Freie Universität Berlin
Chair of Statistics
Location: Berlin
Summer Term 2017
Lecture: Einführung in die Bayes-Statistik
Examiner: Dr. Florian Meinfelder

Program leave-one-out posterior predictive checking in R

Johannes Brinkmann (), jojo-brinkmann@gmx.de
Carlo Michaelis (5043128), carlo.michaelis@gmail.com
Max Reinhardt (579174), max_reinhardt@me.com
Adrian Rolf (), adrian.rolf@gmx.de

Master Statistics
August 21, 2017

Contents

List of Figures

List of Tables

1 Introduction

2 Code

2.1 General code

The Gibbs sampler

```
gibbsSampler <- function(X, Y, R, b = 0, initSigma = 1) {
  # Sample parameters from posterior distribution
  # Prepared for baysean linear regression with ordinary least square approach (OLS)
  #
  # Args:
  #   X: Design matrix
  #   Y: Labels
  #   R: Number of draws (without burn in), R is a scalar in this funcrion
  #   b: Number of burn in draws
  #   initSigma: Initial value of sigma (default 1)
  #
  # Returns:
  #   List containing sampled betas (number depends on design matrix) and sigmas

  # Get number of overall draws (including burn in)
  B <- R + b

  # Size of design matrix
  n <- nrow(X) # Number of data points
  p <- ncol(X) # Number of parameters

  # Variables to store the samples in (initalize sigma with initSigma)
  betas <- matrix(nrow = B, ncol = p)
  sigma <- c(initSigma, rep(NA, B-1))

  # Sampling
  for(i in 1:B){
    # OLS of beta
    V <- solve(t(X)%*%X)      # (X^T X)^-1
    beta_hat <- V%*%t(X)%*%Y # (X^T X)^-1 X^T Y

    # OLS of sigma
    sigma_hat <- t(Y-X%*%beta_hat)%*%(Y-X%*%beta_hat)/(n-p)

    # Sample beta from the full conditional
    betas[i,] <- rmvnorm(1, beta_hat, sigma[i]*V)

    # Sample sigma from the full conditional
    if(i < B) {
      sigma[i+1] <- 1/rgamma(1, (n-p)/2, (n-p)*sigma_hat/2)
    }
  }

  # Remove burn in, if there is some
  if(b != 0) {
    betas <- betas[-(1:b),]
    sigma <- sigma[-(1:b)]
  }
```

```
  return(list(betas = betas, sigma = sigma))
}
```

Cross Validation function

```
crossValidation <- function(X, Y, R, b) {
  # Implementation of leave-one-out cross validation (LOO-CV)
  # Cross validate bayesian linear regression model with OLS
  #
  # Args:
  #   X: Design matrix
  #   Y: Labels
  #   R: Number of draws (without burn in), R can be a vector in this function
  #   b: Number of burn in draws
  #
  # Returns:
  #   List containing estimated parameters and label predictions for every LOO-CV step

  # Size of design matrix
  n <- nrow(X) # Number of data points
  p <- ncol(X) # Number of parameters

  # Get size of R vector (steps = 1, if R is scalar)
  steps <- length(R)

  # Run gibbs sampler to get sampled parameters
  samples <- lapply(1:n, function(i) gibbsSampler(X[-i,], Y[-i], R[steps], b))

  # Initalize lists to store results of estimation and prediction
  Sigma <- list()
  Betas <- list()
  Yhati <- list()

  # Calculate sigma, betas and Yhati for every R step (do it once if R is scalar)
  for(k in 1:steps) {
    Sigma[[k]] <- sapply(samples, function(sample) mean(sample$sigma[1:R[k]]))
    Betas[[k]] <- sapply(samples, function(sample) colMeans(sample$betas[1:R[k],]))
    Yhati[[k]] <- sapply(1:n, function(i) X[i,]%*%Betas[[k]][,i])
  }

  return(list(Sigma = Sigma, Betas = Betas, Yhati = Yhati))
}
```

