Gibbs sampler for missing data: random rounding

Patrick Graham

October 2021

Background

Statistical agencies are concerned to protect the confidentiality of respondents' data. One technique used by Statistics New Zealand is to randomly round cell counts in tables that are either published or specially requested by users. The random rounding algorithm (RR3) is as follows:

1. If a count is a multiple of 3 it is left unchanged;
2. If a count is not a multiple of 3 it rounded to the nearest multiple of 3 with probability 2/3 and to the second nearest multiple of 3 with probability 1/3

Thus an observed count of 7 is rounded to 6 with probability 2/3 and to 4 with probability 1/3. Observed counts of (0,3,6,…) are left unchanged.

Given an observed count Y the random rounding induces a distribution on the non-negative integers that are multiples of three, but this distribution has positive probability mass on, at most, two of these multiples of 3. An unusual distribution.

Since a user of tabular data produced by Stats NZ will see only randomly rounded counts, inference given these counts can be viewed as a missing data problem where the missing data are the true counts. We can use a Gibbs sampler to solve this inference problem by alternately simulating the true counts, given the randomly rounded counts, and simulating the conditional posterior of underlying model parameters given the true counts.

Setting up the model

Let $\mathbf{Y} = (Y_1, Y_2, \dots, Y_k)$ denote a set of counts and $\mathbf{R}$ the corresponding set of randomly rounded counts; e.g we might observe $\mathbf{R} = (0, 0, 3, 6, 3, 0, 6, 3, 0, 6, 0)$.

Suppose that our model for the underlying counts is a simple Poisson model with a common expected value for each cell.

$$\begin{aligned} Y_i|\theta &\overset{\text{indep}}{\sim} \text{Poisson}(\theta);\ , \ i = 1, \dots, k \\ \theta &\sim \text{Gamma}(\alpha, \beta) \end{aligned}$$

However, because we observe only the randomly rounded counts, our full model for the *observed* data is

$$\begin{aligned} R_i|Y_i, \theta &\overset{\text{indep}}{\sim} RR3(Y_i);\ i = 1, \dots, k \\ Y_i|\theta &\overset{\text{indep}}{\sim} \text{Poisson}(\theta);\ , \ i = 1, \dots, k \\ \theta &\sim \text{Gamma}(\alpha, \beta) \end{aligned}$$

where $RR3$ is the distribution induced by the random rounding algorithm. Note the $RR3$ distribution does not depend on θ as indicated by the fact that θ does not appear on the right hand side of the first line in the equation above.

Some useful points to note are:

1. If $R = 0$ the only possible Y values are $(0, 1, 2)$
2. If $R = r > 0$ the only possible Y values are $(r - 2, r - 1, r, r + 1, r + 2)$

Moreover given an observed randomly rounded count we can easily evaluate the probabilities for each possible Y value, by Bayes theorem.

$$\begin{aligned}\Pr(Y = y|R = r, \theta) &= \frac{\Pr(R = r|Y = y)\Pr(Y = y|\theta)}{\sum_a \Pr(R = r|Y = a)\Pr(Y = a|\theta)} \\ &= \frac{\Pr(R = r|Y = y)\text{Poisson}(y|\theta)}{\sum_a \Pr(R = r|Y = a)\text{Poisson}(a|\theta)}\end{aligned}$$

The summation in the denominator above is over the very small set of Y values (at most 5) that are possible given the observed R value. Given our Poisson model for Y we can directly calculate the posterior (to R) probabilities for each possible value of Y. It is therefore straightforward to simulate Y values from the posterior distribution given R. For example we could just use the sample() function to sample from the possible Y values with probabilities for each possible value computed as above. In fact we really need to compute only the numerator value - unnormalised posterior, as sample() samples with probabilities proportional to the specified values.

