Generating constituency-level opinion, with code

HLV

November 2014

1 Introduction

This document describes code used to generate estimates of opinion at the constituency level. It is distributed as part of a zip archive that contains some of the files needed to generate these estimates. The document is built with Knitr, an example of literate programming, such that the R source code used to generate these estimates is embedded within the document itself. This R code can be found at `replication_howto.R`.

2 If life gives you a chainsaw…

We've written this document so that other researchers can estimate opinion at the constituency level without investing substantial effort in sourcing, reconciling and recoding census and survey data. We've used this code (or versions of it) to estimate constituency-level opinion on Europe, on immigration, on taxation, on crime, and on other issues.

If you want to understand this code, you should read the accompanying technical report. The technical report describes what we're doing: the code below describes how we do it.

The technical report gives details on how individual level predictors and post-stratification can be combined with spatial data and constituency-level predictors

to produce estimates of opinions. For the "political" opinions we've studied so far, demographic predictors make sense, and explain a reasonable proportion of the individual level variance in the relevant opinion.

However, there are some circumstances where we would caution against using this code to estimate opinion at the constituency level.

First, don't use this code if the opinion you're interested in is not obviously related to individual respondents' occupation or education. It would be wrong, for example, to estimate constituency level opinion concerning favourite Shakespeare plays, for example.

Second, don't use this code if the opinion you're interested in is, by its very nature, local. When we estimate responses towards Europe (as we do in this document), there is a common stimulus to which people are reacting. Our modelling exploits commonalities in responses amongst people who live in different places but who are alike in their characteristics. If, however, we were to estimate opinion concerning satisfaction with local councils, there would be no such commonality.

3 Dependencies

Estimating constituency level opinion is a complex endeavour, which depends upon several pieces of freely available software. We begin by setting out the software dependencies you need, before moving on to discuss the data dependencies.

3.1 Software dependencies

First, you'll need a recent version of R. This document was created using the following R version:

```
R.Version()$version.string
```

```
[1] "R version 3.1.1 (2014-07-10)"
```

and the following random seed:

```
set.seed(2511)
```

Second, you'll need a version of WinBUGS. You need WinBUGS rather than an other Gibbs sampler (like, say, JAGS) because, to the best of our knowledge, only WinBUGS can draw from a conditional autoregressive normal distribution. If you're not running a Windows computer, you should know that WinBUGS can be run through WINE on Linux, and through Parallels on Macs.

Third, you'll need a number of R libraries. This document requires the following libraries:

```
library(foreign)
library(memisc)
library(car)
library(maptools)
library(spdep)
library(R2WinBUGS)
library(arm)
library(ggplot2)
library(plyr)
library(rgeos)
```

Not all of these libraries are strictly necessary to generate constituency-level estimates. Some – like `car` – are used only for helping in recoding variables.

3.2 Data dependencies

Moving on to the data dependencies, you'll first need a data-set with responses from several thousand individuals. This data-set must contain information on (a) an opinion of interest, (b) the constituency of each respondent; and (c) additional individual-level variables which can be matched with the information collected by the Census. In this document, we use Wave 2 of the British Election Study 2015. We cannot redistribute this data, so you will need to download it and change the `bespath` variable to read the data into R. Additionally, we're going to drop three-quarters of the observations to reduce the computational burden.

```
bespath <- "~/Dropbox/constituency-estimates-data/bes-2015/BES2015_W2_Panel_v2.0.sav"
besdat <- read.spss(bespath,
```

```
    to.data.frame=TRUE)
nrow(besdat)
besdat <- besdat[sample(1:nrow(besdat),nrow(besdat)/4),]
nrow(besdat)
```

Second, you'll need some constituency-level variables and the necessary look-up codes to link these variables to the BES data. These are contained in the zipped folder.

```
auxpath <- "./canonical_seatvars_2011census.csv"
lookuppath <- "./name2pa.csv"
aux <- read.csv(auxpath)
lookup <- read.csv(lookuppath)
```

Third, you'll need some post-stratification weights. The weights we currently use are transitional weights. We use the sample of anonymised records (SAR) from the 2001 census, and rake this to the marginal distributions from the 2011 census. Once the SAR from the 2011 census is available, we can use information from the 2011 census exclusively.

```
pswpath <- "canonical_weights_2001SARS_2011margins_flatfile.csv"
psw <- read.csv(pswpath)
```

