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 $Pr(\beta > 10) = 0.16$.

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

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

I believe this question is being asked in the abstract, i.e., not for the specific dataset in question. Under that pretense:

The target distribution for the Metropolis–Hastings algorithm is the joint posterior

$$\pi(\alpha, \beta \mid y) \propto L(y \mid \alpha, \beta)\, \pi_\alpha(\alpha)\, \pi_\beta(\beta)$$

where the likelihood for independent $Y_i \sim \text{Gamma}(\alpha, \beta)$ is

$$L(y \mid \alpha, \beta) = \prod_{i=1}^{n} \frac{\beta^\alpha}{\Gamma(\alpha)} y_i^{\alpha-1} e^{-\beta y_i}$$

If the proposal distribution is a product of independent random walks,

$$q(\alpha^*, \beta^* \mid \alpha_m, \beta_m) = q_\alpha(\alpha^* \mid \alpha_m) q_\beta(\beta^* \mid \beta_m)$$

and each random walk is symmetric, then the proposal densities cancel in the Hastings ratio.

The acceptance probability is therefore

$$a\big((\alpha_m, \beta_m) \to (\alpha^*, \beta^*)\big) = \min\left\{1,\ \frac{\pi(\alpha^*, \beta^* \mid y)}{\pi(\alpha_m, \beta_m \mid y)}\right\} = \min\left\{1,\ \frac{L(y \mid \alpha^*, \beta^*)\, \pi_\alpha(\alpha^*)\, \pi_\beta(\beta^*)}{L(y \mid \alpha_m, \beta_m)\, \pi_\alpha(\alpha_m)\, \pi_\beta(\beta_m)}\right\}$$

Because $\pi_\alpha(\alpha)$ is uniform on $(0, A)$, this implies that any proposal with $\alpha^* \notin (0, A)$ or $\beta^* \leq 0$ is automatically rejected ($a = 0$).

Using sufficient statistics $S_1 = \sum_{i=1}^{n} \log y_i$ and $S_2 = \sum_{i=1}^{n} y_i$, we can write

$$\log r = n\left[\alpha^* \log \beta^* - \log \Gamma(\alpha^*) - \alpha_m \log \beta_m + \log \Gamma(\alpha_m)\right] + (\alpha^* - \alpha_m) S_1 - (\beta^* - \beta_m) S_2 + (\gamma_0 - 1)\left[\log \beta^* - \log \beta_m\right] - \lambda_0(\beta^* - \beta_m)$$

and

$$a = \min\{1,\ \exp(\log r)\}$$

This is the appropriate acceptance probability for the Metropolis–Hastings update of (α, β).

2.

