# Package 'SequenceSpikeSlab'

October 12, 2022

**Type** Package

**Title** Exact Bayesian Model Selection Methods for the Sparse Normal Sequence Model

**Version** 1.0.0

**Author** Steven de Rooij [aut],
Tim van Erven [cre, aut],
Botond Szabo [aut]

**Maintainer** Tim van Erven <tim@timvanerven.nl>

**Description** Contains fast functions to calculate the exact Bayes posterior for the Sparse Normal Sequence Model, implementing the algorithms described in Van Erven and Szabo (2021, <doi:10.1214/20-BA1227>). For general hierarchical priors, sample sizes up to 10,000 are feasible within half an hour on a standard laptop. For beta-binomial spike-and-slab priors, a faster algorithm is provided, which can handle sample sizes of 100,000 in half an hour. In the implementation, special care has been taken to assure numerical stability of the methods even for such large sample sizes.

**License** GPL (>= 2)

**Imports** Rcpp (>= 0.12.18), RcppProgress (>= 0.4.1), selectiveInference (>= 1.2.5)

**LinkingTo** Rcpp, RcppProgress

**RoxygenNote** 7.1.2

**Encoding** UTF-8

**Suggests** knitr, rmarkdown

**VignetteBuilder** knitr

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2022-01-23 15:22:44 UTC

# R topics documented:



---

`fast_spike_slab_beta` *Compute marginal posterior estimates for beta-spike-and-slab prior*

---

## Description

Computes marginal posterior probabilities (slab probabilities) that data points have non-zero mean for the spike-and-slab prior with a Beta(beta_kappa,beta_lambda) prior on the mixing parameter. The posterior mean is also provided.

## Usage

```
fast_spike_slab_beta(
  x,
  sigma = 1,
  m = 20,
  slab = "Laplace",
  Laplace_lambda = 0.5,
  Cauchy_gamma = 1,
  beta_kappa = 1,
  beta_lambda,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | Vector of n data points |
| `sigma` | Standard deviation of the Gaussian noise in the data. May also be set to "auto", in which case sigma is estimated using the estimateSigma function from the selectiveInference package |

| | |
|---|---|
| m | The number of discretization points used is proportional to m*sqrt(n). The larger m, the better the approximation, but the runtime also increases linearly with m. The default m=20 usually gives sufficient numerical precision. |
| slab | Slab distribution. Must be either "Laplace" or "Cauchy". |
| Laplace_lambda | Parameter of the Laplace slab |
| Cauchy_gamma | Parameter of the Cauchy slab |
| beta_kappa | Parameter of the beta-distribution |
| beta_lambda | Parameter of the beta-distribution. Default value=n+1 |
| show_progress | Boolean that indicates whether to show a progress bar |

## Details

The run-time is O(m*n^(3/2)) on n data points, which means that doubling the size of the data leads to an increase in computation time by approximately a factor of 2*sqrt(2)=2.8. Data sets of size n=100,000 should be feasible within approximately 30 minutes.

## Value

list (postprobs, postmean, sigma), where postprobs is a vector of marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$; postmean is a vector with the posterior mean for the $x[i]$; and sigma is the value of sigma (this may be of interest when the sigma="auto" option is used)

## Examples

```
# Illustrate that fast_spike_slab_beta is a faster way to compute the same results as
# general_sequence_model on the beta-binomial prior

# Generate data
n <- 500          # sample size
n_signal <- 25    # number of non-zero theta
A <- 5            # signal strength
theta <- c(rep(A,n_signal), rep(0,n-n_signal))
x <- theta + rnorm(n, sd=1)

# Choose slab
slab <- "Cauchy"
Cauchy_gamma <- 1

cat("Running fast_spike_slab_beta (fast for very large n)...\n")
res_fss <- fast_spike_slab_beta(x, sigma=1, slab=slab, Cauchy_gamma=Cauchy_gamma)

cat("Running general_sequence_model (slower for very large n)...\n")
res_gsm <- general_sequence_model(x, sigma=1, slab=slab,
                                  log_prior="beta-binomial", Cauchy_gamma=Cauchy_gamma)

cat("Maximum difference in marginal posterior slab probabilities:",
    max(abs(res_gsm$postprobs - res_fss$postprobs)))
cat("\nMaximum difference in posterior means:",
```