Fourth, you'll need some shapefiles giving details of the boundaries of the parliamentary constituencies used in Great Britain. Once again, we cannot redistribute this data, as it is held by the Ordnance Survey. Additionally, the Boundary Line data file is quite large. You will need to download this data file yourself, and change the variable geopath in order to read this into R.

```
geopath <- "~/Dropbox/constituency-estimates-data/westminster_const_region.shp"
geo <- readShapePoly(geopath)
```

You will also need some manual additions – instances of constituencies which are either genuine islands, or which appear such because of faults in the shapefile.

```
joinpath <- "./joins_to_add.csv"
joins.to.add <- read.csv(joinpath, header = TRUE)
```

4 Recoding

Unfortunately, a lot of work goes into recoding variables in order to make sure that the different sources of data play together nicely. We start with the BES data, before moving on to the geodata.

4.1 BES re-codes

4.1.1 Dependent variable

We start with our dependent variable – the opinion of interest. Here, we're going to examine how respondents would vote in a hypothetical referendum on EU membership. This is stored in the BES as a variable called `euRefVoteW2`. The possible values of this variable are as follows:

- 1 (Leave the EU)
- 0 (Stay in the EU)
- 2 (I would not vote)
- 9999 (Don't know)

We will recode these values into a new variable called `y` (a more descriptive variable name would normally be preferable, but by calling the variable y we make this code more reusable). This variable will have value 1 if the respondent would vote to leave the EU.

```
besdat$y <- as.numeric(as.numeric(besdat$euRefVoteW2) == 2)
besdat$y[as.numeric(besdat$euRefVoteW2) > 2] <- NA
```

4.1.2 Individual-level variables

We now embark on a lengthy series of recodes designed to match the BES data with the Census weights. This involves recoding variables for age, gender, marital status, housing tenure, highest level of education, and private sector occupation. If any of these recoding operations are unclear, we recommend consulting the BES codebook.

```
# Gender
besdat$gender <- factor(besdat$gender,exclude=c("Skipped","Not Asked"))

# Age
besdat$ageGroup <- 8 - ((besdat$Age <= 19) +
    (besdat$Age <= 24) +
    (besdat$Age <= 29) +
    (besdat$Age <= 44) +
    (besdat$Age <= 59) +
    (besdat$Age <= 64) +
    (besdat$Age <= 74))
besdat$ageGroup <- factor(besdat$ageGroup,
    labels=c("16-19", "20-24","25-29","30-44","45-59","60-64","65-74","75+"))
besdat$ageGroup <- replace(besdat$ageGroup,
    is.na(besdat$ageGroup),
    "45-59") # assign NAs to most frequent category

# Marital status
besdat$maritalStatus <- car:::recode(besdat$marital,
    "c('Married')='Married or re-married';
    c('Skipped','Not Asked')=NA;
    else='Single (never married), separated, divorced or widowed'")

## Housing tenure
besdat$housing <- car:::recode(besdat$housing,
    "c('Own the leasehold/freehold outright',
        'Buying leasehold/freehold on a mortgage')='Owns';
    c('Rented from local authority',
        'Rented from private landlord',
        'It belongs to a Housing Association')='Rents';
    else = 'Rents'")

## Education
```

```
besdat$qualifications <- car:::recode(besdat$education,
    "'No formal qualifications'='No qualifications';
    'Youth training certificate/skillseekers'='Level 2';
    'Recognised trade apprenticeship completed'='Level 2';
    'Clerical and commercial'='Level 1';
    'City and Guild certificate'='Level 1';
    'City and Guild certificate - advanced'='Level 2';
    'onc'='Level 2';
    'CSE grades 2-5'='Level 1';
    'CSE grade 1, GCE O level, GCSE, School Certificate'='Level 2';
    'Scottish Ordinary/ Lower Certificate'='Level 2';
    'GCE A level or Higher Certificate'='Level 3';
    'Scottish Higher Certificate'='Level 3';
    'Nursing qualification (eg SEN, SRN, SCM, RGN)'='Level 4/5';
    'Teaching qualification (not degree)'='Level 4/5';
    'University diploma'='Level 4/5';
    'University or CNAA first degree (eg BA, B.Sc, B.Ed)'='Level 4/5';
    'University or CNAA higher degree (eg M.Sc, Ph.D)'='Level 4/5';
    'Other technical, professional or higher qualification'='Other';else = NA")

## Private sector occupation
besdat$privateSector <- car:::recode(besdat$work_type,
    "c(1,2)='Private';c(8,9)=NA;else='Public'")
```

