Bayesian ridge regression and Bayesian lasso

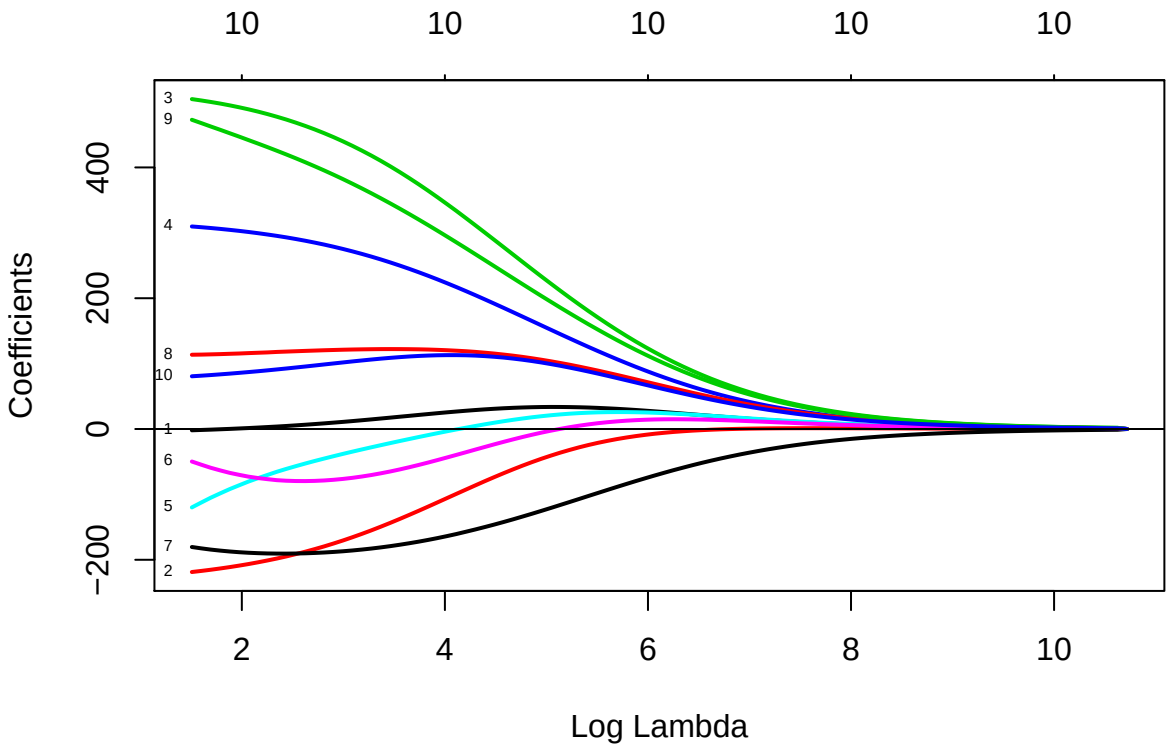
Frank van der Meulen (TU Delft)

1 May 2015

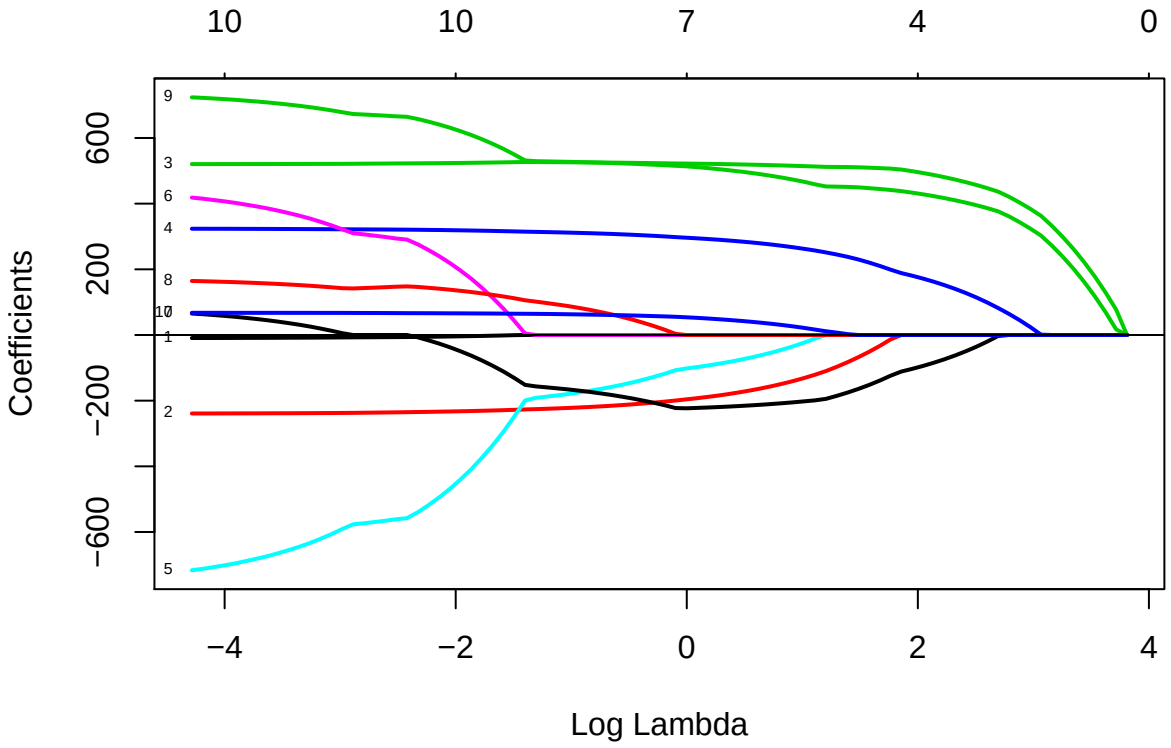
ILLUSTRATION OF THE FULL RIDGE AND LASSO REGULARISATION PATHS USING GLMNET

The glmnet package contains many function for estimation of generalised linear models using ℓ_1 and or ℓ_2 regularisation. We illustrate it with a dataset that is available within R.

