BSTFA Package

Abstract

An overview of the functions for fitting and understanding Bayesian spatio-temporal factor analysis (BSTFA) models for spatio-temporal data. To improve computational efficiency, the package includes a function for a BSTFA model that uses dimension reduction via basis functions. A more computationally intensive model using Gaussian processes is also available. Users can generate posterior interpolations, visualize spatio-temporal trends, and explore latent factor behavior through functions for mapping, time series plotting, and interpolation. The package is designed to support both exploratory data analysis and formal inference in applications with spatial and temporal dependence. Functionality is demonstrated using a historical temperature dataset collected from weather stations across Utah, USA.

Contents

1 Introduction

1.1 Intended Audience

This document is intended to help even novice Bayesian statistics students implement a fully Bayesian spatio-temporal analysis. The functions within the package are designed with this audience in mind. This document is meant to guide any potential user of this package through the basic implementation of the model fitting and inferential processes; in essence, it is an instruction manual. The bulk of this document contains examples of our functions applied on an observed data set.

The outline is as follows. First, we introduce the motivation behind the BSTFA models and why simplifying the model for fast computation is important. Section 2 outlines the available functions and procedures within the `BSTFA` package along with demonstrations on a real data set. Some basic theory and methodology are contained in Section 3, and Section 4 details specific features of the package. The appendix contains a few helpful additional notes on computation and references.

1.2 Motivation

Consider the motivating data set for the `BSTFA` package: a collection of temperature measurements across the state of Utah. The data were collected from May 1912 through January 2015 from 146 weather stations across the state. These measurements are 30-day averages of daily minimum observed temperatures in degrees Celcius, with each location's measurements zero-centered. This environmental process exhibits some of the challenges common in environmental modeling; that is, the data exhibit spatial and temporal dependence and not all contributing agents are known or easy to include in a modeling scheme.

Take, for example, the observations for three weather stations: Moab, Canyonlands National Park, and Logan. The Moab and Canyonlands stations are near one another (within 50 miles) while the Logan station is far away (300 miles). Figure 1 shows these same temperature series zoomed in on the years 1999 through 2001. The difference between low temperatures in winter 2000 and winter 2001 is slight in Moab and the Canyonlands, but in Logan, the low temperature is much lower in winter 2001 than it was in winter 2000. This anecdote illustrates that locations near each other in space exhibit similar environmental behavior. Spatio-temporal factor analysis accounts for such spatio-temporal dependencies and can provide numerical and visual summaries of that dependence.

Estimation of a fully-parameterized Bayesian spatio-temporal factor analysis model is computationally burdensome. The `BSTFA` package accounts for this by using dimension reduction via basis functions, allowing for faster computation. The remainder of this vignette describes the `BSTFA` package and its use of basis functions, as well as all implemented methods for plotting, interpolating, and inference.

2 What is Implemented?

The `BSTFA` package contains implementation of two versions of a spatio-temporal factor analysis model along with functions for interpolation, plotting and visualizing posterior surfaces. The model-fitting functions are defined in Section 2.1, while the methodology associated with these models is described more fully in Section 3. The `BSTFA` package's interpolation methods are discussed in Section 2.2, functions for plotting/visualization are described in Section 2.3, and notes about computational speed are outlined in Section 2.4.

While each of these sections describes some arguments to the functions, the best way to understand all available function arguments is to look at the R help documentation.

2.1 Model-Fitting Functions

The `BSTFA` package contains two model fitting functions: `BSTFA`, the smoother and computationally-efficient spatio-temporal factor analysis model using basis functions for the factor analysis component; and `BSTFAfull`, the fine-grain but computationally-slower spatio-temporal factor analysis model using Gaussian processes for the factor analysis. Both functions return a *list* object containing all information required to summarize

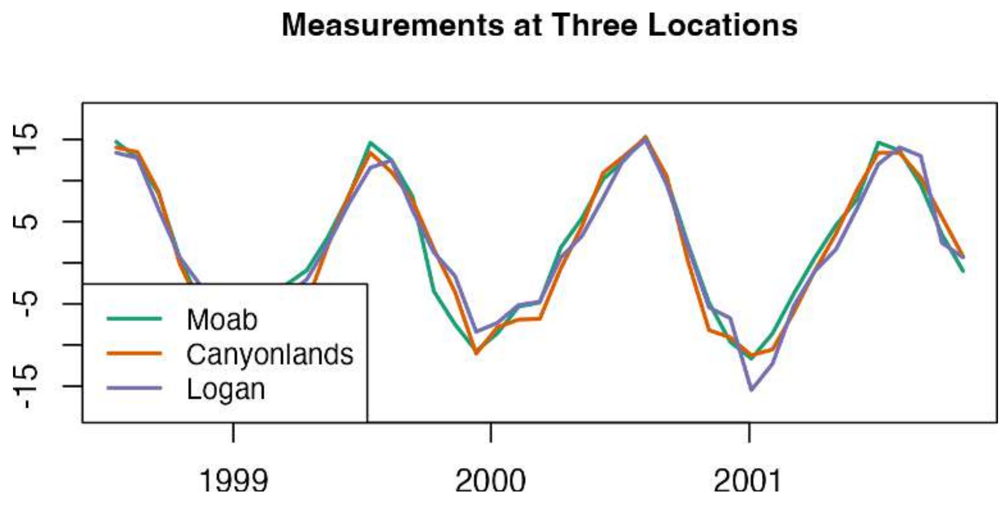

Figure 1: Mean-centered 30-day average daily minimum temperatures for three Utah weather stations (Moab, green; Canyonlands, orangee; Logan, purple) from January 1999 through December 2001.

Table 1: Required arguments for the `BSTFA` and `BSTFAfull` functions.

Argument	Description
`ymat`	A matrix of response values. Each row should represent a point in time; each column should represent a specific location. Missing values should be recorded as NA.
`dates`	A vector of dates of length `nrow(ymat)`. The model functions will transform this vector into 'doy' (day of year) using `lubridate::yday()`. Thus, this vector must either be a lubridate or string object with year-month-day format.
`coords`	A matrix of coordinate values with number of rows equal to `ncol(ymat)` and 2 columns; if using longitude/latitude, longitude should be the first column.

posterior inference, including all posterior draws from each parameter, matrices containing the basis functions, and information about computation time. Each function has only three required arguments, summarized in Table 1. Other arguments relating to model fitting and prior parameter values will be discussed more fully in Section 3.

The `BSTFA` and `BSTFAfull` functions are demonstrated below. Additional Markov-chain Monte Carlo (MCMC) arguments such as `iters`, `thin`, and `burn` have default values, but they can be specified as in any Bayesian model to control the number of posterior draws. Although the `BSTFA` function is much faster than the fully-parameterized `BSTFAfull` function, it is still an MCMC with many parameters and will need many draws to fully represent the posterior distributions. Except for the model comparison figures (where plots were created outside the vignette), for the sake of keeping the vignette's compile time short, we use a data set in the package called `out.sm` which is an object fit using the code below.

```
#Code used to create the "out.sm" data object in the BSTFA package.
#Note: Not run within this vignette.

#Load the full temperature data set
data(UtahDataList)
attach(UtahDataList)
```

```r
#Load the data and select a subset
dates.ind <- 1151:1251
locs.use <- c(3, 8, 11, 16, 17,
                        20, 23, 29, 30, 46,
                        47, 49, 60, 62, 66, 73,
                        75, 76, 77, 78, 85, 89, 94,
                        96, 98, 100, 109, 112,
                        115, 121, 124, 128, 133, 144)
temps.sm <- TemperatureVals[dates.ind, locs.use]
coords.sm <- Coords[locs.use,]
dates.sm <- Dates[dates.ind]
locsm.names <- Locations[locs.use]

#Fit the model
set.seed(466)
out.sm <- BSTFA(ymat=temps.sm,
                    dates=dates.sm,
                    coords=coords.sm,
                    iters=5000,
                    burn=1000,
                    thin=40,
                    factors.fixed=c(14, 22, 15, 20),
                    n.temp.bases=45,
                  save.missing=FALSE)
```

The `verbose` argument controls whether or not the function prints status updates during sampling. Although the default value for this is `TRUE`, for the sake of this vignette, `verbose` will always be set to `FALSE`.

The `utahDataList` list object in the `BSTFA` package contains the observed temperature data (`TemperatureVals`), the corresponding dates (`Dates`), the coordinates (`Coords`), and the weather station names (`Locations`).

```r
out = BSTFA(ymat=utahDataList$TemperatureVals,
                     dates=utahDataList$Dates,
                     coords=utahDataList$Coords,
                     verbose=FALSE)
```

```r
full.out = BSTFAfull(ymat=utahDataList$TemperatureVals,
                         dates=utahDataList$Dates,
                         coords=utahDataList$Coords,
                         verbose=FALSE)
```

```r
#Load small pre-fit model output
data(out.sm)
attach(out.sm)
```

2.1.1 Understanding Output

The `BSTFA` and `BSTFAfull` functions return a list object. It should be noted that a user can use all functions within the `BSTFA` package without understanding or using the specific output. We provide these additional details for more in-depth analyses and summaries. Most objects contained in the list are self-explanatory. A few, however, warrant further explanation.

- Each object that is a parameter (i.e., `beta`) is an MCMC object from the `coda` package (Plummer et al., 2006) with number of rows equal to the number of MCMC draws.
- `time.data` is a matrix with number of rows equal to `iters` and columns indicating the computation

time (in seconds) for the given parameter on that given iteration. This is the object used to create the `compute_summary` mentioned in Section 2.4.

