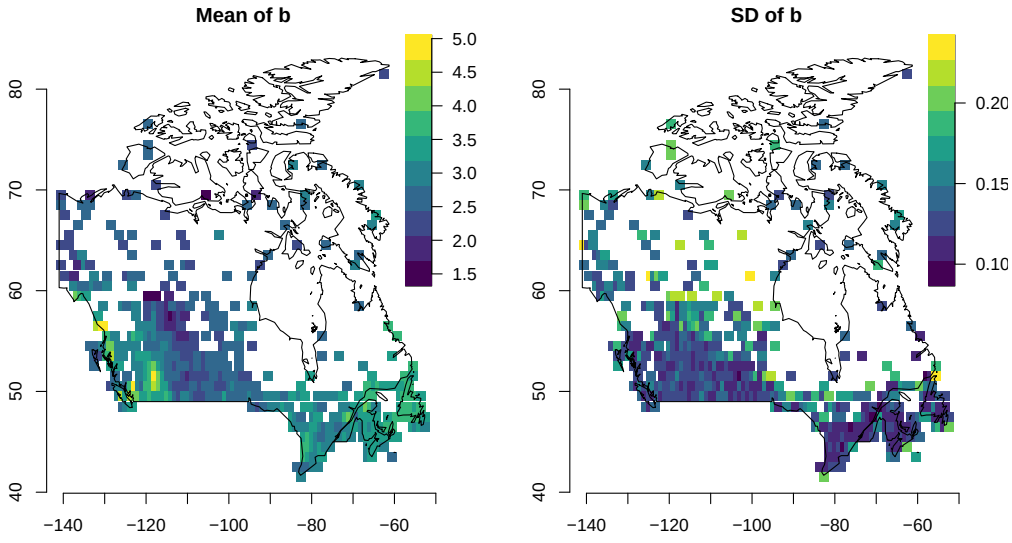

Figure 15: Posterior means and SDs for $b(\boldsymbol{x})$.

```
points(CAsnow$locs[loc1,], cex=2, col="red", lwd=2.5)
points(CAsnow$locs[loc2,], cex=2, col="red", lwd=2.5, pch=2)
points(CAsnow$locs[loc3,], cex=2, col="red", lwd=2.5, pch=5)
map_plot(return10sd, zlim=range(return10sd),
         lon=CAsnow$locs$cell_lon, lat=CAsnow$locs$cell_lat, title="SD of z10")
points(CAsnow$locs[loc1,], cex=2, col="red", lwd=2.5)
points(CAsnow$locs[loc2,], cex=2, col="red", lwd=2.5, pch=2)
points(CAsnow$locs[loc3,], cex=2, col="red", lwd=2.5, pch=5)
```

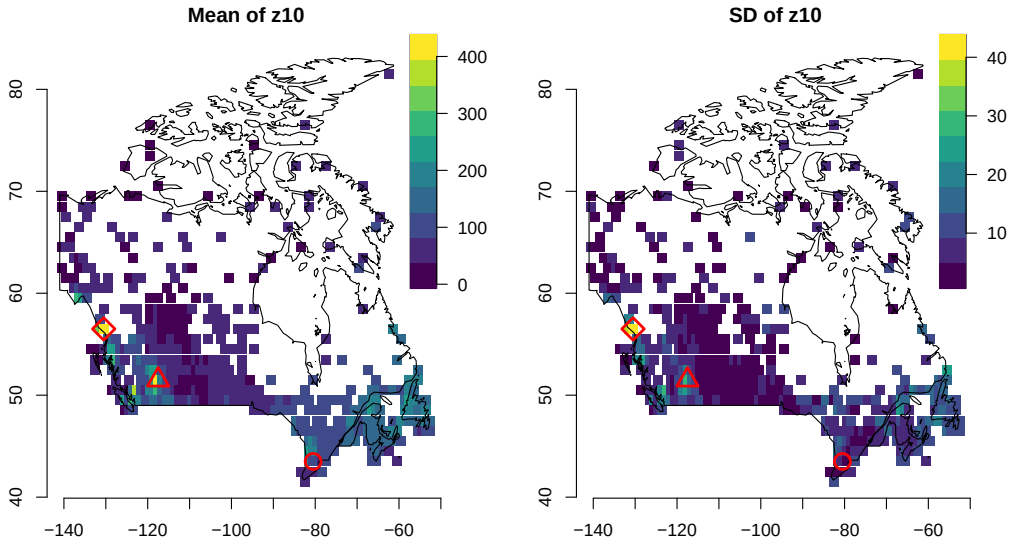

Figure 16: Posterior means and SDs for $z_{10}(\boldsymbol{x})$.

