TMA4300 INLA Practice Solutions

John Paige

The document is intended as a tool for students to practice using INLA with the R-INLA package. Students are welcome to work together in groups, and to ask questions as they progress in this worksheet. When coding, students should take turns:

1. *coding up various components of the samplers, and*
2. *looking over and double checking other group members' code as it is being written.*

Make sure your code is well commented and has understandable variable and function names.

Problem setup

As in the MCMC practice, assume we are interested in the response, relative to a baseline, of a patient after being assigned to one of two groups. We have n patients in a treatment group that receive medication for a condition, and n other patients in a control group that receive a placebo. Measurements are obtained for the n patients in each group, denoted by Y_{Ti} and Y_{Ci} for the treatment and control groups respectively, and for $i = 1, 2, \dots, n$.

The responses are modeled as, for $G \in \{T, C\}$ denoting the patient group,

$$Y_{Gi} = \mu_G + \epsilon_{Gi},$$

where we model μ_G as a Gaussian latent effect with prior $\mu_T, \mu_C \mid \nu^2 \overset{iid}{\sim} N(0, \nu^2)$, we assume Gaussian error, $\epsilon_{Ti}, \epsilon_{CI} \mid \sigma^2 \overset{iid}{\sim} N(0, \sigma^2)$ for $i = 1, \dots, n$, and we place inverse gamma hyperpriors on both σ^2 and ν^2 so that $\sigma^2, \nu^2 \overset{iid}{\sim} \text{Inv-Gamma}(\alpha, \beta)$. Hence, our hyperpriors have density:

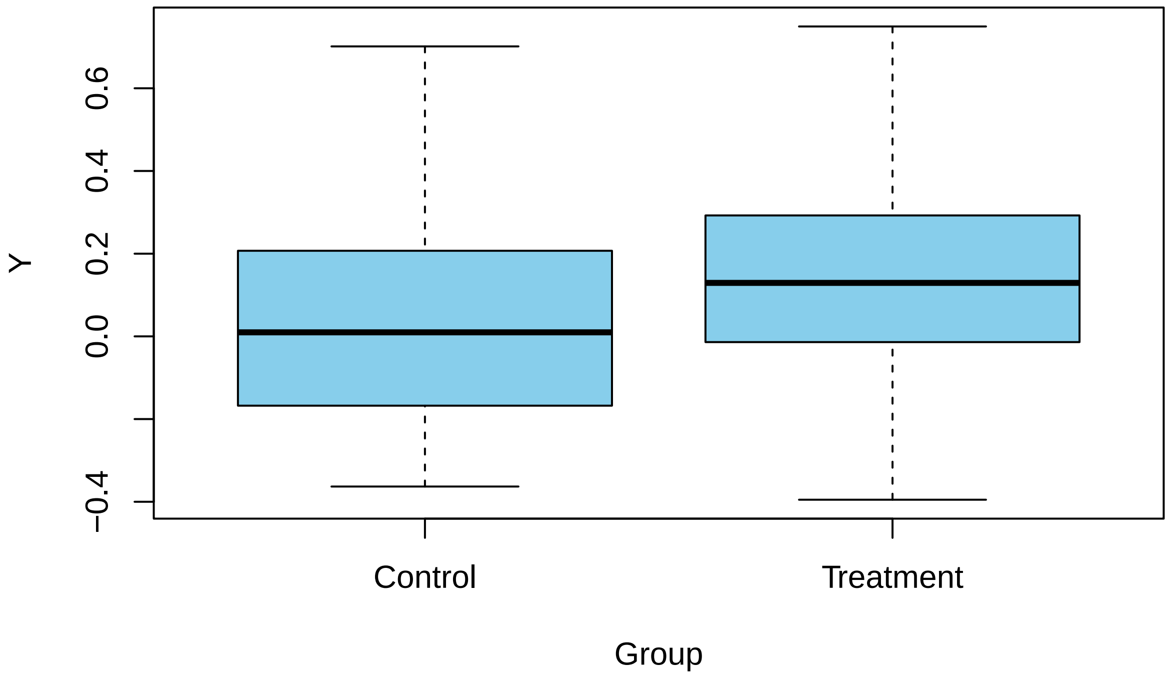
$$p(\sigma^2) = \frac{\beta^\alpha}{\Gamma(\alpha)}(1/\sigma^2)^{\alpha+1}\exp\{-\beta/\sigma^2\}$$
$$p(\nu^2) = \frac{\beta^\alpha}{\Gamma(\alpha)}(1/\nu^2)^{\alpha+1}\exp\{-\beta/\nu^2\}.$$

Let $\boldsymbol{Y}_T = Y_{T1}, \dots, Y_{Tn}$ and $\boldsymbol{Y}_C = Y_{C1}, \dots, Y_{Cn}$.

The following code simulates the data:

```r
library(invgamma)

# simulate data based on true parameters
alpha = 2
beta = 0.05
set.seed(1)
sigma2 = rinvgamma(1, alpha, beta)
nu2 = rinvgamma(1, alpha, beta)
muT = rnorm(1, sd=sqrt(nu2))
muC = rnorm(1, sd=sqrt(nu2))
```

```r
n=100
YT = rnorm(n, muT, sd=sqrt(sigma2))
YC = rnorm(n, muC, sd=sqrt(sigma2))

# make dataset and a boxplot of the responses in the 2 groups
dat = data.frame(Group=c(rep("Treatment", n), rep("Control", n)), Y=c(YT, YC))
boxplot(Y ~ Group, data=dat, col="skyblue")
```



Before you begin, make sure to install R-INLA with the following command:

```r
install.packages("INLA",repos=c(getOption("repos"), INLA="https://inla.r-inla-download.org/R/stable"), 
```

1

Fit the above model in INLA assuming $\alpha = 2$ and $\beta = 0.05$. You may use the following formula:

```r
library(INLA)
```

```
## Loading required package: Matrix

## Loading required package: foreach

## Loading required package: parallel

## Loading required package: sp

## This is INLA_22.12.16 built 2022-12-23 13:36:23 UTC.
##  - See www.r-inla.org/contact-us for how to get help.
##  - To enable PARDISO sparse library; see inla.pardiso()
```

```r
# reorganize data
dat$Group[dat$Group == "Treatment"] = 1
dat$Group[dat$Group == "Control"] = 0
dat = rbind(dat[dat$Group==0,],
            dat[dat$Group==1,])
form = Y ~ -1 + f(Group, model="iid", hyper=list(prec=list(prior="loggamma", param=c(2, .05))))
```

For more information on the log gamma hyperprior on the log precision (i.e.~an inverse gamma prior on the variance), see `{r, eval=FALSE} inla.doc("loggamma")}`. In addition, since we are interested in the quantity $\mu_T - \mu_C$, we can define this linear combination of parameters as a quantity of interest for INLA to calculate the posterior marginal of using the following code:

```
lincomb = inla.make.lincomb(Group=c(-1, 1))
```

This can be used with the `lincomb` argument of the `inla()` function. Give a summary of the fit model. Plot all marginals, and compare estimates of the mean of each distribution to the true values generated above (see, for example, `?inla.qmarginal`, `?inla.mmarginal` and `?inla.tmarginal`). What do the results imply about the treatment? Does it significantly improve (increase) patient outcomes relative to the placebo? Make sure to set all priors correctly (see `?control.family` and the `control.family` argument of `?inla`).