Gibbs- sampler

Our main interest is the posterior for θ, $p(\theta|\mathbf{R})$. However since θ is the parameter of the model for the cell counts $\mathbf{Y}$, we ¨extend the conversation to include $\mathbf{Y}$" by noting

$$p(\theta|\mathbf{R}) = \int p(\theta, \mathbf{Y}|\mathbf{R})\text{d}\mathbf{Y}$$

In practice we therefore need to obtain $p(\theta, \mathbf{Y}|\mathbf{R})$. A Gibbs sampler for approximating this joint posterior alternates between drawing from

1. $p(\theta|\mathbf{Y}, \mathbf{R}) = p(\theta|\mathbf{Y})$
2. $p(\mathbf{Y}|\theta, \mathbf{R})$

Under our model (1) is straightforward since by the usual conjugacy results this amounts to simulating from a $\text{Gamma}(\alpha + \sum_i Y_i, \beta + k)$ distribution. Step (2) of the Gibbs sampler can be implemented by direct simulation using the probabilities $\Pr(Y = y|R = r)$ as outlined above.

Updating functions

The Gibbs sampler alternates by drawing from (1) $p(\theta|\mathbf{Y})$ and (2) and drawing imputed $\mathbf{Y}$ values from $p(\mathbf{Y}|\mathbf{R}, \theta)$.

First we define a function for updating θ assuming a gamma prior with prior parameters α and β. The function is just a call to rgamma

```
update_theta <-function(Y,alpha,beta) {
  theta <- rgamma(n=1,shape=alpha+sum(Y),rate=beta+length(Y))
return(theta)
}
```

Next we define functions to simulate the $\mathbf{Y}$ values given $\mathbf{R}$ and θ. First we define a function to compute $\Pr(R = r|Y = y)$ for all y values compatible with and observed R value

```
RRprobs <- function(R) {
  if (R > 0) {
    y <- seq(from=R-2,to=R +2)
    pRy <- c(1/3,2/3,1,2/3,1/3)
         }
else { #R=0
  y <- c(0,1,2)
  pRy <- c(1,2/3,1/3)

}
return(cbind(y,pRy))
  }

test3 <- RRprobs(3)
test3
```

```
##      y       pRy
## [1,] 1 0.3333333
## [2,] 2 0.6666667
## [3,] 3 1.0000000
## [4,] 4 0.6666667
## [5,] 5 0.3333333
```

```
test0  <- RRprobs(0)
test0
```

```
##      y       pRy
## [1,] 0 1.0000000
## [2,] 1 0.6666667
## [3,] 2 0.3333333
```

We can now define the function to simulate the $\mathbf{Y}$ values - this will call RRprobs().

```
Yimpute_poisson <- function(R,theta){
  ##vector of R randomly rounded counts
  ##Yimpute is the
  ##count theta is the assumed common success probability among
  ##these trials.

  ##Could try and vectorize RRprobs but we will just loop over the
  ##elements of R

  Y <- vector(length=length(R),mode="integer")
  for (i in 1:length(R)) {
    ##obtain possible Y values and RR3 probabilities - the likelihood for this case
    Yprobs <- RRprobs(R[i])
    Yposs <- Yprobs[,1] #possible Y values
    pRy <- Yprobs[,2]   # likelihood for R at each possible value of Y

    ##get Poisson probabilities for the possible Y values
    priory <- dpois(Yposs,lambda=theta)
    likbyprior <- pRy*priory   ##unnormalized posterior pmf we could    normalize but
    ##the sample function is happy with unnormalized pmfs
    Y[i] <- sample(Yposs,size=1,prob=likbyprior)
  }
  return(Y)
}


##test the functions
testR <- c(0,3,6)
Yimpute_poisson(testR,theta=1)
```

```
## [1] 0 3 4
```

```
Y <- Yimpute_poisson(R=testR,theta=1)
Y
```

```
## [1] 0 1 6
```