5 Comparison with Max-and-Smooth implementation of Jóhannesson et al. (2022)

The Max-and-Smooth implementation of Jóhannesson et al. (2022) – henceforth referred to as J22 – is an MCMC algorithm on the Max-and-Smooth approximate posterior similar to the Maxsmooth-MCMC method we describe in Section 2.4. The main difference between the J22 implementation and ours is that J22 adopts a different parametrization (ψ, τ, ϕ) for the GP models:

$$\psi = \log(a), \quad \tau = \log(b_o/s_o), \quad \phi = h(s_o), \tag{3}$$

where

$$h(s_o) = a_\phi + b_\phi \log\left(-\log\{1 - (s + \tfrac{1}{2})^{c_\phi}\}\right),$$

with $a_\phi = 0.062376$, $b_\phi = 0.39563$ and $c_\phi = 0.8$. We refer to Jóhannesson et al. (2022) for how this parameterization is justified and constructed, but note that (3) restricts the shape parameter of the GEV model to $|s_o| < 0.5$. We compare estimation results of J22 to our Laplace-MQ approximation using the J22 parametrization (3). The **TMB** template for this model can be found in the **SpatialGEV** package in `src/TMB/model_ptp_spde.hpp`.

We now regenerate the small-scale simulation study data to be fit by the Laplace-MQ and J22 methods.

```r
set.seed(123)
small_sim <- simulate_data_small()
```