Answer: We set the prior for the Gaussian precision to also be a log gamma prior with parameters $\alpha = 2$ and $\beta = 0.05$, and run INLA with the appropriate arguments:

```
result = inla(formula=form,
              family="gaussian",
              data=dat,
              lincomb=lincomb,
              control.family=list(hyper=list(prec=list(prior="loggamma", param=c(2, .05)))),
              verbose=FALSE)
```

A summary of the fit model is given below:

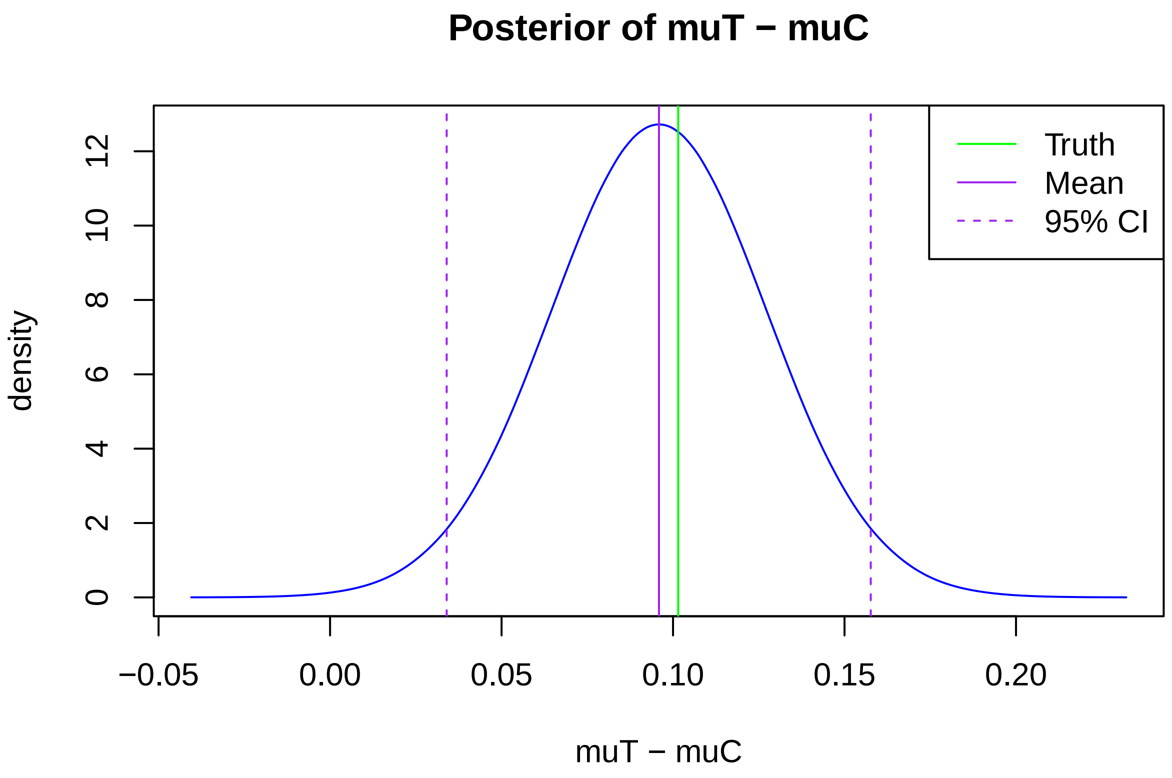
```
summary(result)
```

```
##
## Call:
##    c("inla.core(formula = formula, family = family, contrasts = contrasts,
##    ", " data = data, quantiles = quantiles, E = E, offset = offset, ", "
##    scale = scale, weights = weights, Ntrials = Ntrials, strata = strata,
##    ", " lp.scale = lp.scale, link.covariates = link.covariates, verbose =
##    verbose, ", " lincomb = lincomb, selection = selection, control.compute
##    = control.compute, ", " control.predictor = control.predictor,
##    control.family = control.family, ", " control.inla = control.inla,
##    control.fixed = control.fixed, ", " control.mode = control.mode,
##    control.expert = control.expert, ", " control.hazard = control.hazard,
##    control.lincomb = control.lincomb, ", " control.update =
##    control.update, control.lp.scale = control.lp.scale, ", "
##    control.pardiso = control.pardiso, only.hyperparam = only.hyperparam,
##    ", " inla.call = inla.call, inla.arg = inla.arg, num.threads =
##    num.threads, ", " blas.num.threads = blas.num.threads, keep = keep,
##    working.directory = working.directory, ", " silent = silent, inla.mode
##    = inla.mode, safe = FALSE, debug = debug, ", " .parent.frame =
##    .parent.frame)")
## Time used:
##     Pre = 2.96, Running = 0.398, Post = 0.078, Total = 3.44
## Linear combinations (derived):
##    ID  mean    sd 0.025quant 0.5quant 0.975quant  mode kld
## lc  1 0.096 0.032      0.034    0.096      0.158 0.096   0
##
## Random effects:
```

```
##    Name     Model
##      Group IID model
##
## Model hyperparameters:
##                                          mean    sd 0.025quant 0.5quant
## Precision for the Gaussian observations 19.70  1.96      16.07    19.62
## Precision for Group                     48.50 28.20      12.02    42.63
##                                         0.975quant  mode
## Precision for the Gaussian observations      23.79 19.50
## Precision for Group                         119.17 30.55
##
## Marginal log-Likelihood:  7.33
##  is computed
## Posterior summaries for the linear predictor and the fitted values are computed
## (Posterior marginals needs also 'control.compute=list(return.marginals.predictor=TRUE)')
```

On average, the treatment resulted in an improvement in patient outcomes of 0.096 units relative to the placebo, and 0.139 relative to their baseline prior to treatment. Hence, accounting for both data and prior beliefs, it is very likely the treatment improves patient outcomes relative to the placebo and also overall. A marginal of the contrast is plotted below.

