Bayesian Bridge Regression

In this vignette, the main functions of the package `BBHD` are explained. This vignette is split up into three sections:

1. Brief Introduction to Bayesian Bridge Regression
2. Algorithm Details
3. Comparisons and Examples

First, install the package by typing

```r
library(BBHD)
```

and the load the package. In this vignette we also use `BayesBridge` and `horseshoe` to make comparisons.

```r
library(horseshoe)
library(BayesBridge)
```

- **Note:** Package 'BayesBridge' was removed from the CRAN repository. Formerly available versions can be obtained from the archive. Archived on 2018-01-27 as no corrections were received despite reminders. Please use the canonical form https://CRAN.R-project.org/package=BayesBridge to link to this page.

1. Brief Introduction to Bayesian Bridge Regression

In the standard linear regression framework, the model is defined as:

$$y = X\beta + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2 I).$$

Here, y is the $n \times 1$ response vector, X is the $n \times p$ design matrix, β is the $p \times 1$ vector of coefficients to be estimated, and ε is the $n \times 1$ vector of *i.i.d.* normal errors with mean zero and variance σ^2. Bridge regression stems from the following regularization problem:

$$\min_{\beta}(y - X\beta)'(y - X\beta) + \lambda \sum_{j=1}^{p} |\beta_j|^{\alpha},$$

where $\lambda > 0$ is the tuning parameter controlling the degree of penalization, and $\alpha \in (0, 1]$ adjusts the concavity of the penalty function. This approach bridges the gap between a shrinkage and selection operator category, with the best subset selection penalty at one end and the $L1$ (or Lasso) penalty at the other.

Within the Bayesian framework, rather than minimizing the above equation, samples are drawn from the joint posterior distribution of β and model hyperparameters. The specific setup is:

$$y|\beta, \sigma^2 \sim N(X\beta, \sigma^2 I),$$

$$\pi(\beta_j) \propto exp(-\lambda|\beta_j|^{\alpha})$$

$$\alpha \sim Beta(\frac{1}{2}, \frac{1}{2})$$

Here, α may also be set as a fixed value, such as 0.5.

2. Algorithm Details

Our algorithm, **Metropolis-Hastings within Gibbs with adapted step size**, specifically tailored for linear regression, is detailed below:

Input: Response variable y, predictor matrix X, tuning parameter tuning, burn-in period burn, number of Monte Carlo samples nmc, thinning factor thin, α update method $\mathrm{method.alpha}$.

Output: Estimates and samples for regression coefficients β, variance σ^2, scale parameter λ, shape parameter α.

1. **Initialize and set parameters**:
   - total sampling length:
     $N \leftarrow \mathrm{tuning} + \mathrm{burn} + \mathrm{nmc}$
   - $n \leftarrow \mathrm{nrow}(X)$
   - $p \leftarrow \mathrm{ncol}(X)$
   - $\beta \leftarrow 0.5 \times \mathbf{1}_p$
   - $\sigma^2 \leftarrow 1$
   - $\lambda \leftarrow 1$
   - $\alpha \leftarrow 0.5$
2. **Define adjustment steps**:
   - For β and α define adjustment steps:
     - $S_\beta \leftarrow [0.75, 0.75, \ldots, 0.75]_{1\times p}$
     - $S_\alpha \leftarrow 0.25$
3. **Sampling process**:
   - **for** $i = 1$ to N:
     - **for** each dimension $j = 1$ to p:
       - Compute conditional mean $\mu_j \leftarrow \frac{(X^T y)_j - \sum_{k\neq j}(X^T X)_{jk}\beta_k}{(X^T X)_{jj}}$
       - Compute conditional variance $\sigma_j^2 \leftarrow \frac{\sigma^2}{(X^T X)_{jj}}$
       - Generate proposal $\beta_j^* \leftarrow \beta_j + \mathrm{N}(0, (S_\beta)_j)$
       - Calculate prior $T_1 \leftarrow \lambda(|\beta_j|^\alpha - |\beta_j^*|^\alpha)$
       - Calculate likelihood $T_2 \leftarrow \frac{1}{2\sigma_j^2}((\beta_j - \mu_j)^2 - (\beta_j^* - \mu_j)^2)$
       - Calculate acceptance probability $MH \leftarrow \min(1, \exp(T_1 + T_2))$
       - Sample from uniform distribution $U \sim \mathrm{Uniform}(0, 1)$
       - **if** $U \leq MH$:
         - Update $\beta\_j \leftarrow \beta_j^*$