```
    max(abs(res_gsm$postmean - res_fss$postmean)), "\n")

# Plot means
M=max(abs(x))+1
plot(1:n, x, pch=20, ylim=c(-M,M), col='green', xlab="", ylab="",
     main="Posterior Means (Same for Both Methods)")
points(1:n, theta, pch=20, col='blue')
points(1:n, res_gsm$postmean, pch=20, col='black', cex=0.6)
points(1:n, res_fss$postmean, pch=20, col='magenta', cex=0.6)
legend("topright", legend=c("general_sequence_model", "fast_spike_slab_beta",
                            "data", "truth"),
       col=c("black", "magenta", "green", "blue"), pch=20, cex=0.7)
```

---

general_sequence_model

*Compute marginal posterior estimates*

---

## Description

This function computes marginal posterior probabilities (slab probabilities) that data points have non-zero mean for the general hierarchical prior in the sparse normal sequence model. The posterior mean is also provided.

## Usage

```
general_sequence_model(
  x,
  sigma = 1,
  slab = "Laplace",
  log_prior = "beta-binomial",
  Laplace_lambda = 0.5,
  Cauchy_gamma = 1,
  beta_kappa = 1,
  beta_lambda,
  show_progress = TRUE
)
```

## Arguments

| | |
|---|---|
| x | Vector of n data points |
| sigma | Standard deviation of the Gaussian noise in the data. May also be set to "auto", in which case sigma is estimated using the estimateSigma function from the selectiveInference package |
| slab | Slab distribution. Must be either "Laplace" or "Cauchy". |
| log_prior | Vector of length n+1 containing the logarithms of the prior probabilities pi_n(s) that the number of spikes is equal to s for s=0,...,n. It is allowed to use an unnormalized prior that does not sum to 1, because adding any constant to the log-prior |

| | |
|---|---|
| | probabilities does not change the result. Instead of a vector, log_prior may also be set to "beta-binomial" as a short-hand for log_prior = lbeta(beta_kappa+(0:n),beta_lambda+n-(0:n)) - lbeta(beta_kappa,beta_lambda) + lchoose(n,0:n). |
| `Laplace_lambda` | Parameter of the Laplace slab |
| `Cauchy_gamma` | Parameter of the Cauchy slab |
| `beta_kappa` | Parameter of the beta-distribution in the beta-binomial prior |
| `beta_lambda` | Parameter of the beta-distribution in the beta-binomial prior. Default value=n+1 |
| `show_progress` | Boolean that indicates whether to show a progress bar |

## Details

The run-time is O(n^2) on n data points, which means that doubling the size of the data leads to an increase in computation time by approximately a factor of 4. Data sets of size n=25,000 should be feasible within approximately 30 minutes.

## Value

list (postprobs, postmean, sigma), where postprobs is a vector of marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$; postmean is a vector with the posterior mean for the $x[i]$; and sigma is the value of sigma (this may be of interest when the sigma="auto" option is used)

## Examples

```
# Experiments similar to those of Castilo, Van der Vaart, 2012

# Generate data
n <- 500           # sample size
n_signal <- 25     # number of non-zero theta
A <- 5             # signal strength
theta <- c(rep(A,n_signal), rep(0,n-n_signal))
x <- theta + rnorm(n, sd=1)

# Choose slab
slab <- "Laplace"
Laplace_lambda <- 0.5

# Prior 1
kappa1 <- 0.4   # hyperparameter
logprior1 <- c(0,-kappa1*(1:n)*log(n*3/(1:n)))
res1 <- general_sequence_model(x, sigma=1,
                               slab=slab,
                               log_prior=logprior1,
                               Laplace_lambda=Laplace_lambda)
print("Prior 1: Elements with marginal posterior probability >= 0.5:")
print(which(res1$postprobs >= 0.5))