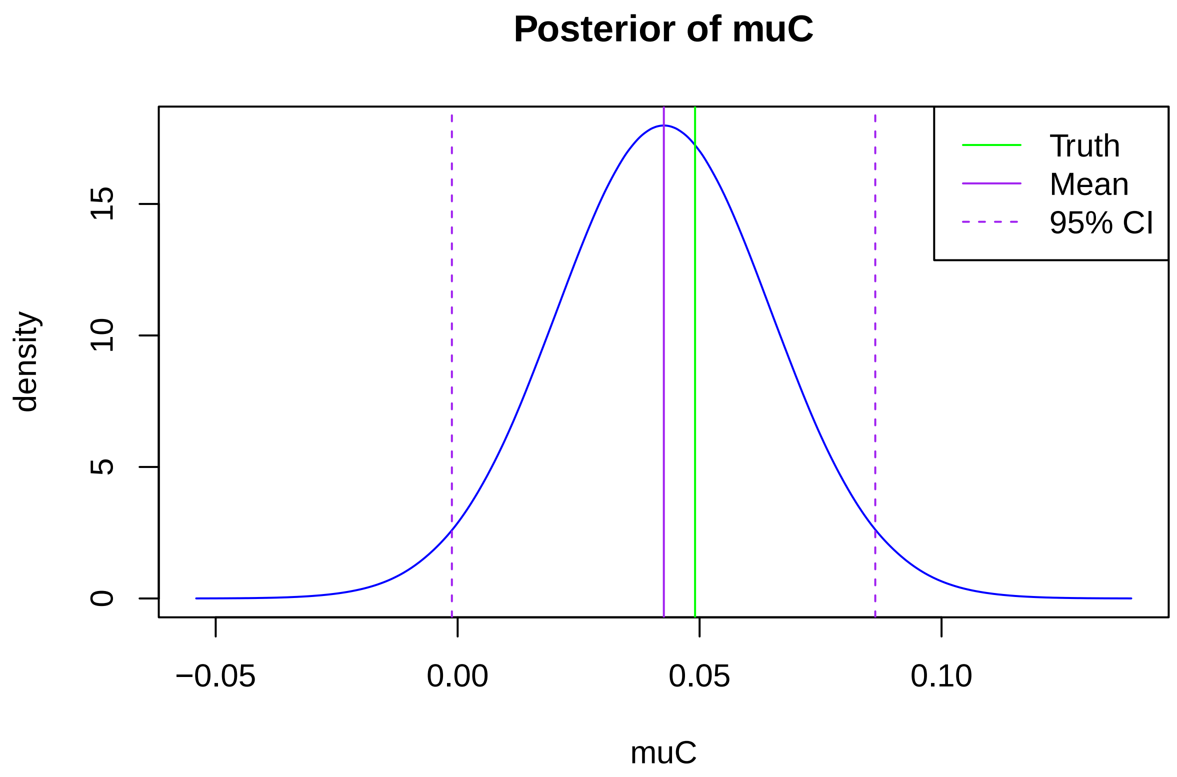
```r
plot(inla.smarginal(result$marginals.lincomb.derived$lc), type="l", col="blue",
     xlab="muT - muC", ylab="density", main="Posterior of muT - muC")
abline(v=inla.qmarginal(c(0.025, 0.975), result$marginals.lincomb.derived$lc), lty=2, col="purple")
abline(v=muT-muC, col="green")
abline(v=result$summary.lincomb.derived[1,2], col="purple")
legend("topright", c("Truth", "Mean", "95% CI"), lty=c(1, 1, 2), col=c("green", "purple", "purple"))
```



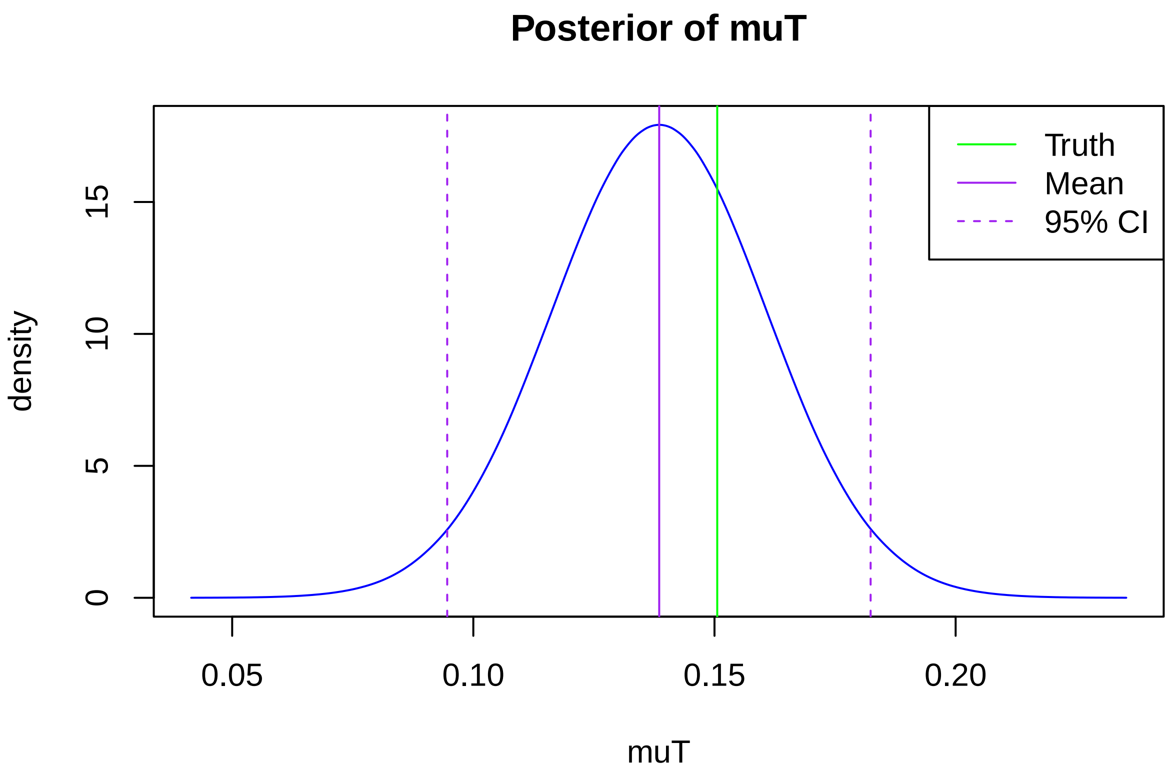
Posteriors for the group effects are shown below:

```r
plot(inla.smarginal(result$marginals.random$Group$index.1), type="l", col="blue",
     xlab="muC", ylab="density", main="Posterior of muC")
```

```
abline(v=inla.qmarginal(c(0.025, 0.975), result$marginals.random$Group$index.1), lty=2, col="purple")
abline(v=muC, col="green")
abline(v=result$summary.random$Group[1,2], col="purple")
legend("topright", c("Truth", "Mean", "95% CI"), lty=c(1, 1, 2), col=c("green", "purple", "purple"))
```



```
plot(inla.smarginal(result$marginals.random$Group$index.2), type="l", col="blue",
     xlab="muT", ylab="density", main="Posterior of muT")
abline(v=inla.qmarginal(c(0.025, 0.975), result$marginals.random$Group$index.2), lty=2, col="purple")
abline(v=muT, col="green")
abline(v=result$summary.random$Group[2,2], col="purple")
legend("topright", c("Truth", "Mean", "95% CI"), lty=c(1, 1, 2), col=c("green", "purple", "purple"))
```



