Project 3

Code

Yunfan Li, Mingyu Yang, Peiqi Jin, Shichao Jia, Yanchen Chen

In your final repo, there should be an R markdown file that organizes **all computational steps** for evaluating your proposed image classification framework.

This file is currently a template for running evaluation experiments of image analysis (or any predictive modeling). You should update it according to your codes but following precisely the same structure.

Hide

Hide

```
if(!require("EBImage")){
  source("https://bioconductor.org/biocLite.R")
  biocLite("EBImage")
}
```

```
Loading required package: EBImage
```

Hide

Hide

```
if(!require("gbm")){
  install.packages("gbm")
}
```

```
Loading required package: gbm
Loaded gbm 2.1.4
```

Hide

Hide

```
library("EBImage")
library("gbm")
```

Step 0: specify directories.

Set the working directory to the image folder. Specify the training and the testing set. For data without an independent test/validation set, you need to create your own testing data by random subsampling. In order to obain reproducible results, set.seed() whenever randomization is used.

Hide

Hide

```
set.seed(2018)
#setwd("./ads_fall2018_proj3")
setwd("~/Documents/GitHub/Fall2018-Proj3-Sec1-grp6")
```

Provide directories for training images. Low-resolution (LR) image set and High-resolution (HR) image set will be in different subfolders.

Hide

Hide

```
train_dir <- "../data/train_set/" # This will be modified for different data sets.
train_LR_dir <- paste(train_dir, "LR/", sep="")
train_HR_dir <- paste(train_dir, "HR/", sep="")
train_label_path <- paste(train_dir, "label.csv", sep="")
```

Step 1: set up controls for evaluation experiments.

In this chunk, we have a set of controls for the evaluation experiments.

- (T/F) cross-validation on the training set
- (number) K, the number of CV folds
- (T/F) process features for training set
- (T/F) run evaluation on an independent test set
- (T/F) process features for test set

Hide

Hide

```
run.cv=TRUE # run cross-validation on the training set
K <- 3  # number of CV folds
run.feature.train=TRUE # process features for training set
run.test=TRUE # run evaluation on an independent test set
run.feature.test=TRUE # process features for test set
```

Using cross-validation or independent test set evaluation, we compare the performance of models with different specifications. In this example, we use GBM with different `depth`. In the following chunk, we list, in a vector, setups (in this case, `depth`) corresponding to models that we will compare. In your project, you might compare very different classifiers. You can assign them numerical IDs and labels specific to your project.

Hide

Hide

```
#model_values <- list(depth = c(3, 5, 7), n.trees = c(150, 200))
#model_values <- expand.grid(depth = c(3,5,7), n.trees = c(150, 200))
model_values <- expand.grid(depth = c(11,13), n.trees = c(250,300))
model_values_xgb <- expand.grid(max_depth = c(4,6,8), eta = c(0.1,0.3,0.5), subsample
= c(0.5,0.8), min_child_weight = c(4,6,8))
#model_labels = paste("GBM with depth =", model_values)
```

Step 2: import training images class labels.

We provide extra information of image label: car (0), flower (1), market (2). These labels are not necessary for your model.

Hide

Hide

```
extra_label <- read.csv(train_label_path, colClasses=c("NULL", NA, NA))
```

```
cannot open file './data/train_set/label.csv': No such file or directoryError in file
(file, "rt") : cannot open the connection
```

Step 3: construct features and responses

`feature.R` should be the wrapper for all your feature engineering functions and options. The function `feature( )` should have options that correspond to different scenarios for your project and produces an R object that contains features and responses that are required by all the models you are going to evaluate later. `+ feature.R +` Input: a path for low-resolution images. + Input: a path for high-resolution images. + Output: an RData file that contains extracted features and corresponding responses

Hide

Hide

```
#setwd("~/Documents/GitHub/Fall2018-Proj3-Sec1-grp6")
source("../lib/feature.R")
tm_feature_train <- NA
if(run.feature.train){
  tm_feature_train <- system.time(dat_train <- feature(train_LR_dir, train_HR_dir))
  feat_train <- dat_train$feature
  label_train <- dat_train$label
}
#save(dat_train, file="./output/feature_train.RData")
```

Step 4: Train a classification model with training images

Call the train model and test model from library.

Model1: GBM

`train.R` and `test.R` should be wrappers for all your model training steps and your classification/prediction steps. + `train.R` + Input: a path that points to the training set features and responses. + Output: an RData file that contains trained classifiers in the forms of R objects: models/settings/links to external trained configurations. + `test.R` + Input: a path that points to the test set features. + Input: an R object that contains a trained classifier. + Output: an R object of response predictions on the test set. If there are multiple classifiers under evaluation, there should be multiple sets of label predictions.

Hide

Hide

```
source("../lib/train.R")
source("../lib/test.R")
source("../lib/train.xgboost.R")
```

Model selection with cross-validation (GBM)

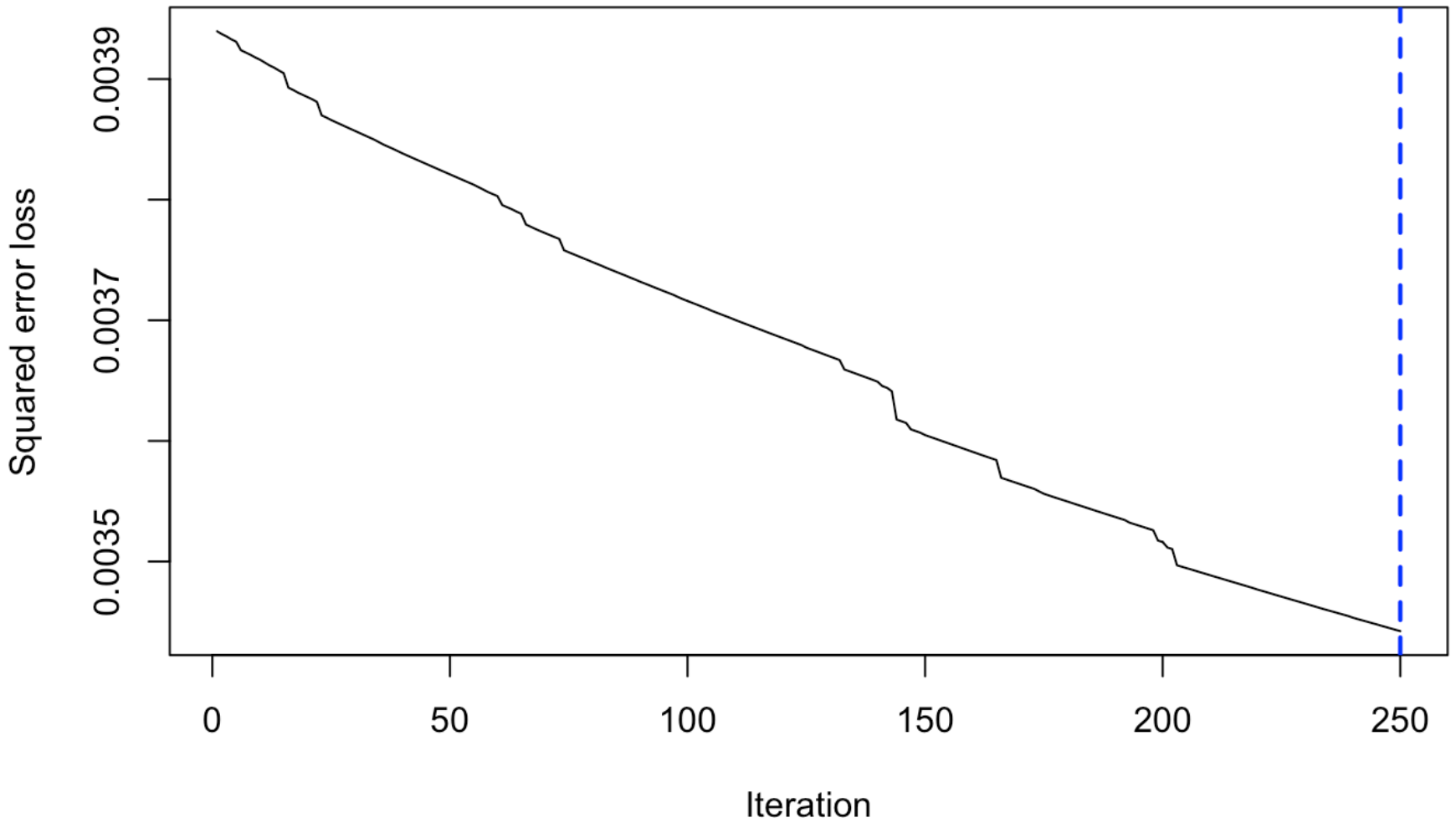
- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

Hide

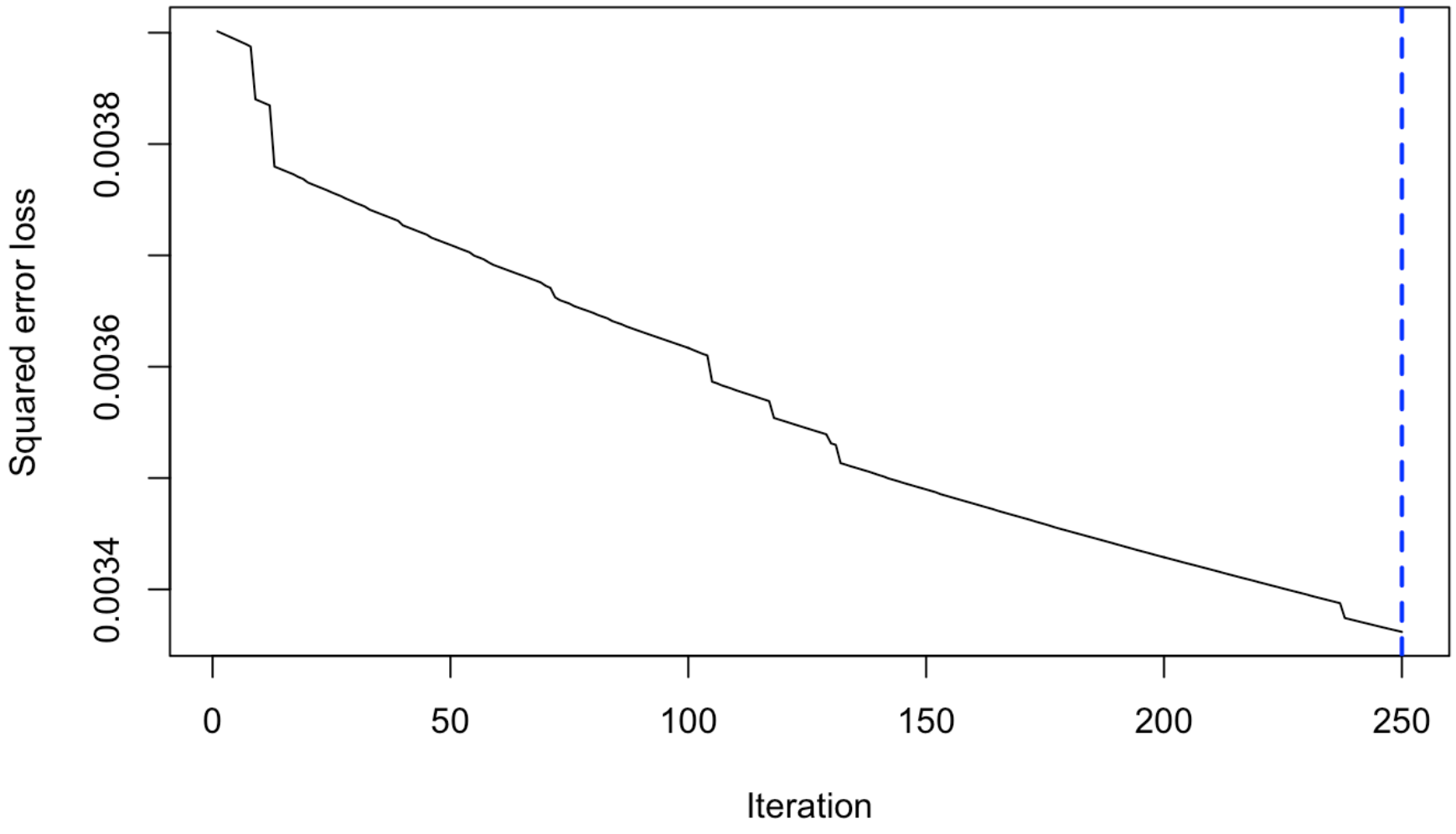
Hide

```
source("../lib/cross_validation.R")
if(run.cv){
  err_cv <- array(dim=c(nrow(model_values), 2))
  for(k in 1:nrow(model_values)){
    cat("k=", k, "\n")
    err_cv[k,] <- cv.function(feat_train, label_train, model_values[k, ], K)
  }
  save(err_cv, file="../output/err_cv.RData")
}
```

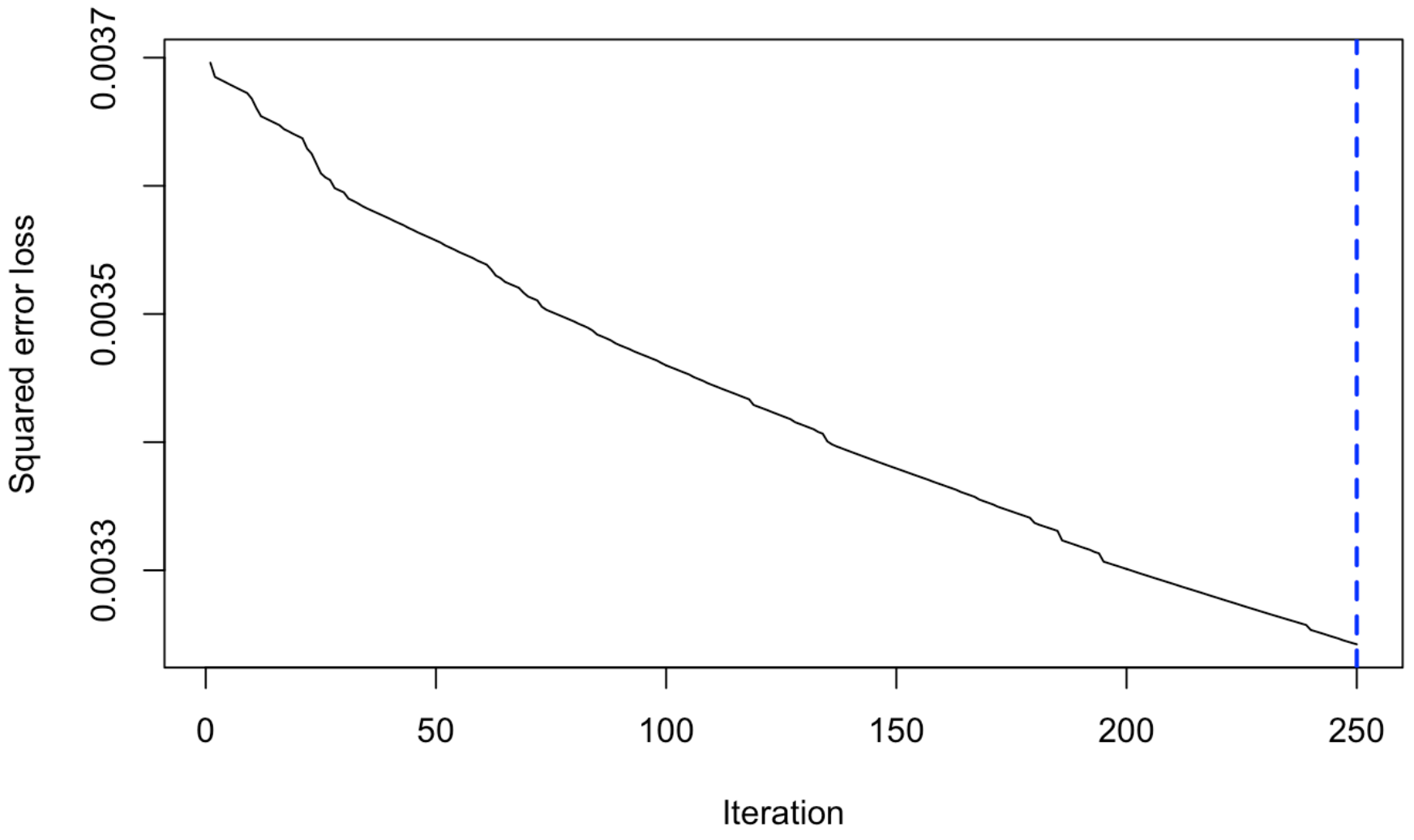
```
k= 1
```


Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration

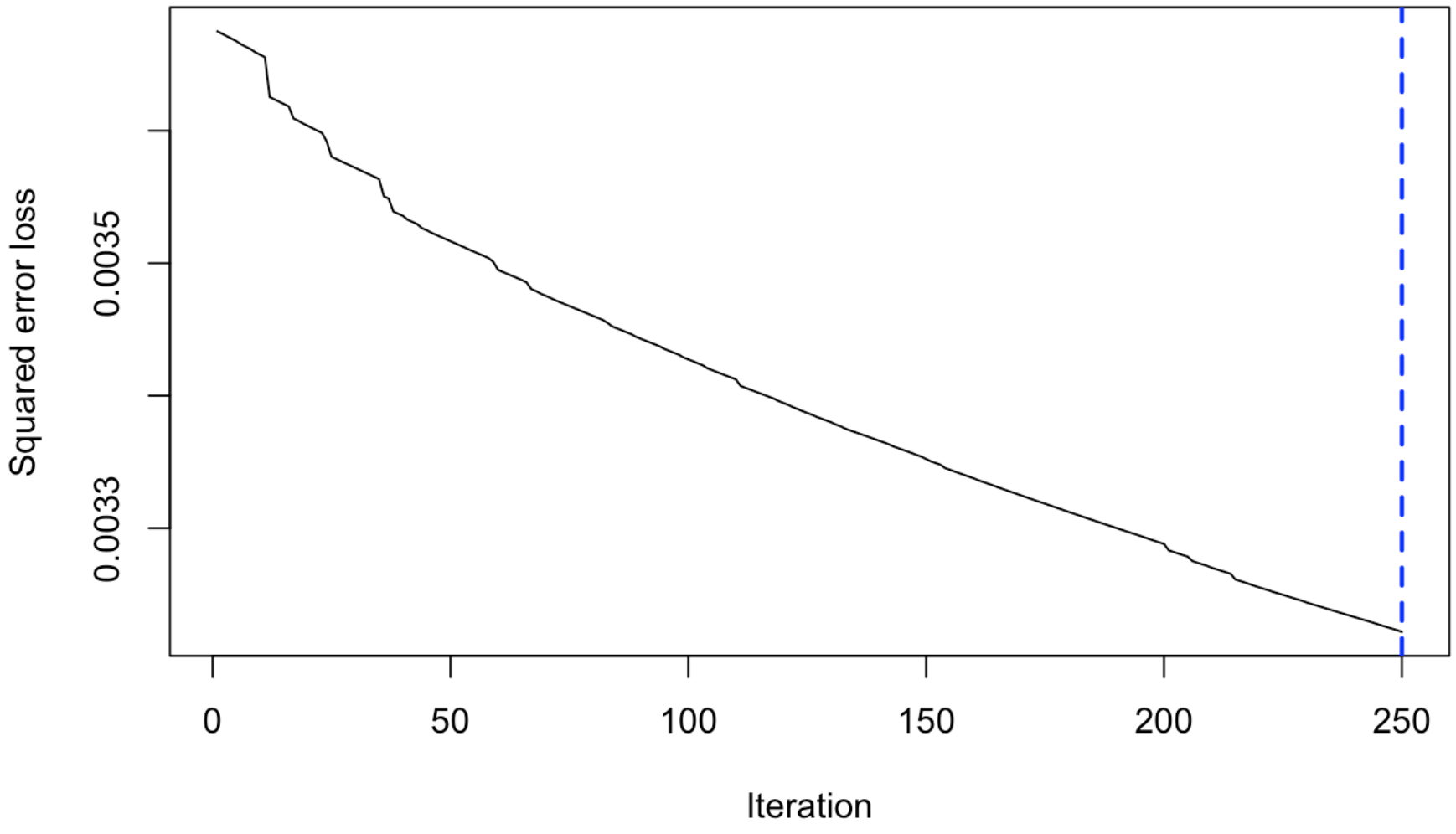
Squared error loss
0.0037
0.0035
0.0033
0
50
100
150
200
250
Iteration

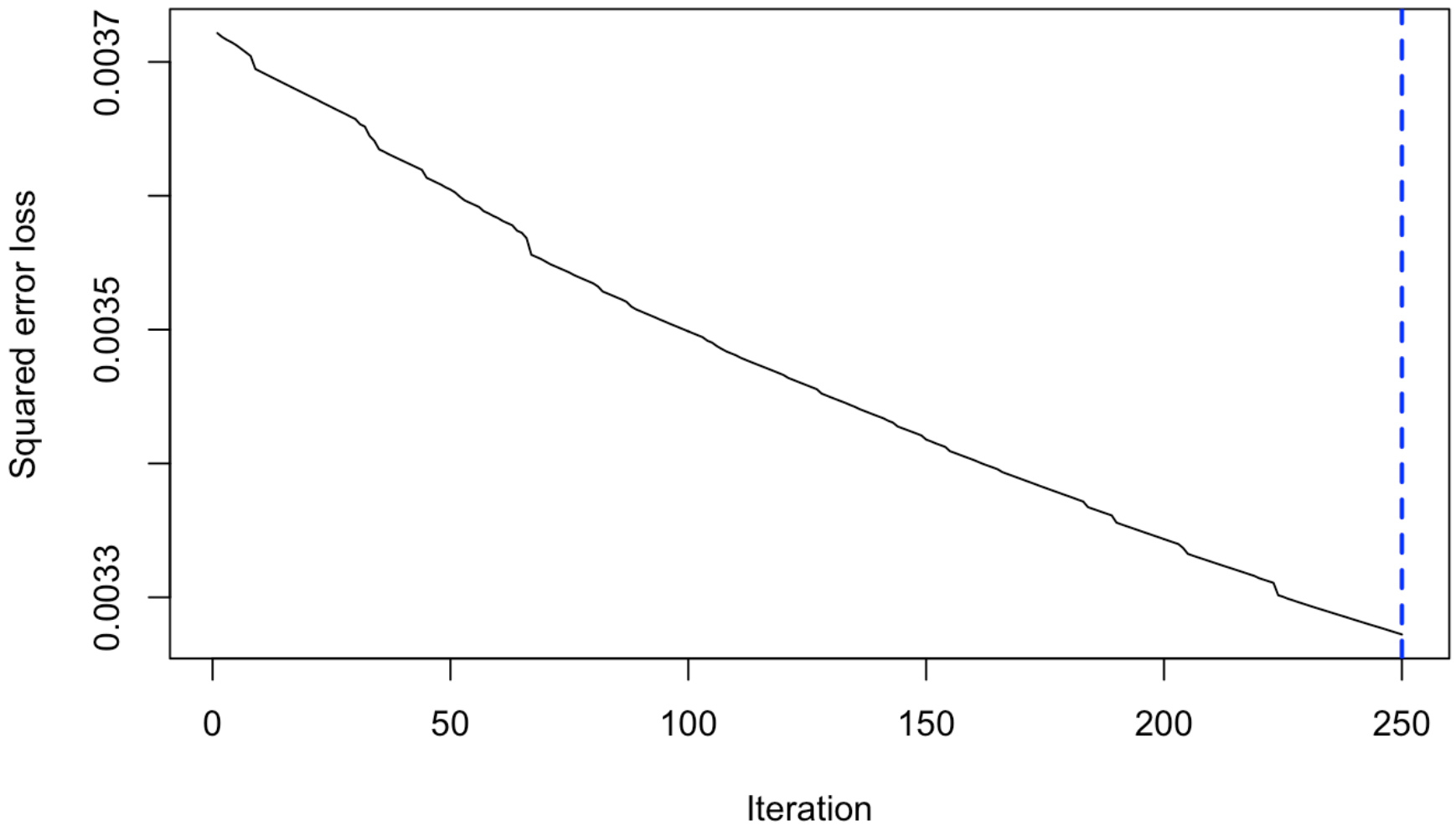

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

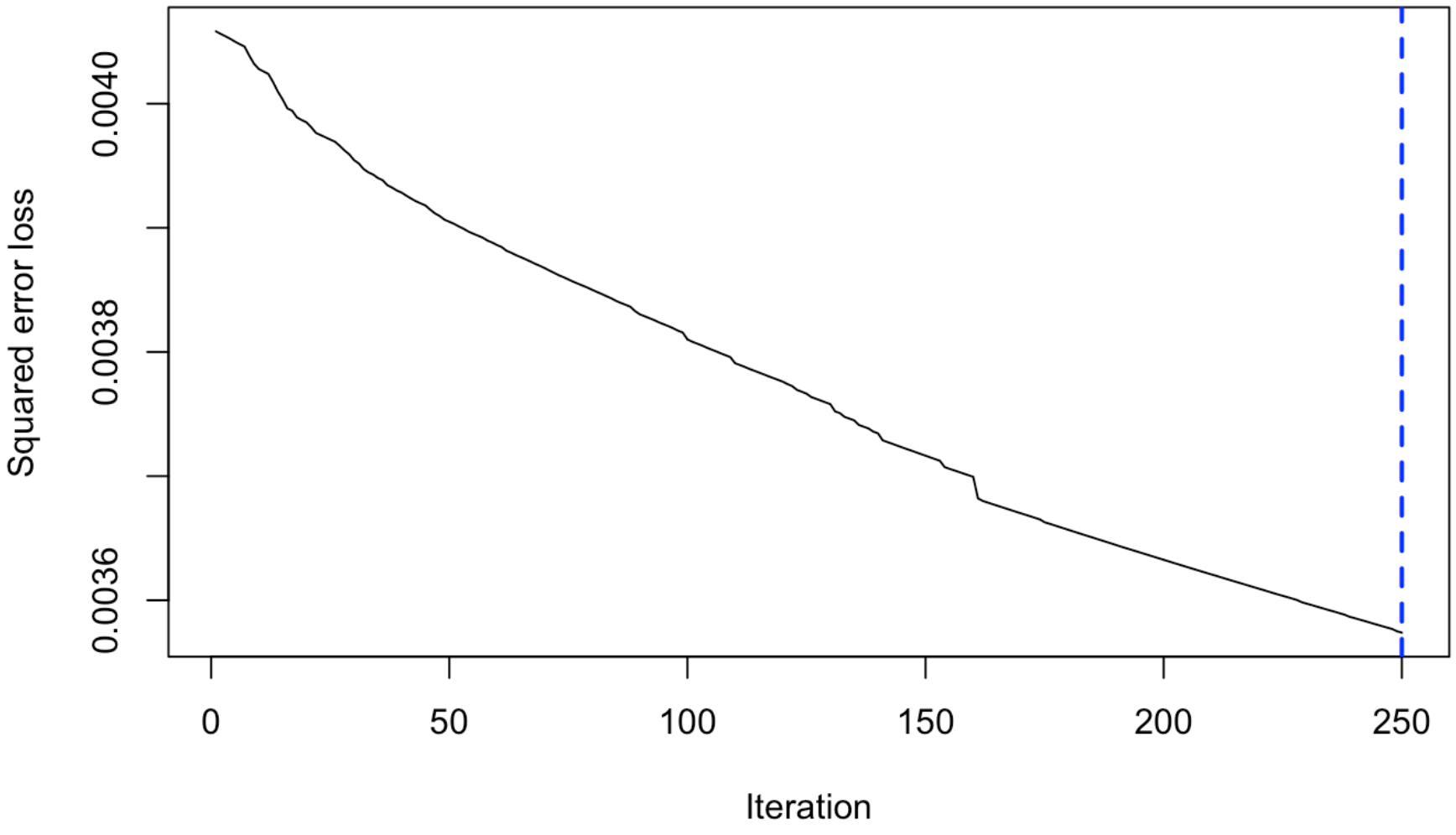
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

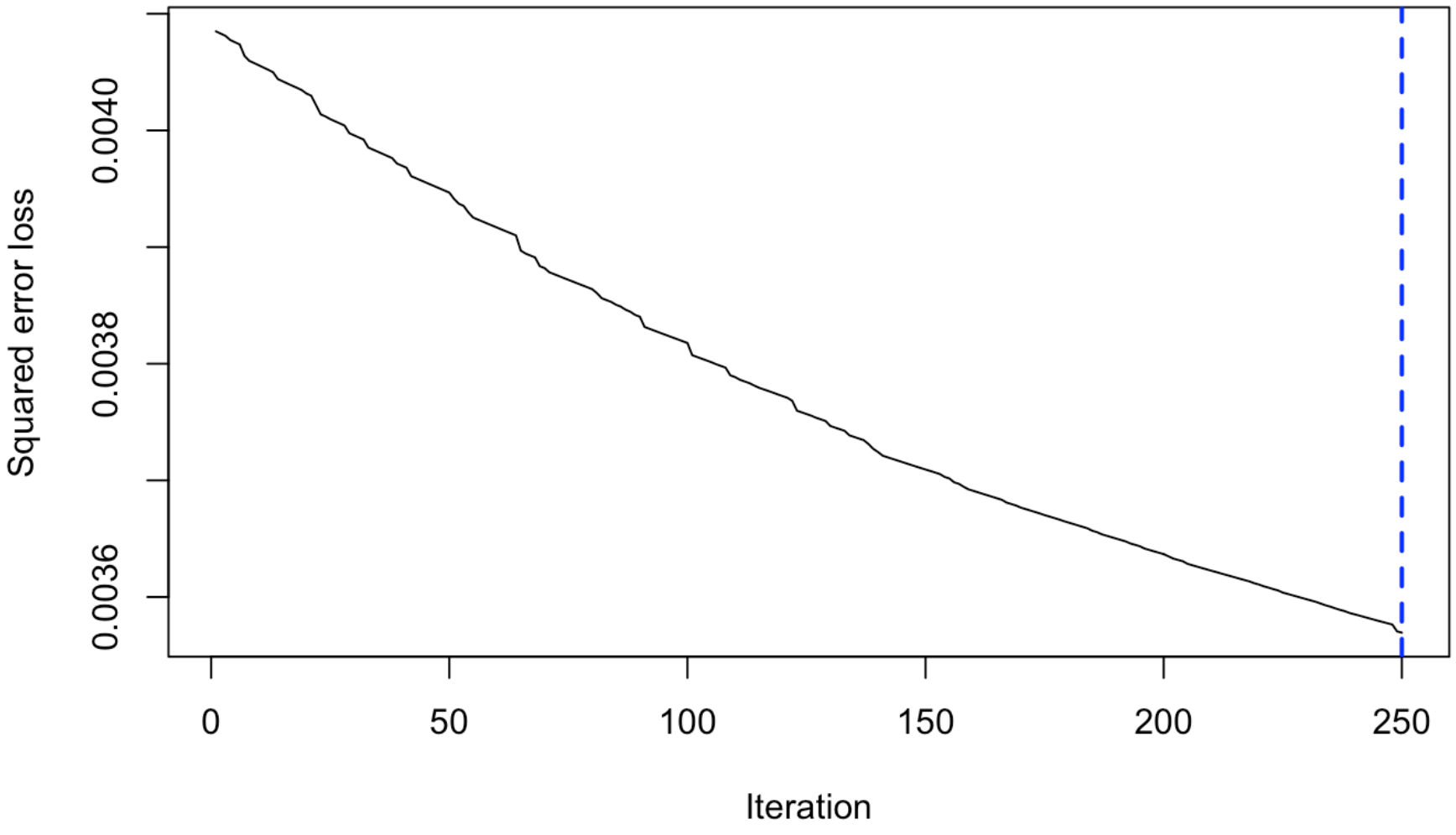
Squared error loss
0.0032
0.0033
0.0034
0.0035
0
50
100
150
200
250
Iteration

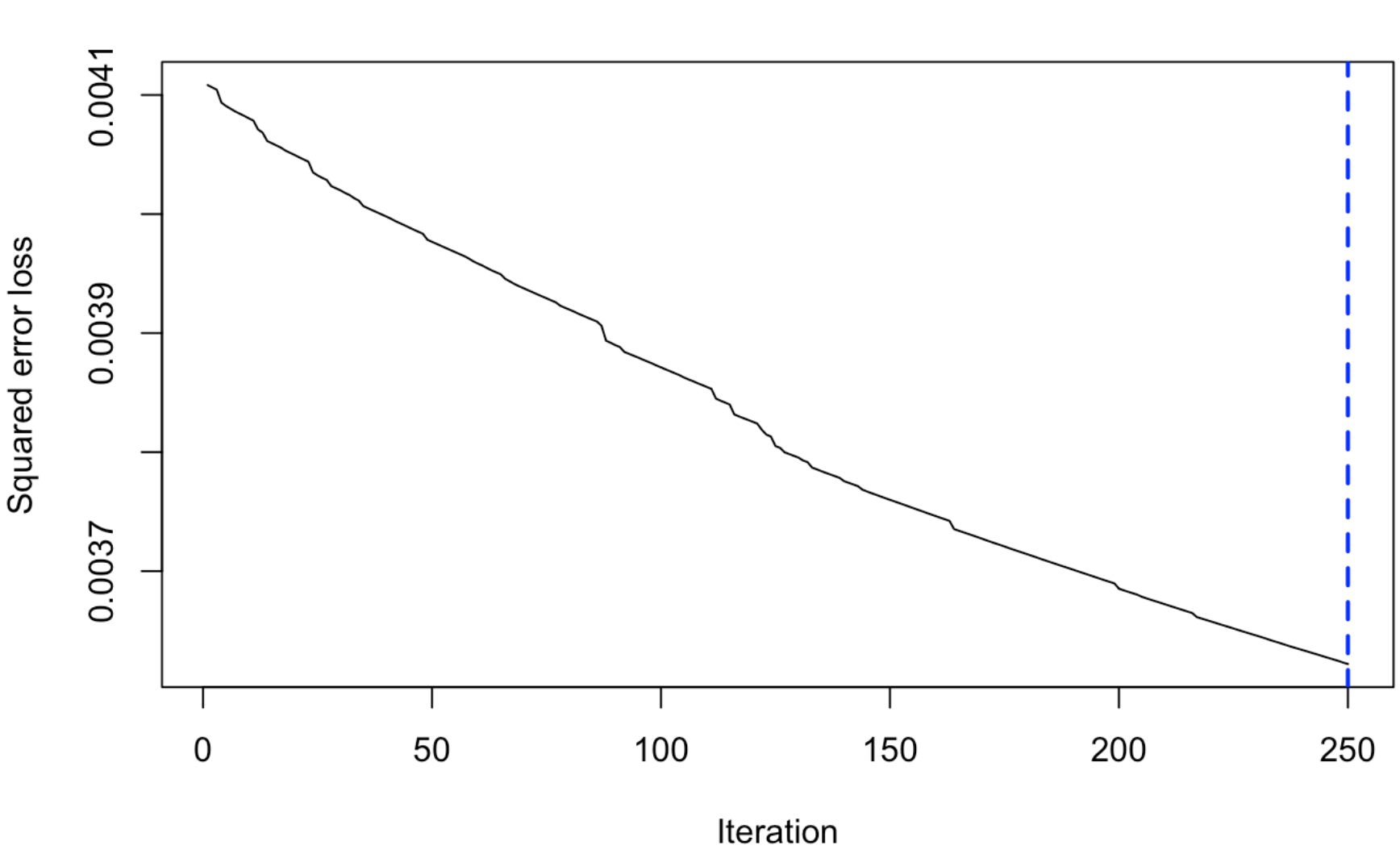

Squared error loss
0.0033
0.0035
0
50
100
150
200
250
Iteration

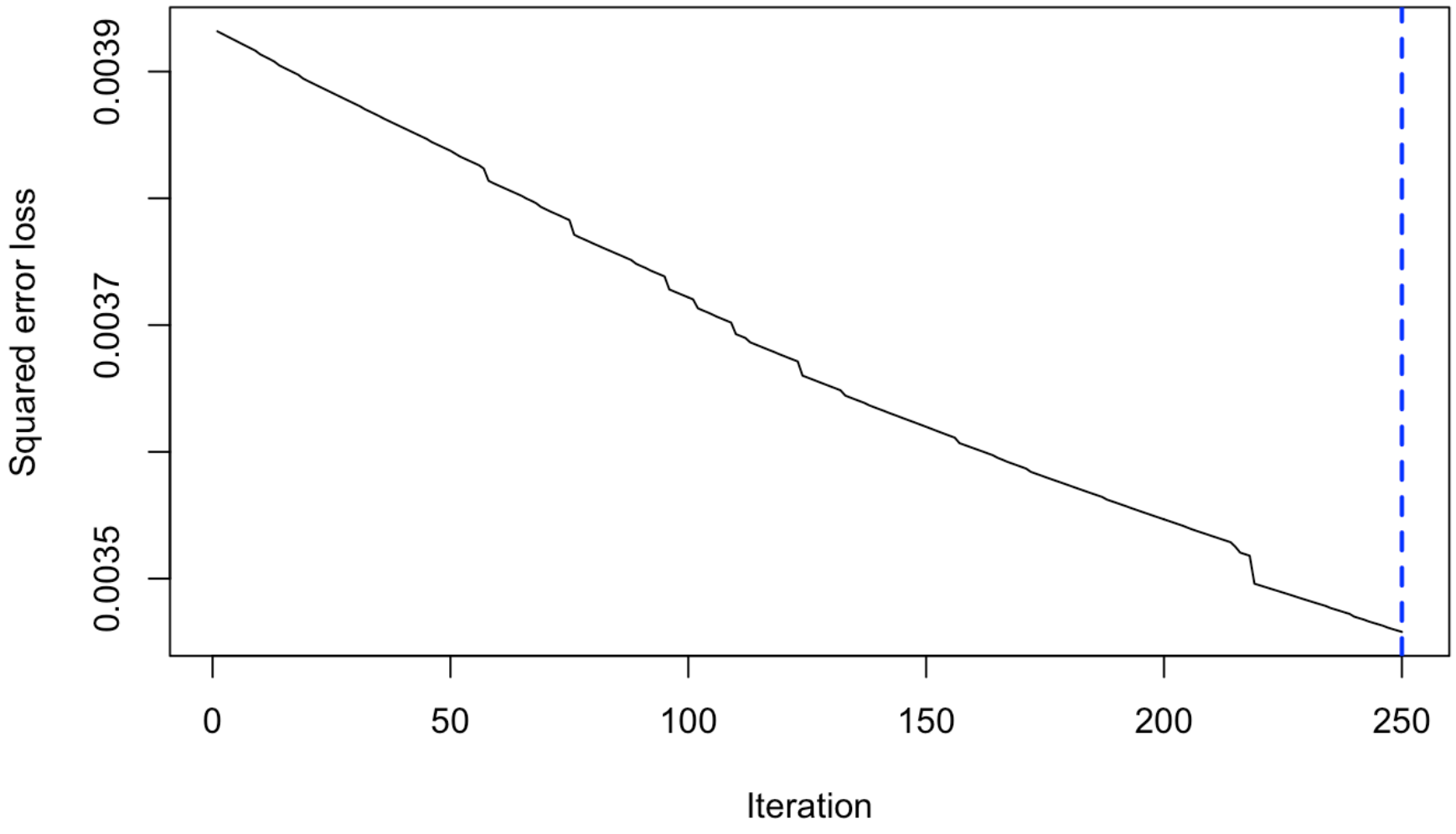

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

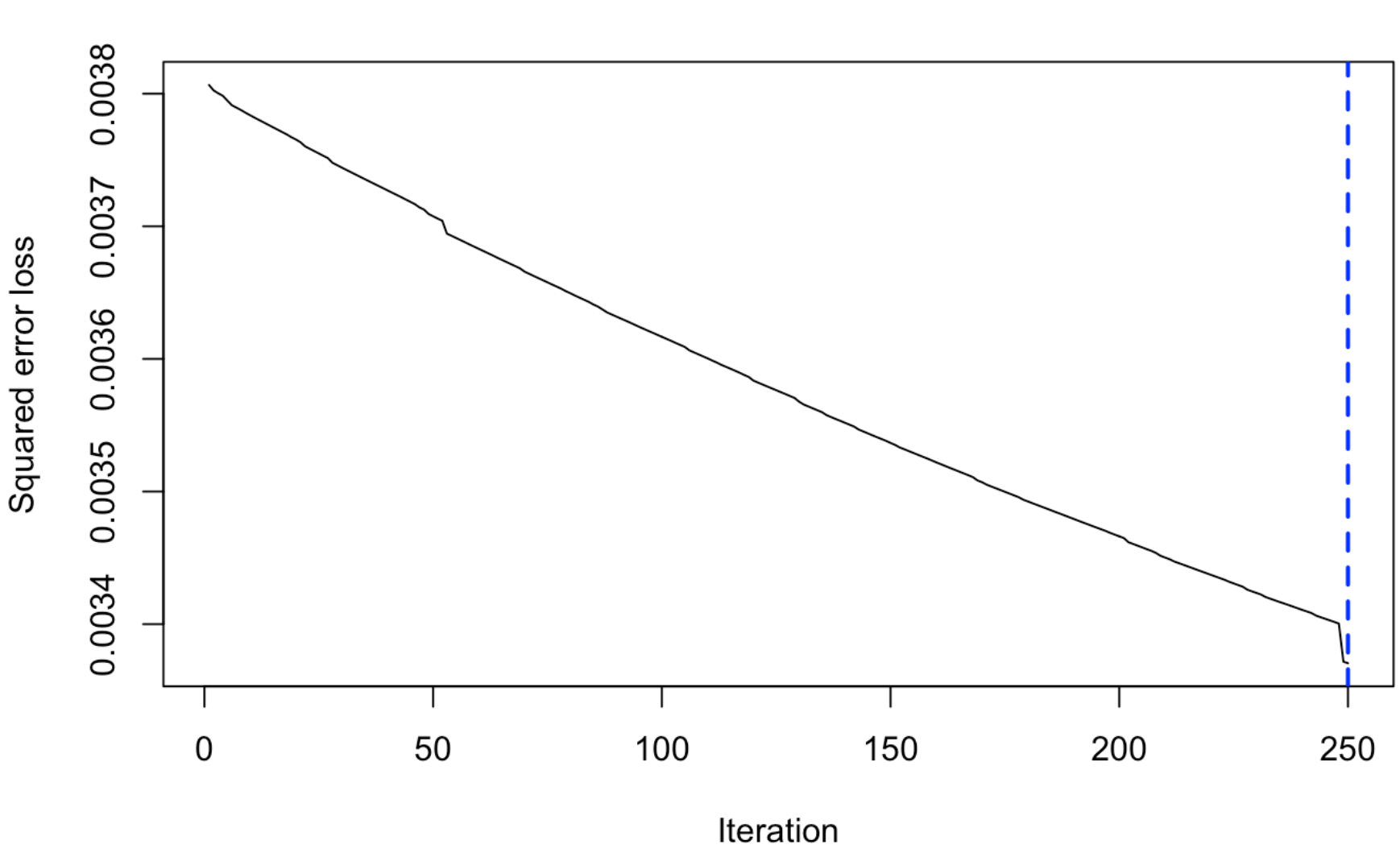

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

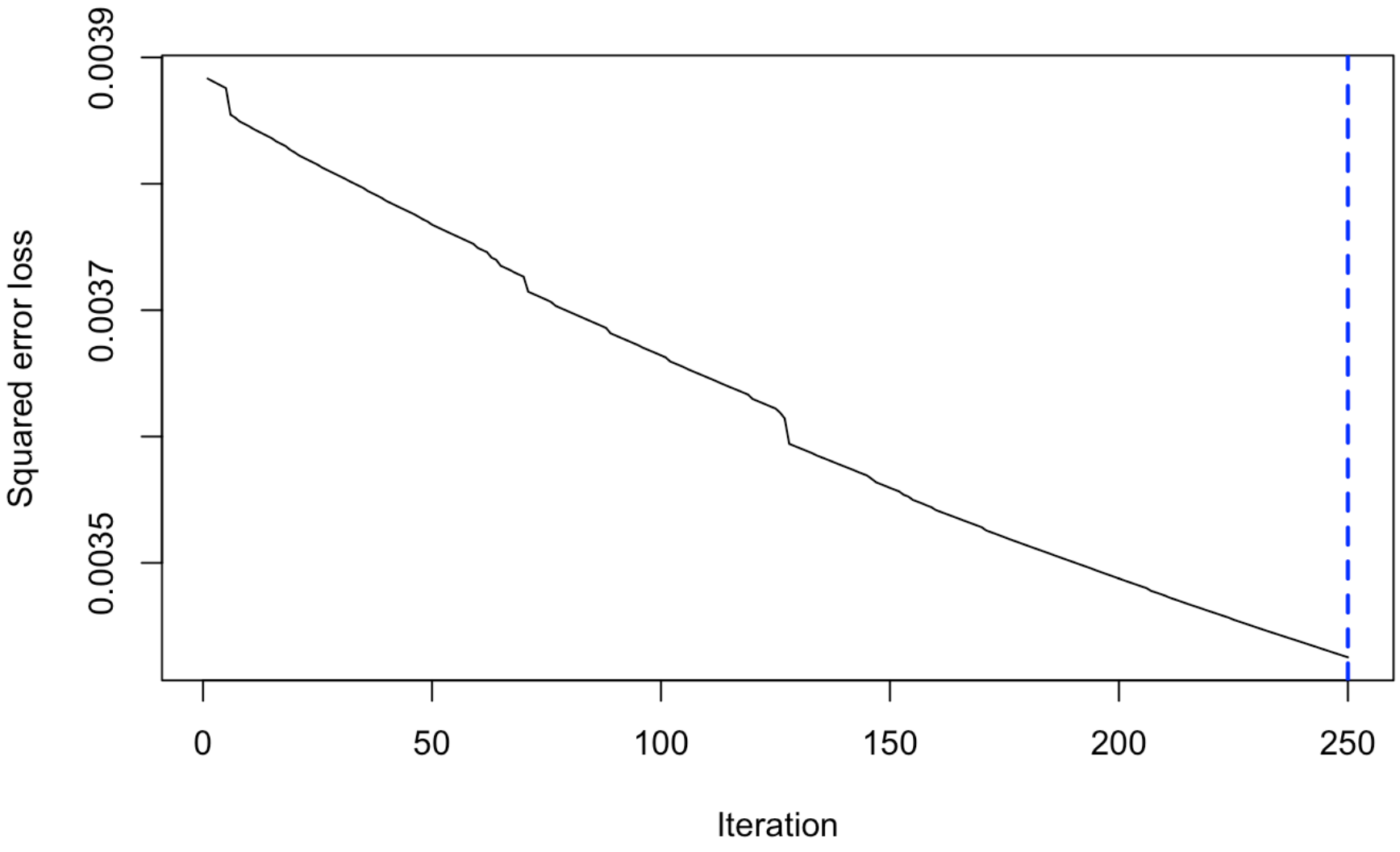
Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration


Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
Iteration

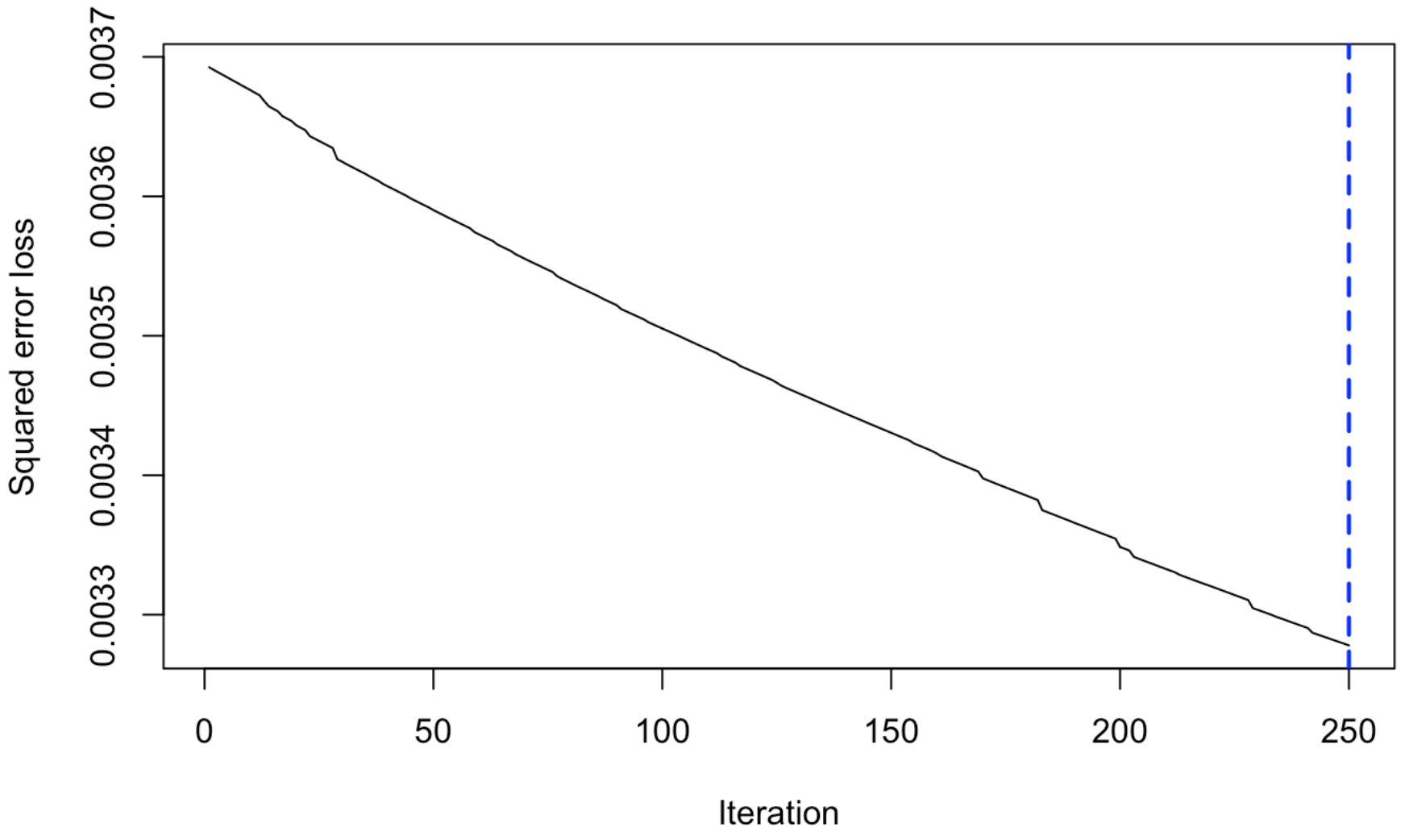

Squared error loss
0.0037
0.0039
0.0041
0
50
100
150
200
250
Iteration

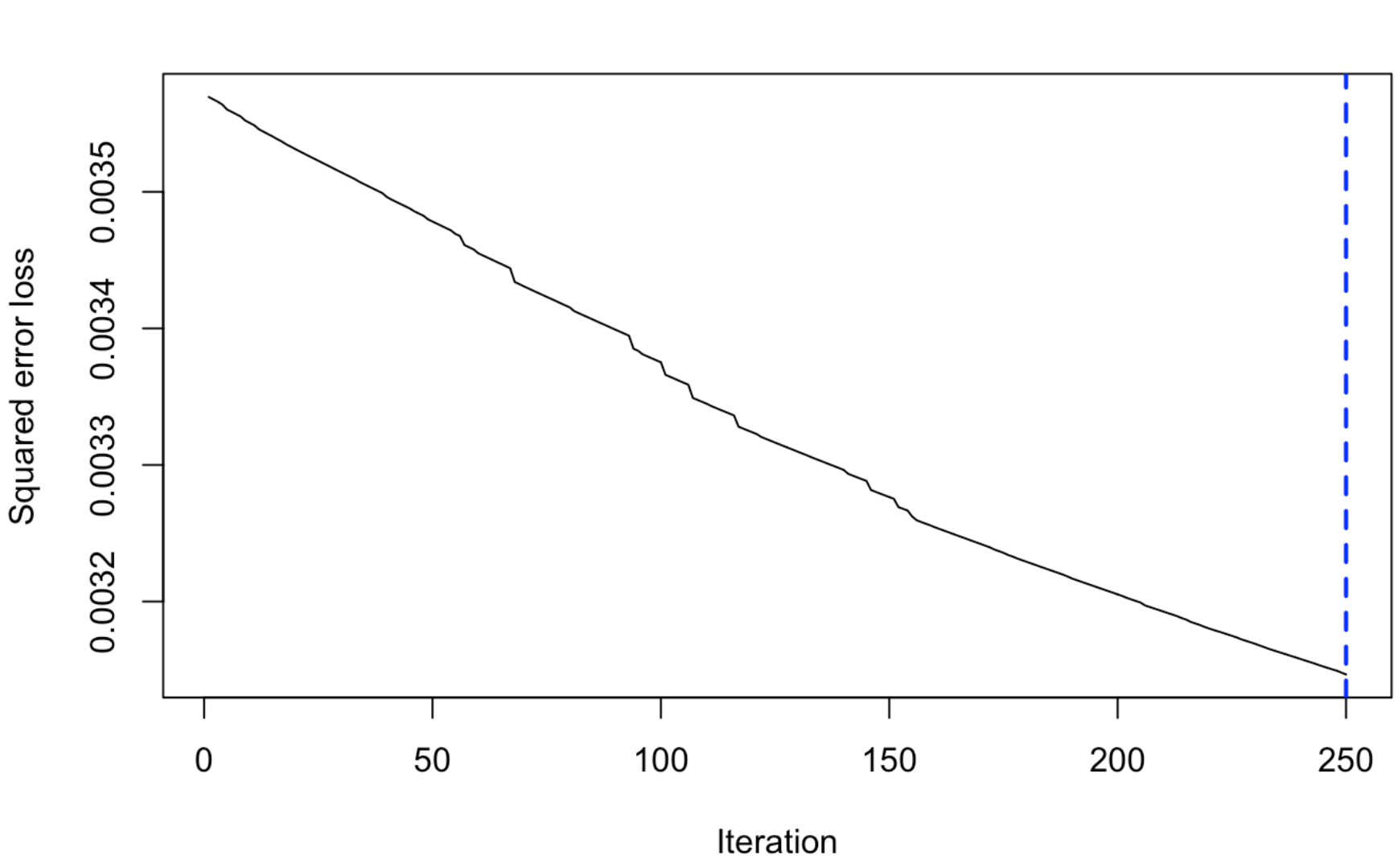

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration

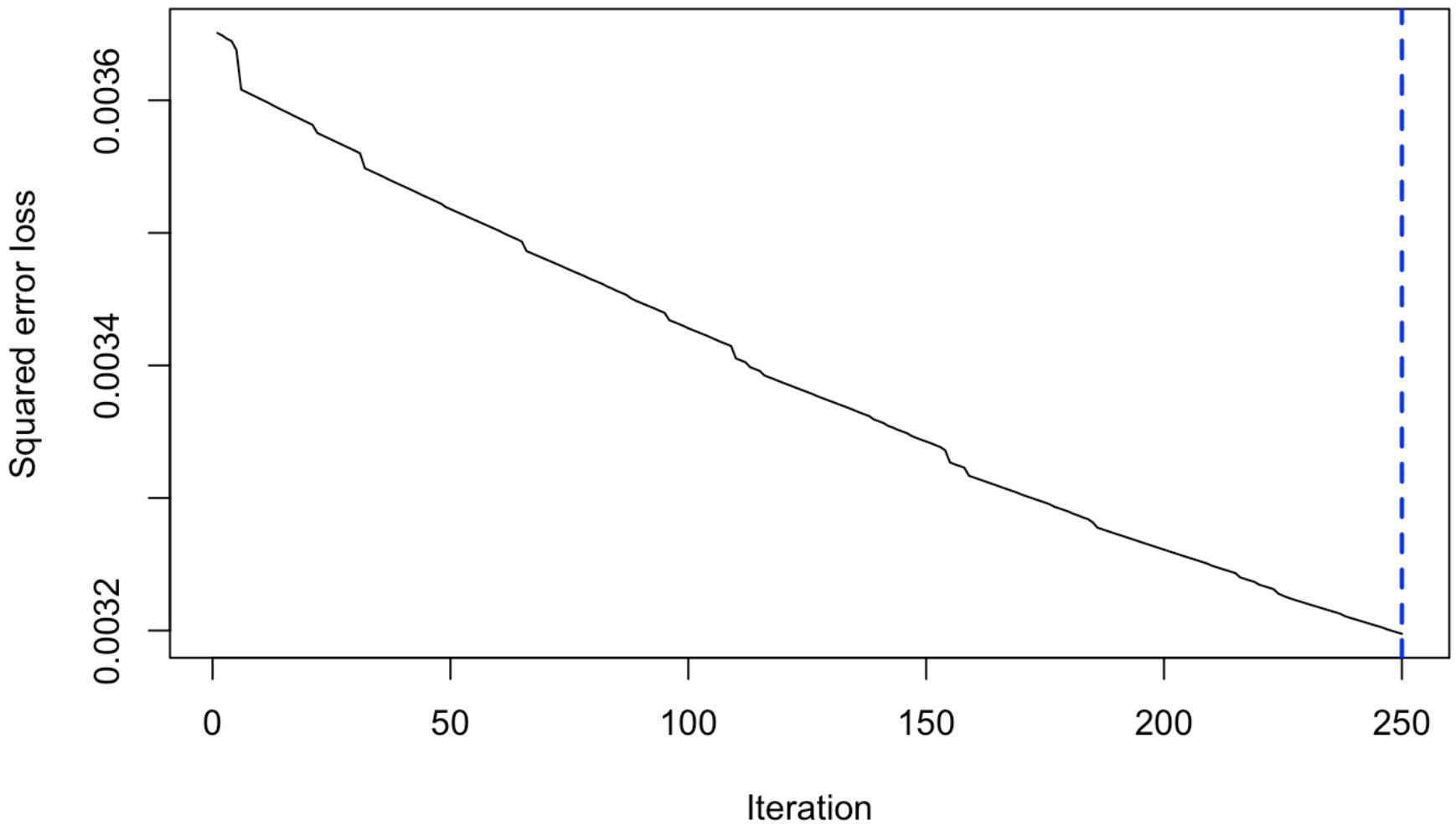

Squared error loss
0.0034
0.0035
0.0036
0.0037
0.0038
0
50
100
150
200
250
Iteration

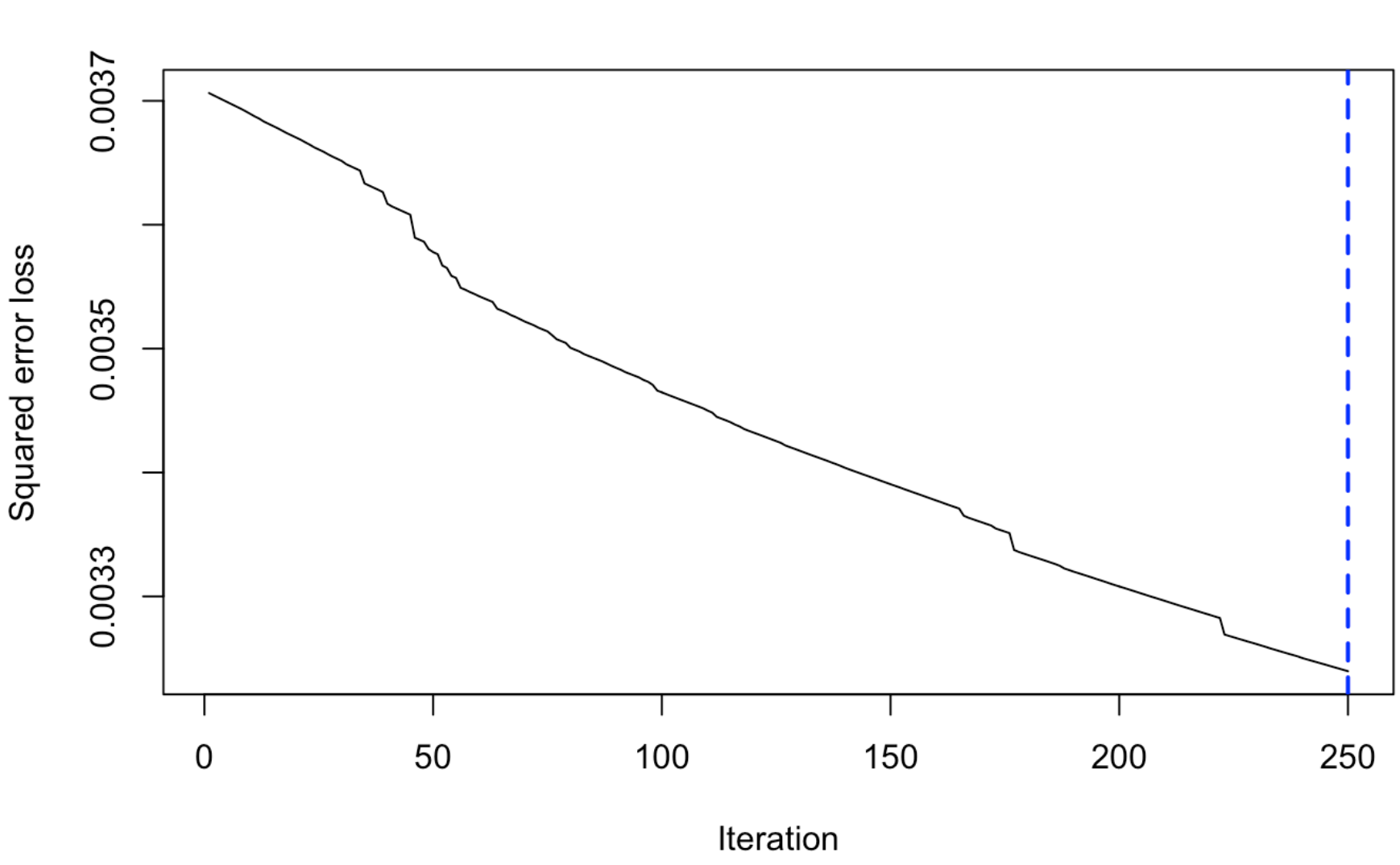

Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration

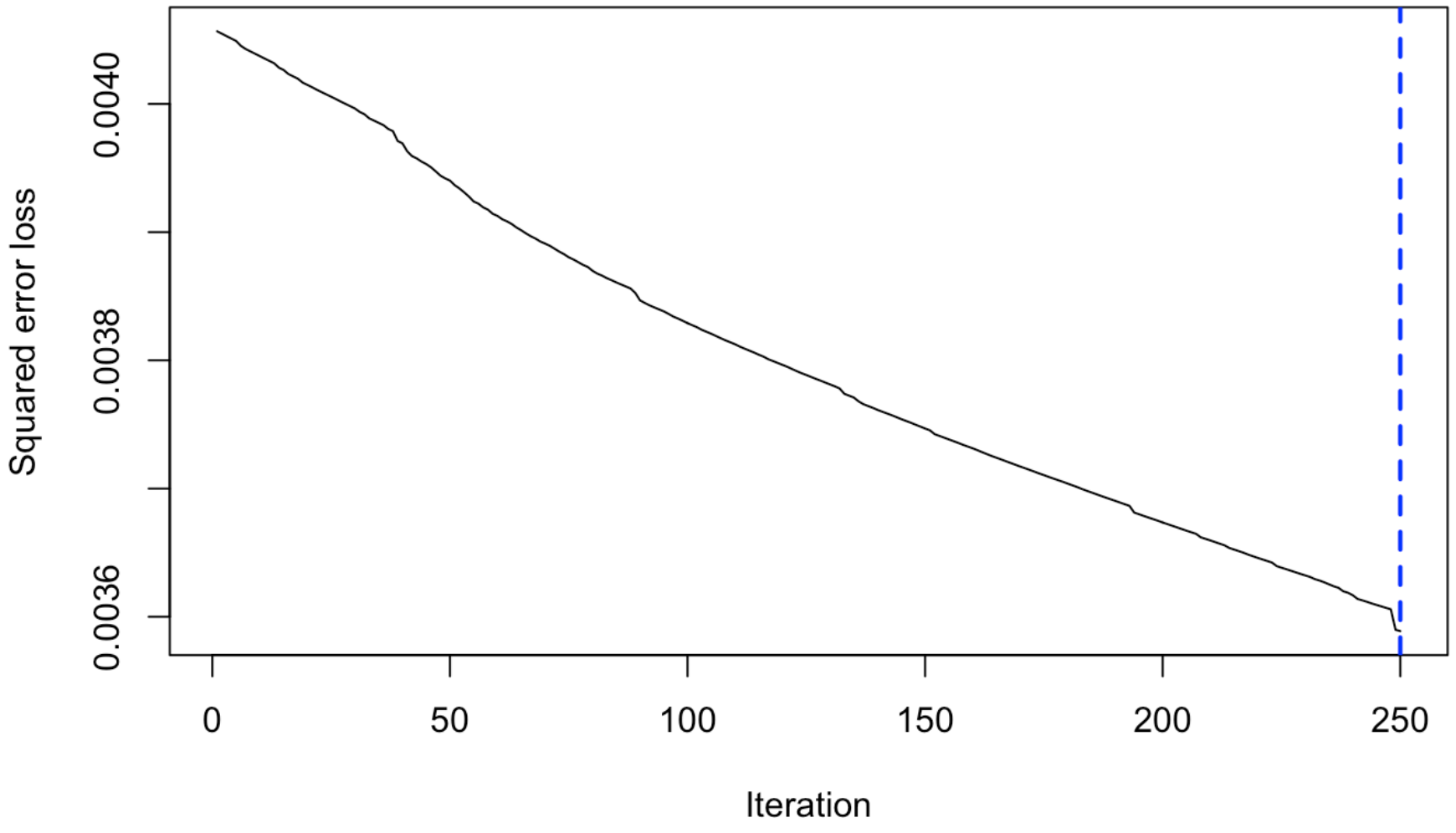
Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0034
0.0035
0.0036
0.0037
0
50
100
150
200
250
Iteration

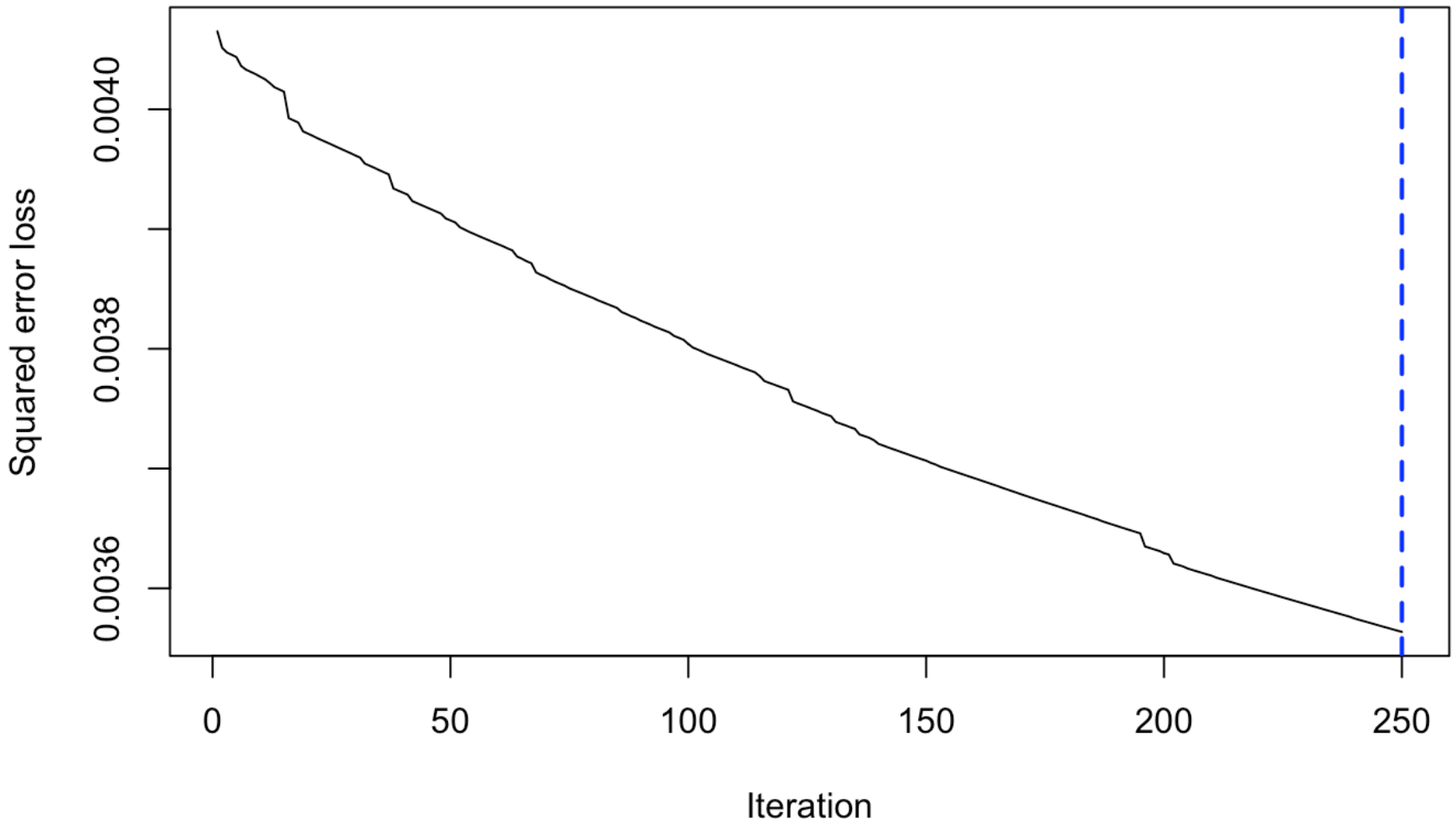

Squared error loss
0.0032
0.0033
0.0034
0.0035
0
50
100
150
200
250
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
Iteration