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

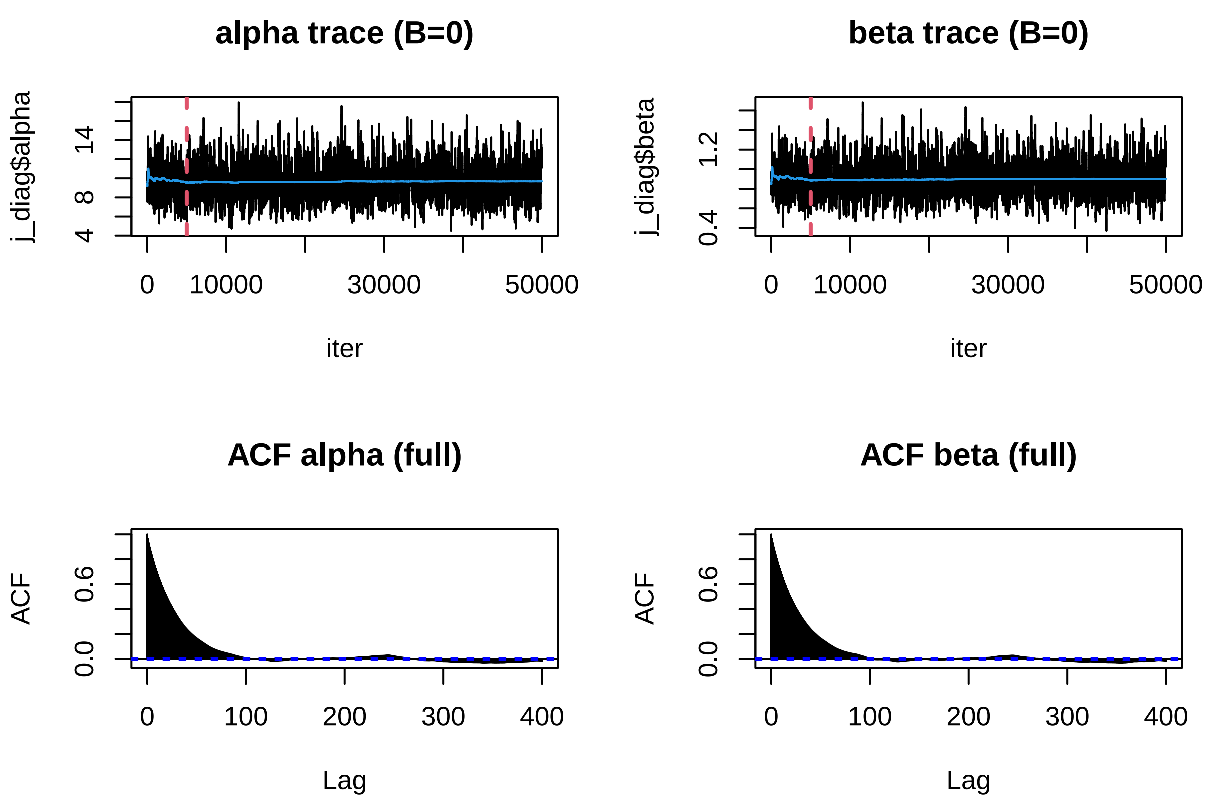
Program a Metropolis–Hastings algorithm and simulate 50,000 values from the joint posterior of α, β, and $\mu = \alpha/\beta$.

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

```
## acceptprob: 0.18012
```



We start with our initial values given in the problem statement. We run a sampling procedure without any tuning, first to determine a suitable Burn-In period.

To that end, we use a "running mean" (blue) to get a sense of when variability in the parameter "stabilizes". We see that by iteration 2,000-3,000, the ACFs decay rapidly; to stay on the conservative side, we double this and set that equal to our Burn-In period, settling on a Burn-In period of 5,000.

We then have some additional tuning.

```
## acceptprob: 0.455125
## acceptprob: 0.29175
## acceptprob: 0.219375
## acceptprob: 0.185375
## acceptprob: 0.134875
```

```
##   scale    valpha       vbeta   accept
## 1   0.5 0.8472632 0.007334346 0.455125
## 2   0.8 2.1689938 0.018775926 0.291750
## 3   1.0 3.3890527 0.029337385 0.219375
## 4   1.2 4.8802360 0.042245834 0.185375
## 5   1.5 7.6253687 0.066009116 0.134875
```