Having recoded these variables, we now drop all rows in the data without a response to the question on EU exit.

```
besdat <- subset(besdat,!is.na(besdat$y))
```

We also drop all rows in the data with missing values for any of our individual level variables. Normally, such a complete-cases strategy would be ill-advisable, but since these individual-level variables are fairly common, the rates of missingness are low, and we can concentrate on complete cases without too much loss of data.

```
tokeepvars <- c("gender","ageGroup","maritalStatus",
    "housing","qualifications","privateSector")
tokeep <- complete.cases(besdat[,tokeepvars])
besdat <- besdat[tokeep,]
rm(tokeep, tokeepvars)
```

Finally, we make sure that the reference codes for the constituency match across data-sets. Because we will eventually need to refer to the constituencies in WinBUGS, we create a continuously numbered version of the constituency reference code. This differs from the Press Association reference codes, which include Northern Irish constituencies which we omit.

```
besdat$refno <- besdat$pconrefno
besdat <- merge(besdat,lookup,
    by="refno",
    all.x=T,all.y=F)
besdat <- subset(besdat, !is.na(besdat$refno))

# Get constituency identifier
# PA numbers are non-consecutive
# So make sure we take number of levels from the factor
besdat$constindex <- as.numeric(factor(besdat$refno))
constindex <- besdat$constindex

## Create look-up table
const.lookup <- data.frame(refno=unique(besdat$refno),
  refno.num = unique(constindex))
```

4.2 Geo-data recodes

We begin by ensuring a match between the codes by which the individual polygons in the shapefile are referred to, and the codes we are using. Because it is possible that the shapefile includes more mainland constituencies than are found in the data (for example, if one constituency was unfortunate enough never to be included in the sample), we must add these on to our lookup list.

```
geo$refno <- lookup$refno[match(as.character(geo$NAME),lookup$ShapeFileName)]

### Index this in the same way as the continuous numbering
geo$refno.num <- NA
geo$refno.num <- const.lookup$refno.num[match(geo$refno,const.lookup$refno)]

# Find additional constituencies not present in the lookup table
extras <- geo$refno[is.na(geo$refno.num)]
extras.num <- seq(from=(max(geo$refno.num,na.rm=T)+1),
  by=1,
  length.out=length(extras))
extras <- data.frame(refno=extras,refno.num=extras.num)
rm(extras.num)

## Add to lookup table
const.lookup <- rbind(const.lookup,extras)

## And now repeat the match,
## Overwriting refno in the process
geo$refno <- const.lookup$refno.num[match(geo$refno,const.lookup$refno)]
geo$refno.num <- NULL
```

We now transform the data into a slightly different R object (a Spatial Polygons data frame), taking care to preserve the order of the polygons.

```
gpclibPermit()
sp.const <- unionSpatialPolygons(geo, geo$refno)
# Lose detail
sp.const <- gSimplify(sp.const,100)

dist.size <- as.data.frame(cbind(geo$HECTARES, geo$refno))
colnames(dist.size) <- c("area","refno")
dist.size<-dist.size[order(dist.size$refno),]
rownames(dist.size)<-dist.size$refno
```

```
area.spdf <- SpatialPolygonsDataFrame(sp.const, dist.size)

## Reorder according to refno
area.spdf <- area.spdf[ order(area.spdf$refno),]
```

We now create the adjacency matrices. Here, we must add the joins we mentioned previously.

```
nb.districts <- poly2nb(area.spdf)

## Join islands to the mainland
for (i in 1:nrow(joins.to.add)) {
    a <- as.integer(const.lookup$refno.num[which(const.lookup$refno==joins.to.add[i,1
    b <- as.integer(const.lookup$refno.num[which(const.lookup$refno==joins.to.add[i,2
    nb.districts[[a]] <- c(nb.districts[[a]],b)
    nb.districts[[a]] <-nb.districts[[a]][nb.districts[[a]]!=0]
    nb.districts[[b]] <- c(nb.districts[[b]],a)
    nb.districts[[b]] <-nb.districts[[b]][nb.districts[[b]]!=0]
}

nb <- unlist(nb.districts)
weight <- rep(1, times=length(nb))
num <- card(nb.districts)

spdata <- list("nb", "weight", "num")
```

