The results below are generated from an R script.

```
# metropolis.R
# collection of R functions for using the random walk Metropolis-Hastings
# algorithm for MCMC
#
# See e.g. Roberts (2015) http://arxiv.org/abs/1504.01896
```

```
# To do:
#  Fix R.hat and plots to cope with 'flattened' output
#
#
```

```
# The MCMC routine. Metropolis-Hastings algorithm
# using a Normal random walk for the proposal
# distribution.
#
#  theta.0 - vector of starting parameter values
#  cov     - covariance matrix for proposal dist.
#  M       - number of parameters in theta
#  N       - number of iterations to run
#  accept  - counter for number of acceptances

mh.mcmc <- function(posterior,
                    theta.0,
                    nsteps = 1E4,
                    nchains = 5,
                    burn.in = 10,
                    update = 5,
                    chatter = 1,
                    cov = NULL,
                    thin = NULL,
                    merge.chains = TRUE,
                    proposal = "normal", ...) {

  # check the input arguments
  if (missing(theta.0)) stop('Must specify theta.0 start position.')
  if (missing(posterior)) stop('Must specify name of posterior function')
  if (!exists('posterior'))
    stop('The specified log density function does not exist.')
  if (is.null(cov))
    stop('Must specify the covariance matrix (cov).')

  # dimensions of the PDF
  M <- length(theta.0)

  # ensure the number of steps, walkers, etc. are integers
  nsteps <- as.integer(nsteps)
  nchains <- as.integer(nchains)
  burn.in <- as.integer(burn.in)

  # prepare a working array
  p.prop <- array(NA, dim = nchains)
```

```r
# number of interations needed
nrows.keep <- ceiling(nsteps / nchains)
if (nrows.keep < 10) stop('Make nsteps larger')
nrows.burnin <- ceiling(burn.in / nchains)
ncycles <- nrows.keep + nrows.burnin

# initialise the array for output. There are two additional 'columns': The M+1
# column stores the accept/reject flag (0=rejected, 1=accepted proposal at
# each update). This is useful for tracking the acceptance rate. The M+2
# column stores the log posterior (PDF) values at the current position of the
# chain. This saves recomputing the posterior density at the current position
# when evaluating the accept probability.
theta <- array(NA, dim=c(ncycles, nchains, M+2))

# starting locations of chains
for (j in 1:nchains) {
   z <- mvtnorm::rmvnorm(1, mean = theta.0, sigma = cov)
   theta[1, j, 1:M] <- z  # randomized start position
   theta[1, j, M+1] <- 1  # accept
   theta[1, j, M+2] <- posterior(z, ...)
}

# loop over the chain

for (i in 2:ncycles) {

  # carry forward the previous theta value
  # will over-write if update occurs
  theta[i, , ] <- theta[i-1, , ]
  theta[i, , M+1] <- 0

  # draw a value from the proposal distribution
  # either Normal or Student's t distribution (df=3)
  if (proposal == "normal") {
    z <- mvtnorm::rmvnorm(nchains, sigma = cov)
  } else {
    z <- mvtnorm::rmvt(nchains, sigma = cov/3, df = 3)
  }
  # add these random vectors to the previous position
  # to produce a new 'proposal' position
  theta.prop <- theta[i, 1:nchains, 1:M] + z

  # now for each chain compute the log posterior density
  # at the proposed position.
  for (j in 1:nchains) {
    p.prop[j] <- posterior(theta.prop[j, ], ...)
  }

  # compute ratio of posteriors at new and old locations
  #   r = p(theta.prop) / p(theta[t-1])
  # in terms of the log posterior function
  # this is log[p(theta.old)] - log[p(theta.new)]
  # (log.r is a vector with nchains elements.)
```

```
  p.pres <- theta[i, , M+2]
  log.r <- p.prop - p.pres

  # decide whether or not to update theta.
  # theta is updated with probability min(r,1)
  # otherwise it is left as before.
  r <- exp( pmin(log.r, 0) )
  u <- runif(nchains)
  mask <- (r >= u)
  theta[i, mask, 1:M] <- theta.prop[mask, 1:M]
  theta[i, mask, M+1] <- 1
  theta[i, mask, M+2] <- p.prop[mask]

  # progress report to user if requested
  i.count <- 1
  if (chatter > 0) {
    if (i %% update == 0) {
      accept.rate <- mean( theta[i.count:i, , M+1], na.rm=TRUE )
      cat("\r-- Cycle", i, "of", ncycles, ". Acceptance rate:",
          signif(accept.rate*100, 2), "%")