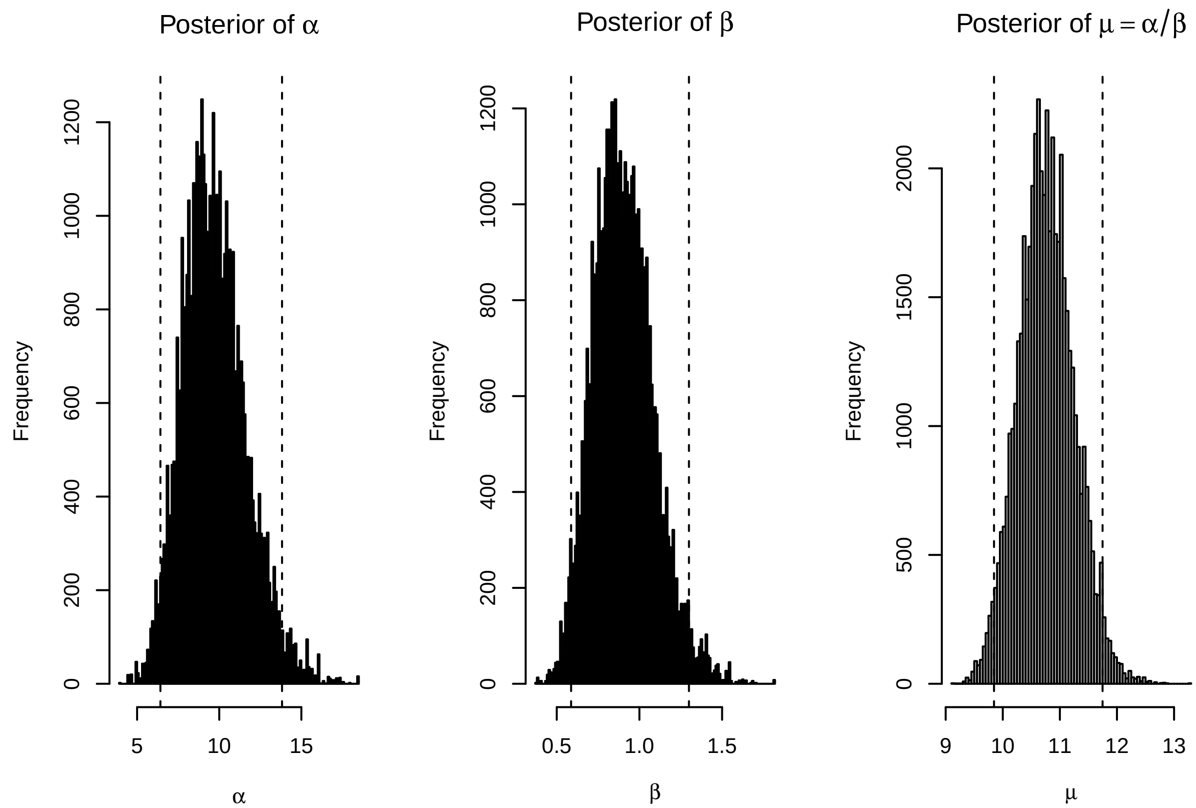
We "tune" by considering a range of parameter values; this involves tuning the random-walk variances to reach a Metropolis-Hastings acceptance rate somewhere in the range of 20-60%, which is a heuristic noted in Chapter 7 notes on Simulation. Ultimately, we decided on using a "scale" of 1.2, corresponding to `jumpvars = jumpvars * 1.2^2`.

This then leads us to run the whole simulation procedure again, this time with the suitable Burn-In Period and tuned parameters.

```
## acceptprob: 0.16006
```

```
## $accept
## [1] 0.16006
##
## $five_num
## $five_num$alpha
## [1]  3.920756  8.367582  9.568963 10.885354 18.469583
##
## $five_num$beta
## [1] 0.3719870 0.7743931 0.8893703 1.0174742 1.8146824
##
## $five_num$mu
## [1]  9.111036 10.434815 10.752619 11.082351 13.257706
##
##
## $ci_95
## $ci_95$alpha
##      2.5%     97.5%
##  6.417408 13.824538
##
## $ci_95$beta
##      2.5%     97.5%
## 0.5868943 1.2996206
##
## $ci_95$mu
##      2.5%     97.5%
```

```
##  9.846528 11.747854
##
##
## $corr_ab
## [1] 0.9753999
```



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

We then do additional posterior predictive checks to validate the results of our simulation.

```
## $ppp_75th
##   p_low    p_up
## 0.56884 0.43116
##
## $ppp_range
##  p_low   p_up
## 0.4469 0.5531
```

Generally, we want posterior predictive p-values that are not too large and not too small (so somewhere in the range of 0.2 to 0.7, or so); the values we observe for the statitics of interest seem suitable.