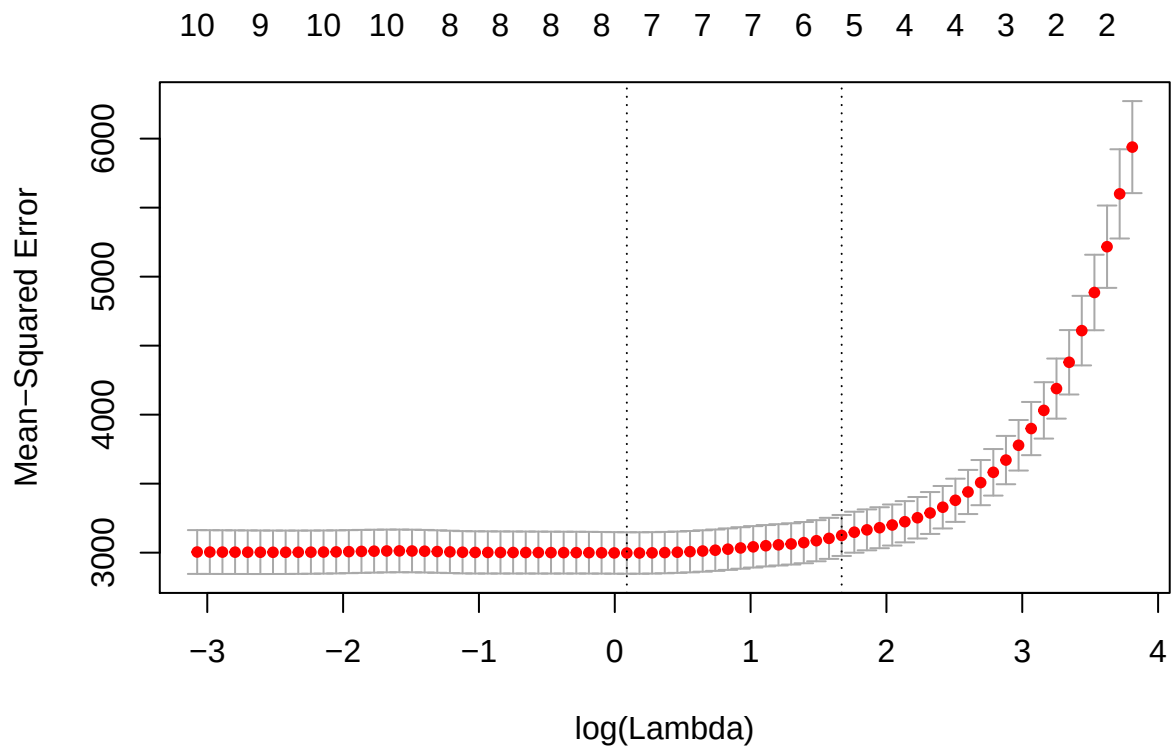
```
library(glmnet)
library(lars)   # contains the diabetes data
data(diabetes)
d  <- diabetes

lasfit <- glmnet(d$x,d$y,alpha=1)  # lasso
ridfit <- glmnet(d$x,d$y,alpha=0)  # ridge

plot(ridfit,lwd=2,xvar='lambda',label=TRUE)
abline(0,0)
```




```
plot(lasfit,lwd=2,xvar='lambda',label=TRUE)
abline(0,0)
```




```
# cross validation plot for lasso
cvfit = cv.glmnet(d$x, d$y, alpha=1,nfolds=20) # 5-fold cv
plot(cvfit)
```



```
cvfit$lambda.min   # value of lambda that gives minimum cvm.
```

```
## [1] 1.092931
```

```
cvfit$lambda.1se  # largest value of lambda such that error is within 1 standard error of the minimum.
```

```
## [1] 5.314486
```

```
coef(lasfit,s=cvfit$lambda.min)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  152.13348
## age            .
## sex         -192.47489
## bmi          521.64423
## map          294.40618
## tc           -97.72765
## ldl            .
## hdl         -222.10137
## tch            .
## ltg          511.10045
## glu           52.15790
```

```
coef(lasfit,s=cvfit$lambda.1se)
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                       1
## (Intercept)  152.13348
## age            .
## sex          -33.35229
## bmi          508.13935
## map          210.34606
## tc             .
## ldl            .
## hdl         -138.84433
## tch            .
## ltg          444.59064
## glu            .
```

ILLUSTRATION OF RIDGE, LASSO, BAYESIAN RIDGE AND BAYESIAN LASSO

Install the monomvn package and call the library.

```
library(monomvn)
```

The following function generates sparse data from the linear regression model:

```
genSparseData <- function(n,p,beta0.nonzero,sd.noise)
# generates the n x p design matrix with all elements drawn from a standard normal distribution
# beta0 is the vector beta0.nonzero, augmented with zeros
# a list containing (x, y, beta0) is returned, where
#  y = x * beta0 + eps,
# where eps is a vector of independent normal random variables
# with mean zero and st.dev equal to sd.noise
{
  r <- length(beta0.nonzero)
  if (p< (r+2))
  {   stop('p is too small in relation with the nr of nonzero elements in beta0.nonzero')
  }  else  {
     x <- matrix(rnorm(n*p),nrow=n)
     beta0 <- c(beta0.nonzero, rep(0,p-r))
     y <- x %*% beta0 + rnorm(n, sd=sd.noise)
     list(x=x,y=y, beta0=beta0)
  }
}
```

TEST CASE: n=25, p=500

Generate the data

```
set.seed(38)   # fix the seed of the RNG
d1 <- genSparseData(25,100,c(5, -3, 1, 0.2),.1)
str(d1)
```

```
## List of 3
##  $ x    : num [1:25, 1:100] -0.2536 -1.0556 0.6865 0.0252 -1.6718 ...
##  $ y    : num [1:25, 1] -1.54 -4.36 4.97 -3.39 -7.52 ...
##  $ beta0: num [1:100] 5 -3 1 0.2 0 0 0 0 0 0 ...
```

Ordinary Least Squares regression

As $p > n$ this will not work:

```
reg.ols <- lm(d1$y ~ d1$x)
summary(reg.ols)$coef[1:50]  # print first 50 coefs
```

```
##  [1] -0.283462782  4.669685040 -2.918809552  0.934978200 -0.150529666
##  [6]  0.593930135 -0.249878902 -0.364522097  0.020569314 -0.090580776
## [11]  0.031054241 -0.180278334  0.417571414  0.410551748  0.108340973
## [16]  0.186894642 -0.006294776 -0.124836427 -0.321128549  0.396752935
## [21] -0.599354235  0.179711848 -0.245206735 -0.110487848  0.251961652
## [26]          NaN          NaN          NaN          NaN          NaN
## [31]          NaN          NaN          NaN          NaN          NaN
## [36]          NaN          NaN          NaN          NaN          NaN
## [41]          NaN          NaN          NaN          NaN          NaN
## [46]          NaN          NaN          NaN          NaN          NaN
```

