```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.72 (7847) x86_64-apple-darwin17.0]

[Workspace restored from /Users/andreasmurr/.RData]
[History restored from /Users/andreasmurr/.Rapp.history]

> setwd('/Users/andreasmurr/Dropbox/PostModernBayes/Replication')
> # Computing quantities of interest and their uncertainty using Bayesian simulation
> # Andreas Murr, Richard Traunmueller, and Jeff Gill
> # Creates Figures 4 and 7 based on union density data
>
> # clear working memory
>
>  rm(list=ls())
>
> # load packages
>
>  library(rjags)
Loading required package: coda
Linked to JAGS 4.2.0
Loaded modules: basemod,bugs
>  library(R2WinBUGS)
Loading required package: boot
>  library(coda)
>  library(pscl)
Classes and Methods for R developed in the
Political Science Computational Laboratory
Department of Political Science
Stanford University
Simon Jackman
hurdle and zeroinfl functions by Achim Zeileis
>
> # load data
>
>  data("unionDensity")
>
> # list data
>
>  N = dim(unionDensity)[1]
>  y = unionDensity$union
>  left = unionDensity$left
>  size = unionDensity$size
>  concen = unionDensity$concen
```

```
>
> # write model
>
>  model <- function(){
+    for(i in 1:N){
+      y[i] ~ dnorm(mu[i], tau)
+      mu[i] <- beta[1] + beta[2]*left[i] + beta[3]*size[i] + beta[4]*concen[i]
+      res[i] <- y[i] - mu[i]   # calculate residuals
+    }
+    beta[1:4] ~ dmnorm(b.0, tau*B.0) # precision matrix needs both tau and B.0
+    tau ~ dgamma(nu.0/2, nu.0*sigma2.0/2)
+    sigma <- 1/sqrt(tau)
+  }
>
>  write.model(model, "simple_linear_informed_res.bug")
>
> # add western and jackman's prior parameters for gamma to data list
>
>  b.0 <- c(0, .3, 0, 10)
>  B.0 <- diag(c(10^-8, 10^4, 10^-8, 9)) # careful: precisions!
>  nu.0 <- 8
>  sigma2.0 <- 169.2
>
>  jags.data <- list(N=N, y=y, left=left, size=size, concen=concen, nu.0=nu.0,
sigma2.0=sigma2.0, b.0=b.0, B.0=B.0)
>
> # pass to jags
>
>  inits = list(".RNG.seed"=514913, ".RNG.name"="base::Mersenne-Twister")
>
>  jags.lm.3 <- jags.model("simple_linear_informed_res.bug", data=jags.data, inits=inits,
n.chains=2)
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100%
>
>  update(jags.lm.3, 1000)
  |**************************************************| 100%
>  mon <- c("beta", "sigma", "res") # add residuals to monitor
>  jags.lm.out.3 <- coda.samples(jags.lm.3, variable.names=mon, n.iter=5000)
  |**************************************************| 100%
>
> # ============
> # = figure 4 =
> # ============
>
> # summarize residuals
>
>  res.mean <- apply(jags.lm.out.3[[1]][,5:24], 2, mean)
>  res.up <- apply(jags.lm.out.3[[1]][,5:24], 2, function(x) quantile(x, .975))
>  res.lo <- apply(jags.lm.out.3[[1]][,5:24], 2, function(x) quantile(x, .025))
>
> # check for Normality
>
>  tq <- qqnorm(res.mean)$x
>
> # plot
>
>  cairo_pdf("figure-4.pdf", width=6, height=3, pointsize=10)
>
>  par(mfrow=c(1,2), mar=c(3,3,2,2), mgp=c(1.5, .5, 0), family="Gill Sans", las=1)
>  # density
>  plot(density(res.mean), col=rgb(0,0,0, .05), ylim=c(0, .06), main="", axes=F,
```

```
xlab="Residuals", ylab="", xlim=c(-40, 40))
>  for(i in 1:1000){
+    points(density(jags.lm.out.3[[1]][i,5:24]), type="l", col=rgb(0,0,0, .05), lwd=.5)
+  }
>  axis(1, lty=0)
>  # qq-Plot
>  plot(tq, res.mean, xlab="Theoretical Quantiles", ylab="Sample Quantiles", ylim=c(-30, 30),
col=rgb(0,0,0, .05), cex=.5, axes=F)
>  qqline(res.mean, col="black")
>  axis(1, col="white")
>  axis(2, col="white")
>  for(i in 1:1000){
+    points(tq, jags.lm.out.3[[1]][i,5:24], pch=19, cex=.5, col=rgb(0,0,0, .05))
+  }
>  dev.off()
quartz
     2
>
> # =============
> # = figure 7 =
> # =============
>
> # compute bayesian r squared
>
>   theta = do.call(rbind, jags.lm.out.3)
>   beta = theta[,grep("beta", colnames(theta))]
>   sigma = theta[,"sigma"]
>
>   m = lm(y ~ 1 + left + size + concen)
>   X = model.matrix(m)
>   Y.hat = X%*%t(beta)
>   exp.var = apply(Y.hat, 2, var)
>   res.var = sigma^2
>  bayes.rsq = exp.var / (exp.var + res.var)
>
> # plot bayesian r squared
>
>  cairo_pdf("figure-7.pdf", width=4, height=3, pointsize=10)
>  par(mfrow=c(1,1), family="Gill Sans", mgp=c(1.5, .5, 0), mar=c(3, 0, 0, 0))
>  plot(density(bayes.rsq), axes=F, main="", xlab=expression(R^2), ylab="", xlim=c(.3, .9))
>  polygon(density(bayes.rsq), col="grey")
>  axis(1, at=seq(.3, .9, .1), lty=0)
>  abline(v=median(bayes.rsq), lwd=2)
>  dev.off()
quartz
     2
>
> # numerical example in model level / explained variance / last paragraph
>
>  round(c(mean(bayes.rsq), median(bayes.rsq)), 2)
[1] 0.69 0.71
>  round(quantile(bayes.rsq, c(.025, .975)), 2)
 2.5% 97.5%
 0.49  0.82
>
> # ====================
> # = end source code =
> # ====================
>
```