4.3 Constituency-level information

We now recode information pertaining to the constituency level. In some instances, this involves mere renaming:

```
### Some renaming
for (i in c("private","female","married","owns","education","socgrd","age")) {
```

```
    names(aux)[names(aux) == i] <- paste0(i,"SL")
}

# transform lagged vote shares for 3 main parties
aux$lab10[is.na(aux$lab10)] <- 1
aux$con10[is.na(aux$con10)] <- 1
aux$ld10[is.na(aux$ld10)] <- 1
aux$lab10logit <- car:::logit(aux$lab10,percents = TRUE)
aux$con10logit <- car:::logit(aux$con10,percents = TRUE)
aux$ld10logit <- car:::logit(aux$ld10,percents = TRUE)
```

In other cases, there is more heavy lifting to do. Here, we construct constituency-level aggregates for age and education from our census weights.

```
education <- aggregate(psw$weight,
    list(refno=psw$parliamentary.constituency.2010,education=psw$education),
    sum,na.rm=T)

education <- ddply(education,.(refno),function(df){
    data.frame(educlevel4plus = sum(df[df$education %in% c("Level 4/5"),"x"],na.rm = TRUE),
    educnoqual = sum(df[df$education %in% c("No qualifications"),"x"],na.rm = TRUE))
})

aux <- merge(aux, education, by = "refno", all.x=TRUE)

ageGroup <- aggregate(psw$weight,
    list(refno=psw$parliamentary.constituency.2010,ageGroup=psw$age0),
    sum,na.rm=T)
ageGroup <- ddply(ageGroup,.(refno),function(df){
    data.frame(age16to24 = sum(df[df$ageGroup %in% c("16-19","20-24"),"x"],na.rm = TRUE),
               age65plus = sum(df[df$ageGroup %in% c("65-74","75+"),"x"],na.rm = TRUE))
})

aux <- merge(aux, ageGroup, by = "refno", all.x=TRUE)
```

```
rm(education,ageGroup)
```

For housekeeping purposes, we'll merge this auxiliary information with the BES data-frame, making sure only to include the relevant constituencies. Note how we scale the variables so that they are approximately normally distributed. Scaling variables is often a necessary step in MCMC sampling so as to improve sampling speed and avoid over/under-run errors.

```
aux <- subset(aux,aux$refno %in% unique(const.lookup$refno))
aux <- merge(aux,const.lookup,all.x=T,all.y=F)
## Order aux seat correctly for later use
aux <- aux[order(aux$refno.num),]
## Scale variables
vars.to.scale <- c("con10","lab10","ld10",
    "log.earn","nonwhite","relig.christian",
    "relig.refused","relig.none","relig.other",
    "ageSL","privateSL","ownsSL","femaleSL",
    "marriedSL","educationSL","socgrdSL",
    "log.density","lab10logit","con10logit","ld10logit",
    "educlevel4plus","educnoqual",
    "age16to24","age65plus")

for (v in vars.to.scale) {
    aux[,v] <- as.vector(scale(aux[,v]))
}
besdat <- merge(besdat, aux,
    by.x = "refno", by.y = "refno",
    all.x = T,
    suffixes = c(".x",""))

constindex <- besdat$constindex
besdat$region.num <- as.numeric(besdat$gor)
```