- **if** $\text{method.alpha} = \text{"beta"}$, **repeat**:
  - Transformation $Z_0 \leftarrow \tan(\pi(\alpha - 0.5))$
  - Generate proposal $Z_{new} \leftarrow Z_0 + \mathrm{N}(0, S_\alpha)$
  - Convert to $\alpha_{new} \leftarrow 0.5 + \frac{\arctan(Z_{new})}{\pi}$
  - Calculate Jacobian ratio

$$T_1 \leftarrow \frac{1 + Z_0^2}{1 + Z_{new}^2}$$

  - Calculate prior ratio

$$T_2 \leftarrow \log(Beta(\alpha_{new}|0.5, 0.5)) - \log(Beta(\alpha|0.5, 0.5))$$

  - Calculate likelihood ratio

$$T_3 \leftarrow \log(\lambda)p\left(\frac{1}{\alpha_{\text{new}}} - \frac{1}{\alpha}\right) + p\left(\log\Gamma(1 + \frac{1}{\alpha}) - \log\Gamma(1 + \frac{1}{\alpha_{\text{new}}})\right) + \lambda\left(\sum|\beta|^{\alpha} - \sum|\beta|^{\alpha_{\text{new}}}\right)$$

  - Calculate acceptance probability

$$MH \leftarrow \min(1, T_1 \exp(T_2 + T_3))$$

  - Sample from uniform distribution

$$Ua \sim Unif(0, 1)$$

  - **if** $U_a \leq MH$:
    - Update $\alpha \leftarrow \alpha_{\text{new}}$
    - **break**
- **else**:
  - Keep α unchanged
- Update λ by sampling from $\mathrm{Gamma}(0.1 + \frac{p}{\alpha}, 0.1 + \sum|\beta|^{\alpha})$
- Update σ^2 by sampling from the inverse of $\mathrm{Gamma}(0.5n, 0.5\sum(y - X^T\beta)^2)$

4. **Adjustment of steps**:
   - Adjust steps based on acceptance rate during burn-in and tuning periods
5. **Storing results**:
   - Store sampling results every thin steps after surpassing the burn-in and tuning periods
6. **Inference statistics**:
   - Calculate posterior mean etc. for $\beta, \sigma^2, \lambda, \alpha$

3. Comparisons and Examples

high dimensions case:

```r
eval.select <- function(Type,Select){
  L = length(Type)
  FP = 0
  FN = 0
  N0 = 0
  N1 = 0
  for (i in 1:L) {
    if(Type[i]==0){ # Negative truth
      N0 = N0 + 1
      if(Select[i]==1){FP = FP + 1}
    }
    if(Type[i]==1){ # Positive truth
      N1 = N1 + 1
      if(Select[i]==0){FN = FN + 1}
    }
  }
  return(list(FP = FP, FN = FN))
}

tuning = 40000
Tb = 400
burn_in = 30000
nmc = 30000
thin = 15

set.seed(20240313)

Time = 250
J = 500
s = 10
psi = 2*sqrt(2*log(J))

X = matrix(rnorm(Time*J,0,1),Time,J);X = X/sqrt(Time)
Type = rep(0,J)
Type[sample(1:J, s)] = 1
beta.true = Type*rnorm(J,0,psi)
beta.true = beta.true/sd(beta.true)
u = rnorm(Time,0,1/3)
y = as.vector(X%*%beta.true + u)

HSSamples <- horseshoe(y,X,
                       method.tau = "halfCauchy",
                       method.sigma = "Jeffreys",
                       Sigma2 = 1,
                       burn = burn_in,
                       nmc = nmc,
                       thin = thin)

BBR <- bridge.reg.stb(y, X, nsamp=nmc, alpha=0.5,
```

```
                         sig2.shape=0.0, sig2.scale=0.0, nu.shape=2.0, nu.rate=2.0, burn = burn_in)

BBR_mycode <- BBLR(y,X,tuning,Tb,burn_in,nmc,thin,method.alpha="beta")

## diagnostics

## estimation
# beta RMSE
sqrt(mean((HSSamples$BetaHat-beta.true)^2))

BBR.mean = colMeans(BBR$beta)
sqrt(mean((BBR.mean-beta.true)^2))

sqrt(mean((BBR_mycode$BetaHat-beta.true)^2))

# noise RMSE
sqrt(mean((HSSamples$BetaHat[Type==0]-beta.true[Type==0])^2))

sqrt(mean((BBR.mean[Type==0]-beta.true[Type==0])^2))

sqrt(mean((BBR_mycode$BetaHat[Type==0]-beta.true[Type==0])^2))

# signal RMSE
sqrt(mean((HSSamples$BetaHat[Type==1]-beta.true[Type==1])^2))

sqrt(mean((BBR.mean[Type==1]-beta.true[Type==1])^2))

sqrt(mean((BBR_mycode$BetaHat[Type==1]-beta.true[Type==1])^2))

## selection
BBR$LeftCI = rep(0,length(BBR.mean))
BBR$RightCI = rep(0,length(BBR.mean))
for (ci in 1:length(BBR.mean)) {
  BBRorder = sort(BBR$beta[,ci])
  BBR$LeftCI[ci] = BBRorder[0.025*nmc+1]
  BBR$RightCI[ci] = BBRorder[0.975*nmc]
}

HS.select = rep(1,J)
BB.select = rep(1,J)
myBB.select = rep(1,J)
for (j in 1:J) {
  if(HSSamples$LeftCI[j]<0 & HSSamples$RightCI[j]>0){HS.select[j]=0}
  if(BBR_mycode$LeftCI[j]<0 & BBR_mycode$RightCI[j]>0){myBB.select[j]=0}
  if(BBR$LeftCI[j]<0 & BBR$RightCI[j]>0){BB.select[j]=0}
}

eval.select(Type,HS.select)
eval.select(Type,BB.select)
eval.select(Type,myBB.select)
```

