

BSTFA

Abstract

The BSTFA package for R is a tool for fitting fully Bayesian spatio-temporal factor analysis models with a normal likelihood. The package implements a computationally rapid approach using dimension reduction via basis functions. The package also supports a non-reduced spatio-temporal factor analysis model, albeit with much slower computation. Also included are functions to predict and plot the response variable at missing/unknown locations, methods to visualize processes (such as the linear trend through time) on a grid or map, and a function to plot estimated behavior through some given timeframe (such as annual seasonal behavior).

Table of Contents

- Intended Audience
- Motivation
- 1 What is Implemented?
 - 1.1 Model-fitting Functions
 - Table 1.1
 - 1.1.1 Understanding Output
 - 1.2 Prediction
 - 1.3 Plotting/Visualization
 - Table 1.2
 - 1.4 Speed
 - Table 1.3
- 2 Methodology
 - 2.1 Model
 - 2.1.1 Linear Component
 - Table 2.1
 - 2.1.2 Seasonal Component
 - Table 2.2
 - 2.1.3 Factor Analysis Component
 - Table 2.3
 - 2.2 Basis Functions
 - Table 2.4
- 3 Useful Features
 - 3.1 Fixing factors
 - 3.2 Choosing Basis Functions
 - 3.2.1 Fourier
 - 3.2.2 Bisquare
 - 3.2.3 Thin-plate splines
 - 3.3 Assessing MCMC Convergence
- 4 Appendices
 - 4.1 Additional Notes
 - References

FIX ME - size of plots.

FIX ME - Include σ^2 hyperparameters?

Intended Audience

This document is intended to help even novice statistics students implement a fully Bayesian spatio-temporal analysis. Every function within the package is designed with this audience in mind. This document is meant to guide any potential user of this package through the basic theory and implementation of our models: in essence, it is an instruction manual. The bulk of this document contains examples of our functions applied on a real dataset.

The outline is as follows. First, we introduce the motivation behind our Bayesian spatio-temporal factor analysis models and why simplifying the model for fast computation is important. Section 1 outlines the available functions and procedures within the BSTFA package. Some basic theory and methodology are contained in Section 2, and Section 3 demonstrates the BSTFA package on simulated and real world data. The appendix details additional resources, such as...

Motivation

Consider for example the motivating data 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 show obvious 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 0.1 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 decreases dramatically from winter 2000 to winter 2001, showing that locations near each other in space exhibit similar environmental trends. We don't know from the data alone what contributed to that difference; we simply know it exists. We need a model that provides inference into the spatio-temporal relationships. Spatio-temporal factor analysis allows us to account for underlying spatio-temporal dependencies and explore numerical and visual summaries of that dependence.

A fully parameterized Bayesian approach to spatio-temporal factor analysis is computationally burdensome. Our 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, prediction, and inference.

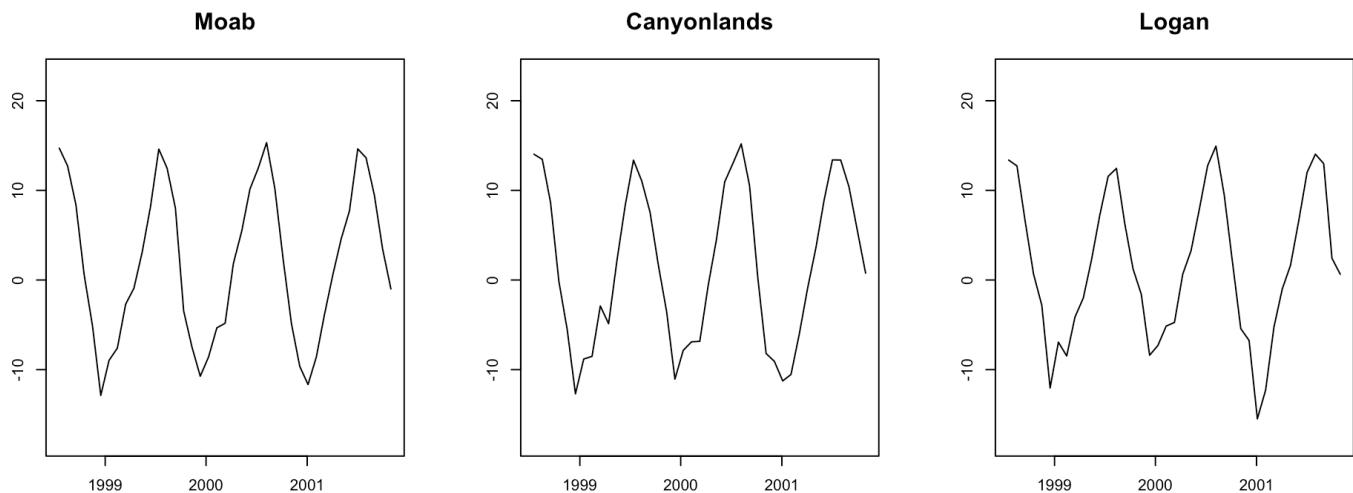


Figure 0.1: Mean-centered 30-day average daily minimum temperatures for 3 Utah weather stations (Moab, left; Canyonlands, middle; Logan, right) from 1999 to 2001.

1 What is Implemented?

The BSTFA package contains implementation of two versions of a spatio-temporal factor analysis model along with functions for prediction, plotting and visualizing posterior surfaces. The model-fitting functions are defined in Section 1.1, while the methodology associated with these models is described more fully in Section 2. The BSTFA package's prediction methods are discussed in Section 1.2, functions for plotting/visualization are described in Section 1.3, and notes about computational speed are outlined in Section 1.4.

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.

1.1 Model-fitting Functions

The BSTFA package contains two model fitting functions: `BSTFA`, the smoother but computationally-efficient spatio-temporal factor analysis model using basis functions for the factor analysis; and `BSTFAfull`, the fine-grain but 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 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 1.1. Other arguments relating to model fitting and prior specification will be discussed more fully in Section 2.

Table 1.1

Table 1.1: This table summarizes required arguments for the `BSTFA` and `BSTFAfull` functions.

Argument	Description
<code>ymat</code>	A matrix of response values. Each row should represent a point in time; each column should represent a specific location. Missing observations should be recorded as NA.
<code>dates</code>	A vector of dates of length <code>nrow(ymat)</code> . The model functions will transform this vector into “doy” (day of year) using <code>lubridate::yday()</code> . Thus, this vector must either be a <code>lubridate</code> or string object with year-month-day format.

Argument**Description**

coords	A matrix/data frame of coordinate values with 2 columns and number of rows equal to <code>ncol(ymat)</code> . If using longitude/latitude, longitude should be the first column.
--------	--

The `BSTFA` and `BSTFAfull` functions are demonstrated below. Additional 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 samples. The `verbose` argument controls whether or not the function gives status updates during sampling. Though the default value for this is `TRUE`, for the sake of this vignette, `verbose` will always be set to `FALSE`. Also, if desired, the function can save output directly to a `.Rdata` file by setting `save.output == TRUE`. This value is set to `FALSE` by default.

```
bstfa.output = BSTFA(ymat=utahDataList$TemperatureVals,
                     dates=utahDataList$Dates,
                     coords=utahDataList$Coords,
                     verbose=FALSE)
```

```
bstfa_full.output = BSTFAfull(ymat=utahDataList$TemperatureVals,
                             dates=utahDataList$Dates,
                             coords=utahDataList$Coords,
                             verbose=FALSE)
```

1.1.1 Understanding Output

The `BSTFA` and `BSTFAfull` functions output a list object. 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 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 `computation.summary` mentioned in Section 1.4.
- `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 estimates for the missing data points. 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 2).
 - `newS` is equivalent to $\mathbf{B}_\beta = \mathbf{B}_{\xi_j} \forall j$ and has dimension $n \times b_\beta$, where `$b_= b_= $ n.spatial.bases`.
 - `linear.Tsub` is $t - \bar{t}$, $T \times 1$ 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 factors.

- QS is the $n \times R_s$ matrix of basis functions used for the 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 `F.tilde` and `Lambda.tilde` (discussed in greater detail in Section 2.1.3). 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.

1.2 Prediction

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 generate predictions either for the spatio-temporal processes, $Y(s, t)$, at observed location s and time t , or for, $Y(s^*, t)$, at unobserved location s^* at time t . The function also takes as input either a location number (corresponding to the appropriate column of your `ymat` argument) for predicting at an observed location, or a matrix/data frame of coordinate values if predicting at a new location.

If the argument `type` is set to "all", the function will return all draws of Y . `type` can also be set to "mean", "median", "ub" (upper bound), or "lb" (lower bound). The option `pred.int` controls whether your uncertainty is a prediction interval (`pred.int == TRUE`; default value) or a credible interval (`pred.int == FALSE`). The function is demonstrated below.

Predicting at an observed location. Since `type == "all"`, the function will return a $t \times d$ MCMC object matrix of posterior draws of Y with d being the number of MCMC draws. Each column represents one draw of Y .

```
loc = 1 # Alpine, Utah in our dataset
preds = predictBSTFA(bstfa.output,
                      location = loc,
                      type='all',
                      pred.int=TRUE,
                      ci.level=c(0.025,0.975))
dim(preds)
```

Setting `type == "mean"`, for example, returns a $t \times 1$ MCMC object vector containing the posterior mean of Y at each time point t .

```
loc = 1 # Alpine, Utah in our dataset
preds = predictBSTFA(bstfa.output,
                      location = loc,
                      type='mean',
                      pred.int=TRUE,
                      ci.level=c(0.025,0.975))
dim(preds)
length(preds)
```

Predicting at an unobserved location.