    }
    if (i == ncycles) cat('', fill=TRUE)
    if (i == nrows.burnin) {
      cat(' - Finished burn-in', fill=TRUE)
      i.count <- nrows.burnin+1
    }
  }
} # end of loop over i = 2, ncycles

# strip off the burn-in period and keep only nsteps
nrows <- nrows.keep
theta <- theta[(1:nrows) + nrows.burnin, , ]

# thin the output by keeping only every few rows
if (!is.null(thin)) {
  nrow.keep <- floor(nrows / thin)
  mask <- (1:nrow.keep) * thin
  theta <- theta[mask, , ]
}

# Strip off the acceptance and log(posterior) columns
accept <- theta[, , M+1]
lpost <- as.vector( theta[, , M+2])
theta <- theta[, , 1:M]

# check the acceptance rate (column M+1).
# Also strip off the log(posterior) values which are no longer needed.
accept.rate <- mean(accept, na.rm = TRUE)
if (chatter > 0) {
  cat('\n-- Final acceptance rate: ', accept.rate, fill = TRUE)
  if (accept.rate < 0.05) {
    cat('-- Low acceptance rate. Consider the following suggestions:',
        fill = TRUE)
```

```
      cat('-- 1. Adjust the start position: theta.0,', fill = TRUE)
      cat('-- 2. Decrease the covariance for proposal distribution.', fill = TRUE
      )
    }
    }

  # reshape the array from [nrows, M, nwalkers] to [nrows*nwalkers, M]
  # so each column is one variable, each row is one sample from the M
  # M-dimensional distribution.
  if (merge.chains == TRUE) {
    nrows <- dim(theta)[1]
    theta <- matrix(theta, nrow = nchains * nrows, ncol = M, byrow = FALSE)
  }

  # return the final array
  return(list(theta = theta,
              func = deparse(substitute(posterior)),
              lpost = lpost,
              method = "mh.mcmc",
              nchains = nchains))
}
```

```
# for each parameter theta[1]...theta[M]
# calculate the R.hat statistic (Gelman & Rubin 1992)
# See also Gelman et al. (2004, sect 11.6)

Rhat <- function(theta, M, L) {

  R.hat <- array(0, dim=M)
  for (m in 1:M) {
    tj <- theta[m,,]
    sj <- apply(tj, 2, var)
    W <- mean(sj)
    mj <- apply(tj, 2, mean)
    B <- var(mj)*L
    v <- (L-1)/L*W + B/L
    R.hat[m] <- sqrt(v/W)

    cat("-- Theta[",m,"] R.hat = ",R.hat[m],sep="")
    if (R.hat[m] > 1.2) {
      cat(" ** High R.hat. Check results. **",fill=TRUE)
    } else {
      cat(" -- Good R.hat.", fill=TRUE)
    }
  }
  return(R.hat)
}
```

```
# Perform checks for convergence of multiple
# Markov chains. Uses Gelman & Rubin's R.hat
# for each parameter, and also a visual check of
# the 80% regions for each parameter.
```

```
mcmc.conv <- function(theta, M, L, J) {

  layout(t(c(1,2)), widths=c(0.3,0.7))

  # Calculate R.hat (Gelman & Rubin 1992) for each
  # parameter as a check for convergence of chains

  R.hat <- Rhat(theta, M, L)

  # plot the R.hat results

  plot(R.hat, 1:M, xlim=c(1,2), ylim=c(0.5,M+0.5), bty="n", pch=16, yaxp=c(1,M,M-1),
       xlab="R.hat", ylab="Parameter")
  abline(v=1.1, lty=2)
  axis(3)

  # for each parameter theta[1]...theta[M]
  # calculate and plot the 80% intervals from each chain

  plot(rep(0,M+1), 1:(M+1), ylim=c(0.5,M+0.5), xlim=c(-2,2), type="n", bty="n",
       ylab="Parameter", xlab="80% region (scaled)", yaxp=c(1,M,M-1))
  axis(3)

  for (m in 1:M) {
    tj <- theta[m,,]
    intv <- apply(tj, 2, quantile, prob=c(0.1,0.9))
    ci0 <- intv[1,]
    ci1 <- intv[2,]
    mt <- apply(tj, 2, mean)
    scale <- mean(ci1-ci0)/2
    offset <- mean(mt)
    ci0 <- (ci0 - offset)/scale
    ci1 <- (ci1 - offset)/scale
    for (j in 1:J) {
      x <- m + (j-1)/J/4
      segments(ci0[j], x,ci1[j], x, col=j)
    }
  }

}
```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()

## R version 3.2.2 (2015-08-14)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 8 x64 (build 9200)
##
## locale:
## [1] LC_COLLATE=English_United Kingdom.1252  LC_CTYPE=English_United Kingdom.1252
## [3] LC_MONETARY=English_United Kingdom.1252 LC_NUMERIC=C
## [5] LC_TIME=English_United Kingdom.1252
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.12.3
##
## loaded via a namespace (and not attached):
## [1] magrittr_1.5  formatR_1.2.1 tools_3.2.2   stringi_1.0-1 highr_0.5.1   stringr_1.0.0
## [7] evaluate_0.8
```

```
Sys.time()
```

```
## [1] "2016-07-13 08:17:17 BST"
```