Ridge regression

The default choice for setting the regularisation parameter is 10-fold cross-validation.

```
rid <- regress(d1$x, d1$y, method="ridge")
# note that this function itself always automatically includes an intercept

ridCoef <- rid$b   # vector of coefficients (note that an intercept is included!)
ridCoef[1:15]      # print first 15 elements
```

```
##  [1] -0.15736762  0.92029473 -0.53315934  0.10373099  0.13218585
##  [6]  0.32623143  0.21728364 -0.03910456  0.25160404 -0.02652536
## [11]  0.15573363 -0.15967645 -0.03379762  0.19125742 -0.03205427
```

```
rid$lambda         # regularisation parameter
```

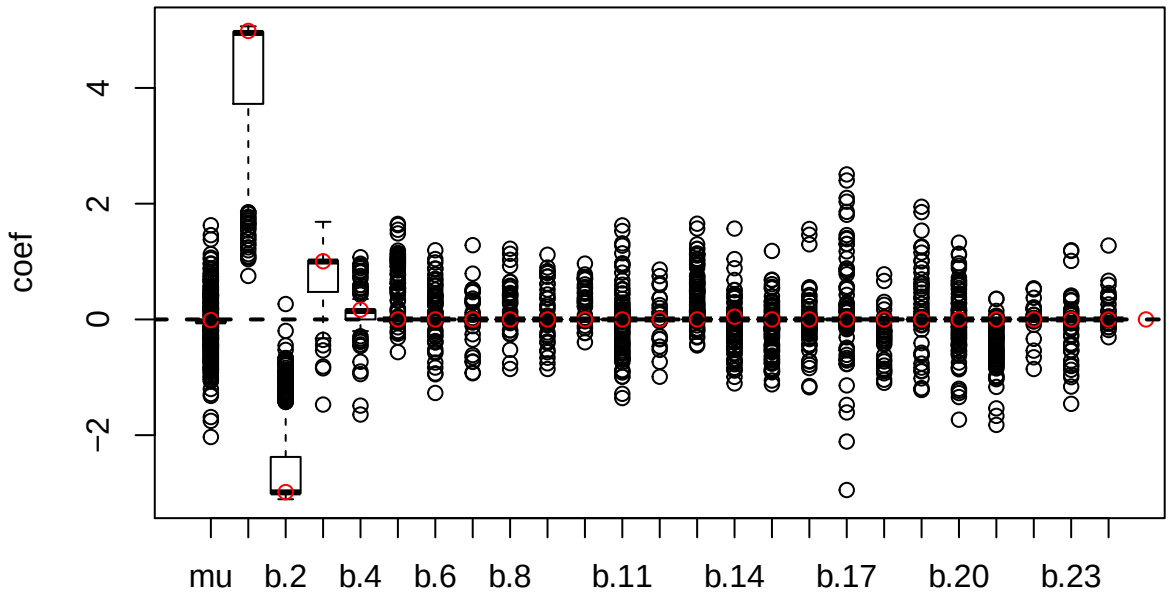
```
## [1] 4.317531
```

Bayesian ridge regression

```
ridBayes <- bridge(d1$x,d1$y,T=1000)
```

```
## t=100, m=23
## t=200, m=24
## t=300, m=24
## t=400, m=23
## t=500, m=6
## t=600, m=4
## t=700, m=7
## t=800, m=4
## t=900, m=7
```

```
BI <- 250   # choose a value for burnin
plot(ridBayes,burnin=BI)
```




```
ridBayesCoef <- cbind(ridBayes$mu, ridBayes$beta)[BI:T,]
ridgeBayesMean <- colMeans(ridBayesCoef)
ridgeBayesMedian <- apply(ridBayesCoef,2,median)
results <- data.frame(truth=c(0,d1$beta0), ridge=rid$b,
                      Bridge.mean=ridgeBayesMean,
                      Bridge.median=ridgeBayesMedian)
results[1:15,]
```

```
##       truth       ridge  Bridge.mean Bridge.median
##         0.0 -0.15736762 -0.078330593   -0.01617783
## b.1     5.0  0.92029473  2.325002119    2.47141466
## b.2    -3.0 -0.53315934 -1.420499366   -1.54504220
## b.3     1.0  0.10373099  0.030292466    0.00000000
## b.4     0.2  0.13218585  0.024359859    0.00000000
## b.5     0.0  0.32623143  0.046947740    0.00000000
## b.6     0.0  0.21728364  0.012758534    0.00000000
## b.7     0.0 -0.03910456 -0.009934238    0.00000000
## b.8     0.0  0.25160404  0.027456727    0.00000000
## b.9     0.0 -0.02652536 -0.011492532    0.00000000
## b.10    0.0  0.15573363  0.129380055    0.00000000
## b.11    0.0 -0.15967645 -0.005949868    0.00000000
## b.12    0.0 -0.03379762 -0.002303823    0.00000000
## b.13    0.0  0.19125742  0.088334363    0.00000000
## b.14    0.0 -0.03205427  0.009160131    0.00000000
```

Here we have taken a fairly small number of iterations, if possible take more in real applications.

Many results can be obtained from the summmary function

```
?summary.blasso
```

For example, the estimated posterior probability that the individual components of the regression coefficients beta is nonzero is obtained from

```
bn0 <- summary(ridBayes)$bn0
bn0[1:15]   # print the first 15 elements
```

```
##   b.1   b.2   b.3   b.4   b.5   b.6   b.7   b.8   b.9  b.10  b.11  b.12
## 0.975 0.979 0.618 0.631 0.104 0.102 0.071 0.078 0.088 0.105 0.116 0.070
##  b.13  b.14  b.15
## 0.178 0.161 0.115
```

Lasso regression

```
las <- regress(d1$x, d1$y, method="lasso")
lasCoef <- las$b
lasCoef[1:15]
```

```
##  [1] -0.03637022  4.90330600 -2.96080008  0.96275693  0.12709471
##  [6]  0.00000000  0.00000000  0.00000000  0.00000000  0.00000000
## [11]  0.00000000  0.00000000  0.00000000  0.00000000  0.00938084
```

```
lasLambda <- las$lambda
lasLambda
```

