Assignment 8

Sam Olson

Problem Description

Consider a problem of conducting a Bayesian analysis with a one-sample gamma model. Assume that random variables $Y_1, \ldots, Y_n$ are independent and identically distributed with common probability density function (for $\alpha > 0$ and $\beta > 0$):

$$f(y \mid \alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} \exp(-\beta y), \quad y > 0.$$

Suppose that we will assign a joint prior to α and β in product form, with particular values of $A > 0$, $\gamma_0 > 0$, and $\lambda_0 > 0$:

$$\pi_\alpha(\alpha) = \frac{1}{A} I(0 < \alpha < A), \quad \pi_\beta(\beta) = \frac{\lambda_0^{\gamma_0}}{\Gamma(\gamma_0)} \beta^{\gamma_0 - 1} e^{-\lambda_0 \beta}, \quad \beta > 0.$$

Recall that in the analysis of an actual data set, A, γ_0 and λ_0 will be given specific numerical values. Since this is a simulated example and we have no actual prior information, use the following hyperparameters:

$$A = 20, \quad \gamma_0 = 0.5, \quad \lambda_0 = 0.1.$$

This gives prior expectation of 5.0 and prior variance of 50. The prior does focus probability on smaller values, but still has $Pr(\beta > 10) = 0.16$.

```
gammaDat <- read.table("C:/Users/samue/OneDrive/Desktop/Iowa_State_PS/STAT 5200/PS/PS8/gammadat_bayes.t
```

1.

Consider using a Metropolis–Hastings algorithm with independent random-walk proposals for α and β. Suppose that our current values are (α_m, β_m), and that the proposal (α^*, β^*) has been generated from

$$q(\alpha, \beta \mid \alpha_m, \beta_m)$$

which is the product of independent random walks.

Identify the appropriate acceptance probability for the jump proposal (α^*, β^*).

Answer

2.

On the course web page is a data set called `gammadat_bayes.txt`.

Program a Metropolis–Hastings algorithm and simulate 50,000 values from the joint posterior of α, β, and $\mu = \alpha/\beta$.

Provide (with supporting evidence if appropriate):

- Information on how you selected a burn-in period. *NOTE: I do not expect you to compute Gelman-Rubin scale reduction factors for this assignment.*
- Information on how you tuned the algorithm for acceptance rate, including the random-walk variances and the final acceptance rate.
- Summaries of the marginal posterior distributions of α and β and $\mu = \alpha/\beta$, including histograms and five-number summaries, 95% central credible intervals, and correlation between α and β in the Markov chain.

Answer

3.

Using both the 75th percentile and the range as data characteristics of potential interest, compute posterior predictive p-values from 10,000 posterior predictive datasets.

Answer

4.

Now consider the use of a Gibbs Sampling algorithm to simulate from the joint posterior of α and β and μ. Derive full conditional posterior densities for α and β Using these distributions, program a Gibbs Sampling algorithm and simulate 50,000 values from the joint posterior. Provide (with supporting evidence if appropriate),

- information on how you selected a burn-in period. Again, there is no need to compute Gelman-Rubin scale reduction factors for this assignment.
- summaries of the marginal posterior distributions of α and β, including histograms and five-number summaries, 95% central credible intervals, and correlation between α and β in the Markov chain.

Answer

5.

Using both the 75th percentile and the range as data characteristics of potential interest, compute posterior predictive p-values from 10,000 simulated posterior predictive data sets.

Answer

6.

Compare your results from the use of Metropolis-Hastings and Gibbs Sampling.

Answer

7.

On this particular assignment, attach your R code for functions you programmed to do the necessary computations as an APPENDIX – not part of the body of your answer.