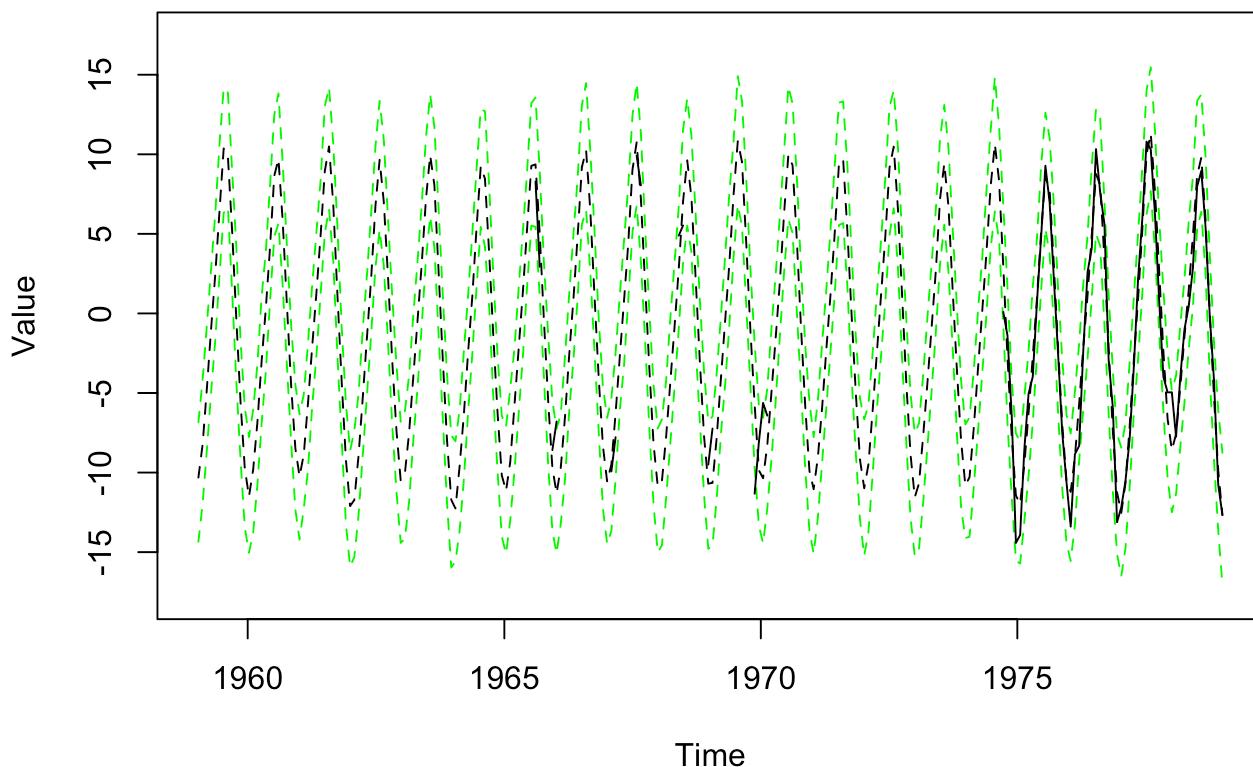
```
loc = data.frame('Longitude' = 111.41, 'Latitude' = 38.29) # Torrey, Utah
preds = predictBSTFA(bstfa.output,
                      location = loc,
                      type='all',
                      pred.int=TRUE,
                      ci.level=c(0.025,0.975))
dim(preds)
```

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` indicates the time points to plot the estimates, with default value `NULL` indicating to plot on all of `t`. `truth` (default value `FALSE`) will plot the observed data underneath the estimates if the `location` comes from the data set. `uncertainty` (default value `TRUE`) indicates whether to plot a prediction interval (`pred.int==TRUE`) or a credible interval (`pred.int==FALSE`) with the estimate.

Plotting predictions at an observed location:

```
loc = 1 # Alpine, Utah in our dataset
plot.location(bstfa.output,
              location=loc,
              type='mean',
              pred.int=TRUE,
              uncertainty=TRUE,
              ci.level=c(0.025,0.975),
              xrange=c('1959-01-01', '1979-01-01'),
              truth=TRUE)
```

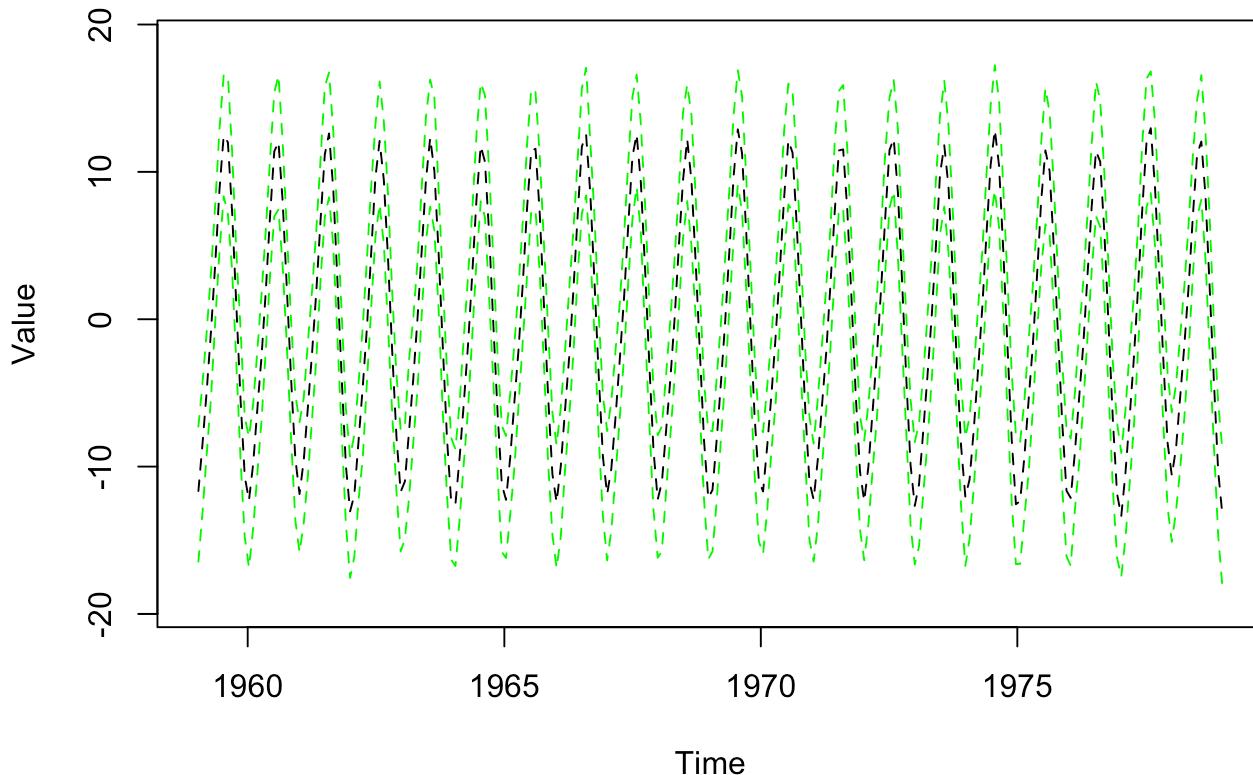
Location 1



Plotting predictions at an unobserved location:

```
loc = data.frame('Longitude' = 111.41, 'Latitude' = 38.29) # Torrey, Utah
plot.location(bstfa.output,
              location=loc,
              type='mean',
              pred.int=TRUE,
              uncertainty=TRUE,
              ci.level=c(0.025,0.975),
              xrange=c('1959-01-01', '1979-01-01'),
              truth=FALSE)
```

Longitude 111.41 Latitude 38.29



1.3 Plotting/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. Table 1.2 below displays a table with each plotting function and a basic description.

Table 1.2

Table 1.2: Plotting/visualization functions in the `BSTFA` package.

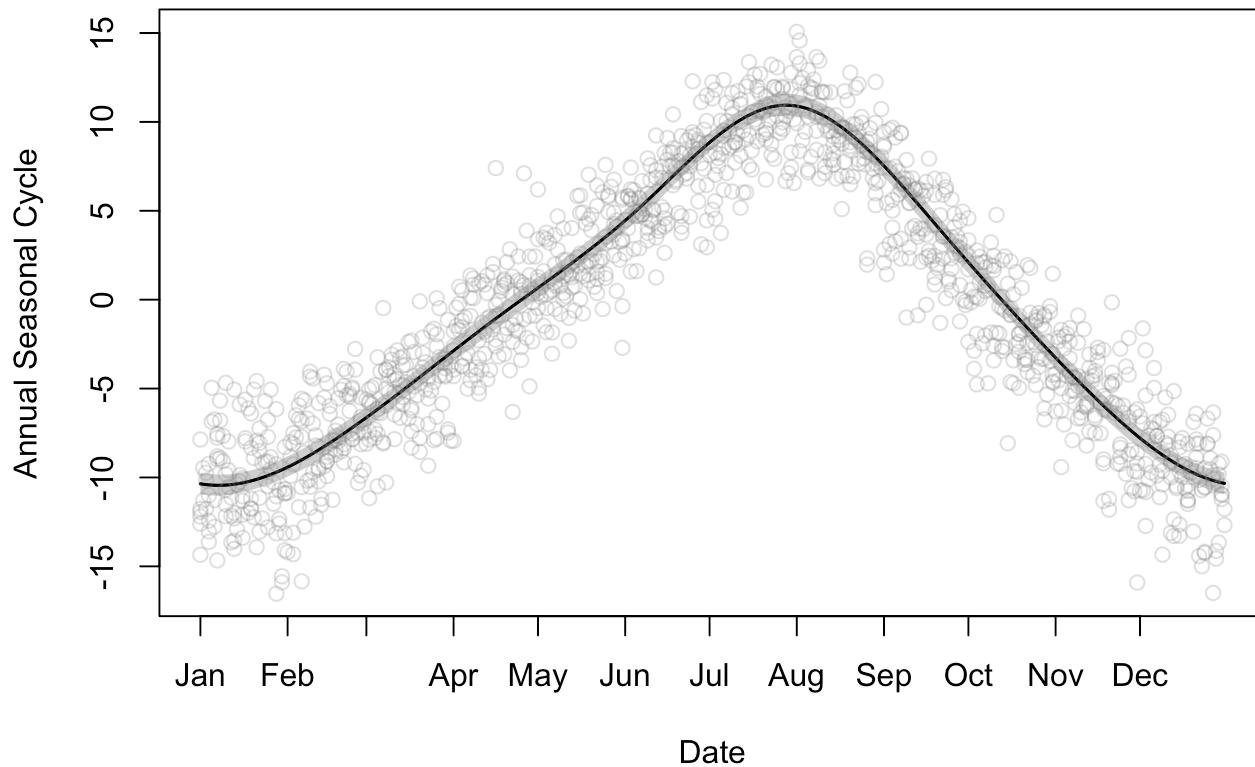
Function	Description
<code>plot.location</code>	Plot predicted response variable at a specific location (either observed or unobserved) for a specified time range. Credible or prediction interval bands for a given probability (default is 95%) can be included. Uses base R for plotting.
<code>plot.annual</code>	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.