- `y.missing` is a matrix with number of rows equal to the number of missing data points in `ymat` and number of columns equal to the number of MCMC draws. These are posterior draws from relevant distributions of the missing values of `ymat`. If `save.missing=FALSE` in the function, this will be `NULL`.
- `model.matrices` stores all basis function matrices and other useful matrices for calculating Y (more details given in Section 3).
  - `newS` is equivalent to $\mathbf{B}_\beta \equiv \mathbf{B}_{\xi_j} \forall j$ and has dimension $n \times b_\beta$. Note that $b_\beta = b_\xi$, and this value is `n.spatial.bases` in the model functions.
  - `linear.Tsub` is $t - \bar{t}$, a $T \times 1$ vector of values $t - \bar{t} \; \forall \; t$.
  - `seasonal.bs.basis` is the $t \times u$ matrix of cubic circular b-spline bases where each row represents $\mathbf{U}(t^*)$ for a given time point t^*.
  - `confoundingPmat.prime` is an orthogonal projection matrix $P^\perp$ used to prevent confounding between the linear and seasonal components and the factor analysis component. For more information about this component, see Berrett et al. (2020).
  - `QT` is the $T \times R_t$ matrix of Fourier basis functions for the temporally-dependent factors.
  - `QS` is the $n \times R_s$ matrix of basis functions used for the spatially-dependent loadings. If `spatial.style == load.style` and `n.spatial.bases == n.load.bases`, this will be equivalent to `newS`.
- **Note** : Care must be taken in understanding the ordering of `F.tilde` and `Lambda.tilde` (discussed in greater detail in Section 3.1.3). Each draw of `F.tilde` (meaning, each column) has dimension $TL \times 1$. This means the first T values correspond to factor one, the next T values correspond to factor two, and so on. However, each draw of `Lambda.tilde` has dimension $Ln \times 1$. This means the first L values correspond to location one, the next L values correspond to location two, and so on. When converting a draw of `F.tilde` into a matrix, setting `byrow=FALSE` provides the appropriate $T \times L$ matrix, while when converting a draw of `Lambda.tilde` into a matrix, setting `byrow=TRUE` provides the appropriate $n \times L$ matrix.

2.2 Interpolation

The function `predictBSTFA` takes as its first argument the output from the `BSTFA` or `BSTFAfull` functions. Within the function, posterior samples from relevant parameters are used to interpolate either spatio-temporal processes, $Y(\mathbf{s}, t)$, at observed location $\mathbf{s}$ and time t, or for, $Y(\mathbf{s}^*, t)$, at *unobserved* location $\mathbf{s}^*$ and time t. The `location` argument takes either a location number (corresponding to the appropriate column of your `ymat` argument) for estimation at an observed location, or a matrix of coordinate values if interpolating to a new location.

If the argument `type` is set to `"all"`, the function will return draws of $Y(\mathbf{s}, t) \; \forall \; t$ for each saved draw of the parameters. `type` can also be set to `"mean"`, `"median"`, `"ub"` (upper bound), or `"lb"` (lower bound). The option `pred.int` controls whether the calculated uncertainty is a prediction interval (`pred.int == TRUE`) or a credible interval (`pred.int == FALSE`; default value).

The code below provides posterior predictive draws of temperatures at an observed location, Loa, Utah. Since `type == "all"`, the function will return a $T \times d$ matrix of posterior draws of $Y(\mathbf{s}, t) \; \forall \; t$ with d being the number of saved MCMC draws. Each column represents one draw of the $T \times 1$ vector, $\mathbf{Y}(\mathbf{s})$.

```
loc = 17 # Loa, Utah in our small data set
preds = predictBSTFA(out.sm,
                     location = loc,
                     type='all',
                     pred.int=TRUE,
                     ci.level=c(0.025,0.975))
```

The code below sets `type == "mean"` and returns a $T \times 1$ vector containing the posterior *mean* of $Y(\mathbf{s}, t) \; \forall \; t$.

```r
loc = 17 # Loa, Utah in our small data set
preds = predictBSTFA(out.sm,
                     location = loc,
                     type='mean')
```

The code below provides posterior predictive draws for temperatures at an *unobserved* location by setting the `location` argument to a matrix of coordinate values. In this case, we consider a city without a monitor, Torrey, Utah (near Loa), with longitude 111.41° W and latitude 38.29° N.

```r
loc = matrix(c(-111.41, 38.29), nrow=1, ncol=2) # Torrey, Utah
preds_new = predictBSTFA(out.sm,
                         location = loc,
                         type='all',
                         pred.int=TRUE,
                         ci.level=c(0.025,0.975))
```

The `predictBSTFA` function is also called within the `plot_location` function, which takes similar arguments as `predictBSTFA` with the addition of a few extra plotting arguments. `xrange` defines the time points $\{t : t \in \mathcal{T}\}$ at which the posterior estimates are plotted, with default value `NULL` indicating to plot on all of $\mathcal{T}$. `truth` (default value `FALSE`) will plot the observed data along with the estimates if `location` comes from the data set. `uncertainty` (default value `TRUE`) indicates whether to include a posterior predictive interval (`pred.int==TRUE`) or a credible interval (`pred.int==FALSE`).

Below is an example of code used to plot the posterior mean temperature values (black line) at an observed location, Loa, Utah, with 95% posterior credible bounds (gray bands), and observed measurements (gray circles). Figure 2 provides the plot. Notice that the posterior mean (black line) closely follows the pattern of the observed data, but, as expected due to basis smoothing, is more smooth than the observed data. Increasing the number of bases will decrease the smoothness (but increase computation time).

```r
loc = 17 # Loa, Utah in our small data set
plot_location(out.sm,
              location=loc,
              type='mean',
              uncertainty=TRUE,
              ci.level=c(0.025,0.975),
              truth=TRUE)
```

Below is an example of code used to plot the estimated posterior mean temperature values at an *unobserved* location, "Torrey, Utah," a location near Loa. Figure 3 provides the plot where again, the black line represents the posterior mean and the gray bands represent the 95% posterior predictive intervals.

```r
loc = matrix(c(-111.41, 38.29), nrow=1, ncol=2) # Torrey, Utah
plot_location(out.sm,
              location=loc,
              type='mean',
              uncertainty=TRUE,
              ci.level=c(0.025,0.975),
              truth=FALSE)
```

2.3 Visualization

The `BSTFA` package contains multiple functions for plotting and visualizing the model output. Of course, all of these can be implemented on your own (the output from the `BSTFA` or `BSTFAfull` functions contain all posterior draws and basis function matrices), but these functions exist for quick plotting and visualization of posterior distributions. Some functions use base R for plotting while others use the `ggplot2` package (Wickham, 2016). Table 2 below displays a table with each plotting function and a basic description.

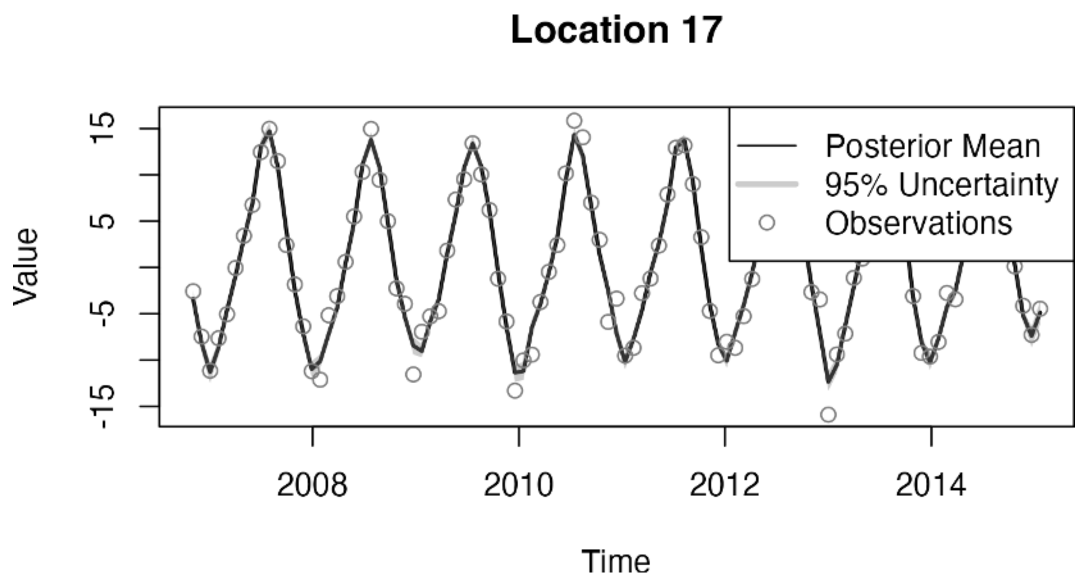

Figure 2: Posterior mean temperature values (black line) at an in-sample location, Loa, Utah, with 95% posterior predictive bounds (gray bands), and observed temperature measurements (gray circles).

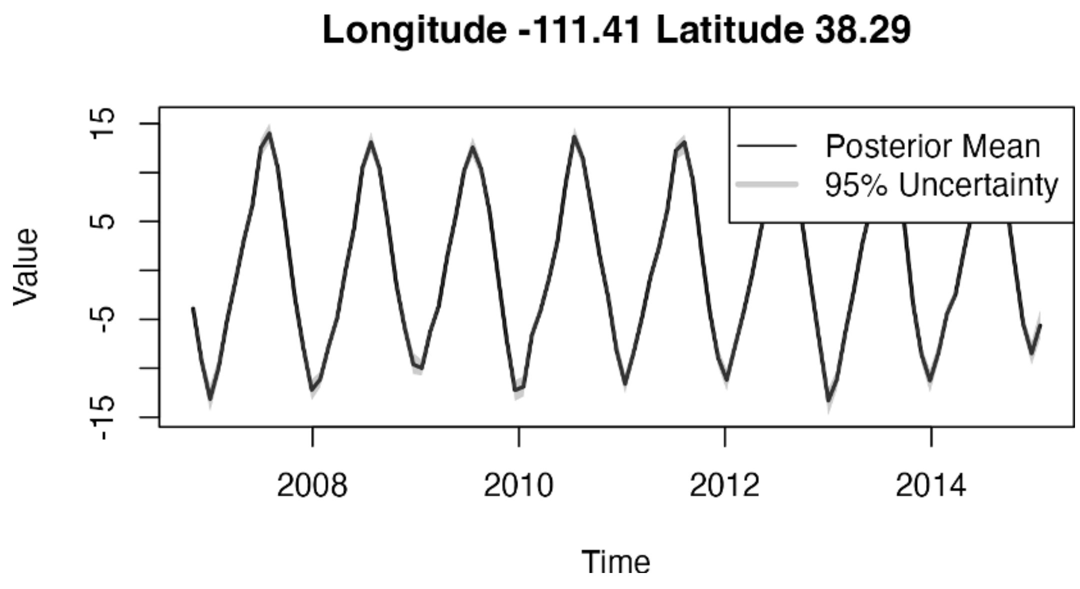

Figure 3: Posterior mean temperature values (black line) at an out-of-sample location, Torrey, Utah, with 95% posterior predictive bounds (gray bands).

