HW3

Michael Miller

2022-11-06

Load Libraries

```
library(bench)
library(devtools)
```

```
## Loading required package: usethis
```

```
devtools::install_github("mikemiller442/fastHierarchicalReg")
```

```
## Skipping install of 'fastHierarchicalReg' from a github remote, the SHA1 (1eec71a0) has not changed s
##   Use `force = TRUE` to force installation
```

```
devtools::install_github("kangjian2016/fastBayesReg")
```

```
## Skipping install of 'fastBayesReg' from a github remote, the SHA1 (5ffa15cd) has not changed since la
##   Use `force = TRUE` to force installation
```

```
library(fastHierarchicalReg)
```

```
## Loading required package: parallel
```

```
## Loading required package: foreach
```

```
## Loading required package: roxygen2
```

```
library(fastBayesReg)
```

```
## Loading required package: Rcpp
```

```
## Loading required package: RcppArmadillo
```

```
## Loading required package: glmnet
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
## Loading required package: horseshoe
```

```
## Loading required package: pgdraw
```

Simulate Data 1

```
n <- 10000   # subjects
numBeta <- 100   # covariates
betaSD <- 0.75   # standard deviation of the randomly sampled covariates
XSD <- 0.5   # standard deviation of the generated features
errorSD <- 2.0   # regression standard deviation
e <- rnorm(n, mean = 0, sd = errorSD)
beta <- rnorm(numBeta, mean = 0, sd = betaSD*errorSD)
Z <- matrix(NA, nrow = n, ncol = numBeta)
for (i in 1:ncol(Z)) {
  Z[,i] <- rnorm(n, mean = 0, sd = XSD)
}

y <- Z %*% beta + e
output <- y
```

Run Gibbs Sampler

```
X <- Z
testX <- Z
resp <- output
testResp <- output

numEpochs <- 10000
numDiscard <- 2000
numChains <- 4
numCores <- 8
lambdaSqPrior <- 1.0
regVarPrior <- 1.0

res <- fastHierarchicalReg::linRegGibbsProcessed(X = X,
                                                 testX = testX,
                                                 Y = resp,
                                                 testY = testResp,
                                                 lambdaSqPrior = lambdaSqPrior,
                                                 regVarPrior = regVarPrior,
                                                 numEpochs = numEpochs,
                                                 numDiscard = numDiscard,
                                                 numChains = numChains,
                                                 numCores = numCores)
```

```
## socket cluster with 8 nodes on host 'localhost'
```

Compare means between model output and true values

```
# Comparing beta
paramBeta <- as.numeric(res$postMeanList$beta)
all.equal(beta,paramBeta)
```

```
## [1] "Mean relative difference: 0.02818151"
```

```
# Comparing lambda
paramLambda <- sqrt(as.numeric(res$postMeanList$lambdaSq))
all.equal(betaSD,paramLambda)
```

```
## [1] "Mean relative difference: 0.08877533"
```

```
# Comparing sigma
paramSigma <- sqrt(as.numeric(res$postMeanList$regVar))
all.equal(errorSD,paramSigma)
```

```
## [1] "Mean relative difference: 0.001023992"
```

Rhat Convergence Diagnostic - Check Convergence to 1.0

```
all.equal(as.numeric(unlist(res$RhatList)),rep(1.0,length(unlist(res$RhatList))))
```

```
## [1] "Mean relative difference: 3.432571e-05"
```

Simulate Data 2

```
n <- 10000
numBeta <- 5
betaSD <- 0.75
XSD <- 0.5
errorSD <- 2.0
e <- rnorm(n, mean = 0, sd = errorSD)
beta <- rnorm(numBeta, mean = 0, sd = betaSD*errorSD)
Z <- matrix(NA, nrow = n, ncol = numBeta)
for (i in 1:ncol(Z)) {
  Z[,i] <- rnorm(n, mean = 0, sd = XSD)
}

y <- Z %*% beta + e
output <- y
```

Benchmark Gibbs Sampler Against fastBayesReg Package For Accuracy

```
X <- Z
testX <- Z
resp <- output
testResp <- output

numEpochs <- 4000
numDiscard <- 2000
```

```
numChains <- 4
lambdaSqPrior <- 1.0
regVarPrior <- 1.0

res <- fastHierarchicalReg::linRegGibbsProcessed(X = X,
                                                 testX = testX,
                                                 Y = resp,
                                                 testY = testResp,
                                                 lambdaSqPrior = lambdaSqPrior,
                                                 regVarPrior = regVarPrior,
                                                 numEpochs = numEpochs,
                                                 numDiscard = numDiscard,
                                                 numChains = numChains,
                                                 numCores = numCores)
```

```
## socket cluster with 8 nodes on host 'localhost'
```

```
numEpochs <- 10000
numDiscard <- 2000

resKJ <- fastBayesReg::fast_normal_lm(y = resp,
                                      X = X,
                                      mcmc_sample = numEpochs,
                                      burnin = numDiscard,
                                      a_sigma = 0.1,
                                      b_sigma = 0.1)
```

Compare regression coefficient posterior means between model output and fastBayesReg

```
# Comparing posterior means
kjBeta <- as.numeric(resKJ$post_mean$betacoef)
paramBeta <- as.numeric(res$postMeanList$beta)
all.equal(kjBeta,paramBeta)
```

```
## [1] "Mean relative difference: 0.0005414884"
```

Benchmark High Dimensional Example For Speed

```
n <- 1000
numBeta <- 500
betaSD <- 0.05
XSD <- 0.5
errorSD <- 2.0
e <- rnorm(n, mean = 0, sd = errorSD)
beta <- rnorm(numBeta, mean = 0, sd = betaSD*errorSD)
Z <- matrix(NA, nrow = n, ncol = numBeta)
for (i in 1:ncol(Z)) {
  Z[,i] <- rnorm(n, mean = 0, sd = XSD)
}
```

```
y <- Z %*% beta + e
output <- y

funMod <- function() {
  X <- Z
  testX <- Z
  resp <- output
  testResp <- output

  numEpochs <- 4000
  numDiscard <- 2000
  lambdaSqPrior <- 1.0
  regVarPrior <- 1.0

  res <- fastHierarchicalReg::linRegGibbs(X = X,
                                          testX = testX,
                                          Y = resp,
                                          testY = testResp,
                                          numEpochs = numEpochs,
                                          regVarPrior = regVarPrior,
                                          lambdaSqPrior = lambdaSqPrior)
  postBeta <- res$coefBeta[,(numDiscard+2):(numEpochs+1)]
  paramBeta <- as.numeric(rowMeans(postBeta))
  return(paramBeta)
}

funKJ <- function() {
  numEpochs <- 4000
  numDiscard <- 2000
  resKJ <- fastBayesReg::fast_normal_lm(y = resp,
                                        X = X,
                                        mcmc_sample = numEpochs,
                                        burnin = numDiscard,
                                        a_sigma = 0.1,
                                        b_sigma = 0.1)
  kjBeta <- as.numeric(resKJ$post_mean$betacoef)
  return(kjBeta)
}

benchMarkRes <- bench::mark(funMod(),
                            funKJ(),
                            iterations = 2,
                            check = FALSE)
```

```
## Warning: Some expressions had a GC in every iteration; so filtering is disabled.
```

```
benchMarkTable <- benchMarkRes[c("expression", "min", "median","mem_alloc","n_gc")]
knitr::kable(benchMarkTable)
```

expression	min	median	mem_alloc	n_gc
funMod()	25.4s	26.6s	22.8GB	710

expression	min	median	mem_alloc	n_gc
funKJ()	50.7ms	52.8ms	299.6KB	0