Function**Description**

plot.grid	Plot the estimated spatially-dependent mean, 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.
plot.map	Plot the estimated spatially-dependent mean, linear slope, or specific factor loading (the user specifies the parameter of interest) at 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 trend at any (observed or unobserved) location:

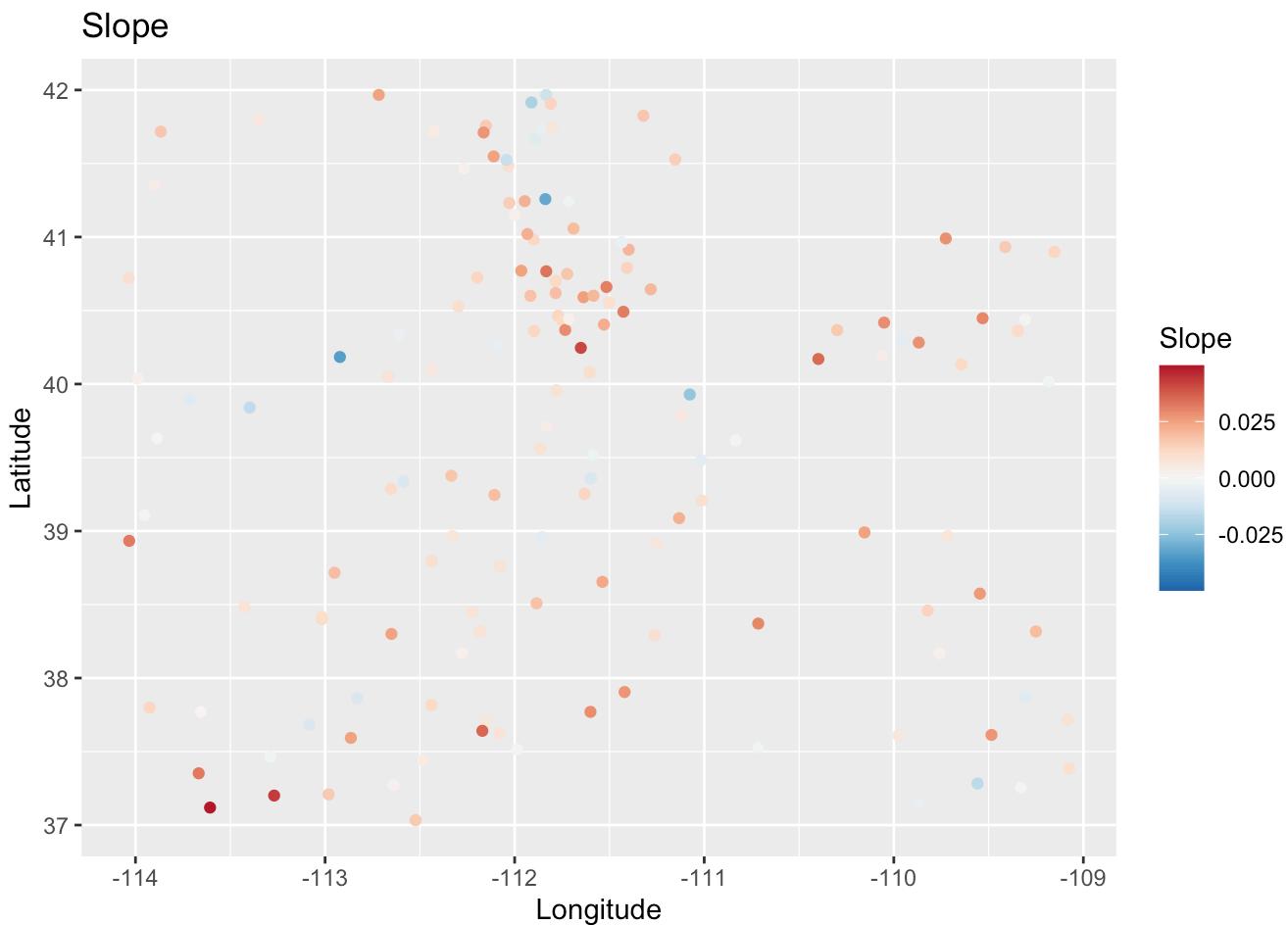
```
plot.annual(bstfa.output,
            location=1,
            years='one')
```



This illustrates the average seasonal behavior in a given year, where the black line is the average and the gray band is the 95% credible interval for the average, 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.

The `plot.grid` function can plot the estimated linear slope, long-term mean 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", "mean" "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 (demonstrated below). First, we can plot the linear slope:

```
plot.grid(bstfa.output,
          type='mean',
          parameter='slope',
          yearscale=TRUE)
```

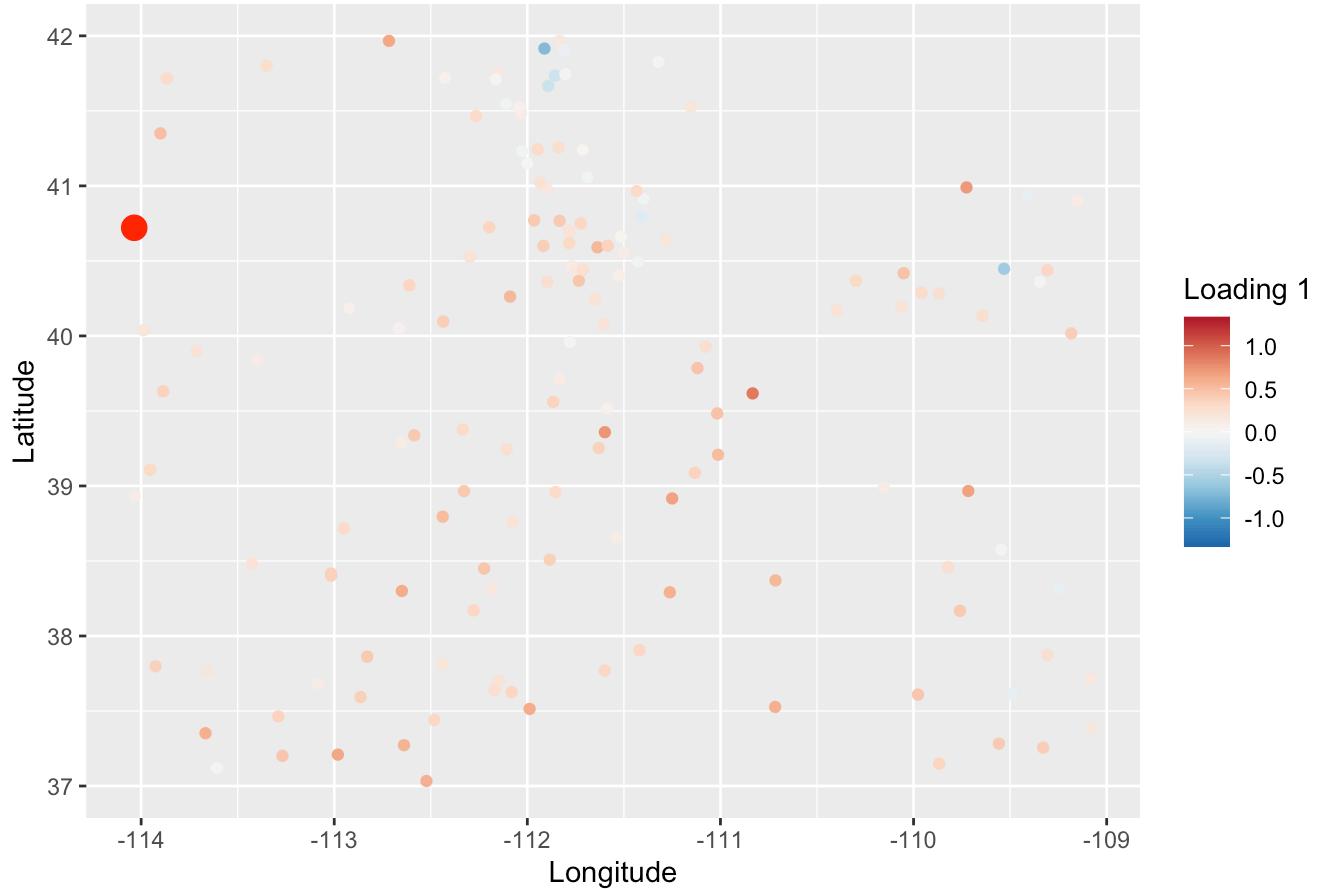


The color of this plot show the long-term increasing (positive and red) or decreasing (negative and blue) behavior over the observed time period for the observed locations. Notice that most locations show an average increase in temperature of between 0 and 0.025 degrees Celcius per year.

Next, we can plot a particular loading:

```
plot.grid(bstfa.output,
          type='mean',
          parameter='loading',
          loadings=1)
```

Loading 1, Fixed Location 144

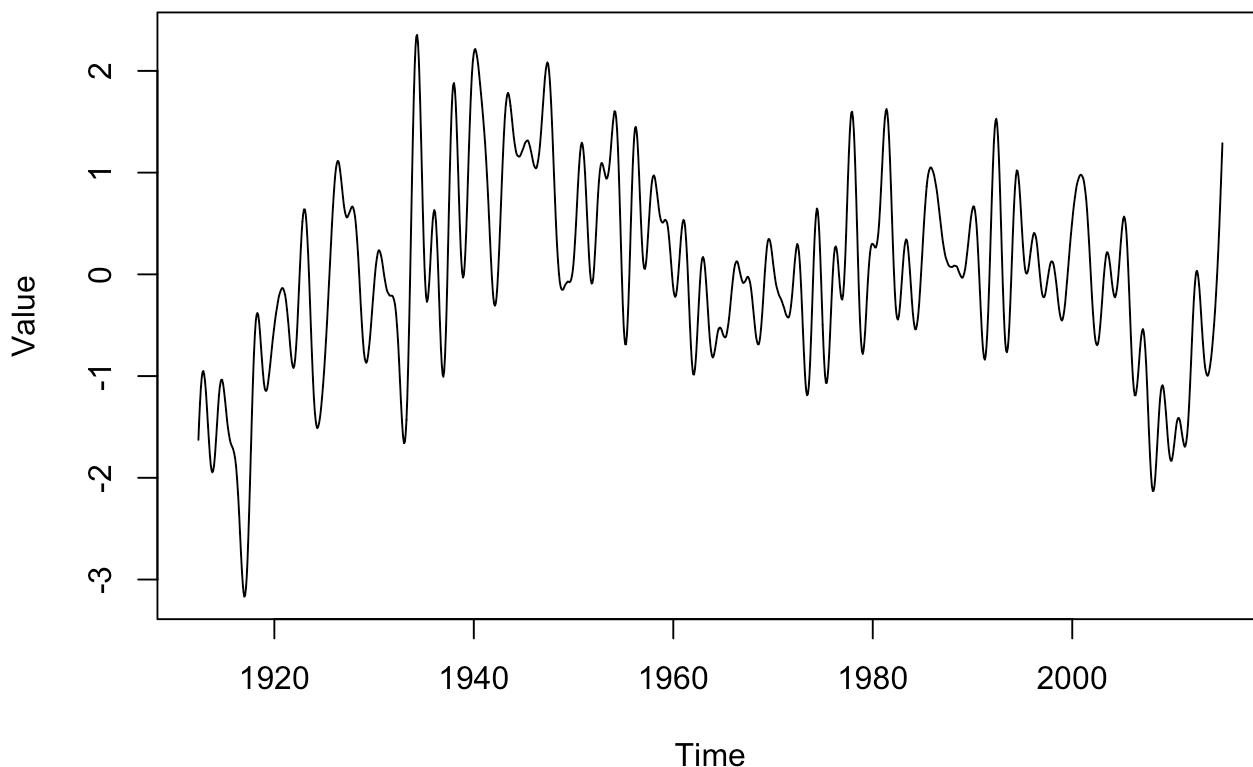


This plot shows the weights that each location places on the first factor, with the color indicating the strength of the weight (darker colors indicate a stronger weight).