Table 2: Plotting/visualization functions in the `BSTFA` package.

Function	Description
`plot_location`	Plot estimated response variable at a specific location (either observed or unobserved) for a specified time range. Credible or prediction interval bands for a given probability (default is 95%) can be included. Uses base R for plotting.
`plot_annual`	Plot the estimated annual seasonal behavior at a specific location (either observed or unobserved). Credible interval bands for a given probability (default is 95%) can be included. Uses base R for plotting.
`plot_spatial_param`	Plot the estimated spatially-dependent linear slope or specific factor loading (the user specifies the parameter of interest) at all observed locations. Credible interval bounds for a given probability (default is 95%) can also be plotted. Uses `ggplot2` for plotting.
`map_spatial_param`	Plot the interpolated spatially-dependent linear slope or specific factor loading (the user specifies the parameter of interest) on a grid of unobserved locations. Credible interval bounds for a given probability (default is 95%) can also be plotted. Contains arguments to import and plot the grid on a map using functions from the `sf` package. Uses `ggplot2` for plotting.
`plot_factor`	Plot the estimated factors, either individually or all together. Credible interval bands for a given probability (default is 95%) can be included. Uses base R for plotting.

For instance, `plot_annual` can plot the estimated annual seasonal behavior at any (observed or unobserved) location. The code for doing this for an observed location, Loa, Utah, is provided below. Figure 4 provides the corresponding plot, where the black line is the posterior mean, the gray band is the 95% credible interval, and the gray dots are the observations plotted on their day of year. This shows the expected seasonal behavior – that the temperatures tend to increase in the spring/summer months and decrease in the fall/winter months.

```r
plot_annual(out.sm,
            location=17, # Loa, Utah in our small data set
            years='one')
```

The `plot_spatial_param` function can plot the posterior mean of the linear slope or factor loadings at all observed locations. The `type` argument (default is `"mean"`, with other options `"median"`, `"ub"`, or `"lb"`) indicates which summary to show. The `parameter` argument can be set either to `"slope"` or `"loading"`. If set to `"slope"`, the argument `yearscale` (default is `TRUE`) controls whether the slope estimates are "per year." If set to `"loading"`, the `loadings` argument indicates which loading to plot. First, we provide an example to plot the estimated linear change in temperature (increase or decrease) across time. Figure 5 provides the corresponding plot, where the color represents the long-term increasing (positive and red) or decreasing (negative and blue) behavior over the observed time period for the observed locations. Notice that most locations show an average increase in temperature of between 0 and approximately 0.3 degrees Celcius per year over the 2007 to 2015 time period.

```r
plot_spatial_param(out.sm,
          type='mean',
          parameter='slope',
          yearscale=TRUE)
```

Next, we provide example code for plotting a particular loading, in this case, the posterior mean loading for the first factor. Figure 6 provides the corresponding plot showing the posterior mean loading values that each location places on the first factor, with the color indicating the strength of the weight (darker colors indicate a stronger weight). The circled dot shows the location of the fixed factor; in this case, the first factor

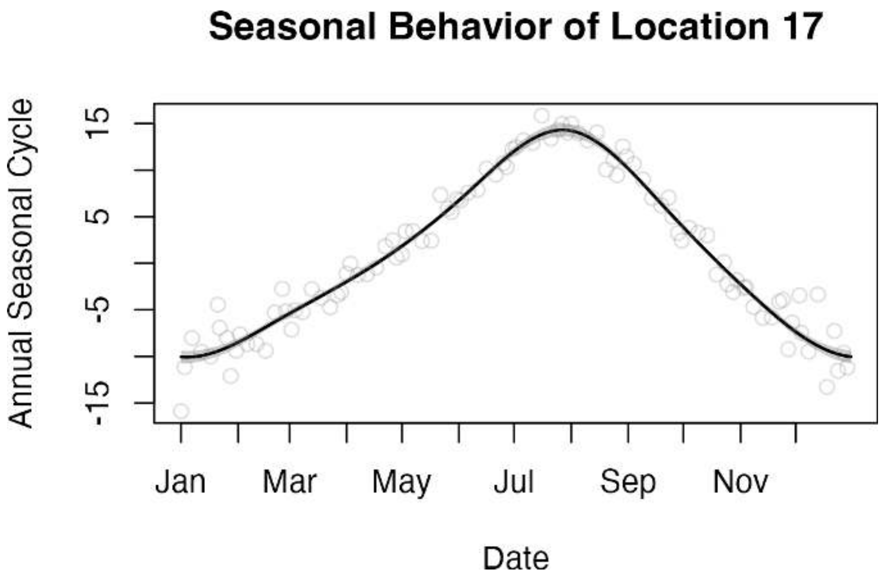

Figure 4: Posterior mean annual seasonal behavior (black line) with 95% credible interval (gray band), and observed data (open gray circles) plotted by day of year.

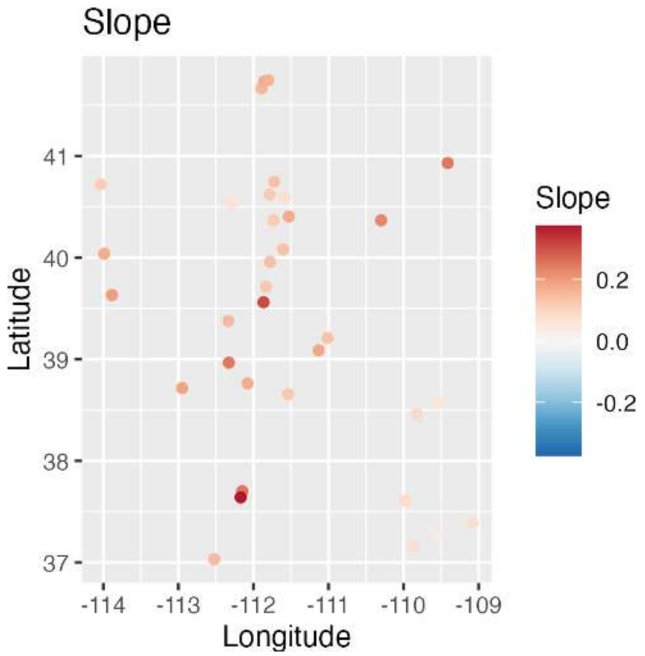

Figure 5: Posterior mean linear change in time (slope) of temperatures at observed locations.

is fixed at Ibapah, Utah. This means that the loading corresponding to factor 1 for that location was fixed at a value of 1 (and fixed to be 0 for factors 2, 3, and 4) and the loadings for other locations are estimated given that value. The model accounts for spatial dependence in the loadings so that locations near to Ibapah are modeled to have posterior mean loadings closer to one.

```r
plot_spatial_param(out.sm,
          type='mean',
          parameter='loading',
          loadings=1)
```

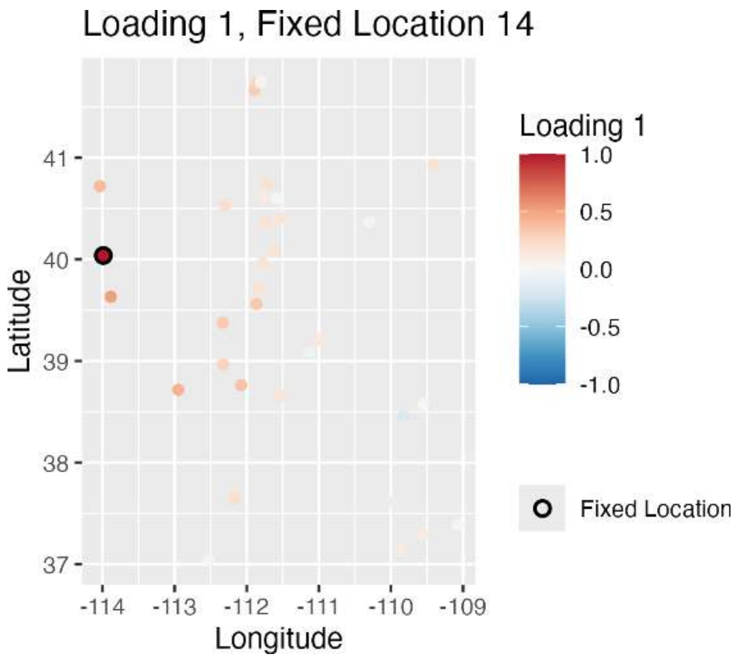

Figure 6: Posterior mean estimates of loadings for factor 1 at observed locations. The circled red dot shows the location of the fixed factor loading.

The `plot_factor` function plots the estimated temporally-dependent factors either together (setting `together=TRUE`) or separate (setting `together=FALSE` and `factor` to the factor number you want to plot). The example code below plots the first factor, corresponding to the loadings shown in the previous example. Figure 7 shows the posterior mean (black line) and 95% credible intervals (gray band) of the first factor across the 2007-2015 time period. Notice how this first factor around 2013 tends to have lower values (values < 0). This means that after accounting for the constant increase/decrease in temperature across time and the seasonal cycle, locations whose loadings weight positively on this factor saw lower-than-typical temperatures during this time period.

```r
plot_factor(out.sm,
            together=FALSE,
            include.legend=FALSE,
            factor=1,
            type='mean')
```

To plot the factors together, set `together==TRUE`, as in the example code below (not run). We can think of the factors as "unknown" environmental behaviors. Thus, the factors capture common behaviors of the temperature measurements across time that weren't explicitly modeled (in contrast to the increasing/decreasing temperature and seasonal behaviors that were explicitly modeled).