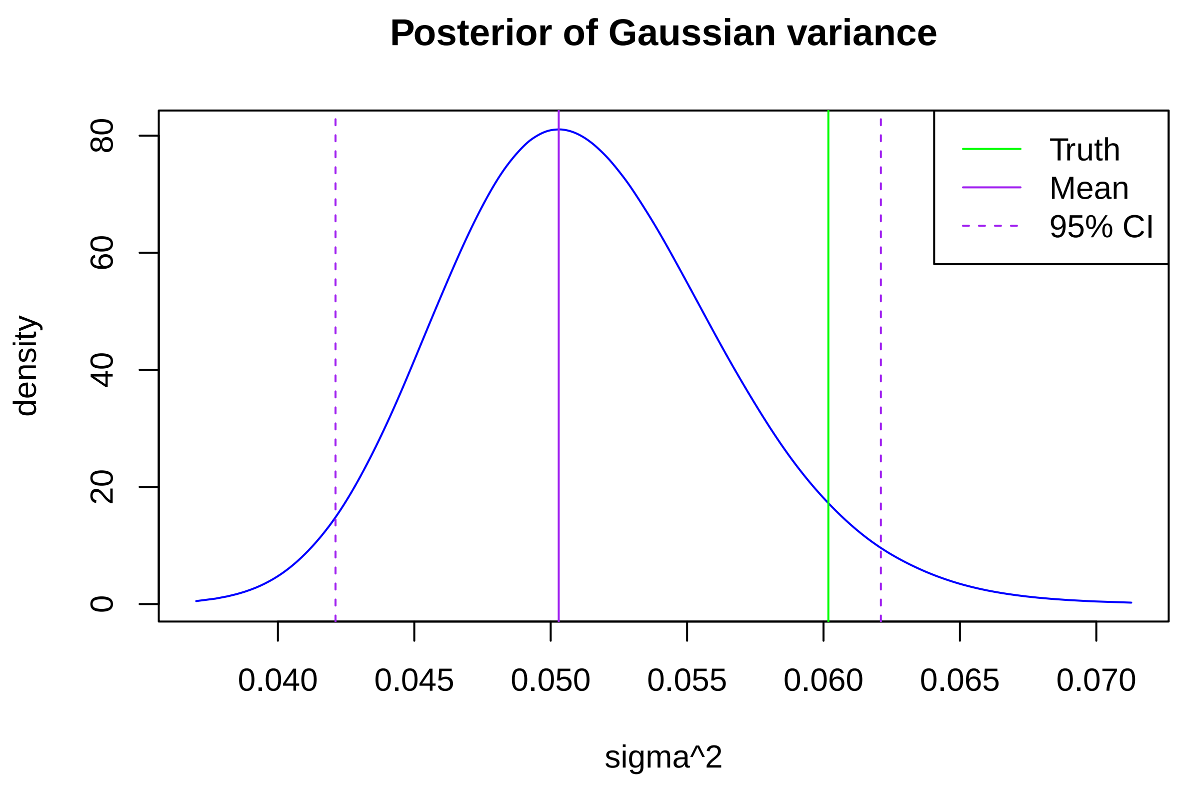
It is also worth nothing that the true value of the Gaussian variance and group variance are respectively 0.1 and 0, and both of these are in INLA's 95% credible intervals of (0, 0.1) for the Gaussian variance and (0, 0.1) for the group variance.

Before plotting the marginals of the variance parameters, we must invert the precision marginals using `{r, eval=FALSE} inla.tmarginal()`:

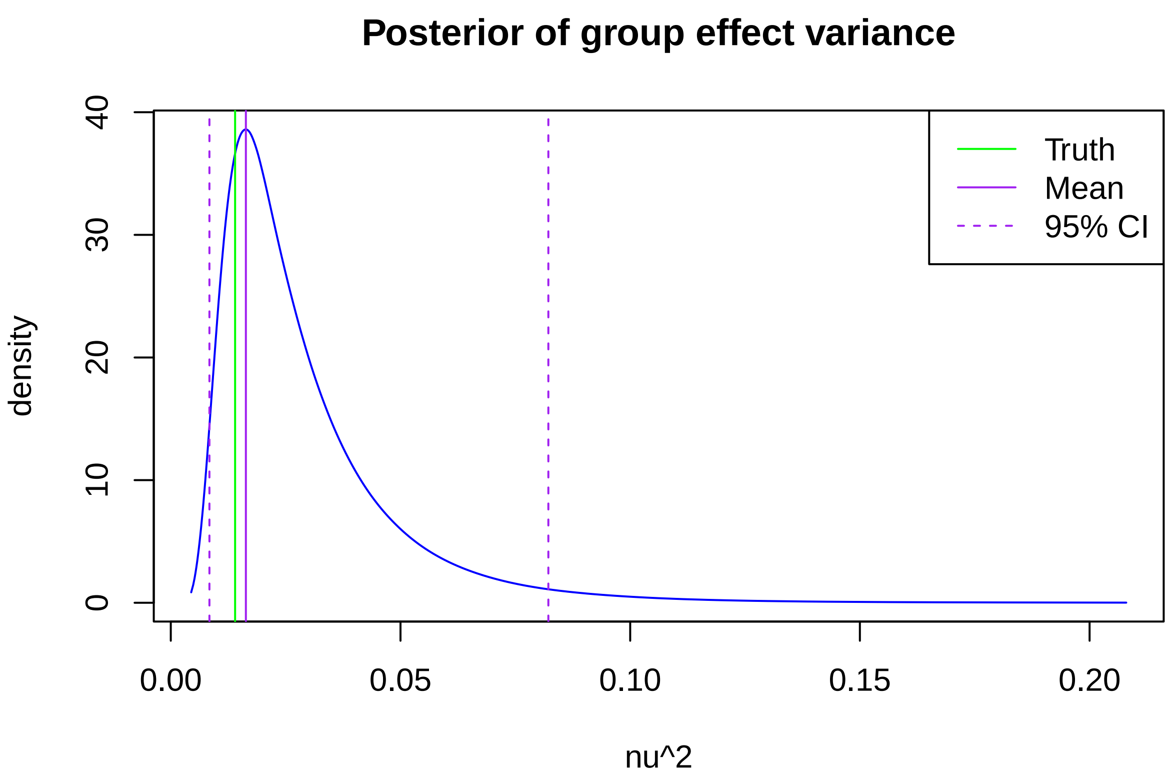
```
sigma2Marg = inla.tmarginal(function(x){1/x}, result$marginals.hyperpar$`Precision for the Gaussian obse
nu2Marg = inla.tmarginal(function(x){1/x}, result$marginals.hyperpar$`Precision for Group`)
```

Now we plot them:

```
plot(sigma2Marg, type="l", col="blue",
     xlab="sigma^2", ylab="density", main="Posterior of Gaussian variance")
abline(v=inla.qmarginal(c(0.025, 0.975), sigma2Marg), lty=2, col="purple")
abline(v=sigma2, col="green")
abline(v=inla.mmarginal(sigma2Marg), col="purple")
legend("topright", c("Truth", "Mean", "95% CI"), lty=c(1, 1, 2), col=c("green", "purple", "purple"))
```