# Prior 2
kappa2 <- 0.8   # hyperparameter
logprior2 <- kappa2*lchoose(2*n-0:n,n)
```

```
res2 <- general_sequence_model(x, sigma=1,
                               slab=slab,
                               log_prior=logprior2,
                               Laplace_lambda=Laplace_lambda)
print("Prior 2: Elements with marginal posterior probability >= 0.5:")
print(which(res2$postprobs >= 0.5))

# Prior 3
beta_kappa <- 1      # hyperparameter
beta_lambda <- n+1   # hyperparameter
res3 <- general_sequence_model(x, sigma=1,
                               slab=slab,
                               log_prior="beta-binomial",
                               Laplace_lambda=Laplace_lambda)
print("Prior 3: Elements with marginal posterior probability >= 0.5:")
print(which(res3$postprobs >= 0.5))

# Plot means for all priors
M=max(abs(x))+1
plot(1:n, x, pch=20, ylim=c(-M,M), col='green', xlab="", ylab="", main="Posterior Means")
points(1:n, theta, pch=20, col='blue')
points(1:n, res1$postmean, pch=20, col='black', cex=0.6)
points(1:n, res2$postmean, pch=20, col='magenta', cex=0.6)
points(1:n, res3$postmean, pch=20, col='red', cex=0.6)
legend("topright", legend=c("posterior mean 1", "posterior mean 2", "posterior mean 3",
                            "data", "truth"),
       col=c("black", "magenta", "red", "green", "blue"), pch=20, cex=0.7)
```

SequenceSpikeSlab *Fast Exact Bayesian Inference for the Sparse Normal Means Model*

## Description

The SequenceSpikeSlab package provides fast algorithms for exact Bayesian inference in the sparse normal sequence model. It implements the methods of Van Erven and Szabo, 2018. Special care has been taken to make the methods scale to large data sets, and to minimize numerical errors (which arise in all software because floating point numbers are represented with finite precision).

## Details

There are two main functions: `general_sequence_model` and `fast_spike_slab_beta`.

For more details see the help vignette: `vignette("SequenceSpikeSlab-vignette", package="SequenceSpikeSlab")`

SSS_discrete_spike_slab

*Compute marginal posterior probabilities (slab probabilities) that data points have non-zero mean for the discretized spike-and-slab prior.*

### Description

Compute marginal posterior probabilities (slab probabilities) that data points have non-zero mean for the discretized spike-and-slab prior.

### Usage

```
SSS_discrete_spike_slab(log_phi_psi, dLambda, show_progress = TRUE)
```

### Arguments

| | |
|---|---|
| log_phi_psi | List {logphi, logpsi} containing two vectors of the same length n that represent a preprocessed version of the data. logphi and logpsi should contain the logs of the phi and psi densities of the data points, as produced for instance by SSS_log_phi_psi_Laplace or SSS_log_phi_psi_Cauchy |
| dLambda | Discretized Lambda prior, as generated by either discretize_Lambda or discretize_Lambda_beta. |
| show_progress | Boolean that indicates whether to show a progress bar |

### Value

Returns a vector with marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$.

SSS_discretize_Lambda

*Given a prior Lambda on the alpha-parameter in the spike-and-slab model, make a discretized version of Lambda that is only supported on a grid of approximately m * sqrt(n) discrete values of alpha. This discretized version of Lambda is required as input for* SSS_discrete_spike_slab*. NB Lambda needs to satisfy a technical condition from the paper that guarantees its density does not vary too rapidly. For Lambda=Beta(kappa,lambda) use* SSS_discretize_Lambda_beta *instead.*

### Description

Given a prior Lambda on the alpha-parameter in the spike-and-slab model, make a discretized version of Lambda that is only supported on a grid of approximately m * sqrt(n) discrete values of alpha. This discretized version of Lambda is required as input for SSS_discrete_spike_slab. NB Lambda needs to satisfy a technical condition from the paper that guarantees its density does not vary too rapidly. For Lambda=Beta(kappa,lambda) use SSS_discretize_Lambda_beta instead.

### Usage