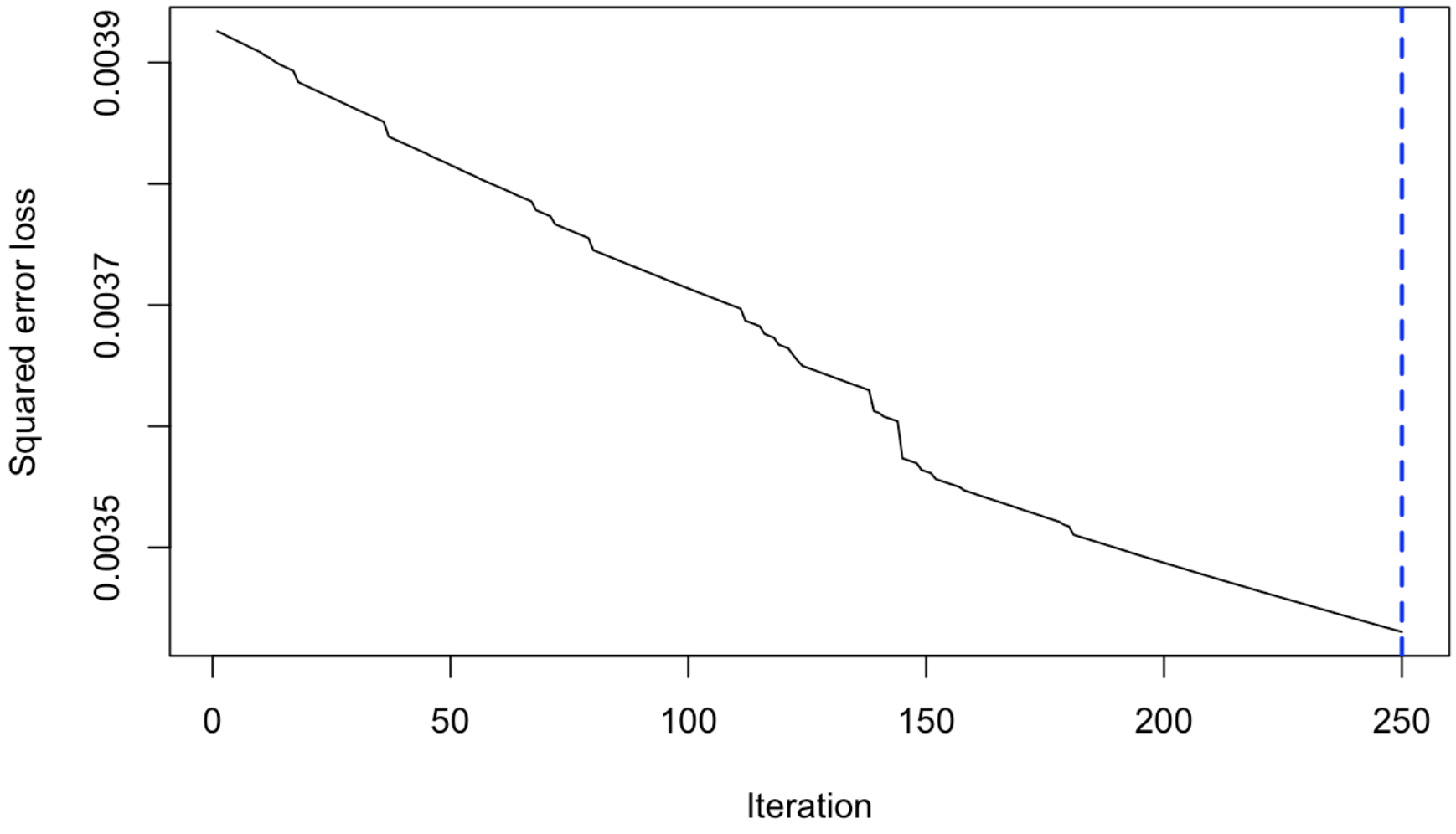

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

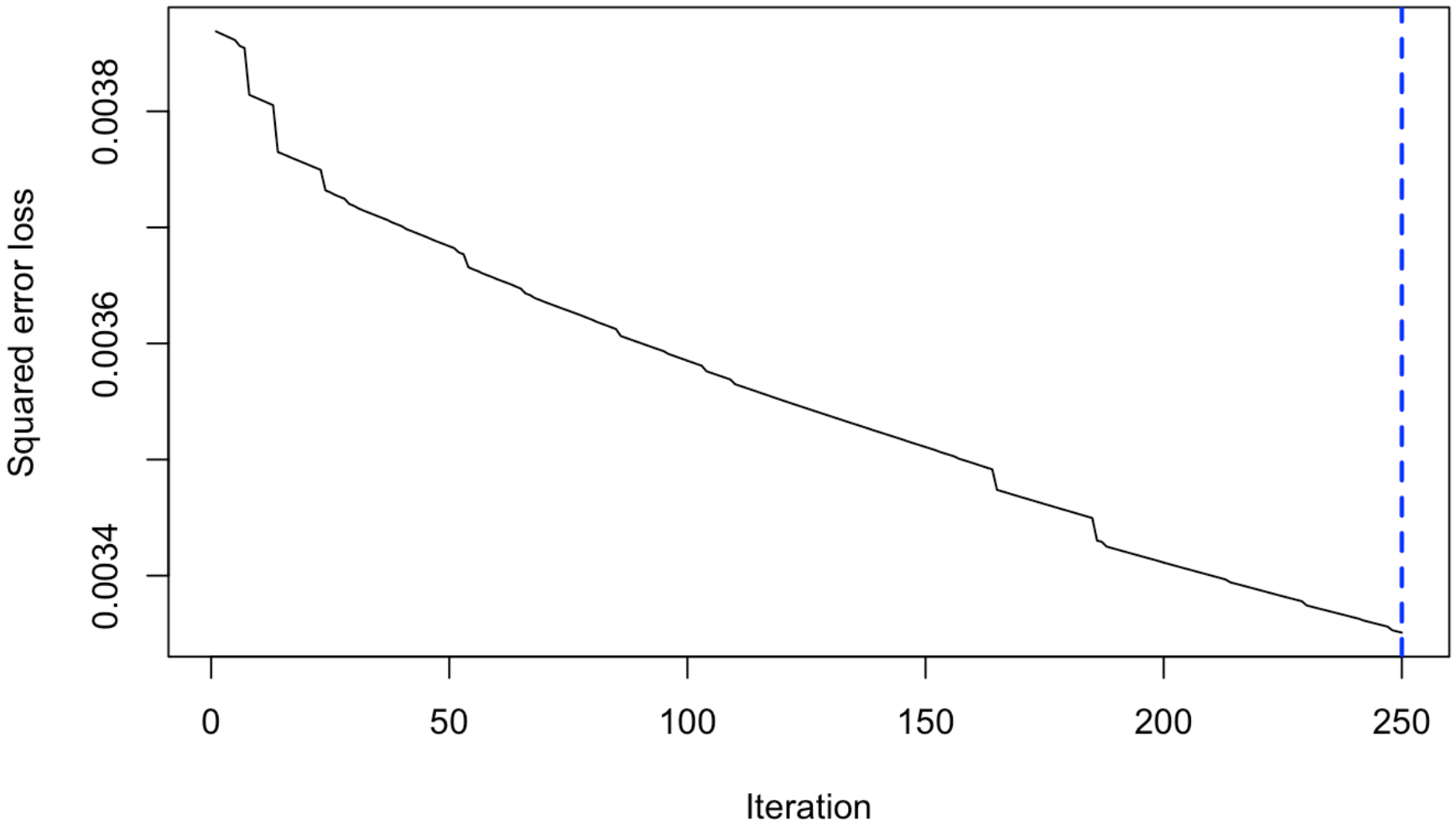
Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

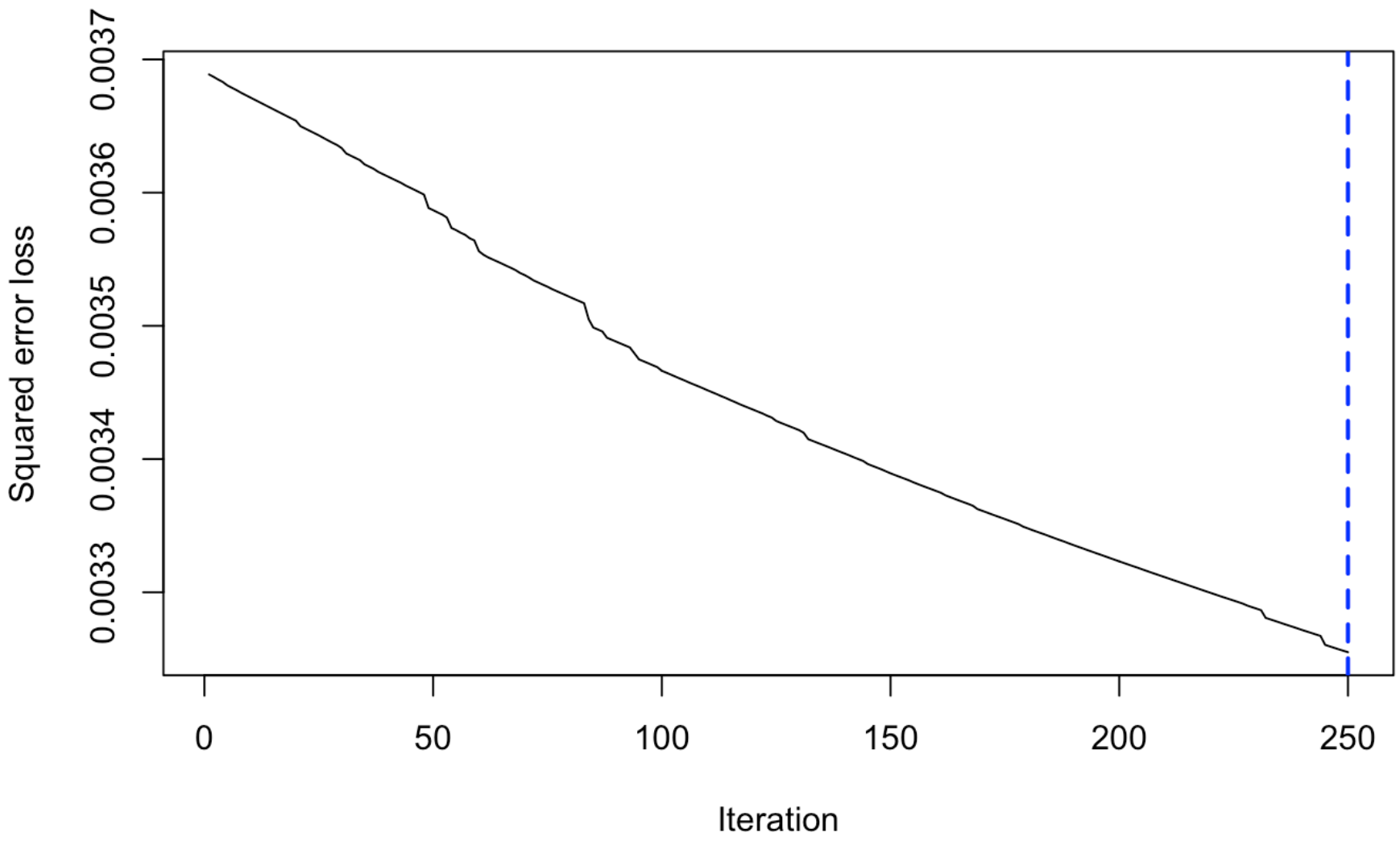
Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

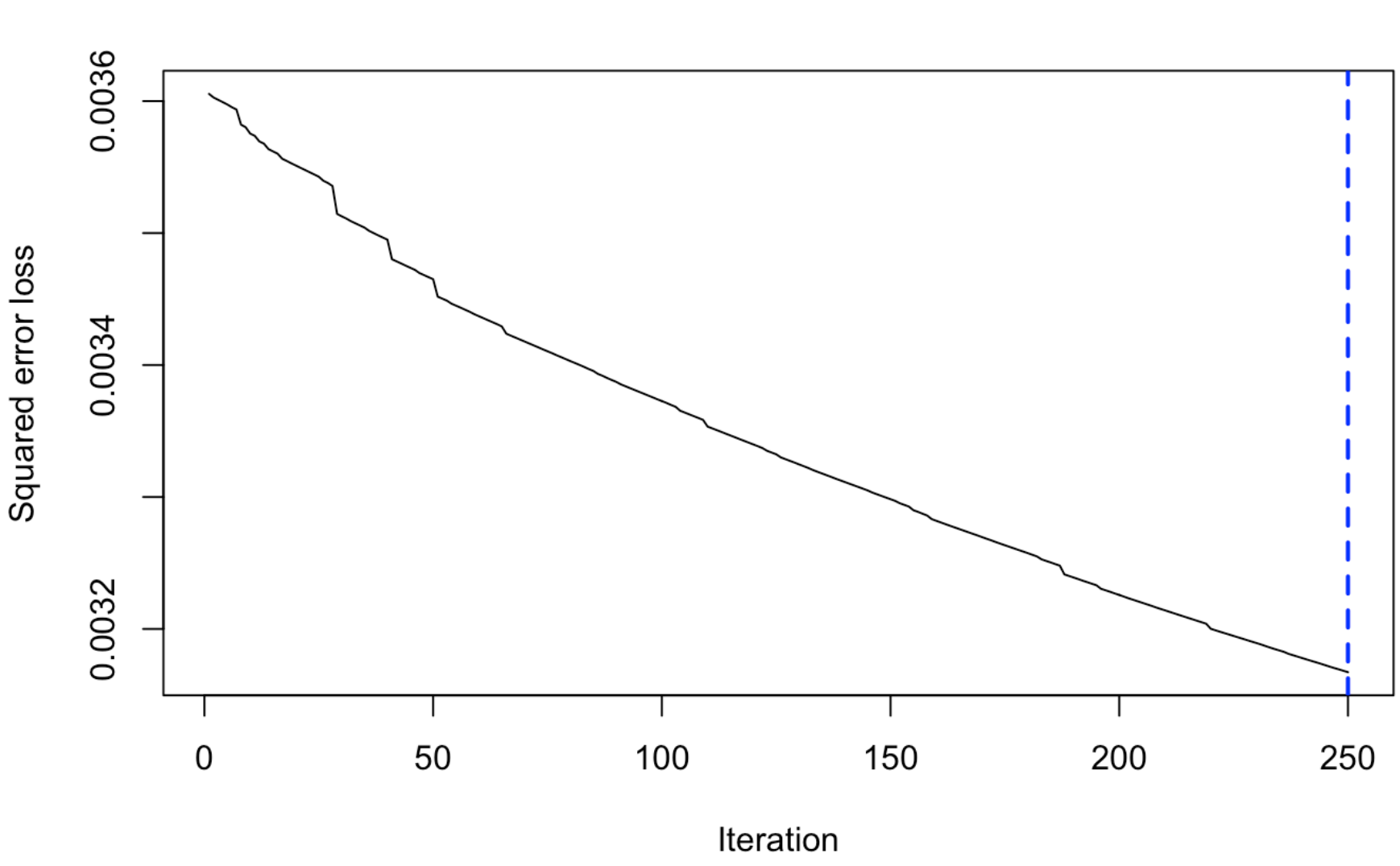

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration

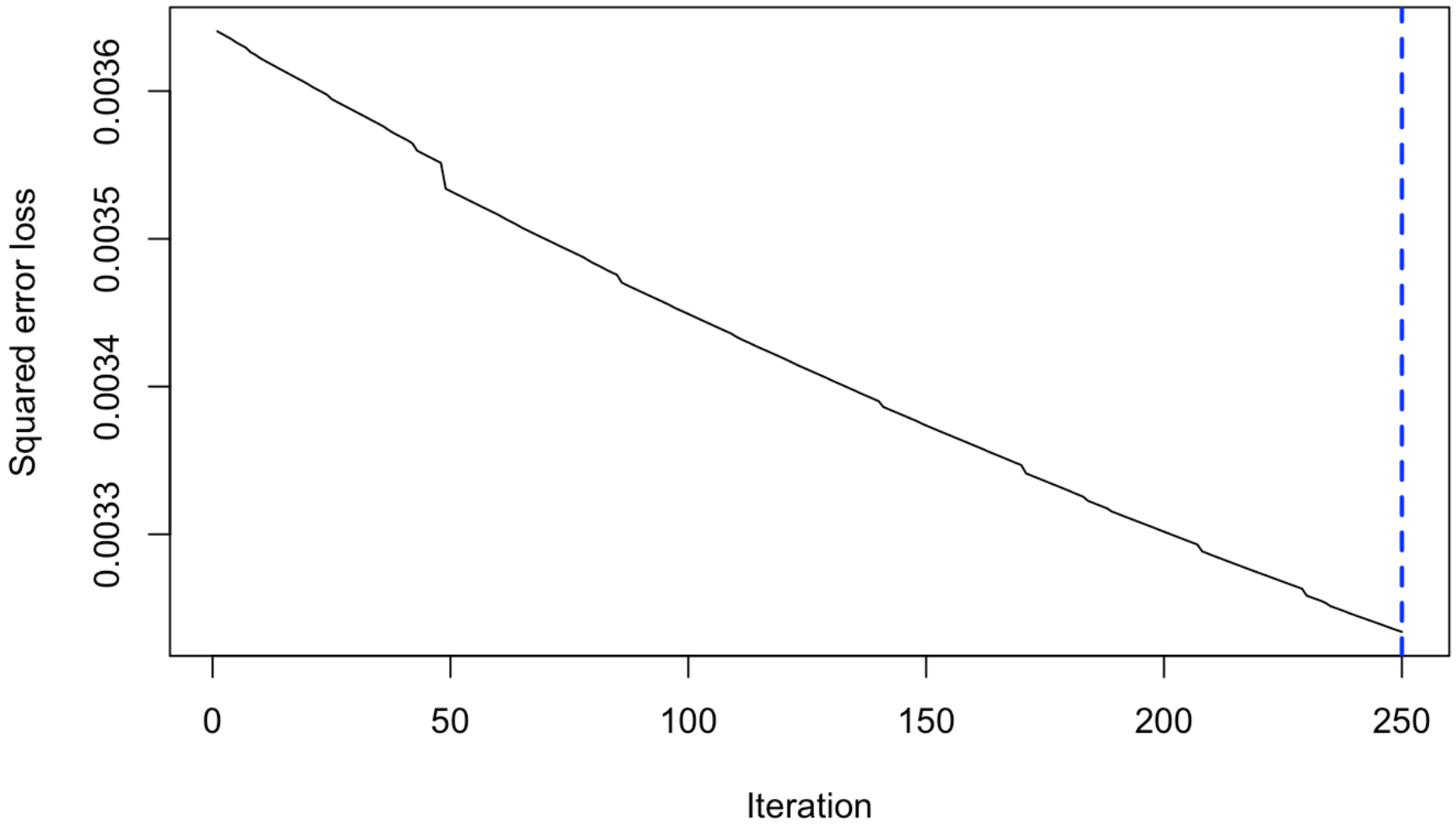
Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

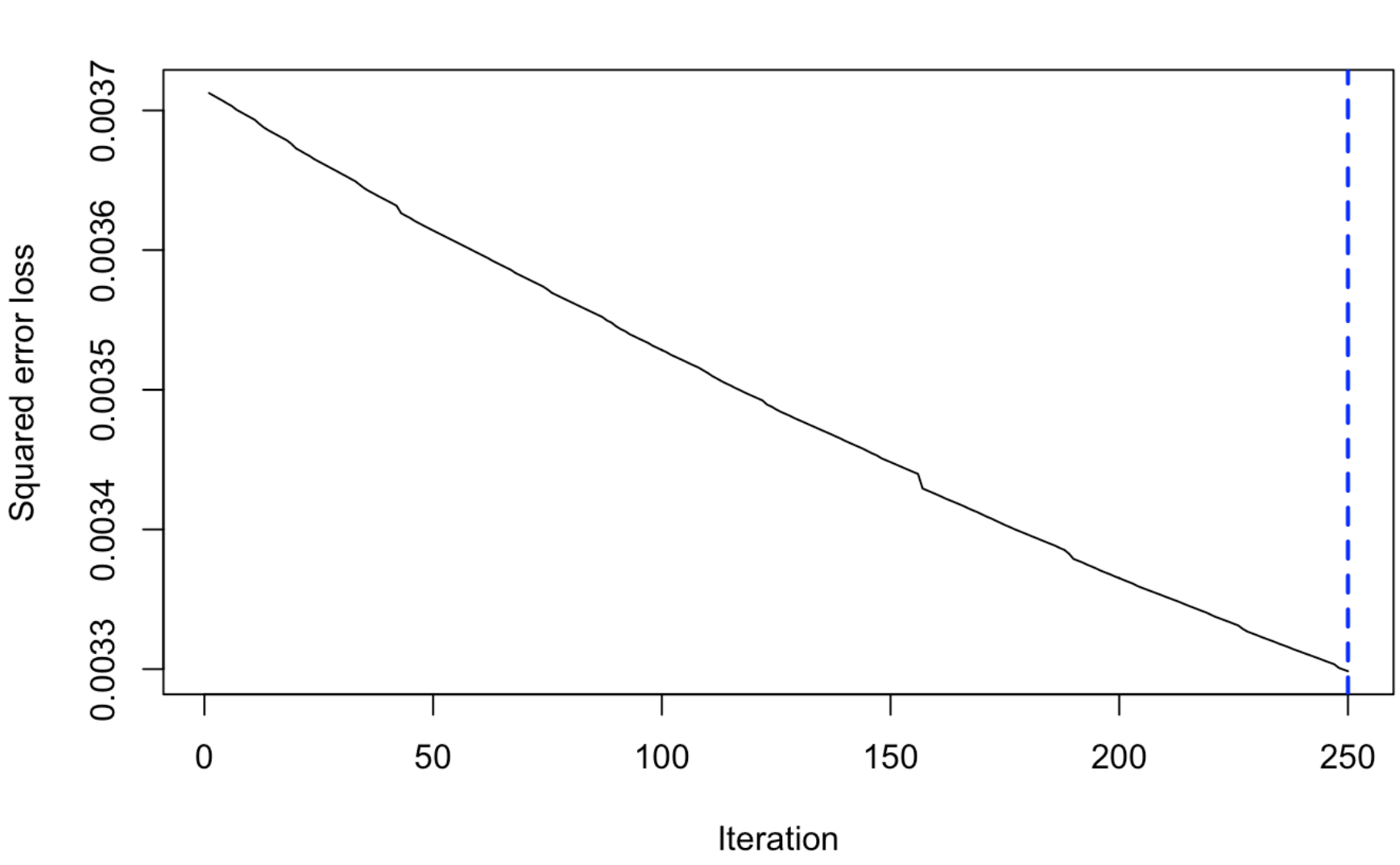

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

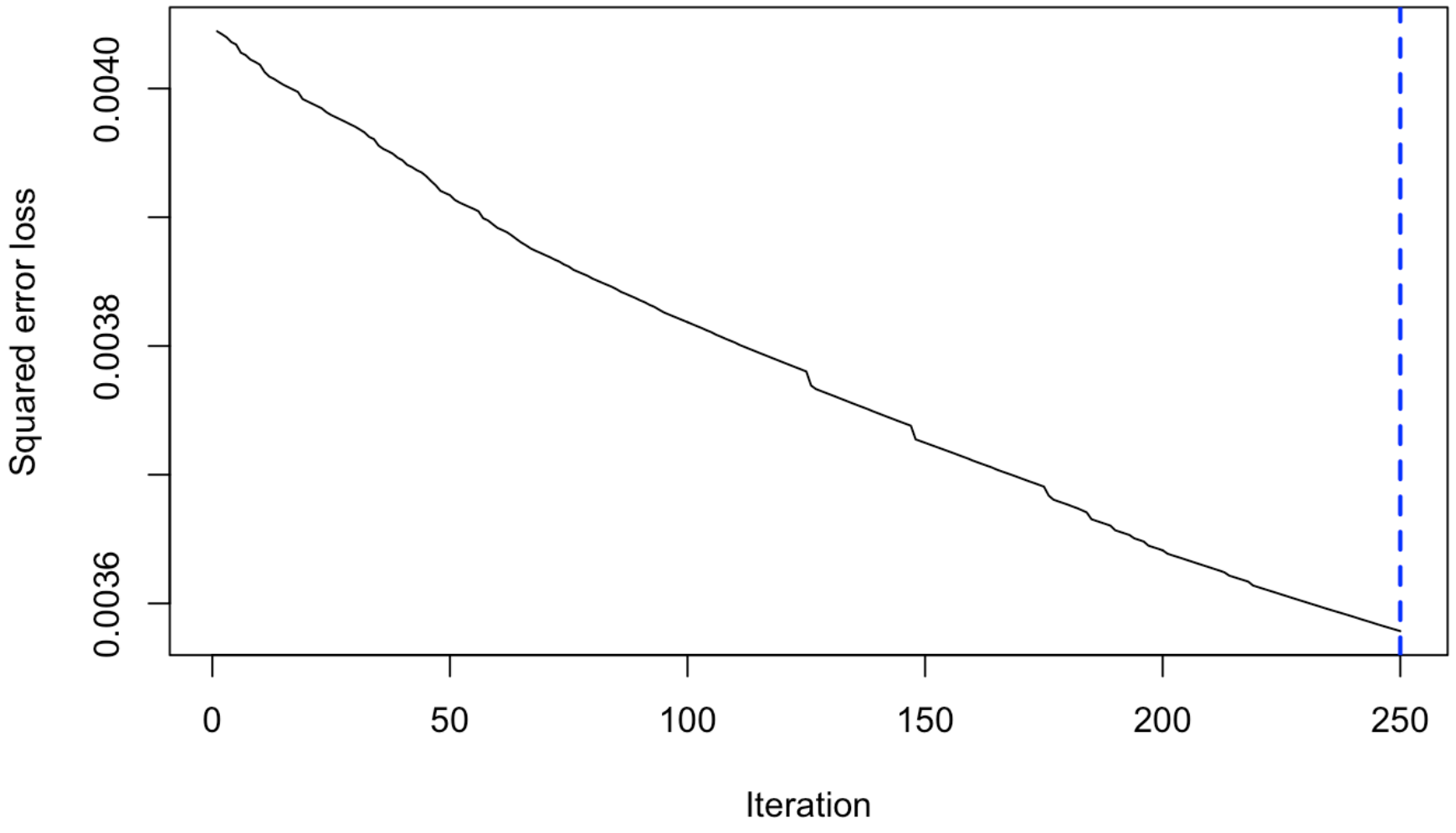
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0034
0.0035
0.0036
0.0037
0
50
100
150
200
250
Iteration

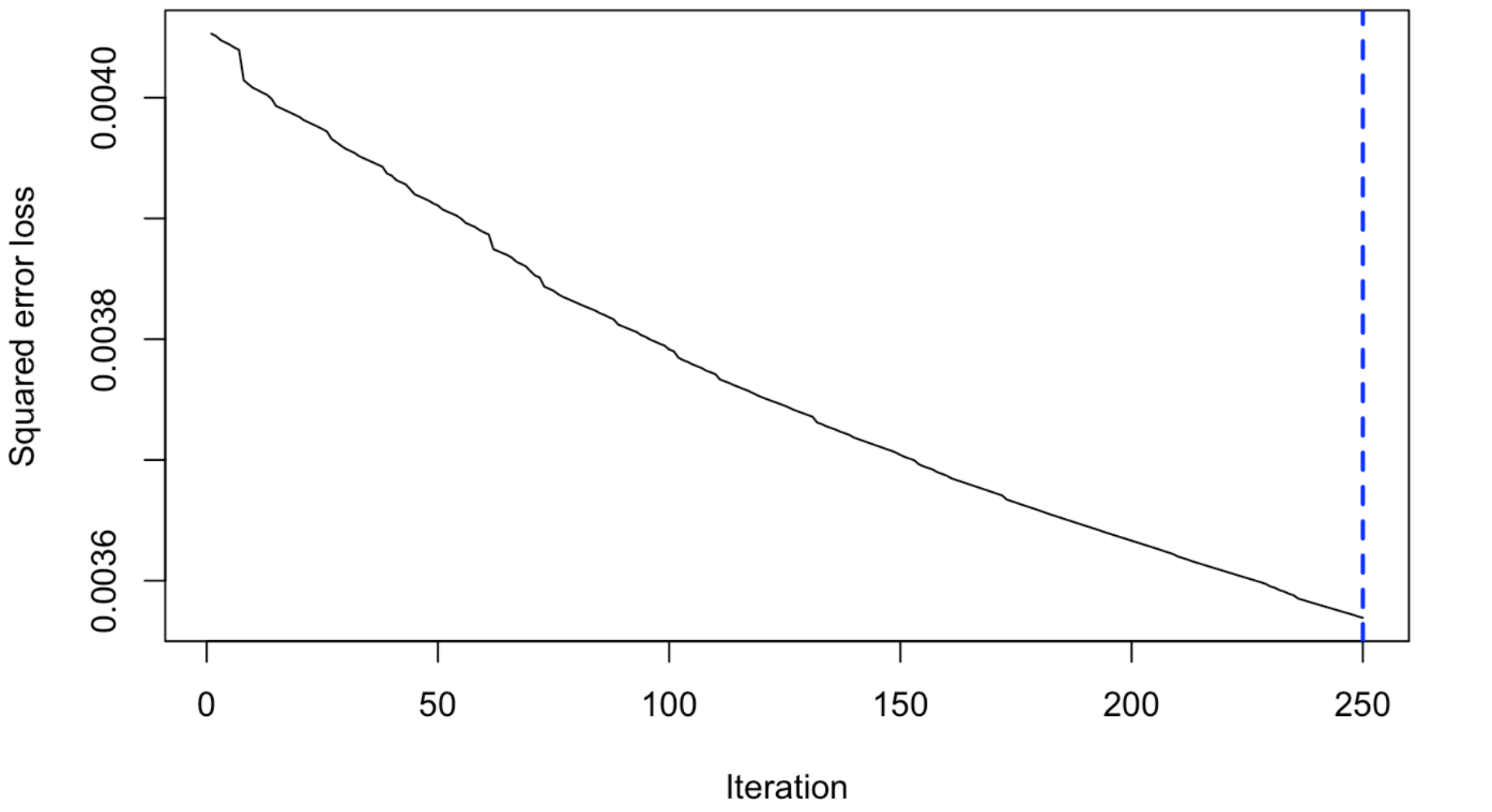

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0034
0.0035
0.0036
0
50
100
150
200
250
Iteration

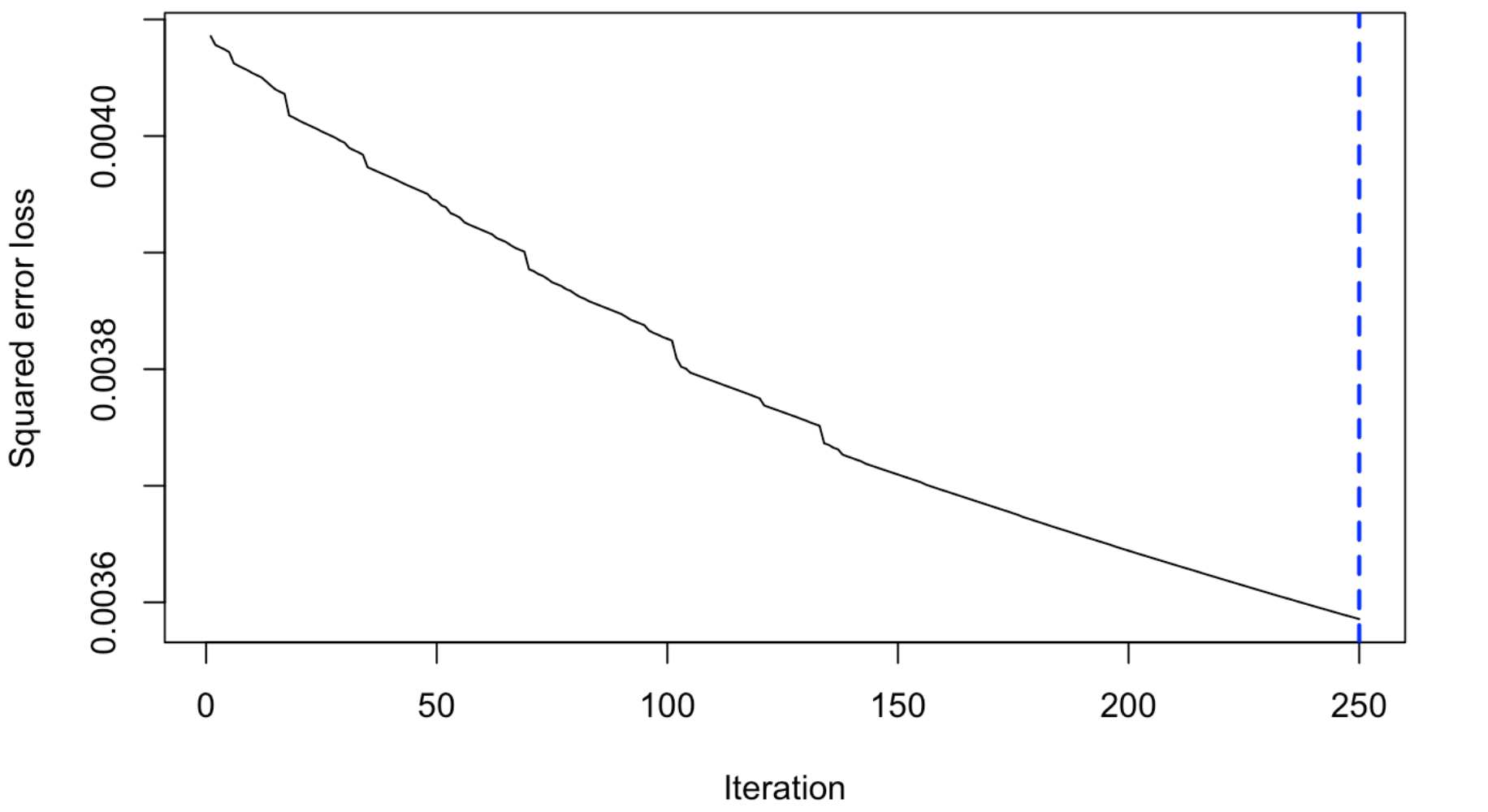

Squared error loss
0.0033
0.0034
0.0035
0.0036
0.0037
0
50
100
150
200
250
Iteration

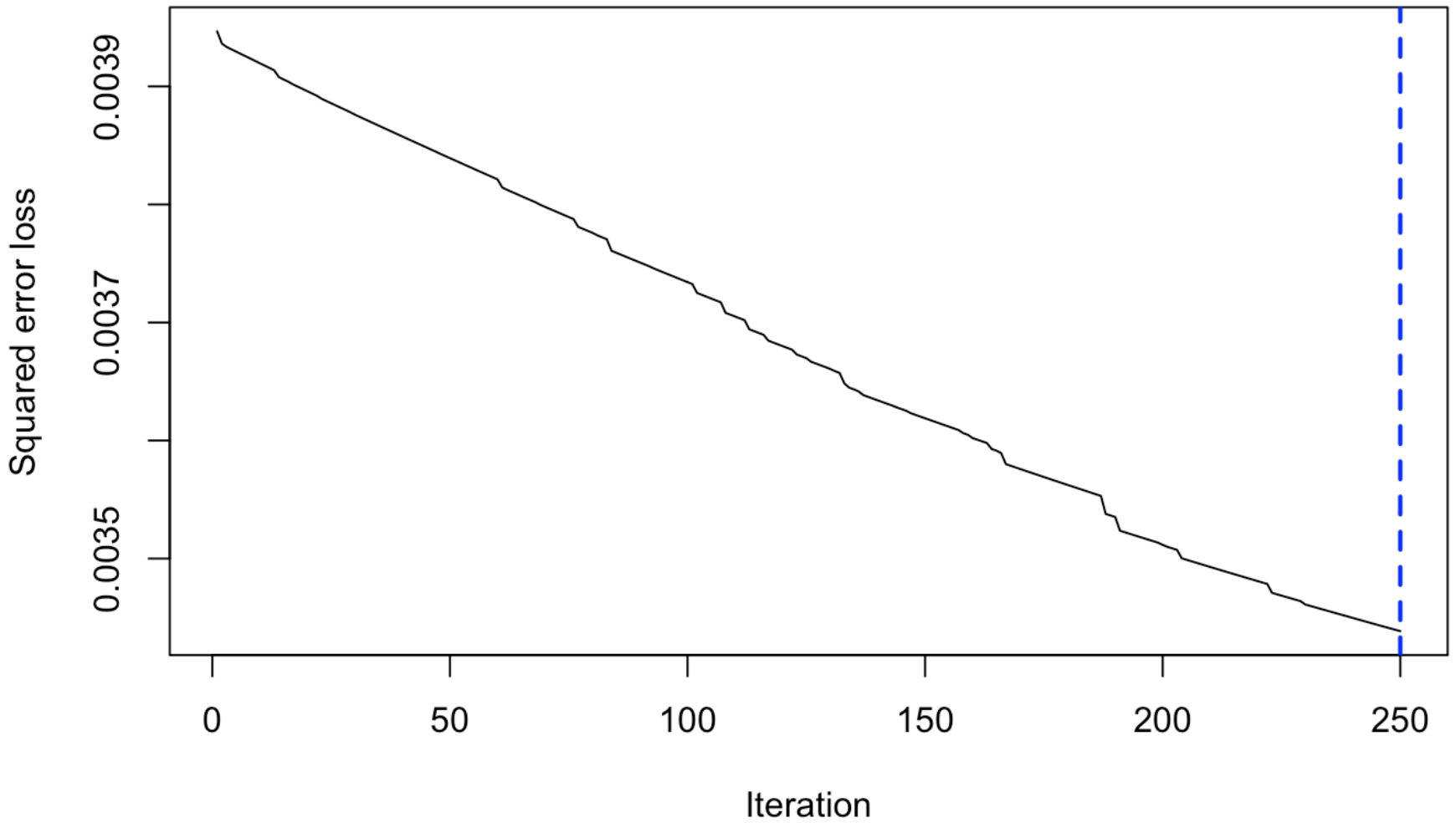

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

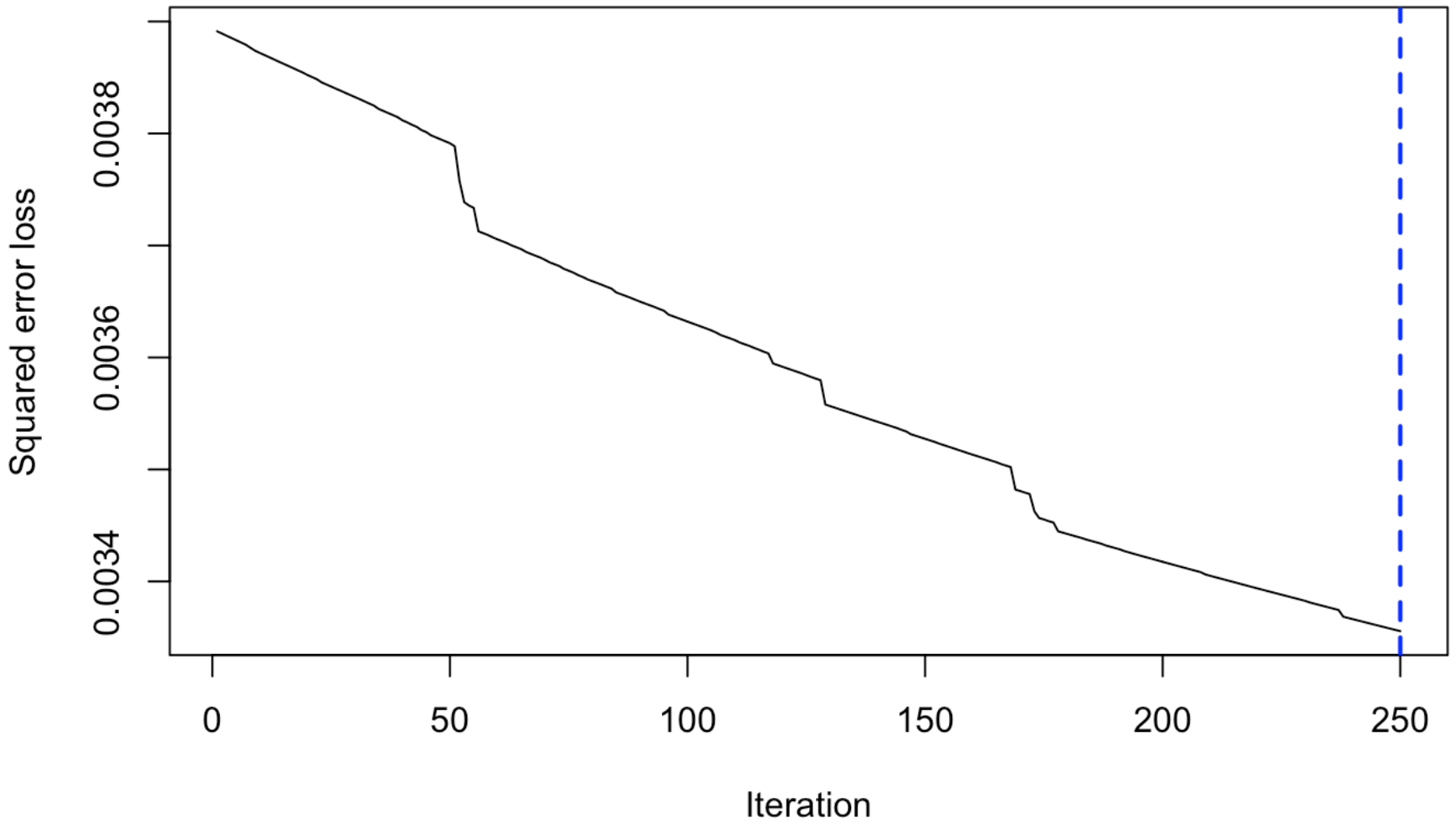

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

```
k=  2
```

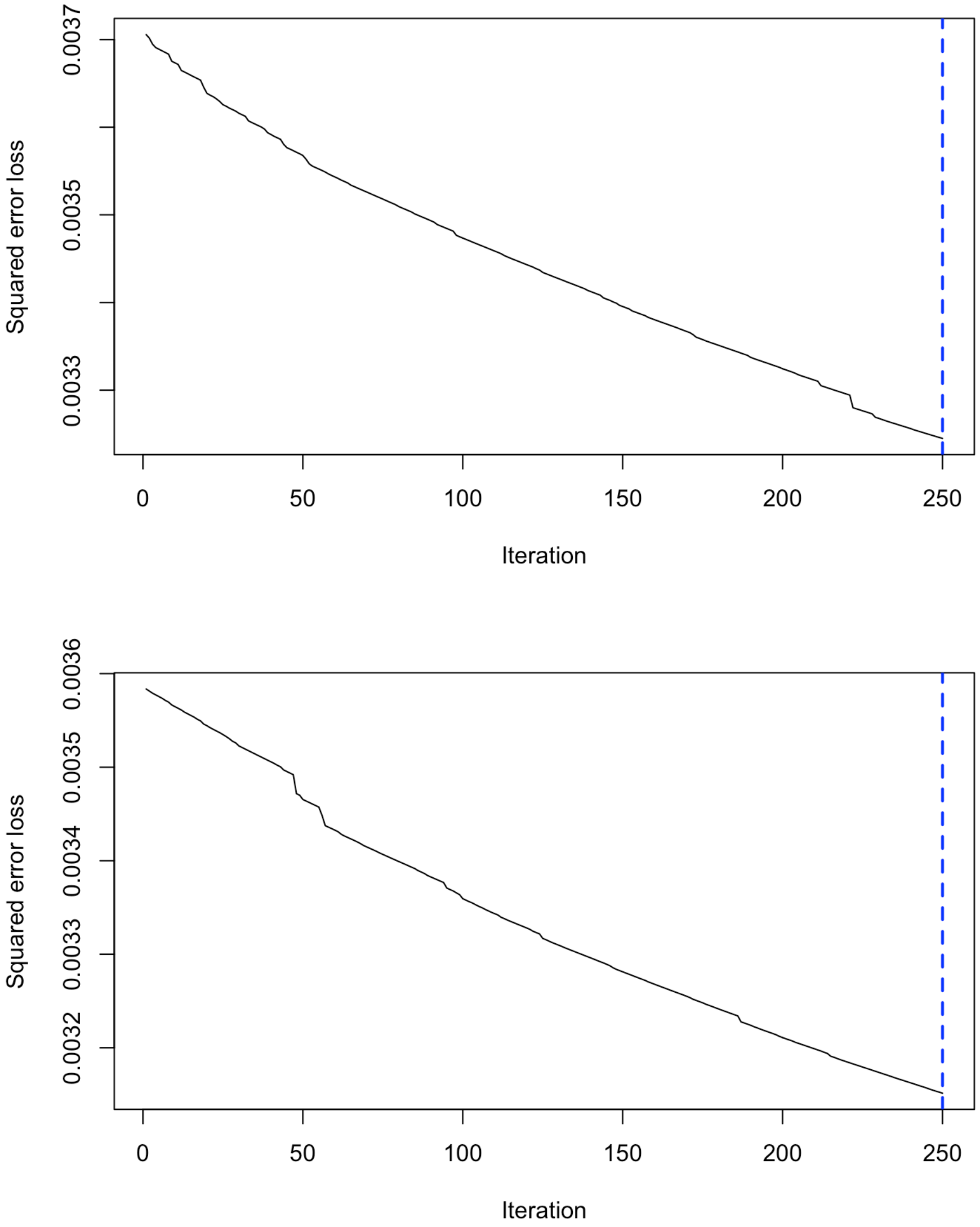

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

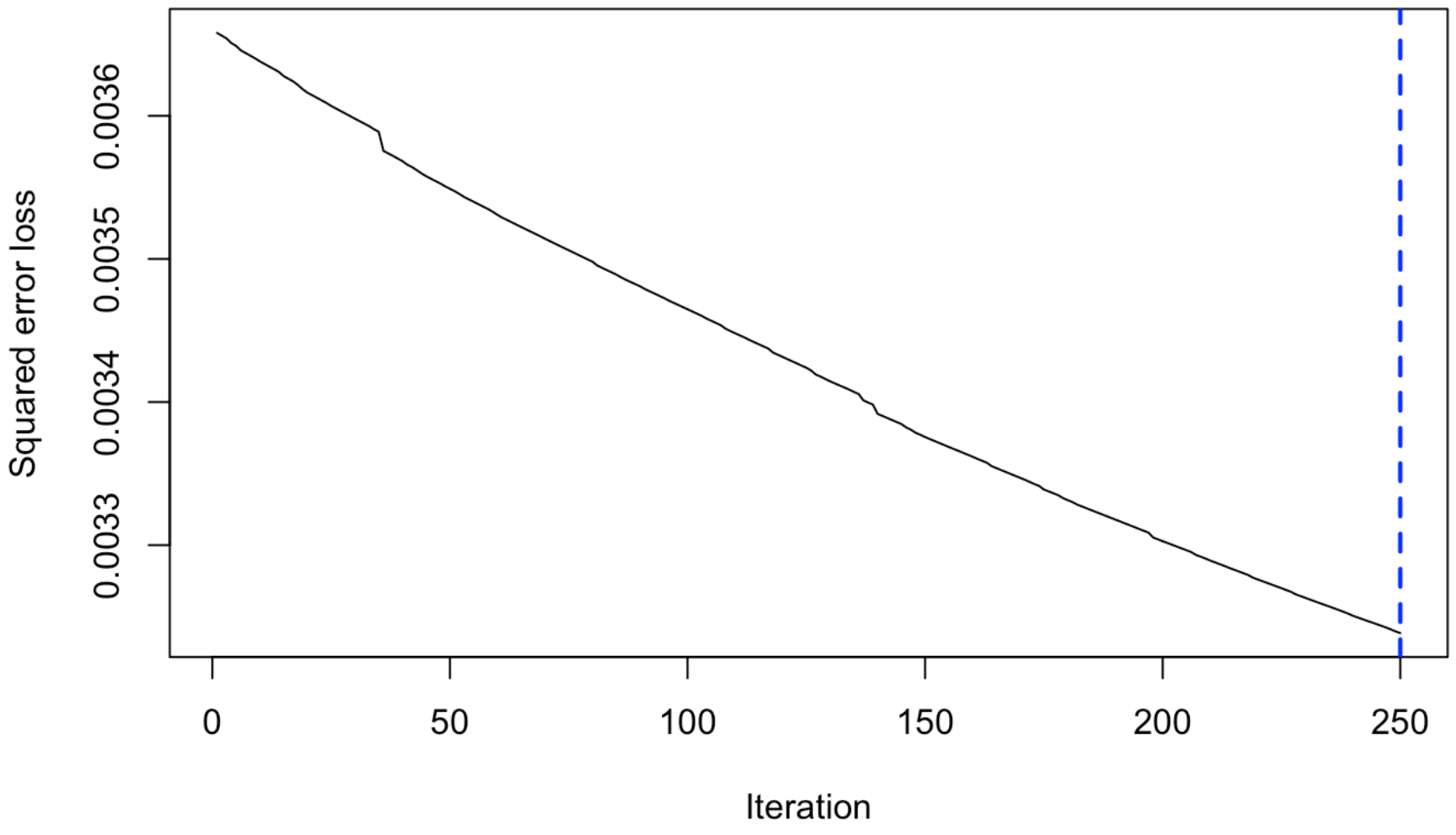

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration

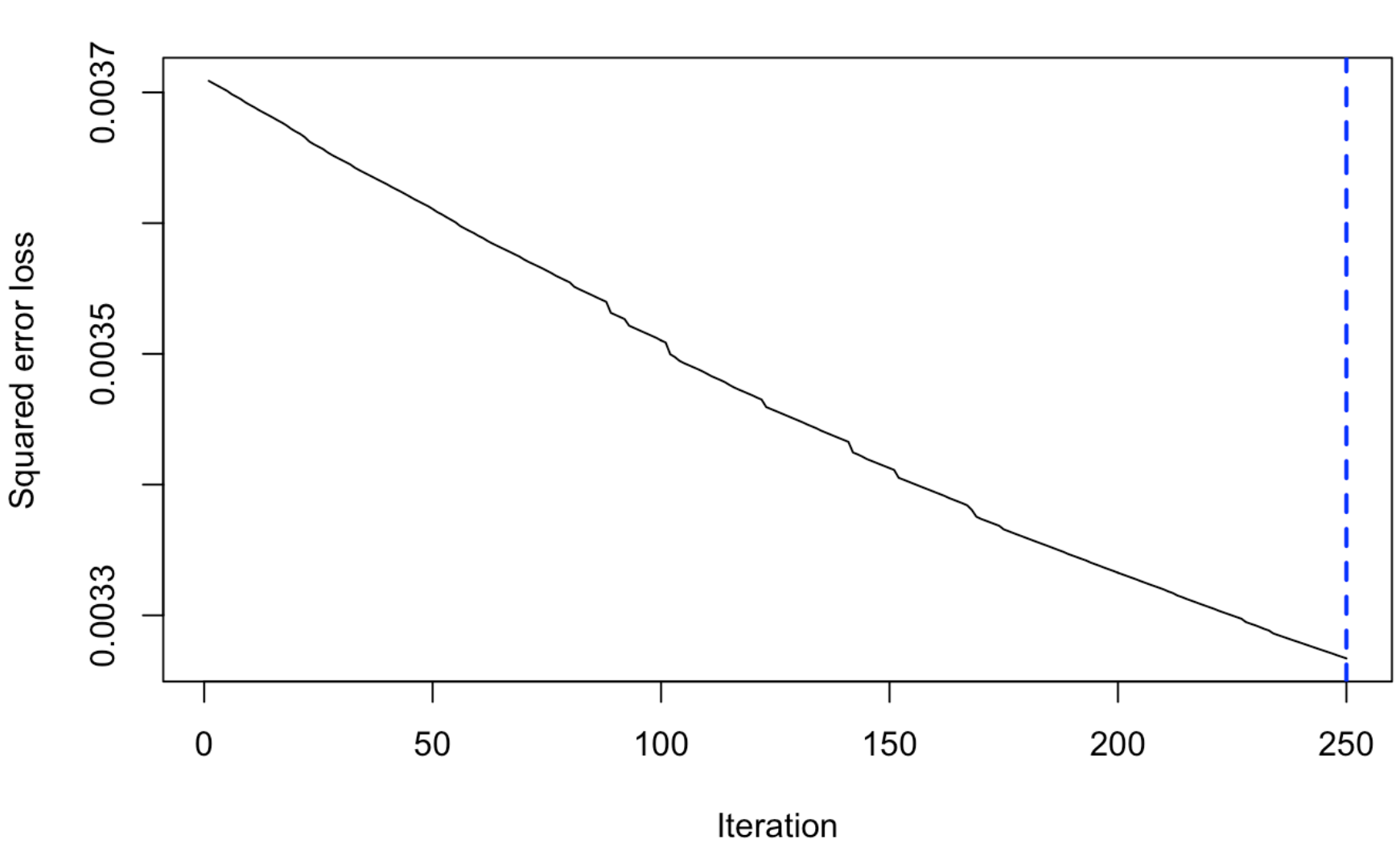
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

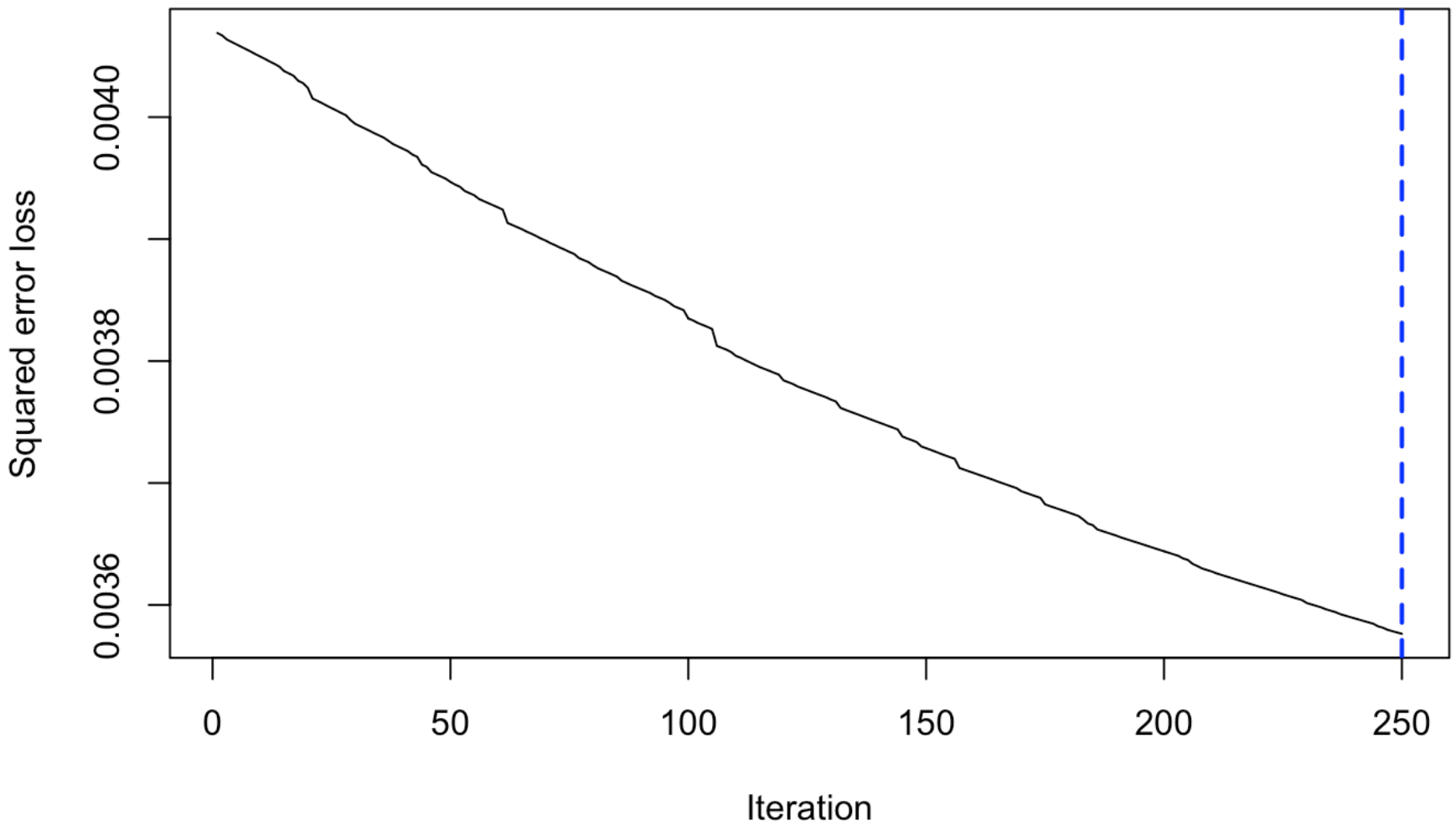

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

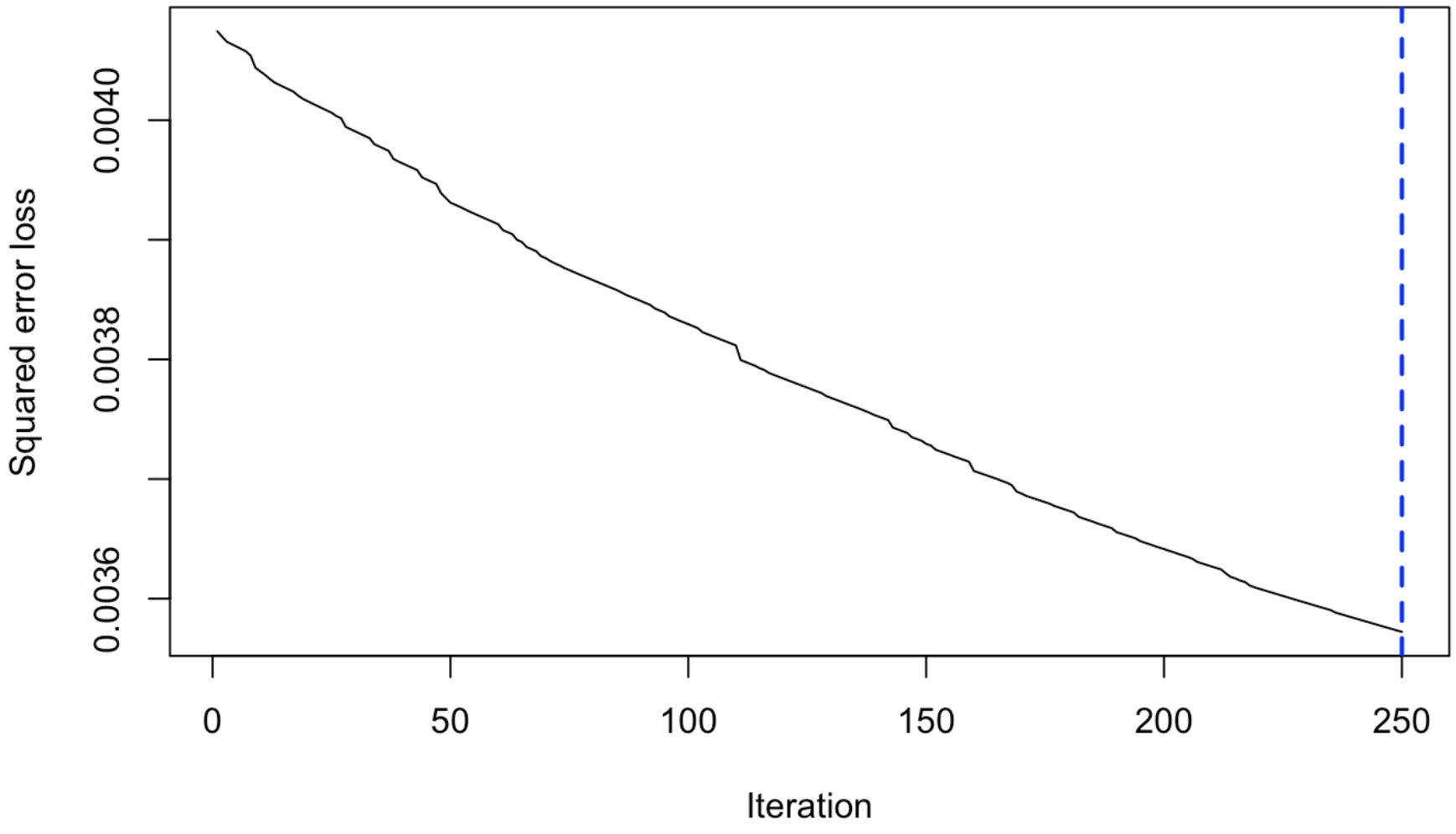

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration
Squared error loss
0.0032
0.0033
0.0034
0.0035
0.003
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0034
0.0035
0.0036
0
50
100
150
200
250
Iteration

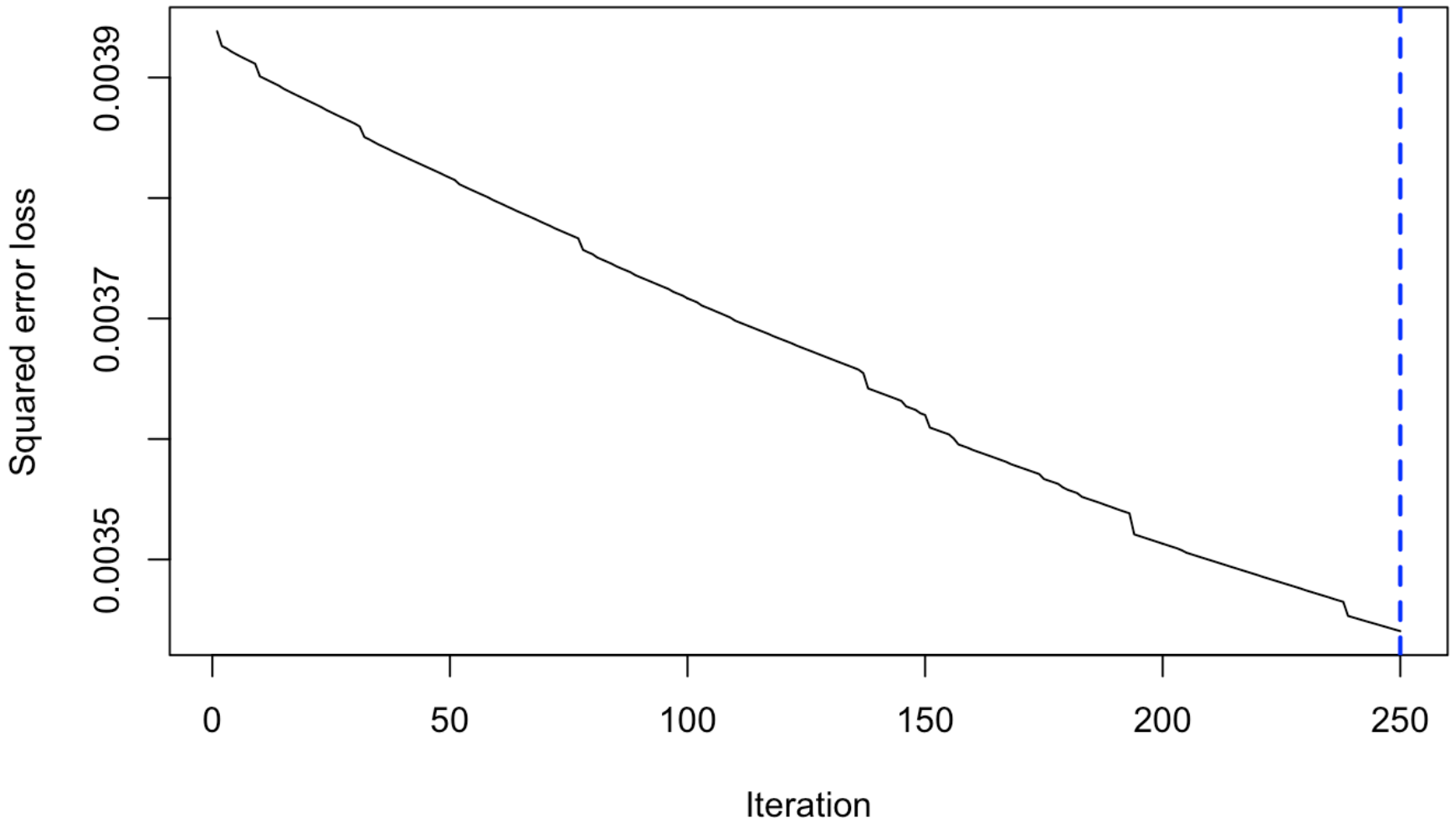

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration


Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
Iteration

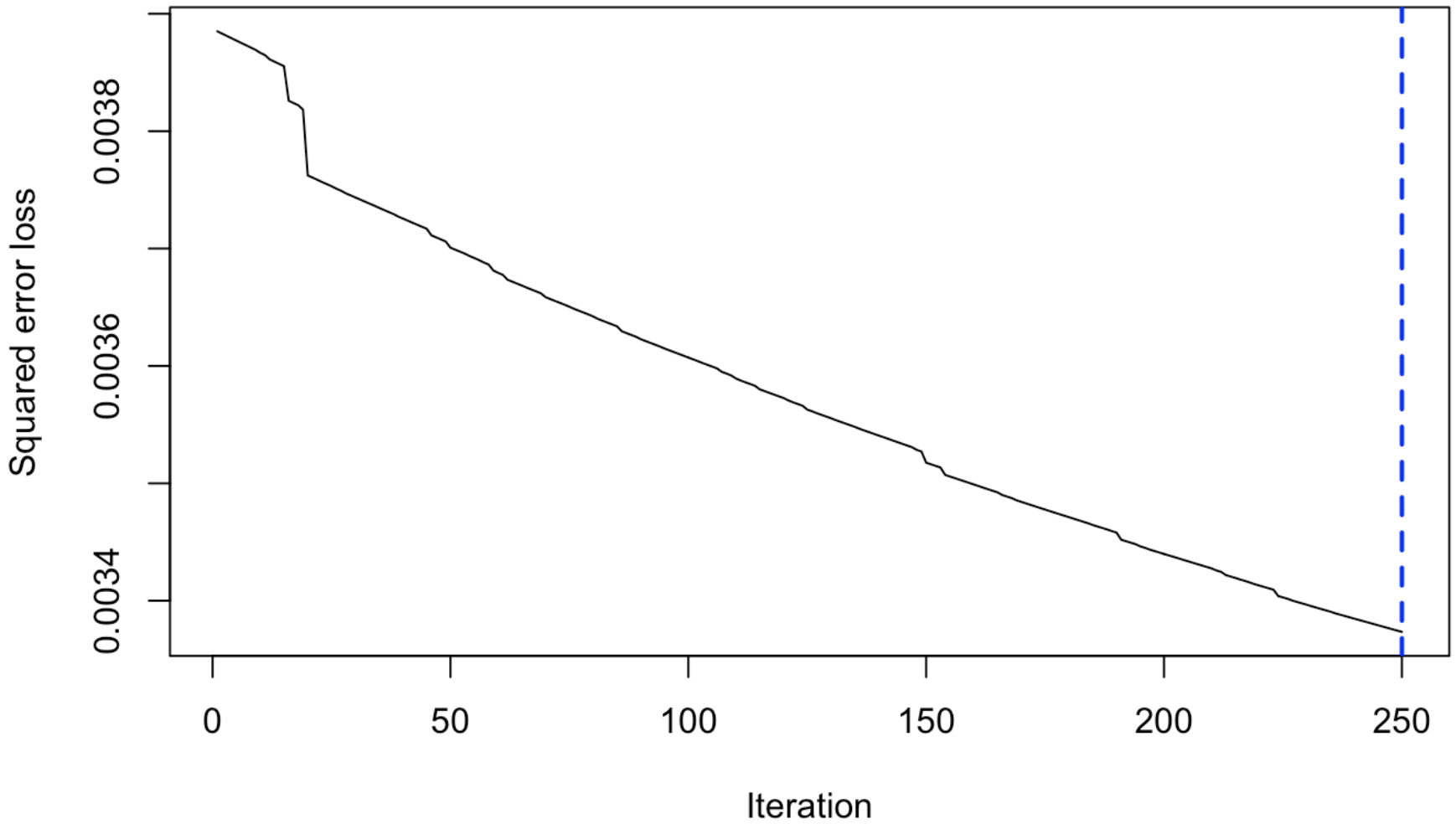
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