The `plot.factor` function plots the estimated temporal factors either together (setting `together == TRUE`) or separate (setting `together == FALSE` and `factor` to whichever factor you want to plot). To match the plot above, we can plot factor 1:

```
plot.factor(bstfa.output,
            together=FALSE,
            include.legend=FALSE,
            factor=1,
            type='mean')
```

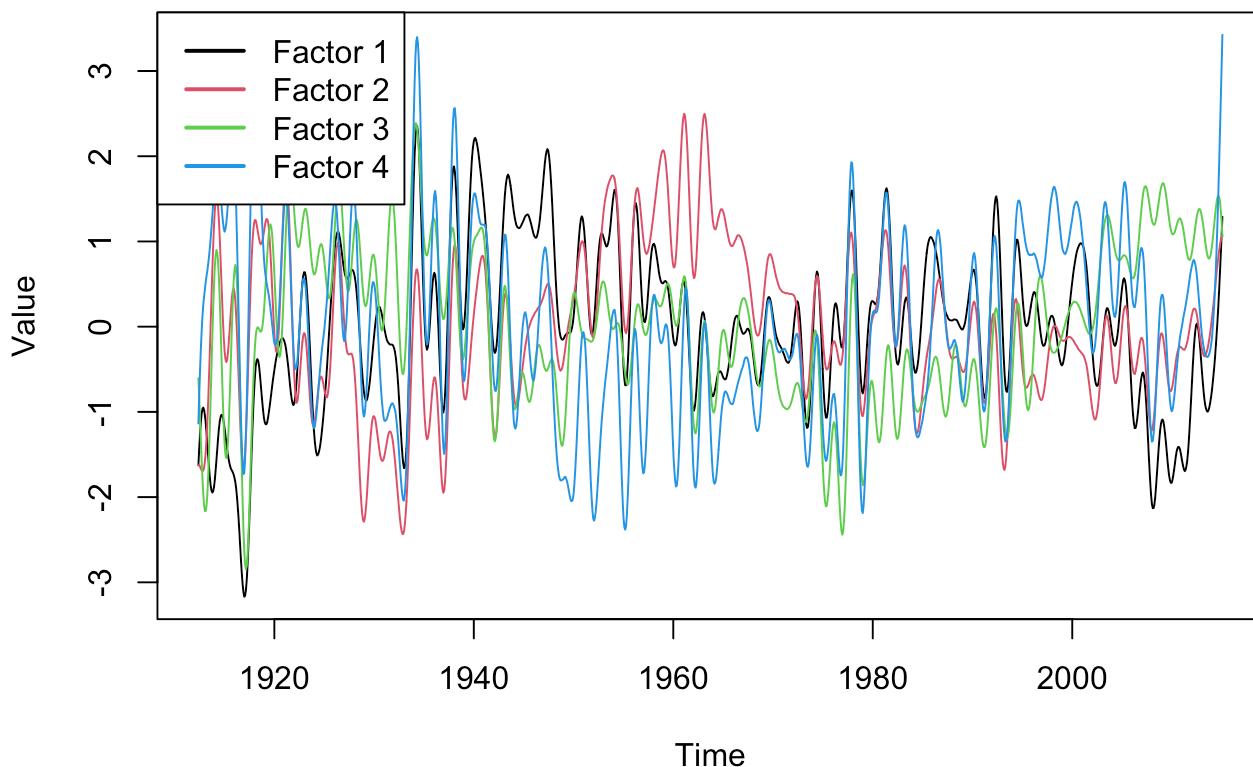
Factor 1



To plot the factors together, set `together==TRUE` :

```
plot.factor(bstfa.output,
            together=TRUE,
            include.legend=TRUE,
            type='mean')
```

All Factors



This figure shows the estimated factors, or common behaviors of the temperature measurements across time that weren't explicitly modeled (in contrast to the linear increasing/decreasing and seasonal behaviors that were explicitly modeled). Notice that factor 1 is higher in the 1940s than the other factors, indicating that locations who load heavily on this first factor tend to have higher temperatures in the 1940s than the other locations.

The `plot.map` is similar to `plot.grid`, but the parameter is plotted at a grid of unobserved values. If `map` is set to `FALSE`, the estimates will appear on a square grid. Setting, `map==TRUE`, `state==TRUE` and `location=='utah'` utilizes the `sf` package to import a map of Utah. The `fine` argument indicates the size of the grid; for instance, setting `fine=100` creates a 100×100 grid of estimates. 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).

```
plot.map(bstfa.output,
         parameter='slope',
         yearscale=TRUE,
         type='mean',
         map=TRUE,
         state=TRUE,
         location='utah',
         fine=100)
#> Coordinate system already present. Adding new coordinate system, which will
#> replace the existing one.
```

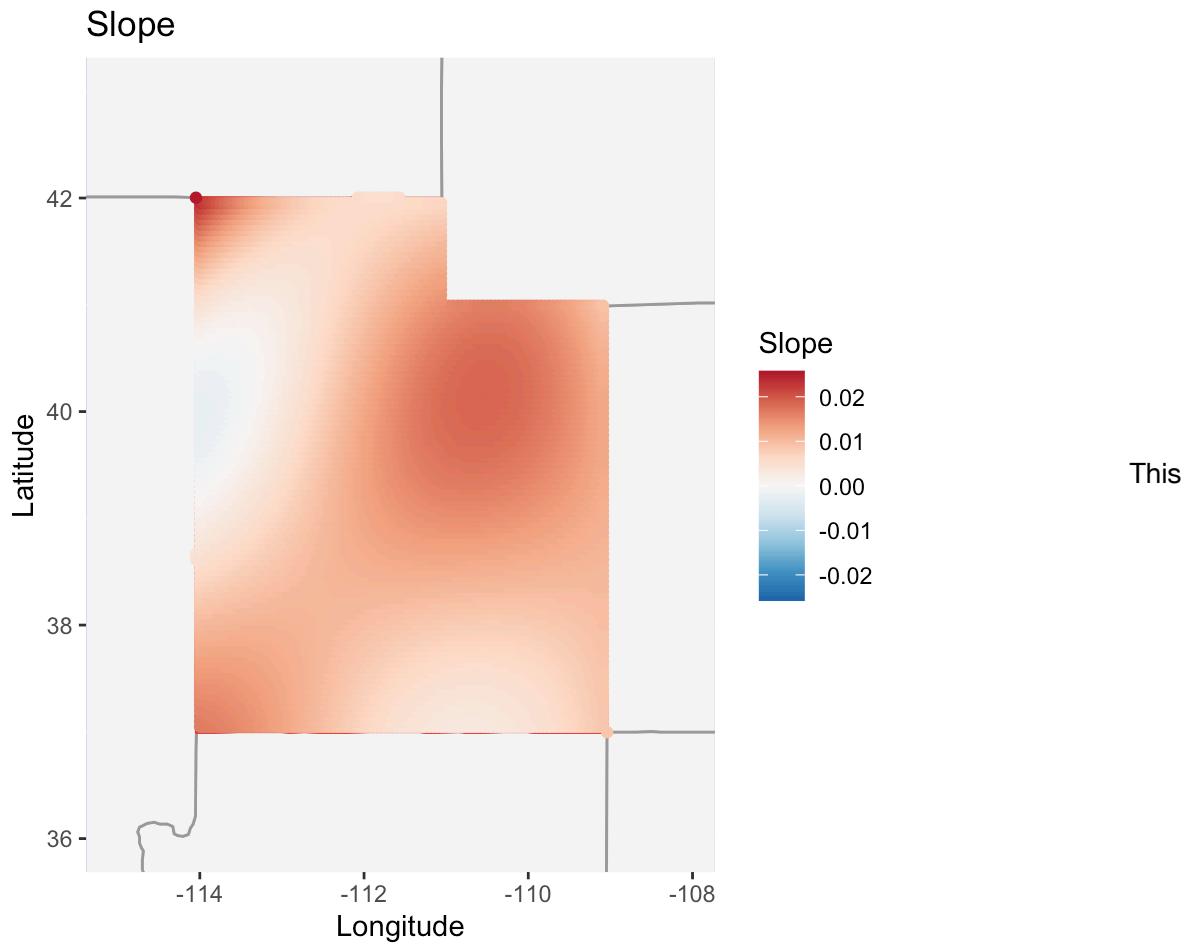


figure shows the temperature tends to be increasing across the entire state, more so in the northeast and southwest corners.

1.4 Speed

As mentioned before, the BSTFA package takes advantage of various mathematical and coding shortcuts to speed up computation. Specifically, we used 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 “full” model for various number of bases.

The model details are covered in greater detail in Section 2, so here we supply a short description as to why `BSTFA` is much faster than `BSTFAfull`. Each function models the factor analysis portion differently. In `BSTFAfull`, we model the factors using a vector autoregressive approach and the factor loadings with an exponential spatial dependence structure. 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 a Bayesian sampler 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 the order $n \times n$) in every iteration of the sampler.
- Estimating certain parameters in this framework requires the Metropolis-Hastings algorithm which introduces increased computation time and potential inefficiency (e.g., smaller effective posterior sample sizes).

The `BSTFA` function instead fits both the factors and the factor loadings using basis functions of various forms, as discussed in Section 2. This solves the problems mentioned above by:

- Removing the need for a vector autoregressive loop.
- Lowering the dimension of the previously-large matrices.
- 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 1.3 below 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 number of bases (indicated by the different columns) for the factor loadings. In each instance, $T = 300$ and the number of temporal bases is $R_t = 60$.