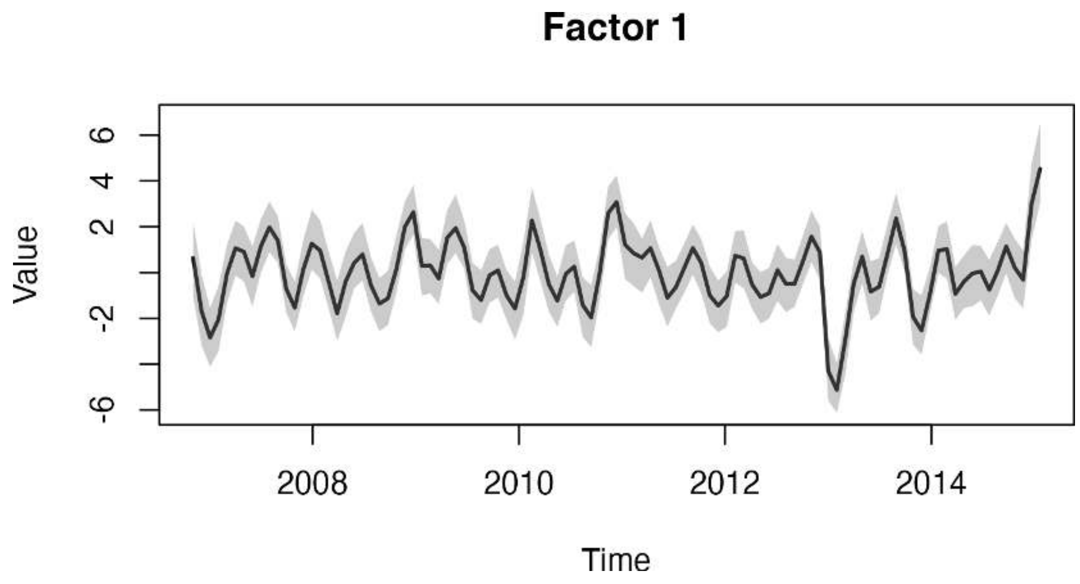

Figure 7: Posterior mean (black line) and 95% credible interval (gray band) of the first factor across the observed time period.

```
plot_factor(out.sm,
            together=TRUE,
            include.legend=TRUE,
            type='mean')
```

The `map_spatial_param` function is similar to `plot_spatial_param`, but the parameter is plotted on a grid of unobserved locations. If `map` is set to `FALSE`, the estimates will appear on a square grid. Setting, `map=TRUE`, `state=TRUE` and `location='utah'` uses the `sf` package (Pebesma, 2018) to import a map of Utah. The `fine` argument indicates the size of the grid; for instance, setting `fine=25` creates a 25×25 grid of locations to estimate the parameter values. This function is demonstrated below for the estimated linear increase/decrease of temperatures across Utah (once again, setting `yearscale=TRUE` to provide "per year" estimates). Figure 8 shows the model estimates that temperatures tend to be increasing across the state by between ≈ 0 and 0.3° C each year.

```
map_spatial_param(out.sm,
         parameter='slope',
         yearscale=TRUE,
         type='mean',
         map=TRUE,
         state=TRUE,
         location='utah',
         fine=25)
```

2.4 Speed

As mentioned before, the `BSTFA` package takes advantage of various mathematical and coding shortcuts to speed up computation. Specifically, the package uses sparse matrices, the vec operator, and basis functions to improve speed. The sparse matrices are implemented using the `Matrix` package. The basis functions reduce the number of parameters, thus reducing needed computation. In this section, we illustrate computational improvement over the fully-parameterized model for various number of bases.

The model details are covered in greater detail in Section 3, so here we supply a short description as to why `BSTFA` is much faster than `BSTFAfull`. Each function models the spatial and temporal dependence of the

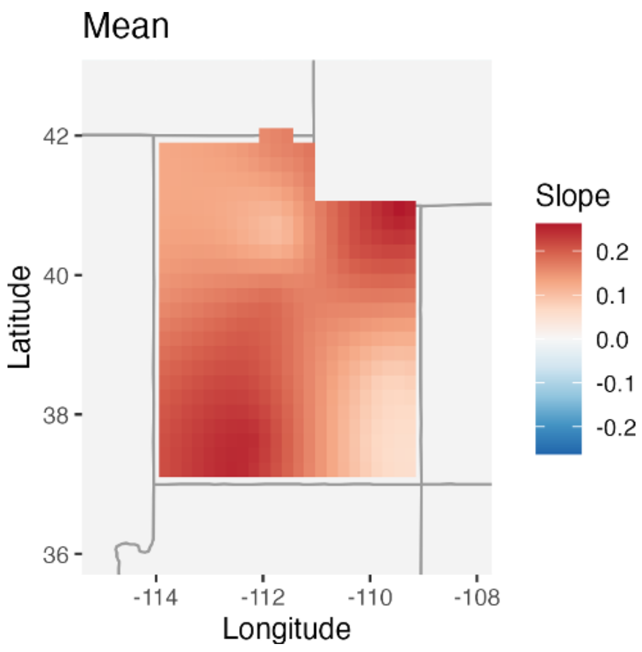

Figure 8: Posterior mean linear change in temperatures across time for locations across the observed space.

factor analysis differently. In `BSTFAfull`, we model the factors using a vector autoregressive model and the factor loadings with an exponential spatial dependence structure as in Berrett et al. (2020). This causes three main computational issues:

- The autoregressive step requires looping through all time points, with each point requiring matrix inversion and multiplication. This process can be sped up by making use of C++ within R, but even that takes too long in an MCMC algorithm when T gets remotely large (around 100 time points).
- Estimating the exponential spatial dependence structure for the loadings requires inversion of large, sometimes dense matrices (of size $n \times n$) in every iteration of the algorithm.
- Estimating the range parameters in this framework requires the Metropolis-Hastings algorithm which introduces increased computation time and potential inefficiency (e.g., smaller posterior effective sample sizes).

The `BSTFA` function instead fits both the factors and the loadings using basis functions of various forms, as discussed in Section 3. This solves the problems mentioned above by:

- Removing the need for a vector autoregressive loop.
- Lowering the dimension of the previously large matrices to invert.
- Introducing conjugacy. The entire model used in the `BSTFA` function is conditionally conjugate, allowing for an efficient Gibbs sampler without needing to use any Metropolis steps.

For reference, Table 3 compares the number of seconds per MCMC iteration for the `BSTFA` and `BSTFAfull` functions on simulated data with differing numbers of locations (first column) and bases (indicated by the different columns) for the factor loadings. In each instance, $T = 300$ and for the `BSTFA` function, the number of temporal bases is $R_t = 60$. The computations were carried out on a MacBook Pro (13-inch, 2022) with an Apple M2 chip (8-core CPU, 3.5 GHz) and 8 GB of RAM, running macOS 15.1.1.

Figures 9 and 10 illustrate that there is a tradeoff between computation speed and smoothness; that is, computation time is reduced at the cost of over-smoothing. The `BSTFAfull` returns fine-grain estimates with a high computational burden, while `BSTFA` provides a smooth representation of the process in a fraction of the time. Figure 9 compares the estimates for the third loading from both the `BSTFAfull` (left) and `BSTFA`

Table 3: Computation time in seconds per MCMC iteration for simulated data with n locations (first column) and $T = 300$ time points fit with the `BSTFA` function for different number of Fourier bases for the loadings (8, 20, and 50, indicated by the 2nd, 3rd, and 4th columns) all with $R_t = 60$ Fourier bases for the factors, compared to the `BSTFAfull` function (last column).

n	`BSTFA`, 8 loading bases	`BSTFA`, 20 loading bases	`BSTFA`, 50 loading bases	`BSTFAfull`
100	0.016	0.017	0.021	0.481
200	0.029	0.031	0.036	1.292
300	0.051	0.050	0.056	1.693
400	0.082	0.078	0.086	2.700
500	0.120	0.119	0.126	4.186

functions with 8 (center) and 6 (right) spatial bases on the loadings. The loading plot for `BSTFAfull` was fit with `fine=50` to save on computation time while the two `BSTFA` loading plots were fit with `fine=100`. Figure 10 compares the factor estimates from `BSTFAfull` (left) and `BSTFA` with 200 (center) and 126 (right) temporal bases on the factors. For both the loadings and the factors, the estimates computed by `BSTFAfull` and `BSTFA` display similar patterns, but the computationally-efficient `BSTFA` estimates are smoother spatially and temporally.

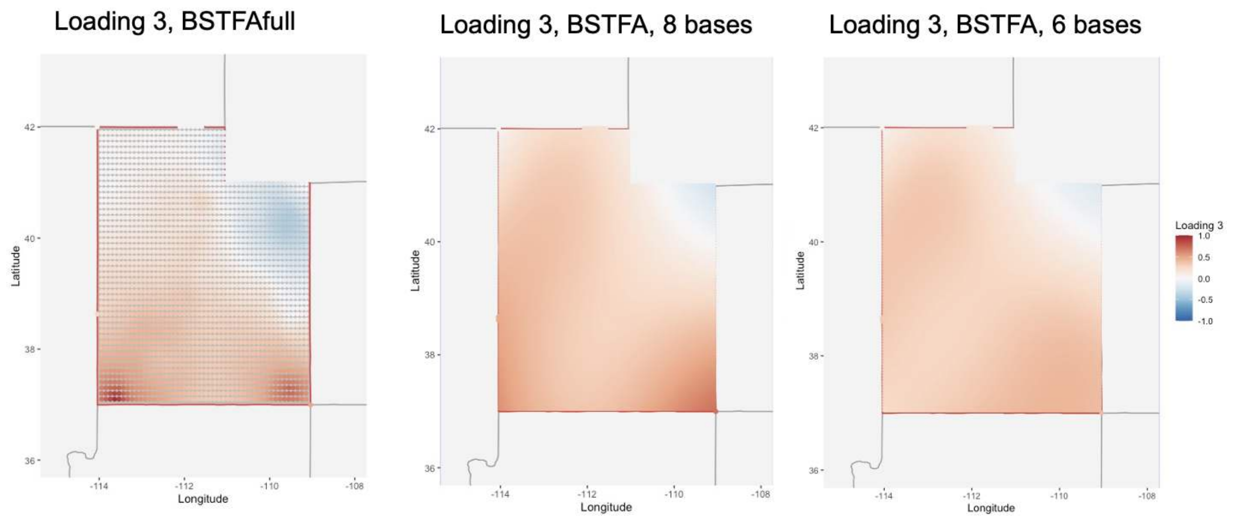

Figure 9: Comparison of estimated loadings for factor 3 using BSTFAfull (left), BSTFA with 8 spatial bases (center), and BSTFA with 6 spatial bases (right).