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

Let $Y_i \overset{\text{iid}}{\sim} \text{Gamma}(\alpha, \beta)$ with density

$f(y \mid \alpha, \beta) = \dfrac{\beta^\alpha}{\Gamma(\alpha)}, y^{\alpha-1} e^{-\beta y}$ for $y > 0$.

Priors: $\alpha \sim \text{Uniform}(0, A)$ and $\beta \sim \text{Gamma}(\gamma_0, \lambda_0)$ in using the rate parametrization.

Denote $S_1 = \sum_{i=1}^n \log y_i$ and $S_2 = \sum_{i=1}^n y_i$.

The joint posterior (up to a constant) is

$$\pi(\alpha, \beta \mid y) \propto \beta^{n\alpha} e^{-\beta S_2} \frac{e^{(\alpha-1)S_1}}{[\Gamma(\alpha)]^n} \beta^{\gamma_0 - 1} e^{-\lambda_0 \beta} \nVdash_{(0,A)}(\alpha)$$

Collecting terms in β gives the kernel

$$\beta^{n\alpha + \gamma_0 - 1} \exp\big(- (\lambda_0 + S_2)\beta \big)$$

so

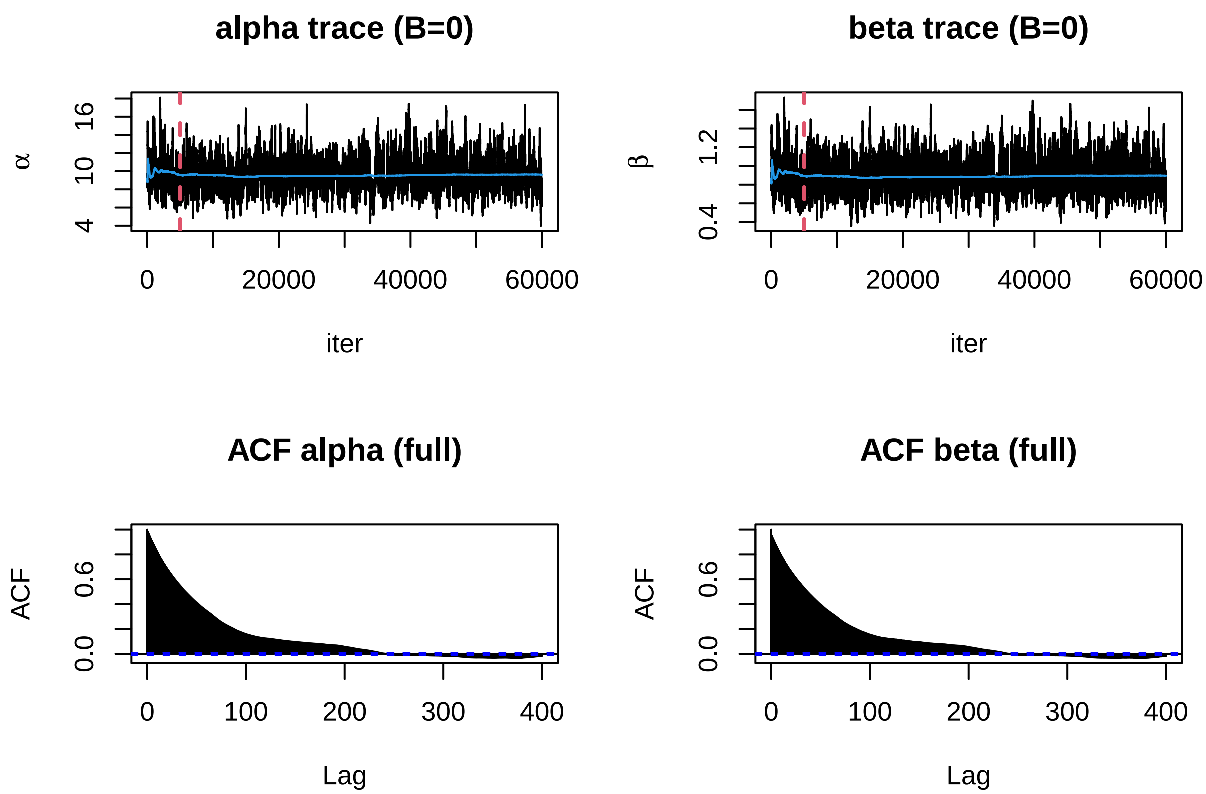
$$\beta \mid \alpha, y \ \sim \ \text{Gamma}\,(\ \gamma_0 + n\alpha,\ \ \lambda_0 + S_2\ )$$