Table 1.3

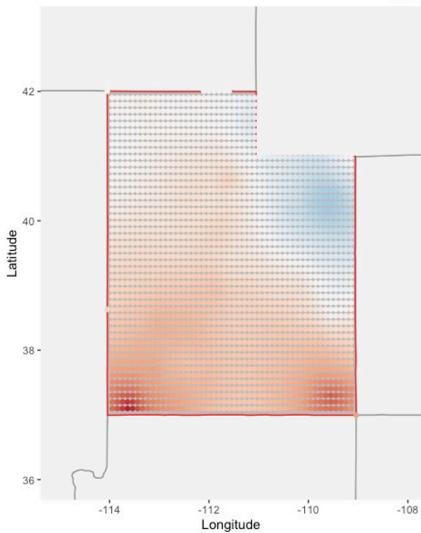
Table 1.3: Computation time in seconds per MCMC iteration for the `BSTFA` and `BSTFAfull` functions.

n	<code>BSTFA , 8 loading bases</code>	<code>BSTFA , 20 loading bases</code>	<code>BSTFA , 50 loading bases</code>	<code>BSTFAfull</code>
100	0.016	0.017	0.021	0.481
200	0.029	0.031	0.036	1.292
300	0.051	0.05	0.056	1.693
400	0.082	0.078	0.086	2.7
500	0.12	0.119	0.126	4.186

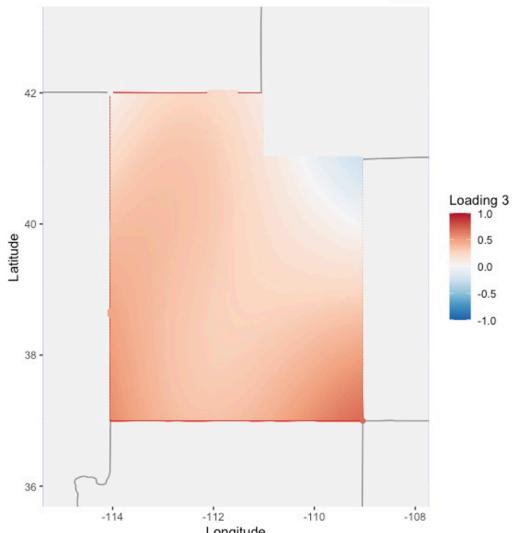
Note: The computations were carried out on a MacBook Pro (13-inch, 2022) with an Apple M2 chip (8-core CPU, 3.5 GHz) and 8 GB of RAM, running macOS 15.1.1.

The plots below illustrate that there is a trade-off between computation speed and smoothness; that is, computation time is dramatically 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 time. The first set of plots compares the grid of estimates for the third loading from both the `BSTFAfull` and `BSTFA` functions. The second set of plots compares the factor estimates.

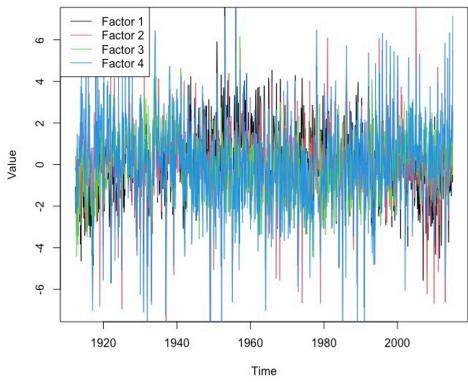
Loading 3, BSTFAfull



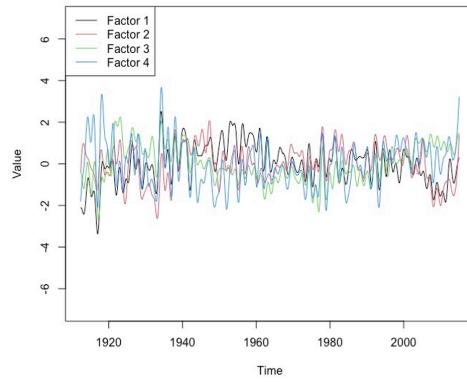
Loading 3, BSTFA



Factors, BSTFAfull



Factors, BSTFA



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

```
computation.summary(bstfa.output)
#> [1] "Setup Time: 0.431 seconds."
#> [1] "PARAMETERS"
#> [1] "Beta: 0.003 seconds per iter."
#> [1] "Xi: 0.005 seconds per iter."
#> [1] "F: 0.058 seconds per iter."
#> [1] "Lambda: 0.011 seconds per iter."
#> [1] "Sigma2: 0.00197 seconds per iter."
#> [1] "OVERALL PER ITERATION"
#> [1] "Pre-Factor Analysis: 0.012 seconds."
#> [1] "Post-Factor Analysis: 0.079 seconds."
#> [1] "TOTAL TIME"
#> [1] "Total time: 79.363 seconds (1.323 minutes) for 1000 iterations."
```

2 Methodology

This section details the methodology implemented in the `BSTFA` package. Specifically, the processes included in the model, basis function dimension reduction techniques, and residual spatio-temporal factor analysis are all discussed. For further details, the reader will be directed to appropriate references, including the original paper to implement the model contained in the `BSTFAfull` function.

2.1 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 $s \in D$ and a time index $t \in T$, let $Y(s, t)$ be the mean-centered response variable such that

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

where $t - \bar{t}$ represents the time t centered by average time over the period of interest, $\beta(s)$ represents a spatially-dependent linear slope in time, $g(\xi(s), t)$ represents a spatially-dependent seasonal periodic process, $\mathbf{f}'(t)\lambda(s)$ is a spatio-temporal confirmatory factor analysis (CFA) process, and $\epsilon(s, t)$ is a zero-mean independent Gaussian residual process with variance σ^2 .

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 residual factor analysis component $\mathbf{f}'(t)\lambda(s)$ 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).

2.1.1 Linear Component

The linear constant changes across time, $\beta(s)$, are allowed to vary spatially by using spatial basis functions. Let $\beta = (\beta(s_1), \dots, \beta(s_n))'$ be the $n \times 1$ vector of coefficients for the locations of interest. We model

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

where \mathbf{B}_β is an $n \times b_\beta$ matrix of basis functions (Berrett et al., 2020), α_β 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 α_β ,

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

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

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

where ϕ is the rate parameter of the Gamma distribution. Table 2.1 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the linear component.

Table 2.1

Table 2.1: 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 β 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 A.
tau2.gamma	2	Value of the shape parameter, γ , for the prior of the variance, τ^2 , associated with β .
tau2.phi	1e-6	Value of the rate parameter, ϕ , for the prior of the variance, τ^2 , associated with β .

2.1.2 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

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

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 corresponding $u \times 1$ vector of coefficients $\xi(s)$. We then model the coefficients using the same approach used for the linear slopes. Namely, let $\xi_j = (\xi_j(s_1), \dots, \xi_j(s_n))'$ represent the coefficients for the j th spline for all locations of interest. Then,

$$\xi_j \sim N(\mathbf{B}_{\xi_j} \alpha_{\xi_j}, \tau_{\xi_j}^2 \mathbf{I}),$$

where \mathbf{B}_{ξ_j} is an $n \times b_{\xi_j}$ matrix of basis functions, α_{ξ_j} represents the corresponding $b_{\xi_j} \times 1$ vector of coefficients, $\tau_{\xi_j}^2$ represents the variances, and \mathbf{I} the appropriate identity matrix. α_{ξ_j} and $\tau_{\xi_j}^2$ are modeled with the same prior distributions (and same hyperparameters) as α_β and τ_β^2 . Table 2.2 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the seasonal component.

Table 2.2

Table 2.2: Arguments to `BSTFA` and `BSTFAfull` associated with the seasonal component.

Argument	Default Value	Description
seasonal	TRUE	TRUE/FALSE value indicating whether the seasonal component is included in the model.
xi	NULL	Vector of starting values for ξ 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).

2.1.3 Factor Analysis Component

Berrett et al. (2020) used a vector autoregressive model on the factors and an exponential Gaussian process model on the loadings, a method implemented in the `BSTFAfull` function. This is quite computationally expensive. The `BSTFA` function focuses on reducing this computational burden and thus we refer the reader to Berrett et al. (2020) for a full discussion of their approach for the spatio-temporal factor analysis process. The computationally-efficient approach for fitting the spatio-temporal factor analysis will be discussed in Section 2.3.

If L represents the number of factors, we point out that $\mathbf{f}(t)$ is an $L \times 1$ vector of factors (a.k.a. scores) at time t and $\lambda(\mathbf{s})$ 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 $\Lambda = [\lambda(\mathbf{s}_1) \cdots \lambda(\mathbf{s}_n)]$ 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 for the loadings for L locations with an L -rank matrix of constants (Rencher & Christensen, 2012). Table 2.3 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with the factor analysis component.

Table 2.3

Table 2.3: Arguments to `BSTFA` and `BSTFAfull` associated with the factor analysis component.

Argument	Default Value	Description
<code>factors</code>	TRUE	TRUE/FALSE value indicating whether the factor analysis component is included in the model.
<code>Fmat</code>	NULL	Matrix of starting values for F of dimension $T \times L$; if none is supplied, realistic starting values are calculated.
<code>Lambda</code>	NULL	Matrix of starting values for Λ of dimension $n \times L$; if none is supplied, realistic starting values are calculated.
<code>factors.fixed</code>	NULL	Vector of indices (representing specific columns of <code>ymat</code>) 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, <code>n.factors = length(factors.fixed)</code> .
<code>n.factors</code>	<code>min(4, ceiling(ncol(ymat)/20))</code>	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 <code>factors.fixed</code> argument.
<code>plot.factors</code>	FALSE	TRUE/FALSE value indicating whether to provide a base R plot of the fixed factor locations.

2.2 Basis Functions

Basis functions act as 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: Fourier bases, bisquare bases, and thin-plate spline bases. For temporal modeling, only Fourier bases are implemented. Because the Fourier basis function is the default for the BSTFA package, that is the only methodology discussed in detail here; for the others, we refer the reader to Cressie & Johannesson (2008) for bisquare bases and Nychka (2000) for thin-plate splines.

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 $QT_r(t)$ and $QT_{r+1}(t)$ be the r^{th} and $(r + 1)^{\text{th}}$ columns of the matrix of bases for time t for $r = 1, 3, 5, \dots R_t$. Then,

$$\begin{aligned} QT_r(t) &= \sin\left(2\pi \frac{r+1}{2} \frac{t}{f_t}\right) \\ QT_{r+1}(t) &= \cos\left(2\pi \frac{r+1}{2} \frac{t}{f_t}\right), \end{aligned}$$

where f_t is the frequency of the Fourier function.