low dimensions case:

```r
eval.select <- function(Type,Select){
  L = length(Type)
  FP = 0
  FN = 0
  N0 = 0
  N1 = 0
  for (i in 1:L) {
    if(Type[i]==0){ # Negative truth
      N0 = N0 + 1
      if(Select[i]==1){FP = FP + 1}
    }
    if(Type[i]==1){ # Positive truth
      N1 = N1 + 1
      if(Select[i]==0){FN = FN + 1}
    }
  }
  return(list(FP = FP, FN = FN))
}

tuning = 40000
Tb = 400
burn_in = 30000
nmc = 30000
thin = 15

set.seed(20240313)

Time = 500
J = 100
s = 10
psi = 2*sqrt(2*log(J))

X = matrix(rnorm(Time*J,0,1),Time,J);X = X/sqrt(Time)
Type = rep(0,J)
Type[sample(1:J, s)] = 1
beta.true = Type*rnorm(J,0,psi)
beta.true = beta.true/sd(beta.true)
u = rnorm(Time,0,1/3)
y = as.vector(X%*%beta.true + u)

HSSamples <- horseshoe(y,X,
                       method.tau = "halfCauchy",
                       method.sigma = "Jeffreys",
                       Sigma2 = 1,
                       burn = burn_in,
                       nmc = nmc,
                       thin = thin)

BBR <- bridge.reg.stb(y, X, nsamp=nmc, alpha=0.5,
```

```
                          sig2.shape=0.0, sig2.scale=0.0, nu.shape=2.0, nu.rate=2.0, burn = burn_in)

BBR_mycode <- BBLR(y,X,tuning,Tb,burn_in,nmc,thin,method.alpha="beta")

## diagnostics

## estimation
# beta RMSE
sqrt(mean((HSSamples$BetaHat-beta.true)^2))

BBR.mean = colMeans(BBR$beta)
sqrt(mean((BBR.mean-beta.true)^2))

sqrt(mean((BBR_mycode$BetaHat-beta.true)^2))

# noise RMSE
sqrt(mean((HSSamples$BetaHat[Type==0]-beta.true[Type==0])^2))

sqrt(mean((BBR.mean[Type==0]-beta.true[Type==0])^2))

sqrt(mean((BBR_mycode$BetaHat[Type==0]-beta.true[Type==0])^2))

# signal RMSE
sqrt(mean((HSSamples$BetaHat[Type==1]-beta.true[Type==1])^2))

sqrt(mean((BBR.mean[Type==1]-beta.true[Type==1])^2))

sqrt(mean((BBR_mycode$BetaHat[Type==1]-beta.true[Type==1])^2))

## selection
BBR$LeftCI = rep(0,length(BBR.mean))
BBR$RightCI = rep(0,length(BBR.mean))
for (ci in 1:length(BBR.mean)) {
  BBRorder = sort(BBR$beta[,ci])
  BBR$LeftCI[ci] = BBRorder[0.025*nmc+1]
  BBR$RightCI[ci] = BBRorder[0.975*nmc]
}

HS.select = rep(1,J)
BB.select = rep(1,J)
myBB.select = rep(1,J)
for (j in 1:J) {
  if(HSSamples$LeftCI[j]<0 & HSSamples$RightCI[j]>0){HS.select[j]=0}
  if(BBR_mycode$LeftCI[j]<0 & BBR_mycode$RightCI[j]>0){myBB.select[j]=0}
  if(BBR$LeftCI[j]<0 & BBR$RightCI[j]>0){BB.select[j]=0}
}

eval.select(Type,HS.select)
eval.select(Type,BB.select)
eval.select(Type,myBB.select)
```