5 The model

Having recoded much of our data, we can now turn to the WinBUGS model. This model loops over the individual observations, pulling into the constituency-level effects, which are modelled respectively as a function of constituency-level covariates and a series of spatially correlated random effects. Here, we describe the model as an R language function, which is then written out to a file.

```
model <- function() {
    for (i in 1:nObs) {
        y[i] ~ dbern(p[i])

        logit(p[i]) <- alpha +
            gamma[i] +
            beta[constindex[i]] +
            v[constindex[i]]

        gamma[i] <- gamma.female*female[i] + gamma.rent*rent[i] +
                    gamma.notMarried*notMarried[i] +
                    gamma.private*private[i] +
                    gamma.ageGroup[ageGroup[i]] +
                    gamma.qualifications[qualifications[i]]
    }

    gamma.female ~ dnorm(0, .0001)
    gamma.rent ~ dnorm(0, .0001)
    gamma.notMarried ~ dnorm(0, .0001)
    gamma.private ~ dnorm(0, .0001)

    for(k in 1:nAgeGroups){
       gamma.ageGroup[k] ~ dnorm(0, tau.ageGroup)
      }
    for(k in 1:nQualifications){
       gamma.qualifications[k] ~ dnorm(0, tau.qualifications)
```

```
  }


tau.ageGroup  <- pow(sigma.ageGroup, -2)
sigma.ageGroup ~ dunif(0, 2)
tau.qualifications  <- pow(sigma.qualifications, -2)
sigma.qualifications ~ dunif(0, 2)

for (j in 1:nConst) {
    beta[j] <- beta.density * density[j]  +
                beta.nonwhite * nonwhite[j]  +
                beta.earnings * earnings[j] +
                beta.religchristian * relig.christian[j] +
                beta.religother * relig.other[j] +
                beta.religrefuse * relig.refuse[j] +
                beta.femaleSL * femaleSL[j] +
                beta.marriedSL * marriedSL[j] +
                beta.privateSL * privateSL[j] +
                beta.ownsSL * ownsSL[j] +
                beta.socgrdSL * socgrdSL[j] +
                beta.age16to24 * age16to24[j] + beta.age65plus * age65plus[j] +
                beta.educnoqual * educnoqual[j] +
                beta.educlevel4plus * educlevel4plus[j] +
                beta.con10 * con10[j] +
                beta.lab10 * lab10[j] +
                beta.ld10 * ld10[j] +
                beta.region[region[j]] +
                u[j]
    u[j] ~ dnorm(0, tauu)
}

for(k in 1:nRegion){
    beta.region[k] ~ dnorm(0, tau.region)
```

```
    }

    v[1:nConst] ~ car.normal(nb[], weight[], num[], tauv)
    alpha ~ dflat()
    beta.density ~ dflat()
    beta.nonwhite ~ dflat()
    beta.earnings ~ dflat()
    beta.religchristian ~ dflat()
    beta.religother ~ dflat()
    beta.religrefuse ~ dflat()
    beta.femaleSL ~ dflat()
    beta.marriedSL ~ dflat()
    beta.privateSL  ~ dflat()
    beta.ownsSL  ~ dflat()
    beta.socgrdSL ~ dflat()
    beta.age16to24 ~ dflat()
    beta.age65plus ~ dflat()
    beta.educnoqual ~ dflat()
    beta.educlevel4plus ~ dflat()
    beta.con10 ~ dflat()
    beta.lab10 ~ dflat()
    beta.ld10 ~ dflat()

    tau.region <- pow(sigma.region,-2)
    sigma.region ~ dunif(0,2)

    tauu <- pow(sigmasquv*sigmamix, -1)
    tauv <- pow(sigmasquv*(1-sigmamix), -1)
    sigmasquv <- pow(sigmauv,2)
    sigmauv ~ dunif(0, 2)
    sigmamix ~  dbeta(1,1)
}
```

```
write.model(model, "model.bug")
```

Unfortunately, passing data to WinBUGS can be quite tedious. The following chunk of code sets up the various variables, which are then passed by name in the string at the end of the code chunk.

```
# Prepare data for WinBugs
y <- as.numeric(besdat$y)
nObs <- as.vector(length(y))
nConst <- as.numeric(as.vector(max(const.lookup$refno.num)))
constindex <- as.vector(constindex)

female <- as.numeric(as.numeric(besdat$gender == "Female"))
notMarried <- as.numeric(besdat$maritalStatus == "Single (never married), separated,
rent <- as.numeric(besdat$housing == "Rents")
private <- as.numeric(besdat$privateSector == "Private")

nAgeGroups <- length(unique(besdat$ageGroup))
ageGroup <- as.numeric(besdat$ageGroup)
nsocialGrades <- length(unique(besdat$socialGrade))
socialGrade <- as.numeric(besdat$socialGrade)
nQualifications <- length(unique(besdat$qualifications))
qualifications <- as.numeric(besdat$qualifications)

## Constituency level stuff
density <- as.vector(aux$log.density)
earnings <- as.vector(aux$log.earn)
nonwhite <- as.vector(aux$nonwhite)
region <- as.numeric(aux$region)
nRegion <- length(unique(region))
relig.christian <- as.vector(aux$relig.christian)
relig.other <- as.vector(aux$relig.other)
relig.refuse <- as.vector(aux$relig.refused)
```

```
femaleSL <- as.vector(aux$femaleSL)
marriedSL <- as.vector(aux$marriedSL)
privateSL <- as.vector(aux$privateSL)
ownsSL <- as.vector(aux$ownsSL)
socgrdSL <- as.vector(aux$socgrdSL)

age16to24 <- as.vector(aux$age16to24)
age65plus <- as.vector(aux$age65plus)
educnoqual <- as.vector(aux$educnoqual)
educlevel4plus <- as.vector(aux$educlevel4plus)

lab10logit <- as.vector(aux$lab10logit)
con10logit <- as.vector(aux$con10logit)
ld10logit <- as.vector(aux$ld10logit)
lab10 <- as.vector(as.numeric(aux$lab10logit))
con10 <- as.vector(as.numeric(aux$con10logit))
ld10 <- as.vector(as.numeric(aux$ld10logit))


bugsdata <- list("nObs","female","y","rent","notMarried","private",
            "ageGroup","nAgeGroups"
            ,"nQualifications","qualifications"
            ,"nConst"
            ,"density"
            ,"nonwhite","earnings"
            ,"constindex"
            ,"relig.christian","relig.other","relig.refuse"
            ,"femaleSL", "marriedSL", "privateSL"
            ,"ownsSL", "socgrdSL"
            ,"age16to24", "age65plus"
            ,"educnoqual","educlevel4plus"
            ,"region","nRegion"
```

```
        ,"con10","lab10","ld10"
    )
```