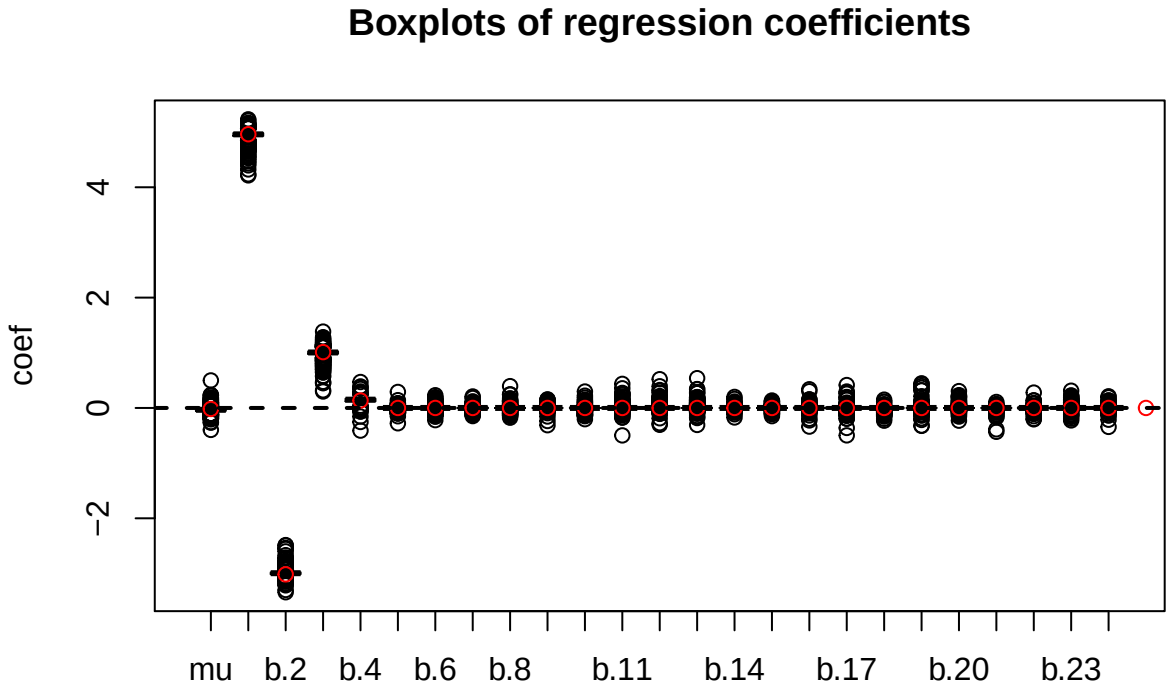
```
## [1] 0.1271622
```

Bayesian lasso regression

```
lasBayes <- blasso(d1$x,d1$y,T=1000)
```

```
## t=100, m=14
## t=200, m=11
## t=300, m=10
## t=400, m=19
## t=500, m=10
## t=600, m=24
## t=700, m=21
## t=800, m=8
## t=900, m=10
```

```
BI <- 250   # choose a value for burnin
plot(lasBayes,burnin=BI)
```




```
lasBayesCoef <- cbind(lasBayes$mu, lasBayes$beta)[BI:T,]
lassoBayesMean <- colMeans(lasBayesCoef)
lassoBayesMedian <- apply(lasBayesCoef,2,median)
results <- data.frame(truth=c(0,d1$beta0), lasso=las$b,
                      Blasso.mean=lassoBayesMean,
                      Blasso.median=lassoBayesMedian)
results[1:15,]
```

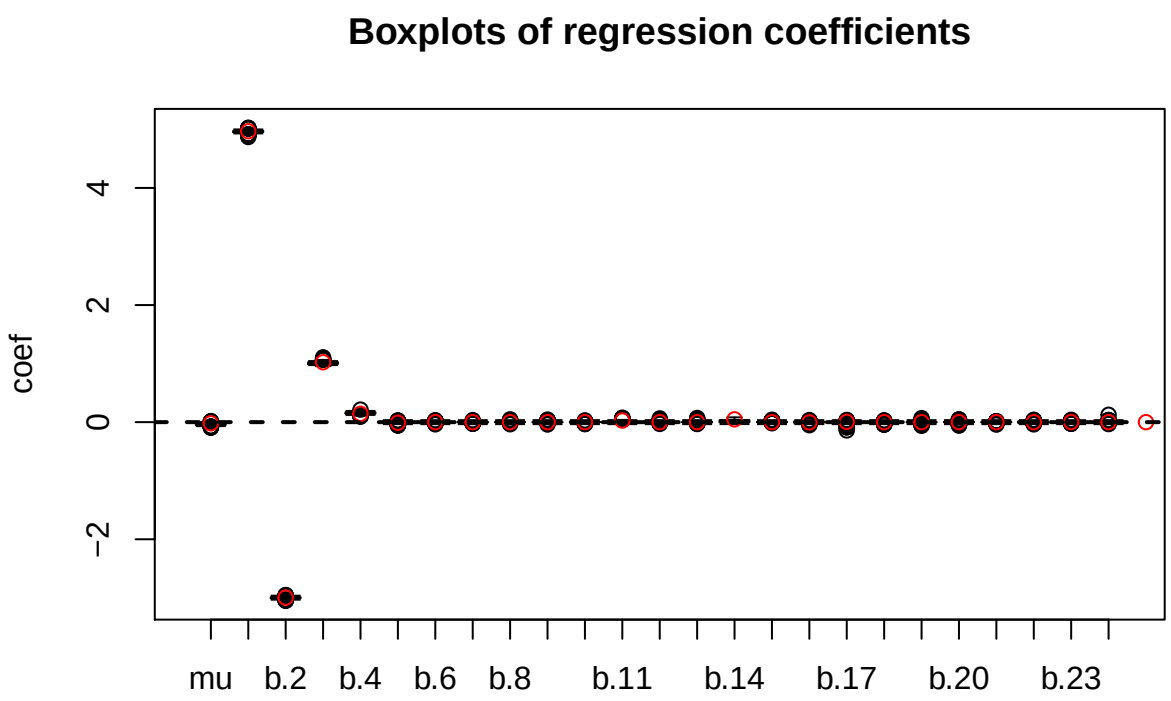
```
##       truth       lasso   Blasso.mean Blasso.median
##         0.0 -0.03637022 -5.528043e-02   -0.03030157
## b.1     5.0  4.90330600  4.714824e+00    4.96114620
## b.2    -3.0 -2.96080008 -2.841021e+00   -2.99477934
## b.3     1.0  0.96275693  9.482718e-01    1.00673329
## b.4     0.2  0.12709471  1.159496e-01    0.15135666
## b.5     0.0  0.00000000 -7.536528e-04    0.00000000
## b.6     0.0  0.00000000 -2.281772e-03    0.00000000
## b.7     0.0  0.00000000  5.834138e-06    0.00000000
## b.8     0.0  0.00000000 -4.407082e-04    0.00000000
## b.9     0.0  0.00000000  5.904413e-04    0.00000000
## b.10    0.0  0.00000000 -3.278002e-05    0.00000000
## b.11    0.0  0.00000000  6.732857e-03    0.00000000
## b.12    0.0  0.00000000 -7.400661e-04    0.00000000
## b.13    0.0  0.00000000  1.134870e-02    0.00000000
## b.14    0.0  0.00938084  1.564100e-02    0.00000000
```