Collecting terms in α yields

$$\pi(\alpha \mid \beta, y) \ \propto \exp\Big(n\alpha \log\beta - n \log\Gamma(\alpha) + (\alpha - 1)S_1 \Big) \mathbf{1}_{(0,A)}(\alpha)$$

which is not a standard family (because of $\log\Gamma(\alpha)$). Therefore α is updated by a random-walk Metropolis step inside the Gibbs sampler (Metropolis-within-Gibbs), with proposals constrained to $(0, A)$.

```
## alpha_acceptprob (within Gibbs): 0.34735
```



We start with our initial values given in the problem statement. We run a sampling procedure without any tuning, first to determine a suitable Burn-In period.

Similar to Question 2, we use a "running mean" (blue) to get a sense of when variability in the parameter "stabilizes". We see that by iteration 2,000-3,000, the ACFs decay rapidly; to stay on the conservative side, we double this and set that equal to our Burn-In period, settling on a Burn-In period of 5,000.

Then, and again, similarly to question 2, we have some additional tuning to do:

```
## alpha_acceptprob (within Gibbs): 0.57275
## alpha_acceptprob (within Gibbs): 0.435875
## alpha_acceptprob (within Gibbs): 0.3685
## alpha_acceptprob (within Gibbs): 0.300625
```

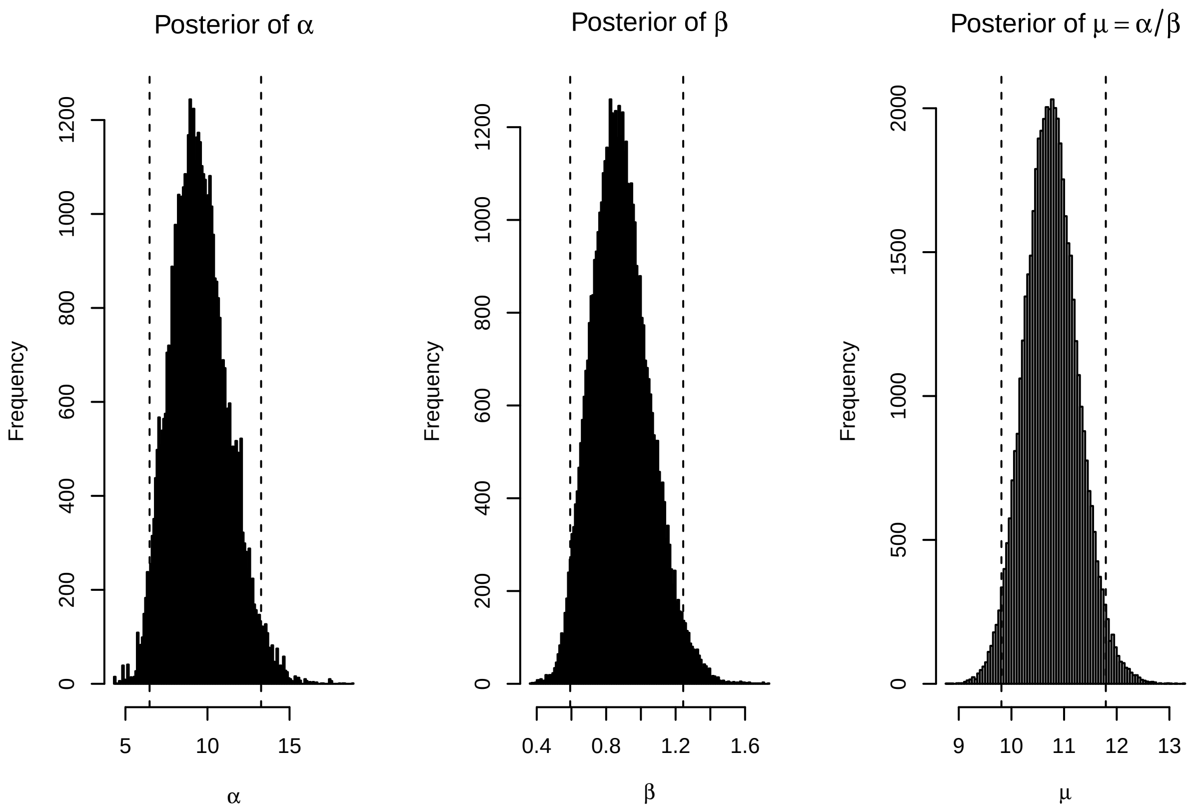
```
##   scale    valpha acc_alpha
## 1   0.7 0.9341077 0.4579322
## 2   1.0 1.9063422 0.3511689
## 3   1.2 2.7451327 0.2966621
## 4   1.5 4.2892699 0.2405301
```