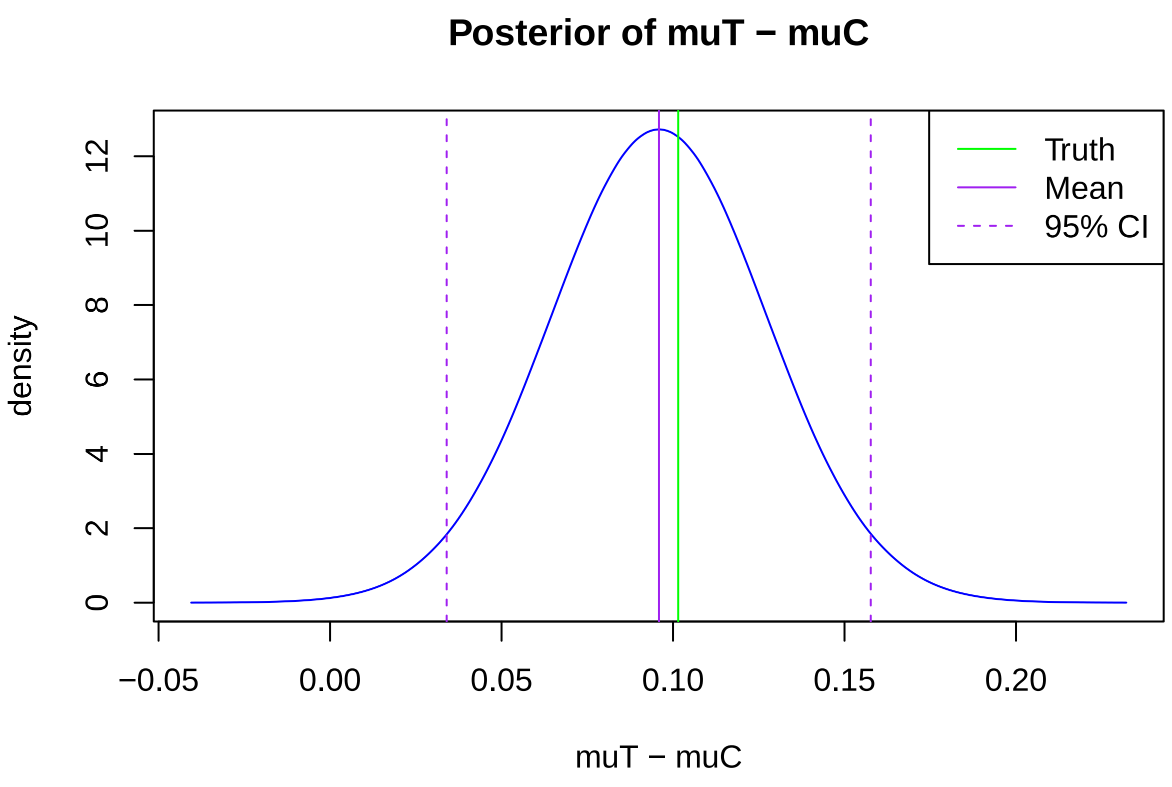


```
plot(nu2Marg, type="l", col="blue",
     xlab="nu^2", ylab="density", main="Posterior of group effect variance")
abline(v=inla.qmarginal(c(0.025, 0.975), nu2Marg), lty=2, col="purple")
abline(v=nu2, col="green")
abline(v=inla.mmarginal(nu2Marg), col="purple")
legend("topright", c("Truth", "Mean", "95% CI"), lty=c(1, 1, 2), col=c("green", "purple", "purple"))
```



Below we plot the marginals of the variances:

```
plot(inla.smarginal(result$marginals.lincomb.derived$lc), type="l", col="blue",
     xlab="muT - muC", ylab="density", main="Posterior of muT - muC")
abline(v=inla.qmarginal(c(0.025, 0.975), result$marginals.lincomb.derived$lc), lty=2, col="purple")
abline(v=muT-muC, col="green")
abline(v=result$summary.lincomb.derived[1,2], col="purple")
legend("topright", c("Truth", "Mean", "95% CI"), lty=c(1, 1, 2), col=c("green", "purple", "purple"))
```



2

How do your results compare to the MCMC results from the MCMC practice exercise? Do the marginals agree with the histograms generated from the MCMC sampler? Plot them together to compare them.

Answer: We rerun the MCMC code below, modified slightly to handle arbitrary datasets:

```
# calculates the acceptance probability on a log scale. Unlike the notation in the solutions,
# I name variables "prop" instead of "tilde", since they are the proposed values.
acceptProb = function(muTprop, muCprop, muT, muC, nu2, sigma2, YT, YC) {
  firstTerm = (-0.5/nu2) * (muTprop^2 + muCprop^2 - muT^2 - muC^2)
  secondTerm = (-0.5/sigma2) * sum((YT - muTprop)^2 + (YC - muCprop)^2 - (YT - muT)^2 - (YC - muC)^2)
  min(c(1, exp(firstTerm + secondTerm)))
}

# function for doing Metropolis-Hastings. Keeps track of acceptance rate.
# returns:
#     sampleMat: (M-burnin) x 4 matrix of samples
#     acceptRate: proportion of MH steps accepted
MH = function(M=10000, burnin=2000, tau2=.1, printEvery=100, verbose=FALSE, YT, YC) {
  # initialize matrix of samples and acceptances and x_0. Note indices start at 1 not 0
  xmat = matrix(0, nrow=M+1, ncol=4)
  xmat[1,] = c(1, 1, 0, 0)
  accepts = rep(FALSE, M)

  # set current state (sigma2, nu2, muT, muC)
  x = xmat[1,]

  # generate samples
  startTime = proc.time()[3]
  for(i in 1:M) {
    if(((i %% printEvery) == 0) && verbose) {
      # print current state and expected computation time left:
      currState = paste(x, collapse=", ")
      print(paste0("current state for iteration ", i, "/", M, ": ", currState))

      currTime = proc.time()[3]
      timeTaken = currTime - startTime
      fracDone = (i-1)/M
      fracLeft = 1 - fracDone
      timeLeftEst = (timeTaken / fracDone) * fracLeft
      print(paste0("estimated time left: ", round(timeLeftEst), " seconds"))
    }

    # 2a: Gibbs step, draw sigma2 from invgam()
    shapeSigma2 = alpha+n
    rateSigma2 = beta + 0.5*(sum((YT - x[3])^2 + (YC - x[4])^2))
    x[1] = invgamma::rinvgamma(1, shape=shapeSigma2, rate=rateSigma2)

    # 2b: Gibbs step, draw nu2 from invgam()
    shapeNu2 = alpha+1
    rateNu2 = beta + 0.5*(x[3]^2 + x[4]^2)
    x[2] = invgamma::rinvgamma(1, shape=shapeNu2, rate=rateNu2)

    # 2c: MH step propose muT, muC from N(x[3:4], tau^2 I_2)
    ## proposal
    muTprop = rnorm(1, mean=x[3], sd=sqrt(tau2))
    muCprop = rnorm(1, mean=x[4], sd=sqrt(tau2))

    ## accept/reject step
    acceptP = acceptProb(muTprop, muCprop, muT=x[3], muC=x[4], nu2=x[2], sigma2=x[1], YT=YT, YC=YC)
    accept = runif(1) < acceptP
    if(accept) {
      x[3:4] = c(muTprop, muCprop)
      accepts[i] = TRUE
    }
    # otherwise, x[3:4] is already set to the previous state, so we don't
    # need to do anything

    # 2d: store our sample. Note the index of xmat starts at 1 not 0, so
    # xmat[i+1,] is the i-th sample
    xmat[i+1,] = x
  }

  acceptRate=mean(accepts)

  # print total time take and acceptance rate
  currTime = proc.time()[3]
  print(paste0("Total time taken: ", round((currTime - startTime)/60, digits=2), " minutes"))
  print(paste0("Acceptance rate: ", round(acceptRate, digits=4)))
  list(sampleMat = xmat[(burnin+1):(M+1),], acceptRate=acceptRate)
}
```