Horseshoe regression

```
hs <- bhs(d1$x,d1$y,T=1000)
```

```
## t=100, m=24
## t=200, m=24
## t=300, m=22
## t=400, m=24
## t=500, m=24
## t=600, m=24
## t=700, m=24
## t=800, m=24
## t=900, m=24
```

```
BI <- 250   # choose a value for burnin
plot(hs,burnin=BI)
```




```
hsCoef <- cbind(hs$mu, hs$beta)[BI:T,]
hsMean <- colMeans(hsCoef)
hsMedian <- apply(hsCoef,2,median)
results <- data.frame(truth=c(0,d1$beta0),
                      hs.mean=hsMean,
                      hs.median=hsMedian)
results[1:15,]
```

```
##      truth       hs.mean   hs.median
##        0.0  0.0390685009 -0.03415112
## b.1    5.0  4.6854096620  4.95795955
## b.2   -3.0 -2.4018622091 -2.99233925
## b.3    1.0  0.6810286939  0.99902745
## b.4    0.2  0.1101165560  0.14864912
## b.5    0.0  0.0083831252  0.00000000
## b.6    0.0  0.0006003356  0.00000000
## b.7    0.0 -0.0112578890  0.00000000
## b.8    0.0 -0.0107512720  0.00000000
```

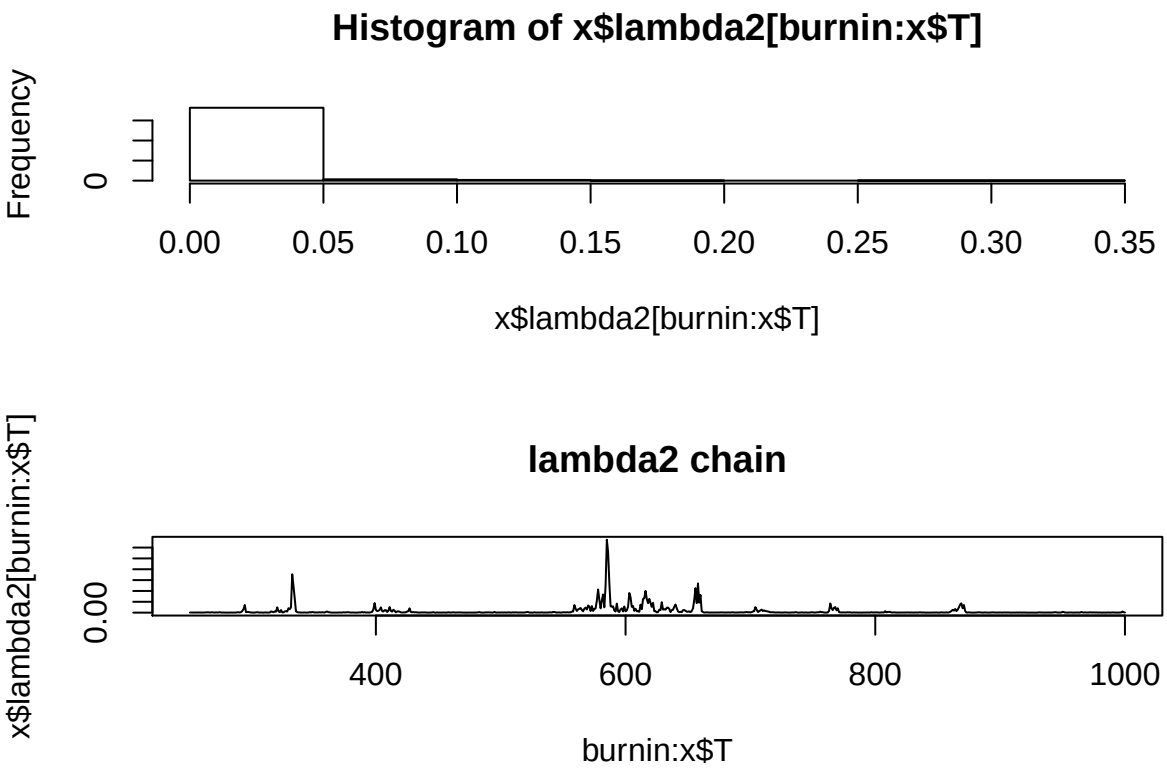
```
## b.9    0.0 -0.0009990126  0.00000000
## b.10   0.0  0.0200152191  0.00000000
## b.11   0.0 -0.0059784126  0.00000000
## b.12   0.0  0.0031806283  0.00000000
## b.13   0.0  0.0014306550  0.00000000
## b.14   0.0  0.0068611827  0.00000000
```

If you wish to make additional plots, consult the help page for the plotting function

```
?plot.blasso
```

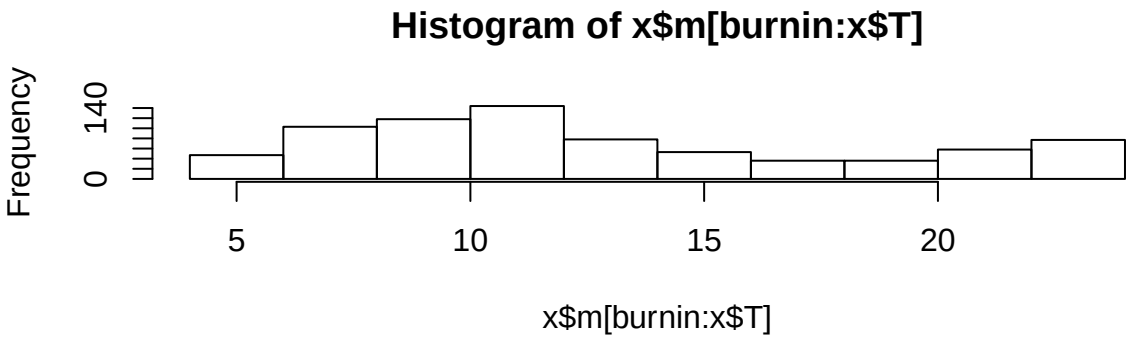
As an example, suppose for the Bayesian lasso we wish to make a traceplot for lambda:

```
plot(lasBayes,burnin=BI,which='lambda')
```