5.1 Laplace method with J22 parametrization (Laplace-MQ)

Model fitting using the Laplace-MQ method with the J22 parametrization (3) is exactly as described in Section 2.1, except that the GPs are now on $\psi(\boldsymbol{x})$, $\tau(\boldsymbol{x})$ and $\phi(\boldsymbol{x})$. However, the small-scale simulation data contained 43 out of 400 locations with $s_o > 0.5$, which are out of the range of the J22 parametrization (3). These locations are thus excluded from the analyses below. We adopt the same hyperparameter priors used in J22, i.e., Normal priors on $(\boldsymbol{\beta}_\psi, \boldsymbol{\beta}_\tau, \boldsymbol{\beta}_\phi)$ and Penalized Complexity priors on the Matérn range and standard deviation.

```r
set.seed(123)
true_theta <- paper2j22_transform(small_sim$a, small_sim$logb, exp(small_sim$logs))
```

```
## Warning in log(1 - (s + 0.5)^alp3): NaNs produced
```

```r
psi <- true_theta$psi
tau <- true_theta$tau
phi <- true_theta$phi

# remove shape > 0.5 not supported by the parameterization in J22
rm_loc_ind <- which(is.na(phi))
psi <- psi[-rm_loc_ind]
tau <- tau[-rm_loc_ind]
phi <- phi[-rm_loc_ind]

data_na_rm <- small_sim$data[-rm_loc_ind]
n_obs <- sapply(data_na_rm, length)
loc_ind <- rep(1:length(data_na_rm), times=n_obs)
```

```r
lap_start_t <- Sys.time() # begin timer
# Create mesh
coords <- as.matrix(small_sim$locs[-rm_loc_ind, ])
mesh <- inla.mesh.2d(
  loc=coords,
  max.edge = 2)
n_s <- mesh$n
X_psi <- X_tau <- X_kappa <- matrix(rep(1, n_s), nrow=n_s, ncol=1)
meshidxloc <- as.integer(mesh$idx$loc)

data <- list(model="model_ptp_spde",
             y=unlist(data_na_rm),
             loc_ind=meshidxloc[loc_ind]-1,
             design_mat_psi=X_psi,
             design_mat_tau=X_tau,
             design_mat_phi=X_kappa,
             nu=1,
             spde=(INLA::inla.spde2.matern(mesh)$param.inla)[c("M0", "M1", "M2")],
             beta_prior=as.integer(1),
             beta_psi_prior=c(0,10000),    # Normal prior on beta coeff for psi
```

```
              beta_tau_prior=c(0,10000),
              beta_phi_prior=c(0,10000),
              psi_pc_prior=as.integer(1),
              range_psi_prior=c(0.5, 0.95), # PC prior on range for the GP on psi
              sigma_psi_prior=c(1, 0.01),   # PC prior on sd for the GP on psi
              tau_pc_prior=as.integer(1),
              range_tau_prior=c(0.5, 0.95),
              sigma_tau_prior=c(1, 0.01),
              phi_pc_prior=as.integer(1),
              range_phi_prior=c(0.5, 0.95),
              sigma_phi_prior=c(1, 0.01)
             )
parameters <- list(
  psi = rep(4.2, n_s),
  tau = rep(-2, n_s),
  phi = rep(0.2, n_s),
  beta_psi = 4.2, beta_tau = -3, beta_phi = 0.2,
  log_sigma_psi = -2, log_kappa_psi = -1,
  log_sigma_tau = -2, log_kappa_tau = -1,
  log_sigma_phi = -2, log_kappa_phi = -1)
random <- c("psi","tau","phi")
map <- NULL

adfun <- TMB::MakeADFun(data = data,
                        parameters = parameters,
                        random = random,
                        map = map,
                        DLL = "SpatialGEV_TMBExports",
                        silent = TRUE)
fit <- nlminb(adfun$par, adfun$fn, adfun$gr)
report <- TMB::sdreport(adfun, getJointPrecision=TRUE)

############ Posterior sampling  ################
n_draw <- 1000
hyperparam_labels <- c("beta_psi","beta_tau","beta_phi","log_sigma_psi",
                       "log_kappa_psi", "log_sigma_tau","log_kappa_tau",
                       "log_sigma_phi","log_kappa_phi")
lap_multilink_res <- get_posterior_from_tmb(report, meshidxloc, n_draw,
                                            "psi", "tau", "phi",
                                            hyperparam_labels=hyperparam_labels)
psi_est <- lap_multilink_res$u1 # extract posterior mean
tau_est <- lap_multilink_res$u2
phi_est <- lap_multilink_res$u3
z_est <- lap_multilink_res$z_s

lap_total_time <- difftime(Sys.time(), lap_start_t) # End timer
cat("Time taken (in sec) by the Laplace-MQ method is", "\n",
    lap_total_time, "\n")
```

```
## Time taken (in sec) by the Laplace-MQ method is
##  23.07274
```

The mean absolute errors from the Laplace method estimation are calculated.

```r
mae_psi_lap <- mean(abs(psi-psi_est))
mae_tau_lap <- mean(abs(tau-tau_est))
mae_phi_lap <- mean(abs(phi-phi_est))
mae_z_lap <- mean(abs(small_sim$z_true[-rm_loc_ind]-z_est))
```

5.2 Max-and-Smooth using J22 code

We now use the J22 implementation to fit the simulation data. This implementation is provided in the Supplementary Material of Jóhannesson et al. (2022) in the form of two R scripts:

1. `fitModelNoTrend.R`: the main script for model fitting and MCMC sampling.
2. `helperFun.R`: provides various helper functions used in the main script.

In order to use the J22 implementation here, we must make a few minimal adjustments:

i. We modified the inputs to `INLA::inla.mesh.2d()` so that the spatial mesh used in Max-and-Smooth is the exactly the one used in our paper.