And now with the suitable Burn-In period and tuned parameters, we run our full simulation using the Gibbs method:

```
## alpha_acceptprob (within Gibbs): 0.33142
```

```
## $valpha_final
```

```
## [1] 2.745133
##
## $alpha_accept_final
## [1] NA
##
## $five_num
## $five_num$alpha
## [1]  4.330349  8.280419  9.399169 10.608981 18.892815
##
## $five_num$beta
## [1] 0.3645297 0.7680160 0.8747092 0.9910929 1.7330084
##
## $five_num$mu
## [1]  8.78953 10.41884 10.74781 11.08679 13.27267
##
##
## $ci_95
## $ci_95$alpha
##     2.5%    97.5%
##  6.47112 13.26830
##
## $ci_95$beta
##      2.5%     97.5%
## 0.5926832 1.2438233
##
## $ci_95$mu
##      2.5%     97.5%
##  9.809745 11.792493
##
##
## $corr_ab
## [1] 0.9697005
```



Posterior of α
Frequency
0
200
400
600
800
1000
1200
5
10
15
α
Posterior of β
Frequency
0
200
400
600
800
1000
1200
0.4
0.8
1.2
1.6
β
Posterior of μ = α/β
Frequency
0
500
1000
1500
2000
9
10
11
12
13
μ

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

```
## $observed_statistics
##      q75    range
## 12.93175 14.95000
##
## $ppp_upper_inclusive
##    q75  range
## 0.4307 0.5660
##
## $ppp_lower_inclusive
##    q75  range
## 0.5693 0.4340
##
## $ppp_two_sided_median
##    q75  range
## 0.8636 0.8687
##
## $notes
## [1] "yrep_i ~ Gamma(alpha_k, rate=beta_k) for i=1..n; inclusive tail areas reported."
```

Generally, we want posterior predictive p-values that are not too large and not too small (so somewhere in the range of 0.2 to 0.7, or so); the values we observe for the statitics of interest do not seem suitable. Further comparison and analysis is discussed in Question 6.

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)
  attr(res, "acceptprob") <- acceptind / M
  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){
```

```r
# propose jump from random walk for beta (rate), enforce positivity
# 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:
```

```r
    #       log p(alpha | beta, y) = n*alpha*log(beta) - n*log(Gamma(alpha))
    #                              + (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))
}
#----------------------------------------------------------------
```