WinBUGS will occasionally throw a hissy fit if its own best guess at initial values causes improbable results. For this reason, it's best to pass WinBUGS an initialization function. Again, this can be unwieldy, hence the tongue-in-cheek name of the function, `tiny.inits`. We'll also use this opportunity to set a random seed to ensure reproducibility.

```
set.seed(2511)
tiny.inits <- function() {
            alpha <- rnorm(1,0,1)
            beta.density <- rnorm(1,0,1)
            beta.religrefuse <- rnorm(1,0,1)
            beta.religother <- rnorm(1,0,1)
            beta.religchristian <- rnorm(1,0,1)
            beta.nonwhite <- rnorm(1,0,1)
            beta.earnings <- rnorm(1,0,1)
            beta.femaleSL  <- rnorm(1,0,1)
            beta.marriedSL  <- rnorm(1,0,1)
            beta.privateSL   <- rnorm(1,0,1)
            beta.ownsSL   <- rnorm(1,0,1)
            beta.socgrdSL  <- rnorm(1,0,1)
            beta.age16to24  <- rnorm(1,0,1)
            beta.age65plus  <- rnorm(1,0,1)
            beta.educnoqual  <- rnorm(1,0,1)
            beta.educlevel4plus  <- rnorm(1,0,1)
            beta.con10  <- rnorm(1,0,1)
            beta.lab10  <- rnorm(1,0,1)
            beta.ld10  <- rnorm(1,0,1)

            gamma.female <- rnorm(1,0,1)
            gamma.rent <- rnorm(1,0,1)
            gamma.notMarried <- rnorm(1,0,1)
```

```
    gamma.private <- rnorm(1,0,1)
    gamma.ageGroup <- rnorm(nAgeGroups,0,1)
    gamma.qualifications <- rnorm(nQualifications,0,1)
    u = rep(0,nConst)
    v = rep(0,nConst)

    return(list(alpha=alpha,beta.density=beta.density,
                beta.religrefuse=beta.religrefuse,
                beta.religother=beta.religother,
                beta.religchristian=beta.religchristian,
                beta.nonwhite=beta.nonwhite,beta.earnings=beta.earnings,
                beta.femaleSL=beta.femaleSL, beta.marriedSL=beta.marriedSL,
                beta.privateSL=beta.privateSL, beta.ownsSL=beta.ownsSL,
                beta.socgrdSL=beta.socgrdSL,
                beta.age16to24=beta.age16to24,
                beta.age65plus=beta.age65plus,
                beta.educnoqual=beta.educnoqual,
                beta.educlevel4plus=beta.educlevel4plus,
                beta.con10=beta.con10,
                beta.lab10=beta.lab10,
                beta.ld10=beta.ld10,
                gamma.female=gamma.female,
                gamma.rent=gamma.rent,
                gamma.notMarried= gamma.notMarried,
                gamma.private=gamma.private,
                gamma.ageGroup=gamma.ageGroup
                ,gamma.qualifications=gamma.qualifications
                ,u=u,v=v))
}
```

6 Estimation

We've now re-coded all our data, made it play nicely together, and specified our model. There are two steps left – estimation of the model, and post-stratification. Estimation can be quite computationally intensive – and post-stratification quite memory intensive. For these reasons, we're going to set the number of MCMC samples to a small number.

```
### You could use these
my.iter <- 50000
my.burnin <- 10000
my.thin <- 100

### But we'll use these
my.iter <- 200
my.burnin <- 100
my.thin <- 1
```