Or plot the size of different models visited

```
plot(lasBayes,burnin=BI,which='m')
```



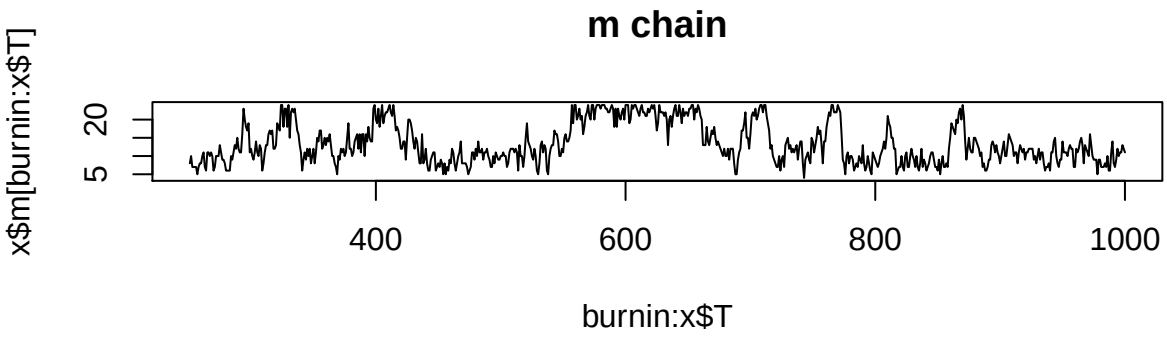


ILLUSTRATION OF RECOVERING A SPARSE SIGNAL

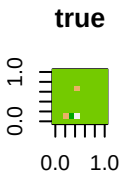
Here is another illustration where we try to recover a sparse signal. We use both the (frequentist) lasso and horseshoe

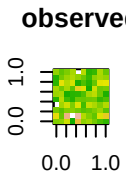
```
IT <- 1000
BI <- 250

# truth
n <- 10
y0 <- rep(0,n^2)
y0[13:15] <- c(3,-2,5)
y0[65] <- 3

# observations
y <- y0 + rnorm(n^2,0.1)

# freq. lasso
las <- regress(diag(n^2),y-mean(y),'lasso')
lasCoef <- las$b[-1]

# horseshoe
hs <- bhs(X=diag(n^2),y,T=IT)
```

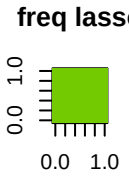
```
## t=100, m=46
## t=200, m=46
## t=300, m=54
```

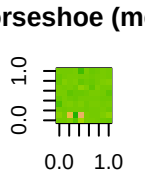
```
## t=400, m=47
## t=500, m=50
## t=600, m=48
## t=700, m=51
## t=800, m=42
## t=900, m=56
```

```
hsPostMean <- colMeans(hs$beta[BI:IT,])
hsPostMed <- apply(hs$beta[BI:IT,],2,median)
par(mfrow=c(3,2))
par(pty="s")
image(matrix(y0,n),main='true',col=terrain.colors(100),zlim=c(-2,5))
image(matrix(y,n),main='observed',col=terrain.colors(100),zlim=c(-2,5))
image(matrix(lasCoef,n),main='freq lasso',col=terrain.colors(100),zlim=c(-2,5))
image(matrix(hsPostMean,n),main='horseshoe (mean)',col=terrain.colors(100),zlim=c(-2,5))
image(matrix(hsPostMed,n),main='horseshoe (med)',col=terrain.colors(100),zlim=c(-2,5))

#data.frame(y0,y,lasCoef,hsCoef)
```









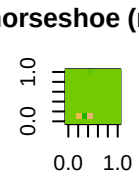


ILLUSTRATION OF GIBBS SAMPLER FOR RIDGE REGRESSION

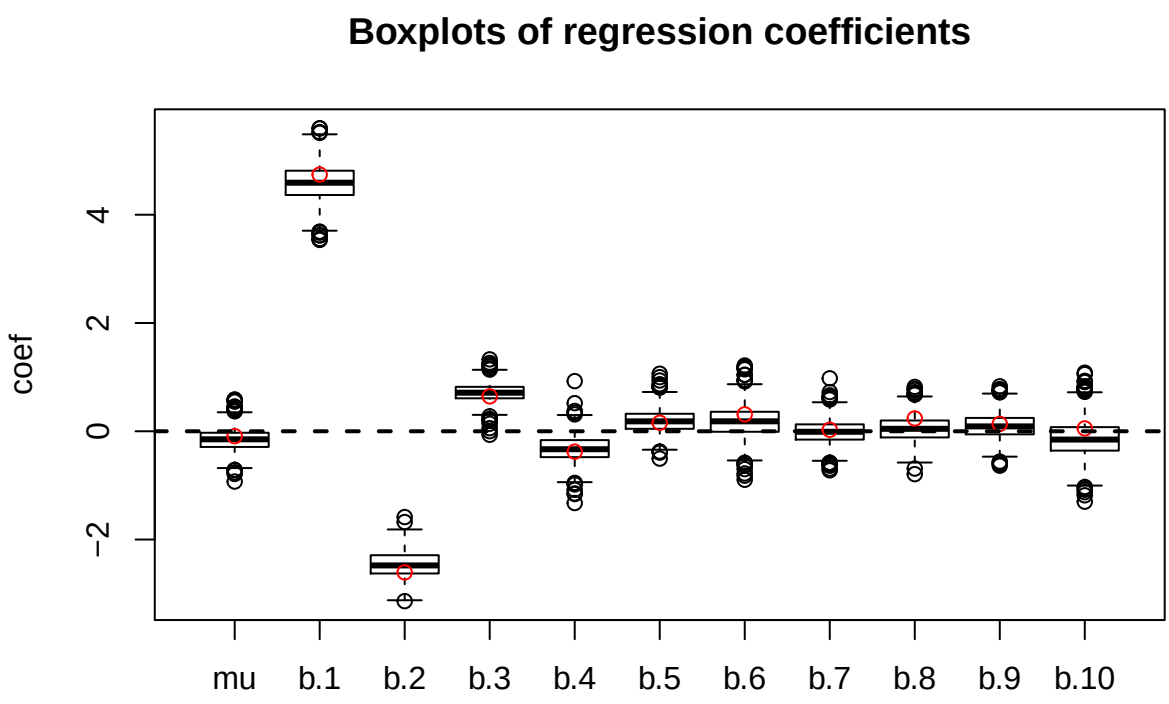
First we call the bridge function, then we use some plots from the coda library. With this library trace plots of the mcmc sampler can easilly be made.

```
IT <- 1000 # nr of mcmc iterations
BI <- 100  # nr of burnin iterations
```