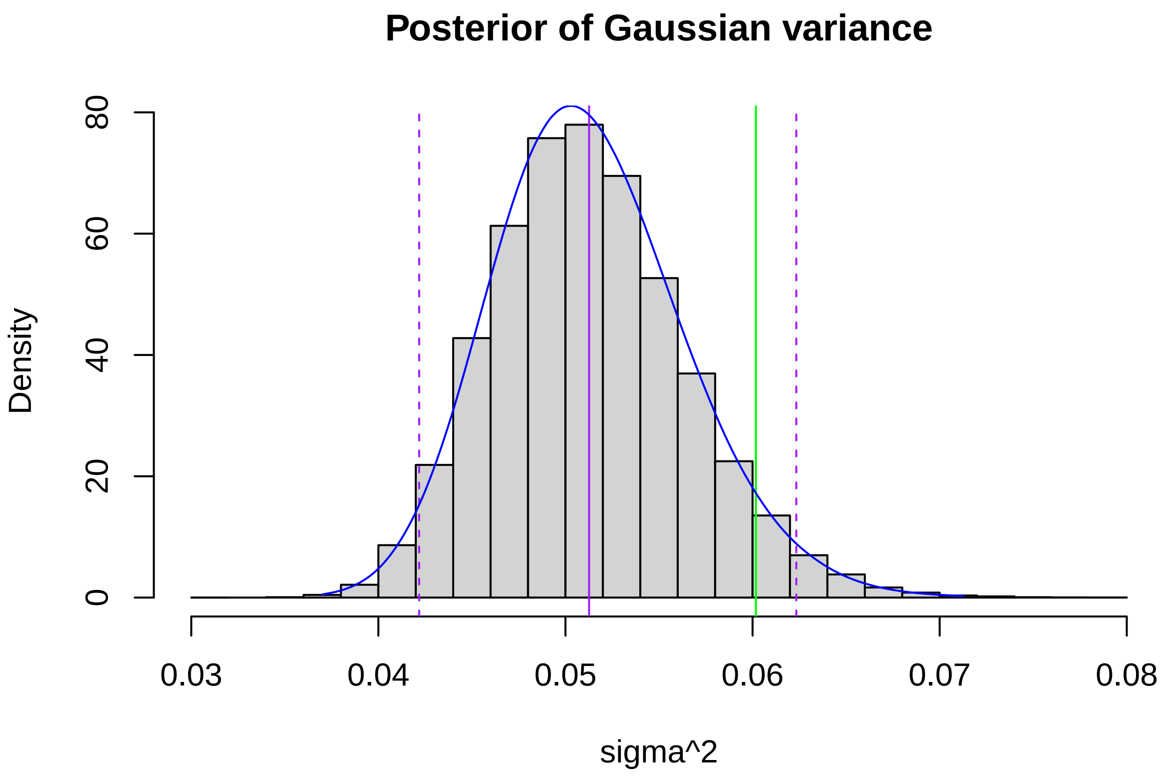
```r
set.seed(123)

# run MH and collect results
results = MH(M=100000, burnin=10000, printEvery=10000, tau2 = .0025, verbose=FALSE, YT=YT, YC=YC)
```

```
## [1] "Total time taken: 0.05 minutes"
## [1] "Acceptance rate: 0.2532"
```

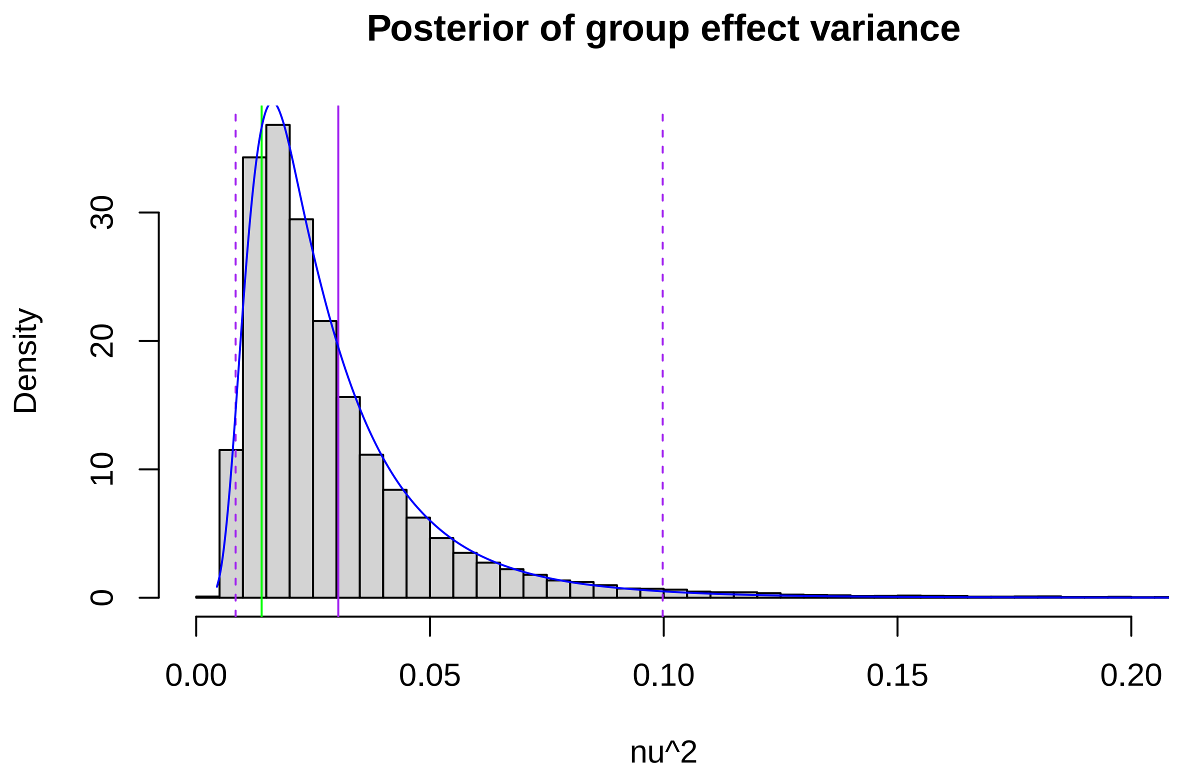
```r
samples = results$sampleMat
acceptRate = results$acceptRate
```

Plot the results:

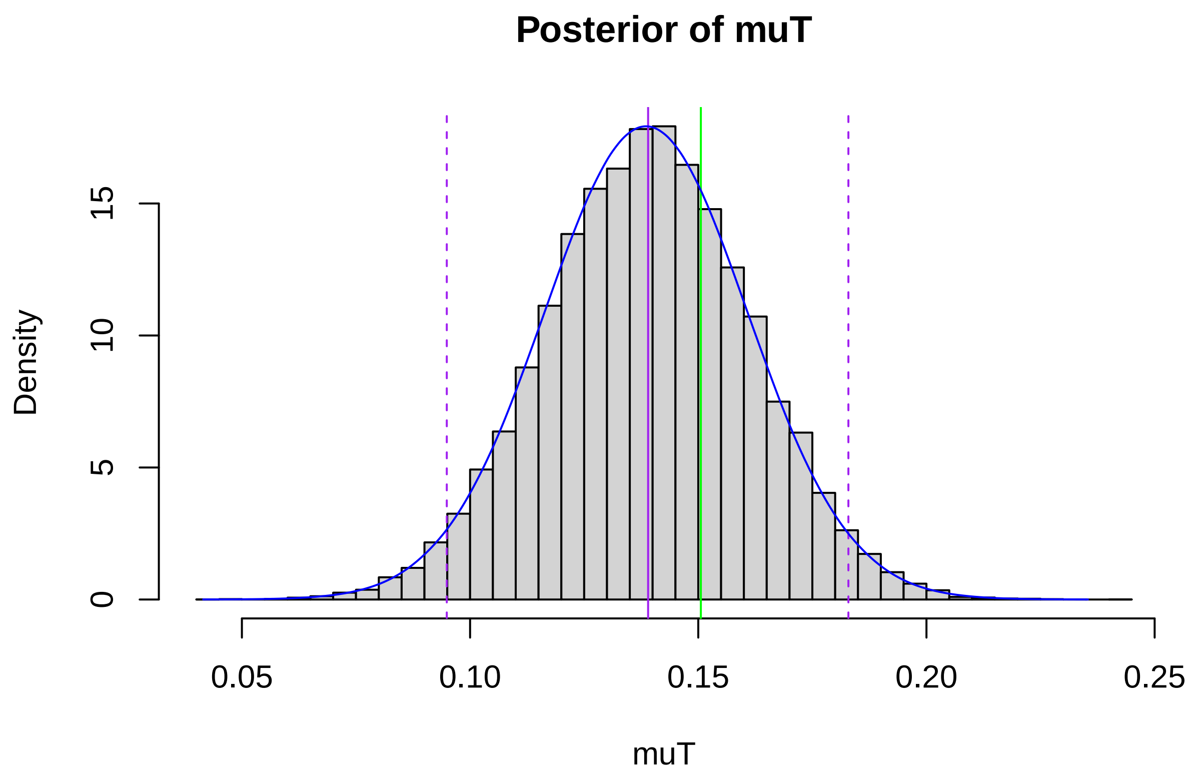
```r
# generate histogram of marginals, adding in marginals from INLA
hist(samples[,1], freq=F, breaks=30, main="Posterior of Gaussian variance", xlab="sigma^2")
abline(v=sigma2, col="green")
abline(v=mean(samples[,1]), col="purple")
abline(v=quantile(samples[,1], prob=c(0.025, .975)), col="purple", lty=2)
lines(sigma2Marg, col="blue")
```



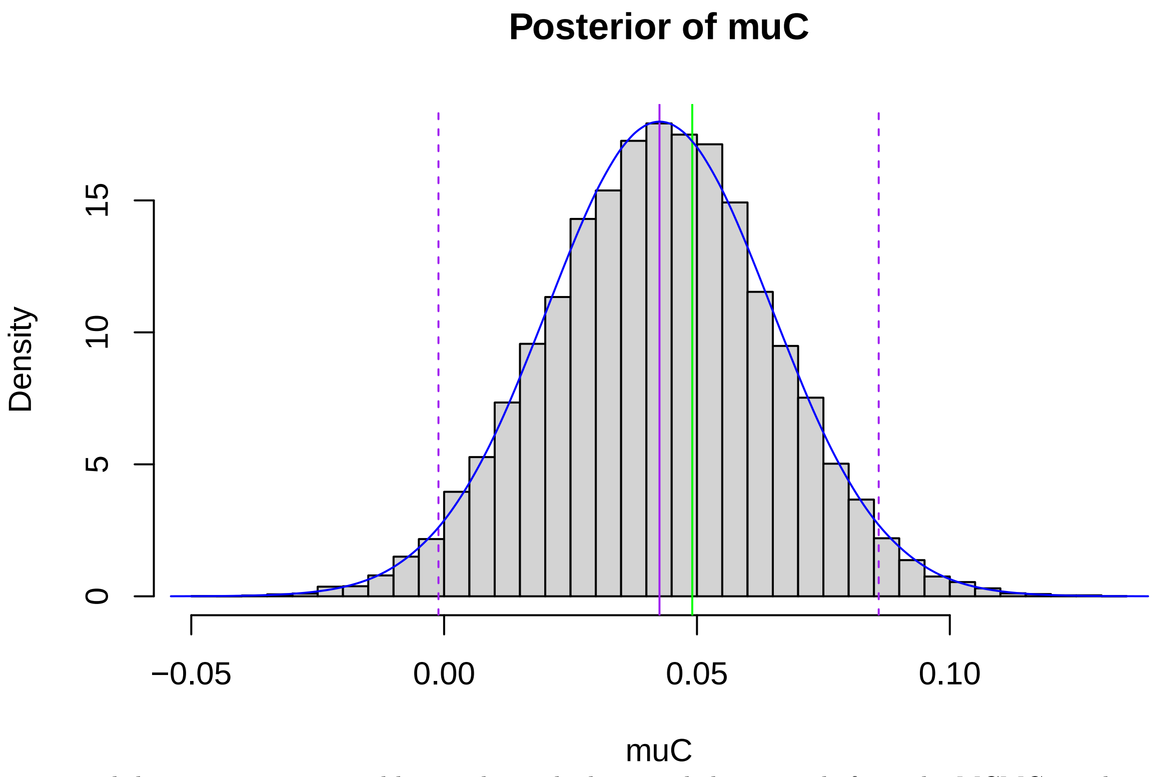
```r
hist(samples[,2], freq=F, breaks=200, xlim=c(0, .2),  main="Posterior of group effect variance", xlab="
abline(v=nu2, col="green")
abline(v=mean(samples[,2]), col="purple")
abline(v=quantile(samples[,2], prob=c(0.025, .975)), col="purple", lty=2)
lines(nu2Marg, col="blue")
```



```
hist(samples[,3], freq=F, breaks=30, main="Posterior of muT", xlab="muT")
abline(v=muT, col="green")
abline(v=mean(samples[,3]), col="purple")
abline(v=quantile(samples[,3], prob=c(0.025, .975)), col="purple", lty=2)
lines(inla.smarginal(result$marginals.random$Group$index.2), col="blue")
```



```r
hist(samples[,4], freq=F, breaks=30, main="Posterior of muC", xlab="muC")
abline(v=muC, col="green")
abline(v=mean(samples[,4]), col="purple")
abline(v=quantile(samples[,4], prob=c(0.025, .975)), col="purple", lty=2)
lines(inla.smarginal(result$marginals.random$Group$index.1), col="blue")
```



INLA's estimated densities are given in blue, and match the sampled marginals from the MCMC nearly exactly.

3

How do the computation times between INLA and MCMC compare? You may use `proc.time()[3]` to get the current time in seconds, taking differences to get total time taken. What if n were 1000 instead of 100? 10000? Make a plot of the computation time of INLA and the MCMC sampler versus n from $n = 100$ to $n = 10000$ on a log scale.