2.2 Model evaluaion

```
bayesModelEvaluation <- function(models, Y, R, b) {
  # Model evaluation based on mean squared error (MSE) and log posterior
  # predictive density (lppd) using leave-one-out cross validation (LOO-CV)
  #
  # Args:
  #   models: A list of design matrices with different size and types of parameters
  #   Y: Labels
  #   R: Number of draws (without burn in), R can be a vector in this function
  #   b: Number of burn in draws
  #
  # Returns:
  #   List containing MSEs and LPPDs for every model
```

```r
  # Evaluate multiple models and return results from all models
  n <- length(Y)
  k <- length(models)

  # Cross validate every model
  results <- lapply(1:k, function(i) crossValidation(models[[i]], Y, R, b))

  # Calculate Mean Squared Errors
  MSEs <- sapply(results, function(el) {
    return((1/(n-nrow(el$Betas[[1]])))*sum((Y-el$Yhati[[1]])^2))
  })

  # Calculate log posterior predictive density (log likelihood)
  # y ~ N(XB, s^2(X^T X)^-1) = N(Yhat, s^2(X^T X)^-1)
  LPPDs <- sapply(results, function(eva) sum(log(dnorm(Y, eva$Yhati[[1]], eva$Sigma[[1]]))))

  return(list(MSEs = MSEs, LPPDs = LPPDs))
}
```

Run model evaluation with data

```r
# Swiss data
dat <- swiss

# Response variable
Y <- dat$Fertility
n <- nrow(dat)

# Design matrices

models <- list(
  matrix(c(rep(1,n), dat$Education), nrow=n),
  matrix(c(rep(1,n), dat$Agriculture), nrow=n),
  matrix(c(rep(1,n), dat$Examination), nrow=n),
  matrix(c(rep(1,n), dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Examination), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture, dat$Examination), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture, dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Examination, dat$Catholic), nrow=n),
  matrix(c(rep(1,n), dat$Education, dat$Agriculture, dat$Examination ,dat$Catholic), nrow=n)
)

# Run model evaluation
criteria <- bayesModelEvaluation(models, Y, R = 50, b = 10) # R = 500, b = 100
```

```r
# Check proportion
print(criteria$MSEs/(-2*criteria$LPPDs))
```

```
##  [1] 0.1829991 0.2735557 0.1936468 0.2456215 0.1947705 0.1840388 0.1646793
##  [8] 0.1761604 0.1531381 0.1753810 0.1561831
```

```r
# TODO: Als Tabelle ausgeben? Explain why not fulfilled

# Choose optimal modal
optIdx <- which.min(-2*criteria$LPPDs)

# Check if MSE and LPPD would choose the same
```

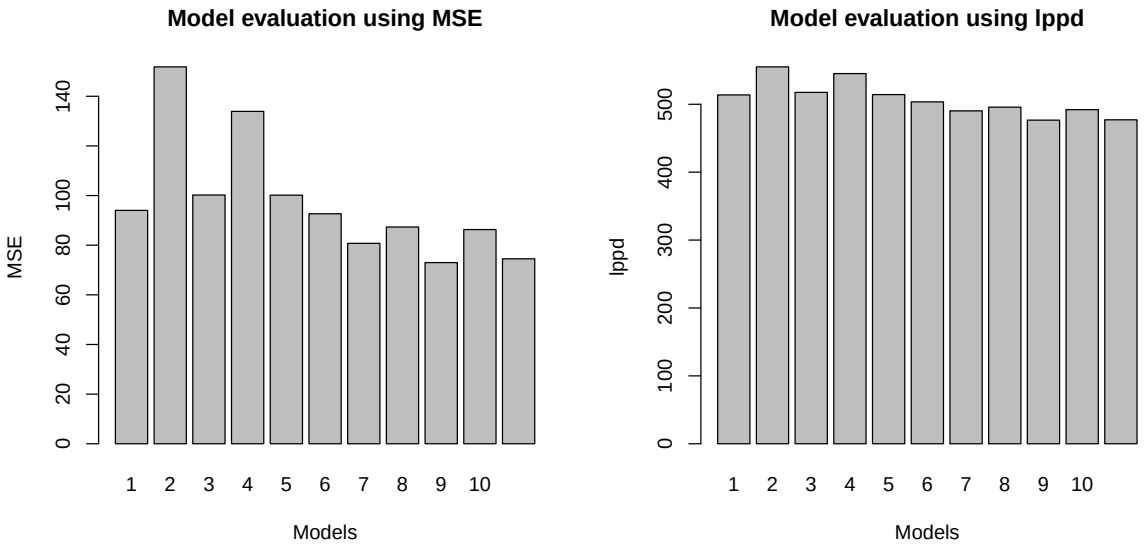

Figure 1: Model evaluation with leave-one-out cross validation (LOO-CV) using mean squared errors (MSE) and log posterior predictive density (lppd) to obtain best model.

```
print(paste(which.min(criteria$MSEs), "=", which.min(-2*criteria$LPPDs)))
```

```
## [1] "9 = 9"
```

```
# TODO den Vergleich einfach nur im Text verwenden?
```