```
SSS_discretize_Lambda(m = 20, n, log_Lambda_cdf)
```

### Arguments

| | |
|---|---|
| `m` | A multiplier for the number of discretization points |
| `n` | The sample size |
| `log_Lambda_cdf` | A function that takes as input a value of alpha and calculates the log of the cumulative distribution function of Lambda at alpha |

### Value

List (alpha_grid, log_probs), where alpha_grid is a vector with the generated grid points, and log_probs are the logs of the prior probabilities of these grid points for the discretized Lambda prior.

---

## `SSS_discretize_Lambda_beta`

*Given prior Lambda=Beta(kappa,lambda) on the alpha-parameter in the spike-and-slab model, make a discretized version of Lambda that is only supported on a grid of approximately m * sqrt(n) discrete values of alpha. This discretized version of Lambda is required as input for SSS_discrete_spike_slab.*

---

### Description

Given prior Lambda=Beta(kappa,lambda) on the alpha-parameter in the spike-and-slab model, make a discretized version of Lambda that is only supported on a grid of approximately m * sqrt(n) discrete values of alpha. This discretized version of Lambda is required as input for SSS_discrete_spike_slab.

### Usage

```
SSS_discretize_Lambda_beta(m = 20, n, kappa, lambda)
```

### Arguments

| | |
|---|---|
| `m` | A multiplier for the number of discretization points |
| `n` | The sample size |
| `kappa` | Parameter of the prior. Needs to be at least 0.5. |
| `lambda` | Parameter of the prior. Needs to be at least 0.5. |

### Value

List (alpha_grid, log_probs), where alpha_grid is a vector with the generated grid points, and log_probs are the logs of the prior probabilities of these grid points for the discretized Lambda prior.

---

`SSS_hierarchical_prior` *Compute marginal posterior probabilities (slab probabilities) that data points have non-zero mean for the hierarchical prior.*

---

### Description

Compute marginal posterior probabilities (slab probabilities) that data points have non-zero mean for the hierarchical prior.

### Usage

```
SSS_hierarchical_prior(log_phi_psi, logprior, show_progress = TRUE)
```

### Arguments

| | |
|---|---|
| `log_phi_psi` | List {logphi, logpsi} containing two vectors of the same length n that represent a preprocessed version of the data. logphi and logpsi should contain the logs of the phi and psi densities of the data points, as produced for instance by `SSS_log_phi_psi_Laplace` or `SSS_log_phi_psi_Cauchy` |
| `logprior` | vector of length n+1 with components logprior[p]=log(pi_n(p)) for $p = 0, ..., n$ |
| `show_progress` | Boolean that indicates whether to show a progress bar |

### Value

Returns a vector with marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$.

---

`SSS_hierarchical_prior_binomial` *Compute marginal posterior probabilities (slab probabilities) that data points have non-zero mean using the general hierarchical prior algorithm, but specialized to the Beta[kappa,lambda]-binomial prior. This function is equivalent to calling `SSS_hierarchical_prior` with logprior = lbeta(kappa+(0:n),lambda+n-(0:n)) - lbeta(kappa,lambda) + lchoose(n,0:n), but more convenient when using the Beta[kappa,lambda]-binomial prior and with a minor interior optimization that avoids calculating the choose explicitly.*

---

### Description

Compute marginal posterior probabilities (slab probabilities) that data points have non-zero mean using the general hierarchical prior algorithm, but specialized to the Beta[kappa,lambda]-binomial prior. This function is equivalent to calling `SSS_hierarchical_prior` with logprior = lbeta(kappa+(0:n),lambda+n-(0:n)) - lbeta(kappa,lambda) + lchoose(n,0:n), but more convenient when using the Beta[kappa,lambda]-binomial prior and with a minor interior optimization that avoids calculating the choose explicitly.

### Usage