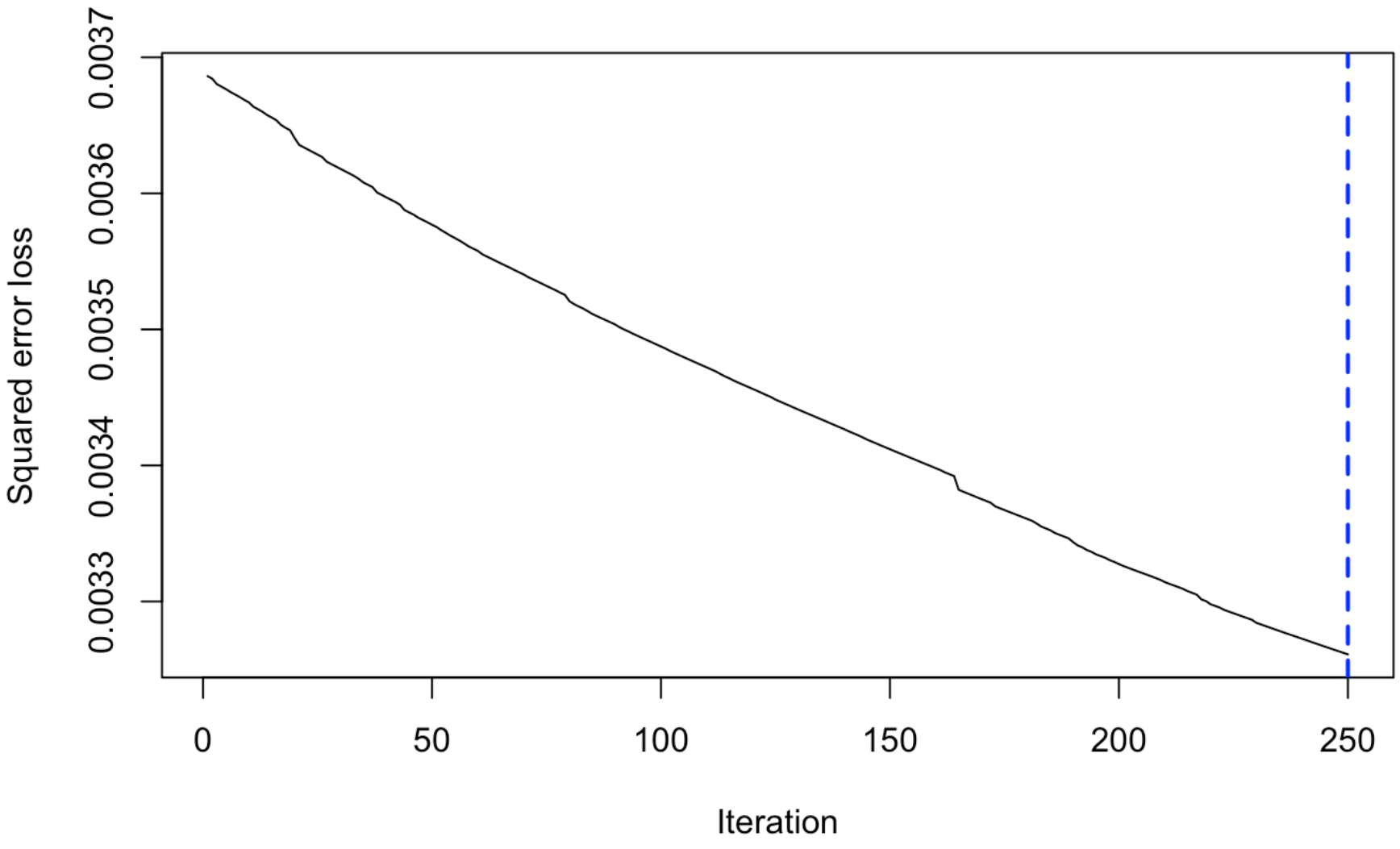
Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration


Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration

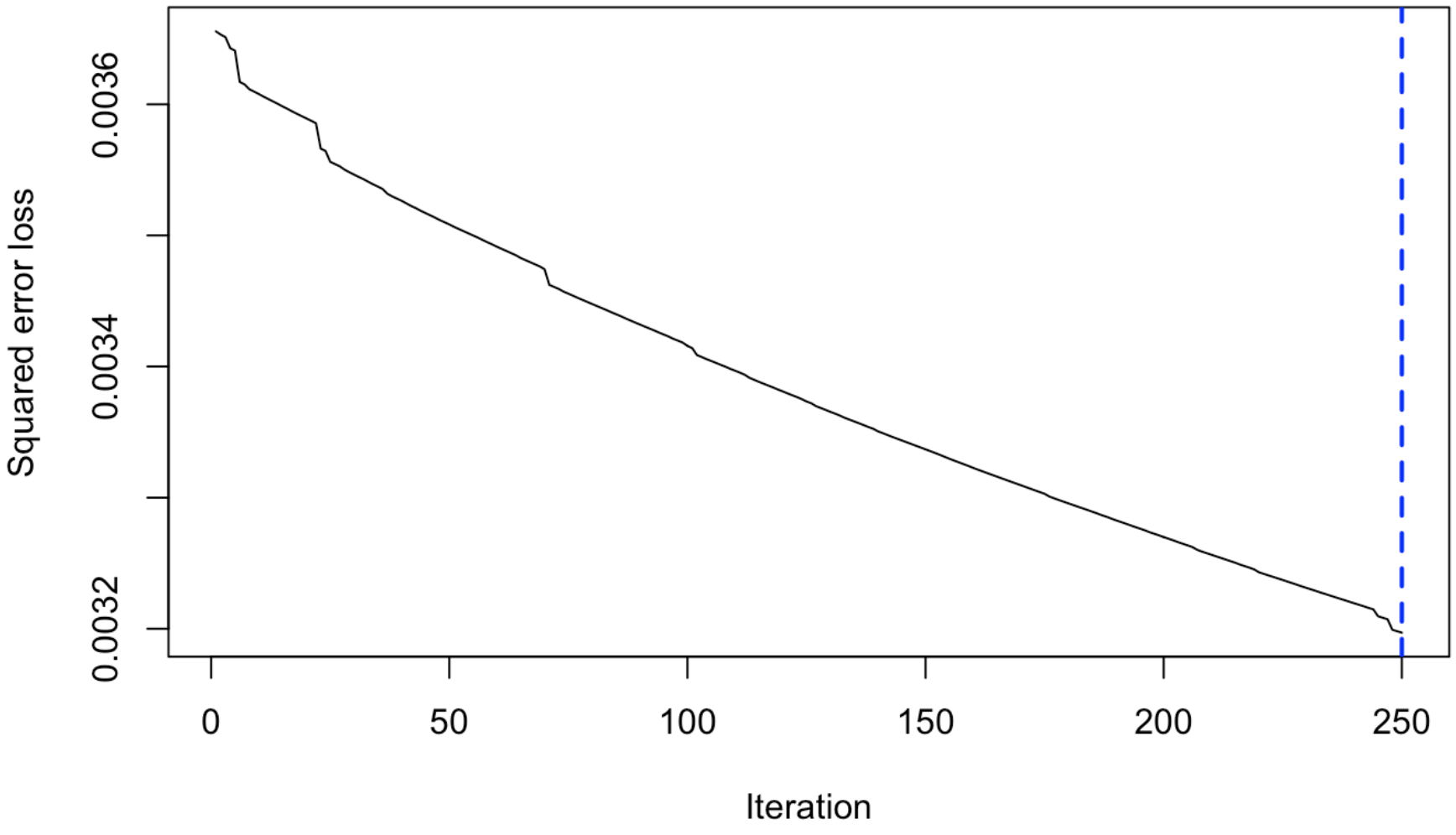
Squared error loss
0.0037
0.0035
0.0033
0
50
100
150
200
250
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

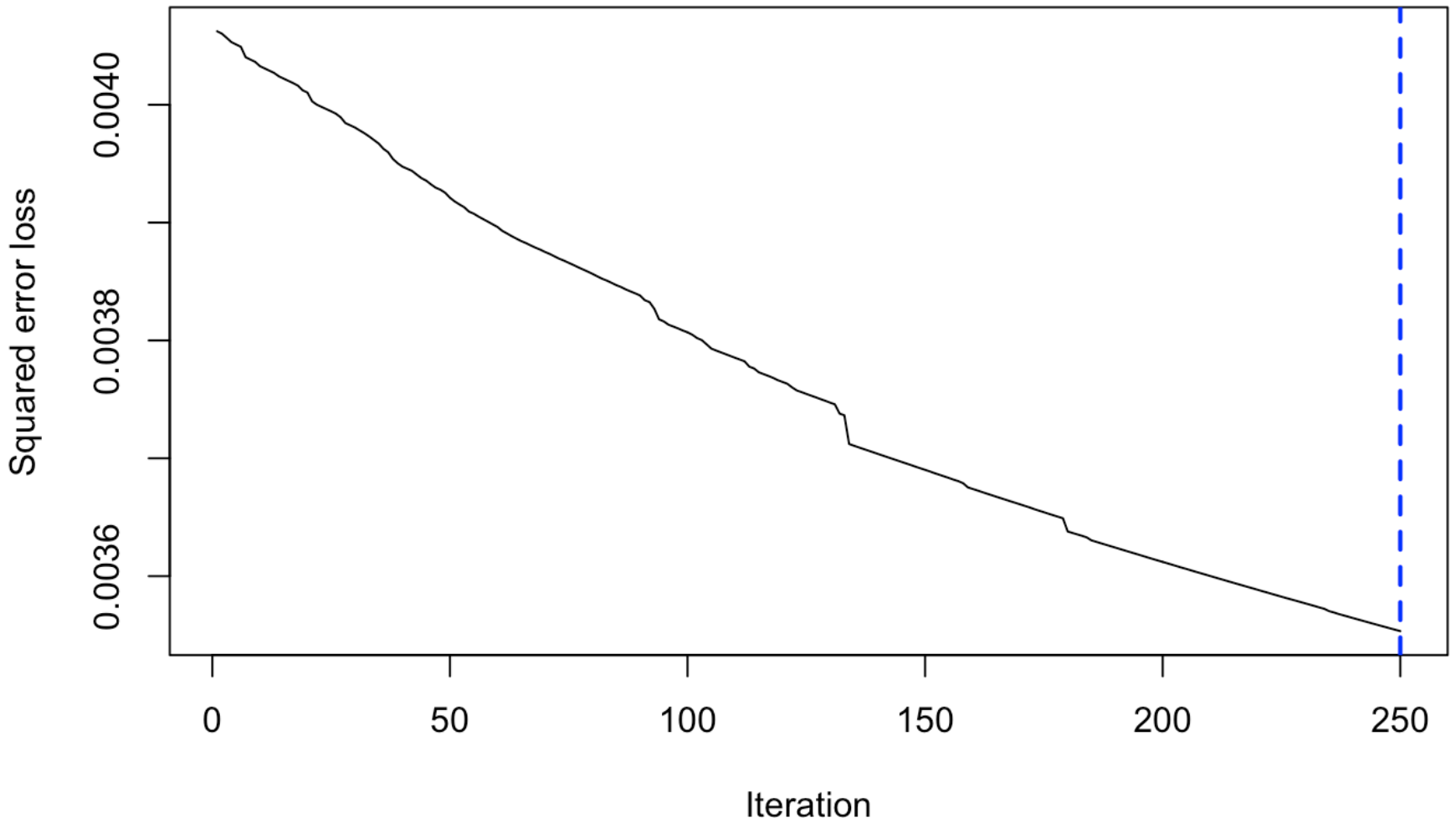
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0034
0.0035
0.0036
0.0037
0
50
100
150
200
250
Iteration

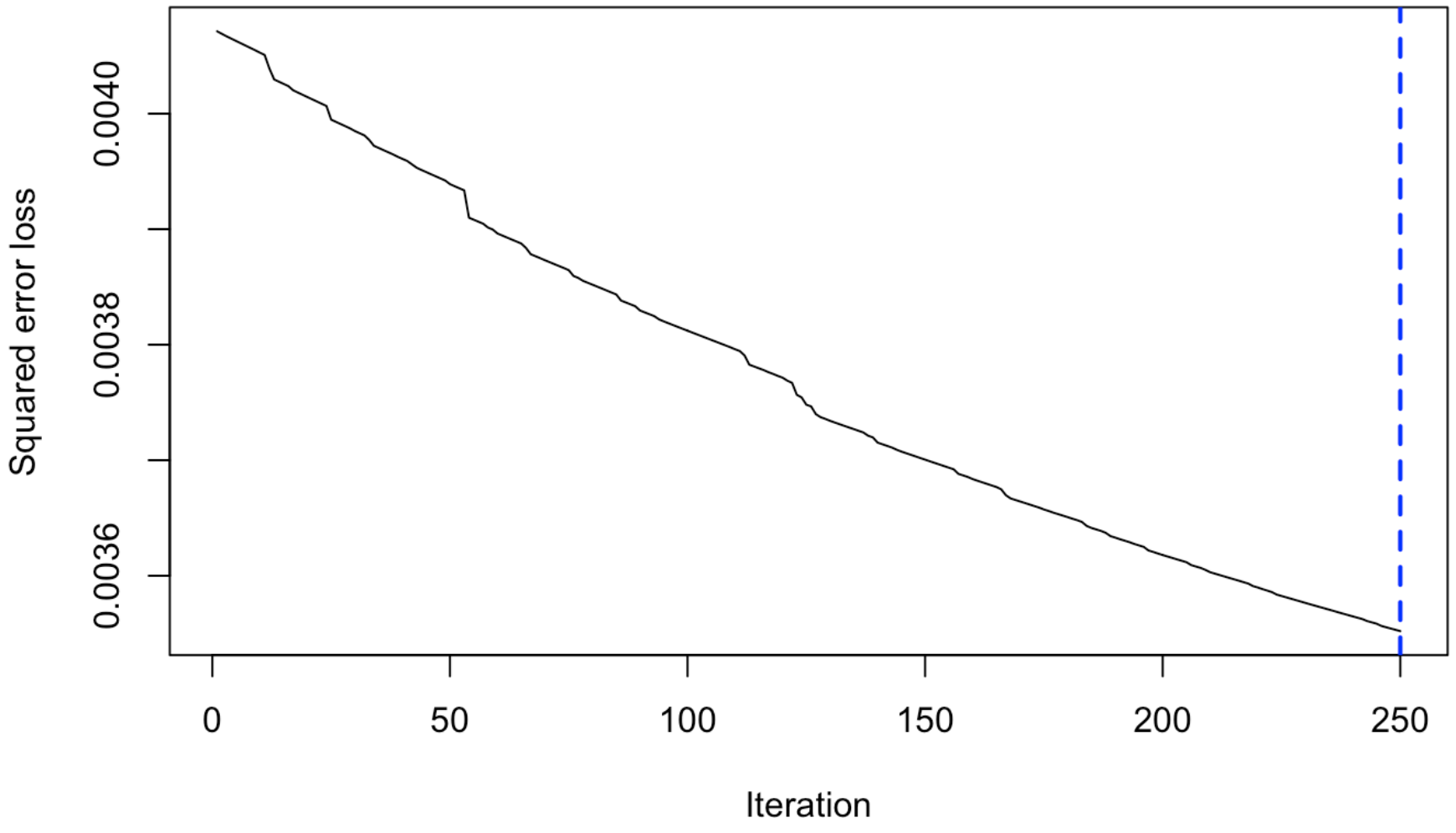
Squared error loss
0.0032
0.0033
0.0034
0.0035
0
50
100
150
200
250
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
Iteration

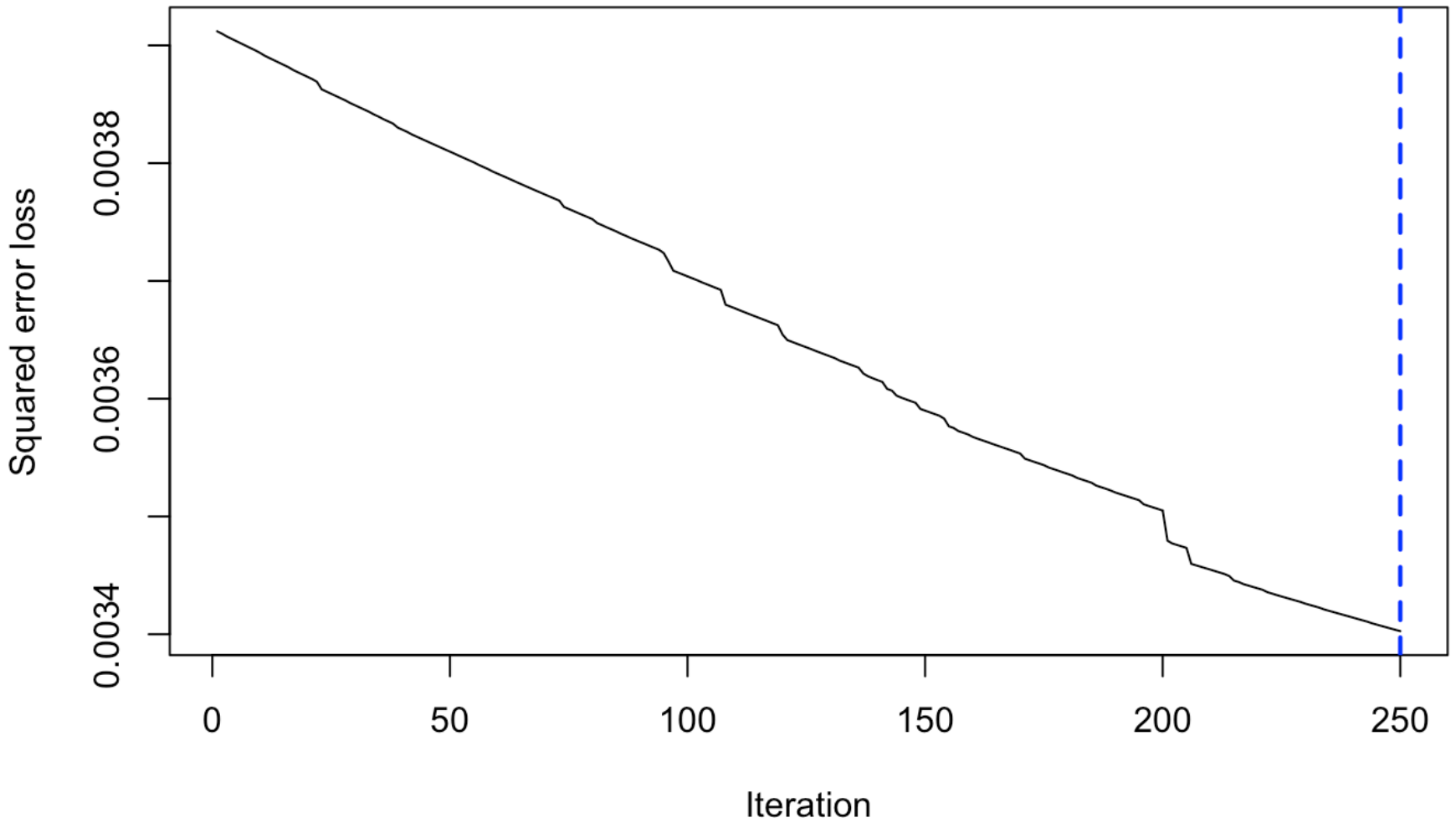
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

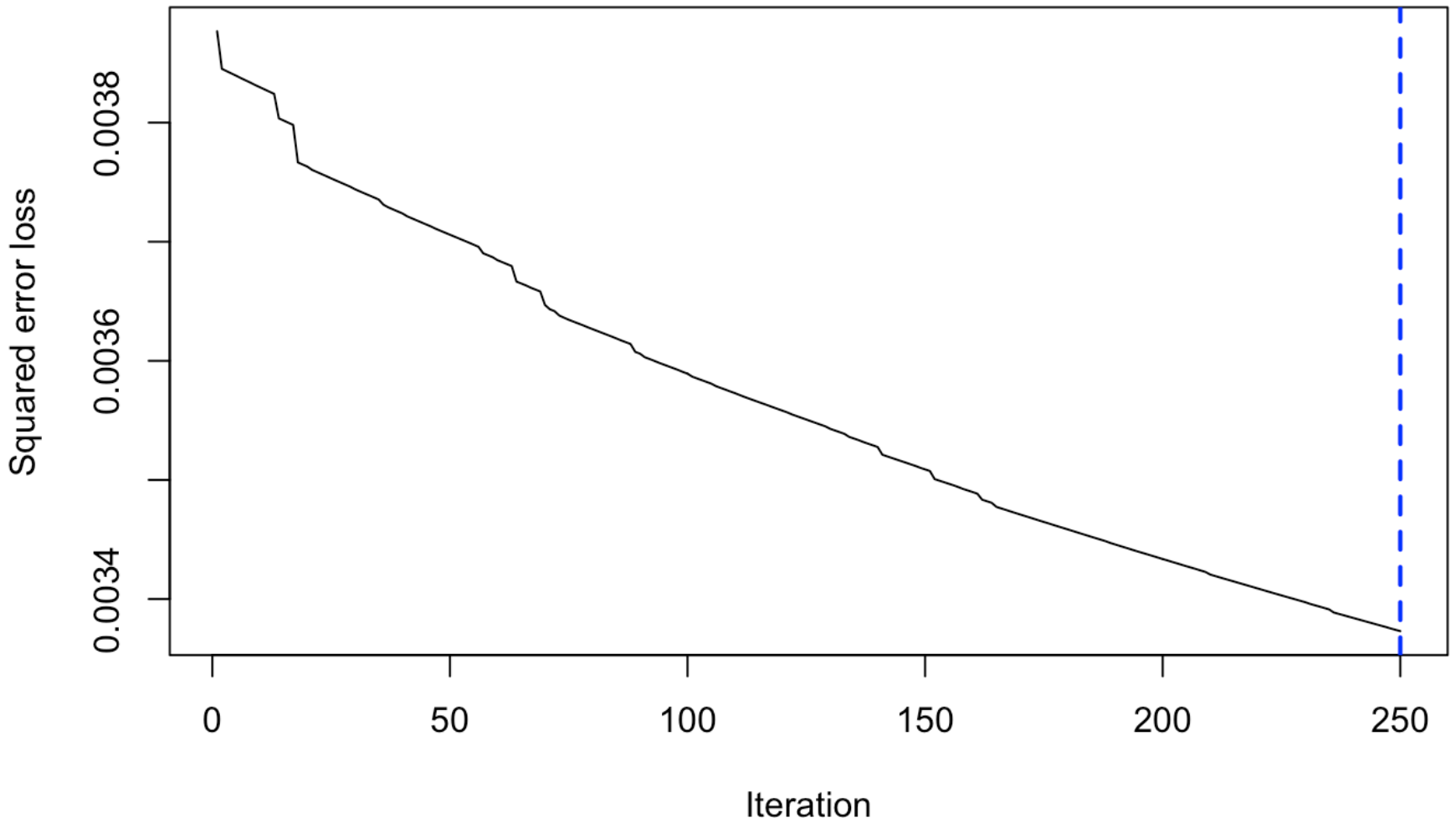
Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration

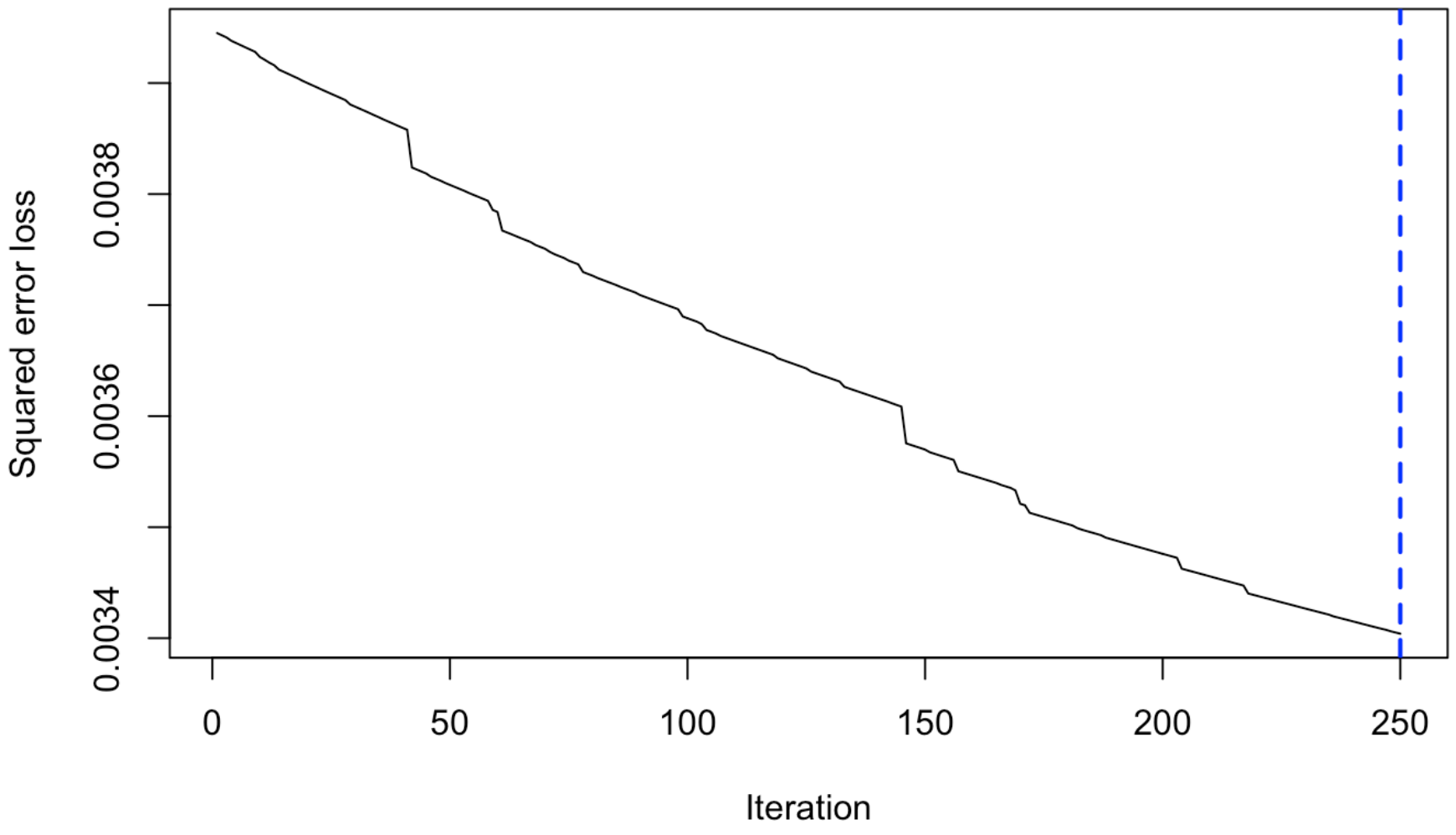

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

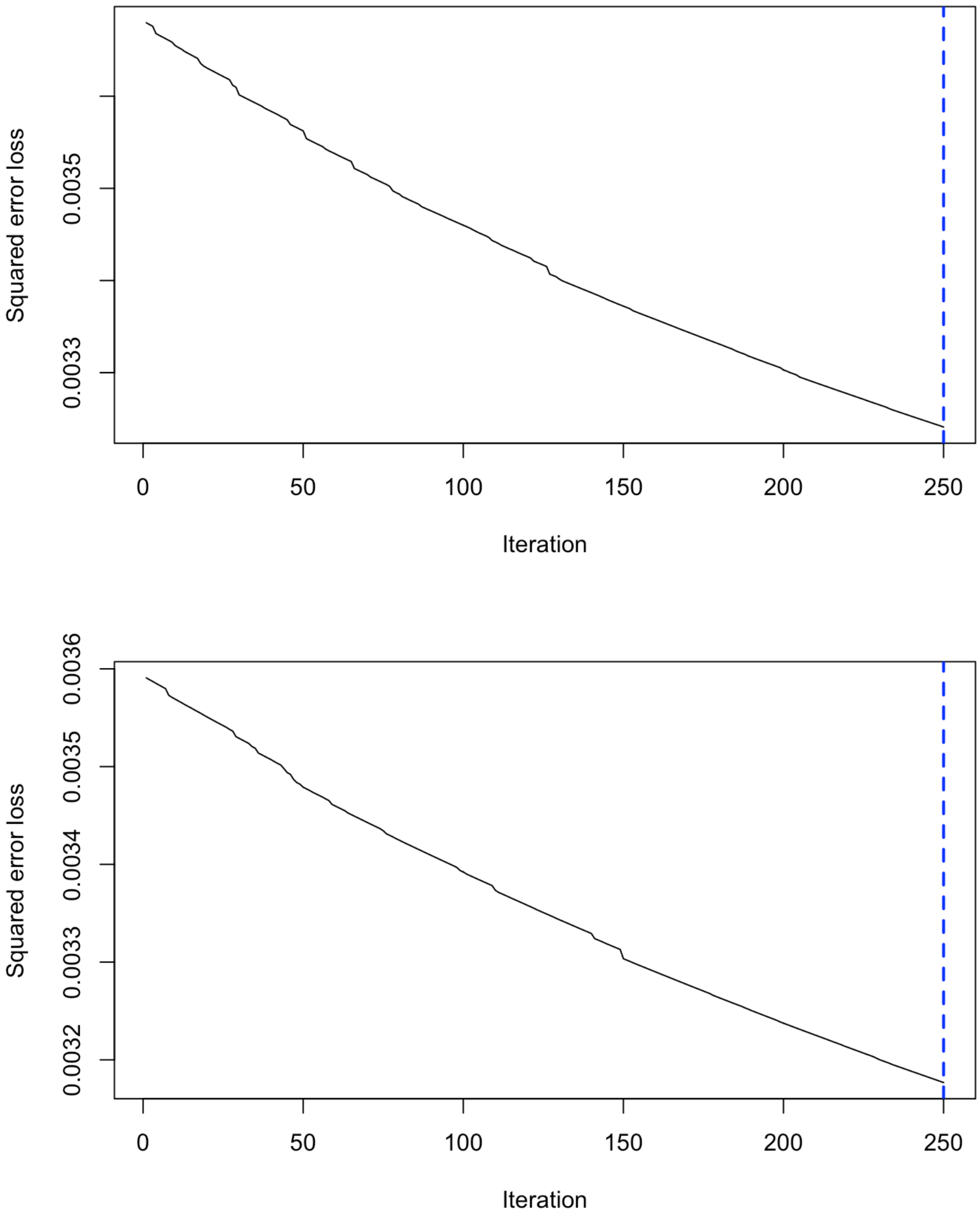
Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

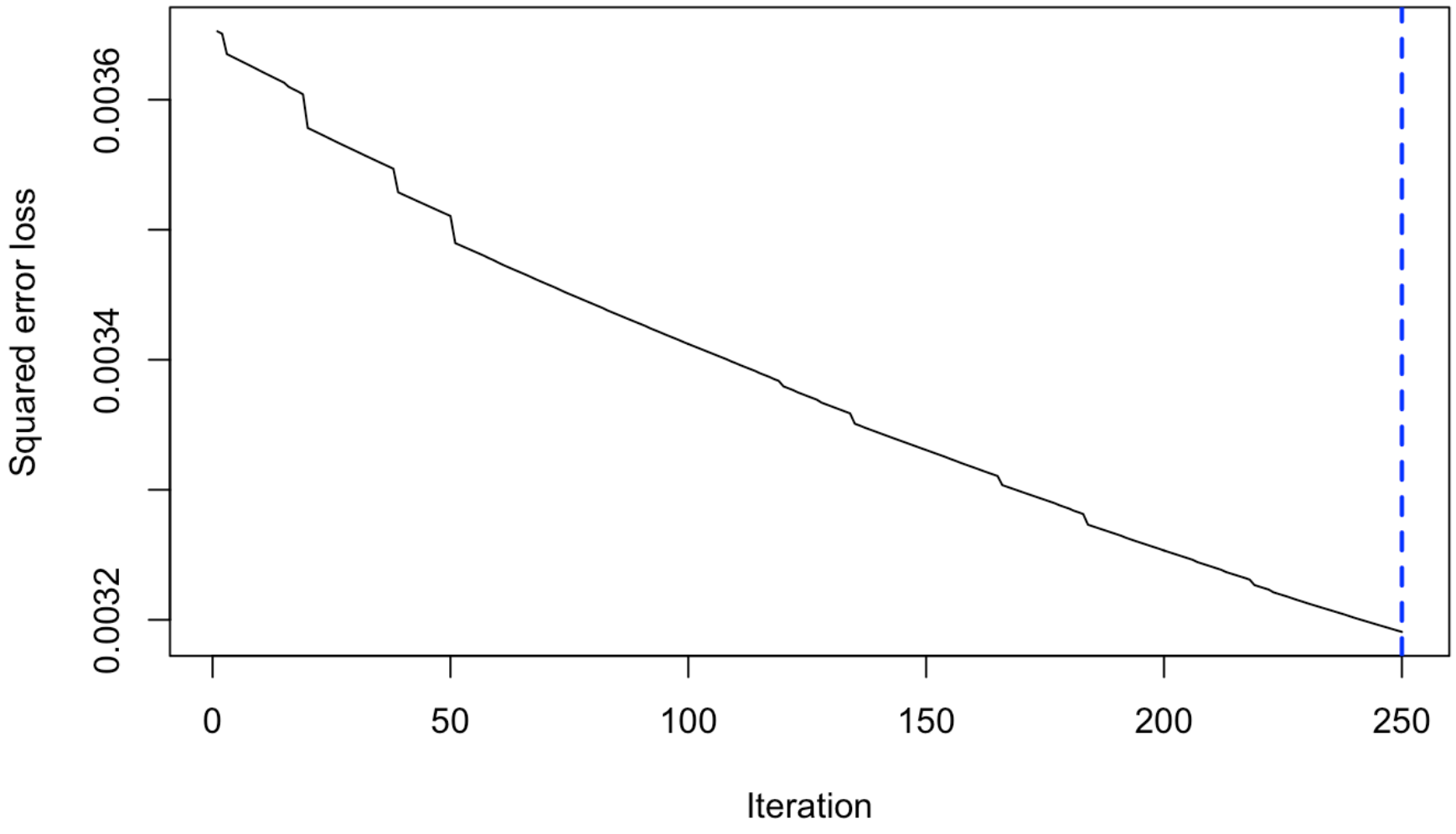

Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration

Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration

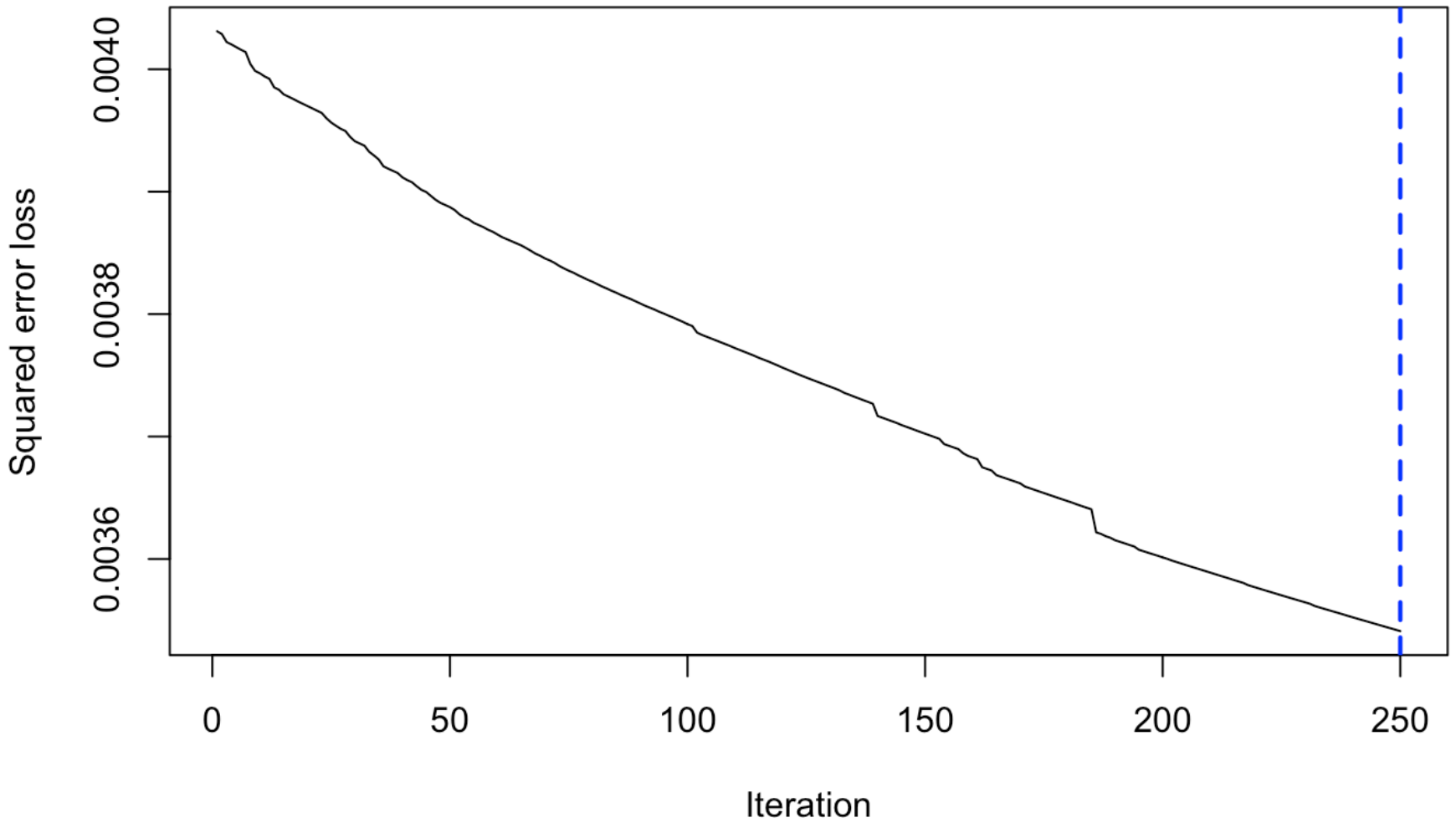

Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration


Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration

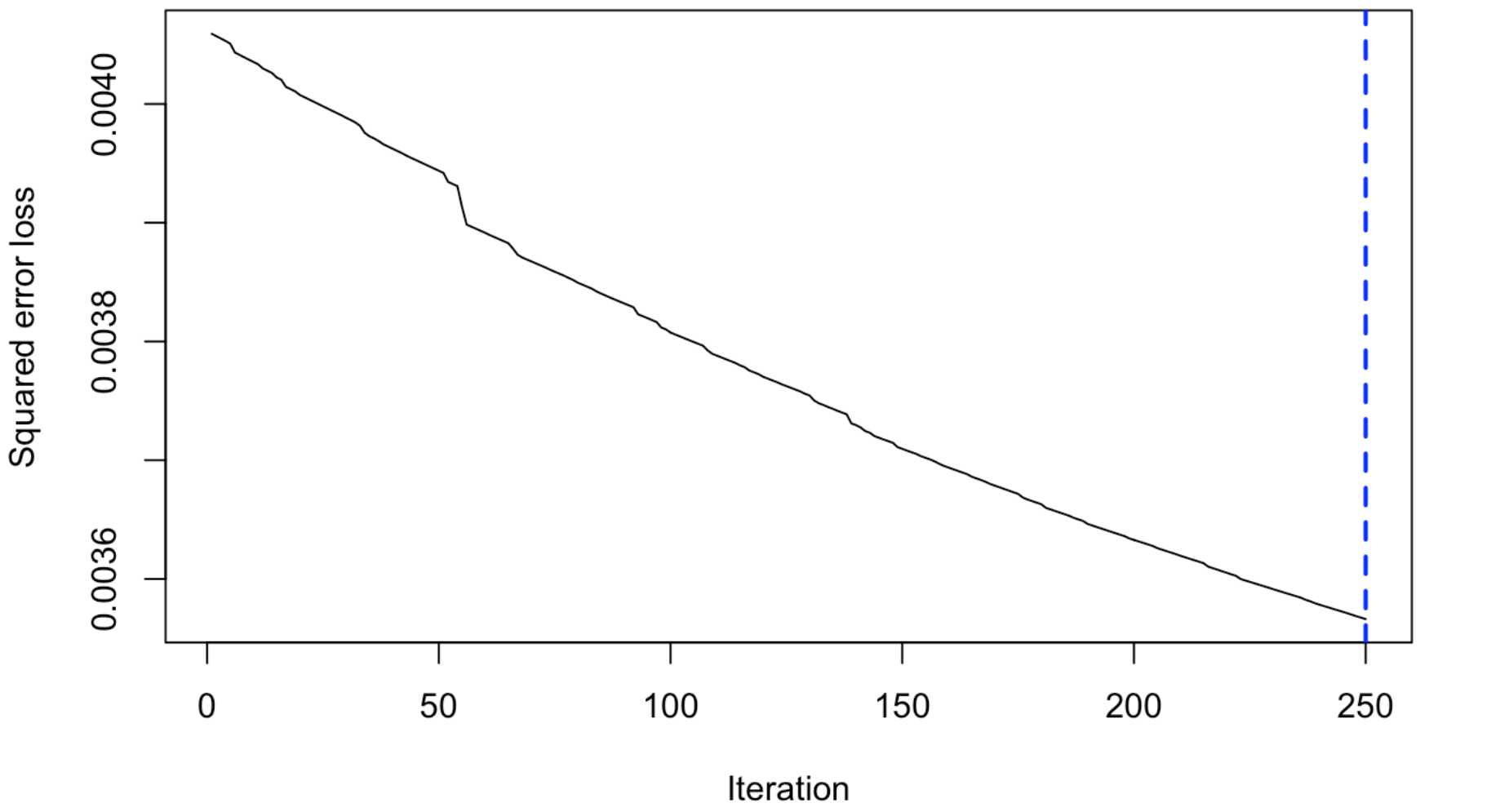

Squared error loss
0.0033
0.0035
0
50
100
150
200
250
Iteration
Squared error loss
0.0032
0.0033
0.0034
0.0035
0.0036
0
50
100
150
200
250
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
Iteration

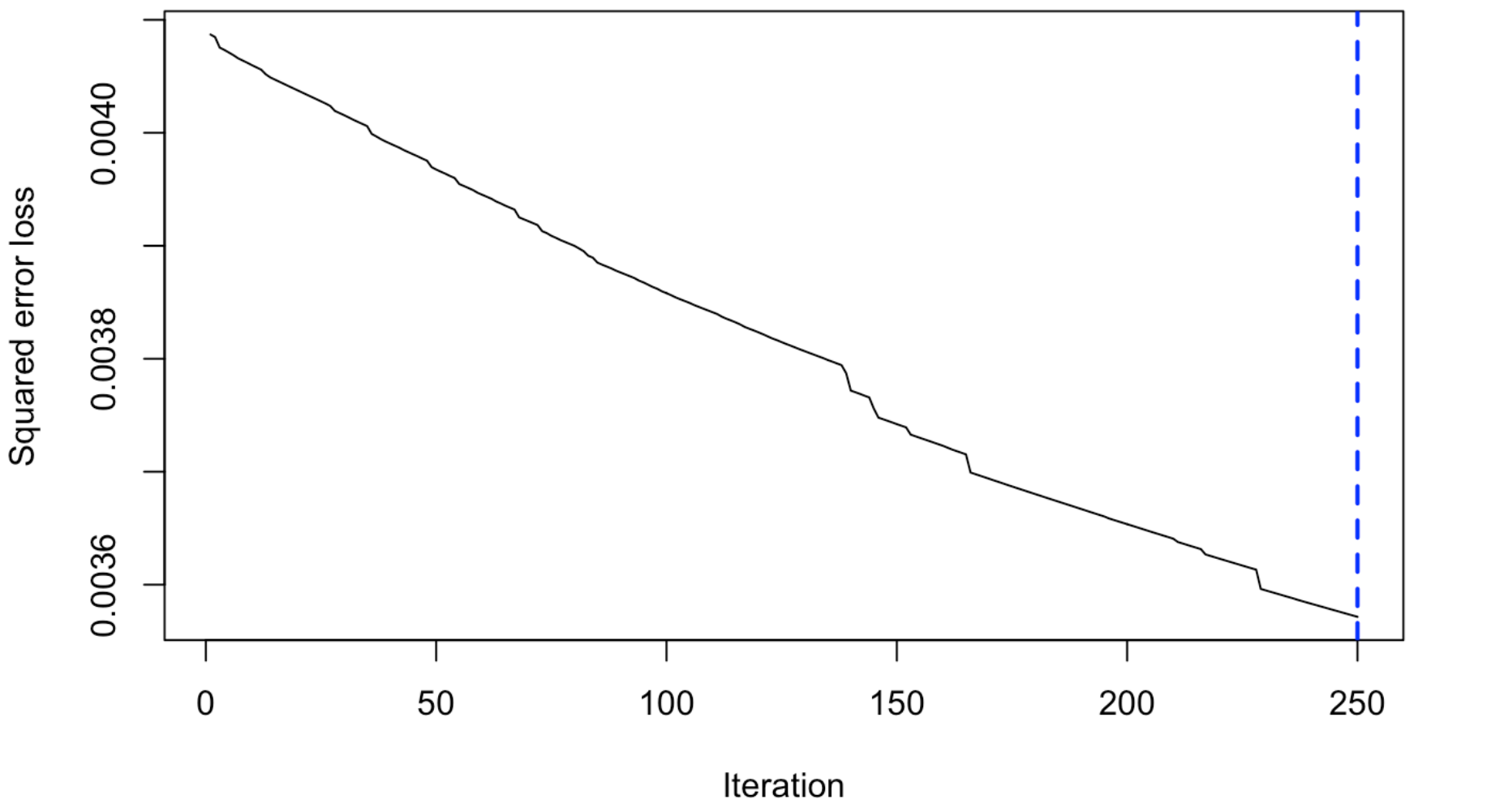
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

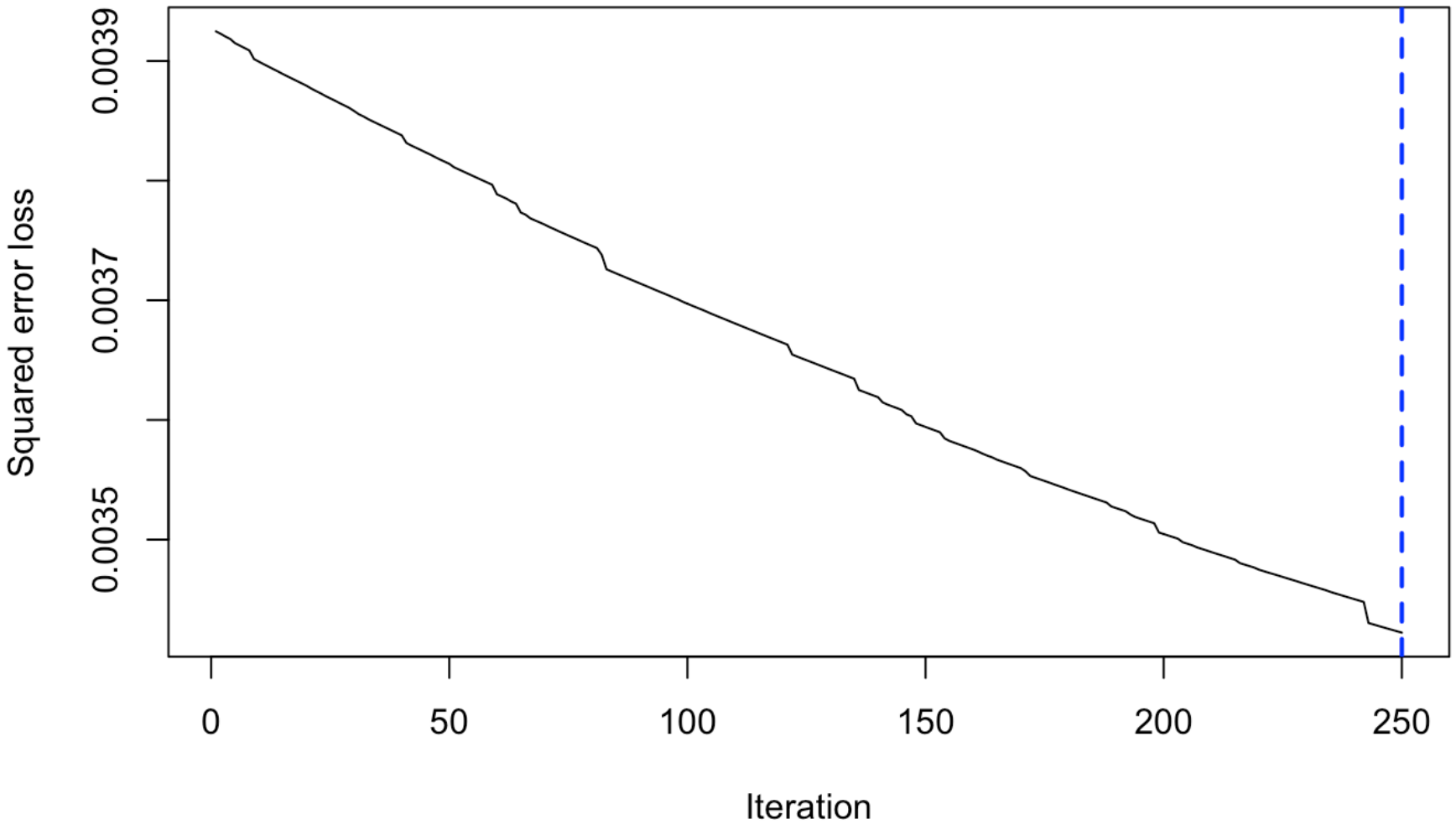

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration

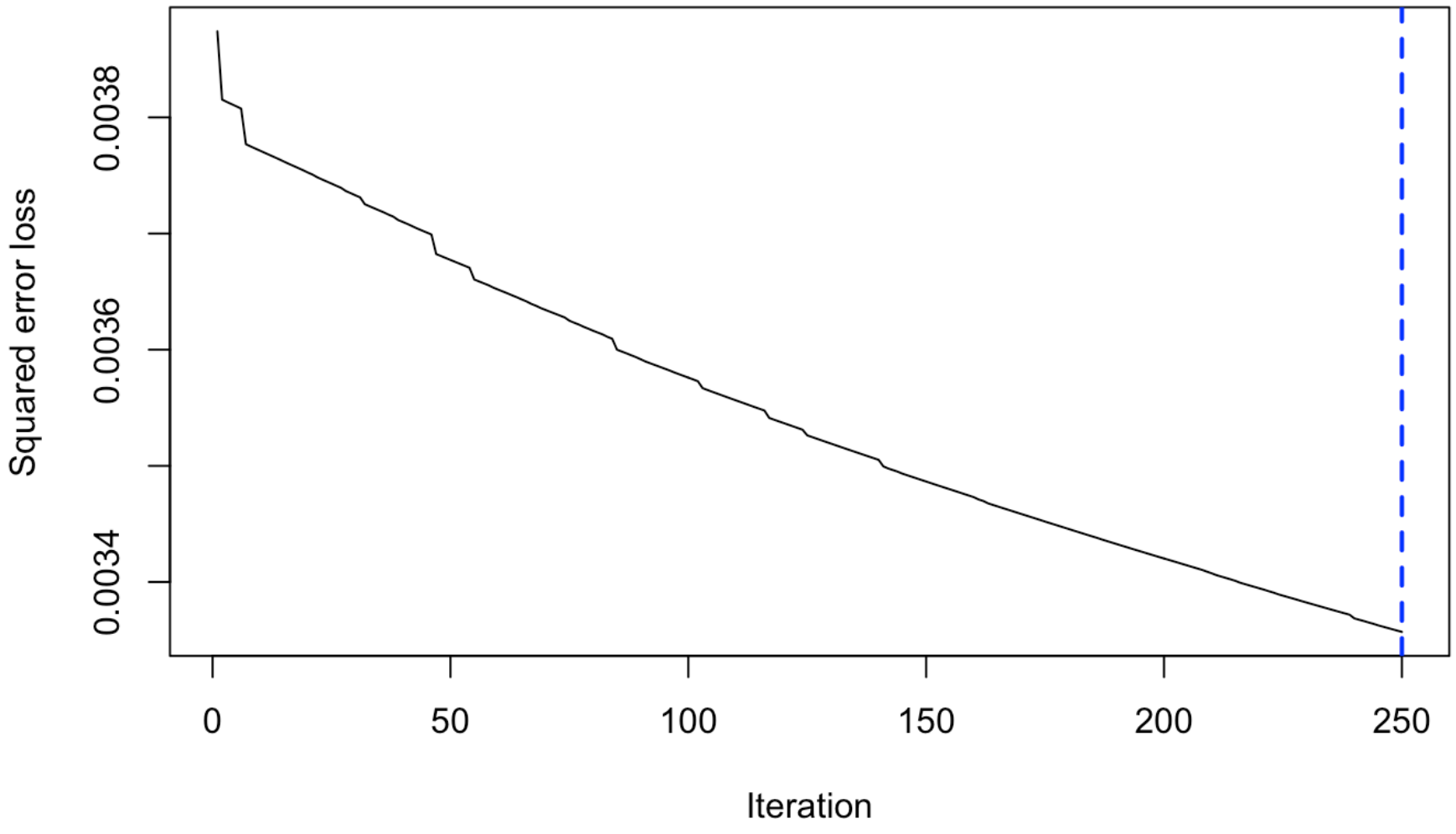

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

```
k=  3
```

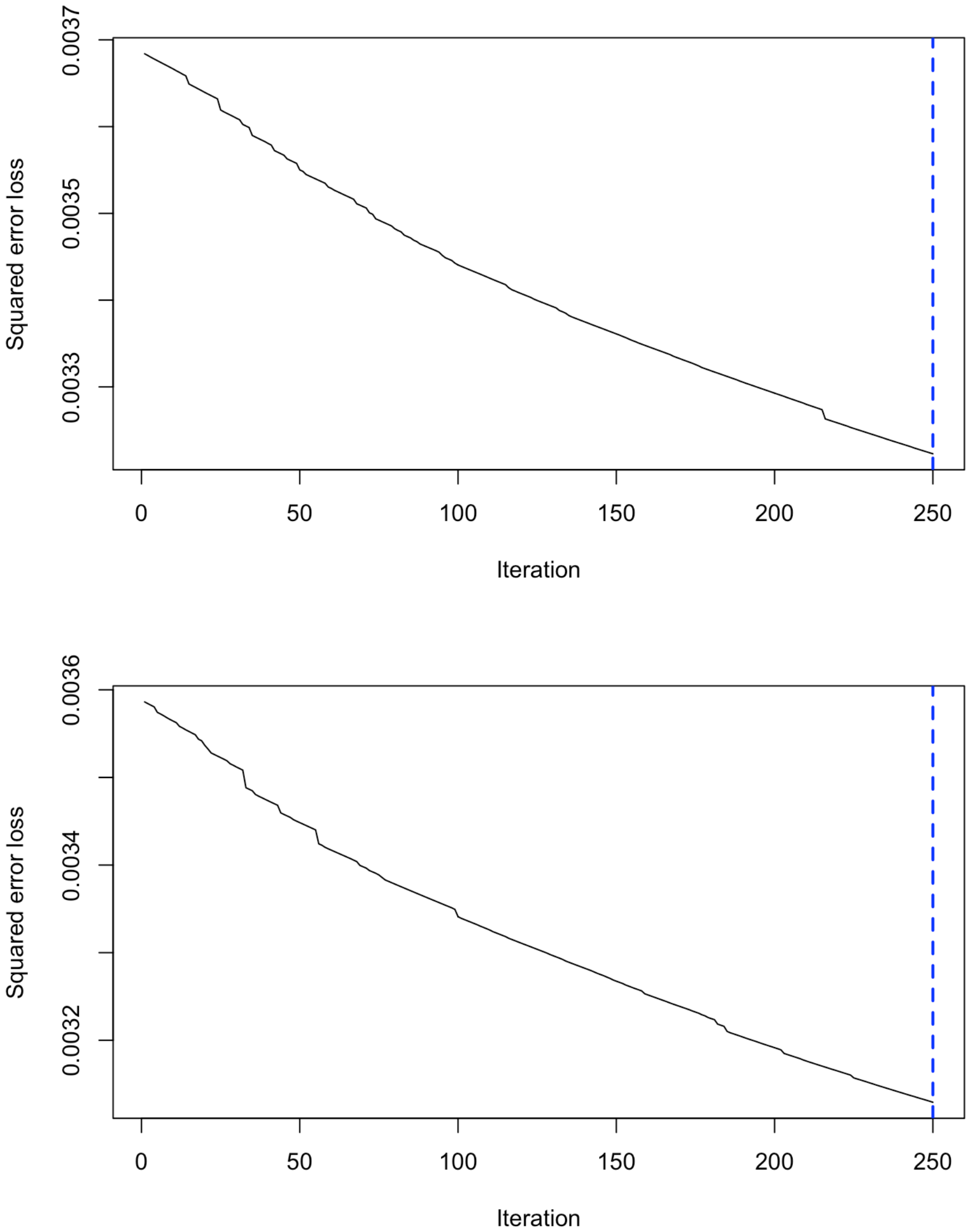

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

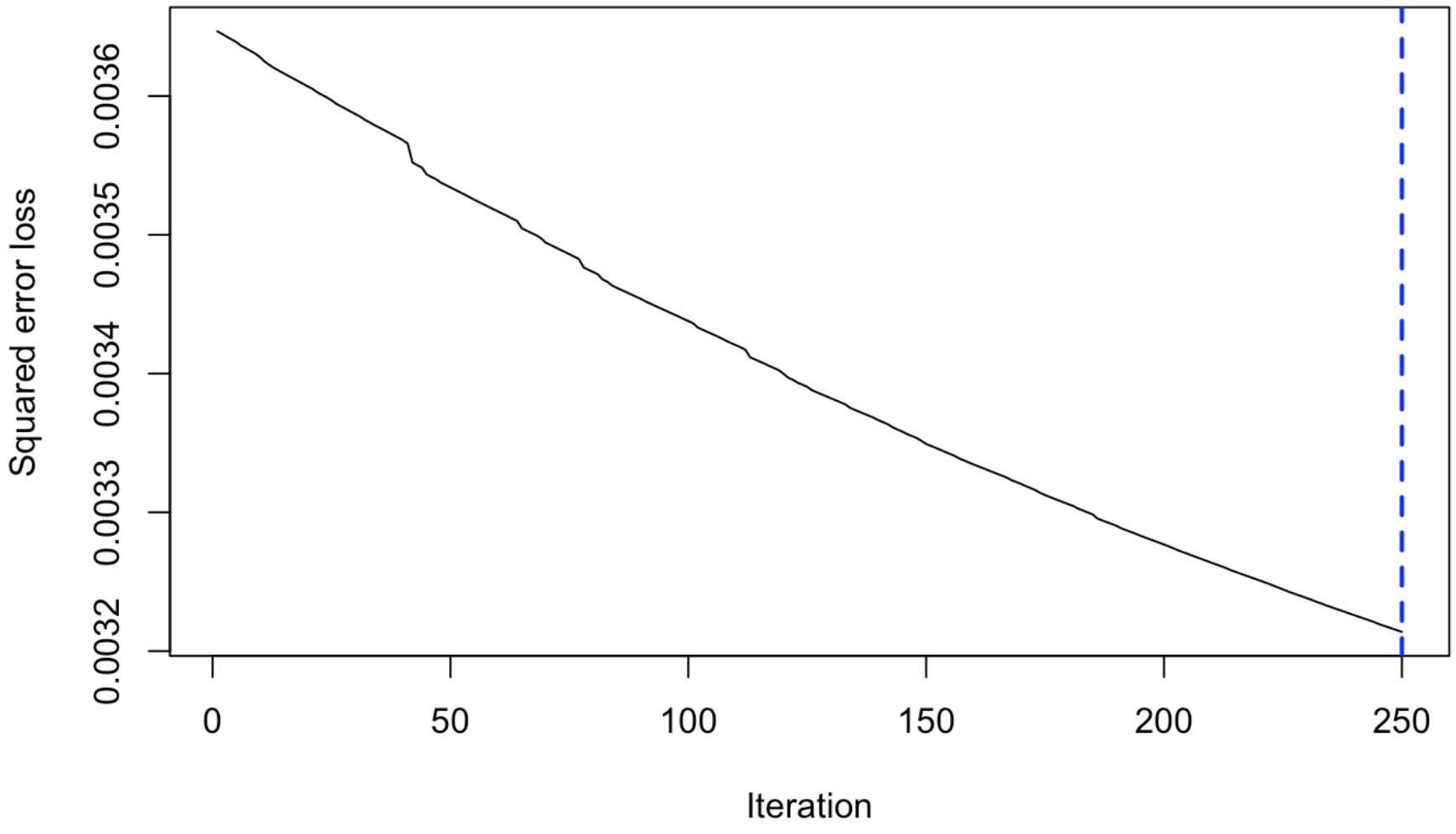

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration

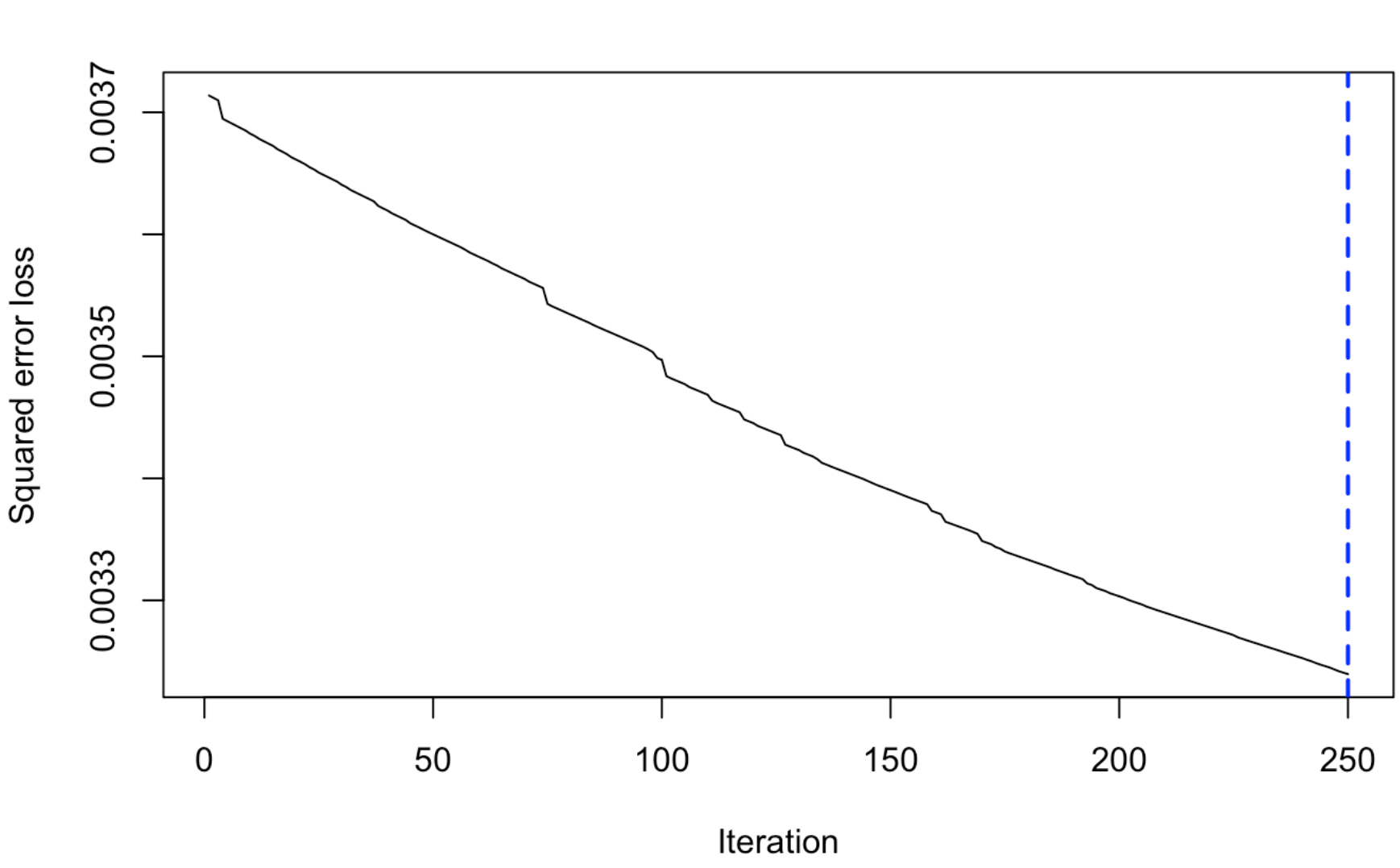
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

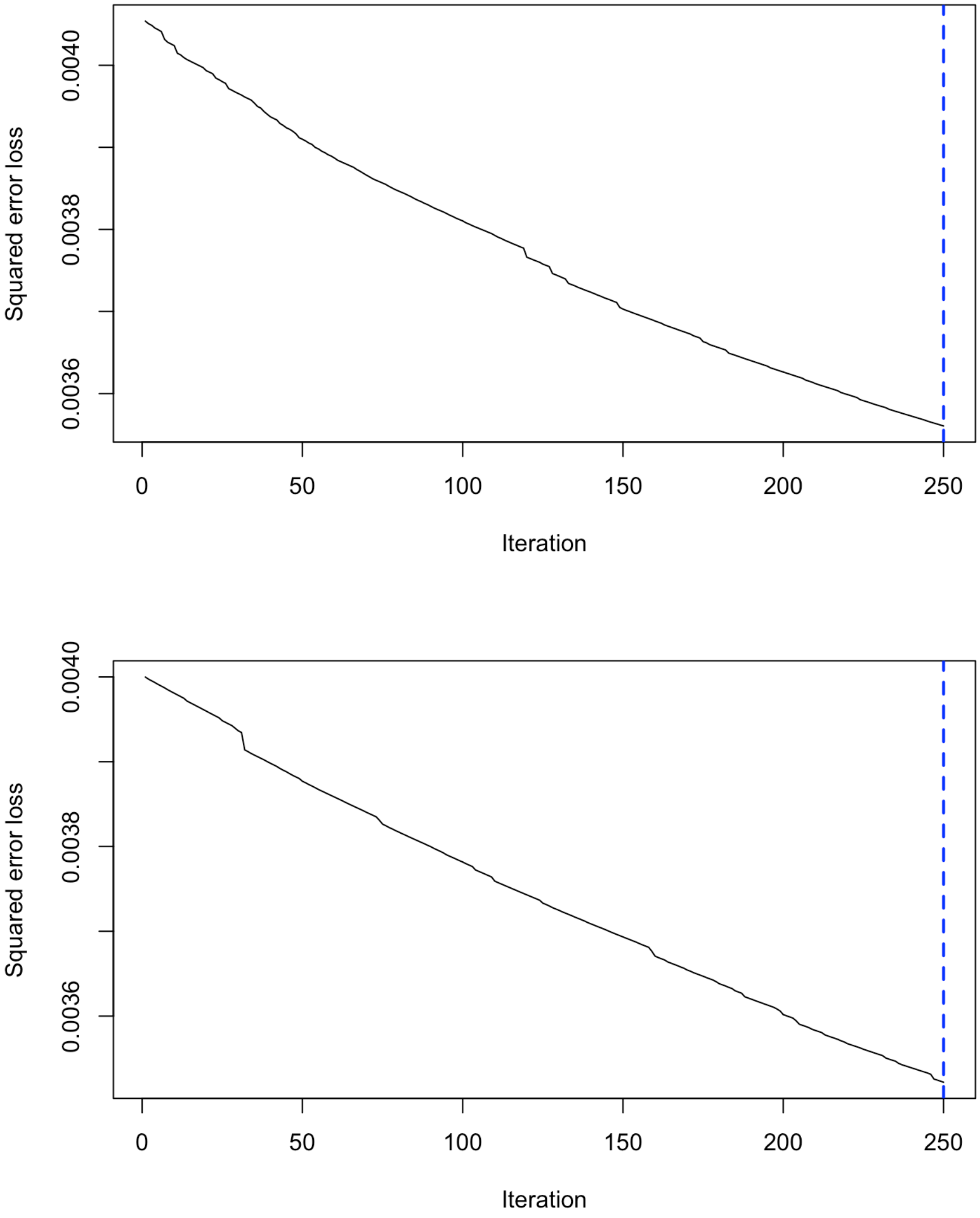

Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration

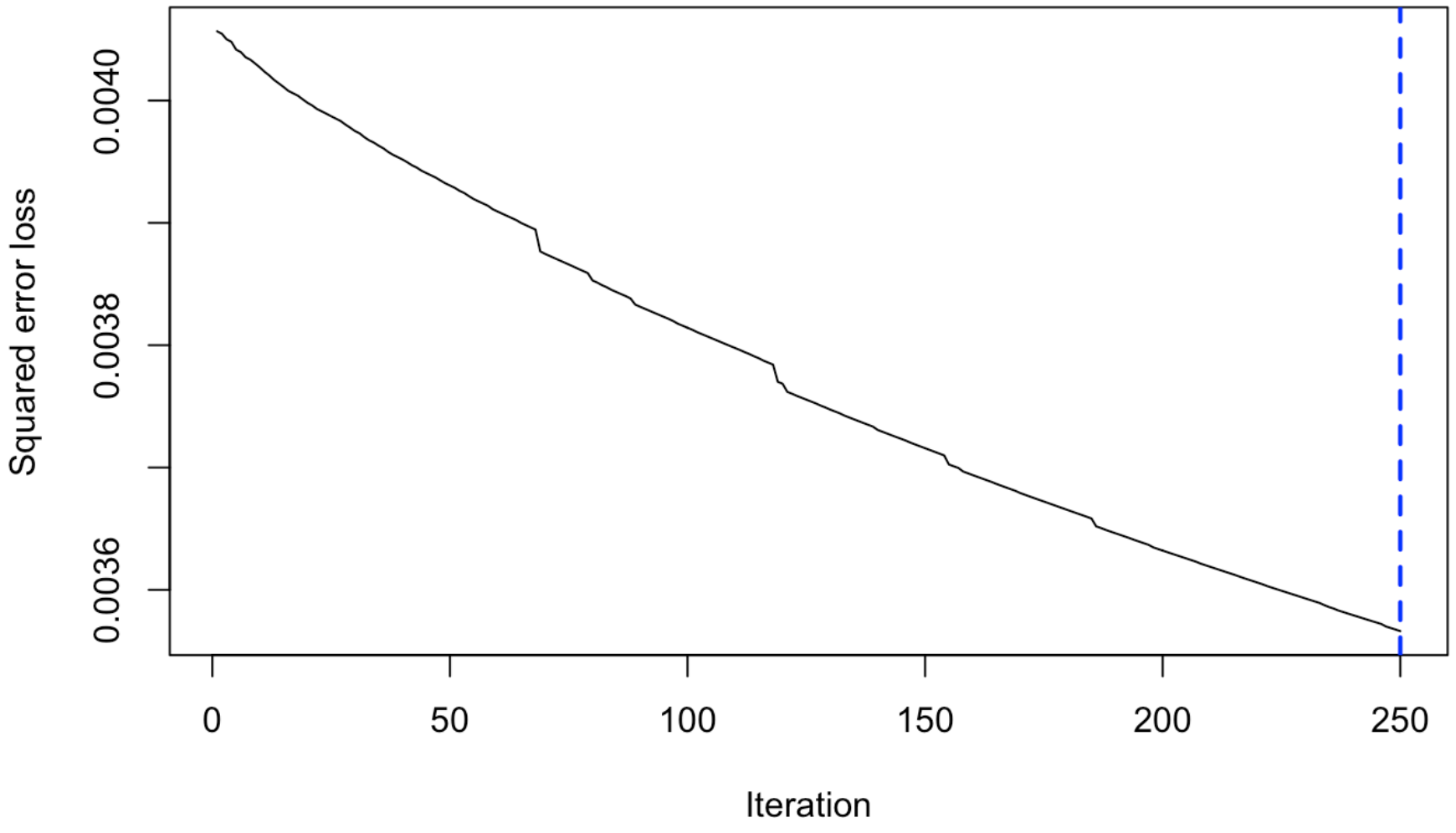
Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration


Squared error loss
0.0037
0.0035
0.0033
0
50
100
150
200
250
Iteration
Squared error loss
0.0034
0.0032
0
50
100
150
200
250
Iteration

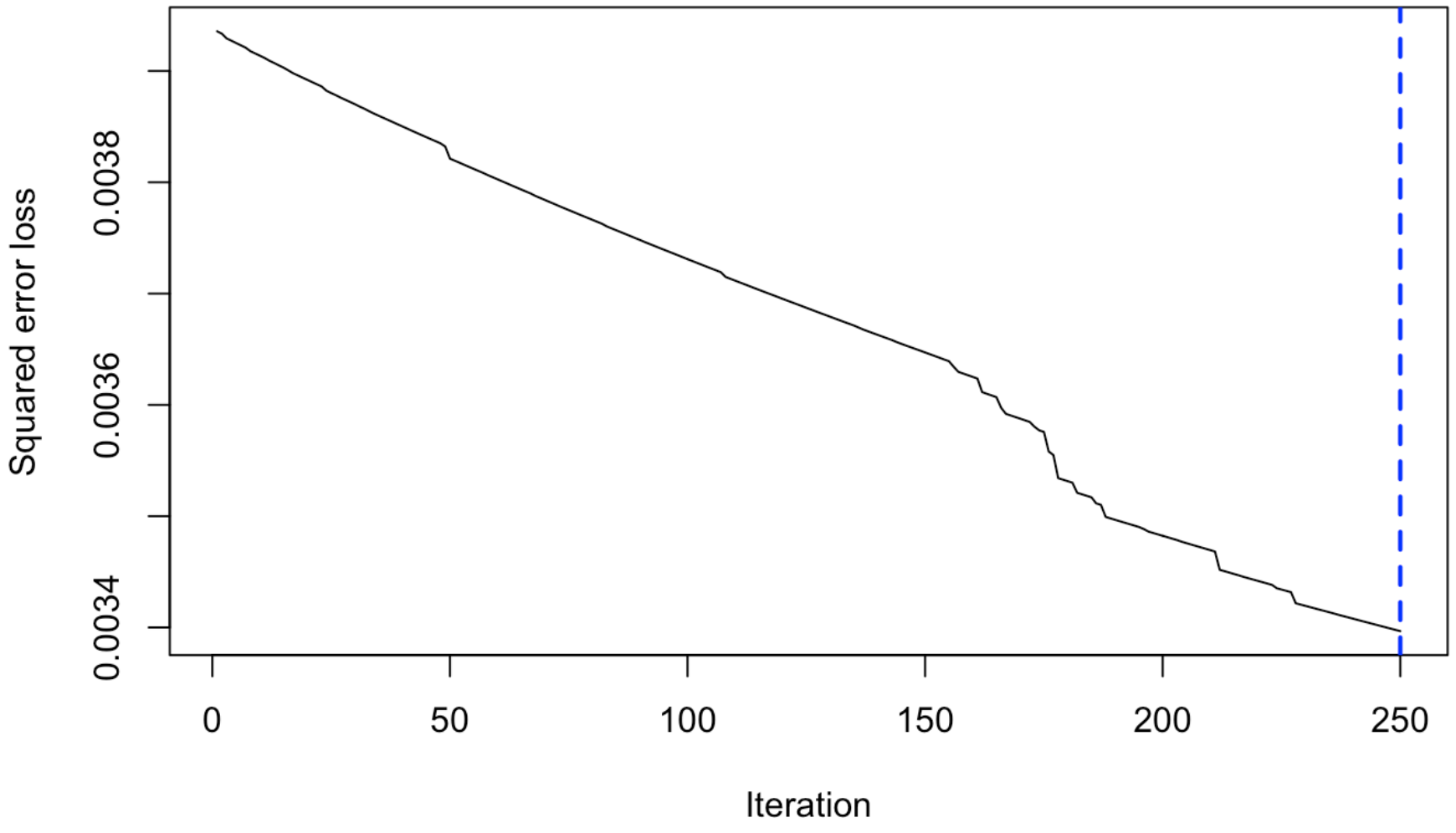

Squared error loss
0.0032
0.0033
0.0034
0.0035
0.0036
0
50
100
150
200
250
Iteration


Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

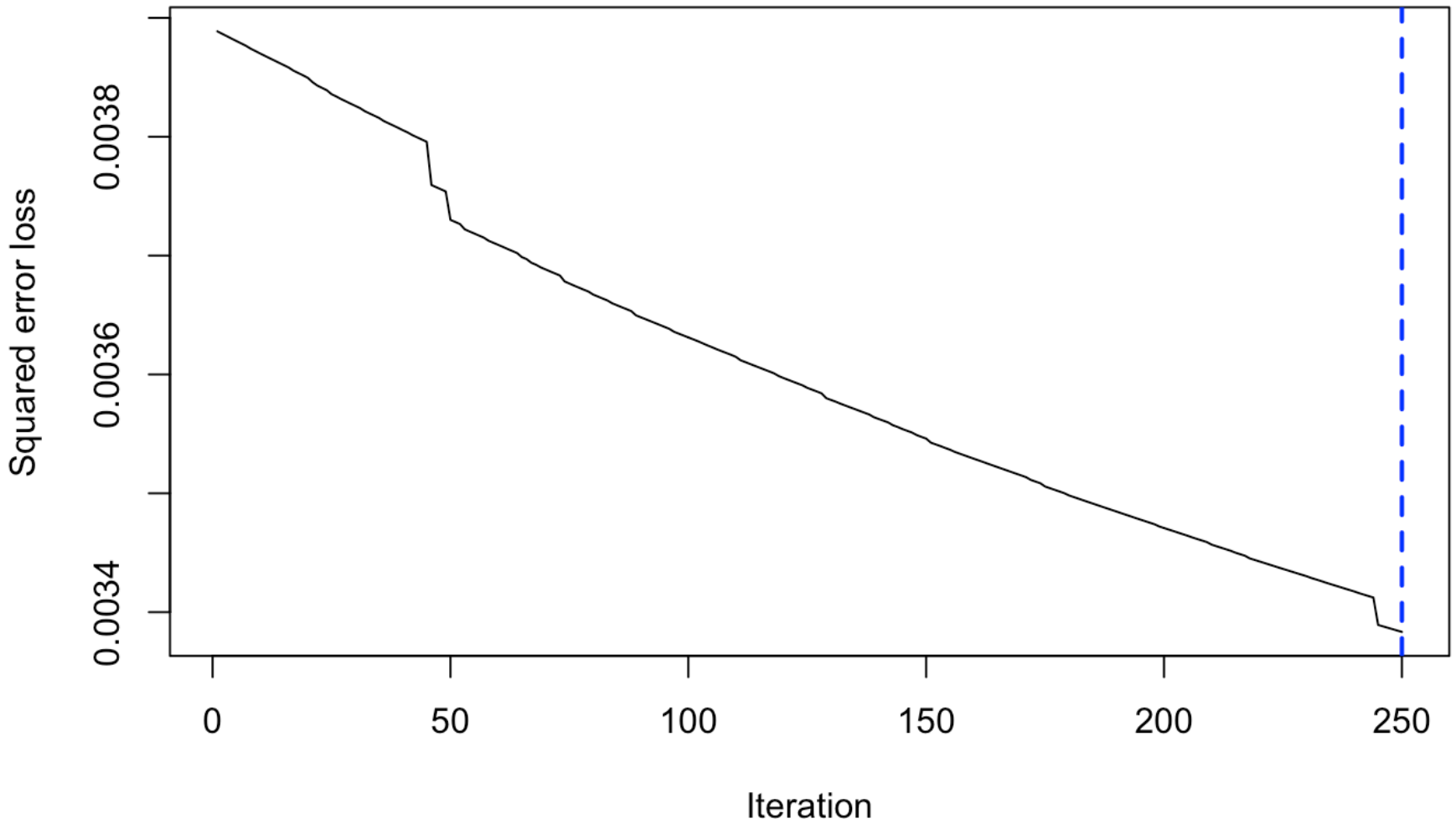

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

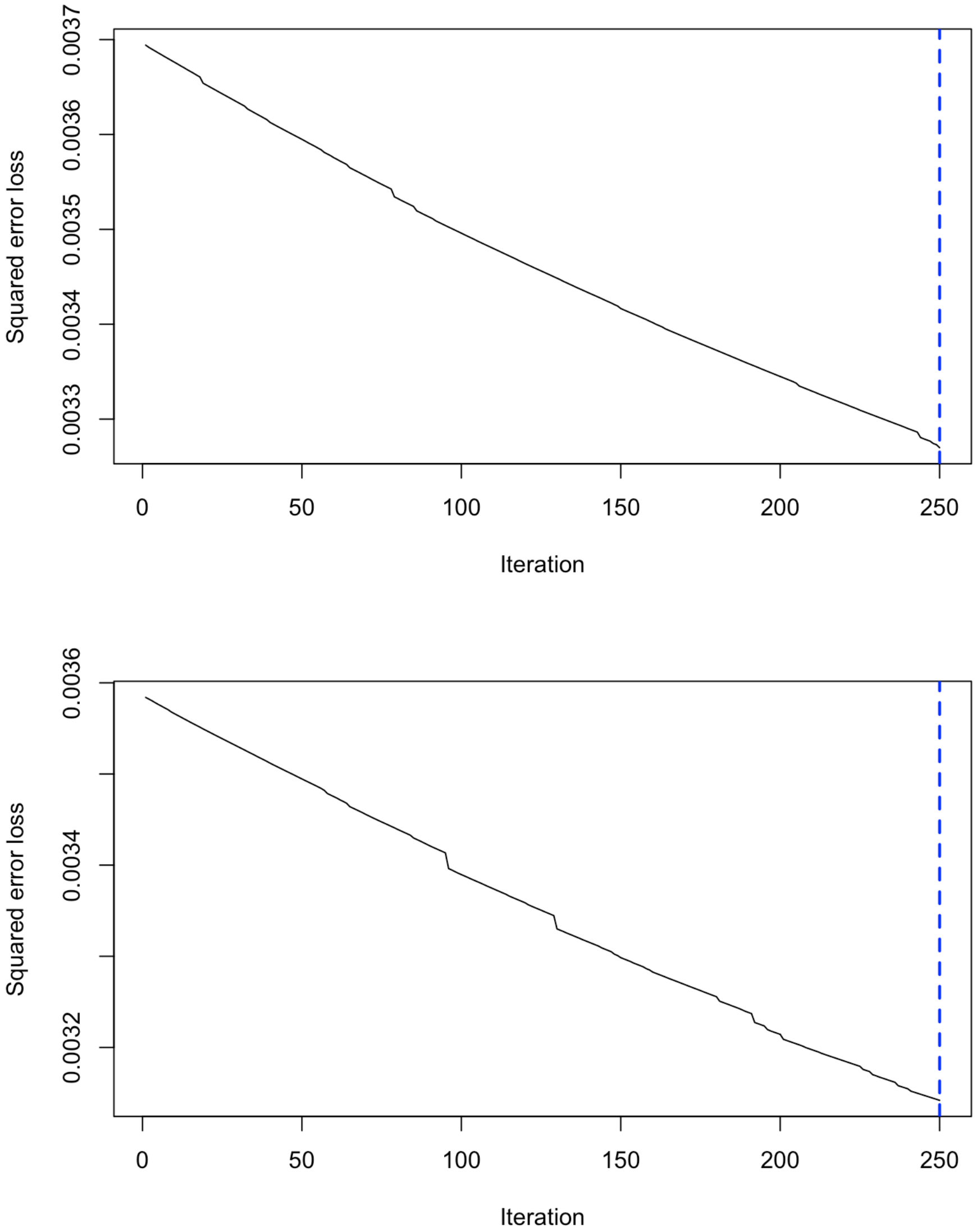
Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

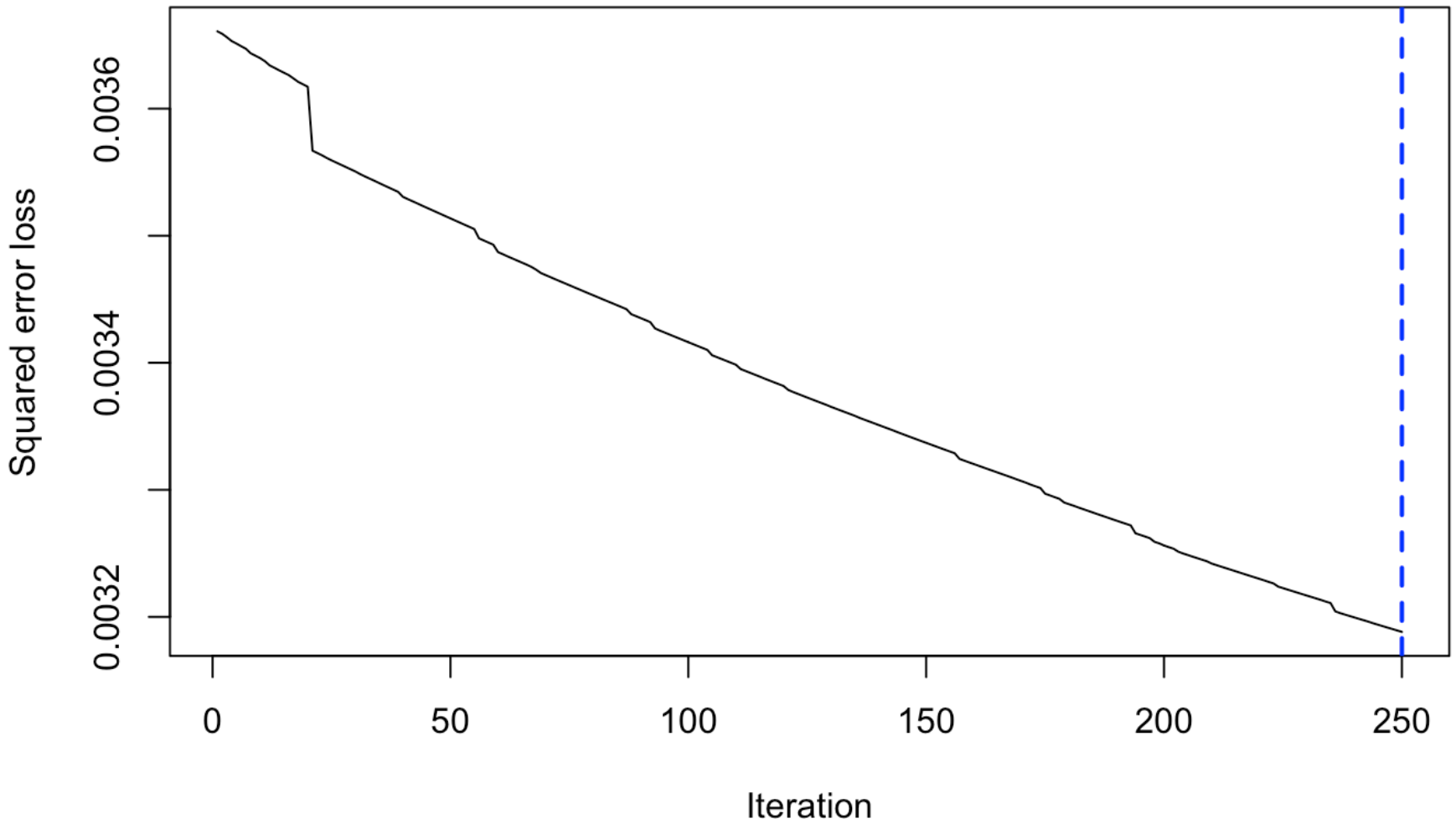

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

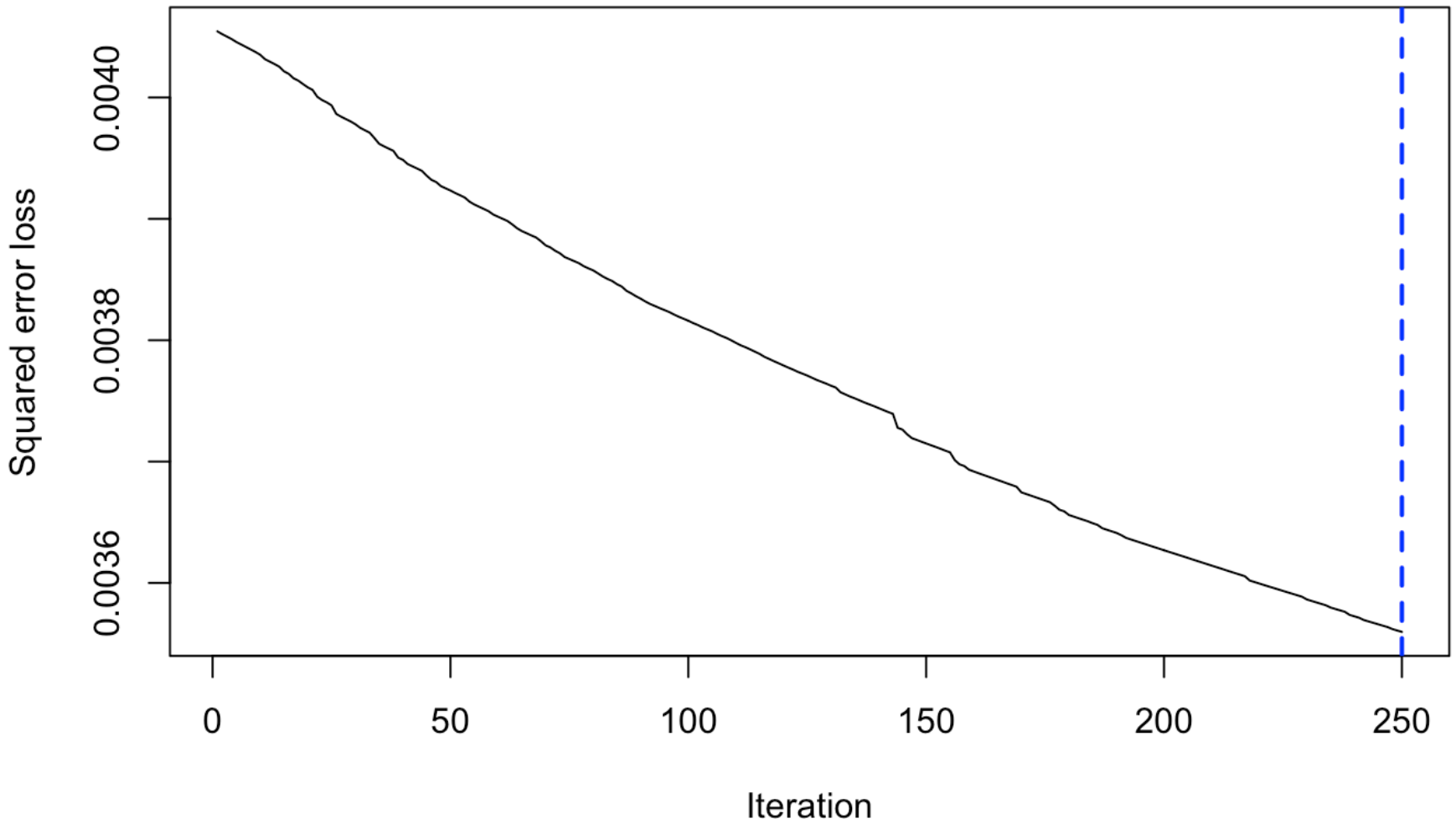

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration

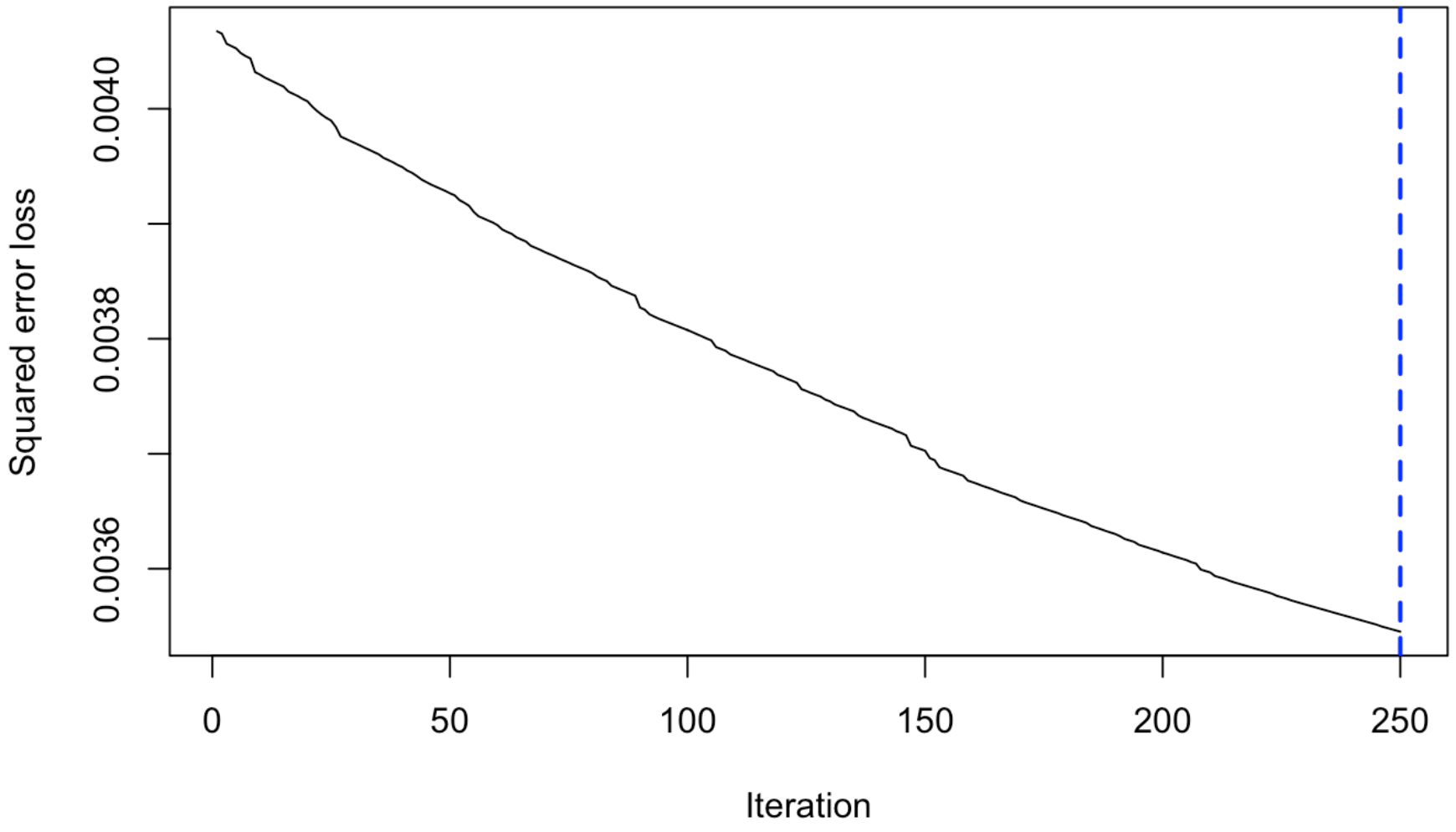

Squared error loss
0.0033
0.0034
0.0035
0.0036
0.0037
0
50
100
150
200
250
Iteration
Squared error loss
0.0032
0.0034
0.003
0
50
100
150
200
250
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
Iteration

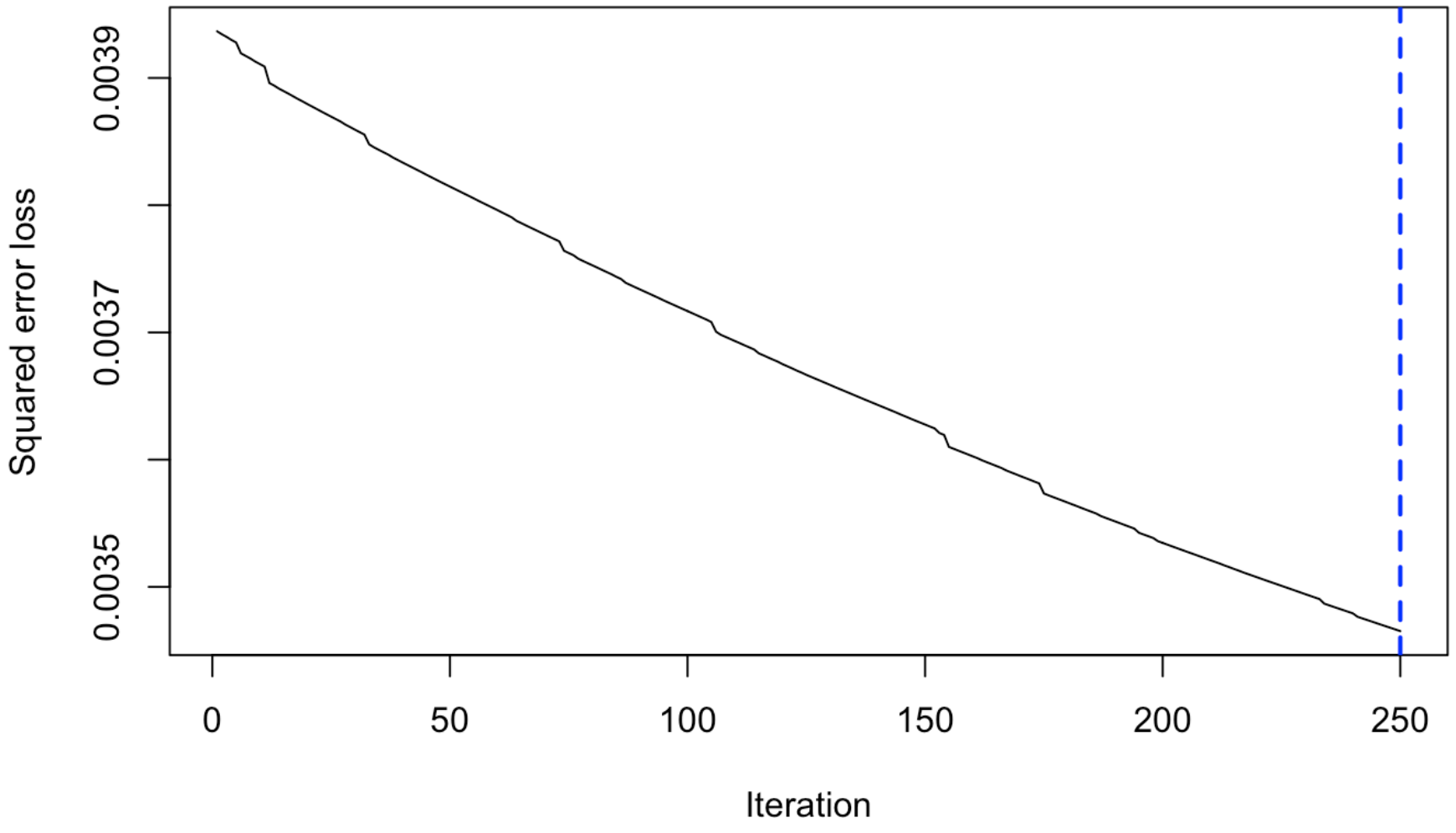
Squared error loss
0.0033
0.0034
0.0035
0.0036
0.0037
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

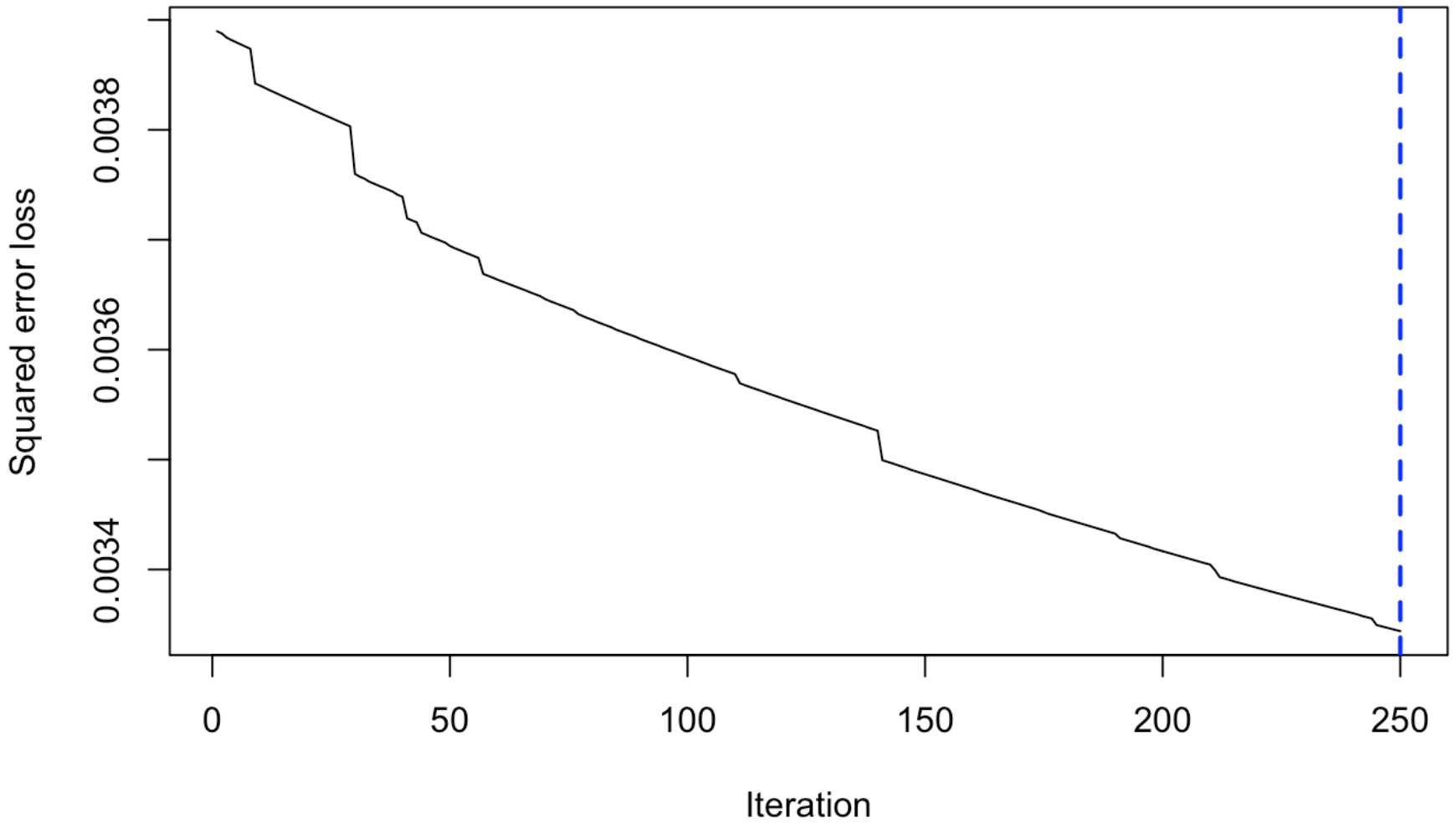
Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration


Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

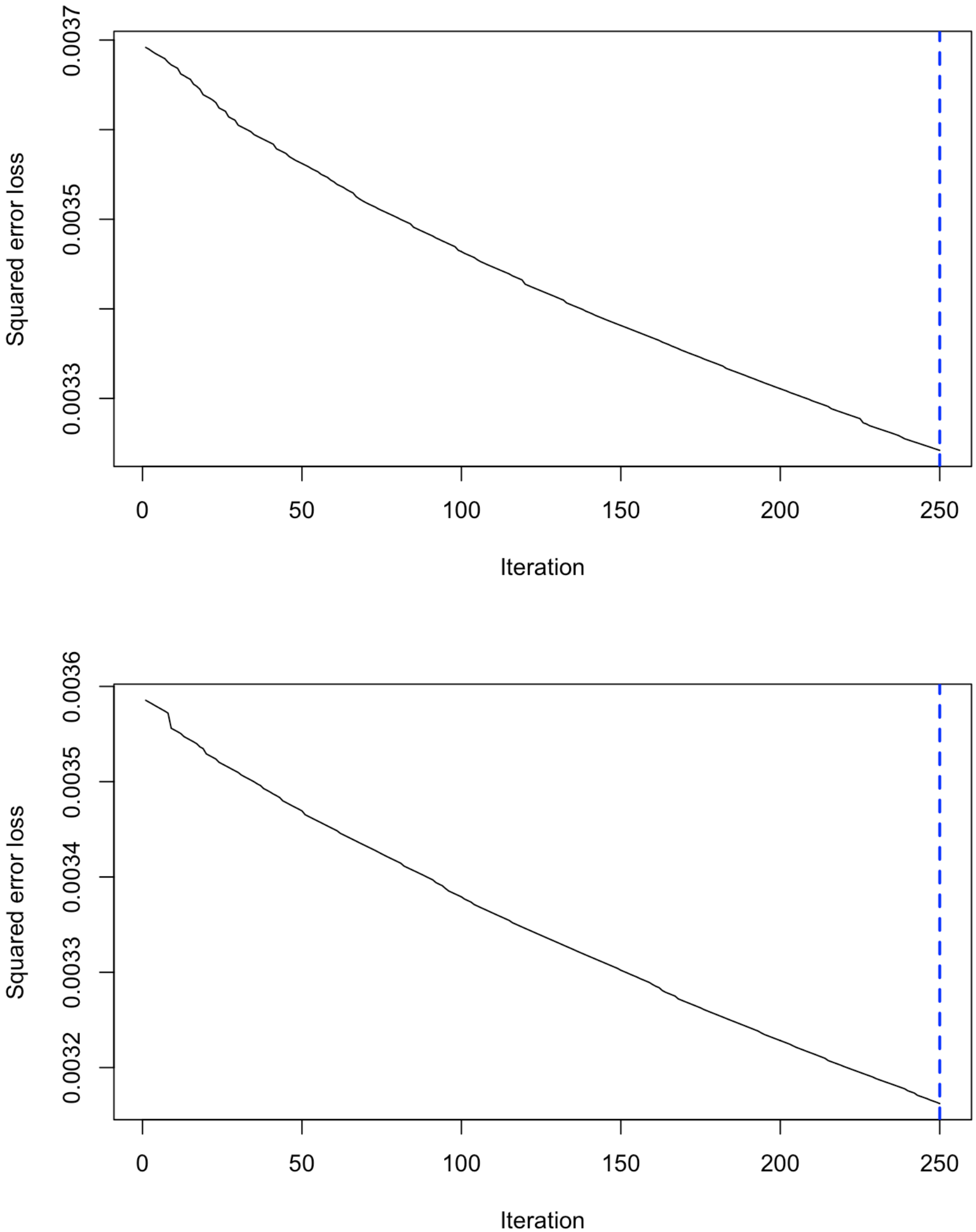
Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

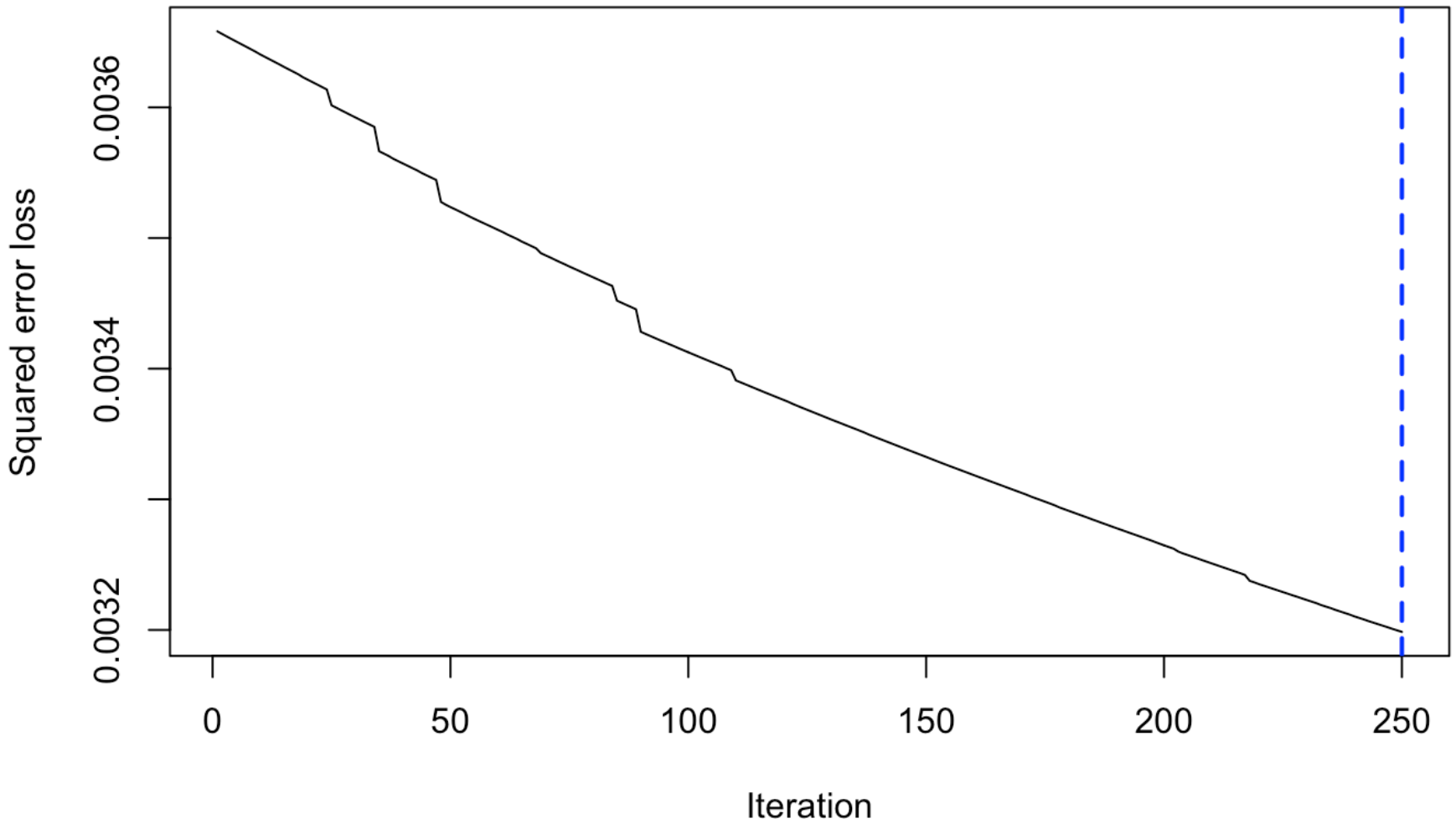

Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
Iteration

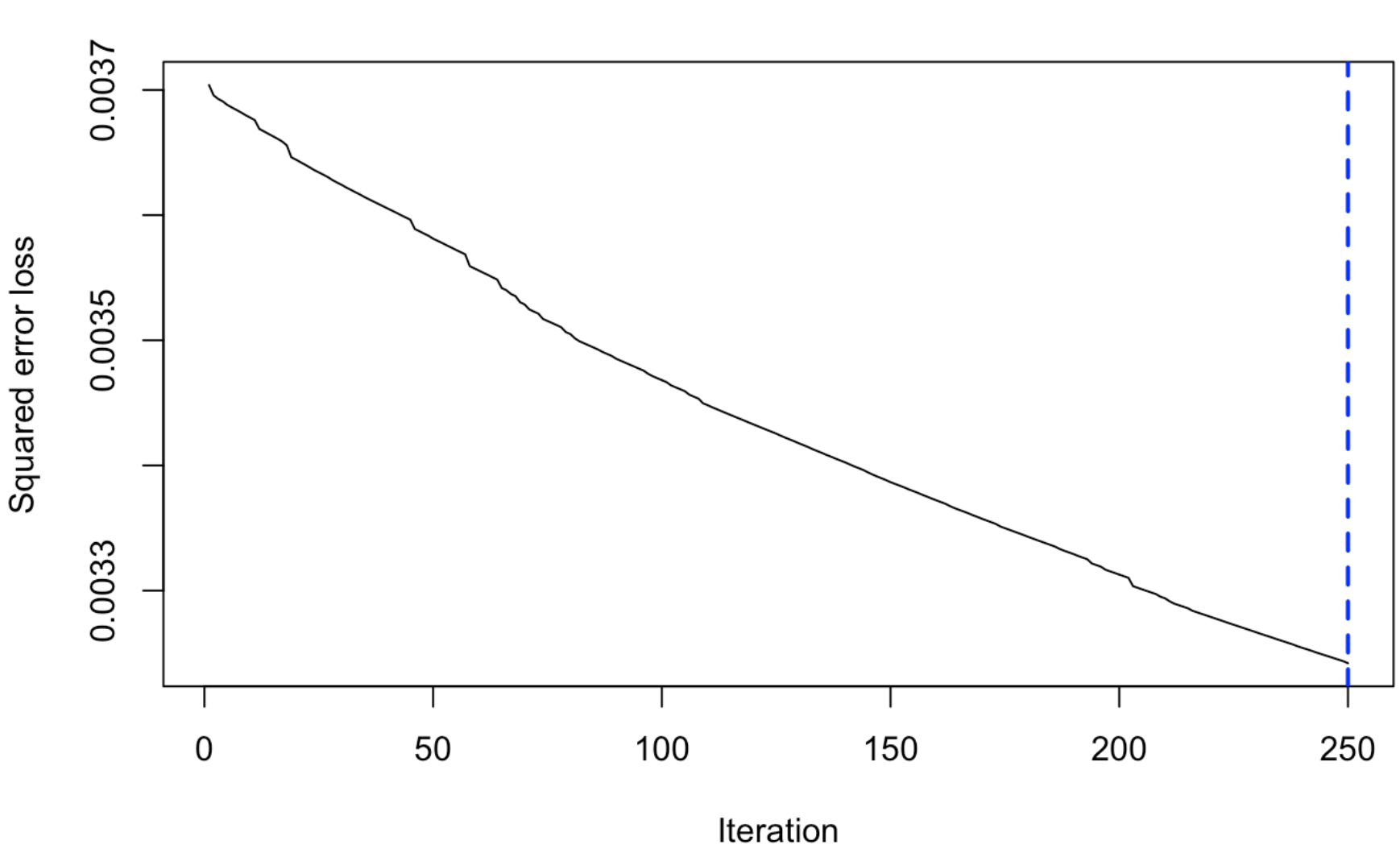
Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
Iteration

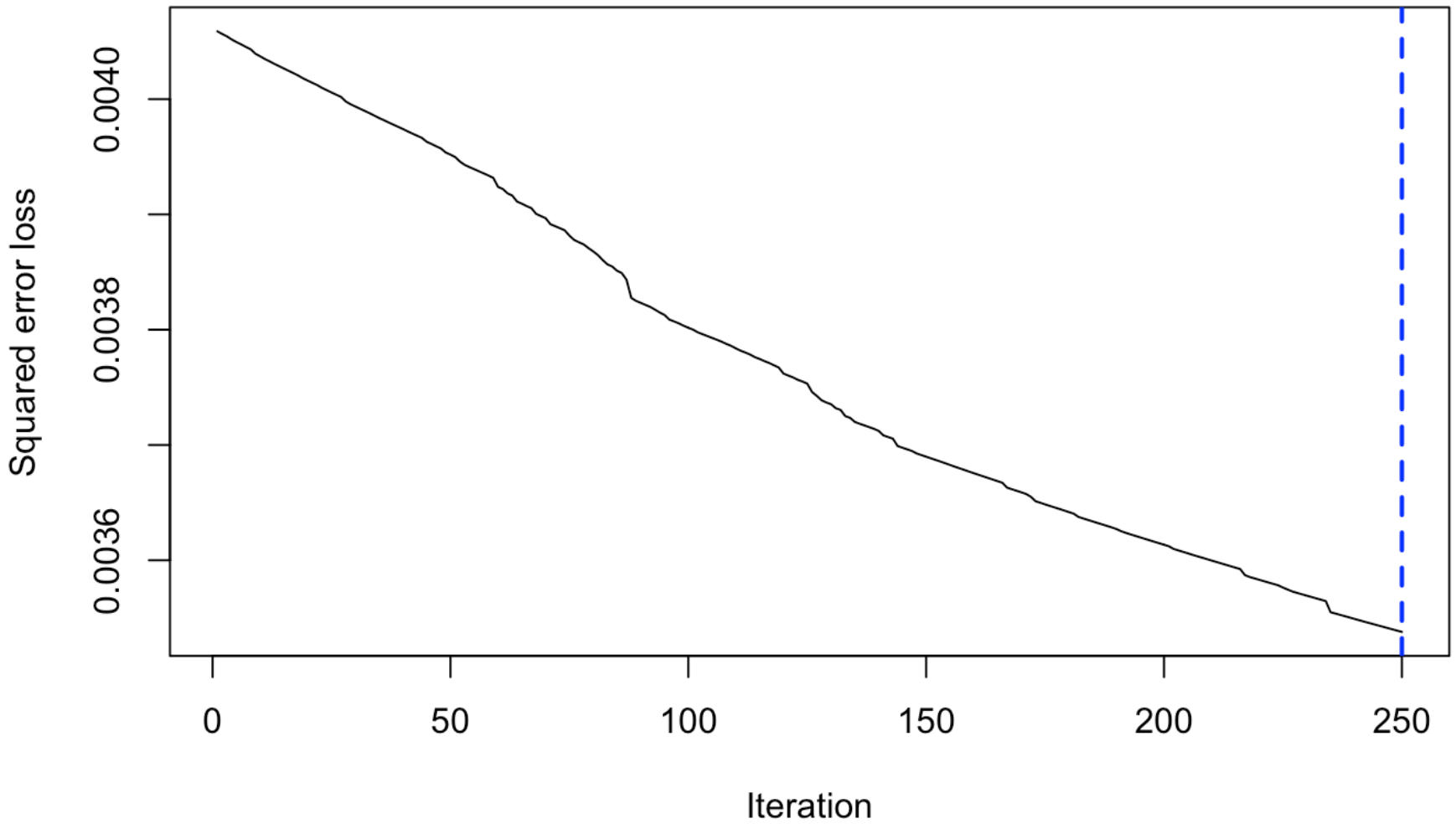

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

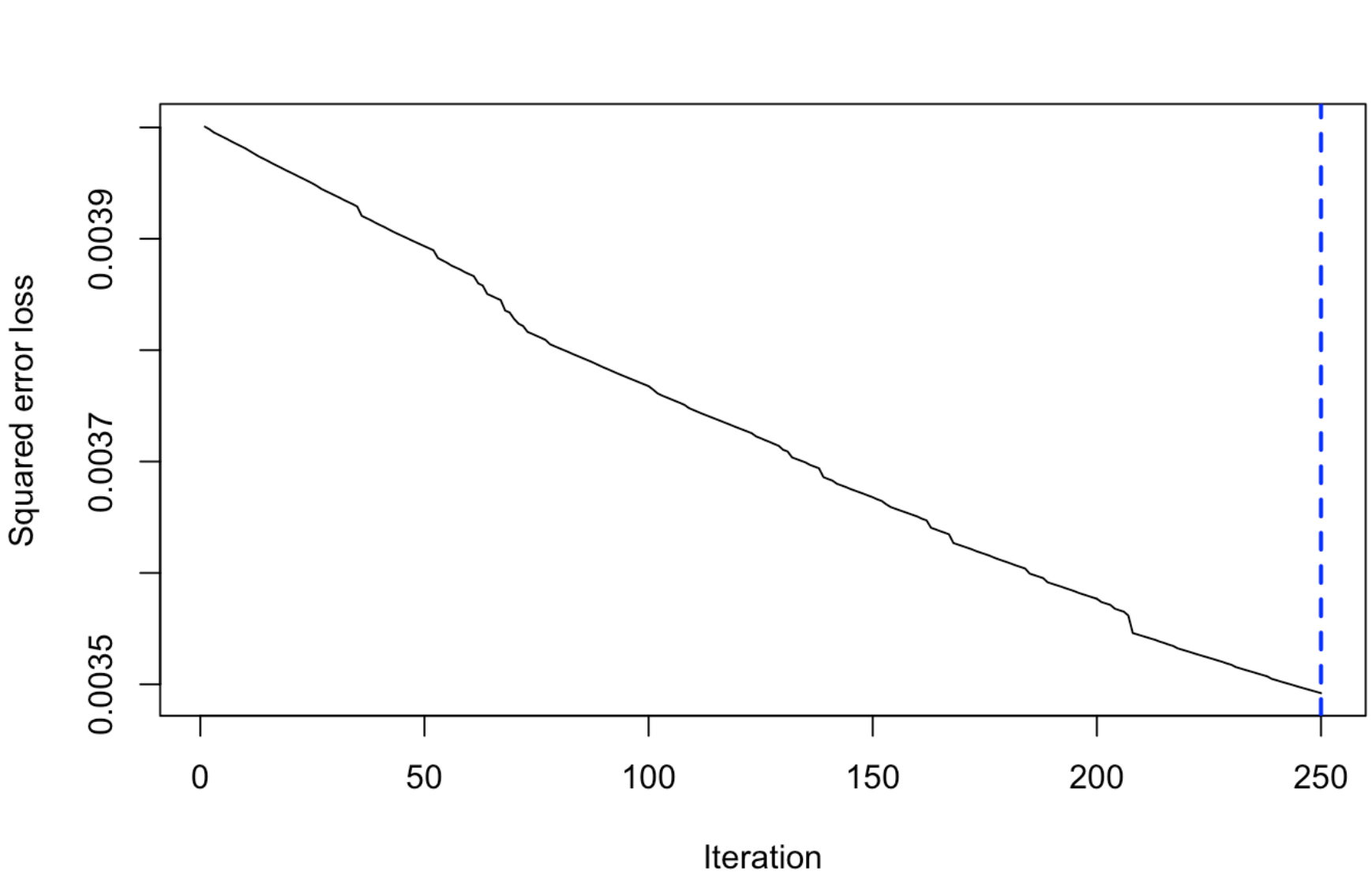
Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
Iteration

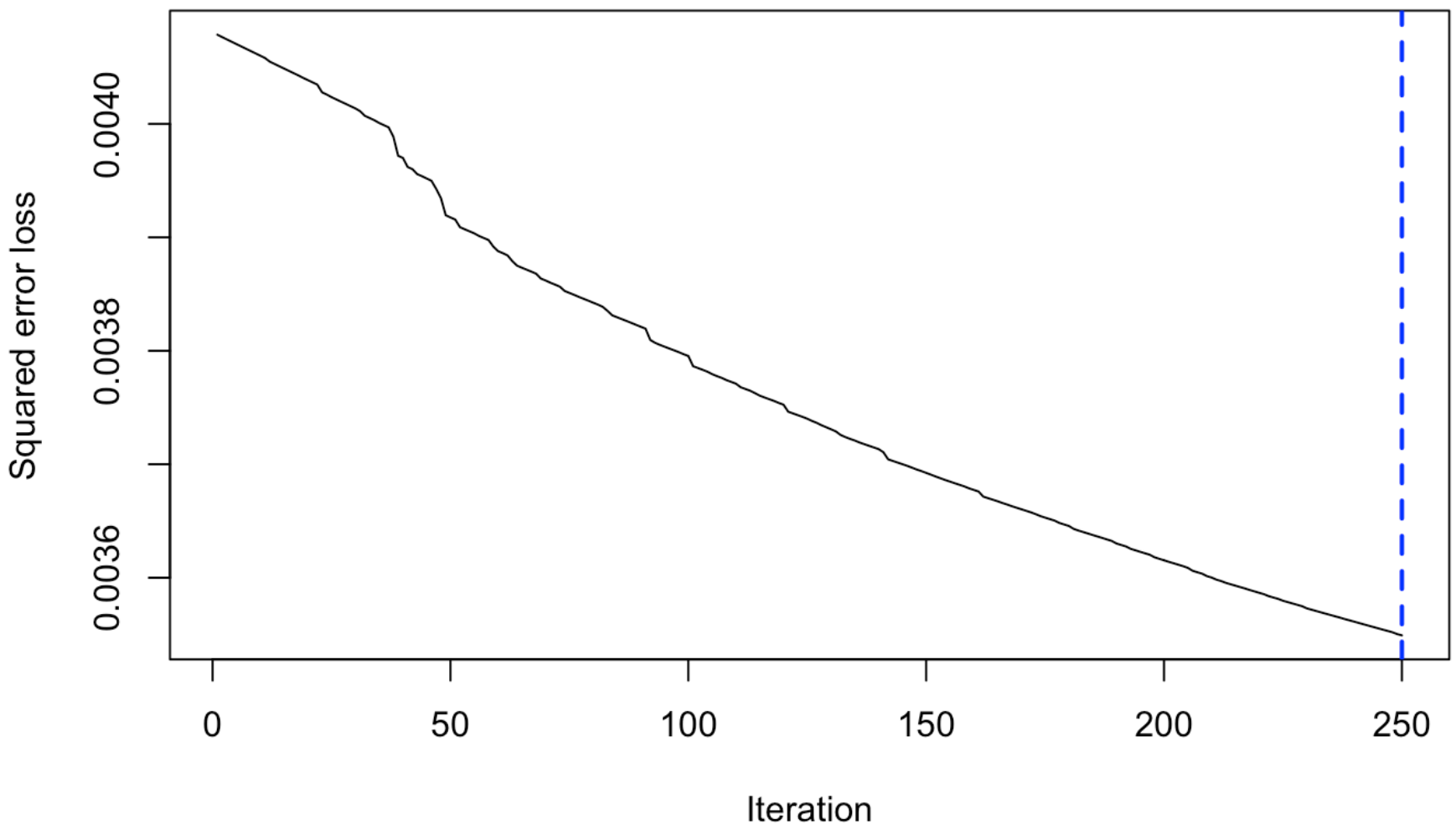

Squared error loss
0.0037
0.0035
0.0033
0
50
100
150
200
250
Iteration
Squared error loss
0.0036
0.0035
0.0034
0.0033
0.0032
0
50
100
150
200
250
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
Iteration