ii. We removed all covariates used the original J22 code since our simulation studies do not include any covariates.

```r
set.seed(123)
# Prepare the data frame in a format used by Max-and-Smooth code
n_loc_rm <- length(data_na_rm)
data_max <- data.frame(Station=rep(1:length(data_na_rm),
                                   times=sapply(data_na_rm, length)),
                       data=unlist(data_na_rm))
stations <- unique(data_max$Station)
locs_ms <- small_sim$locs[-rm_loc_ind,]
colnames(locs_ms) <- c("long","lat")
locs_ms$Station <- 1:n_loc_rm
n_st <- n_loc_rm

# Source the J22 R scripts.
#
# **Note:** if INLA fails with notes about `inla.core.safe`,
# it is most likely due to an incompatible version of the
# **Matrix** package.
source("j22_Rscripts/helperFun.R")
source("j22_Rscripts/fitModelNoTrend.R")
```

```r
cat("Time taken by the original Max-Smooth code is", "\n",
    ms_total_time, "minutes \n")
```

```
## Time taken by the original Max-Smooth code is
##  7.064673 minutes
```

```
# fitModelNoTrend.R provides the random effect samples stored in
# `psi_sam`, `tau_sam` and `phi_sam`. Transform these parameters to get posterior
# samples of the 10% return level z_10.
z_sam <- sapply(1:N_x_loops,
     function(i){
       psi <- psi_sam[,i]
       tau <- tau_sam[,i]
       phi <- phi_sam[,i]
       mapply(qgev, p=0.1, loc=exp(psi), scale=exp(psi+tau),
                shape=phi2s(phi), lower.tail=FALSE)
     })

# Posterior mean of the parameters of interest
psi_ms_est <- rev(apply(psi_sam, 1, mean))
tau_ms_est <- rev(apply(tau_sam, 1, mean))
phi_ms_est <- rev(apply(phi_sam, 1, mean))
z_ms_est <- rev(apply(z_sam, 1, mean))
```

Table 3 summarizes the mean absolute errors (MAEs) for the J22 implementation along with those of Laplace-MQ.

```
mae_psi_ms <- mean(abs(psi-psi_ms_est))
mae_tau_ms <- mean(abs(tau-tau_ms_est))
mae_phi_ms <- mean(abs(phi-phi_ms_est))
mae_z_ms <- mean(abs(small_sim$z_true[-rm_loc_ind]-z_ms_est))
mae_j22 <- data.frame(c(mae_psi_ms, mae_psi_lap), c(mae_tau_ms, mae_tau_lap),
                      c(mae_phi_ms, mae_phi_lap), c(mae_z_ms, mae_z_lap),
                      row.names = c("Max-and-Smooth J22", "Laplace-MQ"))
colnames(mae_j22) <- c("MAE($\\psi$)", "MAE($\\tau$)", "MAE($\\phi$)", "MAE($z_{10}$)")
make_table(mae_j22,
           caption = "Summary of mean absolute errors.")
```

Table 3: Summary of mean absolute errors.

	MAE(ψ)	MAE(τ)	MAE(ϕ)	MAE(z_{10})
Max-and-Smooth J22	0.015	0.107	0.130	6.392
Laplace-MQ	0.006	0.048	0.035	2.327