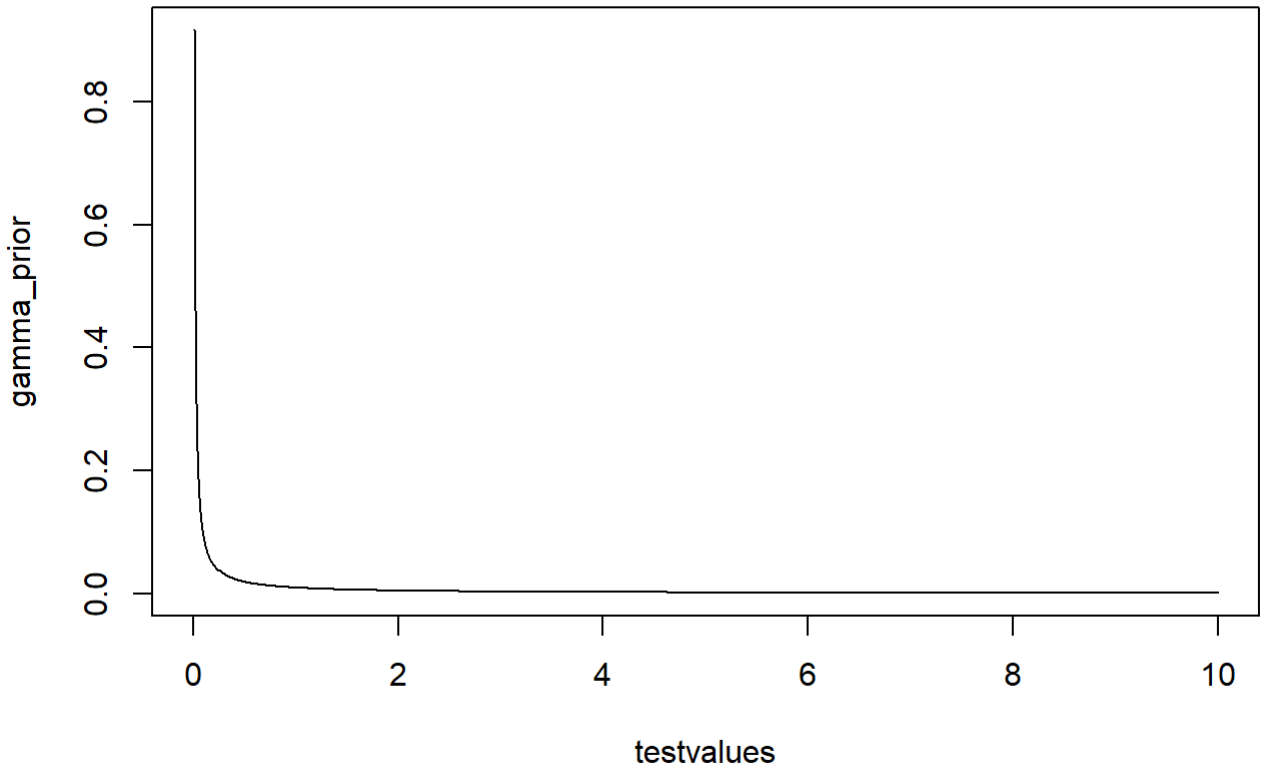
```
testtheta <- update_theta(Y=Y,alpha=1,beta=1)
testtheta
```

```
## [1] 1.043106
```

Gibbs sampler

####Set-up

```
##the observed data
R <- c(0,0,3,6,3,0,6,3,0,6,0,3)  ##These are randomly rounded
##counts from 12 cells;

nchains <- 5
nsim <- 2000
burnin <- 1000

##Structures for storing  simulated values
theta_store <- matrix(NA,nrow=nsim,ncol=nchains)
Ystore <- array(NA,dim=c(length(R),nsim,nchains))

##specify parameters of gamma  prior for theta
##Question specifies alpha=1, beta=1

alpha <- 0.01
beta <- 0.01

##Check  what this looks like
testvalues <- seq(from=0,to=10,by=0.01)
gamma_prior <- dgamma(testvalues,shape=alpha,rate=beta)
plot(testvalues,gamma_prior,type="l")
```



Run the Gibbs sampler

```
for (j in 1:nchains) {
  ##draw an initial value for theta. We draw from an over-dispersed
  #approximation to the
  ##posterior. We base this approximation on the posterior that would
  #be obtained if the R values
  #were the actual Y values but we use only  half the actual number of values
  ##so  we use the first 6 R values as though they were Y values
  newtheta <- rgamma(n=1,shape=alpha+sum(R[1:6]),
                     rate=beta+length(R[1:6]))
  for (i in 1:nsim) {
   # update Y
  newY <-  Yimpute_poisson(R=R,theta=newtheta)
  newtheta <- update_theta(Y=newY,alpha=alpha,beta=beta)

  Ystore[,i,j] <- newY
  theta_store[i,j] <- newtheta

 }  ##end loop over  simulations
} #end loop over chains
```