The `BSTFA` package contains a function `compute_summary` that takes as an argument the object from `BSTFA` or `BSTFAfull` and prints a detailed summary of computation time.

```
out <- BSTFA(ymat=utahDataList$TemperatureVals,
      dates=utahDataList$Dates,
      coords=utahDataList$Coords,
      save.time=TRUE)
compute_summary(out)
```

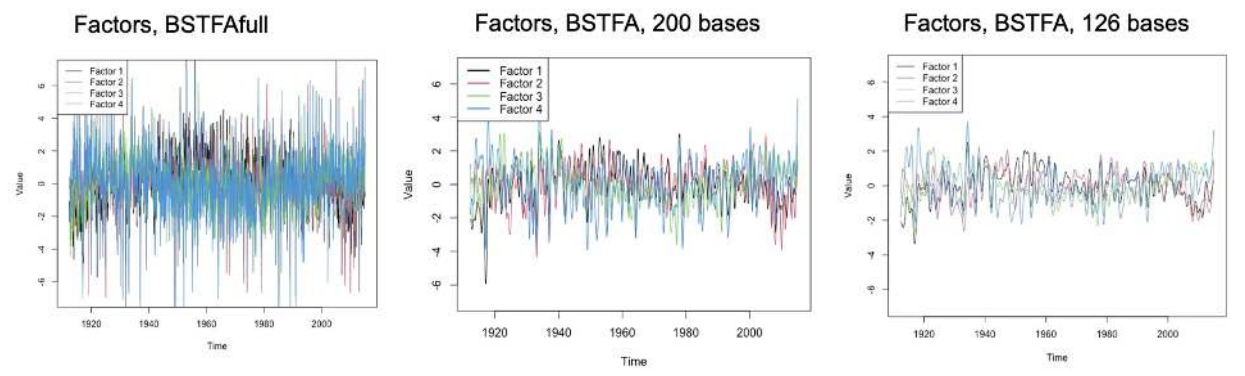

Figure 10: Comparison of estimated factors for factor 2 using BSTFAfull (left), BSTFA with 200 temporal bases (center), and BSTFA with 126 bases (right).

3 Methodology

This section details the methodology implemented in the `BSTFA` package. Specifically, the processes included in the model, basis function dimension reduction techniques, and details about spatio-temporal factor analysis are all discussed.

3.1 Model

The `BSTFA` package implements a Bayesian spatio-temporal factor analysis regression model. Our model follows the structure proposed by Berrett et al. (2020); namely, for a location $\mathbf{s} \in \mathcal{D}$ and a time index $t \in \mathcal{T}$, let $Y(\mathbf{s}, t)$ be the response variable such that

$$Y(\mathbf{s}, t) = \mu(\mathbf{s}) + (t - \bar{t})\beta(\mathbf{s}) + g(\xi(\mathbf{s}), t) + \mathbf{f}'(t)\lambda(\mathbf{s}) + \epsilon(\mathbf{s}, t)$$

where $\mu(\mathbf{s})$ represents the location-specific mean, $t - \bar{t}$ represents the time t centered by average time over the period of interest, $\beta(\mathbf{s})$ represents a spatially dependent linear slope in time, $g(\xi(\mathbf{s}), t)$ represents a spatially dependent seasonal periodic process, $\mathbf{f}'(t)\lambda(\mathbf{s})$ is a spatio-temporal confirmatory factor analysis (CFA) process, and $\epsilon(\mathbf{s}, t)$ is a zero-mean independent Gaussian residual process with variance σ^2. Note that both `BSTFA` and `BSTFAfull` assume the data are zero-centered and sets $\mu(\mathbf{s}) = 0$ for all $\mathbf{s}$ by default.

The approach described in Berrett et al. (2020) is implemented in the `BSTFAfull` function. However, as discussed before, the `BSTFA` function makes adjustments to the factor analysis component $\mathbf{f}'(t)\lambda(\mathbf{s})$ for increased computational speed. We provide a brief overview of the model for each interpretable process here, but for details, we refer the reader to Berrett et al. (2020).

3.1.1 Linear Component

The linear changes across time, $\beta(\mathbf{s})$, are allowed to vary spatially by using spatial basis functions. Let $\boldsymbol{\beta} = (\beta(\mathbf{s}_1), \ldots, \beta(\mathbf{s}_n))'$ be the $n \times 1$ vector of coefficients for the locations of interest. We model

$$\boldsymbol{\beta} \sim N(\mathbf{B}_\beta \boldsymbol{\alpha}_{\boldsymbol{\beta}}, \tau_\beta^2 \mathbf{I}),$$

where $\mathbf{B}_\beta$ is an $n \times b_\beta$ matrix of basis functions evaluated at each location, $\boldsymbol{\alpha}_\beta$ represents the corresponding $b_\beta \times 1$ vector of coefficients, τ_β^2 represents the variances, and $\mathbf{I}$ the appropriate identity matrix. We place conjugate priors on $\boldsymbol{\alpha}_\beta$,

$$\boldsymbol{\alpha}_\beta \sim N(0, \mathbf{A}^{-1})$$

where $\mathbf{A}$ is a diagonal precision matrix, and τ_β^2,

$$\frac{1}{\tau_\beta^2} \sim \text{Gamma}(\gamma, \phi),$$

Table 4: Arguments to `BSTFA` and `BSTFAfull` associated with the linear component.

Argument	Default Value	Description
`linear`	`TRUE`	TRUE/FALSE value indicating whether the linear component is included in the model.
`beta`	`NULL`	Vector of starting values for $\boldsymbol{\beta}$ of length $n \times 1$; if none is supplied, realistic starting values are calculated.
`alpha.prec`	`1e-5`	Value on the diagonal of the precision matrix $\mathbf{A}$.
`tau2.gamma`	`2`	Value of the shape parameter, γ, for the prior of the variance, τ^2_β.
`tau2.phi`	`1e-6`	Value of the rate parameter, ϕ, for the prior of the variance, τ^2_β.

Table 5: Arguments to `BSTFA` and `BSTFAfull` associated with the seasonal component.

Argument	Default Value	Description
`seasonal`	`TRUE`	TRUE/FALSE value indicating whether the seasonal component should be included in the model.
`xi`	`NULL`	Vector of starting values for $\boldsymbol{\xi}$ of length $un \times 1$; if none is supplied, realistic starting values are calculated.
`n.seasn.knots`	`7`	Value representing the value of u, the number of circular B-spline knots.

where ϕ is the rate parameter of the gamma distribution. Table 4 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the linear component.

3.1.2 Seasonal Component

Similarly, the spatially-dependent seasonal periodic process also uses basis functions. First, we use cubic circular b-splines (Wood, 2017) in time on the day of the year to model the periodic seasonal component. Let

$$g(\boldsymbol{\xi}(\mathbf{s}), t) = \mathbf{U}(t^*)\boldsymbol{\xi}(\mathbf{s}),$$

where $\mathbf{U}(t^*)$ is the $u \times 1$ vector of cubic circular B-splines evaluated at the day of year of time t, denoted by t^*, and $\boldsymbol{\xi}(\mathbf{s})$ the corresponding $u \times 1$ vector of coefficients. We then model the coefficients using the same approach used for the linear slopes. Namely, let $\boldsymbol{\xi}_j = (\boldsymbol{\xi}_j(\mathbf{s}_1), \ldots, \boldsymbol{\xi}_j(\mathbf{s}_n))'$ represent the coefficients for the jth spline for all locations of interest. Then,

$$\boldsymbol{\xi}_j \sim N(\mathbf{B}_{\xi_j}\boldsymbol{\alpha}_{\xi_j}, \tau^2_{\xi_j}\mathbf{I}),$$

where $\mathbf{B}_{\xi_j}$ is an $n \times b_{\xi_j}$ matrix of basis functions evaluated at each location, $\boldsymbol{\alpha}_{\xi_j}$ represents the corresponding $b_{\xi_j} \times 1$ vector of coefficients, $\tau^2_{\xi_j}$ represents the variance, and $\mathbf{I}$ the appropriate identity matrix. Each $\boldsymbol{\alpha}_{\xi_j}$ and $\tau^2_{\xi_j}$ are modeled with the same prior distributions (and same hyperparameters) as $\boldsymbol{\alpha}_\beta$ and τ^2_β. Table 5 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the seasonal component.

3.1.3 Factor Analysis Component

`BSTFAfull` uses a vector autoregressive model on the factors and an exponential Gaussian process model on the loadings.

Let L represents the number of factors so that $\mathbf{f}(t)$ is an $L \times 1$ vector of factors (a.k.a. scores) at time t and $\boldsymbol{\lambda}(\mathbf{s})$ is an $L \times 1$ vector of loadings for each factor at location $\mathbf{s}$. Define $\mathbf{F} = [\mathbf{f}(1) \cdots \mathbf{f}(T)]'$ to be the $T \times L$ matrix for all L factors and T times of interest and $\boldsymbol{\Lambda} = [\boldsymbol{\lambda}(\mathbf{s}_1) \cdots \boldsymbol{\lambda}(\mathbf{s}_n)]$ to be the $L \times n$ loading matrix for all n locations of interest. As required for identifiability by CFA, given L number of factors, we fix the values

Table 6: Arguments to `BSTFA` and `BSTFAfull` associated with the factor analysis component.