Figure 17 displays the true versus estimated posterior means of the parameters of interest using both methods. The performance of J22 on this simulated data is much worse than using our original parametrization in Section 2. Specifically, the transformed shape parameter ϕ is severely underestimated by J22, which translates to a much higher MAE in Table 3 on the return level z_{10} than we found under the original parametrization in Table 1. The Laplace-MQ approximation also has trouble estimating ϕ but to a lesser extent. This does not seem to have much of an impact on the MAE for z_{10}, which is about the same in Table 3 as it is in Table 1. A potential reason for the J22 parametrization to produce worse results in our simulation study is that the it has multiple locations with values of s_o close to the J22 boundary value of 0.5, which both the J22 method and the Laplace-MQ method under this parametrization tend to underestimate.

```
mytheme <- create_ggplot_theme()
true_z10_lab <- expression(paste("True ", z[10], "(x)"))
est_z10_lab <- expression(paste("Estimated ", z[10], "(x)"))
```

```
psi_range <- range(c(psi, psi_ms_est, psi_est))
tau_range <- range(c(tau, tau_ms_est, tau_est))
phi_range <- range(c(phi, phi_ms_est, phi_est))
z_range <- range(c(small_sim$z_true[-rm_loc_ind], z_ms_est, z_est))

psi_plot_lap <- true_vs_est_scatterplot(psi, psi_est, psi_range, "Laplace",
                                        expression("True"~psi),
                                        expression("Estimated"~psi),
                                        mytheme)
tau_plot_lap <- true_vs_est_scatterplot(tau, tau_est, tau_range, "Laplace",
                                        expression("True"~tau),
                                        expression("Estimated"~tau),
                                        mytheme)
phi_plot_lap <- true_vs_est_scatterplot(phi, phi_est, phi_range, "Laplace",
                                        expression("True"~phi),
                                        expression("Estimated"~phi),
                                        mytheme)
z_plot_lap <- true_vs_est_scatterplot(small_sim$z_true[-rm_loc_ind],
                                      z_est, z_range,
                                      "Laplace", true_z10_lab,
                                      est_z10_lab,
                                      mytheme)
psi_plot_ms <- true_vs_est_scatterplot(psi, psi_ms_est, psi_range,
                                       "Max-and-Smooth",
                                       expression("True"~psi),
                                       expression("Estimated"~psi),
                                       mytheme)
tau_plot_ms <- true_vs_est_scatterplot(tau, tau_ms_est, tau_range,
                                       "Max-and-Smooth",
                                       expression("True"~tau),
                                       expression("Estimated"~tau),
                                       mytheme)
phi_plot_ms <- true_vs_est_scatterplot(phi, phi_ms_est, phi_range,
                                       "Max-and-Smooth",
                                       expression("True"~phi),
                                       expression("Estimated"~phi),
                                       mytheme)
z_plot_ms <- true_vs_est_scatterplot(small_sim$z_true[-rm_loc_ind], z_ms_est,
                                     z_range,
                                     "Max-and-Smooth", true_z10_lab,
                                     est_z10_lab,
                                     mytheme)
ggarrange(psi_plot_ms, tau_plot_ms, phi_plot_ms, z_plot_ms,
          psi_plot_lap, tau_plot_lap, phi_plot_lap, z_plot_lap,
          nrow=2, ncol=4)
```

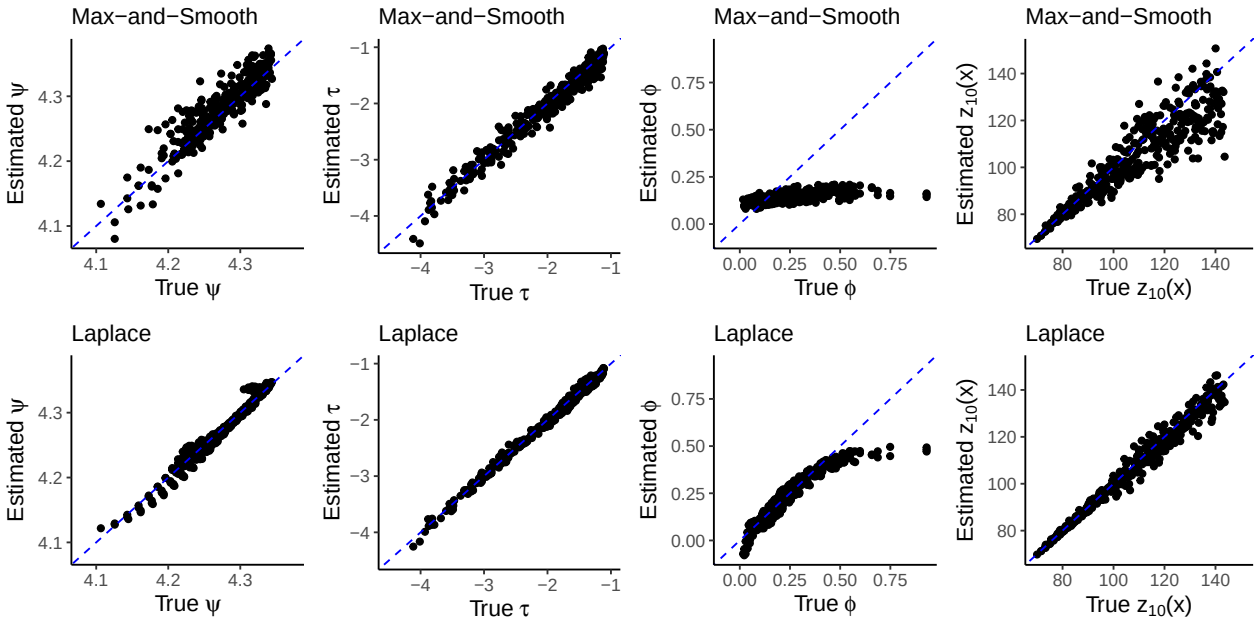

Figure 17: True vs posterior mean of random effects and return levels for Laplace-MQ and J22 Max-and-Smooth estimators.

