Package 'Homework3'

January 1, 2014

Type Package

Title Homework 3 (bayesian computing) for Biostat778

Version 1.0

Date 2014-01-01

Author Stephen Cristiano

Maintainer <scristia@jhsph.edu>

Description Rejection sampling, importance sampling and hybrid gibbs

License GPL

R topics documented:

`postmean` *importance sampling*

Description

does importance sampling

Author(s)

Stephen Cristiano

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (beta, lower, upper)
{
    importance <- function(beta, sigma) {
        n <- length(beta)
        hnorm <- function(N, sigma) {
            replicate(N, qnorm(0.5 * (1 + runif(1)), 0, sqrt(pi/2) *
                sigma))
        }
        ldhnorm <- function(x, sigma) {
            d <- ifelse(x < 0, -Inf, log(2) + dnorm(x, 0, sqrt(pi/2) *
                sigma, log = TRUE))
            return(d)
        }
        lw <- function(x) beta - ldhnorm(beta, sigma)
        U <- hnorm(1000, sigma)
        lps <- lw(U)
        lps <- lps - max(lps)
        I <- mean(exp(lps) * U)/mean(exp(lps))
        return(I)
    }
    sigma <- seq(lower, upper, length = 50)
    postmeans <- sapply(sigma, function(x) importance(beta, x))
    plot(sigma, postmeans, xlab = "Sigma", ylab = "Posterior mean estimate",
        type = "l", main = "Importance sampling")
  }
```

`postpollution` *Hierarchical gibbs sampling with metropolis step*

Description

hybrid gibbs

Author(s)

Stephen Cristiano

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, x, g, N = 10000, burn = 1000)
{
    tn <- max(g)
    lpost.beta <- function(y, x, alpha, beta, mu, tau2) {
```

```
        mu.i <- exp(alpha + beta * x)
        -sum(mu.i) + sum(y * log(mu.i)) - 1/(2 * tau2) * (beta -
            mu)^2
    }
    lpost.alpha <- function(y, x, alpha, beta, sigma2) {
        mu.i <- exp(alpha + beta * x)
        -sum(mu.i) + sum(y * log(mu.i)) - 1/(2 * sigma2) * alpha^2
    }
    a <- b <- c <- d <- 1
    A <- 1
    beta <- rep(0, tn)
    alpha <- rep(0, tn)
    mu <- 0
    tau2 <- sigma2 <- 1
    MU <- TAU <- SIGMA <- rep(NA, N - burn)
    delta = 1.5
    for (i in 1:N) {
        for (j in 1:tn) {
            x.j <- x[g == j]
            y.j <- y[g == j]
            mu.j <- exp(alpha[j] + beta[j] * x.j)
            alpha.star <- rnorm(1, alpha[j], sqrt(delta))
            log.a <- lpost.alpha(y.j, x.j, alpha.star, beta[j],
                sigma2)
            -lpost.alpha(y.j, x.j, alpha[j], beta[j], sigma2)
            if (log(runif(1)) < log.a) {
                alpha[j] <- alpha.star
            }
            beta.star <- rnorm(1, beta[j], sqrt(delta))
            log.b <- lpost.beta(y.j, x.j, alpha[j], beta.star,
                mu, tau2)
            -lpost.beta(y.j, x.j, alpha[j], beta[j], mu, tau2)
            if (log(runif(1)) < log.b) {
                beta[j] <- beta.star
            }
        }
        beta.h <- mean(beta)
        alpha.h <- mean(alpha)
        mu <- rnorm(1, tn * beta.h/(1/A + tn), sqrt(1/(1/A +
            tn)))
        tau2 <- 1/rgamma(1, a + tn/2, b + sum((beta - beta.h)^2)/2 +
            tn/(A * (1/A + tn)) * beta.h^2/2)
        sigma2 <- 1/rgamma(1, c + tn/2, d + sum(alpha^2)/2)
        if (i > burn) {
            MU[i - burn] <- mu
            TAU[i - burn] <- tau2
            SIGMA[i - burn] <- sigma2
        }
    }
    return(list(MU = MU, TAU = TAU, SIGMA = SIGMA))
}
```

`postsample` *Rejection sampling*

Description

Rejection sampling

Author(s)

Stephen Cristiano

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (y, N, sigma)
{
    hnorm <- function(N, sigma) {
        replicate(N, qnorm(0.5 * (1 + runif(1)), 0, sqrt(pi/2)/sigma))
    }
    if (missing(y))
        y <- c(20.100306, 2.272066, 3.796734, 2.265275, 3.480183)
    if (missing(sigma))
        sigma = 0.5
    if (missing(N))
        N = 1000
    mle <- 1/mean(y)
    lik.max <- prod(pexp(y, mle))
    post <- NULL
    while (length(post) < N) {
        u <- runif(1)
        beta <- hnorm(1, sigma)
        lik <- prod(pexp(y, beta))
        if (u <= lik/lik.max)
            post <- append(post, beta)
    }
    return(post)
  }
```

Index