Argument	Default Value	Description
`factors`	`TRUE`	TRUE/FALSE value indicating whether the factor analysis component should be included in the model.
`Fmat`	`NULL`	Matrix of starting values for $\mathbf{F}$ of dimension $T \times L$; if none is supplied, realistic starting values are calculated.
`Lambda`	`NULL`	Matrix of starting values for $\boldsymbol{\Lambda}$ of dimension $n \times L$; if none is supplied, realistic starting values are calculated.
`factors.fixed`	`NULL`	Vector of indices (representing specific columns of `ymat`) indicating locations to fix for the factors. If no vector is supplied, fixed factor locations are optimally chosen according to distance and amount of non-missing data. If this vector is supplied, `n.factors = length(factors.fixed)`.
`n.factors`	`min(4, ceiling(ncol(ymat)/20))`	Number of factors to fit. If the number of locations is greater than 80, the function will always fit 4 factors unless otherwise specified in the `factors.fixed` argument.
`plot.factors`	`FALSE`	TRUE/FALSE value indicating whether to provide a base R plot of the fixed factor locations.

for the loadings for L locations with an L-rank matrix of constants (Rencher & Christensen, 2012). Table 6 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the factor analysis component.

3.1.4 Residual Component

The variance of the zero-mean independent Gaussian residual process, σ^2, is modeled with a conjugate prior similar to the other variance components,

$$\frac{1}{\sigma^2} \sim \text{Gamma}(\gamma_\sigma, \phi_\sigma),$$

where ϕ_σ is the rate parameter of the gamma distribution. Table 7 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the residual variance.

Table 7: Arguments to `BSTFA` and `BSTFAfull` associated with the residual.

Argument	Default Value	Description
`sig2`	`NULL`	A starting value for σ^2; if none is supplied, the starting value will be the variance of the non-missing values of `ymat`.
`sig2.gamma`	`2`	Value of the shape parameter, γ_σ, for the prior of the residual variance, σ^2.
`sig2.phi`	`1e-5`	Value of the rate parameter, ϕ_σ, for the prior of the overall residual variance, σ^2.

3.2 Basis Functions

Basis functions are a projection of a process onto a set of linear combinations of lower-dimension functions (Cressie et al., 2022). We make use of basis functions to allow for smooth estimates for each process across space and to drastically increase computational speed. For spatial modeling, the `BSTFA` package has three basis function forms built in: eigenvectors of an exponential correlation, Fourier bases, bisquare bases, and thin-plate spline bases. For temporal modeling, only Fourier bases are implemented. Eigenvectors (`eigen`) is the default approach in the `BSTFA` package. We discuss this and the Fourier basis approach in this vignette; for the others, we refer the reader to Cressie & Johannesson (2008) for bisquare bases and Nychka (2000) for thin-plate spline bases.

3.2.1 Eigenvector Basis Functions

The exponential correlation function for two locations $\mathbf{s}$ and $\mathbf{s}'$ can be written as,

$$\rho(\mathbf{s}, \mathbf{s}') = \exp(-||\mathbf{s} - \mathbf{s}'||/\phi),$$

where $||\cdot||$ represents the norm (or another distance metric) and ϕ is the range parameter. Note that the value of ϕ is determined in the `BSTFA` function using the value of `freq.lon`. We obtain the eigenvector bases by computing the correlation matrix made up of the correlation for each pair of observed locations and computing the eigenvalue decomposition on this matrix. Thus, let $\mathbf{R}$ represent this correlation matrix, the eigenvalue decomposition can be written using,

$$\mathbf{R} = \mathbf{\Psi}\mathbf{\Omega}\mathbf{\Psi}',$$

where the n columns of $\mathbf{\Psi}$ make up the eigenvectors and the diagonal matrix $\mathbf{\Omega}$ contains the eigenvalues. We use the first eigenvectors (indicated in the package as `n.spatial.bases` for the mean, linear, and seasonal processes, or `n.load.bases` for the loadings) as the bases matrix of spatial bases, $\mathbf{B}$, and a scaled version of the corresponding submatrix of $\mathbf{\Omega}$, denoted by $\dot{\mathbf{\Omega}}$, as the prior covariance of the estimated coefficients. Note that $\mathbf{R} \approx \mathbf{B}\dot{\mathbf{\Omega}}\mathbf{B}'$. Thus, for any spatially-dependent parameter, denoted generally by $\boldsymbol{\theta}$, we model,

$$\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{B}\boldsymbol{\alpha}_\theta, \tau^2_\theta \mathbf{I}),$$

with prior distribution,

$$\boldsymbol{\alpha}_\theta \sim \mathcal{N}(\mathbf{0}, a\,\dot{\mathbf{\Omega}}).$$

To estimate the spatial process at an unobserved location, we need to determine the appropriate values of the eigen bases for the new locations. Let $\mathbf{R}^{[adj]}$ represent the exponential correlation matrix of all observed and unobserved locations for interpolation, such that

$$\mathbf{R}^{[adj]} = \begin{bmatrix} \mathbf{R} & \mathbf{R}_{(obs,new)} \\ \mathbf{R}'_{(obs,new)} & \mathbf{R}_{(new)} \end{bmatrix},$$

where $\mathbf{R}_{(obs,new)}$ represents the correlation matrix specifically between the observed and new locations and $\mathbf{R}_{(new)}$ represents the correlation matrix specific to the new locations. Note that taking the eigenvalue decomposition of this adjusted correlation matrix will *not* result in correct basis values for the new locations. Instead, we can scale $\mathbf{R}_{(obs,new)}$ by the original $\mathbf{B}$ and $\dot{\mathbf{\Omega}}$. Specifically,

$$\mathbf{B}_{(new)} = \mathbf{R}'_{(obs,new)}\,\mathbf{B}\,\dot{\mathbf{\Omega}}.$$

Thus, values of the spatial parameter at the new location(s), $\mathbf{s}_{(new)}$, can be estimated using $\theta(\mathbf{s}_{(new)}) \sim \mathcal{N}(\mathbf{B}_{(new)}\boldsymbol{\alpha}_\theta, \tau^2_\theta \mathbf{I})$.

3.2.2 Fourier Basis Functions

We use Fourier bases because of their connection to Gaussian processes and their computational flexibility. A Gaussian process can be approximated quite well with orthogonal spectral basis functions (Wikle, 2002). One example is to use some number of principal components of the spatial covariance matrix. These spectral basis functions can themselves be represented as a sum of sine and cosine functions (Paciorek, 2007) reminiscent of a trigonometric Fourier series. Fourier bases, then, can capture the frequencies exhibited in the principal components of the underlying Gaussian process, granting a smooth approximation of the process.

We make use of Fourier basis functions for both spatial and temporal dependence. First, consider the Fourier bases for temporal dependence. Let $Q_r^T(t)$ and $Q_{r+1}^T(t)$ be the r^{th} and $(r+1)^{\text{th}}$ columns of the matrix of bases for time t for $r = 1, 3, 5, \ldots R_t$. Then,

$$Q_r^T(t) = \sin\left(2\pi \frac{r+1}{2} \frac{t}{f_t}\right),$$

$$Q_{r+1}^T(t) = \cos\left(2\pi \frac{r+1}{2} \frac{t}{f_t}\right),$$

where f_t is the frequency of the Fourier function.

For the spatial basis functions, we must accommodate the two-dimensional nature of space. Thus, we must multiply the sine and cosine functions for each dimension (Paciorek, 2007). Let $\mathbf{B}_r(\mathbf{s})$ represent the r^{th} through $(r+3)^{\text{th}}$ columns of the matrix of bases evaluated at location $\mathbf{s}$. Then,

$$\mathbf{B}_r(\mathbf{s}) = \begin{bmatrix} \sin\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[1]}}{f_{\mathbf{s}_{[1]}}}\right) \times \sin\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[2]}}{f_{\mathbf{s}_{[2]}}}\right) \\ \sin\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[1]}}{f_{\mathbf{s}_{[1]}}}\right) \times \cos\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[2]}}{f_{\mathbf{s}_{[2]}}}\right) \\ \cos\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[1]}}{f_{\mathbf{s}_{[1]}}}\right) \times \sin\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[2]}}{f_{\mathbf{s}_{[2]}}}\right) \\ \cos\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[1]}}{f_{\mathbf{s}_{[1]}}}\right) \times \cos\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[2]}}{f_{\mathbf{s}_{[2]}}}\right) \end{bmatrix}',$$

where $\mathbf{s}_{[1]}$ represents the first coordinate of $\mathbf{s}$ (e.g., longitude), and $\mathbf{s}_{[2]}$ the second coordinate (e.g., latitude), and $f_{\mathbf{s}_{[1]}}$ and $f_{\mathbf{s}_{[2]}}$ are the corresponding frequencies of the Fourier functions.

The `BSTFA` package contains a helper function to visualize spatial Fourier bases over a given set of coordinates. This can be useful when trying to choose the value of the spatial frequencies, $f_{s_{[1]}}$ and $f_{s_{[2]}}$ (`freq.lon` and `freq.lat`) and number of bases (R_s, or `n.load.bases`) to include in the model. This function is demonstrated below using the Utah temperature data.

```
plot_fourier_bases(utahDataList$Coords,
                   R=6,
                   plot.3d=TRUE,
                   freq.lon = 4*diff(range(utahDataList$Coords[,1])),
                   freq.lat = 4*diff(range(utahDataList$Coords[,2])))
```

3.3 Spatio-Temporal Factor Analysis using Basis Functions

We model the factors and loadings using the bases in the following way. Let $\tilde{\mathbf{F}} = \text{vec}(\mathbf{F})$, be the vectorized $TL \times 1$ vector of all factors, and $\tilde{\boldsymbol{\Lambda}} = \text{vec}(\boldsymbol{\Lambda})$ be the vectorized $Ln \times 1$ vector of all loadings. We model $\tilde{\mathbf{F}}$ and $\tilde{\boldsymbol{\Lambda}}$ using a similar basis function decomposition used for the coefficients of the other processes described in 2.1.1 and 2.1.2; namely,

$$\tilde{\mathbf{F}} = (\mathbf{I}_L \otimes \mathbf{Q}^{\mathbf{T}})\alpha_F,$$

$$\tilde{\boldsymbol{\Lambda}} \sim N\left((\mathbf{Q}^{\mathbf{S}} \otimes \mathbf{I}_L)\alpha_\Lambda, \tau_\Lambda^2 \mathbf{I}_{Ln}\right),$$

where $\mathbf{Q}^{\mathbf{T}}$ is a $T \times (R_t + 1)$ matrix of temporal bases, $\mathbf{Q}^{\mathbf{S}}$ is an $n \times (R_s + 1)$ matrix of spatial bases, $\mathbf{I}_L$ is the $L \times L$ identity matrix, α_F is an $(R_t + 1)L \times 1$ vector of coefficients, α_Λ is an $L(R_s + 1) \times 1$ vector of coefficients, and τ_Λ^2 is the residual variance for the loadings.