6 References

Jóhannesson, Árni V., Stefan Siegert, Raphael Huser, Haakon Bakka, and Birgir Hrafnkelsson. 2022. "Approximate Bayesian Inference for Analysis of Spatiotemporal Flood Frequency Data." *The Annals of Applied Statistics* 16 (2): 905–35.

Kristensen, K., A. Nielsen, C. W. Berg, H. Skaug, and B. M Bell. 2016. "TMB: Automatic Differentiation and Laplace Approximation." *Journal of Statistical Software* 70 (5): 1–21.

Lindgren, Finn, and Håvard Rue. 2015. "Bayesian Spatial Modelling with R-INLA." *Journal of Statistical Software* 63 (19): 1–25. http://www.jstatsoft.org/v63/i19/.

Monnahan, Cole, and Kasper Kristensen. 2018. "No-U-turn Sampling for Fast Bayesian Inference in ADMB and TMB: Introducing the adnuts and tmbstan R Packages." *PloS One* 13 (5). https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0197954.

Stan Development Team. 2020. "RStan: The R Interface to Stan." Version 2.21.2. http://mc-stan.org/.

Stephenson, A. G. 2002. "Evd: Extreme Value Distributions." *R News* 2 (2).

A C++ model template file for GEV-GP with SPDE kernel

```
#ifndef model_abs_spde_hpp
#define model_abs_spde_hpp

#include "SpatialGEV/utils.hpp"

#undef TMB_OBJECTIVE_PTR
#define TMB_OBJECTIVE_PTR obj

/// TMB specification of GEV-GP models with a chosen covariance kernel.
///
/// The model is defined as follows:
```

```
///
/// y ~ GEV(a, b, s),
/// a ~ GP(log_sigma_a, log_kappa_a)
/// log_b ~ GP(log_sigma_b, log_kappa_b)
/// s ~ GP(log_sigma_s, log_kappa_s)
/// where the GP is parameterized using the spde covariance kernel.
///
/// --------- Data provided from R ---------------
/// @param[in] y Response vector of length `n_obs`.  Assumed to be > 0.
/// @param[in] loc_ind Location vector of length `n_obs` of integers
/// `0 <= i_loc < n_loc` indicating to which locations each element of `y` is
/// associated.
/// @param[in] reparam_s Integer indicating the type of shape parameter. 0:
/// `s = 0`, i.e., use Gumbel instead of GEV distribution.  1: `s > 0`, in which
/// case we operate on `log(s)`.  2: `s < 0`, in which case we operate on
/// `log(-s)`.  3: unconstrained.
/// @param[in] beta_prior Integer specifying the type of prior on the design
/// matrix coefficients. 1 is weakly informative normal prior and any other
/// numbers means Lebesgue prior `pi(beta) \propto 1`.
/// @param[in] return_periods Vector of return periods to ADREPORT. If the first
/// element of this vector is 0, then no return level calculations are performed
/// .
/// @param[in] spde Object of type `spde_t` as constructed in R by a call to
/// [INLA::inla.spde2.matern()] consisting of `n_loc` mesh locations.
/// @param[in] design_mat_a Design matrix of size
/// `n_loc x n_covariate` for parameter a.
/// @param[in] beta_a_prior Vector of length 2 containing the mean
/// and sd of the normal prior on `beta_a`.
/// @param[in] design_mat_b Design matrix of size
/// `n_loc x n_covariate` for parameter log_b.
/// @param[in] beta_b_prior Vector of length 2 containing the mean
/// and sd of the normal prior on `beta_b`.
/// @param[in] design_mat_s Design matrix of size
/// `n_loc x n_covariate` for parameter s.
/// @param[in] beta_s_prior Vector of length 2 containing the mean
/// and sd of the normal prior on `beta_s`.