Metropolis

```
metropforgamma <- function(dat, start, priorpars, jumpvars, B, M){
# Metropolis for a one-sample Gamma(shape = alpha, rate = beta) model
# with product prior: alpha ~ Uniform(0, A), beta ~ Gamma(gamma0, lambda0)
#
# dat        : vector of observed positive data (y_i > 0)
# start      : c(alpha0, beta0)  -- starting values for (alpha, beta)
# priorpars  : c(gamma0, lambda0, A)
#              - gamma0, lambda0 are shape/rate of prior on beta
#              - A is the upper bound for alpha's Uniform(0, A) prior
# jumpvars   : c(valpha, vbeta)  -- proposal variances for random-walk jumps
# B          : burn-in iterations
# M          : number of kept Monte Carlo draws
#
# Notes on parameterization/statistics:
# - Likelihood: Y_i ~ Gamma(alpha, beta) with density
#       f(y | alpha, beta) = beta^alpha / Gamma(alpha) * y^(alpha-1) * exp(-beta*y)
#   The log-likelihood is computed in a numerically stable way via sums.
# - Prior on alpha: Uniform(0, A). Inside (0, A) its log prior is constant (0),
#   outside the interval, log prior is -Inf.
# - Prior on beta: Gamma(gamma0, lambda0) with 'rate' parameterization.
# - Proposals: independent Gaussian random walks on alpha and beta, consistent
#   with the reference style. We clip invalid proposals by reverting to current
#   values, mirroring the reference behavior for sig2.
#
  calpha <- start[1]; cbeta <- start[2]
  gamma0 <- priorpars[1]; lambda0 <- priorpars[2]; A <- priorpars[3]
  valpha <- jumpvars[1]; vbeta <- jumpvars[2]

  alphas <- NULL; betas <- NULL; mus <- NULL
  acceptind <- 0
  cnt <- 0

  # Precompute sufficient statistics for the Gamma likelihood
  n <- length(dat)
  sumlogy <- sum(log(dat))
  sumy <- sum(dat)

  repeat{
    cnt <- cnt + 1
    alphastar <- proposealpha(calpha, valpha, A)
    betastar  <- proposebeta(cbeta, vbeta)

    # log-likelihood (current and proposed)
    #   log f(alpha, beta | y) = n * (alpha * log(beta) - log(Gamma(alpha))) +
    #                            (alpha - 1) * sum(log(y)) - beta * sum(y)
```

```
    lfcur  <- n * (calpha * log(cbeta) - lgamma(calpha)) + (calpha - 1) * sumlogy - cbeta * sumy
    lfstar <- n * (alphastar * log(betastar) - lgamma(alphastar)) + (alphastar - 1) * sumlogy - betasta

    # log-prior for alpha: Uniform(0, A)
    #   log pi(alpha) = 0 for alpha in (0, A), -Inf otherwise
    lpi_alpha_cur  <- if(calpha > 0 && calpha < A) 0 else -Inf
    lpi_alpha_star <- if(alphastar > 0 && alphastar < A) 0 else -Inf

    # log-prior for beta: Gamma(gamma0, lambda0), rate parameterization
    #   log pi(beta) = gamma0 * log(lambda0) - log(Gamma(gamma0))
    #                  + (gamma0 - 1) * log(beta) - lambda0 * beta
    lpi_beta_cur  <- gamma0 * log(lambda0) - lgamma(gamma0) + (gamma0 - 1) * log(cbeta)  - lambda0 * cb
    lpi_beta_star <- gamma0 * log(lambda0) - lgamma(gamma0) + (gamma0 - 1) * log(betastar) - lambda0 * 

    lpicur  <- lpi_alpha_cur + lpi_beta_cur
    lpistar <- lpi_alpha_star + lpi_beta_star

    # Metropolis acceptance (symmetric random-walk proposals)
    astar <- min(exp((lfstar + lpistar) - (lfcur + lpicur)), 1)
    ustar <- runif(1, 0, 1)

    newalpha <- calpha; newbeta <- cbeta
    if(ustar <= astar){
      newalpha <- alphastar; newbeta <- betastar
      acceptind <- acceptind + 1
    }

    if(cnt > B){
      alphas <- c(alphas, newalpha)
      betas  <- c(betas,  newbeta)
      mus    <- c(mus,    newalpha / newbeta)  # Posterior samples of mu = alpha / beta
    }

    calpha <- newalpha; cbeta <- newbeta
    if(cnt == (B + M)) break
  }

  cat("acceptprob:", acceptind / M, fill = TRUE)
  res <- data.frame(alpha = alphas, beta = betas, mu = mus)
  return(res)
}
#-----------------------------------------------------------------
proposealpha <- function(calpha, valpha, A){
# propose jump from random walk for alpha (shape), enforce support (0, A)
# Reference-style: if invalid, revert to current (like proposesig2 in the ref)
  z <- rnorm(1, 0, sqrt(valpha))
  alphastar <- calpha + z
  if(alphastar <= 0 || alphastar >= A) alphastar <- calpha
  return(alphastar)
}
#-----------------------------------------------------------------
proposebeta <- function(cbeta, vbeta){
# propose jump from random walk for beta (rate), enforce positivity
```

```r
# Reference-style: if invalid, revert to current
  z <- rnorm(1, 0, sqrt(vbeta))
  betastar <- cbeta + z
  if(betastar <= 0) betastar <- cbeta
  return(betastar)
}
#-----------------------------------------------------------------
```