```
SSS_hierarchical_prior_binomial(
  log_phi_psi,
  kappa,
  lambda,
  show_progress = TRUE
)
```

### Arguments

| | |
|---|---|
| `log_phi_psi` | List {logphi, logpsi} containing two vectors of the same length n that represent a preprocessed version of the data. logphi and logpsi should contain the logs of the phi and psi densities of the data points, as produced for instance by `SSS_log_phi_psi_Laplace` or `SSS_log_phi_psi_Cauchy` |
| `kappa` | First parameter of the beta-distribution |
| `lambda` | Second parameter of the beta-distribution |
| `show_progress` | Boolean that indicates whether to show a progress bar |

### Value

Returns a vector with marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$.

---

## `SSS_log_phi_psi_Cauchy` *Calculate log of phi and psi marginal densities for Cauchy(gamma) slab*

---

### Description

Calculate log of densities phi and psi for data vector x, where

$$phi[i] = Normal(x[i], sigma^2)$$
$$psi[i]) = E_Cauchy(\theta)[Normal(x[i] - \theta, sigma^2)]$$

### Usage

```
SSS_log_phi_psi_Cauchy(x, sigma, gamma)
```

### Arguments

| | |
|---|---|
| `x` | data vector |
| `sigma` | standard deviation of observations |
| `gamma` | parameter of Cauchy slab density |

### Value

list (phi, psi), containing logs of phi and psi densities

`SSS_log_phi_psi_Laplace` *Calculate log of phi and psi marginal densities for Laplace(lambda) slab*

## Description

Calculate log of densities phi and psi for data vector x, where

$$phi[i] = Normal(x[i], sigma^2)$$

$$psi[i]) = E_Laplace(\theta)[Normal(x[i] - \theta, sigma^2)]$$

## Usage

```
SSS_log_phi_psi_Laplace(x, sigma, lambda)
```

## Arguments

| | |
|---|---|
| `x` | data vector |
| `sigma` | standard deviation of observations |
| `lambda` | parameter of Laplace slab density |

## Value

list (phi, psi), containing logs of phi and psi densities

`SSS_make_beta_grid` *Creates a vector of uniformly spaced grid points in the beta parametrization Ensures the number of generated grid points is >= mingridpoints (which does not have to be integer), and that their number is always odd so there is always a grid point at pi/4.*

## Description

Creates a vector of uniformly spaced grid points in the beta parametrization Ensures the number of generated grid points is >= mingridpoints (which does not have to be integer), and that their number is always odd so there is always a grid point at pi/4.

## Usage

```
SSS_make_beta_grid(minngridpoints)
```

## Arguments

| | |
|---|---|
| `minngridpoints` | Minimum number of grid points |

### Value

Vector of betagrid points

---

## SSS_postmean_Cauchy *Compute posterior means of data points for the Cauchy(gamma) slab*

---

### Description

Compute posterior means of data points for the Cauchy(gamma) slab

### Usage

```
SSS_postmean_Cauchy(x, logpsi, postprobs, sigma, gamma)
```

### Arguments

| | |
|---|---|
| x | Data vector of length n |
| logpsi | Vector of length n that represents a preprocessed version of the data. It should contain the logs of the psi densities of the data points, as produced by SSS_log_phi_psi_Cauchy. |
| postprobs | Vector of marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$. |
| sigma | standard deviation of observations |
| gamma | parameter of Cauchy slab density |

### Value

Vector of n posterior means

---

## SSS_postmean_Laplace *Compute posterior means of data points for the Laplace(lambda) slab*

---

### Description

Compute posterior means of data points for the Laplace(lambda) slab

### Usage

```
SSS_postmean_Laplace(x, logpsi, postprobs, sigma, lambda)
```

## Arguments

| | |
|---|---|
| `x` | Data vector of length n |
| `logpsi` | Vector of length n that represents a preprocessed version of the data. It should contain the logs of the psi densities of the data points, as produced by `SSS_log_phi_psi_Laplace`. |
| `postprobs` | Vector of marginal posterior slab probabilities that $x[i]$ has non-zero mean for $i = 1, ..., n$. |
| `sigma` | standard deviation of observations |
| `lambda` | parameter of Laplace slab density |

## Value

Vector of n posterior means

# Index