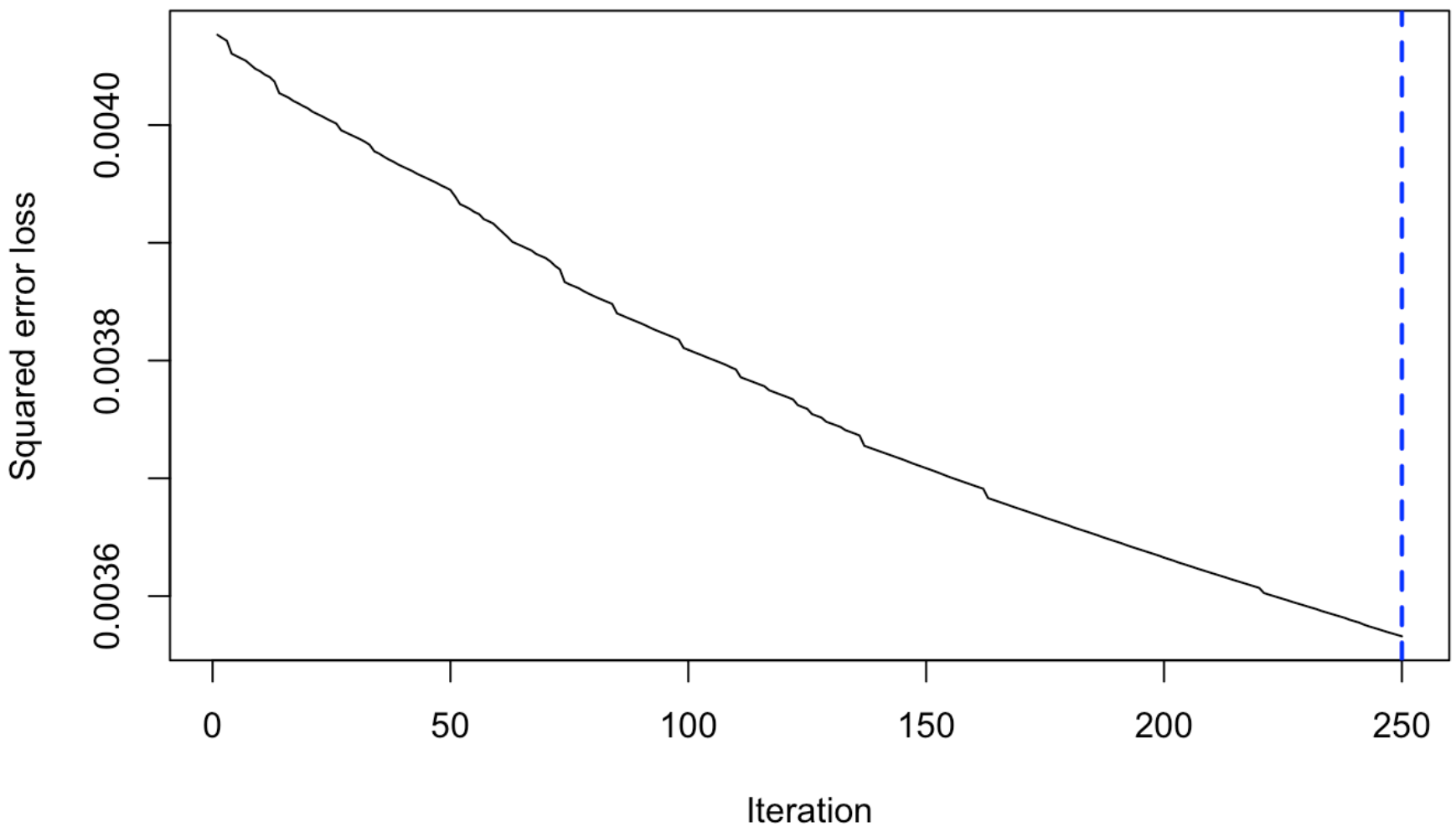

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
Iteration

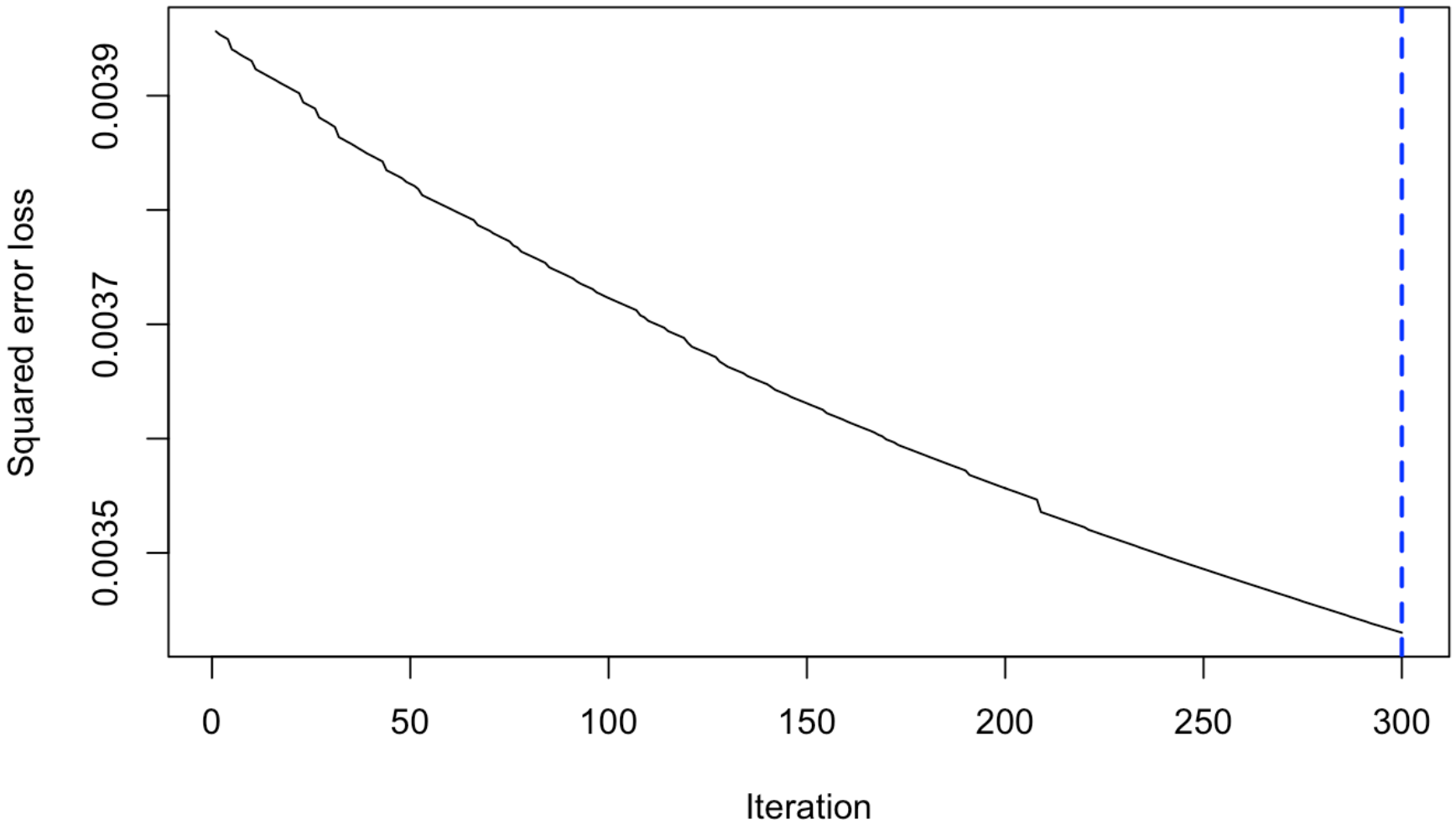

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
Iteration

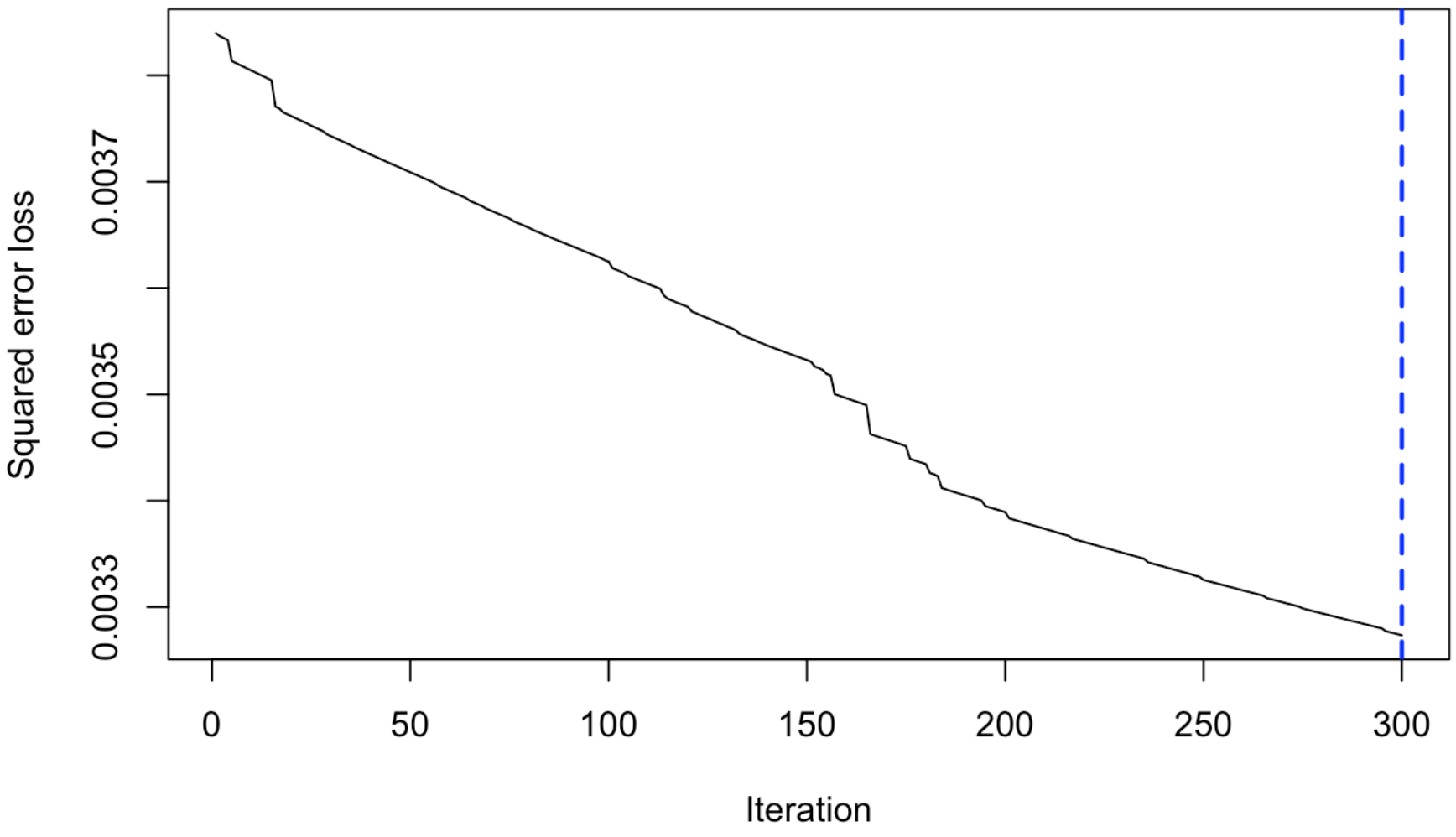

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
Iteration

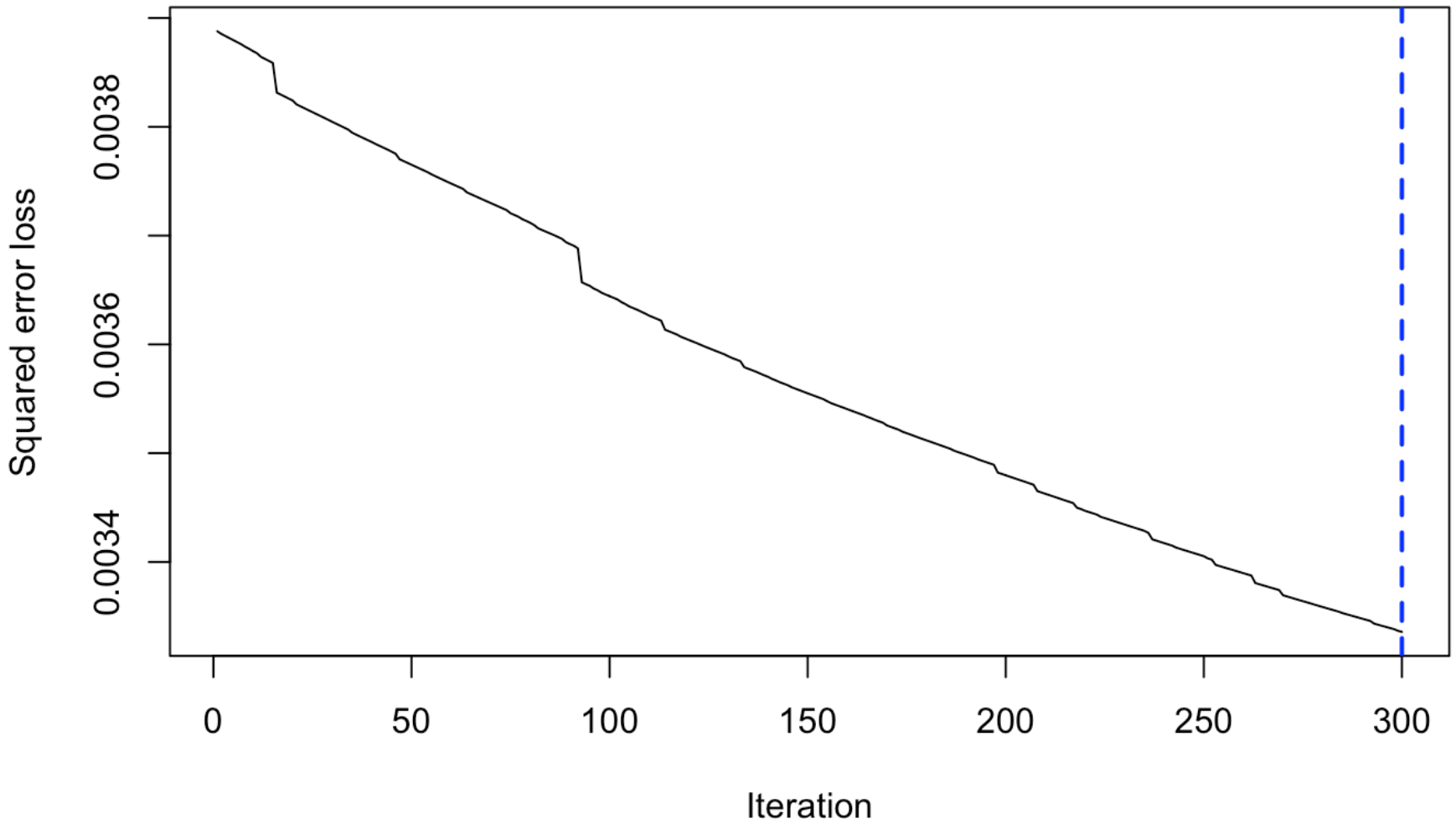

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

```
k= 4
```

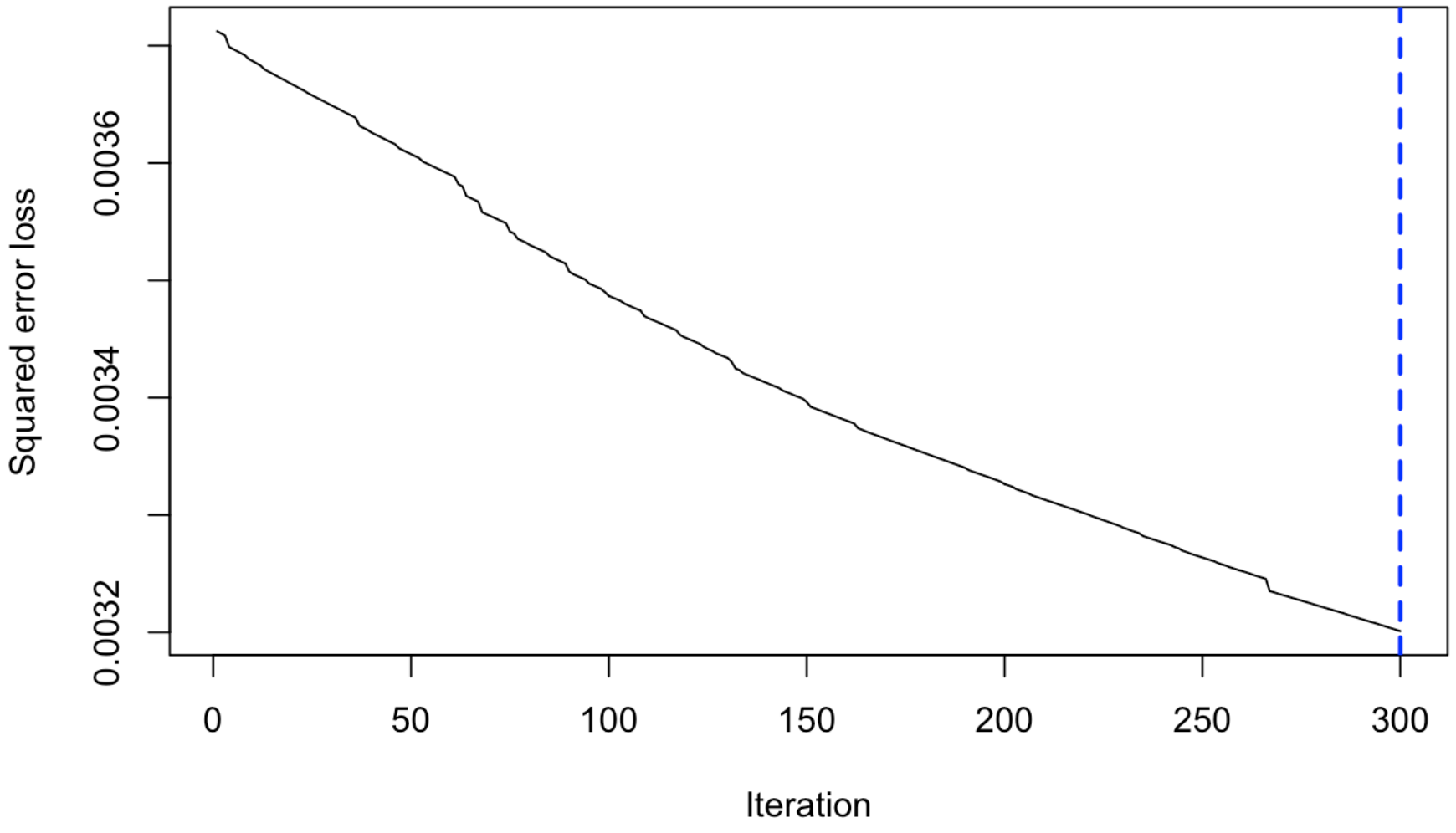

Squared error loss
0.0040
0.0038
0.0036
0
50
100
150
200
250
Iteration

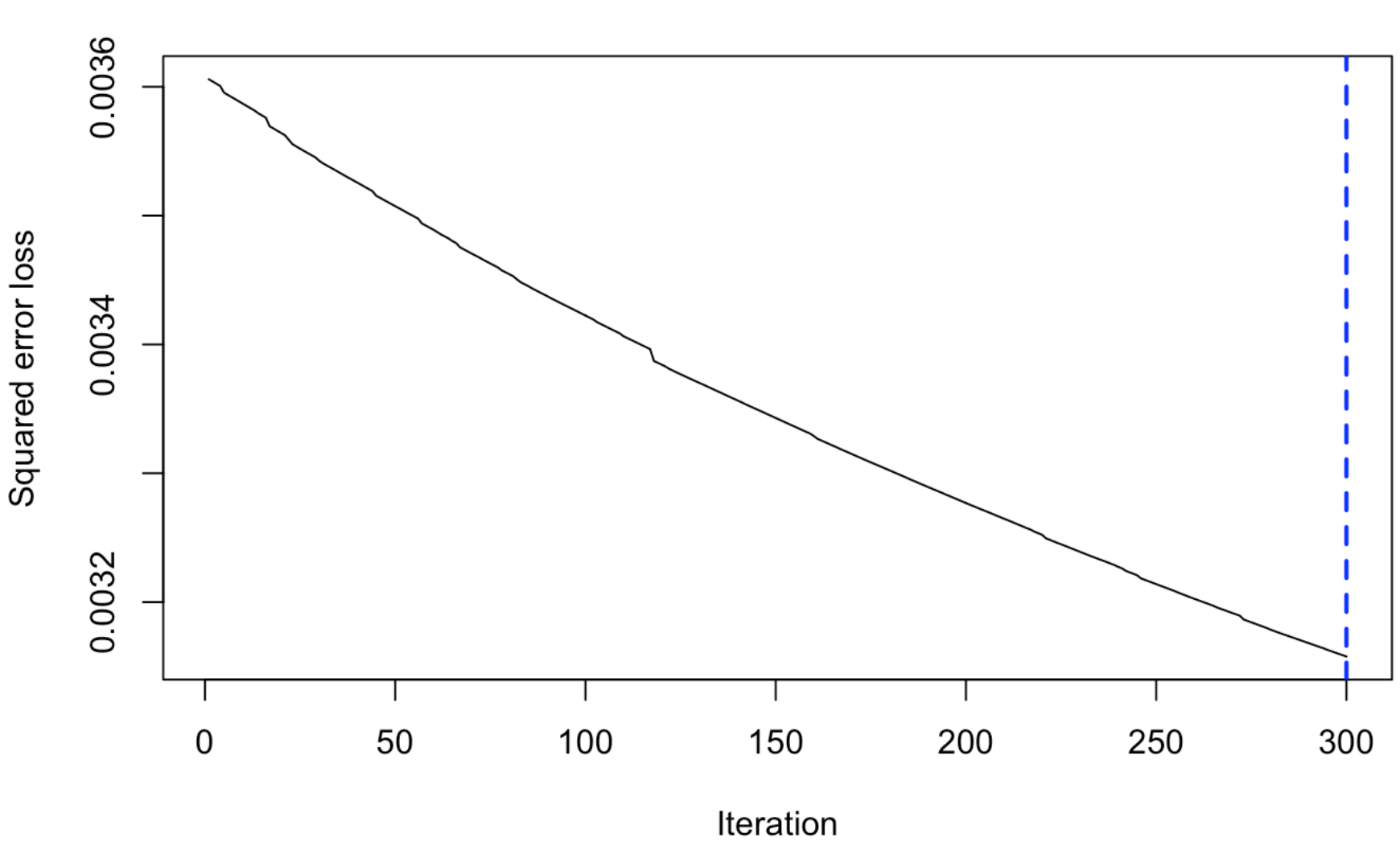

Squared error loss
0.0039
0.0037
0.0035
0
50
100
150
200
250
300
Iteration

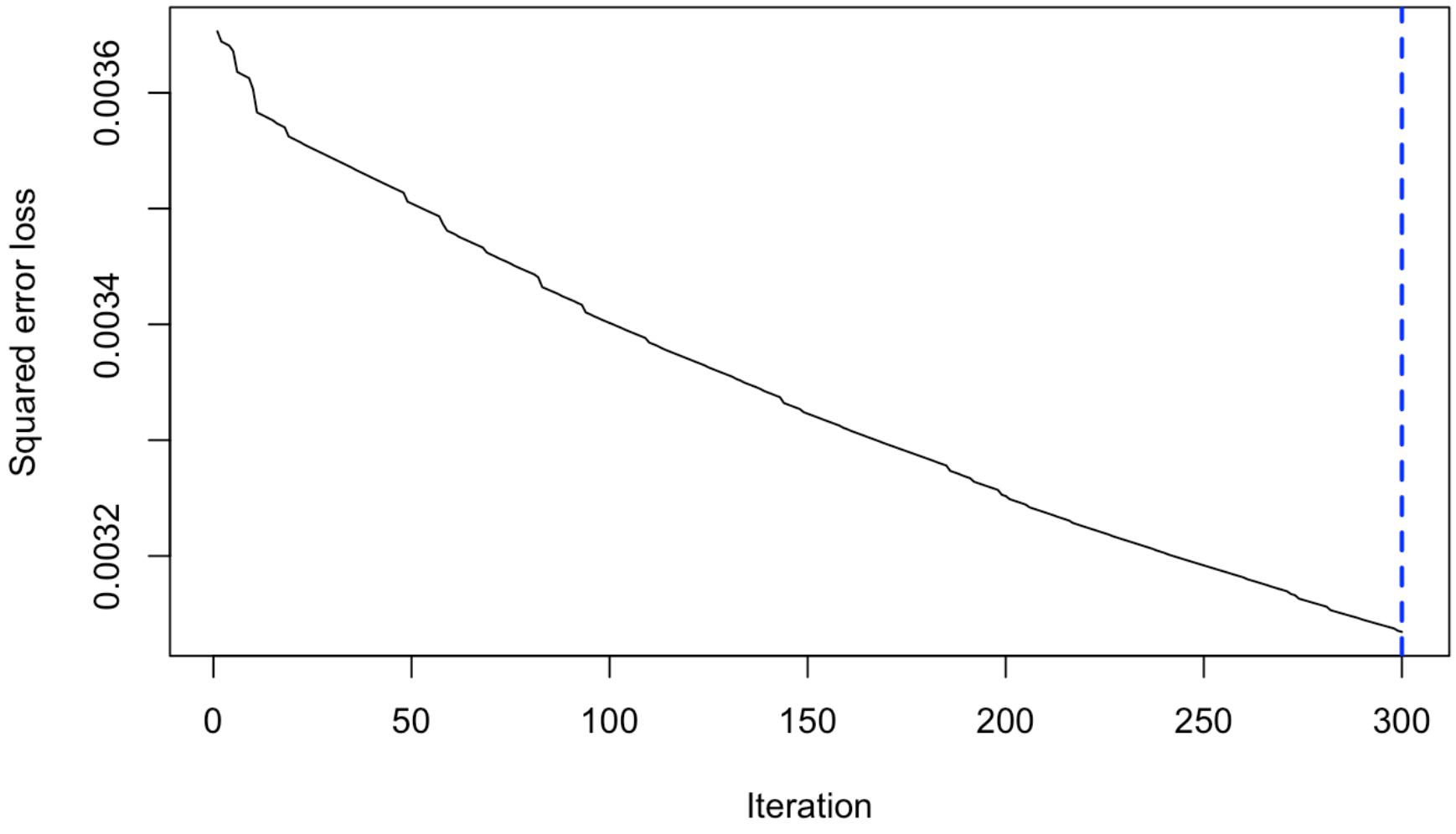

Squared error loss
0.0037
0.0035
0.0033
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

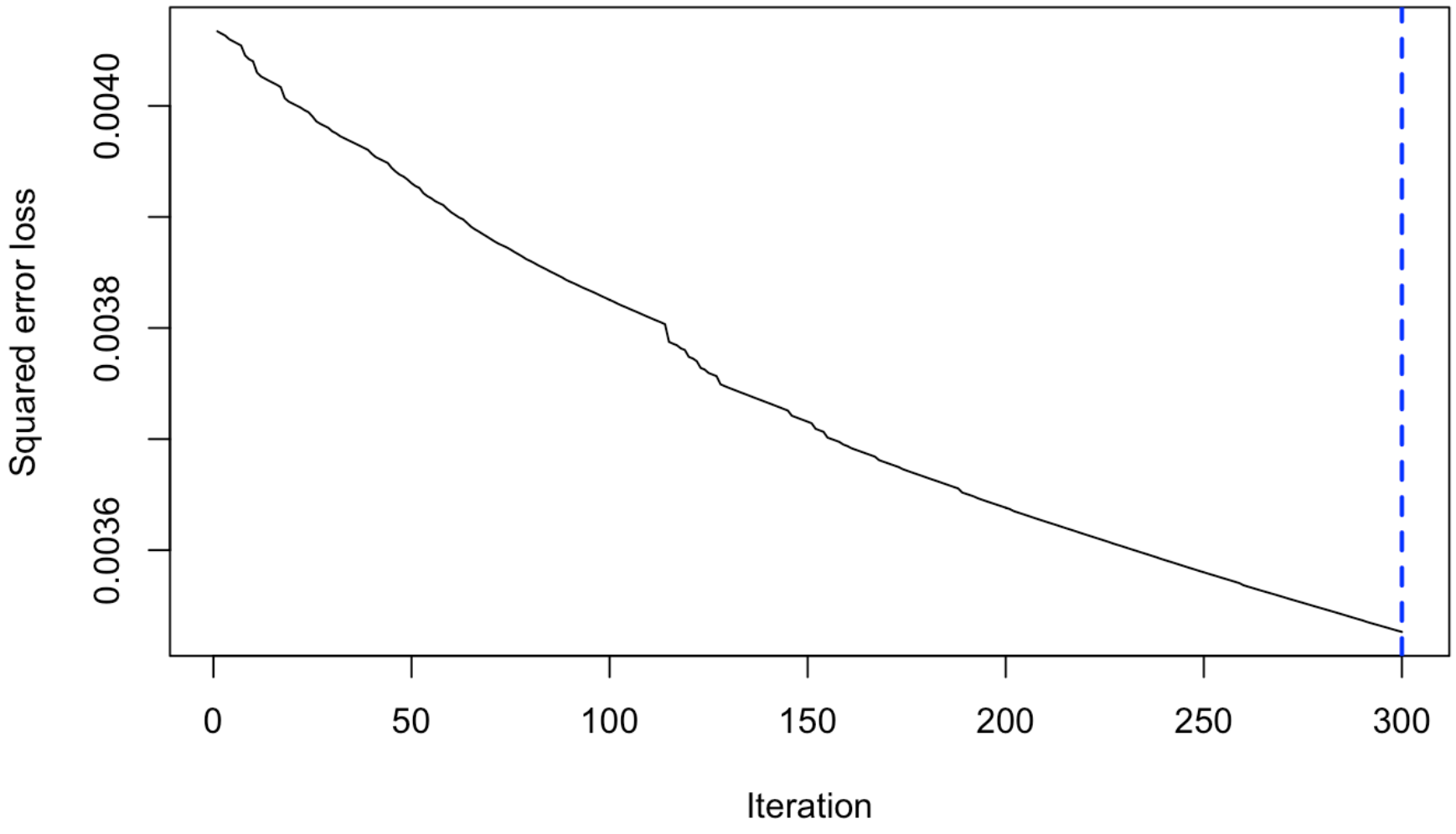
Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

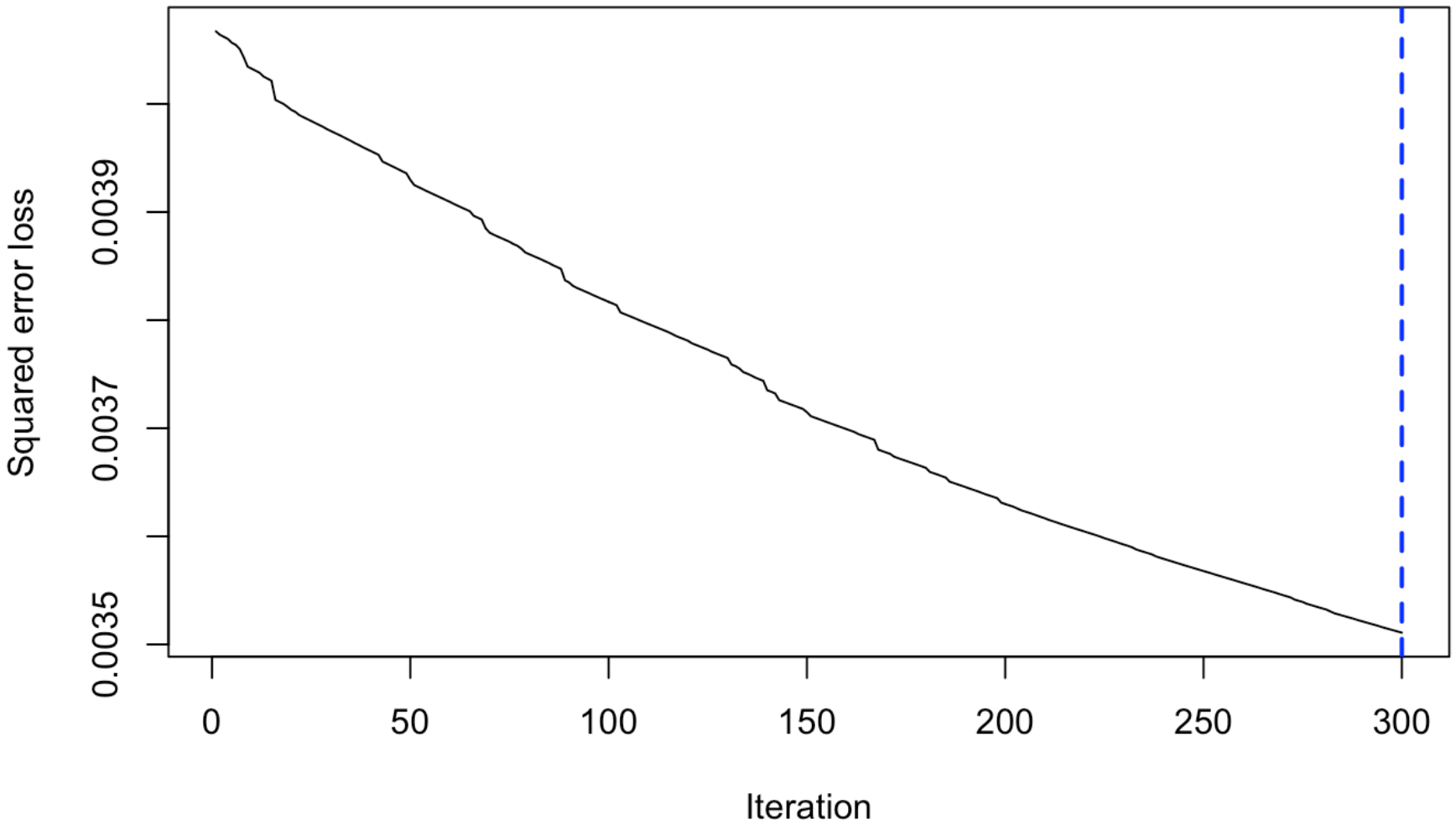

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

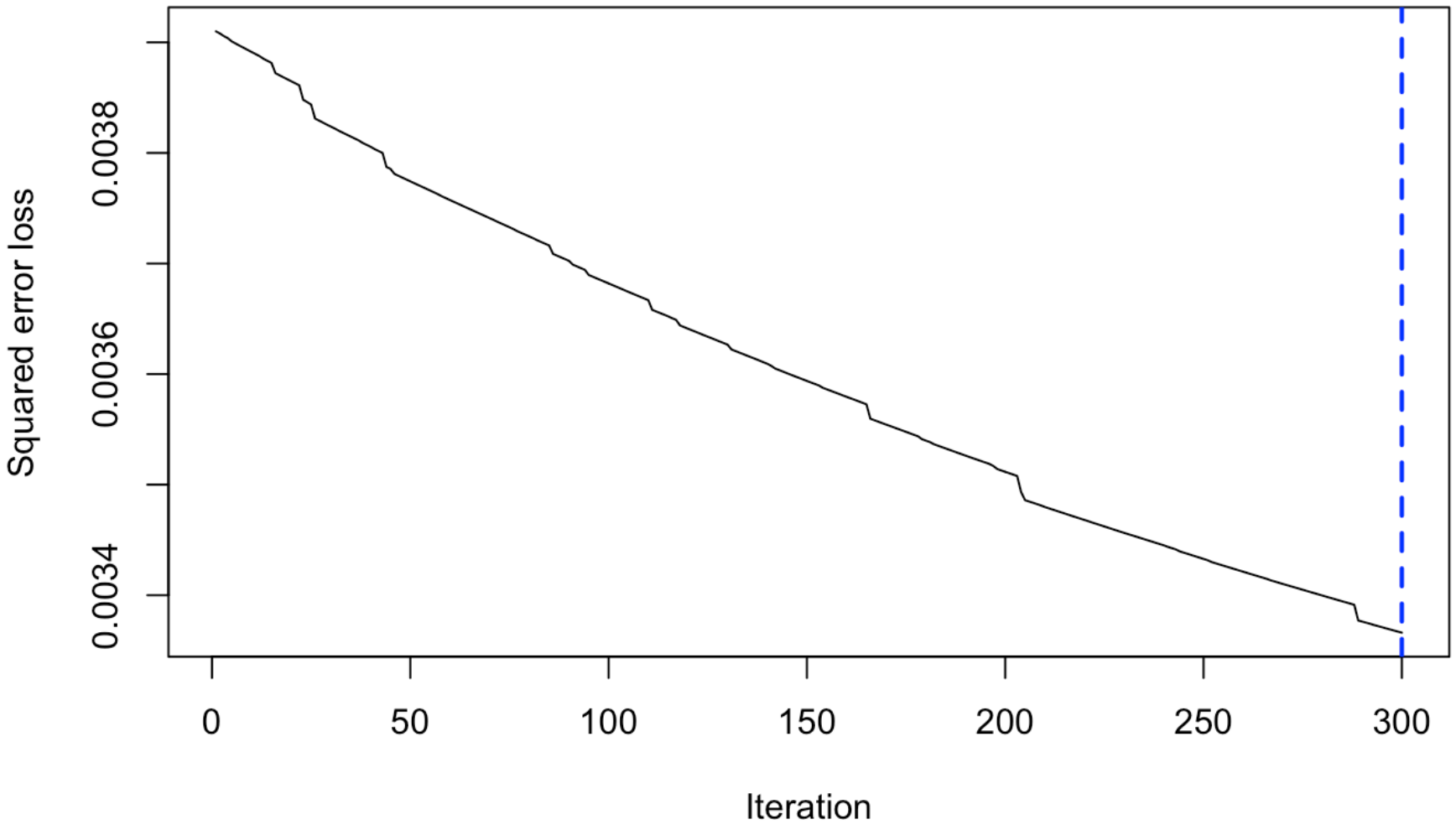
Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration

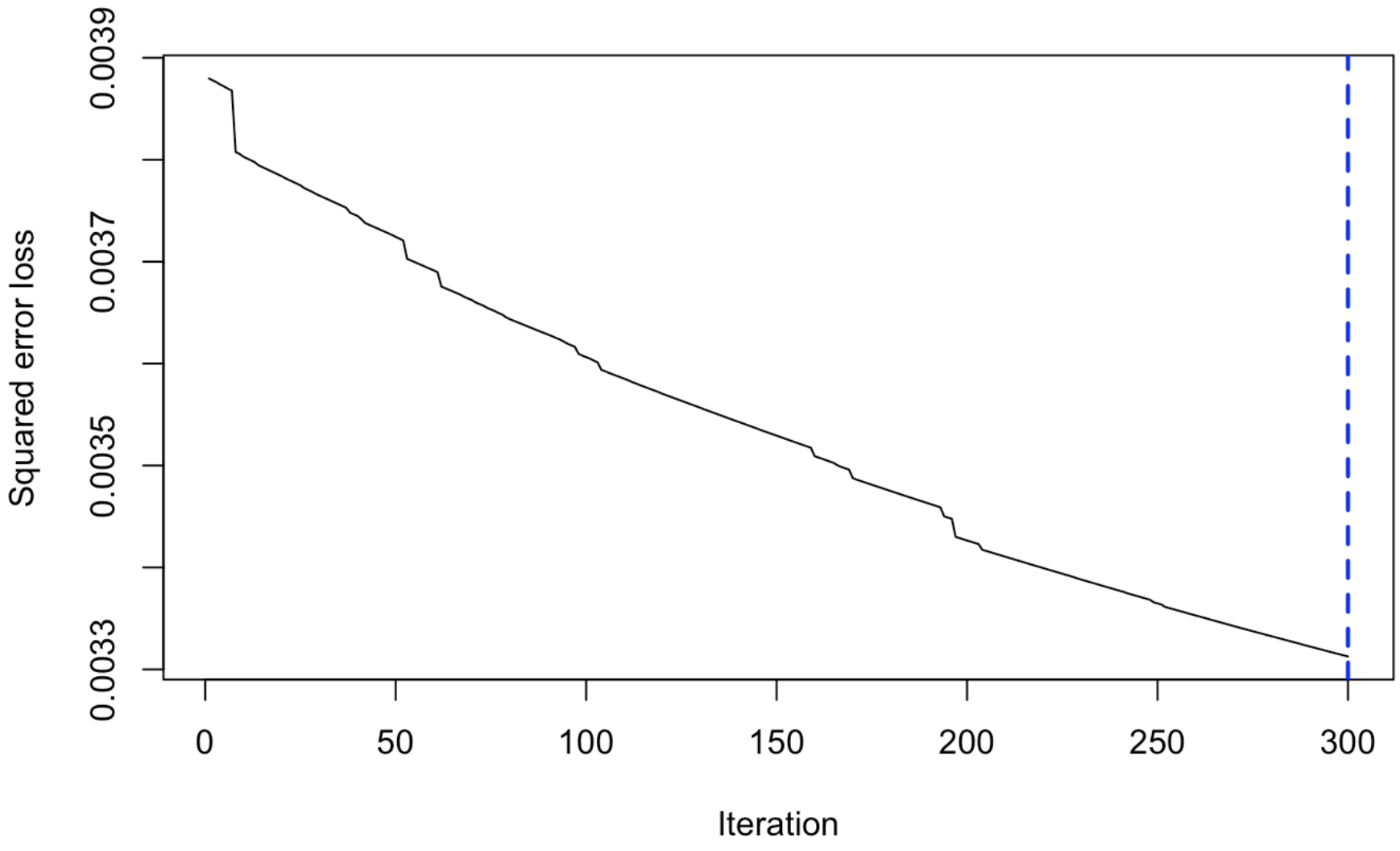
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

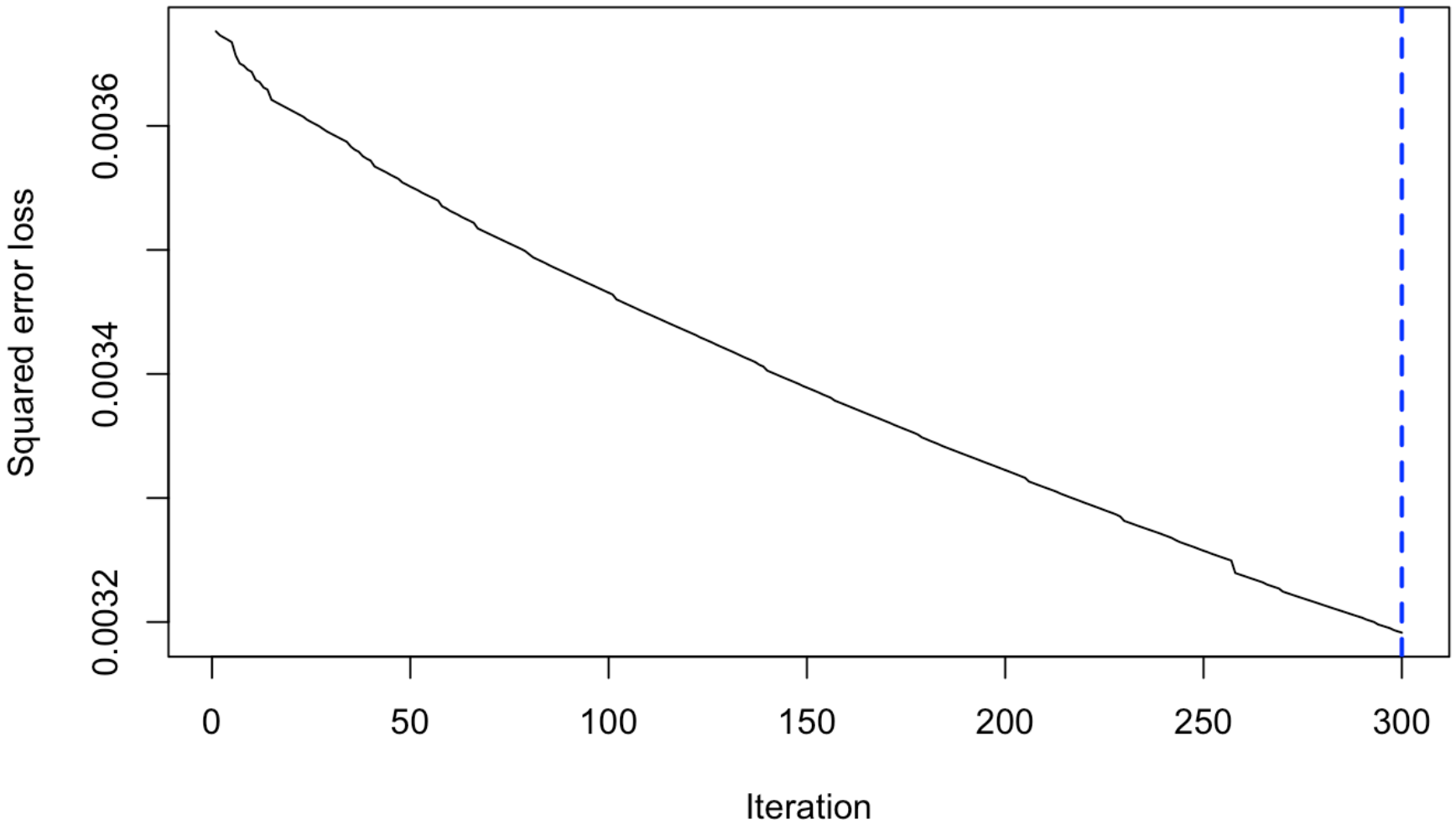
Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

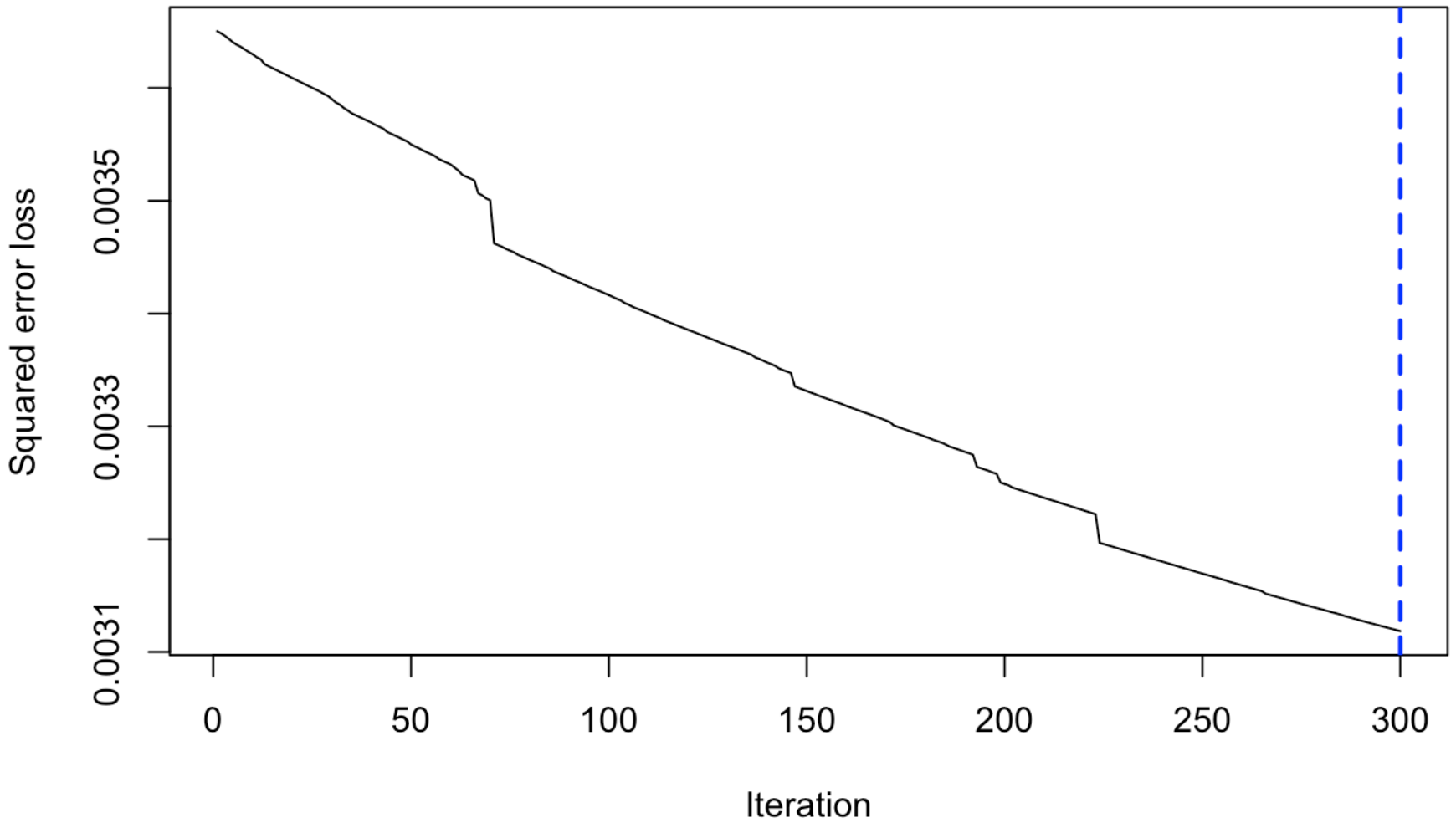
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration

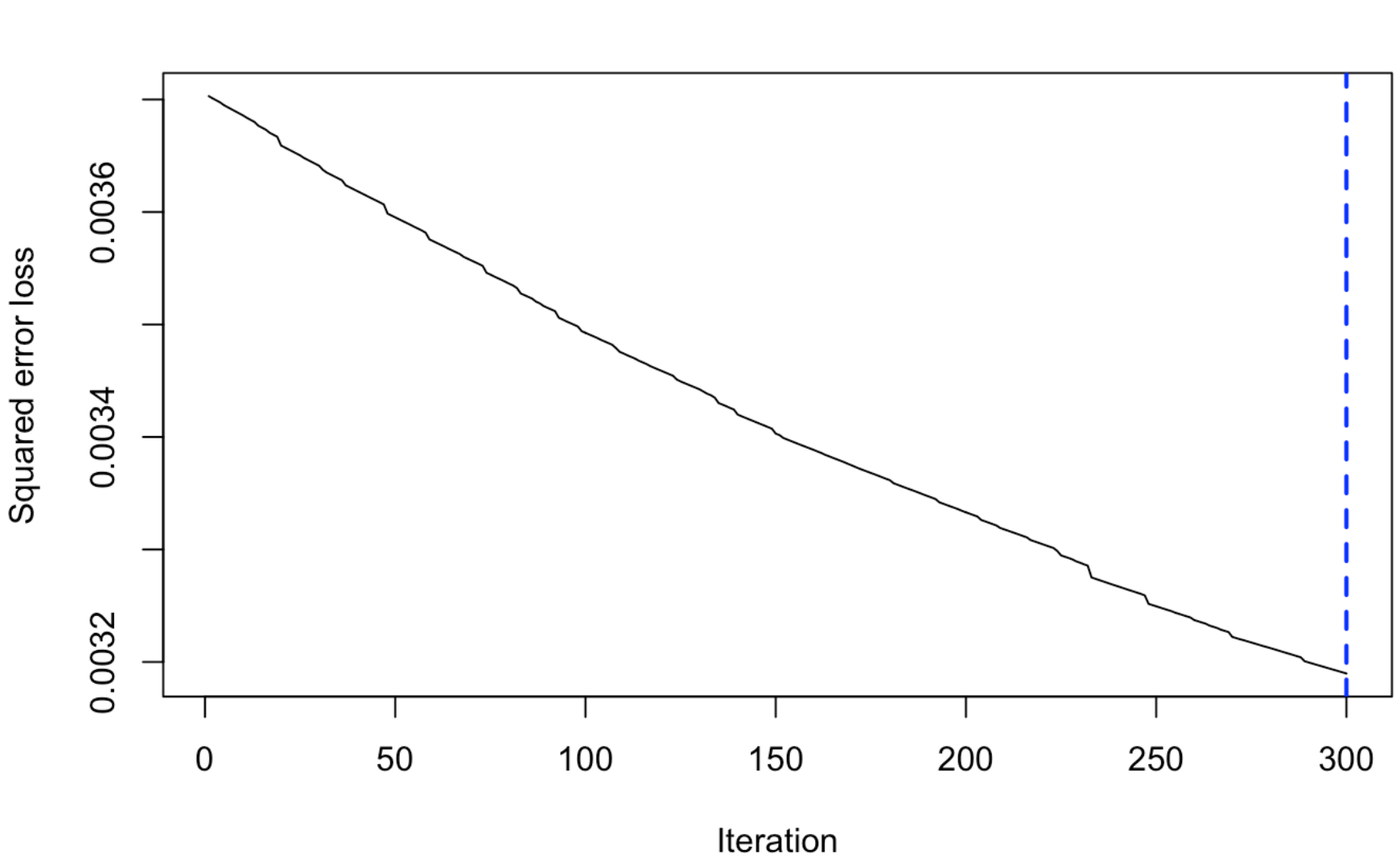

Squared error loss
0.0039
0.0037
0.0035
0.0033
0
50
100
150
200
250
300
Iteration

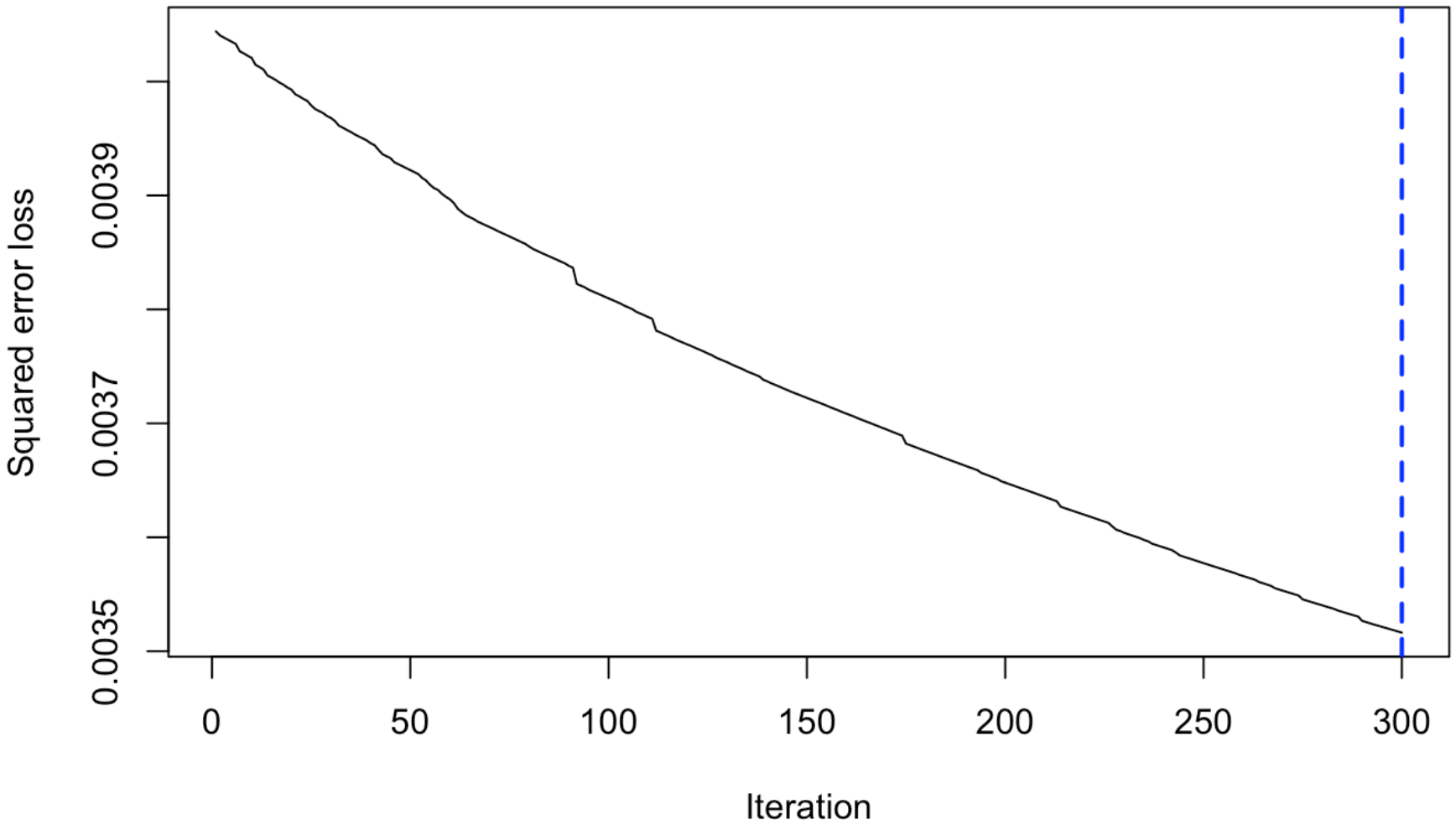
Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

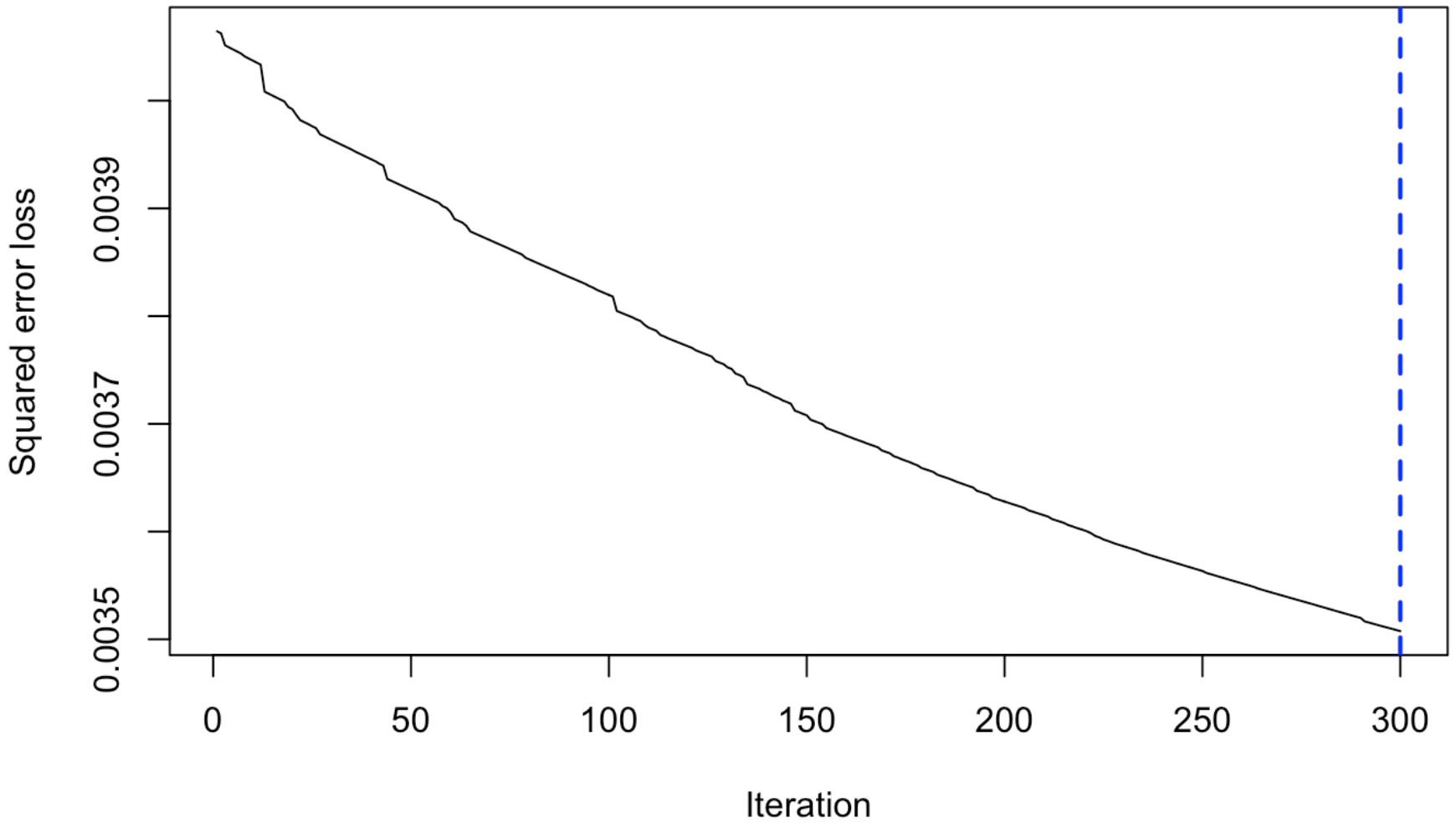
Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration

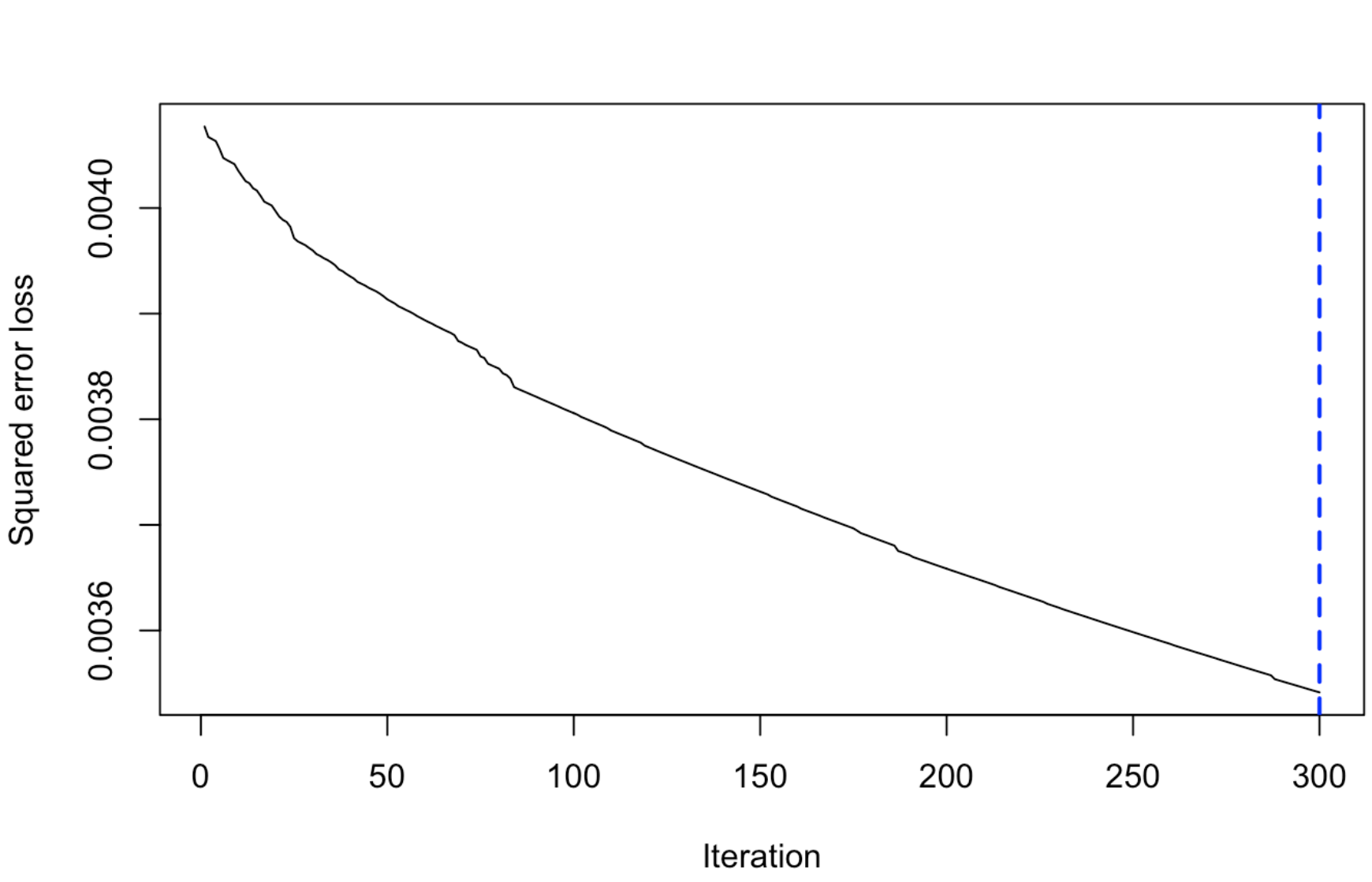

Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration

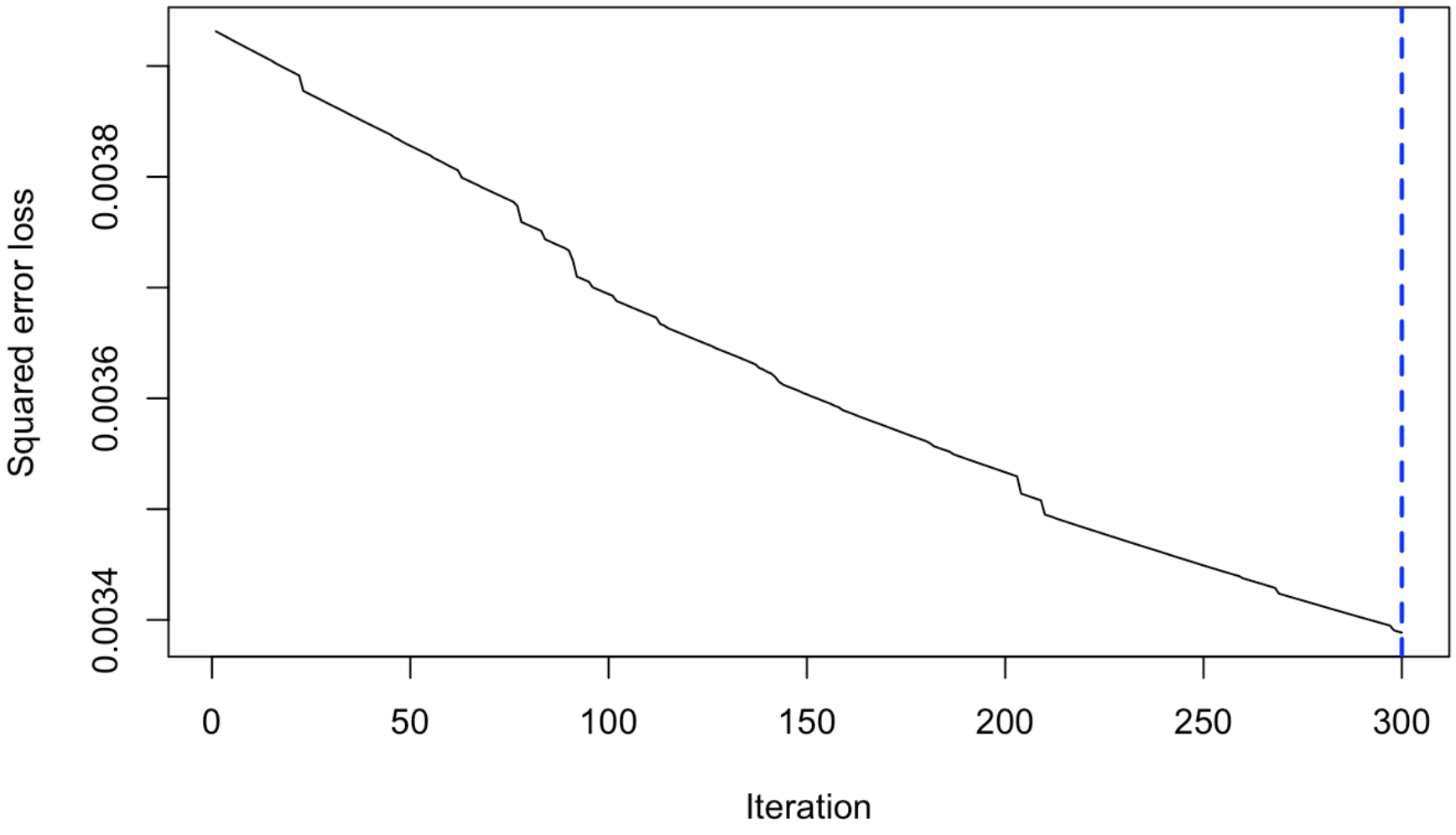

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

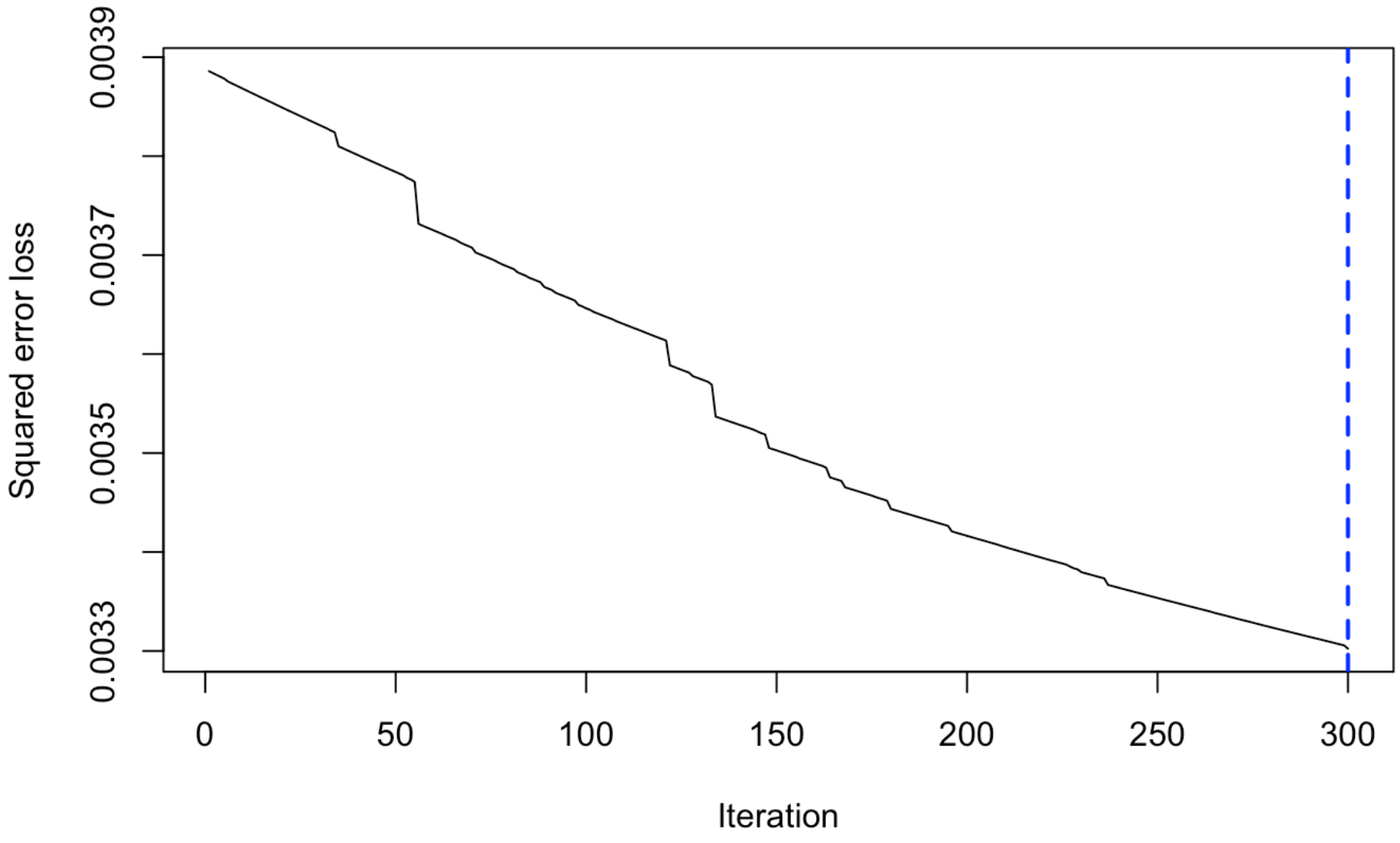
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

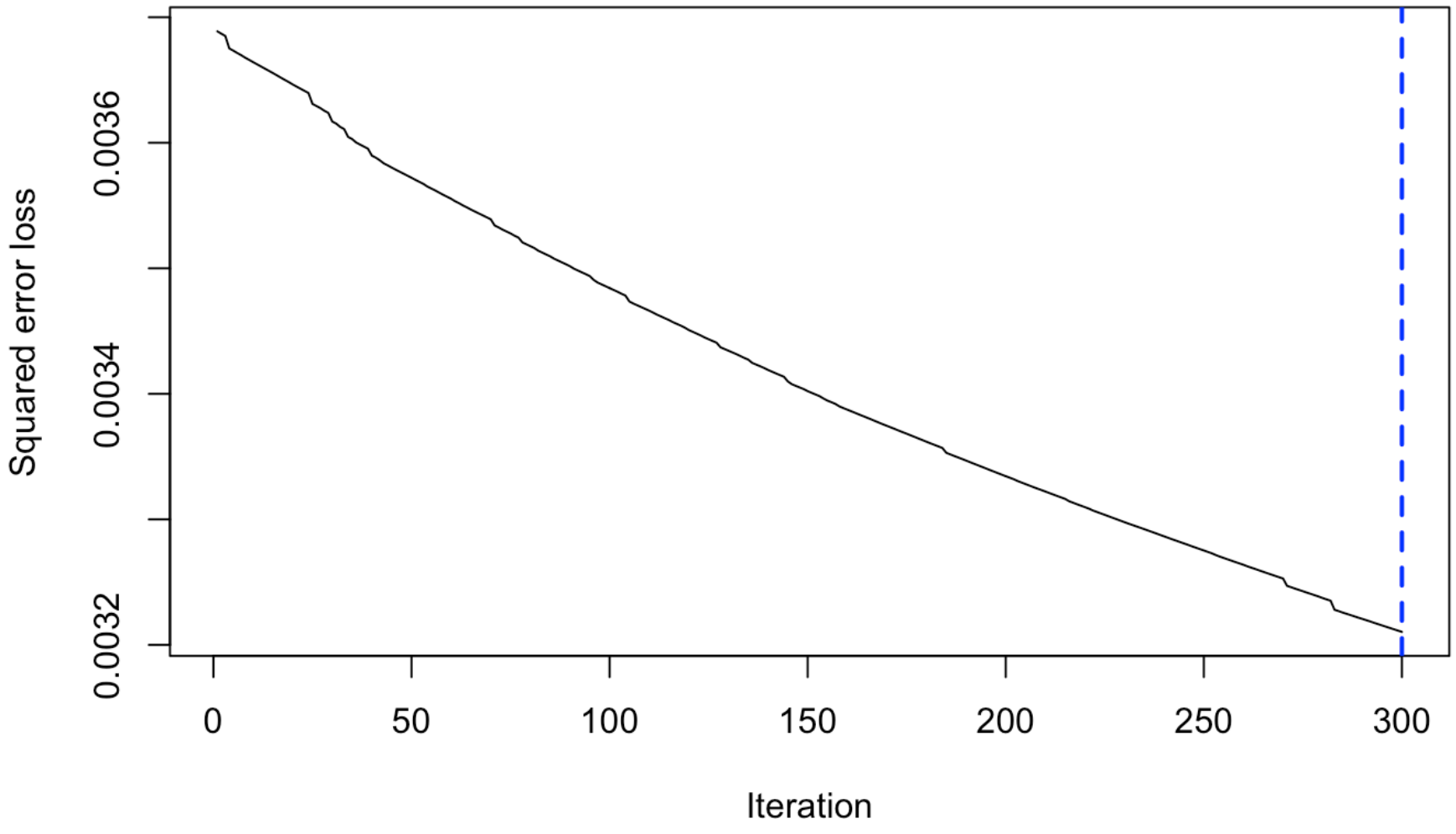

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration

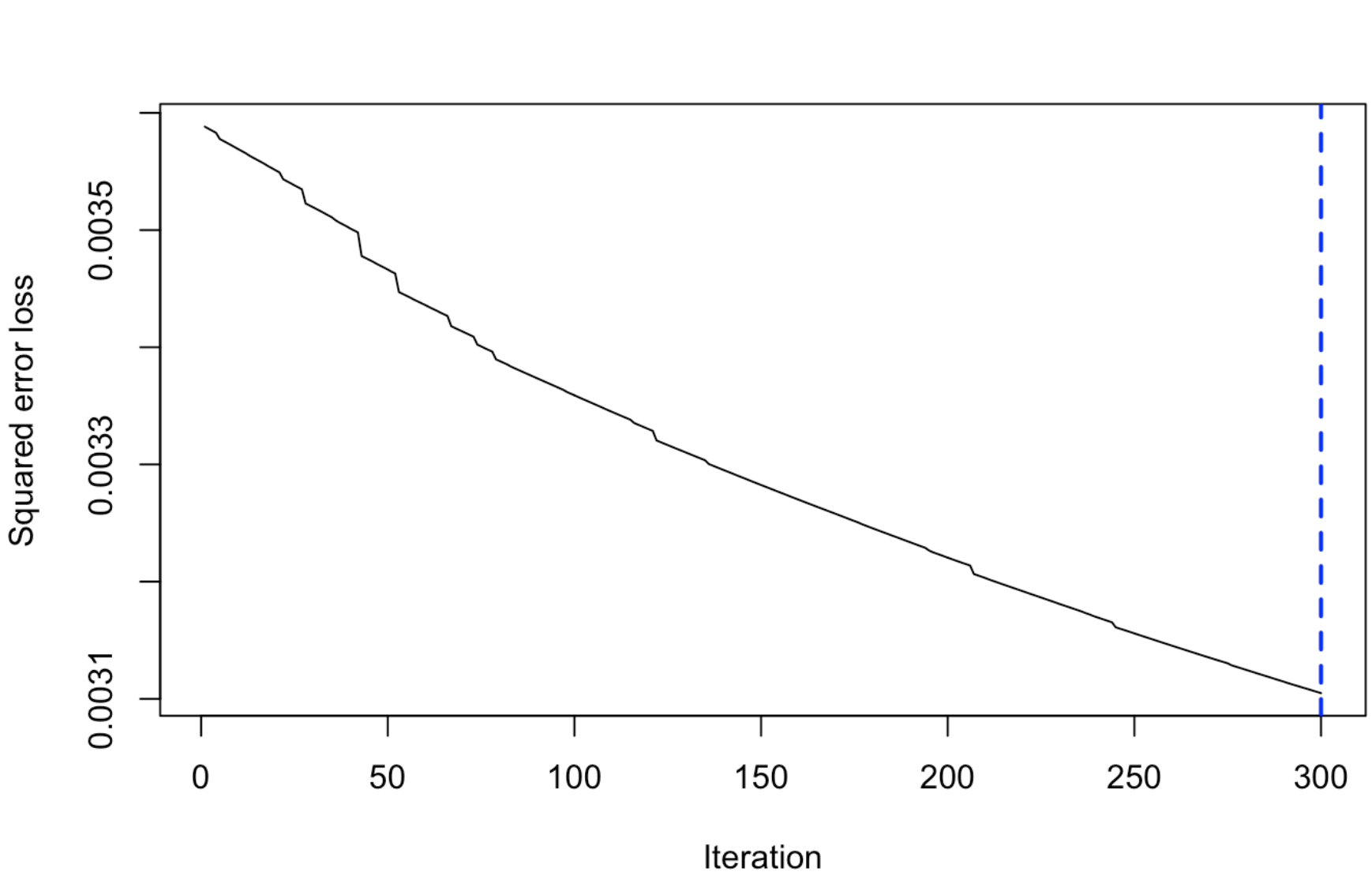

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

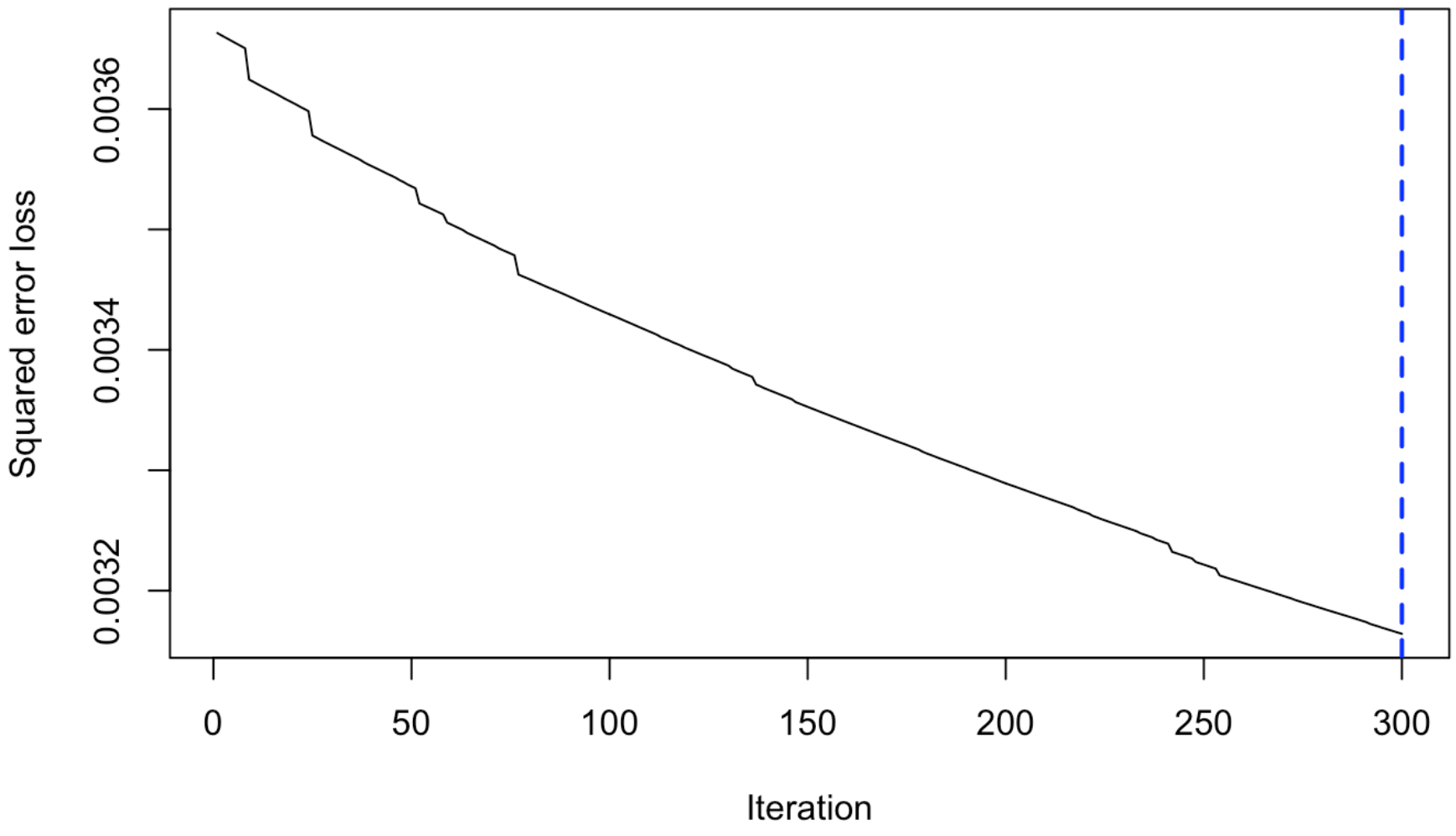
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0039
0.0037
0.0035
0.0033
0
50
100
150
200
250
300
Iteration