2.3 Cook's Distance

```
cooksDistance <- function(X, Y, R, b) {
  # Calculate cooks distances for optimal model from model evaluaion
  # Two methods are used:
  #   (1) Use LOO-CV approach for bayesian linear regression model with OLS
  #   (2) Use analytic solution for frequentist linear regression model with OLS
  #
  # Args:
  #   X: Design matrix
  #   Y: Labels
  #   R: Number of draws (without burn in), R can be a vector in this function
  #   b: Number of burn in draws
  #
  # Returns:
  #   cooksBayesCV:   Cook's distances calculated by LOO-CV (bayes model),
  #                   where cook's distances are calculated for different sample sizes
  #   cooksAnalytic:  Cook's distances calculated by analytic solution
  #   sample:         The gibbs sample from the bayes linear model (for plotting purpose)

  # R is usually a vector (but don't has to)
  B <- R + b
  steps <- length(R)

  # Prepare projection matrix and number of parameters
  H <- X%*%solve(t(X)%*%X)%*%t(X)
  p <- ncol(X)

  # Run cross validation with vector R
```

```
  cv <- crossValidation(X, Y, R, b)

  # Sample whole model (add B = R + b samples, remove b later)
  sample <- gibbsSampler(X, Y, R[steps] + b)

  # Cook's distance: Frequentist appraoch with analytic solution
  betaHat <- solve(t(X)%*%X)%*%t(X)%*%Y
  YhatLinReg <- X%*%betaHat
  E <- Y-YhatLinReg
  cooksAnalytic <- (E^2/((1/(n-p))*sum(E^2)*p))*(diag(H)/(1-diag(H))^2)

  # Cook's distance: Bayesian appraoch with LOO-VC solution
  cooksBayesCV <- matrix(nrow = n, ncol = steps)
  for(k in 1:steps) {
    # Estimate posterior mean from betas
    betas <- colMeans(sample$betas[b:B[k],])

    # Predict values, using posterior mean
    Yhat <- X%*%betas

    # Calculate cook's distance
    dists <- apply(cv$Betas[[k]], 2, function(betas) {
      return((Yhat - X%*%betas)^2)
    })
    mse <- (1/(n-p))*sum((Y-Yhat)^2)
    cooksBayesCV[,k] <- colSums(dists)/(p*mse)
  }

  return(list(cooksBayesCV = cooksBayesCV, cooksAnalytic = cooksAnalytic, sample = sample))
}
```

Calculate cook's distances

```
# Number of samples
b <- 100
R <- seq(100,500,50) # seq(100,5000,50)

# Get optimal model and calculate projection matrix
cooks <- cooksDistance(models[[optIdx]], Y, R, b)
```

Traces from bayes regression sampling

Posterior densities from bayes regression sampling

Apply cook's distance measure

```
# Initialize matrices
steps <- length(R)
matBayesD <- cooks$cooksBayesCV
matFreqD <- matrix(rep(cooks$cooksAnalytic, steps), ncol=steps)

# Calculate differences between Cook's distance methods
cooksMeasure <- colSums((matBayesD-matFreqD)^2)
```