A word about notation: the `BSTFA` function allows the spatial bases for the mean, linear, and seasonal components to be different from the spatial bases used for the loadings. Thus, the notation for the spatial bases for the loadings are distinguished here using Q^S, although they are determined the same as $\mathbf{B}$.

Both sets of coefficients are modeled *a priori* in the same way as α_β and α_{ξ_j}, namely,

$$\alpha_F \sim N(\mathbf{0}, \mathbf{A}_{\alpha_F}^{-1}),$$

$$\alpha_\Lambda \sim N(\mathbf{0}, \mathbf{A}_{\alpha_\Lambda}^{-1}),$$

and the variance component for the loadings τ_Λ^2 the same as τ_β^2 and $\tau_{\xi_j}^2$,

$$\frac{1}{\tau_\Lambda^2} \sim \text{Gamma}(\gamma, \phi).$$

Once again, the hyperparameters for the variance take the same argument values as used in the linear and seasonal components. Table 8 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated the with basis functions.

4 Useful Features

4.1 Fixing Factors

Factor analysis allows for interpretability of the factors and/or loadings. Since loadings are spatially dependent, it makes sense to use a geographic interpretation. Thus, to model the Utah temperature data, we choose locations to fix that will lend factor interpretation to West (Wendover), East (Moab), South (Kanab), and North (Logan) factors, shown by the large red dots in Figure 11. In this instance, with $L = 4$ factors chosen, the L-rank matrix of constants is the $L \times L$ identity matrix. This is the matrix for fixed factors used in the `BSTFA` and `BSTFAfull` functions.

It's important that the fixed factor locations have a low proportion of missing data. If fixed factor locations are not given, they will be smartly chosen by the function according to distance and proportion of missing data.

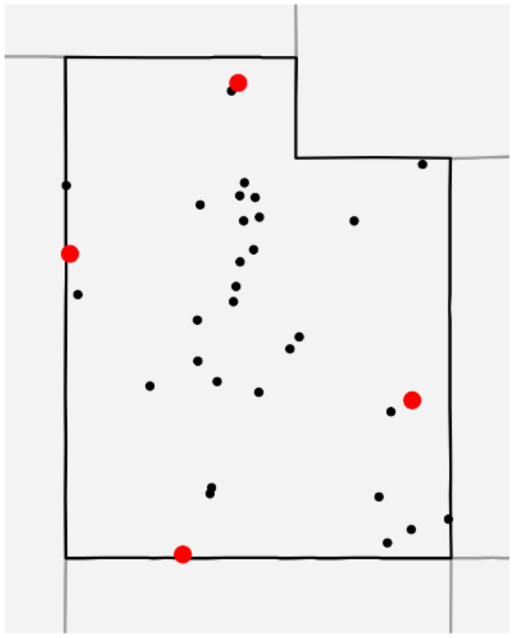

Figure 11: Plot of Utah showing locations of fixed factors (in red) and all locations (in black).

Because Ibapah is the first fixed loading location (western-most red dot in Figure 11), the map of estimates for the first loading indicate the strength of the relationship of each location's temperatures to Ibapah's temperatures, after accounting for linear and seasonal behavior. Higher loading values indicate greater similarity. We can use the `map_spatial_param` to plot the estimated loadings across a grid over the observed locations by using `parameter='loading'` and `loading=1` (for the first loading).

Table 8: Arguments to `BSTFA` and `BSTFAfull` associated with basis functions used for various components of the model.

Argument	Default Value	Description
`spatial.style`	`'fourier'`	Indicates which type of basis functions to use for the linear and seasonal components. The default is `'fourier'`. Other values accepted are `'grid'` (for multiresolution bisquare bases) and `'tps'` (for thin-plate splines).
`n.spatial.bases`	`8`	Number of basis functions to use for the linear and seasonal components. For Fourier bases, this value is R_s. For bisquare bases, this value is ignored. For thin-plate spline bases, the number of bases is `floor(sqrt(n.spatial.bases))`$\hat{2}$ to create an even grid.
`load.style`	`'fourier'`	The same as `spatial.style` but for the factor loadings. This does not have to be the same bases as `spatial.style`.
`n.load.bases`	`6`	The same as `n.spatial.bases` but for the factor loadings. This does not have to be the same vvalue as `n.spatial.bases`.
`freq.lon`	`4*diff(range(coords[,1]))`	Spatial range parameter for the eigenvalue bases (ϕ). For the Fourier bases, this is the frequency for longitude ($f_{s_{[1]}}$; or, if using other coordinate system, the first coordinate value). Default value is two times the range of the longitude coordinates. If not using `'eigen'` or `'fourier'`, this argument is not used.
`freq.lat`	`4*diff(range(coords[,2]))`	Same as `freq.lon` for the Fourier bases but for latitude (or the second coordinate value).
`n.temp.bases`	`floor(n.times/10)`	Number of Fourier basis functions to use for the temporally-dependent factors. This value is R_t. The default value is `floor(n.times/10)`.
`freq.temp`	`n.times`	Frequency of the Fourier bases functions for the temporally-dependent factors. This value is f_t. Default value is T (`n.times`).
`knot.levels`	`2`	The number of resolutions when using the bisquare basis functions. If not using bisquare bases, this argument is not used.
`max.knot.dist`	`mean(dist(coords))`	The distance beyond which a location is considered 'too far' from a knot, meaning its basis function value associated with that knot evaluates to zero. If not using bisquare bases, this argument is not used.
`premade.knots`	`NULL`	A list of coordinates containing pre-specified knots. Each element of the list is a resolution. Each resolution should have the same number of columns as `coords`. If not using bisquare bases, this argument is not used.
`plot.knots`	`FALSE`	TRUE/FALSE value indicating whether to provide a base R plot of the knot resolutions overlaid on top of the given `coords`. If not using bisquare bases, this argument is not used.

4.2 Basis Function Details

The choice of basis functions is assigned using the `spatial.style` and `load.style` arguments. The `spatial.style` argument controls which basis functions to use for the linear and seasonal components ($\mathbf{B}_\beta$ and each $\mathbf{B}_{\xi_j}$) while the `load.style` argument controls the basis functions for the factor loadings ($\mathbf{Q}^\mathbf{S}$). The number of bases for the linear and seasonal components is specified with the argument `n.spatial.bases` while the loadings use the argument `n.load.bases`. The values given to `spatial.style` and `load.style` need not be the same, nor do `n.spatial.bases` and `n.load.bases`. The default value for both style arguments is `"fourier"`. The only basis functions used for the temporally-dependent factors are Fourier bases.

4.2.1 Fourier Bases

When using Fourier bases, the user needs to specify number of bases and the spatial frequency in both the longitude and latitude directions. As demonstrated in Section 3.2, the function `plot_fourier_bases` can help the user visualize Fourier bases and choose the appropriate amount of bases and frequencies. After exploratory analysis methods (demonstrated in Section 4.2.4), it seems that assigning `freq.lon` and `freq.lat` values of 40 and 30 respectively and setting `n.spatial.bases` and `n.load.bases` equal to 8 and 6, respectively, works well for the Utah data set.

The user should also consider the frequency (`freq.temp`) and number of bases (`n.temp.bases`) for the temporal factors, which always use Fourier bases. The default values (see Table 8) tend to work well, but increasing the number of bases can create a finer estimate at the cost of reduced computational speed.

```r
out <- BSTFA(ymat=utahDataList$TemperatureVals,
      dates=utahDataList$Dates,
      coords=utahDataList$Coords,
      spatial.style='fourier',
      load.style='fourier',
      n.spatial.bases=8,
      n.load.bases=6,
      freq.lon=40,
      freq.lat=30,
      n.temp.bases=floor(nrow(utahDataList$TemperatureVals)/10),
      freq.temp=nrow(utahDataList$TemperatureVals))
```

4.2.2 Bisquare Bases

As described in Table 2.5, multiple arguments to `BSTFA` and `BSTFAfull` are used only for bisquare bases. The argument given to `spatial.style` or `load.style` to use these basis functions is `'grid'`. The `knot.levels` argument indicates how many resolutions of knots to create, where the r^{th} resolution uses 2^{2r} bases distributed evenly in a square grid across the coordinates of the data. Setting `plot.knots=TRUE` outputs a plot of knots in all resolutions. The code below shows how to use this version of bases in the `BSTFA` function for two resolutions of knots and the data locations for the Utah temperature data. Using `plot.knots=TRUE` will provide a plot of the locations of the knots.

```r
bstfa.plot_knots = BSTFA(ymat=utahDataList$TemperatureVals,
                         dates=utahDataList$Dates,
                         coords=utahDataList$Coords,
                         spatial.style='grid',
                         load.style='grid',
                         knot.levels=2,
                         plot.knots=TRUE,
                         verbose=FALSE,
                         iters=5)
```

The user can specify custom knot locations with the `premade.knots` argument. This argument takes a list of coordinates containing pre-specified knots. The number of elements in the list represents the number of resolutions. Each element of the list should have the same number of columns as `coords`. An example of how to do this is provided in the code below.

```r
knots=list()
max.lon = max(utahDataList$Coords[,1])
min.lon = min(utahDataList$Coords[,1])
max.lat = max(utahDataList$Coords[,2])
min.lat = min(utahDataList$Coords[,2])
range.lon = max.lon-min.lon
range.lat = max.lat-min.lat
knots[[1]] = expand.grid(c(min.lon+(range.lon/4), min.lon+3*(range.lon/4)),
            c(min.lat+(range.lat/4), min.lat+3*(range.lat/4)))
knots[[2]] = expand.grid(c(min.lon+(range.lon/6),
                           min.lon+(range.lon/2),
                           min.lon+5*(range.lon/6)),
                         c(min.lat+(range.lat/6),
                           min.lat+(range.lat/2),
                           min.lat+5*(range.lat/6)))
bstfa.custom_knots = BSTFA(ymat=utahDataList$TemperatureVals,
                           dates=utahDataList$Dates,
                           coords=utahDataList$Coords,
                           spatial.style='grid',
                           load.style='grid',
                           knot.levels=2,
                           plot.knots=TRUE,
                           premade.knots=knots,
                           verbose=FALSE,
                           iters=5)
```