Answer: We consider 10 values of n from 100 to 10000, spaced out evenly on a log scale, and a function for simulating the data given n:

```r
ns = round(10^(seq(2, 4, l=10)))

simDat = function(thisN) {
  YT = rnorm(thisN, muT, sd=sqrt(sigma2))
  YC = rnorm(thisN, muC, sd=sqrt(sigma2))

  # make dataset
  dat = data.frame(Group=c(rep("Treatment", thisN), rep("Control", thisN)), Y=c(YT, YC))

  # adjust to format expected by INLA
  dat$Group[dat$Group == "Treatment"] = 1
  dat$Group[dat$Group == "Control"] = 0
  dat = rbind(dat[dat$Group==0,],
              dat[dat$Group==1,])

  dat
}

# simulate datasets
set.seed(123)
simDatList = list()
for(i in 1:length(ns)) {
  simDatList[[i]] = simDat(ns[i])
}
```

Now we call INLA for each dataset, keeping track of computation time:

```
timesINLA = rep(0, length(ns)) # initialize
for(i in 1:length(ns)) {
  thisDat = simDatList[[i]]

  startTime = proc.time()[3]
  result = inla(formula=form,
              family="gaussian",
              data=thisDat,
              lincomb=lincomb,
              control.family=list(hyper=list(prec=list(prior="loggamma", param=c(2, .05)))),
              verbose=FALSE)
  endTime = proc.time()[3]

  timesINLA[i] = endTime - startTime

  # print computation time:
  print(paste0("for n=", ns[i], ", INLA took ", round(timesINLA[i], 1), " seconds"))
}
```

```
## [1] "for n=100, INLA took 3.7 seconds"
## [1] "for n=167, INLA took 3.5 seconds"
## [1] "for n=278, INLA took 3.6 seconds"
## [1] "for n=464, INLA took 3.6 seconds"
## [1] "for n=774, INLA took 3.6 seconds"
## [1] "for n=1292, INLA took 3.6 seconds"
## [1] "for n=2154, INLA took 3.8 seconds"
## [1] "for n=3594, INLA took 3.8 seconds"
## [1] "for n=5995, INLA took 4.1 seconds"
## [1] "for n=10000, INLA took 4.5 seconds"
```

```
print(timesINLA)
```

```
##  [1] 3.664 3.477 3.579 3.608 3.610 3.562 3.774 3.787 4.084 4.478
```

We see that the computation time is mostly setup and processing for INLA, even when $n = 10000$. Now we do the same for the MCMC sampler:

```
set.seed(123)
timesMCMC = rep(0, length(ns)) # initialize
for(i in 1:length(ns)) {
  thisDat = simDatList[[i]]
  thisYT = thisDat$Y[thisDat$Group == 1]
  thisYC = thisDat$Y[thisDat$Group == 0]

  startTime = proc.time()[3]
  results = MH(M=100000, burnin=10000, printEvery=25000, tau2 = .0025, verbose=FALSE, YT=thisYT, YC=thi
  endTime = proc.time()[3]

  timesMCMC[i] = endTime - startTime

  # print computation time:
  print(paste0("for n=", ns[i], ", MCMC took ", round(timesMCMC[i], 1), " seconds"))
}
```

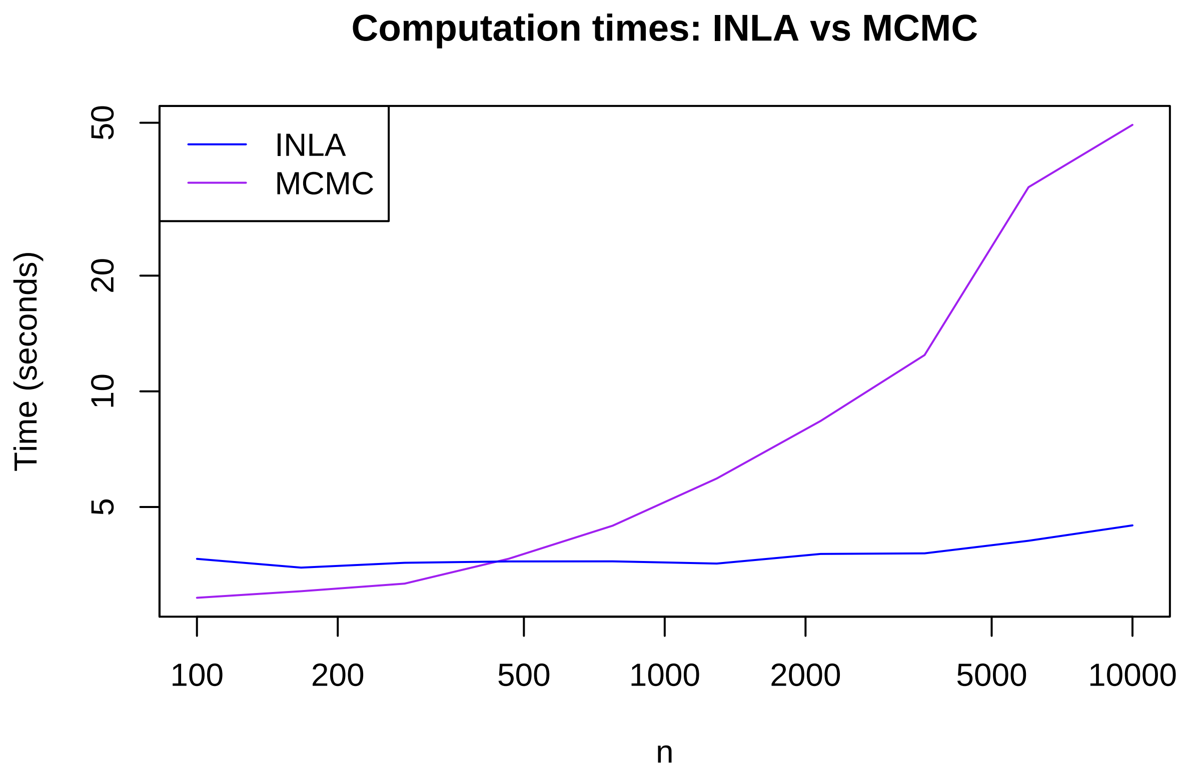
```
## [1] "Total time taken: 0.05 minutes"
## [1] "Acceptance rate: 0.2582"
## [1] "for n=100, MCMC took 2.9 seconds"
## [1] "Total time taken: 0.05 minutes"
## [1] "Acceptance rate: 0.2743"
## [1] "for n=167, MCMC took 3 seconds"
## [1] "Total time taken: 0.05 minutes"
## [1] "Acceptance rate: 0.2907"
## [1] "for n=278, MCMC took 3.2 seconds"
## [1] "Total time taken: 0.06 minutes"
## [1] "Acceptance rate: 0.2819"
## [1] "for n=464, MCMC took 3.7 seconds"
## [1] "Total time taken: 0.07 minutes"
## [1] "Acceptance rate: 0.2726"
## [1] "for n=774, MCMC took 4.5 seconds"
## [1] "Total time taken: 0.1 minutes"
## [1] "Acceptance rate: 0.2786"
## [1] "for n=1292, MCMC took 5.9 seconds"
## [1] "Total time taken: 0.14 minutes"
## [1] "Acceptance rate: 0.2794"
## [1] "for n=2154, MCMC took 8.4 seconds"
## [1] "Total time taken: 0.21 minutes"
## [1] "Acceptance rate: 0.2814"
## [1] "for n=3594, MCMC took 12.4 seconds"
## [1] "Total time taken: 0.57 minutes"
## [1] "Acceptance rate: 0.2786"
## [1] "for n=5995, MCMC took 34 seconds"
## [1] "Total time taken: 0.82 minutes"
## [1] "Acceptance rate: 0.2814"
## [1] "for n=10000, MCMC took 49.4 seconds"
```

```
print(timesMCMC)
```

```
##  [1]  2.902  3.018  3.159  3.666  4.470  5.931  8.373 12.428 33.989 49.379
```

Let's plot the results:

```
plot(ns, timesMCMC, type="l", main="Computation times: INLA vs MCMC",
     xlab="n", ylab="Time (seconds)", log="xy", col="purple")
lines(ns, timesINLA, col="blue")
legend("topleft", c("INLA", "MCMC"), col=c("blue", "purple"), lty=1)
```



We see that the MCMC takes much more time than INLA as the dataset size increases.