Check convergence

Define a function to do convergence and effective sample size checking using the Gelman-Rubin split chains approach

```
 GRchunk <-  function(post){
  require(coda)

  possize <- nrow(post)

  n1 <- round(possize/2)  ##size of first chunk

  chunk1 <- post[1:n1,]
  chunk2 <- post[(round(possize/2)+1):possize,]

  post_chunked <- cbind(chunk1,chunk2)

    ##obtain the chain means
  chain_mean <- colMeans(post)
  ##obtain the within chain variance
  chain_sd <- apply(post,MARGIN=2,FUN=sd)
  chain_var <- chain_sd^2
  W = mean(chain_var)
  ##get between chain variance

  possize_chunked <- nrow(post_chunked)
  B <- possize_chunked*(sd(chain_mean))^2

  ##Now build the components of the Gelman-Rubin statistic

  Vplus <- ((possize_chunked-1)/possize_chunked) * W + (1/possize_chunked)*B
  Rhat <- sqrt(Vplus/W)
  Rhat

  ##compute  estimated effective sample size, using Coda

  post.mcmc <- coda::as.mcmc(post_chunked )

  codaNeff_chains  <- coda::effectiveSize(post.mcmc)

  codaNeff <- sum(codaNeff_chains)

  outlist <- list(Rhat=Rhat,codaNeff=codaNeff,codaNeff_chains=codaNeff_chains)
  return(outlist)
}
```

Check convergence for θ

```
 postheta <- theta_store[(burnin+1):nsim,]
posY <- Ystore[,(burnin+1):nsim,]

possize <- nsim - burnin

GRtheta <- GRchunk(postheta)
```

```
## Loading required package: coda
```

```
GRtheta
```

```
## $Rhat
## [1] 0.9994041
##
## $codaNeff
## [1] 2783.266
##
## $codaNeff_chains
##     var1     var2     var3     var4     var5     var6     var7     var8
## 283.6742 221.2449 300.5721 257.4455 280.8503 287.9361 297.1881 297.9748
##     var9    var10
## 285.9517 270.4280
```

Check convergence of Y's

```
str(posY)
```

```
##  num [1:12, 1:1000, 1:5] 1 1 1 6 2 1 7 2 1 4 ...
```

```
 #We will loop over the posterior  samples for  each of 12 cells extracting the $\hat
{R}$ and Neff values.
 #Set-up  structures for  storing the values.
 RhatY <- vector(length=length(R),mode="numeric")
 NeffY <- vector(length=length(R),mode="numeric")

 for (i in 1:length(R)) {
   postY <- posY[i,,]
   GRpostY <- GRchunk(postY)
   RhatY[i] <- GRpostY$Rhat
   NeffY[i] <- GRpostY$codaNeff
 }

 RhatY
```

```
##  [1] 0.9993558 0.9992224 1.0001400 0.9991890 0.9999582 1.0001919 0.9998094
##  [8] 0.9991818 0.9991458 0.9996396 0.9992984 0.9998293
```