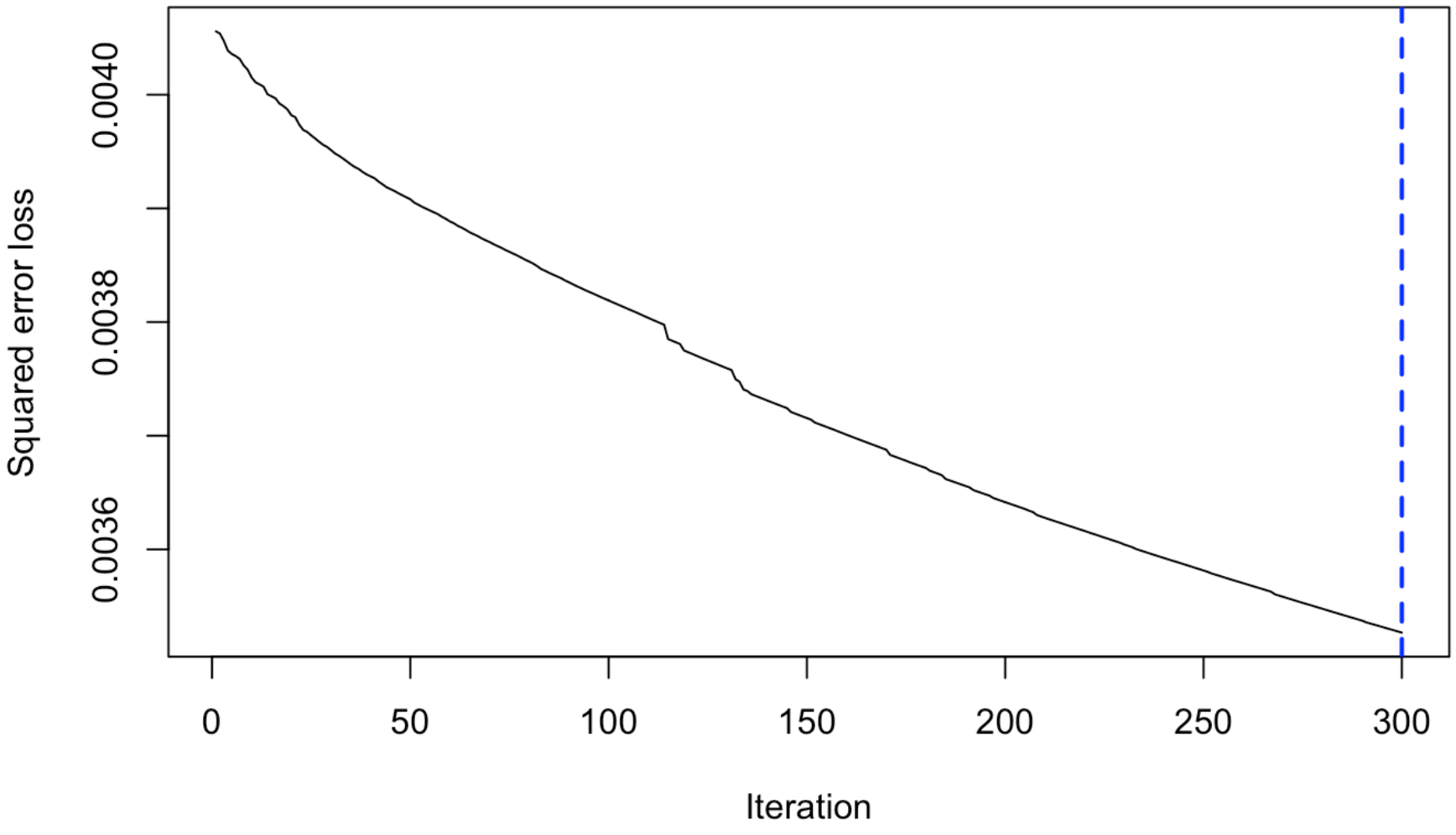
Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

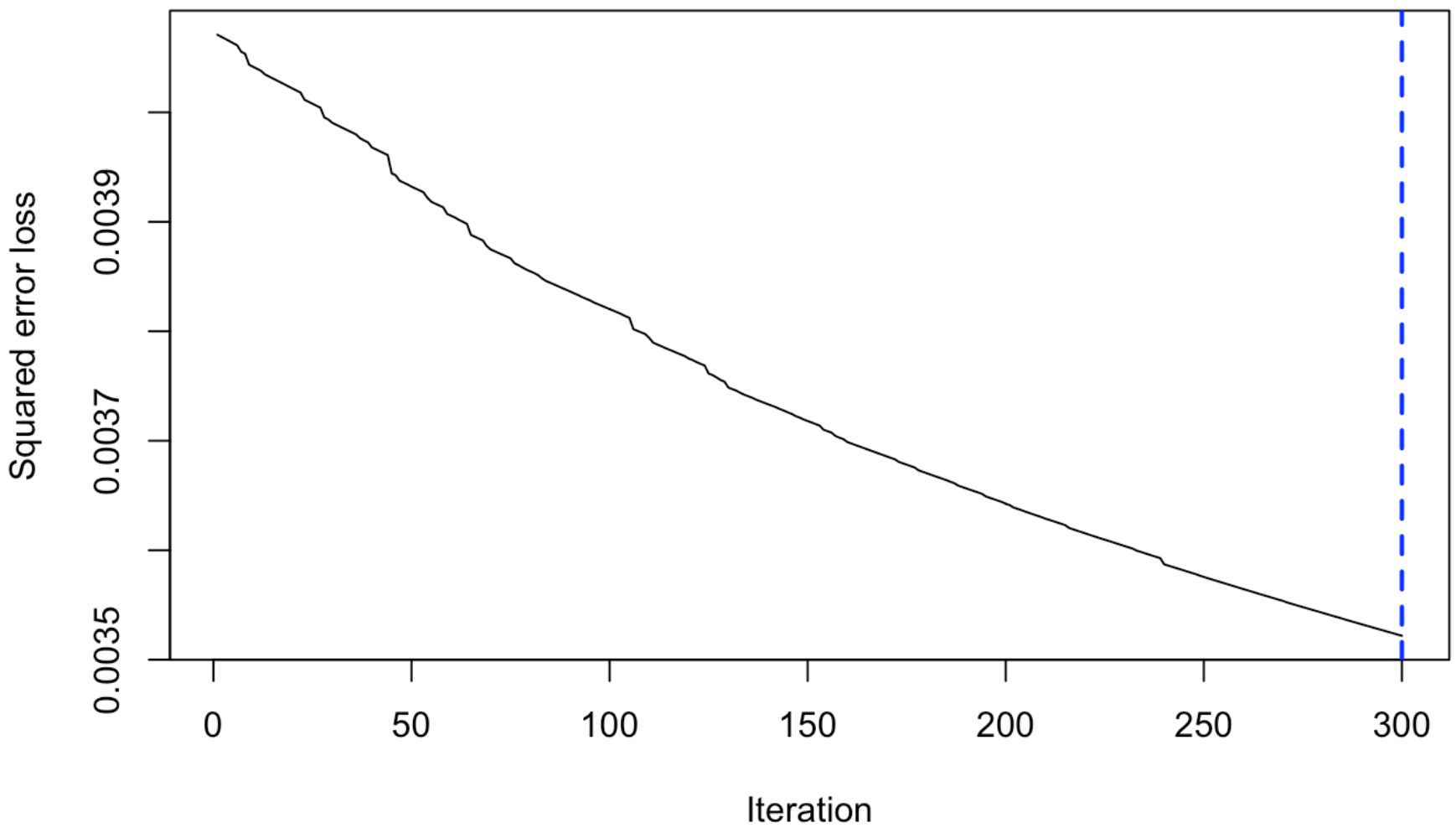

Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

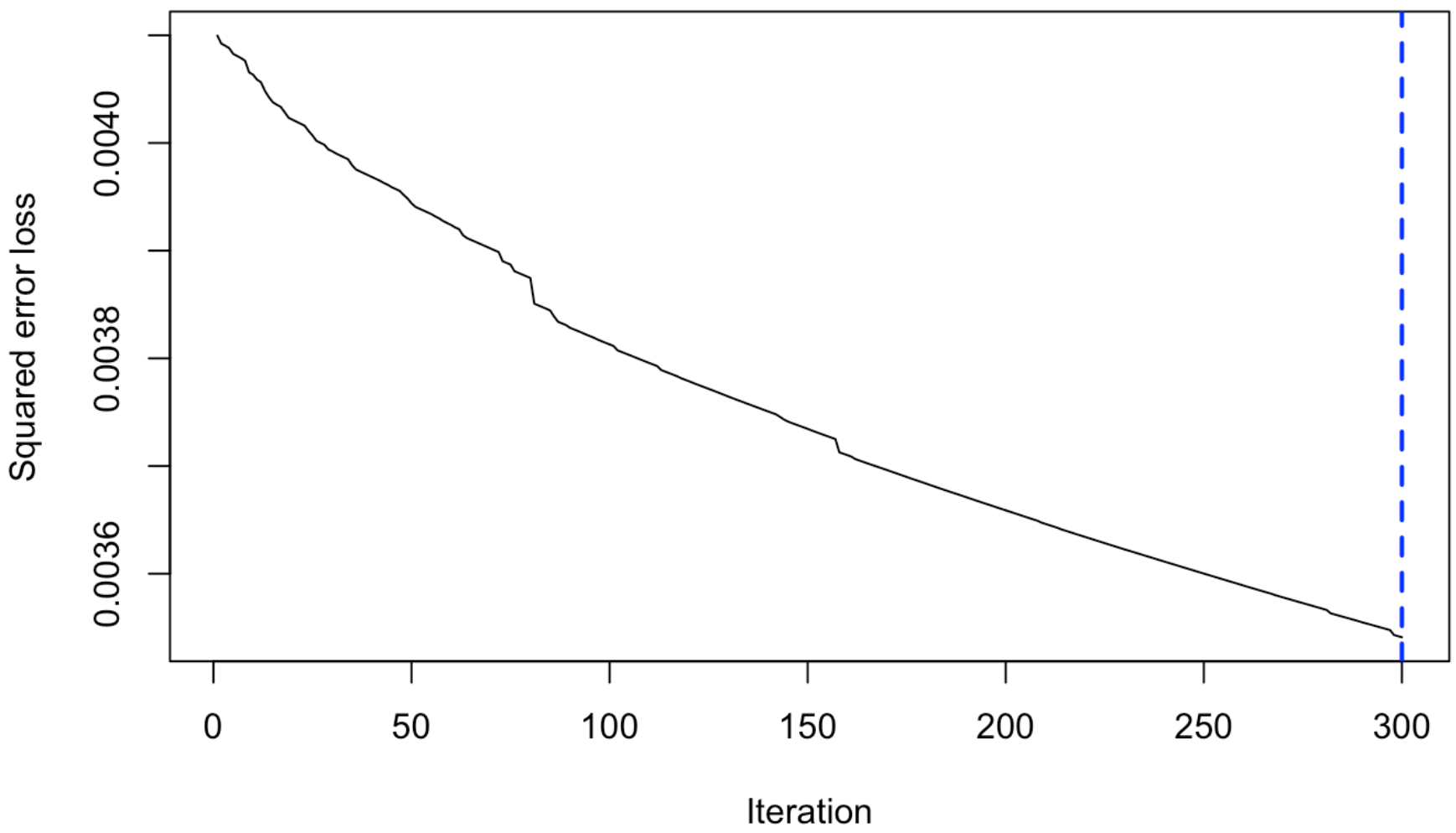
Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

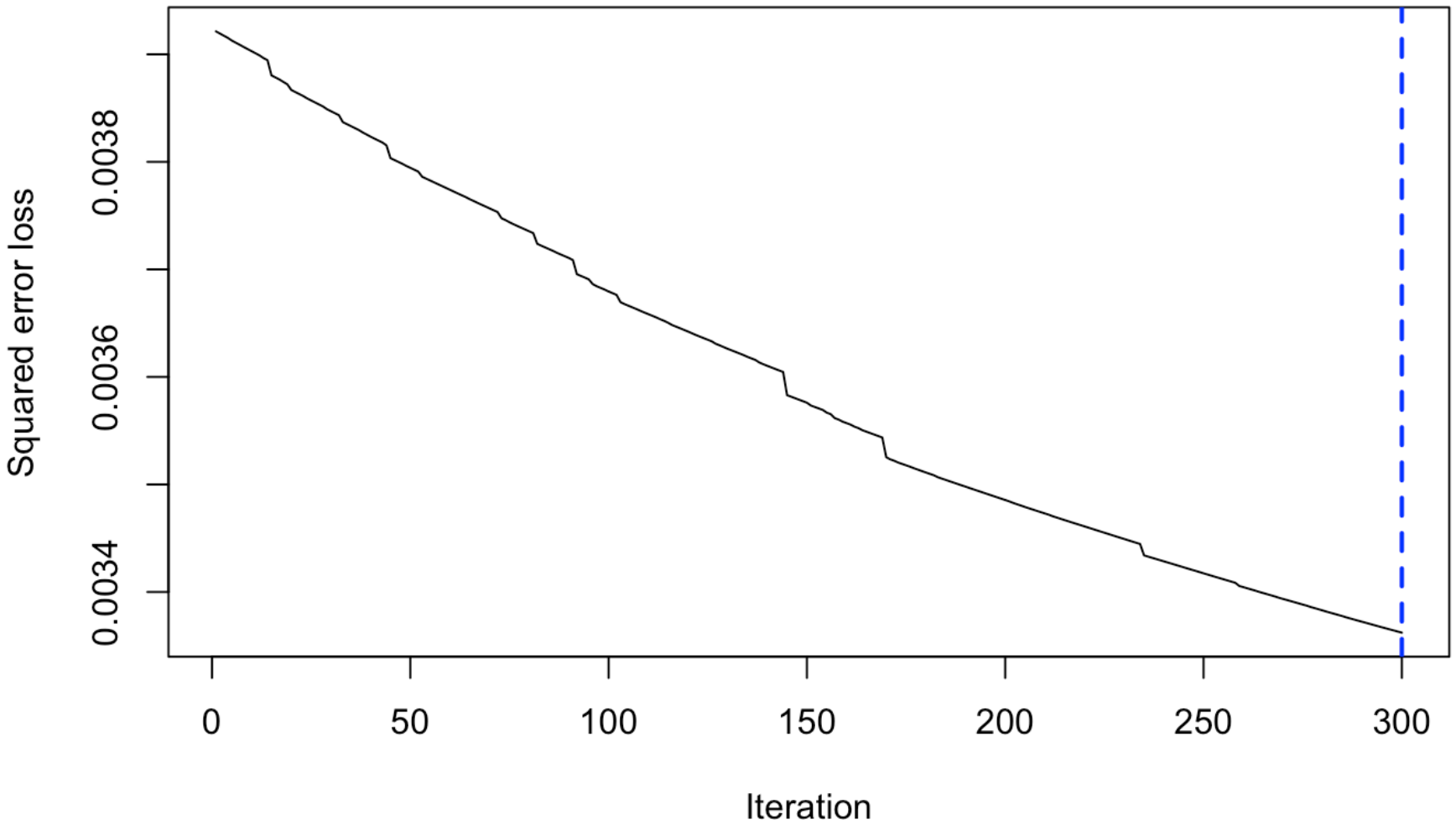

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

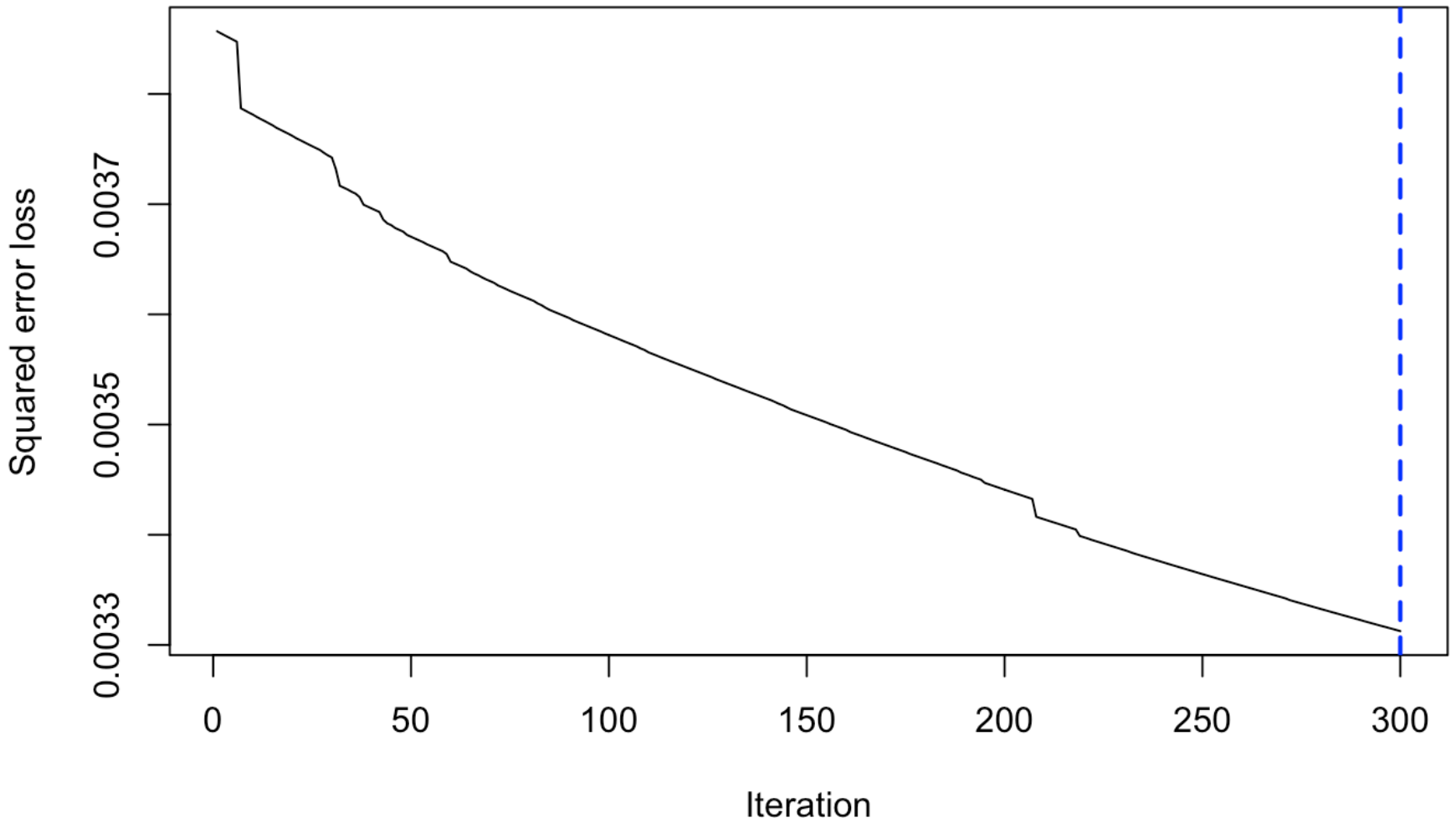

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

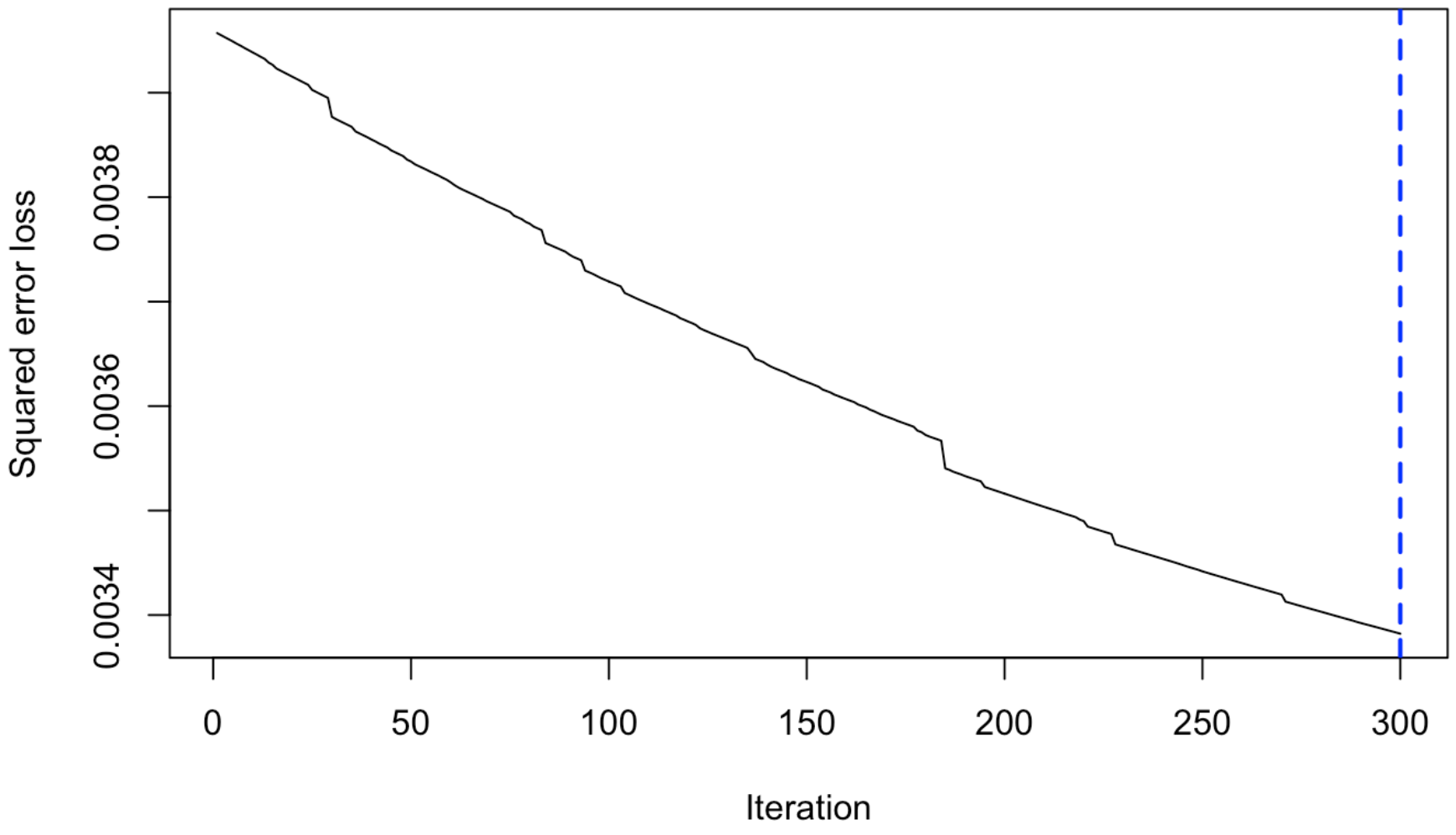
```
k=  5
```

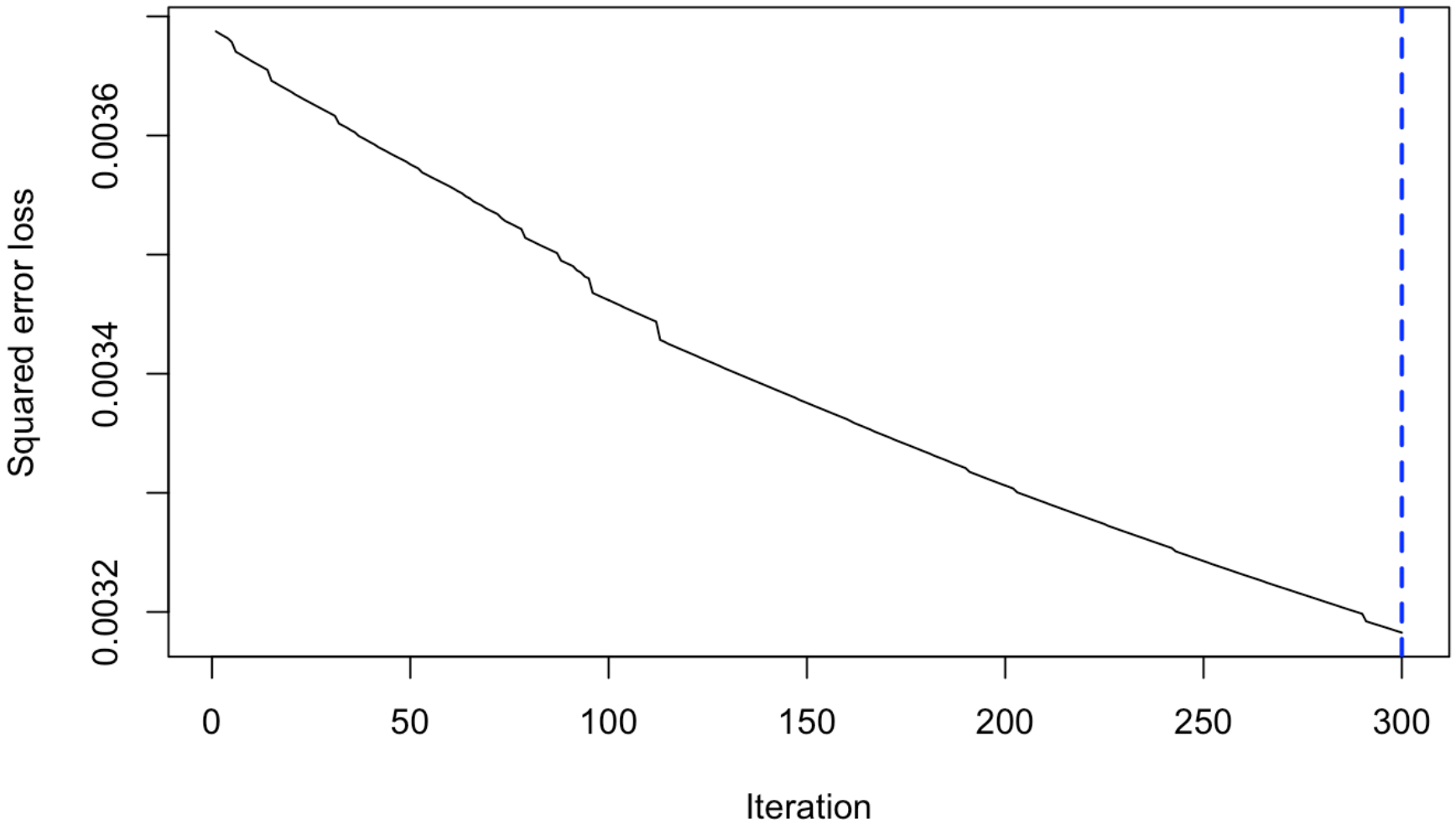

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

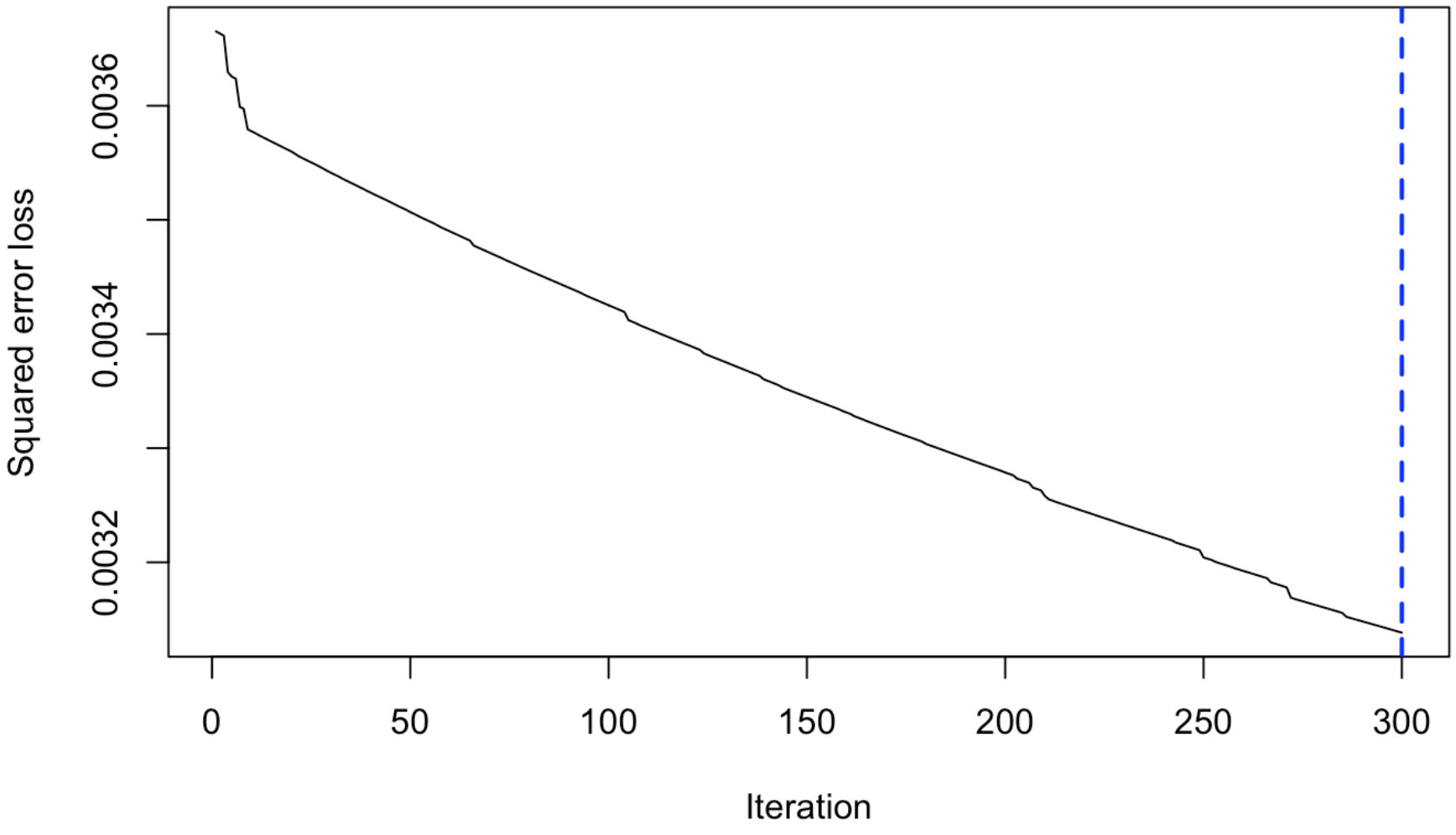
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration

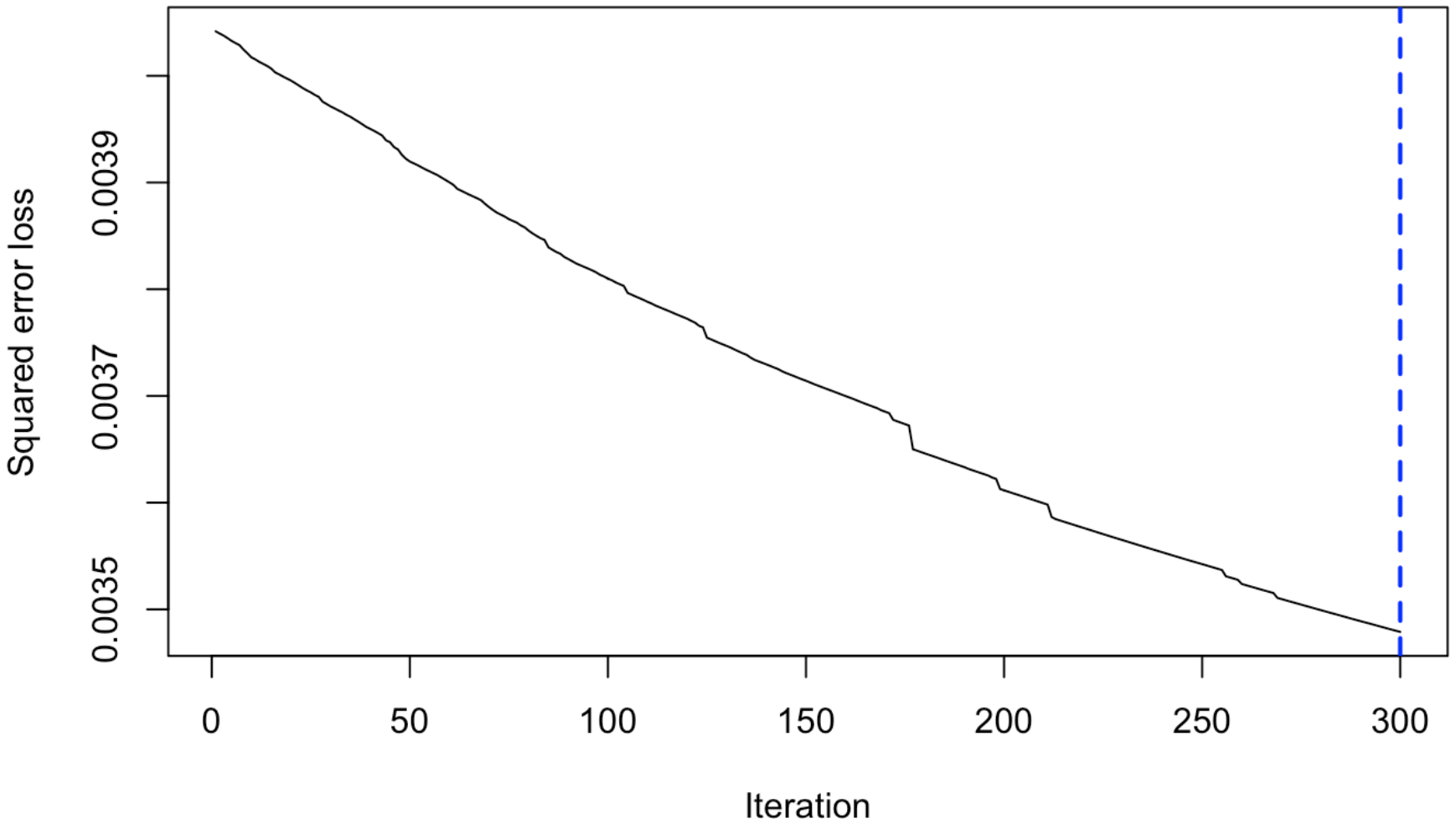

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

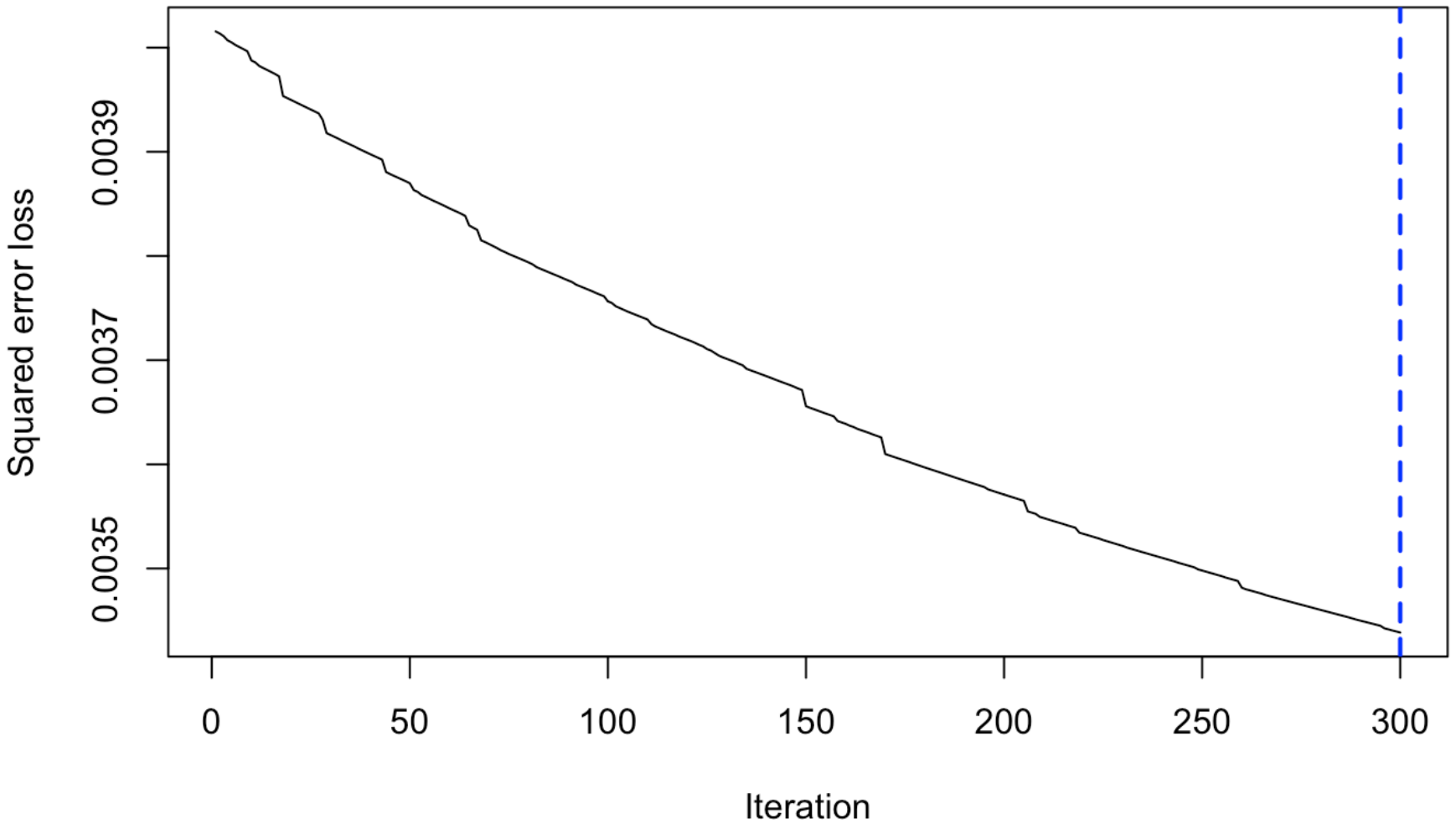

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

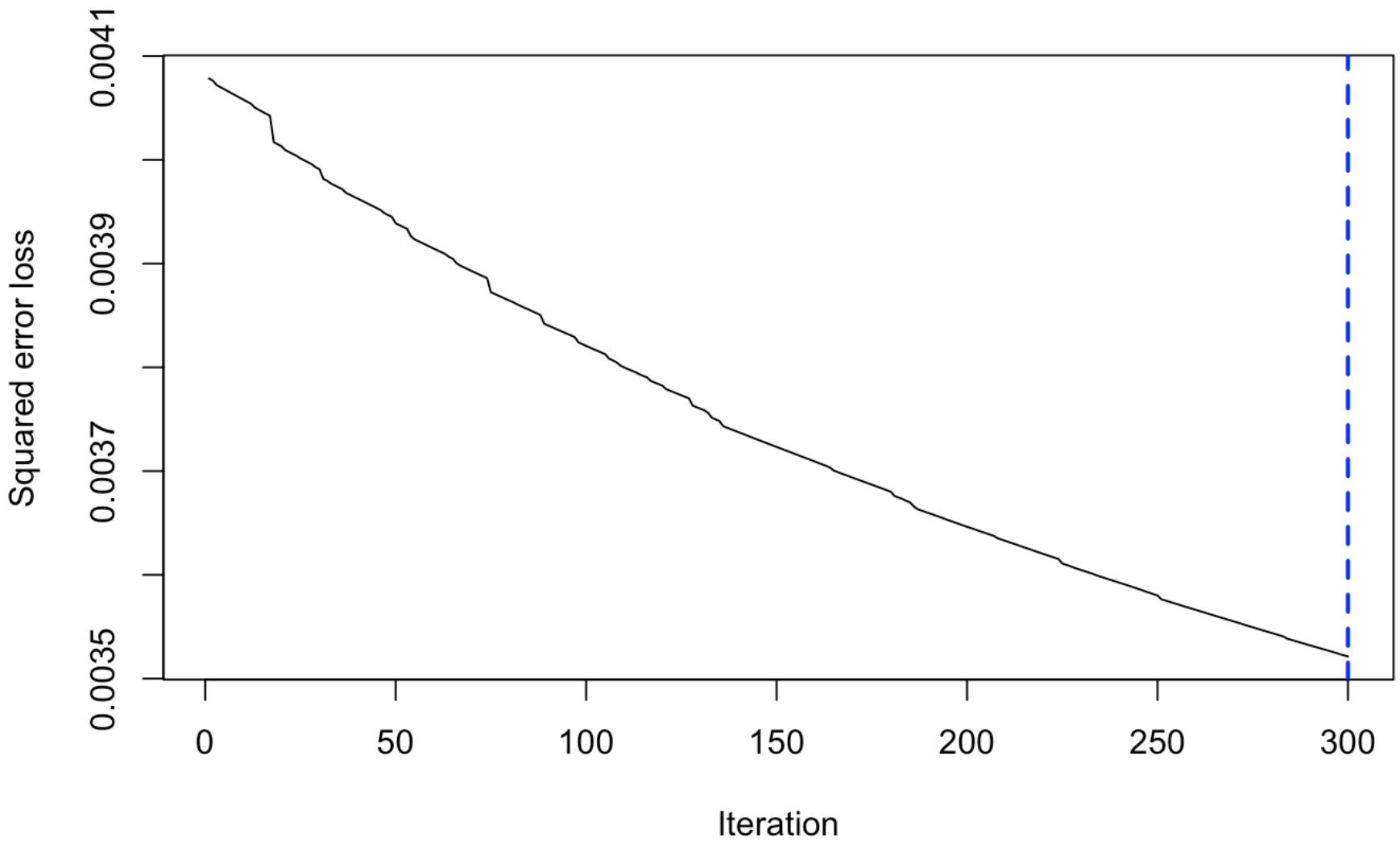
Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

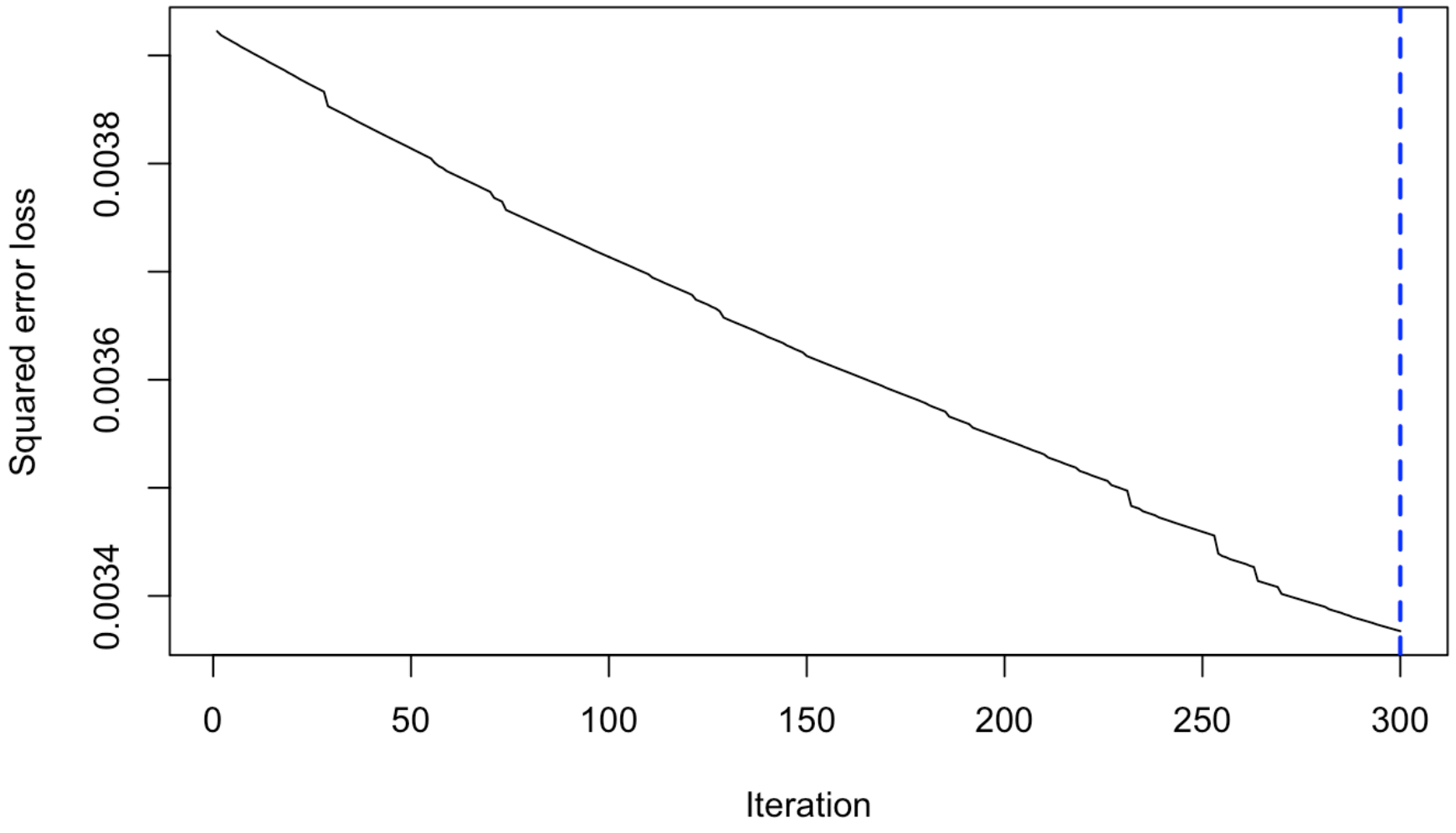
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration

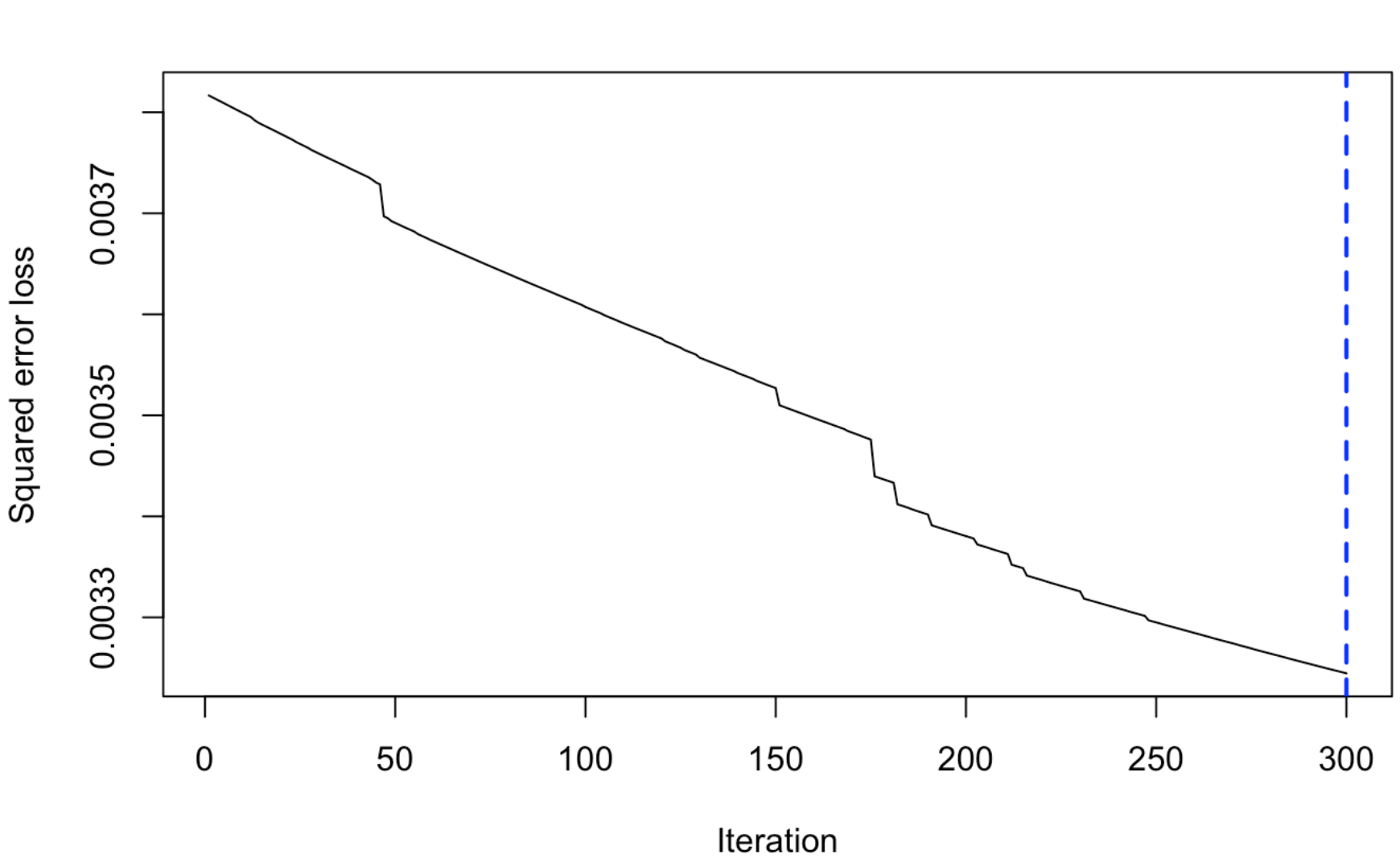

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

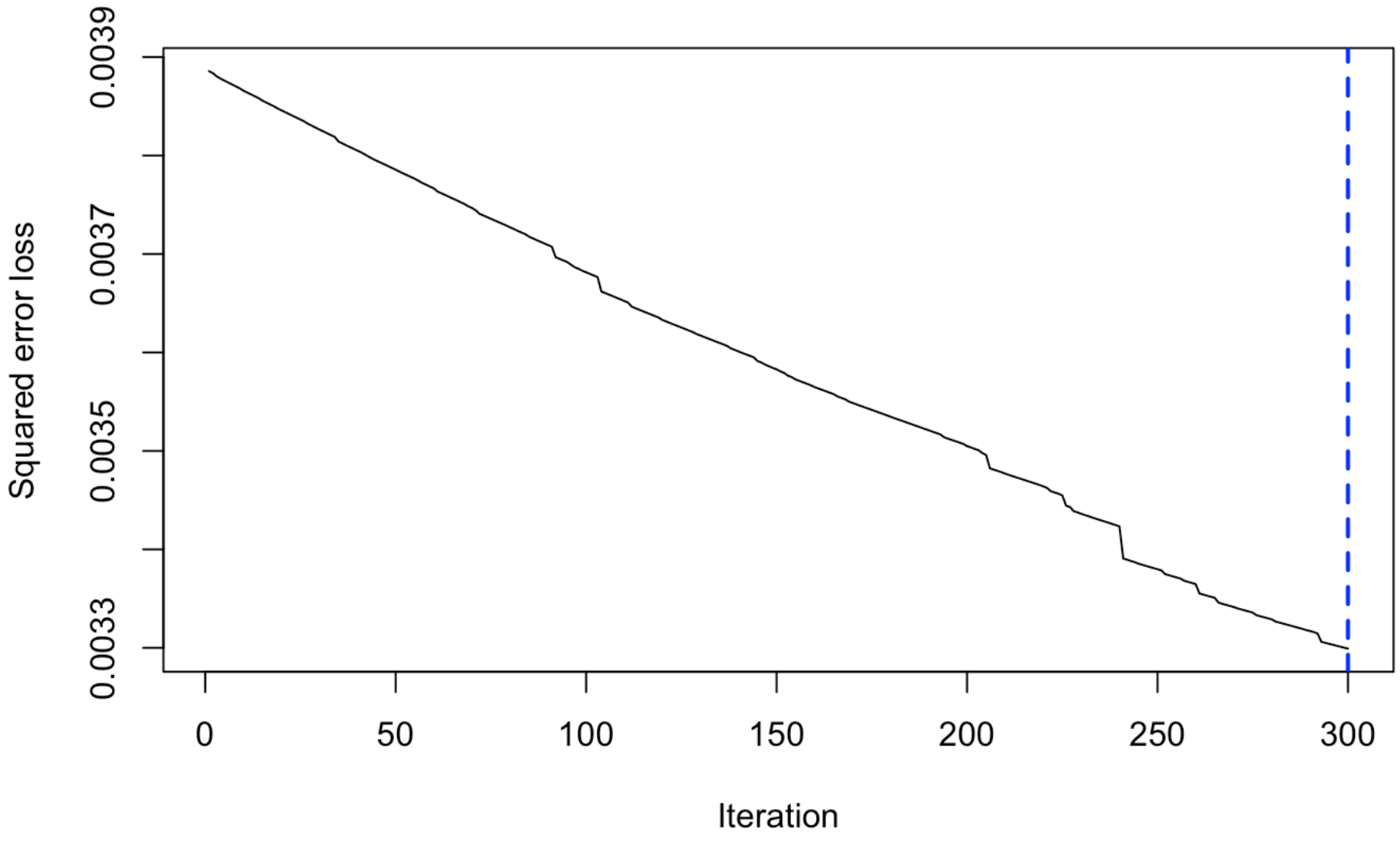

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0.0041
0
50
100
150
200
250
300
Iteration

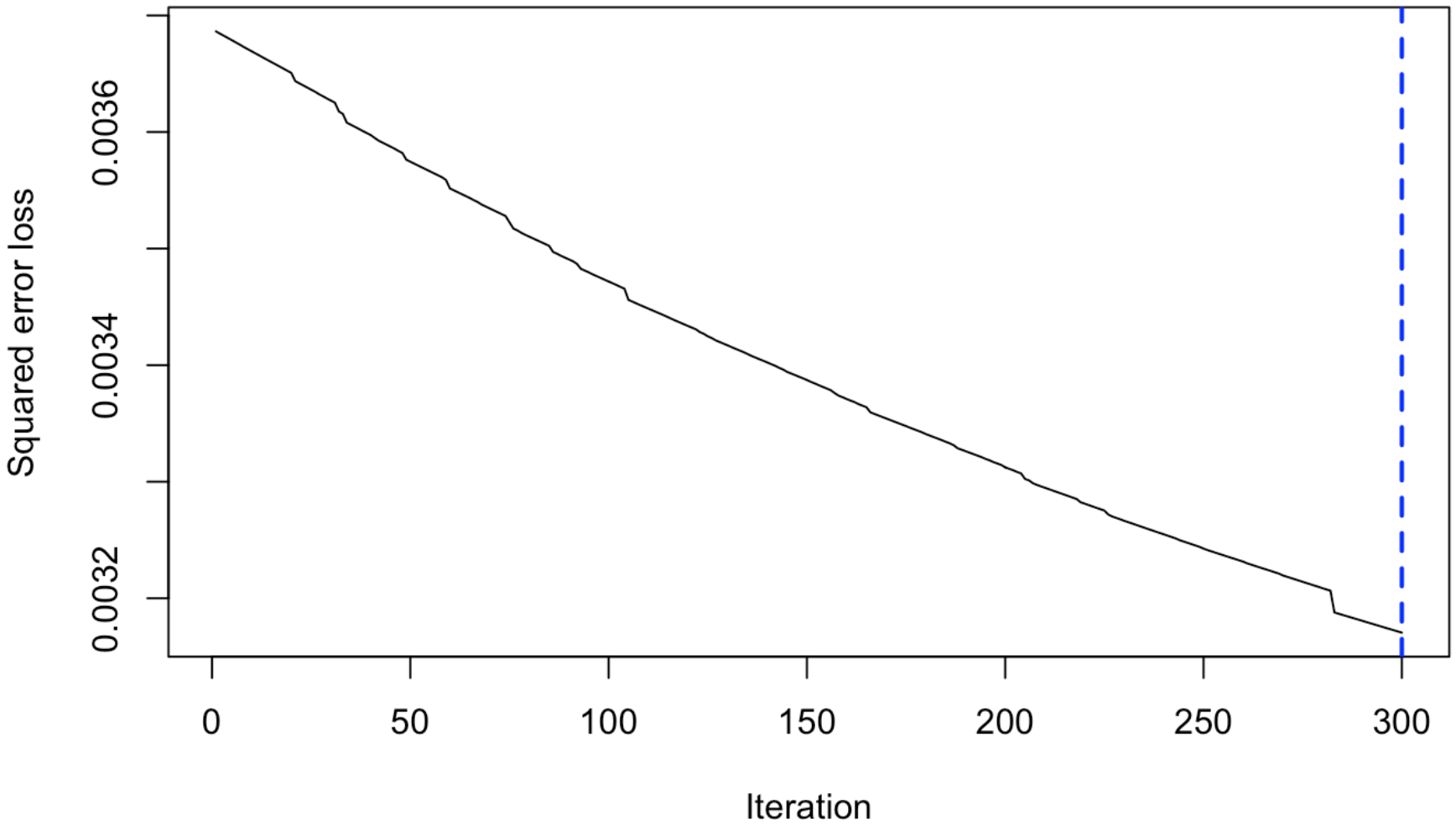
Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

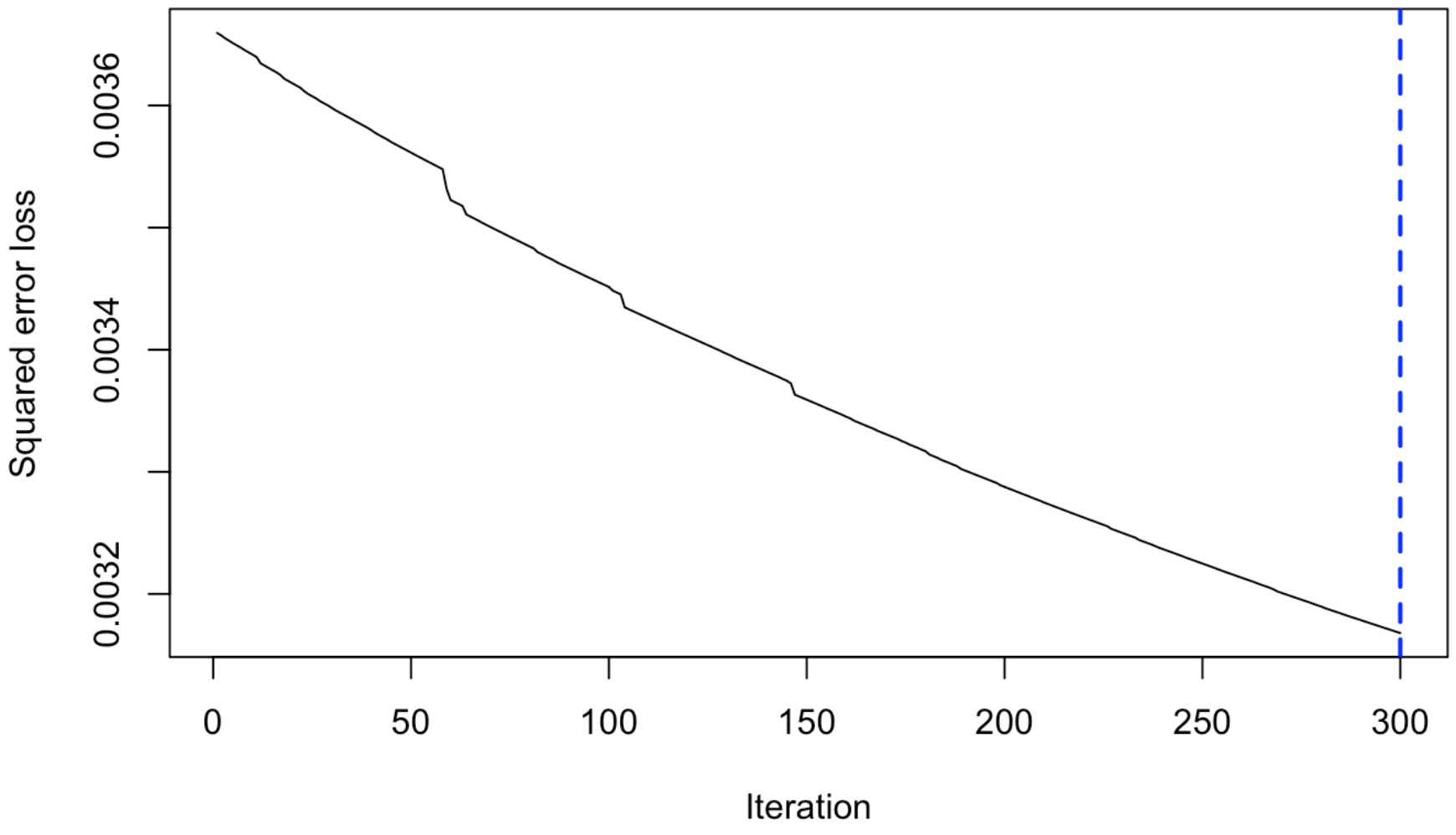

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0033
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

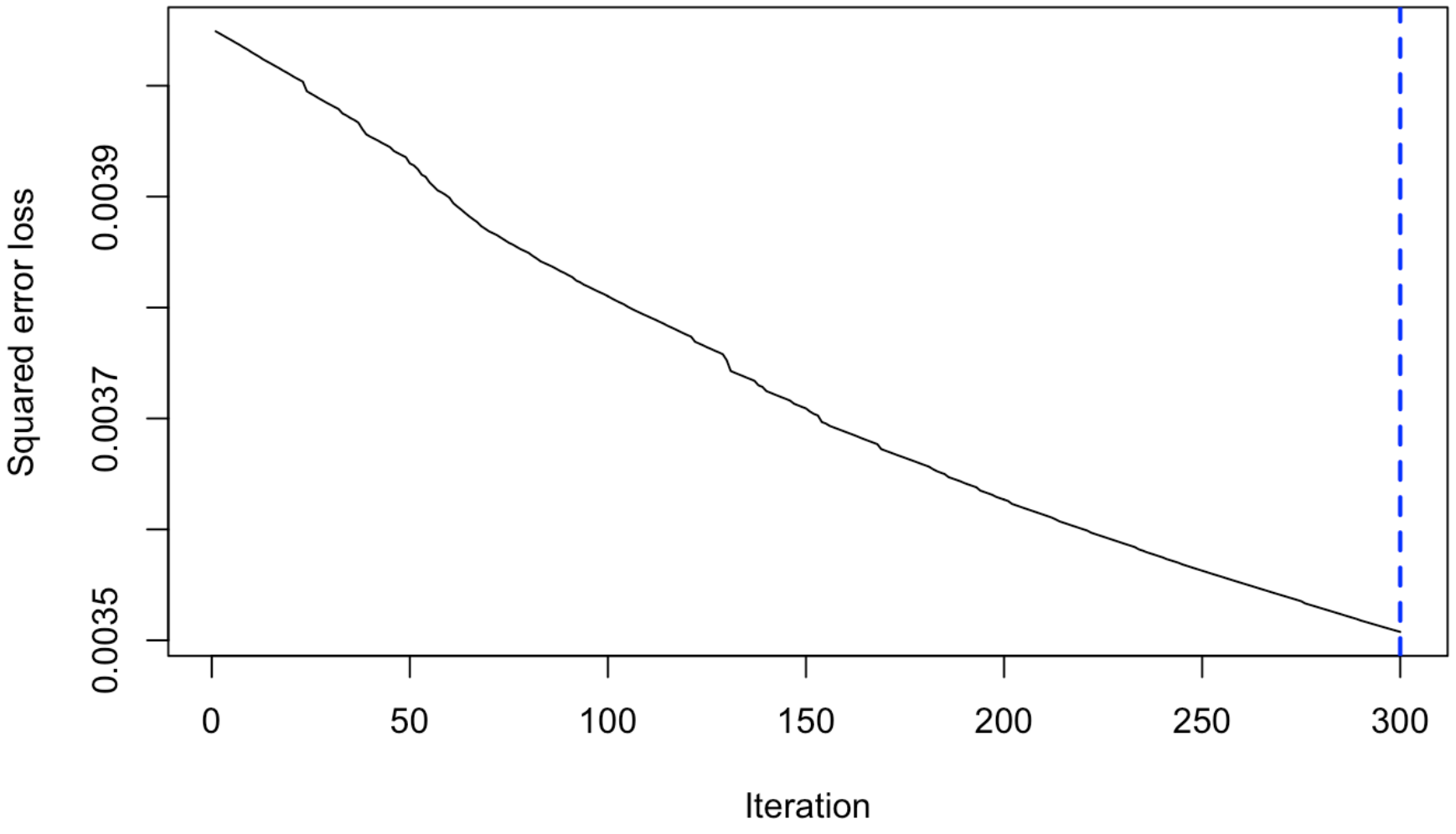
Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