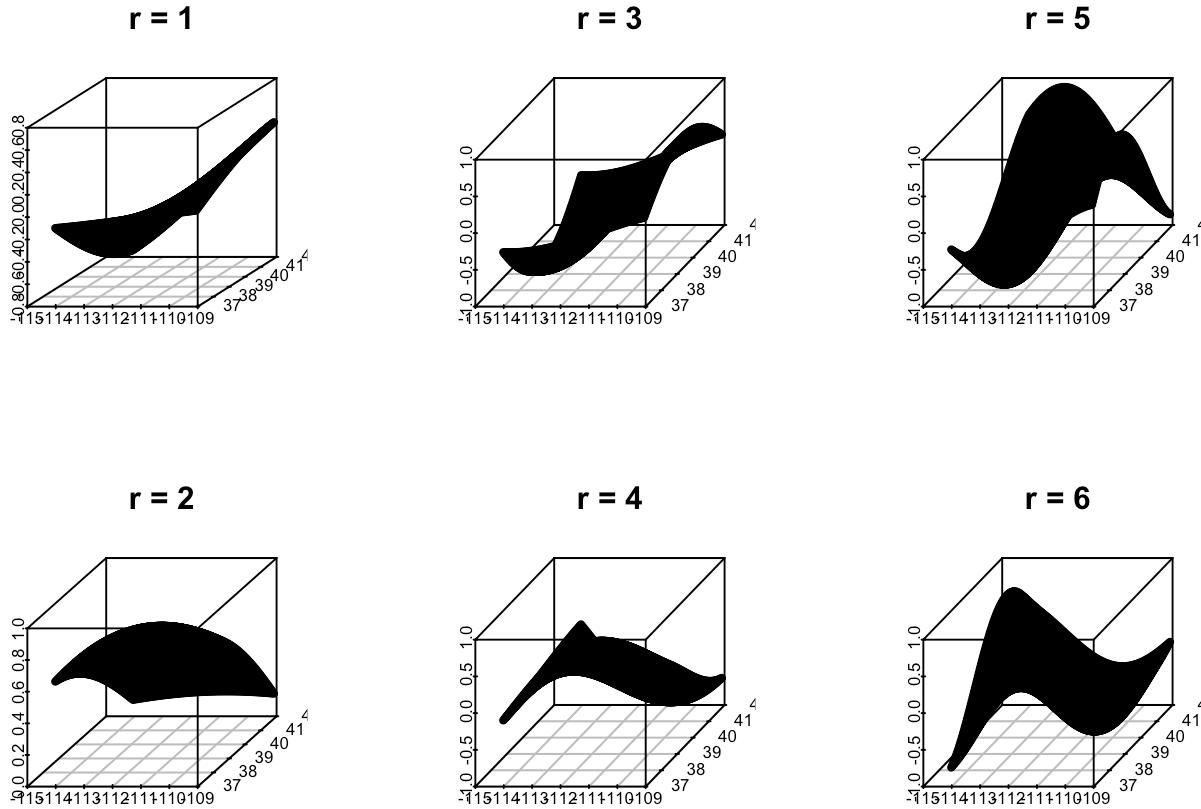
For the spatial basis functions, we must accommodate the two-dimensional nature of space. Thus, we multiply the sine and cosine functions for each dimension (Paciorek, 2007). Let $QS_r(\mathbf{s})$ and $QS_{r+1}(\mathbf{s})$ be the r^{th} and $(r + 1)^{\text{th}}$ columns of the matrix of bases evaluated at location \mathbf{s} for $r = 1, 3, 5, \dots R_s$. Then,

$$\begin{aligned} QS_r(\mathbf{s}) &= \sin\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[1]}}{f_{\mathbf{s}_{[1]}}}\right) * \sin\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[2]}}{f_{\mathbf{s}_{[2]}}}\right) \\ QS_{r+1}(\mathbf{s}) &= \cos\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[1]}}{f_{\mathbf{s}_{[1]}}}\right) * \cos\left(2\pi \frac{r+1}{2} \frac{\mathbf{s}_{[2]}}{f_{\mathbf{s}_{[2]}}}\right) \end{aligned}$$

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 Fourier bases over a given set of coordinates. This can be useful when trying to decide frequencies (freq.lon and freq.lat) and number of bases (R) to fit in your model. This function is demonstrated below using the Utah temperature data.

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



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

$$\begin{aligned}\tilde{\mathbf{F}} &= (\mathbf{I}_L \otimes \mathbf{QT})\alpha_F \\ \tilde{\Lambda} &\sim N((\mathbf{QS} \otimes \mathbf{I}_L)\alpha_\Lambda, \tau_\Lambda^2 \mathbf{I}_{Ln})\end{aligned}$$

where \mathbf{QT} is a $T \times (R_t + 1)$ matrix of temporal bases, \mathbf{QS} is a $n \times (R_s + 1)$ 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, τ_Λ^2 is the error variance for the loadings, and \mathbf{I}_{Ln} is the $Ln \times Ln$ identity matrix.

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

$$\begin{aligned}\alpha_F &\sim N(\mathbf{0}, \mathbf{A}), \\ \alpha_\Lambda &\sim N(\mathbf{0}, \mathbf{A}),\end{aligned}$$

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, these hyperparameters take the same argument values as used in the linear and seasonal components. Table 2.3 provides a list of the arguments to the `BSTFA` and `BSTFAfull` functions that are associated with basis functions

Table 2.4

Table 2.4: Arguments to `BSTFA` and `BSTFAfull` associated with basis functions for all components of the model.

Argument	Default Value	Description
<code>spatial.style</code>	"Fourier"	Indicates which type of basis functions to use for the linear and seasonal components. The default is "Fourier". Other values accepted are "bisquare" and "tps".
<code>n.spatial.bases</code>	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, $\text{floor}(\sqrt{n.spatial.bases})^2$ is calculated to create an even grid.
<code>load.style</code>	"Fourier"	Indicates which type of basis functions to use for the factor loading component. The default is "Fourier". Other values accepted are "bisquare" and "tps".
<code>n.load.bases</code>	6	Number of basis functions to use for the factor loading component. For Fourier bases, this value is R_s . For bisquare bases, this value is ignored. For thin-plate spline bases, $\text{floor}(\sqrt{n.spatial.bases})^2$ is calculated to create an even grid.
<code>freq.lon</code>	<code>diff(range(coords[,1]))^2</code>	Frequency of the Fourier function for longitude (or, if using other coordinate system, the first coordinate value). This value is $f_s[1]$. Default value is the range of the longitude coordinates squared. If not using Fourier bases, this argument is not used.
<code>freq.lat</code>	<code>diff(range(coords[,2]))^2</code>	Frequency of the Fourier function for latitude (or, if using other coordinate system, the second coordinate value). This value is $f_s[2]$. Default value is the range of the latitude coordinates squared. If not using Fourier bases, this argument is not used.
<code>n.temp.bases</code>	<code>floor(nrow(ymat))/10</code>	Number of Fourier basis functions for the temporal factor component. This value is R_t . The default value is $\text{floor}(T/10)$.
<code>freq.temp</code>	<code>nrow(ymat)</code>	Frequency of the Fourier function for the temporal factor component. This value is f_t . Default value is T .
<code>knot.levels</code>	2	The number of knot resolutions when using the bisquare basis function method. If not using the bisquare method, this argument is not used.
<code>max.knot.dist</code>	<code>mean(dist(coords))</code>	The distance beyond which a location is considered 'too far' from a knot, meaning its basis function associated with that knot evaluates to zero. If not using the bisquare method, this argument is not used.

Argument	Default Value	Description
premade.knots	NULL	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 dimension as <code>coords</code> . If not using the bisquare method, 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 <code>coords</code> . If not using the bisquare method, this argument is not used.

3 Useful Features

3.1 Fixing factors

Factor analysis can lend itself to 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 (St. George), and North (Logan) factors, shown in Figure 3.1. 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. Once again, if fixed factor locations are not given, they will be smartly chosen by the function according to distance and proportion of missing data.