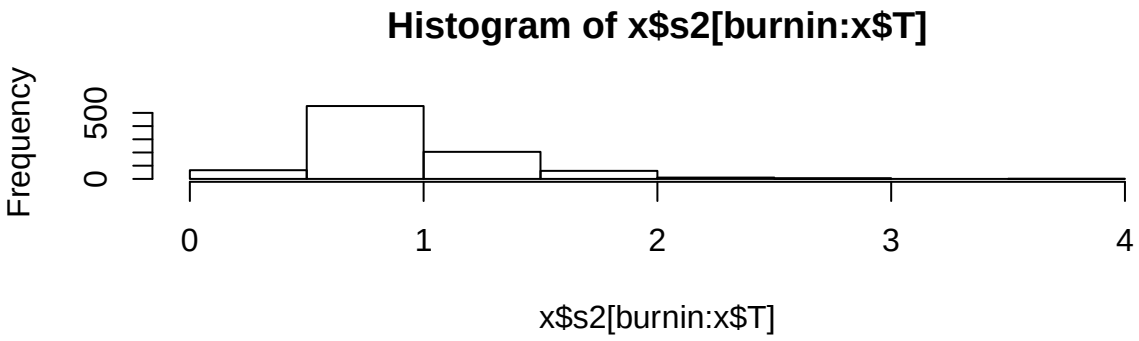
```
d1 <- genSparseData(25,10,c(5, -3, 1, 0.2),1)
ridBayes <- bridge(d1$x,d1$y,T=IT,RJ=FALSE)
```

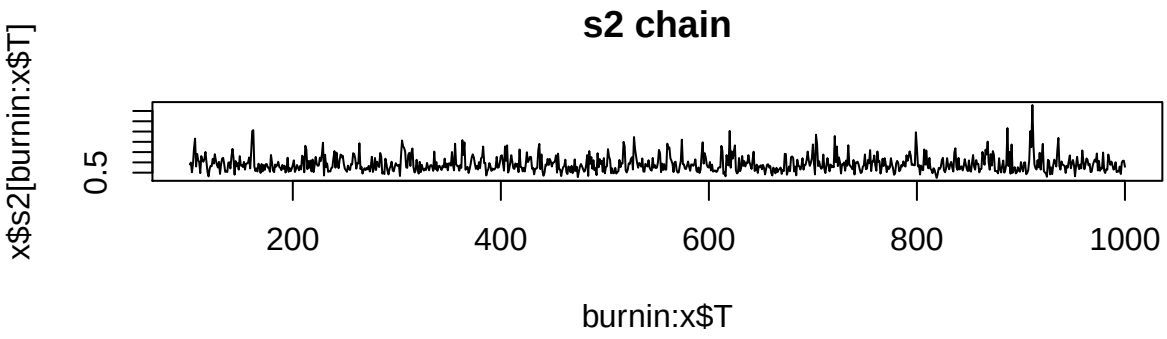
```
## t=100, m=10
## t=200, m=10
## t=300, m=10
## t=400, m=10
## t=500, m=10
## t=600, m=10
## t=700, m=10
## t=800, m=10
## t=900, m=10
```

```
plot(ridBayes, burnin=BI)  # boxplots of coefs
```

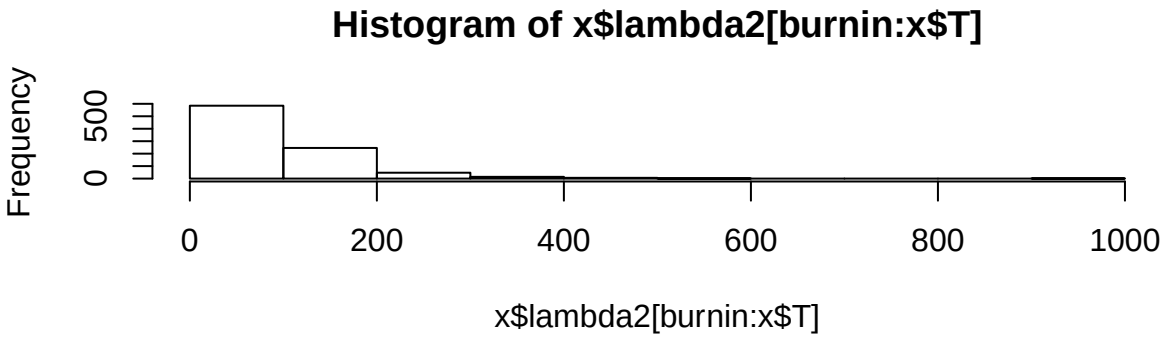


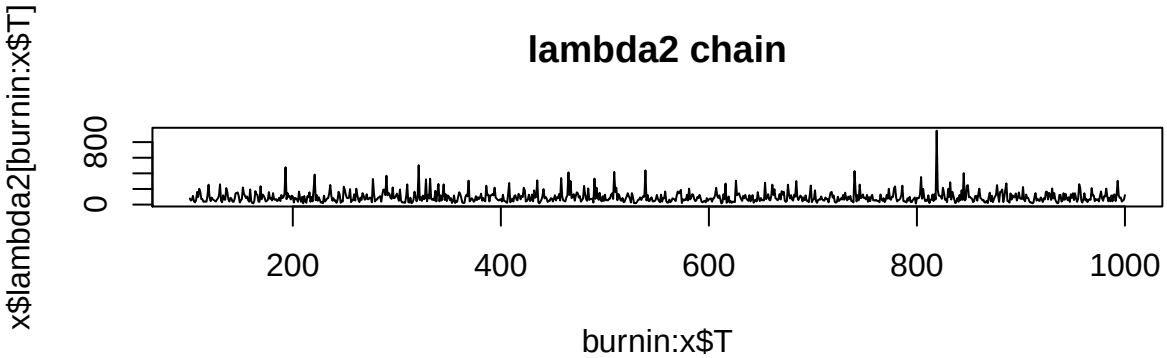

```
plot(ridBayes, burnin=BI,which='s2')  # plots for s2 (variance) chain
```