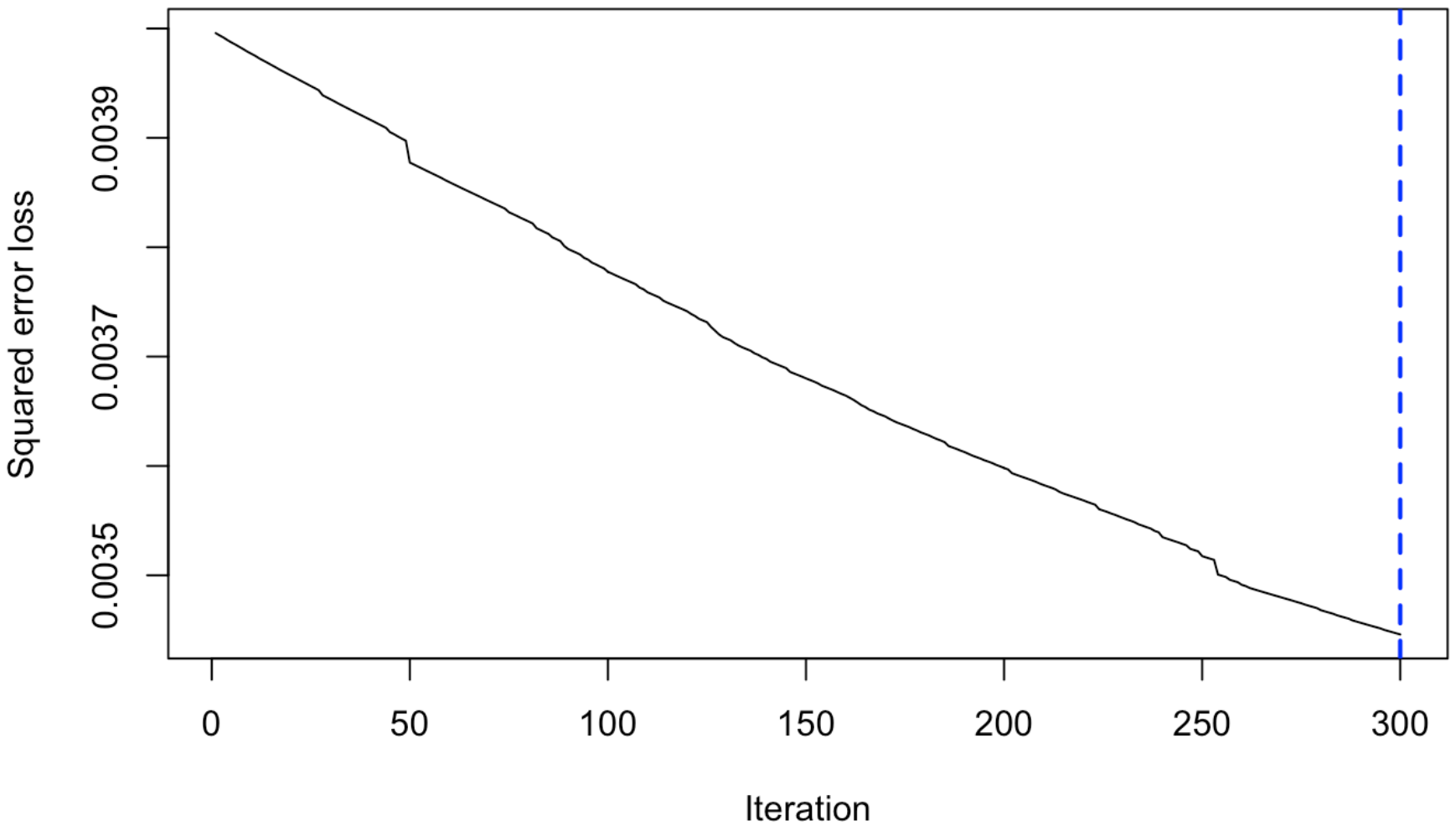

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

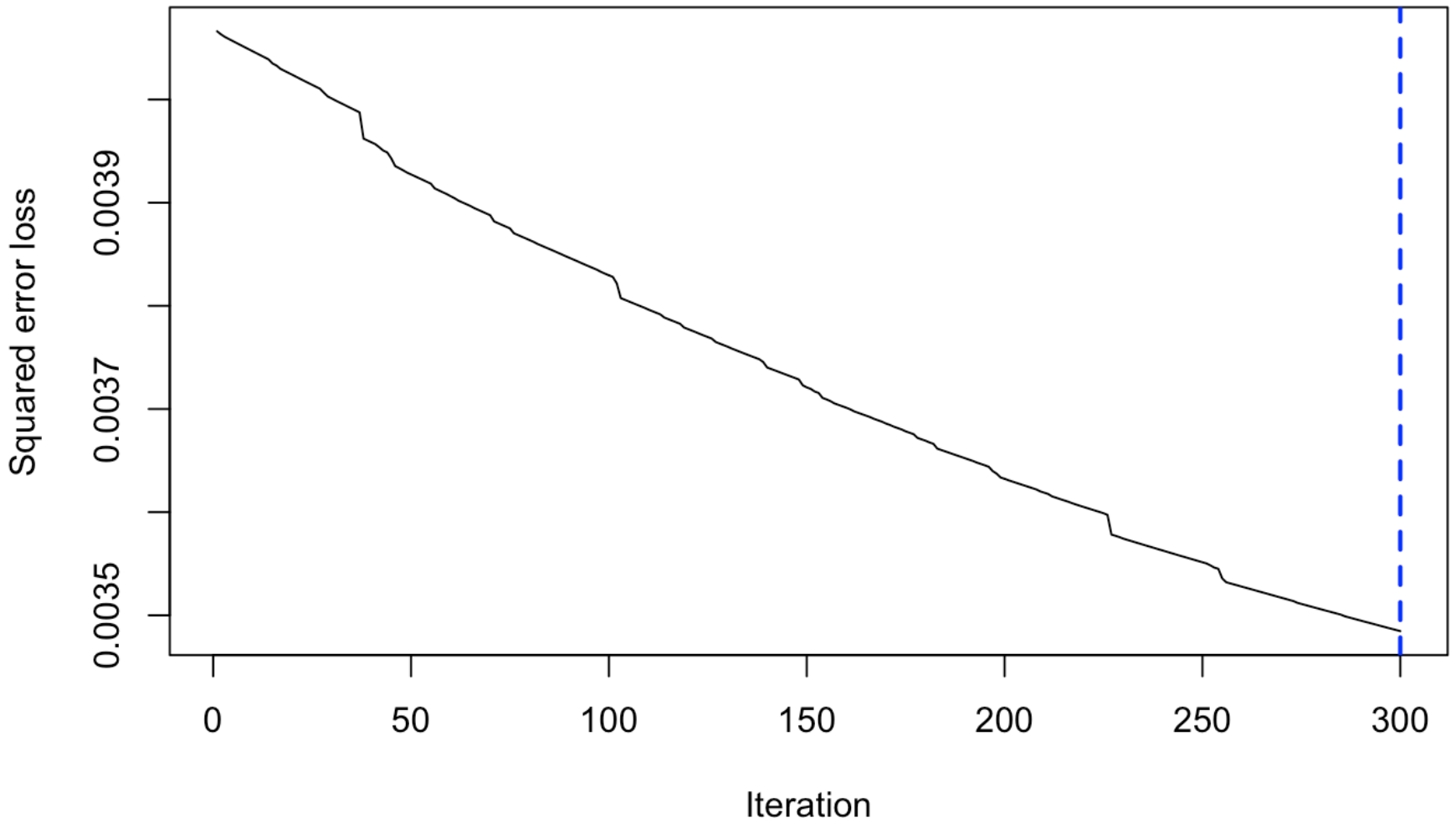
Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration

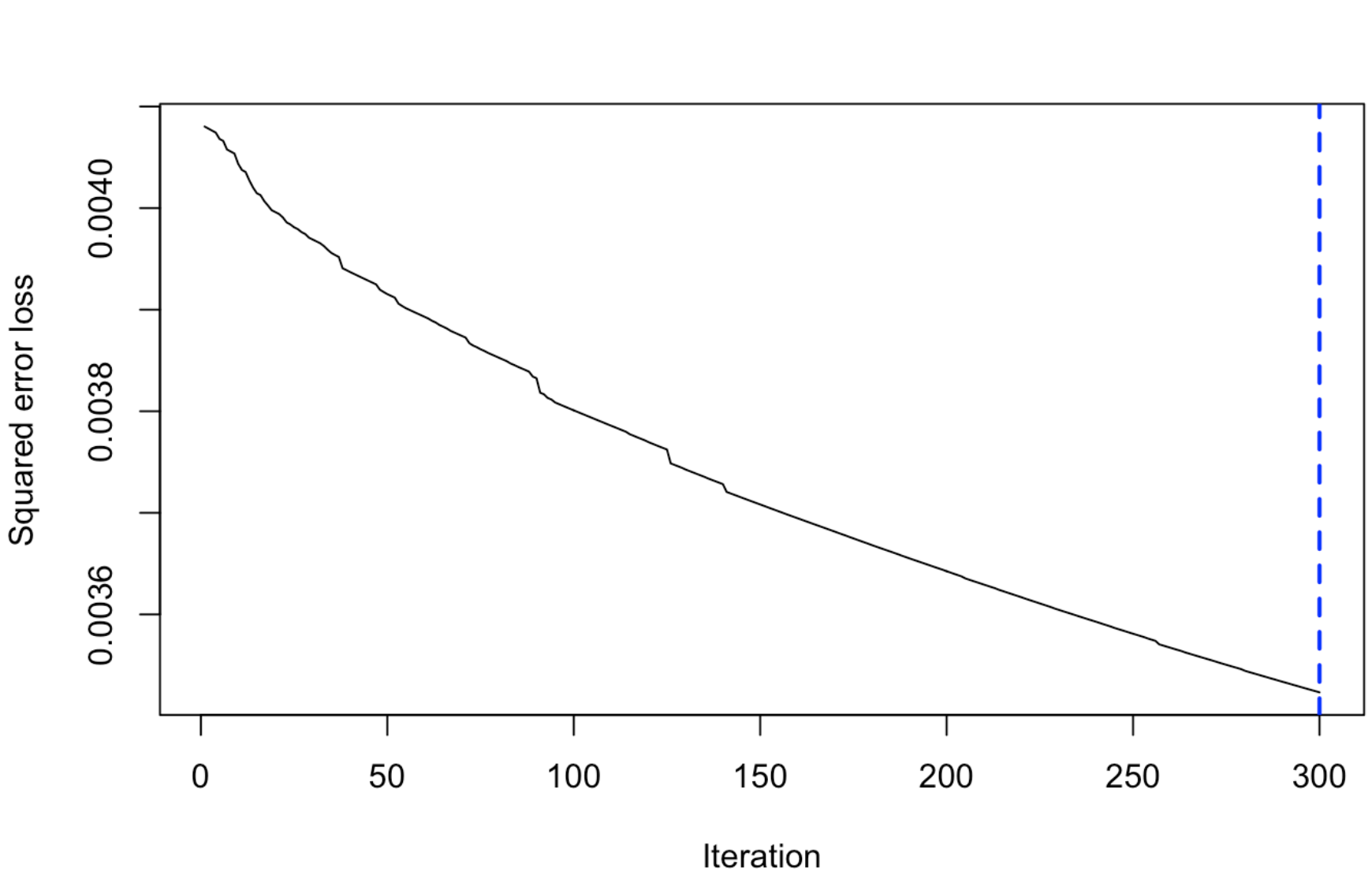

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

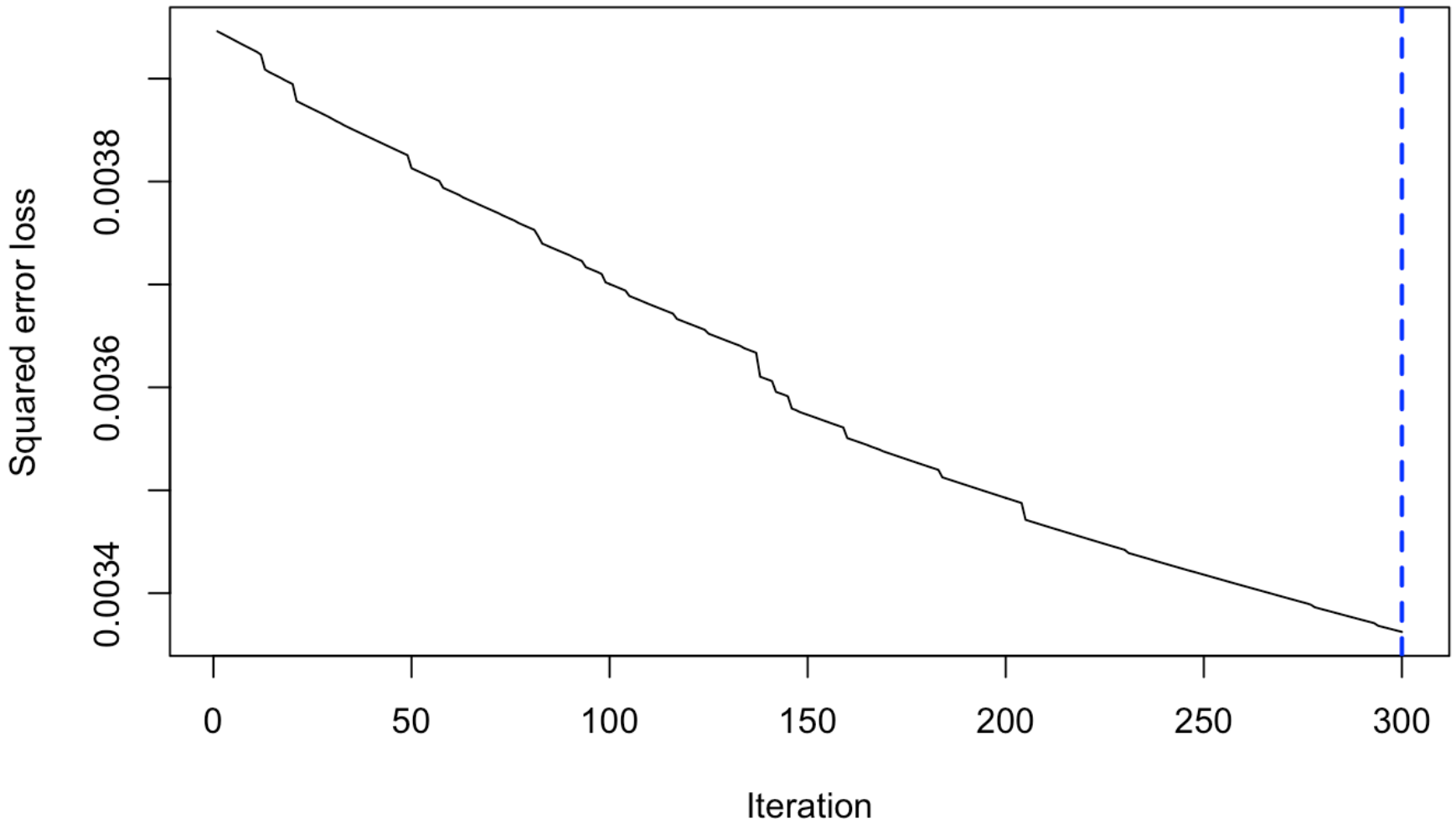
Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

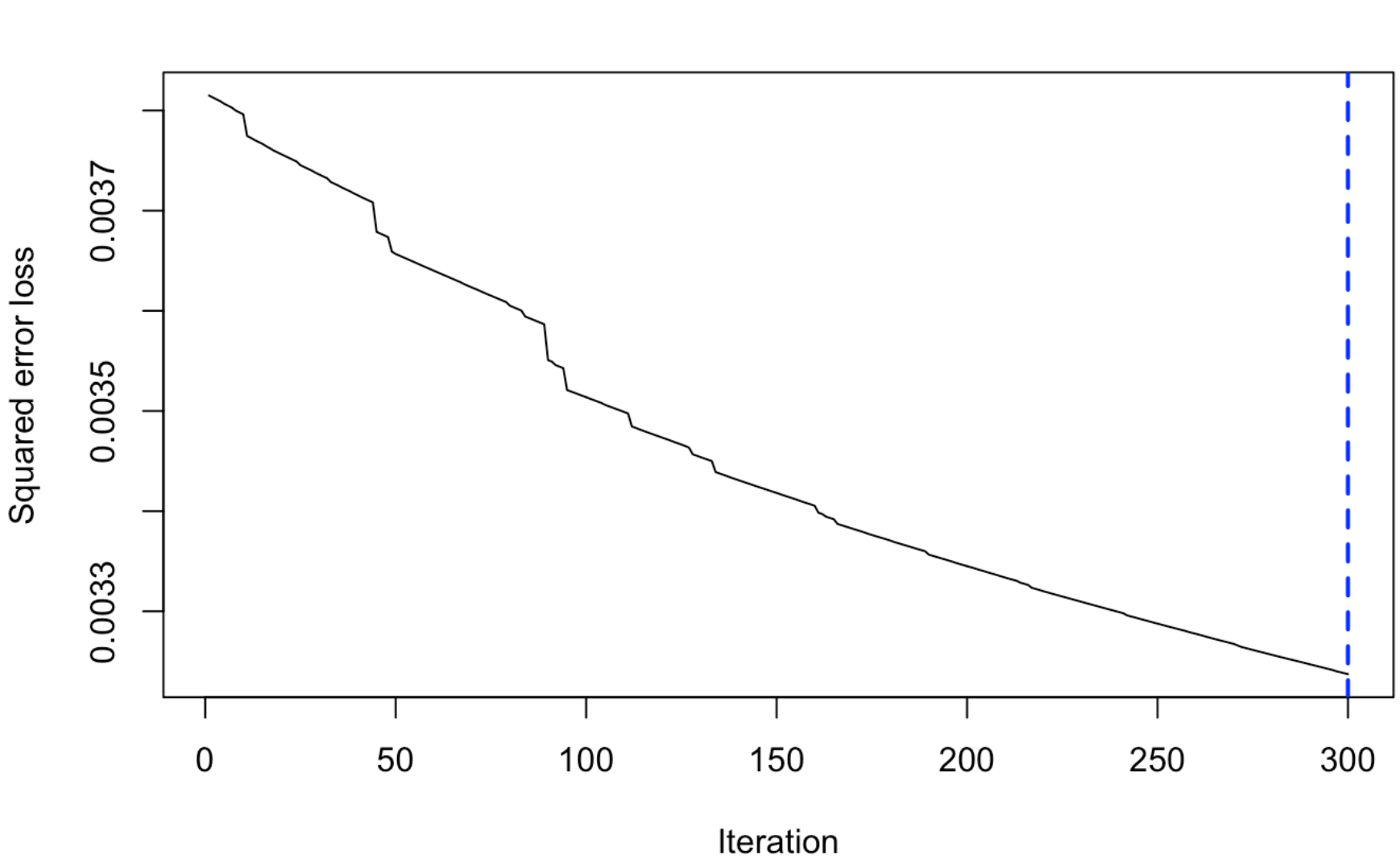

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

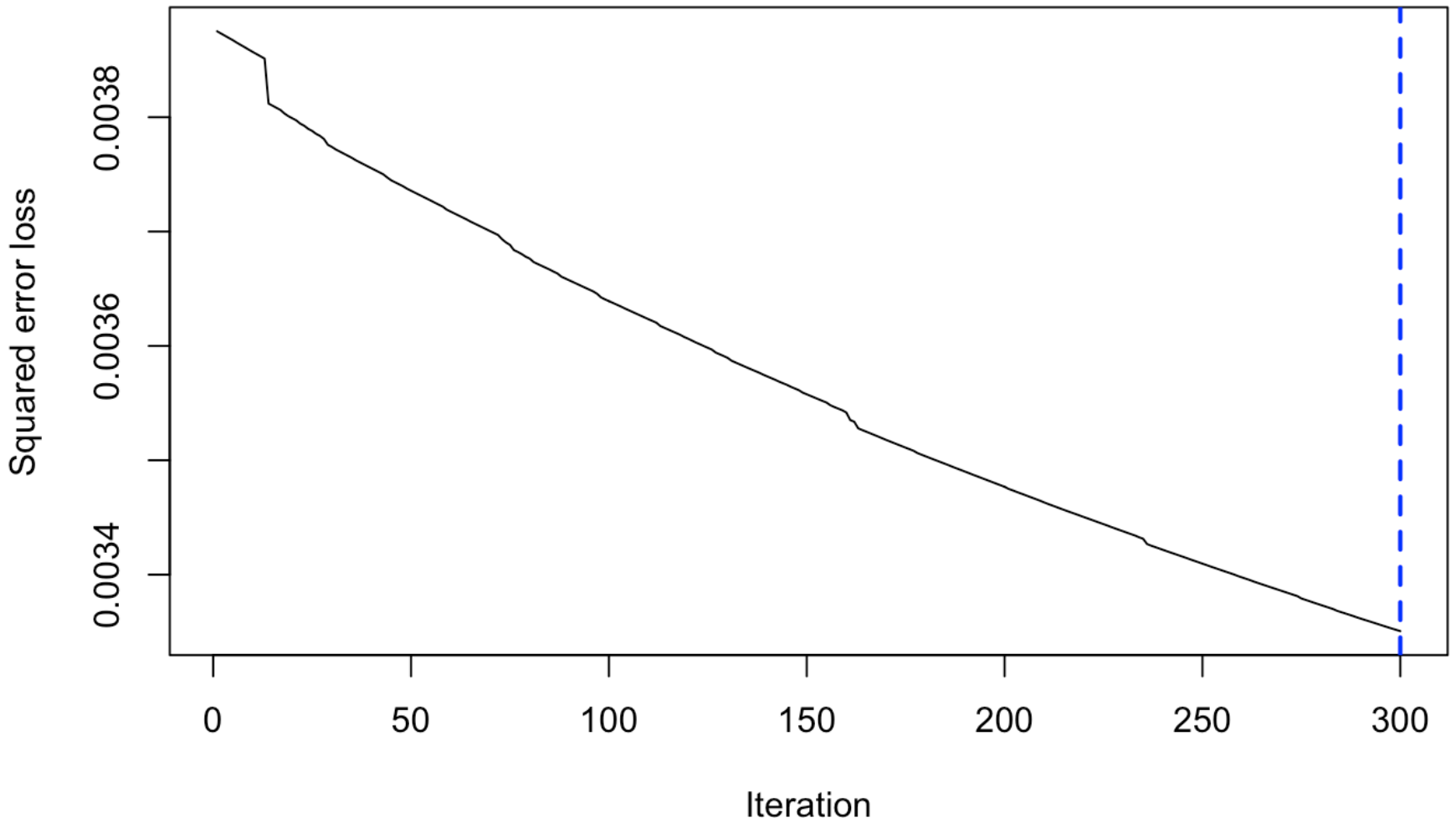

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

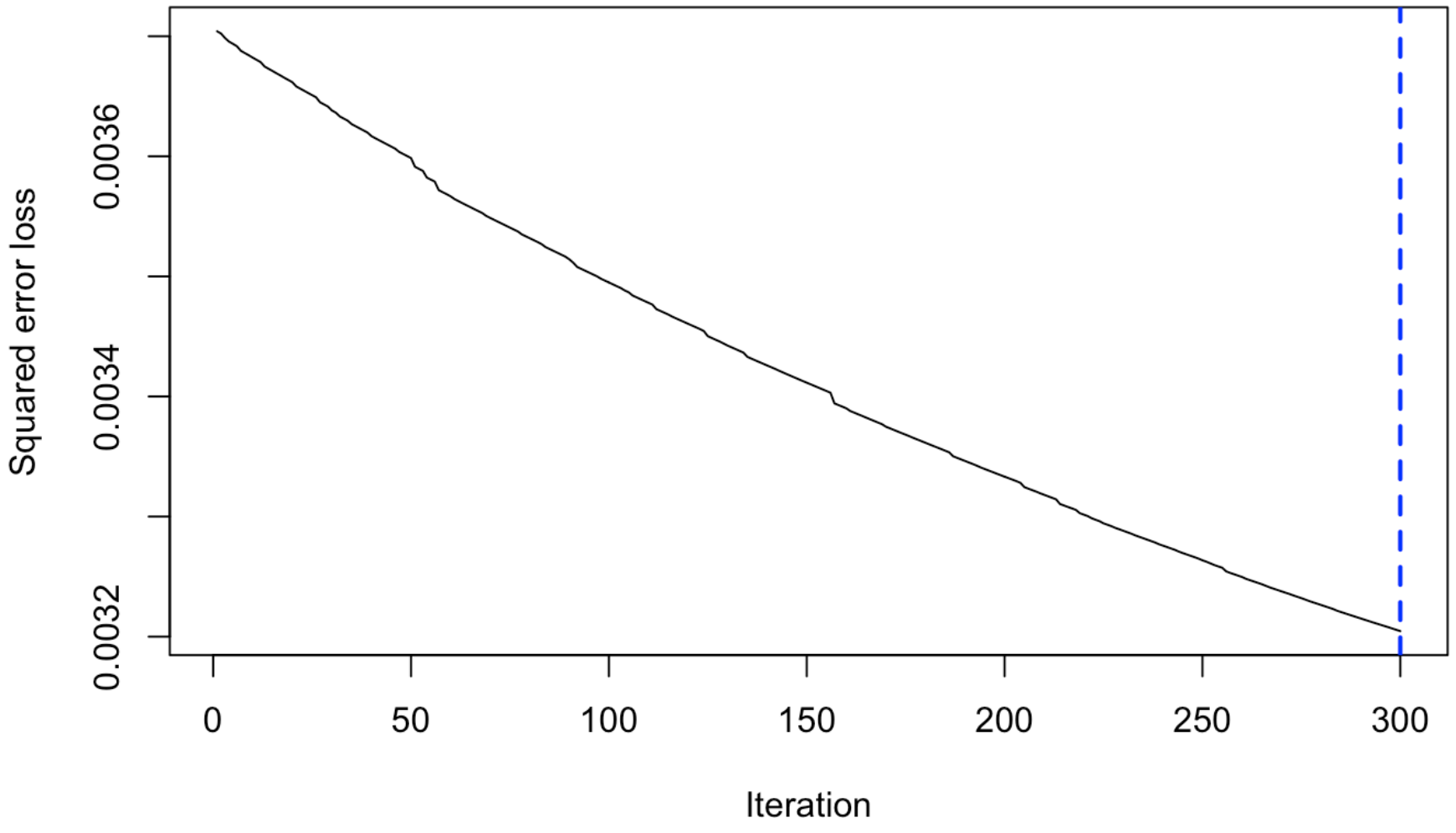

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration

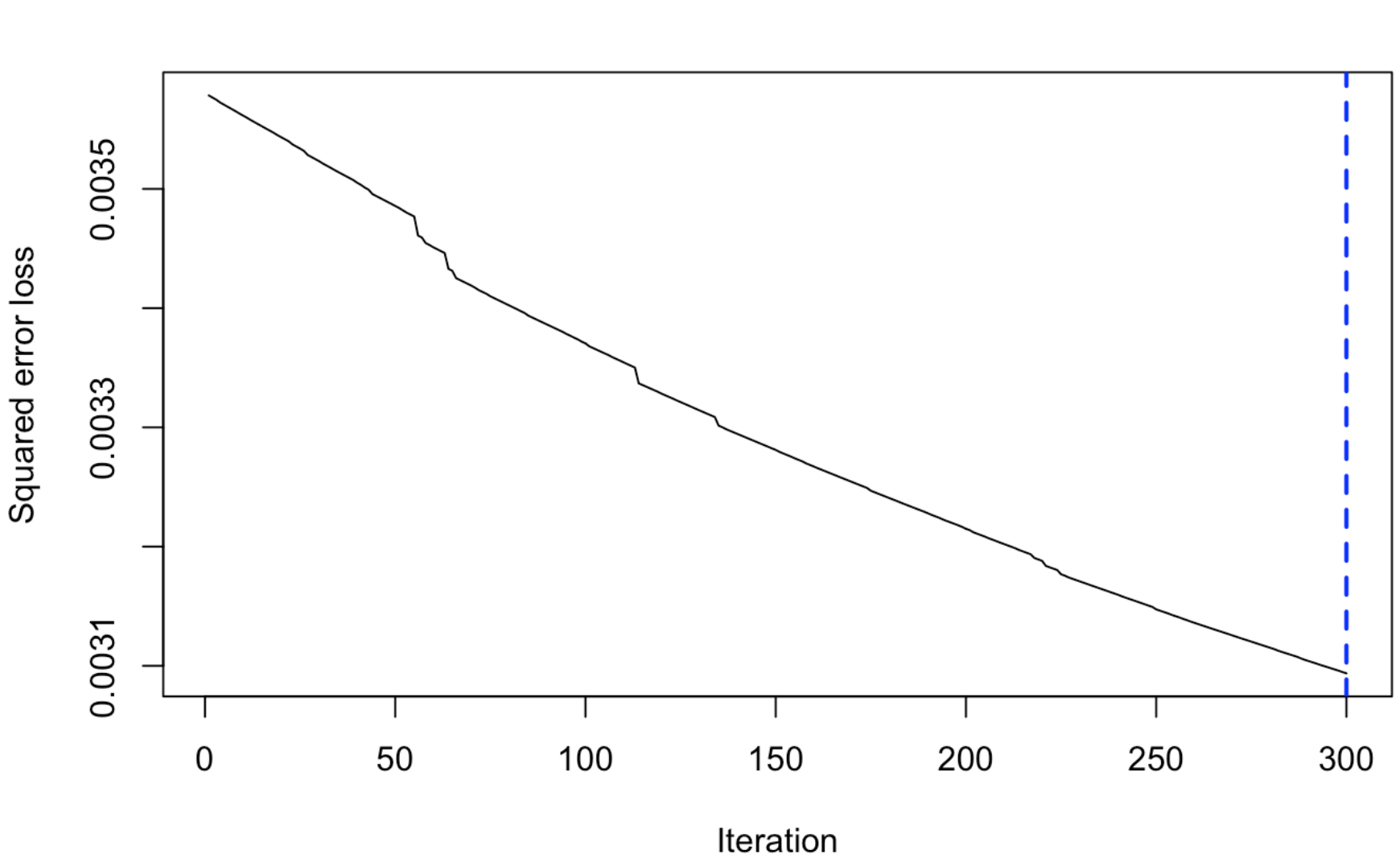

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

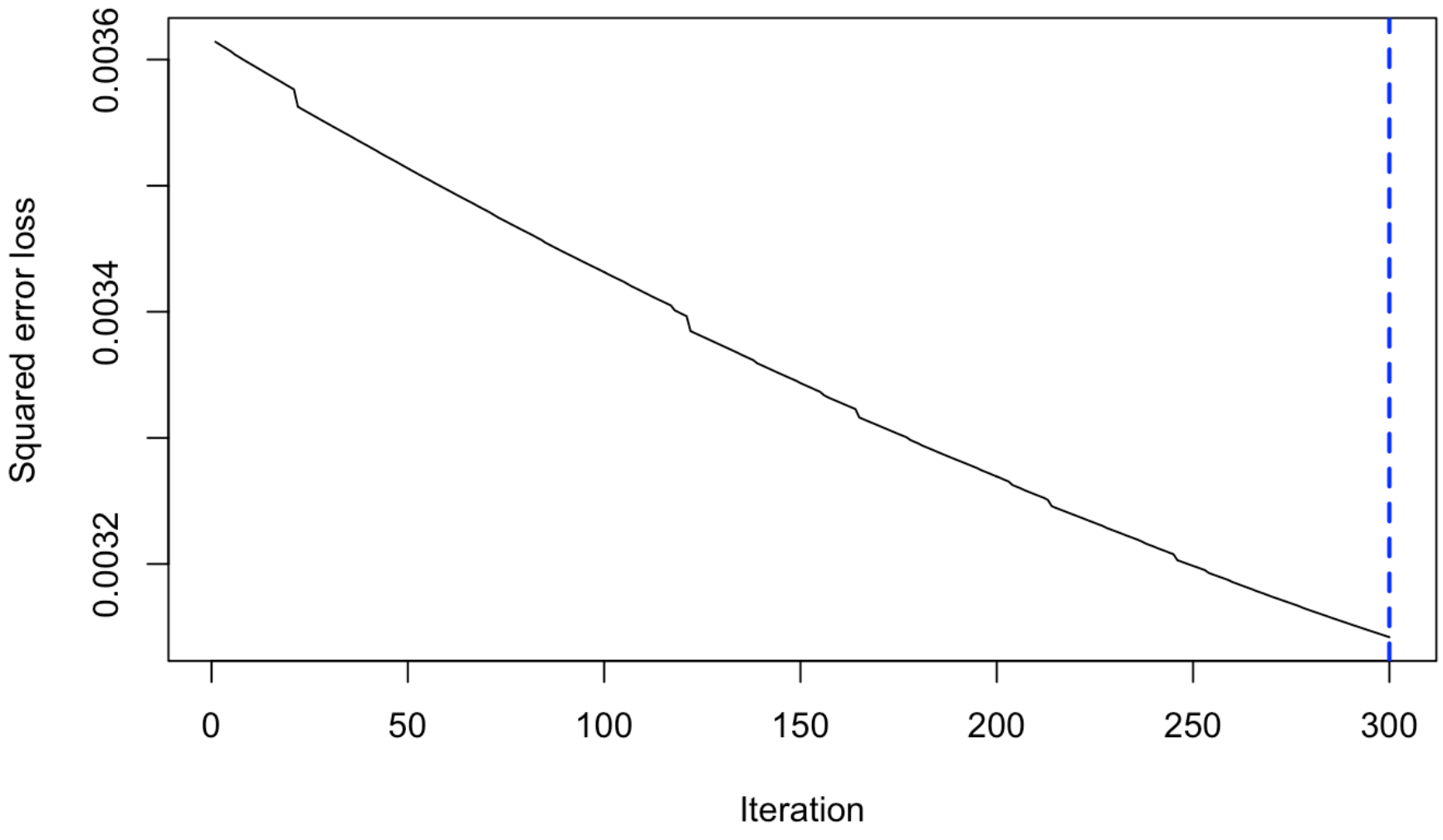

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

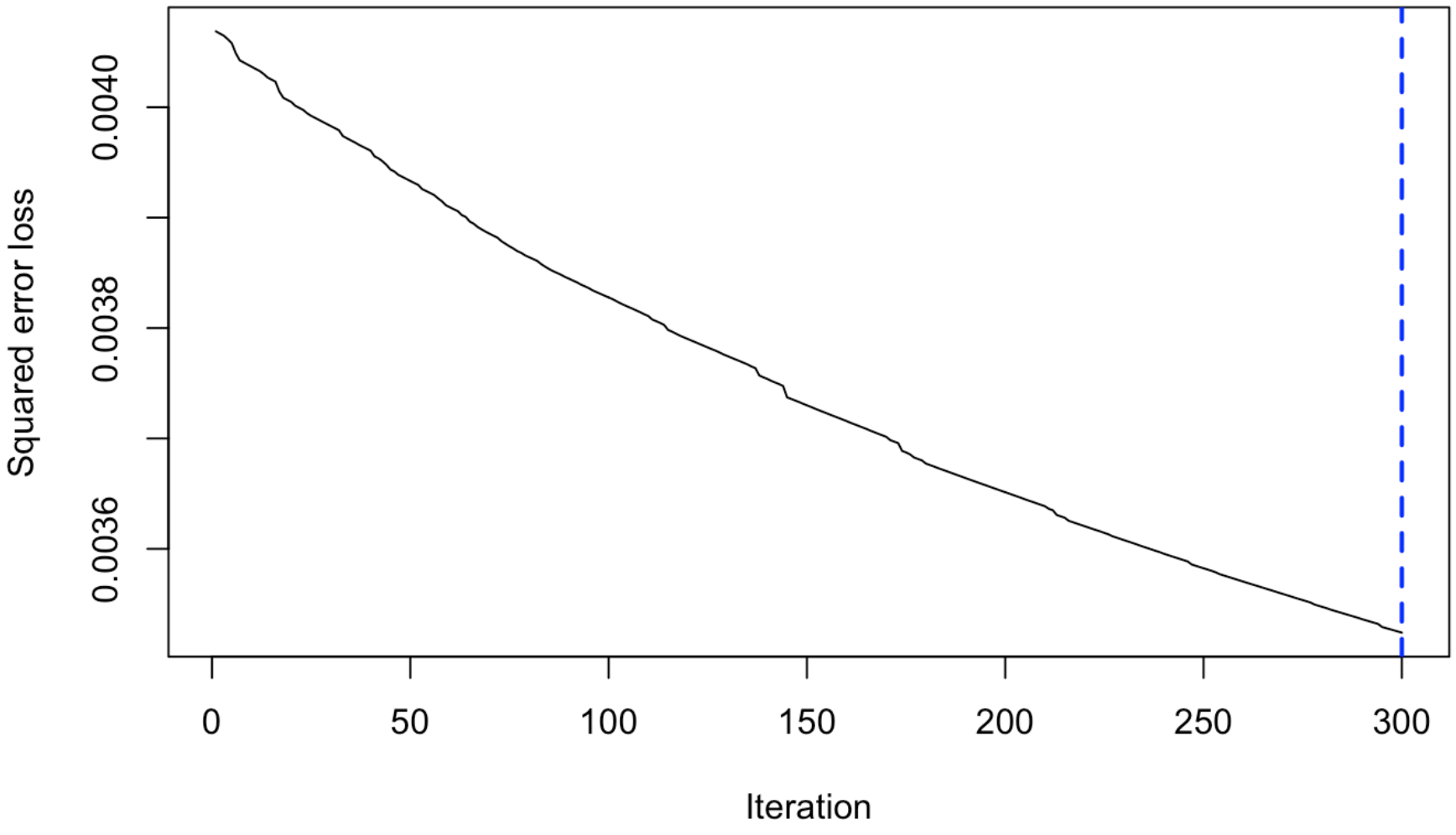
Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

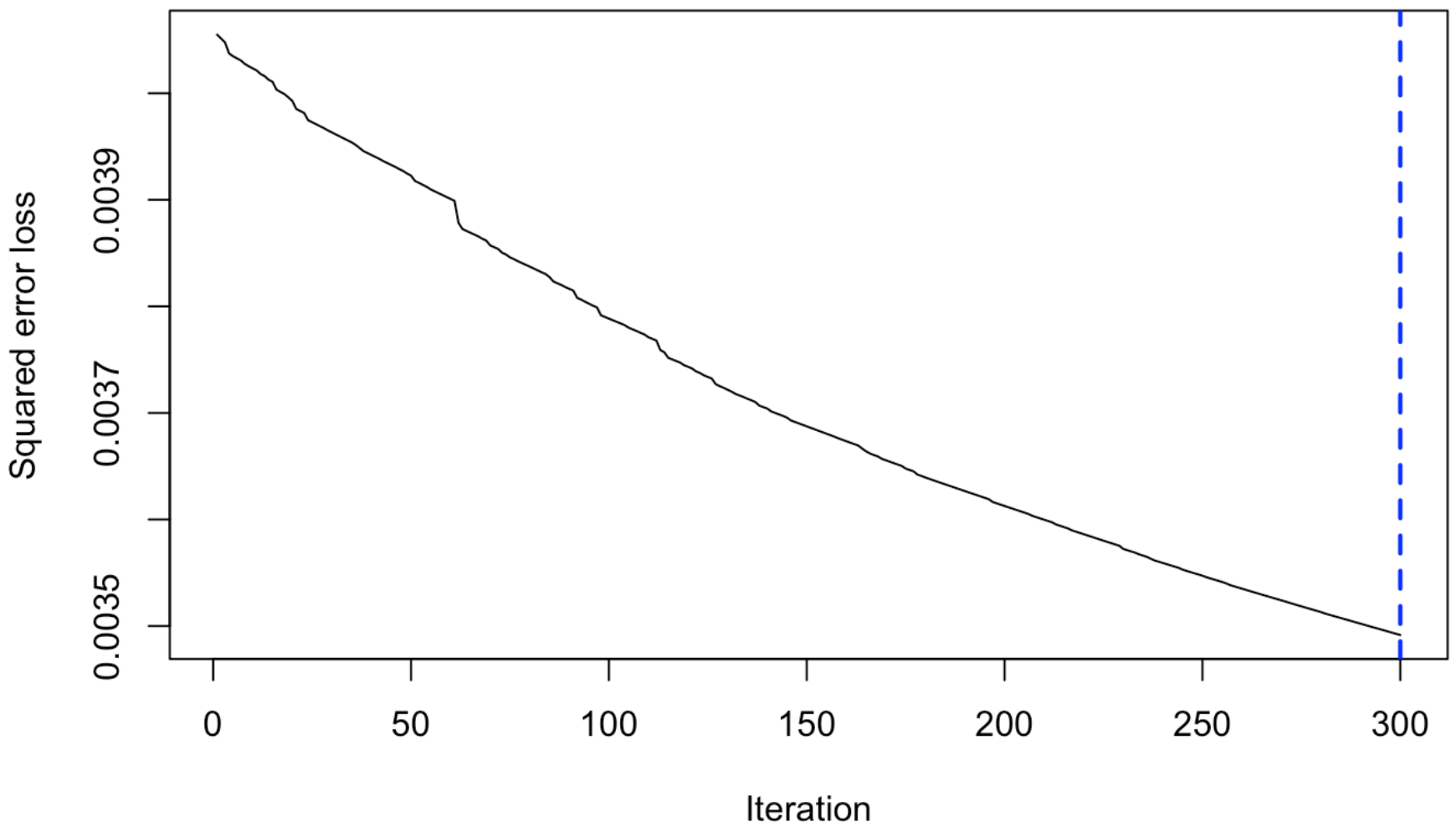

Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

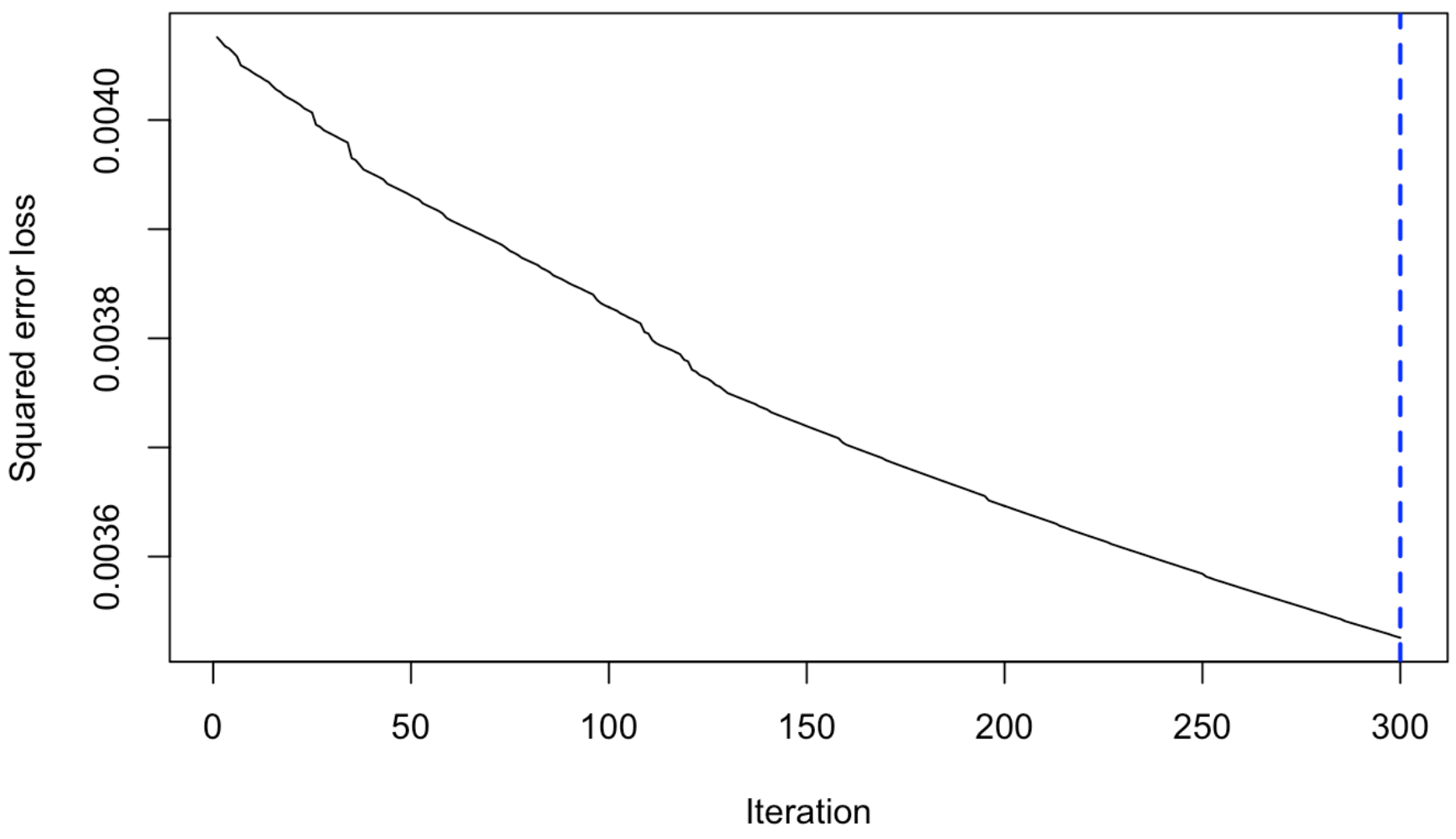
Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

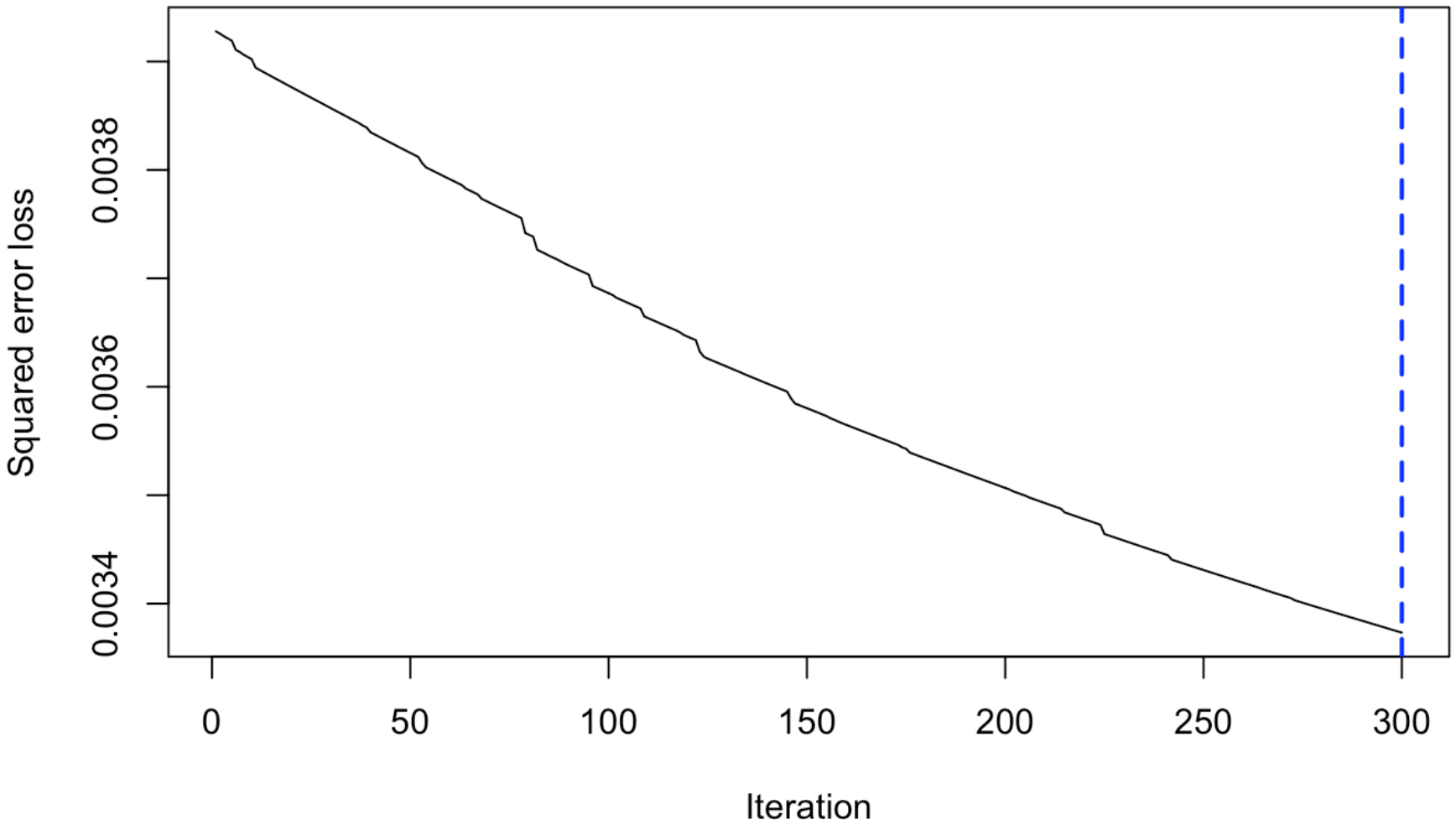

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

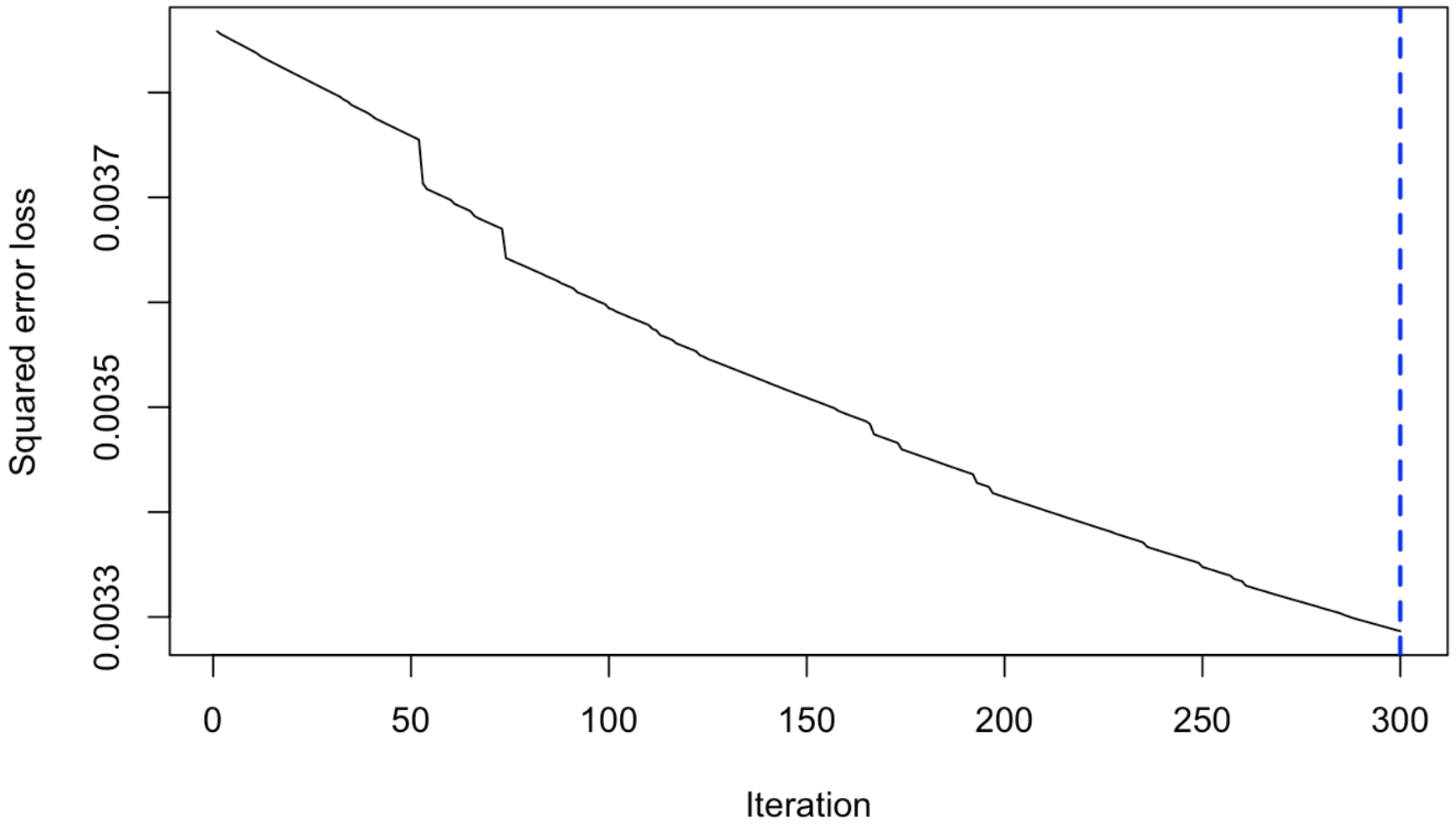

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

```
k=  6
```

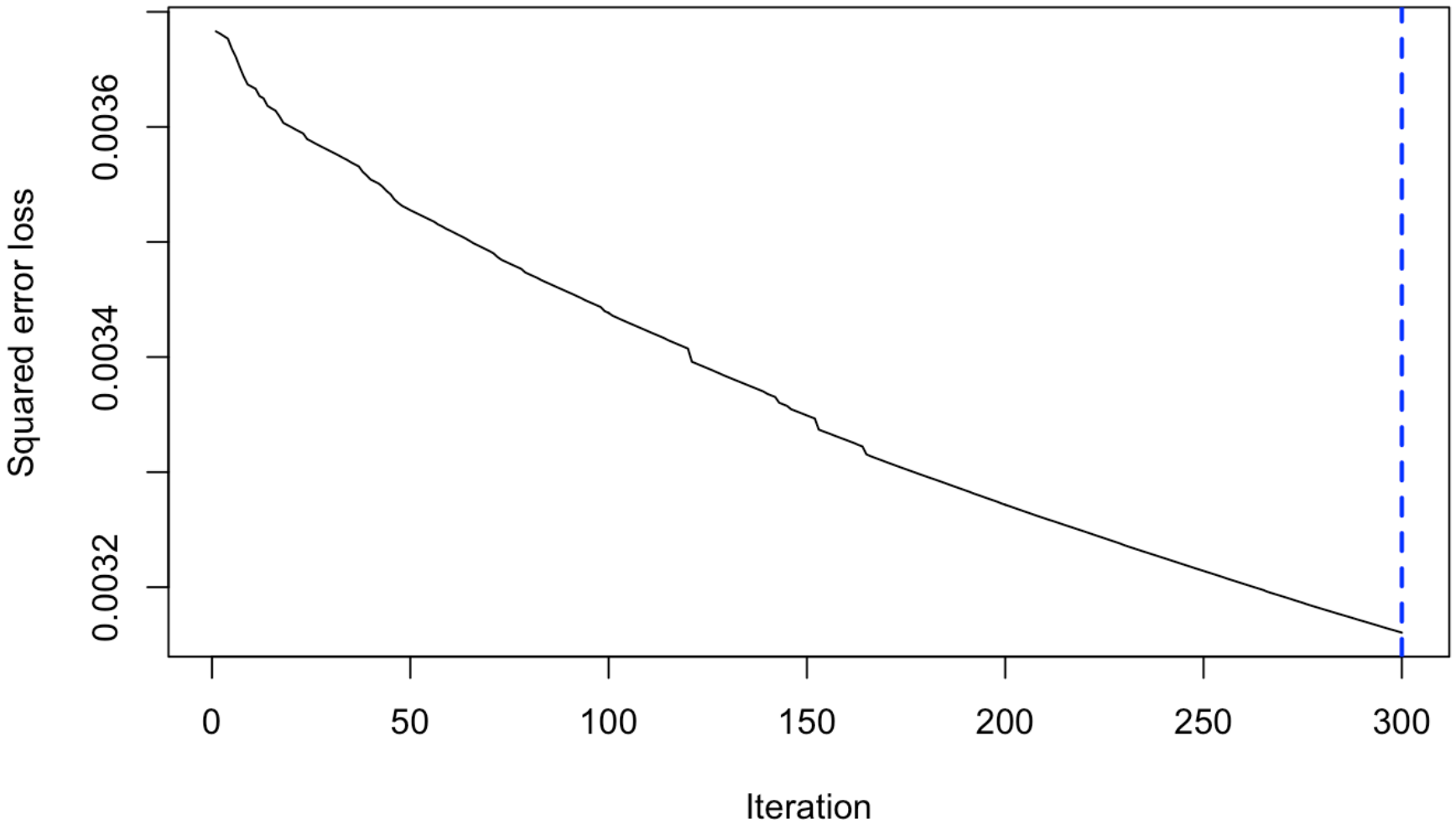

Squared error loss
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

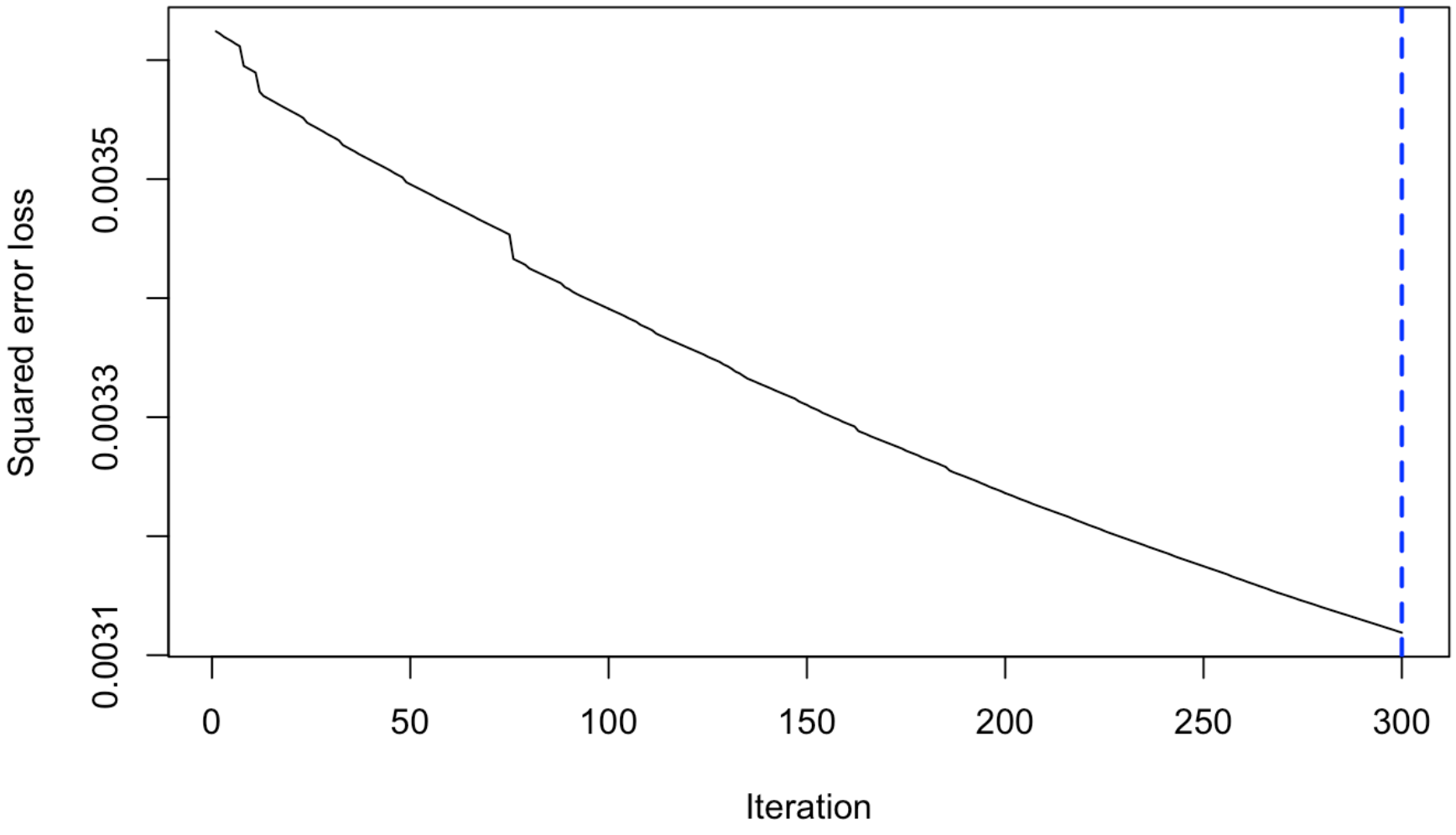
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration

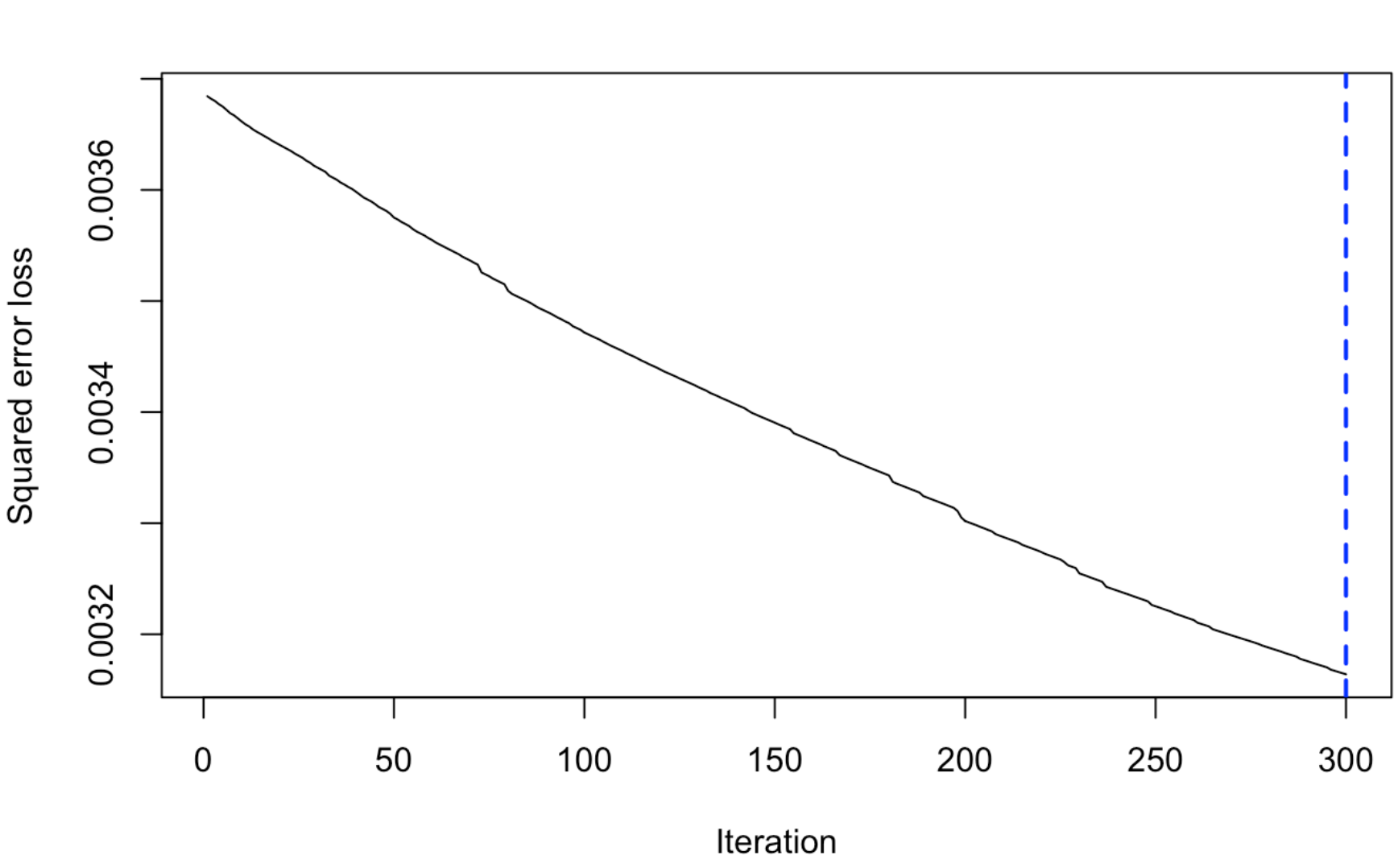

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration

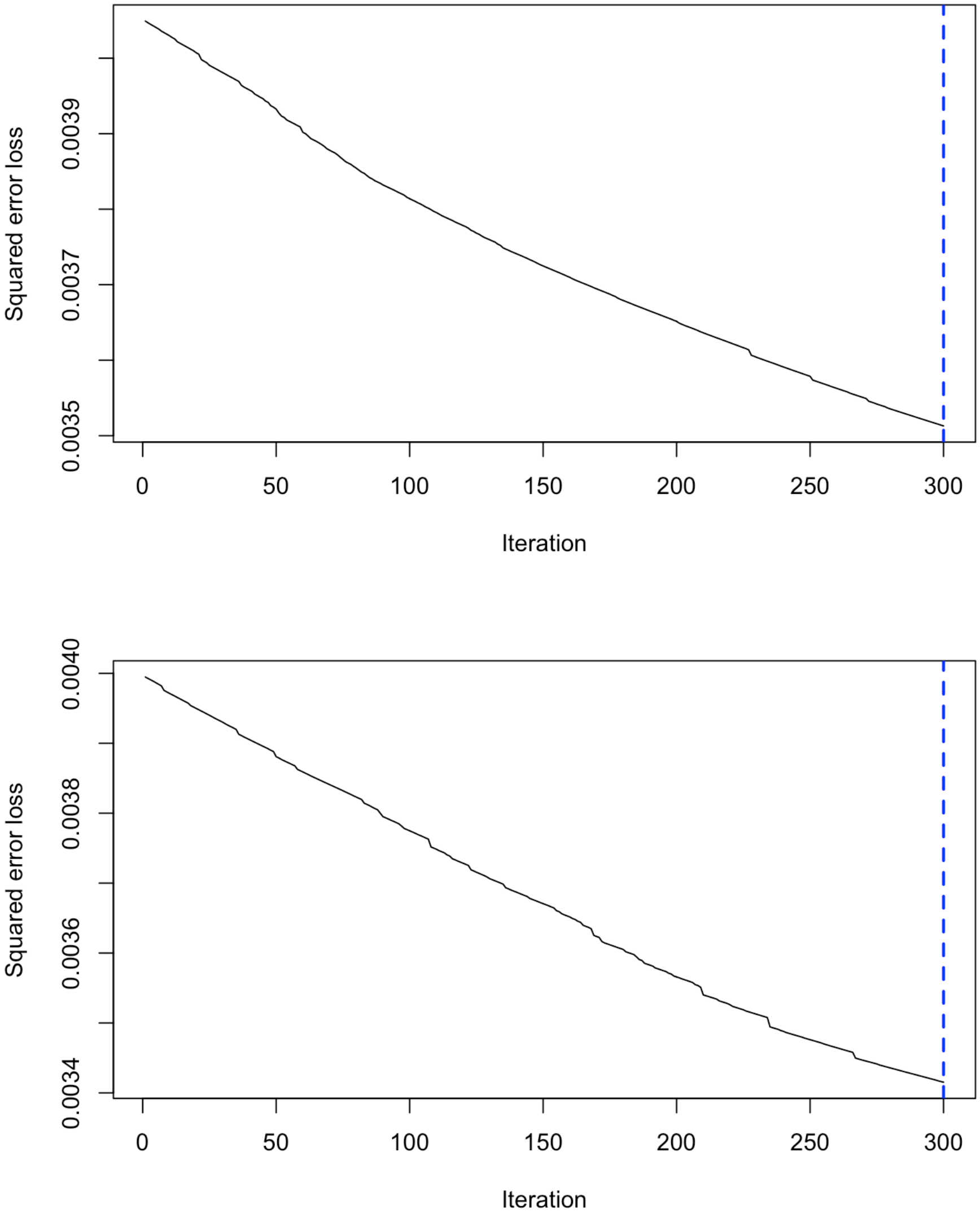
Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