/// @param[in] nu Presepecified smoothness parameter for the Matérn covariance
/// kernel applicable to all random effects.
/// @param[in] a_pc_prior Integer specifying the type of prior to
/// use on the Matérn GP on a. 1 for using PC prior on
/// a, 0 for using Lebesgue prior.
/// @param[in] range_a_prior PC prior on the range parameter for
/// the Matérn GP on
/// a. Vector of length 2 `(rho_0, p_rho)` s.t.
/// `Pr(rho < rho_0) = p_rho`.
/// @param[in] sigma_a_prior PC prior on the variance parameter for
/// the Matérn GP on
/// a. Vector of length 2 `(sig_0, p_sig)` s.t.
/// `Pr(sig > sig_0) = p_sig`.
/// @param[in] b_pc_prior Integer specifying the type of prior to
/// use on the Matérn GP on log_b. 1 for using PC prior on
/// log_b, 0 for using Lebesgue prior.
/// @param[in] range_b_prior PC prior on the range parameter for
```

```
/// the Matérn GP on
/// log_b. Vector of length 2 `(rho_0, p_rho)` s.t.
/// `Pr(rho < rho_0) = p_rho`.
/// @param[in] sigma_b_prior PC prior on the variance parameter for
/// the Matérn GP on
/// log_b. Vector of length 2 `(sig_0, p_sig)` s.t.
/// `Pr(sig > sig_0) = p_sig`.
/// @param[in] s_pc_prior Integer specifying the type of prior to
/// use on the Matérn GP on s. 1 for using PC prior on
/// s, 0 for using Lebesgue prior.
/// @param[in] range_s_prior PC prior on the range parameter for
/// the Matérn GP on
/// s. Vector of length 2 `(rho_0, p_rho)` s.t.
/// `Pr(rho < rho_0) = p_rho`.
/// @param[in] sigma_s_prior PC prior on the variance parameter for
/// the Matérn GP on
/// s. Vector of length 2 `(sig_0, p_sig)` s.t.
/// `Pr(sig > sig_0) = p_sig`.
///
/// --------- Parameters to estimate ------------
/// @param[in] a GEV location parameter.
/// Vector of length `n_loc`.
/// @param[in] log_b GEV scale parameter on the log scale.
/// Vector of length `n_loc`.
/// @param[in] s GEV shape parameter on the scale specified by `reparam_s`.
/// Vector of length `n_loc`.
/// @param[in] beta_a GP mean covariate coefficient vector of
/// length `n_covariate` for a.
/// @param[in] log_sigma_a GP covariance kernel variance
/// hyperparameter for a.
/// @param[in] log_kappa_a GP covariance kernel range
/// hyperparameter for a.
/// @param[in] beta_b GP mean covariate coefficient vector of
/// length `n_covariate` for log_b.
/// @param[in] log_sigma_b GP covariance kernel variance
/// hyperparameter for log_b.
/// @param[in] log_kappa_b GP covariance kernel range
/// hyperparameter for log_b.
/// @param[in] beta_s GP mean covariate coefficient vector of
/// length `n_covariate` for s.
/// @param[in] log_sigma_s GP covariance kernel variance
/// hyperparameter for s.
/// @param[in] log_kappa_s GP covariance kernel range
/// hyperparameter for s.
template<class Type>
Type model_abs_spde(objective_function<Type>* obj){
  using namespace density;
  using namespace R_inla;
  using namespace Eigen;
  using namespace SpatialGEV;

  // ------ Data inputs ------------
  DATA_VECTOR(y);
```

```
DATA_IVECTOR(loc_ind);
DATA_INTEGER(reparam_s);
DATA_INTEGER(beta_prior);
DATA_VECTOR(return_periods);
int has_returns = return_periods(0) > Type(0.0);
DATA_STRUCT(spde, spde_t);
int n_loc = spde.M0.rows(); // number of spatial locations
DATA_SCALAR(nu);

// Inputs for a