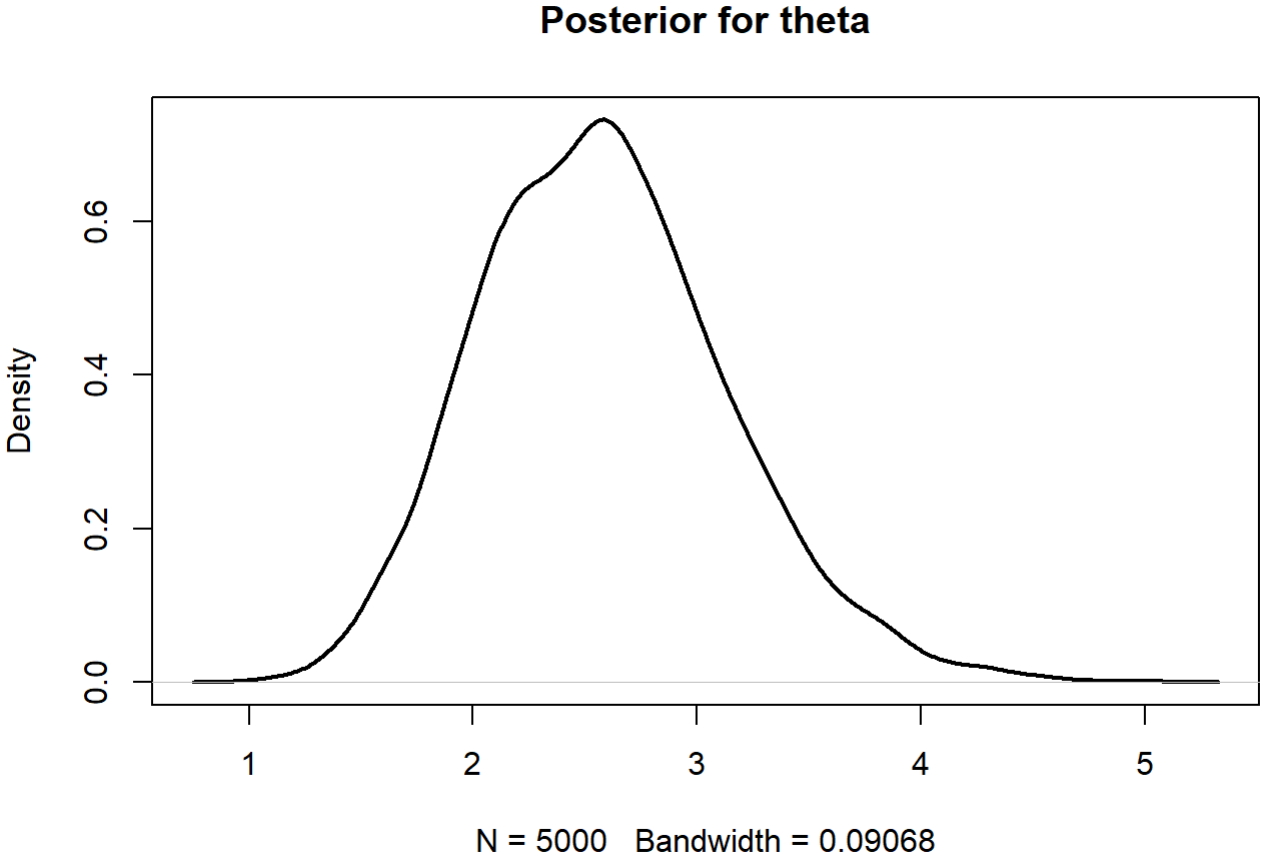
```
 NeffY
```

```
##  [1] 4838.190 4899.516 4931.166 4702.286 4746.767 4996.341 4829.146 4780.166
##  [9] 4672.349 4560.586 4585.306 4726.874
```