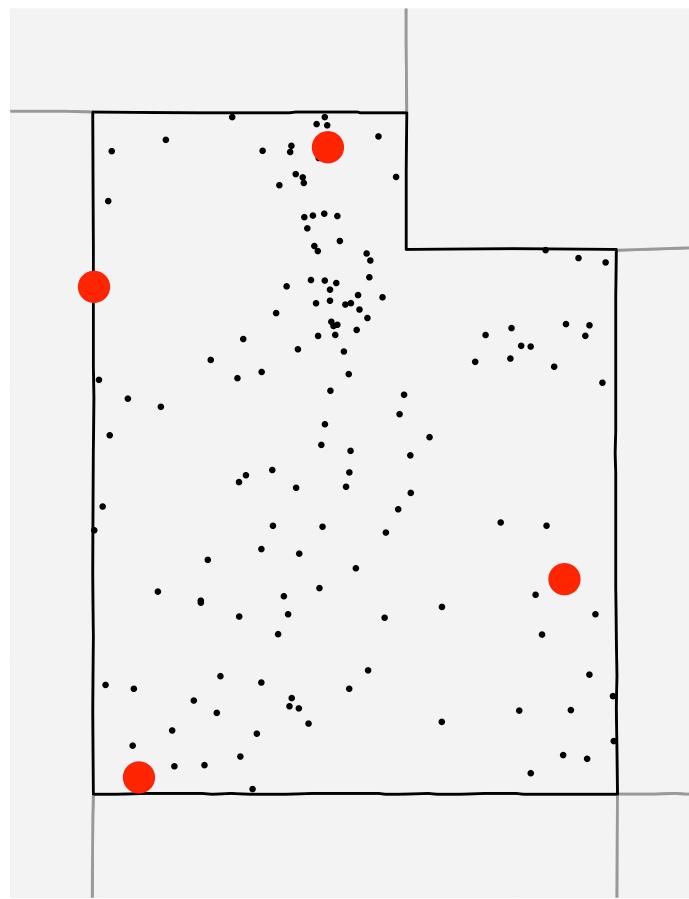


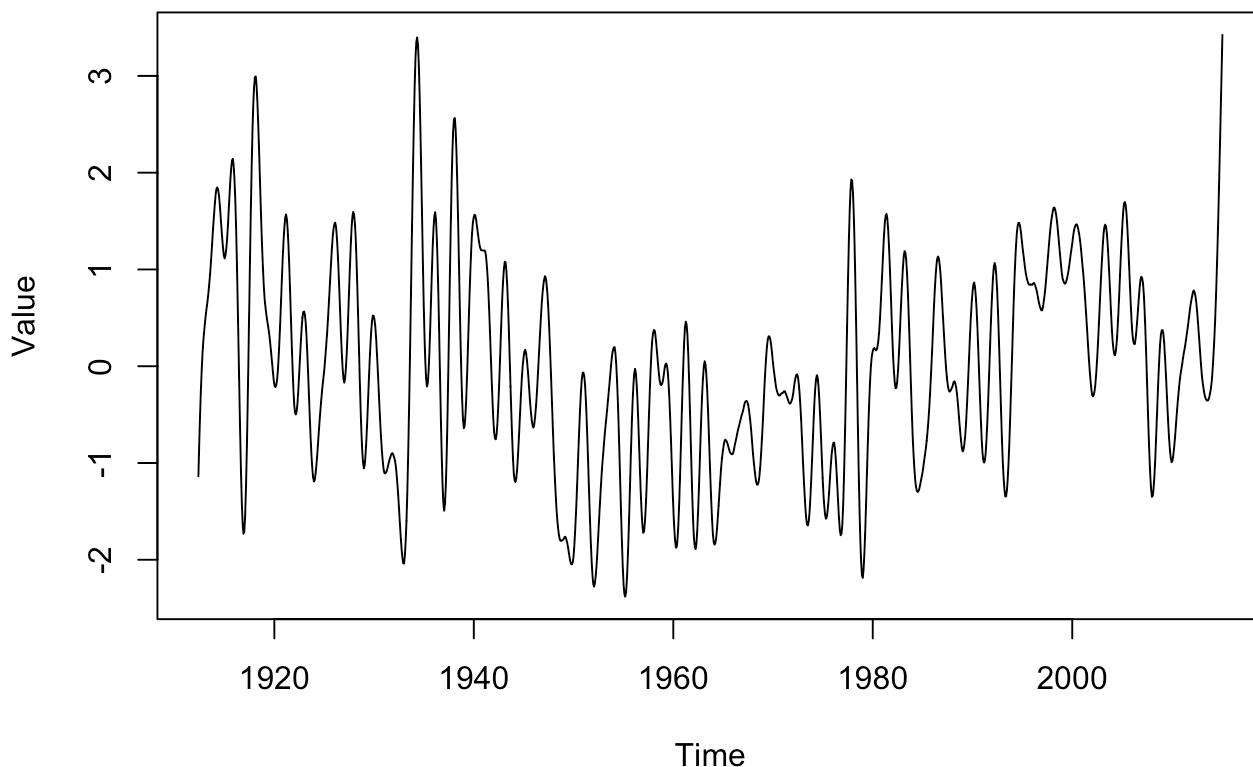
Figure 3.1: Plot of Utah showing locations of fixed factors (in red) and all locations (in black).

Using these fixed factors and $L \times L$ identity matrix, the estimate for the fourth loading at a location indicates the strength of the relationship of the behavior to the behavior at Logan, Utah, after accounting for linear and seasonal processes. A higher estimate of the loading suggests greater similarity.

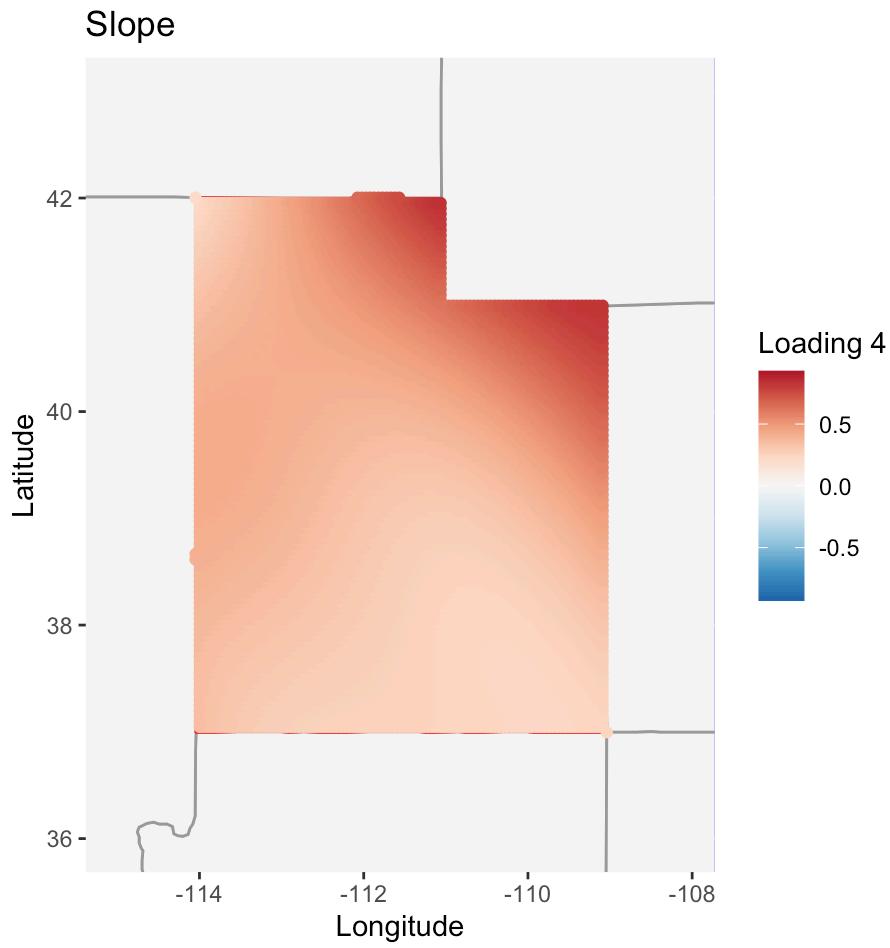
To plot the fourth factor and its loading:

```
plot.factor(bstfa.output,
            together=FALSE,
            include.legend=FALSE,
            factor=4,
            type='mean' )
```

Factor 4



```
plot.map(bstfa.output,
         parameter='loading',
         loading=4,
         yearscale=TRUE,
         type='mean',
         map=TRUE,
         state=TRUE,
         location='utah',
         fine=100)
#> Coordinate system already present. Adding new coordinate system, which will
#> replace the existing one.
```



Here we see that locations in the North/Northeast have behavior similar to that of the fourth factor (Logan, Utah).

3.2 Choosing Basis Functions

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}_β and each \mathbf{B}_{ξ_j}) while the `load.style` argument controls the basis functions for the factor loadings (\mathbf{QS}). 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 is 'fourier'. The only basis functions used for the factors themselves are Fourier bases.

3.2.1 Fourier

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 2.2, the function `plot.fourier.bases` can help the user visualize Fourier bases and choose the appropriate amount of bases and frequencies. After exploratory analysis, 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 works well for the Utah dataset.

FIX ME - what exploratory analysis?

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 tend to work well (see Table 2.4), but increasing the number of bases can create a finer estimate at the cost of reduced computational speed.

Note: 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.

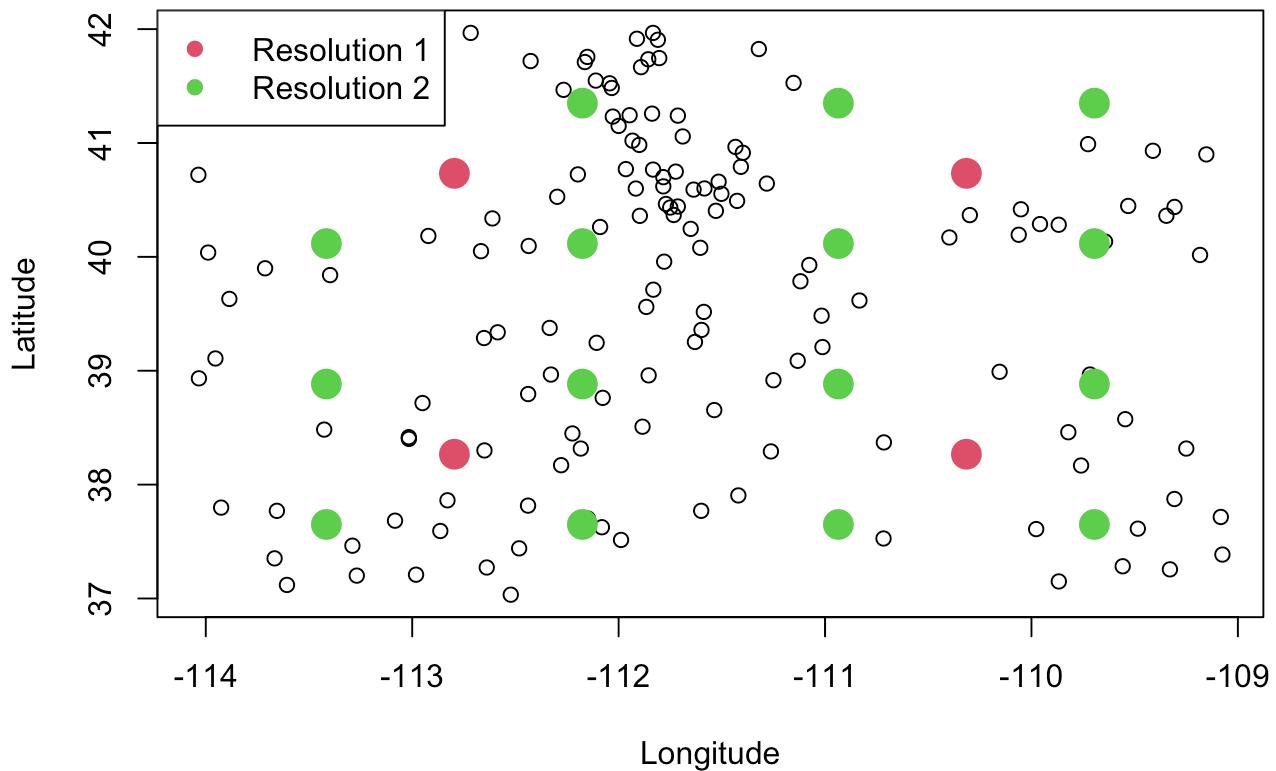
```
BSTFA(ymat=utahDataList$TemperatureVals,
       dates=utahDataList$Dates,
       coords=utahDataList$Coords,
       spatial.style='fourier',
       load.style='fourier',
       n.spatial.bases=8,
       n.load.bases=6,
       freq.lon=40,
       freq.lat=30,
       n.temp.bases=floor(nrow(utahDataList$TemperatureVals)/10),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

3.2.2 Bisquare

As described in Table 2.4, 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.