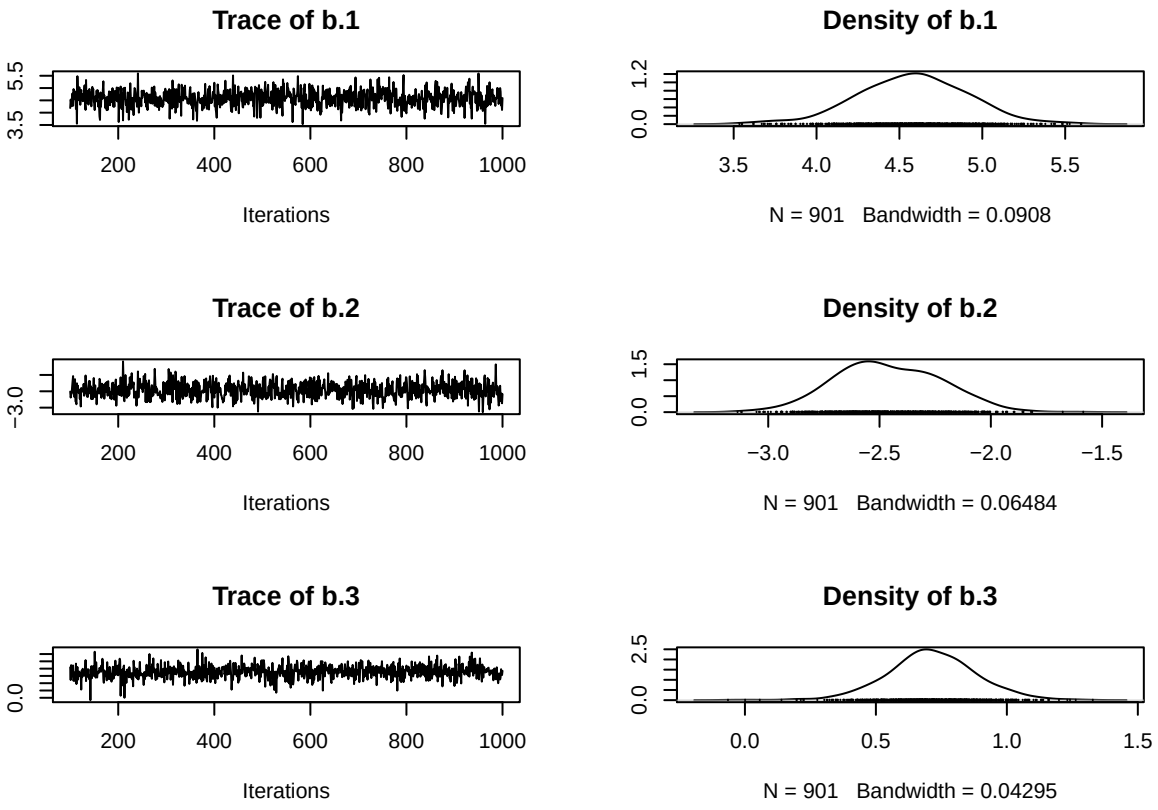



```
plot(ridBayes, burnin=BI,which='lambda2')  # plots for lambda2 chain
```

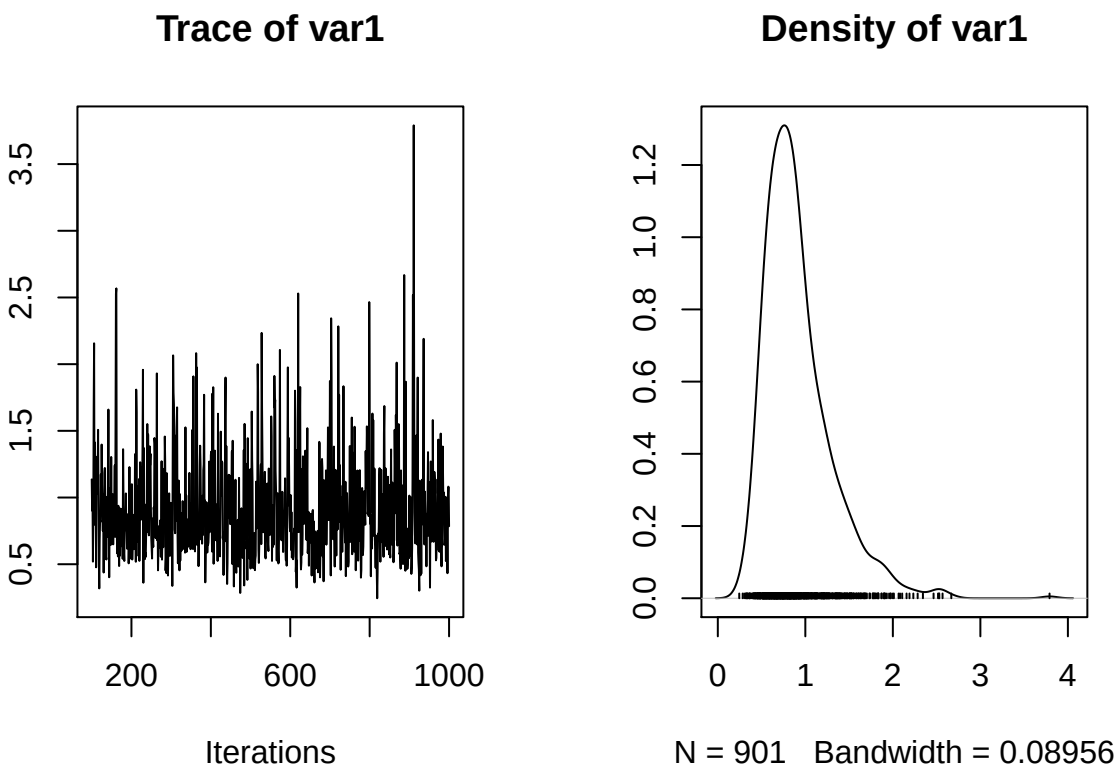




```
library(coda)

mc.ridBayes.coef <- mcmc(ridBayes$beta[BI:IT,1:3], start=BI)
plot(mc.ridBayes.coef)
```




```
mc.ridBayes.s2 <- mcmc(ridBayes$s2[BI:IT], start=BI)
plot(mc.ridBayes.s2)
```




```
mc.ridBayes.lambda2 <- mcmc(ridBayes$lambda2[BI:IT], start=BI)
plot(mc.ridBayes.lambda2)
```