Gibbs

```r
gibbsforgamma <- function(dat, start, priorpars, B, M, valpha){
# Gibbs sampler for one-sample Gamma(shape = alpha, rate = beta) model
# with alpha ~ Uniform(0, A), beta ~ Gamma(gamma0, lambda0)
#
# dat         : vector of observed positive data (y_i > 0)
# start       : c(alpha0, beta0)  -- starting values
# priorpars   : c(gamma0, lambda0, A)
# B           : burn-in iterations
# M           : number of kept Monte Carlo draws
# valpha      : proposal variance for MH step on alpha (Metropolis-within-Gibbs)
#
# Notes on full conditionals and conjugacy:
# - Conditional for beta | alpha, y is Gamma(gamma0 + n*alpha, lambda0 + sum(y))
#   (shape/rate parametrization) -- this is conjugate, so we can sample beta directly.
# - Conditional for alpha | beta, y is NOT standard:
#       p(alpha | beta, y) proportional to [beta^(n*alpha) / Gamma(alpha)^n] *
#       (prod(y_i))^(alpha - 1) * I(0 < alpha < A)
#   We use a random-walk MH step for alpha inside the Gibbs loop
#   (Metropolis-within-Gibbs), mirroring the reference Gibbs code structure.
#
  calpha <- start[1]; cbeta <- start[2]
  gamma0 <- priorpars[1]; lambda0 <- priorpars[2]; A <- priorpars[3]

  alphas <- NULL; betas <- NULL; mus <- NULL
  cnt <- 0
  accept_alpha <- 0

  # Precompute sufficient statistics
  n <- length(dat)
  sumlogy <- sum(log(dat))
  sumy <- sum(dat)

  repeat{
    cnt <- cnt + 1

    # 1) Sample beta | alpha, y  (conjugate Gamma)
    #    shape = gamma0 + n*alpha ; rate = lambda0 + sum(y)
    newbeta <- rgamma(1, shape = gamma0 + n * calpha, rate = lambda0 + sumy)

    # 2) Sample alpha | beta, y  (Metropolis step within Gibbs)
    #    target log-density up to constant:
    #       log p(alpha | beta, y) = n*alpha*log(beta) - n*log(Gamma(alpha))
```

```r
    #                                + (alpha - 1)*sum(log(y)), for 0<alpha<A
    astep <- sampalpha_mh(calpha, newbeta, valpha, sumlogy, n, A)
    newalpha <- astep$alpha
    accept_alpha <- accept_alpha + astep$acc

    if(cnt > B){
      alphas <- c(alphas, newalpha)
      betas  <- c(betas,  newbeta)
      mus    <- c(mus,    newalpha / newbeta)
    }
    calpha <- newalpha; cbeta <- newbeta

    if(cnt == (B + M)) break
  }

  cat("alpha_acceptprob (within Gibbs):", accept_alpha / M, fill = TRUE)
  res <- data.frame(alpha = alphas, beta = betas, mu = mus)
  return(res)
}
#-----------------------------------------------------------------
sampalpha_mh <- function(calpha, beta, valpha, sumlogy, n, A){
# One-step random-walk MH update for alpha (shape) given beta and y.
# Returns a list(alpha = ..., acc = 0/1)
#
# target log-density (up to constant in alpha):
#   log f(alpha | beta, y) = n*alpha*log(beta) - n*log(Gamma(alpha))
#                            + (alpha - 1)*sumlogy
# with support 0 < alpha < A; outside support, log-density = -Inf.
#
  z <- rnorm(1, 0, sqrt(valpha))
  alphastar <- calpha + z
  if(alphastar <= 0 || alphastar >= A){
    # As in reference style, invalid proposal -> revert (equivalent to reject)
    return(list(alpha = calpha, acc = 0))
  }

  # log target at current and proposed
  lfcur  <- n * calpha    * log(beta) - n * lgamma(calpha)    + (calpha    - 1) * sumlogy
  lfstar <- n * alphastar * log(beta) - n * lgamma(alphastar) + (alphastar - 1) * sumlogy

  a <- min(exp(lfstar - lfcur), 1)
  u <- runif(1, 0, 1)
  if(u <= a) return(list(alpha = alphastar, acc = 1))
  return(list(alpha = calpha, acc = 0))
}
#-----------------------------------------------------------------
```