Cook's distance measure

Cook's distance, comarison between bayes and frequentist

3 References

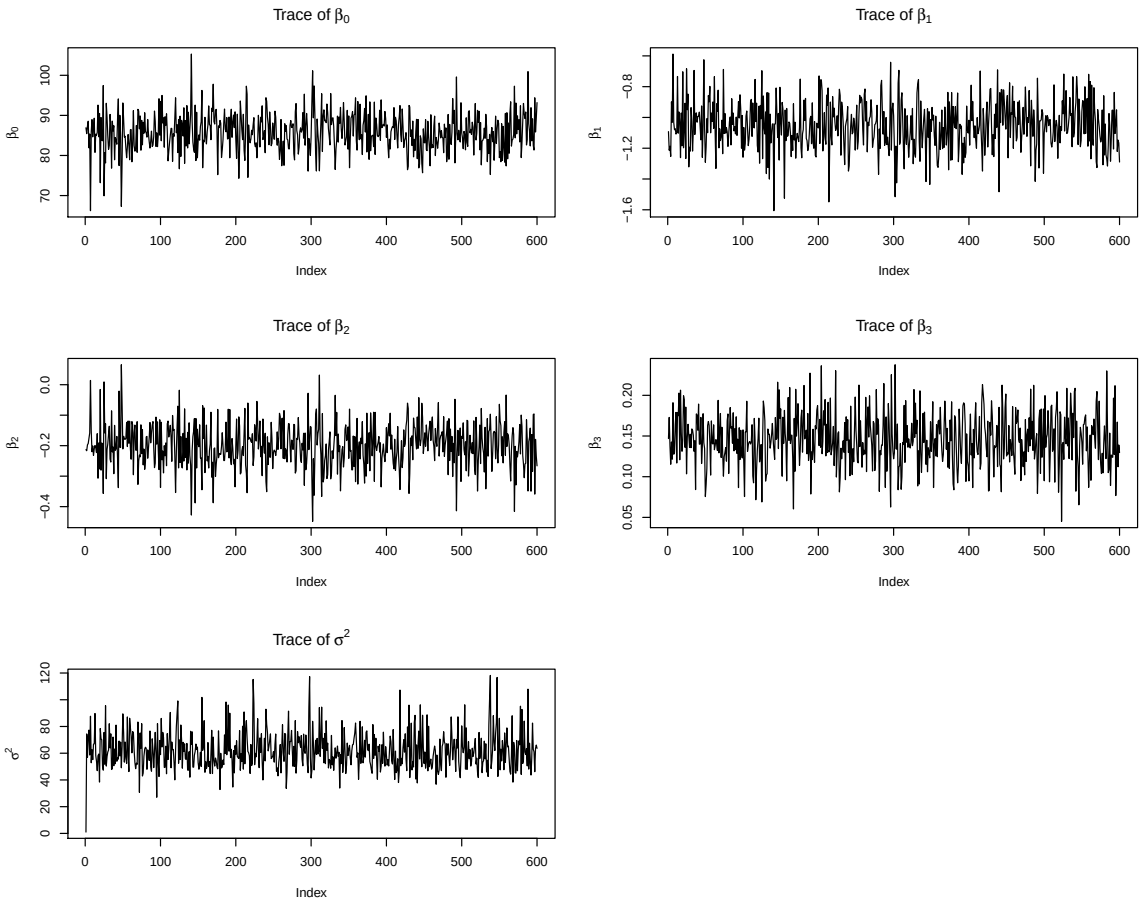

Figure 2: Traces of sampled parameters from posterior distributions regarding bayesian linear regression with ordinary least squares.