```
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=10)
```

Knots and Spatial Locations



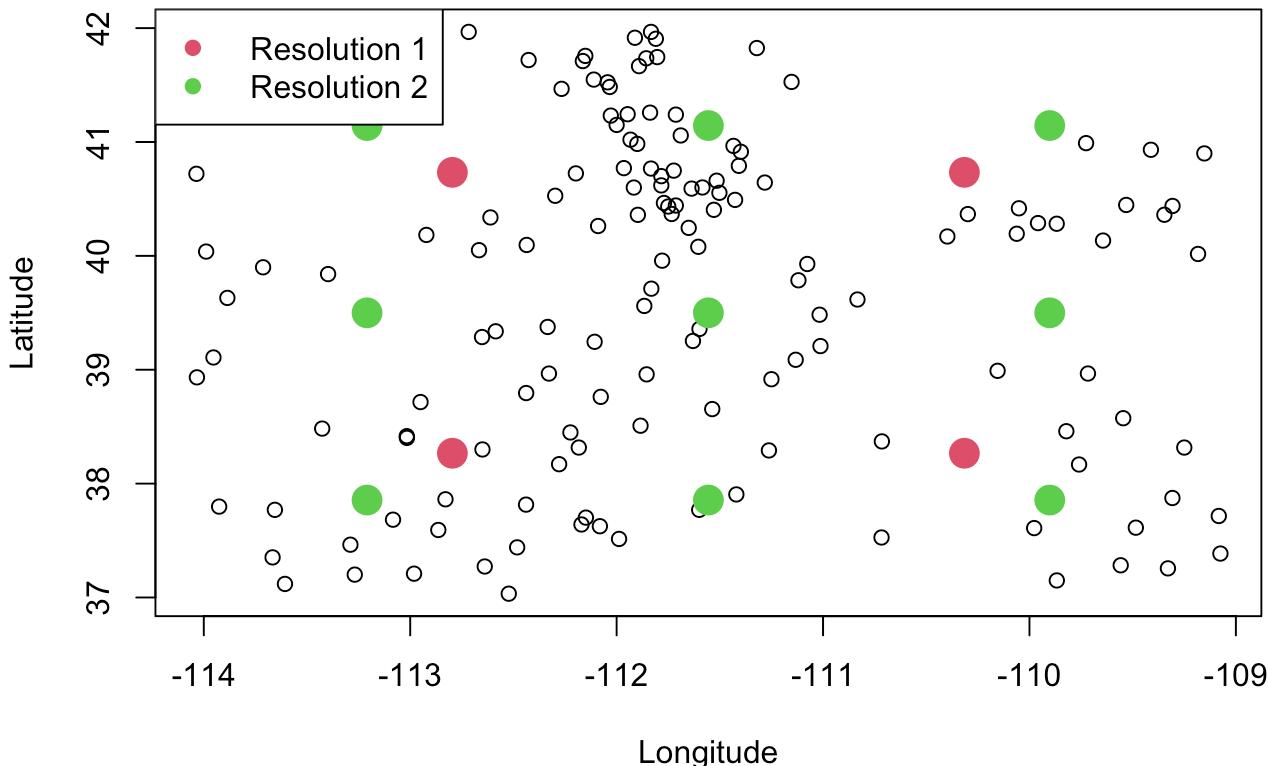
The user can specify custom knot locations with the `premade.knots` argument. This argument takes a list of coordinates containing pre-specified knots. The number of elements in the list represents the number of resolutions. Each element of the list should have the same dimension as `coords`. Below is an example where the first resolution matches the default, but the second uses a 3×3 grid of knots rather than the default 4×4 .

```

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=10)

```

Knots and Spatial Locations



3.2.3 Thin-plate splines

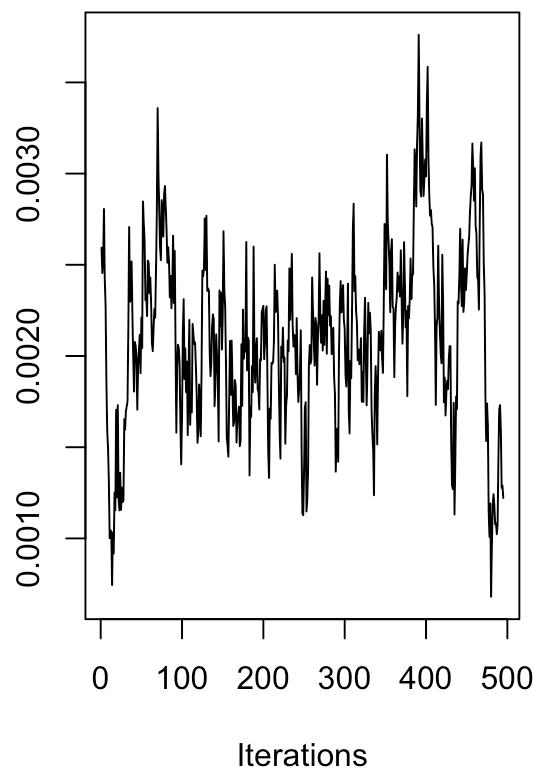
The argument given to `spatial.style` and `load.style` to use these basis functions is 'tps'. The function `basis.tps` from the `npreg` package is used to create the thin-plate spline bases. The knots used to create the bases are on a square grid; thus, the number of bases is equal to `floor(sqrt(n.spatial.bases))^2`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

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

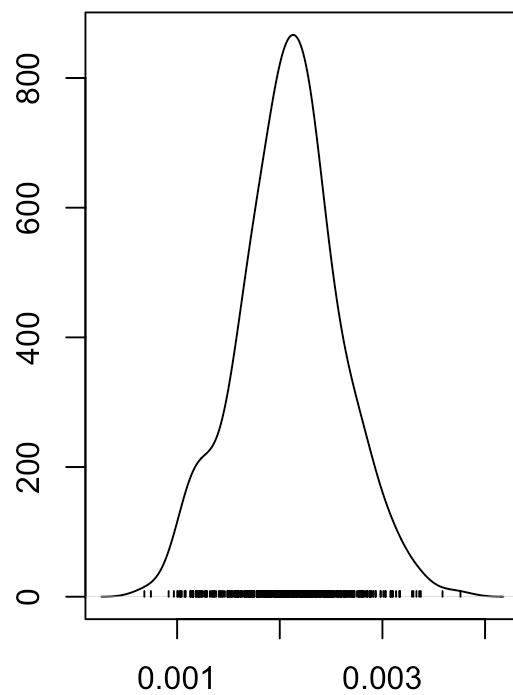
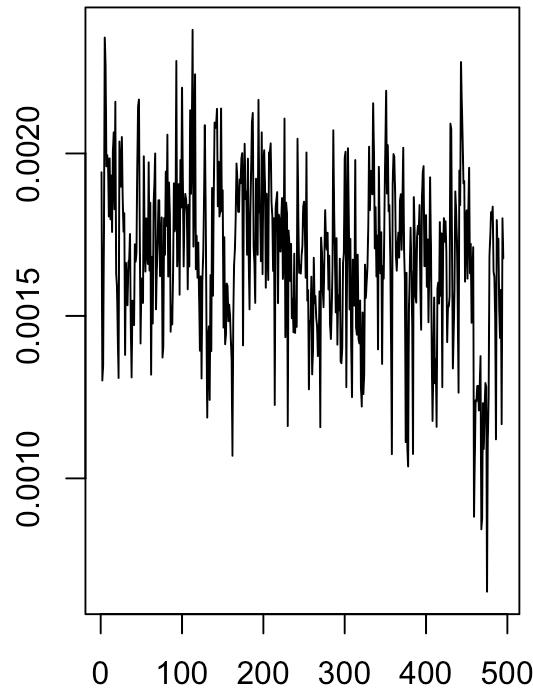
3.3 Assessing MCMC Convergence

The BSTFA package has built in helper functions for assessing convergence. They use the fact that all matrices of parameter draws are MCMC objects from the package `coda`. To look at trace plots, you can use the `plot.trace` function, which 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 indicates which of these parameters you want to view.

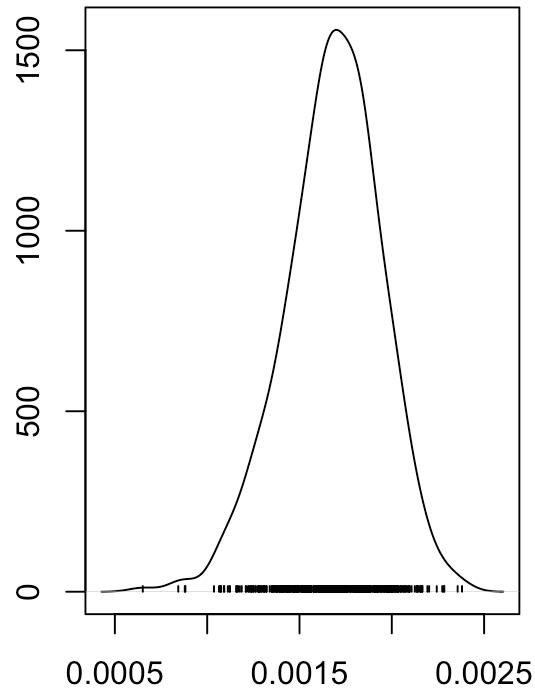
```
plot.trace(bstfa.output,
            parameter='beta',
            param.range=c(2,27))
```

beta 2

Iterations

beta 2 $N = 495$ Bandwidth = 0.0001391**beta 27**

Iterations

beta 27 $N = 495$ Bandwidth = 7.455e-05

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

```
check.convergence(bstfa.output,
                  type='eSS',
                  cutoff = 100)
```

4 Appendices

4.1 Additional Notes

Note: to help with convergence of the residual factor analysis component, the sampler waits to sample \mathbf{F} and Λ until `min(floor(burn/2), 500)`.

References

- Berrett, C., Christensen, W. F., Sain, S. R., et al. (2020). Modeling sea-level processes on the u.s. Atlantic coast. *Environmetrics*, 31(e2609). <https://doi.org/10.1002/env.2609> (<https://doi.org/10.1002/env.2609>)
- Cressie, Noel, & Johannesson, Gardar. (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. *Annu. Rev. Stat. Appl.*, 9. <https://doi.org/10.1146/annurev-statistics-040120-020733> (<https://doi.org/10.1146/annurev-statistics-040120-020733>)
- Nychka, Douglas W. (2000). Spatial-process estimates as smoothers. *Smoothing and Regression: Approaches, Computation, and Application*, 329, 393.
- Paciorek, Christopher J. (2007). Bayesian smoothing with gaussian processes using fourier basis functions in the spectralGP package. *Journal of Statistical Software*.
- Rencher, Alvin C., & Christensen, William. (2012). *Methods of multivariate analysis, third edition*. Wiley.
- Wikle, Christopher. (2002). Spatial modeling of count data: A case study in modelling breeding bird survey data on large spatial domains. In Andrew Lawson & David Denison (Eds.), *Spatial cluster modelling* (pp. 199–209). Chapman & Hall.
- Wood, Simon N. (2017). *Generalized additive models: An introduction with r*. chapman; hall/CRC.