DATA_MATRIX(design_mat_a);
DATA_VECTOR(beta_a_prior);
DATA_INTEGER(a_pc_prior);
DATA_VECTOR(range_a_prior);
DATA_VECTOR(sigma_a_prior);
// Inputs for log_b
DATA_MATRIX(design_mat_b);
DATA_VECTOR(beta_b_prior);
DATA_INTEGER(b_pc_prior);
DATA_VECTOR(range_b_prior);
DATA_VECTOR(sigma_b_prior);
// Inputs for s
DATA_MATRIX(design_mat_s);
DATA_VECTOR(beta_s_prior);
DATA_INTEGER(s_pc_prior);
DATA_VECTOR(range_s_prior);
DATA_VECTOR(sigma_s_prior);

// ------------ Parameters ----------------------

PARAMETER_VECTOR(a);
PARAMETER_VECTOR(log_b);
PARAMETER_VECTOR(s);

PARAMETER_VECTOR(beta_a);
PARAMETER_VECTOR(beta_b);
PARAMETER_VECTOR(beta_s);
PARAMETER(log_sigma_a);
PARAMETER(log_kappa_a);
PARAMETER(log_sigma_b);
PARAMETER(log_kappa_b);
PARAMETER(log_sigma_s);
PARAMETER(log_kappa_s);

// Initialize the negative log likelihood
Type nll = Type(0.0);

// ---------- Likelihood contribution from a ------------------
// GP latent layer
vector<Type> mu_a = a -
  design_mat_a * beta_a;
nll += nlpdf_gp_spde<Type>(mu_a, spde,
                 exp(log_sigma_a),
```

```
                exp(log_kappa_a),
                                nu);
// Priors
nll += nlpdf_beta_prior<Type>(beta_a, beta_prior,
    beta_a_prior(0), beta_a_prior(1));
nll += nlpdf_matern_hyperpar_prior<Type>(log_kappa_a,
                    log_sigma_a,
                    a_pc_prior,
                                        nu, range_a_prior,
                    sigma_a_prior);
// ---------- Likelihood contribution from log_b ------------------
// GP latent layer
vector<Type> mu_b = log_b -
  design_mat_b * beta_b;
nll += nlpdf_gp_spde<Type>(mu_b, spde,
                exp(log_sigma_b),
                exp(log_kappa_b),
                                nu);
// Priors
nll += nlpdf_beta_prior<Type>(beta_b, beta_prior,
    beta_b_prior(0), beta_b_prior(1));
nll += nlpdf_matern_hyperpar_prior<Type>(log_kappa_b,
                    log_sigma_b,
                    b_pc_prior,
                                        nu, range_b_prior,
                    sigma_b_prior);
// ---------- Likelihood contribution from s ------------------
// GP latent layer
vector<Type> mu_s = s -
  design_mat_s * beta_s;
nll += nlpdf_gp_spde<Type>(mu_s, spde,
                exp(log_sigma_s),
                exp(log_kappa_s),
                                nu);
// Priors
nll += nlpdf_beta_prior<Type>(beta_s, beta_prior,
    beta_s_prior(0), beta_s_prior(1));
nll += nlpdf_matern_hyperpar_prior<Type>(log_kappa_s,
                    log_sigma_s,
                    s_pc_prior,
                                        nu, range_s_prior,
                    sigma_s_prior);

// ------------- Data layer -----------------
for(int i=0;i<y.size();i++) {
  nll -= gev_reparam_lpdf<Type>(y(i), a(loc_ind(i)), log_b(loc_ind(i)),
  s(loc_ind(i)), reparam_s);
}

// ------------- Output return levels ----------------------
if(has_returns) {
  matrix<Type> return_levels(return_periods.size(), n_loc);
  for(int i=0; i<n_loc; i++) {
```

```
      gev_reparam_quantile<Type>(return_levels.col(i), return_periods,
                                 a(i), log_b(i), s(i), reparam_s);
    }
    ADREPORT(return_levels);
  }

  return nll;
}
#undef TMB_OBJECTIVE_PTR
#define TMB_OBJECTIVE_PTR this

#endif
```