Here comes the estimation! You will need to change the Bugs directory to match its location on your system.

```
my.bugs.dir <- "/home/chris/.wine/drive_c/Program Files (x86)/WinBUGS14"

model.sim <- bugs(data=c(bugsdata, spdata),
        inits = tiny.inits,
     model.file="model.bug",
      parameters.to.save=c("alpha",
        "gamma.female",
        "gamma.rent",
        "gamma.notMarried",
        "gamma.private",
        "gamma.ageGroup"
        ,"gamma.qualifications"
        ,"beta","v"
```

```
        ,"sigmauv", "sigmamix", "sigma.region"
        ,"beta.density"
        ,"beta.nonwhite","beta.earnings"
        ,"beta.religrefuse","beta.religother", "beta.religchristian"
        ,"beta.region"
        ,"beta.femaleSL","beta.marriedSL","beta.privateSL","beta.ownsSL","beta.socgrdSL"
        ,"beta.age16to24","beta.age65plus","beta.educnoqual","beta.educlevel4plus"
        ,"beta.con10","beta.lab10","beta.ld10"
        ),
      n.chains=3,
      n.iter=my.iter,
      n.burnin=my.burnin,
      n.thin=my.thin,
      debug=FALSE,
      bugs.directory=my.bugs.dir)
```

We probably want to check whether these Markov Chains have converged:

```
summary(model.sim$summary[,"Rhat"])
```

Min. 1st Qu. Median Mean 3rd Qu. Max. 0.997 1.030 1.070 1.140 1.120 9.430

7 Post-stratification

Assuming that convergence has been achieved, we now need to post-stratify the results. This will involve loading in the post-stratification weights, and doing some book-keeping on the outcome of the WinBUGS model.

```
load("./canonical_weights_2001SARS_2011margins_Robj.RData")

PSW <- as.data.frame.table(psw, rownames = NULL, responseName = "N")
names(PSW) <- c("refno", "gender", "ageGroup", "qualifications",
                "maritalStatus", "housing", "socgrd", "privateSector", "N")
```

```
### Social grade not used here
PSW$socgrd <- NULL
# assign re-formatted PSW as census data
census <- PSW
rm(PSW, psw)

## create census model matrix
census.mat <- model.matrix(~ 1 + I(gender=="Female") + I(housing=="Rents") +
    I(maritalStatus=="Single (never married), separated, divorced or widowed") +
    I(privateSector=="Private") +
        ageGroup + qualifications,
        contrasts.arg = list(ageGroup = contrasts(census$ageGroup, contrasts=F),
            qualifications = contrasts(census$qualifications, contrasts=F)
    ),
    data = census)

## Constituency effects
constituency.component <- model.sim$mean$beta + model.sim$mean$v
constituency.component <- data.frame(constout = constituency.component,
    refno.num = 1:nConst)
constituency.component <- merge(constituency.component,const.lookup)
constituency.component$refno.num <- NULL

## Rearrange coefs
the.coefs <- c(model.sim$mean$alpha,
        model.sim$mean$gamma.female,
        model.sim$mean$gamma.rent,
        model.sim$mean$gamma.notMarried,
        model.sim$mean$gamma.private,
        model.sim$mean$gamma.ageGroup
        ,model.sim$mean$gamma.qualifications
        )
```

```
### Get product

census$out <-  rowSums(census.mat %*% the.coefs)

### Add the constituency effects
census <- merge(census,constituency.component,
        by = "refno",
        all=T)
census$cell.pred <- inv.logit(census$out + census$constout)

## Aggregate to constituency yhat
meanpreds <- census$cell.pred * census$N
meanpreds <- aggregate(meanpreds,list(refno=census$refno),sum,na.rm=T)
names(meanpreds)[2] <- "yhat"

meanpreds$yhat <- meanpreds$yhat*100

meanpreds <- merge(meanpreds, lookup, by = "refno",
    all = TRUE)
```

We should now have our results in `meanpreds`, which can be saved, or merged with other data, or enter into another model, or...

```
meanpreds[1:3,c("refno","YouGovName","yhat")]
```

refno YouGovName yhat 1 1 Aberavon 52.90 2 10 Alyn and Deeside 61.65 3 100 Bristol North West 44.23