4.2.3 Thin-Plate Spline Bases

The argument given to `spatial.style` and `load.style` to use these basis functions is `'tps'`. The function `basis.tps` from the `npreg` package is used to create the thin-plate spline bases. The knots used to create the bases are on a square grid; thus, the number of bases is equal to `floor(sqrt(n.spatial.bases))^2` and `floor(sqrt(n.load.bases))^2`. So, even if the values 8 and 10 are given to `n.spatial.bases` and `n.load.bases` as shown in the code below, the number of bases used in the model will be 4 and 9.

```r
bstfaTPS <- BSTFA(ymat=utahDataList$TemperatureVals,
      dates=utahDataList$Dates,
      coords=utahDataList$Coords,
      spatial.style='tps',
      load.style='tps',
      n.spatial.bases=8,
      n.load.bases=10)
```

4.2.4 Choosing Basis Functions

There are few ways to decide which spatial basis functions to use for your data. First, diagnostics such as the Watanabe-Akaike Information Criterion (WAIC) and Leave-One-Out Cross-Validation (LOO-CV) can help the user compare model fits (Vehtari et al., 2017). These can be computed using the `waic` and `loo` functions from the `loo` package. These functions require a matrix of log-likelihood values, which can be generated using the function `computeLogLik` from the `BSTFA` package, supplying as argument the output from `BSTFA` or `BSTFAfull`.

Table 9: Summary of model performance diagnostics, LOO-CV and WAIC, for different basis functions.

Type	# of Spatial Bases	# of Temporal Bases	LOO-CV	WAIC
Fourier	8	126	202.7	195.8
Fourier	16	126	203.2	195.4
Bisquare	4 (1-level)	126	203.4	195.4
Bisquare	20 (2-levels)	126	203.6	195.3
TPS	9	126	**202.6**	195.7
TPS	16	126	203.2	195.6
Fourier	8	200	204.6	191.6
Fourier	16	200	205.7	**191.4**
Bisquare	4 (1-level)	200	204.7	191.6
Bisquare	20 (2-levels)	200	203.9	191.9
TPS	9	200	205.2	191.7
TPS	16	200	204.8	191.8

```r
loglik = computeLogLik(out.sm,
                       verbose=FALSE)
loo::waic(t(loglik))
loo::loo(t(loglik))
```

Table 9 compares the WAIC and LOO-CV for 12 different models fit to the full Utah temperature data. In each instance, the number of spatial bases is the same for both the linear/seasonal components and the factor analysis component – that is, `n.spatial.bases` = `n.load.bases`. The number of temporal bases is the `n.temp.bases` argument; 126 is the default for the Utah data (10% of T).

Notice that in this case, the choice of spatial basis function does not matter much, while increasing the number of temporal bases from 126 to 200 leads to a reduction in WAIC, indicating better model fit. However, increasing the number of temporal bases reduces computational efficiency (in this instance, moving from 126 to 200 temporal bases added about 0.2 seconds to each iteration).

In addition to model diagnostics, it's also important to visually assess estimates using the different basis functions and other settings. For instance, this can be done by fitting a model with each basis function and estimating the linear slope and loadings using `map_spatial_param`, as demonstrated in Section 2.3. Figure 12 shows three estimates of the second loading (based on the "East" factor for fixed loading Moab, Utah) when the loadings are fit with Fourier bases (using default values for frequency), bisquare bases, and thin-plate spline bases. The estimates for the different basis functions look quite different; this is partly because the corresponding estimated factors are different.

4.3 Assessing MCMC Convergence

The `BSTFA` package has built-in helper functions for assessing convergence. These functions use the fact that all matrices of parameter draws are MCMC objects from the `coda` package. To look at trace plots, you can use the `plot_trace` function. This function takes as input your `BSTFA` or `BSTFAfull` object, a string value `parameter` indicating which parameter to view (corresponds directly to what the parameter is called in the `BSTFA` list output), and `param.range` which accepts a numeric vector indicating which of these parameters you want to view.

```r
plot_trace(out.sm,
           parameter='beta',
           param.range=c(27),
           density=FALSE)
```

The other available helper function is `convergence_diag`. This function takes as input the `BSTFA` or `BSTFAfull`

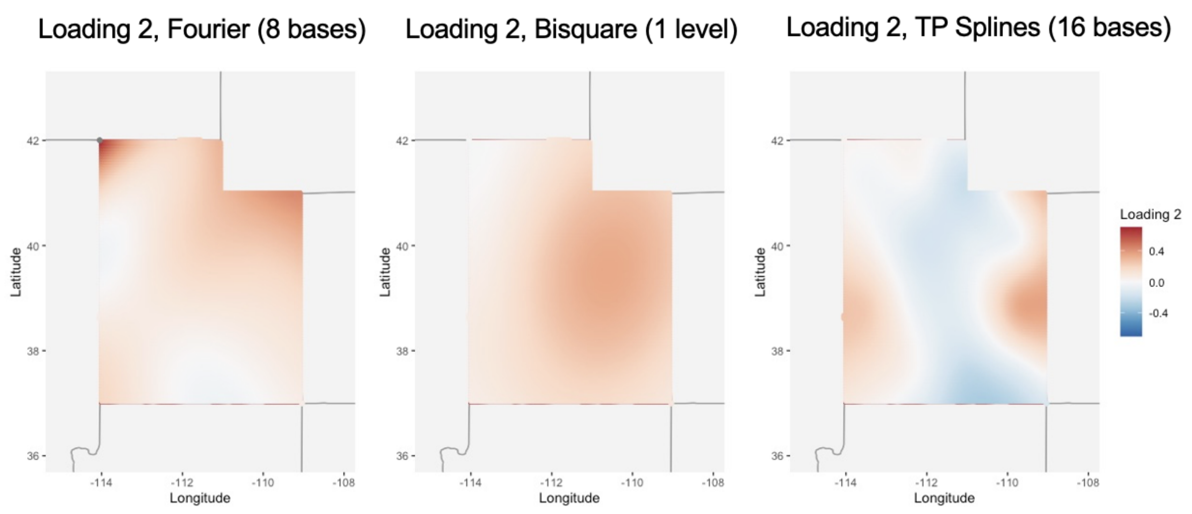
Loading 2, Fourier (8 bases)
Loading 2, Bisquare (1 level)
Loading 2, TP Splines (16 bases)
Latitude
Longitude
42
40
38
36
-114
-112
-110
-108
Loading 2
0.4
0.0
-0.4

Figure 12: Posterior mean estimates of the second loading for different style of spatial bases: Fourier (left), bisquare (center), and thin-plate spline (right).

function output and returns the effective sample size or Geweke diagnostic (indicated by `type='eSS'` or `type='geweke'`) for all parameters above a given cutoff (indicated by `cutoff`). For instance, the function below will return all parameters with an effective sample size below 100.

```
convergence_diag(out.sm,
                 type='geweke',
                 cutoff = 2)
```

We encourage the user to read additional resources on MCMC and convergence, such as Johnson et al. (2022) for introductory readers or Gelman et al. (2013) for more advanced readers.

5 Appendices

5.1 Computation Notes

Below are specific notes about computation not explicitly mentioned in the vignette:

- The values for `n.spatial.bases`, `n.load.bases` and `n.temp.bases` need to be even numbers. If they are not, the function will add 1 to the supplied value.
- To help with convergence of the residual factor analysis component, the sampler waits to sample $\mathbf{F}$ and $\boldsymbol{\Lambda}$ until `min(floor(burn/2), 500)`. That is why, for example, the `compute_summary` function divides computation time into “Pre-Factor Analysis” and “Post-Factor Analysis”.

References

Berrett, C., Christensen, W. F., Sain, S. R., Sandholtz, N., Coats, D. W., Tebaldi, C., & Lopes, H. F. (2020). Modeling sea-level processes on the U.S. Atlantic Coast. *Environmetrics*, *31*, e2609. https://doi.org/10.1002/env.2609

Cressie, N., & Johannesson, G. (2008). Fixed rank kriging for very large spatial data sets. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *70*(1), 209–226.

Cressie, N., Sainsbury-Dale, M., & Zammit-Mangion, A. (2022). Basis-function models in spatial statistics. *Annual Review of Statistics and Its Applications*, *9*, 373–400. https://doi.org/10.1146/annurev-statistics-

040120-020733

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., & Rubin, D. B. (2013). *Bayesian Data Analysis* (Third). CRC. https://stat.columbia.edu/~gelman/book/

Johnson, A. A., Ott, M. Q., & Dogucu, M. (2022). *Bayes Rules! An Introduction to Applied Bayesian Modeling*. CRC. https://doi.org/10.1201/9780429288340

Nychka, D. W. (2000). Spatial-process estimates as smoothers. In M. G. Schimek (Ed.), *Smoothing and Regression: Approaches, Computation, and Application*. Wiley.

Paciorek, C. J. (2007). Bayesian Smoothing with Gaussian Processes Using Fourier Basis Functions in the spectralGP Package. *Journal of Statistical Software*, *19*, 1–38. https://doi.org/10.18637/jss.v019.i02

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

Plummer, M., Best, N., Cowles, K., & Vines, K. (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, *6*(1), 7–11. https://journal.r-project.org/archive/

Rencher, A. C., & Christensen, W. F. (2012). *Methods of Multivariate Analysis* (Third). Wiley.

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*, 1413–1432. https://doi.org/10.1007/s11222-016-9696-4

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org

Wikle, C. (2002). Spatial modeling of count data: A case study in modelling breeding bird survey data on large spatial domains. In A. Lawson & D. Denison (Eds.), *Spatial cluster modelling* (pp. 199–209). Chapman & Hall.

Wood, S. N. (2017). *Generalized Additive Models: An introduction with R* (2nd ed.). Chapman; Hall/CRC.