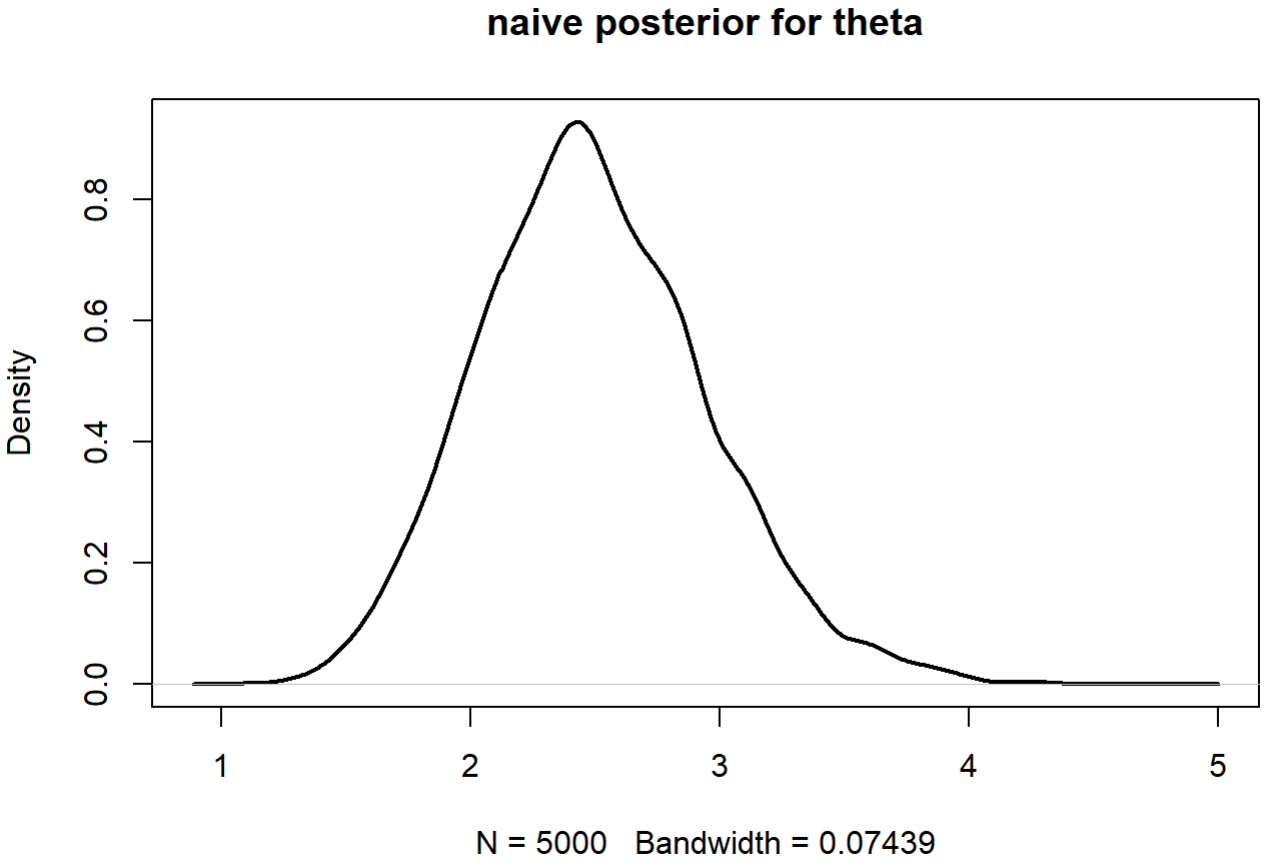
Produce posterior summaries

It looks like the sampler has converged so produce some posterior summaries

```
plot(density(postheta),main="Posterior for theta",lwd=2)
```




```
###Compare with a density plot for a simulation of the same size for
###a naive analysis when the
###randomly rounded counts are treated as actual  counts
plot(density(rgamma(n=5000,shape=alpha+sum(R),rate=beta+length(R))),
     main="naive posterior for theta",lwd=2)
```




```
summary((postheta))  ##get summaries by chain
```

```
##        V1              V2              V3              V4
##  Min.   :1.072   Min.   :1.023   Min.   :1.213   Min.   :1.126
##  1st Qu.:2.147   1st Qu.:2.201   1st Qu.:2.207   1st Qu.:2.184
##  Median :2.528   Median :2.575   Median :2.567   Median :2.553
##  Mean   :2.564   Mean   :2.600   Mean   :2.595   Mean   :2.575
##  3rd Qu.:2.917   3rd Qu.:2.954   3rd Qu.:2.922   3rd Qu.:2.901
##  Max.   :4.863   Max.   :4.468   Max.   :4.493   Max.   :5.055
##        V5
##  Min.   :1.160
##  1st Qu.:2.196
##  Median :2.558
##  Mean   :2.597
##  3rd Qu.:2.973
##  Max.   :4.782
```

```
##Obtain desired quantiles for 90% credible intervals and median
qprobs <- c(0.05,0.5,0.95)

quant_theta <- quantile(postheta,probs=qprobs)
quant_theta
```

```
##       5%      50%      95%
## 1.740068 2.557226 3.558667
```

```
##What would the quantiles be for the naive analysis treating the R values as though
 they were
###the actual counts. There is no justification for such an analysis but it is the wa
y randomly
###rounded counts are usually handled.

quant_naive <- qgamma(qprobs,shape=alpha+sum(R),rate=beta+length(R))
quant_naive
```

```
## [1] 1.798707 2.471052 3.293296
```

```
quant_theta
```

```
##       5%      50%      95%
## 1.740068 2.557226 3.558667
```

Not a huge difference here, but our interval is wider than the naive interval because we correctly account for the extra random-ness induced by the random-rounding.

The approach could clearly be extended to bigger tables of counts and more complex models.