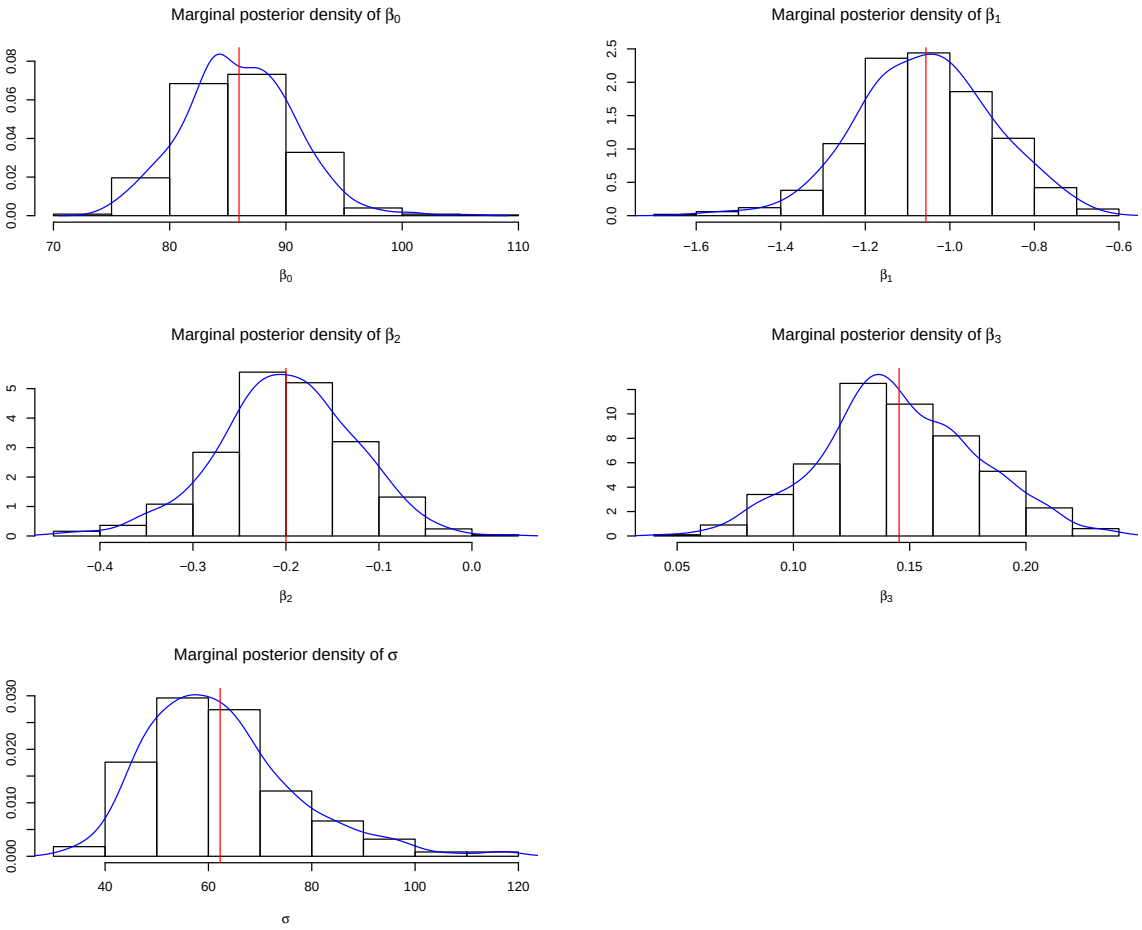

Figure 3: Posterior densities, histograms and posterior means (red vertical line) of sampled parameters regarding bayesian linear regression with ordinary least squares

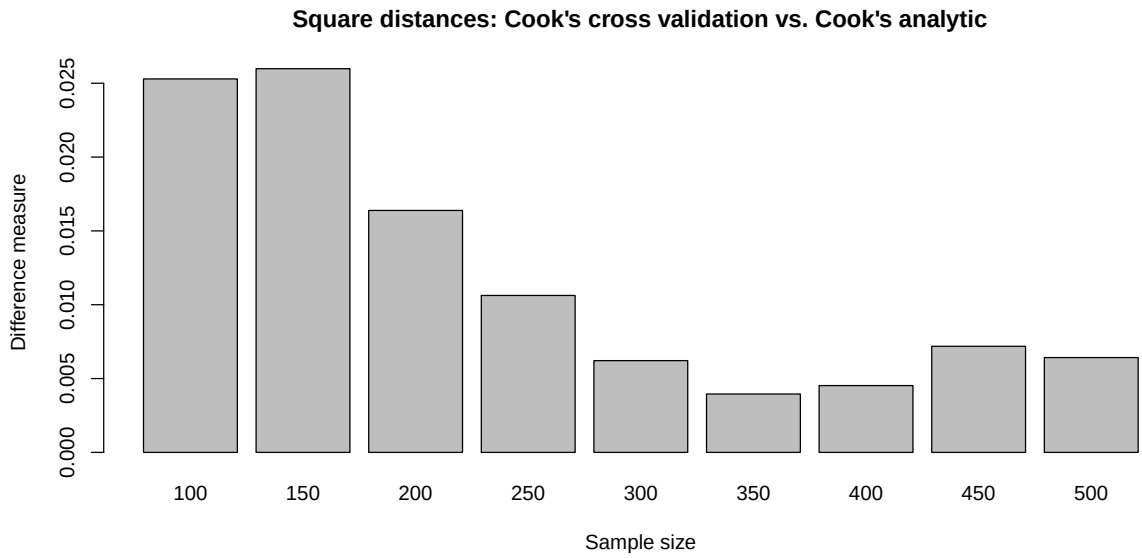

Figure 4: Cook's distance measure to quantify approximation from cross validated cook's distances of analytic cook's distances. The approximate solution with LOO-CV comes closer to the analytic solution, the more samples are drawn.