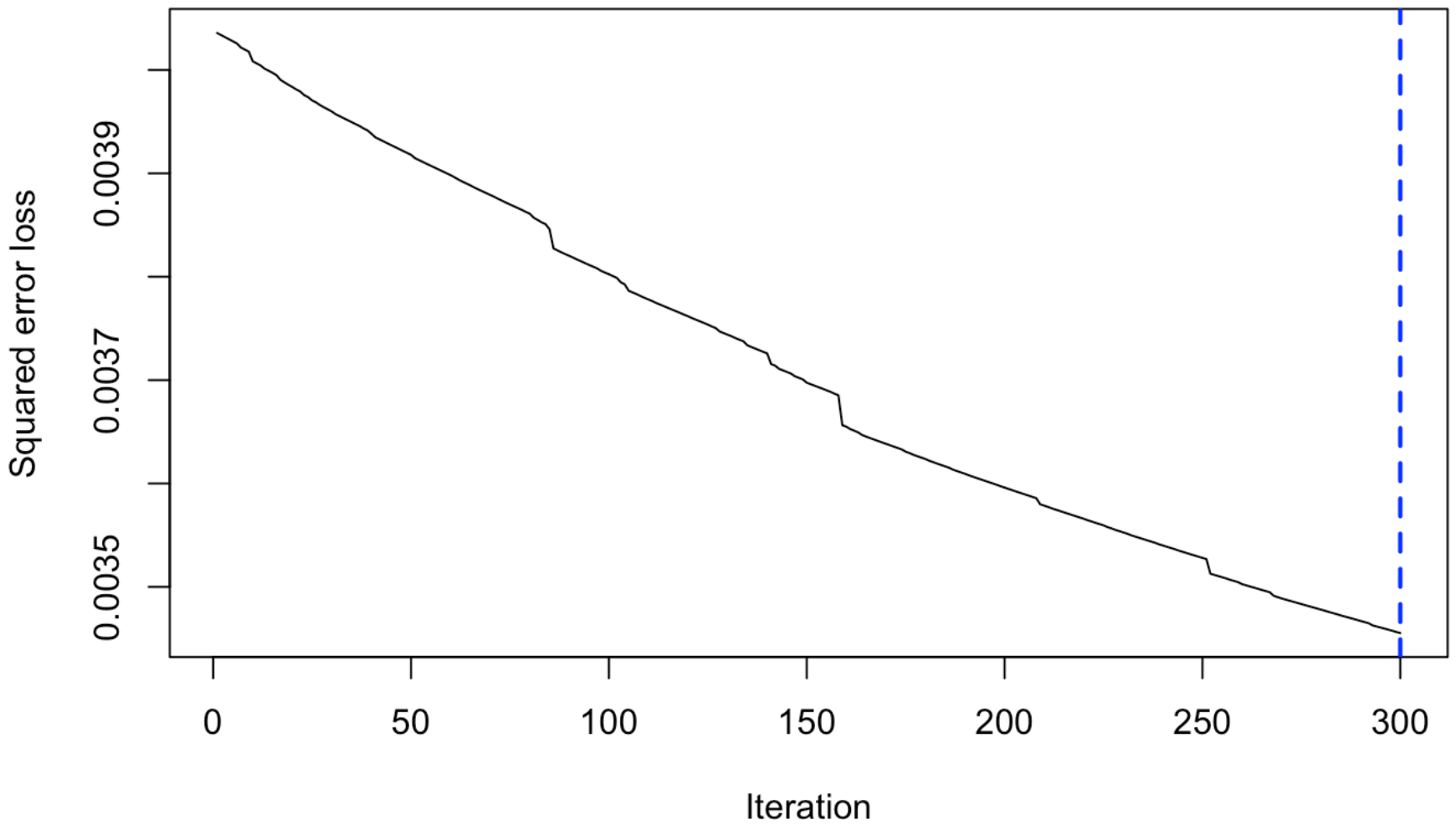

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

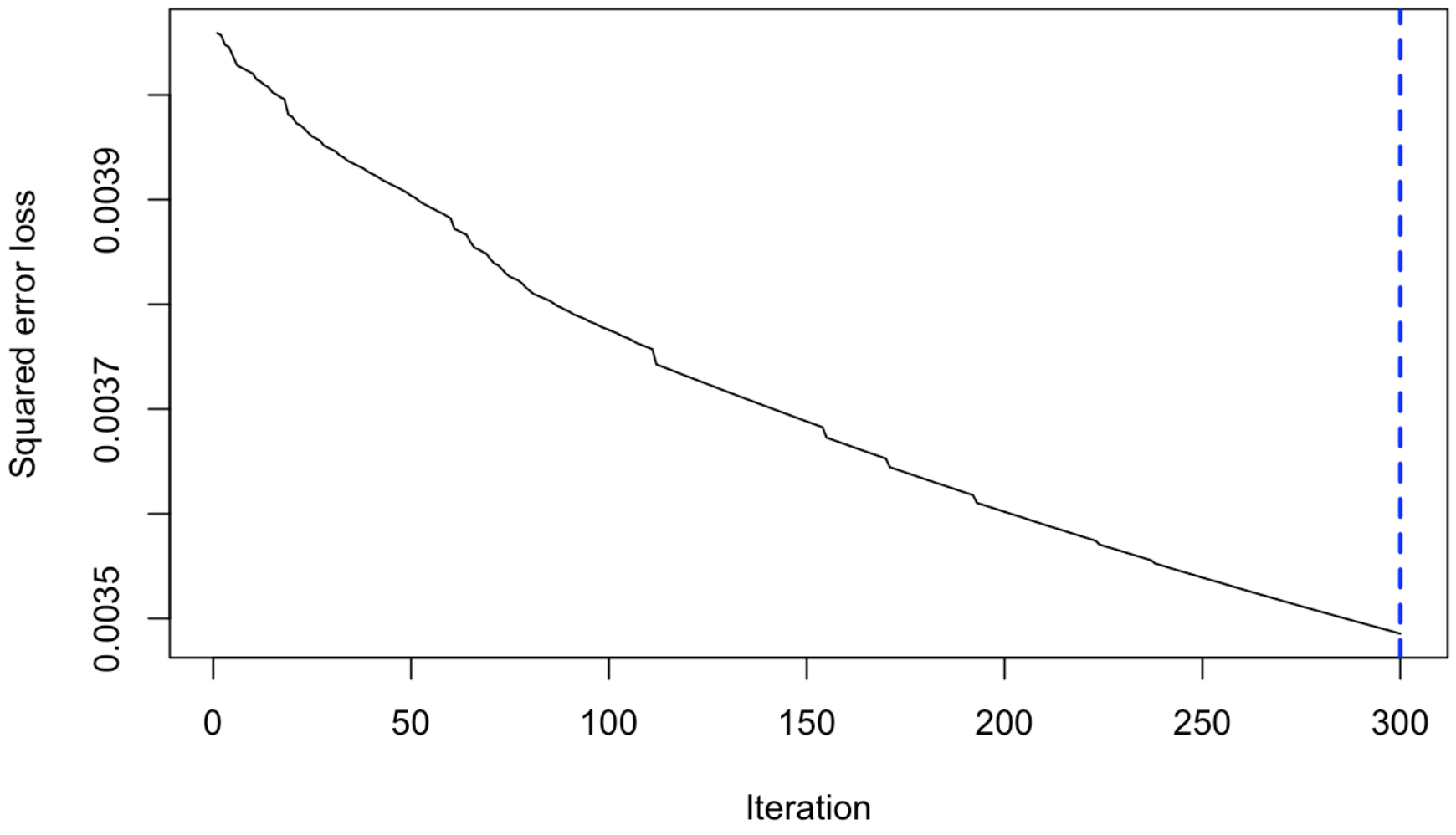
Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration

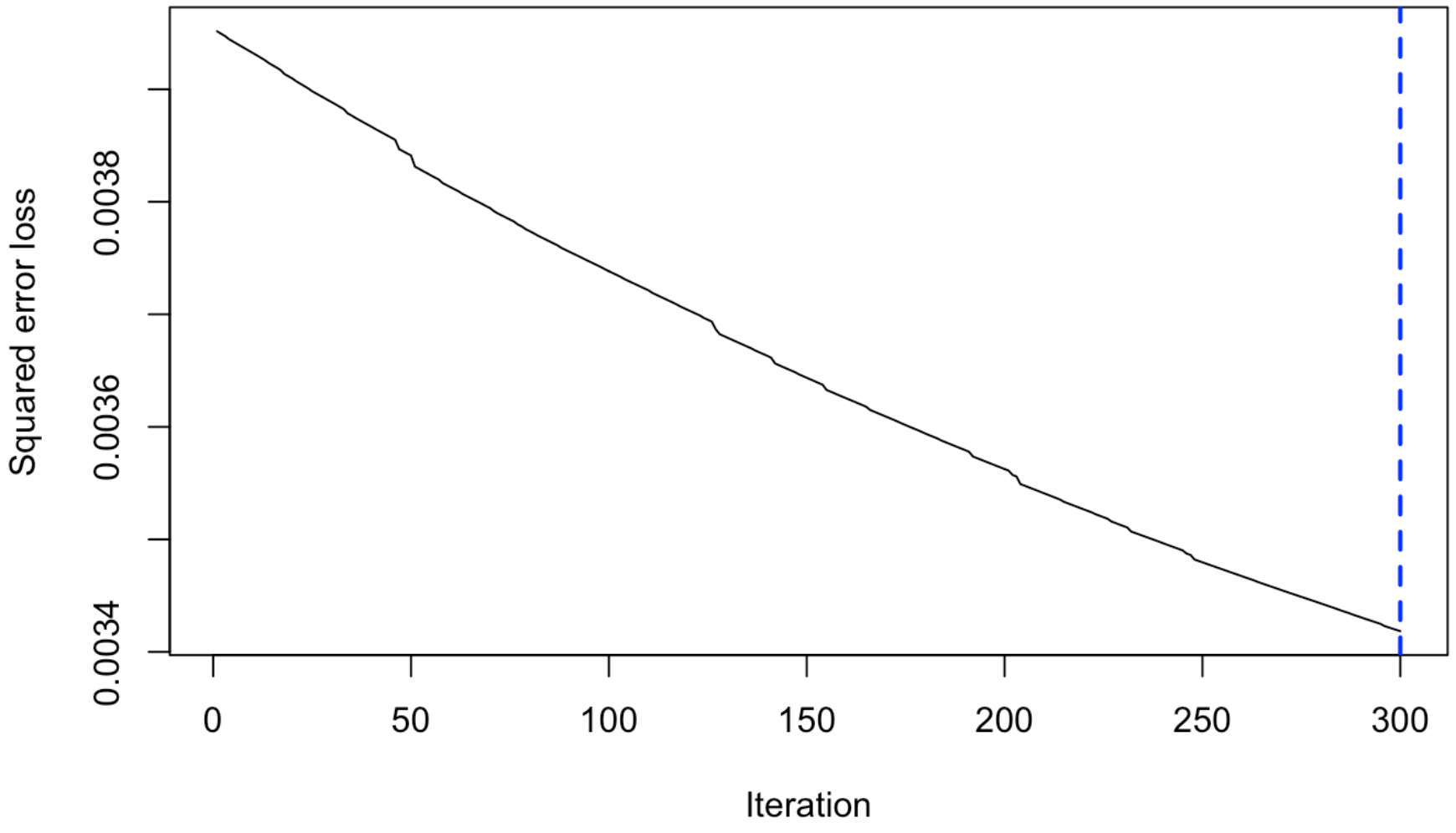

Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

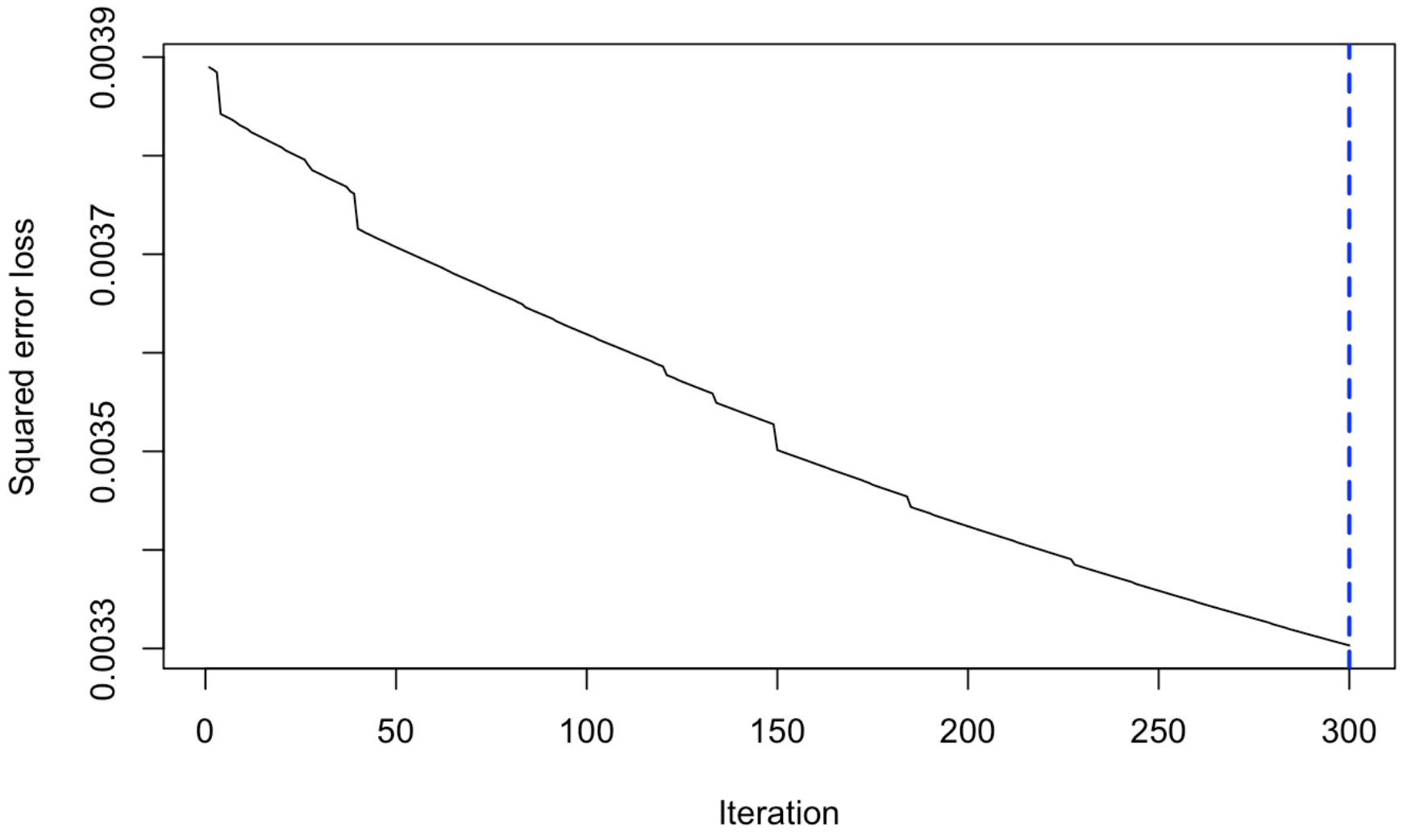

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration
Squared error loss
0.0034
0.0036
0.0038
0.0040
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

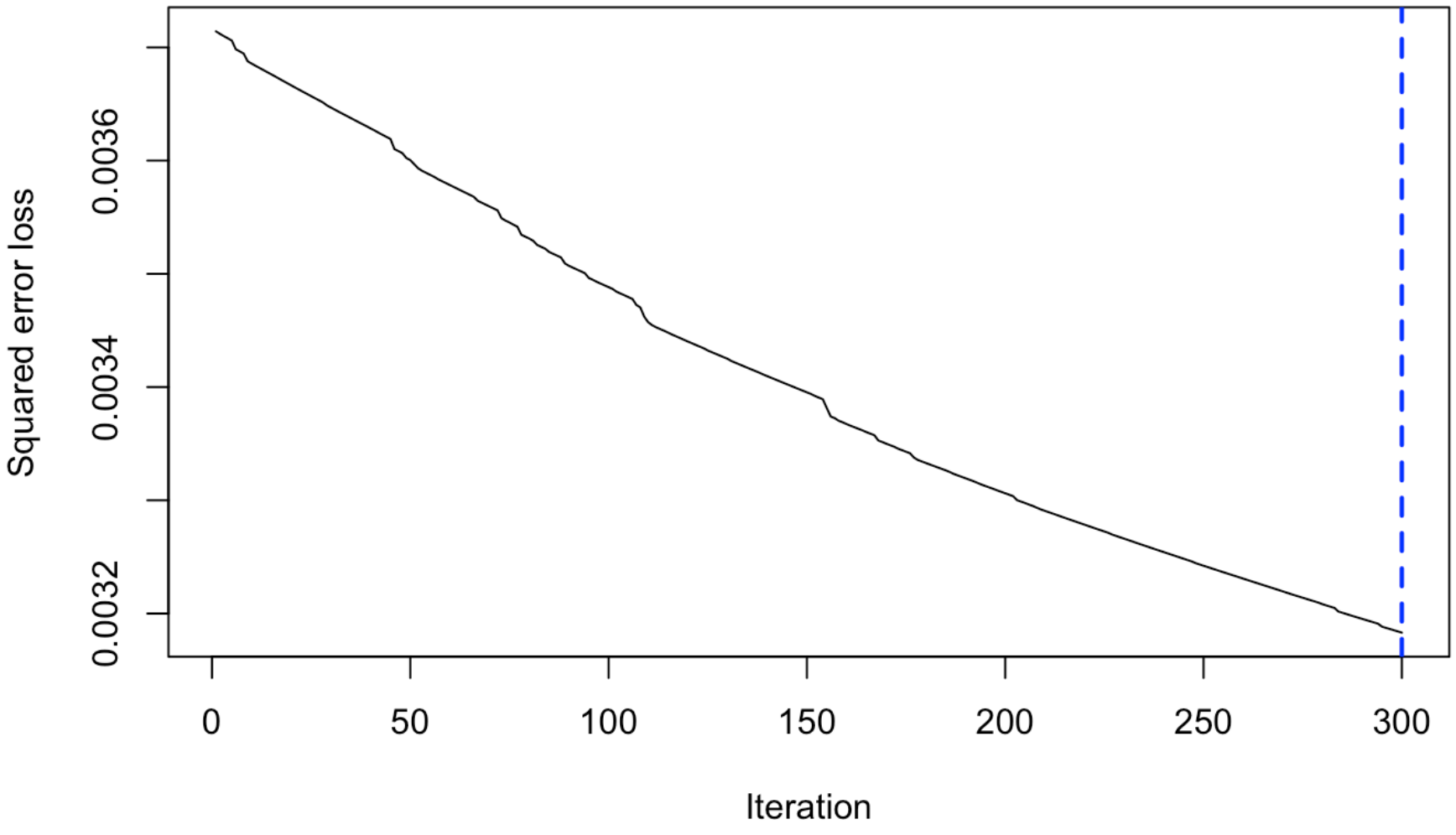

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

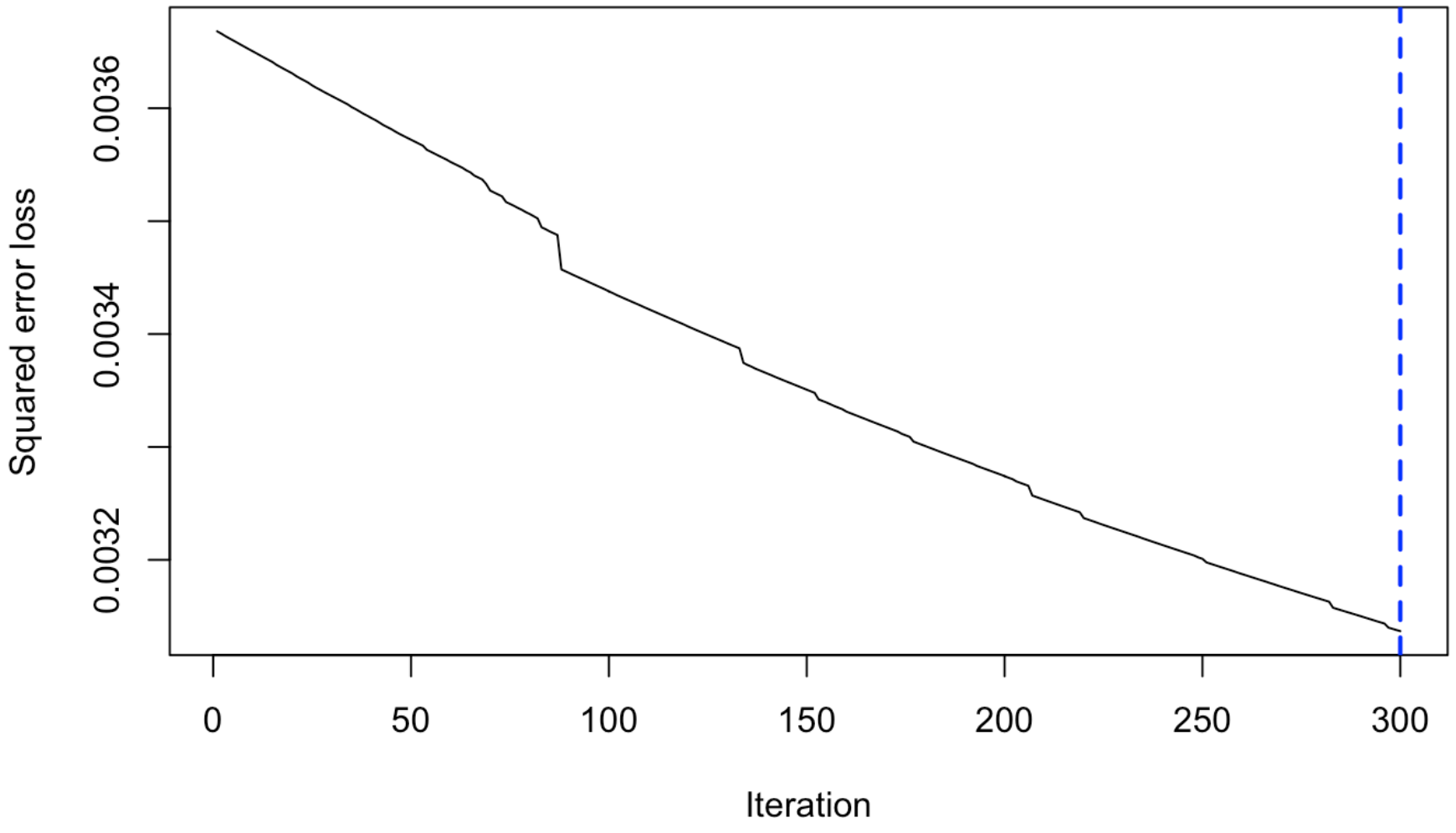
Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0039
0.0037
0.0035
0.0033
0
50
100
150
200
250
300
Iteration

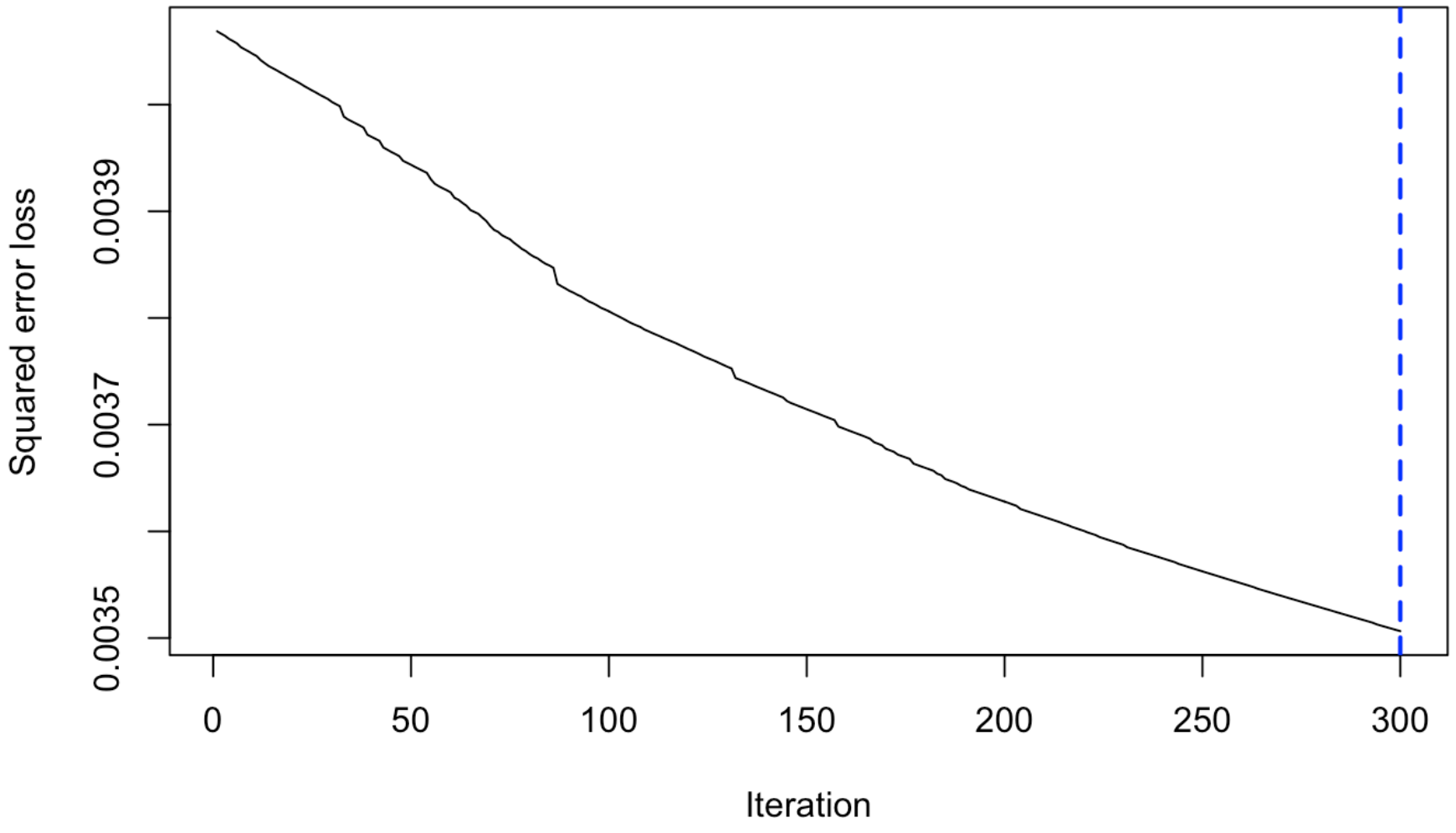
Squared error loss
0.0038
0.0036
0.0034
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

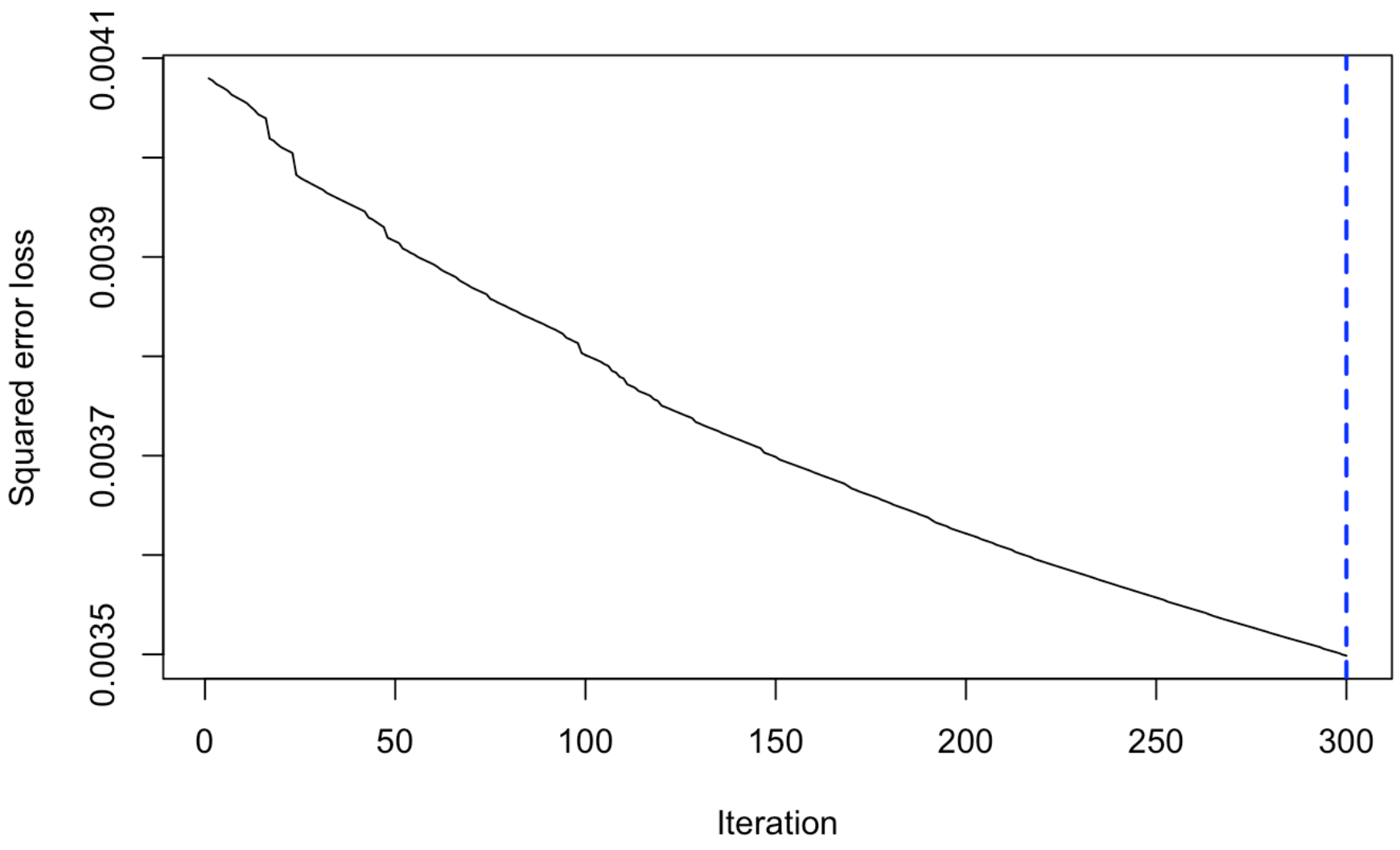
Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

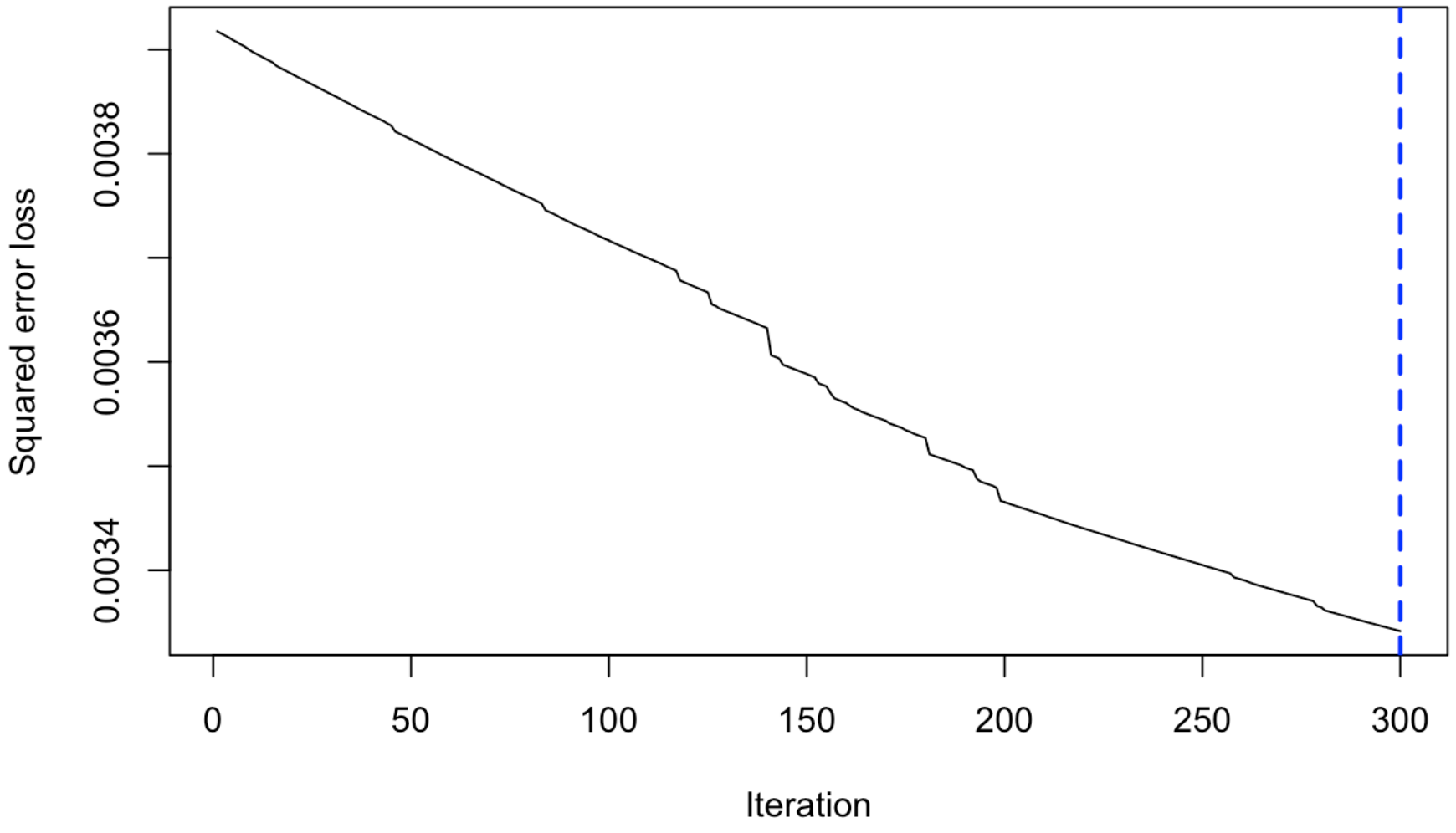
Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

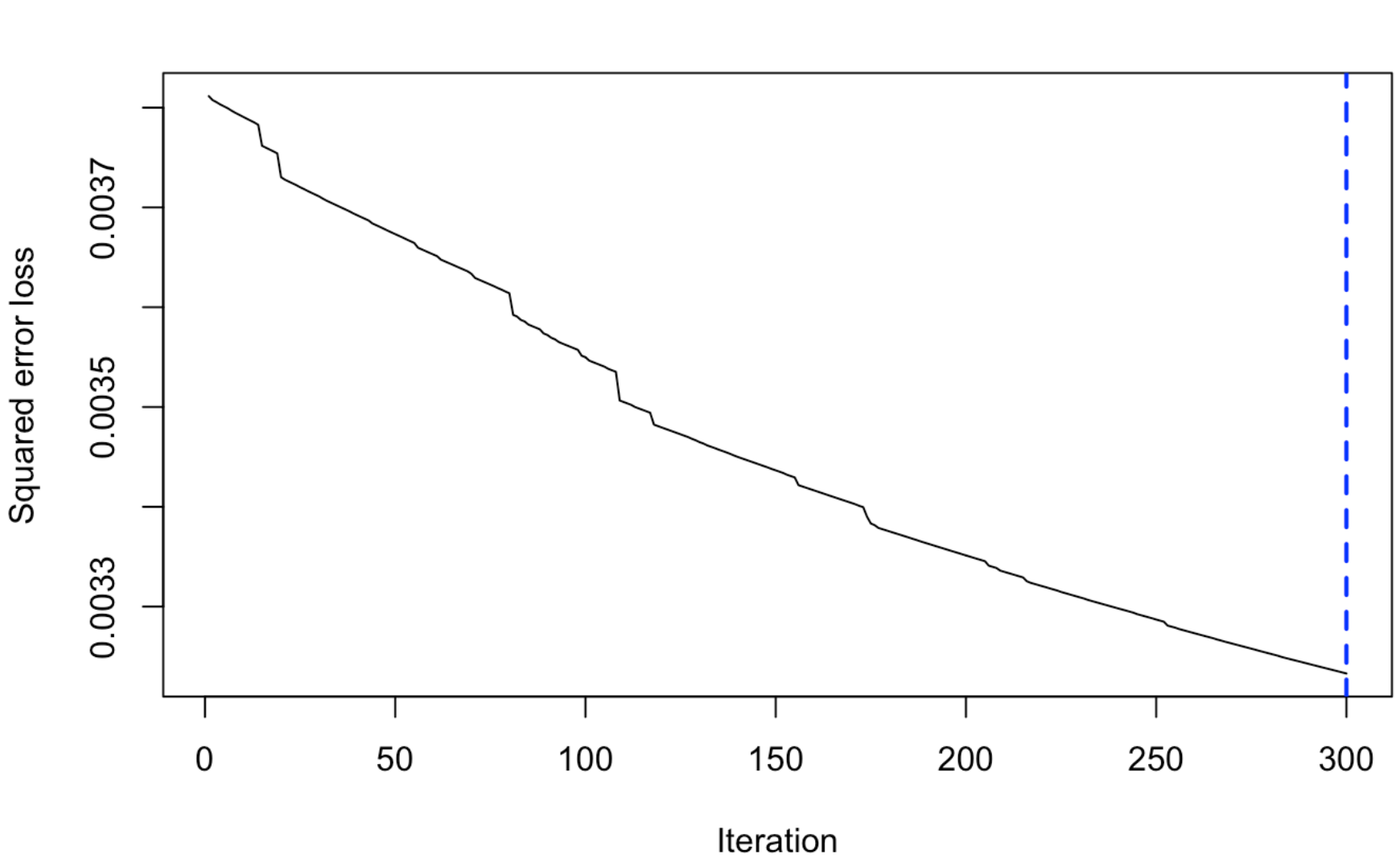

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

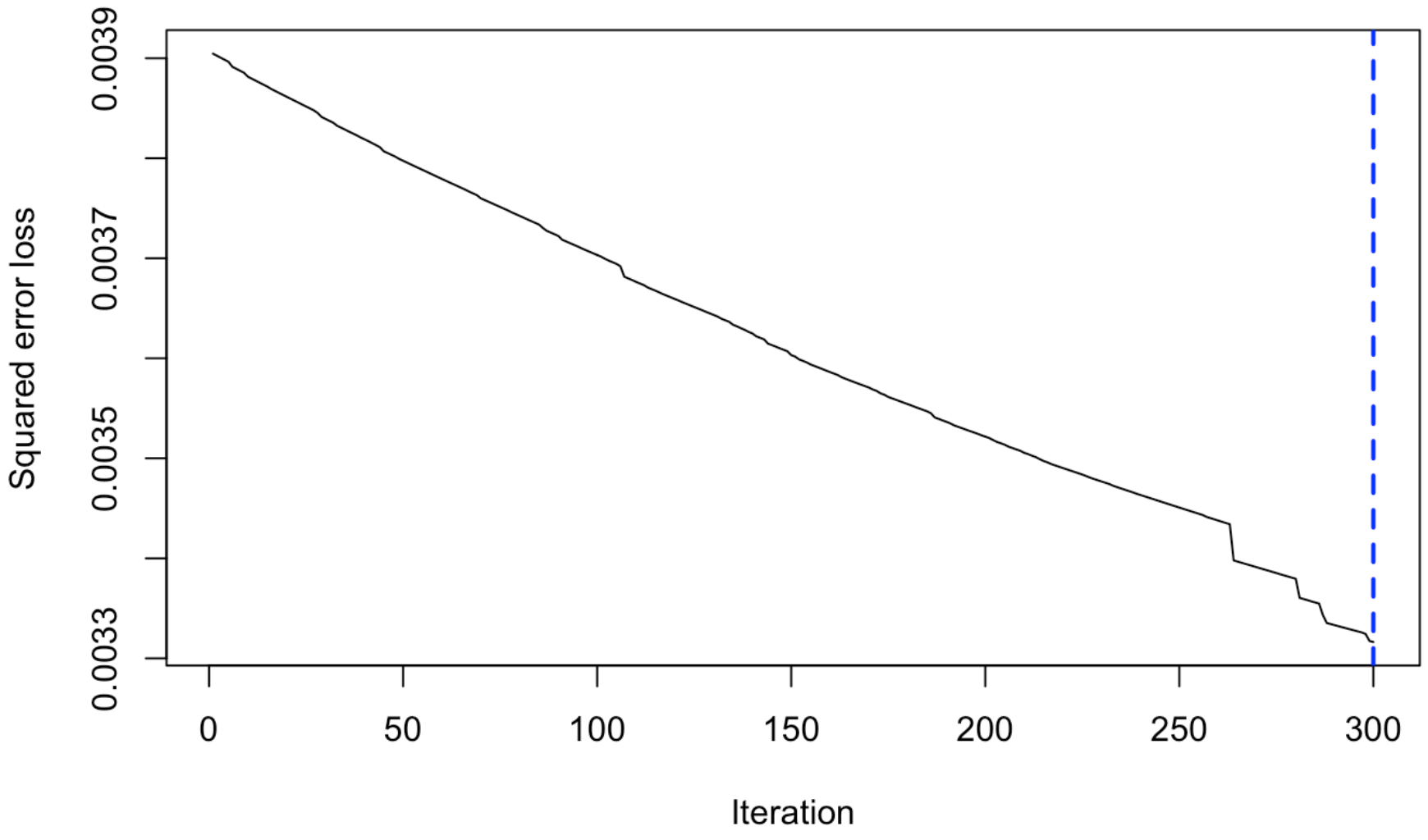
Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

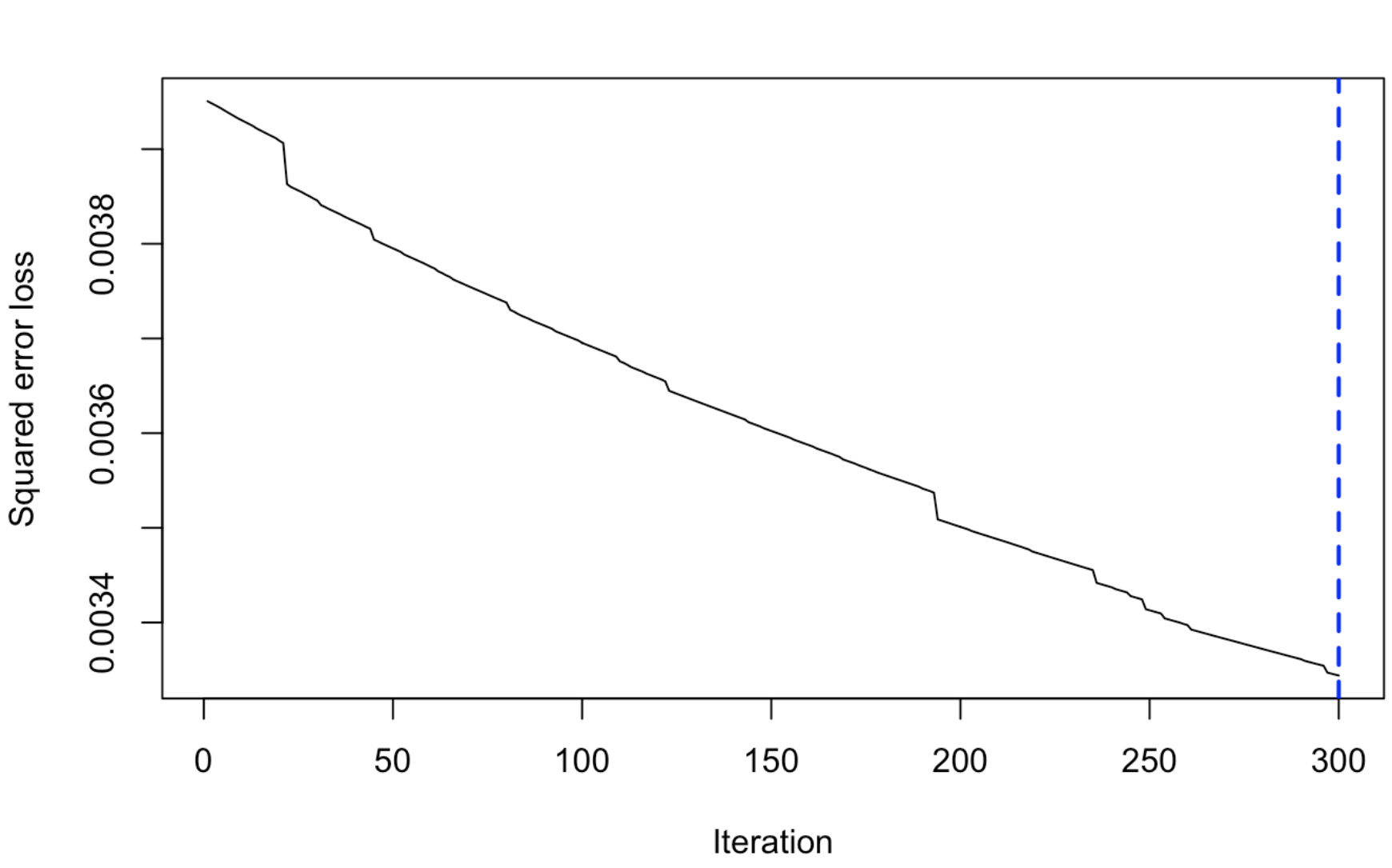

Squared error loss
0.0035
0.0037
0.0039
0.0041
0
50
100
150
200
250
300
Iteration

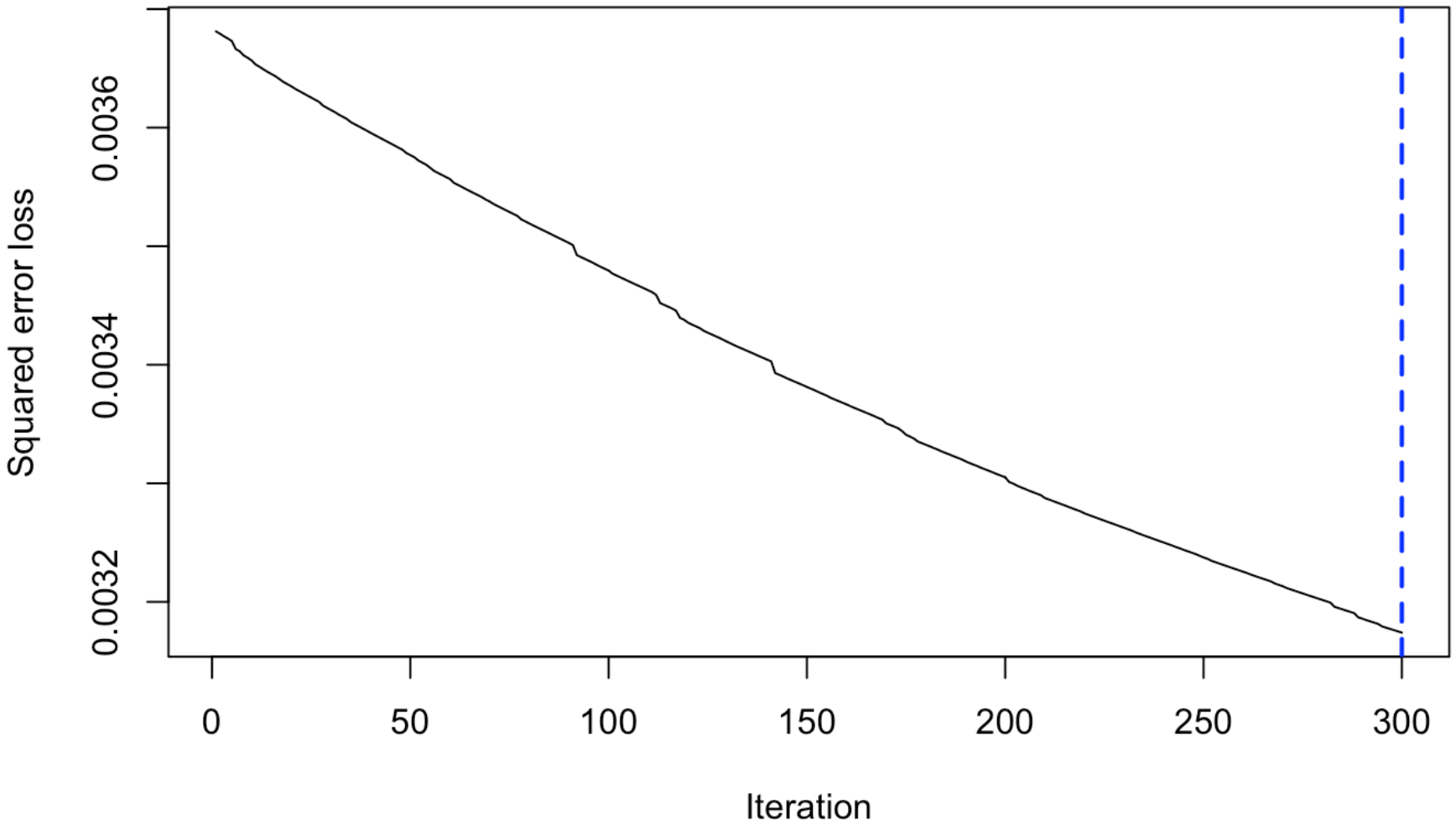
Squared error loss
0.0035
0.0037
0.0039
0.0041
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration

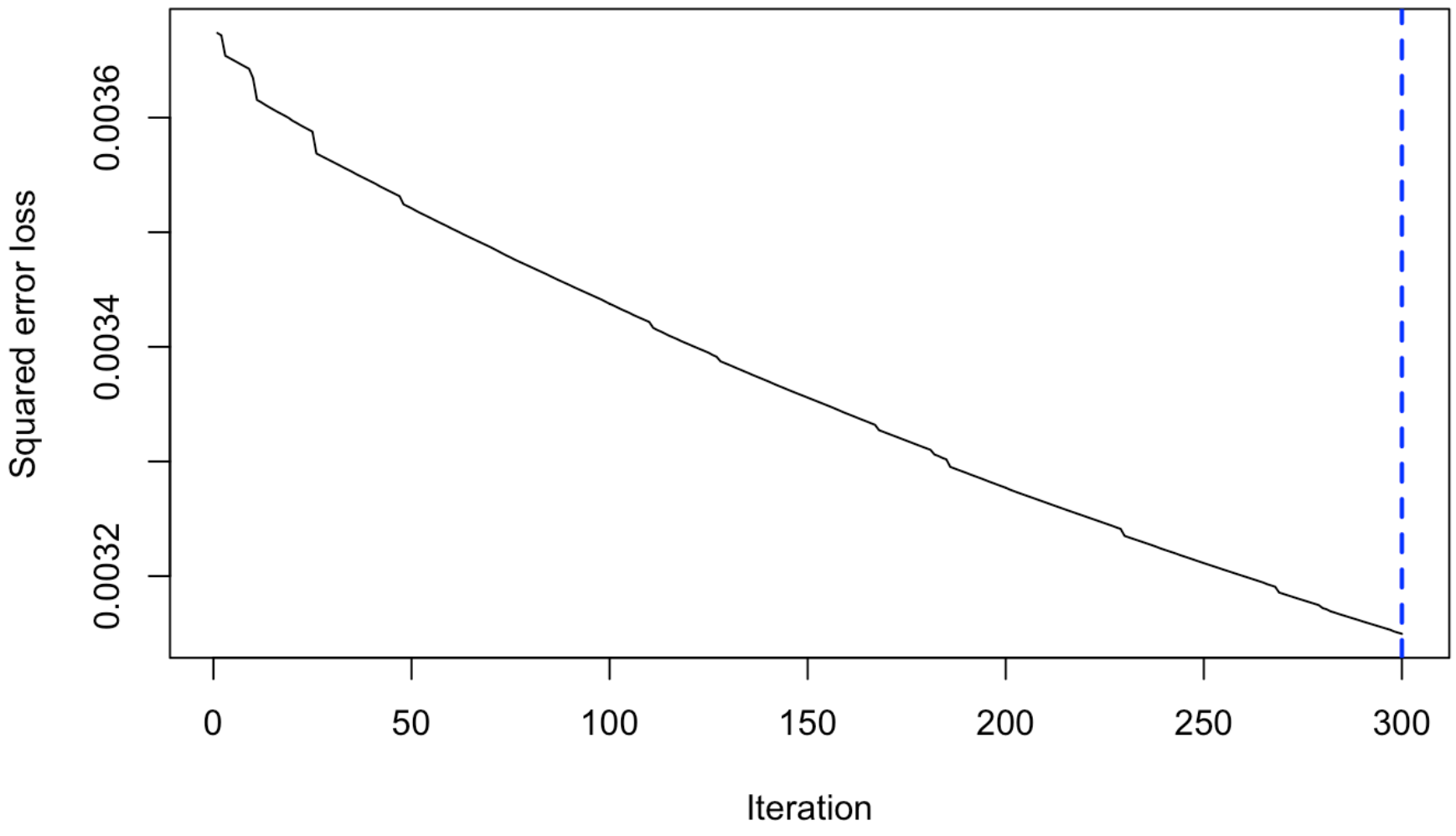

Squared error loss
0.0033
0.0035
0.0037
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0033
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

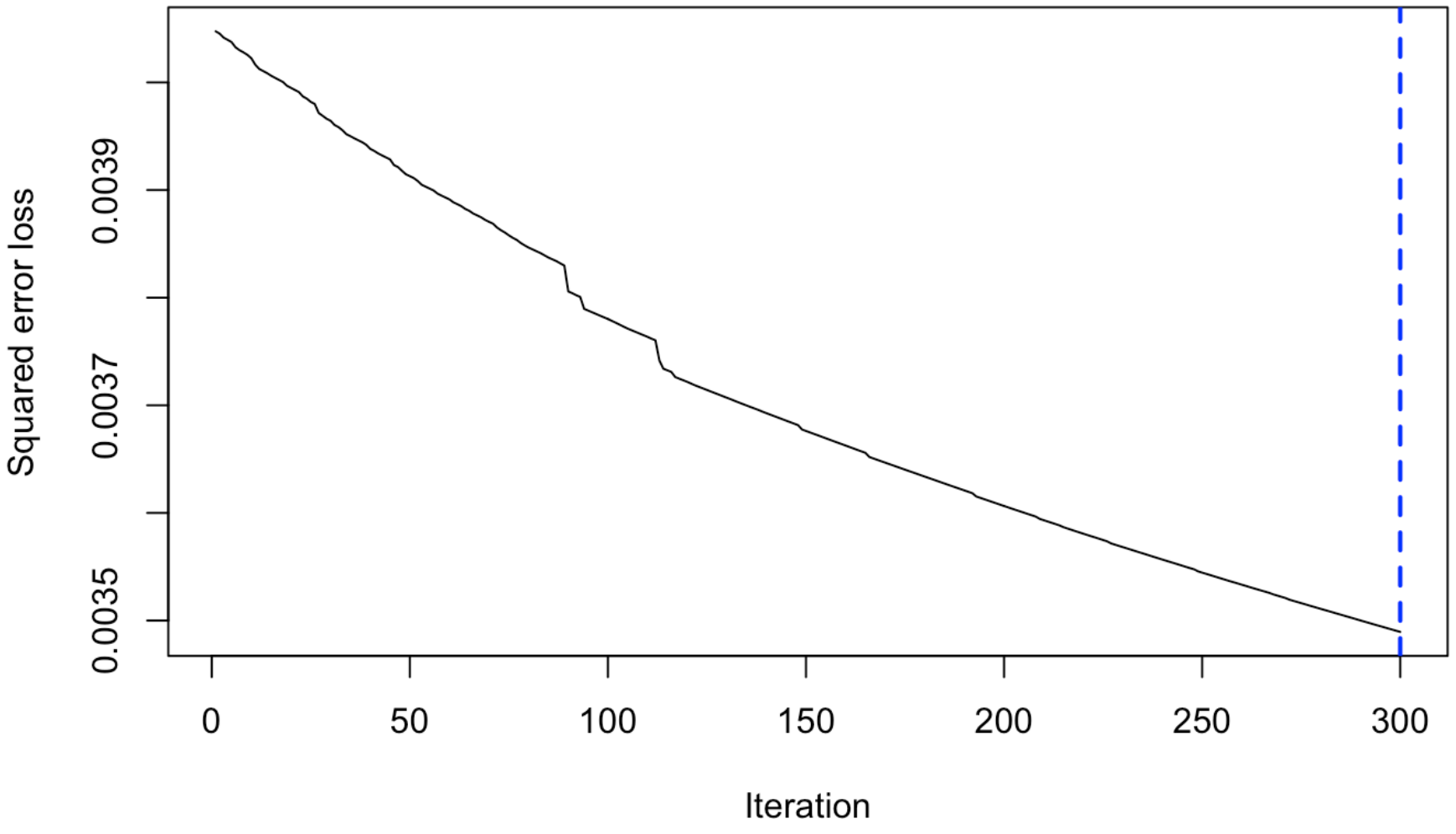

Squared error loss
0.0034
0.0036
0.0038
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

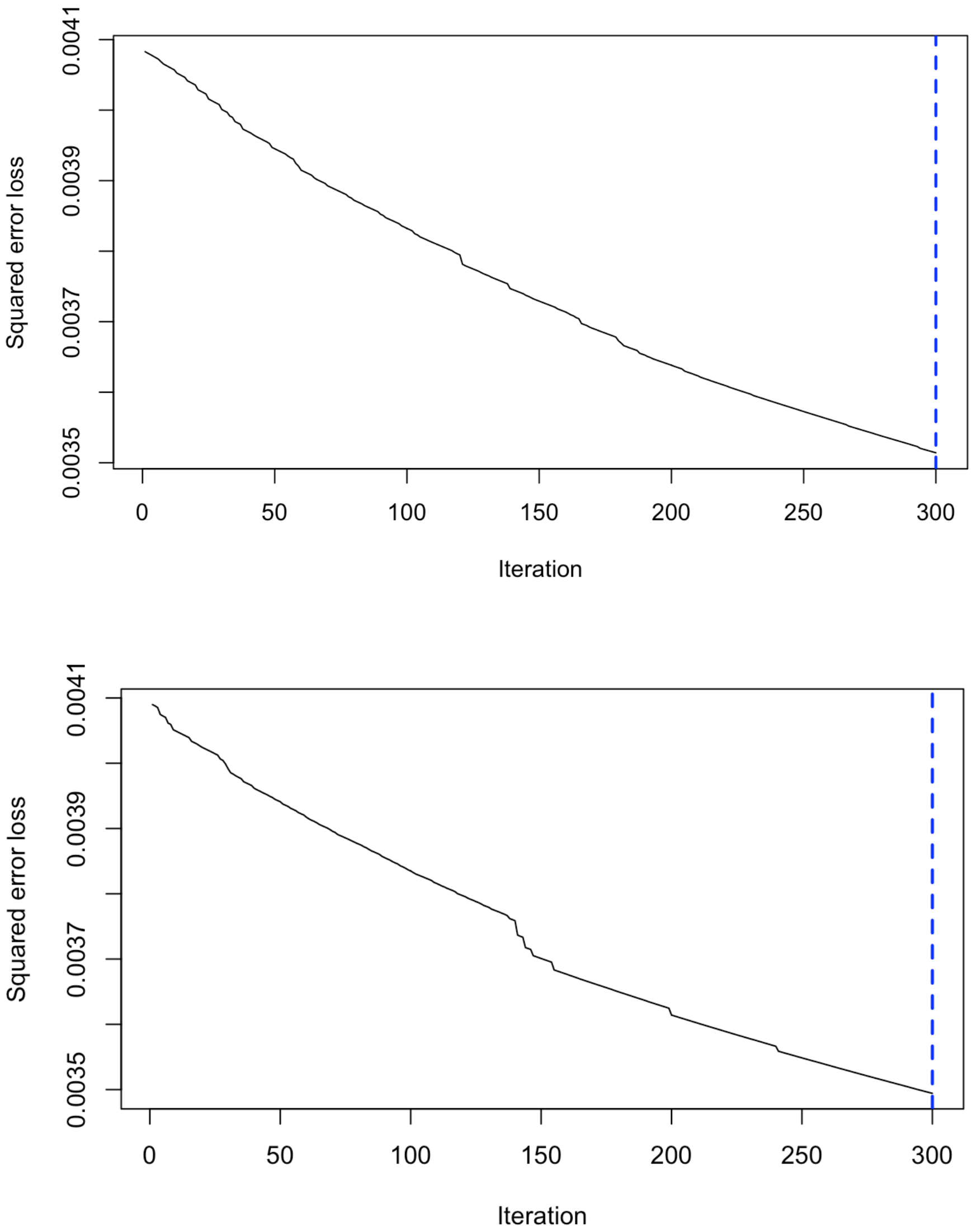
Squared error loss
0.0031
0.0033
0.0035
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration

Squared error loss
0.0032
0.0034
0.0036
0
50
100
150
200
250
300
Iteration


Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration

Squared error loss
0.0035
0.0037
0.0039
0
50
100
150
200
250
300
Iteration



Visualize cross-validation results.

Hide

Hide

```
print(Sys.time())
```

```
[1] "2018-11-07 16:19:27 EST"
```

- Choose the “best”" parameter value

Hide

Hide

```
PSNR_gbm
```

```
[1] 24.79227
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

Hide

Hide

```
tm_train
```

```
[1]      0.0      0.0 144839.4
```

Model 2: XGBoost

Model selection with cross-validation (XGBoost)

- Do model selection by choosing among different values of training model parameters, that is, the interaction depth for GBM in this example.

Hide

Hide

```
model_values_xgb
```

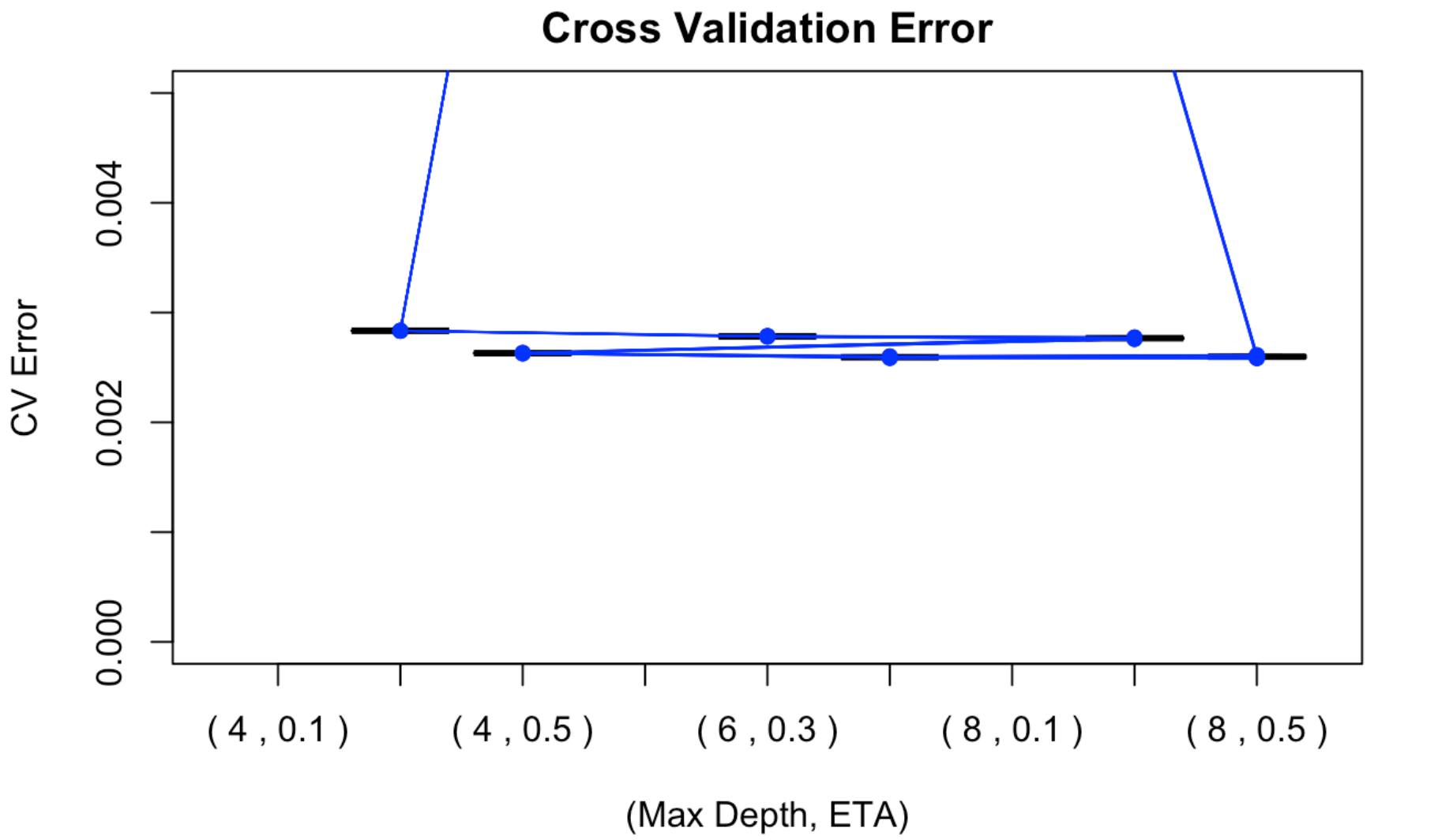
```
Error: object 'model_values_xgb' not found
```

Visualize cross-validation results.

Hide

Hide

```
if(run.cv){
  load("../output/err_cv_xgb.RData")
  mv <- factor(paste("(", model_values_xgb$max_depth,",", model_values_xgb$eta, ")"))
  plot(err_cv_xgb[,1]~mv, xlab="(Max Depth, ETA)", ylab="CV Error",
       main="Cross Validation Error", type="n", ylim=c(0, 0.005))
  points(err_cv_xgb[,1]~mv, col="blue", pch=16)
  lines(err_cv_xgb[,1]~mv, col="blue")
  #arrows(mv,err_cv[,1]-err_cv[,2], mv, err_cv[,1]+err_cv[,2],
        #length=0.1, angle=90, code=3)
}
```



Hide

Hide

```
err_cv_xgb
```

```
             [,1]         [,2]
 [1,] 0.033303251 8.107078e-06
 [2,] 0.033219673 4.185723e-05
 [3,] 0.033194857 6.824574e-06
 [4,] 0.002833976 2.340732e-05
 [5,] 0.002787322 7.660535e-06
 [6,] 0.002773139 9.905559e-06
 [7,] 0.002631797 4.754808e-06
```

```
 [8,] 0.002599917 4.895834e-06
 [9,] 0.002613237 5.898572e-06
[10,] 0.033303418 2.794476e-05
[11,] 0.033216787 1.210148e-05
[12,] 0.033183338 2.232774e-05
[13,] 0.002834271 1.349588e-05
[14,] 0.002781384 6.816008e-06
[15,] 0.002760915 1.064157e-05
[16,] 0.002628875 1.078418e-05
[17,] 0.002587270 1.311060e-05
[18,] 0.002587784 1.274702e-05
[19,] 0.033303832 9.593512e-06
[20,] 0.033218024 2.096164e-05
[21,] 0.033190952 3.597209e-05
[22,] 0.002834502 9.565497e-06
[23,] 0.002784889 1.398329e-05
[24,] 0.002773301 2.387940e-05
[25,] 0.002632815 4.699166e-07
[26,] 0.002601706 9.336253e-06
[27,] 0.002610870 8.171010e-06
[28,] 0.033303596 1.377169e-05
[29,] 0.033217591 3.186970e-05
[30,] 0.033182582 9.295658e-06
[31,] 0.002833533 1.308025e-05
[32,] 0.002780992 3.089930e-06
[33,] 0.002759476 1.134430e-05
[34,] 0.002627532 3.631094e-06
[35,] 0.002587880 7.875359e-06
[36,] 0.002584800 1.392168e-05
[37,] 0.033303893 2.609477e-05
[38,] 0.033219656 5.450917e-06
[39,] 0.033191283 4.016577e-05
[40,] 0.002834413 1.127132e-05
[41,] 0.002786090 5.113980e-06
[42,] 0.002772465 4.718097e-06
[43,] 0.002633315 1.149791e-05
[44,] 0.002599017 9.510827e-06
[45,] 0.002609084 2.153755e-05
[46,] 0.033303488 3.405766e-05
[47,] 0.033217994 1.880239e-05
[48,] 0.033181918 1.412828e-05
[49,] 0.002833863 1.812073e-06
[50,] 0.002781721 7.135867e-06
[51,] 0.002759153 9.123412e-06
[52,] 0.002628601 9.480218e-06
[53,] 0.002586135 9.885180e-06
[54,] 0.002584225 1.278334e-05
```

- Choose the “best”" parameter value

Hide

Hide

```
model_best_xgb=model_values_xgb[1]
if(run.cv){
  model_best_xgb <- model_values_xgb[which.min(err_cv_xgb[,1]),]
}
par_best_xgb <- list(max_depth=model_best_xgb$max_depth, eta=model_best_xgb$eta, subs
ample = model_best_xgb$subsample, min_child_weight = model_best_xgb$min_child_weight)
#par_best_xgb <- list()
PSNR_xgb <- 20*log10(1) - 10 * log10(err_cv_xgb[which.min(err_cv_xgb[,1]),1])
PSNR_xgb
```

```
[1] 25.8767
```

- Train the model with the entire training set using the selected model (model parameter) via cross-validation.

Hide

Hide

```
source("../lib/train.xgboost.R")
tm_train_xgb=NA
tm_train_xgb <- system.time(fit_train_xgb <- train.xgboost(feat_train, label_train, p
ar_best_xgb))
```

```
Timing stopped at: 55.61 2.484 59.13
```

Step 5: Super-resolution for test images

Feed the final training model with the completely holdout testing data. + `superResolution.R` + Input: a path that points to the folder of low-resolution test images. + Input: a path that points to the folder (empty) of high-resolution test images. + Input: an R object that contains tuned predictors. + Output: construct high-resolution versions for each low-resolution test image.

Hide

Hide

```
tm_test
```

```
   user  system elapsed
127.645  70.105 6045.609
```

Hide

Hide

```
tm_test_xgb
```

```
   user  system elapsed
  1.479   0.908 379.421
```

Summarize Running Time

Prediction performance matters, so does the running times for constructing features and for training the model, especially when the computation resource is limited.

Hide

Hide

```
cat("Time for constructing training features=", tm_feature_train[3], "s \n")
```

```
Time for constructing training features= 101.475 s
```

Hide

Hide

```
#cat("Time for constructing testing features=", tm_feature_test[1], "s \n")
cat("Time for training GBM model=", tm_train[3], "s \n")
```

```
Time for training GBM model= 144839.4 s
```

Hide

Hide

```
cat("Time for GBM super-resolution=", tm_test[3], "s \n")
```

```
Time for GBM super-resolution= 6045.609 s
```

Hide

Hide

```
cat("Time for training XGBoost model=", tm_train_xgb[3], "s \n")
```

```
Time for training XGBoost model= 336.628 s
```

Hide

Hide

```
cat("Time for XGBoost super-resolution=", tm_test_xgb[3], "s \n")
```

```
Time for XGBoost super-resolution= 379.421 s
```