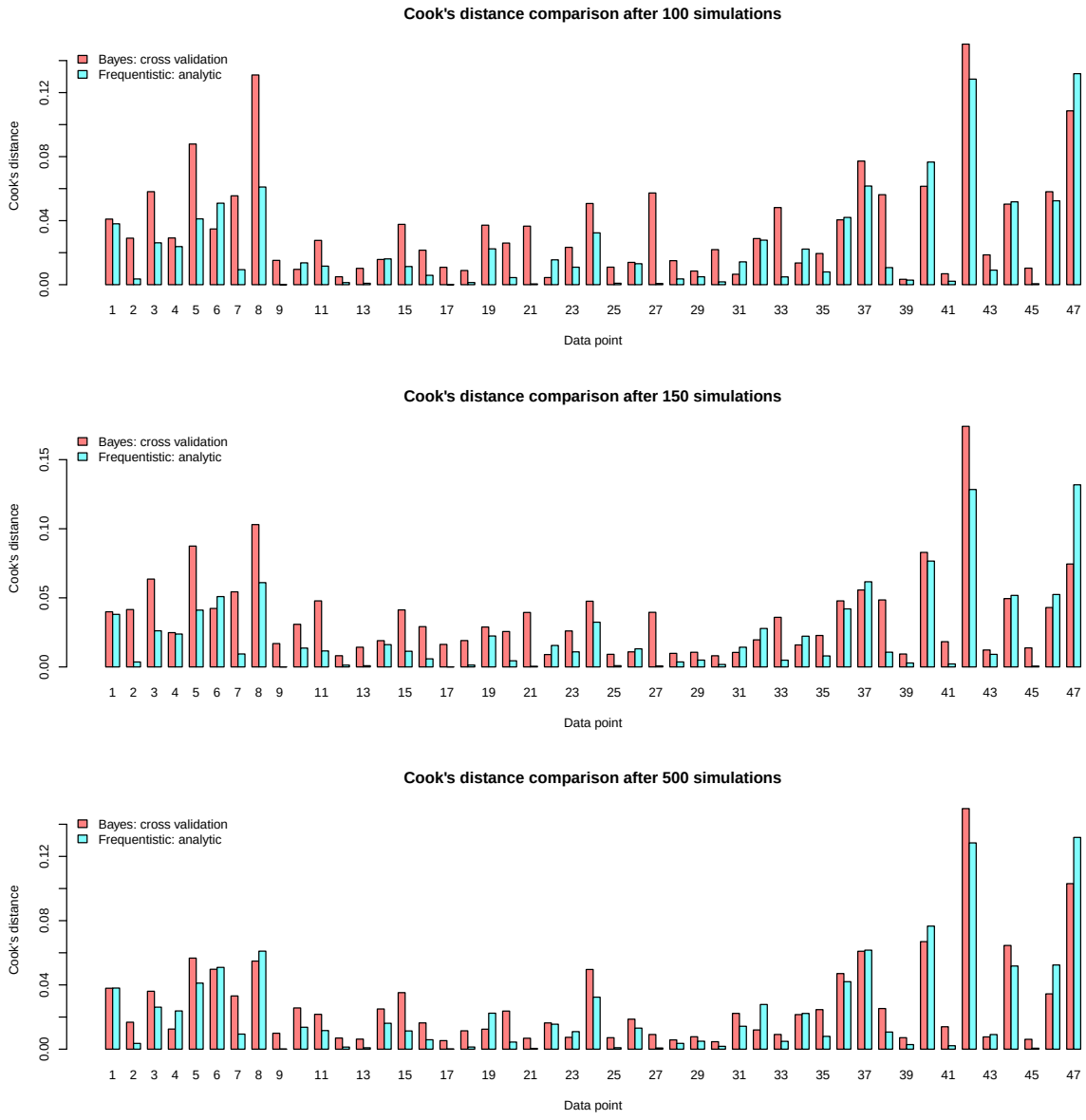

Figure 5: Pointwise cook's distance, calculated with three different sample sizes, compared to analytic cook's distance from frequentist linear regression model using least squares.